Verlässliche Echtzeitsysteme

Zusammenfassung

Wintersemester 2024/25

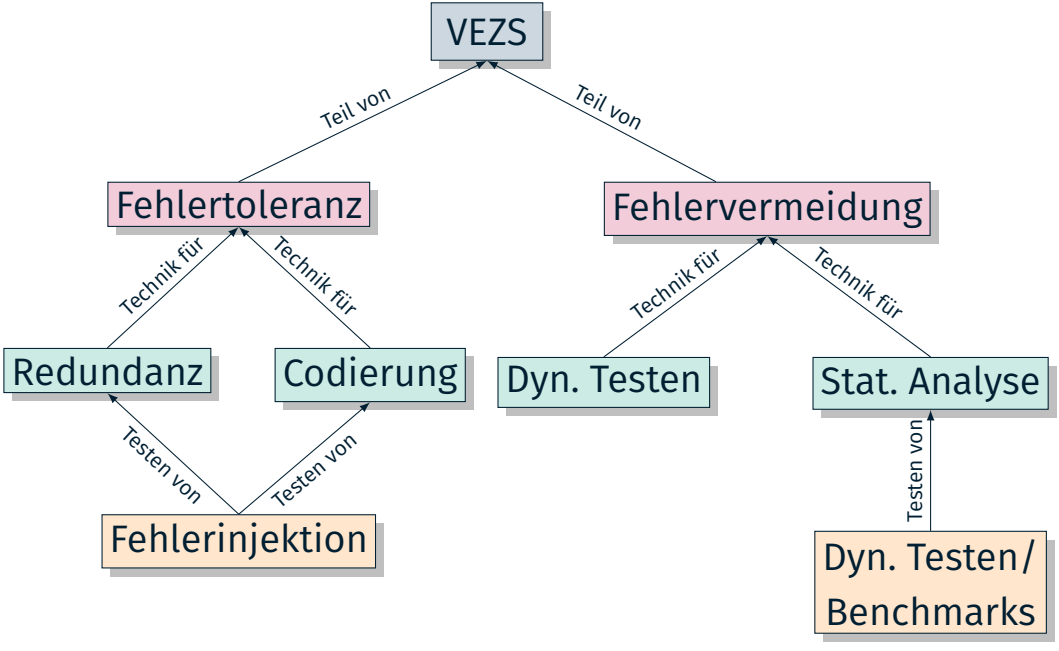
Peter Wägemann

Lehrstuhl für Systemsoftware

Friedrich-Alexander-Universität Erlangen-Nürnberg

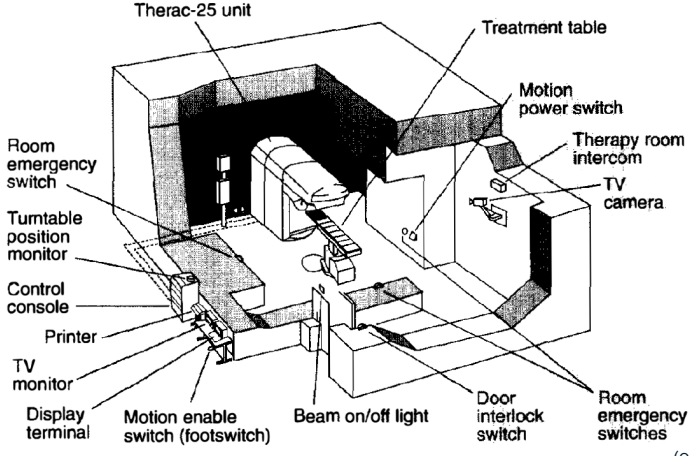
https://sys.cs.fau.de

Inhaltliche Orientierung



Einleitung

⚠ Der **Fehlerfall** verlässlicher Echtzeitsystem übersteigt die Kosten des Normalfalls um Größenordnungen ⇝ Beispiel: Therac 25

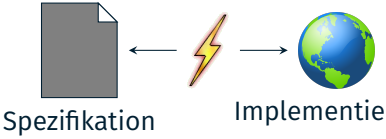


(Quelle: Nancy Leveson)

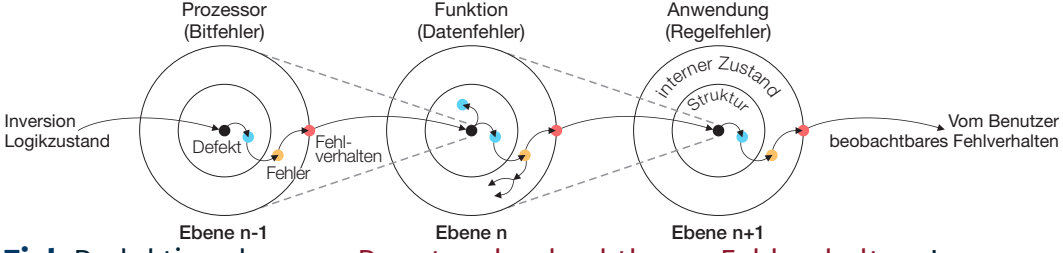
☞ **Ziel:** zuverlässiger Betrieb, minimierte Ausfallwahrscheinlichkeit

Grundlagen

- **Fokus:** Wir kümmern uns ausschließlich um Fehler!
- Fehler bedeuten eine Abweichung von der Spezifikation



- Fehler breiten sich aus und führen zu beobachtbarem Fehlerverhalten



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

Grundlagen (Forts.)

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

- Fehlerfortpflanzung: fault $\rightsquigarrow$ error $\rightsquigarrow$ failure-Kette
- Permanente, sporadische und transiente Fehler
- Vorbeugung, Entfernung, Vorhersage und Toleranz

Verlässlichkeitsmodelle $\rightsquigarrow$ Wie gut kann man mit Fehlern umgehen?

- Verlässlichkeit, Zuverlässigkeit, Wartbarkeit und Verfügbarkeit

Systementwurf $\rightsquigarrow$ Bereits hier werden Fehler berücksichtigt!

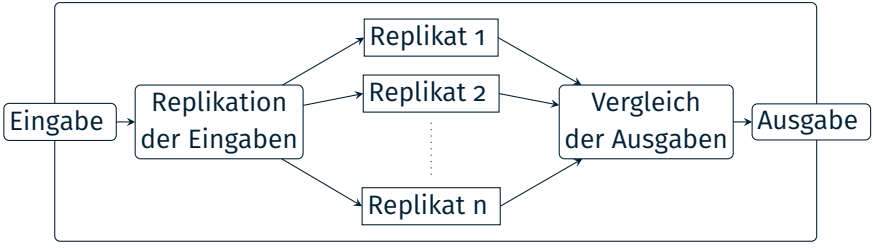
- Gefahren-, Risiko- und Fehlerbaumanalyse

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

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

Redundante Ausführung

- Fehlertoleranz erfordert Redundanz
  - Redundanz in der Struktur, Funktion, Information oder Zeit
- Ausnutzung struktureller Redundanz $\rightsquigarrow$ Replikation
  - Replikation der Eingaben, Abstimmung der Ausgaben
  - Fehlererkennung durch **Relativtest**
  - Zeitliche und räumliche Isolation einzelner Replikate



- Replikdeterminismus
  - Einigung über die Eingabewerte $\mapsto$ Akzeptanzmaskierer
  - Deterministische Umsetzung der Funktion

Redundante Ausführung (Forts.)

Fehlertypen ↦ SDCs und DUEs

Kritische Bruchstellen ↦ Bereiche ohne Redundanz

Hardwarebasierte Replikation ↦ TMR

- **{hot, warm, cold} standby**
- Dreifache Auslegung, toleriert Fehler im Wertbereich
- Zuverlässigkeit von Replikat und Gesamtsystem

Process Level Redundancy ↦ „TMR in Software“

- Reduziert Kosten von TMR, zulasten eines geringeren Schutzes

Diversität ↦ versucht **Gleichtaktfehler** auszuschließen

Härtung von Code & Daten

Fehlererkennung $\mapsto$ Durch Codierung

- $\rightsquigarrow$ Einsatz von **Informationsredundanz** durch Prüfbits
  - Fehlererkennung durch **Akzeptanztest** (Absoluttest)

AN-Codierung $\mapsto$ Codierung von Berechnungen

- Codierung: Multiplikation mit einem konstanten Faktor A
- (nicht-)systematisch und (nicht-)separiert
- Codierte Addition, Subtraktion, Multiplikation, Division
- Aussagenlogik, Schiebeoperatoren, Fließkommaarithmetik

ANBD-Codierung $\mapsto$ Erweitert die AN-Codierung

- Um statische Signaturen und dynamische Zeitstempel
- $\rightsquigarrow$ Fehlererfassung: Operanden-, Berechnungs- & Operatorfehlern
- Codierung des Kontrollflusses $\rightsquigarrow$ Signaturen für Grundblöcke

CoRed-Ansatz $\mapsto$ ANBD-Codierung der Replikationsinfrastruktur

- durchgehende arithmetische Codierung wäre zu teuer

Härtung von Code & Daten (Forts.)

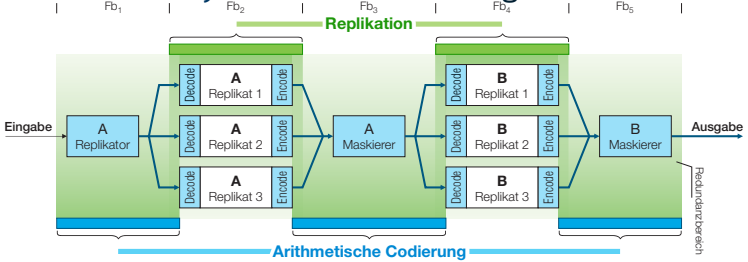
- ANBD-Codierung härtet Daten und Kontrollfluss
  - Operanden-, Berechnungs- und Operatorfehler

$$v_c = Av + B_v + D; \quad A > 1 \wedge B_v + D < A$$

    - Signatur B_v und Zeitstempel D

  ⇝ **Nachteil:** enorm hohe Laufzeitkosten

☞ „Combined Redundancy“ ⇝ ANBD-Codierung **selektiv anwenden**



  - Sichert den „single point of failure“ replizierter Ausführung
    ⇝ Codierte Implementierung des Mehrheitsentscheids

Fehlerinjektion

- Verifikation von Fehlertoleranzimplementierungen
  - Durch das **gezielte Einbringen von Fehlern**

☞ Der Kreis schließt sich

- Evaluation der Fehlertoleranz ist im Produktivbetrieb nicht möglich

  - Der durch Fehler verursachte Schaden ist nicht hinnehmbar
  - Das Auftreten von Fehlern ist nicht deterministisch/reproudzierbar

Fehlerinjektion (Forts.)

FARM-Modell Für Fehlerinjektion

- Fault, Activation, Readout, Measure
- Auswahl, Ausführung, Beobachtung, Auswertung
- Abstraktionsebenen – axiomatisch, empirisch, physikalisch
- Genereller Aufbau und Ablauf von Fehlerinjektionswerkzeugen

Fehlerinjektionstechniken ↦ grundlegende Kategorisierung

- {hardware, software, simulations}-basiert

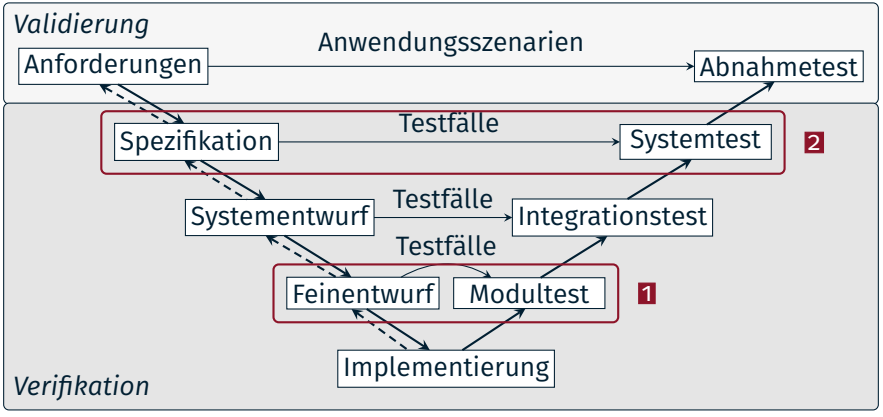
FAIL* ↦ Grundlage für generische Fehlerinjektion?

- Basierend auf virtuellen Zielsystemen
- Flexible Plattform für Fehlerinjektion
- Schnelle Experimentdurchführung durch Parallelisierung

Zuverlässigkeitsmetriken ↦ Messung und Auswertung

- Absolute Zahlen versus Fehlerwahrscheinlichkeit

Testen



1. Modultests ⇝ Grundbegriffe und Problemstellung
   → Black- vs. White-Box, Testüberdeckung
2. Systemtest ⇝ Testen verteilter Echtzeitsysteme
   → Problemstellung und Herausforderungen

Testen (Forts.)

⚠ Testen: weit verbreitet in der Praxis

- Modul-, Integrations-, System- und Abnahmetest

→ Kann die Absenz von Defekten aber nie garantieren

- *Modultests*
  - Black-Box- vs. White-Box-Tests
  - McCabe's Cyclomatic Complexity ⇝ Minimalzahl von Testfällen
  - Kontrollflussorientierte Testüberdeckung
    - Anweisungs-, Zweig-, Pfad- und Bedinungsüberdeckung
    - Angaben zur Testüberdeckung sind immer relativ!
- *Systemtests* für verteilte Echtzeitsysteme sind herausfordernd!
  - Problemfeld: Testen verteilter Echtzeitsysteme
    - SW-Engineering, verteilte Systeme, Echtzeitsysteme
    - Probe-Effect, Beobachtbarkeit, Kontrollierbarkeit, Reproduzierbarkeit

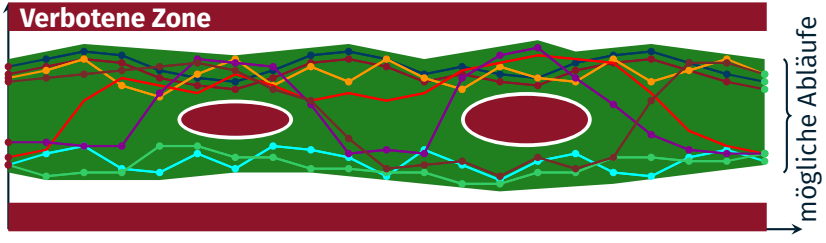
Abstrakte Interpretation

- Enthält das Programm Laufzeitfehler?
  - Ganzzahl- oder Fließkommaüberläufe, nicht-initialisierte Variablen, …
  - Können wir diese Frage vor der Laufzeit beantworten?

⚠ Für die konkrete Programmsemantik geht das nicht

- Eine sichere Abstraktion könnte für diesen Zweck aber ausreichen
  - Für Zugriffe auf Felder ist nur der möglichen Wertebereich des Index wichtig
  - Welcher konkrete Wert wann angenommen wird, ist nicht von Belang

☞ Einsatz einer abstrakten Programmsemantik



Abstrakte Interpretation (Forts.)

- Die *abstrakte Semantik* stellt eine Approximation dar
  - Korrektheit (Vollständigkeit) ist entscheidend
    - Nur so kann man einen Sicherheitsnachweis führen
  - Die Approximation muss präzise sein
    - Nur so kann man Fehlalarme vermeiden
  - Gleichzeitig eine geringe Komplexität aufweisen
    - Nur so kann sie effizient berechnet werden

  → Abstraktion und Konkretisierung implizieren keinen Sicherheitsverlust!

- Analyse und Vereinfachung
  - Pfadsemantiken beschreiben die konkrete Programmsemantik
  - Approximation durch Pfadpräfixe und Sammelsemantik

Der Stapelspeicher (Stack)

In eingebetteten Systemen typischerweise die einzige Form dynamischen Speichers

- *Überabschätzung* führt zu unnötigen Kosten

⚠ *Unterabschätzung* des Speicherverbrauchs führt zu Stapelüberlauf

- Schwerwiegendes und komplexes Fehlermuster
- Undefiniertes Verhalten, Datenfehler oder Programmabsturz

→ Schwer zu finden, reproduzieren und beheben!

☞ Messbasierter Ansatz (Die Praxis!!)

- Water-Marking, Überwachung zur Laufzeit

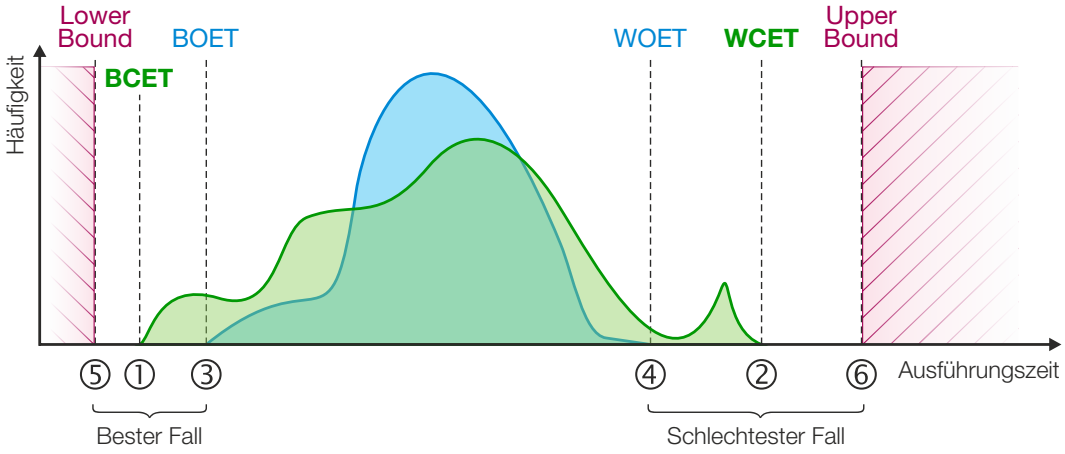
→ Keine Aussagen zum maximalen Verbrauch

☞ Statische Programmanalyse

- Pufferüberlauf als weitere Form von Laufzeitfehlern

→ Bestimmt obere Schranke für den Speicherverbrauch

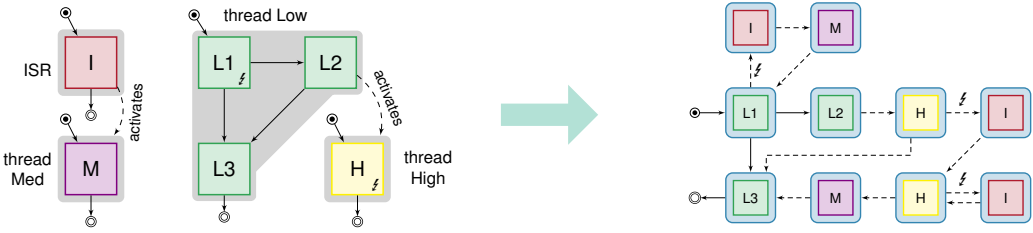
Laufzeitanalyse



- Messbasierte Laufzeitbestimmung $\rightsquigarrow$ Beobachtung
- Statische WCET-Analyse $\rightsquigarrow$ obere/untere Schranke
  - Zu finden: Längster Pfad (Timing Schema, Zeitanalysegraph)
  - Dauer der Elementaroperationen: Hardware-Analyse

  $\rightarrow$ Die Analyse ist sicher (sound) falls Upper Bound $\geq$ WCET

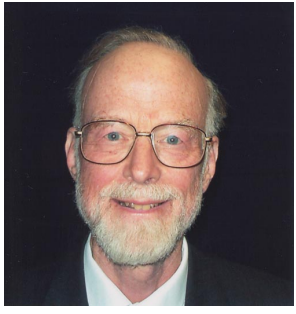
Systemweite Analysen (Stackbedarf, Antwortzeit)



- Transitionen müssen **systemweite Kontrollflüsse** beinhalten
  - Interrupt $\rightsquigarrow$ mehr Speicherbedarf
  - höchpriore Aufgabe $\rightsquigarrow$ mehr Laufzeit
- Stackverbrauch: Interrupt-Preemption-Graph
- Laufzeit: State-Transition-Graph
- Bestimmung des Ressourcenbedarfs: z.B. Tiefensuche, ILP

Design-by-Contract

- Überprüfung benutzerdefinierter Korrektheitsbedingungen
  - Angabe als Vor- und Nachbedingungen $\leadsto$ „Design by Contract“
- Hoare-Kalkül/WP-Kalkül $\leadsto$ denotationelle Semantik
  - Schließt die Brücke zwischen Vertrag und Implementierung



C.A.R. Hoare

Edger W. Dijkstra

Design-by-Contract (Forts.)

Funktionale Programmeigenschaften $\mapsto$ Zusicherungen

- Vorbedingungen, Nachbedingungen und **Invarianten**
- Beschrieben durch Ausdrücke der Prädikatenlogik

Prädikatentransformation $\rightsquigarrow$ symbolische Ausführung

- Bildet Semantik durch Transformation von Zusicherungen nach
- Strongest postcondition, weakest precondition

Hoare-Kalkül $\rightsquigarrow$ deduktive Ableitung von Nachbedingungen

- Hoare-Tripel, Axiome für leere Anweisungen und Zuweisungen
- Ableitungsregeln für Sequenzen, Verzweigungen und Iterationen
- Konsequenzregel passt Vor-/Nachbedingungen an

WP-Kalkül $\mapsto$ „Hoare-Kalkül rückwärts“

Praxisbezug $\rightsquigarrow$ Astreé implementiert dieses Konzept nur teilweise!

Fallstudie: Sizewell B

- Wie werden echte verlässliche Echtzeitsysteme entwickelt?
  - Wie wird die Korrektheit von Software sichergestellt?
  - Welche Laufzeitfehler sind insbesondere von Belang?
  - Welche Fehlertoleranzmechanismen werden implementiert?

☞ Betrachtung am Beispiel des primären Reaktorschutzsystems (PPS) des Sizewell B Kernkraftwerks

Fallstudien (Forts.)

Sizewell B $\rightsquigarrow$ primäres Reaktorschutzsystem
- Einziger Zweck: sichere Abschaltung des Reaktors

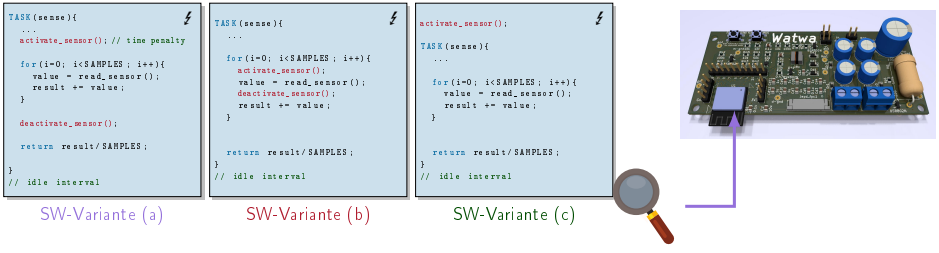
Redundanz $\rightsquigarrow$ Absicherung gegen Systemausfälle
- Vierfach

Diversität $\rightsquigarrow$ Abfedern von Software-Defekten
- Unterschiedliche Hardware und Software
- Analoges Sekundärsystem

Isolation $\rightsquigarrow$ Abschottung der einzelnen Replikate
- Technisch $\mapsto$ optische Kommunikationsmedien
- Zeitlich $\mapsto$ nicht-gekoppelte, eigenständige Rechner
- Räumlich $\mapsto$ verschiedene Aufstellorte und Kabelrouten

Verifikation $\rightsquigarrow$ umfangreiche, statische Prüfung von Software
- Vielschichtiger Prozess, Betrachtung von Quell- und Binärcode

{B, M}-Arbeiten … Promotion



- Überblick der Forschung am Lehrstuhl: `https://sys.cs.fau.de/research`
- Themen an der Hardware/Software Schnittstelle
- Thema zu Echtzeitsystemen: `https://sys.cs.fau.de/research/watwa`


The End