11 Zuverlässigkeit und Sicherheit

- Echtzeitsysteme stellen erhöhte Anforderungen an Zuverlässigkeit und Sicherheit zur Vermeidung von wirtschaftlichem Schaden und Personengefährdung
- Neben der Perfektionierung von Maßnahmen zur Fehlervermeidung werden Fehlertoleranzeigenschaften gefordert

11.1 Definitionen

- Zuverlässigkeit
  (dependability, aber auch reliability)
  Fähigkeit eines technischen Systems, die vorgesehenen Aufgaben unter bestimmten Betriebsbedingungen während der vorgegebenen Zeitspanne zu erfüllen

- Sicherheit
  (safety)
  Fähigkeit eines technischen Systens, Gefährdung und Schaden in bestimmten Betriebszuständen zu vermeiden
- Fehler (Defekt, error, fault)
  Nichterfüllung vorgegebener Anforderungen (Fehler als Zustand)
- Ausfall (failure)
  Ereignis, das Übergang vom fehlerfreien in fehlerbehafteten Zustand bewirkt
- Fehler transient, permanent
- Zuverlässigkeit durch Fehler, dh. Fehlzustände (errors, faults) oder Funktionsausfälle (faults, failures) beeinträchtigt
- Überlebenswahrscheinlichkeit R(t)
  (reliability)
  als Maßzahl ist die Wahrscheinlichkeit, daß die zunächst funktionsfähige Einheit im Zeitraum 0 bis t ohne Reparaturen funktionsfähig bleibt
  Komplement: Ausfallwahrscheinlichkeit

- MTBF (mean time between failures)
  mittlerer Ausfallabstand $= \frac{\Sigma Betriebszeiten}{\Sigma Ausfaelle}$
  Ausfallrate $= \frac{1}{MTBF}$
- MTTR (mean time to repair)
  mittlere Instandsetzungsdauer
  Instandsetzungsrate $= \frac{1}{MTTR}$
- Verfügbarkeit (availability)
  Wahrscheinlichkeit, daß eine reparierbare Einheit zum Zeitpunkt t funktionsfähig ist
  p $= \frac{MTBF}{MTBF+MTTR}$
- Erhöhung der Zuverlässigkeit
  Maßnahmen gegen Fehler aus dem System selbst durch Erhöhung der Verfügbarkeit
- Erhöhung der Sicherheit
  Neben Erhöhung der Zuverlässigkeit noch Maßnahmen gegen gefährliche Auswirkungen; Verminderung des Restrisikos; Prüfdokumente als Nachweis gefordert
- Fehlertoleranz
  Fähigkeit eines Systems, trotz Auftretens einer begrenzten Anzahl von Fehlern seine Funktionen zu erfüllen; wichtig für hochzuverlässige und wartungsarme (–freie) Systeme.

- Verfügbarkeit vs. Sicherheit:
  - ⋆ Auto, das niemals fährt, ist sicher (es geht keine Gefahr von ihm aus), es ist aber nicht brauchbar
  - ⋆ Auto, das immer fahren kann (auch wenn alle Bremsen defekt), hat höchste Verfügbarkeit, ist aber nicht sicher
- Sicherheit muß folgende Fehlerquellen berücksichtigen
  - ⋆ Spezifikationsfehler
  - ⋆ Implementierungsfehler
  - ⋆ Ausfälle Hardware, Geräte usw.
  - ⋆ Bedienungsfehler
  - ⋆ unerwartete Einflüße oder Störungen durch die Umgebung

- Bausteine
  - ⋆ Erkennen von Fehlern, z.B. durch Plausibilität, Redundanz
  - ⋆ Maskieren von Fehlern durch Redundanz
  - ⋆ Begrenzung der Ausbreitung von Fehlern
  - ⋆ Behandlung von Fehlern
  - ⋆ fail-save-Techniken (HW und SW), z.B.
    - ◦ Ventil schließt automatisch bei Stromausfall oder Ausbleiben von Anweisungen des Prozeßrechners
    - ◦ Relais fällt durch Federkraft ab
    - ◦ Überdrucksicherheitsventil

11.2 Fehlertolerante Systeme

(fehlertolerierende Systeme, fault tolerant systems)

- trotz beliebiger (oder nur begrenzter) Anzahl von gleichzeitig auftretenden Fehlern wird (evtl. reduzierte) Funktion erfüllt
- Fehlertoleranzverfahren gibt an, welche und wieviele gleichzeitig auftretende Fehler durch entsprechende Maßnahmen kompensiert werden können
- Fehlermodell beschreibt
  - ⋆ Menge der Komponenten und Funktionen des Systems
  - ⋆ Menge der Fehlermöglichkeiten
  - ⋆ beschreibt Menge der tolerierten Fehler aus der Menge der Fehlermöglichkeiten
  - ⋆ von Fehler betroffene Komponenten und deren Beeinträchtigung
  - ⋆ Problem: An alle Fehler gedacht?
  - ⋆ Problem: Fehler der Umgebung auch einbezogen?

- ⋆ Problem: Wie wahrscheinlich sind die Fehler?
- ⋆ Problem: Soll man den Fehler berücksichtigen oder nicht?
- ⋆ je nach Parameter: Optimistische oder pessimistische Modelle

- Arten von Fehlermodellen
  - ⋆ Komponenten-Liste
    - ◦ Annahme: jeder Fehler einer Komponente ergibt Systemfehler
    - ◦ also Wahrscheinlichkeit, daß System intakt, gleich Wahrscheinlichkeit, daß keine Komponente ausgefallen
    - ◦ System toleriert also keine Fehler
  - ⋆ Kombinatorische Modelle
    - ◦ Beispiel: Fehlerbaum
    - ◦ Attributierung mit der Wahrscheinlichkeit des Zustandes
    - ◦ Erweiterung: explizite Formel zur Verknüpfung der Folgeknoten, um z.B. "beliebige 2 von 3 Knoten dürfen ausfallen", einfach darstellen zu können
  - ⋆ Automaten, Markoff-Modelle

- dynamisches Modell
- enthält u.a. auch Reparaturen, Rekonfigurationen
- relevante Systemzustände werden mit ihren Ubergängen modelliert
- Systemzustände sind gegeben durch die Zustände der Einzelkomponenten, die ausfallen können
- der Systemzustand enthält auch die Angabe, ob das Gesamtsystem arbeitsfähig ist oder nicht
- Übergangswahrscheinlichkeiten festzulegen
- Zeitdauern, z.B. für Reparaturen, festzulegen
- Auswertung numerisch oder simulativ
- Ergebnis z.B. Wahrscheinlichkeit, daß System in einem arbeitsfähigen Zustand ist

- Grundanforderungen beim Auftreten von Fehlern
  - ⋆ "Überleben" beim Ausfall der 1. Komponente
  - ⋆ stoßfreier Übergang (evtl. mit Leistungsreduzierung)
  - ⋆ automatisches Neukonfigurieren
- Fehlertolerante Systeme sind normalerweise redundante Systeme, d.h. zusätzliche Komponenten bzw. Funktionen eingesetzt, dh. mehr technische Mittel als für das funktionierende System allein benötigt würden
- Arten der Redundanz

(1) Strukturelle Redundanz

- ⋆ zusätzliche, im fehlerfreien Fall entbehrliche (Hardware-)Komponenten

(2) Funktionelle Redundanz

- ⋆ Umschalten auf gleichartige Ersatzkomponente
- ⋆ Diversität: verschieden implementierte Komponenten oder Funktionen

- ⋆ Zusatzinformationen zur Wiederherstellung der Konsistenz

(3) Zeitredundanz

- ⋆ Wiederholung
- ⋆ Schwache Auslastung erlaubt zusätzliche Aktionen in einem funktionell redundanten System

(4) statische Redundanz: die zusätzliche Komponente ist dauernd aktiv

(5) dynamische Redundanz: die redundanten Komponenten werden erst im Fehlerfall aktiviert (auch ggf. Abwerfen von Last oder unwichtigeren Aufgaben)

- Umfang der notwendigen Fehlertoleranz anwendungsabhängig
- beliebig hohe Fehlertoleranz = beliebig hohe Kosten
- Zuverlässigkeits- und Sicherheitsanalyse mit Kosten-Nutzen-Abschätzung erforderlich

11.3 Prüfung und Qualitätssicherung von Steuerungssoftware

- Kategorien von Normen
  - ⋆ Begriffsnormen
  - ⋆ Produktnormen
  - ⋆ Prozeßnormen (Verfahren und Vorgehensweisen z.B. bei der Erstellung von Software)
  - ⋆ Verfahrensnormen (Regeln für die Überprüfung der Einhaltung einer bestimmten Norm)
- Normen und Richtlinien zur Qualitätssicherung von Steuerungssoftware
  - ⋆ Begriffsnormen
    - ◦ Software-Qualitätsmerkmale DIN 6627
    - ◦ Qualitätssicherung DIN ISO 8402
    - ◦ Begriffe der Sicherheitstechnik DIN VDE 31000-2
  - ⋆ Produktnormen
    - ◦ Software-Produkt DIN ISO IEC 12119
    - ◦ Benutzerdokumentation DIN 66230

- ⋆ Prozeßnormen für Software-Lebenszyklus
  - ◦ Sicherheitssoftware IEC CD 1508-1 bis IEC CD 1508-7, insbesondere IEC CD 1508-3
  - ◦ Steuerungssoftware DIN V VDE 0801
  - ◦ Kerntechnik DIN IEC 880
  - ◦ Luftfahrt RTCA/DO 178B
- ⋆ Prozeßnormen für Qualitätsmanagement
  - ◦ Entwicklung DIN EN ISO 9001
    gilt universell für materielle Güter, Dienstleistungen, Software
    enthält organisatorische, administrative und technische Bestandteile eines Qualitätsmanagements
    nur Vorschrift, wann was durchzuführen, aber nicht wie (Werkzeuge, Methoden)
  - ◦ Software DIN ISO 9000-3
- ⋆ Verfahrensnormen
  - ◦ Audits DIN ISO 10011

11.4 Fehlererkennung und –behandlung

- Erkennung Ziel: Offenlegung und Lokalisierung mit kurzer Latenzzeit
- Fehlererkennung
  - ⋆ Plausibilitätskontrollen
  - ⋆ Redundanz
- Fehlerausgrenzung
  - ⋆ fehlerhafte Teile werden entfernt
  - ⋆ problematisch ist Zusammenhang mit anderen Komponenten
  - ⋆ Rekonfigurierung ersetzt Funktionen durch neue fehlerfreie Komponenten
- Fehlerkorrektur
  - ⋆ Komponente wird in einen fehlerfreien Zustand überführt
  - ⋆ Vorwärtsbehebung:
    Erstellen eines konsistenten Zustands aus der Momentansituation, z.B. durch Ausnahmebehandlung

- ⋆ Rückwärtsbehebung:
  Rücksetzen (”roll back”) auf einen fehlerfreien Zustand der Vergangenheit (Rücksetzpunkte, checkpoints, recovery points). Problem: Domino–Effekt mit mehrfachem Rücksetzen

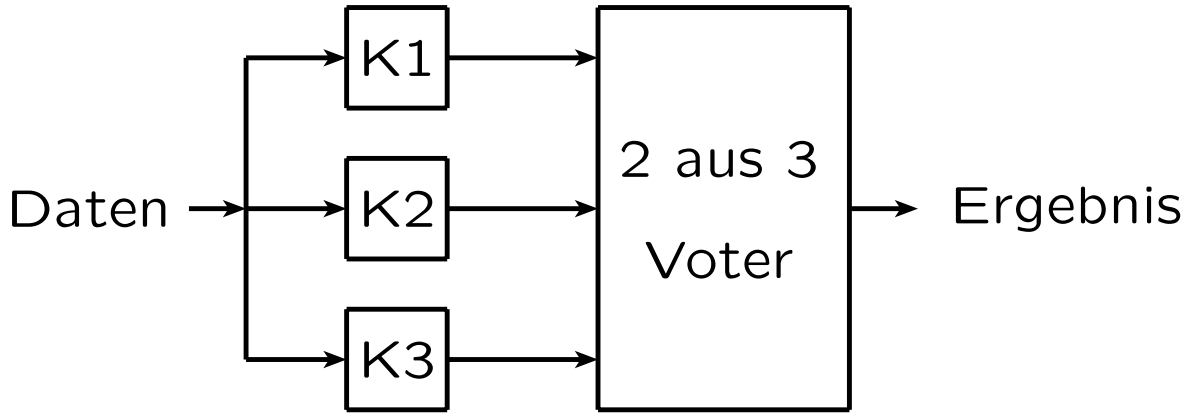
- Fehlerkompensierung
  - ⋆ verhindert die Ausbreitung der negativen Fehlerwirkung
  - ⋆ Fehlzustand bleibt und muß später behoben werden

- Begrenzung der Fehlerausbreitung
  - ⋆ Fehler soll sich nicht auf andere Komponenten ausbreiten, bevor er erkannt wird
  - ⋆ In einem hierarchischen Software–Schichtenmodell sollten Fehler, die in einer Schicht auftreten, nicht in höhere Schichten aufsteigen

11.5 Hardware–Redundanz

- zusätzliche parallele Komponenten
- Statische Redundanz
  - ⋆ mitlaufende Reserve (hot stand-by)
    - ◦ redundante, aktive (mitlaufende) Komponenten
    - ◦ bei Ausfall wird sofort umgeschaltet
    - ◦ Beispiel: Doppelrechnersystem
      2 unabhängige Rechner erfassen mit unabhängigen Geräten die gleichen Daten, jedoch steuert nur einer. Bei einem Fehler wird die Steuerung sofort auf den 2. Rechner geschaltet.
  - ⋆ Fehlermaskierende Struktur
    - ◦ N Komponenten arbeiten parallel (N=2n+1, n=1,2)
    - ◦ mit 1 Voter
    - ◦ selten Voter wieder redundant
    - ◦ Ergebnisse werden votiert (Mehrheitsergebnis gilt)

- Beispiel:TMR Triple Modular Redundancy



K1, K2, K3 sind Hardwarekomponenten, die ausfallen können
der Voter ist gegenüber den Komponenten Ki hoch zuverlässig

⋆ Vorteil: Sofortiges Weiterarbeiten bei Ausfall

⋆ Probleme: Kosten, synchrones Arbeiten der Systeme

- Dynamische Redundanz (cold standby, kalte Reserve)
  - ⋆ erhöht die Verfügbarkeit
  - ⋆ passive Reservekomponenten werden nach Ausfall zugeschaltet
  - ⋆ ersetzen nach einiger Zeit ausgefallene Komponente
  - ⋆ Vorteil: kostengünstig
  - ⋆ Nachteil: Trägheit, Reduktion der Gesamtleistung im Fehlerfall
- Hybride Redundanz
  - ⋆ z.B. statisch redundantes 2 aus 3–System
  - ⋆ und 2 passive Komponenten mit Rekonfigurations–Steuergerät

11.6 Zuverlässige Software

- Software enthält inhärent Fehler
  (kein fail safe–Übergang):

  (1) Spezifikationsfehler
  die Aufgabe ist unvollständig, nicht eindeutig oder falsch beschrieben

  (2) Entwurfsfehler
  der Entwurf erfüllt nicht die Spezifikation bzw. ist in sich widersprüchlich oder falsch

  (3) Implementierungsfehler
  Abweichung vom Entwurf, logische Fehler

  (4) Werkzeugfehler
  Übersetzer, Binder, Betriebssystem oder die Hardware enthalten Fehler

  (5) Betriebsfehler
  Fehlbedienung, Fehler in der Dokumentation, Viren

- Produktion zuverlässiger Software
  - ⋆ straffstes Projektmanagement mit gemeinsamer Team–Datenbank
  - ⋆ soweit möglich formale Spezifikation
    (Problem u.a. Beschreibung Zeitverhalten)

- ⋆ formale Beschreibungstechnik für Teile
- ⋆ umfangreiche Dokumentation
- ⋆ sauberer, robuster Entwurf, u.a. mit Fehlererkennung, Fehlerlokalisierung, Verhinderung der Fehlerausbreitung
- ⋆ Methodiken zur Programmanalyse
- ⋆ Testmethodik
- ⋆ Code-Review
- ⋆ Entwicklung ggf. nach ISO 9000 mit SW-Zertifizierung, TÜV-Abnahme
- ⋆ Einsatz einer Spezifikationssprache, z.B. RT-UML

11.7 Software-Redundanz

(1) Benutzung redundanter Hardware

- Rechner verschiedener Hersteller
- gleiche Algorithmen auf verschiedenen Rechnern compiliert
- parallele Ausführung der gleichen Algorithmen auf verschiedenen Rechnern
- Votieren der Ergebnisse
- Beispiel: Projekt FUTURE

(2) Voll diversitäre Software

- Realisierung in unterschiedlichen Firmen
- Realisierung auf unterschiedlichen Rechnern
- unterschiedliche, unabhängige Teams (mindestens 2)
- unterschiedliche Spezifikation
- unterschiedliche Algorithmen
- unterschiedliche Entwicklungen, Realisierungen
- paralleler Ablauf auf verschiedenen Rechnern
- Akzeptanzkriterium oder Votieren der Ergebnisse

- Alternativ: Vergleich und Abstimmung bei den Entwicklungsschritten und Einsatz nur eines Systems

(3) Partiell diversitäre Software

- Einschränkungen in obiger Liste
- z.B. nur verschiedene Algorithmen: Ergebnis eines Algorithmus wird an Hand Akzeptanzkriterium geprüft; bei Versagen erfolgt Wiederholung mit anderem Algorithmus
- Beispiel: Schreiben eines Datenblocks

```
ENSURE <Auftragswarteschlange konsistent>
      /* Akzeptanzbedingung */
BY
      /* 1. Algorithmus */
   Auftrag optimal in Warteschlange einordnen;
ELSE BY
      /* 2. Algorithmus */
   direkt in vorgegebenen Bereich schreiben;
ELSE
   Fehlermeldung Auftrag nicht ausgeführt;
```

- Erfahrungsbeispiel "PODS"
  (projekt on diverse software)
  - ⋆ P. Bishop et al. in IEEE Trans. on SW–Engineering, Vol SE-12, S. 929-940, 1986
  - ⋆ Aufgabe: kritisches Steuerprogramm beim Hochfahren eines Generators
  - ⋆ Ausführung durch drei Teams in England, Norwegen, Finnland
  - ⋆ beobachtete Phasen
    - ◦ 1 Kundenspezifikation
    - ◦ 3 unabhängige Herstellerspezifikationen
    - ◦ 3 Entwürfe
    - ◦ 3 Codierungen
    - ◦ 3 Abnahmen (2472 Tests)
    - ◦ Votieren der 3 Programme (665 288 Tests)
  - ⋆ Spezifikationsmethode:

    Team 1: informell
    Team 2: Spez. Sprache
    Team 3: Spez. Sprache

⋆ verwendete Sprache:

Team 1: FORTRAN
Team 2: Assembler
Team 3: FORTRAN

⋆ Algorithmus:

Team 1: Polynom–Algorithmus
Team 2: Tabellen
Team 3: Tabellen

⋆ Ergebnis des Votierens:

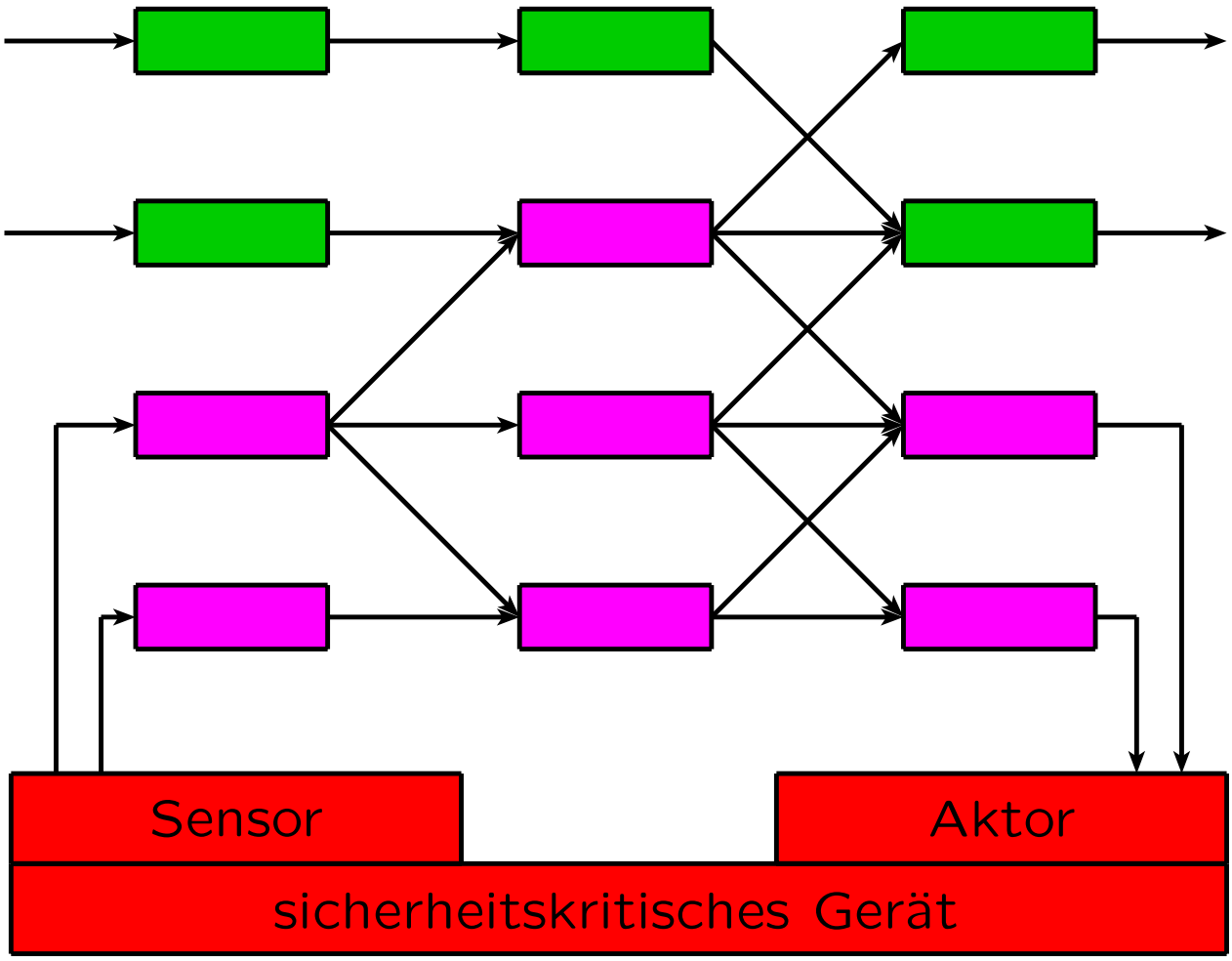
- 7 Fehler entdeckt
- davon 6 aus Kundenspezifikation (2 falsche Angaben, 4 mehrdeutige Angaben)
- davon 3 in gefährlichen Situationen
- davon 2 Fehler doppelt, d.h. sie hätten sich durchgesetzt
- kein Fehler bei der Codierung

- ⋆ Ergebnis der Untersuchung
  - ermöglicht evtl. Erkennung mehrdeutiger Kundenspezifikationen
  - erlaubt viele Tests mit Datensätzen aus Zufallszahlen im zulässigen Zahlbereich ohne das Ergebnis im voraus kennen zu müssen
  - wenig Unterschied von informeller und formaler Spezifikation (vielleicht war Projekt zu klein)
  - nach Abnahme keine Fehler in der Implementierung
  - Anzahl Fehler je Programmzeile unabhängig von der Programmiersprache
  - Laufzeit: Assembler 3 mal schneller als FORTRAN
  - kein Fehler im Compiler entdeckt
  - Restfehler aus ungenauer Kundenspezifikation
  - Kosten: etwa 2,25-fache von einem Projekt (366 Std. Kundenspezifikation, 1892 Std. für 3 Hersteller)

11.8 Maßnahmen zur Sicherheit

- Sicherheitsnachweise durch Dokumente und Prüfungen für TÜV–Abnahme erforderlich
- ausreichende Redundanz zur Fehlertoleranz oder Fehlererkennung
- möglichst einfache, stark modularisierte Systeme, da i.d.R.
  - ⋆ ein Versagen jeder Komponente das Versagen des Gesamtsystems bedeuten kann
  - ⋆ die Entwicklung unübersichtlicher
  - ⋆ Anzahl der Schnittstellen (Schwachstellen und Fehlerquellen) steigt
- Trennung von Sicherheit und Funktion als Entwurfsprinzip
  (sicherheitsorientierte Modularisierung)
  - ⋆ neben üblicher Modularisierung
  - ⋆ sicherheitsrelevante Funktionalität von restlicher Funktionalität getrennt und in wenige sichere Module gekapselt

⋆ Negativbeispiel (nach S. Montenegro)

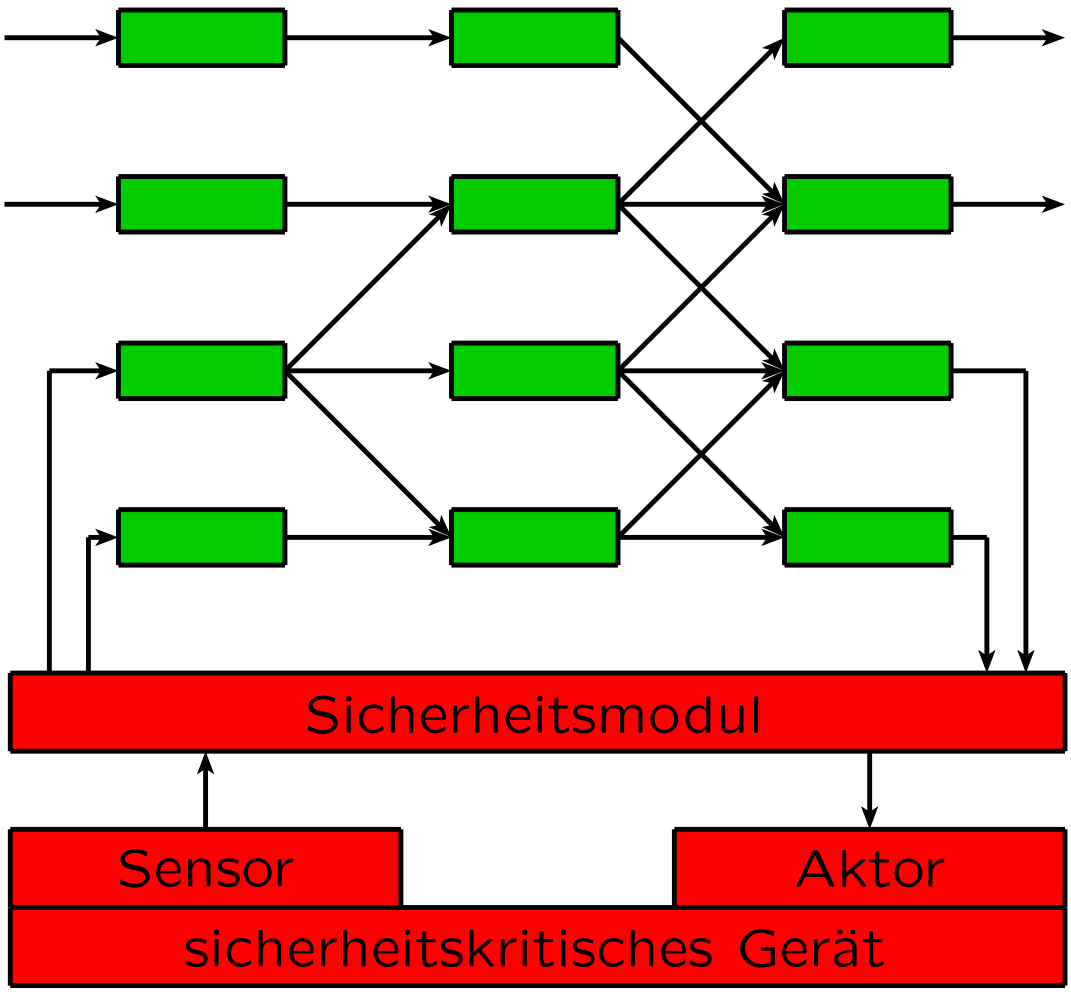


für Sicherheit kritischer Modul

unkritischer Modul

kritisch gewordener unkritischer Modul

- ⋆ empfohlene sicherheitsorientierte Modularisierung
  - ◦ Sicherheitsmodul hat vollständige Kontrolle über Anlage
  - ◦ Funktionsmodul kann Befehle nur über Sicherheitsmodul abgeben
  - ◦ Sicherheitsmodul prüft Sicherheit jedes Befehls bevor es ihn weitergibt



für Sicherheit kritischer Modul

unkritischer Modul

- "fail–safe"–Übergänge in ungefährliche Zustände vorsehen (z.B. sicheres schadenfreies Abschalten)
  - ⋆ Hardware–Einrichtungen für fail–safe–Übergänge, die nicht von der Software gesteuert werden (z.B. Grenzwertüberschreitung durch Schalter verhindern)
  - ⋆ benutzen von physikalischen Eigenschaften zum Übergang in einen sicheren Zustand. Beispiele: Relais fällt im Fehlerfall auf Ausschalten, Eisenbahnsignal fällt auf Halt,
  - ⋆ Senden aktiver Signale im korrekten Ablauf (z.B. periodisch "Weitermachen" an Maschine); beim Ausbleiben eines solchen Signals wird abgeschaltet

- Probleme bei Fehlermeldungen
  - ⋆ man verläßt sich blind auf das Frühwarnsystem und wird nachlässig ⇒ Unfall, wenn gefährliche Situation nicht gemeldet wird, z.B. wegen technischen Fehlers (was zu erwarten ist)
  - ⋆ Frühwarnsystem zu empfindlich, meldet Gefahren, die keine sind ⇒ Warnungen werden ignoriert ⇒ Unfall, wenn Fehlermeldung berechtigt war
- Sehr wichtig: Gestaltung der Mensch-Maschine-Schnittstelle
  - ⋆ benutzerfreundliche Bedienung, jedoch Prüfungen zur Verhinderung inkorrekter Eingaben
  - ⋆ am Leitstand geeignete ermüdungsarme Darstellungen für den menschlichen Überwacher
  - ⋆ vollständige, verständliche und begründete Alarmmeldungen
  - ⋆ Alarmmeldungen können nicht zu schnell beseitigt werden, damit sorgfältig reagiert wird

- ⋆ Bedienbarkeit der Anlage im Normal- und Notbetrieb sicherstellen
- ⋆ Negativbeispiel (Montenegro)
  - ◦ Notfall; aber kein großes Problem: Um Anlage in einen sicheren Zustand zu bringen, nur einige kritische Werte abzulesen und entsprechend gegenzusteuern
  - ◦ kritisch erst durch gutgemeinte Umstände:
    - ▷ Sirene jault in voller Lautstärke
    - ▷ rote Lichter blinken
    - ▷ Leute rennen herum und schreien verschiedene Anweisungen
    - ▷ Rechner ist in Notbetrieb und erlaubt den üblichen Zugriff auf die kritischen Werte nicht mehr
    - ▷ Wo normalerweise die benötigten Daten erscheinen, rauschen Fehlermeldungen und Warnungen über den Bildschirm. Sie sollten alle diese Meldungen lesen, um wissen zu können, was die jetzige Lage ist.

- ▷ Situation war im Training nicht erwähnt. Sie müssen zuerst in einem alten Manual nachsehen. Was da beschrieben ist, entspricht nicht dem, was vor Ihren Augen passiert.
- ▷ Mit Glück schaffen Sie es, die Werte, die Sie brauchen, zu bekommen.
- ▷ Sollwerte in verschiedenen Fenstern einzugeben, bei denen Texte und Überschriften blinken, damit Sie ihre Wichtigkeit erkennen.
- ▷ Um die Anlage manuell zu steuern, müssen Sie noch ein Paßwort eingeben, das vor kurzem geändert wurde, ohne daß Sie etwas davon wußten.
- ▷ Nun müssen Sie nur noch mit zitternder Hand die Maus pixelgenau positionieren, um die Sollwerte eingeben zu können.

◦ ... leider zeitlich nicht mehr geschafft $\Rightarrow$ menschliches Versagen

Meldung SZ v. 2.2.1999

Qualmender Aschenbecher löst 40minütigen Alarm aus

Düsseldorf (dpa) – Ein brennendes Taschentuch hat im Düsseldorfer Flughafen zu einem 40minütigen Fehlalarm geführt. So lange lief im Terminal C eine automatische Evakuierungsdurchsage, weil das Papiertaschentuch in einem Aschenbecher Feuer gefangen hatte. Mitarbeiter beruhigten die Passagiere im Terminal mit Megaphonen. Die Fluggäste hätten die Panne eher humorvoll aufgenommen, berichtete der Sprecher des Flughafens. Panik sei nicht ausgebrochen. „Wir mußten den Rechner der Brandmeldeanlage runterfahren und neu starten." Eine Entwarnung durch Lautsprecherdurchsagen sei nicht möglich gewesen, weil die Evakuierungsaufforderung keine anderen Ansagen zuließe. Bei einem Brand im Gebäude des Düsseldorfer Flughafens im April 1996 waren 17 Menschen getötet worden.