Universität Augsburg
Prof. Dr. Hans Ulrich Buhl
Kernkompetenzzentrum
Finanz- & Informationsmanagement
Lehrstuhl für BWL, Wirtschaftsinformatik,
Informations- & Finanzmanagement

Diskussionspapier WI-311

Softwaredokumentation unter ökonomischen Gesichtspunkten
Ein formal-deduktiver Ansatz

von

Peter Bartmann

erscheint in: Zeitschrift für Betriebswirtschaft (2012)

Universität Augsburg, 86135 Augsburg
Besucher: Universitätsstr. 12, 86159 Augsburg
Telefon: +49 821 598-4801 (Fax: -4899)
www.fim-online.eu

Titel: Technische Softwaredokumentation unter ökonomischen Gesichtspunkten

Untertitel: Ein formal-deduktiver Ansatz

Zusammenfassung:

Der Softwaredokumentation wird offensichtlich ein nur sehr geringer Stellenwert eingeräumt. So zeichnet sich beispielsweise ca. 80% der weltweiten Software durch eine unzureichende Dokumentation aus. Dabei bestätigen empirische Studien, dass mit der Verfügbarkeit einer adäquaten technischen Programmdokumentation eine Reduktion der Fehleranfälligkeit von nachträglichen Änderungen im Rahmen der Softwarewartung einhergeht. Für jedes Softwareprojekt ergibt sich demnach die Frage, in welcher Höhe eine Investition in die Erstellung einer technischen Programmdokumentation ökonomisch sinnvoll ist. Dieses Entscheidungsproblem wurde wissenschaftlich bisher stark vernachlässigt. Da entsprechende Zusammenhänge empirisch vermutlich nur schwer aufgedeckt werden können, stellt der Beitrag einen neuartigen deduktiven Ansatz vor, welcher die barwertigen Auszahlungen für die Erstellung einer technischen Programmdokumentation und für die Fehlerbehebung gegenüberstellt. Grundlage des Modells bildet das Phänomen der Fehlerausbreitung von einer Komponente über deren Abhängigkeiten zu angrenzenden Komponenten. Auf dieser Basis kann folgende Hypothese abgeleitet werden: Auch bei einer beliebig niedrigen Dokumentationsqualität oder einem beliebig hohen Kalkulationszinssatz ist es stets vorteilhaft, eher zu viel als vergleichsweise zu wenig zu dokumentieren. Diese Hypothese erweist sich dabei auch gegenüber der Änderung einiger Annahmen als robust.

Stichworte: Technische Programmdokumentation, Fehleranfälligkeit, Softwarewartung

Title: The economics of technical software documentation

Subtitle: A formal-deductive approach

Abstract:

Only a very low priority seems to be given to software documentation. For example, about 80% of the software systems in production worldwide are badly documented. In contrast empirical studies confirm that an adequate technical documentation reduces the error rate of subsequent changes during software maintenance. This leads to the question about the optimal scale of investment in technical documentation in any software project. In research, this decision problem has been largely neglected. Since underlying cause-effect relationships may be empirically hard to detect, this paper proposes a new deductive approach, which compares the present value of payments for the documentation and bug fixing. The basis of this model is the phenomenon of error propagation from one component to adjacent components through its dependencies. As a result the following hypothesis can be inferred: It is always beneficial to document too much rather than comparatively too little even for an arbitrarily low quality of documentation and for any required rate of interest. This hypothesis shows robustness even against some changes in assumptions.

Keywords: technical software documentation, fault proneness, software maintenance

1 Einleitung

Zahlreiche Unternehmen setzen IT zur Unterstützung ihrer Geschäftsprozesse ein. Da ein dynamisches Marktumfeld und sich ändernde Kundenanforderungen kontinuierliche Anpassungen der Geschäftsprozesse verlangen, ändern sich im Laufe des Betriebs ebenfalls die Anforderungen an die eingesetzte Software. Folglich müssen Softwaresysteme auch nach Inbetriebnahme im Rahmen der Softwarewartung laufend an neue Anforderungen angepasst werden. Dabei nimmt die Wartung von Softwaresystemen laut Zarnekow et al. (2004, S. 181) bereits bei einer Betriebsdauer von fünf Jahren knapp 80% der gesamten Lebenszykluskosten ein. Auch Krishnan et al. (2004, S. 396) beziffern die anteiligen Kosten für die Softwarewartung von in Betrieb befindlichen Altsystemen (*legacy systems*) auf 50% - 80% der gesamten Lebenszykluskosten. Die Softwarewartung stellt also einen erheblichen Kostenfaktor für Unternehmen und für die öffentliche Verwaltung dar.

Nachträgliche Modifikationen eines Softwaresystems im Rahmen der Wartung können fehlerhaft und inkonsistent zur ursprünglichen Struktur der Software sein (Krishnan et al. 2004, S. 396). Die Änderung einer Softwarekomponente kann dazu führen, dass angrenzende Komponenten nicht mehr fehlerfrei mit dieser interagieren. In der Folge sind zusätzliche Modifikationen in den angrenzenden Komponenten erforderlich, welche wiederum neue Änderungen bedingen und so weiter (Beszédes et al. 2007, S. 296). Inkonsistente Modifikationen sowie die Ausbreitung der dadurch entstehenden Fehler können demnach einen erheblichen Einfluss auf die Kosten für die Softwarewartung ausüben.

Ein wesentlicher Einflussfaktor auf Fehler und Inkonsistenzen durch nachträgliche Modifikationen ist das Fehlen einer adäquaten technischen Programmdokumentation (Dzidek et al. 2008, S. 424). Nachträgliche Änderungen von Softwaresystemen werden oftmals einige Monate oder sogar Jahre nach deren Entwicklung und in vielen Fällen von anderen Mitarbeitern als den eigentlichen Entwicklern umgesetzt. Daher nimmt die Analyse und das Verstehen der anzupassenden Software meist einen beträchtlichen Teil des Wartungsaufwands ein (Tan/Mookerjee 2005, S. 238). Für die Verständlichkeit von Softwaresystemen spielt nun die technische Programmdokumentation zum Beispiel in Form von externen Dokumenten, Quellcode-Kommentaren oder einer begreiflichen Bezeichnung von Methoden und Variablen eine unentbehrliche Rolle, da diese Informationen über den Zweck und die Ziele des vorliegenden Codes sowie über die dahinterliegenden Konzepte bereitstellen (Rajlich 2009, S. 4). So kann insbesondere das Fehlen einer adäquaten technischen Programmdokumentation zu einem schlechteren Verständnis der Software durch das Wartungspersonal führen und damit die Ursache für inkonsistente und fehlerhafte Modifikationen bilden (Dzidek et al. 2008, S. 424). Zusätzliche Investitionen in eine zweckmäßige technische Programmdokumentation während der Softwareentwicklung können dagegen zu einer besseren Verständlichkeit der Software beitragen und die Wahrscheinlichkeit für inkonsistente Modifikationen während der Wartung reduzieren.

Demgegenüber wird der Softwaredokumentation im Allgemeinen jedoch offenbar nur ein sehr geringer Stellenwert beigemessen. Softwareprojekte sind oftmals durch einen hohen Zeit- und Kostendruck gekennzeichnet und mit dem Versuch, entsprechende Projektziele einzuhalten oder deren Überschreitung zu reduzieren, werden während der Entwicklung häufig Abstriche bei der Dokumentation vorgenommen (Van Vliet 2008, S. 14). Eine Schätzung aus dem Jahr 2008 beziffert den Anteil schlecht dokumentierter Software weltweit auf ca. 80% (Van Vliet 2008, S. 466). Darüber hinaus haben sich in den letzten Jahren die Ansätze der agilen Softwareentwicklung wie *Scrum* in der Praxis etabliert, welche eine lauffähige Software höher gewichten als eine umfangreiche Dokumentation (Hruschka 2003, S. 398). Sie

folgen unter anderem der Maxime, dass zum Beispiel für die nachgelagerte Wartung, Anpassung und Weiterentwicklung der Software nur so viel wie nötig und so wenig wie möglich in die Dokumentation investiert wird. Mit derartigen Maximen versuchen die agilen Ansätze den bürokratischen Aufwand weitestgehend zu reduzieren und gelten aufgrund dessen als leichtgewichtig und flexibel. Dennoch scheint auch im Rahmen solcher Ansätze ein erheblicher Bedarf an einer technischen Programmdokumentation zu bestehen. Laut einer Umfrage gehören bei agilen Vorgehensmodellen unter anderem Quellcode-Kommentare zu den wichtigsten Informationsquellen für das Verständnis von zu wartender Software (de Souza et al. 2005, S. 73-74). Außerdem existieren Hinweise, dass das Fehlen einer adäquaten technischen Programmdokumentation ebenfalls im Rahmen agiler Entwicklungsmethoden das Verstehen einer Software erheblich erschweren und damit auch die Wahrscheinlichkeit für fehlerhafte Modifikationen erhöhen kann (Hanssen et al. 2009, S. 488).

Hinsichtlich der Dokumentation von Softwaresystemen besteht aus ökonomischer Sicht also ein Trade-off. Einerseits bedeutet die Erstellung einer umfassenderen technischen Programmdokumentation bei der Softwareentwicklung in der Regel höhere Auszahlungen während der Entwicklungsphase. Andererseits bestätigen jedoch mehrere wissenschaftliche Studien, dass die Verfügbarkeit einer adäquaten technischen Programmdokumentation insbesondere bei objektorientierten Softwaresystemen zu einer geringeren Fehleranfälligkeit von nachträglichen Modifikationen führt (Arisholm et al. 2006, S. 365, Dzidek et al. 2008, S. 424, Prechelt et al. 2002, S. 595). Folglich kann über die Bereitstellung einer zusätzlichen technischen Programmdokumentation auch eine Reduktion der Auszahlungen für die Behebung von Fehlern und Inkonsistenzen im Rahmen der Softwarewartung erreicht werden. Für jedes Softwareprojekt ergibt sich demnach die Frage, in welcher Höhe eine Investition in die Erstellung einer zweckmäßigen technischen Programmdokumentation ökonomisch sinnvoll ist. Dieses Entscheidungsproblem wurde wissenschaftlich bisher nur ungenügend behandelt. Darüber hinaus identifizieren empirische Studien neben der technischen Programmdokumentation mit der Anzahl der Abhängigkeiten einer Softwarekomponente zu anderen Komponenten einen weiteren Einflussfaktor auf die Fehleranfälligkeit von nachträglichen Änderungen (Briand et al. 2002, S. 714, Gyimóthy et al. 2005, S. 909, Singh et al. 2010, S. 31). Folglich können entsprechende Zusammenhänge, welche die unterschiedlichen Einflussfaktoren berücksichtigen und als Grundlage für eine befriedigende Beantwortung dieser Problemstellung dienen können, empirisch vermutlich nur schwer aufgedeckt werden. Vor diesem Hintergrund stellt der vorliegende Beitrag mittels eines formal-deduktiven Ansatzes die barwertigen Auszahlungen für die Erstellung einer technischen Programmdokumentation während der Entwicklung den barwertigen Auszahlungen für die Fehlerbehebung während der Softwarewartung gegenüber. Dabei liegt dem Modell unter anderem das Phänomen zugrunde, dass sich Fehler aufgrund der nachträglichen Modifikation einer Komponente auch auf angrenzende Komponenten über deren Abhängigkeiten ausbreiten können (Beszédes et al. 2007, S. 296). Ziel ist die Generierung neuer, empirisch prüfbarer Hypothesen im Sinne einer wissenschaftlichen Erklärung auf Basis der existierenden, überwiegend beschreibenden und empirisch geprägten Literatur.

Dazu wird in den folgenden beiden Abschnitten der Forschungsgegenstand abgegrenzt und die angewandte Forschungsmethodik begründet, bevor der vierte Abschnitt einen Überblick der relevanten Literatur gibt. Anschließend werden im fünften Abschnitt die grundlegenden Annahmen getroffen und das Modell aus diesen abgeleitet. Die Analyse des Modells erfolgt im sechsten Abschnitt. Nach einer kritischen Würdigung folgt eine Diskussion des theoretischen und praktischen Beitrags. Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick.

2 Abgrenzung des Forschungsgegenstands

Im Fokus des vorliegenden Beitrags stehen Anwendungssoftwaresysteme, für deren Implementierung höhere, insbesondere objektorientierte Programmiersprachen Verwendung finden. Derartige Softwaresysteme setzen sich in der Regel aus verschiedenen Komponenten wie beispielsweise Funktionen, Methoden, Klassen oder Modulen zusammen. Diese Komponenten stehen dabei über verschiedenartige Abhängigkeiten wie zum Beispiel Methodenaufrufe, Datenaustausch oder Cross-Cutting Concerns in Beziehung zueinander. Eben solche Abhängigkeiten spielen für das Verstehen und die Anpassung von Software im Rahmen der Wartung eine zentrale Rolle.

Grundsätzlich ist die Analyse und das Verstehen des existierenden Codes durch das Wartungspersonal vorausgehender Bestandteil aller nachträglichen Änderungen im Rahmen der Softwarewartung (Robillard et al. 2004, S. 889, Rostkowycz et al. 2004, S. 94). Bevor eine Änderung umgesetzt werden kann, müssen die zuständigen Entwickler die zu modifizierenden Stellen des Softwaresystems identifizieren und analysieren. Dazu muss eine abstrakt beschriebene Änderungsanforderung mit den relevanten Softwarekomponenten in Verbindung gebracht werden (*concept location*) (Rajlich 2009, S. 3-8, Rostkowycz et al. 2004, S. 94-95). Als einen der ersten Schritte sucht das Wartungspersonal beispielsweise über Stichwörter potenziell betroffene Komponenten des Softwaresystems, um diese bezüglich ihrer Funktionalität und ihrer Interaktionen mit angrenzenden Komponenten zu untersuchen und zu verstehen (*concept recognition* (Rajlich 2009, S. 3)). Anschließend navigieren die Entwickler zu einigen der angrenzenden Komponenten, welche wiederum bezüglich Funktionalität und Interaktionen analysiert sowie auf deren Relevanz für die umzusetzende Modifikation geprüft werden, bevor sie als Ausgangspunkt für die weitere Analyse dienen können (Ko et al. 2006, S. 982-983). Insbesondere im Rahmen dieser Analyse von Komponenten und deren Interaktionen sind zusätzliche Informationen über die im Code implementierten Funktionalitäten und Konzepte für die Verständlichkeit unentbehrlich (Rajlich 2009, S. 4). Diese können beispielsweise als zusätzliche Dokumente, Kommentare im Quellcode oder als eindeutige und begreifliche Namen von Methoden, Variablen etc. vorliegen. All diese Formen zusätzlicher Informationen werden fortan als *Dokumentation* bezeichnet. Dabei bezieht sich diese Definition ausschließlich auf die technische Programmdokumentation einer Software, welche für die Wartungsaktivitäten benötigt wird. Andere Formen der Softwaredokumentation wie beispielsweise die Benutzerdokumentation werden nicht betrachtet.

Besitzt das Wartungspersonal ein nicht ausreichendes Verständnis über die Funktionalität der Softwarekomponenten und über deren Abhängigkeiten, können Komponenten derart modifiziert werden, dass einige der angrenzenden Komponenten nicht mehr fehlerfrei mit der geänderten Komponente interagieren würden. Dies kann zusätzliche Änderungen in den angrenzenden Komponenten erfordern, welche wiederum neue Änderungen verlangen und so weiter (Beszédes et al. 2007, S. 296). Eine Situation, in der eine Komponente aufgrund der inkonsistenten Modifikation einer benachbarten Komponente mit dieser nicht mehr fehlerfrei interagieren würde und einer entsprechenden Änderung bedarf, wird im Folgenden als *Fehler* in der betroffenen Komponente bezeichnet. Diese Definition bezieht sich dabei ausschließlich auf logische oder semantische Fehler im Quellcode (*fault*, *bug*) (Radatz 1990), welche bei der Ausführung des Softwareprogramms die Ursache für Versagensfälle (*failure*) wie beispielsweise Laufzeitfehler oder ein nicht spezifikationskonformes Verhalten bilden können. Das Auftreten von Versagensfällen an sich sowie andere Fehlerarten wie zum Beispiel Fehler in der Bedienung werden nicht betrachtet. Nun können insbesondere schwer ersichtliche und ungenügend dokumentierte Abhängigkeiten wie beispielsweise Cross-Cutting Concerns die Verständlichkeit einer Software beträchtlich einschränken und die Ursache für solche Fehler

bilden (Eaddy et al. 2008, S. 498). Mit der Verfügbarkeit einer adäquaten technischen Programmdokumentation, welche Informationen über die Funktionalitäten und Interaktionen von Komponenten bereitstellt, kann sich die Wahrscheinlichkeit für derartige Fehler erheblich reduzieren (Dzidek et al. 2008, S. 417). Allerdings bedeutet das Bereitstellen einer zusätzlichen technischen Programmdokumentation einen höheren Aufwand für deren Erstellung. Dieser Trade-off bildet den Forschungsgegenstand des vorliegenden Beitrags.

Als Grundlage für das vorgestellte Modell wird der Lebenszyklus einer Software grob in die Phase der initialen Entwicklung und in die nachgelagerte Wartungsphase unterteilt. Vereinfachend wird im Folgenden die Erstellung der technischen Programmdokumentation auf die Entwicklungsphase beschränkt. Bei einer Dokumentation während der Entwicklung wird unterstellt, dass nur ein vernachlässigbar geringer Aufwand für das Verstehen der Software erforderlich ist. Dagegen bedeutet eine nachträgliche Dokumentation im Rahmen der Softwarewartung einen zumindest einmal notwendigen und mit nachträglichen Modifikationen vergleichbaren Aufwand für die Analyse und das Verstehen der existierenden Software (Rostkowycz et al. 2004, S. 92). Eben dieser Analyseprozess kann jedoch mittels einer zuvor bereitgestellten technischen Programmdokumentation unterstützt werden. Daher wird der Fokus dieses Beitrags auf den Effekt der technischen Programmdokumentation, welche während der Entwicklung erstellt wird und damit bereits zu Beginn der Wartungsphase zur Verfügung steht, auf die Fehleranfälligkeit von Modifikationen gelegt und eine nachträgliche Dokumentation während der Wartung nicht betrachtet.

Eine scharfe Trennung zwischen Entwicklung und Wartung für ein gesamtes Softwaresystem dürfte nur bei sequenziellen Vorgehensmodellen wie dem Wasserfallmodell gelingen (Rajlich 2006, S. 68). Iterative, inkrementelle und agile Vorgehensmodelle untergliedern die Softwareentwicklung gewöhnlich in kurze, sich wiederholende Zyklen, womit die Entwicklung und die Wartung eines gesamten Softwaresystems zumindest teilweise ineinandergreifen. Daher wird der Fokus dieses Beitrags primär auf die sequenzielle Softwareentwicklung beschränkt, welche eine klare Trennung zwischen Entwicklung und Wartung erlaubt. Dennoch erscheint eine Untersuchung der Zusammenhänge zwischen der technischen Programmdokumentation und der Fehleranfälligkeit von nachgelagerten Modifikationen im Rahmen iterativer Vorgehensmodelle lohnenswert. Nachfolgende Iterationen haben eine Erweiterung der in vorausgehenden Iterationen entwickelten Softwarekomponenten zum Gegenstand, welche auch eine Modifikation von bereits existierenden Komponenten erfordern kann. Dabei kann für nachgelagerte Modifikationen insbesondere bei größeren Softwaresystemen zumindest in einem gewissen Umfang ebenfalls eine entsprechende Analyse des zu modifizierenden Codes notwendig sein (Rajlich 2006, S. 69-70), welche durch die Bereitstellung einer adäquaten technischen Programmdokumentation unterstützt werden kann. Eine Erweiterung des vorgestellten Modells, welche die Spezifika iterativer Vorgehensmodelle adressiert, wird in den Fokus zukünftiger Forschung gerückt.

Da die Erstellung der technischen Programmdokumentation und die Fehlerbehebung zu verschiedenen Zeitpunkten anfallen, werden für die Untersuchung des oben beschriebenen Trade-off die Barwerte der Auszahlungen für die Dokumentation während der Entwicklung und für die Fehlerbehebung während der Softwarewartung gegenübergestellt. Damit bezieht sich der vorliegende Beitrag primär auf kommerzielle Software und vernachlässigt Open Source Software. Dennoch scheint auch für diese ein hoher Bedarf an technischer Programmdokumentation zu bestehen (Sharif/Buckley 2008, S. 6). Das vorgestellte Modell wäre jedoch nur mit entsprechenden Änderungen, beispielsweise dass statt den Auszahlungen die für die Dokumentation und Fehlerbehebung erforderliche Zeit betrachtet wird, auf Open Source Software anwendbar.

Darüber hinaus beschränkt sich der Fokus auf Unternehmen, die sowohl die Entwicklung als auch die Wartung eines Softwaresystems durchführen. An dieser Stelle sei darauf hingewiesen, dass das vorgestellte Modell nur für diesen spezifischen Fall Gültigkeit besitzt und eine Verallgemeinerung der Ergebnisse ohne eine entsprechende Modellerweiterung unzulässig ist. Dieser Fokus schließt einerseits alle Fälle ein, in denen Unternehmen Individualsoftware für die Eigenverwendung oder für Kunden entwickeln und im Rahmen des Betriebs auch warten. Andererseits betrifft dies auch die Anbieter von Standardsoftware, da diese in der Regel sowohl für die Entwicklung als auch für die Wartung und Weiterentwicklung verantwortlich sind. Nicht betrachtet werden dagegen alle Situationen wie zum Beispiel das Outsourcing der Wartung an spezielle Dienstleister (Sneed 2008, S. 1-4) oder das Application Outsourcing (Riedl/Kepler 2003, S. 8), in denen die Entwicklung und die Wartung einer Software von zwei unterschiedlichen Parteien durchgeführt werden. Der Umstand, dass im Jahr 2010 gegenüber dem weltweiten Investitionsvolumen in Software in Höhe von 395 Mrd. US-Dollar (Statista 2010b) lediglich 27 Mrd. US-Dollar für Application Outsourcing ausgegeben wurden (Statista 2010a), darf an dieser Stelle als Indiz für die im Vergleich vermutlich höhere ökonomische Relevanz der betrachteten Fälle angeführt werden.

3 Forschungsmethodik

Folgt man dem Forschungsrahmen für den Bereich des Operations Management von Meredith et al. (1989, S. 301-304), sollen sich alle Forschungsaktivitäten in einen kontinuierlichen, sich wiederholenden Forschungskreislauf aus Beschreibung (*description*), Erklärung (*explanation*) und Prüfung (*testing*) eingliedern. Dabei geht die Beschreibung den erklärenden und prüfenden Forschungsarbeiten voraus und versucht, noch nicht betrachtete Begebenheiten zu untersuchen oder bestimmte Aspekte existierender Forschungsgegenstände im Detail zu beleuchten. Darauf aufbauend sollen Beiträge die zugrunde liegenden Zusammenhänge und Kausalitäten ableiten, welche die beschriebenen Forschungsgegenstände und Begebenheiten erklären können. Neben der reinen Erklärung von Phänomenen kann dabei auch der Anspruch auf die Entwicklung von Gestaltungsempfehlungen im Sinne normativer Ansätze erhoben werden. Die anschließende Prüfung der erklärenden Beiträge erfolgt durch Untersuchungen, inwieweit diese Beiträge entsprechende Phänomene aus der Realität zuverlässig prognostizieren können (*prediction*). Die dabei gewonnen Erkenntnisse bilden die Basis für die schrittweise Verbesserung der erklärenden Beiträge sowie für bessere Handlungsempfehlungen zur Lösung der Probleme in der Praxis.

Wie bereits erwähnt und im folgenden Abschnitt detailliert dargestellt kann auf Basis der existierenden Literatur aus den Bereichen Softwarewartung und *Program Comprehension* die Verfügbarkeit einer adäquaten technischen Programmdokumentation als Einflussfaktor auf die Fehleranfälligkeit von nachgelagerten Änderungen einer Software angenommen werden. Vor diesem Hintergrund muss für jedes Softwareprojekt aus ökonomischer Sicht entschieden werden, wie viel während der Entwicklung in die Erstellung einer zweckmäßigen technischen Programmdokumentation investiert wird, um die Anzahl der Fehler während der Softwarewartung und damit die Auszahlungen für deren Behebung zu reduzieren. Die im Rahmen der bisherigen Literatur bestätigten Zusammenhänge genügen jedoch nicht für eine befriedigende Beantwortung dieser Problemstellung. Dazu müssen die Auszahlungen für die Erstellung einer technischen Programmdokumentation mit den Auszahlungen für die Fehlerbehebung in Verbindung gebracht werden. Hierbei gilt es, auch andere Einflussfaktoren auf die Anzahl der durch nachträgliche Änderungen eingeführten Fehler zu berücksichtigen. So bestätigen Studien, dass Softwarekomponenten mit einer größeren Anzahl von Abhängigkeiten zu anderen Komponenten auch eine höhere Fehleranfälligkeit im Rahmen der Softwarewartung aufwei-

sen (Briand et al. 2002, S. 714, Gyimóthy et al. 2005, S. 909, Singh et al. 2010, S. 31). Ziel des vorliegenden Beitrags ist somit die Ableitung neuer Zusammenhänge im Sinne einer wissenschaftlichen Erklärung, welche die Grundlage für die Lösung der erläuterten Problemstellung bilden können. Da diese vermutlich nicht unmittelbar aus Beobachtungen in der Realität ableitbar sein dürften, eignet sich hierzu ein deduktiver Ansatz.

Gemäß dem Forschungskreislauf von Meredith et al. (1989, S. 301-304) müssen sich solch deduktiv abgeleitete Zusammenhänge grundsätzlich einer kritischen Prüfung unterziehen, inwieweit sie die erklärten Phänomene prognostizieren können. Dabei können insbesondere formale Modelle beispielsweise empirisch gewonnene Erkenntnisse in eine vom Betrachter unabhängige Form überführen, welche sich für diese nachgelagerte Prüfung der abgebildeten Zusammenhänge eignet (Meredith et al. 1989, S. 308). Ziel ist daher die Entwicklung eines formalen Modells, welches in der existierenden Literatur bisher ungenügend betrachtete aber entscheidungsrelevante Zusammenhänge abbildet. Im Vordergrund steht dabei jedoch weniger die Bereitstellung eines unmittelbar in der Praxis anwendbaren Modells, sondern vielmehr die Deduktion neuer, empirisch prüfbarer Hypothesen. Dennoch erheben die abgeleiteten Zusammenhänge den Anspruch, nach einer entsprechenden Prüfung im Rahmen zukünftiger Forschungsarbeiten und bei einer hinreichend starken Bestätigung als Grundlage für normative Aussagen beziehungsweise Handlungsempfehlungen für die Praxis dienen zu können.

4 Überblick der existierenden Literatur

Der Zusammenhang zwischen der Verfügbarkeit einer adäquaten technischen Programmdokumentation und der Fehleranfälligkeit von nachträglichen Änderungen im Rahmen der Softwarewartung ist bereits durch mehrere Studien empirisch bestätigt. So zeigen Prechelt et al. (2002), dass mit einer expliziten Dokumentation der in einer Software verwendeten Entwurfsmuster (*design patterns*) eine Reduktion der Fehleranfälligkeit von nachträglichen Modifikationen sowie der dafür benötigten Zeit einhergeht. Für eine technische Programmdokumentation in Form von UML-Diagrammen stellen Arisholm et al. (2006) einen analogen Zusammenhang fest. Einen signifikanten Effekt der technischen Programmdokumentation auf die Zeit, welche für die Umsetzung der Änderungen zuzüglich der nachträglichen Anpassung der UML-Diagramme benötigt wird, können sie jedoch nicht bestätigen. Hingegen zeigen Rostkowycz et al. (2004), dass eine nachträgliche, inkrementell während der Softwarewartung erstellte Programmdokumentation zu einer Reduzierung des Wartungsaufwands führt und sich dadurch langfristig auszahlen kann. Dzidek et al. (2008) betrachten ebenfalls eine technische Programmdokumentation in Form von UML-Diagrammen und deren Effekte auf die Softwarewartung. Interessanterweise führt die Verwendung der technischen Programmdokumentation zu einer Verminderung insbesondere der Fehler, welche auf ein Unverständnis existierender Interaktionen von verschiedenen Softwarekomponenten beruhen.

Darüber hinaus bestätigen zahlreiche Studien ebenfalls einen Zusammenhang zwischen der Anzahl der Abhängigkeiten einer Softwarekomponente zu angrenzenden Komponenten und deren Fehleranfälligkeit. So berichten beispielsweise Aggarwal et al. (2009), Singh et al. (2010), Subramanyam/Krishnan (2003) und Yu et al. (2002), dass bei Klassen mit einer höheren Anzahl an gekoppelten Klassen auch eine höhere Fehleranfälligkeit beobachtbar ist. Olague et al. (2007) können diesen Zusammenhang bestätigen, weisen jedoch für die Komplexität einer Klasse einen noch stärkeren Zusammenhang zu deren Fehleranfälligkeit nach. Dagegen betonen Briand et al. (2002), Gyimóthy et al. (2005) und Binkley/Schach (1998), dass gemäß deren Ergebnissen die jeweils betrachtete Kopplungsmetrik zu den besten Indikatoren für die Fehleranfälligkeit einer Klasse gehört. Eaddy et al. (2008) betrachten Cross-Cutting

Concerns als spezielle Art von Abhängigkeiten und bestätigen auch für diese einen Zusammenhang mit der Fehleranfälligkeit.

Bezüglich anderer Einflussfaktoren auf die Fehleranfälligkeit finden sich in der existierenden Literatur widersprüchliche Aussagen. Während beispielsweise Aggarwal et al. (2009), Subramanyam/Krishnan (2003) und Singh et al. (2010) einen Zusammenhang zwischen der Größe von Klassen und deren Fehleranfälligkeit aufzeigen, können Eaddy et al. (2008) und Fenton/Ohlsson (2000) diesen Zusammenhang nicht bestätigen. Auch bezüglich der Komplexität von Klassen besteht Uneinigkeit. Laut Olague et al. (2007) und Singh et al. (2010) sind Komplexitätsmetriken gute Indikatoren für die Fehleranfälligkeit einer Klasse. Zimmermann/Nagappan (2008) und Fenton/Ohlsson (2000) können dagegen keinen signifikanten Effekt der Komplexität einer Klasse auf deren Fehleranfälligkeit zeigen.

Zusammenfassend kann also festgehalten werden, dass die Größe und Komplexität einer Softwarekomponente als Einflussfaktoren auf deren Fehleranfälligkeit empirisch nicht zweifelsfrei bestätigt sind. Gesichert sind dagegen der Zusammenhang zwischen der Fehleranfälligkeit einer Softwarekomponente und der Anzahl ihrer Abhängigkeiten zu angrenzenden Komponenten sowie ein Zusammenhang zwischen der Verfügbarkeit einer adäquaten technischen Programmdokumentation und der Fehleranfälligkeit nachträglicher Änderungen.

Hinweise, warum die Bereitstellung einer zusätzlichen technischen Programmdokumentation zu einer Reduzierung von inkonsistenten Modifikationen im Rahmen der Softwarewartung führt, können Beiträge aus dem Bereich *Program Comprehension* liefern. So beschreibt Rajlich (2009) die sechs fundamentalen Vorgänge des Verstehens von Softwaresystemen. Angelehnt an die entsprechenden Theorien aus der Linguistik und Mathematik charakterisiert er dazu ein Code-Fragment über seine *Extension* (der tatsächliche Code; z.B. $y = x * 0{,}19$), seine *Intension* (der Zweck des Codes bzw. die darin implementierte, abstrakte Funktionalität; z.B. die Berechnung der in einem Bruttopreis x enthaltenen Mehrwertsteuer y) und über den zugeordneten *Namen* (z.B. „Mehrwertsteuerberechnung“). Die sechs Vorgänge des Verstehens von Software beschreiben dabei die gegenseitige, paarweise Zuordnung dieser drei Merkmale. Während beispielsweise der Vorgang *concept location* die Lokalisation der Code-Fragmente, welche eine gegebene Funktionalität implementieren, bezeichnet (Identifikation der entsprechenden Extension zu einer gegebenen Intension), entspricht der Vorgang *concept recognition* dem Erkennen des Zwecks beziehungsweise der abstrakten Funktionalität eines gegebenen Code-Fragments (Identifikation der zu einer gegebenen Extension gehörigen Intension). Analog dazu definiert Rajlich (2009, S. 3) die Vorgänge *naming* (Zuordnung eines Namens zu einer gegebenen Intension), *definition* (Identifikation der einem gegebenen Namen zugehörigen Intension), *annotation* (Zuordnung eines Namens zu einer gegebenen Extension) und *traceability* (Identifikation der einem gegebenen Namen zugehörigen Extension). Das Verstehen eines Code-Fragments bedeutet, dass das Wartungspersonal alle sechs Vorgänge bei diesem Fragment vollziehen kann. Auf dieser Basis begründet Rajlich (2009, S. 4) die zentrale Rolle der technischen Programmdokumentation für das Verstehen und die nachträgliche Modifikation von Software. Eine zweckmäßige technische Programmdokumentation beispielsweise in Form von begreiflichen Bezeichnern, Quellcode-Kommentaren oder separaten Dokumenten stellt Informationen über den Zweck von Code-Fragmenten und den darin implementierten Funktionalitäten (Intension) sowie die zugeordneten Namen bereit und macht damit den Code (Extension) erst verständlich. Für die Umsetzung einer Änderung muss das Wartungspersonal die betroffenen Funktionalitäten aus der abstrakten Änderungsanforderung selektieren und beispielsweise über die zugeordneten Namen die entsprechenden, zu modifizierenden Code-Fragmente lokalisieren. Dabei ist auch umgekehrt die Analyse verschiedener Code-Fragmente auf die darin implementierten Funktionalitäten und deren Interaktionen er-

forderlich. Eben diese Vorgänge des Verstehens können durch die Verfügbarkeit einer adäquaten technischen Programmdokumentation erheblich unterstützt werden.

Als Ergebnis ihrer explorativen Studie beschreiben Ko et al. (2006), wie Entwickler unbekannten Code für die Umsetzung nachträglicher Änderungen im Rahmen der Softwarewartung analysieren. Nachdem sie ein potenziell relevantes Code-Fragment identifiziert haben, navigieren die Entwickler über deren Abhängigkeiten zu angrenzenden Code-Fragmenten, um die Funktionalität des ursprünglichen Fragments sowie deren Interaktionen mit der Umgebung zu verstehen und um gegebenenfalls weitere betroffene Code-Stellen zu suchen. Diese Navigation nimmt dabei durchschnittlich 35% der für die Umsetzung einer Änderung benötigten Zeit ein und kann durch eine adäquate technische Programmdokumentation, welche zusätzliche Informationen über den Zweck und die Verwendung des Codes bereitstellt, unterstützt werden. Ebenfalls im Rahmen einer explorativen Studie arbeiten Robillard et al. (2004) verschiedene Charakteristika der Analyse von Softwaresystemen heraus, bezüglich derer sich bei der Umsetzung von nachträglichen Änderungen erfolgreiche von nicht erfolgreichen Entwicklern unterscheiden. So nehmen insbesondere erfolglose Entwickler für die Umsetzung einer Änderungsanforderung Modifikationen an lediglich einer Stelle im Code vor, obwohl die Modifikation mehrerer Code-Fragmente für eine konsistente und dem existierenden Design besser entsprechende Änderung erforderlich gewesen wäre. Zudem orientieren sich erfolgreiche Entwickler bei der Suche nach relevanten Code-Fragmenten an der Struktur der Software, also unter anderem an den Abhängigkeiten zwischen verschiedenen Code-Fragmenten, und vermeiden eine unstrukturierte Inspektion des Codes. Als weitere Beobachtung beschreiben die Autoren, dass für eine Änderung relevante Code-Fragmente oftmals nicht als solche erkannt werden, sofern die Entwickler diese Code-Fragmente nicht gezielt inspizieren, sondern zufällig oder versehentlich darauf stoßen. Diese Beobachtungen erlauben die Schlussfolgerung, dass insbesondere das solide Verständnis von Abhängigkeiten zwischen verschiedenen Softwarekomponenten eine wichtige Rolle für die Vermeidung inkonsistenter Modifikationen im Rahmen der Softwarewartung spielt, und dass eine adäquate technische Programmdokumentation das Verstehen dieser Abhängigkeiten erheblich unterstützen kann.

Ferner existieren zahlreiche Beiträge, die verschiedene Ansätze zur Prognose von Fehlerhäufigkeiten (*fault*), des Auftretens von Versagensfällen (*failure*) oder allgemeiner von Softwarezuverlässigkeit vorschlagen. Grottke (2005) stellt einen umfassenden Überblick der existierenden Ansätze zur Prognose von Softwarezuverlässigkeit dar. Dabei bilden sogenannte Softwarezuverlässigkeitswachstumsmodelle die bedeutendste Modellklasse. Derartige Modelle berechnen aus dem Verlauf der während der Testphase aufgetretenen Versagensfälle eine Prognose für die Zuverlässigkeit während des Betriebs. Die auftretenden Versagensfälle werden dabei mithilfe stochastischer Prozesse modelliert. Als weitere Klasse werden statistische Modelle dargestellt, welche die aktuelle Fehleranzahl oder die Zuverlässigkeit einer Software mittels Stichproben schätzen. Eine dritte Klasse von Modellen versucht den Fehlergehalt der Software auf Basis von Informationen über das Softwareprodukt und seine Erstellung vorherzusagen. Erstaunlicherweise berücksichtigt keiner der vorgestellten Ansätze die Verfügbarkeit einer technischen Programmdokumentation als Einflussfaktor auf die Anzahl der Fehler aufgrund nachträglicher Änderungen. Zwar zeigen beispielsweise Takahashi/Kamayachi (1989, S. 83-84) einen signifikanten Zusammenhang zwischen dem Umfang der Designdokumentation und der Anzahl der Fehler, welche zu Beginn der Testphase während der Entwicklung in der Software verbleiben. Auf dieser Basis schlägt der Beitrag ein lineares Regressionsmodell zur Prognose der Fehleranzahl vor. Jedoch liegt der Fokus dabei nicht auf dem Einbau von Fehlern während der Softwarewartung.

Die Recherche nach Beiträgen, welche ökonomische Modelle für die Berechnung der Kosten zur Vermeidung von Fehlern im Rahmen der Softwarewartung beinhalten, führte zu keinem

Ergebnis. Beiträge mit entsprechenden Modellen aus anderen Kontexten, welche als Grundlage für eine Adaption in den Kontext der Softwarewartung dienen können, brachte die Suche ebenfalls nicht hervor. An dieser Stelle setzt der vorliegende Beitrag an und stellt ein neu entwickeltes, formal-deduktives Modell vor, welches die barwertigen Auszahlungen für die Erstellung einer technischen Programmdokumentation und für die Fehlerbehebung gegenüberstellt sowie die Generierung neuer Hypothesen ermöglicht.

5 Modell

5.1 Grundlegende Annahmen

Im Folgenden wird die Größe einer Software SW in der Anzahl ihrer Komponenten gemessen. Eine Softwarekomponente kann dabei als Funktion, Methode, Klasse, Modul oder ähnliches verstanden werden. Diese Komponenten stehen über gegenseitige Abhängigkeiten wie beispielsweise Methodenaufrufe, verschiedene Formen des Datenaustausches, Schnittstellen oder Cross-Cutting Concerns in Beziehung zueinander. Jede Komponente, die über derartige Abhängigkeiten eine Verbindung zu einer bestimmten Komponente besitzt, wird als deren *angrenzende Komponente* bezeichnet. Da auf Basis der bisherigen Literatur ein Zusammenhang zwischen der Fehleranfälligkeit einer Komponente und der Anzahl ihrer angrenzenden Komponenten gesichert erscheint (Briand et al. 2002, S. 714, Gyimóthy et al. 2005, S. 909, Singh et al. 2010, S. 31) und sich Fehler bei nachträglichen Änderungen insbesondere über Abhängigkeiten auf angrenzende Komponenten ausbreiten können (Beszédes et al. 2007, S. 296), dient im Folgenden die Anzahl der angrenzenden Komponenten einer Komponente als Maß der strukturellen Komplexität von SW. In der Realität dürften sich verschiedene Komponenten in der Anzahl ihrer angrenzenden Komponenten deutlich voneinander unterscheiden. Dennoch wird im Folgenden vereinfachend angenommen, dass jede Komponente von SW gleich viele angrenzende Komponenten besitzt. Dabei sollte für diesen Wert die durchschnittliche Anzahl der angrenzenden Komponenten pro Komponente gewählt werden. Vergleichbar zu MacCormack et al. (2006, S. 1020) kann dieser Wert als Maß für die durchschnittliche Anzahl der von der Modifikation einer Komponente potenziell direkt betroffenen Stellen in SW dienen. An dieser Stelle sei zudem vorweggenommen, dass der konkrete Wert keine Bedeutung für die im Rahmen der Modellanalyse deduzierte Hypothese besitzt.

Im Gegensatz zur Anzahl der Abhängigkeiten ist der Einfluss der Größe und Komplexität einer Komponente auf deren Fehleranfälligkeit nicht gesichert. So können beispielsweise Fenton/Ohlsson (2000, S. 812) entsprechende Zusammenhänge nicht bestätigen. Daher werden diese im Folgenden vernachlässigt.

(A1) Die Entwicklung von SW findet in $t = 0$ *statt. Die Größe von SW wird in der Anzahl ihrer Komponenten* m $(m \in \mathbb{N} | m \geq 2)$ *gemessen. Die (durchschnittliche) Anzahl der angrenzenden Komponenten, zu denen eine Abhängigkeit besteht, ist für jede Komponente gleich und wird als* a $(a \in \mathbb{R}^{+} | a \geq 1)$ *bezeichnet. Die Größe und Komplexität der Komponenten selbst werden nicht berücksichtigt.*

Gemäß Dzidek et al. (2008, S. 417) geht mit der Verfügbarkeit einer adäquaten technischen Programmdokumentation insbesondere eine Reduktion der Fehler einher, die auf ein Unverständnis der Abhängigkeiten verschiedener Softwarekomponenten beruhen. Zudem zeigen Eaddy et al. (2008, S. 498), dass Cross-Cutting Concerns als schwer ersichtliche und meist ungenügend dokumentierte Abhängigkeiten die Verständlichkeit einer Software erheblich einschränken und die Ursache für Fehler bilden können. Daher wird im Folgenden die Dokumentation von Abhängigkeiten zwischen jeweils zwei Komponenten betrachtet. Diese wird

als technische Programmdokumentation mit zusätzlichen Informationen sowohl über den Zweck und die Funktionalitäten der relevanten Code-Fragmente in beiden Komponenten als auch über deren Abhängigkeit beziehungsweise deren Interaktion verstanden. Die *Dokumentation einer Abhängigkeit* gibt also Aufschluss darüber, welche Funktionalität(en) zwei Code-Fragmente in unterschiedlichen Komponenten umsetzen und wie diese Code-Fragmente in ihrer Funktionalität zusammenwirken. Sie kann dabei zum Beispiel in Form von Quellcode-Kommentaren, Diagrammen, externen Dokumenten oder einer begreiflichen Bezeichnung von Variablen, Methoden, Klassen etc. vorliegen.

Weiterhin wird angenommen, dass die Dokumentation einer Abhängigkeit eine Auszahlung in stets gleicher Höhe bedeutet. In der Realität dürften sich die Auszahlungen für die Dokumentation einer Abhängigkeit je nach Beschaffenheit der Abhängigkeiten sowie der relevanten Code-Fragmente in den jeweils angrenzenden Komponenten voneinander unterscheiden. Auch der Umfang und die Qualität der zu erstellenden technischen Programmdokumentation werden sich vermutlich auf die Höhe der Auszahlungen auswirken. Daher sollte für diese Auszahlungen ein durchschnittlicher Wert verwendet werden. Wie im sechsten Abschnitt gezeigt werden kann, spielt auch dieser konkrete Wert für die im Rahmen der Modellanalyse abgeleitete Hypothese keine Rolle. Da die technische Programmdokumentation zur Verbesserung der Verständlichkeit von SW zusätzliche Informationen über die in den relevanten Code-Fragmenten realisierten Funktionalitäten (Rajlich 2009, S. 4) sowie über deren Zusammenwirken bereitstellen soll, wird zudem ein linearer Zusammenhang zwischen der Anzahl der dokumentierten Abhängigkeiten und den dafür anfallenden Auszahlungen angenommen.

Wie bereits in Abschnitt 2 ausgeführt wird die Dokumentation auf die Entwicklung von SW beschränkt und eine nachträgliche Dokumentation während der Wartung nicht betrachtet.

(A2) Für jede Abhängigkeit zwischen zwei angrenzenden Komponenten können zusätzliche Informationen über die Funktionalitäten der jeweils relevanten Code-Fragmente in den beiden Komponenten sowie über deren Zusammenwirken bereitgestellt werden. Die Dokumentation einer Abhängigkeit führt in $t = 0$ *zu einer (durchschnittlichen) Auszahlung in Höhe von* c_d $(c_d \in \mathbb{R}^+)$. *Der relative Anteil der dokumentierten Abhängigkeiten wird als Dokumentationsgrad* r_d $(r_d \in \mathbb{R} | 0 \leq r_d \leq 1)$ *bezeichnet. Die dokumentierten Abhängigkeiten verteilen sich gleichmäßig auf alle Komponenten in SW.*

Die Softwarewartung wird auf den Zeitraum nach Fertigstellung und Inbetriebnahme von SW beschränkt. Die Anpassung an geänderte Anforderungen erfolgt dabei ausschließlich mittels Modifikationen an einem bestimmten, gleichbleibenden Anteil der existierenden Komponenten in jeder Periode während des Betriebs. In der Realität wird zur Anpassung einer Software an alle sich ändernden Anforderungen die Modifikation eines pro Periode jeweils unterschiedlichen Anteils der existierenden Komponenten erforderlich sein. Jedoch unterliegt die Softwarewartung oftmals einer festen Budgetrestriktion (Krishnan et al. 2004, S. 396) oder einem Unternehmen steht nur eine feste Anzahl von Mitarbeitern für die entsprechenden Änderungen zur Verfügung. Daher können in der Regel auch nicht alle neuen Anforderungen umgesetzt werden. Aufgrund solcher Restriktionen ist meist nur die Änderung eines pro Periode jeweils ähnlich großen Anteils der vorhandenen Komponenten realisierbar. Daher erscheint eine solche Annahme vertretbar. Da sich dieser Anteil dennoch in einem gewissen Umfang von Periode zu Periode unterscheiden wird, sollte bei der Anwendung des Modells ein durchschnittlicher Wert verwendet werden.

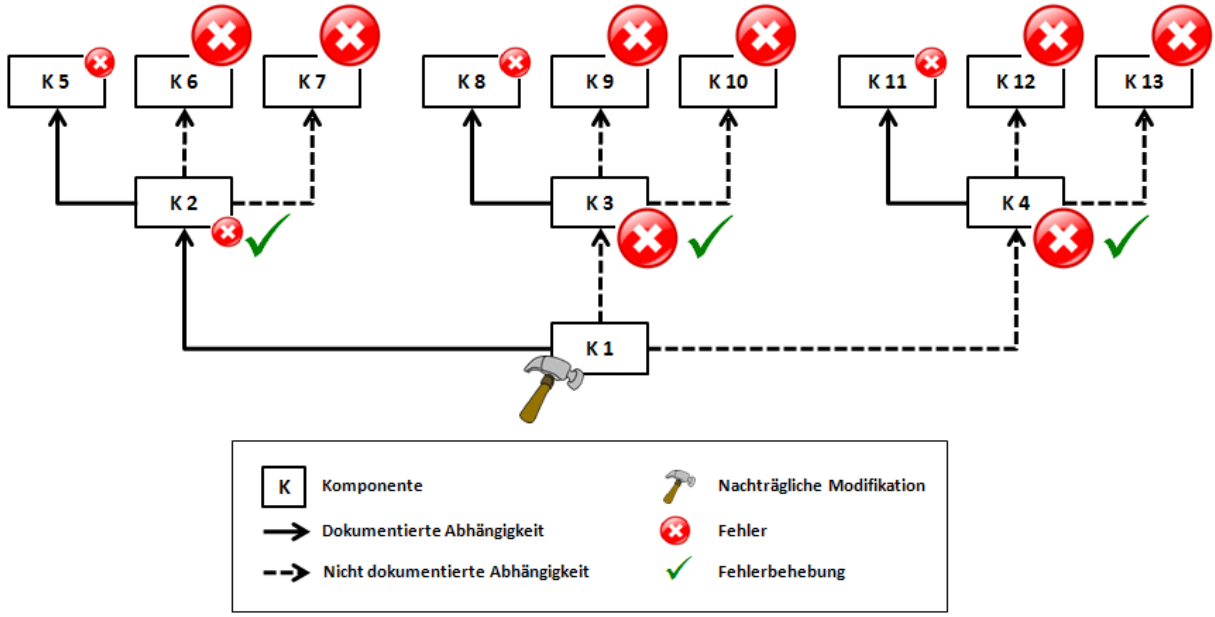
Außerdem erfolgt die Anpassung einer Software in der Praxis oftmals nicht nur über die Änderung bestehender Komponenten, sondern auch über entsprechende Erweiterungen (Krishnan et al. 2004, S. 396). Als Vereinfachung beschränkt sich das vorliegende Modell jedoch ausschließlich auf die Auswirkungen des Dokumentationsgrads auf die Fehleranfälligkeit von

Änderungen der existierenden Komponenten. Dementsprechend werden Erweiterungen von SW um neue Komponenten oder um zusätzliche Abhängigkeiten vernachlässigt. Wie im Rahmen der kritischen Würdigung diskutiert, dürften solche Erweiterungen von SW die Fehleranfälligkeit von nachgelagerten Wartungsmaßnahmen jedoch sogar noch erhöhen.

(A3) Der Betrieb und die Wartung von SW erfolgen von $t = 1$ *bis einschließlich* $t = n$ $(n \in \mathbb{N})$. *Während des Betriebs wird SW jeweils zu Beginn einer jeden Periode an geänderte Anforderungen angepasst. Dies geschieht pro Periode ausschließlich über die Änderung eines jeweils gleichen (durchschnittlichen) Anteils* r_c $(r_c \in \mathbb{R}^+)$ *der existierenden Komponenten. Die Anzahl der Komponenten* m *und die Anzahl der Abhängigkeiten* a *bleiben während der Wartung konstant.*

Abb. 1 veranschaulicht den Zusammenhang, der für die Entstehung von Fehlern aufgrund der Modifikation einer Komponente zur Anpassung an geänderte Anforderungen angenommen wird. Dabei wird insbesondere der Effekt berücksichtigt, dass die Modifikation einer Komponente inkonsistent zu den angrenzenden Komponenten sein kann. Aufgrund der daraus resultierenden Fehler können Änderungen in den angrenzenden Komponenten erforderlich sein, welche wiederum neue Änderungen bedingen und so weiter (Beszédes et al. 2007, S. 296).

Abb. 1 Entstehung und Ausbreitung von Fehlern aufgrund nachträglicher Änderungen



In diesem Beispiel besitzt jede Komponente drei angrenzende Komponenten, wobei jeweils die Abhängigkeit zu einer angrenzenden Komponente dokumentiert ist und die Abhängigkeiten zu den beiden anderen Komponenten nicht dokumentiert sind. Bei einer Modifikation von Komponente K 1 wird unterstellt, dass der zuständige Entwickler die beiden nicht dokumentierten Abhängigkeiten zu den Komponenten K 3 und K 4 unter Umständen nicht vollständig versteht. Als Folge verändert der Entwickler K 1 derart, dass K 3 und K 4 mit einer gewissen Wahrscheinlichkeit nicht mehr konsistent und fehlerfrei mit K 1 interagieren. Dementsprechend entstehen mit einer bestimmten Wahrscheinlichkeit Fehler in den Komponenten K 3 und K 4. Hingegen steht für die Abhängigkeit zwischen K 1 und der angrenzende Komponente K 2 eine entsprechende technische Programmdokumentation zur Verfügung. Abhängig vom Umfang und der Qualität der zusätzlichen Informationen wird angenommen, dass der

zuständige Entwickler ein besseres Verständnis über diese Komponenten und deren Zusammenwirken gewinnen kann (Rajlich 2009, S. 4) und dementsprechend mit einer geringeren Wahrscheinlichkeit einen Fehler in K 2 verursacht (Dzidek et al. 2008, S. 424). Im zweiten Schritt werden die entstandenen Fehler behoben. Dazu müssen die Komponenten K 2, K 3 und K 4 mit den entsprechenden Wahrscheinlichkeiten ebenfalls modifiziert werden. Aufgrund desselben Effekts entstehen wiederum mit einer bestimmten (bedingten) Wahrscheinlichkeit Fehler in den Komponenten K 6, K 7, K 9, K 10, K 12 und K 13 sowie mit einer entsprechend geringeren (bedingten) Wahrscheinlichkeit Fehler in K 5, K 8 und K 11. Analog dazu hat eine wiederholte Fehlerbehebung eine fortlaufende Fehlerausbreitung zur Folge (Beszédes et al. 2007, S. 296). Aus Gründen der Übersichtlichkeit ist diese nicht mehr in Abb. 1 dargestellt.

Wie bereits im vierten Abschnitt dargestellt ist ein reduzierender Effekt einer adäquaten technischen Programmdokumentation auf die Fehlerwahrscheinlichkeit von nachträglichen Änderungen empirisch mehrfach bestätigt (Arisholm et al. 2006, S. 365, Dzidek et al. 2008, S. 424, Prechelt et al. 2002, S. 595). Dennoch ist eine Situation denkbar, in der die verfügbare technische Programmdokumentation beispielsweise aufgrund einer sehr geringen Qualität keinerlei Mehrwert für das Verstehen von SW bietet. Dieser extreme Fall wird im Folgenden jedoch vernachlässigt. Vorausgesetzt wird eine hinreichend hohe Dokumentationsqualität, so dass zumindest ein beliebig kleiner Effekt der technischen Programmdokumentation auf die Fehlerwahrscheinlichkeit angenommen werden kann.

Die Annahme, dass bei der Änderung einer Komponente und bei der Fehlerbehebung pro nicht dokumentierte Abhängigkeit jeweils mit der stets gleichen Wahrscheinlichkeit neue Fehler in den angrenzenden Komponenten entstehen, erscheint auf den ersten Blick nicht mit der Realität vereinbar. So können sich die Komponenten einer Software zum Beispiel erheblich in der Beschaffenheit und Komplexität ihrer Abhängigkeiten unterscheiden. Dementsprechend dürften für verschiedene Komponenten auch die Wahrscheinlichkeiten für die Entstehung neuer Fehler merklich voneinander abweichen. Ähnliches gilt für die Annahme, dass sich die Wahrscheinlichkeit für die Entstehung neuer Fehler bei allen dokumentierten Abhängigkeiten gleichermaßen reduziert. Vergleichbar zu der Anzahl der angrenzenden Komponenten a in Annahme (A1) sollten daher bei der Verwendung des Modells beispielsweise aus Erfahrungen in vergleichbaren Softwareprojekten gewonnene Durchschnittswerte herangezogen werden. Für eine Schätzung der Fehlerwahrscheinlichkeiten kann außerdem auf die Arbeiten von Abdelmoez et al. (2004) und von Abdelmoez et al. (2005) zurückgegriffen werden. Die konkreten Werte spielen jedoch für die in Abschnitt 6 abgeleitete Hypothese ebenfalls keine Rolle. Zeichnet sich ein Softwaresystem dennoch durch eine starke Heterogenität aus, sollte das Modell nicht auf die gesamte Software angewendet werden. Stattdessen sollte die Anwendung jeweils separat auf verschiedene Teile von SW erfolgen, für die ein bestimmtes Maß der Homogenität und damit ähnliche Wahrscheinlichkeiten für die Entstehung neuer Fehler angenommen werden können.

Um die vollständigen Auswirkungen der Fehlerausbreitung monetär abbilden zu können, wird angenommen, dass alle Fehler im Rahmen der Wartung unmittelbar erkannt und behoben werden. Die Korrektur eines Fehlers bedeutet eine Auszahlung in stets gleicher Höhe. Nach der Korrektur werden alle angrenzenden Komponenten getestet, wobei für das Testen einer Abhängigkeit wiederum eine Auszahlung in fester Höhe anfällt. Da sich in der Realität diese Auszahlungen je nach Schwere des Fehlers durchaus unterscheiden dürften, sollten auch für diese Parameter entsprechende Durchschnittswerte herangezogen werden.

(A4) Fehler in SW, die während der Entwicklung eingebaut und bei Inbetriebnahme noch nicht behoben wurden, werden nicht berücksichtigt. Die Änderung einer Komponente

führt pro nicht dokumentierter Abhängigkeit mit einer (durchschnittlichen) Wahrscheinlichkeit e $(e \in \mathbb{R} | 0 \leq e \leq 1)$ *zu einem Fehler in der angrenzenden Komponente. Bei einer dokumentierten Abhängigkeit reduziert sich die Wahrscheinlichkeit für einen Fehler in der angrenzenden Komponente (durchschnittlich) um* ε $(\varepsilon \in \mathbb{R} | 0 < \varepsilon \leq e)$. *Für die Behebung eines Fehlers gilt der gleiche Effekt. Alle eingebauten Fehler werden in derselben Periode erkannt und behoben. Die Behebung eines Fehlers bedeutet eine (durchschnittliche) Auszahlung in Höhe von* c_b $(c_b \in \mathbb{R}^+)$ *für die Fehlerkorrektur in der Komponente sowie pro angrenzende Komponente eine (durchschnittliche) Auszahlung in Höhe von* c_s $(c_s \in \mathbb{R}^+)$ *für das Testen der Abhängigkeit.*

Schließlich werden ein risikoneutraler Entscheider sowie ein gegebener Kalkulationszinssatz unterstellt.

(A5) Der Entscheider ist risikoneutral. Es gilt der Kalkulationszinssatz i $(i > 0)$.

Ziel ist die Modellierung des ökonomischen Trade-off bezüglich der technischen Programmdokumentation von SW. Dazu werden im Folgenden die barwertigen Auszahlungen für die Erstellung der technischen Programmdokumentation und für die Fehlerbehebung als bewertungsunabhängige Finanzstromgrößen aus den Annahmen abgeleitet.

Tab. 1 Notationen

Variable	
r_d	Dokumentationsgrad: Relativer Anteil der dokumentierten Abhängigkeiten
Parameter	
m	Anzahl der Komponenten in SW
a	Anzahl der angrenzenden Komponenten (bzw. Abhängigkeiten) pro Komponente
r_c	Relativer Anteil der zu einem Zeitpunkt t modifizierten Komponenten
e	Wahrscheinlichkeit für die Entstehung eines Fehlers in der angrenzenden Komponente bei einer nicht dokumentierten Abhängigkeit
ε	Subtrahend, um den sich die Wahrscheinlichkeit für die Entstehung eines Fehlers in der angrenzenden Komponente bei Verfügbarkeit einer technischen Programmdokumentation reduziert
θ	Erwartungswert der Fehleranzahl aufgrund der Modifikation einer Komponente
c_d	Auszahlung für die Dokumentation einer Abhängigkeit
c_b	Auszahlung für die Korrektur eines Fehlers
c_s	Auszahlung für das Testen einer Abhängigkeit bzw. einer angrenzenden Komponente
i	Kalkulationszinssatz
n	Anzahl der betrachteten Perioden

5.2 Funktion der barwertigen Auszahlungen für Dokumentation und Fehlerbehebung

Die Auszahlungen CF_d für die Dokumentation der Abhängigkeiten ergeben sich als Produkt der Anzahl aller existierenden Abhängigkeiten in SW, des Dokumentationsgrads r_d und der Auszahlungen c_d für die Dokumentation einer einzelnen Abhängigkeit. Die Anzahl aller existierenden Abhängigkeiten setzt sich wiederum aus der Anzahl der Komponenten m und der Anzahl a der pro Komponente angrenzenden Komponenten zusammen:

$$CF_d(r_d) = m \cdot a \cdot r_d \cdot c_d$$

Da die Dokumentation der Abhängigkeiten während der Entwicklung von SW in $t = 0$ stattfindet, entspricht $CF_d(r_d)$ den barwertigen Auszahlungen für die Erstellung der technischen Programmdokumentation.

Die Auszahlungen CF_r für die Fehlerbehebung während der Wartung von SW setzen sich aus den jeweiligen Auszahlungen $CF_{r,t}$ für die Fehlerbehebung zu den Zeitpunkten $t = 1$ bis $t = n$ zusammen. $CF_{r,t}$ ergibt sich wiederum als Produkt der Anzahl der zum jeweiligen Zeitpunkt t modifizierten Komponenten $r_c \cdot m$, der Anzahl der entstehenden Fehler θ bei der Modifikation einer Komponente und der Auszahlungen für die Behebung eines Fehlers. Dabei bestehen die Auszahlungen für die Behebung eines Fehlers gemäß (A4) aus den Auszahlungen c_b für die Korrektur des Fehlers sowie den Auszahlungen $c_s \cdot a$ für das Testen aller Abhängigkeiten zu den angrenzenden Komponenten:

$$CF_{r,t} = (c_b + c_s \cdot a) \cdot r_c \cdot m \cdot \theta$$

Laut (A4) führt die Modifikation einer Komponente pro nicht dokumentierter Abhängigkeit mit Wahrscheinlichkeit e und pro dokumentierter Abhängigkeit mit Wahrscheinlichkeit $e - \varepsilon$ zu einem Fehler in der jeweils angrenzenden Komponente. Folglich werden bei der Modifikation einer Komponente initial im Erwartungswert $(1 - r_d) \cdot a \cdot e + r_d \cdot a \cdot (e - \varepsilon) = a \cdot (e - r_d \cdot \varepsilon)$ Fehler eingebaut. Da die Behebung dieser Fehler eine entsprechende Modifikation der betroffenen Komponenten erforderlich macht, führt die Behebung eines Fehlers wiederum im Erwartungswert zu $a \cdot (e - r_d \cdot \varepsilon)$ Fehlern. Da alle Fehler unmittelbar behoben werden, tritt dieser Effekt theoretisch unendlich oft in Folge auf. Die erwartete Anzahl aller Fehler θ, welche unter Berücksichtigung dieser Fehlerausbreitung in Summe aus der Modifikation einer Komponente resultieren, kann also durch folgende geometrische Reihe beschrieben werden:

$$\theta = \sum_{k=0}^{\infty} \big(a \cdot (e - r_d \cdot \varepsilon)\big) \cdot \big(a \cdot (e - r_d \cdot \varepsilon)\big)^k$$

Für die geometrische Reihe gilt:

$$\lim_{j \to \infty} \sum_{k=0}^{j} \big(a \cdot (e - r_d \cdot \varepsilon)\big) \cdot \big(a \cdot (e - r_d \cdot \varepsilon)\big)^k = \frac{a \cdot (e - r_d \cdot \varepsilon)}{\big(1 - a \cdot (e - r_d \cdot \varepsilon)\big)}$$

sofern

$$|a \cdot (e - r_d \cdot \varepsilon)| < 1 \qquad \text{(Konvergenzkriterium)}$$

Andernfalls divergiert die Reihe. Da $a \cdot (e - r_d \cdot \varepsilon) \geq 0$ gilt, kann die Bedingung für die Konvergenz dieser Reihe umgeformt werden in

$$r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon}$$

Ist also die Anzahl der pro Komponente nicht dokumentierten Abhängigkeiten $(1 - r_d) \cdot a$ multipliziert mit der Wahrscheinlichkeit e für die Entstehung eines Fehler bei einer nicht dokumentierten Abhängigkeit zuzüglich der Anzahl der pro Komponente dokumentierten Abhängigkeiten $r_d \cdot a$ multipliziert mit der entsprechend reduzierten Fehlerwahrscheinlichkeit $(e - \varepsilon)$ größer oder gleich 1, führt die Modifikation einer Komponente unter den getroffenen Annahmen zu unendlich vielen Fehlern und damit theoretisch auch zu unendlich hohen Auszahlungen für die Fehlerbehebung. Denn in diesem Fall wird mit der Behebung eines Fehlers im Erwartungswert mehr als ein neuer Fehler eingefügt. Da durch die Fehlerbehebung mehr Fehler eingebaut als behoben werden, folgt einer wie in (A4) angenommenen, fortlaufenden

Fehlerbehebung eine stetig steigende Anzahl von Fehlern in SW. Ein solches Phänomen ist beispielsweise laut Parnas (1994, S. 281) auch in der Realität und bereits seit den frühen Jahren der Softwarebranche mehrfach beobachtbar.

Es gilt:

$$a \cdot e > 1 \quad \Rightarrow \quad r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} > 0$$

Zur Vermeidung einer unendlich großen Fehleranzahl ist ein Dokumentationsgrad $r_d > 0$ notwendig, sofern die Anzahl der angrenzenden Komponenten a multipliziert mit der Wahrscheinlichkeit e für die Entstehung eines Fehler bei einer nicht dokumentierten Abhängigkeit größer als 1 ist. Gegeben sei zur Illustration das folgende Beispiel: Jede Komponente in SW besitzt fünf angrenzende Komponenten. Zudem entsteht bei der Modifikation einer Komponente mit einer Wahrscheinlichkeit von 25% ein Fehler in einer angrenzenden Komponente mit nicht dokumentierter Abhängigkeit. Bei Verfügbarkeit einer technischen Programmdokumentation reduziert sich diese Wahrscheinlichkeit um 10% pro angrenzende Komponente. In diesem Fall führt nur ein Dokumentationsgrad $r_d > 50\%$ zu einer endlichen Fehleranzahl. Ein kleinerer Dokumentationsgrad würde zu unendlich vielen Fehlern führen und eine Stabilisierung von SW könnte im Rahmen der Wartung nicht mehr gelingen.

Darüber hinaus gilt:

$$a \cdot (e - \varepsilon) \geq 1 \quad \Rightarrow \quad \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} \geq 1$$

Ist die Anzahl der pro Komponente angrenzenden Komponenten a multipliziert mit der Wahrscheinlichkeit $(e - \varepsilon)$ für die Entstehung eines Fehlers bei einer dokumentierten Abhängigkeit größer oder gleich 1, führt die Modifikation einer Komponente unter den getroffenen Annahmen selbst bei einem Dokumentationsgrad $r_d = 100\%$ zu unendlich vielen Fehlern.

Die Auszahlungen für die Behebung aller Fehler zu einem Zeitpunkt t lassen sich also mit einer entsprechenden Einschränkung der Definitionsmenge folgendermaßen formalisieren:

$$CF_{r,t}(r_d) = (c_b + c_s \cdot a) \cdot r_c \cdot m \cdot \frac{a \cdot (e - r_d \cdot \varepsilon)}{\left(1 - a \cdot (e - r_d \cdot \varepsilon)\right)}$$

mit

$$\mathbb{D} = \left\{ r_d \in \mathbb{R} | 0 \leq r_d \leq 1 \wedge a \cdot (e - \varepsilon) < 1 \wedge r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} \right\}$$

Da gemäß (A3) zu Beginn jeder Periode während des gesamten Betriebs von SW der gleiche Anteil r_c der existierenden Komponenten modifiziert wird, fallen die Auszahlungen für die Fehlerbehebung zu den Zeitpunkten $t = 1$ bis $t = n$ in jeweils gleicher Höhe an. Diese lassen sich somit als gleich bleibende Rente von $t = 1$ bis $t = n$ darstellen. Der Barwert aller Auszahlungen für die Fehlerbehebung während der Wartung ergibt sich folglich als Produkt der Auszahlungen für die Fehlerbehebung zu einem Zeitpunkt t und des Rentenbarwertfaktors β für eine gleich bleibende, nachschüssige Rente (Kruschwitz 2006, S. 48-53):

$$CF_r(r_d) = CF_{r,t}(r_d) \cdot \beta$$

wobei

$$\beta = \frac{(1 + i)^n - 1}{i \cdot (1 + i)^n}$$

Damit kann schließlich der Barwert der Auszahlungen für die Erstellung der technischen Programmdokumentation und für die Fehlerbehebung festgehalten werden:

$$CF(r_d) = CF_d(r_d) + CF_r(r_d)$$

$$CF(r_d) = m \cdot a \cdot r_d \cdot c_d + (c_b + c_s \cdot a) \cdot r_c \cdot m \cdot \frac{a \cdot (e - r_d \cdot \varepsilon)}{\left(1 - a \cdot (e - r_d \cdot \varepsilon)\right)} \cdot \beta$$

mit

$$\mathbb{D} = \left\{ r_d \in \mathbb{R} | 0 \leq r_d \leq 1 \wedge a \cdot (e - \varepsilon) < 1 \wedge r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} \right\}$$

Der Zusammenhang zwischen dem Dokumentationsgrad und den barwertigen Auszahlungen für die Dokumentation und Fehlerbehebung wird im Folgenden näher analysiert. Ziel dieser Analyse ist die Deduktion weiterer, empirisch prüfbarer Hypothesen, welche vermutlich nicht unmittelbar aus Beobachtungen in der Realität ableitbar sein dürften.

6 Analyse des Modells: Diskussion der Auszahlungsfunktion

Für die Bestimmung des auszahlungsminimalen Dokumentationsgrads r_d^* muss das Minimum der Auszahlungsfunktion $CF(r_d)$ berechnet werden. Durch das Gleichsetzen der ersten Ableitung mit 0 und dem anschließenden Auflösen der Gleichung nach r_d kann ein möglicher Extrempunkt von $CF(r_d)$ ermittelt werden:

$$r_{d,1} = \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} + \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon}$$

Dabei liegt $r_{d,1}$ unter folgenden Bedingungen innerhalb des Definitionsbereichs $\mathbb{D}$:

$$c_d \geq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{\left(1 - a \cdot (e - \varepsilon)\right)^2} \quad \Leftrightarrow \quad r_{d,1} \leq 1$$

$$(a \cdot e \geq 1) \quad \vee \quad \left(a \cdot e < 1 \quad \wedge \quad c_d \leq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{(1 - a \cdot e)^2} \right) \quad \Leftrightarrow \quad r_{d,1} \geq 0$$

Da die zweite Ableitung an diesem Punkt größer ist als 0, besitzt die Auszahlungsfunktion $CF(r_d)$ unter den obigen Bedingungen bei $r_{d,1}$ ein globales Minimum. In diesem Fall entspricht $r_{d,1}$ also dem auszahlungsminimalen Dokumentationsgrad r_d^*:

$$\left(c_d \geq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{\left(1 - a \cdot (e - \varepsilon)\right)^2} \right) \quad \wedge$$

$$\left((a \cdot e \geq 1) \quad \vee \quad \left(a \cdot e < 1 \quad \wedge \quad c_d \leq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{(1 - a \cdot e)^2} \right) \right)$$

$$\Rightarrow r_d^* = \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} + \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon}$$

Ist eine der beiden obigen Bedingungen nicht erfüllt, liegt $r_{d,1}$ außerhalb von $\mathbb{D}$:

$$c_d < \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{\left(1 - a \cdot (e - \varepsilon)\right)^2} \quad \Rightarrow \quad r_{d,1} > 1 \quad \Rightarrow \quad r_d^* = 1$$

$$a \cdot e < 1 \quad \wedge \quad c_d > \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{(1 - a \cdot e)^2} \quad \Rightarrow \quad r_{d,1} < 0 \quad \Rightarrow \quad r_d^* = 0$$

Aufgrund der Einschränkung $0 \leq r_d \leq 1$ ist $CF(r_d)$ bei $r_d^* = 1$ minimal, sofern $r_{d,1} > 1$. Analog dazu gilt $r_d^* = 0$, sofern $r_{d,1} < 0$.

Nach der Bestimmung des auszahlungsminimalen Dokumentationsgrads r_d^* werden im Folgenden Abweichungen von r_d^* untersucht:

Ungeachtet der Einschränkung $0 \leq r_d \leq 1$ weist die Funktion der barwertigen Auszahlungen $CF(r_d)$ eine positive Krümmung auf:

$$r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} \quad \Rightarrow \quad \frac{d^2 CF(r_d)}{dr_d{}^2} > 0$$

Die Krümmung nimmt dabei mit steigendem Wert für den Dokumentationsgrad r_d stetig ab:

$$r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} \quad \Rightarrow \quad \frac{d^3 CF(r_d)}{dr_d{}^3} < 0$$

Die Steigung der Auszahlungsfunktion $CF(r_d)$ ändert sich also bei einem niedrigeren Wert für r_d stärker als bei einem höheren Wert für r_d. Daraus folgt, dass die betragsmäßige Steigung von $CF(r_d)$ bei einer negativen Abweichung vom optimalen Dokumentationsgrad r_d^* größer ist als bei einer positiven Abweichung in gleicher Höhe:

$$\left|\frac{dCF(r_d^* - \Delta r_d)}{d(r_d^* - \Delta r_d)}\right| > \left|\frac{dCF(r_d^* + \Delta r_d)}{d(r_d^* + \Delta r_d)}\right| \quad \text{mit} \quad r_d^* - \Delta r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} \quad \wedge \quad \Delta r_d > 0$$

Dementsprechend nimmt auch die Auszahlungsfunktion bei einer negativen Abweichung von r_d^* einen größeren Wert an als bei einer positiven Abweichung in gleicher Höhe:

$$CF(r_d^* - \Delta r_d) > CF(r_d^* + \Delta r_d) \quad \text{mit} \quad r_d^* - \Delta r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} \quad \wedge \quad \Delta r_d > 0$$

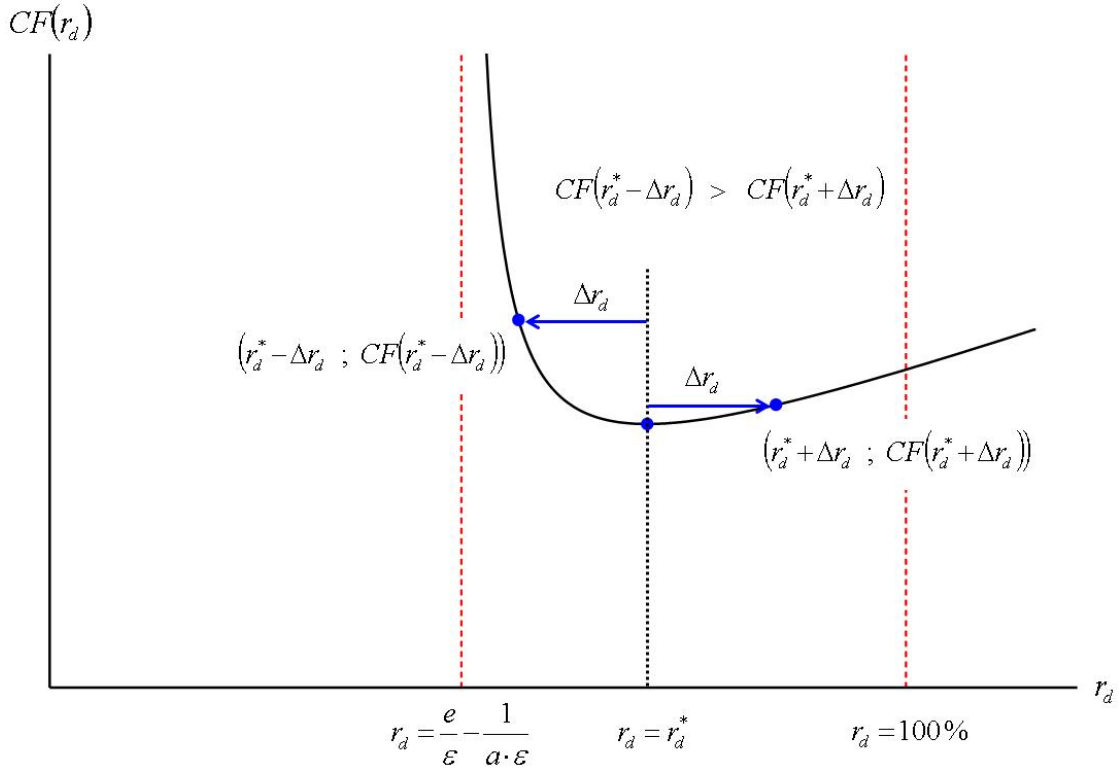
Für den Definitionsbereich $\mathbb{D}$ kann damit folgende Hypothese formuliert werden:

$$\forall\, \Delta r_d > 0 \quad \left(r_d^*, (r_d^* - \Delta r_d), (r_d^* + \Delta r_d) \in \mathbb{D} \longrightarrow CF(r_d^* - \Delta r_d) > CF(r_d^* + \Delta r_d)\right)$$

Unter den Einschränkungen der getroffenen Annahmen ist eine positive Abweichung vom optimalen Dokumentationsgrad stets vorteilhaft im Vergleich zu einer betragsmäßig gleichen, negativen Abweichung vom optimalen Dokumentationsgrad.

Abb. 2 illustriert diese Hypothese anhand eines beispielhaften Verlaufs der Auszahlungsfunktion $CF(r_d)$.

Abb. 2 Abweichungen vom optimalen Dokumentationsgrad r_d^*



7 Kritische Würdigung

7.1 Robustheit der Ergebnisse

Bemerkenswert ist, dass die im vorherigen Abschnitt abgeleitete Hypothese unter den Einschränkungen der getroffenen Annahmen für alle zulässigen Werte aller gegebenen Parameter Gültigkeit besitzt. Diese gilt dabei insbesondere auch für jeden beliebig kleinen Wert des Parameters ε. Das heißt, solange die verfügbare technische Programmdokumentation irgendeinen reduzierenden Effekt auf die Fehlerwahrscheinlichkeit von nachträglichen Modifikationen ausübt ($\varepsilon > 0$), welcher empirisch mehrfach bestätigt ist (Arisholm et al. 2006, S. 365, Dzidek et al. 2008, S. 424, Prechelt et al. 2002, S. 595), bedeutet eine positive Abweichung vom optimalen Dokumentationsgrad immer niedrigere Auszahlungen als eine betragsmäßig gleiche, negative Abweichung. Wie groß dieser Effekt ist, also um welchen Wert sich die Fehlerwahrscheinlichkeit bei Verfügbarkeit einer technischen Programmdokumentation reduziert, spielt dabei keine Rolle. Unter der Annahme, dass sich die Qualität der technischen Programmdokumentation auf den Parameter ε auswirkt, besitzt die Dokumentationsqualität keine Bedeutung für die obige Hypothese. Auch der konkrete Wert für den Parameter e schränkt die Gültigkeit dieser Hypothese nicht ein. Selbst bei einer beliebig kleinen Wahrscheinlichkeit, dass die Modifikation einer Komponente einen Fehler in einer angrenzenden Komponente mit nicht dokumentierter Abhängigkeit verursacht, ist für $\varepsilon > 0$ eine positive Abweichung vom optimalen Dokumentationsgrad stets vorteilhaft im Vergleich zu einer gleich hohen, negativen Abweichung.

Darüber hinaus ist die Hypothese ebenfalls unabhängig vom Wert des Rentenbarwertfaktors β und damit insbesondere auch für jeden beliebig hohen Kalkulationszinssatz i gültig. Gemäß den obigen Annahmen fallen die Auszahlungen für die Erstellung der technischen Programmdokumentation in $t = 0$ und die Auszahlungen für die Fehlerbehebung zu den Zeitpunkten

$t = 1$ bis $t = n$ an. Im Gegensatz zu den gegenwärtigen Auszahlungen für die Dokumentation wertet also ein positiver Kalkulationszinssatz die zukünftigen Auszahlungen für die Fehlerbehebung auf ihren Barwert ab. Eine zunehmend negative Abweichung vom optimalen Dokumentationsgrad würde die gegenwärtigen Auszahlungen für die Dokumentation reduzieren und im Gegenzug die Anzahl der Fehler und damit die zukünftigen Auszahlungen für die Fehlerbehebung erhöhen. Der Zinseffekt würde sich dabei zunehmend positiv auf die barwertigen Auszahlungen $CF(r_d)$ auswirken. Eine zunehmend positive Abweichung vom optimalen Dokumentationsgrad hätte dagegen eine zunehmend negative Auswirkung des Zinseffekts auf $CF(r_d)$ zur Folge, da die gegenwärtigen Auszahlungen für die Dokumentation erhöht und die zukünftigen Auszahlungen für die Fehlerbehebung reduziert würden. Dennoch übertrifft die Veränderung der Auszahlungen für die Fehlerbehebung im Falle einer Abweichung von r_d^* selbst bei einem beliebig hohen Kalkulationszinssatz stets die Auswirkungen des Zinseffekts. Auch beliebig hohe Auszahlungen c_d für die Erstellung der technischen Programmdokumentation oder beliebig niedrige Werte für den Anteil r_c der zu modifizierenden Komponenten schränken die Gültigkeit der obigen Hypothese nicht ein.

Des Weiteren weist die Hypothese ebenfalls eine gewisse Robustheit bezüglich des Zusammenhangs zwischen der Anzahl der dokumentierten Aufrufe und den Auszahlungen für die Erstellung der technischen Programmdokumentation auf. Für die Erstellung des Modells wird in Abschnitt 5 ein linearer Zusammenhang angenommen. Es kann jedoch gezeigt werden, dass die obige Hypothese ebenfalls auf der Basis einer quadratischen Auszahlungsfunktion für die Dokumentation ableitbar ist. Erst bei einem kubischen Zusammenhang, welcher jedoch nicht realistisch erscheint, kann diese Hypothese nicht mehr für alle Werte der gegebenen Parameter deduziert werden.

7.2 Limitationen des Modells und der Ergebnisse

Da das Modell auf vereinfachenden Annahmen beruht, unterliegen die Ergebnisse und deren Aussagekraft entsprechenden Einschränkungen.

So werden in der Annahme (A3) jegliche Erweiterungen von SW um neue Komponenten im Rahmen der Wartung ausgeschlossen. Dementsprechend bildet das Modell nur die Auswirkungen der technischen Programmdokumentation auf die Fehleranfälligkeit von Änderungen existierender Komponenten bei konstantem Umfang und gleichbleibender struktureller Komplexität ab. In der Realität erfolgt eine Anpassung an neue Anforderungen sowohl über Änderungen existierender Komponenten als auch durch Erweiterungen der Software um neue Komponenten (Krishnan et al. 2004, S. 396). Dabei haben derartige Erweiterungen in der Regel einen Anstieg der strukturellen Komplexität zur Folge, da sie die Anzahl der Interaktionen und Abhängigkeiten zwischen den Komponenten einer Software erhöhen (Lehman 1996, S. 109). Da zugleich die Fehleranfälligkeit einer Softwarekomponente umso höher ist, desto mehr angrenzende Komponenten diese besitzt (Briand et al. 2002, S. 714), dürften nachträgliche Erweiterungen um neue Komponenten ebenfalls die Fehleranfälligkeit erhöhen. Folglich blieben die vorgestellten Ergebnisse nicht nur bestehen, sondern würden sogar noch verschärft.

Darüber hinaus betrachtet das Modell ausschließlich die Auswirkungen der Dokumentation von Abhängigkeiten auf Fehler, welche sich über diese Abhängigkeiten auf angrenzende Komponenten ausbreiten (Beszédes et al. 2007, S. 296). Die vorgestellten Ergebnisse besitzen daher ausschließlich Gültigkeit im Rahmen dieser Einschränkung. Vernachlässigt werden die Dokumentation von Code-Fragmenten ohne Abhängigkeiten zu angrenzenden Komponenten sowie alle Fehler innerhalb einer Komponente ohne Auswirkungen auf angrenzende Komponenten. Eine Auflösung dieser Einschränkung kann grundsätzlich eine fundamentale Ände-

rung der vorgestellten Ergebnisse bedeuten. Jedoch lassen sich auch erste Argumente anführen, dass insbesondere die im vorherigen Abschnitt deduzierte Hypothese auch bei einer entsprechenden Modellerweiterung bestehen bleiben könnte. Einerseits ist die obige Hypothese unabhängig von den Parametern e und ε. Solange also eine beliebig kleine Anzahl an Fehlern dem Effekt der Ausbreitung auf angrenzende Komponenten unterliegt, besitzt die Hypothese Gültigkeit. Eine zusätzliche Berücksichtigung von Fehlern ohne Auswirkungen auf angrenzende Komponenten dürfte das Ergebnis daher sogar noch verschärfen. Andererseits würde jedoch eine zusätzliche Betrachtung der Dokumentation von unabhängigen Code-Fragmenten vermutlich auch höhere Auszahlungen für die Erstellung einer technischen Programmdokumentation bedeuten. Da aber selbst in diesem Fall eine Funktion dritten oder höheren Grades für die Dokumentationsauszahlungen unrealistisch erscheint und die obige Hypothese auch für eine quadratische Auszahlungsfunktion deduzierbar ist, dürfte auch eine entsprechende Modellerweiterung zu einem vergleichbaren Ergebnis führen. Dennoch sei darauf hingewiesen, dass zukünftige Forschungsarbeiten diese Vermutung erst noch belegen müssen.

Weiterhin beschränkt sich das Modell ausschließlich auf die technische Programmdokumentation einer Software als Einflussgröße auf die Fehleranfälligkeit von nachträglichen Modifikationen. In der Realität dürften jedoch wesentlich mehr Faktoren wie zum Beispiel die Qualität des Quellcodes oder die Erfahrung des Wartungspersonals einen Effekt auf die Entstehung von Fehlern während der Softwarewartung ausüben. Die vorgestellten Ergebnisse besitzen demnach einzig im Rahmen dieser Einschränkung Gültigkeit. Für die Berücksichtigung weiterer Faktoren auf die Fehleranfälligkeit von Wartungsmaßnahmen ist eine entsprechende Erweiterung des vorgestellten Modells erforderlich.

Zudem wird vereinfachend angenommen, dass jede Komponente gleich viele angrenzende Komponenten besitzt, die Auszahlungen für die Dokumentation einer Abhängigkeit stets gleich hoch sind, sich die dokumentierten Abhängigkeiten gleichmäßig auf alle Komponenten verteilen, in jeder Periode ein jeweils gleicher Anteil der existierenden Komponenten modifiziert wird, die Fehlerwahrscheinlichkeiten von nachträglichen Änderungen bei dokumentierten und bei nicht dokumentierten Abhängigkeiten jeweils gleich hoch sind sowie die Korrektur eines Fehlers eine Auszahlung in stets gleicher Höhe bedeutet. Wie bereits diskutiert werden diese Annahmen in der Realität nur sehr bedingt zutreffen. Daher müssen für diese Parameter entsprechende Durchschnittswerte herangezogen werden. Zeichnet sich ein Softwaresystem dennoch durch eine starke Heterogenität bezüglich dieser Parameter aus, ist eine separate Anwendung des Modells auf einzelne Teile der Software sinnvoll, für die ein ausreichendes Maß an Homogenität angenommen werden kann. An dieser Stelle sei jedoch nochmals betont, dass die konkreten Werte dieser Parameter auf die im vorherigen Abschnitt abgeleitete Hypothese, nämlich dass eine positive Abweichung vom optimalen Dokumentationsgrad im Vergleich zu einer gleich hohen negativen Abweichung stets vorteilhaft ist, keinen Einfluss besitzen. Aufgrund dieser Homogenisierung kann das vorgestellte Modell allerdings keinen Hinweis liefern, welche konkreten Abhängigkeiten von SW bevorzugt dokumentiert werden sollten. Für die Generierung derartiger Hypothesen wäre eine Erweiterung des Modells notwendig, welche die Homogenisierungsannahmen auflöst.

Mit den Auszahlungen für die Erstellung einer technischen Programmdokumentation und für die Fehlerbehebung berücksichtigt das Modell nur einen Teil der zu einem Software-Projekt gehörigen Zahlungsströme. Folglich eignet sich das Modell nicht für eine Beurteilung der absoluten Vorteilhaftigkeit der Erstellung einer technischen Programmdokumentation in Softwareprojekten. Diese muss unter Einbezug aller relevanten Ein- und Auszahlungen beispielsweise mittels der Kapitalwertmethode oder der kapitalwertigen Risikoanalyse (Bamberg et al. 2006) erfolgen.

Ferner bezieht sich die existierende Literatur, auf die sich ein Großteil der obigen Modellannahmen stützt, überwiegend auf objektorientierte Softwaresysteme. Folglich beschränkt sich die Aussagekraft der vorgestellten Ergebnisse nur auf derartige Software. Für eine Prüfung, inwieweit das Modell auch außerhalb dieser Einschränkung Gültigkeit besitzt, müssen die den Annahmen zugrunde liegenden Zusammenhänge ebenfalls im Kontext anderer Programmierparadigmen empirisch geprüft werden.

8 Theoretischer und praktischer Beitrag

Aus einer wissenschaftlichen Perspektive wurden die Auswirkungen der technischen Programmdokumentation eines Softwaresystems auf die Fehleranfälligkeit von nachträglichen Modifikationen im Zuge der Wartung und die damit verbundenen ökonomischen Fragestellungen stark vernachlässigt. Zwar bestätigen einige empirische Studien einen reduzierenden Effekt der technischen Programmdokumentation auf die Entstehung von Fehlern im Rahmen der Softwarewartung (Arisholm et al. 2006, S. 365, Dzidek et al. 2008, S. 424, Prechelt et al. 2002, S. 595). Dieser Zusammenhang genügt jedoch nicht als Grundlage für die Lösung der damit verbundenen, praxisrelevanten Entscheidungsprobleme beispielsweise hinsichtlich des ökonomisch sinnvollen Investitionsvolumens in eine technische Programmdokumentation.

Darüber hinaus beziehen sich die existierenden Ansätze zur Prognose von Fehlerhäufigkeiten oder Versagensfällen weitestgehend auf die Entwicklungsphase oder auf den Zeitpunkt der Inbetriebnahme einer Software (Grottke 2005). Die Entstehung von Fehlern aufgrund nachträglicher Modifikationen während der Wartungsphase wird überwiegend vernachlässigt. Außerdem berücksichtigt ein Großteil dieser Ansätze nicht die Verfügbarkeit einer technischen Programmdokumentation als Einflussfaktor auf die Entstehung und Ausbreitung von Fehlern. Eine Ausnahme bildet beispielsweise der Beitrag von Takahashi/Kamayachi (1989), in dem die Autoren einen signifikanten Zusammenhang zwischen dem Umfang der Designdokumentation und der Anzahl der Fehler zeigen, welche zu Beginn der Testphase während der Entwicklung in der Software verbleiben. Jedoch liegt der Fokus dabei wiederum nicht auf der Entstehung und Ausbreitung von Fehlern während der Softwarewartung. Als Konsequenz eignen sich auch diese Ansätze nicht für eine befriedigende Beantwortung der oben erwähnten Fragestellung, inwieweit eine Investition in die Erstellung einer technischen Programmdokumentation während der Softwareentwicklung zur Reduktion der Fehleranfälligkeit von nachgelagerten Wartungsmaßnahmen ökonomisch sinnvoll ist.

Vor diesem Hintergrund präsentiert der vorliegende Beitrag einen neuartigen Ansatz, welcher die barwertigen Auszahlungen für die Erstellung einer technischen Programmdokumentation während der Softwareentwicklung und für die Behebung der während der Softwarewartung aufgrund von nachträglichen Modifikationen entstehenden Fehler gegenüberstellt. Dazu wird mittels Deduktion ein neuer, über die bisher bekannten Zusammenhänge hinausgehender Zusammenhang zwischen dem Dokumentationsgrad einer Software und der Anzahl der im Zuge der Softwarewartung entstehenden Fehler abgeleitet. Diese Hypothese kombiniert dabei insbesondere die in der bestehenden Literatur bereits empirisch bestätigten Zusammenhänge zwischen der Anzahl der Fehler aufgrund nachträglicher Änderungen und der Verfügbarkeit einer adäquaten technischen Programmdokumentation (Arisholm et al. 2006, S. 365, Dzidek et al. 2008, S. 424, Prechelt et al. 2002, S. 595) sowie zwischen der Fehleranfälligkeit einer Softwarekomponente und der Anzahl ihrer Abhängigkeiten zu angrenzenden Komponenten (Briand et al. 2002, S. 714, Gyimóthy et al. 2005, S. 909, Singh et al. 2010, S. 31). Dabei wird auch dem Umstand Rechnung getragen, dass die technische Programmdokumentation beispielsweise abhängig von ihrer Qualität einen unterschiedlichen Mehrwert für das Verständnis des zu modifizierenden Codes durch das Wartungspersonal bieten und damit auch die

Fehlerwahrscheinlichkeit von nachträglichen Änderungen in unterschiedlichem Maße reduzieren kann. Darauf aufbauend werden die barwertigen Auszahlungen für die Dokumentation während der Softwareentwicklung und für die Fehlerbehebung während der nachgelagerten Wartung modelliert sowie der auszahlungsminimale Dokumentationsgrad berechnet. Insbesondere ein solcher Zusammenhang kann nun eine geeignete Grundlage für die Lösung der oben erläuterten Problemstellung bilden. Ferner ermöglicht die Analyse der Auswirkungen einer Abweichung vom optimalen Dokumentationsgrad die Generierung einer weiteren Hypothese: Eine positive Abweichung vom optimalen Dokumentationsgrad ist stets vorteilhaft gegenüber einer betragsmäßig gleichen, negativen Abweichung. Diese Aussage ist dabei unter den Einschränkungen der getroffenen Annahmen für jeden zulässigen Wert aller unabhängigen Parameter gültig. Gerade eine solche Hypothese lässt sich vermutlich nicht unmittelbar aus Beobachtungen in der Realität induzieren, sondern nur durch einen deduktiven Ansatz, wie er im vorliegenden Beitrag gewählt wurde, ableiten.

Eine Grundlage des Modells bilden aber auch einige vereinfachende und verallgemeinernde Annahmen. Daher möchten das vorgestellte Modell und deren Ergebnisse vorrangig als Grundlage für weitere Forschungsaktivitäten verstanden werden. Eine unmittelbare Ableitung von Handlungsempfehlungen für die Praxis beispielsweise hinsichtlich eines ökonomisch sinnvollen Investitionsvolumens in die Erstellung einer adäquaten technischen Programmdokumentation zur Reduzierung der Fehleranfälligkeit von Wartungsaktivitäten erscheint alleine auf der Basis der vorgestellten Modellergebnisse noch nicht sinnvoll. Zuvor dürften sich entsprechende Modellerweiterungen im Rahmen zukünftiger Forschungsarbeiten lohnen, welche insbesondere die gegenüber der Realität vereinfachenden Annahmen auflösen und weitere Zusammenhänge aufdecken. Ferner muss eine kritische Prüfung der abgeleiteten Hypothesen erfolgen. Erst bei einer empirisch hinreichend starken Bestätigung ist eine fundierte Grundlage für die Generierung normativer Aussagen gegeben.

Sofern jedoch insbesondere die im sechsten Abschnitt vorgestellte Hypothese auch nach der Auflösung einiger vereinfachenden Annahmen bestätigt werden kann, müsste das aktuelle Vorgehen in der Praxis kritisch hinterfragt werden. Aktuell scheint in Softwareprojekten die Ansicht vertreten zu sein, angesichts eines hohen Zeit- und Kostendrucks auf eine adäquate Dokumentation verzichten zu können. Der Fokus scheint überwiegend auf der Erreichung des nächsten Meilensteins zu liegen und die langfristige Lebenszyklusbetrachtung wird vernachlässigt (Zarnekow et al. 2004, S. 181). Sowohl Entwickler als auch die zuständigen Führungskräfte nehmen offenbar die Dokumentation weitgehend als unattraktive Maßnahme wahr, da diese das Erreichen des nächsten Meilensteins nicht beschleunigt, sondern eher verzögert (Parnas 1994, S. 283). Die abgeleiteten Hypothesen stellen an dieser Stelle zumindest ein erstes Indiz dar, dass dieses Vorgehen insbesondere in Bezug auf die technische Programmdokumentation ökonomisch nicht sinnvoll sein könnte. Jedoch stellen Softwareentwickler oftmals eine „knappe Ressource“ dar und mit der Erstellung einer umfassenderen technischen Programmdokumentation sind diese entsprechend länger an ein Softwareprojekt gebunden. Daher stellt sich die Frage nach einem ökonomisch sinnvollen Investitionsvolumen in eine adäquate technische Programmdokumentation nur im ersten Schritt für ein isoliert betrachtetes Projekt und ist darüber hinaus auch in der Frage nach dem ökonomisch sinnvollen Zeitpunkt für den Beginn von Folgeprojekten eingebettet. Dieser Umstand könnte eine Ursache dafür sein, dass der bereits bekannte und empirisch bestätigte Zusammenhang zwischen der Verfügbarkeit einer technischen Programmdokumentation und der Fehleranfälligkeit von Wartungsmaßnahmen offenbar keine Auswirkungen auf das bisherige Dokumentationsverhalten in der Praxis hatte.

9 Zusammenfassung und Ausblick

Wie bereits erläutert ergibt sich für jedes Softwareprojekt die Frage, inwieweit die Erstellung einer technischen Programmdokumentation während der Entwicklung zur Reduzierung der Fehleranfälligkeit von nachgelagerten Wartungsmaßnahmen ökonomisch sinnvoll ist. Die existierende Literatur bietet jedoch mitnichten eine ausreichende Grundlage für eine befriedigende Beantwortung dieser Fragestellung. Vor diesem Hintergrund stellt der vorliegende Beitrag einen neuartigen Ansatz vor, welcher, ergänzend zur bisherigen Literatur, im Sinne einer wissenschaftlichen Erklärung neue Zusammenhänge beispielsweise zwischen dem Dokumentationsgrad einer Software und der Anzahl der im Zuge der Softwarewartung entstehenden Fehler deduziert. Da als Grundlage jedoch einige vereinfachende und verallgemeinernde Annahmen getroffen werden, möchten der vorgestellte Ansatz und deren Ergebnisse weniger als unmittelbar in der Praxis anwendbares Modell, sondern vielmehr komplementär zu den bisherigen Beiträgen beispielsweise zur Prognose von Fehlerhäufigkeiten und Softwarezuverlässigkeit als Grundlage für weitere Forschungsaktivitäten verstanden werden.

Zukünftige empirische Forschungsarbeiten müssen nun die generierten Hypothesen einer kritischen Prüfung unterziehen, inwieweit sie die entsprechenden Phänomene in der Realität prognostizieren können. Aufgrund der Wahl eines formalen Ansatzes liegen diese in einer insbesondere für eine derartige Prüfung geeigneten Form vor (Meredith et al. 1989, S. 308). Darüber hinaus sollten zukünftige Forschungsbeiträge beispielsweise mithilfe entsprechender Modellerweiterungen die Auflösung der gegenüber der Realität vereinfachenden Annahmen sowie die Deduktion zusätzlicher Hypothesen verfolgen. Diese können dann wiederum als Grundlage für die weitere empirische Forschung in diesem Bereich sowie für die Lösung der relevanten Entscheidungsprobleme dienen. Im Konkreten könnten zukünftige Modellerweiterungen eine zusätzliche Betrachtung von Fehlern ohne Auswirkungen auf angrenzende Komponenten sowie die Berücksichtigung von Erweiterungen einer Software im Rahmen der Wartung zum Gegenstand haben. Auch der Einbezug weiterer Faktoren auf die Fehleranfälligkeit von nachträglichen Modifikationen wie zum Beispiel die Code-Qualität oder die Erfahrung des Wartungspersonals erscheint lohnenswert. Derartige Beiträge könnten analysieren, ob und inwieweit insbesondere die im Abschnitt 6 dargelegte Hypothese auch nach Auflösung der entsprechenden Annahmen deduzierbar ist. Für die Generierung weiterer Hypothesen in Bezug auf die Frage, welche Softwarekomponenten und Abhängigkeiten bevorzugt dokumentiert werden sollten, könnten sich zukünftige Forschungsarbeiten auf eine Auflösung der Homogenisierungsannahmen konzentrieren. Darüber hinaus kann das vorgestellte Modell die Grundlage für eine kapitalwertige Risikoanalyse bilden, welche alle relevanten Ein- und Auszahlungen berücksichtigt und die absolute Vorteilhaftigkeit der technischen Programmdokumentation einer Software untersucht. Auch eine Anpassung des Modells auf die Spezifika iterativer beziehungsweise inkrementeller Vorgehensmodelle bei der Softwareentwicklung und auf Situationen, in denen die Entwicklung und Wartung eines Softwaresystems von unterschiedlichen Parteien durchgeführt werden, könnte zu relevanten Erkenntnissen führen.

Zugleich dürften die Zusammenhänge zwischen der technischen Programmdokumentation und der Softwarewartung auch auf weitere praxisrelevante Entscheidungssituationen Einfluss besitzen: Inwieweit ist die Wartung eines Softwaresystems insgesamt ökonomisch sinnvoll? Unter welchen Bedingungen kann sich das Outsourcing der Softwarewartung lohnen? Wann sind welche Maßnahmen des Re-Engineerings sinnvoll und zu welchem Zeitpunkt sollte ein Altsystem durch eine neue Software ersetzt werden? Für die vollständige und befriedigende Beantwortung dieser und vieler weiterer Fragestellungen sind die vorgestellten Ergebnisse jedoch mitnichten ausreichend. Vielmehr ist die Aufdeckung weiterer Zusammenhänge im

Rahmen zukünftiger Forschungsarbeiten erforderlich, für welche der vorliegende Beitrag eine Grundlage bilden kann.

Literatur

Abdelmoez W, Nassar DM, Shereshevsky M, Gradetsky N, Gunnalan R, Ammar HH, Yu B, Mili A (2004) Error Propagation in Software Architectures. In: Proceedings of the International Symposium on Software Metrics (METRICS '04). IEEE Computer Society, 384-393

Abdelmoez W, Shereshevsky M, Gunnalan R, Ammar HH, Yu B, Bogazzi S, Korkmaz M, Mili A (2005) Quantifying Software Architectures: An Analysis of Change Propagation Probabilities. In: Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications (AICCSA '05). IEEE, 124-vii

Aggarwal KK, Singh Y, Kaur A, Malhotra R (2009) Empirical Analysis for Investigating the Effect of Object-Oriented Metrics on Fault Proneness: A Replicated Case Study. Software Process: Improvement and Practice 14(1):39-62

Arisholm E, Briand LC, Hove SE, Labiche Y (2006) The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation. IEEE Transactions on Software Engineering 32(6):365-381

Bamberg G, Dorfleitner G, Krapp M (2006) Unternehmensbewertung unter Unsicherheit: Zur entscheidungstheoretischen Fundierung der Risikoanalyse. ZfB 76(3):287-307

Beszédes Á, Gergely T, Jász J, Tóth G, Gyimóthy T, Rajlich V (2007) Computation of Static Execute After Relation with Applications to Software Maintenance. In: Proceedings of the 23rd IEEE International Conference on Software Maintenance (ICSM '07). IEEE, 295-304

Binkley AB, Schach SR (1998) Validation of the Coupling Dependency Metric as a Predictor of Run-Time Failures and Maintenance Measures. In: Proceedings of the 20th International Conference on Software Engineering (ICSE '98). IEEE Computer Society, 452-455

Briand LC, Melo WL, Wüst J (2002) Assessing the Applicability of Fault-Proneness Models Across Object-Oriented Software Projects. IEEE Transactions on Software Engineering 28(7):706-720

de Souza SCB, Anquetil N, de Oliveira KM (2005) A Study of the Documentation Essential to Software Maintenance. In: Proceedings of the 23rd Annual International Conference on Design of Communication: Documenting & Designing for Pervasive Information. ACM, 68-75

Dzidek WJ, Arisholm E, Briand LC (2008) A Realistic Empirical Evaluation of the Costs and Benefits of UML in Software Maintenance. IEEE Transactions on Software Engineering 34(3):407-432

Eaddy M, Zimmermann T, Sherwood KD, Garg V, Murphy GC, Nagappan N, Aho AV (2008) Do Crosscutting Concerns Cause Defects? IEEE Transactions on Software Engineering 34(4):497-515

Fenton NE, Ohlsson N (2000) Quantitative Analysis of Faults and Failures in a Complex Software System. IEEE Transactions on Software Engineering 26(8):797-814

Grottke M (2005) Prognose von Softwarezuverlässigkeit, Softwareversagensfällen und Softwarefehlern. In: Mertens P., Rässler S. (eds) Prognoserechnung. 6. Auflage, Physica, Heidelberg, S 459-487

Gyimóthy T, Ferenc R, Siket I (2005) Empirical Validation of Object-Oriented Metrics on Open Source Software for Fault Prediction. IEEE Transactions on Software Engineering 31(10):897-910

Hanssen GK, Yamashita AF, Conradi R, Moonen L (2009) Maintenance and Agile Development: Challenges, Opportunities and Future Directions. In: Proceedings of the 25th IEEE International Conference on Software Maintenance (ICSM '09). IEEE, 487-490

Hruschka P (2003) Agility. Informatik-Spektrum 26(6):397-401

Ko AJ, Myers BA, Coblenz MJ, Aung HH (2006) An Exploratory Study of How Developers Seek, Relate, and Collect Relevant Information during Software Maintenance Task. IEEE Transactions on Software Engineering 32(12):971-987

Krishnan MS, Mukhopadhyay T, Kriebel CH (2004) A Decision Model for Software Maintenance. Information Systems Research 15(4):396-412

Kruschwitz L (2006) Finanzmathematik: Lehrbuch der Zins-, Renten-, Tilgungs-, Kurs- und Renditerechnung. 4. Auflage, Vahlen, München

Lehman MM (1996) Laws of Software Evolution Revisited. In: Springer. (eds) Proceedings Lecture Notes in Computer Science 1149. Springer, 108-124

MacCormack A, Rusnak J, Baldwin CY (2006) Exploring the Structure of Complex Software Designs: An Empirical Study of Open Source and Proprietary Code. Management Science 52(7):1015-1030

Meredith JR, Raturi A, Amoako-Gyampah K, Kaplan B (1989) Alternative Research Paradigms in Operations. Journal of Operations Management 8(4):297-326

Olague HM, Etzkorn LH, Gholston S, Quattlebaum S (2007) Empirical Validation of Three Software Metrics Suites to Predict Fault-Proneness of Object-Oriented Classes Developed Using Highly Iterative or Agile Software Development Processes. IEEE Transactions on Software Engineering 33(6):402-419

Parnas DL (1994) Software Aging. In: Proceedings of the 16th International Conference on Software Engineering (ICSE '94). IEEE Computer Society Press, Los Alamitos, USA, 279-287

Prechelt L, Unger-Lamprecht B, Philippsen M, Tichy WF (2002) Two Controlled Experiments Assessing the Usefulness of Design Pattern Documentation in Program Maintenance. IEEE Transactions on Software Engineering 28(6):595-606

Radatz J (1990) IEEE Standard Glossary of Software Engineering Terminology. Std 610.12-1990, Standards Coordinating Committee of the Computer Society of the IEEE

Rajlich V (2009) Intensions are a Key to Program Comprehension. In: IEEE 17th International Conference on Program Comprehension (ICPC '09). IEEE, 1-9

Rajlich V (2006) Changing the Paradigm of Software Engineering. Communications of the ACM 49(8):67-70

Riedl R, Kepler J (2003) Begriffliche Grundlagen des Business Process Outsourcing. Information Management & Consulting 18(3):6-11

Robillard MP, Coelho W, Murphy GC (2004) How Effective Developers Investigate Source Code: An Exploratory Study. IEEE Transactions on Software Engineering 30(12):889-903

Rostkowycz AJ, Rajlich V, Marcus A (2004) A Case Study on the Long-Term Effects of Software Redocumentation. In: Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM '04). IEEE, 92-101

Sharif KY, Buckley J (2008) Developing Schema for Open Source Programmers' Information-Seeking. In: 3rd International Symposium on Information Technology (ITSIM '08). IEEE, 1-9

Singh Y, Kaur A, Malhotra R (2010) Empirical Validation of Object-Oriented Metrics for Predicting Fault Proneness Models. Software Quality Journal 18(1):3-35

Sneed HM (2008) Offering Software Maintenance as an Offshore Service. In: IEEE International Conference on Software Maintenance (ICSM '08). IEEE, 1-5

Statista (2010a) Weltweite Ausgaben von Wirtschaft und öffentlicher Verwaltung für IT-Outsourcing im Jahr 2010 in Milliarden US Dollar. http://de.statista.com/statistik/daten/studie/163365/umfrage/weltweite-ausgaben-fuer-it-outsourcing-in-2010/. Abruf am 2011-18-02

Statista (2010b) Weltweite IT-Ausgaben von Wirtschaft und öffentlicher Verwaltung für ITK Produkte und Services im Jahr 2010 in Milliarden US-Dollar. http://de.statista.com/statistik/daten/studie/163257/umfrage/weltweite-ausgaben-fuer-itk-produkte-und-services-in-2010/. Abruf am 2011-18-02

Subramanyam R, Krishnan MS (2003) Empirical Analysis of CK Metrics for Object-Oriented Design Complexity: Implications for Software Defects. IEEE Transactions on Software Engineering 29(4):297-310

Takahashi M, Kamayachi Y (1989) An Empirical Study of a Model for Program Error Prediction. IEEE Transactions on Software Engineering 15(1):82-86

Tan Y, Mookerjee VS (2005) Comparing Uniform and Flexible Policies for Software Maintenance and Replacement. IEEE Transactions on Software Engineering 31(3):238-255

Van Vliet H (2008) Software Engineering: Principles and Practices. 3. Auflage, John Wiley & Sons, West Sussex, England

Yu P, Systa T, Müller H (2002) Predicting Fault-Proneness using OO Metrics: An Industrial Case Study. In: Proceedings of the 6th European Conference on Software Maintenance and Reengineering (CSMR '02). IEEE Computer Society, Los Alamitos, USA, 99-107

Zarnekow R, Scheeg J, Brenner W (2004) Untersuchung der Lebenszykluskosten von IT-Anwendungen. Wirtschaftsinformatik 46(3):181-187

Zimmermann T, Nagappan N (2008) Predicting Defects Using Network Analysis on Dependency Graphs. In: Proceedings of the 30th International Conference on Software Engineering (ICSE '08). ACM, 531-540

Anhang

In Abschnitt 6 wird der optimale Dokumentationsgrad r_d^* bestimmt, welcher die barwertigen Auszahlungen für die Dokumentation und für die Fehlerbehebung minimiert. Grundlage dafür bildet die im Folgenden dargestellte Berechnung des Minimums der Auszahlungsfunktion

$$CF(r_d) = m \cdot a \cdot r_d \cdot c_d + (c_b + c_s \cdot a) \cdot r_c \cdot m \cdot \frac{a \cdot (e - r_d \cdot \varepsilon)}{\left(1 - a \cdot (e - r_d \cdot \varepsilon)\right)} \cdot \beta$$

mit

$$\mathbb{D} = \left\{ r_d \in \mathbb{R} | 0 \leq r_d \leq 1 \wedge a \cdot (e - \varepsilon) < 1 \wedge r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} \right\}$$

Durch das Gleichsetzen der ersten Ableitung

$$\frac{dCF(r_d)}{dr_d} = m \cdot a \cdot \left(c_d - \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{\left(1 - a \cdot (e - r_d \cdot \varepsilon)\right)^2} \right)$$

mit 0 und dem Auflösen dieser Gleichung nach r_d ergeben sich zwei Kandidaten für Extremwerte:

$$\frac{dCF(r_d)}{dr_d} \stackrel{\text{def}}{=} 0$$

$$\Rightarrow r_{d,1} = \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} + \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon}$$

$$r_{d,2} = \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} - \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon}$$

Es ist leicht ersichtlich, dass $r_{d,2}$ für alle zulässigen Werte der gegebenen Parameter außerhalb des Definitionsbereichs $\mathbb{D}$ liegt:

$$r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} > r_{d,2}$$

Dagegen gilt für $r_{d,1}$:

$$r_d > \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} < r_{d,1}$$

Damit liegt $r_{d,1}$ unter folgenden Bedingungen innerhalb des Definitionsbereichs $\mathbb{D}$:

(B.1) $$c_d \geq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{\left(1 - a \cdot (e - \varepsilon)\right)^2} \quad \Leftrightarrow \quad r_{d,1} \leq 1$$

(B.2) $$(a \cdot e \geq 1) \quad \vee \quad \left(a \cdot e < 1 \quad \wedge \quad c_d \leq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{(1 - a \cdot e)^2} \right) \quad \Leftrightarrow \quad r_{d,1} \geq 0$$

Unter den Bedingungen (B.1) und (B.2) bildet $r_{d,1}$ folglich den einzigen kritischen Punkt von $CF(r_d)$. Da zudem die zweite Ableitung

$$\frac{d^2 CF(r_d)}{{dr_d}^2} = \frac{2 \cdot m \cdot a^2 \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon^2 \cdot \beta}{\left(1 - a \cdot (e - r_d \cdot \varepsilon)\right)^3}$$

an diesem Punkt größer ist als 0, besitzt die Auszahlungsfunktion $CF(r_d)$ bei $r_{d,1}$ ein globales Minimum:

$$\frac{d^2CF(r_{d,1})}{dr_{d,1}^{\ 2}} = \frac{2 \cdot m \cdot a^2 \cdot c_d^{\ 2} \cdot \varepsilon}{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}} > 0$$

Abschließend müssen noch (B.1) und (B.2) bewiesen werden:

<u>Beweis zu (B.1):</u>

$$r_{d,1} = \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} + \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon} \leq 1$$

$$\Leftrightarrow \quad \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon} \leq 1 - \frac{e}{\varepsilon} + \frac{1}{a \cdot \varepsilon}$$

$$\Leftrightarrow \quad \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon} \leq \frac{1 - a \cdot (e - \varepsilon)}{a \cdot \varepsilon}$$

$$\Leftrightarrow \quad \frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{\sqrt{c_d}} \leq 1 - a \cdot (e - \varepsilon) \quad \text{mit} \quad a \cdot (e - \varepsilon) < 1 \text{ und } \sqrt{c_d} > 0$$

$$\Leftrightarrow \quad \frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{1 - a \cdot (e - \varepsilon)} \leq \sqrt{c_d}$$

$$\text{mit} \quad \sqrt{c_d} > 0 \text{ und } a \cdot (e - \varepsilon) < 1 \text{ und } \sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta} > 0$$

$$\Leftrightarrow \quad \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{\left(1 - a \cdot (e - \varepsilon)\right)^2} \leq c_d$$

q.e.d.

<u>Beweis zu (B.2):</u>

$$r_{d,1} = \frac{e}{\varepsilon} - \frac{1}{a \cdot \varepsilon} + \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon} \geq 0$$

$$\Leftrightarrow \quad \frac{\sqrt{c_d \cdot (c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{c_d \cdot a \cdot \varepsilon} \geq \frac{1 - a \cdot e}{a \cdot \varepsilon}$$

$$\Leftrightarrow \quad \frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{\sqrt{c_d}} \geq 1 - a \cdot e$$

Für $a \cdot e \geq 1$ gilt:

$$\frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{\sqrt{c_d}} > 0 \geq 1 - a \cdot e$$

Für $a \cdot e < 1$ ist die Ungleichung erfüllt g.d.w.

$$\frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{\sqrt{c_d}} \geq 1 - a \cdot e \quad \text{mit} \quad 1 - a \cdot e > 0$$

$$\Leftrightarrow \quad \sqrt{c_d} \leq \frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{1 - a \cdot e}$$

$$\text{mit} \quad \sqrt{c_d} > 0 \text{ und } \frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{1 - a \cdot e} > 0$$

$$\Leftrightarrow \quad c_d \leq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{(1 - a \cdot e)^2}$$

Daraus folgt:

$$(a \cdot e \geq 1) \quad \vee \quad \left(a \cdot e < 1 \quad \wedge \quad c_d \leq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{(1 - a \cdot e)^2} \right)$$

$$\Rightarrow \quad \frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{\sqrt{c_d}} \geq 1 - a \cdot e \quad \Leftrightarrow \quad r_{d,1} \geq 0$$

Darüber hinaus gilt:

$$(a \cdot e < 1) \quad \wedge \quad \left(a \cdot e \geq 1 \quad \vee \quad c_d > \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{(1 - a \cdot e)^2} \right)$$

$$\Rightarrow \quad \frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{\sqrt{c_d}} < 1 - a \cdot e \quad \Leftrightarrow \quad r_{d,1} < 0$$

Daraus folgt wiederum:

$$(a \cdot e \geq 1) \quad \vee \quad \left(a \cdot e < 1 \quad \wedge \quad c_d \leq \frac{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}{(1 - a \cdot e)^2} \right)$$

$$\Leftarrow \quad \frac{\sqrt{(c_b + c_s \cdot a) \cdot r_c \cdot \varepsilon \cdot \beta}}{\sqrt{c_d}} \geq 1 - a \cdot e \quad \Leftrightarrow \quad r_{d,1} \geq 0$$

q.e.d.