

# Verlässliche Echtzeitsysteme

---

Wintersemester 2024

Peter Wägemann

Lehrstuhl für Systemsoftware

Friedrich-Alexander-Universität Erlangen-Nürnberg

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

*Die Lehrveranstaltung ist grundsätzlich für alle Studiengänge offen. Sie verlangt allerdings gewisse Vorkenntnisse. Diese müssen nicht durch Teilnahme an den Lehrveranstaltungen von I4 erworben worden sein.*

- **Systemprogrammierung**, Grundlagen der Informatik
- **C / C++**, Java
- Ein gewisses Maß an **Durchhaltevermögen**
- Freude an systemnaher und **praktischer Programmierung**

Wir arbeiten mit eingebetteten Systemen!

Die meisten sind überrascht, wie viel Spaß das macht :-)

- 1 Vorwort
- 2 Die Veranstaltung
  - Lernziele
- 3 Organisatorisches
  - Die Beteiligten
  - Vorlesung und Übung
  - Leistungsnachweise

# Verlässliche Echtzeitsysteme – Motivation

- ☞ **Technik** (von Echtzeitsystemen) **begeistert**
  - Zusteller begrenzen den zeitlichen Einfluss
    - Nicht-periodischer Aktivitäten auf periodische Arbeitsaufträge
  - Neue Verfahren und Architekturen zu entwickeln, ist spannend!
  - Mikrokerne schotten Programme räumlich voneinander ab
  - Verschlüsselungsalgorithmen garantieren Datensicherheit
  - ...



## **Das ist jedoch nur die halbe Miete**

- Erfordert möglichst fehlerfreie Implementierungen
- Implementierung muss mit Laufzeitfehlern umgehen können
- Verfahren und Architekturen müssen **korrekt** arbeiten!



## **Wie lassen sich Ausnahmen vermeiden bzw. behandeln?**

## Im Fokus dieser Veranstaltung: **Software**

### 1. **Zuverlässige (robuste) Software entwickeln**

- Robustheit gegenüber externen Fehlern zur Laufzeit
  - Wie erkenne und toleriere ich solche Fehler?
- Wie testet man, ob man korrekt mit solchen Fehlern umgeht?
- Hier „forschen“ wir (hoffentlich auch zusammen mit euch)

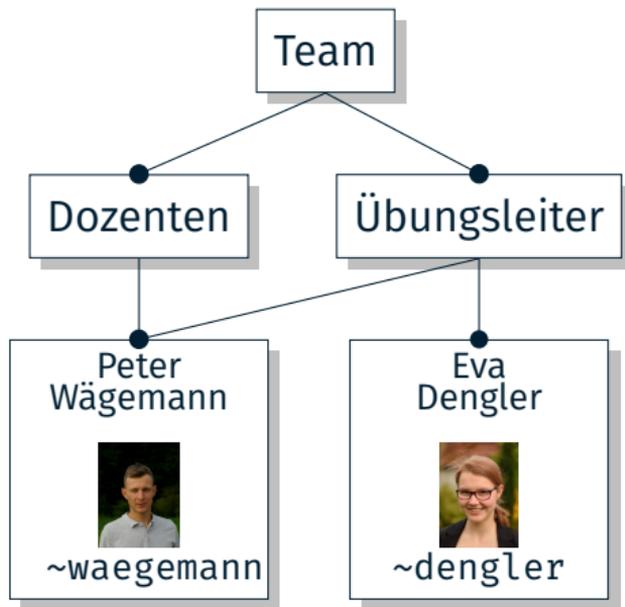
### 2. **Software zuverlässig entwickeln**

- Wie kommt man zu einer möglichst fehlerfreien Implementierung?
- Welche Werkzeuge helfen mir dabei?
  - Was tun diese Werkzeuge eigentlich?
  - Welche Grenzen haben diese Werkzeuge demzufolge?
- Hier „lernen“ wir zusammen mit euch

- ☞ **Zuverlässige (robuste) Software entwickeln**
  - Maskieren von Fehlern durch Redundanz
    - Replizierte Ausführung
    - Homogene und heterogene Redundanz
  - Härtung von Datenstrukturen und Kontrollfluss
    - Informationsredundanz
    - In Daten mithilfe von z.B. Prüfsummen
    - In Berechnungen/Kontrollfluss mithilfe arithmetischer Codierung
  - **Evaluierung** von Fehlertoleranzmaßnahmen
    - Fehlerinjektion und Testen
- ☞ **Anknüpfungspunkte für den praktischen Einsatz aufzeigen**
  - Niemand braucht das 1001. Fehlertoleranzprotokoll!
    - Das den gegenwärtigen Stand der Kunst nicht reflektiert
    - Obendrein (auf Grund der Komplexität) vielleicht fehlerhaft ist

- ☞ **Software zuverlässig entwickeln**
  - Typische Laufzeitfehler in C/C++ suchen+finden
    - Nullzeiger, Ganzzahlüberläufe, nicht initialisierter Speicher
    - Durch Testen oder mittels statischer Analysewerkzeuge
  - Testüberdeckung: Wie gut hat man getestet?
    - die Testüberdeckung für ein gegebenes Programm messen
    - Gibt es Zusammenhänge zwischen der Testüberdeckung, der Testfallanzahl und anderen Metriken?
  - Design-by-contract: statische, Werkzeug-gestützte Verifikation
    - Formulierung/Verifikation von Nachbedingungen für kleine C-Programme
    - Mit Werkzeugen (AbsInt Astrée) wie sie auch Airbus einsetzt
- ☞ **Vorurteile gegenüber formalen Methoden abbauen**
  - Keine **unverwendbaren Monster** mehr
    - Vollbringen aber auch **keine Wunder**
    - Anwendung noch immer mühsam, aber sie lohnt sich

- 1 Vorwort
- 2 Die Veranstaltung
  - Lernziele
- 3 Organisatorisches**
  - Die Beteiligten
  - Vorlesung und Übung
  - Leistungsnachweise



## ■ Wissensvermittlung

- Stoff der Vorlesung  $\rightsquigarrow$  Vorlesung + Folien
- Stoff der Tafelübung  $\rightsquigarrow$  Tafelübung + Folien

## ■ Praktische Arbeit

- Übungs- und Programmieraufgaben

## ■ Interaktion

- Vorlesung und Tafelübung
- Rechnerübung  $\mapsto$  Fragen zu Aufgaben, Abgaben, ...
- öffentliche Mailingliste: [i4ezs@lists.cs.fau.de](mailto:i4ezs@lists.cs.fau.de)
- Mail an Betreuer: [i4ezs-owner@lists.cs.fau.de](mailto:i4ezs-owner@lists.cs.fau.de)

## Termine

- Montag, 12:15 – 13:45, O2.134-113
- Inhalt
  - Aktueller Stoff der Vorlesung und Tafelübung
  - Laufende Übungsaufgaben
  - Allgemein fachliche sowie ethische Fragestellungen
- Ziele
  - Diskussion und Reflexion des Stoffes
  - Beantwortung von Fragen
  - Besprechung der Lernziele der Übungsaufgaben

## Ausnahmen

- siehe Webseite

 Webseite prüfen auf Änderung

## Termine

- Montag, 14:15 – 15:45, 02.133-113

### ■ Inhalt & Ziele

- Laufende Übungsaufgabe
- Lösen konkreter Probleme bei der Umsetzung der Aufgaben

### ■ [cipmap.cs.fau.de](http://cipmap.cs.fau.de)

- Anmelden einer Frage oder Abgabe
- Wir kommen dann nach Eingang der Anfragen zu euch

 **Auch hier: Webseite prüfen!**

## Ausfälle

- siehe Webseite

# Studien- und Prüfungsleistungen (1)

## VL – Vorlesung

2,5

Vorstellung und detaillierte Behandlung des Lehrstoffs

+

## Ü – Übung

2,5

- Praktische Übungen
- Pro Aufgabe: 14 Tage
- Online Abgabe

oder

## EÜ – Erweiterte Übung

5

- Übung (Ü)
- + erweiterte Aufgaben
- + vertiefende Abfrage

+

## RÜ – Rechnerübung

0

- **Betreutes** Arbeiten am Rechner
- Hilfe zu Werkzeugen und Techniken ...

# Studien- und Prüfungsleistungen (2)

## ■ **Wahlpflichtmodul** (Bachelor/Master) der Vertiefungsrichtung **Verteilte Systeme und Betriebssysteme**

- eigenständig (nur VEZS)
- mit weiteren Veranstaltungen

VL + Ü oder VL + EÜ  
siehe Modulhandbuch

## ■ Studien- und Prüfungsleistungen

- Bachelor
- Master

Prüfungsleistung  
Prüfungsleistung

erworben durch

- erfolgreiche Teilnahme an den Übungen
- erfolgreiche Bearbeitung aller Übungsaufgaben
- 30 min. mündliche Prüfung

## ■ Berechnung der Modulnote

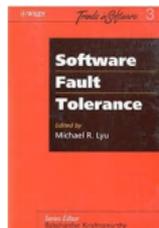
- Note der mündlichen Prüfung + “Übungsbonus” in Zweifelsfällen

[2]: Fehlertoleranz in Software:

M. Lyu, editor. *Software Fault Tolerance*.

**1995.**

[www.cse.cuhk.edu.hk/~lyu/book/sft/](http://www.cse.cuhk.edu.hk/~lyu/book/sft/)



[3]: Der „Klassiker“ für transiente Hardwarefehler:

S. Mukherjee. *Architecture Design for Soft Errors*.

**2008**



[1]: Weiteres Buch zu transienten Hardwarefehlern:

O. Goloubeva, M. Rebaudengo, M. S. Reorda, and M. Violante. *Software-Implemented Hardware Fault Tolerance*.

**2006**

