

# Verlässliche Echtzeitsysteme

## Grundlagen

### **Peter Ulbrich**

Lehrstuhl für Verteilte Systeme und Betriebssysteme  
Friedrich-Alexander-Universität Erlangen-Nürnberg

<https://www4.cs.fau.de>

19. April 2018





Wir kümmern uns ausschließlich um Fehler

→ Das ist nur ein kleiner Aspekt zuverlässiger Systeme!





Wir kümmern uns ausschließlich um **Fehler**



→ Das ist nur ein kleiner Aspekt **zuverlässiger Systeme!**



Fehler stellen jedoch einen **sehr komplexen Aspekt** dar

- Was genau ist ein Fehler überhaupt?
- Beeinflusst jeder Fehler das Verhalten eines Systems?



-  Wir kümmern uns ausschließlich um **Fehler**
  - Das ist nur ein kleiner Aspekt **zuverlässiger Systeme!**
  
-  Fehler stellen jedoch einen **sehr komplexen Aspekt** dar
  - **Was genau ist ein Fehler überhaupt?**
  - **Beeinflusst jeder Fehler das Verhalten eines Systems?**
  
- **Wie schwerwiegend ist ein Fehler?**
  - **Welchen Schaden kann ein Fehler verursachen?**



- ⚠ Wir kümmern uns ausschließlich um **Fehler**
  - Das ist nur ein kleiner Aspekt **zuverlässiger Systeme!**
- ☞ Fehler stellen jedoch einen **sehr komplexen Aspekt** dar
  - Was genau ist ein Fehler überhaupt?
  - Beeinflusst jeder Fehler das Verhalten eines Systems?
- Wie **schwerwiegend** ist ein Fehler?
  - Welchen **Schaden** kann ein Fehler verursachen?
- Wo entstehen Fehler und was beeinflussen sie?
  - **Software-** vs. **Hardware-Fehler**
  - Grundlegende **Klassifikation**, **Ursachen** und **Entstehung**





Wir kümmern uns ausschließlich um **Fehler**

→ Das ist nur ein kleiner Aspekt **zuverlässiger Systeme!**



Fehler stellen jedoch einen **sehr komplexen Aspekt** dar

- Was genau ist ein Fehler überhaupt?
- Beeinflusst jeder Fehler das Verhalten eines Systems?

■ Wie **schwerwiegend** ist ein Fehler?

- Welchen **Schaden** kann ein Fehler verursachen?

■ Wo entstehen Fehler und was beeinflussen sie?

- **Software-** vs. **Hardware-Fehler**
- Grundlegende **Klassifikation**, **Ursachen** und **Entstehung**

■ Was bedeutet es, mit Fehlern umgehen zu können?

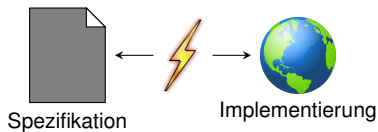
- Worin unterscheiden sich etwa **Zuverlässigkeit** und **Verfügbarkeit**?



- 1 Fehler
- 2 Verlässlichkeitsmodelle
- 3 Fehler und Systementwurf
- 4 Software- und Hardwarefehler
- 5 Zusammenfassung



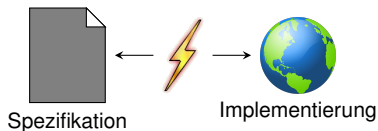
# Definition: Fehler



Gemäß DIN EN ISO 8402:1995-08 [4] ist ein **Fehler** die „Nichterfüllung einer festgelegten Forderung“



# Definition: Fehler

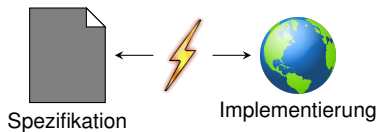


☞ Gemäß DIN EN ISO 8402:1995-08 [4] ist ein **Fehler** die „Nichterfüllung einer festgelegten Forderung“

- Fehler kennen demzufolge viele Ausprägungen
  - Sie können lediglich als **störend** empfunden werden
    - Sie eigentliche Funktion ist noch vorhanden, es geht aber Komfort verloren
  - Sie können die Funktionalität **beeinträchtigen**
    - Das Abspielen eines Videos „ruckelt“, die Bildrate wird nicht erreicht
  - Sie können aber auch zum vollständigen **Systemversagen** führen
    - Eine fehlerhafte Fluglageregelung kann den I4Copter abstürzen lassen



# Definition: Fehler



☞ Gemäß DIN EN ISO 8402:1995-08 [4] ist ein **Fehler** die „Nichterfüllung einer festgelegten Forderung“

- Fehler kennen demzufolge viele Ausprägungen
  - Sie können lediglich als **störend** empfunden werden
    - Sie eigentliche Funktion ist noch vorhanden, es geht aber Komfort verloren
  - Sie können die Funktionalität **beeinträchtigen**
    - Das Abspielen eines Videos „ruckelt“, die Bildrate wird nicht erreicht
  - Sie können aber auch zum vollständigen **Systemversagen** führen
    - Eine fehlerhafte Fluglageregelung kann den I4Copter abstürzen lassen

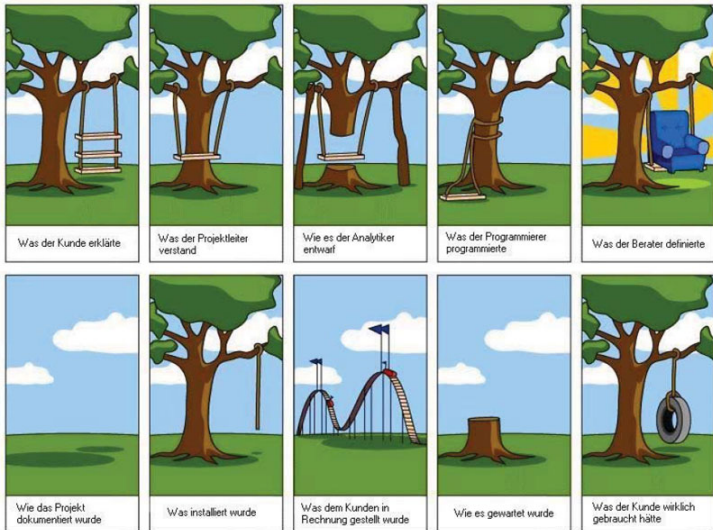
⚠ **Wichtig:** Bezugspunkt ist die Spezifikation  $\rightsquigarrow$  **Verifikation**

→ Haben wir das System korrekt implementiert?



# Abgrenzung: Keine Validierung

Haben wir das korrekte System implementiert?



# Wann ist ein Fehler nun ein Fehler?

---

```
int regelschritt() {  
    int sensorwert = 0;  
    sensorwert = leseSensor();  
  
    int stellwert =  
        regler(sensorwert);  
    return stellwert;  
}
```



# Wann ist ein Fehler nun ein Fehler?

(gutartiger) Defekt ⚡  
verfälscht sensorwert

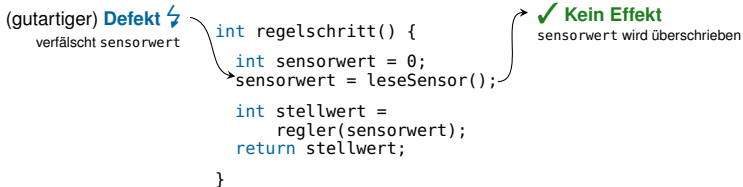
```
int regelschritt() {  
    int sensorwert = 0;  
    sensorwert = leseSensor();  
    int stellwert =  
        regler(sensorwert);  
    return stellwert;  
}
```

## 1 Defekte (engl. *faults*) sind die Quelle allen Übels

- Ursachen: Software-Bugs, Produktionsfehler, äußere Einflüsse, ...
- Beziehen sich auf **Strukturelemente** (Daten, Anweisungen, ...)



# Wann ist ein Fehler nun ein Fehler?

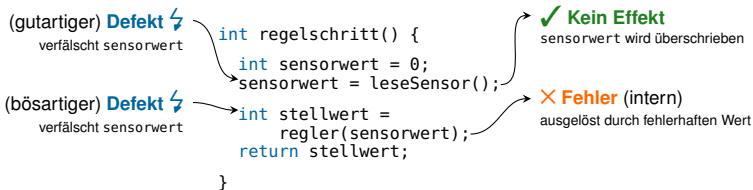


## 1 Defekte (engl. *faults*) sind die Quelle allen Übels

- Ursachen: Software-Bugs, Produktionsfehler, äußere Einflüsse, ...
- Beziehen sich auf **Strukturelemente** (Daten, Anweisungen, ...)
- **Gutartige Defekte** (engl. *benign faults*) führen *nicht* zu einem Fehler



# Wann ist ein Fehler nun ein Fehler?



## 1 Defekte (engl. *faults*) sind die Quelle allen Übels

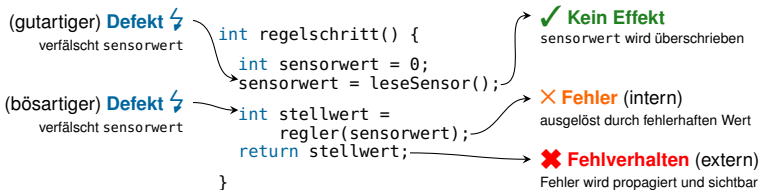
- Ursachen: Software-Bugs, Produktionsfehler, äußere Einflüsse, ...
- Beziehen sich auf **Strukturelemente** (Daten, Anweisungen, ...)
- **Gutartige Defekte** (engl. *benign faults*) führen *nicht* zu einem Fehler

## 2 Manifestation eines Defekts ist ein **innerer Fehler** (engl. *error*)

- Defekt ist also **bösaertig** (engl. *malign fault*)
- Fehler beziehen sich auf den **nicht sichtbaren, inneren Zustand**



# Wann ist ein Fehler nun ein Fehler?



## 1 Defekte (engl. *faults*) sind die Quelle allen Übels

- Ursachen: Software-Bugs, Produktionsfehler, äußere Einflüsse, ...
- Beziehen sich auf **Strukturelemente** (Daten, Anweisungen, ...)
- **Gutartige Defekte** (engl. *benign faults*) führen *nicht* zu einem Fehler

## 2 Manifestation eines Defekts ist ein **innerer Fehler** (engl. *error*)

→ Defekt ist also **bösartig** (engl. *malign fault*)

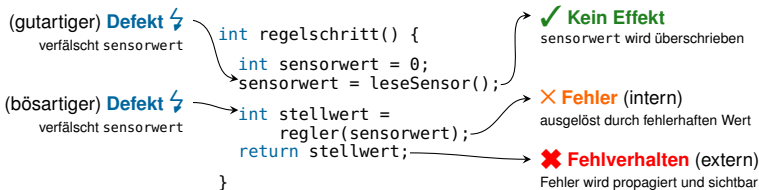
- Fehler beziehen sich auf den **nicht sichtbaren, inneren Zustand**

## 3 Sichtbarkeit des Fehlers führt zum **Fehlverhalten** (engl. *failure*)

- Beziehen sich auf das **beobachtbare Verhalten**



# Wann ist ein Fehler nun ein Fehler?



## 1 Defekte (engl. *faults*) sind die Quelle allen Übels

- Ursachen: Software-Bugs, Produktionsfehler, äußere Einflüsse, ...
- Beziehen sich auf **Strukturelemente** (Daten, Anweisungen, ...)
- **Gutartige Defekte** (engl. *benign faults*) führen *nicht* zu einem Fehler

## 2 Manifestation eines Defekts ist ein **innerer Fehler** (engl. *error*)

→ Defekt ist also **bösartig** (engl. *malign fault*)

- Fehler beziehen sich auf den **nicht sichtbaren, inneren Zustand**

## 3 Sichtbarkeit des Fehlers führt zum **Fehlverhalten** (engl. *failure*)

- Beziehen sich auf das **beobachtbare Verhalten**

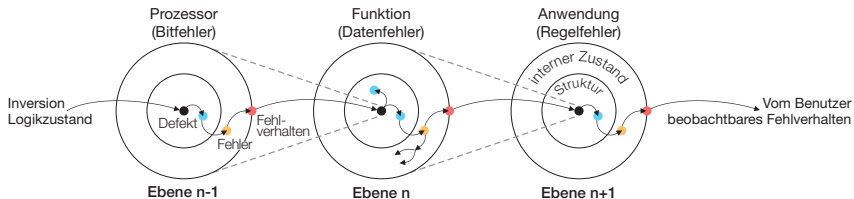


Die sogenannte **Fehlerkette** [12, Kapitel 1] (**fault**  $\rightsquigarrow$  **error**  $\rightsquigarrow$  **failure**)



# Fehlerkette: Sichtbarkeit und Betrachtungsebene

Es ist alles eine Frage der Sichtbarkeit



Über die Fehlerkette pflanzen sich Fehler im System fort

- Gutartige Defekte haben keinen Einfluss
- Bösartige Defekte/innere Fehler beeinflussen den internen Zustand
  - Bezogen auf die aktuelle Betrachtungsebene
  - Intern kann sich der Fehler zunächst unbemerkt weiter verbreiten
- Äußere Fehler bezeichnen ein Fehlverhalten der aktuellen Ebene



Fehlerausbreitung kann schließlich zum beobachtbaren, **vollständigen Systemversagen** führen





Fehler müssen nicht immer auftreten ...

**Permanente Fehler** (engl. *permanent fault/error/failure*)

- Bestehen eine **unbegrenzt lange Zeitdauer**
- Bis sie durch eine **korrigierende Maßnahme** behoben werden

**Sporadische Fehler** (engl. *intermittent fault/error/failure*)

- Treten **unregelmäßig** auf, häufen sich aber in vielen Fällen und ...
- sind oft **Vorboten drohender, permanenter Fehler**

**Transiente Fehler** (engl. *transient fault/error/failure*)

- Treten wie sporadische Fehler **unregelmäßig** auf ...
- Münden i. d. R. aber nicht in einem permanenten Fehler





Fehler müssen nicht immer auftreten ...

**Permanente Fehler** (engl. *permanent fault/error/failure*)

- Bestehen eine **unbegrenzt lange Zeitdauer**
- Bis sie durch eine **korrigierende Maßnahme** behoben werden

**Sporadische Fehler** (engl. *intermittent fault/error/failure*)

- Treten **unregelmäßig** auf, häufen sich aber in vielen Fällen und ...
- sind oft **Vorboten drohender, permanenter Fehler**

**Transiente Fehler** (engl. *transient fault/error/failure*)

- Treten wie sporadische Fehler **unregelmäßig** auf ...
- Münden i. d. R. aber nicht in einem permanenten Fehler



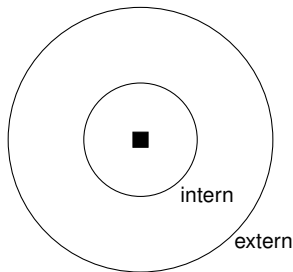
Implikationen aus der Fehlerkette:

- Normalerweise: transiente Defekte  $\nearrow$  permanentes Fehlverhalten
- Möglich: permanenter Defekt  $\rightsquigarrow$  transientes Fehlverhalten
  - Falls sie nur unregelmäßig den inneren Sichtbarkeitsbereich verlassen



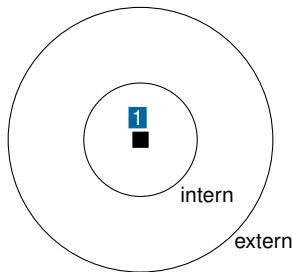
# Maßnahmen zum Umgang mit Fehlern

Versuchen die Fehlerkette aufzubrechen



**1** **Vorbeugung** – versucht die Entstehung von Defekten in der Produktion zu verhindern

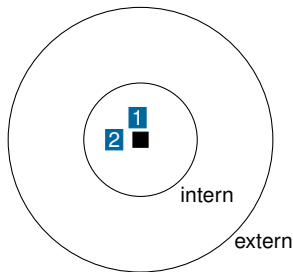
- z.B. durch Entwicklungsmethoden



# Maßnahmen zum Umgang mit Fehlern

Versuchen die Fehlerkette aufzubrechen

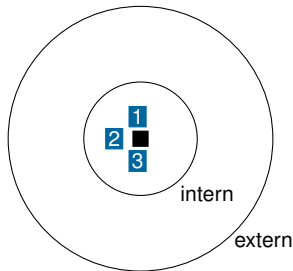
- 1** **Vorbeugung** – versucht die Entstehung von Defekten in der Produktion zu verhindern
  - z.B. durch Entwicklungsmethoden
- 2** **Entfernung** – vor der Auslieferung oder im Zuge einer planmäßigen Wartung
  - Erfordert die Erkennung von Defekten → **Qualitätssicherung**



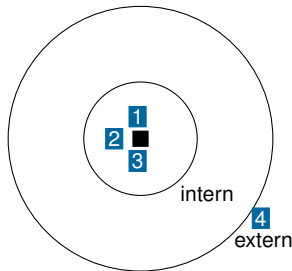
# Maßnahmen zum Umgang mit Fehlern

Versuchen die Fehlerkette aufzubrechen

- 1** **Vorbeugung** – versucht die Entstehung von Defekten in der Produktion zu verhindern
  - z.B. durch Entwicklungsmethoden
- 2** **Entfernung** – vor der Auslieferung oder im Zuge einer planmäßigen Wartung
  - Erfordert die Erkennung von Defekten → **Qualitätssicherung**
- 3** **Vorhersage** – Wo treten evtl. Defekte auf?
  - Ermöglicht die Entfernung



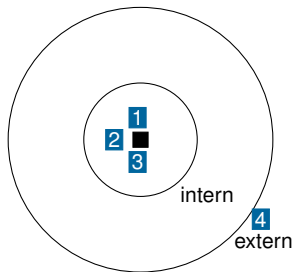
- 1 Vorbeugung** – versucht die Entstehung von Defekten in der Produktion zu verhindern
  - z.B. durch Entwicklungsmethoden
- 2 Entfernung** – vor der Auslieferung oder im Zuge einer planmäßigen Wartung
  - Erfordert die Erkennung von Defekten → Qualitätssicherung
- 3 Vorhersage** – Wo treten evtl. Defekte auf?
  - Ermöglicht die Entfernung
- 4 Toleranz** – verhindert nicht den Defekt, aber die Fortpflanzung zum Fehlverhalten
  - z.B. durch Maskierung **innerer Fehler**



# Maßnahmen zum Umgang mit Fehlern

Versuchen die Fehlerkette aufzubrechen

- 1 Vorbeugung** – versucht die Entstehung von Defekten in der Produktion zu verhindern
  - z.B. durch Entwicklungsmethoden
- 2 Entfernung** – vor der Auslieferung oder im Zuge einer planmäßigen Wartung
  - Erfordert die Erkennung von Defekten → Qualitätssicherung
- 3 Vorhersage** – Wo treten evtl. Defekte auf?
  - Ermöglicht die Entfernung
- 4 Toleranz** – verhindert nicht den Defekt, aber die Fortpflanzung zum Fehlverhalten
  - z.B. durch Maskierung **innerer Fehler**



**Ziel:** Reduktion des vom Benutzer beobachtbaren Fehlverhaltens



 Nicht jedes beobachtbare Fehlverhalten wird entdeckt:

- ☞ Nicht jedes beobachtbare Fehlverhalten wird entdeckt:
- **Unerkannte Datenfehler** (engl. *silent data corruption, SDC*)
  - **Unbemerkte Fehlerfortpflanzung** innerhalb oder außerhalb
    - Fehlerhafte Berechnungsergebnisse oder Ausgabewerte
  - ⚠ **Sehr, sehr schwer ausfindig zu machen**
    - Zusammenhang Ursache ↔ Fehlverhalten nicht erkennbar

☞ Nicht jedes beobachtbare Fehlverhalten wird entdeckt:

## ■ Unerkannte Datenfehler (engl. *silent data corruption, SDC*)

- **Unbemerkte Fehlerfortpflanzung** innerhalb oder außerhalb
  - Fehlerhafte Berechnungsergebnisse oder Ausgabewerte

⚠ **Sehr, sehr schwer ausfindig zu machen**

- Zusammenhang Ursache ↔ Fehlverhalten nicht erkennbar

- **Fehlererkennung** verhindert die unbemerkte Fehlerausbreitung
  - Überführt unerkannte Datenfehler in erkannte Datenfehler

## ■ Erkannte, nicht korrigierbare Fehler (engl. *detected unrecoverable error, DUE*)

- Eine Fortpflanzung kann gezielt unterbunden werden
- Fehlerstellen lassen sich vergleichsweise einfach herausfinden, z. B. durch eine **Ablaufverfolgung** (engl. *backtrace*)

→ Eine Fortführung der Funktion ist jedoch nicht möglich  $\rightsquigarrow$  **fail-stop**



1 Fehler

**2** Verlässlichkeitsmodelle

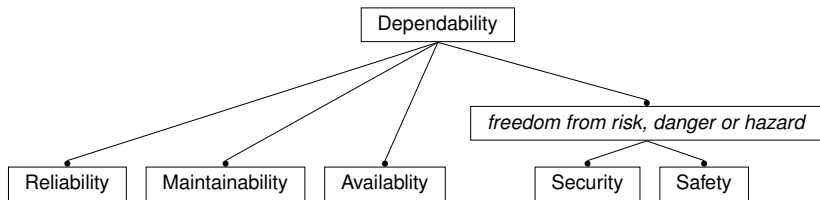
3 Fehler und Systementwurf

4 Software- und Hardwarefehler

5 Zusammenfassung



# „Verlässlichkeit“ ist ein vielschichtiger Begriff



*The trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers. [6]*





$R(t)$ : Wahrscheinlichkeit, dass ein System seinen Dienst bis zum Zeitpunkt  $t$  leisten wird, sofern es bei  $t_0$  betriebsbereit war

- Annahme: eine **konstante Fehlerrate** von  $\lambda$  Fehler/Stunde
- Zuverlässigkeit zum Zeitpunkt  $t$ :  $R(t) = \exp(-\lambda(t - t_0))$ 
  - mit  $t - t_0$  gegeben in Stunden
- Inverse  $1/\lambda$  ist die (engl. *mean time to failure*) (MTTF)





$R(t)$ : Wahrscheinlichkeit, dass ein System seinen Dienst bis zum Zeitpunkt  $t$  leisten wird, sofern es bei  $t_0$  betriebsbereit war

- Annahme: eine **konstante Fehlerrate** von  $\lambda$  Fehler/Stunde
- Zuverlässigkeit zum Zeitpunkt  $t$ :  $R(t) = \exp(-\lambda(t - t_0))$ 
  - mit  $t - t_0$  gegeben in Stunden
- Inverse  $1/\lambda$  ist die (engl. *mean time to failure*) (MTTF)

## ■ Ultra-hohe Zuverlässigkeit $\mapsto \lambda \leq 10^{-9}$ Fehler/Stunde

- Beispiel: elektronisch gesteuerte Bremsanlage im Automobil
  - Kfz sei durchschnittlich eine Stunde täglich in Betrieb
  - > Jährlich nur ein Fehler pro eine Million Kfz
- Beispiele: Eisenbahnsignalanlagen, Kernkraftwerküberwachung





$M(d)$ : Wahrscheinlichkeit, dass das System innerhalb Zeitspanne  $d$  wieder hergestellt ist

- Ansatz: **konstante Reparaturrate** von  $\mu$  Reparaturen/Stunde
- Die Inverse  $1/\mu$  ist dann die *mean time to repair* (MTTR)





$M(d)$ : Wahrscheinlichkeit, dass das System innerhalb Zeitspanne  $d$  wieder hergestellt ist

- Ansatz: **konstante Reparaturrate** von  $\mu$  Reparaturen/Stunde
- Die Inverse  $1/\mu$  ist dann die *mean time to repair* (MTTR)

## ■ **Fundamentaler Konflikt** zwischen Zuverlässigkeit und Wartbarkeit

- Ein wartbares System erfordert einen modularen Aufbau
  - Kleinste ersetzbare Einheit (engl. *smallest replaceable unit*, SDU)
  - Über Steckverbindungen lose gekoppelt mit anderen SDUs
  - Dadurch ist jedoch eine höhere (physikalische) Fehlerrate gegeben
  - Darüberhinaus verbuchen sich höhere Herstellungskosten
- Ein zuverlässiges System ist **aus einem Guss** gefertigt. . .
- Unterschiede zur Softwareentwicklung





$M(d)$ : Wahrscheinlichkeit, dass das System innerhalb Zeitspanne  $d$  wieder hergestellt ist

- Ansatz: **konstante Reparaturrate** von  $\mu$  Reparaturen/Stunde
- Die Inverse  $1/\mu$  ist dann die *mean time to repair* (MTTR)

## ■ **Fundamentaler Konflikt** zwischen Zuverlässigkeit und Wartbarkeit

- Ein wartbares System erfordert einen modularen Aufbau
  - Kleinste ersetzbare Einheit (engl. *smallest replaceable unit*, SDU)
  - Über Steckverbindungen lose gekoppelt mit anderen SDUs
  - Dadurch ist jedoch eine höhere (physikalische) Fehlerrate gegeben
  - Darüberhinaus verbuchen sich höhere Herstellungskosten
- Ein zuverlässiges System ist **aus einem Guss** gefertigt. . .
- Unterschiede zur Softwareentwicklung



Beim Entwurf von Produkten für den Massenmarkt geht die Zuverlässigkeit meist auf Kosten von Wartbarkeit





Maß zur Bereitstellung einer Funktion vor dem Hintergrund eines abwechselnd korrekt und fehlerhaft arbeitenden Systems

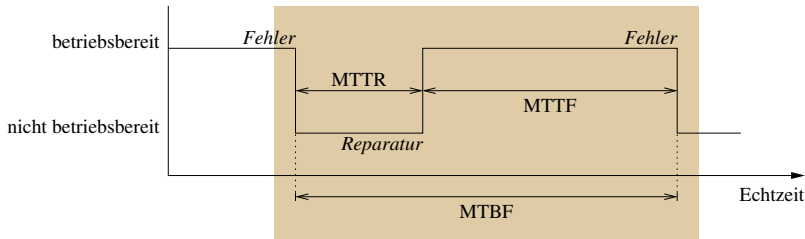
- Zeitanteil der **Betriebsbereitschaft**:  $A = \frac{MTTF}{(MTTF + MTTR)}$
- $MTTF + MTTR$ : *mean time between failures* (MTBF)





Maß zur Bereitstellung einer Funktion vor dem Hintergrund eines abwechselnd korrekt und fehlerhaft arbeitenden Systems

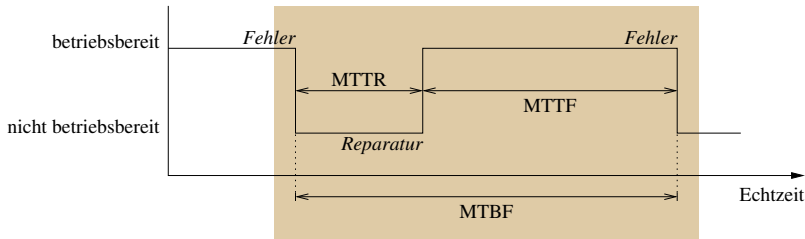
- Zeitanteil der **Betriebsbereitschaft**:  $A = \text{MTTF} / (\text{MTTF} + \text{MTTR})$
- $\text{MTTF} + \text{MTTR}$ : *mean time between failures* (MTBF)





Maß zur Bereitstellung einer Funktion vor dem Hintergrund eines abwechselnd korrekt und fehlerhaft arbeitenden Systems

- Zeitanteil der **Betriebsbereitschaft**:  $A = \text{MTTF} / (\text{MTTF} + \text{MTTR})$
- $\text{MTTF} + \text{MTTR}$ : *mean time between failures* (MTBF)



Hohe Verfügbarkeit bedeutet kurze MTTR und/oder lange MTTF



# Sicherheit $\mapsto$ *Security* und *Safety*

Robustheit des Echtzeitsystems stärken



*Security*: Schutz von Informationen und Informationsverarbeitung vor „intelligenten“ Angreifern

- Allgemein in Bezug auf Datenbasen
  - Vertraulichkeit (engl. *confidentiality*)
  - Datenschutz (engl. *privacy*)
  - Glaubwürdigkeit (engl. *authenticity*)
- Speziell z.B. Diebstahlsicherung
  - Kryptographie (engl. *cryptography*)



*Security*: Schutz von Informationen und Informationsverarbeitung vor „intelligenten“ Angreifern

- Allgemein in Bezug auf Datenbasen
  - Vertraulichkeit (engl. *confidentiality*)
  - Datenschutz (engl. *privacy*)
  - Glaubwürdigkeit (engl. *authenticity*)
- Speziell z.B. Diebstahlsicherung
  - Kryptographie (engl. *cryptography*)

*Safety*: Schutz von Menschen und Sachwerten vor dem Versagen technischer Systeme

- Zuverlässigkeit trotz Fehlverhaltens
  - Kosten liegen um Größenordnungen über dem Normalbetrieb
- Abgrenzung von unkritischen, gutartigen Fehlern
- Oft ist Zertifizierung (engl. *certification*) erforderlich



Automobil eine Bestandsaufnahme vom Jahr 2005 ...



**Automobil** eine Bestandsaufnahme vom Jahr 2005 ...

- Etwa 90 % der Innovationen im Auto bringt die Elektronik ein
  - Gut 80 % davon sind Software



**Automobil** eine Bestandsaufnahme vom Jahr 2005 ...

- Etwa 90 % der Innovationen im Auto bringt die Elektronik ein
  - Gut 80 % davon sind Software
- Etwa ein Drittel aller Pannen liegen an fehlerhafter Elektronik
  - Gut 80 % davon sind Softwarefehler



**Automobil** eine Bestandsaufnahme vom Jahr 2005 ...

- Etwa 90 % der Innovationen im Auto bringt die Elektronik ein
  - Gut 80 % davon sind Software
- Etwa ein Drittel aller Pannen liegen an fehlerhafter Elektronik
  - Gut 80 % davon sind Softwarefehler

*Everything should be made as simple as possible, but no simpler.  
(Albert Einstein)*

*Vollkommenheit entsteht offensichtlich nicht dann, wenn man nichts mehr hinzuzufügen hat, sondern wenn man nichts mehr wegnehmen kann. (Antoine de Saint Exupery)*



# Verlässlichkeit unterscheidet sich je nach System

Je nachdem, wie kritisch sich ein einzelner Fehler auswirkt.

**Hochverfügbare Systeme** z. B. Telekommunikationstechnik

- Müssen ihren Dienst möglichst ununterbrochen verrichten
  - Einzelne Fehler sind jedoch verkraftbar ( $\leadsto$  **fail-soft**)
    - Sie werden meist auf höheren Ebenen abgefangen (z. B. TCP/IP)
- **Kurze Fehlererholung** steht im Vordergrund



# Verlässlichkeit unterscheidet sich je nach System

Je nachdem, wie kritisch sich ein einzelner Fehler auswirkt.

## Hochverfügbare Systeme z. B. Telekommunikationstechnik

- Müssen ihren Dienst möglichst ununterbrochen verrichten
  - Einzelne Fehler sind jedoch verkraftbar ( $\leadsto$  fail-soft)
    - Sie werden meist auf höheren Ebenen abgefangen (z. B. TCP/IP)
- Kurze Fehlererholung steht im Vordergrund

## Langlebige Systeme z. B. Satelliten

- Müssen auch nach Jahren noch funktionieren ( $\leadsto$  fail-slow)
  - Eine Fehlerbehebung ist oft technisch nicht möglich
- Hohe Zuverlässigkeit steht im Vordergrund



# Verlässlichkeit unterscheidet sich je nach System

Je nachdem, wie kritisch sich ein einzelner Fehler auswirkt.

## Hochverfügbare Systeme z. B. Telekommunikationstechnik

- Müssen ihren Dienst möglichst ununterbrochen verrichten
  - Einzelne Fehler sind jedoch verkraftbar ( $\leadsto$  fail-soft)
    - Sie werden meist auf höheren Ebenen abgefangen (z. B. TCP/IP)
- Kurze Fehlererholung steht im Vordergrund

## Langlebige Systeme z. B. Satelliten

- Müssen auch nach Jahren noch funktionieren ( $\leadsto$  fail-slow)
  - Eine Fehlerbehebung ist oft technisch nicht möglich
- Hohe Zuverlässigkeit steht im Vordergrund

## Sicherheitskritische Systeme z. B. Flugzeuge, Kernkraftwerke, Eisenbahn, Industrieanlagen, Medizintechnik . . .

- Zuverlässig und ununterbrochene Funktion ( $\leadsto$  fail-safe)
  - Diese Anlagen sind nur sinnvoll, wenn sie im Betrieb sind!
- Hohe Ansprüche an Zuverlässigkeit und Verfügbarkeit



- 1 Fehler
- 2 Verlässlichkeitsmodelle
- 3 Fehler und Systementwurf**
- 4 Software- und Hardwarefehler
- 5 Zusammenfassung



- Klärung durch eine **Gefahrenanalyse und Risikobeurteilung**

- Klärung durch eine **Gefahrenanalyse und Risikobeurteilung**
  - **Identifikation** gefährlicher Ereignisse und
  - ihre **Klassifikation** hinsichtlich verschiedener Kriterien
- **Faustregel:** Risiko = Wahrscheinlichkeit  $\times$  Schweregrad
  - **Wahrscheinlichkeit:** Auftretenswahrscheinlichkeit eines Ereignisses
  - der Schweregrad bemisst sich häufig als „Konsequenz / Ereignis“
  - Risiko  $\approx$  Wahrscheinlichkeit der Konsequenz
    - der entstehende **finanzielle Schaden** ist oft ein Maß für die Konsequenz

- Klärung durch eine **Gefahrenanalyse und Risikobeurteilung**
  - **Identifikation** gefährlicher Ereignisse und
  - ihre **Klassifikation** hinsichtlich verschiedener Kriterien
- **Faustregel:** Risiko = Wahrscheinlichkeit × Schweregrad
  - **Wahrscheinlichkeit:** Auftretenswahrscheinlichkeit eines Ereignisses
  - der Schweregrad bemisst sich häufig als „Konsequenz / Ereignis“
  - Risiko  $\approx$  Wahrscheinlichkeit der Konsequenz
    - der entstehende **finanzielle Schaden** ist oft ein Maß für die Konsequenz
- Normen reglementieren die Klassifikation, z. B. ISO 26262 [7]

# Schweregrad des Fehlverhaltens

- Klärung durch eine **Gefahrenanalyse und Risikobeurteilung**
  - **Identifikation** gefährlicher Ereignisse und
  - ihre **Klassifikation** hinsichtlich verschiedener Kriterien
- **Faustregel:** Risiko = Wahrscheinlichkeit  $\times$  Schweregrad
  - **Wahrscheinlichkeit:** Auftretenswahrscheinlichkeit eines Ereignisses
  - der Schweregrad bemisst sich häufig als „Konsequenz / Ereignis“
  - Risiko  $\approx$  Wahrscheinlichkeit der Konsequenz
    - der entstehende **finanzielle Schaden** ist oft ein Maß für die Konsequenz
- Normen reglementieren die Klassifikation, z. B. ISO 26262 [7]

## ■ Kriterien:

## Schweregrade nach ISO 26262:

■ Schweregrad	<b>S0</b>	keine Verletzungen
■ Wahrscheinlichkeit	<b>S1</b>	leichte Verletzungen
■ Kontrollierbarkeit	<b>S2</b>	schwere o. lebensbedrohliche Verletzungen
	<b>S3</b>	lebensbedrohliche o. tödliche Verletzungen





Eine Fehlerbaumanalyse (engl. *fault-tree analysis*) [3, 8] ermittelt die zum beobachtbaren Systemverhalten führen Ereignisse:

- Verfeinernde Analyse (engl. *top-down analysis*)
  - Das unerwünschte Fehlverhalten bildet die Wurzel des Fehlerbaumes
  - Ausgehend davon werden die Ursachen des Fehlverhaltens identifiziert
- Arbeitet auf dem Fehlerraum (engl. *failure space*) des Systems
  - Zuverlässigkeitsblockdiagrammen (engl. *reliability block diagrams*)
  - Diese befassen sich mit dessen korrekter Funktion





Eine Fehlerbaumanalyse (engl. *fault-tree analysis*) [3, 8] ermittelt die zum beobachtbaren Systemverhalten führen Ereignisse:

- Verfeinernde Analyse (engl. *top-down analysis*)
  - Das unerwünschte Fehlverhalten bildet die Wurzel des Fehlerbaumes
  - Ausgehend davon werden die Ursachen des Fehlverhaltens identifiziert
- Arbeitet auf dem Fehlerraum (engl. *failure space*) des Systems
  - Zuverlässigkeitsblockdiagrammen (engl. *reliability block diagrams*)
  - Diese befassen sich mit dessen korrekter Funktion
- Beispiel: Reaktorkühlsystem eines Kernkraftwerks fällt aus
  - Das Kühlsystem leckt oder
  
  
  - Die Kühlmittelpumpe funktioniert nicht





Eine Fehlerbaumanalyse (engl. *fault-tree analysis*) [3, 8] ermittelt die zum beobachtbaren Systemverhalten führen Ereignisse:

- Verfeinernde Analyse (engl. *top-down analysis*)
  - Das unerwünschte Fehlverhalten bildet die Wurzel des Fehlerbaumes
  - Ausgehend davon werden die Ursachen des Fehlverhaltens identifiziert
- Arbeitet auf dem Fehlerraum (engl. *failure space*) des Systems
  - Zuverlässigkeitsblockdiagrammen (engl. *reliability block diagrams*)
  - Diese befassen sich mit dessen korrekter Funktion
- Beispiel: Reaktorkühlsystem eines Kernkraftwerks fällt aus
  - Das Kühlsystem leckt oder
    - Eine Dichtung ist defekt oder
    - Eine Rohrleitung hat einen Riss oder
    - Der Reaktordruckbehälter hat einen Riss
  - Die Kühlmittelpumpe funktioniert nicht



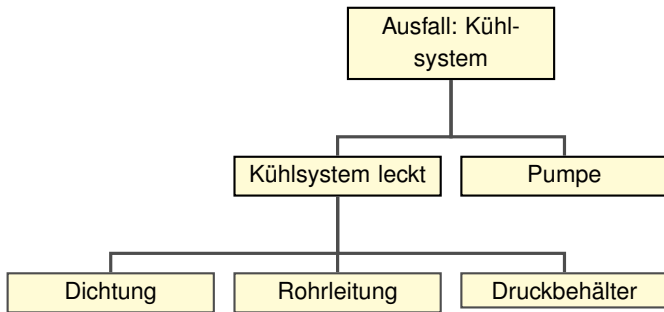


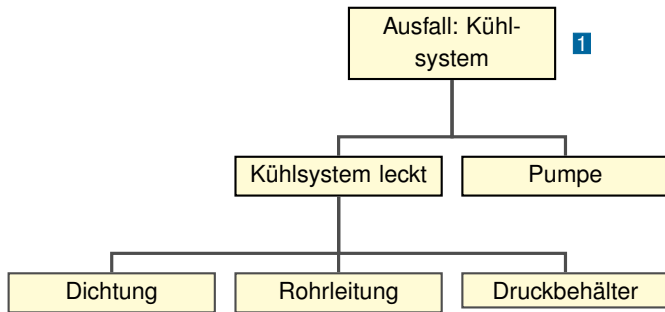
Eine Fehlerbaumanalyse (engl. *fault-tree analysis*) [3, 8] ermittelt die zum beobachtbaren Systemverhalten führen Ereignisse:

- Verfeinernde Analyse (engl. *top-down analysis*)
  - Das unerwünschte Fehlverhalten bildet die Wurzel des Fehlerbaumes
  - Ausgehend davon werden die Ursachen des Fehlverhaltens identifiziert
- Arbeitet auf dem Fehlerraum (engl. *failure space*) des Systems
  - Zuverlässigkeitsblockdiagrammen (engl. *reliability block diagrams*)
  - Diese befassen sich mit dessen korrekter Funktion
- Beispiel: Reaktorkühlsystem eines Kernkraftwerks fällt aus
  - Das Kühlsystem leckt oder
    - Eine Dichtung ist defekt oder
    - Eine Rohrleitung hat einen Riss oder
    - Der Reaktordruckbehälter hat einen Riss
  - Die Kühlmittelpumpe funktioniert nicht
    - Die Pumpe ist defekt oder
    - Die Energieversorgung ist ausgefallen



# Aufbau und Erstellung von Fehlerbäumen

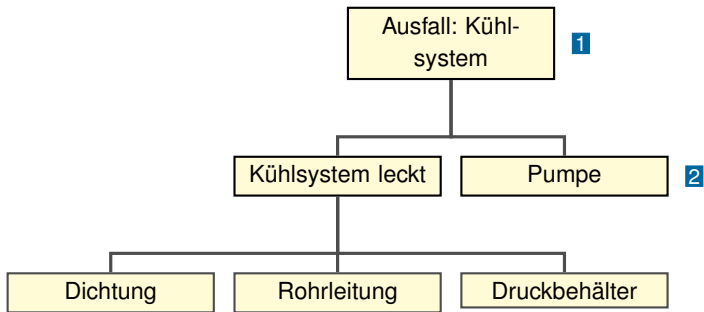




1 Schadensereignis



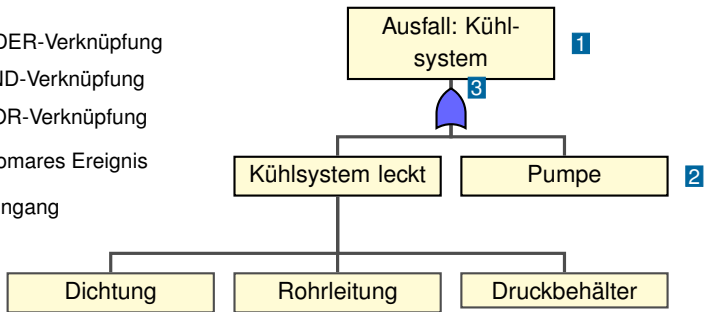
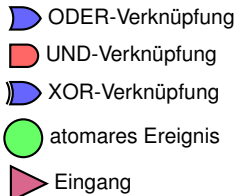
# Aufbau und Erstellung von Fehlerbäumen



- 1 Schadensereignis
- 2 Ereignisse auf Ebene 2



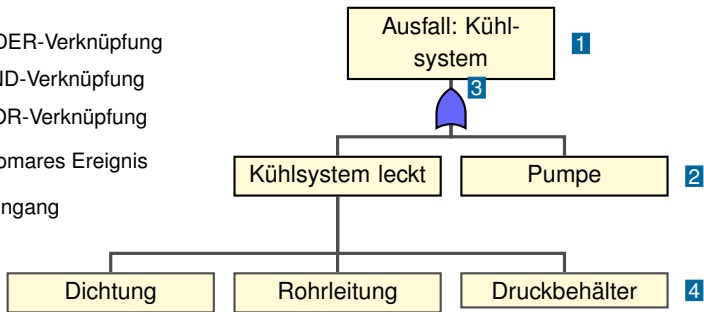
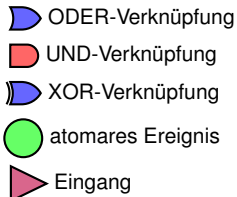
# Aufbau und Erstellung von Fehlerbäumen



- 1 Schadensereignis
- 2 Ereignisse auf Ebene 2
- 3 Logische Verknüpfung



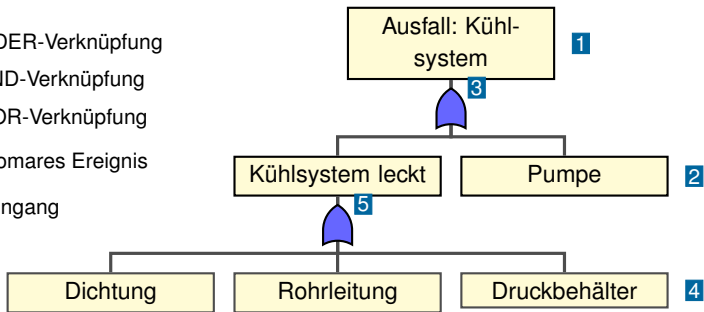
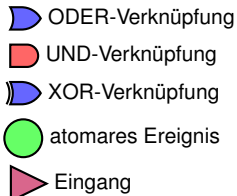
# Aufbau und Erstellung von Fehlerbäumen



- 1 Schadensereignis
- 2 Ereignisse auf Ebene 2
- 3 Logische Verknüpfung
- 4 Ereignisse auf Ebene 3



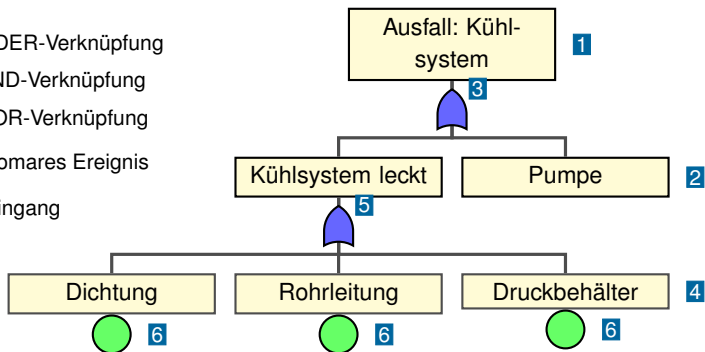
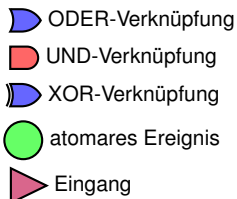
# Aufbau und Erstellung von Fehlerbäumen



- 1 Schadensereignis
- 2 Ereignisse auf Ebene 2
- 3 Logische Verknüpfung
- 4 Ereignisse auf Ebene 3
- 5 Logische Verknüpfung



# Aufbau und Erstellung von Fehlerbäumen

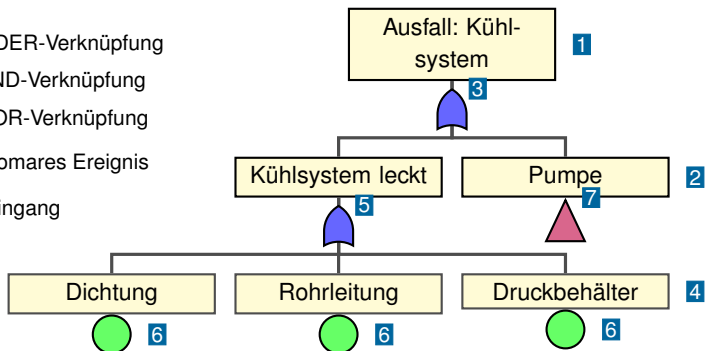
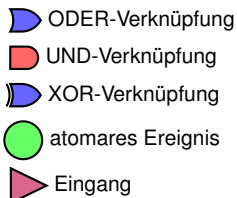


- 1 Schadensereignis
- 2 Ereignisse auf Ebene 2
- 3 Logische Verknüpfung
- 4 Ereignisse auf Ebene 3

- 5 Logische Verknüpfung
- 6 Atomare Ereignisse



# Aufbau und Erstellung von Fehlerbäumen



- |                                 |  |
|---------------------------------|--|
| <b>1</b> Schadensereignis       | <b>5</b> Logische Verknüpfung                              |
| <b>2</b> Ereignisse auf Ebene 2 | <b>6</b> Atomare Ereignisse                                |
| <b>3</b> Logische Verknüpfung   | <b>7</b> Eingänge zerlegen den Fehlerbaum → Neuer Teilbaum |
| <b>4</b> Ereignisse auf Ebene 3 |  |



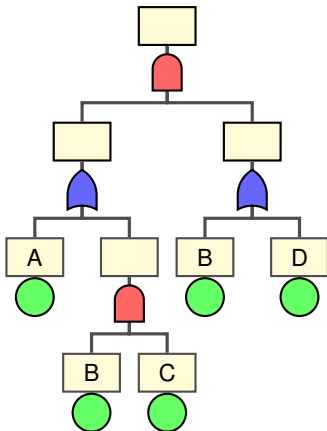


Ein **Schnitt** (engl. *cut-set*) enthält genau die atomaren Ereignisse, die das Schadensereignis verursachen:



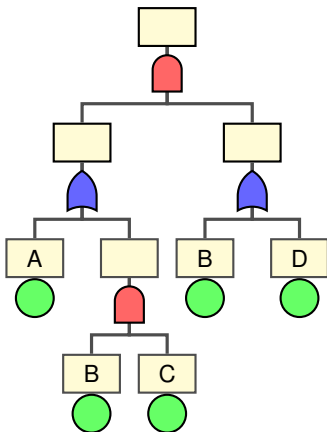


Ein **Schnitt** (engl. *cut-set*) enthält genau die atomaren Ereignisse, die das Schadensereignis verursachen:





Ein **Schnitt** (engl. *cut-set*) enthält genau die atomaren Ereignisse, die das Schadensereignis verursachen:

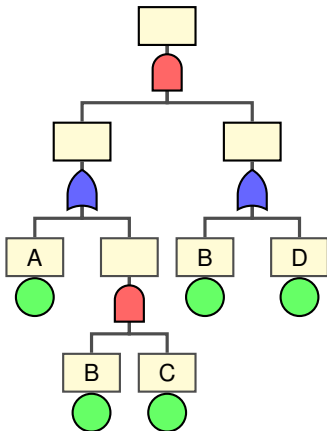


- $\{A, B, D\}$  wäre eine solche Menge





Ein **Schnitt** (engl. *cut-set*) enthält genau die atomaren Ereignisse, die das Schadensereignis verursachen:

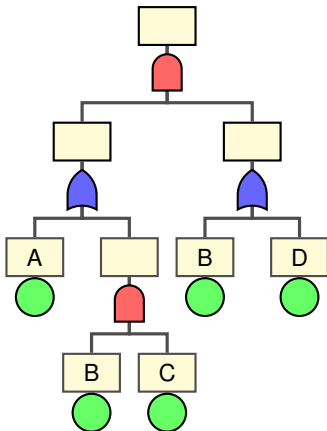


- $\{A, B, D\}$  wäre eine solche Menge
- **Minimalschnitte** (engl. *minimal cut-sets*) besonders interessant
  - Minimaler Auslöser
  - z. B.  $\{A, B\}$ ,  $\{A, D\}$ ,  $\{B, C\}$






Ein **Schnitt** (engl. *cut-set*) enthält genau die atomaren Ereignisse, die das Schadensereignis verursachen:



- $\{A, B, D\}$  wäre eine solche Menge
- **Minimalschnitte** (engl. *minimal cut-sets*) besonders interessant
  - Minimaler Auslöser
  - z. B.  $\{A, B\}$ ,  $\{A, D\}$ ,  $\{B, C\}$
- Fehlerbäume als **logische Ausdrücke**
  - Umformung durch Aussagenlogik
  - Bestimmung durch **SAT-Solving**



 Minimalschnitte liefern genau die **kritischen atomaren Ereignisse**, die ein unerwünschtes Systemverhalten **hervorrufen**

 Diese Defekte(-szenarien) sind zu vermeiden!

☞ Minimalschnitte liefern genau die **kritischen atomaren Ereignisse**, die ein unerwünschtes Systemverhalten **hervorrufen**

⚠ Diese Defekte(-szenarien) sind zu vermeiden!

- Duales Konzept: **Minimalpfade** (engl. *minimal path-sets*)
  - Die minimale Menge **atomarer Ereignisse**, welche das unerwünschte Schadensereignis **verhindern**
    - Sofern die mit ihnen verbundenen Defekte nicht auftreten
  - Es **genügt** also, diese Defekte auszuschließen!

☞ Minimalschnitte liefern genau die **kritischen atomaren Ereignisse**, die ein unerwünschtes Systemverhalten **hervorrufen**



Diese Defekte(-szenarien) sind zu vermeiden!

- Duales Konzept: **Minimalpfade** (engl. *minimal path-sets*)
  - Die minimale Menge **atomarer Ereignisse**, welche das unerwünschte Schadensereignis **verhindern**
    - Sofern die mit ihnen verbundenen Defekte nicht auftreten
  - Es **genügt** also, diese Defekte auszuschließen!
  - Berechnung: Tausche UND- und ODER-Verknüpfungen
    - Bestimme anschließend die entsprechendenden Minimalschnitte



☞ Minimalschnitte liefern genau die **kritischen atomaren Ereignisse**, die ein unerwünschtes Systemverhalten **hervorrufen**



Diese Defekte(-szenarien) sind zu vermeiden!

- Duales Konzept: **Minimalpfade** (engl. *minimal path-sets*)
  - Die minimale Menge **atomarer Ereignisse**, welche das unerwünschte Schadensereignis **verhindern**
    - Sofern die mit ihnen verbundenen Defekte nicht auftreten
  - Es **genügt** also, diese Defekte auszuschließen!
  - Berechnung: Tausche UND- und ODER-Verknüpfungen
    - Bestimme anschließend die entsprechendenden Minimalschnitte
  - Im Beispiel auf Folie III/23 sind dies:  $\{A, B\}$ ,  $\{A, C\}$ ,  $\{B, D\}$



- 1 Fehler
- 2 Verlässlichkeitsmodelle
- 3 Fehler und Systementwurf
- 4 Software- und Hardwarefehler**
- 5 Zusammenfassung



- Softwarefehler sind stets **permanente Defekte**
  - Manifestieren sich nicht zwingend in permanentem Fehlverhalten
    - Beispiel: Therac-25 (vgl. Folien II/3 ff)
  - Beispiel: **Heisenbugs** verursacht durch Nebenläufigkeitsfehler
    - Auch **Bohrbugs**, **Mandelbugs** oder **Schrödinbugs**
    - Treten manchmal auf, manchmal nicht  $\leadsto$  sehr schwer zu reproduzieren



- Softwarefehler sind stets **permanente Defekte**
  - Manifestieren sich nicht zwingend in permanentem Fehlverhalten
    - Beispiel: Therac-25 (vgl. Folien II/3 ff)
  - Beispiel: **Heisenbugs** verursacht durch Nebenläufigkeitsfehler
    - Auch **Bohrbugs**, **Mandelbugs** oder **Schrödinbugs**
    - Treten manchmal auf, manchmal nicht  $\leadsto$  sehr schwer zu reproduzieren
- Ursache: **Fehlerhaften Umsetzung** der Spezifikation
  - In der Regel durch den Programmierer, Architekten, ...
  - Ursprung: Anforderungserhebung, Entwurf, Implementierung, ...



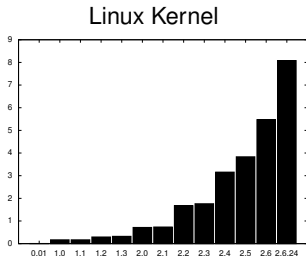
- Softwarefehler sind stets **permanente Defekte**
  - Manifestieren sich nicht zwingend in permanentem Fehlverhalten
    - Beispiel: Therac-25 (vgl. Folien II/3 ff)
  - Beispiel: **Heisenbugs** verursacht durch Nebenläufigkeitsfehler
    - Auch **Bohrbugs**, **Mandelbugs** oder **Schrödinbugs**
    - Treten manchmal auf, manchmal nicht  $\leadsto$  sehr schwer zu reproduzieren
- Ursache: **Fehlerhaften Umsetzung** der Spezifikation
  - In der Regel durch den Programmierer, Architekten, ...
  - Ursprung: Anforderungserhebung, Entwurf, Implementierung, ...
- Softwarefehler sind **systematische Fehler**
  - Betrieb mehrere Instanzen unter **gleichen, äußeren Bedingungen** führt zu **identischen Fehlern**
    - Äußeren Bedingungen sind allerdings nicht ohne Weiteres reproduzierbar
    - Beispiel: Ausfall von SRI 1 und SRI 2 (vgl. Folien II/16 ff) der Ariane 5 [10]





**Komplexität** ist der **natürliche Feind** korrekter Programme

- Komplexität nimmt stetig zu: (Million-)LOC



Microsoft Windows [11]

Jahr	Produkt	Dev	Test	MLOC
1993	NT 3.1	200	140	4-5
1994	NT 3.5	300	230	7-8
1995	NT 3.51	450	325	9-10
1996	NT 4.0	800	700	11-12
1999	NT 5.0	1400	1700	> 29
2001	NT 5.1	1800	2200	40
2003	NT 5.2	2000	2400	50

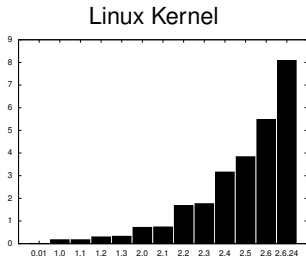
- Angefangen hat Linux in Version 1.0 mit ca. 170 KLOC
- In Version 3.0 ist Linux bei ca. **15 Millionen LOC** angekommen
- In Version 4.15 ist Linux bei ca. **20 Millionen LOC** angekommen





**Komplexität** ist der **natürliche Feind** korrekter Programme

- Komplexität nimmt stetig zu: (Million-)LOC



Microsoft Windows [11]

Jahr	Produkt	Dev	Test	MLOC
1993	NT 3.1	200	140	4-5
1994	NT 3.5	300	230	7-8
1995	NT 3.51	450	325	9-10
1996	NT 4.0	800	700	11-12
1999	NT 5.0	1400	1700	> 29
2001	NT 5.1	1800	2200	40
2003	NT 5.2	2000	2400	50

- Angefangen hat Linux in Version 1.0 mit ca. 170 KLOC
- In Version 3.0 ist Linux bei ca. **15 Millionen LOC** angekommen
- In Version 4.15 ist Linux bei ca. **20 Millionen LOC** angekommen



**Faustregel:** ca. 3 Defekte je 1000 LOC

- Pessimistischere Schätzungen: bis zu 10 Defekten je 1000 LOC aus
- **Hunderttausende Defekte** in Linux 3.0 bzw. Windows NT 5.2



# Software will gepflegt werden!

Anforderungen an langlebige Softwaresysteme unterliegen ständigem Wandel

- Folgender Zusammenhang ist einfach interessant
  - Hier wird explizit **keine Kausalbeziehung** aufgestellt!



# Software will gepflegt werden!

Anforderungen an langlebige Softwaresysteme unterliegen ständigem Wandel

- Folgender Zusammenhang ist einfach interessant
  - Hier wird explizit **keine Kausalbeziehung** aufgestellt!
- Chou, SOSP 2001 [2]: Den Großteil der Softwaredefekte im Linux-Kern findet man in Gerätetreibern
  - Wenig verwunderlich: Der Großteil des Linux-Kerns sind Gerätetreiber
  - **Aber:** auch die Fehlerrate ist in Gerätetreibern am größten



# Software will gepflegt werden!

Anforderungen an langlebige Softwaresysteme unterliegen ständigem Wandel

- Folgender Zusammenhang ist einfach interessant
  - Hier wird explizit **keine Kausalbeziehung** aufgestellt!
- Chou, SOSP 2001 [2]: Den Großteil der Softwaredefekte im Linux-Kern findet man in Gerätetreibern
  - Wenig verwunderlich: Der Großteil des Linux-Kerns sind Gerätetreiber
  - **Aber:** auch die Fehlerrate ist in Gerätetreibern am größten
- Padioleau, EuroSys 2006 [13]: Gerätetreiber und die zugehörigen Bibliotheken wachsen im Linux-Kern am stärksten
  - Bibliotheken und Treiber ändern sich ständig
    - Änderungen an den Bibliotheken erfordern Änderungen in den Treibern
    - **Collateral Evolution** bedingt durch **Refactoring**



# Software will gepflegt werden!

Anforderungen an langlebige Softwaresysteme unterliegen ständigem Wandel

- Folgender Zusammenhang ist einfach interessant
  - Hier wird explizit **keine Kausalbeziehung** aufgestellt!
- Chou, SOSP 2001 [2]: Den Großteil der Softwaredefekte im Linux-Kern findet man in Gerätetreibern
  - Wenig verwunderlich: Der Großteil des Linux-Kerns sind Gerätetreiber
  - **Aber:** auch die Fehlerrate ist in Gerätetreibern am größten
- Padioleau, EuroSys 2006 [13]: Gerätetreiber und die zugehörigen Bibliotheken wachsen im Linux-Kern am stärksten
  - Bibliotheken und Treiber ändern sich ständig
    - Änderungen an den Bibliotheken erfordern Änderungen in den Treibern
    - **Collateral Evolution** bedingt durch **Refactoring**
- Kim, ICSE 2011 [9]: Welche Rolle spielt Refactoring?
  - Ergebnis: Mehr Fehlerbehebungen nach Refactoring
    - Fehler durch **fehlerhaftes Refactoring**, Refactoring **für die Fehlerbehebung**



## ■ Permanente Hardwarefehler

- **Extrinsischer Natur:** Herstellungsbedingte Materialfehler
  - z. B. fehlerhafte Dotierung eines Halbleiters oder Materialunreinheiten
  - Treten meist zu Beginn der Lebenszeit auf (↷ **Säuglingssterblichkeit**)
- **Intrinsischer Natur:** Verschleißerscheinungen
  - Kündigen sich meist durch sporadische Fehler an
  - Treten meist am Ende der Lebenszeit auf



## ■ Permanente Hardwarefehler

- **Extrinsischer Natur:** Herstellungsbedingte Materialfehler
  - z. B. fehlerhafte Dotierung eines Halbleiters oder Materialunreinheiten
  - Treten meist zu Beginn der Lebenszeit auf ( $\leadsto$  Säuglingssterblichkeit)
- **Intrinsischer Natur:** Verschleißerscheinungen
  - Kündigen sich meist durch sporadische Fehler an
  - Treten meist am Ende der Lebenszeit auf

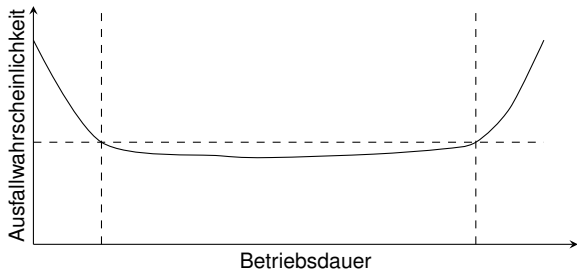
## ■ Transiente Hardwarefehler $\mapsto$ Umwelteinflüsse

- **Mannigfaltige Ursachen**
  - Radioaktive Strahlung
  - Elektromagnetische Interferenz
  - Instabile Spannungsversorgung
  - Fertigungsstreuung bei einzelnen Transistoren
  - Temperaturschwankungen führen zum temporären Materialdefekten
  - ...

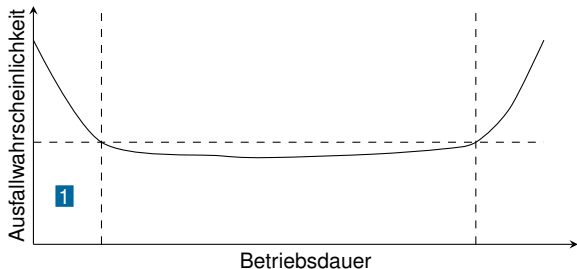
$\rightarrow$  Treten als schwer zu fassende „Bitkipper“ in Erscheinung



# Wahrscheinlichkeit permanenter Hardwarefehler



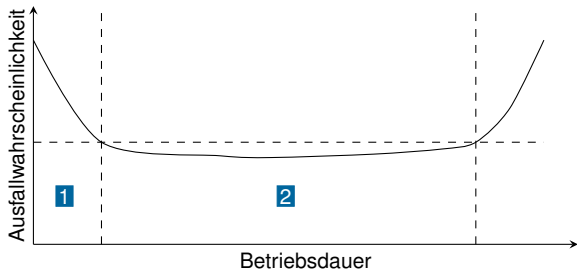
# Wahrscheinlichkeit permanenter Hardwarefehler



- 1 Erhöhte **Säuglingssterblichkeit** durch fertigungsbedingte Defekte
  - Eine **Einbrennphase** (engl. *burn-in*) filtert fehlerhafte Elemente heraus

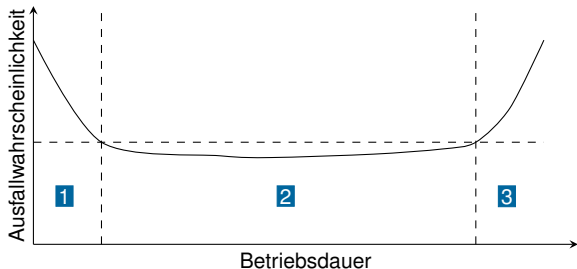


# Wahrscheinlichkeit permanenter Hardwarefehler



- 1 Erhöhte **Säuglingssterblichkeit** durch fertigungsbedingte Defekte
  - Eine **Einbrennphase** (engl. *burn-in*) filtert fehlerhafte Elemente heraus
- 2 Normaler, sinnvoll nutzbarer **Betriebszeitraum**
  - Ausfallrate nahe an der durchschnittlichen Ausfallwahrscheinlichkeit

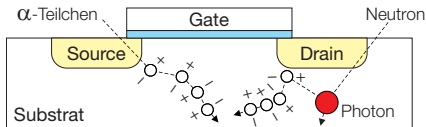




- 1 Erhöhte **Säuglingssterblichkeit** durch fertigungsbedingte Defekte
  - Eine **Einbrennphase** (engl. *burn-in*) filtert fehlerhafte Elemente heraus
- 2 Normaler, sinnvoll nutzbarer **Betriebszeitraum**
  - Ausfallrate nahe an der durchschnittlichen Ausfallwahrscheinlichkeit
- 3 Durch **Verschleiß** bedingte Ausfälle
  - Auch Halbleiterbauelement unterliegen einem Verschleißprozess
  - z. B. Elektromigration, Spannungsrisse durch thermische Belastungen, Verschleiß der Oxidschicht am Gate ...



# Transiente Hardwarefehler (engl. *soft errors*)



☞ **Bitkipper** durch **Umladungen in Speicherzellen und Schaltkreisen**

→ Ionisierende Strahlung erzeugt **Elektronen-Loch-Paaren** (Defektelektronen)

## ■ Direkt durch **Alphateilchen**

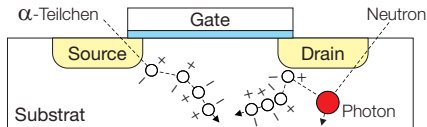
- Quelle: Kontaminierten Chipgehäusen oder Lötkegeln
- Die „ersten transienten Fehler“ [12, Kapitel 1.1]

## ■ Indirekt durch Neutronen aus der **kosmischen Strahlung**

- **Primäre kosmische Strahlung**: Galaktische und solare Partikel
- **Sekundäre Strahlung**: Wechselwirkung in der Erdatmosphäre
- **Terrestrische Strahlung**: Die Erdoberfläche erreichende Partikel



# Transiente Hardwarefehler (engl. *soft errors*)



## Bitkipper durch Umladungen in Speicherzellen und Schaltkreisen

→ Ionisierende Strahlung erzeugt Elektronen-Loch-Paaren (Defektelektronen)

### ■ Direkt durch Alphateilchen

- Quelle: Kontaminierten Chipgehäusen oder Lötkegeln
- Die „ersten transienten Fehler“ [12, Kapitel 1.1]

### ■ Indirekt durch Neutronen aus der kosmischen Strahlung

- Primäre kosmische Strahlung: Galaktische und solare Partikel
- Sekundäre Strahlung: Wechselwirkung in der Erdatmosphäre
- Terrestrische Strahlung: Die Erdoberfläche erreichende Partikel

## Elektromagnetische Interferenzen

- Verfälschung von Kommunikation auf Bussen
- z. B. in Automobilen gibt es starke Quellen für Wechselfelder

→ Sparsame elektronische Abschirmung macht dies zum Problem



# Anfälligkeit für transiente Fehler



Fehlerrate (engl. *soft-error rate*, *SER*) eines Schaltkreises hängt (stark vereinfacht) von folgenden Faktoren ab:

$$\text{SER} = C \times \text{Neutronenfluss} \times \text{Fläche} \times e^{-Q_{\text{crit}}/Q_{\text{coll}}}$$





Fehlerrate (engl. *soft-error rate*, *SER*) eines Schaltkreises hängt (stark vereinfacht) von folgenden Faktoren ab:

$$\text{SER} = C \times \text{Neutronenfluss} \times \text{Fläche} \times e^{-Q_{\text{crit}}/Q_{\text{coll}}}$$

$C$  prozess- und schaltkreisspezifische Konstante

Fläche des Schaltkreises

$Q_{\text{crit}}$  minimale für eine Fehlfunktion notwendige Ladung

- Typischer Wert: 1 fC

$Q_{\text{coll}}$  Effizienz der Ladungsaufnahme

- Bestimmt durch die für die Erzeugung der Elektronen-Loch-Paare notwendige Energie
- Abhängig von Material und Fertigungsprozess  $\rightarrow$  Bremsvermögen (engl. *stopping power*)
- Typischer Wert:  $4.5 \text{ fC } \mu\text{m}^{-1} \rightsquigarrow 22 \text{ keV}$  Teilchenenergie ausreichend



# Anfälligkeit für transiente Fehler



Fehlerrate (engl. *soft-error rate*, *SER*) eines Schaltkreises hängt (stark vereinfacht) von folgenden Faktoren ab:

$$\text{SER} = C \times \text{Neutronenfluss} \times \text{Fläche} \times e^{-Q_{\text{crit}}/Q_{\text{coll}}}$$

$C$  prozess- und schaltkreisspezifische Konstante

Fläche des Schaltkreises

$Q_{\text{crit}}$  minimale für eine Fehlfunktion notwendige Ladung

- Typischer Wert: 1 fC

$Q_{\text{coll}}$  Effizienz der Ladungsaufnahme

- Bestimmt durch die für die Erzeugung der Elektronen-Loch-Paare notwendige Energie
- Abhängig von Material und Fertigungsprozess  $\rightarrow$  Bremsvermögen (engl. *stopping power*)
- Typischer Wert:  $4.5 \text{ fC } \mu\text{m}^{-1} \rightsquigarrow 22 \text{ keV}$  Teilchenenergie ausreichend



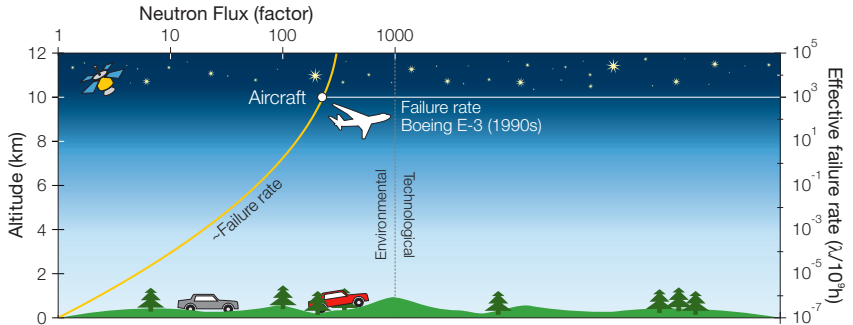
Kleinere Halbleiterstrukturen sind Fluch und Segen zugleich

$\rightarrow$  Kleinere Fläche  $\rightsquigarrow$  kleinere SER

$\rightarrow$  Kleinere  $Q_{\text{crit}}$   $\rightsquigarrow$  größere SER



# Fehlerraten – Entwicklung und Tendenzen



Zahlen aus: [14]

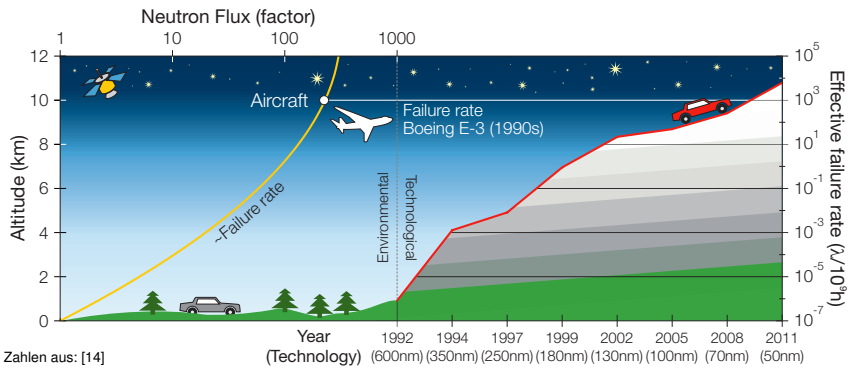


Exakte Fehlerrate ist schwer zu ermitteln

- Fehlerrate **pro Bit** stagniert oder verbessert sich
- Fehlerrate des **Systems** hängt indes von vielen Faktoren ab



# Fehlerraten – Entwicklung und Tendenzen



## Exakte Fehlerrate ist schwer zu ermitteln

- Fehlerrate **pro Bit** stagniert oder verbessert sich
- Fehlerrate des **Systems** hängt indes von vielen Faktoren ab



## Vorhersagen von Forschern und Herstellern [1, 5]

- Mehr Leistung und Parallelität  $\rightsquigarrow$  **Auf Kosten der Zuverlässigkeit**



- 1 Fehler
- 2 Verlässlichkeitsmodelle
- 3 Fehler und Systementwurf
- 4 Software- und Hardwarefehler
- 5 Zusammenfassung**



**Fehler**  $\rightsquigarrow$  Alles dreht sich ausschließlich um Fehler!

- Fehlerfortpflanzung: Defekt  $\rightsquigarrow$  Fehler  $\rightsquigarrow$  Fehlverhalten-Kette
- permanente, sporadische und transiente Fehler
- Vorbeugung, Entfernung, Vorhersage und Toleranz

**Verlässlichkeitsmodelle**  $\rightsquigarrow$  Umgang mit Fehlern?

- Verlässlichkeit, Zuverlässigkeit, Wartbarkeit und Verfügbarkeit

**Systementwurf**  $\rightsquigarrow$  Bereits hier werden Fehler berücksichtigt!

- Gefahren-, Risiko- und Fehlerbaumanalyse

**Software- vs. Hardwarefehler**  $\rightsquigarrow$  Klassifikation & Ursachen

- Softwarefehler  $\mapsto$  permanente Defekte, Komplexität
- Hardwarefehler  $\mapsto$  permanente & transiente Fehler, Fertigung, ionisierende Strahlung, elektromagnetische Interferenz



- [1] Borkar, S. :  
Designing reliable systems from unreliable components: the challenges of transistor variability and degradation.  
In: *IEEE Micro* 25 (2005), November, Nr. 6, S. 10–16.  
<http://dx.doi.org/10.1109/MM.2005.110>. –  
DOI 10.1109/MM.2005.110. –  
ISSN 0272–1732
- [2] Chou, A. ; Yang, J. ; Chelf, B. ; Hallem, S. ; Engler, D. :  
An empirical study of operating systems errors.  
In: Marzullo, K. (Hrsg.) ; Satyanarayanan, M. (Hrsg.): *Proceedings of the 18th ACM Symposium on Operating Systems Principles (SOSP '01)*.  
New York, NY, USA : ACM Press, 2001. –  
ISBN 1–58113–389–8, S. 73–88
- [3] Deutsches Institut für Normung:  
*Fehlerbaumanalyse; Handrechenverfahren zur Auswertung eines Fehlerbaumes*.  
Berlin, Wien, Zürich : Beuth-Verlag, 1990 (DIN 25424)
- [4] Deutsches Institut für Normung:  
*Qualitätsmanagement – Begriffe*.  
Berlin, Wien, Zürich : Beuth-Verlag, 1995 (DIN 8402)

- [5] Dixit, A. ; Heald, R. ; Wood, A. :  
Trends from ten years of soft error experimentation.  
In: *Proceedings of the 5th Workshop on Silicon Errors in Logic – System Effects (SLSE '09)*,  
2009
- [6] IFIP:  
*Working Group 10.4 on Dependable Computing and Fault Tolerance.*  
<http://www.dependability.org/wg10.4>, 2003
- [7] International Organization for Standardization:  
*Part 3: Concept phase.*  
Genf, Schweiz : International Organization for Standardization, 2011 (ISO 26262: Road vehicles  
– Functional safety)
- [8] Kaiser, B. ; Liggesmeyer, P. ; Mäckel, O. :  
A new component concept for fault trees.  
In: *SCS '03: Proceedings of the 8th Australian workshop on Safety critical systems and software.*  
Darlinghurst, Australia, Australia : Australian Computer Society, Inc., 2003. –  
ISBN 1–920–68215–5, S. 37–46



- [9] Kim, M. ; Cai, D. ; Kim, S. :  
An empirical investigation into the role of API-level refactorings during software evolution.  
In: Taylor, R. N. (Hrsg.) ; Gall, H. (Hrsg.) ; Medvidović, N. (Hrsg.): *Proceedings of the 33rd International Conference on Software Engineering (ICSE '11)*.  
New York, NY, USA : ACM Press, Mai 2011. –  
ISBN 978-1-4503-0445-0, S. 151-160
- [10] Le Lann, G. :  
An analysis of the Ariane 5 flight 501 failure – a system engineering perspective.  
In: *Proceedings of International Conference and Workshop on Engineering of Computer-Based Systems (ECBS 1997)*.  
Washington, DC, USA : IEEE Computer Society, März 1997. –  
ISBN 0-8186-7889-5, S. 339-346
- [11] Maraia, V. :  
*The Build Master: Microsoft's Software Configuration Management Best Practices*.  
Addison-Wesley, 2005. –  
ISBN 978-0321332059
- [12] Mukherjee, S. :  
*Architecture Design for Soft Errors*.  
San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2008. –  
ISBN 978-0-12-369529-1



- [13] Padioleau, Y. ; Lawall, J. L. ; Muller, G. :  
Understanding Collateral Evolution in Linux Device Drivers.  
In: Berbers, Y. (Hrsg.) ; Zwaenepoel, W. (Hrsg.): *Proceedings of the ACM SIGOPS/EuroSys European Conference on Computer Systems 2006 (EuroSys '06)*.  
New York, NY, USA : ACM Press, Apr. 2006. –  
ISBN 1-59593-322-0, S. 59-71
- [14] Shivakumar, P. ; Kistler, M. ; Keckler, S. W. ; Burger, D. ; Alvisi, L. :  
Modeling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic.  
In: *Proceedings of the 32nd International Conference on Dependable Systems and Networks (DSN '02)*.  
Washington, DC, USA : IEEE Computer Society Press, Jun. 2002, S. 389-398

