Hast du schon mal ein kleines Elektronik- oder Softwareprojekt gebastelt - vielleicht etwas, das nicht ausfallen darf?

Zum Beispiel eine Steuerung für's Modellflugzeug, ein Futterautomat für deine Tiere, ein Wasserbruchmelder für die alte Waschmaschine im Keller...

In diesem Talk wird dir erklärt, was Funktionale Sicherheit überhaupt ist, warum diese auch für Hobbyprojekte wichtig sein kann, was die sieben Plagen der Datenübertragung sind und was es eigentlich sonst noch so für Fehlerarten bei Hard- und Software gibt.

Außerdem wird dir gezeigt, was du tun kannst, um Fehler zu vermeiden oder wenigstens rechtzeitig aufzudecken; um Projekte zu entwickeln, ohne Sorgen vor Fehlfunktionen haben zu müssen.

Zu diesem Vortrag

- **Thema:** *Funktionale Sicherheit*
- **Zielgruppe:** Programmierer, Bastler, Tüftler, …
- **Zweck:**
  - Vermittlung, warum Funktionale Sicherheit auch in Hobby-Projekten sinnvoll sein kann
  - Verständnis der wichtigsten Begriffe
  - Leitfaden, um eigene Projekte sicher entwickeln zu können
- **Vortragender:** Cypax
  - Softwareentwickler von embedded systems im Bereich Funktionaler Sicherheit für Automatisierungs- und Medizintechnik

Inhalt

1. **Erster Teil**
   - Was ist *Funktionale* Sicherheit?
   - Und wozu brauchen wir das überhaupt?
2. **Zweiter Teil**
   - Wie mache ich mein Projekt sicher?
3. **Dritter Teil**
   - Wie mache ich meinen Code sicher?

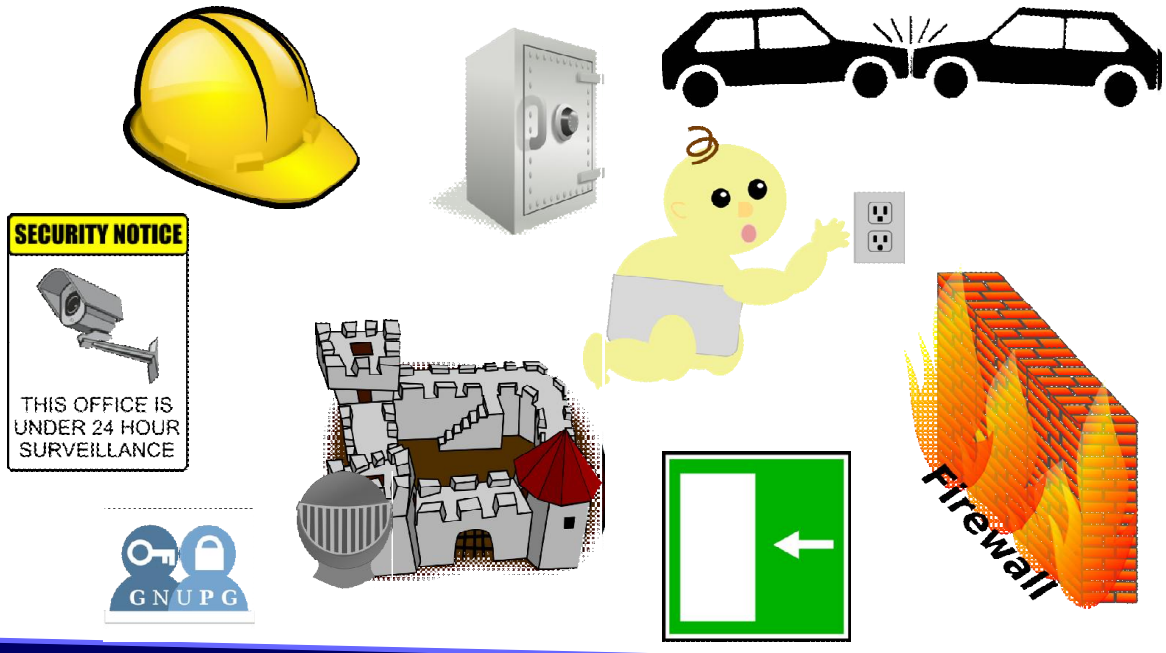
Fragen zum Vortrag / zur Präsentation: einfach reinreden
Fragen zu deinem speziellen Projekt: nach dem Vortrag

Erster Teil

- **Einstieg:**
  Was ist Funktionale Sicherheit?
- **Motivation:**
  Warum sollte uns das interessieren?
- **Begriffe:**
  Was sind Fehler, Schaden, Gefahr?

Einstieg und Definition

Frage an euch: Was ist eigentlich „Sicherheit“?



- Sicherheit ist ein **individuelles Bedürfnis** und kann völlig unterschiedliche Bedeutungen haben.
- Beispiele:
  - Sicherheit vor Verletzungen bei Unfällen im Verkehr oder am Arbeitsplatz
  - Sicherheit vor Diebstahl
  - Sicherheit vor Naturgewalten
  - Sicherheit vor Datenverlust
  - Sicherheit vor Spionage
- Allen Definitionen gemein ist das Streben nach **Schutz vor Risiken**.

Definition von Sicherheit

- Sicherheit ist:
  - Ein Zustand, welcher frei von unvertretbaren Risiken ist
- Absolute Sicherheit gibt es nicht
  - Ziel: angemessenes Maß an Sicherheit

6 / 83

- Das Gefühl *in Sicherheit* zu sein ist ebenfalls individuell und **vom Kontext abhängig**
  (Mit Bodyguard durch die KaJo? Nicht nötig.
  Auf dem Marktplatz in Mogadischu? Eher ja.)
- Um die Sicherheit zu erhöhen sind **Zeit und Kosten** nötig. Oft auch der **Verzicht** auf andere Dinge.
  100%ige Sicherheit ist deshalb nicht erstrebenswert.
- Folgerung: Sicherheit ist eine **Frage der Abwägung**.
- „*sich in falscher Sicherheit wägen*“ – irrtümliche Annahme sich in Sicherheit zu befinden durch Übersehen von potentiellen Gefahren oder Vertrauen in vorgeblichen Schutz (z.B. Überwachungskameras).

Was ist *funktionale* Sicherheit?

- Funktionale Sicherheit bedeutet:
  Die Funktion eines Systems wird sowohl im Normalbetrieb als auch im Fehlerfall garantiert oder die Funktion wird im Fehlerfall eingestellt und das System wechselt in einen ungefährlichen Zustand.

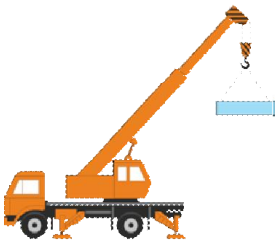
- Begriff „Safety“ steht für Funktionale Sicherheit

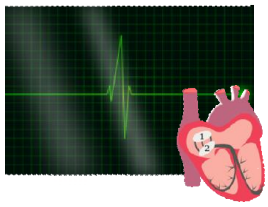
7 / 83

- Safety ist:
  - **Fehlervermeidung**
  - **Fehlererkennung** im laufenden Betrieb
  - **Fehlerbeherrschung**: sicherer Zustand wird eingenommen

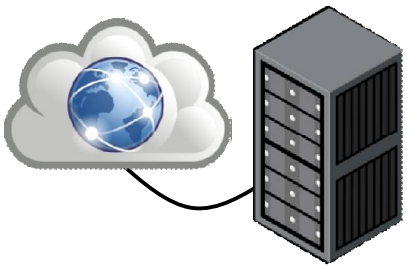
Safety vs. Security

- **Safety**: Schutz der Umgebung vor einem Gerät





- **Security**: Schutz des Gerätes vor der Umgebung



8 / 83

- *Safety* ist nicht *Security*!
  - Safety betrachtet keine Fälle durch gezielte Manipulation bzw. Angriffe von Außen.

Motivation

Warum brauchen wir funktionale Sicherheit?

- Fakt: Safety erfordert viel Zeit und Geld
- Warum also tun wir uns das an?

- In der Industrie:
  - Gesetze, Regulierungen, Prüfbehörden, Zulassungen, Versicherungen, …

- Aber in Hobby-Projekten:
  - What could possibly go wrong???

10 / 83

- In sicherheitskritischen Projekten besteht der Sourcecode bis zu 80% aus Sicherungsmaßnahmen

Ein Beispiel

Du programmierst die Steuereinheit deines Quadrokopters um?

Und willst nicht, dass er sich vielleicht plötzlich selbstständig macht?

11 / 83

Wissenswertes:
Modellautos und deren Betrieb sind in der Regel über die private Haftpflichtversicherung mitversichert.
Modellflugzeuge meist nicht. Auch nicht unter 5kg.
Grund:

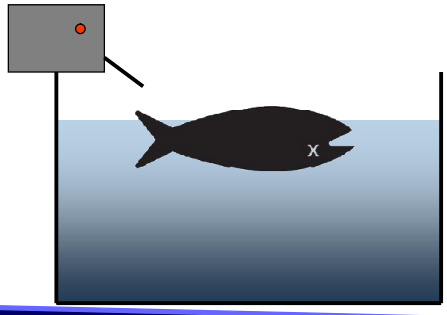
„*In Deutschland sind seit dem 1. Juli 2005 Flugmodelle laut Luftverkehrgesetz nicht mehr von der Versicherungspflicht für Luftfahrzeuge ausgenommen. Daher gelten seither für Flugmodelle die gleichen Ansprüche an die Haftpflichtversicherung wie für manntragende Flugzeuge, dies betrifft vor allem die Haftungssummen. Während die speziellen Modellflugversicherungen sich schnell an die neuen Anforderungen angepasst haben, haben viele allgemeine Versicherer die neuen Regelungen ignoriert oder die Versicherung von Modellflugzeugen ganz aus der allgemeinen Haftpflichtversicherung gestrichen.*“

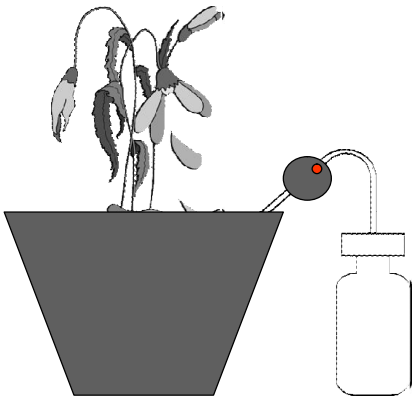
Quelle: http://de.wikipedia.org/wiki/Flugmodell#Rechtliches

Noch ein Beispiel

Oder entwickelst du vielleicht einen Fischfutter-automat oder eine Pflanzengießanlage?

Welche in Zeiten deiner Abwesenheit nicht versagen darf?





12 / 83

Kein Versicherungsschutz bei Hobbyelektronik

- Wohnungsbrand
- Brandursachenermittlung identifiziert DIY-Röhrenverstärker als Ursache

Versicherung:	**Ah, Sie haben gebastelt! Dann zahlen wir nicht.**
Du:	**Aber der Verstärker kann es nicht gewesen sein. Der hat einwandfrei funktioniert!**
Versicherung:	**Das müssen Sie nachweisen.**

13 / 83

- Bei der Beschäftigung mit seinem Hobby macht man sich in der Regel keine Gedanken um Versicherungsschutz. Gefahren durch Brände und elektronischen Schlag sind leider auch in der Hobbyelektronik ein oft ausgeblendetes Thema.

Gründe funktionale Sicherheit zu beachten

Fazit:

Auch in Hobby-Projekten gibt es Gründe, sich um die funktionale Sicherheit Gedanken zu machen.

Aber der wichtigste Grund ist:

Es ist unser eigener Anspruch, stabile und zuverlässige Software zu entwickeln!

Fehlerquellen

„Was kann eigentlich schief gehen?“

Arten von Fehlern

Zwei Arten von Fehlern:

- **Systematische Fehler**
  - Von Anfang an im System
  - Vermeidbar
- **Zufällige Fehler**
  - Treten unvorhersehbar auf
  - Nicht reproduzierbar

16 / 83

Systematische Fehler:

- Fehlerhafte Programmierung
  - z.B. Tippfehler, Denkfehler
- Fehlerhaftes Design
  - z.B. dead-lock
- Tool-Fehler
  - in Compiler, Linker, Editor, …
- Hardware-Fehler
  - z.B. Prozessor-Bug

Zufällige Fehler:

- Hardware-Defekt
  - Prozessor-Fehlverhalten
    - betrifft Befehlssatz, Register, embedded Peripherie, …
  - Speicherdefekte
    - z.B. Bit lässt sich nicht mehr setzen oder löschen
- Übertragungsfehler
  - Elektromagnetische Störungen manipulieren eine Information während der Übermittlung

Safety-Begriffe

Wichtige Begriffe

Wichtige Begriffe

- Erster Fehler
- Gefahr
- Schaden
- Fehlertoleranzzeit
- Sicherer Zustand

Wichtige Begriffe: Erster Fehler

- Erster Fehler = einzelne, erste auslösende Ursache.
- Ein funktional sicheres System muss die Situation beherrschen, in welcher ein beliebiger erster Fehler auftritt.

Wenn so ein Stützpfosten wegrutscht – wird der verbleibende Pfosten den Wagen halten?

19 / 83

- Definition für ersten Fehler: die erste auslösende Ursache in einer Fehlerkette, welche schließlich zu einem Schaden führt.
- Beispiele für erste Fehler:
  - Netzspannung fällt aus
  - Einzelnes Bauteil versagt
  - Einzelnes Bit wird bei Datenübertragung verfälscht
  - Bruch der elektrischen Isolierung
  - …

Wichtige Begriffe: Gefahr

- Eine Gefahr ist eine potentielle Schadensquelle
  - Jede Art von Energie
  - Stoffe, Gase und Flüssigkeiten
- Beispiele
  - Quadrokopter: kinetische Energie
  - Fischfutterautomat: elektrische Energie

Gefahr hier: Gewicht des Wagens (Lageenergie)

Wichtige Begriffe: Schaden

- Arten:
  - Verletzungen
  - Umweltschäden
  - Sachschäden
- Lässt sich anhand Schweregrad klassifizieren
  - Verletzungen: anhand Behandlung und Heilbarkeit
  - Umweltschäden: anhand Menge freigesetzter Stoffe
  - Sachschäden: anhand Schadensersatzkosten

Schaden hier: Knochenbrüche (bestenfalls), evtl. Genickbruch (worst case)

21 / 83

Schadensklassifizierung:

- Beispiele für Verletzungen am Menschen
  - reversibel; kein Arzt nötig
  - reversibel; Arzt nötig
  - irreversibel / dauerhaft
  - Tod
- Beispiele für Umweltschäden (hier internationale Skala für nukleare Ereignisse):
  - INES 3: Freisetzung in Höhe eines Bruchteils der natürlichen Strahlendosis
  - INES 4: (einige 10 bis einige 100 TBq), Freisetzung in Höhe der natürlichen Strahlendosis
  - INES 5: (einige 100 bis einige 1.000 TBq), einzelne Katastrophenschutzmaßnahmen nötig
  - INES 6: (einige 1.000 bis einige 10.000 TBq), komplettes Katastrophenschutzprogramm
  - INES 7: (einige 100.000 TBq), alles im Arsch
- Beispiel für Schadenklassifizierung bei Sachschäden:
  - < 5 EUR
  - 5 EUR bis 500 EUR
  - > 500 EUR bis 5000 EUR
  - > 5000 EUR

Wichtige Begriffe: Fehlertoleranzzeit

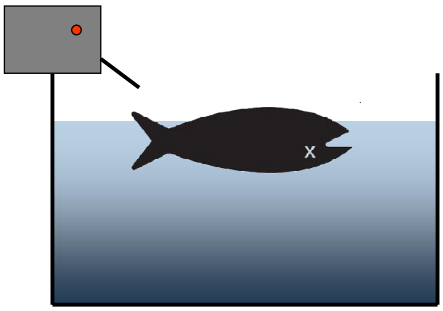
Definition:

- Maximal akzeptierbare Zeit seit dem Auftreten eines kritischen Fehlers bis zum Wirksamwerden der Gegenmaßnahme.
- Voraussetzung: innerhalb dieser Zeit kann der Fehler keinen (inakzeptablen) Schaden verursachen.

- Hängt vom jeweiligen Fehler ab und dessen resultierendem Effekt

Wichtige Begriffe: Sicherer Zustand

- Systemzustand welcher betreten wird, wenn ein kritischer, nicht-behebbarer Fehler erkannt wurde.
- Im sicheren Zustand wird der Benutzer alarmiert.



Sicherer Zustand = gar keine Fütterung mehr?

Im Prinzip ja, aber dann wird auch die Zweckbestimmung nicht mehr erfüllt.

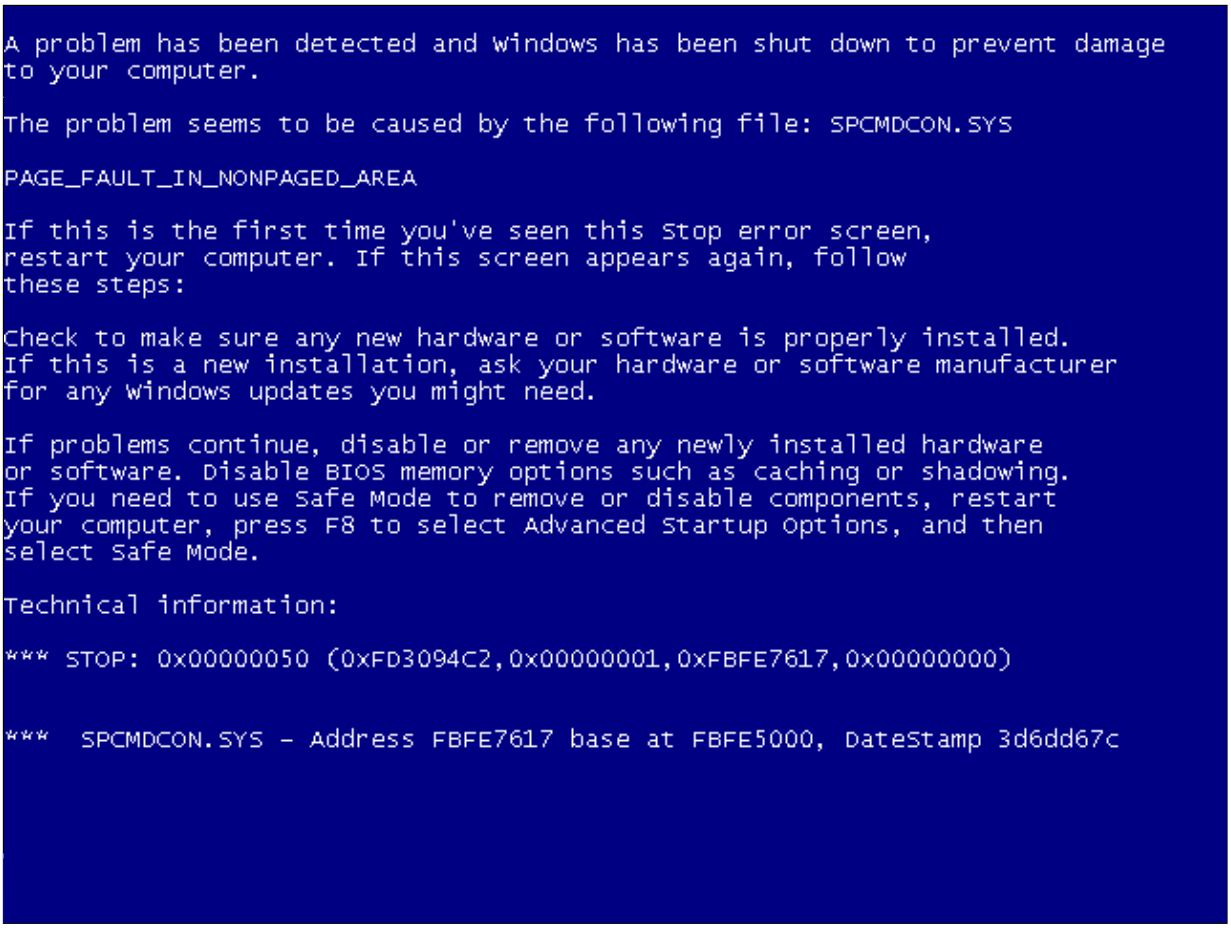
Besser: redundantes System (Zweikanaligkeit)

23 / 83

- Der sichere Zustand ist nicht immer darüber zu definieren, einfach alle Funktionen abzuschalten (Herzschrittmacher, Beatmungsgerät). Unabdingbar ist es jedoch, den Benutzer über den Fehlerzustand zu informieren.

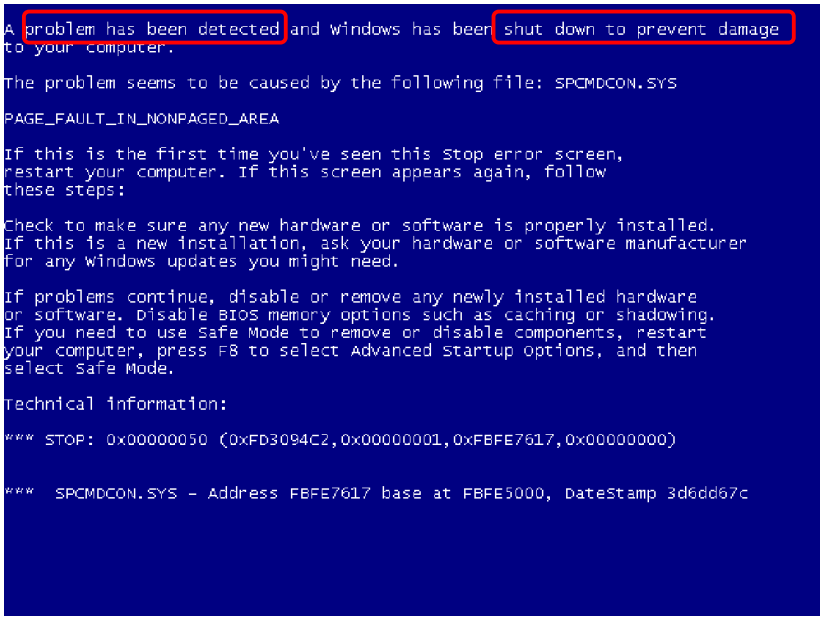
Bilderrätsel

- Gefahr?
  - Lageenergie des Felsen
- Gefährdungssituation?
  - Person unter dem Felsen
- Erster Fehler?
  - Stange bricht
- Schaden?
  - Knochenbrüche, Quetschungen, innere Blutungen, Tod


A problem has been detected and Windows has been shut down to prevent damage
to your computer.
The problem seems to be caused by the following file: SPCMDCON.SYS
PAGE_FAULT_IN_NONPAGED_AREA
If this is the first time you've seen this Stop error screen,
restart your computer. If this screen appears again, follow
these steps:
Check to make sure any new hardware or software is properly installed.
If this is a new installation, ask your hardware or software manufacturer
for any Windows updates you might need.
If problems continue, disable or remove any newly installed hardware
or software. Disable BIOS memory options such as caching or shadowing.
If you need to use Safe Mode to remove or disable components, restart
your computer, press F8 to select Advanced Startup Options, and then
select Safe Mode.
Technical information:
*** STOP: 0x00000050 (0xFD3094C2,0x00000001,0xFBFE7617,0x00000000)
*** SPCMDCON.SYS - Address FBFE7617 base at FBFE5000, DateStamp 3d6dd67c

Bilderrätsel

- Was war denn das jetzt?
  - **Sicherer Zustand** von Windows



Fazit zu Software-Fehlern

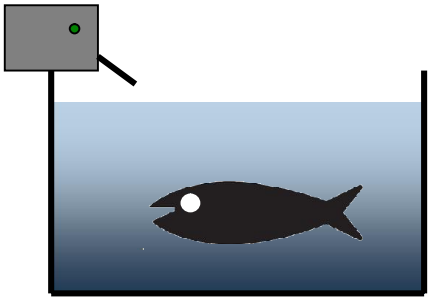
tl;dr - Was sollen wir uns merken?

- **Fakt:** Ein **erster Fehler kann jederzeit eintreten**.
- **Definition:** Wenn ein erster Fehler möglicherweise zu inakzeptablen Risiken führen kann, so ist er ein kritischer Fehler.
- **Unser Ziel:** Kein kritischer Fehler im System soll zu inakzeptablen Schäden führen können.
- **Anforderung:** Kritische Fehler müssen innerhalb der Fehlertoleranzzeit erkannt und behandelt werden.
- **Definition:** Wenn ein erster Fehler nicht direkt Schaden verursacht, wird er schlafender Fehler genannt.
- **Fakt:** Ein weiterer Fehler könnte eintreten, während ein schlafender Fehler noch nicht entdeckt wurde.
- **Unser Ziel:** Schlafender Fehler + weiterer Fehler sollen nicht zu inakzeptablen Schäden führen.
- **Anforderung:** Schlafende Fehler müssen innerhalb einer Mehrfachfehlertoleranzzeit erkannt werden.

27 / 83

Fehlerbeispiele am Fischfutterautomat:

- Display-Hintergrundbeleuchtung fällt aus
  - nicht kritisch
- Notfallakku leer (durch Memory-Effekt)
  - schlafender Fehler
- Feuchtigkeitsdichtung des Gehäuses schadhaft
  - kritisch (wg. Kurzschluss / Korrosion der Platinen)

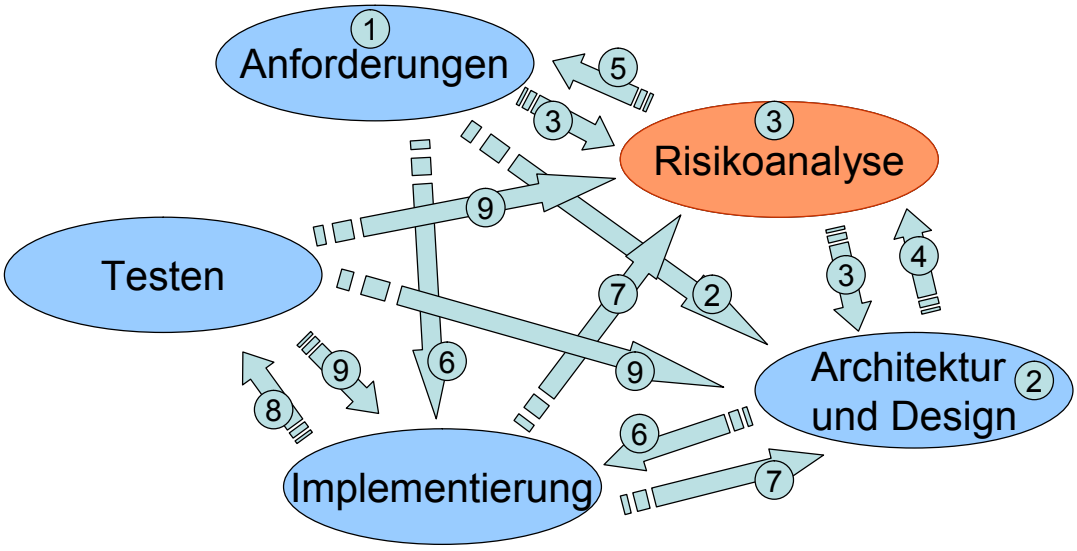


Zweiter Teil: Wie mache ich mein Projekt sicher?

- Wie gehe ich ein Projekt an?
- Risikoanalyse
  - Top-down und bottom-up
- Gegenmaßnahmen einbauen
- Mittel gegen systematische Fehler
- Mittel gegen zufällige Fehler
- Wie mache ich Datentransport und Datensicherung sicher?
- Sicherheitsarchitekturen
  - Zweikanalig, Einkanalig
- Sichere SW auf unsicherer Plattform, geht das?

Wie mache ich mein Projekt sicher?

Wege zum sicheren Projekt



29 / 83

1. Anforderungen formulieren (Was soll das System / die Software machen)
2. Architektur und Design entwerfen
3. Aus den Anforderungen ergeben sich Punkte für die Risikoanalyse, welche bei Architektur und Design berücksichtigt werden müssen
4. Architektur und Design können ebenfalls neue Risiken verursachen
5. Risikomindernde Maßnahmen in die Anforderungen aufnehmen
6. Anhand der Anforderungen und des Designs implementieren
7. Die Implementierung kann Nachbesserungen an Architektur / Design erfordern und ebenfalls neue Risiken bergen, welche in der Risikoanalyse berücksichtigt werden
8. Ergebnis der Implementierung ist das Testobjekt
9. Tests können Nachbesserungen an Implementierung oder Architektur / Design erfordern sowie neue, bislang unbedachte Risiken aufdecken

► **ständige Risikoanalyse** während allen Projektphasen

Risiken

Risikoanalyse

Definition ***Risiko***:

Kombination aus der Wahrscheinlichkeit, mit der ein Schaden eintritt, und dem Ausmaß dieses Schadens.

Risiken

Risikoanalyse

- Wie finden wir heraus, welche Risiken im Projekt bestehen?
- Zwei Ansätze:
  - Top-Down-Analyse
  - Bottom-Up-Analyse

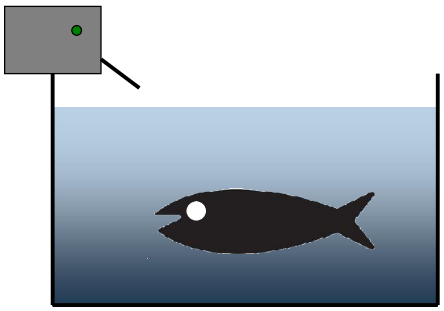
31 / 83

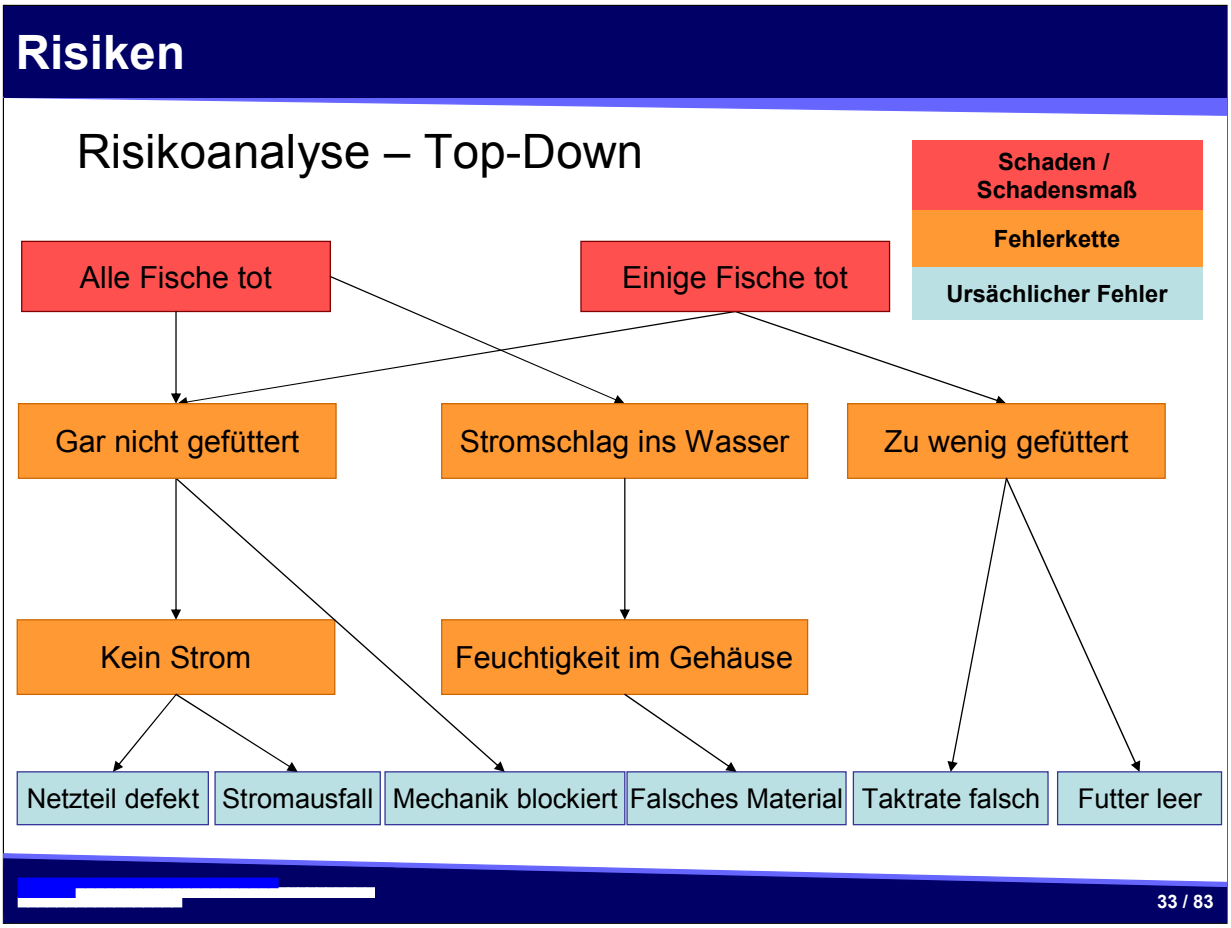
- Bottom-Up:
  - *„Wenn dieses Bauteil ausfällt, dann … und dann … und schließlich … “*
  - Voraussetzung: Systemarchitektur bereits vorhanden
- Top-Down:
  - *„Alle Fische könnten sterben, wenn … weil … weil …“*
  - Systemarchitektur muss noch nicht vorhanden sein.

Risiken - Beispielprojekt

Beispielprojekt – Fischfutterautomat

- Automat soll während Abwesenheit des Benutzers automatisch Fische im Aquarium zeitgesteuert portionsweise füttern.
- Automat hat
  - Behälter mit Futtervorrat
  - Fütterungsvorrichtung
  - Zeitschaltuhr
  - Netzteil




Risiken
Risikoanalyse – Top-Down
Schaden / Schadensmaß
Fehlerkette
Ursächlicher Fehler
Alle Fische tot
Einige Fische tot
Gar nicht gefüttert
Stromschlag ins Wasser
Zu wenig gefüttert
Kein Strom
Feuchtigkeit im Gehäuse
Netzteil defekt
Stromausfall
Mechanik blockiert
Falsches Material
Taktrate falsch
Futter leer
33 / 83

Risiken

Risikoanalyse – Bottom-Up

- Basiert auf Überlegungen anhand von Zwischenergebnissen der Projektentwicklung wie Schaltplan, Code, …
- „Was passiert wenn …“
  - … dieser Transistor ausfällt?
  - … es einen Kurzschluss zwischen diesen Pins gibt?
  - … dieser Interrupt ständig getriggert wird?
  - …
- „Was wären Folgefehler?“

Risiken

Risikomatrix

	Wahrscheinlichkeit (1..3)	Schaden (1..3)	Risiko (1..9)		
Netzteil fällt aus	2	3	inakzeptabel	1	gering
Strom fällt aus	1	2	akzeptabel	2	mittel
Mechanik blockiert	3	2	inakzeptabel	3	hoch
Bedienfehler	2	2	inakzeptabel		
Kurzschluss	1	2	akzeptabel		
Transistor T2 fällt aus	1	2	akzeptabel		

- Risiko = Kombination aus Wahrscheinlichkeit und Schadensmaß
- Welches Risiko wir noch vertreten wollen, definieren wir selbst

35 / 83

- Das Risiko kann das Produkt aus Schaden * Wahrscheinlichkeit sein, es sind aber häufig Abwägungen im Einzelfall. Schließlich sollen vernachlässigbare Schäden, welche aber sehr oft auftreten nicht gleichermaßen bewertet werden, wie ein sehr seltener schwerer Schaden.

Gegenmaßnahmen

Gegenmaßnahmen

Gegenmaßnahmen

Gegenmaßnahmen einbauen

- Nach Bestimmung der Risiken werden geeignete Gegenmaßnahmen eingebaut
- Die ursprünglichen Risiken werden neu bewertet
- Die Gegenmaßnahmen können selbst Risiken verursachen → erneute Risikoanalyse

Gegenmaßnahmen

Gegenmaßnahmen einbauen

Beispiel:

- Gegenmaßnahme gegen Netzteilausfall:
  Akku zur Notversorgung
- Risiko neu bewerten

	Wahrscheinlichkeit (1..3)	Schaden (1..3)	Risiko (1..9)
Netzteil fällt aus	2	1	akzeptabel

- Die Gegenmaßnahmen können selbst Risiken verursachen → erneute Risikoanalyse und evtl. erneute Gegenmaßnahmen, z.B.:
  - Akku defekt (schlafender Fehler)
    - Akkuspannung wird ständig überwacht
  - Akku überhitzt beim Aufladen
    - Temperaturüberwachung

Mittel gegen systematische Fehler

Erinnerung:

- Systematische Fehler sind von Anfang an im System (eingebaute Fehler)
- Systematische Fehler sind vermeidbar

Mittel gegen systematische Fehler

Mittel gegen systematische Fehler = **Vermeidung**:

- Datenblätter von Prozessor und Datenspeicher nach bekannten Fehlern (known issues) untersuchen
- Suche nach errata sheets zu Prozessor und Speicher
- Internetrecherche zu bekannten Problemen mit Compiler, Linker und IDE

Mittel gegen systematische Fehler

- Problemquelle Mensch: Code ist wirr, unwartbar, undokumentiert, enthält ungenutzten Code, enthält Typos, Verwechslungen und logische Fehler
- Abhilfe:
  - Software-Architektur und –Design dokumentieren
  - Codierstandard einhalten
  - Typsichere Entwicklung
  - Defensives Programmieren
  - 4-Augen-Prinzip

41 / 83

- Eine Dokumentation ist besonders bei Projekten mit mehreren Beteiligten hilfreich.
- Aber Achtung: eine Dokumentation kann nur dann helfen Fehler zu vermeiden, wenn sie auch gelesen wird um Abweichungen von Soll (Doku) zu Ist (Code) zu finden.

Mittel gegen zufällige Fehler

Erinnerung:

- Zufällige Fehler treten unerwartet auf
- Nicht vermeidbar
- Immer infolge eines Hardware-Defekts
- Müssen von der Software erst **erkannt** werden, damit Gegenmaßnahmen gestartet werden

Mittel gegen zufällige Fehler

- **Erkennung** zufälliger Fehler:
  - Kontinuierlich und periodisch Tests durchführen von Speicher, Registern, Peripherie, …
  - Hardwareausfälle durch Watchdogs abfangen
  - Nutzung mehrerer Kanäle
  - Sicherung von Daten (z.B. durch Prüfsummen)
  - Logische Programmablaufüberwachung

43 / 83

- *Watchdog* = Aufpasser, welcher nach einer bestimmten Zeit ohne bestimmtes Signal einen Reset oder eine andere sichernde Maßnahme durchführt.

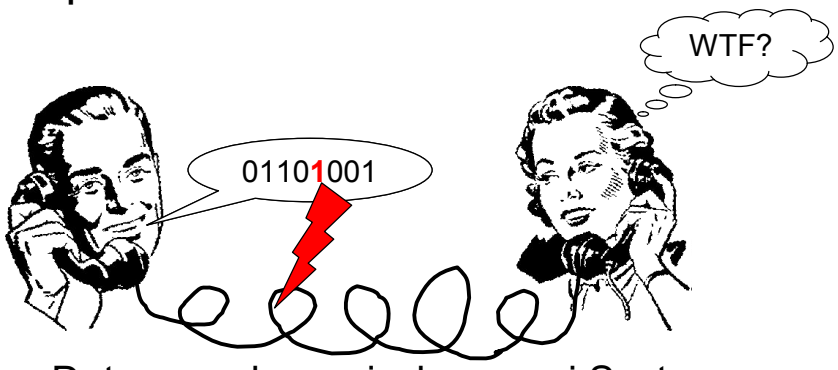
Sichere Datenübertragung

Datenübertragungen

Fehler und Gegenmaßnahmen

Sichere Datenübertragung

Datentransport sichern



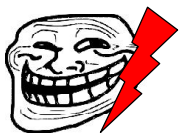
- **Szenario:** Daten werden zwischen zwei Systemen ausgetauscht.
- **Problem:** Daten könnten während dem Transport verändert werden
- **Ziel:** Empfänger soll in der Lage sein, Datenfehler zu erkennen

Sichere Datenübertragung

Rollenspiel: Szenario

- Teilnehmer:
  - Sender
  - Empfänger
  - Übertragungskanal
  - Erster Fehler



Sichere Datenübertragung

Rollenspiel: Ablauf

Mehrere Runden nach diesem Prinzip:

- *Sender* schreibt Nachricht für *Empfänger*, liest sie dem Publikum laut vor und gibt sie an *Übertragungskanal*.
- *Übertragungskanal* leitet Nachricht an *Empfänger*.
- *Empfänger* liest erhaltene Nachricht dem Publikum laut vor.

Regeln:

- Der *Erste Fehler* darf in jeder Runde in einem beliebigen Schritt eine Störung machen.
- *Sender* und *Empfänger* dürfen Sicherungsmaßnahmen untereinander vereinbaren.
- *Empfänger* kennt Inhalt der Nachricht vorher nicht.

Ziel:

- Störungen sollen von *Sender* oder *Empfänger* erkannt werden.

Sichere Datenübertragung

Rollenspiel – Lessons learned

- Gibt deinem Empfänger die Möglichkeit deine Nachricht zu prüfen.
- Prüfe eingehende Nachrichten auf Vollständigkeit, Korrektheit, zeitliche und logische Abfolge.
- Stelle sicher, dass du mit dem richtigen Teilnehmer kommunizierst.

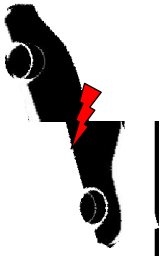
Sichere Datenübertragung

Übertragungsfehler: Verlust

- Eine Nachricht wird nicht empfangen

START

ZIEL



- Abhilfe:
  - Laufende Nummer
    (Fehler wird bei nächster zugestellten Nachricht erkannt)
  - Timeout
    (Empfänger erwartet Nachricht)

Sichere Datenübertragung

Übertragungsfehler: Wiederholung

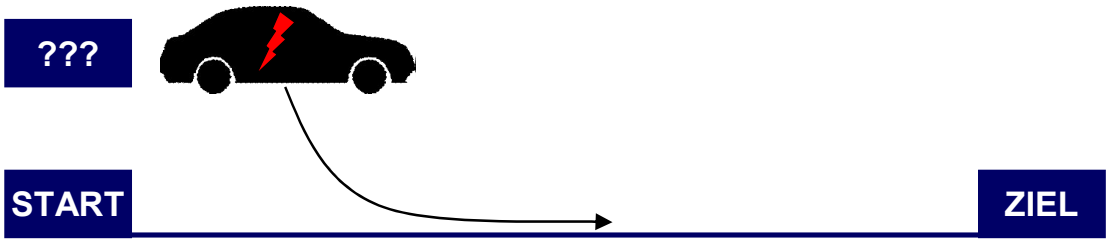
- Alte Nachricht wird unbeabsichtigt nochmals gesendet

- Abhilfe:
  - Laufende Nummer (Erwartung: Nummer immer um 1 höher)
  - Ggf. Zeitstempel (Erwartung: jünger als letzte Nachricht)

Sichere Datenübertragung

Übertragungsfehler: Fehladressierung

- Eine Nachricht einer falschen Quelle wird empfangen

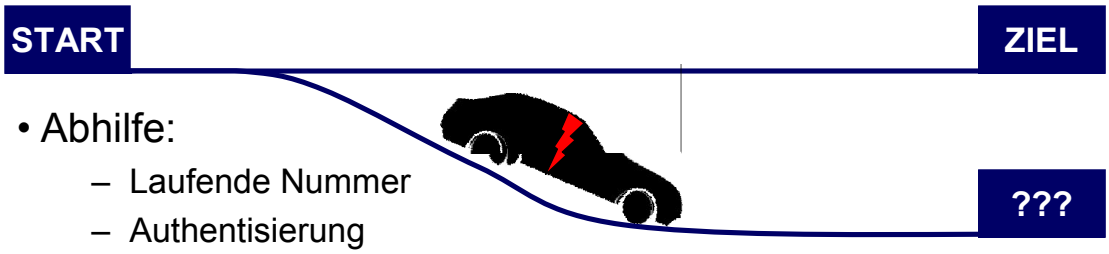


- Abhilfe:
  - Laufende Nummer (falscher Sender kennt aktuelle Nummer nicht)
  - Authentisierung (Verbindungsschlüssel)
  - Rückmeldung (Empfänger bestätigt, was er empfangen hat)

Sichere Datenübertragung

Übertragungsfehler: Fehladressierung

- Eine Nachricht wird an den falschen Empfänger geschickt



- Abhilfe:
  - Laufende Nummer
  - Authentisierung

Sichere Datenübertragung

Übertragungsfehler: Verzögerung

- Eine Nachricht ist über ihr zulässiges Empfangsfenster hinaus verzögert

- Abhilfe:
  - Zeitstempel (Empfänger lehnt zu alte Nachrichten ab)
  - Timeout (Nachricht wird innerhalb eines Zeitfensters erwartet)

Sichere Datenübertragung

Übertragungsfehler: Verfälschung

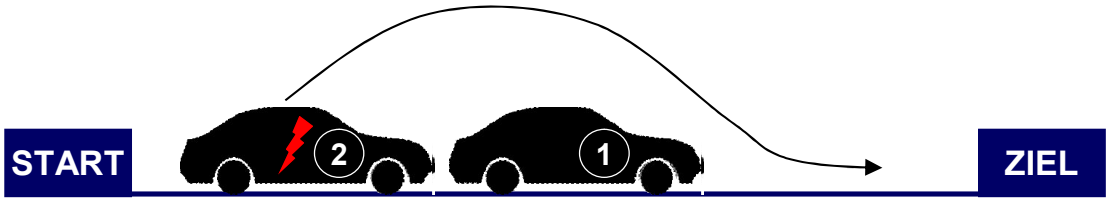
- Nutzdateninhalt der Nachricht wird verfälscht

- Abhilfe:
  - Rückmeldung (Empfänger bestätigt, was er empfangen hat)
  - Checksumme (Prüfsumme über die Daten)

Sichere Datenübertragung

Übertragungsfehler: Falsche Abfolge

- Nachrichten werden in falscher Reihenfolge gesendet oder empfangen



- Abhilfe:
  - Laufende Nummer
  - Zeitstempel
  - Rückmeldung

Sichere Datenübertragung

Übersicht – die sieben Plagen der Datenübertragung

	Laufende Nummer	Zeitstempel	Timeout (Zeiterwartung)	Authentisierung	Rückmeldung (Echo)	Checksumme
Verlust	✓		✓			
Wiederholung	✓	✓				
Einfügung	✓			✓	✓	
Fehladressierung	✓			✓		
Verzögerung		✓	✓			
Verfälschung					✓	✓
Falsche Abfolge	✓	✓			✓	

Sicherheitsarchitekturen

Sichere Systemarchitekturen

Einkanalig

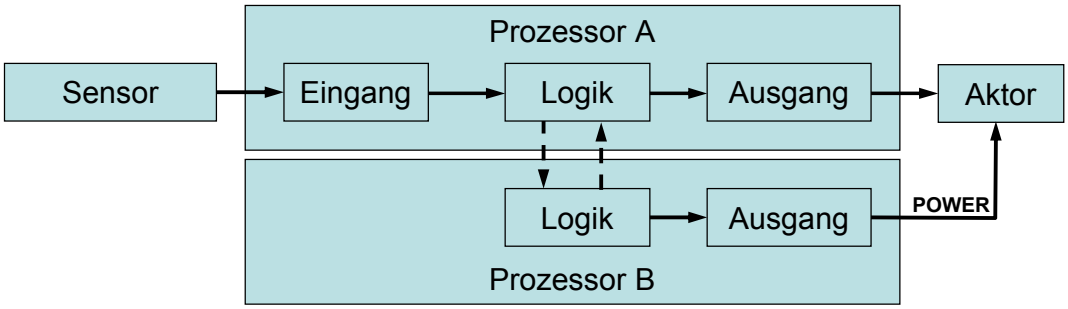
Zweikanalig

Wie mit unsicheren Plattformen umgehen?

Sicherheitsarchitekturen

Einkanalig mit Diagnose

- Aufbau:



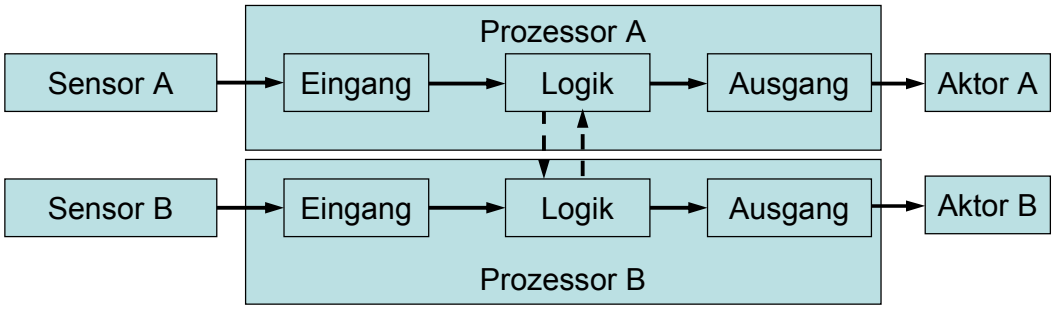
58 / 83

- Steuerungspfad muss Kontrollpfad um Freigabe bitten.
- Kontrollpfad überwacht Steuerungspfad und kann diesen in einen sicheren Zustand bringen (z.B. Aktor abschalten, Steuerungssystem zurücksetzen, …).

Sicherheitsarchitekturen

Zweikanalige Architektur

- Aufbau:



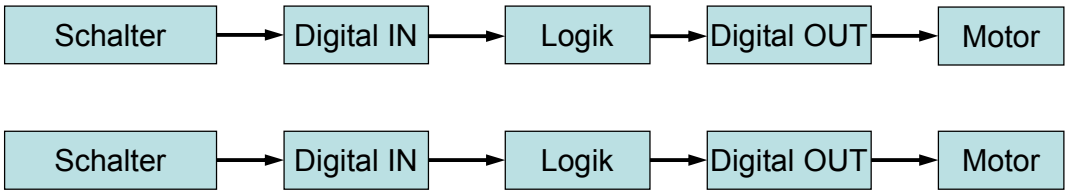
59 / 83

- Redundanz: Jeder Pfad kann bei Ausfall eines anderen Pfads die Aufgabe allein übernehmen.
- Die Pfade wechseln sich ab und kontrollieren sich gegenseitig.

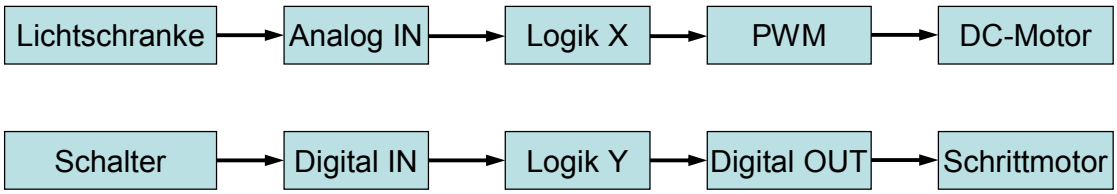
Sicherheitsarchitekturen

Zweikanalige Architekturen

- Redundanter Aufbau:



- Diversitärer Aufbau:



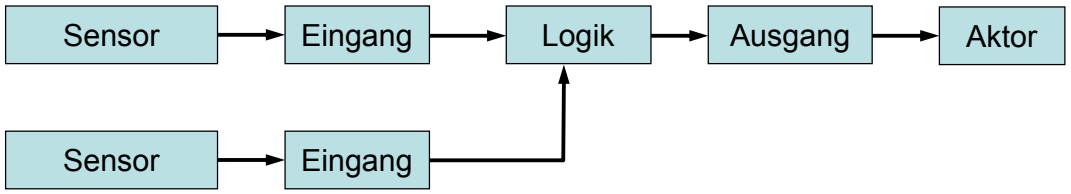
60 / 83

- Diversität kann erreicht werden durch:
  - Unterschiedliche Technologien / Verfahren
  - Unterschiedliche Bauteilhersteller
  - Unterschiedliche Datentypen (Float vs. Integer)
  - Unterschiedliche Vorzeichen (+42 vs. -42)
  - Unterschiedlicher Code von unterschiedlichen Entwicklern / Programmierern

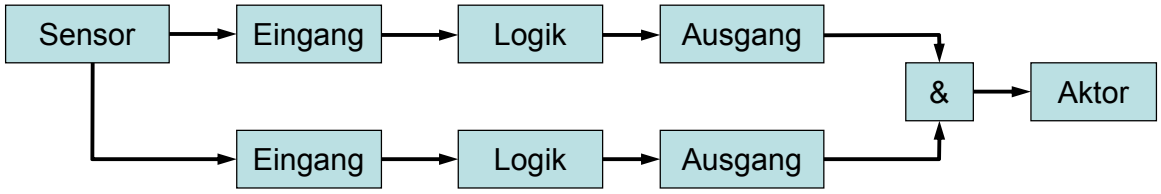
Sicherheitsarchitekturen

Mischformen aus Einkanalig und Zweikanalig

- Beispiel:

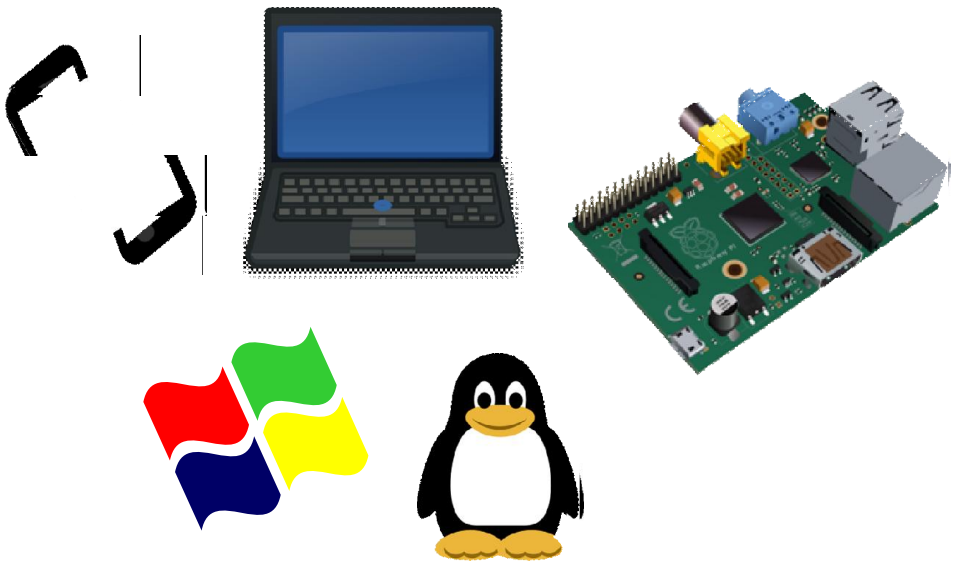


- Beispiel:



61 / 83

Sichere SW auf unsicheren Plattformen

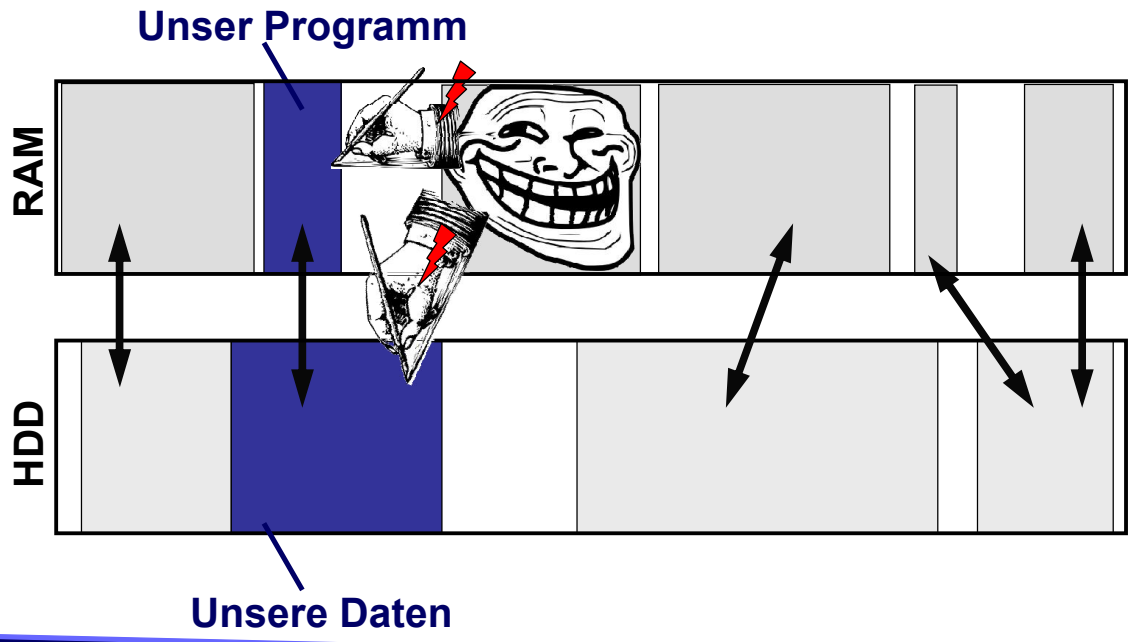


62 / 83

- Unsichere Plattform = Software (Betriebsystem / Laufzeitumgebung) oder Hardware welche nicht für fehlersicheren Betrieb ausgelegt ist.
- Gründe unsichere Plattformen zu verwenden:
  - kurze Entwicklungszeiten; geringerer Aufwand
  - Wiederverwendbarkeit und Portabilität der eigenen Software
- Probleme mit unsicheren Plattformen:
  - kein Verlass auf Echtzeit möglich (z.B. Programm läuft nur langsam während Betriebsystemupdate)
  - Plattform kann abstürzen / einfrieren
  - I/O-Ressourcen und Speicher werden von anderen Programmen blockiert
  - usw., …

Unsichere Plattformen

Die Situation:



- Auf einer unsicheren Plattform gibt es zahlreiche andere Prozesse, welche absichtlich oder unbeabsichtigt Fehlfunktionen verursachen können.
- Weder im Arbeitsspeicher noch auf Datenträgern ist die Datenkontrolle gewährt.
- Es ist damit zu rechnen, dass unsere SW auf einer unsicheren Plattform jederzeit beliebig beeinträchtigt wird.

Unsichere Plattformen: Lösungswege

- Absicherung durch **zweite, sichere Plattform**
  - Kontrollpfad

  oder
  - Redundanter zweiter Pfad
- Sicherung der eigenen Daten auf der unsicheren Plattform
- Aber Vorsicht: unsichere Plattform darf sichere Plattform nicht „mit ins Verderben ziehen“

64 / 83

Abgrenzungsmaßnahmen zwischen unsicherer und sicherer Plattform:

- getrennte Spannungsversorgungen
- getrennte Taktquellen
- galvanische Trennung gemeinsamer Schnittstellen
- DOS (denial of service) der sicheren Plattform ausschließen

Dritter Teil: Sicherer Code

- Grundsätze
- Speichertests
- Registertests
- Stacktest
- Sichere Funktionsaufrufe
  - Parameterübergabe und Rückgabewerte
    - Hamming-Distanz
    - Strukturparameter
  - Logische Programmablaufkontrolle

Sicherer Code: Grundsätze

- Potentielle Probleme vermeiden:
  - Klarer Programmablauf
  - Keine Objekte zur Laufzeit anlegen / freigeben
  - Keine globalen Variablen
  - Interrupts vermeiden
  - Pointer vermeiden
  - Keinen fremden Code verwenden
- Einfache Strukturen und Schnittstellen
- Defensives Programmieren
  - Prüfen von eingehenden Daten vor Verarbeitung
- Kodierrichtlinien einhalten
  - Reduzierter Sprachumfang (nicht alle C/C++ Konstrukte zulassen)
  - Style Guide

66 / 83

Das Einhalten von Kodierrichtlinien verhindert zwar nicht direkt Fehler, ist aber Voraussetzung um Abweichungen des Codes vom Plan (Anforderungen und Design) erkennen zu können.
Negativbeispiel:

```
i % 2 == 0 ? i % 3 == 0 ? "fizzbuzz" : "fizz" : i % 3 == 0 ? "buzz" : i.ToString();
```

Sicherer Code

Selbsttests

Speichertests, Registertests, Stacktest

Speichertests von konstantem Speicher

- Trivialbeispiel: Paritätsbit

  - Diagnosedeckungsgrad 50%

- Zyklische Redundanzprüfung (CRC)
  - Diagnosedeckungsgrad > 99%

68 / 83

CRC:

- Checksumme („Fingerabdruck“) eines Datensatzes wird berechnet
- Berechnung basiert auf Polynomdivision
- Unterschiedliche Verfahren
  - Unterschiedliche Längen der Ergebnisse (16 Bit, 32 Bit), je nach Menge der zu prüfenden Daten
  - Bekannte Polynome: CRC-CCITT, CRC-32 (TCP/IP), CRC-8 (ISDN), u.v.a.

Speichertests von veränderlichem Speicher

- „Galpat-Test“
- Erkennung von statischen Bitfehlern und dynamischen Kopplungen in RAM
- Ablauf:

0	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

1. Speicher löschen mit 0.
2. Erste Zelle invertieren.
3. Prüfen ob Zelle invertiert ist und alle anderen Zellen noch 0 sind.
4. Abbruch bei Fehler oder Ende des Speichers.
5. 1 nach rechts schieben und Test bei Schritt 3) fortsetzen.
6. Anschließend Schritte 1) bis 5) bit-invers wiederholen.

Galpat-Test: Probleme in der Praxis

- Test dauert sehr lange
- Wenn RAM getestet wird, sabotiert sich der Test selbst
- Es ist eher die Ausnahme, dass eine defekte Speicherzelle entdeckt wird, bevor sie bereits verwendet wurde

70 / 83

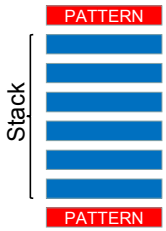
- Gegen lange Testdauer
  - Test nur einmal bei Starten der Software komplett durchlaufen lassen
  - Im normalen Betrieb nur segmentweise Durchführung
- Gegen Selbstzerstörung
  - Test wird in Assembler implementiert (kein Stack nötig und volle Kontrolle über Register)
  - Sichern der Daten in reserviertem Speicherbereich + zusätzliche Maßnahmen um diesen Bereich zu schützen
- Weitere Maßnahmen
  - Weitere Maßnahmen sind für sicherheitskritische Daten erforderlich (z.B. Checksumme)

Registertests

- Register sind im Prinzip veränderbarer Speicher
  - Konsequenz: gleiche Fehlerannahmen wie bei RAM
- Register, welche zur Laufzeit konstant bleiben
  - werden durch CRC geschützt
- Register, welche zur Laufzeit geschrieben werden
  - Problem mit SFR (special function register): Speichertest wie bei RAM nicht möglich
    - Test nur indirekt möglich (z.B. durch Beobachtung einer Reaktion oder durch Vergleich über anderen Weg)

Stacktest

- Stack liegt im RAM → Bitfehler und Kopplungen
  - werden bereits durch Speichertest abgedeckt
- Stack-Overflow und -Underflow
  - Variante 1:



    - Vor und hinter dem Stack wird ein Bereich reserviert, welcher ein festes Prüfmuster beinhaltet.
    - Es wird regelmäßig geprüft, ob diese Bereiche unverändert sind.
  - Variante 2:
    - Bei Eintritt jeder Funktion wird geprüft, ob Stackpointer und Rücksprungadresse noch im gültigen Bereich sind.
    - Zuverlässiger als Variante 1, aber braucht mehr Ausführungszeit und Programmspeicher.

Nachteile der Variante 1:

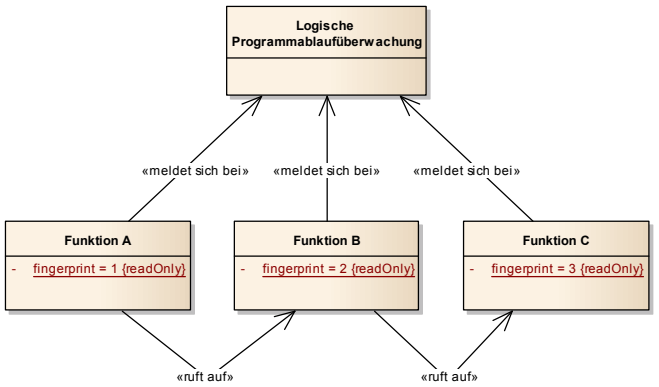
- Falls die Bereiche verändert sind, so sind vermutlich bereits Folgefehler aufgetreten; evtl. funktioniert der Test gar nicht mehr.
- Funktioniert nicht bei endloser Rekursion.

Sicherer Code

Funktionsaufrufe

Funktionsaufrufe: Logische Ablaufüberwachung

- Prinzip:



- Verschärft:
  - Bei jeder CRC-Aktualisierung wird mit einem Referenzwert verglichen (und bei einer Abweichung sofort Gegenmaßnahmen unternommen).
  - Zeitliche Überwachung
    - z.B.: Sequenz muss innerhalb von … durchlaufen sein.
    - z.B.: Nächste Funktion muss innerhalb von ... aufgerufen werden.

74 / 83

- Ablauf:
  - Funktion A meldet sich mit ihrem Fingerprint bei der Ablaufüberwachung, welche den Fingerprint in eine CRC-Berechnung eingehen lässt. Ebenso Funktion B und Funktion C.
  - Am Ende eines Durchlaufs wird das CRC-Ergebnis mit einem Referenzwert verglichen und zurückgesetzt.
  - So ist die korrekte Sequenz A →B →C sichergestellt.
  - Abweichungen (z.B. A →C) werden erkannt und die Programmablaufüberwachung bringt das System in einen sicheren Zustand.

Datenübergabe bei Funktionsaufrufen

Szenario

Beteiligte:

- Funktion A
- Funktion B
- Stack
- Erster Fehler (the bad guy)

• Ablauf:

1. Funktion A ruft Funktion B auf und übergibt einen Wert.
2. Funktion B gibt zurück, ob der Wert in Ordnung ist.

Beispiel zu Funktionsaufrufen

Der Code dazu:

```
void function_A()
{
 unsigned char wert = 42;
 if (true == function_B(wert)) { machwas(); }
}

bool function_B (unsigned char param)
{
 if (42 == param) { return true; }
 else { return false; }
}
```

Beispiel zu Funktionsaufrufen

Was wirklich passiert

Detaillierter Ablauf:

1. Rücksprungadresse wird auf Stack gelegt
2. Wert 42 wird auf Stack gelegt (Parameter *param*)
3. function_B wird aufgerufen (jump-Befehl)
4. Der Parameter (*param*) wird vom Stack gelesen
5. Der Parameter (*param*) wird verglichen mit Wert 42
6. Rückgabewert von function_B wird auf Stack gelegt
7. Rücksprung wird ausgeführt
8. Rückgabewert wird von Stack gelesen
9. Rückgabewert wird verglichen mit Wert true (1)

77 / 83

- Beim Programmieren neigt man schnell dazu die Vielzahl möglicher Fehler zu unterschätzen. Hier hilft es, sich zu vergegenwärtigen, was bei der Codeausführung hinter den Kulissen eigentlich technisch passiert.

Funktionsaufrufe: Lessons learned

1. Prüfe die Integrität eingehender Daten.
2. Gib anderen Funktionen die Möglichkeit deine ausgehenden Daten zu prüfen.

Außerdem:

- Nutze keine fremden Mittel um etwas als sicher zu bewerten (also z.B. keine Bibliotheken).

Sichere Funktionsaufrufe: Lösungswege

Möglichkeit sich gegen Fehler zu wehren:

- Hamming-Distanz
  - Gibt an, in wie vielen Bits sich zwei Werte unterscheiden.
  - Beispiel:

```
typedef enum t_SafeBool {
  E_SAFE_FALSE = 0x55; //01010101
  E_SAFE_TRUE = 0xAA; // 10101010
};
```

Die Werte von true und false unterscheiden sich in mehr als einem Bit (hier Hamming-Distanz = 8).

Sichere Funktionsaufrufe: Lösungswege

Möglichkeit sich gegen Fehler zu wehren:

- Checksummen
  - Prüfsumme wird über die Daten berechnet
- Zweikanal-Konzept
  - Parameter wird doppelt übergeben (Kanal 1 und Kanal 2)
  - Beide Kanäle werden miteinander verglichen
  - Steigerung: Kanal 2 ist bit-invers zu Kanal 1 (z.B. 0x2A und 0xD5)

Was wir heute gelernt haben

1. Uns ist klar geworden, dass auch durch Hobbyprojekte hohe Schäden entstehen können
2. aber wir wissen jetzt, was wir tun können um Risiken zu reduzieren
3. damit wir auch weiterhin Spaß am Tüfteln und Programmieren haben können und werden.

ENDE

Danke für‘s Durchhalten!

82 / 83

Quellen

Referenzen:

- *Basiswissen medizinische Software*
  Johner, Hölzer-Klüpfel, Wittorf

- IEC 61508 (*Funktionale Sicherheit sicherheitsbezogener elektrischer/elektronischer/programmierbarer Systeme)*

Bildquellen:

- http://cathykennedystories.blogspot.ca/2013_07_01_archive.html, Folie 9
- http://de.wikipedia.org/wiki/Datei:Edcor1.jpg (CC-sa-3.0, Settembrini), Folie 13
- http://www.flickr.com/photos/91198056@N00/4583788998 (CC-by-2.0), Folien 18, 19, 20
- http://safetycenter.navy.mil/photo/archive/archive_151-200/photo192.htm, Folie 23
- http://commons.wikimedia.org/wiki/File:Windows_XP_BSOD.png, (public domain), Folie 24
- https://openclipart.org/, (alle anderen)