Risikogesteuerte Aufteilung von Softwaretestkapazitäten

Walter J. Gutjahr
Institut für Statistik, Operations Research und Computerverfahren
Universität Wien, Universitätsstraße 5/9, A-1010 Wien
Tel. (+43–1) 4076355/190, Fax (+43–1) 4076355/88
e-mail: gutjahr@smc.univie.ac.at

Kurzfassung

Die Arbeit beschreibt ein Verfahren zur optimalen Aufteilung der vorhandenen Testkapazität (d.h. der im Rahmen eines gegebenen Testetats abwickelbaren Testfallmenge) auf die einzelnen Funktionen eines zu testenden Softwareprodukts. Als Bewertungskriterium für einen Aufteilungsschlüssel wird die Ableitbarkeit von präzisen Schätzungen des Fehlerrisikos aus dem Test herangezogen. Die praktische Anwendung des Verfahrens wird anhand eines ausführlichen Beispiels erläutert, und Erweiterungen, wie etwa schrittweise Verfeinerung durch Einbeziehen von Unterfunktionen, werden dargestellt.

1 Problemstellung

1.1 Aufteilung der Testkapazität

Es ist eine bekannte Tatsache, daß das vollständige Testen eines Programms im allgemeinen nicht durchführbar ist ([Myers, 1977]): Man muß sich darauf beschränken, einen winzigen Ausschnitt aus der gesamten Menge möglicher Programmeingaben einem Test zu unterziehen. Hier setzen *Testmethoden* an, deren Ziel es ist, die geforderte Auswahl von Testfällen auf systematische und weitgehend objektive Weise zu treffen. Allerdings liefern auch die geläufigen Testmethoden (siehe etwa [Beizer, 1990], [Liggesmeyer, 1990]) keine Regeln, die die Testfälle komplett festlegen; sie lassen stets Freiheitsgrade offen. Bei den sogenannten *Whitebox-* oder *strukturellen* Testmethoden betreffen diese vor allem die konkrete Wahl von Testeingaben, die ein vorgegebenes Abdeckungskriterium erfüllen; bei den *Blackbox-* oder *funktionalen* Testmethoden ist darüber hinaus gewöhnlich nicht einmal der *Testumfang* durch die gewählte Testmethode eindeutig festgelegt. Wenn etwa Äquivalenzklassentest betrieben wird, so kann die Definition von Äquivalenzklassen eher grob oder aber auch äußerst detailliert vorgenommen werden, was zu einer stark unterschiedlichen Anzahl von Testfällen führen wird. Da Testetats stets beschränkt sind, erhebt sich somit beim funktionalen Testen die Frage, wie man die vorhandene Testkapazität (die verfügbaren Mittel an Arbeits- und Rechenzeit) auf die einzelnen Funktionen des zu testenden Programms aufteilen sollte, ohne bei gewissen Funktionen "Übertest" zu betreiben, bei anderen dagegen den Test wichtiger Eingabekonstellationen zu vernachlässigen.

Um auf diese Frage eine wissenschaftlich fundierbare Antwort geben zu können, muß man zunächst entscheiden, was denn eigentlich als Kriterium für eine "gute" Auswahl von Testfällen herangezogen wird. In der Literatur über Softwaretestmethoden wurden eine Reihe von Kriterien diskutiert (siehe etwa [Gourlay, 1983], [Hamlet, 1989], [Weyuker *et al.*, 1991]). Ohne auf die Vor- und Nachteile dieser einzelnen Kriterien näher einzugehen, wollen wir erwähnen, daß sich für unsere Fragestellung folgendes Kriterium als besonders brauchbar erweist:

(K) Eine Menge $\mathcal{T}$ von Testfällen ist umso besser, je genauere Information der Test mit $\mathcal{T}$ über die operationale Zuverlässigkeit des gegebenen Programms liefert.

Die *operationale Zuverlässigkeit* ist dabei definiert als Wahrscheinlichkeit für eine fehlerfreie Programmausführung bei Eingaben gemäß einer Verteilung, wie sie im Echteinsatz des Programms auftritt. Diese Verteilung wird üblicherweise als *operationale Verteilung* (oder operationales Profil) bezeichnet.

Mit der Wahl des Kriteriums (K) beschreiten wir grundsätzlich einmal denselben Weg wie Thayer, Lipow und Nelson in ihrem Buch über Software-Zuverlässigkeit ([Thayer *et al.*, 1978], Kap. 5 und 6). Während aber das Hauptziel dieser Autoren in einer Quantifizierung von Software-Zuverlässigkeit an sich liegt, soll die Möglichkeit einer solchen Quantifizierung hier primär als Bewertungsmaßstab benutzt werden: Wir gehen von der Überlegung aus, daß ein bestimmter Aufteilungsschlüssel für die verfügbare Testkapazität auf die einzelnen Funktionen des zu testenden Programms gerade dann effizient sein wird, wenn er präzise Zuverlässigkeitsschätzungen erlaubt, was ja zwangsläufig mit der Vermeidung schwerwiegender Testlücken einhergeht. Um dies durch einen Vergleich mit der Qualitätssicherung im Hardwarebereich zu erläutern: Solche und nur solche Prüfverfahren, die die Zuverlässigkeit eines produzierten Geräts zutreffend beurteilen, sind in der Lage, defekte Geräte zu erkennen und auszusondern. Es ist daher sinnvoll, die Präzision der Zuverlässigkeitseinschätzung als Maß für die Güte des Prüfverfahrens heranzuziehen.

1.2 Unzuverlässigkeit und Risiko

Wir gehen hier allerdings in einem wesentlichen Punkt über Thayer, Lipow und Nelson hinaus: In [Thayer *et al.*, 1978] wurden alle auftretenden Programmfehler identisch gewichtet; das zu beurteilende Merkmal des gegebenen Programms ist also einfach die Wahrscheinlichkeit, daß — unter der Voraussetzung eines dem Echteinsatz entsprechenden Anwendungsprofils — während eines Programmlaufs ein Fehler auftritt. Dies ist die *operationale Unzuverlässigkeit* U, die Gegenwahrscheinlichkeit zur oben definierten operationalen Zuverlässigkeit $Z = 1 - U$.

Eine solche Null-Eins-Betrachtung ("das Programm läuft fehlerhaft oder richtig") wurde in den letzen Jahren öfters kritisiert, etwa in [Weyuker *at al.*, 1991] oder in [Sherer, 1992]. In der Praxis können Programmfehler ja äußerst unterschiedliche Auswirkungen und Schweregrade haben. Ein Beistrichfehler in einer Programmeldung mag vielleicht nicht einmal die Mühe einer Korrektur lohnen; umgekehrt gibt es Programmfehler, die zu Verlusten in Millionenhöhe oder sogar zum Tod von Menschen führen. Fehler ist also nicht gleich Fehler. Um dem Rechnung zu tragen, ist es notwendig, einen Fehler mit der Höhe des durch ihn verursachten Schadens zu gewichten.

Wir gelangen damit zum Konzept des *Risikos*. In [Boehm, 1989] wird Risiko durch

$$\text{Risiko} = \text{Fehlerwahrscheinlichkeit} * \text{Schadenshöhe}$$

definiert. Mathematisch allgemeiner ist die Definition in [Sherer, 1992], wo Software-Fehlerrisiko als Erwartungswert der durch Softwarefehler verursachten Kosten angesehen wird:

$$R = E(C),$$

wobei R das Risiko, C die Kosten (= Schadenshöhe) und E den Erwartungswert bezeichnet. Liegen zum Beispiel k mögliche Fehlersituationen $\mathrm{FS}^{(1)}, \ldots, \mathrm{FS}^{(k)}$ mit Wahrscheinlichkeiten $p^{(1)}, \ldots, p^{(k)}$ und Kosten $c^{(1)}, \ldots, c^{(k)}$ vor, so ergibt sich

$$R = p^{(1)}c^{(1)} + \ldots + p^{(k)}c^{(k)}.$$

Man sieht sofort, daß die Unzuverlässigkeit U jener Sonderfall des Risikos ist, wo die Schadenshöhe den konstanten Wert 1 hat.

Das Fehlerrisiko eines Programms unter dem in der realen Anwendung auftretenden Anforderungsprofil bezeichnen wir als *operationales Risiko*. Mit Hilfe dieses Begriffs modifizieren wir nun das Kriterium (K) wie folgt:

(K') Eine Menge $\mathcal{T}$ von Testfällen ist umso besser, je genauere Information der Test mit $\mathcal{T}$ über das operationale Risiko des gegebenen Programms liefert.

Eine Theorie über die optimale Auswahl von Testeingaben, gemessen an Kriterium (K'), wurde in [Gutjahr, 1995] entwickelt. In der vorliegenden Arbeit beschreiben wir die praktische Anwendung des dort entworfenen Verfahrens. Auf das "Warum" wird hier nicht eingegangen; der interessierte Leser sei diesbezüglich auf die oben zitierte Arbeit verwiesen.

Statt dessen stellen wir das "Wie" dar und bringen Lösungsvorschläge für einige bei der konkreten Umsetzung auftretende Fragen.

2 Das Verfahren

Es sei P das zu testende Programm. Wir gehen in folgenden Schritten vor:

Schritt 1:

Die Hauptfunktionen $F_1, \ldots, F_s$ des Programms P werden aufgelistet. In vielen Fällen kann man sich dazu einfach an den Hauptabschnitten der Spezifikation orientieren.

Schritt 2:

Für jede Funktion F_j ($j = 1, \ldots, s$) werden folgende Werte geschätzt:

- Die *operationale Aufrufwahrscheinlichkeit* p_j. Dies ist der relative Anteil der Aufrufe von Funktion F_j an allen Funktionsaufrufen während des Echteinsatzes des Programms.
- Die *Fehlerwahrscheinlichkeit* f_j für die Funktion F_j. Der Wert f_j gibt den (geschätzten) Anteil der Funktionsaufrufe von F_j an, die zu einem Programmfehler führen, also etwa $f_j = 0.02$, falls man einen fehlerhaften Lauf unter 50 Aufrufen von F_j erwartet.
- Die *Fehlerkosten* c_j für die Funktion F_j. Der Wert c_j gibt die (geschätzten) mittleren Kosten an, die bei einer fehlerhaften Behandlung eines Aufrufs von F_j auftreten. Die Geldeinheit, in der diese Kosten ausgedrückt werden, spielt dabei keine Rolle; es kommt nur auf das Verhältnis der Werte $c_1, \ldots, c_s$ untereinander an. (Oft sind für eine bestimmte Funktion F_j sehr unterschiedliche Fehler-Schweregrade möglich, sodaß der Mittelwert c_j relativ wenig aussagt. Wir werden daher in Abschnitt 4.1 eine Technik zur Verfeinerung der Schätzung beschreiben.)

Die Schätzung der Werte p_j, f_j und c_j wird in der Regel subjektiv vorgenommen, eventuell unterstützt durch Erfahrungen mit Vorversionen des betreffenden Programms und durch Schätztechniken, wie sie im

Rahmen der DELPHI-Methode entwickelt wurden (siehe etwa [Linstone und Turoff, 1975]). Es sei jedoch erwähnt, daß, sobald das Programm sich im Echteinsatz befindet, alle drei Parametertypen (p_j, f_j und c_j) objektiv meßbar sind: Die relativen Aufrufhäufigkeiten der einzelnen Funktionen und damit auch die Werte p_j kann man nach entsprechender Instrumentierung des Programms leicht messen; die anderen beiden Parameter ergeben sich im Anschluß daran aus einer detaillierten Fehlerstatistik. Somit sind auch subjektiv vorgenommene Schätzungen aller dieser Werte einer nachträglichen objektiven Korrektur zugänglich, was einen Lernprozeß ermöglicht.

Wo eine subjektive Schätzung unmöglich erscheint, kann man sich durch einen "Voreinsatz", verbunden mit Kostenbewertungen auftretender Fehler, objektive statistische Informationen verschaffen. Der Test selbst wird dann natürlich mit einer Version durchgeführt, in der alle im Voreinsatz erkannten Fehler behoben sind.

Man beachte, daß die Bezugseinheit für die Schätzung der Werte p_i, f_i und c_i ein Funktionsaufruf und nicht etwa ein Programmlauf ist: Bei einem Programm, das in einer Eingabeschleife auf Benutzungsanforderungen wartet, geht jede einzelne Benutzungsanforderung in die Zählung ein. Betrifft eine Benutzungsanforderung zwei (oder mehrere) Programmfunktionen, so sind zwei (oder mehrere) Funktionsaufrufe zu zählen (vgl. Kommentar (2) zum Beispiel in Abschnitt 3).

Schritt 3:

Für jede Funktion F_j werden folgende Werte berechnet:

(a) der *Risikoparameter*

$$g_j = \sqrt{f_j} \cdot c_j,$$

und (nach Ermittlung von $g_1, \ldots, g_s$)

(b) die *optimale Testwahrscheinlichkeit*

$$t_j = \frac{p_j g_j}{p_1 g_1 + \ldots + p_s g_s}.$$

Die Werte $t_1, \ldots, t_s$ sind jene Wahrscheinlichkeiten, mit denen man Testfälle für die Funktionen $F_1, \ldots, F_s$ auswählen muß, um eine Optimierung nach Kriterium K' zu erreichen.

Schritt 4:

Der eigentliche Test wird nach untenstehendem Algorithmus durchgeführt. Er liefert als Nebenprodukt einen (im Sinne der Statistik) unverzerrten Schätzwert für das operationale Risiko des Programms P. Die Anzahl der mit der vorhandenen Testkapazität abwickelbaren Testfälle sei n.

Setze Risiko := 0;
für $i := 1$ bis n:
  wähle eine Funktion F_j zufällig aus $F_1, \ldots, F_s$ unter Benutzung der Auswahlwahrscheinlichkeiten $t_1, \ldots, t_s$;
  wähle einen Testfall x, der zum Aufruf von F_j führt, zufällig gemäß der auf die Funktion F_j eingeschränkten operationalen Verteilung;
  teste das Programm P mit dem Testfall x;
  falls ein Fehler auftritt, bestimme die Schadenshöhe C und setze
  Risiko := Risiko + $(1/n) * (p_j/t_j) * C$.

Wie man erkennt, wird hier Random Testing betrieben, d.h. Testeingaben werden zufällig generiert. Dies ist unabdingbar, wenn ein unverzerrter Schätzwert für das Risiko abgeleitet werden soll. Ist der Anwender nicht in erster Linie an Risikoschätzungen, sondern an einer optimalen Aufteilung der Testkapazität interessiert, so kann die Auswahl der Funktionen und Testfälle ebensogut nichtzufällig erfolgen. Im obigen Algorithmus sind dann Wahrscheinlichkeiten durch relative Häufigkeiten zu ersetzen (vgl. das Beispiel im folgenden Abschnitt). Der sich ergebende Risiko–Schätzwert ist in diesem Fall aus theoretisch–statistischer Sicht nur "annähernd unverzerrt", bei größerer Testfallanzahl ist die Näherung jedoch sehr gut.

Ergeben sich stark unausgewogene Auswahlwahrscheinlichkeiten, etwa eine Konzentration aller Testfälle auf eine oder wenige Funktionen, so ist dies ein Hinweis darauf, daß jene Funktionen, die das Gros der Testfälle in Anspruch nehmen, einem gesonderten Vortest oder eventuell auch formalen Verifikationstechniken zu unterziehen wären. Erfolgreiche Vortests bzw. gelungene Korrektheitsbeweise reduzieren dann die

Schätzwerte für die Fehlerwahrscheinlichkeiten, was zu ausgewogeneren Testverteilungen führt.

3 Ein Beispiel

Zur Erläuterung des Verfahrens diene das folgende einfache Beispiel:

Das zu testende Programm P ist eine Literaturdatenbank. Jeder Datensatz entspricht einer Publikation (Buch, Zeitschriftenartikel usw.). Datensätze werden über eine grafische Benutzungsoberfläche eingelesen, geändert oder gelöscht. Es sollen sowohl Abfragen nach Datensätzen anhand bestimmter Feldinhalte als auch eine Erstellung gefilterter, nach bestimmten Feldern sortierter Listen möglich sein. Eine Ausgabe soll auch auf Drucker erfolgen können. Eine Spezialfunktion generiert eine nach Autor alphabetisch sortierte Literaturliste in einem Format, das eine direkte Übernahme in ein bestimmtes Textverarbeitungsprogramm erlaubt. Das Programm P setzt auf einem Standard-Datenbanksystem auf, der Kern der Datenbankfunktionen braucht also nicht mehr getestet zu werden.

Die zu testenden Version des Programms enthält die Generierung der Literaturliste als neue Funktion, die anderen Funktionen waren bereits in einer Vorversion realisiert und sollen nur noch einem Regressionstest unterzogen werden.

Die Hauptfunktionen von P sind somit:

1	Neuaufnahme
2	Ändern
3	Löschen
4	Abfrage
5	Drucken
6	Literaturliste erzeugen

In Schritt 2 müssen nun operationale Aufrufwahrscheinlichkeiten p_j, Fehlerwahrscheinlichkeiten f_j und Fehlerkosten c_j geschätzt werden. Die operationalen Aufrufwahrscheinlichkeiten p_j werden direkt geschätzt. Die Schätzung der beiden anderen Parameter kann durch Einführung eines Stufencodes erleichtert werden:

NN	sehr niedrig
N	niedrig
M	mittel
H	hoch
HH	sehr hoch

Dies führt im Beispiel auf die Werte von Tabelle 1.

Kommentare:

(1) Da die Funktionen 1 – 5 bereits in der Vorversion implementiert waren, kann man ihre Aufrufhäufigkeiten durch Instrumentierung des Programms empirisch ermitteln und die zugehörigen operationalen Aufrufwahrscheinlichkeiten p_i im selben Verhältnis ansetzen. Die operationale Aufrufwahrscheinlichkeit der neuen Funktion 6 muß geschätzt werden.

(2) Bei der Ermittlung der Aufrufhäufigkeiten für die einzelnen Funktionen beachte man die Regel für die Wertung von Benutzungsanforderungen, die mehr als eine Funktion betreffen (siehe Anschnitt 2): Wenn etwa an das Programm die Aufforderung gerichtet wird, eine Liste aller in der Datenbank erfaßten Publikationen eines bestimmten Autors auf Drucker auszugeben, so zählt das sowohl als Aufruf von Funktion 4 (Abfrage) als auch von Funktion 5 (Drucken).

(3) Die Fehlerwahrscheinlichkeit für die neu implementierte Funktion 6 wurde als "sehr hoch" angesetzt. Die anderen Funktionen sind durch den Einsatz in der Vorversion bereits als stabiler anzusehen, ihre Fehlerwahrscheinlichkeiten wurden daher entsprechend niedriger eingestuft.

(4) Bei der Bewertung der Fehlerkosten wurden alle Funktionen, die Schreibzugriffe auf die Datenbank machen, als besonders kritisch angesehen (Fehlerkosten "hoch"); die Fehlerkosten für die anderen Funktionen wurden als "mittel" eingestuft.

Um die Stufencodes auswerten zu können, muß man ihnen (geschätzte) numerische Werte zuordnen. Im Beispiel geht man etwa davon aus, daß eine "sehr hohe" Fehlerwahrscheinlichkeit einem Versagen

Funktion	operationale Aufrufwahrsch.	Fehler-wahrsch.	Fehler-kosten
1 Neuaufnahme	8%	N	H
2 Ändern	2%	N	H
3 Löschen	3%	NN	H
4 Abfrage	39%	H	M
5 Drucken	18%	M	M
6 Literaturliste	30%	HH	M

Tabelle 1: Bewertung der Programmfunktionen im Beispiel.

Funktion	p_j	f_j	c_j
1 Neuaufnahme	0.08	0.0010	100
2 Ändern	0.02	0.0010	100
3 Löschen	0.03	0.0001	100
4 Abfrage	0.39	0.0500	20
5 Drucken	0.18	0.0100	20
6 Literaturliste	0.30	0.2000	20

Tabelle 2: Numerische Umsetzung der Codes in Tabelle 1.

bei ca. jedem fünften Aufruf gleichkommt, d.h. der Code HH entspricht dem numerischen Wert $1/5 = 0.2$. Analog wurden die übrigen Werte geschätzt.

Ebenso sind den Stufencodes für die Fehlerkosten numerische Werte zuzuordnen. Da es auf die Geldeinheit, in der die Kosten ausgedrückt werden, nicht ankommt, wurde hier der Wert für "hohe Fehlerkosten" willkürlich mit 100 angesetzt und "mittlere Fehlerkosten" in Relation dazu geschätzt (mit dem Wert 20). Damit erhält man insgesamt Tabelle 2 als numerisches Äquivalent zu Tabelle 1.

Mit Hilfe von Tabelle 2 können nun die Risikoparameter g_j und die optimalen Testwahrscheinlichkeiten (bzw. relativen Testhäufigkeiten) t_j berechnet werden. Das ergibt die Werte in den betreffenden Spalten von Tabelle 3.

Wenn nun zum Beispiel $n = 100$ Einzeltestfälle abgewickelt werden können, so erhält man die Werte $100 \cdot t_j$ (siehe letzte Spalte von Tabelle 3)

Funktion	g_j	$p_j g_j$	t_j	$100 \cdot t_j$ (gerundet)
1	3.16	0.253	0.049	5
2	3.16	0.063	0.012	1
3	1.00	0.030	0.006	1
4	4.47	1.743	0.340	34
5	2.00	0.360	0.070	7
6	8.94	2.682	0.523	52
Summe		5.131	1.000	100

Tabelle 3: Risikoparamter g_j, optimale Testwahrscheinlichkeiten t_j und Testfallhäufigkeiten bei $n = 100$ im Beispiel.

als Anzahlen der Testfälle, die auf jede der sechs Funktionen entfallen sollten. Dabei wurde von der zufälligen Auswahl von Testfällen abgegangen, d.h. die Wahrscheinlichkeiten t_j wurden als relative Häufigkeiten interpretiert (vgl. die diesbezügliche Anmerkung in Abschnitt 2).

Treten beim Test Fehler auf und möchte man daraus eine Risikoschätzung für die aktuelle Programmversion ableiten, so sind für jeden Fehler die Kosten zu ermitteln und jeweils mit dem Faktor

$$\frac{1}{n} \cdot \frac{p_j}{t_j} = \frac{1}{100} \cdot \frac{p_j}{t_j}$$

zu multiplizieren, wobei j die Nummer der Funktion ist, zu der der betreffende Testfall gehört. Falls etwa in der Funktion "Ändern" ein Fehler mit einem geschätzten Schadensausmaß von 200 Geldeinheiten aufgetreten ist, so ist er mit einem Beitrag von

$$200 \cdot \frac{1}{100} \cdot \frac{0.020}{0.012} = 3.34$$

Geldeinheiten zu veranschlagen. Aufsummiert über jedes Fehlerauftreten (mehrfach auftretende Fehler werden mehrfach gezählt) ergibt das den gesuchten Risiko-Schätzwert.

4 Erweiterungen

4.1 Detaillierte Schätzung des Risikoparameters

Es wurde bereits erwähnt, daß eine bestimmte Funktion F_j verschiedenen Fehlermöglichkeiten mit sehr unterschiedlichen Schweregra-

Fehlersituation	Fehlerwahrsch. $f_3^{(i)}$	Fehlerkosten $c_3^{(i)}$
$\mathrm{FS}_3^{(1)}$	0.00009	20
$\mathrm{FS}_3^{(2)}$	0.00001	500

Tabelle 4: Detaillierte Schätzung von g_j für Funktion F_3.

den (Kosten) ausgesetzt sein kann. In diesem Fall ist die summarische Schätzung des Risikoparameters, wie sie in Abschnitt 2 beschrieben wurde, nicht genau genug. Eine detaillierte Schätzung wird folgendermaßen vorgenommen:

(1) Man erstelle eine Liste $\mathrm{FS}_j^{(1)}, \ldots, \mathrm{FS}_j^{(k)}$ von möglichen Fehlersituationen für die Funktion F_j.

(2) Man schätze für jede Fehlersituation $\mathrm{FS}_j^{(i)}$ ihre Wahrscheinlichkeit $f_j^{(i)}$ und die zugehörige Schadenshöhe $c_j^{(i)}$.

(3) Der Risikoparameter ergibt sich zu

$$g_j = \sqrt{f_j^{(1)}\,[c_j^{(1)}]^2 \;+\; \ldots \;+\; f_j^{(k)}\,[c_j^{(k)}]^2}.$$

Man beachte, daß die Fehlerkosten jeweils zu quadrieren sind, sodaß sich nach dem Ziehen der Quadratwurzel als Einheit für den Risikoparameter wieder die Währungseinheit ergibt, in der die Fehlerkosten ausgedrückt wurden.

Beispiel: Die Funktion F_3 ("Löschen") aus Abschnitt 3 unterliege folgenden möglichen Fehlersituationen:

$\mathrm{FS}_3^{(1)}$: der gewünschte Datensatz wird nicht gelöscht,

$\mathrm{FS}_3^{(2)}$: falsche Datensätze werden gelöscht oder verändert.

Wir schätzen die benötigten Parameter wie in Tabelle 4. Daraus erhält man:

$$g_j = \sqrt{0.00009 \cdot 20^2 + 0.00001 \cdot 500^2} = 1.59.$$

4.2 Schrittweise Verfeinerung

Mit dem Verfahren von Abschnitt 2 läßt sich eine Aufteilung der verfügbaren Testkapazität auf die Hauptfunktionen des gegebenen Programms vornehmen. Jede dieser Hauptfunktionen setzt sich im allgemeinen selbst wiederum aus einer Reihe von Unterfunktionen zusammen, und der Gedanke liegt nahe, das Verfahren, eine Ebene tiefer, für die Unterfunktionen einer gegebenen Hauptfunktion zu wiederholen. Dies ist tatsächlich möglich, und zwar mit beliebiger Schachtelungstiefe.

Ziehen wir unser durchgehendes Beispiel heran. Angenommen, die Funktion F_4, "Abfrage", setze sich aus folgenden Unterfunktionen zusammen:

$F_{4,1}$	Anzeige einzelner Datensätze
$F_{4,2}$	Browse-Funktion
$F_{4,3}$	Filter setzen
$F_{4,4}$	Listenausgabe

Man kann nun bedingte operationale Aufrufwahrscheinlichkeiten (normiert auf Summe 1) sowie Fehlerwahrscheinlichkeiten und Fehlerkosten für jede dieser Unterfunktionen schätzen, und daraus Risikoparameter $g_{4,1}, \ldots, g_{4,4}$ sowie optimale Testwahrscheinlichkeiten $t_{4,1}, \ldots, t_{4,4}$ gemäß dem allgemeinen Verfahren von Abschnitt 2 errechnen. Letztere sind — ebenso wie die operationalen Aufrufwahrscheinlichkeiten — als bedingte Wahrscheinlichkeiten, gegeben F_4 wird aufgerufen, zu interpretieren; sie summieren sich also auf den Wert 1. Die Zahlen könnten etwa folgendermaßen aussehen:

$$p_{4,1} = 0.25, \quad p_{4,2} = 0.10, \quad p_{4,3} = 0.50, \quad p_{4,4} = 0.15,$$

$$t_{4,1} = 0.10, \quad t_{4,2} = 0.05, \quad t_{4,3} = 0.70, \quad t_{4,4} = 0.15.$$

In unserem Beispiel entfielen von insgesamt $n = 100$ Testfällen 34 Testfälle auf die Funktion F_4 (siehe Tabelle 3). Multiplikation mit den Zahlen $t_{4,1}, \ldots, t_{4,4}$ und Rundung ergibt 3, 2, 24 bzw. 5 Testfälle für die Unterfunktionen $F_{4,1}, \ldots, F_{4,4}$.

Möchte man einen Risiko-Schätzwert aus den Testergebnissen ableiten, so muß man die im Test auftretenden Schadenshöhen C wiederum gewichten. Während das erforderliche Gewicht im "einstufigen" Fall den

Wert $\frac{1}{n} \cdot \frac{p_j}{t_j}$ hatte, beträgt es im oben geschilderten "zweistufigen" Fall

$$\frac{1}{n} \cdot \frac{p_j}{t_j} \cdot \frac{p_{j,l}}{t_{j,l}}$$

für einen Fehler bei einem zur Unterfunktion $F_{j,l}$ gehörigen Testfall. Tritt etwa in unserem Beispiel bei der Unterfunktion $F_{4,1}$ ("Anzeige einzelner Datensätze") ein Fehler auf, so ist er mit

$$\frac{1}{n} \cdot \frac{p_4}{t_4} \cdot \frac{p_{4,1}}{t_{4,1}} = \frac{1}{100} \cdot \frac{0.39}{0.34} \cdot \frac{0.25}{0.10} = 0.0287$$

zu gewichten. Die Gewichtung im Fall einer Schachtelungstiefe > 2 wird analog vorgenommen, hier kommen weitere Quotienten $p_{j,l,m}/t_{j,l,m}$ usf. als Faktoren zum Gewicht hinzu.

Die hier beschriebene schrittweise Verfeinerung ermöglicht es, sich bei der Anwendung des Verfahrens eng an die hierarchische Abschnittsgliederung des Spezifikationsdokuments anzulehnen.

5 Ausblick

Wir haben ein Verfahren beschrieben, mit dem die effizienteste Aufteilung einer gegebenen Kapazität an Arbeits– und Rechenzeit auf den Test der einzelnen Funktionen eines Softwareprodukts ermittelt werden kann. Vorausgesetzt wurde dabei, daß sich vor Testbeginn Aufrufwahrscheinlichkeiten im Echteinsatz, Fehlerwahrscheinlichkeiten sowie Fehlerkosten zumindest grob schätzen lassen. Es wurde auch dargelegt, wie sich der Ansatz auf die Situation einer hierarchischen Funktionsgliederung übertragen läßt, und auf welche Weise aus den Testergebnissen genauere Schätzungen des Fehlerrisikos abgeleitet werden können.

Eine abschließende kurze Bemerkung bezieht sich auf ein Nebenergebnis des Verfahrens, das sich nichtsdestoweniger als sehr vorteilhaft herausstellen könnte, nämlich den innerbetrieblichen *Lerneffekt*, den eine Anwendung der dargestellten Techniken fast zwangsläufig mit sich bringt. Wenn Software–Hersteller durch wiederholten Vergleich von geschätzten mit tatsächlich aufgetretenen Fehlerkosten ihre Fähigkeit verbessern, Fehlerrisiken im vorhinein quantitativ zu beurteilen, werden sie vermutlich in die Lage versetzt, mit dem betriebswirtschaftlich wichtigen Faktor der Software–Fehlerkosten angemessener umzugehen. Dies kann eine in Summe kostengünstigere Software–Produktion ermöglichen.

Literatur

Beizer, B. *Software Testing Techniques*, 2. Auflage, New York: Van Nostrand Reinhold, 1990. 550 Seiten. ISBN 0-442-20672-0.

Boehm, B., Software risk management. In: *Proc. ESEC*, Warwick, UK, Sept. 1989, S. 1–19.

Gourlay, J. S., A mathematical framework for the investigation of testing. In: *IEEE Trans. Software Eng.* (1983) Band SE-9, S. 686–709.

Gutjahr, W. J., Optimal test distributions for software failure cost estimation. In: *IEEE Trans. Software Eng.* (1995) Band SE-21, S. 219–228.

Hamlet, R. G., Theoretical comparison of testing methods. In: *Proc. Third Symposion on Testing, Analysis, and Verification*, Key West, Florida, Dez. 1989, S. 28–37.

Liggesmeyer, P. *Modultest und Modulverifikation – State of the Art.* Mannheim, Wien, Zürich: BI Wissenschaftsverlag, 1990. 321 Seiten. ISBN 3-411-14361-4.

Linstone, H. A. und Turoff, M. *The Delphi Method.* London: Addison–Wesley, 1975. 620 Seiten.

Myers, G. J. *The Art of Software Testing.* New York: Wiley, 1977.

Sherer, S. A. *Software Failure Risk.* New York, London: Plenum Press, 1992. 276 Seiten. ISBN 0-306-44293-0.

Thayer, Th. A., Lipow, M., und Nelson, E. C. *Software Reliability.* Amsterdam: North–Holland, 1978. 311 Seiten. ISBN 0-444-85217-4.

Weyuker, E. J., Weiss, S. N., und Hamlet, D., Comparison of program testing strategies. In: *Proc. Symp. on Testing, Analysis, and Verification*, Victoria, British Columbia, Okt. 1991, S. 1–10.