

# Verlässliche Echtzeitsysteme

## Zusammenfassung

### Peter Ulbrich

Lehrstuhl für Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität Erlangen-Nürnberg

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

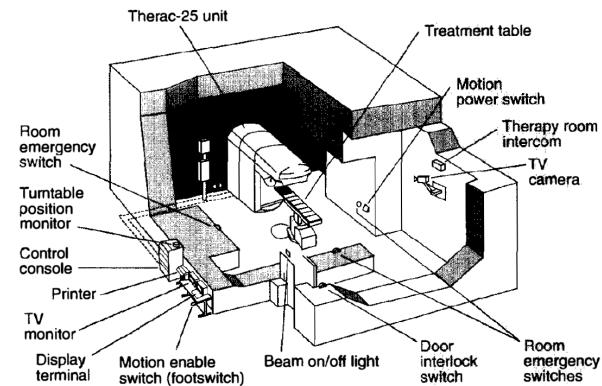
27. Juli 2017



## Einleitung



Der **Fehlerfall** verlässlicher Echtzeitsystem übersteigt die Kosten des Normalsfalls um Größenordnungen ~ Beispiel: Therac 25



(Quelle: Nancy Leveson)



**Ziel:** zuverlässiger Betrieb, minimierte Ausfallwahrscheinlichkeit



## Überblick

27. April 2017

Einleitung

Kapitel II



## Überblick

27. April 2017

Einleitung

Kapitel II

04. Mai 2017

Grundlagen

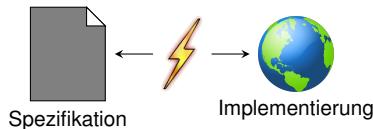
Kapitel III

Softwaredefekte ← Grundlagen → Fehlertoleranz

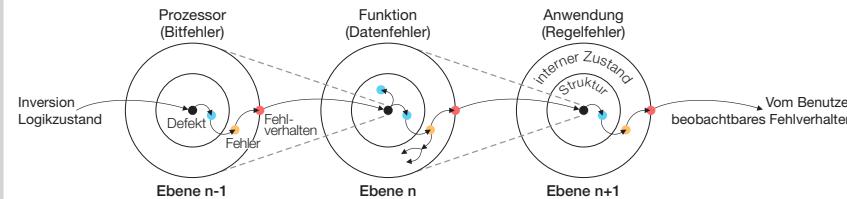


## Grundlagen

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



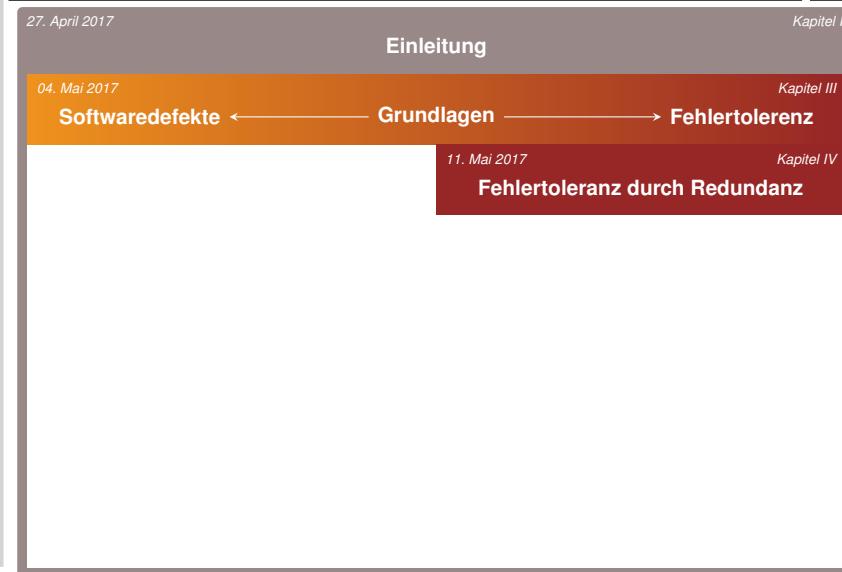
- Fehler breiten sich aus und führen zu **beobachtbarem Fehlerverhalten**



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



## Überblick



## Grundlagen (Forts.)

- **Fehler** → Alles dreht sich ausschließlich um Fehler!
- Fehlerfortpflanzung: fault → error → failure-Kette
- Permanente, sporadische und transiente Fehler
- Vorbeugung, Entfernung, Vorhersage und Toleranz

- **Verlässlichkeitsmodelle** → Wie gut kann man mit Fehlern umgehen?
- **Verlässlichkeit, Zuverlässigkeit, Wartbarkeit und Verfügbarkeit**

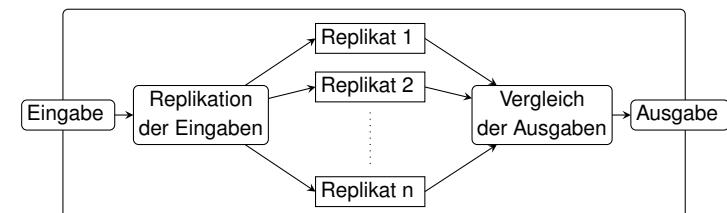
- **Systementwurf** → Bereits hier werden Fehler berücksichtigt!
- **Gefahren-, Risiko- und Fehlerbaumanalyse**

- **Software- vs. Hardwarefehler** → Klassifikation & Ursachen
- **Softwarefehler** → permanente Defekte, Komplexität
- **Hardwarefehler** → permanente & transiente Fehler, Fertigung, ionisierende Strahlung, elektromagnetische Interferenz



## Redundante Ausführung

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



- **Replikdeterminismus**
  - Einigung über die Eingabewerte → Akzeptanzmaskierer
  - Deterministische Umsetzung der Funktion



Fehlerarten → SDCs und DUEs

Kritische Bruchstellen → Bereiche ohne Redundanz

Hardwareasierte Replikation → TMR

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

Process Level Redundancy → „TMR in Software“

- Reduziert Kosten von TMR, zulasten eines geringeren Schutzes

Diversität → versucht Gleichtaktfehler auszuschließen



27. April 2017

Kapitel II

### Einleitung

04. Mai 2017

Kapitel III

### Softwaredefekte

### Grundlagen

### Fehlertoleranz

11. Mai 2017

Kapitel IV

### Fehlertoleranz durch Redundanz

18. Mai 2017

Kapitel V

### Härtung v. Daten- & Kontrollfluss



## Härtung von Code & Daten

Fehlererkennung → Durch Codierung

~ Einsatz von **Informationsredundanz** durch Prüfbits

- Fehlererkennung durch **Akzeptanztest** (Absoluttest)

AN-Codierung → 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 → Erweitert die AN-Codierung

- Um **statische Signaturen** und **dynamische Zeitstempel**
- ~ Vollständige Fehlererkennung von **Operanden-, Berechnungs- und Operatorfehlern**
- Codierung des Kontrollflusses ~ **Signaturen für Grundblöcke**

CoRed-Ansatz → ANBD-Codierung der Replikationsinfrastruktur

- Durchgehende arithmetische Codierung wäre zu teuer



## Härtung von Code & Daten (Forts.)

■ ANBD-Codierung härtet Daten und Kontrollfluss

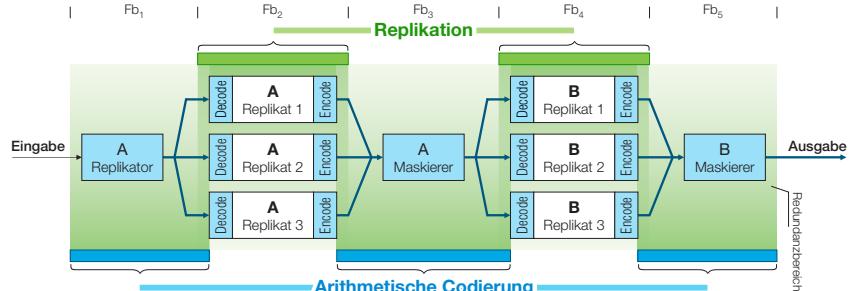
- Operanden-, Berechnungs- und Operatorfehler

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

– Signatur  $B_v$  und Zeitstempel  $D$

~ **Nachteil:** enorme hohe Laufzeitkosten

„Combined Redundancy“ ~ ANBD-Codierung selektiv anwenden



- Sichert den „single point of failure“ replizierter Ausführung

~ Codierte Implementierung des Mehrheitsentscheids



# Überblick

27. April 2017

## Einleitung

Kapitel II

04. Mai 2017

## Softwaredefekte

## Grundlagen

Kapitel III

11. Mai 2017

## Fehlertoleranz durch Redundanz

Kapitel IV

18. Mai 2017

## Härtung v. Daten- & Kontrollfluss

Kapitel V

01. Juni 2017

## Fehlerinjektion

Kapitel VI



# 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



# 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

Zuverlässigkeitssmetriken → Messung und Auswertung

- Absolute Zahlen versus Fehlerwahrscheinlichkeit



# Überblick

27. April 2017

## Einleitung

Kapitel II

04. Mai 2017

## Softwaredefekte

## Grundlagen

Kapitel III

08. Juni 2017

## Dynamisches Testen

Kapitel VIII

11. Mai 2017

Kapitel IV

## Fehlertoleranz durch Redundanz

Kapitel V

18. Mai 2017

## Härtung v. Daten- & Kontrollfluss

Kapitel V

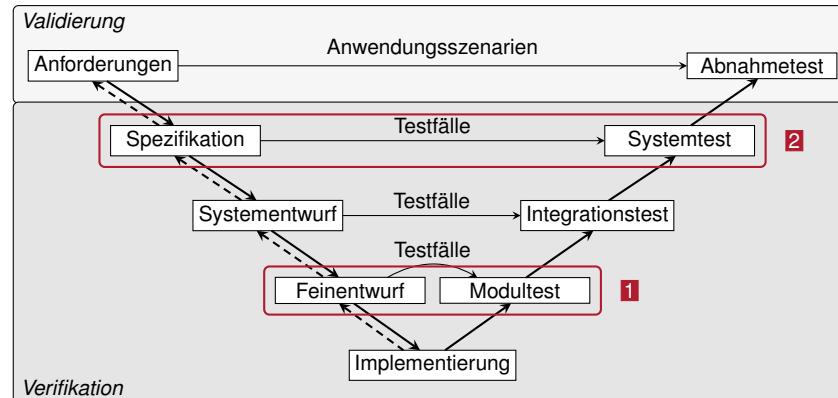
01. Juni 2017

## Fehlerinjektion

Kapitel VI



# Testen



1 Modultests  $\rightsquigarrow$  Grundbegriffe und Problemstellung

$\rightarrow$  Black- vs. White-Box, Testüberdeckung

2 Systemtest  $\rightsquigarrow$  Testen verteilter Echtzeitsysteme

$\rightarrow$  Problemstellung und Herausforderungen



# Testen (Forts.)



Testen ist die Verifikationstechnik in der Praxis!

- Modul-, Integrations-, System- und Abnahmetest
- $\rightarrow$  Kann die Absenz von Defekten aber nie garantieren

## Modultests

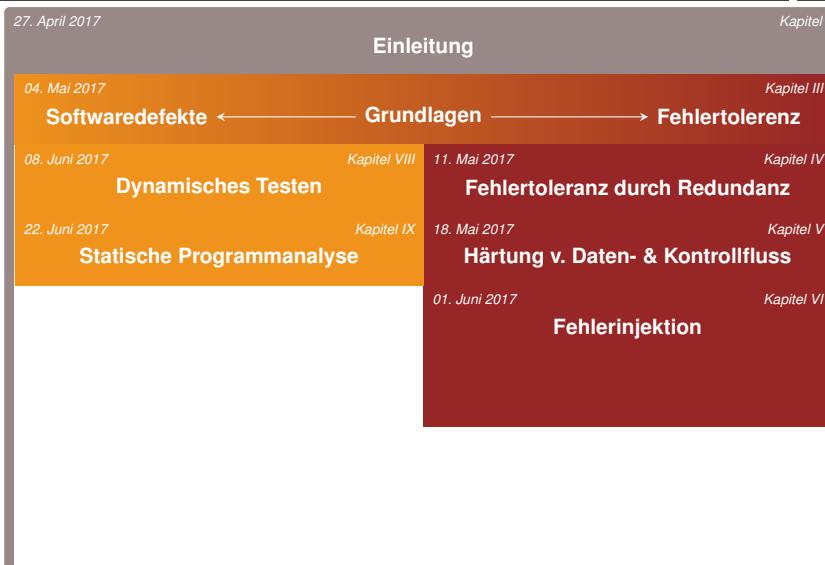
- Black-Box- vs. White-Box-Tests
- McCabe's Cyclomatic Complexity  $\rightsquigarrow$  Minimalzahl von Testfällen
- Kontrollflussorientierte Testüberdeckung
  - Anweisungs-, Zweig-, Pfad- und Bedingungsüberdeckung
  - Angaben zur Testüberdeckung sind immer relativ!

## Systemtests für verteilte Echtzeitsysteme sind herausfordernd!

- Problemfeld: Testen verteilter Echtzeitsysteme
  - SW-Engineering, verteilte Systeme, Echtzeitsysteme
  - Probe-Effect, Beobachtbarkeit, Kontrollierbarkeit, Reproduzierbarkeit



# Überblick



# Abstrakte Interpretation

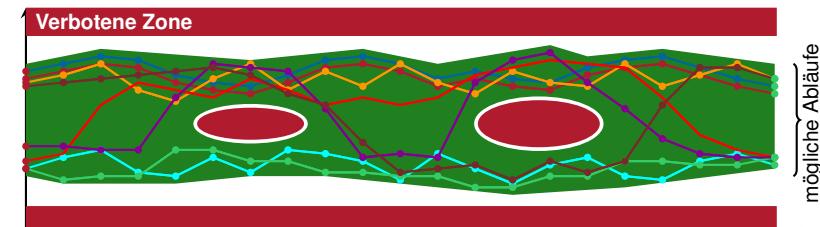
## Enthält das Programm Laufzeitfehler?

- Ganzzahl- oder Fließkommaüberläufe, nicht-initialisierte Variablen, ...
- Können wir diese Frage vor der Laufzeit beantworten?

## Für die konkrete Programmsemantik geht das nicht

- Eine sicher 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 Präzisionsverlust!
- 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
- Reaktiv ∼ Keine Aussagen zum maximalen Verbrauch
- ☞ **Statische Programmanalyse**
  - Pufferüberlauf als weitere Form von Laufzeitfehler
- Bestimmt obere Schranke für den Speicherverbrauch



### Einleitung

### Softwaredefekte ← Grundlagen → Fehlertoleranz

### Dynamisches Testen

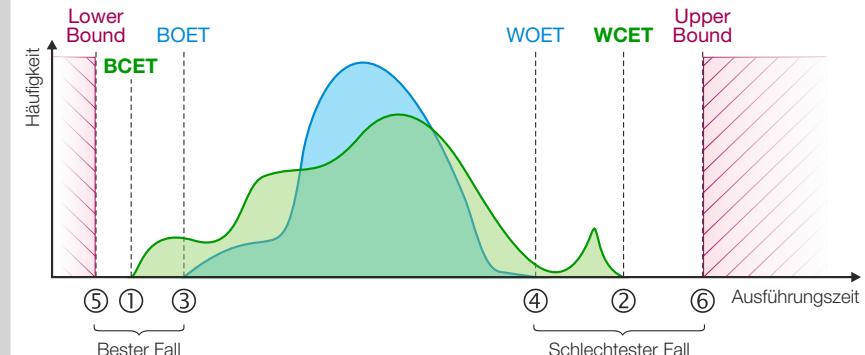
### Statische Programmanalyse

### Verifikation nicht-funkt. Eigenschaften

### Fehlerinjektion



## Die Laufzeit



- **Messbasierte Laufzeitbestimmung** ∼ Beobachtung
- **Statische WCET-Analyse** ∼ **Obere/untere Schranke**
  - Zu finden: Längster Pfad (Timing Schema, Zeitanalysegraph)
  - Dauer der Elementaroperationen: Hardware-Analyse
- Die Analyse ist **sicher** (sound) falls  $\text{Upper Bound} \geq \text{WCET}$



# Überblick

27. April 2017

Kapitel II

## Einleitung

04. Mai 2017

Kapitel III

Softwaredefekte ← Grundlagen → Fehlertoleranz

08. Juni 2017

Kapitel VIII

11. Mai 2017

Kapitel IV

Dynamisches Testen

Fehlertoleranz durch Redundanz

22. Juni 2017

Kapitel IX

Statische Programmanalyse

18. Mai 2017

Kapitel V

Härtung v. Daten- & Kontrollfluss

29. Juni 2017

Kapitel X

Verifikation nicht-funkt. Eigenschaften

01. Juni 2017

Kapitel VI

Fehlerinjektion

06. Juli 2017

Kapitel XI

Verifikation funktionaler Eigenschaften



# Design-by-Contract

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



C.A.R. Hoare



Edger W. Dijkstra



# Design-by-Contract (Forts.)

Funktionale Programmeigenschaften → Zusicherungen

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

Prädikatentransformation → symbolische Ausführung

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

Hoare-Kalkül → 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 → „Hoare-Kalkül rückwärts“

Praxisbezug → Astreé implementiert dieses Konzept nur teilweise!



# Überblick

27. April 2017

Kapitel II

Einleitung

04. Mai 2017

Kapitel III

Softwaredefekte ← Grundlagen → Fehlertoleranz

08. Juni 2017

Kapitel VIII

11. Mai 2017

Kapitel IV

Dynamisches Testen

22. Juni 2017

Kapitel IX

18. Mai 2017

Kapitel V

Statische Programmanalyse

29. Juni 2017

Kapitel X

01. Juni 2017

Kapitel VI

Verifikation nicht-funkt. Eigenschaften

06. Juli 2017

Kapitel XI

01. Juli 2017

Kapitel VII

Verifikation funktionaler Eigenschaften

20. Juli 2017

Kapitel XII

Fallstudie: Sizewell B



## Fallstudie: Sizewell B

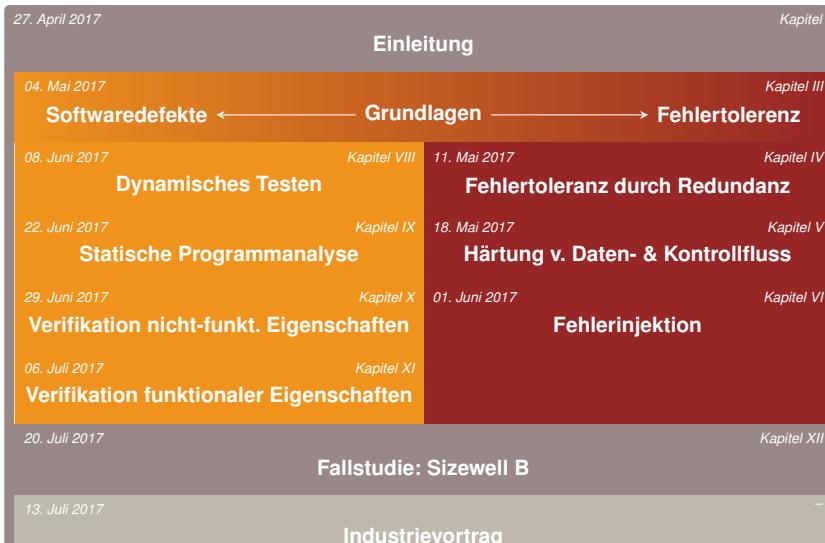
### ■ Wie werden **echte verlässliche Echtzeitsysteme** entwickelt?

- 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



## Überblick



## Fallstudien (Forts.)

**Sizewell B** ↗ primäres Reaktorschutzsystem

- Einziger Zweck: sichere Abschaltung des Reaktors

**Redundanz** ↗ Absicherung gegen Systemausfälle

- Vierfach

**Diversität** ↗ Abfedern von Software-Defekten

- Unterschiedliche Hardware und Software
- Analoges Sekundärsystem

**Isolation** ↗ Abschottung der einzelnen Replikate

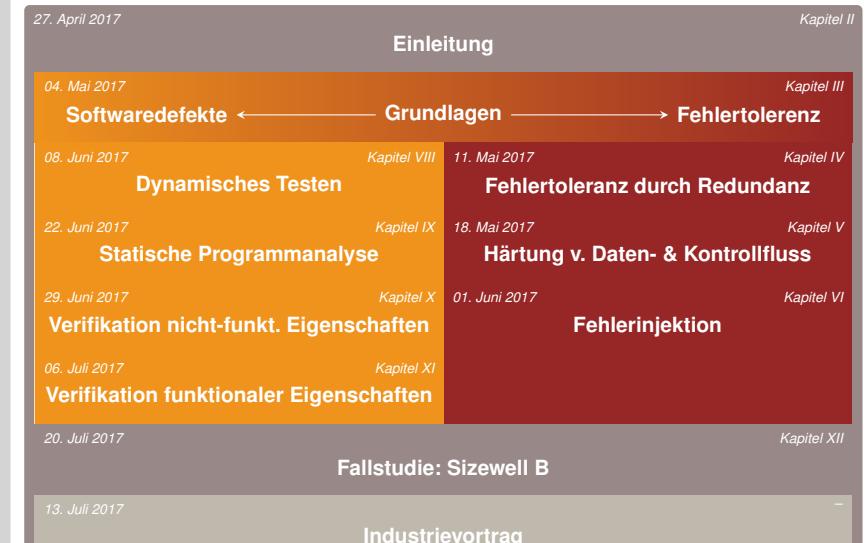
- Technisch ↗ optische Kommunikationsmedien
- Zeitlich ↗ nicht-gekoppelte, eigenständige Rechner
- Räumlich ↗ verschiedene Aufstellorte und Kabelrouten

**Verifikation** ↗ umfangreiche statische Prüfung von Software

- Vielschichtiger Prozess, Betrachtung von Quell- und Binärcode



## Überblick



## 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
- Vorträge
- Prüfungsrelevanz

## 2 Abschlussarbeiten



<https://www4.cs.uni-erlangen.de/Theses>

oder besser noch: Kommt vorbei!

