Mechatronische Systeme

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl für Informatik 11 (Software Engineering)

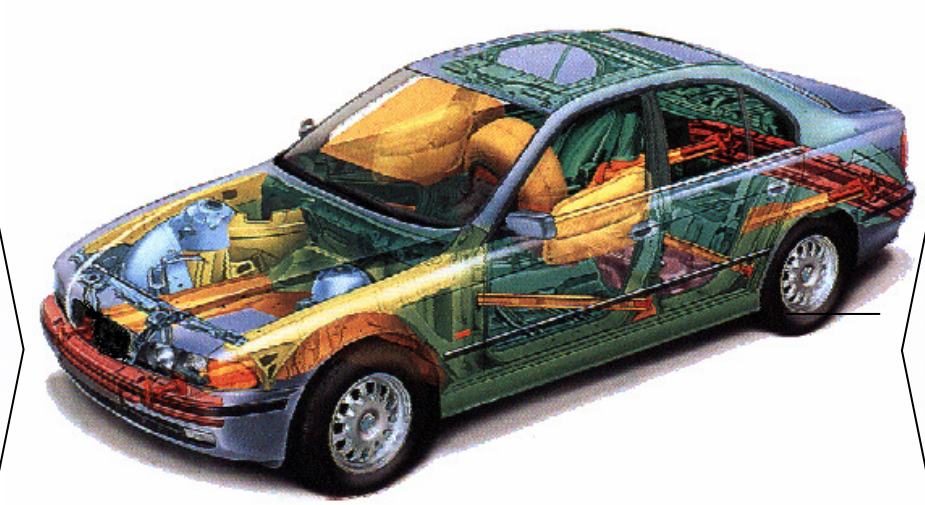
Teilgebiet Informatik / Software

Francesca Saglietti
Lehrstuhl für Software Engineering
Institut für Informatik
saglietti@informatik.uni-erlangen.de

Folien zum Download unter:
URL: http://www11.informatik.uni-erlangen.de/Lehre/SS2006/
→ Weitere Veranstaltungen → Ringvorlesung Mechatronische Systeme

Leitdemonstrator PKW

- Konstruktion
- Mechanik
- Sensorik
- Integrierte Schaltungen
- Regelungen
- Software
- Antriebstechnik
- Material
- Fertigung
- Montage

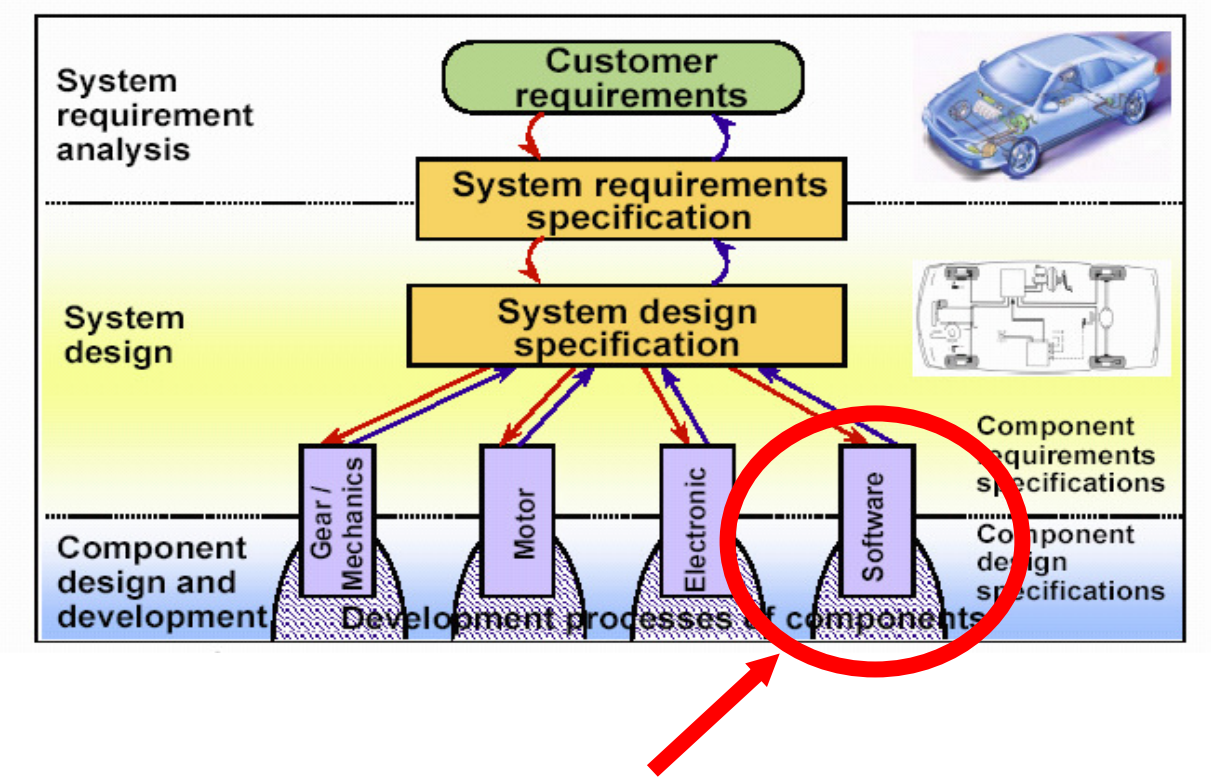


- Servolenkung (EAPS)
- Bremssystem ABS/ESP
- Bordnetzsystem
- Beleuchtungs-anlage
- aktive Dämpfung
- Kommunikations-technologien
- Türmodul

Entwurfsprozess mechatronischer Systeme

- **Definition und Analyse funktionaler Anforderungen**
- **Systementwurf**
  Lösungsumsetzung des Problems durch Interaktion unterschiedlicher Komponenten (mechanische, elektronische, Software-Bausteine)
- **Komponenten-Entwurf und -Entwicklung**
  Realisierungsdetails der einzelnen Komponenten werden definiert und implementiert

(aus [2])



- *Software ist scheinbar leicht zu ändern; muss daher meist als letztes Teilsystem an die von den anderen Teilsystemen bereits vorgegebenen Randbedingungen angepasst werden*

Lernen aus Fehlerszenarien

- Therac 25 Verstrahlung mehrerer Patienten (1985-87)
- AT&T Telefon-Rechnerausfall (1990)
- Scheitern Patriot-Raketenabwehrsystem (1991)
- LH Airbus 320 Notlandung in Warschau (1993)
- Versagen elektronisches Stellwerk in Hamburg (1995)
- Ariane 5 Explosion (1996)
- Antriebloses US-Kriegsschiff Yorktown (1997)
- LH Airbus 320 Unglück bei Landung in Ibiza (1998)
- Verlust Mars Climate Orbiter Sonde (1999)
- Rechnerabsturz der Berliner Feuerwehr (2000)
- Personenschaden durch Aladin Nitrox-Tauchcomputer (1999,2002)
- Notlandung Soyuz Raumkapsel (2003)
- Versagen Mars Exploration Rover Spirit (2004)

"Learn from the mistakes of others,... you'll never live long enough to make them all yourself."

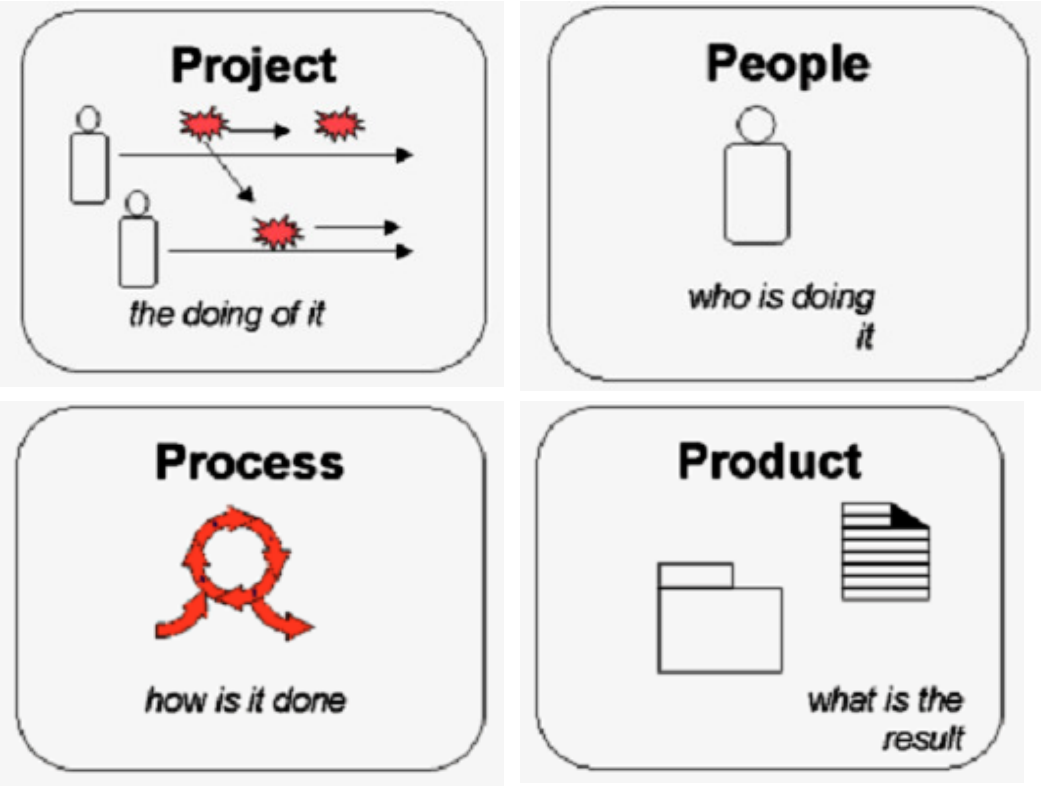
Software Engineering - Zielsetzung

Die Unfälle erklären den Bedarf an einer ingenieurmäßigen Entwicklung von Softwaresystemen, die folgenden Randbedingungen unterworfen sind:

1. Problemkomplexität	Im Wesentlichen durch Produktgröße bestimmt, z.B. im Falle von Programmlänge > 20 K LOC
2. Qualitätsvorgaben	Zuverlässigkeit, Wartbarkeit
3. Kostenlimits	Geld, Personal
4. Terminvereinbarungen	Lieferung, Genehmigung

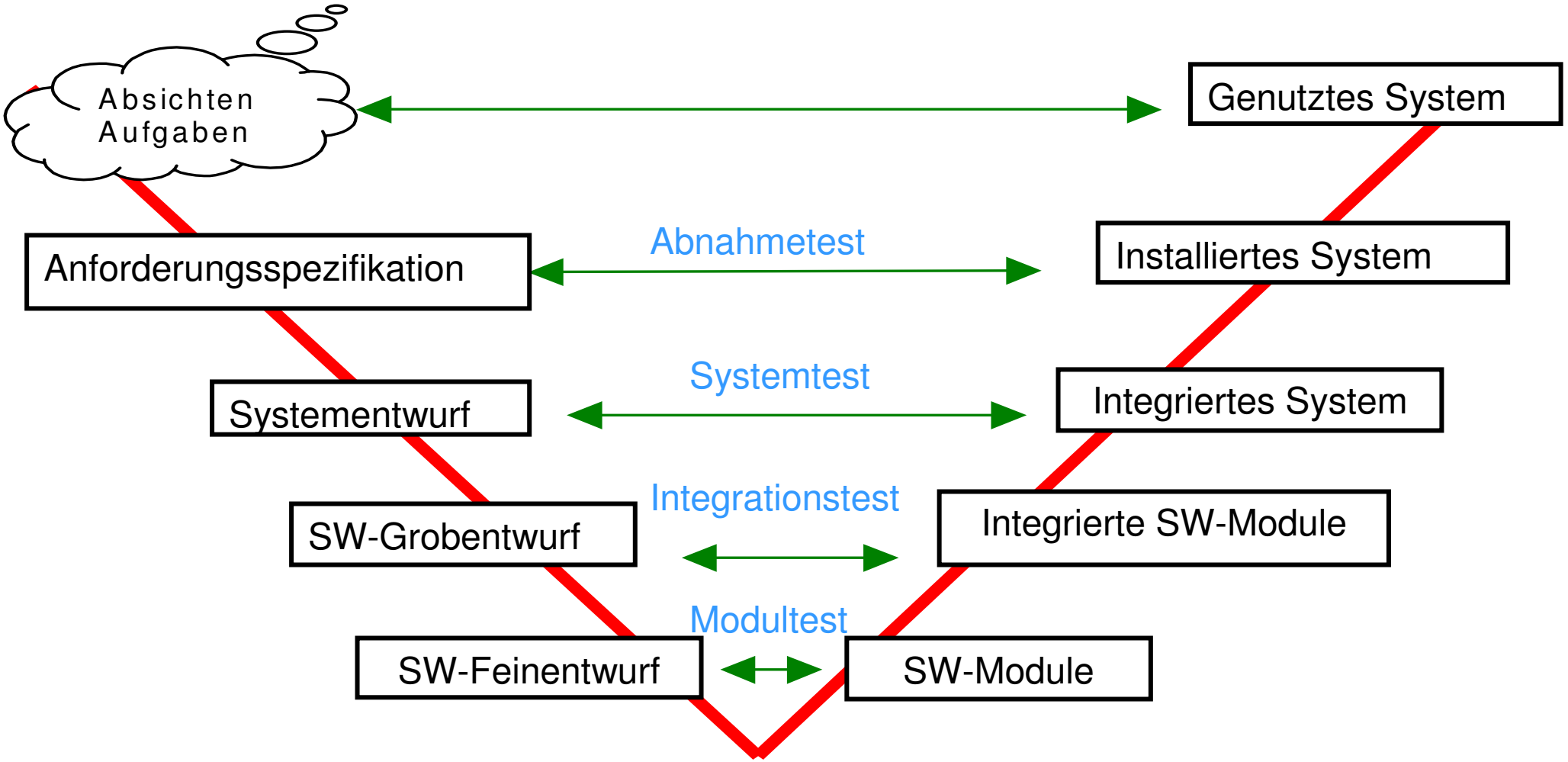
Gegenstand des **Software Engineering** ist daher:
die **ingenieurmäßige** Entwicklung
komplexer, umfangreicher Softwaresysteme
hoher Qualität unter Berücksichtigung
der einzusetzenden **Arbeits-** und **Zeit**ressourcen.

4 P's im Software Engineering



- **Projekt**
  - Gesamte Koordination
- **Personal**
  - Wer entwickelt?
- **Prozess**
  - Wie wird entwickelt?
- **Produkt**
  - Was ist das Ergebnis?

V-Modell


Absichten
Aufgaben
Genutztes System
Anforderungsspezifikation
Abnahmetest
Installiertes System
Systementwurf
Systemtest
Integriertes System
SW-Grobentwurf
Integrationstest
Integrierte SW-Module
SW-Feinentwurf
Modultest
SW-Module

Anforderungsspezifikation

- **Softwareanforderung**
  - Eine Bedingung oder Fähigkeit,
  - die eine Software erfüllen oder besitzen muss,
  - um einen Vertrag, eine Norm oder ein weiteres formelles Dokument

  zu erfüllen.

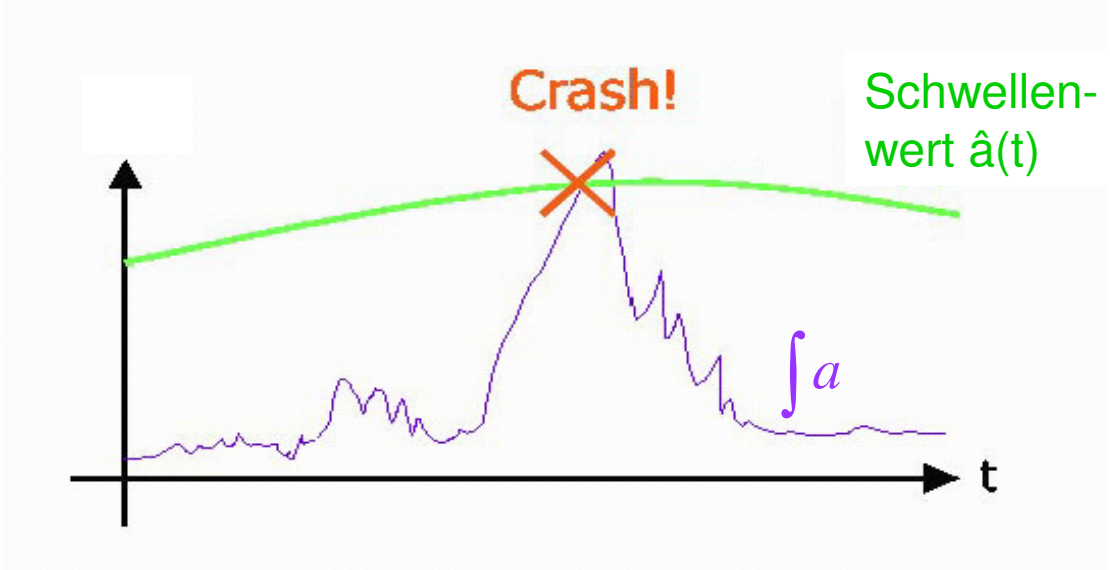
- Eine **Software-Spezifikation** ist eine Zusammenstellung
  - aller Anforderungen an eine Software
  - und der Randbedingungen für ihren Einsatz.

Aufgabe der Spezifikation

- Aufgabe der Spezifikation ist es,
  - die angestrebte Diensterbringung zu beschreiben
  - also auf folgende Fragen zu antworten
  - **WAS** soll die Software machen?
    (z.B. Sortieren)
  - bzw. **WAS** darf die Software **nicht** machen?
    (z.B. eine unsichere Aktion auslösen)

- Aufgabe der Spezifikation ist es **nicht**
  - Lösungsansätze zur Realisierung der Diensterbringung zu beschreiben
  - also etwa folgende Frage zu beantworten:
    - **WIE** soll die Software es machen?
      (z.B. Sortieren durch den QuickSort-Algorithmus)

Airbag: Informale Spezifikation der Funktionalität



(aus [3])

Informale Spezifikation: Funktion Crash gibt

- frühesten Zeitpunkt aus,
  zu dem das Integral über Beschleunigungswerten den vorgegebenen Schwellenwert überschritten hat
- -1 aus, solange der Schwellenwert nicht überschritten wurde

Formale Spezifikation

- Gegeben (aus [3])

$$a : R_0^+ \rightarrow R_0^+$$

$$\hat{a} : R_0^+ \rightarrow R_0^+$$

- Gesucht

$$crash : R_0^+ \rightarrow R_0^+ \cup \{-1\}$$

- Funktionalität

$$\left(\int_0^k a(t)dt \leq \hat{a}(k) \quad \forall k \leq t_0 \right) \Rightarrow crash(a(t_0)) = -1$$

$$\left(\int_0^k a(t)dt \leq \hat{a}(k) \quad \forall k < t_0 \right) \quad \wedge \quad \left(\int_0^{t_0} a(t)dt > \hat{a}(t_0) \right) \Rightarrow crash(a(t_0)) = t_0$$

Zusätzliche Sicherheitsanforderungen

- **Sicherheitsanforderung 1** (aus [3])
  Bei einem Frontcrash über 28 km/h
  muss der Airbag aktiviert werden

$$[frontcrash\ (a,V) \wedge V \geq 28] \Rightarrow crash\ (a) \neq -1$$

- **Sicherheitsanforderung 2**
  Bei einem Frontcrash unter 5 km/h
  darf der Airbag nicht aktiviert werden

$$[frontcrash\ (a,V) \wedge V \leq 5] \Rightarrow crash\ (a) = -1$$

- zu überprüfen, z. B.
  - mit Hilfe eines Theorembeweisers
  - aufgrund der physikalischen Eigenschaften des Wagens

Aufgaben der Entwurfsphase

- Aufgabe des Software-Entwurfs ist es, festzulegen, **WIE** die Dienste in der Software realisiert werden, also
  - die Software-Architektur zu definieren *Grobentwurf*
    - durch Zerlegung des Gesamtsystems in Komponenten und Zuordnung des Funktions- und Leistungsumfangs
    - Beschreibung der Beziehungen zwischen den Komponenten,
    - Festlegung der Schnittstellen, über die die Komponenten miteinander kommunizieren
    - in **objekt-orientierten** Systemen: Identifikation von Klassenkandidaten, jedoch **nicht** ihrer Methoden
  - die Aufgaben jeder Softwarekomponente programmiersprachenneutral zu realisieren *Feinentwurf*
    - in **objekt-orientierten** Systemen: Definition der Methoden

Menschliche Strategien

- **Abstraktion**
  - Ersetzung komplexer Strukturen durch Oberbegriffe, die ihre Bedeutung auf einer höheren Semantikstufe wiedergeben.
  - Vorteil: Verständnis solcher Strukturen jeweils einmal erforderlich

- **Teile und herrsche**
  - hohe Kohäsion — funktionale Bindung
wie eng die Aktivitäten innerhalb eines Moduls miteinander verbunden sind
  - lose Kopplung — niedriger Grad der Interaktion zwischen Modulen
wie sauber Module voneinander abgetrennt sind
  - Kapselung — Zusammenfassung von Datenstrukturen
und auf ihnen operierenden Funktionen
  - information hiding — keine Implementierungsdetails nach außen bekannt geben

Statische und dynamische Fehlerbegriffe

- **Irrtum (mentaler Fehler)** — ***mistake***
  - z. B.: Modellfehler, Unachtsamkeit, Denkfalle, Unvermögen

↓ ! (stets)

- **Produktfehler** — ***fault***
  - Abweichung zwischen Absicht und Realisierung, im Produkt, z. B. falsche Operationen, falsche Operanden

↓ ? (u. U.)

- **Fehlerhafter Zustand** — ***error***
  - Abweichung in internem Zustand, z. B. inkorrektes Zwischenergebnis

↓ ? (u. U.)

- **Versagen** — ***failure***
  - Abweichung im Verhalten z. B. falsches Ergebnis, kein Ergebnis

Maßnahmen zur Erhöhung der Qualität

- **Irrtum vermeiden**
  - Während der Entwicklung:
    Transparenz, Rigorosität, Dokumentation, formale Methoden
- **Produktfehler erkennen**
  - Nach der Entwicklung von (Zwischen-)produkten:
    (Zwischen-)prüfungen (Reviews), Testen, Beweisen

- **Fehlerhaften Zustand beheben**
  - Im Betrieb: Wiederherstellung eines akzeptablen Zustands
    durch Rückwärtsbehebung (recovery), Vorwärtsbehebung (exception)
- **Versagen maskieren**
  - Im Betrieb: Herausfilterung sporadischer inkorrekter Ergebnisse
    Tolerierung von Einzelversagen durch Redundanz und Mehrheitsbildung

Klassen von Nachweisverfahren

- **Informelles Nachweisverfahren (Review, Walkthrough)**
  - manuelle Prüfung, ob Anforderungen in Spezifikation und Entwurf richtig umgesetzt
  - ob festgelegte konstruktive Methoden bei Erstellung angewendet

- **Statische Analyse**
  - Prüfung ohne Programmausführung, ob Codierrichtlinien eingehalten
  - ob Code Inkonsistenzen (z.B. nicht initialisierte Variablen) enthält

- **Testen (Modul-, Integrations-, System- und Abnahmetest**
  - Nachweis durch Programmausführung, dass Code(teile) die spezifizierten Anforderungen bzgl. bestimmter Eingabedaten erfüllen

- **Korrektheitsbeweis**
  - formaler Nachweis, dass die codierte Funktion mit der spezifizierten logisch übereinstimmt (meist Quellcode ohne Rücksicht auf Compiler, Maschinenarithmetik, etc.)

Testen nach Phasen der SW-Entwicklung

- **Modultest**
  Prüfung auf Unstimmigkeiten zwischen Programmmodulen und ihren im Software-Feinentwurf festgelegten Funktionen und Schnittstellen

- **Integrationstest**
  Prüfung des Zusammenspiels zwischen den Modulen

- **Systemtest**
  Prüfung auf Konsistenz zwischen der tatsächlichen und der im Systementwurf verlangten Leistungserbringung; umfasst i.A.
  - Vollständigkeitstest — Checken, ob alle Funktionen implementiert
  - Volumentest — umfangreichem Datenvolumen aussetzen
  - Stresstest — schwerer Beanspruchung unterziehen
  - Leistungstest — Performance-Attribute prüfen: Antwortzeiten, Durchsatz, Speicherplatz, zulässige Parameter...

- **Abnahmetest**
  Prüfung auf Erfüllung der spezifizierten Anforderungen. Wird ausgeführt vom Auftraggeber selbst oder von einem unabhängigen SQA Team im Auftrag des Kunden.

Black- und White-box-Test

- **Black-box-Test (data-driven, functional, input/output driven)**
  - geht nicht von Kenntnis der Struktur aus
  - ignoriert Code
  - funktional, anhand der Anforderungsspezifikation

- Frage
  - wurden alle Anforderungen implementiert?
  - Beispiel: Systemtest

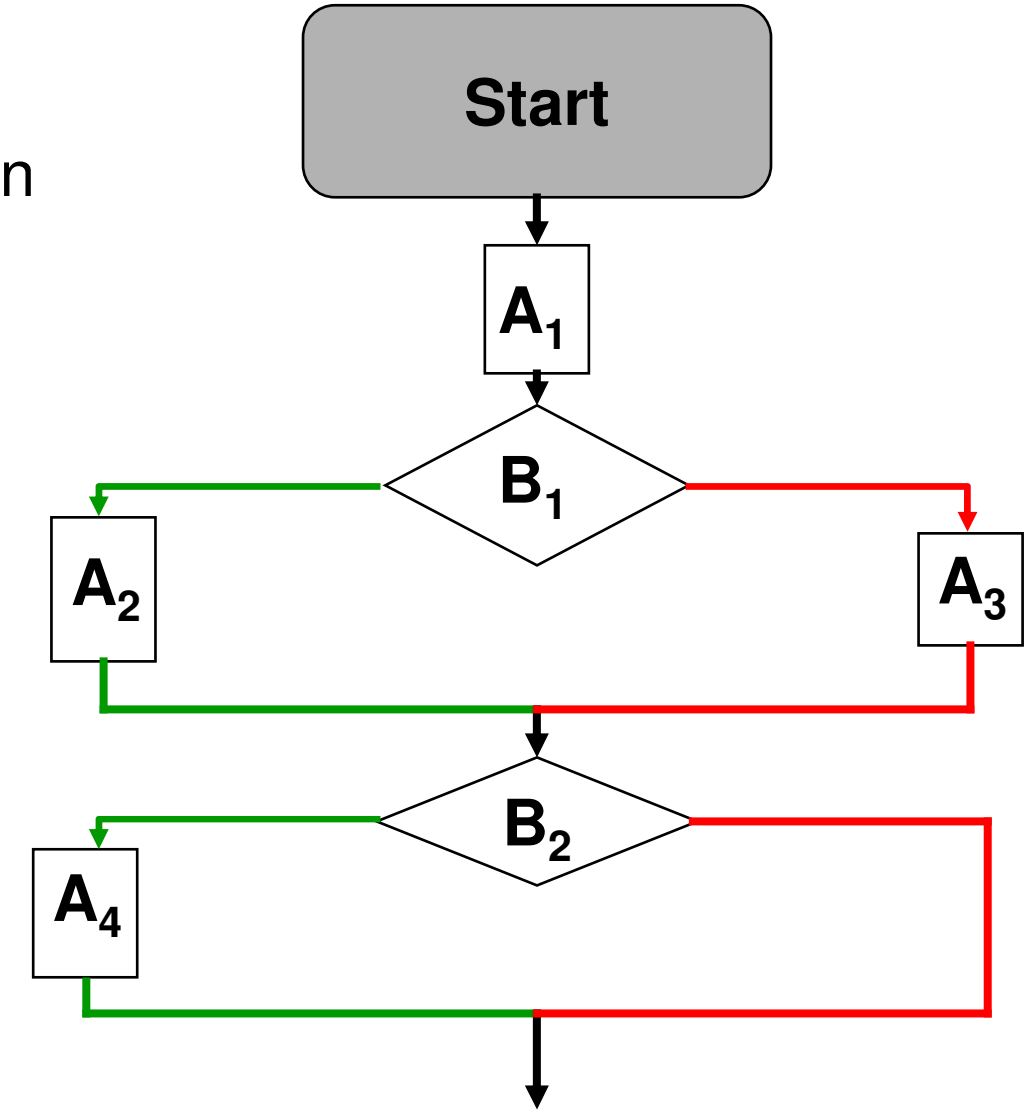
- **White-box-Test (glass-box, logic-driven, structured, path-oriented)**
  - geht von der inneren Struktur aus
  - ignoriert Spezifikation
  - strukturell, anhand von Kontroll- bzw. Datenflüssen

- Frage
  - wurden nur die Anforderungen implementiert?
  - Beispiel: Zweigüberdeckungstest

Beispiel: Zweigüberdeckung

- Menge von Testfällen, die alle Verzweigungen durchlaufen
  - Tool erforderlich
- Anweisungen: A_1, A_2, A_3, A_4
- Fallunterscheidungen: B_1, B_2
- Beispiel:

1. **Pfad: A_1 A_2 A_4**
2. **Pfad: A_1 A_3**



Verlässlichkeitsattribute

- **Korrektheit**
  - Abwesenheit von systematischen Fehlern

- **Zuverlässigkeit** — Kontinuität korrekter Diensterbringung
  - Überlebenswahrscheinlichkeit — *reliability*
  - Durchschnittliche Lebensdauer — *MTTF = mean time to failure*

- **Sicherheit** — Schutz vor Schaden
  - Technische Sicherheit — *safety*
    Schutz der Systemumgebung vor Rechnerversagen
  - Informationssicherheit — *security*
    Schutz des Rechensystems vor Systemangriffen

- **Verfügbarkeit**
  - Funktionsbereitschaft — *availability*
  - MTTF / total time — *mean up time / total time*

Standards / Richtlinien

- **International Standard Organization (ISO)** *ISO 9000-3*
  "Quality Management and Quality Assurance Standards"
  Part 3: Guidelines for the application of ISO-9001 to the development, supply and maintenance of software

- **Deutsches Institut für Normung (DIN)** *DIN V 19250*
  "Messen - Steuern - Regeln, grundlegende Sicherheitsbetrachtungen für MSR-Schutzeinrichtungen"

- **International Electrotechnical Commission (IEC)** *IEC 61508*
  "Functional Safety of Electrical / Electronic / Programmable Electronic Safety-Related Systems"

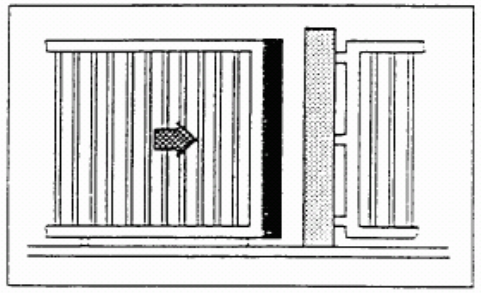
- **Motor Industry Software Reliability Association (MISRA)**
  "Development Guidelines for Vehicle-Based Software"
  published by Motor Industry Research Association (MIRA)

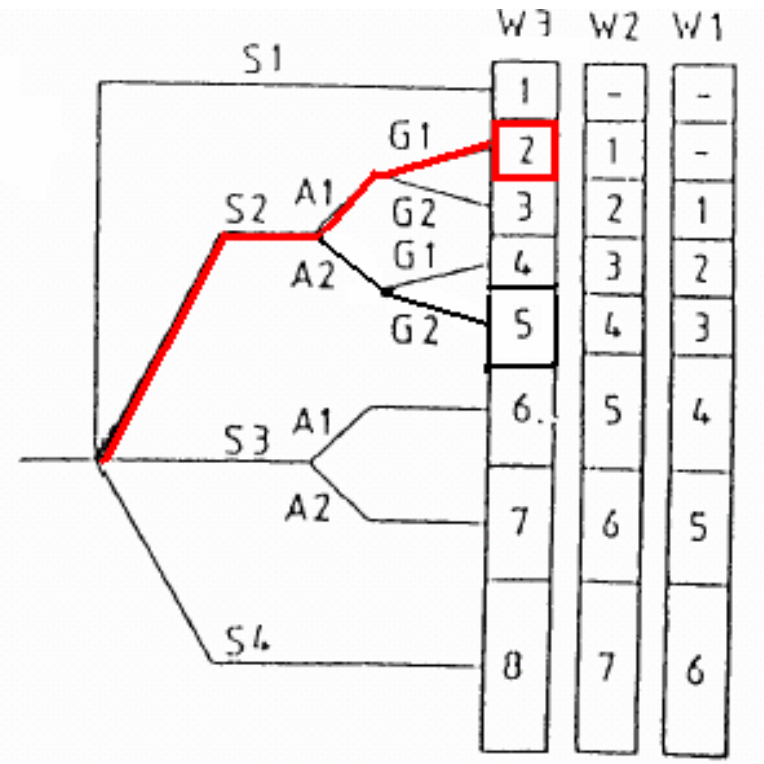
Risikoparameter nach DIN 19250

- **Schadensausmaß**
  - S1 leichte Verletzung
  - S2 schwere irreversible Verletzung einer oder mehrerer Personen oder Tod einer Person
  - S3 Tod mehrerer Personen
  - S4 katastrophale Auswirkung, sehr viele Tote
- **Aufenthaltsdauer**
  - A1 selten (bis öfter)
  - A2 häufig bis dauernd
- **Gefahrenabwendung**
  - G1 möglich unter bestimmten Bedingungen
  - G2 kaum möglich
- **Eintrittswahrscheinlichkeit des unerwünschten Ereignisses**
  - W1 sehr gering
  - W2 gering
  - W3 relativ hoch

Beispiel: Kraftbetätigte Fenster, Türen, Tore

- **Schadensausmaß** S2
  da Quetsch- und Scherstellen
  Ursache für schwere und tödliche Verletzungen
  sein können
- **Aufenthaltsdauer** A1
  da sich Personen in dem Bereich
  selten und kurz aufhalten
- **Gefahrenabwendung** G1
  gefährdete Personen können sich i.a.
  aus dem vom bewegten Flügel gebildeten
  Bereich entfernen
- **Eintrittswahrscheinlichkeit** W3
  höchste Klasse konservativ unterstellt





Es ergibt sich Anforderungsklasse 2

Integritätsniveaus nach MISRA

(aus [1])

Controllability Categories	Definition	SIL
Uncontrollable	This relates to failures whose effects are not controllable by the vehicle occupants, and which are most likely to lead to extremely severe outcomes. The outcome cannot be influenced by a human response.	4
Difficult to Control	This relates to failures whose effects are not normally controllable by the vehicle occupants but could, under favourable circumstances, be influenced by a mature human response. They are likely to lead to very severe outcomes.	3
Debilitating	This relates to failures whose effects are usually controllable by a sensible human response and, whilst there is a reduction in the safety margin, can usually be expected to lead to outcomes which are at worst severe.	2
Distracting	This relates to failures which produce operational limitations, but a normal human response will limit the outcome to no worse than minor.	1
Nuisance Only	This relates to failures where safety is not normally considered to be affected, and where customer satisfaction is the main consideration.	0

MISRA The Motor Industry Software Reliability Association

(aus [1])

Entwicklungsmethoden vs. Integritätsniveaus (SILs)

Development Process	Integrity Level				
	0	1	2	3	4
Specification and design	ISO9001	Structured method.	Structured method supported by CASE tool.	Formal specification for those functions at this level.	Formal specification of complete system. Automated code generation (when available).
Languages and compilers		Standardized structured language.	A restricted subset of a standardized structured language. Validated or tested compilers (if available).	As for 2.	Independently certified compilers with proven formal syntax and semantics (when available).
Configuration management: products		All software products. Source code	Relationships between all software products. All tools.	As for 2.	As for 2.
Configuration management: processes		Unique identification. Product matches documentation. Access control. Authorized changes.	Control and audit changes. Confirmation process.	Automated change and build control. Automated confirmation process.	As for 3.

MISRA Guidelines 2

(aus [1])

Development Process	Integrity Level				
	0	1	2	3	4
Testing		Show fitness for purpose. Test all safety requirements. Repeatable test plan.	Black box testing.	White box module testing - defined coverage. Stress testing against deadlock. Syntactic static analysis.	100% white box module testing. 100% requirements testing. 100% integration testing. Semantic static analysis.
Verification and validation		Show tests: are suitable; have been performed; are acceptable; exercise safety features. Traceable correction.	Structured program review. Show no new faults after corrections.	Automated static analysis. Proof (argument) of safety properties. Analysis for lack of deadlock. Justify test coverage. Show tests have been suitable.	All tools to be formally validated (when available). Proof (argument) of code against specification. Proof (argument) for lack of deadlock. Show object code reflects source code.
Access for assessment		Requirements and acceptance criteria. QA and product plans. Training policy. System test results.	Design documents. Software test results. Training structure.	Techniques, processes, tools. Witness testing. Adequate training. Code.	Full access to all stages and processes.

Fehlerbaumanalyse

- weit verbreitete Technik zur
  - **qualitativen** und
  - **quantitativen** Analyse

  komplexer technischer Systeme mit Sicherheitsrelevanz

- Im Rahmen eines **deduktiven** Prozesses wird
  - für ein angenommenes unerwünschtes Ereignis (etwa für den Ausfall einer bestimmten Systemfunktion)
  - nach allen **Ursachen** gesucht, die zu diesem Ereignis führen können.

- Die Ereignisse und ihre Ursachen werden
  - **top-down** ermittelt und
  - mittels des **Fehlerbaums**

  graphisch dargestellt

Top-Ereignis

- Angefangen mit einem unerwünschten Ereignis (Top Event) werden rekursiv Ereignisse identifiziert, deren
  - **einzelnes** Eintreten oder
  - **gleichzeitiges** Eintreten

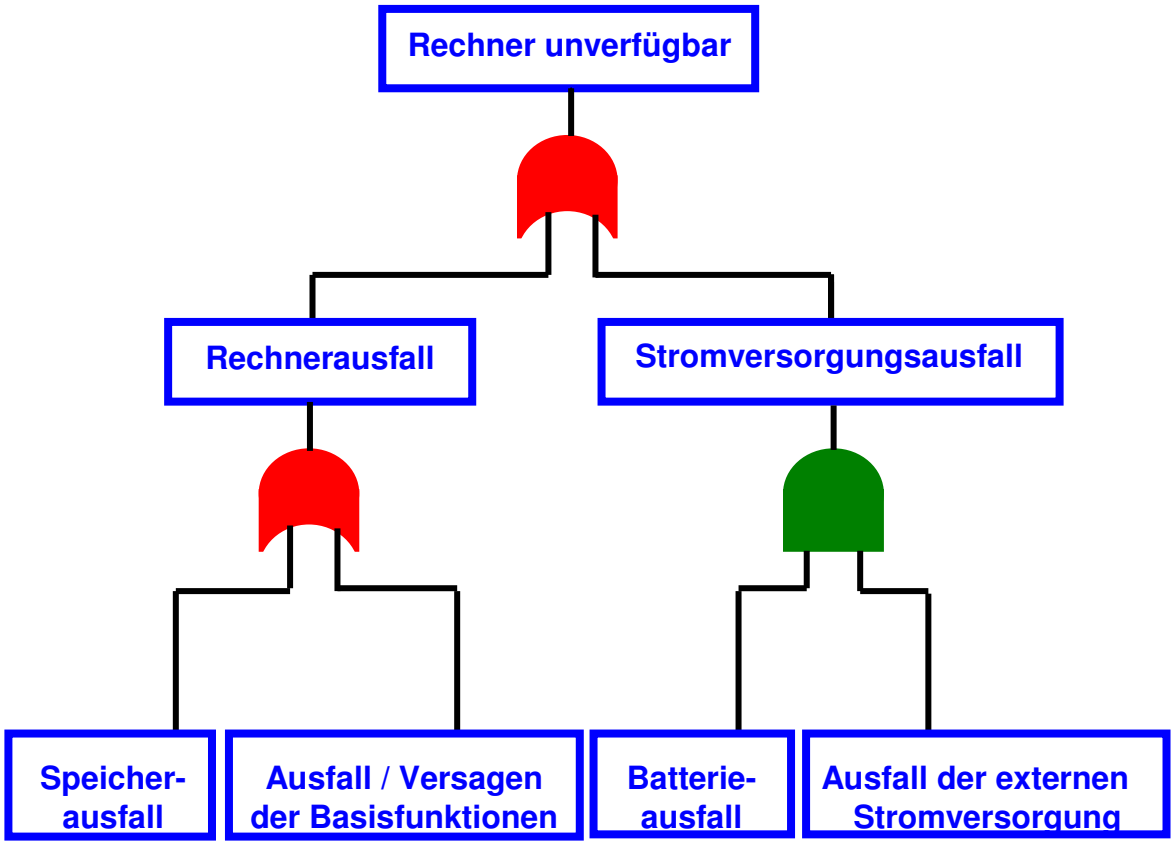
  zu dem jeweils betrachteten Ereignis führen.

- Auf jeder Hierarchieebene werden normierte Symbole für
  - die **ODER-Verknüpfung (Disjunktion)** bzw.
  - die **UND-Verknüpfung (Konjunktion)**

  verwendet, die angeben, ob
  - bereits **das einzelne Eintreten** der Teilereignisse
  - oder erst **ihr gemeinsames Eintreten**

  ausreichend ist, um zum übergeordneten Ereignis zu führen.

Beispiel: Top-Ereignis, 1. + 2. Rekursionstiefe



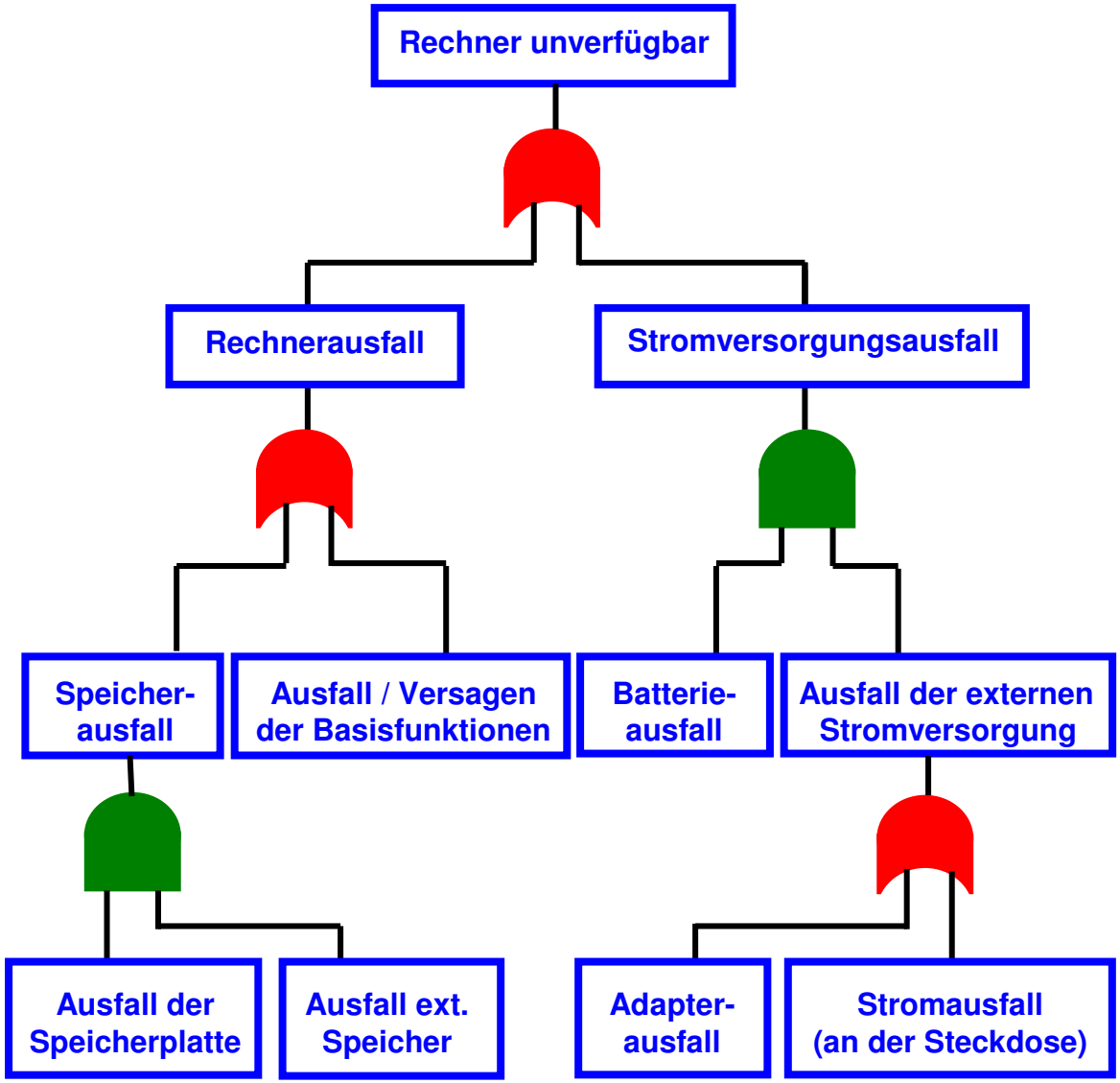
- Rechner unverfügbar lässt sich darstellen als **Disjunktion**
  - Rechnerausfall

    lässt sich darstellen als **Disjunktion**
    - Speicherausfall
      oder
    - Ausfall / Versagen der Basisfunktionen

    oder
  - Stromversorgungsausfall

    lässt sich darstellen als **Konjunktion**
    - Batterieausfall
      und
    - Ausfall der externen Stromversorgung

Beispiel: 3. Rekursionstiefe



- Speicherausfall

  lässt sich darstellen als **Konjunktion**

  - Ausfall der Speicherplatte
    und
  - Ausfall externer Speicher

- Ausfall externer Stromversorgung

  lässt sich darstellen als **Disjunktion**

  - Adapterausfall
    oder
  - Stromausfall

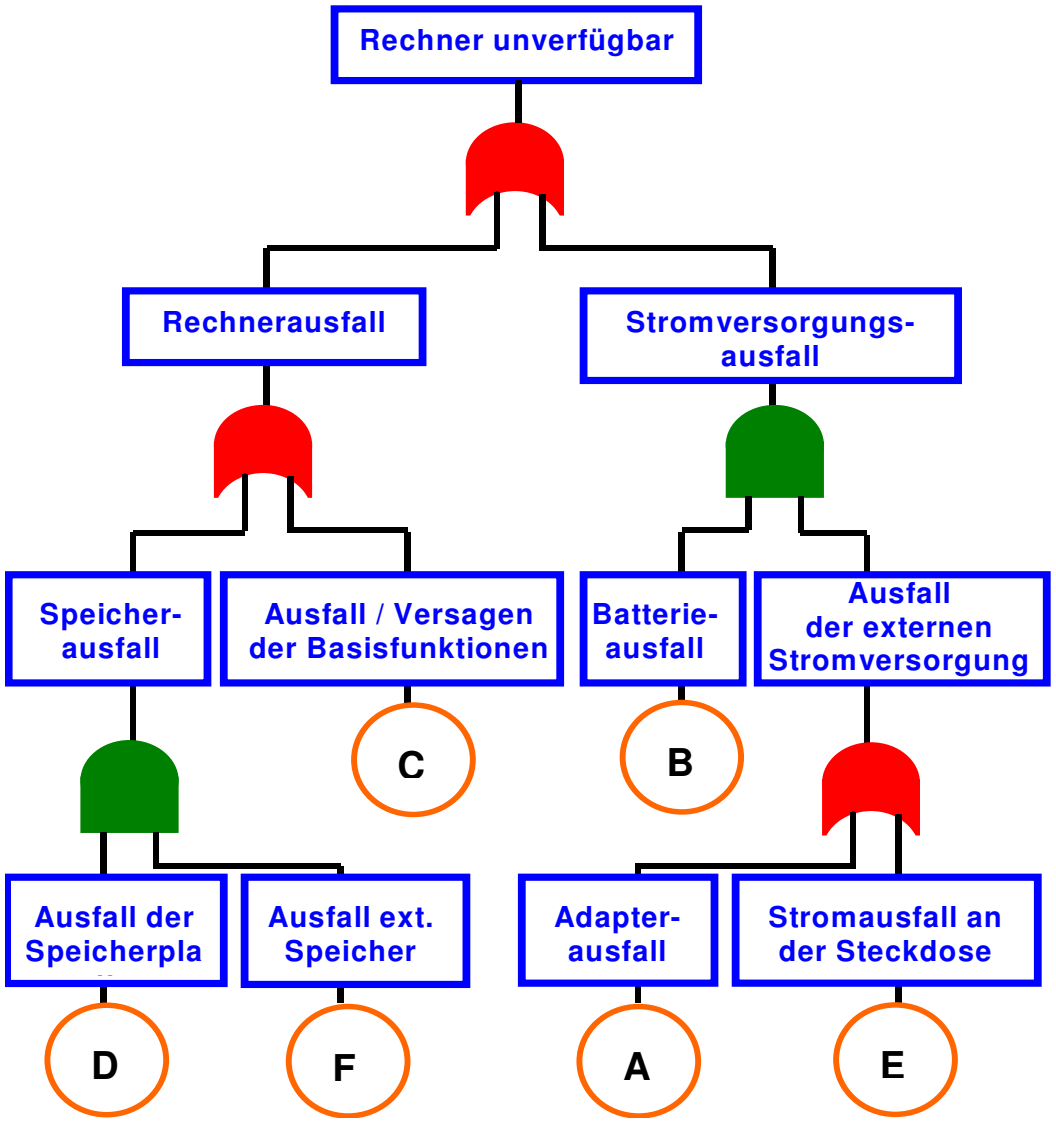
Basisereignisse

- Die Analyse endet auf der Ebene der sogenannten
  - **Basisereignisse**, die vom jeweiligen Auflösungsgrad abhängt; i. a. auf Komponentenebene

- Basisereignisse sind meistens Ausfälle
  - aktiver und passiver **Komponenten**, etwa
  - Ventile, Schalter, Pumpen, Röhre, Menschen, **Software**

- Die Analyse ist damit an sich ein
  - **vollständiges** Verfahren, das allerdings
  - neben dem gewählten **Auflösungsgrad** sicher auch
  - vom **Wissensstand** und der Erfahrung der Analysierenden abhängt

Beispiel: Basisereignisse



- **A: A**dapter - Ausfall
- **B: B**atterie - Ausfall
- **C:** Ausfall / Versagen der Rechner-Basisfunktionen (**C**PU, Keyboard, Bildschirm)
- **D:** Ausfall der Speicherplatte (Hard **D**isk)
- **E:** Ausfall der **E**lektrischen Stromversorgung
- **F:** Ausfall externer Speicher (**F**loppy Disk, …)

Fehlerbaumanalyse *fault tree analysis*

- nach der Fehlerbaumerstellung interessiert man sich für minimale Mengen von Basisereignissen, deren Kombination zum Top-Ereignis führt:
  - die sogenannten **minimalen Schnittmengen**

- Ihre Kenntnis kann in doppelter Hinsicht hilfreich sein:
  - **a priori**
    um die Auslegung einer neu zu planenden Anlage durch fehlerbeherrschende Maßnahmen zu verbessern
  - **a posteriori**
    um Zuverlässigkeitskenngrößen eines vorliegenden Systems durch geschätzte Eintrittswahrscheinlichkeiten der Basisereignisse probabilistisch zu bewerten.

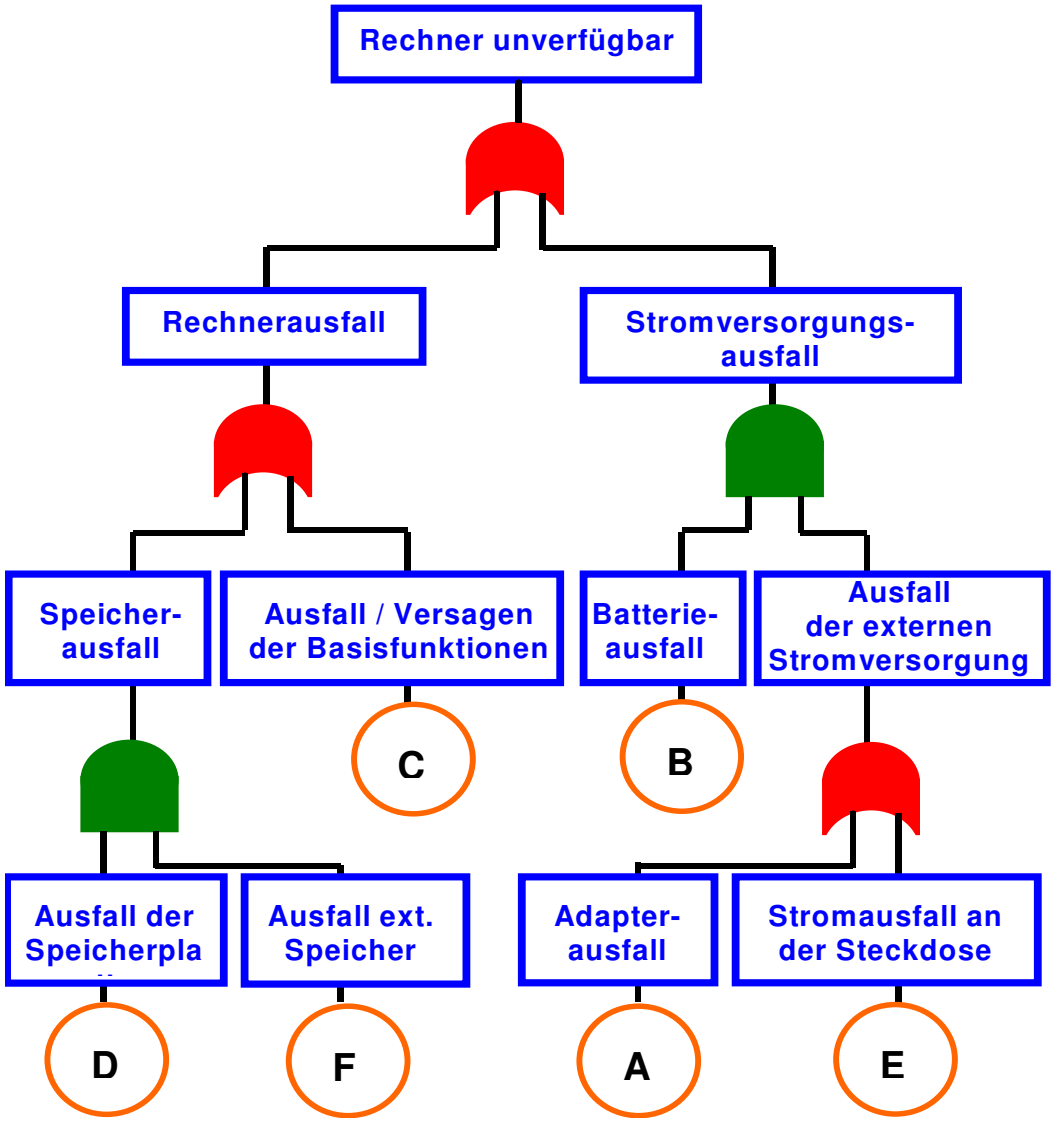
Boolesche Funktion eines Fehlerbaums

- Sei X die Menge aller Basisereignisse, i von 1 bis n
  - $X := \{X_i \mid X_i \text{ Basisereignis}, 1 \leq i \leq n\}$

- f bezeichnet die Abbildung,
  - die jeder eintretenden Basisereigniskombination den Wahrheitswert der Aussage zuweist,
  - ob unter den vorgegebenen Bedingungen das Top-Ereignis eintritt

- $f: P(X) \rightarrow \{0,1\}$, $f(C) := < C \Rightarrow \text{Top-Ereignis} >$
  - wobei
    P(X) die Potenzmenge (alle Teilmengen) von X bezeichnet

Beispiel: Boolesche Funktion f



- X = {A,B,C,D,E,F}
- n= 6
- f (A,B,C,D,E,F) =
  = D ∧ F ∨ C ∨ B ∧ (A∨ E)

Schnittmengen *cut sets*

- Definition: eine Kombination C von Basisereignissen heißt Schnittmenge, genau dann wenn C zum Top-Ereignis führt

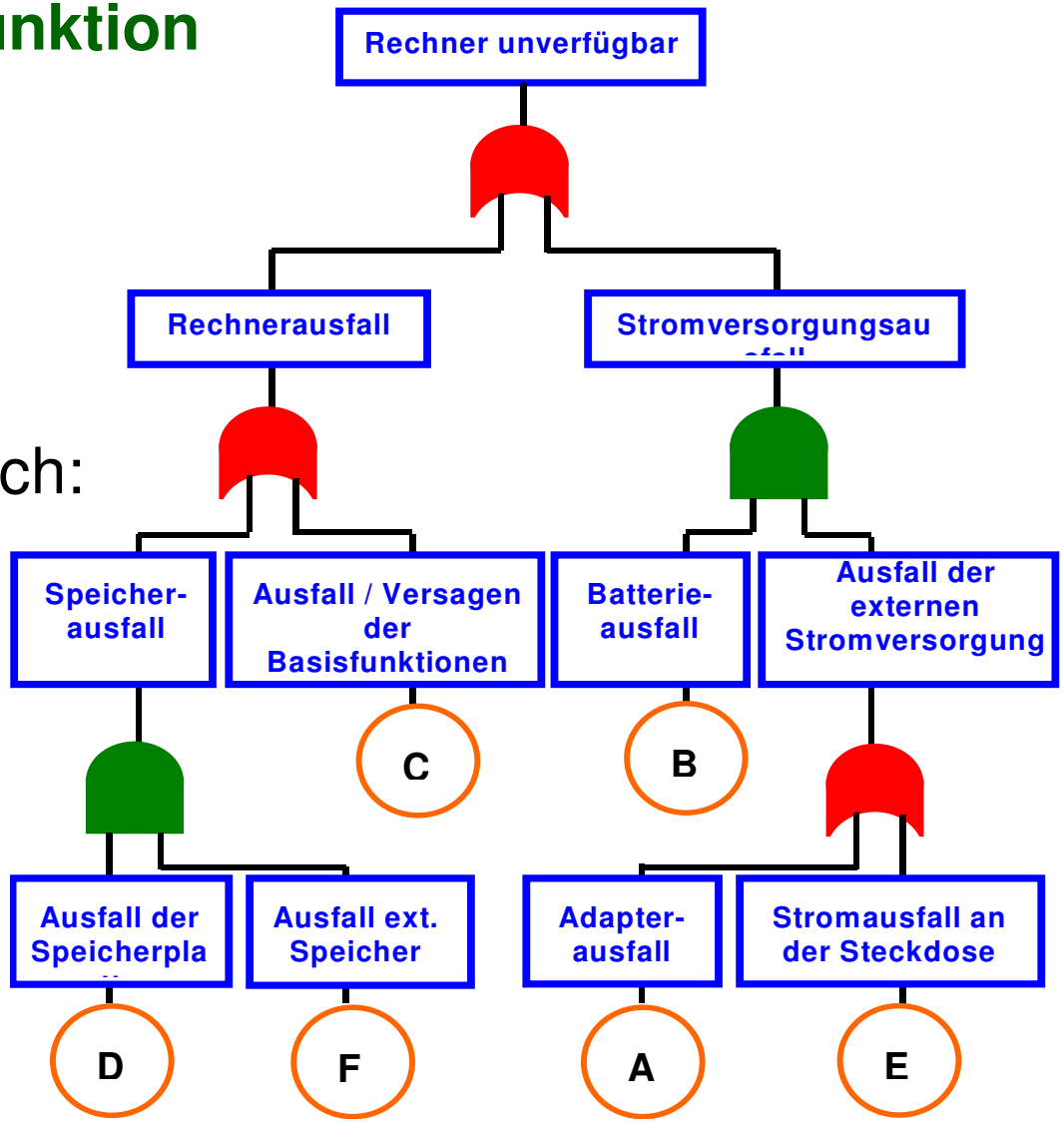
- Formal:
  - f(C) = 1 bzw.
  - die Aussage: C $\Rightarrow$ Top-Ereignis ist wahr

- Definition: eine Schnittmenge heißt minimal,
  wenn keine echte Teilmenge davon eine Schnittmenge ist,

- Formal:
  - f(C) = 1 und
  - $\forall$ C‘ mit: (C‘$\subset$ C $\wedge$ C’$\neq$C) gilt: f(C') = 0

Beispiel: Bestimmung der Schnittmengen

- bereits ermittelte **Boolesche Funktion**
  f (A,B,C,D,E,F) =
  = D ∧ F ∨ C ∨ B ∧ (A ∨ E) =
- umgeformt:
  = D ∧ F ∨ C ∨ B ∧ A ∨ B ∧ E
- Daraus hier unmittelbar ersichtlich:
- 4 minimale Schnittmengen
  - {D, F} hard disk & floppy
  - {C} CPU
  - {B, A} Batterie & Adapter
  - {B, E} Batterie & Strom



Ermittlung minimaler Schnittmengen

- Boolesche Funktionen lassen sich unterschiedlich darstellen, etwa
  - durch **disjunktive Normalformen**
  - oder **Wahrheitswertetafeln**

- zu solchen Darstellungen gibt es klassische Verfahren
  - zur Ermittlung minimaler Schnittmengen,
    die i. w. auf **algebraischen Umformungen**
    boolescher Ausdrücke beruhen;

- Effizientere Alternative:
  - Reduced Ordered **Binary Decision Diagrams** (ROBDDs)
    besonders kompakte Darstellungen Boolescher Funktionen

Zitierte Materialien

- [1] R. S. Rivett: “Emerging Software Best Practice and how to be Compliant“, Rover Group Ltd.
- [2] F. Piwonka: “Mechatronic Systems Engineering – from Methodology to Education“, Bosch GmbH
- [3] W. Reif: “Beweisbar korrekte Software“, Universität Augsburg