

# Verlässliche Echtzeitsysteme

## Grundlagen

Fabian Scheler

Friedrich-Alexander-Universität Erlangen-Nürnberg  
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)  
[www4.informatik.uni-erlangen.de](http://www4.informatik.uni-erlangen.de)

25. April 2012



## Gliederung

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



## Überblick

*Wir kümmern uns ausschließlich um Fehler!*

- ~ Das ist nur ein kleiner Aspekt zuverlässiger Systeme!
  - ...aber dennoch sind nicht alle Fehler gleich ...
    - Beeinflusst jeder Fehler das Verhalten eines Systems?
  - Was bedeutet es, mit Fehlern umgehen zu können?
    - Wie gibt man z. B. an, dass ein korrektes Ergebnis geliefert werden wird?
    - Worin unterscheiden sich etwa Zuverlässigkeit und Verfügbarkeit?
  - Wie schwerwiegend ist ein Fehler?
    - Welchen Schaden kann ein Fehler verursachen?
  - Software- vs. Hardware-Fehler
    - grundlegende Klassifikation, Ursachen und Entstehung



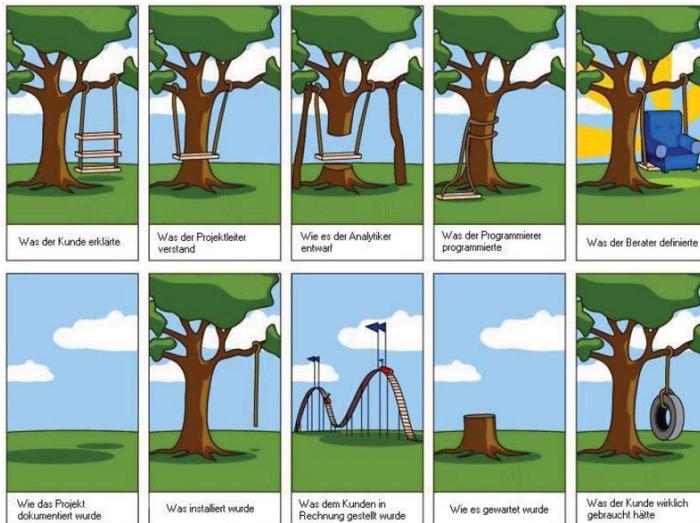
## Definition: Fehler

- laut DIN EN ISO 8402:1995-08 [3] ist ein Fehler die „Nichterfüllung einer festgelegten Forderung“
  - 
  - Spezifikation
  - Implementierung
- Fehler kennen demzufolge viele Ausprägungen, ...
  - sie können lediglich als störend empfunden werden
    - die 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:** der Bezugspunkt ist die Spezifikation ~ **Verifikation**
  - Haben wir das System korrekt implementiert?



## Abgrenzung: Keine Validierung

Haben wir das korrekte System implementiert?



© fs (FAU/INF4)

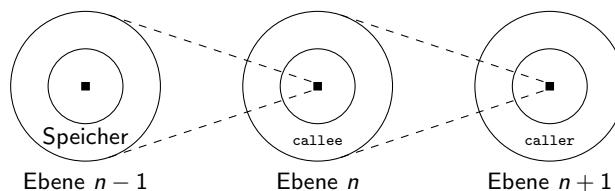
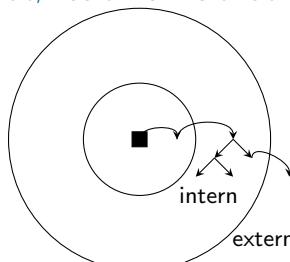
Verlässliche Echtzeitsysteme (SS 2013) – Kapitel III Grundlagen  
2 Fehler

5/38

## Es ist alles eine Frage der Sichtbarkeit

Was ich nicht weiß, macht mich nicht heiß!

- gutartige Defekte haben **keinen Einfluss** auf das korrekte Systemverhalten
- bösartige Defekte/**innere Fehler** beeinflussen den **internen Zustand**
  - intern kann sich der Fehler weiter verbreiten
- wird der innerer Fehler nach außen gereicht, ist er als **Fehlverhalten** sichtbar
- über die fault ~ error ~ failure-Kette **pflanzen sich Fehler fort**



- und können schließlich zum **vollständigen Systemversagen** führen

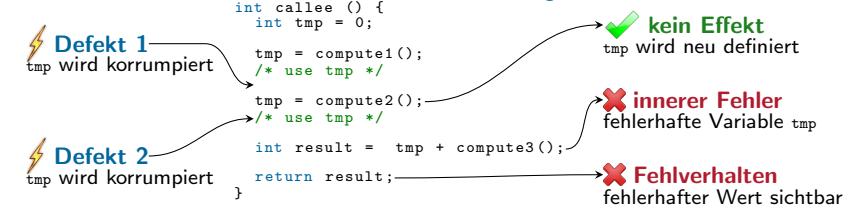
© fs (FAU/INF4)

Verlässliche Echtzeitsysteme (SS 2013) – Kapitel III Grundlagen  
2 Fehler

7/38

## Wann ist ein Fehler nun ein Fehler?

Das Problem beginnt, wenn der Schuh drückt!



- **Defekte** (engl. *faults*) sind die Quelle allen Übels
  - Ursachen: Software-Bugs, Produktionsfehler, äußere Einflüsse, ...
  - beziehen sich auf die **Struktur**
  - **gutartige Defekte** (engl. *benign faults*) führen nicht zu einem Fehler
- die Manifestation eines Defekts ist ein **innerer Fehler** (engl. *error*)
  - ~ der Defekt ist also **bösartig** (engl. *malign fault*)
  - beziehen sich auf den **nicht sichtbaren, inneren Zustand**
- außen sichtbare, innere Fehler heißen **Fehlverhalten** (engl. *failure*)
  - beziehen sich auf das **beobachtbare Verhalten**
  - man spricht von der **fault ~ error ~ failure**-Kette [9, Kapitel 1]



© fs (FAU/INF4) Verlässliche Echtzeitsysteme (SS 2013) – Kapitel III Grundlagen  
2 Fehler

6/38

## Klassifikation nach dem Auftreten

Weiter Eigenschaften von Fehlern

- 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 fault ~ error ~ failure-Kette:
  - normalerweise: transiente Defekte ↘ permanentes Fehlverhalten
  - möglich: permanenter Defekt ~ transientes Fehlverhalten
    - wenn sie nur unregelmäßig den inneren Sichtbarkeitsbereich verlassen



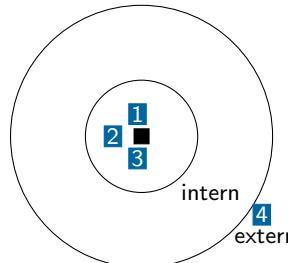
© fs (FAU/INF4) Verlässliche Echtzeitsysteme (SS 2013) – Kapitel III Grundlagen  
2 Fehler

8/38

## Maßnahmen zum Umgang mit Fehlern

Versuchen die fault ~ error ~ failure-Kette 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 oder ihre Umgehung
- 4 **Toleranz** – verhindert nicht den Defekt, aber die Fortpflanzung zum Fehlverhalten
  - z.B. durch Maskierung innerer Fehler



### Ziel zuverlässiger Systeme

Reduktion des vom Benutzer beobachtbaren Fehlverhaltens



© fs (FAU/INF4)

Verlässliche Echtzeitsysteme (SS 2013) – Kapitel III Grundlagen  
2 Fehler

9/38

## Gliederung

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



© fs (FAU/INF4)

Verlässliche Echtzeitsysteme (SS 2013) – Kapitel III Grundlagen  
3 Verlässlichkeitsmodelle

11/38

## Folgen von Fehlern

Wenn die fault ~ error ~ failure-Kette zugeschlagen hat ...

Nicht jedes beobachtbare Fehlverhalten muss auch entdeckt werden:



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

- unbemerkte Fehlerfortpflanzung innerhalb oder außerhalb des Systems
  - fehlerhafte Berechnungsergebnisse oder Ausgabewerte
- sehr, sehr schwer ausfindig zu machen
  - das verursachte Fehlverhalten wird erst viel später sichtbar
- die Fehlererkennung verhindert die unbemerkte Fehlerausbreitung
  - **Zusicherungen** (engl. *assertions*) übernehmen genau diese Aufgabe



### unkorrigierbare Fehler (engl. *detected unrecoverable error, DUE*)

- eine Fortpflanzung kann gezielt unterbunden werden (↪ **fail-stop**)
- die Stellen, an diese Fehler auftreten, lassen sich vergleichsweise einfach herausfinden, z. B. durch eine **Ablaufverfolgung** (engl. *backtrace*)

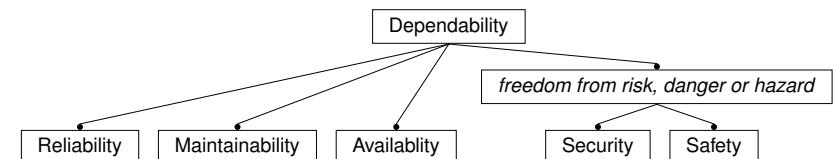


© fs (FAU/INF4)

Verlässliche Echtzeitsysteme (SS 2013) – Kapitel III Grundlagen  
2 Fehler

10/38

## „Verlässlichkeit“ ist ein vielschichtiger Begriff



*The trustworthiness of a computing system which allows reliance to be justifiably placed on the service it delivers. [4]*



© fs (FAU/INF4)

Verlässliche Echtzeitsysteme (SS 2013) – Kapitel III Grundlagen  
3 Verlässlichkeitsmodelle

12/38

## Zuverlässigkeit (engl. reliability)

Mittlere Betriebsdauer

$R(t)$  die Wahrscheinlichkeit, dass ein System seinen Dienst bis zum Zeitpunkt  $t$  leisten wird, sofern es bei  $t = t_0$  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
  - das Kfz sei durchschnittlich eine Stunde täglich in Betrieb
  - dann darf jährlich nur ein Fehler pro eine Million Kfz auftreten
- Beispiele: Eisenbahnsignalanlagen, Kernkraftwerküberwachung

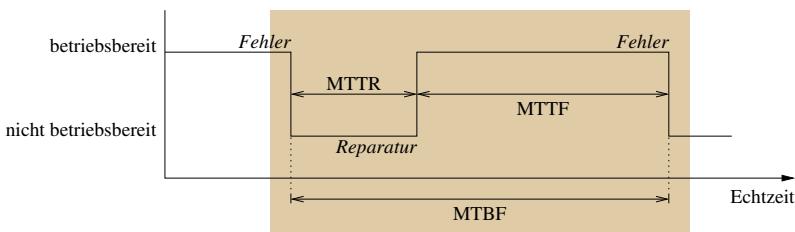


## Verfügbarkeit (engl. availability)

MTTF und MTTR im Zusammenhang

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

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



☞ hohe Verfügbarkeit bedeutet kurze MTTR und/oder lange MTTF



## Wartbarkeit (engl. maintainability)

Mittlere Reparaturdauer

$M(d)$  die Wahrscheinlichkeit, dass das System innerhalb Zeitspanne  $d$  nach einem reparierbaren Fehler 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 (physische) Fehlerrate gegeben
  - darüberhinaus verbuchen sich höhere Herstellungskosten
- ein zuverlässiges System ist aus einem Guss gefertigt...

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



## Sicherheit $\mapsto$ Security und Safety

Robustheit des Echtzeitrechensystems stärken

**security** Schutz von Informationen und Informationsverarbeitung vor „intelligenten“ Angreifern

- allgemein in Bezug auf **Datenbasen** des Echtzeitystems
  - **Vertraulichkeit** (engl. *confidentiality*)
  - **Datenschutz** (engl. *privacy*)
  - **Glaubwürdigkeit** (engl. *authenticity*)
- speziell z.B. Diebstahlsicherung: Zündungssperre im Kfz
  - **Kryptographie** (engl. *cryptography*)

**safety** Schutz von Menschen und Sachwerten vor dem Versagen technischer Systeme

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



## Verlässlichkeit ↔ Komplexität

Automobil eine Bestandsaufnahme vom Jahr 2005 ...

- etwa 90 % der Innovationen im Auto bringen die Elektronik ein
  - gut 80 % davon sind Software
- etwa ein Drittel aller Pannen liegen an fehlerhaften 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)*



## Gliederung

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



## Verlässlichkeit unterscheidet sich je nach System

Je nach dem, wie kritisch sich ein einzelner Fehler auswirkt.

**Hochverfügbare Systeme** z. B. Telekommunikationstechnik

- müssen ihren Dienst möglichst ununterbrochen verrichten
  - einzelne Fehler sind jedoch verkraftbar (~ 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 (~ 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 (~ fail-safe)
  - Diese Anlagen sind nur sinnvoll, wenn sie im Betrieb sind!
- hohe Ansprüche an Zuverlässigkeit und Verfügbarkeit



## Wie schwer wiegt das Fehlverhalten eines Systems?

- Klärung durch eine **Gefahrenanalyse und Risikobeurteilung**
    - **Identifikation** gefährlicher Ereignisse und
    - ihre **Klassifikation** hinsichtlich verschiedener Kriterien
  - **Faustregel:** Risiko = Wahrscheinlichkeit x 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 [5]
- Kriterien: Schweregrade nach ISO 26262:

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



## Zusammenhang: Defekt $\leftrightarrow$ Fehlverhalten

Welche Defekte führen zum beobachtbaren Fehlverhalten?

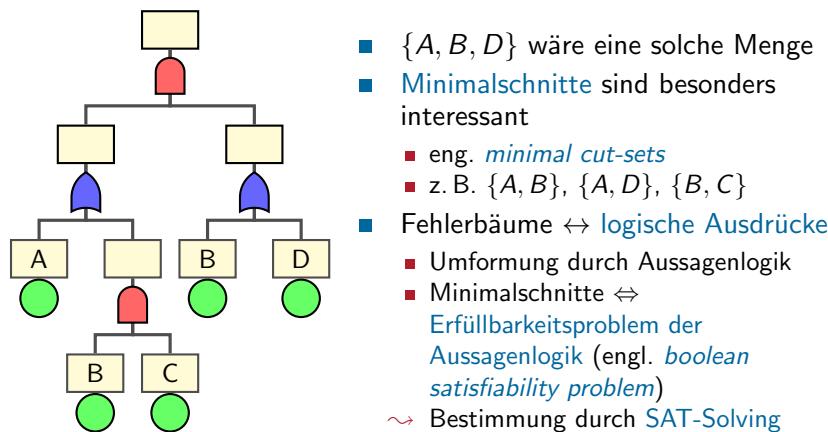
- eine Fehlerbaumanalyse (engl. *fault-tree analysis*) [2] ermittelt die Ereignisse, die zum beobachtbaren Systemverhalten führen
  - verfeinernde Analyse (engl. *top-down analysis*)
    - das unerwünschte Fehlverhalten bildet die Wurzel des Fehlerbaumes
    - ausgehend davon werden die Ursachen des Fehlverhaltens identifiziert
  - arbeitet auf dem **Fehlerraum** (engl. *failure space*) des Systems
    - $\leftrightarrow$  Zuverlässigkeitsschleifendiagrammen (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 ausfallen



## Schnitte und Fehlerbäume

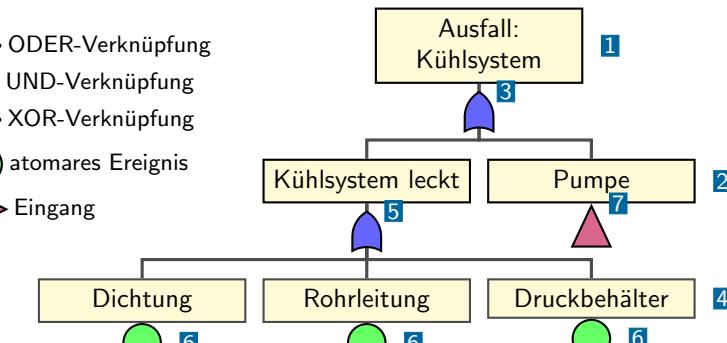
Welche Defekte führen letztendlich zum Systemausfall?

Ein **Schnitt** (engl. *cut-set*) enthält genau die atomaren Ereignisse, die das „Top-Level“-Ereignis verursachen:



## Aufbau und Erstellung von Fehlerbäumen

- ODER-Verknüpfung
- UND-Verknüpfung
- XOR-Verknüpfung
- atomares Ereignis
- Eingang



- 1  $\rightsquigarrow$  „Top-Level“-Ereignis
- 2  $\rightsquigarrow$  Ereignisse auf Ebene 2
- 3 verknüpft sie logisch
- 4  $\rightsquigarrow$  Ereignisse auf Ebene 3
- 5 verknüpft sie logisch
- 6 atomare Ereignisse beenden Gliederung
- 7 Eingänge zerlegen den Fehlerbaum  $\rightsquigarrow$  neuer Fehlerbaum



## Schnitte und Fehlerbäume (Forts.)

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

☞ Es lohnt sich, diese Defekte zu vermeiden!

- duales Konzept: **Minimalpfade** (engl. *path-sets*)
  - die minimale Menge atomarer Ereignisse, die das unerwünschte „Top-Level“-Ereignis verhindern
    - sofern die mit ihnen verbundenen Defekte nicht auftreten
  - Es genügt also, diese Defekte auszuschließen!
  - Berechnung: tausche UND- und ODER-Verknüpfungen
    - $\rightsquigarrow$  bestimme anschließend die entsprechenden Minimalschnitte
  - im Beispiel auf Folie III/23 sind dies:  $\{A, B\}, \{A, C\}, \{B, D\}$



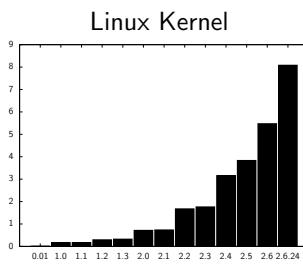
- 1 Überblick
- 2 Fehler
- 3 Verlässlichkeitsmodelle
- 4 Fehler und Systementwurf
- 5 Software- und Hardwarefehler
- 6 Zusammenfassung



## Ursachenforschung – Wie entstehen Softwarefehler?

Eine Facette eines komplexen Problems

- Komplexität ist der natürliche Feind korrekter Programme
  - ... und die Komplexität nimmt stetig zu: (Million-)LOC



- angefangen hat Linux in Version 1.0 mit ca. 170 KLOC
- in Version 3.0 ist Linux bei ca. 15 Millionen LOC angekommen

- Faustregel: ca. 3 Defekte je 1000 LOC
  - pessimistischere Schätzungen gehen von bis 10 Defekten je 1000 LOC aus
  - ~ ca. 1,5 bzw. 5 Millionen Defekte in Linux 3.0 bzw. Windows NT 5.2



## Softwarefehler (engl. software bugs) ...

- sind permanente Defekte
  - manifestieren sich aber nicht unbedingt in einem inneren Fehler, von außen beobachtbarem Fehlverhalten oder einem Systemausfall
  - Beispiel: sog. **Heisenbugs** verursacht durch Nebenläufigkeitsfehler
    - auch **Bohrbugs**, **Mandelbugs** oder **Schrödinbugs**
    - treten manchmal auf, manchmal nicht ~ sehr schwer zu reproduzieren
- resultieren aus einer **fehlerhaften Umsetzung** der Spezifikation
  - in der Regel durch den Programmierer, Architekten, ...
  - Ursprung: Anforderungserhebung, Entwurf, Implementierung, ...
    - betroffen ist der komplette Zyklus der Softwareerstellung
- sind **systematische Fehler**
  - Betreibt man mehrere Instanzen **derselben Softwareversion**
  - mehrfach unter **identischen, äußeren Bedingungen**,
    - ~ werden **alle Instanzen dieselben beobachtbaren Fehler** zeigen.
    - die äußeren Bedingungen sind allerdings nicht ohne Weiteres reproduzierbar
    - vgl. Trägerrakete Ariane 5 [7]: Ausfall von SRI 1 und SRI 2 aufgrund derselben Softwarefehlers



## Software will gepflegt werden!

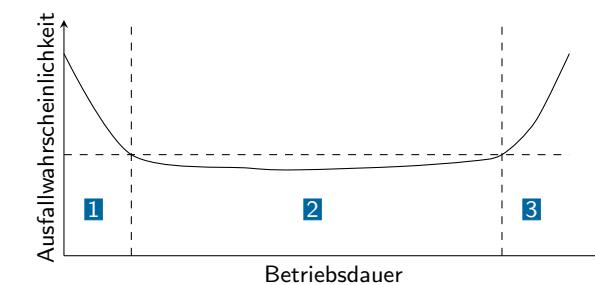
Anforderungen an langlebige Softwaresysteme unterliegen ständigem Wandel

- Folgender Zusammenhang ist einfach interessant
  - Hier wird explizit **keine Kausalbeziehung** aufgestellt!
- Chou, SOSP 2001 [1]: 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
- Padolieau, EuroSys 2006 [10]: 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 [6]: Welche Rolle spielt Refactoring?
  - ein Ergebnis: nach einem Refactoring gibt es mehr Fehlerbehebungen
    - Fehler durch **fehlerhaftes Refactoring**, Refactoring **für** die Fehlerbehebung



## Hardwarefehler

- permanente Hardwarefehler sind ...
  - extrinsischer Natur: herstellungsbedingte Materialfehler
    - z. B. fehlerhafte Dotierung eines Halbleiters oder Materialunreinheiten
    - treten meist zu Beginn der Lebenszeit auf ( $\sim$  Säuglingssterblichkeit)
  - intrinsischer Natur: Verschleißerscheinungen
    - kündigen sich meist durch sporadische Fehler an
    - treten meist am Ende der Lebenszeit auf
- Umwelteinflüsse verursachen transiente Hardwarefehler
  - mannigfaltige Ursachen
    - radioaktive Strahlung
    - elektromagnetische Interferenz
    - instabile Spannungsversorgung
    - Fertigungsstreuung bei einzelnen Transistoren
    - Temperaturschwankungen führen zum temporären Materialdefekten
    - ...
  - treten als schwer zu fassende „Bitkipper“ in Erscheinung



- 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

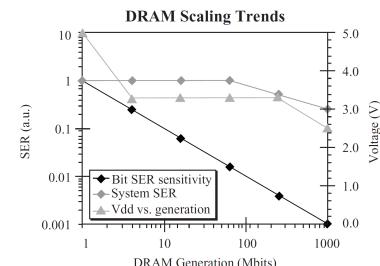
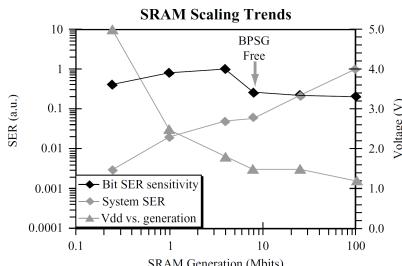
- Bitkipper durch Umladungen in Speicherzellen und Schaltkreisen
  - ☞ verursacht durch ionisierende Strahlung
    - Alphateilchen aus kontaminierten Chipgehäusen oder Lötkegeln
      - das waren „die ersten transienten Fehler“ [9, Kapitel 1.1]
      - direkte Erzeugung transienter Fehler durch Erzeugung von Elektronen und Löchern (engl. *holes*), die sich nicht rekombinieren
    - Neutronen aus kosmischer Strahlung
      - primäre kosmische Strahlung galaktische und solare Partikel
      - sekundäre kosmische Strahlung entsteht durch Wechselwirkung primärer Strahlung mit Atomen aus der Erdatmosphäre
      - terrestrische kosmische Strahlung bezeichnet die Partikel kosmischer Strahlung, die schließlich die Erdoberfläche erreichen
- Verfälschung von Kommunikation auf Bussen
  - ☞ verursacht durch elektromagnetische Interferenz  $\sim$  Rauschen
    - z. B. in Automobilen gibt es verschiedene Quellen für Wechselfelder
      - elektronischer Anlasser, Lichtmaschine, ...
    - eine sparsame elektronische Abschirmung macht dies zum Problem



## Anfälligkeit eines Schaltkreises für transiente Fehler

- die transiente Fehlerrate (engl. *soft-error rate, SER*) eines Schaltkreises hängt (stark vereinfacht) von folgenden Faktoren ab:
$$SER = C \times \text{Neutronenfluss} \times \text{Fläche} \times e^{-Q_{crit}/Q_{coll}}$$
  - **C** prozess- und schaltkreisspezifische Konstante
  - **Fläche** des Schaltkreises
  - **$Q_{crit}$**  minimale für eine Fehlfunktion notwendige Ladung
    - wird mit Hilfe von Simulationen bestimmt
  - **$Q_{coll}$**  Effizienz der Ladungsaufnahme
    - abhängig von der Dotierung und der Versorgungsspannung  $V_{CC}$
    - je größer das Bremsvermögen (engl. *stopping power*) eines Teilchens ist, desto größer ist auch  $Q_{coll}$
    - das Bremsvermögen beschreibt die Energie, die ein Teilchen auf einer bestimmten Wegstrecke an die umliegende Materie abgeben kann
- kleinere Halbleiterstrukturen sind Fluch und Segen zugleich
  - ☞ kleinere Fläche  $\sim$  kleinere SER
  - ☞ kleinere  $Q_{crit}$   $\sim$  größere SER





- ☞ pro Bit: Fehlerrate wurde gehalten (SRAM) oder reduziert (DRAM)
  - SRAM  $\sim$  die Fläche dominiert die kritische Ladung  $Q_{crit}$ 
    - deutet auf das Erreichen eines Sättigungsbereichs hin
    - $\sim$  jede elektrische Beeinflussung bedeutet einen Bitkipper
  - DRAM  $\sim$  signifikante technische Verbesserungen
- ☞ System: Fehlerrate erhöhte sich (SRAM) oder blieb gleich (DRAM)
  - was einfach an der Vielzahl von SRAM- und DRAM-Zellen liegt



## Zusammenfassung

- Fehler  $\sim$  Alles dreht sich ausschließlich um Fehler!
  - Fehlerfortpflanzung: fault  $\sim$  error  $\sim$  failure-Kette
  - permanente, sporadische und transiente Fehler
  - Vorbeugung, Entfernung, Vorhersage und Toleranz
- Verlässlichkeitsmodelle  $\sim$  Wie gut kann man mit Fehlern umgehen?
  - Verlässlichkeit, Zuverlässigkeit, Wartbarkeit und Verfügbarkeit
- Systementwurf  $\sim$  Bereits hier werden Fehler berücksichtigt!
  - Gefahren-, Risiko- und Fehlerbaumanalyse
- Software- vs. Hardwarefehler  $\sim$  Klassifikation & Ursachen
  - Softwarefehler  $\mapsto$  permanente Defekte, Komplexität
  - Hardwarefehler  $\mapsto$  permanente & transiente Fehler, Fertigung, ionisierende Strahlung, elektromagnetische Interferenz



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



## Literaturverzeichnis

- [1] 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
- [2] DEUTSCHES INSTITUT FÜR NORMUNG:  
*Fehlerbaumanalyse; Handrechenverfahren zur Auswertung eines Fehlerbaumes*. Berlin, Wien, Zürich : Beuth-Verlag, 1990 (DIN 25424)
- [3] DEUTSCHES INSTITUT FÜR NORMUNG:  
*Qualitätsmanagement - Begriffe*. Berlin, Wien, Zürich : Beuth-Verlag, 1995 (DIN 8402)
- [4] IFIP:  
*Working Group 10.4 on Dependable Computing and Fault Tolerance*. <http://www.dependability.org/wg10.4>, 2003
- [5] INTERNATIONAL ORGANIZATION FOR STANDARDIZATION:  
*Part 3: Concept phase*. Genf, Schweiz : International Organization for Standardization, 2011 (ISO 26262: Road vehicles – Functional safety)



- [6] 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 33nd International Conference on Software Engineering (ICSE '11)*. New York, NY, USA : ACM Press, Mai 2011. – ISBN 978-1-4503-0445-0, S. 151–160
- [7] 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
- [8] MARAIA, V. :  
*The Build Master: Microsoft's Software Configuration Management Best Practices*. Addison-Wesley. – ISBN 978-0321332059



- [9] MUKHERJEE, S. :  
*Architecture Design for Soft Errors*. San Francisco, CA, USA : Morgan Kaufmann Publishers Inc., 2008. – ISBN 978-0-12-369529-1
- [10] PADOLEAU, 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

