INFORMATIK ▪ CONSULTING ▪ SYSTEMS AG

9. Neu-Ulmer Test-Engineering-Day 2014

Systematisches Testen in sicherheitsgerichteten Echtzeitsystemen

Name	Dr. Thomas Liedtke
	thomas.liedtke@ics-ag.de
Funktion	Unit Manager Research & Development
Datum	05. Juni 2014

Agenda

- **Vorstellung (Wer bin ich, wer ist die ICS AG?)**
- **Motivation**
- **Maßnahmen, Methoden und Vorgehensweisen zur Softwareprüfung**
- **Beispiel aus der Praxis**
- **Zusammenfassung/ Ausblick**

Vorstellung wer bin ich?

- Dr. Thomas Liedtke
  - Seit 01. Dezember 2007 bei der ICS AG tätig als Leiter des Bereichs Research & Development
    - Senior Projektleiter Entwicklung sicherheitsgerichtete Projekte
    - Leiter der Competence Center der ICS AG
    - Implementierung Software Reifegradmodelle
    - Trainer/ Referent/ Consultant: Projektmanagement, Prozessberatung, …
  - Davor:
    - Studium der Informatik mit Nebenfach Mathematik an der Universität Stuttgart
    - Promotion Informatik/ Mathematik an der Universität Stuttgart
    - 14 Jahre bei Alcatel/ Alcatel·Lucent
      - Bereich Vermittlungssysteme:
        - Qualitätsabteilung/ SPI/ SEPG (CMM)/ Projektleiter für Telekom
        - Bereich Mobilfunk:
        - Abteilungsleiter in der Entwicklung
        - Oberprojektleiter für Mobilfunkprojekte (GPRS/ UMTS/ GSM/…)
      - Bereich Übertragungssysteme:
        - Leiter Supply Chain OND
      - Leiter des RSLC (Repair Service Logistic Center) Weilimdorf

Das Unternehmen

Rechtsform Aktiengesellschaft

Grundkapital 2,4 Mio. EUR

Vorstand
Franz-Josef Winkel, Cid Kiefer

Aktionäre
Die Familien: Hämer, Winkel und Kiefer

Hauptstandort
Stuttgart

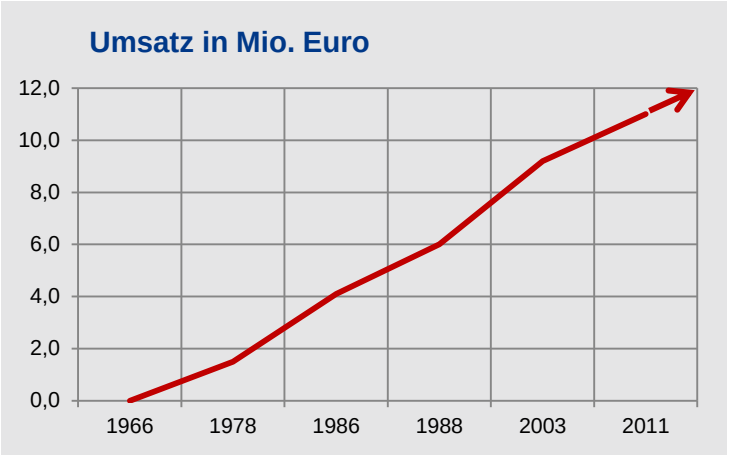
Geschäftsstellen
Berlin, Braunschweig, Ingolstadt, Markdorf, Leipzig, München, Rüsselsheim, Neu-Ulm

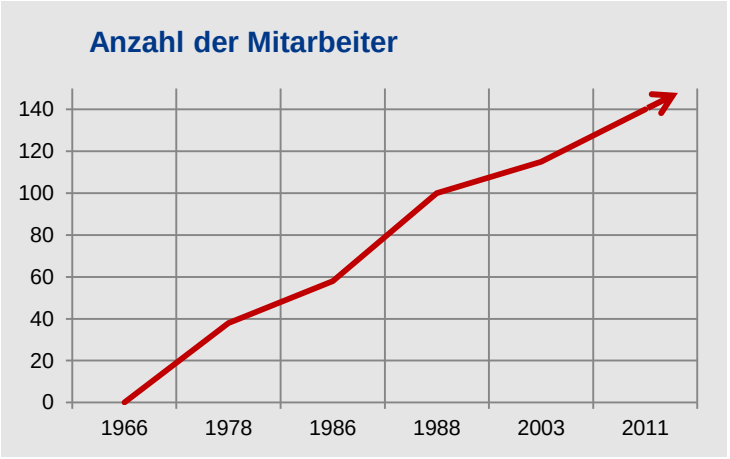
Anzahl der Mitarbeiter 150+

Umsatz 12.500.000.- EUR

Handelsregister
Amtsgericht Stuttgart HRB Nr.: 18569

UST. Id.Nr: DE 147802488





Geschäftsstellen

Hauptsitz Stuttgart

Sonnenbergstraße 13
70184 Stuttgart
Tel.: +49 711 21037 - 00
Fax: +49 711 21037 - 53

Geschäftsstelle Braunschweig

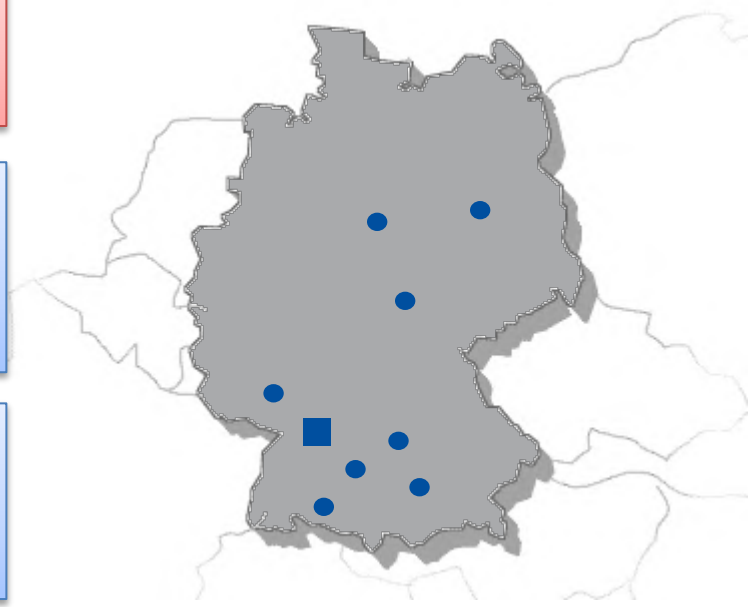
Salzdahlumer Straße 196
38126 Braunschweig
Tel.: +49 531 26306 - 33
Fax: +49 531 28503 - 46

Geschäftsstelle Rüsselsheim

Am Bahnhofsplatz 1
65428 Rüsselsheim
Tel.: +49 6142/701207 0-9

Geschäftsstelle Markdorf

Dornierstr. 1
88677 Markdorf
Tel.: +49 7544 50695 - 10
Fax: +49 7544 50695 - 14



Geschäftsstelle Neu-Ulm

Marlene-Dietrich-Straße 5
89231 Neu-Ulm
Tel.: +49 731 98588 - 709
Fax: +49 731 98588 - 710

Geschäftsstelle Berlin

Alt-Moabit 73
10555 Berlin
Tel.: +49 30 20687 - 262
Fax: +49 30 20687 - 375

Geschäftsstelle Leipzig

Marschnerstraße 3
04109 Leipzig
Tel.: +49 341 98250 - 10
Fax: +49 341 98250 - 11

Geschäftsstelle Ingolstadt

Am Augraben 21
85080 Gaimersheim
Tel.: +49 8458 60301 - 0
Fax: +49 8458 60301 - 12

Geschäftsstelle München

Lise-Meitner-Straße 1
85716 Unterschleißheim
Tel.: +49 89 203504 - 70
Fax: +49 89 203504 - 69

Philosophie

Partner für den gesamten Produkt Life Cycle

Vordenken

- Machbarkeitsstudien, Evaluierungen

Beraten

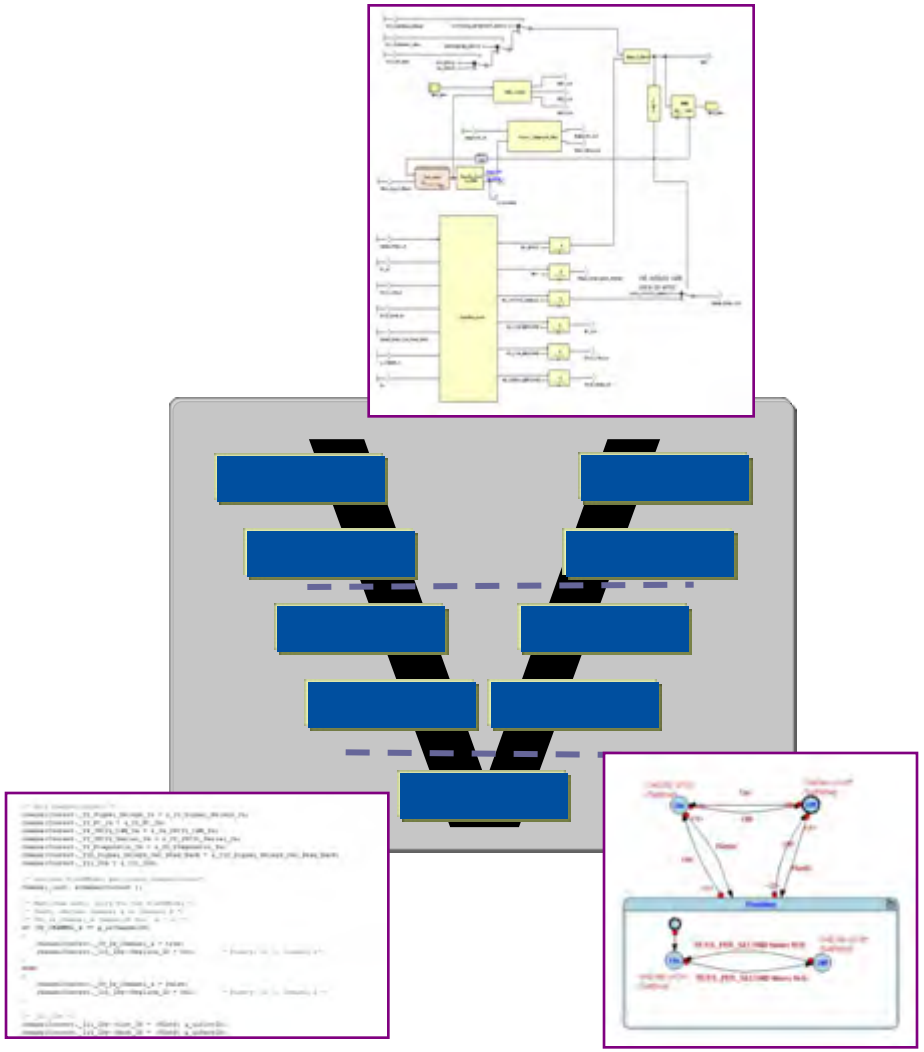
- Technologien
- Methoden
- Werkzeuge
- Standards
- Prozesse

Umsetzen

- Systems und Safety Engineering
- Produkt Design und Entwicklung
- Kapazitäts- und Kompetenzergänzung
- Test und Validierung
- RAM und Safety Management
- Assessment

Betreuen

- Maintenance von Software und Systemen

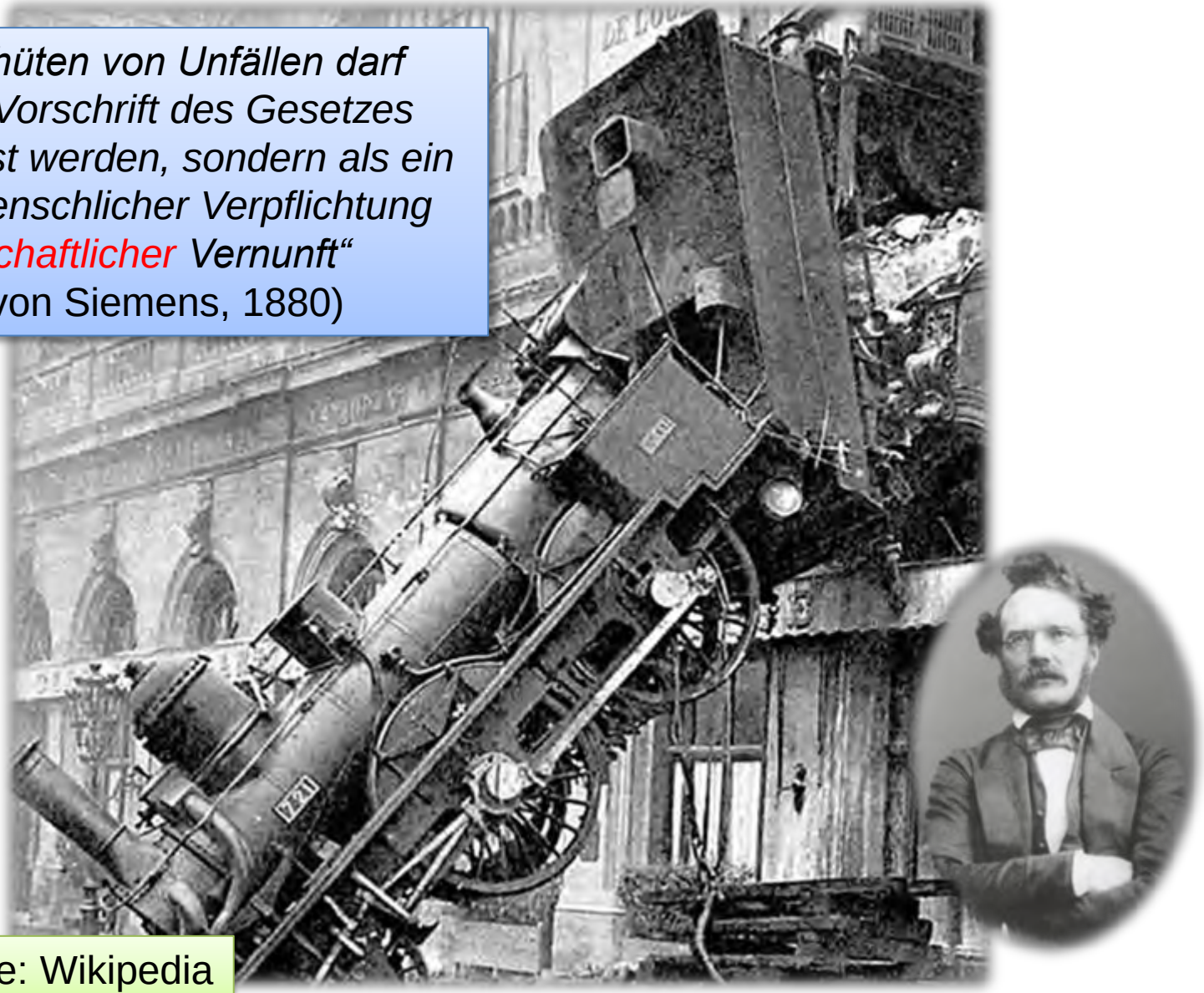


Agenda

- Vorstellung (Wer bin ich, wer ist die ICS AG?)
- **Motivation**
- Maßnahmen, Methoden und Vorgehensweisen zur Softwareprüfung
- Beispiel aus der Praxis
- Zusammenfassung/ Ausblick

Motivation

„Das Verhüten von Unfällen darf nicht als Vorschrift des Gesetzes aufgefasst werden, sondern als ein Gebot menschlicher Verpflichtung und wirtschaftlicher Vernunft“
(Werner von Siemens, 1880)



Bildquelle: Wikipedia

Motivation/ Begriffe

- Verschiedene Begriffe sind in verschiedenen Domänen unterschiedlich belegt
  - **Verifikation**: oft: überprüfen, ob **Dinge richtig gemacht** wurden (vertikal/ unten im V-Modell), meist direkte Verknüpfung mit der Testdurchführung
  - **Validierung**: oft: überprüfen, ob **die richtigen Dinge gemacht** wurden (horizontal/ oben im V-Modell), nachweisen u.U. ohne direkte Testdurchführung
  - **Integration**/ **Testen**/ **Prüfen**
  - …
- Wozu Prüfung?
  - Softwaresysteme spielen in einem zunehmend großen Teil des Lebens eine wichtige Rolle:
    - Steuerung von **Transportmitteln**,
    - Steuerung von **Kraftwerken**,
    - Assistenzsysteme in **Autos**,
    - Smart Grid, IoT, Industrie 4.0
    - Relativ neu: **Security!**
    - …

⇒ Gefahr für Leib und Leben, Umwelt, Personenschäden, Sachschäden, finanzielle Schäden, Imageschäden und Katastrophen sind möglich

Systematischer Ausfall nach IEC 61508-4

Im Folgenden geht es beim Testen/ Prüfen um systematische Ausfälle:

- Systematischer Ausfall:
  - *„Systematisches Versagen/ Ausfall, bei dem eindeutig auf eine Ursache geschlossen werden kann, die nur durch eine Modifikation des Entwurfs oder des Fertigungsprozesses, der Art und Weise des Betreibens, der Bedienungsanleitung oder anderer Einflussfaktoren beseitigt werden kann.“*
- Zufälliger Hardwareausfall:
  - *„Ausfall, der zu einem zufälligen Zeitpunkt auftritt und der aus einem oder mehreren möglichen Mechanismen in der Hardware resultiert, die zu einer Verschlechterung der Eigenschaften der Bauteile führen.“*
- Klassifizierung der Fehler anhand folgender Fragen:
  - „War der Fehler zum Zeitpunkt der Inbetriebnahme bereits eingebaut?“
  - „Ist der Ausfall prinzipiell reproduzierbar?“
- Wichtiges Unterscheidungsmerkmal:
  - *„Systemausfallraten, die aus zufälligen Hardwareausfällen herrühren, können mit vernünftiger Genauigkeit statistisch quantifiziert werden, jene die durch systematische Ausfälle entstehen nicht, weil die Ereignisse, die zu diesen führen, nicht leicht vorausgesagt werden können.“*

Berühmte Softwarepannen

- Fast der dritte Weltkrieg:
  - *„Als Oberst Stanislav Petrov am 26. September 1983 fünf amerikanische Interkontinental-Raketen auf die UDSSR zufliegen sah, hat er nicht auf den berühmten roten Knopf gedrückt, um den Gegenschlag auszulösen. Er fand es unwahrscheinlich, dass die Amerikaner mit nur fünf Cruise Missiles einen nuklearen Holocaust anfangen wollten. Im Gegensatz zu der* ***Satelliten-Software, die Sonnen-Reflektionen auf einer Wolkendecke fälschlicherweise als Raketen-Starts gedeutet hatte****.“*

Quelle: CHIP, Markus Mandau, http://www.zehn.de/die-10-groeszten-software-pannen-732-0

Anforderungen an die Softwareprüfung

- Fehlervermeidung:
  - Fehler wie die auf den vorigen Folien beschriebenen sollten so weit wie möglich ausgeschlossen werden oder gefunden werden, **bevor sie sich auswirken**
- Fehlervorhersagen
  - Aussagen der Form *„in welchem Teil der Software sind anteilsmäßig wie viele Fehler zu erwarten?“* sollen gemacht werden können

⇒ Risiko soll minimiert werden

- **Risiko** =
  Wahrscheinlichkeit seines Eintretens
  x **Schaden** im Falle seines Eintretens

Was sind Softwarefehler?

Genaue Unterscheidung, was mit „**Fehler**“ gemeint ist

- **Fehlhandlung** (error): Mensch verursacht einen Fehler in der Software
- **Fehlerzustand** (defect oder fault): im Code einer Software, in einem System oder in einem Dokument (z.B. inkorrektes Teilprogramm, inkorrekte Anweisung oder inkorrekte Datendefinition), verursacht durch eine Fehlhandlung, kann zu einer Fehlerwirkung führen
- **Fehlerwirkung/ Fehlverhalten** (failure): Wenn der Defekt im Code ausgeführt wird, kann das System nicht das tun, was es tun sollte (oder etwas tun, was es nicht tun sollte) und dabei eine Fehlerwirkung hervorrufen oder ausfallen.

- Im Englischen weitere Begriffe: mistake, bug, malfunction, issue, incident, flaw, vulnerability, error-prone, …
- …

- Beispiel: Telefon funktioniert nicht <- kein Freizeichen <- Codefehler

(Funktionale) Sicherheit in diversen Szenarien

Bildquelle: Wikipedia

Qualitätsmerkmale

- Funktionalität
  - Spezifizierte oder erwartete Funktionen (Verhalten)
- Zuverlässigkeit
  - Aufrechterhalten der Leistungsfähigkeit
- Benutzbarkeit
  - Aufwand zur Benutzung
  - Benutzerfreundlichkeit
- Effizienz
  - Verhältnis von Leistung und dafür verbrauchten Ressourcen
- Änderbarkeit
  - Aufwand zur Durchführung vorgegebener Änderungen
- Übertragbarkeit
  - Möglichkeit des Transfers in andere Umgebung (SW, HW, etc.)

Grundsätze der Softwareprüfung

- Softwareprüfung kann nur **gegen Vorgaben** (Anforderungen/ Vergleichsresultate) erfolgen
- Prüfverfahren müssen **definiert** sein und **reproduzierbare** Ergebnisse liefern
- Prüfergebnisse müssen **dokumentiert** werden (Nachvollziehbarkeit)
- Erkannte Fehler müssen (üblicherweise) anschließend **korrigiert** werden, indem die Fehlerursachen erkannt und behoben werden
- **Systematisch prüfen**
  - Prüfung planen
  - Prüfungen nach Vorschriften durchführen (am besten automatisiert ausführen)
  - Nicht nur die Software selbst prüfen, sondern auch alle Dokumente im Umfeld (zum Beispiel Anforderungen! Siehe Reviews)
- **Testen…**
  - *„ist der überprüfbare und jederzeit wiederholbare Nachweis der Korrektheit eines Softwarebausteines relativ zu vorher festgelegten Anforderungen“* [DEN91]
  - *„can prove the presence of errors, but not their absence“,* bzw. *„program testing can be a very effective way to show the presence of bugs, but is hopelessly inadequate for showing their absence.“* [DIJ72]

Testen: Ziele und Probleme

- Fehlerzustände und –wirkungen nachweisen
  - Durch Fehlerhandlungen Fehlerzustände provozieren
- Vertrauen in das Programm erhöhen
  - Je weniger gefundene Fehler, desto höheres Vertrauen in die Software (manche nennen das „Qualität“)
- Fehlerwirkungen vorbeugen
  - Je früher Fehler entdeckt werden, …

  … desto billiger die Fehlerkorrektur

  … desto weniger Folgefehler

  -> vgl. Methoden, die durch Konstruktion in frühen Phasen Fehler so unwahrscheinlich wie möglich machen (Abstrakte Spezifikation, Modellierung, Generierung der Testspezifikation ...)
- Aber:
  - Vollständiges Testen ist nicht möglich
  - Vorsicht vor der Fehlermaskierung
  - „keine Fehler“ heißt noch nicht „brauchbares System“
  - Korrelation zum Lebenszyklusmodell. Z.B.: *„Wie verträgt sich der Anspruch mit dem agilen Manifest?“*

Agenda

- Vorstellung (Wer bin ich, wer ist die ICS AG?)
- Motivation
- **Maßnahmen, Methoden und Vorgehensweisen zur Softwareprüfung**
- Beispiel aus der Praxis
- Zusammenfassung/ Ausblick

Bewährte Vorgehensweise beim Test

- **Rollen** beachten (z.B. Testmanager, Testanalyst, Testspezialist, Test-Automatisierer, Testsupport)
- **Personaltrennung** (besonders zwischen Entwickler, Planer und Tester!)
- **Anforderungen**/ Testspezifikation: Möglichst **formal**
- Mit dem Testen **so früh wie möglich** beginnen
- Nutzung des **Pareto-Prinzips** (Fehler sind nicht gleichverteilt, sondern häufen sich)
- Nutzung von **Regressiontests** und Ergänzung durch neue Tests → der Testresistenz entgegenwirken
- Anpassung der Tests an das System und die grundlegenden Anforderungen (zum Beispiel sicherheitskritisch?)

- *„Tests sind nicht die einzige Maßnahme im Qualitätsmanagement der Softwareentwicklung, aber oft die letztmögliche. Je später Fehler entdeckt werden, desto aufwändiger ist ihre Behebung …“* [BWS04]

Statische Methoden

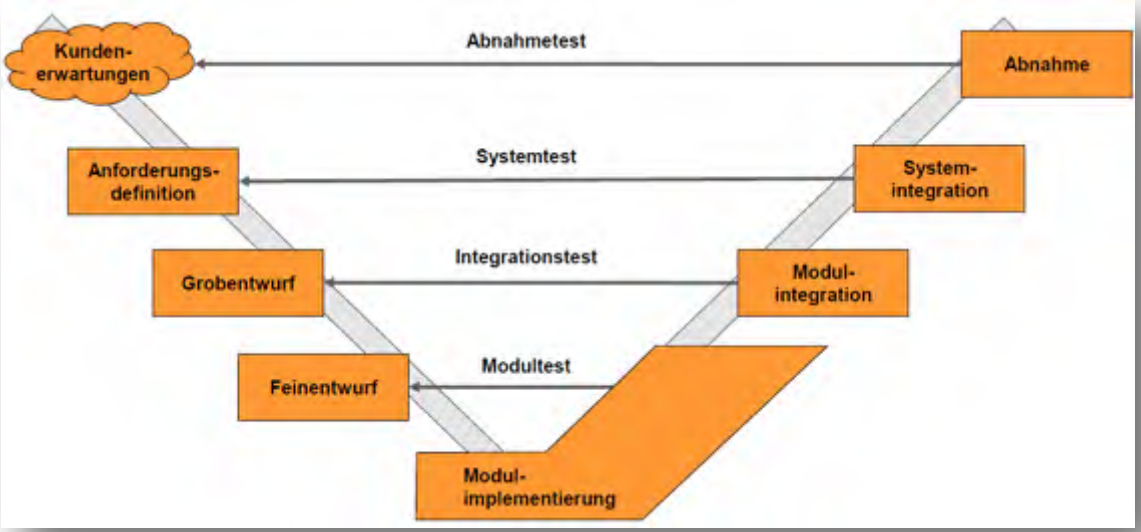
- Review
  - Überprüfung schriftlicher Dokumente durch Gutachter
  - mehr oder weniger formalisierter Prozess zur Überprüfung schriftlicher Dokumente durch Gutachter, um Stärken und Schwächen des Dokuments festzustellen
  - Ziele: Feststellung von Mängeln, Fehlern, Inkonsistenzen, Unvollständigkeiten, Verstößen gegen Vorgaben, Richtlinien, Standards, Pläne
- Code-Inspektion
  - Diskussion von Source-Code durch Softwareentwickler
  - der Autor eines Programms diskutiert Schritt für Schritt sein Programm mit anderen Softwareentwicklern (Moderator, Autor, Tester, Qualitätssicherungs-MA)
- Statische Analyse
  - Werkzeuggestütztes Auffinden von Fehlern ohne direkte Ausführung des Systems
  - Teilweise heutzutage schon vom Compiler/ Entwicklungswerkzeug erledigt, teilweise durch spezielle Werkzeuge zur statischen Analyse

Dynamische Methoden

- Walkthrough
  - Designer/ Programmierer führt Teammitglieder und andere Stakeholder durch ein Software-Produkt, Teilnehmer stellen Fragen und geben Kommentare ab
- Symbolische Ausführung, Simulation
  - Kann früh Probleme zeigen, z.B. Schnittstellenprobleme → ggf. aufwändig
- Testen
  - Durch Testen kann die Qualität eines Softwaresystems nicht direkt nachgewiesen oder gemessen werden
    - Höchstens die Anwesenheit von Fehlern kann nachgewiesen werden, nie die Abwesenheit! Auch „vollständiges Testen“ ist nicht möglich
    - Pareto-Prinzip: Je mehr Fehler gefunden wurden (in einer bestimmten Komponente, Funktion etc.), desto mehr weitere sind dort zu vermuten
  - Aber: Intensives Testen von Systemen und Dokumentation kann helfen, das Risiko, dass Probleme im operativen Betrieb auftreten, zu reduzieren

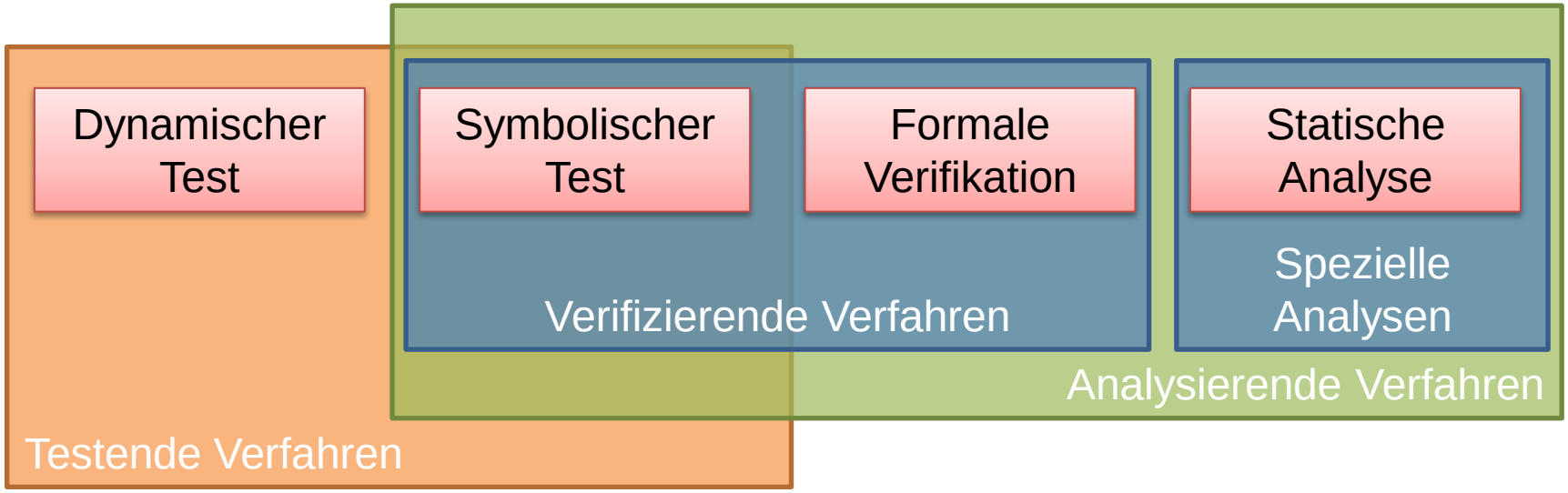
    → Fehler **vor der Freigabe** in der operativen Verwendung finden und beheben
  - Softwaretesten kann auch notwendig sein, um vertragliche oder gesetzliche Vorgaben oder spezielle Industrienormen zu erfüllen.

Teststufen im Beispiel V-Modell



- **Modultest**: Aufdeckung aller Abweichungen der **Implementierung** von der Modulspezifikation
- **Integrationstest**: Aufdeckung von Fehlern in **Schnittstellen** und im Zusammenspiel zwischen integrierten Modulen
- **Systemtest**: Aufdeckung aller Abweichungen des **Systemverhaltens** in Bezug auf das in der Anforderungsdefinition spezifizierte Systemverhalten
- **Abnahmetest**: Aufdecken aller Fehler, die auf Missverständnissen bei Absprachen zwischen Benutzer und Entwickler, falschen Schätzungen von Datenmengen, unrealistischen Annahmen bezüglich der realen Systemumgebung beruhen

Unterscheidung Prüf- und Testverfahren



Testverfahren	White Box	Grey Box	Black Box	diversifi-zierende	funktions-orientiert	struktur-orientiert
Exploratives Testen			X		X	
Überdeckungstests Beispiel: Pfadüberdeckung	X					X
Zustandsbasierter Test			X		X	
Äquivalenzklassenbildung Spezialfall: Grenzwertanalyse		x	X		X	
Back to Back Test		X		X	X	X
Regressionstests		X		X	X	

Testvorgehensweise - Beispiel

1. Vorbereitung des Testobjekts
2. Bereitstellung einer Testumgebung
   - Simulation der realen Einsatzbedingungen für ein Testobjekt
   - Ggf. auch (noch) nicht vorhandener Ressourcen oder Daten oder Nachrichten …
3. Testfallermittlung
   - Blackbox-Ansätze: Spezifikationsbasiert (Äquivalenzklassenbildung, Grenzwertanalyse, Ursachen-Wirkungsanalyse, Error Guessing)
   - Whitebox-Ansätze: Implementierungsbasiert (Anweisungs- / Zweig- / Pfadüberdeckung, Datenflussanalyse, Symbolischer Test)
   - Erfahrungsbasierte Verfahren
4. Auswahl geeigneter Testfälle und Testdaten
   - Meist stichprobenartig, um mit möglichst wenigen Tests möglichst viele Fehler finden zu können, idealerweise (zum Teil) automatisierbar
   - welche Abläufe sind wichtig /häufig / typisch, wichtig auch „Randfälle“
5. Testausführung
6. Testauswertung

Testdokumentation nach der IEEE 829

- Übersicht
  - Testplan: Umfang, Vorgehensweise, Terminplan, Testgegenstände
- Testspezifikation (test specification)
  - Testdesignspezifikation: Die im Testplan genannten Vorgehensweisen im Detail.
  - Testfallspezifikationen: Die Umgebungsbedingungen, Eingaben und Ausgaben eines jeden Testfalls.
  - Testablaufspezifikationen: In Einzelschritten, wie jeder Testfall durchzuführen ist.
- Test-(Ergebnis-)Beschreibung (test reporting)
  - Testobjektübertragungsbericht: Wann wurden welche Testgegenstände an welche Tester übergeben
  - Testprotokoll: Chronologisch alle relevanten Vorgänge bei der Testdurchführung
  - Testvorfallbericht: Alle Ereignisse, die eine weitere Untersuchung erforderlich machen.
  - Testergebnisbericht: Beschreibt und bewertet die Ergebnisse aller Tests.

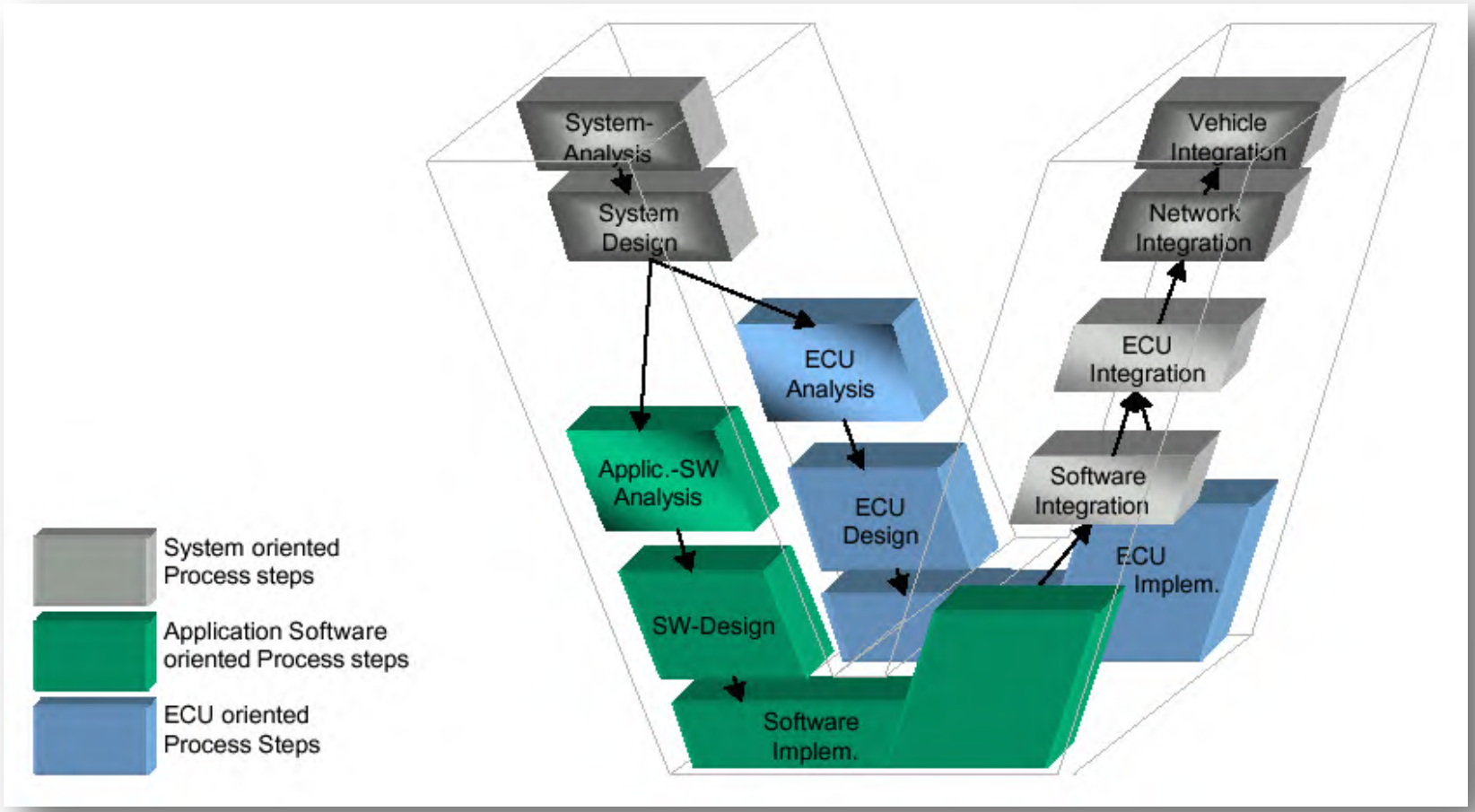
Teststrategie

- Es gibt eine Reihe von Strategien, die z.T. gemischt werden können
  - top-down: Hauptfunktionen werden vor Detailfunktionen getestet; letztere werden beim Test zunächst ignoriert oder (mittels "Stubs") simuliert
  - bottom-up: Detailfunktionen werden zuerst getestet; übergeordnete Funktionen oder Aufrufe werden mittels "Testdriver" simuliert
  - Risk-based: Die Testabdeckung bzw. –reihenfolge wird an den Risiken ausgerichtet, die in den Testobjekten (für den Fall des Nichtfindens von Fehlern) eintreten können.
  - Data-driven: Testfall mehrfach mit dem selben Ablauf, aber unterschiedlichen Eingabe- und Vergleichswerten durchführen
  - Framework-based: Test-Automatisierung mittels Testwerkzeugen für bestimmte Entwicklungsumgebungen/ Programmiersprachen
  - Testgetriebene Entwicklung: Tests werden vor dem oder parallel zum eigentlichen Code geschrieben und durchgeführt

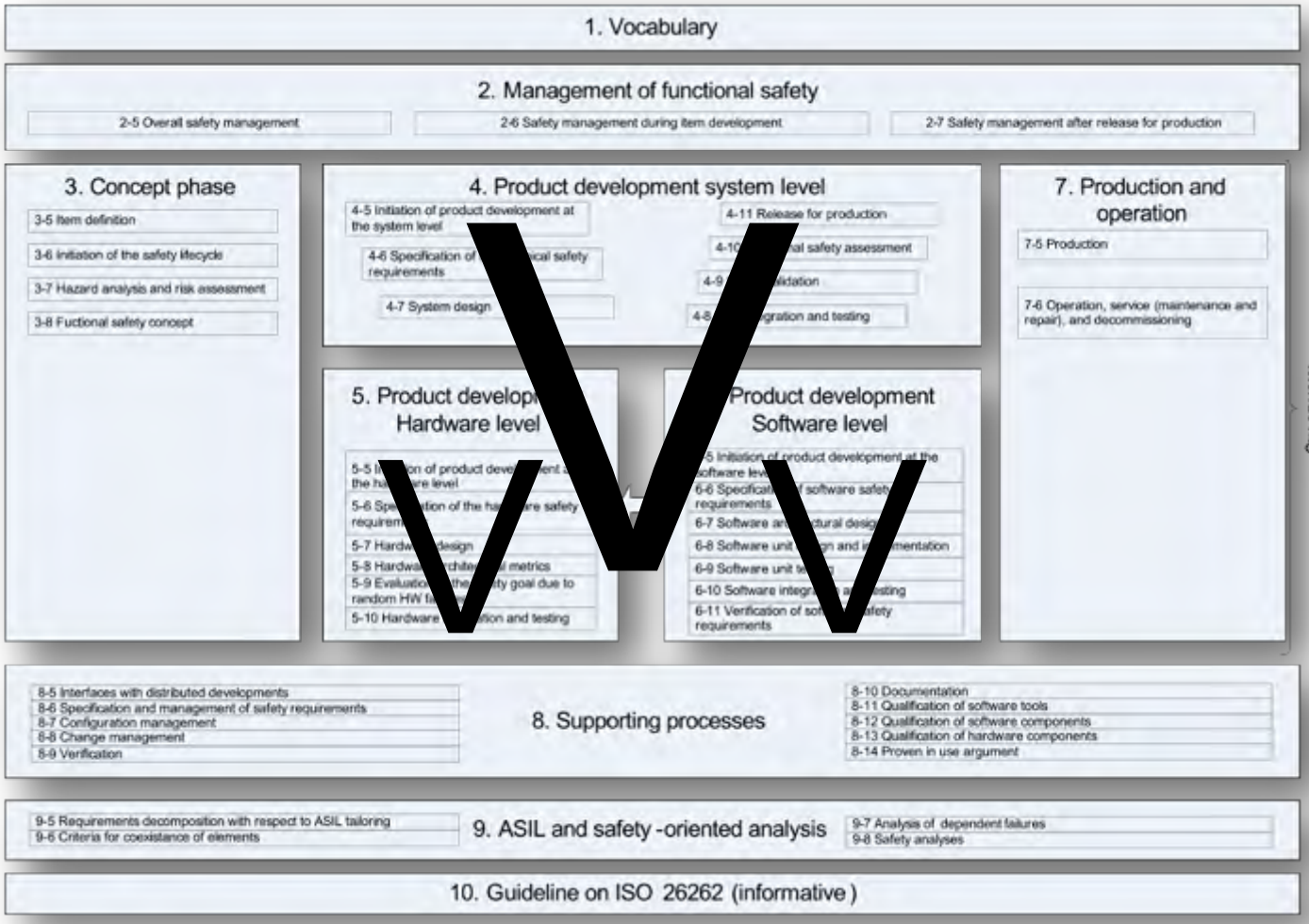
Agenda

- Vorstellung (Wer bin ich, wer ist die ICS AG?)
- Motivation
- Maßnahmen, Methoden und Vorgehensweisen zur Softwareprüfung
- **Beispiel aus der Praxis**
- Zusammenfassung/ Ausblick

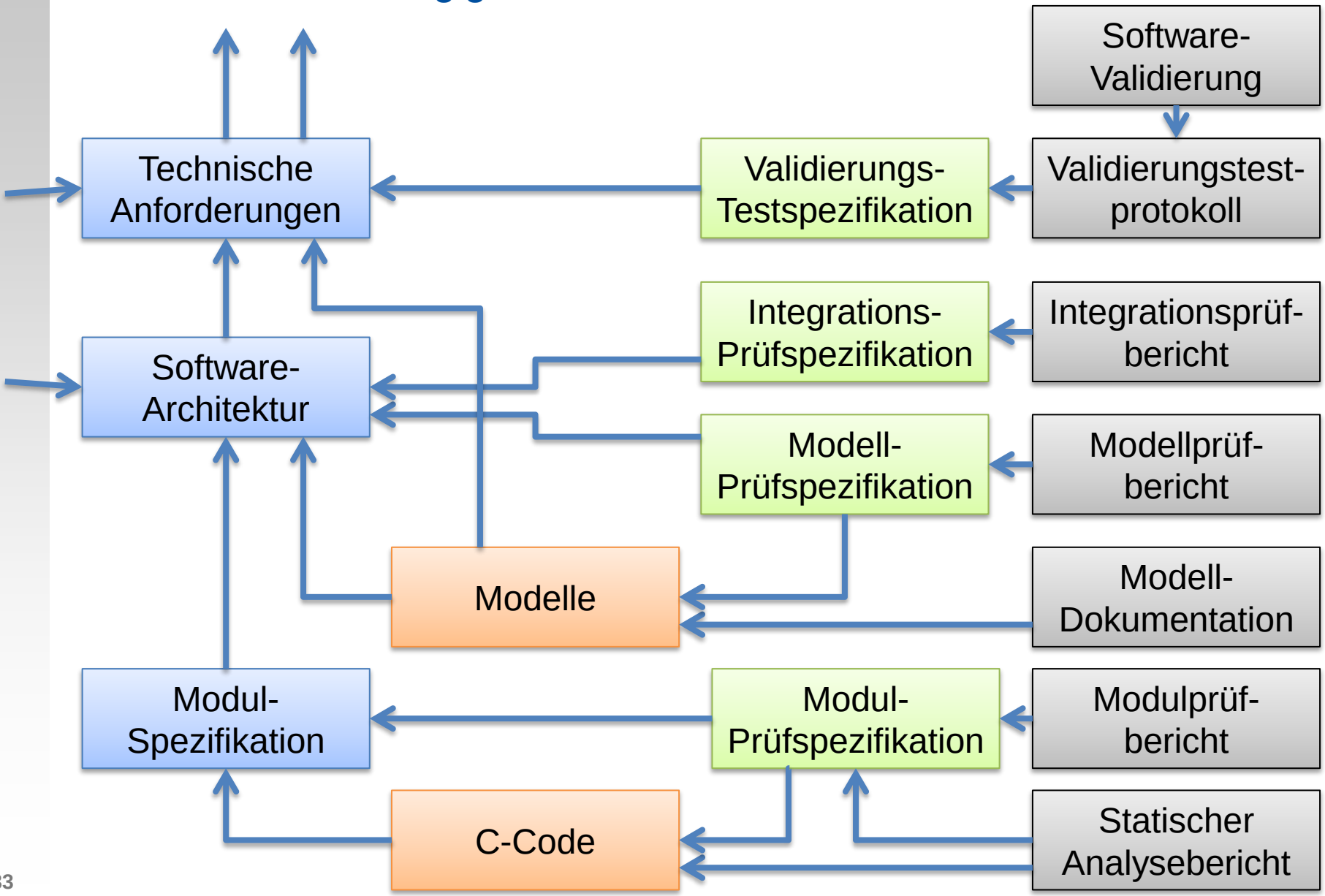
Entwicklungsprozesse und deren Wechselwirkungen



Automotive Safety Standard ISO 26262



Vereinfachte Darstellung einer beispielhaften Struktur von Abhängigkeiten



Modul-Prüfspezifikation

- Statische Analyse der SW-Module (z.B: Werkzeuggestützt)
  - Konformität zu Programmierrichtlinien
  - Bestimmung des tatsächlichen Stackbedarfs
- Durchführen einer Code-Inspektion
  - Übereinstimmung mit SW-Entwurf (Architektur)
  - Korrektheit der nicht im Rahmen des Modultests zu prüfenden SW-Module
  - Konformität des C-Codes zu den nicht mittels Tool prüfbaren Programmierrichtlinien/ Codierregeln (z.B. Dokumentation)
  - Übereinstimmung mit Build-Optionen
  - Einhaltung eines u.U. geforderten Sprachkonstruktsubset
- Testen der SW-Module
  - Testen des Verhaltens an deren Schnittstellen in Bezug auf Vorgaben der Modulspezifikation
  - Testen der SW-Module unabhängig voneinander
  - Äquivalenzklassen durch Aufteilung logischer Modulfunktionen und Partitionierungen der Eingaben (Beachtung von Grenzwerten)
- Nachweis der Testabdeckung
  - Messung der Codeabdeckung

Modell-Prüfspezifikation

- Formale Verifikation der Modelle
  - Prüfung ausgewählter Anforderungen wie z.B. formulierter Invarianten in Anforderungen
- Testen der Modelle
  - Testfälle testen das Verhalten der SW an deren Schnittstellen
  - Funktionale Testfälle/ Black-Box-Anforderungsgetrieben
  - Jede logische Bedingung mit n Teilbedingungen führt bei MC/DC Kriterium zu n+1 Äquivalenzklassen. Mindestens ein Testfall je Äquivalenzklasse
- Messung und Nachweis der Modellabdeckung
  - Hilfe einer MTC
    - Vollständigkeit der Testfälle
    - Nichtvorhandensein von nutzlosem/ totem Code

Integrations-Prüfspezifikation

- Einheiten aus der SW-Architektur sind zu prüfen
- Verifikation C-Compiler
- Test Bootloader
- Integration und Test der Software auf der Zielhardware in mehreren Integrationsstufen
  - Integration und Test der HW-Zugriffsschicht (Stufe 1)
    - Prüfkriterium: Abdeckung der SW-HW-Schnittstellen
  - Integration und Test der Ablaufsteuerung (Stufe 2)
    - Prüfkriterium: Abdeckung der Funktionen der Ablaufsteuerung und deren Schnittstellen zur HW-Zugriffsschicht
- Testaufbau:
  - Software unter Test
  - Testsoftware
  - FPGA
  - I/O-Karten
  - …

Integrations-Prüfspezifikation: Beispiel Stufe 1

1. Die HW wird durch die SW korrekt konfiguriert (CPU-Clock, FPGA-Adressen, CPU-Ports).
2. Die Kommunikation mit dem FPGA zur Synchronisierung eines Berechnungszyklus wurde korrekt realisiert.
3. Die Überwachung der Zykluszeit durch das FPGA funktioniert korrekt
4. CPU-Fehler (z.B. fehlerhafter Speicherzugriff) werden durch die SW korrekt behandelt.
5. Der Vergleich der Ausgänge der redundant ausgeführten SW-Einheiten durch das FPGA (HW-Vergleicher) ist korrekt realisiert.
6. Die fehlerhafte Ansteuerung des Watchdogs der I/O-Karten durch SW (via FPGA) wird erkannt und führt zum sicheren Zustand der I/O-Karten-Ausgänge.
7. Alle Inputs/Outputs werden auf die richtigen HW-Ports abgebildet.

- Diese Integrationsstufe gilt als erfolgreich, wenn 100% der Tests erfolgreich waren.

Integrations-Prüfspezifikation: Beispiel Stufe 2

1. Das zyklische Einlesen und Verarbeiten von Anwendungsdaten wird nach erfolgreicher Initialisierung korrekt durchgeführt.
2. Die logischen Ein-/Ausgabedaten werden durch die Ablaufsteuerung und Daten-I/O korrekt von den zugehörigen HW-Ports gelesen und an die Sicherungslogik übergeben bzw. die von dort erhaltenen Anwendungsdaten auf die entsprechenden HW-Ports geschrieben.
3. Checksumme, Versionskennungen und Fehlermeldungen werden korrekt an die Steuereinheit übertragen.
4. Ansteuerung der LEDs im Normal- und Fehlerfall

- Diese Integrationsstufe gilt als erfolgreich, wenn 100% der Tests erfolgreich waren.

Validierungs-Testspezifikation

- Validierungstests für die vollständig integrierte Software
- Test der Funktionen der komplett integrierten Software-Einheit in der originalen HW-Umgebung
- Herleitung der Testfälle in der Abdeckung der normalen Betriebsfunktionen, sowie Funktionsüberwachungen

Agenda

- Vorstellung (Wer bin ich, wer ist die ICS AG?)
- Motivation
- Maßnahmen, Methoden und Vorgehensweisen zur Softwareprüfung
- Beispiel aus der Praxis
- **Zusammenfassung/ Ausblick**

Zusammenfassung/ Ausblick

Weitere Stichwörter:

- Error seeding
- Model based (risk) testing
- Design oriented testing
- Tools-Qualifikation (normengefordert)
- Nichtfunktionale Anforderungen
- …

Literaturquellen

- [BWS04] Basiswissen Softwaretest, Andreas Spillner, Tilo Linz, Dpunkt Verlag; Auflage: 2. überarb. A. (2004)
- [DIJ72] Turing-Award Lecture von 1972, Dijkstra
- [DZGSP08] CHIP, Markus Mandau, http://www.zehn.de/die-10-groeszten-software-pannen-732-0, 2008
- [DEN91] Ernst Denert: Software-Engineering. Springer, Berlin 1991, 1992
- [EWT10] Expertenwissen Testen, Bert Rotmans, Ernst Müller, DieBirne Bildungsmedien AG, 2010
- [IEC-61508] Standard „Functional safety of electrical/electronic/programmable electronic safety-related systems“
- [ISO-26262] „Road Vehicles – Functional Safety“

Fragen/ Diskussion/ AOB