
Fehler und Fehlertoleranz

– Begriffe und Techniken –

„Perlen“ der „Weisheit“
28.11.2000



Max Breitling



Einführung

Beispiele für Fehler:

- Unerwartet ausgelöste Airbags
- Schreibfehler in Spezifikationen
- *Unerwarteter Systemfehler*
- Verwechslung miles/h und km/h
- Plattenlese-/schreibfehler
- (Nicht-)Terminierung
- Negativer Kontostand
- Verrutschte Bilder in Dokumenten



→ **Sehr unterschiedliche Arten von Fehlern!**

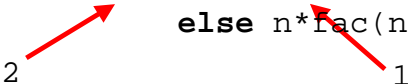
→ **Gemeinsamkeit: Abweichung „Ist“ von „Soll“.**

Einführung

Beispiel: Programm zur Berechnung der Fakultätsfunktion
(für $n > 1$)

Wo steckt der Fehler?

```
• fct fac(nat n) : nat
  if n=1 then 2
    else n*fac(n-1)
```



Wie viele Fehler hat folgendes Programm?

```
• fct fac(nat n) : nat
  if even(n) then n div 2
    else 3*n+1
```

Einführung

Der **Begriff des Fehlers** ist

- relativ zu einer Spezifikation zu definieren,
- möglicherweise schwer lokalisierbar und
- möglicherweise schwer quantifizierbar.

Beobachtung:

- Keine standardisierte Begriffsbildung in der Literatur
- Kein systematischer Umgang mit Fehlern

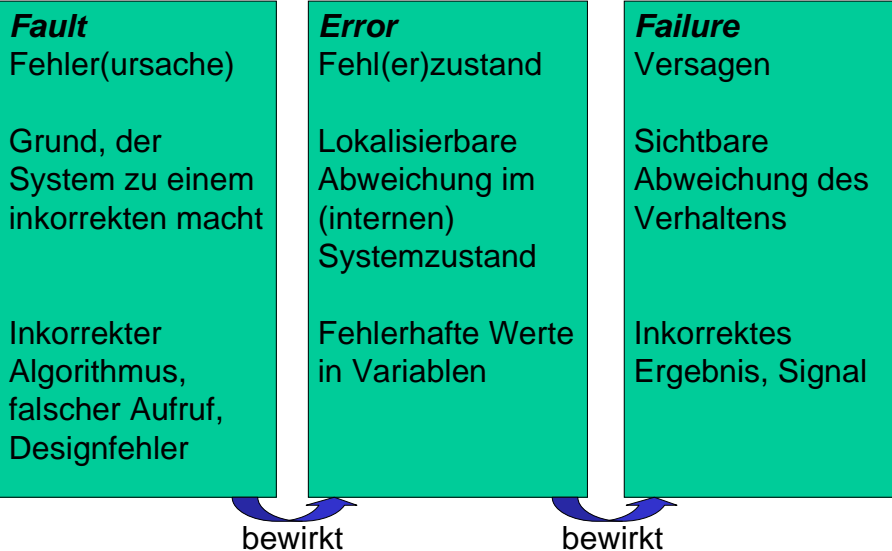
In diesem Vortrag:

- Zusammenstellung **allgemeiner, einigermaßen etablierter und wiederkehrender Ansätze**, aber
- keine anwendungsspezifischen Techniken,
- keine quantitativen Zuverlässigkeitsbewertungen.

Überblick

- Einführung
- Fehler
 - Grundbegriffe: *fault, error, failure*
 - Klassifizierung
 - Formale Ansätze
- Fehlertoleranz
 - Begriffsklärung
 - Redundanz
 - Fehlererkennung
- Techniken der Fehlertoleranz
 - Hardware
 - Software
- Zusammenfassung
- Literaturangaben

Begriffe



Klassifikation: Fehler

- Wo** ist er lokalisiert?
- intern/extern
 - lokal/verteilt
 - im Code/Implementierung
 - im Systemdesign/-entwurf
 - in der Spezifikation
 - in der Umgebungsannahme
 - in verwendeten Komponenten
 - in der Hardware/Software
- Wer** hat ihn gemacht?
- Mensch (Entwickler, Bediener)
 - Physikalische Umwelt
- Warum** wurde er gemacht?
- zufällig (Unkenntnis, Fahrlässigkeit, zu optimistische Annahmen)
 - absichtlich

Fehler und so ...

Max Breitling

7

Klassifikation: Fehler

- Wann** gerät er in System?
- Entwicklungszeit
 - Laufzeit
- Worin liegt sein **Ursprung**?
- Logischer Fehler
 - Physikalische Fehler
- Wie lange** dauert er (oder seine Wirkung) an?
- Permanente Fehler
 - Temporär
 - extern: transient,
 - intern: intermittierend
- Wie **schlimm** ist er?
- kritisch/unkritisch,
 - leicht/schwer behebbar,
 - tolerierbar

Fehler und so ...

Max Breitling

8

Klassifizierung: Versagen

- Art**
- Wertebezogen:
value, omission, fail-stop, crash failure, byzantine
 - Zeitbezogen:
early/late timing
- Auswirkungen**
- harmlos
 - schädlich
 - katastrophal
- Beobachtung**
- konsistent
 - inkonsistent

Ausfall: Versagen mit physikalischer Veränderung.

Umgang mit Fehlern

Zur Entwicklungszeit:

- Fehlervermeidung
Formale Methoden,
Software-Engineering, Requirements-Engineering, ...
- Finden von Fehlern
Reviews, Testen, ...
- Fehlerbeseitigung
Lokalisierung, Analyse, Behebung
- Integration von Fehlertoleranzverfahren

Zur Laufzeit:

- Fehlerentdeckung
 - Fehlerlokalisierung
 - Fehlermeldung
 - Fehlerbehebung
- } Fehlertoleranz

Fehlertoleranz

Fehlertoleranz kann nur relativ definiert werden:

- zur Menge **F** der zu tolerierenden Fehler
- zu einem geforderten Niveau, z.B.

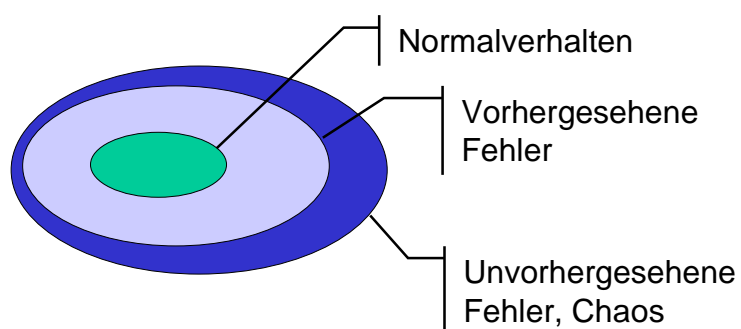
	Sicherheit	Lebendigkeit
Maskierend	X	X
Fail-Safe	X	-
Nicht-maskierend	-	X

Definierte Abschwächung der Eigenschaften:

Graceful Degradation (sanfte Leistungsminderung)

Fehlertoleranz

Abschwächung der Eigenschaften bei auftretenden Fehlern:



Formale Ansätze

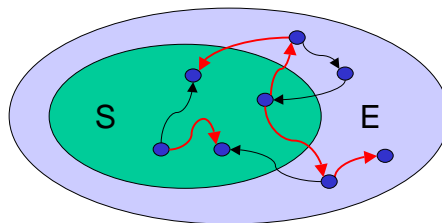
Systemmodell (Liu/Joseph, Arora/Kulkarni, Janowski, Gärtner):

Transitionssysteme mit Zuständen S, Transitionen T und Abläufen.

Fehlermodellierung

- Auch Fehler ändern den Systemzustand:
Fehler sind definierte Mengen **F** zusätzlicher (unerwünschter, aber unvermeidbarer) Transitionen
- Normalsystem P
Fehlerbehaftete Version $F(P)$
Korrigierte Version $R(F(P))$ oder $F(R(P))$
- Keine Berücksichtigung von Spezifikationsfehlern, Ausfällen, externen Fehlern, komponierten Systemen.

Formale Ansätze



S Normalzustände
E Fehlerzustände

Definition von **Fehlertoleranz** (Arora/Kulkarni):

- S abgeschlossen unter T
- E abgeschlossen unter T und **F**
- T führt irgendwann wieder in S

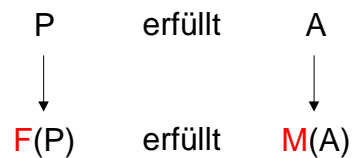
Maskierende Fehlertoleranz: $E = S$

Selbststabilisierende Systeme: $E = \text{true}$

Formale Ansätze

Korrespondierende Modifikationen **F** und **M** (Joseph/Peled):

Programm vs. Eigenschaften



Forderung (Janowski): „Fehler sind nicht verlässlich!“

Wenn S die Fehler **F** toleriert,
dann auch die Fehler $E \subseteq F$.

Fehlererkennung

Replikation	Ergebnisvergleiche
Umkehrungstests	„Probe“
Konsistenztests	Bereichsüberprüfung Datenstruktur-Tests Loopback-Tests
Zeitüberprüfungen	Erkennung verletzter Deadlines Watchdog-Timer
Codierungstests	Prüfsummen Erkennen verfälschter Instruktionen
Diagnoseläufe	Speichertests

Fehlerbehebung und -kompensation

Rückwärts- fehlerbehebung

Rücksprung zu gespeicherten
Rücksetzpunkten.
*allgemein, systematisch, aufwändig,
Rücksetzung von Kommunikation?*

Vorwärts- fehlerbehebung

Korrigierende Vorausrechnung.
*spezifisch, erfordert Schadensbestimmung,
effizient.*
Bsp.: Selbststabilisierende Systeme

Fehler- kompensation

Ausnutzen von Informationsredundanz
zum Kompensieren von Verlusten.
Bsp.: RAID-Systeme

Redundanz

Alle Formen von Fehlererkennung und Fehlertoleranz
erfordern **Redundanz** im System.

Redundanz: Teil des System, der bei Fehlerfreiheit nicht
notwendig wäre.

Es gibt verschiedene **Arten** von Redundanz, die meist
kombiniert auftreten:

Klassifikation: Redundanz

Strukturelle Redundanz	Vervielfältigung von Komponenten <i>baugleich oder alternative Entwürfe</i>
Funktionale Redundanz	zusätzliche, unterstützende Komponenten <i>Test, Konfiguration, Sensoren</i>
Informations-redundanz	Zusätzliche Informationen <i>Prüfbits, zusätzliche Zeiger, Zähler</i>
Zeitliche Redundanz	Zusätzliche Zeit für wiederholte Ausführungen (<i>Retry</i>)
Statische/ Dynamische Redundanz	Redundanz leistet <i>im Normalbetrieb/nur im Ausnahmefall</i> Beitrag zur Systemleistung

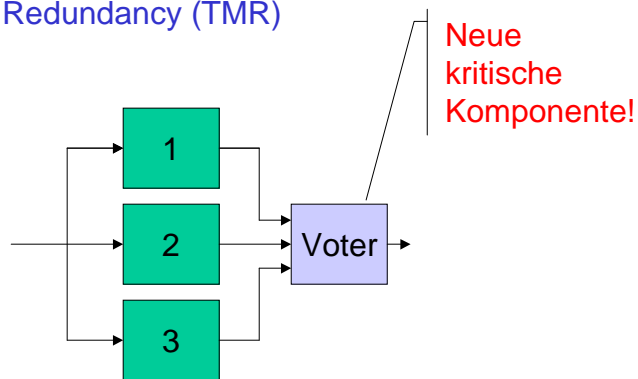
Fehler und so ...

Max Breitling

19

Techniken der Hardwarefehler-Toleranz

Triple Modular Redundancy (TMR)



Strukturelle, funktionale, statische Redundanz

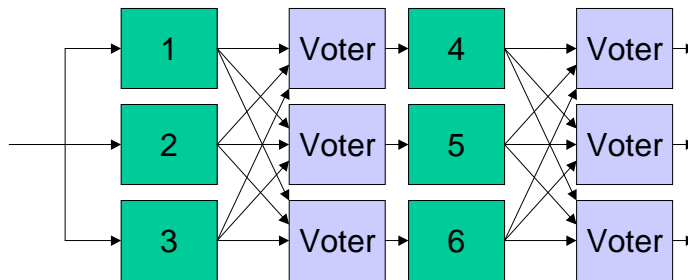
Fehler und so ...

Max Breitling

20

Techniken der Hardwarefehler-Toleranz

Mehrstufiges TMR



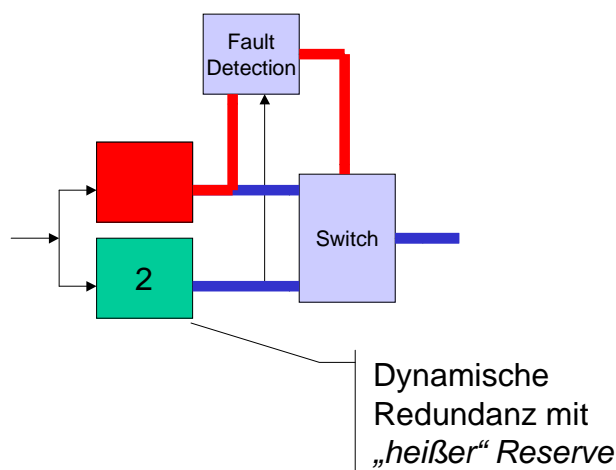
Fehler und so ...

Max Breitling

21

Techniken der Hardwarefehler-Toleranz

Reservekomponenten



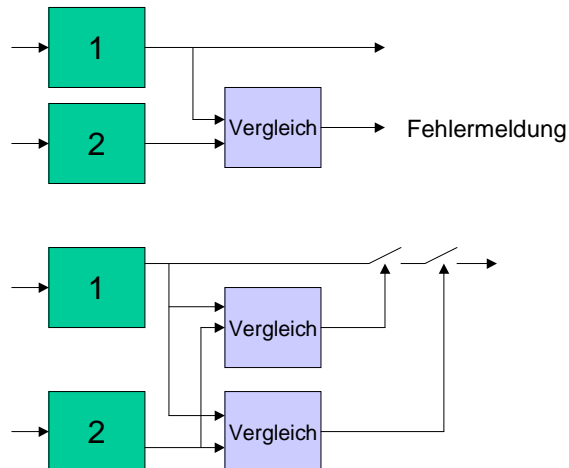
Fehler und so ...

Max Breitling

22

Techniken der Hardwarefehler-Toleranz

Selbstprüfende Paare



Fehler und so ...

Max Breitling

23

Unterschied Hardware/Software

Charakteristika von Fehlern in

Hardware:

- meist temporäre, sporadische, unabhängige Fehler
- Alterungs-, Verschleisserscheinungen
- Replikation kann sinnvoll sein

Software:

- nur permanente Fehler möglich
- keine Effekte von Alterung, Verschleiss
- identische Replikation sinnlos

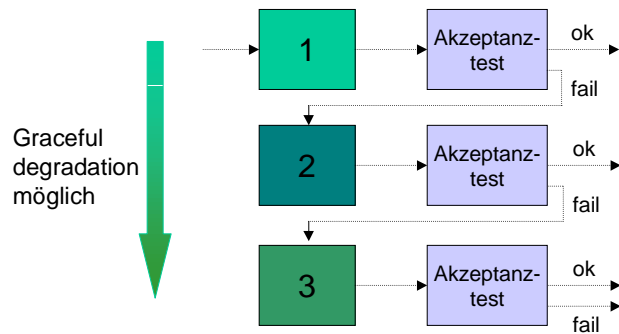
Fehler und so ...

Max Breitling

24

Techniken der Softwarefehler-Toleranz

Recovery Blocks

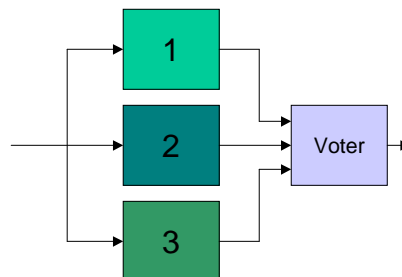


Voraussetzung:

- Existenz eines Akzeptanztests (zu komplex?)
- Möglichkeit zum Rücksetzen

Techniken der Softwarefehler-Toleranz

N-Versionen Programmierung



Voraussetzung:

- Existenz eines Voters (mehrere Ergebnisse?)
- Möglichkeit zur effizienten, nebenläufigen Abarbeitung

Techniken der Softwarefehler-Toleranz

Diskussionspunkte:

- Sicherstellung „echter“ Diversifikation?
- Erhöhter Aufwand zur Erstellung mehrerer Versionen?
- Ersatz für Techniken der Fehlervermeidung?
- Eingeständnis schlechter Systementwicklung?

Wenig Erfahrungsberichte:

- **Recovery Blocks**: Naval C&C System: 165 von 222 Fehler maskiert, 48 Fehler in Recovery, 60% zusätzlicher Entwicklungsaufwand, 40% längere Laufzeit.
- **3-Version-Programmierung**: 27 Programme zur Mustererkennung, Reduktion Fehlerwahrscheinlichkeit um Faktor 20 reduziert, *aber*: Fehler sind *nicht* statistisch unabhängig!

Beispiel für fehlertolerantes System

Space Shuttle

- Kritische Funktionen gesteuert von 5 baugleichen Rechnern
- Davon 4 mit identischer Software, angeordnet in 4-MR, Hardware-Voting an allen Aktuatoren, peer-Vergleiche zwischen Rechnern
- Bei Versagen eines Rechners per Voting Reduktion auf TMR
- Bei weiterem Versagen Reduktion auf *self-checking pair*
- Fünfter Rechner mit alternativer Software, manuell aktiviert

- Hauptsächlich Schutz vor Hardwareausfällen, nur eingeschränkter Schutz vor Designfehler in Software, kein Schutz vor Designfehlern in Hardware.

Zusammenfassung

- Fehler im allgemeinen sehr **vager** Begriff
- Fehler und Fehlertoleranz sind **relative** Begriffe
- Umgang erfordert **Identifikation** und **Beschreibung** von Fehlern und von akzeptablen Modifikationen
- Fehler erhöhen die **Systemkomplexität** deutlich
- **Systematischer** Umgang wünschenswert
Ideal: Trennung Normalverhalten und Fehlerbehandlung
- Fehlertoleranz darf nur **zusätzliches** Verfahren sein, aber andere Verfahren zur Konstruktion korrekter Systeme nicht ersetzen.

Literatur

Überblicksartikel

- Gärtner: Fundamentals of fault-tolerant distributed computing in asynchronous environments, ACM Computing Surveys, 31(1):1-26, 1999
- Rushby: Critical System Properties: Survey and taxonomy. Reliability Engineering and System Safety, 42(2):189-219, 1994

Bücher

- Lee/Anderson: Fault Tolerance - Principles and Practice, Springer 1990
- Laprie: Dependability: Basic Concepts and Terminology, Springer 1992
- Leveson: Safeware - System Safety and Computers, Addison-Wesley 1995
- Rust: Zuverlässigkeit und Verantwortung, Vieweg 1994

Formale Ansätze

- Liu/Joseph: A formal framework for fault-tolerant programs, IMA Conference on Mathematics of Dependable Systems, Oxford Univ. Press, 1993
- Janowski: On bisimulation, fault-monotonicity and provable fault-tolerance, AMAST, LNCS 1349, Springer 1997
- Arora/Kulkarni: Detectors and correctors: A theory of fault-tolerance components, International Conference on Distributed Computing Systems, 1998

Abstract

Der Begriff des Fehlers ist im allgemeinen sehr vielfältig interpretierbar: Fehler können auftreten in der Spezifikation, im Design, in der Implementierung oder während der Ausführung eines Systems. Fehler will man aber eigentlich vermeiden, oder wenigstens erkennen und/oder ihre Auswirkungen tolerieren können. Der Umgang mit Fehlern im Rahmen einer Systementwicklung ist dennoch kaum systematisiert.

Ich möchte im Vortrag einen Überblick geben über den Fehlerbegriff mit zugehörigen Klassifikationen, wie er sich in der Literatur einigermaßen etabliert hat. Ich werde den Begriff der Fehlertoleranz diskutieren mit ihren prinzipiellen Techniken sowohl für Hardware- als auch Softwarefehler, und dabei auch formale Ansätze ansprechen.