e-Journal of Practical Business Research

Analyse von in der Pharmaforschung eingesetzten Anwendungen zur Darstellung und Auswertung biologischer Testergebnisse

Marcel Weier

Erschienen im e-Journal of Practical Business Research
unter: http://www.e-journal-of-pbr.de

Diese Arbeit befasst sich mit der Konsolidierung von Altanwendungen auf Anforderungsebene. Der Fokus liegt hierbei auf Anwendungen in einem besonders dynamischen Umfeld wie der Forschung. Dazu werden im Folgenden die Methoden des Reengineering in Form von Testfällen, Dokumentationsstudium und Bedienung der Anwendungen betrachtet und angewendet. Ebenfalls wird untersucht, wie sich mit dieser Vorgehensweise funktionale und nichtfunktionale Eigenschaften ermitteln lassen. Aus diesen Eigenschaften sollen Rückschlüsse auf die ursprünglich gestellten Anforderungen an die jeweilige Anforderung gezogen werden. Die so ermittelten Anforderungen sind dann zu vergleichen und zusammenzuführen. Die zu ermittelnde konzeptionelle Grundlage dient der Übertragung der Funktionen aus Legacy-Anwendungen in den Entwurf einer fortschrittlicheren Anwendung. Auf diese Weise sollen vorhandenes Wissen und bestehende Konzepte erhalten bleiben.

Zitation: Weier, Marcel (2009): Analyse von in der Pharmaforschung eingesetzten Anwendungen zur Darstellung und Auswertung biologischer Testergebnisse In: e-Journal of Practical Business Research, Ausgabe 8 (03/2009), DOI: 10.3206/0000000022

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Einleitung

Unternehmen stehen heute durch die rasant voranschreitenden Entwicklungen im Zuge der Globalisierung vor besonderen Herausforderungen. Eine hohe Dynamik der Märkte führt häufig zu organisatorischen Änderungen in Form von Fusionen und Übernahmen, wodurch eine Anpassung der internen Strukturen in technischer Hinsicht erforderlich wird. Andererseits zwingt der Konkurrenzdruck zu Innovation, da die Entwicklung neuer Technologien immer mehr an Geschwindigkeit zunimmt. Vor allem in innovationsgeprägten Bereichen wie der Forschung werden die Entwicklung neuer Verfahren und Technologien besonders stark vorangetrieben. Aufgrund der hohen Dynamik ergeben sich beispielsweise oft Änderungen an Datenvolumen, Berechnungsmustern und Arbeitsabläufen. Da die Einsatzgebiete in der Forschung häufig nicht mit Standardsoftware abgedeckt werden können, sind dann Anpassungen der alten Entwicklungen an neuere, fortschrittlichere Frameworks oder veränderte organisatorische Rahmenbedingungen und die damit verbundenen Kundenwünsche notwendig.

1.1 Legacy-Systeme

Die Anpassung an neue informationstechnologische Entwicklungen oder neue Erkenntnisse aus der Forschung können sich sehr umfangreich gestalten. Oftmals investieren Unternehmen viel Geld, Zeit und Arbeit in die Erstellung komplexer Anwendungssysteme, welche dann auch entsprechend lange genutzt werden sollen.[1] Sowohl Software, als auch Hardware und Prozesse wachsen aber mit der Zeit zu einem Legacy-System.[2] Beispielsweise sind die Technologien überholt worden oder zahlreiche Wartungen und Erweiterungen haben die Struktur unübersichtlich und schwer modifizierbar gemacht.
Laut Sommerville handelt es sich bei Legacy-Systemen um „soziotechnische, computerbasierte Systeme, die in der Vergangenheit unter Verwendung [von heute] veralteter Technologien entwickelt wurden. … Änderungen an einem Teil dieses Systems ziehen unvermeidlich Änderungen an anderen Komponenten nach sich. Legacy-Systeme sind oft geschäftskritische Systeme. Sie werden weiterhin verwendet, weil es zu gefährlich ist, sie zu ersetzen“.[3] Daraus lässt sich schließen, dass sich viele Unternehmen nicht dem Risiko einer Abschaffung stellen wollen, aus Angst, totale Ausfälle hinnehmen zu müssen.[4] Doch gibt

[1] Vgl. Sommerville, Ian (2007), S. 65.
[2] Vgl. ebenda, S. 67.
[3] Ebenda, S. 65.
[4] Masak, Dieter (2005), S. 2.

es auch einige Nachteile beim Fortführen solcher Alt-Systeme, zum Beispiel erhöhter Wartungsaufwand[5], eingeschränkte Weiterentwicklungsmöglichkeiten oder vergleichsweise schwache Performanz, teilweise bedingt durch eine hohe Komplexität.[6]
Legacy-Systeme sind also in der Praxis häufiger anzutreffen. Es liegt daher nahe, sich zu fragen, ob Unternehmen besonders bei geschäftskritischen Systemen ein Augenmerk auf Aktualität legen sollten. Auch der richtige Umgang mit Legacy-Anwendungen sowie deren Bearbeitung ohne das Eingehen unnötiger Risiken wie Ausfälle, Verluste und Einbußen sollte bedacht sein. Oftmals bieten Firmenfusionen oder Übernahmen geeignete Möglichkeiten, die Altanwendungen zu überholen, da ohnehin Umstrukturierungen im gesamten Unternehmen stattfinden.
In Frage kommende Maßnahmen für das Behandeln von Legacy-Anwendungen müssen sorgfältig geprüft, geplant und ausgeführt werden. Umfangreiche Systemanalysen spielen dabei eine wichtige Rolle.[7] Diese Analysen sollen Funktionalitäten und Anforderungen genau festhalten, um eine passende Alternative auszuwählen, beziehungsweise eine passgenaue und risikominimierte Übertragung der Funktionalität auf ein anderes System zu ermöglichen.

1.2 Bedeutung für Unternehmen der Pharmabranche

In der pharmazeutischen Industrie gibt es zahlreiche global agierende Unternehmen, die im internationalen Wettbewerb tätig sind. Die nötigen Wettbewerbsvorteile versucht man sich durch ständige Innovationen und Verbesserungen zu verschaffen. Häufig vorkommende Übernahmen rufen das Thema Legacy-Systeme vor dem Hintergrund der Integration und Zusammenführung zweier Unternehmenswelten zusätzlich auf den Plan. Eine besondere Herausforderung stellt der Bereich der Pharmaforschung dar, wo ständig neue Methoden und Erkenntnisse entstehen und einbezogen werden. Hier ist das Umfeld für Softwaretechnik sehr dynamisch und Anforderungen verändern sich stetig.
Um mit anfallenden Daten besonders effektiv umgehen zu können, ist der Einsatz von moderner Technik und leistungsfähiger Software zur Verarbeitung von Forschungsergebnissen unumgänglich. Ebenso wie in anderen Bereichen lässt jedoch die Aktualität und Leistungsfähigkeit benutzter Anwendungen immer stärker nach. Alte Programmiersprachen und –techniken verhindern ab einem bestimmten Zeitpunkt eine Weiterentwicklung.[8] Stän-

[5] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 5.
[6] Vgl. Seacord, Robert S. u.a. (2003), S. 1f.
[7] Vgl. Sommerville, Ian (2007), S. 60.
[8] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 301.

dige Änderungen lassen den Programmcode unübersichtlicher werden und erschweren zukünftige Wartungen und Änderungen potenziell zunehmend.[9] Dann ist es erforderlich, die Systeme zu analysieren und nach Alternativen zu suchen.
Auch mit dem Hintergrund einer Firmenübernahme kommt die Systemanalyse, beziehungsweise in diesem Fall die Anwendungsanalyse zur Sprache. Unterschiedliche Forschungsmethoden und –werkzeuge sowie Redundanzen erfordern unter Gesichtspunkten der Rationalisierung eine Bewertung und Auswahlentscheidungen. Es sollen Synergien genutzt und ähnliche Funktionen vereint werden.

1.3 Struktur der Arbeit

Diese Arbeit setzt sich mit der Analyse von zwei Forschungsanwendungen auseinander. Im Rahmen der vollständigen Integration eines übernommenen Pharmaunternehmens und der Umstellung auf eine neue Architektur sollen die zwei Anwendungen funktional auf einer neuen technoligischen Plattform zusammengeführt werden. Ziel ist es dabei, den Prozess einer Systemanalyse von der Entwicklung, über die Durchführung bis zu einer konkreten Handlungsempfehlung zu gestalten und darzustellen. Die Ergebnisse sowie die Handlungsempfehlung der durchgeführten Fallstudie dienen in Folgeprojekten als Grundlage für die Übertragung der Funktionalitäten auf die moderne Webservice-Technologie, welche jedoch aufgrund des Themenbezugs nicht im Fokus steht.
Zur Erreichung der Zielsetzung werden im zweiten Kapitel theoretische Grundlagen des Reengineering von Software sowie allgemeine Softwareanforderungen diskutiert. Dies soll eine solide Grundlage bilden, welche auch auf die konkreten Feinheiten der beispielhaften Fallstudie anwendbar ist und geeignete Lösungsstrategien für die Analyse aufzeigt.
Das dritte Kapitel zeigt eine exemplarische Anwendung der ausgewerteten Theorie.
Der Schlussteil führt die Erkenntnisse zusammen und thematisiert im Ausblick die Nutzung der Analyseergebnisse für den Einsatz von serviceorientierten Strukturen.

[9] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 301.

2. Grundlagen

Die in diesem Kapitel behandelte Theorie besteht aus den Themenschwerpunkten Reengineering und Anforderungen an Software. Diese sind Vorraussetzung für das Verständnis der Fallstudie, da sie Ansätze zur Lösung der Problemstellung bieten.

2.1 Reengineering

2.1.1 Überblick: Von Legacy zu SOA

Viele moderne Architekturen orientieren sich mittlerweile an der Bereitstellung von Service-Angeboten.[10] Ein wichtiger Vorteil ist dabei unter anderem die lose Kopplung aller Komponenten, sodass ein Austausch ohne weiteres möglich ist und nur die Schnittstellen angepasst werden müssen.[11]

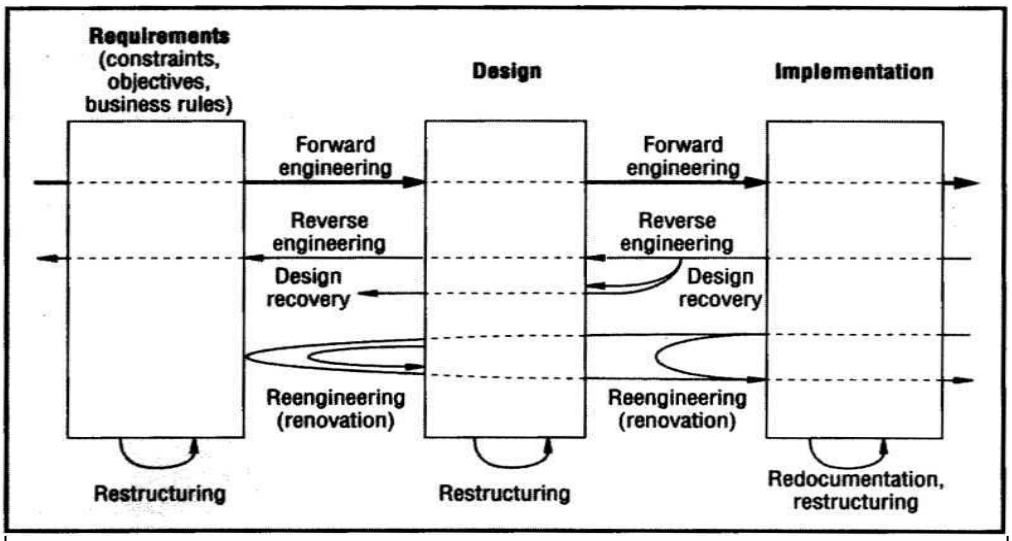
Wie kann man aber einen oder mehrere historisch gewachsene Softwaremonolithen mit zum Teil fehlender oder nicht aktualisierter Dokumentationen in das neue Architekturprinzip überführen? Man könnte sich einfach aller „Altlasten“ entledigen, indem man sie abschaltet und sich etwas Neues überlegt. Es gibt aber durchaus sinnvolle Alternativen zum radikalen Verwerfen des Bestehenden. Eine Lösung bietet das Reengineering.

Der Grundgedanke des Reengineering besteht darin, vorhandene Ressourcen zu analysieren, zu überarbeiten und auf bereits vorhandenen Komponenten aufzubauen.[12] So kann bereits vorhandenes internes Wissen, zum Beispiel über Workflows oder Analysemethoden, aus den Legacy-Anwendungen extrahiert werden.

Eine genauere Beschreibung des Reengineering bezeichnet es als die Untersuchung und Veränderung eines Systems mit dem Ziel, dieses in neuer Form wiederzuerschaffen.[13] Die Verwertung des Vorhandenen ist besonders in einem dynamischen Umfeld mit häufigen Änderungen wesentlich effizienter und vor allem kostengünstiger, als alles zu verwerfen und neu zu beginnen. Dies gilt vor allem, wenn man berücksichtigt, dass Legacy-Anwendungen funktionieren und aufgrund ihrer strategischen Bedeutung noch von Wert sind.[14] Nachforschungen der Gartner Group bestätigen diese Ansicht. So bauen beispielsweise die meisten SOA-Projekte mittels Reengineering auf Legacy-Anwendungen auf.[15]

---

[10] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 139.
[11] Vgl. Richter, Jan-Peter Dr. u.a. (2005), Hauptframe 4. Absatz (Internetquelle).
[12] Vgl. Mertens, Peter u.a. (Hrsg.) (2001), S. 419.
[13] Vgl. Chikofsky, Elliot J. und Cross, James H. (1990), S. 15 (Internetquelle).
[14] Vgl. Sneed, Harry M. (2006), S. 4 (Internetquelle).
[15] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 139.

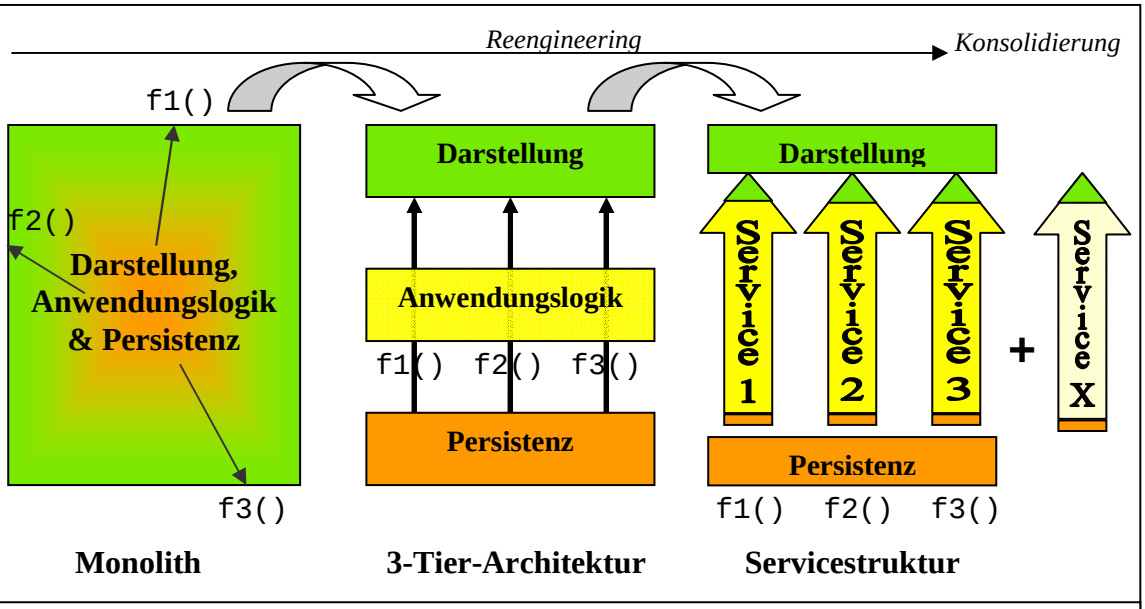


Quelle: Chikofsky, Elliot J. und Cross, James H. (1990), S. 14 (Internetquelle).

Abb. 1: Begriffe des Reengineering in den Phasen der Anwendungsentwicklung

Eine Übersicht über verschiedene Methoden des Reengineering bietet die Abbildung 1. Allgemein lässt sich sagen, dass das Reverse Engineering als eine dieser Methoden bei der Bearbeitung von Legacy-Anwendugnen oft eine entscheidende Rolle spielt. Es wird eine abstrakte Betrachtungsform des Systems angefertigt. Dabei kann die Abstraktion in mehreren Schritten erfolgen, sodass man zunächst das Design der Anwendung analysiert und schließlich sogar die Anforderungen abstrahiert.[16] Für die Realisierung eines Zielsystems muss man sich von der Implementierung durch die Erstellung eines abstrakten Musters in den konzeptionellen Bereich zurückbewegen. In der jeweiligen Abstraktionsebene kann man die Methode Restrukturierung auf das jeweilige Muster anwenden, um Änderungen zu bewirken. Das Reengineering wird schließlich mit der Umsetzung der vorgenommenen Eingriffe abgeschlossen. Die Abbildung 1 verdeutlicht die Verfahrensweise und den Zusammenhang der Begriffe. Eine genauere Behandlung aller Begriffe folgt.
Dank der abstrakten Betrachtungsweise beim Reengineering ist es möglich, ein monolithisches System in mehreren Schritten sowohl technisch als auch funktional zu zerlegen. Technisch könnte ein Monolith somit in eine 3-Schicht-Applikation mit strikter Trennung von Datenhaltung, Anwendungslogik und Darstellung überführt werden. Betrachtet man anschließend die einzelnen gegeneinander abgegrenzten Funktionsbausteine entkoppelt von den Schichten, lassen sich daraus die potenziellen Services ableiten.[17] Hat man erst einmal diese Abstraktionsebene erreicht, sollten sich Komponenten aus anderen Systemen

[16] Vgl. Mertens, Peter u.a. (Hrsg.) (2001), S. 419.
[17] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S.140.



Quelle: Eigene Darstellung

Abb. 2: Die Umwandlung der Monolith-Funktionen f1, f2 und f3 in Services

leicht einbeziehen lassen, sodass man auf fachlicher Ebene mehrere Anwendungen konsolidieren kann.[18] Aufgrund der losen Kopplung der Komponenten kann die neu gebildete Anwendung beliebig erweitert und geformt werden. Die Abbildung 2 veranschaulicht diesen Vorgang.

2.1.2 Reverse Engineering

Das Reverse Engineering versteht sich als Werkzeug des Reengineering. Oberste Prämisse dieser Methode ist, dass alle Handlungen analytisch sind. Reverse Engineering dient der Untersuchung und Abstraktion eines Systems mit dessen Komponenten und deren Beziehungen untereinander, nicht aber dessen Nachbildung oder Veränderung. Anwenden lässt sich die Methode in nahezu jeder Phase des Softwareentwicklungsprozesses. Man muss also nicht mit einem operativen System beginnen.[19] Das Prinzip konstanter Verbesserung durch Rückschritte innerhalb des Entwicklungsprozesses stammt aus der agilen Softwareentwicklung.[20]

Im Zusammenhang mit Reverse Engineering wird häufig auf die Verfahrensweisen Redocumentation und Design Recovery verwiesen. Per Definition von Chikofsky und Cross entsteht bei der Redocumentation eine semantisch mit dem analysierten System übereinstimmende Repräsentation entsprechend der relativen Abstraktionsebene. Mit diesem ein-

[18] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 303.
[19] Vgl. Chikofsky, Elliot J. und Cross, James H. (1990), S. 15 (Internetquelle).
[20] Vgl. Sommerville, Ian (2007), S. 439.

fachen Verfahren erschafft man für den Menschen verständliche alternative Betrachtungsweisen des Systems in Form einer Dokumentation mit Text und / oder Bildern.[21] Gemeint ist damit zum Beispiel eine Code-Dokumentation, oder abstrakter betrachtet eine Funktionsbeschreibung, Komponentendarstellungen oder Use Cases.

Die Ansicht, dass Redocumentation auch als schwache Restrukturierung angesehen werden kann, ist allerdings eher kritisch zu betrachten.[22] Denn die Abstraktion des Systems sollte getreu dem rein analytischen Grundsatz des Reverse Engineering noch keine alternativen Implementierungen beinhalten, sondern ausschließlich dem Verständnis dienen. Streng betrachtet, dürfte dann aber bei der Erstellung der fachlichen Sichten eines Systems auch kein Fehler unterlaufen, da dies sonst schon eine Änderung im Sinne des Restrukturierens bedeuten könnte. Die klare Abgrenzung der Begriffe scheint an dieser Stelle stark von der Realitätsnähe und Präzision der durchgeführten Analyse abhängig zu sein.

Das Design Recovery geht noch über die reine Systembetrachtung hinaus. Durch zusätzliche Berücksichtigung von Fach- und Expertenwissen ebenso wie allgemeinem Fachwissen bis hin zu Schlussfolgerungen und Vermutungen wird eine noch höhere Abstraktionsebene angestrebt.[23] Letztendlich soll gewährleistet sein, dass man das System, wie in Abbbildung 3 dargestellt, vollständig versteht.[24] Ein Vorteil ist hier, dass Gestaltungsmöglichkeiten über die ursprünglich umgesetzten Anforderungen hinaus erkennbar werden können. Die

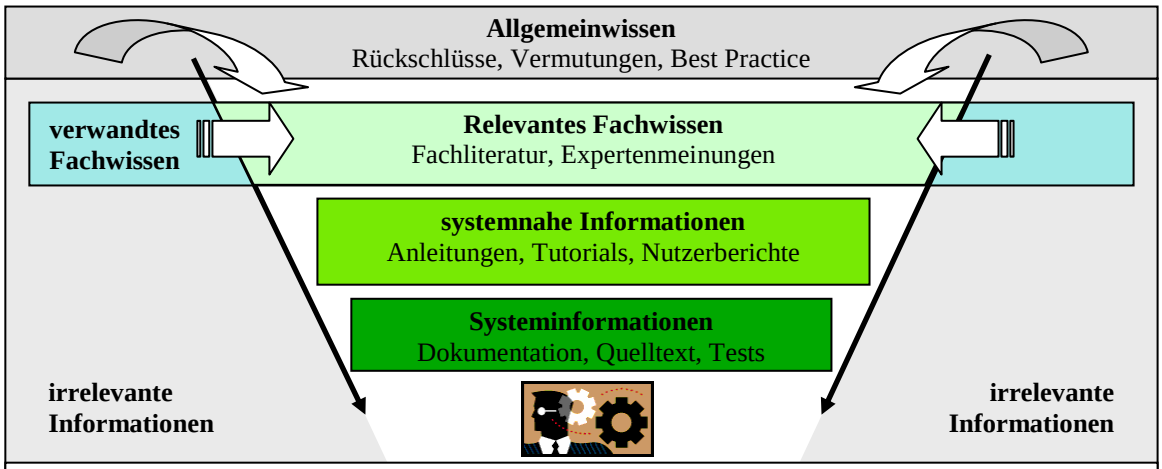


Abb. 3: Mögliche Einflüsse auf das Systemverständnis beim Design Recovery

[21] Vgl. Chikofsky, Elliot J. und Cross, James H. (1990), S. 15 (Internetquelle).
[22] Vgl. ebenda, S. 15.
[23] Vgl. ebenda, S. 15.
[24] Vgl. Biggerstaff, Ted J. (1989), in: Chikofsky, Elliot J. und Cross, James H. (1990), S. 15 (Internetquelle).

umfassende Betrachtung würde es für eine Forschungsanwendung beispielsweise ermöglichen, vorher nicht berücksichtigte Berechnungsmethoden in die Anforderungen mit aufzunehmen. Restrukturierungen könnten dann an diesem Punkt mit vorher nicht erkennbaren Methoden wie der Konsolidierung ansetzen. Wenn man jedoch nicht sorgfältig aus den sich neu ergebenden Möglichkeiten auswählt und das Wesentliche nicht vom Unwesentlichen trennt, könnte die große Menge an zusätzlichen Informationen eine nachteilige Abweichung von den tatsächlich gestellten Anforderungen bewirken. Das auf Grundlage dessen restrukturierte System wäre dann over-engineered.[25] Wie schon bei der Redocumentation geht es auch beim Design Recovery zunächst nur um das Systemverständnis.
Als besonders hilfreich scheint sich das Design Recovery zu erweisen, wenn man nur begrenzten oder teilweisen Zugang zu den Systeminterna wie dem Programmcode hat, welcher ohnehin nur wenige Rückschlüsse auf das Design zulässt. In solchen Fällen muss man auf andere Informationsquellen zurückgreifen. Geeignet scheinen dafür dann Vermutungen und Rückschlüsse aus erkennbaren Zusammenhängen und Testfällen. Man betreibt quasi ein abstraktes Reverse Engineering unter der Verwendung von Blackbox-Testverfahren.[26] Dabei wird ein System nur auf nach außen beobachtbares Verhalten untersucht. In Testfällen werden dazu verschiedene Werte eingegeben oder bestimmte Aktionen ausgelöst und die Ausgaben oder Reaktionen des Systems beobachtet. Das beobachtete Verhalten wird dann mit dem erwarteten Verhalten verglichen. Der Programmcode bleibt dabei bewusst unberücksichtigt.[27] Auf diese Weise wird der Versuch unterstützt, sich mit wenigen Informationen an den ursprünglichen Designentwurf anzunähern. Dabei wird trotz der vorhandenen Grauzone die bestmögliche Genauigkeit angestrebt.
Ist das Design ermittelt, beziehungsweise festgelegt, können konkrete Aktivitäten zur Umgestaltung folgen.

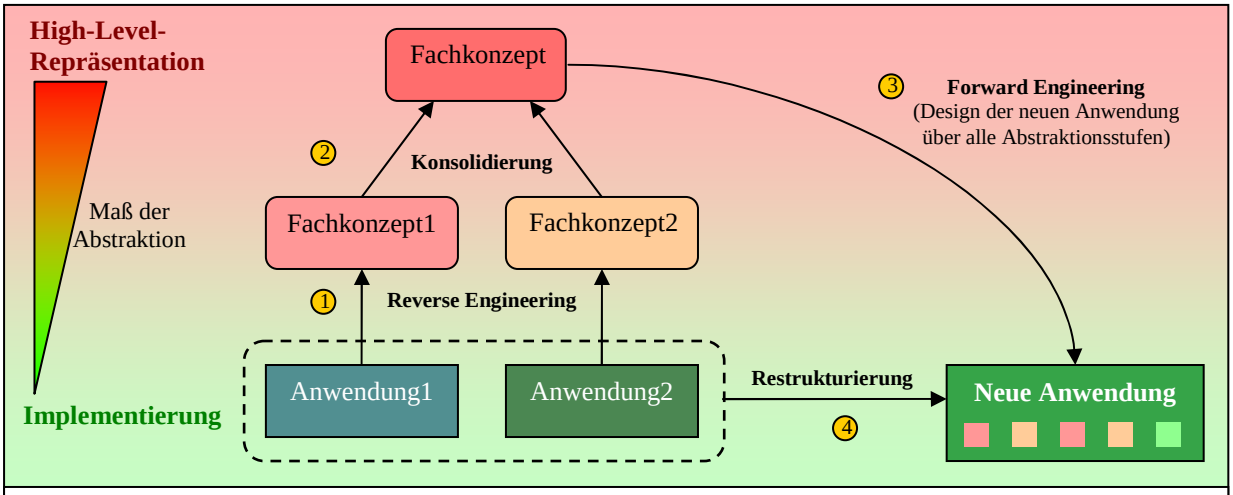
2.1.3 Restrukturierung mit Konsolidierung

Umgestaltet wird ein System mit den Methoden der Restrukturierung. Sie ist die Überführung eines analysierten Musters in eine andere Form innerhalb einer Abstraktionsebene der Softwareerstellung. Das Verhalten des Systems muss dabei nach außen allerdings unverändert bleiben.[28]

---

[25] Vgl. Kerievsky, Joshua (2005), S. 1 f.
[26] Vgl. Seacord, Robert S. (2003), S. 9.
[27] Vgl. Fischer, Joachim u.a. (2002), S. 397.
[28] Vgl. Chikofsky, Elliot J. und Cross, James H. (1990), S. 15 (Internetquelle).

Beispielsweise tauscht man bei einer Anwendung die alte grafische Benutzeroberfläche gegen eine neuere und übersichtlichere mit weniger Speicherverbrauch aus. Häufig werden Restrukturierungen innerhalb der Implementierungsebene durch das Austauschen von Algorithmen und Programmkomponenten vollzogen. In diesem Fall spricht man von Refactoring.[29] Aufgrund der unterschiedlichen Betrachtungsdimensionen können aber auch Änderungen des Datenmodells, des Architekturdesigns oder sogar der Anforderungen vollzogen werden.[30] An dieser Stelle wird die Definition jedoch etwas kritisch. Denn geht man dazu über, funktionale Anforderungen zu modifizieren, ist es laut Definition keine Restrukturierung mehr. Besser wäre zu sagen, dass die Restrukturierung mit ihren verschiedenen, abstrakten Betrachtungsebenen Möglichkeiten zur Aufdeckung potenzieller funktionaler Verbesserungen aufzeigt, sodass Details des Verhaltens nach außen verbessert werden können. Das Grundwesen der Verhaltensweise sollte aber dennoch erhalten bleiben.
Erweitert man das Verständnis von Verbesserung, könnte man darunter auch verstehen, dass die Anwendung durch das Hinzufügen vorher nicht vorhandener Funktionen verbessert wird. Aus der neuen Betrachtungsweise erwachsen möglicherweise neue Anforderungen, die vorher in diesem Zusammenhang nicht erkenntlich waren. Es ist also denkbar, diese Anforderungen zusätzlich einfließen zu lassen.[31] Betrachtet man diese Vorgehensweise noch abstrakter, ist es sogar denkbar, zwei oder mehrere verschiedene Anwendungen auf dieser hohen Abstraktionsebene zusammenzuführen.



Quelle: In Anlehnung an: Bauer, Marcus u.a. (2003), Folie 4 (Internetquelle).

Abb. 4: Konsolidierung von Anwendungen mit Reengineeringmethoden

[29] Vgl. Kerievsky, Joshua (2005) und Fowler, Martin (2005).
[30] Vgl. Chikofsky, Elliot J. und Cross, James H. (1990), S. 15 (Internetquelle).
[31] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 303.

Diese Auslegung und Erweiterung der Definition ermöglicht also das vorher bereits erwähnte Konsolidieren von Anwendungen. Man stelle sich dazu ein System wie in Abbildung 4 vor, in dem sich zwei Anwendungen befinden, welche prinzipiell ähnlich arbeiten. Aus diversen Gründen sind diese Anwendungen allerdings unabhängig von einander auf unterschiedlichen technologischen Plattformen entstanden und weiterentwickelt worden. Beide Anwendungen sollen unter einer serviceorientierten Architektur zusammengeführt werden. Dazu ist es denkbar, das Reverse Engineering durchzuführen. Mit den gewonnenen Erkenntnissen kann eine technologische Plattform gefunden werden, welche alle Kundenanforderungen an beide Anwendungen erfüllt. Ist dies gelungen, so kann man beginnen, die funktional ähnlichen Eigenschaften beider Anwendungen zusammenzuführen. Die konsolidierte High-Level-Repräsentation wird anschließend auf die neue Plattform übertragen, indem das zuvor für alle Abstraktionsebenen angepasste Design mit den darauf basierenden Implementierungsdetails umgesetzt wird.[32]
Dieser Reengineeringprozess ist sehr umfangreich und aufwändig, jedoch lohnt sich dieser Einsatz im Hinblick auf den verringerten Wartungsaufwand sowie die gewonnene Flexibilität der modernen Servicestruktur.

2.2 Allgemeine Softwareanforderungen

Wie bereits in der vorangegangen Betrachtung erwähnt wurde, betrifft das Reengineering von Legacy-Software mit Anwendungskonsolidierung auch die Anforderungen an die jeweilige Software.[33]
Mittels Reverse Engineering können die ursprünglichen Anforderungen an das jeweilige System hergeleitet werden. Diese werden zusammengeführt und bilden dann die Anforderungen, nach welchen das neue System geschaffen wird. Dabei sind kundendefinierte, funktionale Anforderungen von den nicht-funktionalen, technischen zu unterscheiden. Weiterhin unterscheidet Sommerville Benutzeranforderungen und Systemanforderungen.[34]
Im Vordergrund der Betrachtung stehen allerdings eher die etwas unspezifischeren Benutzeranforderungen, welche in der Fallstudie analysiert werden sollen. Benutzeranforderungen sind unabhängig von Softwaresprachen und enthalten noch keine Implementierungsdetails. Außerdem sind sie leicht zu verstehen und insgesamt durch die technologisch ungebundene Spezifikation flexibel anwendbar.[35]

---

[32] Vgl. Sommerville, Ian (2007), S. 542 ff.
[33] Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 303.
[34] Vgl. Sommerville, Ian (2007), S.150 ff.
[35] Vgl. ebenda, S. 159 ff.

2.2.1 Funktionale Anforderungen

Grundsätzlich beschreiben funktionale Anforderungen unabhängig von der Detaillierung die Dienstleistung einer Anwendung. Beschrieben werden primär die Funktionen, welche zur Verwendung bereitgestellt werden. Des Weiteren wird ein Verhalten für den Umgang mit Ein- und Ausgabewerten definiert. Es wird festgelegt, was in das System eingegeben werden kann und in welcher Form dann etwas dazu ausgegeben wird. Dazu gehört auch, wie das System auf Ausnahmen reagiert. Für spezielle Situationen, die nicht der Standardhandhabung entsprechen, muss klar sein, wie sich die Anwendung verhält. In einigen Fällen werden die funktionalen Anforderungen auch dazu benutzt, abzugrenzen, was die Anwendung nicht leisten wird.[36] Das erscheint vor allem dann sinnvoll, wenn die Anwendung in einer sehr flexiblen und vielseitigen Umgebung zum Einsatz kommt. Den Nutzern wird dadurch vermittelt, was sie von der Anwendung erwarten können und was nicht. Abbildung 5 zeigt ein komplexes Beispiel einer funktionalen Anforderung.

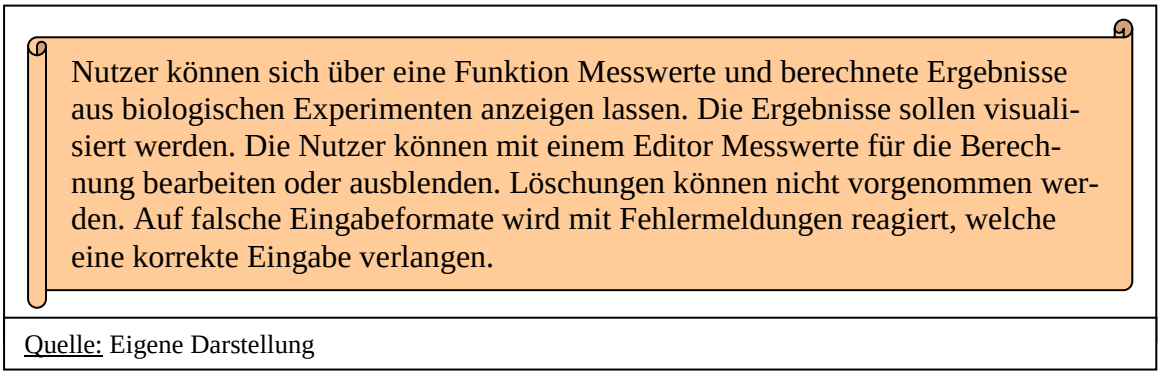


Abb. 5: Beispiel einer funktionalen Anforderung an eine Analyseanwendung aus der biologischen Forschung

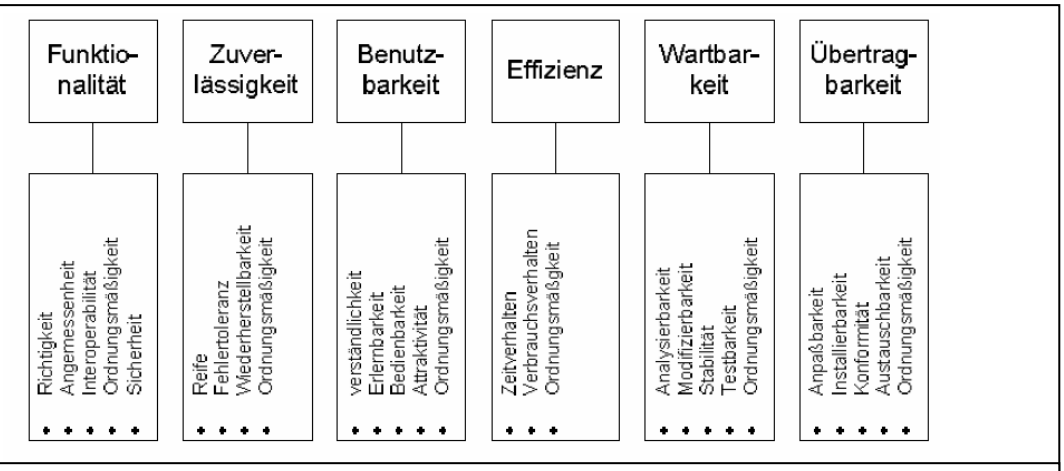
In Reegineering-Projekten sind die funktionalen Anforderungen in der Regel vorgegeben, sei es in der Form, dass sie konkret vorliegen oder durch Reducumentation ermittelt werden, also indirekt in der Anwendung enthalten sind. Durch die Methoden des Reengineering speziell im Zusammenhang mit Konsolidierung werden sie jedoch auch in gewisser Weise neu definiert beziehungsweise umgestaltet.[37] Daher ist es zum Beispiel wichtig, die Betrachtung nicht zu detailliert zu gestalten. Die funktionalen Anforderungen unterschiedlicher Anwendungen können nur auf einer abstrakten Ebene zusammengeführt werden. Zu allgemeine Spezifikationen eröffnen allerdings auch Spielräume für fehlerhafte Anforde-

[36] Vgl. Sommerville, Ian (2007), S. 152.
[37] Vgl. Mens, Tom und Demeyer, Serge (Eds.) (2008), S. 303.

rungsumsetzungen. Eine geeignete Präzision ist zu bestimmen.[38] Außerdem sollte auf Vollständigkeit und Konsistenz geachtet werden, vor allem dann, wenn man nicht auf vordefinierte Anforderungen zurückgreifen kann. [39] Schließlich soll durch die Konsolidierung zwar eine effektivere Anwendung entstehen, jedoch nicht auf Kosten wichtiger Funktionalitäten, welche schlichtweg vergessen wurden.

2.2.2 Nicht-funktionale Anforderungen

Nicht-funktionale Anforderungen beziehen sich nicht darauf, was ein System leistet, sondern wie es seine Funktionen ausführt. Sie beschreiben Systemeigenschaften, die unabhängig von einzelnen Funktionen sind.[40] In der ISO-Norm 9126 sind die in Abbbildung 6 dargestellten Qualitätsmerkmale definiert.



Quelle: Schmietendorf, Andreas Prof. Dr. (2005), Folie 7, (Internetquelle).

Abb. 6: Nicht-funktionale Softwareanforderungen nach ISO 9126

Einige Beispiele für abstrakte nicht-funktionale Anforderungen speziell an Anwendungen aus der Pharmaorschung sind in Tabelle 1 dargestellt.
Diese und viele andere Eigenschaften sind teilweise so kritisch, dass die Brauchbarkeit des gesamten Systems gefährdet sein könnte, sollte eine Eigenschaft nicht erfüllt sein.[41] So wird zum Beispiel eine Echtzeit-Analyseanwendung mit sehr langen Reaktionszeiten und geringer Verfügbarkeit wegen ständigen Ausfällen kaum Chancen auf dem Markt haben, beziehungsweise vom Kunden abgelehnt werden. Daher sollte diese Art von Anforderung idealer Weise immer unter der Verwendung von Metriken wie zum Beispiel Reaktionszei-

[38] Vgl. Sommerville, Ian (2007), S. 153.
[39] Vgl. ebenda, S.153.
[40] Vgl. ebenda, S. 152.
[41] Vgl. ebenda, S. 154.

ten in Sekunden oder Speicherverbrauch in Megabyte verfasst werden. Dies ermöglicht später eine objektive Bewertung der Erfüllung durch Tests des Systems.[42]
Nicht-funktionale Anforderungen können sich neben dem System auch noch auf die Vorgehensweise bei der Entwicklung durch den Bezug auf Qualitätsmerkmale, Standards und Methoden beziehen.[43]

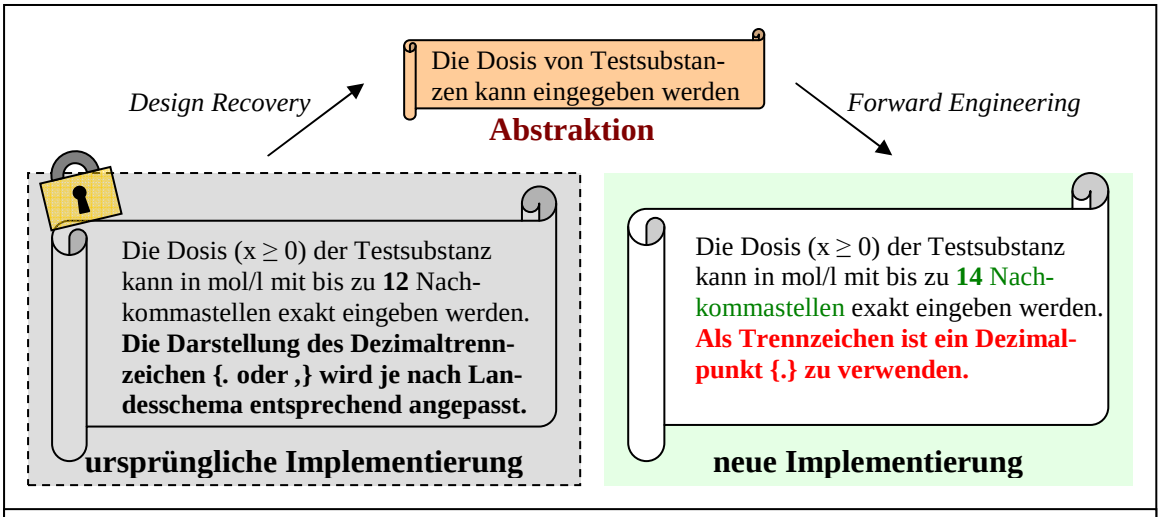
Tabelle 1: Einige Beispiele für nicht-funktionale Anforderungen an Forschungsanwendungen
hohe Zuverlässigkeit durch Wiederherstellbarkeit innerhalb von Stunden nach Ausfällen
hohe Sicherheit zum Schutz sensibler Daten
intuitive Bedienbarkeit, dadurch Erlernbarkeit innerhalb weniger Stunden
Reaktionszeiten innerhalb von Sekunden, komplexe Auswertungen innerhalb von wenigen Sekunden
Speicherverbrauch weniger Megabytes pro Experiment
hohe Flexibilität und Kompatibilität zur Anpassung an veränderte Rahmenbedingungen innerhalb von Tagen
Einhaltung aller rechtlichen und ethischen Vorschriften zu jeder Zeit
Quellen: Vgl. Sommerville, Ian (2007), S. 154 ff.

2.2.3 Der Einfluss des Architektur-Reengineering auf Softwareanforderungen

An dieser Stelle ist es interessant und sinnvoll, sich speziell mit möglichen Einflüssen des architektonischen Restrukturierens auf die Anforderungen auseinanderzusetzen. Die dabei gewonnenen Erkenntnisse über die Auswirkungen können dann in dem folgenden Fallbeispiel berücksichtigt werden.
Nach den bereits diskutierten Grundsätzen der Reengineeringmethoden soll die Verhaltensweise des zu bearbeitenden Systems nach außen unverändert bleiben. Man könnte allerdings argumentieren, dass aufgrund des Reverse Engineering bis hin zum Anforderungsniveau wegen des Spielraumes für Vermutungen die ursprünglichen funktionalen Softwareanforderungen unter Umständen gar nicht mehr reproduzierbar sind. Es könnten also unbeabsichtigt Veränderungen herbeigeführt werden.
In einem einfachen Beispiel wird eine implementierte, aber nicht dokumentierte, funktionale Anforderung wie in Abbildung 7 durch die Verwendung der Methode Design Recovery aus einer Forschungsanwendung ermittelt. Ausgehend von der abgeleiteten Spezifikati-

[42] Vgl. Sommerville, Ian (2007), S. 156.
[43] Vgl. ebenda, S. 154 ff.

on wird das System mit Restrukturierung und/oder Forward Engineering umgestaltet, beziehungsweise sogar noch erweitert. Die abstrakte Sichtweise hat hier neue Möglichkeiten eröffnet. Mit der Konkretisierung der ermittelten funktionalen Anforderung geht allerdings ein Teil der ursprünglichen Forderung verloren. Die automatische Dezimalzeichenanpassung an das jeweilige Eingabegebietsschema ist der einfachen Festlegung der Verwendung eines Dezimalpunktes gewichen. Dies ist im Sinne der Systemverbesserung sicherlich nicht beabsichtigt. Jedoch ist es für den Systemanalytiker in dieser Situation nicht vermeidbar, da bereits beim Design Recovery eine gewisse Grauzone vorhanden ist, wenn das Originaldesign nicht vorliegt. Die neue Betrachtungsweise gestattet aber auch positive Veränderungen wie zum Beispiel eine Erweiterung der maximalen Länge der eingegebenen Ziffernfolge auf bis zu 14 Nachkommastellen.
Es lässt sich also festhalten, dass sich die funktionalen Anforderungen sowohl in positiver als auch negativer Weise beeinflussen lassen, sobald das System einem derartigen Reengineering-Prozess unterzogen wird.



Quelle: Eigene Darstellung

Abb. 7: Veränderung einer funktionalen Anforderung durch Reengineeringmethoden

Die nicht-funktionalen Anforderungen werden beim Architektur-Reengineering ebenfalls verändert. Schließlich ist zum Beispiel eine Verbesserung der Leistungsfähigkeit gewünscht.[44] Hier kollidiert die Vorgehensweise allerdings erneut stark mit dem in der ursprünglichen Definition des Restrukturierens geforderten Beibehalten des äußeren Verhaltens. Einen Ausweg bietet die bereits erläuterte Erweiterung des Begriffsverständnisses.

[44] Vgl. Mens, Tom und Demeyer, Serge (2008), S. 97.

Diese Begriffsauslegung ist notwendig, da eine Restrukturierung hauptsächlich mit der Absicht der Optimierung des bisherigen Verhaltens durchgeführt wird und sonst der Aufwand nicht zu rechtfertigen wäre. In Bezug auf das Reengineering von Architekturen muss das Verständnis von einem „äußeren Verhalten“ folglich mehr an den funktionalen Anforderungen festgemacht werden. Doch auch dabei kommt man nicht an einer Erweiterung der Definition vorbei, sodass letztendlich festgehalten werden muss, dass das Verhalten des Systems nach außen mindestens erhalten werden muss, aber durchaus optimiert werden kann und sollte.

3. Fallbeispiel zur Anwendungsanalyse

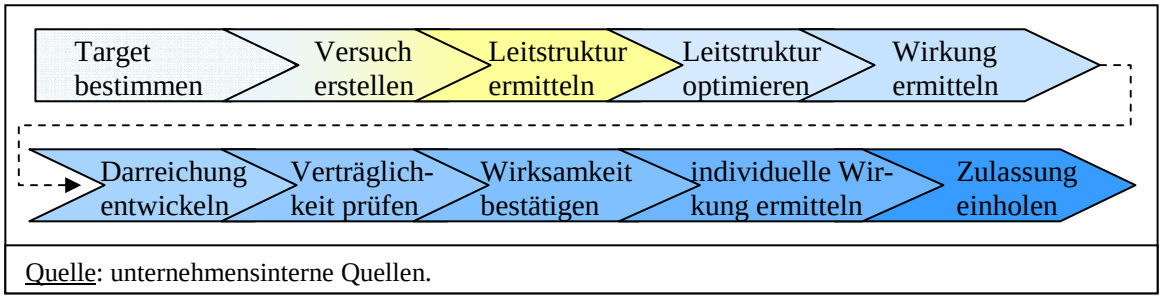
3.1 Vorbetrachtung

Zum besseren Verständnis der Fallstudie wird zunächst ein kurzer Überblick über die Pharmaforschung und mögliche Rahmenbedingungen gegeben.

3.1.1 Methoden der Pharmaforschung

Ein wichtiger Bestandteil der Wirkstofffindung in Pharma-Unternehmen sind biologische Tests von Substanzen zur Feststellung ihrer hemmenden beziehungsweise verstärkenden Wirkung bei krankheitsfördernden beziehungsweise -behandelnden Reaktionen zwischen Enzymen und Substraten. Täglich werden tausende von Proben von modernen Messgeräten untersucht. Die dabei gewonnenen Messwerte werden festgehalten, analysiert, aufbereitet und schließlich für Vergleichs- und Dokumentationszwecke elektronisch archiviert.[45] Um die Arbeit mit Analyseanwendungen der Forschung zu verstehen, ist es daher notwendig, auch den in den Anwendungen abgebildeten Ablauf der Forschung zu erklären.

3.1.2 Workflow



Quelle: unternehmensinterne Quellen.

Abb. 8: Einordnung des betrachteten Forschungsgebiets in den Prozess der Medikamententwicklung

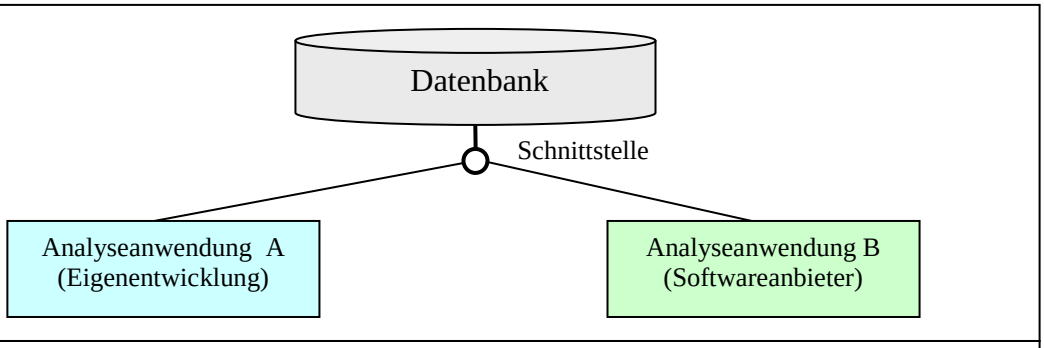
Ein Teil des in Abbildung 8 dargestellten Prozesses bis zur Entwicklung eines Medikaments befasst sich mit dem Ermitteln von so genannten Leitstrukturen. Dazu muss zunächst ein Target, also ein Zielprotein im Körper gefunden werden, an dem sich die Leitstruktur anlagern kann, um eine bestimmte Wirkung zu erzielen. Mit verschiedenen Untersuchungsmethoden werden diejenigen Proben ermittelt, welche ein im Voraus definiertes Maß an Aktivität aufweisen. Diese werden in einer Liste potenziell wirksamer Kandidaten festgehalten.[46]

[45] Unternehmensinterne Quellen.
[46] Ebenda.

Die Hitliste bildet die Grundlage für einen erneuten Test, bei dem die Substanzen zur Bestätigung der ersten Untersuchung in mehrfacher Ausführung mit unterschiedlichen Konzentrationen der Leitstruktur erneut getestet werden. Die bestätigten Treffer werden anschließend im nächsten Experiment noch genauer getestet. Dies dient der weiteren Bestätigung der Wirksamkeit und bietet die Grundlage für die Berechnung von verschiedenen Ergebniskennzahlen in den Analyseanwendungen. Diese begleiten den Ablauf eines Experiments und dokumentieren die Ergebnisse. Dadurch sollen alle Vorgänge langfristig nachvollziehbar gemacht werden.[47]

3.1.3 Rahmenbedingungen

Die Abbildung des zuvor beschriebenen Workflows erfordert ein Zusammenspiel mehrerer Komponenten. Daher müssen in einer umfassenden Anwendungsanalyse auch die Softwarearchitektur und die eingesetzten Technologien näher betrachtet werden.



Quelle: Eigene Darstellung

Abb. 9: Darstellung einer möglichen Systemarchitektur

In diesem vereinfachten Beispiel existieren zwei Forschungsanwendungen A und B zu Analysezwecken. Sie wurden bereits in die gleiche Umgebung integriert und arbeiten mit derselben Datenbank. Abstrakt betrachtet erfüllen beide Anwendungen denselben Zweck, nämlich Datenaufnahme zur Verarbeitung und Speicherung biologischer Testergebnisse. Technisch unterscheiden Sie sich jedoch etwas in der Umsetzung des Workflows, der Funktionalität und der softwaretechnischen Realisierung. Die Anwendung B wurde nach Nutzeranforderungen von einem Dienstleister erstellt. Die Anwendung A ist historisch gewachsen basierend auf einem einfachen MS Excel-Makro, welches zunächst in einem Labor von einem Forschungsmitarbeiter geschrieben wurde.

[47] unternehmensinterne Quellen.

Im Hinblick auf die Einführung einer service-orientierten Architektur wird die Systemlandschaft umstrukturiert. Die beiden Anwendungen müssen entsprechend angepasst werden. Mittelfristig sollen beide Anwendungen konsolidiert werden, um Redundanzen zu beseitigen. Die einzelnen Funktionen sollen als modulare Webservices zur Verfügung stehen. Dazu ist eine Analyse beider Anwendungen durchzuführen, um die Anforderungen an die konsolidierte service-orientierte Zielstruktur erfassen zu können. Wegen des Verzichts auf Einsicht in den Quellcode sind geeignete Testfälle zur Analyse der Struktur, der Darstellungsform und besonderer Eigenschaften zu erstellen. Für jede der beiden Anwendungen muss je ein Fachkonzept nach den Methoden des Reengineering ermittelt werden. Diese Konzepte bilden die Grundlage für die Konsolidierung, aus der eine Handlungsempfehlung abzuleiten ist.

3.2 Erstellung des Analysemodells

Allgemeine Anforderungen an die Konsolidierung der zwei Forschungsanwendungen sind bereits vor Beginn der Analyse festzuhalten. Somit können sie bei der Erstellung des Fachkonzepts berücksichtigt werden.

3.2.1 Allgemeine Nutzeranforderungen

Die allgemeinen Nutzeranforderungen orientieren sich an gängigen Überlegungen bei der Softwareentwicklung. Die erfassten Anforderungen unterteilen sich in verschieden Kategorien wie Anforderungen an die Benutzeroberfläche, Umgang mit Daten, Zugriffssteuerung sowie Vernetzung und Architektur. Angepasst an die spezielle Systemlandschaft ergeben sich für die Konsolidierung der zwei Applikationen die beispielhaft in der Tabelle 2 dargestellten allgemeinen funktionalen Anforderungen.

Wichtig bei der Erfassung der Oberflächenmerkmale ist die Berücksichtigung firmeninterner Designvorschriften. Die Gestaltung soll sich einheitlich in das bestehende System integrieren lassen und dem Nutzer eine gewohnte Handhabung mit bekanntem Look und Feel ermöglichen.

Ein zentraler Aspekt der Anforderungen ist die modulare Gestaltung der verfügbaren Funktionen. Aus den ermittelten Anforderungen der beiden Analyseanwendungen sollen von einander unabhängige Services gebildet werden, die leicht wartbar, erweiterbar oder austauschbar sind. Die Modularisierung ermöglicht diese hohe Flexibilität.[48]

[48] Vgl. IT Wissen Online Lexikon (2008), b) Hauptframe, 1. Absatz (Internetquelle).

Tabelle 2: Nutzeranforderungen an die konsolidierte Forschungsanwendung	
Benutzeroberfläche	Graphical User Interface (GUI) unter Berücksichtigung von softwareergonomischen Anforderungen und Corporate Design Vorschriften
	Verschiedene Anzeigeoptionen für unterschiedliche Auswertungen
	Sprache: Englisch
	Kontextsensitive Hilfestellungen
	Menügeführte Eingabeoptionen
Datenbehandlung	Editierfunktionen (Kopieren, Einfügen, Bearbeiten und Löschen von Feldern)
	Eingabe mindestens über Tastatur und Maus
	Formatfreie Dateneingabe (z.B. Interpretation des Dezimalzeichens je nach Eingabegebietsschema)
	Eingabeprüfung (syntaktisch und semantisch)
	Genormte Schnittstellen zum Datentransfer mit der Systemumgebung
	Kontrolle auf doppelte Datensätze
Zugriffskontrollen	Vergabe von Zugriffsrechten (Lesen, Schreiben, Rechtevergabe)
	Einschränkungen hinsichtlich Nutzergruppen und Bearbeitungsstände
Netzwerk-Zugänge	Anschluss an Firmennetz weltweit (alle Forschungsfachabteilungen)
	Verteilte Datenhaltung mit zentraler Ablage
Architektur	Modularer Funktionsaufbau (Servicestruktur)
	Trennung von Darstellung, Anwendungslogik und Datenhaltung
Quelle: Eigene Tabelle	

3.2.2 Allgemeine Technische Anforderungen

Die allgemeingültigen technischen, beziehungsweise nicht-funktionalen Anforderungen sollten aus Qualitätsgründen bei jedem Softwareprojekt angewendet werden. Eine gute Empfehlung ist die ISO-Norm 9126, die in Tabelle 3 abstrahiert auf die konkreten Gegebenheiten der Konsolidierung der zwei Forschungsapplikationen angewendet wurde. Einige der beschriebenen Anforderungen sind fast selbstverständlich, so kann von einem System immer erwartet werden, dass es korrekt und sachgemäß funktioniert. Im Bezug auf die Interoperabilität ist zu erwähnen, dass die Schnittstellen so gestaltet sein müssen, dass ein Austausch von Daten ohne komplexe Umrechnungsprozesse stattfinden kann.

Tabelle 3: Nicht-funktionale Anforderungen an die konsolidierte Forschungsanwendung nach ISO 9126		
Funktionalität	Richtigkeit	Richtige Ergebnisse und geforderte Genauigkeit
	Angemessenheit	Funktionen erfüllen ihren Zweck
	Interoperabilität	Interagiert mit anderen Auswertungssystemen und Datenbanken
	Ordnungsmäßigkeit	Keine unerwarteten Ausnahmen
	Sicherheit	Erfüllung von IT-Sicherheitsrichtlinien (allgemein und unternehmensspezifisch)
Zuverlässigkeit	Reife	Geringe Versagenshäufigkeit durch gute Fehlerbehandlung
	Fehlertoleranz	Funktion auch bei schweren Ausnahmefehlern aufrecht erhalten
	Widerherstellbarkeit	Schnelle Wiederherstellung ohne Datenverluste
Benutzbarkeit	Verständlichkeit	Leicht zu verstehen durch Abbildung des Forschungsworkflows
	Erlernbarkeit	Mit wenig Übung leicht zu erlernen
	Bedienbarkeit	Menüführung und Eingabehilfen, bzw. präzise Anweisungen
Effizienz	Zeitverhalten	Einige Sekunden Antwortzeit, sogar bei großem Durchsatz
	Verbrauchsverhalten	geringer Aufwand für Dateizugriffe, geringe Belegung von Speicher und CPU-Zeit
Wartbarkeit	Analysierbarkeit	Geringer Aufwand bei der Suche nach Fehlern und/oder Verbesserungsmöglichkeiten
	Modifizierbarkeit	Geringer Aufwand zur Durchführung von Änderungen
	Stabilität	Geringe Fehlerwahrscheinlichkeit nach Änderungen
	Testbarkeit	Geringer Aufwand zum Testen von Änderungen
Übertragbarkeit	Anpassbarkeit	Leichte Anpassung an die Systemumgebung (Unabhängigkeit von Plattform, Sprache, etc.)
	Installierbarkeit	Leichte Installation in einer neuen Umgebung
	Konformität	Einhaltung internationaler Standards und Unternehmensrichtlinien
	Austauschbarkeit	Einzelne Services oder gesamte Anwendung leicht ersetzbar durch lose Kopplung und Modularisierung
Quelle: Eigene Tabelle		

3.2.3 Generierung von Testfällen

Um die Anforderungen noch weiter zu konkretisieren werden Testfälle gebildet. Sie sollen im Sinne der Reengineeringmethoden Redocumentation und Design Recovery konkrete Eigenschaften und die Unterschiede zwischen den Forschungsanwendungen herausstellen. Durch die Durchführung der Anwendungstests im Blackbox-Verfahren besteht die Herausforderung bei dieser Analyse darin, dass der Zugriff auf den Quellcode nicht erfolgt. Es wird nur das nach außen beobachtete Verhalten ermittelt, um daraus Rückschlüsse auf interne Abläufe und unbekannte Anforderungen ziehen zu können. Auf diese Weise kann auch von vornherein ausgesagt werden, dass die zu ermittelnden Fachkonzepte von der Implementierungsebene entkoppelt und somit ausreichend abstrakt erfasst werden.
Eine mögliche Vorgehensweise zur Ermittlung geeigneter Testfälle bietet die Betrachtung von Experimenten, die bisher in einer der beiden Anwendungen durchgeführt worden sind. Dabei werden der Workflow, die vorgenommene Konfiguration sowie Eingabedaten und Ergebnisse mit der Absicht betrachtet, diesen Prozess in der anderen Anwendung nachzustellen und zu vergleichbaren Ergebnissen zu gelangen. Unterscheiden sich zum Beispiel berechnete Analysegrößen, so muss geklärt werden, woraus diese Unterschiede resultieren.

3.3 Analyse der Anwendungen

Zum Verständnis der Anwendungen ist ein Studium aller verfügbaren Dokumentationen und Hilfestellungen oder Tutorials sehr nützlich. Erste Erkenntnisse über den Aufbau und die Funktionsweise können bereits im Sinne der Redocumentation gesammelt und eingeordnet werden. Die konkrete Analyse der Anwendungen folgt darauf.
Anhand der unterschiedlichen Experimentdurchführungen werden hauptsächlich der Aufbau, die Abbildung des Forschungsworkflows und wesentliche Besonderheiten untersucht. Interessant sind dabei die speziellen Unterschiede zwischen beiden Anwendungen, die durch die Untersuchung und den Vergleich der Testreihen erfasst werden sollen. Aus den Untersuchungsergebnissen wird durch Abstraktion das Fachkonzept für die jeweilige Anwendung gebildet.
Das Fachkonzept ist dabei die Abstraktion aus den in der Untersuchung ermittelten Funktionen, welche durch Design Recovery und Reducumentation bestimmt wurden. Zusätzlich werden hierbei auch die nicht-funktionalen Eigenschaften einbezogen, obwohl diese nicht in die Anforderungen an die neu zu entwickelnde Anwendung einfließen. Sie könnten je-

doch Entscheidungshilfen beim direkten Vergleich von Funktionen der beiden Anwendungen bieten und sind somit für die Handlungsempfehlung von Bedeutung.
Für das Fachkonzept der beiden zu untersuchenden Anwendungen sollten Kategorien nach den Hauptfunktionen erstellt werden. Anhand dieser Kategorien sind die Anwendungen später leichter zu vergleichen.

3.3.1 Analyse der Anwendung A

Tabelle 4: Beispiel für ein abstraktes Anforderungskonzept von Anwendung A	
Auswertung	Berechnung der Ergebnisskennzahlen nach Methode X
	Darstellung als Wirkungskurve mit farblicher Kennzeichnung markanter Punkte
	Optionen zur Erstellung von Zusammenfassungen und Übersichten
Automatisierung	…
Bearbeitung	…
Datenbankverbindung	..
Menü	…
Reports	…
Rohdaten	…
Templates	…
Quelle: Eigene Tabelle	

Die in Tabelle 4 dargestellten funktionalen Anforderungen sind das Resultat der Abstraktion der untersuchten Eigenschaften von Anwendung A. Die wesentlichen Funktionen sind zusammengefasst und implementierungsfern nach Kategorien aufgestellt.
Bei der Konsolidierung muss gegebenenfalls entschieden werden, welche Funktionen aus einer der beiden Anwendungen besser in die neue Anwendungen zu überführen sind. Um diese Entscheidung zu erleichtern, werden in diesem Fachkonzept die nicht-funktionalen Anforderungen nach der ISO 9126 festgehalten, sofern eine Untersuchung unter den gegebenen Umständen möglich ist.
Aus Eigenschaften wie beispielsweise langer Ladezeit bei umfangreicheren Auswertungsworkbooks lässt sich schließen, dass die Darstellungsform unter Umständen durch eine effizientere Form ersetzt werden kann.

3.3.2 Analyse der Anwendung B

Das Fachkonzept für die Anwendung B wird durch die Abstraktion der Untersuchungsergebnisse gebildet. Dabei wird so vorgegangen, dass die zuvor beschriebenen Eigenschaften stark verallgemeinert werden. Sie dürfen keine Implementierungsvorgaben enthalten, müssen aber trotzdem spezifisch genug sein, um den Spielraum für abweichende Interpretationen so gering wie möglich zu belassen.
Zentrale Anforderung an Anwendung B ist die Berechnung der Ergebniskennzahlen mit der Erstellung der Analysereports in der besonderen Kurvendarstellung. Auch das Prinzip des Reviews durch einen Verantwortlichen lässt sich als Besonderheit hervorheben und geht als Element der Qualitätssicherung in die fachliche Anforderung mit ein. Ebenso besonders ist die Log-Funktion, welche die Arbeitsprozesse transparent und nachvollziehbar dokumentiert. Beispielhaft ermittelte funktionale Anforderungen nach den Vergleichskategorien sind der Tabelle 5 zu entnehmen.

Tabelle 5: Beispiel für ein abstraktes Anforderungskonzept von Anwendung B	
Auswertung	Berechnung der Ergebniskennzahlen nach Methode Y
	Darstellung aufsteigender Wirkungskurven mit Kennzeichnung markanter Punkte
Automatisierung	…
…	
	Quelle: Eigene Tabelle.

3.4 Implikation

Aufbauend auf den Ergebnissen der Analyse der Anwendungen A und B folgt die Auswertung der Untersuchung. Die beiden Anforderungskonzepte werden zusammengefasst, um den Schritt der Konsolidierung zu realisieren. Danach erfolgt eine Gewichtung und Bewertung der ermittelten Anforderungen, aus denen sich konkrete Handlungsempfehlungen für das weitere Vorgehen ableiten lassen.

3.4.1 Konsolidiertes Fachkonzept

Wenn bei genauerer Betrachtung und einem Vergleich der Tabellen 5 und 6 bereits einige Gemeinsamkeiten der beiden Anwendungen A und B deutlich werden, könnte dies erste Hinweise auf leicht zu konsolidierende Funktionen geben. Generell lässt sich sagen, dass die jeweiligen Vorzüge jeder Applikation möglichst vollzählig in die Konsolidierung einfließen sollten Als Resultat sollte man eine präzise arbeitende Anwendung erhalten. die

vielen Einstellmöglichkeiten bietet, einheitliche Ergebnisse liefert und viele durch einige Optionen flexibel anwendbar ist. Die funktionalen Eigenschaften der konsolidierten Anwendung sind beispielhaft in Tabelle 6 zusammengefasst.

Tabelle 6: Beispiel für ein konsolidiertes Anforderungskonzept für eine Analyseanwendung (Anforderungen, die nur aus Anwendung A stammen, sind grün und die aus B blau markiert)

Auswertung	Berechnung der Ergebnisskennzahlen nach Methode Y
	Darstellung aufsteigender Wirkungskurve mit farblicher Kennzeichnung markanter Punkte
	Optionen zur Erstellung von Zusammenfassungen und Übersichten
…	…

Quelle: Eigene Tabelle.

3.4.2 Bewertung der Anforderungen

Grundsätzlich ist zu sagen, dass weder Anwendung A noch B bevorzugt behandelt werden. Die Anforderungen sollten gleichermaßen in die Konsolidierung einfließen, um auf diesem Abstraktionsniveau viele Nutzerbedürfnisse beider Seiten zu berücksichtigen.
An dieser Stelle entsteht eine Übersicht, welche die zuvor untersuchten Anforderungskategorien geordnet nach ihrer Priorität im Bezug auf den Entwurf der neuen Anwendung enthält, wobei keine der ermittelten Anforderungen als vernachlässigbar gelten soll.

3.4.3 Handlungsempfehlung

Unter Berücksichtigung der gewonnenen Erkenntnisse wird empfohlen, die ermittelten Basisanforderungen an die neu zu entwickelnde Anwendung zusammen mit den Kunden zu spezifizieren. Auf diese Weise kann sichergestellt werden, dass die fachlichen Bedürfnisse erfüllt und mit den technischen Möglichkeiten nach Kundenwunsch realisiert werden.
Aus der vorgenommenen Untersuchung sollten bereits die wesentlichen Funktionen, welche in der neuen Anwendung implementiert werden müssen, hervorgegangen sein. Durch die Spezifizierung der Erkenntnisse sind beim Forward Engineering konkrete Implementierungsentwürfe für die einzelnen Funktionsbausteine zu ermitteln.
Die Handlungsempfehlung könnte anhand der Untersuchungsergebnisse ebenfalls Aussagen über die Funktionen treffen, welche sich besonders als modulare Service-Komponenten realisieren lassen. Zusätzliche Schritte zum Weiteren Vorgehen können ebenfalls in die Handlungsempfehlung einfließen.

4. Fazit

Die Betrachtung hat gezeigt, dass eine Konsolidierung von Altanwendungen auf Anforderungsebene praktikabel und sinnvoll erscheint. Die Bedeutung des Reengineering durch die Erhaltung bewährter Praktiken in Branchen wie der Pharmaforschung soll dabei deutlich werden. Das umgesetzte Wissen bestehender Altanwendungen kann mittels Reengineeringmethoden aus den Altanwendungen extrahiert werden. Durch die Ermittlung und Übertragung funktionaler Anforderungen wird abstrakt betrachtet nur die Struktur der Anwendung an den technischen Fortschritt angepasst. Viele Ähnlichkeiten zwischen betrachteten Anwendungen erleichtern eine Konsolidierung auf Anforderungsebene. Eine Zusammenführung der Eigenschaften kann sich sogar im Sinne der Optimierung lohnen, da die konsolidierte Anwendung die besten Eigenschaften der Vorgänger vereint, wodurch sich beispielsweise die Leistungsfähigkeit erhöhen kann.

Ein großer Vorteil der Verfahrensweise ist, dass keine geschaffenen Werte verloren gehen. Bis zur Fertigstellung der neu entwickelten Anwendung muss nicht einmal in das arbeitende System eingegriffen werden. Daher kann das Risiko von totalen Ausfällen durch die Überholung von Legacy-Anwendungen praktisch ausgeschaltet werden. Die Pharmaforschung des Unternehmens erfährt somit eine sanfte Erneuerung der Analysesoftware, die mit dem Forschritt mithalten kann und auch auf langfristige Sicht durch geringere Kosten für Wartung und Überholung einen Teil zur Wettbewerbsfähigkeit beiträgt.

Literaturverzeichnis

Albers, Sönke u.a. (2006):
Methodik der empirischen Forschung, Wiesbaden, 1. Auflage, 2006.

Bauer, Andreas und Günzel, Holger (Hrsg.)(2008):
Data-Warehouse-Systeme: Architektur, Entwicklung, Anwendung, Heidelberg, 3., überarbeitete und aktualisierte Auflage, 2008.

Berthold, Michael und Hand, David J. (2003):
Intelligent Data Analysis – An Introduction, Berlin, Heidelberg, 2. Edition, 2003.

Biggerstaff, Ted J. (1989):
Design Recovery for Maintenance and Reuse, Los Alamitos, 1989.

Fischer, Joachim u.a. (2002):
Bausteine der Wirtschaftsinformatik. Grundlagen, Anwendungen, PC-Praxis, Berlin, 3., überarbeitete Auflage, 2002.

Fowler, Martin (2005):
Refactoring, oder: wie Sie das Design vorhandener Software verbessern, München, 2005.

Kerievsky, Joshua (2005):
Refactoring to Patterns, Boston u.a., 3rd Printing, 2005.

Kockläuner, Gerhard Prof. Dr. (2000):
Multivariate Datenanalyse – Am Beispiel des statistischen Programmpakets SPSS, Braunschweig, Wiesbaden, 1. Auflage, 2000.

Masak, Dieter (2005):
Legacy-Software: Das lange Leben der Altsysteme, Berlin, Heidelberg, 2005.

Mens, Tom und Demeyer, Serge (Eds.) (2008):
Software Evolution, Berlin, Heidelberg, 2008.

Mertens, Peter u.a. (Hrsg.) (2001):
Lexikon der Wirtschaftsinformatik: Software Reengineering und Software Reverse Engineering, Berlin u.a., 4., vollständig neu bearbeitete und erweiterte Auflage, 2001.

Mertens, Peter und Wieczorrek, Hans Wilhelm (2000):
Data X Strategien – Data Warehouse, Data Mining und operationale Systeme für die Praxis, Berlin, Heidelberg , 2000.

Roock, Stefan und Lippert, Martin (2004):
Refactorings in großen Softwareprojekten – Komplexe Restrukturierungen erfolgreich durchführen, Heidelberg, 1. Auflage, 2004.

Runkler, Thomas A. (2000):
Information Mining – Methoden, Algorithmen und Anwendungen intelligenter Datenanalyse, Braunschweig, Wiesbaden, 2000.

Schmietendorf, Andreas und Dumke, Reiner (Hrsg.)(2006):
Tagungsband 1. Workshop Bewertungsaspekte serviceorientierter Architekturen (BSOA06), Magdeburg, 2006.

Scholz, Roland W. und Tietje, Olaf (2002):
Embedded Case Study Methods: Integrating Quantitative and Qualitative Knowledge, Thousand Oaks (USA) u.a., 2002.

Seacord, Robert S. u.a. (2003):
Modernizing Legacy Systems: Software Technologies, Engineering, Amsterdam, 2003.

Simon, Frank u.a. (2006):
Code-Quality-Management – Technische Qualität industrieller Softwaresysteme transparent und vergleichbar gemacht, Heidelberg, 2006.

Sommerville, Ian (2007):
Software Engineering, München, 8., aktualisierte Auflage, 2007.

Yin, Robert K. (2003):
Case Study Research: Design and Methods, Thousand Oaks (USA), 3rd Edition, 2003.

Internetverzeichnis

Bashir, Faisal Bin (1998):

a) Design Recovery, 29.03.1998, abgerufen am 26.06.2008, http://www.suite101.com/article.cfm/reengineering/6458.
b) Redocumentation, 21.03.1998, abgerufen am 26.06.2008, http://www.suite101.com/article.cfm/reengineering/6263.

Bauer, Markus u.a. (2003):
App2Web: Redokumentation – Altsystem-Analyse, 15.01.2003, abgerufen am 26.06.2008, http://app2web.fzi.de/ereignisse/abschlussseminar/Altsystem-Analyse_Bauer.ppt.

Chikofsky, Elliot J. und Cross, James H. (1990):
Reverse Engineering and Design Recovery; A Taxonomy, 1990, abgerufen am 25.06.2008, http://labs.cs.utt.ro/labs/acs/html/resources/ReengineeringTaxonomy.pdf.

George, Thomas und Westermann, Gerd (2006):
Model Driven Architecture in einer service-orientierten Anwendungslandschaft, Java-Spektrum 1/2006, abgerufen am 04.06.2008
http://www.sdm.de/web4archiv/objects/download/pdf/1/sdm_pub_js_westerman.pdf.

GraphPad Software Inc. (2007):
Dose Response Curves, 2007, abgerufen am 06.06.2008
http://graphpad.com/prism/learn/Dose-response%20curves.pdf

International Union of Pharmacology (2008):
TABLE 3 Experimental measures of drug action: agonists, 2008 abgerufen am 15.06.2008, http://pharmrev.aspetjournals.org/cgi/content/full/55/4/597/TBL3.

IT Wissen Online Lexikon (2008):

a) 4th generation language – 4GL (Viertgenerationssprache), 23.06.2008, abgerufen am 23.06.2008, http://www.itwissen.info/definition/lexikon/4th-generation-language-4GL-Viertgenerationssprache.html.
b) Objektorientierte Programmierung, 25.07.2008, abgerufen am 25.07.2008, http://www.itwissen.info/definition/lexikon/Objektorientierte-Programmierung-OOP-object-oriented-programming.html.

Johnson, Ralph E. (2005):
Reverse Engineering and Software Archaeology, Oktober 2005, abgerufen am 25.06.2008, https://www.softwaretechnews.com/stn_view.php?stn_id=1&article_id=29.

Kanellopoulos, Yiannis (2006):
Employing ISO-9126 Quality Standard and Data Mining Techniques to Support Migration of Legacy Systems to Service Oriented Architecture, 2006, abgerufen am 25.06.2008, http://www.code4thought.org/wp-content/uploads/2007/06/sosornet_paper.pdf.

Kühnel, Andreas (2003):
.NET Konzepte, Vor- und Nachteile, Januar 2003, abgerufen am 23.06.2008, http://www.contentmanager.de/magazin/artikel_274_net_konzepte.html.

Kurlbaum, Jörg (2004):
Nicht-lineare Ausgleichsrechnung, 2004, abgerufen am 16.06.2008, http://www.informatik.uni-bremen.de/~ufrese/teaching/vui04/kurlbaum_vui_04.pdf.

Lewis, Grace u.a. (2005):
a) SMART: The Service-Oriented Migration and Reuse Technique, September 2005, abgerufen am 25.06.2008, http://www.sei.cmu.edu/pub/documents/05.reports/pdf/05tn029.pdf.
b) Migration of Legacy Components to Service-Oriented Architectures, Oktober 2005, abgerufen am 25.06.2008, https://www.softwaretechnews.com/stn_view.php?stn_id=1&article_id=27.

Lourakis, Manolis I.A. (2005):
A brief description of the Levenberg-Marquardt-Algorithm implemented by levmar, 11.02.2005, abgerufen am 16.06.2008, http://www.ics.forth.gr/~lourakis/levmar/levmar.pdf.

Microsoft Corporation (2008):
a) Erstellen und Weitergeben von XML Web Services mithilfe von Visual Studio .NET und Office XP, 2008, abgerufen am 24.07.2008, http://www.microsoft.com/germany/msdn/library/data/xml/ErstellenUndWeitergebenVonXMLWebServicesMithilfeVonVisualStudioNETUndOfficeXP.mspx?mfr=true.
b) Visual Basic,2008, abgerufen am 24.07.2008, http://msdn.microsoft.com/en-us/isv/bb190538.aspx.
c) Visual Basic for Applications, 2008, abgerufen am 24.07.2008, http://msdn.microsoft.com/en-us/isv/bb190538.aspx.
d) Visual-Studio Makros, 2008, abgerufen am 24.07.2008, http://msdn.microsoft.com/de-de/library/b4c73967(VS.80).aspx.

PatentStorm LLC (1995):
US Patent 5446035 20-methyl-substituted vitamin D derivatives, 1995, abgerufen am 20.06.2008, http://www.patentstorm.us/patents/5446035/description.html.

Richter, Jan-Peter Dr. u.a. (2005):
Informatiklexikon der Gesellschaft für Informatik: Serviceorientierte Architektur, 2005, abgerufen am 04.06.2008, http://www.gi-ev.de/service/informatiklexikon/informatiklexikon-detailansicht/meldung/118/.

Schmietendorf, Andreas Prof. Dr. (2005):
Grundlagen der Programmierung - Einführung in die strukturierte Programmierung, 2005, abgerufen am 15.07.2008, http://ivs.cs.uni-magdeburg.de/~schmiete/lehre/vorlesung/paper/ws05/Prg_part4.pdf.

Sneed, Harry M. (2006):
Migration in eine Service-orientierte Architektur, 2006, abgerufen am 25.06.2008, http://pi.informatik.uni-siegen.de/stt/27_2/01_Fachgruppenberichte/WSR/06sneed.pdf.

Straetz, Katja T. (2000):
Redokumentation im Software Reengineering, 06.12.2000, abgerufen am 26.06.2008, http://www.uni-koblenz.de/~ist/lehre/WS0001/Projekt/Ausarbeitungen/straetz.pdf.

Sun Microsystems Inc. (2005):
Java Programming Language, 2005, abgerufen am 25.07.2008, http://java.sun.com/javase/6/docs/technotes/guides/language/.

Sybase Inc. (2008):
a) PowerBuilder – A 4GL rapid application development tool, 2008, abgerufen am 23.06.2008, http://www.sybase.com/products/development/powerbuilder.
b) Product Datasheet Powerbuilder 11.2, 2008, abgerufen am 23.06.2008, http://www.sybase.com/files/Data_Sheets/PowerBuilder-11.2-042108-ds.pdf.

Tamm, Gerrit Dr. und Günther, Oliver Prof. (2005):
Webbasierte Dienste – Technologien, Märkte und Geschäftsmodelle, Heidelberg, 2005 e-Book, abgerufen am 26.05.2008, http://www.springerlink.com/cotent/p80038/?p=02771 f82cc7541eead68c8cf2fcf5c8e&pi=0.

World Wide Web Consortium W3C (2004):
Web Services Glossary, Februar 2004, abgerufen am 01.07.2008, http://www.w3.org/TR/ws-gloss/.