Ein Kosten-Nutzen-Modell für die Softwareprüfung

Von der Fakultät Informatik, Elektrotechnik und Informationstechnik der Universität Stuttgart zur Erlangung der Würde eines Doktors der Naturwissenschaften (Dr. rer. nat.) genehmigte Abhandlung

Vorgelegt von

Tilmann Hampp

aus Ludwigsburg

Hauptberichter:	Prof. Dr. rer.nat. J. Ludewig
Mitberichter:	O. Univ.-Prof. Dr. R. Mittermeir
Tag der mündlichen Prüfung:	19. Juli 2010

Institut für Softwaretechnologie der Universität Stuttgart

2010

Danksagung

Zu dieser Arbeit haben viele Personen beigetragen, denen ich an dieser Stelle danken möchte. Mein ganz besonderer Dank gilt meinem Doktorvater Prof. Jochen Ludewig, der diese Arbeit ermöglicht hat. Seine Sicht auf die Software-Qualitätssicherung, seine Anregungen und seine Kritik haben diese Arbeit geprägt. Mein besonderer Dank geht auch an Prof. Roland Mittermeir für sein Interesse an der Arbeit, die freundliche Übernahme des Zweitgutachtens und die wertvollen Hinweise. Prof. Erhard Plödereder danke ich für seinen Einsatz als Zweitprüfer.

Herzlich danken möchte ich auch allen Kollegen und Kolleginnen der Abteilung Software Engineering für die angenehme Zusammenarbeit, die ermutigenden Gespräche und die anregenden Diskussionen.

Die Partner in der Industrie haben mir in den Gesprächen und mit den Projektdaten einen tiefen Einblick in ihre Projekte gewährt. Vielen Dank! Ebenso danke ich den Teilnehmern und Hilfskräften des Software-Praktikums 2007 für die Datenerhebung und die Unterstützung bei der Analyse. Mein Dank geht auch an die Studenten, die durch ihre Diplom- und Studienarbeiten und Fachstudien zu dieser Arbeit beigetragen haben.

Nicht zuletzt danke ich allen, die mich während der Arbeit begleitet haben, für ihr Vertrauen, ihre Geduld und ihre Unterstützung.

Zusammenfassung

Prüfungen können große Teile des Budgets eines Software-Projekts aufzehren, erlauben aber, die Produktqualität zu beurteilen und zu verbessern. Sie dürfen nicht vernachlässigt werden, da Defizite der Produktqualität nach Projektende teuer werden können. Projektleiter und Verantwortliche für die Qualität müssen bereits in der Planung über Prüfungen entscheiden. Sie sind in einer schwierigen Situation, weil sie dabei viele komplexe und langfristig wirkende Entscheidungen über Prüfungen und über einzelne Parameter der Prüfungen treffen müssen. Die Kosten der Prüfungen sind früh sichtbar und messbar. Im Gegensatz dazu wird der Nutzen durch schwierig zu messende Qualitätsverbesserungen erreicht, die zu langfristigen Einsparungen führen. Zusätzlich hängen Kosten und Nutzen von der Projektsituation ab. Für jedes Projekt ist darum ein individueller Kompromiss zwischen den Kosten für Prüfungen und ihrem Nutzen nötig, so dass minimale Gesamtkosten erreicht werden.

Um diese Entscheidungen zu unterstützen, wird in dieser Arbeit ein Kosten-Nutzen-Modell für Softwareprüfungen, CoBe, entwickelt und validiert. Mit diesem Modell kann untersucht und prognostiziert werden, wie sich Entscheidungen über Prüfungen und über einzelne Parameter der Prüfungen auswirken. Dazu werden die Entscheidungen und die Projektsituation durch Modelleingaben dargestellt. Die Modellresultate sind die Wirkungen dieser Entscheidungen: die Kosten, die durch die Prüfung entstehen, und der daraufhin erreichte Nutzen durch eingesparte Kosten. Kosten und Nutzen zeigen sich während des Projekts, während der Wartung des Produkts und beim Einsatz des Produkts. Damit Kosten und Nutzen abgewogen und Gesamtkosten minimiert werden können, werden die Modellresultate als Geldwerte berechnet. Zur Projektplanung werden Kosten und Nutzen durch Aufwand, Dauer und Personalbedarf einzelner Aktivitäten dargestellt. Dazu enthält CoBe feingranulare Prüfungsmodelle aus einzelnen, quantitativen Wirkungszusammenhängen.

Die Validierung des Modells erfolgte mit Daten aus Software-Projekten. Dabei wurden einzelne Zusammenhänge und das gesamte Modell mit Daten aus über 20 studentischen Projekten geprüft. CoBe ist mit Daten aus zwei iterativen Industrieprojekten mit umfangreicher, paralleler Entwicklung validiert. Das Modellverhalten wird durch Sensitivitätsanalyse untersucht, zusätzlich wird das Kosten-Optimum analysiert. Die Validierung zeigt, dass CoBe ausreichend genau beschreibt, wie sich Entscheidungen über Prüfungen auswirken. Da die Resultate der studentischen Projekte deutlich streuen, ergibt sich eine gewisse Abweichung zwischen den Projektresultaten und den Modellresultaten. Die Resultate sind für die beiden Industrieprojekte genauer. Deutlich wird, dass CoBe für eine bestimmte Umgebung kalibriert werden muss, damit die Resultate ausreichend genau sind. Dazu sind wenige Daten aus abgeschlossenen Software-Projekten notwendig. Die Daten sind oft verfügbar, da sie häufiger als andere Daten erhoben werden. Die Validierung zeigt, dass CoBe gut verallgemeinerbar ist. Die Daten, die für den Einsatz von CoBe notwendig sind, sind in Projekten verfügbar, können gemessen oder erfragt werden.

Abstract

Software quality assurance can consume large parts of a software project's budget. On the other hand, quality assurance permits product quality to be assessed and improved. Cutting quality assurance investments may lead to increased costs after delivery.

Project managers and quality managers have to decide on quality assurance while planning and running a project. They have to make many complex and far-reaching decisions on reviews, tests, and their parameters without having the necessary information available. In particular, quality-related information is hard to get because quality improvements appear as long-term savings, whereas the costs of reviews and tests can be measured early on. As every project has its own special characteristics, a tailored trade-off between costs and benefits of quality-assurance activities is needed to minimise total costs.

In this work CoBe, a quantitative cost-benefit model to support these decisions, is developed and validated. CoBe is able to analyse and predict the effects that decisions on quality assurance activities and on their parameters will have. To do so, decisions on quality assurance and project characteristics are modelled as inputs. The output of the model consists of the costs and benefits resulting from these decisions. Costs and benefits occur during the project, in maintenance, and during product usage. For the purpose of comparing costs and benefits and minimising overall costs, the model results are expressed in monetary values. For project planning, costs and benefits are expressed as effort, duration, and staff of single activities. For calculating the results, CoBe uses fine-grained models built on single quantitative relationships.

CoBe was validated against real-world software project data. Single relationships and the entire model were examined using data from more than 20 student projects. CoBe was validated using data from two iterative industry projects that used extensive parallel development. Model behaviour is subjected to sensitivity analysis, and the optimum cost is analysed. The validation shows that CoBe describes the effects of decisions on quality assurance sufficiently accurate. As student projects scatter, model results differ from project results up to a certain extent. Results are more accurate for industry projects. It is evident that CoBe needs to be calibrated for a certain environment. However, only few data from past projects is required for this. The necessary data is readily available in most projects. Results indicate that CoBe is generalisable, and that the necessary data is either available in the projects or can be measured or obtained by enquiry.

Kapitel 1

Einleitung und Überblick

1.1 Motivation

In einem Software-Projekt können die Kosten, die für Prüfungen und für Nacharbeit nach Prüfungen anfallen, einen großen Teil des Projektbudgets ausmachen. Bereits die Nacharbeit nach Prüfungen kann 40 % des Projektaufwands oder mehr kosten (Haley et al., 1995; Ellims et al., 2006). Für Prüfungen gibt es viele Möglichkeiten: Software kann von Menschen etwa mit technischen Reviews, Walkthroughs oder einfachen Stellungnahmen begutachtet werden. Programme können mit unterschiedlichen Techniken auf unterschiedlichen Integrationsebenen (oder Teststufen nach Spillner und Linz, 2003) getestet werden. Je nachdem, welche Prüfungen wie intensiv durchgeführt werden, entstehen niedrigere oder höhere Kosten.

Darum muss bei der Planung und Durchführung von Projekten entschieden werden: Wie viel soll geprüft werden? Welche Prüfungen sollen durchgeführt werden? Wie intensiv, nach welchen Kriterien sollen diese Prüfungen durchgeführt werden? Daran schließen sich weitere Fragen an: Wie lange werden Prüfung und Korrektur dauern? Wie viele Mitarbeiter werden benötigt? Wie wird sich die Prüfung auf die Qualität auswirken? Reicht eine günstigere und weniger intensive Prüfung aus?

Diese Fragen müssen bereits bei der Planung eines Projekts beantwortet werden. Der Projektleiter, der über diese Fragen entscheidet, muss einen Kompromiss zwischen Termin, Kosten und Qualität (Kerzner, 2006) oder Dauer, Kosten, Umfang und Qualität (Yourdon, 1995) wählen. Dargestellt wird dies als Dreieck oder Viereck (Abbildung 1).

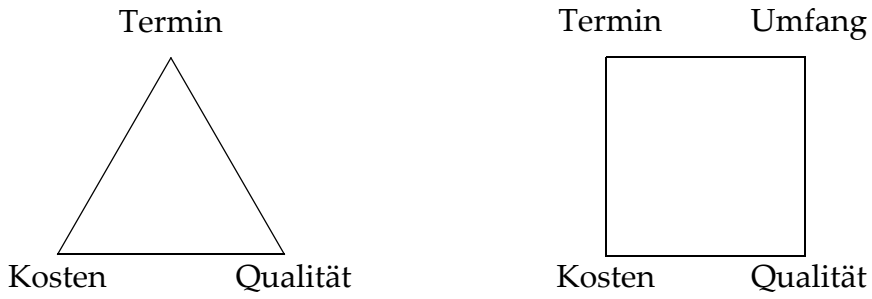

Abb. 1: Der Kompromiss des Projektleiters

Abbildung 1 illustriert, dass diese Dimensionen voneinander abhängen: Die Änderung der einen Dimension führt zu Änderungen an anderen Dimensionen. Wird also der Termin vorgezogen, führt dies zu höheren Kosten, schlechterer Qualität und geringerem Umfang; zumindest eine dieser Dimensionen ist betroffen. Die Bedingungen, Einschränkungen und Ziele gibt der Kunde vor, der Projektleiter sucht für diese Situation den optimalen Kompromiss.

Dieser Kompromiss lässt sich als Optimierungsproblem beschreiben, wenn die Kosten für die Prüfung und die Kosten für Qualitätsdefizite (Fehlerkosten) auf eine gemeinsame Kostenskala abgebildet werden. Abbildung 2 zeigt den Zusammenhang zwischen Qualität und Gesamtkosten (Kerzner, 2006). Dieser stammt aus der Fertigung von Gütern (Juran, 1962; Juran und Godfrey, 1998) und wird auf die Software-Entwicklung übertragen (Krasner, 1998). Die Abbildung stellt dar, dass die Fehlerkosten sinken, je besser die Qualität ist. Um die Qualität zu verbessern, muss in die Qualitätssicherung investiert werden. Die Summe dieser beiden Kosten sind die Qualitätskosten insgesamt. Dabei wird angenommen, dass das Optimum der Qualitätskosten nicht bei der besten Qualität liegt, sondern dass das Optimum durch einen Kompromiss zwischen den Investitionen in die Qualitätssicherung und den Fehlerkosten erreicht wird (Juran und Godfrey, 1998; Krasner, 1998).

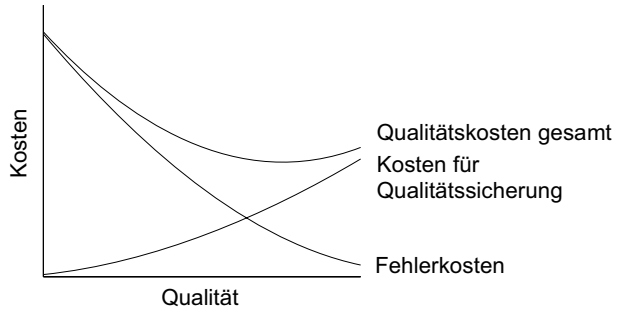

Abb. 2: Optimierung der Qualitätskosten (Kerzner, 2007)

Dieses Problem lässt sich aus mehreren Gründen intuitiv nicht optimal lösen: Prüfungen wirken indirekt, weil mit Prüfungen die Qualität beurteilt und verbessert werden kann. Qualität und Qualitätsverbesserungen sind schwierig zu bewerten. Sie werden erst langfristig als Einsparungen sichtbar. Es muss also ein großer Zeitraum überblickt werden, um diese Einsparungen zu beobachten. Zusätzlich sind die Einsparungen nicht direkt sichtbar, sondern können nur durch Vergleich mit einem Fall ohne Prüfungen oder mit weniger intensiven Prüfungen erfasst werden. Die Zusammenhänge, die durchschaut werden müssen, sind also komplex. Die Entscheidungen müssen überwiegend schon in der Planung getroffen werden. Dabei fehlen aber wichtige Informationen über das Projekt und das Produkt. Beispielsweise kann der Umfang der zu entwickelnden Artefakte nur geschätzt werden. Prozessverbesserungsmethoden wie CMMI (CMMI Product Team, 2002) oder SPICE (Hörmann et al., 2006) beantworten die Fragen nach optimalen Prüfungen und Prüfparametern

nicht konkret. Projekte sind unterschiedlich, so dass eine allgemeingültige Lösung nicht möglich ist.

1.2 Lösungsansatz

Für die Lösung dieses Entscheidungsproblems wird in dieser Arbeit das quantitative Modell CoBe entwickelt. Es unterstützt die Entscheidungen, die über Software-Prüfungen in einem Softwareprojekt getroffen werden müssen. Der Zweck von CoBe ist, den Nutzen und die Kosten von Software-Prüfungen zu demonstrieren, im Nachhinein für bestimmte Projekte darzustellen, zu vergleichen und zu prognostizieren, um zukünftige Projekte zu planen. Dazu werden die Entscheidungen als Eingaben und ihre Wirkungen als Ausgaben dargestellt:

Mit dem Modell wird der Handlungsspielraum, der durch die Wahl der Prüfungen und Prüfparameter gegeben ist, durch Modelleingaben abgebildet. Die speziellen Prozess- und Produktmerkmale werden durch weitere Eingaben in das Modell abgebildet. Die Modellresultate sind Kosten und Nutzen, dargestellt wie folgt: Durch eine Prüfung entstehen Kosten. Der Nutzen ist durch diejenigen Kosten bestimmt, die dank der Prüfung entfallen. Das Modell bildet kurzfristige Auswirkungen im Projekt und langfristige Auswirkungen von Prüfungen ab. Prüf- und Korrekturkosten, organisatorische Kosten, Folgekosten für Kunden und Benutzer und Kosten für die korrektive Wartung werden betrachtet. Kosten und Nutzen werden für einzelne Aktivitäten berechnet, um die Planung einzelner Arbeitspakete mit Aufwand, Dauer und Personalbedarf zu unterstützen. Die Resultate werden zusammengefasst und auf einer gemeinsamen Skala, als Geldwerte, dargestellt.

CoBe ist aus einzelnen Zusammenhängen aufgebaut, die als Funktionen beschrieben werden. Diese Zusammenhänge sind weitgehend empirisch belegt und empirisch quantifiziert. Das Modell bietet die Möglichkeit, diese Zusammenhänge anzupassen und zu ändern. Einzelne Prüfungen können unabhängig von anderen Teilen des Modells modelliert und angepasst werden. Somit ist das Modell generisch genug, damit es an spezielle Projektsituationen angepasst werden kann. Es ist aber auch konkret genug, dass es direkt eingesetzt werden kann. Für den Modelleinsatz wird ein iteratives Vorgehen vorgeschlagen, das sowohl Prozessverbesserungen als auch Modellverbesserungen ermöglicht.

1.3 Überblick

Die Arbeit gliedert sich wie folgt: Kapitel 2 enthält die grundlegenden Begriffe, die in der Arbeit verwendet werden: Modelle, Metriken, Entscheidungsmodelle, Software-Qualität, Software-Projekte und Software-Qualitätssicherung.

In Kapitel 3 wird der Lösungsansatz aus Problemen hergeleitet, die bei Entscheidungen über Qualitätssicherung in Software-Projekten auftreten: Kosten von Prüfungen müssen gegen schwierig zu bewertende Qualitätsverbesserungen aufgewogen wer-

den. Daraus leitet sich die Idee der Arbeit, der Lösungsansatz ab. Der iterative Modelleinsatz und der Modellierungsansatz werden festgelegt.

In Kapitel 4 werden die Arbeiten diskutiert, auf denen das Modell aufbaut. Aus dem QS-Modell (Drappa, 1998) des SESAM-Systems und aus dem Kostenschätzverfahren COCOMO II (Boehm, 2000) können Zusammenhänge direkt übernommen werden. Jones (1996 und 2007) enthält Daten, mit denen Zusammenhänge quantifiziert werden können. CoBe ergänzt vorhandene Arbeiten, weil es erlaubt, die Auswirkungen konkreter, detaillierter Prüfparameter zu untersuchen, und weil es langfristige Kosten darstellt, insbesondere Folgekosten beim Produkteinsatz. Die Analyse für die Zusammenhänge des Modells wird in Kapitel 5 dargestellt.

Kapitel 6 zeigt das Modell CoBe, das aus einem Basismodell, Reviewmodellen und Testmodellen besteht. Diese Modelle sind als Zusammenhänge quantitativ durch Gleichungen beschrieben. Die Quantifizierung beruht auf Datensammlungen, empirisch belegten Modellen und einzelnen Untersuchungen über Prüfungen.

In Kapitel 7 wird die Realisierung des Modells als Tabellenkalkulation und als Java-Programm sowie die Erprobung des Modells beschrieben. Vorgehen und Kriterien für die Validierung werden festgelegt. Ausgewählte Zusammenhänge in CoBe werden mit Daten aus studentischen Projekten überprüft. Dann werden die Modellresultate mit den Istwerten dieser Projekte verglichen. Modelldefizite werden sichtbar. CoBe wird für diese konkrete Situation kalibriert. Dann stimmen Modellresultate und Istwerte gut überein.

Die Validierung erfolgt mit Daten aus zwei Industrieprojekten (Kapitel 8). Die iterativ und parallel ablaufenden Projekte werden auf CoBe abgebildet. Die Modellresultate stimmen gut mit den Istwerten überein. Die Erprobung und Validierung beruht auf anderen Daten als die Modellbildung und Quantifizierung: Während das Modell auf empirischen Studien und Daten aus der Literatur basiert, werden zur Validierung Werte aus konkreten Projekten verwendet. Die Sensitivitätsanalyse und die Analyse optimaler Lösungen ergänzen die Validierung. Den Modelleinsatz demonstriere ich zusätzlich mit Beispielen. Dazu gehört, dass der Nutzen, der in der Realität nicht direkt sichtbar ist, nachträglich berechnet und gezeigt werden kann. Die entfallenden Wartungsaufwände und der entfallende Schaden für Benutzer werden sichbar. Kosten und Nutzen werden vergleichbar, so dass gezeigt werden kann, welche Prüfung in welcher Situation kostengünstiger ist.

Kapitel 9 fasst die Resultate zusammen. Das Modell CoBe wird bewertet und von anderen Modellen abgegrenzt. Kosten, Nutzen und Grenzen des Modells werden diskutiert. Der Ausblick zeigt, wie auf das Modell aufgebaut werden kann.

Kapitel 2

Grundlagen und Begriffe

In diesem Kapitel werden die Grundlagen, auf denen die Arbeit aufbaut, dargestellt. Zentral ist dabei der Begriff des Modells. Metriken sind spezielle, quantitative Modelle. Begriffe für Kosten und Nutzen, Software-Qualität, Software-Projekt und Software-Qualitätssicherung werden geklärt.

2.1 Modelle

Jedes Modell besitzt das Abbildungsmerkmal, das Verkürzungsmerkmal und das pragmatische Merkmal.

Def. **Abbildungsmerkmal.** Modelle sind stets Modelle von etwas, nämlich Abbildungen, Repräsentationen natürlicher oder künstlicher Originale, die selbst wieder Modelle sein können (Stachowiak, 1973). Das Original kann tatsächlich vorhanden, geplant oder fiktiv sein (Ludewig und Lichter, 2007).

Def. **Verkürzungsmerkmal.** Modelle erfassen im allgemeinen nicht alle Attribute des durch sie repräsentierten Originals, sondern nur solche, die den jeweiligen Modellerschaffern und/oder Modellbenutzern relevant scheinen (Stachowiak, 1973). Die präterierten Attribute fallen weg, sie werden verkürzt und nicht in das Modell abgebildet. Abundante Attribute sind nur im Modell, nicht im Original vorhanden (Ludewig und Lichter, 2007).

Def. **Pragmatisches Merkmal.** Modelle sind ihren Originalen nicht per se eindeutig zugeordnet. Sie erfüllen ihre Ersetzungsfunktion a) für bestimmte – erkennende und/oder handelnde, modellbenutzende – Subjekte, b) innerhalb bestimmter Zeitintervalle und c) unter Einschränkung auf bestimmte gedankliche oder tatsächliche Operationen (Stachowiak, 1973). Modelle können unter bestimmten Bedingungen das Original für bestimmte Fragestellungen ersetzen (Ludewig und Lichter, 2007).

Abbildung 3 illustriert das Abbildungsmerkmal mit den Attributen des Originals, die vom Modell erfasst werden. Die präterierten Attribute (weiß dargestellt in der Abbildung links) werden nicht erfasst, die abundanten Attribute (hellgrau dargestellt in der Abbildung rechts) kommen im Modell dazu.

Deskriptive Modelle bilden das Original ab. Diese Beschreibung kann nachträglich oder im Voraus erfolgen. Erfolgt die Beschreibung im Voraus, dann wird von einem prognostischen Modell oder Prognosemodell gesprochen. Präskriptive Modelle sind

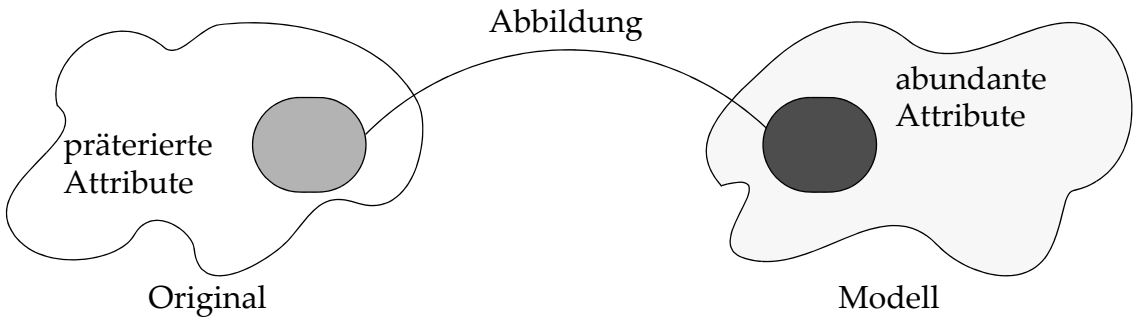

Abb. 3: Original und Modell nach Stachowiak

Vorgaben für das Original. Mit einem explorativen Modell (Abbildung 4) können die Folgen von Entscheidungen beurteilt werden, bevor die Realität verändert wird. Dazu wird zuerst der Ist-Zustand modelliert, das deskriptive Modell rechts oben in Abbildung 4. Dann wird das Modell anstatt der Realität verändert (Modellmodifikation). Sobald man mit dem neuen Modellzustand zufrieden ist, kann das geänderte Modell den neuen Zustand für die Realität vorgeben (Präskriptives Modell rechts unten in Abbildung 4).

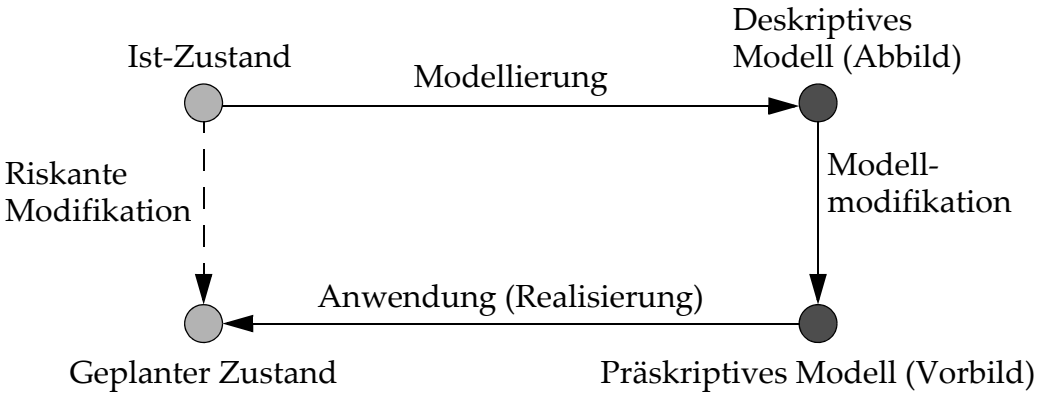

Abb. 4: Explorativer Einsatz eines Modells (Ludewig und Lichter, 2007)

2.2 Deduktive und induktive Modelle

Erklärungs- und Prognosemodelle werden aus einzelnen Aussagen zusammengesetzt, um für konkrete Situationen Folgerungen zu ziehen oder Prognosen zu erstellen (Opp, 2005). Diese Aussagen erklären den Zusammenhang zwischen zwei Sachverhalten (Schnell et al., 2005) und werden als Hypothesen, Gesetze oder Theorien bezeichnet. Sie sind nicht aus logischen Gründen wahr oder falsch, es sind empirische Aussagen. Sie können prinzipiell falsifiziert werden, lassen sich aber nicht beweisen.

Modelle bestehen aus quantitativen Zusammenhängen oder binären Zusammenhängen, die entweder “Ja” oder “Nein” ergeben.

Diese Aussagen sind entweder deterministisch oder nicht-deterministisch. Deterministische Aussagen treffen mit Sicherheit zu. Daraus entstehen deduktive Modelle, aus denen Aussagen logisch abgeleitet werden können. Nicht-deterministische Aussagen treffen mit einer bestimmten Wahrscheinlichkeit zu. Sie werden als statistische oder probabilistische Aussagen bezeichnet. Mit statistischen Aussagen ist eine deduktive Ableitung nicht möglich, es können aber Wahrscheinlichkeitsaussagen getroffen werden. Dies bedeutet, dass eine aus der Theorie abgeleitete Aussage mit einer bestimmten Wahrscheinlichkeit zutrifft. Modelle aus statistischen Aussagen werden als induktive Modelle bezeichnet.

2.3 Metriken

Metriken sind im Software-Engineering Modelle, die das Original auf eine Größe verkürzen. In IEEE 610 (1990) wird Metrik und Qualitätsmetrik definiert:

Def. **metric.** A quantitative measure of the degree to which a system, component or process possesses a given attribute. See also: quality metric. (IEEE 610, 1990)

Def. **quality metric.** (1) A quantitative measure of the degree to which an item possesses a given quality attribute. (2) A function whose inputs are software data and whose output is a single numerical value that can be interpreted as the degree to which the software possesses a given quality attribute. (IEEE 610, 1990)

Stevens (1946) verwendet den Begriff Messung als Zuweisung von numerischen Werten zu Objekten oder Ereignissen nach bestimmten Regeln. Der Begriff Metrik ist als Messung definiert, mit der eine Zahl oder ein Symbol einer Entität der realen Welt zugewiesen wird, um ein Merkmal zu charakterisieren (Fenton und Pfleeger, 1997; Zuse, 1998). Dabei kann auf skalare oder vektorielle Größen abgebildet werden (Ludewig und Lichter, 2007). Der konkrete Wert der Metrik für das Original wird als dessen Bewertung bezeichnet.

2.3.1 Skalen und Skalentypen

Die Abbildung erfolgt auf eine Skala, die anhand ihres Typs unterschieden werden (Stevens, 1946; Fenton und Pfleeger, 1997; Ludewig und Lichter, 2007):

Def. **Nominalskala.** Die möglichen Metrikwerte sind eine ungeordnete Menge.

Def. **Ordinalskala.** Die möglichen Metrikwerte sind eine geordnete Menge.

Def. **Intervallskala.** Die Differenz zwischen den (geordneten) Metrikwerten ist definiert.

Def. **Rationalskala.** Zusätzlich ist der Nullpunkt nicht willkürlich, sondern durch das Original definiert.

Def. **Absolutskala.** Die Metrikwerte sind direkt die Werte des Attributs im Original. Die Metrikwerte sind ganze Zahlen, die durch Zählen ermittelt werden.

Die Skalentypen liegen selbst auf einer Ordinalskala. Dazu muss die Absolutskala aber zur Rationalskala erweitert werden, weil mit natürlichen Zahlen beispielsweise nicht dividiert werden kann. Die Nominalskala ist der schwächste Skalentyp, die Rationalskala der stärkste, weil er die meisten Operationen erlaubt. Mit einer Nominalskala kann nur auf Gleichheit geprüft werden. Die Ordinalskala erlaubt, Perzentile und Median zu bilden, also Aussagen zur Häufigkeit zu treffen. Erst auf der Intervallskala kann gerechnet werden, Mittelwerte und Differenzen sind aussagekräftig. Um Verhältnisse zu bilden, wird der Nullpunkt benötigt, der auf der Rationalskala definiert ist.

2.3.2 Merkmale von Metriken

Metriken lassen sich anhand ihrer Abbildungsvorschrift unterscheiden (Ludewig und Lichter, 2007):

Def. **Objektive Metriken.** Die Abbildung erfolgt nach einem Algorithmus, durch Messung (oder Zählung).

Def. **Subjektive Metriken.** Die Abbildung erfolgt als Beurteilung durch Gutachter, verbal oder auf einer vorgegebenen Skala.

Def. **Pseudometriken.** Die Abbildung erfolgt durch Berechnung aus anderen Messungen, Schätzungen oder Beurteilungen, weil die Metrik nicht direkt gemessen werden kann.

Typisch für Pseudometriken sind Qualitätsbewertungen, weil das Attribut des Originals nicht präzise definiert ist, und Prognosen, weil das Attribut des Originals noch nicht real bewertet werden kann.

Wie bei den Modellen werden deskriptive und präskriptive Metriken unterschieden (Ludewig und Lichter, 2007). Eine deskriptive Metrik beschreibt einen Zustand, wie er ist (oder sein wird), eine präskriptive Metrik gibt den Zustand vor. Eine deskriptive Metrik kann prognostisch sein, dann wird ein zukünftiger Zustand beschrieben, oder diagnostisch, dann wird ein bestehender Zustand beschrieben.

Weil Pseudometriken eine bestimmte Interpretation implizieren, müssen sie validiert werden. Zuse (1998) und Fenton und Pfleeger (1997) definieren dazu die Repräsentationsbedingung. Sie besagt, dass Unterschiede zwischen Attributwerten des Originals als unterschiedliche Metrikwerte erhalten werden sollen.

Für Metriken zur Qualitätsbewertung definiert der IEEE-Standard 1061 (1998) Validierung als Aktivität, um die Übereinstimmung zwischen Metrik und Qualitätsmerkmal (quality factor) zu prüfen:

Def. **metric validation.** The act or process of ensuring that a metric reliably predicts or assesses a quality factor.

Für die Validierung werden unterschiedliche Validierungskriterien mit statistischen Maßen vorgeschlagen. Dazu gehört beispielsweise, dass Qualitätsmerkmal und Metrik korrelieren oder dass bei der Prognose eine bestimmte Mindestgenauigkeit erreicht wird.

Drappa (1998) definiert für die Validierung quantitativer (Simulations-)Modelle:

Def. **Validierung.** Validierung eines Simulationsmodells bedeutet festzustellen, ob das Simulationsmodell eine für den spezifizierten Zweck der Untersuchung hinreichend genaue Repräsentation des betrachteten realen Systems ist (Drappa, 1998).

2.4 Entscheidungen und Entscheidungstheorie

Bei einem Entscheidungsproblem muss eine Handlungsalternative aus mehreren Handlungen ausgewählt werden (Laux, 1998; von Nitzsch, 2002). Damit ein Entscheidungsproblem vorliegt, muss es mindestens zwei mögliche Handlungen geben. Die möglichen Handlungen müssen sich dadurch unterscheiden, dass ein Ziel unterschiedlich gut erreicht wird (Laux, 1998, S. 4).

2.4.1 Modelle in der Entscheidungstheorie

Die deskriptive Entscheidungstheorie beschäftigt sich mit der Frage, wie Entscheidungen ablaufen. Bei Entscheidungen durch Menschen spielen nicht nur rationale Ziele eine Rolle, weil Menschen Informationen beschränkt wahrnehmen und nur beschränkt verarbeiten können. Sie werden durch Gefühle und Gruppen beeinflusst (von Nitzsch, 2002). Die präskriptive Entscheidungstheorie versucht, vorzugeben, wie Entscheidungen rationaler getroffen werden können. Dazu wird das Entscheidungsproblem durch ein Entscheidungsmodell dargestellt (Laux, 1998). Das Modell besteht aus dem Entscheidungsfeld und den Zielfunktionen des Entscheiders. Das Entscheidungsfeld stellt dar:

- Die Handlungsalternativen werden als variierbare Größen dargestellt. Es können mehrere Entscheidungsvariablen relevant sein, so dass der Handlungsspielraum durch Werte-Tupel dieser Variablen beschrieben wird.
- Die Konsequenzen einer Handlung müssen in das Modell abgebildet werden. Es reicht aus, diejenigen Konsequenzen abzubilden, die für den Entscheider relevant sind. Diese Konsequenzen werden als Zielvariablen bezeichnet. Die konkrete Ausprägung wird als Ergebnis bezeichnet.
- Die Umweltzustände sind die Größen, die der Entscheider nicht beeinflussen kann. Diejenigen Größen, die sich auf die Zielvariablen auswirken, müssen berücksichtigt werden. Welche Variablen zu den Entscheidungsvariablen, welche zu den Umweltzuständen gehören, hängt vom Entscheider ab.

Entscheidungsprobleme werden unterschiedlich modelliert, je nachdem, wie viel über den Umweltzustand bekannt ist:

- Ist dem Entscheider der wahre Zustand bekannt, dann wird von Entscheidungen bei Sicherheit gesprochen.
- Ist der wahre Zustand nicht bekannt, aber die Wahrscheinlichkeit der möglichen Zustände, wird von Entscheidungen bei Risiko gesprochen.
- Bei Entscheidungen bei Unsicherheit im eigentlichen Sinne kann kein Wahrscheinlichkeitsurteil über die möglichen Zustände gefällt werden.

Die Zielfunktion stellt die Wünsche des Entscheiders dar. Sie beschreibt den zukünftigen Zustand, der angestrebt wird. In der Regel wird eine Maximierung angestrebt; andere Zielrichtungen wie die Minimierung oder ein angestrebter Wert lassen sich auf eine Maximierung zurückführen (Laux, 1998). Die Lösung des Entscheidungsproblems, d.h. die Identifikation der optimalen Handlungsalternative, kann, muss aber nicht Teil des Entscheidungsmodells sein. Abhängig von der Komplexität des Problems sind graphische oder numerische Lösungen möglich, aber auch Heuristiken können eingesetzt werden (Laux, 1998, S. 49).

2.4.2 Nutzen und Grenzen

Der Nutzen eines Entscheidungsmodells wird von von Nitzsch (2002) in der rationalen, vernünftigen Entscheidung gesehen. Nach Laux (1998) wird der Nutzen dadurch ereicht, dass die Ziele des Entscheiders besser erreicht werden.

Laux (1998) diskutiert die Grenzen der Entscheidungsmodelle. Sie entstehen durch die Verkürzung der Realität und die Subjektivität der Modellbildung und des Modelleinsatzes: Die Verkürzung erfolgt zwangsläufig bei der Modellbildung, aber auch, weil empirisches Wissen über die Realität fehlt. Typisch wird das Modell unübersichtlicher und seine Modellbildung teurer, je weniger verkürzt wird. Ein Entscheidungsmodell ist also ein Kompromiss zwischen den Kosten des Modells und dem damit erreichbaren Nutzen. Zusätzlich ist die Modellkonstruktion in mehrfacher Hinsicht subjektiv geprägt: So wird die Zielfunktion subjektiv bestimmt. Der Entscheider sieht nicht alle Alternativen, die möglich wären, und bewertet die Umweltzustände subjektiv. Die Handlungsalternativen sind in einer konkreten Situation beschränkt.

Entscheidungsmodelle bieten also eine Entscheidungshilfe, sie treffen aber nicht die Entscheidung. Laux (1998) schlägt darum vor, dass Entscheidungsmodelle iterativ eingesetzt und entwickelt werden. Insbesondere muss der Entscheider auf diejenigen Aspekte achten, die im Modell nicht berücksichtigt sind oder die auf Annahmen beruhen. Nicht berücksichtigte Aspekte können zusätzlich berücksichtigt werden; ein iteratives Vorgehen entsteht, wenn das Modell so überarbeitet wird, dass fehlende Aspekte ergänzt oder angenommene Aspekte geprüft und verbessert werden.

2.5 Kosten und Nutzen

2.5.1 Kosten- und Nutzenbegriffe

Der Kosten- und Nutzenbegriff leitet sich aus dem ökonomischen Prinzip ab. Weil Güter zur Bedürfnisbefriedigung knapp sind, besteht ein Spannungsverhältnis zwischen den eingesetzten Gütern und der damit erreichten Bedürfnisbefriedigung (Paul, 2007; Weber und Kabst, 2006; Corsten und Reiß, 1999). Daraus leiten sich Minimalprinzip und Maximalprinzip ab (Paul, 2007), die besagen, dass entweder mit einem gegebenen Aufwand ein maximaler Ertrag erzielt werden soll oder dass ein gegebener Ertrag mit einem minimalen Aufwand erreicht werden soll.

Aufwand und Ertrag sind nicht auf materielle Güter oder Geld beschränkt, sondern können auch immateriell sein (Weber und Kabst, 2006). Corsten und Reiß (1999) nennen für die Betriebswirtschaft vier Kategorien für Kosten und Nutzen, nämlich technische, wirtschaftliche, soziale und ökologische Ziele. Mühlenkamp (1994) erklärt: *"Im Sprachgebrauch der Ökonomen werden im weitesten Sinne Vorteile als "Nutzen" und Nachteile als "Kosten" bezeichnet."* Hanusch (1987) definiert Kosten und Nutzen für die Kosten-Nutzen-Analyse von Projekten der öffentlichen Hand:

Def. **Kosten** sind alle negativen Auswirkungen einer Maßnahme.

Def. **Nutzen** sind alle positiven Auswirkungen einer Maßnahme.

Krauß (2007) bezieht die Perspektive ein, weil die Auswirkungen einer Maßnahme aus einer Perspektive als Kosten, aus einer anderen Perspektive als Nutzen gesehen werden können. Im Rechnungswesen der Betriebswirtschaft müssen Kosten und Leistung durch Geld bewertet werden, dabei zählt Güterverbrauch zu den Kosten (Lück, 2004; Schneck, 2005). Gegenstück zu den Kosten ist die Leistung. Kosten und Leistung werden durch einen Wert (z.B. den Preis) bewertet; in der Investitionsrechnung werden Zahlungsströme betrachtet (Lück, 2004).

2.5.2 Kosten-Nutzen-Analyse und andere Verfahren

Mit der Kosten-Nutzen-Analyse sollen Entscheidungen über Projekte im öffentlichen Sektor unterstützt werden (Mühlenkamp, 1994; Nas, 1996). Im Gegensatz zum betriebswirtschaftlichen Rechnungswesen werden bei der Kosten-Nutzen-Analyse nicht die in aller Regel wirtschaftlichen Interessen eines Unternehmens, sondern die öffentlichen Interessen einer Gesellschaft berücksichtigt.

Überblick über die Kosten-Nutzen-Analyse

Ziel der Kosten-Nutzen-Analyse ist, die Wirkungen von Projekten zu vergleichen. Dazu sollen positive und negative Wirkungen für alle Betroffenen erfasst werden. Sie werden auf einer gemeinsamen Skala dargestellt, Kosten und Nutzen liegen also auf der selben Skala, unterscheiden sich aber im Vorzeichen. Nutzen wird durch negative Kosten dargestellt, Kosten durch negativen Nutzen. Bei der Kosten-Nutzen-Analyse werden in der Regel Geldwerte als Skala verwendet, prinzipiell sind aber auch andere

Größen möglich (Mühlenkamp, 1994). Wirkungen in der Zukunft werden abgezinst (Diskontierung). Es entsteht ein quantitatives Kosten-Nutzen-Modell, mit dem Wirkungen von Entscheidungen auf ein eindimensionales Güte- oder Entscheidungsmaß abgebildet werden. Damit können Alternativen geordnet werden. Eine Alternative kann empfohlen werden (Hanusch, 1987; Mühlenkamp, 1994; Nas, 1996).

Die Vorteile (Mühlenkamp, 1994) sind, dass Kosten und Nutzen von Maßnahmen erfasst, bewertet und sichtbar werden. Zusammenhänge werden klar, die Entscheidung wird transparent. Alle Betroffenen werden einbezogen. Kritikpunkte sind, dass Kosten und Nutzen monetär bewertet werden müssen, auch wenn die Wirkung nicht direkt in Geldwerten zu messen ist. Auch die Diskontierung ist problematisch, weil der Zinssatz nicht sicher ist und weil unklar ist, ob Wirkungen für Dritte und ob Umverteilungen diskontiert werden dürfen.

Die Kosten-Nutzen-Analyse kostet selber nicht unerheblichen Aufwand, insbesondere um das Modell aufzustellen. In einfachen Situationen mit überschaubaren Auswirkungen ist sie nicht notwendig. In komplexen Situationen mit hohen Kosten und hohem Nutzen, zahlreichen und langfristigen Auswirkungen über das Projekt hinaus ist der Aufwand gerechtfertigt (Mühlenkamp, 1994).

Kosten-Wirksamkeits-Analyse und Nutzwert-Analyse

Die Kosten-Wirksamkeits-Analyse und die Nutzwert-Analyse sind zwei weitere Verfahren, um Entscheidungen zu unterstützen (Hanusch, 1987). Die Kosten-Wirksamkeits-Analyse bewertet den Nutzen nicht monetär, sondern bezieht den Nutzen auf die Ziele, die erreicht werden sollen. Für die Bewertung sollen für den Nutzen geeignete Maße oder Indikatoren eingesetzt werden. Die Rangfolge kann nicht mehr eindeutig festgelegt werden, wenn mehrere Ziele berücksichtigt werden. Die Kosten-Wirksamkeits-Analyse kann nur weitgehend verwandte Alternativen ordnen.

Die Nutzwertanalyse betrachtet den Nutzen relativ zu Zielen. Dazu wird der Nutzen zerlegt und mit einem gewichteten Zielerfüllungsgrad bewertet. Kosten werden entweder als gleich für alle Alternativen vorausgesetzt oder als negativer Nutzen in die Bewertung aufgenommen. Bereits der Aufbau des Bewertungssystems ist subjektiv, die Entscheidung wird aber transparent und erfolgt systematisch. Wie die Kosten-Wirksamkeits-Analyse können nur ähnliche Alternativen mit gleichem Zielsystem verglichen werden.

Im Vergleich zur Kosten-Nutzen-Analyse sind Kosten-Wirksamkeits-Analyse und Nutzwertanalyse stärker subjektiv geprägt, weil die Beteiligten der Nutzwert- und Kosten-Wirksamkeitsanalyse die Ziele und ihre Gewichtung direkt bestimmen. Diese können zwar Interessen anderer Betroffener berücksichtigen, es besteht aber die Gefahr, dass mit den Zielen und ihrer Gewichtung das Ergebnis bewusst oder unbewusst verfälscht wird. Subjektive Werturteile spielen auch in der Kosten-Nutzen-Analyse eine Rolle. Die Subjektivität ist aber abgeschwächt (Mühlenkamp, 1994), weil die Auswirkungen und die Betroffenen identifiziert werden müssen und den Auswirkungen Geldwerte zugewiesen werden müssen.

2.6 Software-Projekte

Ein Software-Projekt besteht aus Aktivitäten mit dem Ziel, ein Software-Produkt an einen Kunden auszuliefern:

Def. **software project.** The set of work activities, both technical and managerial, required to satisfy the terms and conditions of a project agreement. (IEEE 1058, 1998; IEEE 1490, 2003)

Das Software-Projekt wird von Mitarbeitern des Herstellers durchgeführt (Ludewig und Lichter, 2007). Der Hersteller wird auch als Lieferant oder Auftragnehmer bezeichnet (V-Modell XT, 2004). Dabei entstehen Artefakte. Artefakte, die an den Kunden ausgeliefert werden, werden als Produkt bezeichnet (CMMI Product Team, 2002). Als Wartung werden Aktivitäten nach der Auslieferung bezeichnet; Wartungsarbeiten können aber bereits während des Projekts anfallen (Ludewig und Opferkuch, 2004).

Def. **maintenance.** (1) The process of modifying a software system or component after delivery to correct faults, improve performance or other attributes, or adapt to a changed environment. Syn.: software maintenance. [...] (2) The process of retaining a hardware system or component in, or restoring it to, a state in which it can perform its required function. [...] (IEEE 610, 1990)

2.6.1 Prozess

Der Begriff des Prozess wird definiert als Sequenz einzelner Schritte, mit denen ein Ziel erreicht werden soll. Der Begriff bezeichnet konkrete Schritte, aber auch ein Modell dieser Aktivitäten (Ludewig und Lichter, 2007). Die einzelnen Schritte können überlappen oder iterativ durchgeführt werden.

Def. **process.** (1) A sequence of steps performed for a given purpose; for example, the software development process. (IEEE 610, 1990)

Def. **software development process.** The process by which user needs are translated into a software product. The process involves translating user needs into software requirements, transforming the software requirements into design, implementing the design in code, testing the code, and sometimes, installing and checking out the software for operational use. Note: These activities may overlap or be performed iteratively. (IEEE 610, 1990)

Def. **process.** A set of interrelated activities, which transform inputs into outputs. (IEEE 12207.0, 1996)

Ludewig und Lichter (2007) unterscheiden zwischen Prozess und Prozessmodell. Das Prozessmodell beschreibt den Prozess, der einem Projekt zu Grunde liegt. Das Prozessmodell besteht im Kern aus einem Vorgehensmodell, ergänzt um Organisationsstrukturen, Vorgaben für Projektmanagement und Qualitätssicherung,

Dokumentation und Konfigurationsverwaltung. In Projekten sind zwei Zeiträume wichtig: Die Entwicklungsdauer und die Lebensdauer:

Def. **software development cycle.** The period of time that begins with the decision to develop a software product and ends when the software is delivered. This cycle typically includes a requirements phase, design phase, implementation phase, test phase, and sometimes, installation and checkout phase. Contrast with: software life cycle. (IEEE 610, 1990)

Def. **software life cycle.** the period of time that begins when a software product is conceived and ends when the software is no longer available for use. the software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and checkout phase, operation and maintenance phase, and, sometimes, retirement phase. Note: These phases may overlap or be performed iteratively. Contrast with software development cycle. (IEEE 610, 1990)

2.6.2 Rollen in Software-Projekten

Der Projektleiter ist für Planung, Steuerung und Durchführung des Projekts verantwortlich. Er wird durch den Qualitätssicherungs-Verantwortlichen (QS-Verantwortlichen) unterstützt (V-Modell XT, 2004):

Def. **project manager (PM).** The individual responsible for managing a project. (IEEE 1490, 2003).

Def. Der **QS-Verantwortliche** ist mit der Überwachung der Qualität im Projekt beauftragt. Er ist damit für die Qualität der Projektergebnisse verantwortlich. (V-Modell XT, 2004)

Die Rollen sind zu trennen (aus V-Modell XT, 2004): "*Die Rolle des QS-Verantwortlichen sollte nicht mit der Rolle des Projektleiters zusammengelegt werden, da dann Interessenkonflikte (Projektleiter zuständig für Zeit und Budget contra QS-Verantwortlicher zuständig für Qualität) entstehen können.*"

Die von einem Projekt Betroffenen werden als Stakeholder bezeichnet; Glinz und Wieringa (2007) diskutieren den Begriff und definieren:

Def. A **stakeholder** is a person or organization who influences a system's requirements or who is impacted by that system. (Glinz und Wieringa, 2007)

Def. **stakeholder.** Individuals and organizations that are actively involved in the project, or whose interests may be positively or negatively affected as a result of project execution or project completion. They may also exert influence over the project and its results. (IEEE 1490, 2003)

Zu den Stakeholdern gehören Kunde und Benutzer (CMMI Product Team, 2002; IEEE 1490, 2003), aber auch das Wartungspersonal (Glinz und Wieringa, 2007). Direkt betroffen sind nach Alexander und Robertson (2004) die Benutzer und das Wartungs-

personal. Zum Wartungspersonal gehören Wartungsingenieure oder Entwickler, die die Wartung durchführen (Ludewig und Opferkuch, 2004), und die Benutzerberatung. Ludewig und Lichter (2007) verwenden den Begriff "Klient" und diskutieren, dass die Unterscheidung zwischen Klienten und den Menschen, die im Auftrag eines Kunden an der Software arbeiten, nicht immer möglich ist. Im Folgenden verwende ich die genannte Definition für Stakeholder und die Begriffe Klient und Stakeholder synonym.

Im V-Modell sind Auftraggeber (Kunde) und Anwender (Benutzer) definiert:

Def. **Anwender.** Der Anwender nutzt das System zur Erfüllung seiner Fachaufgaben nach der Auslieferung. Er leitet aus seiner Erfahrung mit dem Einsatz und Betrieb sowie der Pflege und Wartung von Systemen Anforderungen an das Gesamtsystem ab und bringt entsprechende Änderungsvorschläge ein. (V-Modell XT, 2004)

Def. **Auftraggeber.** Unter einem Auftraggeber wird der Kunde im Rahmen einer Vertragssituation verstanden, also der Empfänger eines vom Auftragnehmer bereitgestellten Produkts. (V-Modell XT, 2004)

2.6.3 Projektleitung

Die Aufgaben des Projektleiters sind Planung, Organisation, Stellenbesetzung, Führung, Überwachung und Steuerung (Thayer und Christensen, 2002; Kerzner, 2006). Die Planung bestimmt, was von wem bis wann durchgeführt werden soll (Kerzner, 2006). Sie gibt den Ablauf des Projekts vor:

Def. **planning.** Predetermining a course of action for accomplishing organizational objectives. (Thayer and Christensen, 2002)

Überwachung und Steuerung bedeutet, die Realität mit dem Plan zu vergleichen, so dass bei Abweichungen vom Plan reagiert werden kann. Dies wird auch als Projektkontrolle bezeichnet:

Def. **controlling.** Establishing, measuring, and evaluating performance of activities towards planned objectives. (Thayer and Christensen, 2002)

2.6.4 Kosten in Software-Projekten

Typische Kostenkategorien in Projekten sind Arbeitskosten, Materialkosten, andere direkte Kosten und indirekte Kosten (Kerzner, 2006, S. 610). Kosten basieren auf den Projektaktivitäten (Kerzner, 2006, S. 544) und der zu leistenden Arbeit (Metzger und Boddie, 1996, S. 49). Wichtige Metriken für die Planung und Kontrolle sind Aufwand, Dauer und Personalbedarf (Boehm, 2000; Goethert et al., 1992; IEEE 1058, 1998). Diese Metriken werden im Folgenden als Planungsmetriken bezeichnet.

Der Arbeitsaufwand (kurz Aufwand, Effort) wird in Entwicklerstunden (Eh), Entwicklermonaten (EM) oder Entwicklerjahren (EJ) gemessen. Er bestimmt über die Personalkosten die Projektkosten (Jones, 2007, S. 13):

Def. **effort.** A staff-hour is an hour of time expended by a member of the staff. (IEEE 1045, 1992)

Def. **effort.** The number of labor units required to complete an activity or other project element. Usually expressed as staff hours, staff days, or staff weeks. Should not be confused with duration. (IEEE 1490, 2003)

Die Dauer wird zwischen zwei Zeitpunkten (Goethert et al., 1992) gemessen, als Arbeitsdauer definiert und von der kalendarischen Dauer unterschieden. Typische Einheiten sind Stunden (h), Tage (d), Monate (M) oder Jahre (J). Unterschiede zwischen Arbeitsdauer und kalendarischer Dauer zeigen sich beispielsweise bei der Umrechnung zwischen Tagen und Wochen, weil eine Woche 5 Arbeitstage hat, aber 7 Tage dauert.

Def. **duration (DU).** The number of work periods (not including holidays or other nonworking periods) required to complete an activity or other project element. Usually expressed as workdays or workweeks. Sometimes incorrectly equated with elapsed time. (IEEE 1490, 2003).

Def. **Personalbedarf** (syn. Mitarbeiterzahl, Zahl der Mitarbeiter). Die Zahl der benötigten Mitarbeiter.

Diese Metriken werden auf unterschiedlichen Abstraktionsebenen definiert. Die kleinste Einheit in einem Software-Projekt wird als Task bezeichnet. Bei der Planung werden die Aktivitäten (activity) definiert und geplant; es sind die einzelnen Arbeitspakete, die von den Mitarbeitern noch weiter unterteilt werden können. Aktivitäten werden zu Projektphasen (syn. Phase) zusammengefasst.

Def. **activity.** An element of work performed during the course of a project. An activity normally has an expected duration, an expected cost, and expected resource requirements. Activities can be subdivided into tasks. (IEEE 1490, 2003)

Def. **task.** A generic term for work that is not included in the work breakdown structure, but potentially could be a further decomposition of work by the individuals responsible for that work. Also, lowest level of effort on a project. (IEEE 1490, 2003)

Def. **project phase.** A collection of logically related project activities, usually culminating in the completion of a major deliverable. (IEEE 1490, 2003)

2.6.5 Kostenschätzung in Software-Projekten

Kostenschätzung bezeichnet die Prognose von Aufwand, Dauer, Personalbedarf und monetären Kosten (Fenton und Pfleeger, 1997). Die Kosten als Geldwerte ergeben sich aus den Kosten der Projektaktivitäten (Kerzner, 2006, S. 544) und der zu leistenden Arbeit (Metzger und Boddie, 1996, S. 49).

Als Bottom-up-Schätzung wird ein Vorgehen bezeichnet, bei dem zuerst Aufwand, Dauer und Personal der einzelnen Aktivitäten im Projekt geschätzt werden. Daraus

folgen Aufwand, Dauer, Personalbedarf und Kosten als Geldwerte für das gesamte Projekt. Bei der Top-down-Schätzung werden zuerst die Gesamtkosten des Projekts geschätzt und dann auf einzelne Aktivitäten aufgeteilt.

2.7 Qualität und Software-Qualität

2.7.1 Der Qualitätsbegriff

DIN 55350 (1995) nennt für den Begriff Qualität drei unterschiedliche Bedeutungen: Qualität kann eine neutrale Eigenschaft bezeichnen. Qualität kann "qualitativ hochwertig" bezeichnen. Qualität kann eine graduelle Eigenschaft oder Güte beschreiben (DIN 55350, 1995). Garvin (1988) nennt Blickwinkel auf die Qualität, die zu widersprüchlichen Zielen führen und begründet damit, dass eine einseitige Betrachtung zu nicht-optimalen Lösungen führt:

- Die transzendente Sicht: Qualität als wahrnehmbare, aber nicht präzis definierbare Eigenschaft, die durch Erfahrung erkannt wird.
- Die produkt-basierte Sicht: Qualität als objektive, messbare, inhärente Merkmale des Produkts.
- Die benutzer-basierte Sicht: Qualität als Bewertung durch Benutzer.
- Die hersteller-basierte Sicht: Qualität als Erfüllung von Anforderungen.
- Die wert-basierte Sicht: Auch Kosten und Preis werden betrachtet.

2.7.2 Software-Qualität

Software-Qualität wird durch Taxonomien (IEEE 1061, 1998) oder als Abwesenheit von Defiziten (ISO 9000, 2000; IEEE 1044, 1993) definiert.

Taxonomien der Software-Qualität

Taxonomien unterteilen den Qualitätsbegriff in einzelne Merkmale. Die Taxonomien von Ludewig und Lichter (2007) und des ISO/IEC-Standard 9126 (2001) unterscheiden zwischen Produkt- und Prozessqualität. Die Prozessqualität beeinflusst die Produktqualität, sie schafft günstige Voraussetzungen für eine hohe Produktqualität, ist aber weder eine Garantie noch eine zwingend notwendige Voraussetzung. Der Standard ISO/IEC 9126 (2001) unterteilt Qualität in interne Produktqualität, externe Produktqualität und Qualität im Einsatz. Der Standard stellt ein hierarchisches Qualitätsmodell für jeden dieser drei Aspekte bereit.

Ludewig und Lichter (2007) unterteilen Prozessqualität und Produktqualität in einer Taxonomie. Die Prozessqualität gliedert sich in Projektleistung, Planungssicherheit und innere Prozessqualität. Planungssicherheit enthält beispielsweise die Termineinhaltung, das Projektklima gehört zur inneren Prozessqualität. Die Produktqualität besteht aus Brauchbarkeit und Wartbarkeit. Zur Brauchbarkeit gehören Aspekte der

Bedienbarkeit, Nützlichkeit und Zuverlässigkeit. Prüfbarkeit, Änderbarkeit und Portabilität sind der Wartbarkeit zugeordnet.

Software-Qualität als Erfüllung von Anforderungen

Die ISO (ISO 9000, 2000) definiert Qualität bezüglich Anforderungen:

Def. **Qualität.** Grad, in dem ein Satz inhärenter Merkmale Anforderungen erfüllt.

Def. **Anforderung.** Erfordernis oder Erwartung, das oder die festgelegt, üblicherweise vorausgesetzt oder verpflichtend ist.

Def. **Fehler.** Nichterfüllung einer Anforderung.

Def. **Mangel.** Nichterfüllung einer Anforderung in Bezug auf einen beabsichtigten oder festgelegten Gebrauch.

Def. **Merkmal.** Kennzeichnende Eigenschaft.

Def. **Qualitätsmerkmal.** Inhärentes Merkmal eines Produkts, Prozesses oder Systems, das sich auf eine Anforderung bezieht.

Der Begriff der Anomalie oder Abweichung (IEEE-Std. 1044, 1993) entspricht dem Fehlerbegriff der ISO. Der IEEE-Standard 982.1 (2005) unterscheidet für Software die Fehlerursache (fault) und das Fehlersymptom (failure), außerdem den Irrtum, der der Fehlerursache zu Grunde liegt. Der Standard schränkt den Fehlerbegriff aber auf Fehlerursachen im Programm und Fehlersymptome beim Verwenden des Systems ein. Er stützt sich auf die Definitionen im IEEE Standard 610 (1990):

Def. **fault.** (1) A defect in a hardware device or component; for example, a short circuit or broken wire. (2) An incorrect step, process, or data definition in a computer program. (IEEE 610, 1990)

Def. **failure.** The inability of a system or component to perform its required functions within specified performance requirements. (IEEE 610, 1990)

Mit Fehlerentdeckung wird die Identifikation einer Abweichung bezeichnet, mit Korrektur die Änderung, mit der der Fehler entfernt, d.h. korrigiert, wird (Dunn, 1984).

Zuverlässigkeit ist über Fehlverhalten definiert:

Def. **reliability.** The ability of a system or component to perform its required functions under stated conditions for a specified period of time. (IEEE 982.1, 2005)

Def. **dependability.** Trustworthiness of a computer system such that reliance can be justifiably placed on the service it delivers. Reliability, availability, and maintainability are aspects of dependability. (IEEE 982.1, 2005)

2.8 Software-Qualitätssicherung

Software-Qualitätssicherung bezeichnet alle geplanten und systematischen Aktivitäten, die das Vertrauen in die Konformität zu technischen Anforderungen sichern (IEEE 610, 1990; Thayer und Christensen, 2002; Ludewig und Lichter, 2007). Qualitätssicherung ist projektbezogen und wird von der projektübergreifenden Prozessverbesserung (oder Qualitätsmanagement) abgegrenzt (Thayer und Christensen, 2002; Ludewig und Lichter, 2007).

Def. **quality assurance (QA).** (1) A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements. (2) A set of activities designed to evaluate the process by which products are developed or manufactured. Contrast with: quality control (1). (IEEE 610, 1990)

Qualitätssicherung besteht aus organisatorischen, konstruktiven und analytischen Maßnahmen (Ludewig und Lichter, 2007). Organisatorische Maßnahmen zielen auf eine systematische Entwicklung und Qualitätssicherung. Mit konstruktiven Maßnahmen sollen Defizite vermieden werden. Analytischen Maßnahmen enthalten Software-Prüfungen, sie ergänzen andere Maßnahmen. Software-Prüfungen finden entweder mechanisch oder nichtmechanisch, d.h. durch Menschen, statt. Die Software, die geprüft wird, wird als Prüfling bezeichnet. Zu den nichtmechanischen Prüfungen zählen Inspektionen (Fagan, 1976), technische Reviews (Freedman und Weinberg, 1982), Walkthroughs und Stellungnahmen (Freedman und Weinberg, 1982; Frühauf et al., 2006). Zu den mechanischen Prüfungen, für die ein Rechner benötigt wird, gehören die dynamischen Tests (z.B. in Liggesmeyer, 2002). Dazu gehören auch statische Analysen (Spinellis, 2006; Louridas, 2006), mit denen Software mit Regeln geprüft, die Konsistenz überprüft oder quantitative Merkmale erfasst werden.

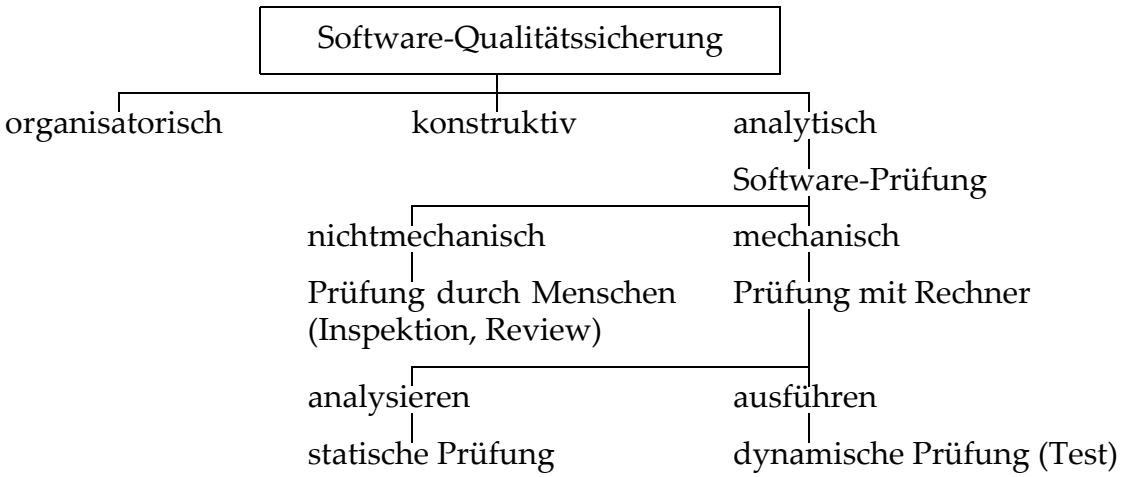

Abb. 5: Gliederung der Qualitätssicherung nach Ludewig und Lichter (2007)

2.9 Qualitätskosten und Software-Qualitätskosten

Software-Kosten können in reine Entwicklungskosten, Qualitätskosten und Kosten für die Wartung (ohne Qualitätskosten) unterteilt werden (Tomys, 1995; Ludewig und Lichter, 2007). Für Qualitätskosten gibt es eine Reihe leicht unterschiedlicher Taxonomien (Ludewig und Lichter, 2007; Frühauf et al., 2006; Jalote, 2000; Krasner, 1998; Slaughter et al., 1998; Demirörs et al., 2000). Abgeleitet aus Juran (1962) und Juran und Godfrey (1998) wird definiert:

Def. **Fehlerverhütungskosten (Prevention Costs).** Kosten der Aktivitäten, mit denen Qualitätsdefizite verhindert werden sollen.

Def. **Prüfkosten (Appraisal Costs).** Kosten der Aktivitäten, mit denen die Qualität festgestellt werden soll und mit denen man sich versichern will, dass die Produktqualität ausreicht.

Def. **Fehlerkosten (Failure Costs).** Kosten, die durch Qualitätsdefizite entstehen. Diese Kosten können in interne und externe unterteilt werden. Interne Fehlerkosten fallen vor der Auslieferung des Produkts an. Externe Fehlerkosten fallen nach der Auslieferung des Produkts an.

Fehlerfolgekosten sind externe Fehlerkosten, die von Fehlern für Kunde und Benutzer verursacht werden. Externe Fehlerkosten fallen auch für den Hersteller an (Demirörs et al.; 2000), dazu gehören Kosten für den technischen Support, Wartungs- und Auslieferungskosten für Fehler. Fehlerbehebungskosten fallen an, um entdeckte Fehler zu entfernen (einschließlich Qualitätssicherung der Korrektur). Zusätzlich können Vertragsstrafen, Kosten für Produktrückrufe, Kosten, um den Kunden zu besänftigen, Markt- und Verkaufseinbußen, Garantie und Gewährleistung anfallen.

Kapitel 3

Die Idee eines Kosten-Nutzen-Modells für Prüfungen

In diesem Kapitel werden die Probleme beschrieben, mit denen Projektleiter und QS-Verantwortliche bei der Planung und Kontrolle der Software-Qualitätssicherung konfrontiert werden. Der Lösungsansatz, ein quantitatives Modell für Kosten und Nutzen von Prüfungen, wird dargestellt. Die Einbettung des Modells in die Projektplanung und -kontrolle und in Prozessverbesserungen wird gezeigt. Der Modellierungsansatz wird festgelegt.

3.1 Schwierigkeiten mit Entscheidungen über Qualitätssicherung

Projektleiter und QS-Verantwortliche treffen Entscheidungen bei der Planung und Steuerung eines Projekts (Kerzner, 2006; Thayer und Christensen, 2002, S. 217; V-Modell XT, 2004). Mit diesen Entscheidungen versucht der Projektleiter, das Projekt erfolgreich durchzuführen. Dazu ist ein Kompromiss zwischen Kosten, Dauer und Qualität notwendig, weil diese Größen voneinander abhängen und Ressourcen beschränkt sind (Kerzner, 2006). Für diesen Kompromiss werden Entscheidungen über die Qualitätssicherung gefällt. Die Entscheidungen sind schwierig, weil es sinnvoll ist, sie früh im Projekt zu treffen, aber nur unzureichend Informationen über die Qualität verfügbar sind:

- Je früher Entscheidungen getroffen werden, desto größer ist der Handlungsspielraum. Wenn mehr Zeit und mehr Ressourcen zur Verfügung stehen, können andere, auch umfangreichere Optionen gewählt werden. Kerzner (2006) zeigt dazu eine Studie des Departement of Defense (Abbildung 6 auf Seite 34). Die Projektphasen, die das Departement of Defense verwendet, sind auf der Zeitachse dargestellt. Das Projektmanagement übernimmt die Verantwortung nach der Konzeptdefinition (Conceptual Definition in Abbildung 6). Je weiter das Projekt fortgeschritten ist, desto weniger Entscheidungen können getroffen werden. Die Abbildung zeigt dies durch den prozentualen Anteil der Entscheidungen, die bis zu einem gewissen Zeitpunkt im Projekt getroffen worden sind (Decisions affecting life-cycle costs in Abbildung 6).
- Mit frühen Entscheidungen ist die Wirksamkeit der Handlungen am größten. Die Studie in Kerzner (2006) zeigt, dass die Einsparmöglichkeiten im Verlauf des Projekts abnehmen (Cost reduction opportunity in Abbildung 6).

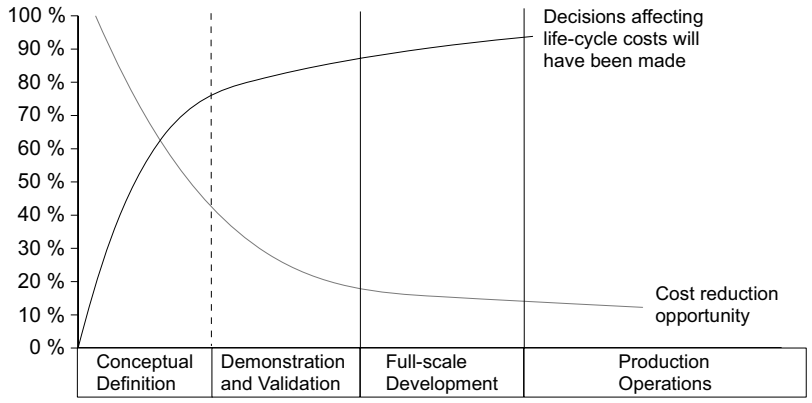

Abb. 6: Entscheidungs- und Einsparmöglichkeiten nach Kerzner (2006)

Weil der Handlungsspielraum und die Wirksamkeit der Handlungen im Verlauf des Projekts abnehmen, folgt, dass die wichtigsten Entscheidungen in der Planung getroffen werden. Zu diesem Zeitpunkt sind aber nur wenig Informationen bekannt. Wie stark sich eine Entscheidung auswirkt, kann darum nur mit beträchtlicher Unsicherheit geschätzt werden. Boehm (2000) gibt an, dass Schätzungen bei der Projektplanung typisch um den Faktor 2 von den tatsächlichen Werten abweichen.

Die Entscheidungen des Projektleiters und des QS-Verantwortlichen über die Qualitätssicherungsmaßnahmen bestimmen die Software-Qualität, weil Qualitätssicherungsmaßnahmen die Prozessqualität und die Produktqualität bestimmen, aber auch, weil die Prozessqualität die Produktqualität prägt (Ludewig und Lichter, 2007; Hunter und Thayer, 2001, S. 290). Dies zeigt sich deutlich bei Prüfungen, also analytischen Qualitätssicherungsmaßnahmen, die die Produktqualität durch Fehlerentdeckung verbessern (Ludewig und Lichter, 2007) und die Prozessqualität durch Kontrolle erhöhen (Thayer und Christensen, 2002; Jalote, 2000). Da Prüfungen teuer werden können, besteht die Gefahr, dass nicht ausreichend geprüft wird und somit die Prozessqualität sinkt. Dann können aber Qualitätsmängel hohe Kosten nach sich ziehen. Entscheidungen, die die Qualität betreffen, sind besonders schwierig zu treffen:

- Der Handlungsspielraum für Prüfungen ist im Prinzip groß: Da Prüfungen nicht zwingend notwendig sind, um ein Produkt zu erstellen, reicht der Handlungsspielraum prinzipiell von der Entscheidung, überhaupt nicht zu prüfen, bis zur Entscheidung, sehr viel und sehr intensiv zu prüfen. Dabei können abhängig von der Projektklasse unterschiedliche und unterschiedlich viele Prüfungen durchgeführt werden (Jones, 2007). Vor allem aber kann jede einzelne Prüfung unterschiedlich intensiv durchgeführt werden. Vorgaben in Standardprozessen schränken diesen Handlungsspielraum ein, sie lassen aber Spielraum und werden an Projekte individuell angepasst (Ellims et al., 2006).

- Die Qualitätsbewertung ist schwierig, da eine Bewertung der Qualität mit einfachen, objektiven Metriken interpretiert werden muss und unplausible Ergebnisse liefern kann. Die Resultate einer solchen Qualitätsbewertung sind nur eingeschränkt plausibel, weil diese Metriken nur einen kleinen Teil eines komplexen Projekts erfassen (Kitchenham et al., 2007; Fenton und Neil, 1999). Zur Qualitätskontrolle können auch Prüfungen (Thayer und Christensen, 2002; Jalote, 2000) und subjektive Bewertungen (z.B. Sunazuka et al., 1985) eingesetzt werden. Sie ermöglichen verlässlichere Aussagen zur Qualität als einfache Metriken. Prüfungen und subjektive Bewertungen sind aber teuer; sie kosten Zeit, Aufwand, Mitarbeiter und Geld.
- Die indirekten Wirkungen der Prüfungen sind komplex und schwierig zu durchschauen und können am Projektende nicht objektiv festgestellt werden (Thayer und Christensen, 2002). Beispielsweise können Qualitätsverbesserungen nicht direkt gemessen werden. Sie werden als Einsparungen über die gesamte Lebensdauer sichtbar. Somit muss also ein langer Zeitraum überblickt werden; Einsparungen können nur im Vergleich zu Kosten gemessen werden.
- Im Gegensatz zu den Qualitätsverbesserungen sind die Kosten für Prüfungen offensichtlich: Prüfungen und Korrekturen dauern, benötigen Mitarbeiter und kosten somit Zeit, Aufwand, Mitarbeiter und Geld.
- Für rationale Entscheidungen über Qualitätssicherung müssen Qualität und Kosten gegeneinander abgewogen und verglichen werden. Diese Abwägung und der Vergleich sind schwierig. Kosten und Dauer von Qualitätssicherungsmaßnahmen sind durch standardisierte Metriken am Ende des Projekts vollständig feststellbar und können für einzelne Prüfungen während des Projekts gemessen werden. Die Qualitätsbewertung liefert im Gegensatz dazu subjektive Einschätzungen und ist während des Projekts und am Projektende teuer. Messen lassen sich Qualitätsverbesserungen nicht direkt, aber Einsparungen werden über die gesamte Produktlebensdauer sichtbar. Ein direkter Vergleich zwischen Qualität oder Qualitätsverbesserungen und den dafür investierten Kosten ist also nicht möglich.
- Die Anforderungen an das Produkt und an den Prozess unterscheiden sich zwischen Projekten. Somit spielen Qualität und Kosten eine unterschiedlich wichtige Rolle (Yourdon, 1995). Darum ist nicht möglich, allgemeingültig Prüfungen mit bestimmter Intensität vorzuschreiben. Die optimalen Entscheidungen über Prüfungen hängen also von der konkreten Situation und den Anforderungen des Projekts ab. Prozessstandards enthalten Vorgaben, welche Art von Prüfungen durchgeführt werden sollen, aber keine Vorgaben, wie intensiv die Prüfungen stattfinden sollen (CMMI Product Team, 2002; Hörmann et al., 2006).

Die Situation des Projektleiters und QS-Verantwortlichen ist also schwierig. Sie entscheiden über Prüfungen mit weitreichenden Folgen auf Basis unzureichender Informationen. Sie können Entscheidungen nicht rückgängig machen und nicht objektiv begründen. Untersuchungen bestätigen diese Schwierigkeiten (Ahonen und Junttila,

2003; Mandl-Striegnitz und Lichter, 1998): Die Planung ist unzureichend, Qualitätssicherung und Fortschrittskontrolle werden vernachlässigt.

3.2 Ziele des Kosten-Nutzen-Modells für Prüfungens

In dieser schwierigen Situation sollen darum Projektleiter und QS-Verantwortliche durch ein quantitatives Modell unterstützt werden (Abbildung 7). Das Modell hat also den Zweck, Projektleiter und QS-Verantwortliche bei Entscheidungen über Qualitätssicherung zu unterstützen. Dieser Zweck lässt sich konkreter aus den Schwierigkeiten ableiten:

- Demonstration der Auswirkungen von Entscheidungen: Weil die Wirkungen der Qualitätssicherungsmaßnahmen indirekt und komplex sind, soll das Modell zeigen, wie sich die konkreten und detaillierten Entscheidungen, die Projektleiter und QS-Verantwortliche treffen, auswirken. Das Modell soll die kurzfristigen Auswirkungen auf das Projekt und langfristige Auswirkungen, d.h. Auswirkungen auf Einsatz und Wartung des Produkts, darstellen.
- Diagnose der Auswirkungen von Entscheidungen: Das Modell soll die Auswirkungen widerspiegeln, die sich in realen Projekten aus den getroffenen Entscheidungen ergeben; es soll die Kosten und den Nutzen der Maßnahmen zur Qualitätssicherung zeigen, damit Entscheidungen nachträglich begründet werden können. Das Modell soll also deskriptiv, diagnostisch, für bestehende Projekte eingesetzt werden können.
- Prognose der Auswirkungen zur Planung von Prüfungen: Das Modell soll die Planung der einzelnen Qualitätssicherungsmaßnahmen unterstützen. Das Modell soll beispielsweise fähig sein, Kosten und Nutzen von Prüfungen zu prognostizieren, um den Projektleiter rechtzeitig zu unterstützen.
- Vergleich und Optimierung: Das Modell soll ermöglichen, die Kosten und den Nutzen der Qualitätssicherungsmaßnahmen miteinander zu vergleichen, um Entscheidungen zu begründen. Dadurch soll das Modell ermöglichen, Kosten zu minimieren oder den Nutzen zu maximieren. Das Modell soll also explorativ eingesetzt werden können.

Um diese Ziele zu erreichen, wird der Handlungsspielraum für Qualitätssicherung auf Modelleingaben abgebildet. Das Modell erlaubt, Handlungen auszuwählen (Abbildung 7). Es berechnet abhängig von diesen Eingaben und Merkmalen des Projekts die Kosten und den Nutzen dieser Entscheidung. Kurzfristige und langfristige Auswirkungen, die in der gesamten Lebensdauer des Produkts auftreten, werden im Modell berücksichtigt und vergleichbar dargestellt. Dazu gehören Kosten und Nutzen für einzelne Phasen des Projekts, für das gesamte Projekt und für wichtige Projektklienten.

Das quantitative Modell erhält den Namen CoBe, abgeleitet aus Cost (Kosten) und Benefit (Nutzen). Es handelt sich um ein Kosten-Nutzen-Modell, da nur diese Modell-

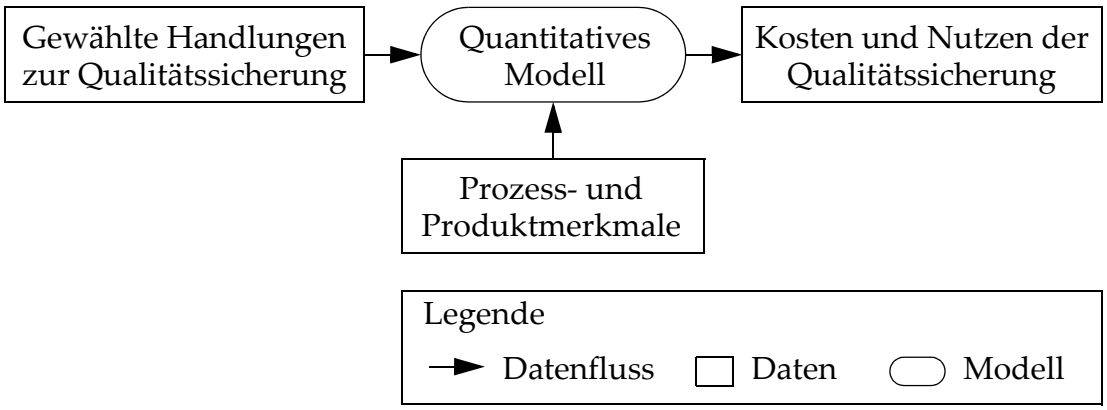

Abb. 7: Idee für das Kosten-Nutzen-Modell CoBe

form erlaubt, Kosten und Nutzen direkt zu vergleichen (Abschnitt 2.5.2). Damit gehört es zu den Entscheidungsmodellen. CoBe liefert aber nicht die optimale Lösung, sondern zeigt die Wirkungen für die (im Modell) getroffenen Entscheidungen. Nur so können auch nicht-optimale Situationen dargestellt werden, etwa um die Folgen von Fehlentscheidungen zu demonstrieren.

Für CoBe werden im Folgenden zuerst die prägenden Merkmale festgelegt, bevor das Modell erstellt wird. Da Entscheidungen unterstützt werden sollen, wird zuerst festgelegt, welche Entscheidungen durch welche Eingaben und durch welche Ausgaben unterstützt werden. Zum Modellzweck gehört auch der vorgesehene Modelleinsatz. Der Modellierungsansatz beschreibt, wie das Modell erstellt und gestaltet wird.

3.3 Unterstützte Entscheidungen

Die Modelleingaben leiten sich aus den Qualitätssicherungsmaßnahmen, für die das Modell Kosten und Nutzen berechnet, und dem Handlungsspielraum, den Projektleiter und QS-Verantwortliche für diese Maßnahmen haben, ab.

3.3.1 Auswahl der Qualitätssicherungsmaßnahmen

Qualitätssicherung kann in drei Bereiche eingeteilt werden (Frühauf et al., 2001): Konstruktive Maßnahmen, analytische Maßnahmen (Prüfungen) und organisatorische Maßnahmen. Das Modell konzentriert sich auf analytische Maßnahmen, weil der Projektleiter über die analytischen Maßnahmen entscheiden kann (PMI, 2000; IEEE-Std. 1490, 2003). Konstruktive Qualitätssicherung, z.B. die Einführung neuer Prozesse, der Einsatz geeigneter Werkzeuge oder Sprachen, die Schulung von Mitarbeitern, werden häufig organisationsweit eingeführt; sie werden dem Prozess-Management zugeordnet (CMMI Product Team, 2002). Organisatorische Maßnahmen sollen mit dem Modell unterstützt werden und sind darum nicht Teil des Modells.

3.3.2 Entscheidungen über Prüfungen und Prüfparameter

Reviews und Tests sind weit verbreitete Prüfungen und werden darum in CoBe dargestellt. Diese Prüfungen können unterschiedlich intensiv durchgeführt werden, je nach Intensität der Prüfung ändern sich die Kosten und der Nutzen. Die Intensität wird durch einzelne Prüfparameter und somit also durch einzelne Entscheidungen bestimmt. Diese Parameter werden für die Planung als klare Vorgaben benötigt (Kerzner, 2006), der Qualitätsplan soll für Tests und Reviews detaillierte Prüfparameter enthalten (IEEE-Std. 12207.1, 1997). Darum werden in CoBe detaillierte Vorgaben, d.h. Prüfparameter, abgebildet.

In Reviews wird Software von Gutachtern geprüft. Das technische Review gehört zu den aufwändigsten und formalsten Review-Varianten. Darum wird es in CoBe dargestellt. Im Review bereiten sich Gutachter zuerst einzeln vor, dann werden dabei entdeckte Befunde in einer Sitzung durchgesprochen (Freedman und Weinberg, 1982; Fagan, 1976). Für Reviews werden die folgenden Vorgaben durch CoBe dargestellt: Die Zahl der Gutachter und die Auswahl der passenden Gutachter (Freedman und Weinberg, 1982), die ausreichende Vorbereitung auf die Sitzung (Fagan, 1986; Frühauf et al., 2006), und wie viel der Software begutachtet werden soll (Frühauf et al., 2006; Schwinn, 2003).

Tests werden üblicherweise in Phasen organisiert, die sich auf unterschiedlichen Integrationsebenen oder Teststufen befinden (Liggesmeyer, 2002). Auf allen Ebenen werden ähnliche Entscheidungen getroffen (Lauterbach und Randall, 1989; Ellims et al., 2006; Liggesmeyer, 2002):

- Im Test werden Testfälle definiert und durchgeführt. Testfälle werden anhand von Testtechniken abgeleitet. Darum werden Entscheidungen über die eingesetzten Testtechniken und ihre Intensität getroffen, d.h. über die Überdeckung des Codes oder die Abdeckung von Anforderungen (Liggesmeyer, 2002; Spillner und Linz, 2003; Frühauf et al., 2006).
- Die Testfälle können definiert werden, bevor der Prüfling zur Verfügung steht, es kann also über den Zeitpunkt der Testvorbereitung entschieden werden (Pressman, 2005; Jalote, 2000; Frühauf et al., 2006).
- Testfälle werden von Testern definiert. Die Auswahl der passenden Tester, d.h. die Kompetenz der Tester, ist eine wichtige Entscheidung (Spillner und Linz, 2003; Liggesmeyer, 2002).
- Über die Testwiederholung nach der Korrektur, die während des Projekts oder in der Wartung erfolgt, wird entschieden (van Megen und Meyerhoff, 1995; Liggesmeyer, 2002; Haley et al., 1995; Sneed et al., 2004; Pigoski, 1997; ISO/IEC 14764, 1999).

Eine weitere Prüfung ist die automatische statische Codeanalyse, im Folgenden kurz als Codeanalyse bezeichnet. Der Programmcode wird von einem Werkzeug auf verdächtige Konstrukte hin untersucht (Spinellis, 2006; Louridas, 2006).

3.4 Kosten und Nutzen von Prüfungen

Prüfungen nützen indirekt. Tabelle 1 fasst Erfahrungen zusammen. Reviews nützen zusätzlich durch frühe Fehlerentdeckung und Schulungseffekte der Reviewteilnehmer, dafür können aber auch weitere Kosten anfallen (Tabelle 2). Durch Tests wird zusätzlich das Vertrauen in das Produkt erhöht.

Nutzen von Prüfungen	Kosten von Prüfungen
• Konkrete Schwächen werden identifiziert (Ludewig und Lichter, 2007; Freedman und Weinberg, 1982; Beizer, 1990), diese können korrigiert werden (Fagan, 1986; Beizer, 1990). • Gute und unbrauchbare Prüflinge werden identifiziert (Ludewig und Lichter, 2007; Freedman und Weinberg, 1982). • Die Projektleitung wird erleichtert, wenn die Qualität durch frühe Prüfungen kontrolliert und durch Korrektur verbessert wird. Dadurch schwankt die Qualität der Artefakte im Projekt und des Produkts weniger stark. Somit wird das gesamte Projekt planbarer (Freedman und Weinberg, 1982). • Das Projekt kann mit konkreten Qualitätskriterien (Ludewig und Lichter, 2007), Meilensteinkriterien und Qualitätsbewertung (Freedman und Weinberg, 1982; Fagan, 1986; Deininger, 1995) besser kontrolliert werden. • Die Erwartung einer Prüfung führt zu besseren Prüflingen (Ludewig und Lichter, 2007). • Prüfdaten ermöglichen Fehlervermeidung und Prozessverbesserungen (Fagan, 1986; Chillarege et al., 1992). • Bessere Produktqualität führt zu höherer Kundenzufriedenheit (Buckley und Chillarege, 1995; Chulani et al., 2003).	Es entstehen Kosten durch • die Prüfungsorganisation, • die Prüfungsvorbereitung, • die Durchführung der Prüfung, • die Auswertung der Prüfung, • die Korrektur der in der Prüfung entdeckten Fehler, • Werkzeuge für die Prüfung, z.B. für Lizenzen, Installation und Wartung, • Schulungen für Prüfungen, • die Einführung, die kurzfristig die Produktivität senkt, • Prozessänderungen, zu denen auch Änderungen der Qualitätssicherung gehören. Sie können zu Chaos und Widerständen führen (DeMarco und Lister, 1999).

Tabelle 1: Kosten und Nutzen von Prüfungen

Nutzen von Reviews	Kosten von Reviews
• Frühe Fehlerentdeckung reduziert die Korrekturkosten (Fagan, 1986; Diaz und King, 2002; Haley, 1996; Freedman und Weinberg, 1982). • Frühe Fehlerentdeckung senkt die Kosten für Testwiederholung (Haley, 1996; Freedman und Weinberg, 1982). • Technische Informationen werden früh sichtbar (Freedman und Weinberg, 1982). Dadurch werden Schätzungen und Pläne besser (Freedman und Weinberg, 1982). • Gutachter lernen aus entdeckten Fehlern (Fagan, 1986). Sie werden geschult und kompetenter (Freedman und Weinberg, 1982). • Technische Informationen werden kommuniziert (Freedman und Weinberg, 1982). • Reviews führen zu einer professionelleren Entwicklungskultur (Fagan, 1986; Freedman und Weinberg, 1982), weniger Personalwechsel und höherer gegenseitiger Wertschätzung (Freedman und Weinberg, 1982).	• Bei der Revieweinführung kann die Produktivität sinken (Weller, 1993). • Ohne Reviewregeln steigen Aufwand und Dauer, das Projektklima leidet (Frühauf et al., 2006; Freedman und Weinberg, 1982). • Das Projekt kann aus organisatorischen Gründen und Zeitmangel der Beteiligten verzögert werden (Freedman und Weinberg, 1982; Porter und Votta, 1997).

Tabelle 2: Kosten und Nutzen von Reviews

3.4.1 Modellierte Kosten und modellierter Nutzen

Im Idealfall stellt ein Modell alle genannten Kosten und den gesamten Nutzen dar. Dieses Ideal lässt sich aus praktischen Gründen kaum erreichen, weil nur messbare Merkmale in ein quantitatives Modell aufgenommen werden können. Die Grenze zwischen messbaren (tangiblen) und nicht-messbaren (intangiblen) Merkmalen ist fließend. Sie ist durch die Kosten zur Datenerhebung bestimmt (Hanusch, 1987). Die möglichen Kosten, um ein Modell zu erstellen und an konkrete Situationen anzupassen, begrenzen also, welche Kosten und welcher Nutzen dargestellt werden können.

Die Auswahl der Kosten und des Nutzens muss aber berücksichtigen, dass die wichtigen, wesentlichen Kosten- und Nutzenmerkmale abgebildet werden, weil das Modell sonst nicht plausibel ist und seinen Zweck nur unzureichend erfüllt.

Wesentlich für alle Prüfungen ist der Nutzen durch die Fehlerentdeckung. Daraus folgt, dass Kosten für die Fehlerkorrektur und Nutzen durch vermiedene Fehlerkosten abgebildet werden müssen. Dieses Konzept der Qualitätskosten wähle ich als

Grundlage für das Modell, weil Fehler und Fehlerkosten leicht zu erheben sind und weil diese Daten in vielen Projekten bereits verfügbar sind.

Die Grenze, ab der Kosten und Nutzen nicht in das Modell abgebildet werden, ergibt sich aus dem Modellzweck. Der Modellzweck ist, Projektleiter und QS-Verantwortliche bei der Planung eines Projekt zu unterstützen. Sie können Entscheidungen treffen, die das Projekt und das Produkt betreffen. Sie können aber keine organisationsweiten Entscheidungen, beispielsweise zur Marktstrategie oder zu organisationsweiten Prozessverbesserungen, treffen. Zu den nicht berücksichtigten Kosten und Nutzen zählen darum die folgenden:

- Schulungseffekte und andere psychologische Auswirkungen können nur indirekt erfasst werden, darum ist eine Bewertung dieser Auswirkungen teuer. Es reicht nicht aus, diese Auswirkungen einmal zu erfassen, um das Modell zu erstellen, weil sich Unternehmenskultur und individuelle Eigenschaften der Mitarbeiter in unterschiedlichen Umgebungen unterscheiden. Soll das Modell in einer neuen Umgebung eingesetzt werden, dann muss zumindest geprüft werden, ob das Modell valide für die neue Umgebung ist. Falls nicht, dann müssen die Auswirkungen erneut modelliert und quantifiziert werden. Modellbildung und Modelleinsatz werden also teuer; CoBe wird weniger verallgemeinerbar. Diese Auswirkungen werden darum nicht im Modell dargestellt.
- Kosten für die Einführung und die damit verbundene Schulung von Prüfungen oder Prüftechniken werden in CoBe nicht dargestellt, weil es sich dabei um strategische und organisationsweite Prozessverbesserungen handelt (CMMI Product Team, 2002). Für eine Kosten-Nutzen-Betrachtung müssten die Auswirkungen über mehrere Projekte der Organisation berücksichtigt werden. Auch der Nutzen, der durch Prüfdaten, ihre Analyse und dadurch mögliche Fehlervermeidung und Prozessverbesserung erreicht wird, wirkt über mehrere Projekte und wird darum nicht berücksichtigt.
- Auswirkungen, die durch eine verbesserte Projektkontrolle erreicht werden, werden nicht dargestellt, weil sie nur indirekt erfasst werden können. Beispielsweise muss erfasst werden, wie sich eine höhere Planungssicherheit auswirkt. Ähnlich wie bei Schulungseffekten wird solch eine Bewertung sehr teuer und ist von der Umgebung und den individuellen Eigenschaften der Projektleiter und QS-Verantwortlichen abhängig. Insbesondere spielt eine Rolle, wie gut die Projektkontrolle bislang durchgeführt wurde, also welches Verbesserungspotential für das Modell überhaupt möglich ist.

Die nicht erfassten Auswirkungen der Prüfungen müssen bei der Interpretation der Modellresultate einbezogen werden. Das Modell kann Entscheidungen nicht treffen (Laux, 1998), weil nur ein Teil der Auswirkungen erfasst wird.

3.4.2 Darstellung der Kosten und des Nutzens

CoBe ist ein Modell, dessen Zweck im Vergleich von Kosten und Nutzen liegt. Für diesen Vergleich müssen die Modellresultate der Kosten und des Nutzens auf der gleichen Skala liegen. Darum ist eine Nutzwert-Analyse oder eine Kosten-Wirksamkeitsanalyse nicht geeignet. Die Kosten-Nutzen-Analyse ist geeignet, weil sie Kosten und Nutzen als Geldwerte ausdrückt. Prinzipiell enthalten Kosten und Nutzen nicht nur materielle, sondern auch immaterielle Auswirkungen, die zur Kosten-Nutzen-Analyse auf Geldwerte abgebildet werden. Daraus folgt, dass in CoBe die Auswirkungen der Prüfungen also mit Geld bewertbar sein müssen. Bei diesem Ansatz unterscheiden sich Kosten und Nutzen im Vorzeichen. Im Modell wird darum der Nutzen als entfallende Kosten dargestellt. Weil CoBe auch zur Planung eingesetzt werden soll und um darzustellen, wann Kosten an- oder entfallen, basiert Cobe auf einzelnen Aktivitäten im Projekt und in der Wartung mit Aufwand, Dauer und Personalbedarf (Abschnitt 2.6.4). Daraus werden Kosten und Nutzen als Geldwerte berechnet. Der Aufbau von CoBe orientiert sich also am Vorgehen zur Kostenschätzung, bei dem Geldwerte aus den Merkmalen der Projektaktivitäten, beispielsweise Aufwand und Dauer, abgeleitet werden (Kerzner, 2006; Metzger und Boddie, 1996).

3.4.3 Abstraktionsebene der Kosten und des Nutzens

Die ideale Abstraktionsebene für Metriken gibt es nicht (Ludewig und Lichter, 2007), da sie vom Zweck der Metrik abhängt. Zur Planung sind für den Projektleiter einzelne Aktivitäten und Arbeitspakete mit Dauer und Personal die kleinsten Planungseinheiten (Abschnitt 2.6.4; PMI, 2000; IEEE 1490, 2003). Seine wichtigen Ziele sind aber das Projekt als Ganzes, also Projekttermin und -kosten (V-Modell XT, 2004). Wartungspersonal, Kunde und Benutzer sind direkt von der Produktqualität betroffen (Alexander und Robertson, 2004). Um Kosten über die Lebensdauer des Produkts zu minimieren oder den Gesamtnutzen über die Lebensdauer zu maximieren, müssen Kosten dieser Klienten dargestellt werden. Darum liefert CoBe Resultate auf diesen drei Abstraktionsebenen. In CoBe werden also Auswirkungen der Entscheidungen auf Aufwand, Dauer und Personal einzelner Aktivitäten, Auswirkungen auf das Projekt. und Auswirkungen über die gesamte Produktlebensdauer abgebildet. Zu den langfristigen Auswirkungen über die Lebensdauer gehören der Aufwand in der Wartung, für den Kunden zu erwartender Schaden oder vermiedene Probleme.

3.5 Prozess- und Produktmerkmale

Kosten und Nutzen der Prüfungen hängen von den individuellen Prozess- und Produktmerkmalen des Projekts ab. Zu diesen Merkmalen gehören diejenigen, die Prüfungen betreffen. Diese werden in CoBe dadurch erfasst, dass die Entscheidungen über Prüfungen und über Merkmale der Prüfungen abgebildet werden. Zu den individuellen Prozess- und Produktmerkmalen gehören aber auch eine Vielzahl von Merkmalen, die das Umfeld, die Anforderungen und die Organisation betreffen. Diese Vielfalt zeigt sich in den Merkmalen, die für die Kostenschätzung mit

COCOMO II (Boehm, 2000) oder für den Projektvergleich in Jones (1996 und 2003) benötigt werden. Diejenigen, die sich auf Prüfungen auswirken, sollen auf das Modell abgebildet werden.

Wichtige Merkmale sind die Organisation der Arbeit, die Stellenbesetzung und das zu Grunde liegende Vorgehensmodell. Modelle, die diese Aspekte und ihre Wirkungen quantitativ beschreiben, werden groß und komplex. Ich verkürze diese Merkmale, um die Auswirkungen der Prüfungen in den Vordergrund zu stellen. Als Grundlage des Modells dient der Software-Lebenslauf mit Spezifikation, Entwurf, Implementierung, Test und Betrieb. Dieses Vorgehen findet sich als Vorgabe in Prozessverbesserungsprogrammen wieder (Chrissis et al., 2003; Hörmann et al., 2006). Ich wähle ein sequentielles Vorgehen für das Modell. Projekte mit nicht-linearem Vorgehen müssen auf das sequentielle Modell abgebildet werden.

3.6 Modelleinsatz

Das Modell muss in den organisatorischen Rahmen des Projekts, in dem Prozessverbesserungen durchgeführt werden, und in die Tätigkeiten des Projektleiters eingebunden werden.

3.6.1 Modellkalibrierung und -anpassung an Projekte und Prozesse

Das Modell bildet eine Menge von Projekten ab; es basiert auf allgemeinen Erfahrungen über Software-Projekte. Für den Modellzweck der Demonstration sind diese allgemeinen Erfahrungen, die sich auf ein fiktives, durchschnittliches Projekt beziehen, ausreichend. Sobald aber Aussagen über reale Projekte und Prozesse möglich sein sollen, d.h. wenn Auswirkungen nachträglich dargestellt, Kosten und Nutzen prognostiziert und optimiert werden sollen, ist es notwendig, die konkrete Situation in das Modell abzubilden:

- Kalibrierung: Erfahrungen mit Kostenschätzmodellen zeigen, dass quantitative Modelle an die Umgebung angepasst, d.h. kalibriert, werden müssen, um ausreichend genaue Resultate zu erhalten (Kemerer, 1987; Boehm, 2000). Diese quantitative Anpassung eines Modells an eine konkrete Umgebung wird als Kalibrierung bezeichnet und wird durch spezielle Kalibrierungsparameter des Modells und eine Kalibrierungsmethode unterstützt und durchgeführt.
- Projekt und Prozess: Projekte unterscheiden sich in den Rahmenbedingungen und im Projektumfang. Diese Unterschiede werden durch Eingabeparameter (Abschnitt 3.5) erfasst.
- Prozess: Prozesse und damit der Prüfprozess ändern sich kontinuierlich im Rahmen von Prozessverbesserungen (Chrissis et al., 2003; Hörmann et al., 2006) und durch den Einsatz des Modells. Verändern sich Prozessmerkmale, die nicht in CoBe modelliert sind, dann soll das Modell entweder kalibriert oder angepasst werden können.

- Laux (1998) argumentiert, dass Entscheidungsmodelle immer revidiert werden müssen, weil sie verkürzen und weil sie subjektive Elemente enthalten (Abschnitt 2.4.2). Diese verkürzten Merkmale stellen sich erst im Lauf der Zeit durch den Modelleinsatz heraus. Es können also keine Eingaben vorgesehen werden. Darum soll das Modell entweder kalibriert oder um diese Merkmale erweitert werden können.

Abbildung 8 skizziert diese Zusammenhänge: Auf der linken Seite werden in CoBe Projekte im Allgemeinen abgebildet. Somit können mit CoBe Auswirkungen der Prüfungen im Allgemeinen demonstriert werden. Auf der rechten Seite soll ein spezielles Projekt in CoBe abgebildet werden, um Auswirkungen von Prüfungen in diesem Projekt zu prognostizieren. Um dieses spezielle Projekt darstellen zu können, muss das Modell kalibriert werden.

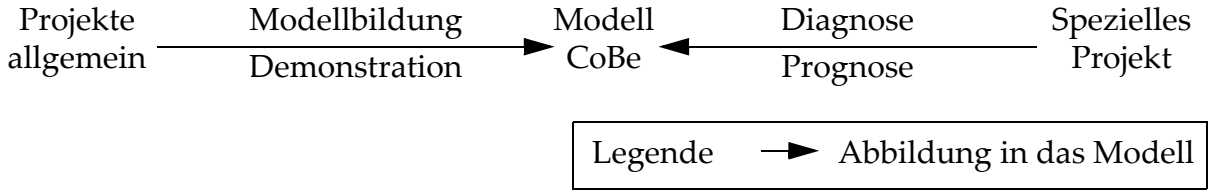

Abb. 8: Abbildung allgemeiner und spezieller Projekte durch das Modell

Die Kalibrierung passt also das Modell einer allgemeinen Realität an eine konkrete Situation an. Die Kalibrierung muss erlauben, das Modell an die Projektumgebung, d.h. an die Organisation und den Einfluss ihrer Kultur oder die Art der Projekte (Anwendungsgebiet, Prozessreife) anzupassen. Dazu ist notwendig, wenige Eingaben zu bieten, die mit Archivdaten belegt werden können. Archivdaten sind Daten abgeschlossener Projekte. Die Änderungen im Rahmen von Prozessverbesserungsmaßnahmen und die Revidierung des Modells, wenn beispielsweise wichtige Aspekte im Modell fehlen und ergänzt werden sollen, zählen zur Modellbildung (Anpassung). Abbildung 9 zeigt die Einbindung des Modells in Projekte und Prozesse für Anpassung und Kalibrierung.

Angelehnt an den Ansatz von Basili (1995) fließen Archivdaten aus abgeschlossenen Projekten als Erfahrungen in das Modell ein (Datenfluss 1 in Abbildung 9). Sie dienen beispielsweise der Kalibrierung. Damit die Modellresultate zur Projektplanung und -kontrolle verwendet werden können, müssen Merkmale des zukünftigen Projekts und Prozesses in das Modell eingegeben werden (1). Die Modellresultate können direkt in der Projektplanung verwendet werden (2), qualitativ, etwa um zu entscheiden, wie eine Prüfung stattfinden soll, oder quantitativ, etwa um die Dauer und die Mitarbeiter einzuplanen. Dadurch wirken die Modellresultate indirekt auf die Projektresultate und Erfahrungen (2). Für den Einsatz des Modells in Projekten mit definiertem, vorgegebenem Prozess, der kontinuierlich verbessert wird, wird der Ansatz erweitert. Der definierte Prozess wird durch Prozessvorgaben in Projekten durchgeführt (3), Erfahrungen aus Projekten fließen als Prozessänderungen ein (5). Das

Modell berücksichtigt die Prozessvorgaben (4), seine Resultate können verwendet werden, um die Prozessvorgaben zu ändern (6).

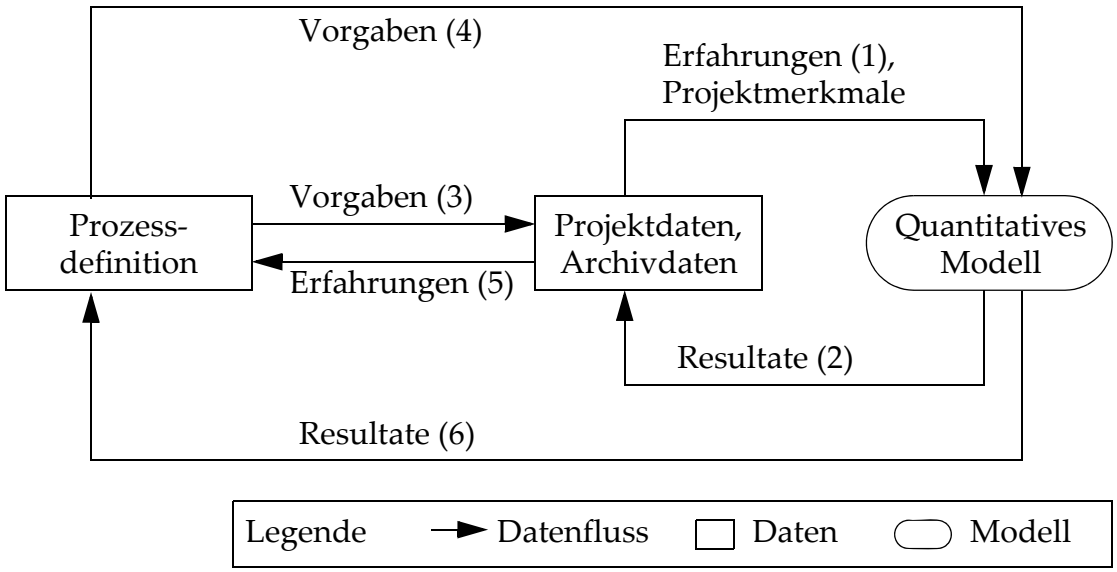

Abb. 9: Modellanpassung und Modellkalibrierung

3.6.2 Planung und Kontrolle

In einem Software-Projekt müssen unterschiedliche Aspekte geplant und kontrolliert werden. Der IEEE-Standard 1490 (2003) und das PMBOK (PMI, 2000) definieren neun solcher Aspekte (Tabelle 3). Das Modell CoBe gehört zum Qualitätsmanagement, insbesondere zur Qualitätsplanung und der verlangten Kosten-Nutzen-Analyse. Unterstützt werden Zeit-, Kosten- und Personalmanagement.

Project Management		
Project Integration Management	Project Scope Management	Project Time Management
Project Cost Management	Project Quality Management	Project Human Resource Management
Project Communications Management	Project Risk Management	Project Procurement Management

Tabelle 3: Gebiete des Projektmanagements nach PMI (2000)

Die Planung findet zu Beginn eines Projekts statt und wird während des Projekts wiederholt; Planung und Kontrolle sind verzahnt, weil bei Abweichungen vom Plan unter Umständen neu geplant werden muss (Kerzner, 2006, S. 396). Der Projektplan wird iterativ erstellt (Kerzner, 2006, S. 486; Ludewig, 1999). Basierend auf dem

Inhaltsverzeichnis eines Projektplans aus Metzger und Boddie (1996) beschreibt Ludewig (1999) die einzelnen Schritte der Planung (Abbildung 10). Termine, Aktivitäten und Meilensteine werden aufeinander abgestimmt, damit das Produkt unter den gegebenen Randbedingungen entwickelt werden kann. Der Modelleinsatz erfolgt iterativ; die Entscheidung kommt im Dialog zwischen Entscheider und Modell zustande (Abschnitt 2.4.2; Laux, 1998).

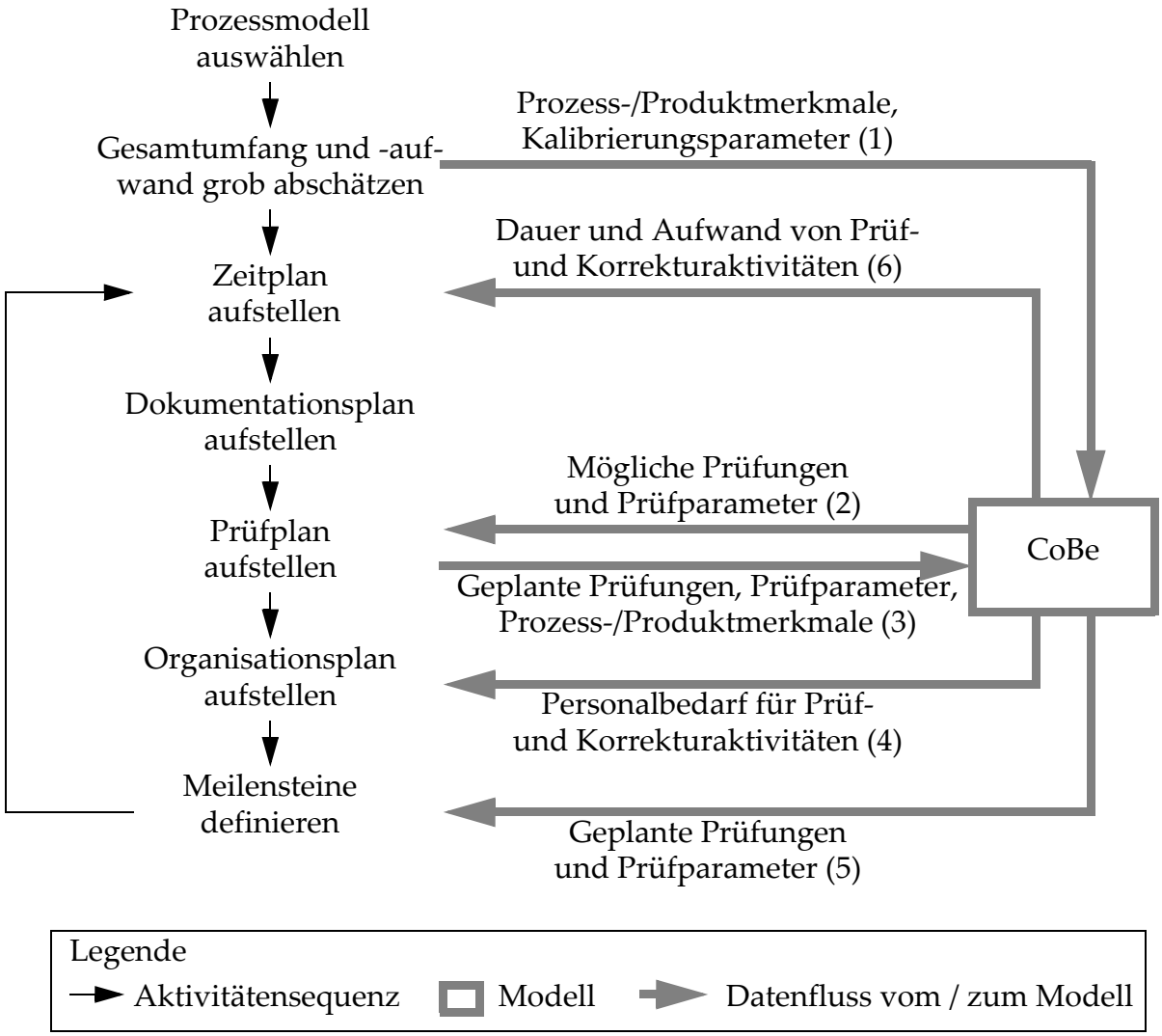

Abb. 10: Modelleinsatz bei der Planung

Die Qualitätsplanung wird von CoBe unterstützt, weil das Modell die Möglichkeiten zur Prüfung und die möglichen Prüfparameter zeigt; es verdeutlicht den prinzipiellen Handlungsspielraum (Datenfluss 2 in Abbildung 10). Prozess- und Produktmerkmale, insbesondere der Umfang, werden eingegeben (1), zusätzlich kann das Modell kalibriert werden. Die geplanten Prüfungen und Prüfparameter werden mit den Pro-

zess- und Produktmerkmalen in CoBe eingegeben (3). Die Organisationsplanung wird durch die Resultate für den Personalbedarf einzelner Prüf- und Korrekturaktivitäten unterstützt (4). Durchgeführte Prüfungen und ihre Prüfparameter können zur Definition inhaltlicher Kriterien der Meilensteine verwendet werden (5). In den Zeitplan fließen die Modellresultate für Dauer, Aufwand und Personalbedarf der einzelnen Aktivitäten ein (6).

3.7 Modellierungsansatz

Bossel (2004) unterscheidet quantitative Modelle nach ihrem Aufbau durch Begriffsgegensätze:

- **Systemerklärend - verhaltensbeschreibend:** Systemerklärende Modelle haben die gleiche Wirkungsstruktur wie das Original, soweit die Struktur erkennbar und bekannt ist. Die Wirkungsstruktur verhaltensbeschreibender Modelle ist vom Original verschieden. Die Begriffe spannen ein kontinuierliches Spektrum auf, Bossel (2004) spricht von Systemen, die durchsichtig, halbdurchsichtig oder undurchsichtig sind.
- **Realparameter - Parameteranpassung:** Modelle mit Realparametern haben Parameter, die direkt gemessen werden können. Bei Modellen mit Parameteranpassung werden die Parameter so gewählt, dass das entsprechende Verhalten erzeugt wird. Auch dieses Begriffspaar lässt Mischformen zu, wenn manche Parameter gemessen werden können, andere nicht messbar sind.

Angelehnt an den Ansatz von Ludewig et al. (1994) wähle ich einen systemerklärenden, realparametrischen Ansatz, soweit die Wirkungsstruktur bekannt ist und die Parameter messbar sind. Bei einem solchen Ansatz stellt das systemerklärende Modell die einzelnen Elemente des Originals und ihre Beziehungen dar. Das Verhalten des Modells entsteht aus dem Verhalten der Elemente und ihren Beziehungen. Empirisch belegte Zusammenhänge werden verwendet, um das Modell zu erstellen. Die Vorteile sind, dass einzelne Bestandteile des Modells ergänzt oder geändert werden können, dass einzelne Bestandteile empirisch untersucht werden können und dass das Modell mit empirisch belegten Bestandteilen plausible Resultate ergibt. Diese Vorteile werden durch ein komplexes Modell erkauft, das aus vielen, miteinander verbundenen Teilen besteht. Die Verwendung von Realparametern ist vorteilhaft, weil erst dadurch die einzelnen Bestandteile kalibriert und empirisch validiert werden können. Für diese Realparameter ziehe ich verbreitete Metriken vor, weil dann Daten leichter verfügbar sind und von den Beteiligten leichter interpretiert werden können. Da empirische Zusammenhänge verwendet werden und da die Realparameter mit empirischen, statistischen Werten quantifiziert werden, sind auch die Resultate statistische Werte. Somit handelt es sich um ein induktives Modell.

Nach Fishwick (1995) können für die systemerklärende Modellierung verschiedene Blickwinkel eingenommen werden. Anhand dieser Blickwinkel lassen sich Ansätze zur Modellierung unterscheiden, die auf unterschiedlichen Abstraktionsebenen auch gemeinsam eingesetzt werden können:

- Declarative Modeling. Diskrete Zustände und die Übergänge zwischen diesen Zuständen werden modelliert. Der Ansatz ist geeignet für die Beschreibung von Ereignissen, Phasen und gekoppelten Zuständen.
- Functional Modeling. Funktionen, die durch Ein- und Ausgaben gekoppelt sind, werden modelliert. Der Ansatz ist geeignet für Objekte, die gerichtet verbunden sind, und um Flüsse durch ein System zu beschreiben.
- Constraint Modeling. Gleichgewichtsbedingungen und andere Einschränkungen bestimmen das Modell. Der Ansatz ist geeignet für Systeme, die durch Konstanten beschrieben werden.
- Spatial Modeling. Räumliche Beziehungen werden modelliert. Der Ansatz erlaubt detaillierte Modelle kleiner Teilchen.

Für Prüfungen und ihre Auswirkungen wähle ich einen funktionsorientierten Ansatz, weil die detaillierten Entscheidungen über Prüfungen, die das Modell unterstützt, kontinuierlich wirken. Auch die einzelnen Aktivitäten, die das Modell beschreibt, laufen kontinuierlich ab. Es ist nicht notwendig, bei der Prognose in den Verlauf des Projekts einzugreifen, da die Modellresultate der getroffenen Entscheidungen interessieren; die Modellresultate für andere Entscheidungen sollen direkt verglichen werden können.

Diskrete Zustandsübergänge spielen in der Software-Entwicklung für Projektphasen eine wichtige Rolle. Für den Modellzweck stehen Entscheidungen über diesen Übergang aber nicht im Mittelpunkt, sondern die detaillierten Entscheidungen über die Prüfungen. Somit ist also nicht notwendig, diese Entscheidungen explizit abzubilden und einzelne Zustände zu unterscheiden.

Modelle mit Gleichgewichtsbedingungen sind für die Problemstellung weniger geeignet, weil das Modell auch erlauben soll, nicht-optimale Entscheidungen darzustellen, beispielsweise zur Demonstration. Werden Gleichgewichtsbedingungen modelliert, so führen diese zu optimalen Lösungen, so dass andere Entscheidungen nicht mehr dargestellt werden können. Ein räumliches Modell ist offensichtlich kaum geeignet, weil Raum im Modell keine Rolle spielt.

Zufallseffekte werden mit dem Modell nicht explizit modelliert, weil die Resultate durch Zufallseffekte zusätzlich unsicherer werden. Dadurch wird auch der Vergleich zwischen Modell und Realität erschwert, weil dann die Zufallsereignisse zwischen Modell und Realität übereinstimmen müssen. Darum handelt es sich um ein deterministisches Modell (Bossel, 2004). Das Modell basiert auf empirischen Aussagen, d.h. aus empirischen Zusammenhängen, die durch statistische Daten quantifiziert werden. Darum handelt es sich um ein induktives Modell. Es erlaubt somit statistische

Aussagen. Das Modell erlaubt keine sicheren Aussagen, weil diese nur für winzige Ausschnitte aus den komplexen Projekten der Software-Entwicklung möglich sind.

Die Modellbildung erfolgt mehrstufig (Drappa, 1998; Bossel, 2004; Fishwick, 1995; Sargent, 2005). Abbildung 11 skizziert die einzelnen Schritte: Die Realität wird zuerst als konzeptionelles Modell natürlichsprachlich beschrieben. Dieses konzeptionelle Modell wird in ein funktionales, mathematisches Modell überführt, das aus Gleichungen und Parametern besteht. Dieses Modell wird durch Metriken und konkrete Werte quantifiziert; dies wird als Quantifizierung bezeichnet. Für die Realisierung wird kein Simulationssystem benötigt. Das Modell kann durch unterschiedliche Programme realisiert werden.

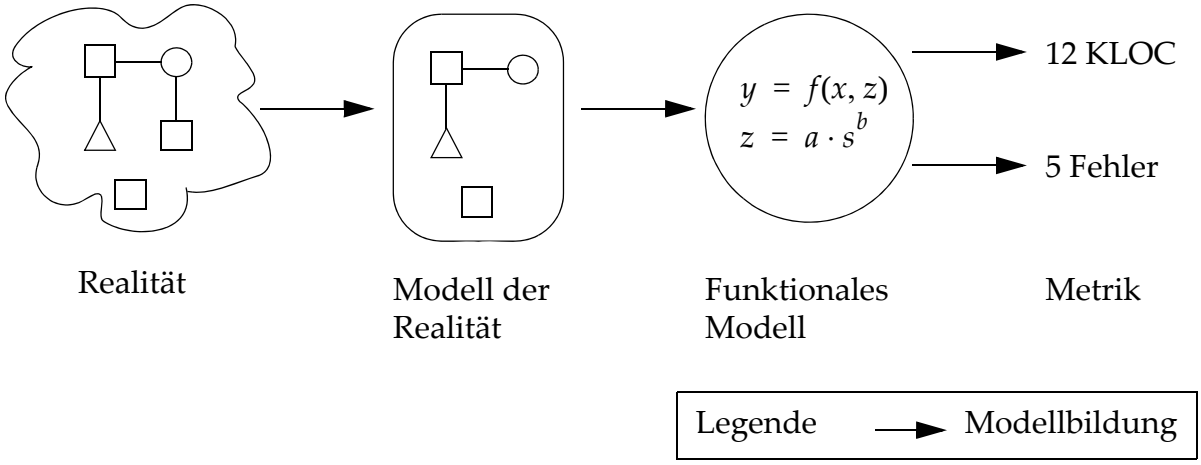

Abb. 11: Modellbildung eines funktionalen Modells (angelehnt an Drappa, 1998)

3.8 Zusammenfassung

3.8.1 Prüfungen und Prüfparameter in CoBe

Das Modell bildet Kosten und Nutzen von Prüfungen ab. Prüfungen werden durch Prüfparameter beschrieben. Reviews werden als Spezifikations-, Entwurfs- und Codereviews modelliert mit Eingaben für die Gutachterzahl, die Kompetenz der Gutachter, die Vorbereitungsintensität, den Umfang des Prüflings mit oder ohne wiederverwendeter Software. Modultest, Subsystem- und Systemintegrationstest, Systemtest werden durch Testtechniken und Parameter für diese Techniken, den Zeitpunkt der Testvorbereitung, die Kompetenz der Tester und die Strategien der Testwiederholung modelliert. Die automatische statische Codeanalyse ergänzt diese Prüfungen.

3.8.2 Kosten und Nutzen in CoBe

Das Modell stellt Kosten und Nutzen von Prüfungen dar. Nutzen ist definiert durch entfallende Kosten, Kosten und Nutzen unterscheiden sich nur im Vorzeichen. Kosten und Nutzen werden auf Geldwerte abgebildet. Die Modellresultate für Kosten und Nutzen beruhen auf Prüf- und Fehlerkosten (Abschnitt 2.9). Für anfallende und entfallende Kosten (Nutzen) werden also Behebungskosten (Korrektur und Prüfung der Korrektur), Fehlerfolgekosten (Kosten beim produktiven Einsatz), Prüfkosten (Vorbereitung, Durchführung, Auswertung) jeweils einschließlich organisatorischer Kosten betrachtet. Im Modell werden Kosten und Nutzen für einzelne Aktivitäten beschrieben. Kosten und Nutzen sind dargestellt durch Planungsmetriken (Aufwand, Dauer, Personalbedarf, Abschnitt 2.6.4) und Geldwerte. Die Resultate werden zusammengerechnet, um Gesamtkosten und Gesamtnutzen zu berechnen.

Kapitel 4

Verwandte Arbeiten

Es gibt bereits eine Reihe quantitativer Modelle für Software-Projekte und für Qualitätssicherung in Software-Projekten. Im Folgenden werden zwei dieser Modelle näher betrachtet, weil sie als Grundlagen für CoBe verwendet werden: SESAM (Software Engineering Simulation durch Animierte Modelle) enthält ein quantitatives Modell mit dem Schwerpunkt Qualitätssicherung, das QS-Modell (Qualitätssicherungs-Modell); COCOMO II ist ein Kostenschätzverfahren (Abschnitt 4.2). Diese Modelle diskutiere ich vor allem unter dem Aspekt der Wiederverwendung von Modellteilen für CoBe. Das Archiv von Jones enthält umfangreich Metriken (Abschnitt 4.3); es bietet sich damit ebenfalls als Fundgrube für quantitative Zusammenhänge an. Daran schließt sich die Diskussion weiterer verwandter Arbeiten an (Abschnitt 4.4). Auch bei dieser Diskussion spielt eine wichtige Rolle, ob Teile dieser Arbeiten für CoBe genutzt werden können.

4.1 Projektsimulation mit SESAM und dem QS-Modell

SESAM (Ludewig et al., 1994) ist ein Simulationssystem, das eingesetzt wird, um Projektleiter zu schulen. Der (angehende) Projektleiter wird in SESAM Spieler genannt (Abbildung 12). Er leitet ein fiktives Projekt, übernimmt also die Rolle des Projektleiters. Dazu wird das fiktive Projekt mit SESAM simuliert. Der Spieler kann während des Projektverlaufs über eine Benutzungsschnittstelle in des Projekt eingreifen. Der Tutor führt die Schulung durch und analysiert die simulierten Projekte. Dabei wird er durch Analysewerkzeuge unterstützt.

In SESAM wird zwischen dem generischen Simulator und dem Modell eines Projekts unterschieden. Der Simulator wird als Basismaschine bezeichnet. Er speichert Informationen über den Projektzustand und führt den dynamischen Teil des Modells aus. Die Simulation erfolgt in einzelnen Zeitschritten. Jeder Zeitschritt entspricht einer festen Dauer im Projekt, z.B. einem Tag. Der Spieler kann nach jedem Zeitschritt eingreifen und den nächsten Schritt anstossen. Die Projektzeit wird auf die Simulationszeit abgebildet, so dass ein Projekt am Simulator in wenigen Stunden durchgeführt werden kann.

Das Modell wird vom Modellbauer erstellt und gewartet. SESAM-Modelle bestehen aus drei Komponenten:

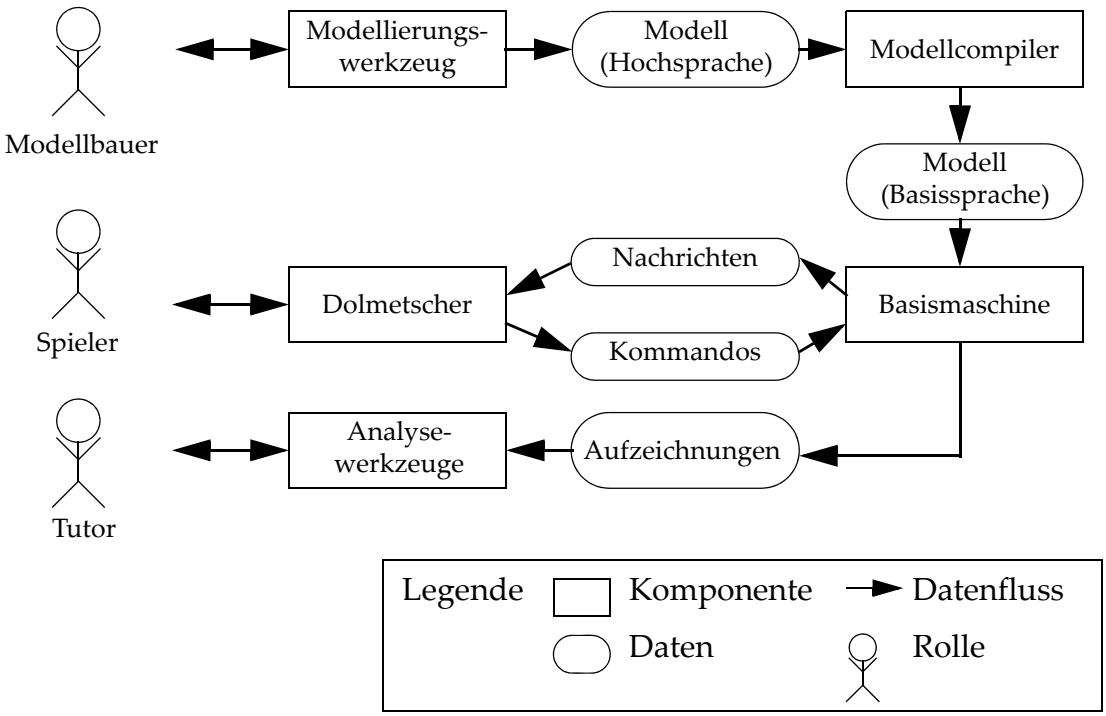

Abb. 12: SESAM-Komponenten und SESAM-Rollen (Reißing, 1996)

- Der Modellbauer definiert im Schemamodell die möglichen Entitäten des Projekts als Entitätstypen. Wichtige Entitätstypen sind Dokumente oder Mitarbeiter. Die möglichen Beziehungen werden durch Relationstypen definiert. Wichtige Relationstypen beschreiben beispielsweise, dass ein Mitarbeiter die Spezifikation erstellt oder korrigiert oder an einem Review teilnimmt. Die Typen werden durch Attribute beschrieben.
- Die Startsituation ist eine konkrete Ausprägung des Schemamodells für den Projektbeginn. Sie beschreibt beispielsweise, welche Mitarbeiter zur Verfügung stehen.
- Dieser Startzustand wird von der Basismaschine anhand des Regelmodells verändert. Das Regelmodell enthält Kommandos, Nachrichten und Regeln. Kommandos und Nachrichten sind zur Kommunikation zwischen Projektleiter und Simulator definiert. Der Spieler kann in jedem Zeitschritt mit Kommandos eingreifen und bekommt durch Nachrichten Informationen mitgeteilt. Die Regeln definieren, wie sich das Projekt verhält. Eine Regel wird durch zwei Teile beschrieben, den Bedingungsteil und den Aktionsteil. Zuerst werden Bedingungen für den Zustand festgelegt. Sind diese Bedingungen im aktuellen Zustand erfüllt, dann werden die Zustandsänderungen durchgeführt, die im zweiten Teil der Regel, dem Aktionsteil, formuliert sind. Zum Beispiel könnte eine Bedingung sein, dass ein Mitarbeiter an der Spezifikation arbeitet. Ist diese Bedingung im Zustand des aktuellen Zeitschritts erfüllt, dann wird die Zustandsänderung durchgeführt. Beispielsweise werden in diesem Zeitschritt Anforderungen zur Spezifikation hinzugefügt.

4.1.1 Das Qualitätssicherungs-Modell

Das Qualitätssicherungs-Modell (QS-Modell) wird in Schulungen eingesetzt (Drappa, 1998). Mit diesem Modell sollen Auswirkungen des Projektmanagements auf die Software-Qualität gezeigt werden. Das Modell umfasst Projektaktivitäten von der Analyse bis zur Übergabe an den Kunden. Modelliert werden kleine und mittelgroße Projekte, die ein simulierter Kunde als Auftrag für ein Informationssystem vergibt. Der Kunde fordert ein Produkt mit bestimmtem Umfang und mit bestimmter Qualität, definiert durch die Fehlerzahl. Er stellt ein Budget zur Verfügung und nennt einen Termin für die Auslieferung.

Aufgaben des Projektleiters

Das QS-Modell stellt den Spieler in der Rolle des Projektleiter vor die Aufgaben der Planung, Stellenbesetzung und Projektführung:

- Die Spieler müssen selbständig planen. Die dazu notwendigen Informationen sind im Modell enthalten und werden zu Beginn der Simulation durch Nachrichten ausgegeben.
- Das Modell bietet dem Spieler Kommandos an, mit denen der Spieler simulierte Mitarbeiter mit unterschiedlichen Erfahrungen und Fähigkeiten zu beliebigen Zeitpunkten in das Projekt aufnehmen und aus dem Projekt entlassen kann.
- Der Spieler kann durch Kommandos den simulierten Mitarbeitern Aufgaben zuteilen, zu beliebigen Zeitpunkten. Dazu gehört, dass Dokumente einschließlich Code erstellt werden und dass der Code integriert wird. Der Spieler kann verschiedene Prüfungen anordnen, nämlich Reviews der Dokumente mit zwei oder drei Gutachtern, Modultest, Integrationstest und Systemtest. Nach jeder Prüfung kann die Korrektur zugeteilt werden. Diese Aktivitäten können frei zu beliebigen Zeitpunkten an Mitarbeiter vergeben werden, so dass unterschiedliche Projektverläufe möglich sind. Die dabei entstehende Software ist nicht real, sondern wird durch Attribute beschrieben. Dazu gehört der Umfang einzelner Anforderungen und die Zahl der enthaltenen Fehler.
- Für die Projektverfolgung können Informationen über das Projekt abgefragt werden. Es sind aber nur diejenigen Informationen zugänglich, die auch in der Realität verfügbar sind. So kann der Projektleiter zwar erfragen, wie viele Fehler im Systemtest entdeckt wurden, aber nicht, wie viele Fehler in der Software enthalten sind.

Organisation spielt eine untergeordnete Rolle. Personalführung lässt sich mit einem Simulationssystem kaum trainieren (Drappa, 1998).

Zusammenhänge des Modells

Das Modell beruht auf empirisch abgesicherten Zusammenhängen, deren Zusammenspiel untereinander und mit den Eingriffen des Projektleiters den Verlauf und das Ergebnis des Projekts bestimmen. Die wesentlichen Zusammenhänge sind:

- Die Software-Entwicklung erfolgt als schrittweise Transformation von Vorgaben. Fehler werden dabei aus der Vorgabe übernommen, zusätzlich werden Fehler gemacht. Fehlen Informationen in der Vorgabe, dann können sie nicht übernommen werden. Informationen aus der Vorgabe können bei der Entwicklung vergessen werden.
- Entwickler haben bestimmte Qualifikationen und Erfahrungen für einzelne Aktivitäten. Sie unterscheiden sich in diesen Eigenschaften. Beispielsweise kann ein Entwickler ein erfahrener Tester sein, ein anderer Entwickler hat dafür mehr Erfahrung im Entwurf. Abhängig von diesen Eigenschaften entstehen bessere oder schlechtere Resultate. Auch das Gehalt hängt von den Qualifikationen und Erfahrungen der Entwickler ab. Die Personalkosten des Projekts hängen also auch davon ab, welche Entwickler mit welchen Qualifikationen und Erfahrungen eingestellt werden.
- Die Kosten des Projekts sind durch die Personalkosten bestimmt.
- Aufwand und Dauer sind eng gekoppelt. So gilt, dass das Resultat desto früher verfügbar ist, je mehr Entwickler gemeinsam daran arbeiten. Je mehr Entwickler zusammenarbeiten, desto höher wird der Aufwand, da der Kommunikationsaufwand steigt. Für jedes Projekt kann eine bestimmte Zahl Entwickler sinnvoll eingesetzt werden. Eine bestimmte Dauer wird mindestens benötigt, um das Projekt abzuschließen.
- Je später ein Fehler entdeckt wird, desto aufwändiger wird seine Korrektur. Der Autor ist für die Korrektur besser als andere Entwickler geeignet.
- Reviews erlauben, Fehler früh zu finden. Die Fehlerentdeckung hängt von den Gutachtern, vom Prüfling und von der Vorgabe ab. Gutachter sollten geeignet und vorbereitet sein. Die Dokumente sollten eine Mindestqualität haben. Der Kunde kann in der Spezifikation und im Handbuch weitere Fehler entdecken.
- Dynamische Modellkonzepte beschreiben die Aufteilung in Arbeitspakete, so dass mehrere Entwickler die gleiche Aktivität durchführen können. Die Konzepte erlauben eine freie Wahl der Reihenfolge und überlappende Aktivitäten.

Metriken des Modells

Die wesentlichen Entitätstypen des Modells sind Mitarbeiter, Dokumente, Code und Prüfberichte. Diese werden durch Attribute quantitativ beschrieben.

- Qualifikation und Erfahrung eines Entwicklers werden jeweils für die unterschiedlichen Aktivitäten auf einer vierstufigen Ordinalskala dargestellt.
- Jedes Dokument wird durch seinen Umfang in Adjusted Function Points (IFPUG, 2004) beschrieben. Der Code-Umfang wird abhängig von der Programmiersprache auf Lines of Code umgerechnet, der Umfang anderer Dokumente wird durch die Zahl ihrer Seiten dargestellt.

- Die Qualität jedes Dokuments wird durch Korrektheit und Vollständigkeit dargestellt. Korrektheit wird durch die Fehlerzahl (IEEE 1044, 1993) beschrieben, unterschieden nach Analysefehlern, Grobentwurfsfehlern, Feinentwurfsfehlern, Codefehlern und Handbuchfehlern. Vollständigkeit wird durch fehlenden Umfang im Vergleich zum geforderten Umfang in Function Points gemessen.

Das Ergebnis der Entwicklungsaktivitäten wird durch die Produktivität (in Function Points pro Stunde), die Fehlerrate (in eingefügte Fehler pro Function Point) und die Verlustquote (Anteil nicht umgesetzter Function Points) bestimmt. Das Ergebnis wird beeinflusst durch die Merkmale des Entwicklers, d.h. seine Erfahrungen und Qualifikationen für die Entwicklungsaktivität und für das Resultat. Prüfungen sind durch die Produktivität (in Funktion Points pro Stunde) und durch Fehlerentdeckungsquoten quantifiziert. Die Fehlerentdeckungsquote beschreibt den Anteil der entdeckten Fehler, bezogen auf die zu entdeckenden Fehler. Analog wird die Verlustentdeckung dargestellt.

4.1.2 Einsatz und Anwendung

Schulungen mit SESAM folgen einem festen Ablauf, um einen Lernerfolg zu ermöglichen (Mandl-Striegnitz, 2001). Der Ablauf beginnt mit einer Einführungsveranstaltung. Danach spielen die Teilnehmer ihr erstes Spiel. Der Tutor analysiert dieses Spiel, um dann in einer Feedback-Runde die Stärken und Schwächen aufzuzeigen. Nach diesem Feedback haben die Spieler in einem zweiten Spiel die Möglichkeit, ihre Schwächen zu verbessern.

Die Analyse durch den Tutor erfolgt von den Projektresultaten ausgehend. Für den Tutor sind dabei alle Modelldaten über den gesamten Verlauf des Projekts zugänglich, also auch diejenigen, die in der Realität nicht bekannt sind. Beispielsweise können enthaltene Fehler analysiert werden. Damit der Tutor konkrete Schwächen aufzeigen kann, muss er einzelne, sich überlagernde Effekte identifizieren (Hampp und Opferkuch, 2007). SESAM ermöglicht also, simulierte Projekte nachträglich zu analysieren. SESAM gibt aber nicht vor, wie ein ideales Projekt auszusehen hat, sondern schränkt den Spieler so wenig wie möglich ein. Dies erlaubt dem Spieler, ganz unterschiedliche Lösungen auszuprobieren, ohne dass der Modellbauer diese Lösungen explizit bei der Modellierung berücksichtigen muss. Durch die Modellkomplexität ist nicht möglich, eine optimale Lösung im Voraus zu bestimmen.

4.1.3 Andere Modelle in SESAM

Für SESAM gibt es weitere Modelle und Modellvarianten. Mit dem strikt atomaren Modell SAM wird die Erstellung einzelner Dokumente durch Entwickler so detailliert wie möglich modelliert (Eisenbarth und Rohrbach, 1998). Das QSVA-Modell erweitert das QS-Modell um Verhaltensaspekte der Entwickler, beispielsweise Motivation oder Krankheit (Kalajzic, 2001). Eine feingranulare Variante des QS-Modells beschreibt Spezifikations- und Entwurfsreviews detailliert (Hampp, 2001). In dieser Modellvariante entdecken einzelne Gutachter Fehler durch Vorbereitung auf die Reviewsitzung,

abhängig vom Umfang des Prüflings, zu prüfenden Aspekten und ihrer Erfahrung. Auch Zusammenhänge in der Reviewsitzung sind modelliert: So wirkt sich aus, wie erfahren der Moderator und die Gutachter sind, oder ob ein Aktuar (oder Protokollführer) anwesend ist.

4.1.4 Bewertung und Folgerungen

Das SESAM-System soll zur Projektleiter-Schulung eingesetzt werden. Dieses Ziel prägt die Basismaschine, die Werkzeuge und die Modelle. CoBe soll zur Planung eingesetzt werden. Daraus ergeben sich Gemeinsamkeiten und Unterschiede für die Gestaltung und Realisierung des Modells:

SESAM bildet die Projektzeit auf die Simulationszeit ab und erlaubt, in jedem Zeitschritt in das Projekt einzugreifen. Da CoBe zur Planung eingesetzt wird, sollen im Gegensatz dazu die Resultate sofort und direkt dargestellt werden.

Mit SESAM soll der Spieler nur die Informationen erfahren können, die auch in der Realität verfügbar sind. CoBe soll im Gegensatz dazu Informationen liefern, die nicht oder noch nicht verfügbar sind.

Das QS-Modell bietet eine Reihe von Modellelementen, die für ein Kosten-Nutzen-Modell für Prüfungen als Grundlage dienen können: Die Zusammenhänge im QS-Modell sind empirisch belegt und können in ein Kosten-Nutzen-Modell übernommen werden. Dazu gehört insbesondere der Zusammenhang der Fehlerentstehung und Fehlerentdeckung, der auch als Fehlerstrommodell bezeichnet wird. Das Modell enthält mögliche Metrikdefinitionen für den Umfang, für Fehler, für Erfahrungen und für Fähigkeiten der Entwickler.

Der Schwerpunkt des QS-Modells liegt auf den Tätigkeiten des Projektleiters; mit CoBe sollen dagegen ganz bestimmte Entscheidungen unterstützt werden. Einige Aspekte des QS-Modells werden darum in CoBe nicht benötigt, andere Aspekte fehlen im QS-Modell: Mit dem QS-Modell können Entscheidungen über Prüfungen und ihre Prüfparameter nicht direkt unterstützt werden, da das Modell mit der Auslieferung des Produkts endet. Wirkungen der Prüfungen während der Wartung und beim Einsatz werden darum nicht dargestellt. Außerdem wird nur ein Teil der Entscheidungen über Prüfungen dargestellt. Dies gilt insbesondere für den Test: Es können unterschiedliche Teststufen, aber nicht die eingesetzten Methoden oder ihre Vollständigkeit gewählt werden. Diese Parameter sind im QS-Modell fest vorgegeben. Auch ein Vergleich zwischen Kosten und Nutzen der Prüfungen ist mit dem QS-Modell nicht direkt möglich. Dazu werden zwei Spielverläufe benötigt, die sich in den Entscheidungen des Spielers über Prüfungen unterscheiden (Hampp und Opferkuch, 2007). Dauer, Aufwand, Kosten, Code- und Handbuchqualität (Korrektheit, Vollständigkeit) bilden eine vektorielle Größe, so dass eine lineare Bewertung mehrerer SESAM-Projekte nicht möglich ist.

Das QS-Modell enthält als wichtigen Aspekt der Simulation den Zeitbezug zwischen Aktivitäten: Ein Prüfling kann etwa nur dann vollständig geprüft werden, wenn er komplett ist. Diese Abhängigkeiten sollen die Spieler kennenlernen und beachten. Für ein Modell, mit dem Entscheidungen über Prüfparameter unterstützt werden, ist dieser Aspekt aber nicht wichtig, da mit diesem Aspekt die Organisation der Arbeit, aber nicht die Prüfparameter im Vordergrund stehen. Die Beziehungen werden darum nicht übernommen.

4.2 Kostenschätzung mit COCOMO II

COCOMO II (Boehm, 2000) ist ein algorithmisches Kostenschätzverfahren. Mit COCOMO II werden Aufwand, Dauer und Personalbedarf für ein Software-Projekt geschätzt. Das Modell wurde auf empirischem Wege erstellt, es basiert auf einem Regressionsverfahren. COCOMO II beschreibt das Verhalten, aber nicht die Wirkungsstruktur in Software-Projekten.

4.2.1 Zusammenhänge in COCOMO II

In COCOMO II wird der Aufwand aus dem Umfang und aus weiteren Einflussparametern berechnet: $PM = A \cdot S^E \cdot EM$ mit PM als Aufwand und S als Umfang. Einflussparameter sind A, E und EM. A ist ein Kalibrierungsparameter, der angepasst werden kann.

- Der Aufwand wächst überproportional mit dem Umfang S, der als Zahl der Anweisungen gemessen wird (Abbildung 13). Der Exponent E ist in der Regel größer als 1. Er wird durch im Wesentlichen durch Prozessmerkmale, z.B. den Reifegrad oder die Flexibilität, bestimmt.

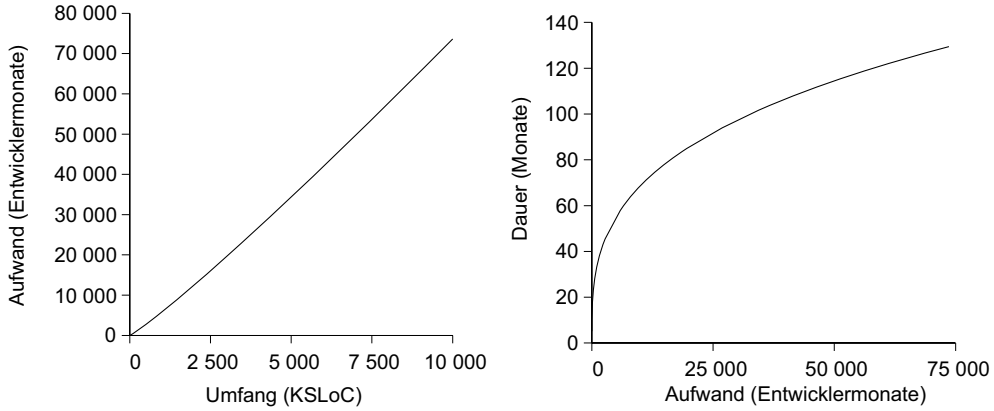

Abb. 13: Umfang, Aufwand und Dauer in COCOMO II

- Der Aufwand wird durch weitere 17 Merkmale bestimmt, die den Einflussfaktor EM ergeben. Die Merkmale sind in Gruppen für Projektmerkmale, Prozessmerkmale, Merkmale der Entwicklungsplattform und Personalmerkmale gegliedert. Ein

einzelnes Merkmal kann den Aufwand um einen Faktor größer 2 beeinflussen, die Personalmerkmale können den Aufwand um den Faktor 3,5 verändern.

Die Dauer wird aus dem Aufwand berechnet:

- Dauer, Aufwand und Personal sind nicht frei wählbar. Das Verhältnis zwischen Dauer und Aufwand ist durch die folgende Gleichung bestimmt: $TDEV = C \cdot PM^D$ mit $TDEV$ als Entwicklungsdauer, PM als Aufwand, C und D als Einflussparameter. Abbildung 13 zeigt diesen Zusammenhang im rechten Diagramm; der Exponent D liegt typisch um den Wert 0,3, ist also deutlich kleiner 1. Der Zusammenhang lässt sich in gewissen Grenzen verändern, etwa um großen Termindruck darzustellen. Dazu kann die Dauer aber höchstens um 25 % verkürzt werden, die Verkürzung führt zu höherem Aufwand.
- Für ein Standardvorgehen, angelehnt an das Wasserfallmodell, gibt es eine typische Verteilung des Aufwands und der Dauer auf die Phasen und innerhalb der Phasen auf die Aktivitäten.

4.2.2 Kalibrierung und Validierung

Das Modell wurde mit 161 Datenpunkten aus der Industrie kalibriert und validiert. Für den Einsatz wird aber eine lokale Kalibrierung empfohlen, weil die Bewertungen der Faktoren subjektiv sind, Prozessunterschiede zum Tragen kommen und Begriffsdefinitionen unterschiedlich sein können.

4.2.3 Bewertung und Folgerungen

COCOMO II bietet eine Top-down-Schätzung für Aufwand, Dauer und Personalbedarf eines Software-Projekts, seiner Phasen und Aktivitäten. Entscheidungen über Prüfungen sind aber nicht Teil des Modells.

Für ein Kosten-Nutzen-Modell, mit dem Entscheidungen über Prüfungen und die Projektplanung unterstützt werden sollen, ist der Zusammenhang zwischen Aufwand, Dauer und Personalbedarf relevant: Der Projektleiter benötigt diese Größen für die Planung der Prüfungen. Mit den COCOMO-II-Zusammenhängen können Personalbedarf und Dauer aus dem Aufwand abgeleitet werden. Dies ist nur möglich, wenn, anders als im QS-Modell, Auswirkungen der Organisation, Verteilung und Reihenfolge der Arbeit nicht gezeigt werden sollen. Dazu muss der Prozess festgelegt werden, in COCOMO II etwa ein sequentieller Prozess.

Die Erfahrungen bei der Kalibrierung und Validierung von COCOMO II zeigen, dass die Formeln für Aufwand und Dauer an die Organisation, Domäne oder Projektart quantitativ angepasst werden müssen. Sie müssen für die lokale Einsatzumgebung kalibriert werden. Die Kalibrierung, die Einflussfaktoren und der überproportionale Einfluss des Umfangs wirken sowohl auf den Gesamtaufwand als auch auf den Aufwand einzelner Aktivitäten im Projekt.

Mit der Entscheidung, die Zusammenhänge aus COCOMO II zu verwenden, sind einige Metriken (Umfang und Projektmerkmale) festgelegt: Die Grundlage bildet der Software-Umfang, gemessen in SLOC (logische Lines of Code, Anweisungen), definiert nach Park (1992). Alternativ können Unadjusted Function Points (IFPUG, 2004) auf Lines of Code abgebildet werden. Die 22 Merkmale werden durch eine siebenstufige Ordinalskala abgebildet. Die einzelnen Kategorien sind teilweise durch quantitative Grenzwerte definiert, teilweise durch Beschreibungen.

4.3 Das Datenarchiv und die Analysen von Jones

Jones (1996, 2003 und 2007) zeigt Metriken und Metrikwerte aus einem umfangreichen Datenarchiv. Im Archiv sind Daten aus mehreren Tausend Projekten. Diese Daten sind durch ein Metrikprogramm beschrieben, das definiert, welche Metriken auf welcher Abstraktionsebene erhoben werden (Jones, 1996).

Die Grundlage bildet der funktionale Umfang von Software in Function Points. Aufwand, Dauer und Personalbedarf sind wesentliche Metriken zur Produktivitätsbewertung, Fehlerzahlen werden zur Qualitätsbewertung erhoben.

Diese Daten werden für das gesamte Projekt und einzelne Aktivitäten archiviert. Dabei ist der Begriff der Aktivität auf einer ähnlichen Abstraktionsebene definiert wie in COCOMO II (Boehm, 2000), orientiert vor allem an den Dokumenten (Jones, 1996): Anforderungen, Entwurf, Benutzerdokumentation, Testdokumentation. Diese Daten werden in 6 Domänen und 6 Umfangsklassen aufgegliedert. Die 6 Domänen sind Endbenutzer-Software, Informationssysteme, Software in Auftragsprojekten, Software für den Markt, Software für das Militär und Systemsoftware. Die 6 Umfangsklassen sind durch den Function-Point-Umfang in 10er-Potenzen definiert, von der Klasse mit Software unter 1 Function Point bis zur Klasse mit Software ab 10 000 bis 100 000 Function Points. Größere Projekte sind nicht dargestellt.

Die Daten werden teilweise als Absolutwerte gezeigt, sind aber meist aber durch den Umfang normiert. In einigen Fällen wird der Mittelwert durch Minimum und Maximum ergänzt.

Die Daten werden unter unterschiedlichen Blickwinkeln betrachtet und ausgewertet:

In Jones (1996) werden diese Daten und ihre Anwendung gezeigt, beispielsweise die Messung von Verbesserungen und die Identifikation der besten Methoden. Der Schwerpunkt liegt aber auf dem Datenarchiv, also dem Metrikprogramm und seinen Definitionen. Analysen und Präsentation der Resultate werden beispielhaft dargestellt.

Jones (2003) stellt strategische Aspekte in den Vordergrund. Dazu gehört der Vergleich mit anderen Organisationen und die Identifikation der erfolgreichsten und der schädlichsten Methoden. Dabei werden vor allem Daten verwendet, die das gesamte Projekt charakterisieren.

Detaillierter wird in Jones (2007) beschrieben, wie die Kosten einzelner Aktivitäten geschätzt werden können, untermauert mit Durchschnittsdaten aus dem Archiv. Das Werkzeug KnowledgePLAN (SPR, 2009) enthält die Archivdaten und macht sie zugänglich. Das Werkzeug bietet Resultate für die Kostenschätzung, also Dauer, Aufwand und Personalbedarf, zusätzlich die Fehlerzahl und Fehlerdichte für das gesamte Projekt und einzelne Phasen.

Bewertung und Folgerungen

Die aktivitätsbezogenen Daten sind eine wichtige Basis für das QS-Modell (Drappa, 1998). Die Datensammlung ist umfangreich und enthält Industriedaten. Sie enthält somit Mittelwerte vieler Projekte, repräsentiert also Projekte im Allgemeinen.

Da die Daten aktivitätsbezogen dargestellt werden, können sie als Grundlage für Modelle dienen, die einzelne Aktivitäten abbilden. Da mit CoBe Aktivitäten und Auswirkungen der Prüfungen auf Aktivitäten dargestellt werden sollen, verwende ich die Daten von Jones, um das Modell CoBe zu quantifizieren.

Die Ziele, die mit CoBe verfolgt werden, können aber mit dieser Datensammlung nicht erreicht werden, da Jones vor allem die Resultate, aber keine Korrelations- oder Ursache-Wirkungs-Beziehungen analysiert und beschreibt. Zusammenhänge, die zur Modellbildung verwendet werden können, werden nicht explizit genannt. Diese Zusammenhänge können aber anhand der Daten analysiert und interpretiert werden.

Das Werkzeug KnowledgePlan enthält solche Zusammenhänge, macht sie aber nicht öffentlich. Prüfparameter werden nicht dargestellt, somit werden auch keine Entscheidungen über die Prüfparameter unterstützt. Fehlerfolgekosten beim Einsatz des Produkts und Kosten in der Wartung spielen in den Diskussionen so gut wie keine Rolle, weil das Augenmerk vor allem auf dem Projekt liegt.

Da Mittelwerte vieler Projekte gezeigt werden, können die Werte eines speziellen Projekts abweichen. Darum ist eine Aussage, wie gut die Werte von Jones auf ein spezielles Projekt übertragen werden können, nicht möglich.

4.4 Weitere Kosten-Nutzen-Modelle

Es gibt eine Reihe weiterer quantitativer Modelle, die sich mit Kosten und Nutzen von Prüfungen auseinandersetzen. Im Folgenden werden diejenigen Modelle betrachtet, die Tests und Reviews enthalten und darum mit dem konzipierten Kosten-Nutzen-Modell CoBe verglichen werden können.

4.4.1 COCOMO-Erweiterungen

Kosten und Nutzen von Qualitätsverbesserungen werden in iDave (Information Dependability Attribute Value Enhancement) hinsichtlich der Zuverlässigkeit bewertet (Huang und Boehm, 2006; Boehm et al., 2003; Boehm et al., 2004). Das Modell verwendet COCOMO II und COQUALMO, ein Fehlerstrommodell (Boehm, 2000). Entscheidungen über Prüfungen werden durch die Prozessreife der Prüfungsarten

dargestellt. Tabelle 4 zeigt dies beispielhaft für Reviews, das gleiche Vorgehen wird für Tests und für die automatisierte Analyse angewendet.

Rating	Peer Reviews
Very Low	No Peer review.
Low	Ad-hoc informal walkthroughs.
Nominal	Well defined sequence of preparation, review, minimal follow-up.
High	Formal review roles with well-trained participants and using basic checklists, follow-up.
Very High	Basic review checklists, root cause analysis. Formal follow-up using historical data on inspection rate, preparation rate, fault density.
Extra High	Formal review roles and procedures. Extensive review checklists, root cause analysis. Continuous review process improvement. Statistical Process Control.

Tabelle 4: Bewertung der Prozessreife von Reviews (Boehm et al., 2004)

Für den Vergleich zwischen Kosten und Nutzen werden Dauer und Qualität durch VERs (Value estimating relationships) auf Geldwerte abgebildet. Diese Beziehungen zwischen Prozess- und Produktqualität auf Geldwerte stammen aus einer Wirtschaftlichkeitsbewertung durch Klienten, sind also nicht vom Modell vorgegeben. Die Beziehungen können unterschiedliche Formen haben, es kann sich um lineare, stufenförmige, s-förmige oder einer Pareto-Verteilung folgende Funktionen handeln. iDave bietet einige einfache generische Beziehungen zwischen Prozess- und Produktqualität und Geldwerten. Dazu gehört, wie viel ein Fehler oder eine bestimmte Ausfalldauer kostet. Es können aber auch Risikokosten zur Zuverlässigkeitsbewertung einbezogen werden.

Bewertung und Folgerungen

Mit iDave wird ein ähnliches Ziel wie mit CoBe verfolgt: Kosten und Nutzen von Qualitätsverbesserungen sollen dargestellt werden. Der Blickwinkel von iDave liegt aber auf strategischer Ebene:

iDave unterstützt nicht Entscheidungen über einzelne Prüfungen und einzelne Prüfparameter, sondern betrachtet die Prozessreife der Prüfungen und ihre Auswirkungen. Damit unterscheidet es sich wesentlich von CoBe.

Der Nutzen der Maßnahmen wird in iDave nicht direkt dargestellt, sondern ist nur durch Vergleich der Resultate unterschiedlicher Eingaben möglich.

Wartungskosten und Prüfungen in der Wartung werden nicht dargestellt. Wirkungen beim Einsatz, etwa durch Fehlerfolgekosten, sind durch generische Beziehungen in iDave abgebildet. Eine direkte Unterstützung der Projektleiter oder QS-Verantwortlichen für eine Einschätzung der Fehlerfolgekosten ist also nicht in iDave enthalten.

Wie im QS-Modell liegt iDave ein Fehlerstrommodell zu Grunde. Dieses Fehlerstrommodell COQUALMO entspricht im Grundsatz dem Fehlermodell des QS-Modells.

4.4.2 El Emams Return-On-Investment-Modell

El Emam (2005) verwendet ein ROI-Modell, um Kosten und Nutzen konkreter Prozessverbesserungen zu demonstrieren. Mit dem ROI-Modell werden die Auswirkungen einer konkreten Prüfung in einer bestimmten Situation bewertet. Die Fehlerentdeckung einer Prüfung wird verwendet, um Kosten und Nutzen darzustellen. Kosten und Nutzen werden durch anfallende und entfallende Korrekturaufwände berechnet; Dauer und Mitarbeiter werden über COCOMO II abgeleitet. Aufwand wird auf Geldwerte umgerechnet. Fehlerfolgekosten für den Kunden sind die Installationskosten der Korrektur.

Bewertung und Folgerungen

Das ROI-Modell von El Emam und CoBe haben einen ähnlichen Zweck, entsprechend ähnlich ist die Modellgestaltung als funktionales Modell. Ein Fehlerstrommodell bildet die Grundlage. Qualitätskosten werden modelliert. Die gesamte Lebensdauer mit Wartung und Betrieb wird betrachtet. El Emam legt den Schwerpunkt auf Bewertungen des Return-On-Investment. Das Modell unterscheidet sich deutlich von CoBe:

- Das ROI-Modell enthält keine Entscheidungen über einzelne Prüfparameter und damit auch keine Zusammenhänge, um die Auswirkungen dieser Parameter darzustellen.
- Ein Modell, um die Planung einzelner Aktivitäten mit Dauer, Aufwand und Personalbedarf zu unterstützen, ist nicht enthalten. COCOMO II wird verwendet, um die Dauer aus dem Aufwand abzuleiten, diese Zusammenhänge werden aber nur zur ROI-Berechnung verwendet.
- Als Fehlerfolgekosten beim Einsatz werden die Installationskosten betrachtet. Die Installationskosten machen einen Teil der Fehlerfolgekosten für den Kunden und die Benutzer aus. Es fehlt eine Unterstützung der Projektleiter und QS-Verantwortlichen, um die Kosten, die für Kunde und Benutzer anfallen, einschätzen zu können. Auswirkungen der Intensität, mit der das Produkt verwendet wird, können beispielsweise nicht direkt dargestellt werden.
- Das ROI-Modell ist mit allgemeinen Daten quantifiziert; für spezielle Projekte kann es neu quantifiziert werden. Es bietet aber keine direkte Kalibrierung, um es an spezielle Projekte anzupassen.

4.4.3 Wagners Modelle zur Kosten-Nutzen-Optimierung

Wagner (2007) entwickelt zwei Modelle mit dem Ziel, Kosten und Nutzen für analytische Qualitätssicherung zu optimieren. Das Modell orientiert sich am V-Modell XT (2004). Das erste, analytische Modell ist so detailliert wie möglich. Das zweite Modell wurde für den praktischen Einsatz vereinfacht. Entscheidungen über Prüfungen wer-

den durch investierten Aufwand dargestellt. Dazu stellt das Modell unterschiedliche Funktionen zur Verfügung, beispielsweise für lineare oder exponentielle Zusammenhänge. Wagner nimmt für den praktischen Einsatz einen linearen Zusammenhang zwischen Aufwand und Fehlerentdeckung an (Wagner, 2007, S. 65). Die Modelle berechnen Prüf-, Korrektur- und Fehlerfolgekosten.

Bewertung und Folgerungen

Mit den Modellen werden ähnliche Ziele wie mit CoBe verfolgt. Sie sind auch ähnlich wie CoBe als funktionale Modelle gestaltet. Sie basieren auf einem Fehlerstrommodell und Qualitätskosten. Der Aufwand einzelner Aktivitäten wird betrachtet. Die Ziele von CoBe können aber nicht mit den Modellen von Wagner erreicht werden, weil wichtige Eigenschaften fehlen:

Die Modelle bieten keine Unterstützung für Entscheidungen über Prüfparameter, sondern modellieren den Aufwand, der investiert wird. Nicht dargestellt ist somit, auf welche Art und Weise der Aufwand in die Prüfung investiert wird und wie der Aufwand sinnvoll investiert werden kann. Unklar bleibt damit beispielsweise, ob sinnvoller in einen intensiveren Black-Box-Test investiert oder ob dieser Aufwand besser in einen Glass-Box-Test investiert wird. Nicht betrachtet wird die Wiederholung von Prüfungen.

Der Nutzen der Prüfungen wird von den Modellen nicht direkt dargestellt. Er wird erst durch Vergleich zwischen den Modellresultaten aus unterschiedlich intensiven Prüfungen sichtbar.

Dauer und Personalbedarf werden nicht dargestellt. Sie fließen nicht in die Bewertung ein; es gibt also keine direkte Unterstützung der Planung.

Fehlerfolgekosten werden als mittlere Kosten pro Fehler im Modell angegeben. Die Modelle bieten somit keine Unterstützung bei der Abschätzung dieser Kosten, etwa um die Auswirkungen der Intensität, mit der die Software eingesetzt wird, zu erfassen.

Das Modell ist mit allgemeinen Daten quantifiziert. Es wurde praktisch erprobt, bietet aber keine Möglichkeit zur Kalibrierung für spezielle Projekte oder Umgebungen.

4.4.4 Müllers Produktlinienmodell

Müller (2007) bewertet Kosten und Nutzen der analytischen Qualitätssicherung in der Produktlinienentwicklung mit dem Simulationsmodell SQASIM. Das Simulationsmodell deckt die Produktlinienentwicklung von der Architektur bis zur Implementierung ab. Die Entwicklung der Produktplattform und der einzelnen Produkte, die aus der Plattform abgeleitet werden, werden betrachtet. Anforderungsentwicklung und Wartung liegen außerhalb der Modellgrenzen.

Die wesentlichen Eingaben sind, an welchen Punkten im Prozess Prüfungen durchgeführt werden, welcher Anteil an Fehlern durch eine Prüfung entdeckt wird, die Produktkomplexität, die Mitarbeiterzahl und die Zahl der eingefügten Fehler. Die

Ausgaben sind Aufwand und Dauer einzelner Phasen und Aktivitäten, Personalbedarf und Fehlerzahlen.

Prozessänderungen werden durch Szenarien beschrieben. Müller (2007) nennt die folgenden Beispiele: Änderung von Prüfungen und ihrer Intensität, Änderung der Personalbelegung, Änderung der Fehlerverteilung auf Software-Module, Änderung der Auftragsrate und Wechsel der Produktplattform.

Bewertung und Folgerungen

Das Modell beruht auf einem Fehlerstrommodell und stellt, wie CoBe, Aufwand, Dauer und Personalbedarf einzelner Aktivitäten dar. Da der Schwerpunkt des Modells auf der Produktlinienentwicklung liegt, werden Betrieb und Wartung der Produkte nicht betrachtet. Fehlerfolgekosten und Kosten für korrektive Wartung werden nicht dargestellt. Dafür steht die Modellierung von Fehlern, ihrer Entstehung und ihrer Entdeckung in der Produktplattform und den Produkten im Vordergrund.

Entscheidungen über Prüfparameter sind im Modell nicht enthalten, auch nicht über die Wiederholung von Prüfungen. Statt dessen muss die Fehlerentdeckung direkt geschätzt werden.

Das Modell wurde in Projekten eingesetzt, es bietet aber keine Unterstützung zur Kalibrierung.

4.4.5 Prozesssimulation von Raffo et al.

Martin und Raffo (2000 und 2001) verwenden ein Modell zur Prozesssimulation, das sich an den ISO-Standard 12207 (IEEE 12207.0, 1996) anlehnt. Mit diesem Modell sollen unter anderem auch Kosten und Nutzen der Qualitätssicherung untersucht werden (Raffo, o.J.; Raffo et al., o. J.). Angestrebt wird der Einsatz in der Planung für betriebswirtschaftliche Kenngrößen (Harrison et al., 1999; Raffo, 2005). Entscheidungen sind für ganze Aktivitäten, ihre Kombination und Reihenfolge möglich. Kosten basieren auf Korrekturkosten, ausgegeben werden Aufwand, Dauer, Personal und entdeckte Fehler der Prüfungen und nach Auslieferung.

Bewertung und Folgerungen

Das Simulationsmodell bietet einen festen Satz an Prüfungen und ergibt Aufwand, Dauer und Personalbedarf der Aktivitäten. Dies entspricht in etwa den Metriken, die auch mit CoBe berechnet werden sollen. Es unterstützt aber keine Entscheidungen über einzelne Prüfparameter. Auch Entscheidungen über die Prüfwiederholung werden nicht unterstützt.

Wartung und Betrieb der Software werden im Modell nicht explizit dargestellt; Fehler können zwar prinzipiell durch Kostenfunktionen bewertet werden (Raffo, 2005), dabei werden die Modellbenutzer und -entwickler aber nicht unterstützt. Die langfristigen Auswirkungen der Prüfungen sind also im Modell nicht ausreichend bewertbar.

Der Nutzen, der durch eine bestimmte Prüfung erreicht wird, wird nicht direkt dargestellt: Dazu müssen verschiedene Simulationsläufe durchgeführt und deren Resultate verglichen werden.

4.5 Bewertungen und Folgerungen

Die Modelle können nicht direkt verwendet werden, weil die einzelnen Prüfparameter nicht abgebildet werden. Die Modelle bilden im Wesentlichen ab, welche Prüfungen stattfinden. Somit werden detaillierte Entscheidungen über Prüfungen nur unzureichend unterstützen. Eine Ausnahme bilden die Reviews in SESAM, die detaillierter modelliert sind. Der Spieler kann beispielsweise entscheiden, ob mehr oder weniger Gutachter teilnehmen (Drappa, 1998; Hampp, 2001). Konkrete Entscheidungen über den Test, beispielsweise über die Testüberdeckung oder Testvorbereitung, werden dagegen auch in diesen Modellen nicht dargestellt.

Ein weiterer Grund, warum bestehende Modelle nicht direkt übernommen werden können, ist, dass die vorhandenen Modelle die langfristigen Kosten nicht oder nur unzureichend darstellen. Dazu gehören Wartungskosten für Korrektur und Testwiederholung, insbesondere aber die Kosten, die Kunden und Benutzer tragen. Diese Kosten werden, wenn überhaupt, unvollständig oder sehr grob dargestellt. Somit werden Kosten und Nutzen für die direkten Klienten des Projekts, also Benutzer, Kunde und Wartungspersonal, nicht oder unvollständig abgebildet. Beispielsweise werden die Fehlerfolgekosten durch Installationskosten der Korrektur (El Emam, 2005) modelliert oder müssen direkt als Kosten pro Fehler angegeben werden (Wagner, 2007; Huang und Boehm, 2006).

Die grundlegenden Begriffe und Zusammenhänge in Software-Projekten sind in ähnlicher Form in allen Modellen enthalten. Dazu gehören das Fehlerstrommodell, der Anstieg der Korrekturkosten mit der Latenzzeit oder der Einfluss des Software-Umfangs. Diese Zusammenhänge sind empirisch belegt und als Teil der Modelle validiert. Sie können also in einem Kosten-Nutzen-Modell für Software-Prüfungen verwendet werden.

Kapitel 5

Analyse der Kosten und des Nutzens von Prüfungen

Da sich die Ziele von CoBe mit bestehenden Modellen nicht erreichen lassen, werden in diesem Kapitel die Zusammenhänge für Kosten und Nutzen von Prüfungen analysiert. Diese Analyse ist der erste Schritt der Modellbildung (Abschnitt 3.7). Dabei werden auch die bereits bestehenden Modelle und andere empirische Arbeiten berücksichtigt, um Teile aus diesen Modellen zu verwenden. Dazu gehören die grundlegenden Begriffe und Zusammenhänge in Software-Projekten. Diese Basiszusammenhänge werden in Abschnitten 5.1 bis 5.3 gezeigt. Die Analyse von Prüfungen ist in den Abschnitten 5.4 bis 5.6 dargestellt.

5.1 Begriffe

Als ersten Schritt der Modellbildung werden im Folgenden die grundlegenden Zusammenhänge beschrieben. Ich beginne mit den Begriffen, bevor die Zusammenhänge beschrieben werden.

Fehler und Fehlermerkmale

Für Fehler gibt es unterschiedliche Definitionen (Abschnitt 2.7; IEEE 1044, 1993; IEEE 982.1, 2005; IEEE 610, 1990; ISO 9000, 2000). Im QS-Modell (Drappa, 1998) wird die Definition aus IEEE 1044 (1993) für Fehler in einem Artefakt verwendet:

Def. **anomaly.** Any condition that deviates from expectations based on requirements specifications, design documents, user documents, standards, etc. or from someone's perceptions or experiences. (IEEE 1044, 1993)

Diese Definition entspricht dem Fehlerbegriff der ISO 9000 (2000), wenn Anforderungen aller Klienten und implizite Anforderungen einbezogen werden:

Def. **Fehler.** Nichterfüllung einer Anforderung. (ISO 9000, 2000)

Unterschieden werden zwischen Abweichungen (anomaly) in einem Artefakt, z.B. in einem Dokument oder im Code (Florac, 1992) und dem Fehlverhalten oder failure (IEEE 610, 1990; IEEE 982.1, 2005; Liggesmeyer, 2002):

Def. **failure.** The inability of a system or component to perform its required functions within specified performance requirements. (IEEE 610, 1990)

Def. **Fehlverhalten**. Ein Fehlverhalten oder Ausfall (failure) zeigt sich dynamisch bei der Benutzung eines Produkts. Beim dynamischen Test einer Software erkennt man keine Fehler, sondern Fehlverhalten bzw. Ausfälle. Diese sind Wirkungen von Fehlern im Programm.

Def. **fault.** (1) A defect in a hardware device or component; for example, a short circuit or broken wire. (2) An incorrect step, process, or data definition in a computer program. (IEEE 610, 1990)

Def. **Fehler**. Ein Fehler oder Defekt (fault, defect) ist bei Software die statisch im Programmcode vorhandene Ursache eines Fehlverhaltens oder Ausfalls.

Fehler haben unterschiedliche Merkmale, die als Kategorie bezeichnet werden (IEEE 1044, 1993). Jede Kategorie besteht aus Klassen. Ein Fehler wird einer Klasse in einer Kategorie durch Klassifizierung zugeordnet, beispielsweise der Klasse "dringend" in der Kategorie "Priorität". Zwei wichtige Kategorien sind die Fehlerart und die Fehlerschwere, weil sie mit Kosten zusammenhängen. Die Fehlerart beschreibt die Aktivität, bei der ein Fehler gemacht wird (Drappa, 1998). Sie wird auch als Abstraktionsebene der Fehlerentstehung (Drappa, 1998; Ludewig und Lichter, 2007) oder Origin (Runeson et al., 2006) bezeichnet:

Def. **Fehlerart.** Die Fehlerart ist durch die Aktivität bestimmt, durch die ein Fehler entstanden ist.

Im QS-Modell werden etwa Analysefehler, Feinentwurfsfehler, Grobentwurfsfehler, Implementierungsfehler und Handbuchfehler unterschieden (Drappa, 1998).

Die Fehlerschwere beschreibt die Auswirkungen eines Fehlers auf die Software-Entwicklung und den Einsatz der Software (IEEE 610, 1992):

Def. **criticality.** The degree of impact that a requirement, module, error, fault, failure or other item has on the development or operation of a system. Syn: severity. (IEEE 610, 1992)

Frühauf et al. (2006) nennen für Prüfungen, insbesondere für Reviews, drei Klassen, die durch Auswirkungen auf Projekt und Betrieb definiert sind:

Def. **Kritischer Fehler.** Prüfling ist für den vorgesehenen Zweck unbrauchbar, Fehler muss vor der Freigabe behoben werden.

Def. **Hauptfehler.** Nutzbarkeit des Prüflings ist beeinträchtigt, Fehler sollte vor Freigabe behoben werden.

Def. **Nebenfehler.** beeinträchtigen den Nutzen kaum.

Die Fehlerschwere kann unterschiedlich definiert und bewertet werden, abhängig von den betrachteten Auswirkungen eines Fehlers. Bassin et al. (2002) betrachten beispielsweise die Auswirkungen auf den Test und unterscheiden auch blockierende Fehler:

Def. **Blockierender Fehler.** Ein blockierender Fehler verhindert die weitere Ausführung des Programms, etwa um Testfälle durchzuführen. (Bassin et al., 2002).

Für verschiedene Fehlerwirkungen nennt der IEEE-Standard 1044 (1993) verschiedene Kategorien. Dazu gehören Dringlichkeit, Auswirkungen auf Projektkosten und Projektdauer und Auswirkungen, die der Fehler beim Einsatz hat (oder hätte). Für jedes Merkmal sind Klassen vorgegeben. Fenton und Pfleeger (1997), Jones (1996, S. 232, S. 367), Dunn (1984), Grady und Caswell (1987) zeigen Beispiele, bei denen sich die Fehlerschwere auf den Schaden bezieht, der beim Einsatz der Software entsteht.

Software-Umfang

Der Umfang des Produkts kann in Function Points mit unterschiedlichen Varianten (ISO/IEC 14143, 2007; Jones, 2007; IFPUG, 2004) gemessen werden. Der Umfang des Codes kann in Anweisungen (logische Zeilen, durch Sprachelemente definiert) oder Lines of Code (physische Zeilen, durch Zeilenumbrüche definiert) gemessen werden (IEEE 1045, 1992; Park, 1992). Im Folgenden verwende ich Anweisungen, wenn logische Zeilen gemeint sind, und Codezeilen oder Zeilen, wenn physische Zeilen gemeint sind. Der IEEE-Standard unterscheidet, ob Software in einem Projekt neu erstellt (hinzugefügt), geändert oder unverändert wiederverwendet wird. Dieses Merkmal wird als Ursprung (Origin) bezeichnet:

Def. **Hinzugefügte Software.** Software, die im Projekt neu erstellt wird.

Def. **Geänderte Software.** Software, die bereits vorhanden war und im Projekt geändert wird.

Def. **Wiederverwendete Software.** Software, die bereits vorhanden war und unverändert im Projekt verwendet wird.

Def. **Neue Software.** Hinzugefügte und geänderte Software.

Function Points erlauben, den Umfang der Software in allen Projektphasen zu beschreiben (Jones, 1996; Drappa, 1998). Zwischen Code-Umfang und Function Points kann umgerechnet werden (Jones, 1996; Boehm, 2000). Der Faktor für die Umrechnung hängt von der Programmiersprache ab. Im QS-Modell wird der Umfang aller Dokumente durch Function Points beschrieben (Drappa, 1998). In COCOMO II können die Zahl der Anweisungen, bezeichnet als SLOC, oder die Zahl der Function Points verwendet werden (Boehm, 2000). Der Zusammenhang zwischen Anweisungen und Function Points schwankt weniger stark als zwischen Lines of Code und Function Points (Jones, 1996). Der Umfang von Dokumenten kann in Seiten gemessen werden (IEEE 1045, 1992). Zwischen Function Points und Seiten kann umgerechnet werden (Drappa, 1998; Jones, 2007). Die Faktoren hängen von der gewählten Notation und Methode zur Dokumentation ab (Drappa, 1998; Jones, 2007).

5.2 Analyse der Fehlerentstehung, -entdeckung und -korrektur

Entwicklungs- und Prüfprozesse

Bei der Software-Entwicklung gehen die einzelnen Dokumente durch einen kreativen Prozess auseinander hervor (Ludewig und Lichter, 2007). Der Entwickler ist dabei schöpferisch tätig. Beispielsweise setzt ein Entwickler Anforderungen in einen Entwurf um; er entwirft. Dabei bleiben Informationen aus der Spezifikation erhalten, andere kommen hinzu, andere fallen weg. Drappa (1998) nennt explizit, dass von einer Vorgabe ausgehend auf die nächstniedere Abstraktionsebene der Entwicklung transformiert wird, dass also die Spezifikation immer den Entwurf vorgibt. Das bedeutet nicht, dass der Entwurf immer dokumentiert werden muss, er kann auch im Kopf des Entwicklers entstehen. Für die Software-Entwicklung werden von Drappa (1998), Jones (1996) und Boehm (2000) und von Begriff- und Prozessstandards (IEEE 610, 1990; Automotive SIG, 2005; CMMI Product Team, 2002; V-Modell XT, 2004) typisch die Spezifikation, Architektur- und Feinentwurf, Implementierung und Integration genannt. Die Integration kann mehrere Schritte umfassen. Ein typischer Prüfprozess mit typischen Prüfungen wird z.B in Jones (1996) dargestellt. Im Prozess werden Dokumente Reviews unterzogen (Spezifikationsreview, Entwurfsreview, Codereviews). Der Test findet auf unterschiedlichen Ebenen statt, mit Modultest, Integrationstest in einzelnen Schritten, Systemtest. Zusätzlich kann die Software durch einen Feldtest beim Kunden geprüft werden. Mit der automatischen statischen Codeanalyse, im Folgenden kurz Codeanalyse genannt, werden verdächtige Konstrukte im Code von einem Werkzeug identifiziert und dokumentiert (Spinellis, 2006; Louridas, 2006).

Fehlerentstehung

Die Zahl der entstehenden Fehler hängt vom Umfang des Produkts ab (Jones, 1996). Fenton und Pfleeger (1997) bezeichen die Fehlerdichte als De-Facto-Standard für die Software-Qualität. Sie ist definiert als die Zahl der Fehler bezogen auf den Software-Umfang. In vielen Fällen werden Erfahrungswerte der Fehlerdichte berichtet (z.B. in Jones, 1996; Kan, 2003; Grady, 1992).

Der Zusammenhang zwischen Umfang und Fehlerzahl ist überproportional. Er wird in den Modellen von Drappa (1998), Huang und Boehm (2006); El Emam (2005) und Martin und Raffo (2000 und 2001) für vollständig neu entwickelte Software verwendet. Software-Änderungen werden in den Modellen nicht direkt betrachtet. Möller und Paulish (1993b) stellen fest, dass sich die Fehlerdichte zwischen neuen und geänderten Modulen bei größerem Änderungsumfang (etwa ab 70 Zeilen) nicht unterscheidet. Basili und Perricone (1984) zeigen etwas mehr Fehler in geänderten Modulen als in neu entwickelten Modulen und machen dafür die falsche Verwendung vorhandener Schnittstellen verantwortlich.

Fehler entstehen bei den Aktivitäten der Software-Entwicklung und werden in das Artefakt eingefügt, das bearbeitet wird (Drappa, 1998). Daraus folgt, dass Fehler bei der Erstellung von Software, aber auch bei der Korrektur von Software entstehen können.

Die Zahl der entstehenden Fehler hängt von vielen Merkmalen ab. Dazu gehört die Projektart (Jones, 1996) und die Prozessreife; in Projekten mit hoher Prozessreife werden halb so viele Fehler (Jones, 1996) oder noch weniger (Gibson et al., 2006) insgesamt entdeckt. Komplexität und Struktur spielen eine Rolle (El Emam et al., 2001; Jiang et al., 2008; Olague et al., 2007), der Einfluss lässt sich aber nicht mehr nachweisen, wenn der Umfang einzelner Klassen einbezogen wird (El Emam et al., 2001; Olague et al., 2007). Für das Modell COQUALMO (Boehm, 2000; Devnani-Chulani, 1997) wurden Experten in einer Delphi-Befragung gefragt, wie stark sich die Fehlerzahl abhängig von COCOMO-II-Parametern verändert. Eine Messung und Validierung des Einflusses dieser Merkmale fand bislang nicht statt.

Die Art des Software-Projekts bestimmt, wie sich die Fehler auf die Fehlerarten verteilen (Jones, 1996). Die konkreten Einflüsse auf die Fehlerzahl und die Verteilung auf die Fehlerart sind nicht bekannt. Die Verteilung der Fehler auf die Klassen der Fehlerschwere hängt von der Definition der Fehlerschwere ab.

Fehlerentdeckung und Fehlerkorrektur

Jede Prüfung entdeckt einen Anteil der Fehler, die im Prüfling enthalten sind. Dieser Anteil wird als Fehlerentdeckungsquote Q (Drappa, 1998; Kan, 2003) bezeichnet.

$$Q = \frac{\text{Zahl durch Prüfung entdeckter Fehler}}{\text{Zahl enthaltener Fehler}}$$

Kan (2003) definiert die Defect Removal Effectiveness DRE nicht mit entdeckten, sondern entfernten Fehlern. Die Zahl der enthaltenen Fehler ist in realen Projekten nicht bekannt, darum wird die Fehlerentdeckungsquote für reale Projekte definiert durch

$$Q = \frac{\text{Zahl in Prüfung entdeckter Fehler}}{\text{Zahl in Prüfung entdeckter Fehler + Zahl nach Prüfung entdeckter Fehler}}$$

Prüfungen entdecken Fehler systematisch auf einer bestimmten Abstraktionsebene (Frühauf et al., 2006; Drappa, 1998). Die Fehlerentdeckungsquote unterscheidet sich also für verschiedene Fehlerarten (Jones, 1996).

Entdeckte Fehler werden in der Korrektur behoben. Dabei können aber durch die Änderung neue Fehler gemacht werden, außerdem kann die Korrektur unvollständig sein (Drappa, 1998). Das Fehlerstrommodell beschreibt, dass Fehler während Entwicklung (und Korrektur) eingefügt und durch Prüfung entdeckt, dann mit der Korrektur entfernt werden. Abbildung 14 nach Boehm (1981, S. 382) zeigt dies vereinfacht, ohne die Fehlerentstehung durch Korrektur.

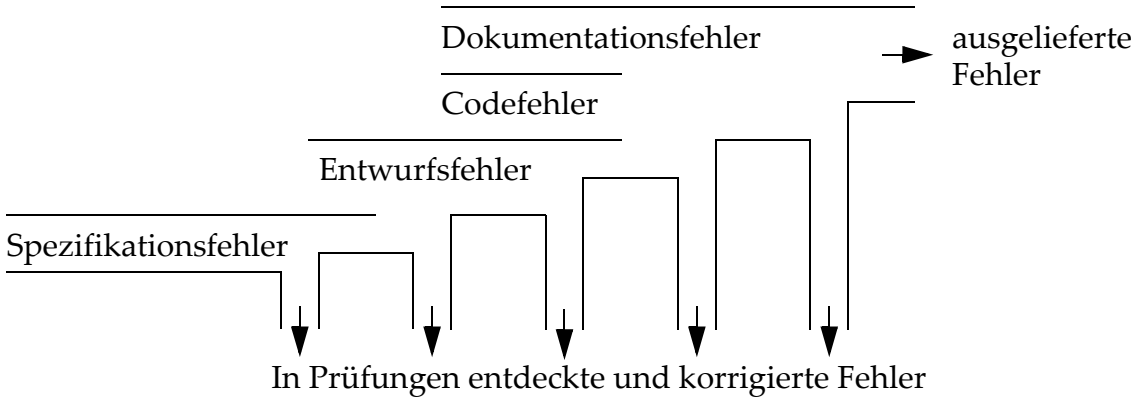

Abb. 14: Das Fehlerstrommodell (nach Boehm, 1981)

Drappa (1998) beschreibt dies anhand der Entwicklung mit Vorgabe. Dabei werden Fehler aus der Vorgabe übernommen. Das Fehlerstrommodell bildet die Grundlage aller Modelle, die sich mit Kosten und Nutzen von Prüfungen auseinandersetzen (Huang und Boehm, 2006; El Emam, 2005; Wagner, 2007; Müller, 2007; Martin und Raffo, 2000 und 2001).

Fehlerentstehung, Fehlerentdeckung und die Korrektur hängen mit den Erfahrungen und Kenntnissen der beteiligten Entwickler zusammen (Drappa, 1998).

5.3 Analyse von Fehlerkosten

Fehlerkosten sind unterteilt in Fehlerbehebungskosten und Fehlerfolgekosten. Fehlerbehebungskosten sind alle Kosten, die mit der Behebung des Fehlers einschließlich der Prüfung der Korrektur zu tun haben. Fehlerfolgekosten entstehen durch Fehler beim Einsatz der Software (Abschnitt 2.9).

5.3.1 Fehlerbehebungskosten

Fehlerkorrektur. Die Korrektur eines Fehlers erfolgt in mehreren Schritten. Falls ein Fehlverhalten entdeckt wird, beispielsweise im Test oder beim Einsatz der Software, dann muss bei der Korrektur zuerst die Ursache für dieses Fehlverhalten identifiziert werden. Dies wird als Fehleranalyse bezeichnet. Falls der Fehler direkt identifiziert wurde, beispielsweise in einem Review oder durch Codeanalyse, entfällt diese Ursachenanalyse. Dann wird die Korrektur entworfen und implementiert (Basili et al., 1996; Sommerville, 2007, Kap. 22), d.h. die Software wird geändert. Je länger ein Fehler unentdeckt bleibt, desto teurer wird die Korrektur. Die Dauer, die der Fehler unentdeckt bleibt, wird als Latenzzeit bezeichnet. Diese hängt ab von der Prüfung, bei der der Fehler entdeckt wurde (Boehm, 1976; Möller und Paulish, 1993a; Kan, 2003), und der Aktivität, bei der der Fehler gemacht wurde (Drappa, 1998; Ludewig und Lichter, 2007; Kan, 2003). Beispielsweise kostet ein Fehler, der beim Spezifizieren

gemacht wurde, das zehnfache, wenn er im Systemtest anstatt im Spezifikationsreview entdeckt wird. Auf diesem Zusammenhang basieren die Modelle, die sich mit Kosten und Nutzen von Prüfungen auseinandersetzen (El Emam, 2005; Wagner, 2007; Müller, 2007; Martin und Raffo, 2000 und 2001; Huang und Boehm, 2006). Wie stark der Korrekturaufwand ansteigt, hängt vom Umfang der Software ab; in kleinen Projekten steigt der Aufwand weniger stark als in großen Projekten (Boehm, 1981 und 1976). Zusätzlich spielt die Fehlerschwere eine Rolle: Je schwerer der Fehler, desto aufwändiger die Korrektur (Zage und Zage, 2003; Kan, 2003).

Falsche Befunde. Wenn eine Prüfung eine Abweichung zeigt, bei der sich etwa bei der Fehleranalyse herausstellt, dass es sich nicht um einen Fehler handelt, wird dieses Prüfresultat als falscher Befund bezeichnet. Falsche Befunde können in jeder Prüfung entstehen. In Reviews werden diese Befunde in der Regel in der Sitzung identifiziert (Sabaliauskaite et al., 2002; Votta, 1993). In Tests können falsche Befunde durch Fehlbedienung entstehen. In der Codeanalyse werden sehr viele falsche Befunde angezeigt (Zheng et al., 2006). Sabaliauskaite et al. (2002) beschreiben den Lebenslauf eines falschen Befunds. Ein falscher Befund wird während der Fehleranalyse als falscher Befund klassifiziert. Diese Analyse gehört zur Korrektur (Pressman, 2005, Kap. 13; Sommerville, 2007, Kap. 22); es fällt Analyseaufwand an. Behebungsaufwand oder Aufwand für die Wiederholung der Prüfung fällt für einen falschen Befund nicht an.

Prüfwiederholung nach der Korrektur. Nach der Korrektur kann die Prüfung wiederholt werden. Der Umfang der Wiederholung ist unterschiedlich und spielt sich zwischen vollständiger und gezielter Wiederholung ab (Thaller, 2002; Sneed et al., 2004; Müller et al., 1998). Bei einer vollständigen Wiederholung wird die Prüfung erneut für den gesamten Prüfling durchgeführt. Bei einer gezielten Wiederholung wird gezielt überprüft, ob der Fehler korrigiert wurde. Es werden also diejenigen Testfälle weggelassen, von denen vermutet wird, dass sie durch die Änderung nicht betroffen sind.

Ob eine Wiederholung durchgeführt wird, ob die Prüfung vollständig oder gezielt wiederholt wird, kann in der Wartung und in der Entwicklung unterschiedlich sein. Bei der Entwicklung kann in Projekten diejenige Prüfung wiederholt werden, bei der der Fehler entdeckt wurde. In anderen Projekten wird die Korrektur durch eine Reihe von Prüfungen überprüft (Hörmann et al., 2006), einem Korrekturprüfprozess. Ein solcher Korrekturprüfprozess kann auch in der Wartung nach der Korrektur durchgeführt werden (ISO/IEC 14764, 1999; Pigoski, 1997). Dabei kann der Korrekturprüfprozess von der Fehlerschwere abhängen, weil beispielsweise besonders schwerwiegende Fehler und weniger schwerwiegende Fehler unterschiedlich behandelt werden und ihre Korrektur unterschiedlichen Prüfungen unterzogen wird (Sneed et al., 2004). In allen Fällen kann die Änderung in einem Review begutachtet werden (Hörmann et al., 2006; ISO/IEC 14764, 1999; Pigoski, 1997). Die Kosten für die Wiederholung eines Tests hängen vom Umfang der Wiederholung und vom Grad der Automatisierung des Tests ab (van Megen und Meyerhoff, 1995).

Prüfung. Blockierende Fehler sind Fehler, die den Test unterbrechen. Sie verursachen durch die Unterbrechung Kosten im Test (Kan, 2003; Bassin et al., 2002). Soll der Test nach der Korrektur weiter durchgeführt werden, dann muss erst der Zustand der Testumgebung, in der der Prüfling ausgeführt wird (IEEE 610, 1990), wiederhergestellt werden. Bassin et al. (2002) beschreiben: *"Blocked" status was used when the test case attempt did not succeed because access to the targeted area was blocked by code that was not functioning correctly.* Kan (2003) nennt diese Fehler "Showstopper". Sie gehören zu den kritischen Fehlern.

5.3.2 Fehlerfolgekosten und Zuverlässigkeit

Fehler zeigen sich beim Einsatz des Produkts als Fehlverhalten: ein Fehler tritt auf. Ein Fehler tritt abhängig von der Art und Intensität der Verwendung der Software durch die Benutzer mehr oder weniger häufig auf (Fenton und Pfleeger, 1997). Beispielsweise kann die Software sehr häufig und auf eine bestimmte Art verwendet werden, dabei werden manche Funktionen mehr, andere Funktionen weniger stark genutzt. Darum kann ein Fehler gar nicht, einmal oder mehrmals auftreten.

Die Zuverlässigkeit wird quantitativ durch Zuverlässigkeitsmetriken, Zuverlässigkeitsmodelle zur Prognose und Zuverlässigkeitstests dargestellt. Zuverlässigkeitsmetriken, beispielsweise MTBF (mittlere Betriebsdauer zwischen Ausfällen, mean time between failures) basieren auf der Häufigkeit des Fehlverhaltens, typisch bezogen auf die Einsatzdauer (IEEE 982.1, 2005). Zuverlässigkeitsmodelle prognostizieren diese Zuverlässigkeit (Lyu, 1995). Sie basieren auf Messungen der Zuverlässigkeit im Test oder auf der Fehlerdichte. Die Auswahl der Zuverlässigkeitsmodelle ist schwierig, weil unklar ist, welches Modell in welcher Situation ausreichend genau ist; der Test muss dem Einsatz ähneln (Pul, 1993). Die Modelle müssen mit vorhandenen Daten quantifiziert werden (Kan, 2003), dabei ist unklar, ob Archivdaten eingesetzt werden können. Zuverlässigkeitstests bestimmen die Zuverlässigkeit, erlauben also auch eine Prognose (Poore und Trammell, 1996). Dafür wird ein Benutzungsprofil benötigt, das beschreibt, wie das Produkt verwendet wird. In diesen Zuverlässigkeitsmetriken, -modellen und -tests werden alle Fehler und ihr Auftreten als gleich schwerwiegend bewertet.

Der Schaden, den ein Fehler beim Einsatz der Software verursachen kann, kann unterschiedliche Wirkungen haben und unterschiedlich hoch sein. Für eine subjektive Einschätzung der Schadenshöhe nennt der IEEE-Standard 1044 (1993) unterschiedliche Kategorien, um zu beschreiben, wie sich der Fehler auswirkt: Severity, Priority, Customer value, Mission safety, Societal. Diese Aspekte beziehen sich nicht auf den Hersteller, sondern auf Klienten des Projekts. Für alle diese Kategorien gibt es Klassen für die Bewertung, die von keinem Schaden bis zu hohem Schaden reichen; für sicherheitskritische Software reicht die Spanne von leichten Verletzungen bis zu mehreren Todesfällen (Smith und Simpson, 2005). Boehm (2000) und Huang und Boehm (2006) verwenden fünf Klassen, um die verlangte Zuverlässigkeit zu beschreiben. Diese Klassen sind definiert durch den Schaden, der verursacht werden kann. Sie reichen von Komfortproblemen bis zu Personenschaden. El Emam (2005) berechnet die Folge-

kosten aus der Zahl der Fehler, die ein Kunde entdeckt, und den organisatorischen Kosten für den Kunden. Wagner (2007) berechnet Fehlerfolgekosten getrennt nach Fehlerschwere aus der Fehlerzahl und einem durchschnittlichen Schaden pro Fehler. Nicht berücksichtigt wird aber, wie häufig die Benutzer das Produkt nutzen (die Verwendungshäufigkeit) und wie intensiv und auf welche Art ein Benutzer das Produkt nutzt (die Verwendungsintensität).

Ludewig und Lichter (2007) diskutieren den Zuverlässigkeitsbegriff ausgehend von der Unzuverlässigkeit. Sie definieren eine Unzuverlässigkeitsmetrik als die Zahl der Fehler, gewichtet mit der Häufigkeit ihres Auftretens und den Folgekosten, bezogen auf die Betriebsdauer. Ein Fehler wird also mit dem Schaden, den er im Einsatz anrichtet, bewertet. Der Schaden hängt von der Häufigkeit des Auftretens und den Kosten jedes einzelnen Auftretens ab.

5.3.3 Organisationsaufwand

Zusätzlich zu den Kosten der Aktivitäten, die zur Fehlerbehebung durchgeführt werden, und den Fehlerfolgekosten, die für den Benutzer anfallen, fallen Kosten für weitere Aktivitäten an: Bei der Software-Bearbeitung werden unterstützende Aktivitäten durchgeführt, da Prüfung und Korrektur geplant und organisiert werden müssen (Quality Management in PMI, 2000). Fehlerkorrekturen und die Prüfungen der Korrektur können dokumentiert werden (IEEE 1044, 1992), etwa um die Fehleranalyse zu unterstützen oder zur Kontrolle der Fehlerbehebung. Kosten für die Verwaltung der Software können anfallen (IEEE 828, 2005), im Wesentlichen zur Definition, Verfolgung, Speicherung und Rückverfolgung betroffener Software-Einheiten.

5.3.4 Aufwand, Dauer und Personalbedarf

Zur Planung von Software-Projekten und zur Kostenschätzung gehört die Schätzung von Aufwand, Dauer, Personalbedarf und Kosten als Geldwerte (CMMI Product Team, 2002; Sommerville, 2007; Kerzner, 2006). Dabei werden die Kosten als Geldwerte aus dem Aufwand, der Dauer und dem Personalbedarf einzelner Aktivitäten abgeleitet (Kerzner, 2006).

Kostenschätzverfahren wie COCOMO II prognostizieren darum den Aufwand, die Dauer und den typischen, idealen Personalbedarf. Dabei geht COCOMO II von einer typischen, idealen Stellenbesetzung und Organisation der Arbeit aus, um Aufwand, Dauer und Personalbedarf zu berechnen. Im Gegensatz dazu wird dies im QS-Modell nicht vorgegeben, da die erfolgreiche Planung zu den Lernzielen des QS-Modells gehört. Den Aufwand zu schätzen und den Personalbedarf zu bestimmen ist in SESAM-Schulungen die Aufgabe des Spielers. Er soll beispielsweise selbst festlegen, wie viele Mitarbeiter für welche Tätigkeit eingesetzt werden (Drappa, 1998).

Weil mit einem Kosten-Nutzen-Modell wie CoBe die Planung unterstützt werden soll, ist aber sinnvoll, diese Größen zu prognostizieren und darum die Zusammenhänge zwischen Aufwand, Dauer und Personalbedarf für typische Situationen zu

modellieren. Die Zusammenhänge zwischen den drei Planungsmetriken Aufwand, Dauer und Personalbedarf fasse ich wie folgt zusammen:

Der Aufwand wird durch Merkmale des Projekts, des Prozesses, der Plattform und des Personals, vor allem aber durch den Umfang bestimmt. Der Umfang bestimmt den Aufwand überproportional. Die Dauer und damit der Personalbedarf hängen mit dem Aufwand zusammen (Boehm, 2000); es gibt also ein typisches, projektspezifisches Verhältnis zwischen Aufwand, Dauer und Personal.

Der Zusammenhang zwischen Umfang und Aufwand muss an die Umgebung angepasst werden können (Boehm, 2000, Kap. 5), um eine hohe Genauigkeit zu erzielen; Kostenschätzverfahren müssen für eine Umgebung kalibriert werden (Kemerer, 1987). Das Verhältnis zwischen Dauer, Aufwand und Personalbedarf kann sich in kleinen Projekten oder in Projekten, die einem anderen Prozess folgen, von den COCOMO-Werten erheblich unterscheiden. Es ist also notwendig, diese Zusammenhänge zu kalibrieren.

Der Aufwand verteilt sich, einen ähnlichen Prozess vorausgesetzt, in bestimmten Verhältnissen auf die einzelnen Phasen und Aktivitäten. Die Dauer verteilt sich, einen ähnlichen Prozess vorausgesetzt, in bestimmten Verhältnissen auf die einzelnen Phasen und Aktivitäten. Damit wirken sich die Kalibrierung, die Merkmale des Projekts, des Prozesses, der Plattform und des Personals und des Umfangs auf den Gesamtaufwand und auf den Aufwand einzelner Aktivitäten in gleichem Maße aus. Sie wirken sich auf die Gesamtdauer und die Dauer einzelner Aktivitäten in gleichem Maße aus.

Diese Zusammenhänge können angelehnt an El Emam (2005) direkt übernommen werden, weil COCOMO II ein funktionales Modell ist (Abschnitt 3.7). Damit sind die Metriken aus COCOMO II vorgegeben. Die Grundlage bildet also der Software-Umfang, gemessen in Anweisungen. Alternativ können Unadjusted Function Points (IFPUG, 2004) auf Lines of Code abgebildet werden. Zusätzlich werden die 22 COCOMO-II-Merkmale verwendet, um die konkrete Situation des Projekts zu beschreiben.

5.3.5 Geldwerte

Kosten und Nutzen werden für den Vergleich durch Geldwerte dargestellt (Hanusch, 1987; Mühlenkamp, 1994). Wie in El Emam (2005) und Huang und Boehm (2006) müssen die Auswirkungen von Prüfungen also auf Geldwerte abgebildet werden. Der Aufwand im Projekt bestimmt durch die Personalkosten die Kosten des Projekts (Drappa, 1998; El Emam, 2005). Verzögerungen des Projekts können Kosten z.B. durch Vertragsstrafen oder entgangene Einnahmen nach sich ziehen (Huang und Boehm, 2006). Huang und Boehm (2006) verwenden Funktionen, um von Aufwand und Dauer auf Geldwerte umzurechnen. Langfristige Kosten oder langfristiger Nutzen wird abgezinst (Hanusch, 1987), diese Abzinsung ist aber umstritten (Mühlenkamp, 1994; Nas, 1996), wenn andere Gruppen von den abgezinsten Werten betroffen sind.

5.4 Analyse von Reviews

Der Begriff Review wird für die Begutachtung eines Prüflings verwendet (Schwinn, 2003). Als Prüfling kommen alle Dokumente und der Quellcode in Frage. Freedman und Weinberg (1982) und Frühauf et al. (2006) grenzen das technische Review von anderen Varianten ab, da der Begriff in unterschiedlichen Bedeutungen verwendet wird:

- Im technischen Review bereiten sich die Gutachter vor, ihre Befunde werden in der Sitzung gewichtet und dokumentiert. Dazu zählt auch die Inspektion mit Einführungssitzung und Vorleser (Fagan, 1976).
- Im Walkthrough stellt der Autor sein Dokument vor, die Gutachter können, müssen aber nicht vorbereitet sein. Sie stellen keine Fehler fest, sondern stellen Fragen.
- Bei der Stellungnahme begutachtet ein Kollege das Dokument. Es handelt sich dabei nicht um ein formales Review.

Technische Reviews sind am teuersten, weil sie formal ablaufen und eine intensive Begutachtung enthalten. Für diese Reviews sind umfangreiche Erfahrungen vorhanden:

Ablauf. Ein technisches Review (Freedman und Weinberg, 1982; Frühauf et al., 2006) wird vom Moderator, dem Autor, einem Notar und mehreren Gutachtern durchgeführt. Der Moderator ist für die Organisation des Reviews und den Ablauf der Sitzung verantwortlich. Der Autor steht in der Sitzung für Fragen zur Verfügung. Der Notar, auch als Protokollführer oder Sekretär bezeichnet, notiert die Befunde, die die Gutachter in der Sitzung nennen. Ein Review besteht aus der Planung, der Vorbereitung, der Sitzung mit Empfehlung über das weitere Vorgehen, der optionalen Nacharbeit (oder Korrektur) und der optionalen Nachprüfung mit Freigabeentscheidung (Frühauf et al., 2006; Freedman und Weinberg, 1982). In den Reviewmodellen von SESAM (Drappa, 1998; Hampp, 2001) wird modelliert, dass der Prüfling auf mehrere Sitzungen verteilt wird, auf die sich die Gutachter vorbereiten.

Fehlerentdeckung durch Gutachter. Die Gutachter bereiten sich getrennt voneinander vor. Jeder Gutachter entdeckt einen Teil der Fehler; manche Fehler werden von mehreren Gutachtern entdeckt. Diese Duplikate werden in der Sitzung identifiziert. Biffl (2001) modellieren dies durch die Wahrscheinlichkeit, mit der ein Fehler von einem Gutachter entdeckt wird; Hampp (2001) modelliert die Fehlerentdeckung als sich überlappende Fehlermengen, die von den Gutachtern entdeckt werden. In der Sitzung werden wenige weitere Fehler entdeckt (Laitenberger et al., 1999; Sauer et al., 2000).

Vorbereitungsintensität. Eine gründliche Vorbereitung ist für eine hohe Fehlerentdeckungsquote notwendig (Freedman und Weinberg, 1982; Weller, 1993; Fagan, 1986). Die Gründlichkeit der Vorbereitung wird im Folgenden als Vorbereitungsintensität bezeichnet. Sie hängt eng mit der Vorbereitungsrate, gemessen als Verhältnis von Umfang und Dauer, zusammen (Fagan, 1986), da die Gutachter ausreichend Zeit

benötigen. Laitenberger et al. (1999) zeigen den Zusammenhang zwischen Vorbereitungsintensität und Aufwand mit 340 Spezifikations-, Entwurfs- und Codereviews: Eine zu kurze Vorbereitung mindert die Fehlerentdeckungsquote, eine zu lange Vorbereitung erhöht hauptsächlich die Dauer; es gibt eine optimale, adäquate Vorbereitungsdauer (Biffl, 2001; Laitenberger et al., 1999; Raz und Yaung, 1997). Jalote (2000), Grady (1992) und Cusumano (1992) nennen Richtlinien für die Vorbereitungrate und den Durchsatz in der Sitzung, gemessen als Verhältnis zwischen Umfang und Dauer. Damit sich die Gutachter intensiv vorbereiten, benötigen sie Managementunterstützung, Schulungen und ausreichend Zeit; der Moderator spielt eine wichtige Rolle (Fagan, 1986).

Gutachterzahl. Freedman und Weinberg (1982) fordern mindestens zwei Gutachter, damit die Objektivität gewährleistet ist, und so viele Gutachter, dass alle wichtigen Merkmale geprüft werden können. Fagan (1976) schlägt Entwerfer, Programmierer und Tester als Gutachter vor. Porter und Votta (1997) zeigen Vorteile von zwei Gutachtern gegenüber einem, aber keine Verbesserung durch vier Gutachter. Bush (1990) nennt drei Gutachter als ausreichend für Code-Inspektionen. Weller (1993) berichtet, dass Inspektionen mit vier Gutachtern den Inspektionen mit drei Gutachtern überlegen sind. Sauer et al. (2000) und Laitenberger et al. (1999) sprechen von einem Sättigungseffekt. In Drappa (1998) sind Reviews mit zwei oder drei Gutachtern möglich, mit drei Gutachtern werden etwas mehr Fehler entdeckt.

Gutachterkompetenz. Freedman und Weinberg (1982) fordern technische Kompetenz. Basili und Selby (1987) zeigen, dass Praktiker den Akademikern in der Fehlerentdeckungsquote und bei der benötigten Dauer überlegen sind. Die Erfahrung spielt aber eine geringe Rolle (Basili und Selby, 1987; Maldenado et al., 2006; Biffl und Halling, 2002). Biffl und Halling (2002) zeigen, dass kompetente Gutachter durch Probereviews identifiziert werden können. Minimale Kenntnisse über Software-Entwicklung und über die Notation des Dokuments sind notwendig. Welche Eigenschaften zu einer hohen Kompetenz gehören, kann aber nicht gezeigt werden. Die vereinte Kompetenz der Gutachter prägt die Fehlerentdeckung (Weller, 1993; Sauer et al., 2000).

Kosten und Umfang. Die Dauer der Sitzung ist durch den Umfang des Prüflings bestimmt, sollte aber auf zwei Stunden beschränkt sein, weil sonst die Konzentration der Gutachter sinkt. Darum wird ein zu umfangreicher Prüfling auf mehrere Sitzungen aufgeteilt (Freedman und Weinberg, 1982; Fagan, 1976; Frühauf et al., 2006). Die Dauer der Vorbereitung ist durch den Umfang des Prüflings und die Intensität der Vorbereitung bestimmt (Fagan, 1986; Biffl, 2001). Ein zu groß gewählter Prüfling hängt mit niedriger Fehlerentdeckung zusammen (Raz und Yaung, 1997). Porter und Votta (1997) nennen organisatorische Gründe (Freedman und Weinberg, 1982) und eine hohe Zahl von Gutachtern für eine lange Dauer zwischen Reviewbeginn und Sitzung von bis zu vier Wochen.

Prüflingsüberdeckung. Fehler können nur in dem geprüften Teil eines Dokuments entdeckt werden. Reviews sollen sich auf die kritischen Entwicklungsresultate kon-

zentrieren (Frühauf et al., 2006), insbesondere in Situationen, in denen umfangreiche Dokumentation nicht vollständig begutachtet werden kann (Schwinn, 2003). Es erfolgt also eine Priorisierung der Dokumententeile; diejenigen, die als kritische angesehen werden, werden begutachtet, andere, weniger kritische Teile werden nicht begutachtet.

Hilfsmittel. Checklisten oder Szenarien können bei der Begutachtung helfen. Es gibt keine eindeutigen Untersuchungsergebnisse: Regnell et al. (2000) stellen fest, dass eine systematische Begutachtung mit diesen Hilfsmitteln den Reviews ohne Hilfsmittel überlegen ist. Maldenado et al. (2006) können keinen Unterschied zwischen Szenarien und Checklisten feststellen. Porter et al. (1995) zeigen aber, dass die Verwendung von Szenarien der Verwendung von Checklisten oder einem Vorgehen ohne Hilfsmittel überlegen ist. Einzelne Gutachter sind mit Checklisten umfassender vorbereitet als mit Szenarien, in der Sitzung gleicht sich dies wieder aus (Biffl und Halling, 2002; Biffl, 2001). Checklisten sind etwas günstiger als Szenarien (Biffl, 2001). Regnell et al. (2000) und Maldenado et al. (2006) können nicht zeigen, dass unterschiedliche Szenarien unterschiedliche Fehler entdecken.

Reviews von Änderungen. Korrekturen können im Rahmen eines Änderungsprozesses (Hörmann et al., 2006) und des Wartungsprozesses (ISO/IEC 14764, 1999; Pigoski, 1997) durch Reviews begutachtet werden. Beispielsweise soll geprüft werden, ob die Änderung korrekt ist und ob sie den Richtlinien entspricht. Dabei wird ein Ausschnitt des Codes betrachtet, nämlich die Änderung und der unmittelbar damit zusammenhängende Code.

5.5 Analyse von Tests

Testen ist das Ausführen eines Programms mit dem Ziel, Fehler zu entdecken (IEEE 610, 1990; Frühauf et al., 2006). Für den Test werden Testfälle definiert:

Def. **test case.** (1) A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement.
(2) (IEEE Std 829-1983 [5]) Documentation specifying inputs, predicted results, and a set of execution conditions for a test item. (IEEE 610, 1990)

Jeder Testfall legt den Anfangszustand, Testeingaben, Bedienungen, Sollresultate und den Endzustand fest. Durch den Vergleich zwischen Sollresultat und Istresultat soll Fehlverhalten erkannt werden.

Der Test erfolgt auf unterschiedlichen Integrationsebenen. In IEEE 610 (1990) werden Unit-, Modul-, Integrations-,Schnittstellen- und Systemtests unterschieden, der Prüfling kann eine einzelne syntaktische Einheit, eine Komponente oder das System sein. Ellims et al. (2006) definieren die Integrationsebenen konkret in ihrer Umgebung. Drappa (1998) unterscheidet im QS-Modell zwischen Modultest, Integrationstest und Systemtest. Auf allen Ebenen werden die gleichen Testtechniken eingesetzt (Lauterbach und Randall, 1989; Ellims et al., 2006; Liggesmeyer, 2002). Einzelne Komponen-

ten, die auch als Subsysteme bezeichnet werden, oder das gesamte Produkt können integriert und anschließend getestet werden (Chrissis et al., 2003; Jones, 2007). Der Systemtest kann unterschiedliche Schwerpunkte setzen, etwa auf Funktionen, Mengengerüst oder Last- und Stressverhalten (Pressman, 2005, Kap. 13; Liggesmeyer, 2002, S. 359).

In einem systematischen Test werden folgende Aktivitäten durchgeführt (Sneed et al., 2007; Spillner et al., 2006; Pressman, 2005, Kap. 13; Frühauf et al., 2006; IEEE 829, 1998; Sommerville, 2007, Kap. 22; ISO/IEC 12207, 1997; Automotive SIG, 2005; Jalote, 2000; van Megen und Meyerhoff, 1995; Chernak, 2001):

- Testplanung,
- Vorbereitung der logischen und konkreten Testfälle und Anordnung der Testfälle in Testsequenzen, die die Reihenfolge bestimmen, in der die Testfälle abgearbeitet werden (IEEE 829, 1998),
- Aufbau des Testgeschirrs, d.h. der Testumgebung mit Testtreibern, Testdaten und Platzhaltern (stubs), in der die Software im Test läuft (Frühauf et al., 2006; Sommerville, 2007, Kap. 23; Chernak, 2001),
- Testdurchführung und -protokollierung der Testfälle mit und ohne Abweichung zwischen Soll- und Istresultat,
- Testauswertung für Entscheidungen über das Testende und das weitere Vorgehen (Sneed et al., 2007; Spillner et al., 2006),
- Optionale Korrektur als eigenständige Aktivität, die auch die Ursachenanalyse für eine Abweichung, die Fehleranalyse, enthält (Pressman, 2005, Kap. 13; Sommerville, 2007, Kap. 22),
- Optionale Testwiederholung nach der Korrektur (Liggesmeyer, 2002; van Megen und Meyerhoff, 1995; Ebert et al., 2005; Automotive SIG, 2005), zwischen den Extremfällen der vollständigen Testwiederholung und der gezielten Wiederholung derjenigen Testfälle, die eine Abweichung zwischen Soll- und Istresultat zeigten (Thaller, 2002; Sneed et al., 2004; Müller et al., 1998),
- Abschluss und Archivierung (Spillner et al., 2006; Frühauf et al., 2006).

Diese Aktivitäten werden für Testprozesse vorgegeben (Jalote, 2000; Kan, 2003) und in der Praxis mehr oder weniger umfangreich, detailliert und systematisch durchgeführt und dokumentiert (Siegwart, 2004; Müller et al., 1998).

Testfalldefinition. In der Testvorbereitung werden Testfälle definiert, d.h. Eingaben ausgewählt und um Sollresultate ergänzt. Die Testfallauswahl (oder Definition) bestimmt die Fehlerentdeckung und dadurch den Testerfolg (Endres und Rombach, 2003, S. 126; Frühauf et al., 2006). In der Praxis wird für jeden Test im Mittel eine typische Zahl von Testfällen definiert (Jones, 2007). Jeder Testfall prüft einen Punkt im praktisch unendlich großen Eingaberaum der Software (Endres und Rombach, 2003; Dahl et al., 1972). Da der Eingaberaum so groß ist, ist ein vollständiger Test praktisch

nicht möglich. Anstatt also den Eingaberaum vollständig zu testen, wird der Eingaberaum in Bereiche geteilt und jeder Bereich durch einen Testfall geprüft. Der Testfall soll zu allen anderen Testfällen des Bereichs äquivalent bezüglich der Fehlerentdeckung sein (Goodenough und Gerhart, 1977; Weyuker und Ostrand, 1980). Der Bereich wird als Äquivalenzklasse bezeichnet. Ein Fehler definiert eine starke Äquivalenzklasse (Ludewig und Lichter, 2007; Weyuker und Ostrand, 1980) durch die Eingaben, die den Fehler wirksam werden lassen. Weil der Fehler erst durch den Test entdeckt wird, werden mit Testtechniken vermutete Äquivalenzklassen abgeleitet. Diese vermuteten Äquivalenzklassen bezeichnen Ludewig und Lichter (2007) als schwache Äquivalenzklassen.

Testfalldefinition im Black-Box-Test. Testfälle werden aus der Vorgabe für den Prüfling abgeleitet, d.h. im Systemtest aus der Spezifikation, im Integrationstest aus dem (Grob-)Entwurf, im Modultest aus dem (Fein-)Entwurf. Dazu werden unterschiedliche Elemente verwendet, um den Eingabebereich aufzuteilen, etwa einzelne Funktionen, Entscheidungen, Zustände, Eingabe- und Ausgabebereiche (Liggesmeyer, 2002). Es werden also verschiedene Testtechniken verwendet, um Äquivalenzklassen abzuleiten. Der Test der Grenzen dieser Eingabebereiche kann als eigene Testtechnik (Pressmann, 2005) oder als Teil der anderen Testtechniken (Liggesmeyer, 2002) aufgefasst werden. In der Praxis sind diese Black-Box-Testtechniken weit verbreitet:

- Eine wichtige Testtechnik, die als minimale Forderung für den Test verwendet wird, ist die Funktionsabdeckung (Frühauf et al., 2006). Sie wird als erstes intuitiv angestrebt (Cornelissen et al., 1995). Diese Testtechnik ist in der Praxis am Weitesten verbreitet (Müller et al., 1998).
- Grenzwerte werden häufig geprüft (Müller et al., 1998).
- Der Test wird ergänzt um weitere Äquivalenzklassen (Spillner und Linz, 2003; Liggesmeyer, 2002; Müller et al., 1998), Zufallstest und Ursache-Wirkungsanalysen (Müller et al., 1998).

Testfalldefinition im Glass-Box-Test. Für den Glass-Box-Test wird gemessen, wie viel Quellcode durch bereits ausgeführte Testfälle überdeckt wird. Dann werden neue Testfälle aus der Spezifikation und der bislang erreichten Überdeckung abgeleitet und ausgeführt, so lange, bis ein Überdeckungskriterium erreicht ist (Liggesmeyer, 2002). Unterschieden werden die Techniken anhand der Definition der Überdeckung. Dabei werden kontrollflussorientierte von datenflussorientierten Techniken unterschieden. Die kontrollflussorientierten Techniken der Anweisungs-, Zweig-, Term- und Schleifenüberdeckung sind praxisrelevant (Liggesmeyer, 2002):

- Bei der Anweisungsüberdeckung wird gemessen, wie viele Anweisungen ausgeführt wurden.
- Bei der Zweigüberdeckung wird gemessen, wie viele Zweige im Kontrollfluss durchlaufen wurden.

- Bei der Bedingungsüberdeckung werden die einzelnen Terme einer Bedingung betrachtet. Liggesmeyer (2002) nennt mehrere Varianten. Insbesondere die Termüberdeckung (MC/DC) ist relevant, weil sie für Software in der Luftfahrt gefordert wird (RTCA, 1992). Sie prüft, ob jeder einzelne Term in einem logischen Ausdruck durch den Test das Resultat bestimmt.
- Bei der Schleifenüberdeckung wird gemessen, ob eine Schleife nicht, einmal und mehrmals durchlaufen wird. Auch dafür gibt es Varianten, etwa wie mit geschachtelten Schleifen umgegangen wird (Liggesmeyer, 2002).

Die kontrollflussorientierten Testtechniken sind nicht unabhängig voneinander:

- Die Zweigüberdeckung impliziert Anweisungsüberdeckung, außer bei totem Code.
- Die Termüberdeckung impliziert die Zweigüberdeckung (Liggesmeyer, 2002; Ludewig und Lichter, 2007).
- Die Schleifenüberdeckung liegt dagegen quer zu den anderen Testtechniken: Durch Schleifenüberdeckung wird ein Teil der Zweige überdeckt, nämlich die Zweige, die durch Schleifen entstehen. Mit Zweigüberdeckung wird nur teilweise Schleifenüberdeckung erreicht, weil für die Zweigüberdeckung zwar eine Schleife mindestens einmal durchlaufen werden muss, dann fehlt aber entweder der mehrmalige Durchlauf oder der einmalige Durchlauf.

Datenflussorientierte Techniken werden nicht durch Werkzeuge unterstützt, diversifizierende Tests spielen nur bei redundant ausgelegter Software eine Rolle. Der kontrollflussorientierte Glass-Box-Test wird durch Standards gefordert (RTCA, 1992; Smith und Simpson, 2005). Müller et al. (1998) zeigen, dass selbst die Anweisungsüberdeckung nur selten eingesetzt werden.

Zusammenhang Black-Box- und Glass-Box-Test. Der Glass-Box-Test wird ergänzend zum Black-Box-Test durchgeführt, erfolgt also, nachdem der Black-Box-Test durchgeführt wurde (Lauterbach und Randall, 1989, zitiert in Grady, 1992; RTCA, 1992). Eine Kombination der Black-Box- und Glass-Box-Techniken wird benötigt (Juristo et al., 2002 und 2004), um möglichst viele, auch kritische Fehler zu entdecken (Dupuy und Leveson, 2000). Weyuker und Ostrand (1980) begründen dies mit der Theorie, dass die maximale Zahl an Äquivalenzklassen durch die Zahl der möglichen Pfade durch das Programm definiert ist. Ein systematischer Black-Box-Test erzielt etwa 50% Anweisungsüberdeckung. Für reale Produkte sind 80% Anweisungsüberdeckung ein realistisches Testkriterium (Grady, 1992; Piwowarski et al., 1993). Die Überdeckung zu steigern wird aufwändiger, je mehr Einheiten bereits überdeckt sind. Kann Code nicht erreicht werden, dann ist eine vollständige Überdeckung nicht möglich. Malaiya et al. (1994) verwenden anstatt der Anweisungsüberdeckung die Blocküberdeckung, d.h. wie viele Blöcke des Programmcodes durchlaufen wurden. Sie formulieren und prüfen die Annahme, dass Block- und Zweigüberdeckung in einem bestimmten Bereich linear zusammenhängen. In diesem Bereich werden alle Blöcke nahezu überdeckt, die Zweigüberdeckung wächst mit der Blocküberdeckung propor-

tional, bleibt insgesamt aber niedriger. In der Untersuchung wurden dann keine weiteren Testfälle ausgeführt, so dass keine Aussage darüber möglich ist, wie die Zweigüberdeckung wächst, wenn bei vollständiger Blocküberdeckung weitere Testfälle durchgeführt werden.

Techniken als Heuristiken. Mit den Testtechniken für Black-Box- und Glass-Box-Test werden keine konkreten Werte für die Testeingaben abgeleitet. Für diese konkreten Werte ist der Tester auf seine Intuition angewiesen (Liggesmeyer, 2002). Selbst im Glass-Box-Test, bei dem der Code sichtbar ist, ist es nicht möglich, konkrete Testeingaben auszurechnen: Um beispielsweise eine bestimmte Codezeile auszuführen, können ganz unterschiedliche Pfade durch das Programm durchlaufen und damit ganz unterschiedliche Eingaben gewählt werden (Beizer, 1990); Bedingungen geben einen Bereich, aber keinen einzelnen Wert vor: "*Kontrollflussorientierte Testtechniken definieren, wie alle strukturorientierten Testtechniken, keine Regeln für die Erzeugung von Testfällen*" (aus Liggesmeyer, 2002).

Fehlerentdeckung. Auf jeder Testebene werden systematisch Fehler einer bestimmten Abstraktionsebene entdeckt (Drappa, 1998). Zusätzlich können Fehler der darunter liegenden Abstraktionsebene entdeckt werden (Drappa, 1998; Jones, 1996). Da der Systemtest gegen die Spezifikation prüft, werden darum systematisch (Grob-)Entwurfsfehler entdeckt. Der Systemtest lässt aber auch Codefehler wirksam werden. Außerdem werden Fehler unsystematisch in der Vorgabe, d.h. auf der nächsthöheren Abstraktionsebene, entdeckt (Drappa, 1998; Jones, 1996). Somit werden Spezifikationsfehler im Systemtest unsystematisch bei der Definition von Testfällen im Zuge der Verwendung der Spezifikation, etwa um Sollresultate zu definieren, gefunden. Ähnliches gilt für den Integrationstest, der den Grobentwurf als Vorgabe hat, und den Modultest, der den Feinentwurf als Vorgabe hat (Drappa, 1998). Im Glass-Box-Test werden tendenziell andere Fehlerarten als im Black-Box-Test entdeckt. Der Glass-Box-Test entdeckt keine nicht-implementierten Anforderungen (Basili und Selby, 1987; Kamsties und Lott, 1995).

Empirische Untersuchungen dazu ergeben aber widersprüchliche Ergebnisse und basieren auf wenigen Fallstudien (Basili und Selby, 1987; Kamsties und Lott, 1995; Lauterbach und Randall, 1989). Die Entdeckung neuer Fehler nimmt mit der Zeit (Kan, 2003; Cornelissen et al., 1995) und mit steigender Zweigüberdeckung (Malaiya et al., 1994) ab. Die Fehlerentdeckung unterliegt starken Schwankungen (Hutchins et al, 1994; Wong et al., 1994), begründet durch Querbeziehungen im Code (Chaar et al., 1993), unterschiedliche Arten der entstandenen Fehler (Juristo et al., 2002 und 2004) und die konkrete Definition der Testfälle, die nicht durch die Testtechniken abgeleitet werden kann (Liggesmeyer, 2002).

Vorbereitungszeitpunkt. Der Black-Box-Test kann vorbereitet werden, bevor der Prüfling erstellt wird. Beim Systemtest kann die Vorbereitung also nach Spezifikation und Architekturentwurf erfolgen (Jalote, 2000; van Megen und Meyerhoff, 1995). Beim Test einzelner Einheiten wird dieses Vorgehen als testgetriebene Entwicklung bezeichnet (Beck, 2003). Bei der Vorbereitung wird die Vorgabe verwendet, so dass

dabei – unsystematisch – Fehler in der Vorgabe entdeckt werden (Drappa, 1998). Beim Systemtest werden somit Fehler in der Spezifikation entdeckt. Erfolgt die Vorbereitung früh, dann können diese Fehler früh korrigiert werden. Diese frühe Korrektur ist günstiger als die Korrektur in der entsprechenden Testphase, weil noch nicht auf den Spezifikationsfehler aufgebaut wurde (Abschnitt 5.3). Kann ein Spezifikationsfehler erst in der Systemtestphase korrigiert werden (späte Vorbereitung und späte Korrektur), dann wurde der Fehler aus der Spezifikation in die Folgedokumente und in den Code übertragen und ist darum aufwändiger zu korrigieren (Abschnitt 5.3).

Testerkompetenz. Die Kompetenz, Intuition und Erfahrung des Testers oder der Tester spielt eine große Rolle (Spillner und Linz, 2003; Liggesmeyer, 2002; Basili und Selby, 1987; Lauterbach und Randall, 1989; Müller et al, 1998), weil die Testtechniken auf Heuristiken beruhen.

Umfang und Aufwand. Der Produktumfang bestimmt den Testaufwand (Jones, 1996; Boehm, 2000). Der Eingaberaum wächst überproportional mit der Zahl der Eingabeparameter (Endres und Rombach, 2003; Dahl et al., 1972). Darum kann vermutet werden, dass die Zahl der möglichen Testfälle überproportional mit dem Umfang wächst. Erfahrungswerte aus der Praxis zeigen aber einen linearen Zusammenhang zwischen Umfang und Testfallzahl (Jones, 2007, S. 506). Testfallzahl und Überdeckung verhalten sich nichtlinear (Liggesmeyer, 2002, S.86), weil ein Sättigungseffekt eintritt: Die ersten Testfälle steigern die Überdeckung stärker, spätere Testfälle steigern die Überdeckung weniger stark.

Folgerungen

Für die Modellierung der Tests folgere ich: Tests unterschiedlicher Integrationsebene unterscheiden sich quantitativ, z. B. in der Fehlerentdeckung, aber nicht in den Testtechniken und qualitativen Zusammenhängen. Testfälle spielen die zentrale Rolle im Test. Da der Eingaberaum von Programmen praktisch unendlich groß ist, können praktisch nahezu unendlich viele Testfälle definiert werden. Mit welcher Testtechnik ein Testfall abgeleitet wurde, lässt sich am Testfall nicht erkennen. Die starken Äquivalenzklassen sind unbekannt, weil die Fehler nicht bekannt sind. Darum handelt es sich bei den Testtechniken um Heuristiken. Somit kann die Zahl der Testfälle, die mit den Testtechniken abgeleitet werden, d.h. die Zahl der schwachen Äquivalenzklassen, nicht berechnet, nur geschätzt werden. Daraus folgt auch: Ob ein Testfall einen Fehler entdeckt, erscheint bei der ersten Ausführung des Testfalls als Zufall, weil die starken Äquivalenzklassen nicht bekannt sind. So ist unklar, ob es sich um einen bereits getesteten oder noch nicht getesteten Pfad handelt (also eine mögliche Äquivalenzklasse), und selbst wenn es sich um einen neuen Pfad handelt, ist unklar, ob er Teil einer bestimmten starken Äquivalenzklasse ist.

5.6 Analyse automatischer statischer Codeanalyse

Mit der automatischen statischen Codeanalyse, im Folgenden kurz Codeanalyse genannt, werden verdächtige Konstrukte im Code von einem Werkzeug identifiziert und dokumentiert (Spinellis, 2006; Louridas, 2006). Es gibt eine Reihe von Werkzeugen für unterschiedliche Programmiersprachen. Ein frühes, nach wie vor eingesetztes Werkzeug für C-Code ist Lint (Johnson, 1978). Die Werkzeuge verwenden unter anderem Syntax-, Kontrollfluss- und Datenflussanalysen. Damit sind sie fähig, bestimmte Fehlerarten zu entdecken, können aber bestimmte Fehlerarten prinzipiell nicht zeigen (Zheng et al., 2006). Die von den Werkzeugen entdeckten Fehler können ohne diese Werkzeuge in Folgeprüfungen entdeckt werden oder im Einsatz auftreten. Welche Fehlerarten entdeckt werden können, hängt von den verwendeten Analyseverfahren ab. Die meisten Werkzeuge zeigen auch falsche Befunde an, weil die Analysen auf möglichen, nur vermuteten Fehlern beruhen (Zheng et al., 2006). Das Werkzeug BEAM (Brand, 2000; Brand und Krohm, 2003; Brand et al., 2007) verfolgt dagegen den Ansatz, nur tatsächliche Fehler anzuzeigen.

Damit die Prüfung durchgeführt werden kann, muss das Werkzeug installiert und eingerichtet werden. Dafür fällt Vorbereitungsaufwand für den Entwickler an. Für die Prüfungsdurchführung ist der Aufwand minimal, weil das Werkzeug nur gestartet werden muss. Es kann auch in den Build-Prozess eingebunden werden (Spinellis, 2006). Die Analyse der Resultate kostet Aufwand, um falsche Befunde und Fehler zu unterscheiden. Für Fehler fällt Korrekturaufwand an.

Kapitel 6

Ein quantitatives Modell für Prüfungen: CoBe

In diesem Kapitel wird das Modell CoBe beschrieben. Abschnitt 6.1 zeigt einen Modellüberblick. Das Modell besteht aus einem Basismodell und Prüfungsmodellen. In Abschnitt 6.2 wird die Architektur von CoBe mit einem Beispiel erläutert. Das Basismodell und seine Zusammenhänge werden in Abschnitt 6.3 dargestellt. Modelle für Reviews werden in Abschnitt 6.4, für die Codeanalyse in Abschnitt 6.5 und für Tests in Abschnitt 6.6 beschrieben. In Abschnitt 6.7 werden die Modellteile gemeinsam dargestellt. Abschnitt 6.8 enthält die Quantifizierung.

6.1 Überblick über das Modell CoBe

Abbildung 15 zeigt CoBe im Überblick. Die Eingaben des Modells gliedern sich in Eingaben für den Prüfprozess des Projekts, für Prüfparameter einzelner Prüfungen und für Prozess- und Produktmerkmale, zu denen auch die Kalibrierung gehört. CoBe stellt Nutzen durch entfallende Kosten dar und gibt Kosten und Nutzen der Prüfungen pro Aktivität im Projekt aus. Die Kosten werden zusammengefasst, zuerst alle Kosten im Projekt und alle Kosten nach Auslieferung, dann die Kosten insgesamt über die gesamte Lebensdauer des Produkts. Nutzen wird auf gleiche Weise zusammengefasst. Die Zusammenhänge zwischen Eingaben und Ausgaben sind durch Gleichungen beschrieben, die durch interne Parameter quantifiziert sind.

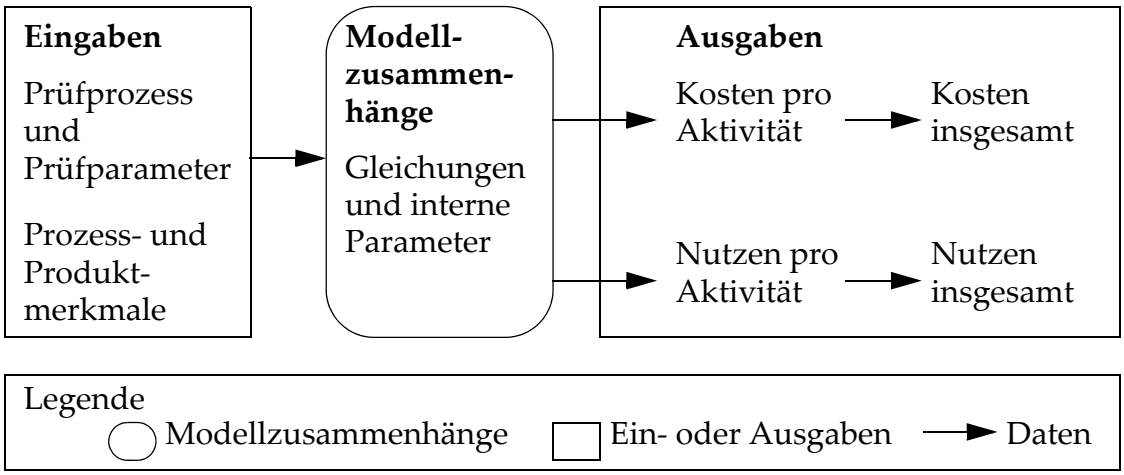

Abb. 15: Überblick über das Modell

CoBe gliedert sich in ein Basismodell und einzelne Prüfungsmodelle. Abbildung 16 skizziert diesen Aufbau. Das Basismodell enthält die grundlegenden Zusammenhänge der Qualitätskosten. Jede Prüfungsart wird durch ein Modell dargestellt. Die Abbildung zeigt beispielhaft Spezifikationsreview, Entwurfsreview und Systemtest. Die einzelnen Prüfungsmodelle sind über Parameter mit dem Basismodell verbunden. Die Struktur erlaubt, das Modell um Prüfungen zu erweitern oder Modelle einzelner Prüfungen zu ändern, z.B. um unterschiedliche Entscheidungen zu unterstützen. Die Modellkomponenten können einzeln geprüft werden, weil Werte direkt an den Schnittstellen sichtbar sind. Diese Struktur erlaubt auch, das Modell an ein anderes Vorgehen im Projekt anzupassen, beispielsweise weil sich die Sequenz der Prüfungen ändert. Diese Änderung ist möglich, ohne die Prüfungsmodelle zu ändern.

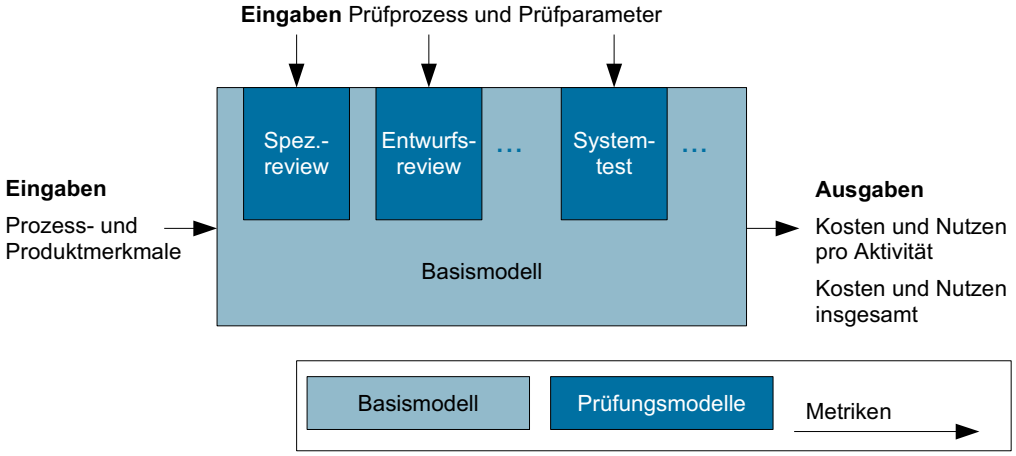

Abb. 16: Überblick über den Modellaufbau

6.2 Die Architektur von CoBe

Das Basismodell und die Prüfungsmodelle bestehen jeweils aus einzelnen Komponenten (Abschnitt 6.2.1). Ein Beispiel zeigt im Folgenden, wie die Modellresultate durch die Komponenten berechnet werden (Abschnitt 6.2.2). Die Parameter zur Kalibrierung sind in Abschnitt 6.2.3 beschrieben.

6.2.1 Die Modellkomponenten von CoBe

Abbildung 17 zeigt die Komponenten von CoBe. Die Eingaben sind oben in der Abbildung, die Ausgaben im unteren Teil. Die Abbildung zeigt, durch welche Komponenten die Ausgaben berechnet werden. Die Komponenten des Basismodells sind in der Abbildung hell dargestellt, die Prüfungsmodelle dunkel. Die Prüfungsmodelle gliedern sich für jede Prüfung in ein Modell der Fehlerentdeckung und ein Modell der Prüfkosten. Kosten für die Prüfwiederholung werden getrennt davon berechnet.

Die Eingaben werden hier skizziert, sie werden mit den Modellkomponenten detailliert beschrieben (Abschnitte 6.3 bis 6.6). Die Komponenten sind im Einzelnen:

- Das Umfangsmodell gehört zum Basismodell. Es berechnet den Umfang verschiedener Artefakte der Software-Entwicklung aus Eingaben für den Umfang neuer und wiederverwendeter Software.
- Die Prüfungsmodelle der Fehlerentdeckung berechnen die Fehlerentdeckungsquoten (Abschnitt 5.2) für unterschiedliche Fehlerarten und Fehlerschwere. Die Eingaben beschreiben den Prüfprozess, also ob eine Prüfung stattfindet, ob und wie die Prüfung wiederholt wird und ob wiederverwendete Software geprüft wird. Eingaben für Prüfparameter sind z.B. die Zahl der Gutachter in Reviews oder Testtechniken und ihre Vollständigkeit.
- Das Fehlerstrommodell beschreibt die Fehlerentstehung und Fehlerentdeckung. Es stellt dar, welche Prüfungen in welcher Reihenfolge stattfinden können, und beschreibt dadurch, welche Prüfsequenzen möglich sind. Für die Fehlerentstehung werden Verteilungen auf Fehlerarten und Fehlerschwere und ein Kalibrierungsparameter für die Fehlerzahl eingegeben. Für die Fehlerentdeckung verwendet das Fehlerstrommodell die Fehlerentdeckungsquoten der Prüfungsmodelle. Das Ergebnis sind Zahlen für entdeckte, korrigierte und entfallende Fehler für jede Prüfung.
- Die Modelle für die Prüfkosten berechnen für jede Prüfung den anfallenden Aufwand. Zusätzlich werden in den Reviewmodellen Personalbedarf und Dauer berechnet, weil sich diese direkt aus den Prüfparametern des Reviews ergeben. In den Testmodellen geben die Prüfparameter keinen bestimmten Personalbedarf oder eine bestimmte Dauer vor, so dass nur der Aufwand berechnet wird.
- Die Modelle der Prüfwiederholungskosten ergeben den Aufwand, der zur Prüfwiederholung nach Korrekturen im Projekt und in der Wartung benötigt wird, abhängig von den Eingaben des Prüfprozesses. Der Aufwand wird als anfallende Kosten oder entfallende Kosten (Nutzen) dargestellt.
- Das Modell für den Korrekturaufwand berechnet aus den Fehlerzahlen, wie viel Aufwand für die entdeckten Fehler anfällt (Kosten) und später entfällt (Nutzen).
- Das Aufwandseinflussmodell beschreibt, wie sich Prozess und Produkt auf den Aufwand auswirken. Es verwendet die Parameter von COCOMO II, einen Zuschlag für die Organisation der einzelnen Aktivitäten und einen Kalibrierungsparameter für den Aufwand.
- Das Dauer- und Personalmodell beschreibt die Zusammenhänge zwischen Aufwand, Dauer und Personalbedarf, um aus dem Aufwand die Dauer und den Personalbedarf abzuleiten. Dazu werden Zusammenhänge und Parameter aus COCOMO II und zusätzlich ein Kalibrierungsparameter für die Dauer verwendet.
- Das Fehlerfolgekostenmodell erlaubt durch Klassifikation und Gewichtung von Fehlern, die Fehlerfolgekosten abzuschätzen.

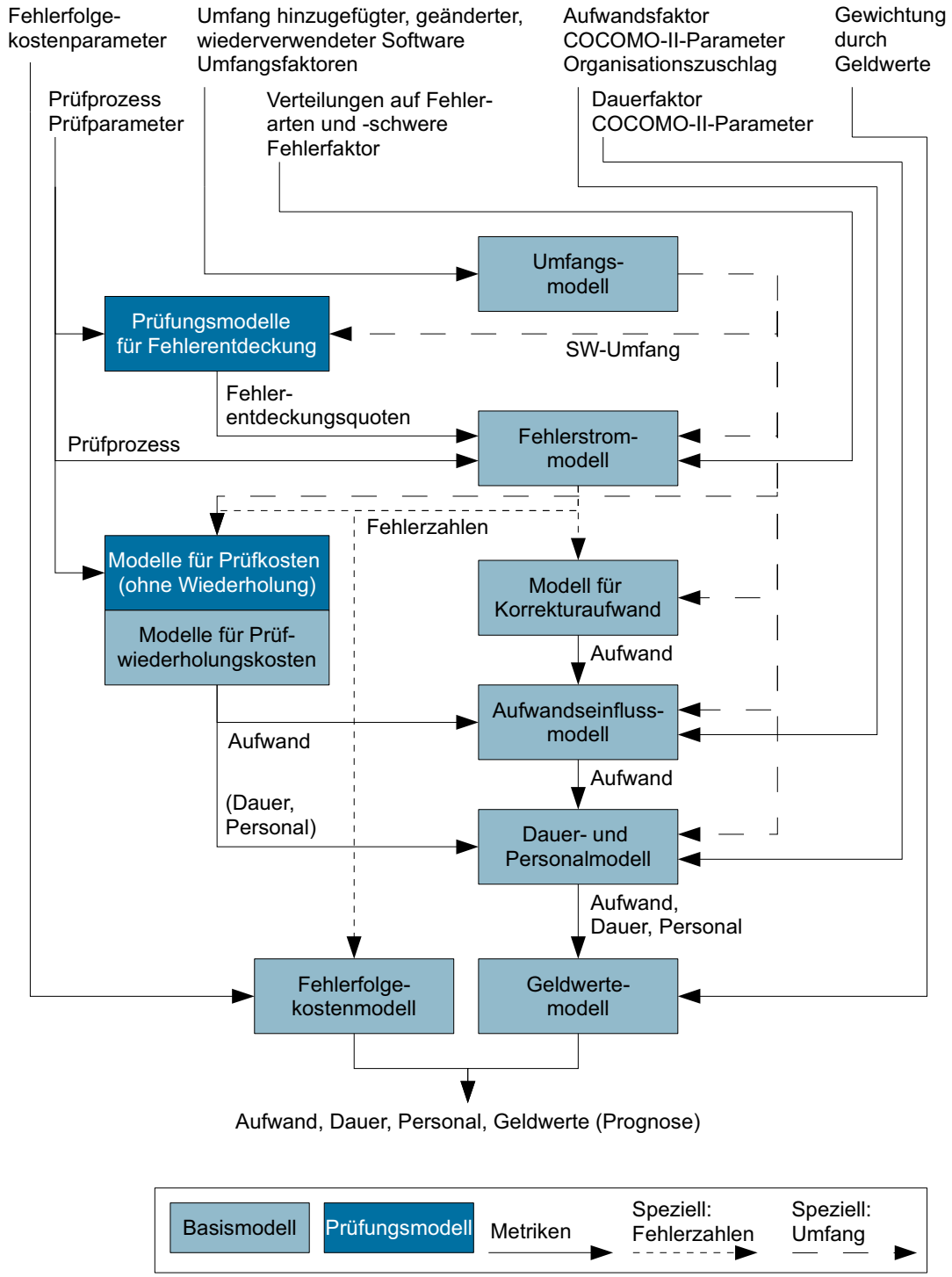

Abb. 17: Struktur von CoBe

- Das Geldwertemodell ermöglicht die Gewichtung von Aufwand, Dauer und Personalbedarf durch Geldwerte als gemeinsame Skala für den Vergleich.

Die Metriken sind aktivitätsbezogen, weil Aufwand, Dauer, Personalbedarf und Geldwerte pro Aktivität dargestellt werden. Diese Werte werden dann zusammengefasst. Resultate für die Kosten einer Prüfung sind anfallender Aufwand, Dauer, Personalbedarf und Geldwerte pro Aktivität. Resultate für den Nutzen einer Prüfung sind entfallender Aufwand, gesparte Dauer, dadurch nicht benötigtes Personal und Geld pro Aktivität. Tabelle 5 zeigt die anfallenden Kosten für die Prüfung (links) und die entfallenden Kosten, der Nutzen, der durch die Prüfung erreicht wird (rechts). Der Nutzen ist in der Tabelle durch ein ' gekennzeichnet. In CoBe werden unterschieden:

Def. **Prüfkosten.** Kosten, die für Prüfungen ohne Prüfwiederholung anfallen.

Def. **Fehlerbehebungskosten.** Kosten für Korrektur mit Fehleranalyse und Änderung der Software und Kosten für die Prüfung der Korrektur (Prüfwiederholung) im Projekt oder in der Wartung.

Def. **Fehlerfolgekosten.** Kosten, die beim Einsatz des Produkts durch Fehler verursacht werden. Kosten für die Fehlerbehebung zählen nicht dazu.

Def. **Fehlerkosten**. Fehlerbehebungs- und Fehlerfolgekosten.

Def. **Projekt-Qualitätskosten.** Summe der im Projekt anfallenden Prüf- und Fehlerbehebungskosten.

Def. **Gesamt-Qualitätskosten.** Summe der anfallenden Prüf-, Fehlerbehebungs- und Fehlerfolgekosten im Projekt, im Produkteinsatz und in der Wartung.

Kosten für Prüfung	Entfallende Kosten (Nutzen) durch Prüfung.
• Prüfkosten • Fehlerbehebungskosten	• Prüfkosten' • Fehlerbehebungskosten' • Fehlerfolgekosten'

Tabelle 5: Kosten und Nutzen durch anfallende und entfallende Kosten

6.2.2 Ein Beispiel zur Illustration von CoBe

Die Berechnung der Resultate durch die Modellkomponenten zeigt das folgende Beispiel für Kosten und Nutzen des Spezifikationsreviews. Die Modellresultate werden nicht vollständig, sondern ausschnitthaft gezeigt. Insbesondere wird der Nutzen nur für den Systemtest, nicht für andere Prüfungen betrachtet.

Prüfprozess. Ein Prüfprozess mit Spezifikations- und Entwurfsreview, Modul-, Systemintegrations-, System- und Feldtest wird in CoBe eingegeben.

Umfangsmodell. Für eine Neuentwicklung werden für den Software-Umfang 200 Function Points neue Software eingegeben. Daraus berechnen sich etwa 11 000 Anweisungen Java-Code, 88 Seiten Spezifikation, 1 Jahr Projektdauer und 3 Mitarbeiter.

Fehlerstrommodell (Fehlerentstehung). Insgesamt entstehen rund 662 Fehler, davon sind 159 Spezifikationsfehler.

Prüfungsmodell für Fehlerentdeckung (Spezifikationsreview). Das vollständige Spezifikationsreview wird durch 5 kompetente Gutachter durchgeführt, die sich gründlich vorbereiten. Das Prüfungsmodell des Spezifikationsreviews berechnet aus diesen Eingaben eine Fehlerentdeckungsquote von etwa 60%.

Fehlerstrommodell (entdeckte Fehler). Das Fehlerstrommodell berechnet, dass 96 Spezifikationsfehler entdeckt werden.

Fehlerstrommodell (entfallende Fehler). Mit dem Fehlerstrommodell wird berechnet, wie viele dieser 96 Spezifikationsfehler in späteren Prüfungen und nach Auslieferung entfallen. Beispielsweise entfallen durch das Spezifikationsreview rund 7 Fehler im Systemtest, die ohne Spezifikationsreview nach dem Systemtest korrigiert werden müssten. Nach Auslieferung entfallen 52 Fehler.

Modell für Prüfkosten (Spezifikationsreview). Das Modell für die Prüfkosten des Spezifikationsreviews ergibt 7 Mitarbeiter (Gutachter, Autor, Moderator). Es werden insgesamt 75 Entwicklerstunden Aufwand investiert. Die drei Sitzungen finden verteilt auf 9 Arbeitstage statt.

Modell für Korrekturaufwand und Aufwandseinfluss. CoBe berechnet, dass die Korrektur nach dem Spezifikationsreview 123 Entwicklerstunden kostet.

Modelle für Korrekturaufwand (entfallend) und für Aufwandseinfluss. CoBe berechnet, dass 95 Entwicklerstunden Korrektur im Systemtest entfallen, weil dabei 7 Fehler entfallen. In der Wartung werden für die 52 Fehler, die durch das Spezifikationsreview entfallen, rund 720 Entwicklerstunden für die Korrektur eingespart.

Modelle für Prüfungswiederholung (entfallend) und für Aufwandseinfluss. Testwiederholung wird im Systemtest nicht eingespart, weil der Test im Beispiel vollständig wiederholt wird. In der Wartung entfallen 1150 Entwicklerstunden für die Prüfung der Korrektur durch Modul- und Systemtest.

Dauer- und Personalmodell. Aus den Aufwänden wird die Dauer und der Personalbedarf berechnet. Die Korrektur nach dem Spezifikationsreview beispielsweise dauert 25 Arbeitstage. Nach dem Systemtest sind mit der Korrektur rund 2 Mitarbeiter beschäftigt, so dass etwa 6 Arbeitstage durch das Spezifikationsreview entfallen.

Geldwertemodell. Der Aufwand wird durch Personalkosten mit 100 Euro pro Entwicklerstunde gewichtet. Damit ergeben sich für Spezifikationsreview und Korrektur etwa 20 000 Euro. Dafür entfallen beispielsweise in der Korrektur nach dem Systemtest 9 500 Euro und nach Auslieferung 186 000 Euro für Wartung.

Fehlerfolgekostenmodell. Die 52 entfallenden Fehler nach Auslieferung würden unter den gegebenen Prozess- und Produktmerkmalen einen Schaden von insgesamt rund 66 000 Euro verursachen. Diese Kosten entfallen.

Zuammenfassung. CoBe berechnet Kosten und Nutzen von Qualitätssicherungsmaßnahmen: Der Nutzen des Spezifikationsreviews besteht im Beispiel aus entfallenden Aufwänden für Korrektur, dazu gehören 95 Entwicklerstunden nach dem Systemtest und 720 Entwicklerstunden in der Wartung, entfallenden Aufwänden für Testwiederholung (1150 Entwicklerstunden in der Wartung), die dafür notwendige Dauer (6 Arbeitstage nach dem Systemtest) und entfallenden Fehlerfolgekosten. CoBe summiert den Nutzen im Projekt auf entfallende Personalkosten von 34 000 Euro[1]. Für die Wartung wird berechnet, dass 186 000 Euro Personalkosten und 66 000 Euro Folgekosten entfallen. Im Beispiel werden die folgenden Kosten berechnet: Die Kosten des Spezifikationsreviews fallen für die Prüfung (76 Entwicklerstunden, 9 Arbeitstage) und die Korrektur (123 Entwicklerstunden, 25 Arbeitstage) an. Daraus ergeben sich 20 000 Euro für das Spezifikationsreview. CoBe berechnet im Beispiel Qualitätskosten, die im Projekt anfallen (Projekt-Qualitätskosten) in Höhe von 239 000 Euro und Qualitätskosten einschließlich Wartung und Einsatz (Gesamt-Qualitätskosten) in Höhe von 926 000 Euro.

6.2.3 Die Kalibrierungsparameter von CoBe

Die Parameter von CoBe sind mit Mittelwerten aus der Industrie quantifiziert. Erfahrungen mit Kostenschätzmodellen zeigen, dass eine solche Quantifizierung an spezifische Projekte angepasst werden muss. Dies wird als Kalibrierung bezeichnet (Abschnitt 3.6.1). Diese Kalibrierung wird in CoBe durch spezielle Kalibrierungsparameter unterstützt. Diese Parameter ergeben sich aus der Analyse in Kapitel 5. In Abbildung 17 sind oben die Eingaben des Modells dargestellt. Zu den Kalibrierungsparametern gehören die folgenden Eingaben:

- CoBe bietet einen Aufwands- und einen Dauerfaktor, weil Erfahrungen mit COCOMO zeigen, dass Aufwand und Dauer an die Umgebung angepasst werden müssen.
- Der Fehlerfaktor zur Kalibrierung der Gesamtfehlerzahl wird benötigt, weil viele Einflüsse auf die Fehlerentstehung unbekannt, zumindest quantitativ unklar sind. Als Gesamtfehlerzahl wird die Zahl der insgesamt entdeckten Fehler bezeichnet.
- Die Fehler verteilen sich abhängig vom Produkt und von der Projektart unterschiedlich auf die Fehlerarten und auf die Fehlerschwere. Insbesondere die Fehlerschwere kann unterschiedlich definiert sein. Darum kann die Verteilung auf die Fehlerarten und die Verteilung auf die Fehlerschwere als Parameter eingegeben werden.

1. Dazu gehören neben den entfallenden Kosten im Systemtest auch entfallende Kosten anderer Prüfungen, die hier im Beispiel nicht gezeigt sind.

- CoBe bietet einen Umfangsfaktor für den Code, für die Spezifikation und für den Entwurf, um den Umfang dieser Artefakte zu berechnen. Die Faktoren sind mit Standardwerten belegt und können aus Daten abgeschlossener Projekte berechnet werden (Abschnitt 6.7.2).

6.3 Das Basismodell mit den grundlegenden Zusammenhängen

Im Folgenden wird jede Modellkomponente mit ihren Eingaben und ihren Resultaten durch Gleichungen und durch Ursache-Wirkungs-Diagramme beschrieben. Die Bezeichner, die in den Gleichungen verwendet werden, sind zusätzlich im Anhang ab Seite 281 aufgeführt. Die Ursache-Wirkungs-Diagramme stellen dar, welche Ursachen oder Gruppen von Ursachen sich auf eine bestimmte Größe auswirken (Kan, 2003).

Abbildung 18 zeigt links den grundsätzlichen Aufbau der Diagramme. Die Pfeile stellen Wirkungen in Richtung der Pfeilspitze dar. Die Ursachen werden gruppiert und in Form von Fischgräten um einen horizontalen Pfeil herum angeordnet. Die zusammengefasste Wirkung wird rechts an diesem Pfeil dargestellt. Die Gruppierung ist vorgegeben, wenn die Diagramme zur Ursachenanalyse eingesetzt werden.

Für das quantitative Modell sind die Parameter die Ursachen, aus denen das Modellresultat berechnet wird (Rechts in Abbildung 18). In den Diagrammen für CoBe folgt die Gruppierung keinem festen Schema, sondern wird eingesetzt, um die Parameter zu gliedern.

Im Folgenden werden zuerst Ausschnitte des Modells gezeigt, um die einzelnen Zusammenhänge zu erläutern. Abschnitt 6.7 zeigt dann die Komponenten gemeinsam.

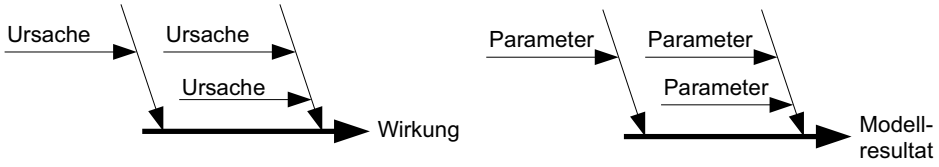

Abb. 18: Ursache-Wirkungs-Diagramme zur Beschreibung von CoBe

Die grundlegenden Begriffe, die in CoBe verwendet werden, sind in Kapitel 2 und in Abschnitt 5.1 definiert. Die Definitionen werden in diesem Abschnitt an den Stellen wiederholt, an denen sie benötigt werden.

6.3.1 Das Umfangsmodell von CoBe

Das Umfangsmodell von CoBe beschreibt den Umfang des Software-Produkts. Dazu wird die Definition aus dem IEEE-Standard 1045 (1992) für Software übernommen. Software wird nach ihrem Ursprung unterschieden (siehe Abschnitt 5.1):

Def. **Hinzugefügte Software.** Software, die im Projekt neu erstellt wird.

Def. **Geänderte Software.** Software, die bereits vorhanden war und im Projekt geändert wird.

Def. **Wiederverwendete Software.** Software, die bereits vorhanden war und unverändert im Projekt verwendet wird.

Def. **Neue Software.** Hinzugefügte und geänderte Software.

Weil CoBe zur Planung eingesetzt werden soll, werden Function Points (IFPUG, 2004) als Umfangsmetrik verwendet. Sie können bereits zur Planungszeit gezählt werden (Metzger und Boddie, 1996) und können als gemeinsame Umfangsmetrik über die Projektdauer und Lebensdauer des Produkts verwendet werden (Drappa, 1998). Der Code-Umfang wird in Anweisungen (logische Zeilen nach IEEE 1045, 1992) dargestellt, definiert nach Park (1992), und über einen Umfangsfaktor aus Function Points abgeleitet (Boehm, 2000). Ich wähle Anweisungen als Metrik und nicht physische Zeilen, weil der Umfangsfaktor zwischen Function Points und Anweisungen, nicht zwischen Function Points und Zeilen, definiert ist (Boehm, 2000) und weil Jones (1996) zeigt, dass der Zusammenhang zwischen Function Points und Anweisungen stärker als der Zusammenhang zwischen Function Points und Zeilen ist (Abschnitt 5.1).

Für die Spezifikation und den Entwurf wird jeweils auf den Umfang in Seiten umgerechnet (Drappa, 1998; Jones, 2007). Der Umfangsfaktor für Spezifikation und Entwurf hängt von der Notation, in der die Dokumente erstellt werden, und der Methode zur Erstellung ab (Drappa, 1998; Jones, 2007). Tabelle 6 zeigt die Eingaben.

Eingabeparameter (Prozess und Produkt)	Wertebereich
Umfang hinzugefügter Software	Zahl der Anweisungen oder Function Points
Umfang geänderter Software	
Umfang wiederverwendeter Software	
Umfangsfaktor Spezifikation	Seiten pro Function Point
Umfangsfaktor Entwurf	Seiten pro Function Point
Umfangsfaktor Code	Anweisungen pro Function Point

Tabelle 6: Eingaben für den Umfang

Die Umfangsfaktoren dienen der Kalibrierung von CoBe, weil die Faktoren durch die speziellen Merkmale eines Projekts beeinflusst werden. Zur Kalibrierung sind darum Daten aus ähnlichen Projekten nötig. Zu den Einflüssen auf die Faktoren gehört, dass die Umrechnung auf Anweisungen durch die Programmiersprache beeinflusst ist, durch den Programmierstil oder durch die vorgegebenen Programmierrichtlinien. Die Umrechnung auf den Umfang der Dokumente in Seiten hängt unter anderem von der Notation ab. Da die verwendete Variante der Function-Points nicht für technisch-wissenschaftliche Anwendungen geeignet ist, kann der Umfang dieser Anwendungen in CoBe entweder durch Anweisungen oder andere Function-Point-Varianten

dargestellt werden. Für eine andere Function-Point-Variante müssen die Umfangsfaktoren kalibriert werden. Wird der Umfang in Anweisungen eingegeben, dann werden Function Points nur zur Umrechnung zwischen Code-Umfang und Umfang der Spezifikation und des Entwurfs verwendet.

6.3.2 Der Fehlerbegriff und das Fehlermodell in CoBe

CoBe basiert auf Fehlerzahlen. Darum wird zuerst der in CoBe verwendete Fehlerbegriff definiert. Die Definition soll erlauben, entdeckte Fehler über die gesamte Software-Lebensdauer zu zählen. Angelehnt an Drappa (1998) wird der Fehlerbegriff "Abweichung" ("anomaly") aus IEEE 1044 (1993) verwendet. Der Begriff wird in Abschnitt 5.1 diskutiert:

Def. **anomaly.** Any condition that deviates from expectations based on requirements specifications, design documents, user documents, standards, etc. or from someone's perceptions or experiences. (IEEE 1044, 1993)

Im Modell werden Fehler von Fehlverhalten abgegrenzt (Abschnitt 5.1; IEEE 982.1, 2005; Liggesmeyer, 2002):

Def. **Fehlverhalten**. Ein Fehlverhalten oder Ausfall (failure) zeigt sich dynamisch bei der Benutzung eines Produkts. Beim dynamischen Test einer Software erkennt man keine Fehler, sondern Fehlverhalten bzw. Ausfälle. Diese sind Wirkungen von Fehlern im Programm.

Def. **Fehler**. Ein Fehler oder Defekt (fault, defect) ist bei Software die statisch im Programmcode vorhandene Ursache eines Fehlverhaltens oder Ausfalls.

Zwischen Fehlern und Fehlverhalten wird also eine Ursache-Wirkungs-Beziehung zu Grunde gelegt (IEEE 982.1, 2005; Liggesmeyer, 2002), weil Fehler im Code beim Ausführen des Codes wirksam werden können und sich dann als Fehlverhalten manifestieren. Fehlverhalten wird auch als Fehlersymptom oder Auftreten des Fehlers bezeichnet. Im Review identifizieren die Gutachter Fehler im Prüfling. Im Test werden Abweichungen vom Sollresultat identifiziert, somit wird das Fehlersymptom oder Fehlverhalten erkannt. Die Ursache, der Fehler, muss identifiziert werden.

Fehler werden in CoBe durch Fehlerzahlen repräsentiert, die auf einer Rationalskala anstatt auf einer Absolutskala dargestellt sind. Somit kann mit Anteilen gerechnet werden. Dies ist aus folgenden Gründen nötig: Das Modell wird mit Mittelwerten aus Datensammlungen der Industrie quantifiziert und mit Mittelwerten der Organisation, in der es eingesetzt wird, kalibriert; es berechnet einen statistischen Erwartungswert; Unstetigkeiten durch Rundung müssen vermieden werden.

Fehlerkategorien in CoBe

Fehler werden nach Fehlerart und Fehlerschwere unterschieden, weil sich diese Merkmale auf Fehlerkosten auswirken. Die Fehler werden durch Verteilungen auf die unterschiedlichen Klassen dieser beiden Merkmale verteilt. Die Definition der Fehlerart orientiert sich an Drappa (1998) und Runeson et al. (2006):

Def. **Fehlerart.** Die Fehlerart ist durch die Aktivität bestimmt, durch die ein Fehler entstanden ist.

In CoBe werden Spezifikationsfehler, Entwurfsfehler und Codefehler unterschieden, da diese Arten in der Literatur verwendet werden (Jones, 1996; Kan, 2003). Sie können entstehen, wenn Software erstellt oder korrigiert wird.

Da die Fehlerschwere ganz unterschiedlich definiert werden kann (Abschnitt 5.1), bietet das Modell die Standard-Definition des IEEE-Standards 610 (1992) und drei Klassen aus Frühauf et al. (2006). Blockierende Fehler gehören zu den kritischen Fehlern.

Def. **Fehlerschwere.** Die Fehlerschwere ist das Maß für den Einfluss eines Fehlers auf die Entwicklung oder den Einsatz eines Systems. (IEEE 610, 1992)

Def. **Kritischer Fehler.** Prüfling ist für den vorgesehenen Zweck unbrauchbar, Fehler muss vor der Freigabe behoben werden. (Frühauf et al., 2006)

Def. **Hauptfehler.** Nutzbarkeit des Prüflings ist beeinträchtigt, Fehler sollte vor Freigabe behoben werden. (Frühauf et al., 2006)

Def. **Nebenfehler** stören, aber beeinträchtigen den Nutzen kaum. (Frühauf et al., 2006)

Def. **Blockierender Fehler.** Ein blockierender Fehler verhindert die weitere Ausführung des Programms, etwa um Testfälle durchzuführen. (Bassin et al., 2002).

In Projekten können andere Definitionen verwendet werden. Dann ist es notwendig, die Verteilung der Fehler auf diese Klassen anzupassen; dies gehört zur Kalibrierung. Wird die Fehlerschwere durch den Schaden definiert, den der Fehler beim Einsatz verursacht oder verursachen würde, dann bietet CoBe die Möglichkeit, die Fehlerschwere anhand des Schadens zu definieren (Abschnitt 6.3.10).

6.3.3 Modellierung der Fehlerentstehung in CoBe

Bei der Planung eines Projekts ist die Zahl der enthaltenen oder entstehenden Fehler in der Software unbekannt. Im Modell werden diese Fehlerzahlen aber benötigt, damit Fehlerkosten berechnet werden können. Darum wird in CoBe die Zahl der entstehenden Fehler aus dem Umfang neuer und wiederverwendeter Software und der Qualität wiederverwendeter Software abgeleitet. Der Kalibrierungsparameter (Fehlerfaktor) und die Verteilung der Fehler auf Fehlerschwere und Fehlerarten beeinflussen die Fehlerzahlen (Abbildung 19, Tabelle 7); andere Einflüsse auf die Fehlerzahl sind in CoBe nicht dargestellt, sondern müssen über die Kalibrierung abgebildet werden, weil diese Einflüsse nicht belegt sind (Abschnitt 5.2).

In CoBe wird die Fehlerdichte *fd* verwendet, um die Zahl der in neuer Software eingefügten Fehler F_{neu} zu berechnen, weil die Fehlerdichte die gebräuchliche Metrik für Software-Qualität ist (Fenton und Pfleeger, 1997). Die Fehlerdichte zwischen geänderter und hinzugefügter Software wird im Modell nicht unterschieden, da Erfahrungen

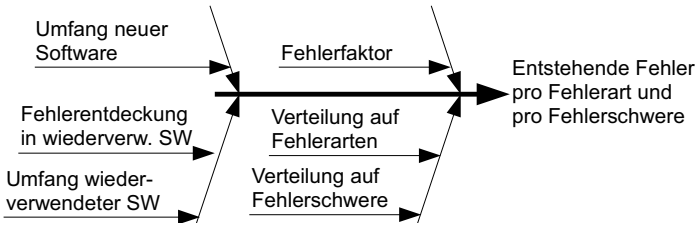

Abb. 19: Ursache-Wirkungs-Diagramm der Fehlerentstehung

Eingabeparameter	Wertebereich
Fehlerfaktor	Verhältnis Istwert und Modellresultat ohne Kalibrierung[a]
Verteilung auf Fehlerschwere (kritische Fehler, Hauptfehler, Nebenfehler)	Anteil jeweils 0...100 %
Verteilung auf Fehlerarten (Spezifikationsfehler, Entwurfsfehler, Codefehler)	Anteil jeweils 0...100 %

Tabelle 7: Eingaben für Fehlerentstehung

a. Zur Kalibrierung werden Archivdaten verwendet.

keinen Unterschied zeigen (Möller und Paulish, 1993b). Die Fehlerdichte *fd* steigt in CoBe mit dem Umfang neuer Software in Function Points (Jones, 1996). Diesen Zusammenhang stelle ich als Logarithmusfunktion zur Basis 10 dar, weil Datensammlungen Projekte nach ihrer Größenordnung in 10er-Potenzen klassifizieren (Jones, 1996 und 2003). Die Parameter r_{0f} und r_{1f} können durch lineare Regression aus Archivdaten berechnet werden. Die Zahl der entstehenden Fehler F_{neu} berechnet sich aus der Fehlerdichte, dem Umfang neuer Software und dem Fehlerfaktor k_F zur Kalibrierung (Abschnitt 6.2.3):[1]

$$fd = r_{0f} + r_{1f} \cdot \log_{10}(S_{FPneu}), \quad F_{neu} = k_F \cdot S_{FPneu} \cdot fd$$

Wiederverwendete Software wird unverändert übernommen, darum entstehen in der wiederverwendeten Software keine neuen Fehler. In der neu erstellten Software, die andere Software wiederverwendet, können Schnittstellenfehler entstehen (Basili und Perricone, 1984). Da es dafür keine weiteren Daten gibt, wird dies im Modell durch die Kalibrierung abgebildet. Die Zahl der Fehler in wiederverwendeter Software F_{wv} wird aus der Fehlerdichte *fd*, dem Kalibrierungsparameter k_F und dem Anteil der Fehler, die bereits vor der Wiederverwendung entfernt wurden (Q_{wv}), berechnet:

$$F_{wv} = k_F \cdot S_{FPwv} \cdot fd \cdot (1 - Q_{wv})$$

1. Die Bezeichner in den Formeln sind im Verzeichnis der Bezeichner ab Seite 281 beschrieben.

Die entstehenden Fehler werden auf die Klassen für Fehlerschwere und Fehlerart anhand der Eingaben in Prozent verteilt.

6.3.4 Modellierung der Fehlerentdeckung und Fehlerkorrektur in CoBe

Fehler werden in Prüfungen entdeckt. Die Sequenz der möglichen Prüfungen in CoBe (Abbildung 20) lehnt sich an Prozessstandards und typische Prozesse an (Abschnitt 5.2). Zu diesen Prozessen und Vorgaben gehören die folgenden Prüfungen: Spezifikationsreview, Entwurfsreview, Codereview und automatische statische Codeanalyse (kurz: Codeanalyse), Modultest, Subsystem- und Systemintegrationstest und Systemtest, außerdem der Test durch den Kunden, der auch als Feldtest bezeichnet wird (Jones, 1996). Diese Prüfungen werden darum in CoBe dargestellt.

Die Prüfsequenz, der CoBe folgt, leite ich aus den gleichen Prozessen und Vorgaben ab; sie folgt den Abstraktionsebenen der Entwicklung (Spezifikation, Entwurf, Implementierung). Die Sequenz basiert auf den folgenden Annahmen:

- Ich nehme ein sequentielles Vorgehen an, so dass zuerst Software erstellt, dann geprüft, dann korrigiert wird. Die Spezifikation wird also beispielsweise nach dem Review erst korrigiert, bevor der Entwurf beginnt.
- Ich nehme an, dass Entwickler die Codeanalyse selbst anstoßen, sie findet also parallel zur Implementierung statt.
- Codereviews finden nach Modultest und Subsystemintegrationstest statt, weil Reviews Zeit zur Organisation benötigen. Modultests werden häufig durch die Entwickler selbst und verwoben mit der Implementierung durchgeführt (Siegwart, 2004). Auch die Subsystem-Integration mit dem Test findet im Modell vor dem Review statt, weil beispielsweise kontinuierlich integriert wird.

Diese Sequenz deckt also das typische Vorgehen vieler Projekte ab (Abschnitt 5.2); für Projekte mit anderer Reihenfolge muss CoBe angepasst werden. Mit den Eingaben über den Prüfprozess kann bestimmt werden, welche dieser Prüfungen durchgeführt werden (Parameter ***P***)[1], und ob in einer Prüfung auch wiederverwendeter Code geprüft wird (Parameter ***WV***, Tabelle 8).

Die Fehlerentdeckungsquote *Q* beschreibt die Fehlerentdeckung einer Prüfung:

Def. **Fehlerentdeckungsquote *Q***. Anteil der entdeckten Fehler an den enthaltenen Fehlern, unterschieden nach Fehlerart und Fehlerschwere.

Jedes Prüfungsmodell berechnet die zugehörige Fehlerentdeckungsquote *Q*, die unterschiedlich hoch für verschiedene Fehlerarten und Fehlerschwere ist.

1. Binäre Eingaben und Auswahleingaben sind **fett** gesetzt.

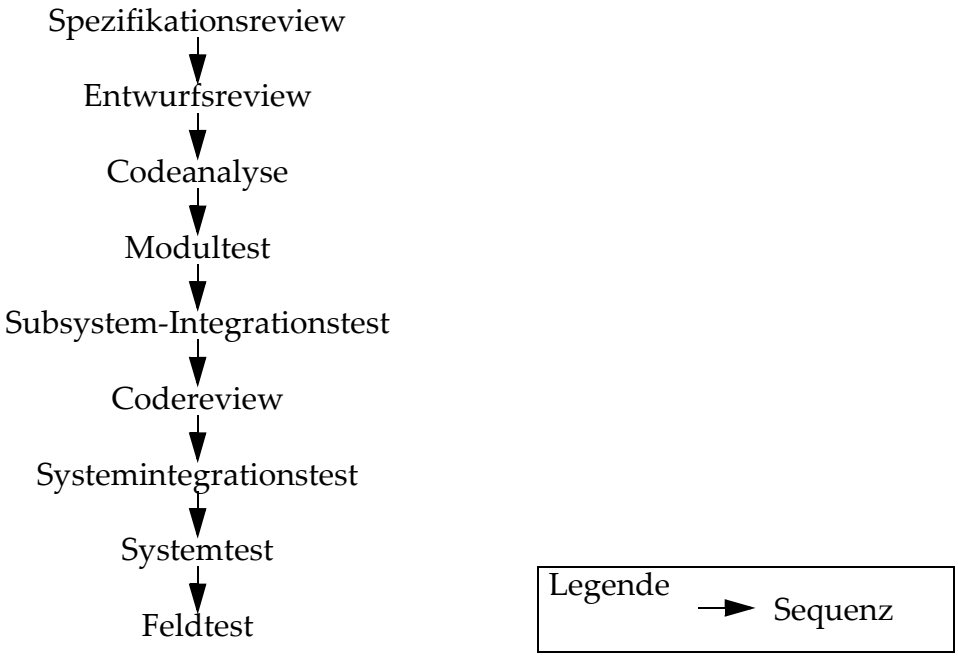

Abb. 20: Prüfsequenz in CoBe

Eingabeparameter für	Wertebereich	
	Durchführung der Prüfung (***P***)	einschl. wiederverwendeter Software (***WV***)
Spezifikationsreview	ja / nein	ja / nein
Entwurfsreview	ja / nein	ja / nein
Codeanalyse	ja / nein	ja / nein
Modultest	ja / nein	ja / nein
Subsystem-Integrationstest	ja / nein	ja / nein
Codereview	ja / nein	ja / nein
Systemintegrationstest	ja / nein	ja / nein
Systemtest	ja / nein	ja / nein
Feldtest	ja / nein	ja / nein

Tabelle 8: Eingaben für Prüfprozess

Bei der Korrektur können Fehler unvollständig korrigiert werden oder neue Fehler eingefügt werden. Weil in CoBe Fehler als Zahl dargestellt werden, gibt es in CoBe keine Unterscheidung, ob entdeckte Fehler nicht korrigiert oder Fehler neu eingefügt wurden. Stattdessen wird durch die Korrekturquote Q_K dargestellt, dass nur ein Anteil der Fehler korrigiert wird:

Def. **Korrekturquote Q_K.** Anteil der entdeckten Fehler, der korrigiert wird.

Im Modell wird die gleiche Korrekturquote für die unterschiedlichen Fehlerarten und für unterschiedlich schwere Fehler verwendet. Das Modell erlaubt unterschiedliche Korrekturquoten, um beispielsweise darzustellen, dass schwere Fehler bevorzugt korrigiert werden. Weil dazu aber detaillierte Erfahrungswerte fehlen, ist dieser Aspekt quantitativ nicht dargestellt. Abbildung 21 zeigt das Fehlerstrommodell mit diesen Parametern.

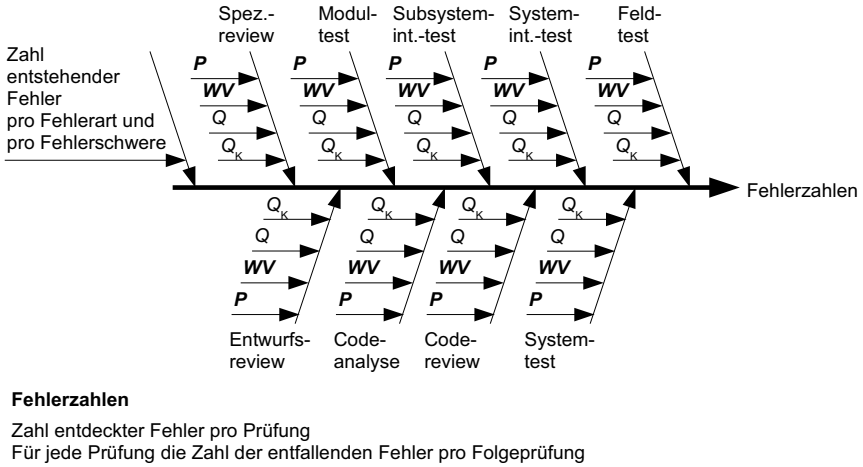

Abb. 21: Ursache-Wirkungs-Diagramm für das Fehlerstrommodell

Entdeckte und korrigierte Fehler pro Prüfung

Weil die Fehlerkosten von der Fehlerart, der Fehlerschwere und der Prüfung, bei der ein Fehler entdeckt wird, abhängen, berechnet CoBe für jede Prüfung die Zahl der entdeckten Fehler, der korrigierten Fehler und der danach in der Software enthaltenen Fehler jeweils getrennt nach Fehlerart und Fehlerschwere (Abbildung 22).

Fehlerzahlen werden mit *F* bezeichnet. Sie werden getrennt nach dem Ursprung der Software, d.h. getrennt für neue und für wiederverwendete Software, berechnet. Fehler in neuer Software werden entdeckt und korrigiert, wenn die Prüfung stattfindet, wenn also ***P*** auf "ja" gesetzt ist (Tabelle 8). Ist zusätzlich ***WV*** gesetzt, dann wird auch wiederverwendete Software geprüft. Somit werden Fehler in wiederverwendeter Software entdeckt und korrigiert. Beispielsweise wird die Zahl der im Spezifikations-

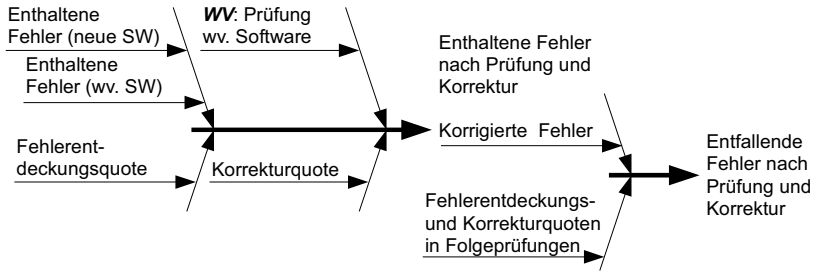

Abb. 22: Ursache-Wirkungs-Diagramm für enthaltene, entdeckte, korrigierte und entfallende Fehler

review (SR) entdeckten Fehler $F_{\text{SR,entdeckt},Ursprung,Art,Schwere}$ für Fehler der *Art* und *Schwere* aus den eingefügten Fehlern für neue und für wiederverwendete Software (*Ursprung*) berechnet:

$$F_{\text{SR,entdeckt},\,Ursprung,\,Art,\,Schwere} = F_{Ursprung,\,Art,\,Schwere} \cdot Q_{\text{SR},Art,\,Schwere}$$

Bei der Korrektur werden Fehler unvollständig korrigiert und neue Fehler eingefügt, so dass nur ein Teil der entdeckten Fehler, $F_{\text{SR,korrigiert},Ursprung,Art,Schwere}$, entfernt wird:

$$F_{\text{SR,korrigiert},\,Ursprung,\,Art,\,Schwere} = F_{Ursprung,\,Art,\,Schwere} \cdot Q_{\text{SR},Art,\,Schwere} \cdot Q_K$$

Nach der Korrektur verbleiben Fehler in der Software; dies sind die enthaltenen Fehler $F_{\text{SR,enthalten},Ursprung,Art,Schwere}$:

$$F_{\text{SR,enthalten},\,Ursrpung,\,Art,\,Schwere} = F_{Ursprung,\,Art,\,Schwere} - F_{\text{SR,korrigiert},\,Ursprung,\,Art,\,Schwere}$$

Beispielsweise werden 50 der 100 Spezifikationsfehler entdeckt ($Q = 50\,\%$). Die Korrekturquote beträgt 90 %, d.h. in der Korrektur werden 90 % der entdeckten Fehler tatsächlich korrigiert (45 Fehler). Damit bleiben 55 Spezifikationsfehler übrig.

Nach dem Spezifikationsreview kommen durch den Entwurf Entwurfsfehler in die Software. Im Entwurfsreview sind also diejenigen Fehler zu finden, die nach der Korrektur des Spezifikationsreviews und nach dem Entwurf enthalten sind; sie stammen aus der Spezifikation oder aus dem Entwurf:

$$F_{\text{ER,entdeckt},\,Ursprung,\,Art,\,Schwere} = F_{\text{SR,enthalten},\,Ursrpung,\,Art,\,Schwere} \cdot Q_{\text{ER},Art,\,Schwere}$$

Mit einer Fehlerentdeckungsquote für Spezifikationsfehler von 10 % entdeckt das Entwurfsreview also 5,5 Fehler der 55 Spezifikationsfehler und zusätzlich Entwurfsfehler, die in diesem Beispiel nicht betrachtet werden. Die Fehlerentdeckung einer Prüfung bezieht sich immer auf die enthaltenen Fehler, also diejenigen Fehler, die

nach der Korrektur der vorherigen Prüfung übrig sind, und diejenigen Fehler, die neu in die Software eingefügt wurden.

Die Zahl entdeckter und korrigierter Fehler der anderen Prüfungen im Modell wird auf die gleiche Art und Weise berechnet. Daraus ergibt sich die Zahl der ausgelieferten Fehler. Ein Teil dieser Fehler tritt nicht auf, wird nicht gemeldet oder erscheint tolerierbar und wird darum auch nicht in der Wartung korrigiert.

Entfallende Fehler durch Prüfung und Korrektur

In CoBe ist der Nutzen durch entfallende Fehlerkosten definiert. Diejenigen Fehler, die bereits korrigiert wurden, können nicht mehr in folgenden Prüfungen und nach Auslieferung entdeckt werden. Diese Fehler werden im Folgenden als entfallende Fehler bezeichnet. Die Zahl entfallender Fehler wird mit den gleichen Fehlerentdeckungs- und Korrekturquoten berechnet wie die Zahl entdeckter Fehler (Abbildung 22); dabei entfallen so viele Fehler in den Folgeprüfungen und nach Auslieferung, wie korrigiert wurden. Im Beispiel oben entfallen also die 45 Spezifikationsfehler, die nach dem Spezifikationsreview korrigiert werden, in den Folgeprüfungen und nach Auslieferung. Ohne das Spezifikationsreview und dessen Korrektur würden z.B. 10 % dieser Fehler im Entwurfsreview entdeckt werden ($F_{\mathrm{SR,entfallend,ER}} = 4{,}5$ Fehler) und entsprechende Korrekturkosten verursachen.

$$F_{\mathrm{SR,entfallend,ER},\, Ursprung,\, Art,\, Schwere} = F_{\mathrm{SR,korrigiert},\, Ursprung,\, Art,\, Schwere} \cdot Q_{\mathrm{ER},\, Art,\, Schwere}$$

Wird zum Beispiel in CoBe eingegeben, dass im Prüfprozess keine Codeanalyse stattfindet, aber ein Modultest und ein Subsystem-Integrationstest, dann entfallen Fehler, die bereits nach dem Spezifikationsreview korrigiert wurden, im Modultest (MT) und dann im Subsystem-Integrationstest (PT[1]):

$$F_{\mathrm{SR,entfallend,MT},\, Ursprung,\, Art,\, Schwere} = F_{\mathrm{SR,korrigiert},\, Ursprung,\, Art,\, Schwere} \cdot (1 - Q_{\mathrm{ER},\, Art,\, Schwere} \cdot Q_{\mathrm{K,ER}}) \cdot Q_{\mathrm{MT},\, Art,\, Schwere}$$

$$F_{\mathrm{SR,entfallend,PT},\, Ursrprung,\, Art,\, Schwere} = F_{\mathrm{SR,korrigiert},\, Ursprung,\, Art,\, Schwere} \cdot (1 - Q_{\mathrm{ER},\, Art,\, Schwere} \cdot Q_{\mathrm{K,ER}}) \cdot (1 - Q_{\mathrm{MT},\, Art,\, Schwere} \cdot Q_{\mathrm{K,MT}}) \cdot Q_{\mathrm{PT},\, Art,\, Schwere}$$

1. Der Subsystem-Integrationstest wird durch PT gekennzeichnet; dabei steht P für “Package”.

Falsche Befunde

Falsche Befunde werden in CoBe durch ihre Anzahl beschrieben. In Studien wird typisch das Verhältnis zwischen falschen Befunden und echten Fehlern dargestellt (Wagner et al., 2005; Zheng et al., 2006). Darum wird im Modell die Zahl der falschen Befunde linear aus der Zahl der entdeckten Fehler berechnet.

6.3.5 Das Modell für den Korrekturaufwand in CoBe

Zu den Kosten der Fehlerbehebung gehören die Korrekturkosten. Dazu gehört auch der Aufwand zur Erkennung falscher Befunde. Das Modell für den Korrekturaufwand beschreibt, welcher Aufwand für die Korrektur der entdeckten Fehler anfällt oder entfällt (Abbildung 23). Dabei spielt die Latenzzeit, der Software-Umfang und die Fehlerschwere eine Rolle.

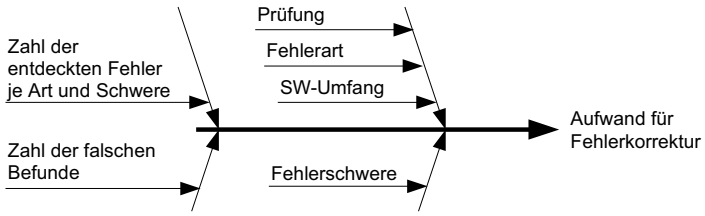

Abb. 23: Ursache-Wirkungs-Diagramm für den Korrekturaufwand

Diese Zusammenhänge werden in CoBe modelliert, in dem ein Basisaufwand pro Fehler, a_{KBasis}, verändert wird. Dieser Basisaufwand unterliegt dem Einfluss der Latenzzeit, also der Zeit, die der Fehler unentdeckt bleibt. Der Einfluss der Latenzzeit wird durch die Fehlerart und den Entdeckungszeitpunkt, d.h. die Prüfung, bei der der Fehler entdeckt wird, modelliert. Für die Fehlerart wird der Basisaufwand mit dem Faktor af_{Art} angepasst. Für den Entdeckungszeitpunkt wird der Basisaufwand mit dem Faktor af_p für die Prüfung p, bei der der Fehler entdeckt wird, und den Faktor af_{W} für die Wartungsphase angepasst. Um den Einfluss des Umfangs auf den Anstieg des Korrekturaufwands zu modellieren, ist der Faktor af_p eine Funktion des Umfangs, der für den Systemtest der Faktor 10 ist, wenn es sich um umfangreiche Software handelt. Er ist 4, wenn es sich um ein kleines Produkt handelt; Zwischenwerte für den Umfang werden interpoliert (Abbildung 24).

Ein Beispiel illustriert diese Berechnung: Mit einem Basisaufwand von einer Entwicklerstunde pro Fehler kostet die Korrektur eines Spezifikationsfehlers nach dem Spezifikationsreview eine Entwicklerstunde. Der Faktor af_{ST} für den Systemtest ist im Beispiel 8, für die Wartung ist af_{W} 15; Spezifikationsfehler sind teurer als Entwurfsfehler und werden durch den Faktor af_{Spez} mit 1,3 angepasst. Codefehler sind günstiger, sie werden mit dem Faktor af_{Code} mit 0,8 angepasst. Damit ist also der Korrekturaufwand für einen Spezifikationsfehler, der im Systemtest entdeckt wurde, 10,4 Entwicklerstunden. Wird ein Spezifikationsfehler in der Wartung korrigiert, kostet dies 19,5 Entwicklerstunden, ein Codefehler kostet 12 Entwicklerstunden.

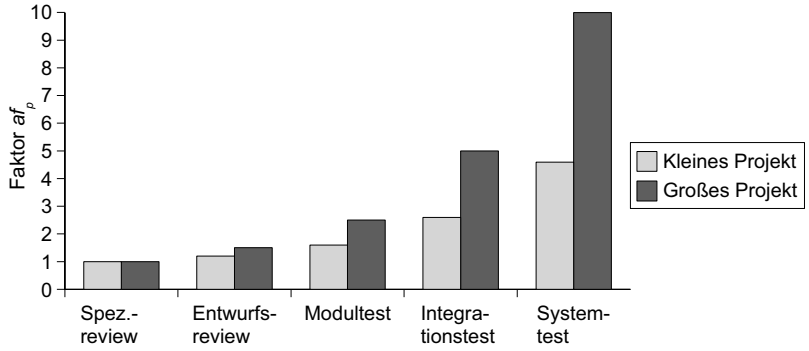

Abb. 24: Einfluss des Umfangs auf den Anstieg der Korrekturkosten

Die Fehlerschwere ist der dritte Einfluss auf den Korrekturaufwand. Der Einfluss ist durch den Faktor $af_{Schwere}$ modelliert, mit af_{NF}, af_{HF} und af_{KF} für Neben-, Haupt- bzw. kritische Fehler.

Für den Korrekturaufwand nach einer Prüfung p werden die Aufwände der entdeckten Fehler aufsummiert. Der Korrekturaufwand $A_{\mathrm{K},p}$ nach der Prüfung p berechnet sich somit aus der Zahl entdeckter Fehler $F_{p,\mathrm{entdeckt}}$, dem Basisaufwand a_{KBasis} und den Faktoren af für Fehlerart, Fehlerschwere und Prüfung:

$$A_{\mathrm{K},p} = \sum_{Ursprung,\,Art,\,Schwere} F_{p,\,\mathrm{entdeckt},\,Ursprung,\,Art,\,Schwere} \cdot a_{\mathrm{KBasis}} \cdot af_{Art} \cdot af_{Schwere} \cdot af_p$$

Die Erkennung falscher Befunde gehört zur Korrektur. Der Analyseaufwand, um einen falschen Befund zu erkennen, wird als Anteil des Korrekturaufwands eines Fehlers berechnet; die Zahl der entdeckten falschen Befunde erfolgt durch das Modell der Fehlerentdeckung.

6.3.6 Das Modell für die Kosten der Prüfwiederholung in CoBe

Zu den Fehlerbehebungskosten gehören Kosten, die entstehen, wenn nach der Korrektur erneut geprüft wird. In CoBe wird darum zwischen der Prüfung und ihrer Wiederholung nach der Korrektur unterschieden. Die erste Prüfung wird im Folgenden als initiale Prüfung bezeichnet, die Prüfung nach der Korrektur als Prüfwiederholung. Die beiden unterschiedlichen Vorgehen zur Prüfwiederholung in CoBe leiten sich aus dem Standard ISO/IEC 14764 (1999) und aus Pigoski (1997) ab:

a) Nach der Korrektur wird die Prüfung wiederholt.

b) Nach der Korrektur werden mehrere Prüfungen wiederholt (Hörmann et al., 2006). Dies wird in CoBe als Korrekturprüfprozess bezeichnet. Eingegeben wird, nach welchen Prüfungen ein Korrekturprüfprozess stattfindet, und welche Prüfungen

zu diesem Prozess gehören. In der Wartung kann sich der Korrekturprüfprozess kritischer Fehler vom Prüfprozess der Haupt- und Nebenfehler unterscheiden (Sneed et al., 2004).

Um diesen Aufwand zu modellieren, wird in CoBe unabhängig vom Vorgehen der Wiederholungsaufwand *einer* Prüfung auf gleiche Art berechnet. Im Fall a) wird der Aufwand nur für die bestimmte Prüfung, im Fall b) für alle Prüfungen des Prüfprozesses berechnet. Die unterschiedlichen Möglichkeiten, eine einzelne Prüfung zu wiederholen (Thaller, 2002; Sneed et al., 2004; Müller et al., 1998), werden in CoBe durch drei Möglichkeiten abgebildet:

- ohne: Die Prüfung wird nicht wiederholt.
- gezielt: Für jeden Fehler wird ein Teil der Prüfung wiederholt.
- vollständig: Die gesamte Prüfung wird wiederholt, nachdem alle Fehler korrigiert wurden.

CoBe enthält die Wiederholung von Tests und die gezielte Wiederholung des Codereviews (Tabelle 9 und Tabelle 11). Für jede Prüfung kann angegeben werden, ob nach der Korrektur die Prüfung wiederholt wird oder ob nach der Korrektur der Korrekturprüfprozess durchgeführt wird (Tabelle 10, Prüfung mit Korrekturprüfprozess). In CoBe wird eingegeben, welche Prüfungen zu diesem Korrekturprüfprozess gehören (Tabelle 10, Prüfung gehört zum Korrekturprüfprozess). Werden beispielsweise Korrekturen nach dem Modultest durch eine Wiederholung des Modultests geprüft, dann wird der Parameter 'Prüfung mit Korrekturprüfprozess' für den Modultest auf 'nein' gesetzt. Werden Korrekturen nach dem Systemtest durch ein Codereview, einen erneuten Integrationstest und einen erneuten Systemtest geprüft, dann wird der Parameter 'Prüfung mit Korrekturprüfprozess' für den Systemtest auf 'ja' gesetzt. Zusätzlich werden die Parameter 'Prüfung gehört zum Korrekturprüfprozess' für Codereview, Integrationstest und Systemtest auf 'ja' gesetzt.

Ich modelliere die vollständige Wiederholung der Spezifikations- und Entwurfsreviews nicht, weil bereits eine erste Begutachtung in vielen Situationen kaum durchsetzbar ist (Schwinn, 2003); diese Möglichkeit ist also kaum praxisrelevant.

Der Aufwand für die Prüfwiederholung basiert auf dem Aufwand der initialen Prüfung. Für den Test ist dies der Aufwand für die Testdurchführung (Abschnitt 6.6.2), weil vorausgesetzt wird, dass die gleichen Testfälle wiederholt werden, und dass diese Testfälle nicht erneut definiert werden. Dieser Aufwand wird in CoBe als initialer Durchführungsaufwand bezeichnet. Er hängt von der speziellen Projektsituation ab, darum enthält CoBe zwei Eingaben (Tabelle 12):

- Der Aufwand für die vollständige Wiederholung eines Tests berechnet sich aus dem initialen Durchführungsaufwand und dem Anteil des Aufwands, der für die Wiederholung benötigt wird ($a_{\text{wdh},Test}$). Dieser Wiederholungsanteil hängt beispielsweise von der Automatisierung des Tests ab.

Eingabeparameter für Prüfung[a]	Wertebereich
	Wiederholung der Prüfung
Modultest	ohne / gezielt / vollständig
Subsystemint.-test	ohne / gezielt / vollständig
Codereview[b]	ohne / gezielt
Systemint.-test	ohne / gezielt / vollständig
Systemtest	ohne / gezielt / vollständig
Feldtest[c]	ohne / gezielt

Tabelle 9: Parameter für Prüfwiederholung

a. Wiederholung der Spezifikations- und Entwurfsreviews ist nicht modelliert.
b. Das gezielte Review einer Codeänderung ist eine eigene Modellkomponente.
c. Im Feldtest wird vereinfacht der Test als Zuschlag zur Korrektur dargestellt, weil der Test beim Kunden durch Probeeinsatz stattfindet.

Eingabeparameter für Prüfung	Wertebereich	
	Prüfung mit **Korrekturprüfprozess**	Prüfung gehört zum **Korrekturprüfprozess**
Modultest	ja / nein	ja / nein
Subsystemint.-test	ja / nein	ja / nein
Codereview	ja / nein	ja / nein
Systemint.-test	ja / nein	ja / nein
Systemtest	ja / nein	ja / nein
Feldtest	ja / nein	-

Tabelle 10: Parameter für Prüfwiederholung

- Zusätzlich wird bei gezielter Wiederholung berücksichtigt, dass pro Fehler nur ein Teil des Tests wiederholt wird. Dieser Wiederholunganteil ($s_{\text{wdh},Test}$) wird benötigt, um beispielsweise einen bestimmten Ausgangszustand der Software herzustellen oder eine Sequenz von Testfällen zu wiederholen, um die Korrektur zu prüfen.

Der Wiederholungsaufwand nach einer Korrektur wird abhängig vom Vorgehen der Wiederholung berechnet (Abbildung 25): Wird eine Prüfung gezielt für jeden Fehler wiederholt, dann wird der Aufwand pro Fehler aufsummiert. Auch der Korrekturprüfprozess wird für jeden Fehler einzeln durchgeführt, darum werden die Aufwände aller Prüfungen des Korrekturprüfprozesses pro Fehler zusammengezählt. Der Aufwand für die vollständige Wiederholung berechnet sich direkt als Anteil des

Prüfstrategie in der Wartung[a]	Wertebereich für **Wiederholung** nach Korrektur von	
	kritischen Fehlern	Haupt- und Nebenfehlern
Modultest	ohne / gezielt / vollständig	ohne / gezielt / vollständig
Subsystemint.-test	ohne / gezielt / vollständig	ohne / gezielt / vollständig
Codereview[b]	ohne / gezielt	ohne / gezielt
Systemint.-test	ohne / gezielt / vollständig	ohne / gezielt / vollständig
Systemtest	ohne / gezielt / vollständig	ohne / gezielt / vollständig

Tabelle 11: Parameter für Prüfstrategie in der Wartung

a. Wiederholung der Spezifikations- und Entwurfsreviews und Feldtest sind für die Wartung nicht modelliert.
b. Das gezielte Review von Codeänderungen ist als einzelne Prüfung modelliert.

Eingabeparameter für Prüfung[a]	Wertebereich für Wiederholungsanteil des	
	Aufwands	Umfangs
Modultest	0...100%	0...100%
Subsystemint.-test	0...100%	0...100%
Codereview[b]	(100%)	(25 Codezeilen)
Systemint.-test	0...100%	0...100%
Systemtest	0...100%	0...100%
Feldtest[c]	0...100%	-

Tabelle 12: Parameter für Prüfwiederholung

a. Wiederholung der Spezifikations- und Entwurfsreviews ist nicht modelliert.
b. Das gezielte Review von Codeänderungen ist als einzelne Prüfung modelliert.
c. Im Feldtest wird vereinfacht der Test als Zuschlag zur Korrektur dargestellt, weil der Test beim Kunden durch Probeeinsatz stattfindet.

initialen Durchführungsaufwands. Für die Berechnung des Nutzens spielen nur die gezielte Prüfung und der Korrekturprüfprozess eine Rolle, weil nur dann der Wiederholungsaufwand von der Zahl entdeckter Fehler abhängt.

Durch die Testwiederholung wird ein Teil der fehlerhaften Korrekturen entdeckt, da ich voraussetze, dass die Testfälle unverändert wiederholt werden. Diese Fehler werden erneut korrigiert. In CoBe wird dies durch Anpassung der Fehlerentdeckungs- und Korrekturquote modelliert. Wie sich unterschiedliche Korrekturprüfprozesse mit mehr oder weniger Prüfungen auf die Fehlerentdeckung nach der Korrektur auswirken, ist in CoBe über die Korrekturquote parametrisierbar. Da keine Daten zu unter-

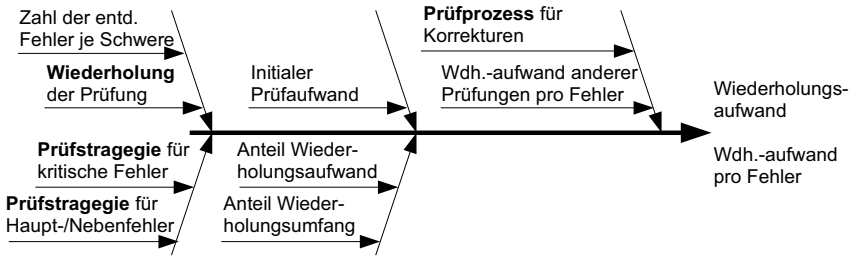

Abb. 25: Ursache-Wirkungs-Diagramm des Aufwands zur Prüfwiederholung

schiedlichen Prüfprozessen verfügbar sind, muss dieser Unterschied speziell für konkrete Situationen quantifiziert werden.

6.3.7 Das Modell für den Aufwandseinfluss in CoBe

Damit der Aufwand an konkrete Situationen angepasst werden kann, wird in CoBe zuerst der nominale Aufwand einzelner Aktivitäten berechnet. Diese nominalen Aufwände werden alle gleichförmig, d.h. durch die gleichen Faktoren mit den gleichen Werten, angepasst. Ich verwende die folgenden Zusammenhänge aus COOCMO II: Die Einflussfaktoren *EM* für Merkmale des Produkts, der Plattform, des Personals und des Prozesses, die Skalierungsfaktoren *SF* und der Einfluss des Umfangs *S* wirken sich auf die Aufwände in CoBe aus. Der Aufwand einzelner Aktivitäten steigt in gleichem Maße, in dem der Gesamtaufwand mit dem Umfang wächst. In diesem Teil von CoBe wird auch der Kalibrierungsparameter für den Aufwand, der Aufwandsfaktor k_A, und der Organisationsaufwand als prozentualer Zuschlag af_O berücksichtigt. Tabelle 13 zeigt die Eingaben, Abbildung 26 die Ursache-Wirkungs-Beziehungen.

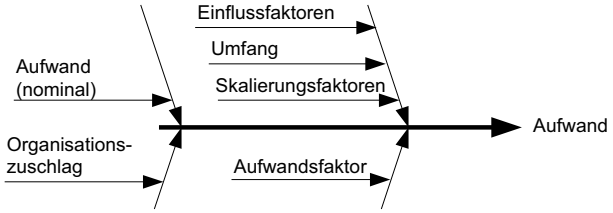

Abb. 26: Ursache-Wirkungs-Diagramm der Aufwandseinflüsse

Der Produktivitätsparameter *af* von CoBe fasst diese Einflüsse zusammen. Die Skalierungsfaktoren von COCOMO II bestimmen über den Exponenten *E* den überproportionalen Einfluss des Umfangs (*S*) auf den Aufwand (*PM*); *A* ist der Produktivitätsfaktor in COCOMO II.

Eingabeparameter	Wertebereich
COCOMO-II-Parameter: Skalierungs- und Einflussfaktoren	Kategorien mit jeweils 7 Klassen (Boehm, 2000)
Aufwandsfaktor k_A	Verhältnis Istwert und COCOMO-II-Resultat ohne Kalibrierung[a]
Organisationszuschlag af_O	0 %... X %

Tabelle 13: Eingaben für Aufwandseinflüsse

a. Zur Kalibrierung werden Archivdaten verwendet.

$$af = k_A \cdot af_O \cdot \prod_{i=1}^{17} EM_i \cdot \left(\frac{S_{\text{Anweisungen}}}{1000}\right)^{(E-1)}$$

$$\text{mit } E = B + 0,01 \cdot \sum_{j=1}^{5} SF_j \text{ und } PM = A \cdot \left(\frac{S_{\text{Anweisungen}}}{1000}\right)^{E} \cdot \prod_{i=1}^{17} EM_i .$$

Wiederverwendete und geänderte Software wird auf äquivalente Anweisungen umgerechnet, um Auswahl und Einarbeitung zu berücksichtigen (Boehm, 2000).

6.3.8 Das Modell für Dauer und Personal in CoBe

Weil Dauer und Personalbedarf wichtige Planungsmetriken sind, werden in CoBe Dauer und Personalbedarf aus dem Aufwand einzelner Aktivitäten berechnet, basierend auf der COCOMO zu Grunde liegenden Annahme, dass es eine typische, ideale Personenzahl für ein Projekt und für seine Aktivitäten gibt.

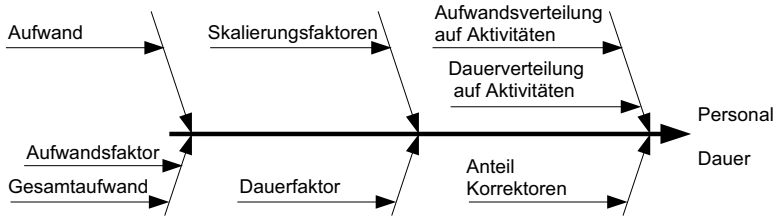

Abb. 27: Ursache-Wirkungs-Diagramm für Dauer- und Personalberechnung

Dieses typische Verhältnis wird in CoBe mit COCOMO II berechnet (Abbildung 27). Dazu wird zuerst das Verhältnis zwischen Aufwand, Dauer und Personalbedarf für das gesamte Projekt berechnet. Dabei spielen die COCOMO-II-Einflüsse eine Rolle. Zusätzlich werden zur Kalibrierung von CoBe der Aufwandsfaktor und der Dauer-

faktor von CoBe berücksichtigt. Dann wird der Personalbedarf für einzelne Aktivitäten mit den folgenden Schritten bestimmt:

- COCOMO und COCOMO II (Boehm, 1981 und 2000) enthalten Tabellen, in denen die Verteilung des Aufwands und die Verteilung der Dauer auf Phasen und Aktivitäten angegeben werden. Beispielsweise benötigt der Entwurf 6 % des Projektaufwands und 19 % der Projektdauer. Da es die Tabellen für feste Umfänge gibt (2 KDSI[1], 8 KDSI, 16 KDSI, 64 KDSI, 128 KDSI, 512 KDSI), werden in CoBe Zwischenwerte linear aus dem Umfang interpoliert.
- Die Tabellen gibt es für die drei Projektarten in COCOMO: organic, semi-detached, embedded. Für jede dieser Projektarten ist ein Exponent für die Berechnung des Aufwands und ein Exponent für die Berechnung der Dauer vorgegeben. In COCOMO II gibt es keine Projektarten mehr, sondern 5 Skalierungsfaktoren, aus denen diese Exponenten berechnet wird. Darum werden in CoBe die Zwischenwerte für den Exponenten linear interpoliert. Resultat sind also Aufwands- und Dauerverteilung für den Umfang. Kalibriert mit Dauer- und Aufwandsfaktor berechnet sich daraus der Personalbedarf pro Aktivität. Die Dauer wird aus dem Personalbedarf der einzelnen Aktivitäten berechnet (Abbildung 28). Die Dauer wird in Arbeitstagen oder Arbeitsstunden ausgegeben, der Personalbedarf als Anzahl Mitarbeiter. Weil CoBe mit Mittelwerten rechnet, sind alle Ausgaben auf einer Rationalskala.

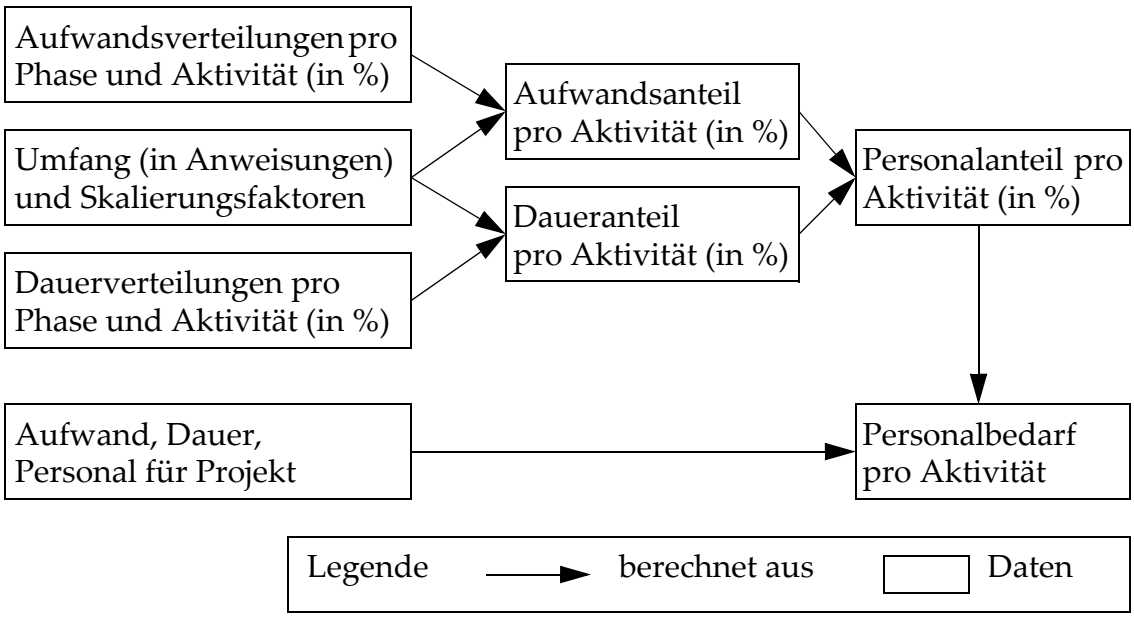

Abb. 28: Berechnung des Personalbedarfs aus COCOMO-II-Parametern

1. KDSI bezeichnet 1000 delivered source instructions, also 1000 gelieferte Anweisungen

CoBe verwendet für den Personalbedarf der Korrektur die Zahl der Entwickler der entsprechenden Phase, da COCOMO die Korrektur nicht darstellt. Es kann eingegeben werden, welcher Teil der Entwickler verfügbar ist (Tabelle 14).

Eingabeparameter	Wertebereich
Dauerfaktor k_D	Verhältnis Istwert und COCOMO-II-Resultat ohne Kalibrierung[a]
Anteil Korrektoren m_K	0...100 %

Tabelle 14: Eingaben für Dauer und Personal

a. Zur Kalibrierung werden Archivdaten verwendet.

6.3.9 Das Geldwerte-Modell von CoBe

Das Geldwerte-Modell beschreibt, wie Planungsmetriken in Geldwerte umgerechnet werden. Dazu können Dauer, Aufwand und Personalbedarf jeweils durch Geld gewichtet werden. Damit wird möglich, Kosten mit Nutzen zu vergleichen, weil alle Auswirkungen der Prüfung auf eine einheitliche Skala abgebildet werden.

Die Gewichtung des Aufwands erfolgt durch Personalkosten. Anders als im QS-Modell (Drappa, 1998) kosten in CoBe alle Mitarbeiter gleich viel, weil Mitarbeiter in CoBe als Zahl dargestellt werden, während sie im QS-Modell unterscheidbar sind und abhängig von ihren Kenntnissen und Fähigkeiten unterschiedlich viel kosten. Die Gewichtung der Dauer und des Personalbedarfs ist projektspezifisch, da in manchen Projekten Verzögerungen teuer sind, etwa weil sie Vertragsstrafen nach sich ziehen; in manchen Projekten ist zusätzliches Personal nicht verfügbar. Diese Faktoren können in CoBe durch die Gewichtung dargestellt werden.

Eine Abzinsung langfristiger Kosten (Hanusch, 1987; Mühlenkamp, 1994; Nas, 1996; Harrison et al., 1999; Raffo, 2005; Kerzner, 2006) ist in CoBe nicht enthalten. Dafür gibt es mehrere Gründe: Die Wartungsphase der Software wird im Modell als eine einzige logische Phase betrachtet, sie kann sich abhängig vom Produkt über einen kürzeren oder einen längeren Zeitraum erstrecken. Abhängig von der Art und Intensität des Einsatzes werden Fehler früher oder später in der Wartungsphase entdeckt. Diese Einflüsse sind im Modell nicht enthalten. Außerdem sind von den langfristigen Auswirkungen andere Personengruppen, die Klienten, betroffen. Dann ist fraglich, ob abgezinst werden darf und wer den Zinssatz bestimmt, der die Resultate stark prägen kann (Hanusch, 1987).

6.3.10 Das Fehlerfolgekostenmodell von CoBe

Fehlerfolgekosten werden nicht im Geldwertemodell, sondern durch ein eigenes Modell dargestellt. Dieses Fehlerfolgekostenmodell beschreibt, welche Kosten beim Einsatz des Produkts durch Fehler verursacht werden. Es gewichtet Fehlverhalten der Software mit Geldwerten, angelehnt an die Unzuverlässigkeitsmetrik von Ludewig

und Lichter (2007). Im Modell der Fehlerfolgekosten von CoBe werden die Folgekosten von Fehlern über die gesamte Lebensdauer des Produkts betrachtet, basierend auf zwei Zusammenhängen:

- Wenn ein Fehler auftritt, dann wird Schaden verursacht; jedes Fehlverhalten kostet. Dazu gehört, dass ein Benutzer Arbeitszeit verliert oder dass die Software direkt finanziellen Schaden verursacht.
- Ob und wie häufig ein Fehler auftritt, hängt von der Verwendung der Software ab. Da die Software auf eine bestimmte Art verwendet werden muss, um den Fehler wirken zu lassen, hängt dies von vielen Einflüssen und einer komplexen Ursache-Wirkungs-Kette ab. So kann ein einzelner Fehler mehrfach auftreten, wenn beispielsweise die gleichen Daten mehrfach eingegeben werden. Ein Fehler kann nicht auftreten, weil genau die Datenkombination, die den Fehler wirksam werden lässt, nicht vorkommt. Ein Fehler kann in einem fehlertoleranten System zwar wirksam werden, sich aber nicht als Fehlverhalten manifestieren, weil dies durch das System verdeckt wird. Obwohl der Zusammenhang zwischen Fehler (Ursache) und Fehlverhalten (Wirkung) also kausal ist, ist eine Aussage über die Häufigkeit, mit der Fehlerursachen wirksam werden, nur im Rückblick möglich. Somit ist nur im Rückblick möglich, die Häufigkeit zu messen, aber nicht im Voraus, die Häufigkeit deterministisch zu bestimmen.

Nachträglich können diese Daten im Prinzip gemessen werden. Zur Planung sind sie aber aus den folgenden Gründen nicht verfügbar:

- Absolute Fehlerzahlen sind bei der Planung nicht bekannt.
- Der Schaden, der durch Fehlverhalten verursacht wird, kann im Prinzip bei den Benutzern gemessen werden. Praktisch sind die Benutzer für die Hersteller in vielen Fällen nicht zugänglich; die Messung ist besonders schwierig, wenn der Schaden indirekt entsteht, z.B. die Kunden des Kunden betrifft. Selbst Archivdaten sind darum kaum verfügbar.
- Die Verwendung der Software kann ganz verschieden sein. Beispielsweise kann ein Produkt bei einem einzigen Kunden regelmäßig verwendet werden. Ein anderes Produkt kann einen Service bieten, der mehr oder weniger häufig genutzt wird. Bei einem Produkt für den Markt ist unklar, wie viele Benutzer die Software haben wird. Bei der Planung ist es also in vielen Fällen kaum möglich, die genaue Häufigkeit der Verwendung zu prognostizieren.
- Die Wahrscheinlichkeit, mit der ein Fehler auftritt, ist durch die Verwendung der Software und durch den Fehler bestimmt. Fehler sind bei der Planung unbekannt. Die konkrete Verwendung der Software ist bei der Planung nicht bekannt. Darum ist eine genaue Prognose kaum möglich.
- Zuverlässigkeitstests (Poore und Trammell, 1996) können nicht zur Planung eingesetzt werden, um z.B. die Häufigkeit des Fehlverhaltens zu prognostizieren (Abschnitt 5.3), weil diese Tests ohne Programm nicht durchgeführt werden kön-

nen. Zuverlässigkeitsmodelle (Lyu, 1995) benötigten Testdaten zur Quantifizierung; es ist unklar, ob und unter welchen Bedingungen Archivdaten verwendet werden können. Zumindest für unterschiedliche Benutzungsprofile müssen sie erneut angepasst werden. Unterschiedlich hohe Schäden unterschiedlicher Fehler werden in den Zuverlässigkeitstests und -modellen nicht berücksichtigt.

Weil also zur Planung wenig Informationen verfügbar sind, wird in CoBe ein statistischer Mittelwert für die Folgekosten eines Fehlers aus Eingaben, die zur Planungszeit bestimmt werden können, abgeschätzt. Daraus werden anfallende und entfallende Fehlerfolgekosten der ausgelieferten Fehler berechnet. Zu diesem Zweck wird für CoBe ein risikobasierter Ansatz (Boehm, 1991) mit Klassifikationen gewählt. Risiko ist definiert als die Wahrscheinlichkeit für einen Schaden. Der Risikowert ist definiert als Produkt aus Eintrittswahrscheinlichkeit und Schaden. CoBe verwendet drei Parameter:

- Auftretenswahrscheinlichkeit: Ein Fehler führt bei einer Verwendung der Software mit einer bestimmten Wahrscheinlichkeit zu einem Fehlverhalten.
- Schaden: Ein Fehlverhalten verursacht einen bestimmten, monetär bezifferbaren Schaden. Ein einzelner Fehler kann in der Realität mehrfach auftreten und dabei unterschiedlichen Schaden hervorrufen, je nachdem, in welcher Situation der Fehler wirksam wird. In Cobe wird pro Fehler ein statistischer Mittelwert verwendet; ein einzelner Fehler verursacht also bei jedem Auftreten einen bestimmten mittleren Schaden. Ein anderer Fehler kann in CoBe beim Auftreten zu einem anderen mittleren Schaden führen.
- Verwendungshäufigkeit: Die Software kann unterschiedlich häufig verwendet werden, bis ein Fehler korrigiert wird. Je häufiger die Software verwendet wird, desto häufiger hat ein Fehler die Chance, wirksam zu werden. Dabei werden nicht alle Fehler gemeldet oder korrigiert, damit kann die Verwendungshäufigkeit bis zur Korrektur also unterschiedlich sein. Im Extremfall bleibt ein Fehler über die gesamte Lebensdauer in der Software.

Mit diesen drei Fehlermerkmalen können die Fehlerfolgekosten eines Fehlers berechnet werden:

$$\text{Fehlerfolgekosten eines Fehlers} = \text{Schaden} \cdot \text{Auftretenswahrscheinlichkeit} \cdot \text{Verwendungshäufigkeit}$$

Da während der Planung nicht auf Messungen zurückgegriffen werden kann, werden in CoBe Fehler nicht einzeln klassifiziert. Stattdessen wird der Anteil der Fehler pro Schadensklasse, pro Klasse der Auftretenswahrscheinlichkeit und pro Klasse der Verwendungshäufigkeit eingegeben. Daraus werden die mittleren Fehlerfolgekosten pro Fehler berechnet. Beispielsweise kann eingegeben werden, dass 20 % der Fehler zur Schadensklasse der Komfortprobleme gehören und 80 % der Fehler zur Schadensklasse für eine verlorene Aufwandsstunde. 20 % der Fehler treten in den Hauptfunktionen auf, d.h. sie treten mit hoher Wahrscheinlichkeit auf, 20 % sind in den

Nebenfunktionen und treten darum selten auf, 60 % der Fehler treten in Ausnahmefällen, also sehr selten, auf. Die Software wird von einem Benutzer zehnmal verwendet, bis der Fehler korrigiert wird. Wie in COCOMO II werden die Klassen durch quantitative Angaben und Beschreibungen definiert. Die Klassifikation für den Schaden (Tabelle 15) orientiert sich an der Zuverlässigkeitsbewertung in COCOMO II. Tabellen 16 und 17 zeigen die Klassifikation für die Auftretenswahrscheinlichkeit und die Verwendungshäufigkeit. Ich nehme an, dass die Fehleranteile, d.h. die Verteilung der Fehler auf die Klassen, aus ähnlichen Projekten übernommen werden können.

Schaden (Euro)	Beschreibung und Beispiele	Wertebereich: Fehleranteil[a]
0	kein Schaden	0...100 %
10	Komfortprobleme, typische Bedienungsprobleme	0...100 %
100	Geringer, leicht auszugleichender Schaden, Workarounds mit einer Dauer bis zu einer Stunde	0...100 %
1000	Mittlerer, auszugleichender Schaden, Verlust von etwa einem Tag Arbeit	0...100 %
10 000	Mittlerer Schaden, Verlust von mehreren Tagen Arbeit	0...100 %
100 000	Hoher finanzieller Schaden	0...100 %
1 000 000	Sehr hoher finanzieller Schaden	0...100 %
10 000 000	Personenschaden (Smith und Simpson, 2005, S. 39)	0...100 %

Tabelle 15: Schadensklassen in CoBe

a. Insgesamt 100% über alle Klassen

Auftretenswahrscheinlichkeit	Beschreibung und Beispiele	Wertebereich: Fehleranteil[a]
0	Fehler tritt nie auf	0...100 %
0,125	Fehler tritt in Ausnahme- und Sonderfällen auf	0...100 %
0,25	Fehler tritt selten bei Verwendung auf, z.B. in einer Nebenfunktion oder in einer Hauptfunktion unter bestimmten Bedingungen	0...100 %
0,5	Fehler tritt bei typischer Verwendung auf, z.B. in einer Hauptfunktion	0...100 %
1,0	Fehler tritt sicher bei Verwendung auf, z.B. beim Starten der Software	0...100 %

Tabelle 16: Klassen der Auftretenswahrscheinlichkeit in CoBe

a. Insgesamt 100% über alle Klassen

Verwendungs-häufigkeit	Beschreibung und Beispiele	Eingabewert: Fehleranteil[a]
0	Fehler in nicht verwendetem Software-Teil	0...100 %
1	Fehler wird sofort korrigiert, Software wird einmal eingesetzt	0...100 %
10	Wenige Benutzer, Software wird 10 mal eingesetzt	0...100 %
100	Über 10 Benutzer oder lange Korrekturdauer	0...100 %
1000	Über 100 Benutzer und lange Korrekturdauer	0...100 %
10 000	Über 1000 Benutzer, lange Korrekturdauer, intensive Verwendung	0...100 %
100 000	Über 10 000 Benutzer, lange Korrekturdauer, intensive Verwendung	0...100 %
1 000 000	Über 100 000 Benutzer, lange Korrekturdauer, intensive Verwendung	0...100 %

Tabelle 17: Klassen der Verwendungshäufigkeit in CoBe

a. Insgesamt 100% über alle Klassen

Die Verwendung definiere ich nicht genauer, weil diese Definition vom Produkt abhängt. Stattdessen orientiere ich mich an der Kritikalitätsbewertung (Smith und Simpson, 2005), für die eine ähnlich allgemeine Definition verwendet wird. Fehler, die nie auftreten oder keinen Schaden verursachen, werden nicht gemeldet und verursachen darum auch keinen Wartungsaufwand.

Für die Verteilung auf den Schaden kann zusätzlich gewählt werden, ob und welche Schadensklasse welcher Fehlerschwere zugeordnet wird. Dies ist für Projekte sinnvoll, in denen die Fehlerschwere ausschließlich durch den Schaden, der durch den Fehler beim Einsatz verursacht werden kann, definiert ist.

6.4 Reviews im Modell

CoBe enthält Prüfungsmodelle für das Spezifikationsreview, das Entwurfsreview und das Codereview. Für alle Reviews gelten die gleichen Zusammenhänge, sie unterscheiden sich aber in der Quantifizierung.

6.4.1 Eingaben für Reviews

Die Eingaben für das Spezifikationsreview, das Entwurfsreview und das Codereview sind in Tabelle 18 beschrieben. Die Tabellen 20 und 21 fassen die Ausgaben zusammen. In CoBe wird berücksichtigt, dass zwischen einzelnen Sitzungen Zeit zur Vorbereitung und Organisation benötigt wird. Die daraus resultierende Dauer wird als Bruttodauer bezeichnet. Die Nettodauer dagegen enthält die Dauer ohne Verzögerun-

gen. Weil Mitarbeiter zu verschiedenen Zeitpunkten und für verschiedene Aktivitäten eingesetzt werden, wird ihre Anzahl für das Gesamtresultat nicht addiert.

Eingabeparameter	Wertebereich
Prüflingsüberdeckung s_{Review}	Anteil des Prüflings am Artefakt
Zahl der Gutachter G_{Review}	Anzahl
Kompetenz der Gutachter KP_{Review}	Siebenstufige Skala von "extrem niedrig" bis "extrem hoch"
Vorbereitungsintensität der Gutachter v_{Review}	Vorbereitungsrate in Stunden pro Seite des Prüflings
Prüfung wiederverwendeter Software $\boldsymbol{WV}$	Prüfung nur von neuer Software oder Prüfung einschl. wiederverwendeter Software

Tabelle 18: Entscheidungsparameter für Reviews

6.4.2 Zusammenhänge im Reviewmodell

Abbildung 29 zeigt, welche Eingaben die Fehlerentdeckung beeinflussen.

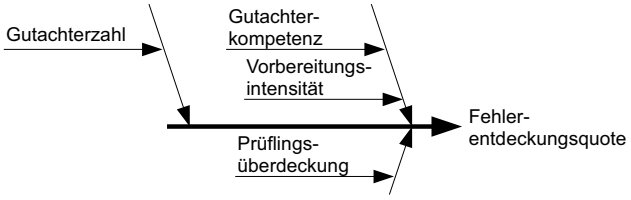

Abb. 29: Ursache-Wirkungs-Diagramm der Fehlerentdeckung

Gutachterzahl. Der Zusammenhang zwischen der Gutachterzahl und der Fehlerentdeckungsquote basiert in CoBe auf einem Modell, das sich an Biffl (2001) anlehnt: Jeder Gutachter entdeckt einen bestimmten Anteil q_{r} der Fehler. Das bedeutet, dass jeder weitere Gutachter einen Teil derjenigen Fehlern, die von keinem anderen Gutachter entdeckt wurden, entdeckt. Abbildung 30 skizziert den Zusammenhang, der sich daraus ergibt: Die Fehlerentdeckungsquote steigt mit der Gutachterzahl. Je mehr Gutachter teilnehmen, desto geringer steigt die Fehlerentdeckungsquote durch einen weiteren Gutachter. Biffl (2001) beschreibt dies durch die Wahrscheinlichkeit, mit der ein Gutachter einen bestimmten Fehler entdeckt.

In CoBe wird die Fehlerentdeckungsquote durch die Gutachterzahl G_{Review}, durch den Fehleranteil q_{r}, den ein Gutachter entdeckt, und durch den Parameter r_{qr} berechnet. Die Form der Gleichung ist so gewählt, dass sie umgeformt und dann mit linearer Regression quantifiziert werden kann:

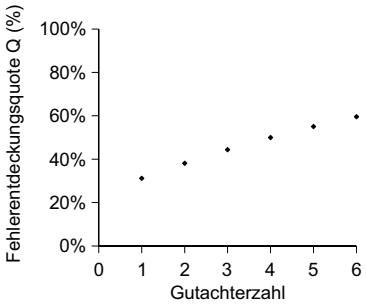

Abb. 30: Gutachterzahl und Fehlerentdeckung (Skizze)

$$Q_{Review} = 1 - r_{\mathrm{qr}}(1 - q_{\mathrm{r}})^{G_{Review}} \Leftrightarrow 1 - Q_{Review} = r_{\mathrm{qr}}(1 - q_{\mathrm{r}})^{G_{Review}}$$

Fehlerart. Spezifikations-, Entwurfs- und Codereview unterscheiden sich in der Fehlerentdeckungsquote für die unterschiedlichen Fehlerarten. Beispielsweise entdeckt das Entwurfsreview vor allem Entwurfsfehler und nur einen geringen Anteil Spezifikationsfehler. Der Faktor $ff_{Review,Art}$ passt die Fehlerentdeckungsquote eines Reviews an verschiedene Fehlerarten an.

Fehlerschwere. Reviews können sich in der Fehlerentdeckungsquote für die Fehlerschwere unterscheiden. Der Faktor $ff_{Review,Schwere}$ passt die Fehlerentdeckungsquote eines Reviews für unterschiedlich schwere Fehler an.

Vorbereitungsintensität. Die Vorbereitungsintensität wird im Modell durch die Vorbereitungsrate v_{Review} in Seiten pro Stunde dargestellt. Der Einfluss der Vorbereitungsintensität wird durch die Funktion $f_{\mathrm{Vorbereitung}}(v_{Review})$ beschrieben (Abbildung 31). Sie basiert auf den folgenden Zusammenhängen: Ein gewisser Mindestaufwand muss von einem Gutachter investiert werden, damit der Gutachter überhaupt Fehler entdeckt. Ich bezeichne diesen Punkt als negative Effektgrenze. Dann steigt die Fehlerentdeckungsquote mit dem Vorbereitungsaufwand über einen typischen Wert hinaus, bis sich die Fehlerentdeckungsquote stabilisiert, weil auch mit noch mehr Aufwand kaum noch mehr Fehler entdeckt werden; im Modell werden keine weiteren Fehler mehr entdeckt. Diesen Punkt bezeichne ich als positive Effektgrenze. Beispielsweise werden für die Spezifikation etwa 10 Seiten pro Stunde als Normalfall angegeben (Abschnitt 6.8.2). Bis 5 Seiten pro Stunde werden noch mehr Fehler entdeckt, mit einer langsameren Vorbereitung als 5 Seiten pro Stunde aber nicht mehr. Ab 35 Seiten pro Stunde werden keine Fehler entdeckt, weil der Gutachter das Dokument überfliegt. Der Einfluss wird abhängig von der durch den Normalfall normierten Vorbereitungsintensität berechnet (Abbildung 31).

Gutachterkompetenz. Die Kompetenz wird angelehnt an Boehm (2000) auf einer Ordinalskala mit 7 Kompetenzklassen dargestellt. Jeder Klasse ist ein Wert für den Faktor $ff_{Review,KP}$ zugeordnet. Hochkompetente Gutachter finden beispielsweise etwa 20% mehr Fehler als der Durchschnitt, der Faktor ist dann 1,2.

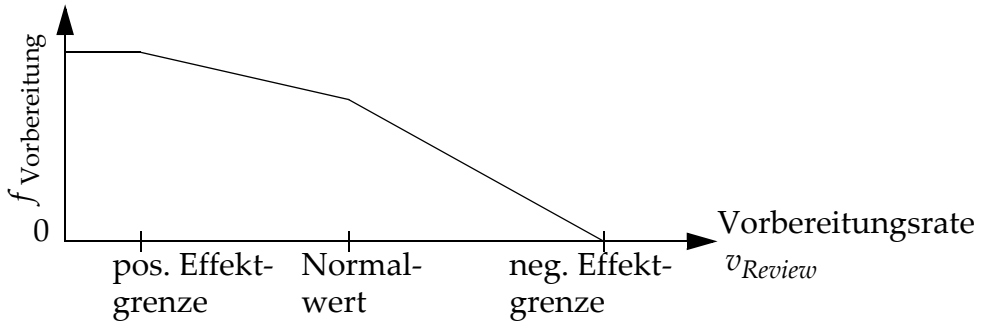

Abb. 31: Funktion für die Vorbereitungsintensität (Skizze)

Prüflingsüberdeckung. Fehler können nur im geprüften Teil eines Artefakts entdeckt werden, abhängig vom geprüften Umfangsanteil s_{Review} des Prüflings. Die Fehler, die im geprüften Umfangsteil enthalten sind, bezeichne ich als entdeckbare Fehler. Erfolgen die Reviews ohne Priorisierung, dann ist der Zusammenhang zwischen Umfangsanteil und den entdeckbaren Fehlern linear. Mit Priorisierung werden die kritischen Teile bevorzugt geprüft. Diese Teile enthalten überproportional viele Fehler. Abbildung 32 illustriert diesen Zusammenhang zwischen dem geprüften Umfang und dem Anteil der entdeckbaren Fehler. Werden 100 % beispielsweise des Codes geprüft, dann können darin 100 % derjenigen Fehler entdeckt werden, die prinzipiell mit der Prüfung entdeckbar sind. Werden 20 % des Codes geprüft, die als kritisch angesehen werden, dann können rund 40 % dieser Fehler entdeckt werden. Der Zusammenhang zwischen Umfang und entdeckbaren Fehlern ist also nicht-linear und wird durch die Funktion $f_{\text{Priorisierung}}$ zwischen Fehlerentdeckungsquote und der Prüflingsüberdeckung s_{Review} modelliert:

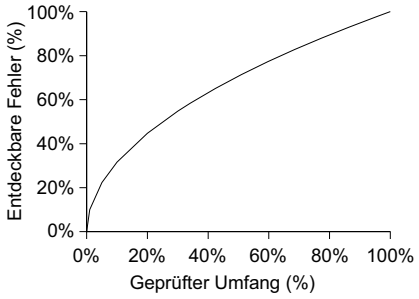

Abb. 32: Geprüfter Umfang und entdeckbare Fehler (Skizze)

$$f_{\text{Priorisierung}}(Schwere, s_{Review}) = s_{Review}^{\,r_{s,\,Schwere}}$$

Der Exponent $r_{s,Schwere}$ beschreibt, wie stark sich Fehler im kritischen Teil konzentrieren. Fehler werden nur dann in wiederverwendeter Software entdeckt, wenn diese auch geprüft wird.

Der Exponent liegt zwischen 0 und 1. Bei dieser Funktion wird die Steigung unendlich, wenn der Umfang nahe Null geht. In diesem Grenzbereich wird nur ein sehr kleiner Bruchteil des Dokuments geprüft, beispielsweise das erste Wort des Dokuments. Es ist aber nicht realistisch, in einem Review ausschließlich das erste Wort zu prüfen. Darum gehe ich davon aus, dass immer ein gewisser Mindestumfang geprüft wird und somit der Bereich nahe Null keine Rolle spielt.

Die Fehlerentdeckungsquote berechnet sich somit aus:

$$
\begin{aligned}
Q_{Review,\,Art,\,Schwere} &= ff_{Review,\,Art} \cdot ff_{Review,\,Schwere} \cdot ff_{Review,\,KP} \\
&\cdot f_{\text{Priorisierung}}(Schwere, s_{Review}) \\
&\cdot f_{\text{Vorbereitung}}(v_{Review}) \\
&\cdot (1 - r_{\text{qr}}(1 - q_{\text{r}})^{G_{Review}})
\end{aligned}
$$

Kosten und Umfang. Spezifikations-, Entwurfs- und Codeumfang werden aus dem Umfang in Function Points abgeleitet. Der Aufwand für die Vorbereitung wird über den Prüflingsumfang, die Zahl der Gutachter und ihre Vorbereitungsrate berechnet. Die Gutachterkompetenz beeinflusst den Normalwert der Vorbereitungsintensität, in dem jeder Kompetenzklasse ein Wert für einen Einflussfaktor zugeordnet ist; die Eingabe für die Vorbereitungsrate wird nicht verändert, weil die Eingabe messbar sein soll. Bei hoher Kompetenz können beispielsweise 13 Seiten pro Stunde gründlich geprüft werden, bei normaler Kompetenz 10 Seiten. Der Durchsatz in Sitzungen wird durch einen Parameter in Seiten pro Stunde oder Anweisungen pro Stunde dargestellt. Aufwand und Nettodauer für Sitzungen berechnen sich direkt aus diesem Sitzungsdurchsatz, dem Prüflingsumfang und der Teilnehmerzahl. Die Teilnehmerzahl ergibt sich aus der Zahl der Gutachter, einem Moderator, einem Autor und einem Protokollführer. Die Bruttodauer enthält den Abstand zwischen Sitzungen, weil dafür der Prüfling so aufgeteilt wird, dass einzelne Sitzungen höchstens zwei Stunden dauern. Der Abstand zwischen zwei Sitzungen beträgt drei Tage (Drappa, 1998). Zusammenhänge eines zu umfangreichen Prüflings, zu langer Sitzungen und den Einfluss des Moderators (Hampp, 2001) modelliere ich nicht, weil für diese detaillierten Planungen der Moderator zuständig ist. Hilfsmittel und ihr Einfluss sind nicht im Modell enthalten, weil die Studien keinen klaren Unterschiede zeigen.

Codereviews von Korrekturen. Für das Review einer Änderung gelten die gleichen Zusammenhänge wie für das Codereview. In einem Review einer Änderung wird nur ein kleiner Ausschnitt des Codes betrachtet, nämlich die Änderung und der unmittelbar damit zusammenhängende Code.

6.5 Automatische statische Codeanalyse

Die Codeanalyse ist in CoBe durch ihre Fehlerentdeckungsquoten, den Vorbereitungsaufwand und die Dauer der Durchführung modelliert: Der Vorbereitungsaufwand zur Einbindung und Konfiguration des Werkzeugs in der individuellen Entwicklungsumgebung ist in CoBe konstant, der Durchführungsaufwand wird vernachlässigt. Als Parameter kann die Dauer der Durchführung angegeben werden, damit rechenintensive Verfahren für umfangreichen Code von CoBe dargestellt werden können.

Prüfparameter werden nicht modelliert, stattdessen wird die Fehlerentdeckungsquote festgelegt und nur die Eingabe, ob die Codeanalyse durchgeführt wird, angegeben. Diese einfache Modellierung wird gewählt, weil sich die Auswirkungen von Prüfparametern der Codeanalyse in CoBe aus den folgenden Gründen nicht darstellen lassen: Da die Prüfparameter die möglichen Entscheidungen über die Codeanalyse darstellen, müssten diese Entscheidungen modelliert werden. Diese Entscheidungen sind die Auswahl des Werkzeugs und die eingesetzten Analysen. Diese Entscheidungen wirken, indem unterschiedliche Fehlerarten unterschiedlich gut entdeckt werden. Dazu gehört beispielsweise, ob die Verwendung von Nullpointern entdeckt wird, oder welche Konstrukte als gefährlich eingestuft werden. Diese Unterschiede sind in CoBe nicht sichtbar, da die Fehlerart über die Fehlerentstehung definiert ist und somit nicht erlaubt, die Verwendung von Nullpointern oder die Verwendung bestimmter Konstrukte darzustellen. Für eine solche detaillierte Darstellung der Fehler fehlen aber auch empirische Daten.

6.6 Tests im Modell

CoBe enthält Prüfungsmodelle für den Modultest, Subsystem- und Systemintegrationstest und den Systemtest. Diese Prüfungsmodelle sind durch die gleichen Zusammenhänge, aber mit unterschiedlicher Quantifizierung beschrieben.

6.6.1 Eingaben von Tests

Die Eingaben für die Testmodelle sind in Tabelle 19 dargestellt. Für den Black-Box-Test kann die Vollständigkeit der Testtechniken eingegeben werden, beispielsweise wie viele der Funktionen durch Testfälle abgedeckt werden. Als zusätzliche Eingabe kann der verfügbare Aufwand für den Black-Box-Test eingegeben werden, da Aufwand im Vergleich zur Zahl entdeckter Fehler als Testendekriterium verwendet wird (Kan, 2003; Stark et al., 1994). Dies wird in CoBe als Brute-Force-Test bezeichnet. Für den Glass-Box-Test werden messbare und gebräuchliche Überdeckungen modelliert. Aus den Varianten der Bedingungsüberdeckung wähle ich die Termüberdeckung (MC/DC), weil sie in RTCA (1992) gefordert wird. Als Schleifenüberdeckung wähle ich den Boundary-Interior-Test, der präzise definiert ist (Liggesmeyer, 2002). Ich unterscheide zwischen Überdeckungskriterium als zu erreichende und Überde-

ckungsgrad als erreichte Überdeckung. Die Tabellen 20 und 21 fassen die Modellresultate zusammen.

Eingabeparameter	Wertebereich
Black-Box-Test mit Funktionsabdeckung, mit Äquivalenzklassen, mit Sonderfällen	Prozentwerte für die Abdeckung der einzelnen Testtechniken
Zusätzlicher Aufwand für weitere Testfälle (Brute-Force-Test)	Aufwand in Entwicklerstunden
Vorbereitungszeitpunkt des Black-Box-Tests (nur für Systemtest)	Entwurfsphase oder Testphase
Glass-Box-Test (ergänzend) mit Kriterien für Anweisungs-, Zweig-, Bedingungs- und Schleifenüberdeckung	Prozentwerte für die Überdeckung der einzelnen Einheiten
Kompetenz der Tester	Siebenstufige Skala von “extrem niedrig” bis “extrem hoch”
Wiederholung des Tests	Ohne, gezielt oder vollständig
Prüfung wiederverwendeter Software	Neue Software oder gesamte Software

Tabelle 19: Entscheidungsparameter für den Systemtest

6.6.2 Zusammenhänge im Testmodell

Abbildung 33 zeigt die Eingaben und ihren Einfluss auf die Fehlerentdeckung.

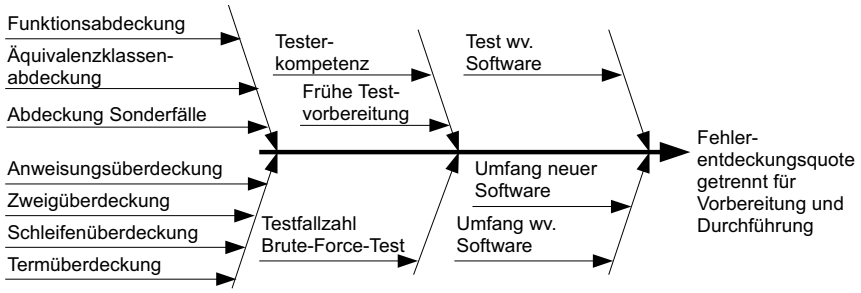

Abb. 33: Ursache-Wirkungs-Diagramm für Fehlerentdeckung durch Test

Die Zahl der Testfälle bildet die Grundlage für das Testmodell (Abbildung 34), weil Testfälle die Fehlerentdeckung und den Aufwand prägen (Abschnitt 5.5).

Testfälle werden im Modell durch die Testfallzahl T auf der Rationalskala dargestellt. Die Testfallzahl ist nach oben nicht beschränkt.

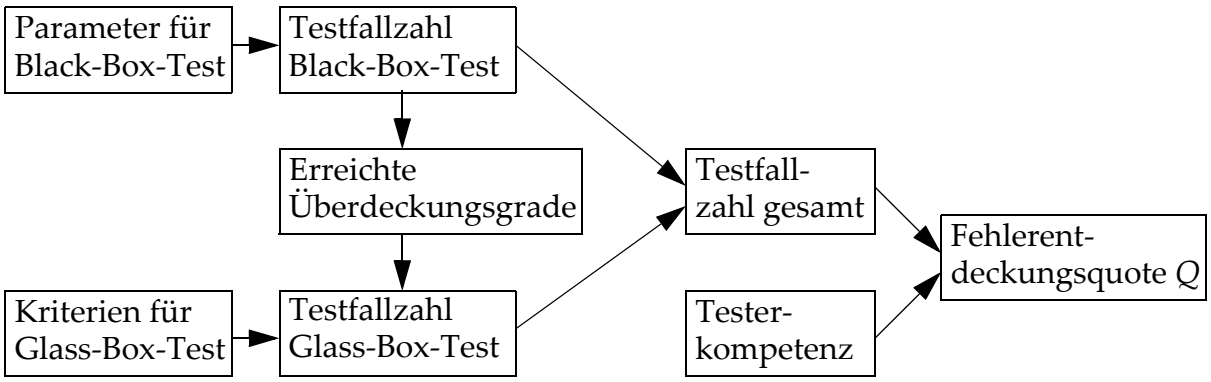

Abb. 34: Überblick über das Testmodell

Einzelne Testfälle sind in CoBe bezüglich der Fehlerentdeckung nicht unterscheidbar, weil sich nicht am Testfall erkennen lässt, mit welcher Testtechnik der Testfall entstanden ist; ein Testfall besteht ausschließlich aus Testeingaben und Sollresultaten. Alle Testfälle werden als gleichwertig betrachtet, obwohl Studien zeigen, dass der Black-Box-Test andere Fehlerarten als der Glass-Box-Test entdeckt. Die Studien zeigen aber widersprüchliche Ergebnisse (Juristo et al., 2004).

Testfälle im Black-Box-Test. Für einen vollständigen Black-Box-Test und zur Normierung wird in CoBe die nominale Zahl der Testfälle T_n berechnet. Sie ist durch den Umfang S_{FP} in Function Points bestimmt, unterschieden nach neuer und wiederverwendeter Software. Die nominale Testfallzahl T_n beschreibt, wie viele Testfälle für einen Black-Box-Test durchgeführt werden, bei dem alle Testtechniken vollständig angewendet werden. Die Parameter r_{0t} und r_{1t} quantifizieren den Zusammenhang: $T_n = r_{0t} \cdot S^{r_{1t}}$. Diese Modellierung setzt voraus, dass Function Points für die Software geeignet sind; in anderen Fällen können Function Points aber als interne Rechengröße verwendet werden, aus der der Code-Umfang berechnet wird (Abschnitt 6.3.1). Wird eine andere Variante der Function Points verwendet, muss der Zusammenhang neu quantifiziert, zumindest überprüft werden.

Den Testtechniken des Black-Box-Tests wird jeweils die gleiche Zahl an Testfällen zugeordnet. T bezeichnet im Modell die Testfallzahl, die sich linear aus der Abdeckung der Testtechniken berechnet. Werden beispielsweise alle Funktionen abgedeckt, dann wird ein Drittel der Testfälle T_n vorbereitet und durchgeführt. Diese Testfallzahl T wird mit T_n zur relativen Testfallzahl t normiert. Dadurch wird der Einfluss des Umfangs herausgerechnet. Grundlage ist also die normierte Testfallzahl t.

Testfälle im Glass-Box-Test. Die Zahl der Testfälle, die im Glass-Box-Test definiert und durchgeführt werden, hängt in CoBe vom Überdeckungsgrad ab, der durch den Black-Box-Test erreicht wurde, und vom Überdeckungskriterium, das für den Glass-Box-Test gefordert wird. Damit die Zahl der Testfälle im Glass-Box-Test berechnet werden kann, muss der Zusammenhang zwischen Testfallzahl einerseits und Überde-

ckungsgrad oder -kriterium andererseits definiert werden. Qualitativ gehe ich davon aus, dass im Mittel jeder Testfall die Überdeckung erhöht, solange noch nicht die vollständige Überdeckung erreicht ist. Denkbar sind verschiedene quantitative Zusammenhänge zwischen Testfallzahl und Überdeckungsgraden:

- Mit einem linearen Zusammenhang überdeckt jeder Testfall konstant viele Einheiten. Dies wird aber durch Erfahrungen widerlegt. Grady (1992) berichtet beispielsweise, dass etwa 80 % Anweisungsüberdeckung leicht zu erreichen sind, dass es dann aber mühsam wird, die Überdeckung weiter zu steigern. Der Umfang neuen Codes, den ein Testfall ausführt, nimmt typisch ab, je mehr Testfälle bereits ausgeführt wurden.
- Legt man dem Modell die Annahme zu Grunde, dass jeder Testfall den gleichen Anteil noch nicht ausgeführter Einheiten überdeckt, führt dies zu einem Zusammenhang zwischen der Überdeckung c und der Testfallzahl t und dem Anteil q, der pro Testfall überdeckt wird, mit der Form $c = 1 - (1 - q_t)^t$. Damit kann aber nicht dargestellt werden, dass 100 % Überdeckung erreicht werden oder erreicht werden sollen. In der Realität kann aber im Glass-Box-Test 100 % Überdeckung gefordert werden. Es sind auch Situationen vorstellbar, in denen 100 % Überdeckung erreicht werden, aber weitere Testfälle durchgeführt werden.

Darum wähle ich als Näherung einen Zusammenhang zwischen dem Überdeckungsgrad c und der normierten Testfallzahl t, der durch die Parameter r_{0c} und r_{1c} quantifiziert ist: $c = \min(1, r_{0c} \cdot t^{r_{1c}})$.

Geht die normierte Testfallzahl gegen Null, dann wird die Steigung dieser Funktion unendlich. Dieser Fall spielt aber in der Realität keine Rolle, da ein Testfall nicht teilbar ist und somit auch die normierte Testfallzahl nicht gegen Null gehen kann. Sie ist entweder Null oder hat eine bestimmte Mindestgröße.

Da die Überdeckung in der Realität nie über 100 % wachsen kann, verhindert die Minimumfunktion, dass c im Modell über 100 % wächst, wenn entsprechend viele Testfälle durchgeführt werden. Abbildung 35 zeigt skizzenhaft den Zusammenhang zwischen der normierten Testfallzahl und der erreichten Überdeckung: Die Überdeckung steigt mit steigender Testfallzahl, bis 100 % erreicht sind, und bleibt dann konstant.

Anweisungen, Zweige, Schleifen und Terme werden bei gleicher Testfallzahl in unterschiedlichem Ausmaß überdeckt. Angelehnt an Malaiya et al. (1994) wird in CoBe angenommen, dass es einen linearen statistischen Zusammenhang zwischen den unterschiedlichen Überdeckungsgraden gibt. Dieser Zusammenhang ist eingeschränkt auf einen bestimmten Bereich, für den gilt, dass die Überdeckung nicht vollständig ist und nicht gezielt angestrebt wird. Abbildung 36 zeigt diesen linearen Zusammenhang mit *c0* für Anweisungsüberdeckung, *c1* für Zweigüberdeckung, *c3* für Termüberdeckung und *c4* für Schleifenüberdeckung[1]. Solange die Anweisungs-

1. Die Nummerierung c_0 bis c_4 orientiert sich an Sneed und Winter (2002)

überdeckung unter 100 % liegt, wachsen Zweig-, Term- und Schleifenüberdeckung linear mit der Anweisungsüberdeckung.

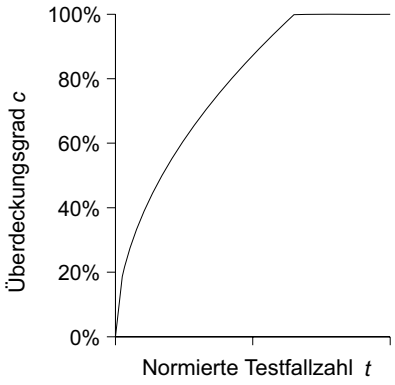

Abb. 35: Normierte Testfallzahl und Überdeckungsgrad

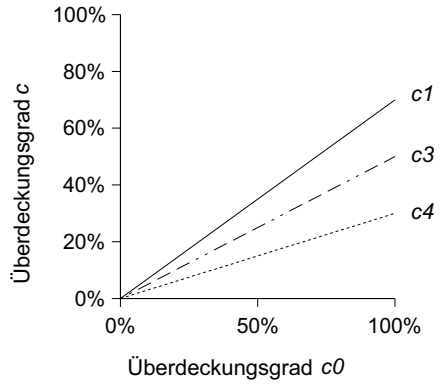

Abb. 36: Anweisungsüberdeckung und andere Überdeckungsgrade

Abbildung 37 zeigt den Zusammenhang zwischen der normierten Testfallzahl und den Überdeckungsgraden. Die Überdeckungsgrade steigen bei niedriger normierter Testfallzahl an (links im Diagramm). Erreicht die Anweisungsüberdeckung *c0* 100 %, dann bleibt sie konstant (etwa im rechten Drittel des Diagramms). Die Zweigüberdeckung *c1* steigt weiter an – solange, bis 100 % Zweigüberdeckung erreicht werden.

Somit wird der Zusammenhang zwischen normierter Testfallzahl und erreichtem Überdeckungsgrad durch die folgenden Gleichungen beschrieben:

- Anweisungsüberdeckung $c_0 = \min(1, r_{0c} \cdot t^{r_{1c}})$
- Zweigüberdeckung $c_1 = \min(1, cf_1 \cdot r_{0c} \cdot t^{r_{1c}})$
- Termüberdeckung $c_3 = \min(1, cf_3 \cdot r_{0c} \cdot t^{r_{1c}})$
- Schleifenüberdeckung $c_4 = \min(1, cf_4 \cdot r_{0c} \cdot t^{r_{1c}})$

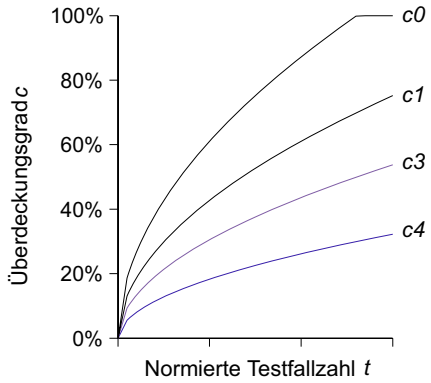

Abb. 37: Normierte Testfallzahl und verschiedene Überdeckungsgrade

Im Glass-Box-Test sind die Überdeckungskriterien vorgegeben, beispielsweise dass 80 % der Anweisungen überdeckt werden müssen. In CoBe wird aus diesen Kriterien die Zahl der Testfälle berechnet, die benötigt werden, um das Kriterium zu erfüllen. Dazu wird der Zusammenhang zwischen Überdeckung und Testfallzahl nach der Testfallzahl aufgelöst.

Anweisungen, die bereits durch den Black-Box-Test überdeckt wurden, müssen nicht mehr im Glass-Box-Test überdeckt werden. Wird also eine bestimme Anweisungsüberdeckung gefordert, dann müssen nur Testfälle definiert und durchgeführt werden, mit denen noch nicht überdeckte Anweisungen ausgeführt werden. Auch die verschiedenen Überdeckungskriterien beeinflussen sich gegenseitig: Testfälle, die durchgeführt werden, um Anweisungen zu überdecken, überdecken auch Zweige. Mit einer bestimmten Anweisungsüberdeckung wird also auch eine bestimmte Zweigüberdeckung erreicht. Wird dann eine bestimmte Zweigüberdeckung gefordert, dann müssen Testfälle für bereits überdeckte Zweige nicht mehr durchgeführt werden. Den gleichen Zusammenhang nehme ich für den Einfluss der Zweigüberdeckung auf die Termüberdeckung und auf die Schleifenüberdeckung an. Diesen Zusammenhang modelliere ich in CoBe wieder durch den statistischen, linearen Zusammenhang zwischen den Überdeckungsgraden aus Abbildung 36. Im Modell wird also ein statistischer Zusammenhang zwischen den Überdeckungskriterien verwendet. Dabei werden beispielsweise Effekte durch nicht erreichbaren Code vernachlässigt.

Die Testfälle im Glass-Box-Test werden gezielt so definiert, dass bestimmte Code-Einheiten überdeckt werden, während im Black-Box-Test der Code nicht sichtbar ist. Trotzdem lege ich den gleichen Zusammenhang zwischen den Testfällen und der Code-Überdeckung zu Grunde, der auf einem Sättigungseffekt beruht:

- Ist bereits eine hohe Überdeckung erreicht, dann können mit einem Testfall nur ganz gezielt einige wenige weitere Einheiten überdeckt werden. Bei niedriger Überdeckung werden mit diesem Testfall nicht nur diese wenigen Einheiten über-

deckt, sondern auch weitere, die auf dem Pfad liegen, der durch diesen Testfall ausgeführt wird.

- Zusätzlich spielt eine Rolle, dass die Testfälle nicht mechanisch aus den zu überdeckenden Code-Einheiten bestimmt werden können. Wie im Black-Box-Test gehe ich davon aus, dass es schwieriger wird, einen Testfall so zu definieren, dass bestimmte und dass möglichst viele Code-Einheiten überdeckt werden, wenn bereits viele Einheiten überdeckt sind.

Darum verwende ich im Glass-Box-Test die gleiche Form des Zusammenhangs wie im Black-Box-Test. Weil aber der Code sichtbar ist und weil der Tester versucht, gezielt noch nicht geprüfte Bereiche zu überdecken, modelliere ich, dass der Überdeckungsgrad stärker als im Black-Box-Test wächst. Ein Testfall im Glass-Box-Test trägt also mehr zur Überdeckung bei als ein Testfall im Black-Box-Test.

Fehlerentdeckung. Um die Fehlerentdeckung im Test zu modellieren, nehme ich an, dass ein einzelner Testfall einen Fehler mit bestimmter Wahrscheinlichkeit entdeckt. Es handelt sich also um einen Zufallseffekt, da die Testtechniken, mit denen Testfälle hergeleitet werden, Heuristiken sind (Abschnitt 5.5). In CoBe wird nicht mit der Wahrscheinlichkeit gerechnet, statt dessen wird mit Anteilen gerechnet. Jeder Testfall entdeckt also einen Teil der unentdeckten Fehler. In CoBe wird dies formal über den Anteil q_t der noch nicht entdeckten Fehler, die ein einzelner Testfall entdeckt, beschrieben. Die Fehlerentdeckungsquote eines Tests ergibt sich somit aus der Zahl der Testfälle, die durchgeführt werden; unabhängig davon, ob die Testfälle im Black-Box-Test oder im Glass-Box-Test definiert und durchgeführt werden.

Daraus folgt, dass die Zahl aller Testfälle zusammengezählt wird. Sie wird normiert, damit das Modell unabhängig vom Software-Umfang quantifiziert werden kann. Somit wird die Fehlerentdeckungsquote Q_{Test} aus der normierten Testfallzahl t_{Test} eines Tests berechnet.

Damit eine Quantifizierung mit linearer Regression möglich wird, wird der zusätzliche Faktor r_qt benötigt. Damit bei sehr niedriger normierter Testfallzahl keine negativen Resultate möglich sind, wird der Wertebereich durch die Maximalfunktion eingeschränkt. Die Formel, mit der die Fehlerentdeckungsquote berechnet wird, lautet damit:

$$Q_{Test} = \max(0, 1 - r_\mathrm{qt}(1 - q_\mathrm{t})^{t_{Test}})$$

Für die Quantifizierung durch lineare Regression wird die Gleichung umgeformt; die Beschränkung des Wertebereichs fällt weg[1]. Die Regression berechnet r_qt und q_t aus einzelnen Datenpunkten durch die Gleichung der Form $1 - Q_{Test} = r_\mathrm{qt}(1 - q_\mathrm{t})^{t_{Test}}$

Abbildung 38 skizziert diesen Zusammenhang zwischen normierter Testfallzahl und Fehlerentdeckungsquote. In einem kleinen Bereich links unten im Diagramm bleibt

1. Bei sehr wenigen Testfälle kann die Fehlerentdeckungsquote im Prinzip negativ werden, abhängig von den Datenpunkten, die bei der Regression verwendet werden. Dies wird in CoBe durch die Maximalfunktion verhindert, die Fehlerentdeckungsquote bleibt dann 0 %.

die Fehlerentdeckungsquote auf 0 %, auch wenn einige wenige Testfälle durchgeführt werden. Dann steigt die Fehlerentdeckungsquote steil an und flacht dann ab. Sie nähert sich asymptotisch der Fehlerentdeckungsquote von 100 % (rechts oben im Diagramm).

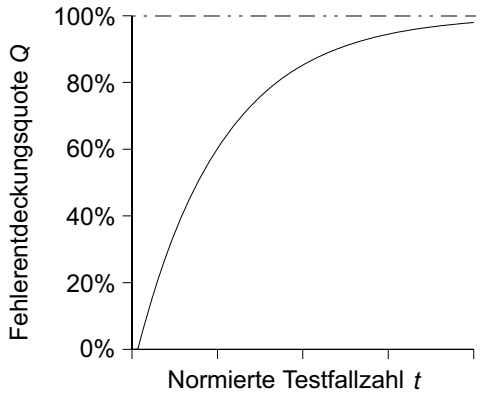

Abb. 38: Normierte Testfallzahl und Fehlerentdeckungsquote

Diese Fehlerentdeckungsquote wird für die unterschiedliche Entdeckung der Fehlerarten und -schwere in den verschiedenen Tests angepasst. Beispielsweise findet der Modultest keine Spezifikationsfehler, der Systemtest einen geringen Teil der Spezifikationsfehler, aber vor allem Entwurfs-und Codefehler. Dazu werden die Faktoren $ff_{Test,Art}$ für den Einfluss der Fehlerart und $ff_{Test,Schwere}$ für den Einfluss der Fehlerschwere verwendet.

Testerkompetenz. Die Kompetenz spielt eine Rolle, weil die Testmethoden Heuristiken bieten. Die Testerkompetenz KP_{Test} wird auf einer siebenstufigen Skala dargestellt, jeder Klasse der Skala ist ein Wert des Faktors $ff_{Test,KP}$ zugeordnet. Hochkompetente Tester finden etwa 20 % mehr Fehler als der Durchschnitt.

Die Fehlerentdeckungsquote für einen Test berechnet sich somit aus

$$Q_{Test,\,Art,\,Schwere} = ff_{Test,\,Art} \cdot ff_{Test,\,Schwere} \cdot ff_{Test,\,KP} \cdot \max(0,\, 1 - r_{\mathrm{qt}}(1 - q_{\mathrm{t}})^{t_{Test}})$$

Frühe Testvorbereitung. Eine frühe Vorbereitung ist nur für den Black-Box-Test möglich, weil die Testfälle im Glass-Box-Test abhängig von der erreichten Überdeckung definiert werden. Die Vorbereitung kann frühestens erfolgen, wenn die Vorgabe für den Test fertig ist. Für den Systemtest ist dies die Spezifikation, der Test kann also während des Entwurfs vorbereitet werden. In der Vorbereitung wird ein Teil der Fehler in der Spezifikation entdeckt. Diese Fehler werden vor der Codierung entfernt. In CoBe ist dieser Zusammenhang für Spezifikationsfehler des Systemtests dargestellt. Die durch frühe Testvorbereitung entfernten Fehler sind in den Prüfungen nach dem Entwurfsreview nicht mehr enthalten. Andere Tests können frühestens nach

dem Entwurf vorbereitet werden, darum ist für den Korrekturaufwand kein Unterschied modelliert.

Aufwand. Jeder Testfall kostet den gleichen Aufwand. Für den Black-Box-Test verteilt sich dieser Aufwand anteilig auf Vorbereitung, Testaufbau und Testdurchführung. Im Glass-Box-Test ist der Aufwand für den Testaufbau konstant. Vorbereitung und Durchführung lassen sich nicht trennen. Der Aufwand wird durch die Kompetenz der Tester und den Produktivitätsparameter *af* (Abschnitt 6.3.7) beeinflusst. Zu diesem Aufwand, der für alle Testfälle gleich ist, kommt Aufwand für blockierende Fehler dazu: Nachdem der blockierende Fehler korrigiert wurde, muss zumindest derjenige Testfall wiederholt werden, mit dem der Fehler entdeckt wurde. Dies entspricht einer gezielten Wiederholung für diesen Fehler, darum wird der gleiche Aufwand wie für die gezielte Testwiederholung berechnet (Abschnitt 6.3.6).

Brute-Force-Test. Beim Brute-Force-Test werden Testfälle mit den Testtechniken des Black-Box-Tests definiert, solange Aufwand verfügbar ist. Die Zahl der Testfälle wird also im Modell aus dem Aufwand berechnet. Sie wird zu den Testfällen im Black-Box-Test addiert.

6.7 Zusammenfassung

In diesem Abschnitt werden die Ursache-Wirkungs-Zusammenhänge in CoBe zusammengefasst und das Vorgehen zur Kalibrierung erläutert.

6.7.1 Zusammenhänge im Überblick

Abbildung 39 zeigt die Zusammenhänge, die die Fehlerzahlen betreffen, Abbildung 40 die Zusammenhänge für Kosten und Nutzen (entfallende Kosten).

Die Abbildungen stellen die anfallenden Fehlerkosten dar, die entfallenden Kosten werden mit den gleichen Zusammenhängen berechnet. Abbildung 39 zeigt beispielhaft das Prüfungsmodell für den Systemintegrationstest (oben links) und das Spezifikationsreview (darunter) mit ihren Eingaben. Beide Modelle berechnen die entsprechenden Fehlerentdeckungsquoten. In das Fehlerstrommodell (rechts unten) fließen die Zahlen für die entstehenden Fehler, die Fehlerentdeckungs- und Korrekturquoten und die Eingaben für den Prüfprozess. Das Resultat sind die Fehlerzahlen. Abbildung 40 zeigt, wie für diese beiden Prüfungen (links oben) Fehlerkosten für die Korrektur und Testwiederholung (links unten) und Fehlerfolgekosten (rechts unten) berechnet werden. Die Abbildung zeigt, wie die Aufwände durch den Aufwandseinfluss angepasst werden und wie daraus Dauer und Personalbedarf berechnet werden (rechts).

Die Zusammenhänge aus dem QS-Modell (Drappa, 1998) und aus COCOMO II (Boehm, 2000) sind markiert. Der Schwerpunkt in CoBe liegt, anders als im QS-Modell, auf den Prüfungsmodellen mit ihren Prüfparametern, Kosten für Prüfwiederholung, langfristigen Kosten der Wartung und des Einsatzes und der direkten Berechnung des Nutzens als entfallende Kosten. Dazu wird auch die Wartungs- und

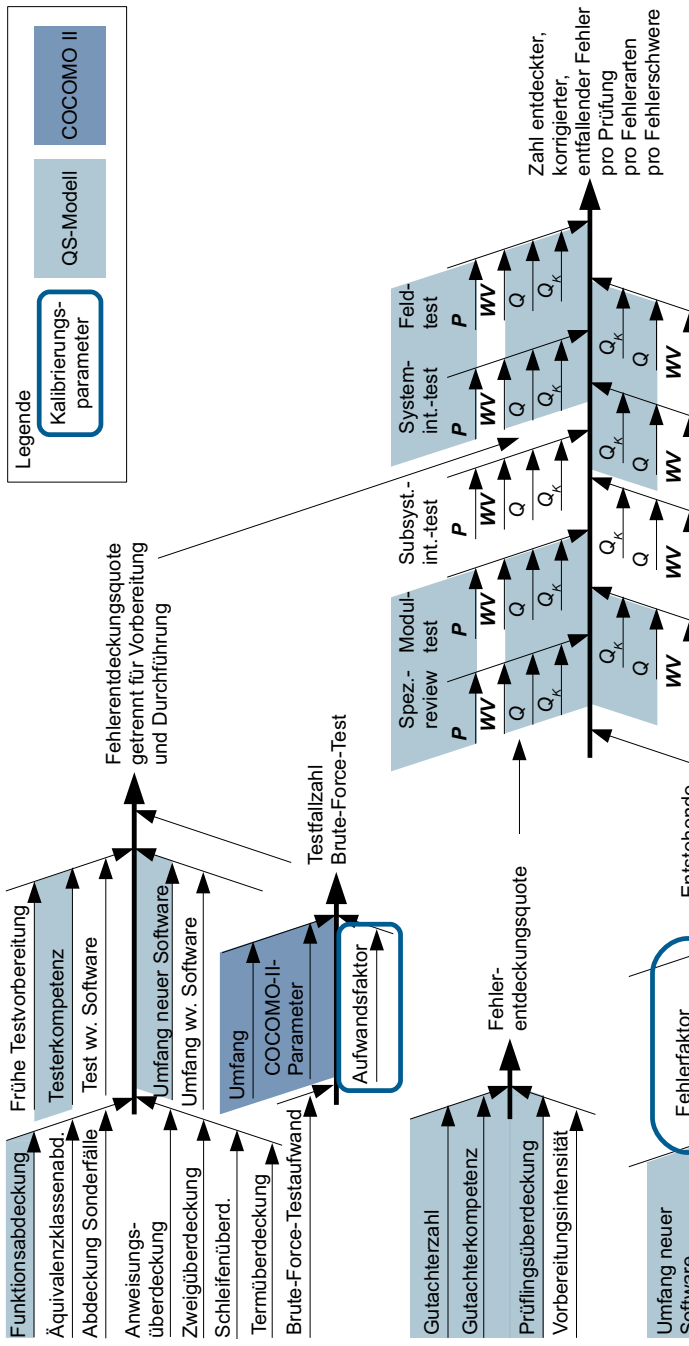

Abb. 39: Fehlerentstehung und -entdeckung in CoBe

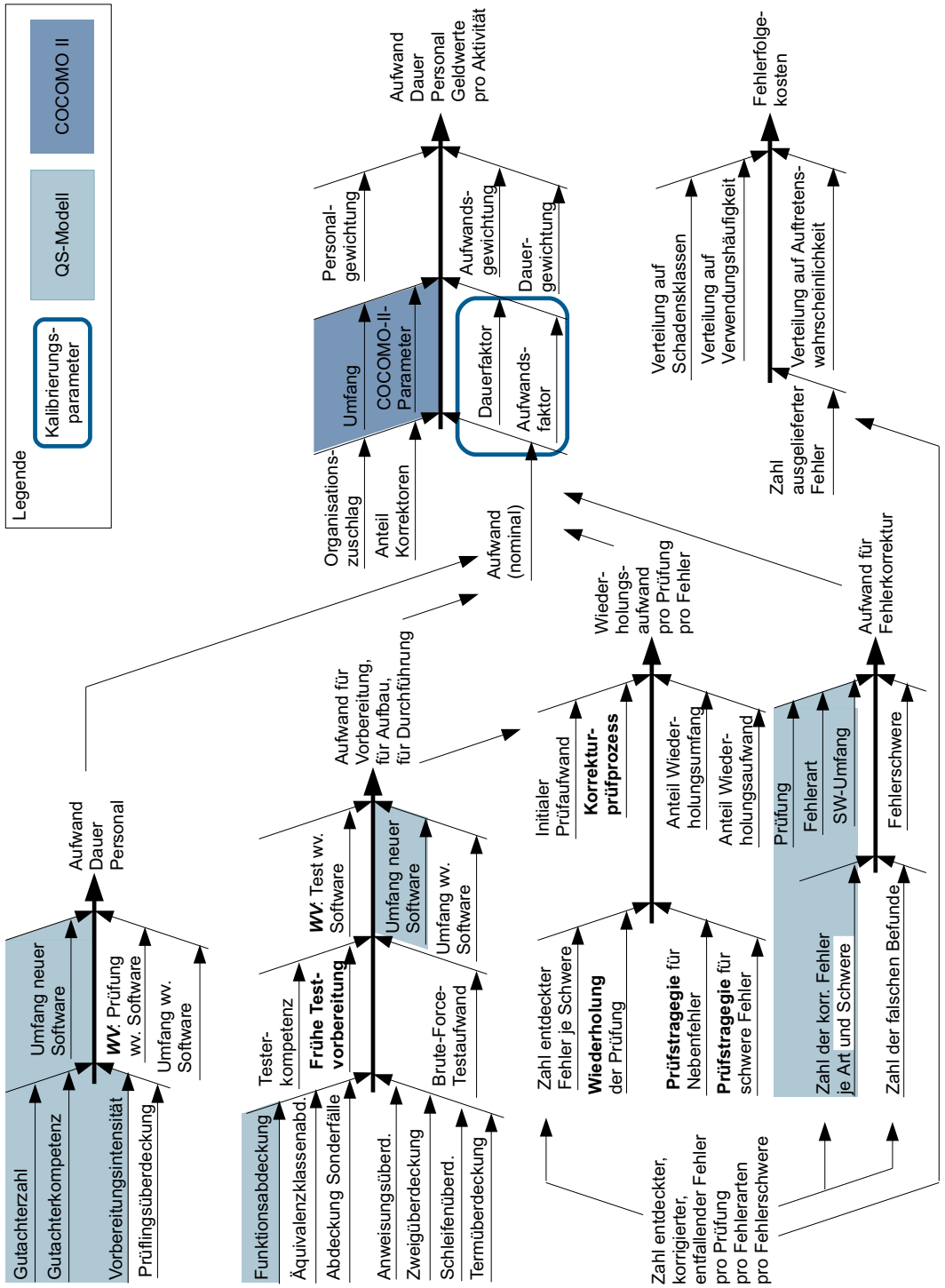

Abb. 40: Anfallende und entfallende Kosten in CoBe

Einsatzphase des Produkts betrachtet. CoBe unterscheidet sich zusätzlich vom QS-Modell, weil in CoBe unterschiedlich schwere Fehler und wiederverwendete Software dargestellt werden. Zusammenhänge aus COCOMO II sind in CoBe integriert, um Zusammenhänge für Dauer und Personal zu beschreiben. Mit COCOMO II werden auch Einflussparameter der Projektumgebung integriert, mit denen der Aufwand kalibriert wird. Zusätzlich werden in CoBe Kalibrierungsparameter bereitgestellt. Diese Kalibrierungsparameter greifen an wenigen Stellen im Modell; die wesentlichen Modellkomponenten für Prüfung, Fehlerentdeckung, Korrektur und Prüfwiederholung werden nicht geändert. Der Umfang wirkt an vielen Stellen und ist an diesen Stellen als Eingabe dargestellt, damit die Diagramme übersichtlicher sind.

Die Ausgaben sind in den Tabellen 20 und 21 dargestellt. Der Organisationsanteil wird auf den Aufwand aufgeschlagen. Für den Feldtest werden die Fehlerbehebungskosten dargestellt; sein Nutzen sind entfallende Wartungskosten und entfallende Fehlerfolgekosten. Kosten und Nutzen werden pro Prüfung schrittweise zusammengefasst: Die Kosten zu Prüfkosten und Fehlerbehebungskosten; entfallende Fehlerkosten zum Nutzen im Projekt und zum Nutzen in der Wartung. Alle Aufwands-, Dauer- und Personalausgaben werden in Geldwerte umgerechnet und als Geldwerte ausgegeben. Qualitätskosten (und Nutzen) im Projekt, in der Wartung und beim Einsatz werden zu den Gesamt-Qualitätskosten zusammengefasst, aber auch getrennt ausgegeben. Für die Projekt-Qualitätskosten werden die im Projekt anfallenden Kosten aufsummiert. Der Gesamtnutzen ist die Differenz zwischen Kosten und Nutzen (Hanusch, 1987).

Ausgabeparameter für Kosten	Ausgabewerte
Vorbereitung	Aufwand, Dauer, Mitarbeiter
Für Tests: Aufbau des Testgeschirrs	Aufwand, Dauer, Mitarbeiter
Durchführung (In Reviews: Sitzung)	Aufwand, Dauer (Netto und Brutto in Reviews), Mitarbeiter
	Summe Prüfkosten
Korrektur (einschließlich Analyse falscher Befunde)	Aufwand, Dauer, Mitarbeiter
Korrektur nach früher Testvorbereitung für Systemtest	Aufwand, Dauer, Mitarbeiter
Wiederholung der Prüfung	Aufwand, Dauer, Mitarbeiter
Wiederholung mit Prüfprozess: Wiederholung insgesamt nach der Prüfung und getrennt für einzelne Prüfungen	Aufwand, Dauer, Mitarbeiter
	Summe Fehlerbehebungskosten

Tabelle 20: Modellresultate für anfallende Kosten einer Prüfung

Ausgabeparameter für Nutzen	Ausgabewerte
Entfallende Korrektur einschließlich Analyse falscher Befunde in Folgeprüfungen	Jeweils Aufwand, Dauer, Mitarbeiter
Entfallende Kosten für blockierende Fehler im Integrations- und Systemtest	Jeweils Aufwand, Dauer, Mitarbeiter
Entfallende Kosten bei gezielter Testwiederholung in den folgenden Prüfungen	Jeweils Aufwand, Dauer, Mitarbeiter
Entfallende Kosten pro Prüfung, falls ein Prüfprozess in Folgeprüfungen durchgeführt wird	Jeweils Aufwand, Dauer, Mitarbeiter
	Summe entfallende Projektkosten
Entfallende Korrektur in der Wartung	Aufwand
Entfallende Testwiederholung für den Prüfprozess in der Wartung	Aufwand
	Summe entfallende Wartungskosten
Entfallende Fehlerfolgekosten	Geldwert

Tabelle 21: Modellresultate für den Nutzen

6.7.2 Vorgehen zur Kalibrierung

Die Kalibrierung des Modells erfolgt in fünf aufeinander folgenden Schritten. Zuerst wird der Umfangsfaktor für den Code gesetzt, bevor Aufwand und Dauer kalibriert werden. Dann werden die Fehlerzahl, die Fehlerverteilungen und schließlich die Umfangsfaktoren der Dokumente kalibriert. Die im Folgenden genannten und verwendeten Daten sind Archivdaten, also Daten aus abgeschlossenen Projekten:

- Umfang in Anweisungen oder in Function Points,
- Gesamtaufwand und -dauer des Projekts,
- COCOMO-II-Parameter,
- Fehlerzahlen,
- Verteilung der Fehler auf Fehlerart und Fehlerschwere,
- Umfang der Spezifikation und des Entwurfs in Seiten.

Kalibrierung des Umfangsfaktors für den Code. Für den Umfangsfaktor werden der Code-Umfang in Anweisungen und der Produktumfang in Function Points für neuen und hinzugefügten Code benötigt. Der Umfangsfaktor ist das Verhältnis dieser Größen. Ist der Umfang in Function Points nicht verfügbar, dann können Erfahrungswerte verwendet werden, die für verschiedene Programmiersprachen das Verhältnis

zwischen der Zahl der Anweisungen und den Function Points beschreiben (Boehm, 2000; Jones, 1996; QSM, 2009).

Kalibrierung des Aufwands und der Dauer. Der Aufwandsfaktor basiert auf der Aufwandsgleichung aus COCOMO II. Aus Archivdaten werden der Umfang in Anweisungen, die COCOMO-II-Parameter oder eine Einschätzung der Parameter und der Gesamtaufwand benötigt. Dann kann entweder der Produktivitätsfaktor von COCOMO II mit dem in Boehm (2000) vorgeschlagenen Regressionsverfahren angepasst werden; der Aufwandsfaktor ist dann das Verhältnis zwischen angepasstem und originalem Produktivitätsfaktor. Alternativ kann der Aufwandsfaktor als Verhältnis zwischen dem berechneten Gesamtaufwand und dem tatsächlichen, gemessenen Gesamtaufwand (dem Istwert) berechnet werden. Der Dauerfaktor wird mit dem gleichen Vorgehen berechnet, aber basierend auf Aufwand und Dauer der Projekte mit der COCOMO-II-Gleichung für die Dauer.

Kalibrierung der Gesamtfehlerzahl. In den Fällen, in denen Archivdaten für alle in Prüfungen und (für einen bestimmten Zeitraum) nach Auslieferung entdeckten Fehler vorhanden sind, wird mit CoBe aus dem Umfang der abgeschlossenen Projekte die zu erwartende Gesamtfehlerzahl berechnet. Der Fehlerfaktor ist das Verhältnis aus der gemessenen Zahl der Fehler (dem Istwert) und dem Modellresultat. Sind nur unvollständige Fehlerzahlen archiviert, beispielsweise nur für einen Teil der Prüfungen, dann ist es notwendig, zusätzlich zum Umfang den Prüfprozess und die Prüfparameter in CoBe zu setzen. Für ein Projekt wird dann die vorhandene Fehlerzahl berechnet. Der Fehlerfaktor ist das Verhältnis aus dem Istwert und dem Modellresultat.

Kalibrierung der Fehlerverteilung auf Fehlerarten und Fehlerschwere. Die Verteilung auf die Fehlerarten und auf die Fehlerschwere kann direkt aus archivierten Fehlerzahlen berechnet werden. Für die Fehlerarten ist wichtig, dass die Fehlerzahlen möglichst aus allen Prüfungen und aus der Wartung stammen. Fehlen zum Beispiel Zahlen aus den stattgefundenen Spezifikationsreviews, dann wird ein großer Teil der Spezifikationsfehler nicht erfasst. Die Verteilung auf die Fehlerarten ist dann durch zu wenig Spezifikationsfehler verzerrt.

Kalibrierung der Umfangsfaktoren für Dokumente. Die Umfangsfaktoren für Spezifikation und Entwurf berechnen sich wie der Umfangsfaktor für den Code, basierend aber auf dem Umfang in Seiten und dem Umfang in Function Points.

6.8 Quantifizierung

Die Quantifizierung des Basismodells erfolgt weitgehend mit Daten aus Jones (1996), die auch die Grundlage für das QS-Modell bilden, und mit Daten aus COCOMO II (Boehm, 2000). Die Zusammenhänge für die Prüfungsmodelle werden, soweit möglich, konsistent mit dieser Datensammlung quantifiziert. Ergänzend wird auf einzelne Untersuchungen zurückgegriffen.

6.8.1 Basismodell

Umfang. Der Umfang des Produkts wird zwischen Function Points und Anweisungen umgerechnet. Für die Spezifikation und den Entwurf wird der Umfang in Seiten abgeleitet (Tabelle 22).

Umfangsfaktor	Wert
Umfangsfaktor Code	Abhängig von der Programmiersprache (Boehm, 2000), 53 Anweisungen pro Function Point in Java
Umfangsfaktor Spezifikation	0,44 Seiten pro Function Point (Drappa, 1998)
Umfangsfaktor Entwurf	0,44 Seiten pro Function Point (Drappa, 1998)

Tabelle 22: Umrechungsfaktoren für den Umfang

Fehlerentstehung. Die Fehlerdichte *fd* wird mit den Daten für Auftragsprojekte abhängig vom Umfang in Function Points quantifiziert (Jones, 1996, S. 230) und beträgt für neue Software $fd = 1,05 + 1,11 \cdot \log_{10}(S_{\text{FPneu}})$. In wiederverwendeter Software ist die Fehlerdichte auf 5 % der Fehlerdichte neuer Software reduziert (Q_{WV} = 95%).

Die Verteilung auf die Fehlerart (Tabelle 23) erfolgt mit Daten aus Jones (1996). Handbuchfehler werden im Modell nicht dargestellt. Fehlerhafte Korrekturen werden auf andere Fehlerarten verteilt, da sie in CoBe bei der Korrektur berücksichtigt werden. Die Verteilung auf die Fehlerschwere (Tabelle 24) stammt aus mozilla (2007), da Jones seine eigenen Daten über die Fehlerschwere in Jones (1996, S. 384) kritisch diskutiert.

Fehlerart (%)	
Spezifikationsfehler	22 %
Entwurfsfehler	28 %
Codefehler	39 %

Tabelle 23: Verteilung auf die Fehlerarten aus Jones (1996)

Fehlerentdeckung. Die Fehlerentdeckung aus Jones (1996) wird für den Nominalfall aller Prüfungen mit typischen Prüfparametern übernommen (Tabelle 25).

Korrekturaufwand. Die Basis für den Korrekturaufwand eines Fehlers nach Prüfungen (Tabelle 26, Jones, 1996) wird mit dem Einfluss aus Boehm (1981) angepasst, so dass der Korrekturaufwand in kleinen Projekten um den Faktor 4 von früher zu später Fehlerentdeckung steigt, in großen Projekten um den Faktor 10. Ich wähle 100 Function Points als kleines Projekt und 10.000 Function Points als großes Projekt. Der Faktor für den Anstieg wird für andere Umfangswerte linear inter- und extrapoliert. Die Erfahrungswerte in Tabelle 26 unterscheiden sich deutlich. Für die Werte ist aber

Fehlerschwere (%)		
Blockierende Fehler	1,1 %	10,8 %
Kritische Fehler	9,7 %	
Hauptfehler	12,7 %	77,7 %
Normale Fehler	65,0 %	
Nebenfehler	8,6 %	11,5 %
Kosmetische Fehler	2,9 %	

Tabelle 24: Verteilung auf die Fehlerschwere (mozilla, 2007)

Fehlerentdeckungsquote aus Jones (1996)	Spezifikations-fehler	Entwurfs-fehler	Codefehler
Spezifikationsreview	40 %	15 %[a]	0 %
Entwurfsreview	15 %	55 %	0 %
Codereview	20 %	40 %	65 %
Modultest	0 %	5 %	20 %
Integrationstest	10 %	15 %	30 %
Systemtest	10 %	15 %	35 %
Feldtest	20 %	20 %	25 %

Tabelle 25: Fehlerentdeckung nach Jones (1996)

a. Im Spezifikationsreview in CoBe werden ausschließlich Spezifikationsfehler entdeckt

unsicher, ob die Testwiederholung enthalten ist; es ist unklar, ob es sich um Messungen oder Schätzungen handelt. Für die Wartung wähle ich den Wert der ISBSG (2005), weil dieser Wert aus Messungen in 54 Projekten unterschiedlicher Organisationen stammt.

Für den Einfluss der Fehlerart gibt es unterschiedliche Zahlen. Grady (1992) berichtet, dass ein Spezifikationsfehler den fünffachen Wartungsaufwand eines Codefehlers benötigt. Basili und Perricone (1984) nennen einen Faktor 1,5 zwischen einem Spezifikationsfehler und einem Entwurfs- oder Codefehler. Kan (2003) nennt ein Verhältnis von 1 : 0,7 zwischen der Korrektur von Entwurfsfehlern und Codefehlern in der Testphase. CoBe verwendet ein Verhältnis von 1,3 : 1 : 0,7 zwischen Spezifikations-, Entwurfs- und Codefehlern, orientiert an Kan (2003) und Basili und Perricone (1984).

Unterschiedlichen Einfluss der Fehlerschwere nennen Kan (2003) für den Test und Zage und Zage (2003). CoBe verwendet die Werte von Kan (2003).

Korrekturaufwand pro Fehler (Eh)	Jones (1996)	Kan (2003)	Erfahrungswerte
Spez.-review	1,0	-	-
Entwurfsreview	1,5	0,5	24 (Leszak et al., 2002); 2,5 (Rico, 2000)
Codereview	1,5	0,5	2,5 (Rico, 2000)
Modultest	2,5	3,0	
Integrations- und Funktionstest	5,0	3,5	13,5 (Mittel), 16 (Median) (Basili et al., 1996); 48 (Leszak et al., 2002)
Systemtest und Feldtest	10	3,8	13,5 (Mittel), 16 (Median) (Basili et al., 1996); 4 (Jalote, 2000); 25 (Rico, 2000)
Wartung	-	-	15 (ISBSG, 2005); 2,7 (Grady, 1992); 250 (Rico, 2000); 1, 2, 4 (Demirörs et al., 2000); 72 (Leszak et al., 2002); 29 (Shull et al., 2002); 17 (Jalote, 2000); 21, 28 (Sneed, 2004); 27 (Basili et al., 1996)

Tabelle 26: Korrekturaufwand pro Fehler nach Prüfungen

Aufwand (Eh)	Nebenfehler	Hauptfehler	krit. Fehler	Verhältnis
Kan (2003)	3,4	4,0	5,1	0,85 : 1,0 : 1,28
Zage und Zage (2003)	3,3	8,2	6,2	0,40 : 1,0 : 0,75

Tabelle 27: Fehlerschwere und Korrekturaufwand

Falsche Befunde. Die Quantifizierung des Aufwands für falsche Befunde stützt sich auf den Analyseaufwand von Fehlern, der in unterschiedlichen Studien genannt wird. Basili et al. (1996) haben Aufwände von Wartungstätigkeiten gemessen (Tabelle 28). Niessink und Van Vliet (1998), Evanco (2001), Rombach und Ulery (1989a und 1989b) nennen zwischen 15 % und 72 % für die Analyse der Korrektur. Für die Quantifizierung werden die Erfahrungswerte der Isolation in Tabelle 28 (Basili et al., 1996) verwendet und auf den Aufwand ohne Organisation (Analyse in Tabelle 28), Prüfung und Beratung bezogen. Somit werden in CoBe 28,8 % des Korrekturaufwands pro Fehler benötigt, um einen falschen Befund zu erkennen.

Testwiederholung. Für jeden blockierenden Fehler nehme ich an, dass 50 % der Testfälle wiederholt werden. Dazu sind, angelehnt an van Megen und Meyerhoff (1995), zusätzlich 25 % des Aufwands der ersten Testdurchführung notwendig. Die gezielte Testwiederholung wird mit den gleichen Werten quantifiziert wie blockierende Fehler. Für den Korrekturprüfprozess werden die gleichen Werte verwendet. Dies sind Annahmen, die an ein konkretes Testvorgehen angepasst werden müssen. Organisatorische Kosten werden mit 0 quantifiziert.

Tätigkeit und Beschreibung	Aufwandsanteil
Analyse: Problemmeldung, Entscheidung über Lösung	6 %
Isolation: Fehlerursache identifizieren	26 %
Entwurf der Lösung	26 %
Implementierung und Modultest der Lösung	38 %
Inspection, Certification, Consulting (Prüfungen, Beratung)	4 %

Tabelle 28: Aufwandsverteilung in der korrektiven Wartung (Basili et al., 1996)

6.8.2 Reviews

Gutachterzahl. Die Fehlerentdeckungsquote wird mit Daten aus Biffl (2001) für Spezifikationsreviews mit Checklisten quantifiziert (Tabelle 29). Die Parameter r_{qr} = 0,7653 und q_{r} = 0,1009 (Abschnitt 6.4.2) werden durch lineare Regression berechnet. Abbildung 41 zeigt den Zusammenhang für die Messwerte (4 bis 6 Gutachter) und für extrapolierte Werte mit weniger als 4 Gutachtern und mehr als 6 Gutachtern.

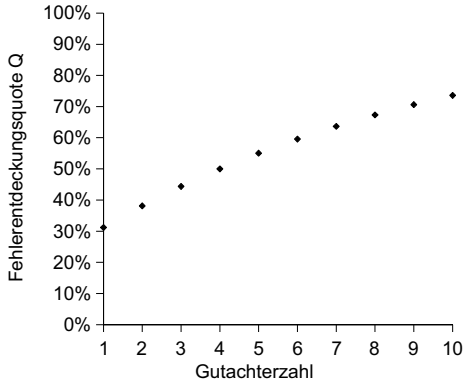

Abb. 41: Gutachter und Fehlerentdeckung

Gutachterzahl	4	5	6
Q_{Review}	0,499	0,550	0,600

Tabelle 29: Fehlerentdeckung im Review (Biffl, 2001, Tab. 5.2.4a)

Prüfling, Fehlerart und Fehlerschwere. Mit dem Faktor $ff_{p,Art}$ wird die Entdeckungsquote eines nominalen Reviews mit 5 Gutachtern an Tabelle 25 angepasst. Für die Anpassung an die Fehlerschwere sind keine Daten verfügbar.

Vorbereitungsintensität. Typisch werden für eine gründliche Vorbereitung von Dokumenten (Spezifikation und Entwurf) 10 Seiten pro Stunde, für Code 300 Zeilen pro Stunde benötigt (Frühauf et al., 2006). Die Quantifizierung für den Einfluss der

Vorbereitungsintensität leitet sich aus der Untersuchung von Biffl und Halling (2003) ab, die den Einfluss des Vorbereitungsaufwands auf die Fehlerentdeckung zeigt: Abhängig vom Vorbereitungsaufwand der einzelnen Gutachter verändert sich deren Fehlerentdeckung (Tabelle 30, linker Teil). Deutlich wird, dass die Fehlerentdeckungsquote zuerst mit mehr Aufwand wächst, dann aber nahezu konstant bleibt. Die Spezifikation in Biffl und Halling (2003) umfasst 35 Seiten. Aus der Vorbereitungsrate von 10 Seiten pro Stunde folgt, dass Gutachter, die zwischen 2 und 4 Stunden aufgewendet haben, dem Normalfall entsprechen (Nominalfall: 3,5 Stunden, in Tabelle 30 2 - 4 Stunden). Mit diesem Bezugspunkt werden die Daten normiert, so dass die lineare Funktion für den Einfluss quantifiziert werden kann (Tabelle 30, rechter Teil). Einen Rückgang der Entdeckungsquote für mehr als sechs Stunden Vorbereitung modelliere ich nicht, weil der Effekt gering ist, und weil der hohe Aufwand vermutlich mit geringer Gutachterkompetenz zusammenhängt.

Daten aus Biffl und Halling (2003)		Abgeleitete Quantifizierung	
Reading Time	Effectiveness (%)[a]	Vorbereitungsaufwand[b]	$f_{\text{Vorbereitung}}$ (%)[c]
0 - 2 h	8,5 %	50 %	48 %
2 - 4 h	17,7 %	100 %	100 %
4 - 6 h	20,9 %	150 %	118 %
6 - 8 h	19,5 %	200 %	118 %

Tabelle 30: Vorbereitungsdauer und Quantifizierung, abgeleitet aus Biffl und Halling (2003)

a. Detect Detection Effectiveness (DDE), entspricht der Fehlerentdeckungsquote Q
b. bezogen auf den Nominalfall
c. Einfluss auf die Fehlerentdeckungsquote Q im Modell

Gutachterkompetenz. In Probereviews mit Studenten werden deutliche Unterschiede zwischen 177 Gutachtern sichtbar (Biffl und Halling, 2002). Um diese Unterschiede in der Studie darzustellen, wurden die Gutachter in vier Klassen eingeteilt. Diese Klassen sind durch die Zahl der entdeckten Fehler definiert. Es gibt also Klassen für Gutachter, die wenige, einige, viele, sehr viele Fehler entdecken. Tabelle 32 zeigt die abgeleitete Quantifizierung des Einflusses. Dazu wurden alle Werte auf den Nominalfall normiert (Tabelle 31), der durch die Klasse mit Gutachtern, die viele Fehler entdecken, definiert ist ("Viele Fehler" in Tabelle 31). Die Zuordnung auf die 7 Kompetenzklassen des Modells erfolgt teilweise durch Inter- und Extrapolation; da die Klassen aber auf einer Ordinalskala definiert sind, ist dies nicht direkt möglich. Statt dessen nehme ich für die Inter- und Extrapolation an, dass der Abstand zwischen den Klassen gleich ist. Der Einfluss extrem niedriger Kompetenz ist durch Daten der Gutachter mit mangelnden Kenntnissen quantifiziert (Biffl und Halling, 2002).

Klasse für Zahl entdeckter Fehler in Probereviews	Wenige Fehler	Einige Fehler	Viele Fehler	Sehr viele Fehler
Einfluss auf Fehlerentdeckung	0,69	0,79	1,00	1,22
Einfluss auf Aufwand pro Fehler	0,69	0,79	1,00	1,31

Tabelle 31: Kompetenzeinfluss auf Fehlerentdeckung und Aufwand pro Fehler aus Biffl und Halling (2002)

Einfluss auf	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Q (Faktor $ff_{Review,KP}$)				(1,00)[a]			(1,65)[b]
	0,62	0,69	0,79	1,00	1,22	1,43	1,65
Vorbereitungsrate	0,69	0,69	0,79	1,00	1,31	(1,62)[c]	(1,93)[c]
Aufwand	1,76	1,44	1,27	1,00	0,76	0,62	0,52

Tabelle 32: Quantifizierung des Kompetenzeinflusses, abgeleitet aus Biffl und Halling (2002)

a. Nominalfall aus Biffl und Halling (2002)
b. Optimalfall aus Biffl und Halling (2002)
c. Extrapolation aus den Werten der Klassen mit geringerer Kompetenz

Prüflingsumfang. Es gibt keine Erfahrungswerte für die unterschiedliche Fehlerentdeckung durch priorisierte Begutachtung verdächtiger Teile der Software. Darum ist dieser Zusammenhang in CoBe parametrisierbar; es kann der Fehleranteil für 20 % des Umfangsanteils angegeben werden. Eine Pareto-Verteilung (20 % des Umfangs enthalten 80 % der Fehler) kann somit beispielsweise direkt angegeben werden. Mit diesem Wertepaar und den Randwerten (kein Umfang, keine Fehler; voller Umfang, alle Fehler) wird der Zusammenhang durch lineare Regression berechnet. Um aber überhaupt eine Quantifizierung in CoBe vorzugeben, orientiere ich mich an der Fallstudie von Do et al. (2006). Die Fallstudie untersucht die Wirkung der Testfallpriorisierung. Im besten Fall, der nur im Rückblick bestimmt werden kann, ergibt sich tatsächlich eine Pareto-Verteilung. Im Gegensatz dazu zeigt die zufällige Anordnung der Testfälle eine lineare Verteilung. Ich gehe davon aus, dass der optimale Fall in der Praxis nicht erreicht werden kann, weil nur zusammenhängende Teile begutachtet werden können und weil nur subjektiv bewertet werden kann, welche Teile kritischer als andere sind. Darum wähle ich einen Kompromiss ($r_{s,Schwere} = 0{,}5$). Es sammeln sich 45% der Fehler in 20% des Prüflings (Abbildung 42).

Codereviews von Korrekturen. Codereviews von Korrekturen werden mit gleichen Werten wie Codereviews neuer Software quantifiziert. Einziger Unterschied ist der Umfang, der begutachtet wird; bei Codereviews der Korrekturen ist dies der Änderungsumfang der Korrektur. Für den Änderungsumfang nennen Lyu et al. (2003) typisch 11 Anweisungen pro Korrektur, Jones (2007, S. 576) nennt 25 Anweisungen.

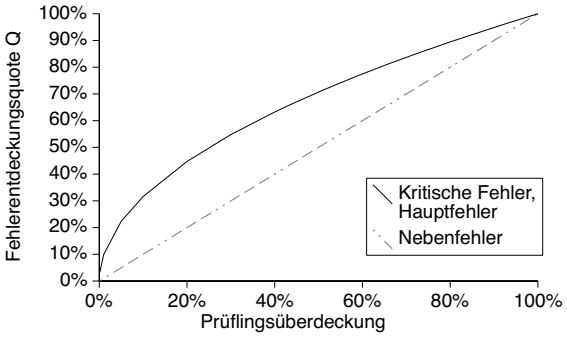

Abb. 42: Priorisierungsfunktion

Ich verwende den größeren Wert, um darzustellen, dass die Gutachter auch Code betrachten, der nicht geändert wurde, aber mit der Änderung zusammenhängt. Es reicht nicht, nur geänderten Code zu begutachten, weil die Gutachter den Kontext der Änderung verstehen müssen.

6.8.3 Codeanalyse

Die Fehlerentdeckungsquote hängt vom Werkzeug und den eingesetzten Analysen ab; sie ist definiert als der Anteil der im Prüfling enthaltenen Fehler, der durch eine Prüfung entdeckt wird. El Emam (2005) nennt 5 % als Erfahrungswert. Studien über die Codeanalyse enthalten keine verlässlichen Zahlen. So werden in den Studien wenige Projekte betrachtet. Fehlerzahlen aus dem Einsatz des Produkts fehlen (Kikuchi und Kikuno, 2001; Zheng et al., 2006). Das Werkzeug wird nachträglich für ein bereits intensiv eingesetztes Produkt angewendet (Wagner et al., 2005). Es werden absolute Zahlen angegeben, der Bezugswert für die Fehlerentdeckungsquote fehlt (Brand und Krohm, 2003). Darum wird die Fehlerentdeckungsquote für Codefehler auf 5 % gesetzt, Spezifikations- und Entwurfsfehler werden nicht entdeckt.

Zheng et al. (2006) messen Aufwand in etwa der gleichen Größenordnung für die Fehlerkorrektur nach Codereview und nach Codeanalyse. Darum kosten in CoBe Fehler, die in der Codeanalyse und im Codereview entdeckt werden, den gleichen Korrekturaufwand. Bei der Codeanalyse werden unterschiedlich viele falsche Befunde identifiziert, abhängig vom Werkzeug und den eingesetzten Analysen. Wagner et al. (2005) und Chou et al. (2001) messen zwischen 30 % und 96 %. Ich wähle 50 % für das Modell, auf jeden Fehler kommt also ein falscher Befund. Den Vorbereitungsaufwand für die Einbindung des Werkzeugs in die individuelle Entwicklungsumgebung des Entwicklers lege ich auf zwei Arbeitstage fest, beeinflusst durch den Produktivitätsfaktor *af*.

6.8.4 Tests

Zahl der Testfälle. Die Testfallzahl wird linear aus dem Umfang berechnet (Tabelle 33 aus Jones, 2007). Für einen überproportionalen Zusammenhang gibt es

keine Daten. Das Maximum ist dem vollständigen Black-Box-Test zugeordnet, etwa im Systemtest mit T_n = 0,6 Testfällen pro Function Point. Werte des Integrationstests werden in CoBe für den Subsystemintegrationstest und den Systemintegrationstest verwendet.

Testfallzahl pro Function Point	Minimum	Mittelwert	Maximum
Modultest	0,20	0,45	1,20
Integrationstest	0,20	0,40	0,75
Systemtest	0,15	0,25	0,60

Tabelle 33: Testfallzahl pro Function Point nach Jones (2007)

Berichte über die erreichte Überdeckung im Black-Box-Systemtest nennen 50 % bis 60 % Anweisungsüberdeckung (Grady, 1992; Burr und Young, 1998; Piwowarski et al., 1993). Dies gilt auch im Modultest (Briand und Pfahl, 1999; Lyu et al., 2003; Horgan et al., 1994; Janzen und Saiedian, 2006; Burr und Young, 1998). 80 % der Anweisungen werden durch Vervollständigung (Burr und Young, 1998; Nagappan et al., 2008) überdeckt. Sehr kleine Programme (Müller und Höfer, 2007) werden leicht nahezu vollständig überdeckt. Daten zur Term- und Schleifenüberdeckung sind nicht verfügbar. Die erreichte Anweisungsüberdeckung ist durch vier Datenpunkten (Tabelle 34) mit linearer Regression quantifiziert:

$$c_0 = 0,6103 \cdot t^{0,5151}.$$

Testparameter	c_0[a]	Quelle	Q (%)[b]	Quelle
Nominaler Black-Box-Test	50 %	Grady (1992), Piwowarski et al. (1993)	35 %	Jones (1996)
Vollständiger Black-Box-Test	60 %		60 %	Piwowarski et al. (1993)
76% der Anweisungen und Zweige	-	-	68 %	Lauterbach und Randall (1989)
Doppelt so viele Testfälle wie im nominalen Black-Box-Test	80 %	Dupuy und Leveson (2000)		-
Ohne Testfälle	0 %		0 %	-

Tabelle 34: Quantifizierung der Überdeckung und der Fehlerentdeckung

a. Erreichte Anweisungsüberdeckung
b. Fehlerentdeckungsquote

Diese Quantifizierung wird im Modell für den Zusammenhang zwischen der Zahl der Testfälle und der Anweisungsüberdeckung verwendet. Die Quantifizierung der Zweig-, Schleifen- und Termüberdeckung erfolgt mit den Daten aus Abschnitt 7.4.2.

Da im Glass-Box-Test gezielt bestimmte Code-Einheiten überdeckt werden, nehme ich an, dass pro Testfall im statistischen Mittel eine höhere Überdeckung als im Black-Box-Test erreicht wird. Diese Annahme wird in Dupuy und Leveson (2000) bestätigt. Angelehnt an diesen Erfahrungsbericht wird in CoBe der Faktor zwei verwendet: $c_0 = 1,221 \cdot t^{0,5151}$.

Fehlerentdeckung. Die Fehlerentdeckung im Systemtest wird mit verschiedenen Quellen quantifiziert (Tabelle 34). Die Parameter r_{qt} und q_{t} werden über lineare Regression ermittelt (Abbildung 43). Die Entdeckungsquoten aller Tests werden mit dem Faktor $ff_{p,Art}$ an die Tests und die Fehlerarten angepasst (Tabelle 25). Die Fehlerschwere hat keinen Einfluss, weil dafür keine Daten verfügbar sind. Die Korrekturquote beträgt 90 %, somit werden 90 % der entdeckten Fehler korrigiert (Jones, 1996).

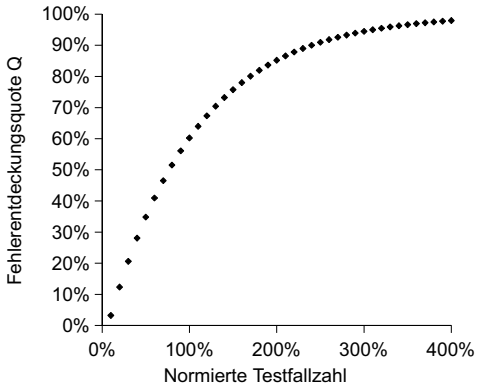

Abb. 43: Testfälle und Fehlerentdeckung

Aufwand. Der Aufwand pro Testfall wird mit den Werten aus Jones (1996, 2007) bestimmt. Die Verteilung auf Vorbereitung, Aufbau und Durchführung orientiert sich an Jones (1996, 2007) und van Megen und Meyerhoff (1995), so dass für Vorbereitung 38 %, für das Testgeschirr 29 % und für die Durchführung 33 % des Aufwands benötigt werden. Im Glass-Box-Test gibt es keine Trennung zwischen Vorbereitung und Durchführung. Der Aufwand für den Testaufbau, insbesondere für die Instrumentierung, ist konstant durch 29% des Testaufwands für 80 % Anweisungsüberdeckung definiert. Die Programmiersprache beeinflusst den Testaufwand im Glass-Box-Test im gleichen Verhältnis, in dem Function Points und Anweisungen zueinander stehen.

Aufwand pro Testfall	
Modultest	0,73 Eh
Subsystem- und Systemintegrationstest	1,00 Eh
Systemtest	1,10 Eh

Tabelle 35: Aufwand für Testfälle abgeleitet aus Jones (1996, 2007)

Testerkompetenz. Die Quantifizierung orientiert sich an Devnani-Chulani (1997) und dem Einfluss der Programmierfähigkeit auf Fehlereinfügerate und Produktivität. Andere Werte sind nicht verfügbar.

Einfluss der Kompetenz auf	Extra Low	Very Low	Low	Nominal	High	Very High	Extra High
Q (Faktor $ff_{Test,KP}$)	0,76	0,76	0,87	1,00	1,15	1,32	1,32
Aufwand	1,34	1,34	1,15	1,00	0,85	0,71	0,71

Tabelle 36: Einfluss der Kompetenz, abgeleitet aus Devnani-Chulani (1997)

Vorbereitungszeitpunkt. Bei früher Vorbereitung werden 100 % der Spezifikationsfehler des Systemtests entdeckt und früh korrigiert. Für den Korrekturaufwand wird der gleiche Wert wie im Entwurfsreview verwendet.

Kapitel 7

Modellrealisierung, Modellprüfung und Modellverbesserung

In diesem Kapitel ist die Implementierung von CoBe beschrieben (Abschnitt 7.1). Dann wird das Vorgehen festgelegt, mit dem CoBe geprüft wird (Abschnitt 7.2). Dazu werden zuerst Daten aus studentischen Projekten verwendet (Abschnitt 7.3). Mit den Daten werden Modellzusammenhänge (Abschnitt 7.4) und Modellresultate (Abschnitte 7.5 und 7.6) geprüft. Zwei Verbesserungen werden identifiziert und umgesetzt. Die Ergebnisse zeigen, dass die Kalibrierung notwendig ist. Modellresultate und Daten aus den Projekten stimmen gut überein.

7.1 Die Realisierung von CoBe

Die Modellrealisierung ist der letzte Schritt der Modellbildung. Dieser Schritt erfolgt, nachdem ein Modell der Realität erstellt und als funktionales Modell dargestellt wird (Abschnitt 3.7). CoBe wurde sowohl als Tabellenkalkulation als auch als Java-Anwendung realisiert.

7.1.1 Vorgehen zur Realisierung des Modells

Prinzipiell können die Modellresultate von Hand ausgerechnet werden. Da dies aufwändig und fehleranfällig ist, wurde das Modell als ausführbare Anwendung realisiert. Da die Modellbildung iterativ erfolgt, angelehnt an das Konzept für den Modelleinsatz (Abschnitt 3.6.1), gehört die Erprobung zur Modellbildung. Sie zeigt, ob wichtige Zusammenhänge fehlen oder falsch sind. Somit ist notwendig, das Modell frühzeitig zu implementieren, auch wenn die Konzepte des Modells noch nicht stabil sind.

Die Realisierung des Modells als Programm und die Verifikation des Programms kann aufwändig werden, vor allem, wenn das Programm und die Programmstruktur häufig umgearbeitet werden. Dies kann der Fall sein, wenn das Modell iterativ entsteht, da sich dann die Modellkonzepte und die Modellzusammenhänge ändern können. Darum erfolgte die Realisierung in mehreren Stufen mit unterschiedlichen Modellen: Zuerst wurde das Modell als Tabellenkalkulation realisiert, um während der Modellbildung flexibel bei umfangreichen Modelländerungen zu sein. Die Tabellenkalkulation ermöglichte, CoBe frühzeitig zu implementieren und zu erproben. Somit können Fragen untersucht werden, die sich bei der Modellbildung stellen.

Dazu gehört etwa: Welche Zusammenhänge sind relevant? Welche Zusammenhänge wirken sich auf die Modellresultate aus? Wie können die Zusammenhänge quantifiziert werden? Welche Eingaben sind notwendig? Die Tabellenkalkulation ist also vor allem ein Werkzeug zur Forschung.

Erst nachdem die Modellbildung weitgehend abgeschlossen war, erfolgte die Realisierung als Java-Anwendung. Damit wird eine komfortable Bedienung und die automatisierte Berechnung für die Sensitivitätsanalyse im Rahmen der Validierung (Abschnitt 7.2.2) ermöglicht.

Tabellenkalkulation und Java-Programm wurden in zwei Ausbaustufen erstellt. Somit konnten zuerst das Basismodell und die Prüfungsmodelle geprüft werden, bevor CoBe vollständig realisiert wurde:

- **Version 1:** Die erste Version enthält das Basismodell und detaillierte Prüfungsmodelle für das Spezifikationsreview, das Entwurfsreview und das Codereview. Der Systemtest ist im Detail modelliert. Da die Zusammenhänge des Testmodells aber nur durch wenige Daten untermauert sind, sind andere Testebenen nicht detailliert modelliert, sondern nur durch Fehlerentdeckungsquoten dargestellt.
- **Version 2:** Für die zweite Version wurde die Version 1 um die Codeanalyse, den Integrationstest von Subsystemen, Korrekturprüfprozesse, Codereviews von Korrekturen, detaillierte Modelle für Tests auf allen Ebenen und ein Modell für die Kosten falscher Befunde erweitert.

Abbildung 44 skizziert den zeitlichen Ablauf der Modellrealisierung für die Tabellenkalkulation und die Java-Anwendung in den beiden Ausbaustufen. Die Version 1 der Tabellenkalkulation erfolgte während der Modellbildung und Erprobung. Sie wurde zur Version 2 für die Validierung erweitert. Dabei entstand die erste Version der Java-Anwendung, die dann zur Version 2 ausgebaut wurde.

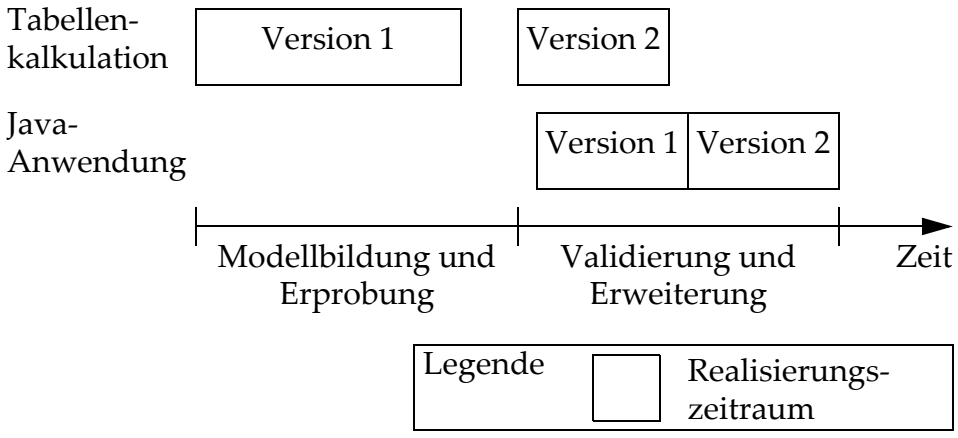

Abb. 44: Zeiträume der Realisierung

7.1.2 Realisierung als Tabellenkalkulation

Die Tabellenkalkulation für CoBe ist auf mehrere Arbeitsblätter aufgeteilt. Die Aufteilung orientiert sich an der Modellstruktur in Abbildung 17. Jedes Prüfungsmodell besteht aus einem Arbeitsblatt für die Zusammenhänge und einem Arbeitsblatt für das Fehlerstrommodell, mit dem entdeckte und entfallende Fehler berechnet werden. Weitere Arbeitsblätter enthalten COCOMO II, Korrekturkosten und Funktionen zur Kalibrierung und Quantifizierung des Modells.

Die Modelleingaben für den Prüfprozess, die Prüfparameter und die Produkt- und Projektmerkmale sind auf einem Arbeitsblatt gemeinsam mit den Modellresultaten enthalten. Die Abbildungen 45 und 46 zeigen beispielhaft einen Ausschnitt aus diesem Arbeitsblatt. Abbildung 45 stellt die Eingaben für das Spezifikationsreview dar, Abbildung 46 zeigt die Eingaben für den Modultest. Dabei sind die Eingaben für den Test Mindestforderungen. 0 % Anweisungsüberdeckung bedeutet also nicht, dass keine Anweisungen überdeckt werden dürfen, sondern dass auch mehr als 0 % der Anweisungen überdeckt werden können. Das Arbeitsblatt enthält die Modellausgaben in tabellarischer Form. Ein zusätzliches Arbeitsblatt enthält Diagramme, die diese Resultate als Balkendiagramme darstellen. Abbildung 47 zeigt als Beispiel den Nutzen des Spezifikationsreviews in Euro.

Spezifikationsreview		
Prüflingsüberdeckung	**100%**	der Spezifikation
Gutachterzahl	**5**	Gutachter
Gutachterkompetenz	**3**	0 (XL) bis 6 (XH)
Vorbereitungsintensität	**10**	Seiten pro Stunde

Abb. 45: Eingaben des Spezifikationsreviews

Modultest		
Black-Box-Test		
Funktionen	**100%**	Abdeckung
Äquivalenklassen	**100%**	Abdeckung
Sonderfälle	**0%**	Abdeckung
Glass-Box-Test		
Anweisungen	**0%**	Überdeckung
Zweige	**0%**	Überdeckung
Schleifen	**0%**	Überdeckung
Terme	**0%**	Überdeckung
Testerkompetenz	**3**	

Abb. 46: Eingaben des Modultests

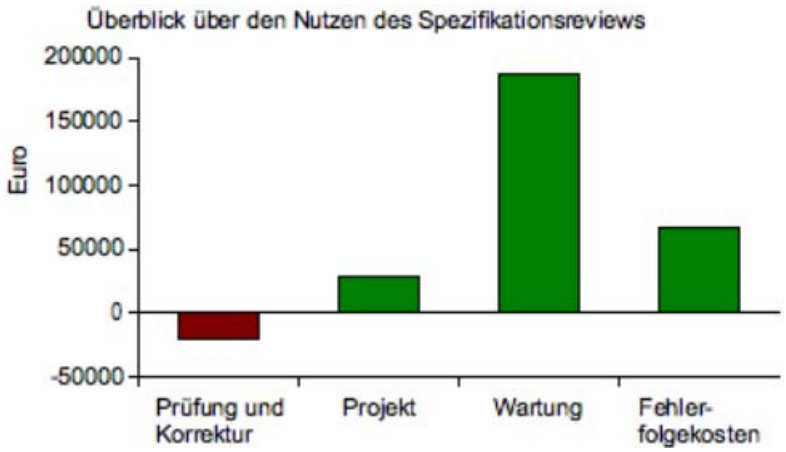

Abb. 47: Resultate für das Spezifikationsreview

Die Quantifizierung des Modells wird durch die Tabellenkalkulation direkt unterstützt, indem wichtige Zusammenhänge des Modells jeweils in einem eigenen Arbeitsblatt quantifiziert werden können:

Späte Korrekturkosten. Für jede Prüfung kann ein Basiswert für den Korrekturaufwand pro Fehler eingegeben werden. Die Faktoren, mit denen der Einfluss der Fehlerschwere und der Fehlerart modelliert sind, können direkt eingegeben werden, auch einzeln für jede Prüfung. Beispielsweise wird ein Faktor von 1,3 als Verhältnis des Korrekturaufwands zwischen kritischen Fehlern und Hauptfehlern eingegeben (Abschnitt 6.3.5). Die weiteren Einflüsse, insbesondere der Produktumfang und der Aufwandsfaktor zur Kalibrierung, werden berücksichtigt. Das Arbeitsblatt enthält als Ergebnis den Korrekturaufwand pro Fehler für jede Prüfung, jeweils für jede Kombination aus Fehlerart und Fehlerschwere.

Priorisierung von Reviews. In diesem Arbeitsblatt können die Datenpunkte, die zur Beschreibung der Priorisierungsfunktion notwendig sind (Abschnitt 6.4.2), direkt eingegeben werden. Diese Eingaben sind jeweils für kritische Fehler, Hauptfehler und Nebenfehler möglich. Daraus wird der Exponent der Priorisierungsfunktion berechnet. Die Funktion wird graphisch dargestellt; Abbildung 42 stammt direkt aus der Tabellenkalkulation.

Test. Eingegeben werden können Datenpunkte für den Zusammenhang zwischen der Testfallzahl und der erreichten Überdeckung (Abschnitt 6.6.2). Daraus berechnet die Tabellenkalkulation die Parameter für die Gleichung und stellt den Zusammenhang graphisch dar. Auch der Zusammenhang zwischen der Testfallzahl und der Fehlerentdeckungsquote wird in diesem Arbeitsblatt durch die Eingabe einzelner Datenpunkte in eine Tabelle quantifiziert. Die Parameter der Gleichung (Abschnitt 6.6.2) werden berechnet. Der Zusammenhang wird graphisch dargestellt (Abbildung 43).

Review. Datenpunkte für den Zusammenhang zwischen der Zahl der Gutachter und der Fehlerentdeckungsquote können in eine Tabelle eingegeben werden. Die Parameter der Gleichung (Abschnitt 6.4.2) werden aus diesen Eingaben berechnet. Das Resultat wird graphisch dargestellt. Abbildung 41 ist aus dem Arbeitsblatt übernommen.

7.1.3 Realisierung als Java-Anwendung

Während die Realisierung als Tabellenkalkulation vor allem zur Erprobung des Modells und der Konzepte gedacht war, wird die Java-Anwendung benötigt, um Resultate vieler Eingabekombinationen zu berechnen. Dies ist notwendig, um das Verhalten des Modells zu untersuchen. Die Anwendung trägt den Namen CoBeCalc.

Einsatzszenarien für CoBeCalc

CoBeCalc bietet Einsatzmöglichkeiten für drei Einsatzszenarien:

- Im interaktiven Modus werden Modellresultate für einen Satz Eingaben berechnet. Dabei wird jede Eingabe über die Benutzungsschnittstelle auf einen einzigen Wert gesetzt. Diese Wertebelegung kann in einer Datei gespeichert werden.
- Damit das Verhalten des Modells untersucht werden kann, müssen Resultate für viele Eingabekombinationen berechnet werden. Dies wird mit CoBeCalc unterstützt, indem Dateien eingelesen werden können, die viele Eingabekombinationen enthalten. Diese Schnittstelle ist speziell für das Werkzeug SimLab (SimLab, 2009) zur Sensitivitätsanalyse des Modells implementiert (Abschnitt 8.1). SimLab erzeugt die Eingabekombinationen durch pseudo-zufällige Auswahl aus den Wertebereichen der Eingaben.
- Eingabekombinationen, die aus fest vorgegebenen Werten gebildet werden, werden mit CoBeCalc durch Klassen unterstützt, die die Erzeugung der Kombinationen und die Berechnung der Resultate realisieren (Abschnitt 8.1).

Für die interaktive Verwendung von CoBeCalc werden die Modellresultate in tabellarischer Form an der Benutzungsschnittstelle ausgegeben und können als csv-Datei gespeichert werden. Sie können somit leicht in andere Anwendungen übertragen werden. Bei den anderen beiden Verwendungsmöglichkeiten werden die Resultate für Projekt-Qualitätskosten und Gesamt-Qualitätskosten in eine Datei geschrieben.

Die Benutzungsschnittstelle von CoBeCalc

Die Benutzungsschnittstelle ist für die Eingaben gegliedert in Prozess- und Produktmerkmale, Eingaben für die Fehlerfolgekosten, den Prüfprozess und die einzelnen Prüfparameter für Reviews und für Tests. Jeder dieser Teile ist in einem Reiter angeordnet. Abbildung 48 zeigt einen Teil der Eingaben für Prozess- und Produktmerkmale, Abbildung 49 zeigt die Eingaben für Reviews.

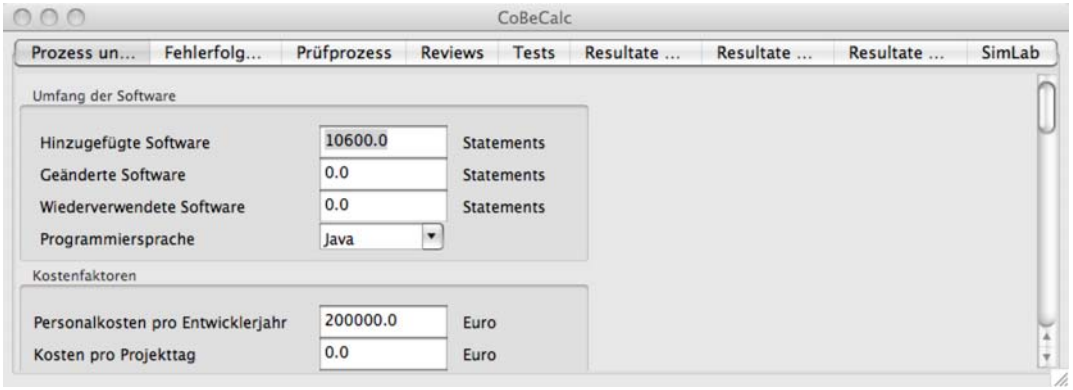

Abb. 48: Modelleingaben für Prozess und Produkt (Ausschnitt)

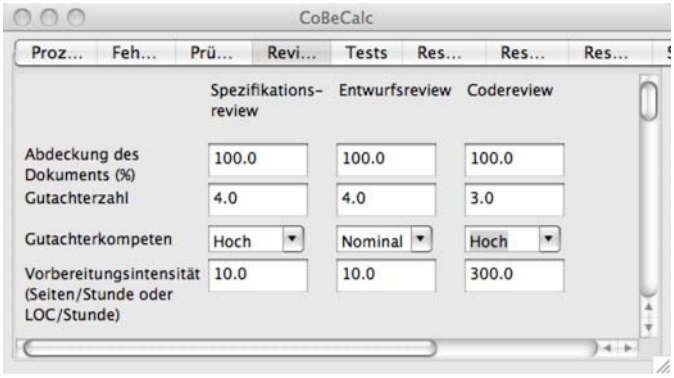

Abb. 49: Eingaben der Review-Prüfparameter

Abbildung 50 zeigt die zusammengefassten Resultate für Qualitätskosten in Euro und die COCOMO-II-Resultate, die zur Kalibrierung benötigt werden. Aufwand, Dauer, Personalbedarf und Geldwerte werden für einzelne Aktivitäten ausgegeben (Abbildung 51). Ergänzend werden enthaltene und entdeckte Fehler ausgegeben.

Die Struktur von CoBeCalc

Bauer (2008) gliedert das Programm in die drei Pakete uI, inputOutput und calculation. Das Paket uI realisiert die graphische Benutzungsschnittstelle. Das Paket inputOutput enthält die Klassen für die Datenein- und -ausgabe.

Die Modellzusammenhänge sind im Paket calculation realisiert. Die Klassen in diesem Paket orientieren sich am Aufbau des Modells. Pro Prüfung gibt es jeweils eine Klasse (Abbildung 53). Diese enthält Methoden und Aktivitäten des Prüfungsmodells zur Fehlerentdeckung, des Modells der Prüfkosten und des Modells der Prüfwiederholungskosten. Das Fehlerstrommodell ist für die verschiedenen Dokumente in einzelnen Klassen realisiert (Abbildung 52). Alle Werte für die Quantifizierung sind in

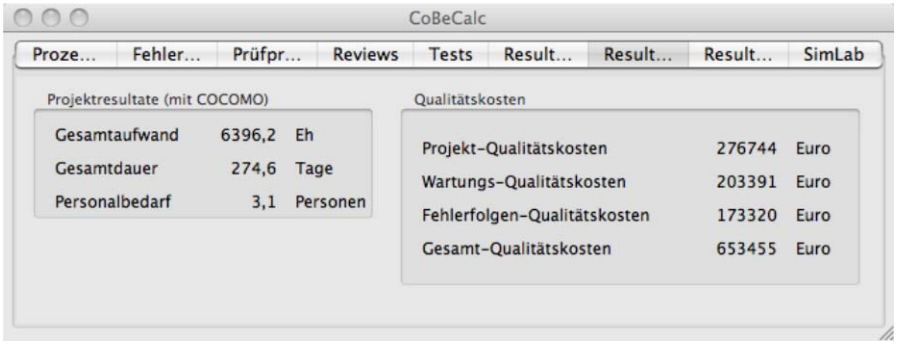

Abb. 50: Modellresultate für Qualitätskosten und COCOMO-II-Resultate

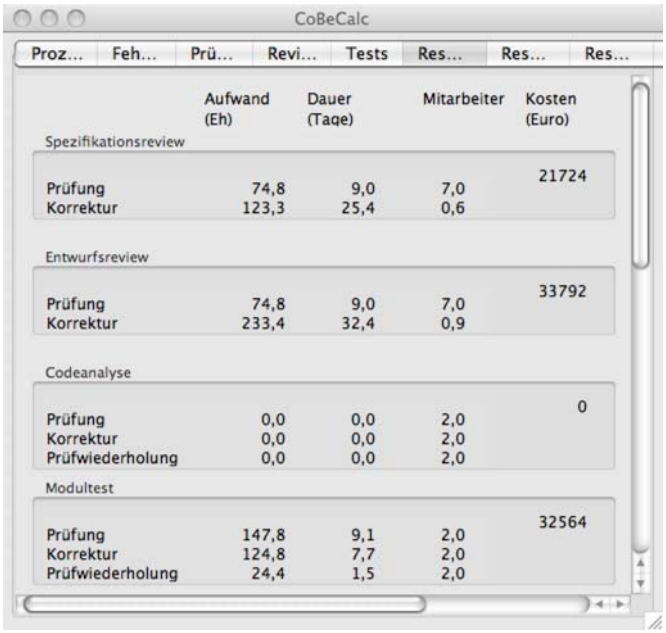

Abb. 51: Modellresultate für Aktivitäten (Auszug)

einer Textdatei enthalten und können dadurch angepasst werden. Insbesondere können spezielle Anpassungen an eine Umgebung durchgeführt werden, ohne dass Code geändert werden muss.

7.2 Überblick über die Modellprüfung

CoBe berechnet aus Eingaben über Prüfungen und Prüfparametern die Resultate, also Kosten und Nutzen. Somit gehört das Modell zu den Pseudometriken (Ludewig und Lichter, 2007), diese müssen validiert werden. Die möglichen Eingabewerte beschreiben den Handlungsspielraum. Die Resultate berechnen sich aus einem gewählten

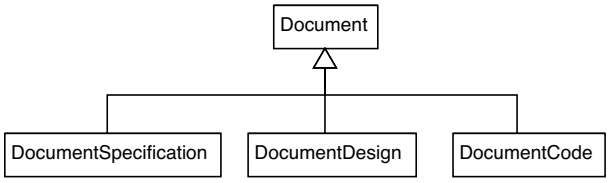

Abb. 52: Klassen für Dokumente

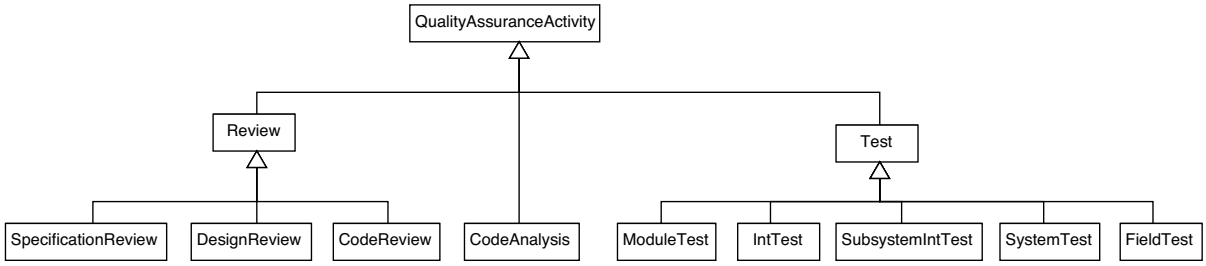

Abb. 53: Klassen für Prüfungen

Punkt in diesem Handlungsspielraum. Somit handelt es sich um ein Entscheidungsmodell, auch wenn die optimale Handlungsalternative nicht direkt berechnet wird (Laux, 1998). Damit treten bei der Validierung von CoBe die speziellen Schwierigkeiten auf, die typisch bei der Validierung von quantitativen Entscheidungsmodellen auftreten. Darum wird zuerst ein Überblick über die Prüfmethoden und die damit verbundenen Schwierigkeiten gegeben, bevor das Vorgehen festgelegt wird.

7.2.1 Verifikation und Validierung für quantitative Modelle

Für die Prüfung quantitativer Modelle wird zwischen Verifikation und Validierung unterschieden (Drappa, 1998; Rykiel, 1995; Sargent, 2005):

Def. **Validierung.** Validierung eines Simulationsmodells bedeutet festzustellen, ob das Simulationsmodell eine für den spezifizierten Zweck der Untersuchung hinreichend genaue Repräsentation des betrachteten realen Systems ist (Drappa, 1998).

Def. **Verifikation.** Verifikation eines Simulationsmodells bedeutet sicherzustellen, dass das Modell mit hinreichender Genauigkeit von einer Repräsentationsform in eine andere überführt wurde (Drappa, 1998).

Die Validierung erfolgt also gegen den Modellzweck. Verifikation und Validierung werden den Schritten der Modellbildung zugeordnet (Sargent, 2005). Abbildung 54 zeigt diese Zuordnung für den gewählten funktionalen Modellierungsansatz (Abschnitt 3.7, S. 49).

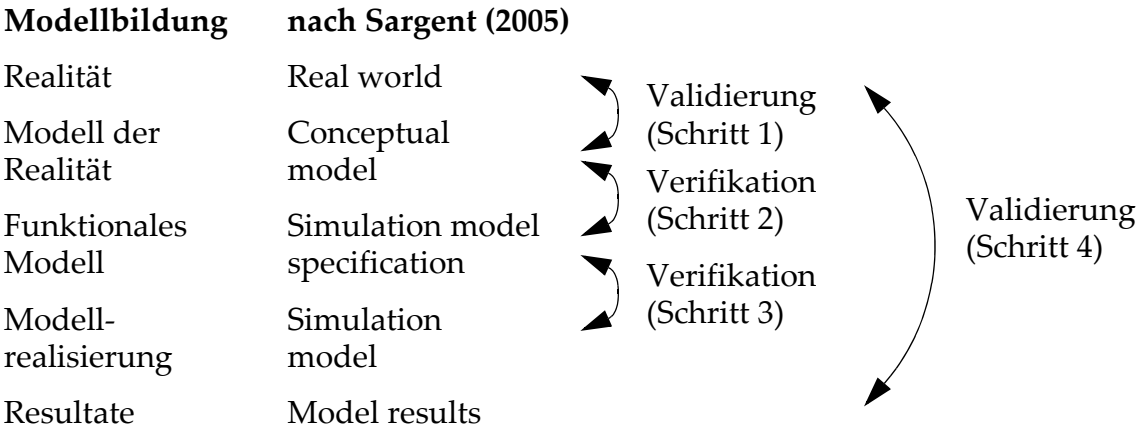

Abb. 54: Modellbildung, Verifikation und Validierung nach Sargent (2005)

Verifikation

Das funktionale Modell wurde gegen das Modell der Realität durch Walkthroughs geprüft (Schritt 2). Konstruktiv ist dieser Schritt abgesichert, weil formalisierte Zusammenhänge aus Modellen verwendet wurden, die bereits geprüft und belegt wurden. Dazu gehören zum Beispiel die COCOMO-II-Zusammenhänge. Die Modellrealisierung (Schritt 3) wurde durch Tests und Walkthroughs geprüft. Tracing, das Nachverfolgen von Berechnungen, und Modularisierung sind wichtige Techniken (Drappa, 1998). Tracing konnte durch die Modularisierung einfach durchgeführt werden, da die Modellstruktur auch Zwischenergebnisse sichtbar macht (Abschnitt 6.2). Somit konnten in den Tests nicht nur Endergebnisse, sondern auch Zwischenergebnisse geprüft werden. Dazu wurden Sollresultate für Zwischen- und Endergebnisse vorgegeben, so dass die Modellberechnungen Schritt für Schritt nachvollzogen und geprüft wurden. Tabellenkalkulation und Java-Programm wurden unabhängig voneinander erstellt und gegeneinander geprüft (Bauer, 2008).

Validierung

Die direkte Methode zur Validierung ist, Modellresultate mit Werten aus der Realität, den Istwerten, zu vergleichen. Bei der direkten Validierung von CoBe treten dann aber die folgenden Schwierigkeiten auf:

1. Entscheidungen können nicht rückgängig gemacht werden. Darum stehen Istwerte alternativer Handlungen für dieselbe Situation prinzipiell nicht zur Verfügung (Gass, 1983).
2. Bei vielen Eingabeparametern werden sehr viele Messungen benötigt, um eine verlässliche Aussage über die Modellvalidität zu treffen (Sargent, 2005). Dies kostet Zeit und Aufwand, beide stehen nicht unbegrenzt zur Verfügung.

3. Weil CoBe Industrieprojekte repräsentieren soll, muss die Validierung mit Industrieprojekten erfolgen. Die Istwerte, d.h. die Werte, mit denen die Modellresultate verglichen werden, müssen also in realen Projekten gemessen werden. Ihre Erhebung kostet Aufwand und dauert lange, weil dazu der Prozess des Projekts analysiert werden muss. Dabei müssen verfügbare Daten identifiziert, dann ausgewertet werden. Nicht verfügbare Daten müssen erhoben werden. Zeit und Aufwand stehen aber in dieser Arbeit nicht unbegrenzt zur Verfügung. Vor allem fällt für die Mitarbeiter der Industrieprojekte Aufwand an; deren Aufwand ist besonders begrenzt, weil in Projekten Zeit und Aufwand knapp sind.
4. Für die Prüfung der Prognose muss zuerst das Projekt und dann zumindest ein Teil der Einsatz- und Wartungsphase abgewartet werden. Dies ist in dieser Arbeit kaum möglich, weil ein reales Projekt und sein Produkt dazu über mehrere Jahre begleitet werden müssen; dazu reicht die Zeit der Arbeit nicht aus.
5. Eine vollständige Validierung eines quantitativen Modells ist nicht möglich, jedes Modell kann nur falsifiziert werden (Sargent, 2005).

Eine direkte, vollständige Validierung ist also nicht möglich. Die Validierung bleibt zwangsweise lückenhaft. Gass (1983) nennt darum weitere Methoden, die ergänzend zur unvollständigen, direkten Validierung eingesetzt werden können:

- Die Bewertung durch Experten,
- die Validierung einzelner Modellteile und
- die Sensitivitätsanalyse, um das Modellverhalten zu betrachten und zu bewerten.

Die Sensitivitätsanalyse ergänzt die Validierung, weil sie nicht auf einem Vergleich zwischen Modellresultat und Realität beruht, sondern zeigt, wie und wie stark sich die Modellparameter auf die Modellausgaben auswirken: Ändern sich die Aussagen des Modells? Welche Eingaben wirken sich stärker, welche wirken sich schwächer aus? Wie verhält sich das Modell? Wie wirken sich unsichere Parameter aus?

7.2.2 Schritte der Modellprüfung

Das Modell CoBe wird zuerst mit studentischen Projekten validiert, weil diese Werte leichter verfügbar sind. Außerdem können die notwendigen Metriken passend zum Modell definiert werden. Vorteilhaft ist auch, CoBe vor dem Industrieeinsatz zu erproben, um Modelldefizite vor einer teuren Industriestudie zu erkennen. Weil studentische Projekte nur eingeschränkt verallgemeinerbar sind (Prechelt, 2001), kann auf eine Validierung in der Industrie aber nicht verzichtet werden. Mit studentischen Projekten und Industrieprojekten steht die Validierung auf einer breiten Datenbasis. Sie erfolgt schritthaltend mit den Schritten der Modellrealisierung (Abschnitt 7.1), so dass die Modellversion 1 mit studentischen Projekten geprüft wurde. Auch diese Prüfung erfolgte in einzelnen Schritten, um nach und nach unterschiedliche Aspekte des Modells abzudecken:

- Die Zusammenhänge in CoBe sind unterschiedlich gut belegt. Darum werden ausgewählte Zusammenhänge mit Daten aus studentischen Projekten untersucht (Abschnitt 7.4). Damit werden die einzelnen Annahmen, aus denen das Modell der Realität besteht, überprüft (Schritt 1 in Abbildung 54). Dies erlaubt, zumindest einen Teil des Wertebereichs eines Parameters zu prüfen (Problem 2) und die Wirkungen von Entscheidungen zu prüfen, ohne dass auf unterschiedliche Projekte zurückgegriffen werden muss (Problem 1).
- Im nächsten Schritt wird CoBe mit Durchschnittswerten aus 21 studentischen Projekten erprobt (Abschnitt 7.5), zuerst mit einem unkalibrierten Modell und dann mit einem kalibrierten Modell. Es erfolgt also zuerst eine Erprobung, bevor teure Industriedaten verwendet werden (Problem 3).
- Mit den Daten der studentischen Projekte wird geprüft, ob CoBe Kosten und Nutzen einzelner Projekte ausreichend genau nachträglich beschreibt (Schritt 4 in Abbildung 54). Da die Projekte ähnlich sind, kann die Wirkung unterschiedlicher Prüfparameter durch Vergleich geprüft werden (Problem 1). Abschnitt 7.6.1 beschreibt die Resultate.
- Die Prognose der Kosten und des Nutzens mit CoBe wird durch eine Kreuzvalidierung in Abschnitt 7.6.2 untersucht (Schritt 4 in Abbildung 54). Mit einer Kreuzvalidierung wird nachgebildet, dass bei der Prognose einige Eingaben nicht gemessen, sondern nur geschätzt werden können oder aus Archivdaten stammen. Damit muss nicht die Wartungs- und Einsatzphase abgewartet werden (Problem 4).
- Die Sensitivitätsanalyse in Abschnitt 8.1 zeigt, wie die Modellresultate von den Eingaben abhängen. Die Wirkungen der Entscheidungen über Prüfungen und die Wirkungen von unsicheren Eingaben werden untersucht (Problem 1). Das Verhalten des Modells kann beurteilt werden (Problem 5), auch für sehr viele Eingaben (Problem 2). Dies wird durch die Optimierung von Prüfprozessen zusätzlich untersucht (Abschnitte 8.2).
- Die Validierung mit Industrieprojekten (Abschnitte 8.4 und 8.5) zeigt, ob CoBe für deren Prozess erweitert werden kann. Sie erfolgt mit der Modellversion 2, die Erweiterungen für die Industrieprojekte enthält. Dieser Vergleich mit Istwerten aus der Industrie zeigt (Problem 3), ob CoBe diese Projekte ausreichend genau repräsentiert (Schritt 4 in Abbildung 54).

Anschließend wird der Modelleinsatz demonstriert:

- Modelleinsatz und Modellverhalten werden mit Beispielen demonstriert. Dazu werden auch Daten aus Berichten über Prozessverbesserungsmaßnahmen verwendet (Abschnitt 8.7, Schritt 4 in Abbildung 54).

7.2.3 Kriterien für die Modellprüfung

Die Definition der Validierung verlangt, dass das Modell das reale System ausreichend genau repräsentiert. Dazu werden Modellresultate mit Istwerten verglichen

(IEEE 1061, 1998); die Prognose soll eine Mindestgenauigkeit erreichen (Tabelle 39). Validierungskriterien definieren dafür Metriken mit Grenzwerten. Die Standard-Metrik für die Bewertung der Genauigkeit ist der Betrag des relativen Fehlers (Magnitude of relative error, *MRE*, IEEE 1061, 1998; Fenton und Pfleeger, 1997; Conte et al., 1986; Kemerer, 1987):

$$MRE = \left| \frac{Istwert - Modellresultat}{Istwert} \right|$$

Conte et al. (1986) fordern, dass Modellresultate im Schnitt weniger als 25 % abweichen und dass 75 % der Modellresultate innerhalb dieser 25 %-Grenze liegen. Dargestellt wird dieses Kriterium als $pred(25\,\%) \geq 75\,\%$.

Der relative Fehler *MRE* ist nur bedingt plausibel, weil zu niedrige Modellresultate zu schwach bewertet werden. Berechnet das Modell beispielsweise 90 % zu wenig Fehler, dann erscheint die Abweichung intuitiv größer, als wenn das Modell 90 % zu viele Fehler berechnet. In beiden Fällen beträgt der relative Fehler aber 90 %. Das Verhältnis zwischen Modellresultat und Istwert ist aussagekräftiger (1 : 10 im ersten, fast 2 : 1 im zweiten Fall) und ist ohne Angabe eines Istwerts anschaulicher. Verhältniswerte zwischen Modellresultat und Istwert sind ebenso wie der relative Fehler üblich, um die Schätzungenauigkeit bei der Planung zu beschreiben (Boehm, 1981). Daran angelehnt verwende ich das logarithmierte Verhältnis zwischen Modellresultat und Istwert: Die logarithmische Abweichung *LE* (Error) ist definiert mit der Hilfsmaßeinheit deziBel (dB)[1]; Tabelle 37 zeigt Abweichungen im Vergleich.

$$LE = 10 \cdot \left| \log_{10}\left(\frac{Modellresultat}{Istwert}\right) \right|$$

LE	Modellresultat zu niedrig		Modellresultat zu hoch	
	Faktor	*MRE*	Faktor	*MRE*
3,0 dB	0,50	50 %	2,00	100 %
2,0 dB	0,63	37 %	1,58	58 %
1,0 dB	0,79	21 %	1,26	26 %

Tabelle 37: Vergleich logarithmierter und relativer Fehler

Der 25 %-Grenze von Conte et al. (1986) entspricht etwa 1 dB. Diese Grenze ist erfahrungsgemäß sehr eng und wird selbst von COCOMO II kaum erreicht:

1. Der Betrag des Logarithmus ermöglicht, dass der gleiche Faktor im Zähler und im Nenner gleich bewertet wird: Ein Verhältnis von 1 : 2 wird gleich wie 2 : 1 bewertet.

- Von 83 Projekten aus 18 Organisationen liegen 49 % der COCOMO-II-Resultate innerhalb der 25 %-Grenze. Nach der Kalibrierung für die Organisation sind 55 % der Resultate innerhalb der 25 %-Grenze (Boehm, 2000, S. 162).
- Eine Kreuzvalidierung von COCOMO II mit 161 Projekten ergibt, dass vor Kalibrierung für die Organisation 68 % der Modellresultate in der 25 %-Grenze liegen. Nach Kalibrierung für die Organisation sind 76 % der Modellresultate innerhalb dieser Grenze (Boehm, 2000, S. 173).
- Abhängig vom Kalibrierungsverfahren und der Datenmenge liegen zwischen 39 % und 63 % der Resultate in der 25 %-Grenze (Boehm, 2000, S. 174).

Boehm (1981 und 2000) zeigt einen Faktor zwei als typische Abweichung der Schätzungen während der Planung. Dies entspricht 3 dB. CoBe soll die Planung verbessern. Damit sind 3 dB zu schwach. 1 dB ist ein sehr gutes Ergebnis, 2 dB wähle ich als Grenze für die Validität (Tabelle 38) und bezeichne dieses Kriterium als 2-dB-Grenze.

Kriterium	Bewertung	Folgerung
$0\text{dB} \leq LE \leq 1\text{dB}$	Modell valide	
$1\text{dB} < LE \leq 2\text{dB}$	Modell valide	Ursachenanalyse mit unsicheren Eingaben und unklarem Prozess
$2\text{dB} < LE$	Validität fraglich, Modell nicht valide	

Tabelle 38: Kriterien für die Validierung

Ein Paar aus Istwert und zugehörigem Modellresultat bezeichne ich im Folgenden als Datenpunkt. Ein einzelner Datenpunkt kann direkt mit der logarithmischen Abweichung beurteilt werden. Werden mehrere Projekte in jeweils eine Instanz von CoBe abgebildet, dann werden für jedes Projekt Resultate berechnet. Es müssen also mehrere Datenpunkte beurteilt werden. Um mehrere Vergleiche zusammenzufassen, verwende ich zwei Kriterien:

- Der Anteil der Datenpunkte, der innerhalb einer bestimmten Grenze liegt, wird nach IEEE 1061 (1998) und Boehm(2000) durch *pred*(x) dargestellt. Dabei bezeichnet "x" die Grenze, die verwendet wird. Ich verwende im Folgenden *pred*(2 dB); liegen zum Beispiel 7 von 10 Vergleichen innerhalb der 2-dB-Grenze, dann ist *pred*(2 dB) = 70 %.
- Den Median aus den Werten der logarithmischen Abweichung des *LE* bezeichne ich im Folgenden durch *MLE*. Ich wähle den Median und nicht den Mittelwert, weil der Median robust gegen Ausreißer ist.

Der IEEE-Standard 1061 (1998) nennt weitere Kriterien für Qualitätsmetriken (Tabelle 39):

Kriterium	Beschreibung
Correlation (Korrelation)	Qualitätsmerkmal und Metrik sollen eng, d.h. mit hoher Bestimmtheit, zusammenhängen.
Tracking (Verfolgung)	Änderungen des Qualitätsmerkmals von einem Zeitpunkt T1 zum Zeitpunkt T2 sollen sich auch als Änderungen in der richtigen Richtung in der Metrik zeigen.
Consistency (Konsistenz)	Werden Produkte anhand des Qualitätsmerkmals in einer bestimmten Reihenfolge geordnet, dann soll auch die Metrik die Produkte in dieser Reihenfolge ordnen.
Predictability (Genauigkeit)	Eine Metrik, die ein Merkmal prognostiziert, soll das Merkmal mit einer bestimmten Mindestgenauigkeit vorhersagen.
Discriminative power (Differenziertheit)	Unterschiedliche Ausprägungen eines Qualitätsmerkmals sollen auch durch die Metrik unterschiedlich bewertet werden.
Reliability (Verlässlichkeit)	Erfolgt die Validierung mit mehreren Bewertungen (etwa mit mehreren Produkten oder zu mehreren Zeitpunkten), dann soll mindestens ein bestimmter Teil der Bewertungen die oben genannten Validierungskriterien erfüllen.

Tabelle 39: Validierungskriterien (IEEE 1061, 1998)

- Correlation, Discriminative Power, Consistency: Werden Datenpunkte über einen Bereich hinweg betrachtet werden, dann werden Aussagen zur Differenziertheit, Konsistenz und Korrelation möglich. Aussagen über die Korrelation zwischen Software-Umfang und Projektaufwand sind beispielsweise nur möglich, wenn der Umfang auch tatsächlich variiert. Bleibt er gleich, dann ergibt sich keine Korrelation.
- Tracking: Werden Datenpunkte über eine Zeitspanne hinweg betrachtet, dann wird geprüft, ob sich Istwert und Modellresultat gleichartig verhalten. Zeitliche Aspekte spielen für die Validierung von CoBe keine Rolle.

7.3 Studentische Projekte für die Modellvalidierung

Die Istwerte für die Validierung einzelner Zusammenhänge und für eine erste Prüfung der Modellresultate stammen aus dem Software-Praktikum. Dieses Praktikum ist eine Pflichtveranstaltung im dritten und vierten Semester des Studiengangs Softwaretechnik der Universität Stuttgart (Ludewig et al., 2001). Die Daten stammen aus dem Praktikum 2007. Es dauerte 21 Wochen von Februar bis August. Die Prüfungsordnung gibt 720 Entwicklerstunden (Eh) Aufwand vor. Das Praktikum wurde von zwei wissenschaftlichen Mitarbeitern betreut. Ein weiterer Mitarbeiter übernahm die Kundenrolle. Er forderte ein Werkzeug zur Erfassung und Verwaltung von Testfällen und Testprotokollen.

7.3.1 Ablauf, Datenerhebung und Datenvalidierung

Im Praktikum arbeiteten in der Regel Dreierteams an dieser Aufgabe. In Ausnahmefällen wurden Zweierteams gebildet. 25 Teams haben das Praktikum begonnen, 23 Teams haben es erfolgreich abgeschlossen, Daten aus 21 Teams liegen vor. Das Praktikum folgte einem definierten Prozess mit vorgegebenen Schritten und Terminen (Tabelle 40). Spezifikation und Entwurf erfolgten mit UML. Der Code wurde in Java geschrieben. Der Modultest erfolgte mit JUnit. Der Systemtest wurde ohne Werkzeugunterstützung durchgeführt. Die Teilnehmer mussten ihr eigenes Produkt zur Verwaltung und Dokumentation der Testfälle und des Tests einsetzen.

Prozessschritte	Abgaben	Termin
Einführungsveranstaltung	-	12.02.2007
Vorbereitung auf Kundenbefragung	-	14.02.2007
Kundenbefragung	-	14.02.2007
Projektplanung und Analyse	Projektplan, Analysenotizen	23.02.2007
Spezifikation	Spezifikation	09.03.2007
Spezifikationsreview	Reviewprotokoll	16.03.2007
Korrektur Spezifikation	Spezifikation	23.03.2007
Entwurf, Review und Korrektur	Entwurf	13.04.2007
Walkthrough mit Betreuer	Präsentation des Entwurfs	19./20.04.2007
Implementierung	Code	01.06.2007
Modultest und Korrektur	Testprotokoll und Code	15.06.2007
Systemtest und Korrektur	Testprotokoll und Code	29.06.2007
Abnahme	Alle Dokumente und Code	05./06.07.2007
Korrektur	Korrigierte Dokumente und Code	20.07.2007

Tabelle 40: Ablauf des Software-Praktikums mit vorgegebenen Terminen

Die Betreuer haben Vorgaben für einen Teil der Prüfungen gemacht (Tabelle 41). Die Anweisungsüberdeckung im Systemtest musste mit dem Werkzeug EMMA gemessen werden. Anweisungen sind in EMMA über den Bytecode von Java definiert.

Tabelle 42 zeigt die erhobenen Daten. Die Umfangsmetriken konnten nachträglich gemessen werden. Dabei wurde Code für Testfälle und wiederverwendeter Code nicht gezählt. Auch Aufwände konnten gemessen werden. Andere Metriken mussten subjektiv klassifiziert und bewertet werden. Dazu gehören die Abdeckung der einzelnen Testmethoden im Black-Box-Test, Fehlerschwere, Fehlerart und die Klassifikationen für die Fehlerfolgekosten. Angelehnt an Prechelt (2001) wurden Kriterien für eine

Prüfung	Vorgaben
Spezifikations-review	Begutachtung der Spezifikation mit Checkliste, im Normalfall durch vier Gutachter und einen Moderator aus anderen Teams. Die Organisationen mit Zeiten und Räumen war vorgegeben.
Entwurfs-review	Begutachtung des Entwurfs mit Checkliste, im Normalfall mit den gleichen Beteiligten wie im Spezifikationsreview. Die Organisationen musste von den Teilnehmern geleistet werden.
Walkthrough des Entwurfs	Der Entwurf musste den Betreuern vorgestellt werden.
Programmier-richtlinie	Der Code musste einer gegebenen Programmierrichtlinie genügen. Die Teilnehmer waren für die Einhaltung selbst verantwortlich.
Modultest	Keine Vorgabe.
Systemtest	Alle Funktionen mussten getestet werden. Mindestens 90 % der Zeilen mussten ausgeführt werden.

Tabelle 41: Vorgaben für die Prüfungen im Praktikum

nachvollziehbare und einheitliche Bewertung definiert und verwendet, damit die Bewertung nachvollziehbar und einheitlich erfolgt. Kosten in Form von Geldwerten wurden im Praktikum nicht erhoben, weil in studentischen Projekten Personalkosten nicht definiert sind.

Die erhobenen Werte wurden auf Konsistenz und Glaubwürdigkeit geprüft. Beispielsweise können im Spezifikationsreview keine Codefehler entdeckt werden. Darum sollte die Zahl der entdeckten Codefehler null sein. Wurden dann tatsächlich Codefehler im Spezifikationsreview dokumentiert, dann konnte etwa mit dem Datum der Entdeckung oder mit dem Fehlerkommentar geprüft werden, ob Fehlerart, Fehlerkommentar, Prüfung und Datum konsistent sind. Zeigten sich extreme Abweichungen von typischen Werten, dann wurden die Dokumente des Projekts herangezogen. Wurden beispielsweise keine Fehler im Systemtest dokumentiert, konnte dies anhand des Testprotokolls kontrolliert werden. Wurde ein extrem hoher Korrekturaufwand für einen Fehler dokumentiert, konnte dies mit dem Kommentar, der für den Fehler dokumentiert wurde, überprüft werden. Prechelt (2001) bezeichnet diesen Schritt als Datenvalidierung.

7.3.2 Interne und externe Validität

Bei der Erprobung des Modells wird das Software-Praktikum stellvertretend für alle Software-Projekte, für die das Modell gültig sein soll, verwendet. Damit ähnelt die Erprobung und auch die Validierung des Modells einem Experiment, obwohl keine unabhängige Variable manipuliert wird, weil eine Stichprobe (das Software-Praktikum) stellvertretend für die Gesamtpopulation (alle Projekte, für die das Modell gül-

Aktivität	Metriken
Korrektur	Die Teilnehmer dokumentierten für jeden Fehler den Korrekturaufwand, die Fehlerart, die Fehlerschwere und die Prüfung[a]. Diese Merkmale sind die konsistent mit CoBe definiert (Abschnitt 6.3.2).
Spez.-review	Der Moderator dokumentierte die Namen der Gutachter (und damit die Zahl der Gutachter) und den Vorbereitungsaufwand pro Gutachter.
System-test	Im Testprotokoll steht die Zahl der Testfälle, für jeden Testfall der Erfolg, der Durchführungsaufwand und die Anweisungsüberdeckung[b].
	Pro Testfall wurden nachträglich Anweisungs-, Zweig-, Term- und Schleifenüberdeckung gemessen[c] und die kumulierte Fehlerzahl[d] berechnet.
	Die Abdeckung der Methoden für den Black-Box-Test wurde anhand des Testprotokolls bewertet.
Software-Umfang	Seitenzahl der Spezifikation (mit Begriffslexikon) vor dem Review
	Seitenzahl des Entwurfs vor dem Walkthrough
	Zahl der Anweisungen und Zeilen der Endabgabe[e]
Fehlerfolgekosten	Fehler wurden nachträglich nach Schaden, Auftretenswahrscheinlichkeit und Verwendungshäufigkeit klassifiziert.[f]

Tabelle 42: Metriken im Software-Praktikum

a. Gemessen mit dem Werkzeug JDefectCollector (Hampp und Knauß, 2008)
b. Gemessen mit dem Werkzeug EMMA
c. Gemessen mit dem Werkzeug CodeCover
d. Die Zahl der Fehler, die bis zu diesem Testfall entdeckt wurden
e. Gemessen mit dem Werkzeug CodeCount
f. Die Auftretenswahrscheinlichkeit beschreibt, mit welcher Wahrscheinlichkeit ein Fehler bei einer Verwendung auftritt. Die Verwendungshäufigkeit beschreibt, wie oft die Software beim Einsatz verwendet wird, bis ein Fehler korrigiert wird (Abschnitt 6.3.10)

tig sein soll) verwendet wird. Zur Gesamtpopulation gehören vor allem Industrieprojekte. Darum stellt sich die Frage nach der Gültigkeit der gewonnenen Aussagen (Prechelt, 2001):

- Die innere Gültigkeit oder interne Validität ist durch unkontrollierte Variablen bedroht.
- Die äußere Gültigkeit oder externe Validität beschreibt, ob und in welchem Grad sich die Resultate auf andere Situationen, in denen das Modell eingesetzt werden soll, übertragen lassen.

Interne Validität. Alle Teams mussten die gleichen Vorgaben für den Prozess und die Prüfungen erfüllen. Die Aufgabe war für alle Teams gleich. Sie befragten gemeinsam den Kunden. Die spezifizierten Anforderungen waren stabil. Dadurch entfallen viele unkontrollierte Variablen, so dass die Projekte gut vergleichbar sind. Der gleich-

förmige Prüfprozess hat aber den Nachteil, dass Unterschiede durch verschiedene Prüfparameter nur schwer gezeigt werden können. Damit wird es schwierig, den Nutzen von Prüfungen zu zeigen, weil der Nutzen als Kostendifferenz sichtbar wird. Die Teilnehmer waren unterschiedlich motiviert und fähig. Sie konnten ihren Prozess und ihr Produkt im gegebenen Rahmen selbst bestimmen. Motivation und Fähigkeit wurden nicht, Prüfungen, Prozess- und Produktmerkmale zumindest teilweise kontrolliert. Die große Anzahl der Teams erleichtert die Analyse, weil Aussagen über Mittelwerte und die Streuung um den Mittelwert möglich sind. Sie stärkt die interne Validität, weil das Ergebnis durch die vielen Datenpunkte robuster gegen zufällige Störeffekte ist.

Externe Validität. Die Verallgemeinerbarkeit leidet unter dem relativ kleinen und einfachen Produkt. Projekt und Produkt sind aber komplett. Solche kleinen Produkte gibt es auch in der Praxis (Jones, 1996). Die Anforderungen im Praktikum sind stabil, eine Situation, die nicht typisch für die Praxis ist. Dieser Aspekt spielt in den Modellannahmen aber keine direkte Rolle. Die Teilnehmer des Praktikums sind im Vergleich zu Entwicklern in der Industrie eher unerfahren, weil das Praktikum im dritten Semester beginnt. Damit ist es für viele Teilnehmer das erste vollständige Software-Projekt, das sie durchführen. Diese Bedrohung der Validität wird durch die gleichartige Vorbereitung der Teilnehmer abgeschwächt: Entwurf und Programmierung sind die Themen der Lehrveranstaltungen "Programmierkurs" und "Programmentwicklung" im ersten und dritten Semester. In der Vorlesung "Einführung in die Softwaretechnik I" haben sie die Schritte eines Projekts kennengelernt und geübt (Ludewig et al., 2001). Eine weitere Bedrohung der Verallgemeinerbarkeit ist der fehlenden Kostenbegriff in studentischen Projekten, weil in studentischen Projekten kein Geld fließt, während in Industrieprojekten die Kosten in Geldwerten anfallen. In beiden Fällen können aber Arbeitsaufwand (z.B. in Entwicklerstunden, Eh), Dauer und Personalbedarf gemessen werden.

7.4 Prüfung ausgewählter Modellzusammenhänge

In dieser Arbeit können nicht alle Annahmen und Zusammenhänge von CoBe geprüft werden, weil dazu die verfügbare Zeit nicht ausreicht. Ich wähle drei Bereiche aus: Erstens die Zusammenhänge zwischen Fehlerentstehung, Fehlerentdeckung und Korrekturaufwand, zweitens die Zusammenhänge zwischen Testfällen, Überdeckung und Fehlerentdeckung im Systemtest und drittens die Bewertung der Fehlerfolgekosten.

Die Zusammenhänge zwischen Fehlerentstehung, Fehlerentdeckung und Korrekturaufwand werden geprüft, weil sie die Basis des Modells bilden. Für diese Basis gibt es nur wenige Daten aus jüngeren Studien (Boehm, 1976, 1981 und 1987; Basili und Perricone, 1984; Humphrey, 1995, Kan, 2003). Zum Teil werden Daumenregeln zusammengefasst (Shull, 2002). Die Zusammenhänge im Systemtest zwischen Testfällen, Überdeckungen und Fehlerentdeckung werden geprüft, weil sie nur punktweise empirisch belegt sind (Abschnitt 5.5) und Daten auf breiter empirischer Basis fehlen.

Diese Zusammenhänge werden statistisch geprüft, dazu wird jeder Zusammenhang als Arbeitshypothese formuliert. Eine Arbeitshypothese kann nicht direkt bestätigt werden. Stattdessen wird eine Nullhypothese formuliert, die die Arbeitshypothese negiert. Diese Nullhypothese kann falsifiziert werden. Wird die Nullhypothese falsifiziert, dann wird dies als "Verwerfen" oder "Abweisen" der Nullhypothese bezeichnet. Dadurch wird die Arbeitshypothese bestätigt.

Nullhypothesen werden durch statistische Tests geprüft. Sind die zu untersuchenden Daten normalverteilt, dann kann der t-Test verwendet werden. Für beliebige Verteilungen der untersuchten Daten kann der Wilcoxon-Rangsummen-Test verwendet werden (Prechelt, 2001; Fahrmeir et al., 2007). Diese Tests ergeben, ob die Nullhypothese statistisch signifikant abgewiesen werden kann. Dabei bedeutet statistisch signifikant, dass die Wahrscheinlichkeit für einen Irrtum ausreichend gering ist. Es wird also betrachtet, mit welcher Wahrscheinlichkeit die Nullhypothese fälschlicherweise abgewiesen wird. Diese Wahrscheinlichkeit wird durch den *p*-Wert dargestellt (Fahrmeir et al., 2007). *p*-Werte sind Wahrscheinlichkeiten und liegen darum zwischen 0 und 1. Je geringer der *p*-Wert, desto weniger wahrscheinlich ist, dass die Nullhypothese fälschlicherweise abgewiesen wird. Ein kleiner *p*-Wert spricht also für ein vertrauenswürdiges Ergebnis. Für eine objektive Bewertung wird der *p*-Wert mit einem geforderten α-Wert verglichen. Der α-Wert ist das geforderte Signifikanzniveau, typische Werte sind 0,1, 0,05 oder 0,01 (10 %, 5 % oder 1 %). Für die Signifikanz *p* wird im Folgenden ein 5 %-Niveau gefordert ($\alpha = 0{,}05$). Mit diesem Signifikanzniveau soll die Wahrscheinlichkeit für einen Irrtum also unter 0,05 liegen. Sobald der *p*-Wert unter dem α-Wert von 0,05 liegt, wird die Nullhypothese abgelehnt und die Arbeitshypothese bestätigt.

Die Stärke eines Zusammenhangs, im Folgenden auch als Bestimmtheit bezeichnet, wird durch das Bestimmtheitsmaß, den Determinationskoeffizienten R^2, gemessen. Er drückt aus, welcher Anteil der Streuung der abhängigen Variable durch die unabhängige erklärt wird (Fahrmeir et al., 2007). Dabei gilt 50 % bereits als guter Wert im wirtschaftswissenschaftlichen Bereich. Humphrey (1995) fordert über 70 %. Für die Messung des Bestimmtheitsmaßes wird eine lineare Regression durchgeführt (Fahrmeir et al., 2007), durch die der Zusammenhang zwischen zwei Variablen als Gerade geschätzt wird. Die Berechnungen erfolgen mit den Statistikpaketen SPSS (SPSS, 2008) und R (R, 2008).

Der dritte Bereich, der untersucht wird, ist die Bewertung der Fehlerfolgekosten, also derjenigen Kosten, die beim Produkteinsatz durch Fehler für den Kunden anfallen. Für diesen Bereich fehlen gemessene Vergleichswerte, weil die Produkte des Praktikums nicht produktiv eingesetzt wurden; nur ein Produkt wurde erprobt und demonstriert. Darum kann nur beurteilt werden, ob die Modellresultate plausibel erscheinen und untereinander konsistent sind.

7.4.1 Fehlerentstehung, Fehlerentdeckung, Korrekturaufwand

Die Zusammenhänge zwischen Fehlerentstehung, Fehlerentdeckung und Korrekturaufwand lassen sich mit drei Hypothesen beschreiben (Hampp und Knauß, 2008):

H1: Prüfungen entdecken nur Fehler, die auf der gleichen Abstraktionsebene oder auf einer tieferen Abstraktionsebene der Entwicklung gemacht wurden.

H2: Je länger ein Fehler unentdeckt bleibt, desto aufwändiger ist seine Korrektur.

H3: Je schwerwiegender ein Fehler ist, desto aufwändiger ist seine Korrektur.

Prüfung H 1: Abstraktionsebene der Fehlerentdeckung.

Tabelle 43 zeigt die im Software-Praktikum erhobenen Fehlerzahlen, unterschieden nach Spezifikations-, Entwurfs- und Codefehlern. Diese Fehlerarten sind konsistent mit CoBe definiert (Abschnitt 6.3.2).

Zahl entdeckter Fehler pro Fehlerart	Spez.-fehler	Entwurfsfehler	Codefehler
Spezifikationsreview	572	0	0
Entwurfsreview	1	209	0
Modultest	0	4	107
Systemtest	13	3	150
Abnahme	0	0	7

Tabelle 43: Zahl entdeckter Fehler im Praktikum, getrennt nach Fehlerart

In jeder Prüfung werden Fehler vor allem einer bestimmten Art entdeckt. Spezifikationsfehler werden hauptsächlich im Spezifikationsreview entdeckt, dann erst wieder im Systemtest. Sie werden nicht auf niedrigerer Abstraktionsebene entdeckt. Entwurfsfehler werden vor allem im Entwurfsreview entdeckt. Ein kleiner Teil der Entwurfsfehler wird im Modultest auf niedriger Abstraktionseben entdeckt, ein Teil erst im Systemtest auf höherer Abstraktionsebene. Codefehler werden sowohl im Modultest, also auf der gleichen Abstraktionsebene, als auch im Systemtest, also durch Test auf höherer Abstraktionsebene, entdeckt.

Bewertung. Die Resultate widerlegen die Hypothese H 1 von CoBe nicht, sondern bestätigen sie. Besonders deutlich wird dies, weil keine Spezifikationsfehler auf niedriger Abstraktionsebene entdeckt werden. Dass einige Entwurfsfehler bereits im Modultest entdeckt werden, führe ich darauf zurück, dass die Teilnehmer bereits während des Modultests kontinuierlich integriert haben. Somit enthält der Modultest bereits einen Integrationstest, der auf höherer Abstraktionsebene liegt. Auf einen statistischen Test verzichte ich, weil nur wenige der Spezifikations- und Entwurfsfehler in Modul- und Systemtests entdeckt wurden. Die interne Validität ist durch diese geringe Fehlerzahl bedroht. Die externe Validität ist bedroht durch die stabilen

Anforderungen; Jones (1996) bestätigt quantitativ den Verlauf der Fehlerentdeckung mit Industriedaten.

Folgerung. Damit werden die unterschiedlichen Fehlerentdeckungsquoten für Fehlerarten und für Prüfungen, die in CoBe verwendet werden, qualitativ bestätigt. Bestätigt wird, dass es in CoBe notwendig ist, die unterschiedlichen Fehlerarten zu modellieren.

Prüfung H 2: Je länger ein Fehler unentdeckt bleibt, desto aufwändiger ist seine Korrektur.

Die Abbildung 55 zeigt den Korrekturaufwand pro Fehler nach den verschiedenen Prüfungen im Software-Praktikum. Rechts sind die Box-Plots ohne extreme Ausreißer[1] dargestellt. Der Korrekturaufwand steigt deutlich im Verlauf des Projekts. Die Aufwände streuen stark, es gibt extreme Ausreißer nach oben.

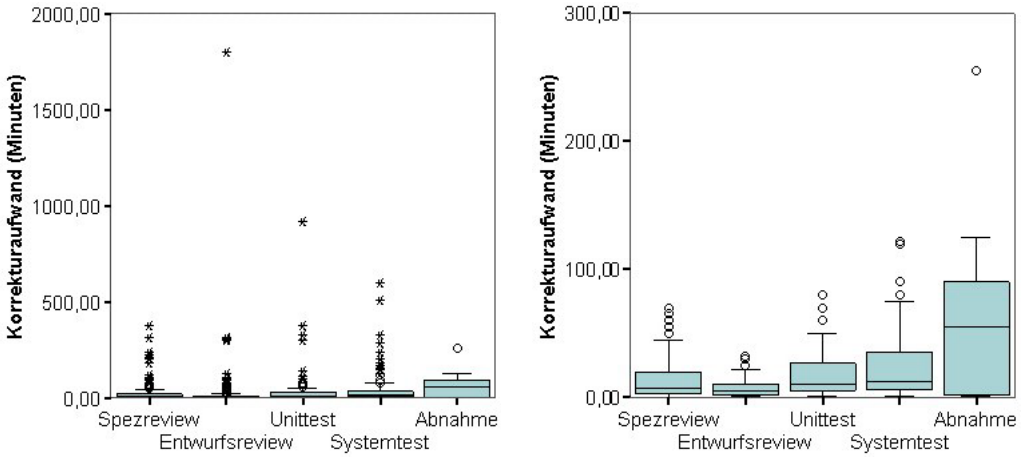

Abb. 55: Korrekturaufwand pro Fehler nach Prüfungen im Praktikum: links mit, rechts ohne extreme Ausreißer

Damit die Hypothese H 2 geprüft werden kann, muss bestimmt werden, wie lange ein Fehler unentdeckt bleibt. Diese Zeitdauer hängt vom Entstehungszeitpunkt und vom Entdeckungszeitpunkt ab. Sie wird auch als Latenzzeit eines Fehlers bezeichnet. Der Entstehungszeitpunkt ist durch die Fehlerart bestimmt, die konsistent zu CoBe definiert ist (Abschnitt 6.3.2). Der Entdeckungszeitpunkt ist durch die Prüfung bestimmt, mit der der Fehler entdeckt wird. Der Einfluss der Latenzzeit wird durch zwei Hypothesen geprüft. In diesen Hypothesen werden Spezifikations- und Entwurfsfehler als frühe Fehler zusammengefasst, weil in den Tests nur wenige dieser frühen Fehler entdeckt wurden. Codefehler werden als späte Fehler bezeichnet. Die frühe Korrektur findet nach Reviews statt, die späte Korrektur nach den Tests.

1. Als extreme Ausreißer oder Extremwerte werden im Boxplot diejenigen Werte bezeichnet, die mehr als 3 Boxlängen vom Rand der Box entfernt sind (SPSS, 2008). Die Boxlänge ist der Abstand zwischen unterer und oberer Quartile.

H 2.1: Die späte Korrektur früher Fehler ist aufwändiger als die frühe Korrektur.

H 2.2: Die späte Korrektur früher Fehler ist aufwändiger als die Korrektur später Fehler.

Tabelle 44 fasst die Daten über die Korrekturaufwände pro Fehler zusammen. Der Mittelwert ist deutlich höher als der Median. Die Maximalwerte erreichen Korrekturaufwände von bis zu 30 Entwicklerstunden pro Fehler. Dies deutet darauf hin, dass die Aufwände nicht normalverteilt sind. Für den Hypothesentest wird darum der Wilcoxon-Rangsummen-Test eingesetzt.

Aufwand (Entwicklerminuten)	Frühe Fehler in Reviews	Frühe Fehler in Tests	Späte Fehler in Tests
Minimum	0,0	4,0	1,0
1. Quartil	3,0	10,0	5,0
Median	6,0	37,5	10,0
Mittelwert	18,7	131,8	32,0
3. Quartil	15,0	150,2	30,0
Maximum	1800,0	918,0	601,0
Std.-Abweichung	73,2	222,7	63,4

Tabelle 44: Korrekturaufwand nach Prüfungen im Praktikum

Im Praktikum waren die frühen Fehler etwa 6 mal so teuer zu korrigieren, wenn sie in den Tests entdeckt wurden. Der Unterschied ist statistisch signifikant (p-Wert < 0,001[1]), so dass die Nullhypothese für H 2.1 abgewiesen werden kann. Die frühen Fehler waren in den Tests etwa 4 mal so teuer zu korrigieren wie die späten Fehler. Der Unterschied ist signifikant (p-Wert = 0,004), so dass die Nullhypothese für H 2.2 abgewiesen werden kann.

Bewertung. Die Hypothese H 2 von CoBe wird durch die Daten des Praktikums bestätigt. Die interne Validität ist bedroht, weil im Praktikum wenig frühe Fehler spät entdeckt wurden und Spezifikations- und Entwurfsfehler für den Hypothesentest zusammengefasst wurden. Die externe Validität ist durch den Produktumfang, das einfache Produkt und die intensiven frühen Prüfungen eingeschränkt. Die Resultate entsprechen aber den bisherigen Beobachtungen. Der Anstieg der Korrekturkosten fällt geringer aus als bei großen Projekten (1 : 6 statt 1 : 10), dies ist konsistent mit Daten in Boehm (1976 und 1981).

1. Die Wahrscheinlichkeit, dass die Nullhypothese fälschlicherweise abgewiesen wird, liegt unter 0,001 und damit unter dem geforderten Signifikanzniveau von 0,05.

Folgerungen. Die Ergebnisse bestätigen den in CoBe modellierten Zusammenhang zwischen Latenzzeit und Korrekturaufwand. Es wird bestätigt, dass der Entstehungszeitpunkt und der Entdeckungszeitpunkt eines Fehlers modelliert werden müssen.

Prüfung H 3: Je schwerwiegender ein Fehler ist, desto aufwändiger ist seine Korrektur.

Die Fehlerschwere ist im Praktikum gleich wie in CoBe definiert, es gibt kritische Fehler, Hauptfehler und Nebenfehler (Abschnitt 6.3.2). Kritische Fehler kosten im Praktikum mehr Korrekturaufwand als Hauptfehler. Hauptfehler kosten mehr Korrekturaufwand als Nebenfehler (Abbildung 56). Für einen statistischen Test wird die Hypothese aufgeteilt:

H3.1: Hauptfehler sind aufwändiger zu korrigieren als Nebenfehler.

H3.2: Kritische Fehler sind aufwändiger zu korrigieren als Nebenfehler.

H3.3: Kritische Fehler sind aufwändiger zu korrigieren als Hauptfehler.

Der Hypothesentest erfolgt mit dem Wilcoxon-Rangsummen-Test. Jede Prüfung im Praktikum wird einzeln betrachtet. Nur die Fehler derjenigen Fehlerart werden untersucht, die vorrangig in der Prüfung entdeckt wird. Alle Hypothesen für das Spezifikationsreview können bestätigt werden. Für das Entwurfsreview kann die Hypothese für den Unterschied zwischen Nebenfehlern und anderen Fehlern bestätigt werden, aber nicht für den Unterschied zwischen kritischen und Hauptfehlern. Dies kann aber auf die geringe Zahl an kritischen Fehlern zurückgeführt werden: Im Entwurfsreview waren 20 Fehler von insgesamt 209 Entwurfsfehlern kritisch. Die Hypothesen für Tests können nicht bestätigt werden (Tabelle 45).

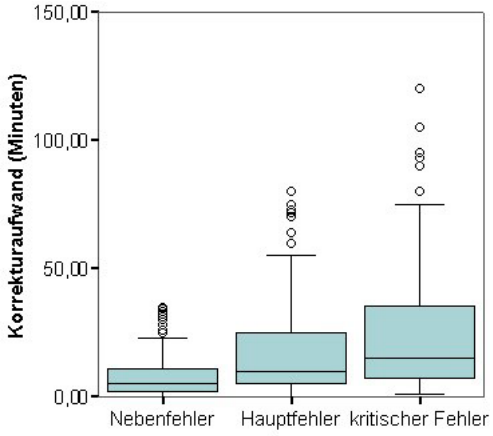

Abb. 56: Korrekturaufwand im Praktikum für unterschiedlich schwere Fehler

Bewertung. Die Ergebnisse für die Korrektur nach Tests widersprechen Industriedaten (Kan, 2003). Die interne Validität ist beeinträchtigt, weil die Teilnehmer wenig Erfahrung bei der Klassifikation der Fehler haben. Dies beeinträchtigt auch die

p-Wert (Signifikanz)[a] der Hypothesen pro Prüfung	H3.1	H3.2	H3.3
Spezifikationsreview und Spezifikationsfehler	**< 0,001**	**< 0,001**	**0,001**
Entwurfsreview und Entwurfsfehler	**0,016**	**< 0,001**	0,079
Modultest und Codefehler	0,487	**0,021**	0,114
Systemtest und Codefehler	0,076	0,575	0,344

Tabelle 45: Hypothesentests für den Einfluss der Fehlerschwere

a. Statistisch signifikante Ergebnisse (5 %-Niveau) sind **fett** gedruckt. Die Wahrscheinlichkeit, dass die Nullhypothese fälschlicherweise abgewiesen wird, liegt dabei unter dem geforderten Signifikanzniveau von 0,05.

externe Validität. Werden alle Fehler im Praktikum betrachtet, dann unterscheiden sich die Korrekturaufwände abhängig von der Fehlerschwere. Dies zeigt sich auch in Industriedaten (Kan, 2003; Zage und Zage, 2003).

Folgerungen. Der Zusammenhang zwischen Fehlerschwere und Korrekturaufwand, der in CoBe modelliert ist, wird für Reviews bestätigt. Er wird für alle Fehler bestätigt, aber nicht einzeln für die Korrektur nach Tests. Weil dazu aber Industriedaten vorhanden sind (Kan, 2003), wird CoBe nicht verändert.

7.4.2 Testfälle, Code-Überdeckung, Fehlerentdeckungsquote

Die Hypothesen der Zusammenhänge für Testfälle, Überdeckung und Fehlerentdeckung im Test beziehen sich auf das funktionale Modell und seine Quantifizierung. Die erste Hypothese über den Test betrifft die Form des Zusammenhangs zwischen Testfallzahl und Überdeckungsgrad:

H4: Der Zusammenhang zwischen der normierten Zahl der Testfälle t und dem Überdeckungsgrad c kann durch einen Zusammenhang der Form $c = \min(1, r_{0c} \cdot t^{r_{1c}})$ dargestellt werden.

Abbildung 57 skizziert den Zusammenhang der Hypothese H 4; die Überdeckung bleibt bei 100 % konstant, auch wenn weitere Testfälle spezifiziert und durchgeführt werden.

Im Modell wird die Testfallzahl durch den Produktumfang normiert; ich nehme also an dass es eine typische Zahl von Testfällen pro Function Point gibt, angelehnt an Jones (2007). Diese Zahl wird zur Normierung verwendet; dazu müssen Function Points für die Art der Software geeignet sein (Abschnitt 6.3.2). Diese Voraussetzung wird im Praktikum erfüllt, weil das Produkt ein Werkzeug zur Testfallverwaltung ist. Ob diese Normierung den Zusammenhang zwischen Testfallzahl und Überdeckung verfälscht, wird mit der folgenden Hypothese untersucht:

H5: Durch die Normierung der Testfallzahl wird der Zusammenhang mit den Überdeckungsgraden nicht schwächer.

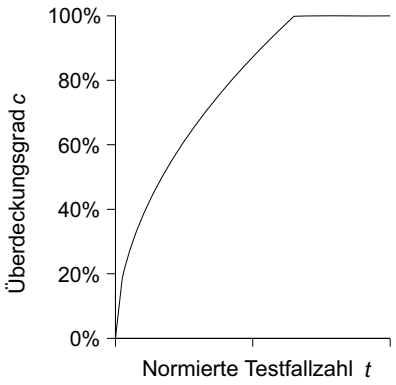

Abb. 57: Normierte Testfallzahl und Überdeckungsgrad

Das Testmodell von CoBe beruht für die Anweisungs-, Zweig-, Schleifen- und Termüberdeckung auf folgender Annahme: Die Überdeckungen hängen linear zusammen; Diesen linearen Zusammenhang zeigt Abbildung 58. Dieser Zusammenhang gilt für den Bereich zwischen 0 % und 100 % Überdeckung.

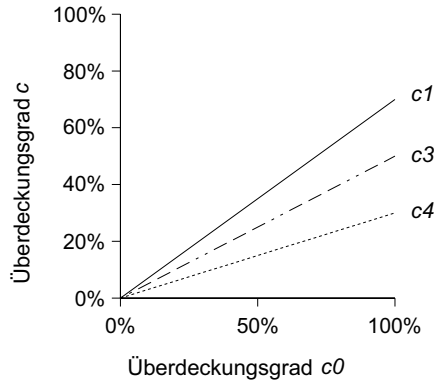

Abb. 58: Anweisungsüberdeckung und andere Überdeckungsgrade

Beispielsweise ist der Zusammenhang für die Anweisungsüberdeckung $c_0 = \min(1, r_{0c} \cdot t^{r_{1c}})$ und für die Zweigüberdeckung $c_1 = \min(1, cf_1 \cdot r_{0c} \cdot t^{r_{1c}})$ (Abschnitt 6.6.2). Die Berechnung der Zweigüberdeckung unterscheidet sich nur durch den Faktor cf_1 von der Berechnung der Anweisungsüberdeckung. Somit muss die folgende Hypothese geprüft werden:

H6: Die Überdeckungsmetriken hängen untereinander linear zusammen.

Außerdem prüfe ich zusätzlich, ob sich die Modellresultate mit einem solchen linearen Modell verschlechtern und ob ein nichtlinearer Zusammenhang geeigneter ist:

H7: Für die Berechnung der Überdeckungsgrade für Zweige, Schleifen und Terme ist dieser lineare Zusammenhang gut geeignet.

Diese Annahme gilt aber nur unter bestimmten Voraussetzungen und in einem bestimmten Bereich, der durch die vollständige Überdeckung nach oben begrenzt wird. Sobald mit einer Metrik 100 % Überdeckung erreicht wird, können andere, unvollständige Überdeckungen trotzdem vervollständigt werden. Der lineare Zusammenhang gilt dann also nicht mehr. Abbildung 59 skizziert dies: Während die Zweigüberdeckung (*c1* in der Abbildung) weiter ansteigt, bleibt die Anweisungsüberdeckung (*c0* in der Abbildung) konstant, sobald 100 % erreicht werden. Unklar ist, ob dieser Zusammenhang nicht mehr gilt, wenn eine bestimmte Überdeckung gezielt erreicht werden soll.

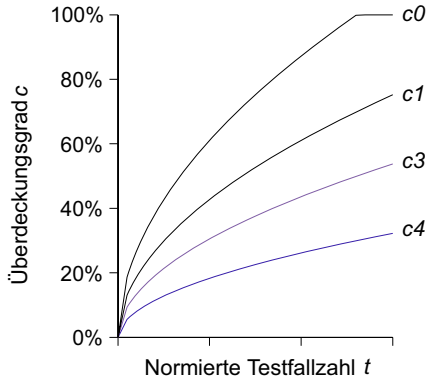

Abb. 59: Normierte Testfallzahl und verschiedene Überdeckungsgrade

Die fünfte Annahme des Testmodells betrifft die Fehlerentdeckung. Das Modell basiert auf der Annahme, dass die Zahl der Testfälle die Fehlerentdeckungsquote nicht-linear bestimmt:

H8: Der Zusammenhang zwischen der normierten Anzahl der Testfälle t und der Fehlerentdeckungsquote Q hat die Form $Q_{Test} = \max(0, 1 - r_{\mathrm{qt}}(1 - q_{\mathrm{t}})^{t_{Test}})$.

Abbildung 60 skizziert diesen Zusammenhang, mit dem sich die Fehlerentdeckungsquote asymptotisch 100 % nähert, wenn sehr viele Testfälle durchgeführt werden.

Für den Test der Hypothesen 7 und 8 werden die Daten der Teams im Praktikum gleichmäßig auf zwei Datengruppen aufgeteilt (Abbildung 61):

- Die Daten der Datengruppe 1 werden verwendet, um die Zusammenhänge zu quantifizieren. Beispielsweise werden im Folgenden die Parameter r_{qt} und q_{t} (Hypothese 8) mit Daten der Datengruppe 1 durch Regression bestimmt. Mit dieser Quantifizierung werden Modellresultate für die Datengruppe 1 berechnet, um Aussagen über die Signifikanz und Bestimmtheit zu treffen.

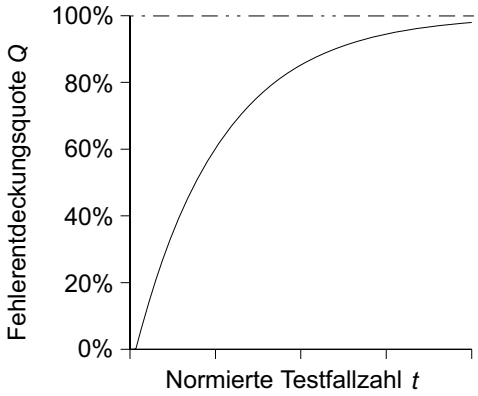

Abb. 60: Normierte Testfallzahl und Fehlerentdeckungsquote

- Die Daten der Datengruppe 2 werden verwendet, um die quantifizierten Zusammenhänge zu prüfen: Die Quantifizierung beruht auf der Datengruppe 1. Mit dieser Quantifizierung werden Modellresultate mit Eingaben aus der Datengruppe 2 berechnet. Beispielsweise wird die Fehlerentdeckungsquote aus der normierten Testfallzahl berechnet. Die Resultate werden mit den Istwerten der Datengruppe 2 verglichen. Somit können Aussagen über die Prognosefähigkeit getroffen werden.

Die Teams sind einer Datengruppe durch Zufallsauswahl zugeordnet. 11 Teams sind in Datengruppe 1, 10 Teams in Datengruppe 2.

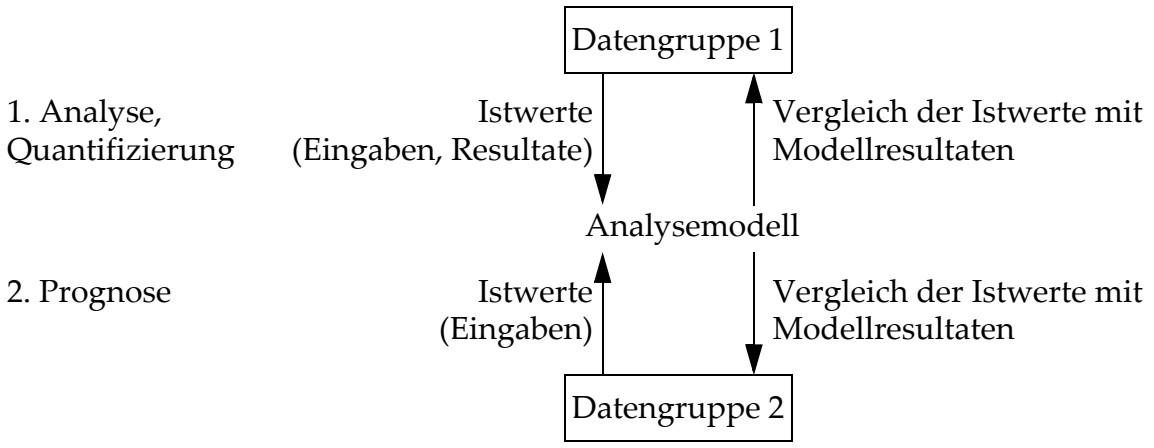

Abb. 61: Analyse und Prognose zur Hypothesenprüfung

Um die Überdeckungsgrade zu messen, wurde der Systemtest mit dem instrumentierten Programm anhand der Testprotokolle wiederholt. Überdeckungsgrade, Testfallzahl und die kumulierte Fehlerzahl aus dem Testprotokoll wurden notiert. Falls

vorhanden, wurde die Abgabe vor dem Systemtest verwendet, weil dies der realen Situation des Tests entspricht.

Prüfung H 4: Der Zusammenhang zwischen der normierten Zahl der Testfälle und dem Überdeckungsgrad kann durch einen Zusammenhang der Form $c = \min(1, r_{0c} \cdot t^{r_{1c}})$ dargestellt werden.

Der postulierte Zusammenhang wird in Abbildung 57 (Seite 169) skizziert. Die Hypothese wird durch eine lineare Regression mit logarithmierten Daten geprüft. Dazu berechnet die Regression die Parameter r_{0c} und r_{1c}, mit denen der funktionale Zusammenhang zwischen normierter Testfallzahl und Überdeckung quantifiziert wird. Abbildung 62 veranschaulicht diesen Zusammenhang mit Daten aus dem Praktikum (Datengruppe 1). Die Diagramme stellen die Daten mehrerer Teams gemeinsam dar, um die Streuung der Daten zu zeigen.

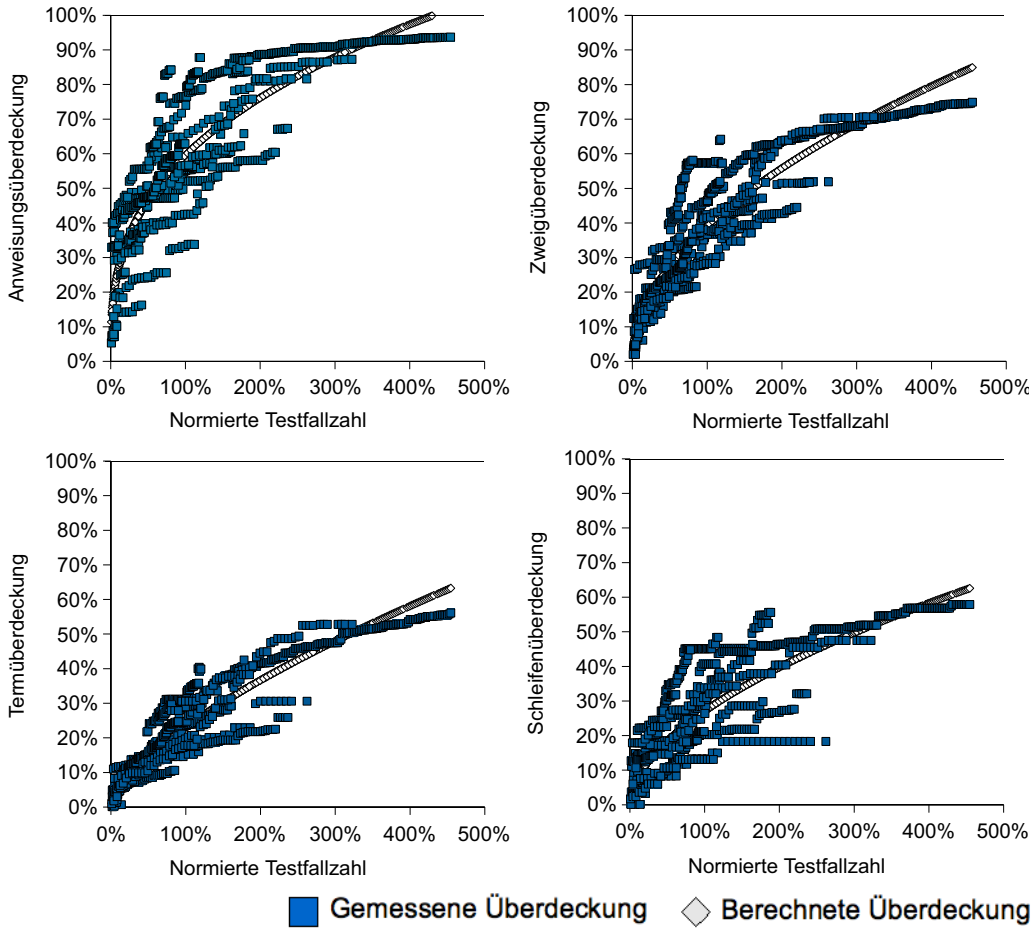

Abb. 62: Normierte Testfallzahl und Überdeckungsgrad im Praktikum

- Die x-Achse zeigt die normierte Testfallzahl, weil in CoBe mit normierten Testfallzahlen gerechnet wird. Dazu wird Zahl der durchgeführten Testfälle mit der typischen Testfallzahl normiert. Die typische Testfallzahl basiert auf dem Mittelwert von Jones (2007) und ist als 100 %-Wert dargestellt.
- Die y-Achse zeigt den Überdeckungsgrad in Prozent für eine bestimmte Überdeckungsmetrik, oben links zum Beispiel für die Anweisungsüberdeckung.
- Jeder dunkle Punkt im Diagramm repräsentiert einen Testfall aus dem Praktikum: Mit diesem Testfall wurden vom Team insgesamt x % der normierten Testfallzahl durchgeführt, mit diesem Testfall wurden y % Überdeckung erreicht.
- Die hellen Punkte stellen die Ergebnisse der Regression dar.

Tabelle 46 zeigt Signifikanz und Bestimmtheit. Die Nullhypothese wird abgelehnt, da der p-Wert weit unter 0,05 liegt. Die Bestimmtheit ist hoch. Somit wird die Hypothese bestätigt.

Überdeckung	Faktor $r_{0c}{}'$ [a]	Exponent $r_{1c}{}'$	p-Wert[b]	R^2
Anweisungen	11,7	0,35	**< 0,001**	0,68
Zweige	3,7	0,51	**< 0,001**	0,83
Schleifen	2,1	0,55	**< 0,001**	0,62
Terme	1,1	0,66	**< 0,001**	0,81

Tabelle 46: Ergebnisse für normierte Testfallzahl und Überdeckungsgrad

a. Zur Unterscheidung zwischen Modellparametern und Regressionsparameter sind die Regressionsparameter durch ein ′ gekennzeichnet.
b. Statistisch signifikante Ergebnisse (5 %-Niveau) sind **fett** gedruckt.

Bewertung. Die Ergebnisse bestätigen die Hypothese H 4 von CoBe, da die Nullhypothese abgewiesen werden kann. Die interne Validität ist bedroht, weil nur wenige Tests sehr intensiv durchgeführt wurden. Dadurch besteht die Gefahr, dass die Regressionsanalyse durch wenige Fälle geprägt ist. Abbildung 62 zeigt dies anschaulich im oberen rechten Teil der Diagramme, in dem die Werte für sehr intensive Tests liegen; dort wird die Überdeckung tendenziell zu hoch berechnet. Die Diagramme bestätigen aber den Zusammenhang für die weniger intensiven Tests: Im Bereich mit weniger intensiven Tests, jeweils in der linken Hälfte der Diagramme, sind viele Datenpunkte vorhanden. Der Zusammenhang ist in diesem Teil trotz einer gewissen Streuung gut erkennbar. Die externe Validität ist eingeschränkt, weil alle Prüflinge in der gleichen Größenordnung liegen (2700 bis 16 000 Anweisungen), so dass keine Aussagen für größere Produkte möglich sind.

Folgerungen. Die im Testmodell von CoBe verwendete Form des Zusammenhangs zwischen Testfällen und Überdeckungsgraden wird bestätigt.

Prüfung H 5: Durch die Normierung der Testfallzahl wird der Zusammenhang mit den Überdeckungsgraden nicht schwächer.

Tabelle 47 zeigt die Bestimmtheit des Zusammenhangs von Hypothese 4 mit normierter Testfallzahl. Im Vergleich dazu wird die Bestimmtheit gezeigt, wenn die absolute statt die normierte Testfallzahl verwendet wird. Die Bestimmtheit mit normierter Testfallzahl ist höher. Alle Zusammenhänge sind signifikant (p-Wert < 0,001). Die Wahrscheinlichkeit, mit der die Nullhypothese fälschlicherweise abgewiesen wird, liegt also unter 0,001 und damit unter dem geforderten Signifikanzniveau von 0,05.

R^2 für die Überdeckung mit	normierter Testfallzahl	absoluter Testfallzahl
Anweisungen	0,68	0,62
Zweige	0,83	0,74
Schleifen	0,62	0,53
Terme	0,81	0,72

Tabelle 47: Bestimmtheit mit normierter und mit absoluter Testfallzahl für Überdeckungsgrade im Praktikum

Bewertung. Da die Bestimmtheit mit normierter Testfallzahl höher ist, ist eine Normierung mit dem Code-Umfang sinnvoll. Der Umfang hat also einen Einfluss auf die Zahl der Testfälle. Dieser Einfluss zeigt sich im Praktikum, obwohl die Programme in der gleichen Größenordnung liegen (2700 bis 16 000 Anweisungen). Die Hypothese H 5 von CoBe wird trotz der geringen Schwankung des Code-Umfangs bestätigt, da die Nullhypothese abgewiesen werden kann.

Folgerungen. Die Normierung der Testfallzahl mit dem Umfang, die in CoBe verwendet wird, ist sinnvoll und kann zur Berechnung des Überdeckungsgrads verwendet werden.

Prüfung H 6: Die Überdeckungsmetriken hängen untereinander linear zusammen.

Dieser lineare Zusammenhang ist in Abbildung 58 (Seite 169) skizziert. Um den Zusammenhang zu prüfen, verwende ich eine lineare Regression durch den Nullpunkt. Die Resultate zeigen für die Daten aus dem Praktikum (Datengruppe 1) einen starken, signifikanten Zusammenhang: Tabelle 48 zeigt die berechneten Faktoren cf_x zwischen den Überdeckungsgraden. Wie im Modell angenommen steigt die Anweisungsüberdeckung schneller als die Zweigüberdeckung. Die Termüberdeckung steigt am langsamsten. Die Bestimmtheit ist mit über 90 % hoch. Alle Zusammenhänge sind statistisch signifikant mit p-Werten unter 0,05.

Bewertung. Die Hypothese H 6 von CoBe wird bestätigt, die Nullhypothese abgelehnt. Eine einfache Quantifizierung mit konstantem Exponenten für alle Überdeckungsmetriken ist möglich. Die Beobachtung von Malaiya et al. (1994) für die Anweisungs- und Zweigüberdeckung wird für die anderen Überdeckungsmetriken

Überdeckung	Faktor cf_x'	p-Wert[a]	R^2
Zweige (cf_1')	0,69	**< 0,001**	0,97
Terme (cf_3')	0,44	**< 0,001**	0,93
Schleifen (cf_4')	0,50	**< 0,001**	0,92

Tabelle 48: Signifikanz und Bestimmtheit zwischen Überdeckungen im Praktikum

a. Statistisch signifikante Ergebnisse (5 %-Niveau) sind **fett** gedruckt. Die Wahrscheinlichkeit, dass die Nullhypothese fälschlicherweise abgewiesen wird, liegt dabei unter dem geforderten Signifikanzniveau von 0,05.

bestätigt. Die interne Validität ist wieder durch die wenigen extrem intensiven Tests bedroht. Ob die Ergebnisse verallgemeinert werden können, ist fraglich. In jedem Fall gilt die Einschränkung, dass diese Zusammenhänge nicht mehr gelten, wenn eine Überdeckung gezielt erreicht werden soll.

Folgerungen. Der in CoBe verwendete Zusammenhang zwischen der normierten Testfallzahl und den Überdeckungsgraden wird bestätigt.

Prüfung H 7: Für die Berechnung der Überdeckungsgrade für Zweige, Schleifen und Terme ist dieser lineare Zusammenhang gut geeignet.

Zur Prüfung dieser Hypothese werden zwei Zusammenhänge verglichen:

- Mit dem vollständigen Modell werden Anweisungs-, Zweig-, Term- und Schleifenüberdeckung aus der normierten Testfallzahl mit den Parametern aus Tabelle 46 berechnet.
- Das lineare Modell berechnet nur die Anweisungsüberdeckung aus der normierten Testfallzahl mit den Parametern aus Tabelle 46. Aus der Anweisungsüberdeckung werden linear Zweig-, Schleifen- und Termüberdeckung mit den Faktoren aus Tabelle 48 berechnet.

In beiden Fällen wird also ein Zusammenhang geprüft, der in Abbildung 63 skizziert ist. Im ersten Fall (vollständiges Modell) werden Faktoren und Exponenten für alle Überdeckungsgrade durch das Regressionsverfahren bestimmt. Im zweiten Fall werden für Zweig-, Schleifen- und Termüberdeckung (*c1*, *c3*, *c4* in der Abbildung) die jeweiligen Faktoren, aber nicht die Exponenten mit dem Regressionsverfahren bestimmt. Der Exponent wird aus der Regression der Anweisungsüberdeckung (*c0* in Abbildung 63) verwendet. Der Fall, dass die Anweisungsüberdeckung von 100 % erreicht wird (rechts oben in der Abbildung), kommt in den Daten aus dem Praktikum nicht vor.

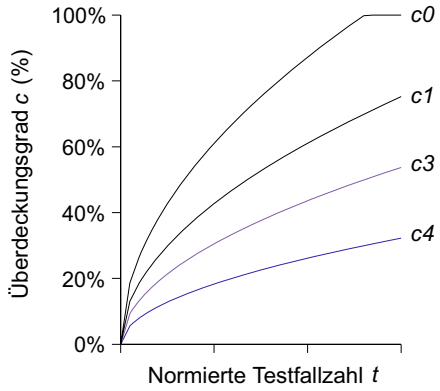

Abb. 63: Testfälle und Überdeckungsgrade

Die Datengruppe 1 wurde zur Analyse und Bewertung des Zusammenhangs verwendet. Ihre Regressionsanalyse ergibt die Parameterwerte für die Modelle. Diese Parameter werden dann verwendet, um die Überdeckungsgrade der Datengruppe 1 zu berechnen. Wie gut diese Berechnung die Istwerte trifft, wird mit dem Bestimmtheitsmaß R^2 bewertet.

Die Analysemodelle zeigen, dass die Überdeckungsgrade durch das lineare Modell in gleichem Maße wie durch das vollständige Modellen bestimmt sind. Tabelle 49 zeigt die Bestimmtheit des vollständigen Modells (linke Spalte) und des linearen Modells (rechte Spalte). Die Bestimmtheit der Zweig-, Schleifen- und Termüberdeckung durch die normierte Testfallzahl ist in beiden Fällen gleich hoch. Die Anweisungsüberdeckung wird in beiden Modellen gleich berechnet und hat damit die gleiche Bestimmtheit.

R^2 der Analyse mit	vollständigem Modell	linearem Modell
Anweisungen	0,74	-
Zweige	0,83	0,83
Schleifen	0,68	0,68
Terme	0,85	0,85

Tabelle 49: Vergleich des vollständigen und des linearen Modells mit Daten aus dem Praktikum (Analyse, Regressionsmodell)

Aussagen über die Fähigkeit der Modelle, Werte zu prognostizieren, können nur getroffen werden, wenn Quantifizierung und Prognose mit unterschiedlichen Daten durchgeführt werden. Für die Prognose werden darum Daten aus der Datengruppe 2 verwendet.

Dazu werden die Zusammenhänge zuerst mit Parametern quantifiziert, die aus der Datengruppe 1 stammen; es werden die Parameter verwendet, die durch Regression in den obigen Abschnitten berechnet wurden. Beispielsweise wird der Faktor 0,69 zwischen Anweisungs- und Zweigüberdeckung aus Tabelle 48 verwendet. Dann werden die normierten Testfallzahlen aus der Datengruppe 2 verwendet, um die zugehörigen Überdeckungsgrade zu berechnen. Diese berechneten Resultate (prognostizierte Werte) werden mit den Istwerten aus der Datengruppe 2 verglichen.

Die Prognose der Daten der Gruppe 2 zeigt eine etwas höhere Bestimmtheit des linearen Modells (Tabelle 50). Die Tabelle zeigt zusätzlich die Abweichung zwischen den Istwerten und den prognostizierten Werten in dB. Sie ist mit dem linearen Modell etwas geringer und unter der 2-dB-Grenze.

Alle Zusammenhänge sind statistisch signifikant; die Nullhypothese kann abgewiesen werden. Abbildung 64 veranschaulicht die Resultate der Prognose. Die Diagramme überlagern die Werte mehrerer Teams, um die Streuung zwischen den Teams zu zeigen. Das Diagramm zeigt auf der x-Achse die normierte Testfallzahl und auf der y-Achse den Überdeckungsgrad. Jeder dunkle Punkt im Diagram stellt die Werte eines Testfalls der Datengruppe 2 dar. Mit dem Testfall wurden vom Team x % der normierten Testfallzahl durchgeführt. Dabei wurden y % Überdeckung erreicht. Die hellen Punkte stellen die Ergebnisse des Prognosemodells mit linearem Zusammenhang dar.

Überdeckung	R^2 des Prognosemodells		*MLE* der Prognose (dB)[a]	
	Vollständiges Modell	Lineares Modell	Vollständiges Modell	Lineares Modell
Anweisungen	0,65	-	0,70	-
Zweige	0,75	0,77	0,79	0,76
Schleifen	0,63	0,67	1,23	1,19
Terme	0,82	0,83	1,21	1,11

Tabelle 50: Vergleich des vollständigen und des linearen Modells mit Daten aus dem Praktikum (Prognosemodell)

a. Median des $LE = 10 \cdot |\log(Modellresultat/Istwert)|$

Bewertung. Die Hypothese H 7 von CoBe wird bestätigt. Die Modellierung durch einen linearen Zusammenhang ergibt eine hohe Übereinstimmung zwischen den Werten, die mit dem Modell berechnet werden, und den Istwerten.

Folgerungen. Der in CoBe verwendete Zusammenhang zur Berechnung der Überdeckung ist stark genug, dass er zur Prognose verwendet werden kann. CoBe wird für die Fälle, in denen Zweige, Terme oder Schleifen nicht gezielt überdeckt werden, mit den Faktoren aus Tabelle 48 quantifiziert; andere Daten sind nicht verfügbar. Den

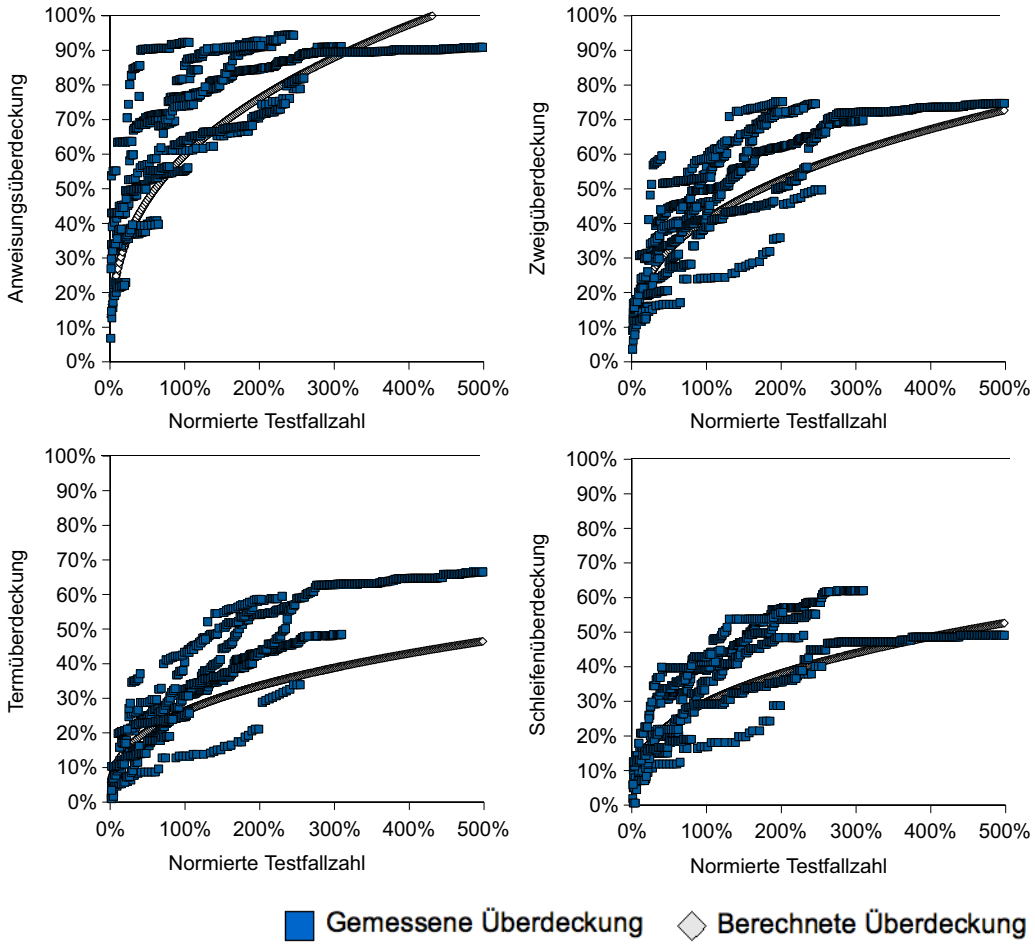

Abb. 64: Prognose der Überdeckung im Praktikum mit linearem Modell

Zusammenhang zwischen der normierten Testfallzahl und der Anweisungsüberdeckung verändere ich nicht, weil im Praktikum eine hohe Überdeckung bereits mit wenigen Testfällen erreicht wird. Dies widerspricht Industriedaten (Abschnitt 5.5); die Produkte des Praktikums sind kleiner als Industrie-Produkte.

Prüfung H 8: Der Zusammenhang zwischen der normierten Anzahl der Testfälle und der Fehlerentdeckungsquote hat die Form $Q_{Test} = \max(0, 1 - r_{qt}(1 - q_t)^{t_{Test}})$.

Im Testmodell von CoBe wird angenommen, dass jeder Testfall einen bestimmten, kleinen Anteil q_t der enthaltenen Fehler entdeckt. Dieser Zusammenhang wird durch eine Funktion der Form $Q_{Test} = \max(0, 1 - r_{qt}(1 - q_t)^{t_{Test}})$ dargestellt (Abbildung 60, Seite 171 und Abschnitt 6.6.2). Q_{Test} ist die Fehlerentdeckungsquote, t_{Test} ist die normierte Testfallzahl, r_{qt} und q_t sind Parameter. Dieser Zusammenhang wird durch Regression mit logarithmierter Fehlerentdeckungsquote mit dem umgeformten Zusammenhang $1 - Q_{Test} = r_{qt}(1 - q_t)^{t_{Test}}$ geprüft:

- Die Fehlerentdeckungsquote wird für jeden Testfall für jedes Team berechnet. Dazu werden die Abweichungen, die bis zu diesem Testfall im Testprotokoll erfasst sind, aufsummiert. Diese Summe bildet den Zähler der Fehlerentdeckungsquote bis zu diesem Testfall. Die insgesamt im Systemtest und in der Abnahme entdeckte Zahl Fehler bildet den Nenner der Fehlerentdeckungsquote.
- Die normierte Testfallzahl ist die Zahl der Testfälle, die bis zum jeweiligen Testfall durchgeführt wurden, normiert mit der typischen Testfallzahl.
- Die Analyse wird wieder mit der Datengruppe 1 durchgeführt. Mit diesen Daten der Gruppe 1 werden r_{qt}' und q_t' durch Regression bestimmt. Dann wird die Fehlerentdeckungsquote Q_{Test} aus den Testfallzahlen der Datengruppe 1 berechnet und mit den Istwerten verglichen. Signifikanz, Bestimmtheit (R^2) und Genauigkeit (Median der logarithmischen Abweichung, *MLE*) werden geprüft.
- Die Parameter r_{qt}' und q_t' werden verwendet, um die Fehlerentdeckungsquote für die Testfallzahlen der Datengruppe 2 zu berechnen. Damit lassen sich Aussagen über die Prognose mit diesem Zusammenhang gewinnen.

Der Zusammenhang zwischen dieser normierten Testfallzahl und der Fehlerentdeckungsquote ist für die Daten aus dem Software-Praktikum statistisch signifikant (p-Wert < 0,001). Tabelle 51 zeigt Signifikanz, Bestimmtheit und Genauigkeit für die Datengruppe 1 (Analyse) in der ersten Zeile, für die Prognose in der zweiten Zeile. In beiden Fällen ist der Zusammenhang statistisch signifikant. Bei der Analyse ist die Bestimmtheit und die Genauigkeit hoch. Bei der Prognose nehmen Genauigkeit und Bestimmtheit ab. Der Median für die logarithmischen Abweichung *MLE* liegt innerhalb der 2-dB-Grenze.

Modell	Signifikanz p[a]	Bestimmtheit R^2	*MLE* (dB)[b]
Analyse (Datengruppe 1)	**< 0,001**	0,50	1,55
Prognose (Datengruppe 2)	**< 0,001**	0,22	1,73

Tabelle 51: Ergebnisse der Regression mit den Daten aus dem Praktikum für die Fehlerentdeckungsquote

a. Statistisch signifikante Ergebnisse (5 %-Niveau) sind **fett** gedruckt.
b. Median des $LE = 10 \cdot |\log(Modellresultat/Istwert)|$

In Abbildung 65 sind die Daten aller Teams der Datengruppe 1 (Analyse, linkes Diagramm) und der Datengruppe 2 (Prognose, rechtes Diagramm) dargestellt. Die Darstellung überlagert die Werte mehrerer Teams, um die Streuung zwischen den Teams darzustellen. An der x-Achse ist die normierte Testfallzahl dargestellt. Jeder Testfall wird anhand der durchgeführten Testfälle des Teams auf der x-Achse und anhand der mit der bis zum Testfall erreichten Fehlerentdeckungsquote auf der y-Achse aufgetragen. Die dunklen Punkte sind Istwerte, die hellen Punkte die Resultate der Berechnung.

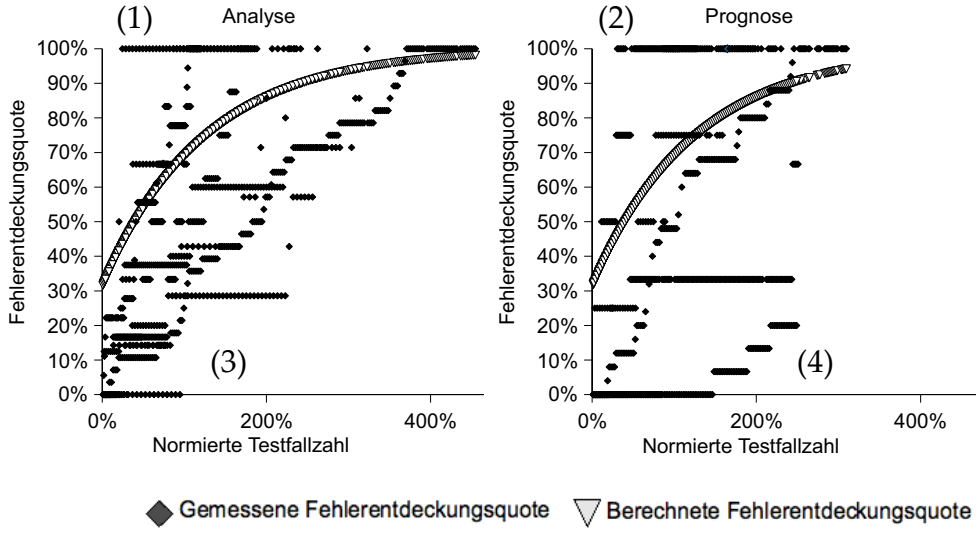

Abb. 65: Zusammenhang der Testfallzahl mit Fehlerentdeckung im Praktikum

Die Diagramme verdeutlichen die niedrige Bestimmtheit. Sie entsteht durch die starke Streuung der Fehlerentdeckungsquote, wenn kein Fehler oder wenige Fehler entdeckt wurden:

- Wird beispielsweise nur ein Fehler im gesamten Test mit einem frühen Testfall entdeckt, dann schnellt die Fehlerentdeckungsquote früh auf 100 % hoch. Dies zeigt sich in beiden Diagrammen. So hat mindestens ein Team in jeder Datengruppe mit einem Bruchteil der normierten Testfallzahl bereits 100 % derjenigen Fehler entdeckt, die im Systemtest und in der Abnahme entdeckt wurden (Punkte 1 und 2 in Abbildung 65).
- Es gibt aber auch Teams, die bis zu 100 % der normierten Testfallzahl durchführen und keine Fehler entdecken (Punkt 3); die Fehlerentdeckungsquote steigt erst mit vielen Testfällen an (Punkt 4).

Bewertung. Die Hypothese H 8 von CoBe wird bestätigt, da die Nullhypothese abgewiesen werden kann; der Zusammenhang ist signifikant. Die Streuung ist aber groß, bedingt durch unkontrollierte Variablen. Dazu gehören die unterschiedliche Intensität des Systemtests und die fehlenden Fehlerzahlen aus dem Einsatz des Produkts. Die starke Streuung wird aber auch in anderen Untersuchungen beobachtet (Abschnitt 5.5).

Folgerungen. Der im Testmodell von CoBe verwendete Zusammenhang wird bestätigt. Weil im Praktikum die Streuung groß und das Produkt klein ist, behalte ich die Quantifizierung von CoBe mit Werten aus der Industrie bei.

7.4.3 Fehlerfolgekosten

Das Modell zur Abschätzung der Fehlerfolgekosten (Abschnitt 6.3.10) wird überprüft, weil dazu keine Erfahrungswerte vorhanden sind. Gemessene Daten sind im Praktikum aber nicht verfügbar, da die Produkte nicht bei einem Kunden durch Benutzer unter realen Bedingungen eingesetzt werden. Darum prüfe ich, wie sich unterschiedlich detaillierte Fehlerklassifikationen auf die Fehlerfolgekosten auswirken. Dazu werden Resultate, die mit einzeln klassifizierten Fehlern berechnet werden (Fall 1), mit Resultaten verglichen, die etwa aus Archivdaten für die Modelleingaben berechnet werden. Insgesamt unterscheide ich vier Fälle:

1. Als Vergleichswerte verwende ich Daten aus dem Software-Praktikum und klassifiziere jeden Fehler, der im Praktikum entdeckt wurde. Daraus lassen sich die Fehleranteile für die Schadensklassen, für die Klassen der Auftretenswahrscheinlichkeit und für die Klasse der Verwendungshäufigkeit berechnen, die in CoBe eingegeben werden. Das Resultat ist ein Vergleichswert für die Fehlerfolgekosten; die Verteilungen können direkt verglichen werden.

Während der Planung von Projekten sind diese Daten nicht verfügbar. Darum muss auf andere Daten zurückgegriffen werden:

2. Es wird angenommen, dass sich die Fehleranteile gleichmäßig auf die Klassen für die Fehlerfolgekosten verteilt. Beispielsweise wird eine maximale Schadensklasse bestimmt (1000 Euro). Die Fehler verteilen sich dann gleichmäßig auf die Schadensklassen 0 Euro, 10 Euro, 100 Euro, 1000 Euro.
3. Es wird angenommen, dass die Fehlerschwere vor allem durch den möglichen Schaden, den der Fehler verursacht, definiert ist. Beispielsweise verursachen kritische Fehler 1000 Euro Schaden, wenn sie auftreten, Hauptfehler 100 Euro Schaden, Nebenfehler 10 Euro Schaden. Die Verteilung auf die Fehlerschwere wird aus Abschnitt 6.8.1 verwendet, mit rund 10 % kritischen Fehlern, 78 % Hauptfehlern und 12 % Nebenfehlern.
4. Wie in Fall 3 wird die Fehlerschwere durch den Schaden definiert ist. Die Verteilung auf die Fehlerschwere aus dem Praktikum wird verwendet (51 % Nebenfehler, 31 % Hauptfehler, 18 % kritische Fehler).

Damit für den Fall 1 Fehler einzeln klassifiziert werden können, betrachte ich die Fehlerkommentare. Die Teilnehmer im Praktikum haben jeden Fehler kommentiert. Anhand dieser Fehlerkommentare wurde jeder Fehler einer Schadensklasse und einer Klasse für die Auftretenshäufigkeit zugeordnet:

- Schadensklasse: Ich gehe von einem maximalen Schaden von 1000 Euro aus und stütze mich dabei auf den gedachten Einsatz der Software. Da mit der Software Testfälle verwaltet werden, ist der größte Schaden, wenn diese Testfälle verloren gehen. Der schlimmste Fall ist, wenn Daten nicht gespeichert oder geladen werden können; dann ist das Werkzeug nutzlos. Dies merkt der Tester spätestens, wenn er nach einem Tag Arbeit die Arbeit fortsetzen will und dazu die gespeicherten Daten

lädt. Somit ist also im schlimmsten Fall ungefähr ein Tag Arbeit des Testers verloren.

- Auftretenswahrscheinlichkeit: Die Auftretenswahrscheinlichkeit wird abhängig davon klassifiziert, ob ein Fehler nur unter ganz bestimmten Bedingungen auftritt, dann wird er als "selten auftretend" klassifiziert. Ein Fehler, der immer bei einer Hauptfunktion auftritt, wird als "sicher auftretend" klassifiziert, beispielsweise, wenn das Sollresultat für einen Testfall nicht eingegeben werden kann.
- Verwendungshäufigkeit: Ich nehme für die Verwendungshäufigkeit bis zur Korrektur einen Mittelwert an, dazu setze ich als Annahme, dass die Software im Mittel 10 mal verwendet wird, bis ein Fehler korrigiert wird. Vermutlich werden schwere Fehler rascher behoben als weniger schwere Fehler. Da es dazu aber keine weiteren Informationen gibt, verwende ich einen Mittelwert.

Für 612 Fehler, die im Praktikum durch alle Teams insgesamt entdeckt wurden, waren Kommentare verfügbar. Diese Fehler wurden klassifiziert.

Die Tabellen 54 und 55 zeigen die Resultate dieser Klassifikation, den Fall 1. Beispielsweise würden immerhin 83 Fehler jedesmal auftreten, wenn die Software verwendet wird[1]; 129 Fehler treten nie beim Einsatz auf, beispielsweise weil es sich um Kommentarfehler handelt. 149 Fehler würden einen Arbeitstag Verlust bedeuten, 177 Fehler den Verlust von etwa einer Arbeitsstunde. Aus diesen absoluten Fehlerzahlen wird direkt die Verteilung in Prozent auf die Fehlerklassen berechnet, d.h. welcher Anteil aller Fehler einer Klasse des Schadens und welcher Anteil aller Fehler einer Klasse der Auftretenswahrscheinlichkeit zugeordnet ist.

Auftretens-wahrscheinlichkeit	Beschreibung der Klasse[a]	Zahl der zugeordneten Fehler
0	Fehler tritt nie auf	129
0,125	Fehler tritt in Ausnahmefällen auf	52
0,25	Fehler tritt selten bei Verwendung auf	172
0,5	Fehler tritt bei typischer Verwendung auf	176
1	Fehler tritt sicher bei Verwendung auf	83
	Insgesamt	612

Tabelle 52: Fehlerklassifikation der Auftretenswahrscheinlichkeit im Praktikum

a. Die Beschreibung ist gekürzt.

1. Klassifiziert wurden alle Fehler, also beispielsweise auch diejenigen, die im Spezifikationsreview entdeckt wurden.

Schaden (Euro)	Beschreibung der Klasse	Zahl der zugeordneten Fehler
0	kein Schaden[a]	129
10	Komfortprobleme	157
100	Geringer, leicht auszugleichender Schaden	177
1000	Mittlerer, auszugleichender Schaden	149
	Insgesamt	612

Tabelle 53: Fehlerklassifikation im Praktikum für den Schaden beim Auftreten

a. Fehler, die nicht auftreten, sind als 0 Euro Schaden klassifiziert.

Für den Vergleich zeigen die Tabellen 54 und 55 die Verteilung der Fehler für die Klassifikation einzelner Fehler (Fall 1) und andere Verteilungen (Fälle 2 bis 4). Im Falle des Praktikums stimmt eine gleichmäßige Verteilung auf die Schadensklasse gut mit der Klassifikation einzelner Fehler überein.

Verteilung (Anteil der Fehler) auf Klassen	Fall 1	Fall 2	Fall 3	Fall 4
Fehler tritt nie auf	21 %	20 %	20 %	20 %
Fehler tritt in Ausnahmefällen auf	8 %	20 %	20 %	20 %
Fehler tritt selten bei Verwendung auf	28 %	20 %	20 %	20 %
Fehler tritt bei typischer Verwendung auf	29 %	20 %	20 %	20 %
Fehler tritt sicher bei Verwendung auf	14 %	20 %	20 %	20 %
Insgesamt	100 %	100 %	100 %	100 %

Tabelle 54: Verteilungen der Fehler für die Auftretenswahrscheinlichkeit

Verteilung (Anteil der Fehler) auf Klassen	Fall 1	Fall 2	Fall 3	Fall 4
kein Schaden	21 %	25 %	0 %	0 %
Komfortprobleme	26 %	25 %	12 %	51 %
Geringer, leicht auszugleichender Schaden	29 %	25 %	78 %	31 %
Mittlerer, auszugleichender Schaden	24 %	25 %	10 %	18 %
Insgesamt	100 %	100 %	100 %	100 %

Tabelle 55: Verteilungen der Fehler für den Schaden beim Auftreten

Die statistischen Fehlerfolgekosten pro Fehler für die vier Fälle zeigt Tabelle 56. Die Fehlerfolgekosten, die sich aus einer gleichmäßigen Verteilung auf die Fehlerklassen ergeben (Fall 2), stimmen gut mit den Fehlerfolgekosten überein, die sich aus der Klassifikation einzelner Fehler berechnen (Fall 1). Die Abweichung liegt unter 2 dB (0,7 dB). Im Fall 3 und im Fall 4 stimmen die Kosten weniger gut überein. Die Klassifikation mit der Fehlerschwere ist nur im Fall 4 unter der 2-dB-Grenze (1,8 dB). Diese Abweichung führe ich auf die Definition der Fehlerschwere im Praktikum zurück. Die Fehlerschwere berücksichtigt nicht nur den möglichen Schaden, sondern auch die möglichen Folgen des Fehlers für das Projekt.

Fehlerfolgekosten pro Fehler (Euro)	Fall 1	Fall 2	Fall 3	Fall 4
	1230	1040	700	810

Tabelle 56: Statistische Fehlerfolgekosten pro Fehler

Bewertung. Ich bewerte die Ergebnisse als plausibel, soweit dies mit den wenigen Erfahrungen möglich ist. Schlussfolgern lässt sich, dass eine Fehlerschwere-Definition, die Auswirkungen auf das Projekt und den Einsatz berücksichtigt, unabhängig vom Schaden ist, den ein Fehler beim Auftreten verursachen kann. Diese Unterscheidung muss bei der Anwendung des Modells berücksichtigt werden.

7.4.4 Folgerungen

Auch wenn die interne und externe Validität der Untersuchung durch die spezielle Situation im Praktikum bedroht wird, werden die Hypothesen bestätigt.

Die Prüfung der Hypothesen zur Fehlerentdeckung, zur Fehlerentstehung und zu den Korrekturaufwänden bestätigt bereits Bekanntes, aber stärker auf das Modell zugeschnitten. Insbesondere werden die Annahmen auch für Java-Programme bestätigt.

Die Annahmen, die dem Testmodell zu Grunde liegen, werden bestätigt. Fraglich ist aber, ob diese Annahmen verallgemeinert werden können. Da wenig empirische Untersuchungen existieren, ist ein Vergleich schwierig. Für eine besser verallgemeinerbare Validierung des Modells müssen Daten auf einer breiteren Basis, vor allem aus Industrieprojekten mit großer und komplexer Software, gesammelt und analysiert werden.

Für die Bewertung der Fehlerfolgekosten fehlen Vergleichswerte, die Ergebnisse können aber als plausibel beurteilt werden. Insbesondere zeigt sich, dass eine detaillierte Klassifizierung einzelner Fehler nicht notwendig ist. Es zeigt sich, dass die Definition der Fehlerschwere berücksichtigt werden muss: In Projekten, bei denen die Fehlerschwere über den möglichen Schaden definiert ist, können Daten über die Fehlerschwere verwendet werden. In anderen Fällen kann eine gleichmäßige Verteilung der Fehler bis zum maximalen Schaden verwendet werden.

7.5 Erprobung im Software-Praktikum

Nachdem einzelne Zusammenhänge, aus denen CoBe besteht, geprüft wurden, werden im nächsten Schritt die Resultate von CoBe mit Istwerten verglichen. Als Eingaben für CoBe werden zuerst Mittelwerte und Mediane aus dem Praktikum verwendet. Tabelle 57 zeigt diese Eingaben. Die Modellresultate, die mit diesen Eingaben berechnet werden, werden mit Mittelwerten und Medianen der Istwerte aus dem Praktikum verglichen.

Eingabeparameter	Wert und Beschreibung
Umfang	7104 Anweisungen[a]
Umfangsfaktor Code	53 Statements pro Function Point für Java
COCOMO-II-Faktoren	Die Parameter sind mit den Vorgaben aus Boehm (2000) belegt. Der Exponent ist 1,05; der Gesamteinfluss 0,63.
Reviews	4 Gutachter mit Nominalvorbereitung und -eignung
Modultest, Integrationstest	Beide Tests werden durchgeführt. Da im Praktikum kontinuierlich integriert wird, beginnend mit der Implementierung bis zu Korrekturen nach dem Systemtest, wird dies in CoBe als Integrationstest dargestellt.
Systemtest	Black-Box-Test der Funktionen und Äquivalenzklassen
	Glass-Box-Test mit 86 % Anweisungsüberdeckung im Mittel
	Keine Testwiederholung, keine getrennte Testvorbereitung, nominale Eignung der Tester
Feldtest	Für die Abbildung des Praktikums in CoBe nehme ich an, dass der Feldtest in CoBe in etwa der Abnahme im Praktikum entspricht.

Tabelle 57: Eingabeparameter für das Software-Praktikum

a. Der Median und nicht der Mittelwert wird verwendet, weil der Median robuster gegen Ausreißer ist, die beispielsweise entstehen, weil bei der Umfangsmessung fremde Bibliotheken oder Testcode nicht erkannt wurde.

7.5.1 Vergleich mit Mittelwerten und Kalibrierung

Zuerst wird eine unkalibrierte Modellversion erprobt. Diese Version enthält einen negativen, aber keinen positiven Einfluss der Gutachterkompetenz. Sie enthält keinen Umfangseinfluss auf den Anstieg der Korrekturkosten mit der Latenzzeit: In dieser Modellversion kostet ein Spezifikationsfehler immer das zehnfache, wenn er im Systemtest anstatt durch ein Spezifikationsreview entdeckt wird, unabhängig davon, ob das Produkt einen geringen oder einen hohen Umfang hat.

Die Istwerte, die für die Eingaben und zum Vergleich der Modellresultate verwendet werden, sind nicht vollständig, weil im Praktikum nicht alle Daten verfügbar waren.

Insbesondere für den Vergleich des Gesamtaufwands muss auf die Vorgabe der Prüfungsordnung zurückgegriffen werden. Diese Vorgabe wird aber von den Betreuern des Praktikums in engen Bandbreiten gehalten, weil der benötigte Aufwand von den Betreuern auf Basis einer intensiven Analyse der Aufgabe geschätzt wird. Da die Termine der Meilensteine in geringem Abstand folgen und die Abgaben kontrolliert werden, sind die Teilnehmer deutlich eingeschränkt. Sie können den Aufwand nicht zu stark minimieren, weil sonst ihre Abgabe nicht abgenommen wird. Sie können nicht zu viel Aufwand investieren, weil sie sonst den nächsten Meilenstein nicht erreichen.

Tabelle 58 zeigt die ersten Resultate. Ohne Kalibrierung weichen die Modellresultate stark von den Istwerten ab, mit Ausnahme der Testfallzahl.

Parameter	Modellresultate[a]	Istwert
Gesamtaufwand in Entwicklerstunden (Eh)	2639	720[b]
Gesamtdauer in Arbeitstagen	227	105[c]
Zahl der Projektmitarbeiter	1,9	3,0[d] Teilzeitmitarbeiter, 1,0 Vollzeitmitarbeiter
Zahl der Fehler im Projekt	324	Mittelwert 74 Median 51
Verteilung auf Fehlerart (Spez. / Entwurf / Code)	22 % / 28 % / 39 %	51 % / 19 % / 30 %
Zahl der Testfälle	80	Mittelwert 91 Median 68

Tabelle 58: Parameter im Überblick

a. für ein Java-Projekt mit 7104 Anweisungen
b. Richtwert der Prüfungsordnung
c. Vorgabe der Betreuer
d. Regelfall, nur in Ausnahmefällen sind Zweier-Teams möglich; 720 Entwicklerstunden entsprechen bei 21 Wochen Dauer insgesamt rund 34 Stunden pro Woche.

Dies zeigt, dass eine Kalibrierung notwendig ist. Die Kalibrierung erfolgte mit dem Aufwandsfaktor, dem Dauerfaktor, dem Fehlerfaktor, der Verteilung auf die Fehlerarten und den Umfangsfaktoren für Dokumente (Tabelle 59).

7.5.2 Modellverbesserungen

Zwei Modellverbesserungen wurden durchgeführt:

- In der erprobten Modellversion war kein positiver Einfluss der Gutachterkompetenz quantifiziert. Die Teilnehmer begutachteten die Spezifikation in der Reviewvorbereitung mit rund 18 Seiten pro Stunde (Median 16 Seiten pro Stunde) anstatt mit 10 Seiten pro Stunde. Die Fehlerentdeckungsquote im Software-Praktikum ist

Modellparameter	Ursprünglicher Wert	Änderung für Praktikum
Aufwandsfaktor	1,00	0,28
Dauerfaktor	1,00	0,71
Fehlerfaktor	1,00	0,20
Umfangsfaktor Spezifikation	0,44 Seiten / FP	0,32 Seiten / FP
Umfangsfaktor Entwurf	0,44 Seiten / FP	0,18 Seiten / FP
Verteilung auf Fehlerart (Spez./Entwurf/Code)	22 % / 28 % / 39 %	51 % / 19 % / 30 %

Tabelle 59: Kalibrierung von CoBe für das Praktikum

mit 72 % hoch, daraus folgt, dass die Begutachtung nicht oberflächlich war, sondern dass die hohe Vorbereitungsrate von 18 Seiten pro Stunde andere Gründe hat. Die Gründe dafür vermute ich in der Gutachterauswahl, da die Gutachter aus anderen Teams des Praktikums stammen. Sie kennen somit die Anforderungen sehr genau, weil sie Analyse und Spezifikation für die gleiche Aufgabe selbst durchgeführt haben. Darum wurde die Quantifizierung des Kompetenzeinflusses in CoBe um einen positiven Effekt hoher Kompetenz auf den notwendigen Aufwand und die Fehlerentdeckung ergänzt.

- Die Korrekturkosten steigen im Verlauf des Projekts weniger stark an als in der erprobten Modellversion quantifiziert (Tabelle 44). Dass die Korrekturkosten in kleinen Projekten weniger stark mit der Latenzzeit ansteigen, zeigt Boehm (1981 und 1976); der Zusammenhang wurde in CoBe ergänzt.

Die beiden Abbildungen 66 und 67 zeigen, welche Teile des Modells von der Kalibrierung und welche Teile des Modells von der Verbesserung betroffen sind. Die Änderungen sind lokal begrenzt.

7.5.3 Modellresultate und Mittelwerte des Praktikums

Der folgende Vergleich zeigt Modellresultate und Mittelwerte des Praktikums. Dazu werden die gleichen Istwerte verwendet, gegen die bereits vor der Verbesserung verglichen wurde (Abschnitt 7.5.1).

Fehlerentdeckung und Fehlerkorrektur.

Die absoluten Fehlerzahlen zeigen eine gute Übereinstimmung (Tabelle 60) zwischen den Modellresultaten und den Istwerten aus dem Software-Praktikum. Auch die Modellresultate für die Fehleranteile stimmen gut mit den Istwerten überein: Die Anteile der Spezifikationsfehler entsprechen den Mittelwerten des Software-Praktikums (Tabelle 61). Es gibt Abweichungen bei den Zahlen der frühen Fehler in den Tests und beim Abnahmetest, die sich aber durch die unklare Situation der Entwurfsreviews, die kontinuierliche Integration und den oberflächlichen Abnahmetest erklä-

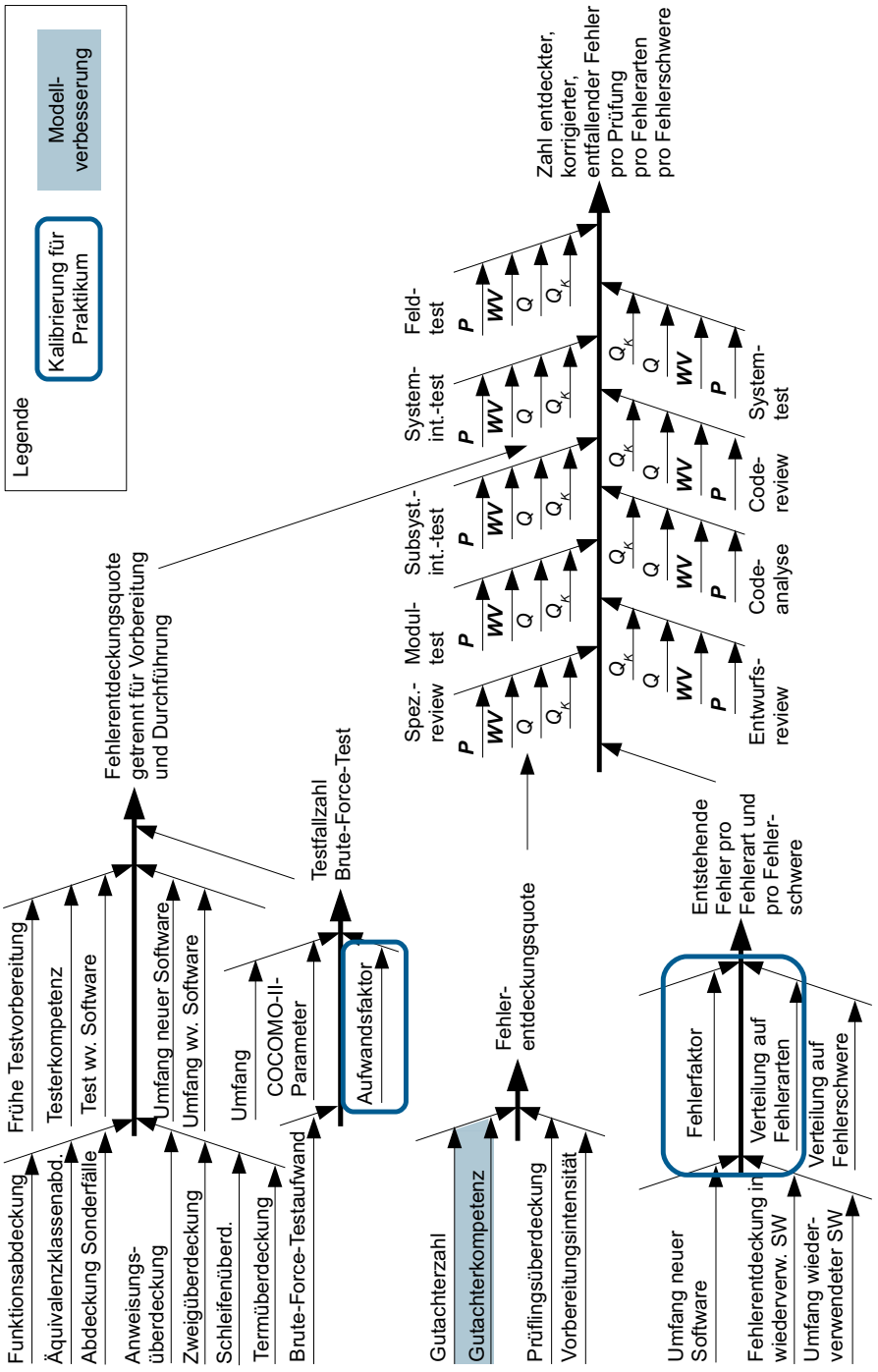

Abb. 66: Modellanpassung der Fehlerentstehung und -entdeckung in CoBe

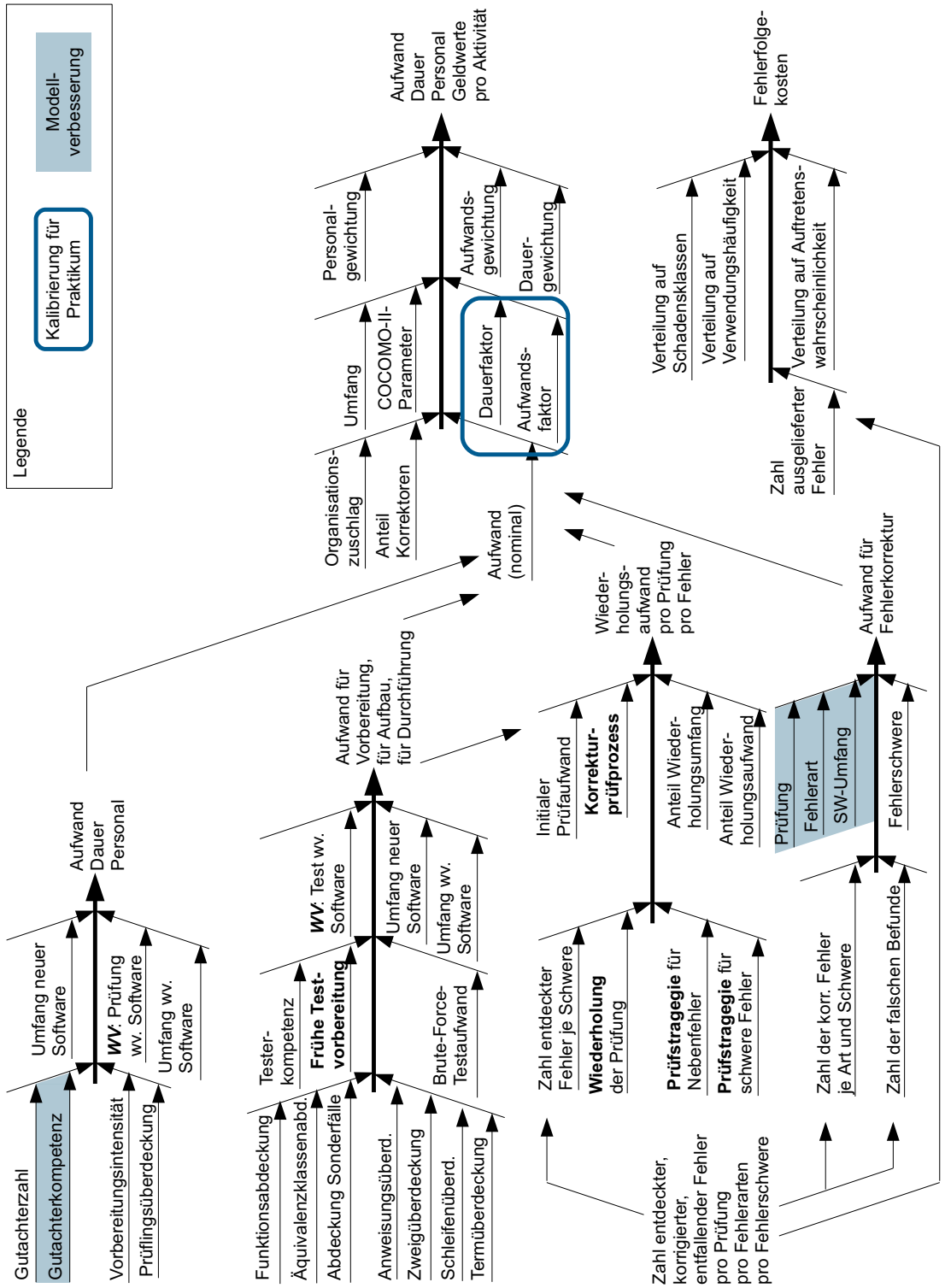

Abb. 67: Modellanpassung der Kostenberechnung in CoBe

ren lassen. Der Korrekturaufwand wird mit hoher Genauigkeit berechnet (Tabelle 62). Die Modellresultate liegen meist zwischen Mittelwert und Median der Istwerte.

	Modellresultate			Software-Praktikum		
Prüfung	Spez.-fehler	Entwurfs-fehler	Code-fehler	Spez.-fehler	Entwurfs-fehler	Code-fehler
Spezifikationsreview	34,6			34,8		
Entwurfsreview	1,8	9,5	0	0,1	11,6	
Unittest	0	0,4	6,0	0	0,3	7,6
Integrationstest	1,1	1,2	7,2			
Systemtest	1,2	1,3	7,2	0,7	0,2	9,2
Abnahme	1,6	1,0	2,2			0,4
Andere				1,9	1,9	4,8

Tabelle 60: Zahl entdeckter Fehler

	Modellresultat			Software-Praktikum		
Prüfung	Spez.-fehler	Entwurfs-fehler	Code-fehler	Spez.-fehler	Entwurfs-fehler	Code-fehler
Spezifikationsreview	86 %			93 %		
Entwurfsreview	4 %	71 %		0 %	83 %	
Modultest	0 %	3 %	27 %	0 %	2 %	35 %
Integrationstest	3 %	9 %	32 %			
Systemtest	3 %	9 %	32 %	2 %	4 %	42 %
Abnahme	4 %	8 %	10 %	0 %	0 %	2 %
Andere				5 %	14 %	22 %

Tabelle 61: Anteile entdeckter Fehler

Prüfaufwand und -dauer. Die vorgegebenen Termine wurden von den meisten Teams leicht erreicht. Sie sind großzügig bemessen. Die Modellresultate passen gut zu den Vorgaben und sind darum plausibel (Tabelle 63). Die Vorgabe für den Systemtest ist, verglichen mit den Modellresultaten, knapp. Stimmt der Mittelwert, den das Modell berechnet, dann könnten etwa die Hälfte der Gruppen den Abgabetermin nur mit Mühe oder nicht einhalten. Dafür gibt es zwei Gründe. Der Durchführungsaufwand für jeden Testfall ist in CoBe etwas höher als im Praktikum, aber auch der Zeit-

Korrekturaufwand (Eh) nach Prüfung	Modell-resultat	Median des Praktikums	Mittelwert des Praktikums
Spezifikationsreview	7,5	6,9	9,6
Entwurfsreview	3,4	2,5	4,5
Modultest	2,0	1,6	4,9
Systemtest	11,9	9,9	9,2

Tabelle 62: Resultate für den Korrekturaufwand

Prüfung und Korrektur	Modellresultate		Vorgabe
	Aufwand (Eh)	Dauer (Tage)	Dauer (Tage)
Spezifikationsreview	32	8	14
Entwurfsreview	19	3	-
Modultest	10	9	14
Systemtest	32	14	14

Tabelle 63: Vorgaben und Modellresultate

rahmen für das Praktikum war tatsächlich etwas eng, weil im Gegensatz zu früheren Praktika die Messung der Anweisungsüberdeckung verlangt wurde, ohne den Zeitplan anzupassen.

7.6 Vergleich mit einzelnen Projekten des Software-Praktikums

Nach dem Vergleich mit den Mittelwerten stellt sich die Frage nach der Bandbreite, d.h. nach den Unterschieden zwischen den einzelnen Projekten. Wie stark streuen die Resultate? Gibt es Ausreißer einzelner Projekte? Wie groß sind die Unterschiede im Prozess? Dazu werden die Projekte individuell analysiert. Jedes Projekt wird durch eine Modellinstanz dargestellt. Damit können folgende Merkmale untersucht werden:

- Die Streuung der unterschiedlichen Projekte mit gleichen Rahmenbedingungen soll für die Projekte und für das Modell untersucht werden.
- Damit kann auch der Einfluss unkontrollierter Variablen bewertet werden. Da mit gleicher Aufgabe und gleichem Prozess viele Projekte durchgeführt wurden, sind die Projekte ähnlich. Trotzdem streuen die Istwerte deutlich. Darum soll der Vergleich mit den Modellresultaten zeigen, welcher Teil dieser Streuung durch das Modell erklärt wird und welcher Teil nicht. Damit kann auf die zu erwartende Ungenauigkeit beim Vergleich mit einzelnen Projekten geschlossen werden.

- Die Genauigkeit des Modells soll anhand von Situationen, in denen viele Informationen zur Verfügung stehen, bewertet werden. Dazu gehört der diagnostische Einsatz von Cobe, mit dem bestehende Projekte nachträglich beschrieben werden, beispielsweise um den Nutzen von Prozessverbesserungen zu zeigen.
- Mit einer Kreuzvalidierung wird die Prognosesituation nachgebildet, weil bei der Prognose weniger Modellparameter bekannt sind als bei einer diagnostischen, nachträglichen Betrachtung. Darum werden die unbekannten Parameter nicht mit den Istwerten des Projekts belegt, sondern mit Durchschnittswerten aus anderen Projekten; dieses Vorgehen entspricht einer Kalibrierung mit historischen Daten.
- Obwohl die Unterschiede der Prüfintensität gering waren, soll untersucht werden, ob der Nutzen von mehr oder weniger intensiven Prüfungen sowohl im Modell als auch in der Realität gezeigt werden kann.

7.6.1 Diagnose einzelner Projekte

Die Eingaben von CoBe werden für die Diagnose auf die verfügbaren Werte der einzelnen Projekte (Tabelle 42, Seite 161) gesetzt. Tabelle 64 zeigt die individuellen Eingaben.

Prozess- und Produktmerkmale	Prüfprozess
• Umfang des Codes • Umfangsfaktoren Spezifikation, Entwurf • Aufwandsfaktor • Dauerfaktor • Verteilung auf Fehlerart	• Gutachterzahl und Vorbereitungsintensität im Spezifikationsreview • Intensität des Black-Box-Tests und Anweisungsüberdeckung des Glass-Box-Tests

Tabelle 64: Individuelle Eingaben für das Modell

Die Genauigkeit von CoBe wird durch den Vergleich zwischen Istwerten und Modellresultaten untersucht. Dafür sind nicht alle Daten verfügbar, die im Idealfall für den Vergleich vorhanden wären:

- Als Gesamtaufwand muss der Richtwert der Prüfungsordnung (720 Entwicklerstunden) verwendet werden, um den Aufwandsfaktor zu berechnen.
- Die Gesamtdauer wurde für alle Projekte von den Betreuern vorgegeben.
- Die Gesamtfehlerzahl wurde nicht individuell kalibriert, weil keine Fehlerzahlen aus dem Einsatz des Produkts verfügbar waren. Der Fehlerfaktor ist somit für alle Projekte gleich.
- Über das Entwurfsreview und über den Modultest gibt es keine individuellen Informationen.

Weil diese Informationen fehlen, werden die Modellresultate ungenauer. Die Aussagen über die Genauigkeit von CoBe, die mit diesen Daten getroffen werden, bewerten CoBe tendenziell als zu ungenau. Diese Situation spiegelt aber die Situation in der Praxis wider, wenn Daten unvollständig sind.

Vergleich mit Istwerten. Tabelle 65 zeigt die Abweichung der Modellresultate von den Istwerten und die Korrelation zwischen den Werten. Die Abbildung 68 veranschaulicht die Resultate für den Korrekturaufwand nach dem Spezifikationsreview und nach dem Systemtest.

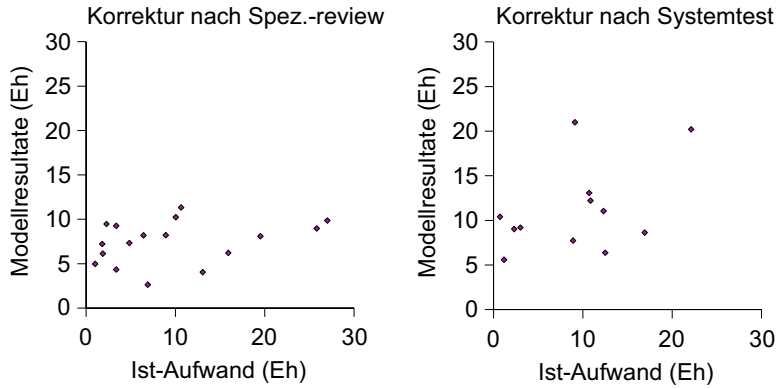

Abb. 68: Korrekturaufwände

Die Modellresultate für die Fehlerzahlen liegen für rund die Hälfte der Projekte innerhalb der 2-dB-Grenze. Im Median übersteigt die Abweichung diese Grenze, bleibt aber unterhalb von 3 dB. Der Korrekturaufwand weicht teilweise stärker ab, die Modellresultate liegen für etwa ein Drittel der Projekte innerhalb der Grenze von 2 dB. Der Median übersteigt zum Teil 3 dB. Die Testfallzahl wird wieder genauer berechnet mit einem Median der Abweichung von 1,6 dB und zwei Drittel der Projekte innerhalb der 2-dB-Grenze.

Die Resultate sind somit nicht mehr plausibel. Dafür gibt es aber Ursachen, die so in Industrieprojekten nicht gegeben sind, sondern speziell für das Praktikum gelten. Die Ursachen diskutiere ich mit den Ergebnissen der Kreuzvalidierung in Abschnitt 7.6.3.

Vergleich des Nutzens. Der Nutzen einer Prüfung ist in den Projekten nicht direkt sichtbar. Er kann nur durch den Vergleich zwischen Projekten mit mehr oder weniger intensiver Prüfung sichtbar werden. Um die Modellresultate für den Nutzen zu prüfen, werden im Folgenden die Auswirkungen mehr oder weniger intensiver Spezifikationsreviews betrachtet. Dazu werden die Teams anhand des Aufwands zur Begutachtung pro Seite geordnet; dieser Aufwand pro Seite berechnet sich aus der Gutachterzahl, die zwischen 3 und 4 liegt, und der Vorbereitungsintensität jeden Gutachters, die sich aus der Anzahl Seiten und dem Vorbereitungsaufwand berechnet. Eine Datengruppe wird aus den 7 Teams gebildet, deren Spezifikation am intensivsten geprüft wurde (rund 3,7 Seiten pro Entwicklerstunde im Median). Die andere

		Spez.-fehler	Entwurfs-fehler	Code-fehler	Fehler gesamt	Korrektur-aufwand
Spez.-review	*MLE* (dB)[a]	2,3				4,2
	pred(2 dB)	47 %				35 %
Entwurfs-review	*MLE* (dB)		2,1		2,0	3,1
	pred(2 dB)		50 %		50 %	44 %
Modultest	*MLE* (dB)			1,6	1,8	4,2
	pred(2 dB)			71 %	50 %	36 %
Systemtest	*MLE* (dB)			1,7	2,7	2,9
	pred(2 dB)			56 %	39 %	42 %

Tabelle 65: Genauigkeit des Modells

a. Median des $LE = 10 \cdot |\log(Modellresultat/Istwert)|$

Datengruppe wird aus den 7 Teams gebildet, deren Spezifikation am wenigsten intensiv begutachtet wurde (rund 7 Seiten pro Entwicklerstunde im Median). Im Folgenden wird die Anzahl der verfügbaren Werte angegeben, weil die Daten nicht immer für alle Teams zur Verfügung stehen.

Der Unterschied zwischen gründlichen und oberflächlichen Spezifikationsreviews zeigt sich konsistent in den Modellresultaten und in den Istwerten des Praktikums (Abbildung 69, Tabelle 66). Die intensiven, gründlichen Reviews finden im Mittel mehr Fehler, entsprechend ist der Korrekturaufwand nach intensiven Reviews höher als nach oberflächlichen Reviews. Abbildung 69 zeigt dies graphisch im linken Teil. Die dunklen Balken sind Werte für den Korrekturaufwand gründlicher Reviews, die hellen Balken für oberflächlichere Reviews. Das erste Balkenpaar links zeigt die Istwerte, das zweite Balkenpaar daneben die Modellresultate für den mittleren Korrekturaufwand nach dem Spezifikationsreview. In beiden Fällen ist die Korrektur teurer, wenn die Spezifikation intensiv geprüft wurde.

Tabelle 67 zeigt Istwerte und Modellresultate für den Systemtest. Im Systemtest wurden im Praktikum bei gründlichen Spezifikationsreviews weniger Spezifikationsfehler entdeckt (Tabelle 67). Dieser Unterschied wird auch durch CoBe berechnet. Der Unterschied ist aber gering. Da die Reviews gründlich waren, wurden im Systemtest wenig Spezifikationsfehler entdeckt. Somit kann sich der Unterschied nicht mehr im Median der Istwerte zeigen.

Nach dem Systemtest ist der Korrekturaufwand für alle Fehler, also auch für Codefehler und für Entwurfsfehler, mit gründlichen Spezifikationsreviews geringer (Tabelle 67). Dieser Unterschied zeigt sich konsistent in den Modellresultaten und in den Istwerten. Abbildung 69 zeigt dies graphisch im rechten Teil. Die dunklen Balken stehen für den Korrekturaufwand in Projekten mit gründlichen Reviews, die hellen

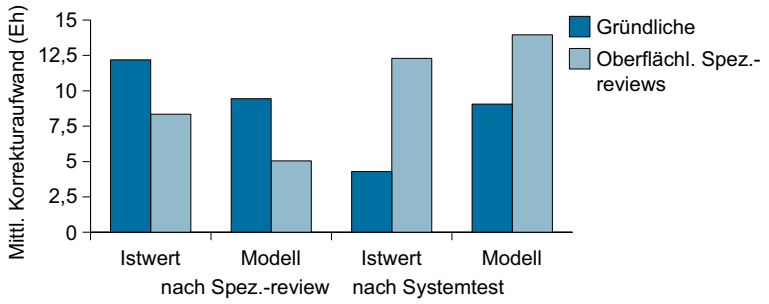

Abb. 69: Auswirkungen der Spezifikationsreview-Unterschiede

Entdeckte Fehler und Korrekturaufwand nach Spezifikationsreview	Spez.-fehler		Korrekturaufwand (Eh)	
	Ist	Modell	Ist	Modell
Mittelwert gründliche Reviews	39,2	39,0	12,2	9,5
Median gründliche Reviews	35,0	39,5	9,8	9,1
Anzahl Teams	6		4	
Mittelwert oberflächliche Reviews	26,4	25,6	8,3	5,0
Median oberflächliche Reviews	24,0	19,1	6,9	5,0
Anzahl Teams	5		5	

Tabelle 66: Fehler und Korrekturaufwand für Spezifikationsreviews

Entdeckte Spezifikationsfehler und Korrekturaufwand im Systemtest	Spez.-fehler		Korrekturaufwand (Eh)	
	Ist	Modell	Ist	Modell
Mittelwert gründliche Reviews	0,4	0,6	4,3	9,1
Median gründliche Reviews	0	0,4	2,1	9,8
Anzahl Teams	6		4	
Mittelwert oberflächliche Reviews	1	1,5	12,3	14,0
Median oberflächliche Reviews	0	1,6	10,9	13,6
Anzahl Teams	6		3	

Tabelle 67: Fehler und Korrekturaufwand im Systemtest

Nutzen durch Spezifikationsreview im Systemtest	Entfallender Korrekturaufwand (Eh)	
	Modell[a]	Modell[b]
Mittelwert gründliche Reviews	9,4	9,4
Median gründliche Reviews	7,2	9,7
Anzahl Teams	7	4
Mittelwert oberflächliche Reviews	4,3	4,3
Median oberflächliche Reviews	4,6	4,9
Anzahl Teams	7	3

Tabelle 68: Berechneter Nutzen durch das Modell

a. Dargestellt werden Mittelwerte und Vergleichswerte für alle Teams.
b. Dargestellt werden nur Werte der Teams, für die Istwerte verfügbar sind.

Balken für den Korrekturaufwand in Projekten mit oberflächlicheren Reviews. Das dritte Balkenpaar von links zeigt die Istwerte, das Balkenpaar rechts die Modellresultate für den Korrekturaufwand nach dem Systemtest. In beiden Fällen ist der Korrekturaufwand niedriger, wenn die Spezifikation intensiv geprüft wurde.

Die Istwerte zeigen für die wenigen verfügbaren Daten einen größeren Unterschied als die Modellresultate: Tabelle 67 zeigt in den Istwerten eine Differenz von etwa 8 Entwicklerstunden für die Korrektur nach dem Systemtest. Die Modellresultate für den Nutzen des Spezifikationsreviews im Systemtest (Tabelle 68) und für die Korrektur aller Fehler nach Systemtest sind dagegen mit etwa 4 Entwicklerstunden in allen Fällen niedriger. Mehrere Erklärungen sind möglich:

- Zufall: Die gründlich begutachteten Teams haben zufällig insgesamt weniger Fehler gemacht und darum auch weniger Fehler nach dem Systemtest zu korrigieren. Dagegen spricht, dass mit gründlicheren Reviews mehr Fehler entdeckt wurden und mehr Korrekturaufwand benötigt wurde (Tabelle 66).
- Höhere Systemtest-Intensität: Eine höhere Intensität des Systemtests der oberflächlich begutachteten Teams spielt keine Rolle, weil sowohl die Teams mit intensiven Reviews als auch die Teams mit oberflächlichen Reviews im Mittel jeweils 91 Testfälle durchgeführt haben.
- Gründliches Entwurfsreview: Eine andere mögliche Ursache liegt in den gründlichen Gutachtern, die nicht nur die Spezifikation, sondern auch den Entwurf begutachtet haben, so dass sich nicht nur der Effekt des gründlichen Spezifikationsreviews, sondern auch des gründlichen Entwurfsreviews zeigt.
- Zufall: Insgesamt sind nur wenige Datenpunkte verfügbar, so dass die Aussagekraft eingeschränkt ist und die Ergebnisse zufällig entstanden sein könnten.

Die Unterschiede der Istwerte sind statistisch nicht signifikant (5 %-Niveau). Auch die Modellresultate unterscheiden sich nicht statistisch signifikant, außer im Korrekturaufwand nach dem Spezifikationsreview (5 %-Niveau). Ich führe darum die mangelnde statistische Aussagekraft auf die wenigen verfügbaren Werte und den geringen Unterschied in der Vorbereitungsintensität zurück.

7.6.2 Kreuzvalidierung als Ersatz für die Prognose

Für die Kreuzvalidierung werden alle Projekte des Praktikums per Zufallsauswahl auf 10 Datengruppen verteilt. Die Eingabewerte einer Datengruppe sind Mittelwerte der anderen 9 Datengruppen. Die Eingaben sind so gewählt, dass die Situation der Projektplanung ungefähr nachgestellt wird (Tabelle 69). Dabei werden die Kalibrierungsparameter aus Daten abgeschlossener Projekte berechnet. Ich nehme an, dass der Code-Umfang hinreichend genau mit Function Points geschätzt werden kann. Der Prüfprozess kann im Voraus festgelegt werden. Der Fehlerfaktor muss wieder für alle Projekte angenommen werden, weil Daten aus dem Einsatz der Software nicht verfügbar sind.

Prozess- und Produktmerkmale	Prüfprozess	Berechnete Werte aus anderen Datengruppen
• Umfang des Codes	• Gutachterzahl und Vorbereitungsintensität im Spezifikationsreview • Intensität des Black-Box-Tests und Anweisungsüberdeckung des Glass-Box-Tests	• Umfangsfaktoren Spezifikation und Entwurf • Aufwandsfaktor • Dauerfaktor • Verteilung auf Fehlerart • Fehlerfaktor[a]

Tabelle 69: Individuelle Eingaben der Kreuzvalidierung

a. aus allen Projekten und nicht aus anderen Datengruppen

Die Daten aus anderen Datengruppen des Software-Praktikums sind aus verfügbaren Metriken der Projekte im Praktikum (Tabelle 42, Seite 161) berechnet:

- Spezifikations-, Entwurfs- und Codeumfang wurden gemessen, daraus können die Umfangsfaktoren berechnet werden.
- Die Kalibrierungsparameter für Dauer und Aufwand werden mit COCOMO II berechnet; für Aufwand und Dauer stehen aber nur die Vorgaben der Prüfungsordnung beziehungsweise der Betreuer zur Verfügung.
- Die Verteilung auf die Fehlerart wird aus gemessenen Werten für die Fehlerzahlen getrennt nach Fehlerart berechnet.

Die Abweichungen bei der Kreuzvalidierung sind ähnlich wie bei der individuellen Analyse. Tabelle 70 zeigt den Median der Abweichungen *MLE*[1] und den Anteil der Projekte innerhalb der 2-db-Grenze für den *MLE*, *pred*(2 dB). Die Genauigkeit nimmt ab, deutlich bei den Korrekturaufwänden. Die Abweichung übersteigt 2 dB im Median. Trotz dieser hohen Abweichung liegen aber zwischen 20 % und 60 % der Resultate innerhalb der 2-dB-Grenze. Dies deutet auf starke unkontrollierte Einflüsse in einem großen Teil der Projekte hin.

		Spez.-fehler	Entwurfs-fehler	Code-fehler	Fehler gesamt	Korrektur-aufwand
Spez.-review	*MLE* (dB)	1,6				3,2
	pred(2 dB)	58 %				35 %
Entwurfs-review	*MLE* (dB)		2,8		2,9	3,5
	pred(2 dB)		44 %		39 %	22 %
Modultest	*MLE* (dB)			3,0	3,4	4,9
	pred(2 dB)			50 %	43 %	21 %
Systemtest	*MLE* (dB)			2,2	2,1	3,2
	pred(2 dB)			44 %	50 %	33 %

Tabelle 70: Genauigkeit der Kreuzvalidierung

Die Resultate für intensive und oberflächliche Spezifikationsreviews sind dagegen in der Kreuzvalidierung kaum weniger genau (Tabellen 71, 72 und 73). Modellresultate und Istwerte sind konsistent für entdeckte Fehler im Spezifikationsreview und im Systemtest. Sie sind konsistent für den Korrekturaufwand. Wieder unterscheiden sich die Modellresultate für den Korrekturaufwand nach dem Systemtest am deutlichsten von den Istwerten.

7.6.3 Bewertung und Folgerungen

Bewertung der Resultate unterschiedlicher Prüfintensität

Der Unterschied zwischen mehr oder weniger intensiven Reviews wird konsistent in den Modellresultaten und in den Projektwerten sichtbar. Dies stützt die Annahmen des Modells für Zusammenhänge in Reviews und für Zusammenhänge der Fehlerentstehung und Fehlerentdeckung. Das Modell ist ausreichend genau, um selbst diesen kleinen Unterschied sichtbar zu machen. Der Unterschied ist aber weder in den Modellresultaten noch in den Projektwerten statistisch signifikant. Da Modell und Realität betroffen sind, führe ich dies aber auf die geringe Anzahl Werte, den gerin-

1. Median des $LE = 10 \cdot |\log(Modellresultat/Istwert)|$

Entdeckte Fehler und Korrekturaufwand nach Spezifikationsreview	Spez.-fehler		Korrekturaufwand (Eh)	
	Ist	Modell	Ist	Modell
Mittelwert gründliche Reviews	39,2	34,7	12,2	8,1
Median gründliche Reviews	35,0	34,2	9,8	7,6
Anzahl Teams	6		4	
Mittelwert oberflächliche Reviews	26,4	27,3	8,3	7,3
Median oberflächliche Reviews	24,0	30,9	6,9	7,9
Anzahl Teams	5		5	

Tabelle 71: Fehler und Korrekturaufwand für Spezifikationsreviews

Entdeckte Spezifikationsfehler und Korrekturaufwand im Systemtest	Spez.-fehler		Korrekturaufwand (Eh)	
	Ist	Modell	Ist	Modell
Mittelwert gründliche Reviews	0,4	0,6	4,3	10,4
Median gründliche Reviews	0	0,4	2,1	9,5
Anzahl Teams	6		4	
Mittelwert oberflächliche Reviews	1	1,9	12,3	12,0
Median oberflächliche Reviews	0	2,2	10,9	16,3
Anzahl Teams	6		3	

Tabelle 72: Fehler und Korrekturaufwand im Systemtest

Nutzen durch Spezifikationsreview im Systemtest	Entfallender Korrekturaufwand (Eh)	
	Modell[a]	Modell[b]
Mittelwert gründliche Reviews	8,9	8,9
Median gründliche Reviews	7,3	9,3
Anzahl Teams	7	4
Mittelwert oberflächliche Reviews	5,4	5,4
Median oberflächliche Reviews	8,5	5,0
Anzahl Teams	7	3

Tabelle 73: Berechneter Nutzen durch das Modell

a. Dargestellt werden Mittelwerte und Vergleichswerte für alle Teams.
b. Dargestellt werden nur Werte der Teams, für die Istwerte verfügbar sind.

gen Unterschied im Vorbereitungsaufwand und die große Streuung durch unkontrollierte Variablen zurück.

Bewertung der Streuung in den Projektdaten und der Modellgenauigkeit

Die 2-dB-Grenze für ein valides Modell wird von etwa der Hälfte der Resultate überschritten. Dies lässt sich aber auf die speziellen Eigenschaften des Praktikums zurückführen, so dass in Industrieprojekten eine höhere Genauigkeit zu erwarten ist:

- Die Korrektur umfasst typisch 10 bis 30 Fehler und Aufwände von wenigen Stunden pro Prüfung. Die Eigenschaften eines einzelnen Fehlers machen sich im Aufwand stärker bemerkbar als in großen Projekten, bei denen sich diese Unterschiede ausmitteln.
- Im Praktikum waren einige Daten nicht verfügbar. Vor allem fehlen Gesamtaufwand und Zahl der Fehler nach Auslieferung. In der Industrie sind diese beiden Daten häufiger verfügbar als andere. Im Praktikum wurden Unterschiede der Entwurfsreviews und des Modultests nicht erfasst. Dieser unkontrollierte Einfluss wirkt sich direkt und in den folgenden Phasen auf Fehlerzahlen und Korrekturaufwände aus.
- Die Istwerte des Praktikums streuen über einen großen Bereich. Ein Teil der Streuung wird nicht durch das Modell erklärt. Er lässt sich auf die unterschiedliche Motivation, Erfahrung und Fähigkeiten der Studenten zurückführen. Sie haben zwar die gleichen Lehrveranstaltungen besucht, und Inhalt und Termine des Praktikums sind vorgegeben und werden kontrolliert. Für viele Studenten ist es aber das erste Projekt. Einige Teilnehmer haben bereits umfangreich Programmiererfahrung, während für andere Teilnehmer die Programmiersprache neu ist. Die Teams sind unterschiedlich motiviert. Ein Teil versucht, das Praktikum mit möglichst wenig Aufwand zu bestehen. Andere Teams versuchen, ein möglichst perfektes Produkt zu entwickeln. Weil sich die Teams selber finden, finden sich auch meist Teilnehmer ähnlicher Motivation, Kenntnisse und Fähigkeiten zu einem Team. In der Industrie werden diese Merkmale innerhalb einer Organisation eingeebnet, weil sich die Entwickler eine gemeinsame Firmen- und Projektkultur teilen; sie haben bereits Projekte durchgeführt, sind also ähnlich motiviert und erfahren. In einem professionellen Umfeld mit Vorgesetzten kann sich die Motivation nicht so stark auswirken.
- Andere nicht kontrollierte Einflüsse können nicht gemessen werden; dazu gehören einzelne Fehler, die extrem aufwändig zu suchen und zu beheben sind. Bei geringem Projektumfang oder wenigen einzelnen Tätigkeiten können sich einzelne Abweichungen vom Mittelwert nicht ausgleichen.

Verglichen mit den Abweichungen der Top-down-Kostenschätzung ganzer Projekte in Kemerer (1987) und Boehm (2000) bewerte ich die Resultate aus einem weiteren Grund als plausibel: Die einzelnen Aktivitäten, die vom Modell abgebildet werden, lassen sich kaum steuern. In einem ganzen Projekt dagegen kann die Projektleitung auf Abweichungen reagieren. Sie kann die Abweichungen darum zumindest teil-

weise ausgleichen, so dass sie weniger stark das Gesamtergebnis bestimmen. Ich folgere daraus, dass bei detaillierten Modellen mit stärkeren Abweichungen als bei Modellen für ganze Projekte gerechnet werden muss.

Die Abweichungen für den Korrekturaufwand und die extremen Ausreißer einzelner Fehler zeigen, dass die Korrektur ein großes Risiko für Projektverzögerungen birgt. Darum folgere ich, dass ein einzelner Wert als Modellresultat kaum ausreicht, sondern dass ein Bereich, der die zu erwartende Spanne zeigt, sinnvoller ist. Die Prognoseergebnisse ergänzen also andere Verfahren.

Der Vergleich der Genauigkeit zwischen der individuellen Analyse und der Kreuzvalidierung zeigt, dass CoBe empfindlich auf die Kalibrierungsparameter reagiert. Daraus folgere ich, dass unbekannte Parameter variiert werden sollten, um diese Unsicherheit abzubilden. Diese Menge von Modellen für ein Projekt wird im Folgenden als Modellvarianten bezeichnet. Ihre Resultate ergeben einen Bereich, der diese Unsicherheit sichtbar macht. Auf der anderen Seite erlaubt das Modell aber, Unterschiede in Projektresultaten ohne diese Streuung darzustellen. Dies ermöglicht, den Nutzen direkt sichtbar zu machen. In realen Projekten kann der Unterschied in Projektresultaten, der durch unterschiedliches Vorgehen in Prüfungen verursacht wurde, durch andere Einflüsse überdeckt werden.

Bewertung der Validierung

Eine eigentliche Validierung wurde nicht erreicht, weil das Modell geändert und dann erneut mit den Daten verglichen wurde. Für die Validität des Modells spricht aber, dass die beiden Modelländerungen auf empirischen, unabhängigen Daten basieren. Unabhängige Daten bedeutet, dass die Daten zur Quantifizierung nicht zum Vergleich herangezogen wurden, sondern aus der Literatur stammen. Die Änderungen sind lokal eng begrenzt (Abbildungen 66 und 67). Auch die Notwendigkeit zur Kalibrierung wird durch unabhängige Quellen bestätigt. Aufwandsfaktor, Dauerfaktor und Fehlerfaktor verändern nicht die Verhältnisse zwischen den einzelnen, detaillierten Modellresultaten. Selbst bei einer Anpassung der Verteilung der Fehler auf die Fehlerarten ändern sich die Verhältnisse, z.B. der entdeckten Fehler, innerhalb einer Art nicht. Der Vergleich vor und nach der Modelländerung erfolgte nicht gegen exakt gleiche Daten, statt dessen wurden individuelle Projektwerte erst nach der Änderung betrachtet.

Gegen die externe Validität spricht, dass es sich um kleine, studentische Projekte handelt. Eine Validierung mit Industrieprojekten ist darum notwendig. Die Projektwerte streuen im Praktikum deutlich, dies lässt sich nicht auf Industrieprojekte übertragen. Die interne Validität ist durch unvollständige Daten und den geringen Unterschied im Prüfprozess bedroht. Die unvollständigen Daten, vor allem für den Gesamtaufwand und für die Zahl ausgelieferter Fehler, verschlechtern die Modellgenauigkeit. Der einheitliche Prüfprozess führt zu geringen Unterschieden, während die individuellen Merkmale der Teams im Praktikum die Istwerte deutlich prägen. Insgesamt führen die Bedrohungen also zu einer zu negativen Bewertung der Modellvalidität.

Kapitel 8

Evaluation des Modells

In diesem Kapitel wird die Sensitivitätsanalyse von CoBe (Abschnitt 8.1), die Optimierung mit CoBe (Abschnitt 8.2) und die Validierung von CoBe mit Industriedaten beschrieben. Für die Validierung in der Industrie wird das Vorgehen geklärt (Abschnitt 8.3). Die Projekte und die Validierungsresultate werden dargestellt (Abschnitte 8.4 und 8.5). Die Modellprüfung von CoBe wird in Abschnitt 8.6 bewertet. Den Einsatz des Modells zeige ich an Beispielszenarien; dabei werden die Modellresultate auch mit Erfahrungen bei Prozessverbesserungen verglichen (Abschnitt 8.7).

8.1 Sensitivitätsanalyse

Durch die Sensitivitätsanalyse wird die Einflussstärke von Modellparametern auf Modellresultate untersucht. Sie zeigt das Verhalten des Modells.

8.1.1 Ziele und Hypothesen der Sensitivitätsanalyse

Die Sensitivitätsanalyse von CoBe orientiert sich an Hypothesen, die aus dem Modellkonzept abgeleitet sind (Kapitel 3). Im Konzept wird festgelegt, dass Entscheidungen über Prüfungen durch Modelleingaben dargestellt werden. Das Konzept basiert also auf den folgenden Hypothesen:

H 1: Die Entscheidungen über Prüfparameter bestimmen die Qualitätskosten.

H2: Es gibt keinen allgemeingültigen Prüfprozess, der in allen Projektsituationen zu einem optimalen Ergebnis führt.

H3: Die Qualitätskosten sind durch die Kombination der Entscheidungen und der Prozess- und Produktmerkmale bestimmt, so dass die Auswirkungen schwierig zu durchschauen sind.

H4: Kurzfristige Kosten zu minimieren und langfristige Kosten zu minimieren sind konkurrierende Ziele.

Zusätzlich wird untersucht, wie sich unsichere Eingaben auswirken. Dazu gehören in CoBe die Kalibrierungsparameter, die aus Archivdaten stammen, der Software-Umfang, der bei der Planung geschätzt wird, und die Fehlerfolgekosten, die grob abgeschätzt werden.

H5: Die Fehlerfolgekosten bestimmen die Modellresultate.

H6: Die Kalibrierung ist notwendig.

H7: Der Software-Umfang hat einen wesentlichen Einfluss auf die Resultate.

Ich lege den Schwerpunkt auf die Frage, welche Parameter möglichst genau sein müssen und wie die Aussagen, die mit CoBe gewonnen werden können, durch unsichere Eingaben beeinflusst werden.

8.1.2 Vorgehen zur Sensitivitätsanalyse

Szenarien

Die Hypothese 3 kann nur mit einer globalen Sensitivitätsanalyse untersucht werden, bei der die Eingaben variiert und daraus Kombinationen gebildet werden (Saltelli et al., 2008). Mit einem naiven Ansatz werden also alle Parameter von CoBe variiert. Dann wird statistisch analysiert, wie stark jeder Parameter wirkt. Dieser Ansatz ist problematisch, weil nicht jede Kombination, die in CoBe möglich ist, auch sinnvoll ist. Beispielsweise sind Projekte mit oberflächlichem Prüfprozess und sehr hohen Fehlerfolgekosten unrealistisch. Diese Kombinationen verzerren die Analyseergebnisse und überdecken andere, relevante Zusammenhänge. Darum wähle ich einen Ansatz mit Szenarien, die durch Prozess- und Produktmerkmale und den Prüfprozess festgelegt werden. Für die Szenarien werden diejenigen Eingaben variiert, die für die Prüfung der Hypothesen notwendig sind.

Analyse

Die Sensitivitätsanalyse von CoBe soll zwei unterschiedliche Fragen beantworten: Wie wirken sich Entscheidungen, d.h. die Eingaben für Prüfprozess und Prüfparameter, aus? Wie wirkt sich die Unsicherheit über das Projekt zum Zeitpunkt der Planung aus? Für die globale Sensitivitätsanalyse wird die Unsicherheit durch ein Monte-Carlo-Experiment mit einer statistischen, pseudo-zufälligen Auswahl der unsicheren Eingaben modelliert (Sobol und FAST nach Saltelli et al., 2008). Für die Analyse der Entscheidungen ist diese Art der Sensitivitätsanalyse nicht geeignet, weil die Entscheidungen bewusst durch Projektleiter und QS-Verantwortliche getroffen werden. Die Entscheidungen sind also nicht unsicher und nicht durch Zufallseffekte geprägt, sondern gegeben. Die beiden Fragen nach den Wirkungen der Entscheidungen und der unsicheren Eingaben können darum nur mit unterschiedlichen Analysen beantwortet werden. Die Sensitivitätsanalyse für die Entscheidungen erfolgt darum mit fest vorgegebenen Eingabekombinationen. Da dann eine Analyse mit Sobol und FAST nicht mehr möglich ist, erfolgt die Analyse graphisch und zusätzlich statistisch durch lineare Regression. Die Kalibrierungsparameter und der Umfang dagegen sind unsicher, weil ihr wahrer Wert bei der Planung unbekannt ist. Dafür werden Analysen mit statistischer Auswahl der Eingaben eingesetzt. Die Eingaben für Fehlerfolgekosten werden dagegen durch die unterschiedlichen Szenarien variiert, weil die verwendeten Klassifizierungen durch Größenordnungen definiert sind und bereits eine Klasse einen großen Bereich abdeckt.

Eingaben

Bei einer Analyse mit Szenarien bleibt die Annahme bestehen, dass die Qualitätskosten durch die Kombination der Entscheidungen bestimmt sind. Diese Annahme kann nur überprüft werden, wenn die Wechselwirkungen oder Interaktionseffekte zwischen den Entscheidungen erkannt werden können. Darum müssen die verschiedenen Kombinationen der möglichen Entscheidungen betrachtet werden.

In CoBe gibt es viele Entscheidungen, weil es viele Prüfungen gibt und weil für jede Prüfung viele Prüfparameter eingegeben werden können. Da es also viele Parameter mit vielen möglichen Werten gibt, sind sehr viele Kombinationen möglich; die kombinatorische Vielfalt wird sehr groß. Dies verursacht Rechenaufwand. Vor allem aber wird die Analyse komplex, weil sehr viele Eingabekombinationen und Resultate betrachtet und in Beziehung zueinander gesetzt werden müssen.

Es ist also notwendig, die Analyse zu vereinfachen. Darum werden nicht alle Eingabekombinationen aller Reviews analysiert, stattdessen wird im Folgenden ein Review stellvertretend für alle Reviews betrachtet. Dies ist möglich, da alle Reviews gleich modelliert sind und sich nur in der Quantifizierung unterscheiden. Für diese Betrachtung ist das Spezifikationsreview gut geeignet, weil Spezifikationsfehler am teuersten sind, wenn sie später entdeckt werden. Dadurch wirken sich die Entscheidungen über das Review also stärker als Entscheidungen über andere Reviews aus. Im Entwurfs- und Codereview können zwar mehr Fehler entdeckt werden, da neue Fehler beim Entwerfen und Codieren entstehen. Deren Entdeckungszeitpunkt wirkt sich aber weniger stark aus.

Auch für die Tests gilt, dass sie gleich modelliert sind und nur unterschiedlich quantifiziert. Es reicht also wieder, einen Test stellvertretend zu untersuchen. Dazu ist der Systemtest gut geeignet, weil er sich stärker als andere Tests auswirkt. Dafür gibt es zwei Gründe: Mit dem Systemtest können mehr Fehler als in anderen Tests entdeckt werden, weil der Systemtest mit den höchsten Fehlerentdeckungsquoten quantifiziert ist. Ein Testfall ist im Systemtest am teuersten.

Modellresultate

Die Modellresultate, die untersucht werden, leiten sich direkt aus der Hypothese H 4 ab: Die Wirkungen auf die Projekt-Qualitätskosten (im Projekt anfallende Prüf- und Fehlerkosten) und die Gesamt-Qualitätskosten (Prüfkosten und Fehlerkosten im Projekt und in der Wartung) werden untersucht.

8.1.3 Szenarien für die Sensitivitätsanalyse

Für die Sensitivitätsanalyse sollen die Szenarien die unterschiedlichen Merkmale von Software-Projekten abdecken, um den Einfluss dieser Merkmale zu zeigen. Produktivität und Qualität in Software-Projekten können durch über 250 Merkmale beeinflusst werden (Jones, 2003). Zur Produktivitätsbewertung und zur Kostenschätzung werden über 20 Merkmale erfasst (IEEE Std. 1045, 1992; Boehm, 2000). Die wichtigsten Merkmale sind nach Jones (2003):

- Die Software-Art: Jones (1996, 2003) unterscheidet System-Software, Software für den Markt, Informationssysteme, Auftragsprojekte, Software für das Militär und Endbenutzer-Software.
- Die Projektmerkmale: Dazu gehören Umfang, Komplexität, Randbedingungen, Entwicklungsart (Neuentwicklung, Wartung oder Verbesserung), Anwendungsart und Scope (System, Programm, Modul).
- Die Technologie mit formalen Methoden, Projektmanagement, Qualitätssicherung, Programmiersprachen und Wiederverwendung.
- Die Soziologie umfasst Erfahrung, Organisation, Moral und Prozessreife.
- Die Ergonomie der Arbeitsplätze und die Kommunikation am Arbeitsplatz.
- Merkmale durch Internationalität sind lokale Gesetze, Personalkosten, Arbeitszeit und Mitarbeiterverhalten.

Für die Analyse werden die Merkmale durch unterschiedliche Parameterwerte für CoBe abgedeckt: Unterschiedliche Software-Arten werden durch unterschiedliche Fehlerfolgekosten und Prüfprozesse abgedeckt. Unterschiede der Projektmerkmale werden durch verschiedenen Umfang, die Technologie durch verschiedene Prüfprozesse und Wiederverwendung dargestellt. Die Kalibrierung fasst alle Merkmale als Einfluss zusammen. Die Erfahrung wird durch die Kompetenz abgebildet. Personalkosten werden nicht variiert, weil sich dieser Faktor gleichförmig auf alle Resultate auswirkt. Die verschiedenen Eingaben bilden folgende Szenarien:

- Nominalszenario: Das Nominalprojekt bildet den Normalfall mit durchschnittlichen Eingabewerten ab; ein Auftragsprojekt.
- Szenario mit Wiederverwendung: Das Projekt mit Wiederverwendung zeigt die Auswirkungen durch wiederverwendete Software. Andere Eingaben entsprechen dem Nominalprojekt.
- Kritisches Szenario: Das Projekt für sicherheitskritische System-Software hat einen kleineren Umfang. Alle Prüfungen werden vollständig durchgeführt. Es werden erfahrene Mitarbeiter eingesetzt. Die Prozessreife ist hoch. Die Technologie unterscheidet sich von anderen Projekten, weil eine andere Programmiersprache eingesetzt wird.
- Initialszenario: In diesem Projekt wird ein Produkt mit kleinem Umfang und geringen Fehlerfolgekosten entwickelt. Das Projekt hat eine niedrige Prozessreife, es werden wenig Prüfungen durchgeführt. Damit die Wirkung des Spezifikationsreviews untersucht werden kann, gibt es das Szenario mit und ohne Spezifikationsreview.
- Großes Szenario: Ein großes Projekt unterscheidet sich im Umfang und in der hohen Prozessreife vom Nominalprojekt.

- Prozedurales Szenario. Es unterscheidet sich vom Nominalprojekt, weil keine objektorientierte Programmiersprache eingesetzt wird.

8.1.4 Eingaben für die Sensitivitätsanalyse

Das Nominalszenario ist die Basis aller Szenarien. Tabelle 74 zeigt die für alle Szenarien gleichen Eingaben, Tabelle 75 die unterschiedlichen Eingaben.

Eingabe	Werte
Prüfprozess	Nominales Spezifikationsreview, nominales Entwurfsreview mit 5 gründlichen Gutachtern, vollständige Prüfung, kein Codereview.
	Nominaler Modul-, Systemintegrations- und Systemtest jeweils mit Funktionen und Äquivalenzklassen und vollständiger Wiederholung, Feldtest.
	Nach der Korrektur eines Fehlers in der Wartung wird die Korrektur gezielt getestet, dazu wird ein Teil des Modultests, des Sytemintegrationstests und des Systemtests wiederholt.
Personalkosten	200 000 Euro pro Entwicklerjahr, 110 Euro pro Entwicklerstunde
Fehlerfolgekosten	Verwendungshäufigkeit: Ein Fehler, der 10 000 Euro Schaden oder mehr verursacht, wird sofort korrigiert. Bei anderen Fehlern wird die Software zehnmal verwendet, bis ein Fehler korrigiert wird. Fehler, die nie im Einsatz auftreten, verursachen keinen Schaden. Auftretenswahrscheinlichkeit: Die Fehler sind auf die Klassen Auftretenswahrscheinlichkeit gleichmäßig verteilt (Abschnitt 7.4.3).
Kalibrierung	Aufwandsfaktor, Dauerfaktor und Fehlerfaktor: 1,0, Umfangsfaktor Code für Java: 53 Anweisungen pro Function Point, Umfangsfaktoren Spezifikation, Entwurf: 0,44 Seiten pro Function Point.

Tabelle 74: Eingaben für alle Szenarien

Tabelle 76 zeigt die Eingabewerte, die für die Sensitivitätsanalyse verwendet werden. Der abgedeckte Bereich orientiert sich an praxistypischen Werten. Im Glass-Box-Test betrachte ich für die Analyse nur die Anweisungsüberdeckung, weil andere Überdeckungsmetriken vor allem für sicherheitskritische Software eingesetzt werden. Die Modellresultate werden für jede Kombination berechnet. Zwei Sonderfälle werden eingefügt: Der Fall, in dem kein Review stattfindet, und der Fall, in dem kein Systemtest stattfindet.

8.1.5 Statistische Sensitivitätsanalyse

Analyse

Mit der Analyse durch lineare Regression wird die Einflussstärke einer Modelleingabe auf ein Modellresultat durch den standardisierten Regressionskoeffizienten dar-

Szenario	Unterschiedliche Eingaben und Eingabewerte
Nominal	Nominaler Prüfprozess, nominale Kalibrierung, Java, 1000 FP[a] neu, maximaler Schaden: 10 000 Euro, Fehler sind auf Schadensklassen (bis 10 000 Euro) gleichmäßig verteilt.
Wiederverwendung	Nominalszenario, aber mit 700 FP wiederverwendeter und 300 FP neuer Software, maximaler Schaden: 10 000 Euro wie im Nominalszenario
Kritisch	Vollständiger Prüfprozess, Kalibrierung für hohe Prozessreife (Aufwands- und Fehlerfaktor 0,5), C++, 100 FP, maximaler Schaden: 10 000 000 Euro, Fehler sind auf Schadensklassen gleichmäßig verteilt
Initial	Prüfprozess ohne Reviews, ohne Integrations- und Feldtest, nominale Kalibrierung, Java, 100 FP, maximaler Schaden: 10 Euro, mit und ohne Spezifikationsreview
Groß	Nominaler Prüfprozess, Kalibrierung für hohe Prozessreife (Aufwands- und Fehlerfaktor 0,5), Java, 10 000 FP neu, maximaler Schaden: 10 000 Euro wie im Nominalszenario
Prozedural	Nominaler Prüfprozess, nominale Kalibrierung, Java, 1000 FP neu, maximaler Schaden: 10 000 Euro wie im Nominalszenario

Tabelle 75: Unterschiedliche Eingaben für Szenarien

a. FP: Function Points

Prüfung	Eingabe	Eingabewerte
Spezifikationsreview	Gutachterzahl	2, 3, 4, 5 Gutachter
	Intensität	5, 10, 20 Seiten pro Stunde
	Gutachterkompetenz	sehr niedrig, nominal, sehr hoch
	Prüflingsüberdeckung	50 %, 100 %
	Ein Sonderfall ohne Spezifikationsreview (0 % Überdeckung).	
Systemtest	Abdeckung Black-Box-Testtechniken	Funktionen, Äquivalenzklassen, Sonderfälle
	Anweisungsüberdeckung Glass-Box-Test	0 %, 80 %, 100 %
	Testerkompetenz	sehr niedrig, nominal, sehr hoch
	Testwiederholung	Mit und ohne vollständige Wiederholung
	Ein zusätzlicher Sonderfall ohne Systemtest.	

Tabelle 76: Variierte Eingaben

gestellt (Saltelli et al., 2008). Standardisierte Regressionskoeffizienten werden mit einer Regressionsgleichung berechnet, die aus Regressionskoeffizienten, den variierten Modelleingaben und dem Modellresultat aufgebaut ist. Die Regressionskoeffizienten werden aus Eingabewerten und zugehörigem Resultatswert berechnet[1]. Die standardisierten Regressionskoeffizienten werden auf den Wertebereich der Eingabe und des Resultats normiert. Sie liegen zwischen -1 und 1. Ein hoher Betrag bedeutet einen starken Einfluss; ein Koeffizient mit Wert 0 bedeutet keinen Einfluss. Ein positiver Koeffizient bedeutet einen positiven Zusammenhang, ein negativer Koeffizient einen negativen. Im Folgenden wird mit Koeffizient der standardisierte Regressionskoeffizient bezeichnet. Die Analyse erfolgt mit dem Statistikpaket R (2008).

Die lineare Regression ist nur dann aussagekräftig, wenn sich das Modell im Bereich der variierten Prüfparameter weitgehend linear verhält. Das Bestimmtheitsmaß R^2 erlaubt, die Linearität zu prüfen, weil es ausdrückt, welcher Anteil der Streuung der abhängigen Variable durch die unabhängige erklärt wird (Fahrmeir et al., 2007). Mit R^2 über 70 % ist die Analyse aussagekräftig (Saltelli et al., 2008). Nicht-lineare Zusammenhänge werden graphisch gezeigt (Fahrmeir et al., 2007; Saltelli et al., 2008).

Interaktionseffekte sind Wirkungen, die ausschließlich durch mehrere Eingaben gemeinsam auftreten. Sie werden von einem einfachen linearen Regressionsmodell nicht erfasst, darum analysiere ich diese Effekte graphisch. Effekte zwischen den Szenarien werden durch Vergleich der Koeffizienten sichtbar.

Analyseergebnisse

Aus den Eingabewerten ergeben sich 3961 unterschiedliche Kombinationen. Die Annahme der Linearität gilt für den untersuchten Bereich, da R^2 in allen Fällen über 70 % liegt (Tabelle 77). Projekt-Qualitätskosten sind Qualitätskosten, die im Projekt anfallen; Gesamt-Qualitätskosten enthalten Qualitätskosten im Projekt, in der Wartung und im Einsatz des Produkts.

R^2 für Qualitätskosten	Nominal	Kritisch	Groß	Initial	Initial Spez.-review	Prozedural	Wiederverw.
Gesamt	0,90	0,90	0,88	0,90	0,90	0,87	0,90
Projekt	0,92	0,91	0,91	0,92	0,92	0,91	0,92

Tabelle 77: Bestimmtheit der Resultate

1. Alle Eingaben mit Ausnahme der Testerkompetenz und der Testwiederholung sind auf einer Rationalskala. Die Testwiederholung kann als Dummy-Variable direkt verwendet werden. Die Testerkompetenz befindet sich auf einer Ordinalskala, ich nehme für die Regression gleiche Abstände zwischen den Werten an.

Tabelle 78 zeigt Minimum und Maximum der Gesamt- und Projekt-Qualitätskosten. Der Faktor ist das Verhältnis zwischen maximalen und minimalen Qualitätskosten. Ein Faktor 2 bedeutet also, dass die maximalen Qualitätskosten doppelt so hoch wie die minimalen sind.

Die Wirkung der Entscheidungen ist groß: Die Gesamt-Qualitätskosten schwanken um Verhältnisse zwischen 1 : 1,2 und 1 : 6,3, die Projekt-Qualitätskosten sogar bis zu einem Verhältnis von 1 : 14,2. Diese Streuung hängt vom Szenario ab: Mit hohen Fehlerkosten schwanken die Gesamt-Qualitätskosten besonders stark, mit niedrigen Fehlerkosten oder Wiederverwendung besonders wenig. Die Projekt-Qualitätskosten streuen besonders bei den Projekten mit einem initialen Prüfprozess; am geringsten beim kritischen Projekt mit den meisten Prüfungen. Die Qualitätskosten steigen überproportional mit dem Umfang: Obwohl im großen Szenario die Prozessreife hoch ist, steigen die Qualitätskosten überproportional, also um mehr als das Zehnfache der Kosten im Nominalszenario.

Szenario	Gesamt-Qualitätskosten			Projekt-Qualitätskosten		
	Min. (Euro)	Max. (Euro)	Faktor	Min. (Euro)	Max. (Euro)	Faktor
Nominal	3 752 495	10 047 472	2,7	1 168 266	2 451 328	2,1
Kritisch	2 961 274	18 771 991	6,3	40 170	55 751	1,4
Groß	43 182 533	174 153 000	4,0	6 156 719	12 586 561	2,0
Initial	292 333	342 814	1,2	8 171	116 186	14,2
Initial Spez.-review	187 653	342 225	1,8	9 065	118 675	13,1
Prozedural	4 366 631	13 779 409	3,2	1 268 339	2 719 070	2,1
Wiederverwendung	1 147 818	2 226 785	1,9	303 676	573 563	1,9

Tabelle 78: Qualitätskosten

Der Vergleich mit COCOMO-II-Resultaten zeigt, dass die Projekt-Qualitätskosten, die CoBe berechnet, plausibel sind (Tabelle 79), weil die Qualitätskosten maximal etwa bei der Hälfte der Projektkosten liegen.

Tabelle 80 zeigt die Koeffizienten für die Gesamt-Qualitätskosten, Tabelle 81 für die Projekt-Qualitätskosten. Ein negativer Koeffizient bedeutet, dass eine höhere Eingabe die Kosten senkt, mehr Gutachter bedeuten beispielsweise weniger Qualitätskosten. Mit positivem Koeffizienten steigen die Kosten, eine höhere Vorbereitungsrate bedeutet also höhere Qualitätskosten. Alle Eingaben bis auf zwei Ausnahmen sind statistisch signifikant mit einem p-Wert < 0.001[1].

COCOMO-II-Resultate	Aufwand (EM)	Dauer (M)	Personal-bedarf	Personalkosten (Euro)
Nominal (1000 FP)	247	25	10	4 129 840
Kritisch (100 FP, hohe Prozessreife)	10	9	1	167 200
Groß (10 000 FP, hohe Prozessreife)	1554	45	34	25 982 880
Initial (100 FP)	20	11	2	334 400

Tabelle 79: Projektkosten mit COCOMO II

Koeffizienten für Gesamt-Qualitätskosten	Nomi-nal	Kritisch	Groß	Initial	Initial Spez.-review	Proze-dural	Wieder-verw.
Gutachterzahl	-0,14	-0,16	-0,10	-	-0,16	-0,11	-0,14
Vorbereitungsrate	0,29	0,33	0,20	-	0,35	0,23	0,31
Gutachterkompetenz	-0,52	-0,57	-0,36	-	-0,65	-0,41	-0,55
Prüflingsüberdeckung	-0,35	-0,40	-0,24	-	-0,42	-0,28	-0,37
Abdeckung Black-Box-Testtechniken	-0,21	-0,28	-0,12	-0,14	-0,06	-0,25	-0,24
Anweisungs-überdeckung	0,04	-0,29	0,32	0,19	0,07	0,39	-0,11
Testerkompetenz	-0,60	-0,32	-0,73	-0,90	-0,37	-0,60	-0,52
Testwiederholung	0,02	-0,01[a]	0,01[b]	0,20	0,08	0,02	0,03

Tabelle 80: Koeffizienten für Gesamt-Qualitätskosten

a. p-Wert < 0,1
b. statistisch nicht signifikant, p-Wert > 0,1

Tabelle 80 zeigt die folgenden Aussagen:

- Die Kompetenz der Gutachter und der Tester spielt die wichtigste Rolle. Die Koeffizienten liegen zwischen -0,32 und -0,90 und haben somit den höchsten Betrag. Je höher die Kompetenz, desto geringer sind die Qualitätskosten; in CoBe kosten Entwickler unabhängig von ihrer Kompetenz gleich viel.
- Die Eingaben für das Spezifikationsreview wirken in allen Szenarien in der gleichen Richtung, weil die Koeffizienten das gleiche Vorzeichen haben. Dies bedeutet für den hier untersuchten Bereich: Je mehr Gutachter prüfen, je mehr vom Prüfling

1. Die Wahrscheinlichkeit, dass die Nullhypothese fälschlicherweise abgewiesen wird, liegt unter 0,001.

Koeffizienten für Projekt-Qualitätskosten	Nominal	Kritisch	Groß	Initial	Initial Spez.-review	Prozedural	Wiederverw.
Gutachterzahl	-0,04	0,18	-0,02	-	0,03	-0,04	-0,03
Vorbereitungsrate	0,12	-0,13	0,09	-	-0,03	0,11	0,1
Gutachterkompetenz	-0,26	-0,24	-0,27	-	-0,01	-0,24	-0,24
Prüflingsüberdeckung	-0,14	0,16	-0,11	-	0,03	-0,13	-0,11
Abdeckung Black-Box-Testtechniken	0,54	0,57	0,56	0,57	0,57	0,48	0,57
Anweisungsüberdeckung	0,55	0,59	0,57	0,58	0,58	0,65	0,57
Testerkompetenz	0,37	-0,21	0,33	0,44	0,43	0,31	0,33
Testwiederholung	0,17	0,13	0,17	0,15	0,15	0,16	0,17

Tabelle 81: Koeffizienten für Projekt-Qualitätskosten

überdeckt wird, je gründlicher die Vorbereitung, desto geringer werden die Gesamt-Qualitätskosten. Dabei wirkt sich die Gutachterzahl am schwächsten aus. Die Einflussstärke verändert sich von Szenario zu Szenario, beispielsweise sinkt im großen Szenario die Einflussstärke aller Revieweingaben.

- Die Einflussstärke und die Einflussrichtung der Testeingaben verändern sich von Szenario zu Szenario: Die Abdeckung im Black-Box-Test senkt die Qualitätskosten. Im Initialszenario mit und ohne Spezifikationsreview und im großen Szenario schwindet dieser Einfluss. Die Wirkung der Anweisungsüberdeckung hängt vom Szenario und vom Prüfprozess ab: Im kritischen Szenario und mit Wiederverwendung sinken die Gesamt-Qualitätskosten; im Nominalszenario und im Initialszenario mit Spezifikationsreview ist der Einfluss gering; in den anderen Szenarien steigen die Qualitätskosten mit der geforderten Überdeckung von 80 % und 100 %, die zusätzlich zum Black-Box-Test erreicht werden soll.

Tabelle 81 zeigt den Einfluss der Prüfparameter auf die Qualitätskosten im Projekt:

- Der wichtigste Einfluss sind die Testparameter im Systemtest, weil ihre Koeffizienten am größten sind; sie liegen nahe oder über 0,5. Dies bedeutet: Je intensiver der Test, desto teurer wird das Projekt.
- Für die Projektkosten spielt die Kompetenz der Gutachter und Tester eine wichtige Rolle, weil die Koeffizienten einen großen Betrag aufweisen. Die Stärke des Einflusses hängt vom Szenario ab. Im kritischen Szenario ändert sich sogar die Richtung des Einflusses.
- Die Projekt-Qualitätskosten werden durch die Prüfparameter für das Spezifikationsreview wenig beeinflusst, weil die Koeffizienten niedrig sind. Abhängig vom

Szenario wird das Projekt etwas teurer oder etwas günstiger: Beispielsweise sinken im Nominalszenario die Kosten, wenn mehr begutachtet wird. Im kritischen Szenario steigen die Kosten im Projekt.

8.1.6 Graphische Sensitivitätsanalyse

Die graphische Analyse ergänzt die Regressionsanalyse, da sie nicht-lineare und nicht-additive Effekte zeigen kann (Saltelli et al., 2008). Um die Diagramme übersichtlich zu halten, werden die Eingaben einer Prüfung variiert, während die Eingaben der anderen Prüfung konstant bleiben.

Spezifikationsreview

Die Analyse des Spezifikationsreviews erfolgt also mit konstanten Eingaben für einen nominalen Systemtest. Abbildung 70 zeigt den Einfluss des Spezifikationsreviews auf die Gesamt-Qualitätskosten im Nominalszenario. Die Gesamt-Qualitätskosten an der y-Achse sind im linken Diagramm abhängig von den Reviewkosten dargestellt, im rechten Diagramm abhängig von der Überdeckung der Spezifikation. Die Datenpunkte in den Diagrammen sind nach der Gutachterkompetenz geordnet: Weiße Punkte stehen für niedrige, schwarze für nominale und graue für hohe Kompetenz. Die Kosten des Spezifikationsreviews ändern sich nicht mit der Kompetenz, weil die tatsächliche, messbare Vorbereitungsrate eingegeben wird; sie ist also fest vorgegeben und verändert sich nicht mit der Kompetenz.

Durch Mehraufwand im Spezifikationsreview können die Gesamt-Qualitätskosten gesenkt werden (Abbildung 70, links), im besten Fall um fast die Hälfte. Die Wirkung ist nicht linear, da mit hohen Kosten nur noch eine geringe Verbesserung erreicht werden kann. So sinken die Gesamt-Qualitätskosten ab rund 40 000 Euro Reviewkosten kaum noch. Notwendig ist eine Mindestkompetenz und die vollständige Prüfung der Spezifikation (Abbildung 70, rechts). Die Spezifikationsüberdeckung verstärkt den Effekt der Kompetenz, es handelt sich darum um Interaktionseffekte.

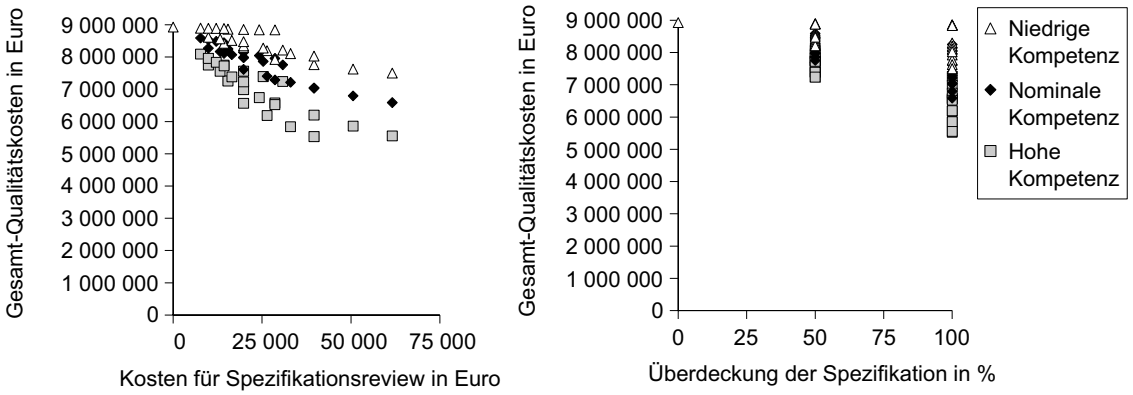

Abb. 70: Wirkung des Spezifikationsreviews auf die Gesamt-Qualitätskosten für Reviewkosten (links) und Prüflingsüberdeckung (rechts)

Die Fehlerfolgekosten und der Prüfprozess beeinflussen die Wirkung des Reviews. Dies zeigt Abbildung 71. Beide Diagramme sind aufgebaut wie das linke Diagramm in der vorigen Abbildung; sie zeigen die Gesamt-Qualitätskosten an der y-Achse in Abhängigkeit von den Kosten des Spezifikationsreviews an der x-Achse. Mit hohen Fehlerfolgekosten und intensivem Prüfprozess nützt das Review mehr, die Qualitätskosten können auf ein Drittel gesenkt werden (Abbildung 71 links, kritisches Szenario); mit oberflächlichem Prüfprozess und niedrigen Fehlerfolgekosten können die Qualitätskosten auf etwa zwei Drittel gesenkt werden (Abbildung 71 rechts, Initialszenario mit Spezifikationsreview).

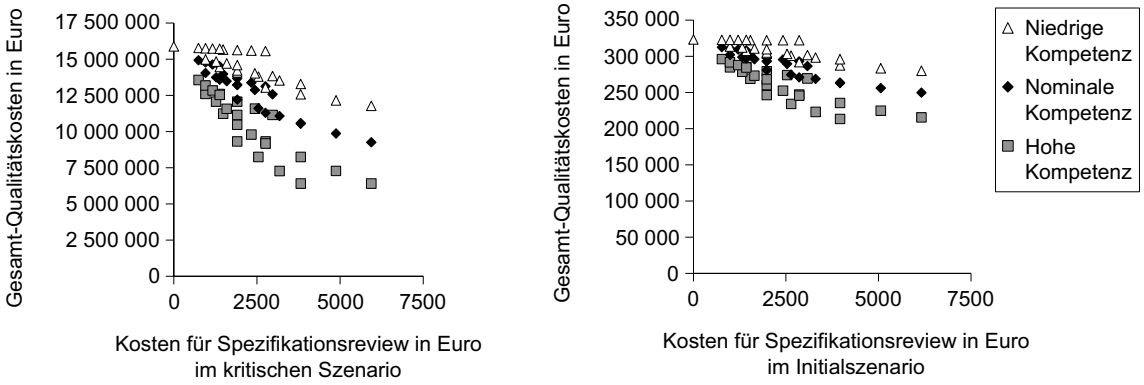

Abb. 71: Reviewwirkung auf die Gesamt-Qualitätskosten im kritischen Szenario (links) und im Initialszenario (rechts)

Abbildung 72 zeigt die Projekt-Qualitätskosten in Abhängigkeit von den Kosten für das Spezifikationsreview. Die Diagramme bestätigen die Ergebnisse in Tabelle 81. Das Spezifikationsreview beeinflusst die Projekt-Qualitätskosten wenig. Im besten Fall werden die Kosten gesenkt (Abbildung 72, links, Nominalszenario), im schlechtesten Fall steigen die Kosten leicht an (Abbildung 72, rechts, Initialszenario).

Systemtest

Die graphische Analyse des Tests basiert auf einem nominalen Spezifikationsreview. Abbildung 73 zeigt die Gesamt-Qualitätskosten in Abhängigkeit von den Kosten für den Systemtest. Die verschiedenen Testtechniken für den Black-Box-Test sind farblich unterschieden. Der Punkt auf der y-Achse kennzeichnet die Kosten ohne Systemtest.

Das Diagramm zeigt den Einfluss der Testparameter des Systemtests, weil sich im Nominalszenario die Parameter in etwa in der gleichen Größenordnung auswirken wie die Parameter des Spezifikationsreviews: In beiden Fällen werden die Gesamt-Qualitätskosten im Bereich zwischen rund 5 und 9 Millionen Euro verändert (Abbildungen 70 und 73). Die niedrigsten Gesamt-Qualitätskosten können im Nominalszenario durch einen Black-Box-Test mit Äquivalenzklassen und Sonderfällen erreicht werden. In ungünstigen Kombinationen steigen die Qualitätskosten aber.

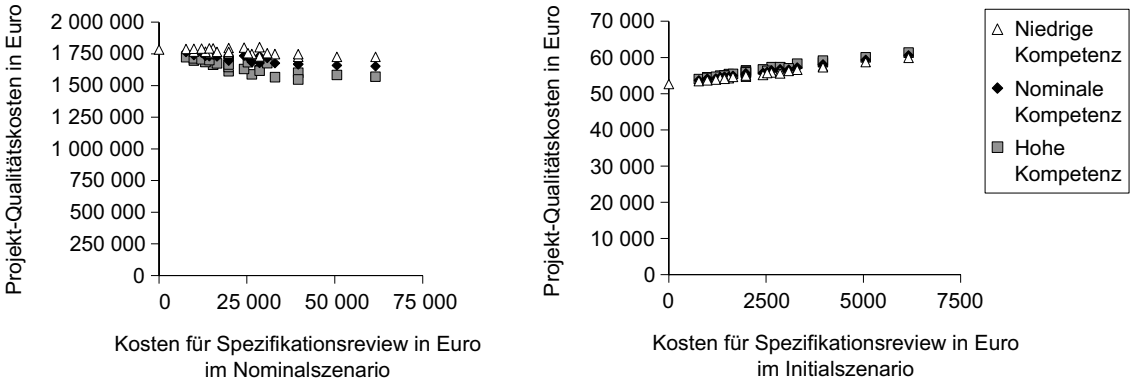

Abb. 72: Reviewwirkung auf die Projekt-Qualitätskosten im Nominalszenario (links) und im Initialszenario mit Spezifikationsionsreview (rechts)

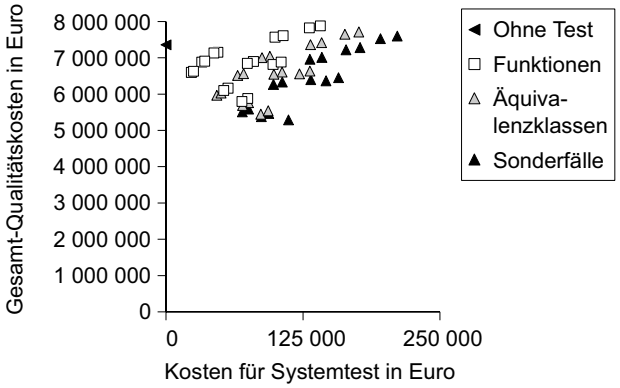

Abb. 73: Wirkung des Systemtests auf die Gesamt-Qualitätskosten, gegliedert nach Prüfparameter für den Black-Box-Test

Abbildung 73 zeigt dies mit den Gesamt-Qualitätskosten in Abhängigkeit von den Kosten für den Systemtest. Die Abbildung zeigt aber auch, dass nicht allein der Einfluss der Black-Box-Testparameter über die Qualitätskosten entscheidet. Zusätzlich spielen weitere Parameter eine Rolle: Der Anstieg der Qualitätskosten durch den Systemtest ist durch die Anweisungsüberdeckung bestimmt und hängt von der Testerkompetenz und den Fehlerfolgekosten ab.

Abbildung 74, links, zeigt dies anhand der Gesamt-Qualitätskosten für das Nominalszenario in Abhängigkeit von der geforderten Anweisungsüberdeckung. Mit niedriger Kompetenz (schwarze Punkte) steigen die Kosten mit der Überdeckung; mit hoher Kompetenz bleiben die Kosten nahezu konstant (graue Punkte). Mit hohen Fehlerfolgekosten sinken die Qualitätskosten (Abbildung 74, rechts, kritisches Szenario). Ein gründlicher Black-Box-Test lohnt sich also langfristig immer, ein Glass-Box-Test nur

bei hohen Fehlerkosten oder mit sehr kompetenten Testern. Daraus folgt, dass die vier Eingaben für Black-Box-Testparameter, dem Kriterium für die Anweisungsüberdeckung, dem Parameter der Kompetenz und den Fehlerfolgekosten durch Interaktionseffekte wirken.

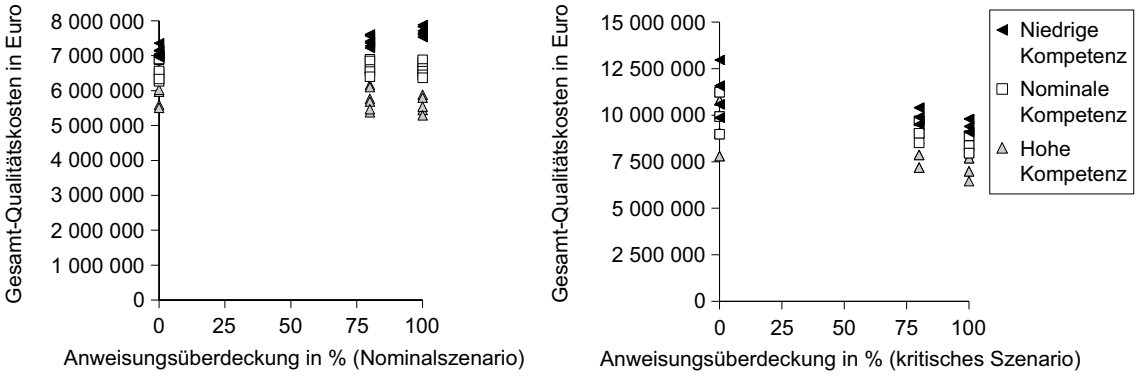

Abb. 74: Wirkung der Testparameter auf die Gesamt-Qualitätskosten im Nominalszenario (links) und im kritischen Szenario (rechts)

Abbildung 75 zeigt, wie die Projekt-Qualitätskosten (y-Achse) von den Kosten für den Systemtest (x-Achse) abhängen. Je intensiver der Systemtest abläuft, desto teurer wird das Projekt. Abhängig vom Szenario steigen die Projektkosten deutlich (Nominalszenario, Abbildung 75, links) oder um ein Vielfaches (Initialszenario, Abbildung 75, rechts).

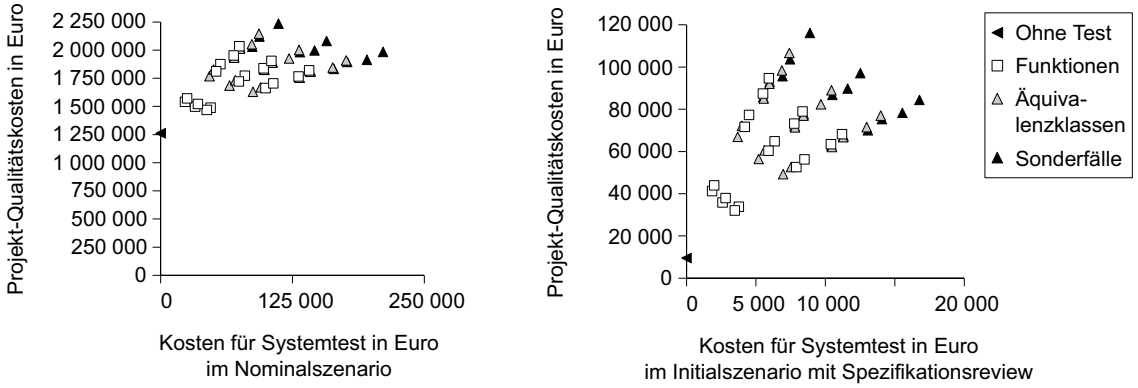

Abb. 75: Wirkung des Systemtests auf die Projekt-Qualitätskosten für Nominalszenario (links) und Initialszenario mit Spezifikationsreview (rechts)

Interaktionseffekte zwischen Spezifikationsreview und Systemtest

Den Interaktionseffekt zwischen Review und Test zeigt Abbildung 76 für den Fall mit nominaler Gutachter- und Testerkompetenz. Wie stark sich das Spezifikationsreview auswirkt, hängt auch von den gewählten Testtechniken des Systemtests ab. Durch die Kombination der Prüfungen wird der Nutzen bestimmt.

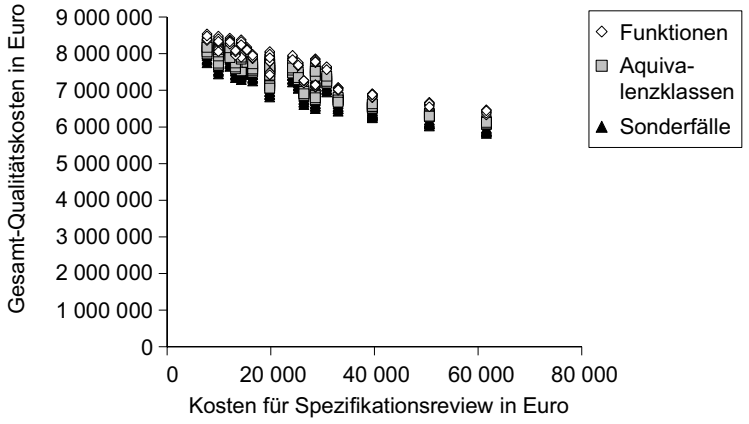

Abb. 76: Interaktionseffekte zwischen Review und Test

8.1.7 Sensitivitätsanalyse für unsichere Eingaben

In CoBe gibt es mehrere unsichere Eingaben: Die Fehlerfolgekosten müssen bei der Planung abgeschätzt werden. Sie bestimmen das Modellresultat qualitativ und quantitativ, weil sich Höhe und Richtung der Koeffizienten des Systemtests mit den Fehlerfolgekosten ändern (Abschnitt 8.1.5). Die Eingaben der Fehlerfolgekosten sind offensichtlich unsicher. Weitere unsichere Modelleingaben sind der Software-Umfang und die Kalibrierungsparameter. Sie sind zum Zeitpunkt der Projektplanung nicht bekannt. Der Umfang muss geschätzt werden. Die Kalibrierungsparameter müssen aus Archivdaten berechnet werden.

Vorgehen

Diese Unsicherheit modelliere ich für die Sensitivitätsanalyse als Zufallseffekt, indem die Parameter anhand einer bestimmten Verteilung variiert werden. Dann wird der Einfluss auf das Modellresultat analysiert (Saltelli et al., 2008). Die Unsicherheit der Eingaben orientiere ich an folgenden Erfahrungswerten: Bei der Planung führt die Unsicherheit für Umfang und Projektkosten zu einem Faktor von 0,5 bis 2,0 für diese Schätzwerte (Boehm, 1981). Die Resultate einer Function-Point-Zählung schwanken typisch um 20 % bis zu 50 %, je nachdem, wer zählt (Kemerer, 1993). Der Umfangsfaktor für den Code schwankt um den Faktor 2 (QSM, 2009). Jones (2007) berichtet eine ähnliche Streuung für Umfangsfaktoren der Dokumente. Darum werden zur Analyse die unsicheren Eingaben um einen Standardwert mit Faktoren zwischen 0,5 und 2 variiert.

Die Werte werden anhand der Standard-Normalverteilung (Mittelwert 0, Standard-Abweichung 1) aus diesem Bereich ausgewählt. Sie wird so angepasst, dass der Bereich zwischen 0,5 und 2 durch die sechsfache Standard-Abweichung abgedeckt wird. Die dreifache Standardabweichung nach unten ergibt folglich den Faktor 0,5 für die Eingabe; keine Abweichung ergibt den Standard-Eingabewert; die dreifache Standardabweichung nach oben ergibt den Faktor 2 für die Eingabe. Tabelle 82 zeigt die Eingabeparameter und den variierten Standardwert für das verwendete Nominalszenario. Die Umfangsfaktoren für Spezifikation und Entwurf werden gemeinsam variiert, damit die kombinatorische Vielfalt eingeschränkt wird. Sie wirken beide auf den Reviewaufwand.

Variierter Eingabeparameter	Standardwert
Umfang neuer Software (Function Points)	1000 Function Points
Umfangsfaktor Spezifikation, Entwurf	0,44 Seiten pro Function Point
Umfangsfaktor Code	53 Anweisungen pro Function Point
Aufwandsfaktor	1,0
Fehlerfaktor	1,0

Tabelle 82: Variierte und kombinierte Eingabeparameter

Die Analyse erfolgt mit dem Werkzeug SimLab (SimLab, 2009) und der Methode von Sobol (Saltelli et al., 2008). Es werden 12 288 Eingabekombinationen generiert und analysiert.

Analyseergebnisse

Tabelle 83 zeigt die Sensitivitätsindizes für den Gesamteffekt und den Haupteffekt. Je höher der Index, desto stärker wirkt sich der Parameter auf das Modellresultat aus; desto stärker wird also das Resultat durch die Unsicherheit beeinflusst. Der Haupteffekt entsteht durch den Parameter allein. Der Gesamteffekt enthält zusätzlich Interaktionseffekte durch die Kombination mit anderen Parametern.

Den stärksten Einfluss haben der Umfang und der Umfangsfaktor für den Code. Danach folgt für die Gesamt-Qualitätskosten der Fehlerfaktor, für die Projekt-Qualitätskosten der Aufwandsfaktor. Der Umfangsfaktor für Spezifikation und Entwurf spielt eine geringe Rolle. Die Haupteffekte ergeben in der Summe weniger als 1, das Modell enthält also Interaktionseffekte. Die Gesamteffekte sind insgesamt größer als 1, das Modell ist darum nicht-additiv (Saltelli et al., 2008).

Gesamt- und Projekt-Qualitätskosten streuen deutlich, wie die Häufigkeitsverteilungen in Abbildung 77 zeigen. Durch die unsicheren Eingaben werden die Resultate von CoBe also quantitativ deutlich verändert.

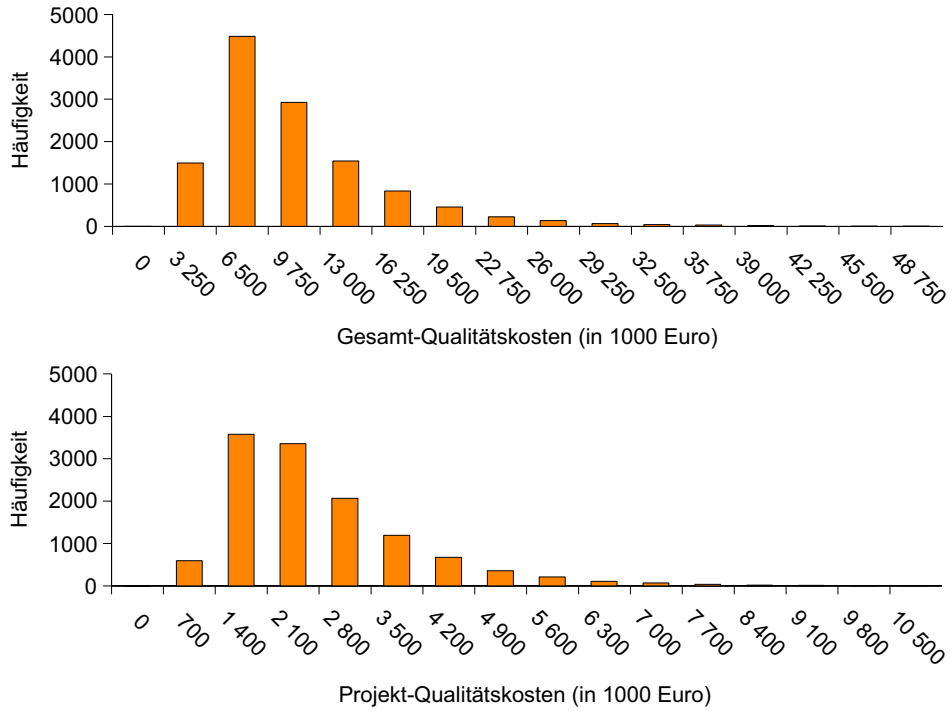

Abb. 77: Häufigkeit der Modellresultate für Gesamt-Qualitätskosten (oben) und Projekt-Qualitätskosten (unten)

Sensitivitätsindex für Parameter	Gesamt-Qualitätskosten		Projekt-Qualitätskosten	
	Haupt-effekt	Gesamt-effekt	Haupt-effekt	Gesamt-effekt
Umfang neuer Software	0,326	0,417	0,304	0,370
Umfangsfaktor Code	0,309	0,411	0,270	0,343
Fehlerfaktor	0,145	0,202	0,134	0,178
Aufwandsfaktor	0,083	0,119	0,182	0,232
Umfangsfaktor Spezifikation, Entwurf	0,000	0,000	0,000	0,001

Tabelle 83: Sensitivitätsindizes der Sobol-Analyse

8.1.8 Analyse der Auswirkungen unsicherer Eingaben

Diese Unsicherheit wirkt sich auf die Ziele des Modells unterschiedlich aus: Für das Ziel, die Kosten zu prognostizieren, muss das Modell auf jeden Fall kalibriert werden, da die Modellresultate quantitativ durch die unsicheren Eingaben bestimmt werden.

Für die Ziele, Kosten und Nutzen zu vergleichen und Qualitätskosten zu optimieren, untersuche ich, ob sich das Modellverhalten qualitativ verändert: Reagiert das Modell also so empfindlich auf die beiden wichtigen Eingabeparameter (Software-Umfang und Code-Umfangsfaktor), dass sich Aussagen ändern?

Dazu werden die Aussagen aus Abschnitt 8.1.5 untersucht:

- Das Spezifikationsreview wirkt sich auf die Gesamt-Qualitätskosten aus, ändert aber wenig an den Projekt-Qualitätskosten.
- Im Nominalszenario lohnt die Anweisungsüberdeckung im Systemtest kaum.
- Die Kompetenz der Gutachter und Tester spielt die wesentliche Rolle. Danach ist im Review die Prüflingsüberdeckung, im Systemtest die Abdeckung durch Black-Box-Testtechniken wichtig.

Für diese Analyse werden die Extremwerte und Standardwerte der beiden Eingaben (Umfang und Umfangsfaktor Code) kombiniert. Es entstehen 9 Kombinationen. Für jede Kombination werden wie in Abschnitt 8.1.5 die standardisierten Regressionskoeffizienten der Prüfparameter für Projekt- und Gesamt-Qualitätskosten berechnet. Schwankt deren Wert stark zwischen den Kombinationen oder ändert sich das Vorzeichen, dann verändert sich die Modellaussage qualitativ durch die Unsicherheit. Tabelle 84 zeigt die Bereiche der Koeffizienten. Sie zeigt für die oben genannten Modellaussagen, dass die Unsicherheit die Aussagen teilweise beeinflusst:

- Die Kompetenz spielt in allen Fällen eine wichtige Rolle. Weil die Koeffizienten in einem engen Bereich liegen und gleiche Vorzeichen haben, ändert sich diese Aussage nicht durch die Unsicherheit.
- Die Koeffizienten der Eingaben für das Spezifikationsreview bewegen sich für die Gesamt- und Projekt-Qualitätskosten in einem engen Bereich und ändern das Vorzeichen nicht. Die Aussagen über das Spezifikationsreview gelten also auch trotz den unsicheren Eingaben.
- Die Koeffizienten des Systemtests reagieren empfindlicher auf die Unsicherheit. Insbesondere wechselt das Vorzeichen für den Koeffizienten der Anweisungsüberdeckung. Diese Aussage ändert sich also durch unsichere Eingaben, sie ist darum unsicher.

Daraus folgt, dass das Modell nur mit Kalibrierung eingesetzt werden kann. Der Umfangsfaktor für den Code ist der wichtigste Kalibrierungsparameter; die Programmiersprache ist ein wichtiger Einflussfaktor.

8.1.9 Zusammenfassung der Sensitivitätsanalyse

Die Ergebnisse der Sensitivitätsanalyse bestätigen die Hypothesen:

Eingabe	Gesamt-Qualitätskosten		Projekt-Qualitätskosten	
	Minimum	Maximum	Minimum	Maximum
Gutachterzahl	-0,14	-0,12	-0,06	-0,03
Vorbereitungsrate	0,25	0,31	0,09	0,15
Gutachterkompetenz	-0,55	-0,44	-0,29	-0,23
Prüflingsüberdeckung	-0,37	-0,30	-0,17	-0,11
Abdeckung BBT	-0,28	-0,07	0,50	0,56
Anweisungsüberdeckung GBT	-0,15	0,31	0,50	0,66
Testerkompetenz	-0,71	-0,52	0,27	0,42
Testwiederholung	0,01	0,03	0,16	0,17

Tabelle 84: Schwankung der Koeffizienten durch Unsicherheit

Die Hypothese H 1 wird bestätigt: Die Eingaben der Prüfparameter wirken sich statistisch signifikant auf die Modellresultate aus. Die Entscheidungen über diese Prüfparameter bestimmen also die Projekt- und Gesamt-Qualitätskosten. Vor allem Kompetenz, Reviewvollständigkeit und Testtechniken prägen die Kosten.

Insbesondere die Effekte im Systemtest bestätigen die Hypothese H 2: Abhängig vom Szenario kann der Glass-Box-Test die Kosten senken oder steigern. Spezifikationsreviews senken die Gesamt-Qualitätskosten in jedem Szenario, wirken sich aber unterschiedlich auf die Projekt-Qualitätskosten aus.

Die unterschiedliche Wirkrichtung der Entscheidungen über den Systemtest, die Interaktionseffekte zwischen den Systemtesteingaben und den Szenarien und die Interaktionseffekte zwischen den Prüfungen bestätigten die Hypothese H 3: Die Qualitätskosten sind durch die Kombination der Entscheidungen und der Prozess- und Produktmerkmale bestimmt, so dass die Auswirkungen schwierig zu durchschauen sind.

Die Analyse zeigt, dass die Senkung der Projektkosten und der Gesamtkosten widersprüchliche Ziele sind (Hypothese H 4). Dies gilt vor allem für den Systemtest, weil er die Projekt-Qualitätskosten erhöht, aber die Gesamt-Qualitätskosten senkt. Das Spezifikationsreview lohnt sich nahezu immer, weil die Projekt-Qualitätskosten nur gering erhöht werden, die Gesamt-Qualitätskosten aber deutlich gesenkt werden.

Die Aussagen, die basierend auf CoBe getroffen werden können, hängen quantitativ und qualitativ von der Genauigkeit der Umfangsschätzung und der Kalibrierung ab. Die Hypothesen H 6 und H 7 werden durch die Sensitivitätsanalyse bestätigt. Die Kalibrierung mit Archivdaten ist notwendig. Zumindest muss überprüft werden, ob Modellresultate mit historische Daten ähnlicher Projekte übereinstimmen. Die Aussa-

gen hängen von den Angaben zu den Fehlerfolgekosten ab; deren Einschätzung ist ein wichtiger Teil des Modells (Hypothese H 5).

Diese Resultate bestätigen also die Folgerung der Prüfung mit studentischen Daten: Da die Modellparameter unsicher sind, ist sinnvoll, einen Bereich einzugeben, so dass die Unsicherheit als Bereich der Modellresultate deutlich wird.

8.2 Analyse des Optimums mit CoBe

Aus den Ergebnissen der Sensitivitätsanalyse lassen sich Folgerungen für optimale Prüfprozesse ziehen: Reviews sind fast immer lohnend, ein Glass-Box-Test ist nur bei hohen Fehlerfolgekosten sinnvoll.

Mit CoBe können solche Aussagen über besonders sinnvolle (oder besonders unsinnige) Prüfprozesse und Prüfparameter erkannt und dargestellt werden. In der folgenden Analyse werden dazu die Projekt-Qualitätskosten, die die Sicht des Herstellers spiegeln, und die Gesamt-Qualitätskosten, die für den Kunden entscheidend sind, betrachtet. Dies entspricht einer Situation, in der der Kunde auch die Projektkosten trägt, beispielsweise wenn Hersteller und Kunde der gleichen Organisation angehören oder nach Aufwand bezahlt wird.

Bei Auftragsprojekten mit Festpreis sind für den Kunden die Kosten nach Auslieferung relevant, also die Gesamt-Qualitätskosten abzüglich der Projekt-Qualitätskosten. Da die Sensitivitätsanalyse zeigt, dass die Projekt-Qualitätskosten wesentlich geringer als die Gesamt-Qualitätskosten sind, kann erwartet werden, dass sich die Aussagen der Optimierung für diese Betrachtungsweise wenig ändern.

8.2.1 Vorgehen

Für diese Analyse werden die Szenarien der Sensitivitätsanalyse verwendet. Zusätzlich werden zwei Varianten des Nominalszenarios gebildet:

- Mit Variante 1 wird untersucht, wie sich eine monetäre Gewichtung der Dauer auswirkt, weil beispielsweise Vertragsstrafen bei verspäteter Lieferung oder Umsatzverluste drohen. Dazu wird jeder Tag, der im Projekt in Prüfungen und Fehlerbehebung investiert wird, mit 0,5 % der Projektkosten gewichtet. Mit diesem Wert ergibt sich für das Nominalszenario in etwa ein Gleichgewicht zwischen Vertragsstrafen und Fehlerfolgekosten. Strafzahlungen in der Praxis sind höher und können pro Tag etwa 3 % des Projektpreises ausmachen. Diese Kosten fallen zusätzlich zu den Personalkosten von 100 Euro pro Entwicklerstunde an.
- Mit Variante 2 wird untersucht, wie sich die Testautomatisierung auswirkt. Dazu wird der Aufwandsanteil für die Wiederholung auf 10 % gesenkt.

Die Analyse erfolgt angelehnt an die Sensitivitätsanalyse. Dabei werden wieder die Eingaben von CoBe für den Prüfprozess und für die Prüfparameter variiert. Daraus werden mit CoBe die Gesamt- und Projekt-Qualitätskosten berechnet. Dann werden

die Qualitätskosten verglichen, um denjenigen Prüfprozess und diejenigen Prüfparameter zu finden, die zu minimalen Kosten führen.

Die kombinatorische Vielfalt, die durch die vielen Eingabeparameter von CoBe entsteht, führt zu einer komplexen Analyse, weil sehr viele Kombinationen und ihre Resultate betrachtet werden müssen. Darum grenze ich die Vielfalt der Eingaben ein und variiere nur die Folgenden:

- Da die Sensitivitätsanalyse die große Bedeutung der Reviews und ihrer Vollständigkeit zeigt, wird die Vollständigkeit des Spezifikationsreviews, des Entwurfsreviews und des Codereviews variiert. Weil Personal knapp ist, werden Reviews mit 2 oder 5 Gutachtern durchgerechnet.
- Für den Systemtest wird die Intensität in einzelnen Schritten gesteigert. Sie reicht von keinem Systemtest über die aufeinander aufbauenden Testtechniken des Black-Box-Tests (Abdeckung von Funktionen, Äquivalenzklassen, Sonderfällen), bis zur vollständigen Überdeckung von Anweisungen, Zweigen, Termen und Schleifen im Glass-Box-Test.

Konstant sind folgende Eingaben: Ein Feldtest, also eine Erprobung beim Kunden, findet auf jeden Fall statt, weil von einem Auftragsprojekt ausgegangen wird. Ich gehe davon aus, dass Entwickler in solchen Projekten einen Modultest durchführen. Auch ein Integrationstest findet statt, wenn die Integration stattfindet und ausprobiert wird, ob das integrierte Programm läuft. Außerdem gehe ich davon aus, dass in den Tests nach der korrektiven Wartung die gleichen Testparameter wie im Projekt gefordert werden: Wird also im Projekt für den Systemtest 80% Anweisungsüberdeckung verlangt, wird dies auch für die Korrektur verlangt. Mit diesen Eingabewerten werden die Qualitätskosten für 1944 Kombinationen je Szenario berechnet.

8.2.2 Resultate

Die Resultate aller Szenarien, bei denen die Dauer nicht gewichtet wird, sind ähnlich:

Projekt-Qualitätskosten

Frühe Reviews lohnen sich bereits für den Hersteller, weil dadurch die Projekt-Qualitätskosten minimal gehalten werden. In allen Szenarien sind intensive Reviews, also mit 5 Gutachern und vollständiger Prüfung der Spezifikation und des Entwurfs, optimal; nur bei Wiederverwendung reichen 2 Gutachter für den optimalen Fall. Der Systemtest verteuert die Projekt-Qualitätskosten; das ist konsistent mit den Resultaten der Sensitivitätsanalyse.

Abbildung 78 zeigt dies für das Nominalszenario. Die Projekt-Qualitätskosten sind an der x-Achse, die Gesamt-Qualitätskosten an der y-Achse abgetragen. Die Datenpunkte sind für einen Prüfprozess ohne Reviews in schwarz dargestellt, für einen Prüfprozess mit frühen Reviews ohne Codereviews in hellgrau, für einen Prüfprozess mit allen Reviews in dunkelgrau und für andere Prüfprozesse in weiß dargestellt. Die Datenpunkte ganz links stehen also für minimale Projektkosten; sie gehören zu Prüf-

prozessen mit ganz unterschiedlichen Entscheidungen über Reviews. So befinden sich Vorgehen ganz ohne Reviews, mit frühen Reviews oder mit allen Reviews links im Diagramm mit nahezu gleichen Projekt-Qualitätskosten. Diese Datenpunkte unterscheiden sich aber erheblich in den Gesamt-Qualitätskosten. Daraus folgt, dass Entscheidungen über Reviews die Projekt-Qualitätskosten kaum beeinflussen, aber deutlich die Gesamt-Qualitätskosten prägen.

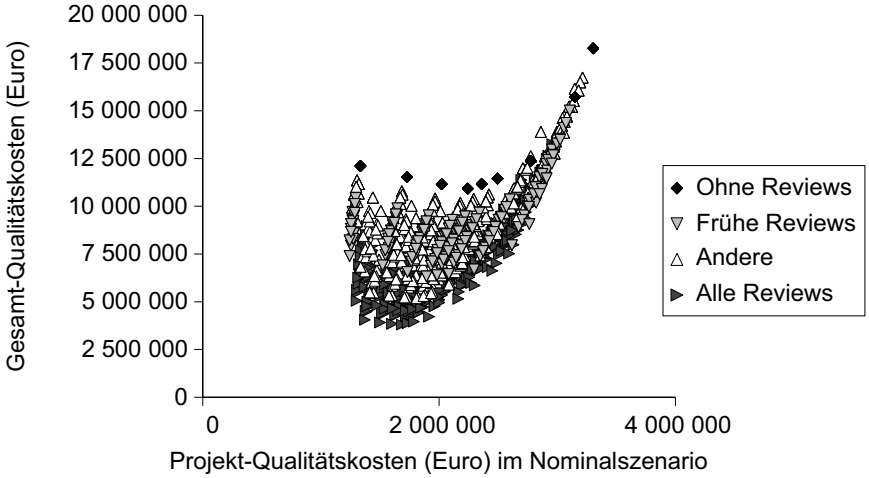

Abb. 78: Projekt- gegen Gesamt-Qualitätskosten im Nominalszenario

Gesamt-Qualitätskosten

Für die Gesamt-Qualitätskosten sind alle Reviews wichtig, da nur mit intensiven Reviews für Spezifikation, Entwurf und Code minimale Gesamt-Qualitätskosten erreicht werden können. Es ist also im Interesse des Kunden, dass diese Reviews durchgeführt werden. Dies zeigt Abbildung 78 für das Nominalszenario: Die minimalen Gesamt-Qualitätskosten sind die niedrigsten Punkte im Diagramm. Sie gehören zu den Prüfprozessen mit allen Reviews.

Der optimale Fall der Systemtest-Prüfparameter hängt von dem Szenario, vor allem von den Fehlerfolgekosten ab:

- Bei geringen Fehlerfolgekosten (Initialszenario) ist ein Systemtest ausreichend, mit dem alle Funktionen getestet werden. Dieses Vorgehen ist typisch für die Praxis.
- In Szenarien mit mittleren Fehlerfolgekosten werden minimale Gesamt-Qualitätskosten mit einem nominalen Systemtest erreicht. Im nominalen Test werden Testfälle für alle Funktionen und für zusätzliche Äquivalenzklassen erstellt.
- Wenn in Szenarien mit mittleren Fehlerfolgekosten die Testwiederholung automatisiert wird, dann können Tests günstig wiederholt werden (Variante 2 des Nominalszenarios). In dieser Situation lohnt sich ein zusätzlicher Glass-Box-Test, in dem Zweigüberdeckung angestrebt wird. Durch den niedrigeren Wiederholungs-

aufwand, der sich vor allem in der Wartung zeigt, können somit um fast 20 % niedrigere Gesamt-Qualitätskosten als im Nominalszenario erreicht werden.

- Im kritischen Szenario mit extrem hohen Fehlerfolgekosten führt ein Glass-Box-Test mit allen Überdeckungskriterien zu minimalen Gesamt-Qualitätskosten.

Abbildung 79 zeigt Projekt- und Gesamt-Qualitätskosten des kritischen Szenarios, Abbildung 80 des Szenarios mit Testautomatisierung. Die Diagramme sind gleich wie Abbildung 78 aufgebaut. Da intensivere Systemtests die Projektkosten verteuern, liegen die Datenpunkte mit hohem Systemtest-Aufwand im rechten Diagrammbereich. Im Nominalszenario (Abbildung 78) steigen die Gesamt-Qualitätskosten mit den Projekt-Qualitätskosten; sie steigen also mit intensiverem Systemtest. Im kritischen Szenario (Abbildung 79) sinken die Gesamt-Qualitätskosten mit steigenden Projekt-Qualitätskosten, d.h. mit steigender Systemtest-Intensität. In Abbildung 80 liegt das Optimum bei mittleren Projekt-Qualitätskosten, also bei einer mittleren Systemtest-Intensität.

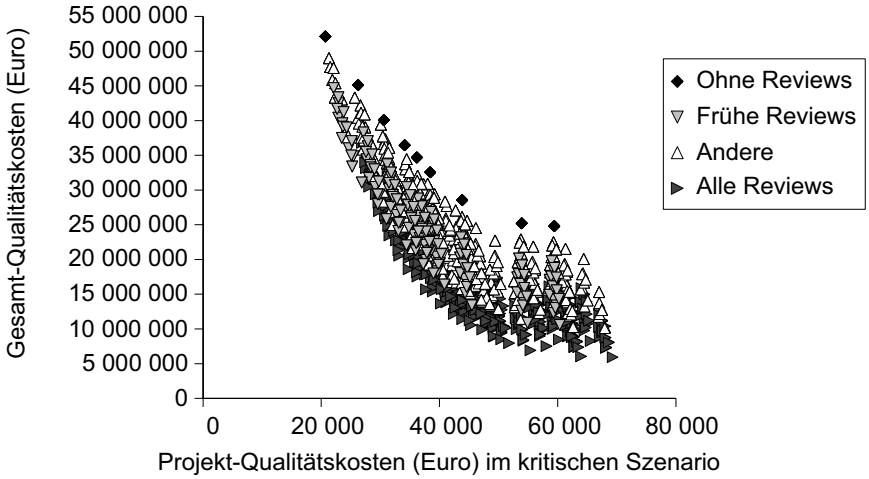

Abb. 79: Projekt- gegen Gesamt-Qualitätskosten im kritischen Szenario

Einfluss der Dauer

Kostet eine spätere Auslieferung Geld, dann lohnt sich im Nominalszenario für den Hersteller tatsächlich nicht, Prüfungen durchzuführen, weil die niedrigsten Projekt-Qualitätskosten ohne Prüfungen erreicht werden. Die geringsten Gesamt-Qualitätskosten werden mit vollständigen Spezifikations- und Entwurfsreviews erreicht. Das Spezifikationsreview führt zur geringsten Steigerung der Projekt-Qualitätskosten. Abbildung 81 zeigt dies: Investitionen in Projekt-Qualitätskosten führen zuerst zu geringeren Gesamt-Qualitätskosten. Dann aber steigen die Gesamt-Qualitätskosten mit den Projekt-Qualitätskosten, weil die Kosten für die Dauer höher als die eingesparten Fehlerkosten werden. Es gibt also Situationen mit hohem Termindruck, in denen es sinnvoll ist, weniger intensiv zu prüfen. Da in diesem Szenario jeder Tag

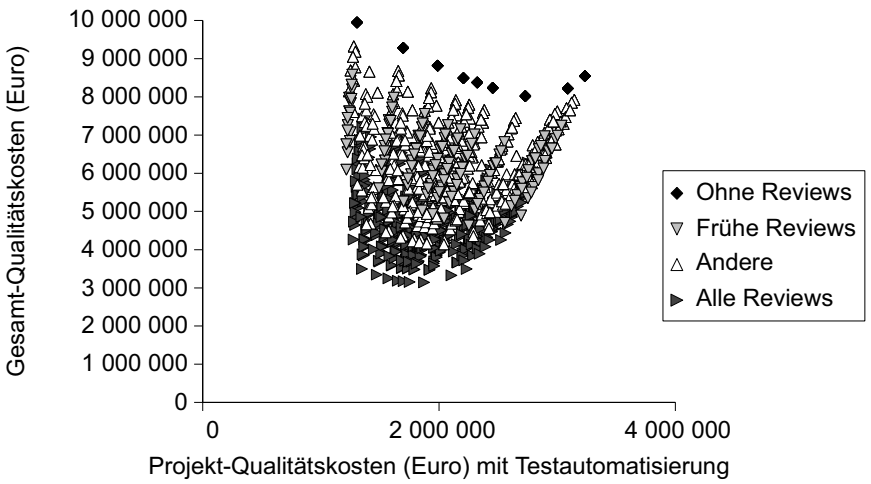

Abb. 80: Projekt- gegen Gesamt-Qualitätskosten mit Testautomatisierung

Dauer mit Geld bewertet wird, entspricht dies einer Situation, in der jeder Tag auf dem Markt zählt. In Situationen mit Vertragsstrafen dürfen nur Terminüberschreitungen gewichtet werden; die Dauer darf also erst ab einem gewissen Schwellwert bewertet werden. Unter dem Schwellwert entsprechen die Modellresultate dem Nominalszenario, über dem Schwellwert steigen die Kosten an und entsprechen dem Szenario mit gewichteter Dauer.

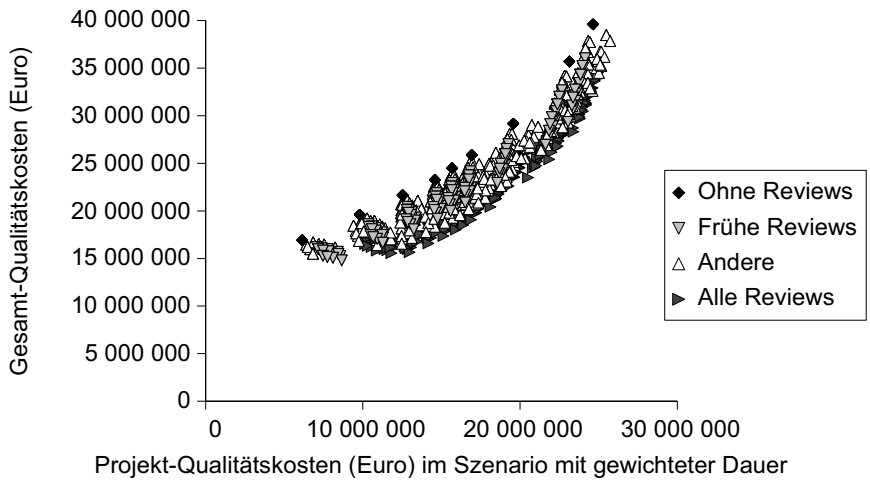

Abb. 81: Projekt- gegen Gesamt-Qualitätskosten mit gewichteter Dauer

Vergleich der Resultate

In allen Fällen widerspricht das Ziel, die Projektkosten zu minimieren, dem Ziel, die Gesamtkosten zu minimieren. Hersteller und Kunde verfolgen also widersprüchliche Ziele. In allen Fällen führt ein Prüfprozess ganz ohne Reviews zu den teuersten

Gesamt-Qualitätskosten (Abbildungen 78, 79, 80 und 81). Die Resultate für diese vier Szenarien, dem Nominalszenario, mit Testautomatisierung, mit gewichteter Dauer und dem kritischen Szenario, zeigt Abbildung 82 im Vergleich.

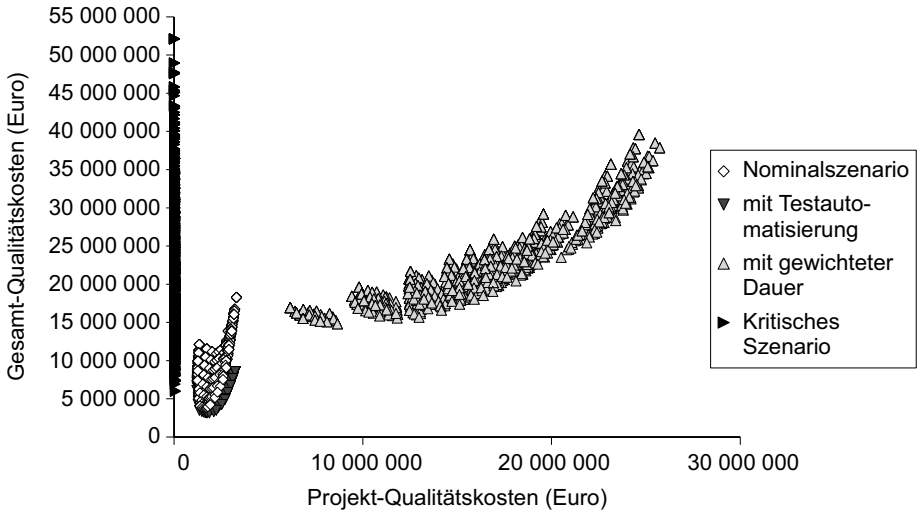

Abb. 82: Kosten im Vergleich

Da das Produkt im kritischen Szenario einen kleineren Umfang als das Produkt der anderen Szenarien hat (100 Function Points statt 1000 Function Points Umfang), sind die Projekt-Qualitätskosten entsprechend gering. Die Datenpunkte werden darum ganz links im Diagramm aufgetragen. Durch die hohen Fehlerfolgekosten können die Gesamt-Qualitätskosten aber sehr hoch werden. Die Nominalszenarien mit und ohne Testautomatisierung (graue beziehungsweise weiße Datenpunkte) überdecken sich nahezu. Die Datenpunkte liegen im linken Viertel des Diagramms. Mit Testautomatisierung sind die Kosten etwas geringer als ohne. Durch die Gewichtung der Dauer steigen die Projekt-Qualitätskosten deutlich (graue Datenpunkte).

8.3 Vorgehen für die Validierung in der Industrie

Mit CoBe sollen Aussagen über Industrieprojekte getroffen werden. Darum ist CoBe mit Industrieprojekten validiert. Dafür konnten zwei externe Partner aus unterschiedlichen Organisationen gewonnen werden. In jeder Organisation wurde ein Projekt analysiert. Projektdaten wurden erhoben und für die Validierung des Modells verwendet. Da die Projekte komplex sind, erfolgte die Validierung in einzelnen Schritten:

1. **Vorgespräche:** In diesen Gesprächen wurden Modell und Projekt grob vorgestellt und umrissen. Das weitere Vorgehen wurde geklärt.

2. **Befragung und Datenerhebung:** Zuerst wurden Informationen über den Prozess, dann über verfügbare Metriken und über Erfahrungswerte gesammelt. Diese Gespräche erfolgten als Interview, die Befragten erhielten einen Fragebogen im Voraus, der als Gesprächsleitfaden diente.
3. **Analyse:** Die Informationen über das Projekt wurden gesichtet, sortiert, zusammengefasst und den Modellparametern zugeordnet. Dabei entstand eine Prozessbeschreibung mit Metriken und Istwerten.
4. **Modellparameter:** Aus dieser Prozessbeschreibung wurden die Modelleingaben und die Istwerte für die Validierung entnommen. Dabei wurden fehlende Informationen identifiziert. Fehlen Informationen, die für Modelleingaben benötigt werden, dann sind diese Eingaben unsicher. In solchen Fällen wurden Modellvarianten gebildet, in dem verschiedene Werte für die unsicheren Eingaben verwendet werden. Die Modellvarianten ergeben einen Bereich der Modellresultate und machen dadurch die Unsicherheit sichtbar.
5. **Kalibrierung:** Zuerst wurden die Modellresultate betrachtet, die zur Kalibrierung benötigt werden: Gesamtaufwand, Gesamtdauer, Gesamtfehlerzahl. Das Modell wurde anhand dieser Daten kalibriert. Wurden nicht alle Fehler gezählt, dann erfolgte die Kalibrierung mit der Summe der verfügbaren Fehlerzahlen.
6. **Validierung:** Die Modellresultate wurden mit den Istwerten verglichen. Abweichungen und Übereinstimmungen wurden identifiziert. Abweichungen wurden diskutiert und, wenn möglich, erklärt.
7. **Referenzmodell:** Für konkrete Fragestellungen wurde ein Referenzmodell erstellt. Es liefert exakte Resultate, während die Modellvarianten einen Bereich liefern. Dieses Modell enthält weitere Anpassungen, die sich aus den vorherigen Schritten ergeben, um das Projekt möglichst genau zu beschreiben.
8. **Diskussion und Prüfung:** Die Resultate wurden gemeinsam mit den Befragten diskutiert und auf Plausibilität geprüft.
9. **Nacharbeit:** Falls Missverständnisse erkannt wurden oder weitere Informationen verfügbar wurden, wurde das Modell überarbeitet.

In den Vorgesprächen hat sich gezeigt, dass das Modell erweitert werden muss; es entstand die Version 2 von CoBe (Abschnitt 8.1). Dazu wurden alle Tests detailliert mit Prüfparametern modelliert, die Codeanalyse und der Korrekturprüfprozess wurden ergänzt. Die Prozessanalyse zeigte also auf, wo das Modell unvollständig ist. Dies stützt das iterative Vorgehen, das dem Modelleinsatz zu Grunde liegt (Abschnitt 3.6).

Bei einem iterativen Vorgehen besteht die Gefahr, dass das Modell so lange angepasst wird, bis es die Realität ausreichend genau beschreibt. Dann ist die Validierung ungültig, weil ein Vergleich der Modellresultate mit Istwerten nicht mehr sinnvoll ist. Bestimmte Modelländerungen stellen aber keine Bedrohung der Validierung dar:

- Das Modell kann für die Validierung kalibriert werden, weil dies auch für den Modelleinsatz vorgesehen ist.
- Erweiterungen, die unabhängig von den Istwerten der Validierung modelliert und quantifiziert werden, können mit den Istwerten validiert werden.
- Bleiben bestehende Modellteile unverändert, dann ändern Erweiterungen nichts an Aussagen zur Validität der unveränderten Teile. Unveränderte Modellteile können von den Erweiterungen unabhängig validiert werden.

Daraus folgt, dass die Erweiterungen für die Industrieprojekte die Validierung nicht bedrohen, weil sie nichts an bestehenden Zusammenhängen ändern. Stattdessen werden bestehende Zusammenhänge ergänzt. Sie bedrohen die Validierung nicht, weil sie auf Daten basieren, die nicht aus den Industrieprojekten stammen. Für die Tests werden Zusammenhänge wiederverwendet, aber neu quantifiziert. Dadurch werden bestehende Zusammenhänge sogar gestützt. Anpassungen sind im Referenzmodell möglich, um das Projekt möglichst genau zu beschreiben. Eine Validierung mit Referenzmodell ist dann aber nicht aussagekräftig.

Als Validierungskriterium wird die logarithmische Abweichung in deziBel (Abschnitt 7.2.3) mit 2 dB als Grenzwert verwendet (Tabelle 85); 3 dB entsprechen einem Faktor 2:

$$LE = 10 \cdot \left| \log_{10}\left(\frac{Modellresultat}{Istwert}\right) \right|$$

Kriterium	Bewertung	Folgerung
$0\text{dB} \leq LE \leq 1\text{dB}$	Modell valide	
$1\text{dB} < LE \leq 2\text{dB}$	Modell valide	Ursachenanalyse mit unsicheren Eingaben und unklaren Prozess
$2\text{dB} < LE$	Validität fraglich, Modell nicht valide	

Tabelle 85: Kriterien für die Validierung

8.4 Industrieprojekt 1

In diesem Abschnitt wird die Validierung mit dem ersten Industrieprojekts beschrieben. Dazu werden zuerst die Analyse des Projekts (Abschnitt 8.4.1), die Abbildung auf das Modell (Abschnitte 8.4.2 und 8.4.3), die Kalibrierung (Abschnitt 8.4.4) und dann die Ergebnisse (Abschnitte 8.4.5 und 8.4.6) gezeigt.

Im Projekt wurde Firmware für ein Rechnersystem erstellt, das für den Markt entwickelt wurde. Ein wesentliches Merkmal ist die hohe Verfügbarkeit des Systems. Für das Rechnersystem wird regelmäßig, etwa im Abstand von zwei Jahren, ein neues Rechnermodell angeboten. Im Projekt wurde die bereits bestehende Firmware des vorherigen Rechnermodells erweitert, teilweise wiederverwendet und teilweise

ersetzt. Die Entwicklung ist in Subsysteme aufgeteilt, jedes Subsystem wird in einem Subsystem-Projekt entwickelt.

Für die Validierung werden das Gesamtprojekt und drei Subsystem-Projekte betrachtet. Entwickler der Subsysteme, Produktmanager und QS-Verantwortliche wurden befragt. Es wurden qualitative Merkmale, aber auch quantitative Daten erfragt. Zusätzlich standen gemessene Werte zur Verfügung; Tabelle 86 zeigt die wesentlichen Daten. Insgesamt waren rund 400 Mitarbeiter in den zwei Jahren mit dem Projekt beschäftigt. Der Umfang des Codes beträgt etwa 11 Millionen Anweisungen, zum großen Teil in C++; etwa 25 % wurden neu erstellt oder geändert.

Verfügbare Istwerte
Korrekturaufwand pro Fehler nach den unterschiedlichen Prüfungen mit Minimal- und Maximalwerten
Reviewprozess mit Gutachterzahl, Vorbereitungsintensität, Sitzungsdauer
Zahl entdeckter Fehler in Reviews, typisch pro Sitzung
Testmethoden und -parameter, Automatisierung, Wiederholung
Dauer und Mitarbeiterzahl des Projekts und der Subsystem-Projekte
Zahl entdeckter Fehler in der Entwicklungsumgebung, im Systemtest und durch Vorablieferung
Umfang des Codes (Hinzugefügt, geändert, wiederverwendet)

Tabelle 86: Istwerte des Industrieprojekts 1

8.4.1 Das Projekt und sein Prozess

Da im Projekt System-Software parallel entwickelt wurde, werden zuerst diese speziellen Merkmale und ihre Abbildung in CoBe dargestellt.

Einbettung in die Projektumgebung und Anforderungen

Die Anforderungen für das gesamte System wurden zentral festgelegt und dann für die Hardware und die Software verfeinert und aufgeteilt. Software bezeichnet in diesem Kontext die Firmware. Die Software-Anforderungen wurden auf einzelne Subsysteme verteilt und beschreiben grob die Funktion. Die Anforderungen waren weitgehend stabil. Im betrachteten Ausschnitt des Projekts sind diese Anforderungen fest vorgegeben. Abbildung 83 zeigt den Zusammenhang zwischen Software-Anforderungen, der Aufteilung auf Subsystem-Projekte und die Integration der Software zum System. Um Subsystem-Projekte vom gesamten Projekt abzugrenzen, wird das gesamte Projekt im Folgenden auch als Gesamtprojekt bezeichnet

Detailliertere Anforderungen für die Software waren im Wesentlichen durch die Schnittstellen zur Hardware und zu anderen Software-Komponenten definiert. Anforderungen und Entwurf wurden für Erweiterungen und für ersetzte

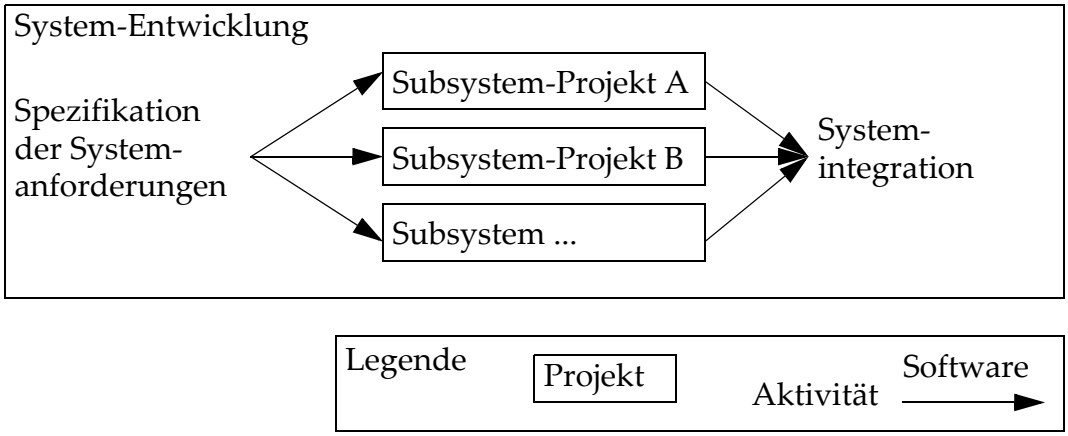

Abb. 83: Aufteilung in Subsystem-Projekte

Komponenten mit UML-Diagrammen und natürlichsprachlich in einem Dokument, dem Entwurf, beschrieben; Auforderungen und Entwurf ließen sich kaum voneinander trennen, da die Anforderungen zum einen durch die Systemanforderungen und zum anderen durch die Schnittstellen des Subsystems und des vorhandenen Codes definiert waren und damit weitgehend vorgegeben waren. Im betrachteten Ausschnitt des Projekts wurden C++ und eine firmenspezifische höhere Programmiersprache verwendet.

Ablauf und Organisation der Entwicklung

Die Entwicklung erfolgte iterativ und parallel, weil jedes Subsystem parallel zu den anderen Subsystemen weiterentwickelt und gepflegt wurde. Ein Entwickler bearbeitete einen Teil, eine Komponente. Dazu wurden die Anforderungen und der Entwurf im Entwurf dokumentiert, der Code wurde implementiert und dann in das Subsystem integriert. Diese Entwicklung erfolgte in einer Entwicklungsumgebung, die das neue Rechnersystem simulierte. Die Systemintegration, bei der die Subsysteme integriert werden, erfolgte alle zwei Wochen. Das Resultat wird als Treiber bezeichnet. Er wurde abwechselnd in der Entwicklungsumgebung eingesetzt oder an unabhängige Tester ausgeliefert.

Prüfungen und Prüfprozesse

Die folgenden Prüfungen wurden durchgeführt:

- Entwurfsreview
- Codeanalyse
- Modultest
- Subsystem-Integrationstest
- Codereview

- Systemintegrationstest
- Systemtest (unabhängig)
- Interne und externe Vorablieferung gegen Projektende

Entwurfsreview, Codeanalyse, Modultest, Subsystem-Integrationstest und Codereview sind der Subsystem-Entwicklung zugeordnet und wurden unterschiedlich intensiv für die verschiedenen Subsysteme durchgeführt. Das Codereview fand zeitnah nach der Implementierung statt, aber erst, nachdem der Entwickler den Modultest durchgeführt hat. Da die Subsysteme kontinuierlich integriert wurden und der Subsystem-Integrationstest in diesen Schritt eingebunden war, erfolgten die Codereviews nach dem Subsystem-Integrationstest. Die Subsysteme wurden zum System integriert und dann durch Systemintegrationstest, Systemtest und die Vorablieferungen geprüft. Systemintegrationstest und Systemtest erfolgten durch unabhängige Testabteilungen. Die Übergabe an diese Testabteilungen erfolgte regelmäßig ab dem Zeitpunkt eines ersten lauffähigen Systems. Der Systemintegrationstest und der Systemtest erfolgten also zeitversetzt, aber parallel zur Entwicklung. Das System wurde während der Entwicklung als Entwicklungsumgebung eingesetzt. Dadurch wird es implizit bei der Subsystem-Entwicklung und -Prüfung mitgeprüft.

Merkmale einzelner Prüfungen

In den Teilprojekten wurden die Prüfungen unterschiedlich intensiv durchgeführt. Die Ergebnisse der Befragung sind in Tabelle 87 zusammengefasst.

Das integrierte System wurde mit dem Systemintegrationstest geprüft, damit ein funktionierendes System an die Entwicklungsumgebung oder an den unabhängigen Systemtest geliefert wird; es wurde geprüft, ob das System reif für den Test ist. Der Systemtest erfolgte durch unabhängige Tester und organisatorisch getrennt. Testfälle wurden in Testpaketen von den unabhängigen Testern aus den Anforderungen abgeleitet. Dabei wurden auch Testfälle früherer Rechnermodelle wiederverwendet und angepasst. Der Testplan, der die Testfälle beschreibt, wurde von Entwicklern begutachtet. Dabei konnten Mängel des Testplans identifiziert und korrigiert werden. Das System wurde, bevor es für den Markt verfügbar wurde, in zwei Vorablieferungen im produktiven Einsatz erprobt, zuerst intern im Konzern, dann extern bei ausgewählten Kunden.

8.4.2 Die Abbildung in das Modell

Zuerst wird die Abbildung des Prozesses in CoBe gezeigt. Die detaillierten Eingaben folgen im Abschnitt 8.4.3, die Resultate in den Abschnitten 8.4.5 und 8.4.6.

Für die Validierung betrachte ich das gesamte Projekt und drei Subsystem-Projekte. Die Subsysteme werden einzeln betrachtet, weil sich der Prüfprozess der Subsystem-Entwicklungen unterscheidet. Darum werden vier Modellinstanzen von CoBe erstellt: Ein Gesamtmodell und drei Subsystem-Modelle. Ich bilde die Projekte auf die vier Modelle wie folgt ab:

Prüfung	Subsystem A	Subsystem B	Subsystem C
Entwurfsreview	Gutachter waren Tester und Entwickler des gleichen oder anderer Subsysteme mit betroffenen Schnittstellen. Die Vorbereitung erfolgte gründlich. Neue Software wurde vollständig geprüft. Eine Sitzung dauerte höchstens zwei Stunden.		
	5 - 6 Teilnehmer mit Autor	Bis zu 10, maximal 15 Teilnehmer insgesamt	Maximal 10 Teilnehmer insgesamt
Codeanalyse	BEAM (Brand, 2000) und Lint (Johnson, 1978) wurden eingesetzt. Durch Konfiguration der Werkzeuge wurden wenig falsche Befunde entdeckt.		
Modultest	Automatisierte Tests überdeckten im Mittel 70 %, bis zu 85 % der Anweisungen.	Testfälle wurden manuell durchgeführt, weil Hard- und Software eng gekoppelt sind.	Zum Teil automatisierte, zum Teil manuelle Testdurchführung.
Subsystem-Int.-test	Das Subsystem wurde von den Entwicklern kontinuierlich integriert. Dann erfolgte der Test dieser Version in der Entwicklungsumgebung. Diese besteht aus der letzten Version des Systems.		
	Nächtliche Wiederholung des Modultests.		Wiederholung des Modultests und weitere Testfälle
Codereview	Rund 25 % des neuen Codes wurden von 3 bis 5 Gutachtern geprüft[a]. Die Gutachter sind betroffende Entwickler. Pro Sitzung wurden 500 bis 600 Zeilen gründlich vorbereitet, pro Sitzung wurden bis 200 Zeilen detailliert besprochen.	Neuer Code wurde von 3 bis 5 Gutachtern geprüft, die sich intensiv vorbereiteten.	Kritischer Code wurde von 5 bis 6 Gutachtern geprüft. Die Vorbereitung war intensiv mit einer Vorgabe von 100 Zeilen pro Stunde. Die Sitzung dauerte maximal 1,5 Stunden.

Tabelle 87: Subsystem-Prüfungen und ihre Unterschiede (Projekt)

a. Dazu wurde kritischer Code ausgewählt.

Der parallele und iterative Prozess kann direkt auf den sequentiellen Prozess von CoBe abgebildet werden, weil die Entwicklung einer einzelnen Komponente innerhalb eines Subsystems dem gleichen Prozess folgte. Ich nehme also an, dass die Entwicklung im Wesentlichen additiv erfolgte und dass die wesentlichen Effekte additiv sind. Das bedeutet, dass die einzelnen Komponenten aufeinander aufbauen und sequentiell erstellt werden können; eine fertige, geprüfte Komponente wird nicht oder nur unwesentlich verändert. Abbildung 84 skizziert diese Zuordnung beispielhaft für ein System, das aus zwei Komponenten besteht, und für den Modultest, die Subsystemintegration und die Systemintegration. Die Meilensteine M1 und M2 für den Abschluss des Modultests der beiden Komponenten fallen in CoBe auf den Mei-

lenstein M, die Meilensteine S1 und S2 auf den Meilenstein S. Da in Cobe eine Prüfung durch Verwendung als Entwicklungsumgebung nicht modelliert ist und die kontinuierliche Integration nicht explizit dargestellt wird, ordne ich diese beiden Prüfungen dem Subsystem-Integrationstest in CoBe zu.

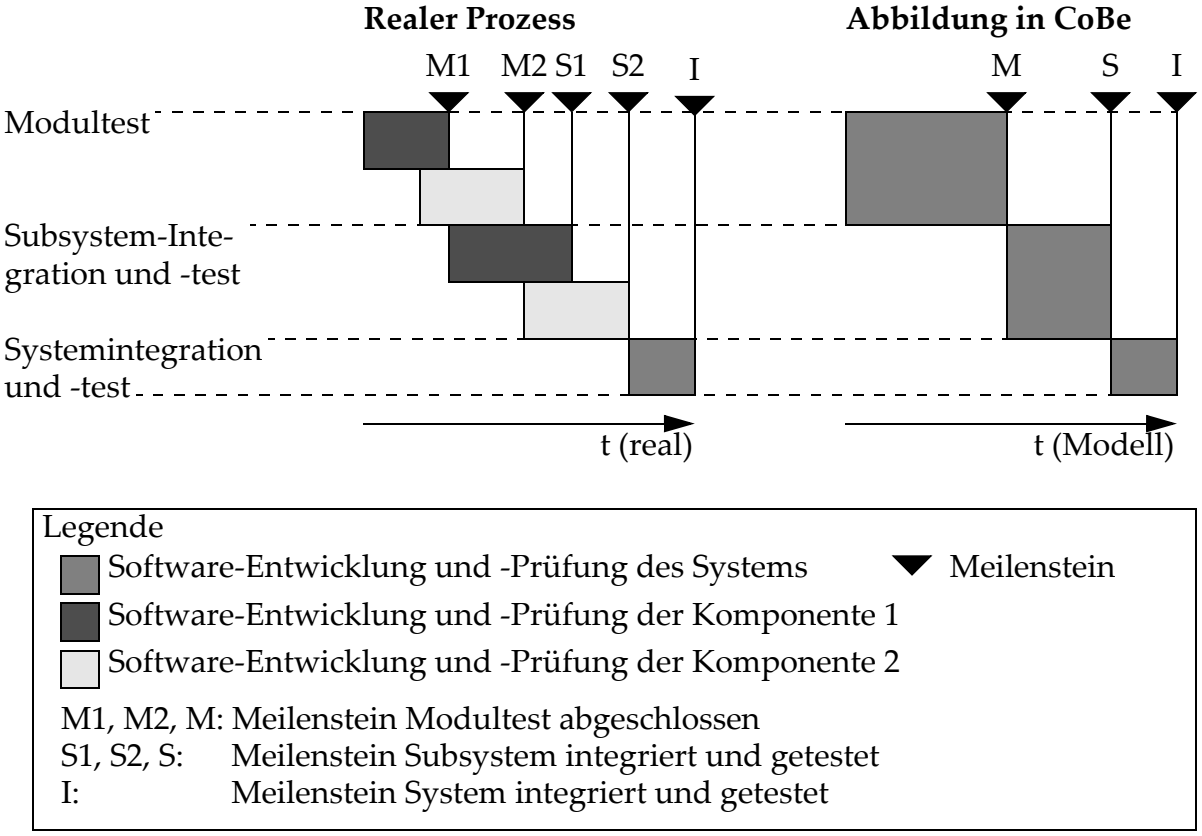

Abb. 84: Realer Prozess und modellierter Prozess

Durch diese Modellierung muss bei der Interpretation und Bewertung der Resultate mit Unschärfen gerechnet werden: Fehlerentdeckung durch Fernwirkungen in anderen Komponenten wird im Modell nicht dargestellt, weil CoBe keine Komponenten kennt. Wirkungen, die dadurch entstehen, dass bereits fertige und geprüfte Komponenten erneut mit neuen Komponenten mitgeprüft werden, sind nicht im Modell enthalten. Es ist unklar, wie parallele Prozesse in COCOMO II abgebildet werden und wie sich diese Art der Organisation dann auf den Zusammenhang zwischen Umfang, Aufwand, Dauer und Personalbedarf auswirkt.

In den Subsystem-Projekten wird zwischen spezifizierten, detaillierten Software-Anforderungen und Entwurf nicht hart getrennt; beides wurde im Entwurf dokumentiert und gemeinsam geprüft. Die Trennung ist für Firmware nicht möglich. Da die groben Anforderungen an die Komponente und die externen Schnittstellen fest vorgegeben waren, werden im Modell keine Spezifikationsfehler abgebildet. In den

erhobenen Fehlerzahlen erfolgt keine Trennung zwischen Entwurfs- und Codefehlern, diese werden aber in CoBe dargestellt. Im Projekt ist die Fehlerschwere über den Schaden definiert. Für den Schaden werden kritische Fehler, die zum Stillstand des Systems führen, und Hauptfehler, die den Betrieb behindern, erfasst. Darum werden in CoBe nur diese schweren Fehler dargestellt.

8.4.3 Modelleingaben im Detail

Die Modelleingaben ergeben sich aus der Analyse des Projekts. Sie werden wie folgt gesetzt: Für die Validierung wird das gesamte Projekt betrachtet, weil dafür die meisten Istwerte verfügbar sind.

Der Code umfasst etwa 10 Millionen Anweisungen, davon ist rund ein Viertel neu. Als Umfangsfaktoren verwende ich 55 Anweisungen pro Function Point (C++, Boehm, 2000) und 0,88 Seiten pro Function Point für den Entwurf, also den doppelten Umfang wie ursprünglich quantifiziert, weil Anforderungen und Entwurf gemeinsam beschrieben werden.

Die Quantifizierung der Fehlerdichte wurde von Auftragsprojekten auf Projekte für Systemsoftware umgestellt (Jones, 1996 und 2007), ebenso die Verteilung auf die Fehlerarten (ohne Spezifikationsfehler) und auf die Fehlerschwere (ohne Nebenfehler). Die Parameter von COCOMO II wurden anhand der Befragungsergebnisse angepasst.

Da sich der Ablauf der Prüfungen in den Subsystem-Projekten unterscheidet, müssen für dieses Gesamtmodell mittlere Prüfparameter ausgewählt werden, die diese Unterschiede in etwa ausgleichen. Tabelle 88 zeigt die Eingaben des Gesamtmodells. Die Tests werden wiederholt; für den Modultest übernimmt dies der Subsystemintegrationstest.

Für einige Eingaben sind keine Istwerte vorhanden. Dies führt zu unsicheren Eingaben und unsicheren Modellresultaten. Darum werden Modellvarianten gebildet, in dem die unsichersten Eingaben variiert werden:

- Die Eingaben für die Intensität der späten Tests sind unsicher. Beispielsweise wird der Systemtest mit erfahrenen Testern durchgeführt, dabei werden auch bewährte Testfälle aus früheren Projekten übernommenen. Die Testfälle werden begutachtet. Dies spricht für einen intensiven Test; es liegen aber keine Werte über Testfälle vor.
- Die Entwickler sind sehr erfahren, der Prozess hat eine hohe Prozessreife. Darum werden möglicherweise weniger Fehler entstehen (Jones, 2003).
- Bei der Vorablieferung handelt sich um einen intensiven, produktiven Einsatz des Systems. Für Erprobungen dieser Art sind kaum Daten verfügbar, insbesondere ist unklar, ob die Quantifizierung des Feldtests in CoBe für eine solche Erprobung geeignet ist. Jones (2007) diskutiert diesen Punkt und zeigt stark unterschiedliche Fehlerentdeckungsquoten.
- Für die Codeanalyse stehen kaum Erfahrungswerte zur Verfügung.

Prüfung	Modelleingaben
Entwurfsreview neuer Software	5 hochkompetente Gutachter bereiten sich mit 10 Seiten pro Stunde vor. Der gesamte Entwurf wird geprüft.
Codeanalyse neuer Software	5 % der Codefehler werden entdeckt und korrigiert, ebenso viele falsche Befunde werden zusätzlich entdeckt
Modultest neuer Software	Vollständiger Black-Box-Test hochkompetenter Tester mit 50 % Aufwand und 1 % Umfang für die Wiederholung, weil der Test zum Teil automatisiert abläuft und einzelne Komponenten entwickelt werden.
Subsystem-Integrationstest neuer Software	Vollständiger Black-Box-Test hochkompetenter Tester mit 50 % Aufwand und 5 % Umfang für die Wiederholung, weil der Test zum Teil automatisiert abläuft und weil für einen bestimmten Test ein bestimmter Systemzustand herbeigeführt werden muss.[a]
Codereview neuen Codes	25 % des Codes werden priorisiert von 5 hochkompetenten Gutachtern mit 200 Anweisungen pro Stunde vorbereitet.
System-integrationstest der gesamten Software	Vollständiger Black-Box-Test hochkompetenter Tester mit 50 % Aufwand und 5 % Umfang für die Wiederholung, weil der Test zum Teil automatisiert abläuft und weil für einen bestimmten Test ein bestimmter Systemzustand herbeigeführt werden muss.[a]
Systemtest der gesamten Software	Vollständiger Black-Box-Test hochkompetenter Tester mit 50 % Aufwand und 5 % Umfang für die Wiederholung, weil der Test zum Teil automatisiert abläuft und weil für einen bestimmten Test ein bestimmter Systemzustand herbeigeführt werden muss.[a]
Vorablieferung der gesamten Software	Fehlerentdeckung des Feldtests (Jones,1998) mit 20 % Entwurfs- und 25 % Codefehler.

Tabelle 88: Modelleingaben für Prüfungen

a. Tritt beispielsweise ein Fehler erst ab einer bestimmten Auslastung des Systems auf, dann muss diese Auslastung erzeugt werden, damit geprüft werden kann, ob der Fehler korrigiert wurde.

- Die Qualität des wiederverwendeten Codes ist vermutlich sehr hoch, weil die Software intensiv erprobt und intensiv eingesetzt wird. Unklar ist aber, wie viele Fehler enthalten sind. Beispielsweise werden nicht alle Fehler sofort korrigiert, weil es sich um ein fehlertolerantes System handelt.
- Die Kompetenz der beteiligten Prüfer ist hoch, weil es sich um erfahrene Entwickler handelt. Sie ist aber trotzdem unsicher, ebenso ist die Zahl der Gutachter im Codereview unterschiedlich und darum unsicher.

Um diese Unsicherheit abzubilden, werden Resultate aus 12 Varianten berechnet und Minimum, Maximum und Median angegeben (Tabelle 89). Ich verwende den Median und nicht den Mittelwert, da die Unsicherheit mit den 12 Varianten nicht durch eine bestimmte Verteilung dargestellt wird, sondern einzelne, auch extreme Werte ver-

wendet. Der Mittelwert würde durch die extremen Fälle verzerrt werden. Insbesondere werden zwei extreme Varianten gebildet. Eine extreme Variante kombiniert alle variierten Eingaben für eine niedrige Fehlerentdeckung in den frühen Phasen, die andere extreme Variante kombiniert die Eingaben mit Werten für hohe Fehlerentdeckung in allen Prüfungen.

Nr.	Modellvarianten für Eingaben, deren Werte nicht verfügbar sind
1	Normalfall (Tabelle 88)
2	Testintensität im Systemintegrationstest, Systemtest und im Feldtest (Faktor 3)
3	Fehlerrate für den besten Fall (50 %)
4	Fehlerentdeckung der Codeanalyse (7 %)
5	Nominale Tester- und Gutachterkompetenz
6	Testintensität im Systemtest und im Feldtest (Faktor 3)
7	Weniger Gutachter im Codereview
8	Feldtest mit Erfahrungswerten für intensiven Test nach Jones (2007), Faktor 3
9	Wenig Fehler in wiederverwendetem Code (0,01 % statt 1 %)
10	Weniger intensive Subsystemtests, weil für Sonderfälle Hardware benötigt wird
11	Worst Case für frühe Fehlerentdeckung
12	Best Case der Fehlerentdeckung

Tabelle 89: Varianten für unsichere Eingaben

Die Unterschiede zwischen den Eingaben für die Subsystem-Projekte fasst Tabelle 90 zusammen. Die Befragten der Subsysteme B und C haben Maximalwerte für die Gutachterzahl im Entwurfsreview angegeben; im Modell rechne ich für alle Subsysteme mit 5 Gutachtern. Obwohl teilweise mehr Gutachter teilgenommen haben, haben manche Gutachter nur einen Teil des Prüflings betrachtet; sie haben beispielsweise nur diejenigen Schnittstellen geprüft, die sie verwendet haben.

Für die Fehlerfolgekosten nehme ich 1000 Verwendungen bis zur Korrektur an, weil mehrere tausend Installationen betrieben werden. Korrekturen werden nicht immer oder spät installiert, weil die Installation ständig verfügbar sein muss. Tritt ein kritischer Fehler auf, dann kostet er 1 Million Euro, weil auf den Rechnern typisch geschäftskritische Anwendungen laufen.

8.4.4 Kalibrierung des Modells

Zuerst werden alle Modellvarianten des Gesamtprojekts mit gleichen Parameterwerten kalibriert. Diese Kalibrierung wird dann auf die Modelle für die Teilprojekte übertragen. Ein erster Vergleich zur Kalibrierung zeigt, dass Aufwand und Dauer

Prüfung	Subsystem A	Subsystem B	Subsystem C
Modultest und Subsystem-Int.-test	Vollständiger Black-Box-Test, 70 % Anweisungsüberdeckung, 10 % Aufwand für Wiederholung.	Nominaler Modultest, 50 % Aufwand für Wiederholung.	Nominaler Modultest, 25 % Aufwand für Wiederholung.
Codereview	25 % des neuen Codes wird priorisiert von 4 Gutachter mit 200 Zeilen pro Stunde vorbereitet.	Hinzugefügter Code wird von 4 Gutachtern mit 200 Zeilen pro Stunde vorbereitet.	25 % des neuen Codes wird priorisiert von 4 Gutachter mit 200 Zeilen pro Stunde vorbereitet.

Tabelle 90: Unterschiede der Modelle für die Subsysteme

kalibriert werden müssen. COCOMO II berechnet zu viel Aufwand, einen zu geringen Personalbedarf und eine zu lange Dauer. Mögliche Gründe sind die starke Parallelisierung mit der Aufteilung in Subsystem-Entwicklungen. Darum wird CoBe für den Aufwand mit dem Faktor 0,66 und für die Dauer mit dem Faktor 0,4 kalibriert. Die Modellresultate für Personalbedarf (Mitarbeiterzahl) und Dauer treffen damit die Istwerte sehr gut (Tabelle 91).

Modellresultat	LE[a]
Mitarbeiterzahl	0,38 dB
Gesamtdauer	0,08 dB
Zahl später Fehler	1,81 dB

Tabelle 91: Abweichungen nach Kalibrierung

a. $LE = 10 \cdot |\log(Modellresultat/Istwert)|$

Für den Vergleich werden die Fehlerzahlen für den Systemintegrationstest und den Systemtest verwendet. Darin sind zusätzlich Fehler enthalten, die während der Entwicklung in anderen Subsystemen entdeckt wurden. Die Modellresultate für Fehler, die in späten Tests entdeckt wurden, treffen diese Zahl gut. Sie liegen etwas niedrig (Tabelle 91). Die Fehlerzahl wird nicht kalibriert, weil die Modellresultate nicht die Fehler, die während der Entwicklung in anderen Subsystemen entdeckt wurden, berücksichtigt.

8.4.5 Vergleich der Modellresultate mit Istwerten

Die Modellvarianten werden nur für das Gesamtprojekt berechnet, weil dafür die meisten Istwerte verfügbar sind. Die folgenden Vergleiche beziehen sich auf das Gesamtprojekt und die Modellvarianten:

Korrekturaufwand pro Fehler in den verschiedenen Phasen

Die Modellresultate für den Korrekturaufwand einzelner Fehler stimmen gut mit den Befragungsergebnissen überein (Tabelle 92). Der Korrekturaufwand ist der Arbeitsaufwand, den ein Entwickler oder mehrere Entwickler für Ursachenanalyse und Änderung investieren. Mehrere Entwickler sind dann beteiligt, wenn sich der Wirkungsmechanismus eines Fehlers über mehrere Subsysteme erstreckt. Dann wurden sehr hohe Aufwände berichtet. Konsistent zum Vergleich mit den Korrekturaufwänden der studentischen Projekte spiegelt CoBe diese Ausreißer nicht, sondern berechnet einen engeren Bereich. Fehler, die in der Codeanalyse entdeckt werden, werden von CoBe zu teuer berechnet.

	Befragungsergebnisse	Modellresultate
Codeanalyse	wenige Minuten, max. 1 Eh	0,8 bis 1,7 Eh
Modultest	normal 1 Eh bis 8 Eh, über 24 Eh möglich	1 bis 1,5 Eh
Codereview	1 Eh normal	0,8 bis 1,7 Eh
Subsystem-Integrationstest	normal 1 Eh bis 16 Eh, über 24 Eh möglich	2,4 bis 5,2 Eh
Systemintegrationstest, Systemtest	2 Eh bis 24 Eh	4,7 bis 10,2 Eh
Wartung	2 Eh bis 24 Eh ohne Prüfung	7,0 bis 15,2 Eh

Tabelle 92: Vergleich des Korrekturaufwands pro Fehler

Fehlerzahlen pro Reviewsitzung

Tabelle 93 zeigt Befragungsergebnisse und Modellresultate für die Zahl der entdeckten Fehler pro Reviewsitzung. Befragungsergebnisse und Modellresultate für das Codereview stimmen sehr gut überein.

Fehlerzahl pro Sitzung	Befragungsergebnis	Modellspanne	Modellmedian
Entwurfsreview	2 bis 10 Fehler	8 bis 16 Fehler	16 Fehler
Codereview	3 bis 15 Fehler	3 bis 12 Fehler	9 Fehler

Tabelle 93: Fehler pro Reviewsitzung

Das Modellresultat für die Fehlerzahl im Entwurfsreview ist dagegen eher zu hoch. Dafür kann es mehrere Gründe geben:

- Der Umfangsfaktor für den Entwurf ist in CoBe zu klein. Dies wird gestützt durch Jones (2007), der einen größeren Faktor angibt (1,25 statt 0,88 Seiten pro Function Point). Mit größerem Umfangsfaktor sinkt die Fehlerdichte, die sich auf die Seitenzahl bezieht.
- Daten über die Zahl der Seiten, die in einer Sitzung besprochen werden, sind nicht verfügbar.
- CoBe berechnet eine zu hohe Fehlerentdeckungsquote. Gegen diese Hypothese spricht, dass CoBe die Fehlerentdeckung durch Reviews der studentischen Projekte gut beschrieben hat. Im Industrieprojekt sind die Gutachter professionelle, erfahrene Entwickler mit viel Domänenwissen, die darum vermutlich sogar eine höhere Fehlerentdeckungsquote als Studenten haben.
- Die Entwickler haben viel Erfahrung und Domänenwissen, bestehende Software wird weiterentwickelt. Darum werden weniger Fehler im Projekt als in durchschnittlichen Projekten gemacht. Dieser Effekt wird aber durch die Kalibrierung der Fehlerzahl berücksichtigt.

Ich führe darum die Abweichungen auf den zu geringen Umfangsfaktor zurück.

Fehlerzahlen in späten Phasen

Die Modellresultate für die Zahl der entdeckten Fehler sind für die späten Prüfungen, d.h. Systemintegrationstest, Systemtest und Vorablieferung, ingesamt etwas zu niedrig, konsistent zur Modellkalibrierung (Abbildung 85[1]). Die Zahl der Fehler sinkt aber konsistent in den Istwerten und in den Modellresultaten. Für den Vergleich werden zwei Fälle unterschieden: Im ersten Fall wird der Median aller Varianten, im zweiten Fall der Median der Varianten mit intensiven späten Prüfungen und intensiver Vorablieferung verwendet. Der Zahl der ausgelieferten Fehler liegt ein erfragter Erfahrungswert für Systemausfälle zu Grunde.

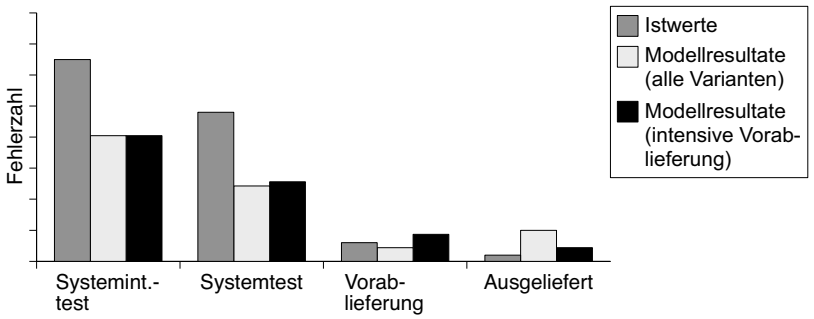

Abb. 85: Median der Fehlerzahlen aus Modellvarianten

1. Die Abbildung enthält aus Gründen der Vertraulichkeit keine absoluten Fehlerzahlen.

Weil die Fehlerzahlen vertraulich sind und darum nicht gezeigt werden, werden Abweichungen der Modellresultate von den Istwerten mit dem logarithmischen Fehler (LE) in dB bewertet. Sie liegt noch unter 3 dB, aber über 2 dB:

- Das Modellresultat für die Fehlerzahl im Systemintegrationstest ist um 2,1 dB zu niedrig. Die Abweichung entsteht, weil für den Istwert auch diejenigen Fehler gezählt werden, die während der Subsystem-Entwicklung in anderen Subsystemen entdeckt wurden.
- Das Modellresultat für die Fehlerzahl im Systemtest ist um 3 dB zu niedrig. Dies erklärt sich durch den Systemtest des Projekts, der intensiver als typische Systemtests ist, weil im Projekt Testfälle vorbereitet, wiederverwendet und begutachtet werden. Der Test erfolgt unabhängig in einer eigenen Testabteilung. Dass der Systemtest sehr intensiv ist, wird durch seine Kosten bestätigt, wie der Abschnitt unten zeigt.
- Die Fehlerzahl der Vorablieferung und der Auslieferung wird von den Modellvarianten mit intensiven späten Tests gut berechnet. Die Abweichung beträgt für Vorablieferung und Auslieferung 0,3 dB. Fehlerzahlen der anderen Modellvarianten sind zu hoch.
- Das Modellresultat für die Zahl der ausgelieferten Fehler wird auch mit intensiven späten Tests zu hoch berechnet. Dafür vermute ich mehrere Gründe: Der Istwert beruht auf der Fehlerzahl pro Jahr und einer Einschätzung für die Einsatzdauer; einzelne Kunden setzen das System aber auch länger ein. Aus den Befragungsergebnissen lässt sich ableiten, dass die Vorablieferung mit internem und externem produktiven Einsatz sehr intensiv prüft. Es werden vor allem kritische Fehler, d.h. Ausfälle, berichtet. Das System ist fehlertolerant, darum treten nicht alle Fehler auf. Da die Fehlerkorrektur risikoreich ist, wurden Fehler, für die das System tolerant ist, nicht korrigiert. Stattdessen wird die Korrektur wird für die nächste Version vorgemerkt, der Fehler wird nicht gezählt.

Kosten für Prüfung und Korrektur

Die Modellresultate der Prüf- und Korrekturkosten sind plausibel:

Die Kosten des Systemtests bestätigen, dass der Systemtest intensiv ist. Die Mitarbeiterzahl weicht unter 0,5 dB ab. Im Projekt findet der Systemtest parallel zur Entwicklung statt, seine Durchführung beginnt etwa zur Halbzeit des Projekts und dauert bis zum Ende. Die Modellresultate für einen intensiven Systemtest ergeben für die Durchführung des Tests 46 % der Projektdauer. Sie stimmen also gut mit der Realität überein.

Die Modellresultate für den Aufwand des Systemintegrationstest stimmen mit den Daumenregeln, die für das Projekt erfragt wurden, überein. Im Modell werden aber zu viele Mitarbeiter zugeordnet, so dass die Dauer zu niedrig berechnet wird. Mit realistischer Mitarbeiterzahl ergibt sich die gleiche Dauer wie für den Systemtest, dieses Resultat entspricht in etwa der Realität.

Die Modellresultate ergeben plausible Aufwandsanteile für Prüfung und Korrektur im Vergleich zum erfragten Gesamtaufwand des Projekts, da die Modellvarianten im Median einen Prüf- und Korrekturaufwand im Projekt von 38 % des erfragten Projektaufwands ergeben, mit intensiven späten Prüfungen 52 %.

Vergleicht man diese Resultate mit der Aufwandsverteilung auf Phasen in COCOMO II, zeigt sich, dass die CoBe-Resultate auch im Vergleich dazu plausibel sind: So machen die Modellresultate für Prüf- und Korrekturaufwand jeweils rund 20 % des Aufwands der Entwurfsphase und der Implementierungsphase in COCOMO II aus (Tabelle 94). Der Median aller Modellvarianten für Integration und Test trifft etwa die untere Grenze des COCOMO-II-Integrations- und Testaufwands. Der Median der Varianten mit intensiven Tests überschreitet die obere Grenze leicht.

Aufwandsanteil der Phasen, mittlere bis große Projekte in COCOMO II		Median der Modellresultate für Prüfung und Korrektur	
Phase[a]	Anteil (%)[b]	Prüfung	Anteil (%)[c]
Product Design	16 % - 18 %	Entwurfsreview	4 %
Programming	48 % - 62 %	Modultest, Codeanalyse, Codereview, Subsystem-Int.-test	11 %
Integration and Test	22 % - 34 %	Systemintegrationstest, Systemtest (Alle Modellvarianten)	23 %
		Modellvarianten mit intensivem Systemint.-Test und Systemtest	37 %

Tabelle 94: Aufwandsanteile im Vergleich

a. Die Phase Plans and Requirements ist nicht dargestellt.
b. Alle Anteile beziehen sich auf den Gesamtaufwand des Projekts.
c. Alle Anteile beziehen sich auf den erfragten Gesamtaufwand des Projekts.

8.4.6 Resultate der Referenzmodelle

Fehlerzahlen des Gesamtprojekt werden gut berechnet, weil die Resultate für Systemtest und Vorablieferung weniger als 2 dB abweichen. Abbildung 86 enthält aus Gründen der Vertraulichkeit keine absoluten Fehlerzahlen. Der Vergleich für die einzelnen Subsystem-Projekte erfolgt für eine Eingabekombination, die als passende Variante gewählt wurde. Die späten Prüfungen sind intensiv; andere Prüfungen sind an den Prüfprozess der Subsysteme angepasst (Tabelle 87). Die Zahl der ausgelieferten Fehler wird von CoBe etwas zu hoch berechnet (LE = 2,1 dB), dies lässt sich aber mit dem fehlertoleranten System begründen.

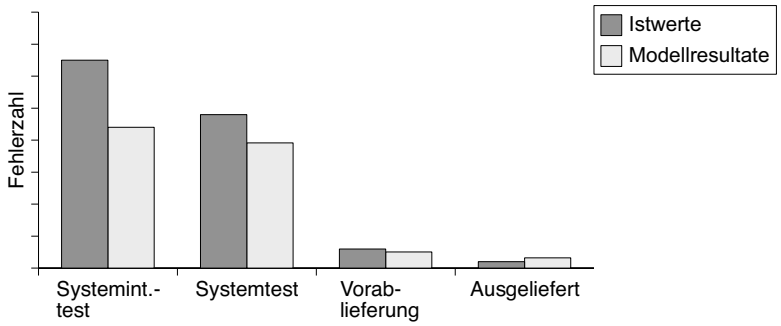

Abb. 86: Fehlerzahlen des Referenzmodells

Mit dieser Kalibrierung wird ein Modell pro Subsystem erstellt und die Resultate gegen Istwerte verglichen. Die Resultate sind plausibel, weil die Zahl der Entwickler mit guter, zum Teil sogar sehr guter Genauigkeit berechnet wird (Tabelle 95). Die Entwicklungsdauer für die Subsystem-Projekte ist in den Modellen auf die Projektdauer der gesamten Entwicklung gesetzt.

Entwicklerzahl	Istwerte	A	B	C
pro Subsystem	etwa 20	28	23	16

Tabelle 95: Entwicklerzahl pro Subsystem

Die Modellresultate zeigen, dass die Prüfung und Korrektur einen großen Anteil des Aufwands der Subsystem-Entwicklung einnimmt (Tabelle 96). Dieser hohe Anteil erklärt sich durch die Einbettung dieser Subsystemprojekte in das gesamte Projekt. Dabei gehören beispielsweise Projektmanagement, Systemintegration und Systemtest nicht zum Subsystemprojekt. Somit sind diese Aufwände, die sich auf das gesamte System beziehen, nicht im erfragten Aufwand für die Entwicklung der einzelnen Subsysteme enthalten.

Modellresultat für den Aufwandsanteil[a]	A	B	C
Entwurfsreview mit Korrektur	19 %	15 %	10 %
Codereview mit Korrektur	8 %	18 %	5 %
Modultest mit Korrektur	18 %	9 %	6 %
Subsystem-Integrationstest mit Korrektur	22 %	13 %	8 %

Tabelle 96: Aufwandsanteil für Prüfungen und Korrektur

a. Der Anteil bezieht sich auf den Aufwand, der sich aus den Befragungsergebnissen für die Zahl der Mitarbeiter im Subsystem-Projekt und die Dauer ergibt.

8.5 Industrieprojekt 2

CoBe wird mit dem Industrieprojekt 2 mit dem gleichen Vorgehen wie mit dem Industrieprojekt 1 validiert: Nach der Analyse (Abschnitt 8.5.1) erfolgt die Abbildung in CoBe (Abschnitte 8.5.2 und 8.5.3). Resultate sind in den Abschnitten 8.5.4 und 8.5.5.

Im Projekt wurde Steuergeräte-Software für PKW im Kundenauftrag entwickelt. Steuergerät und Software bilden eine Einheit. Es handelt sich um eine nahezu vollständige Neuentwicklung, Ausnahme sind mathematische Bibliotheken. Das Projekt wurde als sicherheitskritisch (SIL-3, Smith und Simpson, 2005) eingestuft. Das Projekt ist SPICE Level 3 (Hörmann et al., 2006) mit definiertem Standard-Prozess. Im Projekt waren in etwa 2,5 Jahren rund 14 Mitarbeiter, teilweise in Teilzeit, beschäftigt. Die Software umfasst rund 35 000 Anweisungen.

8.5.1 Das Projekt und sein Prozess

Einbettung in die Projektumgebung und Anforderungen

Die Anforderungen an das System stammten vom Kunden. Das Steuergerät ist standardisiert. Das Projekt verlief iterativ, indem Anforderungen mit dem Kunden abgesprochen wurden; der Kunde bekam dann regelmäßig – alle zwei Monate – eine Lieferung. Der betrachtete Ausschnitt der Realität enthält die Anforderungsanalyse und -prüfung durch den Kunden.

Ablauf und Organisation der Entwicklung

Die Entwicklung erfolgte iterativ. Jede Iteration bildete eine Phase. Für jede Phase wurde die zu realisierende Funktionalität geplant, nach jeder Phase wurde an den Kunden ausgeliefert. Eine Phase dauerte zwei Monate. In jeder Phase wurde ein sequentieller Entwicklungsprozess durchlaufen. Mit diesem Prozess wurden zuerst System-, dann Software-Anforderungen festgelegt, dann entworfen, implementiert, dann integriert, parametrisiert und schließlich ausgeliefert. Anforderungen wurden werkzeuggestützt als identifizierbare Einheiten verwaltet. Es wurde ein Funktionsentwurf erstellt. Die Programmierung erfolgte in C.

Prüfungen und Prüfprozesse

Für die Validierung werden zwei Prüfprozesse in CoBe abgebildet. Sie liefen ineinander verwoben ab: Ein Prüfprozess des Entwicklungsprozesses, d.h. initiale Prüfungen in jeder Phase, und ein Prüfprozess für Änderungen waren vorgegeben. Reviews sind für ein SIL-3-Projekt obligatorisch. In jeder Phase wurden die folgenden initialen Prüfungen durchgeführt:

- Kundenreviews und internes Anforderungsreview
- Entwurfsreview
- Codeanalyse
- Modultest

- Codereview
- Releasetest (entspricht dem Integrationstest)
- Software-Test (HiL- und SiL-Test, entspricht dem Systemtest)[1]
- Interne und externe Fahrzeugtests

Änderungen, z.B. Korrekturen, wurden in einem definierten Änderungsprozess parallel zur Entwicklung geprüft mit:

- Codeanalyse
- Modultest
- Codereview
- Software-Test (HiL- und SiL-Test, entspricht dem Systemtest)

Im Folgenden werden die Prüfungen nicht mit den firmenspezifischen Begriffen (Releasetest, Softwaretest), sondern mit den Begriffen von CoBe (Integrationstest, Systemtest) bezeichnet.

Die Prüfungen der Änderungen erfolgten abgestimmt auf den Entwicklungsprozess (Abbildung 87): Die Integration erfolgte regelmäßig. Dabei wurden sowohl neue Komponenten als auch Änderungen integriert und danach durch einen Integrationstest geprüft. Der Systemtest wurde für jede Änderung erweitert; die Änderungen wurden separat getestet. Der Systemtest wurde für neu entwickelte Teile erweitert. Bestehende Testfälle wurden wiederholt. Die integrierte Software wurde dann internen und externen Fahrzeugtests unterzogen. Tabelle 97 fasst die erhobenen Daten des Projekts zusammen.

Merkmale einzelner Prüfungen

Die detaillierten Modelleingaben basieren auf den folgenden Analyseergebnissen: Die Anforderungen wurden in Reviews mit 4 bis 5 Gutachtern beim Kunden diskutiert und abgesprochen. Interne Spezifikationsreviews fanden mit 2 bis 3 Gutachtern statt. Dokumentationsreviews wurden durchgeführt. Anforderungen und Dokumentation ließen sich aus Sicht der Beteiligten schwer trennen. Auch die Trennung zwischen Anforderungsänderung, Anforderungsanalyse und Fehlerentdeckung ist in diesen Prüfungen unscharf.

Der Entwurf erfolgte mit UML und wurde durch Entwurfsreviews in Form eines Walkthroughs mit einem Gutachter, teilweise mit Moderator, geprüft.

Codereviews wurden entweder informal oder formal durchgeführt. Informale Reviews konnten nur bei unkritischen (SIL-0-)Modulen oder unkritischen Änderungen stattfinden. Befunde informaler Reviews konnten vom Autor direkt korrigiert

1. HiL (Hardware-in-the-Loop): Die Software wird mit dem Steuergerät in einer simulierten Umgebung getestet. SiL (Software-in-the-Loop): Das Steuergerät wird zusätzlich simuliert, nur die Software wird in einer simulierten Umgebung getestet.

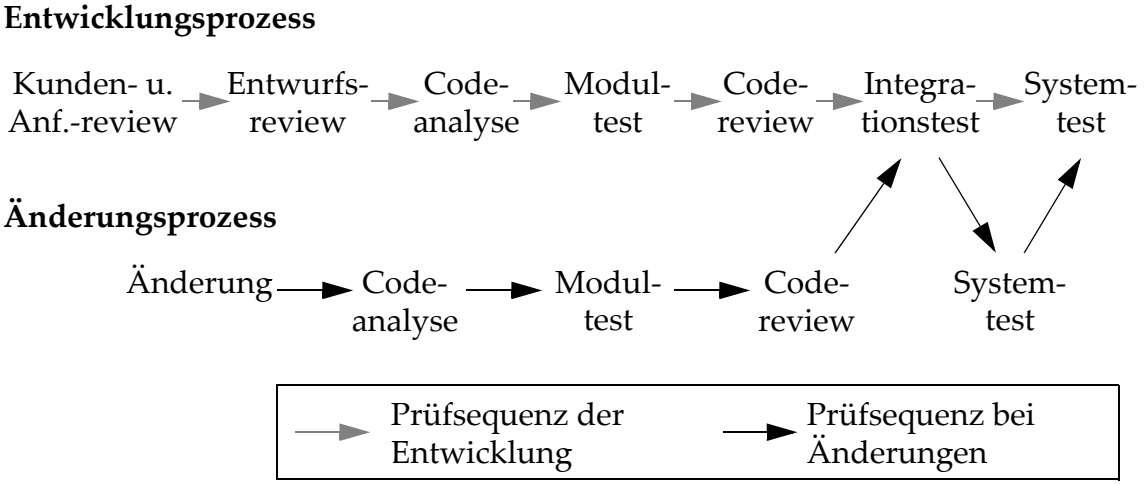

Abb. 87: Einbindung des Änderungsprozesses

Verfügbare Istwerte
Umfang des Codes
Zahl der Anforderungen und Anforderungsänderungen
Zahl der Änderungen
Aufwand und Aufwandsverteilung auf Entwicklung und Prüfung von Änderungen
Zahl der späten Fehler (Systemtest und Fahrzeugtest)
Aufwand für späte Fehler
Anforderungs- und Zweigüberdeckung, Umfang pro Modul im Softwaretest
Aufwand für Prüfungen (Initial und Änderung, ohne Modultest)
Dauer und Mitarbeiterzahl
Gutachterzahl in Reviews

Tabelle 97: Istwerte des Industrieprojekts 2

werden, während Befunde formaler Reviews einem Prozess unterworfen und formal verwaltet wurden: Die Korrektur dieser Befunde folgte dem Änderungsprozess. Codereviews wurden für neue Module und für Änderungen als Walkthrough mit einem Gutachter, teilweise mit Moderator, durchgeführt. Bei Änderungen mit SIL-3 wurde die gesamte geänderte Funktion erneut geprüft. Bei Änderungen mit SIL-0 reichte aus, die geänderte Software zu betrachten.

Der Modultest erfolgte durch die Entwickler nach dem Vier-Augen-Prinzip. Er wurde in der Entwicklungsumgebung durchgeführt und war weitgehend nicht automati-

siert. Der Ablauf des Tests und die Testfälle wurden anhand einer Vorlage dokumentiert. Der Modultest wurde für ein neues Modul und bei jeder Änderung durchgeführt. Im Modultest wird keine Überdeckung gemessen, er findet also nicht als Glass-Box-Test statt.

Die Integration erfolgte durch einen Integrator und nach dem Vier-Augen-Prinzip. Der Integrationstest stellte sicher, dass die Software prinzipiell funktioniert. Dabei wurden externe Schnittstellen, z.B. für Sensoren, und interne Schnittstellen geprüft.

Nach diesem Test wurden Systemtest und Fahrzeugtests parallel durchgeführt. Der Systemtest bestand aus mehreren Teilen: Für jedes Release wurden im HiL-Test automatisiert die Anforderungen geprüft. Testfälle für SIL-3-Anforderungen waren zu 96 % automatisiert, für SIL-0-Anforderungen zu 65 %, die übrigen wurden manuell geprüft. Das Kriterium für die Zweigüberdeckung wurde für jedes Modul individuell festgelegt und wurde im SiL-Test überprüft. Die Testfälle waren für den HiL-Test und für den SiL-Test weitgehend gleich. Ausnahmen waren wenige Testfälle, die nicht im SiL-Test geprüft werden konnten, weil dazu Hardware benötigt wurde. Die Testfälle wurden durch Tester erstellt und dann in einem Review geprüft. Sie wurden als Testsequenz implementiert, so dass sie automatisch durchgeführt werden konnten. Änderungen im Änderungsprozess wurden im HiL-Test geprüft; die Überdeckung wurde im SiL-Test kontrolliert, bei Bedarf wurden die Testfälle erweitert. Die internen und externen Fahrzeugtests fanden teilweise gemeinsam mit dem Kunden statt.

8.5.2 Die Abbildung in das Modell

Dieser Prozess wird in das Modell abgebildet. Die detaillierten Eingaben für Prüfungen sind im Abschnitt 8.5.3, die Resultate in den Abschnitten 8.5.4 und 8.5.5.

Iterativer Entwicklungsprozess mit initialen Prüfungen

Der iterative, stufenförmige Entwicklungsprozess mit den initialen Prüfungen wird auf den sequentiellen Prozess in CoBe nach dem gleichen Prinzip wie im Validierungsprojekt 1 abgebildet (Abbildung 84). Dies ist möglich, weil in den einzelnen Iterationen jeweils der gleiche, sequentielle Prozess durchlaufen wird. Dadurch bleibt das Modell einfach. Diese Abbildung in das Modell hat aber Nachteile: Effekte durch mehrfach wiederholte Prüfungen können nicht sichtbar werden; die grundlegende Annahme ist, dass die Entwicklung additiv verläuft und nur additive Effekte enthält. Sie wächst also stetig, bestehende Teile werden bei der Entwicklung nur geringfügig geändert. Die Alternative, jede Iteration durch ein Modell darzustellen, hätte zu einem extrem komplexen und unübersichtlichen Modell geführt. Die Auswahl der Hardware, Entwurfsentscheidungen zur Hardware und zur Realisierung der Redundanz für SIL-3-Systeme werden im Modell nicht dargestellt, weil CoBe keine Zusammenhänge für Hardware-Entwicklung enthält.

Anforderungskritikalität

Die Entwicklung und Prüfung erfolgte für SIL-3-Module anders als für SIL-0-Module. Diese unterschiedliche Entwicklung bilde ich in CoBe durch zwei unterschiedliche

Modellinstanzen ab. Das SIL-0-Modell beschreibt die Entwicklung und Prüfung der SIL-0-Software, das SIL-3-Modell beschreibt die Entwicklung und Prüfung der SIL-3-Software.

Verfügbare Umfangsdaten sind der Code-Umfang (rund 35 000 Anweisungen), die Zahl der SIL-0-Anforderungen und die Zahl der SIL-3-Anforderungen. Der Code-Umfang wurde proportional zur Zahl der Anforderungen auf SIL-0- und auf SIL-3-Modell verteilt:

Da 67 % der Anforderungen SIL-0-Anforderungen sind, folgere ich, dass 67 % des Codes mit SIL-0 entwickelt wurden. In das SIL-0-Modell wird also eingegeben, dass der Umfang hinzugefügten Codes 67 % der rund 35 000 Anweisungen beträgt, also etwa 24 000 Anweisungen.

Aus den 33 % SIL-3-Anforderungen ergibt sich somit, dass der Umfang hinzugefügten Codes im SIL-3-Modell etwa 11 000 Anweisungen beträgt.

Anforderungsänderungen

Während des Projekts änderte der Kunde Anforderungen. Diese Anforderungsänderungen bilde ich in CoBe als zusätzlichen, geänderten Code ab, angelehnt an Boehm (2000). Diese Modellierung basiert auf der Annahme, dass bestehender Code ersetzt werden muss, wenn sich Anforderungen ändern; Code wird also gelöscht und neu erstellt. Dies entspricht der Definition für geänderten Code aus IEEE 1045 (1992).

Daten über den Umfang geänderten Codes sind nicht verfügbar. Darum nehme ich an, dass für jede Anforderungsänderung genau so viel Software wie für eine neue Anforderung entwickelt wird. Ich modelliere also, dass pro Anforderung und pro Anforderungsänderung der gleiche Umfang Code implementiert und geprüft wird.

Von den SIL-0-Anforderungen wurden 9 % geändert. Daraus folgere ich, dass 9 % der hinzugefügten Anforderungen geändert wurden. Zu den 24 000 Anweisungen kommen also rund 2 200 geänderte Anweisungen dazu.

Rund 6 % der SIL-3-Anforderungen änderten sich. Somit wurden 6 % der hinzugefügten Anweisungen geändert, also etwa 710 Anweisungen der 11 000 Anweisungen im SIL-3-Modell.

Tabelle 98 fasst die Eingaben für die beiden Modellinstanzen zusammen.

Eingabeparameter	SIL-0-Modell	SIL-3-Modell
Hinzugefügte Software (Anweisungen)	24 000	11 000
Geänderte Software (Anweisungen)	2 200	710

Tabelle 98: Eingaben für den Software-Umfang

Abstraktionsebenen und Prüfprozess

Beim Projekt handelt es sich um eine Systementwicklung. Darum kläre ich die Abbildung der Entwicklung in CoBe wie folgt: Ich betrachte nur den Teil des Projekts für die Software-Entwicklung des Steuergeräts. Im Projekt gab es Entscheidungen, die die Wahl der Hardware betreffen. Diese fanden zu Beginn des Projekts statt, so dass ich von vorgegebener Standard-Hardware im Modell ausgehe. System- und Software-Anforderungen können in dieser Situation nicht getrennt betrachtet werden, darum wird im Modell keine Unterscheidung getroffen. Spezifikationsfehler betreffen also alle Anforderungen, ich vernachlässige aber die Anforderungen an das Steuergerät. Entwurfsfehler werden im Modell nur für die Software abgebildet. Codefehler entstehen auch im Projekt nur für die Software.

Alle Reviews der Anforderungen werden im Modell zum Spezifikationsreview zusammengefasst. Unklar ist der Umfang und der Aufwand, der durch die zusätzlichen Kundenreviews entsteht. Entwurfsreview, Codeanalyse und Modultest werden direkt in das Modell abgebildet. Der Releasetest entspricht dem Integrationstest, er hat das gleiche Ziel: Die Schnittstellen und die grundlegende Funktion werden geprüft. Eine Aufteilung in Subsystemintegration und Systemintegration gibt es im Projekt nicht und wird darum auch in CoBe nicht dargestellt. Der Softwaretest des Projekts entspricht dem Systemtest in CoBe: Die gesamte Software wird gegen die Anforderungen geprüft.

Änderungsprozess

Der Änderungsprozess für eine Korrektur wird durch die Korrektur und die darauf folgenden Prüfungen der Korrektur (Korrekturprüfprozess in Cobe) abgebildet: Die Änderung wird vom Entwickler im Modultest geprüft und durch einen Gutachter im Codereview geprüft. Die Änderung wird im Systemtest geprüft.

8.5.3 Modelleingaben im Detail

Im Folgenden werden die detaillierten Modelleingaben gezeigt: Der Umfang des C-Codes wird mit 128 Anweisungen pro Function Point (Boehm, 2000) und 0,44 Seiten pro Function Point für Spezifikation und Entwurf umgerechnet. Aufwand, Personalbedarf und Fehlerzahl werden proportional zum Umfang auf das SIL-0- und das SIL-3-Modell verteilt.

Die Fehlerdichte für entstehenden Fehler und die Verteilung auf die Fehlerschwere und die Fehlerart sind an die Werte für Systemsoftware aus Jones (1996) angepasst. Die Parameter von COCOMO II sind entsprechend den Befragungsergebnissen gesetzt.

Die Korrekturen folgen dem Korrekturprüfprozess, wenn Fehler im Codereview, dem Integrationstest, dem Systemtest und in den Fahrzeugtests entdeckt werden. Da der Modultest durch die Entwickler erfolgt, findet die Korrektur nach diesem Test direkt statt. Im Korrekturprüfprozess werden Codereview, Modultest und Systemtest

gezielt wiederholt. Der Integrationstest wird vollständig wiederholt (Abbildung 87). Tabelle 99 zeigt die Prüfungen und ihre Parameter.

Prüfung	Parameter
Spezifikations-review	Vier hochkompetente Gutachter prüfen die gesamte Spezifikation und bereiten sich gründlich vor.
Entwurfsreview	Ein hochkompetenter Gutachter prüft den gesamten Entwurf und bereitet sich gründlich vor.
Codeanalyse	Durch Codeanalyse wird eine Entdeckungsquote von 5 % der Codefehler erzielt.
Modultest	Der Modultest ist ein vollständiger Black-Box-Test. Die Wiederholung kostet 50 % des Aufwands. Für eine Korrektur muss 1 % der Testfälle wiederholt werden.
Codereview	Ein hochkompetenter Gutachter prüft den gesamten Code und bereitet sich gründlich vor. Nach einer SIL-0-Korrektur werden 25 LoC begutachtet (Jones, 2007), nach einer SIL-3-Korrektur wird ein Modul (gemessen: 387 LoC pro Modul) begutachtet.
Integrationstest	Der Integrationstest erfolgt als normaler Black-Box-Test (Funktionen und Äquivalenzklassen). Die Wiederholung kostet 50 % des Aufwands.[a]
Systemtest	Der Systemtest erfolgt zuerst als Black-Box-Test (Funktionen, Äquivalenzklassen, Sonderfälle) und wird dann für den Glass-Box-Test mit 83 % Zweigüberdeckung (gemessen im Projekt) ergänzt. Die Wiederholung kostet 10 % des Aufwands. Für eine Korrektur müssen 10 % der Testfälle wiederholt werden.
Fahrzeugtest	Die Quantifizierung des Feldtests wird aus Jones (1996) übernommen.

Tabelle 99: Parameter der Prüfungen im Modell

a. Der Test ist weitgehend automatisiert und würde darum weniger für eine Wiederholung kosten. Weil er aber mehrfach wiederholt wird, wähle ich diesen höheren Wert.

Für das Projekt sind viele Istwerte verfügbar. Trotzdem sind Werte einiger Ein- und Ausgaben von CoBe nicht bekannt. Diese Unsicherheit wurde durch Modellvarianten abgebildet:

- Der Einfluss der Sicherheitsanforderungen auf den Aufwand und auf den Aufwandsfaktor ist unklar.
- Die Prozessreife ist hoch, die Mitarbeiter erfahren. Dies senkt die Zahl entstehender Fehler.

- Die Verteilungen auf Fehlerart und -schwere stehen nur für späte Phasen zur Verfügung; die Abbildung auf der im Projekt verwendeten Fehlerschwere-Definition auf die Definition von CoBe ist unklar.
- Die Kompetenz der Prüfer wird subjektiv eingeschätzt, darum werden zwei Varianten mit höherer und niedrigerer Kompetenz gebildet.
- Unklar sind Umfang und Umfangsfaktoren von Spezifikation und Entwurf.
- Es ist unklar, ob Anweisungen oder Zeilen gemessen werden
- Unklar ist die Intensität und Fehlerentdeckungsquote der Fahrzeugtests.
- Es ist unklar, wie viel Aufwand für die Wiederholung von Tests benötigt wird. Darum werden verschiedene Annahmen durchgerechnet.

Tabelle 100 fasst die Varianten zusammen.

Nr.	Modellvarianten
1	Normalfall
2	Produktivität durch Sicherheitsanforderungen
3	Niedrige Fehlerzahl für hohe Prozessreife (Jones, 2003)
4	Fehlerverteilung mit gemessenen Werten der späten Phasen
5	Nominale (anstatt hoher) Kompetenz der Prüfer
6	Sehr hohe Kompetenz der Prüfer
7	Umfangreiche Spezifikation, umfangreicher Entwurf (1,25 statt 0,44 Seiten / FP)
8	Messung des Code-Umfangs in Lines of Code statt in Anweisungen
9	Intensive Fahrzeugtests (Faktor 3 im Vergleich zum Feldtest)
10	Fehlerverteilung auf Fehlerschwere ohne kosmetische Fehler (Jones, 1998)
11	Höherer Wiederholungsanteil im Modultest
12	Höherer Aufwandsanteil für Wiederholung im Systemtest: 25 % mit Funktionsänderung statt 10 % ohne Änderung (van Megen und Meyerhoff, 1995)

Tabelle 100: Varianten für unsichere Eingaben

8.5.4 Vergleich der Modellresultate mit Istwerten

Kalibrierungsfaktoren

Die Fehlerzahl des Modells wurde kalibriert, aber nicht Aufwand und Dauer, weil in diesem Projekt die COCOMO-Resultate für den Gesamtaufwand und für die Gesamtdauer gut mit den Istwerten übereinstimmen (Tabelle 101). Die Modellresultate liegen leicht unter dem tatsächlichen Aufwand. Ich führe dies vor allem darauf zurück, dass

Mitarbeiter nicht immer Vollzeit im Projekt arbeiteten. Hardware-bezogene Aktivitäten und Prozessanforderungen für SIL-3 werden in COCOMO II nicht oder nicht ausreichend berücksichtigt. Aufwand und Dauer werden nicht kalibriert, weil die Prüfparameter für SIL-3 in CoBe explizit berücksichtigt werden und sich Zusatzarbeiten bei der Bottom-up-Schätzung nicht auf die betrachteten Aktivitäten auswirken.

Abweichung von	LE[a]
Mitarbeiterzahl	0,2 dB bis 3,7 dB
Dauer	0,3 dB bis 1,9 dB

Tabelle 101: Abweichungen der COCOMO-II-Resultate in den Modellvarianten

a. $LE = 10 \cdot |\log(Modellresultat/Istwert)|$

Dagegen zeigt der Vergleich der spät entdeckten Fehler für den Normalfall, dass das Modellresultat für die Fehlerzahl etwa doppelt so hoch wie der Istwert ist. Darum wurde die Fehlerzahl mit dem Fehlerfaktor auf 50 % angepasst.

Entdeckte Fehler

Abbildung 88 zeigt den Median der Fehlerzahlen aus den Modellvarianten im Vergleich zu Istwerten für Änderungsaufträge. Bei den Prüfungen, bei denen keine Istwerte als Balken dargestellt sind, sind keine Fehlerzahlen verfügbar. Weil die Daten vertraulich sind, ist die Skala nicht beschriftet.

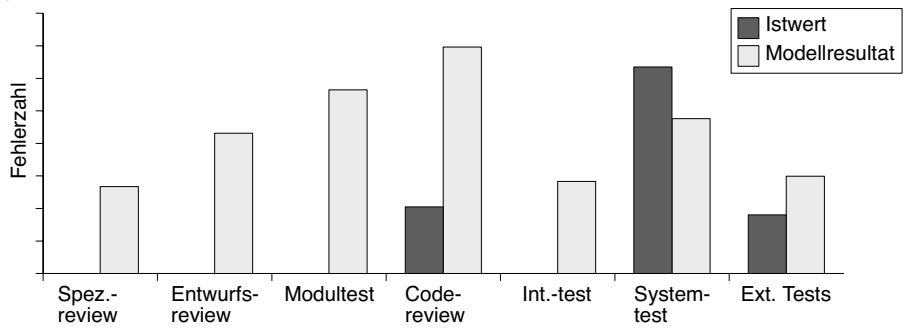

Abb. 88: Fehlerzahlen und Änderungsaufträge im Vergleich

- Istwerte für Fehler in Kundenreviews, internen Spezifikationsreviews, Entwurfsreviews, Modultests, in der Codeanalyse und im Integrationstest sind nicht verfügbar.
- Für die Codereviews ist die Zahl der Änderungsaufträge verfügbar: Entdeckte Fehler in formalen Codereviews wurden über Änderungsaufträge korrigiert. Ein Änderungsauftrag konnte mehrere Befunde enthalten, weil der Auftrag einem zu ändernden Modul zugeordnet ist; er erlaubt, das Modul auszuchecken. Nebenfehler wurden im Zuge späterer Änderungen korrigiert und nicht als eigener Ände-

rungsauftrag behandelt. Das Modell berechnet im Median rund drei mal so viele Fehler wie Änderungsaufträge (Minimum: Faktor 2, Maximum: Faktor 4). Die Befragten bewerteten dieses Verhältnis als plausibel.

- Die berechnete Fehlerzahl und der Istwert für Änderungsaufträge stimmen für den Systemtest (einschließlich interne Fahrzeugtests) gut überein (*LE* = 1,3 dB). Die Abweichung für externe Tests durch den Kunden ist höher (*LE* = 2,2 dB). Ich führe dies auf den Feldtest im Modell zurück, der auf Jones (1996) basiert. Er ist nicht vergleichbar mit einer intensiven Fahrzeugtest. Die Modellvariante mit intensivem Feldtest zeigt sehr geringere Abweichungen (*LE* für Systemtest und interne Fahrzeugtests: 0 dB, *LE* für externe Fahrzeugtests: 1,2 dB).

Initiale Prüfkosten

Die Modellresultate für den Aufwand initialer Prüfungen stimmen gut mit den Istwerten überein. Abbildung 89 zeigt den Aufwand in Entwicklerstunden. Die Werte sind nicht dargestellt, weil die Daten vertraulich sind. Die Ausnahme ist der Integrationstest, für den das Modellresultat zu niedrig ist.

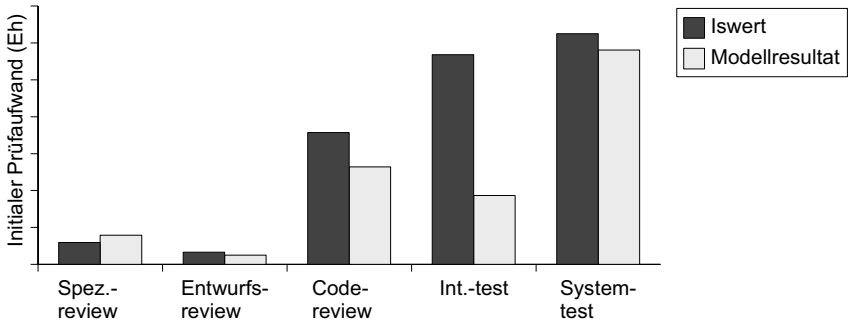

Abb. 89: Aufwand für initiale Prüfungen

Der Vergleich der Istwerte mit dem Median der Modellresultate zeigt:

- Der Aufwand für das Spezifikationsreview weicht um 1,3 dB ab. Das Resultat ist gut. Die Abweichung kann dadurch erklärt werden, dass in CoBe die Kundenreviews nicht dargestellt werden und dass im Projekt Spezifikation und Dokumentation nicht strikt getrennt wurden. Der Prüflingsumfang und der Umfangsfaktor sind für das Projekt nicht bekannt, da Anforderungen als einzelne Einheiten verwaltet wurden.
- Der Aufwand für das Entwurfsreview wird vom Modell mit 1,2 dB gut berechnet. Der Umfang des Prüflings ist unklar, Jones (2007) zeigt einen großen Bereich für den Umfangsfaktor.
- Das Modellresultat für Codereviewaufwand ist gut, liegt aber etwas zu niedrig (1,3 dB). Die Istwerte des Projekts enthalten aber auch Aufwand für formale Codereviews nach der Fehlerkorrektur.

- Für Modultest und Codeanalyse sind keine Istwerte verfügbar.
- Das Modellresultat für den Integrationstest ist zu niedrig, die Abweichung ist 4,8 dB. Die Ursache vermute ich im iterativen Vorgehen, dabei wird der Integrationstest mehrmals wiederholt. Dieser Aspekt wird in CoBe nicht dargestellt.
- Das Modellresultat für den Systemtest-Aufwand stimmt sehr gut mit dem Istwert überein (0,3 dB).

Die Testfallzahl wird zu niedrig berechnet (3,9 dB). Ich vermute, dass sich die Erfahrungswerte von Jones (2007) nicht ohne weiteres auf die Entwicklung von Steuergeräte-Software übertragen lassen, weil die Werte auf Function Points beruhen, die für die betrachtete Domäne nicht geeignet sind. Außerdem wurden im Projekt aus den Testfällen Testsequenzen abgeleitet; dazu müssen die einzelnen Testfälle relativ detailliert definiert sein. Somit entstehen also mehr, dafür feingranulare Testfälle. Aufwand und Fehlerentdeckung sind aber plausibel.

Aufwand für Korrekturen und Prüfung der Korrekturen

Der Aufwand für Korrektur und Prüfung der Fehler, die im Systemtest und den Fahrzeugtests entdeckt wurden, stimmt gut zwischen Modell und Projekt überein (Tabelle 102). Die Istwerte liegen im Bereich, den die Modellvarianten aufspannen. Der Median aus dem Projekt liegt nahe am Minimum der Modellvarianten. Der Mittelwert aus dem Projekt liegt nahe beim Median der Modellvarianten. Da jede einzelne Variante einen statistischen Mittelwert für den Korrekturaufwand pro Fehler berechnet, ist der Vergleich zwischen Mittelwert und Median möglich. Da die Modellvarianten auch extreme Varianten enthalten, wird kein Mittelwert der Modellvarianten berechnet; er würde durch die extremen Varianten verzerrt, der Median ist robuster.

Aufwand (Eh) pro Fehler	Istwert		Modellresultat		
	Mittelwert	Median	Median	Minimum	Maximum
SIL-0	20,1	11,0	20,1	9,6	46,0
SIL-3	19,6	13,0	22,5	12,3	47,7

Tabelle 102: Vergleich des Aufwands zur Korrektur und Prüfung eines Fehlers

Der Aufwand, der für die Behebung der späten Fehler (Korrektur und Prüfung der Korrektur) insgesamt anfällt, wird mit hoher Genauigkeit berechnet. So liegt der Median der Modellresultate nur etwas zu niedrig (1,5 dB). Die Ursache für diese Abweichung sind die zu niedrigen Modellresultate für den Modultest. Dies zeigt die Aufwandsverteilung der Änderungen (Abbildung 90). Die gewählten Wiederholungsanteile für die Tests, vor allem für den Modultest, wurden zu niedrig gewählt; diese Einschätzung wurde von den Befragten bestätigt. Die Verteilung der Istwerte basiert auf allen Änderungen. Sie enthält damit auch Aufwände für Anforderungsän-

derungen und andere Änderungen. Aufwand für Codereviews der Änderungen wurde im Projekt nicht gesondert dokumentiert und ist darum nicht dargestellt.

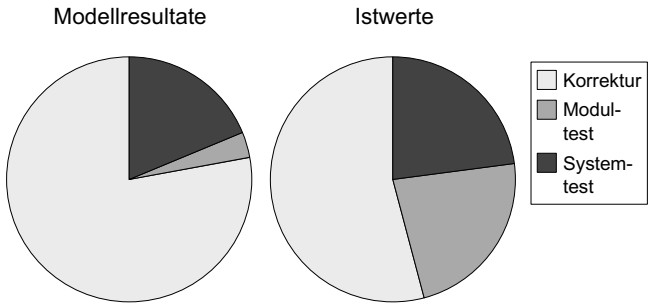

Abb. 90: Aufwandsverteilung auf Korrektur und Prüfung der Korrektur

8.5.5 Resultate des Referenzmodells

Der Median der Modellvarianten stimmt gut mit den Istwerten des Projekts überein. Einige wenige Werte weichen deutlich ab, liegen aber noch im Rahmen des Validierungskriteriums. Diese Abweichungen lassen sich auf den Prozess und die Domäne zurückführen; diese Einschätzung wurde in der Diskussion mit den Projektbeteiligten bestätigt. Für das Referenzmodell werden diese Erkenntnisse als Eingaben übernommen:

- Der Wiederholungsanteil im Modultest wird auf 10 % erhöht. Dies entspricht der Einschätzung der Befragten, die für den Modultest einer Änderung einen wesentlich höheren Aufwand erwarten als im Systemtest.
- Für die Testwiederholung im Systemtest wird 20 % Zusatzaufwand benötigt, ein Kompromiss zwischen dem Zusatzaufwand für Korrekturen ohne Funktionsänderung und dem Zusatzaufwand für Korrekturen mit Funktionsänderung (van Megen und Meyerhoff, 1995).
- Die Fehlerverteilungen für Fehlerart und Fehlerschwere werden vom Industrieprojekt übernommen.
- Für interne Fahrzeugtests wird die Intensität des Feldtests mit dem Faktor 2,5 angepasst; der Kunde entdeckt 95 % der Fehler vor Produktionsbeginn.
- Die Fehlerschwere ist durch den Schaden definiert. Kritische Fehler können Personenschäden verursachen; bei schweren Fehlern erfolgt eine Rückrufaktion. Kritische Fehler werden also mit 10 Millionen Euro Schaden beim Auftreten bewertet, Hauptfehler mit 1 Million Euro.

Mit diesen Eingaben stimmen die Istwerte und Modellresultate noch genauer überein. Abbildung 91 zeigt die Fehlerzahlen; fehlende Istwerte im Diagramm waren nicht verfügbar. Die Zahlenwerte sind vertraulich und darum nicht dargestellt. Fehlerzahlen des Systemtest mit internen Fahrzeugtests und externe Fahrzeugtests

durch den Kunden werden nahezu exakt getroffen; die Fehlerzahl im Systemtest weicht um 0,3 dB, in externen Tests um 0,0 dB ab. Für das Codereview ist der Istwert die Zahl der Änderungsaufträge, die nicht direkt mit der Fehlerzahl vergleichbar ist.

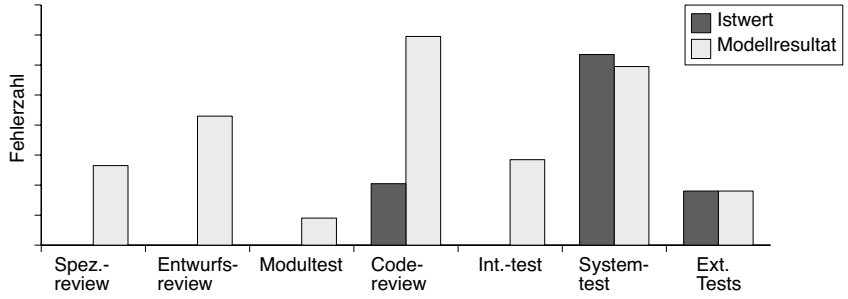

Abb. 91: Fehlerzahlen im Vergleich

Die Modellresultate für den Behebungsaufwand der späten Fehler weichen um nur 0,4 dB ab. Die Verteilung auf die Korrektur und die Prüfungen der Korrektur stimmt gut mit den Istwerten überein (Abbildung 92).

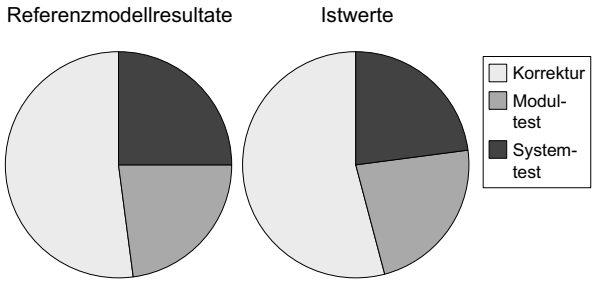

Abb. 92: Aufwandsverteilung auf Korrektur und Prüfung der Korrektur

8.6 Bewertung der Validierung

Ich diskutiere zuerst die Resultate der Validierung (Abschnitt 8.6.1) und bewerte dann ihre Aussagekraft (Abschnitt 8.6.2). Folgerungen enthält Abschnitt 8.6.3.

8.6.1 Resultate der Validierung

Die Validierung mit Industrieprojekten zeigt, dass CoBe fähig ist, umfangreiche, iterativ und parallel ablaufende Projekte zu beschreiben, die unterschiedlich sicherheitskritische und sich ändernde oder stabile Anforderungen haben. Das Modell spiegelt auch die Entwicklung von Systemsoftware gut wieder. Die Kalibrierungsparameter für Aufwand, Dauer und Fehlerzahl, und zusätzlich die Verteilung auf die Fehlerart wurden angepasst. Die dazu notwendigen Daten werden häufiger als andere Metri-

ken erfasst und archiviert (Brodman und Johnson, 1996; Fink und Hampp, 2005; Kasunic, 2006).

Die folgenden Modellresultate stimmen gut mit Istwerten überein, weil sie weniger als 2 dB abweichen:

- Der Korrekturaufwand pro Fehler (verfügbar in Projekt 1 und 2),
- der Prüfaufwand pro Fehler nach Korrektur (verfügbar in Projekt 2),
- die Fehlerzahlen aus späten Tests (verfügbar in Projekt 2); mit Einschränkung, d.h. nur im Referenzmodell, die Fehlerzahlen aus Kundentests und dem Einsatz (verfügbar in Projekt 1 und 2) und intensiven Tests (verfügbar in Projekt 1),
- die Zahl der Fehler pro Reviewsitzung in Codereviews (verfügbar in Projekt 1) und die Fehlerzahl insgesamt der Codereviews (verfügbar in Projekt 2, durch Befragte),
- der Aufwand, Dauer und Mitarbeiterzahl für späte Tests (verfügbar in Projekt 1 und 2),
- der Aufwand für Reviews (verfügbar in Projekt 2),
- der Aufwand für die Wiederholung von Prüfungen und den Korrektur- und Prüfaufwand für späte Fehler (beide verfügbar in Projekt 2).

Insgesamt wurden die Modellresultate von den Beteiligten als plausibel bewertet. Probleme zeigten sich bei der Abbildung intensiver Erprobungen und Vorablieferungen des Produkts. Dies lässt sich dadurch begründen, dass der Feldtest mit der Fehlerentdeckungsquote aus Jones (1996) einem solchen intensiven Probebetrieb nicht entspricht. Jones (2007) diskutiert diesen Punkt und erklärt, dass die Intensität unterschiedlich sein kann und sich die Fehlerentdeckungsquote dementsprechend ändert.

Für die Testwiederholung und den dazu notwendigen Umfangs- und Aufwandsanteil gibt es kaum Erfahrungswerte. Dies führt zu Abweichungen der Kosten für den Modultest und für den Integrationstest bei iterativem Vorgehen und bei gezielter Prüfwiederholung. Die Modelleingaben waren mehr oder weniger frei gewählt und enthalten eine große Unsicherheit.

8.6.2 Gültigkeit der Validierungsresultate

Die externe und die interne Validität der Validierung ist eingeschränkt. Die Ursachen für diese Einschränkungen sind:

- Fehlerzahlen aus den frühen Phasen, d.h. aus den Reviews, waren in beiden Projekten nicht vollständig verfügbar. Für das Industrieprojekt 1 und seine Subsystem-Projekte konnte aber die Zahl der Fehler pro Sitzung erfragt werden. Problematisch an dieser Metrik ist, dass sie durch den Umfang und die Merkmale der Sitzung beeinflusst wird; über den Umfang des Dokuments standen aber keine Istwerte zur Verfügung.

- Es sind nur wenig Projekte untersucht worden. Die Projekte wurden nach Abschluss betrachtet. Somit handelt es sich um zwei Fallstudien. Die Resultate können zufällig gut mit den Istwerten übereinstimmen. Im Idealfall wären Istwerte aus mehreren Projekten in ähnlicher oder der gleichen Umgebung, aber mit unterschiedlichem Prüfprozess, verfügbar. Dann könnte der Nutzen validiert werden. Aussagen über die Streuung wären möglich.
- Es erfolgte keine Prognose, sondern eine nachträgliche Bewertung.

Ich bewerte die Industrievalidierung aber trotzdem als aussagekräftig und erfolgreich. Die Kritikpunkte können nicht vollständig entkräftet werden; sie können aber entschärft werden, wenn die Resultate der studentischen Projekte einbezogen werden:

- Auch wenn wenig Projekte untersucht wurden, so handelt es sich um sehr unterschiedliche Projekte (Abbildung 93). Da CoBe für so unterschiedliche Projekte gute Resultate erbringt, kann vorsichtig auf eine gute Verallgemeinerbarkeit geschlossen werden.

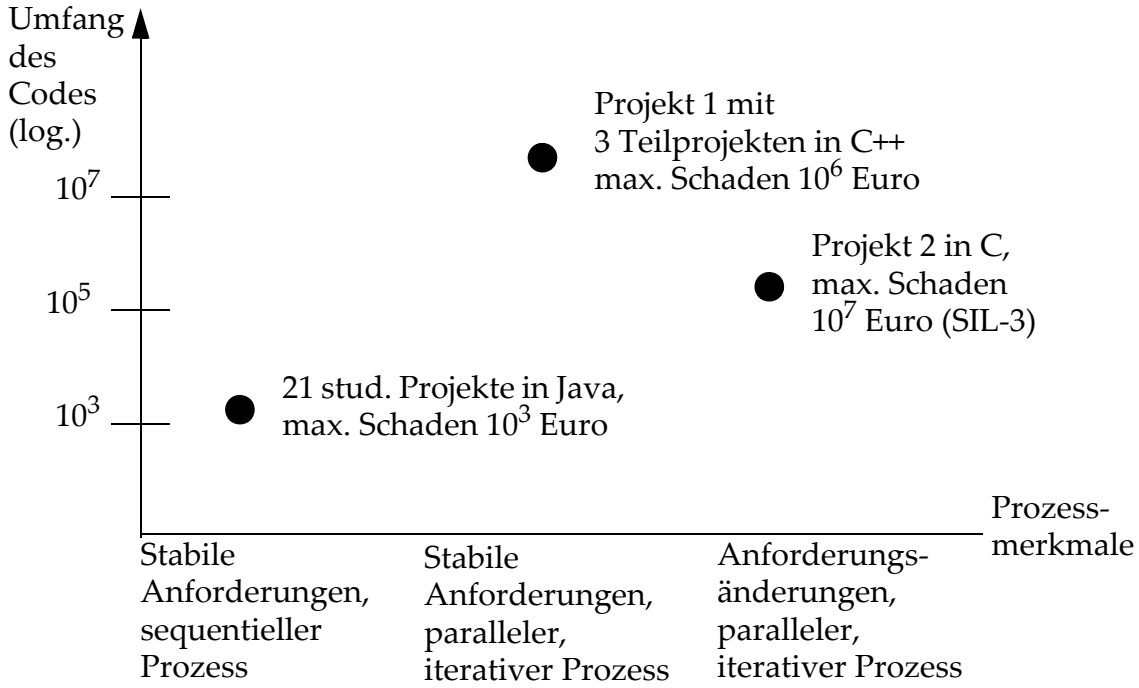

Abb. 93: Überblick über die Projekte für die Validierung

- Der Nutzen des Spezifikationsreviews konnte durch Vergleich der Kosten mit unterschiedlichen Prüfparametern gezeigt werden. Der Nutzen anderer Prüfungen konnte nicht direkt durch einen Vergleich der Kosten durch unterschiedliche Prüfprozesse und Prüfparameter gezeigt und validiert werden. Die Projekte für die Validierung unterschieden sich aber deutlich in den Prüfungen, beispielsweise bei

der Zahl der Gutachter oder bei den Testparametern. Die Modellresultate stimmen gut mit den Istwerten überein. Darum können diejenigen Zusammenhänge, deren Eingabeparameter sich für die Projekte unterscheiden, bestätigt werden. Tabelle 103 zeigt, welche Istwerte und welche Unterschiede im Prüfprozess durch Istwerte geprüft wurden.

Unterschied des Prüfprozesses	Istwerte aus Projekt	Prüfung durch Vergleich mit
Codereview mit einem oder vielen Gutachtern	Industrieprojekte 1 und 2	Fehlerzahl pro Sitzung und Fehlerzahl
Oberflächliche Abnahme oder intensive Erprobung	Stud. Projekte, Industrieprojekte 1 und 2	Fehlerzahl
Systemtest mit Black-Box-Test und Anweisungsüberdeckung, mit intensivem Black-Box-Test, mit Black-Box-Test und Zweigüberdeckung	Stud. Projekte, Industrieprojekte 1 und 2	Testaufwand, Fehlerzahl und Korrekturaufwand
Korrektur oder Korrektur mit Prüfprozess	Stud. Projekte, Industrieprojekte 1 und 2	Aufwand pro Fehler und Gesamtaufwand für Fehler
Mehr oder weniger intensive Spezifikationsreviews mit vielen Gutachtern	Studentische Projekte	Fehlerzahlen und Korrekturaufwand nach Spezifikationsreview und Systemtest

Tabelle 103: Kosten und Nutzen geprüft durch Istwerte

- Die Kreuzvalidierung mit den studentischen Projekten kann eine Prognose für Industrieprojekte nicht ersetzen, zeigt aber, dass die Ungenauigkeit größer wird. Da CoBe für die Industrieprojekte eine höhere Genauigkeit als für die studentischen Projekte in der Diagnose zeigt, kann gefolgert werden, dass auch die Prognose von Industrieprojekten genauer als die Prognose studentischer Projekte wird.

Ein weiteres mögliches Problem der Gültigkeit dieser Validierung ist, dass das Modell für die studentischen Projekte geändert wurde. Für die Industrieprojekte wurden Modellvarianten für nicht verfügbare Eingaben erstellt. CoBe wurde für alle Projekte kalibriert.

- Die Kalibrierung verändert nicht die Zusammenhänge und deren Parameter; sie verändert die Resultate gleichmäßig. Die Abbildungen 39 und 40 (Abschnitt 6.7) zeigen dies deutlich. Wird beispielsweise die Fehlerzahl kalibriert, dann ändert dies nichts an Zusammenhängen der Fehlerentdeckung und anfallenden und entfallenden Fehlerkosten. Die Kalibrierungsparameter können aus historischen Daten berechnet werden. Die Voraussetzung, um gültige Aussagen mit CoBe zu treffen, ist also erfüllbar.

- Die beiden Modellverbesserungen, die nach der ersten Erprobung von CoBe mit studentischen Projekten erfolgten, bedrohen die Gültigkeit nicht: Sie basieren auf empirischen Untersuchungen und nicht auf den Istwerten, mit denen verglichen wurde. Nach der Modellverbesserung wurden die Modellresultate mit anderen Istwerten verglichen: Vor der Modelländerung wurden Mittelwerte der Projekte verwendet. Im Gegensatz dazu wurden nach der Modelländerung einzelne Projekte des Praktikums analysiert und für den Vergleich verwendet.
- Die Erweiterungen der Modellversion 1 von CoBe zur Modellversion 2 (Abschnitt 7.1) mit detaillierten Tests auf allen Integrationsebenen, der Codeanalyse und dem Korrekturprüfprozess ändern nichts an bestehenden Modellzusammenhängen. Sie bedrohen darum nicht die Aussagen der durchgeführten Erprobung des Modells. Da für alle Tests die gleichen Zusammenhänge modelliert werden, werden diese Zusammenhänge sogar zusätzlich bestätigt.
- Kritisch für die Validierung mit den Industrieprojekten sind Modelleingaben, die nicht verfügbar waren. Dabei handelt es sich um Kalibrierungsparameter und Prüfparameter. Die Lösung mit Modellvarianten für unsichere Modelleingaben schlage ich auch für den Praxiseinsatz vor. Die Prüfparameter sind nur in der nachträglichen Validierung nicht bekannt, da beim Modelleinsatz die Prüfparameter vorgegeben werden. Diese Unsicherheit ist darum ein spezielles Merkmal der Diagnose, also des nachträglichen Modelleinsatzes. Die Validierung zeigt zwei Parameter, die in bestimmten Situationen zusätzlich durch Erfahrungswerte angepasst werden sollten: Die Intensität des Feldtests ist unklar. Der Aufwand für die Wiederholung der Tests ist unsicher. Wie für die Kalibrierungsparameter können diese Werte aber aus Archivdaten ähnlicher Projekte erhoben werden.

8.6.3 Folgerungen

Je genauer die Eingaben gesetzt werden können, desto genauer werden die Resultate. Für eine erste Kalibrierung sind vier Faktoren (Aufwand, Dauer, Fehler, Fehlerart) ausreichend, die mit Archivdaten belegt werden können. Zusätzlich können weitere Eingaben mit Archivdaten belegt werden. Dazu gehört etwa der Wiederholungsanteil und -aufwand in Tests. Die Modellresultate sind ähnlich genau wie die Resultate anderer algorithmischer Kostenschätzverfahren. CoBe ergänzt also die Kostenschätzung um eine Bottom-up-Kostenschätzung für Prüfungen, Korrekturen und Wiederholung der Prüfungen. Vor allem aber macht das Modell den Nutzen der Prüfungen sichtbar. Dieser Nutzen ist in der Realität nur durch den Vergleich von Projekten mit unterschiedlichen Prüfungen messbar, weil dazu die Kostendifferenz durch unterschiedliche Prüfungen betrachtet werden muss.

Die Validierung zeigt, dass CoBe im Grundsatz allgemein eingesetzt werden kann. Dazu müssen aber die Kalibrierungsparameter mit Archivdaten ähnlicher Projekte, z.B. der Organisation, belegt werden. Mit diesen Voraussetzungen sind die Resultate, die in der Validierung berechnet werden, für ganz unterschiedliche Projekte und Prüfungen ausreichend genau. CoBe enthält einen umfangreichen Prüfprozess. Die in der

Praxis typisch eingesetzten Prüfungen werden abgedeckt (Liggesmeyer, 2002; Siegwart, 2004; Jones, 1996).

Die Validierung bestätigt, dass die Prüfparameter wichtige Modelleingaben sind. Einzig für unterschiedliche Arten des Feldtests war keine ausreichende empirische Basis für ein solches Modell verfügbar. Es zeigt sich in der Validierung, dass unterschiedlich intensive Feldtests eine Rolle spielen. Beim Modelleinsatz sollten darum Erfahrungswerte zur Verfügung stehen. Falls solche Werte nicht vorhanden sind, zeigt die Validierung, dass unsichere Eingaben durch Modellvarianten dargestellt werden können.

Das iterative Vorgehen für den Modelleinsatz wurde bereits während der Validierung deutlich: Wenn Eingaben unsicher sind oder Ausgaben stark abweichen, dann wird deutlich, dass Informationen über das Projekt fehlen. Damit wird eine gezielte Analyse möglich, die zu einem besseren Prozessverständnis führt. Dadurch kann das Modell genauer angepasst und verbessert werden. Die Analyse zeigt, welche Annahmen über den Prozess oder im Modell nicht haltbar sind. Sie zeigt, ob und wie das Modell erweitert werden muss. Das Modell macht nicht verfügbare Metriken sichtbar und ergänzt somit vorhandene Daten.

8.7 Modellverhalten und Modelleinsatz

Die Validierung zeigt, dass CoBe reale Projekte ausreichend genau widerspiegelt, um Aussagen über Kosten und Nutzen von Prüfungen zu treffen; das Modell kann also die Realität für diesen Aspekt diagnostisch oder prognostisch abbilden. Die Modellziele sind (Abschnitt 3.2):

- Mit dem Modell sollen die Auswirkungen von konkreten und detaillierten Entscheidungen über Prüfungen gezeigt werden können. Dazu gehören Kosten und Nutzen im Projekt, aber auch die langfristigen Auswirkungen, beispielsweise durch Fehlerfolgekosten.
- Mit dem Modell sollen Auswirkungen von Entscheidungen über Prüfungen, die in realen Projekten getroffen wurden, diagnostiziert, d.h. nachträglich dargestellt werden können, es soll Unterschiede zwischen Prozessen deutlich machen, ihre Auswirkungen im Projekt und ihre langfristigen Auswirkungen darstellen können.
- Das Modell soll erlauben, die Kosten mit dem Nutzen zu vergleichen, um Kosten und Nutzen gegeneinander abzuwägen.
- Das Modell soll ermöglichen, den Prüfprozess und die Prüfparameter bezüglich der Kosten und des Nutzens zu optimieren, um die Gesamtkosten zu minimieren.
- Das Modell soll die Planung eines Projekts unterstützen und darum die Auswirkungen von Entscheidungen auf der dazu passenden Abstraktionsebene und als Planungsmetriken darzustellen.

Für den Modelleinsatz schlage ich ein Vorgehen vor, das sich an GQM (Basili und Rombach, 1988) orientiert (Abbildung 94): Im ersten Schritt wird die Frage formuliert, dann werden diejenigen Modellausgaben ausgewählt, mit denen die Frage quantitativ beantwortet werden kann. Im nächsten Schritt wird das Modell kalibriert, dazu werden die Kalibrierungsparameter für Dauer, Aufwand, Fehlerzahlen gesetzt. Falls notwendig, werden weitere Parameter angepasst. Dann werden die anderen Modelleingaben für die Prozess- und Produktmerkmale und für die Prüfungen gesetzt. Für den Vergleich unterschiedlicher Prozesse können dabei auch mehrere Modellinstanzen verwendet werden. Die Modellresultate werden gesammelt und dargestellt; sie müssen interpretiert werden.

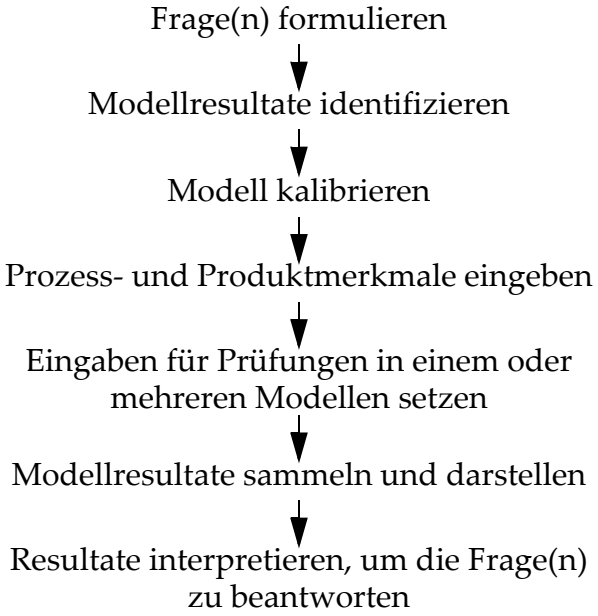

Abb. 94: Schritte beim Modelleinsatz

Im Folgenden zeige ich die Einsatzmöglichkeiten des Modells mit einem fiktiven Projekt und mit Daten aus einem Bericht über Prozessverbesserungen (Haley et al., 1995). Dazu verwende ich fünf Fragen als Beispiele:

1. Wie hoch sind die Kosten, d.h. Dauer, Aufwand und Personalbedarf für einen Prüfprozess, der ausschließlich auf Tests basiert? Wie hoch sind die langfristigen Kosten bei diesem Vorgehen?
2. Wie viel kostet, wieviel nützt das Entwurfsreview bei einem solchen Prüfprozess? Wie wirkt sich das Review im Projekt und wie wirkt sich das Review langfristig in der Wartung und im Einsatz aus? Sind fünf Gutachter notwendig, oder reichen zwei Gutachter aus?
3. Wie wirken sich Prozessverbesserungen mit Reviews aus, wie sie beispielsweise von CMM gefordert werden?

4. Sollen die Modultests verbessert oder Codereviews eingeführt werden?
5. Was kosten Regressionstests, wenn sie von Hand durchgeführt werden müssen? Sind solche Tests in einem Projekt mit beschränkter Dauer und beschränktem Aufwand machbar? Was bringt eine Testautomatisierung?

Tabelle 104 gibt einen Überblick über die Modellziele und das Beispiel, mit dem der Modelleinsatz für dieses Ziel illustriert wird.

Modellziel	Beispiel
Auswirkungen von Entscheidungen im Projekt darstellen	Frage 2: Entwurfsreviews mit Varianten (Abschnitt 8.7.2) Frage 5: Testautomatisierung (Abschnitt 8.7.5)
Langfristige Auswirkungen der Entscheidungen darstellen	Frage 2: Entwurfsreviews mit Varianten (Abschnitt 8.7.3)
Auswirkungen darstellen und Unterschiede deutlich machen	Frage 3: Verbesserung mit Reviews (Abschnitt 8.7.4) Frage 5: Verbesserungen durch Testautomatisierung (Abschnitt 8.7.5)
Kosten und Nutzen vergleichen, abwägen und optimieren	Frage 2: Entwurfsreview mit Varianten (Abschnitt 8.7.3) Frage 4: Codereviews oder Modultest (Abschnitt 8.7.6)
Planungsmetriken darstellen	Frage 1: Kosten von Tests (Abschnitt 8.7.1) Frage 2: Kosten der Entwurfsreviews (Abschnitt 8.7.2) Frage 5: Kosten für Regressionstest (Abschnitt 8.7.5)

Tabelle 104: Überblick über die Modellziele und Beispiele für den Modelleinsatz

8.7.1 Ein fiktives Beispielprojekt

Das Beispielprojekt hat einen Umfang von 200 Function Points und soll in Java realisiert werden. Alle Parameter des Modells werden auf ihre Normalwerte gesetzt. COCOMO II berechnet für das Projekt 42 Entwicklermonate Aufwand, 14 Monate Dauer (rund 280 Arbeitstage) und 3 Mitarbeiter. Als Ausgangspunkt für den Prüfprozess werden typische, normale Tests (Modultest, Integrationstest, Systemtest) durchgeführt und von Hand wiederholt. Es findet ein Feldtest statt. In der Wartung werden Modultest und Systemtest gezielt für die Korrektur wiederholt, der Integrationstest wird als Regressionstest durchgeführt. Die Fehlerschwere ist im Beispielprojekt durch den möglichen Schaden im Einsatz definiert. Dabei handelt es sich um eine Anwendung, bei der ein kritischer Fehler einen Schaden von 10 000 Euro verursachen kann; die Hauptfehler verteilen sich zu gleichen Teilen auf einen Schaden von 1000 Euro und 100 Euro, Nebenfehler führen zu Komfortverlusten. Etwa 10 % der Fehler treten nicht auf und werden auch nicht gemeldet. Die Software wird nach der Auslieferung häufig verwendet. Kritische Fehler werden sofort korrigiert; die Software wird erst wieder eingesetzt, wenn der Fehler behoben ist. Bei anderen Fehlern wird die Software etwa zehnmal verwendet, bis ein Fehler korrigiert wird.

Die Modellresultate für die Qualitätskosten der Tests und Korrekturen zeigt Tabelle 105 mit dem Aufwand in Entwicklerstunden, der Dauer in Arbeitstagen und dem Personalbedarf (gerundet). Die Resultate basieren auf den Fehlerzahlen, die CoBe berechnet (Tabelle 106). Insgesamt fallen 193 Arbeitstage und rund 2702 Entwicklerstunden für die Tests und die Korrektur im Projekt an. Die Fehlerfolge- und Wartungskosten sind im Vergleich zu den Projektkosten und zu den Projekt-Qualitätskosten um Größenordnungen höher. In der Wartung macht die Prüfung nach der Korrektur den wesentlichen Anteil aus (rund 15 000 Entwicklerstunden). Die Fehlerfolgekosten liegen bei rund 657 000 Euro.

Qualitätskosten	Aufwand (Eh)	Dauer (Tage)	Mitarbeiter Prüfung	Mitarbeiter Korrektur
Modultest, Korrektur	325	20	2	0[a]
Integr.-test, Korrektur	685	56	3	2
Systemtest, Korrektur	972	72	3	2
Korrektur Feldtest	720	45	-	2
Wartung, nur Korrektur	3611			
Wartung, nur Retest	11 388			

Tabelle 105: Modellresultate für Qualitätskosten

a. Im Modultest prüfen und korrigieren die gleichen Entwickler

Zahl der Fehler	Insgesamt	Ausgeliefert	Gemeldet und wirksam
	642	290	261

Tabelle 106: Modellresultate für Fehlerzahlen

Bewertung

Diese Planungsmetriken ergänzen die Kostenschätzung, beispielsweise die Top-down-Schätzung mit COCOMO II. Sie zeigen, dass der Zeitplan mit diesen Prüfungen eng ist, wenn strikt sequentiell vorgegangen wird, weil bereits Prüfung und Korrektur rund zwei Drittel der Projektdauer benötigten. Der Vergleich des Wartungsaufwands mit dem Aufwand für Prüfungen und Korrektur im Projekt deutet eine nicht optimale Situation an.

8.7.2 Kosten und Nutzen des Entwurfsreviews

Die Demonstration von Kosten und Nutzen zeige ich am Beispiel des Entwurfsreviews für zwei Fälle. Im ersten Fall wird ein formales Review mit fünf Gutachtern durchgeführt, im zweiten Fall erfolgt das Review mit zwei Gutachtern. CoBe gibt diese beiden Fälle nicht fest vor; es erlaubt, eine andere Gutachterzahl, Gutachter-

kompetenz oder Vorbereitungsintensität zu wählen. In beiden Fällen erfolgt die Vorbereitung gründlich. Die Prüfung wird, falls notwendig, auf mehrere Sitzungen verteilt, die von einem Moderator betreut werden. CoBe zeigt, welche Kosten entstehen. Tabelle 107 zeigt Modellresultate für die Planungsmetriken eines Review mit 5 Gutachtern, Tabelle 108 das Review mit 2 Gutachtern. Das Projekt verzögert sich ohne eine parallele Organisation des Entwurfsreviews und der Korrektur um etwa zwei Wochen.

Entwurfsreview mit 5 Gutachtern		Aufwand (Eh)	Dauer (Tage)	Mitarbeiter Prüfung	Mitarbeiter Korrektur
Review-kosten	Entwurfsreview	75	9	7,0	-
	Korrektur	269	55	-	0,6
Weitere Qualitäts-kosten[a]	Modultest	308	19	2,0	-[b]
	Integrationstest	573	49	2,9	2,0
	Systemtest	790	61	2,9	2,0
	Feldtest[c]	517	32		2,0
Entfallende Kosten (Nutzen)[a]	Modultest	17	1	0,2	2,0
	Integr.-test	113	7	1,8	2,0
	Systemtest	182	11	0,9	2,0
	Feldtest[c]	203	13	-	2,0

Tabelle 107: Modellresultate für ein Entwurfsreviews mit 5 Gutachtern

a. Die Prüfungen sind einschließlich Korrektur dargestellt.
b. Im Modultest prüfen und korrigieren die gleichen Entwickler
c. Nur Korrektur nach Feldtest

Abbildung 95 vergleicht den Nutzen der Reviewvarianten. Mit 2 Gutachtern ist das Review etwas günstiger (221 Entwicklerstunden Aufwand und 47 Tage einschließlich Korrektur), dafür ist der Nutzen geringer, der mit 5 Gutachtern im Projekt 515 Entwicklerstunden beträgt. Berücksichtigt man den Aufwand für das Review und die Korrektur, dann werden mit 5 Gutachtern 171 Entwicklerstunden eingespart. Mit zwei Gutachter (Tabelle 108) werden 135 Entwicklerstunden eingespart.

8.7.3 Langfristiger Nutzen des Entwurfsreviews

Deutlicher wird der Nutzen des Reviews, wenn die langfristigen Auswirkungen nach Auslieferung, d.h. die entfallenden Wartungskosten und die entfallenden Fehlerfolgekosten für Kunden und Benutzer beim Einsatz, betrachtet werden (Tabelle 109). Personalkosten sind mit 200 000 Euro pro Entwicklerjahr gewichtet.

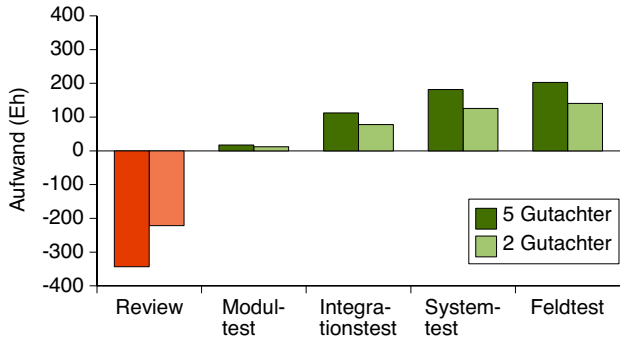

Abb. 95: Nutzen des Entwurfsreviews mit 5 oder 2 Gutachtern

Entwurfsreview mit 2 Gutachtern		Aufwand (Eh)	Dauer (Tage)	Mitarbeiter Prüfung	Mitarbeiter Korrektur
Review-kosten	Entwurfsreview	35	9	4,0	0,0
	Korrektur	186	38	0,0	0,6
Weitere Qualitäts-kosten[a]	Modultest	313	19	2,0	-[b]
	Integrationstest	607	51	2,9	2,0
	Systemtest	846	64	2,9	2,0
	Feldtest[c]	579	36		2,0
Nutzen[a]	Modultest	12	1	0,2	2,0
	Integrationstest	78	5	1,8	2,0
	Systemtest	126	8	0,9	2,0
	Feldtest[c]	141	9		2,0

Tabelle 108: Modellresultate für ein Entwurfsreviews mit 2 Gutachtern

a. Die Prüfungen sind einschließlich Korrektur dargestellt.
b. Im Modultest prüfen und korrigieren die gleichen Entwickler.
c. Nur Korrektur nach Feldtest

Die Gesamt-Qualitätskosten in Euro machen sichtbar, dass das Review nützlich ist. Für Tabelle 110 werden Personalkosten mit 200 000 Euro pro Entwicklerjahr berechnet. Durch das Entwurfsreview werden Personalkosten während des Projekts eingespart. Der größte Teil des Nutzens wird aber durch entfallende Personalkosten in der Wartung erreicht. Fall 1 betrachtet Personalkosten und Fehlerfolgekosten. Personalkosten in der Wartung und Fehlerfolgekosten sinken bei einem Review mit 5 Gutachtern auf etwa zwei Drittel im Vergleich zu einem Projekt ohne Entwurfsreview (Fall 1 in Tabelle 110, Abbildung 96, links).

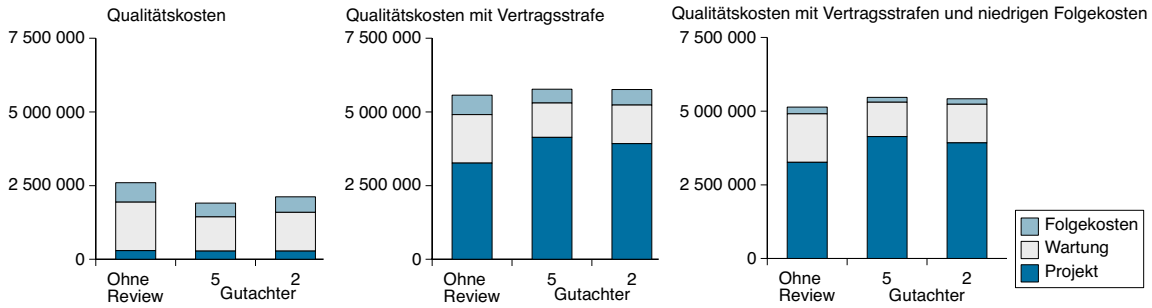

Abb. 96: Qualitätskosten im Vergleich

Langfristige Auswirkungen	5 Gutachter	2 Gutachter
Entfallende Wartung (nur Korrektur) in Eh	1080	748
Entfallende Wartung (Korrektur und Test) in Eh	4377	3033
Entfallende Personalkosten für die Wartung in Euro	479 955	332 578
Entfallende Folgekosten beim Einsatz in Euro	190 236	131 821

Tabelle 109: Langfristige Auswirkungen

Qualitätskosten in Euro im Vergleich	Ohne Review	5 Gutachter	2 Gutachter
Personalkosten für Prüfung und Korrektur im Projekt	296 353	277 506	281 470
Personalkosten für Prüfung und Korrektur in der Wartung	1 644 656	1 164 701	1 312 078
Summe Personalkosten	1 941 009	1 442 207	1 593 548
Fehlerfolgekosten beim Einsatz der Software	657 023	466 787	525 202
Niedrige Fehlerfolgekosten beim Einsatz	228 734	162 506	191 196
Kosten für Dauer bei Vertragsstrafe im Projekt	2 970 416	3 861 552	3 643 188
Fall 1: Summe Personal- und Fehlerfolgekosten	2 598 032	1 908 994	2 118 750
Fall 1: Davon langfristige Kosten (Wartung und Fehlerfolgekosten)	2 301 679	1 631 488	1 837 280
Fall 2: Summe Personalkosten, Fehlerfolgekosten, Vertragsstrafen	5 568 448	5 770 546	5 761 938
Fall 2: Summe Personalkosten, niedrige Fehlerfolgekosten, Vertragsstrafen	5 140 159	5 466 265	5 427 932

Tabelle 110: Qualitätskosten im Vergleich

Spielt aber die Dauer eine wichtige Rolle, beispielsweise weil Vertragsstrafen bei Verzögerungen drohen (Fall 2), dann macht CoBe deutlich, dass – bei ungeschickter Organisation der Reviews und einem strikt sequentiellen Vorgehen – das Projekt teurer wird, vor allem mit 5 Gutachtern. Tabelle 110 zeigt die Qualitätskosten für diesen Fall 2. Für die monetäre Gewichtung der Dauer werden 3 % der gesamten Projektkosten als Vertragsstrafe pro Tag gesetzt. Der langfristige Nutzen gleicht die höheren Projektkosten nahezu aus (Fall 2 in Tabelle 110, Abbildung 96, mitte). Sind die Fehlerfolgekosten niedrig, verursachen also Fehler maximal einen Schaden von 1000 Euro und wird das Produkt selten eingesetzt, dann verändert sich die Aussage von CoBe: Die Dauer wird zum bestimmenden Einfluss. (Fall 3 in Tabelle 110 und Abbildung 96, rechts).

Bewertung

Die Modellresultate machen sichtbar, wann und in welchem Maß Nutzen durch Prüfungen erreicht wird. Mit dem Modell können auch kleine, nicht offensichtliche Verbesserungen dargestellt werden. Der Nutzen ist in realen Projekten nur durch den Vergleich der Kosten sichtbar, weil er durch entfallende Fehler entsteht. In vielen Fällen sind Metriken, die für einen solchen Vergleich nötig sind, nicht verfügbar. Selbst wenn sie verfügbar sind, dann ist der Vergleich schwierig, weil andere Unterschiede der Projekte den Effekt der Prüfungen überlagern können.

Die Modellresultate zeigen den langfristigen Nutzen und machen Unterschiede, die durch unterschiedliche Prüfparameter verursacht werden, deutlich. Die Modellresultate machen deutlich, welche Kosten für die Entscheidungen über Prüfungen relevant sind: Fehlerfolgekosten beim Einsatz, Projektkosten oder Wartungskosten prägen die Gesamt-Qualitätskosten und damit die Gesamtkosten. Nutzen und Kosten werden durch die Abbildung auf Geldwerte vergleichbar; dieser Vergleich ist aber durch die Gewichtung der Basismetriken, also der Dauer, des Aufwands, und der Fehlerkosten, geprägt. Über Prüfungen kann rationaler entschieden werden, weil die Gewichtung dieser Metriken durch plausible Erfahrungswerte möglich ist. Das Beispiel macht deutlich, dass es keinen Standard-Prüfprozess gibt, der in allen Situationen optimal ist; die konkrete Situation bestimmt den Nutzen.

Prinzipiell ist natürlich möglich, die Metriken mit Geldwerten so zu gewichten, dass eine gewollte Entscheidung begründet wird. Insofern kann CoBe unterlaufen werden. Die Gewichtung ist aber sichtbar und mit Erfahrungswerten belegt.

8.7.4 Prozessverbesserung durch Reviews

Mit dem Modell können Effekte von Prozessverbesserungen nachträglich dargestellt werden. Dazu wird der Effekt von formalen Reviews betrachtet; Reviews werden beispielsweise von CMMI (CMMI Product Team, 2002) gefordert. Die Modellresultate für Spezifikations-, Entwurfs- und Codereview mit jeweils 5 Gutachtern und mit den gleichen Prozess- und Produktmerkmalen wie in Abschnitt 8.7.1 zeigen, dass die Korrekturkosten auf rund zwei Drittel gesenkt werden können (Tabelle 111).

Haley et al. (1995) berichten einen ähnlich hohen Nutzen dieser Reviews, die im Rahmen der Prozessverbesserung mit CMM eingeführt wurden. Zuerst wurden formale Entwurfs- und Codereviews, dann Anforderungsreviews etabliert. Es zeigte sich im Verlauf der Prozessverbesserung, dass der Anteil für Nacharbeit an den Projektkosten von rund 40 % auf 20 % gesenkt werden konnte.

Korrekturaufwand (Eh)	Ohne Review	5 Gutachter	2 Gutachter
Spezifikationsreview	-	123	85
Entwurfsreviewkorrektur	-	233	169
Codereviewkorrektur	-	251	220
Modultestkorrektur	141	123	129
Integrationstestkorrektur	506	151	233
Systemtestkorrektur	818	242	375
Feldtestkorrektur	720	197	317
Summe (Eh)	2185	1322	1528
Summe (EM)	14	9	10

Tabelle 111: Einfluss von Reviews auf den Korrekturaufwand

Ich beziehe die Korrekturkosten, die CoBe berechnet, auf den Projektaufwand, den COCOMO II berechnet, damit die Anteile in Prozent direkt verglichen werden können. COCOMO II berechnet 42 Entwicklermonate Aufwand für das Projekt. CoBe berechnet 34 % von diesen 42 Entwicklermonaten für die Korrektur, falls keine Reviews durchgeführt werden. Mit Reviews mit 5 Gutachtern sinkt dieser Anteil in CoBe auf 21 %. Durch Reviews sinkt also der Anteil der Korrekturkosten von 34 % auf 21 % in CoBe, von 40 % auf 20 % im Bericht von Haley et al. (1995) durch CMMI. CMMI wirkt also etwas stärker, vermutlich weil Verbesserungen der Planung, Schulung und Anforderungsdefinition zusätzlich wirken.

Bewertung

Das Modell demonstriert die Verbesserung und macht sie sichtbar, ohne dass reale Projekte in großem Umfang durchgeführt werden müssen. Das Beispiel zeigt aber auch eine wichtige Grenze des Modells, weil die Einführung der Reviews, der Trainingsaufwand und die organisatorischen Schwierigkeiten mit Reviews, damit es zu keinen Verzögerungen kommt, nicht dargestellt werden. Das Beispiel zeigt auch, dass CoBe andere Verbesserungen, z.B. der Anforderungsdefinition, der Planung oder der Schulung, nicht direkt darstellt. Diese Verbesserungen durch Fehlervermeidung können nicht direkt dargestellt werden, aber indirekt durch die Kalibrierung einbezogen werden.

8.7.5 Prozessverbesserung durch Testautomatisierung

Auch dieses Beispiel stammt aus dem Bericht von Haley et al. (1995). Im Projekt, über das berichtet wird, erfolgte die Systemintegration inkrementell. Dazu wurde jedes Release mit den zu integrierenden Komponenten und Funktionen geplant, dann entwickelt. Nach jeder Integration sollte ein Regressionstest das Release prüfen, um die Funktionsfähigkeit zu gewährleisten. Da der Test nicht automatisiert war, wurde er vernachlässigt; er war zu teuer.

Das Modell kann dieses Problem darstellen. Von Hand kostet die Wiederholung des Tests rund 75 % des Aufwands der ersten Durchführung, automatisiert 10 % ohne Funktionsänderung, 25 % mit Funktionsänderung. Für den Integrationstest nehme ich an, dass die Korrektur von 5 Fehlern in ein Release kommt. Dies entspricht etwa 8 Integrationsschritten; es wird in etwa einmal pro Woche integriert.

	Ohne Automatisierung	Mit Automatisierung
Aufwand (Eh)	368	159
Dauer (Tage)	51	22

Tabelle 112: Nutzen der Testautomatisierung

Ohne Automatisierung müssen rund 2,5 Entwicklermonate aufgewendet werden, der Test dauert über 2 Monate. Diese Kosten lassen sich um mehr als die Hälfte reduzieren. Dies entspricht den Erfahrungen von Haley et al. (1995).

Bewertung

Das Modell kann die Auswirkungen einzelner, konkreter Verbesserungsmaßnahmen zeigen. Die Modellresultate stimmen mit Erfahrungen aus Prozessverbesserungsmaßnahmen überein.

8.7.6 Codereview durchführen oder Modultest verbessern

Die Auswahl zwischen zwei Verbesserungen zeige ich mit der folgenden Frage: Soll ein Modultest, der als Black-Box-Test durchgeführt wird, durch einen Glass-Box-Test mit 80 % Anweisungs- und Zweigüberdeckung verbessert werden? Oder soll der gleiche Aufwand in Codereviews investiert werden soll? Diese Frage diskutiere ich für das fiktive Projekt aus Abschnitt 8.7.1. Der Prüfprozess ist mit Spezifikations- und Entwurfsreviews, Modultest, Systemintegrationstest, Systemtest und Feldtest angegeben. Die Reviews finden jeweils mit 5 gründlichen Gutachtern statt. Die Eingaben für alle Tests sind auf den Nominalfall, d.h. Black-Box-Test der Funktionen und Äquivalenzklassen, gesetzt. Alle Prozess- und Produktmerkmale sind auf die gleichen Werte wie in Abschnitt 8.7.1 gesetzt, mit rund 200 Function Points Umfang und einem maximalen Schaden, den ein Fehler beim Auftreten verursachen kann, von 10 000 Euro.

Die Tabellen 113 und 114 zeigen die Modellresultate für diesen Prozess mit den drei unterschiedlichen Varianten. Der gleiche Aufwand, der für den intensiveren Modultest einschließlich der Korrekturen anfällt, wird statt dessen in Codereviews mit 4 Gutachtern investiert. Dies erlaubt, rund 57 % des Codes zu begutachten; die Reviews werden priorisiert, sie konzentrieren sich auf kritischen, risikoreichen Code. In Tabelle 114 sind die Projekt-Qualitätskosten (also Prüfungen, Korrektur, Prüfwiederholung), die Qualitätskosten in der Wartung (Korrektur, Prüfwiederholung) und die Fehlerfolgekosten beim Einsatz des Produkts dargestellt. Die Gesamt-Qualitätskosten bestehen aus den Projekt-Qualitätskosten, den Qualitätskosten der Wartung und den Fehlerfolgekosten.

Kosten für		Aufwand (Eh)	Dauer (Tage)
Modultest (Black-Box-Test) mit normaler Intensität		308	19
Modultest (Black-Box-Test mit normaler Intensität) und Codereview (4 Gutachter, 57 % des Codes)	Modultest	308	19
	Codereview	326	29
	Gesamt	634	48
Modultest (Black-Box-Test mit normaler Intensität) ergänzt um Glass-Box-Test bis 80 % Anweisungs- und Zweigüberdeckung		636	39

Tabelle 113: Kosten der Prüfung und Fehlerbehebung im Vergleich

Qualitätskosten	Projekt	Wartung	Fehler-folgekosten	Gesamt
Modultest (Black-Box-Test) mit normaler Intensität	261 886	851 618	349 241	1 462 745
Modultest (Black-Box-Test mit normaler Intensität) und Codereview (4 Gutachter, 57 % des Codes)	242 983	557 773	222 653	1 023 409
Modultest (Black-Box-Test mit normaler Intensität) ergänzt um Glass-Box-Test bis 80 % Anweisungs- und Zweigüberdeckung	282 661	788 620	318 358	1 389 639

Tabelle 114: Qualitätskosten im Vergleich

Die Modellresultate zeigen, dass der Aufwand für die Prüfung und Korrektur in beiden Fällen etwa gleich ist. Die Codereviews dauern aber länger, falls sie nicht so organisiert werden, dass sie parallel durchgeführt werden. Der Nutzen, d.h. entfallende Personal- und Fehlerfolgekosten, unterscheidet sich deutlich: Mit Codereviews werden bereits im Projekt die Qualitätskosten gesenkt, langfristig sinken die Qualitäts-

kosten noch deutlicher. Mit dem intensiveren Modultest steigen die Qualitätskosten im Projekt, ein geringer Nutzen wird langfristig erreicht.

Am Beispiel des Codereviews wird deutlich, dass die Modellresultate für die Planung des Projekts und der Prüfungen verwendet werden können: CoBe ergibt, dass 6 Reviewsitzungen eingeplant werden müssen, für die ein Gutachter insgesamt 20 Stunden zur Vorbereitung benötigt. Der Aufwand der Korrektur wird mit über einem Entwicklermonat berechnet, die Dauer mit zwei Entwicklern rund 2 Wochen (12 Arbeitstage).

Bewertung

CoBe zeigt den Unterschied zwischen den Alternativen. Dieser direkte Vergleich ist in der Realität kaum möglich, weil dazu vergleichbare Projekte durchgeführt werden müssen, die sich nur in den Prüfungen unterscheiden.

CoBe macht Kosten sichtbar, die sonst nur schwer messbar sind. Dazu gehören Kosten für den Modultest, der häufig von den Entwicklern selbst durchgeführt wird, seine Kosten sind kaum von der Implementierung zu trennen.

CoBe zeigt, dass sich die Dauer der Alternativen unterscheidet – entgegengesetzt zum Aufwand. Dies zeigt, dass dieser Aspekt des Modells für die Entscheidung eine wichtige Rolle spielt und darum nicht verkürzt werden sollte.

Das Beispiel zeigt aber auch die Grenzen von CoBe: Sollen Reviews eingeführt werden, und die Entwickler sind unwillig, Code zu begutachten, aber motivierter, die eher technische Lösung des Glass-Box-Tests durchzuführen, dann müssen die Entwickler von Reviews überzeugt werden. Dies stellt das Modell nicht dar. Der Nutzen durch Schulungseffekte der Codereviews oder durch erhöhtes Vertrauen in das Produkt mit dem intensiveren Test wird nicht deutlich.

CoBe zeigt Kosten und Nutzen von Prüfungen; es kennt aber nur einen bestimmten, abgeschlossenen Handlungsspielraum. Die Resultate beruhen auf Annahmen z.B. über die Reihenfolge von Prüfungen. Das Modell kennt keine Effekte, die sich durch Parallelisierung z.B. von Reviews und der zugehörigen Korrektur ergeben; da das Modell aber einzelne Aktivitäten betrachtet, sind die Grundlagen für eine solche Planung vorhanden.

Kapitel 9

Zusammenfassung und Bewertung

In diesem Kapitel werden die Arbeit und ihre Resultate zusammengefasst. Sie werden im Hinblick auf die Modellziele und den Modelleinsatz bewertet. Darauf folgen Ausblick und Schlussbemerkungen.

9.1 Zusammenfassung

Das Resultat dieser Arbeit ist das quantitative Kosten-Nutzen-Modell CoBe für konkrete Entscheidungen, die in Projekten über Prüfungen und Prüfparameter getroffen werden. CoBe stellt die Wirkung von Entscheidungen über Prüfungen und Prüfparameter als anfallende Kosten und Nutzen als entfallende Kosten dar. Das Modell berücksichtigt Auswirkungen der Entscheidungen im Projekt, in der Wartung und im Einsatz des Produkts. Somit lassen sich die Wirkungen der Entscheidungen vergleichen, demonstrieren und nachträglich für konkrete Projekte diagnostizieren, aber auch während der Planung prognostizieren; die Modellresultate enthalten eine Bottom-up-Kostenschätzung für Prüfung und Fehlerbehebung. CoBe unterstützt somit Projektleiter und QS-Verantwortliche: Sie können die Wirkungen der Entscheidungen direkt aus dem Modell ablesen.

CoBe unterscheidet sich von anderen Modellen, die sich mit Entscheidungen über Prüfungen in Software-Projekten befassen, durch die Abbildung der konkreten Entscheidungen über Prüfparameter. Dazu enthält es ein detailliertes Reviewmodell, ein detailliertes Testmodell und Modelle für die Wiederholung von Prüfungen während des Projekts und in der Wartung. Es berücksichtigt Fehlerfolgekosten, die auf Eingaben über die Verwendung der Software basieren.

Das Modell ist quantitativ und besteht aus einzelnen, überprüfbaren Zusammenhängen. Diese stammen aus anderen Modellen, aus Datensammlungen, Erfahrungsberichten und Experimenten; sie sind empirisch belegt.

CoBe ist in die Planung und in die Prozessverbesserung eingebunden. Die dazu notwendige Kalibrierung erfolgt in Schritten mit definierten Parametern. Die wichtigsten Metriken für die Kalibrierung sind Umfangs- und Aufwandsdaten für das gesamte Projekt und Fehlerzahlen.

9.1.1 Validierung

Die Validierung erfolgte mit Daten, die gezielt für die Validierung erhoben wurden und darum zur Modellbildung nicht zur Verfügung standen. Sie folgt einem Konzept, das sich aus den Schwierigkeiten der Validierung von Entscheidungsmodellen ableitet. Für die Validierung des Modells wurden Daten in studentischen Projekten und in Industrieprojekten erhoben. Diese Projekte decken ein breites Spektrum ab, mit kleinen, studentischen Projekten (typisch 3 Teilnehmer pro Team, 21 Wochen Dauer), mit einem mittelgroßen, sicherheitskritischen Projekt (2,5 Jahre, 14 Mitarbeiter) und mit einem großen Projekt, dessen Subsysteme parallel entwickelt wurden (2 Jahre, 400 Mitarbeiter). Daten aus studentischen Projekten zeigen, dass einzelne Zusammenhänge des Modells gelten. Daten aus den studentischen Projekten und aus der Industrie zeigen, dass das Modell unterschiedlich komplexe und umfangreiche Projekte ausreichend genau darstellen kann. Dazu ist aber eine Kalibrierung notwendig.

Die Zusammenhänge zur Modellbildung waren unterschiedlich gut belegt. Während der Validierung wurden die Zusammenhänge unterschiedlich intensiv validiert. Daraus ergibt sich, welche Modellkomponenten verlässlichere und welche weniger verlässliche Aussagen erlauben:

- Der grundlegende Zusammenhang des Fehlerstrommodells und der Einsparungen durch entfallende Fehler ist empirisch belegt und wurde zusätzlich durch die Validierung belegt.
- Die Korrekturkosten eines Fehlers sind durch umfangreiche empirische Studien und zusätzlich durch die Validierung belegt.
- Die Zusammenhänge von Reviews sind umfangreich untersucht, sie werden durch die Validierung zusätzlich gestützt.
- Die Zusammenhänge im Test sind wenig in der Literatur untersucht, aber in der Validierung einzeln und insgesamt mit studentischen Projekten und mit Industrieprojekten validiert; es fehlen aber breite empirische Daten, vor allem aber quantitative Aussagen zur Term- und Schleifenüberdeckung.
- Kosten zur Testwiederholung werden vom Modell übereinstimmend zur Realität berechnet. Es fehlen aber Erfahrungswerte und eine breite Datenbasis, um die Basiswerte für den Umfang und den Aufwand der Testwiederholung verlässlich zu prognostizieren.
- Eher unklar ist auch die Situation bei Erprobungen oder beim Feldtest, also bei allen Tests, die unter mehr oder weniger realen Einsatzbedingungen ablaufen. Die Literatur enthält einige wenige Daten. Diese Daten zeigen aber eine breite Streuung, abhängig von der Intensität der Erprobung.
- Die Abschätzung der Fehlerfolgekosten ist kaum durch Messungen bestätigt und validiert. Fehlerfolgekosten sind aber ein wesentliches Entscheidungskriterium, so das dieser Aspekt nicht vernachlässigt werden darf.

9.1.2 Aussagen des Modells

Die Sensitivitätsanalyse und die Analyse zur Optimierung ergänzen diese Validierung. Mit diesen Analysen können folgende Aussagen des Modells gezeigt werden:

- Einzelne Prüfparameter prägen Kosten und Nutzen der Prüfungen.
- Ihre Wirkung hängt von der Situation ab, besonders deutlich von den Fehlerfolgekosten, aber auch von Kosten für spätere Auslieferung. Bei den Fehlerfolgekosten stellt CoBe nicht nur einzelne, spektakuläre Schäden dar, sondern auch solche Schäden, die sich über viele Benutzer und Verwendungen der Software aufsummieren.
- Minimale Projektkosten und minimale Qualitätskosten über die Produktlebensdauer können widersprüchliche Ziele sein; die Prüfungen und ihre Parameter sind also ein Kompromiss.

Für typische Situationen zeigt die Sensitivitätsanalyse und die Optimierung:

- Die Kompetenz der Prüfer spielt im Review und im Test die entscheidende Rolle.
- Frühe Reviews lohnen sich immer, weil die Projektkosten höchstens wenig steigen, aber langfristig ein hoher Nutzen erreicht wird. Dazu ist eine vollständige Prüfung mit gründlicher Vorbereitung notwendig.
- In typischen Fällen ist ein gründlicher Black-Box-Systemtest ausreichend; je höher die Fehlerfolgekosten werden, desto nützlicher wird der Glass-Box-Test, wenn langfristige Folgen betrachtet werden. Die Projektkosten steigen mit der Testintensität.

CoBe zeigt, dass es keinen allgemeingültigen optimalen Prüfprozess gibt, sondern dass die individuellen Rahmenbedingungen eines Projekts betrachtet werden müssen. Dabei sind Daten zum Einsatz des Produkts und zum möglichen Schaden durch Fehler wichtig. Das Modell zeigt, dass eine Kalibrierung notwendig ist, und welche Metriken dafür notwendig sind: Produktumfang, Gesamtaufwand, Gesamtdauer und Fehlerzahlen. CoBe enthält quantitative Erfahrungen über Auswirkungen von Prüfungen. Damit enthält das Modell implizit ein Metrikprogramm zur Qualitätsbewertung. Dabei stehen Aufwände der einzelnen Aktivitäten, also der Prüfung, Korrektur und Prüfwiederholung, im Mittelpunkt.

9.2 Bewertung

9.2.1 Modellziele

Abschnitt 8.7 zeigt, dass die Modellziele von CoBe prinzipiell erreicht werden: Auswirkungen von Entscheidungen über Prüfungen können demonstriert werden. Die zur Planung und Optimierung nötigen Informationen, die Planungsmetriken, werden ausgegeben. Kosten und Nutzen werden vergleichbar dargestellt. Die Validierung zeigt, dass mit CoBe Kosten und Nutzen von Prüfungen nachträglich beschrieben

werden können, CoBe kann also diagnostisch eingesetzt werden. Die Prognosefähigkeit des Modells ist durch die Kreuzvalidierung untersucht (Abschnitt 7.6.2).

9.2.2 Verallgemeinerbarkeit

Die Validierung zeigt, dass CoBe prinzipiell für ganz unterschiedliche Projekte geeignet ist (Abschnitt 8.6), aber für eine konkrete Umgebung kalibriert werden muss. Dazu sind Archivdaten notwendig. Diese Metriken stehen häufiger als andere Metriken in der Industrie zur Verfügung (Brodman und Johnson, 1996; Fink und Hampp, 2005; Kasunic, 2006). CoBe basiert auf Function Points, die in der verwendeten Variante nicht direkt für technisch-wissenschaftliche Anwendungen geeignet sind. Die Validierung zeigt, dass Function Points als interne Parameter auch in diesen Anwendungsgebieten eingesetzt werden können.

Die Prüfungen im Modell gehören zu den typischen Prüfungen in Industrieprojekten. Ihre Reihenfolge im Modell ist vorgegeben, CoBe ist also auf eine bestimmte Prüfsequenz eingeschränkt. Einfache, sequentiell ablaufende Projekte mit den Prüfungen oder einem Teil der Prüfungen in der gleichen Reihenfolge können direkt in das Modell abgebildet werden. Für parallele, iterative Prozesse mit Anforderungsänderungen ist eine Modellbildung notwendig. Die Validierung zeigt, dass solche Projekte in das Modell abgebildet werden können. Damit eine andere Prüfreihenfolge oder andere Prüfungen durch CoBe dargestellt werden können, muss das Modell geändert werden.

9.2.3 Kosten und Nutzen des Modelleinsatzes

Der Nutzen des Modells wird erreicht, wenn mit dem Modell günstigere Gesamtkosten als ohne Modell entstehen: Prüfungen können gezielter durchgeführt und kontrolliert werden. Die Planungssicherheit wird durch die Bottom-up-Schätzung von CoBe verbessert. Zusätzlich werden Erkenntnisse über den Prozess durch die Analyse für das Modell gewonnen.

Das Modell ist auch in Situationen nützlich, in denen der Prozess mit Prüfparametern definiert ist und genügend Erfahrung für die Kostenschätzung vorhanden ist: Es erlaubt, Kosten und Nutzen darzustellen, beispielsweise für den Vergleich mit Konkurrenten. Bei sich ändernden Rahmenbedingungen zeigt CoBe, ob und wie der Prozess angepasst werden kann. Experten für die Kostenschätzung können ausfallen oder nicht verfügbar sein. Dann ist CoBe nützlich, weil es quantitativ Erfahrungen aus abgeschlossenen Projekten enthält und dadurch die Kostenschätzung unterstützt. Die Kosten-Nutzen-Analyse wird verlangt (PMI, 2000) und kann mit CoBe auf einfache Weise für ein konkretes Projekt durchgeführt werden.

Basierend auf den Erfahrungen in der Validierung lassen sich die Kosten für den Modelleinsatz abschätzen: Für die Datenerhebung muss für Befragung und Beschaffung von Archivdaten mit etwa einer Arbeitswoche gerechnet werden. Für Analyse und Abbildung komplexer Prozesse werden zwischen zwei und vier Arbeitswochen benötigt. Modelländerungen kosten zwischen einer Stunde, beispielsweise um Ergeb-

nisse zusammenzufassen und übersichtlich darzustellen, und einem Entwicklermonat. Sobald Daten für die Quantifizierung benötigt werden, nimmt vor allem die Literatursuche viel Zeit in Anspruch.

9.2.4 Abgrenzung zu SESAM und zum QS-Modell

In CoBe sind Zusammenhänge aus SESAM-Modellen übernommen, vor allem aus dem QS-Modell (Drappa, 1998). Die beiden Ansätze und die Modelle unterscheiden sich auf mehreren Ebenen:

- Der Modellzweck ist unterschiedlich. Mit SESAM und dem QS-Modell sollen die Spieler Projektleitung erfahren können. Es wird also ein Lernziel verfolgt, das alle wesentlichen Aktivitäten des Projektleiters enthält: Planung, Stellenbesetzung und Projektführung sollen erlernt werden. CoBe dagegen verfolgt kein Lernziel, sondern soll Entscheidungen über Prüfungen unterstützen, in dem es dazu notwendige Informationen liefert. CoBe konzentriert sich darum auf die Prüfplanung.
- Abgeleitet aus dem Modellzweck liefert CoBe Antworten auf bestimmte Fragen, die durch die Eingaben festgelegt sind. SESAM ist im Gegensatz dazu prinzipiell offen: Der Spieler hat mehr Freiheit, er ist z.B. nicht an eine zeitliche Reihenfolge gebunden und kann bestimmen, welche und wie viele Mitarbeiter für welche Aktivität eingesetzt werden sollen. Er kann in jedem Schritt in das Projekt eingreifen. Es ist dann aber notwendig, dass der Tutor die Spiele analysiert. Der Tutor bestimmt dabei, welche Fragen beantwortet werden sollen. Er verdeutlicht die Stärken und Schwächen der Spieler durch Vergleich (Hampp und Opferkuch, 2007), während CoBe den Vergleich bereits enthält: Der Nutzen zeigt die entfallenden Kosten.
- Im QS-Modell stehen konkrete Entscheidungen über den Zeitpunkt und die Stellenbesetzung der Prüfungen und der Korrektur um Vordergrund, dagegen sind in CoBe einzelne Prüfparameter modelliert.
- Planung und Kostenschätzung der Qualitätssicherung ist ein Lernziel des QS-Modells; CoBe unterstützt Planung und Kostenschätzung direkt.
- CoBe berücksichtigt die langfristigen Auswirkungen des Projekts durch den Einsatz und die Wartung des Produkts. Im QS-Modell endet die Simulation, wenn das Produkt ausgeliefert wurde; Qualität wird durch Fehler und Unvollständigkeiten des Produkts dargestellt.
- CoBe und QS-Modell können kalibriert werden. In CoBe ist die Kalibrierung aber explizit durch Parameter modelliert.
- CoBe bildet, anders als das QS-Modell, Wiederverwendung von Software ab.

9.2.5 Grenzen

CoBe bildet Auswirkungen von Entscheidungen für das Projekt und durch das Produkt ab. Nach Auslieferung werden Fehlerfolgekosten im Einsatz und Kosten für korrektive Wartung betrachtet. Wartbarkeit wird nicht betrachtet. Auswirkungen über

das Projekt und sein Produkt hinaus sind nicht enthalten, weil CoBe auf die Entscheidungen des Projektleiters und QS-Verantwortlichen zugeschnitten ist. Unternehmensstrategie oder organisationsweite Prozessverbesserung sind nicht direkt in CoBe abgebildet. Dazu gehören beispielsweise der Lerneffekt, der durch Reviews erreicht wird, oder Marktvorteile durch ein bestimmtes Produkt, die sich erst in folgenden Produktversionen auszahlen.

Wie bei anderen algorithmischen Kostenschätzverfahren handelt es sich bei CoBe um ein induktives Modell, das mit statistischen Mittelwerten arbeitet und dessen Resultate Mittelwerte sind. Die Werte sind zwar durch die Modelleingaben mit den Prüfparametern und der Kalibrierung speziell für das geplante Projekt berechnet. Sie werden aber nicht exakt zutreffen, weil es sich um ein induktives Modell handelt. Der Istwert wird also von den Werten etwas abweichen, so dass die Modellresultate nicht direkt in die Planung übernommen werden können. Mit einer unterstellten Normalverteilung wird eine solche Mittelwertschätzung mit einer Wahrscheinlichkeit von 50 % überschritten.

Da das Modell die Realität verkürzt, kann es nicht direkt zur Optimierung verwendet werden. Insbesondere muss der Entscheider zusätzlich diejenigen Aspekte betrachten, die nicht im Modell enthalten sind (Laux, 1998).

9.3 Ausblick

Das Modell kann als Erfahrungssammlung dienen, um andere Modelle aus einzelnen Zusammenhängen zu konstruieren. Ähnlich wie CoBe Zusammenhänge des QS-Modells verwendet, können andere Modelle Zusammenhänge aus CoBe verwenden.

CoBe enthält die Metriken, mit denen Prüfungen und ihre Auswirkungen erfasst und kontrolliert werden können. Damit gibt es ein Metrikprogramm vor, das die wesentlichen Merkmale zur Qualitätsbewertung enthält. Die Metriken sind gebräuchlich, so dass sie auch ohne Modell interpretiert werden können. Die Modellzusammenhänge klären die Interpretation der Metriken.

9.4 Schlussbemerkungen

Jeder Projektleiter muss sich Entscheidungen über den Kompromiss zwischen den Kosten, zu denen auch Termin und Personal gehören, und der Qualität stellen. In der Software-Entwicklung, bei der ein immaterielles Produkt entsteht, sind diese Entscheidungen schwierig zu treffen und schwierig zu rechtfertigen. Prozesszertifzierungen geben Entscheidungen zwar auf abstrakter, aber nicht auf konkreter Ebene vor. Projekte sind zu unterschiedlich, als dass ein allgemeingültiges Vorgehen vorgegeben werden kann.

In dieser Arbeit wurde, aufbauend auf anderen quantitativen Modellen, ein Kosten-Nutzen-Modell für Entscheidungen über Prüfungen erstellt. Dazu wurden vorhandene Erkenntnisse über die Zusammenhänge zwischen konkreten Entscheidungen

über Prüfungen in Software-Projekten und ihren Auswirkungen auf Kosten und Nutzen analysiert, formalisiert und implementiert. Das Modell wurde in der Industrie validiert. Das Modell erlaubt, die schwierigen Entscheidungen rationaler zu diskutieren und rationaler zu fällen.

Verzeichnis der Bezeichner

In den Gleichungen von CoBe, die in Kapitel 6 dargestellt sind, werden die unten aufgeführten Bezeichner verwendet. Dabei gelten folgende Konventionen:

- Direkte Metriken sind großgeschrieben (z.B. S für den Umfang).
- Abgeleitete Metriken sind in der Regel kleingeschrieben (z.B. m). Ausnahmen sind die Fehlerentdeckungsquote Q und die Überdeckung C, um zwischen geforderter (C) und erreichter Überdeckung (c) zu unterscheiden.
- Variable Parameter sind *kursiv*, binäre Variablen **fett** gesetzt.

Grundlagen

S	Umfang
S_{FP}	Umfang in Function Points
S_{FPneu}	Umfang neuer Software Umfang in Function Points
S_{FPwv}	Umfang wiederverwendeter Software in Function Points
S_{Seiten}	Umfang in Seiten
$S_{\mathrm{Anweisungen}}$	Umfang in Anweisungen
s	Umfangsanteil
Q, Q_{wv}	Fehlerentdeckungsquote, in wiederverwendeter Software
Q_{K}	Korrekturquote
$Q_{p,Art,Schwere}$	Fehlerentdeckungsquote in Prüfung p einer Fehlerart und -schwere
A	Aufwand in Entwicklerstunden (Eh), -monaten (EM), -jahren (EJ)
M	Mitarbeiterzahl
D	Dauer in Arbeitstagen, -wochen, -monaten
fd	Fehlerdichte
af	Einflussfaktoren auf den Aufwand
ff	Einflussfaktoren auf die Fehlerzahlen
cf	Einflussfaktoren auf die Überdeckungen
r_0 und r_1	Regressionsparameter allgemein
$r_{0\mathrm{f}}$ und $r_{1\mathrm{f}}$	Regressionsparameter zur Berechnung der Fehlerdichte

Kalibrierungsparameter

k_{F}	Fehlerfaktor	sf_{Code}	Umfangsfaktor Code
k_{A}	Aufwandsfaktor	sf_{Spez}	Umfangsfaktor Spezifikation
k_{D}	Dauerfaktor	sf_{Entwurf}	Umfangsfaktor Entwurf

Abkürzungen der Prüfungen

SR	Spezifikationsreview	PT	Subsystemsintegrationstest (Package für Subsystem)
ER	Entwurfsreview		
CR	Codereview	IT	Systemintegrationstest
CA	Codeanalyse	ST	Systemtest
MT	Modultest	FT	Feldtest

Eingaben für den Prüfprozess

p	Formelzeichen für eine Prüfung
Review	Formelzeichen für ein Review (SR, ER oder CR)
Test	Formelzeichen für einen Test (MT, PT, IT, ST)
$\boldsymbol{P}$	Findet die Prüfung statt?
$\boldsymbol{WV}$	Wird wiederverwendete Software geprüft?
$s_{\text{wdh},Test}$	Umfangsanteil für die Wiederholung
$a_{\text{wdh},Test}$	Aufwandsanteil für die Wiederholung
m_{K}	Anteil Korrektoren

Fehler und Fehlerkategorien

Schwere	Fehlerkategorie mit Klassen Nebenfehler (NF), Hauptfehler (HF), kritische Fehler (KF); Blockierende Fehler (BF) als Teil der kritischen Fehler.
Art	Fehlerkategorie mit Klassen Spezifikation, Entwurf, Code
Ursprung	Software-Ursprung, Fehlerentstehung: Hinzugefügt, geändert, wiederverwendet (“wv”), “neu” fasst hinzugefügt und geändert zusammen
$fp_{\text{NF}}, fp_{\text{HF}}, fp_{\text{KF}}, fp_{\text{BF}}$	Prozentuale Verteilung auf Fehlerschwere
$fp_{\text{Spez}}, fp_{\text{Entwurf}}, fp_{\text{Code}}$	Prozentuale Verteilung auf Fehlerarten
$F_{\text{neu}}, F_{\text{wv}}$	Zahl eingefügten Fehler in neuer oder in wiederverwendete Software
$F_{\text{entdeckt}}, F_{\text{enthalten}}, F_{\text{korrigiert}}, F_{\text{entfallend}}$	Fehlerzahlen für entdeckte, enthaltene, korrigierte, entfallende Fehler, unterschieden für Prüfungen, Arten und Schwere durch den Index, beispielsweise für korrigierte Fehler und für entfallende Fehler:
$F_{p,\text{korrigiert},Ursrpung,Art,Schwere}$	Zahl korrigierter Fehler der *Art* und *Schwere* nach der Prüfung p
$F_{p,\text{entfallend},p',Ursrpung,Art,Schwere}$	Zahl der entfallenden Fehler in der Prüfung p', weil die Fehler in der Prüfung p bereits entdeckt und dann korrigiert wurden.
KP	Tester- oder Gutachterkompetenz mit 7 Klassen: extra niedrig, sehr niedrig, niedrig, normal, hoch, sehr hoch, extra hoch (XL, VL, L, N, H, VH, XH)

COCOMO-II-Formelzeichen

EM_i	Einflussfaktor i
E, B, SF_j, A[a]	Exponent und Parameter des Exponenten für Aufwandsformel
PM	Mit COCOMO II berechneter Aufwand

a. In CoBe steht A für Aufwand, in COCOMO II für den Produktivitätsparameter

Korrekturaufwand

a_{KBasis}	Basiswert für den Korrekturaufwand pro Fehler
af_p	Einfluss der Prüfung (Latenzzeit), abhängig vom Umfang
af_{W}	Einfluss der Wartung (Latenzzeit), abhängig vom Umfang
af_{Art}	Einfluss der Fehlerart (Latenzzeit)
$af_{Schwere}$	Einfluss der Fehlerschwere (af_{NF}, af_{HF} und af_{KF})
$a_{\text{K},p,Art,Schwere}$	Korrekturaufwand pro Fehler der *Art* und *Schwere* nach Prüfung p
$A_{\text{K},p}$	Korrekturaufwand nach Prüfung p

Aufwandseinfluss

af_{O}	Zuschlag für Organisation (in %)
af	Modellweiter Produktivitätsparameter

Review

G_{Review}	Gutachterzahl im Review *Review*
Q_{Review}	Basis-Fehlerentdeckungsquote für das Review *Review*
q_{r}	Fehlerentdeckungsquote eines Gutachters, durch Regression ermittelbar
r_{qr}	Regressionsparameter für Gutachterzahl und Fehlerentdeckungsquote
$ff_{Review,Art}$	Anpassung der Entdeckungsquote an die Fehlerart *Art*
$ff_{Review,Schwere}$	Anpassung der Entdeckungsquote an die Fehlerschwere
$f_{\text{Priorisierung}}$	Funktion für Fehlerentdeckung mit priorisierten Reviews
$f_{\text{Vorbereitung}}$	Funktion für den Einfluss der Vorbereitung auf die Fehlerentdeckung
v_{Review}	Vorbereitungsrate für das Review
KP_{Review}	Gutachterkompetenz, 7 Klassen (extra niedrig bis extra hoch)
$ff_{Review,KP}$	Einflussfaktor für Fehlerentdeckungsquote durch Gutachterkompetenz *KP*
s_{Review}	Anteil des geprüften Umfangs des Artefakts im Review
$r_{s,Schwere}$	Regressionsparameter für Einfluss auf Fehlerentdeckung verschiedener Fehlerschwere durch priorisiert geprüften Teil eines Artefakts

Test

Testtechniken sind Black-Box-Test (BBT), Glass-Box-Test (GBT) und Brute-Force-Test (BFT). Im Black-Box-Test wird unterschieden: Funktionsabdeckung (Funktion), Äquivalenzklassenabdeckung (Äquivalenzklasse) und Sonderfälle.

T, T_{Test}	Testfallzahl, Testfallzahl für den Test *Test*
T_{n}, $T_{\mathrm{n},Test}$	Nominale Testfallzahl für den vollständigen Black-Box-Test, für den Test *Test*
$r_{0\mathrm{t}}$, $r_{1\mathrm{t}}$	Regressionsparameter zwischen Umfang und nominaler Testfallzahl
$r_{0\mathrm{t},Test}$, $r_{1\mathrm{t},Test}$	Regressionsparameter für den Test *Test*
t	Normierte Testfallzahl: $t = T / T_n$
t_{Test}	Normierte Testfallzahl des Tests *Test*.
c	Erreichte Überdeckung allgemein (Überdeckungsgrad)
C	Geforderte Überdeckung allgemein (Überdeckungskriterium)
c_0, C_0	Erreichte und geforderte Anweisungsüberdeckung
c_1, C_1	Erreichte und geforderte Zweigüberdeckung
c_3, C_3	Erreichte und geforderte Termüberdeckung
c_4, C_4	Erreichte und geforderte Schleifenüberdeckung
cf_1, cf_3, cf_4	Faktoren für den Zusammenhang zwischen den Überdeckungen
$r_{0\mathrm{c}}$, $r_{1\mathrm{c}}$	Regressionsparameter für normierte Testfallzahl und Überdeckungsgrad
q_{t}	Anteil der Fehler, die ein einziger Testfall entdeckt
r_{qt}	Parameter für die Schätzung von q_{t} mit linearer Regression
$ff_{Test,Art}$	Anpassung der Entdeckungsquote an die Fehlerart *Art*
$ff_{Test,Schwere}$	Anpassung der Entdeckungsquote an die Fehlerschwere
KP_{Test}	Testerkompetenz, 7 Klassen (extra niedrig bis extra hoch)
$ff_{Test,KP}$	Einflussfaktor für Fehlerentdeckungsquote durch Testerkompetenz *KP*

Literatur

Ahonen, J. J.; Junttila, T. (2003): A Case Study on Quality-Affecting Problems in Software Engineering Projects. **Proceedings of the IEEE International Conference on Software-Science, Technology & Engineering (SwSTE'03),** 2003.

Alexander, I.; Robertson, S. (2004): Understanding Project Sociology by Modeling Stakeholders. **IEEE Software, vol. 21, no. 1**, 2004, 23-27.

Automotive SIG (2005): **Automotive SPICE Process Reference Model.** Version 4.2, 2005-08-12, http://www.automotivespice.com.

Basili, V. R.; Perricone, B. T. (1984): Software Errors and Complexity: An Empirical Investigation. **Communications of the ACM, vol. 27, no. 1**, 1984, 42-52.

Basili, V. R.; Selby, R. W. (1987): Comparing the Effectiveness of Software Testing Strategies. **IEEE Transactions on Software Engineering, vol. 13, no. 12,** 1987, 1278-1296.

Basili, V. R.; Rombach, H. D. (1988): The TAME Project: Towards Improvement-Oriented Software Environments. **IEEE Transactions on Software Engineering, vol. 14, no. 6,** 1988, 758-773.

Basili, V. R. (1995): The Experience Factory and Its Relationship to Other Quality Approaches. **Advances in Computers, vol. 41,** 1995, 65-82.

Basili, V. R.; Briand, L.; Condon, S.; Yong-Mi, K.; Melo, W. L.; Valett, J. D. (1996): Understanding and Predicting the Process of Software Maintenance Releases. IEEE, **Proceedings of the International Conference on Software Engineering (ICSE 18**), 1996, 464-474.

Bassin, K.; Biyani, S.; Santhanam, P. (2002): Metrics to Evaluate Vendor-Developed Software Based on Test Case Execution Results. **IBM Systems Journal, vol. 41, no. 1**, 2002, 13-30.

Bauer, B. (2008): **Sensitivitätsanalyse von Kosten-Nutzen-Modellen für Software-Prüfungen.** Diplomarbeit Nr. 2793, Universität Stuttgart, 2008.

Beck, K. (2003): **Test-Driven Development.** Addison-Wesley, 2003.

Beizer, B. (1990): **Software Testing Techniques**. 2nd Ed., Van Nostrand Reinhold, 1990.

Biffl, S. (2001): **Software Inspection Techniques to Support Project and Quality Management.** Habilitationsschrift, Shaker Verlag, 2001.

Biffl, S.; Halling, M. (2002): Investigating the Influence of Inspector Capability Factors with Four Inspection Techniques on Inspection Performance. **Proceedings of the Eighth Symposium on Software Metrics (METRICS'02)**, 2002, 107-117.

Biffl, S.; Halling, M. (2003): Investigating the Defect Detection Effectiveness and Cost Benefit of Nominal Inspection Teams. **IEEE Transactions on Software Engineering, vol. 29, no. 5,** 2003, 385-397.

Boehm, B. W. (1976): Software Engineering. **IEEE Transactions on Software Engineering, vol. C-25, no. 12,** 1976, 1226-1241.

Boehm, B. W. (1981): **Software Engineering Economics.** Prentice Hall, 1981.

Boehm, B. W. (1987): Industrial Software Metrics Top 10 List. **IEEE Software, vol. 4, no. 5**, 1987, 84-85.

Boehm, B. W. (1991): Software Risk Management: Principles and Practices. **IEEE Software vol. 8, no. 1**, 1991, 32-41.

Boehm, B.W. (2000): **Software Cost Estimation with COCOMO II.** Prentice-Hall, 2000.

Boehm, B.W.; Huang, L.; Jain, A. (2003): **Reasoning About the Value of Dependability: The iDave Model.** http://csse.usc.edu/csse/TECHRPTS/2003/usccse2003-519/usccse2003-519.pdf (18.09.2008).

Boehm, B.W.; Huang, L.; Jain, A.; Madachy, R. (2004): The ROI of Software Dependability: The iDave Model. **IEEE Software, vol. 21, no. 3**, 2004, 54–61.

Bossel, H. (2004): **Systeme, Dynamik, Simulation.** Books on Demand, 2004.

Brand, D. (2000): A Software Falsifier. **International Symposium on Software Reliability Engineering**, Jun. 28, 2000, 174-185.

Brand, D.; Krohm, F. (2003): **Arithmetic Reasoning for Static Analysis of Software.** IBM Research Report RC22905 (W0304-120), 2003.

Brand, D.; Buss, M.; Sreedhas, V. C. (2007): Evidence-Based Analysis and Inferring Preconditions for Bug Detection. **Proceedings of the International Conference on Software Maintenance (ICSM),** 2007, 44-53.

Briand, L. and Pfahl, D. (1999): Using Simulation for Assessing the Real Impact of Test Coverage on Defect Coverage. **Proceedings of the 10th International Symposium on Software Reliability Engineering (ISSRE'99)**, 1999, 148-157.

Brodman, J. G.; Johnson, D. L. (1996): Return on Investment from Software Process Improvement as Measured by U.S. Industry. **CrossTalk,** April 1996.

Buckley, M.; Chillarege, R. (1995): Discovering Relationships between Service and Customer Satisfaction. **Proceedings of International Conference on Software Maintenance (ICSM), 1995,** 192-201.

Burr, K.; Young, W. (1998): Combinatorial Test Techniques: Table-based Automation, Test Generation and Code Coverage. **Proceedings of the International Conference on Software Testing Analysis and Review**, 1998**,** 503-513.

Bush, M. (1990): Improving Software Quality: The Use of Formal Inspections at the Jet Propulsion Laboratory. **IEEE Proceedings of the 12th International Conference on Software Engineering, (ICSE),** 1990, 196-199.

Chaar, J. K.; Halliday, M. J.; Ghandari, I. S.; Chillarege, R. (1993): In-Process Evaluation for Software Inspection and Test. **IEEE Transactions on Software Engineering, vol. 19, no. 11,** 1993, 1055-1070.

Chernak, Y. (2001): Validating and Improving Test-Case Effectiveness. **IEEE Software, vol. 18, no. 1**, 2001, 81-86.

Chillarege, R.; Bhandari, I. S.; Chaar, J. K.; Halliday, M. J.; Moebus, D. S. Ray, B. K., Wong, M.-Y. (1992): Orthogonal Defect Classification - A Concept for In-Process Measurements. **IEEE Transactions on Software Engineering, vol. 18, no. 11,** 1992, 943-956.

Chou, A.; Yang, J.; Chelf, B.; Hallem, S.; Engler, D. (2001): An Empirical Study of Operating System Errors. **Proceedings of the 18th ACM Symposium on Operating Systems Principles,** 2001, 73-88.

Chrissis, M. B.; Konrad, M.; Shrum, S. (2003): **CMMI. Guidelines for Process Integration and Product Improvement.** Addison-Wesley, 2003.

Chulani, S.; Ray, B.; Santhanam, P.; Leszkowicz, R. (2003): Metrics for Managing Customer View of Software Quality. **Proceedings of the 9th International Software Metrics Symposium (METRICS'03)**, 2003, 189-198.

CMMI Product Team (2002): **Capability Maturity Model Integration (CMMI), Version 1.1.** Carnegie Mellon University, Software Engineering Institute, CMU/SEI-2002-TR-029, 2002.

Conte, S. D.; Dunsmore, H. E.; Shen, V. Y. (1986): **Software Engineering Metrics and Models.** Benjamin/Cummings Publishing Company, INC, 1986.

Cornelissen, W.; Klaassen, A.; Matsinger, A.; van Wee, G. (1995): How to Make Intuitive Testing More Systematic. **IEEE Software vol. 12 no. 5,** 1995, 87-89.

Corsten, H.; Reiß, M. (Hrsg.) (1999): **Betriebswirtschaftslehre**. 3. Auflage, Oldenbourg, 1999.

Cusumano, M. A.(1992): **Objectives and Context of Software Measurement, Analysis, and Control.** Massachusetts Institute of Technology, Sloan School WP#3471-92/BPS, 1992.

Dahl, O.-J.; Dijkstra, E. W.; Hoare, C. A. R. (1972): **Structured Programming.** Academic Press, 1972.

Deininger, M. (1995): **Quantitative Erfassung der Software und ihres Entstehungsprozesses.** Dissertation, Universität Stuttgart, Dr. Kovac, 1995.

DeMarco, T.; Lister, T. (1999): **Peopleware: Productive Projects and Teams.** Dorset House Publ., 2nd Ed., 1999.

Demirörs, O.; Yildiz, Ö.; Güceglioglu, A. S. (2000): Using Cost of Software Quality for a Process Improvement. **Proceedings of the 26th Euromicro Conference, vol. 2, 5-7,** Sept. 2000, 286-291.

Devnani-Chulani, S. (1997): **Results of Delphi for the Defect Introduction Model.** USC-CSE, 1997.

Diaz, M.; King, J. (2002): How CMM Impacts Quality, Productivity, Rework, and the Bottom Line. **CrossTalk,** March 2002.

DIN 55350 (1995): **Begriffe zu Qualitätsmanagement und Statistik -Teil 11: Begriffe des Qualitätsmanagements.** DIN 55350-11:1995-08, Deutsches Institut für Normung e.V., Beuth Verlag, 1995.

Do, H.; Rothermel, G.; Kinneer, A. (2006): Prioritizing JUnit Test Cases: An Empirical Assessment and Cost-Benefits Analysis. **Empirical Software Engineering, vol. 11, no. 1,** 2006, 33-70.

Drappa, A. (1998): **Quantitative Modellierung von Softwareprojekten.** Dissertation, Universität Stuttgart, Shaker, 2000.

Dunn, R. H. (1984): **Software Defect Removal.** McGraw-Hill Book Company, 1984.

Dupuy, A.; Leveson, N. (2000): An Empirical Evaluation of the MC/DC Coverage Criterion on the HETE-2 Satellite Software. **Proceedings of the Digital Avionics Systems Conference (DASC)**, 2000, 1B6/1-1B6/6.

Ebert, C.; Dumke, R.; Bundschuh, M.; Schmietendorf, A. (2005): **Best Practices in Software Measurement.** Springer, 2005.

Eisenbarth, T.; Rohrbach, J. (1998): **Evaluation und Erprobung des SESAM-2-Systems anhand eines strikt atomaren Modells.** Diplomarbeit Nr. 1633, Universität Stuttgart, 1998.

El Emam, K. E.; Benlarbi, S.; Goel, N.; Shesh, N. Rai (2001): The Confounding Effect of Class Size on the Validity of Object-Oriented Metrics. **IEEE Transactions on Software Engineering, vol. 27, no. 7,** 2001, 630-650.

El Emam, K. (2005): **The ROI from Software Quality.** Auerbach Publications, 2005.

Ellims, M.; Bridges, J.; Ince, D. C. (2006): The Economics of Unit Testing. **Empirical Software Engineering, Springer, vol. 11, no. 1,** 2006.

Endres, A.; Rombach, H. D. (2003): **A Handbook of Software and Systems Engineering. Empirical Observations, Laws and Theories.** Pearson, Addison-Wesley, 2003.

Evanco, W. E. (2001): Prediction Models for Software Fault Correction Effort. **Proceedings of the 5th Conference on Software Maintenance and Reengineering, CSMR 2001,** 114-120.

Fagan, M. E. (1976): Design and Code Inspections to Reduce Errors in Program Development. **IBM Systems Journal, vol. 15, no. 3**, 1976, 182-211.

Fagan, M. E. (1986): Advances in Software Inspections. **IEEE Transactions on Software Engineering, vol. SE-12, no. 7,** 1986, 744-751.

Fahrmeir, L.; Künstler, R.; Pigeot, I.; Tutz, G. (2007): **Statistik. Der Weg zur Datenanalyse.** Springer, 6. überarbeitete Auflage, 2007.

Fenton, N. E.; Pfleeger, S. L. (1997): **Software Metrics. A Rigorous & Practical Approach.** PWS Publishing Company, 2nd Ed., 1997.

Fenton, N. E.; Neil, M. (1999): A Critique of Software Defect Prediction Models. **IEEE Transactions on Software Engineering, vol. 25, no. 3,** 1999.

Fishwick, P. A. (1995): **Simulation Model Design and Execution. Building Digital Worlds.** Prentice Hall, 1995.

Fink, M.; Hampp, T. (2005): Eine Untersuchung zum Metrikeinsatz in der Industrie. In: Bühren, G.; Bundschuh, M.; Dumke, R.: **MetriKon 2005 - Praxis der Software Messung.** Shaker, 2005.

Florac, W. A. (1992): **Software Quality Measurement: A Framework for Counting Problems and Defects.** Carnegie Mellon University, Software Engineering Institute, CMU/SEI-92-TR-022, 1992.

Freedman, D. P.; Weinberg, G. M. (1982): **Handbook of Walkthroughs, Inspections, and Technical Reviews.** Little, Brown and Company, 3rd Ed., 1982.

Frühauf, K.; Ludewig, J.; Sandmayr, H. (2001): **Software-Projektmanagement und -Qualitätssicherung.** vdf Hochschulverlag, 4. Aufl., 2001.

Frühauf, K.; Ludewig, J.; Sandmayr, H. (2006): **Software-Prüfung. Eine Anleitung zum Test und zur Inspektion.** vdf, 6. Aufl., 2006.

Gass, S. I. (1983): Decision-Aiding Models: Validation, Assessment, and Related Issues for Policy Analysis. **Operations Research, vol. 31, no. 4**, 1983, 603-631.

Garvin, D. A. (1988): **Managing Quality. The Strategic and Competitive Edge.** Free Press, Macmillan, 1988.

Gibson, D. L.; Goldenson, D. R. Kost, K. (2006): **Performance Results of CMMI-Based Process Improvement.** Carnegie Mellon University, Software Engineering Institute, CMU/SEI-2006-TR-004, 2006.

Glinz, M.; R. Wieringa (2007): Stakeholders in Requirements Engineering. **IEEE Software vol. 24, no. 2,** 2007, 18-20.

Goethert, W. B.; Bailey, E. K.; Busby, M. B. (1992): **Software Effort & Schedule Measurement: A Framework for Counting Staff-hours and Reporting Schedule Information.** Carnegie Mellon University, Software Engineering Institute, CMU/SEI-92-TR-021, 1992.

Goodenough, J. B.; Gerhart, S. L. (1977): Toward a Theory of Testing: Data Selection Criteria. In: Yeh, R. T.: **Current Trends in Programming Methodology, vol. II,** Program Validation, Prentice-Hall, Inc., 1977.

Grady, R. B.; Caswell, D. L. (1987): **Software Metrics: Establishing a Company-Wide Program.** Prentice-Hall, Inc., 1987.

Grady, R. (1992): **Practical Software Metrics for Project Management and Process Improvement.** Prentice-Hall, Inc., 1992.

Haley, T.; Ireland, B.; Wojtaszek, E.; Nash, D.; Dion, R. (1995): **Raytheon Electronic Systems Experience in Software Process Improvement.** Carnegie Mellon University, Software Engineering Institute, CMU/SEI-95-TR-017, 1995.

Haley, T. J (1996): Software Process Improvement At Raytheon. **IEEE Software, vol. 13, no. 6**, 1996.

Hampp, T. (2001): **Eine feingranulare SESAM-Variante.** Diplomarbeit Nr. 1931, Universität Stuttgart, 2001.

Hampp, T.; Opferkuch, S. (2007): Software-Engineering-Simulation als Brücke zwischen Vorlesung und Praktikum. **Software Engineering im Unterricht der Hochschulen (SEUH 10),** 2007.

Hampp, T.; Knauß, M. (2008): Eine Untersuchung über Korrekturkosten von Software-Fehlern. **Softwaretechnik-Trends, Band 28 Heft 2**, Gesellschaft für Informatik, Mai 2008.

Hanusch, H. (1987): **Nutzen-Kosten-Analyse.** Vahlen, 1987.

Harrison, W.; Raffo, D.; Settle, J.; Eickelmann, N. (1999): Technology Review: Adapting Financial Measures: Making a Business Case for Software Process Improvement. **Software Quality Journal, 8,** 1999, 211-231.

Horgan, J. R.; London, S.; Lyu, M. R. (1994): Achieving Software Quality with Testing Coverage Measures. **IEEE Computer, vol. 27, no. 9,** 1994, 40-60.

Hörmann, K.; Dittmann, L.; Hindel, B.; Müller, M. (2006): **SPICE in der Praxis. Interpretationshilfe für Anwender und Assessoren.** dpunkt., 2006.

Huang, L.G.; Boehm, B. (2006): How much Software Quality Investment is Enough: A Value-Based Approach. **IEEE Software, vol.23, no.5,** 2006, 88-95.

Humphrey, W. S.(1995): **A Discipline for Software Engineering.** Addison-Wesley Publishing Company, Inc., 1995.

Hunter, R. B.; Thayer, R. H. (2001): **Software Process Improvement.** IEEE, 2001.

Hutchins, M.; Foster, H.; Goradia, T.; Ostrand, T. (1994): Experiments on the Effectiveness of Dataflow- and Controlflow-Based Test Adequacy Criteria. **Proceedings of the 15th International Conference on Software Engineering (ICSE'94)** 1994.

IEEE 610 (1990): **IEEE Standard Glossary of Software Engineering Terminology,** IEEE Std. 610.12-1990, 1990.

IEEE 828 (2005): **IEEE Standard for Software Configuration Management Plans.** IEEE Std. 828-2005, 2005.

IEEE 829 (1998): **IEEE Standard for Software Test Documentation,** IEEE Std. 829-1998, 1998.

IEEE 982.1 (2005): **IEEE Standard Dictionary of Measures of the Software Aspects of Dependability.** IEEE Std. 982.1-2005, IEEE, 2005.

IEEE 1044 (1993): **IEEE Standard Classification for Software Anomalies.** IEEE Std. 1044-1993, IEEE, 1993.

IEEE 1045 (1992): **IEEE Standard for Productivity Metrics.** IEEE Std. 1045-1992, IEEE, 1992.

IEEE 1058 (1998): **IEEE Standard for Software Project Management Plans.** IEEE Std. 1058-1998, IEEE, 1998.

IEEE 1061 (1998): **IEEE Standard for a Software Quality Metrics Methodology.** IEEE Std. 1061-1998 (R 2004), IEEE, 1998.

IEEE 1490 (2003): **IEEE Guide - Adoption of PMI Standard - a Guide to the Projectmanagement Body of Knowledge.** IEEE Std. 1490-2003, IEEE, 2003.

IEEE 12207.0 (1996): **Industry Implementation of International Standard ISO/IEC 12207:1995. Software Life Cycle Processes.** IEEE/EIA Std. 12207.0-1996, IEEE, 1996.

IEEE 12207.1 (1997): **Industry Implementation of International Standard ISO/IEC 12207:1995. Software Life Cycle Processes - Life Cycle Data.** IEEE EIA Std. 12207.1-1997, IEEE, 1997.

IFPUG (2004): **Function Point Counting Practices Manual.** Release 4.2.1, International Function Point Users Group, 2004.

ISBSG (2005): **Application Software Maintenance and Support. An Initial Analysis of New Data.** International Software Benchmarking Standards Group, February 2005.

ISO 9000 (2000): **Qualitätsmanagementsysteme - Grundlagen und Begriffe (DIN EN ISO 9000:2000-12)**. Deutsches Institut für Normung e.V., Beuth Verlag, Berlin, 2000.

ISO/IEC 9126 (2001): **Software engineering - Product quality - Part 1: Quality model; Part 2: External metrics; Part 3: Internal metrics; Part 4: Quality in use metrics.** ISO/IEC, Beuth Verlag, 2001.

ISO/IEC 12207 (1997): **Standard for Information Technology— Software life cycle processes.** ISO/IEC, 1997.

ISO/IEC 14143: **Informationstechnik - Messung von Software - Messung der funktionalen Größe** - Teil 1: Begriffsfestlegungen. Beuth Verlag, 2007.

ISO/IEC 14764 (1999): **Software Engineering – Software Life Cycle Processes – Maintenance.** ISO/IEC, 1999.

Jalote, P. (2000): **CMM in Practice: Processes for executing software projects at Infosys.** Addison-Wesley, 2000.

Janzen, D. S.; Saiedian, H. (2006): On the Influence of Test-Driven Development on Software Design. **Proceedings of the 19th Conference on Software Engineering Education and Training,** 2006.

Jiang, Y.; Cukic, B.; Menzies, T.; Bartlow, N. (2008): **Comparing Design and Code Metrics for Software Quality Prediction.** PROMISE'08, 2008, 11-18.

Johnson, S. C. (1978): **Lint, a C Program Checker.** Computer Science Technical Report, Bell Laboratories, 1978.

Jones, C. (1996): **Applied Software Measurement: Assuring Productivity and Quality.** McGraw-Hill, 2nd Ed., 1996.

Jones, C. (2003): **Software Assessments, Benchmarks, and Best Practices.** Addison-Wesley, 2nd Ed., 2003.

Jones C. (2007): **Estimating Software Costs. Bringing Realism to Estimating.** McGraw-Hill, 2nd Ed., 2007.

Juran, J. M. (Ed.) (1962): **Quality Control Handbook.** McGraw-Hill, 2nd Ed., 1962.

Juran, J. M.; Godfrey, A. B. (Eds.) (1998): **Juran's Quality Handbook.** McGraw-Hill, 5th Ed., 1998.

Juristo, N.; Moreno, A. M.; Vegas, S. (2002): A Survey on Testing Technique Empirical Studies: How Limited is Our Knowledge. **Proceedings of the 2002 International Symposium on Empirical Software Engineering** (ISESE'02), 2002.

Juristo, N.; Moreno, A.; Vegas, S. (2004): Reviewing of 25 Years of Testing Technique Experiments. **Empirical Software Engineering, vol. 9, no. 1-2**, 2004, 7-44.

Kalajzic, S. (2001): **Revision von SESAM-Modellen anhand von Meta-Regeln.** Diplomarbeit Nr. 1941, Universität Stuttgart, 2001.

Kamsties, E.; Lott, C. M. (1995): **An Empirical Evaluation of Three Defect-Detection Techniques.** Technical Report ISERN 95–02, Department of Computer Science, University of Kaiserslautern, 1995.

Kan, S. H. (2003): **Metrics and Models in Software Quality Engineering.** Addison-Wesley, 2nd Ed., 2003.

Kasunic, M. (2006): **The State of Software Measurement Practice: Results of 2006 Survey.** Carnegie Mellon University, Software Engineering Institute, CMU/SEI-2006-TR-009, 2006.

Kemerer, C. F. (1987): An Empirical Validation of Software Cost Estimation Models. **Communications of the ACM, vol. 30, no. 5,** 1987.

Kemerer, C. F. (1993): Reliability of Function Points Measurement: A Field Experiment. **Communications of the ACM, vol. 36, no. 2**, 1993, 85-97.

Kerzner, H. (2006): **Project Management: A Systems Approach to Planning, Scheduling, and Controlling**. 9th Ed., John Wiley & Sons, Inc., 2006.

Kikuchi, N.; Kikuno, T. (2001): Improving the Testing Process by Program Static Analysis. **Proceedings of the 8th Asia-Pacific Software Engineering Conference (APSEC 2001),** 2001, 195-201.

Kitchenham, B.; Jeffery, D. R.; Connaughton, C. (2007): Misleading Metrics and Unsound Analyses. **IEEE Software, vol. 24, no. 2,** 2007, 73-78.

Krasner, H. (1998): Using the Cost of Quality Approach for Software. **CrossTalk,** Nov. 1998.

Krauß, S. (2007): **Verfahren der Software-Dokumentation.** Dissertation, Universität Stuttgart, Shaker, 2007.

Laitenberger, O.; Leszak, M.; Stoll, D.; El Emam, K. (1999): Quantitative Modeling of Software Reviews in an Industrial Setting. **Proceedings 6th International Software Metrics Symposium,** IEEE, 1999, 312-323

Lauterbach, L.; Randall, W. (1989): Experimental Evaluation of Six Test Techniques. **Proceedings of the Fourth Annual Conference on Computer Assurance, 'Systems Integrity, Software Safety and Process Security', COMPASS '89,** 1989, 36-41.

Laux, H. (1998): **Entscheidungstheorie.** Springer, 4. Aufl., 1998.

Leszak, M.; Perry, D. E.; Stoll, D. (2002): Classification and Evaluation of Defects in a Project Retrospective. **The Journal of Systems and Software 61,** 2002, 173-187.

Liggesmeyer, P. (2002): **Software-Qualität.** Spektrum, 2002.

Ludewig et al. (1994): **SESAM - Software Engineering Simulation durch animierte Modelle.** Bericht 5/94, Fakultät Informatik, Universität Stuttgart, 1994.

Ludewig, J. (1999): **Grundlagen Software Engineering.** Vorlesung, Universität Stuttgart, 1999.

Ludewig, J. (Hrsg.); Krauß, S.; Ludewig, J.; Mandl-Striegnitz, P.; Melchisedech, R.; Reißing, R. (2001): **Praktische Lehrveranstaltungen im Studiengang Softwaretechnik: Programmierkurs, Software-Praktikum, Studienprojekte, Fachstudie.** Bericht der Fakultät Informatik, Universität Stuttgart, 2. Auflage, 2001.

Ludewig, J.; Opferkuch, S. (2004): Software-Wartung - eine Taxonomie. **Softwaretechnik-Trends, Band 24, Heft 2,** Gesellschaft für Informatik, Mai 2004.

Ludewig, J.; Lichter, H. (2007): **Software Engineering. Grundlagen, Menschen, Prozesse, Techniken.** dpunkt, 2007.

Lück, W. (Hrsg.) (2004): **Lexikon der Betriebswirtschaft.** 6. Auflage, Oldenbourg, 2004.

Louridas, P. (2006): Static Code Analysis. **IEEE Software, vol. 23, no. 4**, 2006, 58-61.

Lyu, M. R. (1995): **Handbook of Software Reliability Engineering.** McGraw-Hill, 1995.

Lyu, M. R.; Huang, Z.; Sze, S. K. S.; Cai, X. (2003): **An Empirical Study on Testing and Fault Tolerance for Software Reliability Engineering.** Proceedings of the 14th International Symposium on Software Reliability Engineering (ISSRE'03), 2003.

Malaiya, Y. K.; Li, N.; Bieman, J.; Karcich, R.; Skibbe, B. (1994): The Relationship Between Test Coverage and Reliability. **Proceedings of 5th International Symposium on Software Reliability Engineering,** 1994, 186-195.

Maldenado, J. C.; Carver, J.; Shull, F.; Fabbri, S.; Doria, E.; Martimiano, L.; Mendonca, M.; Basili, V. (2006): Perspective-Based Reading: A Replicated Experiment Focused on Individual Reviewer Effectiveness. **Empirical Software Engineering 11**, 2006, 119-142.

Mandl-Striegnitz, P.; Lichter, H. (1999): **Defizite im Software-Projektmanagement – Erfahrungen aus einer industriellen Studie.** Informatik/Informatique Nr. 5, Oktober 1999, 4-9.

Mandl-Striegnitz, P. (2001): How to Successfully Use Software Project Simulation for Educating Software Project Managers. **Proceedings of the 31st Frontiers in Education Conference (FIE 2001),** 2001.

Martin, R.; Raffo, D. M. (2000): A Model of the Software Development Process Using Both Continuous and Discrete Models. **International Journal of Software Process Improvement and Practice, 5:2/3,** 2000.

Martin, R.; Raffo, D. M. (2001): Application of a Hybrid Process Simulation Model to a Software Development Project. **Journal of Systems and Software 59**, 2001, 237-246.

Metzger, P.; Boddie, J. (1996): **Managing a Programming Project: People and Processes.** Prentice Hall, 2nd Ed., 1996.

Möller, K.-H.; Paulish, D. J. (1993a): **Software-Metriken in der Praxis.** Oldenbourg, 1993.

Möller, K.-H.; Paulish, D. J. (1993b): **An Empirical Investigation of Software Fault Distribution.** Proceedings of the 1st International Metrics Symposium, 1993, 82-90.

mozilla (2007): **Mozilla Bug Tracking,** https://bugzilla.mozilla.org (30.04.2007).

Mühlenkamp, H. (1994): **Kosten-Nutzen-Analyse.** Oldenbourg, 1999.

Müller, U.; Wiegmann, T.; Avc, O. (1998): **"State of the Practice" der Prüf- und Testprozesse in der Softwareentwicklung.** In: Mellis, W.; Herzwurm, G. Stelzer, D.: Studien zur Systementwicklung, Universität zu Köln, Band 16, 1998.

Müller, M.; Höfer, A. (2007): The Effect of Experience on the Test-driven Development Process. **Empirical Software Engineering 12,** 2007, 593-615.

Müller, M. (2007): **Analyzing Software Quality Assurance Strategies through Simulation.** Dissertation, Fraunhofer-Institut für Experimentelles Software Engineering (Fraunhofer IESE), 2007.

Nagappan, N.; Maximilien, E. M.; Bhat, T.; Williams, L. (2008): Realizing Quality Improvement Through Test-driven Development: Results and Experiences of Four Industrial Teams. **Empirical Software Engineering 13,** 2008, 289-302.

Nas, T. F. (1996): **Cost-Benefit Analysis: Theory and Application.** SAGE Publications, 1996.

Niessink, F.; Van Vliet, H. (1998): Two Case Studies in Measuring Software Maintenance Effort. **Proceedings of the International Conference on Software Maintenance (ICSM),** 1998, 76-85.

Olague, H. M.; Etzkorn, L. H.; 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, vol. 33, no. 6,** 2007, 402-419.

Opp, K.-D. (2005): **Methodologie der Sozialwissenschaften: Einführung in Probleme ihrer Theorienbildung und praktischen Anwendungen.** 6. Aufl., VS Verlag für Sozialwissenschaft, 2005.

Park, R. E. (1992): **Software Size Measurement: A Framework for Counting Source Statements.** Carnegie Mellon University, Software Engineering Institute, CMU/SEI-92-TR-020, 1992.

Paul, J. (2007): **Einführung in die Allgemeine Betriebswirtschaftslehre.** Gabler, 2007.

Pigoski, T. M. (1997): **Practical Software Maintenance: Best Practices for Managing Your Software Investment.** Wiley, 1997.

Piwowarski, P.; Ohba, M.; Caruso, J. (1993): Coverage Measurement Experience During Function Test. **Proceedings of the 15th International Conference on Software Engineering,** ICSE'93, 1993.

PMI (2000): **A Guide to the Project Management Body of Knowledge. PMBOK Guide.** 2000 Edition, Project Management Institute, 2000.

Poore, J. H.; Trammell, C. J. (1996): **Cleanroom Software Engineering: A Reader.** NCC Blackwell, 1996.

Porter, A. A.; Votta, L. G.; Basili, V. R. (1995): Comparing Detection Methods for Software Requirements Inspections: A Replicated Experiment. **IEEE Transactions on Software Engineering, vol. 21, no. 4,** 1995, 563-575.

Porter, A. A; Votta, L. G. (1997): What Makes Inspections Work? **IEEE Software, vol. 14, no. 6,**1997, 99-102.

Prechelt, L. (2001): **Kontrollierte Experimente in der Softwaretechnik.** Springer, 2001.

Pressman, R. S. (2005): **Software Engineering. A Practitioner's Approach.** McGraw-Hill, International 6th Ed., 2005.

Pul, M. C. J. van (1993): **Statistical Analysis of Software Reliability Models**. Dissertation, Utrecht, 1993.

QSM (2009): **Function Point Languages Table.** Quantitative Software Management, Inc. http://www.qsm.com/FPGearing.html (27.04.2009).

R (2008): **The R Project for Statistical Computing.** www.r-project.org (04.12.2008).

Raffo, D. M. (2005): Software Project Management Using PROMPT: A Hybrid Metrics, Modeling and Utility Framework. **Information and Software Technology 47**, 2005.

Raffo, D. M. (o.J.): **Applying Process Simulation on NASA Projects.** http://sarpresults.ivv.nasa.gov/ViewResearch/51.jsp (11.06.2008).

Raffo, D. M.; Nayak, U.; Setamanit, S.; Sullivan, P.; Wakeland, W. (o.J.): **Using Software Process Simulation to Assess the Impact of IV&V Activities.** http://sarpresults.ivv.nasa.gov/ViewResearch/51.jsp (11.06.2008).

Raz, T; Yaung, A. T. (1997): Factors Affecting Design Inspection Effectiveness in Software Development. **Information and Software Technology 39,** 1997, 297-305.

Regnell, B.; Runeson, P.; Thelin, T. (2001): Are the Perspectives Really Different? - Further Experimentation on Scenario-Based Reading of Requirements. **Empirical Software Engineering 5**, Kluwer, 2001, 331-356.

Reißing, R. (1996): **Konzeption und Realisierung einer Basismaschine für SESAM-2.** Diplomarbeit Nr. 1345, Universität Stuttgart, 1996.

Rico, D. F. (2000): **Using Cost Benefit Analyses to Develop Software Process Improvement (SPI) Strategies.** DACS State-of-the-Art Report, Data & Analysis Center for Software, 2000.

Rombach. D. H.; Ulery, B. T. (1989a): Establishing a Measurement Based Maintenance Improvement Program: Lessons Learned in the SEL. **Proceedings of the Conference on Software Maintenance,** IEEE, 1989, 50-57.

Rombach, H. D.; Ulery, B. T. (1989b): Improving Software Maintenance Through Measurement. **Proceedings of the IEEE, vol. 77, no. 4,** April 1989, 581 - 595.

RTCA (1992): **Software Considerations in Airborne Systems and Equipment Certification.** RTCA/DO-178B, 1992.

Runeson, P.; Andersson, C.; Thelin, T.; Andrews, A.; Berling, T. (2006): What Do We Know about Defect Detection Methods? **IEEE Software, vol. 23, no. 3,** 2006, 82-90

Rykiel, E. J. (1996): Testing Ecological Models: The Meaning of Validation. **Ecological Modelling, vol. 90, no. 3,** 1996, 229-244.

Sabaliauskaite, G.; Kusumoto, S.; Inou, K. (2002): **Extended Metrics to Evaluate Cost Effectiveness of Software Inspections.** IEICE Trans. Fundamentals/Commun./ Electron./Inf. & Syst., vol. E85-A/B/C/D, no. 1, Jan. 2002.

Saltelli, A.; Ratto, M.; Andres, T.; Campolongo, F. Cariboni, J.; Gatelli, D.; Saisana, M.; Tarantola, S. (2008): **Global Sensitivity Analysis. The Primer.** John Wiley & Sons Ltd., 2008.

Sargent, R. G. (2005): Verification and Validation of Simulation Models. **Proceedings of the 2005 Winter Simulation Conference,** 2005, 130-143.

Sauer, C.; Jeffery, D. R.; Land, L.; Yetton, P. (2000): The Effectiveness of Software Development Technical Reviews. A Behaviorally Motivated Program of Research. **IEEE Transactions on Software Engineering, vol. 26, no. 1**, 2000, 1-14.

Schneck, O. (2005): **Lexikon der Betriebswirtschaft.** 6. Auflage, dtv, 2005.

Schnell, R.; Hill, P. B.; Esser, E. (2005): **Methoden der empirischen Sozialforschung.** 7. Aufl., Oldenbourg, 2005.

Schwinn, T. (2003): **Effiziente Software-Inspektion durch ein Rahmenwerk zur antizipativen Berücksichtigung des Return-On-Investment.** Dissertation. Fakultät für Informatik, Universität Ulm, 2003.

Shull, F.; Basili, V.; Boehm, B.; Brown, A. W.; Costa, P.; Lindvall, M.; Por, D.; Rus, I.; Tesoriero, R.; Zelkowitz, M. (2002): What We Have Learned about Fighting Defects. **Proceedings of the 8th IEEE Symposium on Software Metrics** (METRICS'02), 2002.

Siegwart, K. (2004): **Empirische Untersuchung der analytischen Qualitätssicherung in der Industrie.** Diplomarbeit Nr. 2231, Universität Stuttgart, 2004.

SimLab (2009): **SimLab- Sensitivity Analysis.** http://simlab.jrc.ec.europa.eu/ (13.01.2009).

Slaughter, S. A.; Harter, D. E.; Krishnan, M. S. (1998): Evaluating the Cost of Software Quality. **Communications of the ACM, vol. 41, no. 8**, 1998, 67-73.

Sneed, H.; Winter, M. (2002): **Testen objektorientierter Software.** Hanser, 2002.

Sneed, H. (2004): A Cost Model for Software Maintenance & Evolution. **Proceedings of the 20th IEEE International Conference on Software Maintenance (ICSM'04),** 2004.

Sneed, H. M.; Hasitschka, M.; Teichmann, M.-T. (2004): **Software-Produktmanagement: Wartung und Weiterentwicklung bestehender Anwendungssysteme.** dpunkt., 2004.

Sneed, H. M.; Baumgartner, M.; Seidl, R. (2007): **Der Systemtest: Anforderungsbasiertes Testen von Software-Systemen.** Hanser, 2007.

Smith, D. J.; Simpson, K. G. L. (2005): **Functional Safety. A Straightforward Guide to Applying IEC 61508 and Related Standards.** Elsevier, 2nd Ed., 2005.

Sommerville, I. (2007): **Software Engineering.** Addison-Wesley, 8th Ed., 2007.

Spillner, A.; Linz, T. (2003): **Basiswissen Softwaretest.** dpunkt, 2003.

Spillner, A.; Roßner, T.; Winter, M.; Linz, T. (2006): **Praxiswissen Softwaretest - Testmanagement.** dpunkt., 2006.

Spinellis, D. (2006): Bug Busters. **IEEE Software, vol. 23, no. 2,** 2006, 92-93.

SPR (2009): **SPR KnowledgePlan.** http://www.spr.com/spr-knowledgeplanr.html (27.07.2009).

SPSS (2008): **Enabling the Predictive Enterprise.** www.spss.com (04.12.2008).

Stachowiak, H. (1973): **Allgemeine Modelltheorie.** Springer, 1973.

Stark, G.; Durst, R.; Vowell, C. (1994): Using Metrics in Management Decision Making. **IEEE Computer, vol. 27, no. 9**, 1994.

Stevens, S. S. (1946): On the Theory of Scales of Measurement. **Science, New Series, vol. 103, no. 2684, Jun. 7, 1946,** 677-680.

Sunazuka, T., Azuma, M., and Yamagishi, N. (1985): Software Quality Assessment Technology. **Proceedings of the 8th International Conference on Software Engineering,** IEEE, 1985, 142-148.

Thaller, G. E. (2002): **Software-Test: Verifikation und Validation.** 2. Auflage, Heise, 2002.

Thayer, R. H.; Christensen, M. (2002): **Software Engineering. Volume 2: The Supporting Processes.** 2nd Ed., IEEE, 2002.

Tomys, A.-K. (1995): **Kostenorientiertes Qualitätsmanagement: Qualitätscontrolling zur ständigen Verbesserung der Unternehmensprozesse.** Hanser, 1995.

van Megen, R.; Meyerhoff, D. B. (1995): Costs and Benefits of Early Defect Detection: Experiences from Developing Client Server and Host Applications. **Software Quality Journal, vol. 4, no. 4,** 1995, 247-256.

V-Modell XT (2004): **Das V-Modell XT**, Version 1.2.0. Bundesrepublik Deutschland, 2004.

von Nitzsch, R. (2002): **Entscheidungslehre.** Schäffer-Poeschel, 2002.

Votta, L. G. 1993. Does Every Inspection Need a Meeting? **Proceedings of the 1st ACM SIGSOFT Symposium on Foundations of Software Engineering,** 1993, 107-114.

Wagner, S.; Jürjens, J.; Koller, K.; Trischberger, P. (2005): Comparing Bug Finding Tools with Reviews and Tests. **Proceedings of the 17th International Conference on Testing of Communicating Systems (TestCom'05),** 2005, 40-55.

Wagner, S. (2007): **Cost-Optimisation of Analytical Software Quality Assurance.** Dissertation, TU München, 2007.

Weber, W.; Kabst, R. (2006): **Einführung in die Betriebswirtschaftslehre.** 6. Auflage, Gabler, 2006.

Weller, E. F. (1993): Lessons from Three Years of Inspection Data. **IEEE Software, vol. 10, no. 5,** 1993, 38-45.

Weyuker, E. J.; Ostrand, T. J. (1980): Theories of Program Testing and the Application of Revealing Subdomains. **IEEE Transactions on Software Engineering, vol. SE-6, no. 3,** 1980, 236 - 246.

Wong, W. E.; Horgan, J. R.; London, S.; Mathur, A. P. (1994): Effect of Test Set Size and Block Coverage on the Fault Detection Effectiveness. **Proceedings of the 5th International Symposium on Software Reliability Engineering,** 1994, 230-238.

Yourdon, E. (1995): When Good Enough Software is Best. **IEEE Software, vol. 12, no. 3,** May 1995, 79-81.

Zage, D.; Zage, W. (2003): An Analyses of the Fault Correction Process in a Large-Scale SDL Production Model. **Proceedings of the 25th International Conference on Software Engineering**, 2003.

Zheng, J.; Williams, L.; Nagappan, N.; Snipes, W.; Hudepohl, J. P.; Vouk, M. A. (2006): On the Value of Static Analysis for Fault Detection in Software. **IEEE Transactions on Software Engineering, vol. 32, no. 4,** April 2006.

Zuse, H. (1998): **A Framework for Software Measurement.** de Gruyter, 1998.