Gesellschaft für Informatik (GI)

publishes this series in order to make available to a broad public recent findings in informatics (i.e. computer science and information systems), to document conferences that are organized in cooperation with GI and to publish the annual GI Award dissertation.

Broken down into the fields of
• Seminar
• Proceedings
• Dissertations
• Thematics
current topics are dealt with from the fields of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure the high level of the contributions.

The volumes are published in German or English.

Information: http://www.gi-ev.de/service/publikationen/lni/

ISSN 1617-5468
ISBN 978-3-88579-233-8

This volume contains papers from the Software Management 2008 Conference on Industrialization of Software Management held in Stuttgart November 12 to 14, 2008.The topics covered in the papers range from principles of software industrialization, to papers that discuss organizational and technological aspects of the industrialization of software management.

Georg Herzwurm, Martin Mikusz (Hrsg.): Industrialisierung des Software-Managements

139

GI-Edition

Lecture Notes in Informatics

Georg Herzwurm, Martin Mikusz (Hrsg.)

Industrialisierung des Software-Managements

Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschaftsinformatik (WI-MAW), Stuttgart, 2008

Proceedings

Georg Herzwurm, Martin Mikusz (Hrsg.)

Industrialisierung des Software-Managements

Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschaftsinformatik (WI-MAW)

12. - 14. November 2008
in Stuttgart

Gesellschaft für Informatik e.V. (GI)

Lecture Notes in Informatics (LNI) - Proceedings
Series of the Gesellschaft für Informatik (GI)

Volume P-139

ISBN 978-3-88579-233-8
ISSN 1617-5468

Volume Editors
Prof. Dr. Georg Herzwurm
Lehrstuhl für ABWL und Wirtschaftsinformatik II (Unternehmenssoftware)
Universität Stuttgart
Breitscheidstr. 2c, 70174 Stuttgart
E-Mail: herzwurm@wi.uni-stuttgart.de
Dipl.-Kfm. techn. Martin Mikusz
Lehrstuhl für ABWL und Wirtschaftsinformatik II (Unternehmenssoftware)
Universität Stuttgart
Breitscheidstr. 2c, 70174 Stuttgart
E-Mail: mikusz@wi.uni-stuttgart.de

Series Editorial Board
Heinrich C. Mayr, Universität Klagenfurt, Austria (Chairman, mayr@ifit.uni-klu.ac.at)
Hinrich Bonin, Leuphana-Universität Lüneburg, Germany
Dieter Fellner, Technische Universität Darmstadt, Germany
Ulrich Flegel, SAP Research, Germany
Ulrich Frank, Universität Duisburg-Essen, Germany
Johann-Christoph Freytag, Humboldt-Universität Berlin, Germany
Ulrich Furbach, Universität Koblenz, Germany
Michael Koch, TU München, Germany
Axel Lehmann, Universität der Bundeswehr München, Germany
Peter Liggesmeyer, TU Kaiserslautern und Fraunhofer IESE, Germany
Ernst W. Mayr, Technische Universität München, Germany
Heinrich Müller, Universität Dortmund, Germany
Sigrid Schubert, Universität Siegen, Germany
Martin Warnke, Leuphana-Universität Lüneburg, Germany

Dissertations
Dorothea Wagner, Universität Karlsruhe, Germany
Seminars
Reinhard Wilhelm, Universität des Saarlandes, Germany
Thematics
Andreas Oberweis, Universität Karlsruhe (TH)

© Gesellschaft für Informatik, Bonn 2008
printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Seit geraumer Zeit trägt Software nicht nur zur Industrialisierung anderer Wirtschaftszweige bei, sondern unterliegt selbst v. a. bei der Entwicklung und Wartung betrieblicher Anwendungssysteme sowie deren Management zunehmend einem Industrialisierungsprozess. Dies schafft neue Aufgaben, Herausforderungen, Verbesserungspotentiale und Anforderungen an das Software-Management in Praxis und Wissenschaft. Ihnen widmet sich die diesjährige, 7. Fachtagung Software-Management 2008 mit ihrem Schwerpunkt Industrialisierung des Software-Managements.

Im ersten Teil des Tagungsbandes finden sich die eingeladenen Vorträge von Peter Liggesmeyer, Leiter des Fraunhofer Instituts für Experimentelles Software Engineering (IESE), zur Software-Qualitätssicherung sowie Dirk Wittkopp, Director of Portal and Workflow Development, IBM Deutschland Research & Development, zu IT-Macro Trends.

Das wissenschaftliche Programm der Tagung umfasst 12 ausgewählte Beiträge aus Praxis und Wissenschaft, die einen Review-Prozess durchlaufen haben. Die Beiträge beleuchten Grundsatzfragen sowie technische, produktbezogene, ökonomische und organisatorische Aspekte der Industrialisierung der Softwareentwicklung aus Sicht des Software-Managements. Ihnen widmet sich der zweite Teil des Tagungsbandes.

Beginnend mit den Grundsatzfragen und -erfahrungen befassen sich Sven Markus Walter, Tilo Böhmann und Helmut Krcmar mit den Grundlagen der Industrialisierung der IT und von IT-Services sowie den damit einhergehenden Herausforderungen und Auswirkungen auf das Software-Management. Bruno Schienmann und Ulrich Schmedt stellen in ihrem Beitrag Maßnahmen und Erfahrungen der Postbank Systems zur Professionalisierung der Softwareentwicklung vor. Der Beitrag von Martin Mikusz und Georg Herzwurm beleuchtet am Beispiel von Six Sigma die Adaption und Umsetzung eines aus dem industriellen Kontext stammenden Prozessverbesserungsansatzes unter den Rahmenbedingungen der Softwareentwicklung und zeigt deren Grenzen und Nutzen auf. Die Auswirkungen der Software-Industrialisierung auf das Testmanagement und die sich daraus ergebenden Herausforderungen und notwendigen Maßnahmen stellt Andreas Birk in seinem Beitrag dar.

Im zweiten Themenschwerpunkt werden die ökonomischen und organisatorischen Aspekte der Industrialisierung der Softwareentwicklung beleuchtet. Hierzu stellt zunächst Wolfram Pietsch den IT-Produktkompass vor, welcher die verschiedenen Perspektiven und Entwicklungsstufen von IT-Produkten integriert und somit deren zielgerichtete und fokussierte Positionierung unterstützt. Oral Avci liefert auf Grundlage empirischer Befunde eine Antwort auf die Frage, warum Anforderungsfehler in Softwareentwicklungsprojekten entstehen und wie mit der Kenntnis ihrer Ursachen Anforderungsfehler vermieden oder durch gezielte QS-Maßnahmen früher gefunden werden können.

Anhand praktischer Erfahrungen, die im IT-Labor der Berufsakademie Lörrach gesammelt wurden, hinterfragt Eckhart Hanser einige der wichtigsten Regeln des Extreme Programming und zeigt am Beispiel von Crystal Clear, wie der gelebte Prozess vom gewählten Prozessmodell abweicht. Franziska Rothermel und Carsten König stellen in ihrem Beitrag einen Kriterienkatalog und ein Styleguide vor, die es trotz der zunehmenden Komplexität der Automotive Elektrik/Elektronik-Entwicklung erlauben, nutzeradäquate Prozessbeschreibungen zu erstellen und bestehende Prozessdarstellungen bezüglich ihrer Nutzeradäquatheit zu bewerten. Harry Sneed schildert in seinem Beitrag, wie die Softwarewartung und -weiterentwicklung als Dienstleistung von externen Software-Firmen angeboten werden können und wie ein Service Level Agreement-gestützter Prozess zur Durchführung der vereinbarten Dienstleistungen abläuft. Ob sich die Industrialisierung der Software-Branche in einer zunehmenden unternehmensexternen Spezialisierung wiederfindet, untersuchen Christian Wolf, Knut Geiger, Alexander Benlian, Thomas Hess und Peter Buxmann.

Die Beiträge von Christoph Rathfelder u. a. sowie Oliver Höß u. a. beleuchten die technischen und produktbezogenen Aspekte der Industrialisierung der Softwareentwicklung aus Sicht des Software-Managements. Christoph Rathfelder, Henning Groenda und Ralf Reussner präsentieren einen Zertifizierungsansatz für Software- und Architekturkomponenten, der den Anforderungen einer industrialisierten Softwareentwicklung genügt. Oliver Höß, Jens Drawehn und Anette Weisbecker stellen ein Metamodell für die integrierte Darstellung von Konzepten aus dem Umfeld der Industrialisierung der Softwareentwicklung (SOA, BPM, SPL, CBSE) vor, um die Entwicklung einer Domänen-Ontologie für aktuelle Konzepte in Software-Management und -Technik voranzutreiben.

Im dritten Teil des Tagungsbandes finden sich die Beschreibungen der fünf halbtägigen Tutorien, die einige Aspekte der Schwerpunktthemen aufgreifen: Andreas Birk zum effektiven Testmanagement; Stefan Jesse zum Requirements Engineering; Oral Avci zur Steuerung von Softwareentwicklungsprojekten mit Fehlerkennzahlen; Hans-Bernd Kittlaus zum Software-Produktmanagement und Mareike Solbach zu Veränderungsprojekten.

Wir danken Herrn Thomas Deelmann, Deutsche Telekom, und dem gesamten Team des Lehrstuhls für Allgemeine Betriebswirtschaftslehre und Wirtschaftsinformatik II (Unternehmenssoftware), Univ. Stuttgart, für die Unterstützung bei der Organisation der Tagung. Insbesondere möchten wir unserer Studentischen Hilfskraft Hatice Gögüs für Ihre stets zuverlässige und sehr gute Mitarbeit bei der Organisation der Tagung und der Redaktion des Tagungsbandes danken.

Stuttgart, im November 2008

Georg Herzwurm Martin Mikusz

Programmkomitee

Vorsitz:

Prof. Dr. Georg Herzwurm, Univ. Stuttgart

Mitglieder:

Dr. Martin Bertram, Commerzbank
Dr. Andreas Birk, Software.Process.Management
Prof. Dr. Hans Brandt-Pook, FH Lippe/Höxter
Prof. Dr. Peter Buxmann, TU Darmstadt
Dr. Thomas Deelmann, Deutsche Telekom
Anton Dillinger, SAP
Prof. Dr. Stefan Eicker, Univ. Duisburg-Essen
Martin Engstler, Fraunhofer IAO
Dr. Gerd Große, GFFT e.V.
Prof. Dr. Thomas Hess, Univ. München
Prof. Dr. Wolfgang Hesse, Univ. Marburg
Gerrit Kerber, aragon INTERACTIVE
Dr. Ralf Kneuper, Dr. Kneuper Beratung
Dr. Christian Kop, Univ. Klagenfurt
Dr. Torsten Langner, Microsoft
Prof. Dr. Dr. h.c. Heinrich Mayr, Univ. Klagenfurt
Prof. Dr. Werner Mellis, Univ. zu Köln
Martin Mikusz, Univ. Stuttgart
Prof. Dr. Andreas Oberweis, Univ. Karlsruhe
Bernd Oestereich, oose.de
Prof. Dr. Roland Petrasch, TFH Berlin
Prof. Dr. Wolfram Pietsch, FH Aachen
Werner Simonsmeier, sd&m
Harry Sneed, AneCon
Prof. Dr. Thorsten Spitta, Univ. Bielefeld
Prof. Dr. Dirk Stelzer, TU Ilmenau
Dr. Dirk Voelz, SAP Research
PD Dr. Anette Weisbecker, Fraunhofer IAO
Dirk Wittkopp, IBM Deutschland

Organisatorische Leitung

Martin Mikusz, Univ. Stuttgart
Dr. Thomas Deelmann, Deutsche Telekom

Inhaltsverzeichnis

Teil I – Eingeladene Vorträge **11**

Peter Liggesmeyer
Software-Qualitätssicherung gestern und heute: Theorie und Erfahrung, Standards und Common Sense 13

Dirk Wittkopp
IT-Macro Trends 15

Teil II – Wissenschaftliches Programm **17**

Sven Markus Walter, Tilo Böhmann, Helmut Krcmar
Grundlagen der IT-Industrialisierung 19

Bruno Schienmann, Ulrich Schmedt
Professionalisierung der Softwareentwicklung - ein Praxisbericht 31

Martin Mikusz, Georg Herzwurm
Adaption, Umsetzung, Grenzen und Nutzen von Six Sigma in der Softwareentwicklung 45

Andreas Birk
Testmanagement unter dem Einfluss der Software-Industrialisierung 57

Wolfram Pietsch
Der IT-Produktkompass – Ein Instrument für die Analyse von Geschäftsmodellen und die strategiekonforme Positionierung von IT-Produkten 73

Oral Avcı
Warum entstehen in der Anforderungsanalyse Fehler? Eine Synthese empirischer Befunde der letzten 15 Jahre 89

Eckhart Hanser
Rollenveränderung in der Agilen Software-Entwicklung: Das „Projektmanagement-Labor“ der BA Lörrach 105

Franziska Rothermel, Carsten König
Nutzeradäquate Prozessbeschreibungen in der Automotive E/E Entwicklung 125

Harry M. Sneed
Industrialisierung der Softwarewartung & -weiterentwicklung 139

Christian M. Wolf, Knut Geiger, Alexander Benlian, Thomas Hess, Peter Buxmann
Spezialisierung als Ausprägungsform einer Industrialisierung der Software-Branche – Eine Analyse am Beispiel der ERP-Software von SAP 153

Christoph Rathfelder, Henning Groenda, Ralf Reussner
Software Industrialization and Architecture Certification 169

Oliver Höß, Jens Drawehn, Anette Weisbecker
Ein Metamodell zur integrierten Darstellung von aktuellen Konzepten zur Industrialisierung der Softwareentwicklung 181

Teil III – Tutorien **191**

Andreas Birk
Tutorium: Effektives Testmanagement - Vom notwendigen Übel zum Erfolgsfaktor für IT und Geschäftserfolg 193

Stefan Jesse
Tutorium: Erhebung von Produktanforderungen durch den Requirements Engineer 195

Oral Avci
Tutorium: Mit Fehlerkennzahlen Softwareentwicklungsprojekte erfolgreich steuern 205

Hans-Bernd Kittlaus
Tutorium: Software-Produkt-Management 211

Mareike Solbach
Tutorium: Wie Veränderungsprojekte wirklich funktionieren – eine praktische Anleitung 219

Teil I

Eingeladene Vorträge

Software-Qualitätssicherung gestern und heute:

Theorie und Erfahrung, Standards und Common Sense

Peter Liggesmeyer

Fraunhofer IESE
Fraunhofer-Platz 1
D-67663 Kaiserslautern
Peter.Liggesmeyer@iese.fraunhofer.de

Abstract: Wer heute Software-Qualitätssicherung verantwortet, diese konzipieren und steuern muss, ist „eingeklemmt“ zwischen einem Mangel an Theorie, der hohen Bedeutung der Prüfung und der unvermeidlichen Ressourcenknappheit der Praxis. Standards und empirisch abgesicherte Erkenntnisse zu Themen, die sich rein theoretisch nur schwierig erschließen lassen, können Hilfestellungen bieten. Der Vortrag beleuchtet die unterschiedlichen Kriterien, nach denen die Software-Qualitätssicherung heute beurteilt werden muss, und zeigt Wege zum Ziel.

IT-Macro Trends

Dirk Wittkopp

Director of Portal and Workflow Development
IBM Deutschland Research & Development GmbH
71032 Böblingen

Abstract: Geschäftserfolg erfordert neben der kontinuierlichen Weiterentwicklung des Produktes auch eine innovative und flexible IT Infrastruktur um die zunehmenden Herausforderungen des Marktes zu nutzen und frühzeitig auf globale Veränderungen zu reagieren. Der Vortrag beleuchtet neue IT-Trends um vorausschauend die nötigen Schritte für eine nachhaltige Veränderung der IT Infrastruktur zu erkennen.

Teil II

Wissenschaftliches Programm

Grundlagen der IT-Industrialisierung

Sven Markus Walter, Tilo Böhmann, Helmut Krcmar

Sven Markus Walter
Deloitte Consulting GmbH - CIO Advisory Services
Löffelstr. 42
70597 Stuttgart
SvWalter@deloitte.de

Tilo Böhmann
ISS - International Business School of Service Management
Hans-Henny-Jahnn-Weg 9
22085 Hamburg
boehmann@iss-hh.de

Helmut Krcmar
Technische Universität München
Lehrstuhl für Wirtschaftsinformatik (I17)
Boltzmannstr. 3
85748 Garching b. München
krcmar@in.tum.de

Abstract: Bereits seit einigen Jahren hat die Informationstechnik (IT) nicht nur zur Industrialisierung anderer Wirtschaftszweige beigetragen, sondern unterliegt in zunehmendem Maße selbst einem Industrialisierungsprozess. Die Analyse der strukturellen Grundlagen der Industrialisierung zeigt, dass es im Zuge einer fortschreitenden IT-Industrialisierung zu Veränderungen bei den Faktoren Preis, Produkt, Produktion, Personal und Partner kommt. Neben den Veränderungen bei den Produkten der IT und deren Preisen spielt vor allem das Management der IT-Leistungserstellung, d. h. die Produktion von IT, eine wichtige Rolle.

1 IT-Industrialisierung und deren Auswirkungen auf das Software-Management

Die Rolle der Informationstechnik (IT) wurde lange Zeit als wettbewerbsdifferenzierender Produktionsfaktor diskutiert. IT wurde als „Enabler" neuer Geschäftsmodelle wahrgenommen – der Einsatz von IT war oft eine wichtige strategische Entscheidung. Seit einiger Zeit wird diskutiert, ob die IT nun zur „Commodity" wird, die – ähnlich wie elektrische Energie – eine geschäftskritische Ressource ist, deren Einsatz aber keine Wettbewerbsvorteile mit sich bringt [Ca03, S. 44-48].

Nun stellt sich die Frage, ob IT bereits zu einem Massenprodukt geworden ist, das für alle Nachfrager über einen anonymen Markt in gleichem Ausmaß verfügbar ist und damit keinen Vorteil gegenüber den Konkurrenten verschaffen kann.

Vorschläge für standardisierte IT-Prozesse, wie z. B. die aus der IT Infrastructure Library (ITIL), werden bereits seit mehreren Jahrzehnten entwickelt und angewendet – seit kurzem wird nun unter dem Stichwort „IT-Industrialisierung" intensiv darüber diskutiert, wo Standardisierung und Automatisierung in der IT notwendig sind und in welchen Bereichen eine weitergehende Professionalisierung der Arbeitsabläufe notwendig ist [WBK07, S. 6]. Diese aktuelle Debatte beschäftigt sich mit der Industrialisierung von Hardware, Software und IT-Services. Der vorliegende Artikel zeigt diese wesentlichen Bereiche der IT-Industrialisierung auf und dient dazu die Rahmenbedingungen aufzuzeigen, unter denen die Industrialisierung des Software-Managements voranschreitet.

2 Strukturmerkmale der Industrialisierung

Im Zuge der sog. „industriellen Revolution" setzte sich im neunzehnten Jahrhundert erstmals die industrielle Produktionsform durch [En82]. Davor waren Kleingewerbe und handwerkliche Produktion dominierend. Die Ursachen für die umfangreichen Veränderungen im Produktionsprozess waren einerseits eine zunehmende Automatisierung der Produktion, d. h. die Ersetzung von menschlicher Arbeitskraft durch Maschinen. Andererseits war die Innovation in Produktionstechniken, der Produktionslogistik und die zunehmende Arbeitsteilung innerhalb des produzierenden Gewerbes Ursache des Strukturwandels [La69, S. 15-16]. Dadurch wurde erstmals die Massenproduktion von Gütern möglich [WBK07, S. 6f.].

Bei der Massenproduktion kann mit Hilfe zusätzlicher Produktionsausstattung und mit speziell ausgebildeten Arbeitskräften höherer Output zu stark gesunkenen Stückkosten hergestellt werden. Die Ausnutzung dieser positiven Skaleneffekte („economies of scale") führt zur Spezialisierung auf einzelne Produkte. Um die Investitionen in die speziellen Werkzeuge und Produktionsmethoden zu refinanzieren muss jedoch die Absatzmenge gesteigert werden [PS84]. Das führt zu einer Zentralisierung der Produktion und zu einer Konzentration der Anbieter [PS84].

Im Zeitablauf ergab sich aus der Einführung der Massenproduktion eine generell zunehmende Spezialisierung der einzelnen Hersteller. Höhere Produktionsvolumina und sinkende Stückkosten führten zur Erweiterung von Märkten [WBK07, S. 7]. Die Produzenten engagierten sich zunehmend in der Standardisierung ihrer Austauschbeziehungen und ermöglichten damit die Entstehung von Zwischenprodukten. Diese Zwischenprodukte förderten eine weitergehende Arbeitsteilung zwischen den Produzenten [La69, S. 295-296].

Die zunehmende Arbeitsteilung ist durch die Auslagerung wenig wettbewerbsfähiger, ineffizienter Teile der Wertschöpfung an spezialisierte Anbieter, die ein höheres Potential für positive Skaleneffekte aufweisen, gekennzeichnet. Damit wird die Konzentration auf Kernkompetenzen zur dominanten Strategie. Die Standardisierung von Produkten ermöglicht den Herstellern die Standardisierung der Produktionsprozesse, deren Automatisierung und die Verteilung auf verschiedene Wertschöpfungspartner in der ganzen Welt.

Industrialisierung bedeutet also die, auf technischem und organisatorischem Fortschritt basierende, Automatisierung der Produktion, die zu signifikant sinkenden Kosten für die zentralisierte, spezialisierte und global verteilte Produktion standardisierter Produkte führt [WBK07, S. 7f.]. Für die Informationstechnologie bedeutet Industrialisierung folglich einen Wandel, der dem ähnelt, der sich zu Beginn des neunzehnten Jahrhunderts im Kleingewerbe und in der handwerklichen Produktion von Sachgütern vollzog. In der Folge dieses Veränderungsprozesses wird es auch bei der Informationstechnologie zur Automatisierung der Produktion, sinkenden Stückkosten und zur Spezialisierung von Anbietern kommen, die ihre standardisierten Leistungen global verteilt erstellen.

3 Industrialisierung der IT

3.1 Bereiche der IT-Industrialisierung

Die Industrialisierung der IT vollzieht sich in allen Bereichen, die für die Bereitstellung von Informationstechnologie eine Rolle spielen (Abbildung 1). Zum einen bei der industriellen Produktion von IT-Hardware und bei Standardsoftware-Produkten, wie standardisierten Büroanwendungen oder integrierten ERP-Lösungen. Zum anderen bei IT-Services. Der Begriff IT-Services beschreibt die Planung, den Erwerb und den Betrieb von IT-Anwendungen und -Infrastrukturen und umfasst z. B. individuell angepasste Softwareanwendungen.

Abbildung 1: Bereiche der IT-Industrialisierung

In der Diskussion um die Industrialisierung der IT argumentieren zahlreiche Wissenschaftler und Praktiker, dass sich die IT-Branche drastisch verändert und immer mehr individuelle IT-Lösungen durch Standardprodukte – sog. „Commodities“ – ersetzt würden [z. B. Ca03, S. 44-48; Ta05, S. 293f.]. Dies lässt sich seit längerem im Bereich IT-Hardware (Kapitel 3.2) und bei der Entwicklung von Standardsoftware-Produkten (Kapitel 3.3) beobachten. In neuerer Zeit zeichnen sich vor allem auch im Bereich der IT-Services starke Veränderungen ab, auf die daher im Kapitel 4 vertieft eingegangen wird.

3.2 Industrielle Produktion von IT-Hardware

Bei der IT-Hardware konnte bereits in den letzten Jahrzehnten die Bildung von Standard-Produkten beobachtet werden. Die IT-Hardware ist heute weitgehend standardisiert und kann als „Commodity" betrachtet werden. Individuell für einen bestimmten Einsatzzweck entwickelte IT-Hardware („Custom Built") wird heutzutage nur noch in wenigen Fällen, wie z. B. bei Großrechneranlangen, eingesetzt. Selbst bei früher für den Einzelfall konstruierten Geräten, wie beispielsweise eingebetteten Steuergeräten in der Automobilindustrie, setzen sich zunehmend Standards durch, sodass auch in diesem Bereich von weitestgehend standardisierten Hardwarelösungen ausgegangen werden kann [WBK07, S. 8].

Die Standardisierung der Produktionsprozesse ist bei IT-Hardware – ähnlich wie bei anderen Industrieprodukten – bereits durch frühere Industrialisierungswellen fortgeschritten [WBK07, S. 8]. Es kann also in der IT von Standard-Hardware-Produkten ausgegangen werden, die mit Hilfe standardisierter Prozesse und unter Mitwirkung vieler Zulieferer erstellt werden. Für die nachfolgenden Betrachtungen geht der Faktor Hardware in der zu betrachtenden IT-Infrastruktur auf.

3.3 Standardisierte Software-Produkte

Es gibt deutliche Anzeichen dafür, dass die Industrialisierung auch im Bereich der Software voranschreitet. Das zeigt die wachsende Anzahl von Publikationen zum Software-Produktmanagement und zum Management von Software-Produktlinien [CN02; KRS04; MA07]. Immerhin machen die Ausgaben für Standardsoftware bereits einen Anteil von rund 25 % an den Umsätzen des deutschen IT-Markts aus [eigene Berechnungen auf Basis von: Bi05]. Insbesondere das Beispiel gängiger Büroanwendungen, die vom Benutzer heutzutage nur noch minimal angepasst werden („Off-the-Shelf-Software"), legt den Schluss nahe, dass es bereits Software-Produkte gibt, die als „Commodity" betrachtet werden können.

Bei der Erstellung von Software kommen schon seit langem Tools für das Computer-Aided Software Engineering (CASE) zum Einsatz. Neue Trends, wie z. B. die Model-Driven-Architecture (MDA) erweitern die etablierten Vorgehensweisen und führen zu einer höheren Automatisierung und Produktivität im Erstellungsprozess [KTV07, S. 68]. Gleichzeitig wird diskutiert, in wie weit schlanke Vorgehensweisen wie Extreme Programming (XP) eine „Lean Production" in der IT ermöglichen können [PT07]. Mit der Standardisierung und Automatisierung der Prozesse bei der Erstellung von Standardsoftware-Produkten kommt es zu einer Dekomposition der Wertschöpfungskette. Immer mehr Standardsoftware-Produkte für spezielle Einsatzszenarien werden als Zwischen- oder Endprodukte am Markt zur Verfügung stehen. Es kann davon ausgegangen werden, dass die „Commoditisierung der IT" auch bei den Software-Produkten zunimmt. Auf Besonderheiten, die bei der Entwicklung von individuell angepassten Anwendungssystemen zu berücksichtigen sind, wird im Kontext von IT-Services eingegangen.

4 Industrialisierung von IT-Services

4.1 Entwicklungstrends

Der Industrialisierung von IT-Services wurde bislang weniger Beachtung geschenkt. IT-Services zielen auf die effektive und effiziente Befriedigung des Informationsbedarfs durch die Entwicklung und Anpassung sowie den Betrieb von IT-Anwendungen und -Infrastrukturen.

Bei IT-Dienstleistungen, die eng mit einzelnen Hard- oder Softwareprodukten verbunden sind, ist die Industrialisierung oft schon fortgeschritten. Beispiele hierfür sind standardisierte Wartungspakete für Hardware-Produkte oder die Wartung von Standardsoftware. Im Gegensatz zu solchen Infrastruktur-Dienstleistungen stehen IT-Dienstleistungen, die die Planung, Entwicklung oder den Betrieb von komplexen Geschäftsanwendungen zum Gegenstand haben. Hierbei handelt es sich oft um maßgeschneiderte Individuallösungen. Daher muss bei der Betrachtung von IT-Services zwischen der Bereitstellung von Infrastruktur und der Entwicklung und Wartung von komplexen Anwendungssystemen unterschieden werden.

Durch die Industrialisierung verstärken sich bei den IT-Services drei Entwicklungstrends: die Dienstleistungsorientierung, die Prozessorientierung und die Architekturorientierung (Abbildung 2). Dabei umfasst die Dienstleistungsorientierung die systematische Kommunikation mit den Kunden und die Ableitung und Spezifikation von Leistungsangeboten. Durch Prozessorientierung wird gewährleistet, dass die so erkannten Erwartungen der Kunden und die ihnen zugesicherten Dienstleistungen zuverlässig und effizient erbracht werden. Sie gewinnt besonders durch prozessorientierte Referenzmodelle wie z. B. die ITIL an Bedeutung. Die Architekturorientierung hingegen zeigt auf, wie den wachsenden Forderungen nach bedarfsorientierten, anpassbaren Leistungen durch die Entwicklung modularer Servicearchitekturen für IT-Dienstleistungen begegnet werden kann [BK04, S. 7].

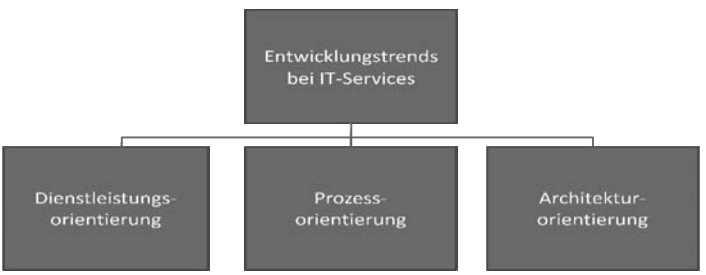


Abbildung 2: Entwicklungstrends bei IT Services

Derzeit gibt es nur einige wenige Studien, die Nachweise für eine Industrialisierung im Bereich der IT-Services liefern, dennoch sprechen eine Reihe von Anzeichen dafür, dass sich die Trends der Dienstleistungs-, Prozess- und Architekturorientierung verstärken.

4.2 Dienstleistungsorientierung

Die Dienstleistungsorientierung umfasst die Identifikation von Kunden, die auf diese Kunden ausgerichtete Bereitstellung von Informationssystemen als Dienstleistung sowie die Planung und Kontrolle der Dienstleistungsqualität. Dadurch wird die Ableitung und Spezifikation von Leistungsangeboten (Service Level Agreements) möglich, auf deren Basis die Leistungserstellung besser auf die Anforderungen des Kunden ausgerichtet wird und die Leistungen transparenter gestaltet werden können. Die Orientierung am Kunden und die Spezifikation der zu erbringenden Leistungen sind die beiden Säulen der Dienstleistungsorientierung bei IT-Services [BK04, S. 8-10].

Im Zuge der IT-Industrialisierung gewinnt die Definition von standardisierbaren IT-Service-Produkten an Bedeutung. Aktuell werden vielfach noch individuelle, speziell auf die Kundenwünsche abgestimmte Software-Anwendungen, eingesetzt [WBK07, S. 10]. Es gibt jedoch Überlegungen, die Softwareprodukte komponenten- bzw. serviceorientiert zu gestalten. Die zunehmende Durchsetzung von Standards für Komponenten ermöglicht die Bildung von Zwischenprodukten („Components“). Damit wird eine größere Arbeitsteilung zwischen Softwareherstellern und eine Dekomposition der Wertschöpfungskette möglich [Ta05, S. 295]. Diese Aufspaltung der Wertschöpfungskette wird unter dem Stichwort „Composite Applications“ diskutiert.

Am Ende der IT-Industrialisierung werden für Endkunden und Unternehmensanwender zahlreichere Standardsoftware-Lösungen und nach industriellem Vorbild aus einzelnen Komponenten zusammengesetzte Individualsoftware zur Verfügung stehen („Componentware“). Die Diskussion um die Bereitstellung von Anwendungen im Sinne eines Application Service Providing (ASP) wird im Moment unter dem Begriff „Software-as-a-Service“ (SaaS) erneut geführt. Dabei werden standardisierte Produkte in kombinierter Form angeboten [Ho07]. Ein Beispiel dafür ist eine Lösung für das Customer Relationship Management, die ohne Installation und aufwändige Tests beim Kunden über eine Web-Oberfläche genutzt werden kann.

Ein Indikator für eine zunehmende „Produktisierung“ von IT-Services ist die Durchsetzung von Produktmanagement bei den IT-Unternehmen [BTK06, S. 10]. Nach einer Studie der TU München kann davon ausgegangen werden, dass Produktmanagement für IT-Dienstleistungen bereits flächendeckend eingesetzt wird. 80% der in dieser Studie befragten IT-Serviceunternehmen gaben an, dass sie bereits über ein Produktmanagement verfügen. Am umfassendsten ist Produktmanagement aktuell für Support- und Maintenance-Leistungen etabliert [BTK06, S. 6]. Durch die Bildung von IT-Service-Produkten können Einsparungspotentiale und Skaleneffekte in der zugrunde liegenden Delivery ausgenutzt und IT-Dienstleistungen als Massenprodukte am Markt angeboten werden.

4.3 Prozessorientierung in der IT

Bei der Prozessorientierung wird die IT-Leistungserstellung aus dem Blickwinkel von funktionsübergreifenden Zusammenhängen und Abhängigkeiten betrachtet [BK04, S. 12f.]. Das Ziel dabei ist es, die Leistungserstellung der IT durch standardisierte Prozesse zu verbessern. Im Unternehmen kann eine Standardisierung der Prozesse die Kommunikation über die Betriebsabläufe erleichtern, reibungslose Übergaben über Prozessgrenzen hinweg ermöglichen und Möglichkeiten für Performancevergleiche schaffen. Zwischen Unternehmen gilt dasselbe: die Kommunikation wird vereinfacht, Übergaben erleichtert und Performancevergleiche möglich.

Ein Weg zur Gestaltung der Leistungserstellung in der IT stellt die Orientierung an Referenzmodellen für IT-Prozesse dar. In der Praxis hat sich eine Reihe solcher Modelle etabliert, die einen gewissen normativen Charakter aufweisen und zum Teil durch ihre praktische Anwendung empirisch bestätigt wurden. Für die Ausgestaltung ihrer IT-Prozesse können Unternehmen auf frei verfügbare oder auf herstellerspezifische Modelle zurückgreifen. Referenzmodelle stehen dabei nicht nur für den Betrieb von IT-Infrastrukturen zur Verfügung (z. B. ITIL, CobiT, eTOM, etc.) – auch für Entwicklung von Anwendungen gibt es Modelle, die eine ganz ähnliche, prozessorientierte Struktur aufweisen (RUP, V-Modell XT, CMMI, usw.) [WK06, S. 3].

Bei den Referenzmodellen für IT-Prozesse hat sich bislang vor allem die ITIL in der praktischen Anwendung etabliert. Durch den Einsatz von ITIL soll eine Kundenorientierung erreicht und eine hohe Dienstleistungsqualität gewährleistet werden. ITIL versucht dazu, die Kommunikation zwischen den IT-Kunden und der IT durch Rollen und die Festlegung der Zusammenarbeit in Prozessen zu institutionalisieren. Dadurch sollen Erwartungen der Kunden besser verstanden und Dienstleistungen besser auf die Anforderungen der Geschäftsaktivitäten ausgerichtet werden. Ausgangspunkt dafür ist der zentrale Kontaktpunkt für die Kunden, der Service Desk. An diesen knüpfen Prozesse des Störungs- und Problemmanagements und der kundenorientierten Disposition von Veränderungen der IT-Dienstleistungen (Change Management) an. Diese werden ergänzt durch die Planung, Überwachung und Steuerung von zugesicherten Leistungen der IT, der Verrechnung dieser Leistungen sowie ihrer Verfügbarkeit, Performanz und Störfallsicherheit [BK04, S. 13].

Obwohl die Umsetzung der Referenzmodelle in der Praxis bislang nur teilweise und zögerlich erfolgt ist [WK06, S. 3], ist dennoch eine fortschreitende Automatisierung der IT-Prozesse festzustellen [Ca06, S. 30]. Ein Beispiel dafür ist der kombinierte Einsatz von Monitoring- und Ticketing-Systemen im Incident Management, mit denen die Verarbeitung von Störungen automatisiert werden kann.

Aktuell gibt es einen Trend zur Zertifizierung von IT-Prozessen nach allgemein gültigen Standards wie ISO 20000 oder dem Defacto-Standard CMMI. Es wird erwartet, dass sich die Struktur der IT-Branche mit zunehmender Reife langfristig der Struktur in der Automobilbranche annähert [Ca06, S. 16]. Denn standardisierte Prozesse vereinfachen – neben der Automatisierung – auch das Outsourcing von Leistungen, weil durch die Standardisierung von Übergabepunkten einzelne Teilprozesse einfacher ausgelagert werden können. Ein Beispiel dafür ist eine automatisierte „Test-Factory", bei der ein Anbieter spezialisierte Dienstleistungsbausteine für die Planung und Durchführung von Softwaretests konzipiert und am Markt anbietet.

4.4 Architekturorientierung in der IT

Die Architekturorientierung eröffnet Optionen zur Weiterentwicklung, zur Anpassung und zur globalen Leistungstiefenbestimmung durch die gezielte Gestaltung von Servicearchitekturen [BK04, S. 7f.]. Damit kann den wachsenden Forderungen nach bedarfsorientierten, anpassbaren Leistungen begegnet werden. Die Anbieter von IT-Dienstleistungen stehen vor der Herausforderung ein variantenreiches Leistungsangebot anzubieten und diese Leistungen gleichzeitig mit möglichst hoher Effizienz und unter Ausnutzung entsprechender Skaleneffekte zu erbringen. Durch die Nutzung modularer Servicearchitekturen und Plattformen wird – in Analogie zu einem Baukasten – die Möglichkeit zur Wiederverwendung bestehender Komponenten für verschiedenen Kunden und zum Fremdbezug einzelner Teilleistungen geschaffen. Außerdem wird eine vereinfachte Implementierung von Veränderungen in späteren Lebenszyklusphasen unterstützt [BK04, S. 19].

Momentan wird für die Gestaltung von Anwendungssystemen die Idee Modell- und Service-orientierter Architekturen (SOA) diskutiert [PW07]. Die Model Driven Architecture (MDA) ist ein Ansatz zur formalisierten Entwicklung von IT-Lösungen, der auf einer klaren Trennung von Funktionalität und Technik beruht. Durch die Formalisierung kann eine Automatisierung bei der Erstellung von Quellcode unterstützt werden. SOA ist ein Konzept zum Entwurf von Systemlandschaften. Die drei grundlegenden Ideen, auf denen das Konzept basiert, sind die Komponentenorientierung, deren lose Kopplung sowie eine Workflowsteuerung. Auf dieser Basis können Services ausgelagert und dadurch ggf. Skaleneffekte genutzt werden. Unternehmen oder Anbieter von Standardsoftware-Lösungen haben die Möglichkeit, nur noch sehr individuelle Leistungen selbst zu erstellen. Das setzt jedoch voraus, dass bestehende Plattformen so angepasst werden, dass Leistungen von Partnern reibungslos integriert werden können [CS07, S. 16].

5 Herausforderungen für IT-Dienstleister durch die IT-Industrialisierung

Erste empirische Evidenz für eine Industrialisierung der IT liegt vor und in vielen Bereichen der IT ist die Industrialisierung bereits voran geschritten. Um eine strukturierte Analyse der Auswirkungen der IT-Industrialisierung zu ermöglichen, sind die wesentlichen Faktoren zu identifizieren, die davon betroffen sind.

Im Zuge eines fortschreitenden Industrialisierungsprozesses kommt es zu Veränderungen bei den Faktoren: Preis, Produkt, Produktion, Personal und Partner [WBK07, S. 7]. Diese Faktoren verändern sich im Zuge eines fortschreitenden Industrialisierungsprozesses komplementär zueinander. Neben Veränderungen bei IT-Produkten und Preisen spielt v. a. die Gestaltung der Produktion der IT eine wichtige Rolle. Zusätzlich verringert sich die Leistungstiefe – die Einbindung externer Partner wird zunehmend wichtig. Schließlich kommt es im Zuge der Industrialisierung zu maßgeblichen Veränderungen beim Faktor Personal [WBK07, S. 7]. Im Folgenden zeigen wir exemplarisch neue Herausforderungen in diesen fünf Bereichen, die sich aus einer Industrialisierung der IT ergeben.

Die Industrialisierung der IT führt zu einer zunehmenden Produktbildung: Es existieren bereits standardisierte Hard- und Softwareprodukte und es gibt erste Anzeichen für eine Evolution in Richtung von standardisierten IT-Service-Produkten [BK01]. Durch die „Produktisierung“ der IT-Leistungen können IT-Dienstleister Einsparungspotentiale und Skaleneffekte in der zugrunde liegenden Delivery nutzen und IT-Leistungen als Massenprodukte am Markt anbieten. Diese standardisierten Produkte können als Vor- bzw. Zwischenprodukte in andere Produkte einfließen. Damit kann die Leistungserstellung aufgespalten und auf verschiedene Anbieter verteilt werden [WBK07, S. 12].

Weiterhin hat die Industrialisierung der IT Auswirkungen auf die Preise, die IT-Dienstleister am Markt erzielen können [WBK07, S. 13]. Insbesondere große Anbieter können die Produktionsmengen erhöhen und ihre sinkenden Durchschnittskosten an die Kunden weitergeben. Gleichzeitig treibt die Forderung nach Kundenorientierung die Variantenvielfalt von IT-Leistungen in die Höhe. Doch die hohe Variantenvielfalt bedingt keine Erhöhung der Zahlungsbereitschaft bei den Kunden. Sie fordern hochgradig individualisierte Lösungen zum Preis eines Standardprodukts [BK04, S. 16]. Dadurch stehen die IT-Dienstleister seit einiger Zeit unter einem steigenden Preis- und Wettbewerbsdruck [BTK06, S. 1]. Die Gewinnmargen sinken und eine Preisspirale nach unten entsteht [Ta05, S. 293f.]. Es ist also mit einer Degression der Preise zu rechnen – Nicholas Carrs These einer Commoditisierung der IT wird damit in Teilen bestätigt.

In der Produktion spielt die Prozessstandardisierung eine wichtige Rolle. Auf der Basis standardisierter Abläufe können einzelne Teile der eigenen Leistungserstellung leichter herausgelöst und durch spezialisierte externe Partner erbracht werden. Projekte zur Einführung neuer IT-Prozesse scheitern jedoch oft an der organisatorischen Umsetzung und an Leadership-Fragestellungen, da veränderte IT-Prozesse v.a. Auswirkungen auf die tägliche Arbeit der an den Prozessen beteiligten Mitarbeiter haben [WBK07, S. 13]. Folglich ist bei der Durchsetzung solcher Systemveränderungen mit Widerstand gegen diese zu rechnen. D. h., die Referenzmodelle für IT-Prozesse können nicht einfach „von heute auf morgen“ im Unternehmen genutzt werden – die Reorganisation von IT-Prozessen ist ein langwieriger Prozess. Um möglichen Widerständen gegen die Reorganisation der IT-internen Abläufe zu begegnen und um die Prozesseinführung zu verkürzen, ist – neben der Einführung der zur Automatisierung der Prozesse notwendigen Tools – die gezielte Unterstützung des notwendigen organisatorischen Wandels unumgänglich [WBK07, S. 13].

Für Anbieter von IT-Dienstleistungen stellt sich zunehmend die Frage, ob sie ihre Leistungen durch Eigenerstellung oder durch Partner (Fremdbezug) bereitstellen wollen. In diesem Zusammenhang wird unter dem Stichwort „Outsourcing“ diskutiert, welche Teilleistungen oder sogar ganze Prozesse ausgelagert und durch externe Partner bereitgestellt werden. Schätzungen zur Folge wird der weltweite Markt für IT-Outsourcing bis zum Jahr 2010 auf über 292 Mrd. US$ anwachsen. Dabei wächst der Markt kontinuierlich [Sc06]. Die Auslagerung von IT-Leistungen ist im Bereich der Softwareentwicklung am weitesten fortgeschritten. Weniger als 30 % der Softwareentwicklung erfolgt „inhouse“ – der Rest wird durch externe Dienstleister erstellt. Es wird angenommen, dass die Leistungstiefe hier in den kommenden Jahren weiter abnehmen wird [CS07, S. 16]. Im Bereich Betrieb, Wartung und Pflege der Infrastruktur sowie im Bereich Anwendungsmanagement lag der interne Anteil an der Wertschöpfung in 2005 noch bei ca. 60 % [Ca06, S. 16f.]. Schätzungen zufolge wird 2010 nur noch ein Fünftel der Individualsoftware intern erstellt und nur noch rund ein Drittel der Infrastruktur vom Unternehmen selbst betreut. Nur noch etwas mehr als 40 % der Wartung von Anwendungen wird in den Unternehmen verbleiben [CS07, S. 16]. Im Hinblick auf die Auslagerungen von Leistungen stehen die IT-Dienstleister vor der Frage, wo – d. h. in welchem Land – die einzelnen Leistungen erbracht werden sollen. Neben der Erstellung im eigenen Land stehen „Offshoring“ und „Nearshoring“ als zusätzliche Optionen zur Auswahl, die in ihrer Bedeutung zunehmen [MS07, S. 83]. Obwohl die internationale Arbeitsteilung an sich kein neues Phänomen ist, stellt sie IT-Dienstleister dennoch vor neue Herausforderungen.

Im Zuge der Industrialisierung der IT kommt es also nicht nur zu Veränderungen bei Produkten, Preisen, der Produktion und der Einbindung externer Partner. Als Folge der Standardisierung von Produkten und Prozessen, dem Trend zur Spezialisierung und aus einer zunehmenden Arbeitsteilung ergeben sich auch veränderte Anforderungen an die Kenntnisse und Fähigkeiten des Personals [WBK07, S. 14]. IT-Dienstleister müssen mit der Beschaffung und Ausbildung entsprechender Kompetenzen auf die neuen Herausforderungen reagieren, die sich aus der Aufspaltung, Auslagerung und aus der internationalen Verteilung ihrer Leistungserstellung ergeben. Für eine spezialisierte Produktion werden z. B. sehr spezifisches Wissen und spezialisierte Fähigkeiten benötigt [Ja05, S. 285]. Gleichzeitig werden sich IT-Facharbeiter herausbilden, die in der Produktion standardisierter IT-Produkte beschäftigt sind. Für die Planung und Definition industrialisierter IT-Dienstleistungen und die zunehmend wichtige Koordination verschiedener Partner im Leistungserstellungsprozess hingegen wird eine breitere Qualifikation der Mitarbeiter benötigt [WBK07, S. 14]. Folglich werden sich bei den Anbietern Verschiebungen hinsichtlich des Qualifikationsbedarfs sowie regionale Verschiebungen der Arbeitsnachfrage ergeben.

Mit der Einführung standardisierter Produkte und Prozesse entstehen neue Anforderungen an die Unternehmen – sowohl bezüglich der Zusammenarbeit mit externen Partnern als auch hinsichtlich der Qualitätssicherung ausgelagerter Leistungen, da diese beiden Themen im Hinblick auf die geringere Leistungstiefe und die zunehmende Arbeitsteilung wichtiger werden.

Einige Autoren gehen dabei von der Notwendigkeit einer Reorganisation der Aufbauorganisation der IT-Funktion aus, da in der IT – ähnlich wie bei der industriellen Produktion – die Steuerung der komplexen Wertschöpfungskette zunehmend wichtig wird. Eine Möglichkeit darauf zu reagieren ist die Abkehr von der bisher üblichen Organisation der IT in Funktionsbereiche („Plan, Build, Run"). [ZB04, S. 16-18] schlagen stattdessen vor, die Leistungserstellung nach dem industriellen Paradigma „Source, Make, Deliver" zu rekonstruieren.

Aus einer Industrialisierung der IT können sich aber auch neue Geschäftsmodelle ergeben. In der Vision sog. „Web service ecosystems" können Anbieter von Basis- bzw. Kern-Diensten ihr Angebot durch zusätzliche Leistungen ergänzen, die auf der selben Plattform bereitgestellt werden [BD06]. Die zusätzlichen Funktionen würden dabei von spezialisierten Intermediären bereitgestellt werden, die z. B. Zahlungsverkehr und Auslieferung übernehmen.

6 Fazit

Die Industrialisierung der IT zeigt sich in der zunehmend industrialisierten Produktion von IT-Hardware, der zunehmenden Bedeutung von Standardsoftware-Produkten und in Veränderungen bei der Erstellung von IT-Services. Im Zuge der IT-Industrialisierung steigt die Bedeutung der Dienstleistungs-, Prozess- und Architekturorientierung von IT-Services und es kommt in der IT zu einer Veränderung der Preise, der IT-Produkte, einer veränderten Produktion von IT, zu Änderungen bei IT-Personal sowie einer veränderten Zusammenarbeit mit externen Partnern.

Um die Potentiale einer der IT-Industrialisierung zukünftig voll auszuschöpfen, müssen IT-Dienstleister ihre Aufmerksamkeit den in diesem Beitrag angesprochenen Themen schenken. Dabei sollte die IT-Leistungserstellung neu strukturiert werden, damit die Möglichkeiten zur Optimierung der Leistungstiefe durch Auslagerung einzelner Komponenten voll ausgeschöpft werden können. In einer vollständig industrialisierten IT werden standardisierte Service-Produkte mit automatisierten, fast komplett standardisierten Prozessen durch spezialisierte Anbieter global verteilt erstellt. IT-Dienstleister müssen dabei auf die veränderten Kundenerwartungen und die Veränderungen in der Branche reagieren bzw. diese vorwegnehmen, um weiterhin erfolgreich zu sein.

Literaturverzeichnis

[Bi05] BITKOM: Kennzahlen zur ITK-Branchenentwicklung Herbst 2005. BITKOM, Berlin, 2005.

[BD06] Barros, A.; Dumas, M.: The rise of web service ecosystems. In: IT Professional 8 (2006) 5, S. 31-37.

[BK01] Brown, R.H.; Karamouzis, F.: The Services Value Chain: Forging the Links of Services and Sourcing. Gartner Research, 2001.

[BK04] Böhmann, T.; Krcmar, H.: Grundlagen und Entwicklungstrends im IT-Servicemanagement. In: HMD - Praxis der Wirtschaftsinformatik (2004) 237, S. 7-21.

[BTK06] Böhmann, T.; Taurel, W.; Krcmar, H.: Produktmanagement für IT-Dienstleistungen in Deutschland (Arbeitspapiere des Lehrstuhls für Wirtschaftsinformatik). Technische Universität München, München, 2006.
[Ca03] Carr, N.G.: IT doesn't matter. In: Harvard Business Review 81 (2003) 5, S. 41-49.
[Ca06] CapGemini: Studie IT-Trends 2006: Unterschiedliche Signale: konsequent sparen, gezielt investieren. 2006.
[CN02] Clements, P.; Northrop, L.: Software Product Lines: Practices and Patterns. Addison Wesley Professional, 2002.
[CS07] CapGemini; sd&m: Studie IT-Trends 2007: IT ermöglicht neue Freiheitsgrade. 2007.
[En82] Engels, F.: Die Entwicklung des Sozialismus von der Utopie zur Wissenschaft. 1882.
[Ho07] Hochstein, A.; Ebert, N.; Übernickel, F.; Brenner, W.: IT-Industrialisierung: Was ist das? In: Computerwoche (2007) 15, S. 5.
[Ja05] Janßen, R.: Die Psychologie des Entwicklers. In: Informatik-Spektrum 28 (2005) 4, S. 284-286.
[KRS04] Kittlaus, H.-B.; Rau, C.; Schulz, J.: Software-Produkt-Management. Springer, Berlin, Heidelberg, 2004.
[KTV07] Kilian-Kehr, R.; Terzidis, O.; Voelz, D.: Industrialisation of the Software Sector. In: Wirtschaftsinformatik 49 (2007) Sonderheft, S. 62-71.
[La69] Landes, D.S.: The unbound Prometheus : technological change and industrial development in Western Europe from 1750 to the present. Cambridge University Press, London, 1969.
[MA07] Meister, J.; Appelrath, H.-J.: Produktgetriebene Entwicklung von Softwareproduktlinien. In: Wirtschaftsinformatik 49 (2007) 3, S. 180-187.
[MS07] Meyer, T.; Stobbe, A.: Offshoring: Welche "Standorte" wählen deutsche Unternehmen? In: Wirtschaftsinformatik 49 (2007) Sonderheft, S. 81-89.
[PS84] Piore, M.J.; Sabel, C.F.: The second industrial divide: possibilities for prosperity. Basic Books, New York, NY, 1984.
[PT07] Padberg, F.; Tichy, W.: Schlanke Produktionsweisen in der modernen Softwareentwicklung. In: Wirtschaftsinformatik 49 (2007) 3, S. 162-170.
[PW07] Pfeiffer, D.; Winkelmann, A.: Ansätze zur Wiederverwendung von Software im Rahmen der Softwareindustrialisierung am Beispiel von Softwarekomponenten, serviceorientierten Architekturen und modellgetriebenen Architekturen. In: Wirtschaftsinformatik 49 (2007) 3, S. 208-216.
[Sc06] Scardino, L.; Potter, K.; Young, A.; Stone, L.; Da Rold, C.; Huntley, H.; Dreyfuss, C.; Longwood, J.; Tramacere, G.; Maurer, W.: Gartner on Outsourcing, 2006-2007 (Gartner Research). Stamford, CT, 2006.
[Ta05] Taubner, D.: Software-Industrialisierung. In: Informatik-Spektrum 28 (2005) 4, S. 292-296.
[WK06] Walter, S.M.; Krcmar, H.: Reorganisation der IT-Prozesse auf Basis von Referenzmodellen: eine kritische Analyse. In: itService Management: Zeitschrift des itSMF Deutschland e.V. 1 (2006) 2, S. 3-9.
[WBK07] Walter, S.M.; Böhmann, T.; Krcmar, H.: Industrialisierung der IT: Grundlagen, Merkmale und Ausprägungen eines Trends. In: HMD - Praxis der Wirtschaftsinformatik 256 (2007) S. 6-16.
[ZB04] Zarnekow, R.; Brenner, W.: Integriertes Informationsmanagement: Vom Plan, Build, Run zum Source, Make, Deliver. In: Zarnekow, R.; Brenner, W.; Grohmann, H.H. (Hrsg.): Informationsmanagement: Konzepte und Strategien für die Praxis. dpunkt, Heidelberg, 2004, S. 3-24.

Professionalisierung der Softwareentwicklung - ein Praxisbericht

Bruno Schienmann, Ulrich Schmedt

Qualität und Standards
Postbank Systems AG
Kennedyallee 62-70
53175 Bonn
bruno.schienmann@postbank.de
ulrich.schmedt@postbank.de

Abstract: In diesem Beitrag werden Maßnahmen und Erfahrungen der Postbank Systems AG zur Professionalisierung der Softwareentwicklung vorgestellt. Die beschriebenen Schritte wurden seit der Gründung der Postbank Systems AG im Jahr 2000 schrittweise umgesetzt und in einem kontinuierlichen Verbesserungsprozess angepasst. Am Beispiel der Business Analyse werden konkrete Beispiele für Professionalisierungsmaßnahmen beschrieben.

1 Übersicht

Die Postbank Systems AG ist der IT-Dienstleister der Deutsche Postbank AG. Das Leistungsspektrum umfasst alle Produkte des IT-Betriebs sowie die IT-Projekte. Die Postbank Systems beschäftigt ca. 1500 Mitarbeiter. Im Februar 2008 hat die Postbank Systems als erster Finanzdienstleister in Deutschland den Reifegrad 2 nach CMMI erreicht (Capability Maturity Model Integration [CKS06]. Grundlage für das Erreichen dieses Reifegrads war eine Reihe von Maßnahmen in den Bereichen:

- **Prozesse**: Prozessorientierung und -verbesserung bilden das Rückgrat der Professionalisierungsschritte. Definierte Entwicklungsprozesse und Reifegradmodelle wie CMMI schaffen in Verbindung mit regelmäßigen Prozessprüfungen die Grundlage für organisationales Lernen - kontinuierliche Verbesserung ist ohne Prozessorientierung nur schwer denkbar.

- **Personen**: Prozesse sind das „Rückgrat" einer Professionalisierung, Personen bilden den „Kopf". Besonders deutlich wird dies bei der Einführung agiler Prozesse, welche z. B. Cockburn auch als „Cooperative Game" interpretiert [Co02]. Als zentrale Maßnahmen haben sich hier die an Prozessen orientierte Spezialisierung von Mitarbeiter und darauf aufbauend deren zielgerichtete Qualifizierung bewährt.

- **Produkte**: Die Postbank Systems hat von Beginn an konsequent auf eine Standardisierung der Produkte Wert gelegt. Dies betrifft sowohl Applikationen mit der Einführung von Standardsoftware, als auch Entwicklungsergebnisse wie Vorstudien oder Fachkonzepte. Zusammen mit der Standardisierung wird eine höhere Automatisierung der Entwicklung durch Entwicklungswerkzeuge angestrebt.

Generell hat es sich gezeigt, dass Professionalisierungsmaßnahmen alle drei Bereiche Prozesse, Personen und Produkte balanciert berücksichtigen müssen, um nachhaltig wirksam zu sein. Als kritischer Erfolgsfaktor hat sich dabei die Ableitung dieser Maßnahmen aus den Unternehmenszielen und Verankerung dieser Maßnahmen in Policies erwiesen [GK08]:

- **Policies**: Explizit formulierte Willenserklärungen des Managements sind als Ausgangspunkt von Professionalisierungsmaßnahmen unbedingt zu empfehlen, um messbare Ziele vorzugeben und die Zielerreichung im Rahmen von Management-Reviews überprüfen zu können.

Die folgende Abbildung fasst diese Bereiche und die zugeordneten zentralen Professionalisierungsmaßnahmen zusammen.

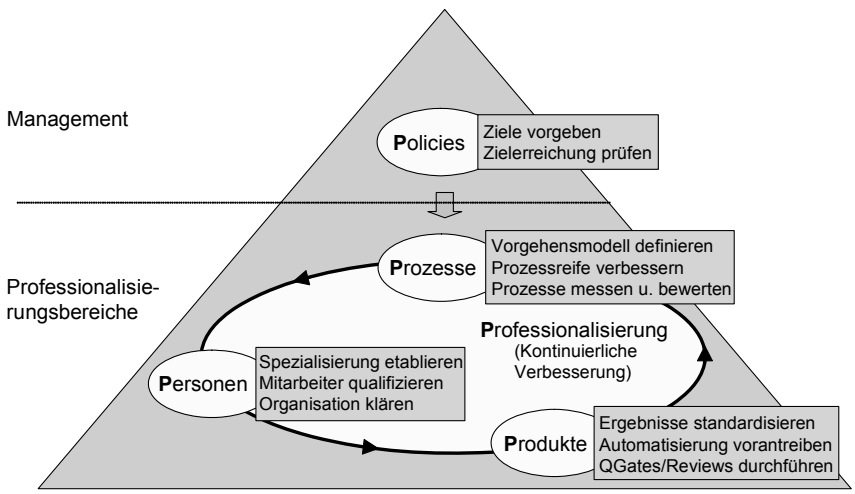


Abbildung 1: Die P der Professionalisierung

Da Investitionen in Entwicklungswerkzeuge ohne entsprechende Prozessgestaltung und Qualifizierungsmaßnahmen verpuffen oder bestenfalls nicht das mögliche Potenzial an Effizienzsteigerung ausschöpfen, fokussiert auch die aktuelle Diskussion bei Themen wie *Software-Fabrik* und *Industrialisierung der SW-Entwicklung* zunehmend stärker auf Prozesse und betroffene Mitarbeiter und stellt nicht nur die Automatisierung in den Vordergrund.

2 Policies und Durchführung von Professionalisierungsmaßnahmen

„Das »Zum-Leben-Bringen« von Arbeitsweisen ist die eigentliche Aufgabe von Verbesserungsinitiativen – eine Aufgabe, die häufig übersehen und meistens völlig unterschätzt wird." [FSR08, S. XV]. Kleinere Professionalisierungsmaßnahmen können durchaus im Rahmen der Linientätigkeit durchgeführt werden. Größere Verbesserungen, wie etwa die Umstellung von Entwicklungsprozessen oder gar die Erreichung eines CMMI-Reifegrads, sollten aber als eigenes Projekt mit einer breiten Beteiligung von Betroffenen gefahren werden. In jedem Fall müssen die konkrete Unterstützung des Managements im Sinne von Zielvorgaben und die laufende Kontrolle der Zielerreichung sichergestellt sein. In der Postbank Systems hat sich dazu das Instrument der Policies in Verbindung mit Management-Reviews bewährt.

Zu Beginn von Professionalisierungsmaßnahmen wird geprüft, ob diese konform zu Management-Policies sind. Ggf. werden diese überarbeitet oder angepasst. Erst wenn klar ist, was das zu erreichende Ziel konkret ist, wird die eigentliche Professionalisierungsmaßnahme durchgeführt. Als wirkungsvoll hat sich die Institutionalisierung dieses Verbesserungsmechanismus im Sinne des PDCA-Zyklus (Plan–Do–Check–Act) erwiesen, wobei in jedem Zyklus die drei Elemente *Prozess*, *Person* und *Produkt* betrachtet werden (vgl. das Beispiel in Abbildung 2).

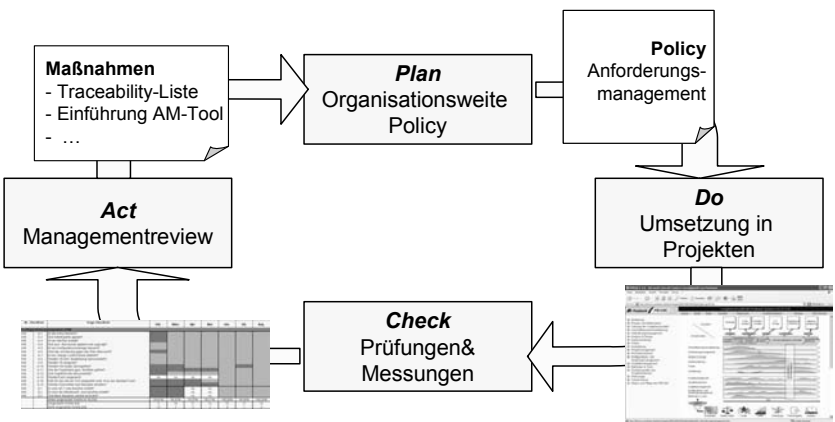


Abbildung 2: Zyklus der Durchführung von Verbesserungsmaßnahmen

- **Plan:** In einer organisationsweit verbindlichen Policy werden zunächst die generellen Zielsetzungen aus Managementsicht festgelegt oder angepasst.
- **Do:** Zur Erreichung dieser Ziele werden konkrete Maßnahmen definiert, im Vorgehensmodell beschrieben und in den Entwicklungsprojekten umgesetzt.
- **Check:** Im Rahmen von Prozessaudits und Produktreviews wird geprüft, in wie weit die aufgesetzten Maßnahmen die Ziele erreichen.
- **Act:** In regelmäßigen Abständen (z. B. monatlich) wird der Stand der Umsetzung dem Management berichtet und ggfs. Ziele oder Maßnahmen angepasst.

Als Beispiel für den ersten Schritt zeigt Abbildung 3 einen Ausschnitt der Management-Policy zum Anforderungsmanagement in der Postbank Systems.

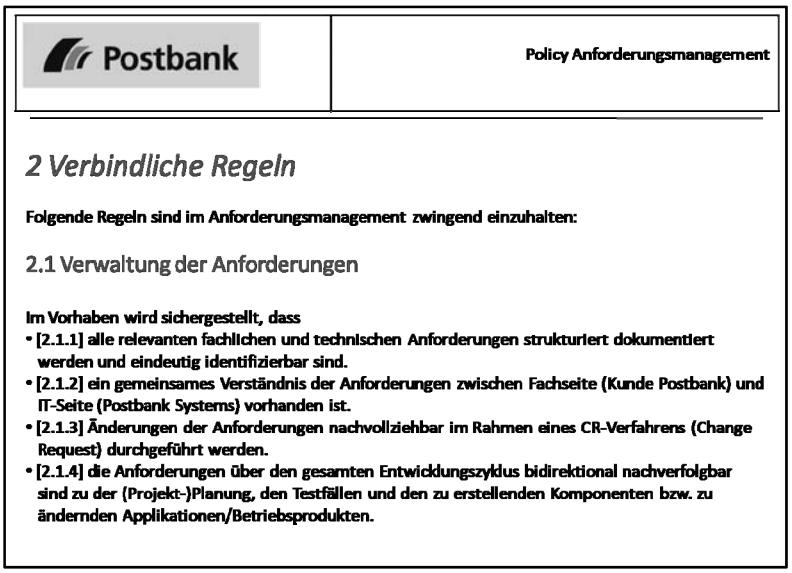


Abbildung 3: Ausschnitt der Policy zum Anforderungsmanagement

Häufig wird in Entwicklungsorganisationen der dritte Schritt (Check) vernachlässigt. Gerade diese Schaffung von Transparenz zum Umsetzungsfortschritt hat sich aber als ein ganz entscheidender Erfolgsfaktor der Professionalisierung erwiesen – W.S. Humphrey bringt dies mit dem bekannten Zitat auf den Punkt: „When you don´t know, where you are, a map won´t help." [Hu89]. Abbildung 4 stellt exemplarisch die ermittelte Prozessreife im Anforderungsmanagement verdichtet über alle Entwicklungsprojekte der Postbank Systems im Rahmen des CMMI-Projektes dar.

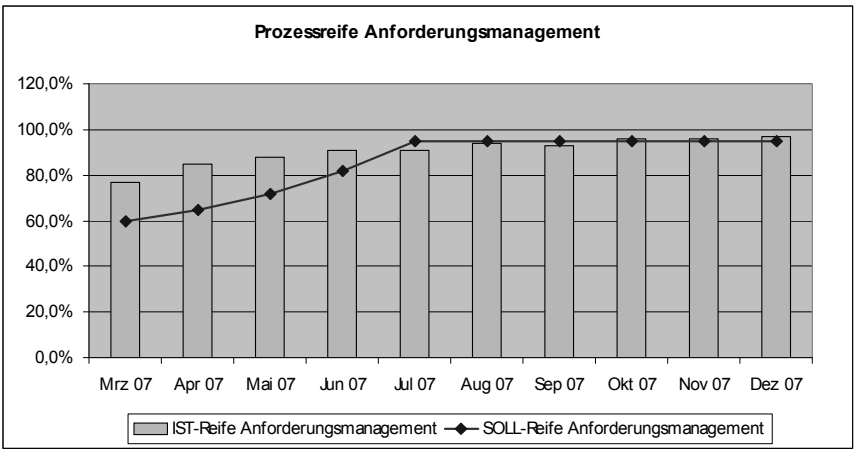


Abbildung 4: Prozessreife im Anforderungsmanagement

Erst anhand einer solchen Analyse kann auf Managementebene schnell gegengesteuert oder neue Ziele vorgegeben werden. Als Spiegel zeigen solche Analysen auf, wo eine Organisation auf dem Weg von „wir wollen“ über „wir können“ zu „wir machen“ steht.

3 Prozesse

Prozesse spielen in jeder Professionalisierungsmaßnahme eine solche zentrale Rolle, dass Prozessverbesserung und Professionalisierung oft synonym verwendet wird. Nachhaltige Professionalisierung basiert zumeist auf Prozessen und deren Verbesserung (vgl. zu verschiedenen Ansätzen des *Software Process Improvement* etwa [Wa07]).

Ab 2001 wurde in der Postbank Systems deshalb schrittweise ein neues Vorgehensmodell (PBVGM) für die SW-Entwicklung aufgebaut, das an den Rational Unified Process (RUP) angelehnt ist [SSS06]. Das PBVGM ist seit Ende 2002 der Standard für alle Entwicklungsprojekte. Die Entwicklung eines Vorgehensmodells ist im Sinne der Professionalisierung ein zentrales Instrument, um gemachte Erfahrungen in Projekten festzuhalten und anderen Mitarbeiter verfügbar zu machen.

Dabei hat sich bewährt, die Weiterentwicklung des PBVGM verantwortlich in einer Einheit zu lassen, die Fortschreibung der Inhalte aber über sog. Communities zu treiben. Abbildung 5 stellt die HTML-Version des PBVGM in der aktuellen Version 3 mit der Einstiegsseite dar. Über den CMMI-Icon in der Bildmitte unten können Mitarbeiter z. B. zu allen CMMI-spezifischen Verbesserungsmaßnahmen navigieren und damit implizit auf Erfahrungen anderer Projekte zur CMMI-Umsetzung zugreifen.

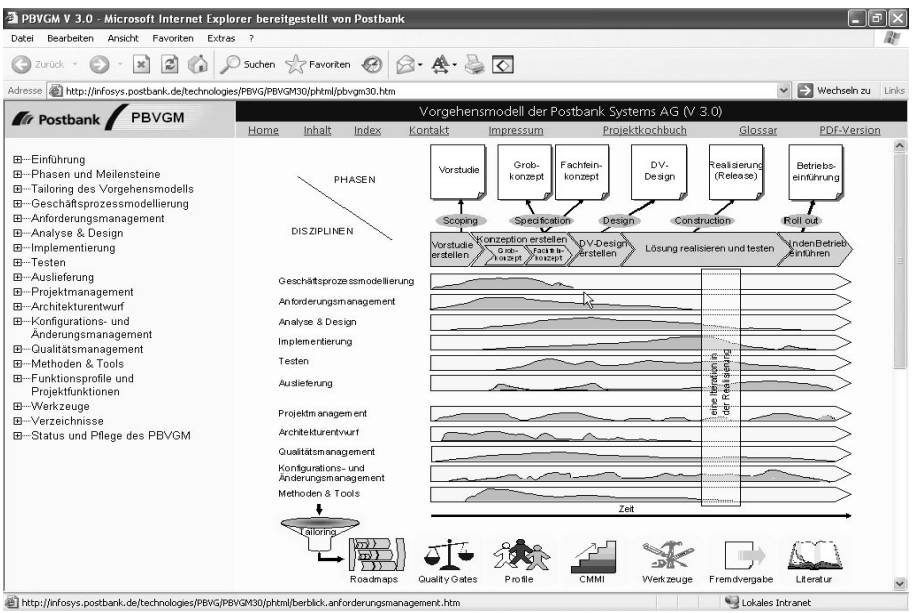


Abbildung 5: Vorgehensmodell der Postbank Systems

Ein Vorgehensmodell definiert, was zu tun ist. Reifegradmodelle beschreiben Prozessziele sowie bewährte Praktiken und geben Empfehlungen, in welcher Reihenfolge welche Prozesse zu verbessern sind. CMMI bietet für Entwicklungsorganisationen anhand der Reifegradstufen eine Roadmap für die aufeinander aufbauende Professionalisierung von Entwicklungsprozessen. Es wird deshalb mittlerweile in vielen Unternehmen als strategischer Weg zur Verbesserung der Entwicklungsprozesse beschritten. Vorreiter in Deutschland waren Unternehmen wie Siemens oder Bosch. Mittlerweile etabliert sich CMMI zunehmend auch bei fast allen großen Finanzdienstleistern Deutschlands.

CMMI ist ein fünfstufiges Modell zur Beurteilung und Verbesserung der Qualität („Reife“) von Entwicklungsprozessen auf der Grundlage von „Best Practices“ [CKS06]. Es ordnet den Reifegraden verschiedene Prozessgebiete mit Zielen und Praktiken zu, die in Entwicklungsprozessen umzusetzen sind, falls dieser Reifegrad erreicht werden soll. CMMI unterscheidet in der aktuellen Version 1.2 insgesamt 22 Prozessgebiete mit etwas über 400 Praktiken [Kn07, S. 25]. Abbildung 6 skizziert die CMMI-Reifegradstufen und führt beispielhaft die für Reifegrad 2 relevanten sieben Prozessgebiete auf (Reifegrad 1 sind keine Prozessgebiete zugeordnet).

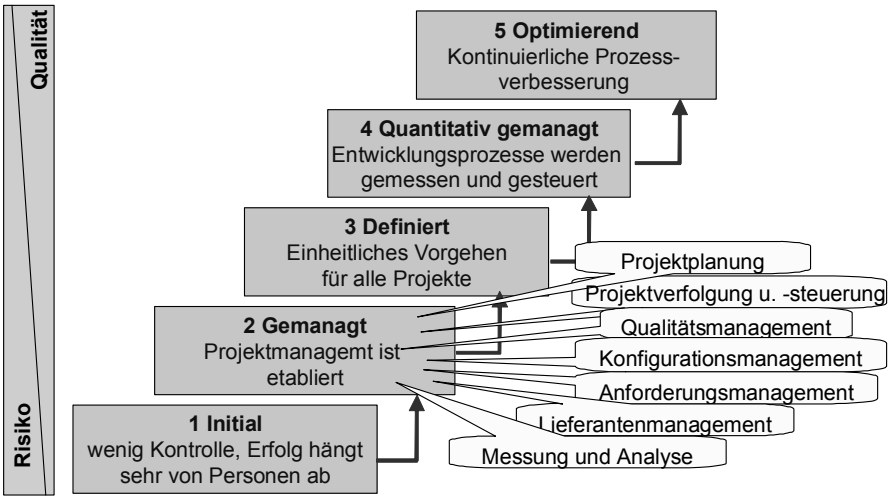


Abbildung 6: CMMI-Reifegradstufen und Prozessgebiete für Reifegrad 2 [Kn07, S. 19]

Im Reifegrad 2 steht die Professionalisierung der Projektmanagementprozesse und wichtiger Unterstützungsprozesse im Mittelpunkt: Projekte werden geplant, gesteuert und quantitativ mit Kennzahlen gemessen. Eine Qualitätssicherung von Prozessen und Produkten erfolgt. Ergebnisse werden versioniert und unterliegen einem Konfigurationsmanagement. Die Schnittstellen zu Kunden und Lieferanten werden über das Anforderungsmanagement und das Management der Lieferantenvereinbarungen im Projekt systematisch behandelt.

Reifegrad 3 erweitert den Fokus über das Projektmanagement hinaus auf alle anderen Entwicklungsaufgaben (z. B. Implementierung, Testen oder Produktintegration) und fordert darüber hinaus ein einheitliches Projektvorgehen. Reifegrad 4 und 5 führen dann zu einer statistischen Prozesskontrolle und darauf aufbauend einer statistisch basierten kontinuierlichen Prozessverbesserung.

Die Postbank Systems setzt das CMMI-Modell seit gut zwei Jahren in der Softwareentwicklung ein, nachdem zuvor auch Alternativen wie etwa SPICE (Software Process Improvement and Capabilty Determination) evaluiert wurden. Zunächst erfolgte zur Positionsbestimmung und Identifizierung von Verbesserungsmaßnahmen eine Begutachtung der gelebten Entwicklungsprozesse in der Postbank Systems im Rahmen eines sog. SCAMPI-B-Appraisals (Standard CMMI Appraisal Method for Process Improvement) durch externe Gutachter. Die empfohlenen Maßnahmen wurden als sinnvoll erachtet und ein Verbesserungsprojekt zur Erreichung des Reifegrads 2 aufgesetzt. Nach zwei Jahren wurde dieser Reifegrad in einem SCAMPI-A-Assessment bestätigt.

In der Postbank Systems hat sich CMMI als Referenzmodell für die Verbesserung der Entwicklungsprozesse bisher bewährt. Das Ergebnis der Begutachtungen mit den identifizierten Stärken und Schwächen und die daraus resultierenden Empfehlungen wiesen einen geeigneten Weg, Entwicklungsprozesse systematisch zu verbessern. Anfänglich durchaus vorhandene Bedenken und Widerstände auf der operativen Ebene konnten durch *Pilot*- oder *Leuchtturmprojekte* abgebaut werden.

Speziell in der Business Analyse bzw. im Anforderungsmanagement ergab das SCAMPI-B-Appraisal, dass geforderte Praktiken wie etwa „Verständnis über Anforderungen herbeiführen“ oder „Anforderungsänderungen managen“ bereits im PBVGM verankert und in den Projekten gelebt werden, die Praktik „Bidirektionale Nachverfolgbarkeit der Anforderungen aufrechterhalten“ aber bisher erst teilweise umgesetzt ist. Eine Nachverfolgbarkeit (Traceability) war zwar in vielen Fällen zwischen verschiedenen Ergebnissen möglich (z. B. zwischen Anforderungen und Testfällen). Oft wurden diese Beziehungen aber nur in eine Richtung gepflegt (d.h. nicht bidirektional) und auch nicht generell zu allen wichtigen Entwicklungsergebnissen aufrechterhalten, z. B. zur Projektplanung oder zur IT-Architektur.

Die Prozessverbesserung zum Thema Nachverfolgbarkeit folgte dem in Abbildung 1 angegeben Schema. Zunächst wurde auf der Grundlage der Management-Policy (vgl. Abbildung 3, Regel 2.1.4) die Nachverfolgbarkeit im Vorgehensmodell verankert. Anschließend wurden die verantwortlichen Mitarbeiter durch eintägige Trainings und begleitende Coaching-Maßnahmen qualifiziert. Die Pilotierung der Prozesse führte schließlich zu Standard-Vorlagen, die in den Projekten zur Sicherstellung der Nachverfolgbarkeit genutzt werden.

Eine wichtige Erfahrung im Rahmen der Professionalisierung betrifft die Verbindung CMMI und Vorgehensmodell. CMMI fordert unternehmensweite Prozessstandards prinzipiell erst auf Ebene 3. Zumindest jede größere Organisation ist jedoch gut beraten, auch schon für Ebene 2 ein unternehmensweites Vorgehensmodell zu etablieren. Ansonsten können Verbesserungsmaßnahmen nur schwer mit der notwendigen Geschwindigkeit umfassend eingeführt werden. Außerdem können Qualitätsfortschritte praktisch nicht mit angemessenem Aufwand gemessen werden, da schlichtweg die Vergleichbarkeit über Projektgrenzen hinaus fehlt.

Diese Vergleichbarkeit ist wichtig, um den organisationsweiten Fortschritt von Prozessverbesserungsmaßnahmen über Prozessaudits oder Kennzahlen in monatlichen Statusberichten messen zu können. Insbesondere die Durchführung regelmäßiger Prozessaudits hat sich als sehr effizient dafür erwiesen, Transparenz über die Umsetzung von Prozessänderungen zu erhalten und frühzeitig steuernd eingreifen zu können, falls vorgegebene Ziele nicht erreicht werden können.

4 Personen

Software-Entwicklung beruht trotz aller Automatisierungsfortschritte und Industrialisierungsansätze noch immer hauptsächlich auf menschlicher Arbeit. Die Qualität der Entwicklungsergebnisse wird wesentlich von der Motivation und Qualifikation der Mitarbeiter bestimmt. Jede Professionalisierungsmaßnahme kann deshalb nur so gut wirken, wie diese die beteiligten Personen berücksichtigt, einbezieht und entwickelt.

Aufgrund der Komplexität und Dynamik der Software-Entwicklung sind Spezialisierung und Qualifizierung zwei zentrale Professionalisierungsmaßnahmen. In der Postbank Systems wird diese Spezialisierung durch sogenannte Funktionsprofile erreicht. Jedem Mitarbeiter ist ein Funktionsprofil im Sinne einer klassischen „Stellenbeschreibung“ *dauerhaft* zugeordnet, wobei diese Profile grundsätzlich prozessorientiert sind.

Im Bereich der Software-Entwicklung orientierten sich die Funktionsprofile an den Disziplinen und Aufgaben des Vorgehensmodells. Abbildung 7 stellt die hauptsächlichen Profile und zugeordnete Disziplinen dar. Die Verantwortung für die Geschäftsprozessmodellierung liegt bei der Fachseite bzw. dem Kunden. Das Anforderungsmanagement und die fachliche Analyse obliegt dem Business Analyst. Im Bereich der Geschäftsprozessmodellierung hat dieser aber eine wichtige beratende Funktion, um das Optimierungspotenzial der Informationstechnologie frühzeitig aufzuzeigen. Das Design der Anwendung, die Implementierung und die Modultests erfolgen durch den SW-Entwickler. Der Systemintegrator führt die Auslieferung durch und bildet die Schnittstelle zum Betrieb. Das Projektmanagement wird durch den Projektleiter verantwortet, der Architekturentwurf durch den Architekten. Das Qualitätsmanagement im Projekt führt der Qualitätsverantwortliche durch. Er sorgt auch für notwendige Anpassungen der Projektvorgaben (Methoden, Vorlagen, Richtlinien). Das Buildmanagement und technische Konfigurationsmanagement liegt beim Software-Entwickler, das Änderungsmanagement, die Planung und das Dokumentenmanagement beim Konfigurationsmanager.

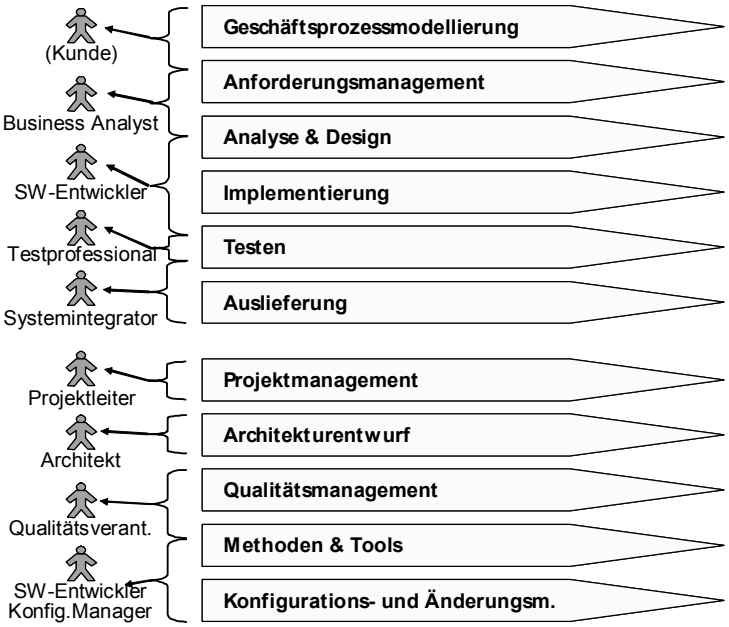


Abbildung 7: Funktionsprofile in der Softwareentwicklung

Eine solche prozessbezogene Spezialisierung weg vom „Zehnkämpfer“ ermöglicht es Mitarbeitern, in verschieden Projekten zu arbeiten, ihr Tätigkeitsfeld interessanter zu gestalten und zu erweitern. Dem Ressourcenmanagement bietet es eine größere Flexibilität bei der Personaleinsatzplanung, da Mitarbeiter fachgebiet- und applikationsübergreifend eingesetzt werden können. Die Spezialisierung der Mitarbeiter sollte allerdings nicht übertrieben werden. Ansonsten führt dies gerade in kleinen Vorhaben zu „Reibungsverlusten“. Auch können Mitarbeiter dann nicht mehr vernünftig in Projekten ausgelastet werden. Gerade in der Softwareentwicklung muss den oft hochqualifizierten Mitarbeitern auch weiterhin genügend Freiraum für agile Vorgehensweisen gegeben werden – d. h. es muss durchaus nicht jedes Detail in der Prozessen verbindlich geregelt werden (hier bestehen sicherlich auch kulturelle Unterschiede etwa zu Amerika oder Indien).

Funktionsprofile ermöglichen in der Postbank Systems die Durchführung zielgerichteter, prozessorientierter Qualifizierungsmaßnahmen. Nachdem man zunächst gute Erfahrungen im Projektmanagement sammelte, wurden vor ca. 3 Jahren umfangreiche Qualifizierungsmaßnahmen mit einer Reihe von Trainingsbausteinen für die Business Analyse bzw. das Anforderungsmanagement aufgesetzt (vgl. Abbildung 6, rechte Seite). Dieses Trainingskonzept und die einzelnen Trainingsbausteine wurden zusammen mit der Personalentwicklung und den Führungskräften der Business Analysten konzipiert. Grundlage dafür war eine Profilanalyse - welche Qualifikationen werden von Business Analysten in Projekten erwartet?

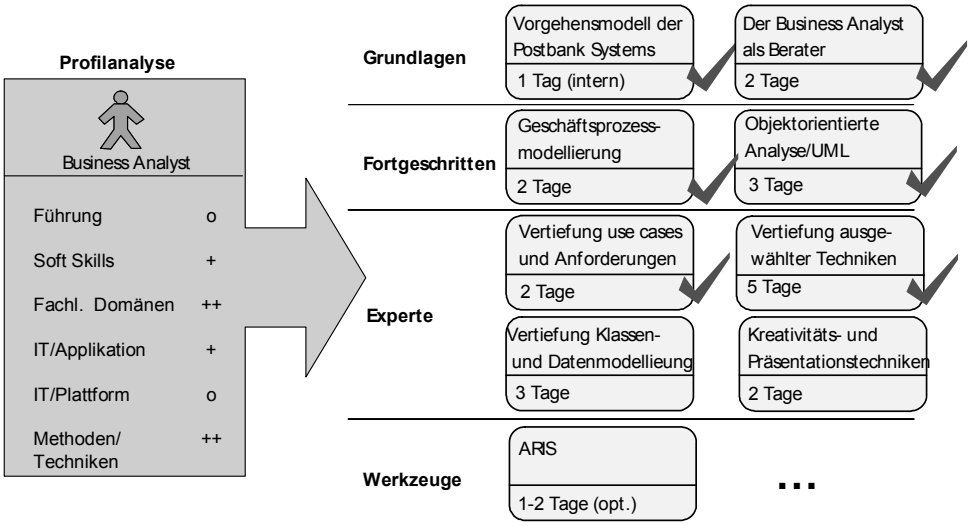


Abbildung 8: Trainingskonzept für Business Analysten

In den Trainings werden primär Methoden und Techniken der Business Analyse vermittelt, nicht konkrete Ergebnistypen oder Richtlinien. Ziel der Trainings ist es, die Business Analysten zu befähigen, die für das jeweilige Vorhaben geeigneten Methoden auszuwählen und anzupassen und weniger, vorgegeben Vorlagen auszufüllen. Mit der detaillierten Ausarbeitung und der Durchführung der Trainings wurde ein renommierter Trainingsanbieter beauftragt. Die markierten Trainingsbausteine werden gebündelt innerhalb von drei Wochen mit ca. 10-12 Teilnehmern durchgeführt. Inzwischen wurden die Trainings von ca. hundert Business Analysten durchlaufen. Spezifische vertiefende Trainings etwa zu Werkzeugen, zur Fachdomäne oder zu Applikationen erfolgen anschließend bedarfsabhängig.

Solche umfangreichen Trainingsmaßnahmen sind natürlich eher die Ausnahme. In der Regel erfolgt die durch Prozessverbesserungsmaßnahmen angestoßene Qualifizierung eher in kleinen „Dosen“, etwa durch ein Coaching bei der täglichen Arbeit oder durch kurze Workshops. Bewährt hat sich auch die Bildung von Communities, um einen direkten Austausch der Praktiker zu ermöglichen und Verbesserungsmaßnahmen zu diskutieren. Erste Erfahrungen mit Wikis liegen vor, eine breitere Nutzung muss sich noch etablieren.

Neben Spezialisierung und Qualifizierung ist in Abbildung 1 als dritter Punkt die Organisation aufgeführt. Damit ist zum einen die Klärung aufbauorganisatorischer Fragen gemeint – etwa wer führt Prozesse durch? - zum anderen aber auch Punkte wie angestrebte Fertigungstiefe bzw. Sourcing in der Softwareentwicklung oder die Klärung der Kernkompetenzen, die wichtige Rahmenbedingungen für Professionalisierungsmaßnahmen darstellen. Die Postbank Systems strebt z. B. im Anforderungsmanagement eine hohe interne Fertigungstiefe an und sieht hierin eine wichtige Kernkompetenz. Entsprechend haben Qualifizierungsmaßnahmen für Mitarbeiter in diesem Bereich hohe Priorität.

5 Produkte

Kostenführerschaft ist für die Postbank eine zentrale Wettbewerbsstrategie. Typische Ziele für diese Strategie sind das Erreichen von Skaleneffekten, gute Kapazitätsausnutzung, effizientes, einfaches Prozessdesign und hohe Produktstandardisierung [Gr02]. In der Postbank Systems bildet die Unterstützung dieser Ziele den Ausgangspunkt für die Gestaltung der Anwendungslandschaft und der Software-Entwicklung. Entsprechend haben die effiziente Gestaltung des Entwicklungsprozesses und eine hohe Standardisierung der IT-Produkte, d. h. sowohl der Applikationen als auch der IT-Konzepte, hohe Priorität.

Die Ausrichtung auf Standards hat im Bereich der Applikation etwa zur Entwicklung und Einführung eines neuen SAP-Kernbanksystems geführt. Hinsichtlich der IT-Konzepte betrifft die Vereinheitlichung sowohl einzelne Ergebnistypen und Notationen (z. B. Anwendungsfälle oder die UML), als auch umfassende Konzepte, wie etwa Vorstudien. Zusammen mit der Festlegung der Entwicklungsstandards hat es sich als zweckmäßig erwiesen, klare Verantwortlichkeiten für die Erarbeitung der Ergebnisse zu definieren. Abbildung 9 stellt exemplarisch die Struktur einer standardisierten Vorstudie im Sinne eines virtuellen Dokuments mit den Ergebnisverantwortlichkeiten über die Funktionsprofile dar (vgl. dazu Abbildung 7).

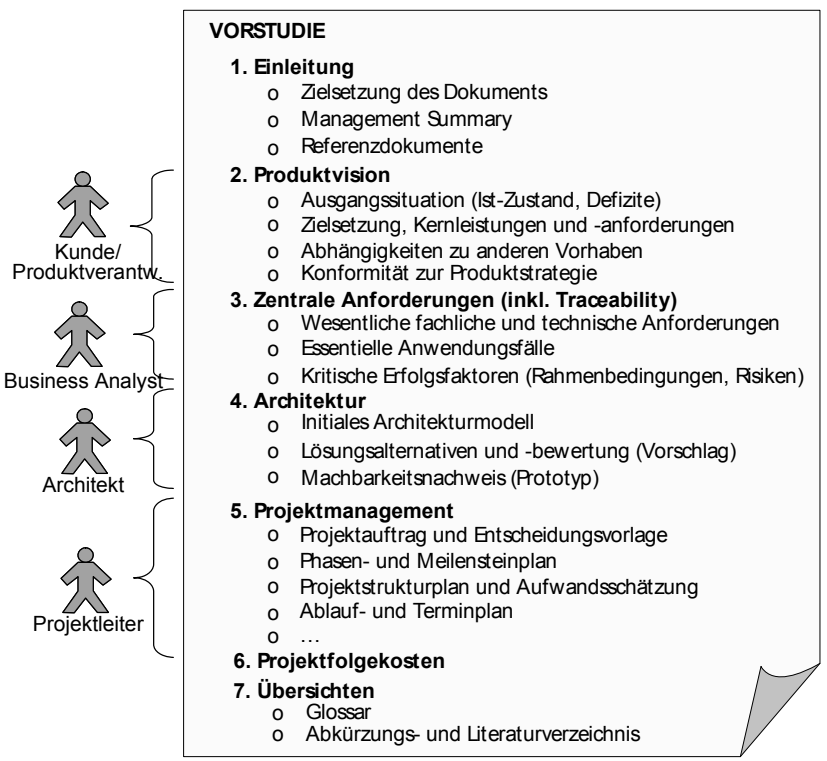


Abbildung 9: Standardvorlage für eine Vorstudie mit verantwortlichen Funktionsprofilen

Eine Vorstudie dient in der Postbank Systems dazu, dem Kunden ein verbindliches Angebot für das Entwicklungsprojekt abzugeben. Die Inhalte der Vorstudie beantworten im Wesentlichen die Fragen nach dem *warum* mit der Produktvision, nach dem *was* mit den zentralen Anforderungen, nach dem *womit* mit der Architektur und dem *wie* mit dem Projektmanagement. Im Rahmen der Vorstudie ist der Business Analyst für die Ermittlung der Anforderungen zuständig (im Sinne einer Scope-Definition des Projektes).

Die Standardisierung von Entwicklungsergebnissen hat sich als sehr nützliches Instrument für die Professionalisierung erwiesen. Standards erleichtern die Einarbeitung und Orientierung neuer Mitarbeiter und schaffen eine einheitliche, verbindliche Kommunikationsbasis. Dadurch, dass etwa generell Anwendungsfälle für die Anforderungsanalyse genutzt werden, können Verbesserungsmaßnahmen und Erfahrungen (z. B. Wie werden Anwendungsfälle gut spezifiziert? Wie erfolgt am besten die Abbildung in einer Traceability-Liste? Wie lassen sich daraus systematisch Testfälle ableiten?) sehr effizient ausgetauscht und von allen Business Analysten genutzt werden. Nach einer kurzen Vorbereitungsphase ist es dadurch beispielsweise gelungen, innerhalb von ca. 3-4 Monaten die von CMMI geforderte bidirektionale Nachverfolgbarkeit in allen Entwicklungsvorhaben der Postbank Systems einzuführen (vgl. dazu auch Abbildung 4 mit der Messung des Umsetzungsfortschritts).

Professionalisierung im Bereich der Softwareentwicklung führt auch immer zu der Frage der Automatisierung durch Entwicklungswerkzeuge. In Verbindung mit einer Standardisierung können dadurch Ergebnisse häufig sehr viel effizienter erstellt werden. Klassische „Werkzeuge“ im Bereich der Business Analyse sind natürlich die bekannten Office-Produkte. Für kleinere Vorhaben sind diese auch durchaus geeignet. In größeren Vorhaben muss die Nutzung spezifischer Werkzeuge aber noch intensiviert werden. In der Geschäftsprozessmodellierung betrifft sie beispielsweise die Nutzung der ARIS-Werkzeuge, im Anforderungsmanagement die Einführung spezifischer Werkzeuge zur Anforderungsverwaltung und zur Sicherstellung der Nachverfolgbarkeit.

Ähnlich wie im Bereich Prozesse sollte auch im Bereich Produkte der Fortschritt von Professionalisierungsmaßnahmen durch regelmäßige Prüfungen, etwa im Rahmen von Reviews, nachgewiesen werden. Ein wichtiger Schritt war in der Postbank Systems die Einführung von sog. Quality Gates [PSM07]. Insgesamt sind aktuell sieben Quality Gates über den gesamten Entwicklungsprozess verteilt. Im Gegensatz zu spezifischen Reviews wird in Quality Gates nicht nur ein bestimmtes Ergebnis geprüft sondern mehrere im Kontext der Phase wichtige Ergebnisse zusammenhängend analysiert. Dabei soll nicht nur bestätigt werden, dass die bisher entwickelten Ergebnisse bestimmten Qualitätskriterien genügen, sondern auch, dass alle Voraussetzungen für eine weitere erfolgreiche Entwicklung gegeben sind.

6 Abschließende Bemerkungen

Eine wichtige Frage jeder Professionalisierungsmaßnahme ist die Unterstützung durch externe Mitarbeiter. Hier sind zwei Aspekte zu unterscheiden.

- Externe Beratung ist sinnvoll, um punktuelle Maßnahmen schnell voran zu bringen und internes Wissen aufzubauen. Ein Beispiel wäre etwa eine spezielle Verbesserungsmaßnahme für eine Reifegraderfüllung nach CMMI. Hier empfiehlt es sich, externe CMMI-Berater hinzu zuzuziehen, um von deren Erfahrungen zu profitieren.

- Zum anderen kann eine externe Unterstützung notwendig sein, um die mit einer Professionalisierungsmaßnahme oft kurzfristig einhergehende Produktivitätsminderung aufzufangen. Dieser Aspekt wird in Abbildung 10 deutlich, welches den typischen Verlauf der Professionalisierung darstellt.

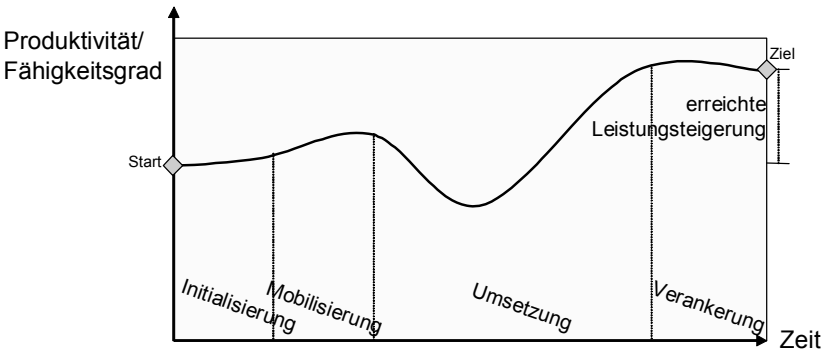


Abbildung 10: Typischer Verlauf von Professionalisierungsmaßnahmen [FSR08, S.89]

Nach dem Aufsetzen einer Verbesserungsmaßnahme werden in einer Mobilisierungsphase zunächst oft sehr schnell einige Produktivitätssteigerungen erzielt. Die breite Einführung führt dann aber häufig zu einer Reduzierung der Produktivität, weil der Umgang mit neuen Abläufen oder Werkzeugen erst eingeübt werden muss. Die Einführung eines komplexen Werkzeugs wie ARIS zur Geschäftsprozessmodellierung oder eines Anforderungsmanagements-Werkzeugs zur Nachverfolgung der Anforderungen kann beispielsweise durchaus zu einem (hoffentlich kurzfristigen) Produktivitätsknick führen, bis der Umgang mit den Werkzeugen zu einer langfristig effizienteren Business Analyse führt.

Ein solcher Produktivitätsknick ist natürlich ein kritischer Punkt jeder Professionalisierungsmaßnahme. Damit die Organisation nicht vorschnell in alte, vermeintlich bewährte Verfahren zurückkehrt, muss eine adäquate Unterstützung sichergestellt werden. Externe Mitarbeiter können helfen, die Kundenauswirkung gering zu halten und ein verbessertes Leistungsniveau schneller zu erreichen. Die anschließenden Verankerung bzw. Institutionalisierung der Verbesserungsmaßnahmen sollte dann allerdings mit internen Mitarbeitern erreicht werden. Eine gewisse Absenkung des Leistungsniveaus ist dieser Phase ist üblich und bedeutet letztlich, dass eine Kultur der kontinuierlichen Verbesserung etabliert werden muss, um nicht langfristig wieder auf das alte Niveau zurückzufallen.

Literaturverzeichnis

[CKS06] Chrissis, M.B.; Konrad, M.; Shrum, S.: CMMI – Guidelines for Process Integration and Product Improvement 2nd ed., Addison-Wesley 2006

[Co02] Cockburn, A: Agile Software Development, Addison-Wesley 2002

[FSR08] Foegen, M.; Solbach, M.; Raak, C.: Der Weg zur professionellen IT, Springer 2008

[GK08] Greb, T.; Kneuper, R.: Business IT Alignment mit CMMI, IT-Management 3/08

[Gr02] Grant, R.M.: Contampory Strategy Analysis, Concepts, Techniques, Applications 4th ed., Blackwell 2002

[Hu89] Humphrey, W.S.: Managing the Software Process, Addison-Wesley 1989

[Kn07] Kneuper, R.: CMMI, dpunkt 2007

[PSM07] Pfeifer, T., Schmitt, R., Masing, W.: Handbuch Qualitätsmanagement 5. Auflage, Hanser 2007

[Sc01] Schienmann, B.: Kontinuierliches Anforderungsmanagement, Addison-Wesley 2001

[SSS06] Schienmann, B.; Schmedt, U.; Swidurski, M.: Systematisches Tailoring von Vorgehensmodellen, OBJEKTspektrum 4/06

[Wa07] Wallmüller, E.: Software Process Improvement mit CMMI, PSP/TSP und ISO 155504, Hanser 2007

Adaption, Umsetzung, Grenzen und Nutzen von Six Sigma in der Softwareentwicklung

Martin Mikusz, Georg Herzwurm

Lehrstuhl für ABWL und Wirtschaftsinformatik II
Universität Stuttgart
Breitscheidstraße 2c
70174 Stuttgart
herzwurm@wi.uni-stuttgar.de
mikusz@wi.uni-stuttgart.de

Abstract: Der Beitrag befasst sich mit der Umsetzung des aus dem industriellen Kontext stammenden Prozessverbesserungsansatzes Six Sigma in der Softwareentwicklung. Um den Rahmenbedingungen der Softwareentwicklung gerecht zu werden, muss Six Sigma größtenteils durch Analogien und bereits vorhandene Ansätze in der Softwareentwicklung adaptiert werden. Hierauf wird einzeln für jede Phase des Six Sigma-Vorgehensmodells zur Qualitätsverbesserung und Prozessoptimierung „DMAIC“ eingegangen. Anschließend werden die Grenzen und Potentiale eines Software-spezifischen Six Sigma aufgezeigt.

1 Six Sigma

Six Sigma wurde in den 80er-Jahren bei Motorola entwickelt und ist ein auf statistischen Methoden basierender Prozessverbesserungsansatz. Die publizierten Erfolge von Six Sigma, beginnend 1988 mit Motorolas Gewinn des *Malcolm Baldridge Quality Award*, bis hin zu dem umfassenden Six Sigma-Einsatz bei General Electric, trugen ebenso zu der großen Beachtung in der produzierenden Industrie bei, wie auch mehrere Management-Bestseller [HS00]. Zunächst bei hochvolumiger industrieller Produktion eingesetzt, anschließend in der im Six Sigma-Jargon sog. *Zweiten Welle* im Dienstleistungssektor, wird Six Sigma aktuell in der sog. *Dritten Welle* in die Softwareentwicklung eingebracht.

Six Sigma liegt die Qualitätsphilosophie von *G. Taguchi* zugrunde. Variationen innerhalb der Prozessergebnisse lassen sich direkt auf Veränderungen der Prozesselemente und -aktivitäten als steuerbare Parameter zurückführen. Das Hauptaugenmerk von Six Sigma liegt dabei auf der Reduzierung von Variation bei den qualitätskritischen Steuerparametern, die damit das Ergebnis besonders stark beeinflussen [RY05, S. 27f.].

Gute Prozesse liefern fortlaufend qualitativ hochwertige Ergebnisse, die kaum variieren. In einem Prozess, der das Niveau von 6 *Sigma* erfüllt, entstehen bezogen auf 1 Million Möglichkeiten nur 3,4 fehlerhafte Ergebnisse (*DPMO; defects per million opportunities*). Es liegen also nur 3,4 von einer Million Werten außerhalb der gesetzten unteren und oberen Toleranzgrenze i. H. v. (*Mittelwert+/- 6 Sigma* bzw. *Standardabweichungen*). Das Prozessergebnis variiert damit nahezu überhaupt nicht. Die Wahrscheinlichkeit, dass eine Produkteinheit bei 3,4 *DPMO* den gesamten Produktionsprozess fehlerfrei passiert, die sog. *rolled throughput yield (rty), beträgt* 99,9966%. Die *rty,* bzw. die hierin konvertierbaren Größen *DPMO* und der *Sigma-Wert* eines Prozesses, sind die zentralen Metriken bei Six Sigma [HS00].

Six Sigma-Projekte folgen einer formalisierten Vorgehensweise mit den Projektphasen *Define*, *Measure*, *Analyze*, *Improve* und *Control* (*DMAIC*). Wie die Mehrzahl aller Vorgehensmodelle zur Qualitätsverbesserung und Prozessoptimierung ist auch *DMAIC* an das von *Deming* eingeführte PDCA-Modell (Plan, Do, Check, Act) angelehnt. Den einzelnen Projektphasen des *DMAIC* sind phasenspezifische Methoden sowie Meilensteine mit definierten Teilnehmern und Zwischenergebnissen als Phasenübergänge zugeordnet (vgl. Abbildung 1).

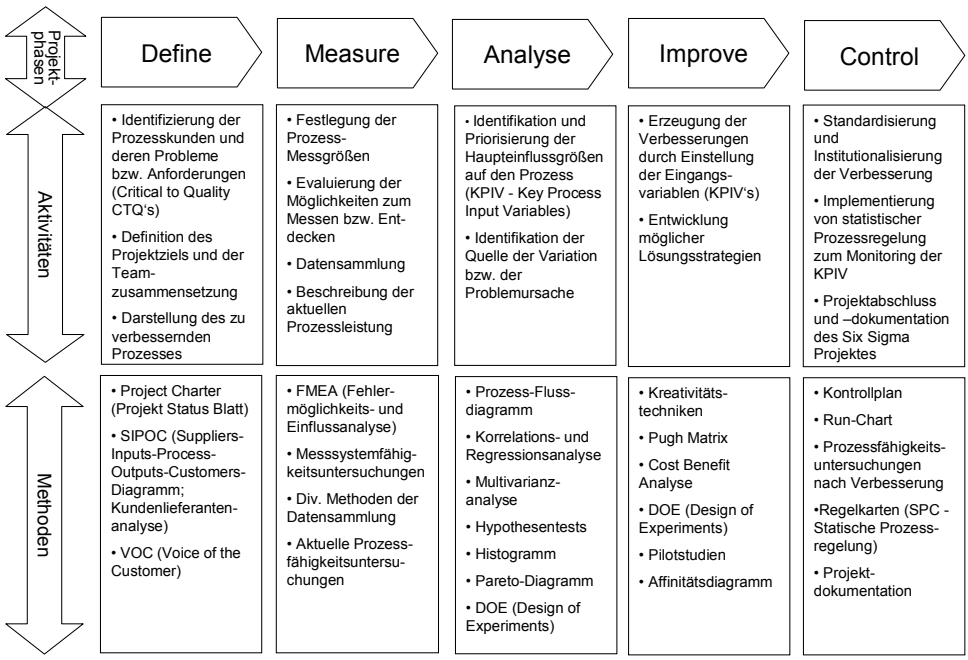


Abbildung 1: Six Sigma Projektphasen [RY05, S. 98]

Zum strikten Projektmanagement gehören des Weiteren definierte Rollen, die parallel zu ihrem ersten Six Sigma-Projekt eine entsprechende Ausbildung mit Schwerpunkt in statistischen Methoden und im Projektmanagement erfahren. Im Vordergrund stehen hier die sog. Black- und Green Belts. Black Belts übernehmen in Vollzeit die Rolle des Projektinitiators, -leiters und Change Managers. Sie werden durch die Green Belts bei der operativen Ausführung von Six Sigma-Projekten unterstützt. Die Dauer eines Six Sigma-Projektes wird üblicherweise auf ca. vier Monate ausgelegt.

Ursprünglich wurde von Motorola und General Electric ein vierstufiges Vorgehensmodell, das MAIC (Measure, Analyse, Improve, Control), entwickelt. Erst im Zuge der „Anreicherung" von Six Sigma wurde das *MAIC* um die *Define*-Phase erweitert, um so die Bedeutung des Projektziels, der Ressourcenplanung, und der eigentlichen Projektdefinition zu betonen. Die bedeutendste Rolle kommt jedoch der Datensammlung und -analyse bzw. der *Measure*- und *Analyse*-Phase zu. Alle Maßnahmen in einem Six Sigma Projekt sollen ausschließlich auf Basis von Daten bzw. Fakten festgelegt werden; Entscheidungen nach dem "Bauchgefühl" sind möglichst auszuschließen. Im Einzelnen bilden quantitative Daten

- die Grundlage für die Visualisierung des Ist-Prozesses und die Bestimmung seiner aktuellen Reife in der *Measure*-Phase, damit
- den Input für die *Analyse*-Phase, in der mit Hilfe der Messwerte die Problemursachen statistisch bestimmt werden, sowie damit
- die Grundlage für Entscheidungen bei der Auswahl der Lösungsalternativen in der *Improve*-Phase und für den Beweis deren Wirkung.

Six Sigma zielt also über den punktuellen Einsatz von statistischen Methoden hinaus auf die Entdeckung von Zusammenhängen zwischen den betrachteten Qualitätscharakteristika und Einflussfaktoren auf den Prozess. Verbesserungen werden aus diesen Beziehungen abgeleitet; vermutete Beziehungen werden gegen empirische Daten getestet, bevor sie als wahr akzeptiert werden. Der gesamte Problemlösungsprozess ist bei Six Sigma durch statistische Analysen (der Variation) als Erkenntnisprozess getrieben.

Zusammenfassend hebt sich Six Sigma v. a. durch drei Eigenschaften von konkurrierenden Ansätzen ab: a) rein datenorientierte Betrachtungsweise und auf statistischen Analysen beruhende Entscheidungsfindung, b) ein klar definiertes Rollenkonzept, sowie c) starke Projektfokussierung mit einer klar festgelegten Vorgehensweise bei der Lösungsfindung (*DMAIC*) [BA04]. Six Sigma ist jedoch keineswegs eine vollkommen neue Methodik und Denkweise. Elemente, die bereits früher vorhanden waren und in geschickter Weise mit den, ebenso wohlbekannten, statistischen Methoden als Six Sigmas methodischem Kern kombiniert wurden, sind Leadership-Ideen, Elemente von TQM wie die ausgesprochene Kunden- und Prozessorientierung, Teambildung, und der frühe Einbezug aller Betroffenen [Se03, S. 268].

2 Software-spezifische Umsetzung der Six Sigma-basierten Vorgehensweise *DMAIC*

2.1 Define-Phase

Hall et al. [HAB02] weisen in ihrer Sekundäranalyse von Erfolgsfaktorenuntersuchungen Merkmale erfolgreicher Prozessverbesserungen in der Softwareentwicklung nach. Interessanterweise bezieht sich keiner der Erfolgsfaktoren auf die konkreten (Verbesserungs-)Methoden, sondern ausschließlich auf die sog. *Soft Facts* (Management Commitment, Mitarbeiterbeteiligung, Kommunikation, Klarheit der Zielsetzung, u. a.). Gerade die Ziele und Aktivitäten der *Define*-Phase decken sich mit den wesentlichen Merkmalen erfolgreicher Prozessverbesserungen [HAB02]. So werden in der *Define*-Phase alle Prozesskunden bzw. Betroffenen identifiziert und mit einbezogen. Management Commitment wird durch die Nominierung eines Projekt-Champion aus dem Management in unmittelbarem Prozessumfeld sichergestellt. Die Festsetzung konkreter, von allen Beteiligten verstandener und akzeptierter Ziele ist Voraussetzung für den Übergang in die *Measure*-Phase. All diese Aktivitäten sind domänenneutral und bedürfen daher keiner Anpassung an die Besonderheiten der Softwareentwicklung. Insofern darf Six Sigma in der Softwareentwicklung nicht ausschließlich anhand der Anwendbarkeit der im Vordergrund stehenden statistischen Methoden bewertet werden.

2.2 Measure-Phase

Die Möglichkeiten zum Messen sind in der Softwareentwicklung nicht in dem Umfang gegeben, wie dies in der Massenproduktion der Fall ist. Die Datenqualität und -verfügbarkeit kann in der Softwareentwicklung als mangelhaft bezeichnet werden [Ka03, S. 470]. Oftmals ist kein direkter Zugang zur Messquelle gegeben, so dass auf Indikatoren, welche lediglich mit der Messquelle korrelieren, ausgewichen werden muss. Dadurch, dass in der Softwareentwicklung nicht nur Fehlerentstehung, sondern auch die Fehlerbeseitigung in allen Phasen stattfindet, kann zudem das Konzept der *rty* und der anderen sinngemäßen Metriken nicht direkt übernommen werden. Mit der *defect removal effectiveness* (*dre*) existiert aber bereits ein analoges Konzept in der Softwareentwicklung, das Aussagen über die Effizienz der Fehlerbeseitigung macht und dabei softwarespezifische Besonderheiten wie u. a. die neuerliche Fehlereinführung bei der Berichtigung berücksichtigt (vgl. Abbildung 2).

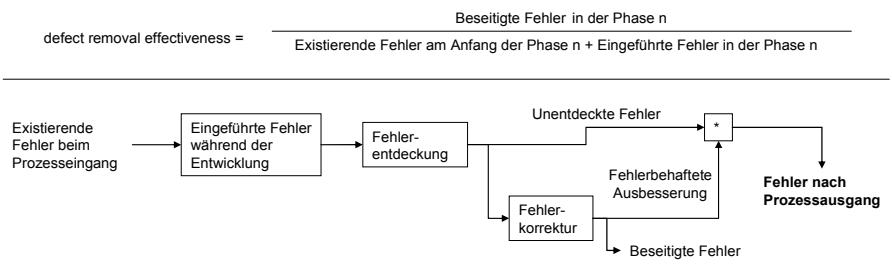


Abbildung 2: defect removal effectiveness [Ka03, S. 164f.]

Die *dre* ist ein ex post-Modell, da die eingeführten Fehler in der Phase *n* zum dortigen Zeitpunkt noch nicht bekannt sein können. Die Operationalisierung des Ansatzes erfolgt mit einer Matrix (Tool-Unterstützung ist zumeist Excel-basiert), in der Fehler der entsprechenden Phase, in der sie gefunden wurden, sowie der Phase, in der sie ihren Ursprung haben, zugeordnet werden. Basierend auf der Matrix und der Definition der *dre* können Effektivitätsmetriken für die einzelnen Phasen der Softwareentwicklung kalkuliert, über mehrere oder alle Phasen zu aggregierten Effektivitätsmetriken verdichtet [Ka03, S. 165ff.], und somit als Datenbasis für Six Sigma-Projekte herangezogen werden.

Einen solchen Ansatz verwirklicht in der Praxis Hallowell [Ha07]. Hier werden, in Analogie zu einem Reifegradmodell, aufeinander aufbauenden Six Sigma-Zielen die erforderlichen und ermöglichten Prozesse sowie, ausgehend vom zu *dre* analogen Konzept der *defect containment effectiveness,* die Implementierung der entsprechenden Metriken zugeordnet (vgl. Abbildung 3).

Six Sigma-Ziel	Erforderliche Prozesse	Ermöglichte Prozesse	Metriken
1. Reduzierung der Defekte (Released); Fokus auf Fehlerkorrektur	Modul-, Integrations-, Systemtest; Fehlerklassifizierung und Fehlerdatenbank-basierte Fehlerlösung	Auswertung von Fehlern vor vs. nach Release; Kausalanalyse	*Total Containment Effectiveness (TCE);* Fehler pro Fehlerklasse
2. Fehlerbeseitigung unmittelbar beim Ursprung	Software-Inspektionen	Analyse der Fehlereinführung und der Effizienz der Fehlerbeseitigung pro Phase	*Phase- und Defect Containment Effectiveness (PCE und DCE),* Fehler pro Modul pro Phase
3. Vorhersage der Fehlereinführung und der Effizienz der Fehlerbeseitigung vor und nach Release	Anwendung von Zuverlässigkeitsmodellen (bspw. Rayleigh)	Ermittlung des Inspektions- und Testaufwands; Schätzungen der Fehlerdichte des ausgelieferten Produktes	*Best fit Rayleigh Model* *Predictive Rayleigh Model*

Abbildung 3: Six Sigma als Metriken-Initiative basierend auf der *defect containment effectiveness* [Ha07]

2.3 Analyze-Phase

Die Prozesse der Softwareentwicklung sind unbeständig und teilweise undefiniert. Auch bei Verwendung eines Vorgehensmodells kann eine leichte Variation der kognitiven Prozesse zu unvorhersehbaren Ergebnissen führen. Besonders wenn die Leistung von Menschen unmittelbar über die Mitarbeiterqualifikation oder mittelbar über die Unternehmenskultur in den Prozess einfließt, müssen grundlegende Schwankungen als gegeben hingenommen werden. Eine zweite, wesentliche Quelle der Prozessvariation sind Produkteinflüsse. Produktcharakteristika wie bspw. Produktkomplexität sowie Produktanforderungen wie bspw. die geforderte Zuverlässigkeit haben ebenso signifikanten Einfluss auf die Prozessproduktivität und -ergebnisse. In der Summe ist die Variation des Softwareentwicklungsprozesses enorm hoch. Nach empirischen Erkenntnissen variiert für ein typisches Entwicklungsprojekt in der Größenordnung von 80.000 LOC (Lines of Code) die Entwicklungszeit zwischen 3 und 100 Monaten, der Aufwand zwischen 8 und 2000 Mann-Monaten und die Zahl der gefundenen Fehler zwischen 10 und 4.000 [PM97, S. 30 und 35]. Für die Anwendung von Six Sigma in der Softwareentwicklung ergeben sich hieraus Konsequenzen und ein Bedarf zur Anpassung der Methodik.

Da statistische Verfahren bestimmte Anforderungen an die Qualität von Auswertungsdaten stellen, steht bei einem Six Sigma-Projekt in der Softwareentwicklung nicht das vollständige statistische Methodeninstrumentarium der *Analyze*-Phase zur Verfügung. Bspw. ist die bei Six Sigma sehr wichtige Methode *DoE* (*Design of Experiments*) in der Softwareentwicklung nur bedingt anwendbar. Denn ein exaktes Messen wie bei physischen Produkteigenschaften in der Massenproduktion und anschließendes Optimieren über eine exakt bestimmbare Faktoreinsatzkombination ist hier nicht realistisch.

Auf der Methodenebene sprechen Magnusson et al. [MKB04] jedoch von insgesamt über 450 potentiellen Six Sigma-Methoden, wobei zu beachten ist, dass diese für sich genommen nicht Six Sigma-spezifisch sind. Die Herausforderung liegt daher in der Verwendung der für die Domäne der Softwareentwicklung und ihrer Besonderheiten zielführenden Methoden. Aufgrund der enormen Variation und der dadurch teilweise fehlenden direkten Korrelation zwischen Prozessinput (Eingangsvariablen) und -output (Produktqualität bzw. Prozessproduktivität) [Ka03, S.143] ist der mögliche Erkenntnisgewinn durch statistische Analysen der Variation in der Softwareentwicklung begrenzt. Sie führen nicht unmittelbar zum Verständnis des Problems und der Ursache-Wirkungszusammenhänge. Daher muss der Fokus bei der Auswahl der richtigen Methoden in der Softwareentwicklung auch auf qualitative Beobachtungs- und Analysetechniken, wie bspw. Post Mortem-Analysen abgeschlossener Projekte, erweitert werden.

2.4 Improve-Phase

Der kognitive Charakter des Softwareentwicklungsprozesses und seine enorme Variation führen in der *Improve*-Phase zu Konsequenzen analog zur *Analyze*-Phase. Denn nicht nur die Problemursachen, sondern auch die Effekte der Prozessverbesserung, werden durch die Produkt- und Mitarbeitereinflüsse überdeckt. So ist bspw. eine moderate Erhöhung der Testeffizienz durch den Einsatz von CASE-Tools nur schwer nachweisbar, da überlagernde Produktivitätsschwankungen durch die Mitarbeiterqualifikation den Faktor 10 erreichen können. Die Effekte der Prozessverbesserung müssen daher normiert werden. Die anforderungsneutrale Definition der Prozesseffizienz durch den Aufwand pro *Function Point* [BF04, S. 78] ist hier ein probates Mittel, um Prozessverbesserungen in der Softwareentwicklung losgelöst von der Produktkomplexität messbar zu machen.

Six Sigma bietet grundsätzlich keine konkreten Lösungsvorschläge oder Ansatzpunkte für die Prozessverbesserung, sondern versteht sich als ein Ansatz, der allgemeine Problemlösungskompetenz zur Lösung sehr verschiedener inhaltsspezifischer Probleme bietet. Verbesserungen resultieren daher neben dem Optimieren der Eingangsvariabeln auf den Prozess i. e. S. (als ein mathematisches bzw. statistisches Problem) mit Hilfe von *DoE* oder durch Änderungen von Prozesstoleranzen, auch aus der Anwendung von Kreativitätstechniken. Während das Optimieren bei kognitiven Prozessen ausscheidet, ist die Anwendung der von Six Sigma bereitgestellten Kreativitätstechniken domänenneutral.

Six Sigma-Projekte versuchen die Probleme aus möglichst vielen Perspektiven zu betrachten, um sie optimal erfassen zu können. Hierfür kann das Projektteam temporär in der *Improve*-Phase um Spezialisten aus dem unmittelbaren Prozessumfeld erweitert werden. Die Lösungsgenerierung erfolgt dabei systematisch mit Hilfe von Kreativitätstechniken wie bspw. Brainstorming. Kreativitätstechniken wenden heuristische Prinzipien in formalisierter Form an. Die Prinzipien Assoziieren und Strukturieren fördern das intuitive Hervorbringen von Ideen, während Variieren, Kombinieren und Abstrahieren durch systematisch analytisches Vorgehen zu neuen Ansätzen führen [Ba98, S. 172f.].

Die *Improve*-Phase von Six Sigma in der Softwareentwicklung ist nicht scharf vom sog. *Design for Six Sigma*-Ansatz (*DFSS*) abzugrenzen. Als *DFSS*(-Prozesse) werden generische Produktentwicklungsprozesse bezeichnet, die den Anspruch haben, kundenorientierte, innovative und fehlerfreie Produkte zu entwickeln [YE03]. Dabei spielt die Methode *Quality Function Deployment* (*QFD*) [HSM00] eine wesentliche Rolle. Es sind Entwicklungsprozesse im Sinne der Philosophie von Six Sigma. Gegenstand von Verbesserungen bei Six Sigma in der Softwareentwicklung ist überwiegend die Einführung von ebensolchen Verbesserungen, die der Philosophie von Six Sigma entsprechen. In der Praxis sind dies:

- Konzepte, die zur datenbasierten Entscheidungsfindung im Entwicklungsprozess führen: Aufwandschätzung, Risikomanagement, RoI, etc. [DC02; Da92];
- Konzepte, die zu weniger Fehlern führen: Inspektionen, Zuverlässigkeitsmodelle, *dre*-Metriken (vgl. Abbildung 3) [Ha07];
- Konzepte, die zur Erhöhung der Kundenzufriedenheit führen: QFD, Kosten-Nutzen-basierte Anforderungspriorisierung [Fe05; DC03; HE00].

2.5 Control-Phase

Ziel der Statistischen Prozessregelung ist es, aus Vergangenheitsdaten die zukünftige Variabilität des Prozesses vorherzusagen, um so präventiv eingreifen zu können, bevor der Prozess mit hoher Wahrscheinlichkeit Ausschuss produzieren würde. Basis dafür bildet die Qualitätsregelkartentechnik, mit der die Stichprobenentnahme im Zeitverlauf visualisiert wird und damit rechtzeitig Unregelmäßigkeiten in den Messdatentrends erkannt werden können [Zo01, S. 1101ff.]. Es wird also im Prozessschritt *n* von der Qualität der letzten baugleichen *x-1*, *x-2*, etc. Zwischenprodukte auf die Qualität des Zwischenproduktes *x* geschlossen. Häuften sich bspw. systematisch nach jedem 1000. Werkstück die Fehler im Prozessschritt *n*, ist es plausibel anzunehmen, dass sich dieses Muster fortsetzen wird. Dabei spielt es weder eine Rolle, welche Prozessschritte *n-1*, *n-2* etc. durchlaufen wurden, noch wie dies der Fall war.

Da in der Softwareentwicklung diese Schlussfolgerung v. a. aufgrund der Heterogenität der Artefakte (bspw. Produktkomplexität) und der Nichtbeachtung des bisherigen Entwicklungsverlaufes bez. Prozessqualität nicht zwingend sinnvoll ist, bietet es sich an, das Konzept wiederum auf Analogien zu vorhandenen Ansätzen der Softwareentwicklung zu untersuchen. Statistische Prozessregelung wird zwar in der dargestellten Form in der Softwareentwicklung thematisiert. Doch sie wurde für die Massenproduktion entwickelt und basiert damit auf dortigen Gesetzesmäßigkeiten, die in der Softwareentwicklung so nicht gegeben sind. Allen voran ist Softwareentwicklung nicht Massenproduktion und nimmt im Gegensatz zu dieser einen relativ langen Zeitraum sowie mehrere Projektphasen in Anspruch.

Daher ist hier konzeptionell ein Lebenszykluskonzept zweckmäßiger als Qualitätsregelkarten, die auf sequentiellen Daten basieren [Ka03, S. 144]. Bei diesem ist die Schlussfolgerung eine in der Softwareentwicklung ungleich sinnvollere, nämlich vom bisherigen Entwicklungsverlauf (bspw. Zeitdruck, Change History, *dre*, etc.) in den Entwicklungsphasen *n-1*, *n-2*, etc. des Moduls *x* auf seine Qualität in der Entwicklungsphase *n*. Wurde bspw. ein komplexes Modul unter hohem Zeitdruck, vielen Änderungsanforderungen, sowie einer hohen Personalfluktuation entwickelt, ist es plausibel anzunehmen, dass in der Phase *n* (bspw. Modultest) viele Fehler gefunden werden.

Solche auf dem Lebenszykluskonzept basierenden Modelle existieren in der Softwareentwicklung und werden als Zuverlässigkeitsmodelle bezeichnet. Six Sigma Ansätze in der Softwareentwicklung adaptieren diese bereits (vgl. Abbildung 3). Die Fehlerdichte in der Phase *n* ist hier eine Funktion der Produkteigenschaften, des bisherigen Projektverlaufs und eben der aktuellen Projektphase.

Zuverlässigkeitsmodelle bedürfen einer zeit- und kostenintensiven Einführung bzw. Justierung im Unternehmen, um verlässliche Vorhersagen treffen zu können. Beispielhaft sei hier das Zuverlässigkeitsmodell von *Rayleigh* genannt. Es kann zur Ermittlung des Inspektions- und Testaufwands bei einem gegebenen Qualitätsziel und zur Schätzungen der Fehlerdichte des ausgelieferten Produktes eingesetzt werden [Ka03, S. 187ff.].

3 Nutzen und Grenzen von Six Sigma in der Softwareentwicklung

Bei der Gestaltung der Softwareprozessverbesserung sind wie bei jedem anderen Gestaltungsvorgang diverse Rahmenbedingungen zu beachten. Sie stellen zum einen Anforderungen an die Gestaltungsmaßnahmen und üben zum anderen Einfluss auf die Wirkung dieser Gestaltungsmaßnahmen aus [Me04, S. 39ff.].

Kap. 2 verdeutlichte, dass aufgrund der besonderen Rahmenbedingungen in der Softwareentwicklung die direkte Übertragbarkeit von Six Sigma aus dem industriellen Kontext nicht bei allen Methoden und Phasen von Six Sigma zielführend ist. Deren Umsetzung muss vielmehr durch die aufgezeigten Analogien und bereits vorhandenen Ansätze in der Softwareentwicklung erfolgen. Grundsätzlich können erst unter dieser Voraussetzung die tatsächlichen Nutzen, Grenzen und Potentiale von Six Sigma in der Softwareentwicklung aufgezeigt werden. Im Einzelfall ist die erfolgreiche Umsetzung und ggf. weitere Anpassung von Six Sigma jedoch von weiteren Rahmenbedingungen abhängig, nämlich von der

- Problemart, vom
- Reifegrad des Softwareentwicklungsprozesses, sowie von der
- Unternehmens- und Problemlösungskultur.

3.1 Problemart als Rahmenbedingung

Die systematische Lösung von Problemen lässt sich in die beiden Phasen Ursache- und Lösungsanalyse aufteilen. Beide Phasen sind vollständig durch das *DMAIC* abgedeckt. Sie müssen jedoch nur bei Problemen i. e. S. durchlaufen werden, d. h. wenn weder die Ursache noch die Lösung des Problems bekannt sind. Ist die Ursache bekannt oder deren Kenntnis nicht notwendig, entfällt die Ursachenanalyse (*Measure* und *Analyze*). Ist zudem gar die Lösung bekannt bzw. vorab beschlossen, kann auch die Lösungsanalyse (*Improve*) entfallen. Denn bei einer offensichtlichen und allen Beteiligten bekannten Ursache erübrigt sich deren Analyse. Analog erübrigt sich eine aufwendige Lösungsanalyse, wenn die Lösung bereits feststeht.

Bei Problemen mit bekannter oder nicht wissenswerter Ursache ist daher ein relativ aufwändiges Six Sigma-Projekt nicht gerechtfertigt. Es bietet es sich daher an, ein lediglich als ein Projektmanagementansatz für Softwareprozessverbesserungen verstandenes Six Sigma anzuwenden. Hierunter fällt v. a. die *Define*-Phase. Somit kann eine zielgerichtete Einführung von vorab festgelegten Verbesserungen in einem projektorganisatorischen Rahmen erfolgen, der den Merkmalen erfolgreicher Prozessverbesserungen in der Softwareentwicklung entspricht (vgl. Kap. 2 zu *Define*).

Das gesamte Potential von Six Sigma kann also nur bei Problemen i. e. S. ausgeschöpft werden. Hier kann Six Sigma in seinem ursprünglichen Sinn als ein systematischer Problemlösungsansatz mit seinem gesamten methodischen Kern, der Problem- und Lösungsanalyse mit statistischen Methoden, Anwendung finden. Voraussetzung für den Erfolg ist jedoch neben der methodischen Anpassung an die Besonderheiten der Softwareentwicklung (vgl. Kapitel 2) eine Problemstellung, die für statistische Methoden zugänglich ist. Bedauerlicherweise gilt dies nur bedingt für die drei bedeutendsten Einflussfaktoren auf die Produktivität des Softwareenwicklungsprozesses, nämlich für die Bereiche Unternehmenskultur, Qualifikation der Mitarbeiter, sowie den Grad der Wiederverwendung. Die Reduzierung von Variation bei diesen qualitätskritischen Steuerparametern ist zwar prinzipiell möglich - bei der Mitarbeiterqualifikation bspw. über eine standardisierte und qualitativ hochwertige Ausbildung. Doch unternehmenskulturellen Problemen, Defiziten in der Mitarbeiterqualifikation sowie mangelnder Wiederverwendung kann nur begrenzt mit statistischen Methoden beigekommen werden.

3.2 Prozessreifegrad der Softwareentwicklung als Rahmenbedingung

Das *CMMI* unterscheidet fünf Reifestufen des Softwareentwicklungsprozesses. Jede Reifestufe zeichnet sich durch bestimmte Prozessfähigkeiten aus, wobei die Stufen geordnet sind und aufeinander aufbauen. Auf Stufe 3 (*Defined*) existiert ein unternehmensweit gültiger Softwareentwicklungsprozesses, der dokumentiert und standardisiert ist. Auf Stufe 4 (*Managed*) sind quantitative Ziele für Produkte und Prozesse vorhanden; die Qualität wird gemessen. Der Einfluss von verwendeten Maßnahmen auf Produktqualität sowie Produktivität ist verstanden, sodass Zielabweichungen mit Hilfe vorhandener Daten erklärt werden können. Auf der *CMMI*-Stufe 5 (*Optimizing*) ist das Unternehmen auf kontinuierliche Prozessverbesserung ausgerichtet. Erfahrungen werden mit quantitativen Daten beschrieben sowie neue Methoden zunächst in Pilotanwendungen erprobt. Anschließend wird ihre Eignung auf Basis quantitativer (Kosten-Nutzen-)Analysen festgestellt [Me04, S. 331f.].

Offensichtlich bestehen zwischen *CMMI* und Six Sigma Berührungspunkte. *CMMI*-Stufe 5 kann durch die Verwendung von Six Sigma umgesetzt werden. *CMMI*-Stufe 4 beschreibt treffend wesentliche Bestandteile der Six Sigma-Philosophie, aber gleichzeitig auch wesentliche implizite Voraussetzungen für die erfolgreiche Anwendung von Six Sigma. Das CMMI als Reifegradmodell geht davon aus, dass eine bestimmte Stufe nicht ohne Prozessfähigkeiten erreicht werden kann (und darf), die den darunter liegenden Stufen zugeordnet werden. Insofern muss auch für die erfolgreiche Anwendung von Six Sigma ein gewisser Reifegrad des Softwareentwicklungsprozesses vorausgesetzt werden. Fehlen bspw. wohl definierte und auf Kundenanforderungen basierte Metriken, kann die Softwareprozessverbesserung mit Six Sigma mit einer aufwendigen Metriken-Initiative einhergehen.

So berichtet Daskalantonakis [Da92] von einem langwierigen Prozess der Schaffung einer Softwaremetrik-Infrastruktur im Zuge der flächendeckenden Anwendung von Six Sigma bei Motorola. Neben der eigentlichen Definition der Metriken, die in Arbeitsgruppen mit Hilfe der *Goal Question Metric*-Methode (*GQM*) abgeleitet wurden, umfasste der Prozess entsprechende Trainingsmaßnahmen, die Einführung von Tools sowie Beraterunterstützung bei der Implementierung der Metriken [Da92, S. 107f.].

3.3 Unternehmens- und Problemlösungskultur als Rahmenbedingung

Eine breite Akzeptanz von Six Sigma im Softwareunternehmen setzt voraus, dass die Softwareentwicklung als eine Ingenieurwissenschaft angesehen wird und nicht als eine vorwiegend künstlerisch-kreative Tätigkeit. Eine Methode in einer Ingenieurwissenschaft wird stets an ihrer Nützlichkeit gemessen, welche sich in den meisten Fällen ausschließlich empirisch nachweisen lässt [Pr01, S. 30]. Six Sigma ist in diesem Punkt rigoros, sodass im Softwareunternehmen vor den Resultaten einer empirischen Bewertung bestehender sowie neu eingeführter Methoden nicht zurückgewichen werden darf. Damit ist Six Sigma gleichzeitig auch wissenschaftlich akzeptabel. Denn sowohl bei Six Sigma als einem Praxisansatz, als auch bei der empirischen Softwaretechnik als einem wissenschaftlichen Ansatz, stützt sich der Wissensgewinn auf die tatsächliche und quantifizierte Wirkung von (Entwicklungs-)Methoden. Im Gegensatz dazu steht die spekulative Bewertung, die auf Basis mehr oder weniger plausibler und größtenteils unausgesprochener Annahmen die erwarteten Eigenschaften ohne Empirie herleitet [Pr01, S. 30]. Insofern steht Six Sigma vor denselben Herausforderungen auf dem Weg zu einer messwertgesteuerten Softwareentwicklung wie die empirische Softwaretechnik.

Auf der anderen Seite können an einer mangelnden Datenverfügbarkeit und -qualität leidende Six Sigma-Projekte die Metrikproblematik im Unternehmen unmissverständlich aufzeigen und sukzessiv aus den Projekten heraus Metriken institutionalisieren. Die rationale Vorgehensweise bei der Entscheidungsfindung in einem Six Sigma-Projekt ist bei allen Entscheidungen im Softwareentwicklungsprozess wünschenswert. Six Sigma trägt zu einer Abkehr von der häufig intuitiven, subjektiven, und im Nachhinein nicht nachvollziehbaren Entscheidungsfindung, zu einem systematischen und rationalen Vorgehen bei der Bewältigung von Problemen und Entscheidungen im gesamten Softwareentwicklungsprozess bei.

4 Zusammenfassung

Um den Rahmenbedingungen in der Softwareentwicklung gerecht zu werden, muss Six Sigma größtenteils durch Analogien und bereits vorhandene Ansätze in der Softwareentwicklung adaptiert werden. So bezeichnet bspw. die SAP ihr derart abgeleitetes Methodenset als SAP Sigma; Sun´s Ansatz lautet Sun Sigma. Resultierende Verbesserungen durch Six Sigma in der Softwareentwicklung liegen u. a. in den Bereichen Aufwandschätzung, Metriken-basierte Entscheidungsfindung, Fehlervermeidung und Erhöhung der Kundenzufriedenheit.

Im Einzelfall ist die erfolgreiche Umsetzung und ggf. weitere Anpassung von Six Sigma von zusätzlichen Rahmenbedingungen abhängig – v. a. muss das ursächliche Problem für statistische Methoden zugänglich sein und es muss bereits ein gewisser Prozessreifegrad mit einer Softwaremetrik-Infrastruktur vorhanden sein. Auf der anderen Seite können an einer mangelnden Datenverfügbarkeit leidende Six Sigma-Projekte die Metrikproblematik aufzeigen und sukzessiv aus den Projekten heraus Metriken institutionalisieren. Insgesamt trägt Six Sigma zu einer Abkehr von der häufig intuitiven, subjektiven, und im Nachhinein nicht nachvollziehbaren Entscheidungsfindung, zu einem systematischen und rationalen Vorgehen bei der Bewältigung von Problemen und Entscheidungen im gesamten Softwareentwicklungsprozess bei.

Literaturverzeichnis

[Ba98] Balzert, H.: Lehrbuch der Software-Technik: Software-Management, Software-Qualitätssicherung, Unternehmensmodellierung, Band 2. Spektrum, Heidelberg 1998.

[Ba04] Basu, R.: Six-sigma to operational excellence: role of tools and techniques. In: International Journal of Six Sigma and Competitive Advantage 1 (2004) 1, S. 44–64.

[BF04] Bundschuh, M.; Fabry, A.: Aufwandschätzung von IT-Projekten. 2. Aufl., mitp-Verlag, Bonn 2004.

[Da92] Daskalantonakis, M.: A Practical View of Software Measurement and Implementation Experiences Within Motorola. In: IEEE Trans. Software Eng. 18 (1992) 11, S. 998-1010.

[DC03] Denne, M.; Cleland-Huang, J.: Software by Numbers. Prentice Hall, Upper Saddle River 2003.

[Fe05] Fehlmann, T.: Six Sigma in der SW-Entwicklung. Vieweg, Wiesbaden 2005.

[FS07] Fröschle, H.-P.; Strahringer, S. (Hrsg.): IT-Industrialisierung, HMD-Praxis der Wirtschaftsinformatik, Heft 256. dpunkt Verlag, Heidelberg 2007.

[HAB02] Hall, T.; Austen, R.; Badoo, N.: Implementing Software Process Improvement: An empirical Study. In: Software Process Improvement and Practice (2002) 7, S. 3-15.

[Ha07] Hallowell, D.: Six Sigma Software Metrics, Part 2, iSixSigma LLC, 2007, abgerufen am 06.04.2007: http://software.isixsigma.com/library/content/c031001a.asp.

[HS00] Harry, M.; Schroeder, R.: Six sigma: the breakthrough management strategy revolutionizing the world's top corporations. Doubleday, New York 2000.

[HSM00] Herzwurm, G.; Schockert, S.; Mellis, W.: Joint requirements engineering - QFD for rapid customer-focused software and Internet-development. Vieweg, Wiesbaden 2000.

[Ka03] Kan, S. H.: Metrics and Models in Software Quality Engineering. 2. Aufl., Addison-Wesley, Upper Saddle River 2003.

[MKB04] Magnusson, K.; Kroslid, D.; Bergman, B.: Six Sigma umsetzen. 2. Aufl., Hanser, München 2004.

[Me04] Mellis, W.: Projektmanagement der SW-Entwicklung. Vieweg, Wiesbaden 2004.

[Pr01] Prechelt, L.: Kontrollierte Experimente in der Softwaretechnik. Springer, Berlin 2001.

[PM97] Putnam, L. H.; Myers, W.: Industrial Strength Software: Effective Management Using Measurement. IEEE Computer Society Press, Washington 1997.

[RY05] Rehbehn, R.; Yurdakul, Z. B.: Mit Six Sigma zu Business Excellence. 2. Aufl., Publics Corporate Publishing, Erlangen 2005.

[Se03] Seghezzi, H. D.: Integriertes Qualitätsmanagement: das St. Galler Konzept. 2. Aufl., Hanser, München 2003.

[YE03] Yang, K.; El-Haik, B.: Design for Six Sigma: a roadmap for product development. McGraw-Hill, New York 2003.

[Zo01] Zollondz, H.-D.: Lexikon Qualitätsmanagement. Oldenbourg, München 2001

Testmanagement unter dem Einfluss der Software-Industrialisierung

Andreas Birk

Software.Process.Management
Gutenbergstraße 99
70197 Stuttgart
andreas.birk@swpm.de

Abstract: Die Software-Entwicklung unterliegt seit einigen Jahren einem Trend zur Industrialisierung, der auf mehreren Ebenen abläuft. So berührt er den Aufbau und die Technologien der Software-Systeme ebenso wie die Vorgehensweise bei Software-Entwicklung und IT-Betrieb. Das Testmanagement ist ein Teilbereich des Software-Managements, das aktuell in besonderem Maße dem Einfluss der Software-Industrialisierung ausgesetzt ist.
Dieser Artikel erfasst die Auswirkungen der Software-Industrialisierung auf das Testmanagement und identifiziert die Herausforderungen, die sich daraus ergeben. Er stellt Maßnahmen vor, mit denen das Testmanagement auf die Herausforderungen reagieren kann. Fallbeispiele veranschaulichen und belegen, wie die Maßnahmen in der Praxis von IT und Software-Entwicklung eingesetzt werden.

1 Einleitung

Industrialisierungsbestrebungen treiben schon seit Längerem Entwicklungen in der Software- und IT-Branche voran. Versteht man Struktur und Auswirkungen der Software-Industrialisierung, kann man die aktuellen und künftigen Anforderungen an die Branche besser überblicken und die Entwicklungen optimal mitgestalten. Dazu gilt es folgende Fragen zu beantworten: Welche Facetten besitzt die Industrialisierung? Welche Herausforderungen bringt sie für die Software-Praxis? Wie reagieren Software-Entwicklung und -Management darauf?

Das Testmanagement ist den Industrialisierungstendenzen zurzeit besonders stark ausgesetzt. Als überschaubarer Teilbereich des Software-Managements bietet es eine geeignete Arena, um die Auswirkungen der Industrialisierung näher zu analysieren.

Dieser Artikel gibt zunächst einen Überblick über die Facetten der Software-Industrialisierung (Kapitel 2). Danach identifiziert er die Herausforderungen der Industrialisierung für das Testmanagement (Kapitel 3) und zeigt, mit welchen Maßnahmen das Testmanagement darauf reagiert (Kapitel 4). Kapitel 5 illustriert die Maßnahmen mit Fallbeispielen. Die abschließende Diskussion (Kapitel 6) fasst die Beobachtungen zusammen und überträgt sie auf das Software-Management allgemein. Die beiden Unterkapitel dieser Einleitung möchten zunächst das Phänomen der Software-Industrialisierung und seine Relevanz für das Testmanagement näher fassen.

Das zentrale Anliegen des Artikels ist es, die Industrialisierungstendenzen rund um das Testmanagement zusammen zu tragen und die Maßnahmen zu identifizieren, mit denen die Praxis auf die Entwicklungen reagieren kann. Es handelt sich um eine erste explorative Untersuchung auf Grundlage von Literaturstudien und der langjährigen Erfahrungen des Autors in verschiedenen Industriedomänen. Weitergehende Studien sollten künftig die identifizierten Tendenzen weiter belegt und untersuchen sowie zusätzliche Maßnahmen identifizieren und ihre Wirksamkeit belegen.

1.1 Software-Industrialisierung

Industrialisierung bezeichnet den Einsatz von Arbeitsteilung und Spezialisierung sowie maschinellen, automatisierten und rationalisierten Verfahren zur Erstellung und Distribution von Gütern und Dienstleistungen [Me]. Es steht im Gegensatz zu den ursprünglich bestehenden handwerklichen und manuellen Verfahren, aus denen es sich schrittweise entwickelt hat. Die Software- und IT-Branche wendet zwar von Beginn an automatisierte Verfahren zur Datenverarbeitung an. Ihre Anfänge hatten jedoch eher handwerkliche als industrielle Züge. Ihr industrielles Potenzial hat sich erst allmählich und schrittweise entwickelt.

Heute, so scheint es, ist die Software-Industrialisierung so weit fortgeschritten, dass zumindest ihr Potenzial dem Niveau anderer etablierter Industriebereiche entspricht - beispielsweise dem Maschinenbau. Zugleich ist die Branche noch intensiv damit beschäftigt, dieses Potenzial auf breiter Front und nachhaltig zu erschließen. Von den Erfolgen zeugen beeindruckend komplexe und leistungsfähige Systeme wie die Verfahren zur Zahlungsabwicklung im Bankenbereich oder die Software-Steuerung der großen Telekommunikationsnetze. Die nach wie vor verbreiteten Herausforderungen und Schwierigkeiten werden klar, wenn man sich die (Miss-)Erfolgsraten von Software-Projekten anschaut oder Nachrichten verfolgt, wie kürzlich über das anfängliche Scheitern der Gepäckverteilung im neuen Terminal 5 des Flughafens London-Heathrow.

Die Industrialisierung im Software- und IT-Bereich weist einige Besonderheiten auf, die man zunächst kennen muss, um ihr Potenzial und ihre Herausforderungen besser zu verstehen. Der grundlegende Unterschied zur herkömmlichen Industrialisierung ist, dass in der Software-Industrie Produktion und Distribution eine untergeordnete Rolle spielen. Im Kern ist die Software-*Herstellung* ein kreativer Problemlösungs- und Entwicklungsprozess. Somit können herkömmliche Verfahren der Industrialisierung von Produktion und Distribution nur eingeschränkt auf Software und IT übertragen werden. Die Industrialisierung der Branche bewegt sich weitgehend auf neuem Terrain.

Recherchiert man in Literatur und Internet, so findet man recht verschiedene, wenn auch miteinander verwandte Ausprägungen der Software-Industrialisierung bzw. ähnlicher Begriffe und Konzepte:

- Software-Industrialisierung [BBT05];
- IT-Industrialisierung, insbesondere Industrialisierung des IT-Betriebs [FS07];
- Prozessverbesserung in der Software-Entwicklung und im IT-Betrieb (z. B. CMMI, ITIL V3 und CobIT) [OCG];
- Produkttechnologien auf industriellem Niveau (z. B. hochintegrierte ERP-Systeme sowie Service-orientierte Architektur, SOA);
- Produktlinien-Entwicklung und Software-Produktmanagement [CN02]; [HSH06];
- Komponentenbasierte Verfahren und Software Factory [Cu91]; [GS04].

Es gibt also Ansätze, die sich auf Software-Entwicklung konzentrieren, während andere den IT-Betrieb betrachten. Teils geht es um Produkttechnologie, teils um Prozessaspekte. Manche Verfahren widmen sich technischen, implementierungsnahen Fragen. Andere adressieren vorwiegend strategische Management-Aspekte.

1.2 Software-Management und Testmanagement

Testmanagement ist ein Teilbereich des Software-Managements. Es umfasst die Planung und Steuerung der Testaktivitäten [Sp06]. Damit hat es besonderen Einfluss auf die Qualität des erstellten Produktes sowie auf die Einhaltung der Zeit- und Kostenvorgaben eines Projektes. Komplexe Testvorhaben besitzen heute häufig den Charakter eines Teilprojektes. Somit decken sich die Aufgaben des Testmanagements stark mit denen des allgemeinen Software-Managements. Alle neun Knowledge-Areas des PMI [PMI] finden eine Entsprechung in Aufgaben des Testmanagements.

Dennoch ist das Testmanagement spezifischer und einfacher zu fassen als das allgemeine Software-Management. Dies macht das Testmanagement zu einem geeigneten Kandidaten, um die Auswirkungen der Software-Industrialisierung näher zu untersuchen. Anschließend können die Erkenntnisse gut auf das Software-Management allgemein übertragen werden (siehe Kapitel 6).

2 Trends in der Software-Industrialisierung

Die Einleitung hat aufgezeigt, dass sich hinter dem Begriff *Software-Industrialisierung* recht unterschiedliche Phänomene verbergen. Um die Auswirkungen auf das Testmanagement besser verstehen zu können, fächert dieses Kapitel die Facetten der Software-Industrialisierung auf. Die Trends fallen in zwei große Gruppen:

- Trends bei den erstellten Software-Systemen (Produktaspekte);
- Trends in Software-Entwicklung und -Betrieb (Prozess-/Organisationsaspekte).

Die nachfolgenden Abschnitte untersuchen diese Trends näher. Ausschlaggebend für diese Trends sind die Geschäftsziele und -anforderungen der Software-erstellenden Organisationen. Sie bestimmen beispielsweise die steigenden Produktanforderungen sowie die Erwartungen an die Entwicklungsproduktivität. Die Geschäftsanforderungen geben auch die Richtung für technische Innovationen vor. Die Innovationen schaffen dann neue Anwendungsfelder der Software und induzieren neue Geschäftsanforderungen.

2.1 Trends rund um die Software-Systeme

Dimension der Systeme: Der Scope der Systeme, die man entwickelt und betreibt, hat sich stetig ausgeweitet. In der Anfangszeit der Informatik waren Software-Systeme für einen einzelnen Rechner konzipiert. Später bestand ein System aus vernetzten Teilsystemen. Heute entwickelt man Netze aus vernetzten Teilsystemen. Früher widmete sich der IT-Betrieb dem Netzwerk einer Unternehmensabteilung. Heute geht es mitunter um weltumspannende Anwendungslandschaften.

Größe der Systeme: Innerhalb einer Klasse von Systemen wird die Software stetig umfangreicher. Zum Beispiel der Code eines PC-Betriebssystems ist heute um viele Zehnerpotenzen größer als zu den Anfängen der Personalcomputer.

Fachliche und funktionale Komplexität: Die Systeme sind nicht nur aufgrund besserer Bedienbarkeit und bunterer Bildschirme gewachsen. Fast überall, insbesondere bei Unternehmensapplikationen, wächst auch stetig die Komplexität der darin verborgenen Abläufe und Daten. Kurz: Die Systeme können immer mehr.

Technische Komplexität: Es gibt immer mehr Möglichkeiten, mit denen Systeme entwickelt werden können. So ist die Vielfalt der technischen Konzepte, die einem gewöhnlichen Anwendungssystem zugrunde liegen, überwältigend - beispielsweise die in modernen Programmierframeworks wie Java Enterprise und .NET enthaltenen Mechanismen und vordefinierten Klassen. Die Software-Industrialisierung treibt die Entwicklung in diesem Bereich immer weiter voran.

Qualitätsvorgaben: Die Qualitätsvorgaben an die Systeme, die sogenannten *Nichtfunktionalen Anforderungen*, sind zunehmend anspruchsvoller geworden. Standen in der Anfangszeit der Datenverarbeitung Benutzbarkeits- und Performance-Anforderungen noch hinter den funktionalen Eigenschaften der Systeme zurück, kann ein modernes Internetsystem oder eingebettete Software darauf nicht mehr verzichten. Weitere Beispiele sind Sicherheits- und Wartbarkeitsvorgaben. Manche dieser Anforderungen gehen direkt auf die Software-Industrialisierung zurück (z. B. Wartbarkeit und Portabilität). Andere sind eine Folge der durch die Industrialisierung erreichten weiten Verbreitung von Software-basierten Systemen und den daraus entstandenen Erwartungen der Benutzer und Anwender.

Standardisierung: Produktstandards - zum Beispiel Branchenarchitekturen oder domänenspezifische Modellierungssprachen - sind eine Reaktion auf die überwältigende Konzeptvielfalt. Zugleich erhöhen sie in anderer Hinsicht die Komplexität: Man muss die Standards erlernen, stringent anwenden und ihre Einhaltung überprüfen (siehe unten: Trends rund um System-Erstellung und Betrieb).

Vernetzung und Integration: Vernetzung ist bereits oben als Triebfeder für die immer komplexer werdenden Größendimensionen angesprochen worden. Als grundlegendes Phänomen verdient sie eine Einzelnennung. Sie ist konzeptuell eng verknüpft mit den Bestrebungen zur Integration von Systemen.

2.2 Trends rund um System-Erstellung und -Betrieb

Größe der Entwicklungs- und Betriebsorganisation: Mit der Größe der entwickelten Systeme wächst meist auch die Größe der Organisationen, die sie entwickeln und betreiben. Ein weiterer Grund für das Wachstum der Organisationen ist die Konsolidierung über Unternehmenseinheiten hinweg: Hatten früher viele Bereiche ihre eigene Software-Entwicklung und IT, so gibt es heute nur noch eine große unternehmensweite Zentralabteilung. Waren Entwicklungs- und IT-Abteilungen früher regional oder global verteilt, konzentrieren sich die Aufgaben heute in wenigen großen Zentren, z. B. am Unternehmenssitz oder in Ländern mit besonders günstiger Kostenstruktur.

Komplexität der Entwicklungs- und Betriebsaufgaben: Die Aufgaben, die Entwicklung und Betrieb zu erfüllen haben, werden immer komplexer. Ein Grund dafür ist die steigende technische Komplexität der Produkte.

Produktbezogene Entwicklungstechnologien: Es entstehen ständig neue und immer komplexer werdende Konzepte zur Systementwicklung, die mit dem Aufbau der entwickelten Produkte verbunden sind, und mit denen sich die Software-Entwicklung auseinander setzen muss. Klassische Beispiele sind komponentenbasierte Verfahren und objektorientierte Entwicklung. Aktuelle Beispiele sind modellgetriebene Entwicklung und Aspekt-orientierte Programmierung.

Prozessbezogene Entwicklungstechnologien: Viele technologische Konzepte spielen sich auf der Prozessebene ab. Sie streben an, die Vorgehensweisen bei der Entwicklung zu verbesser und ihr Industrialisierungsniveau zu erhöhen. Beispiele sind Agile Methoden und Produktlinien-Entwicklung.

Produktstandardisierung: Produktstandardisierung wirkt sich, wie oben bereits erwähnt, auch auf die Vorgehensweise aus. Manchmal fordern sie bestimmte Vorgehensweisen mit dem Ziel, spezielle Produkteigenschaften zu gewährleisten. In jedem Fall muss der Prozess Sorge dafür tragen, dass ein Produktstandard in der Entwicklung berücksichtigt und umgesetzt worden ist.

Prozessstandardisierung: Viele Prozessstandards zielen explizit auf die Industrialisierung von Software-Entwicklung und IT-Betrieb ab. Beispiele sind CMMI und ITIL V3.

Zeit- und Kostenvorgaben, Produktivität: Eine wesentliche Triebfeder der Software-Industrialisierung ist die Einhaltung von Zeit- und Kostenvorgaben, die immer enger werden und auf kontinuierliche Produktivitätssteigerung ausgerichtet sind. Dies kann oft nur durch Maßnahmen im Bereich des Vorgehens und der Organisation von Software-Entwicklung und IT-Betrieb erreicht werden.

Nachweisführung und „Compliance“: Vorgaben wie ISO 9000 in der Entwicklung von Gütern, FDA-Vorgaben im Gesundheitswesen oder Sarbanes-Oxley im Finanzbereich fordern, dass die betriebsunterstützenden oder leistungserbringenden IT-Systeme spezielle Nachweise führen. Sie schreiben Verfahren vor, mit denen die Einhaltung der Vorgaben überwacht werden („Compliance“). Dies sind stark ausgeformte Verfahren zur Qualitätssicherung im Zuge der Software-Industrialisierung, die sich stark auf die Prozesse in Software-Entwicklung und Betrieb auswirken können.

Verteilte Entwicklung und Outsourcing: Im Rahmen der Kostenreduktion und der (globalen) Arbeitsteilung fassen verteilte und ausgelagerte Leistungserbringung in Software-Entwicklung und IT-Betrieb immer stärker Fuß.

3 Herausforderungen für das Testmanagement

Wie wirken sich die Industrialisierungsentwicklungen speziell auf das Testmanagement aus? Welche Herausforderungen bringen sie mit sich? Dieses Kapitel listet diese Bereiche und Fragestellungen im Testmanagement auf. Grundsätzlich wirken die Industrialisierungstrends in zweierlei Hinsicht:

- Auswirkungen auf die zu testenden Systeme sowie auf das Entwicklungs- und Betriebsumfeld des Testens; dadurch wirken sie sich indirekt auf das Testen aus.
- Auswirkungen auf die Testtechniken und Testmanagement-Aktivitäten direkt.

Zur ersten Gruppe zählen vorwiegend die Steigerungen von Größe und Komplexität der Systeme sowie der Entwicklungs- und Betriebs-Organisationen, technologische Entwicklungen (Produkttechnologien), Standardisierungsentwicklungen und Compliance-Bestrebungen.

Die folgenden Unterkapitel beschreiben die verschiedenen Auswirkungen und Herausforderungen für das Testmanagement.

Testdokumente und -informationen: Ein Kernbereich des systematischen Testens ist die explizite Definition von Testkonzepten, Teststrategien und Testfällen, also wichtiger Testdokumente und -informationen. Im Zuge der Software-Industrialisierung steigt die Komplexität der zu testenden Produkte, so dass beim Testen eine Vielzahl von Aspekten berücksichtigt werden müssen, die häufig untereinander interagieren. Dies führt zu besonderen Herausforderungen für:

- Größe und Umfang von Teststrategien, Testfall-Repositories und Testplänen;
- Aufwand zur Testvorbereitung;
- Strukturierung von Testfall-Repository;
- Verwaltung von Testdaten.

Integration von Testarten und Teststufen: Im Rahmen der Komplexitätssteigerung von Software-Systemen steigt der Bedarf an unterschiedlichen Testarten. Traditionell waren funktionale Tests sowie Last- und Performance-Tests erforderlich. Für interaktive Benutzerschnittstellen benötigt man heute oft auch Usability-Tests. Web-Anwendungen wecken den Bedarf für das Security-Testing von Web-Applikationen.

Auch die Integration der Teststufen (typischerweise Modultest, Integrationstest, Systemtest und Abnahmetest) schreitet voran. Sie wird getrieben von Bestrebungen zur Steigerung der Testproduktivität. Das Ziel ist, redundante Tätigkeiten zu vermeiden und Synergien zwischen den Teststufen zu nutzen. Ein Beispiel hierfür ist die Verwendung einer einheitlichen Testfallbasis für System- und Abnahmetests, sowie das teilweise Ersetzen von Abnahmetests durch den Review von Systemtest-Ergebnissen.

Die wesentlichen Herausforderungen bestehen hier darin, die Testvorgehensweise, die Testkompetenzen im Team und die Testinfrastruktur so auszugestalten, dass die angestrebte Integration ohne Einbußen der Testeffektivität gelingt.

Testen mehrstufiger Architekturen: Produktlinien-Entwicklung und andere Verfahren der Komponenten- oder Plattform-basierten Entwicklung erfordern meist ein mindestens zweistufiges Testverfahren. Zunächst muss die Plattform getestet werden, später die auf der Plattform aufsetzenden oder von ihr abgeleiteten Systemvarianten. Beide Teststufen besitzen inhaltliche Bezüge (z. B. teils die gleichen Testfälle) während redundante Tests vermieden werden sollen. Ferner ist es schwierig, für die Plattform relevante Testfälle zu identifizieren, da die Einsatzszenarien - und somit die konkreten Testfälle - erst für die abgeleiteten Produktvarianten konkret bestimmt werden können.

Testen bei verteilter Entwicklung: Der Trend zur Arbeitsteilung und Auslagerung von Aufgaben macht auch vor dem Testen nicht halt. Er fordert das Testmanagement insbesondere in dreierlei Hinsicht: (1) Testen und Abnahme von Zulieferungen, (2) Outsourcing des Testens und (3) Koordination verteilter Testaktivitäten. Alle diese Bereiche sind aufwändiger und anspruchsvoller umzusetzen als die traditionelle Entwicklung als Teil eines Entwicklungsprojektes, das zusammenhängend und an einem Ort durchgeführt wird.

Testen bei Agiler Entwicklung: Agile Software-Entwicklung weist dem Testen eine besonders zentrale Rolle in der Software-Entwicklung zu (z. B. Erstellung von Testfällen vor der Entwicklung, Testfälle als teilweiser Ersatz von Anforderungen). Dadurch sind neue Testtechnologien entwickelt worden, insbesondere für den Unit-Test. Eine besondere Herausforderung bietet die Einbindung komplexer System- und Abnahmetests in die kurzen Entwicklungszyklen der Agilen Entwicklung. Wenn diese Tests umfangreiche Vorbereitungen erfordern, können sie nur schwierig mit dem hohen Änderungstempo der Iterationen in Einklang gehalten werden.

Testcenter: Wenn speziell das Testen ausgelagert wird und an einem anderen Ort oder in einer anderen Organisation erfolgt als die Entwicklung, hat man einen Spezialfall des verteilten Testens vor sich, ein sogenanntes *Testcenter*. Anders als beim Testen bei verteilter Entwicklung (siehe oben) erwartet man von dem Ergebnis des Testcenters nicht, dass man es noch näher prüfen muss. Vielmehr liefert das Testcenter die (weitgehend) verbindliche Qualitätsaussage über das getestete System.

Testkoordination: Mit der Größe der Testvorhaben und der Heterogenität der Testorganisation steigt auch er Koordinationsbedarf. In großen verteilten Projekten, mitunter unter Einbeziehung verschiedener Auftragnehmer und Zulieferer, sind die folgenden Koordinationsaspekte besonders anspruchsvoll: (1) Statusermittlung und Statusdarstellung, (2) Aktivitätsplanung, (3) Aufwandsschätzung sowie (4) Auswahl und Implementierung von Metriken für das Monitoring.

Aufbau und Unterhalt komplexer Testinfrastrukturen: Mit der Komplexität der Systeme steigt auch die Komplexität der erforderlichen Testinfrastrukturen. Beispielsweise erfordern vernetzte oder eventgesteuerte Systeme häufig spezielle Simulatoren als Teil der Testumgebung. Ansonsten können zumindest einzelne Teilsysteme vor Fertigstellung des Gesamtsystems nicht sinnvoll getestet werden. Die Herausforderungen bestehen insbesondere darin, dass die Erstellung geeigneter Testinfrastrukturen rares Expertenwissen erfordern und sehr aufwändig sein kann.

Etablierung, Konsolidierung und Unterhalt von Testmanagement-Werkzeugen: Software-Industrialisierung fordert Automatisierung und Werkzeugunterstützung. Aber viele Werkzeug-Lösungen für das Testmanagement weisen noch Lücken auf. Es ist oft auch schwierig, Tools, die sich inhaltlich ergänzen, technisch wirklich gut zu integrieren. Dadurch entsteht die Herausforderung, einigen Aufwand speziell in die Erstellung und Pflege der Testmanagement-Werkzeuge zu stecken. Ein Teilbereich davon ist die Integration von Defect-, Change- und Requirements-Management.

Einsatz von Testautomatisierung: In engem Zusammenhang mit den Testinfrastrukturen stehen die Herausforderungen für die Testautomatisierung - so sind Simulatoren eine spezielle Form der Testautomatisierung. Außerdem ruft der Bedarf für effizientere Tests nach einem Ausbau der Testautomatisierung. Zugleich erschweren manche neuen Produkttechnologien die Testautomatisierung. Ein Beispiel ist das automatisierte Testen von Web-Oberflächen, das durch generative Verfahren zur dynamischen Erstellung von Webseiten erschwert wird, da sich zwischen Testfallaufzeichnung und -ausführung allein durch Auswirkungen der Generierung die Seiten unterscheiden können und die Tests fehl schlagen, auch wenn sich am eigentlichen Inhalt der Seiten nichts geändert hat.

Aufbau und Pflege von Kompetenzbereichen: Viele der genannten Herausforderungen bringen eine weitere Herausforderung mit sich: Man benötigt kompetente Personen, die die anstehenden Aufgaben erfüllen und die nötige Qualifikation aufweisen. Zunächst müssen die erforderlichen Kompetenzen überhaupt in ausreichendem Maß vorhanden sein. Dann müssen sie auch effektiv miteinander integriert werden. Denn komplexe Aufgaben rufen nach dem gemeinsamen, gut abgestimmten Vorgehen mehrerer Personen. Schließlich muss das Testen attraktive Karrierepfade und Entwicklungsmöglichkeiten für hochqualifizierte Mitarbeiter bieten.

Prozessintegration über das Testen hinaus: Das Testen steht in Wechselwirkung mit vielen anderen Aufgaben der Software-Entwicklung und des IT-Betriebs. Exemplarisch seien das Requirements-Management und das Change- und Defect-Management genannt. Mit der Software-Industrialisierung wird es zunehmend wichtiger, die Schnittstellen zwischen diesen Aktivitäten effizient zu gestalten. Neben die Herausforderungen, das Testen selbst gut auszugestalten, tritt die neue Herausforderung, das Testen auch effizient mit den anderen Aufgabenbereichen zu verzahnen.

Effizientes Testen: Im Rahmen der allgemeinen Produktivitätssteigerungen muss auch das Testen seinen Beitrag liefern. Besondere Herausforderungen im Testmanagement sind dabei: (1) Frühzeitiger Beginn der Testvorbereitungen, (2) Priorisierung von Testfällen, (3) Planung des Testens, (4) Prozesssteuerung der Testaktivitäten und (5) Kontinuierliche Verbesserung des Testens.

4 Maßnahmen des Testmanagements

Welche Maßnahmen ergreift das Testmanagement, um auf die Herausforderungen der Software-Industrialisierung zu reagieren? Dieses Kapitel stellt aktuelle methodische und technologische Entwicklungen im Testmanagement vor und setzt sie in Beziehung zur Software-Industrialisierung. Die Fallbeispiele in Kapitel 5 illustrieren die Maßnahmen näher und belegen ihre Relevanz für die Software- und IT-Praxis. Die abschließende Diskussion in Kapitel 6 untersucht, inwiefern die Maßnahmen im Testmanagement beispielhaft sind für vergleichbare Maßnahmen im Software-Management allgemein, um die Herausforderungen der Software-Industrialisierung in den Griff zu bekommen.

Die Maßnahmen des Testmanagements lassen sich in fünf Gruppen einteilen, die im folgenden näher beschrieben sind: (1) Spezialisierte und zentralisierte Organisationseinheiten, (2) Integration und Anreicherung von Aufgabenbereichen, (3) verteilte Organisation und Outsourcing, (4) Ausbau von Prozesssteuerung und Prozessverbesserung sowie (5) umfassende Werkzeugunterstützung.

4.1. Maßnahme 1: Spezialisierte und zentralisierte Organisationseinheiten

Traditionell war das Testen Aufgabe eines jeden einzelnen Entwicklungsprojektes. Vereinzelt wurden bestimmte Testarten auch von Spezialisten der IT durchgeführt, insbesondere Last- und Performance-Tests. Doch dies war eher eine Nebenaufgabe der IT als dass dafür eine eigene Organisationseinheit eingerichtet worden wäre.

Seit einiger Zeit etablieren immer mehr Unternehmen spezialisierte Testcenter, die ihre Dienstleistungen und Fachexpertise rund um das Testen als interne Auftragnehmer anbieten. Das geht darauf zurück, dass zum Einen das Bewusstsein für die Wichtigkeit des Testens für einen durchgängigen und systematischen Entwicklungs- und Betriebsprozess steigt. Zum Anderen lassen unternehmensweit standardisierte Anwendungsarchitekturen und der hohe Vernetzungsgrad der Applikationen gar keine andere Wahl, als auch die Tests zentral zu koordinieren und durchzuführen.

Beides sind Phänomene der Software-Industrialisierung. Beispiele für zentralisierte Organisationseinheiten im Testmanagement sind:

- Testcenter als unternehmensinterne Dienstleister für Testen und Testkoordination, insbesondere bei funktionalen Systemtests und Abnahmetests;
- Spezialisierte Teams für Testautomatisierung und Verwaltung von Regressionstest-Infrastrukturen;
- Spezialisierte Teams für Last- und Performance-Tests als Teile der Qualitätsmanagement- oder IT-Betriebsorganisationen;
- Usability-Beauftragte und -Gruppen, die Usability-Tests als zentrale Dienstleistungen;
- Zentrales Kompetenzzentrum für Sicherheitstests von Web-Applikationen.

4.2 Maßnahme 2: Integration und Anreicherung von Aufgabenbereichen

Ursprünglich wurde Testen in vielen Entwicklungsprojekten stiefmütterlich als lästiges Übel betrachtet, dem man sich gegen Ende eines Projektes mehr oder weniger widmet, je nach dem wie viel Zeit und Budget noch vorhanden ist.

Inzwischen haben viele Entwicklungsorganisationen gelernt, dass systematisches Testen einen hohen Beitrag für den Projekterfolg leistet. Entsprechend müssen Testvorbereitungen früh im Entwicklungsprozess beginnen, das Testen muss mit anderen Entwicklungsaktivitäten eng verzahnt werden (Stichworte: Anforderungsbasiertes Testen, Risikobasiertes Testen, Issue- und Fehlermanagement) und die Inhalte der Testaktivitäten werden umfangreicher (z. B. neue Testarten adressieren wie das Security-Testing von Web-Applikationen). All dies ist eng verknüpft mit dem Trend zur Software-Industrialisierung - insbesondere mit Prozessstandardisierung, gestiegenen Qualitäts- und Sicherheitsanforderungen bzw. Erwartungen der Auftraggeber und Benutzer sowie die höhere Komplexität der Systeme, die zum Beispiel neue Sicherheitslücken zur Folge hat.

4.3 Maßnahme 3: Verteilte Organisation und Outsourcing

Früher wurden Entwicklungsprojekte - somit auch die Tests - vorwiegend von einem Team an einem einzigen Ort durchgeführt. Inzwischen haben die meisten Unternehmen eine differenzierte Organisationsstruktur für Software-Entwicklung, IT-Betrieb und somit auch für das Testen etabliert. Die Gründe können wiederum klar auf die Software-Industrialisierung zurück geführt werden: Aufgabenstandardisierung und Arbeitsteilung sowie Kostenreduzierung.

Die Ausprägungen dieser Aufgabenverteilung im Testen sind:

- Spezialisierte Testdienstleister vor Ort, im Entwicklungsteam oder als separates Testteam;
- Spezialisierte zentrale Testcenter an einem anderen Standort des Unternehmens als dem, in dem das Entwicklungsprojekt angesiedelt ist;
- Testlabs (intern oder extern) in Ländern mit niedrigerer Kostenstruktur (Offshore-Outsourcing).

4.4 Maßnahme 4: Ausbau von Prozesssteuerung und Prozessverbesserung

Testen wird auch vielfach heute noch lediglich als eine Phase der Software-Entwicklung betrachtet und dementsprechend im Rahmen der Gesamtprojektmanagements geplant und gesteuert. Die Spezialisierung (Maßnahme 1), Integration und Anreicherung (Maßnahme 2) und vielmehr noch die verteilte Organisation und das Outsourcing im Testbereich (Maßnahme 3) erfordern aber eine sehr ausgeprägte Prozesssteuerung speziell für das Testen. Deshalb tun Software- und IT-Organisationen gut daran, Testen als ein eigenes (Teil-)Projekt im Rahmen der Software-Entwicklung zu behandeln und mit einer eigenen Management-Struktur auszustatten. Diese muss freilich eng mit dem Gesamtprojektmanagement verzahnt sein.

Da die umfangreichen und mitunter hoch komplexen Testprozesse für viele Organisationen noch neu sind, und teils die methodischen Grundlagen noch nicht genügend ausgereift sind, muss die Testprozesssteuerung um ein besonderes Augenmerk auf die kontinuierliche Prozessverbesserung im Testen ergänzt werden.

Der Ausbau von Prozesssteuerung und Prozessverbesserung im Testen sind indirekte Folgen der Software-Industrialisierung, wobei die Prozessverbesserung selber von dem erhöhten Bedarf nach Prozesssteuerung getragen wird.

Ausprägungen und Belege für diese Entwicklung hin zu stärkerer Prozesssteuerung und Prozessverbesserung sind:

- Testkoordination als ein zentraler Leistungsbaustein im Angebot der unternehmensinternen Testcenter;
- Partner- und Lieferantenmanagement für Testdienstleistungen (bei Outsourcing und/oder Offshore-Leistungen);
- Methoden, Beratungs- und Schulungsangebote für Testqualifikation und Testprozessverbesserung (z. B. TMap Next, TPI, ISTQB Certified Tester).

4.5 Maßnahme 5: Umfassende Werkzeugunterstützung

Nach wie vor sind Textverarbeitungsprogramme und Tabellenkalkulationen die am meisten verbreiteten Software-Werkzeuge zur Unterstützung der Testphase: Textdokumente definieren Testkonzepte, Teststrategien und Testfälle. Tabellenkalkulationen dienen ebenfalls der Testfalldefinition. Außerdem werden sie zur Planung und Verfolgung der Testdurchführung verwendet.

Dennoch ist das Testen derjenige Bereich in der Software-Entwicklung (nach den unabdingbaren Entwicklungsumgebungen für die Implementierung), für den spezialisierte Werkzeugunterstützung am stärksten verbreitet ist: Große IT-Projekte kommen schon lange nicht mehr ohne Testmanagement-Tools aus, Fehlermanagement ebenso wie Load- und Performancetests sind ohne spezielle Tools kaum vorstellbar, und Testautomatisierung für User-Interface-Tests ist weithin etabliert. Bei Prozess- und Methodenfragen wird in kaum einem anderen Bereich so schnell und konkret die Frage nach Werkzeugunterstützung gestellt.

Auch die Anzahl kommerziell und öffentlich verfügbarer Werkzeuge für die unterschiedlichen Teilaufgaben des Testens ist höher als die der Tools für Requirements-Management, Modellierung und Projektmanagement. Ferner zeigen die Konsolidierungsentwicklungen auf dem Werkzeugmarkt - insbesondere der Trend hin zur übergreifenden Unterstützung des Application Lifecycle Managements - die große Bedeutung, die der durchgängigen Toolunterstützung beigemessen wird.

In Hinblick auf den Einfluss der Software-Industrialisierung rührt der Bedarf für eine umfassende Werkzeugunterstützung einerseits direkt her von der erhöhten Komplexität der Produkte und Abläufe, von den Standardisierungsbestrebungen für Produkttechnologien und Abläufe sowie von den Bestrebungen zur Kostenoptimierung und Produktivitätssteigerung.

Andererseits verstärken die anderen beschriebenen Maßnahmen den Bedarf für umfassende Tool-Lösungen: Insbesondere die Integration und Anreicherung von Aufgabenbereichen (Maßnahme 2) schafft klare Anforderungsprofile für den Werkzeugeinsatz. Der Ausbau der Prozesssteuerung (Maßnahme 4) erfordert zentrale Datenhaltung, die Toolgestützte Konsolidierung von Statusinformationen und effiziente Reporting-Funktionen.

5 Fallbeispiele

Dieses Kapitel stellt drei Fallbeispiele für die Testpraxis in Unternehmen vor, die die oben vorgestellten Maßnahmen illustrieren. Der Autor hat die Fallbeispiele im Rahmen seiner Tätigkeiten in der Software-Industrie kennen gelernt. Sie belegen auch die Zusammenhänge zwischen den Herausforderungen der Software-Industrialisierung und den Maßnahmen. Jedes Unterkapitel beschreibt zunächst den jeweiligen Fall und erläutert anschließend die Zusammenhänge zwischen den Herausforderungen und den Maßnahmen.

5.1 Fallbeispiel 1

Ein Finanzdienstleister mit einem starken Anteil von Internet-basiertem Geschäft (u. a. Online-Banking) verfügt über eine durchgängige Applikationslandschaft aus IT-Anwendungen, die stark miteinander interagieren und von der zentralen IT eng aufeinander abgestimmt worden sind. Dies ist insbesondere deshalb erforderlich, um eine hohe Verfügbarkeit der Anwendungen zu gewährleisten - eine wichtige und unternehmenskritische Anforderung an die Systeme. Auftraggeber und Eigner der Anwendungen sind die Fachabteilungen.

Traditionell sind die Fachabteilungen für die Applikationsgestaltung zuständig und somit auch für die Abnahmetests der Anwendungen. Mit der durchgängigen Applikationslandschaft und aufgrund der hohen Verfügbarkeitsanforderungen hat das Unternehmen jedoch die Zuständigkeit für umfangreiche Last- und Performance-Tests bei der zentralen IT-Qualitätsabteilung angesiedelt. Der Erfolg dieser Last- und Performance-Tests ist eine wesentliche Voraussetzung für die Freigabe eines neuen Releases der Applikationslandschaft.

In diesen Last- und Performance-Tests hat sich die IT-Qualitätsabteilung einen sehr guten Ruf erarbeitet und wird im Unternehmen als wichtiger Garant für die Qualität der Applikationen und den darauf aufsetzenden Geschäftsprozessen betrachtet. Aufgrund dieser positiven Erfahrungen hat die IT-Qualitätsabteilung auch die Aufgabe erhalten, die funktionalen Tests der Fachabteilungen zentral zu verwalten und schrittweise zu automatisieren. Sie wird dadurch zum umfassenden Test-Dienstleister im Unternehmen.

Das Beispiel belegt, wie die Industrialisierungstendenzen Technologie- und Architekturstandardisierung sowie hohe Qualitätsanforderungen den Bedarf für fortgeschrittene Testpraktiken verstärken. In der Folge sind die Organisationseinheiten zentralisiert und mit besonderen Aufgaben versehen worden (Maßnahme 1), der Aufgabenbereich der IT-Qualitätsabteilung ist schrittweise ausgeweitet worden (Maßnahme 2) und eine umfassende Werkzeugunterstützung (Maßnahme 3) ist eingeführt worden.

5.2 Fallbeispiel 2

Ein großer Hersteller von Industriegütern mit mehreren weitgehend eigenständigen Unternehmensteilen unterhält eine zentrale IT-Abteilung. Die IT-Abteilung hat eine einheitliche IT-Anwendungslandschaft definiert, deren Kern ein gemeinsames Applikations-Framework ist. Das Applikations-Framework wird in allen Unternehmensteilen eingesetzt, wobei jeder Unternehmensteil spezifische Anpassungen (Änderungen von Funktionen in Teilen des Applikations-Frameworks) und Erweiterungen (zusätzliche Applikationen) besitzt. Für jeden Unternehmensteil betreibt die zentrale IT eine eigene Instanz des Applikations-Frameworks.

Vor der Einführung des zentralen Applikations-Frameworks hat jeder Unternehmensteil seine eigenen IT-Anwendungen definiert, beauftragt und getestet (Abnahmetest). Durch das zentrale, von der IT verwaltete Applikations-Framework haben sich die Anforderungen an das Testen maßgeblich geändert: Die IT muss zentral Systemtests für die übergreifenden Anforderungen aller Unternehmensteile durchführen. Die anschließenden Abnahmetests der Unternehmensteile überlappen sich sowohl untereinander als auch mit den Systemtests der IT. Daher besteht Bedarf, die Tests aufeinander abzustimmen und Möglichkeiten zur Wiederverwendung von Testfällen zu nutzen.

Die IT des Unternehmens hat ein Testcenter eingerichtet, um die Tests zu koordinieren und zu unterstützen. Es tritt auf als interner Dienstleister, der von den Unternehmensteilen (Eigner der Applikationen) beauftragt werden kann, aber nicht einbezogen werden muss.

Zu den wesentlichen Angeboten des Testcenters gehört die Bereitstellung von Testmanagern als Experten zur Koordination der Tests sowie eine zentrale Werkzeugplattform für das Testen (v. a. Testfall-Repository, Testplanung, manuelle Testdurchführung, Fehlermanagement und Status-Reporting). Das Testcenter hat sich innerhalb kurzer Zeit als Testdienstleister etabliert und unterstützt die großen Testvorhaben im Umfeld des zentralen Applikations-Frameworks.

In diesem Fall wirken verschiedene Industrialisierungstendenzen: Komplexe und zugleich standardisierte IT-Technologien und -Architekturen (z. B. komponentenorientierte Produkttechnologie im Applikations-Framework mit Zügen der Produktlinien-Entwicklung), hohe Komplexität der Software-Entwicklung (zweistufige Entwicklung mit Applikations-Framework und Anpassungen) sowie zunehmend engere Zeit- und Kostenvorgaben.

Das Beispiel veranschaulicht gut, wie eine spezialisierte, zentrale Organisationseinheit (Maßnahme 1) das Testen in einer verteilten Organisation (Maßnahme 3) koordiniert. Ein wichtiger Baustein ist der Ausbau der Prozesssteuerung (Maßnahme 4) in Form von Testmanagement-Experten, Fehlermanagement und Status-Reporting. Das Testcenter kann diese Leistungen so nur dank einer fortgeschrittenen Werkzeugunterstützung erbringen (Maßnahme 5).

5.3 Fallbeispiel 3

Ein Industrieunternehmen entwickelt und pflegt die Software-Systeme für seine Fertigungssteuerung unter Einbeziehung mehrere Zulieferer, teils mit Offshore-Anteilen. Die Zulieferer führen Teile der Systemtests für ihre entwickelten Systemteile auf einer zentralen Testumgebung in Eigenregie durch. Der Auftraggeber - das Industrieunternehmen - koordiniert die Tests und überprüft die Testergebnisse.

Um diese verteilten Tests effizient abzuwickeln, hat das Unternehmen die verpflichtenden Testprozesse der Zulieferer detailliert definiert, durch eine einheitliche Werkzeugumgebung unterstützt und die Rolle eines Testkoordinators geschaffen. Die Werkzeugunterstützung umfasst insbesondere das Fehlermanagement. Dazu mussten Werkzeugbrücken zwischen dem System des Auftraggebers und den Lösungen der Zulieferer geschaffen werden.

In diesem Beispiel tritt insbesondere die Industrialisierungstendenz der komplexen Entwicklungsprozesse (samt Organisation) zu Tage. Es belegt das Zusammenwirken von zentraler Testkoordination (Maßnahme 1) und Prozesssteuerung (Maßnahme 4) in einer verteilten Organisation mit Outsourcing (Maßnahme 3).

6 Zusammenfassung und Diskussion

Dieser Artikel stellt die Facetten der Software-Industrialisierung dar und leitet daraus Herausforderungen für das Testmanagement ab. Auf dieser Grundlage identifiziert er fünf Maßnahmen, mit denen das Testmanagement auf die Herausforderungen reagiert. Die Maßnahmen stellen aktuelle Schwerpunkte des Testmanagements dar, für die weitere methodische Unterstützung besonders benötigt wird. Drei Fallbeispiele illustrieren die Maßnahmen und belegen ihre Relevanz.

Die identifizierten Maßnahmen des Testmanagements haben Entsprechungen im Software-Management allgemein: Dort besteht ebenso Bedarf für spezialisierte und zentralisierte Organisationseinheiten wie Project Management Offices, wenn auch die Aufgabendifferenzierung nicht so ausgeprägt ist wie im Testmanagement. Integration und Anreicherung von Aufgabenbereichen findet sich im Software-Management, zum Beispiel bei der Anwendung spezialisierter Projektmanagement-Methoden. Die verteilte Entwicklung wirkt auf das Software-Management insgesamt. Gleiches gilt für Prozesssteuerung und Werkzeugunterstützung.

Die identifizierten Maßnahmen zeigen, wie das Testmanagement auf die Herausforderungen der Software-Industrialisierung reagieren kann. Empirische Studien sollten auf den beschriebenen Faktoren aufsetzen und die Effektivität der Testmanagement-Maßnahmen näher untersuchen. Bislang gibt es hierzu kaum gesicherte Erkenntnisse.

Literaturverzeichnis

[BBT05] Bode, A; Broy, M; Taubner, D;: "Software-Industrialisierung," Informatik-Spektrum, vol. 28, Aug. 2005, S. 269.

[CN02] Clements, P.; Northrop, L. M.: Software Product Lines: Practices and Patterns. Addison Wesley, Upper Saddle River, NJ (2002)

[Cu91] Cusumano, M.A: Japan's Software Factories, Oxford Univ Pr, 1991.

[FS07] Fröschle, H.; Strahringer, S.: IT-Industrialisierung, Dpunkt Verlag, 2007.

[GS04] Greenfield, J.; Short, K.: Software Factories, Wiley & Sons, 2004.

[HSH06] Helferich, A.; Schmid, K.; Herzwurm, G.: "Product management for software product lines: an unsolved problem?," Commun. ACM, vol. 49, 2006, S. 66-67.

[Wi] "IT-Industrialisierung – Wikipedia"; http://de.wikipedia.org/wiki/IT-Industrialisierung.

[Me] Meyers Online Lexikon, http://lexikon.meyers.de/meyers/Industrialisierung.

[OCG] OGC: ITIL V3"; http://www.ogc.gov.uk/index.asp?id=2261.

[PMI] Project Management Institute, A Guide to the Project Management Body of Knowledge: PMBOK Guide, B&T, 2004.

[Sp06] Spillner A. et al.: Praxiswissen Softwaretest - Testmanagement: Aus- und Weiterbildung zum Certified Tester - Advanced Level nach ISTQB-Standard, Dpunkt Verlag, 2006.

Der IT-Produktkompass – Ein Instrument für die Analyse von Geschäftsmodellen und die strategiekonforme Positionierung von IT-Produkten

Wolfram Pietsch

Fachhochschule Aachen
Fachbereich Wirtschaftswissenschaften
Eupener Str. 70
52066 Aachen
pietsch@fh-aachen.de

Abstract: IT-Produkte werden auf unterschiedlichen Ebenen und für unterschiedliche Entwicklungsstufen angeboten und können je nach Aufgabe und Hintergrund verschieden betrachtet werden. Es wird ein Modell vorgestellt, welches die verschiedenen Perspektiven und Stufen integriert und eine zielgerichtete und fokussierte Positionierung von IT-Produkten unterstützt – der IT-Produktkompass. Das Gestaltungsfeld der Positionierung von IT-Produkten wird anhand von drei idealtypischen Geschäftsmodellen – Lizenzgeschäft, Lizenz plus Service und Projektgeschäft analysiert und Handlungsoptionen abgeleitet.

1 Abgrenzung und Positionierung von IT-Produkten

Das Management von IT-Produkten erfordert eine ganzheitliche Behandlung von betriebswirtschaftlichen und technischen Aspekten. Das Produktmanagement ist die Querschnittsfunktion für die Integration der unterschiedlichen Perspektiven und Aufgaben [AH07; Be01]; die verfügbaren Methoden stellen allerdings vornehmlich Konsumgüter und weniger Dienstleistungen in den Mittelpunkt. IT-Produkte beinhalten jedoch komplexe Dienstleitungen und sind selten Massen-Konsumgüter. Das klassische Produktmanagement ist daher nur bedingt anwendbar, es muss auf die spezifischen betriebswirtschaftlichen und technischen Charakteristika von IT-Produkten abgestimmt werden.

Es gibt mehrere Ansätze für ein spezifisches Software-Produktmanagement einerseits aus einer eher betriebswirtschaftlichen Perspektive [CC02; Dv03; LW04] und andererseits aus einer technisch-ingenieurmäßigen Perspektive [PBL05; We06]. Kein Ansatz erweitert die Betrachtung über die Software hinaus und integriert zudem die betriebswirtschaftliche und die technische Perspektive. Die bestehenden Ansätze zeigen, wie komplex die Integration bereits aus der jeweiligen Teilperspektive ist. Ein umfassendes Detailmodell ist daher kaum praktikabel. In der Praxis erfolgt die Komplexitätsreduktion durch die kreative Leistung der verantwortlichen Personen. Folglich ist die Überzeugungsfähigkeit der wesentliche Bestandteil für den Erfolg des Produktmanagements in der Praxis [Be01].

Die Festlegung der Ausrichtung von Produkten für ein bestimmtes Zielmarktsegment, die Produktpositionierung, ist ein zentrales Aufgabenfeld des Produktmanagements [AH07] und wesentlich für die Koordination der betriebswirtschaftlichen und technischen Produktplanung. So betrachten zuweilen Entwicklungsabteilungen jedes freigegebene Entwicklungswerkzeug als Produkt, während im Marketing zwischen Software-Lizenzen, technischen Einrichtungen und ergänzenden Dienstleistungen differenziert wird. Hier entstehen oft Missverständnisse welche die Effektivität und Effizienz der Entwicklung und Vermarktung von Produkten stark beeinträchtigen.

Langjährige Erfahrungen der Autoren mit der Durchführung von Schulung von IT-Produktmanagern haben gezeigt, dass einschlägige Instrumente wie z. B. Portfolio Management, Geschäftsfeldanalyse und Roadmapping einzelne Aspekte des Planungsprozesses unterstützen, jedoch nur bedingt zur Orientierung beitragen. Die Autoren haben deshalb zuerst für die Ausbildung von IT-Produktmanagern ein Hilfsmittel konzipiert und mittlerweile auch in mehreren Fallstudien erprobt, welches den ganzheitlichen Diskurs über die Ausrichtung von IT-Produkten unterstützt.

2 ‚Himmelsrichtungen' für das Management von IT-Produkten

Das vorgestellte Hilfsmittel soll wie ein Kompass die Orientierung der Beteiligten erleichtern, jedoch nicht die Entscheidung über den Weg vorgeben. Die zentrale Bezugsgröße eines Kompass ist die Himmelsrichtung. Es stellt sich die Frage, welche Bezugsgröße für die Positionierung von IT-Produkten geeignet ist. Im Folgenden werden zuerst die ‚Himmelsrichtungen' für die IT-Produktpositionierung erläutert und anschließend der IT-Produktkompass für die Navigation in verschiedenen Geschäftsmodelltypen eingesetzt.

Aus der betriebswirtschaftlichen Perspektive bieten sich die zentralen Elemente des Marketing-Mix, die „4P" (Product, Price, Place, Promotion) [LW04] als Himmelsrichtungen an. Die 4Ps bilden jedoch nur einen Teilaspekt der Managementaufgabe ab und sind nicht unmittelbar auf IT-Produkte anwendbar. Für den IT-Bereich existiert kein Konzept mit vergleichbarer Stringenz und Verbreitung. Relativ bekannt ist das Konzept des linearen Software- bzw. Systemlebenszyklus, der die Entwicklung stufenweise vom Allgemeinen zum Speziellen führt. Auch im Marketing ist das Konzept des Produktlebenszyklus einschlägig [Be01].

Es existieren sehr viele Lebenszyklusmodelle für IT-Systeme. So wurde das klassische Wasserfallmodell [Bo76] nur für Softwaresysteme entwickelt, während der deutsche Entwicklungsstandard „V-Modell XT" [RHH05] Hard- und Softwaresysteme integriert. Teilaspekte des Systemlebenszyklus finden sich in verschiedenen Standards wieder, allerdings weder annähernd vollständig noch konsistent. Dies liegt unter anderem an der hohen Komplexität und starken Ausdifferenzierung der Geschäftsmodelle im IT-Bereich.

So ist zum Beispiel die Entwicklung eines Navigationssystems nicht mit der Einführung einer CRM-Software zu vergleichen. Falls es überhaupt möglich wäre, ein umfassendes Referenzmodell zu erstellen, so wäre dies sicher sehr komplex. Gesucht wird ein einfacheres Modell, das dennoch der Orientierung dient. Als Ausgangpunkt eignet sich der klassische Lebenszyklus. Um das gesamte Spektrum von IT-Produkten abdecken zu können, wurde dieser erweitert mit anderen Konzepten kombiniert. Systemlebenszyklusmodelle gehen i. d. R. davon aus, dass

a) der Lebenszyklus überwiegend bis zum Ende der Entwicklung betrachtet wird: Einführung und Betrieb von IT-Systemen sind wichtige eigenständige Geschäftsfelder. Sie begründen ebenfalls IT-Produkte und müssen deshalb berücksichtigt werden.

b) für jedes IT-System alle Phasen durchlaufen werden: Im IT-Bereich können auch Ergebnisse aus Teilzyklen als Produkte verkauft werden, z. B. die Konzeptentwicklung oder die Auslagerung der Programmierung von Systemen im Offshore-Bereich. Die Phasen müssen also für IT-Produkte nicht zwangsläufig vollständig durchlaufen werden; die Phasen sind nicht als strenge Sequenz, sondern vielmehr als Komponenten zu sehen.

c) ein IT-System typischerweise „auf der grünen Wiese“ entwickelt wird: Kaum ein IT-System wird vollständig neu entwickelt. Es werden vielfältige Soft und Hardware-Komponenten integriert und spätestens im Betrieb muss es mit anderen IT-Produkten zusammengefügt werden. Genau das führt zum nächsten Punkt.

d) die Lebenszyklen unterschiedlicher IT-Systeme unabhängig voneinander betrachtet werden können: IT-Produkte und die betreffenden Lebenszyklen sind stark miteinander verwoben, z. B. erfordert eine neue Betriebssystemversion Anpassungen in vielen anderen darauf aufbauenden Programmen, z. B. Gerätetreibern.

e) eine gemeinsame herstellerübergreifende Entwicklung (oder auch abteilungsübergreifende Entwicklung) nicht besonders betrachtet werden muss: Ein Beispiel ist die Entwicklung der Grafikkarten und der auf ihnen enthaltenen Chipsätze. Dabei arbeiten verschiedene Hersteller zusammen, die aus ihrer Sicht ein eigenes IT-Produkt entwickeln. Aus Sicht der Anwender handelt es sich dabei um eine Komponente eines IT-Produktes. Die Entwicklung kann dabei durch beide Hersteller vorangetrieben werden, jedoch sind beide in hohem Maße vom jeweils anderen IT-Produkt abhängig.

Für IT-Produkte sind also unterschiedliche Phasenabfolgen und –kombinationen möglich. Für die Positionierung von IT-Produkten kann der Systemlebenszyklus nicht im Sinne eines Vorgehensmodells (normativ), sondern als eine Gliederung möglicher Perspektiven und Aufgaben (deskriptiv) verstanden werden. Dies entspricht der Interpretation des allgemeinen Produkt-Lebenszyklus im Marketing [AH07; Be01].

In IT-Unternehmen existieren häufig verschiedene Sichtweisen auf die IT. So sind Programmierer zumeist auf den Kontext der Entwicklung, die Fachabteilungsleiter auf den Einsatz, Endbenutzer auf den Support und traditionelle IT-Leiter auf den IT-Betrieb fokussiert.Damit das Ganze funktioniert, müssen die Perspektiven stimmig sein und zwar nicht nur für ein System oder Dienst, sondern für alle bestehenden und zukünftigen Systeme/Dienste. Damit ergeben sich folgende Himmelsrichtungen für das Management von IT-Produkten: Entwicklung, Betrieb, Anwendung und Support. Diese sollen in diesem Sinne nicht vorbestimmte Phasen/Zyklen darstellen, sondern idealtypische Perspektiven und Aufgabenkontexte für das Management von IT Produkten, die als Himmelsrichtungen im Sinne der oben dargestellten Metapher verstanden werden können - dies zeigt Abbildung 1.

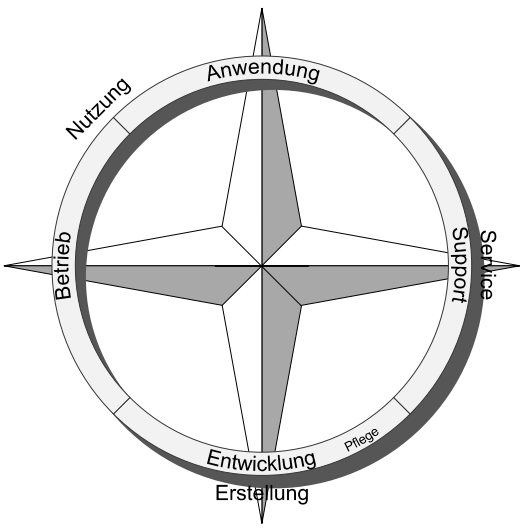


Abbildung 1: Perspektiven (Himmelsrichtungen) für das Management von IT-Produkten

Unter Entwicklung wird die Erstellung des IT-Systems verstanden. Hier beginnen alle IT-Systeme, jedoch ist die Entwicklung kein einmaliger Vorgang, sondern vielmehr ein iterativer Prozess, der mehrfach durchlaufen wird und auch nach der Auslieferung eines IT-Systems nicht beendet ist. Zum Betrieb gehören Aufgaben wie Installation, Sicherung und Operating. Den Bereich der Anwendung kann man in die Administration und den Einsatz unterteilen. Während Einsatz die operative (tägliche) Arbeit mit dem IT-System meint, beschäftigt sich die Administration mit übergreifenden Aufgaben, die sich aus den strategischen Zielen ableiten. Hierzu zählen z. B. die Umsetzung von Sicherheitsregeln nach außen und innen sowie unternehmensweite Einstellungen, die eine Software an unternehmensspezifische Besonderheiten anpasst. Diese Aufgabe stellt damit eine Schnittgrenze zum Betrieb dar, dessen Aufgabe die Unterhaltung eines IT-Systems ist. Der Support lässt sich in einen reaktiven und einen proaktiven Bereich unterteilen, wie dies auch in Standards wie ITIL für das IT-Servicemanagement vorgesehen ist. Den reaktiven Part kann man als Störungsbeseitigung bezeichnen, da sich um die jeweils vorliegenden Störungen gekümmert wird. Im proaktiven Part der Störungsprävention geht es um die Problemerkennung und die Störungsvorbeugung. Außerdem ist im Supportbereich auch der Aufgabenbereich der Beratung anzusiedeln.

Neben diesen Aufgabenkontexten, die gleichzeitig auch eigene Himmelsrichtungen repräsentieren, gibt es weitere Aufgabenkontexte, die in die Segmente der Himmelsrichtungen eingeordnet werden können. Eine Himmelsrichtung stellt eine Richtung dar, auf die sich ein Unternehmen spezialisieren kann. Weitere aus den einschlägigen Lebenszyklusmodellen bekannte Phasen z. B. Installation und Außerdienststellung werden nicht als Segmente des Kompass dargestellt, da jedes Angebot geplant, eingeführt und außerdienst gestellt werden muss. Es handelt sich daher nicht um eine spezifische Ausrichtung für Anbieter von IT-Produkten, sondern um allgemeine Aufgaben, die für jedes Segment anfallen und sozusagen eine weitere (dritte) Dimension bilden. Der IT-Kompass soll die Positionierung von IT-Produkten unterstützen und nicht die Ablaufplanung. Abbildung 2 veranschaulicht, dass der Kompass im Laufe der Zeit mehrfach durchlaufen wird: Es handelt sich um einen Lebens-‚Zyklus' im Sinne des Wortes.

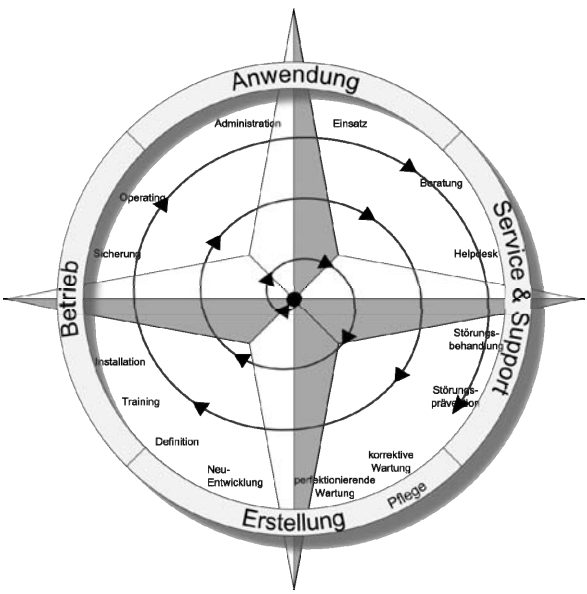


Abbildung 2: Lebenszyklus von IT-Produkten im Kompass

Im Kompass zeigt sich die Folge von Entscheidungen für die Weiterentwicklung und Pflege von IT-Produkten, die in der Praxis oft innerhalb aufeinanderfolgender Projekte getroffen werden. Auf jede Entscheidung für ein bestimmtes Projekt folgt deren Ablauf mit den Phasen: Einführung, Nutzung und Außerdienststellung/Abkündigung. Der Kompass dient als Entscheidungshilfe: Welcher Weg soll eingeschlagen werden? Er ersetzt nicht die Projektplanung, sondern stellt den strategisch/taktischen Rahmen bereit. Für die ausgelieferten Versionen fallen Wartungsaufgaben an, die man als eigenen Bereich „Pflege" zusammenfassen kann, was aber eigentlich ein Teil der Erstellung ist und deshalb auch im Kompass mit diesem Aufgabenkomplex zusammen dargestellt wird. Bei Bedarf lässt sich an dieser Stelle noch zwischen korrektiver Wartung, bei der die Behebung einer Störung oder eines Problems im Vordergrund steht, und perfektionierender Wartung, bei der aus einer Wartung eine erweiterte Funktionalität resultiert, unterscheiden.

Mit dem letzten Aufgabenkomplex schließt sich der Kreis der Aufgaben, die einem Lebenszyklus folgen. Dabei wird hier auch die Besonderheit, dass Wartung (insbesondere perfektionierende Wartung) eine Weiterentwicklung darstellt, berücksichtigt. Mit der entstehenden neuen Version kann der Zyklus erneut beginnen. Der Entwicklungszyklus wird im klassischen Phasenmodell zur Vereinfachung linear aufgelöst. In der Praxis ist der Ablauf jedoch eher unstetig wie in der Abbildung 3 skizziert. Der dargestellte Zyklus spiegelt den evolutionären Charakter der IT im Speziellen und des Managements der IT im Besonderen wider.

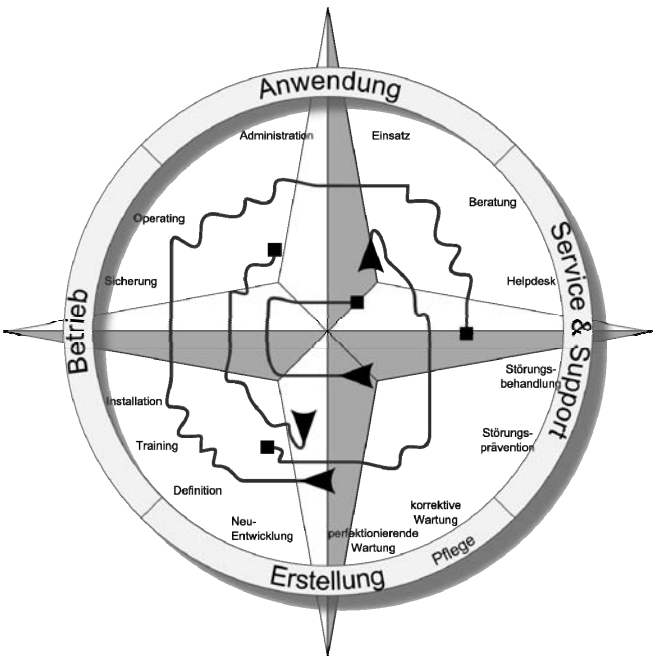


Abbildung 3: Evolution von IT-Produkten im Kompass

3 Abstraktionsebenen für das Management von IT-Produkten

Für eine sinnvolle Positionsbestimmung reichen die Himmelsrichtungen alleine jedoch nicht aus. Es ist ebenfalls erforderlich zu klären, welche Ebene adressiert wird, sonst redet zum Beispiel ein Entwickler über die Datenstruktur mit einem anderen, der an die gesamte Systemarchitektur denkt. Analog kann es zu Missverständnissen kommen, wenn ein Systemanalytiker, der den gesamten Geschäftsprozess vor Augen hat, mit einem Anwender über das Format des Feldes in einer Maske diskutiert. Die erforderlichen Abstraktionsebenen sind nicht an den Aufgabenkontext gebunden, sondern können allgemein bestimmt werden. Für Client-Server-Architekturen ist eine Aufteilung in Daten, Funktionsschicht und Präsentationsschicht üblich, für die Positionierung von IT-Produkten jedoch nicht ausreichend. Abbildung 4 skizziert eine differenziertere Unterteilung für das IT-Produktmanagement.

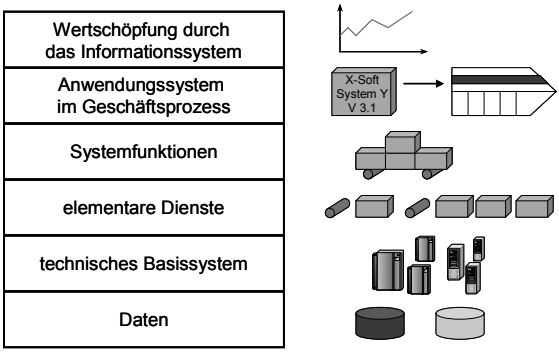


Abbildung 4: Betrachtungsebenen für das Management von IT-Produkten

Ohne ***Daten*** kommt kein IT-Produkt aus, durch IT-Podukte werden daraus verwertbare Informationen, dies gilt für ERP-Systeme ebenso wie für Internet-Dienste. Die Daten sind bei manchen IT-Produkten untrennbar mit einem Basissystem bzw. Dienst verknüpft, sie können jedoch auch wie z. B. Navigationssystemen getrennt vermarktet werden und sollten daher als eine eigenständige Produkt-Schicht betrachtet werden.

Das ***technische Basissystem*** bestehend aus Hardware, Systemsoftware und Peripherie stellt für das Management von IT-Produkten eine eigenständige Betrachtungsebene dar und bildet damit eine Abstraktionsebene bzw. Schicht. Eine weitere Differenzierung ist nicht erforderlich, da das Basissystem im Sinne der zugrundeliegenden Architektur transparent sein sollte. Einen Sonderfall stellen eingebettete Systeme dar. Als technisches Basissystem ist hierbei die unterste Systemschicht, zumeist die unterste Hardwareebene zu verstehen. Die Funktionalität des Systems setzt auf dem technischen Basissystem auf. Dabei kann man im Sinne moderner, mehrschichtiger Systemarchitekturen zwischen elementaren Diensten und komplexen Systemfunktionen unterscheiden.

Elementare Dienste sind Basisfunktionen und werden zum Teil nur intern genutzt wie z. B. ein SMTP-Postausgangsdienst. Sie können nur im Zusammenhang mit anderen Basisfunktionen die Leistungsfähigkeit des Gesamtsystems bereit-/sicherstellen wie z. B. E-Mail. Durch sachgerechte Kombination und Abstimmung von Basisfunktionen (SMTP, POP3, Editor) entstehen ***komplexe Systemfunktionen*** (E-Mail). Durch unterschiedliche Ausgestaltungen einzelner Basisfunktionen eines Virenscanners können z. B. unter Einbeziehung von Zusatzsoftware eine Profi-Systemfunktion und eine Anfänger-Systemfunktion komponiert werden.

Anwendungssysteme kombinieren Einzelfunktionen und technische Einrichtungen zu einem funktionsfähigen technischen System. Die in einer Anwendungssoftware verfügbaren Systemfunktionen ergeben im Kontext eines Geschäftsprozesses ein betriebswirtschaftliches Anwendungs- bzw. Informationssystem (vgl. dazu die Unterscheidung nach [FS06]). Das Anwendungssystem stellt die Funktionalität bereit, erst im ***Geschäftsprozess*** und mit den beteiligten Menschen werden die übermittelten Daten zu betriebswirtschaftlich verwertbaren Informationen. In der modernen Systemanalyse steht der Geschäftsprozess im Mittelpunkt der Betrachtung und die technischen Einrichtungen befinden sich in unserem Modell auf einer tieferen Ebene.

Aus der Perspektive des Top-Managements wird jedoch nicht das gesamte Informationssystem mit seinem Umfeld im Detail betrachtet; dort steht die finale betriebswirtschaftliche Wirkung des Informationssystems, nämlich die erreichte ***Wertschöpfung*** im Mittelpunkt der Betrachtung. Auf der zweiten Ebene von oben werden die Wirkungen im Detail betrachtet: Was leistet das Anwendungssystems im Geschäftsprozess? Auf der nächsten Betrachtungsebene wird das Anwendungssystem in Einzelfunktionen zerlegt und der Nutzwert einzelner Systemfunktionen unabhängig vom Geschäftsprozess adressiert. Mit jeder Betrachtungsebene sind unterschiedliche Fragestellungen, Interessenlagen und dementsprechend auch Argumentationen und Konzeptionen verbunden und erfordern eine differenzierte Behandlung im Management von IT-Produkten.

4 Positionierung von IT-Produkten

4.1 Gegenstand und Ziel der Positionierung

Durch Kombination der Himmelsrichtungen mit den Abstraktionsebenen entsteht ein Polarkoordinatensystem wie in Abbildung 5 dargestellt.

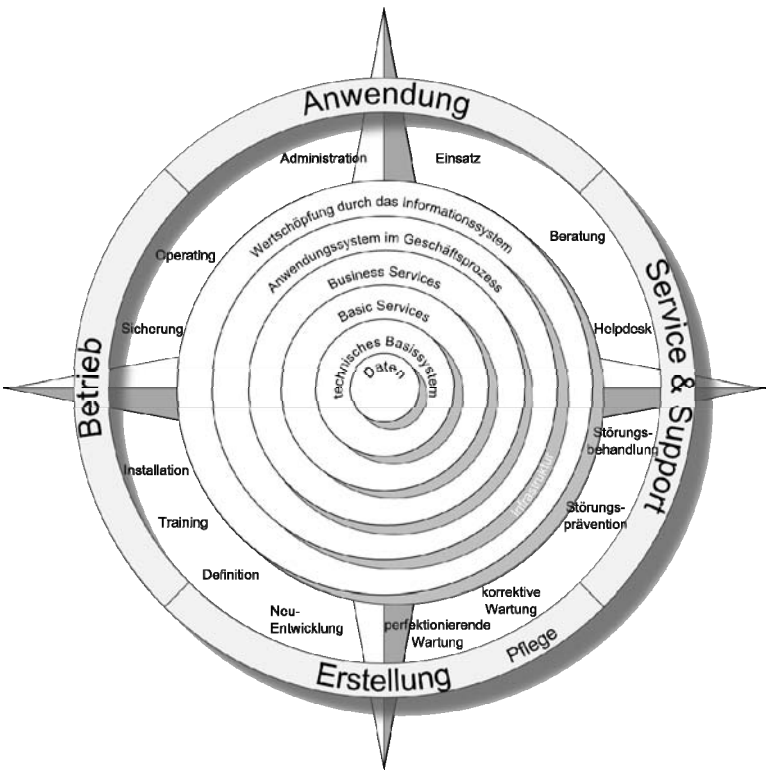


Abbildung 5: Kompass für das Management von IT-Produkten (IT-Produktkompass)

Dieses Angebot passt sicherlich zu einer strategischen Partnerschaft, jedoch nicht zu einer zielgerichteten Produktvermarktung im Sinne der Definition im ersten Kapitel. Das Produkt ist weder abgegrenzt noch ist dessen technische Reife oder die Marktfähigkeit mit einer Partnerschaft begründet. Es ist nicht klar, wo die Stärken des Anbieters liegen und wo er Schnittstellen zu unternehmensinternen Anbietern oder Partnern bzw. Unterschiede zu Mitbewerbern sieht.

Es liegt in der Natur des Produktmanagements, dass es verschiedene Perspektiven, Aufgaben und hierarchische Ebenen in Unternehmen koordiniert und integriert [Be93]. Folglich hängt die Art und Weise wie das Produktmanagement ausgestaltet wird, von dem jeweiligen Kontext ab. Es existieren zwar detaillierte Analysen einzelner Geschäfsmodelle und Branchen (e.g. [Cu98]), es gibt jedoch nur wenige branchenübergreifende Ansatze, die das ganze Spektrum von IT-Produkten und nicht nur das Software-Geschäft adressieren. In einer praxisorientierten Analyse werden die Aufgaben nach dem zugrundeliegenden Geschäftsmodell ausdifferenziert und idealtypische Geschäftsmodelle für IT-Produkte definiert: Lizenzgeschäft, Lizenz plus Service, Projektgeschäft und Systemservicegeschäft [Pi06]. Auf Grundlage von drei dieser vier idealtypischen Geschäftsmodelle soll die Positionierung im IT-Produktkompass veranschaulicht werden.

4.2 Lizenzgeschäft

Das Geschäftsmodell im Lizenzgeschäft basiert auf einem Massengeschäft. Nur für einzelne Konsumgüter ist es möglich, sozusagen aus dem Stand in das Massengeschäft einzusteigen. Beispiele sind z. B. spezifische Dienstprogramme, die im Internet gehandelt werden und sich durch persönliche Empfehlungen verbreiten. IT-Produkte erfordern üblicherweise eine breite Kundenbasis als Ausgangspunkt für die Vermarktung im reinen Lizenzgeschäft ohne ergänzende Dienstleistungen. Damit das IT-Produkt in unterschiedlichen Einsatzfeldern bestehen kann, muss das IT-Produkt einerseits ausgereift sein, d. h. es muss praktisch und technisch funktionieren. In einem Massenmarkt treten die individuellen Anforderungen der Anwender zugunsten der Funktionalität zurück. Innerhalb der Werbung und des Vertriebs werden die angebotenen Funktionen mittels generischer Nutzenargumentationen in die individuellen Anforderungen übersetzt, z. B.: Die Schnittstelle zu PDA's ermöglicht das orts- und zeitunabhängige Arbeiten im Außendienst.

Je klarer die Argumentation auf ein bestimmtes Einsatzszenario oder eine Branche ausgerichtet ist, desto überzeugender ist die Positionierung. Allerdings wird durch die damit verbundene Beschränkung der Zielmarkt kleiner und damit auch die Voraussetzung für ein Massengeschäft. Unabhängig von der Kommunikation sollte das Massenprodukt selbst klar definiert und abgegrenzt sein. Dabei ist die Vermarktung von den Ergebnissen der Produktentwicklung abhängig. Nicht nur bei IT-Produkten entscheidet der Übergang von der Entwicklung zur Vermarktung über den Produkterfolg. Die Fokussierung ist typischerweise keine Stärke von Entwicklern. Daher kann eine wesentliche Leistung der Produktvermarktung darin bestehen, aus den umfangreichen Spezifikationen eine marktgerechte, fokussierte Produktdefinition zu entwickeln.

Das Polarkoordinatensystem spannt den Gestaltungsrahmen für die Positionierung von IT-Produkten auf. Bei der dargestellten Aufteilung in Ebenen und Perspektiven stehen die Aufgaben im Mittelpunkt: Aufgaben der Erstellung von Daten, des technischen Basissystems, der Funktionen etc.; Aufgaben der Nutzung von Daten etc. Für die Ausführung dieser Aufgaben ist eine technische und organisatorische Infrastruktur erforderlich von Servern über Räume bis hin zum Personal. Diese Infrastruktur ist eine notwendige Voraussetzung, steht jedoch nicht im primären Fokus des Managements, sondern stellt eine sekundäre Dimension dar und wird daher nicht im Vordergrund sondern perspektivisch als ‚Tiefe' des Modells dargestellt. Durch die Infrastruktur werden die Schichten verbunden, im IT-Produktkompass sind sie deshalb auf allen Ebenen als ‚Rand' der konzentrischen Kreise dargestellt.

Für die Komposition der komplexen Strategie und zielgerichtete Positionierung von IT-Produkten eignet sich der in Kapitel 1 eingeführte Kompass für das Management von IT-Produkten (IT-Produktkompass) als Orientierungshilfe. Eine unklare Positionierung der IT-Produkte erschwert nicht nur die interne Organisation und auch die Kommunikation nach Außen, sondern ist auch ein Ausdruck eines strategischen Defizits, der in der Praxis oft durch zusätzliche Intensivierung der Bemühungen kompensiert wird: Es werden sehr viele Aktivitäten entwickelt, die jedoch weder effizient noch effektiv sind und dementsprechend verpuffen. Abbildung 6 gibt ein Beispiel für ein mittelständisches Softwarehaus mit einem umfassenden Angebot von der Beratung über Entwicklung, Einführung, Betrieb und Help-Desk bis hin zur Wartung.

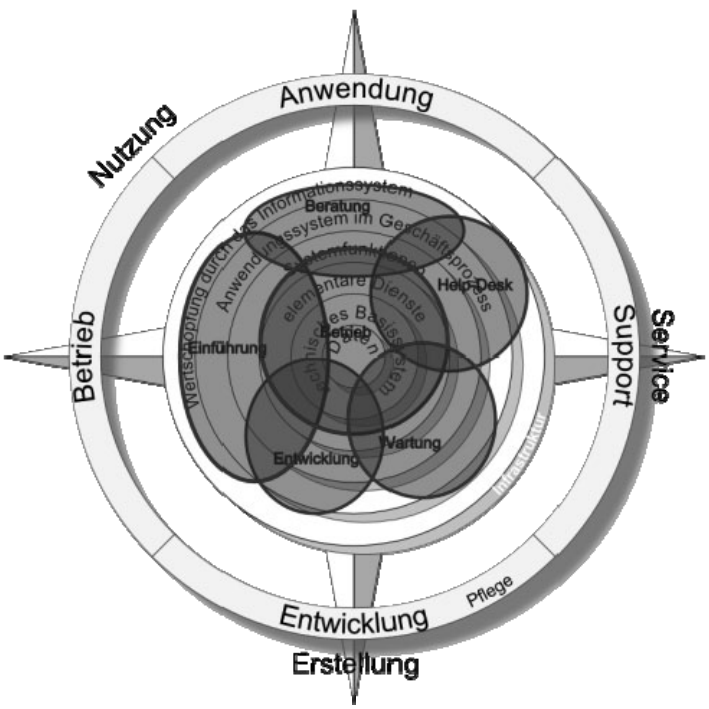


Abbildung 6: Beispiel für eine umfassende und undifferenzierte Positionierung

Während das Geschäftsmodell Vermarktung von IT-Standardprodukten im Lizenzgeschäft die Marktfähigkeit begründet, ist die IT-Standardproduktentwicklung für die technische Reife verantwortlich. Das Ergebnis der Entwicklung wird als Lizenz vertrieben, jedoch nicht die Entwicklung als individuelle Dienstleistung. Ohne IT-Standardproduktentwicklung, unabhängig davon ob diese selbst oder durch Partner ausgeführt wird, ist keine Vermarktung möglich. Deshalb wurde die Entwicklung in Abbildung 7 in den IT-Produktkompass eingeordnet, allerdings nicht in blau wie die Produkte, sondern rot im Sinne einer Vorleistung oder Ressource dargestellt.

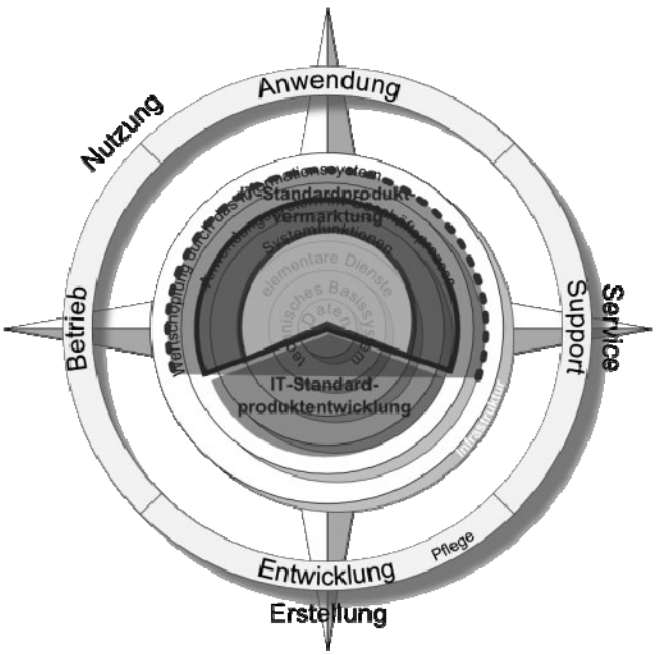


Abbildung 7: Positionierungsbereich für das Geschäftsmodells IT-Standardprodukte

Wenn es sich wirklich um ein Massengeschäft und ein Standardprodukt handelt, erfordern die Entwicklung und die Vermarktung von IT-Produkten andere Ziele, Rollen, Kompetenzen und Prozesse. Grundlegende Innovationen entstehen oft ungeplant aus einer Technologie heraus. Auch die besten Innovationen und auch die leistungsfähigsten Produkte haben geringere Erfolgschancen, wenn Sie nicht die Bedarfe der Anwender treffen. Die beiden Geschäftsmodelle im Lizenzgeschäft sind komplementär. Die Klarheit und Konsequenz der Trennung zwischen Vermarktung und Entwicklung ist Ausdruck der Professionalität im Lizenzgeschäft.

Neben den (technologiegetriebenen) Innovationen ist die Entwicklung für eine kontinuierliche Releaseplanung verantwortlich, um den Anwender die Planungssicherheit zu geben – ein wichtiges Merkmal der Dienstleistungsqualität. Für den langfristigen Produkteinsatz ist die Produktpflege mit Nach-/Verbesserungen sowie die Migration auf moderne Plattformen wichtig: Auch dieses Merkmal, die Investitionssicherung bzw. Zukunftsfähigkeit des IT-Produktes, bestimmt das Vermarktungspotential.

4.3 Lizenz plus Service

Neben dem Lizenzgeschäft bieten fast alle Anbieter von IT-Produkten auch ergänzende Dienstleistungen wie Schulungen, kleinere Anpassungen etc. an. Dienstleistungen können das Produktportfolio im Lizenzgeschäft ergänzen, wie die Zertifizierungen bei Toolanbietern, oder das Projektgeschäft kann durch Lizenzen erst begründet werden, wie bei betriebswirtschaftlicher Standardsoftware. Wenn Lizenz- und Projektgeschäft nicht möglich sind oder auch um das Marktfeld zu erweitern, kann durch zielgerichtete Kombination von Lizenzen und Services ein eigenständiges Geschäftsmodell entstehen. So gibt es IT-Produkte, deren Nutzwert sich nicht allein durch die Funktionsbeschreibungen erschließt, die jedoch die Begründung eines Projektes nicht rechtfertigen. Durch ergänzende Dienstleistungen, z. B. eine Schulung oder eine kleine Vorstudie kann das Produkt vom Anwender erfasst werden oder die Machbarkeit nachgewiesen werden.

Abbildung 8 veranschaulicht den Positionierungsbereich im IT-Kompass. Es werden Lizenzen für die Nutzung von Daten bis hin zu Elementen des Anwendungssystems bereitgestellt und standardisierte IT-Services veredeln diese Lizenzen bezüglich des Nutzwertes im individuellen Unternehmen und in Richtung Einführung/Installation und Support/Pflege.

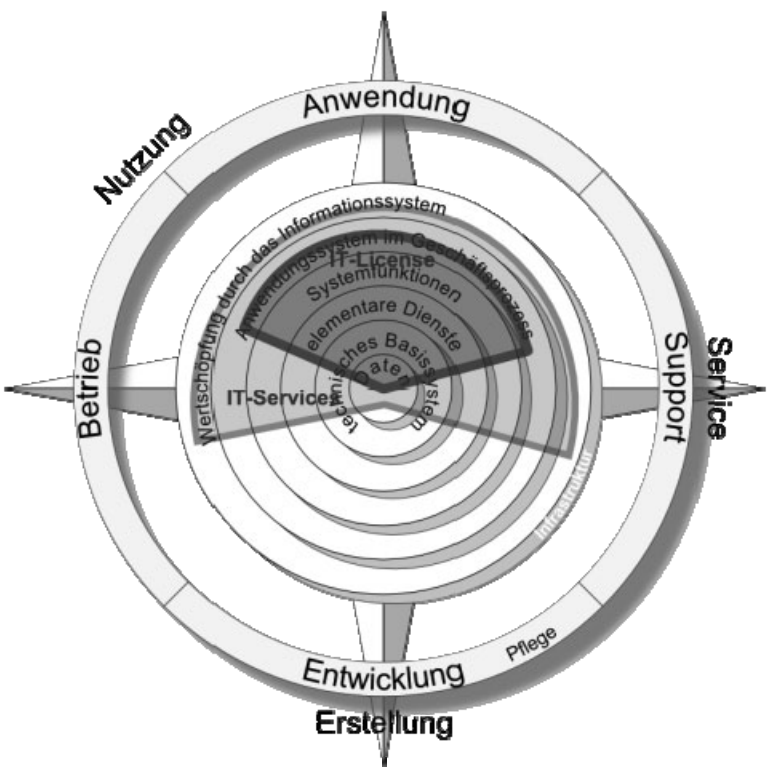


Abbildung 8: Positionierungsbereich für das Geschäftsmodell IT-License plus Service

Das Geschäftsmodell Lizenz plus Service vereint Aspekte von Lizenz- und Projektgeschäft. Anders als beim reinen Lizenzgeschäfts ist das IT-System nicht fest standardisiert, sondern es gibt einen standardisierten Kern, der eingeschränkt angepasst werden kann, z. B. hinsichtlich der Präsentation, dem Reporting oder auch spezifischer lokaler Schnittstellen. Lizenz plus Service wird typischerweise als gemischter Bereitstellungs- und Werkvertrag angeboten. Die ergänzenden Dienste müssen deshalb – anders als beim Projektgeschäft - klar definiert und einfach planbar sein. Wegen der eingeschränkten Anpassung ist das Anwendungsfeld des Geschäftsmodells Lizenz plus Service Geschäft eingeschränkt – einer Durchdringung eines Anwenders wie beim Projektgeschäft sind enge Grenzen gesetzt. Daher liegt der Schwerpunkt zumeist beim Neukundengeschäft – oft werden spezifische Nischen gezielt bearbeitet.

4.4 Projektgeschäft

Das IT-Projektgeschäft wird durch Dienstleistungen in Form von zwei Teil- Geschäftsmodellen, der IT-Projektdurchführung und der IT-Projektbegründung bestimmt; Aspekte des Vertrauensgütermarketings spielen eine zentrale Rolle. Referenzen sind von entscheidender Bedeutung bei der Akquisition und die Kundenzufriedenheit ein zentrales Erfolgsmaß.

Für die IT-Projektdurchführung kann auf einschlägige Software-Prozessmodelle bzw. Industriestandards zurückgegriffen werden. Bei der Projektbegründung dagegen stehen Aspekte des Multi-Pojekt- bzw. Programm-/Portfoliomanagements im Mittelpunkt. Die Projektbegründung wird in größeren Unternehmen vielfach entweder organisatorisch im Unternehmen getrennt von der Durchführung behandelt oder der jeweils nicht im eigenen Unternehmen verrichtete Teil sogar an Vertragspartner vergeben. Standardisierung und Professionalisierung erfolgt beim Projektgeschäft in Form von Geschäftsprozessen – der Prozess ist das Produkt, bzw. dessen Effektivität und Effizienz sowie Glaubwürdigkeit für die Beteiligten. Je nach Zielgruppe können schlanke, persönliche Prozesse oder aufwendige, fest strukturierte Prozesse sinnvoll sein. Firmenimage und Public Relations sind strategische Gestaltungsbereiche im Rahmen der Vermarktung, die produktbezogene Kommunikation eher ein operatives Vertriebsinstrument.

Wird Projektgeschäft als Produkt vertrieben, ist der Prozess das Produkt – von der Anforderungsanalyse bis hin zur Gewährleistung. Im Rahmen des Geschäftsmodells IT-Projektbegründung werden ein oder auch mehrere Projekte aufgesetzt, die dann im Rahmen des Geschäftsmodells IT-Projektdurchführung geordnet abgewickelt werden. Die Verantwortung für das Ergebnis nach Projektabschluss, insbesondere der Betrieb des IT-Systems, liegt prinzipiell beim Kunden. Abbildung 9 stellt das IT-Projektgeschäft im IT-Produktkompass dar.

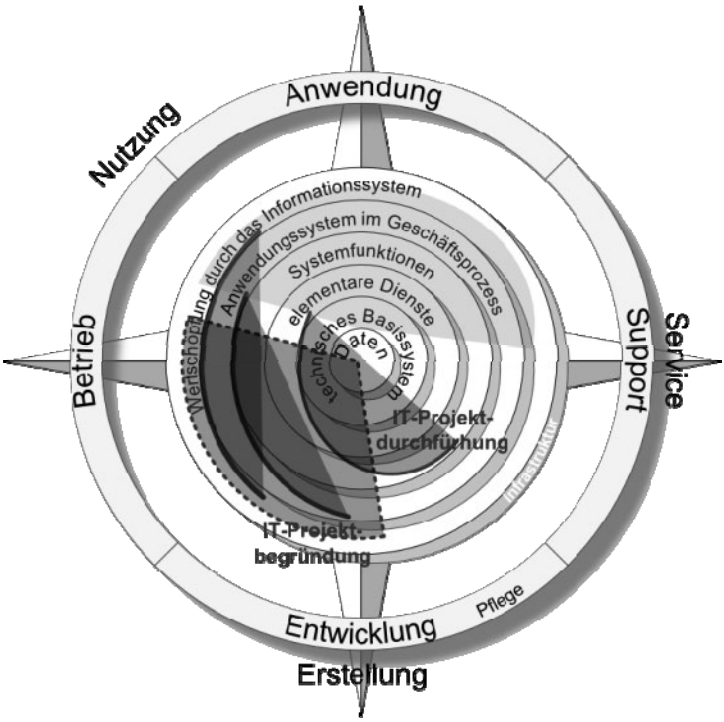


Abbildung 9: Positionierungsbereich für das Projektgeschäft

Bei der IT-Projektbegründung wird eine umfassende Perspektive mit gleitenden Übergängen zu den unterschiedlichen Aufgabenbereichen und Ebenen gewählt, die resultierenden IT-Projekte werden klar definiert und abgegrenzt, damit Sie im Projektabwicklungsprozess abgearbeitet werden können.

5 Fazit und Ausblick

Die Anwendung des IT-Produktkompass auf drei generische Geschäftsfelder zeigt deutliche Unterschiede hinsichtlich des Handlungsfeldes des IT-Produktmanagements. Die einzelnen Geschäftsmodelle sind nicht als Bausteine zu betrachten, die beliebig kombiniert werden können, sondern drücken unterschiedliche Strategiemuster im Sinne Porters klassischen generischen Strategien auf [Po85]. So ist eine Kombination von Lizenz- und Projektgeschäft zwar technisch und organisatorisch praktikabel, jedoch im analog zu Porter's ‚stuck in beetween the middle' als gefährliche Zwischenlage zu bewerten.

Der IT-Produktkompass eignet sich nicht nur als Hilfsmittel für die Positionierung von IT-Produkten, sondern für Vermarktungsstrategie, Vertriebskonzepte, das Servicemanagement und auch für wirtschaftliche Betrachtungen. Eine Monographie zum Management von Softwareprodukten mit Hilfsmitteln wie dem IT-Produktkompass erscheint in Kürze [HP08].

Literaturverzeichnis

[AH07] Albers, S.; Herrmann, A. (Hrsg.): Handbuch Produktmanagement: Strategieentwicklung - Produktplanung - Organisation – Kontrolle, Gabler: Wiesbaden 2007.

[Be01] Becker, J.: “Marketing-Konzeption. Grundlagen des zielstrategischen und operativen Marketing-Managements”, Vahlen, München 2001.

[Bo76] Boehm, B.W.: Software Engineering. In: IEEE Transactions on Computers. Vol. 25, No. 12, 1976.

[CC02] Conde, D.; Condon, D.: Software Product Management: Managing Software Development from Idea to Product to Marketing to Sales, Aspatore 2002.

[Cu98] Cusomano, M.A.: Microsoft Secrets: How the World's Most Powerful Software Company Creates Technology, Shapes Markets and Manages People, Free Press 1998.

[Dv03] Dver, A.S.: Software Product Management Essentials, Meghan Kiffer 2003.

[FS06] Ferstl, O. K.; Sinz, E.J.: Grundlagen der Wirtschaftsinformatik, Oldenbourg 2006.

[HP08] Herzwurm, G.; Pietsch W.: Management von Softwareprodukten, erscheint bei dpunkt vorauss. Ende 2008

[LW04] Lehmann, D.R.; Winer, R.S.: Product Management, McGraw Hill 2004.

[PBL05] Pohl, K.; Böckle, G.; van der Linden, F.: Software Product Line Engineering. Foundations, Principles, and Techniques, Springer 2005

[Pi05] Pietsch, W.: Customer-Oriented Specification and Evaluation of IT Service Level Agreements", in Richardson, I. (ed.): 12th European conference on Software Process Improvement (EuroSPI 2005), Budapest, November 9 - 11, 2005. Lecture notes in computer science 3792, Springer 2005, S. 83 - 94.

[Pi06] Pietsch, W.: Geschäftsmodelle als Grundlage für das Software-Produktmanagement, in Lehner, F. (Hrsg.): Multikonferenz Wirtschaftsinformatik 2006 in Passau, Berlin 2006, Band. 2. S. 211 - 222.

[Po85] Porter, M.E.: Competitive Advantage, New York 1985.

[RHH05] Rausch, A.; Höhn, R.; Höppner, S.: Das V-Modell XT, Springer, Berlin 2005.

[We06] van de Weerd, I.; Brinkkemper, S.; Nieuwenhuis, R.; Versendaal, J.M.; Bijlsma, A.: “On the Creation of a Reference Framework for Software Product Management: Validation and Tool Support”, Proc. of the International Workshop on Software Product Management, St. Paul / Minnesota, 2006, pp. 3-12.

Warum entstehen in der Anforderungsanalyse Fehler? Eine Synthese empirischer Befunde der letzten 15 Jahre

Oral Avci

SQS Software Quality Systems AG
Stollwerckstr. 11
51149 Köln
oral.avci@sqs.de

Abstract: In Softwareentwicklungsprojekten treten viele Anforderungsfehler auf. Diese werden häufig erst entdeckt, nachdem sie zu zahlreichen Folgefehlern im Entwurf und der Implementierung geführt haben. Der vorliegende Beitrag liefert auf der Grundlage empirischer Befunde eine Antwort auf die Frage, warum Anforderungsfehler entstehen. Mit der Kenntnis ihrer Ursachen können Anforderungsfehler vermieden oder durch gezielte Maßnahmen der Qualitätssicherung früher gefunden werden.

1 Einführung

Die Anforderungsanalyse ist eine Teilaufgabe der Softwareentwicklung mit dem Ziel aus Kundenbedürfnissen Softwareanforderungen abzuleiten. Eine Softwareanforderung beschreibt dabei ein gewünschtes funktions- oder qualitätsbezogenes Merkmal einer Software aus Sicht der Kunden, wobei diese Beschreibung unabhängig von dem Entwurf und der Implementierung dieses Merkmals ist. Seit drei Jahrzehnten wird die Anforderungsanalyse als eine eigenständige Teilaufgabe der Softwareentwicklung wahrgenommen. Trotz zahlreicher Fortschritte weist die Praxis der Anforderungsanalyse ein chronisches Problem auf [AW05]: es treten viele Anforderungsfehler auf. Diese werden meist erst entdeckt, nachdem zahlreiche Folgefehler im Entwurf und der Implementierung aufgetreten sind. Der relative Anteil der Anforderungsfehler und ihrer Folgefehler an der Gesamtfehlerzahl in Projekten ist deshalb hoch. In mehreren Studien bilden sie die am häufigsten genannte Fehlerkategorie und können bis zu 50 % aller Fehler ausmachen [BP84], [LV01].

Anforderungsfehler können vermieden werden, wenn ihre Ursachen beseitigt werden. Hierzu ist zu klären, welcher Aspekt der Durchführung der Anforderungsanalyse unter den gegebenen Projektrahmenbedingungen begünstigt hat, dass Anforderungsfehler entstehen. In bestimmten Fällen ist die Beseitigung eines Prozessmangels der Anforderungsanalyse jedoch nicht möglich oder mit erheblichen Nachteilen verbunden, so dass sie ausbleibt. Auch dann ist die Kenntnis der Ursache von Anforderungsfehlern hilfreich, um mit gezielten Maßnahmen der Qualitätssicherung nach entstandenen Anforderungs- und deren Folgefehlern zu suchen.

Folglich stellt sich die Frage, warum Fehler in der Anforderungsanalyse entstehen. Die vorliegende Arbeit liefert einen Beitrag zur Beantwortung dieser Frage. Auf der Grundlage einer systematischen Literaturanalyse erfolgt eine Synthese empirischer Befunde zu Anforderungsfehlern und ihren Ursachen. Die Literaturanalyse berücksichtigt insbesondere Beiträge, die im Zeitraum von Anfang Januar 1990 bis einschließlich Ende Dezember 2005 in sechzehn wissenschaftlichen Zeitschriften und Konferenzbänden von acht Konferenzen veröffentlicht worden sind.

Der vorliegende Beitrag ist wie folgt aufgebaut. In Kapitel 2 wird der theoretische Bezugsrahmen des Beitrags definiert. Kapitel 3 beschreibt, wie die systematische Literaturanalyse als Forschungsmethode eingesetzt wurde. Ergebnis der Literaturanalyse sind Erklärungsmuster für Anforderungsfehler, die auf empirischen Befunden basieren. Diese werden in Kapitel 4 dargestellt. Zentrale Ergebnisse der Literaturanalyse werden in Kapitel 5 diskutiert und mit weiteren empirischen Erkenntnissen in der Literatur verglichen. Die Arbeit schließt mit einem Fazit ab.

Eine detaillierte Beschreibung dieser Untersuchung findet sich in [Av08].

2 Theoretischer Bezugsrahmen

In Anlehnung an [Ie90] und [Ch96] können die Begriffe Irrtum (engl. error), Fehler (engl. fault) und Fehlverhalten (engl. failure) unterschieden werden. Ein Projektmitarbeiter kann sich irren, während er ein Arbeitsergebnis erstellt, d. h. Softwareanforderungen erarbeitet, Entwurfsentscheidungen trifft oder die Software implementiert. Sein *Irrtum* kann dazu führen, dass das bearbeitete Ergebnis mit einem oder mehreren Fehlern behaftet ist (vgl. Abbildung 1). Dabei sind unter einem Irrtum alle Ereignisse zu verstehen, in denen eine geplante Abfolge von geistigen oder körperlichen Aktivitäten nicht zum beabsichtigten Ergebnis führt, sofern dies nicht einem fremden Einwirken zugeschrieben werden kann. Ein *Fehler* ist ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Softwareentwicklung, sofern es eine Änderung in diesem Ergebnis notwendig macht. Es können Anforderungs-, Entwurfs- und Implementierungsfehler voneinander abgegrenzt werden, je nachdem, ob Entscheidungen in der Anforderungsanalyse, dem Entwurf oder der Implementierung betrachtet werden. Insbesondere werden hier also unter einem Fehler nicht ausschließlich Implementierungsfehler aufgefasst.

Da einzelne Entscheidungen der Teilaufgaben zusammenhängen, kann sich ein Anforderungsfehler als Entwurfs- und Implementierungsfehler fortsetzen (vgl. Abbildung 1). Diese besonderen Entwurfs- und Implementierungsfehler sind also *Folgefehler*. Ein *originärer Fehler* liegt dagegen vor, wenn er infolge eines Irrtums neu eingeführt wird und damit unabhängig von einem Fehler einer vorgelagerten Entscheidung ist. Immer wenn ein Projektmitarbeiter einem *Irrtum* unterliegt, finden originäre Fehler Eingang in ein Ergebnis. Eine Software kann ein *Fehlverhalten* offenbaren, wenn sie einen Implementierungsfehler enthält und ausgeführt wird. Dieses Fehlverhalten ist also die Wirkung eines Implementierungsfehlers.

Tritt ein Fehlverhalten auf, folgt daraus ausschließlich, dass die Software mindestens einen Implementierungsfehler hat. Die Kenntnis eines Fehlverhaltens allein erlaubt jedoch keine Rückschlüsse darüber, welche oder wie viele Implementierungsfehler vorhanden sind, da mehrere Implementierungsfehler gemeinsam als ein Fehlverhalten wahrgenommen werden können.

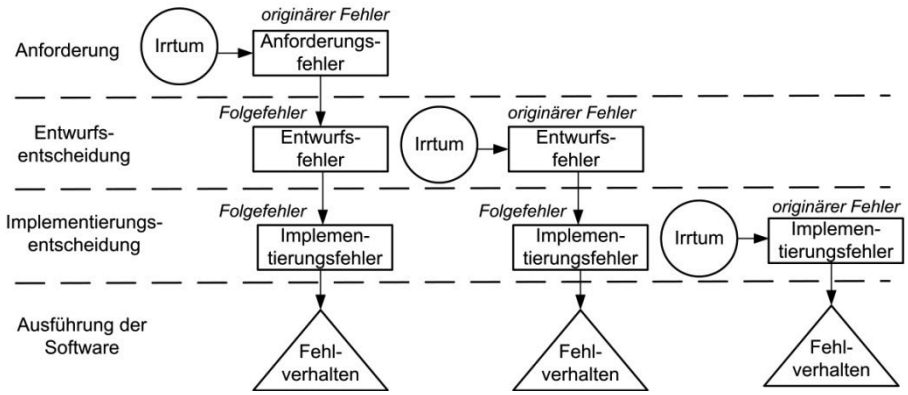


Abbildung 1: Entstehung von Fehlern und ihre Auswirkungen

Ein *Mangel in einem Prozess*, kurz *Prozessmangel*, ist eine Gestaltungsmaßnahme eines Prozesses, die menschliche Irrtümer begünstigt, so dass Fehler in Arbeitsergebnisse der Softwareentwicklung eingeführt werden. Werden diese originären Fehler nicht entdeckt, können zudem im weiteren Verlauf Folgefehler entstehen.

3 Forschungsmethodik

Eine systematische Literaturanalyse untersucht mit einem definierten Ziel einen festgelegten Ausschnitt der Literatur, um eine Synthese relevanter Aussagen zu erreichen. Die Literaturanalyse des vorliegenden Beitrags soll folgende Fragen beantworten:

- Welche Prozessmängel der Anforderungsanalyse sind gemäß Hinweisen in empirischen Untersuchungen bekannt?
- Welche Gemeinsamkeiten weisen die durch Prozessmängel verursachten Fehler jeweils auf?

Es werden ausschließlich Literaturquellen in der Literaturanalyse berücksichtigt, die die folgenden drei Kriterien erfüllen:

- Die Literaturquelle ist empirisch, d. h. gemachte Aussagen beruhen auf Beobachtungen der Realität.
- Die empirischen Aussagen beziehen sich auf die Entwicklung oder Wartung von Anwendungssoftware. Sofern sich Aussagen auf Systemsoftware oder systemnahe Software beziehen, werden sie nur dann berücksichtigt, sofern sie auch auf Anwendungssoftware übertragbar sind.
- Die empirischen Aussagen geben explizit Hinweise auf Maßnahmen im Prozess der Anforderungsanalyse, die Fehler verursachen.

In der Literaturanalyse wurden Beiträge berücksichtigt, die im Zeitraum von Anfang Januar 1990 bis einschließlich Ende Dezember 2005 veröffentlicht worden sind. Insgesamt sechzehn wissenschaftlichen Zeitschriften[1] und Konferenzbände von acht Konferenzen[2] wurden nach relevanten Literaturquellen durchsucht.

Der Ablauf der Suche nach relevanten Literaturquellen orientiert sich an den Empfehlungen von Webster und Watson [WW02]. Im ersten Schritt, der Ausgangssuche, wurden im definierten Literaturspektrum mit sechzehn Zeitschriften und acht Konferenzen nach Literaturquellen gesucht, die die Auswahlkriterien erfüllen. Die Quellen, die den Kriterien genügen, sind die Ausgangstreffer. Im zweiten Schritt wurden die Literaturverzeichnisse der Ausgangstreffer nach weiteren Literaturquellen durchsucht, die die Auswahlkriterien erfüllen (Rückwärtssuche). Schließlich wurde in einer Vorwärtssuche sowohl über Web of Science[3] als auch Citeseer[4] ermittelt, welche Literaturquellen die Ausgangstreffer zitieren. Jede so gefundene Literaturquelle wurde dahingehen überprüft, ob sie die Auswahlkriterien erfüllt.

Die Literaturrecherche wurde im Januar und Februar 2006 durchgeführt. In der Ausgangssuche wurden 17 Literaturquellen identifiziert, die die Auswahlkriterien erfüllen. Die Rückwärtssuche erweitert diese Menge um weitere 3 Literaturquellen, die Vorwärtssuche um eine Literaturquelle. Folglich liegen insgesamt 21 Treffer vor. Diese sind im Einzelnen: Q1 [AS85], Q2 [AE96], Q3 [Bh94], Q4 [Br94], Q5 [CK98], Q6 [Cu88], Q7 [Da04a], Q8 [Da05], Q9 [DZ03], Q10 [Da04b], Q11 [ED05], Q12 [Gü96], Q13 [HL01], Q14 [Ka02], Q15 [LV01], Q16 [Lu93], Q17 [Mo98], Q18 [NK91], Q19 [Su99], Q20 [Ta95], Q21 [Wa93].

[1] Folgende Zeitschriften wurden berücksichtigt: Communications of the ACM, Communications of the AIS, Empirical Software Engineering, European Journal of Information Systems, IBM Systems Journal, IEEE Software, IEEE Transactions on Software Engineering, Information Systems Journal, Information Systems Research, Journal of Management Information Systems, Journal of the Association for Information Systems, Management Science, MIS Quarterly, Requirements Engineering, Software Quality Journal, Wirtschaftsinformatik.

[2] Folgende Konferenzen wurden berücksichtigt: IEEE International Conference on Requirements Engineering, IEEE International Conference on Software Maintenance, IEEE International Symposium on Requirements Engineering, International Conference on Information Systems, International Conference on Software Engineering, International Symposium on Empirical Software Engineering, International Symposium on Software Metrics, International Symposium on Software Reliability Engineering.

[3] Web of Science ist unter der URL http://scientific.thomson.com/products/wos/ zu finden (Stand 26.07.2006). Es ist ein kostenpflichtiges Produkt der Thomson Scientific.

[4] Die CiteSeer Scientific Literature Digital Library ist unter der URL http://citeseer.ist.psu.edu/ zu finden (Stand 26.07.2006).

Ein konstituierendes Merkmal einer systematischen Literaturanalyse ist, dass für den festgelegten Ausschnitt der Literatur eine Synthese relevanter Aussagen angestrebt wird. Im ersten Schritt wurden hierzu die identifizierten Literaturquellen nach Abschluss der Literaturrecherche mit dem Ziel ausgewertet, Beobachtungen zu Ursache-Wirkungszusammenhängen zu extrahieren. Um eine Synthese dieser Beobachtungen vorzubereiten, wurden schrittweise mit jeder gefundene Beobachtung zum einen Typen von Anforderungsfehlern und zum anderen Ursachen von Anforderungsfehlern inhaltlich gruppiert.

4 Erklärungsmuster für Anforderungsfehler

In der Literaturanalyse wurden 21 empirische Literaturquellen ausgewertet. Diese Quellen tragen zu insgesamt 92 relevanten Beobachtungen bei. Eine Beobachtung ist dabei ein Ursache-Wirkungszusammenhang mit folgender Struktur: ein Prozessmangel der Anforderungsanalyse (Ursache) hat einen bestimmten Typ von Anforderungsfehlern oder generell Anforderungsfehler zur Folge (Wirkung). Sofern eine Ursache verschiedene Typen von Anforderungsfehlern bewirkt, wird für jeden Typ eine einzelne Beobachtung erfasst.

4.1 Typologie von Anforderungsfehlern

Ein Anforderungsfehler ist ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Anforderungsanalyse, sofern es eine Änderung in diesem Ergebnis notwendig macht. Aus der Auswertung von 92 Beobachtungen zu Ursache-Wirkungszusammenhängen kann eine Typologie von Anforderungsfehlern hergeleitet werden. Die Abbildung 2 stellt diese Typologie von Anforderungsfehlern anhand eines hierarchischen Baumdiagramms dar.

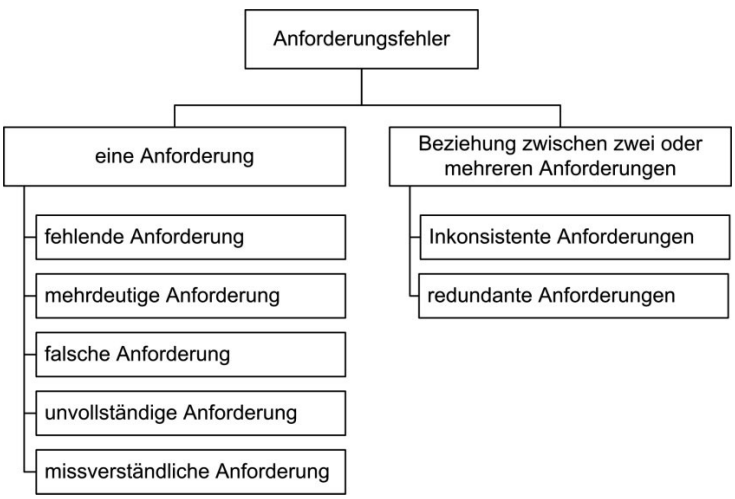


Abbildung 2: Typologie von Anforderungsfehlern

Wenn sich ein Anforderungsfehler auf eine Anforderung bezieht, können insgesamt fünf Fälle voneinander abgegrenzt werden:

- Für ein bestehendes Kundenbedürfnis wurde eine Anforderung gänzlich übersehen (fehlende Anforderung).
- Die Beschreibung der Anforderung kann auf mehrere Arten gedeutet werden (mehrdeutige Anforderung).
- Die Anforderung erfüllt kein Kundenbedürfnis oder erfüllt sie nicht korrekt (falsche Anforderung).
- Die Anforderung ist unvollständig beschrieben (unvollständige Anforderung).
- Die Anforderung ist zwar vollständig beschrieben, jedoch fehlen Kontextinformationen zu dieser Anforderung oder Verknüpfungen zu vorhandenen Kontextinformationen, um die Anforderung zu verstehen (missverständliche Anforderung).[5]

Ein Anforderungsfehler kann sich auch auf eine Beziehung zwischen zwei oder mehreren Anforderungen beziehen. Zwei Fälle können dabei unterschieden werden:

- Eine Anforderung steht im Widerspruch zu einer oder mehreren anderen Anforderungen (inkonsistente Anforderungen).
- Eine Anforderung ist redundant zu mindestens einer anderen Anforderung (redundante Anforderungen).

In 39 Beobachtungen (42 %) werden als Wirkung Anforderungsfehler allgemein beschrieben ohne einen Typ zu konkretisieren. Folglich ist der Typ des Anforderungsfehlers für 53 Beobachtungen ermittelbar. Darunter sind missverständliche (16 %), inkonsistente (12 %) und fehlende Anforderungen (11 %) am häufigsten vertreten.

4.2 Ursachen für Anforderungsfehler

Warum entstehen Anforderungsfehler? Dies ist die Ausgangsfrage des vorliegenden Beitrags. Nachfolgend werden Erklärungsmuster für einzelne Typen von Anforderungsfehlern formuliert. Diese Erklärungsmuster basieren auf den extrahierten Beobachtungen der systematischen Literaturanalyse. Ein Erklärungsmuster beschreibt jeweils, welche Ursache einen bestimmten Typ von Anforderungsfehler verursachen kann.

[5] Kontextinformationen sind z. B. Kundenbedürfnisse, für die Anforderung relevante Kundengruppen oder andere abhängige Anforderungen.

Insgesamt 20 Ursachen können nach der Auswertung der 92 Beobachtungen unterschieden werden. Diese Ursachen können zum einen direkt Prozessmängel der Anforderungsanalyse im Sinne dieser Arbeit sein. Zum anderen können es Rahmenbedingungen der Softwareentwicklung sein, die Anforderungsfehler verursachen. Diese Rahmenbedingungen können nicht geändert werden. Vielmehr muss die Gestaltung eines Prozesses der Anforderungsanalyse diese Rahmenbedingungen angemessen berücksichtigen, um ihre negativen Wirkungen zu reduzieren. Werden relevante Rahmenbedingungen in einem Prozess der Anforderungsanalyse nicht berücksichtigt, stellt dies ebenfalls einen Prozessmangel der Anforderungsanalyse dar. Insofern beschreiben Beobachtungen, in denen Rahmenbedingungen als Ursachen für Anforderungsfehler genannt werden, indirekt Prozessmängel der Anforderungsanalyse.

Die Ursachen für fehlende Anforderungen liegen insbesondere in der Dokumentation von Anforderungen, der Ausgestaltung der Benutzerbeteiligung, dem fehlendem Wissen über das Anwendungsgebiet der Software sowie einer unzureichenden Auseinandersetzung mit Kundenbedürfnissen (vgl. Abbildung 3).

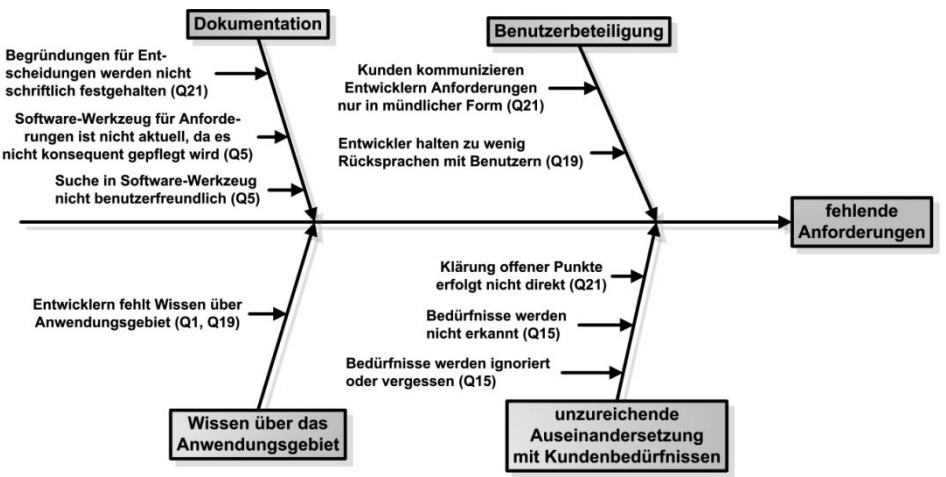


Abbildung 3: Ursachen für fehlende Anforderungen

Die Ursachen für mehrdeutige Anforderungen liegen insbesondere in fehlendem Wissen über das Anwendungsgebiet der Software, der späten Klärung offener Punkte und der Gestaltung des Inhalts des Anforderungsdokuments (vgl. Abbildung 4).

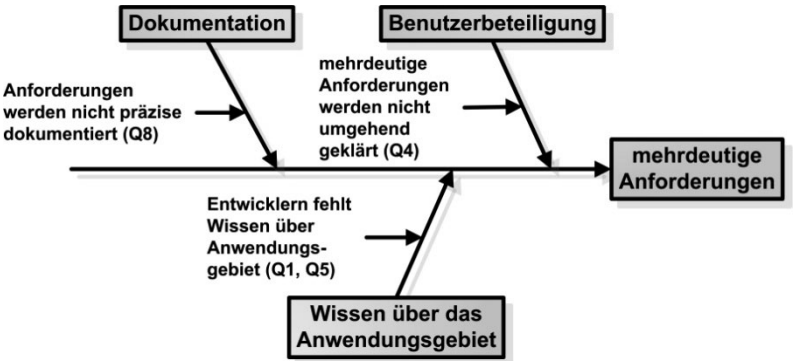


Abbildung 4: Ursachen für mehrdeutige Anforderungen

Die Ursachen für falsche Anforderungen liegen insbesondere in dem Gestaltungsbereichen der Dokumentation von Anforderungen, Wissensmonopolen bei wenigen Entwicklern, einer unzureichenden Auseinandersetzung mit Kundenbedürfnissen und der Vielzahl und Heterogenität der Kunden (vgl. Abbildung 5).

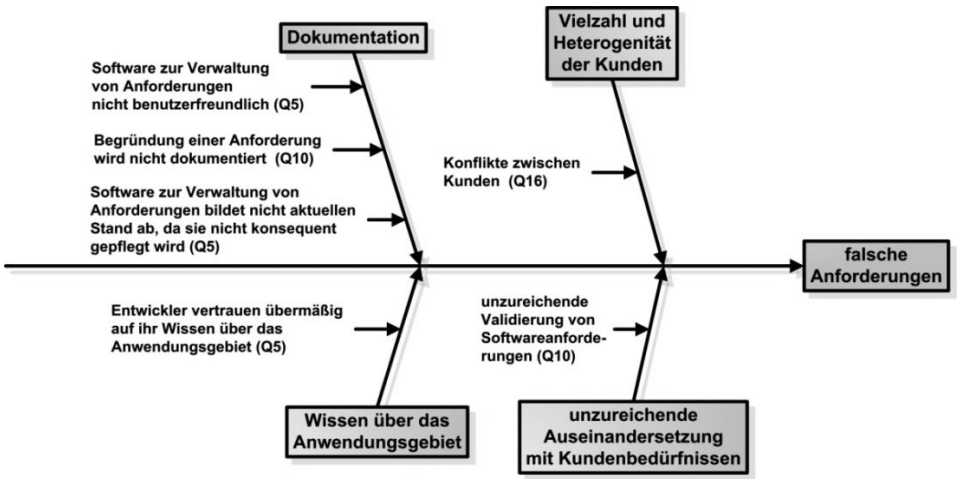


Abbildung 5: Ursachen für falsche Anforderungen

Die Ursachen für unvollständige Anforderungen liegen in der Art der Erhebung von Anforderungen, einem fehlenden Wissen über das Anwendungsgebiet sowie im Projektmanagement und der Gestaltung der Benutzerbeteiligung (vgl. Abbildung 6).

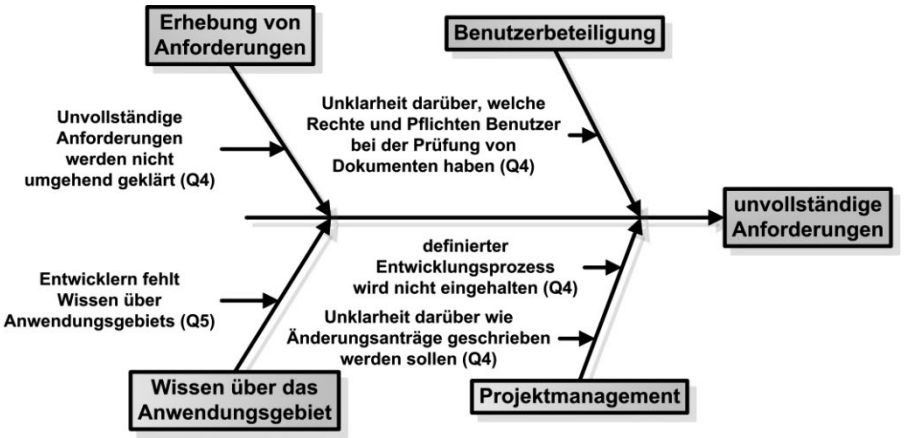


Abbildung 6: Ursachen für unvollständige Anforderungen

Die Ursachen für missverständliche Anforderungen liegen insbesondere in den Gestaltungsbereichen der Dokumentation von Anforderungen (hier insbesondere die Verfolgbarkeit) und der Benutzerbeteiligung. Als weitere Ursachen sind die Kommunikation im Entwicklungsteam, Form und Inhalte der Anforderungsdokumente, Projektmanagement sowie eingesetzte Software-Werkzeuge aufzuführen (vgl. Abbildung 7).

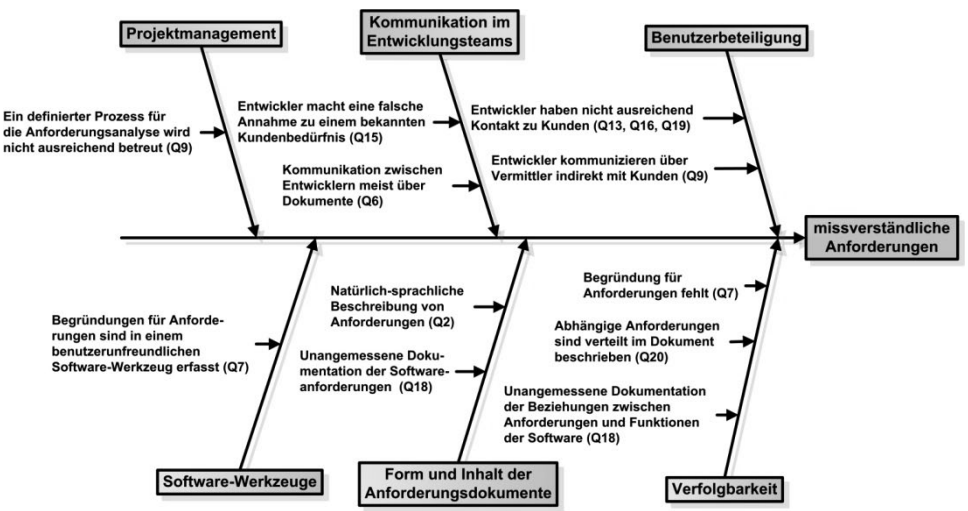


Abbildung 7: Ursachen für missverständliche Anforderungen

Die Ursachen für inkonsistente Anforderungen liegen in der Dokumentation von Anforderungen, der inhärenten Komplexität der Anforderungsanalyse, der Kommunikation im Entwicklungsteam und fehlendem Wissen über das Anwendungsgebiet (vgl. Abbildung 8).

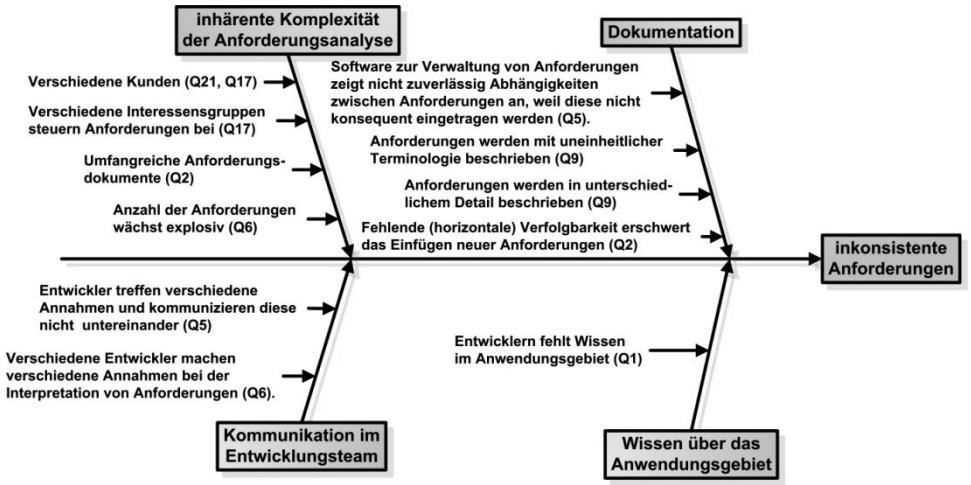


Abbildung 8: Ursachen für inkonsistente Anforderungen

Die Ursachen für redundante Anforderungen liegen in der Dokumentation von Anforderungen. Hier werden im Wesentlichen zwei konkrete Ursachen aufgeführt:

- Anforderungen werden in unterschiedlichem Detailgrad beschrieben.
- Anforderungen werden mit einer uneinheitlichen Terminologie beschrieben.

5 Diskussion der Ergebnisse

Missverständliche, inkonsistente und fehlende Anforderungen machen 67 % der Anforderungsfehler aus, deren Typ bekannt ist (53 von 92, ca. 58 %). Gemäß der systematischen Literaturanalyse tragen die Ursachenkategorien Dokumentation von Anforderungen, Gestaltung der Benutzerbeteiligung, Wissen über das Anwendungsgebiet und Erhebung von Anforderungen zu ca. 73 % der Fehler bei.

Nachfolgend werden zentrale Ergebnisse dieser vier Ursachenkategorien herausgestellt und mit relevanten empirischen Befunden in der Literatur verglichen.

5.1 Dokumentation von Anforderungen

Die identifizierten Ursachen für Anforderungsfehler innerhalb der Dokumentation von Anforderungen beziehen sich im Einzelnen auf die Form und den Inhalt von Anforderungsdokumenten, auf die eingesetzten Software-Werkzeuge und die Verfolgbarkeit von Anforderungen. Prozessmängel in diesem Bereich verursachen insbesondere missverständliche, inkonsistente, fehlende und falsche Anforderungen.

Bei der Gestaltung der Form von Anforderungsdokumenten geht es um die Frage, mit welchen Darstellungsmitteln Softwareanforderungen beschrieben werden sollen. Innerhalb dieses Gestaltungsbereichs ist insbesondere die Frage zu beantworten, ob Anforderungen natürlich-sprachlich oder semi-formal mit Methoden wie der UML beschrieben werden sollen. Einerseits kann eine natürlich-sprachliche Beschreibung gemäß der Literaturanalyse zu Anforderungsfehlern führen. Dies legt nahe, semi-formale Beschreibungsmethoden einzusetzen. Chatzoglou zeigt in einer Untersuchung, dass in Projekten mit Einsatz von Methoden der Aufwand, die Kosten und der Zeitbedarf für die gesamte Entwicklung geringer ausfallen als in Projekten ohne Einsatz von Methoden [Ch97].

Andererseits können jedoch Kunden Schwierigkeiten haben, semi-formale Beschreibungen von Anforderungen zu verstehen. Dieser Sachverhalt wird ebenfalls in weiteren empirischen Untersuchungen festgestellt [EM95], [DS99]. Dem Spannungsfeld zwischen natürlich-sprachlicher und semi-formaler Beschreibung begegnen Entwicklungsteams,

- indem sie semi-formale Beschreibungen von Anforderungen zusätzlich und redundant natürlich-sprachlich für Kunden erläutern,
- indem sie informale Beschreibungen von Anforderungen (überwiegend natürlich-sprachlich) für Kunden und davon getrennt für die Arbeit innerhalb des Entwicklungsteams zusätzlich semi-formale Beschreibungen für Anforderungen erstellen oder
- indem sie sich auf die natürlich-sprachliche Beschreibung beschränken.

Neben der Form von Anforderungsdokumenten gibt es mehrere Beobachtungen zu deren Inhalt. Werden Anforderungen aus technischer Perspektive beschrieben, kann dies zu Anforderungsfehlern führen. Die gleiche Wirkung haben Unklarheiten, in welchem Detailgrad die Anforderungen beschrieben werden sollen. Diese Beobachtungen zum Inhalt von Anforderungsdokumenten stehen im Einklang mit Ergebnissen einer Untersuchung von El Emam und Madhavji [EM95].

Wird ein Software-Werkzeug eingesetzt, um Anforderungen zu dokumentieren oder zu verwalten, kann sein Einsatz zu Anforderungsfehlern führen, sofern es Unzulänglichkeiten wie eine geringe Benutzbarkeit aufweist oder die Nutzung nicht geplant, gesteuert und überwacht wird.

Dies sind einige der Gründe, warum Entwicklungsteams entsprechende Werkzeuge eher als hinderlich wahrnehmen [HL01]. Empirische Befunde zeigen, dass der Einsatz von entsprechenden Software-Werkzeugen wie z. B. CASE-Werkzeugen nur dann effektiv ist, wenn zumindest ein Prozess der Anforderungsanalyse definiert ist, die Entwickler in dem Werkzeug geschult werden und die Unterstützung der Nutzung organisatorisch institutionalisiert wird [EM95].

Insgesamt 10 Beobachtungen der Literaturanalyse beziehen sich auf die Verfolgbarkeit von Anforderungen. Mit 6 Beobachtungen ist die vertikale Vorverfolgbarkeit am häufigsten vertreten. Sie bezieht sich auf Verknüpfungen zwischen einer Anforderung und abhängigen Einzelergebnissen vorgelagerter Teilaufgaben wie z. B. Kundenbedürfnissen. Gotel und Finkelstein kommen in einer Untersuchung zur Verfolgbarkeit von Anforderungen ebenfalls zum Ergebnis, dass Praktiker eine unzureichende vertikale Vorverfolgbarkeit von Anforderungen am häufigsten als Problem nennen [GF94].

5.2 Gestaltung der Benutzerbeteiligung

Die identifizierten Ursachen in der Benutzerbeteiligung sind vor allem eine unzureichende oder zu späte Benutzerbeteiligung oder eine Beteiligung von ungeeigneten Personen. Prozessmängel in der Benutzerbeteiligung verursachen insbesondere missverständliche Anforderungen.

Empirische Befunde einer Untersuchung von Weltz und Ortmann zeigen ebenfalls auf, dass die Weichen für die Benutzerbeteiligung häufig falsch gestellt werden [WO92]. Gemäß ihrer Untersuchung spielen sowohl Entwickler als auch Benutzer in der Formulierung von Anforderungen häufig nur eine marginale Rolle. Nach ihrer Ansicht wird ein Benutzervertreter eher deswegen bestimmt, um die tatsächlich unzulängliche Einbindung der Benutzer zu verdecken. Vergleichbare Beobachtungen machen auch El Emam und Madhavji [EM95]. Sie bemerken, dass gerade jene Benutzer, deren Beteiligung am sinnvollsten für ein Projekt ist, selten für entsprechende Aufgaben zur Verfügung stehen.

Weitere Beobachtungen aus der systematischen Literaturanalyse beschreiben Schwierigkeiten bei der interdisziplinären Zusammenarbeit zwischen Entwicklern und Benutzern sowie eine mangelnde Konsequenz bei der Umsetzung der Benutzerbeteiligung. In einer 1994 veröffentlichten Untersuchung zur Benutzerbeteiligung fasst Heinbokel zusammen, dass die „Benutzerbeteiligung in der gegenwärtig praktizierten Form weder zu ersichtlich besserer Software führt noch für den Entwicklungsprozess selbst förderlich ist, sondern im Gegenteil zunächst ein Hemmschuh für das Projekt ist“ [He94]. Heinbokel kommt zu dieser Schlussfolgerung, weil Projekte mit einer Benutzerbeteiligung beim Gesamterfolg, bei der Termin- und Kosteneinhaltung sowie bei der Teameffektivität schlechter abschneiden als Projekte ohne Benutzerbeteiligung.

5.3 Wissen über das Anwendungsgebiet

Insgesamt 12 Beobachtungen weisen darauf hin, dass die Entstehung von Anforderungsfehlern begünstigt wird, wenn dem Entwicklungsteam Wissen über das Anwendungsgebiet fehlt. Prozessmängel in diesem Bereich verursachen gemäß der Literaturanalyse abgesehen von redundanten Anforderungen alle Typen von Anforderungsfehlern.

Tiwana kommt auf der Grundlage einer Untersuchung von 232 Entwicklungsprojekten in verschiedenen Organisationen zum Ergebnis, dass die Integration von technischem Wissen und Wissen über das Anwendungsgebiet der Software unter anderem die Fehlerhäufigkeit in den einzelnen Teilaufgaben der Softwareentwicklung reduziert [Ti04].

5.4 Erhebung von Anforderungen

Die Erhebung von Anforderungen verursacht gemäß der Literaturanalyse 13 % der Anforderungsfehler. Insbesondere treten unklare Ziele der Software und die unzureichende Auseinandersetzung mit Kundenbedürfnissen als Ursachen auf. Prozessmängel in der Erhebung von Anforderungen verursachen insbesondere fehlende Anforderungen.

Eine Fallstudie von Cusumano und Selby bei Microsoft verdeutlicht, wie wichtig eine klare Vision und klare Ziele für die Entwicklung einer Software sind [CS97]. Microsoft nutzt eine Visionsaussage und eine vorläufige Spezifikation als Projektleitfaden. Statt von Anfang an eine detaillierte Spezifikation zu schreiben, beginnt Microsoft die Projekte mit einer ehrgeizigen Visionsaussage und einer vorläufigen Spezifikation. Die Visionsaussage wird von den Produktplanern der Marketinggruppe und den Programm-Managern verfasst und listet eine Reihe von Zielen auf. Sie enthält keine detaillierten Anforderungen, sondern wird vielmehr als ein Ausgangspunkt genutzt, um systematisch Anforderungen herzuleiten.

Fünf Beobachtungen beschreiben eine unzureichende Auseinandersetzung mit Kundenbedürfnissen als Ursachen für Anforderungsfehler. Dieser Umstand rührt unter anderem daher, dass Entwicklungsteams Methoden zur systematischen Erhebung von Anforderungen wie z. B. Quality Function Deployment (QFD) nicht kennen oder nicht einsetzen [Ha96]. Vielmehr existiert häufig eine unberechtigte Erwartung, dass Methoden zur Beschreibung von Anforderungen zugleich auch zu ihrer Erhebung dienen [Lu93]. Mehrere Untersuchungen zeigen, dass erfolgreiche Teams konsequent Methoden zur Erhebung von Anforderungen wie z. B. QFD einsetzen [Ha96], [HL01].

6 Zusammenfassung und Fazit

Der vorliegende Beitrag beschreibt kausale Zusammenhänge zwischen Prozessmängeln in der Anforderungsanalyse und Typen von Anforderungsfehlern. Diese Erklärungsmuster sind Ergebnis einer Synthese empirischer Befunde. Sie bieten bereits praktische Gestaltungshilfe, da sie ausgehend von Anforderungsfehlern eine Ursache-Wirkungsanalyse unterstützen und z. B. in Fehleranalyseverfahren integriert werden können [Av08]. Mit der Kenntnis der Prozessmängel in der Anforderungsanalyse eröffnet sich die Möglichkeit, diese zu beseitigen oder, sofern dies nicht möglich ist, die Prozesse der Qualitätssicherung zu verbessern, um gezielt Fehler der Prozessmängel früher zu finden.

Literaturverzeichnis

[AE96] Al-Rawas, A., Easterbrook, S.: Communication problems in requirements engineering. A field study. In: Myers, C.; Hall, T.; Pitt, D. (Hrsg.): Proceedings of the First Westminster Conference on Professional Awareness in Software Engineering (PASE 1996), 1-2 February 1996, London (England). London (England) 1996, S. 47-60.

[AS85] Adelson, B., Soloway, E.: The role of domain experience in software design. In: IEEE Transactions on Software Engineering. Nr. 11, Jg. 11, 1985, S. 1351-1360.

[Av08] Avci, O.: Aus Fehlern in der Softwareentwicklung lernen. Anforderungsanalyse und Qualitätssicherung mit Fehleranalysen verbessern. Saarbrücken 2008. VDM Verlag Dr. Müller, ISBN-10: 3836476878, ISBN-13: 9783836476874.

[AW05] Avci, O.; Wagner, H.: Das chronische Problem der Anforderungsanalyse und die Frage: Fehler vermeiden oder früh entdecken? In: Cremers, A.B.; Manthey, R.; Martini, P.; Steinhage, V. (eds.): Informatik 2005 - Informatik LIVE, Band 2, Lecture Notes in Informatics (LNI), Vol. P-68. Bonn 2005, S. 279-283.

[Bh94] Bhandari, I. et al.: In-process improvement through defect data interpretation. In: IBM Systems Journal. Nr. 1, Jg. 33, 1994, S. 182-214.

[BP84] Basili,V.; Perricone, B.T.: Software errors and complexity: an empirical investigation. In: Communications of the ACM. Nr. 1, Jg. 27, 1984, S. 42-52.

[Br94] Briand, L.C. et al.: A change analysis process to characterize software maintenance projects. In: Müller, H.A.; Georges, M. (Hrsg.): Proceedings of the International Conference on Software Maintenance (ICSM 1994), Victoria, BC, Canada, September 1994. Washington, DC, USA 1994, S. 38-49.

[Ch96] Chillarege, R.: Orthogonal Defect Classification. In: Lyu, M.R. (Hrsg.): Handbook of software reliability engineering. Los Alamitos, California u. a. 1996, S. 359-400.

[Ch97] Chatzoglou, P.D.: Use of methodologies: an empirical analysis of their impact on the economics of the development process. In: European Journal of Information Systems. Nr. 4, Jg. 6, 1997, S. 256-270.

[CK98] Crowston, K.; Kammerer, E.E.: Coordination and collective mind in software requirements development. In: IBM Systems Journal. Nr. 2, Jg. 37, 1998, S. 227-246.

[CS97] Cusumano, M.A.; Selby, R.W.: Die Microsoft-Methode. Sieben Prinzipien, wie man ein Unternehmen an die Weltspitze bringt. München 1997.

[Cu88] Curtis, B. et al.: A field study of the software design process for large systems. In: Communications of the ACM. Nr. 11, Jg. 31, 1988, S. 1268-1287.

[Da05] Damian, D. et al.: Requirements engineering and downstream software development: findings from a case study. In: Empirical Software Engineering. Nr. 3, Jg. 10, 2005, S. 255-283.

[Da04a] Damian, D. et al.: An industrial case study of immediate benefits of requirements engineering process improvement at the Australian Center for Unisys Software. In: Empirical Software Engineering. Nr. 1-2, Jg. 9, 2004, S. 45-75.

[Da04b] Daneva, M.: ERP requirements engineering practice: lessons learned. In: IEEE Software. Nr. 2, Jg. 21, 2004, S. 26-33.

[DS99] Dawson, L.; Swatman, P.: The use of object-oriented models in requirements engineering: a field study. In: ACM (Hrsg.): Proceeding of the 20th international conference on information systems (ICIS ‘99), December 12 - 15, 1999, Charlotte (North Carolina, USA). Atlanta (Georgia, USA) 1999, S. 260-273.

[DZ03] Damian, D.E.; Zowghi, D.: Requirements engineering challenges in multi-site software development organisations. In: Requirements Engineering. Nr. 3, Jg. 8, 2003, S. 149-160.

[ED05] Ebert, C.; De Man, J.: Requirements uncertainty. Influencing factors and concrete improvements. In: ACM (Hrsg.): Proceedings of the 27th International Conference on Software Engineering (ICSE 2005), May 15 - 21, 2005, St. Louis (Missouri, USA). New York (New York, USA) 2005, S. 553-560

[EM95] El Emam, K.; Madhavji, N.Z.: A field study of requirements engineering practices in information systems development. In: IEEE (Hrsg.): Proceedings of the Second IEEE International Symposium on Requirements Engineering (RE '95), March 27 - 29, 1995, York (England). Los Alamitos (California, USA) 1995, S. 68-80.

[GF94] Gotel, O.C.Z.; Finkelstein, A.C.W.: An analysis of the requirements traceability Problem. In: IEEE (Hrsg.): Proceedings of the First International Conference On Requirements Engineering (ICRE 1994), April 18-22, 1994, Colorado Springs, Colorado. Los Alamitos (USA) 1994, S. 94-101.

[Gü96] Günther, H. et al.: Kontinuierliche Qualitätsverbesserung in der Software-Entwicklung. Erfahrungen bei der Allianz Lebensversicherungs-AG. In: Wirtschaftsinformatik. Nr. 2, Jg. 38, 1996, S. 160-171.

[Ha96] Haag, S. et al.: Quality Function Deployment usage in software development. In: Communications of the ACM. Nr. 1, Jg. 39, 1996, S. 41-49.

[He94] Heinbokel, T.: Benutzerbeteiligung: Schlüssel zum Erfolg oder Hemmschuh der Entwicklung? In: Brodbeck, F.C., Frese, M. (Hrsg.): Produktivität und Qualität in Softwareprojekten. Psychologische Analyse und Optimierung von Arbeitsprozessen in der Softwareentwicklung. München, Oldenbourg 1994, S. 105-124.

[HL01] Hofmann, H.F.; Lehner, F.: Requirements Engineering as a success factor in software projects. In: IEEE Software. Nr. 4, Jg. 18, 2001, S. 58-66.

[Ie90] IEEE Standard Glossary of Software Engineering Terminology. New York 1990.

[Ka02] Kauppinen, M. et al.: Introducing requirements engineering: how to make a cultural change happen in practice. In: IEEE (Hrsg.): Proceedings of 10th Anniversary IEEE Joint International Conference on Requirements Engineering (ICRE 2002), September 9-13 2002, Essen (Germany).). Los Alamitos (California, USA) 2002, S. 43-51.

[Lu93] Lubars, M. et al.: A review of the state of the practice in requirements modeling. In: IEEE (Hrsg.): Proceedings of IEEE International Symposium on Requirements Engineering (ISRE 1993), January 4-6, 1993, San Diego (California, USA). Los Alamitos (California, USA) 1993, S. 2-14.

[LV01] Lauesen, S.; Vinter, O.: Preventing requirement defects: An experiment in process improvement. In: Requirements Engineering. Nr. 1, Jg. 6, 2001, S. 37-50.

[Mo98] Morris, P. et al.: Requirements Engineering and Industrial Uptake. In: IEEE (Hrsg.): Proceedings of the Third IEEE International Conference on Requirements Engineering (ICRE '98), April 06 - 10, 1998, Colorado Springs, Colorado. Los Alamitos (California, USA) 1998, S. 130-137.

[NK91] Nakajo, T.; Kume, H.: A case history analysis of software error cause-effect relationships. In: IEEE Transactions on Software Engineering. Nr. 8, Jg. 17, 1991, S. 830-838.

[Su99] Sutcliffe, A.G. et al.: Tracing requirements errors to problems in the requirements engineering process. In: Requirements Engineering. Nr. 3, Jg. 4, 1999, S. 134-151.

[Ta95] Takahashi, K. et al.: A comparative study of structured and text-oriented analysis and design methodologies. In: Journal of Systems and Software. Nr. 1, Jg. 28, 1995, S. 69-75.

[Ti04] Tiwana, A.: An empirical study of the effect of knowledge integration on software development performance. In: Information and Software Technology. Nr. 13, Jg. 46, 2004, S. 899-906.

[Wa93] Walz, D.B. et al.: Inside a software design team: knowledge acquisition, sharing, and integration. In: Communications of the ACM. Nr. 10, Jg. 36, 1993, S. 63-77.

[WO92] Weltz, F.; Ortmann, R. G.: Das Softwareprojekt. Projektmanagement in der Praxis. Frankfurt 1992.

[WW02] Webster, J.; Watson, R.T.: Analyzing the past to prepare for the future: writing a literature review. In: MIS Quarterly. Nr. 2, Jg. 26, 2002, S. xiii-xxiii.

Rollenveränderung in der Agilen Software-Entwicklung: Das „Projektmanagement-Labor“ der BA Lörrach

Eckhart Hanser

Studiengang Angewandte Informatik
Berufsakademie (BA) Lörrach – University of Cooperative Education
Hangstraße 46-50
79539 Lörrach
hanser@ba-loerrach.de

Abstract: Ausgehend von der Historie der Industrialisierung der Software-Entwicklung und den sich verändernden Anforderungen an modernes Software-Engineering gibt dieser Artikel einen Einblick in die wohl bekanntesten Agilen Programmiertechniken, Extreme Programming (XP) und Crystal Clear. Anhand praktischer Erfahrungen, die im IT-Labor der Berufsakademie Lörrach gesammelt wurden, werden einige der wichtigsten XP-Regeln u.a. Pair Programming hinterfragt und widerlegt. Am Beispiel von Crystal Clear wird gezeigt, wie der gelebte Prozess vom gewählten Prozessmodell abweicht. Ausgehend von den Ergebnissen des IT-Labors werden die wichtigsten "Zutaten" formuliert, die für eine erfolgreiche Software-Entwicklung im Team notwendig sind.

1 Einleitung: Kurze Historie der Software-Entwicklung

In den 70er und 80er-Jahren des vergangenen Jahrhunderts war die Welt der Software-Entwickler noch „in Ordnung“. Nach einem Prozess der Industrialisierung der Software-Entwicklung stand am Anfang eines Projekts eine ausführliche schriftliche Spezifikation, und man konnte davon ausgehen, dass sich diese selbst bei mehrjährigen Projekten bis zum Schluss kaum änderte. Man entwickelte gemäß dem sog. Wasserfall-Modell, was eine Projektentwicklung von der Spezifikation zum Design, und dann weiter über die eigentliche Programmierung (Implementation) bis zum Test zuließ – ähnlich den Kaskaden eines Wasserfalls.

Das Umfeld von Software-Projekten veränderte sich in den 90er-Jahren rasant. Projekte wurden immer kurzlebiger, die Anforderungen änderten sich immer mehr schon während der Projektlaufzeit und oftmals konnte der Kunde am Anfang des Projekts noch keine detaillierten Aussagen zum Umfang des Gesamtprojekts machen. Gleichzeitig wurde die durchschnittliche Entwicklungs-Teamgröße immer kleiner. Diese Entwicklung hatte einen Höhepunkt im New Economy Hype, der in einem Buch von Lothar Späth [Sp01] als „Revolution“ gepriesen und mit der rasanten Entwicklung des Internets gleichgesetzt wurde.

Besonders Software-Entwickler, die nicht in Nischen wie der militärischen oder pharmazeutischen Entwicklung arbeiteten, waren folglich gezwungen, das bis dahin verwendete „industrielle“ Wasserfall-Prozessmodell durch geeignetere Software-Entwicklungsmodelle zu ersetzen. Ansätze, die für Furore in der Szene gesorgt haben, sind das 1996 von Kent Beck entwickelte Extreme Programming, sowie Crystal von Alistair Cockburn (2004), so genannte „Agile“ Prozessmodelle.

2 Das Agile Manifest

Diese neuen Prozessmodelle sind die Reaktion auf die so genannten „schwergewichtigen“ Prozess- bzw. Vorgehensmodelle, wie den Unified Process (UP) [JBR99] von Rational (jetzt IBM) oder das V-Modell [Ra06], das Vorgehensmodell der deutschen Bundesbehörden. Obwohl beide Ansätze sehr unterschiedlich sind – UP beleuchtet den iterativ-inkrementellen Software-Entwicklungsprozess, während das V-Modell die Sicht des Kunden auf das Software-Projekt in den Vordergrund stellt – gelten beide als sehr „dokumentenlastig“, was zumindest bei „Anfängern“ erheblichen zusätzlichen Aufwand bedeutet.

Branchen wie etwa die militärische oder pharmazeutische Software-Entwicklung, in denen Gefahr für Leib und Leben bestehen kann, können nicht auf ausführliche Dokumentation verzichten. Das verlangen zu Recht die zuständigen Behörden. Es handelt sich jedoch um eng abgegrenzte Felder. In der Mehrzahl der Software-Projekte jedoch, sollte die Entwicklung funktionierender Software im Mittelpunkt stehen, wie Kritiker wie Ward Cunningham, Martin Fowler, Alistair Cockburn oder eben Kent Beck betonen. Dies führte 2001 zur Verkündung des „Manifesto for Agile Software Development“, dem Agilen Manifest [Am01], welches die Vorgenannten und 13 andere unterzeichneten. Leitsätze des mittlerweile weltweit unterstützten Agilen Manifests sind:

- Die individuellen Beteiligten und ihre Interaktion sind wichtiger als Prozesse und Werkzeuge.
- Die Ablieferung lauffähiger Software ist wichtiger als umfassende Dokumentation.
- Die Zusammenarbeit mit dem Kunden ist wichtiger als Vertragsverhandlungen.
- Auf Änderungen der Anforderungen einzugehen ist wichtiger als die sture Verfolgung eines Plans.

3 Extreme Programming (XP)

Kent Beck entwickelte sein Prozessmodell *Extreme Programming* [Be00] 1996 im Rahmen des Chrysler-Projekts *C3 payroll*, das die Entwicklung einer Lohn- und Gehaltslisten-Software zum Ziel hatte. Das Projekt war zum Zeitpunkt des Eintretens von Beck bereits so gut wie gescheitert. Man war also bereit, auf neue Ansätze zu hören und gab somit Beck die Möglichkeit, die zuerst informelle Praxis als neues Prozessmodell zu formulieren und das Smalltalk-Projekt neu zu starten. Nach Becks eigener Aussage war das Projekt ideal für seinen neuen Ansatz,

- weil sich die Spezifikationen schnell änderten,
- weil der Kunde keine endgültige Vorstellung vom Funktionsumfang hatte,
- weil eine hohe Produktivität der Programmierer gefordert war,
- weil das Projekt ein hohes (auch zeitliches) Risiko in sich trug,
- weil die Projektgruppe inkl. Managern, Programmierern und Kunden eine Stärke von weniger als 12 Mitarbeitern hatte
- und weil die zu erstellende Software automatisiertes Testen erlaubte.

In der ersten Phase war C3 sehr erfolgreich und wurde 1997 produktiv. Somit wurde C3 zum „Flaggschiff“ für XP.

Im Gegensatz zu anderen Verfechtern des Agilen Manifests, wie z. B. Alistair Cockburn, dem Begründer des Crystal-Prozessmodells [Co04], schreibt Kent Beck die Vorgehensweisen im Projekt strikt vor. Er definiert „Rules and Practices“ zu Planung, Design, Kodierung und Testen, auf deren Einhaltung er besteht. Über allen Regeln stehen die 4 Schlüsselbegriffe: *Kommunikation, Einfachheit, Feedback und Mut*. Kommunikation zwischen den Team-Mitgliedern, aber auch mit dem Kunden soll ausführliche Dokumentation ersetzen. Ein einfaches Design soll die Fehleranfälligkeit des Systems reduzieren, Feedback speziell des Kunden soll das Projekt in die richtige Richtung treiben. Das Team soll aber auch den Mut haben, XP abzuändern, wenn gewisse Aspekte im speziellen Projekt nicht funktionieren („Fix XP when it breaks.“).

Es fällt auf, dass diese Regeln und Verhaltensweisen meist nicht neu sind. Es ist eher ihre Kombination und ihre strikte Einhaltung, was XP ausmacht. Man kann die XP-Regeln durchaus in 3 Kategorien einteilen: Regeln, die jeder moderne Projektmanager sofort akzeptiert und meist auch schon einsetzt, Regeln, die einleuchten als Alternative zu eigenen Vorgehensweisen und Regeln, die Widerspruch hervorrufen:

Kein erfahrener und zeitgemäßer Projektleiter bezweifelt die Wichtigkeit einer vernünftigen Versions- und Release-Planung, kleiner Releases, einer iterativ-inkrementellen Vorgehensweise und eines einfachen Designs mit Namenskonventionen. Ebenso gehört heutzutage neben Prototyping ein ausführliches und automatisiertes Testen „ab dem ersten Tag“ bereits auf Modul-Ebene zu den modernen Software-Projekttechniken. Ein erfahrener Projektleiter misst die „Geschwindigkeit“ seines Projekts und achtet darauf, dass die Mitarbeiter möglichst keine Überstunden machen (müssen).

Jeder Projektleiter wird „Stand-Up Meetings“ also kurze morgendliche Sitzungen im Stehen als effizientes Kommunikationsmittel schätzen lernen. Ebenso erhöht ein häufiges Verbessern des Designs, ein sog. *Refactoring*, die Qualität der Software und ist mit entsprechender Tool-Unterstützung machbar (*Computer Aided Software Engineering, CASE* [Ba98]). Schließlich gibt Kent Beck die ausdrückliche Erlaubnis, XP anzupassen, wenn das Projekt es erfordert. Dies gibt dem erfahrenen Projektleiter die Sicherheit, den Prozess den praktischen Anforderungen unterordnen zu können und nicht umgekehrt.

Die dritte Gruppe der XP-Regeln gibt mehr Anlass zu Diskussionen, wie der Autor aus eigener Erfahrung weiß, und soll in diesem Artikel näher beleuchtet werden:

(1) Beck fordert die permanente Mitgliedschaft des *Kunden im Team*. Dies soll eine schnelle Reaktion auf sich ändernde Kundenwünsche ermöglichen, selbst in späten Projektphasen. Spezifikationsdetails können „face to face“ mit dem Kunden besprochen werden. Die geforderte dauernde Anwesenheit des Kunden im Projekt ist in der Praxis naturgemäß ein Problem, da erfahrene Kunden kaum vollständig einem Software-Projekt zur Verfügung stehen. Ein anderer Einwand ist, dass der Kunde auf diese Weise zu viele Interna des Projekts (auch gemachte Fehler!) mitbekommen könnte. Würde dadurch nicht sein Vertrauen in die Qualität der Entwickler-Mannschaft sinken? Und wird dadurch die Projekt-Kommunikation insbesondere mit dem Kunden wirklich besser?

(2) *User Stories*: Die Anforderungen sollen nicht in Form von Dokumenten, sondern in Form von kurzen, dreisätzigen Mini-Spezifikationen vom Kunden in seiner „Sprache“ abgefasst werden. Der Aufwand für die Implementierung des zugehörigen Software-Moduls soll 3 Wochen nicht überschreiten. Kann der Kunde das? Wie kann er 3 Wochen Aufwand abschätzen? Macht eine solche zeitlich feine Gliederung überhaupt Sinn? Braucht er Unterstützung (von wem)?

Schließlich die „berühmtesten“ XP-Regeln:

(3) Programmieren in Paaren – *Pair Programming*: 2 Programmierer arbeiten zusammen an einem Computer. Beck verspricht eine Erhöhung der Software-Qualität bei gleicher Effizienz wie bei einem Team zweier Programmierer an 2 Computern. Im Paar soll ein Programmierer taktisch denken, sich also mit dem Funktionsumfang einer aktuellen Klassenmethode beschäftigen, während der andere strategisch denkt, sich also mit der Frage beschäftigt, ob obige Klassenmethode überhaupt in diese Klasse passt. Die Programmierer wechseln regelmäßig (ggf. alle paar Stunden) ihre Rollen. Diese Forderung wird wohl in den wenigsten Projekten erfüllt. Die Frage nach dem Sinn dieser XP-Regel soll im Weiteren untersucht werden.

(4) *Collective Code Ownership, move people around*: Jedes Team-Mitglied soll Verantwortung für den gesamten Code tragen und diesen auch entsprechend verstehen. Jeder, der eine Änderung am Code eines anderen Paares braucht, kann diese selbständig durchführen, da das automatisierte Testen mit einer entsprechenden Versionskontrolle es auf eine Art ermöglicht, den „Urzustand" der Software vor der Änderung wieder herbeizuführen. Dieser Punkt ist heikel. Jeder der schon einmal mit entsprechenden Tools gearbeitet hat, weiß, dass man hierzu ein tiefes Verständnis der zu ändernden Software haben muss oder die Bereitschaft zum Erlangen dieses Verständnis mitbringen muss. Auch diese Regel soll untersucht werden.

(5) *Integration in kurzen Abständen*, idealerweise kontinuierlich, ausgehend von den Paaren, die eigenverantwortlich nacheinander an einem Integrationsrechner die Projektmodule zu einem lauffähigen Release verknüpfen. Hier muss die Frage geklärt werden, ob alle Paare genügend Willen und Know-How mitbringen um eine in der Regel komplexe Integration erfolgreich durchzuführen.

Außerdem soll der Frage nachgegangen werden, ob die klassischen Rollen des Projektleiters und des Qualitätsmanagers in einer agilen Umgebung noch Sinn machen, bzw. wie sie sich ändern.

4 Das IT-Labor der Berufsakademie Lörrach

Das IT-Labor ist eine Veranstaltung im 6. Semester des IT-Studiums, in dem der jeweilige Kurs gemeinsam als Team ein Software-Projekt durchführt. Jeden Frühling wird an 11 Tagen zu jeweils 5 Kursstunden mit ca. 20 Team-Mitgliedern an diesem Projekt gearbeitet. Dies ergibt pro Semester effektiv ca. 800 Projekt-Stunden. Naturgemäß ergibt sich die Zahl der Team-Mitglieder aus der Zahl der Studierenden plus Dozent. Das IT-Labor besteht in dieser Form seit 2004. Der Autor übernimmt dabei als Dozent die Rolle des Kunden (und ab und zu des Beraters), die Rollen Projektleiter und Qualitätsmanager werden entweder von ihm bestimmt oder vom Team gewählt [Ha06].

Neben dem vordergründigen Ziel, eine den Spezifikationen entsprechende stabile Software zu entwickeln, werden jedoch auch tiefer gehende Fragen zur Projektorganisation beleuchtet:

(1) Gibt es XP Projektpraktiken, die nicht ohne Intervention eines (Projekt-) Managements funktionieren? Welche XP Praktiken sind kein Resultat der Selbstorganisation des Teams? Hier wird insbesondere die bereits erwähnte dritte Gruppe der XP-Regeln untersucht (s. III), die Anlass zu Diskussionen gibt.

(2) Praktische Erfahrung zeigt, dass Teams mit mehr als etwa 5 Mitgliedern sich in „Mini-Teams“ unterteilen:

- In XP wird ein solches Mini-Team „Paar“ genannt. Es besteht aus 2 Mitgliedern. Sind diese Paare in der Realität erfolgreich?
- Gibt es eine ideale Größe (größer 2) für erfolgreiche Mini-Team-Teams?
- Wie sieht die Mitgliederstruktur einen solchen (erfolgreichen) Mini-Teams aus?

Anlass zu dieser Fragenstellung geben neben den persönlichen Erfahrungen des Autors auch diverse Artikel in der Literatur. Martin Fowler weist beispielsweise daraufhin, dass das eingangs erwähnte Chrysler C3 Projekt, also das „Initialprojekt“ von Extreme Programming, nur in der ersten Phase bis 1997 ein Erfolg war. Im weiteren Projektverlauf geriet es immer mehr in Schwierigkeiten und wurde 1999 eingestellt. Somit ist bewiesen, dass auch XP kein Garant für einen erfolgreichen Projektverlauf ist [Fo05]. Auch Alistair Cockburn, dessen Crystal in der Basler Pharma eingesetzt wird, äußert sich kritisch. Bekannt ist in diesem Zusammenhang die XP-Karikatur auf seiner Web-Site, die 2 Programmierer vor einem Bildschirm zeigt und an Edvard Munchs Gemälde „Der Schrei“ erinnert [Co08].

5 Das Projekt

Im IT-Labor der BA Lörrach wird eine eMail-Marketing Software entwickelt. Aus einer unterliegenden Datenbank werden Newsletter-Inhalte ausgelesen und personalisiert an Abonnenten verschickt, die sich zuvor für den entsprechenden Newsletter angemeldet haben. Die Software verfügt über eine web-basierte grafische Benutzeroberfläche und ein Administrationsinterface. Die Abbildungen 1 und 2 zeigen zwei Eingabemasken des Newsletter-Systems. Das System ist mittlerweile nach ca. 3.000 Projektstunden funktionsfähig und mit Hilfe des Mono-Projekts auf Linux-Server portiert. Es soll aber auch funktionell noch erweitert werden. Ein „Inbound“-Handling, also die automatische Behandlung eintreffender Mails fehlt beispielsweise noch völlig. Es bleibt also noch genug zu tun für zukünftige Studenten.

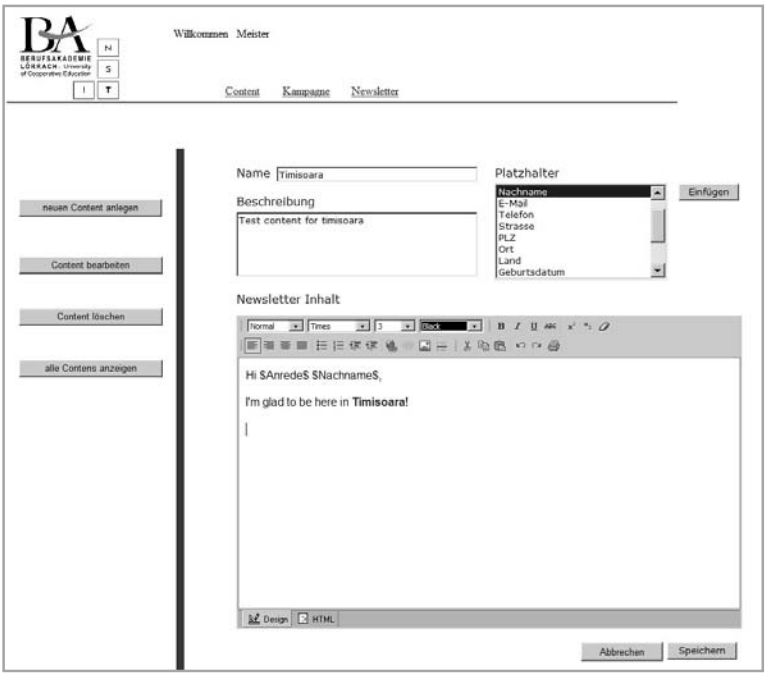
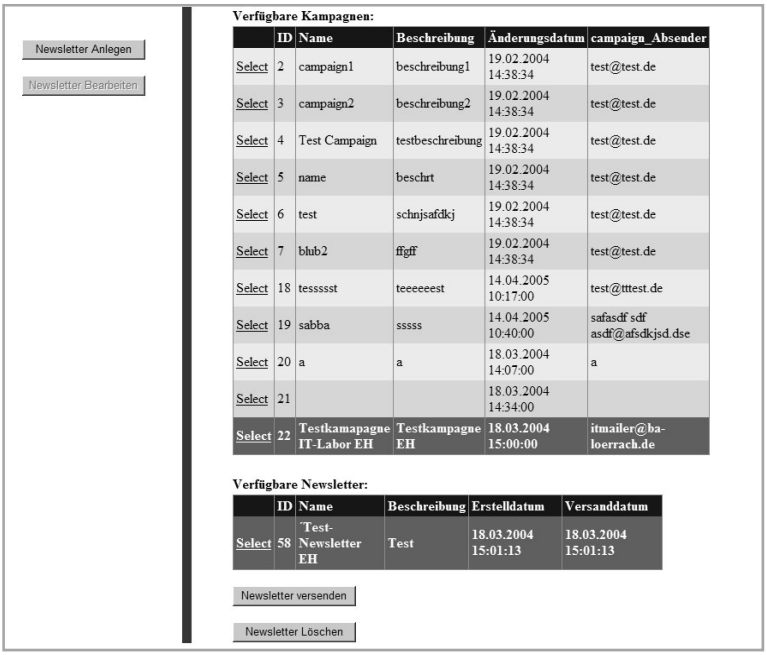


Abbildung 1: Newsletter-Editor



	ID	Name	Beschreibung	Änderungsdatum	campaign_Absender
Select	2	campaign1	beschreibung1	19.02.2004 14:38:34	test@test.de
Select	3	campaign2	beschreibung2	19.02.2004 14:38:34	test@test.de
Select	4	Test Campaign	testbeschreibung	19.02.2004 14:38:34	test@test.de
Select	5	name	beschrt	19.02.2004 14:38:34	test@test.de
Select	6	test	schnjsafdlkj	19.02.2004 14:38:34	test@test.de
Select	7	blub2	ffgff	19.02.2004 14:38:34	test@test.de
Select	18	tesssssst	teeeeeest	14.04.2005 10:17:00	test@tttest.de
Select	19	sabba	sssss	14.04.2005 10:40:00	safasdf sdf asdf@afsdkjsd.dse
Select	20	a	a	18.03.2004 14:07:00	a
Select	21			18.03.2004 14:34:00	
Select	**22**	**Testkamapagne IT-Labor EH**	**Testkampagne EH**	**18.03.2004 15:00:00**	**itmailer@ba-loerrach.de**

	ID	Name	Beschreibung	Erstelldatum	Versanddatum
Select	**58**	**Test-Newsletter EH**	**Test**	**18.03.2004 15:01:13**	**18.03.2004 15:01:13**

Abbildung 2: Newsletter-Manager

6 Randbedingungen 2004 vs. 2005

Jede Session des IT-Labors wird vom Dozenten unter gewisse Randbedingungen gestellt. Somit können gezielt Fragestellungen beleuchtet werden, die sich aus dem Umgang mit Agilen Prozessmodellen ergeben. Diese Randbedingungen ergeben sich auch aus den Ergebnissen des vorigen Jahres. In der folgenden Tabelle sollen die Randbedingungen der Jahre 2004 und 2005 gegenübergestellt werden:

IT-Labor der BA Lörrach	**Session 2004**	**Session 2005**
Vorbedingungen	Der Dozent besetzt die Positionen Projektmanager und Qualitätsmanager mit geeigneten Studierenden.	Der Dozent besetzt die Positionen Projektmanager und Qualitätsmanager mit geeigneten Studierenden.
	Das restliche Team (Kurs) wird anfangs in Dreiergruppen (Mini-Teams) eingeteilt.	Team (Kurs) ist anfangs unstrukturiert, erhält aber den Auftrag, Mini-Teams in Form von XP-Paaren zu bilden.
	Die Teammitglieder haben Hintergrundwissen zum Thema Prozessmodelle, speziell zum Thema Agile Programmierung.	Die Teammitglieder haben Hintergrundwissen zum Thema Prozessmodelle, speziell zu Extreme Programming (XP).
	Der Kunde (der Dozent) ist Teammitglied.	Der Kunde (der Dozent) ist Teammitglied.
Entwicklungsplattform	Microsoft Visual Studio .NET 2003	Microsoft Visual Studio .NET 2003
Projektziel	Erfolgreiches Produkt, erfolgreiches Projekt.	Erfolgreiches Produkt, erfolgreiches Projekt.
	Business Logik und Grafische Benutzeroberfläche sollen von konkurrierenden Mini-Teams jeweils in C# und Visual Basic entwickelt.	Gemeinsame Implementierung eines voll einsatzfähigen eMail-Marketing-Systems in C#.
	Das Team soll sich einen geeigneten **Agilen Prozess wählen**.	**XP wird vom Dozenten gefordert.**
Erfolg der Session	Nur die Minimalanforderungen an Produkt und Prozess werden erfüllt.	Am Ende ein erfolgreiches Produkt und ein erfolgreicher Prozess.

Tabelle 1: Randbedingungen 2004 vs. 2005

Wie sind die unterschiedlichen Erfolge der beiden Sessions erklärbar? Was sind die notwendigen „Zutaten“ für ein erfolgreiches Projekt? Natürlich spielt die Stärke der Team-Mitglieder eine große Rolle, aber in beiden Jahrgängen waren starke Programmierer. Der Schlüssel zum Erfolg muss also auch maßgeblich in der Organisation des Teams liegen. Um dies zu untermauern sollen zunächst die Rollen des Projektmanagers und des Qualitätsmanagers beleuchtet werden. Machen diese klassischen Projektrollen in einer agilen Umgebung noch Sinn, bzw. wie ändern sie sich?

7 Projekt- und Qualitäts-Manager in Session 2005

Der „klassische“ Projektmanager führt sein Team und überwacht den Projektfortschritt. Er/sie erstellt das Projekthandbuch, welches das Projekt beschreibt, und den Projektplan für Aufwand, Zeit und Ressourcen. Der Projektmanager stellt die Weichen für ein erfolgreiches Projekt. In einem agilen Projekt ist diese klassische Projektrolle jedoch nicht eindeutig definiert. Also wurde beschlossen, dass ein Student diese Rolle übernehmen sollte um seine Rolle im agilen Team neu zu definieren.

Der Projektmanager 2005 beschrieb seine Rolle wie folgt [HBM06]:

- Koordinator des Projekts,
- Problemlöser,
- Kommunikator,
- .NET-Experte,
- Co-Designer von User Stories (Anwendungsfällen in Spezifikationen, Abbildungen 3 und 4).

Abbildung 3: User Story Card Board

User Story Card: „Datenbank“

Kurzbeschreibung:

- Prüfen der vorhanden Datenbank
- Erweitern der Datenbank mit fehlenden Tabellen
- Aufzeichnen des Schemas der Datenbank auf ein Flipchartblatt

Personenanzahl: 2

Bearbeiter:	Status
Bensch, Kanaat	[illegible] Tabelle wurde modifiziert. Bitte neue Spalten beachten und bei Validierung die DB-Regel beachten!

Code Review: (Absprache mit QM, ob Review notwendig)

Bearbeiter:	Status

Abbildung 4: Details User Story Card

Die Rolle des Qualitätsmanagers, der im „klassischen“ Projekt die Projektqualität misst und verantwortlich ist für das Erstellen des Qualitätssicherungsplans, der Risikoanalyse und der Testpläne, ist im agilen Projekt nicht definiert. Wie im Fall des Projektmanagers wurde auch hier entschieden, dass ein Student die Rolle des Qualitätsmanagers übernehmen sollte um seine neue agile Rolle individuell zu definieren.

Der Qualitätsmanger 2005 beschrieb seine Rolle wie folgt [HBM06]:

- verantwortlich für Produkt- und Projektqualität,
    - definieren von Projektregeln (Abbildung 5),
    - durchführen von Reviews,
    - testen,
- Problemlöser,
- Kommunikator,
- .NET-Experte,
- Co-Designer von User Stories.

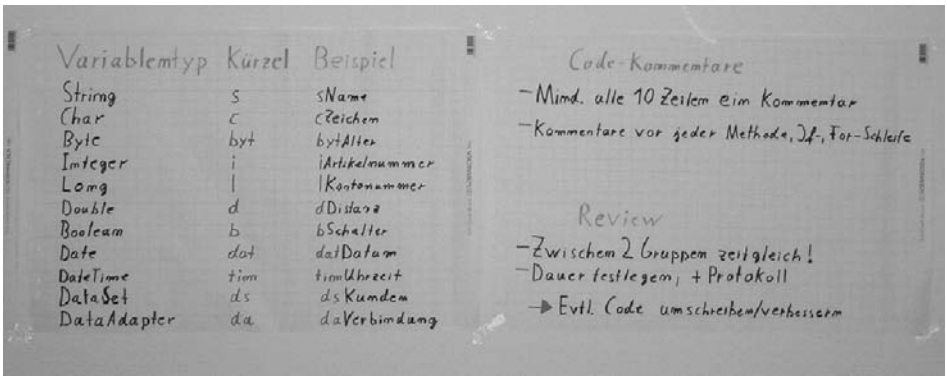


Abbildung 5: Projekt-Regeln

Es ist bemerkenswert, dass beide Studenten ihre neue Rolle sehr ähnlich definierten. Sie unterscheiden sich nur im ersten Punkt, also im „klassischen Teil" ihrer neuen agilen Projektrolle. Die meiste Zeit übten beide Studenten ähnliche Tätigkeiten aus: Sie brachten dem Team (agile) Projektregeln und unterstützten es bei der Durchführung, insbesondere bei der Kommunikation. Sie richteten ein Wiki[1] ein als zentrales, einfaches Archiv für Projektdokumentation. Sie überwachten die Team-Struktur und änderten gegebenenfalls die Struktur der XP-Paare. Im Rahmen des agilen Ansatzes „fixten" die Studenten XP zum ersten Mal und passten eine der XP-Praktiken an, in dem sie die Selbstorganisation der Paare beeinflussten.

8 Design von User Stories und collective code ownership

Es gibt andere XP-Praktiken, die angepasst und verbessert werden müssen. In beiden Sessions 2004 und 2005 findet das Erstellen der User Stories kein breites Interesse bei den Team-Mitgliedern. Der Kunde wird als Team-Mitglied wenig zur Kenntnis genommen. Lediglich Projekt- und Qualitätsmanager (in ihrem neuen Rollenverständnis) und einige wenige Team-Members suchen den Kontakt zum Kunden, um maximale Qualität und Verständnis der User Stories zu erreichen. Oftmals werden fertige User Stories „interpretiert" ohne nachzufragen. Erst wenn Entwickler-Mini-Teams Verantwortung für eine User Story übernehmen, werden sie aktiv und machen ein detailliertes Low-Level-Design.

[1] Ein Wiki (WikiWeb) ist eine im World Wide Web verfügbare Seitensammlung, die von den Benutzern nicht nur gelesen, sondern auch online geändert werden kann. Wikis ähneln damit Content Management Systemen, siehe http://de.wikipedia.org/wiki/Wiki.

Kollektive Verantwortung für den gesamten Code („Collective Code Ownership“) ist im Team nicht sehr beliebt. Außer Projekt- und Qualitätsmanager sind nur noch die wenigen Team-Mitglieder, die schon bei den User-Stories mitgewirkt haben, in der Lage, den gesamten Code zu verstehen – und nicht nur das vergleichsweise kleine Stück, das sie selbst entwickelt haben. Eine Bereitschaft für Collective Code Ownership (XP) ist beiden Sessions nicht erkennbar. Nur die Angst ernsthafte Fehler zu machen, die sich negativ auf das Produkt auswirken, veranlasst wenigstens die oben erwähnten Team-Mitglieder, die genannten XP-Praktiken „auszuprobieren“.

9 Der Weg zur erfolgreichen Software-Integration

Negative Erfahrungen aus der ersten Session des BA-IT-Labors, in der das Team seine Integrationsschritte selbst wählen konnte und dann erst am letzten (!) Tag integrierte, veranlassen den Kunden, in der Session 2005 Integrationszyklen von 3 Wochen zu fordern. Diese werden vom Team aber nicht selbständig eingehalten. Kontinuierliche oder wenigstens konstante Integrationen scheinen für das Team keine Bedeutung zu haben. Jede Minigruppe arbeitet an ihrer eigenen Fragestellung. Massive Intervention des Kunden und ein mehr „klassisches“ Rollenverständnis des Projektmanagers [HBM06] verbessern die Situation. Die Rolle des „Integrations Engineer“ wird geschaffen und mit einem sehr guten, (aber) gerne alleine arbeitenden, Software-Entwickler besetzt. Dieser „Lone Wolf“, der mit seinem Integrations-Rechner immer etwas Abstand zum restlichen Team hat, integriert sehr erfolgreich. Durch die Besetzung dieser Position wird nach ca. 5 Wochen eine kontinuierliche Integration möglich.

Beide Sessions des IT-Labors haben dasselbe Ergebnis: Integration ist nicht das Resultat einer Selbstorganisation des Teams, sondern das Ergebnis einer zentralen Planung des Projektmanagers. Nicht nur der Integrationsrechner muss einzig sein, sondern auch der Posten des Integration Engineer. Integration durch gleichberechtigte Paare, wie in Extreme Programming gefordert, hat im BA-IT-Labor bisher nicht zum Erfolg geführt.

10 Mini-Team-Größe: Sind XP-Paare erfolgreich?

Ein interessanter Parameter für ein Prozessmodell ist die Größe der „Untergruppen“ des Teams (Mini-Teams), in denen die einzelnen User Stories bearbeitet werden. In XP werden „Paare“ gefordert: 2 Entwickler sitzen am selben Rechner, ein Entwickler denkt strategisch, der andere operativ. Um die „Stabilität“ dieser Paare zu analysieren, werden in der Session 2004 zu Beginn Dreierteams gebildet, während in der Session 2005 reine XP-Paare an den Start gehen. Interessant ist die Frage nach der Veränderung der Mini-Teamgrößen über die Zeit (vgl. Tabelle 2).

Session 2004	Session 2005
6 Mini-Teams mit 3 Mitgliedern zum Start.	9 XP-Paare als initiale Mini-Teams.
Im Verlauf des Projekts bilden sich keine XP-Paare spontan.	Die Zahl der XP-Paare nimmt ab.
Die kleinste Teamgröße bleibt 3.	Die meisten Mini-Teams wachsen. Die Zahl der Mitglieder wächst auf 3, 4 und mehr.
Das erfolgreichste Mini-Team hat 4 Mitglieder („Super-Team“ 1).	In speziellen Phasen, z.B. in der Integration, gibt es Teamgrößen mit bis zu 6 Mitgliedern („Super-Team“ 2).

Tabelle 2: Veränderung der Mini-Teamgrößen über die Zeit

Ein paar Beispiele finden sich in den Abbildungen 6 und 7, aufgenommen im Abstand von 2 Wochen:

Abbildung 6: Drei Paare und ein Super-Team im Vordergrund

Abbildung 7: Die Zahl der Paare nimmt ab

Somit scheint es, dass XP-Paare „instabil“ sind. Die Größe der Mini-Teams ändert sich ständig, je nach Aufgabenstellung. Es gibt nach den Erfahrungen der Autoren keinen Grund für die Teamgröße 2, also für Paare. Bei übergreifenden Aufgaben, z. B. bei der Vorbereitung der Integration ist auch eine Sechsergruppe keine Seltenheit. Wenn sich Paare bilden, bestehen sie oft aus einem stärkeren und einem schwächeren Partner, wo der erste die Struktur der Lösung erarbeitet, während der andere die Details erledigt.

Die Organisationspsychologie kennt 6 Verhaltensweisen von Team-Mitgliedern, die alle zu einem erfolgreichen Team gehören müssen: Kommunikatoren, Ideengeber, Fachexperten und Teamworker, die für das Mini-Team wichtig sind, sowie Problemlöser und Qualitätsprüfer, die (zusätzlich) für das gesamte Projektteam von Bedeutung sind (siehe dazu Abbildung 8 sowie [BG01]). Die Mischung aus diesen verschiedenen Verhaltensweisen macht den Erfolg des Teams. „Klassische“ XP-Paare hingegen fordern eher ähnliche Typen, da beide Paar-Mitglieder sich abwechseln sollen im strategischen und taktischen Handeln.

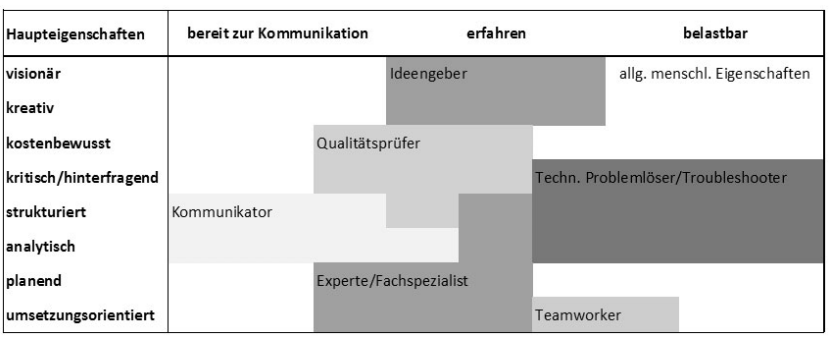


Abbildung 8: Verhaltensweisen im Team

Besonders unter Männern fördert diese Konstellation Rivalitäten, die wiederum den Erfolg des Paars gefährden. Im Rahmen des BA-IT-Labors wurden gut funktionierende, gleichberechtigte XP-Paare nur unter den Studentinnen gefunden!

Mini-Teams, in denen alle oben genannten Projekttypen vertreten sind, und die doch nicht zu groß werden (also vielleicht 4 bis 6 Mitglieder haben), gelten in der Organisationspsychologie als erfolgreich. Dies kann im Rahmen des BA-IT-Labors am Beispiel des Super-Teams aus Abbildung 6 bestätigt werden. Insbesondere dessen erfolgreiches Zusammenführen der Teilaktivitäten im Vorfeld der ersten Integration in der Session 2005 hat den Erfolg dieser Session erst möglich gemacht. Trotzdem ist die Existenz eines Integration Engineers, des „Lone Wolf" unabdingbar, da diese Aufgabe nur zentral bewältigt werden kann.

11 Abweichung des gelebten Prozess vom gewählten Modell

Sowohl die Session 2004 wie auch insbesondere 2005 zeigen, dass der „gelebte" Prozess massiv vom gewählten Prozessmodell abweicht:

2004 gingen die vorgegebenen agilen Dreier-Teams in eine größere Einheit auf. 2005 konnten die XP-Paare nicht „überleben" und wurden ebenfalls durch ein situationsbedingtes größeres Mini-Team „verdrängt" (Abbildung 5 und 6). Außerdem konnte nachgewiesen werden, dass weitere XP-Praktiken, wie das Erstellen von User Stories oder „Collective Code Ownership" in der Praxis nicht gemäß der XP-Regeln, sondern nur stark abgewandelt oder gar nicht umgesetzt wurden. Für die erfolgreiche Software-Integration musste sogar die neue Projektrolle eines „Integrations-Ingenieurs" geschaffen werden.

Diese Erfahrungen wurden in den Sessions 2006 und 2007 bestätigt: 2006 bildete sich statt des vorgegeben XP aufgrund der dominierenden Person des (vom Team gewählten!) Projektmanagers ein streng hierarchischer Prozess mit vielen kleineren Teilprozessen, der aber ebenfalls ein erfolgreiches Produkt erzeugte. Ein Entwicklungsprozess muss eben nicht agil sein, um Erfolg zu haben! 2006 wich der gelebte Prozess also vollständig vom vorgegebenen Prozess ab. Das Team empfand das nicht als Nachteil. Dies ist wieder ein Indiz dafür, dass die Verhaltensweisen der Team-Mitglieder über den (insbesondere von der „Linie") vorgegebenen Prozess dominieren.

12 Crystal Clear – Prozess-Anpassungen durch das Projekt-Team

2007 war ähnlich wie 2005 eine besondere Session: Hier konnte ein Student gewonnen werden, den Prozess des Teams im Rahmen seiner Studienarbeit während des ganzen Semesters zu begleiten. Er wurde zum „Prozessverantwortlichen“ ernannt [Hn07].

Aufgabe des Projekt-Teams war, die bestehende eMail-Marketing Software mit Hilfe des Mono-Projekts [Mp08] auf Linux zu portieren. Als Prozessmodell wurde vom Dozenten „Crystal Clear“ vorgegeben, das agile Prozessmodell für kleinere Teams von Alistair Cockburn [Co04]. Parallel bildete sich bereits beim zweiten Termin eine kleine Untergruppe, die sich mit agiler Dokumentation beschäftigte (Content-Management-System Joomla [Jo08] plus Einsatz eines Wikis der Vorgänger-Teams).

Auf Abbildung 9 und 10 sieht man das zu Projektbeginn gleichmäßig im Raum verteilte Projekt-Team, das bei der ersten größeren Aufgabe (Portierung des bestehenden Systems auf Linux) schon 3 Tage später das erste größere Mini-Team mit 4 Team-Mitgliedern bildet. Damit zeigte die Session 2007 das bereits in den vorhergegangenen Sessions beobachtete Mini-Team-Verhalten: größere Mini-Teams, die sich insbesondere mit „mission-critical“ Aufgaben beschäftigen.

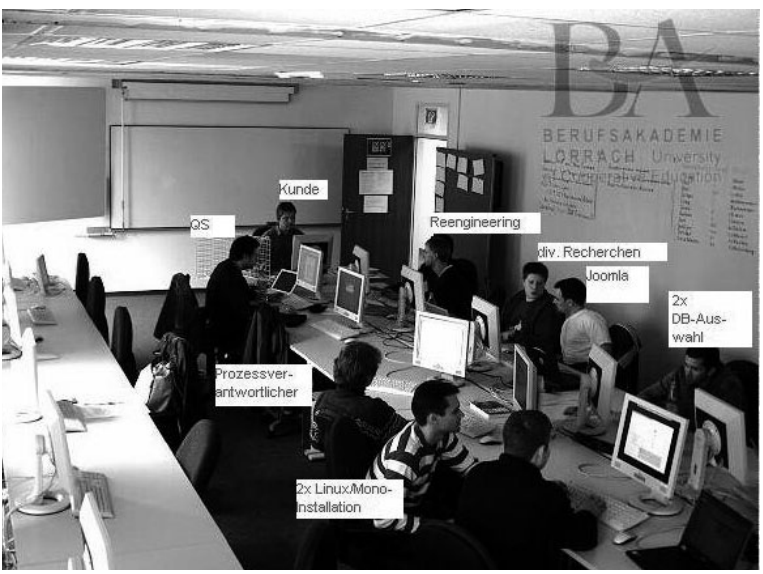


Abbildung 9: Session 2007: Zuordnung der Aufgaben (Projektbeginn)

Abbildung 10: Nur 3 Tage später bildet sich das erste größere Mini-Team (Portierung auf Linux)

2007 wurde das Team zum ersten Mal bezüglich des gelebten Prozess befragt. Crystal Clear fordert im Gegensatz zu XP nur wenige Eigenschaften zwingend:

1. Regelmäßige Lieferung des aktuellen, getesteten Codes,

2. Reflektierte Verbesserung des Prozess, erzielt durch regelmäßig abgehaltene Meetings,

3. Osmotische Kommunikation, d. h. Informationsaustausch im Team findet ganz „nebenbei", ohne zusätzlichen Aufwand (meist durch räumliche Nähe) statt.

Wurden diese Anforderungen an das Team erfüllt? In einer Umfrage sollten die Team-Mitglieder Schulnoten abgegeben bezüglich des Erreichens der oben genannten drei Ziele. Der Prozessverantwortliche analysierte dies im Vorfeld der Umfrage. Insbesondere die unregelmäßigen Code-Lieferungen und die nicht festgestellte Prozess-Reflektion wurden vom Prozessverantwortlichen bemängelt (vgl. Abb. 11).

Wie Crystal-Clear-konform arbeitet das Projektteam?

Projekt	ideal	IT-Labor 2007
Anzahl der Mitarbeiter	• 6 (max. 12)	• 11
Reflexions-Workshop	• < 1 Monat	• nie
Tägliche Standup-Meetings	• ja	• nie
Regelmäßige Lieferung	• Semesterende	• wurde eingehalten
Osmotische Kommunikation	• ja	• ja, ohne Probleme
Iterationslänge	• 3 Wochen	• unregelmäßig
Einfach Kontaktaufnahme mit Endandwendern	• ja	• anfangs nicht oft • während des Projektes immer wieder bei Bedarf
Häufigkeit der Integrationen	• kontinuierlich	• war anfangs nicht möglich, da Portierungsprobleme • gegen Ende wurde häufiger integriert, wenn auch nicht regelmäßig

Abbildung 11: Session 2007: Tabelle aus [Hn07]

Das Team hingegen gab sich selbst hinsichtlich der Erfüllung der Crystal Clear-Anforderungen im Schnitt folgende Schulnoten [Hä07]:

1.	**Crystal-Clear-Eigenschaft**	**Schulnote**
1	Regelmäßige Code-Lieferung	3,0
2	Reflektierte Prozess-Verbesserung	2,9
3	Osmotische Kommunikation	1,7

Diese Umfrage nach ca. drei Viertel der Projektzeit wurde am Ende des Projekts für die Punkte 1 und 2 nochmals durchgeführt und ergab noch bessere Werte:

2.	**Crystal-Clear-Eigenschaft**	**Schulnote**
1	Regelmäßige Code-Lieferung	2,5
2	Reflektierte Prozess-Verbesserung	2,3

Prozessverantwortlicher und Autor konnten lediglich die gute Note für die osmotische Kommunikation erklären. Die anderen Noten waren nicht nachvollziehbar. Deswegen wurden die Team-Mitglieder befragt:

Zum Punkt **(1) Regelmäßige Lieferungen** ergab sich [Hä07]:

„Da es sich bei dem Projekt um eine Portierungsaufgabe handelte, konnte anfangs kaum oder nur sehr schwer fertiger Code geliefert werden. Gegen Ende wurden die Lieferungen dann mehr und auch öfters durchgeführt. Aufgrund der Aufgabe waren regelmäßige Lieferungen nicht möglich“.

Dies ist allerdings nur begrenzt korrekt. Richtig ist wohl, dass das Team keine Notwendigkeit der regelmäßigen Lieferung sah und deshalb das Prozessmodell ungesagt anpasste.

Zum Punkt **(2) Reflektierte Verbesserung** ergab sich [Hä07]:

„Hier wurde vom Team nicht die reflektierte Verbesserung durch ein Meeting verstanden, sondern die vielen Verbesserungen, die durch Gespräche in kleinen Gruppen mit Projektmanager, Qualitätsmanager und auch Kunde erzielt wurden."

Während man bei (1) eventuell noch von einer „Interpretation" der Crystal-Eigenschaft reden könnte, hat das Team bei (2) eindeutig einen eigenen Weg beschritten. Der gelebte Prozess ist nicht mehr Crystal Clear!

13 "Meta Agile Process Model" (MAP)

Im Laufe der (Projekt-) Jahre kommt der Autor mehr und mehr zu der Überzeugung, dass der Ansatz der Entwicklung ständig neuer Prozessmodelle falsch ist. Es ist zwar wichtig, den Studierenden und Auszubildenden fundamentale Prozessmodelle als Projekt-„Baukasten" zur Verfügung zu stellen. Für die Projektpraxis wichtiger aber ist, von diesem „industrialisierten" Ansatz wegzukommen und die minimalen „Zutaten" zu benennen, die ein erfolgreiches Team und damit ein erfolgreicher Prozess benötigt. Man könnte diesen Ansatz als „Meta-Modell" bezeichnen, also als Modell, das beschreibt, welche Elemente ein erfolgreiches Prozessmodell enthalten muss. Der Autor bezeichnet diesen Ansatz als „Meta Agile Process Model" (MAP):

Ein erfolgreiches Projekt in einem nicht zu großen Team (bis vielleicht 20 Mitglieder) braucht nach unseren Erfahrungen aus dem IT-Labor der BA Lörrach folgende Zutaten:

- Ein Kunde, der seine Aufgabe ernst nimmt, und dem Team nicht unbedingt immer, aber genügend oft zur Verfügung steht.
- Ein bis zwei gute und flexible Kommunikations-Manager, die sich Projekt- und Qualitätsmanagementaufgaben aufteilen, aber vor allem die Kommunikation im Team und mit dem Kunden sicherstellen. Sie müssen profunde Kenntnisse in grundsätzlichen „Prototypen" von Prozessmodellen haben und sich nicht scheuen, das angewandte Prozessmodell abzuändern und anzupassen (das ist immer nötig!).
- Ein Integration Engineer, der allein verantwortlich, aber vom Projektmanagement überwacht, eine kontinuierliche Projekt-Integration einfordert und garantiert.
- Ein gut ausgewogenes Team mit einem guten psychologischen Prozess, in dem alle aus der Organisationspsychologie bekannten Typen in ausreichendem Maße vorkommen.

Wenn diese Bedingungen erfüllt sind, spielt nach den Erkenntnissen aus dem BA-IT-Labor das gewählte Software-Entwicklungs-Prozessmodell eine untergeordnete Rolle. Das Team wird im Rahmen der Projekt-Randbedingungen den geeigneten Prozess finden.

Literaturverzeichnis

[Am01] http://www.agilemanifesto.org

[BG01] Baldegger, R.: Erfolgreich im Team. Baldegger Verlag 2004.
Baldegger R.; Gotsmann L.: Ganzheitliches Projektmanagement. Baldegger 2001.

[Ba98] Balzert, H.: Lehrbuch der Software-Technik. Spektrum, 1998.

[Be00] Beck, K.: Extreme Programming Explained. Addison-Wesley 2000.

[Co04] Cockburn, A.: Crystal Clear. Addison-Wesley, 2004.

[Co08] Cockburn, A.: http://alistair.cockburn.us/

[Fo05] Fowler, M.: http://www.martinfowler.com/articles/newMethodology.html#id2251638

[HBM06] Hanser, E.; Brecht, U.; Michelbach, A.: Agile Software-Entwicklung: Extreme Programming im IT-Labor der BA Lörrach. BA Dialog Nr. 6, 2006.

[Ha06] Hanser, E.: Extreme Programming in Real World at the „IT Laboratory“ of BA Loerrach. ISTC Conference Proceedings 2006, Vaal University of Technology, South Africa.

[Hä07] Hättich N.: Das agile Prozessmodell Crystal. Studienarbeit, BA Lörrach, Studiengang Informationstechnik, 2007.

[JBR99] Jacobson I.; Booch G.; Rumbaugh J.: Unified Software Development Process. Addison-Wesley Professional, 1999.

[Jo08] http://www.joomla.de

[Mp08] http://www.mono-project.com

[RBB06] Rausch A.; Broy M.; Bergner K.: Das V-Modell XT, Grundlagen, Methodik und Anwendungen. Springer, 2006.
siehe auch: Dröschel W.; Wiemers M.: Das V-Modell 97. Oldenburg, 2000.

[Sp01] Späth, L.: Die NEW ECONOMY Revolution. Econ 2001.

Nutzeradäquate Prozessbeschreibungen in der Automotive E/E Entwicklung

Franziska Rothermel, Carsten König

Prozessgestaltung
Daimler AG Research & Development
Wilhelm-Runge-Straße 11
89081 Ulm
franziska.rothermel@daimler.com
carsten.koenig@daimler.com

Abstract: Mit steigender Produktkomplexität in der Automotive Elektrik / Elektronik (E/E) Entwicklung steigt auch die Komplexität der zugehörigen Entwicklungsprozesse und damit die Notwendigkeit, Prozessbeschreibungen für den Nutzer optimal zu gestalten. Die vorliegende Arbeit leistet hierzu einen Beitrag, indem sie einen Kriterienkatalog vorstellt, der es erlaubt, nutzeradäquate Prozessbeschreibungen zu erstellen und bestehende Prozessdarstellungen bezüglich ihrer Nutzeradäquatheit zu bewerten. Der theoretische Überbau wird durch eine umfassende Literaturrecherche geschaffen, die insbesondere den im Kontext Prozessbeschreibungen bisher vernachlässigten Bereich der Pädagogik miteinbezieht. Die resultierenden 430 Empfehlungen aus der Literatur wurden im Rahmen einer Befragung von Mitarbeitern der E/E Entwicklung bewertet und gleichzeitig durch weitere Nutzeranforderungen ergänzt. Der Kriterienkatalog und der zugehörige Styleguide wurden durch weitere Befragungen evaluiert und können in dieser ersten Untersuchung als empirisch bestätigt angesehen werden.

1 Prozessdarstellungen in der Elektrik / Elektronik Entwicklung

Elektronik im Automobil spielt eine immer bedeutendere Rolle: Rund 90 % der Innovationen betreffen elektronische Systeme, davon entfallen 80 % auf den Bereich Software [Sc02]. Elektrische und elektronische Bauteile sowie Software machen bereits heute etwa 20 % des Autowertes aus, bis zum Jahr 2015 wird dieser Anteil auf 30 % ansteigen [Wy06] – der Bereich Elektronik stellt somit einen wettbewerbsentscheidenden Faktor dar.

Steuergeräte, in deren Entwicklungsumfeld sich dieser Beitrag bewegt, sind elektronische Systeme, die im Automobil komplexe Regelungs-, Steuerungs- und Überwachungsaufgaben übernehmen. Während Mitte der 1990er Jahre nur einige wenige Steuergeräte im Auto verbaut worden sind, verfügt ein Wagen der Oberklasse heute in der Regel über mehr als 80 Steuergeräte, die miteinander vernetzt sind [WR08].

Abbildung 1 zeigt die E-Klasse von Mercedes-Benz mit ihren verbauten Steuergeräten und Leitungssätzen.

Abbildung 1: Mercedes-Benz E-Klasse mit verbauten Steuergeräten

Der steigende Anteil an Steuergeräten im Automobil und ihr hoher Vernetzungsgrad führen zu zusätzlichen Herausforderungen in der E/E Entwicklung. Dabei gilt: Je komplexer die Produkte, desto komplexer sind auch die Prozesse, die dahinter stehen [KB07]. Zusätzlich wird eine immer stärkere Parallelisierung von Entwicklungstätigkeiten verlangt, die faktisch die Koordination von mehr als 100 Prozessen und ihrer Arbeitsergebnisse erforderlich macht. Dies führt zur Frage, wie die Prozesse einem Prozessanwender präsentiert werden sollten, damit er diese ausführen kann, gerne bei der täglichen Arbeit nutzt und verinnerlicht. Dokumentierte Prozessbeschreibungen[1] leisten hier einen wichtigen Beitrag, da sie Prozesse detailliert darstellen und damit Know-How dauerhaft sichern sowie Prozessabläufe abteilungsübergreifend vereinheitlichen.

Die Prozesse, die im Rahmen der E/E Entwicklung zur Ausführung kommen, sind in Prozessbeschreibungen oder Verfahrensanweisungen dokumentiert, die jedoch nicht einheitlich gestaltet sind. Es existieren überwiegend in Microsoft Word oder PowerPoint erstellte Dokumente, die je nach Stil des Verfassers Prozesse anders inhaltlich aufbereiten, visuell darstellen und andere Schwerpunkte setzen. Dies erschwert den Mitarbeitern, die mit mehreren solcher Prozessbeschreibungen umgehen müssen, die Arbeit. Abbildung 2 zeigt Beispiele für die verschiedenen Darstellungsweisen von Prozessbeschreibungen.

[1] Unter einer Prozessbeschreibung versteht man eine dokumentierte Beschreibung eines Geschäftsprozesses, die dazu dient, betriebliche Prozesse für alle beteiligten Mitarbeiter transparent und nachvollziehbar zu machen [FFO06]. Prozesse bzw. Geschäftsprozesse können als „Folge von Aktivitäten, welche in einem logischen Zusammenhang stehen“ [Ma07, S. 176] definiert werden.

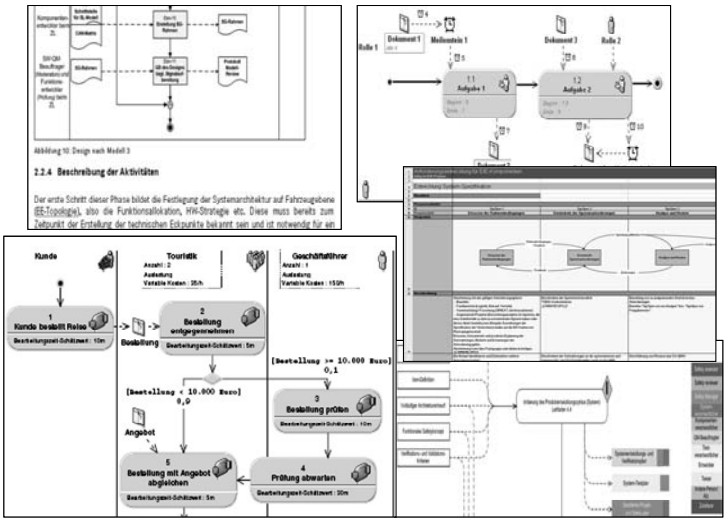


Abbildung 2: Verschiedene Darstellungsformen von Prozessbeschreibungen

Dieses Papier leistet einen Beitrag zur Lösung dieses Problems. Ein Kriterienkatalog wird präsentiert, der auf Basis einer breiten Literaturrecherche und einer Befragung von Mitarbeitern der E/E Entwicklung gründet und konkrete Anhaltspunkte liefert, wie eine nutzeradäquate Prozessbeschreibung aussehen sollte. Als nutzeradäquat werden in diesem Beitrag Prozessbeschreibungen verstanden, die den Bedürfnissen des Nutzers entsprechen. Der Nutzer ist hierbei ein Mitarbeiter der E/E-Entwicklung der Daimler AG. Weiterhin liefert ein Styleguide konkrete Vorgaben zur visuellen Umsetzung des Kriterienkataloges.

2 Empfehlungen aus der Literatur

Vor der Literaturrecherche fand zunächst eine Analyse der bestehenden Prozessbeschreibungen in der E/E Entwicklung statt. Auf dieser Basis wurden Literaturgebiete identifiziert, die potentiell Empfehlungen für nutzerfreundliche Prozessbeschreibungen enthalten. Dabei erwiesen sich fünf Bereiche als relevant: Informatik, Pädagogik, Psychologie, Wissensmanagement und Recht.

Innerhalb dieser Gebiete wurden 430 Empfehlungen aus ca. 100 Literaturquellen an eine nutzeradäquate Prozessbeschreibung identifiziert und in 13 Themenblöcken[2] zusammengefasst. Ein Beispiel ist der Themenblock „Darstellungen“ (Abbildung 3), dem 25 Empfehlungen zugeordnet sind. Eine konkrete Empfehlung beinhaltet beispielsweise, Prozessabläufe in Schaubildern darzustellen, die intuitiv lesbar sein sollten.

[2] Die Themenblöcke lauten: Bedienfreundlichkeit, Darstellungen, Tooleigenschaften, Individualisierbarkeit, Navigation und Orientierung, Kommunikation, Feedback, Umfang der Prozessbeschreibung, Wartung, Text, Nachschlagen, Farben, Bildschirmgestaltung. Die zugehörigen Empfehlungen der Themenblöcke sowie die zugehörigen Quellenangaben finden sich in [Ro08].

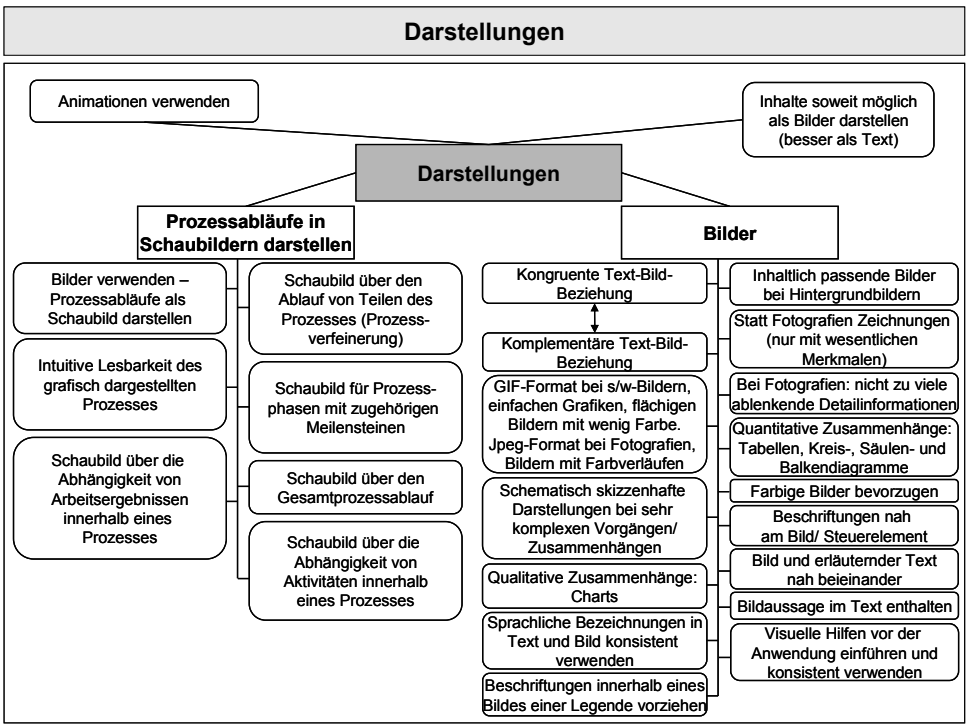


Abbildung 3: Themenblock „Darstellungen“

In der Literatur herrscht bezüglich Empfehlungen an eine nutzerfreundliche Darstellung von Inhalten und letztlich einer Prozessbeschreibung nicht immer Einigkeit. Ein Beispiel ist die Empfehlung, inwiefern im erläuternden Text zu einem Bild auf die Bildinhalte eingegangen werden soll. Es gibt drei Varianten für eine Text-Bild-Beziehung. [Ba97] nennt diese Beziehungen kongruent, komplementär und elaborativ, [Al07] bezeichnet dieselbigen als redundant, komplementär und diskrepant. Unter einer kongruenten bzw. redundanten Text-Bild Beziehung wird verstanden, dass der Text die Inhalte des Bildes beschreibt – d. h. Text und Bild beinhalten die gleichen Informationen. Stehen Text und Bild in einem komplementären Bezug, so ergänzen sich beide, so dass nur durch beide Darstellungsformen ein Gesamtbild entsteht. Die elaborative bzw. diskrepante Text-Bild-Beziehung geht davon aus, dass es im Text oder im Bild Elemente gibt, die zur jeweilig anderen Darstellung keine Verbindung haben – in anderen Worten, z. B. der Textinhalt geht über den Inhalt des Bildes hinaus [Al07; Ba97]. [Ba04; Bu02 sowie BG02] empfehlen eine kongruente Text-Bild-Beziehung.

Die sehr breit angelegte Literaturrecherche lieferte einen Katalog an Empfehlungen an eine nutzeradäquate Prozessbeschreibung und bildete die Grundlage für die Praxisbefragung.

3 Anforderungen aus der Praxis

Um Kriterien für eine nutzeradäquate Prozessbeschreibung aufstellen zu können, reicht eine rein theoretische Betrachtung nicht aus. Es muss die Einschätzung der konkreten Nutzer, also von Mitarbeitern der E/E Entwicklung, eingeholt werden, um eine Prozessbeschreibung entwickeln zu können, die von diesen Anwendern auch als nutzerfreundlich empfunden wird. Daher fand neben der umfassenden Literaturrecherche auch eine Befragung der künftigen Nutzer statt – persönlich im Interview und schriftlich mithilfe von Fragebögen, um eine größere Anzahl an Anwendern befragen zu können.

Ziel des Fragebogens war neben der Erfassung allgemeiner persönlicher Eigenschaften, wie z. B. Geschlecht, Farbfehlsichtigkeit, Qualifikation etc., eine Erfassung der Nutzungsgewohnheiten zu Prozessbeschreibungen. Weiterhin wurden Details über die individuell „optimale“ Prozessbeschreibung erfragt sowie erfolgte eine Bewertung der wichtigsten und via Fragebogen abfragbaren gesammelten Empfehlungen aus der Literatur (vgl. Kapitel 2) durch die Prozessbeschreibungsnutzer.

An der Befragung mittels Fragebögen nahmen insgesamt 28 Mitarbeiter des Bereichs Research & Development der Daimler AG teil. Acht Mitarbeiter entstammen einer Abteilung der E/E Entwicklung, die mit nur einem Prozess arbeiten und weitere 20 Mitarbeiter einer Abteilung, die neben der Arbeit mit ihrem eigenen Prozess in vielfältige weitere Prozesse als Mitarbeiter und Verantwortliche eingebunden sind. Im Folgenden werden einige Ergebnisse aus dem 19 Seiten umfassenden Fragebogen vorgestellt; eine komplette Darstellung der Ergebnisse ist [Ro08] zu entnehmen.

Insgesamt deckten sich die Anforderungen, die aus der Praxis im Rahmen des Fragebogens geäußert wurden, mit denen, die bereits in der Literaturrecherche gesammelt worden sind. So wurde beispielsweise von den Befragten die Anforderung geäußert, nur relevante Informationen in die Prozessbeschreibung aufzunehmen, die knapp und präzise gehalten werden sollte. Eine übersichtliche Gestaltung ist den Nutzern ebenso wichtig wie die intuitive Verständlichkeit der Prozessbeschreibung. Beispiele werden gewünscht, wie auch eine grafische Darstellung des Prozesses[3].

Was die Elemente einer nutzeradäquaten Prozessbeschreibung anbelangt, so wird von allen Befragten das Element „Rolle“ erwartet, gefolgt von „Aktivität“ (93 %), „Output“ (82 %) und „Erläuterungen zu Aktivitäten, Rollen etc.“ (82 %) (Tabelle 1).

[3] Alle geäußerten Anforderungen der befragten Mitarbeiter der E/E Entwicklung sind [Ro08] zu entnehmen.

Prozessbeschreibungselement	**Angaben in %**
Aktivität	93 %
Rolle	100 %
Input	75 %
Output	82 %
Templates	79 %
Mitgeltende Unterlagen	36 %
Erläuterungen zu Aktivitäten, Rollen etc.	82 %
Checkliste	39 %
Meilenstein	54 %
Attribute/Daten	25 %
Quality Gates	39 %
Anfangszeit	25 %
Endzeit	29 %
Dauer	21 %
Weitere	14 % (Zeitbezug, Zustände, best practices, Tools, ggf. Lieferant des Inputs, ggf. Kunde des Outputs, ggf. Links)

Tabelle 1: Erwartete Prozessbeschreibungselemente

Zusätzlich wurden die Empfehlungen aus der Literatur durch die Befragten bewertet. Sie hatten hier die Möglichkeit, vorgegebene Empfehlungen mit den vier Antwortkategorien „++“ (wichtig/stimme zu), „+“ (eher wichtig/stimme eher zu), „-“ (eher unwichtig/stimme eher nicht zu), „- -“ (unwichtig/stimme nicht zu) und „keine Angabe“ zu beurteilen. Zudem bestand die Möglichkeit, zu jeder abgefragten Empfehlung einen Kommentar abzugeben.

Tabelle 2 zeigt die Bewertungen zu ausgewählten Empfehlungen aus dem Themenblock „Darstellungen“ (meist vorgenommene Bewertung im Fettdruck, Angaben in %). Beispielsweise werden Schaubilder über den Gesamtprozess von den Entwicklern gewünscht, wohingegen die Verwendung von Videos in Prozessdarstellungen eher abgelehnt wird.

Empfehlungen aus der Literatur zum Themenblock „Darstellungen“	**--**	**-**	**+**	**++**	**Keine Angabe**
1. Animationen in der Prozessbeschreibung verwenden	36	**39**	7	0	17
2. Videos in der Prozessbeschreibung verwenden	32	**46**	4	0	18
3. Prozessabläufe in Schaubildern darstellen	0	4	29	**64**	4
3.1 Schaubild über die Abhängigkeit von Aktivitäten innerhalb eines Prozesses	0	7	36	**57**	0
3.2 Schaubild über Abhängigkeit von Arbeitsergebnissen innerhalb eines Prozesses	0	18	**46**	36	0
3.3 Schaubild über den Gesamtprozessablauf	0	0	14	**86**	0
3.4 Schaubild über den Ablauf von Teilen des Prozesses (Prozessverfeinerung)	0	11	18	**64**	4
3.5 Schaubild für einzelne Prozessphasen mit zugehörigen Meilensteinen	0	4	43	**54**	0
3.6 Schaubild über einen bestimmten Use Case des Prozesses	4	18	**43**	21	14
4. Graphische Darstellungen zusätzlich in Textform beschreiben	7	25	**46**	21	0

Tabelle 2: Bewertungen zum Themenblock "Darstellungen" (Angaben in %)

Mithilfe eines Leitfadeninterviews sind Anforderungen und insbesondere Use Cases für Prozessbeschreibungen erhoben worden, um die typischen Nutzungsgewohnheiten der Entwickler zu ermitteln. So werden Prozessbeschreibungen vor allem zur Einarbeitung in neue Prozesse, zum Nachschlagen und als Checkliste verwendet. Neben einer intuitiven Navigation, der Abbildung der gesamten Prozesslandkarte sowie der Einbindung von Verfahrensanweisungen, Guidelines und Checklisten kristallisierte sich im Interview der Wunsch nach selbsterklärenden Prozessbeschreibungen, die ohne Schulung erlernt werden können, heraus.[4]

Durch Kenntnis der Nutzungsgewohnheiten können die Ergebnisse der Befragung besser interpretiert und Kriterien für nutzeradäquate Prozessbeschreibungen leichter aufgestellt werden. So ist beispielsweise die Erzeugung von Checklisten aus der Prozessbeschreibung ein Kriterium im Rahmen des Kriterienkataloges, das aufgrund der Nutzungsgewohnheiten, die im Interview erhoben worden sind, trotz Bewertung im Fragebogen mit „+“ (nicht „++“) mit höchster Priorität in den Kriterienkatalog einging.

[4] Sämtliche Ergebnisse des Interviews sind [Ro08] zu entnehmen.

4 Kriterienkatalog und Styleguide

Auf Grundlage der gesammelten Empfehlungen aus der Literatur und der Ergebnisse aus der Praxisbefragung ist unter Berücksichtigung didaktischer Aspekte, beispielsweise Anregungen aus dem E-Learning und aus Lerntheorien, und der Ergebnisse der Zielgruppenanalyse (Zielgruppe sind die Mitarbeiter der E/E Entwicklung) ein Kriterienkatalog für eine nutzeradäquate Prozessbeschreibung erstellt worden. Jedes Kriterium ist mit einer Symbolleiste versehen, die die Priorität des Kriteriums, die Anzahl der Quellen, in denen es fundiert ist – die pädagogischen Quellen sind gesondert ausgewiesen – sowie die Widerspruchsfreiheit in der Literatur bezüglich des Kriteriums auf einen Blick angibt.

Die Priorität des Kriteriums ergibt sich aus den Bewertungen der Mitarbeiter der E/E-Entwicklung zu den Empfehlungen aus der Literatur, die im Rahmen des Fragebogens abgegeben wurden. Es sind drei Prioritäten verwendet worden, wobei Priorität *eins* die höchste, drei die niedrigste Stufe darstellt. Priorität *eins* erfordert eine sehr positive Bewertung in der Praxisbefragung („++“ im Fragebogen oder Nennung als Wunsch im Interview/ Fragebogen). Priorität *zwei* wird vergeben, wenn im Fragebogen mehrheitlich „+“ angekreuzt worden ist. Priorität *drei* umfasst all diejenigen Anforderungen, die bei Direktion A als positiv und bei Direktion B negativ angekreuzt worden sind (bzw. umgekehrt).

Abbildung 4 zeigt einen Auszug aus dem Kriterienkatalog, der einige der Kriterien für den Themenblock „Darstellungen“ enthält. Der gesamte Kriterienkatalog mit insgesamt 96 Kriterien ist [Ro08] zu entnehmen. Die Symbolleiste zeigt zunächst die Priorität des Kriteriums, z. B. „1“. Die Anzeigetafel neben dem Auto-Symbol zeigt die Bewertung aus der Praxis. Die Leiste neben dem Buch-Symbol weist die Anzahl der Literaturquellen auf. Ist der Punkt in dieser Leiste grün schraffiert, so ist in der Literatur hinsichtlich dieses Kriteriums kein Widerspruch aufgetreten, hat der Punkt die Farbe blau, so finden sich Widersprüche. Die Leiste, die sich ganz rechts neben dem Brillen-Symbol befindet, zeigt die Anzahl der Quellen aus der Pädagogik an. Ist der nebenstehende Punkt grün schraffiert, so sind insgesamt pädagogische Quellen vorhanden, ist er blau, so gibt es zum entsprechenden Kriterium keine Quellen aus der Pädagogik.

1. Prozessabläufe als Schaubild darstellen

Kommentar: *Im Rahmen der Zielgruppenanalyse wird eine Kombination aus visueller und textueller Repräsentation von Inhalten angeraten. Der Einbezug von Schaubildern im Gegensatz zu einer rein textbasierten Prozessbeschreibung wird in der Praxisbefragung bestätigt.*
Welche Schaubilder zusätzlich zum eigentlichen Prozess enthalten sein sollen, wird im Folgenden aufgeführt. Wichtig ist eine intuitive Lesbarkeit der vorzugsweise farbigen grafischen Darstellung.

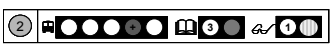
1.1 Schaubild über den Gesamtprozessablauf

Kommentar: *Neben der Darstellung des Gesamtprozessablaufs hat sich im Rahmen des Interviews auch der Wunsch nach einer Gesamtprozesslandkarte herauskristallisiert, in der alle Prozesse z. B. einer Abteilung und deren Schnittstellen zueinander, aufgeführt sind.*

1.2 Schaubild über die Abhängigkeit von Aktivitäten innerhalb eines Prozesses

Kommentar: *-*

2. Grafische Darstellungen zusätzlich in Textform erläutern



Kommentar: *Der Inhalt eines Bildes muss zusätzlich im Text erläutert werden (kongruente Text-Bild-Beziehung). Diese Forderung steht entgegen der Empfehlung von [Ba97] nach einer komplementären Text-Bild-Beziehung. In der Praxisbefragung wird jedoch mehrheitlich für eine kongruente Text-Bild-Beziehung votiert. Im Rahmen einer nutzeradäquaten Prozessbeschreibung ist daher dieser Form der Vorzug zu geben. Eine konsistente Verwendung von Begriffen, die im Bild vorkommen (auch im erläuternden Text) ist anzuraten, ebenso wie die Erklärung von visuellen Hilfen, die in den Bildern enthalten sind (z. B. Pfeile).*

Abbildung 4: Kriterienkatalog

Die Priorisierung der Kriterien hilft, neu zu erstellende Prozessbeschreibungen effizient nutzeradäquat zu gestalten. So sind je nach vorhandenen Ressourcen zunächst die Kriterien der Priorität *eins* zu erfüllen, im Anschluss daran diejenigen der Prioritäten *zwei* und *drei*.

Durch eine Bewertung in einfachen Schulnoten (1 = sehr gut erfüllt/ vorhanden bis 6 = unbefriedigend erfüllt/ nicht vorhanden) können bestehende Prozessbeschreibungen überprüft werden. Die Kriterien der Priorität *eins* werden dreifach, die der Priorität *zwei* doppelt und die Anforderungen der Priorität *drei* einfach gewichtet. So kann für jeden Themenbereich und auch für die gesamte Prozessbeschreibung eine Durchschnittsnote gebildet werden, die eine Aussage über die Nutzeradäquatheit der Prozessbeschreibung erlaubt.

Ein Styleguide macht konkrete Vorgaben zur visuellen Gestaltung eines webfähigen Interfaces für eine nutzeradäquate Prozessbeschreibung auf Basis des Kriterienkataloges. Dabei setzt das Corporate Design der Daimler AG – soweit mit dem Kriterienkatalog vereinbar – wichtige gestalterische Akzente, beispielsweise was die Farbauswahl und die grobe Anordnung der Bildschirmelemente betrifft. Dies hat den Vorteil, dass sich die Nutzer in einer gewohnten Umgebung wiederfinden, die gleichzeitig auf ihre Bedürfnisse zugeschnitten ist.

Auf dieser Basis und unter Berücksichtigung der Kriterien, die keine visuelle Auswirkungen haben und demnach nicht im Styleguide enthalten sind, kann eine Prozessbeschreibung implementiert werden, die im Sinne der Literatur und auf Grundlage der Bewertungen aus der Praxis als nutzeradäquat gelten kann.

Jeder Prozessschritt wird sowohl graphisch mithilfe eines Aktivitätsdiagramms als auch textuell in tabellarischer Form dargestellt. Nach eingehender Abwägung der Vor- und Nachteile sowie der Anforderungen der E/E-Entwicklungsabteilungen haben sich diese beiden Prozessbeschreibungstypen als am besten geeignet herausgestellt. Der obere, linke und in der Regel auch der rechte Bereich des Bildschirms bleiben der Navigation vorbehalten. Sämtliche Designvorgaben und weitere Beispiele sind [Ro08] zu entnehmen. Abbildung 5 zeigt einen Auszug aus dem erarbeiteten Styleguide.

Die oben beispielhaft aufgeführten Kriterien (Abbildung 4) finden sich auch im Styleguide wieder. So dient das Schaubild dazu, Prozessabläufe darzustellen (Kriterium „Prozessabläufe als Schaubild darstellen“, in der Literatur fundiert durch [BRB05], [Et06]). Für jeden Prozessschritt ist neben einem Schaubild auch eine textuelle Erklärung gemäß dem Kriterium „grafische Darstellungen zusätzlich in Textform erläutern“ (nach [Ba04], [Bu02], [BG02]) vorgesehen.

Der Ausschnitt des Styleguides (Abbildung 5) zeigt die Umsetzung einiger Kriterien des Katalogs. So sind beispielsweise die Kriterien „Druckfunktion“ (nach [AB07], [Ba04], [HG06], [Pu01], [Th01], [Wi02]) sowie eine „Sitemap“ (nach [Ba04], [Bu02], [HG06], [Pu01], [Wi05]) realisiert. Auch ist es möglich, die Sicht des Schaubildes zu verändern (Management-Sicht und Entwickler-Sicht). So wird eine Informationsdarstellung in Abhängigkeit der jeweiligen Rolle (nach [BG02], [BRB03], [Eb94], [EG90], [Ko93], [KS04], [Wi07]) gewährleistet und ein Wechsel zwischen verschiedenen Prozesssichten ist möglich (nach [BRB05]). Der Kriterienkatalog empfiehlt ebenso, eine Pfadangabe (nach [Ba04], [Pu01]) bereitzustellen sowie eine verlinkte Navigation (nach [Ba04], [BG02]) zu gewährleisten, was im Styleguide ebenso realisiert wird.

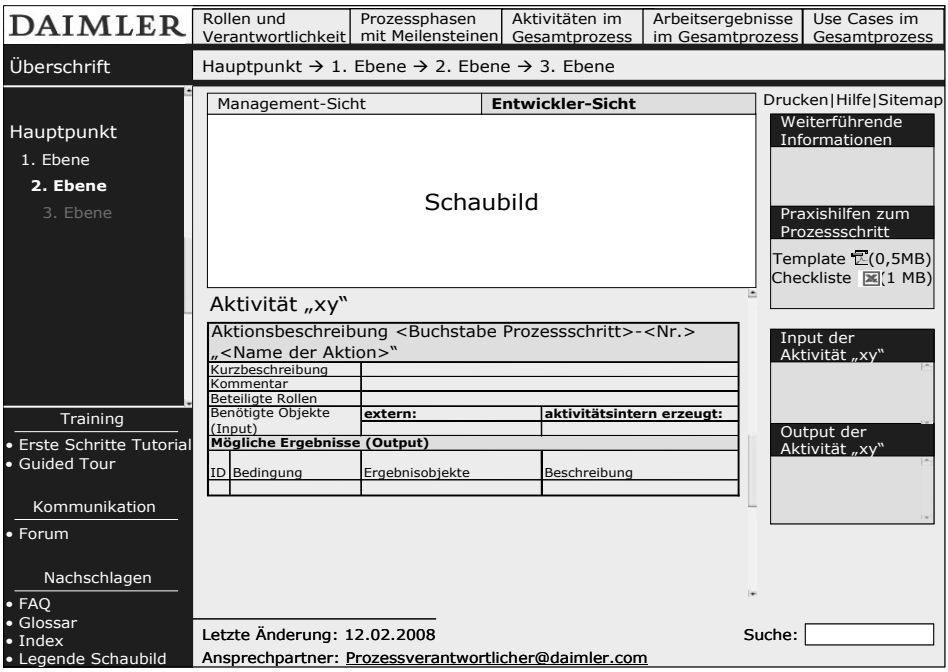


Abbildung 5: Styleguide

5 Validierung

Die Validierung fand mittels einer schriftlichen Befragung statt, an der acht Mitarbeiter der E/E Entwicklung teilnahmen. Ziel des Fragebogens war, die Nutzeradäquatheit des visuell umgesetzten Prozessbeschreibungsbeispiels zu evaluieren. Um dieses Ziel bestmöglich zu erreichen, wurde der Fragebogen im Sinne einer „Entdeckungstour“ gestaltet, in der der Befragte in vier Schritten durch die einzelnen Screenshots der beispielhaften Prozessbeschreibung geführt wird. Zusätzlich wurden allgemeine Fragen über die vorgestellte Prozessbeschreibung gestellt.

Das Ergebnis der Befragung war durchwegs positiv. So sahen alle Befragten (100 %) die vorgestellte, gemäß den erarbeiteten Kriterien *nutzeradäquate Prozessbeschreibung* als wirkliche Alternative zu ihrer bisherigen. Mit Ausnahme eines Befragten, der die Frage nach intuitiver Bedienbarkeit mit „mittelmäßig“ beurteilte, sahen alle befragten Mitarbeiter die erstellte Prozessbeschreibung als intuitiv bedienbar an.

In dieser ersten Untersuchung kann daher davon ausgegangen werden, dass die Anforderungen der Nutzer korrekt erfasst und erfüllt wurden und der erstellte Kriterienkatalog somit als valide angesehen werden kann.

6 Zusammenfassung und Ausblick

Die Komplexität von System- und Komponentenentwicklungsprozessen im Automotive Umfeld erhöht sich stetig, was in der Folge die Güte der Prozessbeschreibungen immer bedeutender werden lässt. Dieser Beitrag präsentiert einen Ansatz, Prozessbeschreibungen nutzerfreundlich darzustellen. Dazu wurde ein Kriterienkatalog vorgestellt, der insgesamt 96 Kriterien an eine nutzeradäquate Prozessbeschreibung enthält, sowie ein Styleguide, der bei der Erstellung von Prozessbeschreibungen unterstützt. Die Basis hierzu bot eine umfassend angelegte Literaturrecherche, aus der 430 Empfehlungen an eine nutzerfreundliche Darstellung einer Prozessbeschreibung hervorgegangen sind. Diese bildete den Grundstock für die Praxisbefragung, die als wichtigstes Ergebnis eine Bewertung ausgewählter Empfehlungen aus Sicht der Mitarbeiter der E/E Entwicklung lieferte. Neben den Anforderungen aus der Praxis flossen die im Rahmen der Bewertung im Fragebogen als positiv befundenen Empfehlungen aus der Literatur in den Kriterienkatalog mit ein. Didaktische Aspekte, wie z. B. Anregungen aus dem Bereich des E-Learning oder aus Lerntheorien sowie eine Zielgruppenanalyse des späteren Nutzers, eines Entwicklers in der E/E Entwicklung, ergänzen die praktische Komponente des Kriterienkatalogs.

In einer weiteren Befragung ausgewählter Mitarbeiter der E/E Entwicklung mithilfe eines Fragebogens erwies sich der vorgestellte Styleguide und somit auch der Kriterienkatalog in dieser ersten Untersuchung als valide.

Verbesserungen der Prozesslandschaften können durch den vorgestellten Kriterienkatalog messbar gemacht, quantitative Ziele zur Verbesserung gesetzt werden. Ebenso ist die Definition von Mindeststandards für Prozessbeschreibungen auf Grundlage der erarbeiteten Kriterien denkbar.

Zukünftige Arbeiten zielen auf die Erstellung nutzeradäquater Prozessportale, weitere Validierung der vorgestellten Kriterien sowie zusätzliche – stärker zielgruppendifferenzierende – Befragungen der Nutzer.

Literaturverzeichnis

[AB07] Armbrust, O.; Bella, F.: Nutzungsangemessene Prozessdarstellung. Internes Dokument der Daimler AG, 2007.

[Al07] Alexander, K.: Kompendium der visuellen Information und Kommunikation. Springer-Verlag, Berlin, Heidelberg, 2007.

[Ba97] Ballstaedt, S.-P.: Wissensvermittlung: Die Gestaltung von Lernmaterial. Psychologie Verlags Union, Weinheim, 1997.

[Ba04] Balzert, H.: Webdesign & Web-Ergonomie: Websites professionell gestalten. W3L-Verlag, Herdekke, Dortmund, 2004.

[BG02] Bruns, B.; Gajewski, P.: Multimediales Lernen im Netz: Leitfaden für Entscheider und Planer. Springer-Verlag, Berlin, Heidelberg, 2002.

[BRB05] Bobrik, R.; Reichert, M.; Bauer, T.: Requirements for the Visualization of System-Spanning Business Processes. In (IEEE Hrsg.): Proceedings of the 16th International Workshop on Database and Expert Systems Applications, Kopenhagen, 2005; S. 948-954.

[Bu02] Bundesministerium der Justiz: Verordnung zur Schaffung barrierefreier Informationstechnik nach dem Behindertengleichstellungsgesetz: BITV. BGB. I 2002; S. 2654-2662.

[Eb04] Eberleh, E.: Industrielle Gestaltungsrichtlinien für graphische Benutzungsoberflächen. In (Eberleh, E.; Oberquelle, H.; Oppermann, R. Hrsg.): Einführung in die Software-Ergonomie. Gestaltung graphisch-interaktiver Systeme: Prinzipien, Werkzeuge, Lösungen. Walter de Gruyter, Berlin, New York, 1994; S. 145-195.

[EG90] Rat der europäischen Gemeinschaften: 90/270/EWG, 1990. Zuletzt besucht am 05. November 2007 unter http://eur-lex.europa.eu/Notice.do?mode=dbl&lang=en&lng1=en,de&lng2=bg,cs,da,de,el,en,es,et,fi,fr,ga,hu,it,lt,lv,mt,nl,pl,pt,ro,sk,sl,sv,&val=161275:cs&page=1&hwords=90%2F270%2FEWG%7E.

[Et06] E-teaching.org: Typographie, 2006. Zuletzt besucht am 22. Januar 2008 unter http://www.eteaching.org/didaktik/gestaltung/screendesign/typographie/.

[FFO06] Fischer, H.; Fleischmann, A.; Obermeier, S.: Geschäftsprozesse realisieren: Ein praxisorientierter Leitfaden von der Strategie bis zur Implementierung. Vieweg Verlag, Wiesbaden, 2006.

[HG06] U.S. Department of Health and Human Services (HHS); U.S. General Services Administration (GSA): Research-Based Web Design & Usability Guidelines, 2006. Zugriff am 29. Januar 2008 unter http://usability.gov/pdfs/guidelines_book.pdf.

[KB07] König, C.; Braun, T.: Process Complexity Management – Bottom up Prozesslandkarten verbessern. In (SQS Software Quality Systems AG Hrsg.): 12th Software & Systems Quality Conferences, Düsseldorf, 2007.

[Ko93] Koller, F.: Benutzergerechte Gestaltung von Multimedia-Systemen. In (Ziegler, J.; Ilg, R. Hrsg.): Benutzergerechte Software-Gestaltung. Standards, Methoden und Werkzeuge. Oldenbourg Verlag, München, Wien, 1993; S. 103-122.

[KS04] Knippel, E.; Schulz, A.: Lessons Learned from Implementing Configuration Management within Electrical/Electronic Development of an Automotive OEM. In (INCOSE Hrsg.): Proceedings of the 14th Annual International Symposium. Systems Engineering: Managing Complexity and Change, Toulouse, 2004.

[Ma07] Masak, D.: SOA? Serviceorientierung in Business und Software. Springer-Verlag, Berlin, Heidelberg, 2007.

[Pu01] Puscher, F.: Das Usability Prinzip: Wege zur benutzerfreundlichen Website. Dpunkt.verlag, Heidelberg, 2001.

[Ro08] Rothermel, F.: Nutzeradäquate Prozessbeschreibung unter Berücksichtigung didaktischer Aspekte im Kontext der Steuergeräteentwicklung. Diplomarbeit, Universität Erlangen-Nürnberg, März 2008.

[Sc02] Schleuter, W.: Automobilindustrie: Innovationen immer stärker durch die Elektronik geprägt. In (IKB Deutsche Industriebank AG Bereich Volkswirtschaft Hrsg.): Zukunftstrends für den Mittelstand. Miniaturisierung und Elektronik auf dem Vormarsch. Köln, 2002; S. 8-9.

[Th01] The, T.-S.: Untersuchungsaufbau der evaluierenden Marktstudie über aktuelle Business Process Management Tools. In (Bollinger, H.-J.; Schreiner, P. Hrsg.): Business Process Management Tools. Eine evaluierende Marktstudie über aktuelle Werkzeuge. Fraunhofer IRB Verlag, Stuttgart, 2001; S. 61-75.

[Wi02] Wirth, T.: Missing Links: Über gutes Webdesign. Carl Hanser Verlag, München, Wien, 2002.

[Wi05] Wirth, T.: Die EN ISO 9241-10, 2005. Zuletzt besucht am 26. Oktober 2007 unter http://www.kommdesign.de/texte/din.htm.

[Wi07] Wilbers, K.: E-Learning didaktisch gestalten. In (Hohenstein, A.; Wilbers, K. Hrsg.): Handbuch E-Learning. Expertenwissen aus Wissenschaft und Praxis. Deutscher Wirtschaftsdienst, Köln, 21. Erg.-Lfg. Juli 2007; 4.0, S. 1-42.

[WR08] Wille, M.; Richter, H.: Das Netzwerk für das Auto von morgen: Ringtopologie bietet Skalierbarkeit, Zuverlässigkeit und Echtzeit-Fähigkeit. Zugriff am 06. März 2008 unter http://www.elektroniknet.de/home/automotive/technik-know-how/uebersicht/l/bussysteme/das-netzwerk-fuer-das-auto-von-morgen/.

[Wy06] Wyman, O.: Oliver Wyman-Studie Autoelektronik: Elektronik setzt die Impulse im Auto. München, 2006. Zuletzt besucht am 27. Februar 2008 unter http://www.oliverwyman.com/de/pdf_files/OW_PM_Autoelektronik.pdf.

Industrialisierung der Softwarewartung & -weiterentwicklung

Harry M. Sneed

ANECON GmbH
Alserhof 4
1090 Wien
Harry.Sneed@anecon.com

Abstract: In diesem Beitrag wird geschildert, wie Software-Wartung & -Weiterentwicklung als Dienstleistung von externen Software-Firmen angeboten werden kann. Es wird argumentiert, dass die Vergabe der Systemerhaltung die einzige Möglichkeit ist die Kosten derselben in Griff zu bekommen. Der Outsourcing Partner braucht allerdings einen straffen, automatisierten Wartungsprozess, der hier beschrieben wird. Einen Nutzen durch diese Dienstleistung wird es erst geben, wenn sowohl der Anwender als auch der Outsourcing Partner genau definierte Rechte und Pflichten haben, die in einem Service-Level-Agreement festgehalten sind. Jener Vertrag ist für den Erfolg des Outsourcing ausschlaggebend. Welche Rechte und Pflichten das sind und wie sie einzuhalten sind, sowie der Prozess zur Durchführung der vereinbarten Dienstleistungen ist das Thema des vorliegenden Beitrags.

1 Hintergrund – Die Wartungsfalle

Die Last für die Erhaltung bestehender IT-Systeme ist ein Mühlstein um den Hals vieler Anwendungsbetriebe, vor allem für diejenigen, die nicht auf ERP-Produkte ausweichen können. Dazu gehören im ersten Range die Finanzdienstleister und die Behörden. Sie betreiben Legacy-Systeme, die größtenteils aus den 80er Jahren stammen. Diese Systeme werden nicht nur korrigiert und angepasst (corrective and adaptive maintenance), sondern auch optimiert und weiterentwickelt (perfective und enhancive maintenance). Um diese Aufgaben zu bewältigen sind Heersscharen alter Programmierer beschäftigt. Auch in Zeiten, in denen sie nur gering ausgelastet sind, müssen die Programmierer trotzdem bereit stehen, falls sie gebraucht werden. Durch ihr Wissen über die alten Systeme sind sie für den Anwender unentbehrlich. Stellvertretend für diese Situation ist das Softwareprofil eines typischen deutschen Finanzdienstleisters. Die IT Abteilung der Firma muss eine Altlast bestehend aus

- 3.652 IMS Datenbanken,
- 2.320 DB2 Tabellen,
- 31.453 IMS Masken,
- 10.933 Assembler Programmen,
- 10.450 PLI Programmen,
- 27.374 COBOL Programmen,
- mit mehr als 70 Millionen Code-Zeilen, bzw. 46 Millionen Anweisungen

betreuen und fortschreiben. Für die Pflege jener Altsysteme stehen gerade 360 Programmierer zur Verfügung. Daraus leitet sich ein Verhältnis zwischen Personal und Anweisungen von

- 1:127.777

ab.

Das heißt, ein Wartungsprogrammierer muss circa 128 Tausend Anweisungen betreuen - und das bei einer mittleren jährlichen Änderungsrate um 5 %. Aus der Literatur geht hervor, dass ein Wartungsprogrammierer nicht mehr als 80 Tausend Anweisungen betreuen sollte [Pi97]. In diesem Falle muss der einzelne Programmierer fast 6.388 Anweisungen im Jahr, bzw. 32 Anweisungen pro Tag ändern, löschen und hinzufügen. Unter diesen Umständen ist es äußerst schwierig, den Wartungsbetrieb überhaupt aufrecht zu erhalten- vor allem, wenn man berücksichtigt, dass die durchschnittliche Produktivität eines Entwicklers bei 25 Anweisungen pro Tag liegt [Bo99].

Codezeilen sind physikalische Einheiten, deren Länge von der Art des Editors bestimmt wird. Auf dem Mainframe in TSO sind sie 72 Zeichen; auf dem PC können sie beliebig lang sein. Codezeilen zu zählen macht wenig Sinn. Anweisungen sind hingegen logische Einheiten wie Sätze in der natürlichen Sprache. Sie werden mit einem Delimiter wie bspw. Punkt oder Semikomma beendet. Es können mehrere Anweisungen in einer Zeile vorkommen. Anweisungen können sich auch über mehrere Zeilen erstrecken, was eher vorkommt. Deshalb ist die Zahl der Anweisungen, bzw. „delivered source instructions", in der Regel kleiner als die der LOCs [Hu00] (siehe Abbildung 1).

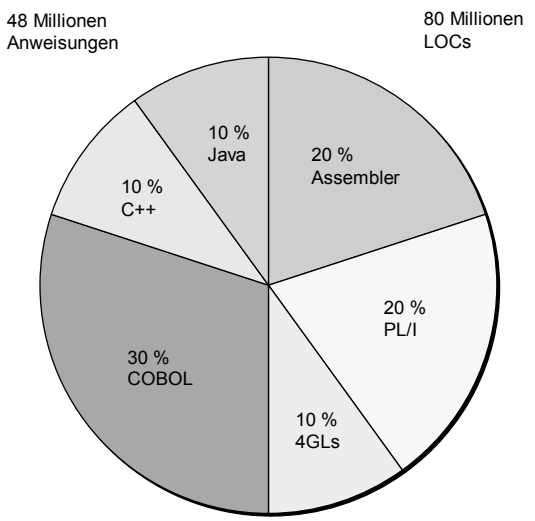


Abbildung 1: Softwareprofil eines Finanzdienstleisters

Die Kosten für die Aufrechterhaltung derartiger Wartungsbetriebe sind immens. Laut einer Umfrage der Gartner-Gruppe betragen sie im Durchschnitt 72 % des gesamten IT-Budgets [Ga05]. Der Wartungsbetrieb bindet wertvolle Ressourcen, vor allem wichtige Know-How-Träger, die für andere Aufgaben dringend gebraucht werden. Die alte Software zwingt die Anwender dazu, die alte Hardware zu behalten und teure Lizenzen zu bezahlen, da die Kosten einer Migration für sie oft unerschwinglich sind. Die wenigen Anwenderbetriebe, die es doch schaffen ihre Legacy-Systeme zu migrieren, müssen dafür einen hohen Preis bezahlen und tun sich schwer, diese Kosten gegenüber dem Geldgeber zu rechtfertigen, denn die Endanwender haben meistens nichts davon. Wenn alles gut geht, dürften sie die Umstellung kaum merken. Demzufolge sind sie wenig geneigt, diese Kosten zu tragen [BS95].

Folglich bleibt der Anwender in einem Dickicht von Abhängigkeiten stecken. Er ist abhängig von den Lieferanten der alten Software- und Hardware-Produkten, abhängig von den externen Spezialisten, die sich mit den alten Produkten auskennen und abhängig von den eigenen Programmierern, die einzig und allein in der Lage sind, die Altsoftware fortzuschreiben. Dem in der Legacy-Falle gefangene IT Anwender bleibt kaum noch Spielraum für Verbesserungsmaßnahmen, geschweige denn für Innovationen und neue Investitionen.

2 Befreiungsmöglichkeiten

Es bieten sich nur wenige Möglichkeiten an, sich aus dieser misslichen Lage zu befreien:

- Zum ersten, könnte der Anwender auf Standard-Software, sprich auf ERP Systeme umstellen.
- Zum zweiten, könnte der Anwender durch einen herkulischen Akt die Altsysteme sanieren und erneuern.
- Zum dritten, könnte der Anwender mit einem noch größeren Kraftakt die Altsysteme durch Neuentwicklungen ablösen.
- Zum vierten, könnte der Anwender die Altsysteme an einen externen Outsourcing-Partner übergeben.

Es sei zu betonen, dass eine Migration zwar den Anwender von der alten Hardware und Basissoftware befreien kann, jedoch nicht von der eigenen Anwendungssoftware, die in der neuen Umgebung unverändert bleibt. Durch eine Migration wird weder an der Größe, der Komplexität noch an der Qualität der Software gerüttelt [SAW08].

Der Hauptgrund für eine Migration ist einzig und allein der Wunsch nach Unabhängigkeit von einer bestimmten Hard- bzw. Software. Die zuständigen Manager wollen den Lieferanten wechseln, die Wartungskosten spielen dabei nur eine geringe Rolle. Die Abhängigkeit von gewissen Schlüsselpersonen scheint niemanden zu stören solange man glaubt, die Leute behalten zu können. Hinzu kommt, dass die meisten IT Leiter ohnehin nur kurzfristig planen und hoffen, die Altsysteme so lange über Wasser halten zu können. Ergo wird weder Sanierung noch Migration als Ausweg aus der Wartungskrise gesehen.

2.1 Standard-Software

Die erste Möglichkeit - die Umstellung auf Standard Software - setzt voraus, dass geeignete Standardlösungen angeboten werden. Für die Industrie ist dies inzwischen weitestgehend der Fall. Bis auf wenige Ausnahmen kann der industrielle Anwender seine IT-Systeme kaufen oder mieten. Er kann sogar zwischen Produktbeschaffung und Servicebeschaffung, sprich Software as a Service, wählen [SS03]. Er muss allerdings seine alten Daten migrieren und seine Geschäftsprozesse anpassen. Dies sei jedoch ein geringer Preis für die Befreiung aus der Wartungsfalle. Er begibt sich zwar in eine neue Abhängigkeit, aber diese ist kalkulierbar und tragbar. Sonst würde nicht der Großteil der Industrie diesen Weg einschlagen. Für Unternehmen mit besonderen Anwendungen gibt es diese Alternative leider nicht.

2.2 Sanierung und Erneuerung

Die zweite Möglichkeit erfordert eine größere einmalige Investition. Die Altsoftware wird im figurativen Sinne durch eine Jungfernmühle getrieben und kommt als erneuerte, wartbare und weiterentwicklungsfähige Software auf der anderen Seite wieder heraus. Dieser alte Traum des Software-Reengineering geht auf die 80er Jahre zurück und bleibt bisweilen weitgehend unerfüllt. Es gibt kaum Beispiele wirklichen Gelingens. Einzelne Bausteine bzw. Module lassen sich restrukturieren und bereinigen, einzelne Klassen lassen sich ebenfalls refaktorieren, aber die Gesamtarchitektur bleibt im Wesentlichen so, wie sie war. Auch die einzelnen Komponenten lassen sich nur bedingt verbessern. Man kann zwar die GOTO Anweisungen entfernen und den Kontrollfluss restrukturieren. Man kann auch die IO-Operationen auslagen und die Programme zerlegen in kleinere Module, aber die Qualität der Komponente steigt nur marginal. Die Software-Bausteine bleiben in deren ursprünglicher Technologie behaftet. Außerdem bleibt die Abhängigkeit zu den alten Programmierern weiter bestehen. Die Wartungslast wird dadurch nur um wenige Tausend leichter [Sn08].

Denn aller Erfahrung zu Folge steigt auf Grund einer Sanierung, sprich Restrukturierung oder Refaktorierung, die innere Qualität einer Software nie mehr als 30%. Das liegt daran, dass eine technische Lösung sich nur begrenzt verändern lässt, ohne die Lösung als solche zu zerstören. Das gleiche gilt auch für ein altes Haus, das saniert wird. Die Grenze zwischen Sanierung und Abbruch ist fließend. Da die Qualität eines Softwareproduktes nur eine von vier Wartungskostentreibern ist, neben

- der Wartungsumgebung,
- dem Wartungsprozess und
- dem Wartungspersonal,

kann sie allenfalls ein Viertel der Wartungskosten verursachen. Auch wenn eine Sanierung die Wartbarkeit um 30 % steigern könnte, würde im besten Falle circa 10 % der Wartungskosten gespart werden, da die vom Produkt verursachten Kosten höchstens einen Drittel der Gesamtkosten ausmachen [Sn91]. Die restlichen zweidrittel der Kosten werden von der Umgebung, dem Prozess und dem Personal verursacht.

Es ist daher fraglich, ob Anwender bereit sind die Kosten eines größeren Software-Reengineering Projektes zu tragen, wenn der Return on Invest so niedrig ist. Sie werden eher nach einfacheren, mehr Gewinn versprechenden Möglichkeiten suchen.

2.3 Neuentwicklung

Die Neuentwicklung bedeutet das alte System nachzudokumentieren und anhand der wieder gewonnen Dokumentation neu zu kodieren. Diese Lösung hat drei Nachteile. Zum einen ist es sehr schwierig herauszubekommen, was das alte System wirklich tut. Der Reverse Engineering Technologie ist es nie gelungen das eigentliche Fachkonzept aus dem Code wieder zu gewinnen. Zum zweiten ist eine neue Entwicklung sehr teuer, insbesondere was den Test anbetrifft. Eine völlig neue Testbasis muss aufgebaut werden. Das treibt die Kosten in die Höhe. Zum dritten ist eine neue Entwicklung immer mit hohen Risiken verbunden. Es hat sich nämlich gezeigt, dass die Ablösung eines bestehenden Systems um das zweifache risikoreicher ist als die Entwicklung eines völlig neuen Systems. Das kommt daher, dass die Anwender alles in dem alten System behalten wollen bei gleichzeitiger Ergänzung des Systems durch alle neue Anforderungen, die sich über die Jahre angesammelt haben. Man spricht hier vom „second system effect" - bzw. der Wunsch alles besser zu machen mit der Folge, dass das System überlastet wird [Th95].

Nur die wenigsten Anwender lassen sich auf dieses Abenteuer ein. Zu hoch scheinen ihnen die Risiken. Mit der Ablösung des alten Systems folgt die Ablösung der alten Programmierer, die jetzt umgeschult werden müssen. Durch die Umstellung auf eine neue Technologie werden die bisherigen Programmierer abgehängt. Sie können oft nur noch als Analytiker oder Tester weiterarbeiten. Nur wenige große Anwender sind in der Lage ihr altes Personal umzuschulen, bzw. auszuwechseln. Es bleibt den Anderen nur übrig, die alte Technologie beizubehalten.

2.4. Outsourcing der Software-Wartung

Als vierte Alternative bietet sich das Outsourcing der Softwarewartung an. Die Verantwortung für die Pflege und Weiterentwicklung wird an einen externen Dienstleister vergeben, der mit dem Anwender ein Service-Level-Agreement abschließt, wonach er Probleme in einer vereinbarten Zeit beseitigt und neue Versionen mit Änderungen und Erweiterungen in vereinbarten Intervallen ausliefert.

Die Idee für die 3rd Party Maintenance ist nicht neu. Sie hat Hillary Calow schon 1988 in England aufgegriffen. Calow hat eine Firma namens FI gegründet, die ausschließlich aus Müttern bestand, die vorher als Programmiererinnen gearbeitet hatten und nun zu Hause bleiben wollten, ohne ihren Beruf aufzugeben. FI schloss Verträge mit der lokalen Industrie ab, wonach die Frauen einmal wöchentlich zu Gesprächen mit den Kunden kamen. Die anderen Tage blieben sie zu Hause wo sie jederzeit erreichbar waren. Sie hatten einen lokalen Netzanschluss zum Hostrechner des Kunden. Sie bekamen ihre Wartungsaufträge über das Netz und hatten Zugriff auf die Source-Bibliotheken des Kunden. Außerdem hatten sie auch eine eigene Testumgebung.

Allen Berichten zu Folge hat das Geschäft mit der 3rd Party Maintenance gut funktioniert und die Kunden waren zufrieden. Die von zu Hause aus operierenden Frauen waren hoch motiviert und haben professionell gearbeitet [Ca88]. Der Untergang des FI Geschäfts setzte ein, als die Inder Ende der 90er Jahre in großem Stil die Software aus England nach Indien abgezogen haben. FI konnte nicht mit den niedrigen Preisen der großen, indischen Anbieter TCS, Wipro und InfoSys konkurrieren. Anwender, die bereit waren, ihre Software aus der Ferne warten zu lassen, haben dies nun in Indien erledigen lassen. Inzwischen ist Business Process Outsourcing ein gängiger Begriff geworden. Das was dahinter steckt ist im Grunde genommen 3rd Party Maintenance. Nur der Name hat sich geändert [Ni08].

Die Vorteile eines Outsourcings des Wartungsbetriebs liegen auf der Hand:

- Erstens, ist es Sache des Outsourcing-Partners, die Software zu sanieren. Wenn er dadurch kosten spart, wird er dies tun, aber ohne die Kontinuität der Dienstleistung zu gefährden oder den Rahmen des vereinbarten Wartungsbudgets zu springen.
- Zweitens, wird die Wartungsdienstleistung in einem Vertrag zwischen dem Kunden und dem Dienstleister bis ins kleinste Detail abgestimmt. Der Prozess wird vertraglich im SLA festgehalten und die geringste Abweichung davon wird mit einer Konventionsstrafe geahndet.
- Drittens, wird der Dienstleister dafür sorgen, dass er mit den besten und modernsten Werkzeugen ausgestattet ist, denn nur so wird er effizient arbeiten können. Es liegt in seinem Interesse, die Wartungsproduktivität zu steigern, denn mit jeder Produktivitätssteigerung verdient er mehr.
- Viertens, werden die Mitarbeiter des jeweiligen Dienstleisters professionelle Wartungsprogrammierer mit einer speziellen Ausbildung sein, die motiviert sind, ihre Arbeit best möglichst zu verrichten. Dies steht im starken Gegensatz zur amateurhaften, unmotivierten Verhalten der meisten Anwenderprogrammierer, die ihren Job als Programmpfleger nur widerwillig ertragen. Es wäre sogar sinnvoll und möglich, das externe Wartungspersonal nach Leistung zu bezahlen, z. B. nach korrigierten Fehlern und durchgeführten Änderungsanträgen.
- Fünftens, hat der Anwender eine Garantie, dass seine Systeme termingerecht und mit einer zugesicherten Qualität fortgeschrieben werden. Er ist nicht länger von einzelnen Personen im seinem Betrieb abhängig, deren Leistung er nicht bestimmen kann.

Das Outsourcing der Systemwartung und -weiterentwicklung ist somit ein wichtiger Schritt in Richtung Industrialisierung des Software-Managements. Dadurch werden die Wartungsabläufe optimiert, die Wartungsleistung messbar und die Wartungskosten transparent. Die Reverse-Engineering Aufgaben sind in der Verantwortung des Dienstleisters. Er hat den Hauptnutzen von einer Nachdokumentation der Software. Deshalb soll er sich um das Reverse Engineering kümmern. Es obliegt ihm ebenfalls, dafür zu sorgen, dass die funktionale Äquivalenz erhalten bleibt. Es werden hochwertige Wartungswerkzeuge eingesetzt, da dies im Interesse des Service Providers liegt, die Wartungsaufträge möglichst effizient und ohne Qualitätsverlust durchzuführen. Dass der Regressionstest möglichst systematisiert und automatisiert wird, versteht sich von selbst, denn dadurch kann der Anbieter die meisten Kosten einsparen [SHT05].

3 Der Wartungsprozess

Solange die Anwendungssoftware hausintern gewartet wird, hat der Anwender kaum eine Möglichkeit den Wartungsprozess zu straffen. Zu eng sind die Verflechtungen zwischen dem IT Personal und dem Fachpersonal. Es läuft alles informal ab. Oft erhalten die zuständigen Wartungsprogrammierer ihre Aufträge per Telefon oder im direkten Gespräch. Erst wenn die Software außer Haus ist, hat der Anwenderbetrieb die Möglichkeit, den Wartungsprozess richtig zu formalisieren und die Wartungsdienstleistungen nach Nutzen zu berechnen.

Die wichtigste Voraussetzung für einen effizienten Wartungsbetrieb ist, zwischen technischen Rahmen und fachlichen Inhalt unterscheiden zu können. Deshalb empfiehlt sich ein Schichtenmodell. Der Dienstleister ist vertraglich verpflichtet den Rahmensoftware – Datenbanksystem, Enterprise Bus und sonstige Middleware – bereitzustellen und zu erhalten. Das Budget dafür ist im Wartungsvertrag vorgesehen. Dazu kommt eine Serviceschicht allgemeingültiger fachlichen Funktionen, die von allen einzelnen Fachanwendungen gemeinsam benutzt werden. Diese könnten als Subroutinen, Klassenbibliotheken oder gar als Web Services implementiert werden. Ihre Erstellung und Erhaltung muss von den Gemeinkosten gedeckt werden.

In der Mitte befindet sich die Anwendungsschicht mit den Applikationen, sprich Geschäftsprozessen, der einzelnen Fachgruppen oder Fachdienste. Für jeden Geschäftsprozess wird ein Softwareprodukt definiert, bestehend aus einem oder mehreren Teilsystemen. Ein Teilsystem setzt sich aus Komponenten zusammen und Komponente bestehen wiederum aus Klassen oder Modulen samt deren Schnittstellen. Diese hierarchische Gliederung der Software – die so genannte Produktstruktur – ist streng einzuhalten (siehe Abbildung 2).

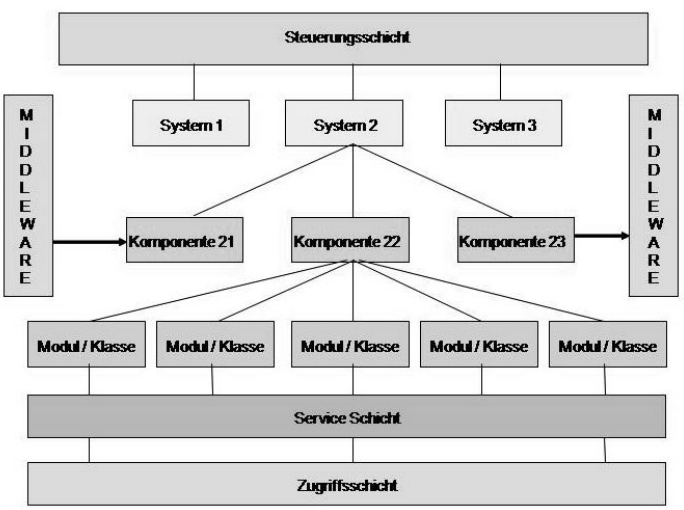


Abbildung 2: Struktur eines Softwareproduktes

Die jeweilige Fachgruppe hat das Recht jederzeit Wartungsaufträge zu stellen um ihr Produkt zu verändern. Die Wartungsaufträge sind in drei Klassen zu teilen:

- Fehlerberichte,
- Change Requests und
- Neue Anforderungen.

Der Dienstleister ist verpflichtet für jeden einzelnen Wartungsauftrag eine Kostenschätzung abzugeben. Sofern der Anwender dies akzeptiert wird der Auftrag durchgeführt und danach der Betrag bezahlt, den der Dienstleister vorgeschlagen hat. Die zuständige Fachabteilung muss wissen, was ihnen die Korrektur, Änderung oder Erweiterung wert ist. Wenn sie meint, die Kosten sind zu hoch kann sie den Auftrag stornieren. Kein Wartungsauftrag muss durchgeführt werden und wenn ja, muss die Fachabteilung bereit sein den vollen Preis dafür zu zahlen. Auch auf Fehlerkorrekturen kann verzichtet werden, wenn die Kosten der Fehlerbehebung zu hoch erscheinen. Es kommt vor allem darauf an Kostentransparenz zu schaffen und dies ist nur möglich wenn jeder Auftrag getrennt abgerechnet wird (siehe Abbildung 3).

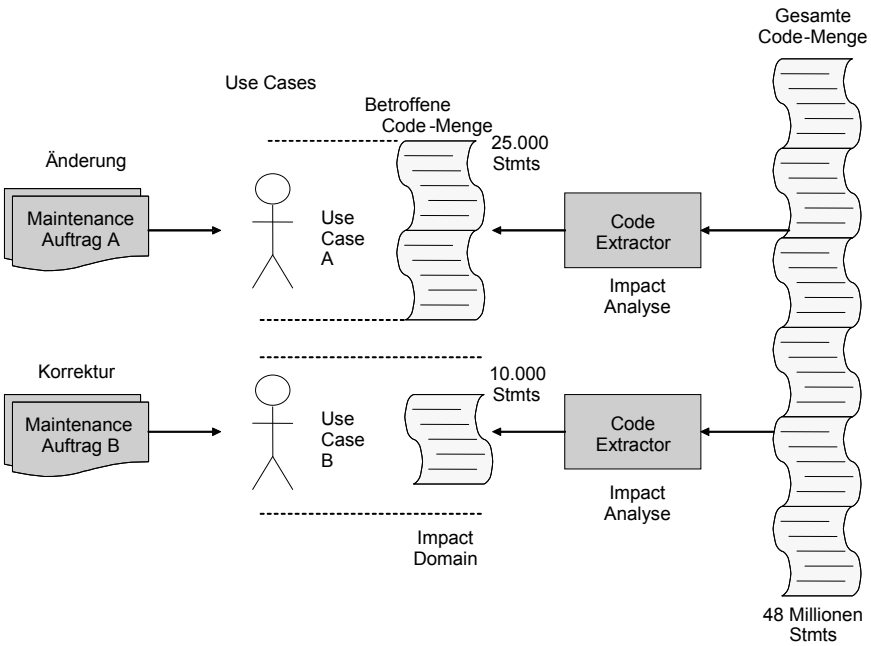


Abbildung 3: Verknüpfung der Wartungsaufträge mit dem Code

Zur Erleichterung der Kostenschätzung und der Wartungsdurchführung müssten alle Wartungsaufträge sich auf bestehende oder neue Anwendungsfälle – Use Cases – beziehen. Der Wartungsbetrieb ist verpflichtet die Beziehungen zwischen den Anwendungsfällen und den Komponenten, Klassen und Moduln in einer Repository zu pflegen. Mit Hilfe der Repository werden betroffene Codebausteine aus der Gesamtcodemenge abgezogen und zur näheren Analyse bereitgestellt. Diese Bausteine unterliegen anschließend einer automatisierten Impact-Analyse aus der ihre Größe und Komplexität hervorgehen. Die Größe lässt sich in Anweisungen, Object-Points oder Function-Points errechnen. Durch eine Verbindung zur Produktivitätsdatenbank können die Kosten der Wartungsaufträge schnell und relativ zuverlässig geschätzt werden [Sn04].

Sofern die zuständige Fachgruppe mit dem Preis und der Dauer einverstanden ist, ist der Dienstleister verpflichtet den Wartungsauftrag innerhalb der vereinbarten Zeit und zu den vereinbarten Kosten abzuwickeln. Wenn er dies nicht schafft, droht eine Konventionsstrafe. Der Anwenderbetrieb ist seinerseits verpflichtet die durchgeführten Änderungen abzunehmen. Dies kann auf zweierlei Weise geschehen, zum einen durch einen dynamischen Abnahmetest und, zum zweiten, durch eine statische Analyse des Codes und der Dokumente. Sie dürfen von der vereinbarten Qualitätsnorm, z. B. die Einhaltung gewisser Codierregel und die Erreichung gewisser Testüberdeckungsziele, nicht abweichen. Wenn ja, hat der Kunde das Recht die Änderungen abzulehnen und braucht dafür nicht zu bezahlen.

Da ein Wartungsbetrieb von einem Anwender allein nicht immer ausgelastet wird, empfiehlt es sich für den Wartungsbetrieb immer mehrere Systeme gleichzeitig zu pflegen. Dies bedarf einer besonders sorgfältiger Planung und einer strenger Organisation. Es versteht sich vom allein, dass der Outsourcing Partner alles daran setzen wird, seine Kosten zu minimalisieren und trotzdem seine vertraglichen Verpflichtungen einzuhalten. Das Ziel der Wartung ist es, einen gerade ausreichenden Wartungsservice zu möglichst niedrigen Kosten anzubieten.

4 Die Wartungsorganisation

Was die Organisation der Softwarewartung anbetrifft, empfiehlt es sich, den Wartungsbetrieb als Spiegelbild des Anwenderbetriebes zu organisieren. Für jede Fachabteilung sollte es eine entsprechende Wartungsprojektgruppe geben. Diese fachlich ausgerichteten Projektgruppen sind für die Wartung und Weiterentwicklung der fachlogischen Komponente zuständig. Die fachlogischen Komponenten sind in dem einheitlichen technischen Rahmen, bzw. in der gemeinsamen Architektur, eingebettet. Die Architektur wird von einer separaten Gruppe gepflegt. Sie besteht aus mehreren Schichten, darunter eine Prozess-Steuerungsschicht, eine gemeinsame Serviceschicht und eine Datenverwaltungsschicht. Es ist wichtig, dass sich die Architektur unabhängig von den Anwendungskomponenten weiterentwickeln lässt. Deshalb muss der fachliche Inhalt von der technischen Verpackung getrennt bleiben (siehe Abbildung 4).

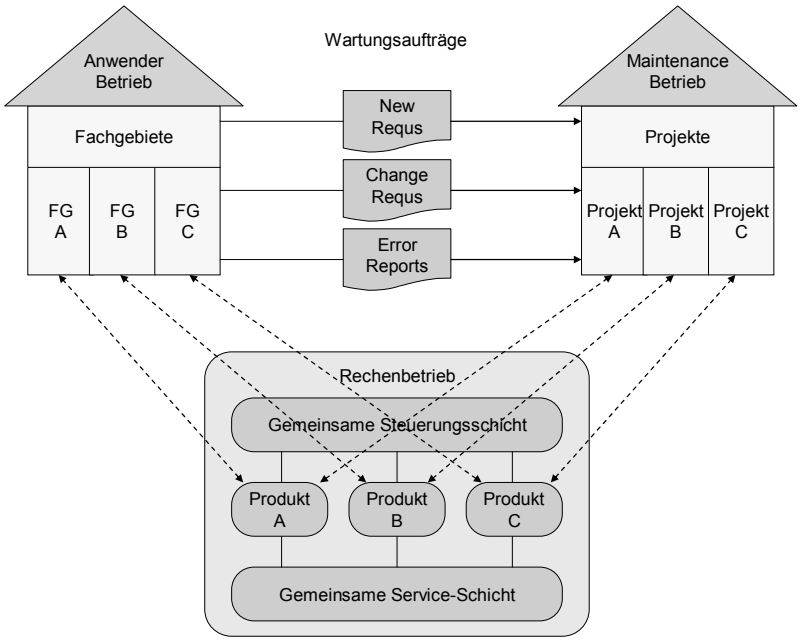


Abbildung 4: Wartungsorganisation entspricht der Software-Architektur

5 Das Service Level Agreement (SLA)

Die rechtliche Grundlage für den oben geschilderten Wartungsprozess ist das Service Level Agreement – eine rechtsverbindliche Vereinbarung zwischen dem Anwenderbetrieb und dem externen Wartungsbetrieb [Bö08]. Die SLA für die Wartung legt die Rechte und Pflichten beider Parteien fest und schreibt den Prozess vor. Eine wichtige Ergänzung dazu ist der Eskalationsplan. Dieser regelt was passiert wenn Pflichten nicht eingehalten werden. Dazu gehört der Ausmaß der Geldstrafen und die Bedingungen für die Terminierung des Vertrages. Es wird z. B. vorgesehen, im Falle einer Termin- oder Kostenüberschreitung, dass der Auftraggeber einlenkt und mehr Zeit bzw. mehr Geld zubilligt. Wenn er aber dazu nicht bereit ist, ist der Auftragnehmer verpflichtet, die Strafe auf sich zu nehmen. Vorausgesetzt wird, dass alle Aufträge nach einer Norm spezifiziert werden. Der Auftragnehmer hat also durchaus das Recht, Aufträge aufgrund mangelnder Qualität abzulehnen, genau so wie der Auftraggeber das Recht hat durchgeführte Änderungen und Korrekturen abzulehnen, falls diese die vereinbarten Qualitätskriterien nicht erfüllen.

Den Qualitätsvorschriften kommt auf dieser Weise eine enorme Bedeutung zu. Im Gegensatz zu hausinternen Wartungsprozessen bei denen solche Qualitätsvorschriften nicht ernst genommen werden, spielen sie hier eine zentrale Rolle. Wer gegen sie stößt muss den Verstoß verantworten. Demzufolge wird auch die Qualität der Normen zunehmen. Beide Parteien – der Kunde wie auch der Dienstleister – müssen mit der Norm zur Gestaltung der Wartungsaufträge einverstanden sein. Das Gleiche gilt für die Norm zur Abnahme der durchgeführten Änderungen am Produkt. Der Auftraggeber muss sich streng an dieser Norm halten, wenn er die Änderungen abnimmt und darf nur Abweichungen von der Norm beanstanden. Dies klingt zwar sehr bürokratisch, aber nur so sind größere Konflikte zu vermeiden. So weiß jeder woran er ist [ZHB05].

Wenn also von Industrialisierung des Wartungsbetriebes die Rede ist, dann müssen als erstes die organisatorischen Rahmenbedingungen geschaffen werden. Dies kann wiederum am besten ein Outsourcing-Vertrag mit einem Service Level Agreement gewährleisten. Allerdings muss der Vertrag richtig formuliert sein, wie die folgenden Fallstudien zeigen.

6 Fallstudien in Maintenance Outsourcing

Hier werden zwei Fälle geschildert bei denen Anwendungssysteme an externe Outsourcing Partner übergeben worden. Im ersten Fall war das Ergebnis wegen der mangelhaften Rechtslage und des ungenügend geregelten Prozesses unbefriedigend. Im zweiten Fall kann man von einer echten Industrialisierung der Softwarewartung sprechen.

6.1 Die Arbeitslosengeldauszahlung ALU-II – ein missglücktes Outsourcing

Die ALU-II Software für die Auszahlung der Arbeitslosengelder ist ein Beispiel für missglückte Outsourcing-Projekte. Die Software wurde ursprünglich von einem kleinen Softwarehaus für die Bundesagentur für Arbeit entwickelt. Ehe es zum Abschluss der Entwicklung kam, ist die besagte Firma in Konkurs gegangen. Da T-Systems Generalunternehmer war, müsste sie die Software übernehmen und das System schlecht oder recht zu Ende bringen. Damit übernahm sie auch die Verantwortung für die Wartung und Weiterentwicklung des Systems. Seit der Freigabe der ersten Version Anfang 2005 gab es ständig Probleme. Schon gleich zu Beginn erwies sich das System als untauglich. So war es nicht möglich, die veränderten Krankenkassenbeiträge im System nachzuvollziehen. Folglich wurden monatlich 25 Millionen Euro zu viel an die Krankenkassen überwiesen. Der Fehlbetrag summierte sich am Ende des ersten Jahres auf 364 Millionen Euro [CW08].

Dies war jedoch nur der Anfang. Es gelang dem Lieferanten weder, die Fehler zu korrigieren noch die Änderungsanträge in einer vertretbaren Zeit durchzuführen. Demzufolge gibt es inzwischen mehr als 146 Umgehungslösungen, welche die Sachbearbeiter ausführen müssen. Dies verursacht den Verlust von durchschnittlich zwei Sachbearbeiterstunden pro Tag. Allein in einem Bundesland produzierten die Fehler der Software jährliche Zusatzkosten in Höhe von 230 bis 380 Millionen Euro. Bundesweit betrachtet sind das mehr als 2 Milliarden Euro pro Jahr. Gutachter haben festgestellt, dass das IT-System mehr kostet, als wenn die Beamten die Gelder manuell auszahlen würden. Das ALU-II Software-System ist also das Beispiel für einen negativen ROI. Schuld an der Lage ist nicht der Dienstleister, Schuld sind die Verträge. Es wurde versäumt, die Pflichten und Rechte beider Partner genau festzulegen. Folglich handelte jeder nach der eigenen Interessenslage.

Sollte es je zu einem neuen Vertrag kommen, muss der Wartungsprozess und die einzelnen Wartungsdienste genauer spezifiziert werden. Es muss auch jede Wartungsleistung vorher kalkuliert und entsprechend vergütet werden. Falls die Fehlerkorrekturen und Änderungsanträge nicht korrekt oder nicht zeitgerecht durchgeführt werden, muss der Auftragnehmer für den Schaden aufkommen. Hinzu kommt, dass die ALU-II Software hier in Deutschland ohne ausreichende Automatisierung fortentwickelt wird. Wenn solche Arbeiten in einem Hochlohnland wie Deutschland verrichtet werden, müssen sie im hohen Grade durch automatisierte Werkzeuge unterstützt werden, so wie in der zweiten Fallstudie der Fall ist. Eine Industrialisierung des Software-Managements setzt nicht nur eine solide Vertragsbasis voraus. Sie verlangt auch ein hohes Grad an Automatisierung.

6.2 Das GEOS Wertschriftenverwaltungssystem - ein Beispiel für den industrialisierten Wartungsbetrieb

Das GEOS System für die Verwaltung von Wertschriften bei den österreichischen Banken liefert ein Beispiel dafür, wie komplexe Systeme gemanaged werden sollten. Dort gibt es einen ausgereiften Wartungsprozess, unterstützt durch zahlreiche automatisierte Werkzeuge. Der Code wurde mehrfach saniert und optimiert. Die 7,5 Millionen Code-Zeilen wurden auf 5,5 Millionen bzw. auf 2,2 Millionen Anweisungen reduziert. Die mittlere Komplexität der Module ist von 0,76 auf 0,59 gesunken und die mittlere Qualität von 0,52 auf 0,63 gestiegen. Schließlich wurde die Wartungsmannschaft von 250 auf 180 Mitarbeiter reduziert. Alle Fehlerkorrekturen und Änderungsanträge werden in einem gemeinsamen Repository abgelegt und analysiert. Eine automatische Auswirkungsanalyse ermittelt die Größe der Auswirkung und setzt dies in einen Aufwand um. Dadurch bekommt der Kunde immer sofort ein Angebot bezüglich der Kosten seiner Änderungswünsche. Ist er damit einverstanden, fließt der Änderungsantrag in das nächste Release ein. Neue Releases folgen in Intervallen von drei Monaten. Fehler werden innerhalb von 48 Stunden korrigiert und getestet. Die Korrekturen werden innerhalb der 48 Stunden auf dem System wirksam. Inzwischen ist die jährliche Fehlerrate unter 0,0002 bzw. auf 2 Fehler pro 10.000 Anweisungen gesunken [BS03].

Der Regressionstest ist zu 100% automatisiert. Nicht nur die Dialoge werden von Robotern automatisch bedient, auch die Batch-Prozesse werden automatisch angestoßen und überwacht. Am Ende werden alle Ergebnisse, sowohl die Bildschirmausgaben als auch die Datenbankinhalte automatisch abgeglichen und fehlerhafte Ergebnisse ausgewiesen. Dazu wird die Testüberdeckung gemessen und eine Teststatistik aufbereitet. Zuständig für die Wartung und Weiterentwicklung vom GEOS ist übrigens das gleiche Haus wie für die Wartung und Weiterentwicklung der ALU-II Software. Das bezeugt wie schwer es ist, einheitliche Wartungsprozesse in einem Unternehmen durchzusetzen (siehe Abbildung 5).

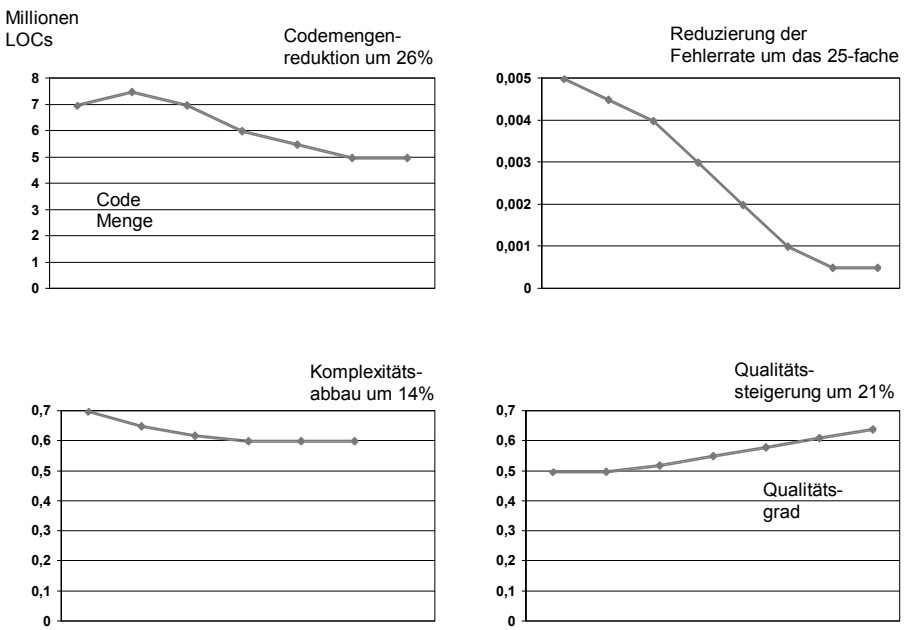


Abbildung 5: Monitoring des GEOS Wartungsbetriebes

7 Zusammenfassung

Das GEOS Projekt bezeugt, dass eine Industrialisierung des Software-Managements durchaus möglich ist. Es ist vor allem eine Frage der Organisation und der Verträge. Wenn sie stimmen, folgen die anderen Maßnahmen fast von alleine:

- Der Wartungsprozess wird optimiert;
- Die Wartungsumgebung wird modernisiert;
- Das Wartungspersonal wird qualifiziert;
- Das Wartungsprodukt wird ständig saniert.

Die regelmäßige Sanierung der Software wird zum festen Bestandteil des Produktlebenszyklus, ebenso, wenn nötig, die Migration auf eine neue Plattform. Wichtig ist, dass die Wartung von einem separaten Dienstleistenden nach fest vereinbarten Kostensätzen auf der Basis eines detaillierten Service Level Agreements gewährleistet wird. Das Ziel muss es sein, den laufenden Prozess ständig zu überwachen und zu verbessern. Die Verbesserungen müssen messbar sein. Deshalb besteht die Notwendigkeit einer Metrik-Datenbank, mit deren Hilfe der Dienstleistende und der Kunde die Qualität der Dienstleistung messen und vergleichen lassen. Industrialisierung erfordert Messbarkeit, erst recht für Software-Management.

Literaturverzeichnis

[Pi97] Pigoski, T.: Practical Software Maintenance – Best Practices for managing your software Investment, John Wiley & Sons, New York, 1997, S. 101

[Bo99] Boehm, B. et al.: Software Cost Estimation with COCOMO-II, Prentice-Hall, Upper Saddle River, N. J., 1999, S. 14

[Hu00] Hughes, B.: Practical Software Measurement, McGraw-Hill, Maidenhead, 2000, S. 47

[Ga05] Gartner Group: "Distribution of Costs in the IT-Industry", Gartner-Report Nr. 1005-4, Chicago, 2005

[BS95] Brodie, M.; Stonebraker, M.: Migrating Legacy Systems–Gateways, Interfaces & Incremental Approach, Morgan Kaufmann Pub., San Francisco, 1995, S. 6

[SAW08] Sneed, H.; Ackermann, E.; Winter, A.: Softwaremigration, dpunkt, Heidelberg, 2008, S. 15

[SS03] Sneed, H.; Sneed, S.: Web-basierte Systemintegration, Vieweg, Wiesbaden, 2003, S. 71

[Sn08] Sneed, H.: „20 Years of Software Reengineering – A Resume", Proceedings of 10th GI Workshop on Software Rreengineering, Ed. Gimnich, Kaiser, Quante, Winter, Bad Honeff, May 2008, S. 115

[Sn91] Sneed, H.: Economics of Software Reengineering, Journal of Software Maintenance, Vol. 3, Nr. 3, Sept 1991, S. 163

[Th95] Thomsett, R.: "Project Pathology – A Study of Project Failures, American Programmer, Vol. 8, Nr. 7, Juli 1995, S. 9

[Ca88] Calow, H.: "Practicing 3rd Party Maintenance", in Proc. of First European Workshop on Software Maintenance, British Centre of Software Maintenance, Durham, Sept. 1988, S. 110

[Ni08] Nicklisch, G.; Borchers, J.; Krick, R.; Rucks, R.: IT-Near- und –Offshoring in der Praxis, dpunkt, Heidelberg, 2008

[SHT05] Sneed, H.; Hasitschka, M.; Teichmann, M.-T.: Software-Produktmanagement, Wartung und Weiterentwicklung bestehender Anwendungssysteme, dpunkt, Heidelberg, 2005

[Sn04] Sneed, H.: „A Cost Model for Software Maintenance", Proc. of 20th International Conference on Software Maintenance, IEEE Computer Society Press, Chicago, Sept. 2004, S. 264

[Bö08] Böttcher, R.: IT-Servicemanagement mit ITIL V3, dpunkt, Heidelberg, 2008, S. 38

[ZHB05] Zarnekow, R.; Hochstein, A.; Brenner, W.: Service-orientiertes IT Management ITIL Fallstudien, Springer, Berlin, 2005, S. 30

[CW08] CW-Redaktion: "ALU-II Pannensoftware vor der Ablösung", Computer Weekly, Nr. 11, Feb. 2008, S. 8

[BS03] Broessler, P.; Sneed, H.: "Critical Success Factors in Software Maintenance", Proc. of ICSM-2003, IEEE Computer Society Press, Amsterdam, Sept. 2003, S. 190

Spezialisierung als Ausprägungsform einer Industrialisierung der Software-Branche – Eine Analyse am Beispiel der ERP-Software von SAP

Christian M. Wolf, Knut Geiger, Alexander Benlian, Thomas Hess, Peter Buxmann

Christian M. Wolf, Knut Geiger, Alexander Benlian, Thomas Hess

Institut für Wirtschaftsinformatik und Neue Medien
Ludwig-Maximilians-Universität München
Ludwigstr. 28, 80539 München
{wolf, benlian, thess}@bwl.lmu.de; mail@knutgeiger.de

Peter Buxmann

Fachgebiet Information Systems / Wirtschaftsinformatik
Technische Universität Darmstadt
Hochschulstr. 1, 64289 Darmstadt
buxmann@is.tu-darmstadt.de

Abstract: In industrialisierten Branchen wie dem Automobilsektor manifestiert sich die Industrialisierung aus Herstellersicht unter anderem in einer zunehmenden unternehmensexternen Spezialisierung. Ziel des Beitrags ist es daher zu untersuchen, ob sich eine solche Industrialisierung auch in der relativ jungen Software-Branche beobachten lässt. Diese Untersuchung wurde für das Wertschöpfungssystem um die ERP-Software von SAP auf Basis einer Inhaltsanalyse von Literatur und Experteninterviews durchgeführt. Insgesamt konnte ein steigender Grad an unternehmensexterner Spezialisierung v. a. für die Wertschöpfungsaktivitäten des Dienstleistungsgeschäfts ermittelt werden, wohingegen Forschung und Produktentwicklung als Kernkompetenzen nahezu vollständig bei SAP selbst verblieben.

1 Einführung

Ausgehend von der automatisierten Fertigung des hoch standardisierten Ford T-Modells durch spezialisierte Arbeitsträger stellt die Entwicklung des Automobilsektors unbestritten ein Musterbeispiel für die Industrialisierung einer gesamten Branche dar. Diese Industrialisierung manifestiert sich unter anderem in einer zunehmenden unternehmensexternen Spezialisierung aus Herstellersicht. Resultat dieser Entwicklung sind zunehmend sinkende Wertschöpfungsanteile der eigentlichen Fahrzeughersteller und somit sinkende Fertigungstiefen. Diese liegen bei den deutschen Fahrzeugherstellern bei zehn bis 25 % und weisen weiterhin eine fallende Tendenz auf [Kr05, S. 80ff]. Die dabei abgegebenen Wertschöpfungsanteile wurden insbesondere von spezialisierten Zulieferern für die Produktion, aber auch von nachgelagerten Stufen bei Vertrieb und Wartung der Fahrzeuge übernommen.

In letzter Zeit finden sich immer wieder Hinweise, dass sich auch in der Software-Branche eine Industrialisierung vollzieht. Wissenschaftliche Untersuchungen dazu finden sich aber bisher kaum. Vor diesem Hintergrund haben wir den Spezialisierungsgrad in verschiedenen Entwicklungsphasen des Wertschöpfungssystems um die ERP-Software von SAP untersucht. Diese Untersuchung ist nachfolgend dargestellt. In Kapitel zwei erfolgt zunächst eine Einordnung der Begriffe „unternehmensexterne Spezialisierung“ und „Industrialisierung“. Im dritten Kapitel stellen wir die Konzeption unserer Untersuchung und dabei insbesondere den zur Anwendung gekommenen Untersuchungsrahmen vor, der in Kapitel vier ausgefüllt wird. Der Beitrag schließt in Kapitel fünf mit einem Fazit und einem Ausblick auf den weiteren Forschungsbedarf.

2 Unternehmensexterne Spezialisierung und Industrialisierung

Industrialisierung ist ein Managementkonzept, das Ansatzpunkte zur kostenminimierenden und produktivitätssteigernden Massenproduktion liefert. Als wesentliche Ausprägungsformen werden eine verstärkte Standardisierung von Prozessen und Produkten, ein erhöhter Grad an unternehmensinterner und -externer Spezialisierung sowie eine Zunahme an Automatisierung gesehen [Bu90, S. 5; He91, S. 10; Sc94, S. 19ff]. Die drei genannten Ausprägungsformen sind nicht ohne Beziehung zueinander: Die Standardisierung ist gewissermaßen die Voraussetzung, um sowohl eine Spezialisierung als auch eine Automatisierung realisieren zu können. Eine zentrale Rolle im Konzept der Industrialisierung spielen Technologien. Sie eröffnen Spielräume für Standardisierung, Automatisierung und Spezialisierung und werden daher im Kontext der Industrialisierung häufig als Treiber bezeichnet [BDH08, S. 132].

Abbildung 1 fasst die genannten Elemente dieses Managementkonzepts im Überblick zusammen.

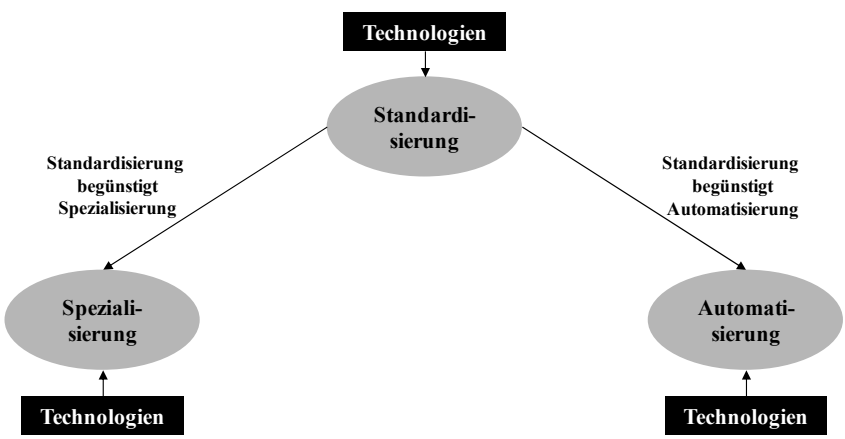


Abbildung 1: Treiber und Ausprägungsformen der Industrialisierung
[in Anlehnung an BDH08, S. 132]

Mit Industrialisierung geht ein Aufbrechen der Wertschöpfungsketten einher, was zu neuen Strukturen in der industriellen Produktion führt. Dies resultiert in einer Aufteilung der Funktionen und bedingt somit die Entstehung teilweise hoch spezialisierter Aufgabenträger [So05, S. 51]. Spezialisierung ist ein mögliches Resultat von Arbeitsteilung, die in der Literatur in Mengen- und Artenteilung unterschieden wird. Mengenteilung (auch horizontale Arbeitsteilung genannt) liegt vor, wenn Aufgaben auf mehrere organisatorische Einheiten verteilt werden und dabei jede Einheit dieselbe Teilaufgabe erfüllt. Artenteilung (auch vertikale Arbeitsteilung genannt) liegt hingegen vor, wenn eine Gesamtaufgabe in mehrere Teilaufgaben unterschiedlicher Art zerlegt wird, die von verschiedenen spezialisierten Akteuren wahrgenommen werden. Nur die Artenteilung führt demnach zu inhaltlichen Merkmalsunterschieden der zu erfüllenden Teilaufgaben und somit zu einer Spezialisierung der beteiligten Akteure [Zä89, S. 157].

Spezialisierung selbst lässt sich wiederum unterteilen in unternehmensinterne und unternehmensexterne Spezialisierung. Bei Ersterer werden die aus der Zerlegung komplexer Aufgaben resultierenden Teilverrichtungen in spezialisierten Organisationseinheiten innerhalb eines Unternehmens gebündelt. Die unternehmensexterne Spezialisierung hingegen bedeutet die Bündelung bestimmter Aufgaben in dafür spezialisierten externen Unternehmen. Spezialisierung verursacht einerseits einen Produktivitätsanstieg, bedingt aber andererseits einen Ressourcenverbrauch durch Tausch und Abstimmung. Aus ökonomischer Sicht gilt es, den daraus resultierenden Nettoeffekt zu maximieren [PDF05, S. 23].

Dieses Maximierungsproblem kann ausgehend von der Transaktionskostentheorie analysiert werden. Die Transaktionskostentheorie geht auf Arbeiten von [Co37] und [Wi85] zurück und basiert auf der Erkenntnis, dass die Koordination arbeitsteiliger Wertschöpfungsaktivitäten Kosten verursacht. Transaktionskosten fallen bei der Übertragung von Verfügungsrechten an [PDF05, S. 57]. Sie umfassen dabei die Kosten für die Anbahnung, Vereinbarung, Abwicklung, Kontrolle und Anpassung der Transaktion [PRW03, S. 49]. Die Transaktionskostentheorie lässt sich sowohl zur Bestimmung des optimalen Grads der Arbeitsteilung innerhalb eines Unternehmens als auch zwischen Unternehmen anwenden. Das Ziel einer auf Transaktionskosten basierenden Analyse ist es – bei gegebenen Eigenschaften der Transaktion und konstantem Produktionskostenniveau – die transaktionskostenminimale Koordinationsform einer Wertschöpfungsaktivität zu ermitteln. Damit sind Transaktionskosten ein Effizienzmaßstab zur Beurteilung und Auswahl unterschiedlicher institutioneller Arrangements. Hierfür kommen die Extrema Markt und Hierarchie sowie hybride Koordinationsformen in Frage [PRW03, S. 49f]. In einer hierarchischen Koordinationsform wird die Wertschöpfungsaktivität selbst vollzogen („make“), während sie auf der Gegenseite des Kontinuums über den Markt fremdbezogen wird („buy“). Zwischen diesen beiden Gegensätzen gibt es eine Vielzahl hybrider Koordinationsformen [St05, S. 47].

Fasst man die obenstehenden theoretischen Vorüberlegungen zusammen, so kann sich Industrialisierung in einer Branche in einem aus Transaktionskostensicht sinnvollen steigenden Grad an unternehmensexterner Spezialisierung manifestieren. Konkret bedeutet dies eine zunehmende Übertragung von einzelnen Wertschöpfungsaktivitäten an spezialisierte Anbieter. Derartige Fragen wurden traditionell für die industrielle Produktion und in den letzten Jahren auch verstärkt für IT-basierte Dienstleistungen etwa im Bankensektor untersucht [Ba05; SL05]. Zur Industrialisierung der Software-Branche findet sich hingegen bisher nur wenig Forschung.[1]

3 Konzeption der Untersuchung

Zielsetzung des vorliegenden Beitrags ist es zu untersuchen, ob sich eine Industrialisierung im Sinne einer zunehmenden unternehmensexternen Spezialisierung in der Software-Branche beobachten lässt. Da die Software-Branche auf Grund ihrer Vielfalt jedoch nicht als Gesamtes hinsichtlich dieser Fragestellung erforscht werden kann, müssen hierfür Teilbereiche abgegrenzt werden, die dann einzeln untersucht werden können. Auf Grund der Bedeutung und Bekanntheit sowie der vergleichsweise gut nachvollziehbaren Historie haben wir das Wertschöpfungssystem um die ERP-Software von SAP für unsere Untersuchung ausgewählt.

Um die Forschungsfrage untersuchen zu können, bedarf es eines geeigneten Untersuchungsrahmens, anhand dessen die unternehmensexterne Spezialisierung adäquat untersucht und visualisiert werden kann. Betrachtungsgegenstand sind die einzelnen Wertschöpfungsaktivitäten, die von den Unternehmen der Branche zur Herstellung und zum fortlaufenden Einsatz des Endproduktes beim Endkunden erbracht werden müssen. Ziel ist es, für jede Wertschöpfungsaktivität den Grad an unternehmensexterner Spezialisierung aus der Perspektive eines fokalen Unternehmens (in der Regel der Hersteller des Kernprodukts) darzustellen. Der Transaktionskostentheorie folgend wollen wir damit die Frage beantworten, ob eine Wertschöpfungsaktivität von diesem alleine bewerkstelligt, fremd über den Markt bezogen oder in Form einer hybriden Koordination partnerschaftlich erbracht wird. Für den konkreten Fall dieses Beitrags bietet sich eine Dreiteilung in die Stufen „SAP“ (als fokales Unternehmen), „Partner“ und „Markt“ an, da SAP ein umfangreiches Geschäft mit Partnern betreibt und eine schärfere valide Abgrenzung anhand der verfügbaren Quellen und der durchgeführten Experteninterviews nicht möglich erscheint. In die Kategorie „SAP“ fällt eine Wertschöpfungsaktivität, die SAP ohne Beteiligung spezialisierter Unternehmen selbst ausführt. Die Kategorie „Partner“ ist dadurch gekennzeichnet, dass die enthaltenen Wertschöpfungsaktivitäten von spezialisierten Unternehmen extern ausgeführt werden, die SAP explizit als Partner bezeichnet und mit denen somit eine längerfristige vertragliche Rahmenbeziehung für eine partnerschaftliche Zusammenarbeit besteht. In die Kategorie „Markt“ fallen Wertschöpfungsaktivitäten, die von am freien Markt tätigen spezialisierten Unternehmen ausgeführt werden, mit denen SAP keine längerfristige vertragliche Rahmenbeziehung eingegangen ist.

[1] Erwähnt sei an dieser Stelle beispielsweise das Sonderheft 2007 der WIRTSCHAFTSINFORMATIK zum Thema „Der Softwarestandort im Zeichen von Industrialisierung und Globalisierung“.

Die von Porter geprägte Darstellung von Wertschöpfungsketten unterstellt einen strikt sequentiellen Ablauf der Wertschöpfungsaktivitäten im Sinne eines Transformationsprozesses vom Rohstoff bis zum Produkt. Nicht nur deshalb ist umstritten, ob die Wertschöpfung in Branchen mit Dienstleistungsanteilen damit überhaupt sinnvoll abgebildet werden kann [AC93, S. 225]. Deshalb greifen wir auf eine Visualisierung in Anlehnung an die Illustration von Business Webs nach [Fr03] zurück, die sich insbesondere zur Darstellung komplementärer Systemleistungen eignen [Ze01, S. 180]. Allerdings stellen Business Webs die Analyse konkreter Unternehmen und ihrer Beziehungen in den Vordergrund und nicht die Klassifikation des externen Spezialisierungsgrads bei der Erfüllung einzelner Wertschöpfungsaktivitäten. Auch wenn Business Webs selbst daher nicht zur Erfüllung des Untersuchungsziels geeignet sind, erscheint ihre grundsätzliche Darstellungsform in kreisförmigen Schichten als sehr geeignet, da sie eine eingängige Visualisierung der Klassifikation des externen Spezialisierungsgrades bei der Erfüllung einer Wertschöpfungsaktivität ermöglicht. Sie wird deshalb für unseren Untersuchungsrahmen übernommen. Die Wertschöpfungsaktivitäten werden dabei nach dem Grad der externen Spezialisierung bei ihrer Erfüllung von „SAP“ über „Partner“ bis „Markt“ eingeteilt und von innen nach außen um den im Zentrum der Wertschöpfung stehenden Kunden herum positioniert. Die Positionierung ist dabei ordinalskaliert: Das heißt je weiter innen eine Wertschöpfungsaktivität verortet werden kann, desto geringer ist der Grad an unternehmensexterner Spezialisierung aus der Sicht des fokalen Unternehmens SAP. Mischformen zwischen Kategorien sind möglich, wenn Wertschöpfungsaktivitäten beispielsweise sowohl von SAP als auch von Partnerunternehmen wahrgenommen werden.

Ein Beispiel hierfür ist die Wertschöpfungsaktivität der Beratung: Große Konzerne werden aktuell hauptsächlich von SAP selbst, mittelständische Unternehmen hingegen größtenteils von SAP-Partnerunternehmen beraten.

Wird der Untersuchungsrahmen für verschiedene Zeitpunkte ausgefüllt, können diese verglichen und somit Veränderungen im externen Spezialisierungsgrad nachgewiesen werden. In Abbildung 2 ist der entworfene Untersuchungsrahmen exemplarisch dargestellt. Illustriert wurden innerhalb dessen drei exemplarische Wertschöpfungsaktivitäten, wobei Aktivität 1 von SAP selbst, Aktivität 2 von Partnerunternehmen und Aktivität 3 sowohl von Partnern als auch vom freien Markt übernommen wird.

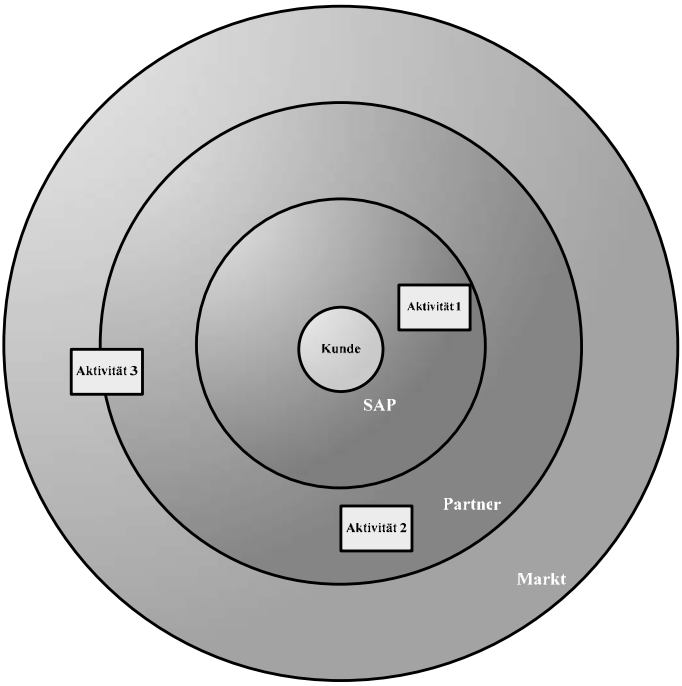


Abbildung 2: Exemplarische Darstellung des Untersuchungsrahmens

Aus methodischer Sicht basiert die zu Grunde liegende Untersuchung auf einer Inhaltsanalyse. Dazu wurde in einem ersten Schritt der zuvor vorgestellte Untersuchungsrahmen auf Basis der vorhandenen Literatur in Form einer argumentativ-logischen Deduktion für mehrere Phasen ausgefüllt. In einem zweiten Schritt wurde das erzielte Ergebnis dann verschiedenen Branchenexperten im Rahmen von leitfadengestützten strukturierten Interviews zur Korrektur vorgelegt. Um dabei eine einseitige Interessensvertretung zu vermeiden, wurde versucht, neben einem Vertreter der SAP AG auch Gesprächspartner aus Unternehmen der unterschiedlichsten SAP-Partnerkategorien und von unabhängigen Marktteilnehmern zu gewinnen.

Insgesamt konnten im Juni und Juli 2007 dreizehn persönliche bzw. telefonische Interviews geführt werden; elf davon mit SAP-Partnern der Kategorien ASP, Services, Software, Technologie Support, All-in-one, Business One und Hosting sowie jeweils ein Interview mit einem Vertreter der SAP AG selbst und mit einem unabhängigen Marktteilnehmer. Die Interviewpartner waren überwiegend in leitender Position tätig und verfügten zumeist über langjährige Branchenkenntnis.

4 Die vier Phasen der Aufgabenteilung im System R von SAP

Die SAP AG ist nicht nur das größte europäische Softwareunternehmen, sondern mit inzwischen über 100.000 Installationen bei mehr als 40.000 Kunden auch weltweiter Marktführer im Bereich von ERP-Software. Mehr als die Hälfte der weltweit 1.000 größten Unternehmen zählen zu ihren Kunden. Dabei lässt sich die heutige Position von SAP auf spezielle Leistungen hinsichtlich der Entwicklung von Standardsoftware, der Bereitstellung integrierter Lösungen sowie der Echtzeitverarbeitung zurückführen. Die Idee der Echtzeitverarbeitung war von Beginn an Kernaspekt und wurde in allen Generationen der R-Produktlinie umgesetzt. Aus ihr ergibt sich auch das Kürzel „R" im Produktnamen, das für „Realtime" steht [Le07, S. 5ff; BDH, S. 5f].

Für unsere weitere Analyse ist die Abgrenzung von Phasen hilfreich, die dabei grundsätzlich von der jeweils betrachteten Branche abhängt. In der Software-Branche bietet sich der Wechsel von einer Programmgeneration zur nächsten (z. B. System R/1 auf System R/2) als Festlegung der Phasentrennung an, da hier am ehesten signifikante Änderungen im Ausmaß der unternehmensexternen Spezialisierung zu erwarten sind. Für die vorliegende Untersuchung wurden die Phasen daher anhand der verschiedenen Generationen der R-Produktlinie von SAP gebildet. Abbildung 3 zeigt die identifizierten Phasen und ordnet diesen die jeweiligen Generationen der R-Produktline zu.

Phase	**I**	**II**	**III**	**IV**
Dominierender Zeitraum	1972-1980	1980-1990	1990-2000	ab 2000
Generation der SAP R-Produktlinie	System R/1	System R/2	System R/3	SAP ERP[2]

Abbildung 3: Generationen der SAP R-Produktlinie [HN05, S. 543; Ka00, S. 114]

Nach der Abgrenzung der zu untersuchenden Phasen werden im Folgenden die Ergebnisse unserer Inhaltsanalyse zur unternehmensexternen Spezialisierung für den Fall des Wertschöpfungssystems um die ERP-Software von SAP dargestellt.

[2] Bis 2007 wurde diese Generation der SAP R-Produktlinie unter dem Namen „mySAP ERP" geführt.

4.1 Phase I: SAP System R/1 (1972-1980)

Die heutige SAP AG geht auf die Firma "Systemanalyse und Programmentwicklung" zurück, die im April 1972 als Gesellschaft bürgerlichen Rechts von fünf ehemaligen IBM-Mitarbeitern gegründet wurde [Le07, S. 36]. Zu dieser Zeit entstand, getrieben von Unbundling und Softwarekrise, eine eigenständige Softwarebranche. Unbundling bezeichnet dabei die Trennung der preislichen Kopplung von Hard- und Software, die 1969 aufgrund eines bevorstehenden Regulierungsverfahrens gegen die Firma IBM wegen ihrer dominierenden Marktstellung vollzogen wurde [Ca03, S. 109ff; Le07, S. 36]. Dadurch wurden die im Softwaremarkt existierenden Eintrittsbarrieren gesenkt. Die Softwarekrise entstand aus der zunehmenden Komplexität von Software, welche durch die immer leistungsfähigere Hardware ermöglicht wurde – gleichzeitig aber das Problem der Bewältigung dieser Komplexität hervorbrachte [Di72, S. 861].

Während ihrer Tätigkeit bei IBM registrierten die späteren SAP-Gründer, dass sich Prozesse wie z. B. Lohnbuchhaltung in verschiedensten Unternehmen sehr ähnlich darstellten. Daraus resultierte die Idee, standardisierte Lösungen in Form von Standardsoftware anzubieten [Ra06, S. 87]. Problematisch für die neu gegründete Firma SAP war es, dass sie nicht über die finanziellen Mittel verfügte, um eigene Rechner und Büroräume zu beschaffen [Me99, S. 34f]. Diese wurden der jungen SAP von ihrem ersten Kunden zur Verfügung gestellt [Le07, S. 37f]. Bei diesem Kunden waren die SAP-Gründer bereits zu ihrer IBM-Zeit damit betraut, Individualsoftware einzurichten. Es gelang ihnen, das Unternehmen derart vom Konzept der Standardsoftware zu überzeugen, dass sie die dort vorhandenen Ressourcen auch für ihre eigene geschäftliche Tätigkeit nutzen konnten.

Das entstandene erste SAP-System R/1 war geprägt von einzelnen Funktionen wie Finanzbuchhaltung (System RF), Materialwirtschaft (System RM) und Anlagenbuchhaltung (System RA) [HN05, S. 543f]. Aufgrund knapper personeller Ressourcen mussten alle Gründer ein breites Spektrum an Aktivitäten übernehmen, die nach Aussage des Gründers Hasso Plattner von der Softwareentwicklung über die Beratung bis zum Verkauf reichten [Pl00, S. 21]. Trotzdem es zu dieser Phase der SAP-Historie nur wenige Quellen gibt, kann aufgrund der Aussage Plattners konstatiert werden, dass die SAP zu Beginn ihrer Geschichte alle Wertschöpfungsaktivitäten selbst erbracht hat.

Zu den genauen Wertschöpfungsaktivitäten der ERP-Branche finden sich keine Quellen. Es kann jedoch davon ausgegangen werden, dass diese auf einem höheren Aggregationsniveau den generischen Wertschöpfungsaktivitäten der Software-Branche entsprechen. Wie in Abbildung 4 dargestellt, wurden die generischen Wertschöpfungsaktivitäten der Software-Branche [Le04, S. 141] für unsere Untersuchung leicht angepasst.

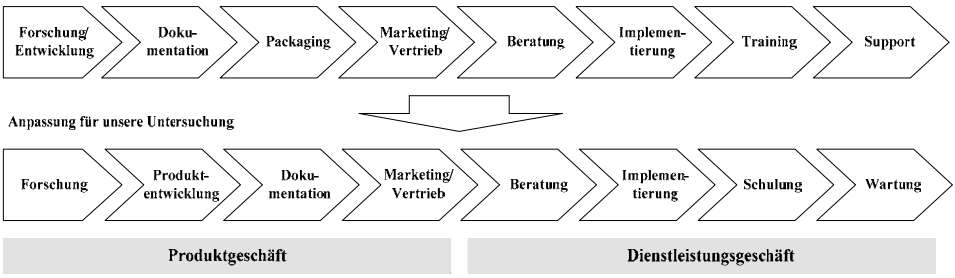


Abbildung 4: Generische [Le04, S. 141] und für unsere Untersuchung angepasste Wertschöpfungsaktivitäten der Software-Branche

So erschien es sinnvoll, die Wertschöpfungsaktivität Forschung/Entwicklung in die beiden Aktivitäten Forschung und Produktentwicklung aufzuspalten. In Anlehnung an das Wasserfallmodell lässt sich unter Forschung die Ideengenerierung, die Durchführbarkeits- und Anforderungsanalyse sowie die grobstrukturierte Planung verstehen [Bo86, S. 30]. Davon abgrenzen lassen sich als Produktentwicklung dann der konkrete Produktentwurf, die Codierung und das Testen der Software. Durch diese Aufspaltung lassen sich Veränderungen im Spezialisierungsgrad des Produktgeschäfts genauer analysieren. Im Rahmen der Wertschöpfungsaktivität Dokumentation wird die Funktionsweise der Software erklärt und dieser z. B. in Form von Handbüchern beigelegt. Die Aktivität des Packaging würde grundsätzlich die Zusammenfassung des Softwareprodukts (auf einem Datenträger) und der Dokumentation in einem Paket umfassen [Le04, S. 141]. Da die ERP-Software von SAP im Gegensatz zu beispielsweise den Betriebssystemen oder Office-Paketen von Microsoft nicht in großen Stückzahlen über den Fachhandel distribuiert wird, kann die detaillierte Betrachtung dieser Aktivität im Rahmen der Untersuchung vernachlässigt und unter die darauffolgende Aktivität Marketing/Vertrieb subsumiert werden. Im Rahmen der Wertschöpfungsaktivität Marketing/Vertrieb wird das erklärungsbedürftige Softwareprodukt entsprechend vermarktet und abgesetzt.

Die bisher erläuterten Wertschöpfungsaktivitäten sind allesamt dem Produktgeschäft zuzuordnen. Im Rahmen des Dienstleistungsgeschäfts sind vier weitere Wertschöpfungsaktivitäten abzugrenzen. Die Beratung umfasst die Unterstützung des Kunden bei der Abbildung seiner Wünsche und Anforderungen vor der eigentlichen Implementierung [Le04, S. 141]. Im Rahmen der sich anschließenden Implementierung wird das Produkt beim Kunden installiert und eingeführt. Um die Übereinstimmung der Nomenklatur mit den SAP-Geschäftsberichten und -Dokumentationen zu wahren, wurde die Wertschöpfungsaktivtät Training in Schulung und Support in Wartung umbenannt. Schulung umfasst somit die Ausbildung des Kunden im Umgang mit der Software, während Wartung die Betreuung der installierten Software im laufenden Betrieb bedeutet. Aus der Erkenntnis, dass die Gründer zu Beginn der SAP-Geschichte alle Wertschöpfungsaktivitäten selbst erbracht haben, lässt sich für die Phase I die in der Abbildung 5 dargestellte Wertschöpfungsstruktur ableiten.

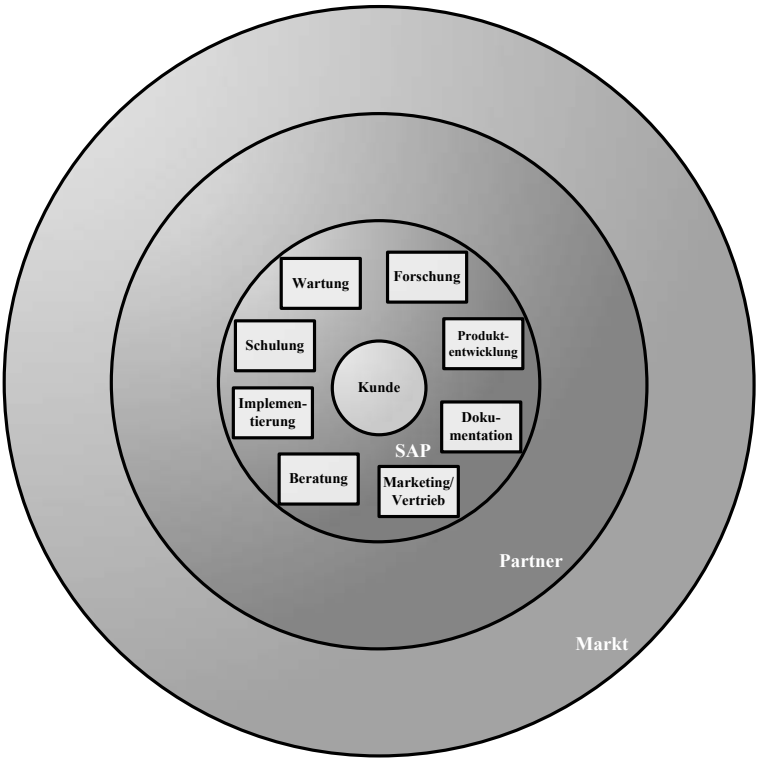


Abbildung 5: Wertschöpfungsstruktur in Phase I der ERP-Software von SAP

4.2 Phase II: SAP System R/2 (1980-1990)

Mit dem System R/2 wurde 1982 ein betriebswirtschaftliches Komplettpaket eingeführt, welches die Unterteilung in einzelne Funktionen ablöste [HN05, S. 543f]. Dieses war nicht mehr nur auf den Großrechnern von IBM einsatzfähig, sondern lief auch auf denjenigen von Siemens. Dadurch wurde die technologische und ökonomische Abhängigkeit von IBM verringert [Le07, S. 43]. Sinkende Hardwarekosten und steigende Rechnerkapazitäten werden in der Literatur als Auslöser für die stark gestiegene Nachfrage nach Standardsoftware genannt. Da bei jeder Installation der SAP-Software Fehler erkannt und beseitigt werden konnten, war diese im Vergleich zu den Eigenentwicklungen der Kunden weniger fehleranfällig. Da sie außerdem auf der aktuellsten Hardware, wie dem IBM Großrechner AS400 (der viermal schneller und zugleich viermal günstiger als das bis dato beste Gerät war) lief, wurde die SAP-Software zu einer günstigen und schnellen Alternative [Ra06, S. 88]. Um die deutlich gestiegene Nachfrage bewältigen zu können, überließ die SAP immer häufiger die Implementierung und Programmpflege ausgewählten Beratungsunternehmen [Me99, S. 52f]. Die Internationalisierungstendenz der SAP-Kunden, welche die SAP-Software auch in ihren ausländischen Tochtergesellschaften einsetzen wollten, war ein weiterer Treiber für die Initialisierung einer Kooperationsstrategie [Le07, S. 45f; Me99, S. 54f; Pl00, S. 24].

Eine bloße Übersetzung der Benutzerschnittstelle reichte nicht aus – vielmehr war zusätzlich Erfahrung betreffend der Besonderheiten in den entsprechenden Ländern, wie z. B. anderen Formen des Zahlungsverkehrs oder länderspezifischen Rechnungslegungsstandards notwendig, welche SAP nur bedingt aufwies. Ein Teil dieser Anpassungen konnte zwar von den Kunden selbst gelöst werden, jedoch übernahmen die Beratungsgesellschaften im Laufe der 80er Jahre zunehmend diese Funktion [Le07, S. 46; Me99, S. 54; He05, S. 533]. Hinsichtlich des Spezialisierungsgrads ändern sich in Phase II somit gemäß der Literatur insbesondere die Wertschöpfungsaktivitäten des Dienstleistungsgeschäfts: Beratung, Implementierung, Schulung und Wartung sowie auch Marketing/Vertrieb wurden neben der SAP nun auch von ihren Partnern übernommen. Im Rahmen der Experteninterviews ergab sich, dass die Partnerunternehmen in geringem Maße auch an der Produktentwicklung beteiligt waren. Aus diesem Grund wurde die Produktentwicklung im Endergebnis der Phase II zu einem kleinen Teil auch bei den Partnerunternehmen verortet. Hingegen wurden Wartung und Schulung in Phase II nach Aussage einiger Experten nahezu ausschließlich von SAP selbst bewerkstelligt, was für den Untersuchungsrahmen übernommen wurde. Abbildung 6 stellt die für Phase II ermittelte Wertschöpfungsstruktur grafisch dar.

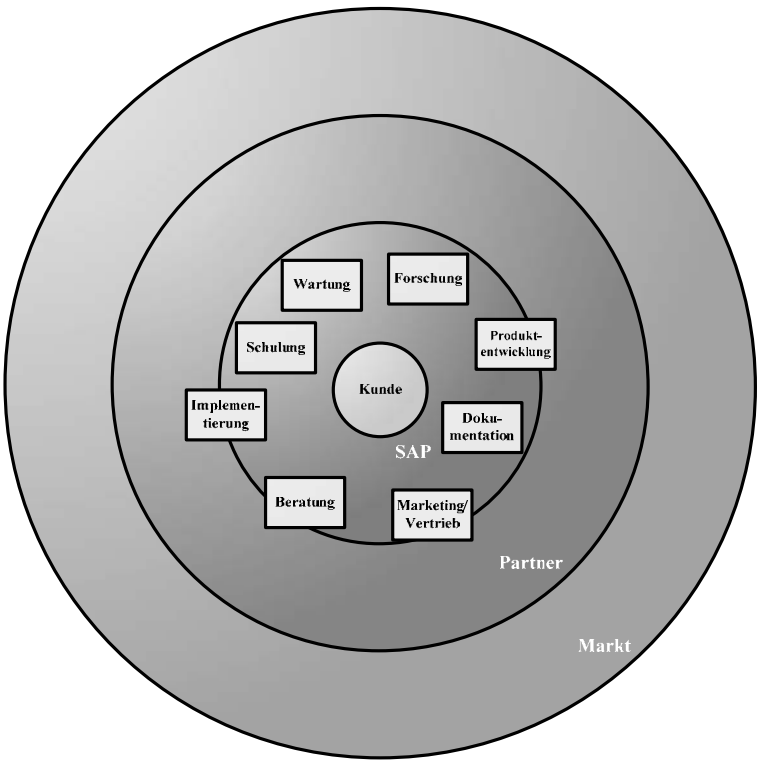


Abbildung 6: Wertschöpfungsstruktur in Phase II der ERP-Software von SAP

4.3 Phase III: SAP System R/3 (1990-2000)

In Phase II war die SAP geprägt von einem rapiden Wachstum. Um dies fortführen zu können, bestand ein erhöhter Finanzbedarf für die Entwicklung weiterer Software-Produkte und für die Umsetzung der angestrebten Internationalisierung [Le07, S. 48]. Der erfolgreiche Börsengang im Jahre 1988 deckte den Finanzbedarf und ermöglichte den Beginn der Entwicklung des Systems R/3 [Ra06, S. 88]. Aufgrund von Umsetzungsproblemen auf IBM-Großrechnern erfolgte kurzerhand eine pragmatische Umorientierung hin zu UNIX [Pl00, S. 33; Le07, S. 48f]. Das in UNIX enthaltene Client-Server-Konzept bot die Vorteile einer Drei-Schichten-Architektur. Im Zuge des eintretenden Bewusstseins für das sog. „Jahr 2000 Problem" entschieden sich viele Unternehmen für den Umstieg auf Standardsoftware, was oftmals ökonomisch sinnvoller als eine Umprogrammierung vorhandener Eigenentwicklungen war [Me99, S. 267ff].

Beide Effekte führten damit zu einem weiteren Wachstum der SAP AG. Dennoch kam es in Folge dieser Nachfragesteigerung auch zu einem erhöhten Wettbewerb auf dem Markt für betriebswirtschaftliche Standardsoftware. Die Entwicklung von Verkäufer- zu Käufermärkten ging einher mit einer erhöhten Anforderung der Kunden an Qualität, Stabilität, Portabilität und Integrationsmöglichkeiten der Software. Im Zuge dieser Entwicklung spezialisierte sich die SAP zunehmend auf ihre Kompetenzen in der Softwareentwicklung und ließ weitere Anteile des Dienstleistungsgeschäfts von Partnerunternehmen übernehmen [Ra06, S. 90]. Diese Entwicklung resultierte in einer weiteren Verschiebung der Wertschöpfungsaktivitäten des Dienstleistungsgeschäfts von SAP hin zu den Partnerunternehmen. Im Produktgeschäft wurden im Rahmen der komponentenbasierten Softwareentwicklung einzelne Klassenbibliotheken bis hin zu ganzen Anwendungskomponenten für das System R/3 von Partnerunternehmen erstellt oder über den Markt bezogen und über offene Schnittstellen integriert [Pl00, S. 168ff]. Die Wertschöpfungsaktivität der Beratung wurde nun auch von am Markt tätigen Unternehmen, die keine offiziellen SAP-Partnerunternehmen waren, angeboten.[3] Zudem entstand in dieser Phase ein breiter und von SAP unabhängiger Markt für Fachbücher. In den Experteninterviews erfolgte des Weiteren der Hinweis, dass SAP und auch Partnerunternehmen in Phase III die neue Wertschöpfungsaktivität des Application Service Providing (ASP) angeboten haben, die dem Kunden ein Outsourcing des kompletten Software-Betriebs offeriert. Im Endergebnis ergibt sich für die Phase III somit die in Abbildung 7 dargestellte Wertschöpfungsstruktur.

[3] Diese Erkenntnis ergab sich bei der Recherche nach potentiellen Interviewpartnern für die Experteninterviews.

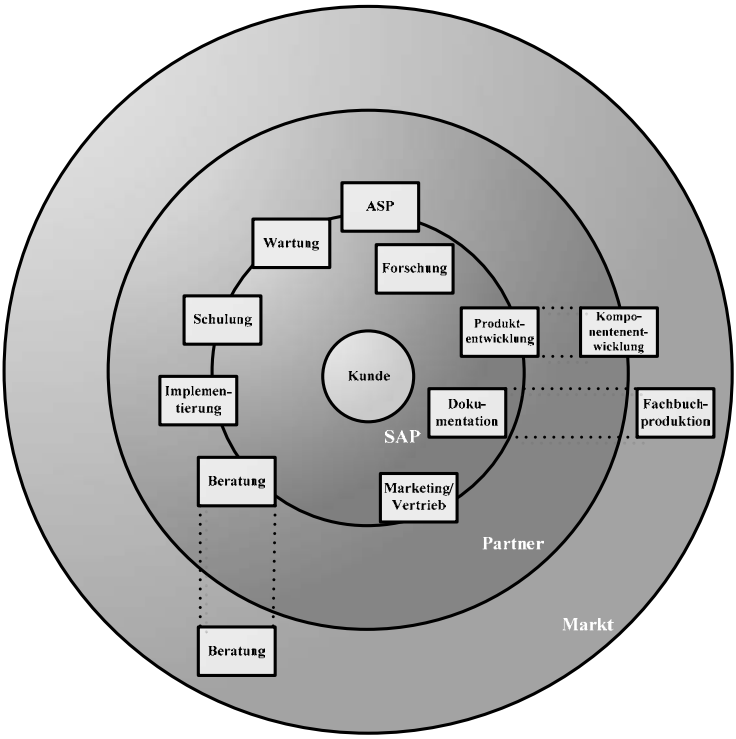


Abbildung 7: Wertschöpfungsstruktur in Phase III der ERP-Software von SAP

4.4 Phase IV: SAP ERP (ab 2000)

Nachdem das betriebswirtschaftliche Komplettpaket R/2 mit dem System R/3 um Branchenlösungen erweitert wurde, ist der wesentliche Entwicklungsschritt hin zur auf der NetWeaver-Plattform basierenden Produktgeneration SAP ERP durch eine Berücksichtigung der zunehmenden Verbreitung des Internets gekennzeichnet, denn es wurden E-Commerce Lösungen in das ERP-Komplettpaket integriert [HN05, S. 544]. Auch die Benutzeroberfläche – im System R/3 zwar schon grafisch, beim System R/2 noch rein textorientiert – wurde zu einem webbasierten Interface weiterentwickelt [HN05, S. 543]. In dieser Phase kamen neue, spezialisierte Wertschöpfungsaktivitäten hinzu, die über die zu Beginn des Kapitels erarbeiteten generischen Aktivitäten hinausgehen. SAP begann mit dem Betrieb des „mySAP Marktplatzes“, der einen internetbasierten Treffpunkt für Kunden, Partnerunternehmen und SAP selbst darstellte und insbesondere dem Austausch von Informationen und der Abwicklung von Geschäften dienen sollte [TRW00, S. 34ff].

Des Weiteren hat sich mit der externen Schulung am Markt eine weitere spezialisierte Wertschöpfungsaktivität entwickelt, die sich von der bisherigen Schulung, die SAP weiterhin mit einigen ausgewählten Partnern betreibt, abgrenzen lässt. SAP benennt außerdem die spezialisierte Wertschöpfungsaktivität der Independent Software Vendor (ISV) als neue Partner-Kategorie.

Diese Aktivität umfasst die Entwicklung und den Vertrieb unabhängiger Anwendungsprogramme, die sich mit der SAP-Software nutzen lassen. Ergebnis der Experteninterviews hinsichtlich Phase IV war, dass neben dem von SAP noch weitere Marktplätze von SAP-Partnern oder Anwendergruppen auf dem freien Markt betrieben wurden. Die sich somit für Phase IV insgesamt ergebende Wertschöpfungsstruktur ist in Abbildung 8 dargestellt.

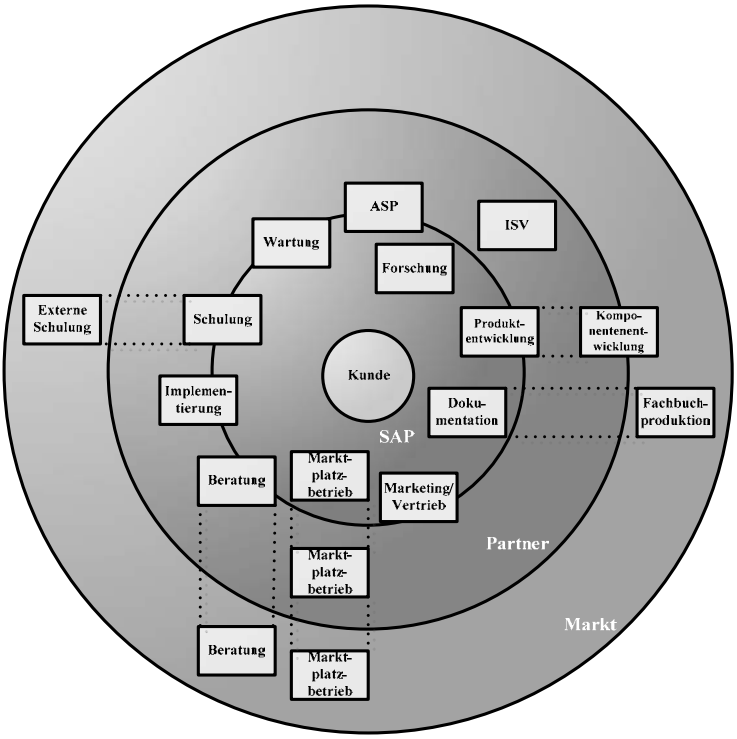


Abbildung 8: Wertschöpfungsstruktur in Phase IV der ERP-Software von SAP

5 Fazit

Ziel des vorliegenden Beitrags war es zu untersuchen, ob sich eine Industrialisierung im Sinne einer zunehmenden unternehmensexternen Spezialisierung in der Software-Branche beobachten lässt. Diese Untersuchung wurde für den Teilbereich des Wertschöpfungssystems rund um die ERP-Software von SAP mittels einer Inhaltsanalyse, die auf einer Literaturanalyse sowie qualitativen Experteninterviews beruht, durchgeführt. Betrachtet man den Untersuchungsrahmen über die vier abgegrenzten Phasen hinweg, so lässt sich feststellen, dass die monolithische Aufbaustruktur der Wertschöpfung bereits in relativ frühen Wachstumsphasen aufgebrochen wurde und der Grad an unternehmensexterner Spezialisierung aus der Sicht von SAP insbesondere im Bereich des Dienstleistungsgeschäfts (Beratung, Implementierung, Schulung, Wartung) stetig gestiegen ist.

Dies lässt sich grafisch deutlich in einem „nach außen Wandern“ der Wertschöpfungsaktivitäten von Phase zu Phase erkennen. In Phase II wurde dies insbesondere durch Partnerschaften zwischen SAP und unternehmensexternen, zumeist auf eine bestimmte Wertschöpfungsaktivität spezialisierten Dienstleistungsunternehmen realisiert. Ab Phase III wurde außerdem die Wertschöpfungsaktivität der Beratung und ab Phase IV auch die der Schulung nicht nur von SAP-Partnern, sondern auch von am freien Markt tätigen Unternehmen übernommen. Am stärksten von der externen Spezialisierung betroffen waren die Wertschöpfungsaktivitäten der Beratung und Implementierung. Als Ziele dieser Abgabe von Wertschöpfungsanteilen seitens SAP lassen sich insbesondere die Bewältigung des Wachstums und die Integration von nicht zu den Kernkompetenzen gehörigem spezifischem Wissen (z. B. länderspezifische Rechnungslegungsstandards, andere Formen des Zahlungsverkehrs) identifizieren.

Ein weiterer Hinweis für einen steigenden Grad an unternehmensexterner Spezialisierung in der Branche sind die in Phase III neu entstandenen, teilweise zu bestehenden Aktivitäten komplementären Wertschöpfungsaktivitäten, die hauptsächlich von spezialisierten Marktteilnehmern übernommen wurden. Dies sind die zur Produktentwicklung komplementäre Komponentenentwicklung, die zur Dokumentation komplementäre Fachbuchproduktion sowie das Application Service Providing. Grafisch zeigt sich der dadurch steigende Grad an unternehmensexterner Spezialisierung an der gestiegenen absoluten Anzahl von Wertschöpfungsaktivitäten. Im Rahmen der Forschung und Produktenwicklung erfolgte hingegen kaum eine unternehmensexterne Spezialisierung. Diese verblieben als Kernkompetenzen nahezu vollständig bei SAP selbst. An dieser Stelle schließt sich der Kreis zur in der Einleitung erwähnten Automobilbranche: Dort bestehen die bei den Fahrzeugherstellern verbleibenden zehn bis 25 % Fertigungstiefe auch zu großen Teilen aus Forschung und der Entwicklung neuer Modelle. Die zur Produktion eines Fahrzeugs notwendigen Komponenten werden – zumeist bis auf den Motor – dann aber von spezialisierten unternehmensexternen Partnerunternehmen oder über den freien Markt bezogen. Auch die Dienstleistungen der Beratung vor dem Fahrzeugkauf und der Wartung des Fahrzeugs werden nur noch zum Teil von zum Hersteller gehörigen Werksniederlassungen durchgeführt. Selbständige Autohäuser übernehmen als Partner der Fahrzeughersteller immer öfter die Beratung und Wartung. Letztere kann des Weiteren auch in einer Vielzahl am Markt tätiger freier Werkstätten erfolgen.

Für weitere Forschung in diesem Bereich bieten sich insbesondere zwei auf unserem Beitrag aufbauende Möglichkeiten an. Zum einen ließe sich unsere Untersuchung auf weitere Teilbereiche der Software-Branche, wie beispielsweise Computerspiele oder kommerzielle Open Source Software ausweiten, um zu überprüfen, ob dort ähnliche Entwicklungen feststellbar sind. Zum anderen könnte ein theoretischer Erklärungsversuch, beispielsweise auf Basis der Transaktionskostentheorie oder des ressourcenbasierten Ansatzes durchgeführt werden, um zu klären, warum es bei den einzelnen Wertschöpfungsaktivitäten zu einer zunehmenden unternehmensexternen Spezialisierung gekommen ist oder warum dies nicht der Fall war.

Literaturverzeichnis

[AC93] Armistaed, C.G.; Clark, G.: Resource Activity Mapping: The Value Chain in Service Operations Strategy. In The Service Industries Journal, 13. Jg., Nr. 4, 1993; S. 221-239.

[Ba05] Bartmann, D.: Die Industrialisierung des Bankbetriebs. Wiley, Weinheim, 2005.

[BDH08] Buxmann, P.; Diefenbach, H.; Hess, T.: Die Softwareindustrie: Ökonomische Prinzipien, Strategien, Perspektiven. Springer, Berlin, 2008.

[Bo86] Boehm, B.W.: Wirtschaftliche Software-Produktion. Edition Elch, Wiesbaden, 1986.

[Bu90] Budäus, D.: Industriebetriebe und Industriezweige. In: (Jacob, H. Hrsg.): Industriebetriebslehre. Gabler, Wiesbaden, 1990; S. 5-27.

[Ca03] Campbell-Kelly, M.: From airline reservations to Sonic the Hedgehog: a history of the software industry. MIT Press, Cambridge, 2003.

[Co37] Coase, R.H.: The Nature of the Firm. In: Economica N. S., 4. Jg., 11, 1937; S. 386-405.

[Di72] Dijkstra, E.W.: The Humble Programmer. In: ACM, 15. Jg., Nr. 10, 1972; S. 859-866.

[Fr03] Franz, A.: Management von Business Webs. DUV, Wiesbaden, 2003.

[He91] Heinen, E.: Industriebetriebslehre. Gabler, Wiesbaden, 1991.

[He05] Heinrich, C.E.: Strategiekonforme Organisationsgestaltung am Beispiel eines globalen Softwarekonzerns. In: Controlling, 17. Jg., Nr. 8/9, 2005; S. 531-536.

[HN05] Hansen, H.R.; Neumann, G. (2005): Wirtschaftsinformatik 1., Lucius & Lucius, Stuttgart, 2005.

[Ka00] Kagermann, H.: Strategische Unternehmensführung bei der SAP AG – Erfahrungen und Lösungen eines Software-Unternehmens. In: Wirtschaftsinformatik, 42. Jg., Nr. 2, 2000; S. 113-122.

[Kr05] Kretschmer, W.: Die Neuen Stärken. In: Brand Eins, 7. Jg., Nr. 4, 2005; S. 80-86.

[Le04] Leiteritz, R.: Open Source-Geschäftsmodelle. In (Gehring, R. A.; Lutterbeck, B. Hrsg.): Open Source Jahrbuch 2004, Lehmanns, Berlin, 2004; S. 139-170.

[Le07] Leimbach, T.: Vom Programmierbüro zum globalen Softwareproduzenten: Die Erfolgsfaktoren der SAP von der Gründung bis zum R/3-Boom. In: Zeitschrift für Unternehmensgeschichte, 52. Jg., Nr. 1, 2007; S. 33-56.

[Me99] Meissner, G.: SAP – die heimliche Software-Macht. Hoffmann und Campe, München, 1999.

[PDF05] Picot, A.; Dietl, H.; Franck, E.: Organisation. Schäffer-Poeschl, Stuttgart, 2005.

[Pl00] Plattner, H. et al.: Dem Wandel voraus. Galileo Press, Bonn, 2000.

[PRW03] Picot, A.; Reichwald, R.; Wigand, R.T.: Die grenzenlose Unternehmung. Gabler, Wiesbaden, 2003.

[Ra06] Rampe, T.: Gesunde Paranoia: SAP ist die erfolgreichste deutsche Unternehmensgründung der Nachkriegszeit. Warum eigentlich? In: Brand Eins, 8. Jg., 10, 2006; S. 84-93.

[Sc94] Schweitzer, M.: Industriebetriebslehre – Das Wirtschaften in Industrieunternehmungen. Vahlen, München, 1994.

[SL05] Sokolovsky, Z.; Löschenkohl, S.: Handbuch Industrialisierung der Finanzwirtschaft. Gabler, Wiesbaden, 2005.

[So05] Sokolovsky, Z.: Industrialisierung der Banken. In (Sokolovsky, Z.; Löschenkohl, S. Hrsg.): Handbuch Industrialisierung der Finanzwirtschaft – Strategien, Management und Methoden für die Bank der Zukunft. Gabler, Wiesbaden, 2005; S. 21-32.

[St05] Steiner, F.: Formation and early growth of business webs. Springer, Heidelberg, 2005.

[TRW00] Teufel, T.; Röhricht, J.; Willems, P.: SAP-Prozesse: Finanzwesen und Controlling. Addison-Wesley, München, 2000.

[Wi85] Williamson, O. E.: The economic institutions of capitalism – firms, markets, relational contracting. Free Press, New York, 1985.

[Zä89] Zäpfel, G.: Taktisches Produktions-Management. De Gruyter, Berlin, 1989.

[Ze01] Zerdick, A. et al.: Die Internet-Ökonomie: Strategien für die digitale Wirtschaft, 3. Aufl., Springer, Berlin, 2001.

Software Industrialization and Architecture Certification

Christoph Rathfelder, Henning Groenda, Ralf Reussner

Christopf Rathfelder, Henning Groenda

Software Engineering
FZI Forschungszentrum Informatik
Haid-und-Neu-Straße 10-14
76131 Karlsruhe
{rathfelder, groenda}@fzi.de

Ralf Reusser

Chair for Software Design and Quality
Institut für Programmstrukturen und Datenorganisation
Universität Karlsruhe (TH)
reussner@ipd.uka.de

Abstract: The industrialization of software development induces several changes to the development process as software development becomes distributed over company borders. They cooperatively develop individual components that are later assembled to software systems. This division of responsibilities requires a stricter quality assurance and in fact, creates a setting where the certification of software products becomes increasingly interesting. Until now, there are a few software product certification approaches, as in non-component-based software development processes, the considerable effort of software certification was only rarely justified. Therefore, existing certification approaches do not consider and support the requirements posed by industrialization, namely the separation of component development (by various providers) and system development. This paper presents a software certification approach which takes these requirements into account and allows certifying individual components as well as system architectures.

1 Introduction

The industrialization of software engineering is in an early stage. Other more mature and well-established engineering disciplines, like car manufacturing or civil engineering are more advanced regarding industrialization of products' production. Here, industrialization is a management approach to lower the costs per unit. However, the main difference between software development and the industrial production is that software is an immaterial good, which can be copied cheaply after its development.

Therefore, the approaches used in the production processes of more mature engineering disciplines cannot directly be mapped to software engineering. However, approaches lowering the costs in software development, deployment and management can be summarized in the process of the "industrialization of software", as discussed below.

One important part of software industrialization is the outsourcing of development tasks to specialized organizations, which means that the software development is distributed over several companies. This division of responsibilities requires a stricter quality assurance. In order to prevent conflicts of interest between the involved companies, a standardized quality assessment is often necessary. Ideally, the results of such quality assessments need to be comprehensible for the commissioning party. In reality, this understanding might require too many resources and is often not feasible, as component developers are often not willing to open a component black box to preserve specific knowhow. Due to this, it is interesting to let an independent third party perform the quality assessments. This independent institution assesses and certifies statements about the software system and its quality. Certification gains interest, if (a) the quality of an artefact is important but (b) hard to assess for an outsider. The latter can be the case due to the lack of resources, knowledge or information.

In the context of software engineering, certification is nowadays mainly used to certify the knowledge of individuals or the execution of mandatory processes and process steps. The certification of software products or artefacts of the development process has still not found a wide distribution. It is only used in a rather limited range in special domains, e.g. safety critical systems like control systems of nuclear power stations.

The contribution of this paper is the presentation of a software certification process. The certification consists of an architecture and component certification and allows the integration of several evaluation techniques. Furthermore, the paper includes an overview of software certification and software industrialization.

The paper is structured as follows. Section 2 presents the foundations of software certification and software industrialization. Section 3 shows the requirements on certification in the context of software industrialisation and presents an approach to realize this certification. Section 4 concludes the paper and provides an outlook.

2 Foundations

2.1 Software and Certification

In the domain of software engineering, three different types of certification are distinguished: certification of products, processes, and personnel. These three disjoined types form the Software Quality Certification Triangle [Vo99], whereas each type affects the development and therefore the quality of software systems.

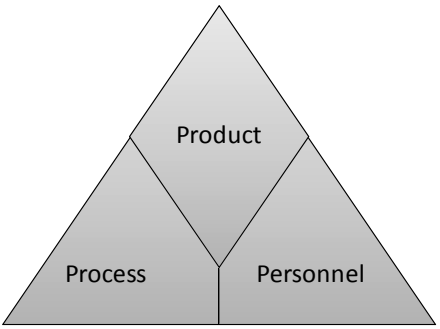


Figure 1: Software Quality Certification Triangle [Vo99]

The **certification of personnel** focuses on the knowledge and competence of individual engineers. Such a certificate can certify the presence of competences necessary to perform certain activities during the software development process. The Certified Tester and the Certified Professional for Requirements Engineering offered by the International Software Quality Institute (iSQI)[1] are two examples for these certifications. In addition to these more general certifications, there are a lot of technology and product dependent certificates available. The respective vendors are responsible for the definition of the curriculum. The certificate attests certified persons the required knowledge in a technology (e.g. Java) or a certain product (e.g. Microsoft Windows). Sun for example offers several different Java certificates within their Java Certification Program[2]. The knowledge helps a lot in handling the technologies or products. Nevertheless, a point of criticism of this certification type is that it is not certified that the certified knowledge is used during the development of a software system.

In contrast to the certification of personnel, the **process certification** is not linked to specific persons. It is rather valid for a department or a whole enterprise. To become certified, the enterprise has to prove that its processes are performed in a documented or specified way. The probably most well-known process certificate is the certification based on ISO9001 [In00] and its predecessor ISO9000, which defines the necessary activities of a general quality management process. Regarding software development, the Capability Maturity Model Integrated (CMMI) [Kn06], which is used to evaluate the software development process, is the most widely used process certification. However, there is no hard evidence that a better process automatically leads to a higher product quality. Even with a certified development process (e.g. CMMI Level 5) it is still possible that the resulting product has bad quality attributes [MW08].

The **product certification** focuses on a product and its related artefacts which were created during its development (e.g., test protocols, interface specifications, models). In the case of car manufacturing, the type approval is an example for a product evaluation and certification. Unfortunately, product certification has nearly no distribution in software engineering. It is used only in some special domains, for example safety critical systems within nuclear power station or airplanes.

[1] http://www.isqi.org
[2] http://www.sun.com/training/certification/java/index.xml

Although disjoined, all these types of certification affect the quality of software. Voas already mentioned in [Vo99] that a balanced combination of them may provide the best results and expressiveness. However, there is still no scientific knowledge on which is the right ratio. An example that combines product and process certifications is the Common Criteria (CC) [Cc07]. The CC is an internationally accepted standard that is used to certify the security of software systems. On the one hand, it defines mandatory activities that have to be performed during development and on the other hand it also includes an assessment of the final system.

Without certification, the software developer makes statements about its software product which the customer has to trust. Software certification aims at increasing the trustworthiness of statements about software. This is achieved by software assessments conducted by an independent third party. The resulting certification scenario and the participating roles are sketched in Figure 2 and described in the following. The certification authority assesses the product of the developer with a standardized and reproducible evaluation method in order to assure comparable evaluation results. A certificate is issued if the assessment leads to the conclusion that the statements are correct. Customers interested in the software can trust the certification authority instead of the developer himself as this is an independent authority which has a business model based on being trustworthy which makes them issuing wrong certificates unlikely.

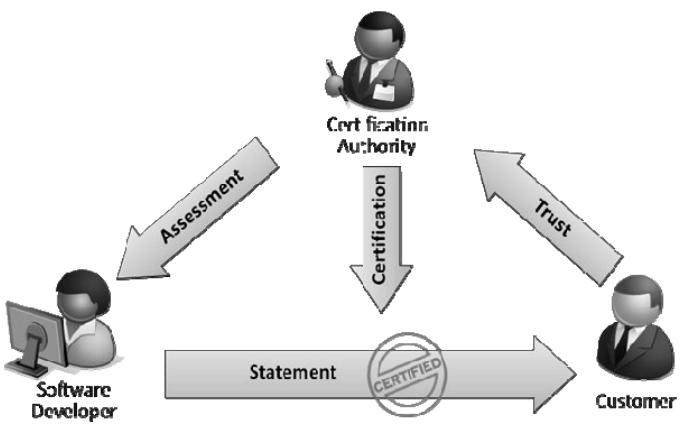


Figure 2: Software Certification Scenario

The acceptance and therewith the success of a certification initiative depends on several factors. One of most important factors is the independence and the trustworthiness of the certification authority. The trustworthiness, in turn, is affected by the performed certification. It has to be reproducible, which means, that a second certification of the same product yields the same results. This is of course the most important part of the trustworthiness.

2.2 Software Industrialization

Regarding the shift from manufacturing to industrial development and production, the software engineering discipline should learn from more mature engineering disciplines which already have undergone industrialization. Following alteration of these mature disciplines during their industrialization can be translated into software development:

- **Component-based development and standardization:**
  Car manufacturing is one example that has obviously a very high degree of standardization and reuse of the same components within several car models. For example, car engines are components which are reused within various car models of a brand, as this reduces the development cost for each individual car model. However, in order to allow flexible adoptions of the product, a standardization of the components respectively their interfaces is necessary. For example, electrical plugs must have the same measures and compatible cable connectors to allow the reuse and assembly of electronic car components.

- **Outsourcing and Offshoring:**
  Outsourcing and offshoring of development and production process steps are driven by specialization of work and are the second major ingredient of industrialization. Specialized companies are given specifications and entrusted with performing these processes steps or with delivering components. However, there is a slight difference between outsourcing and offshoring [Ta05]. In the case of outsourcing the main reason is focusing on the core competences of an enterprise, whereas offshoring is additionally driven by cost differences in a globalized market.

- **Model-based Quality Assessments:**
  In more mature engineering disciplines especially if they have undergone industrialization, an early quality assurance is an essential part of the development process. The quality of a product is thereby assessed on base of blueprints before the production of the product is started. Regarding bridge construction as an example, a structural engineer uses an architect's plan or model to calculate extra-functional properties e.g. the bearing capacity of a bridge *before* it is built. Car manufacturers likely use simulation techniques to conduct crash tests virtually on computers to check the quality of an engineered car, reduce costs, and speed up the development process by reducing the number of necessary prototypes.

With respect to software the approaches of component-based software development (CBSD) [SGM02] and also service oriented architectures (SOA) [KBS06] can be seen as first steps towards the industrialization of software development. Both of these approaches split complex software system into independent smaller parts with thorough requirement specifications - the components respectively services. The software system's architecture model describes the structure of the system and the interconnections between its elements [BCK99]. Due to the seclusiveness of the components and services it is possible to mandate other companies with their development. Quality assurance and in particular its proactive integration into software development processes is still a challenge in software engineering – in theory as well as practice. Especially the quality prediction and evaluation on base of models, respectively the software's architecture model, is a field in which software engineering is still immature compared to other engineering disciplines.

3 Certification and Software Industrialization

In this section we present how software certification should be accounted for in the software development process in order to support striving for software industrialization. Especially the distribution of the development process over several companies, caused by outsourcing and offshoring, requires a strict quality management. Certification of quality by an independent third party is an important part of this quality management.

As already mentioned above, component-based development and early model-based quality evaluations play an important role within software industrialization. The certification process should therefore allow an early quality assessment and certification on base of a system's architecture. This implies evaluations based only on the architecture model and the specification models of the assembled components. In so doing, flaws in the architecture can be detected faster and corrected without having to waste money for an inappropriate implementation first. The components themselves can be either bought from marketplaces or developed by contractors.

The evaluation of a software system's architecture without having a complete implementation needs model-based prediction and evaluation techniques as the system cannot be tested in this case. Each used component has to have a model of its behaviour, which can then be used for the architecture's assessment. The quality of the evaluation results is directly affected by the quality of the component models. Our approach therefore includes, in addition to the architecture certification, a component certification which certifies the correspondence of a component's model and its implementation. Each quality attribute is allowed to have a separate model, as long as the model of different components can be composed to allow inferences on the architectural level. If the component is not implemented yet the models can additionally be used to state the requirements for certain components in order to fulfil the requirements on the system level.

The distinction between architecture and component certification is an often neglected fact, although they differ in the certified statements. The first certifies that a modelled system will fulfil certain quality requirements; the later certifies that the quality model of a component is equivalent to a running component instance. We therefore propose to consider these as different parts of the certification process of software products. In the following, we describe these two parts in more detail.

3.1 Architecture Certification Process

Requirements posed on software systems differ dependent on the application scenarios of the software system. For example, a business information system typically requires only a certain percentage of all response times below a defined threshold. In contrast, a robot control system commonly requires hard deadlines, which means that all responses are below such a threshold. It must be mentioned, that even though most examples are based on performance of a systems, the certification process should allow the certification of other quality attributes (e.g., reliability, maintainability) as well. A certificate simply stating good or bad quality without taking the requirements into account is hence not desirable. For these reasons, the certification approach should be adaptable to allow the evaluation and certification with respect to different quality requirements. Nevertheless, the certification process should also support the usage of a standardized requirements catalogue which is defined for certain usage domains (e.g. business information systems) and thereby eases the comparison of different software systems within the same domain (e.g. a standardized response times for web applications).

The proposed approach to certify software architectures is illustrated in Figure 3. It consists of four activities which are described in the following in more detail.

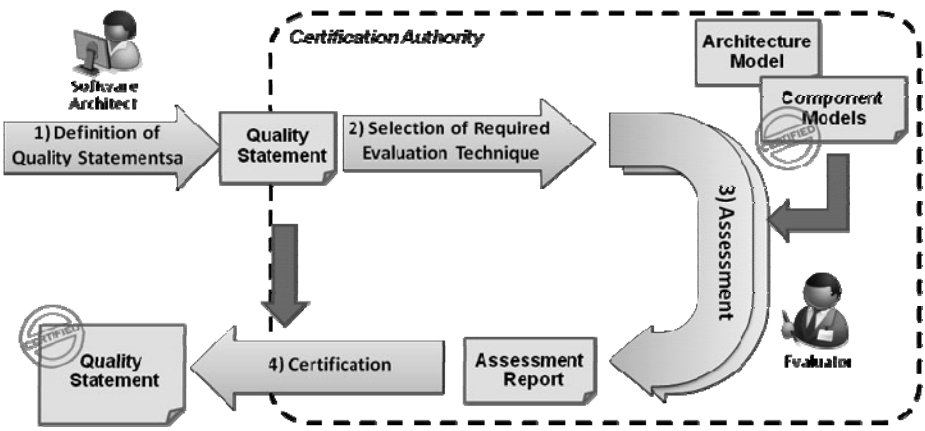


Figure 3: Architecture Certification Process

The certification of an architecture starts with the **Definition of Quality Statements** that should be certified. This is the only activity which can and should be influenced by the software architect. This step ends up with the document, which includes all *quality statements (QS)*. One example of a QS is: "For less than 6 concurrent users, the response time of the service A is below 3 seconds." As already mentioned above, it is also possible to include predefined QS in order to compare different systems of the same application domain. The use of a standardized language for the definition of these QS is necessary as this reduces semantic ambiguities. Only strict semantics of the certified QS avoids misinterpretations. Furthermore the standardization of the QS language eases the automatic processing, which is necessary to guarantee reproducible assessment results.

In the **Selection of Required Evaluation Technique** step, the QS are analyzed and the mandatory evaluation techniques are derived. This step is necessary because there is no evaluation technique available, that can be used for all quality attributes of a system based on its architecture. Even if only one quality attribute is regarded, there are different techniques which differ in the expressiveness of their results, their complexity, and the effort to perform the assessment. There are some quality attributes (e.g. performance) for which evaluation techniques are available that can be performed automatically. Two examples for tool-supported techniques to evaluate the performance of a system are SPE·ED [SW97] and the PCM-Bench [BKR08]. A high ratio of automation throughout the evaluation is expected to produce reproducible evaluation results. However, there are some quality criteria (e.g. maintainability) that still cannot be evaluated automatically. In these cases an evaluation expert is needed which analyzes the architecture based on his knowledge and experience. In order to achieve reproducible and correct results, the evaluation expert should be supported with additional utilities, for example checklists or guidelines. The selection of the adequate utilities for the evaluation expert is also part of this step. The effort for assessing QS can be prohibitive if the statements should be absolutely certainty. Falsification approaches systematically identifying possible counterexamples to invalidate the QS which can then be checked until a certainty threshold is reached. They are used to reduce the complexity although this negatively affects the certainty. The decision, if verification or extensive testing is required is also made in this step.

In the following **Assessment** step, the different evaluation methods are conducted and the results are logged. This step requires a description of the *architecture model* which is either transformed into an input model required by the tool-based and automated evaluation techniques or used by the evaluation expert as base of his analysis. This step additionally requires the already mentioned *component models*. As these models are generally provided by the component developer they have to be certified to guarantee that the model corresponds to the component's implementation. The therefore required component certification is explained in more detail in the next section. The result of this process step is an *Assessment Report* which includes the results of each evaluation.

In the last process step, namely **Certification**, the *assessment report* is checked. It is checked if the evaluation methods indentified in the second step have been performed. The respective assessment results are compared with the quality claims of the software architect, he has formulated as QS in the first step. If the assessment results substantiate these statements the QS are certified. A certificate thereby certifies only the compliance of the software with the quality statements. In order to have a comparable evidence of a system's quality, it is necessary to use QS which were standardized and predefined by the certification authority. The certificate is only valid for the assessed version of the architecture model and the component models and looses its validity if the architecture or components within are changed. Hence, a certificate must contain references to the assessed information.

3.2 Component Certification Process

The focus of the component certification process lies on certifying the correlation of a component's model and the component's implementation. These models are required during the evaluation of the architecture for assessing the QS and reason about the quality of the assembled system. Please note, that there should be at most one model for each quality attribute and in total there can be more than one model. The accuracy information can be seen as a kind of certification level – higher means better investigated but also with a higher assessment effort. This is the reason why the highest level is not always the best, as the reached accuracy has to be weigh against the resulting costs. The complexity of the assessment of the validity of a model for non-trivial components heavily depends on the quality attribute and can be hard or even impossible in practice. For this reason, a falsification approach is used. However, this means that the necessary effort to make a statement with a predefined certainty scales with the assessed degrees of freedom. The whole process for component certification and the participating roles are shown in Figure 4 and explained in the following.

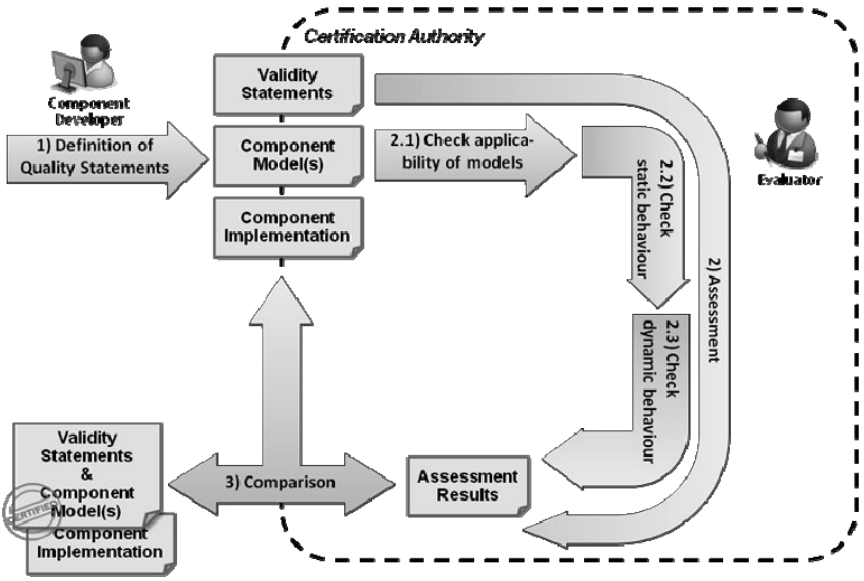


Figure 4: Component Certification Process

In **step 1**, the component developer defines *validity statements* for the component's behaviour *model(s)*. As mentioned above, these *validity statements* specify the range constraints and accuracy which influence the model's validity. For example, a mean response time below 2 seconds can only be provided for less than 10 concurrent users and in a defined environment. The decision which quality attributes should be certified and with which accuracy is the developer's choice, but he can base his decision on common domain-specific standards. The developer has to provide the necessary component model(s), its validity statement, and the implementation of the component to the certification authority.

The assessment in **step 2** can be split into three consecutive process steps. In the beginning in **step 2.1** the *applicability of the models* for the validity statements and the regarded quality attributes is checked. This is especially important with respect to architecture certification as models from different components must be composable to allow analyses on the architectural level. In **step 2.2**, models and implementation are checked in a static context. For example the provided and required interfaces of a component can be checked against their specification stated within the model. However, quality attributes like performance need to be checked in a dynamic context which is assessed in **step 2.3**. The assessment of dynamic behaviour requires a lot more effort for checking, as this is in most cases influenced by the usage profile, component configuration, required components, the used middleware, the operating system, and the hardware environment [BR06].

The costs and effort for provisioning the hardware and software environment for the component scale with the generality of the certificate. This makes test-beds desirable which allow varying these parameters. These test-beds can be realized for example with virtualization or simulation techniques. Software testing approaches [My04; EW01] can be applied for many quality attributes, e.g. performance, to gain confidence in the validity of the model. Most approaches are based on statistical testing, as exhaustive testing requires too much effort for non-trivial components. After step 2 all *assessment results* from the different process steps are available.

In certification **step 3** the decision is made if a certificate can be issued. Therefore the evaluation results are used to assess if the component model(s) are valid abstraction for the *implementation* with respect to the *validity statements*. An issued certificate has to be closely connected to the *component model(s)*, *validity statements*, and of course to the *implementation*. A packaging of model(s) and validity statements which only has a reference to the implementation is reasonable, as these are the interesting artefacts for architectural analyses. If models are proven to be invalid the software developer will not receive a certificate, but should receive feedback about the points of failure.

4 Conclusion and Outlook

This paper provided a short overview about software certification and software industrialization. It additionally pointed out the necessity of software certification for software industrialization. The certification approach sketched in this paper enables to certify the correctness of quality statements concerning quality attributes of a software systems. Based on this foundation, the need for a partition of software certification processes according to the certification on architecture and component level was presented.

The presented certification process allows the usage of user-defined just as well as standardised quality statements. This allows on the one hand using the certification as individual quality check. On the other hand the certification process provides a tool to compare different software systems. The approach certifies a software system based on its architecture and models of the included components. Thus, the certification can help in an early development stage to assess the quality of an assembled system although some components are still not implemented yet (but described in models).

In addition, the approach forms a framework which allows the integration of different software evaluation methods into one common quality certification process. The approach is thereby not limited to certain quality attributes and can be used to certify single quality attribute as well as a combination of different ones.

As future work we plan to develop a tool supported performance certificate. As a first step a language is developed which allows the definition of performance relevant quality statements. In parallel, we develop a test-framework for performance-models of components.

References

[BCK99] Bass, L.; Clements, P.; Kazman, R.:. Software architecture in practice. Addison-Wesley ; Bonn, 1999.

[BKR08] Becker, S.; Koziolek, H.; Reussner; R.: The Palladio Component Model for Model-Driven Performance Prediction. Journal of Systems and Software, To appear, 2008.

[BR06] Becker, S.; Reussner, R.: The Impact of Software Component Adaptation on Quality of Service Properties. Lóbjet, 12(1):105–125, 2006.

[Cc07] CCRA Members. The Common Criteria v3.1, 2007. http://www.commoncriteriaportal.org.

[EW01] El-Far, I.K.; Whittaker, J.A.: Model-Based Software Testing. In J. J. Marciniak, editor, Encyclopedia of Software Engineering. Wiley, 2001.

[In00] International Organization for Standardization. ISO 9001:2000, 2000.

[KBS06] Krafzig, D.; Banke, K.; Slama, D.: Enterprise SOA. Prentice Hall PTR, reprint. edition, 2006.

[Kn06] Kneuper, R.: CMMI. dpunkt, 2nd edition, 2006.

[MW08] Maibaum, T.; Wassyng, A.: A Product-Focused Approach to Software Certification. Computer, 41(2):91–93, Feb 2008.

[My04] Myers, G.J.: The Art of Software Testing. Wiley, Hoboken, NJ, 2nd edition, 2004.

[SGM02] Szyperski, C.; Gruntz, D.; Murer, S.: Component Software: Beyond Object-Oriented Programming. ACM Press and Addison-Wesley, NY, 2002.

[SW97] Smith, C.; Williams, L.: Performance engineering evaluation of object-oriented systems with SPE·ED. In Computer Performance Evaluation Modelling Techniques and Tools, pages 135–154, 1997.

[Ta05] Taubner, D.: Software-Industrialisierung. Informatik Spektrum, 28(4):292–296, 2005.

[Vo99] Voas, J.: Certification: reducing the hidden costs of poor quality. Software*, IEEE*, 16(4):22–25, 1999.

Ein Metamodell zur integrierten Darstellung von aktuellen Konzepten zur Industrialisierung der Softwareentwicklung

Oliver Höß, Jens Drawehn, Anette Weisbecker

Competence Center Software-Management
Fraunhofer-Institut für Arbeitswirtschaft und Organisation (IAO)
Nobelstraße 12
70569 Stuttgart
{Oliver.Hoess | Jens.Drawehn | Anette.Weisbecker}@iao.fraunhofer.de

Abstract: Durch die Notwendigkeit der Steigerung von Effizienz und Qualität in der Softwareentwicklung wurden in den letzten Jahren eine Reihe von Konzepten für die Industrialisierung der Softwareentwicklungsprozesse entwickelt. Beispiele hierfür aus dem eher technologisch orientierten Bereich sind Serviceorientierte Architekturen (SOA), Business Process Management (BPM), komponentenbasierte und modellbasierte Softwareentwicklung sowie Software-Produktlinien. Trotz einer Vielzahl von Arbeiten besteht in diesen Bereichen oftmals Unklarheit über die Bedeutung von einzelnen Begriffen sowie insbesondere über die Zusammenhänge zwischen diesen Konzepten. Der vorliegende Beitrag beschreibt einen Ansatz eines Metamodells für die integrierte Darstellung dieser Konzepte, in dem die wesentlichen Begriffe sowie die Beziehungen zwischen den Begriffen erläutert werden. Der Beitrag ist als Diskussionsgrundlage zu verstehen, um die Entwicklung einer Domänen-Ontologie für aktuelle Konzepte in Software-Management und -Technik voranzutreiben.

1 Einleitung

Die Erkenntnis, dass Software ein wesentlicher Bestandteil der Wertschöpfungskette der Unternehmen und somit ein zentraler Erfolgsfaktor für deren Wettbewerbsfähigkeit ist, hat sich bereits seit einiger Zeit durchgesetzt [Sp04].

Daraus folgt, dass eine Steigerung der Effizienz und Qualität der Softwareentwicklung und damit der resultierenden Software einen wesentlichen Beitrag zur Steigerung der Wettbewerbsfähigkeit der Unternehmen leisten kann. Ansätze hierfür sind dabei u. a. in der Industrialisierung der Softwareentwicklung durch den ingenieurmäßigen Einsatz von aktuellen Methoden, Technologien und Werkzeugen zu sehen [Ro03].

Beispiele von aktuellen Ansätzen, die ein großes Potenzial versprechen, sind Serviceorientierte Architekturen (SOA) und Business Process Management (BPM). Ein wenig länger sind bereits die Konzepte der komponentenbasierten Softwareentwicklung, der modellbasierten Softwareentwicklung und Software-Produktlinien im Gespräch.

Leider ist es im IT-Umfeld üblich, dass der Markt sehr von Schlagworten getrieben wird und Marketing-Interessen oft vor Inhalte gestellt werden. In Kombination mit fehlenden bzw. allgemein akzeptierten Definitionen (z. B. für SOA) werden dadurch oft Missverständnisse hervorgerufen und es werden unnötige Diskussionen geführt, z. B. bezüglich der trennenden Abgrenzung der genannten Konzepte. Aus Sicht der Autoren harmonieren jedoch die oben genannten Konzepte sehr gut und es besteht eine Vielzahl von Zusammenhängen.

Ziel des vorliegenden Beitrags ist es daher, die Konzepte SOA, BPM, komponentenbasierte Softwareentwicklung, modellbasierte Softwareentwicklung und Software-Produktlinien in einen Kontext zu bringen. Dabei wird ein ontologie-basierter Ansatz gewählt, d. h. es wird ein Metamodell definiert, bei dem die jeweiligen Entitäten über unterschiedliche Beziehungen miteinander verknüpft sind. Der Fokus dieses Beitrags liegt weniger in der Tiefe der Betrachtung der einzelnen Themenfelder, sondern in der Darstellung von Beziehungen zwischen den Begriffen aus unterschiedlichen Konzepten.

Nach dieser Einleitung werden in Kap. 2 existierende Arbeiten in diesem Umfeld ohne Anspruch auf Vollständigkeit kurz dargestellt. In Kap. 3 wird eine grobe Definition der Begriffe und Beziehungen im Themenfeld SOA als zentrales Konzept dieses Beitrags durchgeführt. In Kap. 4 wird dieses Basismodell schrittweise um Begriffe aus den Bereichen BPM, komponentenbasierte Softwareentwicklung, modellbasierte Softwareentwicklung und Software-Produktlinien ergänzt. Kap. 5 fasst den Beitrag zusammen und gibt einen Ausblick auf zukünftige Entwicklungen.

2 Existierende Arbeiten

In den für diesen Beitrag relevanten Bereichen wurden für sich genommen jeweils eine Vielzahl von Arbeiten durchgeführt, die die gesamte Palette von der eher grundlagenorientierten Forschung bis hin zur Anwendung in der Praxis abdecken [Hö07b; SW08; We02; SWH05; LSR07].

Im Bereich der Serviceorientierten Architekturen (SOA) hat die herstellerunabhängige Organisation OASIS (Organization for the Advancement of Structured Information Standards) ein SOA-Referenzmodell bzw. ein Metamodell veröffentlicht, das prinzipiell den Bereich SOA abdeckt, aber weder in der Tiefe und Qualität zufriedenstellend ist noch angrenzende Bereiche und Konzepte betrachtet [Oa06].

Auch in [En08] werden an mehreren Stellen Begriffsnetze im Kontext von Serviceorientierten Architekturen definiert, die auch die fachlichen Aspekte im IT-Architektur-Management berücksichtigen. Noch einen Schritt weiter gehen heute bereits Unternehmen, wie z. B. die EnBW (Energie Baden-Württemberg), die bereits ein sehr umfangreiches SOA-Metamodell entwickelt hat, das alle Aspekte von der serviceorientierten Analyse, über das Design bis hin zu Implementierung und Betrieb einer SOA inkl. der dazugehörigen Werkzeuge und Rollen abdeckt [SS07].

Auch im Rahmen der Gesellschaft für Informatik gibt es Bestrebungen, Begriffsnetze zu Themen der Softwaretechnik aufzubauen. Im Rahmen des Aufbaus des Informatik-Begriffsnetzes[1] wurden bereits drei Begriffssammlungen zu den Themen Analyse und Modellierung von Anwendungssystemen, Objektorientierte Modellierung und Vorgehensmodelle entwickelt [Bi07]. Zu dem aktuellen Thema SOA konnten an dieser Stelle jedoch keine Informationen und Erläuterungen gefunden werden.

3 Serviceorientierte Architekturen

Serviceorientierte Architekturen (SOA) sind ein derzeit sowohl in der Wissenschaft als auch in der Praxis stark diskutiertes Thema und versprechen im Gegensatz zu gewachsenen IT-Systemen und IT-Landschaften mehr Flexibilität und Effizienz sowie eine bessere Unterstützung von sich ändernden Geschäftsprozessen [Hö07a; Jo08].

Die Grundideen im Umfeld der Serviceorientierten Architekturen sind nicht neu, dennoch haben sich in den letzten Jahren durch offene Standards aus dem Web-Service-Umfeld sowie aus dem Workflow-Umfeld Möglichkeiten zur standard-konformen Realisierung einer SOA ergeben.

Im Folgenden sollen die wesentlichen und im Kontext dieses Beitrags relevanten Konzepte einer SOA kurz dargestellt werden (siehe auch Abbildung 1).

An zentraler Stelle steht das Konzept des *Service*, der eine wohldefinierte Funktionalität über standardisierte Schnittstellen (z. B. SOAP[2]) zur Verfügung stellt. Ein Service kann ein Basis-Service (*Basic Service*) sein, der relativ einfache Funktionalitäten bereit stellt oder ein zusammengesetzter Service (*Composite Service*), der die Funktionalitäten mehrerer *Basic Services* oder *Composite Services* nutzt und daher aus diesen aggregiert wird.

Ein wesentlicher Vorteil einer SOA in der vollen Ausbaustufe ist, dass die einzelnen *Services* nicht manuell, z. B. mit Java oder .NET-Code, zu *Composite Services* aggregiert werden. Stattdessen wird ein *IT-Prozessmodell*, das den zu realisierenden *IT-Prozess* beschreibt, direkt genutzt, um einzelne *Services* zu höherwertigen *Composite Services* zu aggregieren. Dieser Prozess wird Orchestrierung genannt und wird in den derzeit am Markt verfügbaren Produkten vor allem durch den Einsatz des Standards BPEL (Business Process Execution Language[3]) unterstützt. Dadurch kann eine prozessgesteuerte IT realisiert werden (siehe auch 4.1).

[1] http://public.tfh-berlin.de/~giak/
[2] ursprünglich Simple Object Access Protocol (http://www.w3.org/TR/soap/)
[3] http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

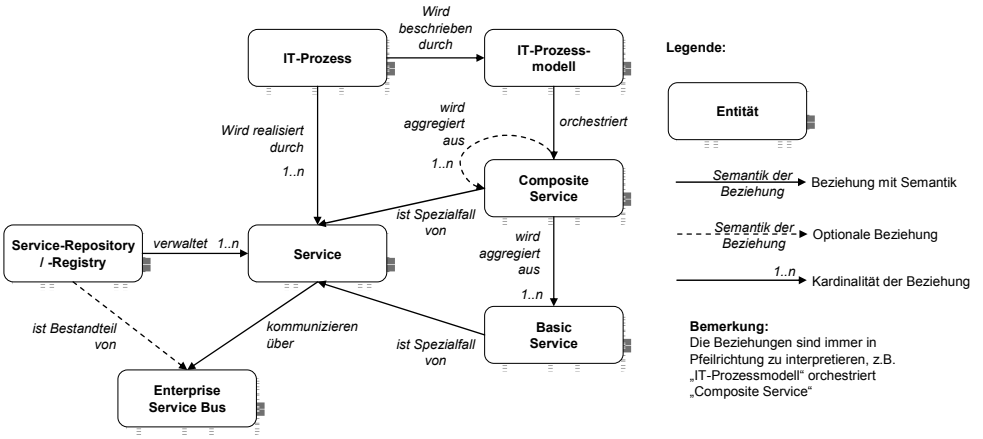


Abbildung 1: Grundkonzepte im SOA-Umfeld

Die Services werden in einem *Service-Repository* bzw. einer *Service-Registry* verwaltet. Oftmals werden *Registry* und *Repository* dadurch unterschieden, dass in der *Registry* vor allem technische Informationen (z. B. Schnittstellen) gespeichert sind, die zur Laufzeit benötigt werden und im *Repository* zusätzliche Meta-Daten bzgl. den einzelnen Services. Im Kontext dieses Beitrags werden die beiden Konzepte jedoch zusammengezogen, was in der Praxis auch oft durch dadurch abgebildet wird, dass sie durch identische Produkte abgebildet werden.

Die *Services* kommunizieren über einen sog. *Enterprise Service Bus (ESB)*, der eine einheitliche Kommunikations- und Sicherheits-Infrastruktur sowie Adapter und Transformationsfunktionalitäten beinhaltet. Ein ESB bildet das Rückgrat einer SOA und muss daher auf Performanz und Ausfallsicherheit ausgelegt sein. Oftmals ist auch ein *Repository* bzw. eine *Registry* ein Bestandteil des *Enterprise Service Bus*.

4 Integration weiterer Konzepte

Nachdem in Kap. 3 die Grundkonzepte der Serviceorientierten Architekturen vorgestellt wurden, wird in diesem Kapitel die Integration der Konzepte Business Process Management, komponentenbasierte Softwareentwicklung, modellbasierte Softwareentwicklung und Software-Produktlinien beschrieben. Der Fokus liegt dabei weniger auf einer Tiefe der Betrachtungen sondern in der Darstellung der wesentlichen Berührungspunkte.

4.1 Business Process Management

Business Process Management (BPM), d. h. das Management von Geschäftsprozessen, ist bereits seit einigen Jahren ein wichtiges Thema bei Unternehmen. Dabei ist sowohl die Analyse als auch die Modellierung und anschließende Optimierung von Geschäftsprozessen von Relevanz. Eine aktuelle Marktübersicht über 18 BPM-Werkzeuge wurde durch das Fraunhofer IAO durchgeführt [SW08].

Derzeit gewinnt das Thema BPM im Kontext von SOA zusätzlich an Bedeutung. Der Fokus liegt dabei auf der Umsetzung der Geschäftsprozesse durch IT-Systeme, insbesondere durch die direkte Ausführung der modellierten Prozesse durch die IT-Systeme. Eine möglichst gute und effiziente Unterstützung der Geschäftsprozesse durch die IT-Systeme ist eines der wesentlichen Versprechen einer SOA [Hö07b].

Daher soll in diesem Abschnitt der Zusammenhang zwischen BPM und SOA vereinfacht dargestellt werden (siehe auch Abbildung 2). Oftmals werden derzeit diese Begriffe - teilweise auch aus Marketing-Gründen - vermischt, was nach Sicht der Autoren unnötig ist, da klar definierte Beziehungen bestehen. Dies wird auch durch den SOA-Check 2007 bestätigt, einer Umfrage, die von der Amadee AG und vom Wolfgang Martin Team durchgeführt wurde und in Form einer Studie veröffentlicht wurde [Am07]. Dort assoziiert nur eine Minderheit von 24% SOA mit der kompletten Restrukturierung von Unternehmensprozessen.

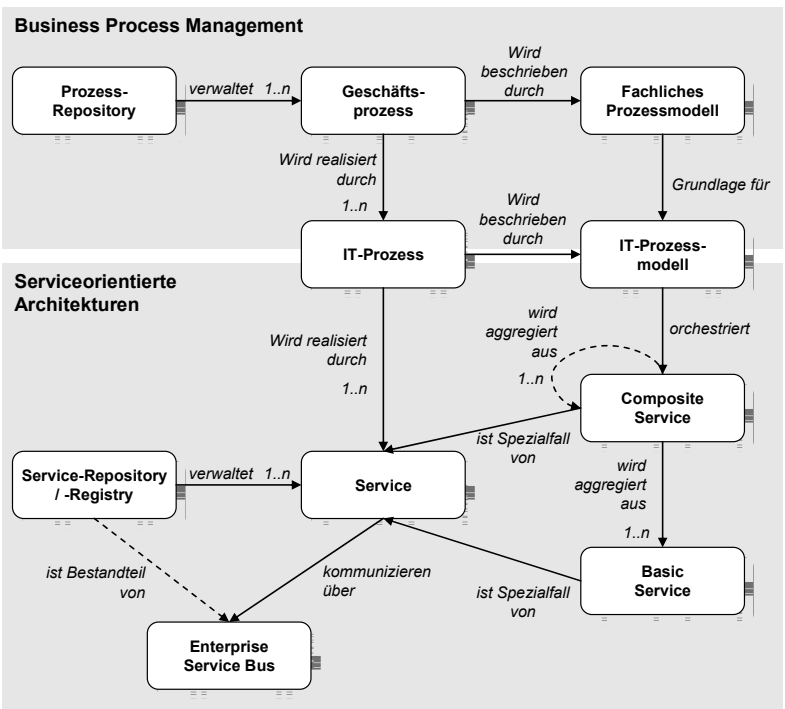


Abbildung 2: Integration von BPM-Konzepten

Die in Kap. 3 beschriebenen *IT-Prozesse* sowie die dazugehörigen *IT-Prozessmodelle*, die eine SOA, die die Orchestrierung im Rahmen einer SOA steuern, stehen in einem engen Zusammenhang mit den fachlichen *Geschäftsprozessen* sowie den dazugehörigen *fachlichen Prozessmodellen*, die diese beschreiben. Die *IT-Prozesse* setzen die *Geschäftsprozesse* in den IT-Systemen auf Basis einer SOA um. Die entsprechenden Prozessmodelle, d. h. das fachliche *Prozessmodell* sowie das *IT-Prozessmodell*, können voneinander abgeleitet werden.

Da zwischen den *fachlichen Prozessmodellen* und den *IT-Prozessmodellen* eine sog. „semantische Lücke" besteht, ist eine automatische Ableitung problematisch. Im Forschungsumfeld arbeiten eine Reihe von Vorhaben an diesem Thema, z. B. das vom Bundesministerium für Bildung und Forschung geförderte Projekt OrViA [SI07].

Wird in einem Unternehmen wirklich ein systematisches Business Process Management umgesetzt, was derzeit leider nur bei einer Minderheit der Unternehmen der Fall ist, ist es notwendig, ein Werkzeug zu verwenden, mit dem die *Geschäftsprozesse* verwaltet werden, also ein *Prozess-Repository*. Dieses kann mit den heute am Markt verfügbaren BPM-Werkzeugen realisiert werden [SW08].

Insgesamt beschäftigt sich BPM also mit der Optimierung von *Geschäftsprozessen* und SOA mit der Optimierung der entsprechenden *IT-Prozesse* bzw. IT-Systeme. Beide Konzepte ergänzen sich jedoch sehr gut und besitzen eine enge inhaltliche Verbindung (siehe Abbildung 2).

4.2 Komponentenbasierte Softwareentwicklung

Vor ca. 10 Jahren war das Schlagwort der „komponentenbasierten Softwareentwicklung" (component-based software development, CBSD) mit dem Aufkommen von Technologien wie CORBA (Common Object Request Broker Architecture), ActiveX oder EJBs (Enterprise Java Beans) in aller Munde. Das Versprechen war vor allem, durch die Wiederverwendung von bestehenden *Komponenten* eine Steigerung der Effizienz und Effektivität der Softwareentwicklung sowie eine gesteigerte Qualität der entstehenden Software-Produkte zu erreichen [We02]. Das Versprechen konnte auch in vielen Bereichen erfüllt werden. Insbesondere durch den extensiven Einsatz von OpenSource-Komponenten kann heutzutage Software teilweise sehr effizient und kostengünstig entwickelt werden [Hö05].

Oftmals ist jedoch der Zusammenhang zwischen dem Konzept der komponentenbasierten Softwareentwicklung und dem Konzept SOA unklar. Oftmals wird dargestellt, dass das Konzept der komponentenbasierten Softwareentwicklung durch das Konzept der serviceorientierten Architektur obsolet geworden ist.

Aus Sicht der Autoren ist diese Aussage jedoch nicht korrekt. Stattdessen kann ein sehr einfacher Zusammenhang hergestellt werden, da auch mit dem Aufkommen von SOA „traditionelle" *Komponenten* benötigt werden, mit denen die *Services* bzw. *Basic Services* auf unterster Ebene realisiert werden (siehe Abbildung 3).

Es macht daher nach wie vor Sinn, die vorhandenen Komponenten in einem *Komponenten-Repository* zu verwalten. Dies kann unternehmensintern eingesetzt werden oder es können öffentliche im Internet verfügbare Repositories, z. B. für Open Source Komponenten, genutzt werden.

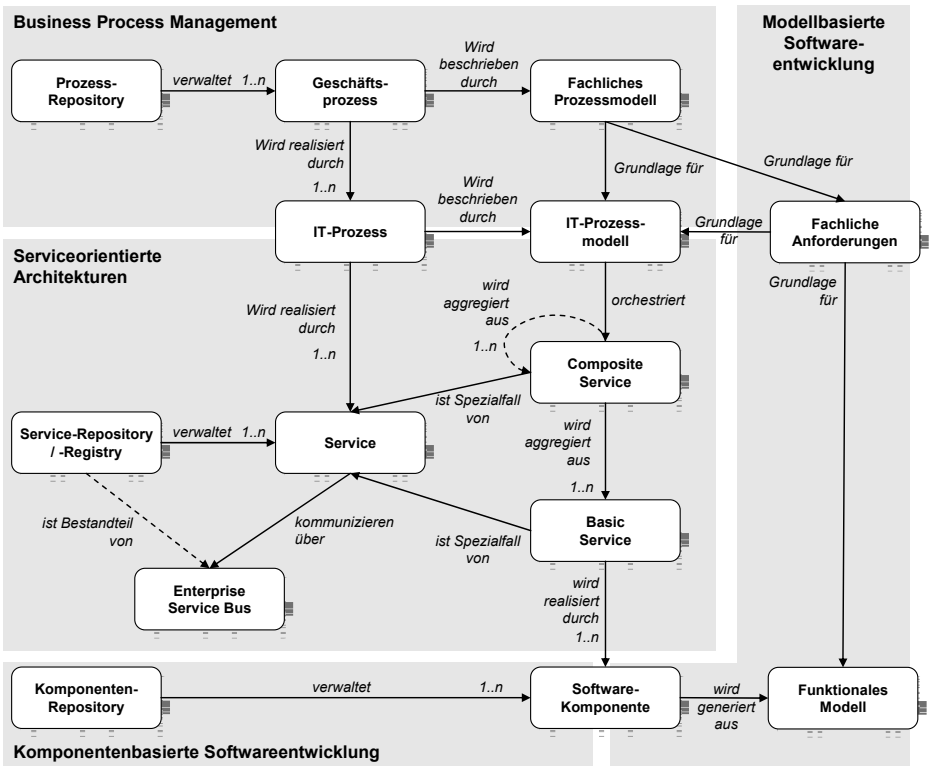


Abbildung 3: Komponentenbasierte und modellbasierte Softwareentwicklung

4.3 Modellbasierte Softwareentwicklung

Auch das Konzept der „modellbasierten Softwareentwicklung“ (model-based software development, MBSD) kann sehr gut in das beschriebene Metamodell integriert werden (siehe Abbildung 3). Die Grundidee dabei ist es, dass die Funktionalität von *Komponenten* nicht manuell in einer herkömmlichen Programmiersprache codiert wird, sondern stattdessen ein logisch abstrakteres *funktionales Modell* erstellt wird, aus dem die *Komponenten* generiert werden [SWH05].

Das *funktionale Modell* wird wiederum aus den *fachlichen Anforderungen* abgeleitet, die wiederum aus dem *fachlichen Prozessmodell* abgeleitet werden. Alternativ zu einer Generierung der *Komponenten* kann das *funktionale Modell* auch zur Konfiguration der *Komponenten* zur Laufzeit verwendet werden. Unter diesem Gesichtspunkt ist auch die Orchestrierung einer SOA durch *IT-Prozessmodelle* (z. B. in BPEL) als modellbasierte Softwareentwicklung zu verstehen (vgl. Kap. 3 und 4.1).

4.4 Software-Produktlinien

Die in 4.2 beschriebene „traditionelle“ Wiederverwendung auf Basis von Komponenten, die in einem Repository verwaltet werden, führt aufgrund einer Vielzahl von Problemfeldern, sowohl im organisatorischen als auch im technischen Umfeld, oftmals nicht zum Erfolg [Hö05].

Wenn man vor allem Systeme bzw. Software-Produkte in einer einzigen Domäne bzw. mit einem gemeinsamen Systemkern entwickelt, hat sich der Ansatz der Software-Produktlinien durchgesetzt, der bereits in einer Vielzahl von Anwendungsfällen erfolgreich eingesetzt wurde [LSR07].

Die über die verschiedenen Produkte identischen Funktionalitäten werden dabei in einer *Produktlinienplattform* zusammengefasst, die dann als gemeinsame Basis für die verschiedenen Produkte verwendet wird [PBL05]. Zu dieser gehören, wenn der Produktlinien-Ansatz eingesetzt wird, innerhalb des in diesem Artikel dargestellten Modells *Software-Komponenten*, *Basic* und *Composite Services* sowie *IT-Prozessmodelle*. Die Produktlinien-Plattform kann auch weitere Bestandteile beinhalten, was jedoch nicht der Schwerpunkt dieses Artikels ist.

Die Entwicklung der *Produktlinienplattform* geschieht nicht ad hoc, sondern auf Basis eines *fachlichen Domänenmodells*, das die gemeinsamen Funktionalitäten beschreibt und das auf Basis der Anteile des *fachlichen Prozessmodells* bzw. der Anteile der *fachlichen Anforderungen* entwickelt wird, die für alle Produkte identisch sind.

5 Fazit und Ausblick

In Abbildung 4 sind alle Konzepte, die in diesem Artikel betrachtet wurden, in einem Modell zusammengefasst, d. h. SOA, BPM, komponentenbasierte und modellbasierte Softwareentwicklung sowie Software-Produktlinien.

Selbstverständlich besteht noch ein großes Erweiterungspotenzial dieses Modells sowohl in der inhaltlichen Tiefe als auch in der Breite. In der Tiefe könnten sicherlich bei jedem der Einzelaspekte zusätzliche Entitäten und Beziehungen ergänzt werden. In der Breite könnten sicherlich noch weitere Konzepte aus interessanten aktuellen Themenfeldern, wie z. B. Software-as-a-Service oder Grid Computing, integriert werden.

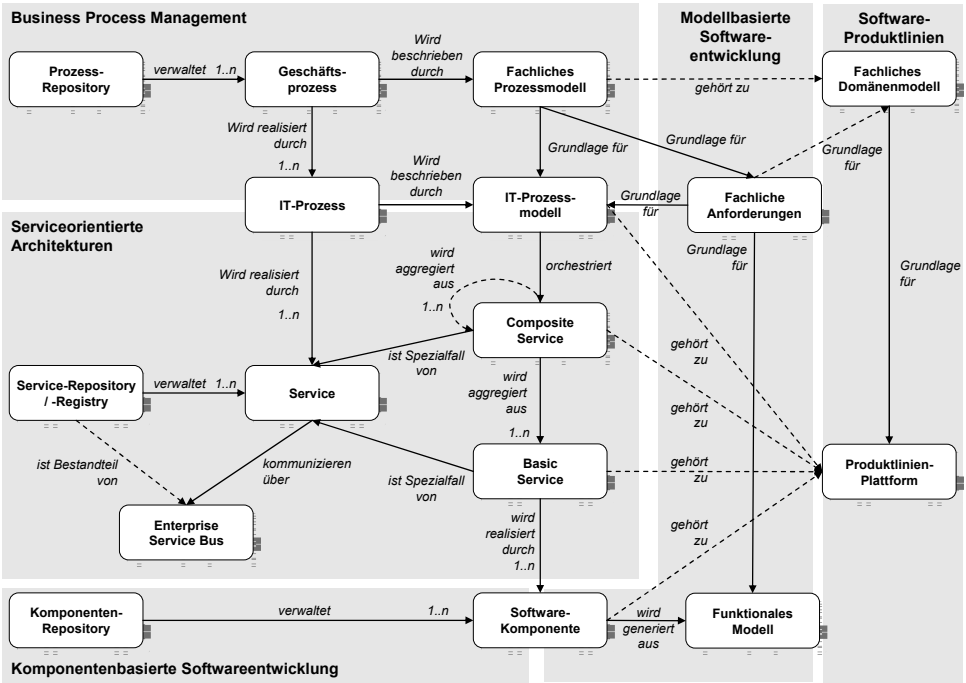


Abbildung 4: Integration von Software-Produktlinien

Zudem wurden in dem Beitrag organisatorische Aspekte der Industrialisierung der Softwareentwicklung weitestgehend ausgeklammert. Neben den hier beschriebenen primär technischen Aspekten dürfen diese in der Praxis jedoch nicht vernachlässigt werden.

Auch bei der Darstellung der einzelnen Konzepte bestehen noch Diskussionspotenziale. Der Beitrag sollte daher auch als Anregung für eine wissenschaftliche Diskussion dienen, die das Ziel besitzt, ein Begriffsnetz für aktuelle Themen im Bereich Software-Management und -Technik zu definieren.

Eine akzeptierte Ontologie in dieser Domäne würde sowohl der Wissenschaft als auch der Praxis helfen, oftmals bei Projekten auftretende langwierige und nicht zielführende Diskussion über einzelne Begriffe zu vermeiden.

Literaturverzeichnis

[Am07] Amadee AG; Wolfgang Martin Team: Studie SOA Check 2007 – Trends im deutschen Markt. http://www.soa-check.net, zuletzt besucht am 23.5.2008.

[Bi07] Biskup, H. et al.: Ein Begriffsnetz für die Software-Entwicklung. In: Informatik-Spektrum Band 30 Heft 3. Springer Verlag, Heidelberg, 2007; S. 217-224.

[En08] Engels, G. et al.: Quasar Enterprise - Anwendungslandschaften serviceorientiert gestalten. dpunkt Verlag, Heidelberg, 2008.

[Hö07a] Höß, O. et al.: Serviceorientierte Architekturen – Potenziale, Herausforderungen und Lösungsansätze. In (Spath, D. et al., Hrsg.): Serviceorientierte Architekturen. Fraunhofer IRB Verlag, Stuttgart, 2007; S. 1-11.

[Hö07b] Höß, O. et al.: Migration zu serviceorientierten Architekturen - top-down oder bottom-up? In (Hildebrand, K., Hrsg.): IT-Integration & Migration. HMD - Praxis der Wirtschaftsinformatik, Heft 257. dpunkt Verlag, Heidelberg, 2007; S. 39-46.

[Hö05] Höß, O.: Ein System für das Wiederverwendungs-Management von Software-Komponenten. Dissertation Universität Stuttgart. Jost-Jetter Verlag, Heimsheim, 2005.

[Jo08] Josuttis, N.: SOA in der Praxis - System-Design für verteilte Geschäftsprozesse. dpunkt Verlag, Heidelberg, 2008.

[LSR07] Linden, F. v. d.; Schmid, K.; Rommes, E.: Software Product Lines in Action - The Best Industrial Practice in Product Line Engineering. Springer Verlag, Heidelberg, 2007.

[Oa06] OASIS: Reference Model for Service Oriented Architecture 1.0. http://www.oasis-open.org/committees/download.php/19679/soa-rm-cs.pdf, zuletzt besucht am 23.5.2008.

[PBL05] Pohl, K.; Böckle, G.; Linden, F. v. d.: Software Product Line Engineering: Foundations, Principles, and Techniques. Springer Verlag, Heidelberg, 2005.

[Ro03] Rombach, D.: Software als Ingenieurprodukt. In (Warnecke, H.-J.; Bullinger, H.J., Hrsg.): Kunststück Innovation: Praxisbeispiele aus der Fraunhofer-Gesellschaft. Springer Verlag, Berlin, 2003; S. 75-82.

[SI07] Stein, S; Ivanov, K.: EPK nach BPEL Transformation als Voraussetzung für praktische Umsetzung einer SOA. In (Bleek, W.-G.; Raasch, J.; Züllighoven, H., Hrsg.): Software Engineering 2007. Lecture Notes in Informatics (LNI), Band 105. Gesellschaft für Informatik (GI), 2007; S. 75-80.

[Sp04] Spath, D.: Software als Innovationsmotor. In (Spath, D.; Haasis, K.; Klumpp, D., Hrsg.): Aktuelle Trends in der Softwareforschung. Tagungsband des doIT Software-Forschungstags 2004. Fraunhofer IRB Verlag, Stuttgart, 2004; S. 30-37.

[SS07] Simon, M.; Schneider, C.: Strukturierung der EnBW SOA anhand eines Metamodells. In (Spath, D. et al., Hrsg.): Serviceorientierte Architekturen. Fraunhofer IRB Verlag, Stuttgart, 2007; S. 83-99.

[SW08] Spath, D.; Weisbecker, A., Hrsg.: Business Process Management Tools 2008. Evaluierende Marktstudie. Fraunhofer IRB Verlag, Stuttgart, 2008.

[SWH05] Spath, D.; Weisbecker, A.; Höß, O.: Modellbasierte Softwareentwicklung. In (Spath, D.; Weisbecker, A.; Höß, O., Hrsg.): Modellbasierte Softwareentwicklung. Fraunhofer IRB Verlag, Stuttgart, 2005; S. 11-14.

[We02] Weisbecker, A.: Software-Management für komponentenbasierte Software-Entwicklung. Habilitation Universität Stuttgart. Jost-Jetter-Verlag, Heimsheim, 2002.

Teil III

Tutorien

Tutorium: Effektives Testmanagement - Vom notwendigen Übel zum Erfolgsfaktor für IT und Geschäftserfolg

Andreas Birk

Software.Prozess.Management
Gutenbergstraße 99
70197 Stuttgart
andreas.birk@swpm.de

Abstract: Das Testmanagement liefert einen wichtigen Beitrag zum Erfolg von Software- und IT-Projekten. Es muss Funktionalität und Qualität hochkomplexer Software-Systeme sichern und dabei oft enge Zeit- und Kostenvorgaben einhalten. Dieses Tutorial zeigt, wie das Testmanagement diese Aufgaben effizient erfüllen kann. Es gibt einen Überblick über Herausforderungen und aktuelle Trends im Testmanagement. Fallstudien aus der Praxis vermitteln Lösungen, mit denen Software-Entwicklung und IT zum Geschäftserfolg des Unternehmens beitragen.

1 Erfolgsfaktor Testmanagement

Das Testmanagement besitzt eine besondere Bedeutung für den Erfolg von IT-Projekten: Es sichert die Qualität der entwickelten Produkte und Services und leistet einen wichtigen Beitrag zur Einhaltung der Zeit- und Kostenvorgaben im Projekt.

Lange Zeit haben Auftraggeber und Software-Entwicklung das Testen als ein notwendiges Übel betrachtet: Man musste es tun, war aber nicht sonderlich daran interessiert. Da Software und IT heute maßgeblich für den Geschäftserfolg der Unternehmen sind, wird auch das Testen zum erfolgskritischen Faktor. Die hohe Komplexität und Vielfalt der Software-Lösungen schafft neue und anspruchsvolle Herausforderungen für das Testmanagement.

2 Effizientes Testmanagement in der Praxis

Anhand von Fallstudien aus der industriellen Praxis zeigt das Tutorial, dass IT-Organisationen die Herausforderungen des Testmanagements meistern können. Die Fallstudien veranschaulichen gängige Probleme und Lösungswege (*Best Practice*) und weisen auch auf die aktuellen Trends im Testmanagement hin.

Das Tutorial erläutert zunächst, durch welche Maßnahmen Testen und Testmanagement als strategische IT-Aufgabe im Unternehmen positioniert werden können:

- Effektivität und Effizienz des Testmanagements gewährleisten;
- Tool-Unterstützung und Automatisierung des Testens;
- Integration der Testaktivitäten in Entwicklung und Betrieb;
- Kontinuierliche Verbesserung von Testprozessen.

Voraussetzungen für Effektivität und Effizienz des Testmanagements sind klar definierte Aufgaben- und Zuständigkeitsbereiche sowie die Etablierung eines einheitlichen Testvorgehens im Unternehmen. Dabei kommt der Tool-Unterstützung für das Testmanagement eine wichtige Rolle zu, denn nur mit ihrer Hilfe behält man die komplexen Testaufgaben im Griff. Testautomatisierung kann zusätzliche Erleichterungen schaffen, sie muss allerdings mit Bedacht geplant und gut vorbereitet sein.

In vielen Organisationen sind Software-Entwicklung und Betrieb traditionell mit unterschiedlichen Testaufgaben betraut: Die Software-Entwicklung typischerweise mit den funktionalen Tests, der IT-Betrieb mit den Performance-Tests. Für ein effizientes Testmanagement müssen beide Aufgabenbereiche aufeinander abgestimmt und durch ein zentrales Testmanagement koordiniert werden. Die kontinuierliche Verbesserung der Testprozesse muss etabliert werden, damit die Prozesse mit neuen Anforderungen und Umfeldsituationen Schritt halten.

3 Aktuelle Trends im Testmanagement

Software-Entwicklung und IT unterliegen einem kontinuierlichen Wandel, dem sich auch das Testmanagement stellen muss. Aktuelle Trends für das Testmanagement sind:

- Testmanagement bei Auftraggeber- und Auftragnehmer-Beziehungen;
- Testen in der agilen Software-Entwicklung;
- Security-Testing von Web-Applikationen;
- Anforderungs- und risikobasiertes Testen.

Die komplexen Abhängigkeiten bei Lieferantenbeziehungen moderner Software-Projekte erfordern eine neue Testorganisation und Prozesse. Agile Software-Entwicklung stellt insbesondere den System- und Abnahmetest vor neue Herausforderungen.

Bei Web-Applikationen muss das Security-Testing schon frühzeitig im Entwicklungsprozess erfolgen und vom Testmanagement koordiniert werden. Anforderungs- und risikobasiertes Testen rücken die Belange der Auftraggeber und Anwender noch stärker in den Mittelpunkt und sichern so in besonderem Maße den Beitrag der IT zum Geschäftserfolg.

Tutorium: Erhebung von Produktanforderungen durch den Requirements Engineer

Stefan Jesse

Universität Stuttgart
Lehrstuhl für Betriebswirtschaftslehre und Wirtschaftsinformatik II
Breitscheidstr. 2c
70174 Stuttgart
jesse@wi.uni-stuttgart.de

Abstract: Das Requirements Engineering als erster Schritt der Systementwicklung entscheidet maßgeblich über den Erfolg oder Misserfolg eines Projektes. Die vier Haupttätigkeiten des Requirements Engineering sind: Erheben, Dokumentieren, Prüfen und Verwalten. Das Requirements Engineering ist eine Schlüsseldisziplin der Systementwicklung und viele weitere Disziplinen hängen von den Ergebnissen des Requirements Engineering ab. Den Stellenwert des Requirements Engineering zeigt der jährliche Bericht der Standish Group "The Scope of Software Development Project Failures". Rund die Hälfte der in dem Bericht untersuchten Projekte erreicht die angestrebten Ziele nicht aufgrund von Fehlern im Requirements Engineering. Ein professionelles Requirements Engineering trägt wesentlich dazu bei, Fehler in der Analyse zu beseitigen. Es beschreibt einen systematischen Weg von der Projektidee über die Ziele zu einem vollständigen Satz an Anforderungen.

1 Einführung

Die wesentlichen erfolgsbestimmenden Merkmale eines IT-Produktes werden im Rahmen der Produktdefinition festgelegt. Hier ist es Aufgabe des Requirements Engineers, die wesentlichen Anforderungen der Stakeholder zu bestimmen und Aufgabe des IT-Produktmanagements eine Lösung zu konzipieren, die in der Lage ist, diese Anforderungen unter Berücksichtigung der zu erzielenden Erlöse bzw. zur Verfügung stehenden Ressourcen zu erfüllen. Während die klassische Anforderungsanalyse in die Definition des Produktes und in den Übergang zur (Neu-)Entwicklung einzuordnen ist, ist heute das Verständnis von Anforderungs- und Lösungsmanagement umfassender: insbesondere auch Betrieb, Nutzung und Pflege müssen berücksichtigt werden. Betrieb und Nutzung können nicht zuletzt für die Weiterentwicklung und Pflege wichtige Impulse liefern. Zahlreiche aktuelle Techniken und Methoden in der Entwicklung von IT-Produkten zielen auf eine Verkürzung der Entwicklungsdauer, auf eine Steigerung der Wiederverwendung zur Kostensenkung und Qualitätsverbesserung und auf eine zumindest teilweise Automatisierung der Softwareentwicklung ab. Als Beispiele sind für die Softwarebestandteile Service-orientierte Architekturen, Modell-getriebene Entwicklung, Software-Produktlinien oder Aspekt-orientierte Programmierung und für die Hardwarebestandteile beispielsweise Simultaneous Engineering oder Plattformbauweise zu nennen.

Letztlich entscheidet offensichtlich nicht (nur) die bessere Prozess- bzw. Realisierungstechnologie, sondern (auch) der Stakeholder über den Erfolg eines Produktes. Hier gilt das Zitat von Meister: „Qualität ist das, was der Stakeholder (!) dafür hält“ [Me94]. Daraus kann man schließen, dass nur derjenige, der die Bedürfnisse der tatsächlichen und potentiellen Stakeholder kennt und versteht, in der Lage ist, diese planbar und wiederholbar in erfolgreiche Produkte umzusetzen.

2 Der Requirements Engineer

Das Requirements Engineering als erster Schritt der Systementwicklung entscheidet maßgeblich über den Erfolg oder Misserfolg eines Projektes. Die vier Haupttätigkeiten des Requirements Engineering sind das Erheben, Dokumentieren, Prüfen und Verwalten von Anforderungen. Das Requirements Engineering ist eine Schlüsseldisziplin der Systementwicklung und viele weitere Disziplinen hängen von den Ergebnissen des Requirements Engineering ab. Den Stellenwert des Requirements Engineering zeigt der jährliche Bericht der Standish Group “The Scope of Software Development Project Failures”. Rund die Hälfte der in dem Bericht untersuchten Projekte erreicht die angestrebten Ziele nicht aufgrund von Fehlern im Requirements Engineering.

Anforderungen dienen allen am Systementwicklungsprozess Beteiligten als Grundlage für Kommunikation, Diskussion und Argumentation. Ihre Aufgabe ist es, das gemeinsame Verständnis und Wissen der Teammitglieder widerzuspiegeln und zu explizieren. Ein professionelles Requirements Engineering und die damit verbundene organisatorische Installation und Positionierung des Requirements Engineers trägt wesentlich dazu bei, Fehler in der Analyse zu beseitigen. Professionelles Requirements Engineering beschreibt einen systematischen Weg von der Projektidee über die Ziele zu einem vollständigen Satz an Anforderungen.

Zu den Fähigkeiten des Requirements Engineers zählen die klassischen „weichen“ Faktoren, die sogenannten Soft Skills. Requirements Engineers müssen und können nicht alle Aufgaben im Rahmen der Ermittlung, Beschreibung, Abstimmung und Prüfung übernehmen, sondern müssen den strukturierten und zielgeführten Prozess der Anforderungsspezifikation einhalten. Dazu können sie jederzeit auf die Fachkompetenzen und Ressourcen der Anwendungsdomänen zugreifen.

Zu den wichtigsten Fähigkeiten des Requirements Engineers zählen analytisches Denken, selbstbewusstes Auftreten, empathische Fähigkeiten, Moderationsfähigkeit, Methodenkompetenz, Überzeugungsfähigkeit, Kommunikationsfähigkeit und sprachliche Kompetenz.

3 Basis-, Leistungs- und Begeisterungsanforderungen

Die Erhebung von Produktanforderungen stellt eine der Hauptaufgaben innerhalb des Aufgabenspektrums des Requirements Engineers dar. Dabei sollte er beachten, dass nicht alle Merkmale eines Produktes gleichermaßen auf Stakeholderzufriedenheit und somit auf den Erfolg wirken. Der Zusammenhang zwischen verschiedenen Leistungskomponenten eines Produktes auf die wahrgenommene Produktqualität und Zufriedenheit der Stakeholder beschreibt das sogenannte – empirisch überprüfte – Kano-Modell [Ka84]. Kano unterscheidet dazu drei Kategorien von Produktanforderungen: Basis-, Leistungs- und Begeisterungsanforderungen. Die Aussagen des Kano-Modells können graphisch in einem zweidimensionalen Koordinatensystem wiedergegeben werden (siehe Abbildung 1).

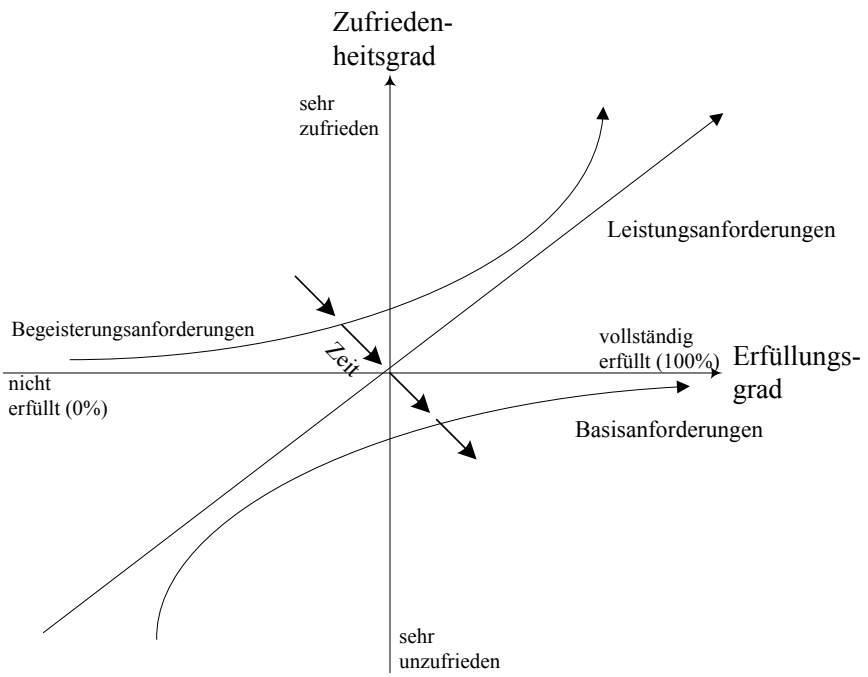


Abbildung 1: Arten von Produktanforderungen im Kano-Modell

Auf der horizontalen Achse wird der Erfüllungsgrad einer Stakeholderanforderung durch die Realisierung entsprechender Produktanforderungen von „nicht erfüllt“ bis zu „vollständig erfüllt“ abgetragen, auf der vertikalen Achse der dadurch bedingte Zufriedenheitsgrad mit der Erfüllung der Anforderung von „sehr unzufrieden“ bis „sehr zufrieden“. Die drei eingezeichneten Graphen repräsentieren die oben genannten Kategorien von Produktanforderungen. Unter diesen spiegeln nur die *Leistungsanforderungen* solche Anforderungen wider, die vom Stakeholder explizit verlangt werden. Sie werden deswegen auch mit „gewünschter“ oder „eindimensionaler Qualität“ gleichgesetzt, denn die Zufriedenheit steigt gleichmäßig mit zunehmendem Erfüllungsgrad.

Die *Basisanforderungen* entsprechen der „erwarteten Qualität“ des Produktes. Der Stakeholder verlangt oftmals nicht explizit nach diesen Produktmerkmalen, die Erfüllung der sich dahinter verbergenden Anforderungen wird als selbstverständlich vorausgesetzt. Deswegen fallen diese Charakteristika des fertigen Produktes nur bei unzureichender Umsetzung als „Quellen der Unzufriedenheit“ auf, werden allerdings auch bei nahezu perfekter Abdeckung auf der Zufriedenheitsskala nicht besonders gewürdigt.

B*egeisterungsanforderungen* beziehen sich auf Produktmerkmale, die vom Stakeholder nicht erwartet werden und in erster Linie der Kreativität und Innovationskraft der Entwickler entstammen. Oftmals geben sie unbewusste und versteckte Wünsche der Stakeholder wieder, die bei unzureichender Umsetzung nicht zu Unzufriedenheit führen, durch ihre Existenz allerdings zu „Quellen hoher Zufriedenheit“ werden können. Daher sind sie auch nicht mit „nice to have“-Funktionen zu verwechseln, da diese zwar vom Stakeholder auch nicht zwangsweise erwartet werden, aber eben auch nicht zu besonderer Begeisterung beim Stakeholder führen, wenn sie enthalten sind.

Ebenso muss betont werden, dass bei weitem nicht alle Produktmerkmale, die der Kreativität der Entwickler entspringen, wirklich vom Stakeholder gewünscht werden. Folglich ist die methodisch unterstützte Rückversicherung, welche Produktmerkmale Begeisterungsanforderungen erfüllen, von großer Bedeutung.

Zu berücksichtigen ist ferner noch der Zeitfaktor. Produktanforderungen, die zum heutigen Zeitpunkt Begeisterungsanforderungen darstellen, werden im nächsten Release vielleicht schon explizit von den Stakeholdern gefordert und in noch fernerer Zukunft als selbstverständlich vorausgesetzt. In den Kategorien von Produktanforderungen spiegelt sich also, in Kenntnis der oft hohen Eigendynamik des Softwaremarktes, der Zwang zur ständigen Verbesserung wider.

4 Erfolgsfaktor Stakeholderorientierung

Der Grundstein für erfolgreiche IT-Produkte wird bereits in der Produktentwicklungsphase gelegt. Entspricht schon die Konzeption nicht den Erwartungen der Stakeholder, kann auch die Gestaltung der nachfolgenden Produktions- und Absatzprozesse an der voraussichtlich entstehenden Stakeholderunzufriedenheit kaum etwas ändern. Die Produktentwicklung in Deutschland weist diesbezüglich jedoch noch zahlreiche Schwächen auf, die sich z. B. in der mangelnden Kommunikation der Produktentwicklung mit den Stakeholder- bzw. absatznahen Bereichen oder in dem Hang zur technischen Perfektion statt der Befriedigung von Stakeholderbedürfnissen manifestieren.

Dass Stakeholderorientierung ein sehr wichtiges Thema für Softwareunternehmen ist, bestätigt eine aktuelle Umfrage unter Verantwortlichen für das Thema Produktmanagement in deutschen Softwareunternehmen. Befragt, wie sehr wettbewerbsentscheidend verschiedene Faktoren sind, wurde die Passgenauigkeit ihrer Lösung/ihres Software-Produktes hinsichtlich der Anforderungen der Stakeholder von den Teilnehmern am stärksten gewichtet. Knapp dahinter und quasi gleichbedeutend landeten die technische Qualität der Software, die richtige Identifikation unklarer Stakeholderanforderungen, die Flexibilität der Software sowie der Gesamtumfang der Funktionalität der Software. Service und Support landeten auf dem vorletzten, der Preis auf dem letzten Platz [HHK07].

Hier ist der Produktmanager gefordert, die Stimme der Stakeholder ins Unternehmen zu tragen und der Requirements Engineer muss sicherstellen, dass die Stakeholderanforderungen richtig erkannt und umgesetzt werden. Der Requirements Engineer muss das Produktmanagement durch die Auswahl geeigneter Befragungs- und Beobachtungstechniken unterstützen, um implizite wie explizite Bedürfnisse der Stakeholder aufzunehmen.

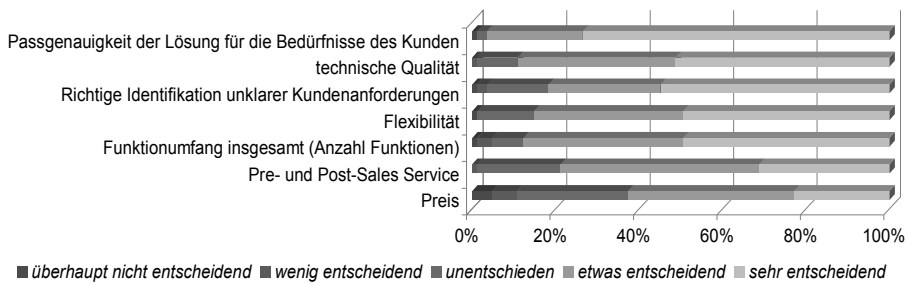


Abbildung 2: Bedeutung verschiedener Faktoren im Wettbewerb [HHK07]

5 Erhebung von Anwenderanforderungen

Theoretisch ist die Erhebung und Verwaltung von Anwenderanforderungen einfach: Man muss die Anwender des Produktes nur fragen, was sie haben wollen, das entsprechend im Produkt umsetzen und anschließend die Wünsche zur Verbesserung, die in aller Regel automatisch kommen, in der Weiterentwicklung und Verbesserung des Produktes berücksichtigen.

Wäre dies in der Praxis genauso einfach, wäre das Leben der meisten Requirements Engineers oder auch IT-Produktmanager viel einfacher. Stattdessen kämpfen diese damit, dass das Management von Stakeholderanforderungen deutlich schwieriger ist. Die Passgenauigkeit der eigenen Lösung für die Bedürfnisse der Stakeholder, aber auch die richtige Identifikation unklarer Stakeholderanforderungen werden von den Teilnehmern der Studie als deutlich wichtiger empfunden als z. B. der Preis. Da die Funktion des Produktmanagers ja gerade sein soll, die Stimme des Stakeholder in das Unternehmen zu tragen und dafür zu sorgen, dass die angebotenen Produkte den Stakeholderwünschen entsprechen, ist das Management von Stakeholderanforderungen eine der wichtigsten Herausforderungen, denen Requirements Engineers sich stellen müssen.

Die Erhebung von Anforderungen ist dabei diejenige Teilaufgabe, die für die Ermittlung der „qualitativen und quantitativen Eigenschaften eines Produkts aus der Sicht des Auftraggebers“ [Ba01] zuständig ist. Das Management von Anforderungen umfasst zusätzlich noch Maßnahmen zur Steuerung, Kontrolle und Verwaltung der Anforderungen.

Fast alle Aufgaben von Anforderungsanalyse und Anforderungsmanagement sind problembehaftet:

1. Definition der Anforderungen: Hier sind Ist-Analysen (also sowohl Beschreibung als auch Bewertung des Ist-Zustands) durchzuführen, die Anwendungsziele zu bestimmen und schließlich System- oder Produktanforderungen festzulegen,

2. Beschreibung der Anforderungen: Die Anforderungen müssen systematisiert und priorisiert werden sowie die Abhängigkeiten zwischen den Anforderungen aufgezeigt werden,

3. Analyse der Anforderungen: Die Anforderungen müssen konsistent, vollständig und richtig beschrieben werden und realisierbar sein,

4. Dokumentation der Anforderungen: Schließlich müssen die Anforderungen noch in einer Anforderungsspezifikation dokumentiert werden, damit man später auf sie zugreifen kann.

5. Verfolgung und Pflege der Anforderungen: Die Anforderungen müssen auch projekt- bzw. produktübergreifend verwaltet und abgestimmt werden.

Im Folgenden werden einige wichtige Probleme, die bei der Erfüllung der o. g. Aufgaben häufig auftreten, noch etwas genauer erläutert; beispielsweise ist es alleine schon schwierig, „den Stakeholder“ zu identifizieren: in aller Regel werden IT-Produkte von verschiedenen Stakeholdergruppen, welche das Produkt zu ganz unterschiedlichen Zwecken einsetzen, verwendet. Besonders groß ist dieses Problem naturgemäß beim Standardprodukt, aber auch bei der Entwicklung einer Individualsoftwarelösung existieren mehrere Stakeholdergruppen, so zum Beispiel häufige Nutzer, unregelmäßige Nutzer, Buying Center (welche die Software nicht nutzen, aber die Anbieterauswahl betreuen), das Management (welches die Software auch nicht nutzt, aber das Budget freigibt) und die IT-Abteilung (welche die Software betreibt und administriert). Sind all diese Stakeholdergruppen identifiziert und beginnt man, ihre Anforderungen an das Produkt zu erheben, so wird man in aller Regel feststellen, dass sie ganz unterschiedliche Anforderungen stellen. Beispielsweise wird der häufige Nutzer tendenziell viele Funktionen fordern, evtl. auch die Möglichkeit, die gesamte Anwendung rein mit Tastenkombinationen zu bedienen[1], der unregelmäßige Nutzer wird vor allem eine übersichtliche Benutzeroberfläche fordern (was einer großen Zahl von Funktionen eher widerspricht), Buying Center und Management werden stark auf den Preis schauen, die IT-Abteilung schließlich achtet auf geringe Systemanforderungen, Schnittstellen und einfache Administration.

Wie man sieht, gilt es, eine Vielzahl an – sich teilweise widersprechenden – Anforderungen unter einen Hut zu bringen. Dabei stellt sich insbesondere auch die Frage, wie wichtig verschiedene Anforderungen sind oder wie man bei sich widersprechenden Anforderungen vorzugehen hat. Diese Priorisierung kann nur durch die Stakeholder geschehen und sollte auch verwendet werden, um bei knappem Budget oder Terminplan entscheiden zu können, welche Funktionen nicht mehr umgesetzt werden können.

Ein weiteres Problem besteht darin, dass Anforderungen oftmals nicht eindeutig formuliert sind, beispielsweise die Anforderung: „Diese Funktion bewirkt, dass alle europäischen Umsätze des letzen Quartals zusammengezählt und dem Benutzer zurückgemeldet werden.“ Während diese Anforderung im ersten Moment relativ klar klingt, wird spätestens dem Entwickler beim Versuch, dies zu implementieren auffallen, dass einige Fragen offen sind: Umsätze aus welchen Ländern sollen berücksichtigt werden? Zählt das Unternehmen nur EU-Mitglieder zu Europa, was ist beispielsweise mit der Türkei (liegt bekanntlich mehrheitlich auf asiatischem Boden) und vor allem auch: in welcher Währung soll der Umsatz berechnet werden? Welcher Wechselkurs ist anzusetzen? In der Praxis kommt es leider oftmals vor, dass der Entwickler hier nicht nachfragt, sondern selbst eine Entscheidung trifft, die aber leider nicht dem entspricht, was der Stakeholder haben wollte. Daher wäre nachfragen besser, nur stellt sich bei einem normalen Pflichtenheft, das schnell fünfzig und mehr oder gar hunderte von Seiten umfasst, die Frage, bei wem man denn nachfragen soll. Folglich ist die Nachverfolgbarkeit von Anforderungen (warum? und wer wollte das so?) von großer Bedeutung.

[1] Während gelegentliche Nutzer eine grafische Oberfläche mit Mausbedienung in aller Regel vorziehen, sind häufige Nutzer oftmals schneller, wenn sie die Software komplett mit der Tastatur bedienen können, also nicht ständig zwischen Tastatur und Maus wechseln müssen.

Neben den primären Aufgaben von Anforderungen (Kommunikationsgrundlage, Vertragsgrundlage, Grundlage für Design und Implementierung, Grundlage für die Systemarchitektur sowie Grundlage für Installation und Wartung) existieren aber auch sekundäre Aufgaben von Anforderungen, die oftmals vergessen werden: insbesondere der angestrebte Geschäftsnutzen der Nutzung des IT-Produktes sollte (quasi als Vision) in der Anforderungsspezifikation mit aufgeführt sein. Beispielsweise sollen durch die Einführung eines neuen IT-Produktes, sei es eine CRM-Software oder mobile Endgeräte in der Lagerverwaltung, im Unternehmen oftmals Rationalisierungspotenziale gehoben werden. Eventuell liegt der hauptsächliche Stakeholdernutzen aber auch in der Integration bislang auf verschiedene Systeme verteilter Daten oder in der Erhöhung der Mitarbeiterzufriedenheit (wenn die Mitarbeiter von stupiden und repetitiven Tätigkeiten entlastet werden).

6 Einsatz von Befragungs- und Beobachtungstechniken zur Ergebung von Anwenderanforderungen

Da das Ermitteln von Anforderungen zu den wichtigsten Aufgaben des Requirements Engineers gehört, werden im Folgenden einige Instrumente zur Unterstützung des Requirements Engineers vorgestellt. Wie bereits im Rahmen des KANO-Modells erläutert, äußern Stakeholder explizit nur Leistungsanforderungen, d. h. Anforderungen die bei Erfüllung Stakeholderzufriedenheit hervorrufen und bei Nichterfüllung zu Stakeholderunzufriedenheit führen. Diese Leistungsanforderungen können gut von den Stakeholdern erfragt werden, da diese Art von Anforderungen dem Stakeholder bewusst sind.

Basisfaktoren sind dem Stakeholder ebenfalls bewusst, aber im Unterschied zu den Leistungsanforderungen sind diese dem Stakeholder oftmals zu selbstverständlich oder zu „trivial“, um explizit verlangt zu werden. Da das Vorhandensein von Basisfaktoren im Produkt jedoch zwingend ist, da ihr Vorhandensein bestenfalls zu einem Zustand der Nicht-Unzufriedenheit führt und das Vergessen von Basisanforderungen zu massiver Unzufriedenheit oder Produktablehnung führt, müssen diese Faktoren ebenfalls vom Requirements Engineer erhoben werden. Zur Erhebung, Definition und genauen Bestimmung von Basis- und Leistungsfaktoren stehen dem Requirements Engineer Beobachtungs- und Befragungstechniken zur Auswahl. Zu den Beobachtungstechniken zählen beispielsweise die Feldbeobachtung sowie das Apprenticing, zu den Befragungstechniken zählen u. a. Fragebogen und Interviews [Ru07].

Beobachtungstechnik Feldbeobachtung

Bei der Feldbeobachtung werden die Stakeholder an ihren Arbeitsplätzen beobachtet. Während des Beobachtens erkennt der außen stehende Betrachter nicht optimale Arbeitsschritte und Fehler, welche dem Stakeholder aus Gewohnheit nicht mehr bewusst sind. Beobachten lassen sich dabei natürlich nur Arbeitsabläufe, die bereits gelebt werden und vom Stakeholder auch tatsächlich durchgeführt werden. Von daher kann diese Technik nicht für völlig neue Prozesse angewandt werden. Vorteil der Methode ist, dass der Analytiker einen unverfälschten Eindruck der Sprache und Terminologie des Fachgebietes gewinnt. Darüber hinaus kann die Methode leicht durch Audio- oder Video-Aufzeichnungen unterstützt werden.

Beobachtungstechnik Apprenticing

Beim Apprenticing führt der Analytiker die Tätigkeiten der Stakeholder konkret aus. Wie ein Lehrling ist der Analytiker gefordert, unklare und unverständliche Handlungsschritte sofort zu hinterfragen, um in diesem Bereich Erfahrung zu sammeln. Der Analytiker kann hierdurch Anforderungen erfahren, welche für den Stakeholder so selbstverständlich wurden, dass er sie nicht mehr äußert. Einschränkend ist, dass die Methode in einem sicherheitskritischen Umfeld nicht uneingeschränkt einsetzbar ist. Bei der Apprenticing-Methode überbrückt der Analytiker durch eigene Erfahrung mangelnde kommunikative Fähigkeiten des Stakeholders.

Befragungstechnik Interview

Beim Interview kann vom Stakeholder eine genaue Aussage über die Anforderungen an das zu untersuchende System ermittelt werden. Beim Interview stellt der Analytiker einem oder mehreren Stakeholdern vorgegebene Fragen und protokolliert deren Antworten. Interviews setzen voraus, dass der Interviewte sein Wissen explizit ausdrücken kann. Gerade zu Beginn der Anforderungsermittlung eignen sich vor allem persönliche Interviews, in denen gemeinsam grobe Anforderungen an das System erarbeitet werden. Nachteil ist, dass (Einzel-)Interviews sehr zeitaufwändig sind und eine sehr persönliche Färbung des Systems durch die jeweiligen Befragten beinhalten.

Befragungstechnik Fragebogen

Bei dieser Befragsungstechnik werden Fragen zur Beantwortung durch den Stakeholder gesammelt. Fragebögen sind insbesondere bei einer Produktentwicklung gut geeignet, geplante Eigenschaften des Produkts von einer großen Zahl von Testpersonen bewerten zu lassen und in offenen Fragen Verbesserungsmöglichkeiten zu ermitteln. Insbesondere für größere Stakeholder-Anzahlen ist diese Befragungstechnik gut geeignet, setzt jedoch voraus, dass der Befragte sein Wissen explizit ausdrücken kann. Der Befragte muss weiterhin bereit sein, Zeit und Engagement in die Ermittlung zu investieren. Problemtisch ist, dass der Analytiker die zu beantwortenden Fragen vorgibt. Anliegen der Stakeholder können unter Umständen verdrängt, vergessen oder vernachlässigt werden.

7 Schwerpunkte des Tutoriums

- Vorstellung des Zertifizierungsprogramms „Certified Professional Requirements Engineerings“
- Grundlagen des Requirements Engineering
  - Einflussfaktoren des Requirements Engineering; Profil des Requirements Engineer
- Scope, Kontext und Schnittstellen
- Anforderungsarten
  - Überblick über Anforderungsarten; Anforderungen an das Verhalten
  - Daten/Informationsanforderungen; Nicht funktionale Anforderungen
  - Schnittstellen-Anforderungen
- Anforderungsdokumente
  - Dokumentengestaltung; Glossar
  - Repräsentationsformen für Anforderungen
- Anforderungen gestalten
  - Quellen für Anforderungen; Ermittlung von Anforderungen; Prüfen von Anforderungen
  - Stakeholder
- Werkzeuge für das Requirements Engineering
- Informationen über neue Entwicklungen im Requirements Engineering
- Checklisten und Templates für Requirements Engineering

Literaturverzeichnis

[Me94] Meister, H.: Qualität ist das, was Kunden (!) dafür halten, in: Marketing Journal, 1994

[Ka84] Kano, N., Seraku, N., Takahashi, F. und Tsuji, S.: Attractive quality and must-be quality, in: Quality, Vol.14, 1984, No.2, S.39-44.

[HHK07] Helferich, A., Hiller, K.; Kneher, S.: Variantenmanagement in der Softwareentwicklung - Ergebnisse einer Umfrage unter deutschen Softwareanbietern, Stuttgarter Schriften zur Unternehmenssoftware Nr. 1, Stuttgart 2007

[Ba01] Balzert, H.: Lehrbuch der Software-Technik – Bd. 1: Software-Entwicklung, 2. Aufl., Heidelberg u. a. 2001

[Ru07] Rupp, C.: Requirements-Engineering und -Management: Professionelle, iterative Anforderungsanalyse für die Praxis, 4. Aufl., München 2007

Tutorium: Mit Fehlerkennzahlen Softwareentwicklungsprojekte erfolgreich steuern

Oral Avci

SQS Software Quality Systems AG
Stollwerckstr. 11
51149 Köln
oral.avci@sqs.de

Abstract: Fehler in der Softwareentwicklung werden nach ihrer Entdeckung meist nur korrigiert, ohne aus ihnen zu lernen. Es ist folglich nicht unwahrscheinlich, dass vergleichbare Fehler wieder auftreten, so lange die Ursachen im Prozess fortbestehen. Wie aber kann man in Softwareentwicklungsprojekten aus Fehlern lernen und Verbesserungspotenziale im Prozess aufdecken? Das gleichnamige Tutorium auf der Fachtagung Software-Management 2008 stellt ein Verfahren vor, um aus Fehlern Prozessverbesserungen ableiten zu können. Dieser Beitrag gibt einen Überblick über die Inhalte dieses Tutoriums.

1 Einführung

Fehler sind in Softwareentwicklungsprojekten allgegenwärtig. Unter einem Fehler wird ein unzureichendes Merkmal oder ein erwartetes, jedoch fehlendes Merkmal eines Arbeitsergebnisses der Softwareentwicklung verstanden, sofern es eine Änderung in diesem Ergebnis notwendig macht. Mängel im Prozess der Softwareentwicklung verursachen häufig viele Fehler in Arbeitsergebnissen (Fachkonzept, Softwarecode etc.). Je später ein Fehler entdeckt wird, desto mehr Folgefehler entstehen und folglich steigt der Aufwand für die Fehlerkorrektur.

Wie aber kann man in Softwareentwicklungsprojekten aus Fehlern lernen und Verbesserungspotenziale im Prozess aufdecken? Im Tutorium „Mit Fehlerkennzahlen Softwareentwicklungsprojekte erfolgreich steuern“ auf der Fachtagung Software-Management 2008 wird ein Verfahren vorgestellt, um aus Fehlern Prozessverbesserungen ableiten zu können. Es wird beschrieben, wie Fehler sinnvoll klassifiziert und ausgewertet werden können, um Prozessmängel als Ursachen der Fehler aufdecken zu können. Neben dem Verfahren werden praktischen Erfahrungen aus einer Fallstudie aufgezeigt.

Der vorliegende Beitrag gibt einen Überblick über die Inhalte dieses Tutoriums. Er ist wie folgt aufgebaut: In Kapitel 2 wird die Motivation für das Fehleranalyseverfahren beschrieben. Kapitel 3 beschreibt die wesentlichen Merkmale des Fehleranalyseverfahrens. Der Nutzen des Verfahrens wird in Kapitel 4 dargestellt. Abschließend wird in Kapitel 5 eine Fallstudie vorgestellt, bei der praktische Erfahrungen bei der Anwendung des Verfahrens gewonnen werden konnte. Eine detaillierte Beschreibung des Verfahrens und der Fallstudie findet sich in [Av08].

2 Das Problem: Fehler in Softwareentwicklungsprojekten

In Softwareentwicklungsprojekten entstehen viele Fehler in Arbeitsergebnissen wie beispielsweise dem Fachkonzept oder dem Softwarecode. Diese Erkenntnis ist zunächst nicht überraschend. Interessant ist jedoch, dass verschiedene empirische Untersuchungen seit Mitte der siebziger Jahre bis heute feststellen, dass ein Großteil der Fehler, nämlich 15% bis 50%, durch die Anforderungsanalyse verursacht werden [AW05]. Hierbei handelt es sich zum einen um Anforderungsfehler und zum anderen um Folgefehler im Entwurf und der Implementierung. Die Untersuchungen zeigen zudem auf, dass es sich bei den Anforderungsfehlern häufig um übersehene oder falsch verstandene Anforderungen handelt, also Fehler, die meist erst sehr spät entdeckt werden. Je später aber ein Fehler entdeckt wird, desto mehr Folgefehler können entstehen und desto aufwändiger wird die Korrektur. Die Kosten für Überarbeitungen können bis zu 41% der gesamten Projektkosten ausmachen [Di93]. Die Auswirkungen dieser erforderlichen Überarbeitungen sind vertraut: Software wird zu spät ausgeliefert und Budgets werden überschritten. Werden hingegen die Überarbeitungen nicht durchgeführt, erhalten die Kunden eine Software minderer Qualität.

Fehler finden Eingang in Arbeitsergebnisse, weil sie durch den Prozess, also der Art und Weise wie eine Aufgabe durchgeführt wird, begünstigt werden. Obwohl Fehler relevante Informationen über den Prozess geben können, werden diese nicht genutzt. Selbst dann nicht, wenn eine Fehlerverfolgung gestützt durch ein Softwarewerkzeug praktiziert wird (meist bezeichnet als „bug tracking tool“ oder „defect tracking tool“). Entsprechende Werkzeuge unterscheiden in der Regel nicht zwischen Fehlern und ihren Symptomen, z. B. dem Fehlverhalten der Software. Außerdem legen sie nahe, ausschließlich rein administrative Informationen zu erfassen: Wann ist das Fehlverhalten aufgetreten? Wurde der Fehler bereits behoben? Wer ist für die Behebung verantwortlich? etc. Beschränkt man sich jedoch auf diese administrativen Informationen, versäumt man es, aus Fehlern zu lernen. Fehler werden dann nur behoben und geraten anschließend schnell wieder in Vergessenheit. Eine nachhaltige Prozessverbesserung bleibt aus.

3 Die Lösung: Prozessrelevante Informationen zu Fehlern erfassen und auswerten

Aus Fehlern lernen heißt, die Ursachen dieser Fehler im Prozess zu identifizieren. Ist ein Mangel im Prozess bekannt und verstanden, kann er ggf. behoben werden. In diesem Fall werden zukünftig vergleichbare Fehler vermieden. Sofern ein Mangel im Prozess nicht behoben werden kann, ist die Kenntnis des Prozessmangels trotzdem wertvoll, um den Prozess der Qualitätssicherung anzupassen. So kann nach Fehlern gezielt gesucht werden, die durch den Prozessmangel verursacht worden sind. Entsprechende Fehler werden folglich früher gefunden und weitere Folgefehler vermieden.

Wie kann man nun aus Fehlern lernen? Hierzu wurde durch den Autor ursprünglich am Seminar für Wirtschaftsinformatik und Systementwicklung an der Universität zu Köln ausgehend von der Orthogonal Defect Classification [Ch96] ein Verfahren entwickelt, das auf Fehlerkennzahlen aufbaut und im Wesentlichen aus drei Schritten besteht.

Schritt 1: Fehlererfassung

Im Kern des Verfahrens steht ein etabliertes Klassifikationsschema für Fehler im Entwurf und der Implementierung, das Anfang der neunziger Jahre durch Ram Chillarege entwickelt und seither verfeinert worden ist. Anhand dieses Klassifikationsschemas werden die Fehler erfasst. Im Unterschied zu anderen Klassifikationsschemata, die in der Literatur oder in Fehlerverfolgungstools vorgeschlagen werden, extrahiert es prozess- und produktrelevante Informationen aus einem Fehler. So werden beispielsweise für jeden Fehler folgende Fragen beantwortet:

- Welches strukturelle Merkmal wurde geändert, um den Fehler zu beheben?
- Wie wurde der Fehler behoben? Indem etwas eingefügt, etwas gelöscht oder etwas geändert worden ist?
- Welches Prüf- oder Testkriterium hat zur Entdeckung des Fehlers geführt?

Schritt 2: Identifikation von Fehlermustern mit Handlungsbedarf

Unter wirtschaftlichen Gesichtspunkten ist es nicht sinnvoll, eine Ursachenanalyse für jeden Fehler einzeln vorzunehmen. Die semantischen Informationen zu den erfassten Fehlern aus dem ersten Schritt erlauben durch einfache ein- und mehrdimensionale Auswertungen Fehlermuster zu identifizieren. Ein Fehlermuster ist eine Menge von Fehlern mit gemeinsamen Merkmalen. Den Fehlern eines Fehlermusters liegen mit großer Wahrscheinlichkeit die gleichen Ursachen zugrunde. Im Rahmen einer Arbeitssitzung werden besonders auffällige Fehlermuster mit ausgewählten Projektbeteiligten analysiert. Diese Arbeitssitzung muss durch eine Person, die das Verfahren beherrscht, moderiert werden. Ziel dieser Arbeitssitzung ist es, dass die Projektbeteiligten festlegen, für welche Fehlermuster Maßnahmen eingeleitet werden sollen.

Schritt 3: Aufdecken der Ursachen der identifizierten Fehlermuster

In einer weiteren moderierten Arbeitssitzung bestimmen Projektmitarbeiter die Ursachen von ein bis zu maximal drei ausgewählten Fehlermustern. Der Moderator konkretisiert hierzu zunächst jeweils die Fehlermuster und stellt exemplarisch einige Fehlerberichte zur Diskussion.

Genauso wie ein Arzt auf der Grundlage festgestellter Krankheitssymptome, die Ursachen für die Krankheit eines Patienten erkennen und schließlich behandeln will, können die Fehlermuster als Symptome aufgefasst werden, die Rückschlüsse über Mängel im Prozess erlauben [Ca98]. Daran schließen Überlegungen an, wie ein Fehlermuster zukünftig vermieden oder entsprechende Fehler früher gefunden werden können. Da die Prozessmängel mit dem Verfahren präzise konkretisiert werden, erleichtert dies eine effektive und effiziente Entwicklung von wirksamen Verbesserungsmaßnahmen mit Projektbeteiligten. Das Ergebnis sind Verbesserungsmaßnahmen, die unternehmensspezifische und projektspezifische Besonderheiten berücksichtigen und in der Summe den Prozess nachhaltig verbessern.

4 Nutzen für Praktiker

Eine Prozessanalyse auf der Grundlage von Fehlerkennzahlen ermöglicht eine leichte, systematische, kontinuierliche Verbesserung der Prozesse. Das Verfahren fördert auf der Grundlage quantitativer Daten eine sachliche Diskussion mit Beteiligten über die Prozesse. Da zudem Fehler bereits während einer Softwareentwicklung erfasst und ausgewertet werden können, eröffnet sich die Möglichkeit, das beschriebene Verfahren begleitend zur Entwicklung einzusetzen. So können bereits während der Laufzeit eines Projekts Mängel im Prozess aufgedeckt werden. Einem Entwicklungs- bzw. Projektleiter liefert das Verfahren Informationen über den Stand des Entwicklungsprozesses und des Softwareproduktes. Einem Testmanager zeigt es auf, wie erfolgreich die Prüf- und Testaktivitäten sind. Ein Prozessverantwortlicher kann durch den Vergleich von historischen mit aktuellen Fehlerdaten untersuchen, ob Änderungen im Prozess zu den gewünschten Verbesserungen geführt haben.

5 Die Fallstudie: Anwendung des Verfahrens in einem großen Softwareentwicklungsprojekt

Das Verfahren wurde im Rahmen einer Fallstudie in einem realen Softwareentwicklungsvorhaben bei dem IT-Dienstleister einer großen deutschen Versicherung angewandt. Es wurden nachträglich reale Fehler von zwei Releaseentwicklungen zu einer geschäftskritischen Anwendungssoftware für Lebensversicherungen erfasst und analysiert, um Mängel im Prozess der Anforderungsanalyse dieses Entwicklungsvorhabens aufzudecken. Im Tutorium werden praktische Erfahrungen und Beispiele aus dieser erfolgreichen Anwendung des Verfahrens vorgestellt.

6 Kurzüberblick über die Inhalte des Tutoriums

Im Tutorial wird ein Kennzahlensystem zur Steuerung laufender Softwareentwicklungsprojekte vorgestellt. Auf der Grundlage von Fallbeispielen wird den Teilnehmern des Workshops aufgezeigt, wie Fehlerkennzahlen die Beantwortung folgender Fragen bei laufenden Projekten unterstützen:

- Ist die Software bzw. ein Zwischenstand der Software reif für die nächste Teststufe?
- Ist die Software reif für die Auslieferung?
- Sind zusätzliche Maßnahmen der Qualitätssicherung (Prüfen und Testen) erforderlich? Wenn ja, welche?
- Was kann im Entwicklungsprozess geändert werden, damit Fehler mit bestimmten Merkmalen vermieden werden?
- Was kann im Testprozess geändert werden, damit entstandene Fehler mit bestimmten Merk-malen früher entdeckt werden?
- Wie weit fortgeschritten sind die Tests?

Ziel des Tutorials ist, nach einer kurzen Einleitung, vor allem der „angeleitete" Erfahrungsaustausch der Teilnehmer, um neue – praxiserprobte - Aspekte für die eigene Arbeit zu erhalten. Zielgruppen sind Testmanager, QS-Beauftragte, Qualitätsmanager und Personen, die Prozesse durch Kennzahlen überwachen und steuern.

Literaturverzeichnis

[Av08] Avcı, O.: Aus Fehlern in der Softwareentwicklung lernen. Anforderungsanalyse und Qualitätssicherung mit Fehleranalysen verbessern. Saarbrücken 2008. VDM Verlag Dr. Müller, ISBN-10: 3836476878, ISBN-13: 9783836476874.

[AW05] Avcı, O; Wagner, H.: Das chronische Problem der Anforderungsanalyse und die Frage: Fehler vermeiden oder früh entdecken? In: Cremers, A.B.; Manthey, R.; Martini, P.; Steinhage, V. (Hrsg.): Informatik 2005 - Informatik LIVE, Band 2, Lecture Notes in Informatics (LNI), Vol. P-68. Bonn 2005, S. 279-283.

[Ca98] Card, D.N.: Learning from Our Mistakes with Defect Causal Analysis. In: IEEE Software. Nr. 1, 1998, S. 56-63.

[Ch96] Chillarege, R.: Orthogonal Defect Classification. In: Lyu, M. R.: Handbook of Software Reliability Engineering. Los Alamitos (California) u. a. 1996, S. 359-400.

[Di93] Dion, R.: Process Improvement and the corporate balance sheet. In: IEEE Software. Nr. 4, 1993, S. 28-35.

Tutorium: Software-Produkt-Management

Hans-Bernd Kittlaus

InnoTivum Unternehmensberatung
Im Sand 86
53619 Rheinbreitbach
hbk@innotivum.de

Abstract: Software-Produkt-Management hat zunehmende Bedeutung und erhebliche Verbreitung sowohl bei Software-Herstellern als auch in Anwenderorganisationen erlangt. Es zielt auf den nachhaltigen ökonomischen Erfolg der Software-Produkte, seien sie selbst entwickelt oder zugekauft. In diesem Tutorium wird ein praxisorientierter Überblick gegeben über das breite Spektrum von Themen, für die ein Software-Produkt-Manager direkt oder indirekt Verantwortung trägt. Außerdem werden offene Fragen in diesem akademisch bislang sehr vernachlässigten Gebiet aufgezeigt.

1 Einführung

Produkt-Management ist eine Disziplin, mit der sich viele Branchen schon seit Jahrzehnten beschäftigen, allen voran die Konsumgüterindustrie. Die Erfindung von Produkt-Management als explizitem Steuerungskonzept wird Procter & Gamble zugeschrieben, die zwei konkurrierende Seifenprodukte 1931 jeweils mit einem Produkt-Manager versahen [Go05]. Seitdem hat sich diese Grundidee weit verbreitet. Es liegt für jedes Unternehmen nahe, die Produkte, mit denen es sein Geld verdient, die als Assets nachhaltige Werte des Unternehmens darstellen, explizit zu managen. Doch was bedeutet dieses Managen der Produkte tatsächlich? Leider ist diese Frage nur in Teilen allgemein zu beantworten, zeigt doch die Praxis, dass die Tätigkeit des Produkt-Managers stark abhängig ist von der Art des Produkts, der Kultur und Organisation des Unternehmens sowie dem Ziel- und Belohnungssystem. Grundsätzlich geht es um die Steuerung und Koordination aller relevanten Bereiche innerhalb und außerhalb des Unternehmens mit dem Ziel, den Produkterfolg nachhaltig zu optimieren. Innerhalb des Unternehmens umfasst dies insbesondere Entwicklung, Produktion, Marketing, Vertrieb und Logistik.

Die Tätigkeit eines Software-Produkt-Managers ist in wesentlichen Teilen geprägt durch die spezifischen Eigenschaften von Software. Am augenfälligsten sind die geringe Bedeutung von Produktion und Logistik und die große Bedeutung von Anforderungsmanagement, um Wunsch und Möglichkeit der häufigen Änderung des Produkts über seine Lebenszeit zu steuern.

Da Software nach Auffassung des Autors das komplexeste Erzeugnis menschlichen Handelns ist, das wir kennen, stellt auch und gerade das Managen von Software-Produkten eine einzigartige Menge von Anforderungen an die Akteure. Dementsprechend schwer tun sich Software-Hersteller mit der Besetzung von Produkt-Management-Positionen, da es keine spezifischen Ausbildungsgänge für diese Tätigkeit gibt. Dies führt typischerweise dazu, dass man in solchen Funktionen einen Mix von sehr erfahrenen Leuten findet, die im Laufe ihrer Karriere in sehr unterschiedlichen Bereichen des Unternehmens oder der Branche gearbeitet haben, und Spezialisten, die durch Ausbildung und Werdegang eine bestimmte Teilaufgabe kompetent abdecken können, z. B. Branding. Auch einschlägige Literatur zu Software-Produkt-Management, die einem Einsteiger helfen könnte sich einzuarbeiten, steht nur in geringem Umfang zur Verfügung. Unser Buch [KRS04] enthält eine umfassende Darstellung des Themas und basiert auf dem Verständnis, dass ein Software-Produkt-Manager eine starke Business-Fokussierung braucht. Mit [KC09] wird dazu ein Update mit einer ausführlicheren Behandlung des Themas Software Pricing erscheinen. [Co02], [Dv07], [La07] und [We06] teilen unsere business-orientierte Sicht mit unterschiedlicher Gewichtung, während [Sn04] ausschließlich Wartung und Weiterentwicklung sehr großer Anwendungssoftware-Systeme behandelt.

Mit der zunehmend isolierten Betrachtung, z. B. Auslagerung oder innerbetrieblicher Leistungsverrechnung, von Unternehmenseinheiten, in denen die Bereitstellung von Software eine zentrale Rolle spielt, wird auch auf der Anwenderseite die Notwendigkeit gesehen, die Software-Produkte als die entscheidenden Assets explizit zu managen. Auch die Erkenntnis, dass Unternehmen IT-fremder Branchen plötzlich zu „Standard-Software-Lieferanten“ für ihre Kunden werden, indem sie auf ihrer Website Java Applets bereitstellen (z. B. Finanzdienstleister), mag zu dieser Entwicklung beigetragen haben. Dabei wird Software-Produkt-Management bei Anwendern bisher überwiegend nicht so konsequent betrieben wie bei Herstellern. Hier liegt Verbesserungspotential.

2 Grundbegriffe

Die Marketing-Lehre definiert den Begriff „Produkt“ im Allgemeinen wie folgt: „Ein Produkt ist jedes Objekt, das auf einem Markt zur Beachtung oder Wahl, zum Kauf, zur Benutzung oder zum Verbrauch oder Verzehr angeboten wird und geeignet ist, damit Wünsche oder Bedürfnisse zu befriedigen.“ [KA07].

Wir verzichten in unserer Definition auf den Begriff des Marktes (der uns die Begriffsbildung zu sehr auf einen „Massenmarkt“ fokussiert) und stellen eher die Beziehung zwischen zwei Parteien in den Vordergrund, so dass auch Individualentwicklungen bzw. interne Kunden-/Lieferantenbeziehungen hiermit abgedeckt werden.

Produkt = Kombination aus (materiellen und/oder immateriellen) Gütern und Dienstleistungen, die eine Partei (genannt Anbieter) unter kommerziellen Interessen zusammenstellt, um definierte Rechte daran einer zweiten Partei (genannt Kunde) zu übertragen.

Software-Produkt = Produkt, dessen vorrangiger Bestandteil Software ist.

Das Wort „Partei" soll ausdrücken, dass es sich dabei nicht unbedingt um eine juristische Person handeln muss. Es könnten also auch Bereiche innerhalb eines Unternehmens oder Einzelpersonen sein. Die Formulierung „unter kommerziellen Interessen" soll deutlich machen, dass es schon um Business geht, aber nicht notwendigerweise eine Bezahlung erfolgen muss. Auch hinter Open Source steckt ein kommerzielles Interesse, und sei es nur, den etablierten Herstellern zu schaden. Auch ein kostenfreies Produkt (z. B. der Microsoft Powerpoint Viewer) hat das kommerzielle Ziel, die Verbreitung eines anderen – natürlich kostenpflichtigen – Produktes des gleichen Software-Herstellers zu erhöhen. Die Formulierung „definierte Rechte" soll ausdrücken, dass es hier Gestaltungsspielraum gibt, z. B. Nutzungsrecht (evtl. mit Beschränkungen), Eigentumsrecht, Besitzrecht, Recht zur Weiterveräußerung etc. Näheres regeln i. d. R. Lizenzbedingungen des jeweiligen Software-Herstellers oder ein Einzelvertrag zwischen den beteiligten Parteien.

Die Produktdefinition soll auch ausdrücken, dass etwas bereits ein Produkt sein kann, wenn es noch kein Kunde erworben hat, d. h. es wird nicht durch den Kaufvorgang zum Produkt, sondern durch die Absicht, es zu verkaufen. Für die Abgrenzung, was ein Software-Produkt ist und was nicht, haben wir bewusst eine „weiche" Formulierung gewählt: Das Wort „vorrangig" soll deutlich machen, dass es hier einen Ermessensspielraum gibt.

Das primäre Ziel des Software-Produkt-Managements liegt im nachhaltigen Erfolg eines Software-Produkts über dessen Lebenszyklus. Damit ist grundsätzlich ökonomischer Erfolg gemeint, der sich letztlich im erzielten Gewinn ausdrückt. Da Gewinne zeitversetzt zu Investitionen entstehen, d. h. einer Investitionsphase mit Verlusten folgt eine anhaltende hochprofitable Phase, wird häufig Kundenzufriedenheit als Maß für den Erfolg des Produkt-Managements herangezogen. Der Software-Produkt-Manager ist verantwortlich für alle Aspekte, die für den Erfolg des Produkts relevant sind.

3 Das Software-Produkt-Management-Framework

Entsprechend der angestrebten Nachhaltigkeit des Produkterfolgs muss Software-Produkt-Management in der Kombination aller relevanten Aufgaben als Dauertätigkeit betrachtet werden. Es kann nicht als Projekt gesehen werden – dann müsste es einen Beginn und ein Ende haben – noch als Prozess – dann müsste es aus einer wohldefinierten Folge von Prozessschritten bestehen. Nur Unteraufgaben können so interpretiert werden, zum Beispiel Anforderungsmanagement als Prozess bezogen auf die einzelne Anforderung, oder Entwicklung als Projekt bezogen auf ein Release.

Auch wenn Software-Produkt-Management nicht als Ganzes als Projekt oder Prozess betrachtet werden kann, ist eine gewisse Struktur wünschenswert. Dazu findet man in der Literatur kaum Ansätze. Eine Ausnahme ist das Referenz-Framework für Software-Produkt-Management, das von Inge van de Weerd, Sjaak Brinkkemper u. a. an der Universität Utrecht, Niederlande, entwickelt wurde [We06] (siehe Abb. 1).

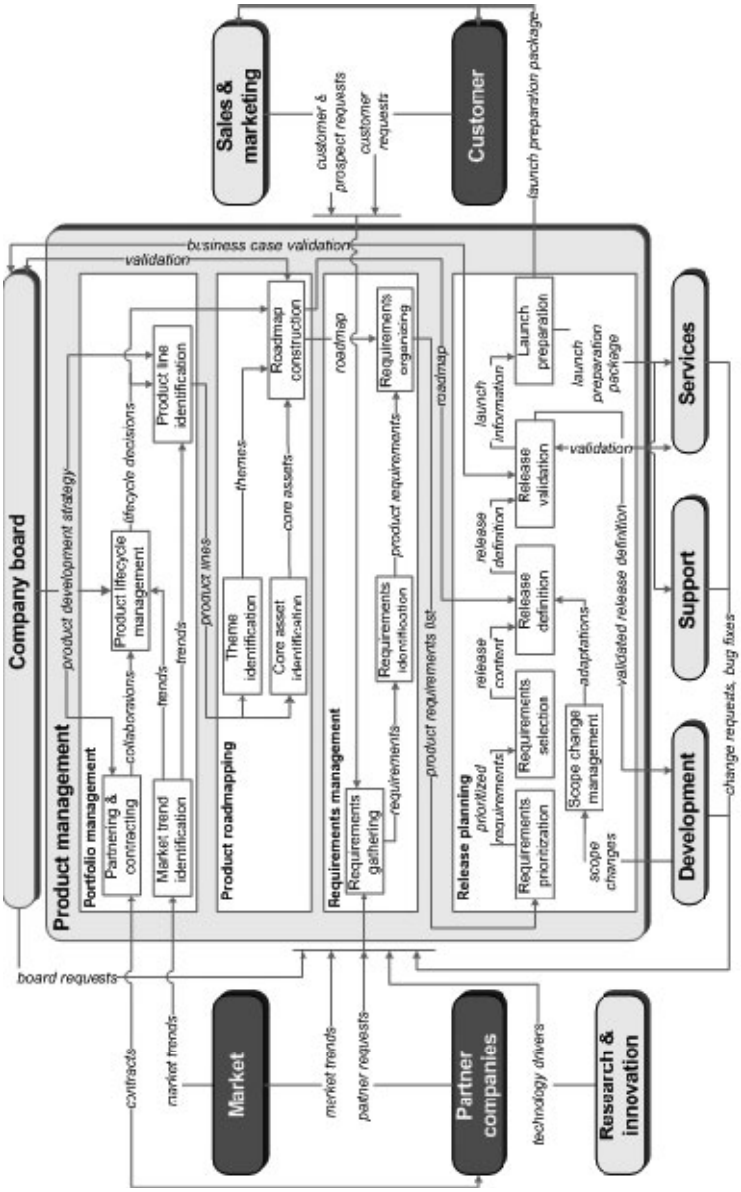


Abbildung 1: Referenz-Framework des Software-Produkt-Managements [We06]

Für dieses Referenz-Framework wurden KMUs (Kleine und mittlere Unternehmen) untersucht. Es ist fokussiert auf die Kernaktivitäten eines Software-Produkt-Managers in den Gebieten Portfolio-Management, Produkt-Roadmap, Anforderungs-Management und Release-Planung. Es basiert auf einer Hierarchie von relevanten Artifakten: Portfolio – Produkt – Release – Anforderung. Diese Artifakte sind in den Kernaktivitäten widergespiegelt. Wir betrachten dieses Framework als sehr hilfreich. Allerdings sind die ökonomischen Aspekte nur indirekt im Modell dadurch enthalten, dass der Business Case an einem der Konnektoren auftaucht. Aus unserer Sicht müssen die ökonomischen Aspekte eine wesentlich prominentere Rolle spielen.

Inspiriert durch das Pragmatic Marketing® Framework [PM08] haben wir ein eigenes Software-Produkt-Management-Framework entwickelt (siehe Abb. 2). Die Spalten zeigen die Hauptfunktionen, mit denen der Software-Produkt-Manager zu tun hat. Markt- und Produktanalyse liefern die quantitativen und qualitativen Daten zu Markt und Produkt, die die Basis der Arbeit des Produkt-Managers darstellen. Produktstrategie und –planung enthalten die Kernaktivitäten des Produkt-Managements aus Abb. 1 plus business-orientierte Aufgaben wie Business Case, Pricing und juristische Aspekte. Entwicklung, Marketing, Vertrieb und Distribution sowie Support und Dienstleistungen sind Funktionen innerhalb des Unternehmens, die typischerweise nicht an den Produkt-Manager berichten, aber von ihm orchestriert werden müssen, da sie einen erheblichen Einfluss auf den Produkterfolg haben. Es wird jeweils zwischen Aktivitäten auf Produktebene und Unternehmensebene unterschieden.

	Marktanalyse	Produktanalyse	Produkt-strategie	Produkt-planung	Entwicklung	Marketing	Vetrieb und Distribution	Support und Dienstleistung
Oft auf Unternehmens-ebene			Portfolio-Management	Ressourcen-Allokation	Ressourcen-Allokation	Marketing-Strategie und -Plan	Vertriebs-strategie und -plan	
Produktebene	Markt-forschung Markt-abschätzung Marktprobleme Technologie-Einschätzung Wettbewerbs-analyse	Produkt-kennzahlen Kunden-zufriedenheit Analyse von Gewinnung und Verlust von Kunden Opportunities	Positionierung Delivery-Modell Preismodell Preisfindung Business Case Make or buy Ökosystem Juristische Bedingungen Schutz des intellektuellen Eigentums	Roadmap Release-Plan Anforderungs-management Funktionale Spezifikation	Projektplan Technische Spezifikation Projekt-Anforderungs-management Implementierung Qualitäts-sicherung Technologie Innovation	Launch-Plan Kunden-Analyse Partner-Management Operationales Marketing Material	Kanal-vorbereitung Operationaler Vertrieb Operationale Distribution Material	Kunden-Support Technischer Support Marketing-Support Vertriebs-Support Vorbereitung von Dienst-leistungen Operationale Erbringung von Dienst-leistungen

Abbildung 2: Software-Produkt-Management-Framework

4 Details zum Tutorium

In diesem Tutorium werden die Grundlagen des Software-Produkt-Managements vorgestellt. Es orientiert sich am Software-Produkt-Management-Framework (Abb. 2) und den Büchern [KC09] bzw. [KRS04]. Die Zielgruppe sind Teilnehmer von Universitäten und aus Unternehmen, die

- Interesse an einer ökonomisch- und markt-orientierten Sicht auf das Thema Software haben,
- die Rolle eines Software-Produkt-Managers verstehen wollen, bereits wahrnehmen oder sich dahin entwickeln wollen, oder
- nach einem interessanten Forschungsgebiet suchen, das bisher im akademischen Bereich vernachlässigt ist.

Das Tutorium wird der folgenden Struktur folgen:

1. Software als Business
2. Software-Produkt: Begriffe und Merkmale
3. Das Software-Produkt-Management-Framework
4. Die Elemente des Software-Produkt-Managements
5. Organisatorische Einbettung
6. Unterschiede zwischen Herstellern und Anwendern
7. Ausblick

Literaturverzeichnis

[Co02] Condon, D.: Software Product Management: Managing Software Development from Idea to Product to Marketing to Sales, Aspatore Books, 2002

[Dv07] Dver, A.S.: Software Product Management Essentials, Anclote, Tampa, 2007

[Go05] Gorchels, L.: The Product Manager's Handbook: The Complete Product Management Resource, 3nd edition, McGraw Hill, New York, 2005

[KC09] Kittlaus, H.-B.; Clough, P.N.: Software Product Management and Pricing – Key Success Factors for Software Organizations, Springer, Heidelberg/New York, to be published in 2009

[KRS04] Kittlaus, H.-B.; Rau, C.; Schulz, J.: Software-Produkt-Management – Nachhaltiger Erfolgsfaktor bei Herstellern und Anwendern, Springer, Heidelberg, 2004

[KA07] Kotler, P.; Armstrong, G.: Principles of Marketing, 12th Edition, Prentice Hall, Upper Saddle River, 2007

[La07] Lawley, B.: Expert Product Management: Advanced Techniques, Tips and Strategies for Product Management & Product Marketing, Happy About, Cupertino, 2007

[PM08] Pragmatic Marketing, Inc.: Pragmatic Marketing Framework: on www.pragmaticmarketing.com

[Sn04] Sneed, H.M.; Hasitschka, M.; Teichmann, M.-T.: Software-Produktmanagement: Wartung und Weiterentwicklung bestehender Anwendungssysteme, dpunkt-Verlag, Heidelberg, 2004

[We06] Weerd, I. van de; Brinkkemper, S.; Nieuwenhuis, R.; Versendaal, J.M.; Bijlsma, A.: On the Creation of a Reference Framework for Software Product Management: Validation and Tool Support, Proc. of the International Workshop on Software Product Management (IWSPM'06 - RE'06 Workshop): Minneapolis/St. Paul, Minnesota, Sept. 2006

Tutorium: Wie Veränderungsprojekte wirklich funktionieren – eine praktische Anleitung

Mareike Solbach

wibas IT Maturity GmbH
Otto-Hesse-Str. 19B
64293 Darmstadt
mareike.solbach@wibas.de

Abstract: Die verbesserten Arbeitsabläufe zum Leben zu bringen und dauerhaft in der Organisation zu verankern, ist die schwierigste Aufgabe in Verbesserungsinitiativen. Im Rahmen dieses Tutoriums erarbeiten Sie zusammen mit Mareike Solbach, warum Veränderungsmanagement in Verbesserungsinitiativen notwendig ist und welche Erfolgsfaktoren dabei beachtet werden müssen. Mareike Solbach wird, gelegentlich auch mit einem Augenzwinkern, von ihren Erfahrungen aus zahlreichen Verbesserungsprojekten zur Umsetzung von CMMI, SPICE und ITIL berichten. Sie wird einen Weg aufzeigen, wie Sie Veränderungen in Ihrer IT-Organisation erfolgreich zum Leben erwecken und nachhaltig in der Organisation institutionalisieren können.

1 Details zum Tutorium

Mit dem Tutorium bekommen die Teilnehmer einen Leitfaden zum erfolgreichen Veränderungsmanagement in Verbesserungsinitiativen. Geplante Verbesserungsprojekte können damit von Anfang an richtig aufgesetzt und durchgeführt werden. Laufende Verbesserungsinitiativen können gegen die kritischen Erfolgsfaktoren geprüft und ggf. korrigiert werden.

Das Tutorium richtet sich an Anfänger, Fortgeschrittene und Experten der Zielgruppen

- IT-Management;
- IT-Projektleiter, -Mitarbeiter, -Serviceverantwortliche;
- Qualitätssicherungsbeauftragte;
- IT-Berater.