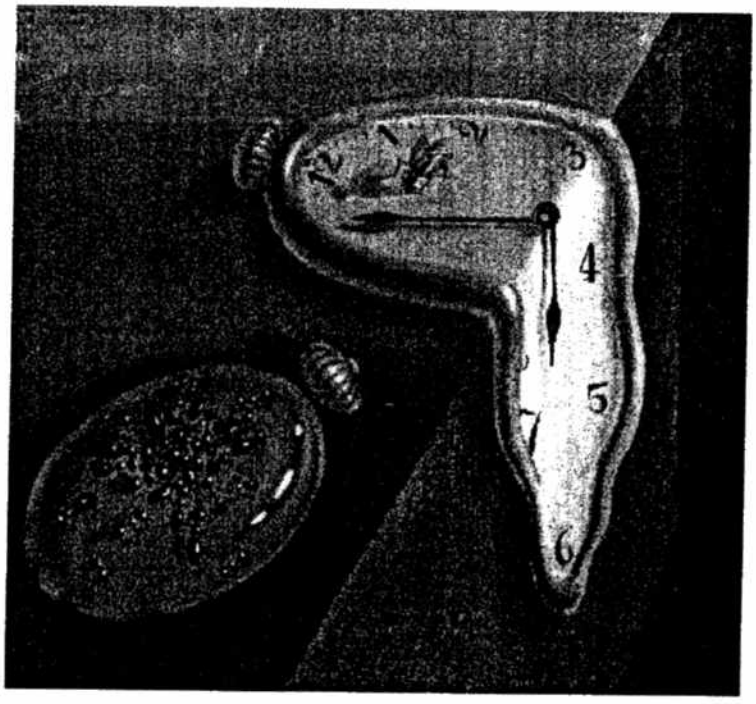
Institut für Automation
Abt. für Automatisierungssysteme

Technische
Universität
Wien

Projektbericht Nr. 183/1–5
Oktober 1988

Industrielle Softwareentwicklung und Qualitätssicherung

G.H. Schildt



Ausschnitt aus: Salvador Dali, "Die Beständigkeit der Erinnerung"

INDUSTRIELLE SOFTWAREENTWICKLUNG UND QUALITÄTSSICHERUNG

Gerhard H. Schildt

1 EINFÜHRUNG

Verfahren der industriellen Softwareentwicklung und Maßnahmen zur Qualitätssicherung von Software haben besondere Bedeutung für sicherheitsrelevante Prozeßsteuerungen. Dazu gehören spurgebundene Verkehrssysteme, die chemische Verfahrenstechnik und Steuerungen/Regelungen im Kraftwerksbereich. Unter Systemen mit Sicherheitsverantwortung versteht man solche Systeme, von denen im Fall einer Störung, eines gerätetechnischen Ausfalls oder eines sich offenbarenden Entwurfsfehlers keine Gefährdung für Personen ausgehen darf. Besonders auf dem Gebiet der Eisenbahnsignaltechnik sind Systeme so zu entwerfen, daß signaltechnische Sicherheit nach DIN 57 831 bzw. VDE 0831 zur Vermeidung unzulässiger Fehlzustände gewährleistet wird [1]. Im folgenden werden Aspekte zur industriellen Softwareentwicklung und Qualitätssicherung bei Prozeßsteuerungen betrachtet.

2 INDUSTRIELLE SOFTWAREENTWICKLUNG

Im Gegensatz zur Hardware, wo Fehlfunktionen auf nach der Inbetriebnahme auftretende Ausfälle oder Störungen zurückgehen, gilt für die Software, daß ihre Fehler Entwurfsfehler sind, und damit bereits vor der Inbetriebnahme existent sind.

Wird solche einkanalige Software auf einem zweikanaligen Rechnersystem eingesetzt, so können Gefährdungen auftreten, wenn diese Software nicht fehlerfrei ist. Beim Einsatz von sicheren Rechnersystemen ((2 von 2) - oder (2 von 3) - Rechnersystemen) für sicherungstechnische Aufgaben der Eisenbahnsignaltechnik ist der Nachweis der Fehlerfreiheit der eingesetzten Software (d.h. Anwendersoftware u n d Betriebssystemsoftware) von entscheidender Bedeutung.

2.1 Entwicklungs-Prozeß-Plan

Bei der Softwareentwicklung geht man zweckmäßigerweise von einem Phasenmodell nach Bild 1 aus. Es gliedert den Prozeß der Softwareentwicklung in insgesamt 8 Phasen beginnend mit der Phase des Projektanstoßes über die eigentliche Softwareentwicklung bis hin zum Einsatz.

A Vorstudie
B Spezifikation
C Systementwurf
D Programmentwurf
E Implementierung
F Integration
G Inbetriebnahme
H Einsatz/Betreuung

Bild 1: Phasen der Softwareentwicklung

Diesem Phasenmodell entsprechend kann ein Entwicklungs-Prozeß-Plan für Software aufgestellt werden, der zu jeder der insgesamt 8 Phasen der Softwareentwicklung für jede Phase dem Entwickler eine C h e c k l i s t e an die Hand gibt. Damit soll erreicht werden, daß zu jeder Entwicklungsphase die der Phase entsprechenden Aktionen ausgelöst werden.
Bild 2 zeigt an einem Beispiel für die Phase 'Systementwurf' als Checkliste eine Übersicht über die am Schluß dieser Phase zu erstellenden Dokumente.

PHASENEINTEILUNG

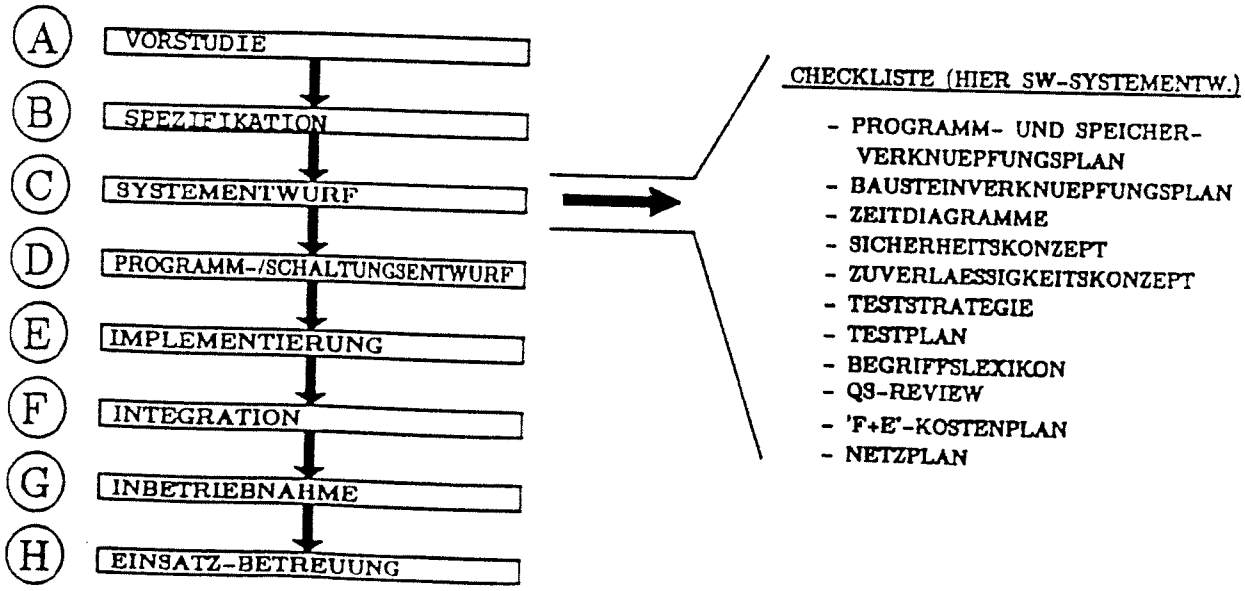


Bild 2: Entwicklungs-Prozeß-Plan mit Checkliste

Bild 3 zeigt den zeitlichen Ablauf der Softwareentwicklung. Dabei werden der ideale und der reale Entwicklungsverlauf mit nicht-planbaren Phasenrücksprüngen untereinander dargestellt. Für den Fall eines unvermeidbaren Phasenrücksprungs, der über mehrere Phasen hinweg gehen kann, ist festzulegen, daß alle Dokumente der erneut zu durchlaufenden Phasen aktualisiert werden.

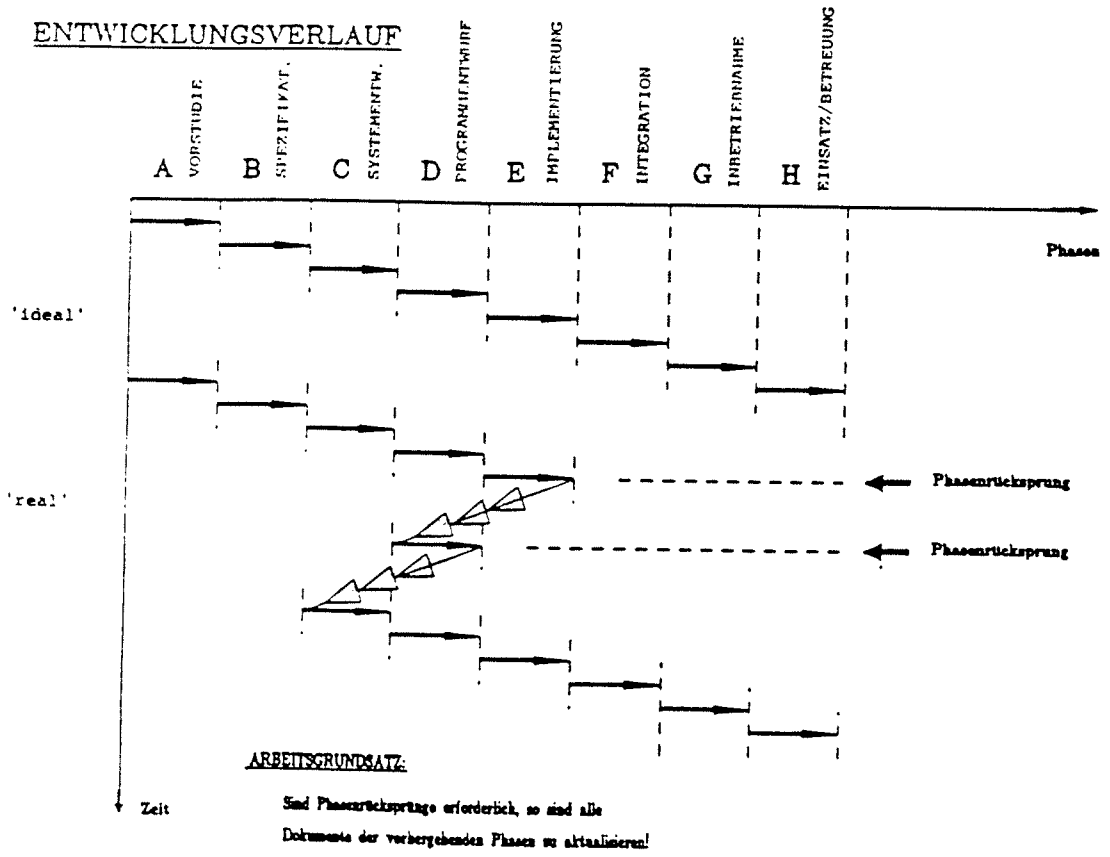


Bild 3: Entwicklungsverlauf, ideal und real mit Phasenrücksprüngen

Der Entwicklungs-Prozeß-Plan ist vor allem als Checkliste für Software-Projektleiter geeignet. Obwohl der Entwicklungs-Prozeß-Plan eine sinnvolle Entwicklungsunterstützung darstellt, muß aber festgestellt werden, daß unter dem Einfluß drängender Termine die Akzeptanz für die Nutzung des Entwicklungs-Prozeß-Plans nicht genügend gegeben ist.

2.2 Rechnerunterstützung

Die Erfahrung der letzten Jahre der Softwareentwicklung zeigt,

daß Fehler in der Spezifikations- und Entwurfsphase besonders hohe Kosten verursachen. Es ist daher in diesen Phasen Rechnerunterstützung einzusetzen mit folgenden Zielen:

- Kostenreduktion
- Qualitätssicherung
- Produktivitätssteigerung
- Projektkontrolle.

Es sind bereits zahlreiche rechnerunterstützte Entwurfssysteme am Markt verfügbar. Bild 4 gibt eine Übersicht über einige Softwareentwicklungsumgebungen mit Angaben über die Rechnerunterstützung in den Phasen 'Requirements Engineering', 'Systementwurf' und 'Implementierung'. Außerdem finden sich Angaben zu den Aufgaben der Software-Projektkontrolle.

	BOIE	CADOS	COMPASS	EPOS	EXCELLERATOR	PRADOS	PROMOD
Requirements Engineering	o	+	o	+	+	+	+
Systementwurf	+	+	+	+	+	+	+
Implementierung	+	o	+	+	o	+	+
Projektmanagement	-	o	-	+	o	o	+
Versionsverwaltung	-	+	-	+	o	+	o
Qualitätssicherung	-	o	-	o	-	-	-

Legende:
+ umfassende ...
o bedingte ...
- keine ...

Rechnerunterstützung

Bild 4: Übersicht über einige Softwareentwicklungsumgebungen

Als eines der am weitesten entwickelten Systeme kann die Softwareentwicklungsumgebung EPOS (Entwurfsunterstützendes

Projektmanagement-orientiertes Spezifikationssystem) benannt werden [2], das außer der Unterstützung bei der Spezifikation, der Entwurfsphase und der Implementierung eine Projektmanagement-komponente zur Projektkontrolle umfaßt.

Erfahrungen. Bisher wurde das System EPOS auch im Bereich der Eisenbahnsignaltechnik eingesetzt (z.B. bei der Spezifikation des 16-Bit-Elektronischen Stellwerks oder zur Spezifikation einer Zuglenkstrategie bei einem Stadtbahnunternehmen). Es muß allerdings festgestellt werden, daß die Akzeptanz gegenüber solchen Softwareentwicklungsumgebungen bei den Softwareentwicklern noch nicht ausreicht. Dies liegt vor allem an den z.T. veralteten Bedienoberflächen (das System EPOS ist seit 1987 auf APOLLO-Rechner portiert worden und stellt sich seither mit einer wesentlich verbesserten Bedienoberfläche dar).

Die in Bild 4 aufgeführten Softwareentwicklungsumgebungen unterstützen die Softwareentwicklung vor allem in den Anfangsphasen. So fehlt es noch an Verfahren, die den g e s a m t e n Softwareentwicklungsprozeß unterstützen. Weiter fehlt z.Zt. noch die Möglichkeit, aus vergangenen Projekten teilweise Software in neue Projekte zu integrieren (sog. Behandlung wiederverwendbarer Software). Hier sind künftig Weiterentwicklungen der bestehenden Verfahren auf der Grundlage wissensbasierter Systeme bzw. Expertensysteme erforderlich.

Die z.Zt. in der Industie noch überwiegend eingesetzten Verfahren zur Erarbeitung der Anforderungsspezifikation ('requirement specification') erzeugen informelle Spezifikationen (z.B. verbale Leistungsbeschreibung, Schnittstellenbeschreibung, Funktionsbeschreibung, Beschreibung einer Datenbasis, Ablaufbeschreibung, ...). Alle so entstandenen Dokumente sind damit im Rahmen eines Funktions- oder Sicherheitsnachweises n i c h t weiterverarbeitbar. Diesem Nachteil kann nur durch den Einsatz geeigneter Softwareentwicklungsumgebungen mit Spezifikationssprache begegnet werden.

2.3 Programmentwicklung

Ausgehend von den Dokumenten der Phasen 'Spezifikation' und

'Systementwurf' schließt sich die Phase des 'Programmentwurfs' auf der Grundlage der strukturierten Programmierung nach N a s s i und S h n e i d e r m a n an [3].
Für sicherheitsrelevante Programme ist bereits die Fehlerfreiheit der Dokumente der Spezifikation und des Systementwurfs zu zeigen. Die Anwendersoftware wird nach dem Top-down-Verfahren entwickelt. Zunächst wird eine funktionsbezogene Baumstruktur nach Bild 5 erstellt.

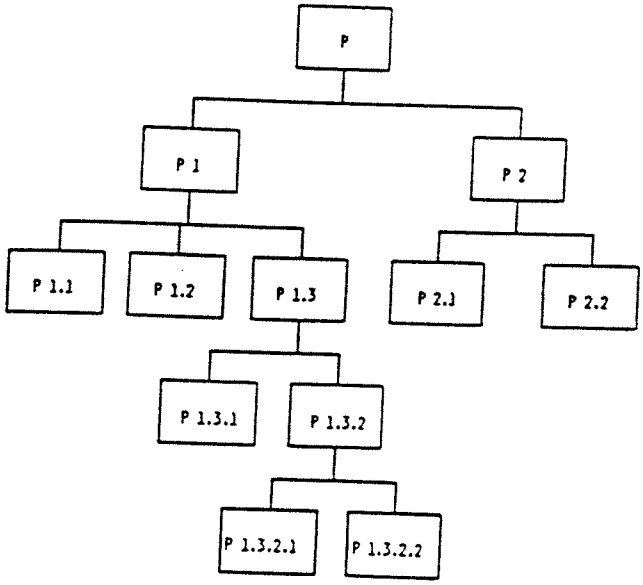


Bild 5: Funktionsbezogene Baumstruktur

Bei jedem Entwicklungsschritt wird eine vollständige Zerlegung des letzten Entwurfs in logische Einheiten durchgeführt, die Strukturblöcke genannt werden [4]. Dabei bilden jeweils alle in einem Entwicklungsschritt neu entworfenen Strukturblöcke eine weitere Entwicklungsebene. Die in den aufeinanderfolgenden Entwicklungsschritten entstehenden Entwürfe sind durch zunehmende Detailtiefe gekennzeichnet. Dabei wird jede Ebene der Softwareentwicklung der Detailtiefe entsprechend dokumentiert. Sobald eine Ebene erreicht ist, die sich bereits in der gewählten Programmiersprache beschreiben läßt, wird mit der Implementierung und dem anschließenden Test begonnen.
Die strukturierte Programmierung geht von den logischen Grundstrukturen der R e i h u n g (Folge, Sequenz), A u s w a h l (Verzweigung, Selektion) und der W i e d e r h o l u n g (Schleife, Iteration) aus.
Für die Entwicklung sicherheitsrelevanter Software beschränkt man sich aus Gründen der Transparenz auf die folgenden sechs

logischen Konstrukte:

- ELEMENTARE ANWEISUNG, -IF-THEN-ELSE-ABFRAGE, -CASE, -WHILE, -CYCLE und UNTERPROGRAMMAUFRUF (Bild 6).

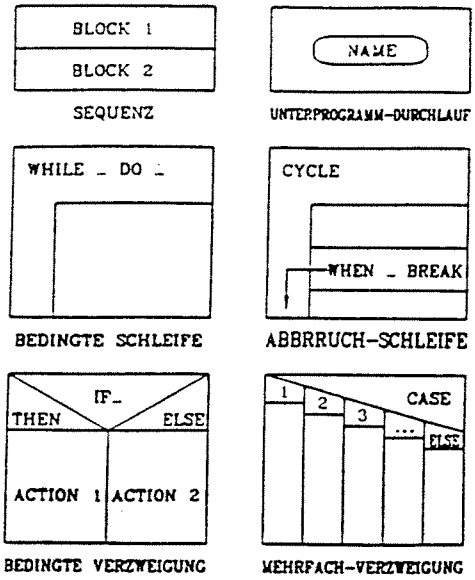


Bild 6: Logische Konstrukte für sicherheitsrelevante Systeme

2.4 Richtlinien

Der Softwareentwicklungsprozeß wird nach dem derzeitigen Stand durch eine Reihe von Richtlinien unterstützt. Tabelle 1 gibt eine Übersicht über vorhandene Richtlinien [5], die sich auf die Gebiete der Programmentwicklung, Dokumentation, Prüfung und Test sowie Änderungsverfahren beziehen:

1. Sicherungstechnische Programmierung, Richtlinie BZA-Entwurf 42 500, 42 520, 42 530, 42 540, 42 550
2. Sicherungstechnische Programmierung, Richtlinie P 25023-A0000-A021-X-35
3. Dokumentationsrichtlinie P 25 023-A0000-A018-X-35
4. QS-Richtlinie Nr. 2.01 Qualitätsmerkmale f. Software
5. Archivierung von Software, Richtlinie P 25 025-A0000-A017-X-35

Tabelle 1: Übersicht über vorhandene Richtlinien

Zum Nachweis der Einhaltung der genannten Richtlinien ist für

sicherheitsrelevante Software eine Zusammenarbeit zwischen dem systementwickelnden Unternehmen, dem künftigen Betreiber und einer unabhängigen Prüferinstitution anzustreben (Bild 7).

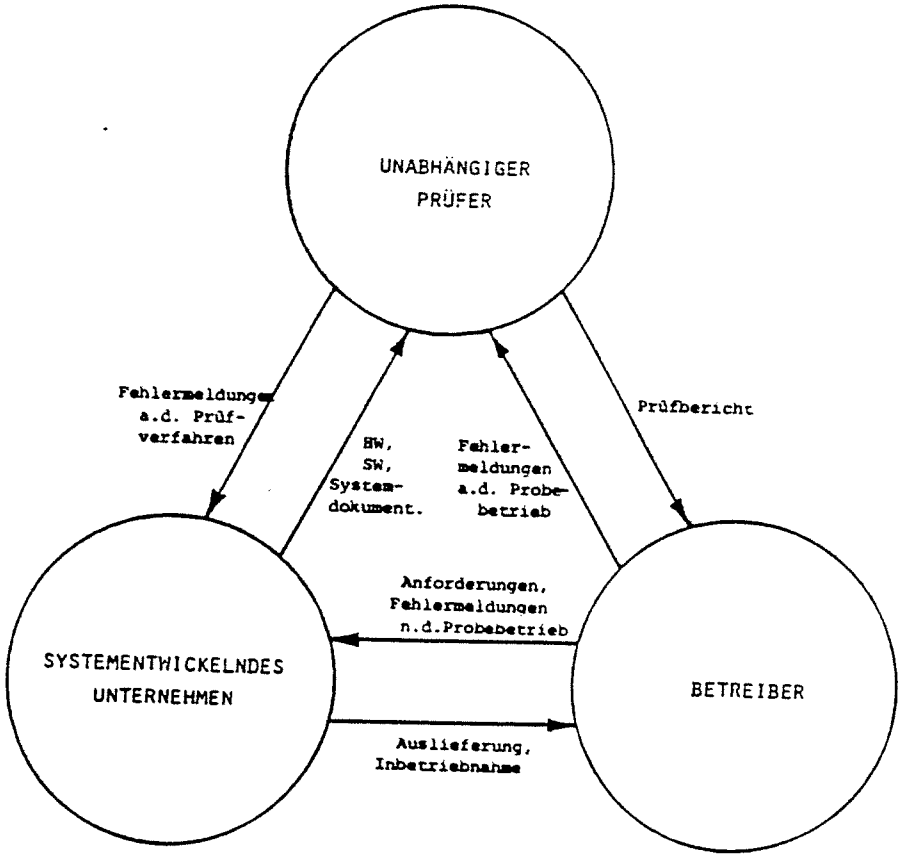


Bild 7: Zusammenarbeit von systementwickelndem Unternehmen, künftigem Betreiber und unabhängiger Prüferinstitution

Insbesondere der Prüferinstitution fällt die Rolle der Herausgabe verbindlicher Richtlinien zu. Wurden Richtlinien vom systementwickelnden Unternehmen oder dem künftigen Betreiber erstellt, so ist für die künftige, verbindliche Anwendung dieser Richtlinien durch die unabhängige Prüferinstitution die Vollständigkeit und Richtigkeit dieser Richtlinien zu zeigen.

3 SOFTWARE-QUALITÄTSSICHERUNG

Die Qualität eines automatisierten Systems wird in den 'requirement specifications' dadurch beschrieben, daß Angaben über das R e a l z e i t v e r h a l t e n und die A u s f a l l s i c h e r h e i t des Gesamtsystems durch Angaben über die S o f t w a r e - Q u a l i t ä t ergänzt werden.

3.1 Begriffe und Normen

Die Norm DIN 55 350 (Teil 1) [6] definiert den Begriff der Qualität wie folgt:

> 'Qualität ist die Gesamtheit von Merkmalen und Eigenschaften einer Betrachtungseinheit, die sich auf die Eignung zur Erfüllung gegebener Anforderungen beziehen.'

Dabei kann unter einer Betrachtungseinheit ein Produkt, eine Dienstleistung, Hardware oder Software verstanden werden. Software-Qualität ist damit genau die Erfüllung der Spezifikation. Kriterien für Software-Qualität sind

- Funktionserfüllung (functional performance)
- Zeitverhalten (time behaviour)
- Wartungsfreundlichkeit (maintainability)
- Mensch-Maschine-Schnittstelle (human interface)
- Portabilität (portability)
- Verbrauchsverhalten (consume behaviour)

Für die Sicherung der Software-Qualität gilt folgender Grundsatz: Software-Qualität kann nicht 'erprüft' werden, sie ist vielmehr kausal an den Software-Entwicklungsprozeß geknüpft. Eine möglichst durchgängige Rechnerunterstützung in den Phasen der Softwareentwicklung in Verbindung mit dem Einsatz leistungsfähiger Tools sind die besten Voraussetzungen für Software-Qualität.

3.2 QS-Richtlinie

Erste quantifizierte Angaben zur Software-Qualität finden sich in einer Firmenrichtlinie zur Qualitätssicherung [7]. Diese Richtlinie gibt meßbare Software-Qualitätsmerkmale an. Bild 8 macht auszugsweise Angaben über Korrektheit, Komplexität, Effizienz und Reserven.

Meßbare Software-Qualitätsmerkmale

1. Korrektheit	O Anzahl der Fehler aus den Phasen	
	- Programmentwurf und Implementierung	20/KLOC
	- Systemintegration und Systemtest	1/KLOC
	- Einsatz (aus dem Restfehlergehalt im 1.Jahr)	1/KLOC
	O Bei Übergabe bekannte, noch nicht behobene Fehler	
	- Fehler hoher Priorität	0
	- Fehler mittlerer und niedrieger Priorität	
	• bei Übergabe an Systemintegration	0.2/KLOC
	• bei Freigabe für Einsatz	0.2/KLOC
	• nach 1 Jahr Einsatz	0.5/KLOC
2. Struktur (Komplexität)	O Maximale Prozedurgröße	200 LOC
	O Maximale Prozedurschachtelung	3
	O Maximale Blockschachtelung	7
3. Effizienz, Performance	O Laufzeit für zeitkritische Funktionen	produktbezogen
	O Speicherbedarf je Funktion	vorgegeben
	O BHCA (Busy Hour Call Attempts), Responsetime	
4. Reserven bei Freigabe	O Dynamische Reserven bei CPU-Auslastung	> 20%
	O Statische Reserven bei Speicherbelegung	> 20%

Bild 8: Auszug aus der QS-Richtlinie 2.01

Im Verlauf des Software-Entwicklungsprozesses ist zu zeigen, daß diese Vorgaben eingehalten werden.

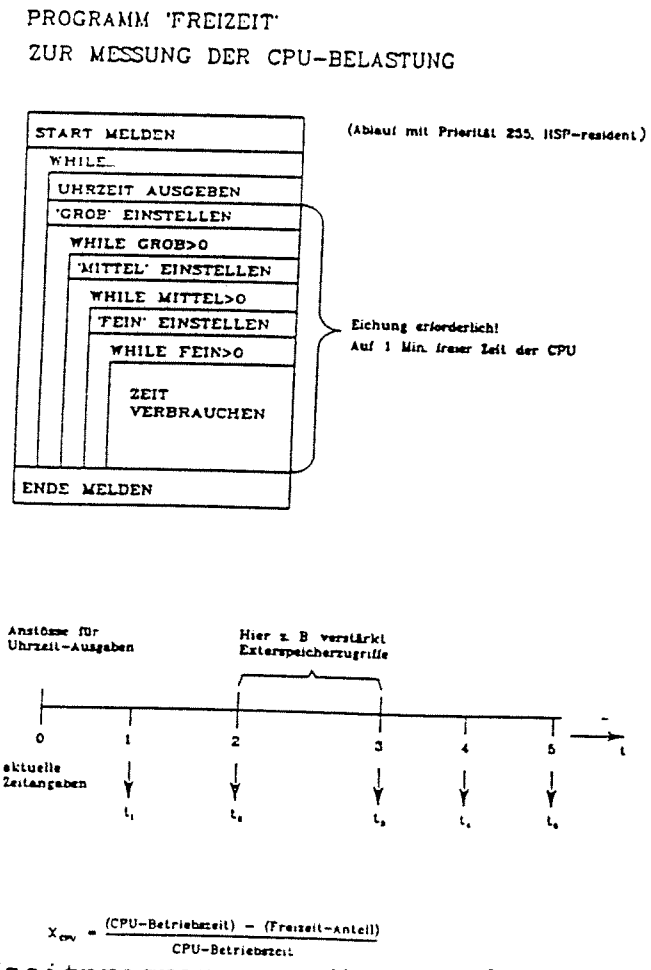
3.3 Maßnahmen zur Software-Qualitätssicherung

Bei jedem Projekt ist entsprechend dem Entwicklungs-Prozeß-Plan für jede Phase ein Qualitätssicherungsplan (QS-Plan [1]) festzulegen. Dieser QS-Plan umfaßt Angaben über einzuhaltende Standards, Software-configuration-management, Reviews/Audits, Dokumentation in der jeweiligen Phase sowie weitere restriktive Festschreibungen [8].
In Abschnitt 3.2 wurden Angaben zur Prozedur- und Blockschachtelung gemacht. Der Nachweis, daß der Sourcecode den dargestellten Qualitätsanforderungen entspricht, kann dadurch geführt werden, daß die Prozedurschachtelung (3) und die Blockschachtelung (7) mit Hilfe eines Source-Code-Analyzers gemessen und dokumentiert wird.

Für Prozeßsteuerungen mit Sicherheitsverantwortung war es bisher

1) Nach dem IEEE-Standard 730-1984 wird der QS-Plan als SQAP (Software Quality Assurance Plan) bezeichnet.

möglich, einkanalige Fail-Safe-System in Hardware zu realisieren. Mit zunehmender Komplexität der Automatisierungsaufgaben wurde zu dem Prinzip 'Sicherheit per Verfahren' übergegangen. Diese Vorgehensweise erweist sich auch bei rechnergesteuerten Anlagen als sinnvoll, wenn man neben den Anwenderprogrammen noch Laufzeitprogramme installiert. So kann z.B. die CPU-Belastung dadurch erfaßt werden, daß wenn kein Prozeßprogramm bearbeitet wird, ein Programm (z.B. FREIZT) mit schlechtester Priorität gestartet wird. Sobald ein Anwenderprogramm geladen und bearbeitet wird, wird das Laufzeitprogramm FREIZT vom Prozessor verdrängt. Das Prinzip des Sondenprogramms besteht darin, Programmlaufzeit zu verbrauchen. Bild 9 zeigt ein Struktogramm für das Programm FREIZT. Es besteht aus drei geschachtelten Laufanweisungen. Das Programm wird maschinenspezifisch kalibriert, daß -wenn kein anderes Anwenderprogramm bearbeitet würde- zu jeder Minute eine Uhrzeitausgabe ausgestoßen wird. Bild 9 zeigt im unteren Bereich ein zufälliges Zeitdiagramm mit angestoßenen Uhrzeitausgaben zu den Zeitpunkten t_1 bis t_5. Die CPU-Belastung während einer Betriebsphase kann nachfolgend durch die Reihenfolge der Uhrzeitausgaben festgestellt werden.



$$x_{CPU} = \frac{(\text{CPU-Betriebszeit}) - (\text{Freizeit-Anteil})}{\text{CPU-Betriebszeit}}$$

Bild 9: Laufzeitprogramm zur Messung der CPU-Belastung

Um die Eignung einer Prozeßrechneranlage für den Echtzeitbetrieb nachzuweisen, kann ein Protokollierungsprogramm (PROT) eingesetzt werden, das den aktuellen Füllstandsgrad von Pufferspeichern A_v und den in einer Betriebsphase maximal aufgetretenen Füllstandsgrad S_v ('Schleppzeigerfunktion') auflistet. Auf diese Weise kann unter Echtzeitbedingungen nachgewiesen werden, daß sowohl die Pufferspeicher genügend groß dimensioniert wurden als auch keine Verklemmungen im System auftreten.

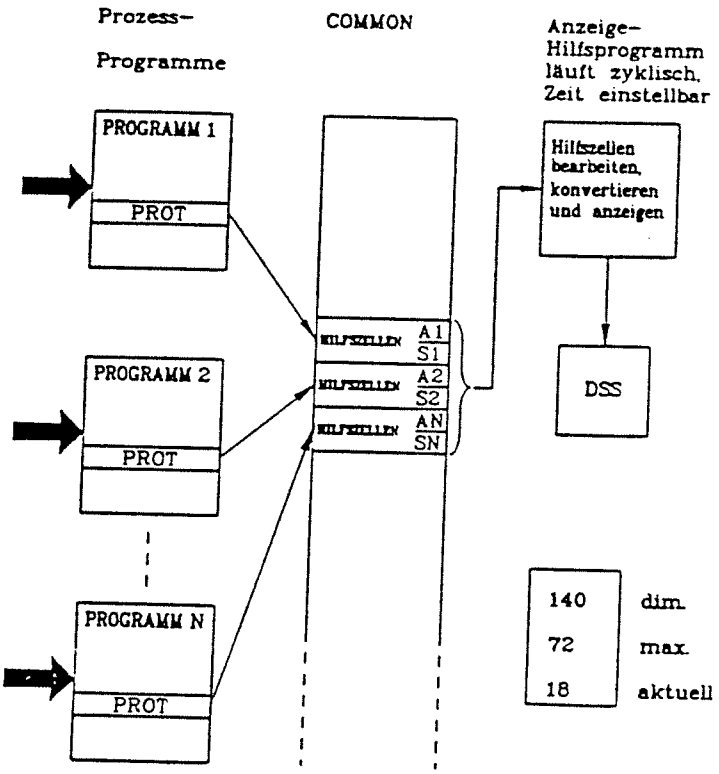


Bild 10: Protokollierungsprogramm zur Dokumentation von Füllstandsgraden von Pufferspeichern

Die Echtzeitfähigkeit eines Prozeßrechnersystems kann bezüglich der eintreffenden Prozeßmeldungen dadurch nachgewiesen werden, daß während einer Betriebsphase alle Prozeßmeldungen gespeichert werden und dann dem Prozeßrechnersystem mit n-facher Geschwindigkeit ($2 \leq n \leq 3$) eingespeist werden.

Eine noch ungelöste Aufgabenstellung bei Prozeßrechneranlagen besteht darin, das Systemverhalten zu analysieren, wenn Interrupt-Lawinen auftreten. Weiter ist bei interruptgesteuerten Systemen zu zeigen, wie das Prozeßrechnersystem gegenüber dem Prozeß reagiert, wenn Interrupts ausfallen.

In ähnlicher Weise ist nachzuweisen, daß der STACK-Bereich für PUSH-POP-Operationen hinreichend groß dimensioniert wurde und ein STACK-OVERFLOW ausgeschlossen werden kann. Um diesen Überlauf zu vermeiden ,sind rekursive Programmiertechniken für sicherheitsrelevante Programme auszuschließen. Als geeignetes Werkzeug zum Auffinden rekursiver Aufrufstrukturen kann das Programm BAUM eingesetzt werden. Dieses Programm analysiert die modulare Aufrufstruktur eines Programmsystems und kennzeichnet Bausteine, die an einer rekursiven Aufrufstruktur beteiligt sind (Bild 11).

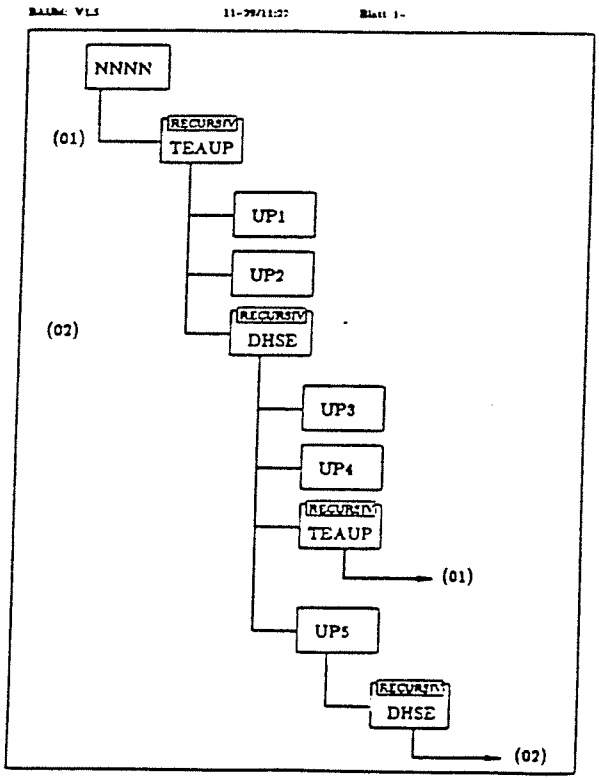


Bild 11: Baumdiagramm mit unzulässiger Rekursiv-Programmierung

Abschließend muß noch auf ein Problem bei der Software-Qualitätssicherung hingewiesen werden: Die Sicherung des Software-Qualitätsstandards bei zugekauften Programmen sowie bei im Unterauftrag erstellter Software. Hier sind an der Übergabestelle zum systementwickelnden Unternehmen 'Wareneingangskontrollen' auf der Grundlage des Qualitätsstandards des Auftraggebers durchzuführen.

4 ZUSAMMENFASSUNG

Im vorliegenden Beitrag wurden die industrielle Softwareentwicklung und Maßnahmen zur Software-Qualitätssicherung betrachtet. Insbesondere wurden sicherheitstechnische Aspekte behandelt. Von besonderer Bedeutung für die Softwareentwicklung sind rechnergestützte Softwareentwicklungsumgebungen und bei der Software-Qualitätssicherung die Entwicklung verbindlicher Richtlinien. Auf beiden Gebieten sind erste Schritte getan, es sind jedoch noch weiterführende Arbeiten erforderlich.

LITERATUR

1	DIN 57831 VDE 0 831	Signaltechnische Sicherheit (Normblatt)
2	Lauber, R., Lemp, P.:	Integrierte Rechnerunterstützung für Entwicklung, Projektmanagement und Produktverwaltung mit EPOS Elektron. Rechenanlagen 27 (1985), S. 68-74
3	Nassi, I., Shneiderman, B.:	Flowchart Techniques For Structured Programming, SIGPLAN Notices,1973, Nr.8 S. 12-16
4	Elbeshausen, E., Schildt, G.H.:	Automatisierter Testablauf Für Anwendersoftware bei Prozeßsteuerungen, Fachtagung Bremen, 1986
5	Siemens AG Bereich Eisenbahnsignaltechnik	Sicherheitsrichtlinien für den Sicherheitsnachweis Hardware und den Funktionsnachweis Software, Braunschweig 1985
6	DIN 55 350	Begriffe der Qualitätssicherung und Statistik, Teile 11-13 (Qualitätsmerkmale für Software)

7 Siemens AG
UB Kommunikationstechnik
QS-Richtlinie Nr. 2.01
Qualitätssicherung, Ausgabe
7/83 (Firmeninterne Norm)

8 IEEE
Standard for Software
Quality Assurance Plans,
IEEE Std 730-1984, New York,USA

9 Asam, R.,
Drenkard, N.,
Maier, H.-H.:
Qualitätsprüfung von Softwareprodukten, Siemens AG, München
1986

10 Bishop, P.G.:
Description of safety methods
and techniques, EWICS guidelines
TC 7 1987

Verfasser:
o.Univ. Prof. Dr.-Ing. Ing.(grad) Gerhard H. Schildt
Senior Member of IEEE
Institut für Technische Informatik der TU Wien
Treitlstraße 3
A-1040 WIEN