von dietmar kropfitsch und uwe vigenschow

mehr zum thema:
www.oose.de

DAS FEHLERMODELL: AUFWANDSSCHÄTZUNG UND PLANUNG VON TESTS

Mit Hilfe eines empirische Ansatzes lassen sich mit einem Tabellenkalkulationsprogramm nicht nur die Aufwände für alle Arten von Tests inklusive der Fehlerkorrekturaufwände in den Griff bekommen. Auch über die aktuelle Produktqualität kann man Gewissheit erlangen. In dem Artikel wird gezeigt, wie Vorhersagen über die Anzahl der durch den Test zu findenden Fehler gemacht werden können. Daraus können dann Aufwandsschätzungen für die Testfallerstellung, Testdurchführung und die Fehlerbehebung gewonnen werden. Quasi nebenbei wird so die Basis für ein Controlling der Testaktivitäten gelegt und es werden Kriterien für die Bestimmung des Endes unserer Tests aufgestellt.

Das Dilemma des Testens

Der Softwaretest war schon immer heikel. Das Management sieht ihn als nicht produktiv an. Wenn die Zeit oder das Budget eng werden, sparen wir dort als erstes. Solange der Test primär am Ende des Projekts stattfindet, ist diese Einsparung sehr riskant, solange wir uns noch kein abgesichertes Urteil über den Zustand der Software machen konnten. Diese Entscheidung kann (und wird) sich später rächen.

Nicht zuletzt deshalb werden in inkrementell-iterativen Vorgehensweisen die Testaktivitäten bereits parallel zu Analyse- und Entwicklungsaktivitäten vorgesehen. Gerne fällt dabei jedoch unter den Tisch, dass die Testaktivitäten einen großen Anteil an den Gesamtaktivitäten haben, der durch die häufigen Regressionstests, die mit dem iterativen Vorgehen verbunden sind, nur noch verstärkt wird. Der Druck, die meisten Tests zu automatisieren, ist enorm hoch und schon haben wir dadurch ein Softwareentwicklungsprojekt zur Testautomatisierung innerhalb unseres eigentlichen Entwicklungsprojektes. Unsere Testaktivitäten müssen also genauso gewissenhaft geplant werden wie die Programmierung selber.

Trotzdem müssen wir uns irgendwann die Frage stellen, *ob wir genug getestet haben?* Budget und Termindruck zerren an der Qualitätssicherung. Solange die Risiken nur schwer abzuschätzen sind, können wir dieses Dilemma nicht befriedigend auflösen. Haben wir aber unsere Risiken im Griff, können wir Ursache und Wirkung aufzeigen und haben so eine Entscheidungsgrundlage.

Die Lösung des Dilemmas: ein Fehlermodell

Ein Weg aus diesem Dilemma ist der Einsatz von Fehlermodellen. Unter einem *Fehlermodell* verstehen wir ein quantitatives, empirisches Verfahren zur Bewertung der Qualität eines Softwareproduktes, das gleichzeitig ein *Aufwandsschätzmodell* für die dazu notwendigen Testaktivitäten ist. Das Verfahren baut auf der Tatsache auf, dass in der Softwareentwicklung unausweichlich eine bestimmte Anzahl von Fehlern auftritt. Diese Fehleranzahl muss zu einem überwiegenden Teil reduziert werden, um ein Produkt mit einem vorab definierten Qualitätsniveau ausliefern zu können.

Das Ziel ist es nun, mit Hilfe dieses empirischen Ansatzes den Grad der Ungewissheit über die Produktqualität zu vermindern und gleichzeitig den für die zu erreichende Qualität notwendigen Testaufwand zu ermitteln. Dabei ist zu beachten, dass – wie bei allen Modellen – mit Annahmen gearbeitet wird und daher alle Aussagen immer unter Berücksichtigung dieser Annahmen zu interpretieren sind. Wie bei jedem Schätzverfahren sind Unsicherheiten zu berücksichtigen.

Das *Aufwandsschätzmodell* berücksichtigt die Aufwände für die Testfall-Erstellung und gegebenenfalls die Testautomatisierung, aber auch die Testdurchführung, Testergebnisprüfung, Fehlerkorrektur und Wiederholungstests korrigierter Fehler. Die Aussagen betreffen also nicht nur die Aktivitäten einer Qualitätssicherungsgruppe sondern auch die Entwicklung direkt.

Ein Fehlermodell beantwortet daneben weitere Fragen, die sich im Zusammenhang mit den Managementaspekten der Qualitätssicherung stellen:

- Welche Aufwände sind für die Fehlerkorrekturen sowie deren Wiederholungstests einzuplanen?
- Wie viele Fehler sind zu erwarten und wurde diese Anzahl an Fehlern auch gefunden?
- Wie viele (schwerwiegende) Fehler sind noch im Produkt versteckt?
- Wo stehen wir im Projekt-Testverlauf und wann kann der Test beendet werden?

Antworten auf diese Fragen liefern die Abschätzungen aus einem Fehlermodell, die somit auch die quantitativen Argumente für die Managementdiskussionen liefern. Wohlgemerkt: Diese Abschätzungen geben *keine* Garantie, aber ein sicheres Gefühl. Die im Abschnitt

die autoren

Dietmar Kropfitsch (E-Mail: dietmar_kropfitsch@eunet.at) ist selbständiger Berater für Qualitäts-, Konfigurations- und Projektmanagement und Testen in leitenden Positionen internationaler technischer und kaufmännischer Großprojekte.

Uwe Vigenschow (E-Mail: uwe.vigenschow@oose.de) ist Berater und Trainer der oose.de GmbH in Hamburg mit langjähriger Berufserfahrung in der technischen und kaufmännischen Softwareentwicklung. Seine Interessenschwerpunkte sind Objektorientierung und Qualitätsmanagement.

Abschätzung der Programmgröße

Die Problematik der Ermittlung der Größe der Software begleitet jede Aufwandsschätzung eines Softwareprojekts. Es ist möglich, von allen Schätzverfahren, die nicht auf *Lines fo Code (*LOC) basieren, in *LOC* umzurechnen. Als Beispiel sei die *Function-Point*-Zählung erwähnt. Für einige populäre Sprachen haben wir die Werte aus den Tabellen von Capers Jones (vgl. [Jon01]) angegeben.

Programmiersprache	Level	Mittlere Anzahl der Statements pro *Function Point*
Standardwert 3. Generation	4.00	80
Standardwert 4. Generation	16.00	20
Standardwert 5. Generation	70.00	5
C	2.50	128
C++ und Java	6.00	53
PERL	15.00	21
SQL	25.00	13
Visual Basic 5	11.00	29

Mit zunehmendem Level der Programmiersprache sinkt die Anzahl der *Statements*, um einen *Function Point* zu programmieren. Von Krasemann wurden Aktualisierungen, Vereinfachungen und als Erweiterung der Bezug zu *Widget Points* entwickelt (vgl. [Oes01]). Es gibt also gute Möglichkeiten die Programmgröße vorab mit ausreichender Genauigkeit auf Grundlage von Analyseergebnissen abzuschätzen.

Kasten 1

„Dilemma des Testens" angesprochenen Risiken werden kalkulierbar.

Das Basis-Fehlermodell für System- und Integrationstests

Die Grundlage für jedes Fehlermodell bilden empirische Werte. Deshalb können wir Ihnen hier nur den Ausgangspunkt vorstellen. Je mehr Erfahrungen Sie in Ihren Projekten damit sammeln, desto aussagekräftiger werden Ihre Modelle. Fehlermodelle vermindern zwar den Grad der Ungewissheit, aber es wird eben mit *Annahmen* gearbeitet. Versuchen Sie daher, Ihre Firmen-, Abteilungs- und Projektmodelle so individuell wie möglich zu entwerfen.

Das Basis-Fehlermodell, mit dem wir gute Erfahrungen gemacht haben, baut auf den im Folgenden beschriebenen Parametern auf.

Fehleranzahl je 1.000 „Lines of Code" und Gesamtfehlerzahl

Auf der Basis der erwarteten Größe der Anwendung und eines angenommenen Werts für die Fehleranzahl je 1.000 Zeilen Code extrapolieren wir die Gesamtfehlerzahl der Anwendung durch einfache Multiplikation.

Die angenommene Fehleranzahl je 1.000 Zeilen Code kann auch als Komplexitätsfaktor interpretiert werden: Eine höhere Anzahl der Fehler wird bei höherer Komplexität des Programms gewählt, eine geringere Anzahl bei niedriger Komplexität. Details, wie man vorab aus Analyseergebnissen die Programmgröße abschätzen kann, finden Sie in **Kasten 1**.

Anteil in der Testphase zu findender Fehler

In jeder Phase des Entwicklungsprozesses, so auch in der Testphase, kann nur ein bestimmter Anteil der Gesamtfehlerzahl gefunden werden. Dieser Anteil wird als Prozentsatz mit der Gesamtfehlerzahl multipliziert und reduziert so die Fehlerzahl, die in der Testphase gefunden werden sollte.

Mit diesem Parameter sollte auch der Reifegrad des Entwicklungsprozesses in die Berechnungen einfließen. Was ist damit gemeint?

Ganz einfach! Es ist zu prüfen, ob ausreichend genaue Spezifikationen vorliegen, ob Analyse-, Design-, Code-Reviews und Unit-Tests durchgeführt werden. Wenn dies der Fall ist, können wir den Anteil der im Test zu findenden Fehler unverändert lassen oder sogar verringern, ohne die Produktqualität zu gefährden. Werden die genannten Qualitätssicherungsmaßnahmen nicht oder nur zum Teil durchgeführt, ist es ratsam, mehr in den Integrations- oder Systemtest zu investieren. Um die Produktqualität sicherzustellen, sollte der Anteil zu findender Fehler erhöht werden.

Sie sehen also, der Anteil in der Testphase zu findender Fehler kann als Prozessparameter betrachtet werden.

Testabdeckung

Nun haben wir noch die Möglichkeit die Testabdeckung zu bestimmen. Mit der Festlegung dieses Parameters betreiben wir Risikomanagement. Mit der Wahl einer höheren Testabdeckung gehen wir weniger Risiko ein, mit der Wahl einer geringeren Testabdeckung akzeptieren wir ein höheres Risiko.

Eine Testabdeckung unter 100% reduziert entsprechend unsere zu findende Fehleranzahl. Auf diese Art und Weise nehmen wir die Teststrategie – also die Bewertung, welche Teile kritisch sind und welche nicht – mit in das Fehlermodell auf. Damit haben wir die endgültige Anzahl im Test zu findender Fehler ermittelt.

Anzahl der Testfälle, um einen Fehler zu finden

Des Weiteren benötigen wir eine Abschätzung, wie viele Testfälle notwendig sind, um einen Fehler zu finden. Dieser Wert lässt sich nach ein bis zwei Testdurchläufen gut ermitteln. Solange dies nicht der Fall ist, können wir vorerst von der Annahme ausgehen, dass sechs bis zehn Testfälle notwendig sind, um einen Fehler zu finden. Jetzt lässt sich die Anzahl der notwendigen Testfälle durch Multiplikation mit der Fehleranzahl errechnen. Mit der Anzahl der notwendigen Testfälle haben wir nun die Basis zur Errechnung unserer Testaufwände ermittelt.

Aufwandszahlen zur Erstellung und Durchführung von Tests

Als nächstes benötigen wir Mittelwerte für die Aufwände zur Erstellung eines Testfalls und für dessen Durchführung. Jetzt lassen sich erste Aufwandsschätzungen für die Testfallerstellung und -durchführung leisten.

Aufwandszahlen für die Fehlerbehebung

Nicht vergessen sollten wir auch, den Aufwand, der zur Fehlerbehebung notwendig ist, festzulegen. Damit geben wir ▸

Die Formeln für das Black-Box-Fehlermodell

Erwartete Gesamtfehlerzahl	=	*KLOC * Fehler/KLOC*
Erwartete Fehlerzahl	=	*KLOC * Fehler/KLOC * Anteil zu findender Fehler[%] * Testabdeckung[%]*
Anzahl der Testfälle	=	*Erwartete Fehleranzahl * Anzahl Testfälle, um einen Fehler zu finden*
Nettoaufwand Testfallerstellung	=	*Anzahl Testfälle * Aufwand Testfallerstellung*
Nettoaufwand Testfall-durchführung	=	*Anzahl Testfälle * Aufwand Testfalldurchführung*
Nettoaufwand Fehlerbehebung	=	*Erwartete Fehlerzahl * Aufwand Fehlerbehebung*

Kasten 2

dem Projektleiter ein Mittel in die Hand, auch diese Aufgabe abzuschätzen und in der Projektplanung zu berücksichtigen.

Zusammengefasst benötigen wir also die folgenden Parameter:

- Fehleranzahl je *1.000 Lines of Code (KLOC)*
- Anteil in der Testphase zu findender Fehler
- Testabdeckung
- Anzahl notwendiger Testfälle, um einen Fehler zu finden
- Aufwandszahlen für die Erstellung und die Durchführung von Tests
- Aufwandszahlen für die Fehlerbehebung

Die Formeln für die Verknüpfung der Parameter sind **Kasten 2** zu entnehmen.

Erfahrungswerte für die Parameter

Fehleranzahl je 1.000 Lines of Code

In der Literatur werden 40 bis 180 Fehler pro KLOC als Erfahrungswerte über alle Entwicklungsphasen angegeben (**siehe Tabelle 1**). Dabei handelt es sich um die Fehler, die wir zu finden in der Lage sein sollten – also die Fehler, die *vor* dem Einsatz von Testmaßnahmen im Programm sind.

Als Fehler werden üblicherweise alle Probleme mitgezählt (inklusive ungültige, doppelt erfasste, ...). Ungültige und doppelt erfasste Fehler machen ca. 15% bis 20% der Gesamtfehler aus. Wichtig ist es bei allen Zahlenwerten für Fehler nur diese eindeutigen und *echten* Fehler zu berücksichtigen und Mehrfachbeschreibungen ebenso wie Wünsche oder indifferente Angaben auszuschließen.

Humphrey gibt für vier erfahrene Programmierer für dieselbe Programmiertätigkeit von drei Programmen den Wert von 92,6 Fehler/KLOC an (vgl. [Hum95]). Die Werte für die vier Programmierer variieren zwischen 73 bis 113 Fehlern/KLOC. Alle Werte beziehen sich auf Programme, die weder Reviews noch Code-Inspektionen unterzogen wurden. Nach intensiver Review- und Inspektionstätigkeit reduzierte sich dieser Wert nach seinen Angaben auf 49 Fehler/KLOC.

Andere Autoren geben einen Bereich von 30 bis 70 Fehlern/KLOC vor der Übergabe an den Test an. Der praktische Einsatz des Fehlermodells hat gezeigt, dass die Erfahrungswerte der Entwicklungslabors eines großen Systemhauses bei 48 Fehlern/KLOC liegen.

Aufgrund unserer Erfahrungen empfehlen wir als Startwert 50 Fehler/KLOC. Diesen Wert sollten Sie möglichst rasch an ihre Produkt- und Teamrealität anpassen.

Anteil der im Test zu findender Fehler

Um unsere Fehlermodelle an unseren Entwicklungsprozess anzupassen, sollten wir berücksichtigen, in welcher Phase wie viele Fehler gefunden werden sollen. Normalerweise werden keine Listen über gefundene Fehler bei der Analyse und im Design geführt. Um so wichtiger ist es dann zu wissen, dass knapp die Hälfte der Fehler dort bereits gefunden werden sollte.

Tabelle 2 können wir entnehmen, dass ein Testteam in der Testphase zumindest einen Anteil von 23% der Gesamtfehler einer Anwendung finden sollte, um mit nur 3% Restfehlern rechnen zu können. Dies gilt aber nur, wenn vor der Testphase fast 75% der Gesamtfehler beseitigt sind. Als Startwert für den Anteil im Test zu findender Fehler empfehlen wir 25%.

Sie erinnern sich: Wir haben festgestellt, dass in diesem Parameter der Reifegrad des Entwicklungsprozesses berücksichtigt werden soll. Wenn Sie nun wissen, dass in Ihrem Entwicklungsprozess z. B. nur teilweise Reviews durchgeführt werden, empfehlen wir den Startwert von 25% zu erhöhen. Dazu könnten sie festlegen, dass zum Ausgleich dieses Mankos für die Phase 1 beispielsweise 10% von 42,7% = 4,2% und für die Phase 2 beispielsweise 20% von 31% = 6,2%, in Summe also zusätzlich 10,4% der Fehler, d. h. insgesamt 35% statt 25% im Test gefunden werden müssen, um den Restfehlerwert von 3% zu erreichen. Entsprechende Festlegungen können Sie auch für die Phasen iterativer Prozesse treffen.

Testabdeckung

Die Testabdeckung kann vorgegeben werden. In Abhängigkeit von den ermittelten Risiken kann diese natürlich schwanken. Bei einer Abdeckung von 100% wird die ermittelte Fehlerzahl nicht weiter reduziert. Setzen wir die Abdeckung geringer an, verringern wir entsprechend auch die Anzahl der zu findenden Fehler. Bei einer erfahrenen Entwicklungsgruppe mit gut laufenden Kundenprojekten kann die Abdeckung auf 30 bis 60% gesetzt werden; ansonsten sollte sie um die 80% liegen. Bei lebenskritischen Systemen sollte der Wert aber bei 100% bleiben.

Erfahrungswerte für Testfall-Erstellung und -Durchführung

Die Parameter bzgl. der Testdurchführung variieren ebenfalls von Projekt zu Projekt. Gute Mittelwerte sind **Tabelle 3** zu entnehmen. Es handelt sich dabei um heuristische Durchschnittswerte.

Ein Beispiel

Art der Anwendung	Fehler pro KLOC
Einzel-PC-Anwendung	ca. 40
Anwendung mit über 10.000 LOC	ca. 50
Komplexere Grafikanwendung	ca. 80
Selbstcodierte, verteilte Client/Server-Anwendung	Bis zu 180

Tabelle 1: Für verschiedene Anwendungen ergeben sich über alle Entwicklungsphasen aufsummiert die oben angegebene Werte für die Anzahl an Fehlern pro 1.000 LOC. Es handelt sich dabei um ungefähre Werte, die verschiedene 3GL-Programmiersprachen zusammenfassen.

Entwicklungsphasen		Prozentsatz Teilphase	Gesamtprozentsatz Phase
Phase 1	HLD (Analyse)	23.5 %	
	LLD (Design)	19.2 %	42.7 %
Phase 2	Code & Unit-Test	31.0 %	31.0 %
Phase 3 – Test	Funktions-/ Integrationstest	15.3 %	
	Systemtest	7.0 %	
	Validierung/ Verifikation (UOST – „User Oriented Systemtest")	1.0 %	23.3 %
Restfehler		3.0 %	3.0 %

Tabelle 2: Literaturwerte, bestätigt durch mehr als 15.000 Projekte eines großen Systemhauses

Die Codegröße unseres Beispiels beträgt 131 KLOC. Da keine Erfahrungswerte für die Fehler/KLOC verfügbar sind, wird mit 50 Fehlern/KLOC gestartet.

Der Entwicklungsprozess (V-Modell) hat in der Analyse- und Designphase einige Schwächen. Daher legen wir fest, dass 20% der Fehler aus Phase 1, also 8,5 % zusätzlich zu finden sind (laut Tabelle 2 sind 42,7% zu finden), um die Qualität von nur 3% Fehlern im Echteinsatz mit realistischer Wahrscheinlichkeit erreichen zu können. In der Phase 2 – Codierung und Unit-Test – sind umfangreiche Tests geplant. Wir benötigen daher für diese Phase keinen Zuschlag. Daraus folgt, dass der Anteil im Test zu findender Fehler 8,5% + 25,0% = 33,5% betragen sollte. Wir legen 33% fest.

Die Testabdeckung sehen wir mit 60% als ausreichend hoch an.

Des Weiteren benötigen wir fünf Testfälle, um einen Fehler zu finden. Für den Aufwand der Testfallentwicklung und -durchführung greifen wir auf Erfahrungswerte zurück und legen diese mit 30 bzw. 20 Minuten fest. Für die durchschnittliche Fehlerbehebung gehen wir von zwei Stunden aus.

Nun können wir alle Parameter in unser Berechnungsblatt (**siehe Abb. 1**) eintragen und sehen mit einem Blick den notwendigen Netto-Testaufwand sowie den Fehlerbehebungsaufwand.

Aktivität	Aufwandsmittelwert
Anzahl notwendiger Testfälle, um einen Fehler zu finden	6
Entwicklungsaufwand pro Testfall	30 bis 60 min
Aufwand pro Testfalldurchführung	10 bis 20 min

Tabelle 3: Heuristische Durchschnittswerte für Aufwandszahlen für die Testdurchführung. Bei der Anzahl der notwendigen Testfälle müssen auch ungültige oder doppelte mit berücksichtigt werden, da deren Aufwände ebenfalls entstehen.

Test-Ende-Kriterien

Aus dem Fehlermodell lassen sich Kriterien für das Test-Ende ableiten. Aus Min/Max-Betrachtungen mit dem Fehlermodell können wir einen Zielkorridor bezüglich der zu findenden Fehler definieren. Die Anzahl der tatsächlich gefundenen Fehler kann dann mit einer Tabellenkalkulation in Bezug zu den erwarteten Werten gebracht und grafisch dargestellt werden. Wie dies aussehen könnte, kann **Abbildung 2** entnommen werden.

Ein sinnvolles Test-Ende-Kriterium aus dem Fehlermodell ist dann gegeben, wenn

- die Anzahl der gefundene Fehler größer ist als die Anzahl der zu findenden Fehler und
- die Kurve der gefundenen Fehler verflacht, obwohl neue Testfälle zum Einsatz kamen.

In dem Beispiel aus **Abbildung 2** ist es danach fraglich, ob das Testende wirklich erreicht ist, obwohl die Zahl der gefundenen Fehler eigentlich ausreichen würde. Ein Abflachen der Kurve ist nicht wahrzunehmen.

Hinweise für die Projektplanung

Für die vollständige Projektplanung sollten noch ein paar weitere Aspekte berücksichtigt werden. Die berechneten Aufwände für die Testfallerstellung und -durchführung sind Nettoaufwände. Es ist aber immer notwendig auch Abstimmungsgespräche zu führen. Daher empfiehlt es sich diese Nettoaufwände mit einem 20%igen Zuschlag für den Koordinationsaufwand zu versehen.

Bei der Testfalldurchführung und der Fehlerbehebung sind auch die Regressionen zu berücksichtigen, die durch Nachtests entstehen. Diese Fehlerkorrekturen sind aus Erfahrung zu einem gewissen Anteil immer noch fehlerhaft, was ebenfalls in unsere Annahmen eingeht. Als Ausgangswert sollen uns die folgenden Werte dienen:

- 20% der Fehlerbehebungen sind erneut fehlerhaft und
- von diesen Korrekturen sind erneut 20% fehlerhaft
- usw.

Wir müssen also eine mehrfache Regression sowohl bei der Testfalldurchführung als auch der Fehlerbehebung einkalkulieren. Beide Werte sind daher mit einem Faktor zu multiplizieren, der sich aus unseren Regressionsannahmen bestimmen lässt und in unserem Fall die beiden Aufwände um je 25% erhöht. Die Berechnung des Aufwandsfaktors ergibt sich aus dem Grenzwert der geometrischen Reihe für

$$|q| < 1: \lim p \to \infty \sum_{k=0}^{p} q^k = \frac{1}{1-q}$$

in unserem Beispiel also

$$\frac{1}{1-0{,}2} = 1{,}25$$

Planungstechnisch sollten Sie diese Regressionstests als eigene Arbeitspakete ▶

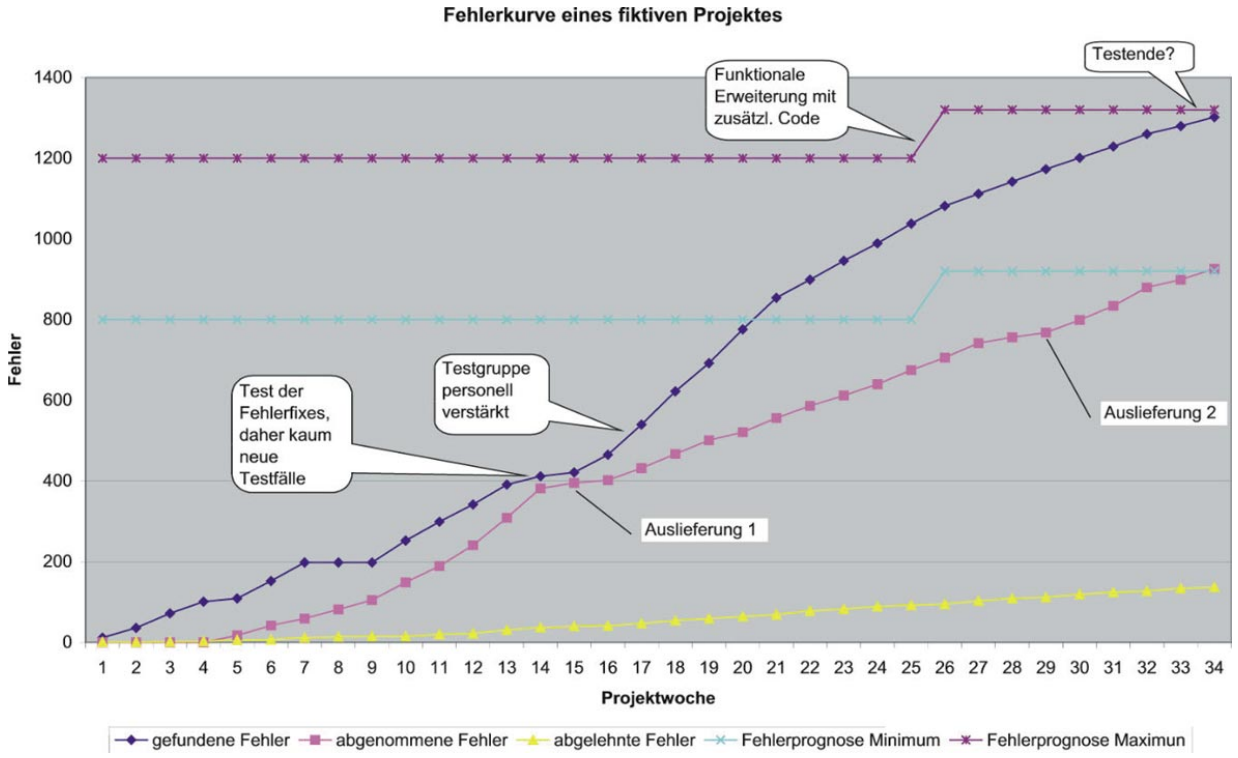

Abb. 2: Beispiel einer Fehlerkurve auf Basis eines Fehlermodells und ermittelter Werte. Spezielle Bereiche der Kurve sind entsprechend markiert.

berücksichtigen – das macht den Testprozess sehr transparent und gut verfolgbar.

Des Weiteren sollten Sie nicht von einem 8-Stunden-Tag ausgehen. Seien wir ehrlich: Die Netto-Zeit wird eher bei sechs Stunden liegen. Lassen Sie sich die Projektdauer durch ein Projektmanagement-Tool ausrechnen, indem Sie die Verfügbarkeit auf 80% setzen.

Weitere Aufwände, die gerne vergessen werden, sind der Aufbau und die Wartung der Testumgebung, Schulung für Testspezifikationstechniken und die Aufbereitung von Testdaten, um nur einige exemplarisch zu nennen.

Weiterentwicklungsmöglichkeiten und Ausblick

Das hier vorgestellte Basis-Fehlermodell kann in verschiedene Richtungen erweitert und ausgebaut werden. So sind sowohl individuelle Anpassungen als auch grundsätzliche methodische Erweiterungen möglich.

Fehlermodell Berechnungsblatt Black-Box-Tests

Parameter

Codegröße [KLOC]	Fehler/KLOC	im Test zu findende Fehler	Testabdeckung	notwendige Testfälle/Fehler	Fachliche Testfälle	Aufwand Testfallentwicklung [min]	Aufwand Testdurchführung [min]	Aufwand je Fehlerfix [h]	Anteil fehlerhafter Fehlerfixes	1 PT hat Stunden
131	50	33%	60%	5	70%	30	20	2	20%	8

Berechnungen

Gesamtzahl Fehler	Anteil verbliebener Fehler	zu findende Fehler	Anzahl zu erstellender Testfälle	Anzahl fachlicher Testfälle	Aufwand fachliche Testfallerstellung [PT]	Aufwand fachliche Testdurchführung [PT]	Aufwand fachl. Fehlerfixing [PT]	Faktor aus fehlerhaftem Fehlerfixing
6550	2162	1297	6485	4539	284	189	284	1,25
				Anzahl technischer Testfälle	Aufwand technischer Testfallerstellung [PT]	Aufwand technische Testdurchführung [PT]	Aufwand tech. Fehlerfixing [PT]	
				1945	122	81	122	

		Aufwand Fehlerfixing [PT]	Gesamtaufwand Test [PT]	Gesamtaufwand [PT]
Summen	Testfälle	Aufwand Testfallerstellung [PT]	Aufwand Testdurchführung [PT]	
	6485	405	270	405	675	1081

PT: Personentag

Abb. 1: Beispiel eines Fehlermodells für Integrationstests in Form eines einfachen Berechnungsblatts einer Tabellenkalkulation. So werden die Aufwände, die beim Integrations- und Systemtest anfallen, sowie die notwendigen Entwicklungsaufwände für die Fehlerkorrektur transparent.

Dies sind Modifikationen des Basis-Fehlermodells (black-box-orientiert), wie die Beschränkung der Betrachtungen auf die kritischen Fehlerklassen, differenziertere, risikomanagementorientierte Ansätze (Auswahl unterschiedlicher Werte für die Parameter, Berücksichtigung der Charakteristika des Systems etc.) oder auch die Anpassung an iterative Verfahren für System- und Integrationstests sowie die Erweiterung zu einem Entwickler-Fehlermodell (white-box-orientiert) mit Klassentests, Kettentests und Modultests.

Ebenso sind individuelle Anpassungen an spezielle Projektrealitäten oder agile Entwicklungsprozesse meist durch kleine Modifikationen am Fehlermodell einfach umsetzbar. ■

Literatur & Links

[Frü00] K. Frühauf, J. Ludewig, H. Sandmayr, Software-Prüfung – Eine Anleitung zum Test und zur Inspektion. Vdf 2000

[Gra87] R.B. Grady, D.L. Caswell, Software Metrics, Prentice Hall 1987

[Hum95] W.S. Humphrey, A Discipline for Software Engineering, Addison-Wesley 1995

[Jon01] C. Jones, die „Programming Language Table" kann bestellt werden unter: www.spr.com/products/programming.htm

[Oes01] B. Oestereich (Hrsg.) et al., Erfolgreich mit Objektorientierung, 2. Auflage, Oldenbourg 2001