Empirische Analyse der Leistungsfähigkeit von strukturorientierten Testmethoden

Eike Hagen Riedemann
Universität Dortmund, Informatik I
riedeman@ls1.cs.uni-dortmund.de

1. Einleitung

Mit der vorliegenden Untersuchung[1] soll die Effektivität und Effizienz von strukturorientierten Softwaretestmethoden bestimmt werden, d.h. der Nutzen bei der Fehleraufdeckung im Vergleich zum Aufwand für die Testdatengenerierung. In der Literatur findet man zu diesem Thema ähnliche Studien und Experimente aus den 80er Jahren mit Fehleraufdeckungsquoten von 21% bis 92% (siehe *Ntafos (1984)*, *Girgis;Woodward (1986)*, *Howden (1978)*). Der große Nachteil dieser Untersuchungen besteht darin, dass wenige, relativ kleine Programme getestet wurden, die eher wissenschaftliche und nicht kommerzielle Probleme lösen. Außerdem treten bei diesen Programmen keine Datenbankzugriffe und keine Probleme durch verteiltes Rechnen auf. Daher wurde in der vorliegenden Untersuchung ein Warenwirtschaftssystem untersucht, das seit Juni 1993 im produktiven Einsatz ist und eine Client-Server-Architektur hat. Das Warenwirtschaftssystem besteht zu 95 % aus Cobolprogrammen und zu 5 % aus RPG-Programmen. Zu Beginn bestand das System aus 479 Programmen, zum heutigen Zeitpunkt aus ca. 3000 Programmen.

2. Vorgehensweise der Untersuchung von Testkriterien und Fehlern

In der vorliegenden Studie wurden vier kontrollflussbezogene Testmethoden bzw. Testkriterien untersucht:

- **Anweisungsüberdeckung** (C_0-Überdeckung)
- **Zweigüberdeckung** (C_1-Überdeckung)
- **Segmentpaareüberdeckung**
- **Schwache Grenze-Inneres-Überdeckung**

und drei datenflussbezogene Testkriterien:

- **Alle Definitionen**
- **Alle Entscheidungsreferenzen / einige Berechnungsreferenzen** (kurz: **Alle E- / einige B-Referenzen**)
- **Alle B- / einige E-Referenzen**

(Genaueres zur Definition der Testkriterien: siehe *Riedemann (1997)*.)

Da uns keine aktuellen Fallstudien vorlagen und uns das Personal bzw. die Zeit für Experimente fehlte, haben wir uns für eine **formale Analyse** entschieden, um Testkriterien und damit gefundene Fehler in Programmen zu untersuchen. Dabei wird für jeden Fehler bestimmt, welches (erfüllte) Testkriterium diesen Fehler **zuverlässig** aufdecken würde, d.h. dass *jeder* Test, der das betrachtete Testkriterium erfüllt, diesen Fehler **aufdecken** muss. Letzteres bedeutet, dass beim Testablauf der fehlerhafte Programmabschnitt ausgeführt wird und sich ein Ergebnis beobachten lässt, welches vom spezifizierten (Soll-)Ergebnis abweicht.
Bei Tests im Rahmen der formalen Analyse gibt es also zwei Fälle:

Fall 1): Man findet einen Test, der den Fehler *nicht* aufdeckt, aber das betrachtete Testkriterium erfüllt. Dann besagt die formale Analyse: das Testkriterium deckt den Fehler nicht (zuverlässig) auf. In diesem Fall sind die anderen Testkriterien für diesen Fehler zu untersuchen. Dabei kann man sich die partielle Ordnung der Testkriterien zu Nutze machen und die Testkriterien in folgender Reihenfolge betrachten:
Zu Beginn wird mit der jeweils „schwächsten" Methode der Menge der kontrollfluss- und datenflussbezogenen Testkriterien getestet. (Dies sind die „Anweisungsüberdeckung" und das Kriterium „Alle Definitionen".) Wird der dokumentierte Fehler aufgedeckt, wird der Test abgebrochen. Wird der Fehler nicht aufgedeckt, wird jedes der nächst stärkeren Kriterien angewandt, also nach der „Anweisungsüberdeckung" die „Zweigüberdeckung" und die „Schwache Grenze-Inneres-Überdeckung" sowie nach der „Zweigüberdeckung" die „Segmentpaareüberdeckung", nach dem Testkriterium „alle Definitionen" sowohl „alle E-/ einige B-Referenzen" als auch „alle B-/ einige E-Referenzen".
Diese Vorgehensweise ist korrekt, da Fehler, die mit einem Testkriterium zuverlässig aufgedeckt werden, mit den stärkeren Testkriterien ebenfalls zuverlässig aufgedeckt werden.

Fall 2): Man führt einen Test aus, der den Fehler *aufdeckt* und das betrachtete Testkriterium erfüllt. Dann ist ein weiterer Test zu finden, der den Fehler *nicht* aufdeckt, *oder* es ist zu beweisen, dass alle Tests, die das betrachtete Testkriterium erfüllen, den Fehler aufdecken[2].

3. Fehler aus dem Fehlermeldungssystem und ihre Klassifikation

Um die Ergebnisse der formalen (Fehler-)Analyse des Warenwirtschaftssystems verallgemeinern zu können, sind die gemeldeten 487 Fehler aus der Wartungsphase klassifiziert worden, und zwar in Anwenderfehler, Datenfehler und Programmfehler. Als **Anwenderfehler** gelten Fehler, bei denen durch Fehleingaben (z.B. falsche Werte) ein falsches Ergebnis erzeugt wird. Zudem kommt es auch häufig zu Fehlern, denen eine

[1] basierend auf der Diplomarbeit *Austel (2001)*

[2] Der Beweis ist mit Mitteln der symbolischen Ausführung des Programmteils zu erbringen (siehe z.B. *Riedemann (1997)*, S. 283 und Kap. 12.3).

Fehlinterpretation der Funktionalität durch den Anwender zugrunde liegt. **Datenfehler** sind Fehler, die im betrachteten Warenwirtschaftssystem i. allg. durch Altdaten oder ihre fehlerhafte Übernahme hervorgerufen werden. Die Ursache dafür ist meistens ein **Programmfehler**: Das sind fehlerhafte Konstrukte im Quellcode der Programme, die fehlerhafte Ergebnisse produzieren (können). Sie werden weiter unterteilt in *Ausführungsfehler* und *Codierfehler* im engeren Sinne.

Es macht nur Sinn, *Ausführungsfehler* bezüglich des Nutzens von (strukturorientierten) Testmethoden zur Aufdeckung dieser Fehler genauer zu studieren[3]. Daher haben wir die Ausführungsfehler noch nach der „technischen" Fehlerart (Fehler in Programmkonstrukten) klassifiziert und uns dabei für eine Einteilung in fünf Klassen entschieden (vgl. Tabelle 3.1, erste Spalte).

Aufgrund der vorgenommenen Zuordnung der verbleibenden 80 Ausführungsfehler auf die fünf programmtechnischen Fehlerklassen ergibt sich die in Tabelle 3.1 dargestellte Verteilung. Man beachte, dass etwa die Hälfte aller Fehler Logikfehler sind.

Fehlerklasse	**Anzahl**	**Anteil in %**
Definition/Deklaration	19	23,75
Logik	42	52,50
Referenzierung	11	13,75
Ausgabe	5	6,25
Übergabe	3	3,75
Fehlergesamtzahl	**80**	**100,00**

Tabelle 3.1: Fehlerverteilung der ausgewählten Ausführungsfehler

4. Ergebnisse und ihre Bewertung

Die Fähigkeit der betrachteten sieben Testkriterien, Fehler zuverlässig aufzudecken, ist summarisch in Tabelle 4.1 dargestellt.

	Anweisungs-überdeckung	**Segmentpaare-überdeckung**	**Alle Definitionen**
aufgedeckte Fehler			
Anzahl	17	1	10
Prozentsatz	21,25%	1,25%	12,5%

Tabelle 4.1: Fehleraufdeckungsfähigkeit der betrachteten strukturorientierten Testkriterien

Die Tabellenwerte sind wie folgt berechnet worden:
Ein aufgedeckter Fehler wurde nur bei den ersten (schwächsten) Testkriterien gezählt, die den Fehler in der (in Kap. 2, Fall 1) angegebenen Testreihenfolge aufdecken. Anschließend wurden die Werte pro Testkriterium summiert. Das Kriterium *Anweisungsüberdeckung* deckt also beispielsweise insgesamt 17 der 80 Fehler auf, die *Segmentpaareüberdeckung* nur *einen zusätzlichen* Fehler, der von den schwächeren Kriterien nicht aufgedeckt wird. Alle anderen Kriterien finden keine zusätzlichen Fehler.

Aus den Ergebnissen der Tabelle 4.1 ergibt sich folgendes: Eines der schwächsten Testkriterien - die Anweisungsüberdeckung - deckt (bis auf eine Ausnahme) alle Fehler auf, die überhaupt mit den hier betrachteten strukturorientierten Testkriterien gefunden werden können. Also ist das Testen gemäß der Anweisungsüberdeckung die effektivste und effizienteste[4] Testmethode unter den strukturorientierten Testmethoden. Andererseits liegt die Fehleraufdeckungsquote dieser Kriterien maximal bei enttäuschenden 22,5 %.[5] Diese Ergebnisse zeigen die geringe Wirksamkeit der betrachteten Kriterien: Mehr als 3/4 aller Fehler werden damit nicht zuverlässig gefunden. Sie empfehlen sich aber aufgrund der möglichen Automatisierung und des relativ geringen Aufwands zur Testdatenerzeugung.

Ausgehend von der Verteilung der betrachteten 80 Fehler auf die fünf „programmtechnischen" Fehlerklassen (wie in Tabelle 3.1 angegeben), stellt sich folgende Frage:

Wie ist die prozentuale Fehleraufdeckungsquote der Testkriterien pro Fehlerklasse?

Diese Verteilung ist in Tabelle 4.2 angegeben, wobei bei der Segmentpaareüberdeckung wieder nur der zusätzlich aufgedeckte Fehler gezählt wird (und leere Felder bedeuten, dass keine zusätzlichen Fehler gefunden werden).

	Anweisungs-überdeckung	**Segmentpaare-überdeckung**	**Alle Definitionen**
Fehlerklasse			
Definition	26%		16%
Logik	17%	2%	7%
Referenzierung	36%		27%
Ausgabe	0%		0%
Übergabe	33%		33%

Tabelle 4.2: Fehleraufdeckungsquote der betrachteten Testkriterien pro Fehlerklasse

Auffallend ist der geringe Anteil von aufgedeckten Logikfehlern (17% + 2%), obwohl diese etwa die Hälfte aller (Ausführungs-)Fehler ausmachen (vgl.

[3] *Anwenderfehler* sind mit besseren Schulungen und/oder Benutzerhandbüchern zu vermeiden, bei *Datenfehlern* ist der ursächliche Programmfehler zu finden. *Codierfehler* werden in der Regel vom Compiler erkannt.

[4] Die Anzahl der notwendigen Testdaten ist höchstens linear in der Anzahl n der Zweige bzw. Segmente eines zu testenden Programms. Bei den stärkeren strukturorientierten Testkriterien kann die Anzahl der notwendigen Testdaten dagegen quadratisch oder sogar exponentiell mit n steigen (s. *Riedemann (1997)*, S. 257 f.).

[5] bei Kombination von *Anweisungsüberdeckung* und *Segmentpaareüberdeckung*: (21,25 + 1,25) % = 22,5 %.

Tabelle 3.1: Logikfehler). Diese Fehler werden durch *fehlende* Funktionen und Anweisungen in dem betrachteten Softwareprodukt hervorgerufen. Die Ursache sind die Zwänge der betrachteten Wartungsphase: durch neue Funktionalitäten, die häufig durch das Kopieren von bestehenden Funktionen mit Ergänzungen umgesetzt wurden, sind im Laufe des Produktionsbetriebes Fehler erzeugt worden.

Von den 80 betrachteten Fehlern befinden sich 19 in kleinen, 35 in mittelgroßen und 26 in großen Programmen[6]. Ermittelt man nun die *Fehleraufdeckungsquote* der Kriterien *in Abhängigkeit von der Programmgröße*, so ergibt sich z.B. folgendes: Die Fehleraufdeckungsquote ist bei der Anweisungsüberdeckung für kleine Programme 31,6% und für große Programmen nur 7,7%.

5. Zusammenfassung und Empfehlungen

Die vorliegende Untersuchung zeigt, dass ein Test von kommerziellen Dialogprogrammen *allein* mit *strukturorientierten* Testmethoden eine relativ schlechte allgemeine Fehleraufdeckungsquote (beim Warenwirtschaftssystem im besten Fall 22,5 %) und eine noch schlechtere Aufdeckungsquote (maximal 17%+2% = 19%) für die häufigsten Fehler, die Logikfehler, ergibt. Dies steht in klarem Gegensatz zu vielen erfolgreichen Studien der 80er Jahre für kleine Berechnungsprogramme, bestätigt aber ungefähr die Ergebnisse von Howden, der ebenfalls als Erfolgskriterium für Testmethoden die formale Analyse von zuverlässig gefundenen Fehlern heranzieht (vgl. *Howden (1978)*).
Fehler in größeren Programmen sind schwieriger bzw. in kleineren Programmen leichter aufzudecken, d.h. z.B. dass die Fehleraufdeckungsquote für kleine Programme ca. um den Faktor 4 größer ist als bei großen Programmen.

Die vorliegende Untersuchung zeigt folgende **Probleme** des Testens in der Wartungsphase auf:

- Die Analyse eines Großteils der Logikfehler legt die Vermutung nahe, dass die Fehler ihre Ursache in der Entwurfsphase haben. Dort sind einige Funktionalitäten der Anwendung nicht umfangreich genug, falsch oder gar nicht betrachtet worden.
- In Dialoganwendungen ist der Einfluß und die Abhängigkeit von Dateninhalten und Dialogeingaben zu beachten, was bei den hier betrachteten strukturorientierten Testkriterien nicht berücksichtigt wird. Die entsprechenden Tests können einen Überdeckungsgrad von 100% haben – die Fehler werden trotzdem nicht zuverlässig gefunden, da die Datenkonstellation auf den Pfaden entscheidend ist für die Aufdeckung der Fehler. Es hat sich gezeigt: Je größer die Anzahl der Ein-/Ausgangsobjekte ist, desto weniger zuverlässig arbeiten die strukturorientierten Methoden.

Empfehlungen:
Strukturorientierte Testkriterien, die sich nur am Kontroll- oder Datenfluss orientieren, reichen nicht aus. Wegen der Abhängigkeit der Programmergebnisse von den Werten einer Vielzahl von Ein-/ Ausgangsobjekten sind Tests mit datenbezogenen Methoden (s. *Sneed (1986)*) notwendig. Ein sorgfältiger funktionaler Test in der Entwurfsphase ist nützlich, da ein Großteil der betrachteten Fehler dieser Phase zuzuordnen ist.
Um Fehler durch ein falsches *„cut and paste"* bei der Pflege und Wartung eines kommerziellen Softwareprodukts aufzuspüren, sind spezifikationsorientierte Testverfahren anzuwenden. Sie setzen aber entweder eine entsprechend sorgfältig dokumentierte Spezifikation der geänderten Anwendungsfälle voraus oder eine Kenntnis des Testpersonals von allen Parametern und Variablen des Programms mit ihren (Extrem-)Werten, da davon die Fallunterscheidungen, insbesondere Sonderfälle abhängen, die jeweils zu testen sind.

6. Literatur

Austel, F. (2001): *Anwendungsorientierte Analyse und Bewertung von Softwaretestmethoden.* Diplomarbeit, Universität Dortmund, Fachbereich Informatik, 2001

Girgis, M.R.; Woodward, M.R. (1986): "An Experimental Comparison of the Error Exposing Ability of Program Testing Criteria". In: *TAV (1986)*, S. 64-73

Howden, W.E. (1978): "Empirical Studies of Software Validation". In: *Tutorial: Software Testing & Validation Techniques*, E. Miller, W.E. Howden (Eds.), IEEE, S. 280-285

Ntafos, S.C. (1984): "An Evaluation of Required Element Testing Strategies". In: *Proc. 7th International Conference on Software Engineering*, Orlando, S. 250-256

Riedemann, E. H. (1997): *Testmethoden für sequentielle und nebenläufige Software-Systeme*. Teubner, Stuttgart, 512 S.

Sneed, H. M. (1986): "Data Coverage Measurement in Program Testing". In: *TAV (1986)*, S. 34-40

TAV (1986): *Proc. Workshop on Software Testing*. 15.-17. Juli 1986, Banff, Kanada, ACM/IEEE

[6] Für diese Untersuchung haben wir die Programmgröße wie folgt in drei Bereiche eingeteilt:
klein / mittel / groß ⇔ Anzahl der Quelltextzeilen
≤ 1000 / > 1000 und ≤ 3000 / > 3000