PRUEFUNG SICHERHEITSKRITISCHER SOFTWARE

Agnes Kroell
Dornier-System GmbH
7990 Friedrichshafen

ZUSAMMENFASSUNG :

Das Bewusstsein, dass Software nie frei von Fehlern ist, bedeutet fuer den Hersteller, Entwickler und Betreiber von Software-Systemen, die in sicherheitskritischen Bereichen eingesetzt werden, die Verpflichtung, nach bestem Wissen und Gewissen dafuer zu sorgen, dass Auswirkungen moeglicher Fehler nicht zu einer Gefahr werden fuer Menschen und - in abgeschwaechter Form - fuer die Gesamtfunktion des Systems. Ausser konstruktiven Massnahmen wie z.B. die fehlertolerante, redundante Auslegung dieser Systeme, sind hierzu vor allem analytische Massnahmen der Software-Qualitaetssicherung einzuplanen und durchzufuehren. Diese analytischen Massnahmen sind als integrale Bestandteile des Software-Life-Cycle-Modells, d.h. der Software-Entwicklungs- und Betriebsphasen zu sehen. Ihre Anwendung bzw. ihr Einsatz verfolgt drei Ziele: Durch Validation wird die Anwendbarkeit des Systems in seiner endgueltigen Systemumgebung/Betriebsumgebung geprueft; der Entwicklungsgang wird verifiziert hinsichtlich Problem-Angemessenheit, Korrektheit, Vollstaendigkeit und Konsistenz; die Teil- und Endprodukte des SW-Entwicklungs-Prozesses, Dokumente und Programme, werden auf Fehler ueberprueft. Damit Software-Fehler moeglichst fruehzeitig erkannt werden koennen, bzw., damit man sie vermeiden oder umgehen kann, ist es notwendig, die verschiedenen Fehlerarten, ihre Ursachen, Dauer und ihre Auswirkungen auf die Umgebung zu untersuchen. Nur dadurch koennen konkrete Zuverlaessigkeitsaussagen gemacht werden und die analytischen Massnahmen effizient geplant und eingesetzt werden.

KEYWORDS :

Software-Entwicklung, Software-Qualitaetssicherung, Validation, Verifikation, Test, Fehler-Klassifizierung, Software-Zuverlaessigkeit

Auswirkungen der Defekte	Anzahl Ereignisse/Betriebsstunde
Misstaende	10^{-1} bis 10^{-2}
Betriebseinschraenkungen	10^{-3} bis 10^{-4}
Notfallbehandlung	10^{-4} bis 10^{-5}
Sofortlandung noetig (Personenschaden oder schwerer Schaden)	10^{-5} bis 10^{-8}
Notlandung (mit Toten bzw., schwerste Schaeden)	10^{-8} bis 10^{-9}
Absturz	kleiner 10^{-9}

Bild 1: von der FAA geforderte Systemzuverlaessigkeit

1 ZUVERLAESSIGKEIT

Ausgangspunkt der Betrachtungen ist der Begriff Zuverlaessigkeit. Die Zuverlaessigkeit eines digitalen Systems wird definiert als:

> Die Wahrscheinlichkeit, dass ein System die von ihm geforderten Funktionen unter festgelegten Bedingungen und fuer eine festgelegte Zeitperiode ausfuehrt.

Die Zuverlaessigkeit gilt als eine messbare Groesse. Ihr Wert gibt das Ausmass an, mit welchem eine definierte Qualitaet der Ausfuehrung (Zufriedenheit) erreicht wird. In der Norm DIN 40 041 "Zuverlaessigkeit in der Elektrotechnik" wird eine Reihe von Masszahlen definiert, z.B. der mittlere Ausfallabstand (MTBF), die mittlere Ausfalldauer (MDT) und die Verfuegbarkeit.

Fuer HW-Komponenten wird die Masszahl fuer die HW-Zuverlaessigkeit bestimmt aus der Ausfallwahrscheinlichkeit. Aufgrund durchgefuehrter Messungen kann hier angenommen werden, dass die Ausfallwahrscheinlichkeit eines HW-Bauteils exponentialverteilt ist. Die Masszahl je HW-Bauteil wird durch Ausfall-Messungen vorgealteter Bauteile bestimmt. Durch Summation und Multiplikation der Ausfallwahrscheinlichkeiten der Teilkomponenten laesst sich die Ausfallwahrscheinlichkeit des Gesamtsystems errechnen. Die Zuverlaessigkeit wird dann bestimmt durch die Formel:

$$\text{Zuverlaessigkeit} = \frac{1}{\text{Ausfallwahrscheinlichkeit}}$$

d.h., die Zuverlaessigkeit ist umgekehrt proportional zur Ausfallwahrscheinlichkeit.

Die Bestimmung einer Zuverlaessigkeitsmasszahl fuer die Software ist jedoch nicht ohne weiteres moeglich. Software gilt als "save life", sie "altert" nicht wie HW-Bauteile; ihre Qualitaet bleibt daher immer gleich. Die Masszahl MTBF, die bei der Berechnung der Ausfallwahrscheinlichkeit von HW benutzt wird, hat hier eine andere Bedeutung. Daten, die zur Berechnung der HW-Zuverlaessigkeit dienen, koennen nicht ohne weiteres auf das Gebiet der SW uebertragen werden.

Es gibt verschiedene theoretische Verfahren die Zuverlaessigkeit von SW zu errechnen. Ihr Ansatz beruht zum einen auf dem Versuch bestimmte HW-Gesetzmaessigkeiten auch auf das Gebiet der SW zu uebertragen, zum anderen werden statistische Ausfallraten "aehnlicher" Systeme ermittelt. In Bild 1 sind z.B. die von der FAA geforderten Ausfallwahrscheinlichkeiten fuer "fliegende Systeme" aufgelistet (FAA - Federal Aviation Administration der USA). Fuer SW-Systeme haben derartige Zahlenwerte wenig Aussagekraft.

Fuer die Erfassung von SW-Zuverlaessigkeitsmassen ist eine Vielzahl von Messwerten ueber einen langen Beobachtungszeitraum erforderlich. Allgemein anerkannte (bzw. veroeffentlichte) Masszahlen fuer die SW-Zuverlaessigkeit fehlen noch. Und die bekannten Zahlen lassen sich fuer Vergleiche, selbst im gleichen Anwendungsgebiet, nur bedingt heranziehen; die diese Masszahlen beeinflussende Vielfalt der verwendeten Entwurfs- und Implementierungstechniken ist zu gross.

Gemeinsam ist allen statistischen Methoden, dass die Zuverlaessigkeit des SW-Systems erst im nachhinein, d.h., wenn das System bereits installiert und im Betrieb ist, errechnet werden kann. Sie bieten daher keine Hilfestellung bei Definition und Entwurf.

Die statistischen Untersuchungen verbessern die Zuverlaessigkeit eines Systems nicht direkt. Sie ermoeglichen es aber dem Projektmanagement durch die Kenntnis der Analysedaten (Fehlerverteilung, Fehlerursachen, usw.) fruehzeitig die Aufmerksamkeit auf bestimmte kritische Stellen zur richten. Dadurch koennen Fehler, wenn schon nicht vermieden, dann doch kurz nach ihrer Entstehung identifiziert und beseitigt werden.

Daher gilt:

Eine hohe SW-Zuverlaessigkeit kann nur erreicht werden durch intensive Anstrengungen waehrend des gesamten Entwicklungsprozesses d.h., durch ein Zusammenspiel von

- konstruktiven Methoden bei Definition und Entwurf

  und

- analytischen Massnahmen zur Ueberpruefung von Entwicklungsgang und entstehenden Produkten.

Art und Intensitaet durchzufuehrender Pruefmassnahmen zur Erhoehung der Zuverlaessigkeit werden bestimmt durch die sicherheits-kritischen Auswirkungen der SW-Komponenten. Hierzu werden die SW-Systeme in verschiedene Kritikalitaetskategorien eingeteilt. In der Luftfahrt wird unterschieden zwischen 3 Kritikalitaetsklassen (RTCA-Do 178):

1. Kategorie: Kritische Funktionen

   - bei deren Ausfall oder Fehlverhalten die Sicherheit des gesamten Systems gefaehrdet ist.

2. Kategorie: Essentielle Funktionen

   - bei deren Ausfall oder Fehlverhalten die Faehigkeit des Gesamtsystems reduziert wird, mit widrigen Betriebsbedingungen fertig zu werden.

3. Kategorie: Nonessentielle Funktionen

- bei deren Ausfall oder Fehlverhalten die Sicherheit des Gesamtsystems nicht wesentlich beruehrt wird.

Die Einteilung in diese Kritikalitaetsklassen hat Auswirkung auf Anzahl, Verschiedenartigkeit und Intensitaet von einzuplanenden und durchzufuehrenden Pruefaktivitaeten. Moegliche Folgen von Fehlern und die fuer die analytischen Massnahmen entstehenden Kosten muessen im projektspezifischen Einzelfall gegeneinander abgewogen werden.

2 FEHLERANALYSEN

Damit SW-Fehler moeglichst fruehzeitig erkannt werden koennen, bzw., damit man sie moeglichst vermeiden oder umgehen kann, ist es notwendig, die verschiedenen Fehlerarten, ihre Ursachen, ihre Dauer und ihre Auswirkungen auf die Umgebung zu untersuchen.

2.1 Zur Definition des Begriffs Fehler

Vor einer Klassifizierung und Kategorisierung der verschiedenen Fehlertypen, erscheint es sinnvoll, zuerst den Begriff "Fehler" naeher zu betrachten. Fuer den deutschen Begriff "Fehler" gibt es im englischen Sprachraum gleich 5 Begriffe, deren Bedeutung durch die NTG 3004 folgendermassen angegeben wird:

FAULT

darunter versteht man:

- jedes Merkmal, das die Zuverlaessigkeit eines Systems beeinflusst;
- eine unzulaessige Eigenschaft, die das Versagen eines Systems bewirken kann.

ERROR

wird gebraucht als:

- eine Abweichung zwischen einem berechneten Wert-/Bedingung und dem wahren, spezifizierten oder theoretisch richtigen Wert/Bedingung

  Man kann hier noch unterscheiden zwischen:

- DEFECT

  einer unzulaessigen Abweichung eines Merkmals vom erwarteten Wert

- MISTAKE

  einer menschlichen Handlung, die ein unerwuenschtes Ereignis zur Folge hat.

In diesem Zusammenhang ist dann FAILURE eine Fehlfunktion, d.h., das fehlerhafte (Fault) Verhalten eines Systems.

Entsprechend diesen Definitionen umfasst die Bedeutung der Begriffe "Failure" und "Fault" auch die Bedeutung der Begriffe "Error", "Defect" und "Mistake".

Das Risiko fuer das Auftreten eines SW-Fehlers wird in der englischsprachigen Literatur mit "Software Hazard" bezeichnet. Die Gruende fuer SW-Hazards werden in 3 Klassen eingeordnet:

1. Unerwartete Ereignisse:

   Ein unerwartetes, unbekanntes Ereignis tritt auf.

2. Out-of-Sequence Ereignis:

   Ein bekanntes und auch erwartetes Ereignis tritt zu einem Zeitpunkt auf, an dem es gerade nicht erwartet wird.

3. Ausfall eines Ereignisses:

   ein erwartetes Ereignis tritt nicht auf.

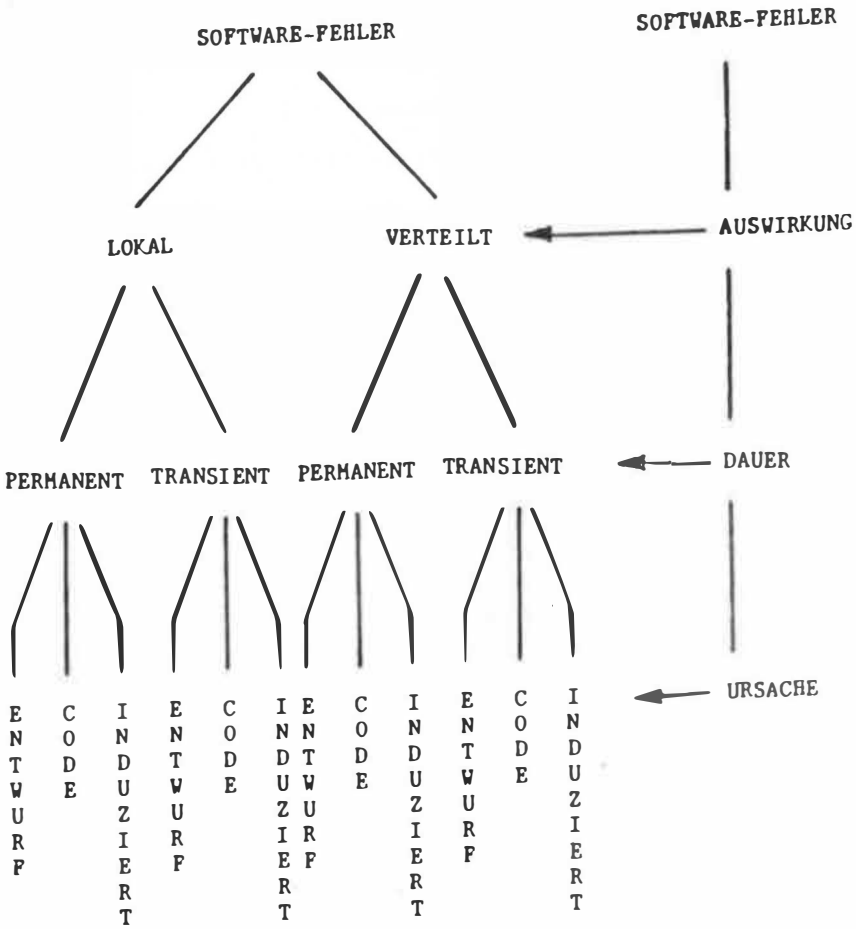

Bild 2: Klassifikation von SW-Fehlern

2.2 Fehlerklassifizierung

Eine Klassifizierung von SW-Fehlern kann jetzt erfolgen (siehe auch Bild 2) :

1. Aufgrund ihrer Auswirkungen auf:

   - vom Fehler betroffene Systembereiche,
   - den Kontroll- und Datenfluss im digitalen System selbst,
   - die Anwendungsbestimmung (Mission) des Systems,
   - die Betriebsumgebung, in der der System-Ausfall/-Fehler beobachtet wurde,

   als:

   1. lokale Fehler:

      - es ist lediglich eine einzige Komponente davon betroffen,

   2. verteilte Fehler:

      - betroffen sind mehrere Systemkomponenten, evtl., sogar das Gesamtsystem.

2. Aufgrund ihrer Dauer als:

   1. Permanente/persistente Fehler:

      Ein Fehler ist persistent, wenn die Haeufigkeit seines Auftretens einen vorgegebenen Grenzwert ueberschreitet.

      oder:

      Eine Systemkomponente ist permanent defekt, wenn sie derzeit ausgefallen ist und fuer alle folgenden Benutzungen ebenfalls ausfaellt (bis zur Fehlerbehebung / Reparatur).

   2. Temporaere/transiente Fehler:

      Ein Fehler ist transient, wenn er nicht persistent ist.

      Eine Systemkomponente ist temporaer defekt, wenn sie zwar fuer eine (einzelne) Anwendung ausgefallen ist, sich aber wieder erholt (ohne Reparatur) und fuer

folgende Anwendungen wieder voll zur Verfuegung steht.
Bsp.: durch kurzzeitigen Spannungabfall induzierter Fehler.

Im allgemeinen gilt fuer temporaere bzw. permanente Fehler:

- Die meisten, waehrend des Betriebs auftretenden Fehler sind temporaerer Natur. (Geschaetztes Verhaeltnis: 10:1)

- Permanente Fehler haben schwerwiegendere Auswirkungen auf die Sicherheit des Systems, sind jedoch leichter zu entdecken und zu korrigieren.

3. Aufgrund der Fehlerursache:

   1. Entwurfsfehler

      dazu gehoeren Fehler in der Anforderungsdefinition, in der Spezifikation, im Grob- und Feinentwurf.

      Moegliche Ursachen:

      - das Problem wurde nicht verstanden,

      - der verwendete Algorithmus ist falsch oder arbeitet ungenuegend,

      - die Problembeschreibung ist nicht eindeutig und wird verschieden / falsch interpretiert.

   2. Kodierungsfehler

      Bsp.: Syntaxfehler, Fehler aufgrund falscher Variablendefinition

   3. Induzierte Fehler

      verursacht durch externe Ereignisse, die spezifizierte Schnittstellen-Charakteristiken verletzen.

      Bsp.: Fehlfunktionen durch ausfallende HW-Komponenten

Die Einordnung eines aufgetretenen Fehlers nach den obigen Gesichtspunkten ist sehr wichtig. Sie erlaubt den Einsatz angemessener Recovery-Techniken fuer die Korrektur bzw. eine Begrenzung des hervorgerufenen Schadens. In der Praxis ist eine exakte Zuordnung zumeist jedoch ziemlich schwierig, wenn nicht gar unmoeglich.

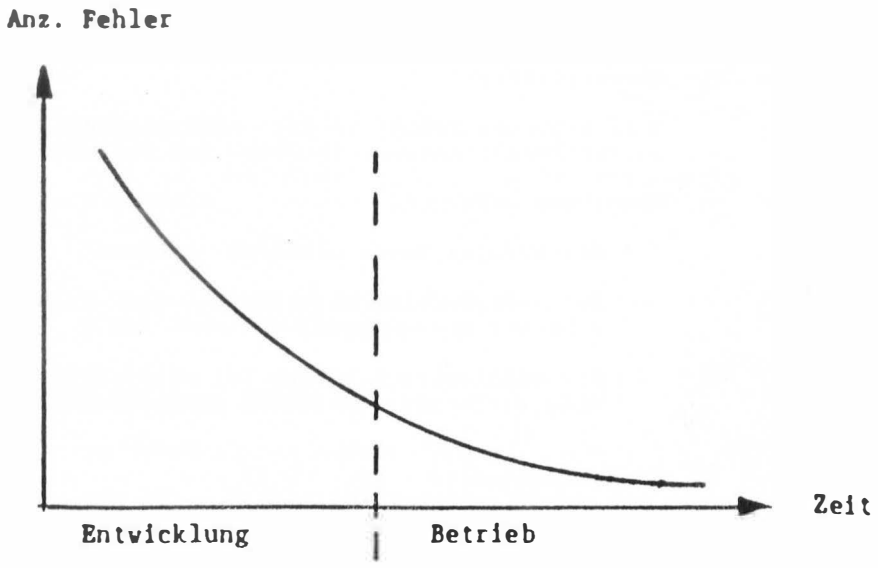

Bild 3: Restfehlerkurve

2.3 Restfehler

Die im System vorhandene Fehlerquote nimmt aufgrund von Fehlerkorrekturen waehrend der Entwicklung und in der Wartungsphase mit der Zeit ab, erreicht aber nie den Nullwert, d.h., den fehlerfreien Zustand (siehe Bild 3). Es verbleibt immer eine Restfehlermenge, die sog. latenten Fehler. Diese Fehler treten irgendwann oder auch nie in der Betriebs-, Wartungs- und Aenderungsphase auf. Die Zuverlaessigkeit eines Systems wird erhoeht durch die Minimierung dieser Restfehlerquote. Die Schwierigkeit besteht jetzt darin, gegen diese latenten Restfehler Vorkehrungen zu treffen, ohne zu wissen, wann, wo und wie sie auftreten, bzw., welcher Art sie sind.

Eine Minimierung der latenten Fehler und damit die Erhoehung der Zuverlaessigkeit des SW-Systems kann nur erreicht werden durch die sorgfaeltigste Anwendung von Entwurfs-, Test- und Verifikationsmethoden im gesamten Lebenszyklus. Dissimilare und diversitaere Entwicklung, sowie die Entwicklung von Referenzsystemen fuer Test und Verifikation tragen zusaetzlich zur Zuverlaessigkeit des Endprodukts bei.

Fuer sicherheitskritisch eingestufte SW-Komponenten muessen bereits bei der Definition und der Konstruktion des Gesamtsystems Vorkehrungen fuer die Vermeidung und Umgehung von Restfehlern getroffen werden.

Das Umgehen von Fehlern ist nur moeglich durch den Einsatz von Redundanz und der Realisierung von Fehlertoleranztechniken mit folgenden Schwerpunkten:

1. Fehler-Entdeckung
2. Schadens-Feststellung
3. Fehler-Recovery
4. Aufrechterhaltung/Weiterfuehrung des Betriebs

Voraussetzung fuer ihren Einsatz ist die Fehleranalyse des Gesamtsystems durch dafuer ausreichend qualifiziertes Personal, moeglichst frueh im Entwicklungszyklus. Fuer HW-Systeme, aber auch zunehmend fuer die SW werden hierzu in der Systemdefinitionsphase FMECA (Failure Mode, Effects and Criticality Analysis) oder Fehlerbaum-Analysen durchgefuehrt.

3 VALIDATION, VERIFIKATION UND TEST

Fehlererkennung und Beseitigung entsprechend den Ergebnissen von Fehleranalysen bedeutet die Einplanung und Durchfuehrung geeigneter Pruefmassnahmen im SW-Life-Cycle. Grundlegende Bedeutung bei diesen analytischen Massnahmen haben dabei die Begriffe Validation, Verifikation und Test.

Sie werden definiert als:

VALIDATION

Ist die Pruefung der Anwendbarkeit eines HW/SW-Systems in seiner letztendlichen Systemumgebung. Hier steht die Frage im Vordergrund: "Erstellen wir das richtige Produkt".

VERIFIKATION

Umfasst die Pruefung des Entwicklungsprozesses. Hier muss die Frage beantwortet werden: "Erstellen wir das Produkt richtig".

TEST

Bedeutet: die Suche nach Fehlern.

Wichtig fuer die Wirksamkeit aller Pruefmassnahmen ist:

- sie muessen geplant erfolgen,
- mehrere und verschiedene Methoden muessen miteinander verknuepft werden, bzw. sich ergaenzen,
- sie muessen auf alle entstehenden Zwischen- und Endprodukte des Entwicklungsganges angewandt werden,
- der Pruef-Prozess muss nachvollziehbar sein,
- er muss parallel und im Einklang mit den konstruktiven Entwicklungsmethoden durchgefuehrt werden,
- und: er muss unabhaengig vom Entwicklungsprozess sein.

3.1 Test

Test ist die offensichtlichste Methode der Fehlersuche. Er besteht im wesentlichen aus dem Vergleich des beobachteten Verhaltens mit dem erwarteten Verhalten des Systems oder Programms. Voraussetzung fuer eine erfolgreiche Testdurchfuehrung ist die unbedingte Absicht, Fehler zu finden. Das bedeutet eine kritische Haltung gegenueber dem zu testenden SW-Produkt und die Ausrichtung saemtlicher Massnahmen auf die Fehlererkennung.

Testobjekte sind dabei sowohl Dokumente als auch Programme. Die Testausfuehrung kann manuel erfolgen oder durch Tools unterstuetzt sein.

An einen "guten" Test kann man folgende Anforderungen stellen:

- er muss einen Fehler finden;
- er muss einfach sein;
- er muss dokumentiert werden;
- er muss nachvollziehbar sein;
- er muss auf eine ganz bestimmte Fehlerklasse ausgerichtet sein (z.B. Modultest, Integrationstest, Funktionstest, Systemtest).

In den Testvorgaben, d.h., bei der Testplanung muss beachtet werden:

- die Umgebung des Testobjekts;
- moegliche Zwischen- und Vorprodukte;
- zugehoerige Entwicklungsdokumente;
- moeglichst reale Verarbeitungsfaelle und Situationen;
- der Entwicklung zugrundeliegende Regeln und Standards;
- Qualitaetsanforderungen an das Testobjekt;
- potentielle Fehler.

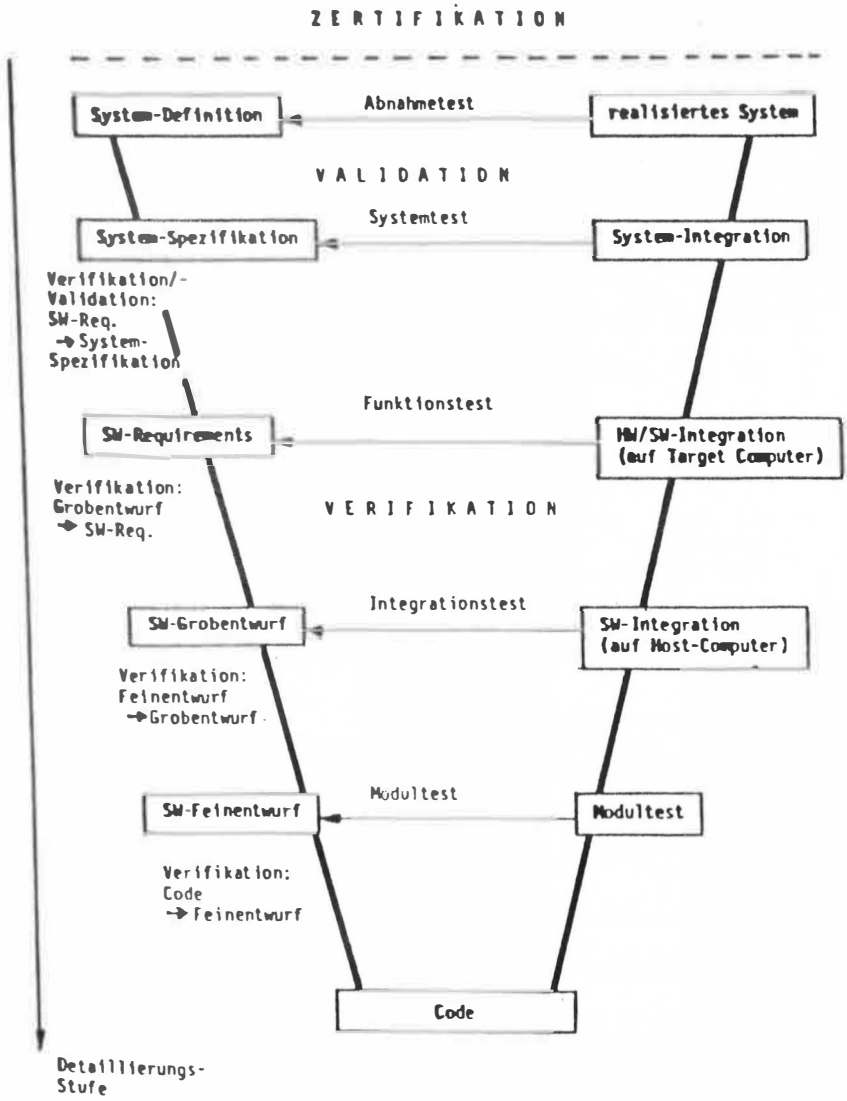

Bild 4: Validations- Verifikations- und Testaktivitaeten im SW-Life-Cycle

3.2 Verifikation und Validation im SW-Life-Cycle

Die Beziehungen der Verifikations- und Validations-Aktivitaeten untereinander werden durch das Diagramm Bild 4 wiedergegeben. In diesem Diagramm ist der zeitliche Projektablauf V-foermig aufgezeichnet, beginnend mit der System-Life-Cycle Phase "System-Design", endend mit der Systemphase "Systemintegration". Diese Systemphasen umschliessen den SW-Life-Cycle. Tiefste, d.h. detaillierteste Phase ist die SW-Codierungsphase.

Die meisten Validations- und Verifikationsaktivitaeten sind Test-Taetigkeiten. In den fruehen Life-Cycle-Phasen steht der Test der Definitions- und Entwurfs-Dokumente im Vordergrund. Mit Beginn der Codierungs- und Integrationstaetigkeiten kommt der Test der ablauffaehigen Module und Programme hinzu.

In den fruehen Life-Cycle Phasen verlaeuft die Verifikationsrichtung vertikal, d.h.,

- Die SW-Requirements werden gegenueber dem System-Design verifiziert.

  Sie werden ueberprueft auf interne Konsistenz, moegliche Konflikte mit anderen Requirements, Problembezogenheit, Klarheit im Aufbau und in der Definition, sowie Eindeutigkeit. Es muss gewaehrleistet sein, dass die SW-Requirements die sie betreffenden System-Spezifikationen korrekt und vollstaendig konkretisieren.

- Der SW-Grobentwurf wird gegenueber den SW-Requirements verifiziert.

  Die verwendeten Entwurfs-Strukturen werden analysiert bezueglich Korrektheit, Klarheit im Aufbau und in der Definition, Konsistenz, Eindeutigkeit und Problembezogenheit. Es muss nachgewiesen werden, dass der Entwurf die zugrundeliegenden SW-Requirements korrekt implementiert.

- Der SW-Feinentwurf wird gegenueber dem SW-Grobentwurf verifiziert.

  als eine weitere Detaillierungsstufe der Entwurfsverifikation.

- Der Code wird gegenueber dem SW-Feinentwurf verifiziert.

  Der Code wird untersucht auf syntaktische und semantische Korrektheit, die Einhaltung von Codierstandards, Klarheit im Aufbau, Konsistenz, Eindeutigkeit und Problembezogenheit. Es muss nachgewiesen werden, dass der Source-Code alle spezifizierten Entwurfs-Elemente realisiert.

In den weiteren SW-Life-Cycle-Phasen verlaeuft die Verifikationsrichtung horizontal. Verifiziert wird hier ueberwiegend durch Tests, die abgestuft auf verschiedene Fehlerklassen ausgerichtet werden.

Im einzelnen:

- Der Modultest erfolgt gegen die Modul-Testvorgaben der SW-Feinentwurfs-Phase.

  Fehlerklasse: Codierfehler.

- Die SW-Integrationstests (Modulintegrationstests) auf den Host-Computern werden entsprechend den Testplaenen der SW-Grobentwurfs-Phase durchgefuehrt.

  Fehlerklasse: Schnittstellenfehler zwischen den einzelnen Modulen.

- Der HW/SW-Integrationstest erfolgt anhand der Funktionstestplaene der SW-Requirement-Phase.

  Fehlerklasse: Schnittstellenfehler zwischen HW und SW-Komponenten, Test des dynamischen Verhaltens des realisierten Systems in Echtzeit.

- Danach werden in der Systemintegrationsphase die Systemtests gemaess den Vorgaben der Systemdesign-Phase durchgefuehrt.

  Fehlerklassen: Funktionsmaengel, Stress, Konfiguration, Verfuegbarkeit, Maengel/Fehler in der Mensch-Maschine-Schnittstelle.

- Erst dann werden Abnahmetests durchgefuehrt, die dann evtl. mit der Zertifikation des Systems abgeschlossen werden.

In den ersten und in den letzten Phasen tritt bei der Verifikation der Validations-Aspekt besonders in den Vordergrund d.h. die SW-Requirements und das realisierte HW/SW-System werden verstaerkt gegenueber den nutzer-spezifischen Anforderungen oder Vorgaben ueberprueft.

4 SCHLUSSBEMERKUNGEN

Es gibt zwei sich ergaenzende Methoden zur Erreichung zuverlaessiger Software:

1. Konstruktive Verfahren fuer die Entwicklung der SW;

   -- um Fehler erst gar nicht entstehen zu lassen.

2. Analytische Verfahren zur Pruefung der SW;

   -- um Fehler moeglichst bald nach ihrer Entstehung zu finden und zu beseitigen.

Keines der Verfahren allein erzielt eine "magische" Wirkung. Der Weg zum SW-Produkt, das hohen Zuverlaessigkeitsanspruechen genuegt und dessen Erstellung gleichzeitig effizient und wirtschaftlich erfolgt, fuehrt nur ueber den Einsatz beider Methoden: die Verwendung einer dem zu erstellenden System angepassten Entwicklungsmethode, begleitet durch Verifikations- und Test-Taetigkeiten.

Fuer die Wirksamkeit analytischer Verfahren ist ein integriertes und abgestuftes Verifikations- und Testkonzept erforderlich. Es muss ausgerichtet sein auf die Zuverlaessigkeits-Anforderungen des Systems und auf die projektspezifischen Gegebenheiten, insbesondere auf die konstruktive Entwicklungsmethodik. Voraussetzung fuer eine effektive Planung der Pruefmassnahmen ist die Kenntnis von Fehlerarten, Ursachen und Auswirkungen. Dies erleichtert die Ausrichtung der verfuegbaren Mittel auf das Erkennen und die Behandlung moeglicher Schwachstellen und traegt so zur Erhoehung der Zuverlaessigkeit des SW-Systems bei.

5 LITERATUR

[1] BMV-Studie, 1986 — Pruefung und Zulassung von softwareabhaengigen Systemen in Luftfahrzeugen

[2] MIL-STD-882B — System Safety Program Requirements

[3] HQ-AFISC/SESD — Software Safety Handbook, Draft March 84

[4] G. Heiner, AGARD-CP-261 — Introduction to Software Reliability

[5] J. Goldberg, AGARD-CP-261 — Formal Methods for Achieving Reliability

[6] Rault/Memmi/Pimont, AGARD-CP-261 — Quantitative Assessments of Software Reliability

[7] B.W. Boehm, Prentice-Hall — Software Engineering Economics

[8] NTG-Empfehlung 3004

[9] RTCA /DO 178 — Software Considerations in Airborne Syste and Equipment Certification

[10] B. W. Boehm, EURO IFIP 1979 — Guidelines for Verifying and Validating Software-Requirements and Design-Specifications

[11] W. E. Howden, ACM Computer Survey, Bd. 14, 1982 — Validation of Scientific Programs

[12] W. R. Adrion, M. A. Branstad, ACM Computer Survey, Bd. 14, 1982 — Validation, Verification and Testing of Computer Software

[13] G. J. Myers, Oldenburg Verlag GmbH, 1982 — Methodisches Testen von Programmen

[14] D. J. Reifer, AGARDograph NO. 258, 1980 — Software Verification and Validation

[15] K. Grimm, NTG/DGQ-Tagung Mai 1985, Nuernberg — Klassifizierung und Bewertung von Softwar Verifikationsverfahren

[16] K. Grimm, G. Heiner
Prozessrechner-Tagung
September 1984, Karlsruhe
Richtlinien zur Erstellung und Pruefung sicherheitsrelevanter Software

[17] EWICS TC7
Verlag TUEV-Rheinland
Savety Related Computers

[18] M. S. Deutsch
Prentice-Hall Inc.
Software Verification and Validation

[19] P. Schmitz, H. Bons
R. van Megen
Software-Qualitaetssicherung - Testen im Software-Lebenszyklus

[20] M. L. Shooman
McGraw-Hill
Software Engineering

[21] P. Elzer, T. Schuler
ZTL-Studie, 1980
Programmverifikation beim Einsatz hoeherer Programmiersprachen in digitalen Flug-fuehrungssystemen

[22] K.H. Moeller
NTG/DGQ-Tagung
Mai 1985, Nuernberg
Fehlerverteilung als Hilfsmittel zur Qualitaetsverbesserung und Fehlerprognose