Gesellschaft für Informatik e.V. (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
• seminars
• proceedings
• dissertations
• thematics
current topics are dealt with from the vantage point of research and development, teaching and further training in theory and practice. The Editorial Committee uses an intensive review process in order to ensure high quality contributions.

The volumes are published in German or English.

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

ISSN 1617-5468
ISBN 978-3-88579-603-9

The 2012 Software Management Conference on "Nachhaltiges Software Management" (Sustainable Software Management) took place in Bielefeld from November 14th to 16th.

This volume contains the papers of four invited lectures and ten selected lessons as well as short descriptions of two tutorials giving insights into the managerial and technical aspects of sustainable software management.
All papers are in German language.

H. Brandt-Pook, A. Fleer, T. Spitta, M. Wattenberg (Hrsg.): Nachhaltiges Software Management

209

GI-Edition

Lecture Notes in Informatics

Hans Brandt-Pook, André Fleer, Thorsten Spitta, Malte Wattenberg (Hrsg.)

Nachhaltiges Software Management

14. – 16. November 2012
Bielefeld

Proceedings

Hans Brandt-Pook, André Fleer, Thorsten Spitta
und Malte Wattenberg (Hrsg.)

Nachhaltiges Software Management

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

14. – 16. November 2012
in Bielefeld

Gesellschaft für Informatik e.V. (GI)

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

Volume P-209
ISBN 978-3-88579-603-9
ISSN 1617-5468

Volume Editors

Prof. Dr.-Ing. Hans Brandt-Pook
Fachbereich Wirtschaft und Gesundheit,
Fachhochschule Bielefeld
Universitätsstraße 25
33615 Bielefeld
Email: hans.brandt-pook@fh-bielefeld.de

Dr. André Fleer
arvato systems GmbH
An der Autobahn 200
33333 Gütersloh
Email: andre.fleer@bertelsmann.de

Prof. em. Dr.-Ing. Thorsten Spitta
Fakultät für Wirtschaftswissenschaften,
Universität Bielefeld
Postfach 100131
33601 Bielefeld
Email: thspitta@wiwi.uni-bielefeld.de

M.A. Malte Wattenberg
Fachbereich Wirtschaft und Gesundheit,
Fachhochschule Bielefeld
Universitätsstraße 25
33615 Bielefeld
Email: malte.wattenberg@fh-bielefeld.de

printed by Köllen Druck+Verlag GmbH, Bonn

Vorwort

Sollte eine eher technische Fachdisziplin eine Tagung unter das Motto *Nachhaltigkeit* stellen, ein Begriff, der sicher zu den Hypes der heutigen Zeit gehört? Wir haben dies getan und denken, der Community ein interessantes und vielschichtiges Ergebnis vorlegen zu können. Auch die Veranstalter haben an Hand der Beiträge und des Reviewprozesses gelernt, den Begriff nicht einfach intuitiv zu füllen – jeder nach seiner persönlichen Sicht der Welt. Fast alle der knapp 50% angenommenen Beiträge haben entsprechende Auflagen der Gutachter zur Präzisierung erfüllt. Für die Unterzeichner dieses Vorwortes könnte eine Präzisierung wie folgt aussehen:

> *Nachhaltigkeit* bedeutet sparsame **Ressourcen-Verwendung** statt Verschwendung, **Investition** statt Verbrauch und **Langlebigkeit** eines Gegenstandes statt Zeitminimierung bei seiner Herstellung. Die drei Attribute gehen mehr oder weniger auf das ökonomische Prinzip der sparsamen Mittelverwendung in einer spezifischen Perspektive zurück.

Der Fachausschuss *Management der Anwendungsentwicklung und -wartung* des Fachbereichs Wirtschaftsinformatik der GI als inhaltliche Klammer um die eigenständigen Fachgruppen *Vorgehensmodelle*, *Projektmanagement* und *Produktmanagement* will mit dieser Tagung zur Klärung des Begriffs Nachhaltigkeit beitragen.

Zunächst ist es uns gelungen, Sprecherinnen und Sprecher für **Keynotes** zu gewinnen, die den großen Raum aufspannen, in dem sich das Tagungsthema bewegt.

Jan Jarre zeigt uns andere, allgemeinere Blickwinkel auf den Begriff Nachhaltigkeit.

Simone Rehm, IT-Managerin bei einem bedeutenden Werkzeugmaschinen-Hersteller und GI-Vizepräsidentin, macht deutlich, was Nachhaltigkeit heißen kann, wenn Software „in die Jahre“ kommt.

Karin Vosseberg und Andreas Spillner sind bekannt im Bereich des Testens, ein Gebiet, das von Universitäten eher gemieden wird. In ihrem Zwischenruf für unsere Tagung greifen Sie ein Ziel auf, das die Softwaretechnik schon seit über 30 Jahren wie einen Gral vor sich her trägt, ohne hier nennenswert weiter gekommen zu sein – die Wiederverwendung.

Karsten Lienau von arvato systems, Gütersloh, spricht mit dem Portfolio-Management ein Thema an, das in größeren Unternehmen üblich ist und auch Hochschulen gut anstünde, *bevor* sie sich in Millionen schwere Projekte stürzen, mit denen sie sich u. E. für Jahrzehnte festlegen.

Die wissenschaftlichen Beiträge sind in drei Themenbereiche gegliedert:

1. **Allgemeine Softwaretechnik** – Nachhaltigkeit muss bei der Entwicklung beginnen.
2. **Testen und Betrieb** – Erst in der Betriebs- und Evolutionsphase zeigen sich Nachhaltigkeit oder schnelle Vergänglichkeit.
3. **Software in Hochschulen** – Führt der Effizienzdruck, unter dem Hochschulen seit dem Bologna-Umbruch stehen, zur Nachhaltigkeit der Software-Dienste?

Zum ersten Themenkomplex gibt es drei Beiträge.

Klaus Quibeldey-Cirkel et al. berichten über klassische Softwareentwicklung mit Open Source-Werkzeugen und einer damit verknüpften, Schritt haltenden Qualitätssicherung. Der Fokus liegt auf einer möglichst geringen Personen-Abhängigkeit schon im Entwicklungsprozess. Dies erscheint als wichtiger Aspekt für die spätere Betriebsphase.

Katharina Peine et al. beleuchten den Entwicklungsprozess vom Ergebnis her, indem sie das IT-Produktmanagement von Unternehmen hinterfragen. Wie gehen Unternehmen mit ihrer Ressource *Software* um? Geschieht das planmäßig oder eher ad hoc? Gezeigt wird ein erster Entwurf für einen Aufgabenkatalog.

Harry Sneed stellt den herkömmlichen Begriff des *Projekts* in Frage und arbeitet heraus, dass und warum wir immer nur an Releases von *Produkten* arbeiten. Dies ist nicht neu (s. Spiralmodell von Boehm 1988, auf das er verweist). Angesichts der Verlagerung vieler Produkte und Dienste in „Clouds“ ist diese Sichtweise wieder höchst aktuell.

Zum zweiten Themengebiet liegt ein Beitrag zum wichtigen Gebiet Testen vor, drei weitere beschäftigen sich mit der Betriebsphase.

Andre Heidt et al. machen deutlich, dass Nachhaltigkeit nur durch automatisierte Retests erreichbar ist. Bei der Testautomatisierung besteht Nachholbedarf vor allem bei kleinen und mittleren Unternehmen (KMU).

Meik Teßmer präsentiert ein als Dissertation ausgearbeitetes Qualitätsmodell für Software, das ausschließlich auf dem Quellcode aufsetzt, und zwar einer maschinellen Analyse. Er greift damit das in jedem Einzelfall gepflegte Artefakt heraus, wie auch immer gearbeitet wird. Die Praktikabilität des Modells zeigt er durch eine Analyse des seit zehn Jahren in der Universität Bielefeld in allen Fakultäten eingesetzten Systems.

Bernhard Peischl et al. fokussieren ihren Beitrag auf die Betriebsphase und stellen Controlling-Aspekte in den Vordergrund. Die Autoren zeigen, dass dies nur möglich ist durch eine kontinuierliche Erhebung von Qualitätskennzahlen, die aus einer Datenbasis für ein sog. Cockpit abrufbar sind.

Christian Weber et al. geben den Gegenstand ihrer Fallstudie schon im Titel bekannt und auch den Zeitraum, von dem sie berichten. Bei 14 Jahren erscheint es nicht unangebracht, von Nachhaltigkeit zu sprechen. Bei kaum einem Medium der Benutzeroberfläche wird u. E. die Überlebensfähigkeit von Software auf eine so harte Probe gestellt, wie bei einem Intranet.

Das dritte Thema fasst die Aspekte der ersten beiden zusammen und wechselt die Perspektive von Software Management allgemein auf die Domäne *Hochschulen*. Bezeichnender Weise heißt die IT vieler Hochschulen noch immer „Rechenzentrum". Die Hochschulen befinden sich bezüglich Software und der internen Dienstleistung IT durch die gesetzlichen Vorgaben (Bachelor und Master) in einer „Aufholjagd" gegenüber der Wirtschaft, aber u. W. auch vielen Behörden. Es gibt drei Beiträge.

Thomas Degenhardt et al. betrachten das IT Service Management, das den oben erwähnten „Rechenzentren" im Sinne eines Infrastruktur- und Applikationsbetriebs noch am nächsten steht. Sie zeigen, wie man das ITIL-Rahmenwerk (*IT Infrastructure Library*) in einer Hochschule einführen und Nutzen bringend anwenden kann.

Thomas Grechenig et al. berichten über die Entwicklung, stufenweise Einführung und den Betrieb des neuen Campus-Management-Systems TISS der TU Wien. Die verschieden Aspekte von Nachhaltigkeit werden explizit beleuchtet, nicht zuletzt die Migration der Daten der Altsysteme und die neue Betriebsarchitektur, die auch für industrielle Maßstäbe ungewöhnlichen Spitzenlasten Stand halten muss.

Sebastian Frodl et al. zeigen, wie man mit einem Data Warehouse für einheitliche Daten und damit konsistente Informationen auch im Hochschulbereich sorgen kann, selbst wenn man die operative Basis nicht so schnell erneuern kann, wie man das gerne hätte.

Wir bedanken uns bei den Unterstützerinnen und Unterstützern der Tagung. Die Fachhochschule Bielefeld bietet Raum und viele helfende Hände, die zu einer erfolgreichen Tagung erforderlich sind. Die arvato systems GmbH hat finanzielle Mittel beigesteuert – auch die sind bekanntermaßen unabdingbar.

Bielefeld, im Oktober 2012

Hans Brandt-Pook André Fleer Thorsten Spitta Malte Wattenberg

Programmkomitee

Dr. Martin Bertram, Commerzbank, Frankfurt
Jens Borchers, SCHUFA Holding AG, Wiesbaden
Prof. Dr. Hans Brandt-Pook, FH Bielefeld (Leiter Programmkomitee)
Dr. Thomas Deelmann, T-Systems International GmbH, Bonn
Prof. Dr. Stefan Eicker, Uni Duisburg-Essen
Prof. Dr. Martin Engstler, Hochschule der Medien, Stuttgart
Dr. André Fleer, arvato systems, Gütersloh
Prof. Dr. Georg Herzwurm, Uni Stuttgart
Prof. Dr. Wolfgang Hesse, Uni Marburg
Reinhard Höhn, KMA Wien
Gerrit Kerber, aragon INTERACTIVE, Fellbach
Dr. Ralf Kneuper, Darmstadt
Dr. Christian Kop, Alpen-Adria-Universität Klagenfurt
Prof. Dr. Franz Lehner, Uni Passau
Dr. Oliver Linssen, Liantis GmbH & Co. KG, Krefeld
Prof. Dr. Heinrich C. Mayr, Alpen-Adria-Universität Klagenfurt
Prof. Dr. Andreas Oberweis, Karlsruher Institut für Technologie (KIT),
Prof. Dr. Wolfram Pietsch, FH Aachen
Werner Simonsmeier, Capgemini, Berlin
Harry M. Sneed, AneCon GmbH, Wien
Prof. Dr. Thorsten Spitta, Uni Bielefeld
Prof. Dr. Wolffried Stucky, Uni Karlsruhe
Prof. Dr. Karin Vosseberg, HS Bremerhaven

Inhaltsverzeichnis

Software in Hochschulen

Tutorials

Die sozialwissenschaftliche Nachhaltigkeitsdiskussion und mögliche Konsequenzen für das Softwaremanagement

Jan Jarre

Öcotrophologie
FH Münster
Corrensstr. 25
48149 Münster
Jarre@FH-Muenster.de

Abstract

Was ist mit Nachhaltigkeit eigentlich gemeint? Salopp gesprochen, geht es für uns alle darum, nicht weiter so zu tun, als hätten wir eine zweite Welt als ständige Notfallreserve im Kofferraum. Wissenschaftlicher formuliert heißt das: Die konventionelle, material- und energieintensive Wirtschaftsweise ist nicht zukunftsfähig. Wir leben in unserem „Raumschiff Erde" über unsere Verhältnisse, weil Rohstoffe und Schad-/Abfallstoffe – beim derzeitigen Stand der Technik und vielleicht sogar prinzipiell – nicht beliebig beschafft, recycelt bzw. entsorgt werden können. Also brauchen wir „nachhaltige" Handlungsalternativen.

„Nachhalt" meint etymologisch „etwas, das man für Notzeiten zurückbehält", das Rückhalt liefert. Nachhaltige Entwicklung ist das Leitbild der internationalen Umwelt- und Entwicklungspolitik. Nachhaltigkeit ist ein Normen setzendes Konzept, das uns alle zu einem veränderten Verhalten auffordert. Danach sollte jeder Mensch an seinem Arbeitsplatz, in seinem Alltag und in seinem Konsumverhalten so handeln, dass er globalen Problemlagen (Umweltschäden, Ressourcenschwund, Armut, sozialen Instabilitäten usw.) Rechnung trägt. Insbesondere sollten auch zukünftige Generationen genügend Gestaltungsmöglichkeiten haben, ihre Bedürfnisse ebenfalls befriedigen zu können.

Die Bundesregierung versteht unter „Nachhaltigkeitsmanagement" die Formulierung von Managementregeln sowie die Bindung der Politik an Ziele und messbare Indikatoren. Ausgangszustände (Nachhaltigkeitsindikatoren) werden mit angestrebten Endzuständen (politisch fixierten Nachhaltigkeitszielen) verbunden. Nachhaltigkeit wird auf diesem Wege messbar, steuerbar und evaluierbar. Die nationale deutsche Nachhaltigkeitsstrategie verfolgt so heterogene Ziele wie: Begrenzung des Flächenverbrauchs; Erweiterung der Ausgaben für die Entwicklungszusammenarbeit; Ausbau der erneuerbaren Energieträger, Erhöhung des Anteils des ökologischen Landbaus an der landwirtschaftlichen Nutzfläche usw.

Bildlich veranschaulicht wird das Nachhaltigkeitsprinzip häufig durch das so genannte Nachhaltigkeitsdreieck. Die drei Ecken sind gekennzeichnet durch die Begriffe Ökologie, Ökonomie und Soziales/Kultur. In der Praxis soll es darum gehen, alle drei Ecken derart in Einklang zu bringen, dass gerechte Lebenschancen weltweit für die heutige wie für die folgenden Generationen möglich sind. Umstritten ist dabei, ob alle drei „Ecken" gleichgewichtig zu berücksichtigen sind oder ob der Ökologie Vorrang zukommt. Ein ganzheitliches und gleichgewichtiges „Dreisäulentheorem" ist ausgesprochen zerbrechlich, weil es zum „Hineininterpretieren" beliebiger Interessen geradezu einlädt.

Offenkundig ist, dass zwischen den Ecken des Dreiecks zahlreiche Zielkonflikte lauern, die nicht so ohne weiteres entschärft werden können. Beispielsweise ist eine Verlagerung von Arbeitsplätzen in die ärmeren Länder der Welt unter weltweiten Gerechtigkeitsgesichtspunkten sinnvoll. Dadurch aber wird zugleich die aktuelle Arbeitslosigkeit, Armut und soziales Leid in den Industrieländern erhöht und in der Folge die Akzeptanz für eine Politik der Nachhaltigkeit in Frage gestellt. Spannungsfelder dieser Art werden häufig im gesellschaftlichen Nachhaltigkeitsdiskurs überdeckt, indem vorrangig über „Win-Win-Situationen" gesprochen wird. Dies ist z.B. der Fall, wenn Individuen Geldvorteile erlangen, weil sie Energie einsparen und dadurch zugleich die nationale Energie- wie auch die CO2-Bilanz verbessert wird.

Wie kann Nachhaltigkeit realisiert werden? Johan Galtung, ein weltweit bekannter Friedens- und Konfliktforscher, hat auf die Frage, wie die Umsetzung der Nachhaltigkeitsforderung möglich sei, geantwortet: "Buddhistische Eskimos!" und wollte sagen: Wir müssen nicht nur Buddhisten in unseren Ansprüchen, wir müssen auch Eskimos in unseren Anpassungsfähigkeiten werden. Das bedeutet, wir müssen die Technik weiter verbessern, bei gleichen Produktionsmengen die Schadstofffreisetzungen verringern, den Energie- und Ressourcenverbrauch reduzieren, insgesamt also die Effizienzverbesserung unserer Wirtschaftsprozesse vorantreiben. Wir müssen -besser noch- die Anpassung so gestalten, dass die in der Produktion verwendeten Stoffe im Kreislauf geführt oder biologisch vollständig wieder verwertet werden (Konsistenz). Als dritte Umsetzungsstrategie wird die Bescheidenheit, die Genügsamkeit, der Verzicht (Suffizienz) propagiert, nicht im Sinne der Askese, sondern im Sinne einer neuen positiven Lebensqualität („Ballast abwerfen"). „Mehr sein als haben": Ein Mehr an Lebensfreude könnte mit der Erfüllung immaterieller Bedürfnisse und z.B. mit einer Befreiung von der „Tyrannei des Auswählen-Müssens" einhergehen. Ein „immer mehr" könnte durch ein „immer besser" abgelöst werden, so die Hoffnung.

Das Angenehme auf dieser Welt soll nicht abhandenkommen und muss, wenn es abhanden zu kommen droht, durch Wandel auch für nachfolgende Generationen erhalten bleiben: Das ist die tiefere Hoffnung des Suchprozesses „Nachhaltigkeit". Aus der Sicht der Bundesregierung ist Nachhaltigkeit ein „dynamischer, gesamtgesellschaftlicher Reformprozess". Aber Reform und Wandel schaffen stets Gewinner **und** Verlierer, damit Interessengegensätze und gesellschaftliche Kontroversen. An die von einigen ersehnte, neue Harmonie, an ein „nachhaltiges Gleichgewicht", kann sich die Gesellschaft realistischer Weise nur dann herantasten, wenn zunächst einmal ein längerer Konflikt um Richtungen, Macht, Ressourcen, Technologien, aber auch um Begriffe, Geschichten und verheißungsvolle Perspektiven einkalkuliert und ausgetragen wurde.

„Verheißungsvolle Perspektiven“ gilt es auch im Bereich des Softwaremanagements zu erörtern. Anknüpfungspunkte sind sicherlich der Ressourcenschutz, insbes. die Energieeffizienz, aber z.B. auch die Frage, wo und inwieweit Hardware gezielt durch Software ersetzt werden kann: Früher musste ein zusätzlicher Apparat angeschafft werden, heute reicht häufig eine neue App (Stichwort „iPhone-Ökonomie“). Weitere nachhaltige Perspektiven sind mit Sicherheit noch zu entdecken.

Sanierung, Kauf oder Neubau – was tun, wenn Software in die Jahre kommt?

Simone Rehm

IT + Prozesse
TRUMPF GmbH + Co. KG
Johann-Maus-Straße 2
71254 Ditzingen
simone.rehm@de.trumpf.com

Abstract

Menschen, die nicht aus der Software-Branche kommen, mag das überraschen, aber Software altert. Softwareexperten wissen das zwar, stehen aber dennoch vor einer schwierigen Aufgabe, wenn sich abzeichnet, dass eine Sanierung oder Modernisierung ansteht. Meist sind es gar nicht die Anwender, die diesen Handlungsbedarf erkennen. Vor allem bei Eigenentwicklungen, die über Jahre und Jahrzehnte immer wieder an die Bedürfnisse der Anwender angepasst worden sind, zeigen sich die „Alterserscheinungen" eher in begrenzter Erweiterbarkeit oder mangelnder Wartbarkeit der Software als in der Funktionalität. Das heißt, der Impuls für die Erneuerung kommt meist aus der IT. Trotzdem muss man die Anwender für ein solches Projekt gewinnen, denn häufig ist mit dem Renovieren ein Einfrieren des bestehenden Systems verbunden. Das heißt, die Anwender müssen ihre Anforderungen nach Weiterentwicklung vorerst zurück stellen. Hinzu kommt die Furcht, dass sie sich von Alt-Vertrautem verabschieden müssen und der erworbene Komfort bei einer Neugestaltung möglicherweise verloren geht. Beides erfordert ein hohes Maß an Überzeugungsarbeit im Vorfeld eines solchen Projekts.

Hat man erst einmal die Zustimmung aus der Anwenderschaft, so stellt sich die Frage: Sanierung, Neubau oder gar Neukauf? Am realen Beispiel eines eigenentwickelten Systems zur Planung und Durchführung von Servicetechnikereinsätzen zeigt der Vortrag auf, in welchen Stufen bei TRUMPF eine Bestandssoftware systematisch analysiert wurde und welche Aspekte im vorliegenden Fall gegen ein Software-Refactoring und für eine komplette Neuentwicklung auf Basis einer Smart-Client-Zielarchitektur sprachen. Die geschätzten Kosten für den Neubau lagen allerdings in derselben Größenordnung wie die Kosten für die Einführung eines Standardprodukts. Deshalb gilt es nun, weitere Kriterien für die Entscheidung „Make-or-Buy" herauszuarbeiten.

Dazu zählen zum einen strategische Aspekte, wie etwa die Abhängigkeit vom Hersteller. Da von diesem Hersteller auch die ERP-Software stammt, die TRUMPF einsetzt, würde sich das neue Softwareprodukt leicht mit den ERP-Prozessen integrieren lassen. Auf der

anderen Seite steigt dadurch die Abhängigkeit vom Hersteller. Wer hat die Weiterentwicklung in der Hand? Wie flexibel kann der Anbieter auf neue Anforderungen eingehen? Welche Flexibilität wird aus fachlicher Sicht überhaupt gebraucht? Wie schnell kann der Anbieter auf technologischen Wandel eingehen?

Diese und weitere qualitative Kriterien fließen nun in eine Nutzwertanalyse ein. Gemeinsam mit dem Fachbereich werden sie zusammengetragen, gewichtet und bewertet. Parallel dazu wird der Funktionsumfang des Bestandssystems durchleuchtet. Ziel dabei ist es, ebenfalls gemeinsam mit dem Fachbereich zu ermitteln, welche Use Cases das System ausmachen, in welchen dieser Use Cases bereits heute ein hoher Grad an Individualität steckt und wo diese Individualität für den Geschäftserfolg entscheidend ist und daher beibehalten werden soll.

In einem weiteren Schritt wird dann für alle Use Cases mit hohem Individualisierungsgrad Fall für Fall bewertet,

a) ob und wie sich in der Zielarchitektur für das eigenentwickelte Neusystem diese Individualität abbilden ließe,

b) ob sie sich auch beim Einsatz eines Standardprodukts im Rahmen eines Customizings abbilden ließe und

c) wie hoch der Aufwand dafür wäre und ob er die Grenzen eines vertretbaren Customizings sprengen würde.

Auf Basis dieser Bewertungen wird dann in einem finalen Schritt entschieden, welchen Weg das Unternehmen TRUMPF bei der Ablösung des Bestandssystems gehen wird. Im Vortrag werden die sich ergänzenden Bewertungsschemata detailliert vorgestellt. Auf die Ergebnisse der Bewertung wird ebenso wie auf die vorstellbaren Migrationsszenarien beim Übergang auf ein neues System näher eingegangen.

Same procedure as ...

Karin Vosseberg, Andreas Spillner

Hochschule Bremerhaven, Informatik/Wirtschaftsinformatik
An der Karlstadt 8, 27568 Bremerhaven
karin.vosseberg@hs-bremerhaven.de

Hochschule Bremen, Fakultät Elektrotechnik und Informatik,
Flughafenallee 10, 28199 Bremen
andreas.spillner@hs-bremen.de

Abstract

Bei der Entwicklung von Softwaresystemen ist agiles Vorgehen derzeit angesagt. In vielen Softwarefirmen wird bereits agil entwickelt oder darüber nachgedacht agil zu werden. Scrum ist dabei das bevorzugte Vorgehen. Kernstück der agilen Entwicklung ist die Umsetzung von kleinen Aufgaben, die innerhalb eines Sprints (in der Regel max. 4 Wochen) vollständig zu erledigen sind. Das Gesamtsystem entsteht in iterativen Schritten.

Als in den 90er Jahren die Objektorientierung Einzug in die softwareentwickelnden Firmen genommen hat, ist die Wiederverwendung von Softwareteilen als ein großer Vorteil der Objektorientierung propagiert worden. Wiederverwendung »innerhalb« des Systems über die Vererbung ist umgesetzt worden; systemübergreifende Wiederverwendung von Klassen eher weniger. Wiederverwendung von ganzen Systemteilen (Komponenten) ist selten in der Praxis anzutreffen.

Cloud Services stellen Dienste zur Verfügung, die in Kombination ganze Anwendungen (oder Teile davon) realisieren. Hier ist Wiederverwendung umgesetzt. Services werden von Dienstleistern zur Nutzung angeboten und Softwaresysteme werden aus unterschiedlichen Diensten »kombiniert« (und nicht mehr programmiert).

Im Vortrag werden unter anderen folgende Fragen diskutiert:

- Wie kann bei agilem Vorgehen mehr Wiederverwendung betrieben werden?
- Welche Anknüpfungspunkte bestehen zwischen Cloud Services und agiler Entwicklung?
- In welche Richtung muss sich die Softwareentwicklung bewegen, um Wiederverwendung besser zu ermöglichen?

Wir werden eher Fragen aufwerfen, ohne Antworten oder Lösungen parat zu haben, hoffen aber damit eine intensive Diskussion mit den Zuhörern und nach der Tagung auch in den Firmen zu initiieren.

Aktives Portfolio-Management als Wertbeitrag in einem IT-Service Unternehmen

Karsten Lienau

arvato Systems
An der Autobahn 200
33333 Gütersloh
karsten.lienau@bertelsmann.de

Abstract

Die unterschiedlichsten IT Unternehmen werden am Markt häufig als einheitliche Klasse „IT Unternehmen“ gesehen. Bei genauerer Betrachtung kann eine Unterscheidung in Anbieter von (1) IT-Geräten/Standardsoftware, (2) IT-Projekten und (3) IT-Services getroffen werden. Obwohl es auch Unternehmen gibt, die zwei oder drei dieser Klassen unter einem Firmennamen vereinen, sind diese Unternehmen typischerweise pro Klasse in eigenständige Business Units gegliedert. Diese agieren betriebswirtschaftlich und operativ wie eigenständige Firmen. Betrachtet man die Bedeutung eines aktiven Portfolio-Managements in einem IT Unternehmen, können somit auch Mischunternehmen jeweils nach den o. g. Klassen getrennt betrachtet werden.

Im Folgenden sollen IT Unternehmen des Typs (3) betrachtet werden, deren Geschäftsmodell darin besteht, ihren Kunden einen ständig laufenden IT Service zu gewährleisten, der typischerweise über vertraglich geregelte Service Level Agreements (SLA) qualitativ beschrieben ist. Bei IT Unternehmen des Typs (1), die entweder IT- Geräte (wie bspw. Drucker, PCs und Notebooks, Server, Netzwerkrouter) oder Standardsoftware (wie bspw. PC-Betriebssysteme, Office-Anwendungen, ERP-Anwendungen) herstellen, wird der Begriff „Produkt“ unmittelbar intuitiv richtig interpretiert: Das „Produkt“ sind eben solche Geräte oder Software. Im Gegensatz dazu fällt die Begriffsbestimmung des „Produkts“ bei IT-Services nicht so einfach aus. Was ist ein Service, und was ist ein Produkt? Worin bestehen die Unterschiede? Der Schlüssel zur Beantwortung dieser Frage liegt in der Herkunft des Wortes „Produkt“. Jedes Wirtschaftsunternehmen verbindet Material und Arbeitskraft zu einem neuem Wert, dem „Produkt“ seiner Arbeit. Der Wert bei IT-Service Unternehmen ist aber genau der IT-Service; somit ist in diesem Umfeld der Begriff *Produkt* synonym zum Begriff *Service* zu sehen.

Gleichzeitig verbindet man mit dem Begriff *Produkt* eine Reproduzierbarkeit, d.h. eine wiederholte Nutzbarkeit einer gleichen Sache. Gerade im IT Service Geschäft waren die Anfänge der Branche meist durch *Projekte* geprägt. Diese zeichnen sich durch eine konkrete Anforderung eines Unternehmens an die IT aus, die dann durch eine speziell auf diese Situation zugeschnittene IT Lösung abgedeckt wird. Die Realisierung der IT Lö-

sung erfolgt in einem dedizierten *Projekt*. Die Wiederverwendbarkeit im Sinne eines standardisierten *Produkts* ist häufig nicht gegeben. Für ein IT Unternehmen des Typs (3) ergibt sich somit ein Widerspruch, denn das Geschäftsmodell sollten keine *Projekte* sondern *Produkte* sein. Der Weg aus dem Projektgeschäft hin zu standardisierten IT Services, also *Produkten*, beginnt bei den Anforderungen. Werden individuelle Anforderungen an einen IT Service Provider heran getragen, mündet dies regelmäßig in einem individuellen Projekt. Dreht man dies um, also stellt der IT Service Provider die Anforderungen auf und trägt diese zu den Kunden, kann dies standardisiert geschehen und es können in der Tat *Produkte* etabliert werden, die vielfach produziert und an Kunden verkauft werden.

Auf den ersten Blick kommen wir somit aber zu einem naiven Geschäftsmodell: der Dienstleister diktiert die Anforderungen und der Kunde kauft diese. Prinzipiell ist dies richtig, allerdings noch nicht vollständig. Es kommt ein weiterer Aspekt hinzu, der das Produkt- respektive Portfolio-Management bei einem IT Service Provider erst sinnvoll macht: der Vertriebsprozess. Der IT Markt sowie die funktionale Machbarkeit in der IT werden zunehmend komplex und für die Verantwortlichen in einem beliebigen Wirtschaftsunternehmen immer weniger durchschaubar. Mit welchem IT Angebot am Markt werden die Bedarfe des Unternehmens gedeckt? Hier steckt der Ansatzpunkt für den Vertrieb von IT Produkten. Hat der Provider die Kompetenz aufgebaut, den Business Bedarf seiner Kunden zu kennen und zu verstehen, lässt sich ein potentieller Kunde bei seinem individuellen Bedarf adressieren und daraus systematisch eine Lösung ableiten, aufbauend auf vordefinierten und standardisierten Produkten. An dieser Stelle findet die eigentliche Leistung statt: Beim Ableiten der Lösung müssen stets die Produkte im Fokus stehen, ansonsten droht erneut ein individuelles Projekt. Argumentativ muss besonders Wert auf die Zukunftsfähigkeit der Lösung gelegt werden, die umso höher ist, je mehr Marktstandards und -trends berücksichtigt werden. Diese Kompetenz innerhalb des Vertriebs wird häufig als *Sales Consulting* oder *Solution Architecture* bezeichnet und stellt die eigentliche Brücke zwischen Serviceerbringung (Herstellung) und Kunde (Nutzung) dar.

Zusammenfassend lässt sich feststellen, dass eine strategische und langfristige Service-Entwicklung im Sinne eines *Produkt Managements* in einem IT Service Unternehmen nur dann sinnvoll ist, wenn alle Bereiche des Unternehmens, insbesondere die Vertriebsbereiche, Teil dieser Struktur sind. Fehlt diese Integration, bleibt nur noch das Angebot von selbsterklärenden Services, wie man sie meist bei Anbietern von *Cloud Services* findet, die ihre Produkte über sogenanntes *Self Provisioning* an die Kunden vertreiben. Hierbei ist das Aufdecken des Kundenbedarfs in Form einer Beratung (Sales Consulting) nicht ausschlaggebend. Anders formuliert stellt sich der Wertbeitrag eines aktiven *Produkt* oder *Portfolio Managements* für das IT-Service Unternehmen primär durch das operative Einhalten des strategischen Wegs des Unternehmens dar. Ein nicht intendiertes Abgleiten sowohl zum Projektgeschäft als auch zum Cloud Anbieter wird effektiv verhindert. Aspekte wie Kosten/Nutzen oder Qualität können vielmehr als sekundär betrachtet werden.

Nachhaltige Software-Entwicklung mit Open-Source-Tools und automatisierten Workflows

Klaus Quibeldey-Cirkel, Christoph Thelen

Fachbereich Mathematik, Naturwissenschaften und Informatik
Technische Hochschule Mittelhessen
Wiesenstr. 14
D-35390 Gießen
klaus.quibeldey-cirkel@mni.thm.de
christoph.thelen@mni.thm.de

Abstract: Durch automatisierte Abläufe, eine darin fest verankerte Qualitätssicherung und zahlreiche Feedback-Stufen lässt sich der Wartungsaufwand von Software deutlich verringern. Zum einen werden neue Entwickler schneller in das Projekt integriert, da sie einen roten Faden vorfinden. Zum anderen entlasten automatisierte Build-, QA- und Deployment-Prozesse die Hauptentwickler und Administratoren. Umgesetzt mit renommierten Open-Source-Werkzeugen sorgt dieser pragmatische Ansatz dafür, dass die Software-Entwicklung in Teams mit hoher Fluktuation nachhaltig, das heißt unter Wahrung der erreichten Softwaregüte erfolgen kann.

1 Einleitung

Der Begriff „Nachhaltige Entwicklung“[1] fokussiert auf die verantwortungsvolle Nutzung von Ressourcen. Die Ressourcen in der Software-Entwicklung sind vor allem die Entwickler und die Qualität der bislang entwickelten Software. Beide sind eng verknüpft: Werden die Entwickler-Ressourcen strapaziert, geht das meist auf Kosten der Qualitätssicherung; eine Verschlechterung der Qualität bindet die Entwickler an Wartungsaufgaben. Das kann dazu führen, dass Software entwickeln zu einer nervenaufreibenden Angelegenheit wird. Besonders dann, wenn das Projekt ein Altsystem ist, das einen gewissen Umfang überschritten hat, eine hohe Fluktuation an Entwicklern aufweist und nur wenige Hauptentwickler kontinuierlich daran arbeiten.

Müssen neue Entwickler in ein Projekt integriert werden, zum Beispiel bei Neueinstellungen, Umstrukturierungen oder Firmenzusammenschlüssen, haben die Hauptentwickler und Administratoren alle Hände voll zu tun, die Arbeit zu koordinieren und dafür zu sorgen, dass die Qualität und der Betrieb der Software durch Code-Änderungen nicht beeinträchtigt werden. Die eigentlichen Wartungsaufgaben bleiben dabei oft auf der Strecke. Dies ist auf Dauer dem Projekt nicht zuträglich und widerspricht dem Gedanken einer nachhaltigen, das heißt *qualitätserhaltenden* Software-Entwicklung.

[1] http://de.wikipedia.org/wiki/Nachhaltige_Entwicklung (Abruf 28.07.2012)

Die an der THM entwickelte und betriebene Kollaborationsplattform *eCollab*[2] stand vor der Herausforderung, neue Entwickler so schnell wie möglich in das Projekt zu integrieren. Die Entwickler sind Studierende aus Informatik-Lehrveranstaltungen, in denen die Plattform weiterentwickelt wird. Da sich die Weiterentwicklung am Semesterbetrieb orientiert, herrscht eine hohe Fluktuation an Entwicklern: Die meisten Studierenden sind nur wenige Monate in das Projekt involviert und scheiden am Ende des Semesters wieder aus. Somit bleibt für alle Beteiligten nur wenig Zeit, sich intensiv mit der Software auseinanderzusetzen. Die ständigen Projektmitarbeiter waren hauptsächlich damit beschäftigt, neue Entwickler in das Projekt einzuweisen und deren erste Änderungsversuche zu überprüfen.

Auf Grund dieser Problematik entstand im Laufe der Zeit mit Hilfe von Open-Source-Werkzeugen ein automatisierter Prozess. Er besteht aus mehreren miteinander verbundenen Abläufen (Abb. 1), die für ein kontinuierliches Feedback und eine automatisierte Qualitätssicherung (QA) sorgen. Der Prozess ist für jeden Entwickler als individueller Entwicklungsablauf zu verstehen und schafft so einen roten Faden, an dem er sich orientieren kann.

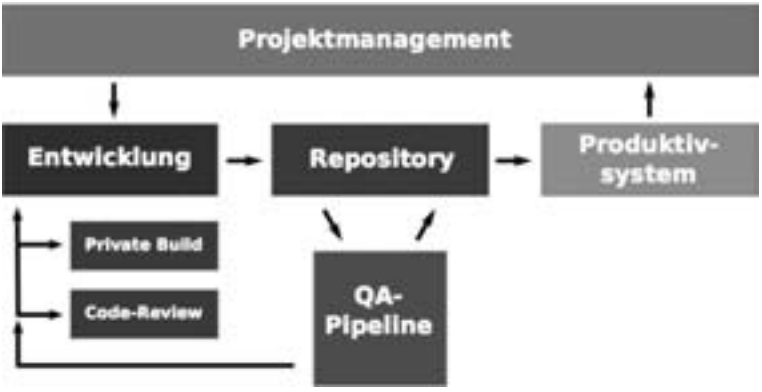


Abbildung 1: Entwicklungs-Workflow der Kollaborationsplattform *eCollab*

Konkret bringt der Prozess zwei Vorteile: Erstens wird neuen Entwicklern die Hand gereicht, sodass ihnen durch einen beinahe geführten Ablauf der Einstieg in das Projekt erleichtert wird. Zweitens sorgt die automatisierte Qualitätssicherung dafür, dass Fehler nicht in das produktive System gelangen. Die Hauptentwickler werden dabei gleichzeitig entlastet, da sie den Ablauf nicht überwachen müssen. Des Weiteren ist im Prozess die Möglichkeit vorgesehen, vor der Qualitätssicherung bei Bedarf ein formales Code-Review durchzuführen, von dem die Entwickler zusätzlich profitieren können.

Dies ist es, was unter nachhaltiger Software-Entwicklung zu verstehen ist: Die Software bleibt dauerhaft und für alle Beteiligten in einem beherrschbaren Zustand. Um diesen Zustand beizubehalten, ist eine ständige und möglichst umfassende Qualitätsüberwa-

[2] Die Plattform *eCollab* ist in Umfang (ca. 900.000 LOC) und Funktionalität vergleichbar mit den Lernplattformen *Moodle* und *ILIAS*, siehe die Gegenüberstellung unter https://ecollab.thm.de/infos/impressum.php.

chung notwendig. Sie beinhaltet neben den funktionalen Aspekten vor allem auch eine Bewertung der Design-Güte in Form von Metriken, anhand derer ein möglicher Qualitätsverfall sichtbar wird. Nur wenn sie automatisiert ist, kann sie an verschiedenen Stellen in den Prozess integriert werden: während der Entwicklung durch den *Private Build* auf dem lokalen System und später für alle in der *QA-Pipeline.* Letztere verhindert aktiv Raubbau am Software-Design, indem problematische Änderungen im *Repository* abgewiesen und zur Überarbeitung zurück an die Entwickler gesendet werden.

Für die Umsetzung des Prozesses kommen in der Open-Source-Welt bekannte Tools zum Einsatz: *Git* und *Gitorious* für die Versionsverwaltung aller Projektquellen und der Ablaufsteuerung sowie *Jenkins* als Continuous-Integration-Server für die automatisierte Qualitätssicherung. Mit *Redmine* steht außerdem ein Projektmanagement-Werkzeug mit Unterstützung der Scrum-Methode zur Verfügung.

2 Evolution des Quellcodes

Um die Nachhaltigkeit der Softwaregüte bewerten zu können, sind Software-Metriken erforderlich; sie liefern Fakten über die quantitativen und qualitativen Merkmale des Projekts. Am Beispiel der Kollaborationsplattform *eCollab* werden im Folgenden die Metriken zusammengetragen, die für eine Bewertung der Softwaregüte infrage kommen.

Der Online-Dienst *Ohloh* berechnet für Open-Source-Projekte aus den Repository-Daten der gesamten Projekthistorie (Quellcode und Commits) verschiedene quantitative Metriken. Diese geben einen Überblick, wie das Projekt während seiner Laufzeit gewachsen ist. Dabei wird zum einen auf die Code-Zeilen Bezug genommen, zum anderen aber auch auf die Aktivität der Entwickler. Beide Punkte werden verknüpft, um den Einfluss einzelner Personen auf das Projekt zu verdeutlichen:

- „Contributors“: Es wird der Prozentsatz der Commits berechnet, die einzelne Entwickler am Projekt insgesamt, in den letzten zwölf Monaten und in den letzten 30 Tagen getätigt haben (Abb. 2). Für *eCollab* ist interessant, dass 50% aller Commits von Entwicklern stammen, die nicht mehr in das Projekt involviert sind. Für die vergangenen zwölf Monate beträgt dieser Wert 25%.
- „Languages“: Übersichten der im gesamten Projektverlauf zum Einsatz kommenden Programmiersprachen, deren Anteil an der Gesamtgröße und welcher Art die Zeilen sind (Code, Kommentar, Leerzeile). Es wird auch ermittelt, welcher Entwickler welche Sprachen in welchem Umfang in das Projekt eingebracht hat.

Interessante qualitative Metriken liefern „Instability vs. Abstraction“ [Ma94] und die „Metriken-Pyramide“ [LM06], siehe Abb. 3 und Abb. 4. Im Vergleich zur Code-Analyse von *Ohloh* wird hier aber nicht die gesamte Projekthistorie, sondern der Status quo der Software berücksichtigt. Dies ist insofern als problematisch anzumerken, da sich die Nachhaltigkeit so nicht direkt erkennen lässt. Erst über eine Trend- bzw. Langzeitanalyse kann auf einen Blick geurteilt werden, ob die Maßnahmen letztlich zu einer Wahrung oder gar Verbesserung der Qualität führen.

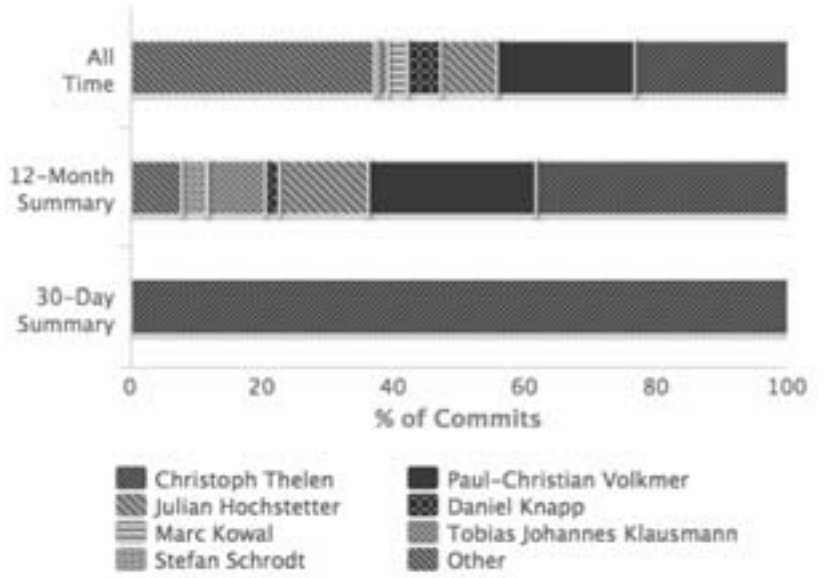


Abbildung 2: Anteil der Commits einzelner Entwickler am Gesamtprojekt.[3]

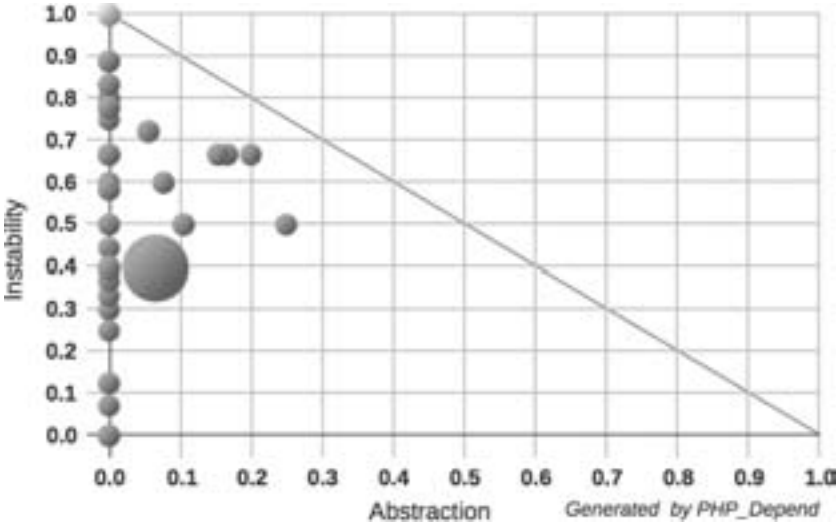


Abbildung 3: „Instability vs. Abstraction"-Diagramm nach [Ma94]

[3] https://www.ohloh.net/p/estudy/contributors/summary (Abruf 28.07.2012)

„Instability vs. Abstraction“ charakterisiert Pakete[4] anhand ihrer Instabilität (I), also wie stark sie an andere Pakete gekoppelt sind, und der Abstraktheit (A), das heißt, wie hoch der Anteil an abstrakten Klassen oder Schnittstellen innerhalb eines Pakets ist (Abb. 3). Berechnet wird die Instabilität anhand der eingehenden und ausgehenden Kopplung. Es wird eine Ideallinie (A+I=1) definiert, welche das beste Verhältnis aus Instabilität und Abstraktheit anzeigt. Je weiter sich die Pakete von der Ideallinie entfernen, desto anfälliger sind sie für Änderungen in anderen Paketen.

Ein wichtiger Aspekt der Nachhaltigkeit der Softwaregüte wird somit ausgedrückt: Werden Änderungen an einem Teil der Software durchgeführt, kann das Konsequenzen für andere Bereiche haben. Diese müssen dann ebenfalls geändert werden. Dadurch erhöht sich der Aufwand für Wartung und Qualitätssicherung.

Mehr ins Detail geht die Metriken-Pyramide (Abb. 4), welche drei strukturelle Aspekte der Software beleuchtet: Größe und Komplexität im linken Teil, Kopplung im rechten und Vererbung in der Spitze der Pyramide. Der linke Teil besteht zunächst erneut aus einer quantitativen Analyse, die allerdings viel tiefer geht. Hier findet eine Inventur der Software statt: Anzahl der Pakete oder Namensbereiche (NOP), Anzahl der Klassen (NOC), Anzahl der Methoden (NOM), Gesamtzahl der Code-Zeilen (LOC) und schließlich die zyklomatische Komplexität nach McCabe (CYCLO). Im rechten Teil wird die Anzahl der Methodenaufrufe (CALLS) und Kopplungen zu anderen Klassen (FANOUT) gezählt. Die Spitze bildet schließlich die durchschnittliche Anzahl von abgeleiteten Klassen (ANDC) und die durchschnittliche Höhe der Klassenhierarchie (AHH). Die ermittelten Werte werden im inneren Teil der Pyramide abgelegt.

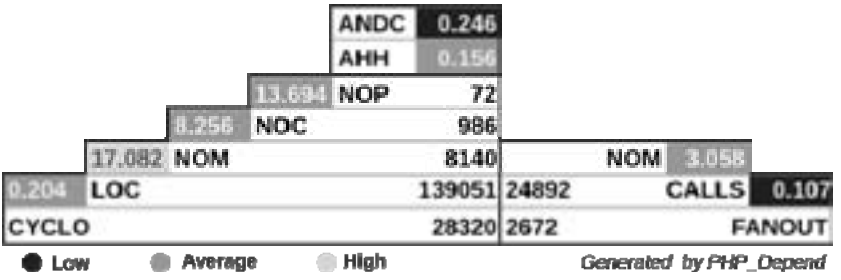


Abbildung 4: Metriken-Pyramide nach [LM06]: Jeweils zwei übereinander stehende Werte in der Mitte der Pyramide werden dividiert und das Ergebnis an der Kante des jeweils oberen Wertes niedergeschrieben. Zusammen mit der farblichen Klassifizierung sorgen diese Zahlen für projektübergreifend vergleichbare Werte.

Die Zählungen allein sind noch nichts Besonderes; zusätzlich wird an den Kanten der Pyramide das Verhältnis zweier aufeinanderfolgender Werte festgehalten, wobei jeweils der untere Wert der Dividend des nächsthöheren ist [LM06]. In Abb. 4 ist zum Beispiel

[4] Robert C. Martin spricht von „Kategorien“ und bezieht sich auf die Modellierungssprache von Grady Booch.

die zyklomatische Komplexität pro Zeile Code (CYCLO/LOC = 0,204), die Anzahl Methoden pro Klasse (NOM/NOC = 8,256) oder die Anzahl externer Kopplungen pro Methodenaufruf (FANOUT/CALLS = 0,107) zu sehen. Die Durchschnittswerte sind gemäß den Referenzwerten aus [LM06] farblich klassifiziert (Low/Average/High) und erlauben so die Vergleichbarkeit mit anderen Projekten, selbst wenn diese größer oder kleiner sind. Insgesamt gibt die Metriken-Pyramide einen Einblick in die Qualität der Code-Organisation.

3 Kontinuierliches Feedback

Für die Entwickler ist ständiges und individuelles Feedback wichtig, wenn sie schnell auf Probleme reagieren sollen. Fehler sollten daher möglichst schon bei oder kurz nach ihrer Entstehung entdeckt werden. Je mehr Zeit zwischen dem Einführen und dem Erkennen eines Fehlers liegt, desto schwieriger kann später auch dessen Korrektur sein, da sich die Entwickler möglicherweise bereits mit völlig anderen Aufgaben beschäftigen. Daher werden mehrere Feedback-Stufen definiert, an denen sich die Entwickler orientieren können (Abb. 5). Jede Stufe gibt den Entwicklern eine zusätzliche Absicherung, dass eventuell auftretende Probleme sofort aufgedeckt werden. Dabei sind die ersten Stufen noch auf dem Entwickler-PC angesiedelt, sodass Fehler bereits dort behoben werden können. Sie werden damit erst gar nicht an die übrigen Entwickler verteilt.

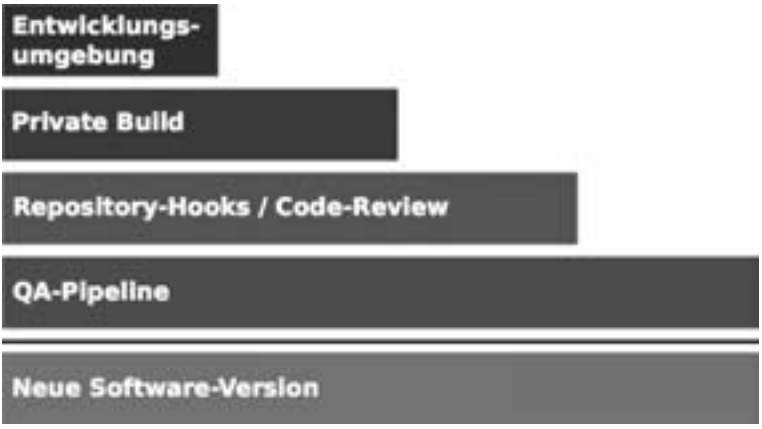


Abbildung 5: Feedback-Phasen im Entwicklungsprozess

Feedback wird auf verschiedenen Wegen von der automatisierten Qualitätssicherung zunächst nur an die beteiligten Entwickler, später im Entwicklungsprozess dann an das gesamte Team geliefert. Jede Stufe ist als Miniatur-Meilenstein in der Entwicklung anzusehen, sodass eine Code-Änderung stets überwacht wird, während sie den gesamten Entwicklungsprozess durchläuft.

Idealerweise werden Probleme bereits während der Arbeit in der Entwicklungsumgebung (IDE) markiert, sodass sie schon mit ihrer Entstehung verhindert werden können.

Typischerweise betrifft das Verstöße gegen die im Projekt vereinbarten Programmierrichtlinien und Fehlschläge der Unit-Tests.

Im zweiten Schritt hilft der „Private Build“, mögliche Integrationsfehler zu beheben: Der gesamte QA-Prozess kann auf dem eigenen Entwickler-PC ausgeführt und nachvollzogen werden. Dies geschieht kurz vor dem Check-in neuer Änderungen in das Entwickler-Repository.

Sollen die Code-Änderungen mit anderen Entwicklern ausgetauscht werden, müssen sie den im Projekt vereinbarten Vorgaben entsprechen. Sollten Verstöße zuvor in der IDE ignoriert worden sein, werden die Änderungen spätestens an dieser Stelle abgewiesen. Somit wird verhindert, dass von den Programmiervorgaben im Laufe der Zeit immer weiter abgewichen wird.

Haben die Änderungen schließlich ihren Weg in das Entwickler-Repository gefunden, wird die QA-Pipeline angestoßen. Diese bildet die letzte Instanz der Qualitätssicherung. Die Änderungen durchlaufen dabei mehrere automatisierte „QA-Schleusen“. Ist die Pipeline vollständig durchlaufen, entsteht ein Gesamtabbild der Anwendung, das für den produktiven Einsatz bereit ist.

4 Entwicklungsumgebung

Abseits der stark kommerzialisierten Sprachen wie Java oder C# gibt es nur eine sehr eingeschränkte Auswahl an Werkzeugen. Mit *Eclipse* gibt es zwar für die meisten Sprachen eine passende IDE, jedoch sind viele Operationen nicht verfügbar, die zum Beispiel in der *Java-Eclipse* oder auch in *Visual Studio* selbstverständlich sind. Dazu gehören in der Regel Standardaufgaben wie automatisierte Refactorings, Unit-Tests oder Code-Formatierungen. Unter dem Gesichtspunkt der Nachhaltigkeit kann das ein Problem sein, da in großen Projekten die Produktivität darunter leidet.

Die Kollaborationsplattform *eCollab* ist in der Sprache PHP geschrieben und diese gehört zu einem Kreis, in dem die Tool-Landschaft eher eingeschränkt ist. Für *eCollab* ist *Eclipse* die IDE der Wahl, doch um die Anbindung an die Infrastruktur zu ermöglichen, muss sie erst noch durch Plugins erweitert werden. Dazu gehört zunächst die Unterstützung von *Git*-Repositorys mit dem Plugin *EGit*. Für PHP gibt es mit den *PHP Developer Tools* ein Paket mit zahlreichen Erweiterungen, die unter anderem das Ausführen von Unit-Tests und die Prüfung auf Programmiervorgaben erst ermöglichen. Alle Plugins müssen aber nach ihrer Installation einzeln konfiguriert werden, was ein nicht zu unterschätzender Aufwand ist. Um diesen zu vermeiden, wurde die gesamte Entwicklungsumgebung virtualisiert. Allen Entwicklern steht das gleiche System mit der gleichen Software und Konfiguration in einem sofort funktionstüchtigen Zustand zur Verfügung. Diese Einheitlichkeit erleichtert letztlich auch die Kommunikation und Zusammenarbeit im Team.

5 Private Build

Jeder Entwickler muss dafür sorgen, dass immer ein „grüner“ Build-Status vorhanden ist, sprich die Software erfolgreich kompiliert und getestet werden konnte. Dies kann dazu führen, dass als Vorsichtsmaßnahme eine Integration erst dann durchgeführt wird, wenn etwa ein Feature komplett fertiggestellt wurde. Das widerspricht allerdings dem Grundgedanken der Continuous Integration (CI), da dadurch die Gefahr erhöht wird, dass die eigenen Änderungen nicht mehr mit denen der anderen Entwickler harmonieren [DMG07]. Ein fehlerhafter Build ist im Grunde aber auch nichts Schlechtes, zeigt er doch, dass ein Fehler in der Software aufgedeckt wurde, bevor dieser im späteren Verlauf der Entwicklung Probleme verursachen kann.

Um dennoch frühestmöglich die Sicherheit zu erhalten, dass die eigenen Änderungen keine Probleme bereiten, gibt es den Private Build: Dieser erlaubt das Bauen und Testen der Software auf dem eigenen Entwicklungsrechner. Voraussetzung hierfür ist, dass alle Testskripte als Teil der Anwendung mitgeliefert werden und von jedem Entwickler ausgeführt werden können. Die Tests können somit auch ohne zentrale QA-Infrastruktur verwendet werden, was besonders für die schnelle Erkennung von selbst verursachten Fehlern nützlich ist.

Einige CI-Werkzeuge bieten deswegen auch die Möglichkeit, einen inoffiziellen Versuchs-Build durchzuführen [Wi11]. Dieser versteht sich als Private Build innerhalb der zentralen QA-Umgebung. Hier wird der gesamte Integrations-Vorgang simuliert und dem Entwickler später das Ergebnis mitgeteilt, ohne dass die anderen Projektbeteiligten etwas davon erfahren. Ginge das Ergebnis an alle Beteiligten, wäre der „Private“-Aspekt verloren: Experimentieren wäre nicht mehr möglich, da jeder Testlauf offiziell wird und dem Entwickler möglicherweise der „Gesichtsverlust“ droht.

Der Private Build wird nach dem Aktualisieren des eigenen Repositorys mit den Änderungen anderer Entwickler durchgeführt. Die Tests sorgen dafür, dass zum Beispiel Erweiterungen von öffentlichen Schnittstellen zu keinen unvorhergesehenen Fehlern führen. Treten keine Probleme auf, können die eigenen Änderungen in das von allen Entwicklern genutzte Repository eingestellt werden.

6 Repository-Workflow

In den letzten Jahren hat sich die Entwickler-Community weg bewegt von den klassischen Werkzeugen der Versionsverwaltung, vertreten durch *CVS* und *Subversion*, und wechselt nun zur neuen Generation: den verteilten Versionsverwaltungssytemen (DVCS) [Sp12, GDB11]. Prominentestes Werkzeug dieser Gattung ist das vom Schöpfer des Linux-Kernels entwickelte Open-Source-Werkzeug *Git* [Sp12].

Ein Unterschied zu den klassischen Tools ist, dass jeder Entwickler im Vergleich zu früher nicht mehr bloß eine „Arbeitskopie“ besitzt, sondern das gesamte Repository mit der kompletten Versionsgeschichte. Damit entfällt im Prinzip die Notwendigkeit eines zentralen Servers, mit dem alle Entwickler ihre Arbeitskopie synchronisieren müssen.

Das verteilte Modell zeichnet sich vor allem durch seine Flexibilität aus, denn es bietet viele verschiedene Einsatzmöglichkeiten; unter anderem kann damit auch das alte, zentrale Modell realisiert werden. In der Open-Source-Szene setzt sich zunehmend das Fork-und-Pull-Modell durch, welches vor allem durch den Hosting-Anbieter *GitHub* für viele greifbar wurde.

Software wird bei einem DVCS grundsätzlich „geforkt“, das heißt, die gesamten Quellen des Projektes inklusive seiner Versionsgeschichte werden kopiert und sind dadurch im Prinzip als eigenständiges Projekt zu betrachten. Software lebt aber von der Zusammenarbeit der Entwickler, deswegen wird durch einen „Pull-Request“ den ursprünglichen Entwicklern des Projektes mitgeteilt, dass in einem Fork diverse Änderungen vorliegen, die in das Ursprungsprojekt übernommen werden können. Es obliegt den Empfängern des Pull-Requests, diesen zu akzeptieren und die Änderungen zu übernehmen, Korrekturen zu fordern oder die Anfrage gänzlich abzuweisen. Typischerweise dient ein Pull-Request als Diskussionsplattform, in der alle Beteiligten die Änderungen diskutieren können, sollte es Klärungsbedarf geben.

Dieses Modell lässt sich für die nachhaltige Entwicklung abwandeln und mit dem klassischen zentralen Modell verbinden. Der Vorteil des Fork-und-Pull-Modells liegt zunächst in dem integrierten Code-Review: Bei Bedarf können sich die Entwickler einen Pull-Request zusenden, der dann von einer oder mehreren Personen abgenommen wird. Ansonsten kann die Entwicklung gemäß dem zentralen Modell erfolgen, indem alle Entwickler ihre Änderungen in ein gemeinsames Repository einfließen lassen. Hier besteht eine weitere Möglichkeit: Die Änderungen können zunächst in einen speziellen Branch oder in ein Zwischenrepository eingestellt werden, von wo sie erst nach einer automatisierten Qualitätssicherung in den Hauptzweig oder das Hauptrepository gelangen [GDB11] (Abb. 6).

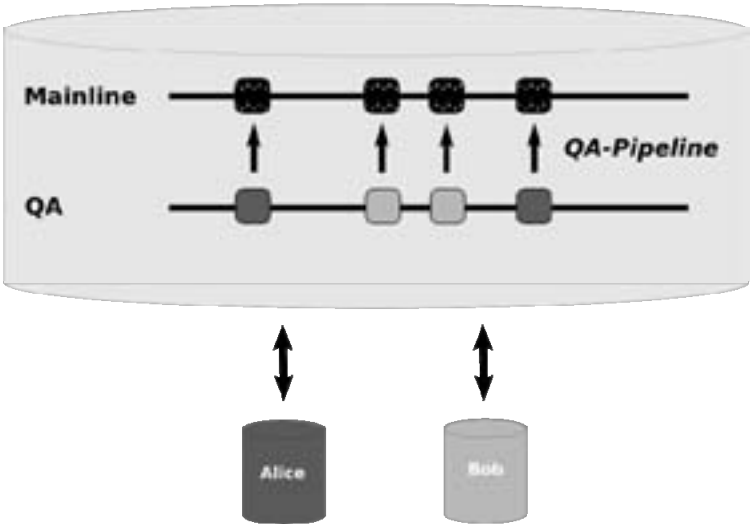


Abbildung 6: Entwickler arbeiten mit einem zentralen Repository, das sich in einen Entwicklungs- und automatisierten Produktionszweig aufteilt.

Dieses Verfahren bietet den Vorteil, dass jederzeit eine qualitätsgesicherte Version der Software vorliegt. Voraussetzung ist, dass der QA-Prozess automatisch ausgeführt wird, sobald eine neue Änderung eingestellt wurde, ohne dass die Entwickler hier manuell eingreifen müssen. Damit ist gewährleistet, dass der Prozess kontinuierlich für jede Änderung neu gestartet wird. Des Weiteren muss sichergestellt werden, dass erfolgreich getestete Änderungen automatisch in den Hauptzweig (*Mainline* in Abb. 6) eingebracht werden und diese somit allen Entwicklern als funktionstüchtige Basis zur Verfügung stehen.

Als Open-Source-Variante von *GitHub* steht *Gitorious* zur Verfügung, welches ebenfalls das Fork-und-Pull-Modell auf einfache Weise verfügbar macht. Entwickler-Teams können über diese Plattform ihre Repositories verwalten und Änderungen über Pull-Requests („Merge-Request" in der *Gitorious*-Terminologie) austauschen. Es steht des Weiteren immer auch der direkte Weg über die *Git*-Werkzeuge zum Austausch der Quellen zur Verfügung.

Es gilt zu beachten, dass die Grundprinzipien der Continuous Integration durch die verteilte Versionsverwaltung nicht untergraben werden: Änderungen sollen mindestens einmal täglich – spätestens am Ende des „Programmiertages" – in das zentrale Repository eingestellt werden, damit diese den übrigen Entwicklern zur Verfügung stehen. Änderungen, die nicht ihren Weg in das zentrale Repository finden, können nicht automatisiert getestet werden. Hier ist auch die sogenannte „Promiscuous Integration" zu vermeiden, also das Austauschen von Quellen untereinander – vorbei am zentralen Repository und damit der Qualitätssicherung [Fo09].

Hooks

Jedes Projekt definiert typischerweise einen „Wertekanon". Darunter sind zum einen die aus *Extreme Programming* bekannten, eher abstrakten Werte wie Mut, Respekt oder Einfachheit, zum anderen aber auch konkrete Elemente in Form von Programmierrichtlinien zu verstehen, zu denen sich alle Entwickler bekennen. Ziel solcher Richtlinien ist es, den Quelltext nicht nur einheitlich, sondern vor allem optimal lesbar zu gestalten. Des Weiteren sollen mögliche Probleme durch das Einfordern eines bestimmten Stils gänzlich ausgeschlossen werden. Am besten gelingt dies, wenn die Richtlinien lebendig, also direkt in den Entwicklungsablauf integriert sind. Dadurch wird die Einhaltung der Richtlinien zum Automatismus.

Alle gängigen Versionierungstools bieten Hooks an, womit die Abläufe innerhalb des Repositorys durch eigene Skripte angepasst werden können. Hooks sind dabei an Ereignisse geknüpft: So gibt es bei *Git* etwa den Commit-Hook. Hooks erscheinen immer in einer Pre- und einer Post-Variante. Erstere kann dazu genutzt werden, den Vorgang unter Umständen abzuweisen; bei der Post-Variante ist der Commit zu diesem Zeitpunkt bereits geschehen.

Ein Pre-Commit-Hook eignet sich dafür, die Änderungen innerhalb eines Commits auf Verstöße gegen die Programmiervorgaben zu überprüfen. Werden Verstöße festgestellt, kann der gesamte Commit deswegen abgebrochen werden. Dem Entwickler wird an-

schließend mitgeteilt, welche Unstimmigkeiten entdeckt worden sind, sodass die Probleme behoben werden können. Somit gelangt am Ende kein Quelltext in das Repository, der gegen die im Projekt definierten Vorgaben verstößt [Th11].

Weitere Ansatzpunkte für Skripte ergeben sich beim Empfang von Änderungen aus einem anderen Repository. *Git* nennt sie Receive-Hooks und sie eignen sich dafür, weitere Aktionen vor oder nach dem Empfang auszuführen. Sie können dazu genutzt werden, um bei soeben eingetroffenen Änderungen den Qualitätssicherungsprozess anzustoßen.

7 QA-Pipeline

Die automatisierte Qualitätssicherung nimmt einen zentralen Platz ein. Hier wird entschieden, ob Änderungen in den Hauptzweig übernommen werden oder nicht (Abb. 6). Die Entscheidung stützt sich dabei auf eine ganze Reihe von Tests, über deren Ausführung alle Entwickler Schritt für Schritt informiert werden.

Die Durchführung wird von der QA-Pipeline übernommen: Durch sie werden alle Aspekte der Anwendung überprüft und dadurch wichtiges Feedback gewonnen, das den Entwicklern übermittelt wird. Die QA-Pipeline, detailliert beschrieben von Humble und Farley, teilt sich dabei in mehrere Phasen auf, die jeweils für einen bestimmten Aspekt der Qualitätssicherung vorgesehen sind [HF10]. Sie beginnt mit Unit-, Integrations- und Akzeptanztests. Diese überprüfen die Funktionalität der Anwendung über mehrere Abstraktionsschichten hinweg – vom Code bis zum Feature aus Kundensicht. Darauf folgen Phasen für die nicht-funktionalen Anforderungen, etwa Performance oder Sicherheitsaspekte. Weitere Phasen sind für die Prüfung der Code-Qualität und das Testen eines möglichen Deployments zuständig (Abb. 7).

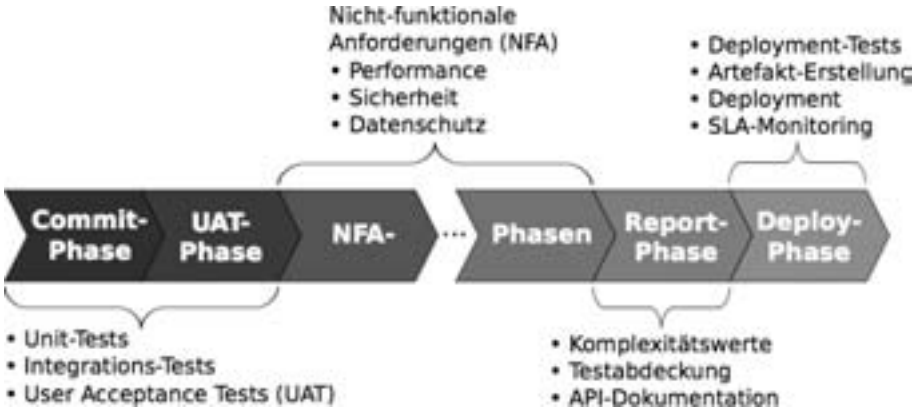


Abbildung 7: Schema einer QA-Pipeline [QT12]

Auf jeder Stufe erhalten alle Entwickler Rückmeldung, ob eine Änderung Fehler verursacht hat. Wird ein Fehler entdeckt, wird die gesamte Pipeline abgebrochen und die entsprechenden Phasen als fehlerhaft markiert. Die Probleme sollten anschließend so schnell wie möglich behoben werden, da wegen eines dauerhaften Fehlerzustands der gesamte Feedback-Aspekt verloren geht. Die Granularität der Fehlererkennung nimmt mit jeder Stufe ab, weswegen es wichtig ist, einen Fehler möglichst früh aufzudecken. Andernfalls kann die Fehlerquelle unter Umständen nicht genau ermittelt werden, wodurch sich die Fehlerbehebung erschwert.

Das Konzept „QA-Pipeline“ ist entstanden, damit die zuvor in der Regel getrennten Build- und Deployment-Abläufe zusammengeführt werden konnten. Es ist also das Bindeglied, durch das aus einer Code-Änderung die fertige, vollständig qualitätsgesicherte und produktionsreife Anwendung entsteht. Für die Kollaborationsplattform *eCollab* ist die Pipeline mit dem Continuous-Integration-Server *Jenkins* umgesetzt worden, da dieser dank zahlreicher Plugins ein äußerst flexibles Werkzeug darstellt.[5]

Welche Phasen letztendlich in der Pipeline definiert werden, hängt von den konkreten Anforderungen der Software ab. Hier können also bewusst Schwerpunkte auf ausgewählte QA-Aspekte gelegt werden. Die Pipeline verbietet nicht, dass etwa im nächtlichen Betrieb weitere Aspekte geprüft werden. Das können zum Beispiel länger laufende Performance-Tests sein. Dabei gilt es jedoch zu beachten, dass die zusätzlichen Tests nicht entscheidend für den Betrieb der Anwendung sein dürfen, da die Pipeline genau dies sicherstellen soll.

7.1 Commit und User Acceptance Tests

Diese Phasen bilden den Start der Pipeline. Zunächst wird auf Unit-Tests gesetzt, um die Grundfunktionalität der Anwendung auf Code-Ebene zu überprüfen. Die erste Phase muss so schnell wie möglich durchlaufen werden, damit den Entwickler ein zeitnahes Feedback zu seiner Änderung erreicht. Im weiteren Verlauf wird die Abstraktionsebene immer weiter erhöht: Integrations-Tests überprüfen das Zusammenspiel mehrerer Komponenten, bis schließlich mit den User Acceptance Tests die Features aus Sicht des Kunden überprüft werden [QT12]. Bei Letzterem werden die Use Cases automatisch „durchgeklickt“ und die Ergebnisse mit vordefinierten Erwartungswerten verglichen.

7.2 Nicht-funktionale Anforderungen

Weitere Phasen überprüfen beispielsweise die Performance, die Sicherheit oder auch den Datenschutz. Für Performance-Tests wird die Anwendung gezielt unter Last gesetzt, um Aufschlüsse über das Antwortzeitverhalten bei einem hohen Daten- oder Benutzeraufkommen zu gewinnen. Sicherheitstests werden mit Fuzzing-Werkzeugen ausgeführt, die diverse Angriffsvektoren überprüfen und im Prinzip versuchen, die Anwendung zum Absturz zu bringen. Durch Vulnerability Scans können Sicherheitslücken aufgedeckt werden. Wird bei Performance- oder Sicherheitstests ein bestimmtes Fehler-Level über-

[5] Siehe die Visualisierung der QA-Pipeline unter: https://scm.thm.de/pipeline

schritten, gelten die Phasen als gescheitert. Datenschutz-Aspekte können in Form eines *aktiven Verfahrensverzeichnisses* für jedes neue Software-Release veröffentlicht werden, indem innerhalb der Anwendung durch Oberflächentests überprüft wird, wer in welcher Benutzerrolle personenbezogene Daten anderer Benutzer sehen kann.[6] [QT12]

7.3 Reporting

Für die nachhaltige Software-Entwicklung ist es ratsam, diverse Qualitätsmetriken ständig im Auge zu behalten. Verschlechtern sich etwa die Komplexitätswerte in einem bestimmten Modul, müssen die Entwickler darüber informiert werden. Dies kann so weit gehen, dass die Pipeline abgebrochen wird, wenn ein gewisses Maß überschritten wird [Th11]. Nur so ergibt sich aus den Software-Metriken auch eine Konsequenz.

Am Beispiel der Metrik „Instability vs. Abstraction“ (Abb. 4) kann der QA-Verantwortliche des Projekts eine mittlere Distanz (D-Wert) aller Pakete von der Ideallinie festlegen, die nicht innerhalb einer vorgegebenen Toleranz überschritten werden darf. Je näher die Pakete der Ideallinie kommen, desto eher sind sie erweiterbar, wiederverwendbar und wartbar, desto höher also die Softwaregüte [Ma94]. Alternativ kann der QA-Verantwortliche festlegen, dass sich der mittlere D-Wert von Build zu Build nicht verschlechtern darf, ansonsten wird der aktuelle Build abgebrochen. In beiden Fällen erfährt der Entwickler, welche D-Werte sich durch den abgebrochenen Build verschlechtert hätten. Die Nachhaltigkeit der Softwaregüte kann so konsequent und automatisiert eingefordert werden.

7.4 Deployment

In der letzten Phase der Pipeline wird die endgültige Produktionsreife der Software festgestellt: In einer produktionsnahen Umgebung wird ein Test-Deployment durchgeführt, welches anschließend besonders unter Migrations- und Rollback-Gesichtspunkten überprüft wird. Des Weiteren kann das Test-Deployment als Schaufenster für den Kunden genutzt werden, der sich dadurch ständig ein Bild vom aktuellen Entwicklungsstand machen kann [HF10].

8 Zukünftige Arbeiten

Im Rahmen eines Forschungsprojekts zur Qualitätssicherung in globalen Open-Source-Entwicklungsprojekten soll die Nachhaltigkeit in der Software-Entwicklung mit weiteren Automatismen und Tools unterstützt werden. Unter anderem sollen die folgenden Herausforderungen gelöst werden:

- Nachhaltigkeits-Checks in der QA-Pipeline: Während Build-spezifische Qualitätsmetriken umfassend und detailliert ermittelt werden (Testabdeckung, Richtlinientreue, D-Wert, C.R.A.P.-Faktor [Sa07]), fehlt eine automatisierte Trend-Analyse zur Soft-

[6] Beispiel eines aktiven Verfahrensverzeichnisses: https://ecollab.thm.de/infos/datenschutz.php

waregüte auf der Basis aller Metriken aller Builds. Eine anhaltende Degenerierung einzelner Softwarekomponenten auf Paket-, Klassen- und Methodenebene soll erkannt werden und in konkrete Gegenmaßnahmen münden.

- Integration der Nachhaltigkeits-Checks in global verteilte Entwicklungsprojekte: Der gezeigte Prozess soll beispielhaft für die Weiterentwicklung der weltweit verbreiteten Open-Source-Lernplattform *Moodle* adaptiert werden. Dafür sind zunächst die technischen und strukturellen Voraussetzungen in Form eines Online-QA-Labors zu schaffen. Anschließend muss der QA-Prozess in die bestehenden Prozesse der *Moodle*-Community[7] integriert werden.

9 Zusammenfassung

Für eine nachhaltige Software-Entwicklung ist es wichtig, die Entwickler an zentralen Stellen im Entwicklungsprozess zu entlasten. Für Neulinge soll der Einstieg in das Projekt so einfach wie möglich gestaltet werden, sodass sie innerhalb kürzester Zeit an der Entwicklung teilnehmen können. Gleichzeitig soll sichergestellt werden, dass die Qualität der Software durch die Fluktuation der Entwickler nicht leidet.

Sich immer wiederholende Abläufe sind prädestiniert, automatisiert zu werden. Dazu zählen besonders die Integration der Code-Änderungen und die Qualitätssicherung. Entlang mehrerer Stufen wird den Entwicklern dabei Feedback geliefert, sodass sie im Problemfall sofort reagieren können.

Nachhaltige Software-Entwicklung fokussiert auf die Wahrung der bisher erreichten Softwaregüte trotz hoher Fluktuation der Entwickler. Der Beitrag hat gezeigt, dass mithilfe von Open-Source-Werkzeugen QA-Automatismen implementiert werden können, um eine schleichende Degenerierung der Softwaregüte von Build zu Build zu erkennen und die QA-Verantwortlichen auf den Plan zu rufen.

Danksagung

Die Autoren bedanken sich bei den Gutachtern für die konstruktive Kritik und bei Julian Hochstetter (M.Sc.) und Paul-Christian Volkmer (M.Sc.) für ihr Engagement und ihre Hilfe bei der Recherche und Evaluation der Open-Source-Tools sowie beim Aufbau und Betreiben der Source-Code-Management-Infrastruktur und QA-Pipeline.

[7] http://moodle.org/development/ (Abruf 28.07.2012)

Literaturverzeichnis

[DMG07] Duvall, P. M.; Matyas, S.; Glover, A.: Continuous Integration – Improving Software Quality and Reducing Risk. Addison-Wesley Professional, 2007.

[Fo09] Fowler, M.: FeatureBranch. 2009. http://martinfowler.com/bliki/FeatureBranch.html (Abruf 28.07.2012).

[GDB11] Geisler, M.; Digulla A.; Bucka-Lassen, K.: Mercurial – Schnelle und skalierbare Versionsverwaltung. In: Objekt Spektrum (Jan. 2011), S. 80–85.

[HF10] Humble, J.; Farley, D.: Continuous Delivery – Reliable Software Releases through Build, Test, and Deployment Automation. Addison-Wesley Professional, 2010.

[LM06] Lanza, M.; Marinescu, R.: Characterizing the Design. In: Object-Oriented Metrics in Practice. Springer, Berlin Heidelberg, 2006, S. 23–44.

[Ma94] Martin, R. C.: OO Design Quality Metrics – An Analysis of Dependencies. 1994. http://www.objectmentor.com/resources/articles/oodmetrc.pdf (Abruf 28.07.2012).

[QT12] Quibeldey-Cirkel, K.; Thelen, C.: Continuous Deployment. In: Informatik-Spektrum 35.4 (2012), S. 301–305.

[Sa07] Savoia, A.: Pardon My French, But This Code Is C.R.A.P. (2). 2007. http://www.artima.com/weblogs/viewpost.jsp?thread=210575 (Abruf 28.07.2012).

[Sp12] Spinellis, D.: Git. In: IEEE Software 29.3 (2012), S. 100–101.

[Th11] Thelen, C.: Qualitätssicherung von Software-Altsystemen durch automatisierte Verfahren. Master-Thesis. Technische Hochschule Mittelhessen, 2011.

[Wi11] Wiest, S.: Continuous Integration mit Hudson. dpunkt, Heidelberg, 2011.

Open-Source-Tools und Online-Dienste

Eclipse	http://www.eclipse.org
EGit	http://www.eclipse.org/egit
Git	http://git-scm.com
GitHub	https://github.com
Gitorious	http://gitorious.org
Jenkins	http://jenkins-ci.org
Ohloh	http://www.ohloh.net
PDT	http://www.eclipse.org/projects/project.php?id=tools.pdt
Redmine	http://www.redmine.org (Abruf 28.07.2012)

Nachhaltige Anwendungssysteme dank IT-Produktmanagement

Katharina Peine, Andreas Helferich, Sixten Schockert

Lehrstuhl für ABWL und Wirtschaftsinformatik II (Unternehmenssoftware)
Universität Stuttgart
Keplerstr. 17
70174 Stuttgart
{peine|helferich|schockert@wi.uni-stuttgart.de}

Abstract: Aufgrund des ständigen Wandels ihrer Umwelt stehen sowohl Anwenderunternehmen aus vielen Branchen als auch Softwareunternehmen verstärkt vor der Herausforderung, ihre Anwendungssystemlandschaft bzw. IT-Produkte den sich ändernden Bedürfnissen anzupassen und gleichzeitig wartbar zu halten. Zahlreiche Unternehmen und Behörden haben auf diese Herausforderung mit der Einführung der Funktion des IT-Produktmanagements reagiert. Dieses soll den nachhaltigen Betrieb der Anwendungssysteme durch die kontinuierliche Berücksichtigung marktlicher und technischer Anforderungen sicherstellen. Dieser Beitrag verfolgt das Ziel, die Rolle des IT-Produktmanagements in Form eines auf Basis einer theoretischen Analyse und einer Expertenbefragung abgeleiteten Aufgabenkatalogs zu beschreiben. Anwender- und Softwareunternehmen soll durch eine erste mögliche Kategorisierung von Typen des IT-Produktmanagers eine Orientierung im jeweiligen situativen Umfeld gegeben werden.

1 Einleitung

Unternehmen zahlreicher Branchen sehen sich einem Strom tiefgreifender Veränderungen ihrer Umwelt ausgesetzt, sei es auf Grund technologischer Innovation, legislativer Änderungen, globalen Wettbewerbs oder extremer Konjunkturschwankungen. Dies induziert einen großen Veränderungsdruck auf die interne IT, da nur wenige Unternehmen heutzutage ohne weitreichende IT-Unterstützung auskommen [HP09]. Als Reaktion auf diese Herausforderungen entstand in der Praxis die Funktion des IT-Produktmanagements, die primär durch die Rolle des IT-Produktmanagers wahrgenommen wird (in IT-Abteilungen auch häufig als IT-Koordinator bezeichnet [HJP06]).

Das Produktmanagement stellt das Produkt in den Mittelpunkt seiner Aktivitäten und ist verantwortlich für dessen gesamten Lebenszyklus von der Einführung bis zur Eliminierung. Somit ist das Produktmanagement mit den verschiedensten Bereichen in einem Unternehmen konfrontiert und verfolgt das Ziel, ein Produkt zu nachhaltigem Erfolg zu führen [KC09]. Ein Produkt ist dabei eine „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“ [KRS04]. Darauf aufbauend wird unter einem IT-Produkt ein Produkt verstanden, dessen vorrangiger Bestandteil Software ist.

Das IT-Produktmanagement hat sich sowohl in Softwareunternehmen (insb. im Bereich der Standardsoftware) als auch in Anwenderunternehmen etabliert. Bei Zweiteren ist das IT-Produktmanagement eine Schnittstellenfunktion zwischen den Fachabteilungen, der internen IT-Abteilung und den Anbietern der im Unternehmen genutzten IT-Produkte. Das IT-Produktmanagement soll durch ein gezieltes Management von IT-Produkten über deren ganzen Lebenszyklus hinweg sicherstellen, dass diese zum einen regelmäßig an die Umweltveränderungen angepasst werden und zum anderen die IT-Produkte trotz zahlreicher Änderungen wartbar bleiben und somit möglichst lange genutzt werden können. Demnach ist ein professionelles, kundenorientiertes IT-Produktmanagement ein wichtiger Erfolgsfaktor für nachhaltiges Wirtschaften der Unternehmen.

Während umfangreiche Literatur über die Aufgabenbereiche des Produktmanagements in unterschiedlichen Branchen existiert (z. B. [AH07, Ka07]), ist die spezielle Rolle des IT-Produktmanagements in Theorie und Praxis noch weitgehend unerforscht. Auch hat sich weder in der praxisorientierten noch in der wissenschaftlichen Literatur ein eindeutiges Rollenverständnis herausgebildet [BTK06]. Dem Argument, Produktmanagement-Konzepte aus anderen Branchen auf IT anzuwenden, steht die Erkenntnis entgegen, dass das IT-Produktmanagement besonders durch die Software selbst geprägt wird, deren Eigenschaften sich als so spezifisch erweisen, dass herkömmliche Konzepte nur bedingt übertragbar sind [KRS04]. Weiterhin beleuchten Konzepte aus der Betriebswirtschaftslehre häufig nur Teilbereiche wie z. B. das Marketing. Das Management von IT-Produkten agiert jedoch an der komplexen Schnittstelle zwischen technischen und betriebswirtschaftlichen Aspekten des Produktmanagements. [HP09].

Ziel dieses Beitrags ist es deswegen, die vielfältigen Aufgaben des IT-Produktmanagements auf Basis einer theoretischen Literaturanalyse sowie einer Expertenbefragung zu untersuchen. Daraus wird ein Aufgabenkatalog abgeleitet, der die Rolle des IT-Produktmanagements klar umreißt. Zudem wird ein erster Schritt der Identifikation von verschiedenen Typen der IT-Produktmanager gemacht, die sich als Reaktion auf die unterschiedlichen situativen Bedingungen in den Unternehmen etabliert haben.

2 Methodische Vorgehensweise

Um das Aufgabenspektrum des IT-Produktmanagements als Förderer der Nachhaltigkeit der betreuten IT-Produkte umfassend zu analysieren und einen ersten Schritt in Richtung Typenbildung zu gehen, wurde eine vierstufige Vorgehensweise gewählt (siehe Abbildung 1).
Zur Schaffung einer theoretischen Grundlage werden zunächst die Aufgaben des IT-Produktmanagements aus der praxisorientierten wie wissenschaftlichen Literatur abgeleitet. Die Literaturanalyse erfolgt dabei in Anlehnung an Richtlinien zur Durchführung von systematischen Literaturrecherchen [KC07, MNS11, VS^{+}09]. Die identifizierten Aufgaben werden Aufgabenbereichen zugeordnet und auf diese Weise übersichtlich in

Katalogform dargestellt (siehe Kapitel 3.1). Zur Berücksichtigung der praktischen Anwendungsseite wurden in den Jahren 2004 bis 2011 während insgesamt 28 Schulungen[1] im Themenumfeld des IT-Produktmanagements über 230 Teilnehmer aus Anwender- und Softwareunternehmen des deutschsprachigen Raums zu den Aufgaben des IT-Produktmanagements im jeweiligen Unternehmen befragt (siehe Kapitel 3.2). Im dritten Schritt werden der aus der Literatur abgeleitete Aufgabenkatalog und die sich aus den Antworten der Seminarteilnehmer ergebenden Aufgaben miteinander verglichen. Zur Verbindung der praktischen und theoretischen Perspektive wird dabei ein integrierter Aufgabenkatalog entwickelt (siehe Kapitel 3.3). Kapitel 4 zeigt dann, angelehnt an Mintzberg[2] [Mi83, Mi92], einen Ansatz zur Typenbildung von IT-Produktmanagern im jeweiligen situativen Kontext auf. Fazit und Ausblick in Kapitel 5 runden den Beitrag ab und zeigen weitere Forschungsmöglichkeiten auf.

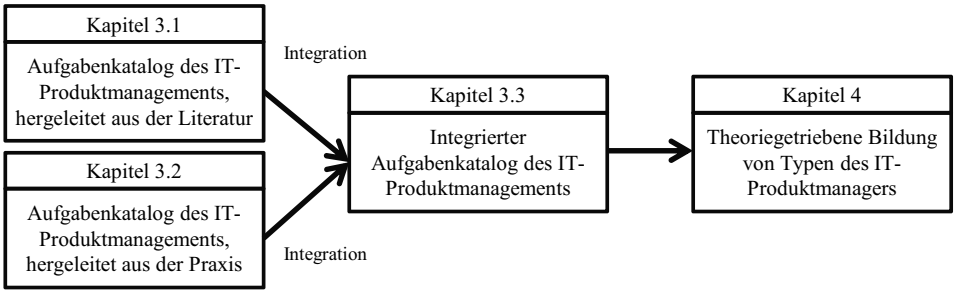


Abbildung 1: Methodischer Zugang

3 Herleitung eines integrierten Aufgabenkatalogs des IT-Produktmanagements

3.1 Aufgaben des IT-Produktmanagements in der Literatur

Bezüglich der Aufgaben des IT-Produktmanagements besteht in der Literatur keine Einigkeit. So äußern sich zahlreiche Autoren zu den Aufgaben des IT-Produktmanagements, ohne aber eine komplette Auflistung der Aufgaben zu liefern (siehe z. B. [Co02, Dv03, KC09, KRS04, SHT05]). Folglich war das erste Ziel unserer Forschung, einen Überblick über die verschiedenen Verständnisse der Aufgaben sowie eine einheitliche Terminologie zu schaffen. Die Literaturanalyse beruht dabei auf einer Recherche in neun elektronischen, frei zugänglichen Literaturdatenbanken der Wirtschaftswissenschaften und der Informatik/Informationstechnik.[3] Insgesamt wurden darin ca. 300 deutsch- und

[1] Die Schulungen wurden im Auftrag der Management Circle AG gehalten, die zu den renomiertesten Schulungsagenturen im deutschsprachigen Raum zählt. Teilnehmer waren Experten aus Industrie und öffentlicher Verwaltung mit mehrjähriger Berufserfahrung im Bereich IT-Produktmanagement.

[2] Das Standardmodell von Mintzberg gilt als „Klassiker der Organisationsforschung“ [NV07] und bildet bis heute eine Grundlage für zahlreiche Forschungsarbeiten, siehe z.B. [St04], S. 100 ff. oder [Ga12]

[3] Literaturrecherche erfolgte in folgenden Datenbanken: Association for Computing Machinery (ACM DL), CiteSeerX Library, ELSEVIER – Science Direct, Ebsco Host/Business Source Premier (BSP), Google Scholar, Institute of Electrical and Electronics Engineers (IEEE Xplore), ISI Web of Science, SpringerLink, FIZ Karls-

englischsprachige Quellen mit Bezug zu den Aufgaben des IT-Produktmanagements identifiziert (siehe Tabelle 1).

Aufgabenbereich	Aufgabe	Quellen
Entwicklung	Anforderungsmanagement	[SHT05] S. 2, [VB+06] S. 5
	Testmanagement	[Sa02] S. 110, [St05] S. 335
	Auftraggeberrolle für Entwicklung	[KRS04] S. 132
	Änderungswesen/Change Management	[He02] S. 181
	Releaseplanung	[VB+06] S. 5
	Softwarewartung	[SHT05] S. 3
	Konfigurationsmanagement	[HP09] S. 51, 59
	Produkt-Roadmapping	[VB+06] S. 4
Projektmanagement	Programm-/Portfoliomanagement	[VB+06] S. 3 f.
	Management von Projekten	[SHT05] S. 20, 53, [Ve02] S. 29
	Projektdurchführung/Administration	[HP09] S. 85, [So07] S. 4
Marketing	Zielmarktdefinition	[HP09] S. 52, [KC09] S. 96, [KRS04] S. 56-58
	Produktbeschreibung	[HP09] S. 52, [KC09] S. 96-97, [KRS04] S. 56
	Produktpositionierung	[HP09] S. 52-53, [KRS04] S. 56
	Marketing Controlling	[SHT05] S. 37-70
	Produktstrategie	[Eb06] S. 2, [Go06] S. 23, 71, [KRS04] S. 51-55
Vertrieb	Marketing Mix, 4 Ps (product, price, place, promotion)	[Dv03] S. 101-107, [HP09] S. 11, [KRS04] S. 44-51
	7 Ps (physical facilities, personnel, process-management)	[HP09] S. 53
	Sales	[KRS04] S. 67-74, [SHT05] S. 6
Technologie-management	Technologiemanagement	[Hu02] S. 83-97, [VB+06] S. 4
Organisation	Produktmanagementprozesse/ Prozessmanagement	[SHT05] S. 137-140
	Aufbauorganisation	[SHT05] S. 107, 130-135
	Wissensmanagement	[SHT05] S. 321-356
Professional Services	Service und Support	[HP09] S. 55, [HM00] S. 21, [KC09] S. 93-95
	Consulting	[KRS04] S. 74
	Stakeholdermanagement	[Co02] S. 50, [HP07] S. 40, [VBB10] S. 3, [Wi02] S. 1

Tabelle 1: Aufgabenkatalog des IT-Produktmanagements, hergeleitet aus der Literatur

Die Untersuchung dieser Quellen ergab ein sehr breites Aufgabenspektrum des IT-Produktmanagements. Um dieses übersichtlicher zu gestalten, wurden die Aufgaben mit Hilfe von Affinitätsdiagrammen inhaltlich zu Aufgabenbereichen strukturiert und gruppiert. Die interne Homogenität eines Bereiches besagt dabei, dass sich Elemente innerhalb einer Gruppe möglichst ähneln. Die externe Heterogenität zielt auf eine möglichst starke Unterscheidung der Bereiche untereinander ab [KK10, La05]. Insgesamt ergeben sich sieben Aufgabenbereiche mit 26 verschiedenen Aufgaben (siehe Tabelle 1). Auf-

ruhe (io-port.net) – Von einer speziellen Suche in Publikationen aus Konferenzen wurde abgesehen, da die Tagungsbände in aller Regel in den Datenbanken gelistet sind.

grund der Vielzahl der gefundenen Literatur werden im Aufgabenkatalog nur jeweilig einige der bedeutsamsten Quellen zu den Aufgaben aufgeführt (eine Vorversion des Katalogs wurde bereits veröffentlicht in [HP09]). Zudem sind die Aufgaben dabei allgemein gehalten, da sie von verschiedenen Unternehmen individuell ausgestaltet werden und je nach Situation sowie der organisationalen und fachlichen Aufhängung verschiedene Schwerpunkte einnehmen können (siehe dazu auch Kapitel 4).

3.2 Aufgaben des IT-Produktmanagements in der Praxis

Die Analyse der Aufgaben des IT-Produktmanagements aus Praxissicht hatte den Fokus, die wichtigsten Aufgabenfelder zu identifizieren. Aus diesem Grund war die Befragung der ca. 230 Seminarteilnehmer nicht als tiefgreifende Analyse angelegt, sondern es sollten kurz die Aufgabenschwerpunkte genannt werden. Die Ergebnisse haben somit explorativen Charakter und sagen weder etwas aus über Beziehungen der Aufgaben untereinander noch über die individuellen Umfelder der Organisationen, aus welchen die Befragten kommen (siehe Abbildung 2).

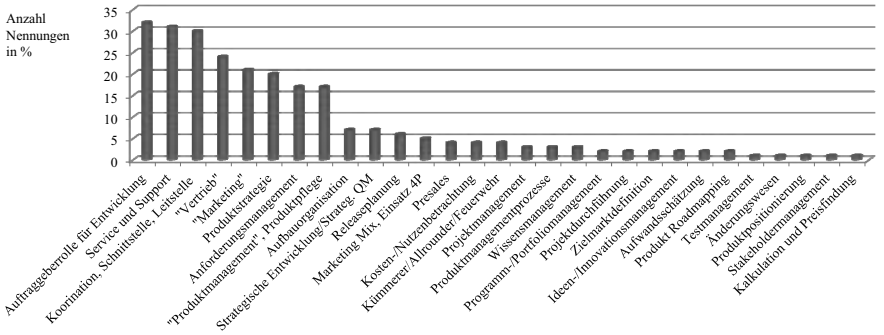


Abbildung 2: Ergebnisse der Expertenbefragung zu den Aufgaben des IT-Produktmanagements aus Schulungen im IT-Produktmanagement-Umfeld

Die Befragung erbrachte insgesamt 29 Aufgaben, die das IT-Produktmanagement in der derzeitigen Unternehmenspraxis wahrnimmt. Abbildung 2 zeigt prozentual die Antworten der Seminarteilnehmer auf die offen gehaltene Frage nach den Aufgabenschwerpunkten des IT-Produktmanagements in ihren Unternehmen, sortiert nach der Häufigkeit der Nennung. Dabei ergab sich bei den letzten Befragungen ein gewisser Sättigungsgrad, so dass durch eine weitere Iteration, bezogen sowohl auf die Aufgaben selbst wie auch die Häufigkeit der Nennung, keine weiteren Erkenntnisse zu erwarten sind. Auch wenn eine gewisse Verzerrung durch Vorselektion (die Teilnahme an den Seminaren war kostenpflichtig) nicht ausgeschlossen werden kann, so sollten die Antworten der Teilnehmer ein stimmiges Abbild der Aufgaben des IT-Produktmanagements in der Praxis geben.

Die befragten Experten nannten am häufigsten die Aufgaben Auftraggeberrolle für die Entwicklung, Service und Support, Koordination, Schnittstelle, Leitstelle, Vertrieb und Marketing. Dies unterstreicht die Rolle des IT-Produktmanagements als Koordinator aller produktbezogenen Maßnahmen zwischen Kunde und Entwicklung. Somit sorgt es

für die Verbesserung des produkt- und marktrelevanten Informationsflusses, der Kommunikation und Kooperation der einzelnen Stellen im Unternehmen und des Marktes. Oft werden auch Aufgaben des Vertriebs und Marketings erfüllt. Die strategische Ausrichtung des Produktes liegt häufig im Verantwortungsbereich, ebenso wie das Product Life Cycle Management. Ebenfalls von besonderer Bedeutung ist die Verwaltung der Kundenanforderungen an das Produkt (sowohl interner als auch externer Kunden).

3.3 Ableitung eines integrierten Aufgabenkatalogs des IT-Produktmanagements

Aufbauend auf dem aus der Literatur abgeleiteten Aufgabenkatalog des IT-Produktmanagements und den Aufgaben, die durch die Experten genannt wurden, wird im Folgenden ein integrierter Aufgabenkatalog synthetisiert (siehe Tabelle 2). Dieser aggregiert in einem ersten Schritt sämtliche identifizierten Aufgabenbereiche und Aufgaben. In der Spalte „Theorie“ mit einem x gekennzeichnet sind die Aufgaben(-bereiche), welche aus der Literatur hergeleitet wurden. Ein x in der Spalte „Praxis“ kennzeichnet die Aufgaben(-bereiche), welche von den Seminarteilnehmern genannt wurden. Die zehn identifizierten Aufgabenbereiche mit insgesamt 36 Aufgaben werden im Folgenden beschrieben und dabei wird versucht, Besonderheiten und Abweichungen zwischen der theoretischen und praktischen Perspektive zu erklären.

Der Aufgabenbereich *Entwicklung* ist derjenige, welcher von den Experten am häufigsten genannt wurde. Auffällig ist, dass das Konfigurationsmanagement und die Softwarewartung nicht genannt wurden. Hier lässt sich vermuten, dass diese in der Praxis eher operativ-technisch verstanden und daher direkt der Entwicklung bzw. dem Qualitätsmanagement zugeschlagen werden, während die eher strategisch-marktorientierten Aufgaben dem IT-Produktmanagement als „Stimme des und Schnittstelle zum Kunden“ zugeordnet werden. Der Bereich *Projektmanagement* birgt keine Überraschungen, Theorie- und Praxissicht stimmen überein. Obwohl im Bereich *Marketing* die in der Literatur genannten Aufgaben nicht im Detail durch die Aussagen aus der Praxis bestätigt wurden, wird dennoch klar, dass durch das IT-Produktmanagement sehr häufig Marketingaufgaben wahrgenommen werden („Marketing“, am fünfthäufigsten genannt). Basierend auf den Aussagen der Experten wird der Fokus auf strategisch ausgerichteten Aufgaben liegen. Ebenso wie Marketingaufgaben werden vom IT-Produktmanagement häufig *Vertrieb*saufgaben wahrgenommen („Vertrieb“, am vierthäufigsten genannt). Wie bei den Marketingaufgaben ist auch hier weiterer Forschungsbedarf zur Ausdifferenzierung der Aufgaben im Detail gegeben. Die 7Ps wurden durch die Seminarteilnehmer nicht genannt, eventuell wegen des geringen Bekanntheitsgrades. Dementgegen wurde das Presales in den Katalog aufgenommen, was den strategischen Charakter der wahrgenommenen Vertriebsaufgaben unterstreicht. Das Technologiemanagement, welches in der Literatur zu den Aufgaben des IT-Produktmanagements gezählt wurde, wurde von den Befragten unter diesem Namen nicht genannt. Jedoch ist der Aufgabenbereich *Innovationsmanagement* mit der Aufgabe Ideen-/Innovationsmanagement zum Katalog hinzugekommen. Das Technologiemanagement als Teil des Innovationsmanagements [SBA02] wurde in diesen Bereich integriert, der ebenfalls eher strategisch ausgerichtet ist. Der Aufgabenbereich *Organisation* entspricht in seinen Aufgaben denen aus der Literaturanalyse.

Aufgabenbereich	Aufgabe	Theorie	Praxis
Entwicklung		x	x
	Anforderungsmanagement	x	x
	Testmanagement	x	x
	Auftraggeberrolle für Entwicklung	x	x
	Änderungswesen/Change Management	x	x
	Releaseplanung	x	x
	Konfigurationsmanagement	x	0
	Softwarewartung	x	0
Projektmanagement		x	x
	Programm-/Portfoliomanagement	x	x
	Management von Projekten	x	x
	Projektdurchführung/Administration	x	x
Marketing		x	x
	Zielmarktdefinition	x	x
	Produktbeschreibung	x	0
	Produktpositionierung	x	x
	Marketing Controlling	x	0
	„Marketing“	0	x
Vertrieb		x	x
	Marketing Mix, 4 Ps (product, price, place, promotion)	x	x
	7 Ps (physical facilities, personnel, process-management)	x	0
	„Vertrieb“/Sales	x	x
	Presales	0	x
Innovationsmanagement		0	x
	Technologiemanagement	x	0
	Ideen-/Innovationsmanagement	0	x
Organisation		x	x
	Produktmanagementprozesse/Prozessmanagement	x	x
	Aufbauorganisation	x	x
	Wissensmanagement	x	x
Professional Services		x	x
	Service und Support	x	x
	Consulting	x	0
	Stakeholdermanagement	x	x
Steuerung der Wirtschaftlichkeit		0	x
	Kosten-Nutzenbetrachtung	0	x
	Kostenschätzung	0	x
	Kalkulation und Preisfindung	0	x
Strategieentscheidung		0	x
	Strategische Entwicklung, Strategisches Qualitätsmanagement	0	x
	Produkt-Roadmapping	x	x
	Produktstrategie	x	x
IT-Produktmanagement in der Rolle des multifunktionalen Generalisten		0	x
	Koordination, Schnittstelle, Leitstelle	0	x
	Produktmanagement, Product Life Cycle Management/Produktpflege und Kümmerer	0	x
	Allrounder/Feuerwehr/Mädchen für alles	0	x

Tabelle 2: Integrierter Aufgabenkatalog des IT-Produktmanagements

Service und Support im Bereich *Professional Services* wurden durch die Experten für die Aufgaben des IT-Produktmanagements am zweithäufigsten genannt. Das Consulting wurde nicht benannt, dafür aber das Stakeholdermanagement, was darauf schließen lässt, dass nicht der direkte Kontakt zum Kunden entscheidend ist, sondern eher die fachliche Unterstützung der Consultants/des Help Desks des jeweiligen Unternehmens. Auf Basis der Antworten der Experten wurde der Aufgabenbereich *Steuerung der Wirtschaftlichkeit* zusätzlich zu den aus der Literatur hergeleiteten Bereichen in den Katalog aufgenommen. Dieser beinhaltet die Aufgaben Kosten-/Nutzenbetrachtung, Aufwandsschätzung und Kalkulation und Preisfindung. Diese Aufgaben gehen in Richtung Produkt-Controlling, was wiederum die Steuerungs- und Koordinationsfunktion des IT-Produktmanagements unterstreicht. Die Ausrichtung des IT-Produktmanagements ist aus Praxissicht vermehrt eine strategische, weshalb der Aufgabenkatalog um den expliziten Bereich *Strategieentscheidung* erweitert wurde. Hier werden die Aufgaben der Entwicklung und Pflege der Produktstrategie (ursprünglich dem Marketing zugeordnet), das Produkt-Roadmapping (ursprünglich in der Entwicklung) und zusätzlich die strategische

Entwicklung/das strategische Qualitätsmanagement aufgenommen. Der letzte Bereich des Kataloges fasst Tätigkeiten zusammen, die die Besonderheit des IT-Produktmanagements in der *Rolle des multifunktionalen Generalisten* hervorheben, zu der die Koordination, Schnittstellen- und Leitstellenfunktion gehören sowie die Verantwortung für Produkte während des gesamten Life Cycles.

Einen ähnlichen Ansatz, die Aufgaben des IT-Produktmanagments zu strukturieren, stammt von der International Software Product Management Association[4] [GK11, KC09]. Zwar sind die Aufgaben hier teilweise auf einer anderen Ebene aggregiert, jedoch bestätigt dieser Ansatz die abgeleiteten Ergebnisse und Aussagen des hier aufgeführten integrierten Aufgabenkatalogs: Explizit ausgewiesen sind ebenfalls strategische Bereiche sowie die Bereiche Entwicklung, Marketing, Vertrieb und Service.

4 Theoriegetriebene Bildung von Typen des IT-Produktmanagers

Die Grundlage für die Typisierung des IT-Produktmanagements bildet die Stelle, welche als kleinste, selbständig handelnde Organisationseinheit gilt und mit Zuständigkeiten (Kompetenzen) ausgestattet ist, um einen definierten Aufgabenkomplex wahrnehmen zu können [BG10]. Demnach beschreibt die Stelle des (oder der) IT-Produktmanager(s) als organisatorische Einheit im Rahmen der Aufbauorganisation den Kompetenzbereich eines oder mehrerer gedachter Handlungsträger/Stelleninhaber und gründet sich auf einer Aufgabenanalyse und -synthese, wobei die Aufgaben des IT-Produktmanagements auf unterschiedliche Weise synthetisiert und wahrgenommen werden können. Nach Kosiol ist die Dimension, die sich auf die Art der Aufgabenerfüllung bezieht, die Ranganalyse, welche eine Aufteilung in Entscheidungsfindungs- und Ausführungsaufgaben vorsieht [Ko62]. Ein weiteres Kriterium für den Rang einer Aufgabe basiert auf der Führungshierarchie und kann nach folgender Klassifizierung von Stellen umgesetzt werden: Instanz mit Weisungsbefugnis (instance position), Ausführungsstelle (executive position), Stabsstelle (staff position) und Servicestelle (service position) [Ko62, Mi83]. Diese Abgrenzungen beschreiben die Art, wie das IT-Produktmanagement im jeweiligen Unternehmen eingebettet ist und spiegeln somit dessen situativen Kontext wieder:

Die Strategische Spitze (strategic apex) eines Unternehmens erfüllt Planungs-, Organisations- und Kontrollaufgaben und ist besetzt durch eine *Instanz*, also Managern mit Entscheidungs- sowie Weisungsbefugnis und Führungsautorität. Ist das IT-Produktmanagement als Instanz in einem Unternehmen verankert, gleicht es verschiedene Aufgaben einander an, fungiert als Informationsschnittstelle und delegiert Aufgaben. Eine *Ausführungsstelle* ist im Betrieblichen Kern (operating core) angesiedelt und mit der operativen Umsetzung von Aufgaben betraut. Dabei ist es wichtig, dass das Kongruenzprinzip [Ha69, Re82] erfüllt ist, d. h. dass das IT-Produktmanagement in dieser Situation sowohl die Kompetenz als auch die Verantwortung für eine Aufgabe innehat, die erfüllt werden soll. *Stabsstellen* liegen im Hilfsstab (support staff) außerhalb des Produktionsflusses. Auf diese Weise in die Unternehmensorganisation eingebunden assistiert das IT-Produktmanagement einer Instanz in der Entscheidungsvorbereitung und ist in einer

[4] Siehe http://ispma.org/spmbok/

Vorschlagsrolle. Eine *Servicestelle* ist in die Technostruktur (techno structure) außerhalb der formalen Linie eingebettet, definiert und überwacht Prozesse und passt die Unternehmensstruktur an die Umgebung an. Das IT-Produktmanagement als Servicestelle assistiert anderen Stellen in der Erbringung von Dienstleistungen, erfüllt Aufgaben unterstützend und bringt ein umfangreiches Fachwissen ein.

Im Folgenden wird ein Ansatz der situativen Einbettung und Typisierung des IT-Produktmanagements auf Basis der dargelegten unterschiedlichen Arten der organisationstheoretischen Aufgabenerfüllung entwickelt. In dieser Phase der Forschung ist es weder die Absicht noch möglich, eine vollständige und genaue Abgrenzung der Typen des IT-Produktmanagements zu geben. So werden sich die Typen in der Praxis teilweise überschneiden, ebenfalls wird ein Typ nicht unbedingt alle Aufgaben, die ihm im Folgenden zugeschrieben werden, wahrnehmen. Die vorgestellten Typen bilden im Sinne von Arbeitshypothesen eine Grundlage für weitere empirische Forschungsarbeit.

Typ 1 (siehe Tabelle 3) ist eine Ausführungsstelle im betrieblichen Kern des Unternehmens. Viele Aufgaben im Aufgabenbereich Entwicklung sind operative Aufgaben, allen voran das *Anforderungsmanagement*. Diese Aufgabe wurde sehr häufig von den Experten genannt und verursacht viele Probleme, da der Grad der Differenzierung von IT-Produkten oft sehr hoch ist. Da insbesondere Unternehmen im IT-Umfeld einem starken Wandel ausgesetzt sind und nachhaltige IT-Produkte durch ein kontinuierliches Anforderungsmanagement gewährleistet werden können, sollte Typ 1 seinen Fokus auf diese Aufgabe legen. Der Wandel hat auch Einfluss auf das *Testmanagement* und die *Wartung* der Legacy Systeme eines Unternehmens. *Change Management* wurde von den Praktikern nicht genannt, sollte jedoch in den Aufgabenbereich von Typ 1 aufgenommen werden, um diesen Herausforderungen zu begegnen. Eine Ausführungsaufgabe im Aufgabenbereich Projektmanagement ist die *Projektdurchführung/-administration*. Die Aufgaben des Marketing wurden nicht sehr oft durch die Praktiker benannt, jedoch steht die allgemein gehaltene Aussage *Marketing* (an fünfter Stelle) einen wichtigen Schwerpunkt für das IT-Produktmanagement dar, genau wie bei den operativen Vertriebsaktivitäten *Presales* und *Sales*. Diese Aussagen werden von den Experten oft genannt, jedoch, nicht zuletzt auf Grund der offenen Fragestellung, wenig spezifiziert, hier ist eine weitere Vertiefung notwendig. *Service* und *Support* werden, im Gegensatz zu Customer Consulting Aktivitäten, häufig durch das IT-Produktmanagement ausgeführt. Ein engerer Kontakt zum Kunden ist jedoch notwendig für eine gute Kommunikation und langlebige Produkte, daraus lässt sich schließen, dass *Consulting* und *Stakeholdermanagement* in den Geltungsbereich von Typ 1 aufgenommen werden sollten.

IT-Produktmanager Typ 1	
Aufgabenbereich	**Aufgabe**
Entwicklung	Anforderungsmanagement
	Testmanagement
	Änderungswesen/Change Management
	Softwarewartung
Projektmanagement	Projektdurchführung/Administration
Marketing	„Marketing“
Vertrieb	„Vertrieb“/Sales
	Presales
Professional Services	Service und Support
	Consulting
	Stakeholdermanagement

Tabelle 3: IT-Produktmanager Typ 1

Somit liegt der Fokus von Typ 1 nicht auf dem Produkt selbst, sondern auf der tatsächlichen Erfüllung der Aufgaben, die ihm von höheren Stellen aufgetragen werden, weshalb er manchmal sogar nicht als IT-Produktmanager wahrgenommen wird. Die kurzfristige Sicht prägt Typ 1, wobei er idealer Weise die Verantwortung tragen sollte für die Aufgaben, mit denen er betraut ist.

Typ 2 (siehe Tabelle 4) erfüllt eine Serviceposition. Die *Releaseplanung* wurde von den Praktikern genannt und bedarf einer mittelfristigen Sicht auf die Dinge, genau wie das *Konfigurationsmanagement*. Letzteres wurde nicht genannt, sollte jedoch erfüllt werden, um Probleme, die durch den Wandel und hohe Produktkomplexität entstehen, zu begegnen. Indem Typ 2 das *Management von Projekten* übernimmt, setzt er Typ 1 einen Rahmen, in dem dieser das Projekt dann durchführt. Aufgaben im Bereich Marketing sind die *Zielmarktdefinition* und *Produktpositionierung*. Obwohl diese Aufgaben von den Teilnehmern nicht explizit genannt wurden, ist es naheliegend, dass sie sich hinter der Aufgabe Marketing verbergen. Erfüllt Typ 2 diese Aufgabe, kann dem stark anwachsenden Wettbewerbsdruck eher begegnet werden. Booms und Bitner [BB81] fügen dem ursprünglichen *Marketingmix*-Modell für Industriegüter die Erweiterung zu den *7P*s hinzu für serviceorientierte Unternehmen, zu welchen softwarelastige Unternehmen zählen, weshalb Typ 2 auch diese Aktivitäten in sein Aufgabenfeld aufnehmen sollte. Das *Technologie-* und *Ideen/-Innovationsmanagement* sind ebenfalls wichtig, um Wandel und Wettbewerb standzuhalten und die Produkte innovativer zu gestalten. Problemen mit dem Produktportfolio, Änderungen, Grad der Differenzierung und Projekt- versus Produktgeschäft sollten durch die selten erwähnte Aufgabe *Produkt-Roadmapping* begegnet werden, um die Nutzung der Ressourcen und ihrer Einsatzgebiete zu planen und zu dokumentieren wie auch ihre Beziehung untereinander in bestimmten Perioden [VB$^+$06]. Dasselbe gilt für das *Product Life Cycle Management* bzw. *Produktpflege*, hier wird die Perspektive des gesamten Unternehmens auf das Produkt gelenkt.

IT-Produktmanager Typ 2	
Aufgabenbereich	**Aufgabe**
Entwicklung	Releaseplanung
	Konfigurationsmanagement
Projektmanagement	Management von Projekten
Marketing	Zielmarktdefinition
	Produktpositionierung
Vertrieb	Marketing Mix, 4 Ps (product, price, place, promotion)
	7 Ps (physical facilities, personnel, process- management)
Innovationsmanagement	Technologiemanagement
	Ideen-/Innovationsmanagement
Strategieentscheidung	Produkt-Roadmapping
IT-Produktmanagement in der Rolle des multifunktionalen Generalisten	Produktmanagement, Product Life Cycle Management/Produktpflege und Kümmerer

Tabelle 4: IT-Produktmanager Typ 2

Typ 2 ist auf das Produkt selbst fixiert und hat eine mittelfristige Sicht auf die Transaktionen, die damit verbunden sind. Er konzentriert sich auf die Produktpositionierung, kennt die Produktdefinition und weiß Bescheid über die strategische Ausrichtung der Produkte und des Unternehmens. Vor allem aber unterstützt er das Unternehmen mit seinen Fachkenntnissen und unterstützt andere Stellen durch Services.

Typ 3 (siehe Tabelle 5) ist als Stabsstelle im Unternehmen eingegliedert und führt Controlling- und Koordinationsaufgaben aus, unter anderem das *Marketing Controlling*, das *Management der IT-Produktmanagementprozesse* und das *Wissensmanagement*. Diese

Aufgaben sind wichtig, um die Langlebigkeit der IT-Produkte zu gewährleisten. Ein ausgereiftes *Prozessmanagement* unterstützt Unternehmen dabei, das Kongruenzprinzip und produktspezifisches Denken umzusetzen und das IT-Produktmanagement an das Umfeld anzupassen. Da Typ 3 durch seine Position als Drehscheibe im Unternehmen fungiert und einen guten Überblick über die Prozesse und Informationen, die mit dem IT-Produkt zusammenhängen, hat, eignet er sich besonders, um *Kostenschätzungen* durchzuführen. Diese übermittelt er an Typ 4 als Entscheidungsvorbereitung und um Kosten zu minimieren. Typ 3 kann somit als produktorientierte *Koordinations-, Schnitt-* und *Leitstelle* gesehen werden. Durch diese von den Seminarteilnehmern oft genannte Aufgabe (an dritter Stelle) kann vielen Problemen begegnet werden, z.B. Kommunikationsproblemen oder Schnittstellenproblemen aufgrund fehlender Vereinbarung zwischen den verschiedenen Interessengruppen, aber auch dem Wettbewerb zwischen beteiligten Abteilungen. Die hohe Produktkomplexität der Produkte führt zu weiteren Schnittstellenproblemen, die große Anzahl der Beteiligten macht die Sache nicht leichter. Gibt es jedoch eine dedizierte Position innerhalb des Unternehmens, die explizit für die *Koordination* zuständig ist, kann diesen Problemen entgegengewirkt werden. Einige der befragten IT-Produktmanager bezeichneten sich selbst als *Allrounder, Feuerwehr oder „Mädchen für alles“*, die in den unterschiedlichsten Situationen angesprochen werden und einspringen. In dieser Rolle des multifunktionalen Generalisten mit umfangreichem Aufgabenspektrum ist es schwierig, die Timeline einzuhalten. Hier kann eine klare Abgrenzung der Stelle und Definition der Aufgaben Abhilfe schaffen.

IT-Produktmanager Typ 3	
Aufgabenbereich	**Aufgabe**
Marketing	Marketing Controlling
Organisation	Produktmanagementprozesse/Prozessmanagement
	Wissensmanagement
Steuerung der Wirtschaftlichkeit	Kostenschätzung
IT-Produktmanagement in der Rolle des multifunktionalen Generalisten	Koordination, Schnittstelle, Leitstelle
	Allrounder/Feuerwehr/Mädchen für alles

Tabelle 5: IT-Produktmanager Typ 3

Typ 3 könnte als zentrale Kommunikationsplattform gesehen werden, die produktbezogene Entscheidungen koordiniert und kommuniziert. Er wirkt unterstützend und bereitet Grundlagen für übergeordnete Instanzen zur Entscheidungsfindung.

Typ 4 (siehe Tabelle 6) ist als Instanz in der *Auftraggeberrolle für die Entwicklung* (an erster Stelle genannt), eine Aufgabe mit Entscheidungsbefugnis. Das *Programmmanagement* ist ebenfalls eine Instruktionsaufgabe und hilft dabei, das Produktportfolio zu definieren und neue Produkte zu integrieren. Typ 4 ist verantwortlich für die *Produktbeschreibung*, um im Unternehmen eine einheitliche und allgemein gültige Differenzierung und Beschreibung der Produkte festzulegen. Weiterhin setzt er die *Aufbauorganisation* des IT-Produktmanagements im Unternehmen um, wodurch Kongruenzprinzip und produktspezifisches Denken unterstützt werden. Der im Management angesiedelte Typ 4 führt *Kosten-Nutzenbetrachtungen* durch und entscheidet über die Politik der *Kalkulation und Preisfindung*. Hier liegt der Fokus auf dem Produkt im unternehmerischen Kontext, die Kostenminimierung steht im Vordergrund und der starke sowie zunehmende Wettbewerb wird in Entscheidungen mit einbezogen. Weitere Aufgaben mit langfristiger Perspektive sind die *strategische Entwicklung* bzw. *Qualitätsmanagement* und auch die *Produktstrategie*, durch welche den Schwierigkeiten bei der Definition des Produktport-

folios und den Änderungen im Unternehmens- und Produktumfeld besser begegnet werden und die Nachhaltigkeit der IT-Produkte unterstützt werden kann.

IT-Produktmanager Typ 4	
Aufgabenbereich	**Aufgabe**
Entwicklung	Auftraggeberrolle für Entwicklung
Projektmanagement	Programm-/Portfoliomanagement
Marketing	Produktbeschreibung
Organisation	Aufbauorganisation
Steuerung der Wirtschaftlichkeit	Kosten-Nutzenbetrachtung
	Kalkulation und Preisfindung
Strategieentscheidung	Strategische Entwicklung, Strategisches Qualitätsmanagement
	Produktstrategie

Tabelle 6: IT-Produktmanager Typ 4

Typ 4 ist demnach fokussiert auf die Geschäftsentwicklung und hat eine langfristige Sicht auf die Dinge. Er erfüllt fast ausschließlich strategische Aufgaben, ist verantwortlich für Finanzen und das IT-Alignment, so dass die Ziele des Unternehmens mit der Produktstrategie und dem IT-Produktmanagement selbst korrelieren. Hierbei wird er von Typ 3 unterstützt.

5 Zusammenfassung und Ausblick

Ziel dieses Beitrags war es, die Rolle des IT-Produktmanagements intensiv zu analysieren sowie erste Hypothesen bzgl. Typen des IT-Produktmanagements zu generieren. Hierzu wurde sowohl eine detaillierte Literaturanalyse als auch eine kurze Befragung von Teilnehmern an Seminaren zum IT-Produktmanagement durchgeführt. Auf dieser Basis wurde ein integrierter Aufgabenkatalog entwickelt, der Anwender- und Softwareunternehmen zusammen mit der Bildung theoriegeleiteter Typen des IT-Produktmanagements eine Orientierung zur Ausgestaltung des IT-Produktmanagements geben kann.

Dabei wurde deutlich, dass eine Vielzahl der Aufgaben des IT-Produktmanagements die Langlebigkeit von Anwendungssystemen unterstützt. Dementsprechend leisten alle Typen von IT-Produktmanagern ihren Beitrag zur Nachhaltigkeit: Typ 1 vor allem durch das Management der Anforderungen, um deren Verständlichkeit, Eindeutigkeit, Nachweisbarkeit, Widerspruchsfreiheit, Vollständigkeit und Testbarkeit zu gewährleisten. Typ 2 setzt einen Rahmen, um das Produkt selbst nachhaltig zu konfigurieren. Typ 3 in der Rolle des multifunktionalen Generalisten stimmt koordinierend soziale, wirtschaftliche und technische Vorgänge aufeinander ab und trägt so zum nachhaltigen Wirtschaften bei, ebenfalls wie Typ 4 durch seine strategische, langfristige Sicht. Der aus Theorie und Praxis hergeleitete Aufgabenkatalog des IT-Produktmanagements und die Typenbildung können bereits als Leitfaden für die Ausgestaltung des IT-Produktmanagements in der Praxis dienen, wobei hier jedoch die individuelle Situation und Ausgestaltung des jeweiligen Unternehmens nicht berücksichtigt ist. Die in diesem Beitrag zugrundegelegte Empirie ist eine gute Basis zur Bildung von Typen, jedoch bei weitem nicht ausreichend. Zur Erreichung des zukünftigen Ziels, ein theoretisch fundiertes, situatives Modell des IT-Produktmanagements mit Idealtypen zu kreieren, welches zusätzlich Gestaltungsempfehlungen für die Ausgestaltung des IT-Produktmanagements in der Praxis geben kann, ist weitere Forschungsarbeit notwendig. In einem nächsten Schritt soll durch vertiefende

Experteninterviews die Stelle des IT-Produktmanagers untersucht werden. Es ist zu vermuten, dass sich die unterschiedlichen Typen von IT-Produktmanagern nicht nur von ihren Aufgaben ableiten werden, sondern auch von weiteren Rahmenbedingungen. Daher sollen durch fundierte empirische Arbeit zudem Ziele, Gestaltungsalternativen, hemmende/fördernde Rahmenbedingungen, Verbesserungsmöglichkeiten und gemeinsame Merkmale identifiziert werden. Eine Basis hierfür wurde bereits geschaffen durch eine Studie von Bekkers u.a., welche den Einfluss von situativen Faktoren auf IT-Produktmanagement herausstreicht [BV+08]. Für die Typenbildung soll das Stufen-Modell der Typisierung von Kelle und Kluge [KK10] auf das IT-Produktmanagement angepasst und angewandt werden. Aufbauend auf der Typisierung können anschließend quantitativ empirische Arbeiten durchgeführt werden, um z. B. Häufigkeitsverteilungen der verschiedenen Typen in unterschiedlichen Branchen oder Regionen aufzudecken.

Literaturverzeichnis

[AH07] Albers, S.; Herrmann, A.: Handbuch Produktmanagement: Strategieentwicklung - Produktplanung - Organisation - Kontrolle. 3. Aufl., Gabler, Wiesbaden, 2007.

[BB81] Booms, B.H.; Bitner, M.J.: Marketing strategies and organization structures for service firms. In (Donnelly, J.H.; George, W.R., Hrsg.): Marketing of Services. American Marketing Association, Chicago, 1981; S. 47-51.

[BG10] Bea, F. X.; Göbel, E.: Organisation. 3. Aufl., UTB, Stuttgart, 2010.

[BTK06] Böhmann, T.; Taurel, W.; Krcmar, H: Produktmanagement für IT-Dienstleistungen in Deutschland. Arbeitspapier Nr. 23, Garching, 2006.

[BV+08] Bekkers, W.; Van de Weerd, I.; Brinkkemper, S.; Mahieu, A.: The Influence of Situational Factors in Software Product Management: An Empirical Study. 2nd Int. Workshop on Software Product Management (IWSPM '08), Barcelona, 2008; S. 41-48.

[Co02] Condon, D.: Software Product Management. Aspatore Books, USA, 2002.

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

[Eb06] Ebert, C.: The impacts of software product management. http://itmpi.org/assets/base/images/itmpi/Ebert-Impacts.pdf, 2006. Zugriff: 01.08.2012.

[Ga12] Gabler Verlag: Gabler Wirtschaftslexikon, Stichwort: Organisatorische Einheit. http://wirtschaftslexikon.gabler.de/Archiv/2504/organisatorische-einheit-v8.html. Zugriff: 15.08.2012.

[GK11] Gorschek, T.; Kittlaus, H.-B.: Towards a Software Product Management Certification. 5th Int. Workshop on Software Product Management (IWSPM '11), Trento, 2011; S. 1-2.

[Go06] Gorchels, L.: The Product Manager's Handbook. Mcgraw-Hill, New York, 2006.

[Ha69] Hauschildt, J.: Initiative. In (Grochla, E., Hrsg.): Handwörterbuch der Organisation. Poeschel, Stuttgart, 1969; S. 734-741.

[He02] Heinold, R.: Änderungen im Griff. In (Versteegen, G., Hrsg.): Software-Management. Springer, Berlin u.a., 2002; S. 181-202.

[HJP06] Herzwurm, G.; Jesse, S.; Pietsch, W.: Der IT-Koordinator. In: Das Wirtschaftsstudium 2, 2006; S. 184-186.

[HM00] Hofmann, M.; Mertiens, M.: Customer-Lifetime-Value-Management. Gabler, 2000.

[HP07] Herzwurm, G.; Pietsch, W.: Der IT-Manager. Schriftlicher Lehrgang in 12 Lektionen, Management Circle Verlag GmbH, Eschborn, 2007.

[HP09] Herzwurm, G.; Pietsch, W.: Management von IT-Produkten. dpunkt, Heidelberg, 2009.

[Hu02] Hubert, R.: Plattformunabhängige Softwareentwicklung. In (Versteegen, G., Hrsg.): Software-Management. Springer, Berlin u.a., 2002; S. 83-97.

[Ka07] Kairies, P.: Produkt-Management für die Investitionsgüterindustrie – Praxis und moderne Arbeitstechniken. 8. Aufl., expert, Renningen, 2007.
[KC07] Kitchenham, B.; Charters, S.: Guidelines for performing Systematic Literature Reviews in Software Engineering. Keele and Durham Univ., EBSE Technical Joint Report, 2007.
[KC09] Kittlaus, H.-B.; Clough, P.: Software Product Management and Pricing. Springer, Berlin und Heidelberg, 2009.
[KK10] Kelle, U.; Kluge, S.: Vom Einzelfall zum Typus: Fallvergleich und Fallkontrastierung in der qualitativen Sozialforschung. VS Verlag, Wiesbaden, 2010.
[KRS04] Kittlaus, HB.; Rau, C.; Schulz, J.: Software-Produkt-Management. Springer, 2004.
[Ko62] Kosiol, E.: Organisation der Unternehmung. Gabler, Wiesbaden, 1962.
[La05] Lamnek, S.: Qualitative Sozialforschung. 4. Aufl., Beltz Psychologie Verlags Union, Weinheim, 2005.
[Li74] Linnert, P.: Produkt-Manager. Deutscher Betriebswirte-Verlag, Gernsbach, 1974.
[Mi83] Mintzberg, H.: Structure in Fives. Englewood Cliffs, New Jersey, Prentice-Hall, 1983.
[Mi92] Mintzberg, H.: Die Mintzberg-Struktur: Organisationen effektiver gestalten. Landsberg/Lech 1992.
[MNS11] Maglyas, A.; Nikula, U.; Smolander, K.: What Do We Know about Software Product Management?. 5th Int. Workshop on Software Product Management (IWSPM'11), Trento, 2011; S. 26-35.
[NV07] Nicolai, A.; Vollmar, B.H.: Zwischen Sein und Sollen: Henry Mintzbergs Beitrag für die Managementwissenschaften. In: Organisationsentwicklung, 4, 2007, 5; S. 86-91.
[Re82] Reiß, M.: Das Kongruenzprinzip der Organisation. In: Wirtschaftswissenschaftliches Studium (WiSt), 11, 1982, 2; S. 75-78.
[Sa02] Salomon, K.: Das ständige Testen nicht vergessen. In (Versteegen, G., Hrsg.): Software-Management. Springer, Berlin, 2002.
[SBA02] Specht, G.; Beckmann, C.; Amelingmeyer, J.: F&E-Management - Kompetenz im Innovationsmanagement. 2. Aufl., Schäffer-Poeschel, Stuttgart, 2002.
[SHT05] Sneed, H.; Hasitschka, M.; Teichmann, M.: Software Produktmanagement – Wartung und Weiterentwicklung bestehender Anwendungssysteme. dpunkt.verlag, Heidelberg, 2005.
[So07] Sommerville, I.: Software Engineering. Pearson Studium, USA, 2007.
[St04] Stadler, C.: Unternehmenskultur bei Royal Dutch/Shell, Siemens und DaimlerChrysler. Franz Steiner Verlag, Stuttgart, 2004.
[St05] Steinweg, C.: Management der Software-Entwicklung – Projektkompass für die Erstellung von leistungsfähigen IT-Systemen. Vieweg, Wiesbaden, 2005.
[VBB10] Van de Weerd, I.; Bekkers, W.; Brinkkemper, S.: Developing a Maturity Matrix for Software Product Management. In (Tyrväinen, P.; Jansen, S.; Cusumano, MA., Hrsg.): Proceedings, Software Business - 1st Int'l Conf., ICSOB 2010, Jyväskylä, 2010.
[VB+06] Van de Weerd, I.; Brinkkemper, S.; Nieuwenhuis, R.; Versendaal, J.; Bijlsma, L.: On the Creation of a Reference Framework for Software Product Management. 1st Int. Workshop on Software Product Management (IWSPM '06), Minneapolis, 2006; S. 3-12.
[Ve02] Versteegen, G.: Software-Management. Springer, Berlin et al, 2002.
[VS+09] Vom Brocke, J.; Simons, A.; Niehaves, B.; Riemer, K.; Plattfaut, R.; Cleven, A.: Reconstructing the giant: on the importance of rigour in documenting the literature search process. In: 17th European Conf. on Information Systems (2009), 3, 2009, 15; S. 1-13.
[Wi02] Windley, P.: The Discipline of Product Management. http://www.windley.com/docs/Product%20Management.pdf, 2002. Zugriff: 01.05.2012.

Nachhaltigkeit durch gesteuerte Software-Evolution

Harry M. Sneed

ANECON GmbH, Wien,
Universität Regensburg, Bayern
Harry.Sneed@T-Online.de

Abstract: In diesem Beitrag zur Förderung der Softwarenachhaltigkeit wird der herkömmliche Begriff des Projektes in Frage gestellt. Es komme weniger darauf an, Softwareprojekte auszuführen als vielmehr, Softwareprodukte zu bauen und über die Zeit ständig auszubauen und nachzubessern. Ein Softwareprodukt ist nie fertig bzw. „Done", es wird so lange weiterentwickelt, bis keiner mehr daran Interesse hat. Ergo kann ein Projekt niemals abgeschlossen sein. Der Beitrag beschreibt, wie Softwareprodukte entstehen und über viele aufeinander folgende Releases immer reifer und nützlicher werden. Der Anwender arbeitet stets mit einem vorübergehenden Zustand, nie mit einem endgültigen. Da die IT-Welt immer im Wandel begriffen ist, kann es keinen Endzustand geben. Dieser Ansatz wird von einer Service-orientierten Architektur sowie von der aufsteigenden Cloud-Technologie geradezu gefördert. In einer betrieblichen SOA werden die angebotenen Services fortlaufend erneuert. Da diese unfertigen Teile in die eigene Anwendungssoftware eingebaut sind, kann auch die eigene Software nie fertig werden. Sie hat immer nur einen Stand erreicht. Dies gilt umso mehr für Services aus der Cloud, wo der Anwender noch weniger Einfluss auf die Weiterentwicklung seiner Komponente hat. Das ändert wie unsere Systeme geplant und finanziert werden. Die Planung und Kostenkalkulation beschränkt sich auf ein Release, bzw. auf einen Zeitrahmen von maximal drei Monaten. Das Budget für ein Produkt muss offen bleiben und nach jedem Release neu aufgestellt werden. Die IT ist in einer dynamischen Umwelt eingebettet und muss sich dieser anpassen. Dennoch darf diese Anpassung nicht in Chaos ausarten. Die Antwort ist eine gesteuerte Evolution im Einklang mit den wandelnden Anforderungen und den neuesten Serviceangeboten.

Keywords: Projektmanagement, Produktmanagement, Cloud-Computing, Cloud-Services, SOA, Change-Management, Softwarewartung, Softwareevolution

1 Das Wesen von IT-Projekten

IT-Projekte sind zeitlich begrenzte, einmalige Anstrengungen, um ein vorgegebenes Ziel bzw. Ergebnis zu erreichen. Wichtig sind hier die zeitliche Begrenzung und die Zielorientierung. Projekte steuern auf ein bestimmtes Ziel hin und haben dafür nur begrenzt Zeit und Geld. Meistens ist die Zeit wichtiger als das Geld, weil der Anwender damit rechnet, mit dem Ergebnis des Projektes ab einem bestimmten Zeitpunkt arbeiten zu können. Ist das Ziel einmal erreicht, ist das Projekt zu Ende bzw. „Done" [Mira02]. Die

Entwicklungsmannschaft wird aufgelöst und die daraus resultierende Software geht in die sogenannte Wartung.

Diese Denkweise ist verheerend, was die Nachhaltigkeit der Software anbetrifft. Um den Termin zu halten, werden sämtliche anderen Ziele wie Performanz, Sicherheit, Wartbarkeit und Ausbaufähigkeit geopfert. Alles was nicht sofort erkennbar ist, wird unter den Teppich geschoben. Der Anwender wird damit getröstet, dass alles später folgen kann. Dass dies aber nicht möglich ist, weil die Weichen dafür gar nicht gestellt sind, kann der naive Anwender nicht erkennen. Abgesehen davon ist das Personal, das diese Nachbesserung eventuell noch durchführen könnte, nicht mehr verfügbar. Kurzum, der Anwender wird betrogen, damit er glaubt, dass die Software wirklich fertig und das Projekt damit abgeschlossen ist.

Projekte beanspruchen Ressourcen bzw. Betriebsmittel, um zum gewünschten Ergebnis zu gelangen. Das Ausmaß der Ressourcen bestimmt die Größenordnung des Projektes. In IT-Projekten sind die Ressourcen von dreierlei Art:

- Hardware,
- Software,
- Personal.

Die Personalressourcen sind maßgeblich. Die Hardwarekapazität wie auch die Höhe der Softwarenutzungsgebühr hängt von der Anzahl der Köpfe ab. Somit gehen die Projektkosten aus der Anzahl der beteiligten Personen mal die Projektdauer hervor. Da der Anwender bemüht ist, seine Kosten so niedrig wie möglich zu halten, ist er auch bemüht, die Projektlaufzeit so kurz wie möglich zu halten. Der zu geringe Aufwand geht auf Kosten der Nachhaltigkeit. Statt die Software ordentlich zu konstruieren und die Qualität sorgfältig zu prüfen, wird sie in kurzen Sprints ohne unabhängige Qualitätssicherung „zusammengehauen“ und als eine benutzergerechte Lösung verkauft. Der Anwender, als Produkt Owner, sieht die Software nur von außen her über die Benutzeroberfläche. Wie es innen hinter der Oberfläche aussieht, weiß er nicht. Er bekommt es nur später zu spüren, wenn er sie ändern und erweitern möchte. Die Rechnung für die kurze Entwicklungszeit und die geringen Kosten folgt später und sie überwiegt bei Weitem das, was man durch die schnelle Entwicklung eingespart hat. Nicht nur das; die Masse an schlechtem, schwer handhabbarem Code nimmt immer mehr zu. Bald ertrinkt der Anwender in einer Flut von redundanten Codezeilen, aus dem er nicht wieder herauskommt – Code der eigentlich nie hätte geschrieben werden sollen, wenn er nur die Zeit genommen hätte zu prüfen, ob der Code nicht schon existiert.

Das Problem mit IT-Systemen ist, dass solche einmaligen, zeitlich begrenzten Anstrengungen eines Teams besessener Entwickler nicht ausreichen, um ein befriedigendes Ergebnis zu erzielen. Denn während sie darauf hin arbeiten, verschiebt sich das Ziel. Schon allein das Erreichen des Ziels verändert die Bedingungen, unter denen das Ziel angestrebt wurde (siehe das Heisenberg Prinzip). Deshalb gibt es kaum IT-Projekte, mit deren Ergebnissen die angeblichen Nutznießer am Ende wirklich zufrieden sind, auch nicht mit denen der agilen Entwicklungsprojekte [ZhPa11]. Die betriebswirtschaftlichen und

technischen Ausgangsbedingungen ändern sich zu schnell. Kaum ist die erste Version ausgeliefert, da schielen die Anwender schon auf die nächste. Die Ausgangsbedingungen haben sich geändert. Daher ist der Begriff „Projekt“ im herkömmlichen Sinne in Bezug auf IT-Systeme eher irreführend. Er verleitet zu der Annahme, es gebe so etwas wie eine endgültige Lösung, wo in Wirklichkeit nur Zwischenlösungen erzielt werden können [Howa01].

Ein geeigneterer Begriff wäre der Begriff „Produktevolution“. Ein Softwareprodukt durchzieht mehrere Evolutionsphasen. Es beginnt mit der Konzeption und Prototypbildung. Danach folgen mehrere Releases, mit denen der Anwender schon arbeiten kann. Bennett und Rajlich unterscheiden in ihrem Evolutionsmodell zwischen der Entwicklungsphase und der Evolutionsphase, aber diese Unterscheidung ist künstlich. Eigentlich ist alles nach dem Bau des ersten Prototyps eine Evolution bzw. eine permanente Weiterentwicklung bis hin zur Ablösung des Produktes. Laut dem Modell von Bennett und Rajlich folgt nach einigen Jahren Evolution eine ausgesprochene Erhaltungsphase (Maintenance Phase) bei der nur noch Restfehler korrigiert und kleine Änderungen vorgenommen werden [BeRa00] (siehe Abbildung 1).

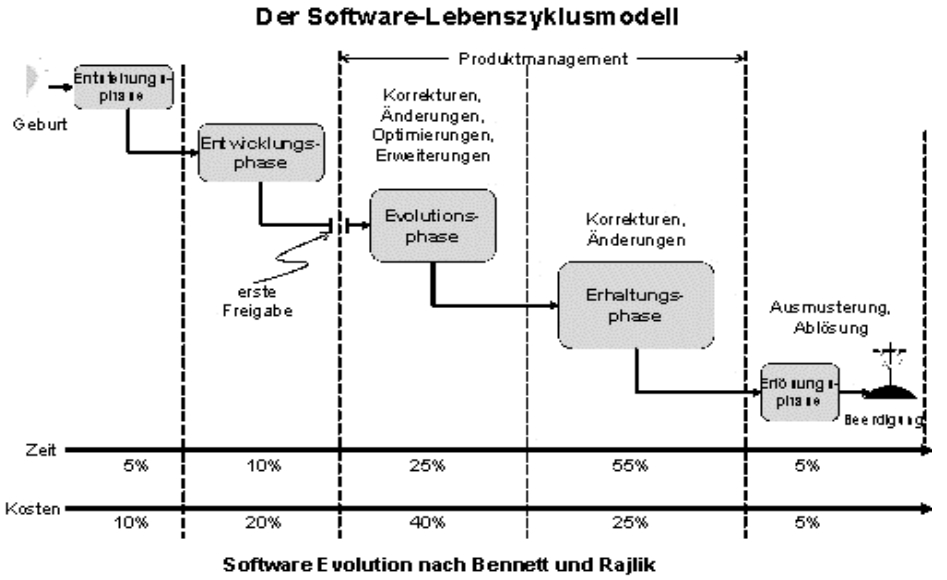


Abbildung 3: Modell der Software-Evolution

Das fünfstufige Phasenmodell trifft in Anbetracht der Dynamik einer Service-orientierten Softwarewelt nicht mehr zu. Ein modernes Service-basiertes System muss bis zu seinem Lebensende ständig weiter entwickelt werden. Am Ende bleiben nur drei Phasen übrig:

1. die Entstehungsphase, bzw. die Prototypentwicklung,

2. die Evolutionsphase, bzw. die Weiterentwicklung und

3. die Erlösungsphase, bzw. die Ausmusterung des Systems.

Wir müssen uns also von dem Begriff „Projekt“ befreien, ebenso wie die Deutschen sich von jeder Menge anderer alter Begriffe aus der NS-Zeit mit einem schlechten Beigeschmack befreit haben. Der Begriff „Projekt“ ist zu sehr mit den Begriffen „Zeit“ und „Aufwand“ assoziiert. Ein Evolutionsvorhaben zum Bau eines Softwareproduktes ist weder Zeit- noch Kostenverbunden. Es findet eben statt und dauert so lange wie das Produkt noch gebraucht wird und kostet so viel wie der Benutzer bereit zu bezahlen ist. Wenn er merkt, dass die Kosten ausufern kann er das Tempo der Evolution verlangsamen, die Mannschaft reduzieren oder das Vorhaben insgesamt aufgeben. Das Ziel des Projektes ist also nicht, ein bestimmtes Problem endgültig zu lösen, sondern ein Produkt bereitzustellen, mit dem die Anwender ihre Probleme immer besser lösen können. Das Produkt hat immer einen aktuellen Zustand. Die Zustände erfolgen in Intervallen. Der Übergang von einem Zustand zum anderen könnte man zwar als Projekt bezeichnen aber der Begriff „Release“ passt besser. Ein Release ist auch zeitlich und kostenmäßig begrenzt, aber im Gegensatz zum Projekt steht nicht das Ziel als Erstes an, sondern die Zeit und die verfügbare Kapazität. Das Ziel eines Release wird der gesetzten Zeit und dem erlaubten Aufwand angepasst. D.h. man setzt erst einen Termin, dann wird entschieden was man in der Zeit mit den vorhandenen Ressourcen erreichen kann. Wenn die Anforderungen mehr oder dringend sind, obliegt es dem Anwender das Budget für das Vorhaben aufzustocken. Insofern darf das Budget nicht fixiert sein. Es wird immer der Situation angepasst.

Software ist letztendlich ein geistiges Produktionsmittel, die eingesetzt wird, um bestimmte menschliche Arbeiten zu erleichtern oder gänzlich zu übernehmen. Derartige IT-Produkte kommen nicht auf Anhieb in einem einzigen einmaligen Projekt zustande, sondern sind das Ergebnis eines langen Reifeprozesses, das sich über einen längeren Zeitrahmen hinstreckt. Laut ISO-Standard 12207 ist dies der Produktlebenszyklus [ISO95]. Bis das Produkt gut nutzbar ist, bedarf es viele Releases. Auch danach folgen immer weitere Releases, allerdings nicht unbedingt in den gleichen kurzen Intervallen wie zuvor. Fertig bzw. „Done“ wird es nie. Der Weg ist das Ziel.

2 Risiken in der Produktentwicklung

Das eigentliche Thema ist demzufolge nicht das Projektmanagement sondern das Produktmanagement. Es müssen zunächst die IT-Produkte definiert und modelliert werden, ehe über ein Projekt nachgedacht wird, denn es sind die Produkte, welche die fachlichen und technischen Anforderungen erfüllen. Produkte dienen einem wirtschaftlichen Zweck, und sie haben eine technische Basis. Sie existieren unabhängig von Projekten. Deshalb müssen wir weg vom Projektdenken hin zum Produktdenken. Produkte und ihre Architektur sollen künftig im Vordergrund der Betrachtung stehen, statt wie bisher Projekte.

Wenn dies gelingt werden die Risiken und damit die vielen Misserfolge der Softwareentwicklung zwar nicht aus der Welt geschafft aber erheblich reduziert. Die fünf Hauptrisiken, die nach DeMarco und Lister immer wieder vorkommen sind

- falsche Einschätzungen der Termine und Aufwände,
- keine Einigung über die angestrebten Ergebnisse,
- ständig veränderte Anforderungen (*creeping requirements*),
- Verlust an Schlüsselpersonen und
- Überschätzung der eigenen Leistung [DeLi03].

Der Verlust maßgeblicher Personen ist ein Risiko, wovor kein menschliches Vorhaben gefeit ist. Nicht nur IT-Projekte, sondern auch IT-Produkte und sogar Softwarefirmen, sind von Schlüsselpersönlichkeiten abhängig. Dies liegt am Wesen von Software als geistige Substanz. Durch eine langfristige Strategie können aber die Folgen vom Personalausfall abgemildert werden. Man hat mehr Zeit um auf den Personalverlust zu reagieren und kann auch mit Personalreserven besser vorbauen. Die anderen vier Risiken haben jedoch mit dem Wesen des Projektmanagements zu tun. Sie sind alle vier Folgen einer falschen zeitlichen und finanziellen Begrenzung der Projekte.

2.1 Falsche Einschätzung der Termine und Aufwände

Das Unvermögen der Menschen, IT-Projekte richtig abschätzen zu können, liegt hauptsächlich daran, dass es unmöglich ist, die Dimensionen eines komplexen IT-Systems im Voraus zu bestimmen. Eine zuverlässige Schätzung ist nur dann möglich, wenn genug Erfahrung mit dem Sachgebiet und der anzuwendenden Technologie vorliegt, d.h., wenn das gleiche Problem mit den gleichen Mitteln schon mehrmals gelöst wurde. Je öfter ähnliche Anwendungen mit denselben technologischen Mitteln entwickelt werden, desto solider die Schätzbasis.

Leider ist dies in der IT Welt nur selten der Fall. Sowohl die Anwendungen als auch die technischen Mittel werden stets komplexer und differenzierter. Selten hat man die Gelegenheit, die gleiche Projektart zu wiederholen. Etwas ist immer anders – das Sachgebiet, die Rahmenbedingungen, die Technologie oder die Menschen. Die Produktivität, die ein Betrieb mit CICS/COBOL auf dem Host oder mit CORBA und C++ in einer Client/Server Umgebung erlangt hat, ist auf neue web-basierte Systeme nur bedingt übertragbar. Dies zwingt dazu, die Erfahrungsbasis immer weiter auszubauen, in der Hoffnung, es könnte vielleicht für das nächste Projekt ausreichen.

Erfahrungswerte können nur aus der Entwicklungspraxis gewonnen werden. Jede Produktentwicklung ist ein Lernprozess. Also muss es ein Release geben, das nur dem Zweck dient, Erfahrungen zu sammeln, die den nachfolgenden Releases zu Gute kommen. Dies setzt aber voraus, dass die Menschen, die jene Erfahrung gewinnen, zusammen bleiben und zwar über den ganzen Lebenszyklus des Produktes hinaus. Mit jedem

Release wachsen die Erfahrungsbasis und die Genauigkeit, mit der das nächste Release geschätzt werden kann. Ausschlaggebend ist, dass auf dem gleichen Fachgebiet mit der gleichen Technologie und der gleichen Mannschaft weitergearbeitet wird.

2.2 Keine Einigung über die angestrebten Ergebnisse

Auch der angestrebte Konsens bezüglich der Ziele eines Produktes, ist nur über einen Annäherungsprozess zu erreichen. Anwender müssen sich erst an das neue System gewöhnen, ehe sie darüber ein endgültiges Urteil abgeben können. Es fällt ihnen schwer, einem abstrakten Modell zuzustimmen, sei es noch so eindringlich präsentiert. Sie brauchen einen lauffähigen Prototyp, mit dem sie sich auseinandersetzen können. Das bedeutet, erst mit Hilfe einer Vorabversion ist ein Konsens über die endgültige Version wirklich möglich.

Die Entwicklung eines funktionsfähigen Prototyps gehört daher als erste Stufe zum Lebenszyklus eines jeden komplexen IT-Produktes, um erstens die Anforderungen der Anwender herauszulocken und zweitens eine Einigung darüber herbeizuführen, wie die Anforderungen mit der Technologie umzusetzen sind. Zu diesem Zweck ist ein eigenes Projekt – ein Prototypbau – unabdingbar.

2.3 Ständig veränderte Anforderungen

Das dritte große Projektrisiko – *creeping requirements* – ist eigentlich eine Folge des zweiten. Solange die potentiellen Anwender keine endgültige Meinung über ihre eigentlichen Anforderungen haben, werden sie ihre Meinung ändern und Neues anfordern. Das Ganze ist auch für sie ein Lernprozess, wofür sie genügend Zeit brauchen. Durch die stufenweise Entwicklung eines IT-Systems haben sie die Möglichkeit, neue Anforderungen einzubringen, ohne auf die Nutzung der bisher realisierten Anforderungen zu verzichten. Da das System offen bleibt, kann es im Sinne von Open Source immer neue Funktionen aufnehmen. Einen Redaktionsschluss gibt es nur für das jeweilige Release.

2.4 Überschätzung der eigenen Produktivität

Die Überschätzung der eigenen Produktivität liegt daran, dass die Projektbeteiligten noch keine Erfahrung mit der vorgesehenen Anwendung in der vorgeschriebenen technologischen Umgebung haben. Sie müssen ihre Produktivität erst entdecken. Erst durch die Messung der Produktivität in den ersten Evolutionsphasen haben die Produktmanager die Daten, die sie brauchen, um die Produktivität in den Folgephasen abzuschätzen. Mit jedem zusätzlichen Release werden die Produktivitätsdaten zu diesem Produkt immer mehr erhärtet und die Schätzgenauigkeit immer besser. Die Team Performance lässt sich immer besser von Release zu Release hochrechnen [PlPo12].

Es bleibt also am Ende nur das fünfte Risiko – der Verlust von Teammitgliedern – übrig. Die restlichen Risiken können über die Einführung des Produktmanagements erheblich gemildert werden. Sie sind alle Folgen eines falsch verstandenen Projektbegriffes mit

starren Terminen und fest vereinbarten Leistungsumfängen in einer Welt, in der es auf Flexibilität und Anpassungsfähigkeit ankommt. Es mag zwar andere Bereiche geben, in denen dies unumgänglich ist, vor allem in der Prozesssteuerung und bei der Entwicklung von Gerätesoftware, aber im Bereich sozio-ökonomischer Systeme sind sie fehl am Platz. Die meisten betriebswirtschaftlichen Systeme sind das Produkt eines langen Evolutionsprozesses. Sie entstehen nicht durch eine einzige, einmalige Anstrengung, sondern durch viele aufeinander aufbauende Entwicklungsschübe, die das IT-Produkt stufenweise quantitativ erweitern und qualitativ verbessern.

3 Das Wesen der Produktevolution

Eins ist klar zu stellen: IT-Produkte sind von einem völlig anderen Wesen als Software-Produkte, die in Geräte eingebaut werden oder die physikalischen Prozesse steuern. Die anderen beiden Produktarten sind statischer Natur. Sie werden einmal erstellt, getestet, ausgebessert und von da an nur geringfügig geändert. Insofern spielt die Qualität der Entwicklung eine entscheidende Rolle. Sie muss den Funktionsumfang auf Anhieb zum größten Teil abdecken. Der Funktionsumfang ist auch einigermaßen fassbar. Das trifft jedoch für einen großen Teil der IT-Systeme nicht zu. Für die, die es zutrifft – die klassischen Backoffice Systeme, bzw. operative Hintergrundprozesse – sind schon längst Standardlösungen im Einsatz. Für die meisten IT-Systeme, die es noch zu entwickeln gibt – die Frontoffice Systeme, bzw. kundennahe Vordergrundprozesse – trifft es nicht zu. Ihr Funktionsumfang ist eine Variable, die sich beliebig einstellen lässt – mal mehr, mal weniger – je nachdem, wie viel er dem Unternehmen im Moment nutzt [Wood99].

Funktionalität von IT-Produkten ist deshalb keineswegs fest. Sie liegt auf einer großen Bandbreite von einer minimal akzeptierbaren zu einer maximal wünschbaren. Welche Funktionalität zu welchem Zeitpunkt realisiert wird, ist eine Frage des Nutzwertes und der verfügbaren Ressourcen. Man kann mit einer minimalen Lösung anfangen und sich allmählich an eine optimale Lösung herantasten. Obwohl, je näher man an das Wunschziel herankommt, desto mehr rückt es in weite Ferne, denn für jedes Problem, das gelöst wird, werden zwei neue geschaffen. Von einer Erfüllung aller Wünsche kann nicht die Rede sein. Allenfalls von einem ausreichenden Zustand, mit dem sich alle Beteiligten abfinden können.

Sowohl Funktionalität als auch Qualität von IT-Produkten sind relativ, relativ zu dem, was man gerne hätte, zu dem, was gerade geläufig ist und zu dem, was man bisher gehabt hat. Darum ist es so schwer, einen Konsens über den Umfang und die Qualität eines Produktes zu finden. Das, was ausreichend ist, stellt sich erst in der Nutzung des Produktes heraus. Daher kann ein IT-Produkt nur stückweise entstehen.

Die Vorausdefinition der erforderlichen Funktionalität und Qualität ist eine reine Hypothese, die erst durch die Benutzung derselben bestätigt wird. Diese Binsenweisheit hat Gilb schon Mitte der 80er Jahre unter dem Begriff „evolutionäre Software Entwicklung“ propagiert [Gilb88]. Kurz darauf folgte Boehm mit seinem sogenannten Spiralen Modell der Produktentstehung [Boeh88]. Es ist also keineswegs neu, dass IT-Produkte das Ergebnis eines evolutionären Prozesses sind. Umso erstaunlicher ist es, dass IT-Manager

immer noch das klassische Projektmanagement als Leitbild pflegen. Das dies immer noch so ist, haben wir einem falsch verstandenen Projektbegriff zu verdanken.

Das Leitbild für das IT-Produktmanagement ist die objektorientierte Denkweise. Im Mittelpunkt steht nicht die Aktivität bzw. das Projekt, sondern das Objekt bzw. das Produkt. Releases sind mit den Methoden zu vergleichen, die einem Objekt zugeordnet sind. Sie versetzen das Produkt von dem einen Zustand in den anderen.

So gesehen sind Releases als Zustandsübergänge auf dem langen Lebensweg eines Produktes zu betrachten. Innerhalb eines einzelnen Releases finden unterschiedliche Aktivitäten statt, die parallel zu einander ablaufen:

- Fehlerbehebung,
- Änderung,
- Sanierung,
- Weiterentwicklung und
- Integration (siehe Abbildung 2).

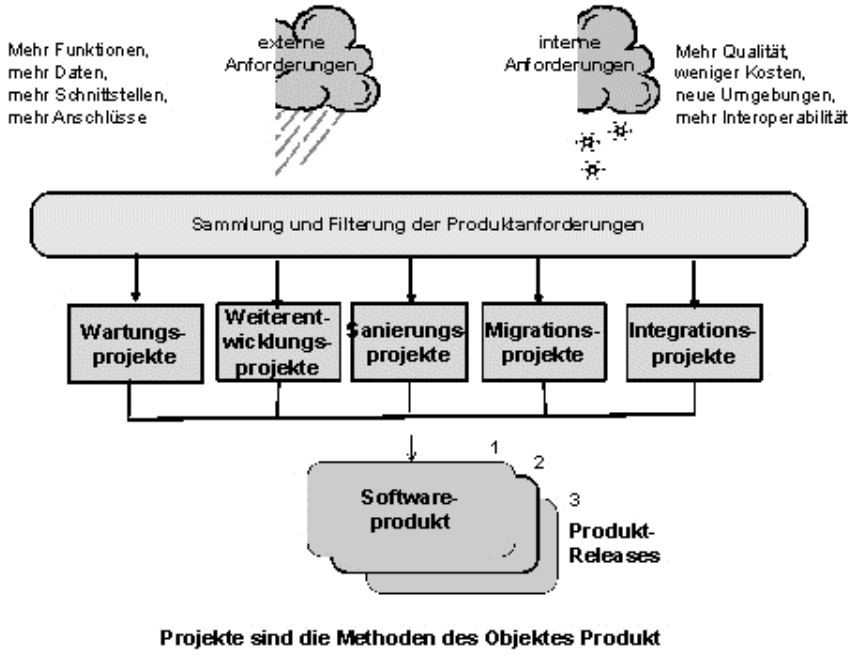


Abbildung 2: Software-Evolutionsprojekte

Fehlerbehebungsaktivitäten korrigieren das Produkt und versetzen es in einen Zustand, den es von Anfang an hätte haben sollen.

Änderungsaktivitäten verwandeln das Produkt, ohne es funktional zu erweitern. Bestehende Komponenten werden fachlich den veränderten fachlichen Anforderungen und technisch den veränderten technischen Anforderungen angepasst.

Sanierungsaktivitäten verbessern die innere Qualität des Produktes. Über diverse Reengineering, bzw. Refactoring Maßnahmen wird die Software in einen technisch besseren Zustand versetzt.

Weiterentwicklungsaktivitäten bauen die Funktionalität des Produktes aus. Es werden zusätzliche Funktionen und Daten eingefügt, die den Nutzwert des Produktes steigern.

Integrationsprojekte verbinden das Produkt mit fremden Produkten, damit es mit ihnen interagieren kann.

Die Auswirkung dieser vielen Aktivitäten auf das Produkt lässt sich am besten mittels eines Zustandsübergangsdiagramms darstellen. Der Ausgangszustand des Produktes vom letzten Release ist der Anfangszustand zum nächsten Release (siehe Abbildung 3).

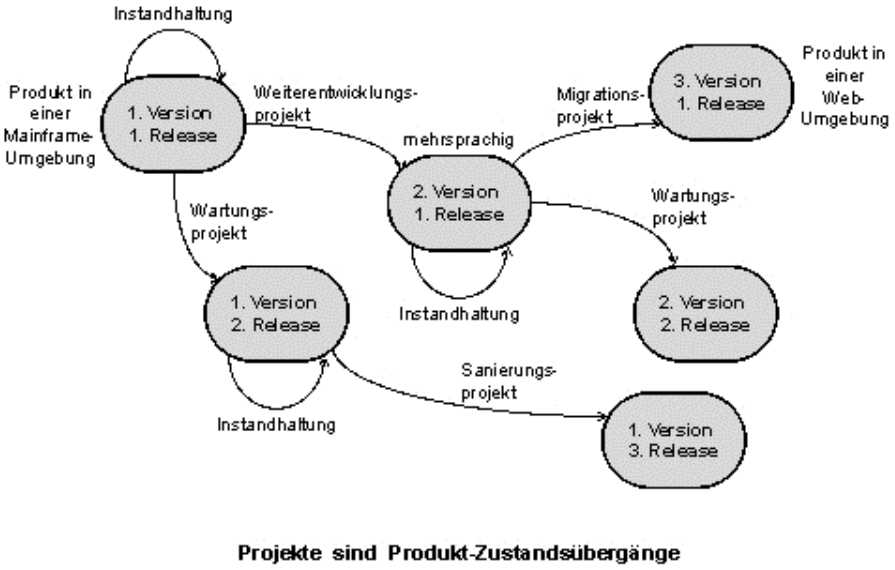


Abbildung 3: Produktzustandsübergänge

4 Planung der Software-Evolution

Die Gesetze der Software-Evolution erfordern einen gesteuerten Evolutionsprozess als Alternative zu einer willkürlichen, ad hoc Weiterentwicklung [Lehm85]. Um den Nutzwert der Software möglichst lange zu erhalten, müssen Wartung und Weiterentwicklung eine konsequente Strategie folgen. Natürlich ist es nicht möglich sämtliche Änderungen vorauszusehen und exakt zu planen. Die Evolution eines Software-Systems ist vom Wesen her stochastisch. Fehler treten unerwartet auf, notwendige Änderungen ergeben sich aus der technischen sowie aus den organisatorischen und marktwirtschaftlichen Umgebungen. Die Produkt-verantwortlichen sind gezwungen zu reagieren auf die Ereignisse, wie Gesetzesänderungen, Kundenwünsche und Konkurrenzsituationen. Sie müssen auch auf technische Erneuerungen in der Hardware und Software reagieren und ihr System anpassen ob sie wollen oder nicht. Sie stehen unter Zugzwang [CMKC03].

Man wäre geneigt zu sagen, es hat keinen Sinn irgendwas zu planen. Die zuständigen Entwickler sollten sich nur zurücklehnen und warten, bis etwas passiert. In dem Fall passte wirklich der Begriff „Wartungsmannschaft“ zu ihnen. Sie warten auf den nächsten Wartungsauftrag. Diese passive, auftragsgetriebene Haltung ist jedoch ein Hauptgrund für den Verfall eines Software-Systems. Damit treten die Gesetze der Software-Evolution in Kraft und das System verliert ständig an Wert. Um dies zu verhindern, müssen die Systemverantwortlichen eine aktive Strategie verfolgen. Die unerwarteten Ereignisse sollten in einen Planungsrahmen hineingezwungen werden. Es werden hierfür keine Kapazitäten vorgesehen, aber der Großteil der Kapazität wird für Weiterentwicklungs- und Sanierungsaufgaben geplant. Sie können, falls erforderlich, davon abgezogen werden aber sie sind zunächst verplant. Die Kunst der Evolutionsplanung liegt darin, das nicht Planbare zu planen.

Der Evolutionsprozess soll zunächst in Release-Intervalle aufgeteilt werden. Neue Releases werden in Intervallen von einem Monat bis zu einem Jahr geplant. Ein neues Release umfasst ein lauffähiges, produktionsreifes System und die dazu gehörige Dokumentation und Testumgebung. Es sollte möglich sein, die Dokumentation mit der Software abzugleichen und den Test der Software jederzeit zu wiederholen. Jedes Release hat diese Mindestkriterien zu erfüllen. Dokumentation, die mit der Produktentwicklung nicht Schritt hält, ist wegzulassen. Es werden nur jene Dokumente fortgeführt, die mit der Software im Einklang stehen. Der Test muss ebenfalls im Gleichschritt mit dem Code fortgeschrieben werden, d. h. Testfälle und Testprozeduren sind ein Spiegelbild des aktuellen Systems [MNS01].

Jedes neue Release ist als Projekt zu betrachten, auch wenn es auf einen Monat beschränkt ist. Es soll dafür zumindest einen groben Plan mit messbaren Planungszielen geben. Es gilt, die Einhaltung der Planungsziele zu kontrollieren, auch wenn sie aus welchen Gründen auch immer nicht einzuhalten sind. Es ist besser, nicht erreichbare Ziele zu haben als gar keine. Die Ziele sollten sowohl qualitativer als auch quantitativer Natur sein. Auf der einen Seite soll die Funktionalität wachsen, auf der anderen Seite die Qualität steigen. Beide Zielarten sind stets im Auge zu behalten. Ohne Evolutionsplan ist es nicht möglich, den Fortschritt der Produktevolution zu verfolgen. Daher müssen die

Evolutionsaktivitäten geplant werden, auch wenn der Plan durch die letzten Ereignisse überholt wird.

5 Ansätze zur Software-Evolution

Ein Problem der Software-Evolution ist, die Beschreibung eines Systems mit dem System selbst synchron zu halten, bzw. den Code mit dem Modell zu synchronisieren, insbesondere dann, wenn das System sich oft und signifikant ändert. Hier werden vier Ansätze zur Lösung dieses Problems geschildert:

- der anforderungsgetriebene Ansatz
- der Top-Down modellgetriebene Ansatz
- der Bottom-Up modellgetriebene Ansatz
- der testgetriebene Ansatz.

5.1 Anforderungsgetriebener Ansatz

Nach dem anforderungsgetriebenen Ansatz zur Software-Evolution bildet das Anforderungsdokument die Basis, auf der das Produkt weiterentwickelt wird. Als Erstes wird das Anforderungsdokument geändert, bzw. erweitert, dann der Code. Jede Anforderung verweist auf den Anwendungsfall, der diese Anforderung erfüllt, und jeder Anwendungsfall verweist auf die Codekomponente, bzw. auf die Services, die diesen Anwendungsfall implementieren. Es bestehen also Links zwischen dem Anforderungsdokument und dem Code. Die Aufrechterhaltung dieser Links ist eine der wichtigsten Aufgaben der Evolution – sie dürfen nicht verloren gehen. Gepflegt werden diese Links über eine System-Repository.

Der Vorteil des anforderungsgetriebenen Ansatzes ist, dass die Anforderungen in natürlicher Sprache beschrieben sind und somit für den Endanwender sowie für den Systemanalytiker verständlich sind. Sie können die Änderungen verfolgen und selbst das Anforderungsdokument fortschreiben. Sie brauchen nur ein Tool, um die Verbindungen zum Code zu erhalten. Eine zusätzliche Modellierungsprache wird nicht benötigt, weil die natürliche Sprache genügt, die Anforderungen und Anwendungsfälle zu beschreiben. Falls das Wartungspersonal einen Überblick über den Code in graphischer Form haben möchte, kann es ein Reverse-Engineering Werkzeug verwenden, um diesen bei Bedarf zu erstellen. Es ist sicher; die Anwender, Manager und Tester werden nicht danach fragen. Sie werden bei ihren Beschreibungen in natürlicher Sprache bleiben. Der Wartungsmannschaft ist in der Regel am besten gedient mit einem Software-Repository und einem flexiblen Abfragedienst, der sie nach Bedarf mit Information versorgt. Keine Studie bezüglich Software-Wartung hat jemals bestätigt, dass UML Diagramme die Wartungskosten wirklich reduzieren. Warum sollte man sie also pflegen [MuNi05]?

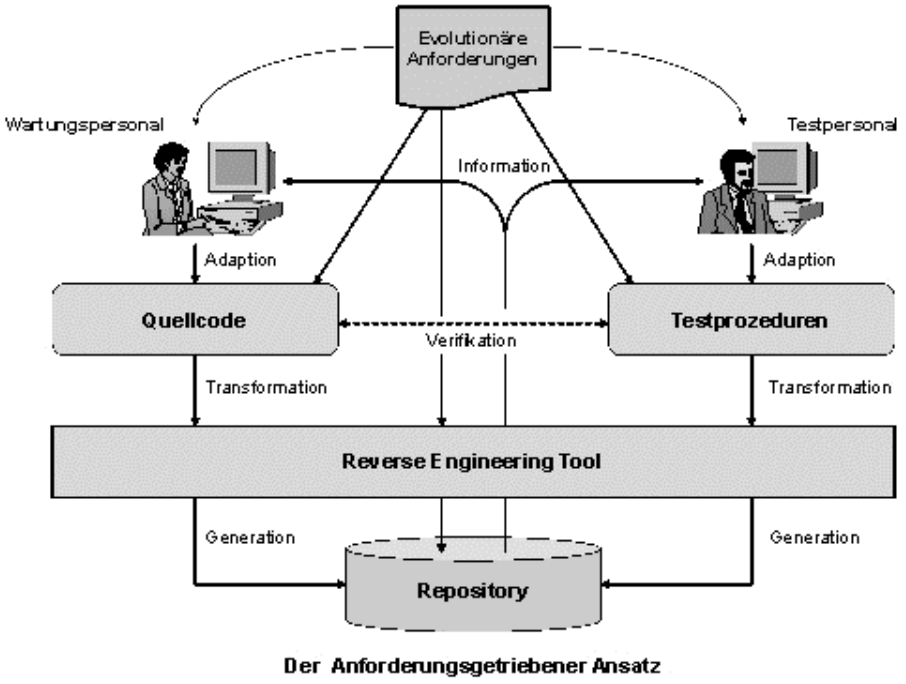


Abbildung 4: Anforderungsgetriebene Evolution

In einem anforderungsgetriebenen Modell werden insgesamt drei getrennte Beschreibungen des Zielsystems nebeneinander existieren

- Das Anforderungsdokument
- Der Source Code und
- Die Testprozeduren (siehe Abbildung 4).

Bei jeder Änderung bzw. Erweiterung wird sowohl der Source Code als auch die Testprozedur angepasst. Wichtig ist, dass diese Anpassungen von zwei verschiedenen Personen mit zwei unterschiedlichen Sichten auf das System ausgeführt werden – dem Programmierer und dem Tester. Auf der Seite des Codes wird der Programmierer die angeforderten Änderungen von den Change-Requests auf die Code-Komponente übertragen. Auf der Seite des Tests wird der Tester bestehende Testfälle ändern und neue Testfälle einfügen, um den Änderungsanforderungen gerecht zu werden. Alle beide – der Programmierer wie auch der Tester – werden Informationen über die Auswirkung ihrer Änderungen benötigen. Zu diesem Zweck wird ein unsichtbares Modell der Anwendung in einem Software Repository mit sämtlichen Systemelementen und deren Beziehungen abgebildet. Eingefügt werden die Elemente und Beziehungen durch eine automatisierte statische Source Code Analyse sowie durch eine automatisierte Analyse der Anforde-

rungstexte und der Testfälle. Durch Querverweise werden die Change-Requests mit den entsprechenden Anforderungselementen und diese wiederum mit den entsprechenden Code- und Testelementen gekoppelt. Auf diese Weise werden Änderungsanforderungen direkt oder indirekt mit den betroffenen Software Elementen verbunden. Über eine Abfragesprache können Programmierer und Tester die Abhängigkeitspfade durch die Repository verfolgen [HDS05].

Im Bereich der Softwarewartung und -weiterentwicklung sind die Tester die Vertreter der Anwender. Es ist ihre Aufgabe, dafür zu sorgen, dass die Anwender das bekommen, was sie verlangen. Zu diesem Zweck arbeitet die Testmannschaft als Gegenpol zur Wartungsmannschaft. Beide holen jedoch ihre Informationen über das System aus der gleichen Quelle – aus der Repository – und zwar über den gleichen Abfragedienst. Ihre Suchfragen sollten möglichst direkt beantwortet werden, ohne dass sie lange Listen von UML Diagrammen durchsuchen müssen. Sie werden auf Anhieb erfahren, wo was zu ändern ist. Dennoch wird die Änderung niemals automatisch durchgeführt. Es obliegt dem Menschen, den Code und den Test fortzuschreiben, denn nur so wird das bewährte Prinzip der doppelten Buchführung aufrechterhalten. Die Kosten werden zwar höher, aber so auch die Qualität der Software. Qualität hat schließlich ihren Preis und der Preis hier ist die Erhaltung zweierlei Systembeschreibungen.

5.2 Top-Down modellgetriebener Ansatz

Das Ziel der modellgetriebenen Software-Evolution wäre, dass die Änderungen am Modell automatisch an den echten Programmcode übertragen werden, wie in Abbildung 5 dargestellt.

Dies setzt eine automatische Transformation zwischen den abstrakten Beschreibungen und den weniger abstrakten voraus. Wird einem Modell eine Funktion hinzugefügt, so sollte sie auch an einer oder vielleicht mehreren Stellen im Code auftauchen. Die Voraussetzung für eine solche Transformation ist, dass die Modellierungssprache eng verbunden ist mit der Implementierungssprache, d.h. die abstrakte Beschreibung ist nicht viel abstrakter als die weniger abstrakte. Desto weiter sich die Modellierungssprache über dem Code erhebt, desto schwieriger und fehleranfälliger ist die Transformation [HaRu04].

Der modellgetriebene Ansatz basiert auf einem klassischen Top-Down-Ansatz der Software-Entwicklung, der an sich schon einen Trugschluss birgt. Die Fürsprecher dieses Ansatzes haben den naiven Glauben an die Fähigkeit durchschnittlicher Entwickler. Sie gehen davon aus, dass diese wüssten, was sie tun. In Wirklichkeit haben sie nicht die geringste Ahnung. Sie spielen mit einem Problem herum, bis sie eine akzeptable Lösung gefunden haben. Wie Balzer schreibt „ist die Implementierung in der Tat weniger eine Verfeinerung der ursprünglichen Spezifikation als vielmehr eine ständige Neudefinition derselben. Es gäbe zwischen Spezifikation und Implementierung eine viel größere Verflechtung als uns die allgemeine Meinung glauben lassen will...“ [BaSw82].

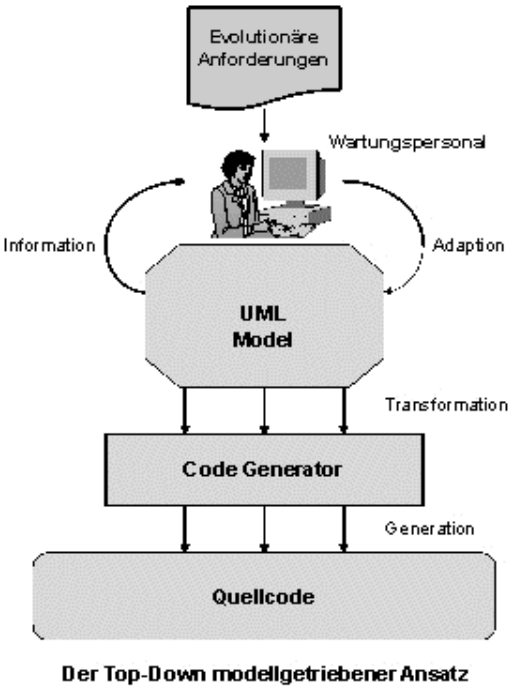


Abbildung 5: Modellgetriebene Evolution

Der Autor konnte in einem Zeitraum von beinahe 40 Jahren zahlreiche Entwickler in verschiedenen Kulturkreisen bei der Arbeit beobachten. Er tut sich schwer, jene Hypothese zu akzeptieren, nach der Entwickler mit UML Werkzeugen die zu lösenden Probleme besser verstehen, als die Entwickler vor 20 Jahren mit CASE Werkzeugen, basierend auf strukturierter Analyse und Entwurf, getan haben. Das Problem damals war der menschliche Anwender dieser Werkzeuge und er ist es immer noch. Die durchschnittlichen Entwickler in der Industrie sind nicht fähig, die Lösung zu einem komplexem Problem zu konzeptionalisieren, unabhängig davon, welche Sprache sie dabei benutzen, um sich auszudrücken oder welches Werkzeug sie für die Implementierung besitzen. Wie Michael Jackson es so treffend formulierte: „Systemanforderungen können niemals ganz im Vorfeld definiert werden, nicht einmal im Prinzip, denn der so genannte Anwender kennt sie nicht im Vorfeld, und dies nicht einmal im Prinzip“ [JaMc82].

5.3 Bottom-Up codegetriebener Ansatz

Ein entgegengesetzter Ansatz ist der Bottom-Up-Ansatz. Die Änderungen werden an der untersten semantischen Ebene der Software, nämlich am Code selbst, vorgenommen und werden nachher über diverse Reverse-Engineering Techniken auf die abstrakteren Beschreibungen übertragen. Wird also eine Schnittstelle im Code implementiert, wird diese Schnittstelle, automatisch übertragen, auch im Modell auftauchen. Dieser Ansatz ge-

währleistet, dass das Modell immer eine aktuelle und wahre Beschreibung des eigentlichen Systems ist. Jedoch muss auch hier die Implementierungssprache mit der Modellierungssprache eng verbunden sein. Alle Konstrukte der Implementierungssprache müssen über ein Äquivalent in der Modellierungssprache verfügen, anderenfalls werden sie übergangen, wie es oft der Fall ist bei der Übersetzung natürlicher Sprachen [Seli03]. Abbildung 6 beschreibt diesen Ansatz.

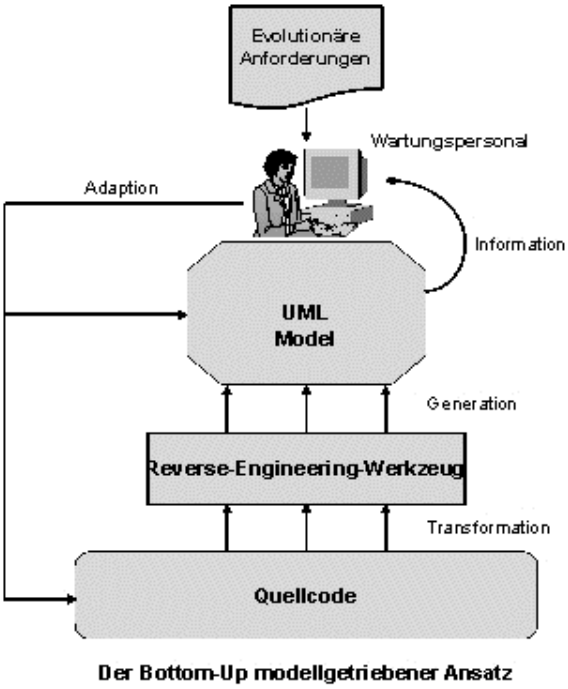


Abbildung 6: Codegetriebene Evolution

Das größte Manko der modellgetrieben Software-Evolution ist aber in beiden Ansätzen, dem Top-Down wie auch dem Bottom-Up-Ansatz, dass es nur eine einzige Beschreibung des Systems gibt. Die andere Beschreibung ist nur eine Übersetzung der ursprünglichen in eine andere Sprache. Im Fall des Top-Down-Ansatzes ist die ursprüngliche Beschreibung ein Modell. Der Code ist nur eine Kopie des Modells in einer anderen detaillierten Form. Im Falle des Bottom-Up-Ansatzes ist wiederum der Code die ursprüngliche Beschreibung und das Modell wird daraus generiert. Als solches ist das Modell nur eine weitere, etwas abstraktere Beschreibung des Codes. In beiden Fällen handelt es sich um eine und dieselbe Beschreibung des echten Systems [Sned89].

Die Frage, die sich hier stellt, ist: Was ist leichter zu ändern – die graphische, abstraktere Beschreibung oder die schriftliche, detaillierte Beschreibung? Theoretiker würden argumentieren, dass es leichter und besser wäre, die Diagramme oder abstrakteren Notatio-

nen zu ändern. Praktizierende Programmierer würden für eine Änderung im Code oder in den detaillierten Beschreibungen plädieren. Beide Gruppen haben gute Gründe für ihre Argumente. Nach 15 Jahren Forschung im Bereich des automatisierten Programmierens kamen Rich und Waters zu dem Schluss, dass „das Verfassen einer vollständigen Spezifikation in einer allgemeinen Spezifikationssprache selten leichter und oft unbeschreiblich viel schwieriger ist als ein Programm zu schreiben. Außerdem gab es nur wenige Erfolge bei der Entwicklung automatischer Generatoren, die effiziente Programme aus Spezifikationen generieren…“ [RiWa88]. Bis heute sieht der Autor keinen Grund, die Programmgeneratoren als wesentlich besser einzuschätzen.

Theoretiker werden behaupten, dass Diagramme leichter zu verstehen sind und einen besseren Überblick bieten. Praktiker werden argumentieren, dass der Code die exakteste Beschreibung von dem ist, was vor sich geht und der Code einen genaueren Einblick in die Softwarekonstruktion ermöglicht. Abgesehen davon wird der Praktiker behaupten, dass er noch nicht weiß, was passieren wird, wenn er eine Änderung macht und dass er erst einmal viele Varianten ausprobieren muss, bis er die richtige findet. Als Praktiker neigt der Autor dieses Papiers dazu, die Sicht des Programmierers zu teilen. Der Teufel liegt in den Details und in den meisten Fällen sind es die letzten 10% der Details, die den Unterschied ausmachen [Math86].

Der Top-Down-Ansatz geht davon aus, dass die Wartungsprogrammierer genau wissen, was sie tun und dass sie in der Lage sind, die Änderungen an ihren Modellen auf den darunter liegenden Code zu projizieren. Die hier zitierten Forscher wussten, dass dies nicht der Fall ist. Wartungsprogrammierer sind geborene Hacker. Wenn sie eine Änderung implementieren müssen, experimentieren sie mit mehreren Alternativen, bis sie eine passende gefunden haben. Konsequent gesehen ist Software-Evolution auf Code-Ebene ein Trial-and-Error Prozess, der so oft wiederholt wird, bis die richtige Lösung gefunden ist. Aus diesem Grund bleibt die Debatte offen, welcher Ansatz wirklich der bessere ist. Es kann vom Systemtyp abhängen, sowie von den Fachkenntnissen des Wartungspersonals [Glas04].

5.4 Testgetriebener Ansatz

Die Frage, ob Top-Down oder Bottom-Up, ist jedoch nicht das Wichtigste. Wesentlicher ist, dass beide Ansätze auf einer einzigen Beschreibung des Software-Systems basieren, da die andere Beschreibung nur eine Übersetzung ist. Dieser Fakt macht beides, modellgetriebene Entwicklung wie auch Evolution, inakzeptabel für die Verifikation und die Validierung. Um ein System zu verifizieren, also um sicher zu stellen, dass das System wahr ist, benötigt man mindestens zwei voneinander unabhängige Beschreibungen des Systems. Testen bedeutet Vergleichen. Ein System wird getestet, indem man das tatsächliche Verhalten gegen das spezifizierte abgleicht [DLP79]. Wenn der Code aus der Spezifikation abgeleitet wurde, ist er nichts anderes als die Spezifikation in einer anderen Form. Der Test des Systems ist dann nichts weiter als ein Test des Übersetzungsvorganges. Um die Qualität eines Systems sicherzustellen, ist es nötig, einen dualen Ansatz zu verfolgen, wie Abbildung 7 veranschaulicht.

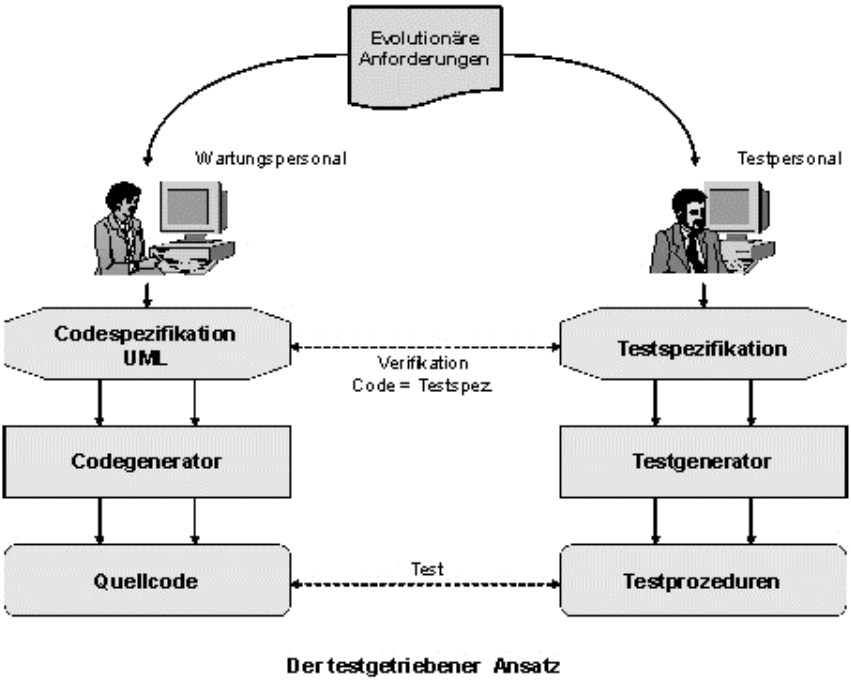


Abbildung 7: Testgetriebene Evolution

Beim Test müssen die Testfälle und Testdaten aus einer anderen Beschreibung des Systems abgeleitet werden, als aus der, von welcher der Code abgeleitet wurde. Dies bedeutet, es muss zwei unabhängige Beschreibungen der endgültigen Lösung geben, vorzugsweise in zwei verschiedenen Sprachen. Eine sollte in der Sprache der Entwickler vorliegen, die andere in der Sprache der Benutzer – und das ist die natürliche Sprache [Fetz88].

6 Die Herausforderung der Software-Evolution

Es ist die Aufgabe des Produktmanagements, ein Anwendungssystem von Release zu Release durch alle fünf Phasen des Lebenszyklus von der Entstehung bis zur Erlösung hindurch zu steuern. Der Produktmanager sorgt für die Kontinuität der Dienstleistung, die das System erbringt, bei gleichzeitig fortdauernder Weiterentwicklung. Dafür braucht er ein langfristiges Produktziel mit einem allgemeinen Evolutionsplan und eine Evolutionsstrategie, wie er dahin kommen will. Dieses Ziel darf jedoch nicht starr sein, es kann sich ändern im Laufe der Zeit. Hingegen muss es für jedes Release ein kurzfristiges und wohldefiniertes Release-Ziel geben. Dieses Ziel beinhaltet die Ergebnisse der parallel laufenden Aktivitäten – die Änderungen, Korrekturen, Erweiterungen, Sanierungen und Integrationsschritte. Nach jedem Release kann neu überlegt werden, wie das nächste Release auszusehen hat. Wichtig ist, dass das Produkt offen bleibt und sich in

jede Richtung weiterentwickeln lässt. Gleichzeitig muss jede Evolutionsstufe eine solide und nachhaltige Basis für die nächste Stufe bieten. Es komme darauf an, sowohl die Flexibilität als auch die Nachhaltigkeit des Softwareproduktes zu bewahren. Jedenfalls sollten wir uns vom dem Begriff „Projekt“ entfernen [Froh02].

Die Herausforderung des Produktmanagers besteht darin, die Erhaltung und Evolution der Anwendung von Release zu Release zu steuern. Er hat dafür zu sorgen, dass kein einziges Release weder die Leistung noch die Nachhaltigkeit des Produktes beeinträchtigt. Im Vordergrund stehen immer die Interessen der aktuellen Benutzer. Sie dürfen in ihren Arbeitsabläufen keineswegs behindert werden – und dies, obwohl das Produkt, mit dem sie arbeiten, stets verändert wird. Ein IT-Leiter verglich dieses Kunststück mit dem Umbau eines Flugzeugs während des Flugs. Es ist wirklich eine echte Herausforderung, die große technische und soziale Kompetenz voraussetzt [Rose03]. Dennoch werden nur solche Software-Anwendungen überleben, mit denen dieses Kunststück gelingt.

Literaturverzeichnis

[BaSw82] Balzert, R.; Swartout, V.: “On the inevitable intertwining of Specification and Implementation”, Comm. of ACM, Vol. 25, No. 7, July, 1982, s. 27

[BeRa00] Bennett, K.; Rajlich, V.: “Software Maintenance and Evolution – A Staged Model”, in the Future of Software Engineering, Proc. of ICSE2000, IEEE Computer Society Press, Limerick, 2000, s. 73

[Boeh88] Boehm, B.: “A Spiral model of Software Development and Enhancement”, IEEE Computer, May 1988, s. 64

[CMKC03] Cusumano, M.; MacCormack, A./ Kemerer, C./ Crandall, B.: “Software Development Worldwide – The State of the Practice”, IEEE Software, Nov. 2003, s. 28

[DeLi03] De Marco, T.; Lister, T.: “Risk Management during Requirements”, IEEE Software, Sept. 2003, s. 99

[DLP79] De Milo: Lipton; Perlis: “Social Processes and Proofs of Theorems and Programs”, Comm. Of ACM, Vol. 22, No. 5, May, 1979.

[Fetz88] Fetzer, J.: “Program Verification – The very Idea”, Comm. Of ACM, Vol. 31, No. 9, Sept. 1988

[Froh02] Fröhlich, A.: Mythos Projekt – Projekte gehören abgeschafft, Galileo Press, Bonn, 2002

[Gilb88] Gilb, T.: Principles of Software Engineering Management, Addison-Wesley Pub., Wokingham G.B., 1988, s. 83

[Glas04] Glass, R.: “Learning to distinguish a Solution from a Problem”, IEEE Software, May, 2004, s. 111

[HaRu04] Harel, D.; Rumpe, B.: “Meaningful Modelling – The Semantics of Semantics”, IEEE Computer, Oct. 2004, s. 64

[Howa01] Howard, A.: “Software Engineering Project Management”, Comm. of ACM, Vol. 44, No. 5, May 2001, s. 23

[HDS05] Hayes,J.; Dekhtyar, A./ Sundarian, S.: “Improving after the Fact Tracing and Mapping of Requirements”, IEEE Software, Dec. 2005, s.30

[Mira02] Miranda, E.: “Planning and Executing Time-Bound Projects”, IEEE Software, March 2002, s. 73

[ISO95] ISO/IEC: ISO Standard 12207 – Software Life cycle processes, International Standard Organization, Geneva, 1995

[JaMc82] Jackson, M.; McCracken, D.: “Life cycle Model considered harmful”, SE-Notes, Vol. 7, No. 1, April 1982, s. 11

[Lehm85] Lehman, M.: Program Evolution, Academic Press, London, 1985, s. 12

[Math86] Mathis, R.: “The last 10%”, IEEE Trans. On S.E., Vol. 12, No. 6, June, 1986, s. 572

[MNS01] Murphy, G., Notkin, D., Sullivan, K.: “Software Reflexion Models - Bridging the Gap between Design and Implementation”, IEEE Trans. on S.E., Vol. 27, Nr. 4, April 2001, s. 364

[MuNi05] Munson, J.; Nikora, A.: “An Approach to the measurement of Software Evolution”, Journal of Software Maintenance and Evolution, Vol. 17, No. 1, Jan. 2005, s. 65

[PlPo12] Plewan, H.J. / Poensgen, B.: Produktive Softwareentwicklung - Acht Faktoren für mehr Produktivität und Qualität, Objektspektrum, 2-2012, s. 12

[RiWa88] Rich, C.; Waters, R.: “The Programmer’s Apprentice”, IEEE Computer, Nov., 1988, s.15

[Rose03] Rosenberg, M.: „Ergo-Töchter leiden unter IT-Bereinigung“, Computerwoche, Nr. 44, Okt. 2003, s. 1

[Seli03] Selic, B.: “The Pragmatics of Model-Driven Development”, IEEE Software, Sept. 2003, s. 19

[Sned89] Sneed, H.: “The Myth of Top-Down Development and its Consequences for Software Maintenance”, Proc. Of 5th ICSM, IEEE Computer Society Press, Miami, Nov. 1989, s. 3

[Wood99] Woodward, S.: “Evolutionary Project Management“, IEEE Computer, Oct. 1999, s. 49

[ZhPa11] Zhang, Y./Patel, S.: “Agile Model-Driven Development in Practice”, IEEE Software, March 2011, s. 84

Kontinuierliche Prozessverbesserung durch Testautomatisierung

Andre Heidt, Stephan Kleuker

Archimedon Software GmbH
Marienstr. 66
D-32427 Minden
Andre.Heidt@archimedon.de

Hochschule Osnabrück
Barbarastr. 16
D-49076 Osnabrück
S.Kleuker@HS-Osnabrueck.de

Abstract: Für kleine und mittelständische Unternehmen (KMU), die Software-Produkte herstellen, stellt sich kontinuierlich die Herausforderung, die Software wart- und erweiterbar zu halten. Dabei müssen die Maßnahmen meist im laufenden Betrieb eingeführt werden und der Return of Investment (ROI) möglichst garantiert sein. Dieser Bericht beschreibt, mit welchen Schritten eine Testautomatisierung zu einer kontinuierlichen Softwareentwicklungsprozessverbesserung bei einem KMU geführt hat. Weiterhin wird andiskutiert, welche Rahmenbedingungen analysiert werden müssen, um die unternehmensindividuell passendsten Maßnahmen zur Prozessverbesserung zu bestimmen, die auch unmittelbaren Einfluss auf die Nachhaltigkeit der Produkte haben.

1 Einleitung

Die typische Historie eines KMU im Bereich Software-Entwicklung besteht meist darin, dass Experten eines Anwendungsbereichs feststellen, dass sie eine Software suchen, die auf ihre individuellen Anforderungen zugeschnitten ist, und diese nicht finden. Dies führt entweder unmittelbar zur Neugründung einer Firma oder eine Software wird in einer Firma des Anwendungsbereichs entwickelt, dann festgestellt, dass diese auch für andere Unternehmen nutzbar sein kann, und dann eine Ausgründung einer Entwicklungsfirma durchgeführt. Für den Erfolg des KMU ist es essenziell, die speziellen Wünsche des Anwendungsbereichs zu kennen, um sich gegen andere Hersteller vielleicht funktional mächtigerer aber aufwändig anzupassender Software durchzusetzen, oft auch ohne Nachhaltigkeit im Kalkül zu haben.

Mit ersten Markterfolgen werden Schritt für Schritt neue eigene Ideen und individuelle Kundenwünsche in die Software eingebaut. Dabei muss ein oftmals kleines, eng zusammenarbeitendes Team von Entwicklern dann personell ergänzt werden. Irgendwann

muss dann der Übergang von der „individuellen Heldenprogrammierung“, bei der alle Beteiligten alle Details des Programm-Codes kennen, zum systematischen Software Engineering stattfinden. Zwar gibt es hierzu eine Vielzahl hilfreicher Literatur, allerdings bleibt die Frage, welcher Teilprozess zunächst verbessert werden sollte, nicht einfach beantwortbar. Falsche Entscheidungen können dabei leicht die Existenz eines KMU gefährden, da das tägliche Geschäft kontinuierlich weiterlaufen muss.

Dieser Bericht zeigt ein Beispiel, wie auf Grundlage einer Analyse eine erfolgreiche Verbesserung des Software-Entwicklungsprozesses im Bereich der Testautomatisierung mit Schwerpunkt auf dem GUI-Test durchgeführt wurde. Zunächst werden im folgenden Kapitel einige Hintergründe zu den Herausforderungen von KMUs zusammengefasst, dann die Ausgangssituation im Unternehmen beschrieben und dann die konkrete Umsetzung der Prozessverbesserung vorgestellt. Abschließend wird eine Analyse skizziert, wann der genutzte Weg auch für andere KMUs interessant ist. Eine effiziente Systementwicklung hat als wichtigen Nebeneffekt unmittelbaren Einfluss auf die Ressourcen, die bei der Produkterstellung benötigt werden.

Die Arbeiten wurden teilweise mit Mitteln des Bundesministeriums für Bildung und Forschung (BMBF) im Projekt KoverJa (Korrekte verteilte Java-Programme) durchgeführt.

2 Hintergrund

Software-Engineering [Ba00][Wi05][LL06][Kl11] hat sich als wissenschaftliche Ingenieurdisziplin schrittweise in der Informatik beginnend mit der Erkenntnis etabliert, dass die Entwicklung von Software für komplexere Systeme systematisch organisiert werden muss, um die Erfüllung von Qualitätsansprüchen sowie die Wart- und Erweiterbarkeit zu ermöglichen. Das Software-Engineering zerfällt in einige Teilprozesse, die Hand in Hand arbeiten können. Statt eines detaillierten Ausblicks werden hier einige dieser Prozesse mit Beispielen aus der Praxis aufgezählt, wie eine schwächere Umsetzung in einem KMU zu Problemen führte. Die Beispiele stammen aus den Tätigkeiten des zweiten Autors als Berater und Betreuer von Abschlussarbeiten.

Anforderungsanalyse: Durch die steigende Komplexität des Systems verstanden die Entwickler nicht mehr, was wozu in die Software eingebaut wurde. Bei unklaren Aufgabenstellungen wurden Annahmen getroffen, die in der Praxis nicht passten.

Design und Architektur: Durch eine unsaubere Trennung der Datenhaltung von der Business-Schicht und mangelnde Aufteilung in Software-Komponenten musste für jede Änderung das gesamte Programm geändert werden, was mangels Kenntnis des Gesamtsystems auch zu Doppelentwicklungen führte.

Implementierung: Dadurch, dass Entwicklern zu viele Freiheiten in der Codierung und der Umsetzung gegeben wurden, war der Code für andere Entwickler unleserlich und wurde ein konzipiertes Mehrschichtensystem nicht eingehalten.

Test: Dadurch, dass alle Tests manuell durchgeführt werden mussten, stieg mit wachsender Systemgröße die benötigte Zeit für das teilweise auch unsystematische Testen enorm an, wobei sich trotzdem die Fehlerquote erhöhte.

Konfigurationsmanagement: Dadurch, dass für verschiedene Kunden Varianten der gleichen Software entwickelt wurden, die nicht miteinander integriert wurden, musste jede Variante sehr aufwändig individuell gewartet und gepflegt werden, so dass es sich von außen um eigenständige Softwaresysteme handelte.

Aufwandsschätzung: Dadurch dass es keinen funktionierenden Prozess gab, in dem von Entwicklern angegebene, konsolidierte Aufwände in die Schätzung eingingen, Endzeiten aber eingehalten werden mussten, sank die Qualität durch nicht durchgeführte Tests.

Risikomanagement: Durch ein wachsendes Unternehmen wurde die Distanz zwischen Entwicklern und der Geschäftsleitung immer größer, so dass den Entwicklern bekannte potenzielle Problemquellen nicht an die Leitung kommuniziert wurden.

Die Beispiele zeigen deutlich, dass zwar alle Prozesse auch in KMUs sinnvoll umgesetzt sein müssen, es aber häufig einzelne oder kleine Gruppen von Prozessen gibt, bei denen entweder schon unmittelbarer Handlungsbedarf besteht oder zumindest das größte Optimierungspotenzial mit den größten mittelfristig zu erwartenden ROI liegt, der besonders durch die benötigte Arbeitszeit von Mitarbeitern beeinflusst wird.

3 Prozessanalyse

Das Projekt wurde in der Firma Archimedon durchgeführt. Hierbei wurde das Ziel verfolgt, den Qualitätssicherungsprozess für die Project and Business Software „admileo“ zu verbessern. admileo ist das Hauptprodukt der Archimedon Software + Consulting GmbH & Co. KG [@arc], einem KMU mit derzeit 16 Mitarbeitern, das auf die Entwicklung von Java Software spezialisiert ist. Die Firma mit Hauptsitz in Minden (NRW) wurde im Jahr 2004 gegründet und unterhält seit 2010 eine Zweigstelle in Osnabrück (NDS). Archimedon arbeitet eng mit der Hochschule Osnabrück zusammen. Die Firma legt sowohl bei der Softwareentwicklung als auch in anderen Geschäftsprozessen großen Wert auf systematisches Vorgehen und wiederholbare Prozesse. Dies zeigt sich beispielsweise darin, dass Archimedon ISO-9001:2008 zertifiziert ist und ausschließlich auf Entwickler mit Hochschulabschluss in der Informatik setzt. Bei der Softwareentwicklung hat Archimedon von Anfang an langfristig geplant. Ausgehend von einem Lebenszyklus der Software admileo von 15–20 Jahren, sind schon in der Designphase Entscheidungen für eine möglichst hohe Wartbarkeit und Wiederverwendbarkeit getroffen worden. Hierzu zählen vor Allem die Modularisierung und die komponentenbasierte Softwareentwicklung. Für eine nachhaltige Sicherstellung dieser Ziele, führt die Firma regelmäßig Projekte zur Analyse und zur Verbesserung der eingesetzten Prozesse durch und evaluiert neue Techniken und Technologien. Dank dieser firmeninternen Zielsetzung ist die Idee für die Einführung der Testautomatisierung, die im Folgenden ausführlicher beschrieben wird, sowohl bei der Geschäftsführung als auch bei den Entwicklern kooperativ begrüßt worden.

Das Projekt zur Verbesserung der Qualitätssicherung ist in erster Linie darauf ausgerichtet, die Project and Business Software admileo, das Hauptprodukt mit den Bereichen Verkauf, Beratung und Anpassung bei der Einführung sowie der optionalen Möglichkeit die Software zentral zu hosten, der Firma Archimedon, zu testen. Alle Ergebnisse und Erfahrungen sollen aber auch in weitere, zukünftige Projekte von Anfang an mit einflie-

ßen. Bei admileo handelt es sich um ein modular aufgebautes Client-Server-System mit Datenbankanbindung, welches fast ausnahmelos in Java implementiert ist. Der Server bildet die Schnittstelle zur Datenbank und übernimmt die Benutzerverwaltung und Steuerung der Kommunikation mit den Clients. Er kann auf diese Weise zum Beispiel die Aktualisierung der Clients veranlassen. Die Benutzeroberfläche von admileo basiert auf dem Swing Framework und verwendet auch zahlreiche modifizierte und erweiterte Swing Elemente für die Darstellung und die Funktionalität der GUI-Elemente. Anwendungsbeispiele hierfür sind die grafische Aufwertung von Navigationsbäumen und die Darstellung und Editierung von Graphen und Diagrammen.

Zur Zeit des Projekts bestand admileo aus 3 Haupt- und 46 erweiternden Modulen und besaß einen Umfang von weit über 1,5 Millionen Lines of Code (LoC). Das mit steigender Tendenz, da die Weiterentwicklung und Verbesserung von admileo ein ständiges Ziel der Firma darstellt. Ein erster Rückblick auf die Art und Weise der vorangegangenen Softwareentwicklung hat verdeutlicht, dass das modulare Konzept eine Weiterentwicklung des Systems generell zwar wesentlich vereinfacht hat, der Prozess zum Testen der Software für große Releases jedoch stark angestiegen ist. Wegen dieser Steigerung, ausgehend von wenigen Personenmonaten auf über das Doppelte, wurde der Bereich „Test“ als Optimierungsfeld identifiziert.

Hierbei ist eine Anforderung an die zu entwickelnde Lösung ihre Integrierbarkeit in die bisherigen Entwicklungsprozesse gewesen. Um zu beurteilen, welche weiteren Schritte dabei sinnvoll sind, wurde der bereits etablierte Entwicklungs- und Testprozess analysiert. Dieser Entwicklungsprozess von admileo ist angelehnt an das V-Modell, mit klar definierten Entwicklungs- und Testphasen. In den folgenden konkreten Testphasen wird vollständig manuell getestet:

- Fortlaufender Test in der Entwicklungsumgebung
- Kontinuierlicher Integrationstest im Testsystem
- Betatest der Module auf Testsystemen
- Qualitätssicherungstest
- Abnahmetest

Das Bauen von admileo und die Integration zu Gesamtsystemen für Tests und Releases findet mit Hilfe von Build-Tools wie Ant [@ant] und dem Continuous Integration (CI)-Server Jenkins [@jen] vollständig automatisiert statt. Der Release-Zyklus von admileo weist zwei Versionen pro Jahr auf. Hierbei gehen jedem Release Phasen des intensiven manuellen Testens aller Bestandteile sowie des Gesamtsystems voraus.

Als Resultat der Analyse ist die Idee entstanden, durch eine Testautomatisierung oft wiederholte Testvorgänge abzudecken. Die Abläufe sollen dabei stets identisch sein und somit zwischen den einzelnen Versionen konkret vergleichbare Ergebnisse liefern. Hierdurch soll die Regression der Qualität des Codes verhindert werden. Darüber hinaus eignet sich die unbeaufsichtigte Durchführbarkeit von automatisierten Tests dazu, die Entwickler von monotonen, wiederkehrenden Testvorgängen zu entlasten und ihnen mehr Zeit für anspruchsvollere Tests sowie den Ausbau und die Pflege der Testfallsätze zu verschaffen. Automatisierte Tests können zudem regelmäßig ausgeführt werden und Fehler möglichst zeitnah zu ihrer Entstehung aufzeigen.

Das kontinuierliche, automatisierte Bauen und Integrieren von admileo mit Hilfe von Ant und Jenkins lässt es zu, diese Testvorgänge in das System zu integrieren. Es soll somit ein CI-System [Hu09] entstehen, das kontinuierlich und automatisiert bauen und testen kann. Folgende Testmethoden wurden für solch ein System im Rahmen einer Teststrategie [Li09] festgelegt:

- Unit Tests und Integrationstests
- Funktionstests über die Benutzeroberfläche (GUI-Tests)
- Messung der Codeüberdeckung (Code Coverage)
- Statische Quellcodeanalysen

Um nach einem Einstieg in die Testautomatisierung mitten im Entwicklungszyklus von admileo möglichst schnell umfangreiche Teile erfolgreich testen zu können, wurde beschlossen, das Hauptaugenmerk auf die GUI-Tests zu legen, wodurch eine hohe Systemabdeckung mit den Tests erzielt werden kann. Die Multiprojektmanagement-Software admileo ist ein typisches Beispiel für eine Software, deren Funktionalität meist unmittelbar von einem GUI aus zugänglich ist. Durch die Verwendung der Überdeckungsmessung sowohl bei den Unit Tests als auch bei den GUI-Tests, können sich beide Testverfahren ergänzen. Codestellen mit geringer Codeüberdeckung können so durch die Ergänzung sinnvoller Tests aus einer der beiden Testmethoden abgedeckt werden.

Für die Umsetzung der Testmethoden mussten passende Werkzeuge ausgewählt werden, die in ein CI-System integrierbar sind. Ausgewählt wurde sowohl aus dem kommerziellen Bereich, als auch aus Open Source Werkzeugen.

4 Werkzeugauswahl und Umsetzung des Prozesses

Für das CI-System wurden Testwerkzeuge ausgewählt, die sowohl die Anforderungen erfüllen sollen, die durch die Teststrategie vorgegeben sind, als auch die Integrierbarkeit in bestehende Prozesse und Systeme gewährleisten. Die Verwendung weiterer Werkzeuge, um diese Testwerkzeuge zu einem automatisierten Gesamtsystem zu integrieren und die Benutzbarkeit und Wartung zu erleichtern, wurde während prototypischer Umsetzungen zu Evaluationszwecken nach Bedarf beschlossen.

Folgende Testwerkzeuge wurden nach der Analyse für das CI-System ausgewählt:

- QF-Test [@qfs] für GUI-Tests
- Sikuli X [@sik] als Ergänzung zu QF-Test
- JUnit [@jun] für Unit Tests und Integrationstests
- JaCoCo [@jac] zur Überdeckungsmessung
- Sonar [@son] mit den Checkstyle und PMD Plugins für Quellcodeanalyse, Softwaremetriken und die Verwaltung der Testergebnisse

Für die Automatisierung des Bau- und Testprozesses kommen der CI-Server Jenkins und das Build-Werkzeug Ant zum Einsatz.

Weitere Werkzeuge in unterstützenden Rollen sind:

- Hyper-V [@hyp] zur Virtualisierung von Testsystemen
- Jython als Skriptsprache

- Apache Tomcat [@tom] als Plattform für Sonar und Jenkins
- PostgreSQL [@pos] als Datenbank
- Subversion (SVN) [@svn] zur Versionsverwaltung

Das System soll als Benutzerschnittstellen die Weboberflächen von Jenkins und Sonar zur Verfügung stellen, wobei Änderungen an dem Bau- und Testprozess sowie manuelle Builds in Jenkins vorgenommen werden können. Sonar hingegen bietet neben einer umfangreichen Übersicht über den Projektstand anhand eines „Cockpits“ mit konfigurierbaren Metriken und Trends auch die Testergebnisse und stilistische Auswertungen des Quellcodes. Das CI-System soll also, gesteuert von Jenkins, möglichst jedes neue Build einer Reihe von automatisierten Tests unterziehen und die Ergebnisse von Sonar auswerten lassen, um sie darauf folgend in einer Datenbank zu persistieren. Testfälle für admileo werden in einem SVN-Repository verwaltet.

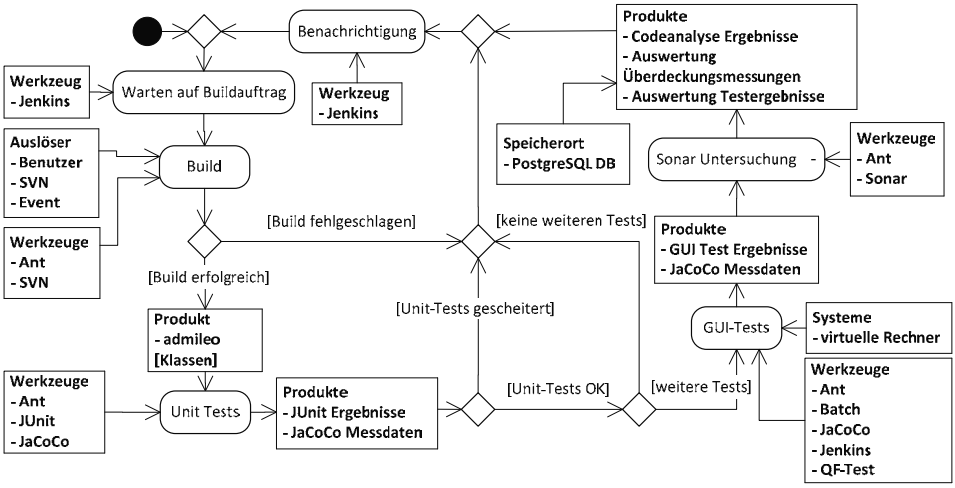


Abbildung 4: Build-Prozess mit integrierter Qualitätssicherung

Der bisherige Ablauf eines Builds (ohne Tests) bestand aus dem Bauen von admileo, gefolgt von der Integration zu unterschiedlichen Testsystemen. Dieser Ablauf wurde nun dahingehend erweitert, dass unmittelbar nach dem Bauen Tests mit möglichst kurzer Ausführungszeit durchgeführt werden. Daraus resultiert der Vorteil, dass Entwickler eine schnelle Rückmeldung erhalten, falls in diesem Schritt bereits Fehler entdeckt wurden. Weiteres, zeitaufwändiges Testen wäre in diesem Fall nämlich ineffizient, bevor die Fehler behoben sind. Im ersten Schritt sollen zu diesen „schnellen“ Tests alle Unit Tests zählen, bis deren Anzahl und ihre Laufzeit so weit angewachsen sind, dass eine weitere Unterteilung sinnvoll wird. Diesen ersten schnellen Tests folgen schließlich weitere Tests, wozu vor Allem die GUI-Tests gehören. Vorausgesetzt in den vorhergehenden Tests wurden keine kritischen Fehler gefunden, wird der Code abschließend einer statischen Codeanalyse unterzogen. Hierzu kommen diverse Plugins, unter Sonar, darunter Checkstyle und PMD, zum Einsatz. Alle bis zu diesem Zeitpunkt akkumulierten Testergebnisse fließen in die Berechnungen der Softwaremetriken ein. Der weitere Ablauf kann nach erfolgreichem Bestehen der Tests wie bisher stattfinden. Der Gesamtablauf ist auch in Abb. 1 dargestellt, Verfeinerungen der Aktionen stehen in [He12].

Die technische Umsetzung des CI-Systems besteht hauptsächlich aus der Installation und der Konfiguration der Werkzeuge in der Einsatzumgebung, der Automatisierung aller einzelnen Teilschritte im Prozessablauf und der Integration der einzelnen Komponenten zum Gesamtsystem.

Bei der Installation musste geplant werden, wie die Verteilung des Systems auf die vorhandene Hardwarelandschaft des Unternehmens sinnvoll umzusetzen wäre. Jenkins und Sonar wurden auf einem Tomcat Applikationsserver installiert, anstatt ihre eingebetteten Webcontainer zu verwenden oder sie als Dienste zu installieren. Dies geschah, damit sie Betriebssystem-unabhängig und zentral verwaltet werden können. GUI-Tests sollen mit virtuellen Testrechnern auf einem extra dafür vorgesehenen Server durchgeführt werden, wobei die Virtualisierung als Nebeneffekt die Energiekosten senkt. Somit können auch gleich mehrere gleichkonfigurierte Rechner einfach angelegt werden, um beispielsweise Testfälle parallel durchzuführen. Zum Austausch von Daten und gemeinsam genutzter Software und Bibliotheken wird ein Fileserver verwendet.

Ein Konfigurationsaufwand ist hauptsächlich bei Jenkins und Sonar angefallen. In Jenkins mussten die entsprechenden Projekte für die einzelnen Testschritte angelegt werden. Hierbei war einige Einarbeitungszeit notwendig, um die Funktionsweisen und die möglichen Projektstrukturen kennenzulernen. Es hat sich als ein Problem herausgestellt, das gewünschte Auslösen eines neuen Builds bei Änderungen in SVN-Repositories umzusetzen, da admileo für jedes Modul ein eigenes Repository verwendet. Jenkins kann zwar in aktuellen Versionen mit mehreren Repositories gleichzeitig umgehen, jedoch überprüft es diese durch Polling, was bei ca. 50 Repositories für keine attraktive Lösung befunden wurde. Eine mögliche Alternative ist das Verwenden von „Hook-Skripten“, die nach dem Commit auf dem SVN-Server ausgeführt werden und einen neuen Build anstoßen. Bei Sonar musste eine sinnvolle Kombination aus den verfügbaren Überprüfungsregeln und Metriken gefunden werden, die den Konventionen und Erwartungen der Entwickler und Geschäftsführer entspricht. Darüber hinaus musste Sonar die Projektstruktur von admileo beigebracht werden. Dies ist grundsätzlich über Kommandozeilenparameter, Ant oder Maven, machbar. Allerdings unterstützen nur die beiden letzteren die Multi-Modul Projekte dahingehend, dass sie entsprechend der Paketstruktur navigierbar sind, anstatt alle Klassen in einer flachen Struktur abzubilden. Ein Problem dabei ist jedoch, dass für jedes Modul und Paket, das gesondert auswertbar sein soll, ein eigenes Ant-Skript angelegt werden muss, in dem ein eindeutiger Schlüssel und Pfadangaben zu Quellcodes und Klassen vorhanden sind. Bei einem großen Projekt wie admileo würde dies einiges an Aufwand bedeuten, weshalb ein Jython Skript erstellt wurde, das die Projektstruktur automatisiert auswertet und passende Ant-Skripte generiert.

Die Abhängigkeiten zwischen einzelnen Schritten im Prozess sind grundsätzlich nicht sonderlich hoch, von der Voraussetzung abgesehen, dass der Build eine lauffähige Version von admileo liefern muss. Ansonsten sollten die Testergebnisse alle in einer von Sonar unterstützten Formatierung bereitliegen, so dass sie alle ausgewertet werden können. Dies gestaltete sich weitestgehend problemlos, da Sonar JUnit-formatierte Ergebnisse sowie JaCoCo Messdaten unterstützt und QF-Test auf Wunsch seine Ergebnisse auch im JUnit-Format generieren kann.

Bei der Evaluation von QF-Test ist festgestellt worden, dass das Werkzeug mit einigen Situationen wie „Drag and Drop“ nicht ohne weiteres zurechtkommt. Elemente von admileo, die nicht in Java implementiert sind wie beispielsweise der Update- und Launch-Mechanismus, kann es nicht bedienen. Dies war auch bei anderen Werkzeugen der Fall, die nach einem ähnlichen Prinzip arbeiten. Zur Abhilfe wurde auf den Testrechnern das zusätzliche, technologieunabhängige Werkzeug Sikuli eingesetzt, das anhand von Bildverarbeitung GUI-Elemente in Screenshots wiedererkennt und bedient. Der Nachteil an diesem Werkzeug ist, dass es trotz einstellbarer Toleranzgrenzen bei der Erkennung stark vom gleichbleibenden Aussehen der GUI abhängig ist. Aus diesem Grund wurde beschlossen, es möglichst sparsam einzusetzen, um ein häufiges Anpassen der Testfälle bei Änderungen an admileo zu vermeiden. Damit die Testergebnisse einheitlich und zusammen mit den anderen Testergebnissen verwendbar bleiben, wird Sikuli aus QF-Test Testfällen heraus gestartet. Dies ist problemlos möglich, da QF-Test Kommandozeilenbefehle und Skripte ausführen und anhand ihrer Rückgabewerte den Ausgang der Sikuli Tests auswerten kann.

Nach der Evaluation der prototypischen Implementierung des beschriebenen CI-Systems wurde eine Umsetzung für den produktiven Einsatz beschlossen. In die Prozessverbesserung sind Ressourcen eingeflossen. Ein ROI wird nach vier Testphasen (Bei 2 Releases pro Jahr: 2 Jahre) erwartet, eine Anzahl, die im typischen Rahmen der häufiger genannten drei bis fünf benötigten Wiederholungen liegt. Ein wichtigeres Ziel für Archimedon als das direkte ROI ist jedoch die Verbesserung der Softwarequalität. Es muss positiv bewertet werden, dass jetzt häufiger getestet werden kann. Kritisch ist zu beachten, dass Änderungen an der GUI mehr Nacharbeiten in den Testfällen benötigen, als früher.

5 Analyse der Erfolgsfaktoren

Naiv betrachtet, kann man aus den vorherigen Kapiteln ableiten, dass man alle KMUs zur Testautomatisierung als offensichtlich schnell ROI bringenden Ansatz auffordern muss. Kritischer betrachtet, handelt es sich sicherlich um eine Maßnahme mit großem Potenzial, allerdings trafen in dieser Fallstudie mehrere Randbedingungen zu, die den Erfolg gesichert haben, die kurz skizziert wie folgt aussehen.

- klar formulierte Anforderungsspezifikation: Durch strukturiert erfasste Anforderungen wird es überhaupt erst möglich, zu überprüfen, ob das gewünschte System entwickelt wird.
- wiederholbar spezifizierte Systemtests: Durch die strukturierte textuelle Ausformulierung von Systemtests mit den Vorbedingungen, der Ausführung und den erwarteten Ergebnissen wurde eine gute Grundlage zur Automatisierung geschaffen.
- Client-Server-System: Durch die klassische Architektur kann diese unmittelbar auch als Testarchitektur umgesetzt werden, wodurch realistische Testszenarien ermöglicht werden und das Zusammenspiel zwischen Client und Server analysierbar wird.
- modular aufgebaute Software mit meist unveränderten GUI-Anteilen: Durch die modulare Software-Architektur ist gewährleistet, dass neue Komponenten typischerweise nicht zu vielen kleinen Änderungen an unterschiedlichsten Stellen führen, die den Testaufwand deutlich erhöhen.

- Erfahrung in automatisierten Prozessen: Durch die Nutzung automatisierter Build-Prozesse, z. B. mit den Werkzeugen Ant und Jenkins, ist die Ergänzung dieser Prozesse ein kleinerer Schritt, als zunächst ein Build Management einzuführen und zu ergänzen.
- nutzbare Werkzeuge zur Testautomatisierung: Durch die detaillierte Analyse konnten Werkzeuge gefunden werden, die eine Testautomatisierung der GUI-Tests überhaupt erst ermöglichen. Der Einsatz von Swing trug dazu bei, dass es überhaupt passende Werkzeuge gibt.
- akademisch qualifizierte Software-Entwickler: Die systematische Ausbildung der Entwickler und ihre bisherigen Erfahrungen ermöglichten es, dass eine Akzeptanz für schrittweise Prozessänderungen vorliegt.

Man kann sicherlich folgern, dass unter exakt gleichen Randbedingungen die Automatisierung der GUI-Tests eine höchstwahrscheinlich gewinnbringende Maßnahme ist. Allerdings sind die Software-Entwicklungsprozesse und die entstehenden Produkte bei genauer Analyse so unterschiedlich, dass es unwahrscheinlich ist, eine identische Situation vorzufinden. Die Analyse der genannten Randbedingungen kann aber wesentlich bei der Auswahl der zuerst umzusetzenden Optimierungsmaßnahme helfen.

6 Zusammenfassung und Ausblick

Die vorherigen Kapitel zeigen ein erfolgreiches Beispiel, wie durch Verbesserungen eines aktuell gelebten Entwicklungsprozesses eine Effizienzsteigerung in einem Software-herstellenden KMU durch Automatisierung von GUI-Tests erreicht wird. Weiterhin wird deutlich, dass der Weg von der Idee zur Automatisierung bis hin zur Umsetzung nicht trivial ist, genau geplant werden muss und es durchaus Faktoren geben kann, die zur Nichtumsetzung der Idee führen können. Weiterhin soll der beschriebene Automatisierungsansatz noch weiter entwickelt werden, so kann durch eine Parallelisierung die Ausführungszeit reduziert, können Testergebnisse noch weiter aufbereitet und kann eine visuelle Komponente zur Auswahl bestimmter Teiltestgruppen umgesetzt werden.

Nicht betrachtet wurde aus Platzgründen eine Diskussion über notwendige Ausbildungsmaßnahmen von Mitarbeitern in KMUs. Dies umschließt die Frage, wann ein Selbststudium ausreicht und wann Schulungsmaßnahmen oder die temporäre Begleitung der Entwicklung durch einen Coach sinnvoll sind. Werden z. B. Mitarbeiter neu in die Qualitätssicherung eingearbeitet, ist eine ISTQB-Schulung [@ist] unterstützt durch Literatur [SL10] und evtl. durch Coaching sehr zu empfehlen.

Neben der offenen Frage nach der Ausbildung besteht eine wichtige, hier andiskutierte, Aufgabe darin, die richtigen Maßnahmen in KMUs zu treffen, um ihnen einen langfristigen Erfolg zu ermöglichen. Im vorherigen Kapitel wurden erste Randbedingungen andiskutiert, die bei der Auswahl geeigneter Maßnahmen beachtet werden müssen. Dabei stellt sich die offene Frage, welche Randbedingungen zu welcher Maßnahme führen sollen. Natürlich ist die Liste aus dem vorherigen Kapitel nicht vollständig; viele weitere Randbedingungen, wie das Potenzial und die Fähigkeiten der Konkurrenten und die besonderen Erwartungen von Kernkunden, können Einfluss nehmen. Auch die Anzahl potenziell interessanter Maßnahmen geht weit über die bisher andiskutierten hinaus und umfasst z. B. präzisierte Aufwandsschätz- und Risikomanagement-Prozesse, beinhaltet

aber auch Ansätze zur modellgetriebenen Entwicklung oder auch den Einsatz formaler Methoden [Kl09]. Dieses Netzwerk von Randbedingungen und Maßnahmen soll an der Hochschule Osnabrück im Rahmen eines Forschungsschwerpunkts „Long Term Software-Engineering“ für KMU entschlüsselt werden.

Vereinfacht kann man für KMU folgern, dass nur durch einen hochwertigen, auf das Unternehmen passenden Entwicklungsprozess langlebige Software entstehen kann, die schon während ihrer Erstellung und Nutzung Aspekte der Nachhaltigkeit berücksichtigt.

Literaturverzeichnis

Letzte Aufrufe der genannten Web-Seiten stammen vom 6.8.2012.

[@ant] Apache Ant, http://ant.apache.org/
[@arc] Archimedon, http://www.archimedon.de/
[@hyp] Hyper-V, http://www.microsoft.com/de-de/server/hyper-v-server/default.aspx
[@ist] ISTQB International Software Testing Qualifications Board, http://www.istqb.org/
[@jac] JaCoCo Java Code Coverage Library, http://www.eclemma.org/jacoco/
[@jen] Jenkins, http://jenkins-ci.org/
[@jun] JUnit, http://kentbeck.github.com/junit/javadoc/latest/
[@pos] PostgreSQL, http://www.postgresql.org/
[@qfs] Quality First Software, http://www.qfs.de/de/qftest/index.html
[@son] Sonar, http://www.sonarsource.org/
[@sik] Project Sikuli, http://sikuli.org/
[@sub] Subversion, http://subversion.tigris.org/
[@tom] Apache Tomcat, http://tomcat.apache.org/
[@wq] Werkzeuge für die Qualitätssicherung, http://home.edvsz.hs-osnabrueck.de/skleuker/CSI/Werkzeuge/kombiQuWerkzeuge.html

[Ba00] Balzert, H.: Lehrbuch der Software-Technik: Software-Entwicklung, 2. Auflage, Spektrum Akademischer Verlag, Heidelberg Berlin Oxford, 2000

[He12] Heidt, A.: Konzeption und Realisierung einer automatisierten Testumgebung in einem Continuous Integration Prozess für admileo, Bachelorarbeit, HS Osnabrück, 2012

[Hu09] Humble, J.; Farley, D.: Continuous Delivery: Reliable Software Releases Through Build, Test, and Deployment Automation, 1. Auflage, Addison-Wesley Longman, Amsterdam, 2010

[Kl09] Kleuker, S.: Formale Modelle der Softwareentwicklung, Vieweg+Teubner, Wiesbaden, 2009

[Kl11] Kleuker, S.: Grundkurs Software-Engineering mit UML, 2. Auflage, Vieweg+Teubner, Wiesbaden, 2011

[Li09] P. Liggesmeyer, Software-Qualität. Testen, Analysieren und Verifizieren von Software, 2. Auflage, Spektrum Akademischer Verlag, Heidelberg Berlin Oxford, 2009

[LL06] Ludewig, J.; Lichter, H.: Software Engineering – Grundlagen, Menschen, Prozesse, Techniken. dpunkt.verlag, Heidelberg, 2006

[SL10] Spillner, A.; Linz, T.: Basiswissen Softwaretest, 4. Auflage, Dpunkt Verlag, Heidelberg, 2010

[SW02] Sneed H. M.; Winter, M.: Testen objektorientierter Software, Hanser, München Wien, 2002

[Wi05] Winter, M.: Methodische objektorientierte Softwareentwicklung, Dpunkt Verlag, Heidelberg, 2005

Entwurf eines Quellcode basierten Qualitätsmodells für die Softwarewartung

Meik Teßmer

Bereich Computergestützte Methoden
Fakultät für Wirtschaftswissenschaften
Universität Bielefeld
Universitätsstraße 25
33615 Bielefeld
mtessmer@wiwi.uni-bielefeld.de

Abstract: Die Wartbarkeit eines Softwaresystems ist ein wichtiger Faktor für seinen langfristigen Einsatz. In diesem Beitrag wird ein Qualitätsmodell auf Quellcode-Basis entworfen, das den Wartbarkeitsbegriff konkretisiert und messbar macht. Ziel ist die Untersuchung der Qualität eines an der Universität entwickelten großen Systems zur Prüfungsverwaltung.

1 Nachhaltigkeit als Qualitätsaspekt von Quellcode

Der langfristig aufrecht erhaltene Betrieb eines großen Softwaresystems führt zu zwei Erkenntnissen: (1) Wartung und Evolution sind Teil des sich über den Zeitpunkt der Auslieferung hinaus erstreckenden Entwicklungsprozesses, denn ohne eine kontinuierliche Anpassung verliert das System für den Anwender nach und nach den Nutzen (s. [BL76, Leh+97]). (2) Die vorgenommenen Änderungen beeinflussen Umfang und Qualität des Quellcodes des Systems (s. [BL76, Eic+98, Leh+97, MWT94]).

Vergegenwärtigt man sich die Kosten der Wartungsarbeiten, die auf 70% und mehr der Gesamtkosten des Systems geschätzt werden (s. [Boe79, Ede93, Erl00]), ist die Berücksichtigung des Faktors Nachhaltigkeit bei der Entwicklung und Wartung für das betreibende Unternehmen eine ökonomische Notwendigkeit. Besonderes Augenmerk muss dabei auf den Quellcode gerichtet werden. Er ist im Gegensatz zur Softwaredokumentation die einzige *zuverlässige* Informationsquelle (s. [DP09, Mar09]), die alle (Struktur)-Entscheidungen der an der Entwicklung Beteiligten vereint.

Nach Müller, Wong und Tilly führen vorgenommene Änderungen wie Erweiterungen oder Updates bei einem durchschnittlichen Fortune 100-Unternehmen zu einer jährlichen Zunahme des Quellcode-Umfangs von ca. 10% (s. [MWT94]). Zugleich wird 60% der für die Wartung aufgewendeten Zeit allein für die Suche nach den relevanten Quellcode-Stellen benötigt (s. [Ede93]). Dabei erschwert nicht nur der Umfang die Analyse und das Verständnis des Quellcodes. Mit fortschreitender Wartung werden die Systeme immer

komplexer, während die Code-Qualität gleichzeitig abnimmt (s. [Leh+97]). Es kommt zu einem Verfall der Strukturen, der auch als *Code Decay* oder *Software-Entropie* bezeichnet wird (s. [Eic+98, HT03]). Ohne entsprechende Gegenmaßen besteht das Risiko, dass jede Fehlerkorrektur selbst wieder zu neuen Fehlern führt und das System schließlich nicht mehr wartbar ist (s. [BL76]).

Sinnvolle Gegenmaßnahmen erfordern die Entwicklung eines geeigneten *Qualitätsmodells*. Zusammen mit einem projektspezifischen *Softwaremodell* ermöglicht es die Ermittlung und Bewertung des Ist-Zustands der für die Wartung relevanten Strukturen im Quellcode. Aus den Analyseergebnissen lassen sich dann Gegenmaßnahmen ableiten, die auf die Verbesserung der Wartbarkeit abzielen. Ein solches Qualitätsmodell hat die Aufgabe, das allgemeine und unpräzise Verständnis von „Qualität“ durch eine Taxonomie von Einzelfaktoren operationalisierbar zu machen. Dazu werden Qualitätsunterbegriffe abgeleitet und diese weiter zu Indikatoren verfeinert, um sie einem Messinstrumentarium zugänglich zu machen. Die Interpretation der Messergebnisse erlaubt schließlich, Rückschlüsse auf die Gesamtqualität des Systems zu ziehen. Wartbarkeit als relevanter Aspekt der Nachhaltigkeit kann damit in die Qualitätssicherung integriert werden und den langfristigen Betrieb des Systems sicherstellen. Zusätzlich erreicht man eine Vereinheitlichung der Vorstellung von Qualität und macht sie dadurch für Entwickler und Management kommunizierbar (s. [Wal01]).

Die Entwicklung eines Qualitätsmodells wird von verschiedenen Fragen geleitet: Was ist „guter“ Code? Wodurch zeichnet sich eine wartbare Struktur aus? Wie stehen die verschiedenen Strukturmerkmale in Beziehung und welche Auswirkungen hat das auf die Wartbarkeit? Welche Grenzwerte sollten angelegt werden? Welche spezifischen Strukturierungsmöglichkeiten der verwendeten Programmiersprache haben Einfluss auf die Wartbarkeit? Erste Hinweise auf Antworten zu diesen Fragen liefern die verschiedenen Methoden zur Softwarevermessung durch *Metriken*. In Anlehnung an das Messen, wie es bspw. in der Physik stattfindet, wird versucht, durch die Quantifizierung qualitativer Aspekte Aussagen zu bestimmten Qualitäten von Softwaresystemen zu machen. Die Abbildung der Messwerte soll Rückschlüsse auf bestimmte Eigenschaften des Systems erlauben. Problematisch ist jedoch die Bestimmung der Grenzwerte, da sich bspw. absolute Zahlen für gute oder schlechte Größen von Klassen, Packages, Subsystemen oder Schichten nicht angeben lassen (s. [RL04]). Erschwerend kommt die nicht standardisierte Interpretation der Messergebnisse hinzu (s. [And+04]). Gründe dafür sind u.a. die unklare Definition von Begriffen wie Wartbarkeit. Zum Teil besteht aber auch Zweifel am Nutzen einzelner Metriken. Eine Untersuchung von Aggarwal et al. hat ergeben, dass von 14 untersuchten Metriken nur sechs hinreichend gute eigenständige Aussagen lieferten, während die anderen Metriken entweder Teilmengen dieser sechs bildeten oder dieselbe Information auf andere Weise darstellten (s. [Agg+06]).

Trotz der schon angeführten Probleme unterstützen die Metrik-Suiten von McCabe (s. [McC76]), Halstead (s. [Hal77]), Chidamber und Kemerer (s. [CK91]), das MOOD-Set von Harrison et al. (s. [HCN98]) sowie Fokussierungen und Erweiterungen verschiedener qualitätsorientierter Betrachtungen des Codes und erlauben eine Lokalisierung wartungsbedürftiger Code-Bereiche und die Abschätzung des Wartungsumfangs (s. dazu u. a. [BBM96, BDW99, BMB99, Cos+05, FN01, Gos+03, HS01, LC01, WYF03]).

Weitere Hinweise auf relevante Strukturen und ihre Bedeutung für die Wartbarkeit lassen sich aus den Arbeiten zu Entwurfsmustern ableiten (s. [Gam+96, Lar04]). Sie bieten Entwicklern zu häufig wiederkehrenden Problemstellungen bewährte Code-Strukturen als Lösungsvorschlag an und dienen als „Best Practices"-Leitfaden. Ebenso hilfreich sind die sog. *Code Smells*. Sie weisen auf kritische Stellen im Code hin, die durch Refactoring-Maßnahmen behoben werden können (s. [Fow99]). Solche Refactorings können auch auf der Architekturebene durchgeführt werden, wenn sog. *Architektur-Smells* vorliegen (s. [RL04]). Sie korrigieren dann keine Klassen(-verbände), sondern wirken auf der Ebene der Komponenten. Auf Entwurfsmuster ausgerichtete Refactorings, *Refactoring to Patterns*, sind eine weitere Möglichkeit, die Evolution von Systemen zu unterstützen (s. [Ker06]). Das *Clean Code*-Prinzip von Martin hingegen setzt auf die Intuition des Entwicklers als maßgebliches Qualitätskriterium: „Sauber" ist Quellcode genau dann, wenn er mit wenig Aufwand in kurzer Zeit richtig verstanden werden kann (s. [Mar09]). Diese Richtlinie lässt sich jedoch nicht direkt in ein Qualitätsmodell übertragen: Wo bei Entwurfsmustern und Code Smells mehr oder weniger eindeutige Qualitätsmerkmale identifiziert werden können, lässt sich die Intuition des Entwicklers nicht auf eindeutige Merkmale abbilden und so quantifizieren. Mit Hilfe von Ersatzmerkmalen, sog. *Prädiktoren* (s. [10e]), kann dennoch versucht werden, Eigenschaften wie Lesbarkeit und Verständlichkeit messbar zu machen und in ein Qualitätsmodell zu integrieren. Auf diese Weise lassen sich auch andere ähnlich unpräzise Hinweise wie bspw. Faustregeln zur Entwicklung einer guten Architektur (s. u. a. [BCK98]) messbar machen.

Der nächste Abschnitt befasst sich mit der Sammlung von Informationen, die für die Entwicklung eines Qualitätsmodells notwendig sind. Dabei werden unterschiedliche Quellen konsultiert: Der Standard ISO 9126, Veröffentlichungen zu Qualitätsmodellen sowie Arbeiten zur Softwarevermessung und zur Softwarearchitektur. Die Entwicklung zielt darauf ab, ein großes Softwaresystem der Universität Bielefeld, das BIS[1], zu analysieren und hinsichtlich seiner Wartbarkeit zu bewerten. Im dann folgenden Abschnitt wird das Qualitätsmodell entworfen und mit einem Softwaremodell konkretisiert. Die Untersuchung des BIS und abschließende Betrachtungen zu Möglichkeiten und Grenzen dieses Bewertungsansatzes bilden den letzten Abschnitt.

2 Vorüberlegungen

2.1 Aufbau eines Qualitätsmodells

Zunächst stellt sich die Frage, wie ein Qualitätsmodell entworfen sein sollte. Der ISO-Standard 9126 (mittlerweile von ISO 25000 abgelöst) beschreibt in vier Teilen ein Qualitätsmodell für die Bewertung der *Produktqualität* und der *Gebrauchsqualität*. Der erste Teil (ISO 9126-1) beschreibt die Zerlegung des Qualitätsbegriffs, der zweite und dritte

[1] Das BIS ist ein Web-basiertes Informationssystem für die Universität Bielefeld. Es besteht aus dem elektronischen kommentierten Vorlesungsverzeichnis, der Prüfungsverwaltung und dem Personen- und Einrichtungsverzeichnis. Sämtliche BIS-Anwendungen werden hausintern entwickelt und betrieben.

spezifiziert Kenngrößen für die externe bzw. interne Produktqualität. Im letzten Teil finden sich Kenngrößen zur Gebrauchsqualität.

Der Standard schlägt folgende Zerlegungssystematik vor:

Abbildung 5: Zerlegungssystematik nach ISO 9126-1

Die Bewertung der Wartbarkeit betrifft die *interne* Produktqualität, die der ISO-Standard zusammen mit der externen Produktqualität wie folgt untergliedert:

- Funktionalität: angebotene Funktionalität (Eignung, Genauigkeit, Interoperabilität, Sicherheit, Einhalten von Standards)
- Zuverlässigkeit: Leistungsniveau (Reife, Fehlertoleranz, Wiederherstellbarkeit, Einhalten von Standards)
- Nutzbarkeit: Einsatz des Systems aus Sicht des Anwenders (Verständlichkeit, Erlernbarkeit, Betriebsfähigkeit, Attraktivität, Einhalten von Standards)
- Effizienz: technische Aspekte des Einsatzes (Zeitverhalten, Ressourcenverwendung, Effizienzstandards)
- Wartbarkeit: Änderungen und Erweiterungen des Systems (Analysierbarkeit, Änderbarkeit, Stabilität, Testbarkeit, Einhalten von Standards)
- Portabilität: Übertragen auf andere Hardware-/Software-Plattformen (Anpassungsfähigkeit, Installierbarkeit, Koexistenz, Ersetzbarkeit, Portabilitätsstandards)

Die Wartbarkeit eines Systems wird vom Standard als eine Untereigenschaft der internen Produktqualität definiert. Angesichts der hohen Änderungsrate sowohl von Software- als auch Hardware-Plattformen erfordert der langfristige Betrieb eines Softwaresystems auch die Berücksichtigung von Portierungsfragen, weshalb diese Untereigenschaft des Modells ebenfalls mit in die Überlegungen einbezogen werden sollte.

Die in ISO 9126-3 vorgeschlagenen Kenngrößen zur Ermittlung von internen Merkmalen, wie sie in ISO 9126-1 beschrieben werden, beschränken sich auf Vorschläge zum Zählen von Funktionen, Kommentarzeilen, Logging-Funktionen usw. und liefern damit nur Hinweise auf mögliche Ausgestaltungen. Konkrete Bezüge zu wartungsrelevanten Strukturen im Quellcode fehlen und die Verwendung bekannter Metriken, die bei der

Beschreibung der Kenngrößen zu erwarten wäre, findet nur am Rande im Anhang E Erwähnung.

Die vom Standard verwendete Zerlegungssystematik basiert auf der Factor-Criteria-Metrics-Methode (FCM) von Cavano und McCall (s. [CM78]). Auf der obersten Ebene finden sich die Beschreibungen der jeweiligen Produktqualitäten (die Qualitätsfaktoren), die auf der nächsttieferen Ebene in ein oder mehrere Attribute des Systems aufgelöst werden (Qualitätskriterien und Unterkriterien). Diese wiederum können durch Metriken zum Quellcode in Beziehung gesetzt und bspw. in Zahlform erfasst werden, welche auf Programmierempfehlungen aus der Fachliteratur basieren. Die im FCM beschriebene Zerlegung besitzt nur drei Ebenen, die Erfahrung zeigt aber, dass zusätzliche Ebenen sinnvoll sind. Entsprechend wurde die Zerlegungssystematik des ISO-Standards entworfen. In [10d] wird bspw. eine weitere Schicht mit Entwurfsregeln vorgeschlagen, die Angaben zur Merkmalsausprägung macht (z. B. „hohe Kohäsion“).

Der Katalog der Eigenschaften aus ISO 9126, ihre Zerlegungssystematik und der Versuch, dazu konkrete Kenngrößen zu definieren, sind nachvollziehbar und decken sich vom Ansatz her mit den Vorschlägen anderer Autoren (s. bspw. [Bot+04, Wal01]). Dementsprechend kann dieser Teil des Standards als Vorlage für die Definition eines Qualitätsmodells dienen. Bevor jedoch ein vollständiges Modell entwickelt werden kann, müssen zunächst die relevanten Artefakte und Strukturen identifiziert werden, die die Wartbarkeit beeinflussen können, um sie anschließend als Merkmal zu integrieren.

2.2 Wartungsrelevante Merkmale

Die Programmstruktur und die Modularität sind wichtige Aspekte bei der Entwicklung zuverlässiger Softwaresysteme. Die Relevanz dieser Aspekte wurde u. a. von Parnas (s. [Par75]) beschrieben. Das erste und auch heute noch sehr verbreitete Maß ist Lines of Code (LOC). Dieses 1955 vorgestellte Volumenmaß zählt die Anweisungen im Quellcode und erfasst so die Größe von Systemen und Systembestandteilen. Zudem gibt es eine starke Korrelation zu anderen Maßen (s. [SI93]).

Eine populäre Reihe von Komplexitätsmaßen wurde 1976 von McCabe aus der Graphentheorie abgeleitet (s. [McC76]). Halstead prägte in [Hal77] den Begriff „Software Science“ mit dem Ziel, wissenschaftliche Methoden auf Strukturen und Eigenschaften von Programmen anzuwenden. Seine Maße und die von McCabe gehören bis heute zu den bekanntesten Maßen überhaupt. Yourdon und Constantine befassten sich 1979 mit dem damals populären Paradigma der Strukturierten Programmierung (s. [YC79]). Sie stellten dabei auch Überlegungen zum Entwurf von Systemen an. Aufbauend auf den Arbeiten von Myers (s. [Mye75]) untersuchten sie die verschiedenen Strukturierungsmöglichkeiten im Hinblick auf ihre Auswirkungen bei Änderungen. Sie entwarfen dazu Richtlinien zur Modularisierung, die sich an der Kopplung und Kohäsion von Bausteinen orientieren. Henry und Kafura versuchten 1981, mit Hilfe von Fan-In/Fan-Out die Komplexität von Prozeduren und Modulen zu bestimmen und sie zu Änderungen in Beziehung zu setzen (s. [HK81]).

Mit der Einführung der Objektorientierung mussten Maße angepasst und neue entwickelt werden, um die neuen Strukturierungsmöglichkeiten entsprechend zu berücksichtigen. Lieberherr und Holland übertrugen im Rahmen des Demeter-Projekts die Ideen der Kapselung und Kohäsion auf die Objektorientierung und prägten den Begriff des „shy code“ (s. [LH89]). Sie versuchten, den Impact einer Änderung so effektiv wie möglich zu begrenzen. Chidamber und Kemerer veröffentlichten 1991 eine Reihe bis heute verbreiteter Maße für die Vermessung objektorientierter Systeme (s. [CK91]). 1993 entwarf Lorenz elf Maße, mit deren Hilfe das Design eines OO-Systems bewertet werden können sollte. Dazu machte er auch Vorschläge zu Grenzwerten für die Sprachen Smalltalk und C++ (s. [LK94]). In dieselbe Richtung arbeiteten auch Abreu und Carapuça mit ihrem MOOD-Metrik-Set (s. [AC94]).

Alle bisher aufgeführten Metriken untersuchen mehr oder weniger feingranulare Strukturen wie Funktionen, Methoden oder Klassen. Die Organisation großer Softwaresysteme erfordert jedoch eine Strukturierung auf höheren Abstraktionsebenen wie sie bspw. Module oder Pakete anbieten. Eine Qualitätsbewertung muss daher auch diese Ebene berücksichtigen. Martin überträgt dazu das Kopplungs- und Kohäsionskonzept der Klassen auf die Ebene der Pakete, indem er sechs Entwurfsprinzipien beschreibt (s. [Mar02]). Diese greifen z. T. Aspekte aus anderen Arbeiten auf, bspw. das Open-Close-Prinzip von Meyer (s. [Mey97]) oder das Liskovsche Ersetzungsprinzip (s. [Lis87]). Neben der Übertragung der Konzepte ist sein Ansatz, die positionale Stabilität eines Pakets zu ermitteln, sowie der Versuch, seine Abstraktheit zu berechnen, ein wichtiger Beitrag zur Thematik.

Lanza und Marinescu versuchen mit einem auf logischen UND- und ODER-Operationen basierenden Kompositionsmechanismus, die durch Filter auf die relevanten Werte reduzierten Messergebnisse zu kombinieren. Auf diese Weise können auch komplexere Design-Regeln untersucht werden (s. [LM06]). Damit eröffnen sich zwei Wege, um ein System mit solchen Detektoren zu analysieren: (1) Man versucht, „gutes Design“ mit Regeln und heuristischen Daten zu definieren, oder (2) Symptome „schlechten Designs“ zu finden und damit die Artefakte, die einer Überarbeitung bedürfen. Diesen letzten Ansatz verfolgt auch Fowler in mit den sog. Code Smells. Sie weisen auf Code-Abschnitte hin, die strukturell verbessert werden müssen (s. [Fow99]).

Insgesamt gibt es für die Bewertung von Strukturen auf den unteren Abstraktionsebenen eine ganze Reihe von Metriken, die jeweils bestimmte Aspekte der Strukturierung erfassen können. Die Frage ist, welche Merkmale für die Beurteilung der Wartbarkeit relevant sind. Laut Bass et al. zeigt sich die Güte der Änderbarkeit durch den *Impact*, der von einer Änderung ausgelöst wird (s. [BCK98]). Je geringer der Impact, desto größer die *Lokalität* der Änderung. Sie lässt sich zurückführen auf die *Kopplung* und *Kohäsion* von Artefakten der jeweiligen Abstraktionsebene und zielt damit auf die *Modularität* des Systems.

Die größten Risiken für die Strukturen eines Systems sowohl im Großen als auch im Kleinen sind nach Perry und Wolf die *Architekturdrift* (Verwässerung der Struktur durch Unkenntnis) und die *Architekturerosion* (massive Verletzung von Strukturen, bspw. durch Umgehung einer Schicht) (s. [PW92]). Um diese Risiken zu vermeiden bzw. zu

minimieren, muss der Entwickler sich ihrer bewusst werden. D .h., er muss in der Lage sein, die architekturrelevanten Artefakte und ihre Verbindungen entweder der zugehörigen Dokumentation zu entnehmen oder aus dem Quellcode ermitteln zu können. Damit spielt die Lesbarkeit des Quellcodes eine wichtige Rolle: Geringe Komplexität der Strukturen, überschaubare Größen und möglichst explizite Verbindungen der Artefakte [CY90, Szy99]. Das entspricht dem Clean Code-Prinzip von Martin und den Code Smells von Fowler: Der Entwickler muss ausgehend vom Quellcode in möglichst kurzer Zeit ein korrektes mentales Modell des Systems aufbauen und dabei die Architektur(en), die Regeln des Architekturstils und die daran geknüpften Bedingungen erfassen. Daraus lässt sich für eine Architektur ableiten:

> *Eine gute/verständliche Architektur ist hoch modular, besitzt eine geringe Komplexität bei gleichzeitig geringer Kopplung zwischen und hoher Kohäsion in den Komponenten. Sie folgt außerdem einer zentralen Strukturierungsidee.*

Hinsichtlich der Bewertung der Lesbarkeit, Komplexität und Größe darf jedoch der menschliche Faktor nicht vergessen werden, der bei der Entwicklung eines Systems und später bei der Wartung immer eine Rolle spielt. Wallmüller weist mit Bezug auf Pirsig darauf hin, dass Software-Produkte zu einem erheblichen Teil das Ergebnis intuitiver und kreativer Arbeit sind (s. [Wal01]). Er unterscheidet messbare Qualität in Form von Kenngrößen von der ebenfalls als qualitativ relevant anzusehenden kreativen Tätigkeit, die sich jedoch einer Messung entzieht. Damit erhält der Qualitätsbegriff neben einer logisch/rationalen Seite zusätzlich eine subjektive, vom direkten Wahrnehmen und Empfinden geprägte Seite, die sich durch Begriffe wie Originalität, Einfachheit und Mächtigkeit beschreiben lässt, aber sich nicht unbedingt in Form von Merkmalen niederschlägt. Wie bei den angesprochenen Metriken generell gilt hier im Besonderen, dass etwaige Grenzwerte je nach Projekt und Entwickler u. U. sehr unterschiedlich ausfallen können und ein Vergleich auch ähnlicher Projekte aufgrund dieser Subjektivität nur bedingt sinnvoll ist.

3 Entwurf des Qualitätsmodells

Wallmüller schlägt für die Entwicklung und Anwendung eines Qualitätsmodells sieben Schritte vor (s. [Wal01]):

1. Definieren der Messziele
2. Ableiten der Messaufgaben aus den Messzielen
3. Bestimmung der Messobjekte
4. Festlegen der Messgröße und Messeinheit
5. Zuordnung der Messmethoden/-werkzeuge zu den Messobjekten/-größen

6. Ermitteln der Messwerte

7. Interpretation der Messwerte

Auf diese Weise soll verdeutlicht werden, *warum* gemessen wird, denn die Interpretation der Messwerte basiert immer auf einer bestimmten Hypothese. Zunächst stellt sich also die Frage, welche Qualitätsfaktoren durch eine Messung bewertet werden sollen. Anschließend müssen Hypothesen aufgestellt werden, welche Merkmale – bzw. an ihrer Stelle die Prädiktoren – als Konkretisierung geeignet sind. Handelt es sich um die Bewertung von systeminternen Qualitätseigenschaften wie die Wartbarkeit, spielt auch die verwendete Programmiersprache eine Rolle bei der Auswahl der Prädiktoren. An dieser Stelle ist ein Softwaremodell notwendig, welches einerseits die Menge der möglichen Prädiktoren beschreibt, die die eingesetzte Programmiersprache bietet, und das andererseits auch Messobjekte anbietet. Dabei muss das Softwaremodell nicht auf die Möglichkeiten der Programmiersprache beschränkt bleiben, sondern kann darüber hinaus auch weitere Abstraktionsstufen abbilden, die bspw. die Architektur des Systems betreffen. Das Softwaremodell beeinflusst auch die Messmethoden und Werkzeuge, die für die Ermittlung der Messwerte in Frage kommen:

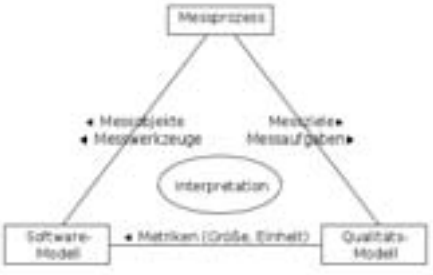


Abbildung 6: Beziehung von Messprozess, Software- und Qualitätsmodell

3.1 Begriffszerlegung

Aus den Vorüberlegungen und im Hinblick auf die geplante Untersuchung des BIS wird zunächst folgendes Messziel abgeleitet: *Das Messziel umfasst die Untersuchung und Bewertung der internen Qualität eines Softwaresystems in Form der Qualitätseigenschaften* Wartbarkeit *und* Portabilität.

Die Zerlegung orientiert sich an der im Standard ISO 9126 vorgeschlagenen Zerlegungssystematik:

Abbildung 7: Zerlegungssystematik für das Qualitätsmodell

Wartbarkeit wird vom ISO-Standard in die Untereigenschaften *Analysierbarkeit*, *Änderbarkeit*, *Stabilität*, *Testbarkeit* sowie *Standardkonformität* zerlegt. Portabilität geht nach näheren Untersuchungen nur über die Untereigenschaft *Anpassungsfähigkeit* in das Qualitätsmodell ein, die auf dieselbe Weise bewertet wird wie die *Änderbarkeit*. Damit ist eine nähere Untersuchung dieser Untereigenschaft nicht erforderlich. [Teß12]

Analysierbarkeit zielt auf das Systemverstehen ab. Einen Überblick zu Theorien und Untersuchungen findet sich in [BR04]. Wichtige Merkmale sind der Umfang von Entitäten, die Größe feingranularer Elemente (z. B. Anzahl Methoden pro Klasse) sowie die Anzahl der Elemente pro Struktureinheit (Hierarchisierung). Ebenso relevant sind die Anzahl von Kopplungen der Elemente sowie die Anzahl möglicher Pfade durch ein Element (Schleifen, Bedingungen). Die verwendete Programmiersprache definiert die möglichen hierarchischen Strukturen: Die Vererbung, die Organisation von Elementen in Klassen und Paketen sowie die Komposition von Komponenten. Zu beachten ist weiterhin der Einfluss redundanten Codes, sowohl was die Anzahl als auch seinen Umfang betrifft.

Die Güte der Änderbarkeit zeigt sich anhand der Strukturierung, genauer: in der Modularität des Quellcodes. Sie wird bestimmt durch die Kopplung und Kohäsion der einzelnen Elemente sowie durch ihre Schnittstellen. Relevant sind die Komplexität der Methodenschnittstellen, die Anzahl der Verbindungen zu anderen Elementen über die Schnittstelle (=Kopplung) und die Breite der gesamten Klassenschnittstelle (=Anzahl öffentlicher Variablen und Methoden).

Die Bewertung der Korrektheit und der Robustheit als Merkmale der Stabilität ist nur eingeschränkt möglich. Für Exceptions kann ein Vergleich der Anzahl gefangener und weitergeleiteter Exceptions Hinweise auf die Robustheit geben. Die Korrektheit hingegen kann abgeschätzt werden durch Assertions, die die eingehenden Parameter einer Methode prüfen.

Die Testbarkeit ist abhängig vom Grad der Autonomie der Elemente und der Vererbungstiefe und sind damit Teil der Modularität.

Damit ergibt sich folgende Basis des Qualitätsmodells:

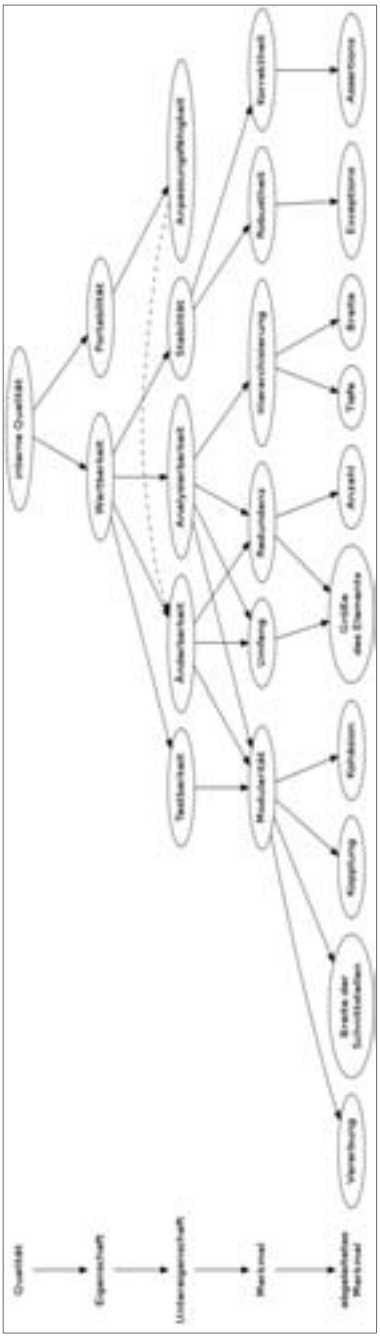


Abbildung 8: Basis des Qualitätsmodells

3.2 Softwaremodell

Softwaremodelle sind die Grundlage jeder statischen Analyse von Quellcode und haben die Aufgabe, die im Kontext der jeweiligen Untersuchung relevanten Attribute herauszustellen. Dabei beeinflussen sowohl die verwendete Programmiersprache als auch das Messziel die Menge dieser Merkmale. Moderne Sprachen besitzen ein Typenkonzept, kennen Schleifen und Konditionalausdrücke, und können mit Hilfe von Funktionen bzw. Prozeduren eine Sichtbarkeitsbeschränkung von Elementen definieren. Damit kann das Information Hiding und der Modularisierungsgedanke von Parnas praktisch umgesetzt werden (s. [Par72]).

Die zur Implementierung der Fallstudie verwendete Programmiersprache Java gehört zu den objektorientierten Sprachen. Das für die Bewertung benötigte Softwaremodell muss daher die durch dieses Paradigma eingeführten Strukturierungsmöglichkeiten berücksichtigen. Aus Sicht des Qualitätsmodells sind folgende Merkmale relevant:

- Breite der Schnittstelle: Als Messobjekte kommen nur Strukturelemente in Frage, die entweder eine eigene Schnittstelle besitzen (Methoden und Klassen) oder für sich eine Schnittstelle herleiten lässt (Pakete). Subsysteme besitzen ebenfalls eine Schnittstelle, oft in Form einer Fassade (s. [Gam+96]). Die zuverlässige automatische Erkennung von Subsystemen ist jedoch schwierig, daher werden sie hier nicht als Messobjekt berücksichtigt. Ähnlich verhält es sich mit Komponenten.

- Kopplung: Eine Methode ist definitionsgemäß immer an die sie definierende Klasse gekoppelt. Damit bestehen potentiell weitere Verbindungen zu Attributen und Methoden in ihren verschiedenen Ausprägungen. Kopplungen zu anderen Klassen erfolgen durch die Benennung von Klassen als Typ in der Parameterliste, durch die Erzeugung von Instanzen von Klassen innerhalb einer Methode oder über die Definition einer lokalen Klasse. Bei Klassen entsteht Kopplung zunächst durch die Zugehörigkeit zu einem Paket und weiter über die Deklaration von Attributen oder die Definition lokaler Klassen. Da das Verhalten einer Klasse auch von ihrer Superklasse abhängt (soweit vorhanden), besteht auch hier eine Abhängigkeitsbeziehung und damit eine Kopplung. Diese Abhängigkeit gibt es auch bei Schnittstellen untereinander, die ebenfalls Teil einer Vererbungshierarchie sein können. Die Implementierung durch Klassen ist eine weitere Kopplungsform. Die Abhängigkeit einer Methode bzw. Klasse von einer Klasse kann auch zu einer Abhängigkeit von einem „Fremd“-Paket führen, wenn erst der Import diese Klasse verfügbar macht. Die Kopplung von Komponenten untereinander ist abhängig davon, wie ein System sie integriert, eine Ermittlung der Kopplung ist daher nicht ohne Zusatzwissen möglich. Das gilt ebenso für die Kopplung von Subsystemen. Pakete definieren über die import-Anweisung eine Abhängigkeitsbeziehung zu anderen Paketen. Weiterhin gilt, dass eine Abhängigkeitsbeziehung aus Sicht des Gesamtsystems transitiv ist, d. h. instabile Pakete sind (in-)direkt abhängig von stabilen Paketen (s. [Mar02]). Aus diesem Grund ist die Position eines Pakets innerhalb dieses Abhängig-

keitsgraphen von Interesse. Zusammen mit der Abstraktheit kann eine Überprüfung hinsichtlich des Stable-Abstraction-Prinzips erfolgen.

- Kohäsion: Die verschiedenen Formen der Kohäsion, wie sie u. a. von Yourdon und Constantine beschrieben wurden (s. [YC79]), untersuchen die Kohäsion auf der Ebene von Funktionen bzw. Prozeduren. Da in der Objektorientierung die Klasse das eigentliche Modellierungselement ist, steht ihre Kohäsion bei der Bewertung im Vordergrund. Ermittelt wird die Kohäsion anhand der Methoden und deren Zugriffe auf disjunkte Attributmengen der Klasse. Die Kohäsion von Komponenten kann aufgrund ihrer zur Klasse identischen Struktur auf dieselbe Weise ermittelt werden. Eine Übertragung dieses Ansatzes auf die Paketebene ist nicht möglich. Die Bewertung der Kohäsion von Paketen ist nach Auffassung von Martin abhängig vom gewünschten Grad der Wiederverwendung (s. [Mar02]). Dementsprechend beschreiben die von ihm aufgeführten drei Prinzipien REP, CRP und CCP Kriterien, anhand derer Klassen in Paketen organisiert werden sollten. Inwieweit ein System diesen Prinzipien Folge leistet, ist aus dem Quellcode allerdings nicht zu ermitteln. Da auch die Anzahl der die Wiederverwendung nutzenden Client- Klassen nicht vorherzusehen ist, kann für Pakete keine zuverlässige Kohäsionsbestimmung vorgenommen werden.

- Vererbung: Die Bewertung der „Güte“ einer Vererbung im Sinne des Liskovschen Ersetzungsprinzips ist aufgrund der semantischen Bedeutung der Vererbung in der Problemdomäne nicht möglich. Hinweise können allerdings die Zahlen geerbter, eigener und überdeckter Attribute bzw. Methoden liefern. Darüber hinaus ist die Anzahl von Kindklassen und die Position der Klasse in der Vererbungshierarchie relevant, um die Stabilität der Klasse zu bewerten. Dazu gehört auch die Anzahl finaler Attribute und Methoden. Besonders wichtig ist die Vererbung über Paketgrenzen hinweg, wobei sie für Pakete selbst keine Rolle spielt. Bis auf die Ermittlung der Attribute und finalen Bestandteile sind die Messaufgaben für Schnittstellen identisch.

- Größe der Entität: Der Analyseaufwand steht in Relation zum Umfang eines Messobjekts und lässt sich recht einfach in Zeilen erfassen. Für die Bewertung des Umfangs einer Methode ist zusätzlich die Anzahl möglicher Pfade wichtig. Der Umfang von Klassen wird ebenso anhand ihrer Zeilenanzahl bestimmt; relevant ist außerdem die Anzahl der Attribute und Methoden. Für Schnittstellen kann nur die Anzahl von Methoden ermittelt werden. Die Größe eines Pakets kann sinnvoll nur in Form von Zeilen, Klassen oder Schnittstellen angegeben werden.

- Anzahl: Redundanz kann auf verschiedenen Strukturierungsebenen vorkommen. Die mögliche Spannbreite reicht von nahezu identischen Blöcken (vergleichsweise geringer Umfang) über Methoden bis hin zu kompletten Klassen (umfangreiche Redundanz), die kopiert und nur marginal verändert werden (die entspräche einem Umgehen der Vererbung).

- Breite und Tiefe von Hierarchien: Die Vererbung als eine Möglichkeit der hierarchischen Strukturierung wird als eigenes abgeleitetes Merkmal separat betrachtet. Eine weitere statische Hierarchie bildet sich durch die möglichen Zugehörigkeiten von Elementen, die Java bietet: Ein Block gehört zu einer Methode, die wiederum einer Klasse oder Schnittstelle (hier geht allerdings nur die Schnittstelle der Methode ein) zugeordnet ist usw. Lokale Klassen müssen auch berücksichtigt werden, solange sie einen Bezeichner tragen. Anonyme Klassen fallen damit weg, was angesichts ihres geringen Einflusses auf die Struktur unproblematisch ist. Die Elemente dieser Hierarchie definieren jeweils einen Sichtbarkeitsbereich und bestimmen die Lebensdauer der eingeschachtelten Elemente und besitzen optional eine Schnittstelle. Für eine Bewertung ist die Breite und Tiefe, also die Morphologie dieser Dekompositionshierarchie, von Interesse. Da Blöcke keine Bezeichner tragen, beginnt die Untersuchung der Hierarchie auf der Ebene der Methoden. Aus demselben Grund werden auch Attribute nicht mit aufgeführt.

- Assertions und Exceptions: Als Messobjekte für die Ermittlung kommen in Java nur Methoden in Frage, da für sie eine Schnittstelle mit Parametern definiert werden kann. Relevant ist der Grad der Abdeckung bei der Überprüfung eingehender Parameter durch Assertions und die Anzahl gefangener und weitergeleiteter Exceptions.

3.3 Bewertung von Kenngrößen

Für alle strukturrelevanten Merkmale sollten die ermittelten Kenngrößen im Intervall [0, 9] mit 0 als natürlichem Infimum und 9 als Indikatorgrenzwert liegen; das Teilintervall [3, 7] beschreibt einen akzeptablen Wertebereich für Kenngrößen. Im Detail gilt für die einzelnen Messaufgaben:

- Die Größe von Methoden sollte weniger als 30 Zeilen betragen.

- Die Kohäsion von Klassen sollte möglichst maximiert und entsprechend die Anzahl disjunkter Attributmengen minimiert werden.

- Die Anzahl möglicher Pfade sollte < 9 sein.

- Redundanzen sollten möglichst nicht auftreten, daher wird hier der Wert 0 angestrebt.

- Die Kenngrößen bei der Vererbung überschneiden sich, es gilt aber als Richtwert die in der jeweiligen Klasse bekannte Menge von Attributen bzw. Methoden. Damit gilt für die Bestandteile von Klassen: #geerbter Attr./Meth. + #eigene Attr./Meth. − #überdeckter Attr./Meth. < 9. Die Vererbung über Paketgrenzen hinweg sollte allerdings minimal sein.

- Die lokale Kopplung sollte weniger als 9 Verbindungen ausweisen. Das gilt auch für die Kopplung über Paketgrenzen hinweg.

- Die Abstraktheit und Stabilität eines Pakets liegt immer zwischen 0 und 1. Sie hängt ab von der Position der Klassen in der Vererbungshierarchie und der Kopplungsrichtung der Pakete. Als Richtlinie gilt: Die Stabilität eines Pakets A sollte immer größer sein als die Stabilität eines Pakets B, das von A abhängt. Für die Abstraktheit gilt: Ein Paket sollte so abstrakt wie stabil sein.

- Die Anzahl in einer Methode gefangenen im Vergleich zur Anzahl der weitergeleiteten Exceptions sollte möglichst groß sein, da sie ein Hinweis auf die Robustheit der Klasse ist.

- Die Bewertung von Assertions kann nur vor dem Hintergrund des Defensive-Programming-Ansatzes erfolgen, d. h. es ist ein Abgleich der Parameterliste und etwaiger zugehöriger Assertions notwendig. Ein sinnvoller Grenzwert kann hier aber nicht angegeben werden, so dass die Bewertung nur nominal erfolgt (Assertion vorhanden oder nicht).

Bei der Bewertung der Morphologie steht die Gleichmäßigkeit der Baumstruktur im Vordergrund, es wird also auf eine möglichst ausgewogene Verteilung der bei der Dekomposition von Java-Quellcode entstehenden Elemente auf die Strukturierungsebenen abgezielt. Die Bewertung eines Teilbaums der hierarchischen Struktur ergibt sich aus seiner Abweichung von einer idealen Struktur (max. 9 Methoden/Klasse, 9 Klassen/Paket usw.). Die Grenzwerte nehmen bei der Bewertung die Rolle einer Indikatorfunktion ein. Das dazu benötigte Intervall ergibt sich aus dem jeweiligen angegebenen Maximalwert und dem Nullpunkt der zugehörigen Skala (hier immer 0). Liegt ein Wert x nun außerhalb eines solchen angegebenen Intervalls T , so ist er ein Indikator für ein potentiell risikobehaftetes Messobjekt. Damit ist allerdings noch nicht die Frage beantwortet, wie die Ergebnisse der einzelnen Messaufgaben zu einer Aussage bzgl. des abgeleiteten Merkmals kombiniert werden können. Zuse hat in [Zus98] auf die verschiedenen Probleme solcher Kombinationen hingewiesen. Daher wird beim vorliegenden Qualitätsmodell die von Lanza und Marinescu vorgeschlagene Detection Strategy verwendet, die letztendlich nur eine aussagenlogische Komposition der Indikatorergebnisse ist. Praktisch bedeutet das, dass die Ergebnisse der Indikatorfunktionen eines abgeleiteten Merkmals mit Hilfe eines logischen Operators wie ∧ und ∨ kombiniert werden. Diese Kombinationsform ist auf allen Ebenen des Qualitätsmodells anwendbar, so dass schließlich eine Antwort auf die Frage nach der internen Qualität der Art „in Ordnung“ oder „nicht in Ordnung“ möglich wird.

4 Fallstudie

4.1 Das BIS

Das „Bielefelder Informationssystem“ (BIS) entstand aus dem Bedarf heraus, die Überschneidung von Veranstaltungen für die Studierenden zu reduzieren. Die ersten Arbeiten dazu begannen 1998 im Kontext der Lehramtsausbildung, die besonders mit diesem Problem zu kämpfen hat. Der Kern des BIS, das elektronische kommentierte Vorle-

sungsverzeichnis (eKVV) vereint die vorher von den einzelnen Fakultäten und Lehrstühlen auf eigenen Webseiten bereit gestellten Veranstaltungsbeschreibungen und bietet Lehrenden wie Studierenden einen einheitlichen Zugang zu diesen Informationen über das Internet.

Nach Einführung der Prüfungsverwaltung als letzte große Neuerung im Jahr 2006 gibt es seit 2011 die Möglichkeit, die an der Universität Bielefeld existierenden Studiengänge zu modellieren und damit eine automatische Notenberechnung anzustoßen.

4.2 Untersuchung der Morphologie

Eine Untersuchung befasste sich mit der Morphologie des Systems, insbesondere der des Pakets org.unibi.common. Ermittelt wurde Kopplung, Breite und Tiefe der Hierarchisierung sowie die Größe der Elemente.

Die Kopplung muss aufgrund der zyklischen Abhängigkeit als gravierend fehlerhaft bewertet werden. Die enge Kopplung von org.unibi.common und org.unibi.data mit 6770 Verbindungen bzw. 913 in der Gegenrichtung liegen aus Sicht des Qualitätsmodells weit außerhalb aller empfohlenen Grenzwerte. Mit einer Tiefe von maximal 7 Ebenen, jedoch einer Breite von 17 und mehr Elementen ist die Hierarchisierung nicht gleichmäßig. Die berechnete mittlere Gesamtabweichung beträgt 1,23 bei einem angenommenen Schachtelungsfaktor von 7 und ist damit ebenfalls zu groß. Sowohl Modularität als auch Hierarchisierung sind aufgrund dieser Ergebnisse als „nicht in Ordnung“ anzusehen.

Insgesamt zeigen sich für die Untereigenschaften Analysierbarkeit, Änderbarkeit und Testbarkeit z. T. erschreckende Mängel, die dazu führen, dass die Wartbarkeit des Systems insgesamt hinterfragt werden muss. Diesbezügliche Gespräche mit dem Entwickler-Team sind geplant, inwieweit es aber zu entsprechenden Refactoring-Arbeiten kommen wird, ist angesichts des derzeit geplanten neuen Campus-Management-Systems unklar.

5 Möglichkeiten und Grenzen des Ansatzes

Das vorgestellte Qualitätsmodell ist nicht als Konkurrent zu bestehenden Ansätzen zu verstehen, die von den beschriebenen Softwaremaßen verwendet werden, sondern als Ergänzung. Das gilt sowohl für die Ermittlung und Verarbeitung von Kenngrößen als auch für die Bewertung nicht erfasster Messaufgaben wie Redundanz, Assertion- und Ausnahmebehandlung sowie der Morphologie.

Die etablierten Maße haben den Vorteil, dass sie ihre Tauglichkeit bei der Bewertung von Strukturen in mehreren Untersuchungen gezeigt haben. Gerade die für die Bewertung der Wartbarkeit wichtigen Merkmale Modularität und Vererbung werden auf verschiedenen Strukturebenen erfasst und stellenweise sogar systemweit bewertet. Eine am jeweiligen Projekt ausgerichtete Kalibrierung vorausgesetzt bilden sie ein relativ zuverlässiges Instrumentarium, um problematische Elemente und Strukturen zu identifizieren.

Gerade auf der Architekturebene zeigen sich jedoch für den Entwickler auch Nachteile. Eine Bewertung soll ihm als Orientierung dienen, die entsprechend als schlecht bewerteten Bereiche zu finden und zu korrigieren. Dazu ist es allerdings erforderlich, dass klar wird, welche Elemente bzw. Strukturen in welchem Maße für das Bewertungsergebnis verantwortlich sind. Speziell für einige abgeleitete Maße ist das nicht der Fall. Dazu kommt, dass sich eine Bewertung am Kenntnisstand des Entwicklers ausrichten sollte. Dies lässt sich durch Veränderung der Grenzwerte jedoch relativ problemlos realisieren. Es ist dann aber wiederum nicht eindeutig ersichtlich, wie sich die Bewertung der Strukturen im Detail ändert.

Im Gegensatz zu den etablierten Qualitätsmodellen liefert das in dieser Arbeit entworfene Modell keine höherwertigen quantitativen Aussagen, sondern funktioniert als reiner Indikator. Aus diesem Grund können keine quantitativen Abschätzungen zu den einzelnen Qualitätseigenschaften, geschweige denn zum potentiellen Aufwand von Wartungsarbeiten erbracht werden. Derartige Angaben haben jedoch immer nur im Kontext der ganz spezifischen Messsituation eine Bedeutung und sind zu einem guten Teil abhängig von der subjektiven Einschätzung des Messenden.

Die Anwendung des Qualitätsmodells wird derzeit noch von der fehlenden Werkzeugunterstützung behindert. Zwar können viele der elementaren Kenngrößen durch die gängigen Werkzeuge zur Metrikanalyse ermittelt werden, aber die Interpretation dieser Werte und die Kombination der Ergebnisse der Indikatorfunktion fehlt. Für die Untersuchungen im Rahmen dieser Arbeit wurden die XML-Exporte des Eclipse Metrics-Plugins mit Hilfe von selbst entwickelten Python-Skripten ausgewertet. Zweckmäßig wäre eine Erweiterung des Plugins um diese Auswertungsfunktionalität.

Weiter untersucht werden muss außerdem die Berücksichtigung der Stabilität. Wann ist eine Methode bzw. Klasse so mit Assertions und Exceptions ausgestattet, dass sie als stabil gelten kann? Auch eine Ermittlung der Testabdeckung könnte dazu herangezogen werden, die allerdings aufgrund des Late Bindings schwierig zu ermitteln sein wird.

Literaturverzeichnis

[10d] Metrikbasierte Qualitätsanalyse - Einstieg. Virtuelles Software Engineering Kompetenzzentrum (ViSEK), Fraunhofer IESE. 2010. URL: http://www.softwarekompetenz.de/ (besucht am 10. 10. 2011).

[10e] Prädiktor. 2010. URL: http://de.wikipedia.org/w/index.php?title=Prognose&oldid= 80317094 (besucht am 15. 10. 2010).

[AC94] Fernando Brito e Abreu und Rogério Carapuça. „Object-Oriented Software Engineering: Measuring and Controlling the Development Process“. In: Proceedings of the 4th Int. Conf. on Software Quality. McLean, VA, USA, Okt. 1994.

[Agg+06] K. K. Aggarwal u. a. „Empirical Study of Object-Oriented Metrics“. In: Journal of Object Technology 5.8 (Dez. 2006), S. 149–173.

[And+04] Christoph Andriessens u. a. QBench Projektergebnis: Stand der Technik. 2004. URL: http://www.qbench.de.

[BBM96] Victor R. Basili, Lionel C. Briand und Walcelio L. Melo. „A Validation of Object-Oriented Design Metrics as Quality Indicators“. In: IEEE Transactions on Software Engineering 22.10 (Okt. 1996), S. 751–761.

[BCK98] Len Bass, Paul Clements und Rick Kazman. Software Architecture in Practice. Reading et al.: Addison-Wesley, 1998.

[BDW99] Lionel C. Briand, John W. Daly und Jürgen K. Wüst. „A Unified Framework for Coupling Measurement in Object-Oriented Systems“. In: IEEE Transactions on Software Engineering 25.1, Jan. 1999.

[BL76] L. A. Belady und M. M. Lehman. „A model of large program development“. In: IBM Systems Journal 15.3 (1976), S. 225–252.

[BMB99] Lionel C. Briand, Sandro Morasca und Victor R. Basili. „Defining and Validating Measures for Object-Based High-Level Design“. In: IEEE Transactions on Software Engineering 25.5 (Sep. 1999), S. 722–743.

[Boe79] Barry W. Boehm. „Software Engineering: R and D trends and defense needs“. In: Research Directions in Software Technology. Hrsg. von Peter Wegener. Cambridge, Massachusetts und London, England: M. I. T. Press, 1979.

[Bot+04] P. Botella u. a. „ISO/IEC 9126 in practice: what do we need to know?“ In: Procs. First Software Measurement European Forum (SMEF). Rom, Jan. 2004.

[BR04] Marcel Bennicke und Heinrich Rust. Programmverstehen und statische Analysetechniken im Kontext des Reverse Engingeering und der Qualitätssicherung. Projektbericht ViSEK/025/D. Kaiserslautern: Fraunhofer IESE, Feb. 2004.

[CK91] Shyam R. Chidamber und Chris F. Kemerer. „Towards a Metrics Suite for Object Oriented Design“. In: Conference proceedings on Object-oriented programming systems, languages, and applications 06–11 (Okt. 1991), S. 197–211.

[CM78] Joseph P. Cavano und James A. McCall. „A framework for the measurement of software quality“. In: Proceedings of the software quality assurance workshop on Functional and performance issues. ACM, 1978, S. 133–139.

[Cos+05] Gennaro Costagliola u. a. „Class Point: An Approach for the Size Estimation of Object-Oriented Systems“. In: IEEE Transactions on Software Engineering 31.1, Jan. 2005), S. 52–74.

[CY90] Peter Coad und Edward Yourdon. Object Oriented Analysis. 2. Auflage. Englewood Cliffs, New Jersey 07632: Yourdon Press, Nov. 1990.

[DP09] Stéphane Ducasse und Damien Pollet. „Software Architecture Reconstruction. A Process-Oriented Taxonomy“. In: IEEE Transactions on Software Engineering 35.4 (Juli 2009), S. 573–591. ISSN: 0098-5589.

[Ede93] D. Vera Edelstein. „Report on the IEEE STD 1219-1993–standard for software maintenance“. In: SIGSOFT Softw. Eng. Notes 18.4 (1993), S. 94–95.

[Eic+98] Stephen G. Eick u. a. „Does Code Decay? Assessing the Evidence from Change Management Data“. In: Technical Report NISS 81 (März 1998).

[Erl00] L. Erlikh. „Leveraging Legacy System Dollars for EBusiness“. In: IT Pro May/June (2000), S. 17–23.

[FN01] Fabrizio Fioravanti und Paolo Nesi. „Estimation and Prediction Metrics for Adaptive Maintenance Effort of Object-Oriented Systems“. In: IEEE Transactions on Software Engineering 27.12 (Dez. 2001), S. 1062–1084.

[Fow99] Martin Fowler. Refactoring. Improving the Design of Existing Code. Amsterdam: Addison-Wesley Longman, 1999. ISBN: 0-201-48567-2.

[Gam+96] Erich Gamma u. a. Entwurfsmuster. 5. korrigierte Auflage. München et. al.: Addison-Wesley, 1996. ISBN: 3-8273-1862-9.

[Gos+03] Katerina Goseva-Popstojanova u. a. „Architectural-Level Risk Analysis Using UML“. In: IEEE Transactions on Software Engineering 29.10 (Okt. 2003), S. 946–960.

[Hal77] Maurice H. Halstead. Elements of Software Science. Amsterdam: Elesevier Scientific Publishing Company, 1977.

[HCN98] Rachel Harrison, Steve J. Counsell und Reuben V. Nithi. „An Evaluation of the MOOD Set of Object-Oriented Software Metrics“. In: IEEE Transactions on Software Engineering 24.6 (Juni 1998), S. 491–496.

[HK81] S. Henry und D. Kafura. „Software Structure Metrics Based on Information Flow“. In: IEEE Trans. Softw. Eng. 7.5 (Sep. 1981), S. 510–518.

[HS01] T. E. Hastings und A. S. M. Sajeev. „A Vector-Based Approach to Software Size Measurement and Effort Estimation“. In: IEEE Transactions on Software Engineering 27.4 (Apr. 2001), S. 337–350.

[HT03] Andrew Hunt und David Thomas. Der Pragmatische Programmierer. München, Wien: Carl Hanser, 2003.

[Lar04] Craig Larman. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. 3. Auflage. Prentice Hall, Nov. 2004.

[LC01] Victor Laing und Charles Coleman. Principal Components of Orthogonal Object-Oriented Metrics. White Paper. NASA Software Assurance Technology Center, 2001.URL: http://satc.gsfc.nasa.gov/support/OSMASAS_SEP01/Principal_Components_of_Orthogonal_Object_Oriented_Metrics.pdf.

[Leh+97] M. M. Lehman u. a. „Metrics and Laws of Software Evolution - The Nineties View“. In: Proceedings Metrics 97 Symposium, Nov. 5-7th (1997).

[LH89] Karl J. Lieberherr und Ian Holland. „Assuring Good Style for Object-Oriented Programs“. In: IEEE SOFTWARE 6 (1989), S. 38–48.

[Lis87] Barbara Liskov. „Data Abstraction and Hierarchy“. In: ACM SIGPLAN Notices. Bd. 23 (5). Mai 1987.

[LK94] Mark Lorenz und Jeff Kidd. Object-Oriented Software Metrics. Upper Saddle River, NJ, USA: Prentice Hall, Juli 1994.

[LM06] Michele Lanza und Radu Marinescu. Object-Oriented Metrics in Practice: Using Software Metrics to Characterize, Evaluate, and Improve the Design of Object-Oriented Systems. Berlin, Heidelberg, New York: Springer, Okt. 2006.

[Mar02] Robert Cecil Martin. Agile Software Development. Principles, Patterns, and Practices. Upper Saddle River, New Jersey 07458: Prentice Hall International, Nov. 2002.

[Mar09] Robert C. Martin. Clean Code. Refactoring, Patterns, Testen und Technik für sauberen Code. Heidelberg et al.: mitp, 2009. ISBN: 978-3-8266-5548-7.

[McC76] Thomas J. McCabe. „A Complexity Measure“. In: IEEE Transactions on Software Engineering SE-2.4 (1976), S. 308–320.

[Mey97] Bertrand Meyer. Object-oriented Software Construction. 2. Auflage. Upper Saddle River 07458: Prentice Hall International, Mai 1997.

[MWT94] H. Müller, K. Wong und S. Tilley. „Understanding software systems using reverse engineering technology“. In: The 62nd Congress of L'Association Canadienne Francaise pour l'Avancement des Sciences Proceedings (ACFAS) (1994).

[Mye75] Glenford J. Myers. Reliable Software Through Composite Design. New York: Petrocelli/Charter, 1975.

[Par72] D. L. Parnas. „On the criteria to be used in decomposing systems into modules“. In: Commun. ACM 15.12 (1972), S. 1053–1058.

[Par75] D. L. Parnas. „The influence of software structure on reliability“. In: Proceedings.of the international conference on Reliable software. Los Angeles, California: ACM, 1975, S. 358–362.

[RL04] Stefan Roock und Martin Lippert. Refactoring in großen Softwareprojekten. Heidelberg: dpunkt, 2004. ISBN: 978-3898642071.

[SI93] Martin Shepperd und Darrel Ince. Derivation and Validation of Software Metrics. Oxford: Oxford University Press, Sep. 1993.

[Szy99] Clemens Szyperski. Component Software. Harlow, England et. al.: Addison-Wesley, 1999.

[Teß12] Architekturbezogene Qualitätsmerkmale für die Softwarewartung: Entwurf eines Quellcode basierten Qualitätsmodells. Dissertation. Bielefeld: Universität Bielefeld, 2012.

[Wal01] Wallmüller, Ernest: Software – Qualitätssicherung in der Praxis. München et al.: Hanser Fachbuch, 2001.

[WYF03] Hironori Washizaki, Hirokazu Yamamoto und Yoshiaki Fukazawa. „A Metrics Suite for Measuring Reusability of Software Components“. In: 2003, S. 211–223.

[Zus98] Horst Zuse. A Framework of Software Measurement. Berlin, New York: de Gruyter, 1998.

Nachhaltiges Software Management durch Lebenszyklus-übergreifende Überwachung von Qualitätskennzahlen

Sandra M. Lang, Bernhard Peischl[1]

Softnet Austria
Inffeldgasse 16b/II, 8010 Graz
bernhard.peischl@soft-net.at
sandra.lang@soft-net.at

Abstract: In diesem Artikel motivieren wir den Begriff der Nachhaltigkeit und argumentieren, dass zur Bewertung der Nachhaltigkeit einer Software der gesamte Lebenszyklus betrachtet werden sollte. In diesem Kontext ist es notwendig, Kennzahlen aus verschiedenen Bereichen (Ressourcen, Produkt, Prozess) zu operationalisieren, um Auswirkungen auf Qualität und Nachhaltigkeit quantitativ erfassbar zu machen. Wir gehen in diesem Artikel auf unseren Cockpit Ansatz ein und präsentieren eine Fallstudie aus dem Vorgangsmanagement eines Automobilzulieferers. Daran sieht man, dass für praktische Fragestellungen, die Operationalisierung von Kennzahlen ohne flexible Messinfrastruktur nicht zu bewerkstelligen ist und ein Software Cockpit hier wesentliche Unterstützung leisten kann. Dies gilt im Speziellen für die vernetzten Fragestellungen im Umfeld des nachhaltigen Software Managements.

1 Motivation

Nachhaltigkeit gewinnt in vielen Bereichen der Wirtschaft und des gesellschaftlichen Lebens an Bedeutung. Im ursprünglichen Wortsinn („längere Zeit anhaltende Wirkung“) [Du01] entstammt das Wort von „nachhalten“, mit der Bedeutung „längere Zeit andauern oder bleiben“. Im modernen Sinn kommt der Gedanke hinzu, dass etwas andauern, bleiben, nachwirken oder haltbar sein kann noch lange nachdem es gebaut oder in Bewegung gesetzt wurde. Auf das Management von Software übertragen meint der Begriff eine langlebige, die Ressourcen schonende Nutzbarkeit. Ob und vor allem wie nachhaltig eine Software entwickelt wurde, zeigt sich allerdings erst bei Betrachtung des gesamten Lebenszyklus der Software – Entwurf, Umsetzung, Betrieb, inkrementelle Verbesserung und Wartung müssen in die Nachhaltigkeitsbetrachtung einfließen.

[1] Die Autoren sind in alphabethischer Reihenfolge gelistet. Diese Arbeiten wurden teilweise im Rahmen des Kompetenznetzwerkes Softnet Austria II (www.soft-net.at, COMET Programm der FFG) durchgeführt und vom Bundesministerium für Wirtschaft, Familie und Jugend (bmwfj), dem Land Steiermark, der Steirischen Wirtschaftsförderungsgesellschaft mbH (SFG), und der Stadt Wien durch das Zentrum für Innovation und Technologie (ZIT) unterstützt.

Die hohe IT-Durchdringung von Unternehmen, Behörden und dem Privatbereich impliziert auch, dass Entscheidungen im Software Management nicht alleine (oft quantifizierbare) technische Aspekte beeinflussen, sondern viel mehr auch (kaum oder nur sehr schwer messbare) die Ressourcen betreffende, ökonomische und soziale Auswirkungen haben. Diese Aspekte müssen in die Nachhaltigkeitsbetrachtung einfließen. So gehen z.B. durch Offshoring und Cloud Computing Arbeitsplätze an Billiglohnländer. Gerade in Zeiten von globalen geopolitischen Veränderungen (Währungsturbulenzen, Ressourcen-Knappheit, Herausbildung einer multipolaren Weltordnung, demographische Entwicklungen) wird die Bewertung von Nachhaltigkeit im Sinne einer quantifizierbaren Abschätzung der Auswirkungen von Entscheidungen im Software Management weiter an Bedeutung gewinnen. Während Nachhaltigkeit in manchen Ingenieurdisziplinen (z.B. Produktionsprozesse und Dienstleistungen im Umweltmanagement und im Zivilingenieurwesen) bereits Einzug in die Standardisierung gehalten hat (z. B. ISO 14000 [Is04]) stehen wir diesbezüglich im Management von Software Systemen erst am Beginn dieses Prozesses.

2 Nachhaltiges Software Management

Nachhaltigkeit in der Softwareentwicklung sollte als orthogonaler Aspekt des Gesamtsystems, ähnlich wie andere Qualitätsattribute [Pe11, ZM07] wie z.B. funktionale Korrektheit, Sicherheit (Safety und Security) oder Verfügbarkeit, betrachtet werden. Noch in größerem Ausmaß als diese wohlbekannten (aber nach wie vor nicht einfach zu operationalisierenden) Qualitätsattribute betrifft dies den Aspekt der Nachhaltigkeit. Idealerweise sollte das zusätzliche Attribut der Nachhaltigkeit in einem sehr frühen Stadium eingeführt werden und alle weiteren Phasen wie Entwicklung, inkrementelle Verbesserung, Betrieb und Wartung begleiten. Auch muss sich die Operationalisierung der Attribute zur Bewertung der Nachhaltigkeit in die vorherrschenden Prozessabläufe und – soweit möglich – in die bestehende Toollandschaft integrieren. Ansätze, um die Nachhaltigkeit im Software Management messbar zu machen, werden z.B. in [AXTLZL10] diskutiert. Die Autoren schlagen vor, die Nachhaltigkeit (genauer Sustainability Performance, siehe [AXTLZL10]) als eine Menge an spezifischen Qualitätsattributen zu messen. Durch Erfassung dieser Metriken können so ökonomische, soziale oder ressourcenbezogene Ziele quantifiziert werden. Die Autoren dieser Arbeit zeigen qualitativ die Messung der Nachhaltigkeit und präsentieren eine Fallstudie für eine Software Plattform zum Wassermanagement. Die umfangreiche Liste der Metriken umfassen Ressourcen, Prozessaspekte und produkt-zentrische Metriken. Jedoch lassen die Autoren offen, wie in diesem Modell für Nachhaltigkeit diese Vielzahl von Kennzahlen in der Entwicklung und im Betrieb mit vernünftigem Aufwand operationalisiert werden kann.

Ein nicht unwesentliches Problem in der Erfassung von Kennzahlen über den gesamten Lebenszyklus ist die automatisierte Speicherung und flexible Darstellung der Kennzahlen. Abhilfe kann ein Software Cockpit [Pe11, RBKWL10, ZM07] schaffen, jedoch sind viele dieser Lösungen auf ausgewählte Phasen spezialisiert wie z.B. ausschließlich auf die Entwicklungsphase.

Die sinnvolle Integration dieser hochspezialisierten Werkzeuge ist oft nicht nahtlos möglich, wodurch sich komplexere Abfragen auf den erfassten Datenbeständen, die die Grundlage für Berechnung der Kennzahlen darstellen, nicht oder zumindest nicht projektbegleitend in der Entwicklung oder dem operativen Betrieb einer Software umsetzen lassen. Zur Bewertung der Nachhaltigkeit muss die Auswahl der zu erfassenden Kennzahlen nach Kriterien wie ökonomische Vorteile, soziale Vorteile und Schonung bzw. vorteilhafte Aufteilung von Ressourcen erfolgen. Bevor auf das Cockpit und die den Lebenszyklus übergreifende Erfassung von Kennzahlen eingegangen wird, zeigen wir beispielhaft einige für die Nachhaltigkeit relevante Kennzahlen aus drei Bereichen: Vorgangmanagement (im Sinne einer Prozessmetrik), Aufwandsmanagement (im Sinne einer Ressource Metrik) und Wiederverwendbarkeit (im Sinne einer produkt-zentrischen Metrik, die direkt am Produkt abgegriffen wird).

Kategorie	**Nachhaltigkeitsaspekt**	**Fragestellung**	**Metrik**	**Beispiel**
Ressource	Ökonomisch	Wird das Budget überschritten?	Qualität der Aufwandschätzung	SOLL / IST Aufwände
	Sozial	Werden Überstunden vermieden?		
	Ressourcen	Wer arbeitet an welchen Arbeitspaketen (optimale Teams)?		
Prozess	Ökonomisch	Wie schnell wird auf die Kundenanfrage reagiert?	Bearbeitungszeit	Zeit, die benötig wird, um vom Zustand Neu in den Zustand Fertig zu kommen
	Sozial	Gibt es kritische Zeitschwellen für Feedback?		
	Ressourcen	Ist der 1st Level Support ausreichend dimensioniert?		
Produkt	Ökonomisch	Verbesserung im Time-to-Market?	Wiederverwendbarkeit	Verhältnis Anzahl Abstrakte Klassen / Anzahl konkreter Klassen
	Sozial	Mehr Erfolge mit weniger Aufwand?		
	Ressourcen	Wieviel Ersparnis an Personal bei Wiederverwendung?		

Tabelle 7: Beispielhafte Fragestellungen für nachhaltiges Software Management

Wie die obigen Beispiele zeigen, erfordern konkrete Fragestellungen in Hinblick auf ökonomische, soziale und Ressourcen schonendes Management einer Software die Erfassung und Darstellung von Kennzahlen der verschiedenen Kategorien.

3 Software Cockpits

Metriken lassen sich nach verschiedenen Kriterien klassifizieren. Wie oben angeführt kann z.B. zwischen Produkt- (z.B. Lines of Code, Halstead, Function Points, Bang, McCabe Cycliomatic Complexity, Testabdeckung etc.) und Prozess- (z.B. Durchlaufzeiten, Dauer, Anzahl von Ereignissen, Anzahl abstrakter oder konkreter Klassen etc.) und Ressourcemetriken (z.B. Teamgröße, Werkzeuge, Methoden etc.) unterschieden werden. Schließlich kann man noch bzgl. der Auswertung unterscheiden zwischen primitiven und berechneten Metriken, sowie zwischen Momentaufnahmen und laufenden Betrachtungen, etwa von Trends oder Raten. Gerade im Hinblick auf Fragestellungen der Nachhaltigkeit, die oft die Verknüpfung der drei oben angeführten Kategorien erfordert, ist die Zusammenführung dieser Kennzahlen zu einem Nachhaltigkeitsmodell von entscheidender Bedeutung.

3.1 Vom Ziel zur Metrik: Wie findet man relevante Metriken und aggregiert diese sinnvoll?

Wie in der Motivation erwähnt, ist es wichtig, sich von Anfang an klar zu machen, welche Metriken in einem konkreten Anwendungsfall relevant sind. Da die Fragestellungen im Bereich der Nachhaltigkeit sehr vernetzt sein können, geht es oft nicht nur um einzelne Kennzahlen (oder Gruppen von Kennzahlen); vielmehr muss der CIO oder der Prozessverantwortliche die Daten systematisch analysieren können um geeignete Hypothesen für Nachhaltigkeitsbetrachtungen aufstellen zu können. Daher ist eine effiziente Speicherung und flexible Darstellung der Daten ein zentraler Bestandteil eines modernen Software Cockpits (OLAP [Ni98]). Das Kernstück unseres Cockpits stellt ein Datawarehouse dar, das die Daten in Cubes, wie beispielsweise in [GBLP96] beschrieben, organisiert. Dies ermöglicht eine effiziente aggregierte Darstellung der Metriken (Drill Down, Zoom, Slicing, Dicing). So können Daten aus den für Nachhaltigkeitsbetrachtungen oben erwähnten drei Bereichen (Prozess-, Ressource- und Produktmetriken) anhand verschiedener Dimensionen (Zeit, Module, Versionen) analysiert und zusammengeführt werden. Innerhalb der einzelnen Dimensionen können die Metriken auf verschiedenen Ebenen aggregiert werden. Z.B. kann für die ausgewählte Dimension Zeit über Jahre, Monate, Tage oder auch Quartale und Wochen aggregiert werden. Dabei greifen wir auf MDX Queries (Multi-Dimensionale Expressions [AADG96]) zurück, wobei diese MDX Queries für Endanwender auch durch eine GUI unterstützt werden. Weiter ermöglichen MDX Queries auch das Filtern und Sortieren der großen Datenbestände. Vorgefertigte Schablonen ermöglichen auch dem Endanwender solche Queries zu erstellen.

3.2 Architektur

Abbildung 9 zeigt die Architektur des Cockpits. Das Kernstück ist ein Datawarehouse, dem einerseits ein ausgeflachtes Datenbankschema, in dem die Dimensionen und Fakten gespeichert sind, und andererseits die Definition der Cubes, Metriken und Aggregate, zu Grunde liegen.

Abbildung 9: Architektur unseres Cockpits

Die Daten werden unter Verwendung von Adaptern aus den Artefakten gemappt und importiert. Auch Metriken und eventuell notwendige Aggregattabellen werden in dieser Schicht berechnet. Zu den Artefakten zählen Daten aus dem Issue Management, wie z.B. Bugzilla, und UML Modelle und Source Code aus der Versionsverwaltung, wie beispielsweise SVN. Die Adapter sind so gebaut, dass sie Ressourcen aus der Versionskontrolle holen können und zusätzlich Details wie Datum und Revisionsnummer bzw. Version berücksichtigen und abgleichen.

Ein Administrationsinterface bietet die Möglichkeit Einstellungen für die Adapter, Produkte und Projekte vorzunehmen und zusätzlich den gesamten Batchprozess, der die Adapter anstößt, zu parametrisieren. Die Darstellung erfolgt mittels eines Webinterfaces, das wie vorhin schon erwähnt durch MDX Anfragen die graphische Darstellung, so wie die Drill Down Ansichten steuert.

3.3 Messung der Nachhaltigkeit durch die Frage nach der Wiederverwendbarkeit

Wir zeigen hier eine Produktmetrik für die Wiederverwendbarkeit [Sn97]. Nach der Goal-Question Metrik ist das zugrundeliegende Ziel die Messung der Nachhaltigkeit. Die Frage, die wir uns stellen, ist, inwieweit Klassen wiederverwendet werden können. Klar ist, dass abstrakte Klassen, sofern sie korrekt entworfen und implementiert wurden, sehr wahrscheinlich in anderen Projekten und Produkten, denen ähnliche Use Cases und Problemstellungen zu Grunde liegen, wiederverwendet werden können.

Wir betrachten daher als Maß für die Wiederverwendbarkeit, das Verhältnis von abstrakten Klassen zu konkreten Klassen. Exemplarisch zeigen wir hier wie dieses Verhältnis zwischen der Anzahl der abstrakten Klassen und der Anzahl konkreter Klassen in OCL (Object Constraint Language) realisiert wird. Zur Ausführung einer OCL Query auf UML Diagrammen haben wir das Werkzeug SQUAM [Op09] in das Cockpit eingebunden. So ist die Menge der abstrakten Klassen definiert durch:

```
self.oclIsTypeOf(Class) and self.isAbstract = #true
```

und die Menge der konkreten Klassen:

```
self.oclIsTypeOf(Class) and self.isAbstract = #false
```

Damit lassen sich die Anzahl der abstrakten bzw. konkreten Klassen wie folgt definieren:

```
context Model
def numAbstractClasses:
numAbstractClasses (): Integer =
       self.allOwnedElements() ->    select(e:Element|e.oclIsTypeOf(Class)
and    e.isAbstract =#true ) ->  size()
def numImplClasses:
numImplClasses (): Integer =
       self.allOwnedElements() >     select(e:Element|e.oclIsTypeOf(Class)
and    e.isAbstract =#false ) -> size()
```

daraus ergibt sich das Verhältnis zwischen den beiden durch:

```
context Model
```

Tabelle 1:GQM an einem Beispiel aus der Automobil-Zulieferindustrie

```
query RelAbstrClassesImplClasses:
       let result: Integer =
              self.numAbstractClasses()/self.numImplClasses()
       message result endmessage
```

4 Fallstudie: Vorgangsmanagement bei einem Software Produkt für die Automobilzulieferindustrie

Wir gehen hier nach der Goal-Question-Metric, wie in [BW84] beschrieben, vor. Zuerst wird ein Ziel (Goal) definiert, woraus dann Fragestellungen (Questions) abgeleitet werden. Erst aus diesen ergeben sich die tatsächlichen Metriken. Wir betrachten hierzu eine Fallstudie aus der Automobilzulieferindustrie[2]. Das Ziel bei diesem Projekt ist es die Qualität des Software Engineering Prozesses, wie Durchlaufzeiten von Change Requests und die korrekte Einhaltung des Workflows, zu messen und einen Überblick über die Change Requests und Defects zu bekommen.

[2]Aufgrund der Sensibilität und Vertraulichkeit der Informationen nennen wir kein Unternehmen und präsentieren die Daten anonymisiert. Unser Anwender entwickelt ein Software Werkezeug für die Automobilindustrie.

Die folgende Tabelle zeigt die aus diesem Ziel abgeleiteten Fragestellungen und Metriken jeweils unter Angabe eines einfachen Beispiels.

Fragestellung	**Metrik**	**Beispiel**
Wie viele Issues sind in einen bestimmten Zustand übergegangen?	Number of Transitions	Anzahl Issues im Zustand *new*
Wie alt sind diese Issues?	Age (min, max, avg)	Alter der Issues im Zustand *new*
Wie viele Issues haben einen konkreten Zustand verlassen?	Number	Anzahl der Issues die den Zustand *new* verlassen haben
Wie lange hat es gedauert bis Issues einen bestimmten Zustand verlassen haben?	Elapsed Time (min, max, avg)	Zeit, die Issues im Zustand *new* verbracht haben
Wie viele Issues wechseln von einem konkreten Zustand in einen anderen bestimmten Zustand? (durch eine oder mehrere Transitionen)	Number	Anzahl der Issues die von dem Zustand *new* in den Zustand *ready to review* übergegangen sind (mehrere Transitionen)
Wie lange dauert es bis ein Issue von einem konkreten Zustand in einen anderen bestimmten Zustand wechselt?	Processing Time (min, max, avg)	Zeit, die es benötigt, bis Issues von dem Zustand *new* in den Zustand *ready to review* übergegangen sind (mehrere Transitionen)

Tabelle 2: CQM an einem Beispiel aus der Automobil-Zuliefererindustrie

In unserem konkreten Anwendungsfall ist der Großteil der Metriken einfach in MDX umzusetzen. Es gibt allerdings auch einen Teil an Metriken, die weitaus komplexer sind, da sie die Definition eines Calculated Members, Named Sets oder Funktionen erfordern. Als Beispiel möchten wir hier Metriken nennen, die Prozessdauer von einem Zustand zu einem nicht direkt darauf folgenden berechnen, wie wir es in Tabelle 2 dargestellt haben. Abbildung 2 stellt einen Screenshot des Dashboards dar, das mittels graphischer Darstellung einen Überblick ausgewählter Metriken gibt. Gleichzeitig bietet es die Möglichkeit einer Trendanalyse sowohl im Allgemeinen aber auch auf spezialisierte Weise, nach Produkten, Zeitrahmen und Personen. Das Dashboard ist dahingehend flexibel, als dass es die Möglichkeit bietet, einzelne Ansichten anzupassen.
Das zweite Set an Ansichten, wie in Abbildung 3 gezeigt, stellt Metriken aggregiert dar und bietet dabei die Möglichkeit in einzelne Dimensionen hinein zu zoomen. Dies wird durch die oben erwähnten MDX Anfragen umgesetzt. Die Herausforderung besteht bei der Darstellung der Metriken darin, die Dimensionen und Hierarchieebenen so zu wählen, dass die Tabellen einerseits übersichtlich und andererseits das Hineinzoomen in detaillierte Daten ermöglicht. Auch die Auswahl der Aggregate, ob beispielsweise die Summe, der Durchschnitt oder Maximum verwendet wird, spielt hier eine große Rolle und hängt stark von den anfangs definierten Zielen und Fragestellungen ab.

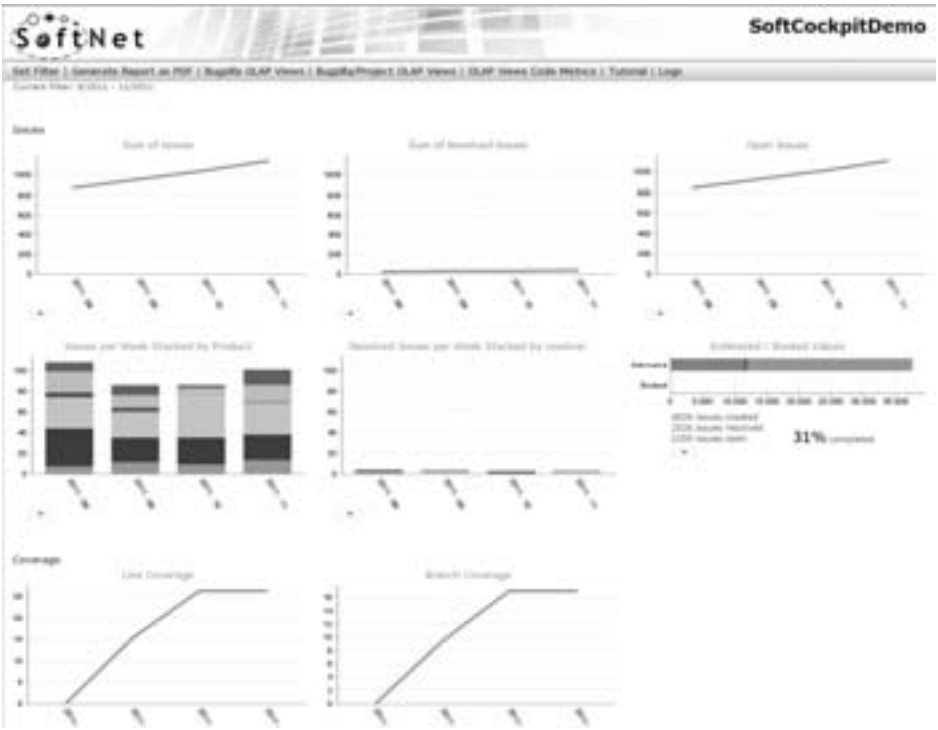


Abbildung 10: Screenshot des Dashboards

So wird in dem Beispiel aus Abbildung 11 die Anzahl der zugewiesenen Vorgänge sowie Minimum, Maximum und Durchschnittsalter dieser Issues in den Dimensionen Zeit x (Produkt und Issue-Art) dargestellt. Da das Alter der Issues in unserem Datenmodell gespeichert ist und die Aggregate im Cube definiert sind, ist dies eine einfache MDX Anfrage, die formal folgender Weise definiert werden kann:
Sei I die Menge aller Issues die im Datawarehouse gespeichert sind. Um Slicer zu beschreiben, die diese Menge nach bestimmten Kriterien $c_0, \cdots, c_l$ einschränken, definieren wir eine Menge $C \subseteq I$ für die gilt $C = \{x | c_i(x) = true, 0 \leq i \leq l\}$.
Zur Darstellung des Drilldowns auf den Spalten 1 bis n, führen wir für jede Spalte eine Untermenge $S_i \subseteq C, 0 \leq i \leq n$ ein, welche die Menge der Issues beschreiben, die das Kriterium der Drilldown-Ebene der Spalte i erfüllen (also beispielsweise in einem gewissen Zeitraum erstellt wurden). Analog dazu definieren wir Mengen $Z_0, \cdots, Z_m$ für die Zeilen 1 bis m.
So können nun die aggregierten Werte in der Tabelle, die sich auf Messwerte $m(x)$ wobei $m: I \mapsto \mathbb{R}$ (z.B. Alter eines Issues) und einer Funktion f (z.B. min, max, sum) beziehen, folgenderweise definiert werden:

$$agg: S_i \times Z_j \mapsto \mathbb{R}$$
$$agg(i,j) = f(\{m(x): x \in S_i \cap Z_j\})$$

Analog dazu kann auch die Wiederverwendbarkeit, wie sie in Abschnitt 3.3 beschrieben wurde, modelliert und nach verschiedenen Gruppierungen, wie beispielsweise Modulen oder Zeitrahmen, abgefragt werden.
Dazu definieren wir eine Messwertfunktion auf der Menge der Klassen K:
Sei $K_a \subseteq K$ die Menge der abstrakten Klassen dann ist $m_a: K \mapsto \{0,1\}$ definiert durch:

$$m_a(x) := \begin{cases} 1 \; iff \; x \in K_a \\ 0 \; else \end{cases}$$

Weiters liefert die von MDX vordefinierte Aggregatfunktion $count: S_i \times Z_j \mapsto \mathbb{N}$ für jede Spalten-Zeilen Kombination die Anzahl der Objekte, die die Kriterien erfüllen, also:
$count(i,j) := |\{x | x \in S_i \cap Z_j\}|$

Die Anzahl der abstrakten Klassen ergibt sich durch folgende Aggregatfunktion:

$$count_a(i,j) := \sum \{m_a(x) | x \in S_i \cap Z_j\}$$

Die Wiederverwendbarkeit ist definiert durch die Anzahl der abstrakten Klassen K_a und der Anzahl der konkreten Klassen K_k als $\frac{|K_a|}{|K_k|}$ also $\frac{|K_a|}{|K|-|K_a|}$.
Somit können wir für die Wiederverwendbarkeit folgende Aggregatfunktion $agg_W: S_i \times Z_j \mapsto \mathbb{R}$ definieren:

$$agg_W(i,j) := \frac{count_a(i,j)}{count(i,j) - count_a(i,j)}$$

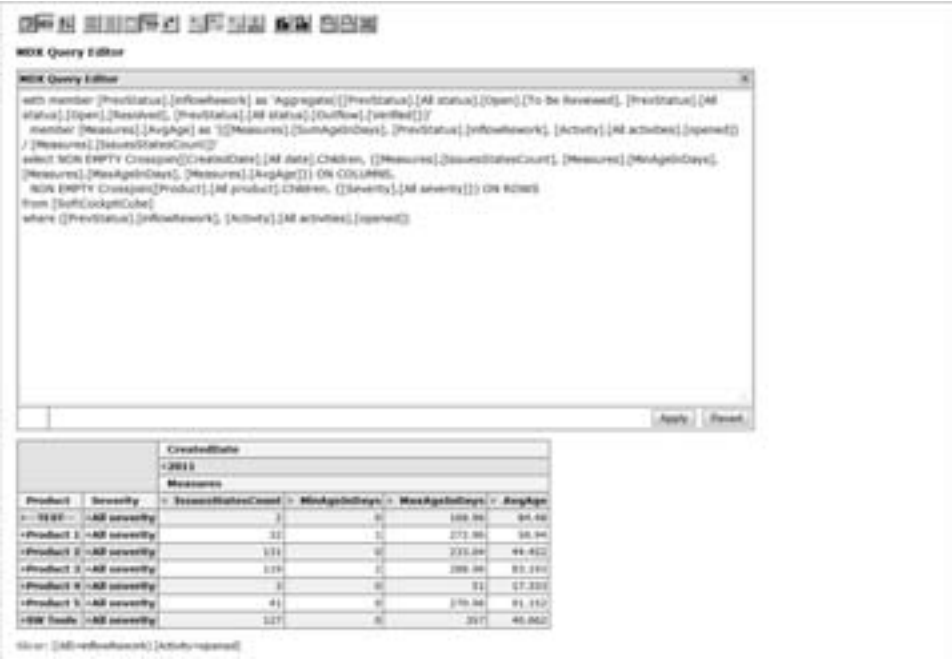
Alternativ könnte man statt der Messwertfunktion auch Members definieren, um die Menge der abstrakten Klassen und die Menge der konkreten Klassen festzulegen, und mit der Hilfe des count Aggregats die Wiederverwendbarkeit definieren.

Das komplexere Beispiel aus Abbildung 4 zeigt exemplarisch die Verwendung einer solchen Memberdefinition am Fallbeispiel des Vorgangsmanagement.

		CreatedDate			
		+2011			
		Measures			
Product	Issue	+ IssuesStatesCount	+ MinAgeInDays	+ MaxAgeInDays	+ AvgAgeInDays
+Product 1	+All Issue	1	0	102.96	39.302
	-All Issue	664	0	907	33.963
	Defect	327	0	207.96	22.306
	Enhancement	144	0	302	24.526
	Ticket	4	0	0	0
Acoustics	-All Issue	1	0	0	0
	Defect	1	0	0	0
	Enhancement	1	0	0	0
Aftertreatment	-All Issue	58	0	214.96	33.378
	Defect	31	0	106.96	18.937
	Enhancement	27	0	214.96	49.059
Analysis	-All Issue	1	0	0	1
	Enhancement	1	0	0	1
Cycle Simulation	-All Issue	95	0	202.96	19.960
	Defect	51	0	202.96	23.755
	Enhancement	34	0	100	14.277
	Ticket	4	0	0	0
EHI	-All Issue	113	0	302	27.381
	Defect	85	0	100	22.613
	Enhancement	24	0	302	40.421
Hydsim	-All Issue	29	0	207.96	11.583
	Defect	7	0	207.96	24.651
	Enhancement	13	0	72.96	6.771
Tools & Methods	-All Issue	2	0	0	0
	Enhancement	2	0	0	0
+Product 3	+All Issue	618	0	283	13.799
+Product 2	+All Issue	996	0	304	20.14
+Product 4	+All Issue	27	0	83	10.319
+Product 5	+All Issue	279	0	229	24.571
+SW Tools	+All Issue	518	0	261	12.127

Slicer: [Activity.managedTo]

Abbildung 11: Drill-Down Ansicht zu Issues, die bereits zugewiesen wurden



MDX Query Editor

```
MDX Query Editor
with member [PrevStatus].[inflowFework] as 'Aggregate({[PrevStatus].[All status].[Open],[To Be Reviewed], [PrevStatus].[All
status].[Open],[Resolved], [PrevStatus].[All status].[Outflow].[Verified]})'
  member [Measures].[AvgAge] as '([Measures].[SumAgeInDays], [PrevStatus].[inflowFework], [Activity].[All activities].[opened])
/ [Measures].[IssuesStatesCount]'
select NON EMPTY Crossjoin({CreatedDate].[All date].Children, {[Measures].[IssuesStatesCount], [Measures].[MinAgeInDays],
[Measures].[MaxAgeInDays], [Measures].[AvgAge]}) ON COLUMNS,
  NON EMPTY Crossjoin([Product].[All product].Children, {[Severity].[All severity]}) ON ROWS
from [SoftCockpitCube]
where ([PrevStatus].[inflowFework], [Activity].[All activities].[opened])
```

Apply Reset

		CreatedDate			
		+2011			
		Measures			
Product	Severity	+ IssuesStatesCount	+ MinAgeInDays	+ MaxAgeInDays	+ AvgAge
+Product 1	+All severity	1	0	102.96	39.302
+Product 1	+All severity	32	1	272.96	38.94
+Product 2	+All severity	131	0	233.04	44.452
+Product 3	+All severity	119	1	288.96	83.193
+Product 4	+All severity	1	0	51	17.333
+Product 5	+All severity	41	0	170.96	41.112
+SW Tools	+All severity	127	0	317	46.862

Slicer: [[All].inflowFework].[Activity.opened]

Abbildung 12: Beispiel einer mdx query unter Verwendung einer Member Definition

5 Diskussion und verwandte Forschungs- und Innovationsprojekte

Fallstudien im Bereich Monitoring von Kennzahlen wurden in den letzten Jahren vermehrt durchgeführt. Larndorfer und Ramler [LRB09a, LRB09b] berichten über die Herausforderungen bei der Einführung von Software Cockpits bei einem Software Produktentwickler und diskutieren auch dysfunktionale Effekte bei der Einführung von Software Dashboards. Das Projekt Soft-Pit [ZM07] zielt – wie die hier präsentierte Fallstudie – auf eine Verbesserung der Prozesstransparenz durch Auswertung von Prozesskennzahlen ab. Ähnlich wie auch das hier präsentierte Cockpit Ansatz stellt auch Soft-Pit eine Integrationsumgebung dar, die den Kristallisationspunkt einer standardisierten Infrastruktur für ein umfassendes Management von Kennzahlen bilden soll [BSRR07]. Jedoch greift Soft-Pit die Integration von Kennzahlen aus verschiedenen Qualitätsdimensionen – Prozesse, Produkte, Ressourcen – kaum auf. Unsere Lösung verfolgt den GQM-Ansatz [BW84], der es ermöglicht, ein für den Anwendungsfall und für die Überwachung von bestimmten Qualitätszielen geeignetes Kennzahlsystem festzulegen. Dabei lässt sich das Kennzahlsystem entlang unabhängiger Hierarchien (Zeit, Module, Versionen, Personalressourcen, …) strukturieren. Die Autoren von [BSRR07] berichten über die Anwendung eines Software Cockpits auf das Software Cockpit selbst und auch über erste Erfahrungen in Pilotprojekten.

Als wichtigster Vorteil gegenüber dem Einzeleinsatz von Analysewerkzeugen wird quellen-übergreifende Erfassung von Daten, die miteinander in Beziehung gesetzt werden können, angeführt womit sich Entscheidungsträger ein umfassendes Bild der Qualitäts- oder Nachhaltigkeitsziele verschaffen können. Wie und mit welcher Granularität die Kennzahlen dabei in Relation gesetzt werden ist rollen-, projekt- und auch branchenspezifisch[3]. Neben der Zeitreihenanalyse haben sich die Tabellendarstellungen in den Pilotprojekten bewährt. Weiter berichten die Autoren, dass die angebotene Drill-Down Fähigkeit nur spärlich benutzt wird. Dies steht im Gegensatz zu unserem Anwender, wo eine Drill-Down Fähigkeit entlang der verschiedenen Dimensionen wie z.B. Produkte oder Versionen gewünscht ist.

5.1 Fazit

In diesem Papier argumentieren wir, dass die Beurteilung der Nachhaltigkeit im Management von Software den gesamten Lebenszyklus der Software erfassen muss. Damit z.B. Behörden und Unternehmen Fragestellungen im Bereich des nachhaltigen Software Managements beantworten können, ist eine kontinuierliche Vermessung der Software über den gesamten Lebenszyklus – Entwurf, Umsetzung, Betrieb und Wartung – vonnöten. Die Erfassung solcher Kennzahlen stellt eine große Herausforderung dar, da Fragestellungen in Bezug auf Nachhaltigkeit das Operationalisieren von produktspezifischen Kennzahlen (z.B. Grad an Wiederverwendbarkeit), Prozessmetriken (z.B. Bearbeitungszeit Change Requests) und Ressource-Metriken (z.B. Genauigkeit der Aufwandschätzung) erfordern.

[3] Diese Sicht wurde in einem informalen Gespräch mit erfahrenen Beratern der Capgemini bei den Software Quality Days in Wien aus praktischer Sicht bestätigt.

Wir stellen ein Software Cockpit vor, dass es ermöglicht, Kennzahlen aus den verschiedenen Bereichen entlang verschiedener Dimensionen (Zeit, Module, Version) zu integrieren. Die so aggregierten Datenbestände erlauben es verschiedene Fragestellungen – auch solche aus dem Bereich nachhaltiges Software Management – qualitativ und quantitativ zu beantworten. Neben der Methodik zur automatisierten Erfassung der Kennzahlen präsentieren wir eine Fallstudie im Bereich des Vorgangsmanagements für eine Software aus der Automobilzulieferindustrie. Daran ist klar zu sehen, dass die für praktische Fragestellungen zu operationalisierenden Kennzahlen ohne flexible Messinfrastruktur nicht zu bewältigen sind. Dies gilt im Speziellen für die vernetzten Fragestellungen aus dem Bereich des nachhaltigen Software Managements.

Literaturverzeichnis

[AADG96] S. Agarwal, R. Agrawal, P. M. Deshpande, A. Gupta, J. F. Naughton, R. Ramakrishnan, S. Sarawagi. On the computation of multidimensional aggregates, Proc. 22nd VLDB, pages 506-521, Mumbai, Sept. 1996.

[AXTLZL10] F. Albertao, J. Xiao, C. Tian, Y. Lu, Q. Zhang, C. Liu, Measuring the Sustainability Performance of Software Projects, Proceedings of IEEE 7th International Conference on e-Business Engineering., IEEE, 2010.

[BD02] J. Bansiya, C. Davis. A Hierarchical Model for Object-Oriented Design Metrics as Quality Assessment, IEEE Transactions on Software Engineering, Vol. 28, Nr. 1, pages 4-17, 2002.

[BSRR07] Marcel Bennicke, Frank Steinbrückner, Mathias Radicke, Jan-Peter Richter: Das sd&m Software Cockpit: Architektur und Erfahrungen. GI Jahrestagung (2) 2007: 254-260.

[BW84] Victor R. Basili, David M. Weiss: A Methodology for Collecting Valid Software Engineering Data. IEEE Trans. Software Eng. 10(6): 728-738 (1984).

[Du01] Duden - Deutsches Universalwörterbuch. 4. Aufl. Mannheim 2001.

[GBLP96] J. Gray, A. Bosworth, A. Layman, H. Pirahesh. Data Cube: A Relational Aggregation Operator Generalizing Group-By, Cross-Tab, and Sub-Totals. Proc. 12th Int'l IEEE Conf. on Data Engineering (ICDE), New Orleans, February/March 1996.

[LRB09a] Stefan Larndorfer, Rudolf Ramler, Clemens Buchwiser: Experiences and Results from Establishing a Software Cockpit at BMD Systemhaus. EUROMICRO-SEAA 2009: 188-194.

[LRB09b] Stefan Larndorfer, Rudolf Ramler, Clemens Buchwiser Dashboards, Cockpits und Projektleitstände: Herausforderung Messsysteme für die Softwareentwicklung, Object Spektrum - Die Zeitschrift für Software-Engineering und -Management, Ausgabe 04 2009.

[Is04] ISO14000, International Organization for Standardisation, 2004 http://www.iso.org. (15.05.2012).

[Ni98] Nils Clausen: OLAP – Multidimensionale Datenbanken. Addison-Wesley-Longman, Bonn 1998, ISBN 3-8273-1402-X.

[Op09] Joanna Chimiak-Opoka. OCLLib, OCLUnit, OCLDoc: Pragmatic Extensions for the Object Constraint Language. In Andy Schuerr and Bran Selic, editors, Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, Colorado, USA, October 4-9, 2009, Proceedings. LNCS 5795, pages 665-669. Springer Verlag, 2009.

[Pe11] Bernhard Peischl, Das Softnet Integrationsprojekt: Kennzahl-gestützte kontinuierliche QS im Software Engineering, Software Quality Days 2011, Wien, Österreich, 18-20, January 2011.

[RBKWL10] Rudolf Ramler, Wolfgang Beer, Claus Klammer, Klaus Wolfmaier, Stefan Larndorfer: Concept, Implementation and Evaluation of a Web-Based Software Cockpit. EUROMICRO-SEAA 2010: 385-392.

[Sn97] H.M. Sneed, Metriken für die Wiederverwendbarkeit von Softwaresystemen,.Informatikspektrum, Vol. 6, pages 18-20, 1997.

[ZM07] Th. Zehler, J. Münch, , Soft-Pit: Ganzheitliche Projekt-Leitstände zur ingenieurmäßigen Software-Projektdurchführung, Frauenhofer IESE, VSEK/051/D, http://www.software-kompetenz.de (15.05.2012).

14 Jahre Intranet der Bertelsmann AG – Resümee und Erfolgsfaktoren

Christian Weber[1], Hans Brandt-Pook[2], Antonia Krieg[2]

[1] Nionex GmbH
Ringstr. 16-20
Rheda-Wiedenbrück
christian.weber@bertelsmann.de

[2] FH Bielefeld
Universitätsstr. 25
33615 Bielefeld
hans.brandt-pook@fh-bielefeld.de
antonia.krieg@fh-bielefeld.de

Abstract: Dieser Beitrag beleuchtet die Erfolgsfaktoren für den nachhaltigen erfolgreichen Betrieb eines Intranets. Dazu werden zunächst aktuelle Anforderungen an Intranets dargelegt. Es wird vorgeschlagen, diese in die Dimensionen Kern-Intranet, Portal, Suche, Collaboration und Social Intranet zu gliedern. Anschließend werden das Intranet der Bertelsmann AG vorgestellt und seine Entwicklung sowie Erfolgsfaktoren für den nachhaltigen Betrieb beschrieben.

1 Einleitung

Intranets bilden heute ein Kernstück in der unternehmensinternen Kommunikation. Oft sind sie sogar Dreh- und Angelpunkt für das Wissensmanagement im Unternehmen. Dieser Beitrag beleuchtet die Erfolgsfaktoren für den nachhaltig erfolgreichen Betrieb eines Intranets aus der Perspektive des IT-Dienstleisters. Er fokussiert auf die Elemente eines modernen Intranets und resümiert langjährige Erfahrungen im Betrieb eines sehr großen Intranets.

Dazu werden im zweiten Abschnitt zunächst aktuelle Anforderungen an Intranets dargelegt, die in einem Modellvorschlag zusammengefasst werden. Der dritte Abschnitt stellt das BENET – das Intranet der Bertelsmann AG – vor. Hier wird aufgezeigt, wie sich das BENET im Laufe der Jahre im Sinne des oben vorgeschlagenen Modells weiterentwickelt hat. Der vierte Abschnitt formuliert – basierend auf dem BENET – Erfolgsfaktoren für einen nachhaltigen Betrieb. Der Beitrag schließt mit einem Fazit und Ausblick.

2 Aktuelle Anforderungen an Intranets

Im Kontext von Internetauftritten bezeichnet ein Intranet eine Website, die nur für die Mitarbeiter eines Unternehmens zugänglich ist[1]. Intranets werden heute auch als Intranet-Portal oder Intranet-Website bezeichnet. Sie sollen den Mitarbeitern eines Unternehmens den zentralen Einstieg zu allen unternehmensinternen Informationen ermöglichen [Ki11].

Intranets werden daher auch dem Thema Wissensmanagement zugeordnet, welches die Identifikation, den Erwerb, die Entwicklung, Verteilung, Bewahrung und Nutzung des Wissens im Unternehmen behandelt [PRR97].

Des Weiteren dienen Intranets der internen Unternehmenskommunikation und informieren die Mitarbeiter durch Mitteilungen und Nachrichten über aktuelle Entwicklungen und Prozesse im Unternehmen. In diesem Sinne haben Intranets das klassische Mitarbeitermagazin abgelöst und sind somit häufig auch organisatorisch im Bereich der internen Unternehmenskommunikation angesiedelt [Wü11].

Da Intranets als Portal einen wesentlichen Beitrag bei der Integration von Prozessen und Anwendungen leisten können [No03], ist das Intranet aus Sicht der Unternehmens-IT eine zentrale Infrastrukturkomponente. Die Basisinfrastruktur eines Intranets basiert daher häufig auf Portalservern, die Prozessintegration (Enterprise Application Integration „EAI“) sowie Zugriffssteuerung durch Single-Sign-On ermöglichen [GH03].

Intranets nehmen zunehmend aktuelle Trends aus dem Web 2.0 auf. Während in den vergangenen zehn bis fünfzehn Jahren Intranets, ebenso wie das Web 1.0, durch einseitige Kommunikation geprägt waren, wandeln sich Intranets heute zunehmend zu sozialen Netzwerken („Social Intranets“) und Plattformen für kollaborative Zusammenarbeit („Collaboration“) [Wo11], [Wü11].

Mit Collaboration ist die webbasierte, häufig in Teams organisierte gemeinsame Bearbeitung von Inhalten gemeint. In diesem Kontext wird in der Praxis auch der Begriff „Virtuelle Teamräume“ [Ko09] genutzt, die sich besonders dann anbieten, wenn Teams räumlich getrennt sind oder sich aufgabenbezogen zusammen finden.

Social Intranets gehen über das Prinzip Collaboration hinaus und bieten Funktionen wie Nutzerprofile, Empfehlungen und Aktivitäten. Sie ermöglichen den Aufbau von unternehmensinternen sozialen Netzwerken und können neue Formen der Unternehmenskultur bewirken.

In der Praxis spiegeln Intranets verschiedene der hier genannten Aspekte wider. Ausgehend von einem „Kern-Intranet“, das die zentralen Aufgaben der internen Unternehmenskommunikation und Informationsbereitstellung bietet, sind Ausprägungen in verschiedenen Dimensionen möglich [Ca11].

[1] Diese Verwendung des Begriffs „Intranet“ geht über die ursprüngliche Nutzung hinaus, die mit Intranet ein Rechnernetz bezeichnet, das nur innerhalb eines Unternehmens erreichbar ist.

Zur Zusammenfassung der aktuellen Diskussion bietet sich eine Strukturierung in die fünf Dimensionen Kern-Intranet, Suche, Portal, Social Intranet und Collaboration an (sh. Abbildung 1). Zunächst stellen wir diesen Modellvorschlag vor und erläutern dann am Beispiel des Intranets der Bertelsmann AG seine flexible Implementierung.

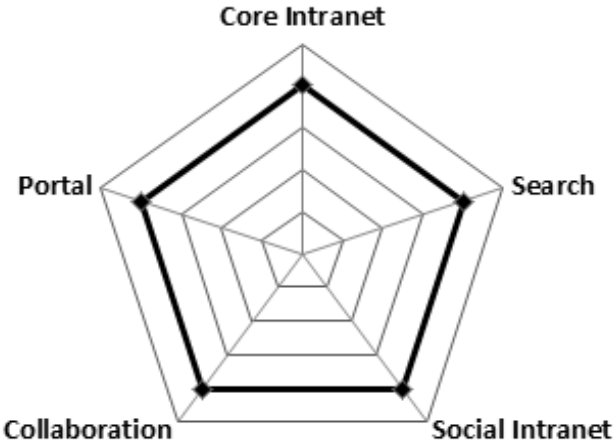


Abbildung 13: Ausprägungen von Intranets

Kern-Intranet

Das Kern-Intranet erfüllt die klassischen Anforderungen an ein Intranet, die sich aus dem Auftrag der internen Unternehmenskommunikation ergeben. Das Kern-Intranet folgt dem Prinzip der Push Kommunikation. Ein relativ kleiner Kreis von Redakteuren, also Intranetnutzern, die Inhalte erstellen, liefert Informationen für einen großen Kreis von Mitarbeitern [Wü11].

Wesentliche Bestandteile von Kern-Intranets sind Informationen für Mitarbeiter, über das Unternehmen und über die vom Unternehmen produzierten Produkte und Dienstleistungen. Durch ein Kern-Intranet soll auch die Identifikation der Mitarbeiter mit dem Unternehmen gefördert werden, in dem die Leitlinien der Unternehmenskultur wiedergegeben werden [Wü11]. Außerdem dient es als „Single Point of Truth“, also als die eine Informationsquelle im Unternehmen, die verbindliche und aktuelle Informationen liefert.

Da sich Strukturen, Funktionen und Inhalte eines Kern-Intranets wenig von klassischen Webseiten unterscheiden, werden Kern-Intranets typischerweise mit Web Content Management Systemen (WCMS) umgesetzt. Diese bieten die optimalen Funktionen um Inhalte strukturiert zu erfassen und über Templates zu veröffentlichen. Bei Bedarf auch für unterschiedliche Ausgabekanäle wie Web und Mobile. Außerdem ermöglichen WCMS eine effiziente Verwaltung und Erstellung von mehrsprachigen Inhalten, was eine wichtige Voraussetzung für Intranets in internationalen Unternehmen ist.

Moderne WCMS bieten außerdem einen hohen Grad an Komfort („Usability"), wodurch die Erstellung von webbasierten Inhalten für Redakteure deutlich erleichtert wird.

Portal

Portale, und hier im engeren Sinne Unternehmensportale, die sich an Mitarbeiter richten, sind die direkte Weiterentwicklung von Intranettechnologien. Als wesentliches neues Merkmal zur Informationsverarbeitung kommt die Prozessorientierung hinzu [GH03]. Intranets, die die Rolle des Unternehmensportals einnehmen, reichen von einer Arbeitshilfe für die Mitarbeiter bei ihrer täglichen Arbeit bis hin zu einem „Workplace", der das alleinige Arbeitswerkzeug für Mitarbeiter darstellt [KGH04], [No03].

Merkmale von Portalen sind insbesondere:

- Single Point of Entry: Das Portal ist der zentrale Einstiegspunkt, um alle Anwendungen und Prozesse des Unternehmens zu erreichen. Dies kann in einfachster Form durch Hyperlinks erfolgen und in der maximalen Ausbaustufe in der direkten Integration sämtlicher Prozesse als Webanwendung im Portal.

- Personalisierte rollenbasierte Informationsbereitstellung, d.h. Auslieferung von Inhalten abhängig von den Rechten des Intranetnutzers.

- Individualisierung: Portale weisen häufig Funktionen für die individualisierte Bereitstellung von Informationen auf, unabhängig von der jeweiligen Rolle des Benutzers.

Technologisch erfolgt die Umsetzung eines Unternehmensportals häufig durch Portal Server. Referenzarchitekturen beschreiben Funktionen wie Rechte- und Benutzerverwaltung, Prozessunterstützung, Single-Sign-On, Transaktions- und Integrationsdienste. Portal Server, die diese Anforderungen erfüllen, werden auch häufig den Themen „Enterprise Application Integration" (EAI), „Service Oriented Architecture" (SOA) oder „Business Process Integration" (BPI) zugeordnet. Nennenswerte Beispiele sind die Produkte SAP Portal, Microsoft Sharepoint und IBM Websphere Portal sowie Lifreray Portal [KGH04], [No03].

Search

Neben der Navigation ist die Suchfunktion im Intranet eine wichtige Funktion, um Inhalte für den Nutzer auffindbar zu machen. Eine Suchfunktion in einfachster Form bietet Volltextsuche in sämtlichen Intranetinhalten, sowohl Seiteninhalte als auch Dokumente in üblichen Dateiformaten. Funktionen, die eine Suche außerdem erfüllen sollte, sind:

- „type ahead search" oder auch „auto completion", d.h. der Benutzer erhält bereits während der Eingabe des Suchbegriffs Vorschläge für die Vervollständigung des Suchbegriffs, die auf möglichen Treffern basieren

- „spell checking" / „Rechtschreibkorrektur", d.h. der Benutzer erhält alternative Suchbegriffe mit ähnlicher Schreibweise vorgeschlagen, die zu einer besseren Trefferzahl führen

- „facetted search" / „Facettierte Suche", d.h. über Kategorien ist ein inkrementelle Verfeinerung des Suchergebnisses möglich („Drill Down")

Soll die Suche über das reine Intranet hinaus auch Inhalte aus anderen Informationsquellen durchsuchen so erweitert sich das Thema zur „Enterprise Search". Die Suche kann als Bestandteil einer Portalinfrastruktur bezeichnet werden [KGH04]. In diesem Sinne bietet eine unternehmensweite Suche ebenfalls einen Single Point of Entry zu allen Informationen an. Diese können in der einfachsten Ausbaustufe über mehrere Informationssilos verstreut sein (unterschiedliche Datenbanken, Fileserver etc.) bis hin zu dem Fall, in dem in einem Unternehmensportal sämtliche Prozesse und Inhalte konsolidiert sind.

Eine Suchfunktion ist häufig Bestandteil einer Standardsoftware für die Umsetzung von Intranets, z.B. von Web Content Management Systemen oder Portal Servern. Darüber hinaus existieren spezialisierte Suchlösungen, sog. „Enterprise Search Solutions", die auf die Suche in unterschiedlichsten Informationsquellen spezialisiert sind.

Um eine Intranet-Suche langfristig und nachhaltig sinnvoll einsatzfähig zu halten, ist die Umsetzung einer Enterprise Search Strategie erforderlich. Durch organisatorische Begleitmaßnahmen muss eine Suche regelmäßig optimiert und verbessert werden.

Collaboration

Collaboration versammelt Methoden wie Online-Meetings, Desktop-Sharing, Video-Konferenzen, Instant Messaging, Wikis und File-Sharing. Sie fördern die Zusammenarbeit in virtuellen oder aufgabenbezogenen Teams. Informationen werden häufig ad hoc erstellt und haben eine begrenzte Lebensdauer. Problematisch ist die Überführung der Inhalte in langfristig für das Unternehmen gesichertes Wissen, da die Inhalte typischerweise unstrukturiert sind.

Social Intranet

Social Intranets nutzen social software, um jegliche Art von Inhalten innerhalb einer Organisation zu teilen [Ca11]. Sie beziehen also alle Mitarbeiter als wesentliches Element eines Intranets mit ein. Voraussetzung dafür sind Mitarbeiterprofile, die jedem Mitarbeiter ein Gesicht geben und über seine Position, Kontaktdaten und Zuständigkeiten informieren. Soziale Intranets fördern den Wissens- und Ideenaustausch durch Kommentarfunktionen, Foren und Abstimmungstools („Like Button", „Voting"). Auch Elemente von Collaboration, z.B. Wikis und Blogs („Online-Tagebücher") sind typische Medienformen.

Durch sog. Aktivitätenströme („Activity Streams") werden die Beiträge eines Nutzers für jeden anderen Intranetnutzer transparent dargestellt. Für Mitarbeiter wird es dadurch einfacher, Ideen auszutauschen oder den richtigen Ansprechpartner, vor allem in großen Organisationen, zu finden.

Ausprägungen

In der Regel werden nicht alle fünf Dimensionen gleichgewichtig entwickelt. Die Stärke der verschiedenen Ausprägungen von Intranets hängen von verschiedenen Faktoren ab. Für eine besonders starke Ausprägung des Social Intranets seien hier beispielhaft zwei Aspekte skizziert:

- Ein wesentlicher Faktor ist die Unternehmenskultur: Die Entwicklung eines Social Intranets passt besonders gut zu einer sehr offenen, kommunikationsorientierten Unternehmenskultur. Werden gewünschte Change Prozesse in Richtung einer solchen Kultur allerdings nur technologisch durch das Intranet vorangetrieben, sind sie häufig zum Scheitern verurteilt. [SH11]

- Sehr sorgfältig müssen auch die Belange des Datenschutzes beachtet werden, da Soziale Intranets unter Umständen auch eine neue Form der Überwachung von Mitarbeitern ermöglichen. Unternehmen mit einer konservativen Unternehmenskultur werden hier abwarten, wie sich die Trends aus dem Web 2.0 weiter durchsetzen.

Aus der Perspektive des IT-Dienstleisters ist zum Betrieb eines Intranets in allen Ausprägungen häufig ein Mix von Standardsoftware und Individualentwicklung erforderlich, da alle Funktionen nach heutigem Stand nicht von einer einzelnen Standardsoftware geliefert werden können. Hier sind daher „best of breed" Ansätze erforderlich, die entsprechende Budgets und Projektsteuerungsfähigkeiten der Organisation erfordern.

3 Das Intranet der Bertelsmann AG

Das Intranet der Bertelsmann AG (BENET) ist das zentrale Kommunikationsmedium des Vorstands an die Mitarbeiter. Weltweit sollen über 100.000 Mitarbeiter über Neuigkeiten aus dem Unternehmen, über die Unternehmenskultur und über Dienstleistungen des Konzerns für seine Mitarbeiter informiert werden. Außerdem dient das BENET als Informationskanal für Führungskräfte.

Historie

Der Grundstein für das BENET wurde 1998 in Form eines Intranets für Führungskräfte gelegt. Zwei Jahre später folgte ein zweites Intranet für Mitarbeiter mit News und Mitarbeiterinformationen. Die technische Plattform basierte auf Vignette Story Server.

Abbildung 2 zeigt die Ausprägungen dieser ersten Intranet Version. Das Kern-Intranet sowie Portal und Suche waren aus heutiger Sicht relativ gering ausgeprägt. Collaboration und Social Intranet kamen überhaupt nicht vor.

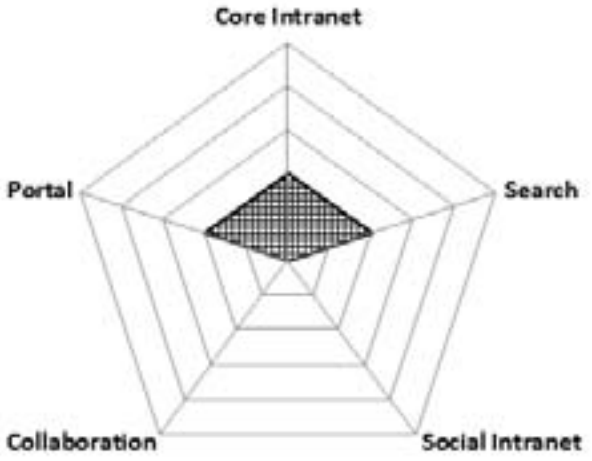


Abbildung 2: BENET 1999 mit gering ausgeprägtem Core Intranet, Portal und Suche

In 2003 wurden die Intranets für Mitarbeiter und Führungskräfte zusammengeführt und unter dem heutigen Namen BENET auf einer einheitlichen Plattform veröffentlicht. In der Folgezeit wurde das BENET regelmäßig um Funktionen erweitert und das Design erneuert.

Zuletzt wurde das BENET nach neun Jahren erneut auf eine neue technische Plattform migriert. Seit 2012 basiert das BENET auf dem CMS FirstSpirit[2]. Mit diesem Launch wurde das Intranet auch um Funktionen für ein Social Intranet erweitert und die Suche wurde drastisch verbessert (siehe Abbildung 3).

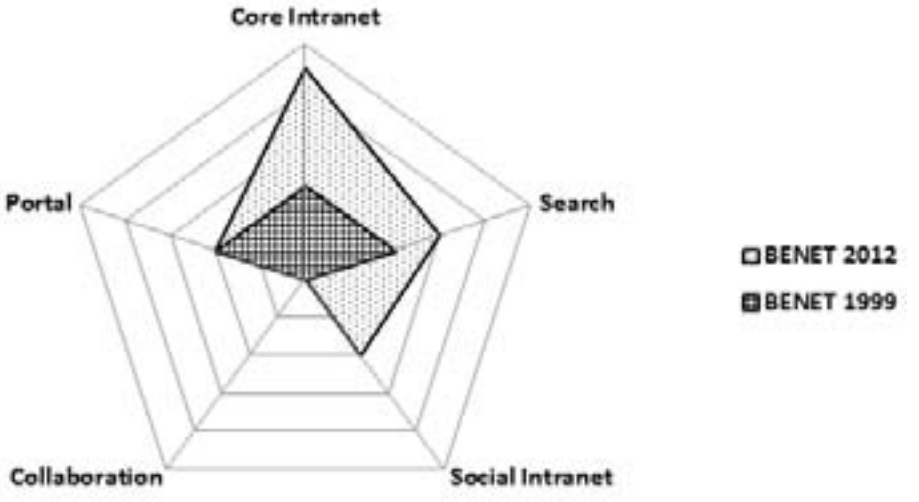


Abbildung 3: BENET 2012 im Vergleich zu 1999 mit stärker ausgeprägtem Kern-Intranet und Suche sowie neuen Social Intranet Funktionen.

Ausblick

Der Kommunikationsauftrag des BENET fokussiert sich auf die Anforderungen des Kern-Intranets. Die Suche umfasst lediglich Intranet-Inhalte und adressiert nicht das Thema Enterprise Search. Collaboration spielt derzeit keine Rolle bzw. wird über andere Plattformen ermöglicht. Erste Schritte in Richtung Social Intranet sowie Portal als zentraler Zugangspunkt werden jedoch abgebildet und sollen in Zukunft stärker ausgebaut werden. Im Mittelpunkt steht weiterhin die zentrale Unternehmenskommunikation, die jedoch durch Elemente eines Social Intranets an Attraktivität und somit Reichweite gewinnen soll. Ein weiteres Entwicklungsthema befasst sich mit der Usability die durch ein erneutes Redesign verbessert werden soll – dies gilt insbesondere auch für die Nutzung des Intranets auf mobilen Endgeräten.

[2] Siehe http://www.e-spirit.com/de/, Zugriff 14.8.2012

4 Erfolgsfaktoren

Das BENET wird seit 14 Jahren betrieben und ist mittlerweile ein konstanter Faktor in der internen Unternehmenskommunikation. Rückblickend lassen sich Erfolgsfaktoren benennen, die zum einen in der Organisation der Betreuung liegen und zum anderen in dem zugrunde liegenden technischen System begründet sind.

Intranet Team

Eigentümer des BENET und inhaltlich verantwortlich ist das Intranet Team der internen Kommunikation innerhalb der Unternehmenskommunikation. Die Aufgaben dieses Redaktionsteams umfassen:

- Erstellung von täglich aktuellen Nachrichten
- Definition der Inhaltsstruktur und des Funktionsumfangs
- Steuerung des technischen Dienstleisters und Koordination der Erweiterungen
- Unterstützung und Kanalisierung der Redakteure mit Beiträgen aus den Fachabteilungen.

Durch das Intranet Team wird die Attraktivität und inhaltliche Qualität des Intranets dauerhaft gesichert. Die Rolle des Intranet Teams als notwendiges Element für eine nachhaltige Verbesserung belegt auch die umfassende Nielsen Intranet Studie [SCN11, S. 13].

Kontinuierliche Verbesserungen

Ein Intranet ist kein statisches Softwaresystem. Kontinuierliche Verbesserungen, in Form von neuen Funktionen und regelmäßigen Redesigns sind wichtig für die Nutzerakzeptanz und den Erhalt des Know-Hows im Betreuungsteam. Vorbild sind hier die großen Portale im Internet, z.B. Google, Amazon oder Ebay, die durch kontinuierliche Verbesserungen punkten anstatt mit großen Relaunches.

Kontinuierliche Verbesserungen erfordern eine entsprechende Budgetplanung, ein permanentes Projektmanagement und kontinuierlich verfügbare Ressourcen zur Umsetzung. Die Voraussetzung hierfür können durch den Einsatz eines Intranet Teams und eines Service Management erreicht werden.

Service Management

Verantwortlich für den technischen Betrieb und die Umsetzung von Erweiterungen ist der Service Manager des technischen Dienstleisters. Der Service Manager stellt das „One Face to the Customer" dar, ist also Hauptansprechpartner für das Intranet Team in allen technischen Belangen, egal ob diese den Betrieb oder die Softwareentwicklung betreffen.

Die Rolle des Service Managers entspricht der Rollenbeschreibung eines Service Advisors, wie sie vom BITKOM definiert wurde [Ro08]. Als Verantwortlicher für den Betrieb koordiniert er sämtliche Änderungen und Probleme an der Infrastruktur und stimmt diese mit dem Eigentümer des BENET ab.

Bei anhaltenden Betriebsstörungen, die das Betriebsteam nicht sofort lösen kann, koordiniert er die Störungsbehebung und zieht ggf. weitere Experten, z.B. aus der Anwendungsentwicklung hinzu. Des Weiteren ist der Service Manager für das Reporting der Betriebskennzahlen an den Kunden verantwortlich.

Content Management System

Als Standardsoftware für den nachhaltigen Aufbau und Betrieb eines Intranets empfiehlt sich der Einsatz eines Content Management Systems (CMS). Ein CMS ist optimiert für die Erfassung, Verwaltung und Veröffentlichung von Inhalten und unterstützt daher die elementaren Anforderungen an ein Intranet optimal.

Um eine nachhaltige Verwendung von Inhalten sicher zu stellen ist die strukturierte Erfassung und Verwaltung erforderlich. Als Basistechnologie hat sich hier XML etabliert. Mit Hilfe von XML können Inhalte in ihre Informationsbausteine zerlegt und mit Metadaten angereichert werden. Durch die Transformation auf Basis von XML ist eine Veröffentlichung in verschiedene Ausgabeformate möglich. Dies umfasst neben Layoutänderungen bei einem Redesign auch zukünftige Ausgabekanäle, z.B. mobile Endgeräte.

4 Fazit und Ausblick

Dieser Beitrag hat die verschiedenen Ausprägungen von Intranets aufgezeigt und am Beispiel des Bertelsmann Intranets Erfolgsfaktoren für einen langfristigen nachhaltigen Betrieb benannt. Auf Basis der aktuellen Diskussion und 14 Jahren BENET Erfahrung lässt sich folgendes Fazit ziehen:

- Ein Intranet Kernteam und ein Service Management schaffen organisatorische Rahmenbedingungen, die einen langfristigen und nachhaltigen Betrieb und den dafür erforderlichen Erhalt des Know-Hows ermöglichen

- Kontinuierliche Verbesserungen und der Ausbau von Intranet-Funktionalitäten sorgen für eine nachhaltige Attraktivität des Mediums bei den Nutzern

- Der Einsatz von Standardsoftware in Gestalt eines Content Management Systems mit offenen Schnittstellen ermöglicht er, das Inhalte nachhaltig verfügbar und nutzbar bleiben, auch über lange Zeiträume hinweg.

Mit Hinblick auf die verschiedenen Ausprägungen von Intranets zeichnet sich ab, dass folgende Aspekte die wesentlichen Entwicklungsrichtungen der kommenden Jahre sein werden:

- Social Media und Collaboration Funktionen werden weiter ausgebaut und das Intranet von einem Informationsmedium weiter in Richtung eines digitalen Arbeits- und Kommunikationsmediums entwickeln.

- Suchfunktionen werden aufgrund der steigenden Informationsflut weiter verbessert werden. Insbesondere die Verknüpfung von Suche und Social Media wird hier qualitative Verbesserungen bringen.

- Portale als alleiniger Workplace werden möglicherweise eine Vision bleiben oder nur in klar definierten Arbeitsumgebungen realisierbar sein. Portale werden jedoch eine wichtige Infrastrukturkomponente bleiben und insbesondere die fortschreitende Personalisierung und die mobile Nutzung des Intranets ermöglichen.

Nach dem anfänglichen Intranet Hype der 2000er Jahre trat nach zehn Jahren eine Ernüchterung bei den Intranets ein. Ursachen waren häufig die in die Jahre gekommene Technik und durch fehlende Pflege veraltete Inhalte. Die Folgen sind Desinteresse bei den Nutzern und Resignation bei den Redakteuren. Durch aktuelle Innovationen im Bereich des Social und Mobile Web werden auch Intranets beflügelt und für die nächsten zehn Jahre fit gemacht.

Literaturverzeichnis

[Ca11] Cavazza, F.: From social intranets to collaboration ecosystems, verfügbar unter: http://www.forbes.com/sites/fredcavazza/2011/11/30/from-social-intranets-to-collaboration-ecosystems/ Zugriff: 16.07.2012

[GH03] Gurzki, T.; Hinderer, H.: Eine Referenzarchitektur für Software zur Realisierung von Unternehmensportalen. In: (Reimer, U.; Andreas Abecker, A.; Staab, S.; Stumme, G. Hrsg.): Professionelles Wissensmanagement - Erfahrungen und Visionen: Beiträge der 2. Konferenz Professionelles Wissensmanagement - Erfahrungen und Visionen. Lecture Notes in Informatics (LNI) - Proceedings 28, Bonn, 2003; S. 157-160.

[Ki11] Kiellisch, T.: Die Intranet-Redaktion: Planen, organisieren, strukturieren, Fachartikel, verfügbar unter: http://www.kernpunkt.de/de/presse/fachartikel-intranet-redaktion.html; Zugriff: 11.07.2012.

[KGH04] Kirchhof, A. ; Gurzki, T. ; Hinderer, H. ; Vlachakis, J. : Was ist ein Portal? - Definition und Einsatz von Unternehmensportalen, Whitepaper Frauenhofer Institut Arbeitswirtschaft und Organisation, 2004.

[Ko09] Koch, M.: Von E-Mail zur Social Software. In Fachzeitschrift 'technische kommunikation', 31.Jahrgang, 6/2009, S. 24.

[No03] Nohr, H.: Geschäftsprozessorientiertes Wissensmanagement mit Unternehmensportalen. In: (Reimer, U.; Andreas Abecker, A.; Staab, S.; Stumme, G. Hrsg.): Professionelles Wissensmanagement - Erfahrungen und Visionen: Beiträge der 2. Konferenz Professionelles Wissensmanagement - Erfahrungen und Visionen. Lecture Notes in Informatics (LNI) - Proceedings 28, Bonn, 2003; S. 151-155.

[PRR97] Probst, G; Raub, S.; Romhardt, K.: Wissen managen: Wie Unternehmen ihre wertvollste Ressource optimal nutzen. Gabler, Wiesbaden, 1997.

[SCN11] Schade, A. ; Caya, P. ; Nielsen, J. : Intranet Design Annual 2011 - The Year's 10 Best Intranets, Nielsen Norman Group, 2011.

[SH11] Schopp, B.; Hörner, B.: One size fits all? Warum das Intranet zur Organisation passen muss. In: Netzwoche 07/2011, S. 25-28.

[Wo11] Wolf, F.: Social Intranet – Kommunikation fördern, Wissen teilen, Effizient zusammenarbeiten, Carl Hanser Verlag, München, 2011.

[Wü11] Wünsche, V.: Intranet – die Plattform für unternehmensweite Prozesse und Kommunikation. In: IHK Magazin Wirtschaft, Nr. 07/2011, Stuttgart, 2011, S. 53- 55.

Einführung eines nachhaltigen IT Service Managements an der FH Bielefeld

Thomas Degenhardt, Michael Korff, Ulrich Schäfermeier

Fachhochschule Bielefeld
Universitätsstraße 25
33615 Bielefeld
thomas.degenhardt@fh-bielefeld.de
michael.korff@fh-bielefeld.de
ulrich.schaefermeier@fh-bielefeld.de

Abstract: Mit der Einführung seines IT Service Managements (ITSM) verfolgen Unternehmen das Ziel, an den Geschäftsprozessen ausgerichtete IT-Infrastrukturen kosten- und leistungsoptimiert einzusetzen[1]. Dieser Anspruch, die Wertschöpfung von Unternehmen seitens der IT-Bereiche nachhaltig zu unterstützen, trifft in realen Umsetzungsprojekten auf verschiedenste Probleme. Am Beispiel der Einführung von ausgewählten Prozessen des ITIL V3 an der Fachhochschule Bielefeld, mit ihrer von Dezentralität geprägten IT-Organisation, zeigt der folgende Beitrag die Eckpunkte eines stufenweise eingeführten und effektiven IT Service Managements auf. Die Einführung hat die Hochschule in die Lage versetzt, sowohl eine umfangreiche Standardisierung in der Service-Bereitstellung als auch eine Entlastung bei den IT-bezogenen Routinetätigkeiten herbeizuführen. Zudem wurden die Grundlagen für die nachhaltige Konsolidierung des ITSM im Zuge des für 2013 geplanten Umzugs auf einen gemeinsamen Campus gelegt.

1 Ausgangslage und Problemstellung

Die FH Bielefeld ist mit derzeit ca. 8.350 Studierenden eine der größten Fachhochschulen Deutschlands. An elf Standorten in drei Städten arbeiten 190 Professoren und 278 Mitarbeiter organisiert in sechs Fachbereichen sowie den fünf Dezernaten der Hochschulverwaltung[2]. Diese Dezentralität impliziert eine heterogene IT-Organisation in Matrix-Form[3]: Jeder Fachbereich und Standort hält eigenes IT-Personal vor, dass die lokalen Anforderungen an die Infrastruktur und IT-Services erfüllen muss. Eine zentrale IT-Einheit, die DV-Zentrale (DVZ), unterstützt die Fachbereiche als Shared Service Center zudem mit Standardleistungen (Commodities) wie hochschulweiten Beschaffungen, Rechenzentrumsdienstleistungen und allgemeinen Systemen wie Email-Server, Verzeichnisdienst, eLearning-Plattform etc. Einige dieser Dienstleistungen werden aber

[1] Vgl. [Ti09], Seite 129.
[2] Aus [FH12], Stand 20. April 2012.
[3] Vgl. [HS07], Seiten 92f.

auch in den Fachbereichen bzw. Standorten dezentral verwaltet oder ihr Betrieb operativ unterstützt. Die Kosten für hochschulweite Systeme und IT-Dienste trägt i. A. die DVZ. Eine volumenbasierte Weiterbelastung sämtlicher IT-Leistungen an die Fachbereiche und Dezernate in Form einer Leistungsverrechnung findet nicht statt, obwohl das zentrale Budget der DVZ häufig kritisiert wird.

Aufgrund der einheitlichen Struktur der öffentlichen Fachhochschulen und Universitäten in Deutschland ist dieses Spannungsfeld zwischen zentralen sowie dezentralen und weitgehend autarken IT-Organisationseinheiten keine Besonderheit der FH Bielefeld[4]. Zudem lässt sich die Ausgangslage vergleichen mit der von Konzernstrukturen, in denen die IT-Dienstleistungen sowohl durch zentrale als auch dezentrale Organisationseinheiten erbracht werden.

Diese dezentralen Organisations- und Dienstleistungsstrukturen im IT-Bereich bringen einige tiefgreifende Probleme mit sich:

- Intransparente Ansprechpartner für die Nutzer der IT-Systeme. Im Falle der FH Bielefeld trifft das insbesondere für die Studierenden und Mitarbeiter der Fachbereiche zu.

- Vorhalten redundanter Funktionsträger

- Schwierige Durchsetzbarkeit einer einheitlichen Ablauforganisation bzw. von Prozess- und Systemstandards[5] und mithin Verbesserungspotenzial bei der Ausnutzung von Skalenvorteilen in der Beschaffung und dem IT Service Management

- Keine verursachungsgerechte Leistungsverrechnung und damit kaum Anreize zur Kostendisziplin

- Fehlinvestitionen durch Orientierung an Budgets, die nicht immer die Erfordernisse der Geschäftsprozesse widerspiegeln, in Falle der Hochschule die der Lehre und Forschung

- Inhomogene Infrastruktur mit eingeschränkter Unterstützung mobiler Nutzer, die zwischen Standorten wechseln müssen

Die großen FH-Standorte in Bielefeld werden Mitte 2013 auf dem neuen Campus geografisch zusammengeführt werden. Die Fachbereichsstruktur bleibt bestehen, dennoch eröffnet dieser Umzug zahlreiche Verbesserungsmöglichkeiten für die IT-Dienstleistungen und deren Erbringung. Insbesondere bietet es sich an, die Dienstleistungen zu vereinheitlichen und zentrale Kanäle für die Kommunikation mit den Benutzern zu schaffen um sowohl Skaleneffekte auszunutzen als auch die Zufriedenheit der

[4] Z. B. beschreibt [Vo10], Seiten 27ff, eine ähnliche Ausgangslage der TU München.
[5] Auch als IT-Governance bezeichnet. IT-Governance umfasst *„… die Organisation, Steuerung und Kontrolle der IT eines Unternehmens zur konsequenten Ausrichtung der IT-Prozesse an der Unternehmensstrategie…“*. Aus [FG07], Seite 17.

Kunden bzw. IT-Anwender zu erhöhen. Eine dem Umzug zeitlich vorgelagerte und nachhaltige[6] Harmonisierung des ITSM wurde daher seitens der DVZ bereits 2009 als Voraussetzung für den Umzug definiert. Aufgrund der Bedeutung und Verbreitung entschied sich die FH Bielefeld, geeignete Prozesse der IT Infrastructure Library Version 3 (ITIL V3) in mehreren Phasen vor und nach dem Umzug einzuführen. Die Einführung musste dabei so nachhaltig erfolgen, dass die eingeführten Abläufe bzw. Prozesse, Aufbauorganisationen und Systeme sowohl in der jetzigen dezentralen als auch der zukünftigen zentralisierten Umgebung die o. a. Probleme beseitigt. Dabei sollte die Umsetzung derjenigen Prozesse vorgezogen werden, die eine Homogenisierung der IT-Infrastruktur und der IT-Dienstleistungen fördern, um die Transition auf den neuen Campus zu erleichtern.

Weitere umzugsbedingte Projekte und Maßnahmen, wie die Einführung von IPv6, werden im Rahmen der nachfolgenden Ausführungen aus Gründen der Lesbarkeit und Vereinfachung nicht angeführt.

Trotz der Planung des Ablaufs um den wichtigen Meilenstein des vollzogenen Umzugs in den neuen Campus, lässt sich diese Ausgangslage übertragen auf Organisationen, denen eine Umstrukturierung bevorsteht: Die ITSM-Prozesse und -Funktionen müssen so nachhaltig aufgebaut werden, dass diese ausreichend robust für zukünftige Umstrukturierungen sind bzw. diese unterstützen.

Zudem empfiehlt die Kommission für IT-Infrastruktur der Deutschen Forschungsgemeinschaft Hochschulen für die Unterstützung der Geschäftsabläufe ein ITSM, damit die IT-Dienste entsprechend der Anwenderbedürfnisse bereitgestellt werden können.[7] Entsprechend dieser Empfehlung wird die Bedeutung eines ITIL-konformen Managements der IT-Dienste in Hochschulen für Kooperationen mit Industriepartnern und für die flexible Ausgestaltung des Technologieeinsatzes zunehmen.

[6] Nachhaltig im Sinne einer auch noch nach dem Umzug tragfähigen und sich verbessernden Servicestruktur.
[7] Vgl. [DF11], Seite 28.

2 Prozesse des ITIL V3

Der seit 2007 gültige ITSM-Standard der britischen Central Computer and Telecommunication Agency ITIL V3[8] ist „*ein Satz an Best Practice Leitlinien für das IT Service Management.*“[9] Mit ITIL V3 sind, in fünf Bänden gruppiert, für die 22 Prozesse des ITSM Ziele, Inhalte, Vorgehensweisen, Strukturen und Rollen beschrieben. Zudem sind einige für das ITSM notwendige Einrichtungen bzw. Funktionen einer Service-Organisation gegliedert. Der Ausbau der Darstellung orientiert sich an dem Lebenszyklus von IT-Services[10]:

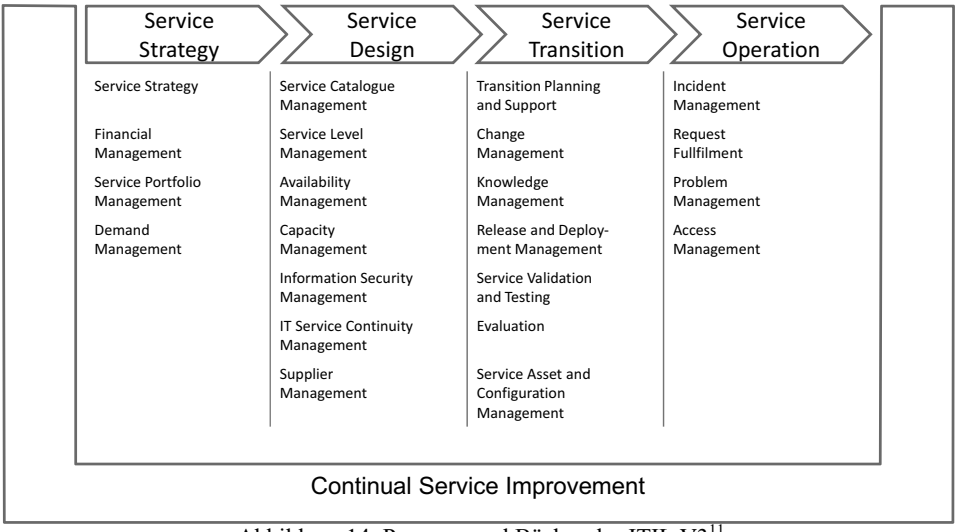


Abbildung 14: Prozesse und Bücher des ITIL V3[11]

Neben den o. a. Prozessen definiert der Band „Service Operation“ folgende Funktionen bzw. Organisationseinheiten[12]:

[8] Vgl. [JG11], Seite 196.
[9] Aus [OG07a], Seite 283.
[10] Sämtliche Prozesse, Strukturen, Rollen, Funktionen und Messgrößen sind auch in der deutschsprachigen Referenzliteratur [OG07a], [OG07b], [OG07c], [OG07d] und [OG07e] mit englischen Begriffen belegt und werden im Folgenden – wie in ITSM-Publikationen üblich – nicht ins Deutsche übertragen, um den Sinn nicht zu verstellen. Wichtige Begriffe werden in Fußnoten erläutert.
[11] In Anlehnung an die Struktur und Darstellung der ITIL Referenz Literatur [OG07a], [OG07b], [OG07c], [OG07d] und [OG07e]. Siehe auch Anmerkungen zu den Begriffen in Fußnote9.
[12] Vgl. [OG07d], Seiten 124ff.

- *Service Desk*
  Das Service Desk dient als zentralisierte Schnittstelle zwischen den Konsumenten von IT-Dienstleistungen bzw. Anwendern einerseits und der IT-Service-Organisation andererseits.[13]

- *Technical Management*
  Das Technical Management stellt eine anforderungsgerechte technische Infrastruktur von der Planung bis zur Verwaltung bereit.[14]

- *IT Operations Management*
  Die Funktionen des IT Operations Management verfolgen das Ziel, die vereinbarten Bereitstellungsgüten[15] der IT Services zu erreichen und regelmäßig zu überprüfen. Das IT Operations Management gliedert sich in die Funktion IT Operations Control und das Facilities Management.[16]

- *Application Management*
  Das Application Management verantwortet das Management der IT-Anwendungen über den gesamten Software Lebenszyklus und unterstützt damit etliche ITSM-Prozesse.

Die Implementierung eines ITSM auf der Basis des ITIL V3 erfolgt i. A. schrittweise, wobei häufig nicht alle Prozesse umgesetzt werden[17]. Als die die Umsetzungstiefe und Vorgehensweise beeinflussenden Aspekte der Einführung werden genannt[18]:

- Qualifikation der IT-Mitarbeiter

- Strategie des IT-Bereichs im Hinblick auf zukünftig fokussierte ITSM-Prozesse und IT-Dienstleistungen

- Strategie der Fachbereiche bzgl. angestrebter Schwerpunktsetzungen bei den Geschäftsprozessen

- Homogenität und Schnittstellen bzw. Querbeziehungen der ITSM-Prozesse untereinander

- Bestehende Schwachpunkte der IT-Service-Organisation

[13] In Organisationen ist der Service Desk im Allgemeinen unter einheitlichen Rufnummern und Email-Adressen für alle IT-bezogenen Probleme zu erreichen.
[14] Das Technical Management ist daher die funktionale Organisationseinheit, die IT-Infrastrukturen wie Rechenzentren oder Netzwerke plant und betreibt.
[15] Service Level
[16] Während das Facilities Management die baulichen Rahmenbedingungen für IT-Dienstleistungen wie Gebäude, Klimaanlagen, Stromversorgung etc. bereitstellt, überwacht das IT Operations Control sämtliche IT-Infrastruktur-Komponenten.
[17] Bereits in einer Studie des CIO-Magazins im Jahre 2005 (vgl. [Bu08], Seite 38) zeigte sich, dass der überwiegende Teil der ITIL einführenden Unternehmen zunächst die Einführung eines Service Desks mit dem Incident Management Prozess anstreben. Weitere, nachrangige Einführungsschritte betreffen i.A. das Problem und das Change Management. Keines der befragten Unternehmen hat ITIL vollständig umgesetzt.
[18] Vgl. [Ti09], Seiten 197f.

3 Gewählte Schritte für die Umsetzung

Aus Sicht der FH Bielefeld dominierte für die Entscheidungsfindung einer geeigneten schrittweisen Einführung die Nutzung einheitlicher Prozess- und Systemstandards bei Aufrechterhaltung der dezentralen Verantwortlichkeiten und Entscheidungshoheiten im Sinne der o. a. Nachhaltigkeit. Zudem mussten die Qualifikationen und Arbeitsweisen der durch unterschiedliche fachliche und organisatorische Einbettungen geprägten IT-Mitarbeiter aufgrund der angestrebten durchgängigen Prozessstandards im ITSM ausgebaut werden. Daher wurden zunächst die erläuterten vordringlichen Probleme der hergebrachten ITSM-Organisationen adressiert und im Hinblick auf den zukünftigen gemeinsamen Campus als wesentlichen Eckpunkt eingeplant.

Die Interdependenzen zwischen den Prozessen des ITIL V3[19] geben zudem Bedingungen für die Reihenfolge der Umsetzung von Prozessen im Rahmen einer schrittweisen Einführung vor. So ist zum Beispiel ein Knowledge Management im ITSM sinnlos, wenn noch keine die das Knowledge Management anwendenden Prozesse wie das Incident und das Problem Management umgesetzt sind.

Diese Erfordernisse führten zu der folgenden Phasenplanung, die weitgehend deckungsgleich ist mit der Reihenfolge der ITIL V3-Umsetzung in Unternehmen[20]:

Abbildung 15: Phasenplanung ITSM-Einführung

Mit der ersten Phase wurde ein hochschulweiter zentraler Ansprechpartner (Single Point of Contact – SPOC) für alle IT-Probleme etabliert. Dazu wurde ein virtueller Service Desk aufgebaut, der alle Incidents[21] aufnimmt, dokumentiert und – im Falle von Standards-Incidents – direkt, d. h. ohne Eskalation an spezialisierte Organisationseinheiten,

[19] Siehe Abbildung 1.

[20] Vgl. [Bu08], Seite 38: Der überwiegende Teil der ITIL einführenden Unternehmen startet mit der Implementierung eines Service Desks mit dem Incident Management Prozess. Es folgt dann das Problem und das Change Management.

[21] Incident: *„Eine nicht geplante Unterbrechung eines IT Service oder eine Qualitätsminderung eines IT Service. Auch ein Ausfall eines Configuration Item ohne bisherige Auswirkungen auf einen Service ist ein Incident."*
Incident Management: *„Der Prozess, der für die Verwaltung des Lebenszyklus aller Incidents verantwortlich ist. Wichtigstes Ziel des Incident Management ist eine schnellstmögliche Wiederherstellung des IT Service für die Anwender."*
Aus [IT07], Seite 23.

löst. Da weder die zentrale Datenverarbeitung der Fachhochschule noch die dezentralen IT-Gruppen in den Fachbereichen ausreichend Kapazität besitzen, um die notwendigen Service-Zeiten abzudecken, wurde ein dezentraler Ansatz gewählt. Äquivalent zum Follow-the-Sun-Prinzip[22] gemäß ITIL[23] agieren mittels eines hochschulweit eingesetzten Ticketing-Systems in einem abgestimmten Zeitplan verschiedenste, örtlich und organisatorisch distribuierte IT-Bereiche als virtueller Service Desk mit hoher Verfügbarkeit und zeitlicher Abdeckung.

Mit der ersten Phase wurden die ITIL-Prozesse Incident und Problem Management[24] umgesetzt. Die operative Incident-Bearbeitung erfolgt durch die Mitarbeiter des Service Desk und wird unterstützt mittels des Open-Source-Systems OTRS[25]. Das System wurde ergänzt durch die Anbindung des zentralen Verzeichnisdienstes der Hochschule als Kundendatenbank. Dadurch und mittels einer sehr detaillierten Abstimmung mit den dezentralen IT-Einheiten konnte ein Workflow zur Abdeckung der beiden Prozesse hochschulweit einheitlich ausgerollt werden:

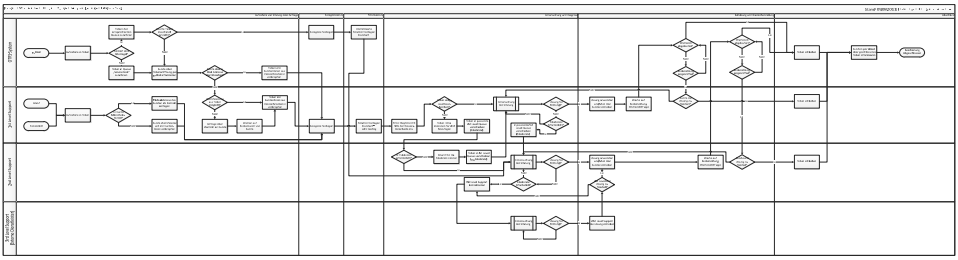


Abbildung 16: Schematische Darstellung des Incident Management Workflows

Im Gegensatz zum Service Desk unterstützen die dezentralen IT-Teams den 2nd Level Support[26] und übernehmen im Wesentlichen das Problem Management sowie die Bearbeitung von Incidents, die sich auf nur lokal vorhandene Systeme beziehen. Somit wurde die Anforderungen nach Dezentralität und Nachhaltigkeit erfüllt: Der virtuelle Service Desk übernimmt das Incident Management und bearbeitet aktiv alle Incidents und Probleme, die hochschulweite Dienste betreffen. Die lokalen IT-Teams fungieren – eingebettet in einheitliche ITSM-Prozesse und Systeme – als 2nd Level Support für fachbe-

[22] Follow-the-Sun-Prinzip: „*Eine Methode, bei der Service Desks und Support-Gruppen weltweit eingesetzt werden, um einen reibungslosen Service 24 Stunden am Tag und an sieben Tagen in der Woche bereitstellen zu können. Anrufe, Incidents, Problems und Service Requests werden zwischen den Gruppen in unterschiedlichen Zeitzonen weitergeleitet.*“ Aus [IT07], Seite 21.

[23] Vgl. [OG07d] Seite 127.

[24] Problem: „*Die Ursache für einen oder mehrere Incidents. Zum Zeitpunkt der Erstellung eines Problem Record ist die Ursache in der Regel unbekannt. Für die weitere Untersuchung ist der Problem Management Prozess verantwortlich.*“
Problem Management: „*Der Prozess, der für die Verwaltung des Lebenszyklus aller Probleme verantwortlich ist. Wichtigstes Ziel des Problem Management ist es, Incidents zu verhindern bzw. die Auswirkungen von Incidents zu minimieren, die nicht verhindert werden können.*“
Aus [IT07], Seite 38.

[25] Open Ticket Request System, vgl. www.otrs.org.

[26] 2nd Level Support: „*Die zweite Ebene in einer Hierarchie von Support-Gruppen, die mit der Lösung von Incidents und der Untersuchung von Problemen befasst sind. Mit jeder Ebene sind mehr Know-how und Fertigkeiten von Experten bzw. mehr Zeit oder weitere Ressourcen verfügbar.*“ Aus [IT07], Seite 43.

reichsindividuelle, die DV-Zentrale als 2nd Level Support für fachbereichsübergreifende Services und Systeme. Diese Struktur ist zudem vorbereitet für den anstehenden Umzug und die damit verbundene Konsolidierung, da auf dem neuen Campus die derzeitigen Service-Desk-Gruppen zu einer Einheit zusammengelegt werden, ohne dass sich für die Anwender und anderen IT-Bereiche die Schnittstellen ändern.

In der derzeit bearbeiteten zweiten Phase erfolgt die Definition, Umsetzung und Messung von Service Level Agreements (SLA)[27] mit den verschiedenen Konsumentengruppen der IT-Services. Vorrangig erfolgt die Definition mit den Dezernaten der Hochschulverwaltung sowie den Fachbereichsvertretern anhand eines Service-Katalogs, der in der Vorbereitung der ITSM-Einführung durch eine studentische Projektgruppe ausgearbeitet und mit den Beteiligten abgestimmt wurde.

Zudem soll zur SLA-Überwachung das OTRS um das ITSM-Modul sowie eine verfeinerte Kategorisierung und Priorisierung von Tickets ergänzt werden. Dieses ist notwendig, um zukünftige SLA messen zu können, da die derzeitige Ausbaustufe des ORTS lediglich die Überwachung von Lösungszeiten und Eskalationsquoten zulässt. Die detaillierte Analyse und Nachverfolgung der Incidents ist Voraussetzung für die langfristig angestrebte verursachungsgerechte Leistungsverrechnung des ITSM.

Eine weitere systemtechnische Ergänzung ist für die dritten Phase projektiert, in der eine Configuration Management Data Base (CMDB)[28] im Rahmen des Prozesses Service Asset and Configuration Management[29] aufgebaut und mit dem Ticketsystem OTRS gekoppelt werden soll. Da die Hochschule in weiten Bereichen bereits über ein Desktop Management System inkl. Verwaltung der IT-Assets[30] verfügt, ist die systemtechnische Implementierung des Service Asset and Configuration Managements in erster Linie eine organisatorische Hürde. Die Umsetzung soll noch vor dem Umzug auf den gemeinsamen Campus erfolgen, da somit auch das gesamte Umzugsmanagement unterstützt wird. So soll beispielsweise auf dem neuen Campus eine auf IPv6 basierende Netzwerkstruktur aufgebaut werden. Die einhergehende Planung und Konfiguration der Clients und Server kann bereits mittels der CMDB vorweggenommen werden.

---

[27] SLA: *„Eine Vereinbarung zwischen einem IT Service Provider und einem Kunden. Das SLA beschreibt den jeweiligen IT Service, dokumentiert Service Level Ziele und legt die Verantwortlichkeiten des IT Service Providers und des Kunden fest. Ein einzelnes SLA kann mehrere IT Services oder mehrere Kunden abdecken.“* Aus [IT07], Seite 44.

[28] CMDB: Eine Datenbank, die verwendet wird, um standardisierte Informationen über alle Komponenten (Configuration Items) einer IT-Infrastruktur und deren Parametrisierung während ihres gesamten Lebenszyklus zu speichern. In Anlehnung an [IT07], Seite 14.

[29] Service Asset and Configuration Management: *„Der Prozess, der sowohl für das Configuration Management als auch das Asset Management verantwortlich ist.“* Aus [IT07], Seite 43. Das Configuration Management wiederum ist die Pflege von Informationen zu allen Komponenten zur Bereitstellung eines IT-Services und deren Beziehungen untereinander über den ganzen Lebenszyklus des IT-Services. Im Gegensatz dazu ist das Asset Management der Prozess zur Verfolgung der Werte und Besitzverhältnisse in Bezug auf finanzielle Ressourcen und Fähigkeiten des IT-Dienstleisters. In Anlehnung an [IT07], Seiten 14 und 3.

[30] Novell ZENworks.

Eine weitere Ausbaustufe betrifft die Einführung des Event Managements[31] mit der Installation entsprechender Überwachungssysteme für die IT-Infrastruktur. Zudem wird das Change Management[32] implementiert, das sowohl zur Durchsetzung der Harmonisierung als auch zur angestrebten Leistungsverrechnung notwendig ist.

Mit dem Umzug auf den neuen Campus und der örtlichen Konsolidierung der Einheiten der FH Bielefeld erschließen sich weitere Potentiale des ITSM. So ist angestrebt, den virtuellen Service Desk zu einer – dann nicht mehr virtuellen – Einheit zusammenzulegen. Damit kann der zentrale IT-Bereich als hochschulweites Shared Service Center fungieren, das mittelfristig eine höher standardisierte IT-Infrastruktur ermöglicht. Zur Sicherung der Interessen der IT-Konsumenten werden dann entsprechende Gremien[33] etabliert.

4 Partizipation der dezentralen IT-Organisationseinheiten an der Umorganisation

Das ITSM-Einführungsvorhaben der FH Bielefeld führt zu zwei Problemen: Zum einen mussten die Fachbereiche als Konsumenten der IT-Services, die gleichzeitig selbst eigene IT-Teams beschäftigen, in die vereinheitlichten Prozesse eingebunden werden. Zum anderen sind die Mitarbeiter der dezentralen IT-Teams von der Vorteilhaftigkeit der über die ITSM-Prozesse indirekten Beauftragung[34] zu überzeugen. Der in vielen Unternehmen gewählte hierarchische Ansatz[35] zur Durchsetzung einheitlicher Prozesse ist aufgrund der autark arbeitenden Fachbereiche nicht möglich.

Daher wurde eine partizipative Strategie gewählt, bei dem die Fachbereiche nicht nur als mitspracheberechtigte Kunden auftreten, sondern aktiv an der Ausgestaltung des ITSM mitwirkten: So wurden Schulungen, Beraterbesprechungen und Präsentationen immer mit Akteuren[36] der Fachbereiche durchgeführt. Zudem wurden einzelne Arbeitspakete durch dezentrale Teams abgearbeitet. Als dritter Baustein der Partizipation verrichten einige Mitarbeiter der DVZ, die aktiv an der Implementierung des ITSM mitwirkten, ihre Aufgaben an in den Fachbereichen verorteten Arbeitsplätzen. Somit wurden Fachbereiche vom Kunden zum Mitgestalter.

---

[31] Event Management: Der Prozess, der für die Verwaltung von Statusänderungen von IT-Komponenten und IT-Services während ihres Lebenszyklus verantwortlich ist. Das Event Management ist eine der wichtigsten Aktivitäten des IT-Betriebs. In Anlehnung an [IT07], Seite 19.

[32] Change Management: Der Prozess, der für die Steuerung des Lebenszyklus aller Änderungen an der IT-Infrastruktur mit Auswirkungen auf die IT-Services verantwortlich ist. Wichtigstes Ziel des Change Management ist es, die Durchführung von lohnenden Changes bei einer minimalen Unterbrechung der IT-Services zu ermöglichen. Zudem ist das Change Management wichtig zur formalen Klärung der Verantwortlichkeiten für Kosten und Risiken. In Anlehnung an [IT07], Seiten 11f.

[33] Insbesondere eines gemeinsamen IT-Lenkungsausschusses.

[34] Die Beauftragung der dezentralen IT-Gruppen erfolgt indirekt über das zentrale Service Desk, d. h. alle Anfragen und Meldungen werden im Service Desk als Incidents aufgegeben und von dort ggf. an die dezentralen IT-Gruppen weitergeleitet.

[35] Vgl. [Ti08], Seiten 199ff.

[36] Dabei handelt es sich um die betroffenen IT-Mitarbeiter und eben nicht nur um Vertreter aus den Fachbereichen.

Die dezentralen IT-Mitarbeiter wurden also entsprechend dieses Konzeptes der Einbindung der Fachbereiche gestaltend in den Prozess einbezogen. Sie konnten in einem breit angelegten Schulungsprogramm eine ITIL-Foundation-Prüfung absolvieren, so dass das persönliche Qualifikationsinteresse zusätzlich motivatorisch wirkte. Jeder Fachbereich hat zudem einen ITSM-Spezialisten aufgebaut, der – mit entsprechender höheren Zertifizierung – als interner Treiber des Umsetzungsprozesses auf die dezentralen Teams einwirkte.

Aufgrund der o. a. Maßnahmen erfolgte die bisherige Umsetzung vergleichsweise harmonisch und reibungslos. Zudem konnten die bisherigen Umstrukturierungen im Sinne einer Nachhaltigkeit so durchgeführt werden, dass sämtliche betroffenen Teams diese auch mittragen und aktiv unterstützen. Somit entstand eine sich selbst stabilisierende, virtuelle ITSM-Organisation, da alle Beteiligten ein hohes Eigeninteresse haben, an den Prozessen operativ mitzuwirken.

5 Status Quo und Erfahrungen

Bis Ende April 2012 konnten 2.483 Incidents mittels des eingeführten Incident-Management-Prozesses bearbeitet werden, wobei die Lösungsquote[37] im Service Desk bei 51,4% liegt. Angestrebt wird aber eine Quote von über 80% bis Ende 2012. Als wesentliches Problem der derzeit noch mangelhaften Zielerreichung wurde in Workshops und Ticketanalysen die Wissensübertragung zwischen den überwiegend studentischen Mitarbeitern des Service Desk identifiziert: Es wird weder adäquat dokumentiert, noch wird zielgerichtet in den vorhandenen Dokumentationen, insbesondere im Ticketing-System im Sinne einer Known-Error-Datenbank[38], nach Lösungen für Incidents gesucht. Um die angestrebte Erstlösungsquote von 80% zu erreichen, wurden daher weitere flankierende Maßnahmen eingeleitet:

- Schulungen ins ITIL sowie in die implementierten Prozesse und Systeme an der FH Bielefeld für alle im Service Desk eingesetzten Mitarbeiter.

- Die bereits erläuterte ITIL Foundation Schulung und Zertifizierung aller IT-Mitarbeiter der Fachhochschule, d. h. sowohl der Mitarbeiter in der DV-Zentrale als auch der den Fachbereichen und Dezernaten direkt zugeordneten Spezialisten.

[37] Lösungsquote im Service Desk: Anteil der als Ticket eröffneten Incidents, die direkt, d. h. ohne Eskalation an andere Organisationseinheiten des ITSM, durch die Mitarbeiter des Service Desks im Rahmen der definierten Service Level bearbeitet und geschlossen werden. Die Service Level dienen dabei lediglich als interne Vorgaben und betreffen ausschließlich die durchschnittliche Zeit zwischen Eröffnung und Schließung eines Tickets, gruppiert in verschiedenen Prioritätsklassen. Die Kennzahl Lösungsquote wird monatlich ausgewertet, wobei alle in dem Monat gemäß den internen Service-Level-Vorgaben zu schließende sowie bereits überfällige Tickets ausgewertet werden. Das OTRS lässt in der derzeitigen Ausbaustufe lediglich eine Auswertung mittels Export der Ticketparameter zu.

[38] Known-Error-Datenbank: Ein System, das sämtliche Aufzeichnungen zu bekannten Fehlern enthält. Jeder Eintrag ist dabei mit der zugrundeliegenden Ursache, einer Übergangslösung (Workaround) zur schnellen Wiederherstellung der Arbeitsfähigkeit der Anwender und der Ursachenbeseitigung dokumentiert. In Anlehnung an [IT07], Seite 28.

- Incentivierung der Dokumentation und Lösung durch regelmäßige Veröffentlichung der o. a. Kennzahl „Lösungsquote“.

6 Fazit

Verglichen mit anderen Einführungsprojekten eines organisationsweiten ITSM an Hochschulen[39], verlief das Vorhaben an der FH Bielefeld trotz der sehr verteilten Organisations- und Servicestruktur ohne wesentliche organisatorische Störungen und ohne explizite finanzielle Zusagen. Die ersten Ergebnisse der initialen Projektphasen entsprechen weitestgehend den Erwartungen, mit Ausnahme der noch geringen Lösungsquote im Service Desk. Die Entlastung der lokalen Fachbereichsteams bei Routinetätigkeiten aufgrund der Zentralisierung der Zugriffskanäle stellt trotzdem einen sehr großen Gewinn dar: Das eingeführte ITSM ermöglicht erst die Verlagerung dieser Aufgaben auf studentische Mitarbeiter und wirkt sich daher nachhaltig auf die organisatorische und technische Innovationsfähigkeit der IT-Bereiche der FH Bielefeld aus. Zudem konnten erfolgreich die Grundlagen geschaffen werden, zukünftig eine volumenbasierte Leistungsverrechnung der IT-Services zu etablieren.

Die entstandenen ITSM-Strukturen wurden, vor dem Hintergrund des anstehenden Umzugs, im Sinne einer Nachhaltigkeit so ausgelegt, dass zukünftige strukturelle Änderungen ohne Beeinflussung der Schnittstellen zu IT-Anwendern möglich sind. Zudem werden die ITSM-Prozesse durch sämtliche betroffenen IT-Bereiche selbstregulierend aufrechterhalten und ausgebaut.

Literaturverzeichnis

[Bu08] Buhl, U.: ITIL Praxisbuch – Beispiele und Tipps für eine erfolgreiche Prozessoptimierung. 2. überarbeitete Auflage, mitp Redline, Heidelberg 2008.

[DF11] Deutsche Forschungsgemeinschaft (Hrsg.): Informationsverarbeitung an Hochschulen – Organisation, Dienste und Systeme – Empfehlungen der Kommission für IT-Infrastruktur für 2011-2015. http://www.dfg.de/download/pdf/foerderung/programme/wgi/empfehlungen_kfr_2011_2015.pdf. Aufgerufen am 20. April 2012.

[FG07] IT Governance – Leitfaden für eine praxisgerechte Implementierung. Betriebswirtschaftlicher Verlag Dr. Th. Gabler, Wiesbaden 2007.

[FH12] http://www.fh-bielefeld.de/ueber-uns. Aufgerufen am 20. April 2012.

[HS07] Hofmann, J.; Schmidt, W.: Masterkurs IT Management. Vieweg, Wiesbaden 2007.

[IT07] IT Service Management Forum Deutschland e. V., Arbeitskreis Publikation, ITIL Version 3 Translation Project: ITIL V3 Glossar. Version 31.08.2007. http://www.itsmf.de/fileadmin/dokumente/AK_Publikationen/20070831_ITIL_V3_GLossary_Germany_Translation-Table_EN-DE.pdf. Aufgerufen am 20. April 2012.

[JG11] Johannsen, W.; Goeken, M.; Referenzmodelle für IT-Governance: Methodische Unterstützung der Unternehmens-IT mit COBIT, ITIL & Co. 2. erweiterte und überarbeitete Auflage, dpunkt Verlag, Heidelberg 2011.

[39] Bspw. [Ve10], Seiten 87f.

[OG07a] Office of Government Commerce (Hrsg.): ITIL Service Strategy. Deutschsprachige Ausgabe. TSO The Stationary Office, Norwich 2007.
[OG07b] Office of Government Commerce (Hrsg.): ITIL Service Design. Deutschsprachige Ausgabe. TSO The Stationary Office, Norwich 2007.
[OG07c] Office of Government Commerce (Hrsg.): ITIL Service Transition. Deutschsprachige Ausgabe. TSO The Stationary Office, Norwich 2007.
[OG07d] Office of Government Commerce (Hrsg.): ITIL Service Operation. Deutschsprachige Ausgabe. TSO The Stationary Office, Norwich 2007.
[OG07e] Office of Government Commerce (Hrsg.): ITIL Continual Service Improvement. Deutschsprachige Ausgabe. TSO The Stationary Office, Norwich 2007.
[Ti09] Tiemeyer, E.: Handbuch IT-Management: Konzepte, Methoden, Lösungen und Arbeitshilfen für die Praxis. 3. Auflage, Carl Hanser Verlag, München 2009.
[Ve10] Vellguth, K.: Erfahrungen im Aufbau des IT Service Desks der Technischen Universität München. In: Bode, A.; Borgeest, R. (Hrsg.): Informationsmanagement an Hochschulen. Springer, Heidelberg, Dordrecht, London, New York 2010. Seiten 79-88.
[Vo10] Vogg, H.: Von der Verwaltungs-DV zum IT-Servicezentrum. In: Bode, A.; Borgeest, R. (Hrsg.): Informationsmanagement an Hochschulen. Springer, Heidelberg, Dordrecht, London, New York 2010. Seiten 27-32.

Entwicklung und Betrieb eines Campus-Management-Systems

– Aspekte zur Nachhaltigkeit am Beispiel TISS –

Thomas Grechenig[1)], Thorsten Spitta[3)], Monika Suppersberger[1)], Wolfgang Kleinert[2)], Ronald Steininger[1)], Christof Kier[1)], Martina Pöll[1)]

[1)] TU Wien, Industrielle Softwaretechnik,
{thomas.grechenig, monika.suppersberger, ronald.steininger, christof.kier, martina.poell}
@inso.tuwien.ac.at

[2)] TU Wien, Zentraler Informatikdienst,
wolfgang.kleinert@zid.tuwien.ac.at

[3)] Universität Bielefeld, Angewandte Informatik/Wirtschaftsinformatik
thSpitta@wiwi.uni-bielefeld.de

Abstract: Hochschulen stehen wie alle großen Institution stärker als früher vor der Herausforderung, Prozesse der Lehre, Forschung und Administration mit geeigneten IT-Mitteln effizienter machen zu müssen. Der folgende Beitrag erläutert anhand des Fallbeispiels von TISS, dem Campus-Management-System (CaMS) der TU Wien, Aspekte zu nachhaltiger Einführung und Betrieb eines solchen Systems. Zusätzlich zeigt der Beitrag Kernfaktoren auf, die für die Ablöse von Altsystemen und die Einführung eines modernen und zukunftssicheren CaMS elementar sind.

1 Problemstellung und Ziel

„Der Leidensdruck deutschsprachiger Hochschulen im Bereich Campus-Management (CM) nimmt derzeit permanent zu“ [Bo09, S. 451]. Seit dem Desaster um das Zulassungssystem der deutschen Hochschulen ist auch der Öffentlichkeit deutlich, dass die Einschätzung *„*nimmt zu*“* von 2009 richtig war. Der Anlass für das *„*Leiden*“* ist schnell benannt. Seit dem Übergang auf BA/MA-Studiengänge mit studienbegleitendem, iterativem Prüfen kann keine Hochschule mehr dies mit dem althergebrachten Instrument Excel-Tabellen bewerkstelligen. Die Universitäten brauchen heute für ihre Kernprozesse buchende Systeme, die die Daten der vielen Vorgänge transaktions- und rechtssicher (> 10 Jahre nicht änderbares Archiv) verwalten. Für diesen Komplex spricht man gesamtheitlich von *Campus-Management-Systemen* (CaMS), unter denen es gegenwärtig noch keines gibt, das den Status *Standardsoftware* beanspruchen könnte [BGS10].

Auf der Tagung WI2009 im Februar in Wien [Bo09] gab es Berichte der Universitäten Göttingen, Hamburg, Osnabrück, Bielefeld [Br09] und Darmstadt, die zeigten, dass viele Akteure sich bemühen, wenigstens mit behelfsmäßigen Lösungen zu „überleben". Auf der Tagung *Software Management* im Dezember 2010 in Aachen wurde erstmals im deutschsprachigen Raum ein normatives Konzept vorgestellt, was ein CaMS überhaupt sein könnte und eine kurze Bestandsaufnahme zu Projekten geliefert, die auf ein Standardsystem abzielen [BGS10].

In diesem Beitrag wird das in [Gr10, S. 91ff.] als sehr anschauliches Beispiel, damals "HISS" genannte System, gezeigt und seine Entwicklung beschrieben. Es befindet sich inzwischen als TISS (*TU Wien Informations-Systeme und Services*) vollständig im Echtbetrieb für 32.000 *Core User* (Beschäftigte + aktiv Studierende) und geschätzt mehreren 100.000 weiteren *Usern* (Interessenten, Absolventen und Öffentlichkeit) pro Jahr. Es wird als Beispiel für ein „nachhaltig" betreibbares System vorgestellt. Darunter verstehen wir zunächst eher intuitiv *Langlebigkeit* bei gleichzeitiger *Ressourcenschonung.* Die Anforderungen an in diesem Sinne nachhaltige Softwaresysteme wollen wir unter vier Aspekten betrachten:

- Methodische Migration der Daten aus den Altsystemen und Archiven
- Software-Infrastruktur (Datenbasis, Framework, Komponenten-Konzept)
- Einbeziehung der Stakeholder in Entwicklung und Evolution des Systems
- Betriebskonzept.

Besonders der letzte Punkt muss bei der Entwicklung und Evolution von Software heute umfassend bedacht werden, obwohl er in der „schulmäßigen" Lehre zum Software Engineering nach wie vor kaum vorkommt. Langlebige Softwaresysteme bestehen heute beobachtbar aus einem entwicklungszentrierten Teil und einer IT-betrieblich zentrierten Serviceumgebung, die aus dem Gesamtintegrations- und Betriebsbedarf besteht und letztlich nie frei von Funktions- und Applikationsaspekten ist. Egal wie „sauber" die Architektur des Systems zu Beginn etabliert wird, die Volatilität der informationstechnischen Evolution „erzwingt" diese Drift hin zur Betriebstechnik.

2 Ausgangssituation

Seit 1968 setzt die Technische Universität Wien IT-Systeme zur Unterstützung der administrativen Tätigkeiten ein. Das erste Verwaltungssystem namens TUWIS (TU Wien Informationssystem) war eine COBOL-Applikation, die vorrangig die Administration von Studierendendaten, deren Studien und das Erfassen von Zeugnissen ermöglichte. Die Weiterentwicklung des Systems war über Jahrzehnte von der Implementierung neuer Funktionen „auf Zuruf" und Fehlerkorrekturen geprägt. Trotz einer grundlegenden Umstellung der Datenhaltung von Dateien auf Oracle-Tabellen und den dafür nötigen Softwareanpassungen sah man weder auf Daten- noch auf Applikationsebene den Bedarf für ein umfassendes, integriertes Redesign des Systems. Nachdem TUWIS über Jahrzehnte

hinweg ausschließlich von Fachabteilungen der Universität genutzt wurde, konnte es Ende der 90er Jahre um eine Webapplikation ergänzt werden, so dass auch Studierenden erste Funktionen, beispielsweise die Möglichkeit zur Prüfungsanmeldung, zur Verfügung standen. Mit der Neuentwicklung dieser Webapplikation unter der Bezeichnung TUWIS++ 2003 wurden erste Stimmen laut, die mittlerweile stark gewachsene und mit anderen Systemen der TU Wien verstrickte COBOL-Basis zu ersetzen. Die Realisierung wurde jedoch nie in Angriff genommen; die Systeme wuchsen weiter. Das Ergebnis war ein wartungsintensives Flickwerk, bestehend aus inhomogenen und stark interdependenten Teilsystemen mit komplexen und stark verwachsenen Strukturen. Datenhaltung, Anwendungslogik und Präsentationsebene waren nur sehr mangelhaft getrennt, Dokumentation war kaum vorhanden oder unzuverlässig. Eine Etablierung von verlässlichen Software Engineering Methoden war unmöglich.

Die TU Wien befand sich damit in der Situation, dass auch die Anpassung interner, organisatorischer Prozesse auf Grund der fehlenden Beweglichkeit der IT-Systeme kaum noch möglich war. Um diesen Mangel zu beseitigen, entschied man sich Ende 2007 für eine Neuentwicklung der Systeme in Eigenregie mit Unterstützung eines erfahrenen Entwicklungshauses unter dem Projekttitel TISS (Kurzbezeichnung für „TU Wien Informations-Systeme und Services“). Ziel des Projekts war *„die Etablierung einer langlebigen IT-Strategie, die*

- *eine gemeinschaftliche technische Architektur bereitstellt,*
- *ein modernes interagierendes Applikationsmanagement erlaubt,*
- *die Altsysteme schrittweise ablöst, wo, wenn und wann dies zweckmäßig ist,*
- *die langfristige Wartung sicherstellt,*
- *das Einpflegen neuer Dienste organisch bereitstellt,*
- *benachbarten Systemen einen qualifizierten Docking-Partner anbietet und diesen als kompaktes Leitsystem dient.“* [Kl08]

Das wörtliche Zitat aus der Universitätszeitung zeigt, dass die Einbeziehung der Stakeholder von Anfang an aktiv betrieben wurde.

3 Das Migrationsproblem: Altdaten und -systeme

Bei jeder Systementwicklung setzt man schon lange auf den Daten von Altsystemen auf. Dieses Problem der Migration von Daten wurde eindringlich von dem Schweizer IBM-Berater Max Vetter, habilitiert 1982 an der ETH Zürich, in einer Reihe weit verbreiteter Bücher problematisiert (z.B. [Ve90]). Vetters Veröffentlichungen basieren auf den Erfahrungen mit den Datenbeständen vieler Kunden der IBM, darunter mehrerer Schweizer Großbanken (s. auch, recht aktuell aus der Credit Suisse, [MWF08]).

Alle seine Bücher leitet Vetter, seit der ersten Auflage 1982, mit dem folgenden Satz ein:

„Das Jahrhundertproblem der Informatik besteht in:

- *Der Bewältigung des Datenchaos, das infolge unkontrolliert gewachsener Datenbestände fast überall entstanden ist. [2. ...]"* [Ve90, S. 5]

Vetter mag noch 1990 geglaubt haben, das Problem sei im 20. Jahrhundert lösbar. Heute wissen wir, dass dies noch immer nicht zutrifft.

Bei der Konzeption von TISS wurde das Problem erkannt und bearbeitet. Einer der ersten Schritte zu Beginn des Projekts war die Analyse und (Re-)Dokumentation der Datenstrukturen und -inhalte, um darauf basierend ein konzeptionelles Datenmodell für das neue Zielsystem TISS entwickeln zu können. Insgesamt waren in den Altsystemen TUWIS und TUWIS++ beinahe 350 Tabellen mit über 6500 Spaltendefinitionen vorhanden, die detailliert analysiert werden mussten. Die einzig valide Ressource für eine derartige Analyse war die Produktivdatenbank. Den Analyseprozess maßgeblich erschwert und zugleich umso mehr erforderlich gemacht haben insbesondere die folgenden vier Punkte:

- Veraltete oder fehlende Dokumentation: Die Dokumentation der Datentabellen und -attribute war nur sehr spärlich vorhanden und derart veraltet, dass keine zuverlässigen Schlüsse daraus gezogen werden konnten. Auch die Namen der Tabellen und Attribute waren wenig hilfreich bei der Interpretation möglicher Inhalte, da eine systembedingte Beschränkung auf 8 Zeichen die Namensgebung limitiert hatte (Bsp.: Studenten-Stammdaten wurden in zwei Tabellen mit den Namen "STFIX" und "STVAR" abgelegt).

- Fehlende Normalisierung: Bei der Umstellung von dateibasierter Datenhaltung auf eine relationale Datenbank in den 80er Jahren erfolgte kein Redesign der Datenstrukturen. Das Datenmodell entsprach daher in keiner Weise den gängigen Standards der Datenmodellierung, Normalisierungen wurden nie durchgeführt. Neben der vielfach redundanten Datenhaltung gab es auch keinerlei Fremdschlüssel-Definitionen (referenzielle Integritäten). Relationen zwischen Tabellen wurden nur in der Logik des Altsystems aufgelöst.

- Intransparentes Datenmodell: Zu Beginn des Analyseprozesses war nicht ersichtlich, welche Tabellen noch benutzt wurden und welche nicht. Zusammen mit fehlender Dokumentation und den vorherrschenden Namenskonventionen war dies ein großer Unsicherheitsfaktor.

- Unklare Datenhoheit[1]: Viele in TUWIS und TUWIS++ verwendeten Daten wurden zusätzlich in weiteren Systemen ver- und bearbeitet, häufig in separaten Datenbanken. Meist war unklar, welches System das führende und damit ver-

[1] Mit *Datenhoheit* fassen wir die strenge Koppelung der Operationen von Modulen mit den verwalteten Datentypen im Sinne der Objektorientierung und die klare Zuweisung der Änderungsrechte an Organisationseinheiten (Datenverantwortung) zusammen.

antwortlich für die Datenkonsistenz war. Die organisatorischen Datenverantwortlichkeiten waren diffus (s. [SB08, Kap. 8]).

Um unter diesen Voraussetzungen die nötigen Informationen über die Datenstrukturen und deren Inhalte zu erhalten, wurde ein eigens konzipiertes Verfahren angewandt. Im ersten Schritt wurden alle Bestandsdaten auf der Granularität von Tabellen analysiert und die Tabellen nach zwei Gesichtspunkten kategorisiert. Einerseits erfolgte eine fachliche Zuordnung zu definierten Themenkreisen (Personal- und Studentenstammdaten, Studium, Lehrveranstaltungen, Prüfungen, ...), andererseits wurde eine Unterscheidung nach der Relevanz der Daten getroffen. Tabellen, die nicht mehr in Verwendung waren oder keine geschäftsrelevanten Daten beinhalteten (z. B. temporäre Tabellen für Batch-Prozesse) konnten ausgeschieden und dadurch deren Umfang für eine weitere und tiefer greifende Analyse drastisch reduziert werden. Automatisch generierte Basisdokumentation über vorhandene Strukturdefinitionen und Wertebereiche unterstützten die anschließende Dokumentation der Geschäftsdaten und ermöglichten ein frühes Erkennen potentieller Konflikte mit geplanten Constraint-Definitionen. Zur besseren Abschätzung des Transaktions- und Datenvolumens wurde zusätzlich auch in einem 6 Monate andauernden Analyse-Zeitraum die Interaktion des Altsystems mit den Tabellen auf Basis von Datenbank-Statistiken analysiert. Aus den so gewonnenen Rohdaten konnte die Entwicklung des Datenbestands als auch die Lastverteilung (nach Tagen oder Wochen) aufbereitet werden. Details zur gewählten Vorgehensweise und den Ergebnissen sind in [St09] dargestellt.

Auf Basis der gewonnenen Information mussten also in einem mühsamen Prozess alle Attribute aller alten zu migrierenden Datenobjekte gegen das Soll-Datenmodell abgeglichen werden. Dieser Match verlangt eine exakte Spezifikation für jedes Attribut, und zwar für ein Programm, das zwar viele Male im Test läuft, aber nur genau einmal vor der Inbetriebnahme der neuen Datenbasis.

Entwicklungen, die diesen Aspekt vernachlässigen, können in keinem Fall *nachhaltig* genannt werden, selbst wenn die neue Software noch so elegant oder jahrelang positiv bewährt ist (Beispiel Standardsoftware). Warum? Systeme, die auf korrupten Daten aufsetzen, sind nicht nur unbrauchbar, sie sind ganz einfach falsch, weil sie „Informationen“ aus falschen Daten erzeugen.

Die Problematik wird am besten anhand eines Beispiels illustriert: Die Matrikelnummer eines Studierenden gilt zwar als eindeutig und nicht veränderbar, durch Fehler bei der Immatrikulation kam es aber regelmäßig zu Änderungsbedarf der TUWIS und TUWIS++ Datenbestände. Eine Änderung der Matrikelnummer zu einem Zeitpunkt, zu dem bereits Prüfungsanmeldungen, Zeugnisse und sonstige Daten im System gespeichert waren, erforderte eine manuelle und fehleranfällige Datenkorrektur an vielen verschiedenen Stellen. Inkonsistenzen waren vorprogrammiert. Verschärft wurde die Problematik durch unzureichende Validierungsmechanismen, wodurch beispielsweise die kürzeste erfasste Studiendauer eines Studierenden einen negativen Wert, die längste Studiendauer mehrere hundert Jahre betrug. So anekdotisch verhaltensoriginell dieses Beispiel auch sein mag, es ist typisch für viele in den 80er Jahren entstandene IT-Systematiken, die nie ganz abgeschaltet oder einem vollständigen Reengineering unterzogen wurden.

Für dieses gewichtige Problem der Informationsgenerierung auf Basis korrupter Daten aus eigentlich abgelösten Altsystemen ist bezeichnend, dass außer in [Gr10, Abschn. 8.3] die Datenmigration in heutigen Lehrbüchern der Informatik und der Wirtschaftsinformatik nicht ([So01], [Pf01], [BD04], [AGW05]) erwähnt wird.

Dank des intensiven Analyseprozesses zu Beginn konnte dieses Problem bei der Entwicklung und Einführung von TISS zufrieden stellend gelöst werden. Inkonsistenzen wurden lange vor der Inbetriebnahme des Systems aufgedeckt und geeignete Algorithmen entwickelt, um fehlende oder fehlerhafte Attribute im Rahmen der Migration zu ergänzen bzw. zu korrigieren.

4 Kurzbeschreibung des Systems

4.1 Überblick

TISS wurde als umfassendes Campus-Management-System für die TU Wien konzipiert. Die Autoren bedienen sich hierbei der Definition eines CaMS gemäß [Br09] als ein Softwaresystem, das die relevanten Prozesse einer Hochschule operativ unterstützt und der Führung daraus geeignete Informationen liefert. Die Grundfunktionen von TISS werden in Abschnitt 4.2 zusammenfassend dargestellt.

Abbildung 17 zeigt die Einbettung des CaMS in die Systemlandschaft der TU Wien. Der Umfang der CaMS Funktionalitäten lässt die Dimension des Projekts erkennen. Dem Bedarf nach wirksamer Unterstützung aller Prozesse aus den Bereichen Lehre, Organisation und Forschung ist umfassend Rechnung getragen.

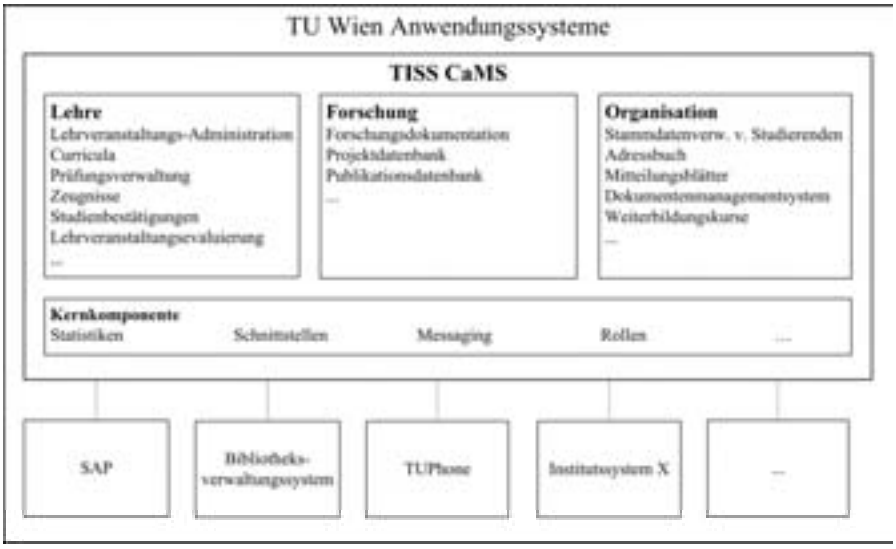


Abbildung 17: TU Wien Anwendungssysteme

Die saubere Definition und Integration von erweiterbaren und standardisierten Schnittstellen zu Drittsystemen, wie zum Beispiel zu SAP, zum Bibliotheksverwaltungssystem oder zur Steuerung der VoIP-Telefonie („TUPhone“), war ein zentrales Ziel von TISS. Anbindungen zu verschiedenen, oftmals auf Standardsoftware basierenden Systemen müssen ebenso leistungsfähig und änderungsstabil gebaut sein, wie das Kernsystem selbst.

4.2 Fachliche Struktur

Die fachlichen Prozesse der TU Wien, die von TISS unterstützt werden müssen, lassen sich in die drei Bereiche *Lehre*, *Forschung* und *Organisation* gliedern.

Der Bereich *Lehre* umfasst vorrangig die Prozesse rund um die Verwaltung und Abwicklung von Lehrveranstaltungen auf Basis von Curricula und bietet dabei Unterstützung in der Prüfungsverwaltung, bildet Zulassungsbedingungen und Anmeldebeschränkungen ab, ermöglicht das Ausstellen und den Druck von Zeugnissen und Studien-Bestätigungen, erlaubt die Evaluation von Lehrveranstaltungen durch Studierende, das Reservieren von Hörsälen und vieles mehr.

Für die *Forschung* sollen den Mitarbeitern und Mitarbeiterinnen geeignete Hilfsmittel die geforderte Forschungsdokumentation erleichtern. Derzeit wird dies bereits durch eine Leistungs- und Projektdatenbank unterstützt. Die Entwicklung einer integrierten Datenbank zur Verwaltung von Publikationen wird 2013 folgen.

Der Bereich *Organisation* bietet neben der Stammdatenverwaltung und Zulassung von Studierenden vorrangig Funktionalitäten für die administrativen Services der TU Wien. Ein Adressbuch stellt Kontaktdaten der Mitarbeiter und Organisationseinheiten bereit, Mitteilungsblätter ermöglichen die Veröffentlichung relevanter Informationen der Universitätsleitung, ein Dokumentenmanagementsystem unterstützt die digitale Verwaltung von Akten und ein Weiterbildungskatalog bietet Mitarbeitern eine Auswahl zahlreicher Möglichkeiten zur Fortbildung.

Eine zusätzliche Komponente, hier als *Kernkomponente* bezeichnet, stellt zentrale und übergreifende Services wie Message-Funktionen und Statistiken bereit, definiert Benutzerrollen für Zugriffsberechtigungen und einiges mehr.

4.3 Softwarearchitektur

Die fachliche Unterteilung in einzelne Themenkreise findet sich in der Modularisierung der Softwarearchitektur wieder: TISS besteht aus vielen Komponenten, die jeweils für sich ein bestimmtes fachliches Thema bedienen. Zwischen ihnen ist die Kommunikation nur über genau definierte Schnittstellen möglich. Der Ansatz hat Vorteile sowohl in der Entwicklungsphase des Gesamtsystems als auch in der Wartung und Weiterentwicklung:

- Die Modularisierung auf technischer Ebene erzwingt, dass fachliche Bereiche strikt getrennt bleiben.

- Die Datenhoheit wird ebenso durch die fachliche Trennung sichergestellt. Die einzelnen Module sind auf diese Weise nur für einen überschaubaren Teil der im Gesamtsystem gespeicherten Daten zuständig.

- Die Module können unabhängig voneinander (weiter-)entwickelt werden, was gerade bei großen Systemen eine schrittweise Einführung erlaubt und so die planmäßige Ablösung von Altsystemen erst ermöglicht.

- Alle Komponenten sind einzeln lauffähig und kompilierbar: Entwickler und Tester können an einem Modul arbeiten, ohne das Gesamtsystem zu beeinträchtigen. Das sorgt gerade in schnellen Entwicklungs- und Testzyklen für eine höhere Produktivität in der Entwicklung.

Allen Komponenten gemeinsam sind der grundsätzliche technische Aufbau und einige architekturelle Vorgaben, die insbesondere die Kommunikation über fachliche Grenzen hinweg festlegen (Datenaustausch und Zusammenarbeit von Benutzern). Als Beispiel seien die Kommunikation zwischen den Komponenten „*curriculum*" (verantwortlich für die Verwaltung und Darstellung von Studienplänen) und „*course*" (Verwaltung von Lehrveranstaltungen) genannt: Der Studienplan besteht aus in einer Baumstruktur gruppierten Einheiten, die jeder Student während seiner Studienzeit absolvieren muss. Für jede dieser Einheiten werden eine oder mehrere Lehrveranstaltungen geboten. Bei der Anzeige des Studienplans bedient sich *curriculum* über die Schnittstelle der in *course* gehaltenen Informationen über die Lehrveranstaltungen. So erhalten die Studierenden eine Gesamtsicht der angebotenen Vorlesungen. Umgekehrt verwendet *course* während des Ankündigungsprozesses für neue Lehrveranstaltungen Informationen aus *curriculum*, um einige Attribute der neuen Lehrveranstaltung mit passenden Daten zu füllen.

Diese Schnittstellen sind zum überwiegenden Teil als rein lesende Schnittstellen definiert. In genau dokumentierten Ausnahmefällen, vor allem bei fachlichen Überschneidungen, wird über diese Schnittstellen auch in „Fremdmodule" geschrieben.

Eine Sonderstellung nimmt die Komponente „*core"* ein: Sie stellt Funktionalitäten zur Verfügung, die von allen Komponenten benötigt werden, zum Beispiel Logging, Monitoring, Zugriff auf das Authentifizierungs- und Rollensystem und Templates für die Benutzeroberfläche. Durch die Verwendung gemeinsamer Templates, einer gemeinsamen Menüstruktur und durchgängig verwendeter Designvorgaben erscheint TISS dem Benutzer homogen.

Die Komponenten sind als *Java EE Applications* implementiert. Die in der folgenden Beschreibung genannten Technologien sind Schnittstellenspezifikationen, die aus der Spezifikation für *Java EE 6* stammen.

Die Architektur folgt einem Mehrschichtenmodell (4-Tier), und zwar die des Systems und jeder einzelnen Komponente (s. Abbildung 2). Die dort nicht gezeigte Schicht 0, die *Präsentationsschicht,* dient der Anzeige und Aufbereitung der Informationen für den User und zur Behandlung von Benutzereingaben. In TISS übernimmt der Webbrowser des Users diese Aufgaben; das dazu nötige HTML und JavaScript wird von *JavaServer Faces* generiert.

Abbildung 2 zeigt zwei Komponenten und ihre Interaktion und damit den Kern des Systems, genannt die *Logikschicht*. Hier ist die gesamte Anwendungslogik implementiert, aufgeteilt in die Teile Frontend (Aufbereitung und Verarbeitung der Daten für die Präsentationsschicht, Interaktion mit den Benutzern) und Backend (Geschäftsprozesse, Anwendungslogik). Die Trennlinie zwischen diesen beiden Teilen bildet eine *Serviceschicht* (API), über die die gesamte Kommunikation zwischen Front- und Backend läuft. Auch alle Mechanismen für die von dieser Komponente angebotenen Schnittstellen (API-EXT) sind hier vereint. Als grundlegendes Programmiermodel wird *CDI* verwendet (*Contexts and Dependency Injection for the Java EE Platform*, ebenfalls Teil der Java EE 6 Spezifikation).

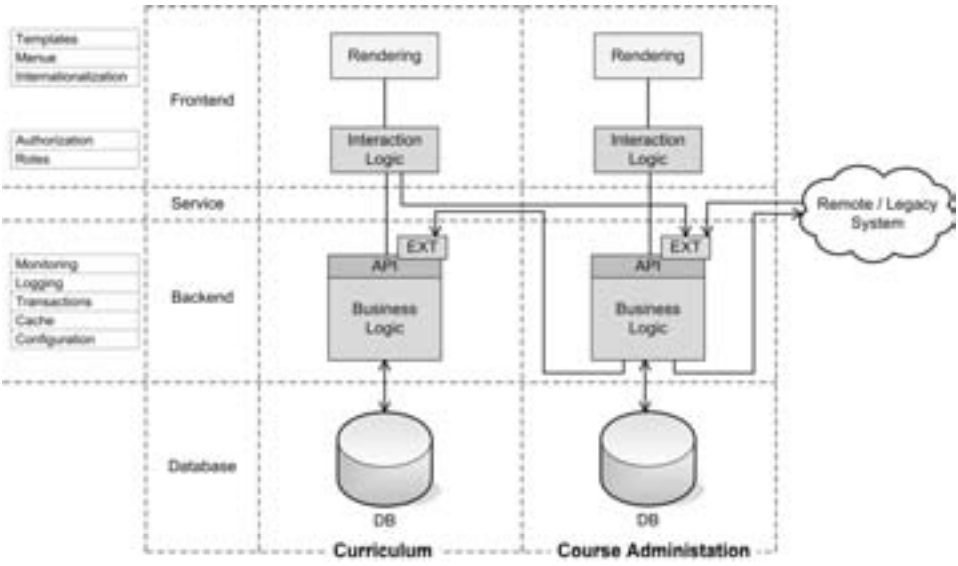


Abbildung 18: TISS Softwarearchitektur

Die *Datenhaltungsschicht* speichert die für den fachlichen Bereich einer Komponente relevanten Daten und übernimmt das Speichern und Laden von Daten für die Logikschicht dieser Komponente. Es gibt keine Zugriffsmöglichkeit der Datenhaltungsschichten zweier Komponenten untereinander; der gesamte Datenaustausch muss über die APIs stattfinden. Das sorgt für eine wirksame Entkopplung der einzelnen Datenbanken und für eine klare Festlegung der Datenhoheit für jede einzelne Tabelle. Auf Anwendungslogik auf DB-Ebene (zum Beispiel *Stored Procedures* oder DB *Trigger*) wird bewusst verzichtet. Die Abstraktion des Datenbankzugriffs erfolgt über *JPA* (*Java Persistence API*).

4.4 Betriebsarchitektur

Hohe Last, Datensicherheit, agile Weiterentwicklungsprozesse und Wartbarkeit bei gleichzeitigen Hochverfügbarkeitsanforderungen sind nur einige der Stresspunkte für den Betrieb von Softwaresystemen, denen durch Einsatz entsprechender Technologien

auch für den Betrieb entgegnet werden muss [Ka11]. Abbildung 19 zeigt das Grundkonzept der fehlertoleranten Infrastruktur, mit der TISS den genannten Punkten begegnet.

Jede Anfrage wird von einem Apache Webserver Cluster entgegen genommen. Statische Inhalte werden zur Entlastung des Backends aus einem Cache geladen, Anfragen auf dynamischen Inhalt werden an das Backend System weitergeleitet. Welcher der vorhandenen Application Server die Anfrage bearbeitet, wird von einem vorgeschalteten Load Balancer Cluster entschieden, der für eine gleichmäßige Verteilung der Last und damit für eine gute Auslastung der Server sorgt. Neben der Lastverteilung übernimmt der Load Balancer auch die zyklische Prüfung der Verfügbarkeit und leitet Anfragen bei Nicht-Erreichbarkeit auf andere Server um. Der Application Server wiederum prüft die Verfügbarkeit der Verbindung zum Datenbankserver. Kann dieser Application Server die Datenbank nicht erreichen, leitet der Load Balancer alle Anfragen auf einen anderen Application Server um.

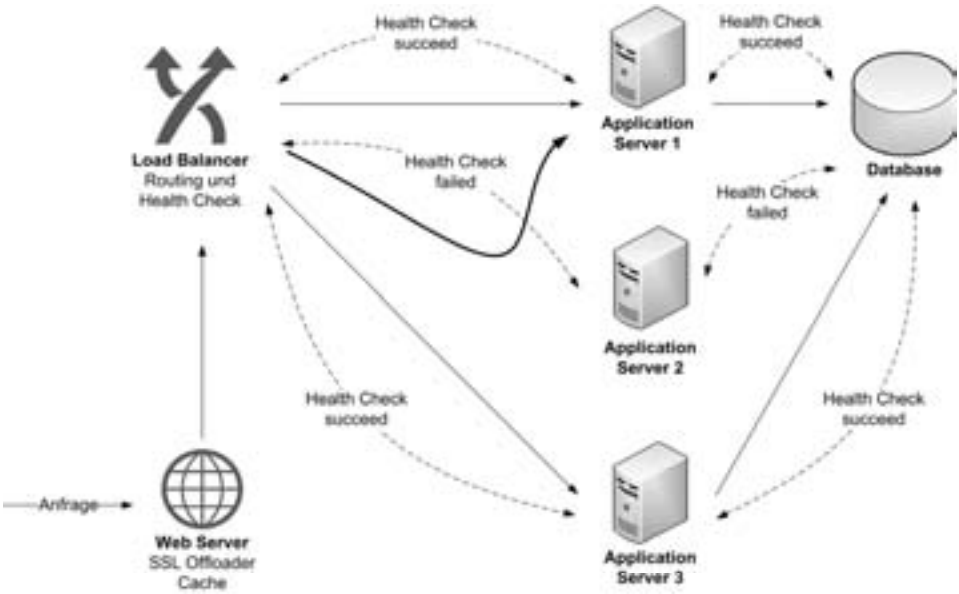


Abbildung 19: Hochverfügbarkeit durch Redundanz und Fehlertoleranz (nach [Ka11])

Im Falle eines Komplettausfalls aller Komponenten, beispielsweise auf Grund eines Feuers, ermöglicht eine Verteilung aller IT-Infrastrukturkomponenten auf zwei Standorte einen desaster-toleranten Betrieb. Im Ernstfall liegt die Wiederanlaufzeit im Bereich von einer Minute. Der mögliche Verlust von Daten bei einer derartigen Katastrophe wird durch den Einsatz einer zentralen Storage-Lösung verhindert, die die Daten zwischen beiden Standorten repliziert. Ein weiteres, wichtiges Merkmal der TISS-Betriebsarchitektur ist die Virtualisierung. So können Dienste de facto unabhängig von der darunter liegenden Hardware betrieben werden. Weitere Details zum Betrieb der TISS-Infrastruktur finden sich in [Ka11].

5 Kriterien für langfristige Lebensfähigkeit

Eingangs hatten wir vier Aspekte genannt, die wir für nachhaltige Software im Sinne von „langlebig bei gleichzeitiger Ressourcenschonung“ (s. Kap. 1) als essenziell betrachten. Dies wollen wir jetzt anhand der Aspekte Datenbasis, Software-Infrastruktur, Anforderungs- bzw. Changemanagement und Betriebskonzept näher beleuchten.

Bevor wir jedoch diese Punkte im Einzelnen behandeln, muss ein wesentlicher Bestandteil des Artefakts *Software* angesprochen werden, der meist fälschlicher Weise als „Dokumentation“ bezeichnet wird. Hierzu müssen wir kurz die Entstehungsgeschichte agiler Vorgehensmodelle betrachten. Es ändern sich ja *alle* Artefakte, nicht nur der Quellcode.

5.1 Die sogenannte Dokumentation

Ein CaMS gehört zur Klasse der *Embedded Systems*, die Lehmann *E-Programs* genannt hat [LP76, Le80]. Die Software ist nicht in ein technisches, sondern in ein soziotechnisches System eingebettet, gemeinhin *Organisation* genannt. Solche Softwaresysteme müssen sich evolutionär verändern können, weil ihre Umwelt dynamisch ist. Diesen Zusammenhang hat Christiane Floyd schon früh in dem unseres Wissens ersten evolutionären Entwicklungsmodell aufgezeigt [Fl81]. Dieses Modell hat viele Nachfolger in Form agiler Vorgehensmodelle gefunden. Eines davon war eine Arbeit, die sich früh mit der Frage der Bildung geeigneter Teilsysteme befasste und auch die sogenannte Dokumentation hinterfragte [Sp89, Kap. 3]. Das Buch wurde verfasst, als der Hype *Prototyping*[2] die Zeit der agilen Modelle einläutete. Das Thema machte es schon damals notwendig, die Rolle der sogenannten Dokumentation zu untersuchen.

Bei der Erstentwicklung ist jede denkbare Repräsentation (z. B. Use Cases, Prosa, Grafik, Testfälle) ein Teil der fachlichen und technischen Konstruktion des Systems und *keine* „Dokumentation“. Spätestens bei der Inbetriebnahme der Software müssen alle Bestandteile, besonders auch die der fachlichen Konstruktion daraufhin überprüft werden, ob sie mit der weiteren Evolution des Systems gepflegt werden können oder nicht. Auf dem Weg zum Quellcode gibt es viele redundante Bestandteile der fachlichen Konstruktion (s. [Sp89, S.78, 81ff]). Nur die essenziellen dürfen erhalten bleiben und müssen auf Dauer gepflegt werden. Geschieht das nicht, sind die Daten der sogenannten Dokumentation keine Information für die Entwickler, sondern *Desinformation*, die schädlicher sein kann als ein reiner, sprachlich und strukturell „sauberer“ Quellcode[3]. Hierauf wurde bei TISS geachtet: *„Alibi-Dokumentation ist noch schlechter als keine“* [Su10]. Teile der Dokumentation wurden im Projektumfeld von TISS oft erst dann erstellt, wenn relevante Freeze-Zustände des Systems erreicht waren. Seriöse Dokumentation muss als wesentliche Voraussetzung für Nachhaltigkeit gesehen werden, allerdings nur, wenn sie die Gegenwart korrekt wiedergibt. Will man ältere Dokumente aufbewahren, nennt man sie *Archiv*.

[2] Als erste würden wir CASE ansehen, *Computer Aided Software Engineering* (s. auch [Gr10, S. 38]).
[3] Damit sind insbes. fachlich verständliche Variablen-, Prozedur- bzw. Methodennamen gemeint.

5.2 Die Datenbasis

Über die Schwierigkeiten, die eine schlecht entworfene Datenbasis mit sich bringt, wurde schon in Kap. 3 gesprochen. Daten sind diejenige Ressource des Produktionsfaktors Information – im Gegensatz zu Hardware und Software – die eine Organisation nicht kaufen kann. Die Leitlinien von TISS für eine nachhaltige Datenbasis waren:

(1) Jede Komponente hält ihre Daten in einer eigenen Datenbank. Dadurch entstehen fachlich zusammenhängende, aus wenigen Tabellen bestehende Datenbanken für die Einzelkomponenten.

(2) Datenbank-Schemata werden aus dem (gut dokumentierten) Programmcode generiert, und damit aus originären Quellen. Diese sind das Datenmodell, aber auch Quellcode-Templates. Auch Änderungen an Datenbank-Schemata werden so behandelt.

(3) Integritätsprüfungen werden in der Anwendungsschicht durchgeführt. So kann schon oberhalb der Datenbankebene ein hohes Maß an Datenintegrität erreicht werden. Dies macht von den sehr verschiedenen Implementierungen des Relationenmodells[4] unabhängig und lässt sich benutzerfreundlich implementieren.

(4) Die Basis für Variablennamen ist das logische Datenmodell, gerade auch in Quellprogrammen. Hierdurch erreicht man eine Homogenität der „Dokumente“ *Datenmodell*, *Datenbankschema* und *Quellcode* (s. hierzu auch: [Sp96]).

5.3 Die Software-Infrastruktur

Für die diversen Entscheidungen zur Software-Infrastruktur galten folgende Leitlinien:

(1) Implementierung nur gegen Spezifikationen und Standards.

(2) Hersteller-Unabhängigkeit.

(3) Offene Standards und Open-Source Basissoftware.

(4) Stabile Plattformen (Frameworks).

(5) Technische Prototypen bei neuen Technologien.

(6) Prinzip der „Austauschbarkeit“ im architekturellen Systemdesign.

Punkt (1) ist eigentlich eine softwaretechnische Binsenweisheit und bedarf hier keiner weiteren Erklärung.

[4] Als „relevante“ Systeme haben am Markt nur überlebt: DB2, Oracle, SQL-Server. Als „relevant“ bezeichnen wir die einzigen nach CC zertifizierten Systeme. Nur diese können professionellen Ansprüchen genügen. Vgl. https://www.bsi.bund.de/DE/Home/home_node.html

Dies ist anders bei Punkt (2), der leicht in einen dogmatischen Diskurs abdriftet. Es kann für kleine Organisationen sinnvoll sein, auf die Informationstechnik genau eines Herstellers zu setzen. Doch dies ist nicht unser Kontext. Ein CaMS für große Hochschulen ist sogar aus industrieller Sicht auf Grund der ungewöhnlich hohen Benutzerkomplexität ein äußerst anspruchsvolles System (siehe hierzu Abb. 2.5 in [Gr10]). Für große Systeme, die lange betrieben werden müssen, verbietet sich eine Herstellerbindung, insbesondere bei den Basis-Infrastrukturen Betriebssystem, Webserver und Datenbank. Wie an den Beispielen SAP und START in [BG10, S.72f] gezeigt wurde, darf noch nicht einmal eine Programmiersprache als langlebig genug angesehen werden. Da man aber eine Wahl treffen muss, ist eine standardisierte Sprache unabdingbar. Als Beispiel für die Hersteller-Unabhängigkeit sei die Datenbank gewählt. TISS läuft auf den Produktivsystemen mit einer Oracle DB, auf verschiedenen Testsystemen werden aber (ohne Änderungen am Code) MySQL beziehungsweise PostgreSQL verwendet. Die eingesetzte Datenbank ist also mehr eine Betriebs- und Lizenzfrage als ein die Infrastruktur prägendes Produkt. Dies schließt den punktuellen Einsatz herstellerspezifischer Module nicht aus, wenn sie offene Schnittstellen haben. So wird etwa ein Oracle-Modul im Produktivsystem benutzt, das Suchfunktionen für die Benutzer besonders effizient ausführt.

Punkt (3), offene Standards, hängt natürlich stark mit Punkt (2) zusammen. TISS setzt durchgängig auf Open-Source-Implementierungen von offenen Standards. Durch den Zugriff auf den Sourcecode dieser Projekte kann bei Bugs die Fehlerursache schnell gefunden und der entwickelnden Community gemeldet werden. Oft ist es möglich, den Fehler selbst zu beheben. Bei wichtigen Bugfixes ist es sogar denkbar, eine selbst gefixte Variante des fehlerhaften Codes zu verwenden, bis der Hersteller eine fehlerbereinigte Version bereitstellt. Nicht zuletzt eröffnet eine aktive Beteiligung von Teammitgliedern in der Open-Source-Community die Möglichkeit, die Entwicklung der Software mitzugestalten und so für das eigene Projekt wichtige Features schnell zu integrieren und Mitspracherechte in der Weiterentwicklung zu erhalten.

Die Verwendung von Frameworks, unser vierter Punkt, ist schon lange Stand der Technik (s. [Sp89, Kap. 10], [Gr10, Kap. 6]). Ohne ein Framework (oder mehrere, komponenten- und ebenenspezifische) lässt sich kein System über viele Jahre stabil halten. Ohne Framework droht schnell *Architectural Decay* (Erosion der Struktur in der Evolution). Es versteht sich, dass für TISS gerade im Bereich der Architektur auch technische Prototypen (s. [Sp89, Kap.7]) evaluiert und die am besten geeigneten ausgewählt wurden. Das frühe Herstellen von echten betrieblichen „Durchstichen“ in das Zielsystem verbessert die prinzipielle Betriebskonformität nachhaltig. Dies ist unser Punkt (5).

Punkt (6): Entwicklungs- und Betriebsarchitektur wurden bei TISS nachhaltig so gestaltet, dass die Kernkomponenten alle prinzipiell austauschbar sind, ohne dass das Gesamtsystem auszutauschen ist. Eine mittelfristige Bindung an einen Hersteller, an ein Produkt, an eine Sprache bedeutet somit nicht die zwangsweise Herstellung eines langfristigen Software-Monolithen. Ein solcher „kaum entwirrbarer“ Monolith war mit den Altsystemen an der TU Wien gegeben und sollte durch dieses Architekturprinzip verhindert werden.

5.4 Die Einbindung der Organisation – Anforderungsmanagement

Gerade bei einem System mit hoher Benutzerkomplexität und Nutzerklassen mit sehr unterschiedlichen Anforderungen (etwa Dozenten, interne Verwaltung und Studierende) ist die Einbindung der Akteure der Organisation essenziell. „Eindeutige Geschäftsprozesse" lassen sich leicht einfordern [FH09], sind auch notwendig, aber schwer durchsetzbar. Historisch etablierte Abläufe müssen hinterfragt und gegebenenfalls neu geordnet und durch elektronische Workflows unterstützt oder abgelöst werden. Dieser einschneidende Veränderungsprozess muss von allen Akteuren mitgetragen werden; nur mit deren Mitwirkung ist die Neugestaltung möglich. *Befehlen* ist nicht nur unmöglich, sondern kontraproduktiv[5].

Der folgende Unterschied eines CaMS zu einem industriellen Anwendungssystem wiegt schwer: Beim Unternehmens- oder Behördensystem gehört der Kunde zur Umwelt, beim CaMS ist er Systemelement. Ein Student ist in einer Hochschule *kein* klassischer „Kunde". Er ist jedoch ein Nutzer, der sehr ernst genommen werden muss und z. B. in Österreich ein Nutzer, der oft deutlich höhere Anforderungen einbringen kann, als es klassische Angestellte oder typische Kunden tun würden. Als Beispiel seien die Datenschutz- und Privacy-Erfordernisse an den Universitäten und z. B. die e-Voting-Debatte in Österreich 2010 genannt.

Bei Beschäftigten, die Veränderungen weniger gewohnt sind als Mitarbeiter in der Wirtschaft, erlebt man schnell den Effekt, dass die Software sich zunehmend zum Sündenbock für negativ bewertete Veränderungen entwickelt, die andere organisatorische Ursachen haben [Ja09]. Zudem wird vielen Anwendern erst in der praktischen Anwendung bewusst, welche Implikationen ihre fachlichen Vorstellungen auf die technischen Prozesse, Abläufe und die Administration haben [De09]. An der TU Wien ging der Einführung von TISS eine sehr intensive Phase der Anforderungsanalyse voraus. Bewährte Kernfunktionalitäten des Altsystems sollten auf einer dem State of the Art entsprechenden technologischen Basis nachgebildet und erweitert werden. Hierzu erfolgte zum Teil eine (Re-)Dokumentation der alten Verfahren mit zahlreichen Workshops und Interviews. *Akzeptanz* verlangt, dass man erst einmal eine gemeinsame Sprache findet.

Bei Hochschulen mit einer traditionell großen Autonomie der Substrukturen darf man nicht unterschätzen, dass die Entscheidungs- und damit die Entwicklungsprozesse zur Umsetzung spezifischer Funktionalitäten iterativ und oftmals auch langwierig sind. Während der Entwicklung eines Systems müssen mehrere 10.000 Einzelentscheidungen getroffen werden, über die eine Führung nicht bis ins letzte Detail informiert sein kann. Erfahrene Analysten und eine starke Projektleitung, die von der Hochschulleitung unterstützt wird, sind für diese zeitkritischen Entscheidungsprozesse unerlässlich. Diese Prozesse waren für das Projekt TISS und sind auch jetzt für einen reibungslosen Betrieb ein wesentlicher Erfolgsfaktor.

TISS hatte trotz guter Vorplanung starke, für die User teilweise anstrengende „Einführungsschmerzen" zu vermerken, weniger auf Seite der studentischen Nutzergemeinschaft

[5] Zitat einer Psychologin, das die soziale Dimension des Problems auf den Punkt bringt: *„Entscheidungen, die nicht getragen werden, werden unterlaufen."*

als auf Seite der Lehrenden und Verwaltungsmitarbeiterinnen. Letzteres war nicht auf mangelnde Methodik sondern auf die strategische Entscheidung des Rektorats zu einer um ein Jahr vorgezogenen Abschaltung des Altsystems mit der damit verbundenen Einführung eines Roh-Produktes TISS zurückzuführen, das erst im Zuge der ersten 4 Monate des Betriebes fertiggestellt wurde. Viel Stress für die Kern-User aber auch eine beträchtliche Ersparnis für die Universität durch den Wegfall von Wartungsarbeiten für ein Jahr am Altsystem sowie eine deutliche Verdichtung der Entwicklungsarbeiten am Neusystem.

5.5 Der Betrieb des Systems

Mit Verbreiterung der Funktionalitäten und steigender Anzahl der Nutzer muss nicht nur die softwaretechnologische Basis sondern auch der Betrieb eine entsprechende Skalierbarkeit erlauben. Besonders hohe Erwartungen an die Verfügbarkeit eines Systems werden naturgemäß dann gestellt, wenn die Anzahl der gleichzeitigen Nutzer am höchsten ist und diese Nutzer für sie kritische Aktionen durchführen müssen. Ein angemessenes Betriebskonzept muss auf die erwarteten Lastspitzen ausgelegt werden, um einen reibungslosen Betrieb mit entsprechender Verfügbarkeit des Systems zu ermöglichen. Bei einem CaMS werden diese Lastspitzen zu Beginn (und ggf. am Ende) eines Semesters erreicht. Abbildung 4 zeigt die Anzahl der Seitenaufrufe pro Stunde im Bereich Lehre zu Beginn des Sommersemesters 2011 – bei Lastspitzen wurden bis zu 12.000 Seitenaufrufen pro Minute verarbeitet.

Der Kampf um beschränkte Plätze bei Lehrveranstaltungen beginnt, Anmeldungen zu mehreren tausend Lehrveranstaltungen finden innerhalb kurzer Zeit statt, oft zeitgleich. Ein wesentlicher Vorteil der in Abschnitt 4.4. dargestellten Betriebsarchitektur ist ihre gute horizontale Skalierbarkeit. Erhöhen sich die Anforderungen an die Performance zu Semesterbeginn, können bei Bedarf einfach und ohne Betriebsunterbrechung zusätzliche Application Server kurzfristig integriert und so die steigende Anzahl von Anfragen auf mehrere Server verteilt werden.

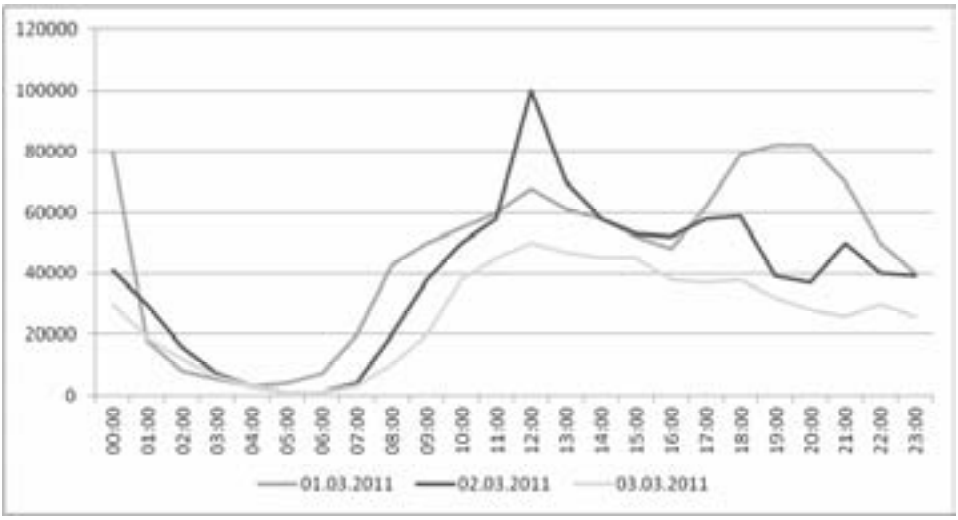


Abbildung 20: Seitenaufrufe pro Stunde im Bereich Lehre zu Semesterbeginn (aus [Ka11])

Vor der Inbetriebnahme von TISS war der Semesterbeginn an der TU Wien von stunden- und teilweise tagelangen Systemausfällen auf Grund von Überlastungen geprägt. Weder die Software- noch die Hardwarearchitektur waren für derartige Lastspitzen ausgelegt, alle damals verfügbaren Mittel zur Erhöhung der Verfügbarkeit waren ausgeschöpft. Mit der Inbetriebnahme von TISS erlebten die Studierenden und Mitarbeiter der TU Wien das erste Mal eine unterbrechungsfreie Serviceverfügbarkeit zu Semesterbeginn.

5.6 Ergebnis zur Nachhaltigkeit

Wir hatten in diesem Kapitel die Eigenschaften und Entwicklungshistorie von TISS untersucht, die Nachhaltigkeit im Sinne unserer Arbeitsdefinition unterstützen. Dabei hatten wir festgestellt, dass ein in eine Organisation eingebettetes Softwaresystem *nachhaltig* genannt werden kann, wenn es:

- die Organisationsziele dauerhaft unterstützt und nicht behindert,
- über lange Zeit (> 20 Jahre) evolutions- und informationsfähig[6] bleibt,
- möglichst unabhängig von den Entscheidungen einzelner liefernder Akteure ist,
- technologisch allgemein akzeptiertem Fortschritt folgen kann.

[6] Längsschnitt-Daten müssen auch nach 10 Jahren noch korrekt sein und nicht – wie den Autoren bekannte, gängige Systeme in Hochschulen – plötzlich Nullwerte in Tabellenspalten enthalten. Dann liefern Datenbankanfragen bekanntlich falsche Ergebnisse. Dieses Problem löst auch kein Werkzeug, etwa ein Data Warehouse. Ein Beispiel aus der TISS-Entwicklung wurde in Kap. 2 berichtet.

Zwischen Hardware und Software bestehen hinsichtlich Nachhaltigkeit zwei fundamentale Unterschiede:

(1) Hardware ist sehr viel kurzlebiger. Durch den raschen technologischen Fortschritt sind die sinnvollen Nutzungszeiten erheblich kürzer als bei Software.

(2) Als materielles Gut spielt bei Hardware die Frage der Wiederverwendung materieller Ressourcen eine wichtige Rolle, bei Software fast keine.

Wir fassen zusammen:

> Unter **Nachhaltigkeit** eines technischen Systems verstehen wir seine Nützlichkeit für die Akteure über lange Zeiträume zu günstigen Kosten.

6 Ausblick und Weiterentwicklung

Die gewonnen Erfahrungen, die die TU Wien bei der Einführung und der seither laufend stattfindenden Weiterentwicklung von TISS machen durfte, bestätigen, dass die unter Kapitel 5 genannten Faktoren für den Erfolg eines CaMS essentiell sind. Schon in [Kl09] wird darauf hingewiesen, dass *„organisatorische, personelle und technologische Faktoren für ein erfolgreiches Informationsmanagement gleichermaßen berücksichtigt werden müssen“* und *„Die Akzeptanz der Informationssysteme steigt, wenn sie in die Organisation eingebettet werden“*. Die TU Wien wird den mit der Einführung von TISS gewählten und seither ausgezeichnet bewährten Kurs stetig weiter verfolgen. Das System wird weiter wachsen – ggf. auch schrumpfen –, kurzum, es wird laufend an die Bedürfnisse der Hochschule und der Nutzer angepasst werden. Dank der stabilen und flexiblen technologischen Basis, der offenen, standardisierten Schnittstellen und der etablierten Projektstrukturen ist TISS als Kernsystem der TU Wien auf zukünftige Anforderungen vorbereitet. Verstärkte Unterstützung mobiler Services, z. B. die Integration von *Near Field Communication* (NFC)-Technologien oder die Anbindung weiterer (externer) Systeme werden dabei ebenso eine Rolle spielen wie die laufende Verbesserung durch Einarbeitung des Benutzerfeedbacks.

Literaturverzeichnis

[AGW05] Alpar, P.; Grob, H.L.; Weimann, P.; Winter, R.: Anwendungsorientierte Wirtschaftsinformatik. 4. überarb. und erw. Aufl., Vieweg, Braunschweig – Wiesbaden, 2005.

[Bo09] Bode, A. et.al.: Integrierte Campus-Managementsysteme. In: [HKF09], S. 451-552.

[BD04] Brügge, B.; Dutoit, A.H.: Objektorientierte Softwaretechnik. Pearson Studium, München, 2004.

[BGS10] Bick, M.; Grechenig, T.; Spitta, T.: Campus-Management-Systeme – Vom Projekt zum Produkt. In: *Pietsch, W.; Krams, B.* (Hrsg.): Vom Projekt zum Produkt. Fachtagung Aachen, Dez. 2010, Lecture Notes in Informatics 178, Koellen, Bonn, S. 61-78.

[Br09] Brune, H. et.al.: Ein Campus-Management-System als evolutionäre Entwicklung. In: [HKF09], S. 483-492.

[De09] Degenhardt, L. et.al.: Campus-Management-Systeme erfolgreich einführen. In: [HKF09], S. 463-472.
[FH09] Fischer, H.; Hartau, C.: STiNE an der Universität Hamburg zur Einführung eines integrierten Campus Management Systems. In: [HKF09], S. 533-542.
[Fl81] Floyd, C.: A Process-oriented Approach to Software Development, in: Systems Architecture, Proc. of the 6th European ACM Regional Conference. Westbury House 1981, pp.285-294.
[Gr10] Grechenig, T. et.al.: Softwaretechnik – Mit Fallbeispielen aus realen Entwicklungsprojekten. Pearson Studium, München, 2010.
[HKF09] Hansen, H.R.; Karagiannis, D.; Fill, H-G. (Hrsg.): Business Services – Konzepte, Technologien, Anwendungen (Bd 2). 9. Int. Tagung Wirtschaftsinformatik, Febr. 2009 Wien.
[Ja09] Janneck, M. et.al.: Von Eisbergen und Supertankern: Topologie eines Campus-Management-Einführungsprozesses. In: [HKF09], S. 453-462.
[Kl08] W. Kleinert, T. et.al.: The Making of TISS. ZIDline, Nr. 18, Juli 2008, S. 3-8.
[Kl09] Klug, H.: Erfolgsfaktoren bei der Umstellung von Informationssystemen an Hochschulen. In: [HKF09], S. 473-482.
[Ka11] Kamenik, R. et.al.: Betrieb der TISS-Infrastruktur. ZIDline, Nr. 23, April 2011, S. 14-18.
[Le80] Lehmann, M.M.: Programs, Life Cycles and Laws of Software Evolution, in: IEEE Proceedings 68(1980) 9, pp.1060-1076.
[LP76] Lehman, M.M., Parr, F.N.: Program Evolution and its Impact on Software Engineering, in: 2nd ICSE, San Francisco 1976, 350-357.
[MWF08] Murer, S.; Worms, C.; Furrer, F.J.: Managed Evolution – Nachhaltige Weiterentwicklung großer Systeme. Informatik Spektrum 31(2008) 6, S. 537-547.
[Pf01] Pfleeger, S.L.: Software Engineering. 2nd ed. Prentice Hall, Upper Saddle River 2001.
[SB08] Spitta, T.; Bick, M.: Informationswirtschaft. 2. überarb. und erw. Aufl., Springer, Berlin – Heidelberg, 2008.
[So01] Sommerville, I.: Software Engineering. 6th ed., Addison-Wesley, Boston et al. 2001.
[Sp89] Spitta, T.: Software Engineering und Prototyping. Springer, Berlin et al. 1989.
[Sp96] Spitta, T.: „CASE“ findet im Kopf statt. INFORMATIK/INFORMATIQUE 3(1996) 3, 17-25.
[St09] Strobl, S. et.al.: Digging Deep: Software Reengineering supported by Database Reverse Engineering of a System with 30+ Years of Legacy. Wien, Vienna University of Technology (TU Wien), 2009.
[Su10] Suppersberger, M. et.al.: TISS Epistemologie II. ZIDline, Nr. 22, Juni 2010, S. 16-21.
[Ve90] Vetter, M.: Aufbau betrieblicher Informationssysteme mittels konzeptioneller Datenmodellierung, 6. Aufl., Teubner, Stuttgart, 1990.

Data Warehousing an Hochschulen
– Ein Statusbericht –

Sebastian Frodl[1], Peter Hartel[2]

Fachhochschule Bielefeld – University of Applied Sciences
Kurt-Schumacher-Str. 6
33615 Bielefeld
sebastian.frodl@fh-bielefeld.de
peter.hartel@fh-bielefeld.de

Abstract: Die Veränderungen der Hochschullandschaft in den letzten Jahren führten zu neuem Informations- und Steuerungsbedarf an Hochschulen. An der Fachhochschule Bielefeld wurde das etablierte, dezentral organisierte Berichtswesen als ungeeignet erkannt, um diesem steigenden Steuerungsbedarf effizient zu begegnen. Dieser Statusbericht zeigt, wie an der Fachhochschule Bielefeld ein Data Warehouse System als nachhaltige Lösung zur Erfüllung von Anforderungen an eine moderne Hochschulsteuerung eingesetzt wird.

1 Einleitung

Das Konzept der Nachhaltigkeit ist heute vielfach definiert und findet in Bereichen wie Umwelt- und Naturschutz, Politik oder Finanzwirtschaft Beachtung. Der Begriff der Nachhaltigkeit als „längere Zeit anhaltende Wirkung“ definiert darüber hinaus auch für das Softwaremanagement erstrebenswerte Ziele: „Langlebige, Ressourcen schonende Nutzbarkeit“ von Informationssystemen. Der Begriff des nachhaltigen Softwaremanagements rückt somit wesentliche Ziele der Entwicklung und des Betriebs von Softwaresystemen in den Vordergrund, deren Umsetzung zweifelsfrei langfristige Vorteile mit sich bringt.

Bezogen auf analytische Informationssysteme, stellt sich die Frage der Nachhaltigkeit insbesondere bezüglich des Aspekts der langfristigen Verfügbarkeit von Daten zur Gewinnung prozessrelevanter Informationen. Dabei sind Kernaspekte eine organisationsweite Datenbasis, die eine ganzheitliche Sicht zur Unterstützung von Steuerungs- und Entscheidungsprozessen ermöglicht sowie die zeitnahe Verfügbarkeit von Informationen für Adressaten auf möglichst direktem Weg.

Betrachtet man die Entwicklung eines solchen Informationssystems unter Aspekten der Langlebigkeit, gilt es in der Konzeptionsphase insbesondere die Fragen zu beantworten:

[1] Als wissenschaftlicher Mitarbeiter zuständig für Data Warehousing und Business Intelligence
[2] Professor für Wirtschaftsinformatik am Fachbereich Wirtschaft und Gesundheit

- Wie können Prozesse der Informationslogistik langfristig vereinheitlicht und verstetigt werden?

- Wie können Informationsprozesse von (veränderlichen) operativen Systemen und Prozessen entkoppelt werden?

- Wie wird dabei dauerhaft Ressourcen schonend die Verfügbarkeit von Informationen optimiert?

Wie in der Wirtschaft längst etablierte Ansätze des Data Warehousing und der Business Intelligence im Hochschulwesen helfen, den gestellten Fragen bei der Lösung des Problems von steigendem Steuerungsbedarf an Hochschulen gerecht zu werden, soll ein Erfahrungsbericht der Fachhochschule Bielefeld aufzeigen.

Im Folgenden werden zunächst der Steuerungsbedarf an Hochschulen und Probleme des dezentral organisierten Berichtswesens dargestellt. Nach einer Erläuterung der Architektur von Data Warehouse Systemen, wird anhand des Teilprojekts „Einführung eines Data Warehouse Systems“ im Rahmen der Qualitätsoffensive der Fachhochschule Bielefeld, das Vorgehen zur Etablierung eines nachhaltigen Softwaresystems zur Hochschulsteuerung vorgestellt.

2 Steuerungsbedarf an Hochschulen

2.1 Bologna Prozess und Hochschulfreiheit

Die Hochschullandschaft in Deutschland befindet sich seit einigen Jahren in einem Prozess ständiger Veränderung und Weiterentwicklung [HRK04]. Moderne Hochschulen müssen sich in ihrem Selbstverständnis als Dienstleister in Sachen Bildung und Forschung begreifen. Dabei sind sie einer Vielzahl von Herausforderungen ausgesetzt. U.a. sind dies:

- Der Bologna-Prozess [BMB12], der zu einer Vereinheitlichung von Studienabschlüssen geführt hat und damit für mehr Transparenz auf dem Bildungsmarkt sorgt. Damit hat sich der Wettbewerb um die besten Studienanfänger noch weiter verschärft. Hochschulen müssen mehr denn je ihr Profil schärfen, um ihre Attraktivität zu erhalten und zu stärken.
- Neue Bachelor- und Masterstudiengänge, die am Reissbrett konzipiert wurden. Die Studierbarkeit dieser Studiengänge wird häufig von Studierenden in Frage gestellt. Unter dem Schlagwort „Bildungsstreik” hat sich diese Kritik gebündelt und entladen. Hochschulen sind aufgefordert, ihre neuen Studiengänge zu überarbeiten und dabei auf die Kritik der Studierenden zu reagieren.
- Die zunehmende Autonomie [HFG06], die den Hochschulen die Verantwortung für das Budget, die Entwicklung neuer Studiengänge, die Personalplanung etc. überträgt. Dies führt automatisch zu einem zunehmenden internen Wettbewerb um die Ressourcen. Bei der Vergabe von Mitteln an die einzelnen Einheiten

(Fachbereich, Forschungseinheiten etc.) spielt das Leistungsprinzip eine zentrale Rolle.

- Die doppelten Abiturjahrgänge, die durch die Verkürzung der Schulzeiten zum Erwerb der Hochschulreife zustande kommen. Die zu erwartende Studentenwelle muss bei der Planung von Studienplatzangeboten und Personalkapazitäten Berücksichtigung finden.

Diese (unvollständige) Liste zeigt, dass Hochschulen eine Vielzahl von Entscheidungen selbst treffen müssen, die ihre zukünftige Entwicklung maßgeblich bestimmen. Die verschiedenen Themenfelder sind eng miteinander verknüpft, so dass einzelne Entscheidungen häufig Auswirkungen auf ganz unterschiedliche Bereiche haben. Oberstes Prinzip bei allen Entscheidungen ist dabei der effiziente und langlebige Einsatz der zur Verfügung stehenden Ressourcen und daraus resultierend die Notwendigkeit, bestehende Geschäftprozesse so optimal wie nur möglich zu gestalten. In den letzten Jahren ist der Bedarf an Definition und Messung von Leistung und Qualität auch an Hochschulen deutlich gestiegen.

Die zugleich steigende Komplexität und Dynamik, sowohl im Bereich der Lehre und Forschung als auch in der Finanzierung von Hochschulen, lassen die Notwendigkeit des Einsatzes von betriebswirtschaftlichen Systemen und Instrumenten zur effektiven und effizienten Hochschulsteuerung entstehen.

2.2 Dezentrales Berichtswesen der FH Bielefeld

Organisationsweite Informationen und steuerungsrelevante Kennzahlen haben auch an der Fachhochschule Bielefeld in den letzten Jahren zunehmend an Bedeutung gewonnen. Das etablierte Berichtswesen, in welchem verschiedene Organisationseinheiten regelmäßig manuell Standardberichte verfassen und auf Ad-hoc-Anfragen weitere (uneinheitliche) Berichte zu Analysezwecken liefern, zeichnet sich dabei immer häufiger als lückenhaft und ineffizient ab. Die Umsetzung von Berichten basiert zudem in vielen Fällen auf Individualwissen statt auf organisationalem Wissen, womit sich Prozesse nur eingeschränkt nachhaltig definieren lassen.

In technischer Hinsicht herrscht durch die Heterogenität operativer Systeme zudem eine Vielfalt an verwendeten Berichts-Werkzeugen, jeweils optimiert oder gar ausschließlich verwendbar für einzelne datenhaltende Systeme. Die veränderliche Datenbasis dieser operativen Systeme führt darüber hinaus zu dem Problem, dass Stichtagsberichte nicht reproduzierbar sind. Ist eine Nachvollziehbarkeit erforderlich oder die Entwicklung über Zeitverläufe von Relevanz, führt dies unter Umständen zu dezentral vorgehaltenen Selektionsergebnissen durch Einzelpersonen. Neben der problematischen Bindung von organisationsrelevanten Informationen an Personen, sind die Existenz und Verwahrung von erhobenen Daten und Informationen nicht transparent für Entscheidungsträger.

Abbildung 1 stellt schematisch einen Auszug dieser Strukturen der Informationsbeschaffung im dezentralen Berichtswesen der Fachhochschule Bielefeld dar. Neben vielfältigen Wegen macht die Abbildung deutlich, dass in einigen Szenarien die Informationsbeschaffung von verschiedenen Stellen erforderlich ist, um schließlich benötigte Information selbst zu generieren, beispielsweise für den jährlichen Rechenschaftsbericht. Dies verlangsamt Prozesse und führt unter Umständen zu verschiedenen Interpretationen und Abweichungen in kommunizierten Ergebnissen (Inkonsistenzen, Interpretationsfehler, Archivierungslücken).

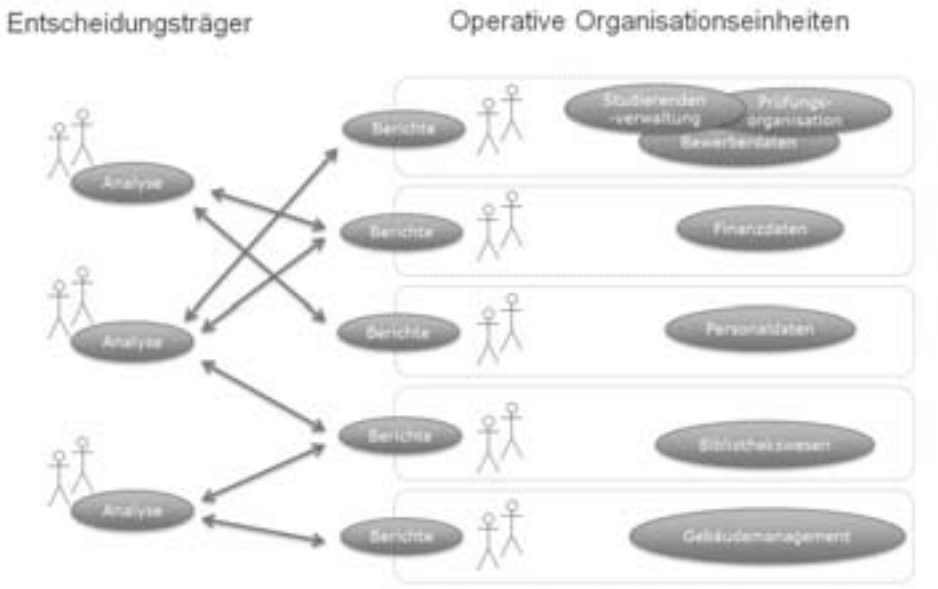


Abbildung 21: Dezentrales Berichtswesen an der FH Bielefeld

Insgesamt bietet sich ein Bild der Instabilität des Berichtswesens für Prozessbeteiligte und Adressaten auf organisatorischer wie auch auf technischer Ebene. Es existiert kein zentrales System, welches eine ganzheitliche Sicht auf die Organisation ermöglicht und fehlende Möglichkeiten zur Reproduzierbarkeit von Informationen (Entscheidungsgrundlagen) erschweren Erfolgskontrollen. Es ergibt sich der Bedarf nach einer technischen Lösung zur Restrukturierung des Berichtswesens, um der erkannten Notwendigkeit betriebswirtschaftlicher Steuerungsinstrumente zu begegnen.

3 Nutzen zentraler Data Warehouse Lösungen

Um die Probleme und die Einschränkungen des dezentralen Berichtswesens zu beseitigen und gleichzeitig ein modernes Führungsinstrument zur Verfügung zu haben, hat sich die FH Bielefeld im Jahr 2008 im Rahmen einer strategischen Qualitätsoffensive dazu entschlossen, ein Data Warehouse System aufzubauen.

3.1 Eigenschaften von Data Warehouse Systemen

Nach W. H. Inmon (In05) zeichnet sich ein Data Warehouse durch folgende vier charakteristische Eigenschaften aus:

- die Subjektorientierung, d.h. die Datenhaltung orientiert sich an den Informationsbedürfnissen des Managements und bietet Informationen zu den sie interessierenden Themenfeldern.
- die Integration, d.h. Daten aus unterschiedlichen operativen und externen Quellen werden zu einer widerspruchsfreien Datensammlung zusammengeführt.
- den Zeitraumbezug, d.h. es findet eine historisierte Datenspeicherung statt.
- Nicht-Volatilität, d.h. die Daten werden dauerhaft gespeichert und unterliegen keinen Änderungsoperationen.

Nach dieser Definition muss sich die Gestaltung eines Data Warehouses an zwei zentralen Punkten orientieren. Zum einen müssen die Informationsbedürfnisse der Entscheidungsträger ermittelt werden (*Subjektorientierung*). Zum anderen ist eine sorgfältige Analyse der operativen Vorsysteme und der darin abgebildeten Geschäftsprozesse erforderlich (*Integration*). Die Beachtung dieser beiden Prinzipien ist Voraussetzung dafür, dass das Data Warehouse auf lange Zeit den Benutzeranforderungen gerecht wird und dauerhaft betrieben werden kann.

Der langlebige Einsatz eines Data Warehouses hängt maßgeblich davon ab, dass die Informationsstrukturen so modelliert werden, dass sie das mögliche zukünftige Nutzungsverhalten antizipieren. Vor dem Hintergrund der Nachhaltigkeit bedeutet dies, dass die in der Entwicklungs- und Einführungsphase eines Data Warehouse Systems erforderlichen Aufwendungen sich in der anschließenden Betriebsphase wieder amortisieren. In dem, in diesem Arbeitsbericht vorgestellten Projekt der FH Bielefeld, wird davon ausgegangen, dass der zur Entwicklung und zum anschließenden Systembetrieb erforderliche Ressourceneinsatz in der anschließenden Nutzungsphase überkompensiert wird und sich ein positives Gesamtergebnis in Bezug auf die eingesetzten Mittel ergibt. Diese Annahme wird von zwei Thesen gestützt:

- Der aktuelle Einsatz von Personal und Technik zur Befriedigung der von der Politik geforderten Berichtspflicht ist extrem hoch. Die Erstellung von Einzellösungen ist unflexibel und erfordert sowohl eine fortlaufende Anpassung an sich ändernde Gegebenheiten der operativen Vorsysteme als auch eine ständige wiederkehrende Abbildung der Anforderungen der Informationskonsumenten.

- Entscheidungsprozesse stellen sich als sehr langwierig dar, da belastbare Informationen fehlen. Eine Vielzahl von Diskussionen basiert auf Vermutungen und Einzelfallbeobachtungen, die kaum tragfähig sind, um Entscheidungen von strategischer und langfristiger Bedeutung zu treffen. In der Vergangenheit wurden externe Dienstleister eingebunden, um Informationen aus den Informationssystemen der Hochschule zusammen zu tragen und für Entscheidungsprozesse aufzubereiten. Dadurch sind Kosten entstanden, die zukünftig beim Betrieb eines Data Warehouse vermieden werden können. Gleichzeitig stellt die Datenaufbereitung durch einen externen Dienstleister eine Momentaufnahme dar. Eine wiederholte Auswertung oder Fortschreibung der Ergebnisse ist nicht möglich.

3.2 Vorstellung des Projekts der FH Bielefeld

Im Rahmen der Qualitätsoffensive an der Fachhochschule Bielefeld galt es unter anderem, dem steigenden Informations- und Steuerungsbedarf innerhalb der Hochschule gerecht zu werden. Zudem wird der erkannte Bedarf durch gesetzliche Forderungen nach einem ganzheitlichen Controlling an Hochschulen, inkl. Kennzahlensteuerung und Berichtswesen[3], bekräftigt. Als geeignete Lösung zur Optimierung des Berichtswesens und Schaffung neuer Informationsstrukturen wurde die Einführung eines Data Warehouse Systems als Teilprojekt beschlossen.

Grundlegende Ziele für die Einführung des Data Warehouse Systems bildeten:

- Die Zentralisierung des Berichtswesens mit hochschulweiter Verfügbarkeit
- Reproduzierbarkeit von Stichtagsdaten und historischen Analysen
- Tagesaktuelle Kennzahlen (z.B. Bewerberzahlen, Budgetinformationen)
- Verknüpfung von Daten unterschiedlicher, heterogener Quellsysteme
- Die Möglichkeit zur Durchführung individueller Analysen integrierter Daten über Zeitverläufe (z.B. Studienverläufe)

Als geeignete Vorgehensweise zur Zielerreichung wurde die Kombination aus Top-Down- und Bottom-Up-Vorgehen[4] angesehen. Zum einen sollen demnach fachliche Anforderungen und objektiver Informationsbedarf die technische Implementierung bestimmen (top-down), zum anderen sollen parallele Analysen der Quellsysteme verfügbare Daten und Möglichkeiten zur Integration aufdecken (bottom-up).

Zum Erreichen definierter Ziele sind folglich Informationsbedarfsanalysen sowie die Etablierung von Arbeitsgruppen zur Beteiligung von Nutzern wesentlicher Bestandteil der Projektarbeit geworden. Bereits in der Phase der Konzeptionierung können so langfristige Bedarfsgerechtigkeit und Nutzerakzeptanz gesichert werden. Parallel dazu

[3] Vgl. § 5 Abs. 2 HFG NRW
[4] Vgl.[JSWW11] S. 18 ff.

durchgeführte und etablierte Validierungen von operativen Daten geben stets Aufschluss über deren Qualität und „fitness for use“.

Aus inhaltlichen, wie auch aus funktionalen Anforderungen ist ein Gesamtkonzept zur Integration und Auswertung organisationsweit anfallender Daten entstanden, welches die folgende Abbildung 22 zeigt. Als Datenbasis am unteren Ende steht ein Data Warehouse, dem umfassende Logik zur Validierung und Integration faktisch anonymer Daten aus den Quellsystemen zugrunde liegt. Auf dieser Datengrundlage werden sowohl die Abbildung statischer Berichte als auch komplexe Analysen innerhalb eigens modellierter, auswertungsorientierter Themenbereiche ermöglicht, deren Implementierung etwa der virtueller, abhängiger Data Marts[5] entspricht.

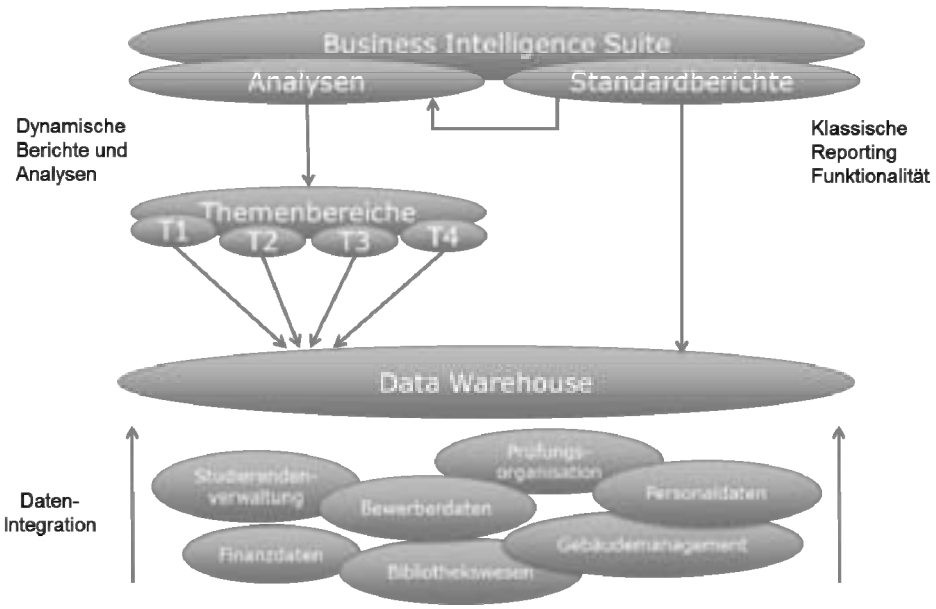


Abbildung 22: Konzeptioneller Aufbau zur DWH Nutzung

Diese Analysen können einfache Abfragen tagesaktueller Absolventenzahlen umfassen, können aber auch als Grundlage zu komplexeren Analysen wie Studienverlaufsanalysen dienen. Über die einzelnen Themenbereiche hinweg kann ein Kennzahlensystem zur ganzheitlichen Hochschulsteuerung abgebildet werden, insbesondere auch zur effektiven Steuerung der Mittelverwendung im Rahmen der Finanzierung und Wirtschaftsführung. Beispielsweise können Kennzahlen wie „Drittmittel je Professor“ oder „Laufende Kosten je Studienplatz“ gebildet werden oder die „Laufenden Ausgaben für ein erfolgreich abgeschlossenes Studium“ analysiert werden.

[5] vgl. [BG09] S. 61 ff.

Übliche Standardberichte aus den verschiedenen Bereichen der Hochschule sind hingegen eher statisch und können auf Basis von konkreten Designvorlagen dargestellt werden. Dies kann beispielsweise Studierendenzahlen oder Budgetinformationen umfassen, die nach dem Corporate Design angefertigt werden. Solche Berichte können unter anderem auch vollständig automatisiert und termingesteuert per E-Mail als PDF-Datei versandt werden. Vereinigt werden die Bereiche des klassischen Berichtswesens und der Funktionalität für dynamische Berichte und Analysen in der „Business Intelligence Suite", der grafischen Oberfläche für Anwender.

Über die dargestellten Inhalte hinaus ermöglicht das Konzept um das Data Warehouse an der FH Bielefeld einige weitere Optimierungen des Berichtswesens. Beispielsweise bedeutet die Zentralisierung und einhergehende Datenoptimierung die Verkürzung von Informationsbeschaffungsprozessen, da Mitarbeiter auf benötigte Informationen stets direkten Zugriff haben und nicht auf Kapazitäten verschiedener Organisationseinheiten angewiesen sind.

Zudem können gewonnene Informationen künftig neue Steuerungsmöglichkeiten etablieren, welche bislang auf Annahmen basieren mussten. Beispielsweise kann im Rahmen des Hochschulmarketing auf Fragen und Aspekte eingegangen werden, wie

- „Woher kommen erfolgreiche Studierende"?
- „Welche Art der HZB[6] haben erfolgreiche Studierende in bestimmten Studiengängen?"

Aber auch Analysen von Prüfungsinformationen können wertvolle Erkenntnisse zur Verbesserung von Studium und Lehre liefern.

- Sind bestimmte Prüfungsleistungen ein Indikator für Studienerfolg / Misserfolg?
- Welche Prüfungen werden häufig nicht nach dem Studienverlaufsplan absolviert?

Neben dem direkten Mehrwert durch den Betrieb der Business Intelligence Suite ergeben sich weitere positive Auswirkungen auf Quellsysteme und operative Prozesse. So kommt beispielsweise der Datenqualität in eigenständigen, operativen Systemen meist eine geringere Bedeutung zu als im organisationsweiten Berichtswesen. Hintergrund ist, dass fehlerhafte Einzeldatensätze von Sachbearbeitern mit Fachwissen häufig noch interpretiert werden können, bei der Aggregation für das Berichtswesen allerdings Unschärfe in Kennzahlen entstehen kann. Deren Ursache ist nachträglich gar nicht oder nur schwer zu erkennen. Zudem sind Inkonsistenzen zu Daten und Prozessen anderer Organisationseinheiten auf operativer Ebene in der Regel nicht erkennbar.

[6] Die Hochschulzugangsberechtigung (HZB) für ein Studium an Fachhochschulen kann auf vielfältige Weise erworben werden. Neben der allgemeinen Hochschulreife oder der Fachhochschulreife zählt hierzu beispielsweise auch durch berufliche Qualifikation.

Bei der Integration der Daten in das Data Warehouse können und müssen alle Daten einigen Validierungen unterzogen werden, in deren Verlauf Inkonsistenzen auffallen. Datenfehler und Unvollständigkeiten können (automatisiert) an die datenführenden Stellen kommuniziert und dort korrigiert werden[7]. Neben der Verbesserung der Qualität des Berichtswesens, führt die Verbesserung der Datenqualität auch im operativen Bereich zu einer Erhöhung der Effizienz von Arbeitsabläufen.

Über die elementare Datenqualität hinaus können bei der Integration von Daten in das Data Warehouse auch Unstimmigkeiten in der Abbildung von Prozessen in den datenführenden Stellen erkannt werden. Beispielsweise sind dies unterschiedliche Vermerke im System für erbrachte Zusatzleistungen oder für Rücktritte von angemeldeten Prüfungen im Rahmen des Prüfungswesens. Die Aufdeckung dieser unterschiedlichen Abbildungen gleicher Geschäftsvorfälle im Rahmen des Data Warehouse führt zu einer Vereinheitlichung und Verstetigung der definierten Prozesse.

Prozesse, die zu dezentral gespeicherten Daten durch Einzelpersonen führen, sind grundsätzlich als problematisch anzusehen. Beispielsweise ist das im Rahmen der Bewerberdatenerfassung an der FH Bielefeld unter verschiedenen Umständen noch der Fall, was aber zu Problemen unter Aspekten der Datensicherheit und Ineffizienz im Berichtswesen führt. Im Idealfall werden erhobene Daten zentral gespeichert, zugriffsgeschützt und gegen Verlust gesichert. Da dies im Rahmen des Data Warehouse ebenfalls einen Vorteil zur Datenintegration mit sich bringt, ist parallel zur Business Intelligence Suite ein zentrales System aufgebaut worden, das die dezentrale Datenerfassung durch flexible, zugriffsgeschützte Oberflächen per Webbrowser ermöglicht. Informationen sind so personenunabhängig in operativen Prozessen verfügbar und die Vollständigkeit der Datenbasis für das Data Warehouse wird gewährleistet. Noch darüber hinaus wird die Anreicherung der Datenbasis um Plandaten sowie Zuordnungen für Auswertungen ermöglicht, wie beispielsweise der auswertungsorientierten Gruppierung von Kostenstellen.

4 Fazit

Die Entwicklungen der letzten Jahre führten dazu, dass in der Hochschullandschaft neuer Steuerungsbedarf entstand, dem ohne eine ganzheitliche Sicht auf die Organisation nicht begegnet werden kann. Ein klassisches, dezentral organisiertes Berichtswesen zeigt sich dabei als nicht geeignet, um effizient und Ressourcen schonend den steigenden Informationsbedarf zu decken.

Durch ein Data Warehouse als zentrale Datenbasis für das Berichtswesen wird die ganzheitliche Betrachtung der Organisation ermöglicht. Durch Entkopplung des Berichtswesens von operativen Systemen wird dabei eine langlebige und stabile Datenbasis geschaffen, die unabhängig von Strukturen operativer Systeme ist und die Reproduzierbarkeit von Informationen gewährleistet.

[7] Vgl. [KC04] S.123 ff.

Durch die frühzeitige Einbindung von Entscheidungsträgern und Betroffenen im Allgemeinen in die stetige Weiterentwicklung wird eine hohe Bedarfsgerechtigkeit und daraus resultierende Nutzerakzeptanz erreicht. Mit dieser Arbeit in Teilprojektgruppen verlagern sich zudem individuelle Kenntnisse zu Organisationswissen, was der nachhaltigen Definition und Verstetigung von Prozessen, auch über die Informationsbeschaffung hinaus, zugute kommt.

Insgesamt ist durch die Zentralisierung der Informationsquellen, die Vereinheitlichung des Berichtswesens in technischer Hinsicht und das Management der Datenqualität ein System etabliert worden, das durch Langlebigkeit, Transparenz und Schonung von Ressourcen eine nachhaltige Lösung für heutige Probleme der Hochschulsteuerung bietet. In wie weit sich eine weitere Verbesserung der Prozessqualität auf operativer Ebene einstellt, wird sich in Zukunft zeigen.

Literaturverzeichnis

[BG11] Bauer, Andreas; Günzel, Holger: Data Warehouse Systeme, 3. Auflage, Heidelberg, 2009

[BMB12] Bundesministerium für Bildung und Forschung: Bericht über die Umsetzung des Bologna-Prozesses in Deutschland, Berlin, 2012; (http://www.bmbf.de/pubRD/umsetzung_bologna_prozess_2012.pdf; abgerufen am 02.05.2012)

[HFG06] Ministerium für Innovation, Wissenschaft und Forschung NRW: Hochschulfreiheitsgesetz NRW, Düsseldorf, 2006; (http://www.wissenschaft.nrw.de/objektpool/download_dateien/hochschulen_und_forschung/HFG.pdf; abgerufen am 02.05.2012)

[HRK04] Entschließung des 202. Plenums der HRK: Professionalisierung als Leitungsaufgabe, Bonn, 2004; (http://www.hrk.de/de/download/dateien/Beschluss_Plenum_8.6.2004.pdf; abgerufen am 02.05.2012)

[In05] Inmon, W.H.: Building der Data Warehouse, John Willey&Sons, Indianapolis, 2004

[JSWW11] Jordan, Claus; Schnider, Dani; Wehner, Joachim; Welker, Peter: Data Warehousing mit Oracle, München, 2011

[KC04] Kimball, Ralph; Casserta, Joe; The Data Warehouse ETL Toolkit – Practical Techniques for Extracting, Cleaning, Conforming, and Delivering Data, Wiley Publishing, Indianapolis, 2004

Intranets erfolgreich entwickeln und betreiben

Christian Weber, Hans Brandt-Pook

Nionex GmbH
Ringstr. 16-20
Rheda-Wiedenbrück
christian.weber@bertelsmann.de

FH Bielefeld
Universitätsstr. 25
33615 Bielefeld
hans.brandt-pook@fh-bielefeld.de

Beschreibung des Tutorials

Intranets bilden heute ein Kernstück in der unternehmensinternen Kommunikation. Oft sind sie sogar Dreh- und Angelpunkt für das Wissensmanagement im Unternehmen. Das Tutorial befasst sich mit der Frage, wie Intranets erfolgreich entwickelt und betrieben werden können.

Intranets können je nach Anforderungen unterschiedlich ausgeprägt sein. Im Kern steht dabei eine klassische interne Informationszentrale, auf die Mitarbeitende eines Unternehmens mittels eines Content Management Systems zugreifen können. Zunehmend werden allerdings weitergehende Anforderungen an ein Intranet formuliert und können auch heute bereitgestellt werden:

- Neue Anforderungen werden an die Unterstützung der Zusammenarbeit im Unternehmen gestellt. Die gemeinsame Struktur für Projekte oder die Wissensbasis eines Teams sollte abbildbar und rollenspezifisch zugreifbar sein.

- Eine komfortable Suche soll nicht nur vielfältige funktionale Anforderungen wie bspw. inkrementelle Vorschläge oder automatische Rechtschreibkorrektur bieten sondern auch weitere interne Quellen bei der Suche einbeziehen können.

- Eine umfassende Portalfunktion soll den Beschäftigten einen einzigen, wohldefinierten Zugangspunkt zu den zentralen Informationsquellen und Anwendungen eines Unternehmens bieten.

- Schließlich geht es darum, die Kultur der öffentlichen sozialen Netzwerke auch im Intranet abzubilden. In einem Social Intranet haben Mitarbeitende die Möglichkeit auf der Basis von persönlichen Profilen ihre Aktivitäten zu vernetzen und so die Zusammenarbeit zu optimieren.

Nach einem einleitenden Vortrag steht der Meinungs- und Erfahrungsaustausch im Mittelpunkt des Tutorials. Fragen, die je nach Bedarf der Teilnehmenden vertieft werden könnten, sind bspw.:

- Welche aktuellen Anforderungen an Intranets werden gesehen?
- Wie können die aktuellen Anforderungen organisatorisch und technisch umgesetzt werden?
- Wie kann ein Entwicklungsprojekt erfolgreich gestaltet werden?
- Wie kann der Erfolg eines Intranets gemessen werden?

Das Tutorial bietet Platz für verschiedene Sichten auf das Thema und richtet sich daher sowohl an Verantwortliche und Mitarbeitende auf Seiten des Intranet-Betreibers als auch auf Seiten des IT-Dienstleisters.

Referenten

Christian Weber ist Teamleiter Content Management Systeme bei dem IT-Dienstleister Nionex, einem Unternehmen der Bertelsmann SE & Co. KGaA. In zahlreichen Projekten war Herr Weber verantwortlich für die Konzeption und Umsetzung von Intranets für große und mittelständische Unternehmen.

Hans Brandt-Pook ist Professor für Wirtschaftsinformatik, insbesondere eCommerce und WebApplikationen, an der FH Bielefeld.

Einführende Literatur

[SCN11] Schade, A.; Caya, P.; Nielsen, J.: Intranet Design Annual 2011 - The Year's 10 Best Intranets, Nielsen Norman Group, 2011.

[Wo11] Wolf, F.: Social Intranet – Kommunikation fördern, Wissen teilen, Effizient zusammenarbeiten, Carl Hanser Verlag, München, 2011.

ITIL an der Hochschule, Praxiserfahrungen und Austausch

Thomas Degenhardt

Fachhochschule Bielefeld
Universitätsstraße 25
33615 Bielefeld
thomas.degenhardt@fh-bielefeld.de

Abstract: ITIL ist in vielen Unternehmen ein etabliertes und gelebtes Framework. Aber wie ist der Status Quo an den Hochschulen? Die IT an den meisten Hochschulen ist ebenfalls einerseits Kostenzwängen unterworfen, muss aber andererseits mehr leisten denn je. Ist die Einführung von ITIL ein Allheilmittel? Dies soll in einer Diskussionsrunde vertieft werden.

Inhalt des Tutorials

Viele Hochschulen setzen sich mit der IT-Service-Management-Thematik auseinander. ITIL ist mittlerweile der de Facto-Standard bei der Implementierung von IT-Servicemanagement-Prozessen (ITSM). Auch an der Fachhochschule Bielefeld ist dies ein Thema:

Die FH Bielefeld verfügt derzeit über sieben Standorte in drei Städten. An diesen Standorten ist jeweils eine eigene, dezentrale IT vorhanden, die vor Ort tätig ist. Zentrale Services, wie z.B. Identity Management, E-Mail oder Verzeichnisdienste werden zentral durch die DV-Zentrale für alle Standorte bereitgestellt. Dadurch ergaben sich in Vergangenheit zwangsweise Problematiken wie Intransparenz bzgl. der Servicebereitstellung und -unterstützung (Wer kann mir helfen?) und der Generierung von „Wissensinseln" für bestimmte Problematiken und Einsatzgebiete. Diese und weitere Problematiken führten zu der Entscheidung, sich mit dem Thema ITSM in Form von ITIL v3 zu beschäftigen. Nach der Bildung eines Projektteams wurde eine Umsetzungsplanung erarbeitet und entschieden, die ITIL-Einführung mit dem Prozess Incident Management und der Funktion des ServiceDesks als „Single Point of Contact (SPOC)"[1] zu starten. Dieser Prozess ist mittlerweile für nahezu alle Services und Standorte der Hochschule implementiert und ist im Rahmen des Continual Service Improvements in einem stetigen Veränderungsprozess. Weitere Prozesse des „Service Design" wie Service Level- und Service Catalog Management sowie die Prozesse Change Management und Service Asset und Configuration Management der Phase „Service Transition" des ITIL Service Lifecycle[2] sollen in Kürze folgen. In einem einleitenden Vortrag wird dieser bisherige „ITIL-Weg" skizziert und das weitere Vorgehen vorgestellt.

[1] Vgl. [OGC07d] S. 128
[2] Vgl. [OGC07a] S. 8 ff.

Nach diesem Vortrag soll die Vorgehensweise in einem „Round Table“ diskutiert werden. Teilnehmer anderer Hochschulen sind eingeladen, ihre „Good Practices“ in diesem Rahmen vorzustellen.

Es soll im Detail auf die folgenden Aspekte eingegangen werden:

- Der „interne“ Blickwinkel: Reorganisation der Prozesse
  - „Mitnahme“ des Personals
  - Reihenfolge der Einführung der Prozesse
  - Toolauswahl
  - Datenschutz, Dienstvereinbarungen

- Der „externe“ Blick: Kunden-Lieferantenbeziehungen
  - Die Hochschul-IT als Service-Provider
  - Bereitstellen von Services statt purer Technik

- Erfahrungen/Ergebnisse
  - Quickwins
  - Kennzahlen
  - Weiteres Vorgehen

2 Zielgruppe

Das Tutorial richtet sich sowohl an EntscheiderInnen als auch IT-MitarbeiterInnen, die aktiv an der Planung und Implementierung von ITSM-Prozessen beteiligt sind. Es sind auch Hochschulen eingeladen teilzunehmen, die sich bisher noch nicht mit dem IT-Servicemanagement auseinandergesetzt haben.

3 Referent

Thomas Degenhardt ist Mitarbeiter in der DV-Zentrale der Fachhochschule Bielefeld. Er ist maßgeblich beteiligt an dem ITIL-Einführungsprozess an der FH Bielefeld und derzeit u.a. in der Rolle des ServiceDesk-Managers an der Hochschule tätig. Im Rahmen der ITIL-Einführung erlangte Herr Degenhardt eine Zertifizierung als „ITIL Expert“.

Literatur

[OGC07a] Office of Government Commerce (Hrsg.): ITIL Service Strategy. Deutschsprachige Ausgabe. TSO The Stationary Office, Norwich 2007.

[OGC07d] Office of Government Commerce (Hrsg.): ITIL Service Operation. Deutschsprachige Ausgabe. TSO The Stationary Office, Norwich 2007.

[ZSOW10] Zielke, F; Schinkel, A.; Oldag J.;Weber G.:ITIL überzeugend einführen – Methoden und soziale Kompetenzen. Symposium Publishing 2010

GI-Edition Lecture Notes in Informatics

P-1 Gregor Engels, Andreas Oberweis, Albert Zündorf (Hrsg.): Modellierung 2001.

P-2 Mikhail Godlevsky, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications, ISTA'2001.

P-3 Ana M. Moreno, Reind P. van de Riet (Hrsg.): Applications of Natural Lan-guage to Information Systems, NLDB'2001.

P-4 H. Wörn, J. Mühling, C. Vahl, H.-P. Meinzer (Hrsg.): Rechner- und sensorgestützte Chirurgie; Workshop des SFB 414.

P-5 Andy Schürr (Hg.): OMER – Object-Oriented Modeling of Embedded Real-Time Systems.

P-6 Hans-Jürgen Appelrath, Rolf Beyer, Uwe Marquardt, Heinrich C. Mayr, Claudia Steinberger (Hrsg.): Unternehmen Hochschule, UH'2001.

P-7 Andy Evans, Robert France, Ana Moreira, Bernhard Rumpe (Hrsg.): Practical UML-Based Rigorous Development Methods – Countering or Integrating the extremists, pUML'2001.

P-8 Reinhard Keil-Slawik, Johannes Magenheim (Hrsg.): Informatikunterricht und Medienbildung, INFOS'2001.

P-9 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Innovative Anwendungen in Kommunikationsnetzen, 15. DFN Arbeitstagung.

P-10 Mirjam Minor, Steffen Staab (Hrsg.): 1st German Workshop on Experience Management: Sharing Experiences about the Sharing Experience.

P-11 Michael Weber, Frank Kargl (Hrsg.): Mobile Ad-Hoc Netzwerke, WMAN 2002.

P-12 Martin Glinz, Günther Müller-Luschnat (Hrsg.): Modellierung 2002.

P-13 Jan von Knop, Peter Schirmbacher and Viljan Mahni_ (Hrsg.): The Changing Universities – The Role of Technology.

P-14 Robert Tolksdorf, Rainer Eckstein (Hrsg.): XML-Technologien für das Semantic Web – XSW 2002.

P-15 Hans-Bernd Bludau, Andreas Koop (Hrsg.): Mobile Computing in Medicine.

P-16 J. Felix Hampe, Gerhard Schwabe (Hrsg.): Mobile and Collaborative Business 2002.

P-17 Jan von Knop, Wilhelm Haverkamp (Hrsg.): Zukunft der Netze –Die Verletzbarkeit meistern, 16. DFN Arbeitstagung.

P-18 Elmar J. Sinz, Markus Plaha (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2002.

P-19 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund.

P-20 Sigrid Schubert, Bernd Reusch, Norbert Jesse (Hrsg.): Informatik bewegt – Informatik 2002 – 32. Jahrestagung der Gesellschaft für Informatik e.V. (GI) 30.Sept.-3. Okt. 2002 in Dortmund (Ergänzungsband).

P-21 Jörg Desel, Mathias Weske (Hrsg.): Promise 2002: Prozessorientierte Methoden und Werkzeuge für die Entwicklung von Informationssystemen.

P-22 Sigrid Schubert, Johannes Magenheim, Peter Hubwieser, Torsten Brinda (Hrsg.): Forschungsbeiträge zur "Didaktik der Informatik" – Theorie, Praxis, Evaluation.

P-23 Thorsten Spitta, Jens Borchers, Harry M. Sneed (Hrsg.): Software Management 2002 – Fortschritt durch Beständigkeit

P-24 Rainer Eckstein, Robert Tolksdorf (Hrsg.): XMIDX 2003 – XML-Technologien für Middleware – Middleware für XML-Anwendungen

P-25 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Commerce – Anwendungen und Perspektiven – 3. Workshop Mobile Commerce, Universität Augsburg, 04.02.2003

P-26 Gerhard Weikum, Harald Schöning, Erhard Rahm (Hrsg.): BTW 2003: Datenbanksysteme für Business, Technologie und Web

P-27 Michael Kroll, Hans-Gerd Lipinski, Kay Melzer (Hrsg.): Mobiles Computing in der Medizin

P-28 Ulrich Reimer, Andreas Abecker, Steffen Staab, Gerd Stumme (Hrsg.): WM 2003: Professionelles Wissensmanagement – Er-fahrungen und Visionen

P-29 Antje Düsterhöft, Bernhard Thalheim (Eds.): NLDB'2003: Natural Language Processing and Information Systems

P-30 Mikhail Godlevsky, Stephen Liddle, Heinrich C. Mayr (Eds.): Information Systems Technology and its Applications

P-31 Arslan Brömme, Christoph Busch (Eds.): BIOSIG 2003: Biometrics and Electronic Signatures

P-32 Peter Hubwieser (Hrsg.): Informatische Fachkonzepte im Unterricht – INFOS 2003

P-33 Andreas Geyer-Schulz, Alfred Taudes (Hrsg.): Informationswirtschaft: Ein Sektor mit Zukunft

P-34 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 1)

P-35 Klaus Dittrich, Wolfgang König, Andreas Oberweis, Kai Rannenberg, Wolfgang Wahlster (Hrsg.): Informatik 2003 – Innovative Informatikanwendungen (Band 2)

P-36 Rüdiger Grimm, Hubert B. Keller, Kai Rannenberg (Hrsg.): Informatik 2003 – Mit Sicherheit Informatik

P-37 Arndt Bode, Jörg Desel, Sabine Rathmayer, Martin Wessner (Hrsg.): DeLFI 2003: e-Learning Fachtagung Informatik

P-38 E.J. Sinz, M. Plaha, P. Neckel (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2003

P-39 Jens Nedon, Sandra Frings, Oliver Göbel (Hrsg.): IT-Incident Management & IT-Forensics – IMF 2003

P-40 Michael Rebstock (Hrsg.): Modellierung betrieblicher Informationssysteme – MobIS 2004

P-41 Uwe Brinkschulte, Jürgen Becker, Dietmar Fey, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle, Thomas Runkler (Edts.): ARCS 2004 – Organic and Pervasive Computing

P-42 Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Economy – Transaktionen und Prozesse, Anwendungen und Dienste

P-43 Birgitta König-Ries, Michael Klein, Philipp Obreiter (Hrsg.): Persistance, Scalability, Transactions – Database Mechanisms for Mobile Applications

P-44 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): Security, E-Learning. E-Services

P-45 Bernhard Rumpe, Wofgang Hesse (Hrsg.): Modellierung 2004

P-46 Ulrich Flegel, Michael Meier (Hrsg.): Detection of Intrusions of Malware & Vulnerability Assessment

P-47 Alexander Prosser, Robert Krimmer (Hrsg.): Electronic Voting in Europe – Technology, Law, Politics and Society

P-48 Anatoly Doroshenko, Terry Halpin, Stephen W. Liddle, Heinrich C. Mayr (Hrsg.): Information Systems Technology and its Applications

P-49 G. Schiefer, P. Wagner, M. Morgenstern, U. Rickert (Hrsg.): Integration und Datensicherheit – Anforderungen, Konflikte und Perspektiven

P-50 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 1) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-51 Peter Dadam, Manfred Reichert (Hrsg.): INFORMATIK 2004 – Informatik verbindet (Band 2) Beiträge der 34. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 20.-24. September 2004 in Ulm

P-52 Gregor Engels, Silke Seehusen (Hrsg.): DELFI 2004 – Tagungsband der 2. e-Learning Fachtagung Informatik

P-53 Robert Giegerich, Jens Stoye (Hrsg.): German Conference on Bioinformatics – GCB 2004

P-54 Jens Borchers, Ralf Kneuper (Hrsg.): Softwaremanagement 2004 – Outsourcing und Integration

P-55 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): E-Science und Grid Adhoc-Netze Medienintegration

P-56 Fernand Feltz, Andreas Oberweis, Benoit Otjacques (Hrsg.): EMISA 2004 – Informationssysteme im E-Business und E-Government

P-57 Klaus Turowski (Hrsg.): Architekturen, Komponenten, Anwendungen

P-58 Sami Beydeda, Volker Gruhn, Johannes Mayer, Ralf Reussner, Franz Schweiggert (Hrsg.): Testing of Component-Based Systems and Software Quality

P-59 J. Felix Hampe, Franz Lehner, Key Pousttchi, Kai Ranneberg, Klaus Turowski (Hrsg.): Mobile Business – Processes, Platforms, Payments

P-60 Steffen Friedrich (Hrsg.): Unterrichtskonzepte für inforrmatische Bildung

P-61 Paul Müller, Reinhard Gotzhein, Jens B. Schmitt (Hrsg.): Kommunikation in verteilten Systemen

P-62 Federrath, Hannes (Hrsg.): „Sicherheit 2005“ – Sicherheit – Schutz und Zuverlässigkeit

P-63 Roland Kaschek, Heinrich C. Mayr, Stephen Liddle (Hrsg.): Information Systems – Technology and ist Applications

P-64 Peter Liggesmeyer, Klaus Pohl, Michael Goedicke (Hrsg.): Software Engineering 2005

P-65 Gottfried Vossen, Frank Leymann, Peter Lockemann, Wolffried Stucky (Hrsg.): Datenbanksysteme in Business, Technologie und Web

P-66 Jörg M. Haake, Ulrike Lucke, Djamshid Tavangarian (Hrsg.): DeLFI 2005: 3. deutsche e-Learning Fachtagung Informatik

P-67 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 1)

P-68 Armin B. Cremers, Rainer Manthey, Peter Martini, Volker Steinhage (Hrsg.): INFORMATIK 2005 – Informatik LIVE (Band 2)

P-69 Robert Hirschfeld, Ryszard Kowalcyk, Andreas Polze, Matthias Weske (Hrsg.): NODe 2005, GSEM 2005

P-70 Klaus Turowski, Johannes-Maria Zaha (Hrsg.): Component-oriented Enterprise Application (COAE 2005)

P-71 Andrew Torda, Stefan Kurz, Matthias Rarey (Hrsg.): German Conference on Bioinformatics 2005

P-72 Klaus P. Jantke, Klaus-Peter Fähnrich, Wolfgang S. Wittig (Hrsg.): Marktplatz Internet: Von e-Learning bis e-Payment

P-73 Jan von Knop, Wilhelm Haverkamp, Eike Jessen (Hrsg.): "Heute schon das Morgen sehen"

P-74 Christopher Wolf, Stefan Lucks, Po-Wah Yau (Hrsg.): WEWoRC 2005 – Western European Workshop on Research in Cryptology

P-75 Jörg Desel, Ulrich Frank (Hrsg.): Enterprise Modelling and Information Systems Architecture

P-76 Thomas Kirste, Birgitta König-Riess, Key Pousttchi, Klaus Turowski (Hrsg.): Mobile Informationssysteme – Potentiale, Hindernisse, Einsatz

P-77 Jana Dittmann (Hrsg.): SICHERHEIT 2006

P-78 K.-O. Wenkel, P. Wagner, M. Morgenstern, K. Luzi, P. Eisermann (Hrsg.): Land- und Ernährungswirtschaft im Wandel

P-79 Bettina Biel, Matthias Book, Volker Gruhn (Hrsg.): Softwareengineering 2006

P-80 Mareike Schoop, Christian Huemer, Michael Rebstock, Martin Bichler (Hrsg.): Service-Oriented Electronic Commerce

P-81 Wolfgang Karl, Jürgen Becker, Karl-Erwin Großpietsch, Christian Hochberger, Erik Maehle (Hrsg.): ARCS´06

P-82 Heinrich C. Mayr, Ruth Breu (Hrsg.): Modellierung 2006

P-83 Daniel Huson, Oliver Kohlbacher, Andrei Lupas, Kay Nieselt and Andreas Zell (eds.): German Conference on Bioinformatics

P-84 Dimitris Karagiannis, Heinrich C. Mayr, (Hrsg.): Information Systems Technology and its Applications

P-85 Witold Abramowicz, Heinrich C. Mayr, (Hrsg.): Business Information Systems

P-86 Robert Krimmer (Ed.): Electronic Voting 2006

P-87 Max Mühlhäuser, Guido Rößling, Ralf Steinmetz (Hrsg.): DELFI 2006: 4. e-Learning Fachtagung Informatik

P-88 Robert Hirschfeld, Andreas Polze, Ryszard Kowalczyk (Hrsg.): NODe 2006, GSEM 2006

P-90 Joachim Schelp, Robert Winter, Ulrich Frank, Bodo Rieger, Klaus Turowski (Hrsg.): Integration, Informationslogistik und Architektur

P-91 Henrik Stormer, Andreas Meier, Michael Schumacher (Eds.): European Conference on eHealth 2006

P-92 Fernand Feltz, Benoît Otjacques, Andreas Oberweis, Nicolas Poussing (Eds.): AIM 2006

P-93 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 1

P-94 Christian Hochberger, Rüdiger Liskowsky (Eds.): INFORMATIK 2006 – Informatik für Menschen, Band 2

P-95 Matthias Weske, Markus Nüttgens (Eds.): EMISA 2005: Methoden, Konzepte und Technologien für die Entwicklung von dienstbasierten Informationssystemen

P-96 Saartje Brockmans, Jürgen Jung, York Sure (Eds.): Meta-Modelling and Ontologies

P-97 Oliver Göbel, Dirk Schadt, Sandra Frings, Hardo Hase, Detlef Günther, Jens Nedon (Eds.): IT-Incident Mangament & IT-Forensics – IMF 2006

P-98 Hans Brandt-Pook, Werner Simonsmeier und Thorsten Spitta (Hrsg.): Beratung in der Softwareentwicklung – Modelle, Methoden, Best Practices

P-99 Andreas Schwill, Carsten Schulte, Marco Thomas (Hrsg.): Didaktik der Informatik

P-100 Peter Forbrig, Günter Siegel, Markus Schneider (Hrsg.): HDI 2006: Hochschuldidaktik der Informatik

P-101 Stefan Böttinger, Ludwig Theuvsen, Susanne Rank, Marlies Morgenstern (Hrsg.): Agrarinformatik im Spannungsfeld zwischen Regionalisierung und globalen Wertschöpfungsketten

P-102 Otto Spaniol (Eds.): Mobile Services and Personalized Environments

P-103 Alfons Kemper, Harald Schöning, Thomas Rose, Matthias Jarke, Thomas Seidl, Christoph Quix, Christoph Brochhaus (Hrsg.): Datenbanksysteme in Business, Technologie und Web (BTW 2007)

P-104 Birgitta König-Ries, Franz Lehner, Rainer Malaka, Can Türker (Hrsg.) MMS 2007: Mobilität und mobile Informationssysteme

P-105 Wolf-Gideon Bleek, Jörg Raasch, Heinz Züllighoven (Hrsg.) Software Engineering 2007

P-106 Wolf-Gideon Bleek, Henning Schwentner, Heinz Züllighoven (Hrsg.) Software Engineering 2007 – Beiträge zu den Workshops

P-107 Heinrich C. Mayr, Dimitris Karagiannis (eds.) Information Systems Technology and its Applications

P-108 Arslan Brömme, Christoph Busch, Detlef Hühnlein (eds.) BIOSIG 2007: Biometrics and Electronic Signatures

P-109 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 1

P-110 Rainer Koschke, Otthein Herzog, Karl-Heinz Rödiger, Marc Ronthaler (Hrsg.) INFORMATIK 2007 Informatik trifft Logistik Band 2

P-111 Christian Eibl, Johannes Magenheim, Sigrid Schubert, Martin Wessner (Hrsg.) DeLFI 2007: 5. e-Learning Fachtagung Informatik

P-112 Sigrid Schubert (Hrsg.) Didaktik der Informatik in Theorie und Praxis

P-113 Sören Auer, Christian Bizer, Claudia Müller, Anna V. Zhdanova (Eds.) The Social Semantic Web 2007 Proceedings of the 1st Conference on Social Semantic Web (CSSW)

P-114 Sandra Frings, Oliver Göbel, Detlef Günther, Hardo G. Hase, Jens Nedon, Dirk Schadt, Arslan Brömme (Eds.) IMF2007 IT-incident management & IT-forensics Proceedings of the 3rd International Conference on IT-Incident Management & IT-Forensics

P-115 Claudia Falter, Alexander Schliep, Joachim Selbig, Martin Vingron and Dirk Walther (Eds.) German conference on bioinformatics GCB 2007

P-116 Witold Abramowicz, Leszek Maciszek (Eds.) Business Process and Services Computing 1st International Working Conference on Business Process and Services Computing BPSC 2007

P-117 Ryszard Kowalczyk (Ed.) Grid service engineering and manegement The 4th International Conference on Grid Service Engineering and Management GSEM 2007

P-118 Andreas Hein, Wilfried Thoben, Hans-Jürgen Appelrath, Peter Jensch (Eds.) European Conference on ehealth 2007

P-119 Manfred Reichert, Stefan Strecker, Klaus Turowski (Eds.) Enterprise Modelling and Information Systems Architectures Concepts and Applications

P-120 Adam Pawlak, Kurt Sandkuhl, Wojciech Cholewa, Leandro Soares Indrusiak (Eds.) Coordination of Collaborative Engineering - State of the Art and Future Challenges

P-121 Korbinian Herrmann, Bernd Bruegge (Hrsg.) Software Engineering 2008 Fachtagung des GI-Fachbereichs Softwaretechnik

P-122 Walid Maalej, Bernd Bruegge (Hrsg.) Software Engineering 2008 - Workshopband Fachtagung des GI-Fachbereichs Softwaretechnik

P-123 Michael H. Breitner, Martin Breunig, Elgar Fleisch, Ley Pousttchi, Klaus Turowski (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Technologien, Prozesse, Marktfähigkeit
Proceedings zur 3. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2008)

P-124 Wolfgang E. Nagel, Rolf Hoffmann, Andreas Koch (Eds.)
9th Workshop on Parallel Systems and Algorithms (PASA)
Workshop of the GI/ITG Speciel Interest Groups PARS and PARVA

P-125 Rolf A.E. Müller, Hans-H. Sundermeier, Ludwig Theuvsen, Stephanie Schütze, Marlies Morgenstern (Hrsg.)
Unternehmens-IT:
Führungsinstrument oder Verwaltungsbürde
Referate der 28. GIL Jahrestagung

P-126 Rainer Gimnich, Uwe Kaiser, Jochen Quante, Andreas Winter (Hrsg.)
10th Workshop Software Reengineering (WSR 2008)

P-127 Thomas Kühne, Wolfgang Reisig, Friedrich Steimann (Hrsg.)
Modellierung 2008

P-128 Ammar Alkassar, Jörg Siekmann (Hrsg.)
Sicherheit 2008
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 4. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)
2.-4. April 2008
Saarbrücken, Germany

P-129 Wolfgang Hesse, Andreas Oberweis (Eds.)
Sigsand-Europe 2008
Proceedings of the Third AIS SIGSAND European Symposium on Analysis, Design, Use and Societal Impact of Information Systems

P-130 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
1. DFN-Forum Kommunikations-technologien Beiträge der Fachtagung

P-131 Robert Krimmer, Rüdiger Grimm (Eds.)
3rd International Conference on Electronic Voting 2008
Co-organized by Council of Europe, Gesellschaft für Informatik and E-Voting. CC

P-132 Silke Seehusen, Ulrike Lucke, Stefan Fischer (Hrsg.)
DeLFI 2008:
Die 6. e-Learning Fachtagung Informatik

P-133 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 1

P-134 Heinz-Gerd Hegering, Axel Lehmann, Hans Jürgen Ohlbach, Christian Scheideler (Hrsg.)
INFORMATIK 2008
Beherrschbare Systeme – dank Informatik
Band 2

P-135 Torsten Brinda, Michael Fothe, Peter Hubwieser, Kirsten Schlüter (Hrsg.)
Didaktik der Informatik –
Aktuelle Forschungsergebnisse

P-136 Andreas Beyer, Michael Schroeder (Eds.)
German Conference on Bioinformatics GCB 2008

P-137 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2008: Biometrics and Electronic Signatures

P-138 Barbara Dinter, Robert Winter, Peter Chamoni, Norbert Gronau, Klaus Turowski (Hrsg.)
Synergien durch Integration und Informationslogistik
Proceedings zur DW2008

P-139 Georg Herzwurm, Martin Mikusz (Hrsg.)
Industrialisierung des Software-Managements
Fachtagung des GI-Fachausschusses Management der Anwendungsentwick-lung und -wartung im Fachbereich Wirtschaftsinformatik

P-140 Oliver Göbel, Sandra Frings, Detlef Günther, Jens Nedon, Dirk Schadt (Eds.)
IMF 2008 - IT Incident Management & IT Forensics

P-141 Peter Loos, Markus Nüttgens, Klaus Turowski, Dirk Werth (Hrsg.)
Modellierung betrieblicher Informations-systeme (MobIS 2008)
Modellierung zwischen SOA und Compliance Management

P-142 R. Bill, P. Korduan, L. Theuvsen, M. Morgenstern (Hrsg.)
Anforderungen an die Agrarinformatik durch Globalisierung und Klimaveränderung

P-143 Peter Liggesmeyer, Gregor Engels, Jürgen Münch, Jörg Dörr, Norman Riegel (Hrsg.)
Software Engineering 2009
Fachtagung des GI-Fachbereichs Softwaretechnik

P-144 Johann-Christoph Freytag, Thomas Ruf, Wolfgang Lehner, Gottfried Vossen (Hrsg.)
Datenbanksysteme in Business, Technologie und Web (BTW)

P-145 Knut Hinkelmann, Holger Wache (Eds.)
WM2009: 5th Conference on Professional Knowledge Management

P-146 Markus Bick, Martin Breunig, Hagen Höpfner (Hrsg.)
Mobile und Ubiquitäre Informationssysteme – Entwicklung, Implementierung und Anwendung
4. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2009)

P-147 Witold Abramowicz, Leszek Maciaszek, Ryszard Kowalczyk, Andreas Speck (Eds.)
Business Process, Services Computing and Intelligent Service Management
BPSC 2009 · ISM 2009 · YRW-MBP 2009

P-148 Christian Erfurth, Gerald Eichler, Volkmar Schau (Eds.)
9th International Conference on Innovative Internet Community Systems
I²CS 2009

P-149 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
2. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-150 Jürgen Münch, Peter Liggesmeyer (Hrsg.)
Software Engineering
2009 - Workshopband

P-151 Armin Heinzl, Peter Dadam, Stefan Kirn, Peter Lockemann (Eds.)
PRIMIUM
Process Innovation for Enterprise Software

P-152 Jan Mendling, Stefanie Rinderle-Ma, Werner Esswein (Eds.)
Enterprise Modelling and Information Systems Architectures
Proceedings of the 3rd Int'l Workshop EMISA 2009

P-153 Andreas Schwill, Nicolas Apostolopoulos (Hrsg.)
Lernen im Digitalen Zeitalter
DeLFI 2009 – Die 7. E-Learning Fachtagung Informatik

P-154 Stefan Fischer, Erik Maehle Rüdiger Reischuk (Hrsg.)
INFORMATIK 2009
Im Focus das Leben

P-155 Arslan Brömme, Christoph Busch, Detlef Hühnlein (Eds.)
BIOSIG 2009:
Biometrics and Electronic Signatures
Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-156 Bernhard Koerber (Hrsg.)
Zukunft braucht Herkunft
25 Jahre »INFOS – Informatik und Schule«

P-157 Ivo Grosse, Steffen Neumann, Stefan Posch, Falk Schreiber, Peter Stadler (Eds.)
German Conference on Bioinformatics 2009

P-158 W. Claupein, L. Theuvsen, A. Kämpf, M. Morgenstern (Hrsg.)
Precision Agriculture Reloaded – Informationsgestützte Landwirtschaft

P-159 Gregor Engels, Markus Luckey, Wilhelm Schäfer (Hrsg.)
Software Engineering 2010

P-160 Gregor Engels, Markus Luckey, Alexander Pretschner, Ralf Reussner (Hrsg.)
Software Engineering 2010 – Workshopband
(inkl. Doktorandensymposium)

P-161 Gregor Engels, Dimitris Karagiannis Heinrich C. Mayr (Hrsg.)
Modellierung 2010

P-162 Maria A. Wimmer, Uwe Brinkhoff, Siegfried Kaiser, Dagmar Lück-Schneider, Erich Schweighofer, Andreas Wiebe (Hrsg.)
Vernetzte IT für einen effektiven Staat
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2010

P-163 Markus Bick, Stefan Eulgem, Elgar Fleisch, J. Felix Hampe, Birgitta König-Ries, Franz Lehner, Key Pousttchi, Kai Rannenberg (Hrsg.)
Mobile und Ubiquitäre Informationssysteme
Technologien, Anwendungen und Dienste zur Unterstützung von mobiler Kollaboration

P-164 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2010: Biometrics and Electronic Signatures Proceedings of the Special Interest Group on Biometrics and Electronic Signatures

P-165 Gerald Eichler, Peter Kropf, Ulrike Lechner, Phayung Meesad, Herwig Unger (Eds.)
10th International Conference on Innovative Internet Community Systems (I^2CS) – Jubilee Edition 2010 –

P-166 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
3. DFN-Forum Kommunikationstechnologien
Beiträge der Fachtagung

P-167 Robert Krimmer, Rüdiger Grimm (Eds.)
4th International Conference on Electronic Voting 2010
co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-168 Ira Diethelm, Christina Dörge, Claudia Hildebrandt, Carsten Schulte (Hrsg.)
Didaktik der Informatik
Möglichkeiten empirischer Forschungsmethoden und Perspektiven der Fachdidaktik

P-169 Michael Kerres, Nadine Ojstersek Ulrik Schroeder, Ulrich Hoppe (Hrsg.)
DeLFI 2010 - 8. Tagung der Fachgruppe E-Learning der Gesellschaft für Informatik e.V.

P-170 Felix C. Freiling (Hrsg.)
Sicherheit 2010
Sicherheit, Schutz und Zuverlässigkeit

P-171 Werner Esswein, Klaus Turowski, Martin Juhrisch (Hrsg.)
Modellierung betrieblicher Informationssysteme (MobIS 2010)
Modellgestütztes Management

P-172 Stefan Klink, Agnes Koschmider Marco Mevius, Andreas Oberweis (Hrsg.)
EMISA 2010
Einflussfaktoren auf die Entwicklung flexibler, integrierter Informationssysteme
Beiträge des Workshops der GI-Fachgruppe EMISA (Entwicklungsmethoden für Informationssysteme und deren Anwendung)

P-173 Dietmar Schomburg, Andreas Grote (Eds.)
German Conference on Bioinformatics 2010

P-174 Arslan Brömme, Torsten Eymann, Detlef Hühnlein, Heiko Roßnagel, Paul Schmücker (Hrsg.)
perspeGKtive 2010
Workshop „Innovative und sichere Informationstechnologie für das Gesundheitswesen von morgen“

P-175 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 1

P-176 Klaus-Peter Fähnrich, Bogdan Franczyk (Hrsg.)
INFORMATIK 2010
Service Science – Neue Perspektiven für die Informatik
Band 2

P-177 Witold Abramowicz, Rainer Alt, Klaus-Peter Fähnrich, Bogdan Franczyk, Leszek A. Maciaszek (Eds.)
INFORMATIK 2010
Business Process and Service Science – Proceedings of ISSS and BPSC

P-178 Wolfram Pietsch, Benedikt Krams (Hrsg.)
Vom Projekt zum Produkt
Fachtagung des GI-Fachausschusses Management der Anwendungsentwicklung und -wartung im Fachbereich Wirtschafts-informatik (WI-MAW), Aachen, 2010

P-179 Stefan Gruner, Bernhard Rumpe (Eds.)
FM+AM`2010
Second International Workshop on Formal Methods and Agile Methods

P-180 Theo Härder, Wolfgang Lehner, Bernhard Mitschang, Harald Schöning, Holger Schwarz (Hrsg.)
Datenbanksysteme für Business, Technologie und Web (BTW)
14. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme“ (DBIS)

P-181 Michael Clasen, Otto Schätzel, Brigitte Theuvsen (Hrsg.)
Qualität und Effizienz durch informationsgestützte Landwirtschaft, Fokus: Moderne Weinwirtschaft

P-182 Ronald Maier (Hrsg.)
6th Conference on Professional Knowledge Management
From Knowledge to Action

P-183 Ralf Reussner, Matthias Grund, Andreas Oberweis, Walter Tichy (Hrsg.)
Software Engineering 2011
Fachtagung des GI-Fachbereichs Softwaretechnik

P-184 Ralf Reussner, Alexander Pretschner, Stefan Jähnichen (Hrsg.)
Software Engineering 2011
Workshopband
(inkl. Doktorandensymposium)

P-185 Hagen Höpfner, Günther Specht, Thomas Ritz, Christian Bunse (Hrsg.)
MMS 2011: Mobile und ubiquitäre Informationssysteme Proceedings zur 6. Konferenz Mobile und Ubiquitäre Informationssysteme (MMS 2011)

P-186 Gerald Eichler, Axel Küpper, Volkmar Schau, Hacène Fouchal, Herwig Unger (Eds.)
11th International Conference on Innovative Internet Community Systems (I^2CS)

P-187 Paul Müller, Bernhard Neumair, Gabi Dreo Rodosek (Hrsg.)
4. DFN-Forum Kommunikations-technologien, Beiträge der Fachtagung 20. Juni bis 21. Juni 2011 Bonn

P-188 Holger Rohland, Andrea Kienle, Steffen Friedrich (Hrsg.)
DeLFI 2011 – Die 9. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V. 5.–8. September 2011, Dresden

P-189 Thomas, Marco (Hrsg.)
Informatik in Bildung und Beruf
INFOS 2011
14. GI-Fachtagung Informatik und Schule

P-190 Markus Nüttgens, Oliver Thomas, Barbara Weber (Eds.)
Enterprise Modelling and Information Systems Architectures (EMISA 2011)

P-191 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2011
International Conference of the Biometrics Special Interest Group

P-192 Hans-Ulrich Heiß, Peter Pepper, Holger Schlingloff, Jörg Schneider (Hrsg.)
INFORMATIK 2011
Informatik schafft Communities

P-193 Wolfgang Lehner, Gunther Piller (Hrsg.)
IMDM 2011

P-194 M. Clasen, G. Fröhlich, H. Bernhardt, K. Hildebrand, B. Theuvsen (Hrsg.)
Informationstechnologie für eine nachhaltige Landbewirtschaftung
Fokus Forstwirtschaft

P-195 Neeraj Suri, Michael Waidner (Hrsg.)
Sicherheit 2012
Sicherheit, Schutz und Zuverlässigkeit
Beiträge der 6. Jahrestagung des Fachbereichs Sicherheit der Gesellschaft für Informatik e.V. (GI)

P-196 Arslan Brömme, Christoph Busch (Eds.)
BIOSIG 2012
Proceedings of the 11th International Conference of the Biometrics Special Interest Group

P-197 Jörn von Lucke, Christian P. Geiger, Siegfried Kaiser, Erich Schweighofer, Maria A. Wimmer (Hrsg.)
Auf dem Weg zu einer offenen, smarten und vernetzten Verwaltungskultur
Gemeinsame Fachtagung Verwaltungsinformatik (FTVI) und Fachtagung Rechtsinformatik (FTRI) 2012

P-198 Stefan Jähnichen, Axel Küpper, Sahin Albayrak (Hrsg.)
Software Engineering 2012
Fachtagung des GI-Fachbereichs Softwaretechnik

P-199 Stefan Jähnichen, Bernhard Rumpe, Holger Schlingloff (Hrsg.)
Software Engineering 2012
Workshopband

P-200 Gero Mühl, Jan Richling, Andreas Herkersdorf (Hrsg.)
ARCS 2012 Workshops

P-201 Elmar J. Sinz Andy Schürr (Hrsg.)
Modellierung 2012

P-202 Andrea Back, Markus Bick, Martin Breunig, Key Pousttchi, Frédéric Thiesse (Hrsg.)
MMS 2012:Mobile und Ubiquitäre Informationssysteme

P-203 Paul Müller, Bernhard Neumair, Helmut Reiser, Gabi Dreo Rodosek (Hrsg.)
5. DFN-Forum Kommunikations-technologien
Beiträge der Fachtagung

P-204 Gerald Eichler, Leendert W. M. Wienhofen, Anders Kofod-Petersen, Herwig Unger (Eds.)
12th International Conference on Innovative Internet Community Systems (I2CS 2012)

P-205 Manuel J. Kripp, Melanie Volkamer, Rüdiger Grimm (Eds.)
5th International Conference on Electronic Voting 2012 (EVOTE2012)
Co-organized by the Council of Europe, Gesellschaft für Informatik and E-Voting.CC

P-206 Stefanie Rinderle-Ma, Mathias Weske (Hrsg.)
EMISA 2012
Der Mensch im Zentrum der Modellierung

P-207 Jörg Desel, Jörg M. Haake, Christian Spannagel (Hrsg.)
DeLFI 2012: Die 10. e-Learning Fachtagung Informatik der Gesellschaft für Informatik e.V.
24.–26. September 2012

P-208 Ursula Goltz, Marcus Magnor, Hans-Jürgen Appelrath, Herbert Matthies, Wolf-Tilo Balke, Lars Wolf (Hrsg.) INFORMATIK 2012

P-209 Hans Brandt-Pook, André Fleer, Thorsten Spitta, Malte Wattenberg (Hrsg.) Nachhaltiges Software Management

P-210 Erhard Plödereder, Peter Dencker, Herbert Klenk, Hubert B. Keller, Silke Spitzer (Hrsg.) Automotive – Safety & Security 2012 Sicherheit und Zuverlässigkeit für automobile Informationstechnik

The titles can be purchased at:

Köllen Druck + Verlag GmbH
Ernst-Robert-Curtius-Str. 14 · D-53117 Bonn
Fax: +49 (0)228/9898222
E-Mail: druckverlag@koellen.de