

Verlässliche Echtzeitsysteme

Zusammenfassung

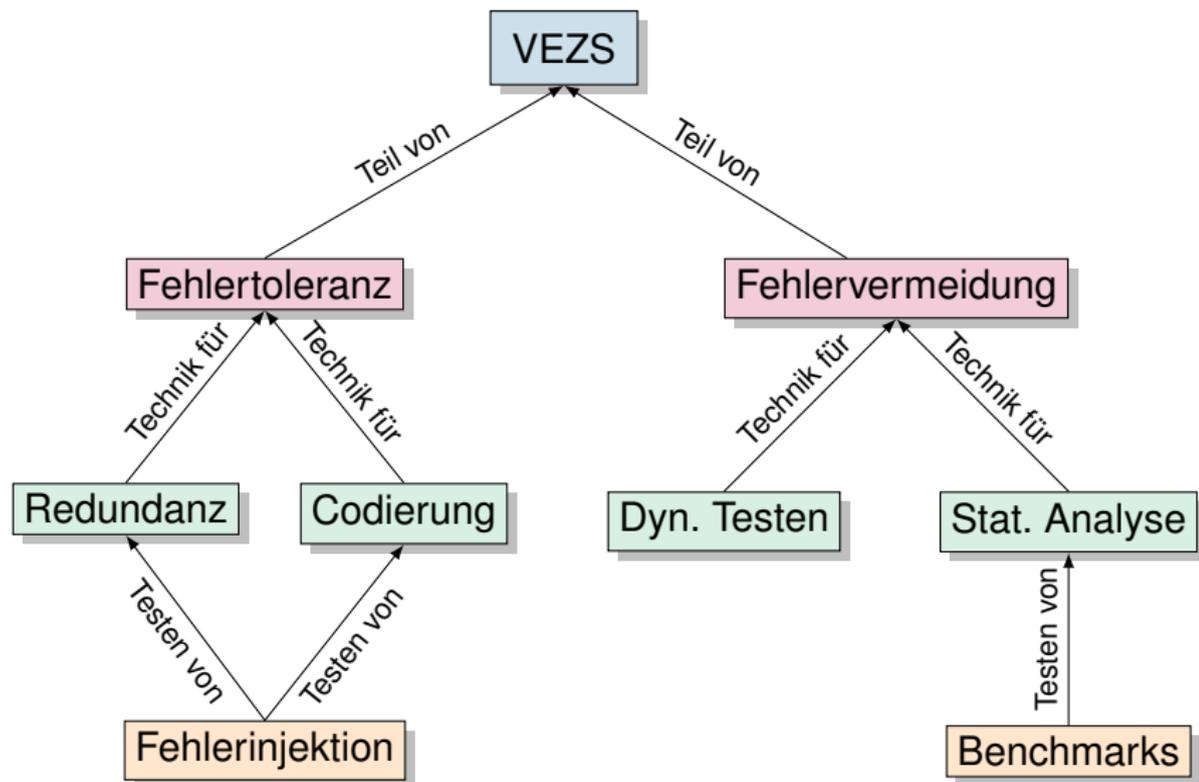
Peter Wägemann

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

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

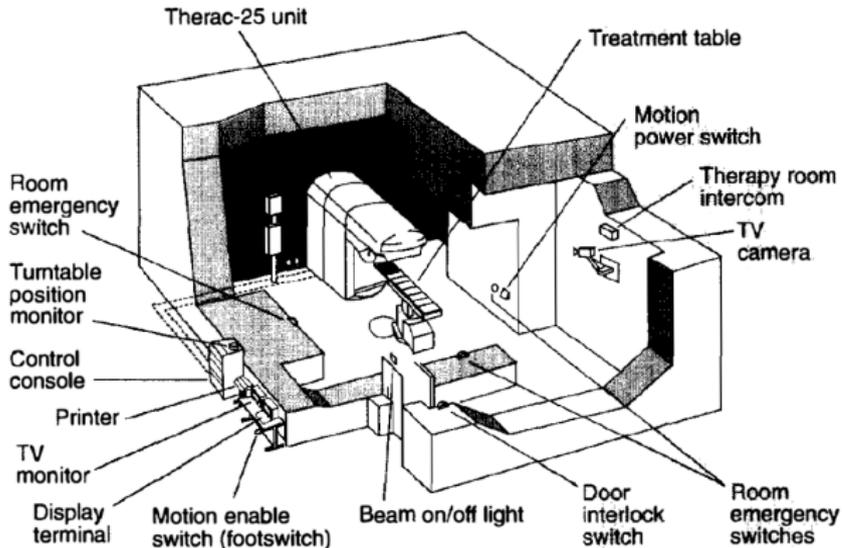
KW05 2023







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



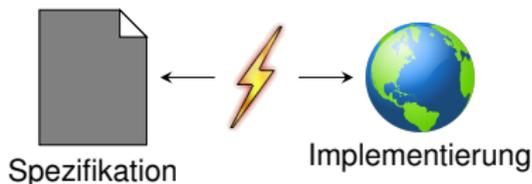
(Quelle: Nancy Leveson)



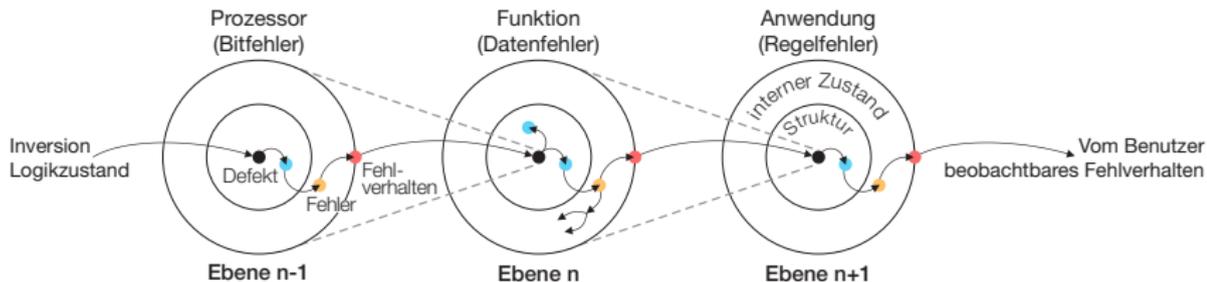
Ziel: zuverlässiger Betrieb, minimierte Ausfallwahrscheinlichkeit



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



- Fehler breiten sich aus und führen zu **beobachtbarem Fehlverhalten**



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



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

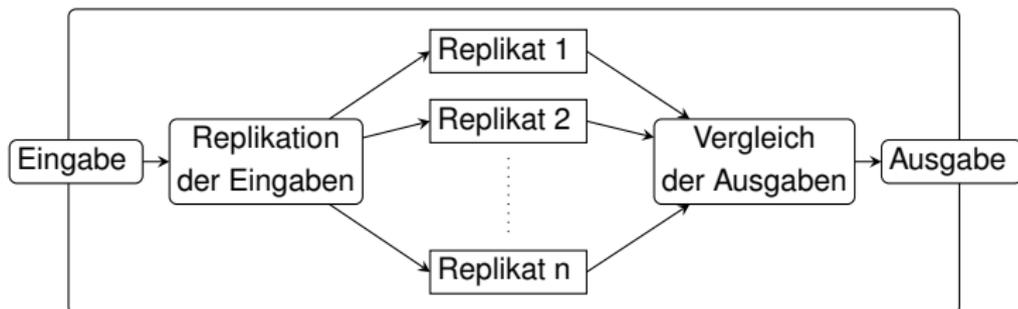
Software- vs. Hardwarefehler \rightsquigarrow Klassifikation & Ursachen

- 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 \leadsto **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



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



Fehlererkennung \mapsto Durch Codierung

- \leadsto 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
- \leadsto Vollständige Fehlererfassung von Operanden-, Berechnungs- und Operatorfehlern
- Codierung des Kontrollflusses \leadsto Signaturen für Grundblöcke

CoRed-Ansatz \mapsto ANBD-Codierung der Replikationsinfrastruktur

- Durchgehende arithmetische Codierung wäre zu teuer



Härtung von Code & Daten (Forts.)

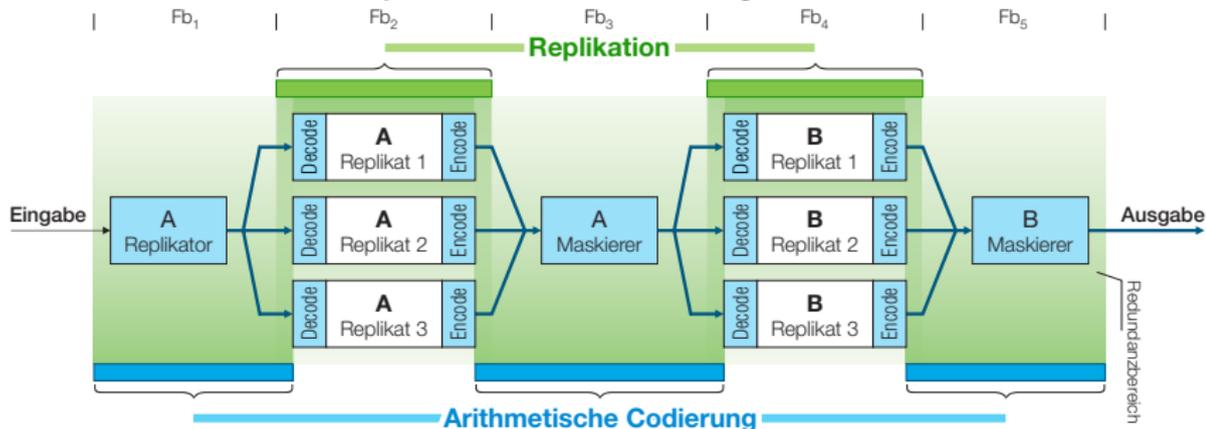
- ANBD-Codierung härtet Daten und Kontrollfluss
 - Operanden-, Berechnungs- und Operatorfehler

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

- 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/reproduzierbar



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

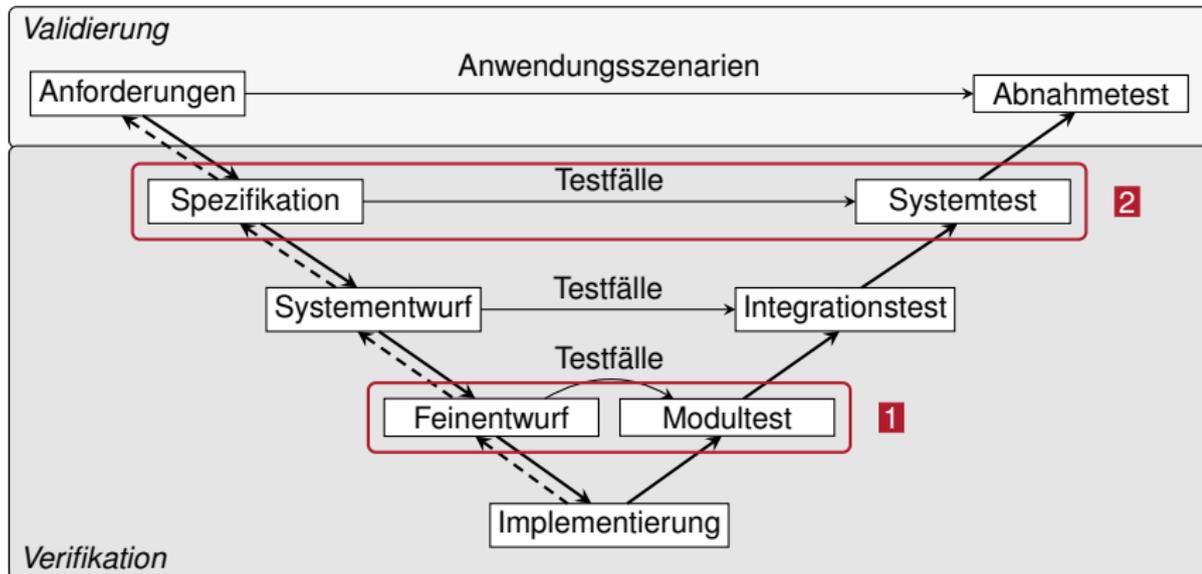
FAIL* → Grundlage für generische Fehlerinjektion?

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

Zuverlässigkeitsmetriken → Messung und Auswertung

- Absolute Zahlen versus Fehlerwahrscheinlichkeit





1 **Modultests** \leadsto Grundbegriffe und Problemstellung

\rightarrow Black- vs. White-Box, Testüberdeckung

2 **Systemtest** \leadsto Testen verteilter Echtzeitsysteme

\rightarrow Problemstellung und Herausforderungen





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 \leadsto Minimalzahl von Testfällen
- Kontrollflussorientierte Testüberdeckung
 - Anweisungs-, Zweig-, Pfad- und Bedingungsüberdeckung
 - 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?

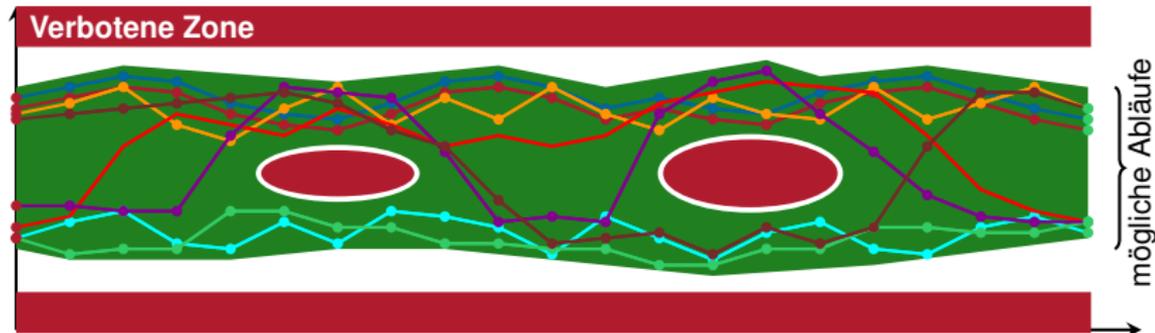


Für die **konkrete Programmsemantik** geht das nicht

- 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**



- 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

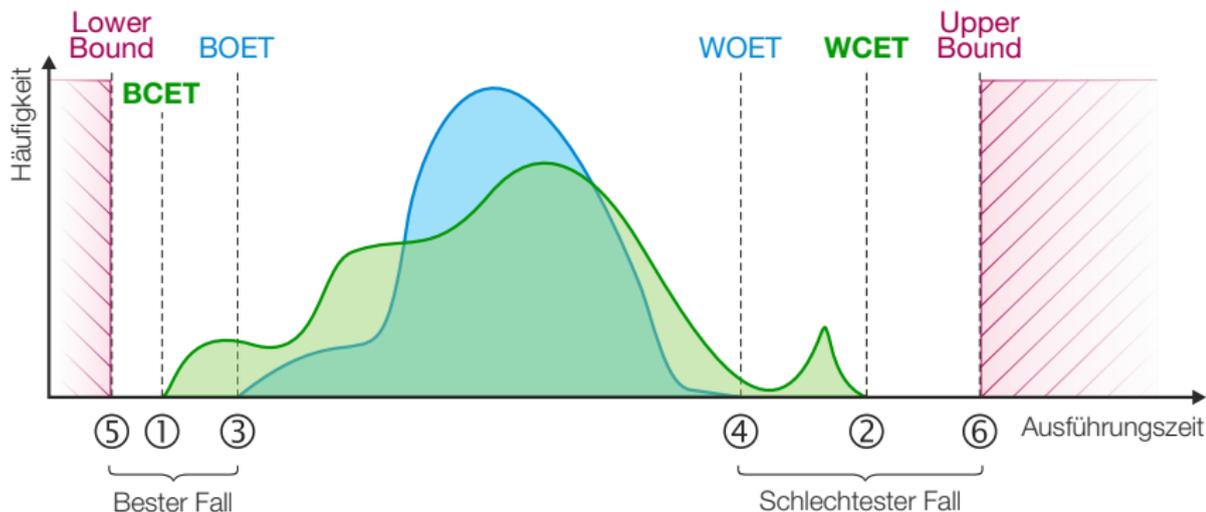


Statische Programmanalyse

- Pufferüberlauf als weitere Form von Laufzeitfehlern
- Bestimmt obere Schranke für den Speicherverbrauch



Die Laufzeit



- Messbasierte Laufzeitbestimmung \leadsto Beobachtung
 - Statische WCET-Analyse \leadsto 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



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



C.A.R. Hoare



Edger W. Dijkstra

Funktionale Programmeigenschaften \mapsto Zusicherungen

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

Prädikamentransformation \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 Astaré implementiert dieses Konzept nur teilweise!



■ 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



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

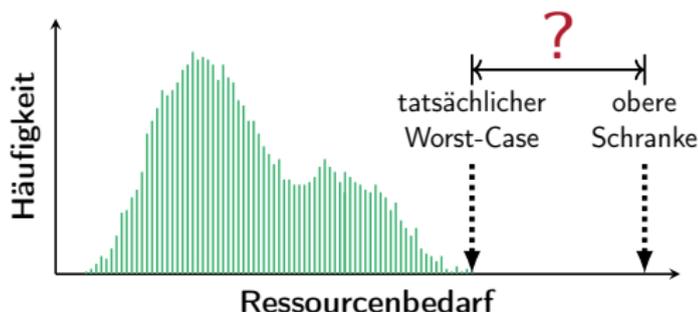
Isolation \rightsquigarrow Abschottung der einzelnen Replikate

- 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





Ist Compiler *korrekt*? Sind Analysen *korrekt & präzise*?

- Verifizierter Compiler [2] nicht immer verfügbar
- Wissen über Korrektheit schwer zu garantieren
- **Strukturierte Testverfahren** als Möglichkeit Qualität zu verbessern
 - Generierung von **Referenz** analog zu Fehlerinjektion
 - Generierung **geeigneter Testmuster**/Benchmarks
 - Für Bug-Analysatoren [1]
 - Für WCET-Analysatoren [3]

1 Zusammenfassung

- Einleitung
- Grundlagen
- Redundante Ausführung
- Härtung von Daten- und Kontrollfluss
- Fehlerinjektion
- Testen
- Statische Programmanalyse
- Statische Analyse nicht-funktionaler Eigenschaften
- Statische Analyse funktionaler Eigenschaften
- Fallstudie: Sizewell B
- Benchmarking: Testen von Analysen

2 Abschlussarbeiten



{B, M}-Arbeiten ... Promotion

Forschungs-/Entwicklungsprojekte: Universität, Forschungseinrichtungen, Industrie

```
task {name}() {
  ...
  while_true { // time penalty
    ...
  }
  for {i=0; i<SURPLUS; i++}
  value = read_sensor();
  result += value;
}
end_while_true();
return result/SURPLUS;
} // 100k iterations
```

SW-Variante (a)

```
task {name}() {
  ...
  for {i=0; i<SURPLUS; i++}
  while_true {
    value = read_sensor();
    result += value;
  }
  return result/SURPLUS;
} // 100k iterations
```

SW-Variante (b)

```
while_true {name}();
task {name}() {
  ...
  for {i=0; i<SURPLUS; i++}
  value = read_sensor();
  result += value;
}
return result/SURPLUS;
} // 100k iterations
```

SW-Variante (c)



- 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>



- [1] Kashyap, V. ; Ruchti, J. ; Kot, L. ; Turetsky, E. ; Swords, R. ; Pan, S. A. ; Henry, J. ; Melski, D. ; Schulte, E. :
Automated customized bug-benchmark generation.
In: Proceedings of the 19th International Working Conference on Source Code Analysis and Manipulation (SCAM '19), 2019, S. 103–114
- [2] Leroy, X. ; Blazy, S. ; Kästner, D. ; Schommer, B. ; Pister, M. ; Ferdinand, C. :
CompCert – A Formally Verified Optimizing Compiler.
In: Proceedings of the 8th European Congress on Embedded Real Time Software and Systems (ERTS '16), 2016
- [3] Wägemann, P. ; Distler, T. ; Eichler, C. ; Schröder-Preikschat, W. :
Benchmark Generation for Timing Analysis.
In: Proceedings of the 23rd Real-Time and Embedded Technology and Applications Symposium (RTAS '17), 319–330