Automatisierte Ermittlung von Umweltwirkungen entlang der Wertschöpfungskette

Funk, Burkhardt; Niemeyer, Peter

Published in:
Facettenreiche Wirtschaftsinformatik

Publication date:
2009

Document Version
Verlags-PDF (auch: Version of Record)

19. Jahrgang, Heft 1, August 2009, ISSN 0939-8821

LEUPHANA
UNIVERSITÄT LÜNEBURG

FinAL
Forum Informatics At Leuphana

Facettenreiche Wirtschaftsinformatik

Controlling, Compiler & more

Festschrift für
Prof. Dr. rer. nat. Karl Goede

01000110 01001001 01101110 01000001 01001100

Information Science: Technical Reports and Working Papers
Hrsg: Hinrich E. G. Bonin
Volgershall 1, D-21339 Lüneburg
Phone: +49.4131.6775175 Fax: +49.4131.6775140

Inhaltsverzeichnis

Prof. Dr. rer. nat. Karl Goede
goede@uni.leuphana.de

Jahrgang: 1944

Ausbildung:

* Abitur	1964 (Gymnasium in Gifhorn)
* Diplom	1971 (Technische Universtiät Braunschweig)
* Promotion	1978 (Technische Universtiät Braunschweig) Dissertation: Ein Prozessbegriff und seine Anwendung in der Definition der Semantik von Programmiersprachen

Berufliche Entwicklung:

* 1971 -1973	wissenschaftlicher Mitarbeiter am Rechenzentrum der TU Braunschweig
* 1973 - 1978	wiss. Assistent am Lehrstuhl für formale Sprachen und Compilerbau (Informatik) der TU Braunschweig Arbeitsschwerpunkte: - Compilertechnologie - Semantik von Programmiersprachen
* 1978 – 1982	Softwarehaus mbp in Dortmund und Entwicklungsbüro Wulf Werum (heute Werum Software & Systems AG) in Lüneburg Arbeitsgebiete: - Umsetzung von Compiler- Betriebssystem-technologien in der Prozeßtechnik - Projektierung und Konstruktion eines Echtzeit-datenhaltungssystems
* 1982 - 1984	Selbständige Tätigkeit als Softwareentwickler (hauptsächlich Mikrorechner und deren Einsatz)
* seit 1984	Professor für „Systemprogrammierung und Rechnersysteme“ im FB W der Fachhochschule Nordostniedersachsen bzw. Universität Lüneburg während dieser Zeit unter anderem tätig: - als Dekan - im Fachbereichsrat - im Praxissemesterausschuss - als Mitglied im Institute of Computer Sciences ICS (heute Institut für Wirtschaftsinformatik IWI) der Leuphana Universität Lüneburg

Hobbies: Golf, Bridge

Compilerbau in der Wirtschaftsinformatik!

— Festschrift für Prof. Dr. Karl Goede —

Vorwort des Herausgebers

"The first Fortran[1] compiler, for example,
took 18 staff-years to implement [Ba57].
..., a substantial compiler can be implemented
even as a student project in a one-semester compiler-design course."
[ASU86, p. 2.]

Compilerbau — Verstehen von „Computerarbeit“

Der Compilerbau, deutsch Übersetzerbau (z. B., [ALSU08, Wi96]), ist unstrittig eine Kernaufgabe der *Praktischen Informatik,* wenn man die Gesamtdisziplin Informatik klassisch gliedert und zwar in *Theoretische Informatik, Technische Informatik, Praktische Informatik* und *Angewandte Informatik(en);* Kategorien, die die Gesellschaft für Informatik e.V, Bonn (GI[2]) gepflegt hat. Dabei werden die *Wirtschaftsinformatik (WI)* oder beispielsweise auch die *Informatik in Recht und Öffentlicher Verwaltung* den Angewandten Informatiken zugerechnet.

Was hat dann der Entwurf und die Programmierung von Compilern in der WI zu suchen? Ein Compiler setzt einen Quellcode, in einer höheren Programmiersprache verfassten Programmtext, in einen Zielcode, d. h. vorrangig in Maschinensprache, um. Üblicherweise gehören Algorithmen, Datenstrukturen, Programmiersprachen, Betriebssysteme, Datenbanken - und daher auch Compiler - zur *Praktischen Informatik.* Unzweifelhaft, der Wissenschaftler der Wl-Community lötet nicht. Dies überlässt er den Kollegen in der *Technischen Informatik.* Warum sollte er sich dann mit der Praktischen Informatik befassen, beispielsweise mit den klassischen Werkzeugen Yacc *(YetAnotherCompilerCompiler ≡ Parsergenerator)* oder *Lex*[3]*?*

Als ich dem Kollegen *Karl Goede* zum ersten Mal begegnete (≈ 1989), dachte ich wie so viele Informatiker in genau diesen klassischen GI-Kategorien. Ich war daher erstaunt, dass im Studiengang WI der damaligen *Fachhochschule Nordostniedersachsen* (FHNON) ein so versierter Compilerbauer lehrte. Lehrkapazität war und ist stets ein knappes Gut. Was hat also Compilerbau in der WI zu suchen?

[1] FORTRAN Formula Translation) ist eine der ersten höheren Programmiersprachen, die ≈ 1953 von John W. Backus spezifiziert wurde. Fortran 2003 ist der derzeitige Standard (IS0/IEC 1539-2004).
[2] GI e. V, Bonn, gegründet 1969, derzeit ≈ 24.500 Mitglieder.
http://www.gi-ev.de/wir-ueber-uns.html (online 19-Mar-2009)
[3] Yacc dient der syntaktischen und *Lex* der lexikalischen Analyse eines Textes (z. B. [He03]).

Die allgemein in der WI-Lehre angebotene Pflichtveranstaltung „Anwendungsentwicklung in ...[4]“ fand zu jener Zeit, wie überall im (Fach-)Hochschulbereich, praxisorientiert in COBOL[5] statt. Erstaunlicherweise konnten die FHNON-Studierenden sehr fachkundig über komplexe Programmstrukturen im betriebswirtschaftlichen Anwendungsbereich diskutieren. Beispielsweise hatten sie über Zuweisungen, Iterationen oder Rekursionen präzisere Vorstellungen als ihre Kommilitonen in der Hochschule Bremerhaven, der ich vorher angehörte. Offensichtlich legte *Karl Goedes* Lehre ein tragfähiges Fundament zum Verstehen von „Computerarbeit“[6]. Nicht die Befassung mit Bits und Bytes oder mit Registern und Addierwerken, sondern die konsequente Zerlegung einer komplexen Anweisung in konkrete Abarbeitungsschritte, die ein Computer vollziehen kann, trainiert effektiv für die Gestaltung von nützlichen Systemen auch bei betriebswirtschaftlichen Aufgaben. Die Transformation eines *Source Program* in ein *Target Program* schafft Verständnis für die Transformation eines Projektauftrages in ein produktives Softwaresystem. Es trainiert, die „Lyrik von Auftragstexten“ in unmissverständliche Funktionsspezifikationen zu übertragen.

Klar ist, dass Compilerbau nicht eine Kernaufgabe der WI ist. Klar ist aber auch, dass ohne Compilerbau-Kenntnisse ein hinreichendes WI-Verständnis für „Computerarbeit“ nicht effektiv erzielbar ist. Kurz gesagt: eine bildungs-orientierte Lehre der WI braucht auch kompetente Compilerbauer, wie den Kollegen *Karl Goede*[7]. Diese Forderung ist sicherlich dann unstrittig, wenn es um die Frage der Wahl der problemadäquaten Programmiersprache geht [Go08].

Controlling - Realisieren von Computernutzen

Eine völlig andere Facette der WI ist sicherlich ihre Kernaufgabe Controllingsysteme zu realisieren. Beim Controlling (z.B. [Ho08]) geht es um die Durchdringung „*alle(r) Führungsphasen von der Willensbildung, der Willensdurchsetzung bis zur Ergebniskontrolle von Führungshandlungen. Controllingsysteme beinhalten als Teilmodule* (Beitrag von Sven Piechota in diesem Band):

- *Zielsetzungssysteme*
- *Planungssysteme*
- *Entscheidungssysteme,*
- *Dokumentationssysteme,*
- *Systeme der Erfassung und Ermittlung führungsrelevanter Daten,*
- *Bewertungssysteme, die rationale Vergleiche ermöglichen,*
- *Methodensysteme, die rationale betriebswirtschaftliche Diagnostik ermöglichen.“*

---

[4] Zum Beispiel „Anwendungsentwicklung in der öffentlichen Verwaltung“

[5] COBOL Common Business Oriented Language) ist in der Wirtschaft und Verwaltung seit vielen Jahren weit verbreitet — aktueller Standard 2002 (ISO/IEC 1989:2002).

[6] Nicht thematisiert soll hier die kritische Frage werden, ob Computer überhaupt im engeren Sinne Arbeit verrichten. Vielmehr geht es hier beim Begriff „Computerarbeit“ um die Übertragung und Durchführung von Arbeit, die mit Menschen verknüpft ist, auf Computer (universelle Automaten).

[7] Keinesfalls soll hier das Schaffen von Karl Goede auf die Thematik Compilerbau verkürzt werden. Karl Goede hat sich mit vielfältigen WI-Themen befasst; siehe zum Beispiel [Go94].

Auf den ersten Blick stellt sich damit die Frage, was dabei der originäre Beitrag der WI-involvierten Informatik ist. Solche Managementaspekte weisen doch eine dominante Orientierung zur Wirtschaftswissenschaft auf, oder plakativ formuliert: sie fordern primär die W-Seite der WI heraus. Konsequenterweise wird dann die ganze WI-Disziplin zu einem Teilgebiet der Wirtschaftswissenschaften deklariert.[8]

Auf den zweiten Blick jedoch wird deutlich, dass die Erfolge des Controlling ohne Computer nicht erzielbar sind. Erst eine allumfassende Nutzung von Computern schafft die Voraussetzungen für das Controlling, insbesondere wenn man die hohe Komplexität der heutigen Wertschöpfungsketten unterstellt. Damit stets das Positive beim Computereinsatz im Bereich Controlling überwiegt, bedarf es bei den Akteuren eines fundierten Verständnisses über die Art und Weise wie eine Systemspezifikation in einen formalen Text, der vom Computer abarbeitungsfähig ist, transformiert wird. Dies gilt beispielsweise für ein Entscheidungssystem genauso wie für ein Dokumentationssystem.

So gesehen gilt es, die Fachsprache der Controlling Community in ein *Target Program* zu transformieren, quasi zu compilieren. Also profitiert die WI-Kernaufgabe Controlling von der Perspektive des Compilerbauers.

Da die WI die skizzierte Verknüpfung zur Praktischen Informatik braucht, mag es kaum angebracht sein, die WI zu einem Teilgebiet der Wirtschaftswissenschaften zu erklären. Selbst wenn die Leuphana Universität Lüneburg die WI-Kernaufgabe Controlling zukünftig massiv betonen und wissenschaftlich in den Mittelpunkt stellen wollte, wäre ein Compilerbauer vom Format eines *Karl Goede* geboten.

Die Beiträge in diesem Band beleuchten sehr unterschiedliche Facetten der WI. Controlling, Fragen der Qualitätssicherung stehen ebenso zur Debatte wie Methoden zur Maschinisierung von Kopfarbeit. Wir veröffentlichen die Beiträge zu den unterschiedlichen Aspekten der WI, um das Wirken von *Karl Goede zu* würdigen. Ohne ihn, unseren Promotor der Praktischen Informatik in der WI, wäre zumindest das Institut für Wirtschaftsinformatik (IWI[9]) der Fakultät III der Leuphana Universität Lüneburg, nicht in dieser Form und mit diesem WI-Profil entstanden. Im Goede'schen Sinne mögen die Beiträge zu neuen Einsichten und Aktivitäten für Lehre, Forschung und Technologietransfer anregen.

Lüneburg, im August 2009
Hinrich E. G. Bonin

[8] So wird bei der Neugliederung der Leuphana Universität Lüneburg die WI im Jahre 2009 von der (ehemaligen) Fakultät III mit ihrem Schwerpunkt Technik (≈ Ingenieurwesen) zur Fakultät II mit ihrer angestrebten Ausrichtung auf Wirtschaftswissenschaften verlegt [PrL09].
[9] http://www.leuphana.de/iwi (online 3-Aug-2009) — vormals ICS (*Institute of Cumpter Sciences*)

Literaturverzeichnis

[ALSU08] Alfred V Aho / Monica S. Lam / Ravi Sethi / Jeffery D. Ullman; Compiler. Prinzipien, Techniken und Tools, (Pearson Studium) 2. Auflage Januar 2008, ISBN 978-3-8273-7097-6 [Das „Drachenbuch" ist ein unumstrittenes Referenzbuch des Compilerbaus; aktuelle Fassung von [ASU86].]

[ASU86] Alfred V Aho / Ravi Sethi / Jeffery D. Ullman; Compilers. Principles, Techniques, and Tools, (Addison-Wesley Publishing Company) 1986, ISBN 0-201-10194-7.

[Ba57] J. W Backus u. a. ; The Fortran automatic coding system, in: Western Joint Computer Conference, 1957, pp. 188-198. Reprinted in Rosen, 1967, pp. 29-47. (zitiert nach [ASU86])

[Go08] Karl Goede; Ist D ein ernstzunehmender Nachfolger für C/C++?, in: FINAL *(Forum Informatics at Leuphana),* 18. Jahrgang, Heft 1, Juni 2008, ISSN 0939-8821, S. 35-59.

[Go94] Karl Goede; EDV-gestützte Kommunikation und Hochschulorganisation, in: FINAL *(Forum Informatics at Leuphana),* 4. Jahrgang, Heft 6, 1994, ISSN 0939-8821.

[He03] Herold, Helmut; lex & yacc. Die Profitools zur lexikalischen und syntaktischen Textanalyse, (Addison Wesley) März 2003, ISBN 3-82732-096-8.

[Ho08] Péter Horváth; Controlling, (Verlag Vahlen) 11., vollständig überarbeitete Auflage (November 2008), ISBN 3-827-32096-8.

[PrL09] Präsidium der Leuphana (Sascha Spoun); Künftige Binnenorganisation der Leuphana Universität Lüneburg; Vorlage zur 43. Senatssitzung, Drucksache-Nr.: 159/43/4 SoSe 2009; ergänzte Fassung gegenüber dem Stand vom 17. Juni 2009 (Drs. Nr. 155/42/3 SoSe 2009).

[Wi96] Niklaus Wirth; Grundlagen und Techniken des Compilerbaus, (Oldenbourg Wissenschaftsverlag) 1996, ISBN 3486-24374-8, 2. bearbeitete Auflage 2008 ISBN 3-486-58581-9. [Anhand einer selbst definierten Sprache entwickelt der Autor einen vollständigen Compiler.]

Grußwort des Dekans

Mit dem Ausscheiden von Herrn Prof. Dr. rer. nat. Karl Goede aus dem aktiven Dienst geht nun ein weiterer Vertreter der Wirtschaftsinformatik der ersten Stunde an unserer Universität in den verdienten Ruhestand.

Herr Goede kam 1984 an die damalige Fachhochschule Nordostniedersachsen in den Fachbereich Wirtschaft, um in dem noch jungen Studiengang Wirtschaftsinformatik, den es seit dem Wintersemester 1981/82 gab, zu lehren und ihn weiter zu entwickeln. Er vertrat bis heute die Lehrgebiete Systemprogrammierung und Rechnernetze, später auch das Fach Datenbanken. Herr Goede hat sich mit großem Engagement um die inhaltliche Ausrichtung der Wirtschaftsinformatik an seinem Fachbereich erfolgreich eingesetzt. So konnte er stets überzeugen, dass sich die Wirtschaftsinformatik nicht nur durch interdisziplinäre Lehrinhalte zwischen Informatik und Betriebswirtschaftslehre definiert, sondern dass es angehenden Wirtschaftsinformatiker/innen durchaus dienlich ist, auch fundierte Kenntnisse der praktischen Informatik zu erwerben. Diese hat er mit großem didaktischem Geschick vermittelt. Studierende und Kollegen schätzten das Fachwissen und die freundliche und verbindliche Art im persönlichen Umgang. Auf der anderen Seite hat Herr Goede sehr erfolgreich Projekte mit Kollegen der Betriebswirtschaftslehre durchgeführt und sich so als Wirtschaftsinformatiker im besten Sinne gezeigt. Hierbei waren ihm sicherlich sein fundiertes Fachwissen, sein Praxisbezug und seine Erfahrungen aus seiner beruflichen Tätigkeit außerhalb der Hochschule von Nutzen.

Darüber hinaus hat Herr Goede seine Erfahrungen und Fähigkeiten auch in die akademische Selbstverwaltung als Dekan, Fachbereichsrats- und Senatsmitglied und in unterschiedlichen Gremien zum Nutzen des Fachbereichs und der Hochschule eingebracht.

In der Zeit von Herrn Goede genoss die Wirtschaftsinformatik seines Fachbereichs mit ihren verschiedenen Studiengängen (Diplom-, Bachelor- und Masterabschlüsse, letztere erfolgreich akkreditiert) über die Hochschule hinaus einen hervorragenden Ruf. Herr Goede hat zu dieser Stellung aktiv und konstruktiv beigetragen. Die Wirtschaftsinformatik an der Leuphana Universität Lüneburg verliert mit dem Ausscheiden von Herrn Goede einen wichtigen Vertreter, zumal seine Stelle leider nicht wiederbesetzt wird.

Als Dekan der Fakultät Umwelt und Technik der Leuphana Universität Lüneburg und als Kollege bedanke ich mich für die langjährige erfolgreiche und erfreuliche Zusammenarbeit und wünsche Ihnen, lieber Herr Goede, weitere glückliche Jahre in Ihrem nun beginnenden neuen Lebensabschnitt.

Wissenschaft trifft Wirtschaft.

Förderverein Netzwerk Wirtschaft
der Leuphana Universität Lüneburg e.V.

Lieber Herr Professor Goede,

es ist mir eine besondere Freude, mich als Vorsitzender des Fördervereins an diesem besonderen Wendepunkt Ihres Lebens an Sie zu wenden.
Sie sind dem Förderverein praktisch gleich, nachdem Sie Ihre Professorenstelle in Lüneburg angetreten hatten, beigetreten und haben die Arbeit des Vereins von Anfang an unterstützt.

Sie hatten immer ein offenes Ohr für die Belange der Studierenden, das hat sich besonders zur Zeit Ihres Dekanats gezeigt. Als Verfechter einer wissenschaftlich fundierten und gleichzeitig auch praxisbezogenen Lehre haben Sie sich stets für einen gedeihlichen Austausch zwischen Wissenschaft und Wirtschaft eingesetzt und damit die Arbeit des Fördervereins wesentlich mitgetragen und auch voran gebracht. Und im Namen all derer, denen Sie durch Ihr Wirken den gewissen, kleinen Schubs und Vorteil beim Start in das Berufsleben verschafft haben, möchte ich Ihnen ganz herzlich danken.

Lieber Herr Professor Goede, im Namen des Vorstands und der Mitglieder unseres Vereins wünsche Ihnen von Herzen alles Gute für die nun vor Ihnen liegende nächste „Projektphase“.
Ich würde mich sehr freuen, wenn wir in Kontakt blieben und weiterhin auf Ihre Erfahrungen und Unterstützung zählen dürften.

Ihr

Stefan Manzke
Dipl. Wirtschaftsinformatiker (FH)
1. Vorsitzender
des Fördervereins Netzwerk Wirtschaft
der Leuphana Universität Lüneburg e.V.

Informatikmethoden zur Maschinisierung von Kopfarbeit[1]

Hinrich E. G. Bonin[2]

"Computer Science is no more about computers
than astronomy is about telescopes"
Edsger Wybe Dijkstra[3]

Abstract:

Primär basiert die *Maschinisierung der Kopfarbeit* auf Methoden aus den Bereichen der Konstruktions- und der Verhaltenswissenschaften. Hier sind Methoden zur Modellierung von Informatik-Produkten aus der Perspektive der Konstruktionswissenschaften skizziert. Die Modelle sind geprägt durch die beiden Informatik-Hauptwurzeln *Mathematik* und *Elektrotechnik*.

1 Maschinisierung von Kopfarbeit

1.1 Was ist Informatik?

Die Frage nach dem Selbstverständnis der Disziplin *Informatik* (engl.: *Computer Science(s), informatics, information technology*) beantwortet die größte Informatikfachvertretung im deutschsprachigen Raum, die Gesellschaft für Informatik e. V. (GI[4]), wie folgt ([GI2005] S. 4): *Informatik — ist die Faszination, sich die Welt der Information und des symbolisierten Wissens zu erschließen und dienstbar zu machen.* **Informatik schafft neue Zugänge, neue Denkmodelle**[5] *und zahllose automatisierte Helfer und Dienste. Informatik ermöglicht multimediale Kommunikation überall, zu jeder Zeit und sofort. Informatik überwacht, steuert und vernetzt Prozesse.*
Beginnend mit dem Bau und der Programmierung reiner ≫Rechenmaschinen≪ hat sich die Informatik rasch weiter Arbeitsbereiche in Produktion, Organisation und Verwaltung angenommen. Inzwischen macht sie den Computer nicht mehr nur zur Arbeitsmaschine, sondern auch zum Medium, Wissensträger, Manager, Unterhaltungskünstler und Steuerungsinstrument, ja sogar zu einer Art neuen Wahrnehmungsorgans für die meisten Wissenschaften".

---

[1] Die Formulierung *Maschinisierung von Kopfarbeit* stammt von Frieder Nake [Na 92].
[2] Hinrich Bonin, Leuphana Universität Lüneburg, mailto:bonin@uni.leuphana.de
[3] Informatiker *Edsger Wybe Dijkstra* (∗ 11.05.1930 in Rotterdam, † 06.08.2002 in Nuenen, Niederlande) war der Promotor der *strukturierten Programmierung → Go To Statement Considered Harmful*, in: Communications of the *Association for Computing Machinery* (ACM) 11, 3 (1968), S. 147--148.
[4] GI gegründet 1969, derzeit ≈ 24.500 Mitglieder.
→ http://www.gi-ev.de/wir-ueber-uns.html (online 19-Mar-2009)
[5] Hervorhebung von Bonin.

Etwas weniger euphorisch formuliert die erste wissenschaftliche Gesellschaft für Informatik, die *Association for Computing Machinery* (ACM[6]) ihr Ziel: *„Advancing Computing as a Science & Profession"*. Üblicherweise[7] ist Informatik die Wissenschaft von der systematischen Verarbeitung von Informationen. Im Mittelpunkt steht die automatische Verarbeitung durch Computer. Hervorgegangen ist die Informatik als Wissenschaft aus der Mathematik in Verbindung mit der Elektrotechnik. Für die Informatik sind Computer nur ein Werkzeug und/oder ein Medium zur Realisierung ihrer theoretischen Modelle und Konzepte.

Was beschreibt Informatik in der Vergangenheit, in der Gegenwart und in der Zukunft? Klar ist, dass das Selbstverständnis einer Disziplin grundlegend für die Theoriebildung ist und sich in seiner Begriffsklärung manifestiert. Klar ist aber auch, dass es nicht darum gehen kann, feinste Feinheiten der Begriffsdefinitionen auszudifferenzieren, sondern darum, die konkreten Probleme der Praxis in den Fokus zu nehmen. Statt eine formale Definition der Disziplin Informatik zu liefern, wird hier nur angenommen: Informatik sei eine Wissenschaft, die sich primär mit Informationen im Kontext von Maschinen befasst, oder anders formuliert, geht es um die *Maschinisierung von Kopfarbeit* ([Na92]).

1.2 Was ist Information?

Damit stellt sich die Frage nach der Klärung des Begriffs Information. Im Lateinischen heißt *informare*: bilden, eine Form geben, und *Information* bedeutet: Bildung, Belehrung.[8] Und es ist bemerkenswert, dass das Zeitalter der Information sich letztlich immer noch nicht über den Begriff *Information* einigen kann.[9] Je nach Wissenschaftsgebiet finden man unterschiedliche Definitionen.

Im Allgemeinen[10] geht man von zwei Aspekten aus:

1. Information ist gedeutete (interpretierte) Nachricht oder Mitteilung. Sie entsteht nach einer Interaktion von Sender und Empfänger letztlich erst beim Empfänger der übermittelten Nachricht.
2. Information an sich, als reale Erscheinung, die für sich selbst existiert, die transportiert und womöglich gemessen, in Teile zerlegt oder aus Teilen zusammengesetzt werden kann, gibt es nicht. Der Begriff Information ist vielmehr eine Verdinglichung des Informierens (*Hypostasierung*, wie die Philosophen sagen), so wie viele abstrakte Substantive Hypostasierung sind, zum Beispiel „Sein", „Nichts", „Identität", „Denken".

Praxisrelevant ist jedoch eine Unterscheidung von syntaktischer und semantischer Information[11]:

- Syntaktische Information
  - ist ein Maß für die kürzeste Codierung der Nachricht
  - ist quantifizierbar

[6] ACM gegründet 1947, derzeit ≈ 92.000 Mitglieder in ≈ 100 Ländern.
→ http://www.acm.org/ (online 19-Mar-2009).
[7] Z. B. → http://de.wikipedia.org/wiki/Informatik (online 19-Mar-2009)
[8] Beim Begriff „Information" ist das Diskussionsfeld sehr weit und reicht tief in die Vergangenheit zurück. So wird beispielsweise Moses von lateinischen Autoren *informator populi* genannt und bei Thomas von Aquin heißt Bildung „informatio". ([Kl03] S. 268)
[9] Z. B. [Kl03] S. 267.
[10] Z. B. [Rech04] S. 92.
[11] Z. B. [Rech03] S. 321.

- braucht keinen Empfänger, steckt objektiv in der Nachricht
- ist bestimmt durch Alphabet und Auftrittswahrscheinlichkeit seiner Zeichen

• Semantische Information
- bezeichnet die Bedeutung einer Nachricht für den Empfänger
- ist nicht quantifizierbar
- braucht einen Empfänger und entsteht erst bei ihm
- ist bestimmt durch die Bedeutung für den Empfänger

Mit diesem weit gefassten Begriff *Information* rücken nicht nur die Methoden für eine Abbildung der „Kopfarbeit“ auf Computern in den Fokus, sondern auch die Methoden zur Optimierung der „Nutzung“ in den Köpfen der Betroffenen und Beteiligten.

1.3 Forschungsmethoden der Informatik

Daher entstammen die Forschungsmethoden der Informatik nicht nur aus der Konstruktionswissenschaft, sondern ebenfalls der Verhaltenswissenschaften. Tabelle 1 möge dies verdeutlichen.

Wissenschaftstyp	**Prozess** ⟶	**Realitätsabbildung** ⟶	**Analyse**
Kostruktions-wissenschaft	Vorgehens-modell ⇝	Modellierungsregeln ⇝	Basis z. B. Logik(Mathematik)
Verhaltens-wissenschaft	Forschungs-Design ⇝	Operationalisierung Erhebungstechniken ⇝	Basis z. B. Statistik

Legende:

⟶ folgt

⇝ ≡ führt primär zu

Text ≡ Schwerpunkt der Betrachtung

Idee/Quelle ähnlich → [WiHe06] S. 8.

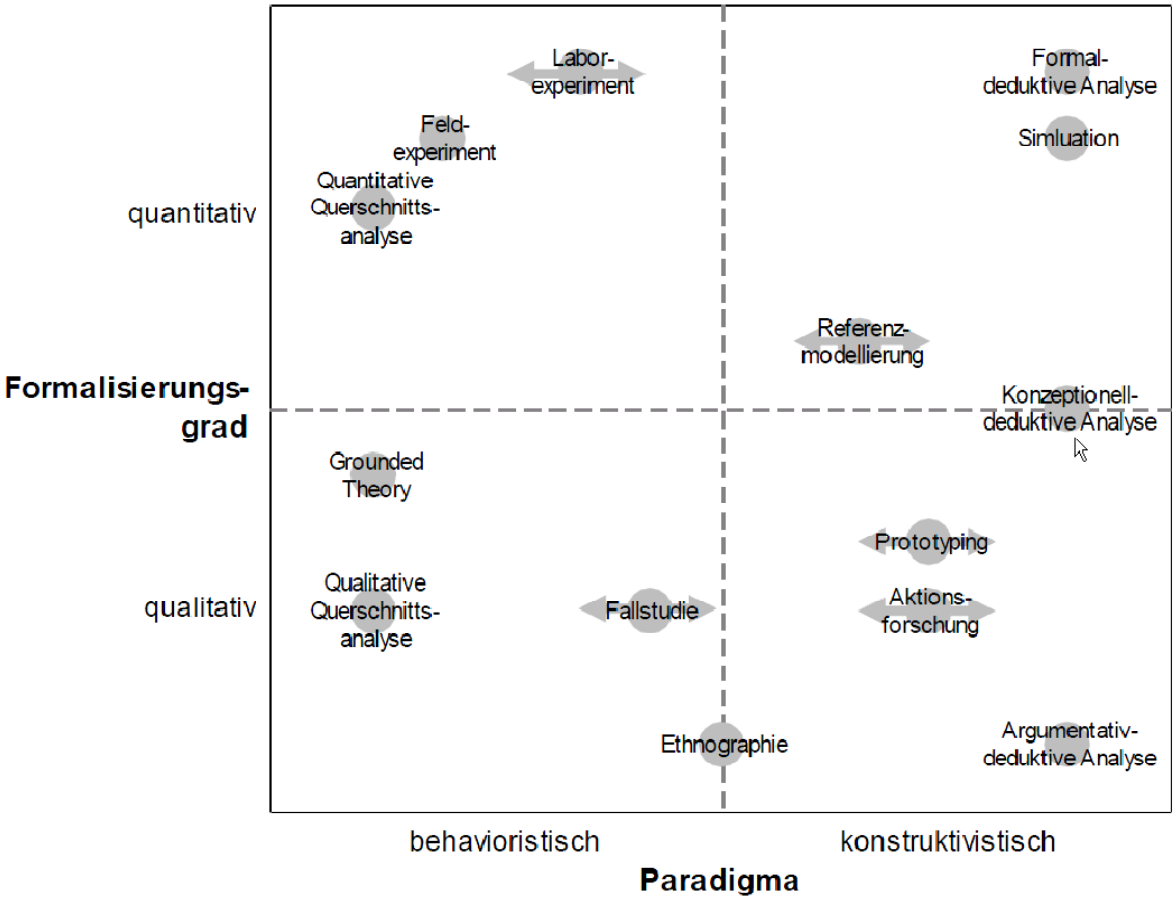
Tabelle 1: Forschungsmethoden der Informatik — Fokus Wirtschaftsinformatik

Unstreitig standen in den Anfängen der Informatik die Realisierungs- und Validierungsprobleme (Stichworte z. B. korrekte Ergebnisse[12] und unterbrechungsfreie Lauffähigkeit) der IT-Systeme im Mittelpunkt. Mit wachsender Leistungsfähigkeit der Computer, ihrer Ubiquität und

[12] Korrekte Berechnung der Fakultätsfunktion von $n = 700$???? → Abbildung 2

Benutzermassen (Stichwort z.B. Web 2.0) kommen zu den Methoden der Konstruktionswissenschaften zunehmend die Methoden der Verhaltenswissenschaften.[13]

Im Folgenden liegt der Schwerpunkt auf Methoden, die dem konstruktivistischen Paradigma[14] zuzuordnen sind (Abbildung 1).



Legende:
Quelle → [WiHe06] S. 16 und [WiHe07] S. 284.
Methodenbeispiel *Prototyping* ≡ konstruktivistisches Paradigma bei qualitativem Formalisierungsgrad (Abschnitt 2.2)

Abbildung 1: Einordnung von Methoden

[13] Zum Beispiel beeinflusste das Web-Angebot der SPD zur Europawahl (7. Juni 2009) statistisch (vermutete?) Forschungsergebnisse über potentielle Nachfrager das IT-System.

[14] Ein Paradigma ist ein von der wissenschaftlichen Fachwelt als Grundlage der weiteren Arbeiten anerkanntes Erklärungsmodell, eine forschungsleitende Theorie. Es entsteht, weil es bei der Lösung von als dringlich erkannten Problemen erfolgreicher ist (zu sein scheint) als andere, bisher geltende Ansätze (Kritik am klassischen Paradigma der Softwareentwicklung [Bonin88].

1.4 Konstruktionswissenschaften: *Informatik* ≈ *Mathematik* ∧ *Elektrotechnik*

Im konstruktivistischen Paradigma basiert die Disziplin *Informatik* primär auf Erkenntnissen der *Mathematik*[15] und der *Elektrotechnik*[16]. Die *Informatik* benutzt daher die zielführenden Methoden ihrer Wurzel-Disziplinen. Wobei die Wurzel Elektrotechnik auf die enge Verwandtschaft mit den Ingenieurwissenschaften[17] verweist.

Üblicherweise assoziiert der Begriff *Methode*[18] eine Denkwelt für ein planmäßiges, zielorientiertes Verfahren, Vorgehen, Handeln und Forschen. Obwohl der Begriff Methode in der Informatik vielfältig[19] genutzt wird, wie auch diese Ausführungen zeigen, stellt sich die „ketzerische" Kernfrage, ob die Informatik überhaupt eigenständige Methoden hat. Oder ergeben sich die feststellbaren Detailabweichungen zu den Methoden ihrer Wurzel-Disziplinen nur aufgrund einer Art von „Schutzterminologie" der Informatiker?

2 *Methoden* im Kontext der Konstruktionswissenschaften

„Das passiert einem
bei der Arbeit an Maschinen immer wieder,
dass man einfach nicht mehr weiter weiß.
Man sitzt da und starrt vor sich hin und denkt nach,
sucht planlos nach neuen Anhaltspunkten,
geht weg und kommt wieder,
und nach einer gewissen Zeit
beginnen sich bisher unbemerkte Faktoren abzuzeichnen."
[Pir1974, S. 61]

Im Informatik-Kernbereich der Softwarekonstruktion ist der Begriff *Methode* relativ aussagekräftig und eindeutig, allerdings abhängig vom eigentlichen Paradigma, das die Softwarekonstruktion prägt. Geht es um die „Denkwelt":

- *Programming-in-the-Large* oder um
- *Programmin-in-the-small*[20].

Neben der „Größe der Konstruktionsaufgabe" (Tabelle 3) hat die „Problemart" (Tabelle 2) einen wesentlichen Einfluss auf die Arbeitstechniken. Arbeitstechniken umfassen einerseits Prinzipien (Handlungsgrundsätze) und Methoden (≡ erfolgversprechende, begründbare Vorgehensweisen) und andererseits Tools (Produktionshilfen → Abschnitt 2.3).

[15] Z. B. Themen: Algorithmen und Berechenbarkeit
[16] Z. B. Themen: Integrierte Bausteine und Hochfrequenz bei der Kommunikation
[17] Beispielsweise bezeichnet sich einer der Gründungsväter der Informatik, der Erfinder des Computers, *Konrad Ernst Otto Zuse* (* 22.06.1910, † 18.12.1995), stets als Ingenieur und nicht als Informatiker.
[18] méthodos ≈ ein Weg, ein Gang, das Verfolgen, die Wegebnung
[19] → Abschnitt 2.4
[20] Z. B. [DeKr76]

Zu Projektbeginn Klarheit über:				
die Arbeitstechniken			die Ziele	
			groß I	gering II
1	Erfolgversprechende Prinzipien, Methoden und Instrumente zur Problemlösung	bekannt	Vollzugsprobleme	Definitionsprobleme der Automationsaufgabe
2		unklar	Zielerreichungs-probleme	Steuerungsprobleme eines kontinuierlichen Herantastens

Tabelle 2: Problemarten

Ist die Konstruktionsaufgabe ein Vollzugsproblem (Feld I.1 in Tabelle 2), dann sind die Methoden charakterisiert durch:

- eine schrittweise Verfeinerung (vertikale Ebene) und
- eine modulare Strukturierung (horizontale Ebene)

von Anforderungen.

Die Produktgröße (≈ Komplexität) des Systems bestimmt, welche Vorgehensweise mit welchen *Tools* anzuwenden ist. Bei einer kleinen Konstruktionsaufgabe ist die Spezifikation der einzelnen Verarbeitungsprozesse und der Datenrepräsentation zu meistern. Bei einer sehr großen Aufgabe sind zusätzlich die Fortschritte der systemnahen Software und der Hardware während der benötigten Planungs- und Realisierungszeit einzukalkulieren. Außerdem ändern sich Anforderungen in dieser relativ langen Zeit. Zu spezifizieren ist daher eine Weiterentwicklung, pointiert formuliert: ein Evolutionskonzept.

Tabelle 3 skizziert benötigte Arbeitstechniken in Abhängigkeit zur Konstruktionsgröße. Dort ist der Umfang des geschätzten Quellcodetextes nur ein grober Maßstab. Die Angabe des Aufwandes in „Mannjahren“ (kurz: MJ)[21] ist umstritten und berechtigt kritisierbar (*The Mythical Man-Month* [Bro1975]). Hier dienen die LOC-[22] und MJ-Werte nur zur groben Unterscheidung, ob ein kleines oder großes Team die Aufgabe bewältigen kann.

[21] Im Zeitalter der Gleichberechtigung wäre es angemessen einen anderen Begriff zu verwenden -- zum Beispiel „Personenjahr“. Ein solcher Begriff hat jedoch bisher noch keinen Eingang in die Fachliteratur gefunden.
[22] Auch als ELOC ≡ *Executable Lines of Code* bezeichnet (z. B. [LudLich2007]).

Lfd	Konstruktionskategorie	Grösse [LOC]	Aufwand [MJ]	Bedarf an Mitteln (Prinzipien, Methoden, *Tools*) zur:
	A	B	C	D
1	Kleine Konstruktion	$< 10^3$	< 0.5	o funktionalen Strukturierung o Datenrepäsentation
2	Mittlere Konstruktion	$< 10^4$	< 4	o Projektplanung o Projektüberwachung für den gesamten Lebenszyklus o Anforderungsanalyse (Requirements Engineering) o plus (1)
3	Große Konstruktion	$< 10^5$	< 25	o Durchführbarkeitsstudie o Definition von Datennetzen und Datenbankmanagement o Konfigurationsmanagement o plus (2)
4	Sehr große Konstruktion (noch grössere „zerfallen“ in eigenständige Teile)	mehrmals $< 10^5$	> 25	o Softwareevolution o Hardwareevolution o Dynamik der Anforderungen o plus (3)

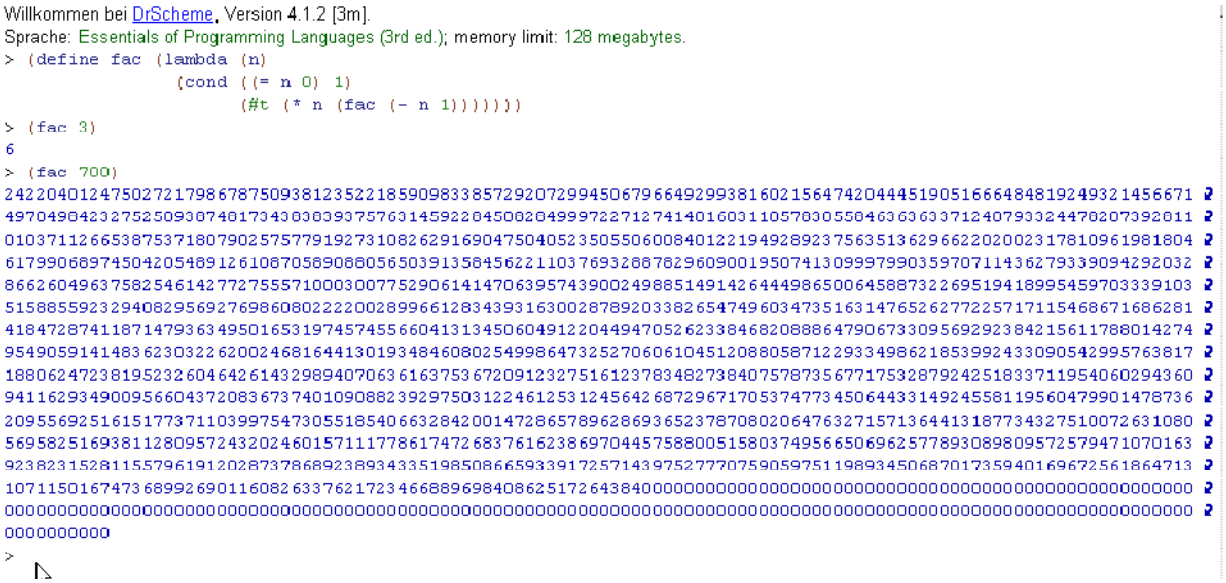
Legende:

LOC ≡ Umfang der Quellcodetexte (Lines of Code)
MJ ≡ Mannjahre

Tabelle 3: Größenkategorien & Arbeitstechniken

Aufgrund der hervorgehobenen Bedeutung ist im Folgenden das Problem der *Validität* (→ Abschnitt 2.1) thematisiert. Aus den vielfältigen Methoden ist exemplarisch das *Prototyping* (→ Abschnitt 2.2) skizziert (→ Abbildung 1 rechtes untere Quadart). Zusätzlich wird auf die Implementierung von Methoden in *Tools* eingegangen. (→ Abschnitt 2.3)

2.1 Fokus: Validität



Legende:
Big numbers ≡ Symbole im *List Processing*-Derivate *Algorithmic Language Scheme*
Scheme-Sprachstandard-Definition vom 26-Sep-2007 (Revised [6]Report):
http://www.r6rs.org/final/r6rs. (online 29-May-2009)

Abbildung 2: Validität von Ergebnissen --- Beispiel Fakultätsfunktion von $n = 700$

Im Kontext der Softwarekonstruktion spielt die Korrektheit (Gültigkeit) der erzielten Ergebnisse (Berechnungen) eine dominante Rolle. Die strikte Einhaltung von bewährten Methoden dient der Sicherung der Validität (Gültigkeit der Ergebnisse). Stellt man beispielsweise die Frage, ob die obige Berechnung der Fakultätsfunktion von $n = 700$ (Abbildung 2) korrekt ist, dann kann man einerseits versuchen, die einzelnen Konstruktionsschritte auf Korrektheit zu überprüfen und sich dann auf das Endergebnis verlassen, oder man versucht mit einer weiteren, möglichst andersartigen Softwarekonstruktion „nachzurechnen“. Kommt man auf das selbe Ergebnis, dann schließt man auf Validität. Es könnte allerdings sein, dass beide Konstruktionen unter derselben „Denkbeule“ (oder auch „Denkfalle“ [23]) erstellt wurden und damit doch ein objektiv falsches Ergebnis ausweisen. Wenn Softwarekonstrukteure (Programmierer) --- wie jeder Mensch --- den gleichen optischen Täuschungen unterliegen, warum sollte man nicht auch von gleichen „Denkbeulen“ ausgehen?

[23] Zum Thema „Denkfalle“ siehe zum Beispiel die Publikationen von *Timm Grams* http://www2.hs-fulda.de/grams/dnkfln.htm (online 10-Jun-2009)

2.2 Fokus: Prototyping

Prototyping, früher auch als Prototypenbau bezeichnet, ist eine Methode der Softwareentwicklung, die schnell zu Ergebnissen führt und ein *frühzeitiges Feedback* ermöglicht. *Prototyping* bietet die Chance durch Tatsachen zu überzeugen.[24] Prototyping kann nützlich sein:

- zur Klärung und Festlegung der Systemleistungen (Anforderung, Automationsumfang),
- zur Überprüfung der Machbarkeit des Designs und
- als iterative Vorgehensweise, wobei der Prototyp[25] zu einem immer leistungsfähigeren Produktionssystem heranwächst.

Das Risiko liegt in der Wahl und Modifikation des Prototypen. Es besteht die Gefahr, dass man sich nicht dem gewünschten Ziel nähert (Konvergenz) und die geforderte Qualität nicht erreicht (Qualitätssicherung). Vergleichbar mit der Softwareentwicklungsmethode des „bootstrapping“[26], bei der man sich von einem primitiven Programm zu einer wesentlich leistungsfähigeren Software „hochzieht“, ist auf der Basis einer Strategie des Überzeugens durch Tatsachen und Prototypen, die akzeptable Lösung zu entwickeln.

Ähnlich dem *Bootstrapping*, bei dem zunächst eine funktionsfähige Urzelle erforderlich ist, setzt der erste Schritt ein minimales Akzeptanzpotential voraus. Bedingung ist es daher, den ersten Realisierungsschritt für einen Prototyp, ein Pilotprojekt oder einen Probebetrieb in einer Umgebung zu starten, bei der dieses Minimalpotential nachweislich vorhanden ist. Diese Methode betont das Lernen[27] anhand von konkreten Beispielen (Prototypen).

Das Lernen anhand von Prototypen ist an die Bedingung geknüpft, dass einerseits Fehler und andererseits erkannte Fehlentwicklungen und Unzulänglichkeiten kurzfristig ausräumbar sind. Es bedarf daher signifikanter Vorkehrungen, um die einzelnen Veränderungsschritte des Prototyps so weit wie möglich zielorientiert zu steuern. Dazu gehört, dass Prototyping nicht im aufsichtsfreien Raum stattfinden kann.

Die Feststellung, dass die Fortentwicklung des Prototyps in die gewünschte Zielrichtung läuft und nicht von dieser divergiert, ist vom Auftraggeber im zeitlichen Zusammenhang mit dem jeweiligen Modifikationsschritt zu treffen. Ausreichende Reversibilität besteht selten über eine relativ lange Zeitspanne vieler Veränderungsschritte, sondern im wesentlichen zum jeweiligen Vorgängerschritt.

Die Erfolgskontrolle im Hinblick auf die Konvergenz mit dem angestrebten Ziel ist deshalb synchron zur Modifikationsschrittfolge zu vollziehen. Grundlage dafür ist die resultatsbezogene Kommunikation zwischen allen Akteuren (Beteiligten und Betroffenen). Eine solche

---

[24] Heute auch im Kontext der *Agilen Softwareentwicklung*; siehe dazu das Manifest von 2001 zur Agilen Softwareentwicklung → http://www.agilemanifesto.org (onlie 10-Jun-2009).

[25] Im Rahmen von Software ist „*Prototyp ein unglückliches Wort*“ → [Lud1989]. Häufig bezeichnet es, zum Beispiel in der Fertigungsindustrie, ein Produktmuster, dessen Stückkosten zwar relativ hoch sind, das aber keine Investitionen für eine Serienfertigung beansprucht. Der Prototyp eines Fahrrades ist natürlich kostenintensiver als ein Fahrrad aus der Serie.

[26] Bootstrapping (abgeleitet von bootstrap: Hilfsschlaufe zum Anziehen von Stiefeln) ist eine Methode, die insbesondere beim Compilerbau und in Zusammenhang mit dem Laden eines Betriebssystems angewendet wird.

[27] In diesem Zusammenhang kommen außerdem in Betracht: Lernen anhand von Analogien, Lernen von Heuristiken. Lernen in der trivialsten Form besteht in der Aufnahme von Fakten als Auswendiglernen.

Kommunikation, gestützt auf konkrete eigene Erfahrungen mit dem Leistungspotential des Prototyps, ist weit mehr geeignet, für den Entwicklungsprozeß bedeutsame Fakten aufzudecken als Befragungen, die ohne derartigen Erfahrungsfundus durchgeführt werden.

2.3 Fokus: Methoden implementiert in Tools — Beispiel „*Knowledge Management*“

Die einzelnen Schritte auf dem Weg zur gewünschen Softwarekonstruktion können selbst in Software (Werkzeuge, Tools) gegossen werden. In einer sehr euphorischen IT-Terminologie spricht man heute vom *Knowledge Management* („Wissensmanagement“). Beispielsweise lassen sich Tools zum *Knowledge Management* holzschnittartig klassifizieren aufgrund

- einer produktorientierten Perspektive (Archiv) und

  Diese Tools dienen der Gewinnung, Pflege, Wiederverwendung und Nutzung von Daten („Wissen“) in computerbasierter Form. Paradigmen und Arbeitstechniken (incl. Konzepte, Modelle, *Methoden*, Schrittfolgen etc.) sind dominant geprägt von den Arbeitsfeldern Archiv und Bibliothek.

- einer prozessorientierten Perspektive (CSCW).

  Diese Tools unterstützen die Zusammenarbeit von Gruppen von Personen mit dem Ziel, (verteilt) vorliegendes Wissen und Fähigkeiten optimal zu nutzen. Paradigmen und Arbeitstechniken (incl. Konzepte, Modelle, *Methoden*, Schrittfolgen etc.) sind dominant geprägt von den Arbeitsfeldern *Computer Supported Cooperative Work* (CSCW) und *Workflow Management*.

2.4 *Object-oriented Software Engineering* — Methode

Wenn man im Kontext von Konstruktionswissenschaften Methoden der Informatik skizziert, dann muss auf den Begriff *Methode* im Rahmen der objekt-orientierten Softwareentwicklung verwiesen werden, auch wenn hier der Begriff *Methode* mehr im Sinne eines „Teekesselchens“[28] verwendet wird. In der Objektorientierung werden die einer Klasse von Objekten zugeordneten aktivierbaren Beschreibungen (Algorithmen) als *Methoden* bezeichnet. Üblichwerweise wird diese Bezeichnung synonym zu den Begriffen *Funktion* und *Prozedur* verwendet. Wird einem Objekt $\mathcal{X}$ eine Nachricht $\mathcal{Y}$ geschickt, dann *„ruft $\mathcal{X}$ die zur Nachricht $\mathcal{Y}$ passende Methode $\mathcal{M}$ auf“*.

[28] Ein *Teekesselchen* ist ein Spiel, bei dem die Spieler ein Wort mit mehreren Bedeutungen, aufgrund der Definition der Bedeutungen erraten müssen.
→ http://de.wikipedia.org/wiki/Teekesselchen (online 10-Jun-2006)

3 Das „*weite Feld*“ der Informatik wächst permanent

Unstreitig vergrößert sich das ≫*weite Feld*≪[29] der Informatik permanent und damit auch ihre Methoden, zumindest die, die in Tools gegossen werden. In welche Richtungen geht es in Zukunft? Was sind Hoffnungen, Visionen und Pläne der Promotoren der Informatik? Was kennzeichnet die nächste Technikgeneration, die der Informatik prägende Impulse gibt? Welche (neuen ?) Methoden sind damit verbunden?

Die bisherigen Stufen der Computernutzung (Generationen) können schlagwortartig bezeichnet werden als:

1. *Generation:* COBOL-IBM-Mainframe,
   Charakteristisches Stichwort im Bereich Methoden: Structured Programming (Lineare Kontrollstrukturen)
2. *Generation:* UNIX-DEC-Minicomputer,
   Charakteristisches Stichwort im Bereich Methoden: Verteilte Prozesse (Concurrency pattern)
3. *Generation:* Windows-Intel-PC und
   Charakteristisches Stichwort im Bereich Methoden: Visualisierung
   (Model-View-Controller (MVC)[30])
4. *Generation:* HTTP-Client/Server-Generation.
   Charakteristisches Stichwort im Bereich Methoden: Globale Architektur (service-oriented architecture (SOA))

Wie lautet dann das charakteristsiche Motto der nächsten Generation? Vielleicht *Cloud-Computing*[31] oder ...? Halt, jetzt mutieren die Aussagen zu einem Science-Fiction-Beitrag. Deshalb beschränke ich mich hier auf allerdings plakativ formulierte Thesen, die aber hinreichend realistisch erscheinen:

Hardware: **Alle Technik am Netz**
Der Visionär sieht jede Waschmaschine, jede Drehbank, jede Produktionsstation am Netz. Der Produktmanager telefoniert mit seiner „Waschmaschine“ um den aktuellen Zustand von ihr zu erfahren.

Software: **Selbstreplizierende Programme**
Der Visionär sieht autonome Programme im Netz, die sich durch „Kopulation“ mit anderen Programmen weiterentwickeln. Der Virus von heute wird das Arbeitspferd von morgen.

---

[29] Theodor Fontane [Fon95]; zitiert bei Günter Grass (Grass-1995).

[30] Das MVC-Konzept wurde ≈ 1979 erstmals für die Programmierumgebung *Smalltalk* am *Xerox Palo Alto Research Center* (Xerox PARC) entwickelt. Derzeit ist MVC sogenannter „Stand der Technik“ für die Konstruktion von Softwaresystem, die von User-Interaktionen geprägt sind.

[31] *Cloud Computing* bezeichnet eine Computernutzung bei der die Infrastruktur, also Computer, Speicher und Netze, und die benötigte Software, also Betriebssysteme, Anwendungen, Middleware, Management- und Entwicklungs-Tools, je nach dem aktuellen Bedarf dynamisch zusammengestellt werden. Beispielsweise laufen dann Wertschöpfungsketten (Geschäftsprozesse) auf IT-Komponenten, die sich als Dienstleistungsknoten in globalen Netzen befinden. Für die Datensicherheit und den Datenschutz (Gewährleistungsarchitektur) ergeben sich damit besondere Anforderungen.

Organisation: **Virtuelle juristische Personen**
Der Visionär sieht virtuelle Organisationseinheiten, die eine juristische[32] Person sind, und eine steuerliche Ansässigkeit haben.

Nutzer & Betroffene: **Mehr Lebensqualität durch „Maschinisierung von Kopfarbeit"**
Der Visionär sieht, wie sich die „Maschinisierung von Kopfarbeit" zur bedeutsamen Quelle für mehr Lebensqualität entwickelt. Der Verstärker des Wissens mutiert zum Verstärker der Lebensqualität für Viele.

[32] Beispielsweise ähnlich haften wie eine Aktiengesellschaft.

Literaturverzeichnis

[Bonin88] Hinrich Bonin; Die Planung komplexer Vorhaben der Verwaltungsautomation, Heidelberg (R. v. Decker & C. F. Müller), 1988 (Schriftenreihe Verwaltungsinformatik; Bd. 3).

[Bro1975] Frederick Brooks, Jr.; The Mythical Man-Month, Reading Massachusetts, Menlo Park California (Addison-Wesley) 1975. [Hinweis: Die klassische Kritik der Planungsdimension *Lines of Code* (LOC).]

[Coy+92] Wolfgang Coy / Frieder Nake / Jörg-Martin Pflüger / Arno Rolf / Jürgen Seetzen / Dirk Siefkes / Reinhard Stransfeld (Hrsg.); Sichtweisen der Informatik, Braunschweig, 1992. [Hinweis: Eine gesellschaftspolitisch geprägte Analyse und Perspektive der Informatik.]

[DeKr76] Frank DeRemer / Hans Kron; PROGRAMMING-IN-THE LARGE VERSUS PROGRAMMING-IN-THE-SMALL, University of California, Santa Cruz
http://www.cs.umd.edu/class/spring2003/cmsc838p/General/pitl.pdf
(online 15-May-2008)

[GI2005] Gesellschaft für Informatik e. V.; Was ist Informatik? — GI-Positionspapier Juli 2005 —, Stand Mai 2006, Wissenschaftszentrum, Ahrstraße 45, D-53175 Bonn
http://www.gi-ev.de/fileadmin/redaktion/Download/was-ist-informatik-lang.pdf
(online 19-Mar-2009)

[Fon95] Theodor Fontane; Effi Briest, Roman, erstmals 1894 in der *Deutschen Rundschau* erschienen; 1895 in Buchform. Heute in vielfältigen Ausgaben verfügbar. [Hinweis: Ein Klassiker, der häufig verfilmt wurde.]

[Grass95] Grass, Günter; Ein weites Feld, Göttingen (Steidl Verlag), 1. Auflage 1995, ISBN 3-88243-366-3. [Hinweis: Eine Auseinandersetzung mit der deutschen Wiedervereinigung, verbunden mit einem Disput über das Werk von Theodor Fontane. Unterstellt wird eine Analogie der Konstellationen bei der Reichsgründung 1870/71, in der Art und Weise wie Theodor Fontane (1819-1898) sie erlebte, mit der Wiedervereinigung von BRD und DDR im Jahre 1989.]

[Kl03] Helmut Klemm; Ein großes Elend — Das Informationszeitalter kann sich nicht einigen über den Begriff „Information“, in: Informatik Spektrum, Band 26, Heft 4, August 2003, S. 267-273. [Hinweis: Primär eine Fortführung der Begriffsdiskussion, die von Günter Ropohl (Technikwissenschaftler, Universität Frankfurt) in der Zeitschrift „Ethik und Sozialwissenschaften“ angestoßen wurde.]

[Lud1989] Jochen Ludewig; Modelle der Software-Entwicklung — Abbilder oder Vorbilder? in: GI (Gesellschaft für Informatik) Software Trends, Band 9 Heft 3, Oktober 1989, S. 1-12. [Hinweis: Ein erster fundierter Überblick über Modelle zur Vorgehensweise.]

[LudLich2007] Jochen Ludewig / Horst Lichter; Software Engineering — Grundlagen, Menschen, Prozesse, Techniken — Heidelberg (dpunkt.verlag GmbH) 2007, ISBN 3-89864-268-2. [Hinweis: Gut geeignet für das Selbststudium.]

[Me+04] Peter Mertens / Reimut Bodendorf / Wolfgang König / Arnold Picot / Matthias Schumann /Thomas Hess; Grundzüge der Wirtschaftsinformatik, Berlin Heidelber u. a. (Springer) 1991, fünfte, neubearbeitete Auflage 1998, ISBN 3-540-63752-4; achte Auflage 2004, ISBN 3-540-40687-5. [Hinweis: Ein Lehrbuch, das integrierte Anwendungssysteme in den Mittelpunkt der Betrachtungen stellt.]

[Na92] Frieder Nake; Informatik und Maschinisierung von Kopfarbeit, in: [Coy+92], S. 181-201. [Hinweis: Zum Schaffen von Frieder Nake [Rö03].]

[Pir1974] Robert M. Pirsig; Zen und die Kunst ein Motorrad zu warten — Ein Versuch über Werte, Frankfurt (S. Fischer Verlag GmbH) 1974, amerikanischer Originaltitel *Zen and the Art of Motorcycle Maintenance*, übersetzt von Rudolf Hermstein, ISBN 3-10-061901-3. [Hinweis: Im Kontext Systemanalyse ist die fundiert dargelegte Frage nach der „richtigen Einstellung“ relevant. — Kultbuch der ersten Informatiker-Generation!]

[Rech03] Peter Rechenberg; Zum Informationsbegriff der Informationstheorie, in Informatik-Spektrum, Band 26, Heft 5, 2003, S. 317-326. [Hinweis: Eine fundierte Erläuterung des Begriffs „Information“.]

[Rech04] Peter Rechenberg; Stellungnahme des Verfassers zur Diskussion des Begriffs Information, in Informatik-Spektrum, Band 27, Heft 1, 2004, S. 92-93. [Hinweis: Eine Zusammenfassung von Diskussionsbeiträgen über den Begriff „Information“.]

[Rö03] Karl-Heinz Rödiger (Hrsg.); Algorithmik — Kunst — Semiotik, Hommage für Frieder Nake, Heidelberg (Synchron Wissenschaftsverlag) 2003, ISBN 3-935025-60. [Hinweis: Festschrift für Frieder Nake, einer der großen Pioniere der Computergraphik.]

[WiHe07] Thomas Wilde / Thomas Hess; Forschungsmethoden der Wirtschaftsinformatik — Eine empirische Untersuchung, in: WIRTSCHAFTSINFORMATIK 49 (2007) 4, S. 280-287. [Hinweis: Eine systematische Darstellung der Forschungsmethoden.]

[WiHe06] Thomas Wilde / Thomas Hess; Methodenspektrum der Wirtschaftsinformatik: Überblick und Portfoliobildung, Arbeitsbericht Nr. 2/2006, Institut für Wirtschaftsinformatik und Neue Medien der Ludwig-Maximilians-Universität München, Ludwigstr. 28 VG, D-80539 München
http://www.wim.bwl.uni-muenchen.de/pubdb/work_papers/2006-003.html
(online 29-May-2009)

* * *

Automatisierte Ermittlung von Umweltwirkungen entlang der Wertschöpfungskette

Burkhardt Funk, Peter Niemeyer[1]

Abstract: Die Integration von BUIS und ERP-Systemen ist seit langem als wichtiger Schritt in Richtung eines holistischen und langfristig orientierten betrieblichen Umweltmanagements erkannt worden. In den letzten Jahren wurden eine Reihe von Forschungsprojekten durchgeführt, deren Ziel darin bestand die Umweltwirkungen von Produkten in ERP-Systemen abzubilden und in die operativen Entscheidungsfindungsprozesse zu integrieren. In dem vorliegenden Beitrag[2] stellen wir eine Referenzarchitektur zur automatisierten Ermittlung der Produkt-Umweltwirkung entlang der Wertschöpfungskette von Produkten vor. Dabei gehen wir davon aus, dass zur Ermittlung der Umweltwirkung der eigenen Fertigungsprozesse BETRIEBLICHE UMWELTINFORMATIONSSYSTEME (BUIS) IM EINSATZ SIND, WÄHREND DIE UMWELTWIRKUNGEN ZU FREMDBEZOGENEN PRODUKTEN ENTWEDER VON LIEFERANTEN ODER ÜBER GEEIGNETE LCA-DATENBANKEN ZUR VERFÜGUNG GESTELLT WERDEN.

1 Einleitung

In [FMN09] haben wir dargestellt, dass die Integration von betrieblichen Umweltinformationssystemen (BUIS) und ERP-Systemen eine wichtige Voraussetzung für die Berücksichtigung von Umweltwirkungen in der unternehmerischen Entscheidungsfindungen auf operativer und strategischer Ebene ist. Eine solche Integration war schon in den frühen 1990er Jahren Gegenstand verschiedener Forschungsprojekte. Die daraus resultierenden Konzepte beziehen sich oft auf vollständige Ökobilanzen, in denen die Umweltwirkungen von Produkten über den gesamten Lebenszyklus. (cradle-to-grave) gesammelt werden. Eine Etablierung der Konzepte über den prototypischen Status hinaus fand bislang nicht statt.

ERP-Systeme können heute nicht nur zur Abbildung der Materialströme, sondern auch zur Abbildung der zugehörigen Stoffströme eingesetzt werden, welche ihrerseits als Datenbasis für die Sachbilanzen dienen können. Der Einsatz von ERP-Systemen zur Ermittlung von Sachbilanzen erfordert allerdings erheblichen manuellen Aufwand bei der Ermittlung und Erfassung der Umweltwirkungen fremd beschaffter Materialien. Zwar stehen entsprechende Daten teilweise auf externen LCA-Datenbanken zur Verfügung, aufgrund der in der LCA-Community anzutreffenden Methodenvielfalt sind diese Daten aber nur selten maschinell verarbeitbar. Da es darüber hinaus kein allgemein akzeptiertes Vokabular für Umweltwirkungen gibt, haben wir es hier mit einer sogenannten semantischen Lücke zu tun. Wir sehen darin den wichtigsten Grund dafür, dass die Integration von BUIS und ERP-Systemen bis heute nicht über prototypische Implementierungen hinausgekommen ist.

[1] {funk|niemeyer}@uni.leuphana.de
[2] Der vorliegende Beitrag basiert auf einem Vortrag auf der Jahrestagung der GI 2009 in Lübeck

Wenn ERP-Systeme zur Verwaltung von Umweltwirkungen eingesetzt werden sollen, müssen daher neue Methoden zur Überwindung der semantischen Lücke entwickelt werden.

Um die Umweltwirkung eines Unternehmens und seiner Produkte zu berechnen schlagen wir vor, sich in einem ersten Schritt auf den Teil der Wertschöpfungskette zu konzentrieren, für den das Unternehmen selbst verantwortlich ist. Um die Zuständigkeiten klar zu trennen, und die Komplexität zu reduzieren nehmen wir an, dass alle Lieferanten Dienste zur Verfügung stellen, mit denen die Umweltwirkungen der von ihnen gelieferten Produkte abgefragt werden können. Da solche Dienste heute nur selten verfügbar sind, sehen wir eine wichtige Aufgabe darin (grobe) Abschätzungen für die Umweltwirkung fremd beschaffter Produkte zur Verfügung zu stellen. Dieses Problem werden wir in zukünftigen Arbeiten untersuchen.

Die vorliegende Arbeit ist wie folgt aufgebaut: zunächst geben wir einen Überblick über die bisherigen Forschungsprojekte zur Integration von BUIS und ERP-Systemen. Anschließend stellen wir eine Architektur vor, die es Unternehmen erlaubt, Kunden Auskunft über die Umweltwirkungen der an sie gelieferten Produkte zu geben.

2 Bisherige Forschungsergebnisse

Für die hier diskutierte Fragestellung sind Arbeiten aus zwei Forschungsfeldern relevant. In der Wirtschaftsinformatik gab es verschiedene Projekte zur BUIS/ERP-Integration, in denen als Forschungsmethoden unter anderem Prototyping, Referenzmodellierung und Feldstudien zum Einsatz kamen. Daneben nutzen wir für unsere Untersuchung Ergebnisse aus dem Bereich Life Cycle Impact Assessment.

2.1 Neuere Ergebnisse zur EMIS/ERP-Integration

Verschiedene Autoren (vgl. z.B. [Rau97, KDF+00]) haben schon früh die Relevanz der BUIS/ERP-Integration erkannt und untersuchten vor allem die Integration von Stoffstrommanagementsystemen und Systemen zur Produktionsplanung und -steuerung (PPS) [IR07]. Hierzu wurden Stoffstromnetzwerke [Möl94] aus Materialstücklisten, Arbeitsplänen und Fertigungsaufträgen konstruiert. Wesentliche Problemfelder sind hierbei (i) die unterschiedlichen Anforderungen bezüglich Granularität und Verfügbarkeit von Daten [MGPR04, MGGS06], sowie (ii) mögliche Redundanzen zwischen PPS und Stoffstrommanagementsystemen.

Unter den Forschungsprojekten zum Thema BUIS/ERP-Integration findet sich auch das ECO-Integral-Projekt [KDF+00] im Rahmen dessen eine Referenzarchitektur für die horizontale Integration von BUIS und ERP-Systemen entwickelt wurde, sowie eine Reihe weiterer Projekte unter Federführung des Instituts für Arbeitswissenschaften und Technologiemanagement (IAT) an der Universität Stuttgart und am Fraunhofer Institut für Arbeitswissenschaften und Organisation (FAO) [BEH00, BB04, LRW+03, LSL+04, LKH07].

Ein ähnlicher Ansatz wurde im Rahmen des Transferbereiches Umweltgerechte Produkte durch optimierte Prozesse, Methoden und Instrumente in der Produktentwicklung an der TU Darmstadt verfolgt [AABR07]. Zur automatischen Erstellung von produktbezogenen Ökobilanzen wurde ein ERP-System prototypisch dahingehend erweitert, dass die im System vorhandenen Material- und Prozessdaten als Basis für eine semi-automatische Ökobilanz verwendet werden können. Ein auf der SAP-Komponente EHS aufbauender Prototyp

unterstützt dabei alle Phasen des Lifecycle Assessment nach ISO 14040. Zusammenfassend ist festzustellen, dass ERP-Systeme grundsätzlich in der Lage sind Stoffströme abzubilden und zumindest in Teilen als Datenbasis für ein Life Cycle Impact Assessment zu dienen. Der praktische Einsatz der skizzierten Lösungen erfordert allerdings in hohen Maßen manuelle Eingriffe, etwa zur Pflege der stofflichen Zusammensetzung von Materialien und der Beschreibung ihrer bisherigen Umweltwirkungen.

2.2 Life cycle impact assessment

Auf der Basis von Emissions- und Abfalldaten versucht das Life Cycle Impact Assessment (LCIA) die Umweltbelastung von Produkten, Prozessen und ganzen Unternehmen zu berechnen. Hierzu werden einzelne Positionen der Sachbilanz zu möglichen Umweltschäden oder Umweltwirkungen in Beziehung gesetzt. Nach Jolliet et al. [JMWB+04] werden in diesem Bereich zwei Ansätze unterschieden:

- Unter Einsatz quantitativer Modelle leiten klassische LCIA Methoden (z.B. [GHH+02]) aus der Sachbilanz sogenannte midpoint level categories ab, etwa global warming potential oder acidification. Der Begriff midpoint level weist darauf hin, dass klassische LCIA Methoden nicht das Ende der Wirkungsketten der Umweltschäden (etwa die Auswirkungen auf die menschliche Gesundheit) in Betracht ziehen.
- Schadensorientierte Methoden (vgl. [GS01]) modellieren die Ursache-Wirkungs-Kette bis zu den folgenden Endpunkten: (i) menschliche Gesundheit, gemessen in DALY (Disability Adjusted Life Years), (ii) Qualität des Ökosystems, gemessen in dem Anteil ausgestorbener Arten, und (iii) verbleibenden Menge natürlicher Ressourcen und ihre Abbaubarkeit.

Offensichtlich können Ergebnisse der schadensorientierten Methoden einfacher interpretiert werden, sie enthalten aber auch einen höheren Grad von Unsicherheit, da die modellierten Wirkungszusammenhänge noch nicht abschließend erforscht sind. Um solche Unsicherheiten in der Ermittlung von Umweltwirkungen entlang der Wertschöpfungskette möglichst gering zu halten, schlagen wir vor, die Methode der midpoint level categories anzuwenden.

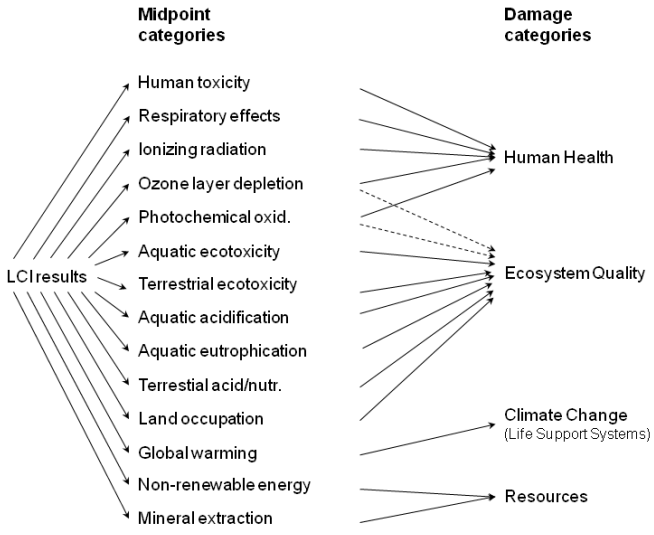


Abbildung 1: Die Zuordnung von LCI Daten über midpoint level categories zu Schadenskategorien (aus [JMC+03])

3 Referenzarchitektur

Im folgenden betrachten wir den Kunden eines Unternehmens, der die Umweltwirkung eines Produktes untersucht, das er von dem Unternehmen bezogen hat. Das Unternehmen stellt hierzu einen Webservice zur Verfügung, der die Umweltwirkung der Produkte über spezielle Wirkungskategorien beschreibt (vgl. Abschnitt 2.2). Betrachten wir hierzu die in Abb. 2 skizzierte Systemlandschaft. Die Informationssysteme (IS) des Unternehmens (Abb. 2, schraffierter Bereich) werden in erster Linie zur Berechnung der Umweltwirkung der eigenen Fertigungsprozesse verwendet. Darüber ermittelt dieses System die zu einem Produkt gehörenden fremd beschafften Materialien und stellt sie dem Prozess zur Verfügung. Das IS des Unternehmens besteht aus einem ERP-System, einer Integrationsplattform und einem BUIS. Zur Ermittlung der im Rahmen der vorangehenden Wertschöpfung verursachten Umweltwirkung stellen die Lieferanten entweder einen entsprechenden Webservice zur Verfügung oder die Abschätzung erfolgt unter Verwendung von LCA Datenbanken oder Input/Output-Tools.

Um sicherzustellen, dass die einmal ermittelten Umweltwirkungen reproduzierbar sind wird ein so genannter Environmental Impact Server eingebunden, der alle einmal bezogenen Umweltwirkungen (von Lieferanten, Datenbanken oder dem internen BUIS) persistiert. Die Unternehmens IS wird durch den Environmental Impact Workplace vervollständigt, über den manuell in den Datenermittlungsprozess eingegriffen werden kann, etwa zur Anreicherung maschinell nicht ermittelbarer Umweltwirkungen.

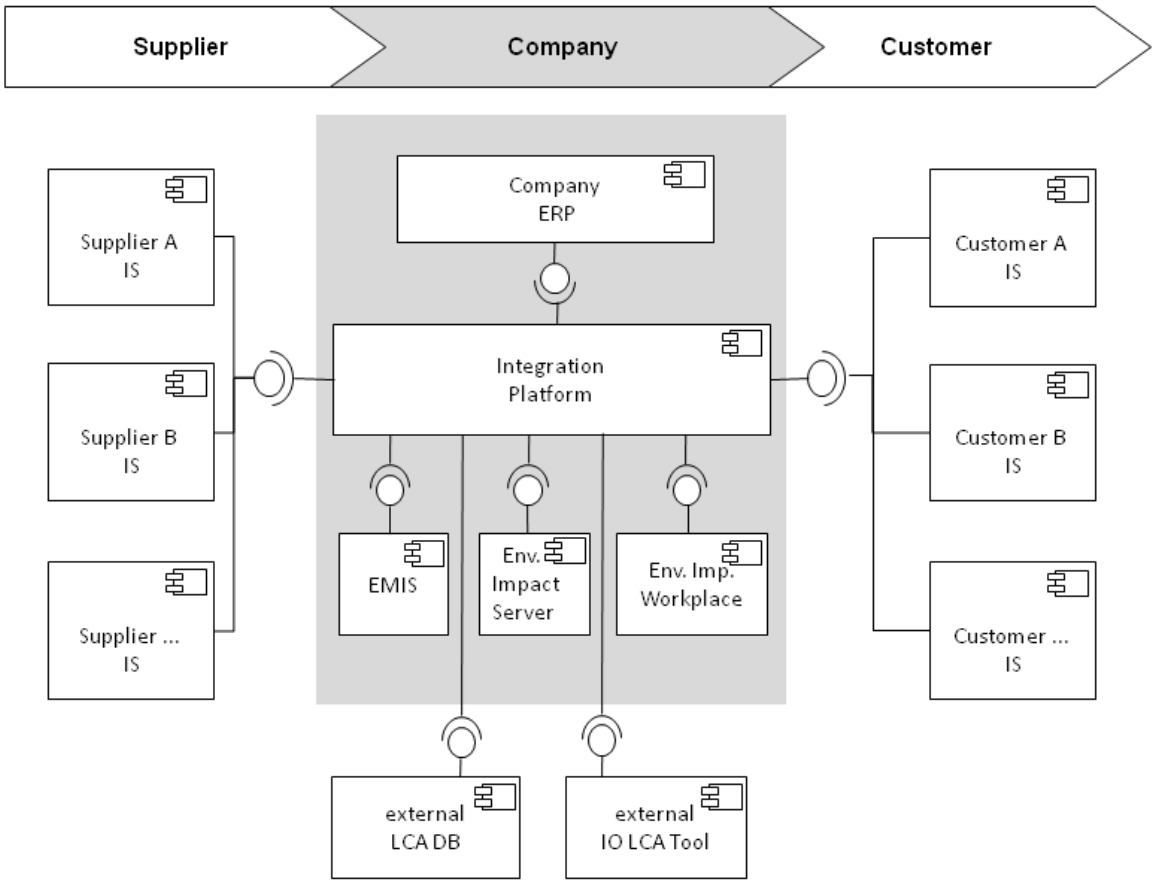


Abbildung 2: Am Prozess beteiligte Systeme (hervorgehobene Bereiche beschreiben interne Systeme des betrachteten Unternehmens)

Folgende Systeme und zugehörige Schnittstellen werden verwendet (vgl. Abb.2):

- Customer IS: Diese Systeme gehören Kunden, welche die Umweltwirkung zu einem Produkt dokumentieren möchten, für das sie sich interessieren, oder das sie bereits gekauft haben. Bezieht sich die Anfrage auf ein schon gekauftes Produkt, ist in der Schnittstelle die konkrete Lieferscheinposition des zugrundeliegenden Kaufs zu spezifizieren. Handelt es sich hingegen um eine informelle Anfrage, so ist die zugehörige Artikelnummer anzugeben. In beiden Fällen liefert die Schnittstelle einen Vektor mit den Umweltwirkungen bezüglich fest vorgegebener Wirkungskategorien (midpoint level categories) zurück.

- Supplier IS: Idealerweise bieten die Lieferanten denselben Webservice an, der auch vom betrachteten Unternehmen angeboten wird (identische Schnittstellen). Über die Lieferscheinposition der fremdbeschafften Produkte kann das Unternehmen dann die Umweltwirkung der vorangehenden Wertschöpfungskette ermitteln. Falls ein solcher Service nicht angeboten wird, muss eine Abschätzung unter Einsatz von LCA-Datenbanken oder IO-Tools oder auch manuell erfolgen.

- Company ERP: dieses System kennt die Transaktionsdaten zu den Beschaffungs-, Fertigungs- und Vertriebsprozessen des Unternehmens. Es stellt einen Service zur Verfügung, der zu jedem verkauften Produkt die konkreten Fertigungsprozesse und alle verarbeiteten Produkte bereit stellt. Zu den Fertigungsprozessen werden insbesondere Stücklisten und Arbeitspläne angeliefert. Zu den verarbeiteten Produkten werden die Lieferanten und die der Beschaffung zugrundeliegende externe Lieferscheinnummer angegeben. In der Praxis lässt sich in vielen Fällen nicht nachverfolgen, zu welcher Beschaffungsposition ein im Rahmen der Fertigung verbrauchtes Material gehört (z.B. aufgrund von Sammelbestandsführung). In diesen Fällen müssen im Company ERP-System konsistente Annahmen über die Verbrauchsfolge der beteiligten Materialien getroffen werden.

- EMIS: Das interne Betriebliche Umweltinformationssystem kennt die Umweltwirkung der Fertigungsprozesse des Unternehmens. Darüber hinaus kennt das EMIS die sonstige Umweltwirkung des Unternehmens, die nicht einzelnen Fertigungsprozessen zugeordnet werden kann, und ist in der Lage, eine Umlage dieser Overhead Wirkung auf die einzelnen Fertigungsprozesse vorzunehmen. Das EMIS stellt einen Service zur Verfügung, mit dem zu einem angegebenen Fertigungsbeleg die Umweltwirkung des zugrundeliegenden Fertigungsprozesses ermittelt werden kann.

- LCA DB: Über diese externen Datenbanken sollen anhand von Produktklassifikation Abschätzungen der Produktumweltwirkung ermittelt werden. Die heute üblichen nationalen und internationalen LCA Datenbanken erfüllen diesen Zweck nicht (z.B. [FJA+05]), da sie darauf ausgerichtet sind komplette LCA-Studien zur Verfügung zu stellen, die für eine automatisierte Verarbeitung nicht verwertbar sind. Der Aufbau muss von der LCA-Community geleistet werden.

- IP: Die Integrationsplattform stellt den Webservice zur Abfrage der Produkt-Umweltwirkung zur Verfügung und koordiniert die prozessorientierte Kommunikation mit internen und externen Systemen.

- IO LCA tool: Basierend auf Preis und Branchenzuordnung eines Produktes, sowie den Input/Output-Daten der Volkswirtschaft und den Environmental interventions der einzelnen Sektoren ermittelt ein solches System Abschätzungen für die Umweltwirkung von Produkten (z.B. [SK05, WMBW06]).

- Environmental impact server: Die Aufgabe des Environmental Impact Servers besteht darin, alle ermittelten Produkt-LCIs und die zugehörigen Rechenschritte (Zwischenergebnisse) zu persistieren. Er dient dadurch als Puffer für einmal berechnete Umweltwirkungen. Darüber hinaus ist er in der Lage, alle einmal nach außen gegebenen Umweltwirkungen zukünftig detaillieren zu können und so die Nachvollziehbarkeit einmal erteilter Auskünfte zu gewährleisten.

- Environmental impact workplace: Hierbei handelt es sich um ein Workflow Management System, über das in bestimmten Situationen die extern ermittelten Umweltwirkungen nachbearbeitet werden können. Dieses ist immer dann erforderlich, wenn die Umweltwirkung der vorangehenden Wertschöpfungskette weder über den Lieferanten, noch über LCA DB oder IO LCA Tools ermittelt werden können. Eine Überprüfung könnte aber auch abhängig von der Qualität der ermittelten Umweltwirkung angestoßen werden (z.B. immer dann, wenn von Lieferanten gemeldete Umweltwirkungen unerwartet gering ausfallen).

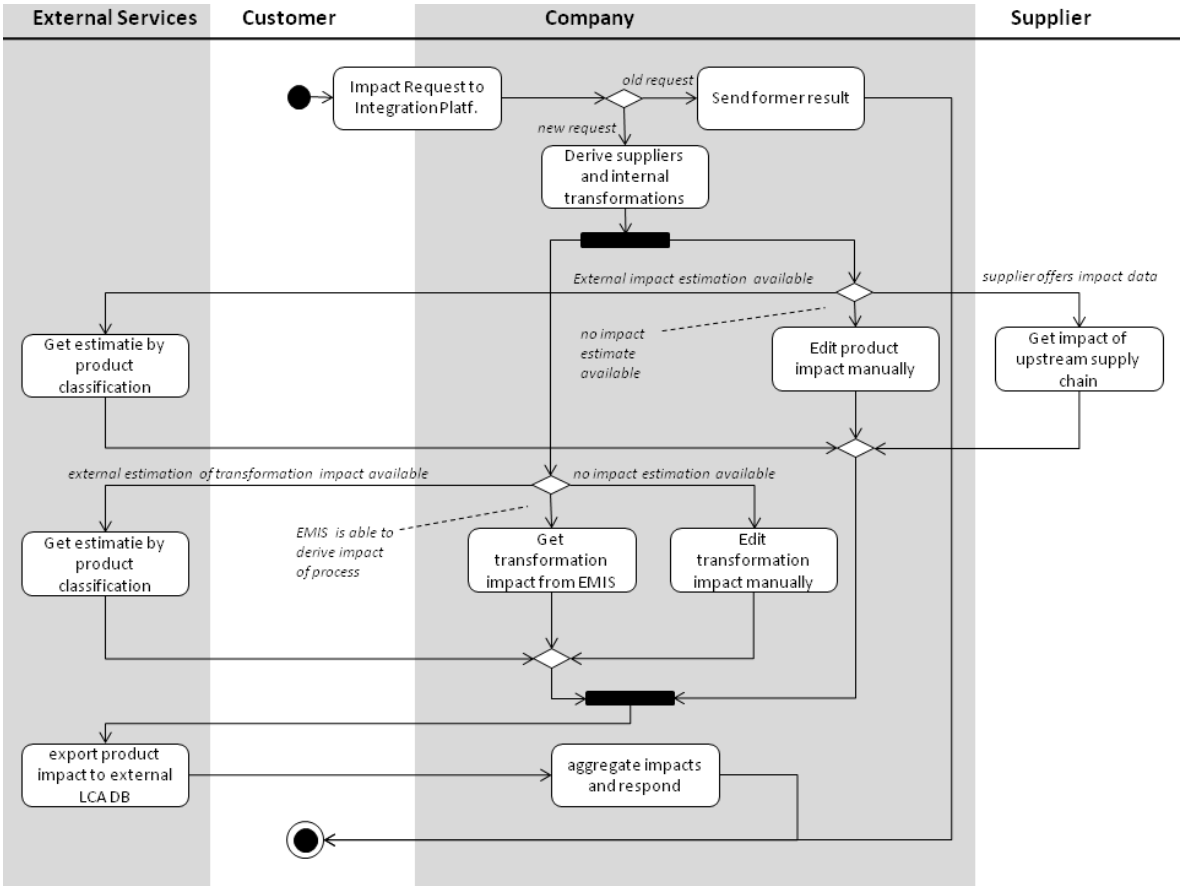


Abbildung 3: Das Aktivitätsdiagramm stellt den vorgeschlagenen Prozess zur Ermittlung der Umweltwirkung eines Produktes dar

Aufbauend auf dieser Infrastruktur kann der folgende Prozess (vgl. Abb. 3) zur Ermittlung produktbezogener Umweltwirkungen implementiert werden. Ein Kunde fragt zu einem von ihm bezogenen Produkt beim Hersteller die Umweltwirkung ab. Die Anfrage wird über einen dafür vorgesehenen Service gestellt, der auf der Integrationsplattform (IP) des Unternehmens zur Verfügung gestellt wird. Der Kunde spezifiziert das Produkt über die zugehörige Lieferscheinposition. Die Integrationsplattform fragt nun zunächst beim Environmental Impact Server nach, ob die Umweltwirkungen zu dem konkreten Lieferschein schon früher ermittelt wurden und liefert sie ggf. an den Kunden zurück. Handelt es sich hingegen um eine neue Anfrage, ermittelt die Integrationsplattform zunächst die konkreten Fertigungsprozesse und Beschaffungspositionen, die der Erzeugung des Produktes zugrundeliegen. Eine entsprechende Anfrage wird unter Angabe der Lieferscheinnummer an das ERP-System gestellt, das eine Liste der Beschaffungspositionen (Lieferant, Datum, Produkt, Betrag, Preis etc.) und eine Liste der Fertigungsbelege (Stückliste, Arbeitsplan, in der Fertigung eingesetzte Arbeitsplätze) bereitstellt. Auf Grundlage der so ermittelten Beschaffungspositionen ermittelt die Integrationsplattform im nächsten Schritt die Umweltwirkung der vorangehenden Wertschöpfungsstufen. Hierzu wird zunächst überprüft, ob der Lieferant einen Service zur Abfrage entsprechender Daten zur Verfügung stellt. Ist dies nicht der Fall, wird eine Anfrage an eine der angeschlossenen LCA-Datenbanken gesendet. Führen auch diese Anfragen nicht zum Erfolg, wird die Anfrage an ein angebundenes IO-Tool weitergereicht. Bleibt auch diese Anfrage erfolglos, wird die entsprechende Position für die manuelle Nachbearbeitung im Environmental Impact Workplace vorgemerkt. Sobald alle Daten vorliegen, meldet IP die Summe der Umweltwirkungen aus den relevanten Beschaffungspositionen und den Fertigungsprozessen als Umweltwirkung des Produktes an den Kunden zurück.

4 Zusammenfassung und Ausblick

In der vorliegenden Arbeit haben wir eine Referenzarchitektur vorgestellt, die den automatischen Ausweis von Produkt-Umweltwirkungen gegenüber Kunden ermöglicht. Die vorgestellte serviceorientierte Architektur zeichnet dadurch aus, dass

- minimale Eingriffe in bestehende operative Unternehmensanwendungen (z.B. ERP-Systeme) erforderlich sind,
- die Berechnung von Umweltwirkungen der vorangehenden Wertschöpfungsketten nur die Einbeziehung der direkten Lieferanten erfordert,
- extern vorliegende Daten zur Umweltwirkung von Produkten weitestmöglich berücksichtigt werden,
- eine initiale Pflege der Umweltwirkungen auf der Ebene von Materialien bzw. Fertigungsprozessen nicht notwendig ist, da entsprechende Daten erst bei konkreten Kundenanfragen ermittelt werden.

Zur Zeit wird an der Leuphana Universität Lüneburg auf Basis der vorgestellten Architektur ein Prototyp entwickelt, der dann als Ausgangspunkt für eine umfangreiche Machbarkeitsanalyse verwendet werden soll. Als Integrationsplattform für die Entwicklung der Schnittstellen und Prozesse setzen wir zur Zeit die SAP Exchange Infrastructure ein.

In einem nächsten Schritt werden wir untersuchen, wie die zur Verfügung gestellten Umweltwirkungen zur Prozessverbesserung in den beteiligten ERP-Systemen verwendet werden können: als Basis für operative Entscheidungen ebenso wie im Rahmen der strategischen Planung und des Unternehmensrisikomanagements.

Literatur

[AABR07] E. Abele, R. Anderl, H. Birkhofer und B. Rüttinger, Hrsg. *EcoDesign: von der Theorie in die Praxis.* Springer-Verlag, Berlin, Heidelberg, 2007.

[BB04] T. Busch und S. Beucker. Computergestützte Ressourceneffizienz-Rechnung, 2004.

[BEH00] H. J. Bullinger, W. Eversheim und H. Haasis. *Auftragsabwicklung optimieren nach Umwelt- und Kostenzielen. OPUS - Organisationsmodelle und Informationssysteme für einen produktionsintegrierten Umweltschutz.* Springer, Berlin, 2000.

[FJA+05] R. Frischknecht, N. Jungbluth, H.-J. Althaus, G. Doka und R. Dones et al. The ecoinvent Database: Overview and Methodological Framework. *The International Journal of Life Cycle Assessment,* 10(1): 3-9, 2005.

[FMN09] B. Funk, A. Möller und P. Niemeyer. Integration of Environmental Management Information Systems and ERP systems using Integration Platforms. In I. N. Athanasiadis, P. A. Mitkas, A.-E. Rizzoli und J. Marx-Gómez, Hrsg., *Information Technologies in Environmental Engineering,* Seiten 53-63, Berlin, 2009. Springer.

[GHH+02] J. B. Guinée, R. Heijungs, G. Huppes, R. Kleijn und A. de Koning et al. *Handbook on Life Cycle Assessment. Operational Guide to the ISO Standards.* Klumwer, Dordrecht, 2002.

[GS01] M. Goedkoop und R. Spriensma. The Eco-indicator 99 - A damage oriented method for LCIA, 2001.

[IR07] R. Isenmann und C. Rautenstrauch. Horizontale und vertikale Integration Betrieblicher Umweltinformationssysteme (BUIS) in Betriebswirtschaftlichen Anwendungsszenarien. uwf - *UmweltWirtschaftsForum,* 15(2): 75-81, 2007.

[JMC+03] O. Jolliet, M. Margni, R. Charles, S. Humbert und J. Payet et al. IMPACT 2002+: A New Life Cycle Impact Assessment Methodology. *International Journal on Life Cycle Assessment,* 8(6): 324-330, 2003.

[JMWB+04] O. Jolliet, R. Müller-Wenk, J. Bare, A. Brent, M. Goedkoop und R. Heijungs et al. The LCIA Midpoint-damage Framework of the UNEP/SETAC Life Cycle Initiative. *The International Journal of Life Cycle Assessment,* 9(6): 394-404, 2004.

[KDF+00] H. Krcmar, G. Dold, H. Fischer, M. Strobel und E. K. Seifert. *Informationssysteme für das Umweltmanagement: Das Referenzmodell eco-integral.* Oldenbourg, München, 2000.

[LKH07] C. Lang-Kötz und D. Heubach. *Umweltcontrolling umsetzen - Erstellung von Kennzahlen für Stoff- und Energieströme und deren Integration in die betriebliche IT.* Fraunhofer IRB Verlag, Stuttgart, 2007.

[LRW+03] C. Lang, U. Rey, V. Wohlgemuth, S. Genz und S. Pawlytsch. PAS *1025 - Austausch umweltrelevanter Daten zwischen ERP-Systemen und betrieblichen Umweltinformationssystemen.* Beuth Verlag, 2003.

[LSL+04] C. Lang, M. Steinfeldt, T. Loew, S. Beucker, D. Heubach und M. Keil. Konzepte zur Einführung und Anwendung von Umweltcontrollinginstrumenten in Unternehmen: Endbericht des Forschungsprojekts INTUS, 2004.

[MGGS06] J. Marx Gómez, J. Görmer und D. Schlehf. Behandlung von Datendefekten in Stoffstromnetzen. In Wittmann, J., Hrsg., *ASIM - Arbeitsgemeinschaft Simulation, Workshop Simulation in Umwelt- und Geowissenschaften,* Seiten 85-98, Aachen, 2006.

[MGPR04] J. Marx Gómez, S. Prötzsch und C. Rautenstrauch. Data Defects in Material Flow Networks - Classification and Approaches. Cybernetics and Systems. *An International Journal,* 35(5-6): 549-558, 2004.

[Mö194] A. Möller. Stoffstromnetze. In L. M. Hilty, A. Jaeschke, B. Page und A. Schwabl, Hrsg., *Informatik für den Umweltschutz,* Jgg. 2, Seiten 223-230, Hamburg, 1994.

[Rau97] C. Rautenstrauch. *Fachkonzept für ein integriertes Produktions-, Recyclingplanungs- und -steuerungssystem.* de Gruyter, Berlin, 1997.

[SK05] S. Suh und S. Kagawa. Industrial Ecology and Input-Output-Economics: an Introduction. *Economic Systems Research,* 17(4): 349-364, 2005.

[WMBW06] T. Wiedmann, J. Minx, J. Barretta und M. Wackernagel. Allocating ecological footprints to final consumption categories with input-output analysis. *Ecological Economics*, 56(1): 28-48, 2006.

Testmethoden zur Qualitätssicherung von Tabellenkalkulationsanwendungen

Jürgen Jacobs, Phillip Tiburtius

1 Einleitung

Die erste Tabellenkalkulationssoftware VisiCalc erlangte in den späten 1970er Jahren Marktreife und wird gemeinhin als „Killer“-Applikation für den Personal Computer bezeichnet (Power 2004). Seitdem hat sich der Markt für Tabellenkalkulationssoftware und damit das sogenannte End-User-Computing rasant entwickelt. Schätzungen gehen davon aus, dass im Jahr 2012 allein in den USA 55 Millionen Endbenutzer Tabellenkalkulations- oder Datenbankanwendungen nutzen werden, wovon sich 13 Millionen als Programmierer bezeichnen würden. Dies übertrifft die erwartete Anzahl professioneller Programmierer von etwa 3 Millionen um mehr als das Vierfache (Scaffidi et al. 2005). Diese Zahlen und die schwer von der Hand zu weisende Befürchtung, dass die Entwicklung von Tabellenkalkulationsanwendungen durch Endbenutzer ohne Ausbildung in der Softwareentwicklung einem erhöhten Fehlerrisiko unterliegen, machen die Bedeutung von Testmethoden zur Qualitätssicherung von Tabellenkalkulationsanwendungen deutlich. Dies umso mehr, als in der Vergangenheit zahlreiche spektakuläre Fälle von Fehlern in Tabellenkalkulationsanwendungen mit schwerwiegenden Auswirkungen bekannt geworden sind. Der Verlust von 24 Million US-Dollar eines kanadischen Energieanbieters, der durch einen cut-and-paste-Fehler entstand, ist nur einer von 89 Fällen, die auf den Seiten der European Spreadsheet Risks Interest Group (EuSpRIG) beschrieben werden (O'Beirne 2003, 2004). Auch eine aktuelle Studie von Powell et al. (2009) belegt die Bedeutung der Thematik.

Umso überraschender ist, dass es nur wenig Forschungsarbeiten auf dem Gebiet der Qualitätssicherung von Tabellenkalkulationsanwendungen gibt. Im Folgenden wollen wir einige ausgewählte Forschungsansätze vorstellen, die sich mit dem Test von Tabellenkalkulationsanwendungen durch Endbenutzer befassen.

Zugunsten einer besseren Lesbarkeit des Textes haben wir uns dazu entschlossen, auf die Nennung der jeweils weiblichen Form zu verzichten.
In der maskulinen Form sind Frauen selbstverständlich mitgemeint.

2 Endbenutzer-gerechte Testunterstützung für Tabellenkalkulationsanwendungen

Testmethoden und -werkzeuge, die für den professionellen Softwareentwickler konzipiert wurden, können in der Regel nicht ohne Kenntnisse in der Testtheorie eingesetzt werden und sind daher für Endbenutzer ungeeignet. Für Endbenutzer ist ein hoher Grad an Systemunterstützung bei der Testdurchführung erforderlich - sowohl hinsichtlich einer leichten Bedienbarkeit des Testwerkzeugs als auch hinsichtlich des Automatisierungsgrads. Dazu bietet sich an, die Testdurchführung visuell zu unterstützen und das Testwerkzeug direkt in die Tabellenkalkulationssoftware einzubetten, damit die gewohnte Entwicklungsumgebung nicht verlassen werden muss. Alle im Folgenden beschriebenen Vorschläge zur Testunterstützung basieren auf diesem Ansatz. Unterschiedlich sind die Methoden zur weitgehenden Automatisierung der Testdurchführung und das Ziel der Testunterstützung: die Fehlererkennung oder die Fehlerbehebung.

2.1 Fehlererkennung durch datenflussorientierte Testverfahren

In ihrer Grundfunktionalität realisieren Tabellenkalkulationssysteme ein sehr einfaches deklaratives Programmierparadigma: In Zellen einer zweidimensionalen Struktur aus Zeilen und Spalten lassen sich Werte sowie funktionale Ausdrücke (Formeln) eintragen. Eine Formel verknüpft Werte einer oder mehrerer Zellen zu einem Ausgabewert. Dabei sind zirkuläre Abhängigkeiten verboten. Bei jeder Wertänderung werden die Formeln unter Berücksichtigung der bestehenden Abhängigkeiten automatisch neu berechnet. Rothermel et al. (2001) adaptiert Konzepte datenflussorientierter Testmethoden vom imperativen Programmierparadigma auf das zellenbasierte Programmierparadigma von Tabellenkalkulationssoftware. Wegen der im Vergleich zum imperativen Programmierparadigma reduzierten Komplexität ergeben sich erweiterte Möglichkeiten für eine effiziente und präzise Testunterstützung von Tabellenkalkulationsprogrammen. Aufgrund des visuellen und inkrementellen Charakters der Anwendungsentwicklung erfolgt auch die Testunterstützung visuell. Daher wird der Ansatz auch als „What You See Is What You Test“ (WYSIWYT)-Methode bezeichnet (Burnett et al. 2004).

2.1.1 What You See Is What You Test (WYSIWYT)

Das WYSIWYT-Konzept ermöglicht den systematischen, mit der Entwicklung voranschreitenden Test der im Tabellenblatt enthaltenen Formeln und Zellwerte. Dabei wird jede Zelle in den Test einbezogen, die keine reine Eingabe-Zelle ist. Eine Eingabe-Zelle zeichnet sich dadurch aus, dass sie keine Verweise auf andere Zellen enthält. Die zu testenden Zellen sind anfangs durch eine rote Umrandung gekennzeichnet. Die Farbe der Umrandung zeigt den Testfortschritt. Dabei wird zwischen verschiedenen Zuständen unterschieden: ungetestet (rot), teilweise getestet (verschiedene Abstufungen von Lila), vollständig getestet (blau). Ein Fortschrittsbalken zeigt dem Benutzer den gesamten Testfortschritt an. Für die Messung des Testfortschritts wird ein datenflussorientiertes Kriterium (definition-use test) verwendet. Jede zu testende Zelle verfügt über eine Checkbox, die drei Eingaben akzeptiert: (v), (leer) und (?). Der Zustand (v) gibt an, dass der aktuelle Wert in der Zelle durch den Benutzer validiert wurde. Die Zustände (leer) bzw. (?) geben an, dass ein Wert noch nicht überprüft wurde bzw. dass eine Überprüfung den gesamten Testfortschritt erhöhen würde. Die folgenden Abbildungen zeigen anhand einer Tabelle für Kursnoten beispielhaft die Anwendung des Verfahrens: In Abbildung 1 wurden die Werte in den Spalten 3 bis 5 der Zeile 7 durch den Benutzer bestätigt. Durch die Validierung der Kursnote B in Zeile 4, Spalte 7

erhält die benachbarte Zelle in Spalte 6 eine blaue Umrandung, weil diese zur Bestimmung der Kursnote verwendet wird (und damit eine Vorgänger-Zelle einer validierten Zelle darstellt) und keine weiteren ungetesteten Abhängigkeiten bestehen. Dagegen ist die Zelle selbst, die die Kursnote enthält, noch nicht ausreichend getestet. Dies folgt aus der Tatsache, dass für die in der Zelle befindliche IF-Anweisung noch nicht alle Bedingungen getestet wurden (Abbildung 2 zeigt diesen Zustand).

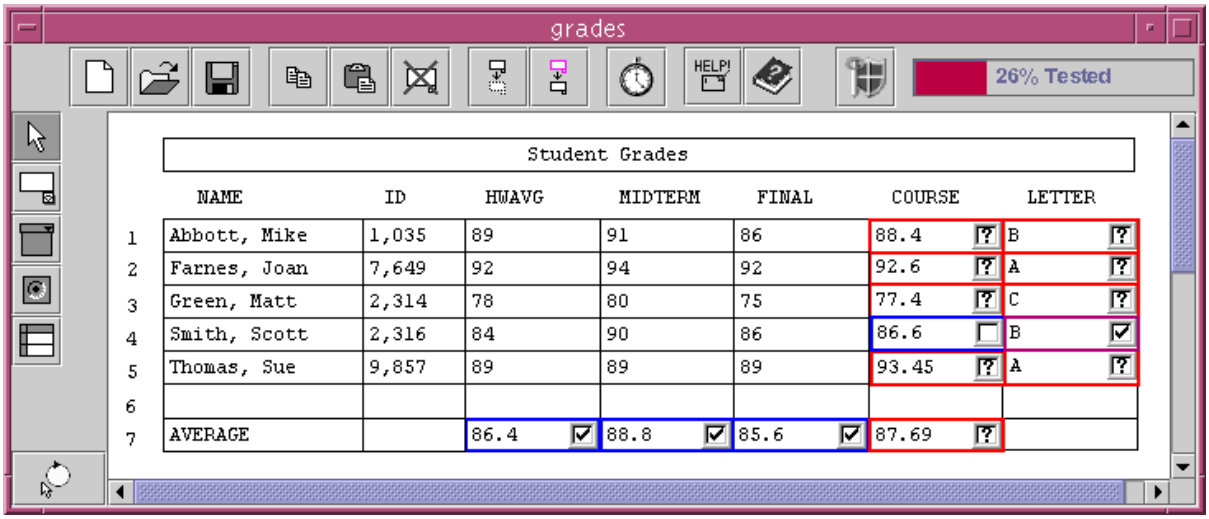


Abbildung 1: WYSIWYT-Testwerkzeug (Burnett et al. 2004)

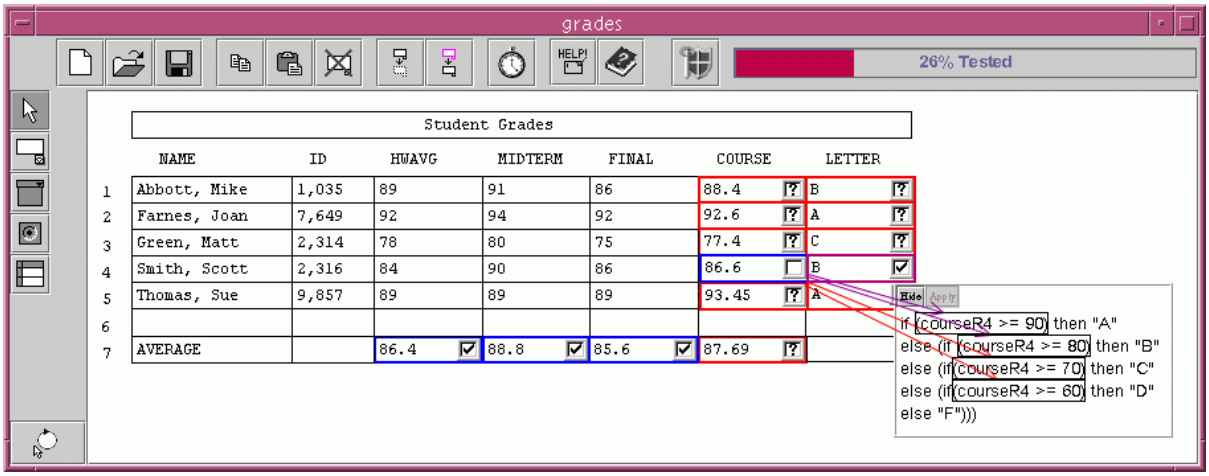


Abbildung 2: Der Benutzer lässt sich Details zu einer Zelle anzeigen (Burnett et al. 2004)

Abbildung 3 zeigt, wie das Konzept die Priorisierung bei der Suche nach fehlerhaften Formeln unterstützen kann: In diesem Beispiel hat der Benutzer fehlerhafte Werte in zwei Zellen identifiziert und diese mit (x) markiert. Die Zelle, deren Formel als erste überprüft werden sollte, wird durch eine stärkere Einfärbung gekennzeichnet. In dem Beispiel hat die Zelle in Spalte 6 (COURSE) eine höhere Priorität, da sie selbst fehlerhaft ist und zusätzlich eine Wirkung auf Spalte 7 (LETTER) hat.

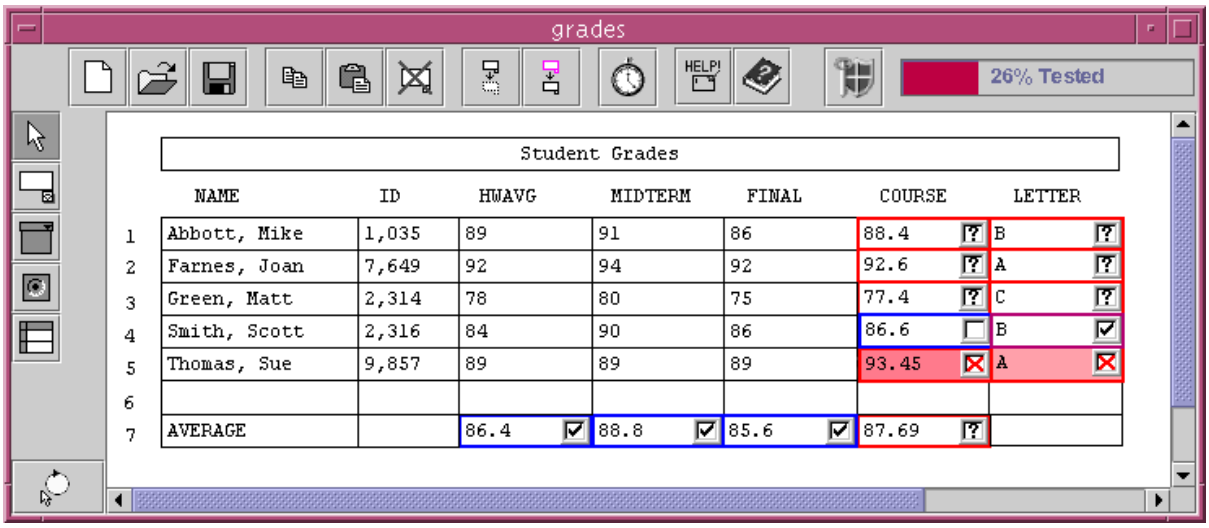


Abbildung 3: Priorisierung bei der Fehlersuche (Burnett et al. 2004)

Studien der zitierten Autoren haben gezeigt, dass die Anwendung der WYSIWYT-Methode die Effektivität und Effizienz von Test- und Debugvorgängen erhöht. Darüber hinaus animiert das Verfahren die Benutzer zum Testen der Anwendung und reduziert den Grad des Sicherheitsgefühls, dass die erstellte Anwendung frei von Fehlern ist.

Die Help-Me-Test-Funktion erweitert die WYSIWYT-Methode um eine Unterstützung des Benutzers bei der Erzeugung von Testeingaben, indem unter Berücksichtigung bestehender Datenabhängigkeiten Eingabewerte derart generiert werden, dass eine Bestätigung resultierender Ausgaben zur Erhöhung des Testfortschritts beiträgt (Fisher et al. 2002). Studien zeigen, dass die Help-Me-Test-Funktion in ca. 90 % der Fälle annehmbare Ergebnisse in unter 4 Sekunden erzielt und von den Benutzern positiv bewertet wird. Darüber hinaus sind die Benutzer bereit, längere Zeit auf Ergebnisse der Funktion zu warten und verlieren auch bei Fehlschlägen der Funktion nicht das Vertrauen in Help-Me-Test.

Als kritischer Punkt ist anzumerken, dass das Visualisierungskonzept der WYSIWYT-Methode bei umfangreichen Tabellenkalkulationsanwendungen zur Unübersichtlichkeit führen könnte. Dies gilt insbesondere für die automatische Validierung von Vorgänger-Zellen einer validierten Zelle. Dies mag zwar in vielen Fällen praktisch sein, kann aber den Benutzer auch täuschen. In Abbildung 1 könnte der Wert in Zeile 4, Spalte 6 durchaus noch falsch sein, da dieser aus den Zellen der Spalten 3 bis 5 berechnet wird.

2.1.2 Assertions

Das in der professionellen Softwareentwicklung bewährte Konzept der Assertions (Zusicherungen) in Form von Vorbedingungen, Nachbedingungen und Invarianten wurde von Burnett et al. (2003) auf die Endbenutzerprogrammierung von Tabellenkalkulationsprogrammen übertragen. Dabei werden Assertions dem Benutzer als Guards vorgestellt (in Anlehnung an die Schutzfunktion, die das Assertion-Konzept mit sich bringen soll) und als boolesche Ausdrücke über den erlaubten Wertebereich einer Zelle realisiert. Es werden zwei Arten von Assertions unterschieden: Zum einen können Assertions direkt vom Benutzer eingegeben werden (Benutzer-generierte Assertions), zum anderen gibt es Assertions, die vom System festgelegt werden (System-generierte Assertions). Die System-generierten Assertions werden durch eine Propagierung von Assertions entlang des Datenflusses erzeugt. Mit diesen zwei Ausprägungen ergeben sich drei mögliche Arten einer

Fehlerentdeckung: (1) Zwischen Benutzer- und System-generierten Assertions kann ein Konflikt bestehen (siehe Beispiel weiter unten), (2) ein Zellenwert kann eine Assertion verletzen oder (3) eine vom System generierte Assertion kann dem Benutzer falsch erscheinen. Die nachfolgende Abbildung 4 zeigt die Verwendung von Assertions. In diesem Fall wurden vom Benutzer in Zeile 5 für die Spalten 3 bis 6 Assertions festgelegt. Für Spalte 6 wird zusätzlich eine vom System generierte Assertion eingefügt. Da in diesem Beispiel eine falsche Gewichtung in der Formel für die Zelle in Zeile 5, Spalte 6 vorliegt, weichen die vom Benutzer und die vom System generierte Zusicherung voneinander ab. Dieser Konflikt wird durch eine rote Ellipse angezeigt und signalisiert dem Benutzer, dass die Formel überprüft werden sollte.

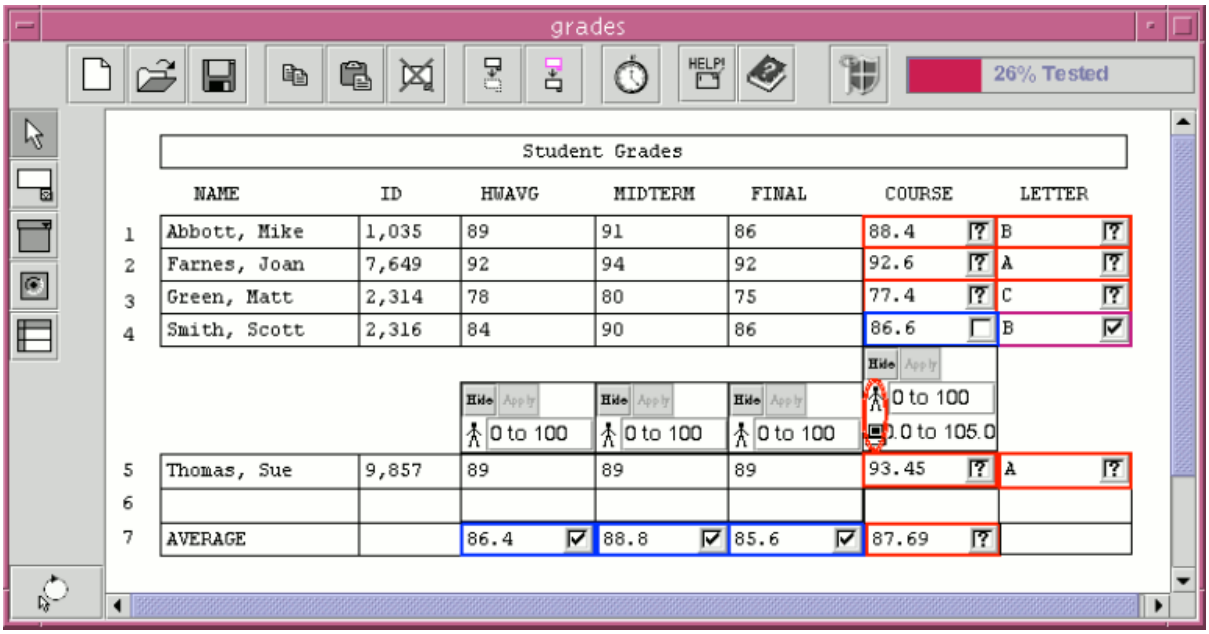


Abbildung 4: Verwendung des Assertion-Konzepts zusammen mit WYSIWYT. Die rote Umrandung gibt einen Assertion-Konflikt wieder. (Burnett et al. 2004)

In diesem Beispiel wird nur ein einzelner, relativ einfacher Ausdruck verwendet (eine Zahl zwischen 0 und 100). Die Syntax, die zur Verfügung steht, erlaubt jedoch auch wesentlich komplexere Angaben. Beispielsweise sind die Verwendung von ODER-Verknüpfungen und der Einsatz von Vergleichsoperatoren möglich. Empirische Studien belegen, dass das Assertion-Konzept von End-Benutzern angewendet werden kann und zu einem effektiveren und effizienteren Debugging-Prozess führt (Burnett et al. 2003).

Wilson et al. (2003) schlägt die sog. Surprise-Explain-Reward-Strategie vor, um die Benutzer zur Verwendung des Assertion-Konzepts zu animieren. Um dieses Ziel zu erreichen, wird ein dreistufiges Konzept verwendet. Als erstes wird der Benutzer überrascht, indem seine Annahme über das Verhalten der Tabellenkalkulationsanwendung gestört wird. Anschließend wird auf die so entstandene Informationslücke des Benutzers durch Hinweise und Erklärungen eingegangen. Abschließend wird der Benutzer durch den Belohnungseffekt, welchen er durch die Fehlerbehebung erfährt, zur Verwendung von Assertions animiert. Der Überraschungseffekt wird durch Assertions ausgelöst, die von der Help-Me-Test-Funktion generiert werden (HMT-Assertions). Wilson et al. zielen mit dieser Strategie darauf ab, dass der Benutzer überrascht reagieren wird, da ihm Assertions unbekannt sind. Gleichzeitig wird eine Informationslücke aufgedeckt, die den Benutzer animieren wird, Nachforschungen anzustellen, um die Informationslücke zu schließen. Durch die Verwendung von Tooltips erhält der Benutzer Informationen über das ihm bis dahin unbekannte Konzept.

Die Belohnung erfolgt durch die verschiedenen Arten der Fehlerbehebung, die mit Hilfe des Einsatzes der Assertions erreicht wurde. Abbildung 5 zeigt für Zeile 5 generierte HMT-Assertions und einem Tooltip:

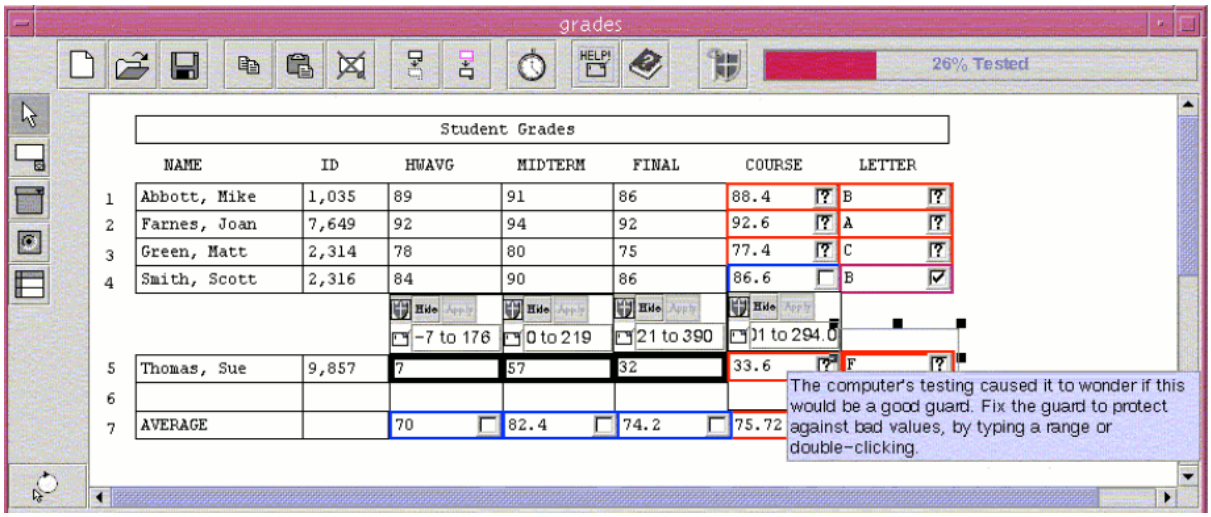


Abbildung 5: Verwendung der Suprise-Explain-Reward-Strategie (Burnett et al. 2004)

Neben der Vorstellung des Konzepts haben Wilson et al. auch die Ergebnisse einer Studie zu Akzeptanz und Nutzen des Konzepts veröffentlicht: Für die Studie wurde 16 Studenten aus dem Bereich Wirtschaftswissenschaften die Aufgabe gegeben, zwei Tabellenkalkulationsanwendungen auf Fehler zu überprüfen. Dabei wurde die Funktionsweise von WYSIWYT und Help-Me-Test erläutert, das Assertion-Konzept jedoch verschwiegen. Von allen Teilnehmern verwendeten 94 % Assertions in wenigstens einer der beiden Aufgaben und 87 % verwendeten Assertions in beiden Aufgaben. Durchschnittlich wurden von jedem Teilnehmer 18 Assertions eingegeben (insgesamt 279 von 15 Teilnehmern). Aufgrund der Tatsache, dass 87 % der Teilnehmer in beiden Fällen mit Assertions gearbeitet haben, lässt sich auf einen Erfolg des Belohnungssystems schließen. Darüber hinaus begannen die Benutzer bei der Bearbeitung der zweiten Aufgabe wesentlich früher mit der Nutzung von Assertions.

2.2 Fehlererkennung durch Unit-Tests

Während die zuvor dargestellten datenflussorientierten Testverfahren die Abhängigkeitsstruktur der miteinander verknüpften Zellen zur Grundlage haben, versuchen Unit-Tests Fehler dadurch zu lokalisieren, dass potentielle Inkonsistenzen von Zellenwerten und Formeln auf der Basis von Einheiten (Units) ermittelt werden, gemäß der Redewendung, dass Äpfel mit Birnen vergleichen (im Englischen: to compare apples and oranges) bedeutet, Dinge miteinander zu vergleichen, die gar nicht vergleichbar sind.

2.2.1 UCheck

Die Idee der Fehlerlokalisierung durch Unit-Vergleiche geht auf einen Beitrag von Burnett und Erwig (2002) mit dem Titel „Adding Apples and Oranges“ zurück. Sie wurde durch Arbeiten von Abraham und Erwig (2004) um die sog. Headerinferenz erweitert und als Testsystem UCheck implementiert. Das zugrunde liegende Konzept ist an die statische Typprüfung angelehnt. Einheiten sind vom Entwickler der Anwendung definierte Bezeichner

für Kollektionen gleichartiger Zellen wie z. B. Zeilen- oder Spaltenüberschriften. Einheiten sind also Zellenwerte, die den Wertetyp von anderen Zellen beschreiben. Derartige Bezeichner werden Header genannt. Potentiell kann jeder Wert in einer Zelle, die nicht leer ist, eine Einheit definieren. Dies bedeutet jedoch nicht, dass jeder Wert auch als Einheit verwendet wird (beispielsweise die Bezeichnung „Total“ zum Abschluss einer Zeile oder Spalte). Die Besonderheit von UCheck ist die automatische Identifizierung der Header mit Hilfe der Headerinferenz. Sind die Header identifiziert, lassen sich mit Hilfe der sog. Unitinferenz Fehler in der Tabellenkalkulationsanwendung lokalisieren.

Zunächst soll die Unitinferenz anhand eines Beispiels beschrieben werden.

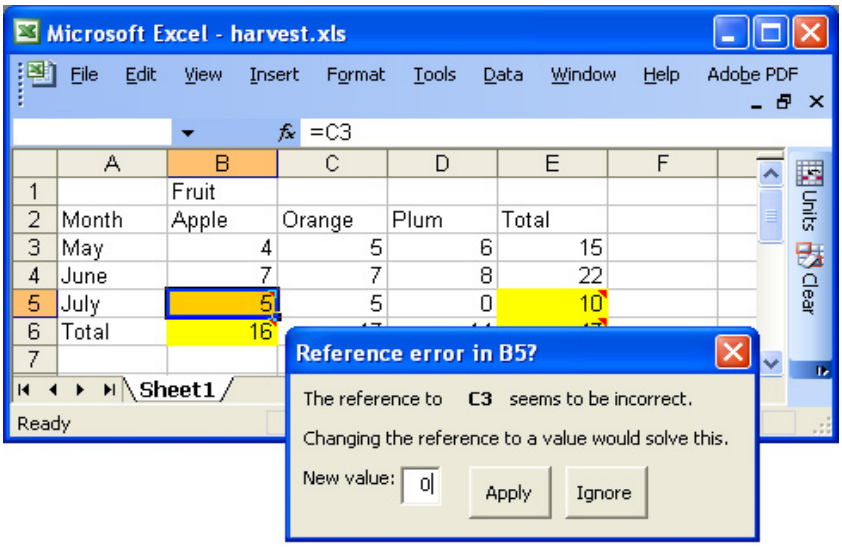


Abbildung 6: Ein einfaches Beispiel für die Verwendung von UCheck (Abraham, Erwig 2005a)

Es werden drei Arten von Einheiten unterschieden: abhängige Einheiten, UND-Einheiten und ODER-Einheiten. Wenn der Wert v einer Zelle c eine Einheit darstellt, die wiederum einer Einheit u zugehörig ist, dann wird die Einheit von c als abhängige Einheit u[v] bezeichnet. Eine solche Generalisierung würde für die Zelle B3 in Abbildung 6 als Fruit[Apple] notiert werden. Das Beispiel zeigt, dass Werte in Zellen sich gleichzeitig mehreren Einheiten zuordnen lassen. Der Wert in Zelle B4 ist gleichzeitig mit Month[May] und Fruit[Apple] assoziiert. Dies lässt sich in Form einer UND-Einheit notieren: Fruit[Apple]&Month[May]. ODER-Einheiten lassen sich anhand der Summe in Zelle E3 erläutern. Die Einheiten der Zellen B3, C3 und D3 stellen Ausprägungen des Headers Fruit dar, beziehen sich aber alle auf den Monat May. Dies lässt sich in Form einer ODER-Einheit wie folgt notieren: Fruit[Apple]&Month[May]|Fruit[Orange]&Month[May]|Fruit[Plum] &Month[May] oder kompakter unter Ausnutzung des Distributiv- und Kommutativgesetzes: Month[May] & (Fruit[Apple|Orange|Plum]).

Grundlage der Fehlererkennung bildet das Konzept wohl-geformter Einheiten. Nur Einheiten, die einer der folgenden vier Regeln entsprechen, sind wohl-geformt und gelten als fehlerfrei:

1. Jeder Wert, zu dem es keinen Header gibt, ist wohl-geformt. In dem oben gezeigten Beispiel ist das für die Zelle B1 (Fruit) der Fall.
2. Jede abhängige Einheit ist wohl-geformt, also jede Einheit, die einem Header zugehörig ist wie bspw. Fruit[Apple].
3. UND-Verknüpfungen zwischen verschiedenartigen Headern (genauer: zwischen Headern ohne gemeinsamen Vorgänger) sind wohl-geformt. Es ist also bspw.

Fruit[Apple]& Month[May] eine wohl-geformte Einheit, während Fruit[Apple]& Fruit[Orange] fehlerhaft ist.

4. ODER-Verknüpfungen, bei denen alle Header bis auf den innersten übereinstimmen, sind wohl-geformt. So ist Fruit[Apple]|Fruit[Orange] wohl-geformt, dagegen Fruit[Apple[Green]]|Fruit[Orange] fehlerhaft.

An Abbildung 6 lässt sich zeigen, wie anhand der Regeln Fehler identifiziert werden können: Wie zu sehen ist, enthält die Zelle B5 einen Verweis auf die Zelle C3. Für B5 ergibt sich als Einheit Fruit[Apple]&Month[July], für C3 ergibt sich Fruit[Orange]&Month[May]. Aufgrund des Verweises kombiniert das System die beiden Ausdrücke zu Fruit[Apple]&Month[July]& Fruit[Orange]&Month[May]. Nach der dritten Regel ist dieser Ausdruck ungültig, da hier dieselben Header mit & verknüpft werden. Das System wird nach der Aktivierung von UCheck darauf hinweisen, dass die Referenz unter Umständen fehlerhaft ist (mögliche Folgefehler werden ebenfalls durch einen helleren Zellhintergrund angezeigt). Würde für die Summe in Zelle B6 versehentlich ein Intervall von B2:B4 anstelle von B3:B5 angegeben, so würde auch dieser Fehler vom System bemerkt werden. Für die Zelle B6 würde sich in diesem Fall eine nach der vierten Regel ungültige ODER-Verknüpfung Fruit|Month[May|June]&Fruit[Apple] ergeben. Zudem würde der Wert in E6 als Folgefehler erkannt werden.

Für die Durchführung der Unitinferenz müssen Informationen über die Header vorliegen. Die Angabe von Headerinformationen den Benutzern zu überlassen, wird dabei von den Autoren als wenig erfolgversprechend angesehen, da dieser i. d. R. nicht gewillt sein wird, zusätzliche Angaben zu den Headern in der Tabellenkalkulationsanwendung zu machen. Es wird also eine automatische Headerinferenz benötigt, welche die für die Unitinferenz benötigten Headerinformationen zur Verfügung stellt. Die benötigten Informationen werden aus einer räumlichen Analyse des Tabellenblatts gewonnen. Dafür werden die Zellen der Anwendung in folgende Kategorien unterteilt:

1. Header: Zellen zur Beschreibung von Daten,
2. Footer: Zellen, die typischerweise am Ende von Zeilen oder Spalten zu finden sind und Aggregationsformeln (wie z. B. die Summenformel) enthalten,
3. Core: Zellen, die Daten enthalten,
4. Filler: Zellen (wie z. B. leere Zellen) die zur Formatierung verwendet werden.

Um die Klassifizierung vorzunehmen, werden verschiedene Algorithmen angewendet: Jeder Klassifizierung wird ein Wert von 1 bis 10 zugeordnet (Vertrauens-Level). Für den Fall, dass Uneinigkeit über die Art der Zelle herrscht, werden die Werte für jede Kategorie summiert und der höchste Wert entscheidet, wie die Zelle zu klassifizieren ist. Beispielsweise werden über den sog. Footer-to-Core-Expansion-Algorithmus alle Zellen mit Aggregationsformeln gesucht und mit einem niedrigen Vertrauens-Level als Footer klassifiziert. Die in den Aggregationszellen referenzierten Zellen werden mit einem höheren Vertrauens-Level als Core-Zellen klassifiziert. Die direkt benachbarten Zellen werden ebenfalls als Core-Zellen klassifiziert, falls sie vom selben Datentyp sind, ansonsten als Filler (falls sie leer sind) oder als Header. Nachdem die Klassifizierung abgeschlossen ist, werden den Zellen die entsprechenden Header zugeordnet. Dabei erfolgt die Zuordnung von innen nach außen, also von der untersten Ebene der Hierarchie zur oberen, und es wird jeweils der am nächsten links bzw. oben liegende Header herangezogen. In dem oben beschriebenen Beispiel würden der Zelle in B3 zum Beispiel als Erstes die Header Apple und May zugeordnet werden.

Von Abraham und Erwig (2007) durchgeführte Tests mit insgesamt 28 unterschiedlichen Tabellenkalkulationsanwendungen (darunter einige mit einem unüblichem Layout) zeigen, dass das System zur Fehlersuche geeignet ist. Teilweise wurden zwar Header, die weiter oben in der Hierarchiestufe liegen, vom System falsch interpretiert. Nach Angaben der Autoren führte dies jedoch nicht zu Fehldeutungen. Eine weitere Evaluation, bei der UCheck und die WYSIWYT-Methode getestet wurden, hat gezeigt, dass Benutzer die automatische Zuordnung der Header durch UCheck positiv bewerten. Trotz dieser positiven Beurteilungen durch die Benutzer bleibt die Headerinferenz eine Achillesferse von UCheck. Wenn der Entwickler andere Formatierungen realisiert als der Inferenzmechanismus unterstellt, werden falsche Header abgeleitet.

2.2.2 Slate

Ein weiterer Ansatz für die Fehlerlokalisierung durch Unit-Tests ist die von Coblenz vorgestellte Spreadsheet Language for Accentuating Type Errors (Slate) (Coblenz 2005 und Coblenz et al. 2005). Slate verzichtet auf die Realisierung einer Headerinferenz, da diese bei unerwartetem Layout fehlerhaft sein kann. Zudem können die Units von UCheck sehr komplex werden und sind daher nicht für eine Auswertung durch den Benutzer vorgesehen. Aus diesen Gründen wird ein neues Einheiten-Konzept eingeführt, welches direkt vom Benutzer zur Fehleridentifikation herangezogen werden kann. Slate verlangt vom (bzw. erlaubt dem) Benutzer die Spezifikation von sog. „objects of measurement“ (z. B. 312 lb. of apples) in Form von Units und Labels. Durch automatische Propagierung der Labels können Fehlerhinweise gegeben werden. Abbildung 7 zeigt die Wirkungsweise: In der Tabelle soll der mit Äpfeln und Orangen erzielte Umsatz berechnet werden. In Zeile 6 wurde jedoch fälschlicherweise mit dem Preis für Äpfel multipliziert. Der Fehler wird dem Benutzer durch das Label (apples, oranges) angezeigt.

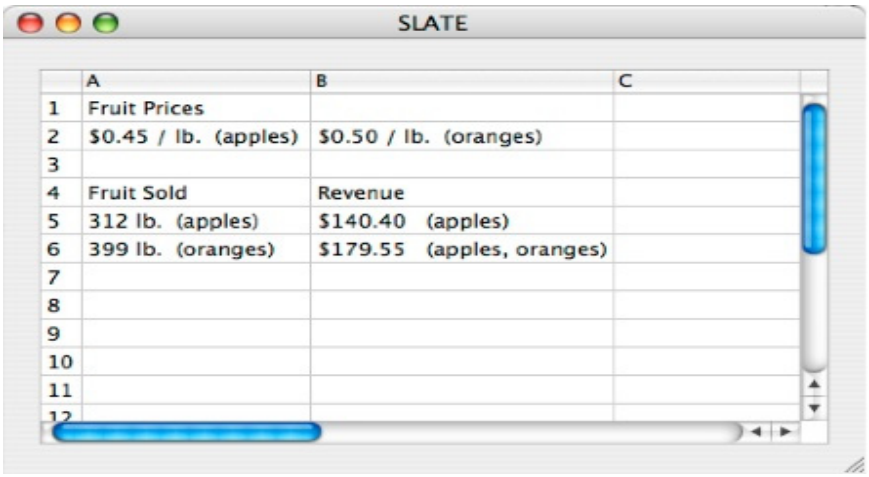


Abbildung 7: Fehlerhafte Multiplikation von Einheiten in Slate (Coblenz et al. 2005)

Ein Ausdruck wie in Zelle B5 verfügt über drei Attribute: Value (144.40), Unit ($) und Label (apples). Das Label dient der Auszeichnung oder Kategorisierung eines Values und seiner Unit. Grundlage für die Ableitung von Labels entsprechend dem oben gezeigten Beispiel ist der Label-Kontext. Der Label-Kontext repräsentiert die hierarchische Struktur der Labels einer Tabellenkalkulationsanwendung. Formal betrachtet handelt es sich beim Label-Kontext um einen gerichteten azyklischen Graphen Λ mit als Concepts bezeichneten Knoten C und Kanten E, also $\Lambda = (C,E)$. Die Wurzel eines jeden Label-Kontexts heißt Something und bezeichnet das leere Label {}. Ein Label λ besteht aus einer Menge von Knoten $c \in C$, die

keinen gemeinsamen Pfad haben. Für den nicht notwendigerweise direkten Nachfolger c2 eines Knotens c1 schreibt man descendent(c1, c2). Zur Illustration sei der Label-Kontext aus Abbildung 8 betrachtet. Gültige Labels wären nach der Definition z. B. (apples) oder (apples, September). Dagegen wäre (fruit, apples) kein gültiges Label, da fruit und apples auf einem gemeinsamen Pfad in Λ liegen.

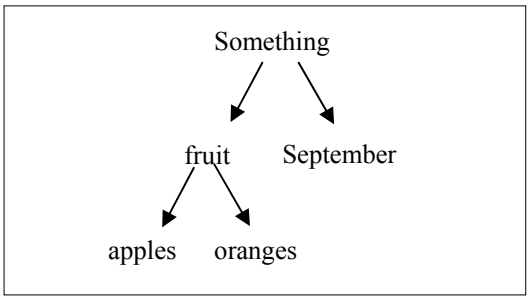


Abbildung 8: Ein einfacher Label-Kontext

Um das Zustandekommen der Labels in dem oben gezeigten Beispiel zu verdeutlichen, werden im Folgenden die Addition und Subtraktion sowie die Multiplikation und Division von Ausdrücken in Slate erläutert:

Addition und Subtraktion von Ausdrücken

Auf Ausdrücken mit identischen Einheiten lassen sich Addition und Subtraktion anwenden. Um das Label des Ergebnisses einer solchen Verknüpfung zu bestimmen, wird die Funktion add-sub-labels (λ1, λ2) verwendet. Die Funktion ist rekursiv und bricht ab, sobald das Ergebnis {} ist. In Pseudocode sieht die Funktion folgendermaßen aus:

```
fun add-sub-labels (λ1, λ2) =
   if λ1 = {} then {}
   else if  λ2 = {} then {}
   else
      let intersection =
        intersection-with-paths (λ1, λ2)
      in
        intersection ∪
        add-sub-labels (parents (λ1\intersection),
                        parents (λ2\intersection))
end
```

Abbildung 9: Funktion add-sub-labels

Die Funktion intersection-with-paths (λ1, λ2) liefert folgende Menge von Knoten: {n | (n ∈ λ1 and ∃n' ∈ λ2:descendent (n', n)) or (n ∈ λ2 and ∃n' ∈ λ1:descendent (n', n))} Die Funktion parents liefert für die Übergabe einer Menge λ die Eltern der Elemente von λ. Sei folgender Ausdruck gegeben:

5 lbs. (apples, September) + 2 lbs. (oranges, September) = 7 lbs. (fruit, September).

Dann ist λ1 = {apples, September} und λ2 = {oranges, September}. Da es sich weder bei λ1 noch bei λ2 um das leere Label {} handelt, wird intersection-with-paths (λ1, λ2) aufgerufen und liefert die Menge {September}. Für den zweiten Aufruf von add-sub-labels (parents (λ1\intersection), parents (λ2\intersection)) liefert die Funktion parents für λ1 und λ2 jeweils

die Menge {fruit} (da der Vorgänger von September in diesem Beispiel Something ist, liefert die Funktion für September {}). Das Ergebnis ist {September} ∪ {fruit}, da die Funktion nach dem dritten Aufruf abbricht, denn es ist intersection = {September, fruit} und add-sub-labels (parents (λ1\intersection), parents (λ2\intersection)) = {}.

Multiplikation und Division von Ausdrücken

Bei der Multiplikation und Division werden die Labels durch folgende Funktion ermittelt:

```
fun mult-div-labels (λ1, λ2) =
  eliminate-ancestors (λ1 ∪ λ2)
```

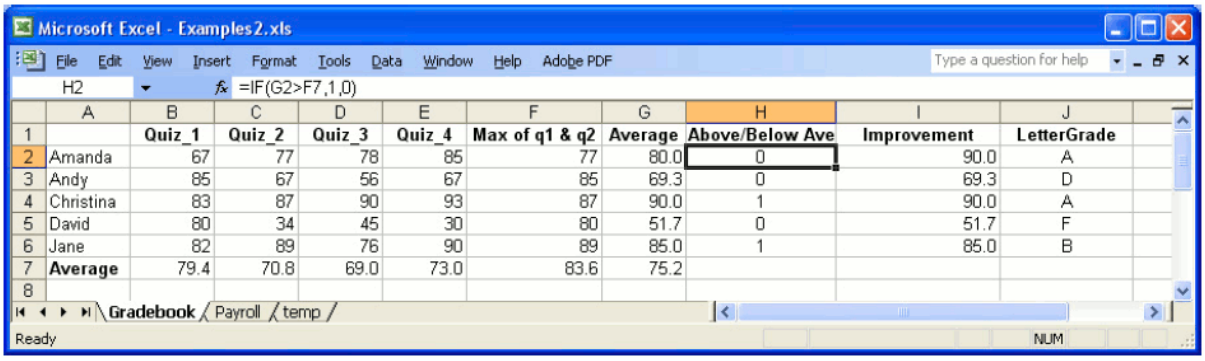
Abbildung 10: Funktion mult-div-labels

Die Funktion eliminate-ancestors (λ1 ∪ λ2) entfernt alle Elemente aus λ1 ∪ λ2, die Vorgänger eines anderen Elements aus λ1 ∪ λ2 sind, also:
eliminate-ancestors (λ1 ∪ λ2) = λ1 ∪ λ2 \ {n ∈ λ1 ∪ λ2 | ∃n' ∈ λ1 ∪ λ2: descendent(n, n')}.
Beispielsweise erhält man für den Ausdruck 2 qty. (apples) * 0.50 $ / qty. (fruit) das Ergebnis $1.00 (apples).

Empirische Studien über die Benutzerakzeptanz und die Effektivität von Slate liegen nicht vor. Leider finden sich auch keine konkreten Angaben darüber, wie die Benutzerinteraktion mit dem Testsystem gestaltet ist, wie also z. B. die Eingabe von Spezifikationen für die units of measurement erfolgen soll.

2.3 Fehlerbehebung mit GoalDebug

Während die bisher beschriebenen Methoden den Fokus auf das Auffinden von Fehlern legen, macht GoalDebug dem Benutzer konkrete Korrekturvorschläge. GoalDebug wurde als Proof of Concept von Abraham und Erwig (2005) vorgestellt und später als Erweiterung von Microsoft Excel implementiert (Abraham, Erwig 2008). Das System basiert auf dem Algorithmus der sog. Changeinferenz. Ausgangspunkt ist ein vom Benutzer vorgegebener richtiger Wert oder Wertebereich für eine als fehlerhaft erkannte Zielzelle. Über die Changeinferenz werden vom System Vorschläge für eine Änderung der Formel in der Zielzelle gemacht oder für die Zellen, die direkt auf die Zielzelle einwirken. Dabei sind die Änderungsvorschläge dergestalt, dass der Wert der Zielzelle in den vorgegebenen Bereich fällt. Akzeptiert der Benutzer keinen der Änderungsvorschläge, wird die Changeinferenz rückwärts durch die Zellabhängigkeitsstruktur propagiert. Die Änderungsvorschläge werden über Heuristiken ermittelt. Beispielsweise haben Änderungen von Zellreferenzen auf benachbarte Zellen eine höhere Priorität als auf entfernte Zellen. Ein einfaches Beispiel soll dies verdeutlichen. Abbildung 11 zeigt ein Tabellenblatt, in dem die Kursnoten von Studenten eingetragen wurden: Die Zellen G2 bis G6 werden jeweils mit ihrem in G7 ermittelten Durchschnittswert verglichen. Für den Fall, dass der Wert einer Zelle in Spalte G größer als der Durchschnittswert ist, wird in der entsprechenden Zelle in Spalte H der Wert 1 notiert, andernfalls der Wert 0.



	A	B	C	D	E	F	G	H	I	J
1		Quiz_1	Quiz_2	Quiz_3	Quiz_4	Max of q1 & q2	Average	Above/Below Ave	Improvement	LetterGrade
2	Amanda	67	77	78	85	77	80.0	0	90.0	A
3	Andy	85	67	56	67	85	69.3	0	69.3	D
4	Christina	83	87	90	93	87	90.0	1	90.0	A
5	David	80	34	45	30	80	51.7	0	51.7	F
6	Jane	82	89	76	90	89	85.0	1	85.0	B
7	Average	79.4	70.8	69.0	73.0	83.6	75.2			
8										

Abbildung 11: Ausgangsbeispiel für die Verwendung von GoalDebug (Abraham, Erwig 2005)

Wird ein fehlerhafter Wert v in einer Zelle c bemerkt, wie dies in dem Beispiel für H2 der Fall ist, hat der Benutzer die Möglichkeit, den eigentlich erwarteten Wert oder Wertebereich für die Zelle mit Hilfe eines Vergleichsoperators $\omega \in \{<, <=, =, >=, > \}$ anzugeben (vgl. Abbildung 12):

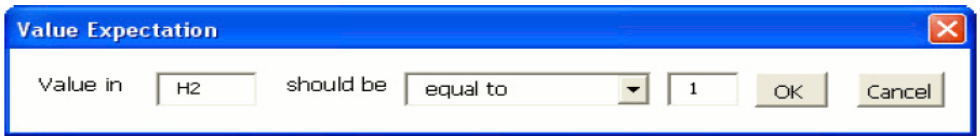


Abbildung 12: Angabe des erwarteten Werts für Zelle H2 (Abraham, Erwig 2005)

Auf Grundlage dieser Angabe liefert GoalDebug Vorschläge für die Abänderung der Formel. In diesem Fall wurde in der Formel anstelle von G7 auf F7 referenziert. Der richtige Korrekturvorschlag wird an 2. Stelle präsentiert (vgl. Abbildung 13).

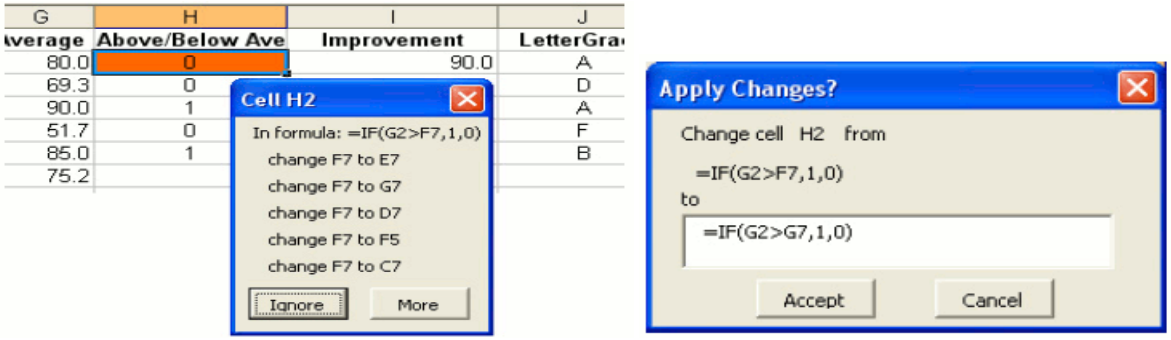


Abbildung 13: Vom System ermittelte Änderungsvorschläge und Anwendung von Vorschlag 2 (Abraham, Erwig 2005)

Neben der Funktionsweise von GoalDebug zeigt das verwendete Beispiel gleichzeitig die Einschränkungen des Konzepts auf: Als Erstes muss ein Fehler durch den Benutzer entdeckt werden. Würden z. B. anstelle von sechs Datensätzen die Daten von 100 Studenten vorliegen, wäre ein solcher Fehler dem Benutzer unter Umständen nicht aufgefallen. Dementsprechend ist eine Kombination mit einem weiteren Konzept zur Auffindung von Fehlern empfehlenswert. Abraham und Erwig schlagen dafür eine Integration mit der WYSIWYT-Methode vor. Untersuchungen zur Effektivität des Systems haben gezeigt, dass durch die Integration eines Testfallgenerators der Anteil erfolgreicher Korrekturen eingepflanzter Fehler von ca. 50% auf ca. 90% erhöht werden konnte (Abraham, Erwig 2008).

3 Schlussbemerkung

Kommerzielle Tabellenkalkulationssysteme wie Microsoft Excel verfügen zwar zunehmend über Funktionen, die eine Unterstützung bei der Fehlererkennung bieten, diese reichen aber nicht aus, um beispielsweise fehlerhafte Referenzen, nicht aktualisierte Aggregatfunktionen oder (semantische) Fehler in Formeln aufzudecken. Die vorgestellten Forschungsarbeiten versuchen, diesen Mangel zu beseitigen.

Selbstverständlich stellen Tests nur einen Teil von möglichen Maßnahmen zur Qualitätssicherung von Tabellenkalkulationsanwendungen dar. Sie stehen am Ende des Entwicklungsprozesses. Konstruktive Maßnahmen in den vorgelagerten Entwicklungsphasen sind ebenso von erheblicher Bedeutung für die Qualität. Einige Arbeiten wie die von Raffensperger (2003) beschäftigen sich mit der Modellbildungsphase aus Endbenutzersicht. Auch IT-Werkzeuge, die die Modellbildung unterstützen, wurden entwickelt. Beispielsweise ermöglicht ClassSheets die objektorientierte Modellierung von Tabellenkalkulationsanwendungen (Bals et al. 2007). Aber wie für die Testthematik gilt auch hier, dass es nur wenige Forschungsarbeiten gibt, die sich speziell mit Tabellenkalkulationssoftware beschäftigen. Es bleibt zu hoffen, dass weitere Anstrengungen von der Wissenschaft unternommen werden, diese für die Praxis überaus bedeutende Thematik voranzubringen.

Literaturverzeichnis

Abraham, R.; Erwig, M. (2004): Header and Unit Inference for Spreadsheets Through Spatial Analyses. IEEE Symposium on Visual Languages and Human-Centric Computing, September 2004, Rome, Italy, S. 165−172. Los Alamitos, Calif.: IEEE Computer Society.

Abraham, R.; Erwig, M. (2005): Goal-Directed Debugging of Spreadsheets. Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing, Sept. 2005, S. 37-44. Washington, DC, USA: IEEE Computer Society.

Abraham, R.; Erwig, M. (2005a): How to communicate unit error messages in spreadsheets. In: ACM SIGSOFT Software Engineering Notes, Jg. 30, H. 4, S. 1−5.

Abraham, R.; Erwig, M. (2007): UCheck: A spreadsheet type checker for end users. In: Journal of Visual Languages and Computing, Jg. 18, H. 1, S. 71−95.

Abraham, R.; Erwig, M. (2008): Test-driven goal-directed debugging in spreadsheets. IEEE Symposium on Visual Languages and Human-Centric Computing, September 2008, S. 131-138.

Bals, J. C.; Christ, F.; Engels, G.; Erwig, M. (2007): ClassSheets - model-based, object-oriented design of spreadsheet applications. In: Journal of Object Technology, Jg. 6, H. 9, S. 383−398.

Burnett, M. M.; Cook, C. R.; Pendse, O.; Rothermel, G.; Summet, J.; Wallace, C. (2003): End-user software engineering with assertions in the spreadsheet paradigm. Proceedings of the 25th International Conference on Software Engineering, May 2003, Portland, Oregon, USA, S. 93- 103. Los Alamitos, Calif.: IEEE Computer Society.

Burnett, M. M.; Cook, C. R.; Rothermel, G. (2004): End-User Software Engineering. In: Communications of the ACM, Jg. 47, H. 9, S. 53−58.

Burnett, M. M.; Erwig, M. (2002): Adding Apples and Oranges. Proceedings of the 4th International Symposium on Practical Aspects of Declarative Languages, January 2002, Portland, OR, USA, Lecture Notes In Computer Science, Vol. 2257, S. 173−191. London, UK: Springer-Verlag.

Coblenz, M. J. (2005): Using Objects of Measurement to Detect Spreadsheet Errors. Technical Report CMUCS-05-150. School of Computer Science, Carnegie Mellon University, Pittsburgh, Pennsylvania, USA, July 2005.

Coblenz, M. J.; Ko, A. J.; Myers, B. A. (2005): Using objects of measurement to detect spreadsheet errors. IEEE Symposium on Visual Languages and Human-Centric Computing, September 2005, Dallas, Texas, S. 314- 316. Los Alamitos, Calif.: IEEE Computer Society.

Fisher, M.; Cao, M.; Rothermel, G.; Cook, C. R.; Burnett, M. M. (2002): Automated test case generation for spreadsheets. Proceedings of the 24th International Conference on Software Engineering. May 2002, Orlando, Florida, S. 141- 151. New York, NY: Association for Computing Machinery.

O'Beirne, P. (2003, 2004): Spreadsheet mistakes - news stories. Online verfügbar unter http://www.eusprig.org/stories.htm, zuletzt geprüft am 23.06.2009.

Powell, S. G.; Baker, K. R.; Lawson, B. (2009): Impact of Errors in Operational Spreadsheets. In: Decision Support Systems, Jg. 47, H. 2, S. 126−132.

Power, D. J.: A Brief History of Spreadsheets. DSSResources.COM. Online verfügbar unter http://dssresources.com/history/sshistory.html, version 3.6, 08/30/2004. Photo added September 24, 2002., zuletzt geprüft am 15.6.2009.

Raffensperger, J. F. (2003): New Guidelines for Spreadsheets. In: International Journal of Business and Economics, Jg. 2, H. 2, S. 141−154.

Rothermel, G.; Burnett, M.; Li, L.; DuPuis, C.; Sheretov, A. (2001): A methodology for testing spreadsheets. In: ACM Transactions on Software Engineering and Methodology, Jg. 10, H. 1, S. 110−147.

Scaffidi, C.; Shaw, M.; Myers, B. (2005): Estimating the numbers of end users and end user programmers. In: IEEE Symposium on Visual Languages and Human-Centric Computing, S. 207−214.

Wilson, A.; Burnett, M. M.; Beckwith, L.; Granatir, O.; Casburn, L.; Cook, C. et al. (2003): Harnessing curiosity to increase correctness in end-user programming. Proceedings of the SIGCHI conference on Human factors in computing systems, Ft. Lauderdale, Florida, USA, 2003, S. 305 - 312. New York, NY, USA: ACM.

QUALITY RISKS OF RISK MANAGEMENT SOFTWARE FOR BANKING COMPANIES

Heinz-Dieter Knöll, Robert Moreton, Alexander Schulz-Sacharow[1]

Abstract

Over the last decades, ERP (Enterprise Resource Planning) systems have become the de facto standard for translating business best practices into corporate information technology. Thus, integrated, configurable "off-the-shelf" solutions that can be readily implemented offer an effective way to optimise business processes. However, the success of standard ERP software implementation projects heavily depends on their quality assurance which, in turn, is influenced by the employed implementation life-cycle. Case studies at two major German banks which implemented complex standard software systems for risk controlling illustrate how the main belongings and peculiarities of standard ERP software as well as life-cycle-inherent weaknesses affect the testing strategy.

The evaluation results strikingly demonstrate the strong need for a testing strategy that supports the selection of test cases, quantification of test results, and estimation of the probability that the system implements its functional specification correctly. Therefore this paper includes a preliminary analysis of statistical approaches to testing. In addition, the life-cycle-inherent software quality management as defined by the SEI Capability Maturity Model (CMM) is assessed. The assumption is that improved implementation process maturity or capability leads to a decreased number of observed faults during test.

Keywords: quality assurance, standard software, implementation life-cycles, Basel II.

[1] Knöll, Heinz-Dieter, Leuphana Universität Lüneburg, Volgershall 1, Raum VA122, 21339 Lüneburg, Germany, knoell@uni.leuphana.de

Moreton, Robert, University of Wolverhampton, Technology Centre (MI Building), City Campus south, Wulfruna Street, Wolverhampton, WV1 1LY, United Kingdom, r.moreton@wlv.ac.uk

Schulz-Sacharow, Alexander, Abendrothsweg 50, 20251 Hamburg, Germany, a.schulz-sacharow@gmx.de

1 INTRODUCTION

Projects for the replacement of companies' old software systems by standard ERP systems require huge amounts of resources, staff, and money. Decisions for or against a new software system are mainly based on the suitability of the software for the business domain and on the total costs of implementation, covering software licenses, company staff, and consultants. Software suppliers reacted to these market demands and developed life-cycle implementation models for the rapid implementation of their software and, in addition, reference models for the different business areas and industrial branches (Knoell et al. 2004, p. 271).

In literature, much attention has been given to the analysis and design phases of ERP software development and their support by techniques and tools. Concerning the evaluation and improvement of implementation models only little research can be found. Knoell et al. (2001, pp. 129-169) give an overview of the research in this area but concentrate on the approaches' capability for business process optimization. No research can be found that evaluates conventional quality assurance methods with regard to the corresponding phases of ERP implementation life-cycle models.

The purpose of this paper is to examine how typical implementation life-cycles support quality assurance of standard ERP software systems. Here we distinguish product quality from process quality. Product quality assurance means assessment of the correctness of an implementation built to the requirements specification, based on observed test data. For process quality assurance, which is in the responsibility of the project management, a growing number of maturity models is available.

Two case studies are discussed in this context; both based on implementations of a Basel II standard software solution at two major German banks. Basel II summarizes the regulatory requirements for the calculation of credit risk and risk weight assets in banks – an enormous analytical task affecting almost every business area of a bank and vital for its stability.

In order to derive a quality framework for implementation life-cycles that integrates both product quality assurance and process quality management, the influence of process quality factors on the parameters of product quality models has to be explored. Therefore, this paper describes a preliminary analysis of testing strategies and the potential application of statistical methods like the binomial model, reliability growth models, and coverage designs.

2 STANDARD ERP SOFTWARE SYSTEMS

The term Enterprise Resource Planning (ERP) characterizes the entrepreneurial task to organize business processes effectively and efficiently. ERP aims at the optimal planning and control of all available resources of an enterprise (Curran et al. 1998). Software systems for ERP typically support and integrate all important functions of administration, disposition, and

steering into one total system (Stahlknecht et al. 1999, p. 344) and are tailored to the demands of specific markets such as banking, healthcare, or public administration (Botella et al. 2003, p. 226).

Standard ERP software systems consist of computer program components that can be readily implemented and that – when interfaced to each other – cover a complete business area (Oberniedermaier 1999, p. 46, Meyers et al. 2001). Within such an integrated system, all modules are interdependent and able to communicate with each other. A central database avoids redundancies and provides actual information. Changes immediately become active throughout the system (Rolf 1998, p. 99). These program systems can be customized, that means adapted to the specific requirements of an enterprise, by configuration (modularization), parameterisation, and individual programming (Jahnke et al. 2001, p. 117). Configuration means merely the selection and arrangement of specific software components, whereas parameterisation stands for the initialisation of wanted functions by setting the relevant parameters. Individual programming does not stand for intervening into the standard software's code but for the individual programming of additionally required functions (Oberniedermaier 1999, pp. 46-47).

The alternative to a customizing of the standard software is the reorganization of business processes according to the standard software. An adaptation of both software and business processes at the same time frequently is the wisest solution (Oberniedermaier 1999, p. 47).

The building of large integrated software systems brings along several challenges to the techniques and tools that support it. Common problems are data integration, reduction of complexity, improvement of reliability, and supporting middleware. To categorize the problems that affect the quality assessment of components and component based systems following classification can be used (Cechich et al. 2003, p. 3):

- The internal processing of a component according to the provider's specification is generally unknown to the customer, except for what the component's interface is providing. This reduces controllability and testability of the component. Furthermore, a component can be deployed in a new context that was not foreseen by the provider. During development, the component can only be assessed according to a particular usage profile.
- Interactions of modules may have explicit interdependencies about the operating system or middleware platform required. The middleware works as an adapter between the components. But even if the adapter between two components translates the interactions correctly, there is no guarantee that the semantic interaction between the two will be meaningful.
- The composition of modules in different contexts can cause problems because an otherwise fault-free component may fail to function as expected when it is integrated into a system of other context. The components have initially been developed in mutual ignorance for each other. As a result, they may completely fail when they are deployed and integrated into a new system.[2]

[2] Issues related to acquisition risks, such as vendor maturity level, are not considered in this paper.

Special implications arise from updated software releases. Eventually, further customer-specific parameterisation of the system is required or data migration, if data structures have changed. Other aspects are the modification of business processes or the training of users. From this point of view, the impact of a release changeover can be compared to a standard software implementation and therefore deserves identical quality assurance procedures, e. g. regression testing (Sneed 2005).

3 LIFE-CYCLE APPROACHES

3.1 Overview of Standard Software Implementation Approaches

A success factor for a felicitous go-live of the standard software system is the applicability of the underlying implementation life-cycle model. There is no unique definition of life-cycle models. As different application domains for standard ERP software require individual approaches, flexibility is a major concern of life-cycle models (Knoell et al. 2001, pp. 130-131).

From the viewpoint of project management, life-cycle models mainly fulfil three tasks. They define the project activities, they promote consistency between projects by providing a uniform and transparent process, and - as a result - they provide the necessary transparency for the controlling of the project. They therefore very much contribute to the maturity of the implementation process (Liggesmeyer 2002, pp. 345-346).

According to Balzert (1999, p. 98), a life-cycle should contain following elements:

- The sequence of project-phases
- Activities of each phase
- Definition of partial results including layout and content
- Criteria for completion (when is a milestone reached?)
- Required qualification of staff
- Responsibilities and competencies
- Standards, guidelines, methods, and tools to be applied

Traditional life-cycles for software development, e. g. the waterfall model or spiral model, laid the foundation for implementation life-cycles (Balzert 1999, p. 98). In addition, implementation life-cycle models for standard software generally contain reference models, tools, or preliminary parameter settings for rapid implementation.

The variety of implementation life-cycles ranges from purely software driven approaches - mainly offered by standard software vendors - to purely process driven approaches mainly invented by consulting firms who focus on business process optimization. In the course of this paper, we evaluate three exemplary life-cycle models: Strategy-Based Implementation by Kirchmer (1998) as an example of a business process driven approach, ASAP by SAP as an

example of a software driven approach (Geiss 1998, pp. 27-33), and Target Enterprise by Baan as an example of a software driven and process oriented approach (Knoell 2001, pp. 159-168).

Knoell et al. (2001) give a detailed analysis and comparison of the life-cycles but focus on their capability to optimize business processes. Within this paper we assess these implementation life-cycles with regard to their support of software quality assurance. Since it can be assumed that improved implementation process capability or maturity leads to a decreased number of faults experienced during test, the outcomes of such an assessment will contribute to establishing a new, integrated quality framework for implementation life-cycles that combines product and process quality assurance.

3.2 Evaluation of the Implementation Life-Cycles with Respect to Their Support of Quality Assurance

In the following the well-known Capability Maturity Model (CMM) developed by the Software Engineering Institute (SEI 2006) is used to assess the implementation life-cycles for their process capability. Like the ISO/IEC 15504 model, it represents a scientifically developed concept that has been proven in practice. However, a complete assessment of the life-cycles is prohibitive for our purposes. Therefore, according to the objective of investigating the influence of the implementation process capability on product quality assurance, emphasis is put on processes closely related to software quality.

The approaches for Standard Software implementations discussed in this paper are graded using the evaluation criteria provided by the CMMI process areas at level 3. Each implementation approach is evaluated on a scale with four different attribute values:

- "--" The goals and practices of the process area are neither realized nor described or mentioned as a principle, objective or characteristic.
- "-" The goals and practices of the process area are mentioned or described, but only on an abstract level. It is therefore not realized or lacks a precise description of project activities, guidelines, or work products.
- "+" Goals and practices of the process area are realized within single project phases by means of defined project activities, guidelines, etc.
- "++" Goals and practices of the process area realized throughout the implementation lifecycle, i.e. all project phases contain sufficient project activities, guidelines, etc.

Table 1 summarizes the gathered findings: All approaches define appropriate project activities and results that are able to realize a certain level of process quality. Especially ASAP provides an inherent process capability, as it primarily focuses on function oriented implementation by means of a reference model. Target Enterprise contains phases similar to prototyping. This contains the danger of unclear specifications due to changes of requirements that can occur in late phases of the life-cycle. This danger is aggravated by the use of ARIS.

Process Area	Strategy Based	ASAP	Target Enterprise
Configuration Management (CM)	+	+	++
Integrated Project Management (IPM)	++	++	++
Measurement and Analysis (MA)	-	-	-
Organizational Process Definition (OPD)	++	+	+
Organizational Training (OT)	-	+	+
Product Integration (PI)	-	++	++
Project Monitoring and Control (PMC)	-	+	+
Project Planning (PP)	+	+	+
Process and Product Quality (PPQA)	-	-	-
Requirements Development (RD)	++	--	-
Requirements Management (REQM)	+	-	+
Risk Management (RSKM)	-	-	-
Technical Solution (TS)	+	++	+
Validation (VAL)	-	+	+
Verification (VER)	-	-	+

Table 1: Assessment of SSW-Implementation Approaches

As it can be seen, all life cycle models aim for an efficient implementation of systems but lack support for quality assurance. The proprietary approaches have developed sophisticated toolkits to ensure an accelerated project execution and more uniform and consistent project results. These approaches also re-use existing solutions such as reference models, document patterns or system templates. ASAP disregards existing business processes entirely and offers no opportunities for iterations as it is based on the sequential waterfall approach. The dynamic applicability, which allows later requirements to be included within the project life cycle, is mainly considered by Target Enterprise. In addition, adaptability or tailoring of the life cycle is consistently realized only by Target Enterprise, as it provides predefined project scenarios with related phases, activities, results, etc.

4 CASE STUDIES

4.1 Basel II Overview

Basel II is a revision of the capital adequacy framework and expands the guidelines that banks must follow for managing and disclosing risk. The new framework is designed to foster stability in the international finance sector and to eliminate the undesirable effects of the old Capital Accord. More specifically, it aims to improve banks' risk management by linking capital requirements more closely to the actual risks of portfolios (Bank for International Settlements 2008).

Basel II consists of three pillars:

- capital requirements intended to improve capabilities to manage and assess risk;
- supervisory review for the use of best practices to manage and reduce risk;
- and market-discipline rules requiring detailed public disclosure of capital structures, risk exposures, ratings models, and capital adequacies.

Meanwhile, Basel II has become an integral part of the regulatory legislation in all member states of the European Union.

Basel II was created in response to the realisation that risk-adequate capital requirements alone are no guarantee for the solvency of a bank and the stability of the banking system. Another key factor is the risk/return profile defined by the bank's management, and the ability to control and sustainably bear risk exposures. Basel II thus aims to create incentives to further improve banks' internal risk-management systems. These systems also have to be approved by the regulators. Finally, the market will also exert discipline of its own through the enhanced disclosure requirements of the banks.

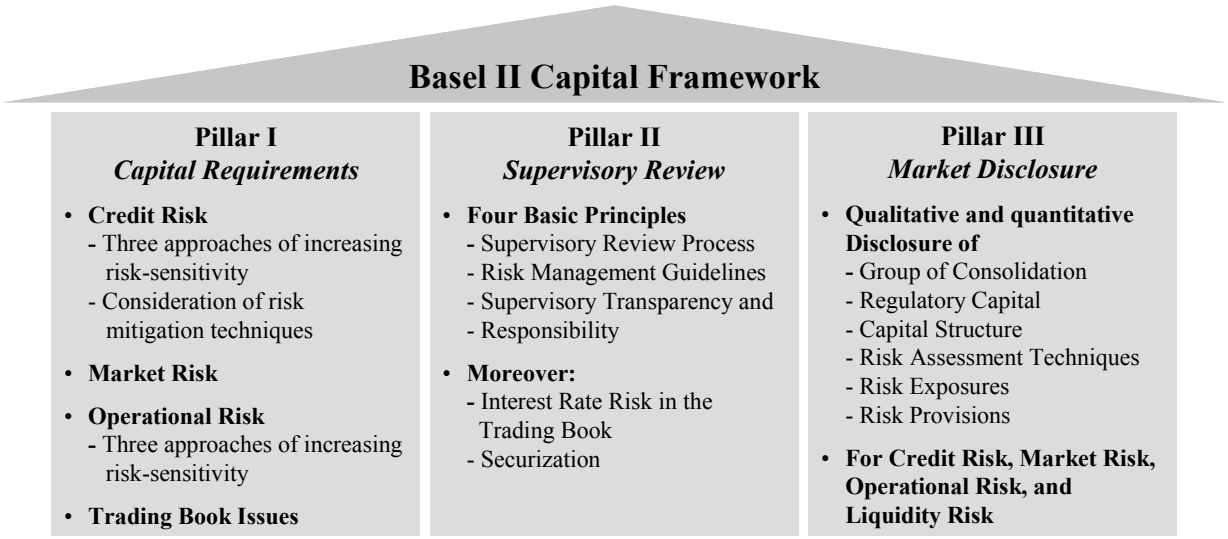


Figure 1: The Three Pillars of the Basel II Framework (Hartmann-Wendels 2003, p. 10)

Concerning credit risk in Pillar I, Basel II gives banks three different approaches for measuring capital: the Standardised Approach, the IRB Foundation Approach and the IRB Advanced Approach (IRB: Internal Ratings Based). In the Standardised Approach, which is based on external ratings offered by rating agencies, the capital requirements tend to be higher than in the IRB Approaches based on internal ratings, but calculation procedures are less complex. With the IRB Approaches, the minimum capital requirements depend on the probability of default (PD) based on the rating, the exposure at default (EAD), the loss given default (LGD), and the remaining effective maturity (M) (Hartmann-Wendels 2003, pp. 1-27).

Basel II urges banks to adapt their organizations, business processes, and technology in order to be able to perform the enhanced processing and disclose the portfolio information required. That, in turn, calls for improved data collection, data management, data quality, and data analysis across all risk types and business divisions the bank handles. The institution must also be prepared to integrate information within its IT systems to satisfy the new documentation and reporting rules (SAP 2006, p. 2).

Several vendors of standard software and consulting companies developed solutions for the technical implementation of credit risk calculation. The object of investigation in this paper represents the most elaborate Basel II package and is provided by the biggest German producer of standard software for ERP. The application supports the standardized and the two IRB approaches, all of which provide values for risk-weighted assets and expected loss. Preconfigured to comply with Basel II, the package reflects comprehensive business experience that was accumulated in working together with customer banks. The software supplier proposes an implementation life-cycle model derived from the original SAP reference model. Typical for standard software, the package allows adding or adapting of functions to the individual business processes.

The two case studies focus on the implementation of the Basel II package for the calculation of capital requirements at two major German banks. The projects took place from 2004 to 2008. One bank employed the software-oriented life-cycle for rapid implementation recommended by the standard software provider, whereas the other bank chose a more process driven life-cycle that can be compared to the strategy driven approach. Due to the long legislatory process of Basel II that ended in 2006, the Basel II solution was delivered and completed in four releases, containing regulatory and technical amendments.

	Bank A	Bank B
Calculation Approach	IRB advanced	IRB foundation
Implementation Life-Cycle employed	Strategy based	Software oriented
Total Number of Employees as of Dec 31st 2007	ca. 4.800	ca. 21.500
Average Number of Implementation Project Staff	ca. 35	ca. 70
Capital as of Dec 31st 2007	ca. € 5 bn	ca. € 5 bn
Balance Sheet Total Dec 31st 2007	€ 205 bn	€ 203 bn

Table 2: Corporate fact sheet, anonymized, taken from annual reports and data collection.

4.2 Assessment of Product Quality Assurance during the Implementation Projects

Concerning product quality assurance, test plan management tools kept a tree representation of the test specification. Both banks chose a systematic approach to testing, aiming at an exhaustive test of the system's conformance to the specification based on equivalence class testing. The reason for this test strategy was the audit-proof documentation of the complete test design and results.

Because of the complex and variable calculation required by Basel II, synthetic test data for equivalence class testing was generated manually and used for module- as well as for system-testing. For each synthetic test case, the test results were collected according to its test specification together with the kind of failure occurrence. The way in which this information

was stored does not facilitate the analysis from different angles, e.g. test progress with respect to overall test case coverage.

Operational data were used for system- and integration-tests. Though this would have suggested the use of statistical methods, they were not employed. Assessment of operational test data was carried out by inspecting the calculation of capital requirements for financial transactions that were determined by the testers or as plausibility checks for portfolios of risk types, e.g. for credit derivatives or specialized lending. If failures were detected, the test was documented, corrected, and re-executed by an example financial instrument or financial transaction according to the procedures of synthetic test cases.

In order to assess test effectiveness, providing the necessary data was a first issue. For several years, during each release, both banks had already collected information about the failures themselves, like a detailed description, the day of occurrence and the severity. However, a measure of testing effort or a record stating the test case whose execution led to the failure was not ready at hand. The determination of the order in which the test cases were executed posed another practical problem, because the test cases were not tested in strict succession.

Therefore, to gather the data appropriately, additional effort had to be invested. For logging the release intervals in which a test case was executed, and the failure occurrences, Microsoft Excel files had to be screened manually. In the analysis, the test cases are chronologically ordered. For Bank A the results show that the failure pattern and the coverage information gathered over time are inconsistent.

As each project under consideration implemented the software system according to the release-steps, the planned tests were executed several times. The analysis therefore shows improvements in test effort and test results towards the end of the projects. For Bank A, modelling failure occurrences as a function of test cases executed is not possible due to the inconsistent failure pattern and coverage information. Therefore, a prognosis of reliability or a statement of coverage is not possible.

4.3 Assessment of Process Quality Assurance during the Implementation Projects

The strengths and weaknesses connected to the employed life-cycle models (see chapter 3.2) are reflected by the shortcomings of the two implementation projects, though Bank B even took parallel activities to receive a CMMI level 2 certification during the second half of the project.

It turned out that the reference model recommended for the rapid implementation of the Basel II package did not provide all necessary functions for a complete and audit-proof coverage of the Basel II requirements. It was therefore necessary to specify the additional requirements and adapt the software functions. Another promising process candidate with well-known weak-points for which a reference model would possibly pay off - but didn’t - was the labour-intensive reconciliation of the Basel II data layer with the legacy system. The legacy system containing most of the credit transactions was also programmed by the provider of the Basel II package.

Moreover, the life-cycle model lacked sufficient proprietary tools for a support of the implementation, e. g. test case generation. Initially, undefined work products, e. g. specifications and documentations, were another weakness of the employed approach. Later on, this problem was eliminated by the introduction of CMMI level 2 conform documentation guidelines. Still, the CMMI assessment of the general IT-processes of the bank allowed no forecast or prediction of the software quality to be expected.

Bank A, employing the strategy based approach, set up an own business specific data model and therefore completely refused to employ the suggested reference model. There were two important points contributing to an overall success of this implementation project: The selection of project staff that was experienced in the implementation of standard ERP software systems and elaborate and well-defined work products as a result of a long record of standard software implementations.

Obviously, the outcomes of the business process oriented life-cycle were of better quality. However, there was no formal assessment of the life-cycle-inherent process capability or maturity that would have allowed to explain the acquired quality goals.

4.4 Lessons learned

Regarding process quality, the strategy based life-cycle results delivered sufficient results. Still, there is a strong need to integrate key performance indicators that help to deliberately judge and steer the quality factors of the project. In both banks a strong need for control of the implementation maturity and capability can be observed.

The effort for quality assurance in both projects can be judged as very high. Still, there are no objective figures that document coverage or levels of confidence that the system is built to its requirements. As the calculation of capital requirements is vital for every bank, uncertainty about the software quality may come at a high price: if the capital requirements are calculated too low, the bank may collapse. If they are calculated too high, this will lead to opportunity costs.

The test strategies aimed at a systematic exhaustive test approach. Instead, the test strategy should have been deducted from the implementation approach. For example, if there are only few changes within the software system due to the technical orientation of the life-cycle, emphasise should be put on integration testing. For both projects, the testing strategy as well as the documentation can be judged as inefficient. This raises two questions: What is the best selection of test data? And how can be judged that sufficient testing was performed?

Statistical methods allow quantifications of remaining uncertainty and test coverage. Furthermore, the assumptions connected to these models hint on the test strategy that should be employed. In the following section, a preliminary analysis of the potential application of statistical methods to testing will lay the foundation for more focused research. Furthermore their potential impact on black-box testing of standard ERP software systems will be estimated.

5 ANALYSIS OF INSTRUMENTS FOR PRODUCT QUALITY ASSURANCE

5.1 Overview of Testing Techniques

Systematic approaches to the design of test cases are meant as guidelines for specifying test cases that avoid redundancies. Here we distinguish black box testing from white box testing. Contrary to white box tests, black box techniques don't consider the code-structure of a software program. Instead, the test cases are generated according to the specification and concentrate on input and output parameters. The specification is also used to control the correctness and completeness of the test cases (Spillner et al. 2007, pp. 101-133).

The fundamental concept of functional equivalence partitioning is to divide the input domain of each input parameter into several classes in a way that the test problem is reduced to a very simple set of test cases. However, for standard ERP software the behaviour of each component is determined by quite a number of input parameters. Therefore the number of test cases for covering the whole system has to be based on the Cartesian product of the equivalence classes of these parameters. As a result, the number of combinations gets very high. (Liggesmeyer 2002, p. 47-53).

White-box techniques state that each test case should execute code-constructs not tested before. If the total number of code constructs in the software under test is known, the metric coverage (ratio of constructs executed) can easily be derived. Though customers of standard software generally have no insight into a components code, we may well have a look at this technique because each single functionality provided by a software module can be regarded as a code construct or structure element (Liggesmeyer 2002, p. 37).

Operational testing means the execution of test cases according to the expected field usage of the productive software. The assumed operational profile consists of possible software operations and their execution probabilities. These occurrence probabilities may be derived from field usage data available or using expert opinion. The operational profile may be described by a probability mass function or by a tree diagram in which consecutive decisions are modelled together with their selection probabilities (Grottke 2003, p. 6).

Operational test cases may be redundant since more than one fault might be associated with one operation. In addition, a tool is required that allows to control the correctness of the output produced by the software under test. E.g., this can be the old system or an independently developed implementation of similar software. However, manual compilation is only practicable for a restricted number of test cases (Grottke 2003, p. 7).

With regard to the case studies, the specification of systematic test cases turns out to be too involved. Even if the systematic test cases had a higher fault detection probability this would be outweighed by the huge number of test cases derived from operational data. Concerning testing of standard ERP software systems, the arguments for operational testing prevail (Grottke 2003, p. 8):

- Almost all statistical methods for the estimation of reliability assume that the data analyzed were collected while testing the software according to its operational profile.

- For operational testing, input values are selected randomly and not according to a sophisticated strategy. Therefore tool support for the automated generation of test cases is facilitated. If the test case specification and execution are automated, then a vast number of test cases can be run. The case studies showed that generation of test data based on productive data improved test efficiency even more. This observation supports the following argument.
- If usage during testing does not differ from usage after final release of the software into production, the reliability observed can directly be applied. This means that in addition to the detection of faults operational testing also serves for the assessment of the current reliability of the piece of software.

5.2 Binomial Trials

The binomial model describes the number of successes in n independent trials, where each trial has probability of success p. It is widely used in statistical analyses, and offers a flexible and robust approach. The paradigm application is the determination of the probability of obtaining exactly k heads in n tosses of a (possibly unfair) coin. Formally, the binomial model states that

(1) $P(X = k) = \binom{n}{k} p^k (1 - p)^{n-k}$ (Callagher et al. 1998, p. 11).

To use this model for testing, n is supposed to be the number of tests built from the requirements specification. Assumed that a functionality of a standard software component will be independently executed for test with unknown probability p, the interest of the assessor is in estimating p. Based on binomial trials, Gallagher et al. (1998, p. 11-15) describe applications of the model that help to quantify the uncertainty about the performance of the software under test, make statistical forecasts of that software's future performance, and reduce testing costs.

Their result is that the application of the binomial model brings about modest decreases in the number of tests needed to achieve the same level of certitude as present practice. Moreover, they describe variations of the model allowing testers to make explicit probability statements about the degree of uncertainty in the test results, as opposed to current practice, in which the uncertainty in the acceptance of the system is neglected.

5.3 Software Reliability Growth Models

Software reliability measurement and estimation can be defined as the measurement and prediction of the probability that the software will perform its intended function (according to specifications) without error for a given environment and period of time (Callagher et al. 1998, pp. 21-22).

During testing phases, the observation of a failure leads to the detection and correction of the underlying fault. As testing proceeds, the number of failures decreases while reliability increases. Statistical methods that describe this effect are called software reliability growth models. Continuous software reliability growth models refer to testing progress as a steady

measure over time, e.g. the overall time spent for test case execution. Discrete software reliability growth models regard testing progress as separate fault detection periods, e. g. a discrete execution of test runs or test cases. In tests of standard software systems for ERP, the focus is on the time-independent case. Continuous time measures are difficult to collect in implementation projects due to varying test intensity (Liggesmeyer 2002, pp. 451-468).

Various reliability models exist. Their advantage is their early, preventive applicability. But the precision of estimates depends on the applicability of the respective reliability model on the information gathered during test (Liggesmeyer 2002, p. 466-468). Generally, reliability growth models rely on data gathered by operational testing. Still, some of the models assess software reliability based on systematic testing (Grottke 2003, p. 27).

As we have seen, software systems are usually subject to an intensive test suite in order to discover errors. Traditionally, this is done in a systematic manner. But in practice, as demonstrated by the case studies, this approach to testing of complex standard software systems is simply not possible because of the large number of states and the lack of regularity in its structure. Therefore, the testing domain should be restricted to an input domain that reflects the usage profile. This will surely reduce the magnitude of the testing problem. Then, a statement can be computed for software reliability. This statement, of course, will be usage specific, but represents a reasonable approach.

The analysis of the suitability of software reliability growth models for implementation life-cycles therefore has to evaluate, if reliability models connected to operational testing provide better estimates of software reliability than those connected to systematic testing. As indicated in chapter 5.1, the reduced effort for operational testing compared to the effort for developing systematic test cases supports the application of reliability models based on operational testing.

5.4 Coverage

Coverage information can also be obtained by a stochastic analysis of the test strategy employed. Not only determines the test strategy the choice of a suited reliability growth models but also the coverage of code constructs as testing proceeds. The reason is that systematic testing avoids repeated executions of code constructs already exercised before whereas operational testing can lead to redundant executions of code constructs.

Piwowarski et al. (1993, pp. 287-301) make following assumptions to the coverage of code constructs:

- The program under test consists of G code constructs.
- Per test case, p constructs are sensitized.
- The p code constructs are always randomly chosen from the entire population. The fact that a construct has already been executed does not lower its chances of being executed in future.

With regard to the comparison of testing techniques in chapter 5.1 these assumptions correspond to operational testing with a homogenous profile, in which all equally-sized operations have identical occurrence-probabilities. Regarding systematic testing, the third

assumption has to be changed to: “Code constructs which have already been tested are not exercised a second time”. Both assumptions mark extremes. Neither will operational testing lead to endless repetitions of code executions, nor can systematic testing perfectly avoid redundancies (Grottke 2003, p. 35).

The potential of coverage designs to drastically reduce the number of runs needed to test a program is connected to a systematic test design. The difficulty in applying it is that it is not obvious how one identifies a module or code construct, especially in the context of black-box testing. The use of specified functionalities representing a code construct may serve as a substitute. Then, a further and more practical issue is the difficulty to generally determine modular structure in a program from functional requirements (Callagher et al. 1998, pp. 15-16).

6 CONCLUSION AND NEXT STEPS

The case studies in section 4 proved the inadequacy of current quality assurance methods for standard ERP software systems. For product quality assurance of standard ERP software systems exhaustive testing is not practical. As the comparison in section 5 shows, statistic methods provide a level of confidence or probability that a program implements its functional specification correctly. Furthermore, they include approaches to coverage designs and are generally based on operational testing. Concerning product quality assurance the results will serve as a basis for a unifying model framework identifying the driving factors of the expected number of failures, coverage, and test effort.

We also identified life-cycle specific weaknesses that affect software quality management. Since it can be assumed that increased process maturity or capability leads to an improvement in the mean outcomes of the implementation process, it can be expected that process quality has an influence on the estimated number of faults. However, the costs connected to an assessment that follows all the rules of the official guidelines to performing capability evaluations are often considered inadequate for the purpose of standard software implementation projects. Therefore, we will suggest a check list for a quick-assessment of implementation process capability relieving the task of applying the generic capability criteria to the specific life-cycle model employed. Based on the potential influence of process characteristics on the model parameters that drive test effort and reliability, the check list is meant to bridge the gap between product and process quality assurance. The validity and reliability of this instrument have to be analyzed. The results of this investigation will be presented in a follow up paper.

References

Balzert, Helmut (1999). Lehrbuch der Software-Technik, Software Management, Software-Qualitätssicherung, Unternehmensmodellierung. Spektrum Akademischer Verlag, Berlin.

Bank for International Settlements (2008). Basel II: Revised international capital framework. http://www.bis.org/publ/bcbsca.htm as of 30 Nov. 2008.

Botella, P., Burgues, X., Carvallo, J.P., Franch, X., Pastor, J.A., and Quer, C (2003). Towards a Quality Model for the Selection of ERP Systems. In Cechich, A., Piattini, M., Vallecillo, A. (Eds.). Component-Based Software Quality. Springer. Berlin, Heidelberg, New York, 225-245.

Callagher, Leonard (Ed.), D. Banks, W. Dashiell, C. Hagwood, R. Kacker, L. Rosenthal (1998). Software Testing by Statistical Methods. National Institute of Standards and Technology. Information Technology Laboratory. Gaithersburg

Cechich, Alejandra, Piattini, M., Vallecillo, A. (2003). Assessing Component-Based Systems. In Cechich, A., Piattini, M., Vallecillo, A. (Eds.). Component-Based Software Quality. Springer. Berlin, Heidelberg, New York, 1-20.

Curran T. and G. Keller (1998). SAP R/3 Business Blueprint: Understanding the Business Process Reference Model. Addison Wesley. Prentice Hall.

Geiss, Marcus and Soltysiak, R. (1998). SAP R/3 dynamisch einführen. 1st Edition. Add. Wes., Bonn.

Grottke, Michael (2003). Modeling Software Failures during Systematic Testing. The Influence of Environmental Factors. Shaker, Aachen.

Hartmann-Wendels, Thomas (2003). Basel II: Die neuen Vorschriften zur Eigenmittelunter von Kreditrisiken. Economica, Heidelberg.

Jahnke, Bernd and Wall, F. (Eds.) (2001). IT-gestützte betriebswirtschaftliche Entscheidungsprozesse. Dieter Pressmar zum 65 Geburtstag. Gabler. Wiesbaden, p. 117.

Kirchmer, Mathias (1998). Business Process Oriented Implementation of Standard Software. Springer, Heidelberg.

Knoell, Heinz-D., Kuehl, L., Kuehl, R., and Moreton, R, (2004). Evaluation of Standard ERP Software Implementation Approaches in Terms of their Capability for Business Process Optimization. Journal of Computing and Information Science in Engineering, Sep. 2004, Vol. 4, 271-277.

Knoell, Heinz-D., Kuehl, L., Kuehl, R., and Moreton, R, (2001). Optimising Business Performance with Standard Software Systems. 1st Edition. Vieweg, Braunschweig/Wiesbaden.

Liggesmeyer, Peter (2002). Software-Qualität: Testen, Analysieren und Verifizieren von Software. Spektrum Akademischer Verlag. Heidelberg, Berlin.

Meyers, Craig and Oberndorf, P. (2001). Managing Software Acquisition : Open Systems and COTS Products. Addison-Wesley, Boston, MA.

Oberniedermaier, G. and Geiß, M. (1999). SAP R/3-Systeme effizient testen. Addison-Wesley, Munich.

Piwowarski, P., Ohba, M., Caruso, J. (1993). Coverage Measurement Experience during Function Test. In Proceedings 15th International Conference on Software Engineering. 287-301.

Rolf, Arno (1998). Grundlagen der Organisations- und Wirtschaftsinformatik. Springer. Berlin, Heidelberg.

SAP AG (2006). Improving Bank Credit-Risk Management with SAP® Basel II. SAP Solution Brief. SAP for Banking. SAP AG, Walldorf.

SEI (2006), Software Engineering Institute. CMMI for Development, Version 1.2, CMMI Product Team, Carnegie Mellon University, Pittsburgh.

Sneed, Harry M., Hasitschka, M., and Teichmann, M.-T. (2005). Software-Produktmanagement: Wartung und Weiterentwicklung bestehender Anwendungssysteme. 1st Edition. Dpunkt, Heidelberg.

Software Engineering Institute (2006). CMMI for Development. Carnegie Mellon, Pittsburgh, PA.

Spillner, A., Linz, T., Schaefer, H. (2007). Software Testing Foundations. 2nd Edition. RockyNook, Santa Barbara, CA.

Stahlknecht, Peter and Hasenkamp, U. (1999). Einführung in die Wirtschaftsinformatik (Introduction to commercial information technology), 9th Edition. Springer Verlag, Berlin, Heidelberg.

IT-Organisation in Hochschulen

Horst Meyer-Wachsmuth

Abstract

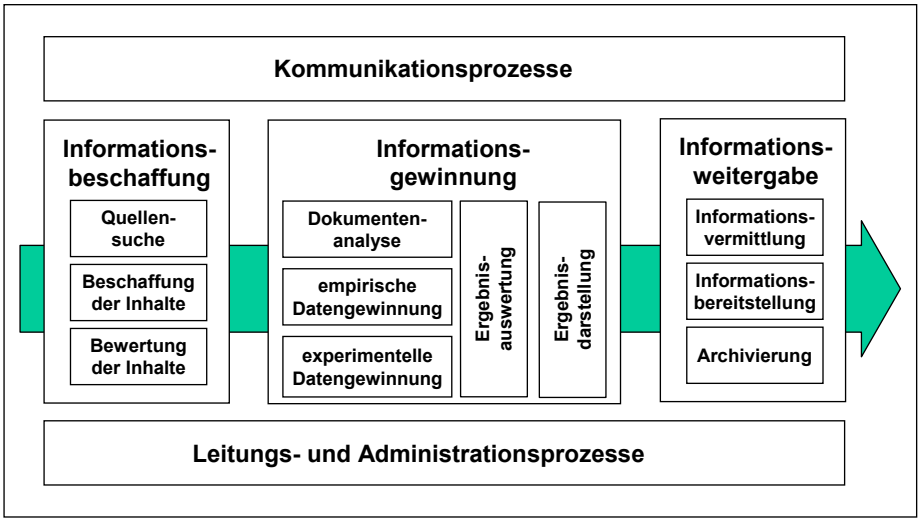
Ausgehend von einer Untersuchung der HIS und von Erfahrungen des Autors in der Leitung einer Hochschule wird die Notwendigkeit einer Professionalisierung der IT-Versorgung aufgezeigt und ein IT-Versorgungskonzept verargumentiert.
Anlass dieser Arbeit war die Ausformulierung und Aktualisierung der Ergebnisse eines Beratungsauftrags für mongolische Hochschulen, die in der Mongolei veröffentlicht werden sollen. Insofern berücksichtigt dieser Beitrag die Situation in der Mongolei; jedoch ist das vorgestellte IT-Versorgungskonzept aktuell und übertragbar auf andere Hochschulen auch in Deutschland.
Eine Implementierung dieses Konzeptes allerdings erfordert hochschulspezifische Anpassungen und Detaillierungen.

Einleitung

Die Kernprozesse (Abb. 1) für Forschung und Lehre in modernen Hochschulen werden - ebenso wie in anderen Organisationen - immer stärker durch Informationstechnik (IT) unterstützt bzw. entscheidend geprägt. Die Kosten für IT und IT-Unterstützung steigen. Deshalb muss der Organisation der IT, die in vielen (z. B. deutschen) Hochschulen historisch gewachsen ist, selbst verstärkt Aufmerksamkeit geschenkt werden, damit nicht unnötig Ressourcen vergeudet und die Prozesse optimal unterstützt werden. So erscheint es sinnvoll, der Frage nachzugehen, wie die IT-Dienstleistungen organisiert werden sollen. Hier können jedoch nur einige subjektiv ausgewählte Aspekte der IT-Organisation diskutiert werden, die nach langjähriger Erfahrung des Autors in der Hochschulleitung als wichtig erachtet werden – weitgehend unabhängig vom Grad der „Computerisierung“ der Hochschule.

Der vorliegende Artikel ist entstanden als schriftliche Ausarbeitung und Aktualisierung eines Vortrags den der Autor im September 2005 an der *School of Business and Commerce* in Ulaanbaatar (Mongolei) gehalten hat. Zur Vorbereitung dieses Vortrages wurde dem Autor seinerzeit von der *Hochschul Informations System GmbH (HIS)* der Entwurf einer umfangreichen Arbeit von H. Moog „IT-Dienste an Universitäten und Fachhochschulen“

überlassen[1] auf die sich der Vortrag weitgehend abstützte. Ihr gilt mein Dank. Fehler in der vorliegenden Ausarbeitung gehen zu meinen Lasten.



[Quelle: HIS GmbH]

Abb. 1 Referenzmodell der Informationsverarbeitungsprozesse an Hochschulen

Situation der IT-Versorgung an den Hochschulen

Unter *IT-Versorgung* soll die Bereitstellung und der Betrieb der Informations- und Kommunikationstechnologien sowie die Unterstützung ihrer Nutzer verstanden werden[2]. Dabei wird im Folgenden unterstellt, dass es sich um die IT-Versorgung von **nicht technischen** (keine Informatik, keine naturwissenschaftlichen/technischen) Fächern in der Hochschule handelt, die für ihre Aufgaben in der Forschung und Lehre keine eigene spezielle IT-Kompetenz brauchen.

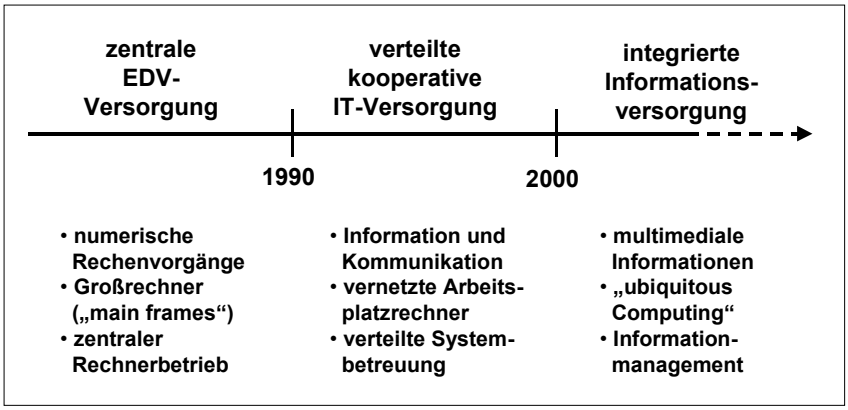
Die IT-Versorgung an Hochschulen hat sich analog zur Entwicklung in Organisationen außerhalb der Hochschulen in drei Stufen entwickelt (Abb. 2):

1. IT-Versorgung über zentrale DV-Anlage, angeboten von einer zentralen EDV-Dienstleistungsorganisationseinheit
2. IT-Versorgung durch verteilte DV-Anlagen, die - zunehmend vernetzt – in Kooperationsverbünden eingesetzt und dezentral betreut[3] wurden
3. Allgegenwärtige IT-Unterstützung („Ubiquitous Computing“) bei multimedialer Informationsgewinnung, -beschaffung und –weitergabe (s. auch Abb. 1) sowie bei allen administrativen Prozessen

[1] Moog, H.: Hochschulplanung 178: **IT-Dienste an Universitäten und Fachhochschulen** (2005) *[vergriffen]* (HIS GmbH Hannover 2005 gefördert vom BMBF) ISBN 3-930447-71-1; abrufbar unter http://www.his.de/abt3/ab33/FuL/itdienste

[2] Dr. Horst Moog (HIS GmbH): Alternative IT-Versorgungskonzepte a. a. O.

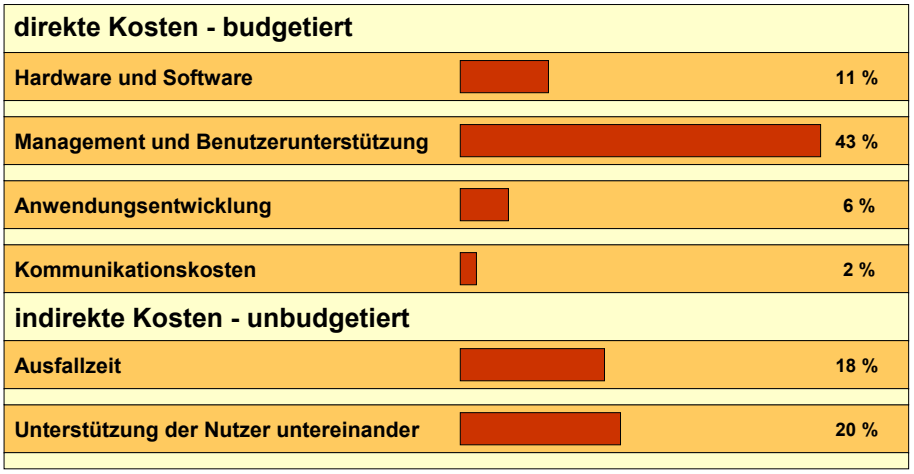
[3] Unter Betreuung soll hier, die Aufrechterhaltung der Betriebsbereitschaft der DV-Anlage (inkl. Betriebssystem und ggf. – abhängig von der jeweiligen Situation - Standardanwendungssoftware) verstanden werden.



[Quelle: HIS GmbH]

Abb. 2: Entwicklung der IT-Versorgung

Die zunehmende Durchdringung aller Aktivitäten an den Hochschulen in der dritten Stufe führte zu steigenden Kosten für die IT-Versorgung. Analog zu der in der Wirtschaft gewonnenen Erkenntnis über die gesamten Kosten (*Total Cost of Ownership (TCO)*[4] gilt auch an den Hochschulen, dass **indirekte** Kosten einen großen Teil der Kosten ausmachen (Abb. 3).



nach [Gartner und Forrester Research, B. Höhmann, Philipps-Universität Marburg]

Abb. 3: IT-Kosten in Universitäten

[4] Gartner Group (o. V. 2002)

Nach Einschätzung des Autors ist der Anteil indirekter Kosten an den Hochschulen eher höher als die in Abb. 3 angegebenen 38% oder die anderen Orts[5] angegebenen 55%, da die Autonomie der Professoren (und Institutsleiter) [6], zu intransparenten Kosten und Leistungen im Kontext der IT-Versorgung führen: Wissenschaftliche Mitarbeiter, Hilfskräfte und unterstützendes Personal werden oft auf jeder Ebene der Organisationshierarchie (innerhalb von Fakultäten, Instituten, Forschungsgruppen (Abb. 4)) für Aufgaben der Systembetreuung eingesetzt. Diese Art der IT-Versorgung kann deshalb als „dezentral“ bezeichnet werden.

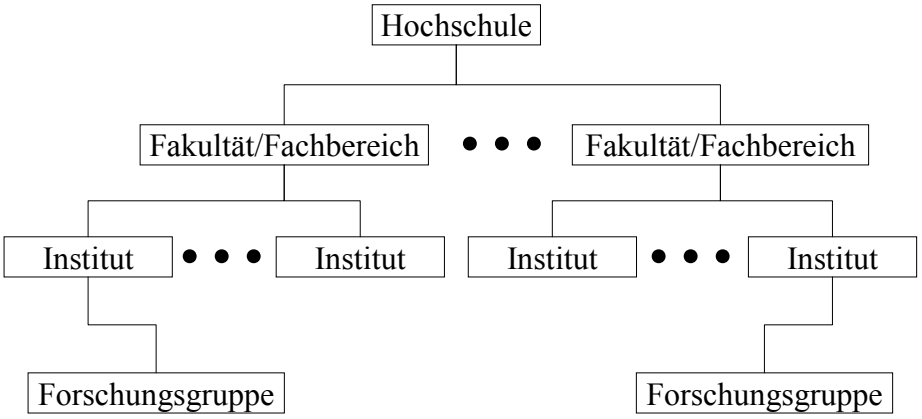


Abb. 4: Organisationsstufen einer Hochschule

Bei der dezentralen IT-Versorgung werden die notwendigen Aufgaben häufig zwar engagiert aber nicht professionell erledigt. Sie lenken von eigentlichen Aufgaben in Forschung und Lehre ab und könnten besser und effizienter von professionellem IT-Personal erledigt werden. Da solches zentrales IT-Personal jedoch nicht der Disposition der Leiter dezentraler Organisationseinheiten unterliegt, wird die Inanspruchnahme zentraler IT-Kompetenz häufig vermieden.

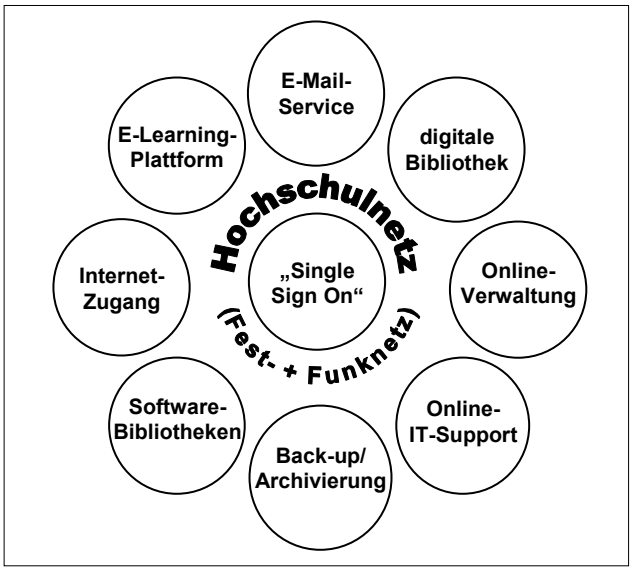
Die dezentrale IT-Versorgung führt oft zu parallelen Aktivitäten und Problemen bei der Koordination und bei der Durchsetzung hochschulweiter IT-Strategien. Wenn die Hochschule bzw. die Organisationseinheit zu klein ist, um größere Gruppen von dezentralen Systembetreuern zu unterhalten, entstehen oft Probleme mit der Weiterentwicklung und Kontinuität der Erbringung der Dienstleistungen. An Hochschulen ist die Nachhaltigkeit dezentraler Systembetreuung besonders problematisch, da der Wissensverlust durch wechselndes Personal sehr hoch ist; denn der Anteil befristet Beschäftigter (wissenschaftliche Hilfskräfte, Diplomanden bzw. Bachelor-/Masterexamenskandidaten, Doktoranden) ist im allgemeinen größer als in Organisationen außerhalb der Hochschulen.

---

[5] Gadatsch, A., Mayer, E.: Masterkurs IT-Controlling; 2. Aufl. Vieweg-Verlag, Braunschweig, Wiesbaden 2005 (S. 80)

[6] Peter Altvater/ Yvonne Bauer/ Harald Gilch (Hrsg.): Organisationsentwicklung in Hochschulen (http://www.his.de/pdf/pub_fh/fh-200714.pdf abgerufen am 11.1.09)

Die z. B. durch ein „Single Sign On“ zunehmend notwendige Integration[7] vieler Felder der IT-Versorgung (Abb. 5) führt zu steigender, schwer beherrschbarer Komplexität durch eine große Vielfalt gekoppelter Hard- und Software.



[Quelle: HIS GmbH]

Abb. 5: Beispiel: Integrationsfelder der Informationsversorgung

Wegen der zunehmenden Bedeutung der IT erhält die Beherrschung dieser Komplexität eine Schlüsselstellung für die Wettbewerbsfähigkeit der Hochschulen. Deshalb können Kompetenzmangel und fehlende Kooperationsbereitschaft zur die Entwicklung der Hochschulen gravierend beeinflussen. Kooperation wird jedoch durch die (in Deutschland garantierte) Unabhängigkeit der Leistungsträger in Hochschulen eher behindert als gefördert, so dass Veränderungen in zentralen technischen und organisatorischen Problemfeldern nur langsam vorankommen. So stellt eine Reorganisation der IT-Versorgung unter dem stetigen Veränderungsdruck neuer IT-Technologien eine hochschulpolitische Herausforderung dar.

IT-Technik ist grundsätzlich überall verfügbar („IT doesn’t matter!“[8]); jedoch ihr Einsatz ist – vorsichtig formuliert - an Hochschulen oft nicht optimal. Es erscheint unstrittig: „Das IT-Management an Hochschulen muss professionalisiert und personell verstärkt werden.“[9]

Da nur selten an Hochschulen IT-Kompetenz in den Leitungsgremien vorhanden[10] ist, muss diese Erkenntnis in vielen Hochschulleitungsgremien (und Ministerien) noch verbreitet werden, damit in diesen Gremien die notwendigen finanziellen Mittel bewilligt werden, um eine Reorganisation der IT-Organisation der Hochschulen umsetzen zu können.

[7] Hildebrand, K. (Hrsg.): IT-Integration & Migration; HMD 257; dpunkt.verlag (Heidelberg) 2007
[8] Carr, N. G.: IT Doesn’t Matter, Havard business review, 5,2003, S. 41-58
[9] Campus Innnovation und V. Konferenztag Studium und Lehre 20.-21. Nov. 2008 in Hamburg (http://www.campus-innovation.de/node/235 abgerufen am 11.1.09)
[10] Campus Innnovation und V. Konferenztag Studium und Lehre 20.-21. Nov. 2008 a. a. O.

Varianten der IT-Organisation

Die vielfältigen Aufgabenfelder von Informationsstrukturdiensten (Abb. 6) lassen sich, wenn der wichtige Bereich der IT-Dienste für die Verwaltung der Hochschule nicht explizit ausgewiesen wird, grob in drei Bereiche einteilen:

- Bibliothek
  (Erwerb, Erschließung, Bereitstellung von Informationen ...)
- Multimediadienste
  (technische und konzeptionelle Unterstützung bei Medienerstellung und –nutzung in Forschung und Lehre...)
- IT-Dienste
  (Bereitstellung und Betrieb informationstechnischer Infrastruktur...(auch für die Verwaltung) ....)

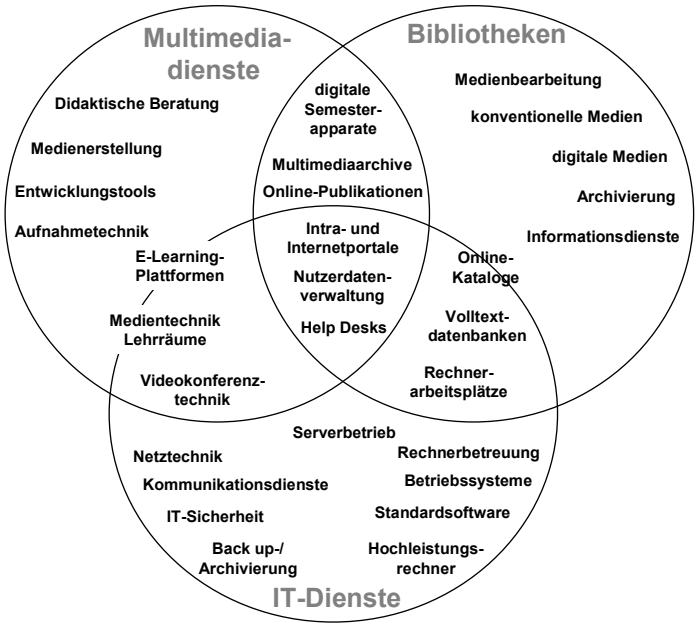
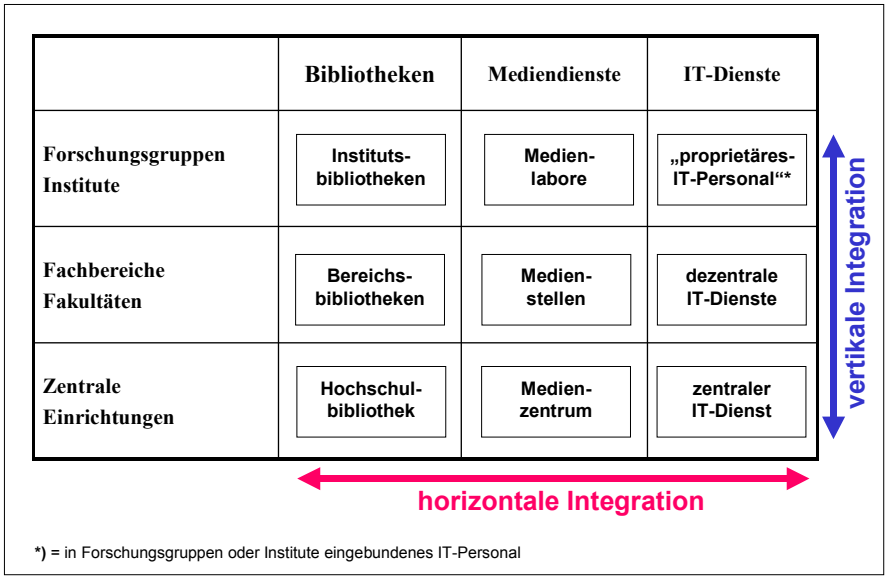


Abb. 6: Aufgabenfelder von Informationsstrukturdiensten

Nicht alle diese Aufgaben werden heute von den Hochschulen (gleichermaßen) wahrgenommen. Die Vielfalt anzutreffender unterschiedlicher Organisationsstrukturen kann gemäß Abb. 7 kategorisiert werden. Wobei in den Hochschulen – historisch gewachsene, sehr unterschiedliche Mischformen anzutreffen sind.

Wie bereits erwähnt, erfordert die Professionalisierung der Aufgabenwahrnehmung stabile, große Organisationseinheiten, in denen ausreichend Kompetenz und Kapazität nachhaltig verfügbar gehalten werden können. Die Bildung von solchen großen Organisationseinheiten erfordert i. d. R. die Integration der verschiedenen möglicherweise vorhandenen, gewachsenen Organisationseinheiten, die für die IT-Versorgung zuständig sind.
Zwei grundsätzliche Arten von Integration (Abb. 7) können unterschieden werden:

- vertikale Integration:
  Zentralisierung der Dienstleistungen für die drei verschiedenen Aufgabenfelder (nach Abb. 6) in drei separate Organisationseinheiten
- horizontale Integration:
  Zusammenfassung der drei fachlichen Aufgabenfelder in dezentrale und zentrale Organisationseinheiten.



[Quelle: HIS GmbH]

Abb. 7: Dimensionen der organisatorischen Integration

Eine Entscheidung für oder gegen organisatorische Integration wird geprägt durch die Einschätzung ihrer Auswirkung gemäß verschiedener Kriterien, die der Übersicht halber ohne Anspruch auf Vollständigkeit und Priorisierung in folgender Tabelle (Abb. 8) aufgeführt werden.

		Bewertung		Auswertung	
Kriterium	**Gew. d. Kriter.**	**horiz. Integr.**	**vert. Integr.**	**horiz. Integr.**	**vert. Integr.**
Innovationsförderung	2	2	3	4	6
Effizienz durch Kompetenz	3	2	3	6	9
Vermeidung paralleler Aktivitäten	2	3	2	6	4
Entwicklung/Durchsetzung von Strategien	3	0	3	0	9
Nachhaltigkeit durch Größe/Kapazität	3	1	3	3	9
Effektivität für die ganze Hochschule	3	1	3	3	9
Steuerbarkeit	2	1	3	2	6
Förderung einer Dienstleistungsmentalität	2	0	3	0	6
Summe				24	58

Abb. 8: Bewertungen der verschiedenen Integrationsmöglichkeiten

Ebenso werden – nur der Übersicht halber - die subjektiven groben Gewichtungen der Kriterien und die subjektiven groben Bewertungen der möglichen Integrationsformen durch den Autor dargestellt.

Im Folgenden werden Kriterien, Gewichtungen und Bewertungen genauer erläutert. Dabei ist zu berücksichtigen, dass Abweichungen zu den Bewertungen im konkreten Einzelfall immer auftreten können, da unterschiedliche Leistungsbereitschaft und –fähigkeit sowie Kreativität von Individuen zu völlig unterschiedlichen Ergebnissen bzgl. der Erfüllung der Kriterien führen können.

Eine Verteilung der IT-Versorgung über viele organisatorische Ebenen der Hochschulorganisation führt i. d. R. zu so kleinen IT-Dienstleistungseinheiten, dass eine Spezialisierung Einzelner innerhalb dieser Einheiten schwer organisierbar ist. Dies führt zu weniger fachlicher Kompetenz. Gleichzeitig fehlt in den kleinen IT-Dienstleistungseinheiten die hilfreiche, weiterbildende Kommunikation mit Kollegen und Kolleginnen, die gleiche oder verwandte Tätigkeiten ausüben. Dies verstärkt den Mangel an Kompetenz der Mitarbeiter, führt zu weniger Effizienz und Innovationen; denn für innovative Projekte kann i. d. R. keine Kapazität freigemacht werden. Zusätzlich werden durch redundante Leistungsangebote parallele Aktivitäten (z. B. Erwerb von Wissen und Kompetenz, Koordinationsbedarf) ausgelöst, die Kapazität binden.
Gleichartige IT-Dienstleistungsangebote führen zu redundanten Kapazitäten für eine Grundausstattung (Abb. 9) und den damit verbundenen Personalkosten.

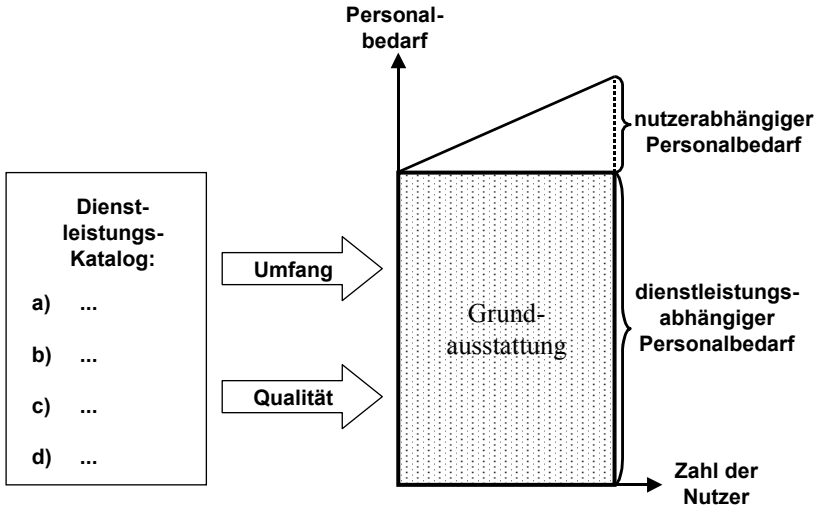


Abb. 9: Einflussgrößen auf den Personalbedarf

Bei der Entwicklung und Durchsetzung hochschulweiter Strategien sind bei dezentraler IT-Versorgung oft mühselige und aufwändige Abstimmungsprozesse zu organisieren, die helfen können, eine Lösung zu optimieren. Jedoch sind nicht immer vermeintlich fachliche Argumente hilfreich zur Verbesserung von Lösungen, sondern oft sind sie nur vorgeschoben,

um lieb gewordene Gewohnheiten oder eingeführte inkompatible Lösungen/Anwendungssysteme zu schützen. Die aus der dezentralen IT-Versorgung resultierenden inkompatiblen Systeme führen zu Aufwendungen für den Austausch von Daten. Die inkompatiblen Benutzeroberflächen (Bedienung) behindern einen flexiblen Einsatz von Nutzern dieser Anwendungssysteme in den Fachabteilungen.

Dezentrale kleine IT-Dienstleistungseinheiten führen zu mangelnden Karrierechancen für die Mitarbeiter, da nur wenige Führungspositionen vorhanden sind. Dieses wiederum kann zu häufigem Personalwechsel führen, der seinerseits resultiert in einem Verlust von Wissen und gefährdet somit die Nachhaltigkeit von Entwicklungen. Nur größere Einheiten können nachhaltig eine ununterbrochene kompetente IT-Versorgung garantieren. Zu kleine dezentrale IT-Dienstleistungseinheiten haben i. d. R. weder den Anspruch noch geeignete Führungskräfte, IT-Lösungen hochschulweit effektiv durchzusetzen.

Die vorstehenden Argumente und die sich überschneidenden Aufgabenbereiche (Abb. 6) sprechen für eine vollständige Zentralisierung von IT-Dienstleistungen insgesamt also für vertikale **und** horizontale Integration.

Jedoch zeigt die Erfahrung, dass große zentrale Einheiten i. d. R. eigene Kulturen entwickeln. Diese sind erfahrungsgemäß in den drei Bereichen Multimedia-Dienste, Bibliothek und IT-Dienste so unterschiedlich, dass eine horizontale Integration[11] nicht zweckmäßig erscheint, sondern die Aufgabenbereiche durch klare Regelungen der Zuständigkeiten für alle Beteiligte und Betroffene klar definiert und separiert werden müssen, um eine sachgerechte Dienstleistung und eine entsprechende Mentalität bei den Mitarbeitern zu erreichen. Dann unterscheiden sich Medienzentren und Bibliothek prinzipiell in ihrer Rolle als IT-Nutzer nicht von Fachbereichen bzw. Fakultäten[12].

Große Organisationseinheiten können andererseits – gerade in Hochschulen - ein Eigenleben entfalten, so dass nicht mehr die Kunden (Lehrende, Lernende und Forscher in den Fakultäten) und das Wohl der ganzen Hochschule im Fokus stehen. Einer solchen Entwicklung kann durch entsprechende Vereinbarungen zwischen den Organisationseinheiten der Nutzer und den IT-Dienstleistungseinheiten, sog. *Service Level Agreements (SLA)*[13], gegen gesteuert werden. Damit kann eine Kosten-/Leistungsverrechnung verbunden und Kostentransparenz geschaffen werden, die zur Steuerung notwendig sind. Solche Regelungen sind bei einer dezentralen IT-Versorgung i. d. R. nicht durchzusetzen.

Die IT-Versorgung der Verwaltung (*Verwaltungs-IT*) wurde trotz ihrer Bedeutung und spezifischen Aufgabe nicht einer besonderen Betrachtung unterzogen; denn die Verwaltung ist nur eine Fachabteilung/Fachbereich mit spezifischen Anforderungen und zentraler Bedeutung.

[11] auf einer Tagung des Bad-Wiesseerkreises der Fachhochschulleitungen wurde die Zusammenfassung der IT-Versorgung unter der Bibliotheksleitung diskutiert.
[12] Moog, H.: a.a.O. S. 81
[13] z. B. : Tiemeyer, E.: Handbuch IT-Management , Hanser (2006), München;
Klaus Wannemacher/Horst Moog/Bernd Kleimann (Hrsg.): ITIL goes University?
Serviceorientiertes IT-Management an Hochschulen http://www.his.de/pdf/pub_fh/fh-200808.pdf

Organisationsstruktur

In der Hochschulleitung (Präsidium) sollten die Belange der IT durch einen kompetenten *Chief Information Officer (CIO)*[14] vertreten sein, dem die drei Bereiche Multimedia-Dienste, Bibliothek und IT-Dienste mit der Verwaltungs-IT hierarchisch unterstellt sind. (Abb. 10)

Zur Koordination übergreifender Belange sollte ein IT-Lenkungsauschuss etabliert werden, in den alle Fakultäten ihre IT-Beauftragten entsenden (Abb. 10) und der vom CIO moderiert wird.

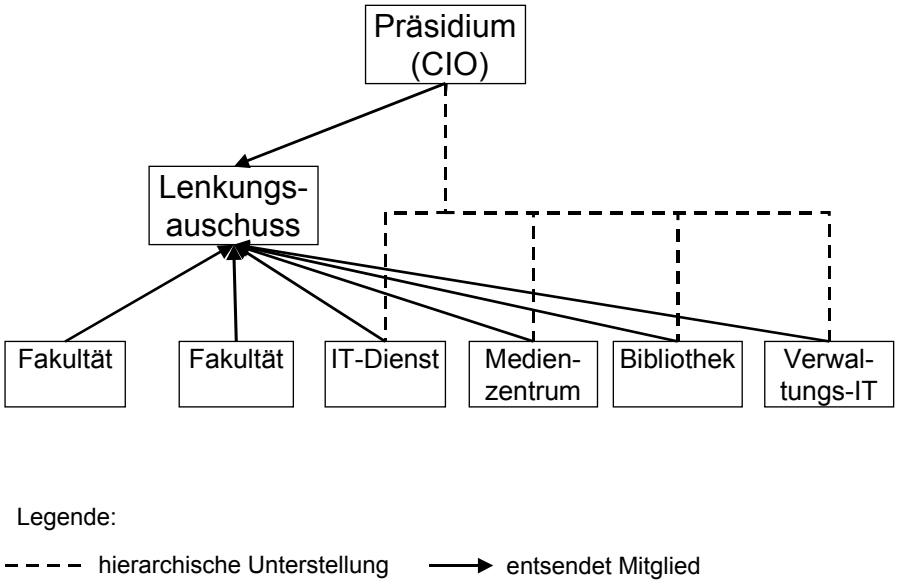


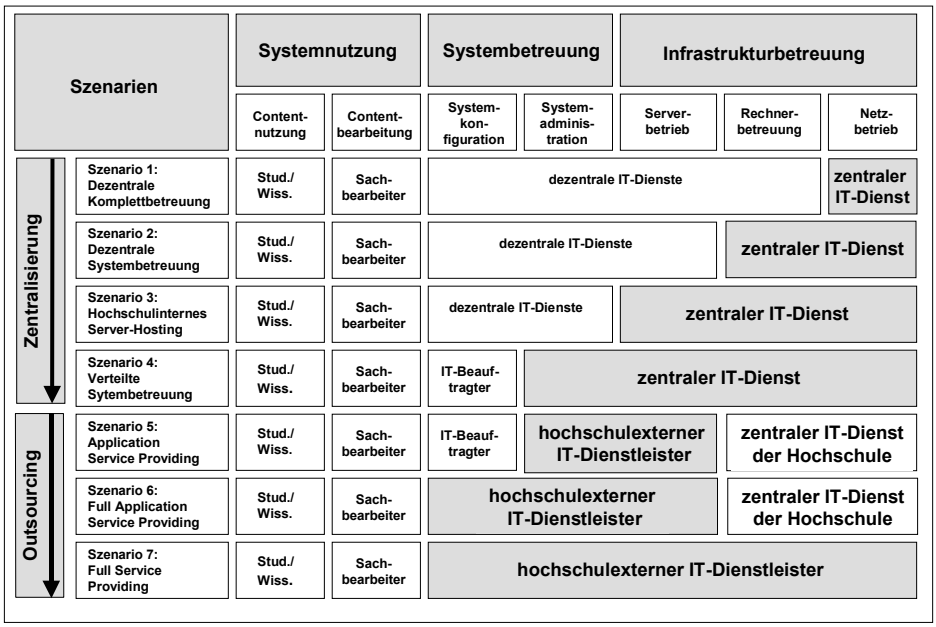
Abb. 10: Lenkungsausschuss

Mit der Einführung von *Service Level Agreements (SLA)* wird es notwendig und möglich, die vereinbarte Leistungserbringung transparent zu organisieren und möglichst messbar zu machen. Dazu muss für ein professionelles IT-Servicemanagement[15] ein Service-Desk etabliert werden, das als zentrale Kontaktstelle zwischen IT-Dienstleistern und Benutzern fungiert und alle Anforderungen an die IT-Dienstleistungseinheiten entsprechend ITIL[16] über einen wohldefinierten *Customer Relationship (CRM)* Prozess verwaltet und steuert.

Nur wohldefinierte Prozesse werden langfristig eine Professionalisierung der IT-Versorgung ermöglichen. Sie sind die Voraussetzung für ein mögliches Outsourcing[17]. Wenn Outsourcing angestrebt wird, muss geregelt werden, wo die Schnittstelle zwischen der Hochschule und dem externen IT-Dienstleister liegen soll. Hierbei sind die verschiedensten Varianten denkbar (s. Abb. 11).

[14] z. B.: Krcmar, H.: Informationsmanagement ; 4. Aufl.; Springer (Berlin) 2005; S. 303
[15] z. B.: Fröschle, H.-P.; Zarnekow, R.: Wertorientiertes IT-Servicemanagement; HMD 264; dpunkt.verlag (Heidelberg) 2008
[16] z. B.: Olbrisch, A.: ITIL kompakt und verständlich; 2. Aufl.; vieweg (Wiesbaden) 2004
[17] z. B.: Krcmar, H. a.a.O.

Bei der Auswahl muss berücksichtigt werden, dass das Wissen über ausgelagerte Tätigkeiten in den Hochschulen i. d. R. verloren geht und dass die Kommunikation und die Kooperation mit externen Dienstleistern mehr Planung, Kontrolle und regelbasiertes Handeln erfordert.



[modifiziert nach: Moog a. a. O.]

Abb. 11: Alternative Betreuungsszenarien

Schlussbemerkung

Einige Hochschulen in Deutschland haben bereits ihre IT-Versorgung reorganisiert. Dabei wurden unterschiedliche Organisationsmodelle implementiert[18], die Rücksicht nehmen auf unterschiedliche Hochschulstrukturen und –größen. Auch die von den Hochschulen angebotenen Fächer (z. B. Informatik, naturwissenschaftliche und technische Fächer) beeinflussen die Organisation der IT-Versorgung.
Bei vielen Hochschulen steht die Reorganisation noch aus[19]. Sie bedarf starker Promotoren und wird viel Zeit erfordern sowie erhebliche Kosten verursachen. Einsparungen werden nicht sofort zu erzielen sein, da die Entlastungen der Nutzer nicht sofort und messbar zu Kostenreduktionen führen werden und weil sich in vielen Hochschulen erst eine Kultur, eine Bereitschaft zu geplanter Inanspruchnahme professioneller IT-Dienstleistungen etablieren muss.

Um jedoch langfristig IT erfolgreich werthaltig einzusetzen, bedarf es einer Professionalisierung der IT-Versorgung und diese ist in vielen Hochschulen nur mit einer veränderten IT-Organisation erreichbar.

[18] Moog, H.: a.a.O.
[19] Campus Innnovation und V. Konferenztag Studium und Lehre 20.-21. Nov. 2008 a. a. O.

Integration in Controllingsystemen

Prof. Dr. Sven Piechota

Auf einen Blick

Controllingsysteme sind die Umsetzung theoretischer Controllingkonzepte in unternehmensindividuelle, rationale Führungssysteme. Controllingsysteme bestehen ihrerseits aus verschiedenen Teilsystemen, die untereinander und nach außen zu anderen Führungssystemen integriert werden müssen. Integration bedeutet, aus verschiedenen Teilsystemen ein inhaltlich kohärentes Gesamtsystem zu gestalten. Grundsätzlich kann man Konzepte der technischen Integration von organisatorisch-konzeptionellen Integrationsarten unterscheiden. Bei der technischen Integration in Controllingsystemen ist der einfachere Weg, die Integration in technisch-physikalischer Hinsicht, bei der neue Technologien die Zusammenführung der Teilsysteme übernehmen, von der Integration mittels Metadaten zu unterscheiden. Bei der organisatorisch-konzeptionellen Integration gibt es vier Integrationsarten: die klassifizierende Integration, die definitorische, die explikative und die konzeptionelle Integration. Für alle vier Arten der Integration werden wichtige Anwendungsbeispiele wie etwa die Integration der Finanz- und Erfolgsplanung, die Integration in mehrdimensionalen Performancemanagementsystemen und die konzeptionelle Integration in der **w**ertorientierten Unternehmensführung aufgezeigt.

1 Controllingsysteme

Unter einer **Controllingkonzeption** versteht man die theoretische Ausrichtung, Zielorientierung und Fokussierung des Controllings. In der deutschen Betriebswirtschaftslehre haben sich verschiedene Schulen des Controllings etabliert, die verschiedene Controllingkonzeptionen entwickelt haben. Die konkrete Umsetzung der Controllingkonzeption in der Unternehmenspraxis nennt man ein **Controllingsystem**. Die verschiedenen Schulen sowie ihre begründenden Theorievertreter sind in Abb.1 im Überblick dargestellt.

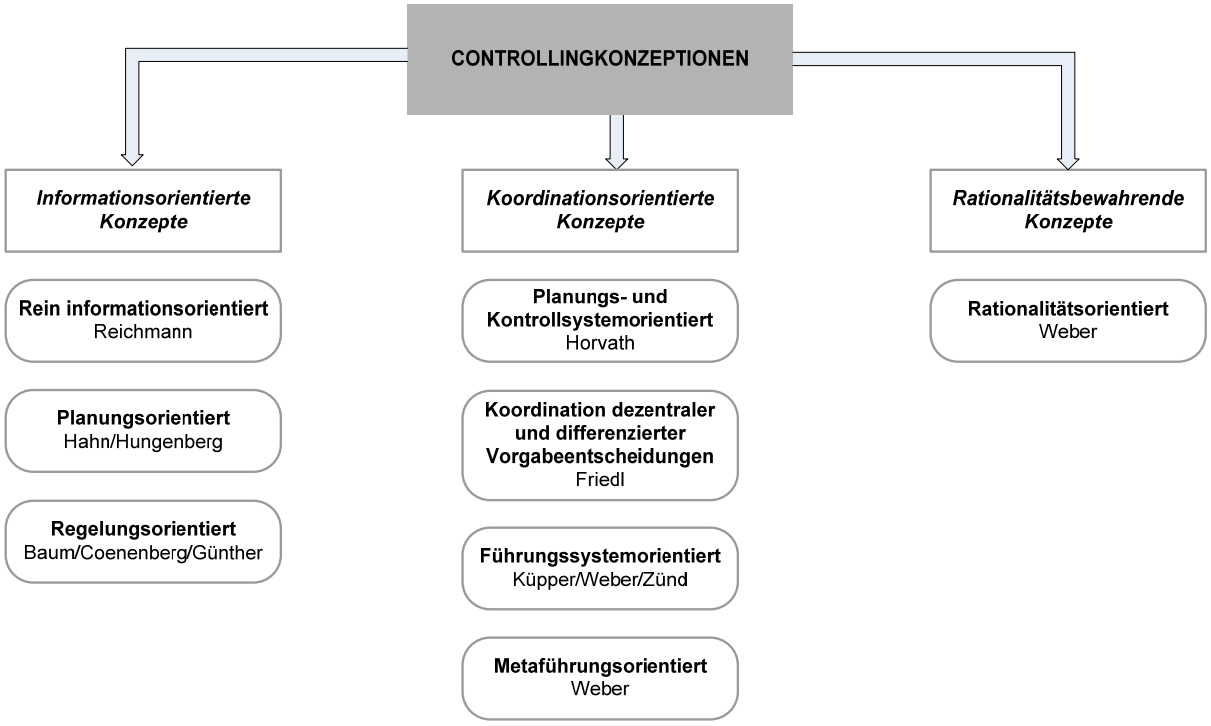


Abbildung 1: Controllingkonzeptionen

Diese Darstellung entspricht zugleich einer von links nach rechts jünger werdenden zeitlichen Entwicklung. Der Schritt von einer grundlegenden, die theoretische Fundierung des Controllings bestimmenden Controllingkonzeption zu einem Controllingsystem erfolgt durch die Einbeziehung der konkreten Umfeld- und Unternehmensbedingungen des Unternehmens, das „Führung durch Controlling“ umsetzen will. Vereinfachend kann man sagen:

Controllingkonzeption + betriebliche Einsatzbedingungen des Controllings = Controllingsystem.

Die theoretische Grundauslegung sowie die konkreten Anwendungsbedingungen der Unternehmen bestimmen somit die in der Realität vorzufindenden Controllingsysteme, die den Führungsbereich von Unternehmen maßgeblich gestalten. Allen oben genannten Controllingkonzeptionen ist gemein, dass sie die Führung in den Unternehmen unter dem Blickwinkel einer **rationalen Sachorientierung** sehen und nicht im Lichte der personalorientierten Durchsetzung des Willens des Führenden im Sinne eines Einwirkens auf den Geführten.
In Abb.2 ist visualisiert, dass Controllingsysteme also alle Führungsphasen von der Willensbildung, der Willensdurchsetzung bis zur Ergebniskontrolle von

Führungshandlungen zum Gegenstand haben. Controllingsysteme beinhalten als Teilmodule

- Zielsetzungssysteme
- Planungssysteme
- Entscheidungssysteme
- Dokumentationssysteme
- Systeme der Erfassung und Ermittlung führungsrelevanter Daten
- Bewertungssysteme, die rationale Vergleiche ermöglichen
- Methodensysteme, die rationale betriebswirtschaftliche Diagnostik ermöglichen.

.

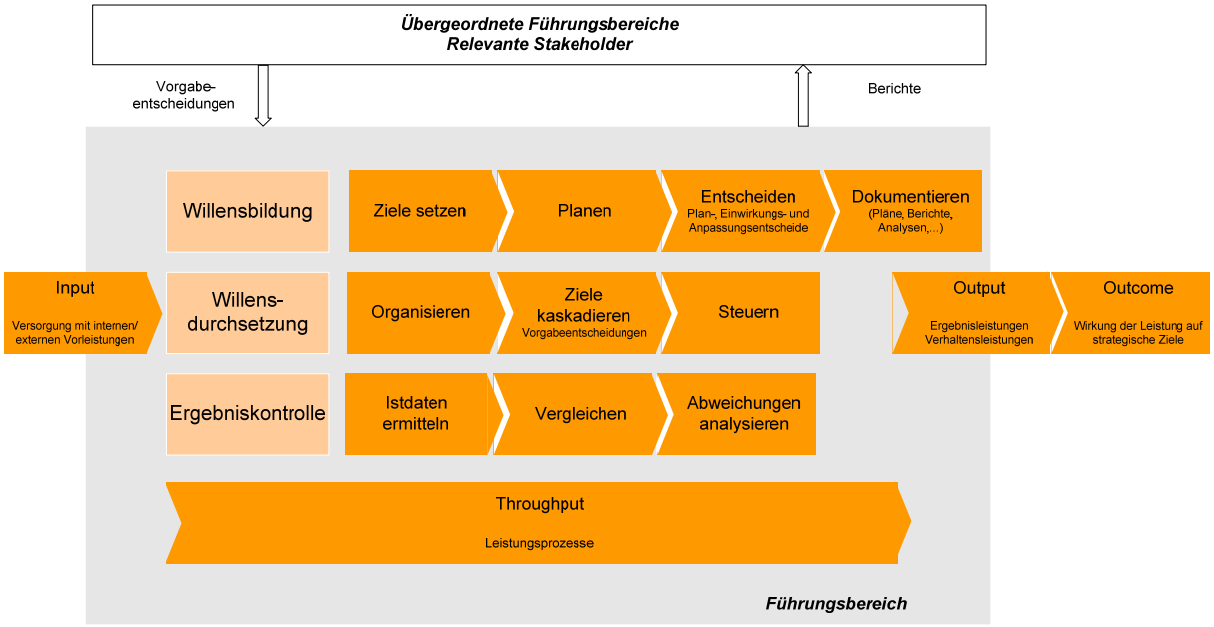


Abbildung 2: Controllingsysteme im Führungszusammenhang

2 Integration in Controllingsystemen

2.1 Technische Integration

Das Wort „Integration“ leitet sich vom lateinischen Ausdruck *integer* ab, der mit „unberührt, unversehrt, ganz“ übersetzt werden kann. Wer Dinge integriert, stellt somit ein Ganzes her aus Elementen, die zuvor nicht zusammengehörten: aus zwei Unternehmen entsteht eines, vorher durch Grenzen und Aktionshemmnisse getrennte Wirtschaftsräume werden zu einem großen gemeinsamen Wirtschaftsraum, Menschen unterschiedlicher Kulturkreise und Nationalitäten werden eine neue, zusammengehörende Gemeinschaft, Audiosysteme mit unterschiedlichen Datenformaten werden homogenisiert und so durch alle Endgeräte nutzbar.

In dem hier interessierenden Zusammenhang der Integration in Controllingsystemen kann die Integration von Teilsystemen der Planung, Berichterstattung und betriebswirtschaftlichen Analyse auf unterschiedliche Weise durchgeführt werden. Grundsätzlich kann man Konzepte der technischen Integration von organisatorisch-konzeptionellen Integrationsarten unterscheiden.

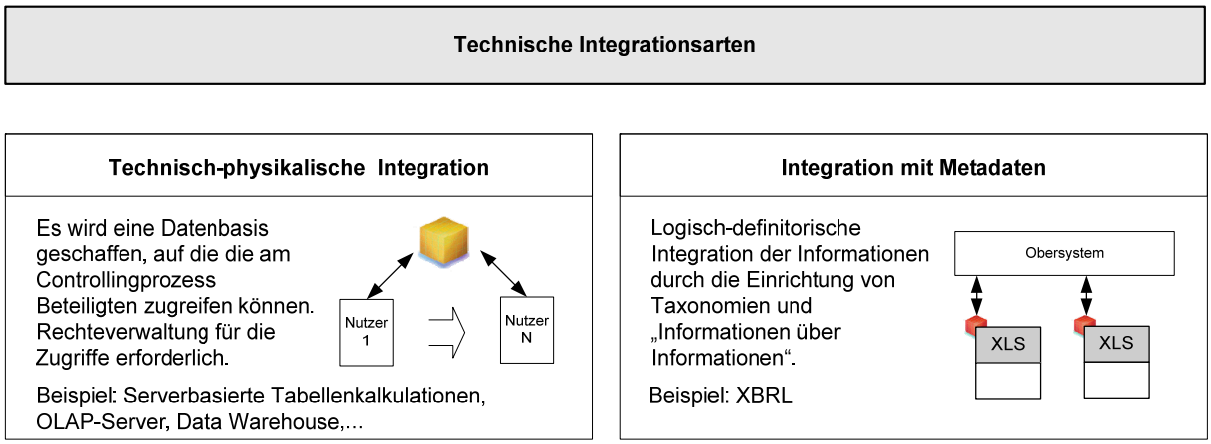


Abbildung 3: Technische Integrationsarten in Controllingssystemen

Zu der Abbildung 3 sind ein paar kurze Ergänzungen erforderlich:

1. Nicht integrierte Controllingsysteme sind heute noch zahlreich anzutreffen. Sie kommen als Tabellenkalkulationsanwendungen daher, die im Controlling eine sehr weite Verbreitung gefunden haben. Der Nicht-Integrationseffekt solcher Anwendungen besteht darin, dass jedes Kalkulationsblatt dezentral gespeichert wird, Änderungen nur gelegentlich in einer Versionierung vorgenommen werden und andere am Controllingprozess Beteiligte keinen Zugang zu diesen Änderungen haben. Diese Nicht-Integration äußert sich in erheblicher Ineffizienz, Fehleranfälligkeit und Informationsasymmetrien der so organisierten Controllingprozesse. Mit der Verfügbarkeit von OLAP-Technologie (Online Analytical Processing), in mehrdimensionalen Datenbanken das Zellenmaterial für Tabellenkalkulationen zentral bereitstellen, entstand die Möglichkeit, durch eine technische Integration der controllingrelevanten Basisdaten eine dringend notwendige Problemlösung der zuvor nicht integrierten Prozesse bereit zu stellen. In den beginnenden 90er Jahren des letzten Jahrtausends entstand eine sehr aggressiv geführte Diskussion, ob vieldimensionale OLAP-Systeme oder relationale Data Warehouses die Kerntechnologie des Integrationsfortschrittes seien. Eigentlich war die Antwort schon evident: Controllingsysteme sind überwiegend modellgetriebene Steuerungssysteme. Das Modell besteht in der Art der Erlös- und Kostenrechnung, des Investitionskalküls, der Finanzrechnung usw.. Deswegen ist eine Technologie, die auf die Abbildung solcher Modellstrukturen in „Dimensionen" ausgelegt ist, grundsätzlich zweckgerecht. Data Warehouses sind eine Repräsentationsform von Daten, die sich mit der Abbildung und Handhabung solcher Strukturen schwer tut und eher geeignet ist, explorative Fragen an den Datenbestand zu beantworten: „Wer kauft unsere Produkte", „Wie werden unsere Produkte gekauft?" sind typische Fragen, die mit den entsprechenden Analyseformen bestens mit Data Warehouses beantwortet werden können. Allerdings: solche Fragen sind eher im Bereich der betrieblichen Marktforschung

angesiedelt und nur selten in besonderen Analysesituationen für Controller wichtig. Mit diesen Überlegungen ist der Siegeszug der OLAP-Datenbanken im Controlling erklärbar. Eine neuer Trend ist das Auftauchen von Tabellenkalkulationsservern: mit Hilfe von neuen Mechanismen werden sehr viele strukturierte Spreadsheets auf zentralen Servern, meist Internetservern zur Verfügung gestellt, die dann einen dezentralen Zugriff von Informationsnutzern auf die Tabellenkalkulationen unter einem Webbrowser zulassen. Allen technisch-physikalischen Integrationsformen ist gemeinsam, dass sie einen neuen, meist redundanten Datenbestand schaffen, auf den die Controllingsysteme zugreifen können. Für wichtige Controllingsysteme wie Planungen ist es dabei unverzichtbar, dass der Anwender auf den zentralen Datenbestand per Eingabe zurück schreiben kann und die Datenbasis dann selbstständig eine Rekalkulation des Planungsmodells durchführt. Die Performance von solchen rechnenden OLAP-Servern hängt im Wesentlichen von dieser Kalkulationsfähigkeit des Servers und der rechnerischen Komplexität des Planungsmodells ab und nicht — wie immer noch teilweise behauptet wird — von der Größe des Datenmodells.

2. In der jüngeren Vergangenheit verbreitet sich zunehmend die hier als „Integration mit Metadaten“ bezeichnete technische Integration von Controllingsystemen. Zahlreiche Unternehmen, die unter die Berichtspflicht der SEC in den USA fallen, publizieren ihre Abschlussinformationen in dem Format der XBRL-Taxonomie.

> On September 29, the SEC announced a $54 million investment to update its 20-year-old EDGAR database of corporate regulatory filings and turn it into an interactive database that uses XBRL. The SEC has promoted the program as a perk for investors, who would be able to easily compare several companies' financial data on their desktops rather than retyping or printing out forms, but it only works if all those companies being compared have used XBRL to tag their data.
>
> Quelle: CFO Magazine, 4.10.2006

Taxonomien sind verkürzt erläutert fest definierte Datenstrukturen, die einem Text oder einem Spreadsheet und seinen Zellen zugeordnet werden. Diese Datenstrukturen sind von hierfür zuständigen Gremien erlassen und quasi allgemeinverbindlich. Solche Taxonomien können natürlich auch konzernintern verbindlich erklärt werden. Es gibt also mehrere Ebenen der Verbindlichkeit von Taxonomien: die höchste Allgemeinverbindlichkeit haben solche, die von unternehmensübergreifenden Organen wie etwa den XBRL-Gremien fixiert werden. Innerhalb von Konzernen können intern Taxonomien festgelegt werden, Unternehmensübergreifend können Taxonomien für vertriebliche Marktdaten oder etwa Benchmark im einzelnen Anwendungsfall vereinbart werden. In der Konsequenz wird über dieses Format sowohl eine physikalische Speicherung unnötig wie auch eine Fixierung der Controllingapplikation: durch einfachen Datenimport können Informationen importiert und weiteren Controllingsystemen bereitgestellt werden. Die Kosten für solche Taxonomien sind gering und die Anwendungsflexibilisierung erheblich. Das für die Taxonomien verantwortliche Organ (www.xbrl.org) erweitert ständig die Taxonomien für das interne, externe und Performancereporting, aktuell entstehen die ersten technischen Taxonomien für den Bereich des Engineering. Wenngleich Prognosen über die zukünftige Penetration im IT- Bereich schwer fallen (was wurde nicht schon alles als die „Killing Technology“ bezeichnet?), die Protagonisten von XBRL sind

einflussreich, aktiv und scheinen entschlossen, auch lange Innovationszyklen bis zum Diffusions- und Markterfolg durchstehen zu wollen.

2.2 Organisatorisch-konzeptionelle Integrationsformen

Bei organisatorisch-konzeptionellen Integrationsarten steht nicht die technische Perspektive, sondern die ablauforganisatorische und logisch-konzeptionelle Perspektive im Mittelpunkt der Problembetrachtung.

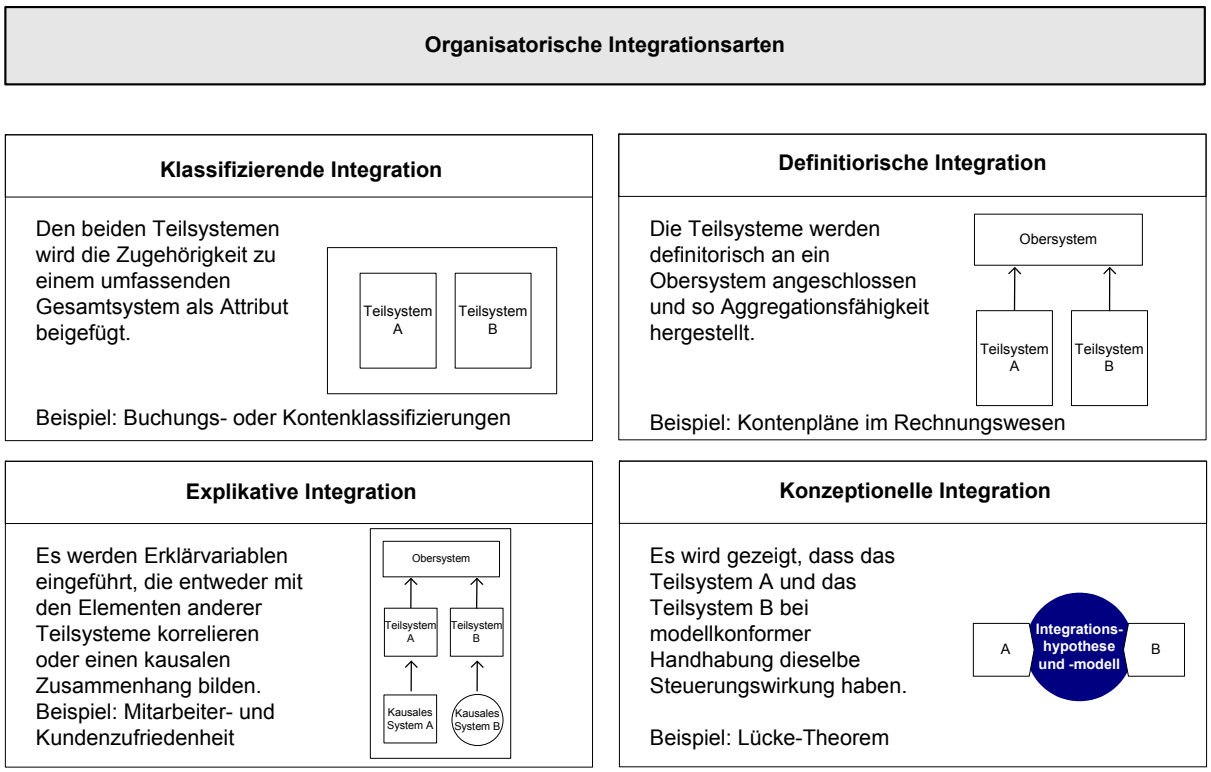


Abbildung 4: Organisatorische Integrationsarten in Controllingsystemen

Folgende Segmentierungen dieser Integrationsformen werden vorgeschlagen:

1. Bei einer **klassifizierenden Integration** wird den Elementen der beiden Teilsysteme die Zugehörigkeit zu einem übergeordneten System gewissermaßen als Merkposten (Attribut) beigefügt. Ein wichtiges Beispiel für diese Integrationsart ist die direkte Ermittlung des Total Cash Flows (s. Abb.5). In dem Beispiel wird gezeigt, wie die einzelnen Transaktionen eines Geldkontos zeilenweise mit einem Attribut gekennzeichnet werden, das beschreibt, ob sich diese Transaktion dem Invest Flow, dem Finance Flow oder dem Cash Flow zuzuordnen ist. Die Entscheidung über diese Klassifizierung kann automatisch anhand des Gegenkontos identifiziert oder manuell zugesteuert werden. Wenn diese Klassifizierung abgeschlossen ist, können die gefilterten Aggregate der einzelnen Klassen (Invest-, Finance- oder Cash Flow) ermittelt werden und bilden so die Verbindung zwischen der Bilanz- und Finanzrechnung.

2. Würde in dem Beispiel eine **definitorische Integration** auf der Ebene der einzelnen Transaktion gewählt werden, dann müsste der Buchungsvorgang in der Buchhaltung diese neue Definitionswelt widerspiegeln. Beispiel: wenn in der Definitionswelt der doppelten Buchhaltung Zielverkäufe abgebildet werden, dann erfolgt zunächst die Erfassung des Erfolges in der Erfolgsrechnung („per Debitoren an Umsatzerlöse“) und bei Zahlungseingang ein Aktivtausch („per Geldkonto an Forderungen“). Wenn die Finanzrechnung definitorisch in das bestehende System der doppelten Buchhaltung integriert würde, dann würde der Erfolgserfassung („per Umsatzerlöse an Debitoren“) beim Zahlungseingang eine Erfassung in einer (definitorisch gleichberechtigten) Finanzrechnung („per Umsatzeinzahlungen an Debitoren“) folgen. Die beiden Teilsysteme Erfolgs- und Finanzrechnung würden mit eigenen Abschlussbuchungen in die Bilanzrechnung integriert („per Erfolgsrechnung an Bilanzrechnung“ und „per Bilanzrechnung an Finanzrechnung“, wenn jeweils Gewinne oder Finanzüberschüsse erzielt wurden). Freilich hat sich diese Form der Integration auf Transaktionsebene nicht durchgesetzt, weil sich die Kontierung im Rechnungswesen hierbei anders dargestellt hätte und ein weitflächiges Umlernen der Mitarbeiter im Rechnungswesen erforderlich gewesen wäre. Deshalb erfolgte die definitorische Integration der drei Rechnungswesenperspektiven auf der Ebene aggregierter Konten und wird „indirekte Ermittlung der Cash Flows“ genannt, die im folgenden Kapitel kurz beschrieben wird.

Datum	Text	AB	Betrag Inflow	Betrag Outflow	EB	Zuordnung Finanzrechnung	Summe CF
		100.000,00					
	Zahlungseingang Forderungen		2.500.000,00			CF	2.500.000,00
	Zahlungseingang Forderungen		100.000,00			CF	100.000,00
	Zahlungseingang Forderungen		1.250.000,00			CF	1.250.000,00
	Zahlungseingang Forderungen		250.000,00			CF	250.000,00
	Zahlungseingang Forderungen		3.500.000,00			CF	3.500.000,00
	Zahlungseingang Forderungen		2.500.000,00			CF	2.500.000,00
	Zahlungseingang Forderungen		1.700.000,00			CF	1.700.000,00
	Erhaltene Anzahlungen		75.000,00			CF	75.000,00
	Erhaltene Anzahlungen		125.000,00			CF	125.000,00
	Materialkauf			1.125.000,00		CF	-1.125.000,00
	Materialkauf			1.425.000,00		CF	-1.425.000,00
	Materialkauf			1.800.000,00		CF	-1.800.000,00
	Materialkauf			750.000,00		CF	-750.000,00
	Überweisung Personalkosten			800.000,00		CF	-800.000,00
	Überweisung Personalkosten			800.000,00		CF	-800.000,00
	Überweisung Personalkosten			800.000,00		CF	-800.000,00
	Überweisung Personalkosten			1.090.000,00		CF	-1.090.000,00
	Überweisung Mieten			150.000,00		CF	-150.000,00
	Überweisung Mieten			150.000,00		CF	-150.000,00
	Überweisung Mieten			150.000,00		CF	-150.000,00
	Überweisung Mieten			150.000,00		CF	-150.000,00
	Überweisung Versicherungen			250.000,00		CF	-250.000,00
	Barauslagen			70.000,00		CF	-70.000,00
	Spesen			180.000,00		CF	-180.000,00
	Geleistete Anzahlungen			40.000,00		CF	-40.000,00
	Geleistete Anzahlungen			40.000,00		CF	-40.000,00
	Vorauszahlung Dividende Tochtergesellschaft		120.000,00			CF	120.000,00
						Summe CF	**2.350.000,00**
	Einzahlung aus Verkauf AV		100.000,00			IF	100.000,00
	Kauf Maschine 4711			200.000,00		IF	-200.000,00
						Summe IF	**-100.000,00**
	Ausschüttung 1			35.000,00		FF	-35.000,00
	Ausschüttung 2			35.000,00		FF	-35.000,00
	Entnahme			50.000,00		FF	-50.000,00
	Nachschuss		100.000,00			FF	100.000,00
	Tilgung Kredit 4712			75.000,00		FF	-75.000,00
						Summe FF	**-95.000,00**
	Summe	100.000,00	12.320.000,00	10.165.000,00	2.255.000,00		2.155.000,00

Abbildung 5: Direkte Cash Flow-Ermittlung als Beispiel einer klassifizierenden Integration

3. Bei einer **explikativen Integration** wird die definitorische Logik um Erklärvariablen ergänzt. Die unabhängigen Erklärvariablen stehen zu den abhängigen Variablen in einem qualitativen Zusammenhang, der entweder durch Korrelation empirisch gefunden oder durch Kausalität bestimmt wird. Während Korrelationen statistische Zusammenhänge von Beobachtungen aufzeigen, die ansonsten keinen direkten Handlungszusammenhang aufweisen müssen („Westliche Flugrichtung der Störche über Aurich korreliert mit der Geburtsrate in

Husum“), wird bei kausalen Zusammenhängen gerade dieser Handlungsbezug gefordert: nur wenn die unabhängige Variable Ursache (causa) der Veränderung der abhängigen Variablen ist, kann Kausalität zwischen den beiden Variablen unterstellt werden („Freigang der Matrosen in Husum korreliert und ist kausal für den (späteren) Anstieg der Geburtenrate in Husum“). Wenn in Controllingsystemen Zusammenhänge zwischen qualitativen Variablen („Hohe Mitarbeiterqualifikation erklärt hohe Kundenzufriedenheit“) oder zwischen qualitativen Variablen und solchen mit Formalzielcharakter („Hohe Mitarbeiterqualifikation erklärt niedrige Prozesskosten“) unterstellt und hoffentlich empirisch bestätigt werden, dann erfolgt eine solche explikative Integration. Nachstehend werden mehrdimensionale Performance-Managementsysteme als Beispiel dieser Integrationsart erläutert.

4. Bei einer **konzeptionellen Integration** wird gezeigt, dass verschiedene Teilsysteme der Steuerung prinzipiell eine richtungsgleiche Steuerungswirkung haben. Ein bekannter Anwendungsfall sind die Verfahren des Investitionscontrollings und des Erfolgscontrollings. Entscheidungen im Investitionsbereich, die beispielsweise nach dem Kapitalwertverfahren durchgeführt werden, haben eine optimierende Wirkung auch auf die periodische Erfolgssteuerung. Später werden wir für diese Integrationsart für die wertorientierten Verfahren der Übergewinne und der Discounted Cash Flow-Methode aufzeigen.

3 Die integrierte Erfolgs-, Finanz- und Bilanzplanung als Beispiel einer definitorischen Integration in Controllingsystemen.

Wir hatten bei der Beschreibung der definitorischen Integration dargestellt, dass im realen betrieblichen Rechnungswesen eine Integration auf Transaktionsebene sachgerecht wäre, aber aus Handhabungsgründen erst auf der Konten(gruppen-)ebene des Rechnungswesens erfolgt. Diese Integration ist für den Aufbau von Controllingsystemen grundlegend, weil sie die Ebene der Erfolgssteuerung mit der strukturellen und taktischen Liquiditätssteuerung integriert. Dieser Integration kommt eine herausragende Bedeutung zu, weil in allen nicht planwirtschaftlichen und auf Privateigentum basierenden Wirtschaftssystemen neben der Überschuldung (Summe der kumulierten Verluste ist größer als das Eigenkapital) die Illiquidität als Fallierungsgrund von Unternehmen festgelegt ist. Deswegen ist eine ausschließliche Erfolgssteuerung für junge oder stark wachsende Unternehmen nicht ausreichend: hier können Planungssituationen auftreten, in denen zwar in allen Perioden Gewinne ausgewiesen werden, die Innenfinanzierungskraft aus diesen Gewinnen aber nicht ausreicht, um die Finanzierungsbedarfe aus dem Wachstum zu decken. Ohne die Bereitstellung von Eigen- oder Fremdkapital können solche Unternehmen – obwohl sie immer positive Periodenerfolge aufweisen in den Konkurs wachsen. Controllingsysteme müssen also zwingend die Integration zwischen der Erfolgs- und Finanzrechnung vorsehen, um einen relevanten Planungs- und Kontrollausschnitt der betrieblichen Realität abzubilden. In vielen Rechnungslegungsnormen wird deshalb auch neben der Veröffentlichung von Erfolgsrechnung und Bilanz die Publikation eines Cashflow-Statements verlangt.

Diese Integration erfolgt auf der Grundlage der Ordnungssysteme des Rechnungswesens. Auf der höchsten Ebene gilt hier, dass die Summe aller aktiven Kontensalden der Summe der passiven Kontensalden entspricht („Bilanzsumme Aktiva = Bilanzsumme Passiva“). Unterstellt man form- und strukturgleiche Anfangs- und Endbilanzen, dann gilt diese Aussage auch für die Veränderungen der aktiven und passiven Bilanzsummen und ihrer erklärenden aktiven und passiven Bestandteile:

(1) $A = P$

(2) $\Delta A = \Delta P$

mit A=Aktiva, P=Passiva, ΔA,ΔP=Veränderung Aktiva und Passiva zwischen zwei Abschlüssen

Eine Mindestuntergliederung der Bilanzsumme (genauer: ihrer Veränderung) erfolgt aktivisch in die Veränderung des Anlagevermögens (ΔAV), des nichtmonetären Umlaufvermögens (ΔUV^{nm}) sowie des monetären Umlaufvermögens (ΔUV^{m}), auf der Passivseite wird zwischen der Veränderung des Eigen- und des Fremdkapitals (ΔEK, ΔFK) unterschieden:

(3) $\Delta AV + \Delta UV^{nm} + \Delta UV^{m} = \Delta EK + \Delta FK$

Durch einfache Umstellung ergibt sich:

(4a) $\Delta UV^{m} = TCF = \Delta EK + \Delta FK - \Delta AV - \Delta UV^{nm}$ mit TCF=Total Cash Flow bzw.

(4b) $\Delta UV^{m} = (+PAT - LAT + EZ^{EKA} - AZ^{EKR} - AZ^{DIV}) + \Delta FK - \Delta AV - \Delta UV^{nm}$

mit PAT,LAT= Profit, Loss after Tax

(4c) $TCF = +PAT - LAT + [EZ^{EKA} - AZ^{EKR} - AZ^{DIV} + \Delta FK - \Delta AV - \Delta UV^{nm}]$

Die Gleichungen 4a – 4c können als die **Basisgleichungen integrierter Erfolgs- und Finanzsteuerungen** angesehen werden. Auf der linken Seite wird als zu erklärende Größe die Veränderung des monetären Umlaufvermögens, auch Total Cash Flow genannt, ausgewiesen. Auf der rechten Seite sehen wir als Erklärgrößen die Veränderungen der Positionen des Eigen- und Fremdkapitals, die Veränderungen des Anlage- und nichtmonetären Umlaufvermögens werden subtrahiert. In Gleichung (4b) wird die Veränderung des Eigenkapitals weiter definitorisch detailliert und die periodischen Erfolge (PAT „Profit After Tax“ und LAT „Loss After Tax“) von den Aufnahmen und Rückzahlungen von Eigenkapital sowie den Dividendenzahlungen getrennt. In Gleichung 4c sieht man dann, dass die Finanzrechnung, abgebildet als Total Cash Flow sich aus der Erfolgsrechnung (PAT bzw. LAT) erklärt und die Bilanzpositionen die Differenzen zwischen Erfolgs- und Finanzrechnung aufnehmen. Die theoretische Grundlegung der Bilanz ist folglich dynamisch: sie stellt den Speicher zwischen Erfolgs- und Finanzrechnung dar. Im Anhang ist das hier dargestellte Integrationsmodell noch weiter differenziert und in Kontenform dargestellt. Dort wird deutlich erkennbar, dass die Bilanzrechnung neben den Anfangs- und Endbeständen einerseits Veränderungen aus der Finanzrechnung, andererseits Veränderungen aus der Erfolgsrechnung aufnimmt:

- In der Bilanz werden Erfolgsvorgänge gespeichert, die entweder noch nicht zu Zahlungsvorgängen geführt haben (Debitoren, Kreditoren, Bildung von Rückstellungen) oder eine Nachverrechnung der Erfolgsrechnung erfolgt (Abschreibungen, Materialverbräuche nach Materialbeschaffungen).

- Weiter werden in der Bilanz Zahlungsvorgänge gespeichert, denen eine Erfolgsverrechnung zeitlich folgt (transitorische Rechnungsabgrenzungspositionen, Verbräuche von zuvor gebildeten Rückstellungen).
- Letztlich nimmt die Bilanzrechnung Zahlungsvorgänge auf, die nie in der Erfolgsrechnung aufscheinen, beispielsweise Kapitalaufnahmen und -tilgungen.

Durch weitere Aufgliederung der Terme in den eckigen Klammern von Gleichung 4c erhält man die geläufigen Unterkategorien des TCF, den Cash Flow, in dem wiederum die Erfolgsrechung eingeschlossen ist, den Finance Flow und den Invest Flow :

(5) $$TCF = (+PAT - LAT + \Delta FK^{RÜ} + \Delta FK^{uvz,Kreditoren} - \Delta UV^{nm,RUF} - \Delta UV^{nm,Debitoren}) + (EZ^{EKA} - AZ^{EKR} - AZ^{DIV} + \Delta FK^{vz}) + (-\Delta AV^{FA} - \Delta AV^{SA})$$

(6) $$CashFlow = +PAT - LAT + \Delta FK^{RÜ} + \Delta FK^{uvz,Kreditoren} - \Delta UV^{nm,RUF} - \Delta UV^{nm,Debitoren}$$

(7) $$InvestFlow = -\Delta AV^{FA} - \Delta AV^{SA}$$

(8) $$FinanceFlow = EZ^{EKA} - AZ^{EKR} - AZ^{DIV} + \Delta FK$$

Algorithmisch ist diese Integration sehr anspruchsarm, da sie durch schlichte Addition und Subtraktion zu bewältigen ist und sehr gut in Tabellenkalkulationen abgebildet werden kann. Planungsrechnungen sind immer dann als integrierte Rechnungen aufzubauen wenn sie die Grundlage einer wertorientierten Unternehmensführung nach den Discounted Cash Flow-Methode (DCF) sein sollen, weil das Basiselement eines Unternehmenswertes nach der DCF-Methode der Free Cash Flow ist, der die Summe aus Cash Flow und Invest Flow darstellt. Eine lediglich auf einer Erfolgsplanung basierende Unternehmensplanung macht deshalb die grundlegenden Elemente einer wertorientierten DCF-Rechnung nicht transparent.

4 Mehrdimensionale Performancemanagementsysteme als Beispiel der explikativen Integration von Controllingsystemen

Explikative Integrationen gehen weiter als definitorische Integrationen: die Ableitung definitorischer Integrationen erfolgt nach dem Prinzip der **Deduktion** „vom Allgemeinen zum Besonderen“. So integrierte Controllingsysteme können eigentlich nicht falsch sein, sondern entweder unlogisch (dann verstoßen sie gegen logische Konventionen) oder unzweckmäßig (dann erfüllt die Spezialisierung nicht den beabsichtigten Erklärungszuwachs). Definitorisch integrierte Controllingsysteme sind deshalb effizient und weit verbreitet, weil die Definitionen in einem dezentralen Führungszusammenhang, beispielsweise bei Tochtergesellschaften, dezentralen Projektleitern oder Niederlassungen gut durchsetzbar sind und ihre Einhaltung wirksam kontrolliert werden kann.

Explikative Integrationen dagegen werden nach dem Prinzip der **Induktion** aufgebaut: hier wird von der Beobachtung zahlreicher einzelner Phänomene hypothetisch auf einen erklärenden Zusammenhang gefolgert. Explikative Integrationen können deshalb immer nur solange richtig sein, bis sie signifikant widerlegt werden. Natürlich gibt es Zusammenhänge, die eine empirisch gute Bestätigung erfahren haben, allerdings darf die Controllingpraxis von der Wissenschaft nicht allzu viel Hilfe erwarten. Zahlreiche empirische Studien sind in ihrem definitorischen Kern nicht vergleichbar, so dass man nur unsicher sagen kann, ob eine kausale Hypothese von einer empirischen Studie wirklich bestätigt oder abgelehnt wurde.

Für Controllingsysteme hat diese Unterscheidung Bedeutung mit einer geradezu epidemischen Verbreitung mehrdimensionaler Performancemanagementsysteme (Balanced Scorecard (BSC) et aliud) erlangt. Die Dimensionen dieser Leistungssteuerungssysteme sind Klassen strategischer Ziele, die als zusammengehörig empfunden werden. Bei der BSC sind dies standardmäßig die Dimensionen Finanzen, Kunden, Prozesse und immaterielle Ressourcen. In der BSC wird zwingend verlangt, in so genannten Ursache-Wirkungsketten (UWK) die unterstellten Beziehungen zwischen den strategischen Zielen der verschiedenen Kategorien abzubilden. Die Grundhypothese dafür lautet: (1) Gute immaterielle Ressourcen führen zu (2) bedarfsgerechten und fehlerfreien Prozessen, die (3) Kundenzufriedenheit bewirken, was langfristig zu (4) finanziellen Erfolgen führt.

Alle Berichte von praktischen BSC-Einführungen thematisieren, dass sich die Unternehmenspraxis mit der Einrichtung der UWK schwer tut, vielleicht auch, weil methodische Hilfen in diesem Problembereich von den Protagonisten der BSC nur zurückhaltend publiziert werden. Nach der hier vorgestellten Unterscheidung zwischen deduktiver und induktiver Zielableitung kann man den pragmatischen Vorschlag machen, die Pfeilrichtungen zwischen den Dimensionen 4,3 und 2 in der Findungsphase einer BSC einfach umzudrehen und deduktiv vorzugehen. Hat man beispielsweise Umsatzwachstum auf der Ebene der finanziellen Ziele auserkoren, dann kann man nunmehr deduktiv fragen, wie man Umsatzwachstum auf der Kundenebene bewerkstelligen kann. Zunächst liegt nahe, Umsatzzugewinne mit neuen Kunden zu erzielen, zum anderen kann man die Kaufsummen bei bestehenden Kunden durch vermehrte Käufe oder höherwertige Verkäufe zu erzielen. Diese gedankliche Ableitung kann man auf dem Übergang von Kundenzielen zu Prozesszielen fortsetzen, was einer transparenten Herleitung der strategischen Bedeutung von Geschäftsprozessen zugute kommt. Ab der letzten Dimension wird das deduktive Vorgehen schwierig, weil der Zusammenhang zwischen Prozessen und immateriellen Ressourcen nicht mehr definitorischer sondern hypothetischer Natur ist, also der Übergang von der Deduktion zu einer Induktion vollzogen werden muss: man vermutet, dass gut ausgebildete Mitarbeiter und Mitarbeiterinnen, sinnvolle Organisationsformen und bestimmte Informationssysteme einen positiven Einfluss auf die Effektivität und Effizienz der Prozessabwicklung haben. Zur Überprüfung muss man die unterstellten Korrelationen fortlaufend überprüfen und verifizieren.

5 Beispiele konzeptioneller Integration von Controllingsystemen

5.1 Wertorientierung als Beispiel der unternehmensinternen und -externen Integration

Bei der konzeptionellen Integration von Controllingsystemen wird eine neue Ebene der Zusammenführung unterschiedlicher Controllingteilsysteme betreten: es bleiben unterschiedliche Teilsysteme des Controllings bestehen, es erfolgt aber der Nachweis, dass die Wirkung der verschiedenen Teilsysteme in dieselbe betriebswirtschaftliche Richtung weisen, genauer: dass die Zielgrößen beider Teilsysteme zu einer übergeordneten Zielgröße komplementär wirken.

Ein wichtiges Beispiel für diese Integrationsart ist die wertorientierte Unternehmensführung. Auf der Basis der weiter oben dargestellten integrierten Finanz-, Erfolgs- und Bilanzplanung lässt sich der Unternehmenswert nach der Discounted Cash Flow- Methode (Variante Entitymethode) wie folgt ermitteln:

(9) $$UGW_{DCF/Entity} = \sum_{t=1}^{T} \frac{FCF_t}{(1+k)^t} + \frac{FW}{(1+k)^T} = \sum_{t=1}^{T} \frac{FCF_t}{(1+k)^t} + \frac{FCF_T}{i(1+k)^T}$$ bzw.

$$UGW_{DCF/Entity} = \sum_{t=1}^{\infty} \frac{FCF_t}{(1+k)^t}$$

Wobei gilt: $UGW =$ Unternehmensgesamtwert

$FCF =$ Free cash Flow zum Zeitpunkt t bzw. T

$FCF = CashFlow + InvestFlow$ (siehe oben (6) und (7))

$T =$ Ende des Detailplanungszeitraumes

$FW =$ Fortführungswert zum Zeitpunkt T

$k =$ Kalkulationszinssatz mit

$k = r_{EK} \frac{EK}{C} + r_{FK}(1-s)\frac{FK}{C}$

$C =$ Gesamtkapital

$s =$ Steuersatz

$r_{FK} =$ Fremdkapitalkostensatz nach dem gewogenen Durchschnitt

$r_{EK} =$ Eigenkapitalkostensatz nach CAPM mit

$r_{EK} = i_r + \beta(\mu(r_M) - i_r)$ mit $i_r =$ Zinssatz risikoloser Kapitalanlage,

$\beta =$ Betafaktor, $r_M =$ Marktrendite der Portfolioklasse

Formel (9) zeigt, dass die Ermittlung des Unternehmenswertes ein mehrperiodisches Kalkül ist: es werden die verschiedenen Freien Cash Flows für die Detailplanungsperiode

sowie der Fortführungswert am Ende dieser Periode ermittelt, deren summierte Barwerte ergeben den Unternehmensgesamtwert, aus dem sich nach Abzug des Marktwertes des Fremdkapitals der Wert des Eigenkapitals (Shareholder Value (SHV) ergibt:

(10) $SHV = UGW - FK_0$

Dieses Controllingsystem zur Steuerung des Unternehmenswertes eignet sich also dazu, die verschiedenen Planungsentwürfe eines Unternehmens mit ihrer Wirkung auf den Unternehmenswert zu beurteilen: für die unterschiedlichen Planungsentwürfe werden die Free Cash Flows ermittelt und hieraus die jeweils zugehörigen SHV. Durch Vergleich der unterschiedlichen SHV kann entschieden werden, ob ein Planungsentwurf den Unternehmenswert gesteigert oder gesenkt hat.

So gut sich dieser Ansatz zur Beurteilung von Planentwürfen eignet, so ungeeignet ist er zur Beurteilung der Leistung eines Unternehmens in den verschiedenen Teilperioden. Das Basisrechenelement des SHV sind die Freien Cash Flows der Teilperioden, aus denen aber definitionsgemäß nicht auf die Leistung in der Teilperiode geschlossen werden kann: so kann ein negativer FCF durch operative Fehlleistung wie auch durch eine gute operative Leistung bei gleichzeitiger Expansion der investiven Cash Flows (etwa bei der Ausdehnung eines erfolgreichen Geschäftskonzeptes) bewirkt sein. Die Nichteignung des Free Cash Flows ist im Ursprung auf dieselbe Problematik zurückzuführen, die im Investitionscontrolling zu dem Problem der laufenden Ergebniskontrolle von Investitionsmaßnahmen führt: mehrperiodische Kalküle auf der Basis von Ein- und Auszahlungen eignen sich nicht zur periodischen Überprüfung ihres Erfolgsbeitrages. Nun wissen wir seit dem Nachweis des Lücketheorems, dass die Definition eines Periodenerfolges zu der mehrperiodischen Rechnung passen muss. Die Schlüsselrolle eines Ausgleichsventils zwischen den beiden Kalkülen nimmt die Berechnung der Zinsen ein: wenn sie so erfolgt, dass die Zinsen die durch die zeitliche Inkongruenz zwischen Erfolgs- und Zahlungsgrößen entstehende Kapitalbindung ausgleicht, führen beide Kalküle zu einem identischen Ergebnis, der Kapitalwert auf der Basis der Zahlungsgrößen entspricht dann dem Kapitalwert auf der Basis der so definierten Erfolgsgrößen. Übertragen auf das hier interessierende Problem bedeutet die Anwendung des Lücketheorems, dass der UGW auf der Basis von FCF zu demselben Ergebnis führt wie die Ermittlung des UGW auf der Basis einer Erfolgsgröße, die die Zinsen auf die gesamte Kapitalbindung eines Unternehmens errechnet. Die Unterteilung zwischen Zinsen als Entgelt für die Überlassung von Fremdkapital und Gewinn als Entgelt für die Überlassung von Eigenkapital wird hier aufgehoben: Zinsen haben in ihrer Gesamtheit den Ausgleich für die gesamte Kapitalbindung zu erfüllen, Gewinne nach Abzug dieser Zinsen sind **Übergewinne** oder **ökonomische Gewinne**, die die Werterhöhung über die Anlage einer risikoadäquaten Anlage widerspiegeln:

(11) $$UGW_{\ddot{U}G} = \sum_{t=1}^{\infty} \frac{\ddot{U}G_t}{(1+k)^t} + NOA_{t_0} = \sum_{t=1}^{\infty} \frac{\ddot{U}G_t^{vorZinsen} - iNOA_{t-1}}{(1+k)^t} + NOA_{t_0} = \sum_{t=1}^{\infty} \frac{(r-i)NOA_{t-1}}{(1+k)^t} + NOA_{t_0}$$

Wobei gilt:

$\ddot{U}G_t^{vorZinsen}$ = Periodenerfolg Periode t vor Zinsen und nach (fiktiver) Unternehmenssteuer

NOA = Net Operating Assets (Kapitalbindung zu Marktwerten)

$$r = \frac{\ddot{U}G_t^{vorZinsen}}{NOA_{t-1}}$$

Die so definierten Übergewinne bestimmen barwertig summiert neben der zu Marktwerten angesetzten Kapitalbindung den UGW und werden als „Markt Value Added“ bezeichnet:

(12) $$MVA = \sum_{t=1}^{\infty} \frac{\ddot{U}G_t}{(1+k)^t}$$

(13) $$UGW_{\ddot{U}G} = MVA + NOA_{t_0}$$

Wegen des Lücketheorems gilt nun, dass der UGW nach der DCF-Methode bei gleichen Bewertungsansätzen identisch ist dem UGW nach den Übergewinnkonzeptionen:

(14) $$UGW_{\ddot{U}G} = UGW_{DCF/Entity}$$ bzw.

(15) $$\sum_{t=1}^{\infty} \frac{\ddot{U}G_t}{(1+k)^t} + NOA_{t_0} = \sum_{t=1}^{\infty} \frac{FCF_t}{(1+k)^t}$$

Diese Überlegungen sind in der nachfolgenden Tabelle in einem Beispiel noch einmal anhand eines Zahlenbeispiels gezeigt. In ein Unternehmen sind 1000 Geldeinheiten (GE) investiert, davon 600 GE als abzuschreibendes Anlagevermögen (Abschreibungsrate 20%), der Rest stellt betrieblich zu nutzendes Umlaufvermögen dar. Der Kalkulationszins beträgt 10%, das Fremdkapital zum Kalkulationszeitpunkt beträgt 600 GE.

	t1	t2	t3	Folgejahre	Summe der Barwerte
NOA /Periodenanfang	1000	1200	1320	1440	1000
Investflow	400	350	380	260	
Abschreibung	200	230	260	260	
Brutto Cash Flow nach Steuern	600	620	640	650	
ÜG vor Zinsen	400	390	380	390	
Kapitalkosten	100	120	132	144	
ÜG	300	270	248	246	
ÜG - Barwert	273	223	186	1848	2530
UGW ÜG					3530
Fremdkapital - Periodenanfang	600				-600
Shareholder Value					2930
Free Cash Flow	200	270	260	390	
Free Cash Flow - Barwert	182	223	195	2930	3530
UGW DCF					3530
Fremdkapital - Periodenanfang	600				-600
Shareholder Value					2930

Wie in (15) gezeigt, stimmen beide Unternehmensgesamtwerte (und damit auch die Shareholder Values) überein. Mit den Übergewinnen bestehen nun Messgrößen für die Leistung des Unternehmens in den Teilperioden. Bei positivem Übergewinn wurde der Unternehmenswert gesteigert, bei negativem Übergewinn wurde er gesenkt.

Die konzeptionelle Integration in der wertorientierten Unternehmensführung hat erhebliche Folgen für die hierauf aufbauenden Controllingsysteme:

1. Es kann die mehrperiodische Sicht zur Beurteilung der Entwicklung des Unternehmenswertes ergänzt werden um ein Performancemaß Übergewinn, dessen Entwicklung eine Aussage über die Leistung einer Teilperiode ermöglicht und die konzeptionell mit dem mehrperiodischen Modell integriert ist.
2. Es kann die Außensicht der Eigenkapitalgebenden Stakeholder („Wieviel MVA erwarten die Shareholder?“) systematisch mit der Innensicht des Managements („Wie hoch ist unsere derzeitige Fähigkeit, Übergewinne zu erzielen und stehen diese in Einklang mit den MVA-Erwartungen der externen Kapitalgeber?“) verbunden werden. Hierdurch ist eine transparente Beurteilung von Aktienkursen möglich.

5.2 Gefahren der konzeptionellen Integration

Bei allen Vorteilen einer gelungenen konzeptionellen Integration, sei abschließend doch noch auf die Gefahren hingewiesen, die sich durch eine unreflektierte Anwendung ergeben können. Ein illustratives Beispiel mag die Anwendung der aus der Finanztheorie stammenden CAPM-Methode bei der Festsetzung des risikoadäquaten Kalkulationszinssatzes dienen. Bei allen Integrationen in der Wertsteuerung, die dieses Verfahren anwenden, werden finanztheoretische Überlegungen in die Leistungssteuerung von Unternehmen integriert, deren Prämissen bei der betrieblichen Anwendung ausgesprochen fraglich sind. So unterstellt das CAPM-Verfahren beispielsweise vollkommene Information aller Beteiligten. Ist dies für einzelne Unternehmen plausibel anzunehmen? Ist es nicht eher so, dass Asymmetrien in der Information über den jetzigen Leistungsstand, zukünftige Chancen und Risiken zwischen externen Stakeholdern und dem Management bestehen? Wenn dies bejaht wird, ist die Integration über solche Verfahren abzulehnen, weil sie zu Fehlsteuerungen und Missinterpretationen von Controllinginformationen führen können. Für solche Fälle sind geeignete konzeptionelle Integrationen zu finden.

Die Fachhochschule Nordostniedersachsen

in Volgershall um 1984 :

Endlich in eigenen Räumen:

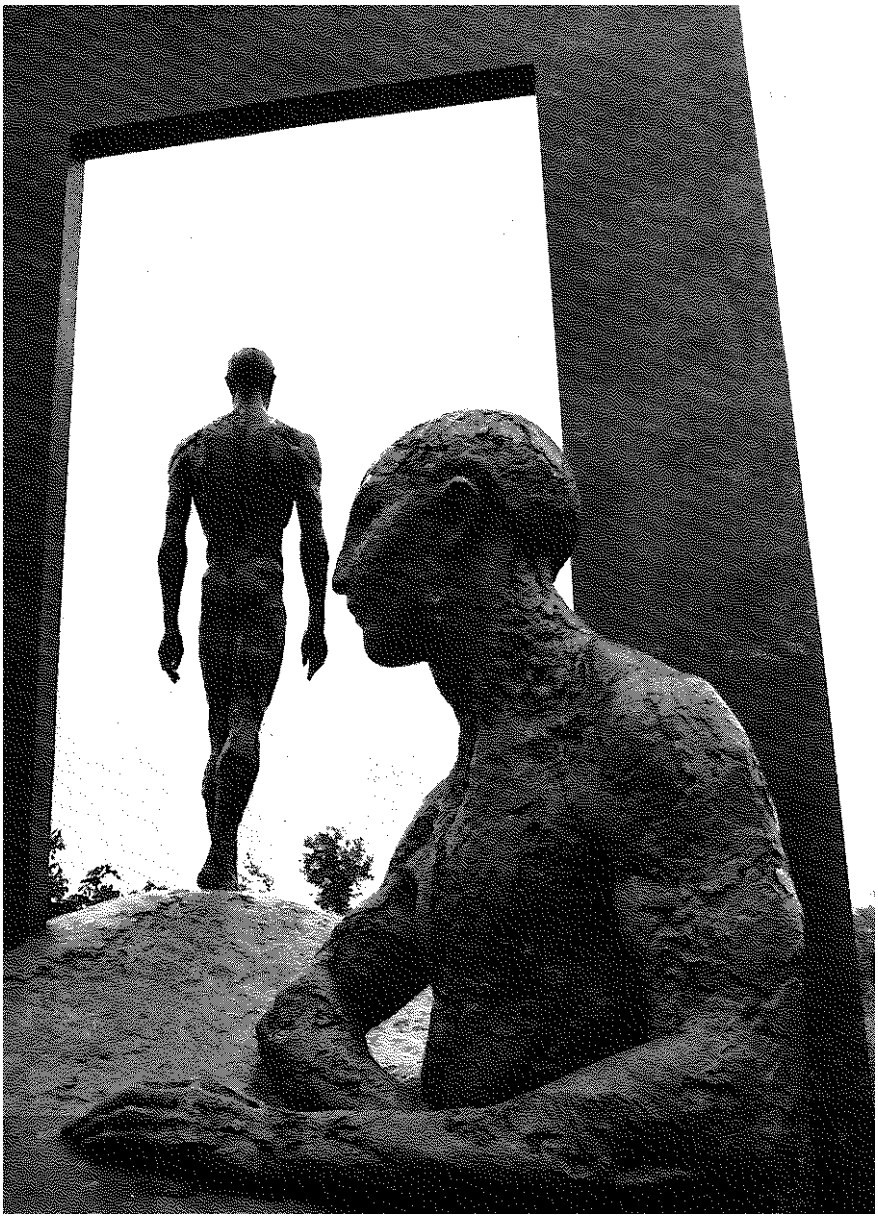
ein neues Gebäude für den neuen Dozenten.

Die Schließung der "Anstalt" ist tatsächlich im Gespräch,

denn die Leuphana Universität will sich auf dem

Campus Scharnhorststraße konzentrieren.

Nun verlässt der junge Mensch die Anstalt.



Aus Lünewirt Nr. 9, September 1987 (Mitteilungen des Vereins der Absolventen und Freunde des Fachbereichs Wirtschaft der Fachhochschule Nordostniedersachsen e.V., Herausgeber: Förderverein Fachhochschule Lüneburg)

Ein kleiner Streifzug durch verschiedene Ausgaben der

ViP! **V**ery **I**mportant **P**aper

das Studentenmagazin der Fachhochschule Nordostniedersachsen

.....

PROF-TRADINGCARDS

Prof. Dr. Karl Goede

Geb.: 22.9.1944

Motto:Ein guter (Software-)Fehler kommt immer wieder

Seit wann an der FH?

1984

Veranstaltungen

Systemnahe Programmierung, Datenbanken, DC, RABS

Sonstige FH-Aktivitäten

Fachbereichsrat, Praxissemesterausschuß

Werdegang

Physik-Studium an der TU Braunschweig; 2 fi J. im Rechenzentrum der TU; Assistententätigkeit am Lehrstuhl für formale Sprachen und Compilerbau; Promotion; MBP Dortmund; Softwarehaus Werum; Selbständige Tätigkeit als SW-Entwickler in der Baubranche

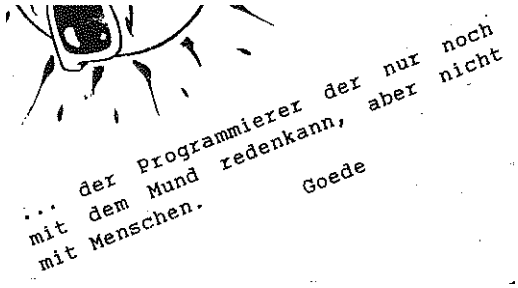
Lieblingsessen in der Mensa

Currywurst-Pommes, Erdbeeren mit Sahne

Goede

$\lim (\text{COBOL}) = \{\text{Pascal, C}\}$

t ->2100 Jahre



Spruch aus dem Hörsaal

Ich mache das jetzt mal so wie der Betrunkene, der seinen Schluessel unter der Laterne sucht, obwohl er ihn woanders verloren hat -
na ja, sie kennen das ja.
Goede [SPG2]

Das liegt meistens daran, dass die Betriebswirte geistig gar nicht in der Lage sind, das nachzuvollziehen.
(Goede, VIP! 18, S.28)

How To Handle Profs

Der kleine Führer für persönliche Anliegen

Abkürzung: ***FF*** **Feilsch-Faktor: saloppe Bezeichnung für die Erfolgsaussicht auf Notenanhebung bei nachfolgender Klausurbesprechung**

Goede

Raum 121; Di 2. Block

Sehr umgänglich, sehr offen und hilfsbereit. Dementsprechend jederzeit und überall ansprechbar; wenn man ihn noch nicht kennt, sollte man sich zunächst aber an die Sprechstunde halten. Auch nach Vorlesungen gut belastbar. Ist der Anerkennung von Auslandsleistungen aufgeschlossen.

FF: nicht unmöglich, wenn gut argumentiert wird.

Prof. Dr. Goede am 30.10.02
in Systemprogrammierung:

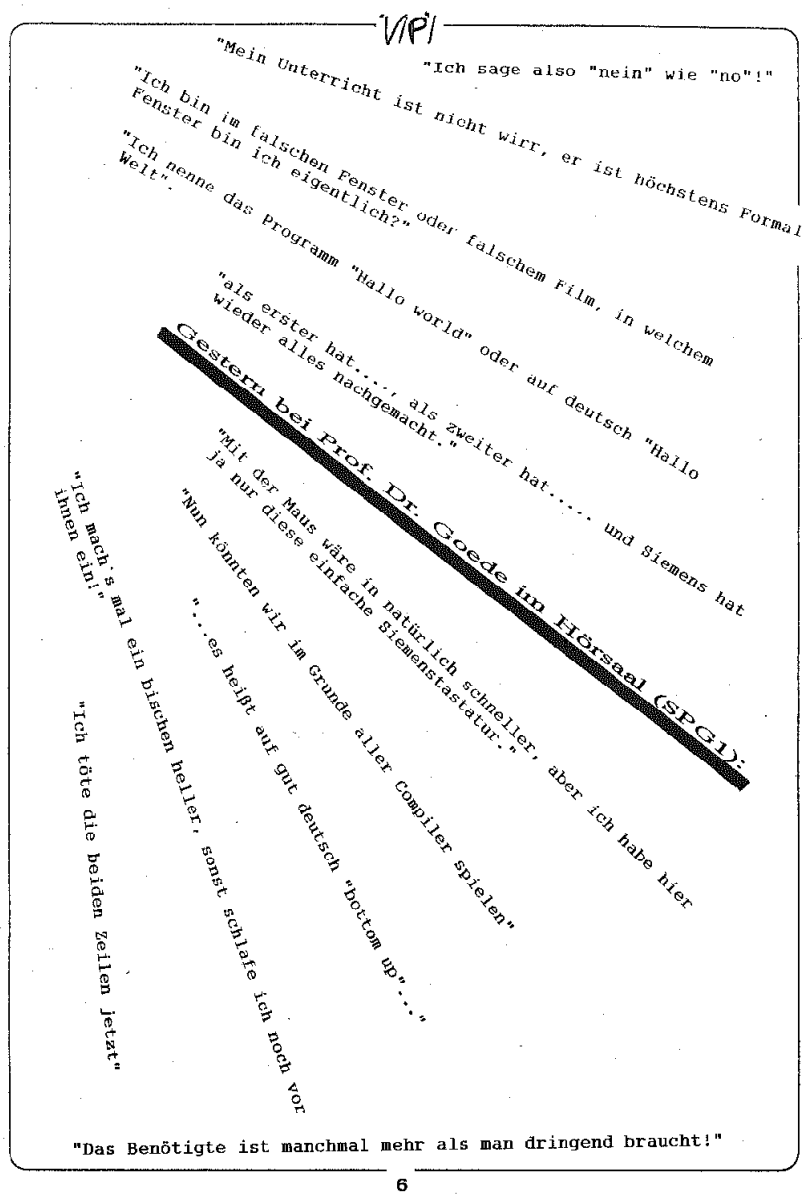
"Microsoft's Strategiewechsel zu .NET ist das Gleiche, als ob man in voller Fahrt einen Autoreifen wechseln würde!"

Erfahrung gem

ein paar

ich mich gut verste-

Kumar (Indien):



Der Neue...

...Dekan an unserem Fachbereich ist

Prof.Dr.Karl Goede,

mit dem exclusiv die VIP! sprach

VIP! Interview

VIP!: Welche Vorlesungen halten Sie?

Der generelle Rahmen besteht aus den Vorlesungen SPG (Systemprogrammierung), DB/DC (Datenbank-Systeme/Datenkommunikation) und DVS (Datenverarbeitungssysteme). Ansonsten halte ich turnusgemäß noch weitere Veranstaltungen im Bereich Rechnungswesen und Controlling zusammen mit Herrn Weinrich.

VIP!: Ist Ihrerseits eine Veranstaltung zur Programmierung unter WINDOWS geplant?

Sobald ich mal wieder freie Kapazität habe, biete ich solch eine Veranstaltung auf jeden Fall an.

VIP!: Wie wird man Dekan?

Es gibt keinen festen Algorithmus. Ansosnsten gilt: Jeder der hier Dekan wird, wird dies mit gemüschten Gefühlen, denn dessen Aufgaben führen zur Einschränkung der normalen Lehrtätigkeit. Aber durchschnittlich wird jeder zweite Professor im Laufe seiner Tätigkeit einmal Dekan.

VIP!: Von welchem Gremium wird der Dekan gewählt?

Der Fachbereichsrat wählt den Dekan.

VIP!: Kann ein Professor die Wahl ablehnen?

Theoretisch ist dies unter Umständen möglich. Aber es wird im Vorfeld bereits geklärt, wer die nächste Amtsperiode ohne Widerstand übernehmen wird.

VIP!: Welche Aufgaben hat ein Dekan?

Es gibt zwei Grundaufgaben. Die eine besteht in der Leitung des Fachbereichsrats. Die andere besteht darin, sicherzustellen, daß jeder Professor seine Lehre auch wahrnimmt, daß also die Veranstaltungspläne erfüllt werden. Er ist aber nicht der Vorgesetzte der Professoren. Der Vorgesetzte ist immer der Minister für Wissenschaft und Kunst; zu Zeit eine Ministerin. Er ist allerdings der Vorgesetzte der anderen Lehrenden und der Verwaltung hier im Haus .

VIP!: Inwieweit ist eine Einflußnahme auf die bildungspolitische Entwicklung gegeben, z.B. Erweiterung der FH?

Auf diese Entwicklungen besteht ein indirekter Einfluß. Man ist einmal Kraft seines Amtes Mitglied im Senat, allerdings nicht stimmberechtigt, sondern nur beratend. Es bestehen andererseits regelmäßige Kontakte zwischen den Dekanen aller Fachbereiche und dem Rektor. Dort werden schon Grundlagen für Entscheidungen zukünftiger Entwicklungen gelegt. Das betrifft natürlich auch die Ausstattung der Fachhochschulen.

VIP!: Es besteht kein professionelles "FH-Management"?

Nein! Normalerweise braucht man als Dekan etwa ein 3/4 Jahr, um sich einzuarbeiten und zu wissen, über welche Kanäle Entscheidungen gefällt werden. Meine vorherige Tätigkeit als EDV-Beauftrager unseres FB kommt mir jetzt zugute, da ich den Ablauf im Entsche idungsapparat bereits kenne.

VIP!: Wie lange dauert die Amtszeit eines Dekans?

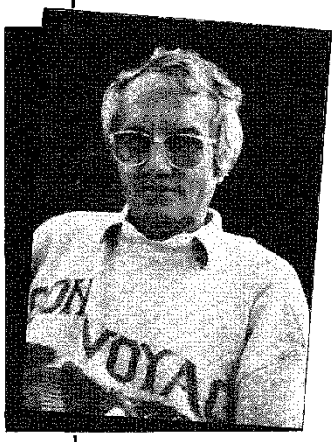
Die Amtszeit beträgt zwei Jahre.

VIP!: Haben Sie sich spezielle Ziele in Ihrer Amtszeit gesetzt?

Ja! Zusammen mit anderen Kollegen möchte ich die immer länger werdenen Studienzeiten an der FH verkürzen. Wobei noch nicht ganz klar ist, wie lang die Studienzeiten tatsächlich sind - ob wir nicht Fehler in der Statistik machen, z. B. sogenannte Karteileichen mitzählen oder andere Sünden begehen. Das gilt es, nochmals nachzuprüfen?

VIP: Wo, liegt Ihrer Meinung nach, die optimale Studienzeit?

Prinzipiell sollte es möglich sein, ein Studium in 6 Semestern zu absolvieren. Es gibt einige Studenten, die das schaffen. Faktisch sind wir aber sehr zufrieden, wenn das Studium in 7 Semestern durchlaufen wird. Werden es mehr als 8 (oder 9 Semester inkl. Auslandsaufenthalt), ist es zu lang. Gerade diese Semesterzahl wird sehr häufig überschritten.



VIP: Wie wollen Sie diesem Problem entgegensteuern?

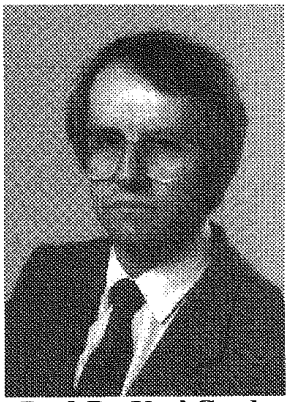
Es gibt einmal die "genial einfache" Idee, auf Lehrinhalte, von denen wir zuviel haben, zu verzichten. Faktisch versucht jeder auf dem Stand der Technik zu bleiben, und baut darauf seine Vorlesungen auf, vergißt aber, alte Lehrinhalte wegzuwerfen.Ein anderer Weg besteht darin - obwohl man da Vorsicht vor Einengungen walten lassen muß, den Studenten noch mehr Informationen über einen günstigen Studienweg an die

Fortsetzung >>

Rückblick auf zwei Jahre Dekanat

Prof. Dr. Karl Goede

An sich ist ein Rückblick auf die Amtsperiode eine hübsche Gelegenheit, den Unterhaltungswert einer Zeitschrift zu vermindern und die Verkaufszahlen der Folgeausgabe zu senken. Es war nicht leicht für mich, diesem möglicherweise so nicht gemeinten Wunsch der Herausgeber dieser Zeitschrift zu widerstehen und all das ordentlich zusammenzufassen, was ich in verschiedenen *Lünewirt*-Ausgaben unter dem Generaltitel „Neues aus dem Dekanat" geschrieben habe.



Prof. Dr. Karl Goede

Stattdessen „blicke ich zurück" in Form jener Artikel, die ich **nicht** geschrieben habe, die aber in erstklassigem AMI Pro-Format als Arbeitstitel mit Rohtext noch die Cluster meiner PC-Magnetplatten verstopfen. Deren Überschriften bieten einen Rückblick anderer Art:

Betriebsausflüge
... eine originelle Aufgabe für den Fachbereich, den die Ausrichtung trifft.

Rußland -> Tartu
... vom Land, in dem der Aufenthalt unserer Kontaktprofessoren gefährlicher wird, zum Land, in dem der Zoll gekaufte Musikinstrumente einzieht und in den Verkaufszyklus zurückführt.

Laptops als Platinfüller
... Auslöser waren permanente Anträge an den Fachbereichsrat, statt der Beschaffung ordentlicher PCs mit
- ordentlicher Tastatur
- tatsächlich einsetzbarer Maus statt eines „roten Fummels" oder eines Trackballs für Artisten
- mit ordentlichem Bildschirm vernünftiger Größe von optimal 17" (mehr bringt den Schreibtisch zum Wanken)
- tatsächlich anschließbarer preisgünstiger Magnetplatten, Grafikkarten etc.

der Beschaffung ICE-/flugzeugfähiger Laptops zuzustimmen.

CASE ist Käse?
... nach Durchsicht vieler Diplomarbeiten zum Thema Computer Aided Software Engineering.

Controller-Materialschlachten am Computer.
... wie ich das anstrengende Denken an den Computer delegiere.

Studentenaustausch - auch die Etikette ist wichtig.
... nach beeindruckendem Auftreten einiger unserer Austauschstudenten in Laredo („Vier gegen Texas").

Studienzeiten
... ein leidiges Dauerthema für uns alle.

Groupware
... nach Anfangserfolgen von LOTUS Notes im Fachbereich Wirtschaft.

RZ-Spaltung
... nachdem jene Institution, die heutzutage von hohem Spottwert für die Fernsehshow „Berg & Talk" ist, nicht mehr an ein Fachhochschulrechenzentrum angeschlossen bleiben möchte.

Ist der *Lünewirt* out - kommt die *Lünewirt*-Mailbox?
.... nachdem Studenten unseres Fachbereichs erfolgreich eine Mailbox betreiben.

Vom Unsinn der Umfragen zu Lehre und Forschung
... nach einigem Ärger über die Überschwemmung des Dekanats mit mäßig durchdachten Fragebögen unterschiedlichster Herkunft (Rektorenkonferenz, DASA, Rektorat, IHK, Hochschulführer-Verlage, diverse Zeitschriftenverlage, ...) unter Unwillen der Frager, existierende Dokumente zu lesen.

Ein Datenmodell für das Controlling in Lehre und Forschung
... nach dem Rückfall in konstruktives Denken.

Wahrscheinlich war es nicht ganz dumm, auf die meisten jener Titel zu verzichten und sich auf sicheres Terrain wie:

Von AIX, WNT und anderen Buchstabentripeln (SAP, IBM, MVS, DEC, VMS) - Tripelitis
... im **Lünewirt 15** unter dem Titel „Multitasking, Marketing und Mystik (MMM)" erschienen, zu beschränken - frei nach meinem Namensvetter (in sächsischer Sprechweise) Goethe: In der Beschränkung (nicht *Beschränktheit*) zeigt sich erst der Meister.

Ein Beitrag aus Lünewirt Nr. 16, September 1994:

Neurofuzzy - neues Schimpfwort oder neues Problemlösungsverfahren

Prof. Dr. Karl Goede

Die Begriffe *Neuronale Netze und Fuzzy Logik besetzen zunehmend die Überschriftenzeilen von Computerzeitschriften. Liest man nicht allzu genau, drängt sich folgender Eindruck auf: Ohne Arbeitskreise zu diesen Themen sind Informatiker- und Biologenkongresse nicht mehr denkbar. Ohne Fuzzy Logik läuft keine Waschmaschine, funktioniert kein Fotoapparat, arbeitet keine moderne Fabrik, fließt kein ordentlicher Straßenverkehr.*

Kommt nun *Neuronalen Netzen und Fuzzy Logik wirklich die in einschlägigen Medien unterlegte Bedeutung zu oder haben Skeptiker recht, die hier eine marketinggerechte Verschwörung von Halbleiterindustrie und Forschungsinstitutionen sehen?*

Für die Halbleiterindustrie könnte ein Boom *Neuronaler Netze (nervensystemartig verschaltete Netze aus Mikroprozessoren als Abstraktion einfacher Gehirne) einen warmen Nachfrageregen auslösen. Nachdem kaum jemand so recht weiß, mit welcher Software ein anderer erhoffter Nachfrageauslöser, die aus großen Anzahlen parallel organisierter Prozessoren bestehenden Superrechner, betrieben werden sollen, böten die "Neuronennetze" eine brauchbare Alternative.*
Tatsächlich muß man die Verschwörungstherie hier zurückweisen. Während Supercomputing-Gräber für alte Motorola-Prozessoren (68K-Linie: u. a. SUPRENUM-Rechner) oder neue Motorola/IBM/Apple-Prozessoren (POWER-Chip: u. a. MANNA-Rechner) noch für die schöne Anordnung der "begrabenen" Prozessoren gelobt werden, sind für in Hardware "gegossene" neuronale Netze bereits die Anwendungen zusammen mit der sie unterstützenden Software klar formulierbar. Für den parallelen Einsatz vieler Prozessoren besteht von vornherein ein einfaches Konzept.

Für Forscher bieten *Neuronale Netze aus der Sicht von Skeptikern eine hübsche Gelegenheit, sich einer geregelten Arbeit zu entziehen und sich einer unproduktiven Kunstwelt mit esoterischem Touch hinzugeben. Es ist schon lange klar, daß jedes spezielle Problem weitaus besser durch ein speziell darauf zugeschnittenes Programm als durch ein für das Problem "trainiertes" Netzwerk zu lösen ist. Neuronale Netze bieten aus Sicht eines Informatikers stets nur die zweitbeste und aus Sicht eines Mathematikers nur die drittbeste Methode, ein wohldefiniertes Problem zu lösen. Zudem sieht einiges nach "altem Wein in neuen Schläuchen" aus, wenn beispielsweise Gauß-Approximation und Gradientenverfahren in Form eines wohlklingenden Trainingsverfahrens namens*

"Backpropagation" fröhliche Urständ feiern. Noch ältere Rechte (vielleicht mit noch zu entdeckendem Copyright-Vermerk im Genom) der alten Götter (1. Mose 26: Lasset uns den Menschen machen) seien hier nicht weiter erörtert.

Der Verdacht könnte naheliegen, daß hier wieder einmal (Lindenmeyer-Systeme, Logik mit Petri-Netzen etc. lassen grüßen) eine geschickte Methode gefunden wurde, Reisen zu Forschungskongressen an den schönsten Plätzen dieser Erde zu begründen.

Tatsächlich trägt der Verdacht aber nicht, wie viele schnell erreichbare reale Anwendungen zeigen. Was aus bestimmten Blickwinkeln als zweit- oder drittklassig ist, erweist sich aus dem Blickwinkel schneller sicherer Anwendungsentwicklung als erstklassig.

Fuzzy Logik scheint neben dem Antrieb von Waschmaschinen das Mittel zu sein, entscheidungsschwachen Mitbürgern das Leben zu erleichtern. Sie müssen nicht mehr klar sagen, was sich hinter ihren Worten verbirgt. Stattdessen dürfen sie sich schwammig ausdrücken.

Tatsächlich ist aber Unschärfe in vielen Situationen ein Gewinn. Wenn beispielsweise gefragt ist, welche Menschen jung, welche mittelalt und welche alt sind, wird nicht mehr die schwachsinnige knappe Aussage

Jung sind Menschen im Intervall [0, 24], mittelalt im Intervall [25, 54] und alt sind alte Menschen ab 55 (und damit basta)

(o. ä.) verlangt. Vielmehr lassen sich überlappende Altersintervalle festlegen - mit der in klassischer Logik unmöglichen Folgeerscheinung, daß das Alter 30 etwa zugleich jung und mittelalt bedeuten kann. Im allgemeinen läßt die Fuzzy Logik zu, daß mit einer Aussage zugleich ihr Gegenteil wahr sein darf. Dies bewegte den Erfinder Lotfi Zadeh dazu, Fuzzy-Mengen (fuzzy sets) mit geeigneten Mengenoperationen zu betrachten und erst später deren Isomorphie mit einer "fadenscheinigen" Logik zu zeigen, in welcher der Satz vom ausgeschlossenen Dritten nicht gilt und Widerspruchsfreiheit nicht zu finden ist.

Was verbindet nun die Begriffe neuronal und fuzzy? Mittelalten ehemaligen Filmbesuchern fällt es vermutlich leicht, jeden hinreichend in Neuronale Netze verstrickten Forscher in Anlehnung an einen "Filmhelden" des alten Schwarz-Weiß-Kinos als Neurofuzzy zu bezeichnen.

Dies aber ist nicht die Verbindung, die eine ganze Klasse realer Anwendungen neuronaler Netze zu bestimmen beginnt. Ausgangspunkt für diese Anwendungen ist vielmehr die Erkenntnis, daß Fuzzy Systeme stets als Netze darstellbar sind, deren Neuronen Fuzzy-Operationen realisieren. Eine bestätigende Tradition für solch eine Art von Neurofuzzy-Systemen bietet das menschliche Auge beim Sehen von Farben.

Die Arbeitsteilung zwischen Fuzzy- und Netz-Sicht ist folgende: Die Fuzzy-Sicht definiert die Benutzerschnittstelle. Der Benutzer wendet sich mit ihm vertrauten - ggf. auch unscharfen - Begriffen an das System und erhält ihm geläufige - ggf. wiederum unscharfe - Aussagen zurück. Das Training des Systems (Gewichtsfaktoren an den "Synapsen" einstellen) und numerische Zwischenrechnungen werden vom zugrundeliegenden neuralen Netz bewerkstelligt.

Sehr gut brauchbar ist die Neurofuzzy-Kombination bei der Lösung schwer modellierbarer technischer Probleme mit stückweise nur qualitativ faßbaren Eigenschaften . Beispiele sind
die Steuerung/Regelung fahrbarer Kräne, die pendelnde Lasten (und wann pendeln die nicht) möglichst schnell transportieren und dann exakt absetzen müssen.
Fehlervorhersagen bei Motoren. Die Firma Siemens berichtet von einer Verbesserung der Vorhersagegenauigkeit von konventionell ca. 30 % auf 80 bis 90 % in ihrem SAMMS (Siemens Advanced Motor Master System) bei großen Elektromotoren.
Qualitätskontrolle in der Fertigung. Die japanische Firma Komatsu Corp. prüft so fortlaufend die Güte von Schweißnähten.
Im ersten Beispiel ist die Möglichkeit qualitativer Eingaben (z. B. "*langsam dem Ziel nähern" statt "3 m vor dem Ziel Geschwindigkeit auf 3,4 ms^{-1} absenken, mit 2 ms^{-2} weiter verzögern") bedeutsam, in den letzten beiden sind es verständliche Ausgaben (z. B. "Haarriß" statt eines Zahlenwerks, welches das Antwortverhalten auf einen prüfenden Utraschallimpuls darstellt).*

Sehr gut brauchbar sind Neurofuzzy-Systeme aber auch für Betriebswirte:
Statt sich bei schwer formalisierbaren Problemen weiterhin mit statistischen Lösungsverfahren zu plagen, können sie jetzt ein neuronales Netz nehmen. Der Bestimmung von Regressionsgeraden (Betriebswirte finden glücklicherweise immer eine Möglichkeit, nichtlineares wegzudiskutieren) entspricht das "Lernen" des Netzes. Bei mehrschichtigen Netzen kann man dann - genauso glücklich - ohne größere Inanspruchnahme der eigenen biologisch-echt-neuronal besetzten Gerhirnwindungen auch nichtlineare Fälle in den Griff bekommen und verstehen lernen.
Die Fuzzy-Komponente gestattet wie in den "guten alten Zeiten" unter Verwendung unscharfer Begriffe wieder Regeln der Art "Wenn der Gewinn *stimmt, das Management gut und die Angestellten motiviert sind, hat eine Firma gute Aussichten für die nähere Zukunft" zu formulieren.*

In der Reihe FINAL sind bisher erschienen:

1. Jahrgang 1991:

1. Hinrich E. G. Bonin; Softwaretechnik, Heft 1, 1991 (ersetzt durch Heft 2, 1992).
2. Hinrich E. G. Bonin (Herausgeber); Konturen der Verwaltungsinformatik, Heft 2, 1991 (überarbeitet und erschienen im Wissenschaftsverlag, Bibliographisches Institut & F. A. Brockhaus AG, Mannheim 1992, ISBN 3-411-15671-6).

2. Jahrgang 1992:

1. Hinrich E. G. Bonin; Produktionshilfen zur Softwaretechnik --- Computer-Aided Software Engineering --- CASE, Materialien zum Seminar 1992, Heft 1, 1992.
2. Hinrich E. G. Bonin; Arbeitstechniken für die Softwareentwicklung, Heft 2, 1992 (3. überarbeitete Auflage Februar 1994), PDF-Format.
3. Hinrich E. G. Bonin; Object-Orientedness --- a New Boxologie, Heft 3, 1992.
4. Hinrich E. G. Bonin; Objekt-orientierte Analyse, Entwurf und Programmierung, Materialien zum Seminar 1992, Heft 4, 1992.
5. Hinrich E. G. Bonin; Kooperative Produktion von Dokumenten, Materialien zum Seminar 1992, Heft 5, 1992.

3. Jahrgang 1993:

1. Hinrich E. G. Bonin; Systems Engineering in Public Administration, Proceedings IFIP TC8/ WG8.5: Governmental and Municipal Information Systems, March 3--5, 1993, Lüneburg, Heft 1, 1993 (überarbeitet und erschienen bei North-Holland, IFIP Transactions A-36, ISSN 0926-5473).
2. Antje Binder, Ralf Linhart, Jürgen Schultz, Frank Sperschneider, Thomas True, Bernd Willenbockel; COTEXT --- ein Prototyp für die kooperative Produktion von Dokumenten, 19. März 1993, Heft 2, 1993.
3. Gareth Harries; An Introduction to Artificial Intelligence, April 1993, Heft 3, 1993.
4. Jens Benecke, Jürgen Grothmann, Mark Hilmer, Manfred Hölzen, Heiko Köster, Peter Mattfeld, Andre Peters, Harald Weiss; ConFusion --- Das Produkt des AWÖ-Projektes 1992/93, 1. August 1993, Heft 4, 1993.
5. Hinrich E. G. Bonin; The Joy of Computer Science --- Skript zur Vorlesung EDV ---, September 1993, Heft 5, 1993 (4. ergänzte Auflage März 1995).
6. Hans-Joachim Blanke; UNIX to UNIX Copy --- Interactive application for installation and configuration of UUCP ---, Oktober 1993, Heft 6, 1993.

4. Jahrgang 1994:

1. Andre Peters, Harald Weiss; COMO 1.0 --- Programmierumgebung für die Sprache COBOL --- Benutzerhandbuch, Februar 1994, Heft 1, 1994.
2. Manfred Hölzen; UNIX-Mail --- Schnelleinstieg und Handbuch ---, März 1994, Heft 2, 1994.
3. Norbert Kröger, Roland Seen; EBrain --- Documentation of the 1994 AWÖ-Project Prototype ---, June 11, 1994, Heft 3, 1994.
4. Dirk Mayer, Rainer Saalfeld; ADLATUS --- Documentation of the 1994 AWÖ-Project Prototype -- -, July 26, 1994, Heft 4, 1994.
5. Ulrich Hoffmann; Datenverarbeitungssystem 1, September 1994, Heft 5, 1994. (2. überarbeitete Auflage Dezember 1994).
6. Karl Goede; EDV-gestützte Kommunikation und Hochschulorganisation, Oktober 1994, Heft 6 (Teil 1), 1994.
7. Ulrich Hoffmann; Zur Situation der Informatik, Oktober 1994, Heft 6 (Teil 2), 1994.

5. Jahrgang 1995:

1. Horst Meyer-Wachsmuth; Systemprogrammierung 1, Januar 1995, Heft 1, 1995.
2. Ulrich Hoffmann; Datenverarbeitungssystem 2, Februar 1995, Heft 2, 1995.
3. Michael Guder / Kersten Kalischefski / Jörg Meier / Ralf Stöver / Cheikh Zeine; OFFICE-LINK --- Das Produkt des AWÖ-Projektes 1994/95, März 1995, Heft 3, 1995.
4. Dieter Riebesehl; Lineare Optimierung und Operations Research, März 1995, Heft 4, 1995.
5. Jürgen Mattern / Mark Hilmer; Sicherheitsrahmen einer UTM-Anwendung, April 1995, Heft 5, 1995.

6. Hinrich E. G. Bonin; Publizieren im World-Wide Web --- HyperText Markup Language und die Kunst der Programmierung ---, Mai 1995, Heft 6, 1995.
7. Dieter Riebesehl; Einführung in Grundlagen der theoretischen Informatik, Juli 1995, Heft 7, 1995.
8. Jürgen Jacobs; Anwendungsprogrammierung mit Embedded-SQL, August 1995, Heft 8, 1995.
9. Ulrich Hoffmann; Systemnahe Programmierung, September 1995, Heft 9, 1995 (ersetzt durch Heft 1, 1999).
10. Klaus Lindner; Neuere statistische Ergebnisse, Dezember 1995, Heft 10, 1995.

6. Jahrgang 1996:

1. Jürgen Jacobs / Dieter Riebesehl; Computergestütztes Repetitorium der Elementarmathematik, Februar 1996, Heft 1, 1996.
2. Hinrich E. G. Bonin; "Schlanker Staat" & Informatik, März 1996, Heft 2, 1996.
3. Jürgen Jacobs; Datenmodellierung mit dem Entity-Relationship-Ansatz, Mai 1996, Heft 3, 1996.
4. Ulrich Hoffmann; Systemnahe Programmierung, (2. überarbeitete Auflage von Heft 9, 1995), September 1996, Heft 4, 1996 (ersetzt durch Heft 1, 1999).
5. Dieter Riebesehl; Prolog und relationale Datenbanken als Grundlagen zur Implementierung einer NF2-Datenbank (Sommer 1995), November 1996, Heft 5, 1996.

7. Jahrgang 1997:

1. Jan Binge, Hinrich E. G. Bonin, Volker Neumann, Ingo Stadtsholte, Jürgen Utz; Intranet-/Internet-Technologie für die Öffentliche Verwaltung --- Das AWÖ-Projekt im WS96/97 --- (Anwendungen in der Öffentlichen Verwaltung), Februar 1997, Heft 1, 1997.
2. Hinrich E. G. Bonin; Auswirkungen des Java-Konzeptes für Verwaltungen, FTVI'97, Oktober 1997, Heft 2, 1997.

8. Jahrgang 1998:

1. Hinrich E. G. Bonin; Der Java-Coach, Heft 1, Oktober 1998, (CD-ROM, PDF-Format; aktuelle Fassung).
2. Hinrich E. G. Bonin (Hrsg.); Anwendungsentwicklung WS 1997/98 --- Programmierbeispiele in COBOL & Java mit Oracle, Dokumentation in HTML und tcl/tk, September 1998, Heft 2, 1998 (CD-ROM).
3. Hinrich E. G. Bonin (Hrsg); Anwendungsentwicklung SS 1998 --- Innovator, SNiFF+, Java, Tools, Oktober 1998, Heft 3, 1998 (CD-ROM).
4. Hinrich E. G. Bonin (Hrsg); Anwendungsentwicklung WS 1998 --- Innovator, SNiFF+, Java, Mail und andere Tools, November 1998, Heft 4, 1998 (CD-ROM).
5. Hinrich E. G. Bonin; Persistente Objekte --- Der Elchtest für ein Java-Programm, Dezember 1998, Heft 5, 1998 (CD-ROM).

9. Jahrgang 1999:

1. Ulrich Hoffmann; Systemnahe Programmierung (3. überarbeitete Auflage von Heft 9, 1995), Juli 1999, Heft 1, 1999 (CD-ROM und Papierform), Postscript-Format, zip-Postscript-Format, PDF-Format und zip-PDF-Format.

10. Jahrgang 2000:

1. Hinrich E. G. Bonin; Citizen Relationship Management, September 2000, Heft 1, 2000 (CD-ROM und Papierform), PDF-Format.
2. Hinrich E. G. Bonin; WI>DATA --- Eine Einführung in die Wirtschaftsinformatik auf der Basis der Web_Technologie, September 2000, Heft 2, 2000 (CD-ROM und Papierform), PDF-Format.
3. Ulrich Hoffmann; Angewandte Komplexitätstheorie, November 2000, Heft 3, 2000 (CD-ROM und Papierform), PDF-Format.
4. Hinrich E. G. Bonin; Der kleine XMLer, Dezember 2000, Heft 4, 2000 (CD-ROM und Papierform), PDF-Format, aktuelle Fassung.

11. Jahrgang 2001:

1. Hinrich E. G. Bonin (Hrsg.): 4. SAP-Anwenderforum der FHNON, März 2001, (CD-ROM und Papierform), Downloads & Videos.
2. J. Jacobs / G. Weinrich; Bonitätsklassifikation kleiner Unternehmen mit multivariater linear Diskriminanzanalyse und Neuronalen Netzen; Mai 2001, Heft 2, 2001, (CD-ROM und Papierform), PDF-Format und MS Word DOC-Format

3. K. Lindner; Simultanttestprozedur für globale Nullhypothesen bei beliebiger Abhängigkeitsstruktur der Einzeltests, September 2001, Heft 3, 2001 (CD-ROM und Papierform).

12. Jahrgang 2002:

1. Hinrich E. G. Bonin: Aspect-Oriented Software Development. März 2002, Heft 1, 2002 (CD-ROM und Papierform), PDF-Format.
2. Hinrich E. G. Bonin: WAP & WML --- Das Projekt Jagdzeit ---. April 2002, Heft 2, 2002 (CD-ROM und Papierform), PDF-Format.
3. Ulrich Hoffmann: Ausgewählte Kapitel der Theoretischen Informatik (CD-ROM und Papierform), PDF-Format.
4. Jürgen Jacobs / Dieter Riebesehl; Computergestütztes Repetitorium der Elementarmathematik, September 2002, Heft 4, 2002 (CD-ROM und Papierform), PDF-Format.
5. Verschiedene Referenten; 3. Praxisforum "Systemintegration", 18.10.2002, Oktober 2002, Heft 5, 2002 (CD-ROM und Papierform), Praxisforum.html (Web-Site).

13. Jahrgang 2003:

1. Ulrich Hoffmann; Ausgewählte Kapitel der Theoretischen Informatik; Heft 1, 2003, (CD-ROM und Papierform) PDF-Format.
2. Dieter Riebesehl; Mathematik 1, Heft 2, 2003, (CD-ROM und Papierform) PDF-Format.
3. Ulrich Hoffmann; Mathematik 1, Heft 3, 2003, (CD-ROM und Papierform) PDF-Format und Übungen.
4. Verschiedene Autoren; Zukunft von Verwaltung und Informatik, Festschrift für Heinrich Reinermann, Heft 4, 2003, (CD-ROM und Papierform) PDF-Format.

14. Jahrgang 2004:

1. Jürgen Jacobs; Multilayer Neural Networks; Heft 1, 2004, (CD-ROM und Papierform) PDF-Format.

15. Jahrgang 2005:

1. Ulrich Hoffmann; Mathematik für Wirtschaftsinformatiker; Heft 1, 2005, (CD-ROM und Papierform) PDF-Format.
2. Ulrich Hoffmann; Übungen & Lösungen zur Mathematik für Wirtschaftsinformatiker; Heft 1, 2005, (CD-ROM und Papierform) PDF-Format.
3. Ulrich Hoffmann; Datenstrukturen & Algorithmen; Heft 2, 2005, (CD-ROM und Papierform) PDF-Format.

16. Jahrgang 2006:

1. Hinrich E. G. Bonin; Systemanalyse für Softwaresysteme; Heft 1, August 2006, (CD-ROM und Papierform) PDF-Format.
2. Hinrich E. G. Bonin; Faszination Programmierung; Heft 2, August 2006, (CD-ROM und Papierform) PDF-Format.
3. Dieter Riebesehl; Strukturanalogien in Datenmodellen, Heft 3, Dezember 2006, (CD-ROM und Papierform) PDF-Format.

17. Jahrgang 2007:

1. Ulrich Hoffmann; Ausgewählte Kapitel der Theoretischen Informatik; Heft 1, August 2007, (CD-ROM und Papierform) PDF-Format.
2. Ulrich Hoffmann; Mathematik für Wirtschaftsinformatiker und Informatiker; Heft 2, August 2007, (CD-ROM und Papierform) PDF-Format.
3. Hinrich E. G. Bonin; Der Java-Coach, Heft 3, September 2007, (CD-ROM und Papierform) PDF-Format.
4. Jürgen Jacobs; Dichteprognose autoregressiver Zeitreihen, Heft 4, September 2007, (CD-ROM und Papierform) PDF-Format.

18. Jahrgang 2008:

1. Hinrich E. G. Bonin; Festschrift für Prof. Dr. Meyer-Wachsmuth; Heft 1, Juli 2008, (CD-ROM und Papierform) PDF-Format.
2. Ulrich Hoffmann; Ausgewählte Kapitel der Theoretischen Informatik; Heft 2, Dezember 2008, (CD-ROM und Papierform) PDF-Format.

Herausgeber:
Prof. Dr. Dipl.-Ing. Dipl.-Wirtsch.-Ing. Hinrich E. G. Bonin
Leuphana Universität Lüneburg, Volgershall 1, D-21339 Lüneburg, Germany
email: bonin@uni.leuphana.de

Lektorat für diese Ausgabe:
Maja Irmhild Schütte-Hoof MA
Lektorin für Deutsch im Fremdsprachenzentrum der Leuphana Universität Lüneburg

Verlag:
Eigenverlag (Fotographische Vervielfältigung), Leuphana Universität Lüneburg (vormals Fachhochschule Nordostniedersachsen)

Erscheinungsweise:
ca. 4 Hefte pro Jahr.
Für unverlangt eingesendete Manuskripte wird nicht gehaftet. Sie sind aber willkommen.

LEUPHANA
UNIVERSITÄT LÜNEBURG

Prof. Dr. Karl Goede