Jung, Reinhard

Working Paper

Wirtschaftlichkeitsfaktoren beim integrationsorientierten Reengineering: Verteilungsarchitektur und Integrationsschritte aus ökonomischer Sicht

Arbeitsberichte des Instituts für Wirtschaftsinformatik, No. 16

Provided in Cooperation with:
University of Münster, Department of Information Systems

Suggested Citation: Jung, Reinhard (1993) : Wirtschaftlichkeitsfaktoren beim integrationsorientierten Reengineering: Verteilungsarchitektur und Integrationsschritte aus ökonomischer Sicht, Arbeitsberichte des Instituts für Wirtschaftsinformatik, No. 16, Westfälische Wilhelms-Universität Münster, Institut für Wirtschaftsinformatik, Münster

This Version is available at:
http://hdl.handle.net/10419/59373

Arbeitsberichte des Instituts für Wirtschaftsinformatik
Herausgeber: Prof. Dr. J. Becker, Prof. Dr. H. L. Grob, Prof. Dr. K. Kurbel,
Prof. Dr. U. Müller-Funk, Prof. Dr. R. Unland

Arbeitsbericht Nr. 16

Wirtschaftlichkeitsfaktoren beim integrationsorientierten Reengineering

Verteilungsarchitektur und Integrationsschritte aus ökonomischer Sicht

Reinhard Jung

Institut für Wirtschaftsinformatik der Westfälischen Wilhelms-Universität Münster,
Grevener Str. 91, 48159 Münster, Tel. (0251) 83-9750, Fax (0251) 83-9754
Juli 1993

Inhalt

Zusammenfassung

Im Rahmen des integrationsorientierten Reengineering sollen in Betrieb befindliche und neu zu entwickelnde Softwaresysteme in *ein* integriertes verteiltes Anwendungssystem (IVAS) integriert werden. Zu diesem Zweck muß vor Beginn des Integrationsprozesses die Verteilungsarchitektur festgelegt werden. Anschließend können durch einzelne nacheinander erfolgende Integrationsschritte vorhandene bzw. neu entwickelte Softwaresysteme in das IVAS integriert werden. Der vorliegende Arbeitsbericht beschäftigt sich zum einen mit den Faktoren, die aus ökonomischer Sicht einen Einfluß auf die Wahl der Verteilungsarchitektur haben. Zum anderen wird untersucht, welche Faktoren einen Einfluß auf die Entscheidung haben, ob ein Altsystem einem Reengineering-Prozeß unterzogen oder ersetzt werden soll.

1 Einleitung

1.1 Problemstellung

Bei den Entscheidungen, die im Rahmen der Integration von Software-Altsystemen anstehen, spielen neben den softwaretechnischen vor allem ökonomische Aspekte eine herausragende Rolle. Für den Entscheidungsträger stellt sich insbesondere die Frage, welche Architektur als Integrationsbasis zu wählen ist und ob bezüglich eines Software-Altsystems eine Ersetzung (Neuentwicklung, Kauf) oder Reengineering sinnvoller ist.

Bei der Beantwortung muß beachtet werden, daß neben Faktoren, die durch Entscheidungen beeinflußt werden können (z.B. Laufzeitverhalten des Gesamtsystems), auch Faktoren zu berücksichtigen sind, die durch unternehmensspezifische Gegebenheiten vorgegeben sind (Kontextfaktoren).

1.2 Begriffsbestimmungen

Im vorliegenden Arbeitsbericht werden Faktoren, denen direkte ökonomische Konsequenzen zuzuordnen sind, als *Wirtschaftlichkeitsfaktoren* bezeichnet. Alle weiteren Faktoren gehen mittelbar (als Determinanten von Wirtschaftlichkeitsfaktoren) in die Untersuchung ein.

Die Untersuchung der Wirtschaftlichkeitsfaktoren, die für oder gegen das integrationsorientierte Reengineering sprechen[1)], führt quasi automatisch in den Bereich der Wartung. Die Softwaremerkmale, die für eine gute Wartbarkeit sprechen, sind - zumindest zum Teil - auch auf das Reengineering anwendbar. Um diesen Zusammenhang herstellen zu können, erfolgt zunächst eine Definition des Begriffs *integrationsorientiertes Reengineering* auf Basis der Definitionen der Begriffe *Reengineering* und *Wartung*.

Balzert definiert *Reengineering* wie folgt: "Reengineering (...) umfaßt die Überprüfung und den Umbau des vorhandenen Systems, so daß eine Wiederherstellung in neuer Form erreicht wird. Reengineering schließt *Reverse Engineering* zum Teil ein und wird gefolgt von *Forward Engineering* oder *Restructuring*. Modifikationen bezogen auf neue Anforderungen gegenüber dem Originalsystem können eingeschlossen sein."[2)]

---

1) Vgl. Kapitel 3.
2) Balzert (1992), S. 95. Balzert gibt ebenfalls Definitionen der Begriffe *Reverse Engineering*, *Forward Engineering* und *Restructuring*.

Für den Begriff (Software-)*Wartung* finden sich unterschiedliche Definitionen, die aber grundsätzlich gleichbedeutend sind mit: "Die Änderung, Erweiterung oder Korrektur eines Programms bezeichnet man als Wartung."[3)]

Die Definition von Balzert zeigt explizit, daß beim *Reengineering* der (Neu-)Implementierung (Forward engineering) die "Anhebung" des Quelltextes auf ein abstrakteres Niveau (Reverse engineering) vorausgeht. Diesen umfassenderen Charakter besitzt die Wartung nicht[4)]. Abbildung 1 verdeutlicht die Zielrichtung des Reengineering schematisch.

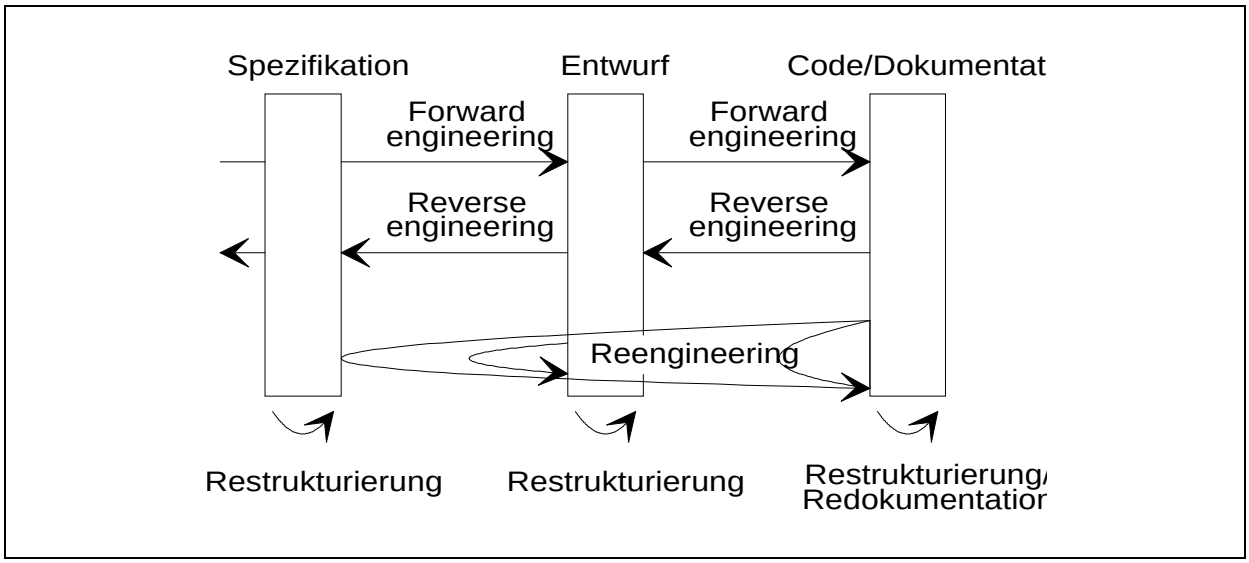

Abb. 1: Zusammenhang zwischen Reengineering, Reverse engineering und Forward engineering[5)]

Definition: (integrationsorientiertes Reengineering)

Integrationsorientiertes Reengineering ist die methodische Analyse eines bereits vorhandenen Softwaresystems, die eine Anhebung der vorliegenden Dokumente (Code/Dokumentation, Entwurf, Spezifikation) auf ein abstrakteres Niveau (Reverse engineering) einschließen kann, gefolgt von der ebenfalls methodengestützten Modifikation (Forward engineering), mit dem Ziel, das System in eine Anwendungsarchitektur einbinden zu können, die insbesondere die Daten der Teilsysteme integriert.

---

3) Kurbel (1985), S. 1; weitere Definitionen finden sich z.B. bei Pomberger (1990), S. 221, oder Sneed (1983), S. 172.

4) Boehm schränkt die Wartung sogar quantitativ ein; dabei werden Richtgrößen wie "less than 50% new code" angegeben (vgl. Boehm (1981), S. 54).

5) Mit Änderungen entnommen aus Chikofsky, Cross (1990), S. 14.

Um die für eine Integration relevanten Wirtschaftlichkeitsfaktoren und deren Determinanten herausarbeiten zu können, folgen zunächst Definitionen der Begriffe *integriertes verteiltes Anwendungssystem* (IVAS) und *Verteilungsarchitektur* sowie eine schematische Beschreibung eines Integrationsschritts und der dabei relevanten Objekte (vgl. Abbildung 2).

Definition: (integriertes verteiltes Anwendungssystem, IVAS)

Ein integriertes verteiltes Anwendungssystem (IVAS) ist ein Anwendungssystem, das aus einem oder mehreren bereits miteinander integrierten, eventuell auf verschiedenen Netzknoten angesiedelten Anwendungssystemen besteht und dessen Architektur und Dokumentation darauf ausgerichtet sind, die Integration weiterer Anwendungssysteme zu ermöglichen.

Definition: (Verteilungsarchitektur)

Die Verteilungsarchitektur ist die softwaretechnische Struktur eines integrierten verteilten Anwendungssystems. Sie legt insbesondere fest, wie der Zugriff bereits integrierter Teilsysteme auf Daten des IVAS erfolgt und wie die Aktualisierung redundanter Daten bzw. wie die Einhaltung von Integritätsbedingungen gewährleistet wird.

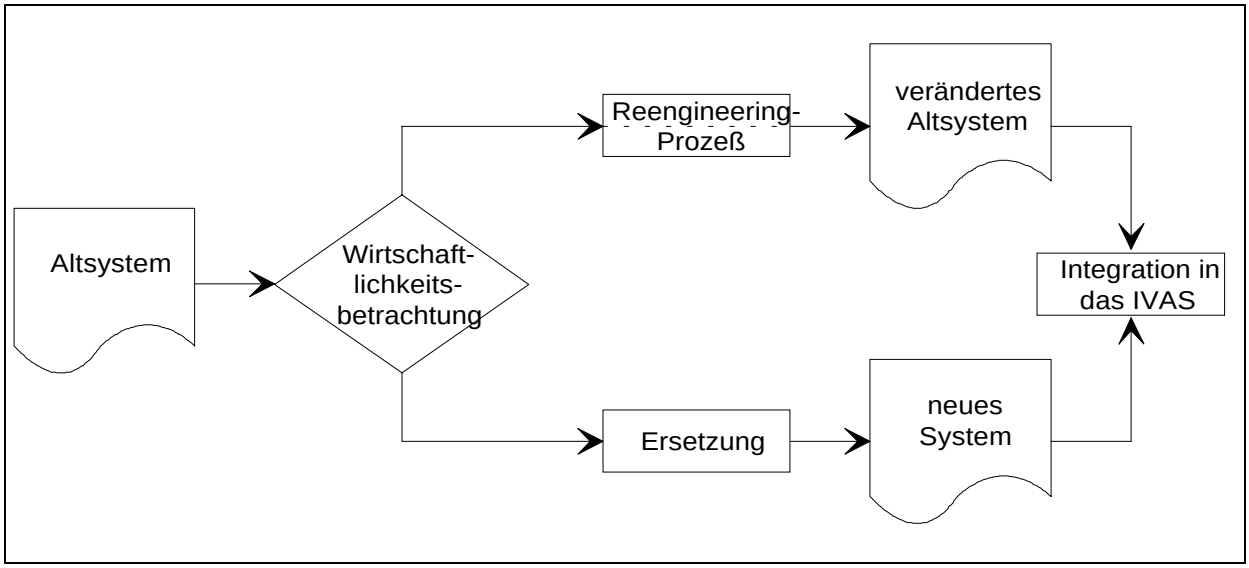

Abb. 2: Schematische Darstellung eines Integrationsschritts[6)]

Ein Altsystem wird entweder einem Reengineering-Prozeß unterzogen, um in das IVAS integriert werden zu können, oder es wird durch ein neues System ersetzt. In beiden Fällen können neben dem Integrationsziel weitere Ziele, wie z.B. die Portierung in eine andere Umgebung oder

---

6) Sinnbilder in Anlehnung an DIN 66001 (vgl. Deutsches Institut für Normung e.V. (1977)); das Sinnbild für Schriftstücke wird hier für Programme verwendet.

die Verbesserung der Benutzeroberfläche, verfolgt werden. Der Aufwand eines (integrationsorientierten) Reengineering-Projekts wird durch die Beschaffenheit des Altsystems, durch die Ziele und Merkmale des Reengineering-Prozesses und durch die Beschaffenheit des IVAS bestimmt.

2 Konfigurierung der Verteilungsarchitektur

2.1 Parameter der Verteilungsarchitektur

Die Konfigurierung einer Verteilungsarchitektur ist von grundlegender Bedeutung für den späteren Integrationsprozeß, da ihre Gestaltung wesentlichen Einfluß auf den Aufwand hat, der bei einzelnen Integrationsschritten zur Einbindung von Anwendungssystemen notwendig ist. Darüber hinaus werden bei der Konfigurierung der Verteilungsarchitektur bedeutende Charakteristika des resultierenden Gesamtsystems festgelegt (Laufzeitverhalten und Ausfallsicherheit des IVAS).

Eine Verteilungsarchitektur läßt sich im wesentlichen durch drei Eigenschaften beschreiben, die im weiteren als Parameter der Verteilungsarchitektur bezeichnet werden[7]:

- Der Parameter *Datenzugriffsmanagement* legt fest, in welcher Art und Weise ein Anwendungssystem jeweils auf bereits in das IVAS integrierte Daten zugreift, die es nicht selbst verwaltet.

- Anhand des Parameters *Integritätsmanagement* wird ersichtlich, welche Instanz für die Integritätssicherung der bereits in das IVAS integrierten Daten zuständig ist.

- Über den Parameter *Aktualisierungsmanagement* wird eingestellt, wie die Aktualisierung redundanter Daten im IVAS erfolgen soll.

In Abbildung 3 sind die möglichen Belegungen der drei Parameter der Verteilungsarchitektur dargestellt; sie sind voneinander unabhängig, so daß sich 24 mögliche Kombination ergeben.

Im folgenden Abschnitt werden die Faktoren, die Auswirkungen auf die Zweckmäßigkeit einer Verteilungsarchitektur haben (Wirtschaftlichkeitsfaktoren der Verteilungsarchitektur), einschließlich ihrer Interdependenzen untersucht.

---

[7] Zu einer detaillierten Beschreibung vgl. Eicker et al. (1993), S. 73 ff.

Parameter Variante	Datenzugriffs-management	Integritäts-management	Aktualisierungs-management
1	alle Anwendungssysteme wenden sich an einen zentralen Daten-Broker	globale und lokale Integritätsregeln*) werden von einer zentralen Instanz überwacht	das System, das ein redundantes Datum verwaltet, meldet die Änderung an einen zentralen Aktualisierungsmanager weiter, der die Aktualisierung weiterer Ausprägungen des Datums übernimmt
2	jedem Anwendungssystem ist ein Daten-Broker zugeordnet, der die Zugriffe dieses Systems durchführt	globale Integritätsregeln werden von einer zentralen Instanz überwacht, lokale Integritätsregeln von dem System, das die Daten verwaltet	das System, das ein redundantes Datum verwaltet, führt die Aktualisierung weiterer Ausprägungen des Datums durch
3	jedes Anwendungssystem wendet sich direkt an das System, das die benötigten Daten verwaltet	globale Integritätsregeln werden von einer zentralen Instanz überwacht, lokale Integritätsregeln von dem System, das die Speicherung/ Änderung ausgelöst hat	---
4	---	globale und lokale Integritätsregeln werden von dem System überwacht, das die Speicherung/Änderung ausgelöst hat	---

*) Lokale Integritätsregeln sind solche Regeln, zu deren Überwachung ein System, das Daten verwaltet, nur auf Daten seiner Datensicht zugreifen muß. Als globale Integritätsregeln werden alle übrigen Regeln bezeichnet, d.h. Regeln, zu deren Überwachung eine anwendungsübergreifende Datensicht notwendig ist.

Abb. 3: Parameter der Verteilungsarchitektur und ihre Varianten[8)]

2.2 Wirtschaftlichkeitsfaktoren der Verteilungsarchitektur

Die Ausgestaltung der einzelnen Parameter der Verteilungsarchitektur beeinflußt das *Laufzeitverhalten*[9)] und die *Ausfallsicherheit* des IVAS sowie den *Integrations- bzw. Änderungsaufwand*, der bei der späteren Einbindung eines Anwendungssystems entsteht. Neben diesen drei Wirtschaftlichkeitsfaktoren ist aber auch die *Einsatzform des IVAS*, die z.B. "dialogorientierte

8) Im folgenden werden die einzelnen Belegungen eines Parameters als Varianten bezeichnet.

9) Das Laufzeitverhalten eines betrieblichen Informationssystems kann je nach Einsatzform ein wichtiger Wirtschaftlichkeitsfaktor sein (vgl. z.B. Kador (1992)); gleiches gilt analog für dialogorientierte Datenbanksysteme (vgl. Ambichl, Heinrich (1992), S. 24).

Nutzung durch Sachbearbeiter in der Kundenbetreuung" sein kann, als Kontextfaktor in die Betrachtung einzubeziehen.

2.2.1 Laufzeitverhalten

Das Laufzeitverhalten des IVAS wird einerseits dadurch festgelegt, wieviele Komponenten (Anwendungssysteme und Komponenten der Verteilungsarchitektur) beim Zugriff auf Daten aktiv werden bzw. wieviele Nachrichten zwischen den Komponenten auszutauschen sind[10]. Andererseits ist die dominierende Determinante des Laufzeitverhaltens (bei schreibenden Zugriffen) der *Aktualisierungsmodus*[11], da unter der Voraussetzung, daß ein Datenobjekt in mehreren redundanten Ausprägungen existiert, zusätzlich zu dem primären schreibenden Datenzugriff Aktualisierungsaufwand in Form weiterer Datenzugriffe erforderlich ist.

Bei *lesenden Zugriffen* auf Daten sind die Belegungen der Parameter *Integritätsmanagement* und *Aktualisierungsmanagement* nicht von Bedeutung, da Integritätsprüfungen und Aktualisierungen nur bei schreibenden Zugriffen erforderlich sein können.

Die Anzahl der Nachrichten ist bei den Varianten 1 und 2 des Parameters *Datenzugriffsmanagement* identisch; die erste Anfrage wird an den zuständigen Daten-Broker abgeschickt, der wiederum eine Nachricht an das für dieses Datum zuständige Datenverwaltungssystem abschickt. Die Rückmeldung an den Daten-Broker und von dort an das anfragende Anwendungssystem schließen den Zugriff ab. Bei den Varianten 1 und 2 sind für den lesenden Datenzugriff eines Anwendungssystems also vier Nachrichten erforderlich.

Bei Variante 3 des Parameters *Datenzugriffsmanagement* sind hingegen nur zwei Nachrichten notwendig, da der Daten-Broker als zwischengeschaltete Instanz entfällt. Das anfragende System wendet sich direkt an das datenverwaltende System und erhält von diesem das gewünschte Datum.

Bei *schreibenden Zugriffen* sind grundsätzlich mehr Nachrichten auszutauschen als bei lesenden Zugriffen. Je nach betroffenem Datum müssen zusätzlich zum eigentlichen Zugriff Integritätsprüfungen vorgenommen oder Aktualisierungen redundanter Ausprägungen des Datums

---

[10]) Eine *Nachricht* besteht aus einem Transaktionscode und zugehörigen Daten (vgl. dazu auch Rahm (1988), S. 6 ff.). Der Umfang der Nachrichten kann ebenfalls Auswirkungen auf das Laufzeitverhalten haben; eine Abschätzung dieser Auswirkungen ist aber erst möglich, wenn die technische Realisierung der Verteilungsarchitektur spezifiziert ist.

[11]) Vgl. Eicker et al. (1993), S. 76; Wedekind spricht in diesem Zusammenhang auch von der *Aktualisierungsform* (vgl. Wedekind (1988), S. 269).

durchgeführt werden. Die Anzahl der notwendigen Nachrichten läßt sich ermitteln, indem die drei Parameter zunächst unabhängig voneinander betrachtet werden.

Die durch die Belegung des Parameters *Datenzugriffsmanagement* "verursachten" Zugriffe ergeben sich analog aus den Ausführungen über lesende Zugriffe. Wenn ein Daten-Broker für den Zugriff zuständig ist (Varianten 1 und 2), sind vier Nachrichten notwendig, bei direktem Zugriff (Variante 3) hingegen nur zwei Nachrichten.

Bei den Varianten 1 bis 3 des Parameters *Integritätsmanagement* muß vom datenverwaltenden System eine Nachricht - in Form einer Anfrage nach der Integritätsverträglichkeit der Schreiboperation - an eine zentrale Instanz abgesetzt werden, die von dieser durch eine weitere Nachricht beantwortet wird. Bezogen auf die Anzahl der Nachrichten ist es dabei unerheblich, ob die zentrale Instanz nur globale oder zusätzlich auch lokale Regeln überwacht. Bei Variante 4 ist hingegen keine Nachricht notwendig, da das datenverwaltende System die Einhaltung sowohl globaler als auch lokaler Regeln selbst überwacht.

Der dritte Parameter der Verteilungsarchitektur, das *Aktualisierungsmanagement*, erfordert nur für Variante 1 eine zusätzliche Nachricht, mit der dem zentralen Aktualisierungsmanager die Änderung eines Datum mitgeteilt wird. Bei Variante 2 ist keine Nachricht notwendig, da das datenverwaltende System die eventuell notwendigen Aktualisierungen selbst auslöst.

Die Anzahl der auszutauschenden Nachrichten ist - differenziert nach Parametern - in Abbildung 4 zusammengefaßt.

Parameter / Variante	Datenzugriffs-management (lesend/schreibend)	Integritäts-management (schreibend)	Aktualisierungs-management (schreibend)
1	4	2	1
2	4	2	0
3	2	2	-
4	-	0	-

Abb. 4: Anzahl der auszutauschenden Nachrichten bei lesenden bzw. schreibenden Zugriffen

Wie bereits ausgeführt, ist der Aktualisierungsmodus bei schreibenden Zugriffen die wesentliche Determinante des Laufzeitverhaltens. Je mehr redundante Ausprägungen zu einzelnen Datenobjekten vorhanden sind, desto größer wird der Einfluß des Aktualisierungsmodus. Im Be-

reich der *replizierten Datenbanken* existieren verschiedene Ansätze zur Lösung der Update-Problematik[12]:

- *Write-All-Read-Any-Ansatz*: Dieser Ansatz sieht vor, daß bei einem schreibenden Zugriff zunächst alle Replikate gesperrt und erst dann aktualisiert werden.

- *Primary-Copy-Verfahren*: Bei diesem Verfahren wird nur eine sogenannte Primärkopie aktualisiert. Alle weiteren Replikate können später aktualisiert werden.

- *Voting-Verfahren (Quorum Consensus)*: Bei einem schreibenden Zugriff wird zunächst die Mehrheit der Replikate gesperrt und aktualisiert. Die aktualisierten Replikate werden dabei mit einem Änderungszeitstempel versehen. Beim späteren Lesen wird ebenfalls die Mehrheit der Replikate gesperrt und anhand des Änderungszeitstempels das aktuellste Replikat ausgewählt. Dadurch ist gewährleistet, daß immer eines der zum Lesen gesperrten Replikate auf dem aktuellsten Stand ist.

Bei der Auswahl des Aktualisierungsmodus ist zu beachten, daß bestimmte Verfahren schreibende Zugriffe mit nachfolgender Aktualisierung stark diskriminieren (z.B. der Write-All-Read-Any-Ansatz), andere wiederum eher lesende Zugriffe benachteiligen (z.B. das Primary-Copy-Verfahren).

2.2.2 Ausfallsicherheit

Die Ausfallsicherheit einer Verteilungsarchitektur kann - bezogen auf die Parameter - daran gemessen werden, welche "Dienstleistungen" der Verteilungsarchitektur (z.B. lesender Datenzugriff) nach dem Ausfall einer beteiligten Komponente (z.B. eines Daten-Brokers) noch möglich sind.

Die Ausfallsicherheit sowohl bei lesenden als auch bei schreibenden Zugriffen nimmt, bezogen auf den Parameter *Datenzugriffsmanagement*, von Variante 1 nach Variante 3 zu. Der Ausfall des zentralen Daten-Brokers bei Variante 1 des Parameters Datenzugriffsmanagement stellt den *Worst case* bezüglich der Ausfallsicherheit dar. In diesem Fall sind weder schreibende noch lesende Datenzugriffe möglich. Dadurch ist prinzipiell das gesamte IVAS nicht mehr funktionstüchtig. Wenn hingegen einer von mehreren Daten-Brokern ausfällt (Variante 2), sind nur die Anwendungssysteme, für die dieser Daten-Broker zuständig ist, bezüglich ihrer Datenzugriffe lahmgelegt. Bei Variante 3 (direkter Zugriff) schließlich hat der Ausfall eines datenverwaltenden

---

[12] Eine Übersicht findet findet sich in Rahm (1988), S. 56 ff.

Systems weniger weitreichende Auswirkungen, da nur die Zugriffe auf eine kleine Untermenge der im IVAS vorhandenen Datenobjekte blockiert wird.

Die Ausfallsicherheit wird nur durch die Varianten 1 und 2 des Parameters *Integritätsmanagement* beeinflußt, da in diesen beiden Fällen die Integritätsprüfung vom Datenzugriff abgekoppelt ist. Wenn die zentrale Instanz zur Integritätssicherung ausfällt, können bei Variante 1 keine schreibenden Zugriffe ausgeführt werden und bei Variante 2 nur Zugriffe auf Datenobjekte, die in keiner globalen Integritätsregel enthalten sind.

Auch bei dem Parameter *Aktualisierungsmanagement* ist primär nur eine Ausprägung für die Ausfallsicherheit relevant, nämlich Variante 1. Wenn der zentrale Aktualisierungsmanager ausfällt, sind keine schreibenden Zugriffe auf Datenobjekte möglich, zu denen weitere redundante Datenobjekte existieren. Die Ausfallsicherheit wird allerdings, wie auch das Laufzeitverhalten, stark von dem gewählten Aktualisierungsmodus beeinflußt. Die Verwendung des Write-All-Read-Any-Ansatzes führt beispielsweise schon beim Ausfall *eines* datenverwaltenden Systems zur Blockierung *aller* schreibenden Zugriffe auf Datenobjekte, die eine redundante Ausprägung im "Verwaltungsbereich" des ausgefallenen Systems haben.

2.2.3 Integrations- und Änderungsaufwand

Neben den Merkmalen des zu integrierenden Anwendungssystems[13)] hat auch die Gestaltung der Verteilungarchitektur einen Einfluß auf den Aufwand, der bei der späteren Integration von einzelnen Anwendungssystemen anfällt. Der Teil des Aufwands, der durch eine bestimmte Verteilungsarchitektur verursacht wird, muß natürlich bei der Konfigurierung bzw. bei der Festlegung der einzelnen Parameter berücksichtigt werden.

Als *Integrationsaufwand* wird im folgenden der Aufwand bezeichnet, der bei Änderungen von Komponenten anfällt, die sich bereits vor der Integration eines weiteren Systems im IVAS befinden. Unter *Änderungsaufwand* ist hingegen der Aufwand zu verstehen, der für die Überarbeitung des Altsystems (z.B. Erstellung eines Zugriffsmoduls) notwendig ist. Der Integrationsaufwand ist gering, wenn die Einbindung eines weiteren Anwendungssystems nur an wenigen Systemen innerhalb des IVAS Änderungen erforderlich macht. Aus dieser Gesetzmäßigkeit kann man ableiten, daß eine Verteilungsarchitektur - bezogen auf den Integrationsaufwand - umso wirtschaftlicher ist, je zentralistischer sie organisiert ist. Wenn beispielsweise nur ein zentraler Daten-Broker vorgesehen ist (Variante 1 des Parameters Datenzugriffsmanagement), muß ein zu integrierendes Datenobjekt nur im Schema dieses einen Brokers ergänzt werden. Der Ände-

---

13) Vgl. dazu die Ausführungen in Kapitel 3.

rungsaufwand ist in diesem Fall ebenfalls relativ gering, da die Zugriffe des Altsystems lediglich zum Daten-Broker umzuleiten sind. Wenn die Anwendungssysteme im IVAS hingegen direkt auf die datenverwaltenden Systeme zugreifen (Variante 3), muß das Altsystem u.U. um Zugriffe auf viele verschiedene andere Systeme ergänzt werden.

Wenn ein Altsystem vor der Integration lokale Integritätsregeln überwacht, führen die Varianten des Parameters Integritätsmanagement, die eine Verlagerung dieser Aufgabe implizieren (Varianten 1, 3 und 4), sowohl zu einem hohen Integrationsaufwand als auch zu einem hohen Änderungsaufwand. Wertebereichsprüfungen im Quelltext müssen untersucht, dort entfernt und bei einer zentralen Instanz hinterlegt werden. Nur Variante 2 beläßt die Überprüfung lokaler Integritätsregeln bei dem System, das die Daten verwaltet, und verursacht diesbezüglich weder Integrations- noch Änderungsaufwand. Wenn die Wertebereichsprüfungen noch zu implementieren sind, verursacht diese Variante allerdings einen hohen Änderungsaufwand.

Die Realisierung globaler Regeln führt unabhängig von der gewählten Variante immer sowohl zu Integrationsaufwand als auch zu Änderungsaufwand, da erst durch die Integration die Notwendigkeit solcher Regeln entsteht. Bei den Varianten 1 bis 3 (Überwachung durch eine zentrale Instanz) muß einerseits jeder schreibende Zugriff um eine Anfrage an die zentrale Instanz erweitert werden (Änderungsaufwand), damit der Zugriff dort auf seine Verträglichkeit mit eventuell vorhandenen Regeln geprüft werden kann. Andererseits müssen die globalen Regeln formuliert und bei der zentralen Instanz hinterlegt werden (Integrationsaufwand). Bei Variante 4 müssen globale Regeln durch das System überwacht werden, das den schreibenden Zugriff auslöst. Bei dieser Variante entsteht "nur" Änderungsaufwand, der aber einen großen Umfang annehmen kann, wenn viele zusätzliche anwendungsübergreifende Zugriffe im Altsystem realisiert werden müssen. Zusätzlich besteht die Möglichkeit, daß eine globale Integritätsregel redundant abgelegt wird, nämlich bei allen Systemen, die schreibend auf ein bestimmtes Datenobjekt zugreifen. Eine spätere Änderung dieser Regel ist dann mit einem großen (Integrations-)Aufwand verbunden.

Variante 1 des Parameters Aktualisierungsmanagement ist - bezogen auf den Änderungsaufwand - optimal, weil die Datenzugriffe lediglich um eine entsprechende Meldung an den zentralen Aktualisierungsmanager ergänzt werden müssen. Der Integrationsaufwand ergibt sich aus der Anzahl der Datenobjekte, die durch die Integration zusätzlich redundant werden. Diese Objekte sowie jeweils der zugehörige Aktualisierungsmodus müssen dem Aktualisierungsmanager "mitgeteilt" werden. Bei Variante 2 sind hingegen Prozeduren zu implementieren bzw. zu generieren, die die Aktualisierungen durchführen[14]. Der Änderungsaufwand ist in diesem Fall sehr hoch. Der Integrationsaufwand ist hier allerdings als noch höher einzustufen, da die Aktualisie-

---

[14] Vgl. Eicker et al. (1993), S. 76.

rungsprozeduren aller Systeme, die ein - bezüglich des Altsystems - redundantes Datenobjekt verwalten, angepaßt werden müssen.

2.2.4 Einsatzform des IVAS

Die Einsatzform des IVAS ist im weitesten Sinne ebenfalls ein Wirtschaftlichkeitsfaktor der Verteilungsarchitektur, nämlich als Kontextfaktor. Wenn das IVAS beispielsweise in der Sachbearbeitung eingesetzt wird (Dialogbetrieb), führen lange Antwortzeiten des Systems zu Wartezeiten des Benutzers und damit zu vermeidbaren Personalkosten. Wenn die vorherrschende Verarbeitungsform hingegen der Batch-Betrieb ist, kommt dem Laufzeitverhalten eine wesentlich kleinere Bedeutung zu.

So hängt die Auswahl des Aktualisierungsmodus im wesentlichen von der Einsatzform bzw. vom Umfeld des IVAS ab. Wenn ein deutliches Übergewicht an lesenden bzw. schreibenden Zugriffen vorausgesetzt werden kann, ist die Auswahl des Aktualisierungsmodus unproblematisch: es kann jeweils ein Modus ausgewählt werden, der die weniger wichtigen Zugriffe "diskriminiert". Anderenfalls müssen geeignete Kompromißlösungen realisiert werden. Eventuell ist sogar die Unterteilung des IVAS-Datenbestandes in - bezogen auf die Häufigkeit von lesenden bzw. schreibenden Zugriffen - homogene Bereiche unter Verwendung des jeweils optimalen Verfahrens sinnvoll. In Abbildung 5 sind die Interdependenzen und Determinanten der Wirtschaftlichkeitsfaktoren dargestellt.

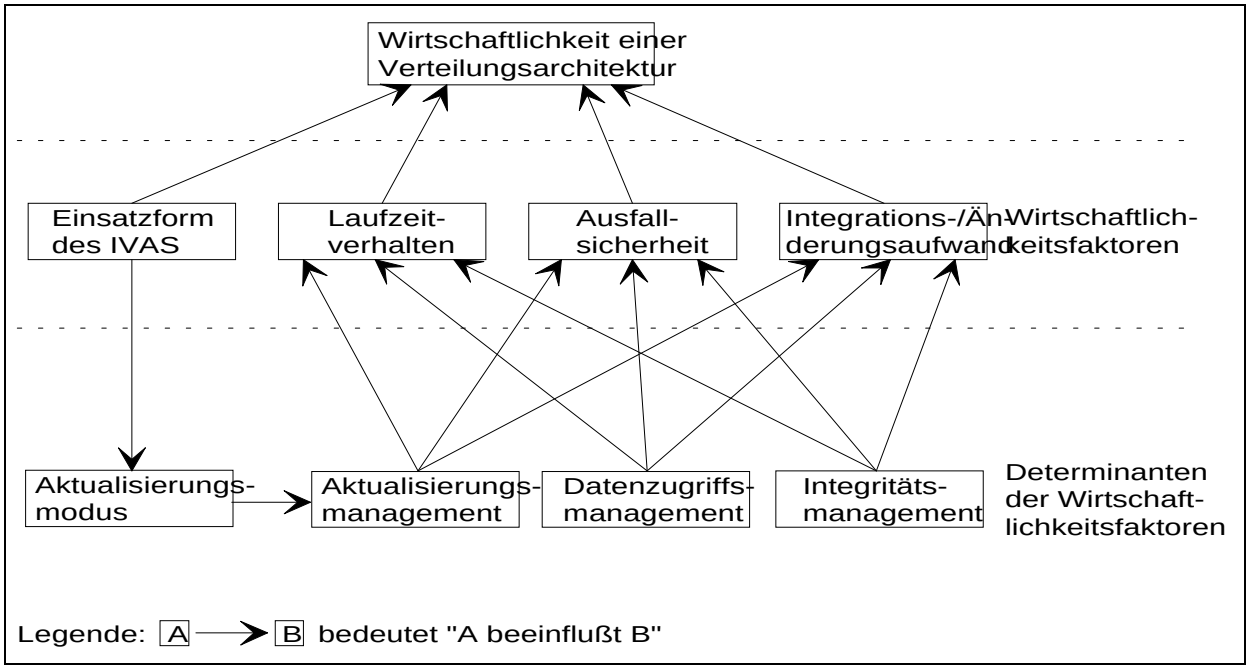

Abb. 5: Interdependenzen und Determinanten der Wirtschaftlichkeitsfaktoren (bezogen auf die Verteilungsarchitektur)

3 Reengineering vs. Neuentwicklung

3.1 Wirtschaftlichkeit als Entscheidungskriterium vs. Wirtschaftlichkeitsfaktoren als Entscheidungsgrundlage

Die Frage, ob ein vorhandenes Software-Altsystem einem Reengineering-Prozeß zu unterziehen oder aber neu zu entwickeln ist, sollte - idealtypisch betrachtet - letztlich anhand der Wirtschaftlichkeit beantwortet werden. Die Definition des Begriffs *Wirtschaftlichkeit* ist jedoch nur in einer sehr abstrakten Form eindeutig, nämlich als Verhältnis von erzielten Ergebnissen zu dem dazu getätigten Mitteleinsatz[15]. Konkrete Ausgestaltungen des Wirtschaftlichkeitsbegriffs existieren in großer Zahl. In der Investitionsrechnung wird die Wirtschaftlichkeit ausschließlich basierend auf monetären Größen beurteilt. Daneben finden sich Verfahren, die den Anspruch erheben, auch andere quantitative sowie qualitative Effekte (i.S.v. erzielten Ergebnissen) zu erfassen[16].

Die Anwendung der statischen und dynamischen Verfahren der Investitionsrechnung (Kosten- und Gewinnvergleichsrechnung, Kapitalwertmethode, Methode des internen Zinsfußes usw.) zur Beurteilung der Wirtschaftlichkeit von betrieblichen Anwendungssystemen scheitert zum einen, da die monetären Auswirkungen des Informationssystemeinsatzes (i.S.v. Aus- und Einzahlungen) nicht hinreichend genau prognostizierbar sind[17]. Zum anderen kann der nicht-monetäre Nutzen, wenn er sich nicht in monetäre Größen überführen läßt (z.B. im Fall besserer und schnellerer Informationen), nicht berücksichtigt werden.

Die Nachteile der Investitionsrechenverfahren sollen durch die Verwendung einer modifizierten Wirtschaftlichkeitskennzahl überwunden werden, die den Kosten, statt einer rein monetären Größe, den Nutzen als Aggregat von monetärem und nicht-monetärem Nutzen gegenüberstellt[18]:

$$Wirtschaftlichkeit = \frac{Nutzen}{Kosten}$$

Die Verwendung dieses Beurteilungsmaßes ist jedoch ebenfalls problematisch, da drei Voraussetzungen zu erfüllen sind:

1. Der Nutzen muß prognostizierbar und insbesondere quantifizierbar sein.

---

15) Vgl. Horváth (1988), S. 2 ff.
16) Eine Übersicht findet sich in Nagel (1990), S. 41 ff.
17) Vgl. Nagel (1990), S. 70.
18) Vgl. z.B. Scherff (1986), S. 6.

2. Die Kosten müssen prognostizierbar sein.

3. Sowohl Nutzen als auch Kosten müssen dem Untersuchungsobjekt (in diesem Fall: dem Anwendungssystem) eindeutig zugeordnet werden können.

Die Probleme bei der Quantifizierung von Nutzen allgemein[19] und insbesondere bei der Messung von Nutzeffekten des Einsatzes betrieblicher Anwendungssysteme sind in der Literatur bereits ausführlich diskutiert worden[20]. Die Schätzung der mit der Entwicklung eines Anwendungssystems verbundenen Kosten stellt ebenfalls ein Problem dar, da die existierenden Verfahren mit Ungenauigkeit behaftet sind[21].

Schließlich muß auch die Erfüllbarkeit der dritten Voraussetzung (Zuordenbarkeit von Nutzen und Kosten) bezweifelt werden[22]. Ob eine bestimmte positive oder sogar negative Entwicklung auf den Einsatz eines bestimmten Anwendungssystems zurückzuführen ist, läßt sich nicht immer genau sagen, da während der Nutzungsdauer auch andere Einflüsse auf den Einsatzbereich einwirken; eine ceteris paribus-Betrachtung kann somit nicht unterstellt werden.

Darüber hinaus erschwert die zunehmende strategische Ausrichtung von Informationssystemen[23] eine verursachungsgerechte Zurechnung des gestifteten Nutzens zu den damit verbundenen Kosten[24]. Die Berücksichtigung von Integrationseffekten in einer Wirtschaftlichkeitsuntersuchung stellt ein weiteres noch ungelöstes Problem dar[25]. Anselstetter beschreibt das zugrundeliegende Problem treffend wie folgt: "Die Wirkungen der DV lassen sich um so schwerer ermitteln und zurechnen, je mehr Unternehmensebenen bzw. -bereiche von einer Anwendung betroffen sind"[26]. Ähnliche Probleme sind aus dem Bereich des Computer integrated manufacturing (CIM) bekannt; hier versagen die traditionellen Verfahren der Wirtschaftlichkeitsberechnung ebenfalls, da eine partielle (isolierte) Untersuchung von Investitionsobjekten (CIM-Komponenten) den Integrationsnutzen nicht berücksichtigt[27].

---

19) Vgl. Laux (1988), S. 3 ff.
20) Vgl. z.B. Weingart (1987), S. 26 ff., oder Eisele (1992), S. 15. Griese et al. sprechen in diesem Zusammenhang auch von einer analytischen Lücke, die von der betriebswirtschaftlichen Forschung noch zu schließen sei (vgl. Griese et al. (1987), S. 551).
21) Vgl. Heemstra (1992), S. 627, oder Knöll, Busse (1991).
22) Vgl. Bacon (1992), S. 337.
23) Vgl. Ott, Rausch (1992), S. 199.
24) Vgl. Nagel (1990), S. 25 ff.
25) Vgl. Schumann (1993), S. 177.
26) Anselstetter (1984), S. 10.
27) Vgl. Zahn, Dogan (1991), S. 10, oder Siegwart, Singer (1991), S. 63.

Betrachtet man die Möglichkeit, ein Anwendungssystem "lediglich" zu überarbeiten (Reengineering), finden sich zwar vereinzelt Berichte über erfolgreich verlaufene Projekte[28], Hilfsmittel zur Beurteilung ihrer Wirtschaftlichkeit stehen jedoch nicht zur Verfügung[29].

Da die Wirkungen einer Entscheidung für eine Neuentwicklung oder für das Reengineering eines existierenden Altsystems nicht hinreichend genau und isoliert von anderen Einflüssen prognostiziert werden können, wird in diesem Arbeitsbericht eine andere Vorgehensweise vorgestellt. Sie basiert auf den folgenden Prämissen:

1. Die Integration eines Anwendungssystems (durch Ersetzung oder durch Reengineering) wird durch einen betrieblichen Handlungsbedarf begründet; mit der Integration werden somit konkrete Ziele verfolgt.

2. Da die Alternativen den gesetzten Zielen entsprechend zu gestalten sind, wird davon ausgegangen, daß der Zielerreichungsgrad bei beiden Alternativen identisch ist.

3. Der Nutzen, der über das durch die Ziele implizierte Maß hinausgeht, wird als äquivalent vorausgesetzt.[30]

Bei Gültigkeit der Prämissen kann man schließen, daß die wirtschaftlichste Alternative diejenige ist, die den geringeren Realisierungsaufwand verursacht. Dazu sollen im folgenden die Wirtschaftlichkeitsfaktoren und insbesondere deren Determinanten herausgearbeitet werden. Eine Bewertung der Alternativen anhand der Wirtschaftlichkeitsfaktoren soll schließlich eine Entscheidungsempfehlung ermöglichen. Der Fall, daß der Nutzen der beiden Alternativen differiert (d.h., Prämisse 3 gilt nicht), ist vom Entscheider unter Hinzuziehung der Entscheidungsempfehlung zu berücksichtigen[31].

3.2 Wirtschaftlichkeitsfaktoren

Die Wirtschaftlichkeitsfaktoren, die für oder gegen die Reengineering-Alternative sprechen, sind grundsätzlich aus der Software-Lebenszyklus-Phase *Wartung* abzuleiten, da zum Teil ähnliche Arbeiten, allerdings in größerem Umfang, anfallen. In Anlehnung an die unterschiedlichen War-

---

28) Vgl. Kador (1992).
29) Vgl. Rochester, Douglass (1991), S. 7.
30) Dieser Prämisse liegt die Annahme zugrunde, daß bei Durchführung einer Alternative positive Effekte (i.S.v. Nutzen) auftreten können, deren Erzielung nicht geplant bzw. beabsichtigt war.
31) Einen ähnlichen Ansatz - mit implizit identischen Prämissen - schlägt Kargl vor: Der "Kostenvergleich ist für jede Alternative durch die Analyse qualitativer Wirtschaftlichkeitsfaktoren und durch die Analyse von Risiken zu ergänzen." (Kargl (1993), S. 102).

tungstätigkeiten (vgl. Abbildung 6) sind insgesamt vier Wirtschaftlichkeitsfaktoren des integrationsorientierten Reengineering zu unterscheiden, die bezogen auf einen Integrationsschritt zu untersuchen sind: der *Einarbeitungsaufwand*, der *Änderungs-/Integrationsaufwand*, der *Testaufwand* sowie der *Dokumentationsaufwand*.

- *Einarbeitungsaufwand:* Der Einarbeitungsaufwand ist ein wesentlicher Wirtschaftlichkeitsfaktor, da die Einarbeitung in das vorliegende Altsystem - wie bei der Wartung - einen wesentlichen Teil der zeitlichen Ressourcen für das integrationsorientierte Reengineering beanspruchen wird. Zu beachten ist, daß im Rahmen der Wirtschaftlichkeitsuntersuchung bereits eine Einarbeitung in das Altsystem notwendig ist, deren Ergebnisse - sofern sie geeignet dokumentiert wurden - bei einem eventuellen späteren Reengineering-Projekt verwendet werden können.

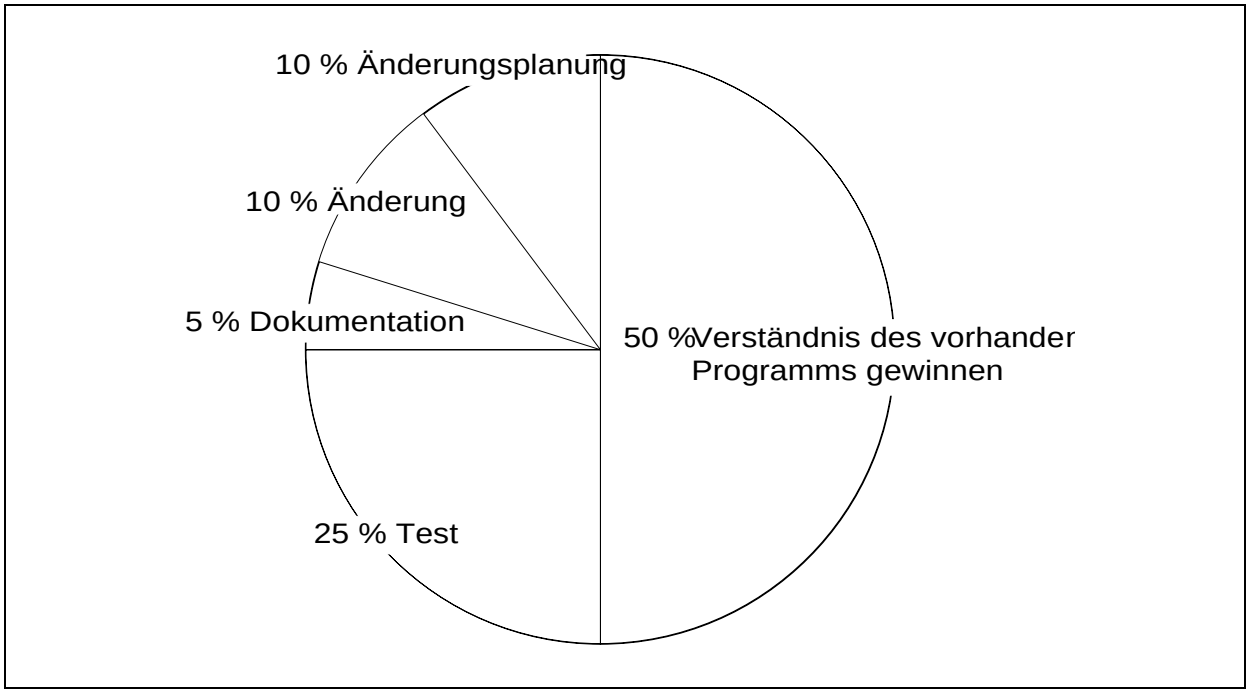

Abb. 6: Zeitliche Verteilung der Wartungstätigkeiten[32)]

- *Änderungs-/Integrationsaufwand:* Unter Änderungs- bzw. Integrationsaufwand ist der Aufwand zu verstehen, der unter der gegebenen Zielsetzung entsteht, wenn ein vorhandenes Altsystem geändert und den Vorgaben der Verteilungsarchitektur entsprechend in das IVAS integriert wird.

---

32) Mit dem Quellenhinweis "Source: IBM: Guide Study on Maintenance" entnommen aus Tempel (1992), S. 219.

- *Testaufwand:* Nach der Integration eines Altsystems in das IVAS muß geprüft werden, ob die Datenzugriffe, Aktualisierungen und Integritätsprüfungen korrekt ablaufen. Der Aufwand, der für Arbeiten in diesem Zusammenhang anfällt, ist der Testaufwand.

- *Dokumentationsaufwand:* Parallel zur Integration eines Altsystems in das IVAS, muß die Dokumentation des IVAS aktualisiert werden. Darunter fallen Erweiterungen der Benutzer- und Programmierhandbücher und die Aktualisierung der Dokumentation des IVAS-Datenmodells. Der für diese Tätigkeiten notwendige Aufwand ist der Dokumentationsaufwand.

3.3 Determinanten der Wirtschaftlichkeitsfaktoren

Die Determinanten der genannten Wirtschaftlichkeitsfaktoren haben nicht notwendigerweise Auswirkungen auf nur einen der Faktoren. Aus diesem Grund werden die einzelnen Determinanten im folgenden erläutert und auf ihre jeweiligen Konsequenzen bezüglich der einzelnen Wirtschaftlichkeitsfaktoren untersucht.

Die Beschaffenheit des vorliegenden Altsystems hat einen wesentlichen Einfluß auf den Aufwand, der bei der Integration des Systems in das IVAS entsteht. Dabei handelt es sich neben bestimmten Merkmalen des Programms (i.S.v. Quelltext) auch um Merkmale weiterer Komponenten des Altsystems, wie z.B. Verfügbarkeit bzw. Qualität der Entwurfsdokumente. Die Untersuchung aller relevanten Merkmale des Altsystems ist *eine* Voraussetzung, um abschätzen zu können, ob die Integration in das IVAS und eventuell weitere Modifikationen bzw. Ergänzungen mit vertretbarem Aufwand gegenüber einer Neuentwicklung durchgeführt werden können.

Verständlichkeit des Quelltextes

Aus der Menge der Softwarequalitätsmerkmale ist im Rahmen des Reengineering das Merkmal *Änderbarkeit* bzw. *Wartungsfreundlichkeit* von Bedeutung. Insbesondere ist die Verständlichkeit der vorliegenden Quelltexte[33)] zu untersuchen, die im wesentlichen den Aufwand für das Reverse engineering bestimmt.

Die Verständlichkeit des Quelltextes wird determiniert durch die Merkmale Modularisiertheit/ Modul-Vollständigkeit, Parametrisiertheit und Datenkommunikation, Datenstrukturiertheit, Programmkomplexität, Selbstdokumentiertheit und durch die verwendete Programmiersprache.

---

33) Vgl. Debest et al. (1992), S. 60.

- Die *Modularisiertheit* soll, analog zum Begriff der Modularisierung[34], ein Maß dafür sein, inwieweit ein Programm in Unterprogramme (Module) zerlegt ist, die jeweils weitgehend unabhängig von anderen Unterprogrammen einzelne Aufgaben lösen. In diesem Zusammenhang ist auch die *Modul-Vollständigkeit* (Module completeness) zu nennen; darunter wird die Eigenschaft verstanden, daß ein Modul ohne Kenntnis der über- und untergeordneten Module verständlich ist[35]. Das setzt voraus, daß die Datenkommunikation mit anderen Unterprogrammen sowie das Unterprogramm selbst ausreichend dokumentiert sind.

- Die *Parametrisiertheit* beschreibt zusammen mit der *Datenkommunikation* die Art und Weise, wie die Weitergabe von Daten zwischen den einzelnen Unterprogrammen erfolgt. Neben der Verwendung von Parametern ist zum Beispiel auch die Weitergabe mit Hilfe von globalen Variablen möglich. Diese Variante kann insbesondere dann negative Auswirkungen auf die relative Vorteilhaftigkeit des Reengineering haben, wenn durch Seiteneffekte die Verständlichkeit des Quelltextes stark herabgesetzt wird[36]. Wenn die Seiteneffekte darüber hinaus nicht dokumentiert sind, ist auch keine Modul-Vollständigkeit gegeben.

- *Datenstrukturiertheit* ist ein Merkmal, unter dem die Verwendung von Typen, die Strukturierung von Daten mit Hilfe von Records und Arrays sowie die Verwendung "sprechender" Bezeichner zu verstehen ist.

- *Programmkomplexität* ist ein Maß "für Umfang und Vielfältigkeit der Beziehungen der Elemente in einem Programmsystem"[37]. Die Komplexität "im Großen" wird durch das Merkmal *Modularisiertheit* erfaßt. Die Komplexität "im Kleinen" wird hingegen anhand der Struktur des Quelltextes ermittelt, wobei die Verwendung weniger, standardisierter Steuerkonstrukte (Sequenz, Selektion, Repetition) bei gleichzeitiger Vermeidung von Sprunganweisungen (GOTO oder EXIT) als optimal anzusehen ist[38]. Ein Beispiel für vermeidbare Programmkomplexität ist die Verwendung mehrerer geschachtelter IF-Statements statt der Verwendung eines äquivalenten CASE-Statements.

- *Selbstdokumentiertheit* beschreibt die Eigenschaft eines Programms, "daß Daten- und Prozedurnamen, Anweisungen und Kommentare ihre Bedeutung erkennen lassen (...), ohne daß zusätzliche Erläuterungen notwendig sind"[39]. Diese Eigenschaft unterstützt hauptsächlich die Lesbarkeit und damit auch die Verständlichkeit eines Programms.

---

[34]) Vgl. z.B. Kurbel (1985), S. 11 ff.
[35]) Vgl. Wei, Eldridge (1991), S. 35; diese Forderung spiegelt sich auch im Lokalitätsprinzip wider.
[36]) Vgl. Tempel (1992), S. 224 f.
[37]) Heinemann (1987), S. 115.
[38]) Vgl. Kurbel (1985), S. 15
[39]) Kurbel (1985), S. 3.

- Die verwendete *Programmiersprache* hat einen indirekten Einfluß auf den Reengineering-Aufwand, weil die Berücksichtigung einiger der genannten Merkmale (z.B. Datenstrukturiertheit) von der Qualität der Programmiersprache[40)] abhängig ist.

Die Verständlichkeit des Quelltextes determiniert im wesentlichen den Einarbeitungsaufwand. Wenn Änderungen direkt im Quelltext durchgeführt werden können, ist auch der Änderungs- bzw. Integrationsaufwand und der Testaufwand beeinflußt.

Daten-Programm-Abhängigkeit

Ein Merkmal des vorliegenden Softwaresystems, das in besonderem Maß den Änderungs- bzw. Integrationsaufwand beeinflußt, ist die Daten-Programm-Abhängigkeit. Wenn ein Programm seine Daten selbst in Form von Dateien verwaltet, besteht eine stärkere Daten-Programm-Abhängigkeit als bei der Verwendung eines Datenbankmanagementsystems. Änderungen sowohl am Datenmodell als auch am Programm sind in letzterem Fall wesentlich leichter[41)] und damit auch kostengünstiger durchzuführen. Insbesondere der Vergleich der beiden Datenmodelle (das des IVAS einerseits, das des Altsystems andererseits) und die Überführung des Altsystems in das IVAS wird vereinfacht, da die Datendefinitionen bereits isoliert in den Systemtabellen des Datenbankmanagementsystems vorliegen.

Wartungszustand

Der Einarbeitungsaufwand hängt auch davon ab, ob und in welcher Form das vorliegende Altsystem bereits gewartet worden ist. Wenn im Laufe der Zeit durch Wartung viele Änderungen vorgenommen werden, die nicht dokumentiert sind, verliert das Altsystem unter Umständen stark an (Software-)Qualität und auch an Homogenität[42)]. Sofern Änderungen im vorliegenden Quelltext durchgeführt werden, beeinflußt der Wartungszustand auch den Änderungs- bzw. Integrationsaufwand und den Testaufwand.

Verfügbarkeit und Qualität der Software-Dokumentation

Eine weiteres wichtiges Merkmal des Altsystems ist die Verfügbarkeit möglichst aller Dokumente, die während der Entwicklung des Altsystems erstellt wurden. Darunter fallen das Pflichtenheft, die Systemspezifikation, der System- und Komponentenentwurf, die Testdokumente sowie weitere Dokumente wie z.B. das Benutzerhandbuch u.a. Insbesondere die Verfügbarkeit

---

40) Vgl. Kurbel (1985), S. 6 ff.
41) Vgl. Stucky, Krieger (1990), S. 840.
42) Conte et al. sprechen in diesem Zusammenhang auch vom *Law of Increasing Entropy*: "The entropy of a system increases with time unless specific work is done to maintain it or to reduce it" (Conte et al. (1986), S. 10).

des System- und Komponentenentwurfs würde das Reengineering begünstigen, da das Reverse engineering auf einer höheren Abstraktionsebene ansetzen könnte. Die mühsame Analyse des Quelltextes könnte entfallen oder wäre zumindest wesentlich leichter. Eine hohe Qualität der Software-Dokumentation (i.S.v. Verständlichkeit, Vollständigkeit, Zweckmäßigkeit, Fehlerfreiheit, Aktualität, maschinelle Weiterverarbeitbarkeit) begünstigt deren Nutzung innerhalb eines Reengineering-Projekts. Eine zwar verfügbare, aber inaktuelle Software-Dokumentation ist ein gegen Reengineering sprechendes Argument. Die Verfügbarkeit und Qualität der Entwicklungsdokumente ist für alle vier Wirtschaftlichkeitsfaktoren von Bedeutung. Der Einarbeitungsaufwand, der Änderungs- bzw. Integrationsaufwand, der Testaufwand und der Dokumentationsaufwand könnten durch eine gute und vollständige Dokumentation wesentlich reduziert werden.

Verwendung moderner Entwicklungsmethoden

Obwohl der positive Einfluß moderner Entwicklungsmethoden auf die Wartbarkeit von Softwaresystemen empirisch nicht eindeutig erwiesen ist[43)], läßt sich doch vermuten, daß ihr Einsatz zu einem homogeneren und damit besser zu analysierenden Altsystem führt. Zum einen ist es wesentlich wahrscheinlicher, daß Entwicklungsdokumente erstellt wurden und verfügbar sind, zum anderen ergeben sich aus einem strukturierten Entwurf positive Auswirkungen auf die bereits genannten Merkmale des Quelltextes (z.B. bessere Modularisiertheit).

Restnutzungsdauer der Hardware- und Softwareplattform

Wenn die dem Altsystem zugrundeliegende Hardware- und/oder Softwareplattform eine kurze Restnutzungsdauer besitzt, muß im Rahmen des Reengineering zusätzlich eine Portierung in die neue Umgebung durchgeführt werden. Der (Änderungs-)Aufwand für eine solche Portierung reduziert zwar einerseits die Vorteilhaftigkeit der Reengineering-Alternative, andererseits sprechen bestimmte positive Eigenschaften des vorliegenden Altsystems (z.B. gute Änderbarkeit) gleichzeitig für ein wirtschaftliches Reengineering als auch für eine unproblematische Portierung.

Umfang des Zielsystems

Eine wesentliche Determinante des Wirtschaftlichkeitsfaktors Änderungsaufwand ist der Umfang des Zielsystems, das den Handlungsbedarf begründet. Die Integration eines Altsystems in das IVAS ist innerhalb des Zielsystems nur ein Teilziel. Weitere Teilziele (neue Benutzerober-

---

43) Lientz und Swanson belegen in einer Untersuchung die positiven Auswirkungen auf die Software-Qualität (vgl. Lientz, Swanson (1981), S. 767), Dekleva konnte hingegen nur eine Verschiebung zwischen einzelnen Wartungstätigkeiten feststellen (vgl. Dekleva (1992), S. 370).

fläche, zusätzliche Funktionen usw.), die häufig sogar der Anstoß für ein Reengineering-Projekt[44] sind, müssen berücksichtigt werden. Je umfangreicher die Ziele sind, die eine Erweiterung des vorliegenden Altsystems implizieren, desto ungünstiger ist die Reengineering-Alternative. Erweiterungen könnten bei einer Neuentwicklung wirtschaftlicher realisiert werden, da die Einarbeitung in den diesbezüglichen Teil des Altsystems (bzw. der Aufwand dafür) entfallen würde.

Qualifikation, Erfahrung und Motivation des Reengineering-Personals

Die Qualifikation, Erfahrung und Motivation des Reengineering-Personals ist für alle vier Wirtschaftlichkeitsfaktoren von besonderer Bedeutung. An das Personal sind bei einem Reengineering-Projekt insbesondere bezüglich der Qualifikation und Erfahrung höhere Anforderungen zu stellen als bei einer Neuentwicklung. Es ist anzunehmen, daß langjährige Erfahrung in Programmierung und Wartung den Aufwand eines Reengineering-Projekts reduzieren und gleichzeitig dessen Erfolg begünstigen[45]. Darüber hinaus erfordert die Analyse von eventuell sehr schwer lesbaren (fremden) Quelltexten oder einer "schlechten" Software-Dokumentation eine hohe geistige Flexibilität.

Die *Motivation* der Mitarbeiter ist, im Gegensatz zur Qualifikation und Erfahrung, eine modellinterne Größe, d.h., sie kann von Seiten des Projektmanagements positiv beeinflußt werden. Der umfassendere Charakter eines (integrationsorientierten) Reengineering-Projekts kann - im Gegensatz zu Wartungsarbeiten[46] - die Motivation der Mitarbeiter eventuell fördern.

Verfügbarkeit der Entwickler

Der Einarbeitungsaufwand kann durch die Verfügbarkeit der Entwickler des vorliegenden Altsystems eventuell reduziert werden[47]. Nicht dokumentierte Entwurfsentscheidungen und auch bestimmte Codierungen können im Dialog mit den Entwicklern eventuell leichter nachvollzogen werden.

Der mit einem Reengineering-Projekt verbundene Aufwand hängt auch davon ab, inwieweit der Reengineering-Prozeß bzw. der Integrationsschritt automatisiert werden kann[48]. Dabei stellt

---

44) Vgl. Debest et al. (1992), S. 62.
45) Im Bereich der Wartung wird dieser Zusammenhang ebenfalls vermutet (vgl. Heinemann (1987), S. 146).
46) Die Motivation von Wartungsprogrammierern wird eher als problematisch eingestuft (vgl. Heinemann (1987), S. 146 f.). Wartung wird z.B. wie folgt charakterisiert: "Maintenance work is dirty. Good maintenance is done behind the scenes, with the code and the machines. There is very little glory, noticeable progress, or chance for success." (Glass, Noiseaux (1981), S. 21).
47) Dieser Zusammenhang wurde, bezogen auf die Wartung, von Lientz und Swanson bereits nachgewiesen (vgl. Lientz, Swanson (1981), S. 767 f.).
48) Vgl. Tempel (1992), S. 214.

sich die Frage nach der Verfügbarkeit von Wartungswerkzeugen einerseits und von speziellen Reengineering-Werkzeugen[49)] andererseits.

Verfügbarkeit von Wartungswerkzeugen

Die Verfügbarkeit von "konventionellen" Wartungswerkzeugen kann je nach Funktionalität alle vier Wirtschaftlichkeitsfaktoren positiv beeinflussen. Glass und Noiseaux teilen Wartungs-Werkzeuge in zwei Kategorien ein, zum einen in Werkzeuge bzw. Hilfsmittel zur technischen Unterstützung und zum anderen in Werkzeuge zur administrativen Unterstützung der Wartung[50)]. Zur ersten Kategorie zählen Compiler, Assembler, Linker, Betriebssystem, Debugger, sogenannte Komparatoren zum Vergleich von Quelltexten oder Testfällen (Regressionstests[51)]), Cross reference-Listings, Texteditoren, Verifizierungs-Werkzeuge, Formatierungswerkzeuge und Präprozessoren zur Anpassung an die verwendete Programmiersprache. Zur zweiten Kategorie zählen Werkzeuge für das Wartungs-Projektmanagement (Erstellen von Reports u.ä.) und für die Dokumentation[52)].

Verfügbarkeit von Werkzeugen zur Unterstützung des Daten-Reengineering

Die angestrebte Datenintegration macht bei jedem Integrationsschritt die (möglichst weitgehende) Extraktion des Altsystemdatenmodells und den Vergleich mit dem Datenmodell des IVAS notwendig, damit Überschneidungen aufgedeckt werden können[53)]. Die Verfügbarkeit von Werkzeugen, die das Daten-Reengineering unterstützen, reduziert den Einarbeitungsaufwand[54)].

Verfügbarkeit von Integrationswerkzeugen

Sofern das Altsystem integrationsfähige Datenobjekte verwaltet, müssen diese in das IVAS integriert werden. Werkzeuge, die die Integration unterstützen sollen, müssen in ihrem Lei-

---

49) Vgl. Eicker, Schnieder (1992), S. 15 ff.

50) Vgl. Glass, Noiseaux (1981), S. 55 ff; Heinemann unterteilt die erste Kategorie in Software-Tools zur Unterstützung der analysierenden Tätigkeiten, zur Erzeugung, Modifizierung und zum Testen von Programmen (vgl. Heinemann (1987), S. 148).

51) Bei einem Regressionstest werden zwei Versionen eines Programms (i.d.R. die alte und die überarbeitete) mit identischen Daten getestet. Anschließend werden die Ergebnisse auf Abweichungen untersucht.

52) In jüngeren Publikationen zur Softwarewartung werden z.B. Werkzeuge für das Reverse engineering und das Daten-Reengineering als Wartungswerkzeuge klassifiziert (vgl. Lehner (1991), S. 238 ff.). Darin kommt implizit die Ausweitung des Wartungsbegriffs in Richtung Reengineering zum Ausdruck.

53) Vgl. Eicker et al. (1992), S. 142 ff.

54) Vgl. dazu auch Brosda, Herbst (1993), S. 369.

stungsumfang an die Merkmale der Verteilungsarchitektur des IVAS[55)] angepaßt sein. Im einzelnen muß ein Werkzeug folgendes leisten:

- Erweiterung des IVAS-Schemas um die zu integrierenden Datenobjekte, abhängig von der Art des *Datenzugriffsmanagements* im IVAS;

- Realisierung aller notwendigen Aktualisierungsmechanismen für redundante Datenobjekte, abhängig von der Art des *Aktualisierungsmanagements* im IVAS; darüber hinaus müssen geeignete Aktualisierungsmodi implementiert werden;

- Realisierung aller notwendigen integritätssichernden Maßnahmen (lokale und globale Integritätsregeln), abhängig von der Art des *Integritätsmanagements* des IVAS;

- Umstellung der Datenzugriffe des Altsystems, wenn bestimmte Datenobjekte nach der Integration nur indirekt über einen Daten-Broker erreicht werden können;

- Generierung von Zugriffsmodulen oder Unterstützung bei ihrer Erstellung, wenn Dateien in das IVAS integriert werden. Ein Zugriffsmodul dient als Schnittstelle zwischen der Datei und dem Daten-Broker und synchronisiert die Zugriffe des Altsystems und des Daten-Brokers.

Die Verfügbarkeit von Werkzeugen, die möglichst viele der genannten Anforderungen abdekken, kann den Änderungs- bzw. Integrationsaufwand stark reduzieren.

Verfügbarkeit von Testwerkzeugen für das integrationsorientierte Reengineering

Schließlich sind auch Werkzeuge notwendig, mit deren Hilfe nach erfolgter Integration die Funktionsfähigkeit des IVAS getestet werden kann (Reduktion des Testaufwands). Insbesondere ist zu prüfen, ob Datenzugriffe und Aktualisierungen korrekt ablaufen. Darüber hinaus sollte die Suche nach sich widersprechenden globalen Integritätsregeln unterstützt werden.

Größe des IVAS

Im Laufe des Integrationsprozesses nimmt die Größe des IVAS (bzw. die Anzahl der Datenobjekte im IVAS) stetig zu[56)]. Der Vergleich des IVAS-Datenmodells und des Datenmodells des zu integrierenden Altsystems wird immer aufwendiger (Einarbeitungs- und Änderungs- bzw. Integrationsaufwand steigen). Synonyme und Homonyme sind zunehmend schwieriger zu identifi-

---

55) Vgl. Eicker et al. (1993), S. 73 ff.
56) Vgl. ebenda, S. 72.

zieren, da nur auf Seiten des IVAS-Datenmodells von einer guten Dokumentation ausgegangen werden kann. Bei einer Neuentwicklung könnte man stattdessen auf dem IVAS-Datenmodell aufbauen und es eventuell um weitere, von dem zu entwickelnden Anwendungssystem benötigte Datenobjekte erweitern. Auch der Testaufwand nimmt mit der Größe des IVAS, d.h. mit der Größe des Testobjekts, zu.

Anhand dieses Merkmals wird ersichtlich, daß die Vorteilhaftigkeit der Reengineering-Alternative auch davon abhängig ist, ob das Altsystem sofort oder erst später in das IVAS eingebunden wird. Diese Tatsache muß auf der übergeordneten Ebene, nämlich bei der Planung des Integrationsprozesses, Berücksichtigung finden.

3.4 Metriken

Metriken können als Maße zur quantitativen Bewertung von Software unter verschiedenen Zielsetzungen verstanden werden. Die grundsätzliche Frage lautet: "Is it possible to identify or define indices of merit that can support quantitative comparisons and evaluations of software and of the process associated with its design, development, use, maintenance, and evolution?"[57] Analog zu dieser informellen Abgrenzung werden Metriken in zwei Kategorien eingeteilt[58]:

- "Process metrics quantify attributes of the development process and of the development environment."

- "Product metrics are measures of the software product."

Bezogen auf die Merkmale des Altsystems bietet sich die Verwendung von Produkt-Metriken an, die im Bereich der Software-Qualitätssicherung verbreitet sind. Besonders vielfältige Varianten finden sich zur Bewertung des Quelltextes (Size metrics, Data structure metrics, Logic structure metrics), aber auch Design-Metriken, die Verallgemeinerungen von Code-Metriken darstellen und ähnliche Merkmale auf höherem Niveau bewerten, wurden bereits vorgestellt[59].

Die Bewertung des Altsystems kann mit Hilfe existierender Metriken und zugehöriger Werkzeuge teilweise automatisch durchgeführt werden (z.B. bezüglich der Art der Datenkommunikation oder der Komplexität). Andere Eigenschaften des Quelltextes, wie z.B. die Qualität von Kommentaren, sind nur subjektiv und damit nicht algorithmisch bestimmbar.

---

[57] Conte et al. (1986), S. VII f.
[58] Vgl. ebenda, S. 19 f.
[59] Vgl. z.B. Conte et al. (1986), oder Brandl (1990).

Zur statischen Bewertung des Quelltextes ist in einem Kooperationsprojekt zwischen der École Polytechnique de Montréal und Bell Canada ein Werkzeug mit dem Namen DATRIX entwickelt worden, das 36 verschiedene Metriken auf prozedurale Programmiersprachen anwendet[60]. Ein solches Werkzeug könnte Bestandteil einer Reengineering-Umgebung sein[61] und die Abschätzung der Vorteilhaftigkeit eines Reengineering-Projekts unterstützen.

4 Ausblick

Das integrationsorientierte Reengineering ist ein Ansatz, mit dessen Hilfe die Integration betrieblicher Daten bei gleichzeitiger (und möglichst weitgehender) Weiterverwendung vorhandener Anwendungssysteme erreicht werden soll. Ob die Weiterverwendung eines Software-Altsystems mit entsprechenden Modifikationen - gegenüber einer Ersetzung - dem Wirtschaftlichkeitsprinzip entspricht, hängt einerseits von den Merkmalen bzw. Eigenschaften des Systems ab, andererseits aber auch von den zur Verfügung stehenden Methoden. Insbesondere die Verfügbarkeit von Werkzeugen, die entsprechende Methoden unterstützen, stellt eine wesentliche Voraussetzung für die Wirtschaftlichkeit des integrationsorientierten Reengineering dar.

---

60) Vgl. Robillard et al. (1991).
61) Vgl. Maiocchi (1990), S. 124.

Literatur

Ambichl, E., Heinrich, L.J.: Leistungsbewertung dialogorientierter Datenbanksysteme in Client/Server-Architekturen, Information Management 7 (1992) 3, S. 24-31.

Anselstetter, R.: Betriebswirtschaftliche Nutzeffekte der Datenverarbeitung, Berlin u.a. 1984.

Bacon, J.C.: The Use of Decision Criteria in Selecting Information Systems/Technology Investments, MIS Quarterly 16 (1992) 3, S. 335-353.

Balzert, H.: CASE - Systeme und Werkzeuge, Mannheim u.a. 1992.

Boehm, B.W.: Software Engineering Economics, Englewood Cliffs 1981.

Brandl, D.L.: Quality Measures in Design, ACM SIGSOFT Software Engineering Notes 15 (1990) 1, S. 68-72.

Brosda, V., Herbst, A.: Werkzeugunterstützte Datenintegration - Die Realisierung eines CIM-Systems, in: Kurbel, K. (Hrsg.): Wirtschaftsinformatik ´93, Heidelberg 1993, S. 362-377.

Chikofsky, E.J., Cross, J.H.: Reverse Engineering and Design Recovery: A Taxonomy, IEEE Software 7 (1990) 1, S. 13-17.

Conte, S.D., Dunsmore, H.E., Shen, V.Y.: Software Engineering Metrics and Models, Menlo Park u.a. 1986.

Debest, X.A., Knoop, R., Wagner, J.: REVENG: A cost-effective approach to reverse-engineering, ACM SIGSOFT Software Engineering Notes 17 (1992) 4, S. 60-67.

Dekleva, S.M.: The Influence of the Information Systems Development Approach on Maintenance, MIS Quarterly 16 (1992) 3, S. 355-372.

Deutsches Institut für Normung e.V.: DIN 66001, Informationsverarbeitung - Sinnbilder für Datenfluß- und Programmablaufpläne, Berlin 1977.

Eicker, S., Jung, R., Kurbel, K.: Anwendungssystem-Integration und Verteilungsarchitektur aus der Sicht des Reengineering, Informatik - Forschung und Entwicklung 8 (1993) 2, S. 70-78.

Eicker, S., Kurbel, K., Pietsch, W., Rautenstrauch, C.: Einbindung von Software-Altlasten durch integrationsorientiertes Reengineering, Wirtschaftsinformatik 34 (1992) 2, S. 137-145.

Eicker, S., Schnieder, T.: Reengineering, Arbeitsbericht Nr. 13 des Instituts für Wirtschaftsinformatik der Westf. Wilhelms-Universität Münster, Münster 1992.

Eisele, R.: Controlling integrierter Systeme, controller magazin (1992) 1, S. 15-18.

Griese, J., Obelode, G., Schmitz, P., Seibt, D.: Ergebnisse des Arbeitskreises Wirtschaftlichkeit der Informationsverarbeitung, zfbf 39 (1987) 7, S. 515-551.

Heemstra, F.J.: Software cost estimation, Information and Software Technology 34 (1992) 10, S. 627-639.

Heinemann, K.: Software-Wartung - Ein modellgestützter Ansatz zur Planung von Softwarewartungsstrategien, Münster 1987.

Horváth, P.: Grundprobleme der Wirtschaftlichkeitsanalyse beim Einsatz neuer Informations- und Produktionstechnologien, in: Horváth, P. (Hrsg.): Wirtschaftlichkeit neuer Produktions- und Informationstechnologien, Stuttgart 1988, S. 1-14.

Kador, J.: Reengineer To Boost Software Productivity, Datamation (1992) December 15, S. 57-58.

Kargl, H.: Controlling im DV-Bereich, München, Wien 1993.

Knöll, H.-D., Busse, J.: Aufwandsschätzung von Software-Projekten in der Praxis, Mannheim u.a. 1991.

Kurbel, K.: Programmierstil in Pascal, Cobol, Fortran, Basic, PL/I, Berlin u.a. 1985.

Laux, H.: Entscheidungstheorie, Band II: Erweiterung und Vertiefung, Berlin u.a. 1988.

Lehner, F.: Softwarewartung - Management, Organisation und methodische Unterstützung, München, Wien 1991.

Lientz, B.P., Swanson, E.B.: Problems in Application Software Maintenance, Communications of the ACM 24 (1981) 11, S. 763-769.

Maiocchi, M.: Reengineering: Can a program put intelligence in stupid programs?, in: 12th International Conference on Software Engineering, IEEE Computer Soc. Press, New York 1990, S. 123-124.

Nagel, K.: Nutzen der Informationsverarbeitung, München, Wien 1990.

Ott, H.J., Rausch, T.O.: Rechnet sich CASE für ein mittelständisches Unternehmen?, in: Schweiggert, F. (Hrsg.): Wirtschaftlichkeit von Software-Entwicklung und -Einsatz, Stuttgart 1992, S. 197-212.

Pomberger, G.: Methodik der Softwareentwicklung, in: Kurbel, K., Strunz, H. (Hrsg.): Handbuch Wirtschaftsinformatik, Stuttgart 1990, S. 215-236.

Rahm, E.: Synchronisation in Mehrrechner-Datenbanksystemen: Konzepte, Realisierungsformen und quantitative Bewertung, Berlin u.a. 1988.

Robillard, P.N., Coupal, D., Coallier, F.: Profiling Software Through the Use of Metrics, Software - Practice and Experience 21 (1991) 2, S. 507-518.

Rochester, J.B., Douglass, D.P.: Re-Engineering Existing Systems, I/S Analyzer 29 (1991) 10, S. 1-12.

Scherff, J.: Ermittlung der Wirtschaftlichkeit moderner Informations- und Kommunikationssysteme, HMD 23 (1986) 131, S. 3-15.

Schumann, M.: Wirtschaftlichkeitsbeurteilung für IV-Systeme, Wirtschaftsinformatik 35 (1993) 2, S. 167-178.

Siegwart, H., Singer, U.: Neues Verfahren für die Wirtschaftlichkeitsbeurteilung von Investitionen in neue Produktionstechnologien, Kostenrechnungspraxis krp (1991) 2, S. 63-70.

Sneed, H.: Software-Qualitätssicherung für kommerzielle Anwendungssysteme, Köln-Braunsfeld 1983.

Stucky, W., Krieger, R.: Datenbanksysteme, in: Kurbel, K., Strunz, H. (Hrsg.): Handbuch Wirtschaftsinformatik, Stuttgart 1990, S. 837-856.

Tempel, H.G.: Re-Engineering - ein vielversprechender Ansatz zur Investitionssicherung bei großen Software-Systemen, in: Schweiggert, F. (Hrsg.): Wirtschaftlichkeit von Software-Entwicklung und -Einsatz, Stuttgart 1992, S. 213-228.

Wedekind, H.: Grundbegriffe Verteilter Systeme aus der Sicht der Anwendung, it 30 (1988) 4, S. 263-271.

Wei, Y., Eldridge, K.: Module Completeness as a Useful Guideline for Programming, ACM SIGSOFT Software Engineering Notes 16 (1991) 1, p. 35.

Weingart, J.: Wirtschaftlichkeitsanalyse des Einsatzes der Informationstechnik in der Kommunalverwaltung, Diss., Hochschule für Verwaltungswissenschaften, Speyer 1987.

Zahn, E., Dogan, D.: Strategische Aspekte der Beurteilung von CIM-Installationen, CIM Management (1991) 3, S. 4-11.

Anhang: Zusammenhang zwischen den Wirtschaftlichkeitsfaktoren und ihren Determinanten

Wirtschaftlichkeitsfaktor wird beeinflußt von Determinante	Einarbeitungsaufwand	Änderungs-/Integrationsaufwand	Testaufwand	Dokumentationsaufwand
Verständlichkeit des Quelltextes	●	●	●	
Daten-Programm-Abhängigkeit		●		
Wartungszustand	●	●	●	
Verfügbarkeit/Qualität der Softwaredokumentation	●	●	●	●
Verwendung moderner Entwicklungsmethoden	●	●	●	
Restnutzungsdauer d. Hardware- u. Softwareplattform		●		
Umfang des Zielsystems		●		
Qualifikation, Erfahrung u. Motivation des Reengineering-Personals	●	●	●	●
Verfügbarkeit der Entwickler	●			
Verfügbarkeit von Wartungswerkzeugen	●	●	●	●
Verfügbarkeit von Werkzeugen für Daten-Reengineering	●			
Verfügbarkeit von Integrationswerkzeugen		●		
Verfügbarkeit von Testwerkzeugen			●	
Größe des IVAS	●	●	●	

Arbeitsberichte des Instituts für Wirtschaftsinformatik

Nr. 1 Bolte, Ch., Kurbel, K., Moazzami, M., Pietsch, W.: Erfahrungen bei der Entwicklung eines Informationssystems auf RDBMS- und 4GL-Basis; Februar 1991.

Nr. 2 Kurbel, K.: Das technologische Umfeld der Informationsverarbeitung - Ein subjektiver 'State of the Art'-Report über Hardware, Software und Paradigmen, März 1991.

Nr. 3 Kurbel, K.: CA-Techniken und CIM; Mai 1991.

Nr. 4 Nietsch, M., Nietsch, T., Rautenstrauch, C., Rinschede, M., Siedentopf, J.: Anforderungen mittelständischer Industriebetriebe an einen elektronischen Leitstand - Ergebnisse einer Untersuchung bei zwölf Untenehmen; Juli 1991.

Nr. 5 Becker, J., Prischmann, M.: Konnektionistische Modelle - Grundlagen und Konzepte; September 1991.

Nr. 6 Grob, H.L.: Ein produktivitätsorientierter Ansatz zur Evaluierung von Beratungserfolgen; September 1991.

Nr. 7 Becker, J.: CIM und Logistik; Oktober 1991.

Nr. 8 Burgholz, M., Kurbel, K., Nietsch, Th., Rautenstrauch, C.: Erfahrungen bei der Entwicklung und Portierung eines elektronischen Leitstands; Januar 1992.

Nr. 9 Becker, J., Prischmann, M.: Anwendung konnektionistischer Systeme; Februar 1992.

Nr. 10 Becker, J.: Computer Integrated Manufacturing aus Sicht der Betriebswirtschaftslehre und der Wirtschaftsinformatik; April 1992.

Nr. 11 Kurbel, K., Dornhoff, P.: A System for Case-Based Effort Estimation for Software-Development Projects; Juli 1992.

Nr. 12 Dornhoff, P.: Aufwandsplanung zur Unterstützung des Managements von Softwareentwicklungsprojekten; August 1992.

Nr. 13 Eicker, S., Schnieder, T.: Reengineering; August 1992.

Nr. 14 Erkelenz, F.: KVD2 - Ein integriertes wissensbasiertes Modul zur Bemessung von Krankenhausverweildauern - Problemstellung, Konzeption und Realisierung; Dezember 1992.

Nr. 15 Horster, B., Schneider, B., Siedentopf, J.: Kriterien zur Auswahl konnektionistischer Verfahren für betriebliche Probleme; März 1993.

Nr. 16 Jung, R.: Wirtschaftlichkeitsfaktoren beim integrationsorientierten Reengineering: Verteilungsarchitektur und Integrationsschritte aus ökonomischer Sicht; Juli 1993.