FEHLERANALYSE UND FEHLERURSACHEN IN SYSTEMPROGRAMMEN

A. Endres

Kurzfassung

Die während eines internen Tests des Betriebssystems DOS/VS aufgedeckten Programmfehler bilden die Grundlage für eine Untersuchung von Fehlerverteilungen in Systemprogrammen. Aus einer Klassifikation der Fehler nach mehreren Gesichtspunkten, wird auf die mögliche Ursache dieser Fehler geschlossen. Die dabei gewonnenen Erkenntnisse werden angewandt auf eine Diskussion der wichtigsten Methoden zur Verhütung bzw. Aufdeckung von Fehlern.

1. EINLEITUNG

Für alle, die sich die Aufgabe stellen, die Zuverlässigkeit von Software-Produkten zu erhöhen, sollte es von Nutzen sein, möglichst viel darüber zu wissen, welcher Art die Fehler sind, die in tatsächlich geschriebenen Programmen gemacht werden. Fast jeder, der einmal ein Programm geschrieben hat, das nicht auf Anhieb das tat, was es tun sollte, - und das ist ja bekanntlich der Normalfall - wird sich seine persönliche Theorie dafür entwickelt haben, was bei ihm in diesem speziellen Fall daneben ging und warum. Als Folge davon hat man seinen Programmierstil beim nächsten Mal geändert, d. h. man hat die Tricks, mit denen man keinen Erfolg hatte, vermieden oder auf typische Gefahrenstellen ein erhöhtes Maß an Aufmerksamkeit verwendet.

Diesen Lernprozeß, den ein guter Programmierer individuell durch macht, möchte man natürlich auch bei einer größeren Gruppe von Programmierern erleben, wenn nicht sogar bei dem gesamten Berufsstand. Was dafür erforderlich ist, ist daß man die Erfahrungen, die jeder einzelne Programmierer macht, versucht zu generalisieren. Das heißt, man muß sich damit beschäftigen, welche Fehler von einer größeren Zahl von Leuten über eine größere Klasse von Programmen hinweg gemacht werden.

Die bisher veröffentlichten Untersuchungen in dieser Richtung weisen in meinen Augen einige erhebliche Mängel auf. Als Beispiel sei die Arbeit von Moulton und Muller [1] erwähnt. Diese Untersuchung bezog sich auf FORTRAN Programme im Universitätsmilieu (University of Michigan). Zwar wurde eine beachtliche Anzahl von Programmen (ca. 5.000) analysiert, jedoch war die durchschnittliche Programmgröße nur 38 Statements. Was jedoch diese Art von Untersuchungen noch mehr kennzeichnet, ist die Tatsache, daß die Analyse von Fehlerarten sich auf eine rein syntaktische Klassifizierung beschränkt. Dasselbe gilt für die Auswertung von Rubey [2], die er bei einem Vergleich von FORTRAN, COBOL, JOVIAL und PL/I machte. Die Schlußfolgerung, zu der man etwa auf Grund dieser Untersuchungen kommen kann, heißt, daß man sich in FORTRAN vor Assignment und I/O Statements hüten soll. Daß die Probleme normalerweise nicht in der Syntax einer Sprache zu suchen sind, zeigt z. B. die Untersuchung von Boies und Gould [3]. Hier findet man bereits die Schlußfolgerung, daß der Anteil syntaktischer Fehler deutlich unter 15% liegt. Der Versuch, den ganzen Komplex von der Problemdefinition bis zur Codierung in einer vorgegebenen Programmiersprache mit in die Betrachtung hineinzuziehen, findet sich sehr gut dargestellt bei Henderson und Snowdon [4]. Obwohl hier sozusagen nur die Geschichte eines einzigen Fehlers (dazu allerdings in einem vorher als richtig bewiesenen Programm) beschrieben wird, dürfte jedoch diese Art der Untersuchung am erfolgversprechendsten sein.

Der folgende Beitrag basiert auf einer Untersuchung von Fehlern in Systemprogrammen. Die Besonderheit von Systemprogrammen liegt darin, daß sie im Vergleich zu Anwendungsprogrammen eine besonders starke Parameterisierung, ein breites Benutzerspektrum und eine lange Lebensdauer aufweisen. Dies hat nicht nur zur Folge, daß an sie besonders hohe Qualitätsansprüche gestellt werden, sondern auch, daß ihre Struktur sich oft als besonders komplex herausstellt.

Das Ziel dieses Beitrags ist es, sowohl die Frage zu klären, welche fundierten Aussagen überhaupt auf Grund einer derartigen Fehleranalyse gemacht werden können, als auch die Schlußfolgerungen zu präsentieren, die sich aus den speziellen Daten dieser Auswertung ableiten lassen.

2. GEGENSTAND UND METHODE DER UNTERSUCHUNG

Der Gegenstand der Untersuchung waren die bei internen Tests entdeckten Fehler an den im Böblinger IBM Labor entwickelten Komponenten des Betriebssystems DOS/VS (Rel. 28).

Zum besseren Verständnis muß folgende, dieses Projekt betreffende Information vorausgeschickt werden. DOS ist ein Betriebssystem, dessen erste Version etwa 1966 auf den Markt kam. In zuerst viertel- und später halbjährigen Intervallen wurden Erweiterungen bzw. Verbesserungen des Systems freigegeben. Der Änderungsumfang der einzelnen Versionen (sog. Releases) war sehr unterschiedlich. Die Version, die hier zur Diskussion steht, beinhaltete wohl die tiefgreifendsten Änderungen, die überhaupt an diesem System gemacht wurden. Bei den Erweiterungen, die für Release 28 im Böblinger Labor entwickelt wurden, handelte es sich im wesentlichen um folgende Teilprojekte, wobei alle sich auf das Kontrollprogramm im engeren Sinne bezogen:

a) Unterstützung des virtuellen Speicherkonzepts,
b) Erweiterung des Systems von 3 auf 5 Programmbereiche (partitions), incl. variabler Prioritätsvergabe,
c) Unterstützung neuer Kartenein- und Kartenausgabegeräte,
d) Unterstützung eines optischen Anzeigegerätes als Operateur-Konsole,
e) mehrere kleinere Erweiterungen (katalogisierte Prozeduren, Zeitgeber je Bereich, Anpassung für VSAM),
f) Anpassung des Spooling-Subsystems "POWER" an die o. a. Systemänderungen.

Zeitlich parallel zu den erwähnten Erweiterungen wurden andere Zusätze des Kontrollprogramms vor allem im holländischen Labor und neue Assembler, Kompiler und Datenzugriffsmethoden in mehreren Labors u. a. in den USA entwickelt.

Der das System darstellende Code is physikalisch gegliedert in Macros und Moduln. Macros sind jene Routinen, die im ausgelieferten System im Assembler-Macroformat enthalten sind; Moduln sind in übersetzter Form vorhanden. (Da im folgenden diese Unterscheidung nicht wesentlich ist, wird der Begriff Modul benutzt, wenn Modul oder Macro gemeint ist.)

Von den Tätigkeiten im Böblinger Labor wurden etwa 500 Moduln "berührt". Die durchschnittliche Größe dieser Moduln lag bei etwa 360 Instruktionen/Modul, wenn man nur den ausführbaren Code betrachtet, und bei 480 Instruktionen/Modul, wenn man Kommentare mitzählt. Global gesehen, hatte das Projekt folgenden Effekt auf das System:

	Moduln	Instruktionen alt	Instruktionen neu
a) ganz neu geschrieben wurden	169	--	53K
b) alten und neuen Code enthalten	253	97K	33K
c) nur Kommentare geändert	100	7K	--
insgesamt	522	104K	86K

Die angegebenen 190K Instruktionen stellen ausführbaren Code dar. Dazu kommen noch ca. 60.000 Zeilen Kommentare und dergleichen. Alle Moduln und Macros sind im DOS Macro-Assembler geschrieben.

Wie Tabelle 2-1 zeigt, schwankt die Größe der einzelnen Moduln sehr. Ebenso ist der relative Umfang der Änderung je Modul sehr unterschiedlich. Tabelle 2-2 zeigt dies für die 253 Moduln, die sowohl alten als auch neuen Code enthalten. Aus beiden Tabellen zusammen kann man schließen, daß die wohl typischste Projekttätigkeit etwa darin bestand, in einem vorhandenen Modul von ca. 200 Instruktionen etwa 50 Instruktionen zu ändern, bzw. hinzuzufügen. Daß in dieser Situation viele der für das Erstellen fehlerfreier Programme angepriesenen Methoden (Top-down Entwurf, strukturiertes Programmieren usw.) kaum zur Anwendung kommen konnten, dürfte das bisher Gesagte klar gemacht haben.

Das Material für die Untersuchung waren die Aufzeichnungen über diejenigen Fehler, die während einer formellen Testperiode von insgesamt 5 Monaten in den oben angegebenen Moduln gefunden wurden. Die fragliche Testphase umfaßte nur einen Abschnitt, wenn auch den kritischsten, in dem gesamten Testverlauf des Systems. Vorweggegangen waren die Tests, die die jeweils für einen Modul oder für ein Teilprojekt verantwortliche Programmierergruppe dezentral durchgeführt hatte. Jedes Teilprojekt war soweit ausgetestet, daß

es "integrationsreif" war, d. h. die neuen Funktionen waren so vollständig verifiziert worden, wie dies möglich war, ohne daß alle in nicht unmittelbarer Abhängigkeit voneinander stehenden Komponenten auf demselben Entwicklungsniveau waren.

Auch durch den anschließenden Integrationsprozeß evtl. eingeführte Konflikte waren ihrerseits gelöst, so daß der besagte zentrale Test auf einem "lauffähigen" System begann. Ziel dieser Phase war es nun, sozusagen das fertige System mit allen seinen Komponenten in möglichst vielen verschiedenen Konfigurationen und in möglichst vielen verschiedenen Funktionsbeanspruchungen zu testen. Zu diesem Zwecke wurden von zwei Gruppen zwei Arten von Testfällen benutzt. Eine zum Entwicklungsbereich gehörige Gruppe ließ Testfälle, die bereits auf einer früheren Version des Systems gelaufen waren, in möglichst stark geänderter Konfiguration und Systemzusammensetzung erneut laufen (Regressionstest). Eine zweite, unabhängige Gruppe hatte, ausgehend von der externen Beschreibung des Systems, neue Testfälle entwickelt, die eine Abnahme des Systems aus Kundenperspektive simulieren sollten (Beta Test).

Vor der Auslieferung des Systems an die Kunden folgten noch weitere Tests, so z. B. eine Leistungsuntersuchung, spezielle Tests für Datenfernverarbeitung und ein Test im Rechenzentrum von ausgewählten Kunden.

Das Material, das hier untersucht wurde, kann also kein Bild abgeben für alle Fehlerarten, die im Laufe eines Projekts entdeckt werden, sondern nur einen Ausschnitt. Typische Fehler, wie sie in frühen Stadien eines Projekts (vollständig fehlende Routinen), bei Anfängern (Syntaxfehler) oder nach einer hektischen Änderungsphase aufzutreten pflegen, dürften etwas in den Hintergrund treten. Dasselbe gilt für Fehler, die normalerweise durch andere Methoden als durch das Durchführen von Testläufen gefunden werden.

Die von den beiden Testgruppen entdeckten (oder vermuteten) Unregelmäßigkeiten des Systems wurden ihrem äußeren Erscheinungsbild

nach, d. h. nach dem Effekt auf einen bestimmten Testfall, dokumentiert. Diese Information bezeichnen wir als das Problem. Sie wurde der ursprünglichen Entwicklungsgruppe übergeben, diese nahm eine Analyse vor und schrieb (auf demselben, im folgenden als Fehlerprotokoll bezeichneten, Formblatt) eine Antwort (siehe Bild 2-3).

Die Antwort klassifizierte das Problem zunächst nach folgenden 6 Gruppen:

a) Maschinenfehler
b) Bediener- oder Benutzerfehler
c) Verbesserungsvorschlag
d) Duplikat (eines bereits bekannten Programmfehlers)
e) Dokumentationsfehler
f) (noch nicht bekannter) Programmfehler.

Die Verteilung auf die einzelnen Gruppen hängt im allgemeinen von der Art und auch der Organisation eines Projektes ab. So ist z. B. der Anteil der Duplikate um so niedriger, je schneller eine Korrektur eines bereits bekannten Problems der Testgruppe zur Verfügung gestellt wird.

Obwohl auch in den anderen Gruppen durchaus relevante Information enthalten sein kann, wollen wir uns im folgenden auf die Gruppe (f) konzentrieren. Es sind dies die von der Entwicklungsgruppe akzeptierten Fehler im Code. In unserem vorliegenden Falle umfaßte die gesamte Datenbasis insgesamt etwa 740 Probleme, von denen 432 als Programmfehler klassifiziert worden waren.

Es sei hier bemerkt, daß aus der Benutzerperspektive die oben angegebene Aufteilung nicht immer ohne weiteres akzeptabel ist. Für ihn sind Fehler in den Gruppen (a), (d) und (e) ebenso ärgerlich wie die eigentlichen Programmfehler. Wir wollen sie deshalb nicht weiter untersuchen, weil sie ihren Ursprung nicht in der Programmierertätigkeit per se haben.

Der Ordnung halber sei außerdem vermerkt, daß alle erwähnten Fehler vor Auslieferung des Systems behoben waren.

3. MÖGLICHKEITEN UND GRENZEN EINER FEHLERANALYSE

Die für die Auswertung zur Verfügung stehenden 432 Fehlerprotokolle enthalten je Fehler folgende Angaben:

a) Administrative Angaben über die Problementdeckung (benutzte Systemversion, Konfiguration, benutzter Testfall, Datum des Testlaufs, Name des Testers usw.).

b) Beschreibung des Problems.

c) Administrative Angaben über die durchgeführte Korrektur (geänderte Moduln; Datum der Änderung; Name des Programmierers; Systemversion, in die die Korrektur integriert werden soll usw.).

d) Codeschlüssel für Ursache des Fehlers; verursachendes Teilprojekt.

e) Beschreibung der durchgeführten Korrektur.

Sobald man eine derart umfassende Datenbasis vorliegen hat, fallen einem eine ganze Reihe von Fragen ein, die man stellen möchte. Mir erschienen folgende Fragen als sinnvoll:

a) Wo wurde der Fehler gemacht? Wie ist die Verteilung der Fehler nach Moduln? Gibt es Häufungspunkte, d. h. Moduln, die besonders betroffen waren? Wenn ja, was tun diese Moduln? Wie sind sie strukturiert?

b) Wann wurde der Fehler gemacht? In jeder Phase des Entwicklungszyklus' können Fehler gemacht werden, angefangen bei der Festlegung der externen Zielsetzung des Projekts, während der Detailplanung des logischen Aufbaus, während der ursprünglichen Codierungsphase, bei der Korrektur eines Fehlers usw.

c) Wer hat den Fehler gemacht? Man kann dies beziehen einmal auf die Verantwortlichkeit einzelner Gruppen während des Projektzyklus' (Entwurf, Implementierung), aber auch auf einzelne Teilprojekte oder sogar auf einzelne Programmierer.

d) Was wurde falsch gemacht? Welche programmtechnische Teilaufgabe wurde nicht oder falsch gelöst? Die daraus sich ergebende Gliederung nach Fehlerarten kann dann die Basis bilden für folgende weitere Fragen:

e) Warum wurde der betreffende Fehler gemacht? Was hat den Fehler verursacht? Eng damit zusammen hängt (wie später noch gezeigt wird) die Frage:

f) Was hätte getan werden können, um den betreffenden Fehler zu verhüten? Und schließlich:

g) Wenn der Fehler schon nicht verhütet werden kann, durch welche Maßnahme kann diese Fehlerart aufgedeckt werden?

Natürlich kann man diesen Fragenkatalog noch erweitern und ergänzen. Relevant wäre eventuell die Frage: Welche Art von Testfall hat welche Fehlerart aufgedeckt? Auch kann man sich jedwede Kombination der oben angegebenen Fragen als interessant vorstellen, so z. B. (b) und (d) zusammen, was dann hieße: Wann werden welche Fehler gemacht? Man wird mir zugestehen, daß der Fragenkatalog, wie er steht, schon recht umfassend ist. Wären wir in der Lage, auf diese Fragen vollständige und gültige Antworten zu finden, so wären wir für die Zukunft vieler unserer Sorgen enthoben.

Welche Schwierigkeiten bei einem derartigen Unterfangen jedoch auftreten können, und welchen Beschränkungen wir uns daher unterwerfen müssen, sollen die folgenden Bemerkungen verdeutlichen.

Da ist zunächst die Frage, wie man überhaupt feststellen kann, worin der Fehler bestand. Um es gleich vorwegzunehmen: Bei der

beschriebenen Auswertung wurde in der Regel tatsächlicher Fehler gleich durchgeführter Korrektur gesetzt. Daß dies nicht immer ganz richtig ist, rührt daher, daß mitunter der eigentliche Fehler zu tief liegt und der Zeitaufwand oder das Risiko, neue Fehler einzuführen, zu groß sind, um das eigentliche Problem zu lösen. Die Korrektur, die gemacht wurde, hat in solchen Fällen vielleicht nur eine Folge des Fehlers behoben oder um das Problem herumgeleitet. Um die entsprechende größere Exaktheit der Auswertung zu erreichen, müßte man eigentlich, statt sich die Korrekturen anzusehen, einen Vergleich machen zwischen der ursprünglich beabsichtigten Implementierung und der tatsächlich durchgeführten. Dies ist jedoch weder aufwandsmäßig noch von der Sache her durchführbar, da die ursprünglich beabsichtigte Implementierung entweder nicht mehr erkennbar oder nicht mehr anwendbar sein kann.

Aus derselben Überlegung folgt, daß der Modul, an dem die Korrektur angebracht wurde, nicht auch der Modul sein muß, der den Fehler verursacht hat. Oft wird auch die Änderung da gemacht, wo gerade am meisten Platz frei ist.

Für die Fehleruntersuchungen an einem Betriebssystem scheint es mir als typisch, daß die Problembeschreibung sehr selten sehr viel Relevanz hat für Art und Ursache des Fehlers. Abgesehen von den Fällen, wo z. B. ein Bibliothekswartungsprogramm eine Ausgabeliste produziert und man an ihr direkt einige Fehler erkennt, ist im Normalfall der Effekt eines Fehlers ein sehr indirekter. Typische Problembeschreibungen bei einem Betriebssystem sind:

- Das System hängt in einer Schleife.
- Das Gerät X konnte nicht gestartet, die Datei Y nicht gelesen werden.
- Das System stoppt mit ungültigem Operationscode, ungültiger Speicher-, Platten- oder Geräteadresse.
- Der Kartenleser K läuft nur mit halber theoretischer Geschwindigkeit, usw.

Es dürfte auch nicht eindeutig sein, was man beim Zählen von Fehlern als Einheit ansieht. Als Folge eines Problems kann es vorkommen, daß man eine oder 20 Konstanten ändert, eine oder 20 Instruktionen einfügt, an einer oder an 5 Stellen in einem Modul, in einem oder 5 Moduln, usw. Man kann auch nicht ausschließen, daß mit einem Problem gleich andere Probleme mitgelöst wurden, auf die der Programmierer beim erneuten Durchgehen des Programms stieß (oder von denen er insgeheim längst wußte). Bei dieser Auswertung wurde Anzahl der Fehler gleich Anzahl der Probleme gesetzt; wobei Probleme, die auf demselben Fehler beruhen (Duplikate), bereits vorher abgezogen waren.

Unter Beachtung der gerade erläuterten Einschränkungen kann man die Fragen (a), (b), (c) und (d) aus dem vorhandenen Material mit einiger Zuverlässigkeit beantworten.

Bei einigen Fragen muß man allerdings noch Informationen hinzuziehen, die sich nicht direkt aus den Fehlerprotokollen ergeben. Will man z. B. die Frage nach dem Verursacher eines Fehlers bis auf einen einzelnen Programmierer herunterverfolgen, so muß man Information mit heranziehen, die in der Systembibliothek über die Entstehungsgeschichte jedes Moduls vorhanden ist. Es sei hier erwähnt, daß manche Fragen einen gewissen personalpolitischen Explosionsstoff enthalten und man sie daher, wenn überhaupt - nur in einer sehr abgeklärten und sachlich fairen Weise angehen kann.

Während wir im folgenden den Fragenkomplex (b) und (c) ganz ausklammern und (a) nur kurz behandeln, bietet der Fragenkomplex (d) vermutlich die interessanteste Information. Die dabei entwickelte Kategorisierung nach Fehlerarten dient anschließend als Basis für weitergehende Betrachtungen bezüglich der Fragen (e), (f) und (g).

4. DARSTELLUNG EINIGER ERGEBNISSE UND SCHLUSSFOLGERUNGEN

In den folgenden Abschnitten wird eine Auswahl der Information präsentiert, die sich aus der erwähnten Untersuchung ergab. Es sei darauf hingewiesen, daß die tabellarischen Zusammenstellungen oft einen Grad an Detailinformation enthalten, auf die in der unmittelbar folgenden Diskussion nicht vollständig eingegangen werden kann. Ich hielt es dennoch für zweckmäßig, diese Information mit einzufügen, damit man auf sie zurückgreifen kann, wenn die Zusammenfassungen für das Verständnis nicht ausreichen sollten.

4.1 Fehlerverteilung nach Moduln

Diese Information ist in 3 Tabellen zusammengefaßt. Tabelle 4-1 zeigt die Auswirkungen eines Fehlers, bezogen auf die Anzahl der Moduln, die geändert werden mußten. Daß über 85% der Fehler durch Änderung je eines einzelnen Moduls behoben werden konnten, überrascht etwas. Es widerspricht der Vorstellung der Interdependenz der Moduln in einem Betriebssystem und der allgemein so häufig beobachteten Fehleranfälligkeit von Schnittstellen zwischen verschiedenen Moduln.

Tabelle 4-2 zeigt die umgekehrte Aufteilung, nämlich die Anzahl der Fehler, die je Modul gefunden wurden. Dabei wurden Fehler, die wie in Tabelle 4-1 gezeigt, mehrere Moduln betrafen, mehrfach gezählt. Die drei Spitzenreiter mit 28, 19 bzw. 15 Fehlern stellten keine Überraschung dar; es handelte sich dabei um drei der umfangreichsten Moduln des Systems (alle mehr als 3.000 Instruktionen). Der vierte Platz in dieser negativen Reihenfolge wird von einem relativ kleinen, aber als sehr instabil bekannten Modul belegt. Generell dürfte auffallen, daß von 422 geänderten oder neu geschriebenen Moduln nur 202 Fehler aufwiesen (48%).

Klammert man noch die Moduln mit je einem Fehler aus, so ergibt sich, daß 78% der Fehler (400) auf 17% der Moduln (90) fallen.

Tabelle 4-3 enthält diverse Vergleiche der Fehlerhäufigkeit. Dabei ist jedesmal unterschieden zwischen den Moduln, die nur neuen Code enthalten und den Moduln, die sowohl alten als auch neuen Code enthalten. Was auffällt ist folgendes: Das Verhältnis zwischen Moduln mit Fehlern und Moduln insgesamt, also die Fehleranfälligkeit ist dieselbe (48%). Bei der Anzahl der Fehler je Modul scheinen die Moduln mit nur neuem Code zunächst schlechter abzuschneiden als die Moduln mit gemischtem Code. Das Verhältnis dreht sich jedoch wieder um, wenn man den Umfang des neu geschriebenen Codes ansieht. Daß diese Angaben sehr viel Relevanz haben, möchte ich bezweifeln. Sie scheinen jedoch das unter Programmierern weitverbreitete Gefühl zu bestätigen, daß es von einem gewissen Punkt ab besser ist, ein Programm neu zu schreiben als zu versuchen, möglichst viel alten Code zu retten.

4.2 Verteilung nach Fehlerart

Die unter diesem Abschnitt dargebotene Information stellt den eigentlichen Kern der vorliegenden Auswertung dar. Die Zahlen dieses Abschnitts ergaben sich aus einer nachträglich durchgeführten Analyse aller beschriebenen Programmkorrekturen.

Um sinnvolle Aussagen machen und das Material überhaupt darstellen zu können, mußten die vorhandenen Angaben klassifiziert und abstrahiert werden. Dies hat nebenbei den Vorteil, daß die Angaben verständlich werden auch für jemanden, der keine Vorkenntnisse hat bezüglich dieses speziellen Betriebssystems. Der Nachteil ist natürlich ein Verlust an Genauigkeit und Prägnanz.

Wegen der Fülle des Materials ist die Darstellung in den Tabellen 6-1 bis 6-10 zweistufig strukturiert. Die Tabellen 6-1, 6-5 und 6-10 ergeben eine übergeordnete Klassifikation, die hier mit Gruppe A, Gruppe B und Gruppe C bezeichnet ist. Die dazwischen liegenden Tabellen enthalten eine weitere Detaillierung für einige der Angaben in den Gruppen A bzw. B. Bei Gruppe A ist diese zusätzliche Erläuterung nur für 3 von 6 Untergruppen gegeben, bei Gruppe B für 4 von 7.

Alle Zahlenangaben sind Prozentzahlen, und zwar sind alle auf dieselbe Grundmenge von 432 Fehlern bezogen.

Die Aufteilung in die Hauptgruppen A, B und C ist im nachhinein erfolgt und ist wie folgt zu motivieren:

a) Die Fehler in Gruppe A sind problemspezifisch. Es handelt sich um Fehler im Verständnis des Problems und in der Auswahl eines entsprechenden Lösungsalgorithmus'. Man kann dazu auch sagen, daß vielfach das falsche Problem gelöst wurde oder daß der gewählte Algorithmus für das gegebene Problem nicht adäquat oder nicht ausreichend war. Die Untergruppierungen sind charakteristisch für die Problemstellungen in einem Betriebssystem. Bei einem anderen Projekt, z. B. bei einem Kompiler dürften hier andere Gruppierungen auftreten.

b) Die Fehler in Gruppe B sind verfahrensspezifisch. Hier liegen die Fehler in der mehr oder weniger vollständigen und richtigen Implementierung eines gegebenen Algorithmus',in der Übertragung eines Algorithmus' in eine Programmiersprache und dergleichen. Bei verschiedenartigen Problemstellungen dürften dieselben Fehlerarten auftreten, sofern dieselben Verfahren und Hilfsmittel für die Programmierung benutzt werden. Werden andere Verfahren benutzt, z. B. höhere Sprachen, vorgefertigte Hilfsroutinen, so dürften hier andere Gruppierungen auftreten.

c) Gruppe C schließlich sind keine Programmierfehler im engeren Sinne. Es sind schon Fehler im Code, die nicht drin bleiben können. Sie können aber, nachdem sie erkannt worden sind, zumindest teilweise auch von Leuten behoben werden, die keine Programmierer sind oder keine Detailkenntnisse bezüglich dieses Projekts haben.

Was sagen uns diese Zahlen? Zunächst zu der globalen Aufteilung. Fast die Hälfte (46%) aller Fehler liegen im Bereich des Problemverständnisses, der Problemkommunikation, dem Wissen um Lösungsmöglichkeiten und Lösungsverfahren. Die andere Hälfte (38%) sind Dinge, wo wir mit anderen Verfahren, andere d. h. bessere Ergebnisse erwarten können.

Diese Aufteilung in je zwei fast gleiche Hälften scheint sich auch bei anderen gleichgerichteten Untersuchungen zu bestätigen. Diese Tatsache ist alarmierend oder ermutigend, je nachdem wie hoch die Erwartungen waren bezüglich der hundertprozentigen Automation der Software-Erstellung. Genauer gesagt, nur die Hälfte der Fehler sind mit programmtechnischen Mitteln (bessere Programmiersprachen, umfassendere Testhilfen) in den Griff zu bekommen. Der anderen Hälfte muß mit besseren Methoden der Problemdefinition (Spezifikationssprachen), dem besseren Verständnis der grundlegenden Systemkonzepte (Schulung, Ausbildung) und dem Verfügbarmachen einschlägiger Algorithmen zu Leibe gerückt werden.

Nun zu den einzelnen Gruppen. Was sich in den Zahlen der Gruppe A ausdrückt, kann man wie folgt umreißen: Die Aufgabenstellungen in einem Betriebssystem sind außerordentlich diffus. Die Abhängigkeit von der Maschinenarchitektur und -konfiguration ist groß. Die funktionalen Anforderungen an das System sind nicht sehr präzise formulierbar. Es werden häufig Dinge geändert, wenn man einmal gesehen hat, was ihr Effekt auf das System ist. Das dynamische Verhalten

des Systems ist das Hauptproblem. Die Parallelität von Abläufen und Ereignissen stellt bekanntlich große Anforderungen an die Vorstellungskraft. Auch sind hier die Hilfsmittel schlecht.

Bei der Gruppe B fällt auf, daß hier typische Probleme der Assembler-Programmierung eine starke Rolle spielen. Andere Gruppierungen, wie z. B. das Problem der Initialisierung, sind derart bekannte und universelle Phänomene, so daß nicht ihr Erscheinen auf dieser Liste, sondern lediglich die prozentuale Häufigkeit von Interesse sein dürfte. Insgesamt liefern die für diese Gruppe gewählten Formulierungen etwas den Eindruck, als ob triviale Verwechslungen und Flüchtigkeitsfehler eine sehr große Rolle spielen. Bei den Untergruppen B2 und B4 dürfte dieser Eindruck zu Recht bestehen. Bei den Untergruppen B1 und B3 steckt sicherlich sehr oft ein viel komplexerer und tiefer liegender Fehler hinter der etwas trivial klingenden Klassifizierung.

Die Gruppe C schließlich illustriert, daß es keinen Weg daran vorbei gibt, auch die technisch relativ unattraktiven Aufgaben im Zusammenhang mit der Erstellung großer Systeme mit pedantischer Sorgfalt zu erledigen.

5. URSACHE UND VERHÜTUNG VON FEHLERN

Die Frage nach der Ursache eines Programmfehlers, nach dem Warum, ist mehrschichtig. Da man ja Programmieren als eine menschliche Aktivität ansehen muß, kann man eigentlich nicht umhin, einen sehr weiten Bogen zu spannen. Bei einer einigermaßen umfassenden Betrachtungsweise kann man folgende Fehlerursachen unterscheiden:

a) technologische, (Problemdarstellbarkeit, Lösungsmöglichkeiten, verfügbare Verfahren und Hilfsmittel),

b) organisatorische, (Arbeitsteilung, vorhandene Information, Kommunikation, eingesetzte Mittel),

c) historische, (Vorgeschichte des Projekts, des Programms; Situationen und äußere Einflüsse),

d) gruppendynamische, (Kooperationswillen, Rollenverteilung innerhalb der Projektgruppe),

e) individuelle, (Erfahrung, Veranlagung und Verfassung der einzelnen Programmierer),

f) sonstige und

g) nicht erklärbare.

Es kann nicht geleugnet werden, daß die Ursachen menschlicher Fehlleistungen - und darum handelt es sich ja auch bei Programmfehlern - zu einem wesentlichen Teil im Psychologischen (also im unteren Teil der oben angegebenen Skala) liegen. So hat z. B. G. Weinberg [5] in seinem bekannten Buch viele Argumente für eine derartige Betrachtungsweise geliefert.

Ich möchte im folgenden eine stark eingeschränkte Betrachtungsweise wählen. Ich möchte als Fehlerursache das verstanden wissen, was hätte anders sein müssen, damit der Fehler nicht eingetreten wäre. Fehlerursache und Fehlerverhütung sind dann ein und dasselbe, lediglich mit verschiedenen Vorzeichen. Wenn ich außerdem für die weiteren Ausführungen mich beschränke auf die beiden oberen Gruppen, nämlich die technologischen und organisatorischen Ursachen, so ist

a) Fehlerursache: Die Diskrepanz zwischen Schwierigkeit der Aufgabe und Angemessenheit der aufgewandten Mittel und

b) Fehlerverhütung: Alle Maßnahmen, die geeignet sind, diese Diskrepanz zu verringern.

Daß selbst bei dieser sehr eingeschränkten Betrachtungsweise noch viele Freiheitsgrade und Unsicherheiten vorhanden sind, liegt auf der Hand. Dazu kommt, daß in einer bestimmten Situation die beschriebene Diskrepanz auf zwei Arten reduziert werden kann. Man kann entweder die aufgewandten Mittel vermehren, um eine gegebene Aufgabe zu lösen, oder aber die Aufgabe so modifizieren, daß sie den vorhandenen Mitteln besser angepaßt ist. Aus beiden Betrachtungsrichtungen resultieren Forderungen, die geeignet sind, das Auftreten von Fehlern zu vermeiden.

Auf der Basis dieser Überlegungen können wir uns noch einmal die Fehlerarten (Tabellen 6-1 bis 6-10) ansehen und versuchen, jeder dieser Gruppen die technischen und organisatorischen Gründe gegenüberstellen, die zu diesem Fehler geführt haben können. Mit den Tabellen 7-1 bis 7-7 ist dies für die sieben wichtigsten Gruppen der in unserem Fall aufgetretenen Fehlerarten gemacht.

Wie sich aus dem vorher Gesagten ergibt, deuten die so spezifizierten "Fehlerfaktoren" gleichzeitig an, was getan werden könnte, um den betreffenden Fehler zu verhüten. Als Beispiel: Wenn man akzeptiert, daß die Ursache eines Fehlers in der Gerätebehandlung (Gruppe A1) in der mangelnden Klarheit der Hardware-Dokumentation lag, so kann man diesen Fehlertyp verhüten, in dem man die Klarheit der Hardware-Dokumente verbessert.

Was für diese Betrachtungsweise spricht - um das deutlich zu betonen - ist ihr ausgesprochen konstruktiver Charakter. Alles was sich aus diesen Überlegungen ergibt, sind Ansatzpunkte für Veränderungen und Verbesserungen. Es sind nicht immer die vollständigen und wissenschaftlich absoluten Erkenntnisse, die wir erhalten; es sind jedoch die Dinge, die mit technischen und organisatorischen Mitteln beeinflußt werden können. Ähnlich motivierte Vorschläge zur Verhütung von Programmfehlern findet man u. a. bei Kosy [6], Elspas [7] und Lowry [8].

Die in den Tabellen 7-1 bis 7-6 angedeuteten Fehlerfaktoren geben lediglich die Art der technischen oder organisatorischen Maßnahmen an, die diese betreffende Fehlerart beeinflussen. Die jeweiligen Listen erheben keinen Anspruch auf Vollständigkeit.

Was die Darstellungsweise jedenfalls verdeutlicht, ist die Tatsache, daß offensichtlich für jede Fehlerart andere Ursachen, und daher andere Verhütungsmaßnahmen in Frage kommen. Oder anders ausgedrückt, es gibt kein einzelnes Allheilmittel. Berücksichtigt man ferner noch die hier außer acht gelassenen Schichten bezüglich möglicher Fehlerursachen, so ergibt sich ein sehr ernüchterndes Gesamtbild: Genau so mannigfaltig wie die Fehlerarten und so vielschichtig wie die Fehlerursachen, müssen auch die Maßnahmen sein, die man ergreifen muß, um Fehler zu verhüten. Jeder Katalog derartiger Maßnahmen, den aufzustellen wir in der Lage wären, bleibt seiner Natur nach bruchstückhaft. Was allerdings diese Art der Untersuchung dazu liefern kann, ist eine Konkretisierung und eine Gewichtung.

6. DAS AUFDECKEN VON FEHLERN

Wenn man zu dem Schluß kommt, daß die Möglichkeiten der Fehlerverhütung nur Stückwerk sein können, drängt sich die Frage auf, ob man aus den bisherigen Überlegungen etwas ableiten kann bezüglich der Aufdeckungsmöglichkeit von Fehlern. In der Tat erscheint es sinnvoll, der in Abschnitt 4.2 gewonnenen Klassifizierung in Fehlerarten, die Verfahren entgegenzustellen, die heute angewandt werden, um Fehler in Programmen aufzudecken. Entsprechend der Aufteilung der Fehlerarten in Gruppe A und B (Gruppe C wollen wir hier übergehen) kommen unterschiedliche Verfahren in Frage; zumindest liegt der Schwerpunkt der Verfahren anders.

Für die Gruppe A erscheinen folgende, in der Praxis angewandte Verfahren als die Wichtigsten:

a) Sorgfältiges Prüfen der funktionalen (externen) und der logischen (internen) Spezifikationen durch unbeteiligte Dritte. Es sind dies in der Regel andere als die am Projekt direkt beteiligten Mitarbeiter. Im Falle eines Betriebssystems könnten dies sein: Hardware-Entwickler, Produktplaner, Vertriebsspezialisten, Anwendungsprogrammierer u. a.

b) Ergänzende oder überlappende Beschreibung des Systems durch formale Methoden (z. B. VDL, Petri-Netze, Markov-Modelle).

c) Benutzung von analytischen oder Simulationsmodellen, (die normalerweise für Leistungsanalysen erstellt werden), um das funktionale Verhalten des Systems zu veranschaulichen bzw. zu verstehen.

d) Inspektion des geschriebenen Programmtextes durch Dritte. Hierfür kommen andere erfahrene Programmierer des Projektes in Frage, die zwar die Problemstellung, aber nicht den gewählten Lösungsweg kennen.

e) Durchführung von Testläufen durch Mitarbeiter des Projektes oder durch unabhängige Dritte.

Für die Gruppe B sieht der typische Katalog der Maßnahmen etwas anders aus. Hier ist der Schwerpunkt deutlich auf die Verfahren der Programmverifikation im engeren Sinne zu legen. Es gehören dazu:

a) Programminspektion durch Dritte (siehe d) oben),

b) Beweisen von Programmen nach der Methode von Floyd oder Hoare.

c) Handsimulation von Testläufen, d. h. Durchrechnen von Beispielen mit Bleistift und Papier.

d) Durchführen von Testläufen durch den Ersteller des Programms.

e) Durchführung von Testläufen durch unabhängige Dritte.

Es wäre sicherlich vermessen, exakte Angaben über die relative Wirksamkeit dieser Verfahren und Maßnahmen aus den vorliegenden Daten ableiten zu wollen. Basierend auf Erfahrungen und subjektivem Urteilsvermögen kann man bestenfalls eine relative Gewichtung der einzelnen Verfahren vornehmen. Wenn die dadurch gewonnene Aussage auch einen leicht spekulativen Charakter hat, so ist sie doch geeignet, die damit verbundene Problematik zu beleuchten. Mit anderen Worten, es wird nicht erwartet, daß man den jeweiligen Zahlenwerten eine große Genauigkeit beimißt, stattdessen hofft man, daß sie einen gewissen Anhalt dafür geben, wie man überhaupt die Wirksamkeit der erwähnten Maßnahmen messen und ausdrücken kann.

Die Darstellung, die für Bild 8-1 und Bild 8-2 gewählt wurde, zeigt für jedes Verfahren in Prozent die geschätzte Wahrscheinlichkeit dafür, wie sehr dieses Verfahren geeignet erscheint, die betreffende Fehlerart aufzudecken. Die Aussagen sind nicht absolut, sondern nur relativ, d. h. es wird nichts darüber gesagt, ob der Fehler überhaupt gefunden werden kann; sondern nur, wenn er gefunden wird, welches Verfahren dies vermutlich bewirken kann. Die Zeilensumme der Wahrscheinlichkeit je Fehlerart ist also 100%. Die Frage nach der absoluten Erfolgswahrscheinlichkeit wäre natürlich von noch größerem Interesse. Da der Grad der Spekulation in diesem Falle noch um einiges größer wäre, wollen wir uns davon enthalten.

7. SCHLUSSBEMERKUNGEN

Die vorangegangenen Ausführungen haben - so hoffe ich - gezeigt, daß die Analyse von Programmfehlern ein zwar schwieriger, aber ein recht notwendiger und nützlicher Prozeß ist. Wir können auf

diese Art zu Aussagen kommen, die uns helfen können, Gegenmaßnahmen zu finden gegen gewisse wiederkehrende Fehlerarten. Die vorliegende Untersuchung befaßte sich mit einer ganz speziellen Gruppe von Systemprogrammen. Es gibt daher noch eine Vielzahl von Fragen, zu denen auf Grund des vorliegenden Materials keine Antworten gegeben werden konnten. Zu diesen Fragen gehören:

a) Was ist der Effekt der Verwendung höherer Programmiersprachen in der Systemprogrammierung? Welche Fehlerarten treten seltener auf?

b) Welche generellen Unterschiede gibt es zwischen Kontrollprogrammen und Kompilern?

Einige der Schlußfolgerungen, die auf Grund der vorliegenden Daten gezogen wurden, sind sicherlich alles andere als zwingend. Vielleicht regt gerade das den einen oder anderen Kollegen an, in der angedeuteten Richtung nachzudenken und meine Vorschläge zu ergänzen, zu rechtfertigen oder zu widerlegen.

Literaturverzeichnis

[1] Moulton, P. G. and Muller, M. E.: DITRAN - A compiler emphasizing diagnostics, CACM 10, 1 (Jan. 1967)

[2] Rubey, R et al: Comparative evaluation of PL/I, ESD-TR-68-150 (Apr. 1968)

[3] Boies, S. J. and Gould J. D: A behavioral analysis of programming - on the frequency of syntactical errors. IBM Research, Yorktown Heights, N.Y., RC-3907 (June 1972)

[4] Henderson, P. and Snowdon, R: An experiment in structured programming, BIT 12 (1972), pp 38 - 53

[5] Weinberg, G: The psychology of computer programming, Van Nostrand Reinhold, New York, 1971

[6] Kosy, D.W: Approaches to improved program validation through programming language design, in W. G. Hetzel (ed): Program Test Methods Prentice Hall, Englewood Cliffs, N. J., 1973

[7] Elspas, B.et al: Software reliability, IEEE Computer 4,1 (January/February 1971)

[8] Lowry, E: Proposed language extensions to aid coding and analysis of large programs, IBM Systems Development Division, Poughkeepsie, N.Y., TR 00.1934 (1969)

Instr./Modul	nur alter Code	alter + neuer Code	nur neuer Code	insge-samt	%
0- 99	84	27	52	163	
100- 199	5	63	38	106	
200- 299	5	43	30	78	81,5%
300- 399	4	27	19	50	
400- 499	-	23	6	29	
500- 749	1	22	8	31	
750- 999	1	12	3	16	14,0%
1000-1249	-	13	1	14	
1250-1499	-	5	6	11	
1500-1999	-	6	5	11	
2000-2499	-	6	-	6	3,5%
2500-2999	-	2	-	2	
3000-4200	-	4	1	5	1,0%
Σ Moduln	100	253	169	522	100,0%

Tabelle 2-1: Verteilung der Modulgrößen und Modularten

geänderte Instruktionen / Modul-größe	<100	200	300	400	<500	<750	1000	1250	1500	2000	2500	<3000	>3000	Σ
1 - 9	19	11	9	5	3	10	4	4	2	-	-	-	-	67
10 - 99	8	49	26	12	9	4	3	3	-	1	-	-	-	115
100 - 999	-	3	8	10	11	8	5	6	2	5	5	2	1	66
>1000	-	-	-	-	-	-	-	-	1	-	1	-	3	5
	27	63	43	27	23	22	12	13	5	6	6	2	4	253

Tabelle 2-2: Verteilung der Moduln nach Änderungsumfang

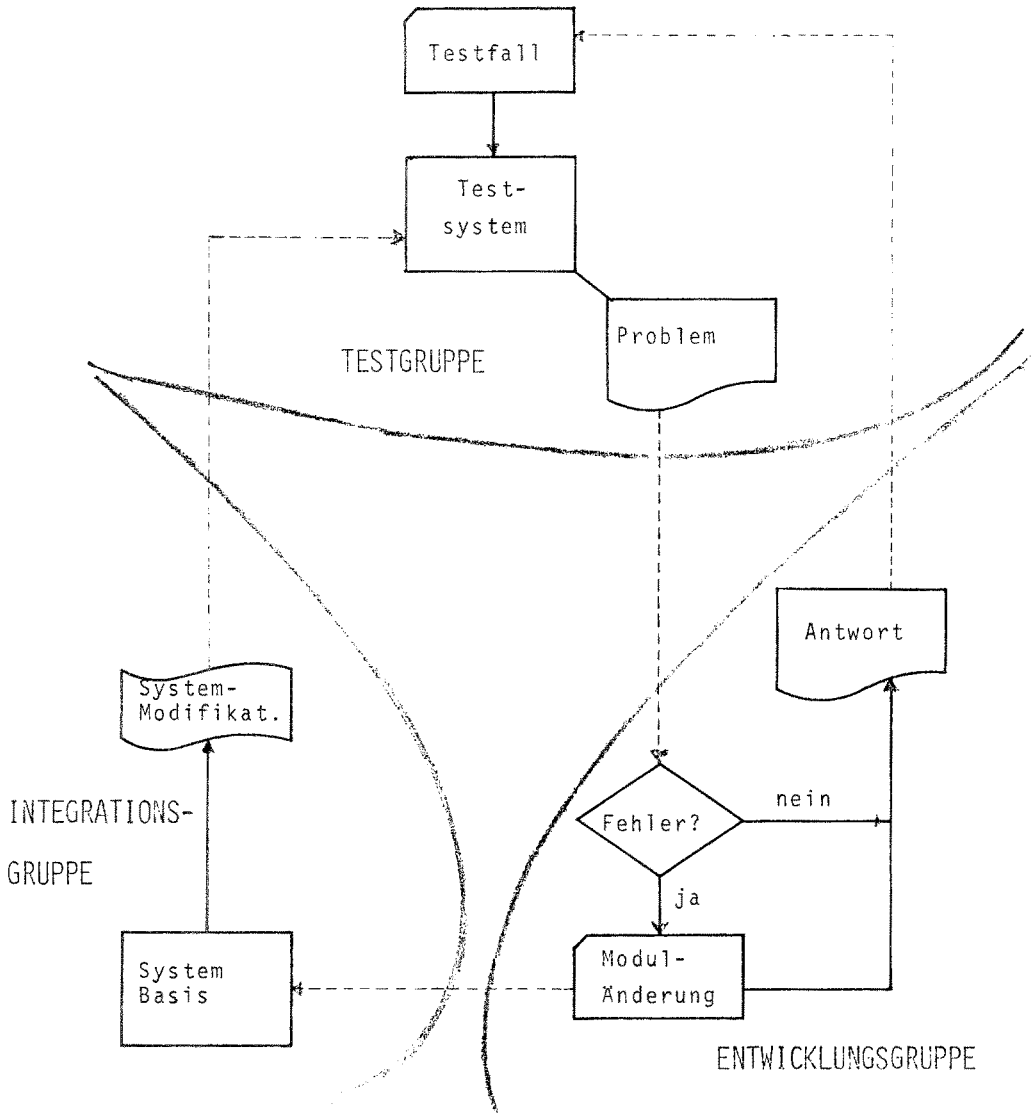

Bild 2-3: Informationsfluß während eines Programmtests

Anzahl der Fehler	Anzahl der betroffenen Moduln
371	1
50	2
6	3
3	4
1	5
1	8
432	

Tabelle 4-1: Anzahl der von einem Fehler betroffenen Modu

Anzahl der Moduln	Anzahl der Fehler je Module
112	1
36	2
15	3
11	4
8	5
2	6
4	7
5	8
3	9
2	10
1	14
1	15
1	19
1	28
202	512

Tabelle 4-2: Anzahl der Fehler je Modul

Modulart ↓	Anzahl Moduln insgesamt	Anzahl Moduln mit Fehler	Prozent der Moduln mit Fehler
nur neuer Code	169	81	48
alter und neuer Code	253	121	48
insgesamt	422	202	48

	Anzahl Moduln	Anzahl Fehler	Fehler pro Modul
nur neuer Code	169	254	1,5
alter und neuer Code	253	258	1,0
insgesamt	422	512	1,2

	Umfang des neuen Codes	Anzahl Fehler	Fehler pro 1K Code
nur neuer Code	53K	254	4,8
alter und neuer Code	33K	258	7,8
insgesamt	86K	512	6,0

Tabelle 4-3: Fehlerhäufigkeit nach Modulart

Gruppe A

1.	Maschinenkonfiguration und Architektur	10
2.	Dynamisches Verhalten und Kommunikation zwischen Prozessen	17
3.	Angebotene Funktionen	12
4.	Ausgabelisten und -formate	3
5.	Diagnostik	3
6.	Leistungsverhalten	1
		46%

Tabelle 6-1: Fehlerarten Gruppe A

A1	Maschinenkonfiguration und Architektur	
(a)	Gerätetyp oder -zusatz nicht berücksichtigt; gültiger E/A Befehl als ungültig angesehen	1,5
(b)	E/A Fehlersituation oder Gerätestatus falsch oder garnicht behandelt (vor allem bei dezentralen E/A Routinen)	3
(c)	E/A Befehl falsch benutzt, unvollständig oder falsch simuliert (bei Abbildung eines Gerätes auf einem anderen)	4
(d)	Fehlerstatistik für Gerät nicht oder unnötigerweise generiert	1
(e)	Externer Bedienungsmodus eines Gerätes falsch behandelt	0.5
		10

Tabelle 6-2: Fehlerarten Gruppe A1

A2 Dynamisches Verhalten und Kommunikation zwischen Prozessen

(a)	Dynamisch eingetretener Systemzustand nicht genau genug abgetestet	2
(b)	Bei sequentiellem Übergang auf anderen Prozeß (vor allem bei erzwungenem Abbruch) Zustand nicht aufgearbeitet. Nachfolgeprozeß findet nicht erwartete Parameter (z. B. Registerinhalt) vor	3
(c)	Register und Kontrollblöcke, die mehrfach benutzt werden, wurden nicht gerettet. Unterbrechung zerstört Information, die noch gebraucht wird	4
(d)	Unterbrechungen waren zugelassen, die ausgeschlossen werden können. Andere Unterbrechungen waren ausgeschlossen, wurden ignoriert, obwohl für Systemablauf wichtig	3
(e)	Logisch erforderliche Schritte (z. B. Öffnen einer Datei) fehlten. Falsche Sequentialisierung, falscher Rücksprung	3
(f)	Betriebsmittelvergabe falsch; Verklemmung, Belegung nicht vorhandener, bereits belegter Resourcen. Falsche Eigenschaft angenommen	1,5
(g)	Bei nicht generierter oder nicht durchgeführter Funktion damit zusammenhängende Folgefunktionen nicht weggeneriert oder abgeschaltet	0,5
		17

Tabelle 6-3: Fehlerarten Gruppe A2

A3 Angebotene Funktionen

(a)	Funktionen werden vervollständigt, da für beabsichtigte Arbeitsweise des Systems wesentlich	2
(b)	Funktionen werden hinzugefügt, verallgemeinert, obwohl nicht ursprünglich beabsichtigt	1.5
(c)	Funktionen werden vervollständigt bezüglich Extremfällen, Ausnahmesituationen	1.5
(d)	Funktionen werden geändert, um Benutzerfreundlichkeit, Sicherheit, Kompatibilität etc. zu verbessern	2
(e)	Extern veranlaßte Änderungen (z.B. Produktstrategie)	2
(f)	Grundannahme für weggelassene Angaben (defaults) geändert	1
(g)	Zusätzliche Nachricht für Operateur/Benutzer eingefügt	1.5
(l)	Funktion wird eliminiert, da nicht mehr benötigt	0.5
		12

Tabelle 6-4: Fehlerarten Gruppe A3

Gruppe B

1.	Initialisierung (von Feldern und Bereichen)	8
2.	Adressierbarkeit (i. S. des Assemblers)	7
3.	Bezugnahme auf Namen	7
4.	Abzählen und Berechnen	8
5.	Masken und Vergleiche	2
6.	Abschätzung von Bereichsgrenzen (für Adressen und Parameter)	1
7.	Plazierung von Instruktionen innerhalb eines Moduls, schlechte Korrekturen	5
		38

Tabelle 6-5: Fehlerarten Gruppe B

B1	Initialisierung	
(a)	Kontrollblock, Register, Schalter nicht gelöscht oder zurückgesetzt bei Übergang von Routine, Prozeß, Job etc. auf anderen	5
(b)	E/A Bereich, Puffer und dergleichen vor Benutzung nicht gelöscht	1
(c)	Felder als "Define Storage" (o. Initialisierung) statt als "Define Constant" (m. Initialisierung bei Laden des Programms) erklärt	1
(d)	Statt ganzen Feldes, ganzer Tabelle nur Teil davon gelöscht	0,5
(e)	Initialisierung zum falschen Zeitpunkt oder mit falschem Wert	0,5
		8

Tabelle 6-6: Fehlerarten Gruppe B1

B2 Adressierbarkeit

(a)	Zuordnen, Laden, Retten von Adreßregistern vergessen (vor allem bei Anwachsen des Kodes)	2
(b)	Auswirkung von Längenänderungen bei Konstanten, Nachrichten auf Folgebereiche übersehen	1
(c)	Kode, Bereiche, Nachrichten überlagert, da Speicherplatz mehrfach benutzt	1
(d)	Verwechslung absolute + verschiebliche, reelle und virtuelle Adressen (vor allem bei Zugriff zu unteren Kernspeicherbereichen)	1
(e)	Anpassung an Wortgrenzen (Alignment) fehlerhaft	1
(f)	ORG, LTORG eingefügt, geändert	0,5
(g)	Aufteilung einer Phase wegen Überschreitens des vorgegebenen Speicherplatzes	0,5
		7

Tabelle 6-7: Fehlerarten Gruppe B2

B3 Bezugnahme auf Namen

(a)	Feld hat andere als angenommene Bedeutung (z.B. Pointer enthält nicht Adresse, sondern Adresse von Adresse)	2
(b)	Bezugnahme auf falsches Register oder falschen Feldnamen (evtl. wegen Ähnlichkeit der Abkürzung)	2
(c)	Bezugnahme auf falschen relativen Eintrag in richtig gefundenem Kontrollblock. Tabelle mit falschem Suchargument adressiert	1,5
(d)	Verwechslung diverser Konstanten (Partition Nr., SVC Nr.)	1,5
		7

Tabelle 6-8: Fehlerarten Gruppe B3

B4 Abzählen und Berechnen

(a)	Falsche Berechnung oder Abzählung von Feld- oder Satzlängen, Bereichsgrößen (nicht genannter Betrag der Abweichung)	2
(b)	wie (a) (mit Abweichung um 1 Byte)	1,5
(c)	Falsche relative Adresse (Displacement)	1
(d)	Falsches Abprüfen einer Schleifenbedingung (frühzeitiger Abbruch, endlose Schleife)	1
(e)	Programmierter Zähler (von Sätzen, Zeilen etc.) liefert falsche Werte	1
(f)	Dezimal nach Hex. Umwandlung fehlt Binär nach Dezimal Umwandlung falsch	0,5
(g)	Berechnung von Plattenadressen, Ermittlung von Spurenzahl oder Spurenkapazität falsch	1
		8

Tabelle 6-9: Fehlerarten Gruppe B4

Gruppe C

1.	Schreibfehler in Nachrichten und Kommentaren	4
2.	Fehlende Kommentare oder Flowcharts (Standards)	5
3.	Unverträglicher Stand von Macros oder Moduln (Integrationsfehler)	5
4.	Nicht klassifizierbar	2
		16%

Tabelle 6-10: Fehlerarten Gruppe C

A1 Maschinenkonfiguration und Architektur

1. Anzahl verschiedener Gerätetypen und Zusatzeinrichtungen

2. Gerätespezifische Eigenarten und Variationen der Fehlerbehandlung

3. Zugänglichkeit, Klarheit der h/w Dokumentation

4. Kontakt zu / Kommunikation mit h/w Entwicklern

5. Zentrale / dezentrale Behandlung der E/A Geräte im System

6. Bedienungserfahrung mit Gerät

Tabelle 7-1: Fehlerfaktoren Gruppe A1

A2 Dynamisches Verhalten und Kommunikation zwischen Prozessen

(1) Darstellung der Prozeßsteuerinformation im System (Übersichtlichkeit, Sicherung)

(2) Strukturierung der Prozeßhierarchie

(3) Beschreibung der Schnittstellen und Kommunikationsbedürfnisse aller Prozesse
a) explizite Parameter (Reihenfolge, Bedeutung, Format)
b) gemeinsame Datenbereiche (implizite Parameter)

(4) Standardisierte Routinen (Macros), die Prozeßsteuerung auf höherem Niveau unterstützen, gewisses Aufbereiten des Systemstatus' erzwingen etc.

(5) Darstellbarkeit der dynamischen Abläufe, Interaktion von Prozessen etc.

(6) Beschreibung der Betriebsmittel, ihrer Eigenschaften, ihres Zustands

(7) Zentrale / dezentrale Behandlung der Prozeßsteuerung, Betriebsmittelvergabe etc.

Tabelle 7-2: Fehlerfaktoren Gruppe A2

A3 Angebotene Funktionen

(1) Qualität der Spezifikationen

(2) Erfahrung mit ähnlichen Systemen

(3) Statistische Information über Benutzerprofil, Datenmengen, Bedienungsmodus

(4) Verdeutlichung / Konzentration auf Extremfälle, Ausnahmesituationen

(5) Selbstbeschränkung gegenüber eigenen "netten Ideen". Disziplin gegenüber externen Wünschen.

Tabelle 7-3: Fehlerfaktoren Gruppe A3

B1 Initialisierung

(1) Erzwungenes Initialisieren oder Warnnachricht durch Übersetzer bei fehlender Initialisierung

(2) Automatisches Anpassen der Operationen an Feldlänge (z. B. lösche ganzes Feld)

(3) Analyse von Routinen bezüglich ihres Effekts auf diverse Kontrollblöcke, Register und Datenfelder

(4) Darstellung der expliziten und impliziten Parameter; der erlaubten und erwarteten Wertebereiche.

Tabelle 7-4: Fehlerfaktoren Gruppe B1

B2 Adressierbarkeit

(1) Ausdehnung der symbolischen Adressierung

(2) Erweiterbarkeit des Adreßraums

(3) Abgrenzung der Adreßräume je Routine
("Need to know")

Tabelle 7-5: Fehlerfaktoren Gruppe B2

B3 Bezugnahme auf Namen

(1) Syntax von Namen

(2) Qualifikationsmöglichkeit von Namen

(3) Darstellung der Rolle eines Feldes und
Angabe der Routinen mit Zugriffsrechten

(4) Assoziative Adressierung von Tabellen

Tabelle 7-6: Fehlerfaktoren Gruppe B3

B4 Abzählen und Berechnen

(1) Selbstbeschreibende Daten und Bereiche

(2) Mächtige, an Datenbeschreibung gekoppelte Schleifenbefehle (z. B. Iteriere über alle Einträge einer Tabelle)

(3) Hilfstabellen oder leicht verfügbare Umrechnungsroutinen für
a) Plattenadressen
b) Spurkapazitäten

(4) Regelmäßigere Adreßstruktur für alle Geräte; symbolische Adressierung

Tabelle 7-7: Fehlerfaktoren Gruppe B4

Maßnahme / Fehlerart	Prüfen der Specs durch Dritte	Formale Beschreibungsmethoden	Simulation, Modell-Bildung	Programm-Inspektion durch Dritte	Durchführen von Testläufen
A1 Maschinenkonf. + Architektur	30	10	10	20	30
A2 Dyn. Verhalten + Kommunikation	10	20	20	20	30
A3 Angebotene Funktionen	40	-	-	20	40
A4 Ausgabelisten	30	10	-	10	50
A5 Diagnostik	20	20	-	20	40
A6 Leistungsverhalten	10	10	30	10	40

Tabelle 8-1: Fehleraufdeckung Gruppe A

Maßnahme / Fehlerart	Programm-Inspektionen durch Dritte	Floyd/Hoare Beweis-methode	Simulation von Testläufen	Durchführung von Test-läufen (selbst)	Durchführung von Testläufen (durch Dritte)
B1 Initialisierung	30	10	10	20	30
B2 Adressierbarkeit	20	-	10	30	40
B3 Bezug auf Namen	20	20	10	20	30
B4 Abzählen und Berechnen	20	20	20	20	20
B5 Masken und Vergl.	20	10	20	20	30
B6 Bereichsgrenzen	30	10	10	20	30
B7 Plazierung v. Code	30	-	-	30	40

Tabelle 8-2: Fehleraufdeckung Gruppe B