

Verlässliche Echtzeitsysteme

Einleitung

Wintersemester 2024

Peter Wägemann

Lehrstuhl für Systemsoftware

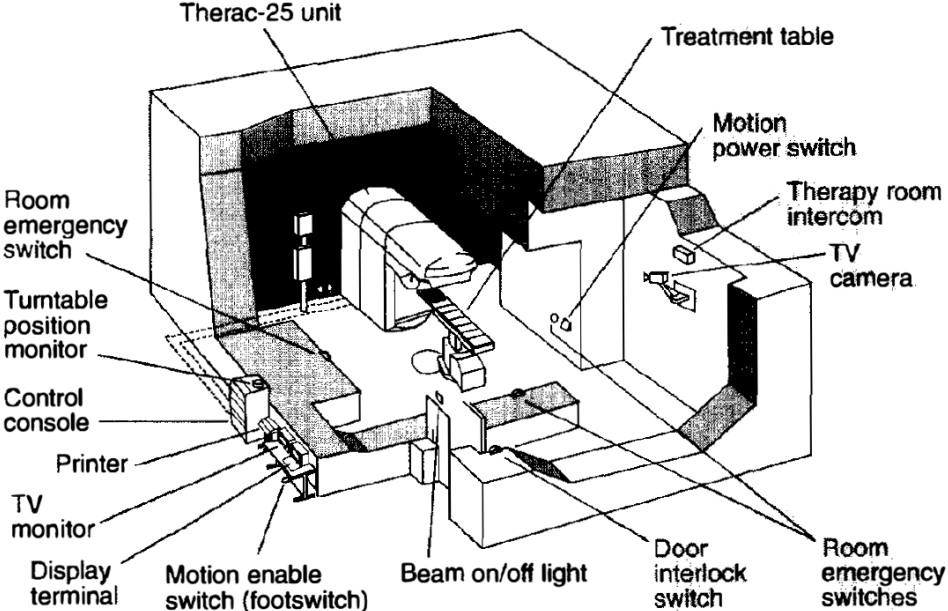
Friedrich-Alexander-Universität Erlangen-Nürnberg

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

- 1 Therac-25
- 2 Ariane 5
- 3 Mars Climate Orbiter
- 4 Toyota Camry
- 5 Boeing 737 Max
- 6 Weitere berühmte Softwarefehler

Softwarefehler und ihre Konsequenzen

- Echtzeitsysteme sind häufig in unser **tägliches Leben eingebettet**
 - Interagieren vielfältig und häufig mit anderen Systemen und Menschen
 - Fehlfunktionen können **katastrophale Folgen** haben
 - Gefahr für Leib und Leben, finanzieller Schaden, ...
 - Einsatz erfordert großes Vertrauen in die verwendete Technik
 - Beispiele: Automobile, Industrieanlagen, Medizingeräte, Luftfahrt
- ☞ Sicherheitskritische Systeme (engl. *safety-critical systems*)
 - Mit hohen Anforderungen an die funktionale Sicherheit (engl. *functional safety*)
- Korrekte Funktion zu garantieren ist eine große Herausforderung ...und gelingt leider nicht immer ...



Entstehungsgeschichte

frühe 70er

Therac-6 6 MeV, Röntgenstrahlung

Therac-20 20 MeV, Röntgenstrahlung und Elektronenstrahlen

- Sicherungssysteme waren allesamt **mechanisch/elektrisch**

Mitte der 70er AECL begann die Entwicklung des Therac-25

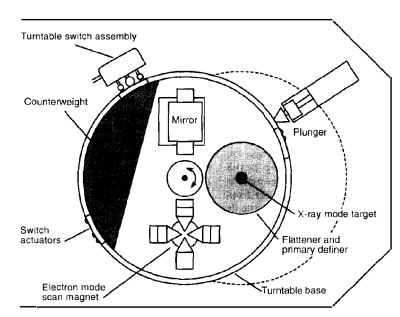
- Neuartiger Doppelweg-Linearbeschleuniger (kleiner, billiger)
- Betriebsmodi: Röntgenstrahlung (25 MeV), Elektronenstrahlen
- Kontrollrechner (DEC PDP11) und Bedienterminal (VT100)
- **Sicherungssysteme durch Software ersetzt**

1976 Erster Prototyp ohne Steuerung durch den Kontrollrechner

1982 - 1985 Fertigung und Auslieferung

- Installationen in elf amerikanischen und kanadischen Kliniken

- Gerät unterstützte **verschiedene Modi**
 - Ausrichtung des Strahlengangs
 - Mithilfe eines Lichtkegels/Spiegels
 - Elektronenstrahlen variablen Energieniveaus
 - Justierung durch Ablenkmagnete
 - Röntgenstrahlen (25 MeV)
 - Erzeugt durch ein Wolfram-Target
 - Mit einem Kollimator gebündelt/ausgerichtet



(Quelle: Nancy Leveson [6])

Der Operateur ...

1. Im Behandlungsraum
 - Patienten \leadsto Behandlungstisch
 - Stellt Strahlengang etc. ein
2. Verlässt den Behandlungsraum
3. am Bedienterminal
 - Eingabe der Behandlungsparameter
 - Behandlungsart, Energieniveau, ...
4. Steuerrechner überprüft Eingabe
 - Freigabe im Erfolgsfall

- Basierend auf der Therac-6-Firmware (Entwicklungsbeginn 1972)
 - Ein Entwickler, PDP11-Assembler, Portierung ab 1976
- In Software implementierte Aufgaben
 - Systemüberwachung** Behandlung verhindern/pausieren/abbrechen
 - Parameterprüfung** Für manuelle Eingaben des Operators
 - Initialisierung** Für die Behandlung (Magnete aktivieren ...)
 - Elektronenstrahl** Kontrollieren: deaktivieren/aktivieren
- Proprietäres Echtzeitbetriebssystem (in Assembler implementiert)
 - Vorranggesteuerte, verdrängende Ablaufplanung
- Programmartefakte der Anwendung
 - Daten – zur Kalibrierung und über den Patienten
 - Unterbrechungsbehandlungen – Zeitgeber, „Power up“, Konsole ...
 - Zeitkritische Aufgaben – Treatment Monitor, Servo, Housekeeper
 - Nicht-zeitkritische Aufgaben – Checksummenberechnung, Verarbeitung der Konsole (Tastatur, Bildschirm), Kalibrierung, Snapshot, ...

Reihe schwerer Zwischenfälle

- Kennestone Regional Oncology Center – 3. Juni 1985
 - Geplant: 10 MeV Elektronenstrahl, Patientin beklagt Schmerzen, nie aufgeklärt
- Ontario Cancer Foundation – 26. Juli 1985
 - Geplant: Elektronenstrahl \leadsto HTILT (NO DOSE) (Operateur wiederholt 4x)
 - Patient erhält Überdosis ($\geq 13\,000$ Rad), verstirbt jedoch krankheitsbedingt
 - AECL gibt fehlerhaftem Taster Schuld
- East Texas Cancer Center – 21. März 1986
 - Geplant: 22 MeV Elektronenstrahl (180 Rad) \leadsto Malfunction 54 (wiederholt)
 - Patient beschreibt „elektrischer Schlag“ und seine Hand „verließe den Körper“
 - Patient verstirbt 5 Monate später an Überdosis (16 500 – 25 000 Rad)
- East Texas Cancer Center – 11. April 1986
 - Geplant: 10 MeV Elektronenstrahl \leadsto Malfunction 54
 - Patient beschreibt „Feuer“, „Lichtblitze“, „Geruch von Verbranntem“
 - Patient verstirbt 2 Wochen später an Überdosis ($\sim 25\,000$ Rad)
- Yakima Valley Memorial Hospital – 17. Januar 1987
 - Geplant: Filmüberprüfung und anschließend Photonenbestrahlung (78 Rad)
 - Patient beschreibt „brennen“ im Brustbereich, sichtbare Verbrennungen
 - Patient verstirbt 3 Monate später an Überdosis (8 000 – 10 000 Rad)

Softwarefehler 1: Was war passiert?

```
PATIENT NAME   : JOHN DOE
TREATMENT MODE : FIX      BEAM TYPE: X      ENERGY (MeV): 25

                ACTUAL      PRESCRIBED
UNIT RATE/MINUTE      0          200
MONITOR UNITS         50 50      200
TIME (MIN)            0.27      1.00

GANTRY ROTATION (DEG) 0.0          0      VERIFIED
COLLIMATOR ROTATION (DEG) 359.2      359      VERIFIED
COLLIMATOR X (CM)     14.2          14.3      VERIFIED
COLLIMATOR Y (CM)     27.2          27.3      VERIFIED
WEDGE NUMBER          1            1      VERIFIED
ACCESSORY NUMBER      0            0      VERIFIED

DATE   : 84-OCT-26  SYSTEM : BEAM READY  OP.MODE: TREAT AUTO
TIME   : 12:55. 8   TREAT  : TREAT PAUSE   X-RAY 173777
OPR ID : T25VO2-RO3 REASON  : OPERATOR   COMMAND:
```

(Quelle: Nancy Leveson [6])

- Bedienung nach einigen Monaten Eingewöhnung ...
 - Operateur verlässt den Raum, gibt Behandlungsparameter ein
 - Eingabefehler: x anstelle von e (Röntgen- statt Elektronenstrahl)
 - Schnelle Korrektur des Fehlers mit der Cursor-Taste
 - Behandlung wurde mit der Meldung „**Malfunction 54**“ pausiert
 - Bedeutung: „dose input 2“ - die Strahlendosis ist zu hoch/niedrig
 - Behandlung wurde gewohnheitsmäßig mit p fortgesetzt

Softwarefehler 1: Kritischer Wettlauf (1)

Rekonstruktion [6] basiert auf Information von AECL, aber nicht umfassend

- Aufgabe „Treatment Monitor“ (Treat) kontrolliert

Behandlungsablauf

- Besteht aus acht Subroutinen
- Steuerung durch die Variable Tphase
- Plant sich am Ende erneut ein

```
void Task_Treat() {  
    switch(TPhase) {  
        case 0: Reset(); break;  
        case 1: DataEnt(); break;  
        ...  
        case 3: SetUp_Test(); break;  
        ...  
        default: ...  
    }  
    reschedule_task(Task_Treat);  
}
```

- Subroutine DataEnt kommuniziert mit der Tastaturbehandlung

- Nebenläufig zu Treat \leadsto geteilte Variable DataEntComplete
 - DataEntComplete == 1 \leadsto Tphase = 3: Dateneingabe fertig
 - Sonst: Tphase bleibt unverändert, DataEnt wird erneut ausgeführt
- DataEntComplete == 1 garantiert, dass Endposition erreicht wurde
 - **Nicht**, dass Cursor noch dort \leadsto spätere Eingaben gehen u. U. verloren
 - Dateneingabe wird u. U. beendet, bevor alle Änderungen eingegeben

- Tastaturbehandlung sichert Modus \mapsto Variable meos

- Byte 0 \mapsto Position der Drehscheibe je nach Betriebsmodus
- Byte 1 \mapsto weitere Betriebsparameter (Konsistenz zu Byte 0 ist wichtig!)

Softwarefehler 1: Kritischer Wettlauf (2)

```
void DataEnt() {  
    if(specified(meos)) {  
        init_params(meos);  
        Magnet();  
        if(changed(meos))  
            return;  
    }  
    if(DataEntComplete)  
        Tphase = 3;  
    if(!DataEntComplete) {  
        if(reset())  
            Tphase = 0;  
    }  
}}
```

```
void Magnet() {  
    setMagnetFlag();  
    while(moreMagnets()) {  
        setNextMagnet();  
        Ptime();  
        if(changed(meos))  
            return;  
    }  
}}
```

```
void Ptime() {  
    while(delay()) {  
        if(magnetFlag()) {  
            if(editing() &&  
                changed(meos) )  
                return;  
        }  
    }  
    resetMagnetFlag();  
}
```

■ Routine DataEnt

- Setzt Betriebsparameter (↪ siehe meos)
- Initialisiert die Ablenkmagnete (↪ Magnet)
- Aktualisiert ggf. Tphase

■ Routine Magnet

- Initialisiert Magnet für Magnet
 - Angezeigt durch das Flag MagnetFlag
- Wartet mit Ptime eine Zeitspanne ab
 - Ca. 1 Sekunde je Ablenkmagnet
 - ↪ Insgesamt ca. 8 Sekunden für 8 Magnete

■ Routine Ptime

- Wartet die Verzögerung aktiv ab
- Setzt MagnetFlag zurück
 - Eingaben werden nur beim 1. Aufruf erkannt
 - Weitere Aufrufe führen diese Überprüfung nicht durch

Softwarefehler 1: Auslösung & Behebung

Auslösung: Fehleingabe durch Operateur (falscher Modus)

- ~> Korrektur innerhalb von 8 Sekunden
- ~> Änderung blieb unbemerkt (Ptime hatte das Flag zurückgesetzt)
- ~> DataEnt beendet die Dateneingabe
- ~> Aufgabe „Hand“ übernimmt **neuen Wert** aus meos
 - Der Drehteller aktiviert den Elektronenstrahlmodus
 - übrige Betriebsparameter sind für Röntgenstrahlung eingestellt

Fehlerbehebung: (siehe Folie II/10 und Folie II/11)

- Zusätzliches Flag `cursorOnCommandLine`
 - Eingabe dauert an, falls Cursor nicht auf der Kommandozeile
- `MagnetFlag` wird am Ende von `Magnet` zurückgesetzt
 - Nicht mehr durch Ptime wie ursprünglich implementiert
 - Etwaige Änderungen werden nun nicht mehr „übersehen“

Softwarefehler 2: Ein fataler Ganzzahlüberlauf

```
void Setup_Test() {  
    if(test()) {  
        Class3++;  
    }  
  
    if(F$mal == 0)  
        Tphase = 2;  
  
    return;  
}
```

```
void Lmtchk() {  
    if(Class3 != 0)  
        Chkcol();  
}
```

```
void Chkcol() {  
    if(col != treat)  
        F$mal |= 0x100;  
}
```

- Variable `Class3` wird gesetzt, wenn der „Lichtkegel/Spiegel“(-Testmodus) aktiviert wird
- Routine `Setup_Test`
 - Inkrementiert `Class3` im Testmodus
 - Fragt `F$mal` ab, um den Kollimator zu prüfen
- Routine `Lmtchk`
 - Ruft `Chkcol` auf, falls `Class3` gesetzt ist
- Routine `Chkcol` prüft die Kollimatorposition
 - Setzt ggf. Bit 9 der Variable `F$mal`

Problem: `Class3` ist eine 1 Byte große Ganzzahlvariable

- `Setup_Test` wird wiederholt und häufig aufgerufen
 - Beim 256. Aufruf läuft `Class3` über
 - Die Kollimatorposition wird nicht überprüft
 - Routine `Setup_Test` wird beendet, der Elektronenstrahl aktiviert

Softwarefehler 2: Auslösung und Behebung

Auslösung: Wechsel des Betriebsmodus

- Operateur kontrolliert die Position des Patienten
 - Hierfür wird der Modus „Lichtkegel/Spiegel“ aktiviert
- Anschließend: Set-Knopf oder Set-Kommando
 - Exakt wenn `Class3` überläuft
- Fehlstellung des Kollimators wird nicht überprüft/erkannt
 - Variable `F$ma1` hatte den Wert 0 (`Chkcol` wurde nicht angerufen)
 - Der Elektronenstrahl wurde mit 25 MeV aktiviert

Fehlerbehebung: die Variable `Class3` wird nicht inkrementiert

- Stattdessen wird `Class3` auf einen Wert > 0 gesetzt

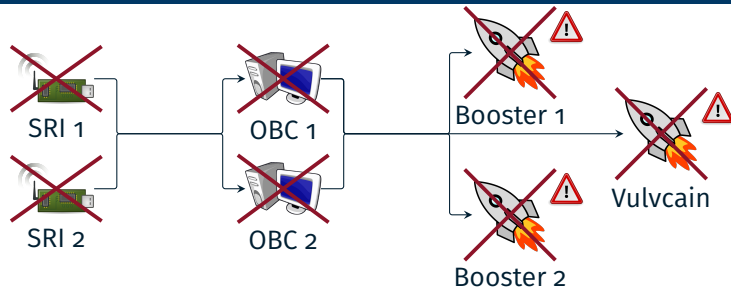
- Musterbeispiel für **schlechte Softwareentwicklung**
 - **Mangelhafte Qualität** des Softwareprodukts
 - Produkt wurde schlampig entworfen und implementiert
 - Entwicklungsdokumentation war praktisch nicht vorhanden
 - Kryptische Fehlermeldungen, die häufig auftraten
 - ...
 - **Mangelhafte Organisation** der Softwareentwicklung
 - Ein einziger Entwickler für Entwurf, Implementierung und Test
 - Praktisch keine Qualitätssicherungsmaßnahmen
 - Kein systematisches Vorgehen beim Testen (nur Systemtest)
 - ...
- Negativbeispiel für den **Umgang mit den Geschehnissen**
 - Nutzer wurden nicht umfassend über Vorkommnisse informiert
 - Die Operateure glaubten, eine Überdosis könne nicht auftreten
 - Fehler wurden nicht rigoros untersucht und beseitigt
 - Was sicherlich mit der mangelhaften Qualität der Software zu tun hat
 - ...

- 1 Therac-25
- 2 Ariane 5**
- 3 Mars Climate Orbiter
- 4 Toyota Camry
- 5 Boeing 737 Max
- 6 Weitere berühmte Softwarefehler

- ESA-Ministerrat bewilligt die Entwicklung (1987)
 - Nachfolgerin der Ariane 4
 - 60% höhere Nutzlast, bei 90% der Kosten
 - Angestrebte Zuverlässigkeit: 99% bzw. 98,5% (für ein- bzw. zweistufige Ariane 5-Variante)
 - Entwicklungskosten: 5,8 Milliarden €
- Technische Merkmale der Grundausführung
 - Zwei Feststoffbooster
 - 238 Tonnen Festtreibstoff, Brenndauer: 130s
 - Durchschnittlich 4400 kN (max. 6650 kN) Schub
 - Eine große Hauptstufe
 - 158 Tonnen Treibstoff, Brenndauer: 605s
 - Vulcain-Triebwerk: 1180 kN Schub



Ariane 5, Flugnummer 501, 4. Juni 1996



$H_0 + 36,70s$ die Inertialmesssysteme SRI1 und SRI2 fallen aus

$H_0 + 37,00s$ starke Schwenkung der Rakete

$H_0 + 39,10s$ Bordcomputer OBC1 fällt aus

$H_0 + 39,80s$ Nutzlast und Verkleidung wird abgetrennt

$H_0 + 40,25s$ Booster2 wird abgetrennt, Selbstzerstörung eingeleitet

$H_0 + 41,90s$ Bordcomputer OBC2 und Steuer-Telemetrie fallen aus

$H_0 + 43,00s$ Hauptstufen-Telemetrie fällt aus

$H_0 + 66,00s$ manueller Zerstörungsbefehl

Was ist geschehen?

■ Unbehandelter Ganzzahlüberlauf im Inertialmesssystem

```
P_M_DERIVE(T_ALG.E_BH) := UC_16S_EN_16NS(TDB.T_ENTIER_16S  
                                     ((1.0/C_M_LSB_BH) *  
                                     G_M_INFO_DERIVE(T_ALG.E_BH)))
```

- Bestimmt die Horizontalbeschleunigung als 64-bit Fließkommazahl
- Konvertiert das Ergebnis in eine 16-bit Ganzzahl
- Folge ist ein **Absturz und Ausfall beider Inertialmesssysteme**
 - Statt Lageinformation werden **nur noch Diagnosenachrichten** übertragen
- Bordcomputer **interpretieren die Diagnoseinformation falsch**
 - Und gehen von einer großen Abweichung der Trajektorie aus
 - Ein **fatales Korrekturmanöver** wird eingeleitet
 - Die Düsen der Booster und der Hauptstufe werden voll ausgeschwenkt
- Die Ariane 5 hält den enormen Luftwiderstand nicht aus
 - Sie beginnt zu zerbrechen
 - Die automatische Selbstzerstörung wird eingeleitet

Wie konnte das geschehen?

- Warum trat der Ganzzahlüberlauf auf?
 - Betroffene Implementierung wurde von der Ariane 4 übernommen
 - **Unterschiedliche Trajektorien** von Ariane 4 und Ariane 5
 - Höhere Horizontalbeschleunigungen und Nickwinkel
 - Letztendlicher Auslöser für den Überlauf
- Warum wurde der Überlauf nicht behandelt?
 - Beschränkung der CPU-Auslastung auf 80%
 - Nur 4 von 7 Variablen wurden gegen Operandenfehler geschützt
- Warum fielen beide Inertialmesssysteme zugleich aus?
 - SRI1 und SRI2 waren identisch \leadsto **homogene Redundanz**
 - In SRI1 und SRI2 trat **derselbe Überlauf** auf

Brisant: eigentlich hätte es das nicht gebraucht ...

- Kalibrierung liefert nur **vor dem Start** sinnvolle Daten
 - Nach dem Start werden die Daten nicht mehr benötigt
 - In der Ariane 4 lief die Kalibrierung noch weitere 40 Sekunden
 - In der Ariane 5 gab es diese Anforderung nicht mehr

- Beispiel für Fehler bei **Entwurf und Auslegung auf Systemebene** [5]
 - Anforderungen an das Inertialmesssystem waren fehlerhaft
 - 16 Bit waren einfach zu wenig
 - Homogene Redundanz war in diesem Fall nicht adäquat
 - Sonst hätte man entsprechende Gleichtaktfehler ausschließen müssen
 - Die Kalibrierung hätte nicht mehr ausgeführt werden dürfen
 - Die Anforderung der Ariane 4 existierte bei der Ariane 5 nicht mehr
 - ...
- Konsequenzen: ein sehr, sehr teurer Fehlschlag ...
 - Finanzieller Schaden: ca. 290 Millionen €
 - Verzögerung des Cluster-Programms (Nutzlast) um 4 Jahre
 - Glücklicherweise keine Personenschäden

- 1 Therac-25
- 2 Ariane 5
- 3 Mars Climate Orbiter**
- 4 Toyota Camry
- 5 Boeing 737 Max
- 6 Weitere berühmte Softwarefehler

Mars Climate Orbiter (MCO)

- Mars-Sonde der NASA
 - Untersuchungen
 - Marsklima, Marsatmosphäre
 - Veränderungen der Marsoberfläche
 - Kommunikationsrelais für den „Mars Polar Lander“
 - Missionsstart: 11.12.1998
- technische Eckdaten
 - 338 kg, 2,1 m x 1,6 m x 2 m
 - Sonnensegel: 5,5 m, 500 W
 - NiH_2 -Batterien: 16 Ah
 - Steuerung: Schubdüsen
 - Trajektorie – 4 x 22 N
 - Lage – 4 x 0,9 N

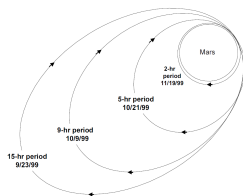


(Quelle: NASA)

- Steuerrechner: IBM RAD6000
 - Takt: 5, 10 oder 20 MHz
 - 128 MB RAM, 18 MB Flash
- Kosten
 - Orbiter&Lander Mission: 327,6 M\$
 - Entwicklung: 193,1 M\$
 - Start: 91,7 M\$
 - Durchführung: 42,8 M\$

Eintritt in den Orbit durch „Aerobraking“

- Manöver zum Eintritt in den Orbit
 - „Berühren“ der Mars-Atmosphäre
 - Der MCO wird dadurch abgebremst
 - Sonnensegel verstärkt Bremseffekt
- MCO umkreist den Mars elliptisch
 - Ellipsen ziehen sich enger
 - Aufgrund der Abbremsung
 - Bis kreisförmiger Orbit erreicht ist
- „Trajectory Correction Maneuver 4“ (TCM4) am 8. September 1999
 - Als Vorbereitung auf den Eintritt in den größten elliptischen Orbit
 - Angepeilt war eine erste Periapsisdistanz von ca. 226 km
- „Mars Orbital Insertion“ (MOI) am 23. September 1999
 - Eintritt in den Funkschatten: 09:04:52 UTC, Austritt ...
- bereits vorher musste man die Periapsisdistanz korrigieren
 - Zwischen TCM4 und MOI: ca. 150km - 170km, 24h vorher: ca. 110km



(Quelle: NASA)

Was war passiert?

- Die Trajektorie des MCO musste korrigiert werden \leadsto TCM₄
 - Grund war vor allem das asymmetrische Sonnensegel
 - Schwungräder auf dem MCO mussten in eine ausgeglichene Lage gebracht werden („Angular Momentum Desaturation“ – AMD)
- Ablauf der Kurskorrektur



1. Bei jedem AMD-Ereignis werden Sensordaten zur Basisstation geschickt
2. Die Daten für die Ansteuerung der Schubdüsen werden berechnet
3. Die Kurskorrektur wird mit den berechneten Daten durchgeführt

Wenn zwei sich nicht verstehen ...

- MCO \mapsto **metrische Größen**, Bodenstation \mapsto **imperiale Größen**
 - Die Werte unterscheiden sich um den Faktor 4,45
- Kräfte der Schubdüsen wurde um den Faktor 4,45 unterschätzt
 - Überkorrektur der Trajektorie \leadsto Periapsisdistanz von ca. 57 km

Mars Climate Orbiter: Resümee

- Untersuchungskommission: zahlreiche organisatorische Mängel [2]
 - Zu wenig Personal für die Überwachung der Mission
 - Zu wenig erfahrenes Personal
 - ...
- ☞ Der Fehler hätte korrigiert werden können
 - Auch noch während des Anflugs zum Mars
- Andere Betrachtungsweise aus Informatik-Sicht:
- ☞ Schnittstellen sollten statisch überprüfbar sein [8]
 - Laut dem Autor – Bjarne Stroustrup – eignet sich dafür natürlich vor allem C++ besonders gut für diese Aufgabe ;-)

- 1 Therac-25
- 2 Ariane 5
- 3 Mars Climate Orbiter
- 4 Toyota Camry**
- 5 Boeing 737 Max
- 6 Weitere berühmte Softwarefehler

Toyota – ungewollte Beschleunigungen



- Toyota Camry L4 (2005)
- Ungewollte Beschleunigungen (engl. unintended acceleration)
- 89 Opfer [4]
- Nachträgliche Analyse der Software: NASA, Barr Group [1]
- Fehlende Mechanismen zur Härtung
 - Keine redundante Ausführung
 - Kein ECC-Speicher
 - Keine manuelle Duplizierung einiger kritischer Variablen
 - Kein Speicherschutz (MPU)

Typen von Software-Defekten

- Puffer-Überläufe
- Ungültige Pointer & fehlerhafte Zeigerarithmetik
- Nebenläufigkeitsprobleme
- Unsicheres Casting
- Stack-Überläufe (und fehlende Laufzeitüberwachung des Stacks)

Toyota – Metriken & Analysen der Software [1]

- McCabe's Cyclomatic Complexity (MCC-Werte kleiner 10 sind akzeptabel)
- 67 Funktionen > 50
- Funktion zur Bestimmung des Gaspedalwinkels: MCC > 100
- MISRA-C Regeln missachtet
 - Verwendung von Rekursion
 - Verwendung von Funktionszeigern
- 80000 Verletzungen von MISRA-C-Regeln
- Unzureichende Testüberdeckung
- Fehlerhafte Analyse des **Worst-Case-Stackverbrauchs**
 - Teile von Bibliotheksfunktionen nicht mit analysiert
- Fehlende OSEK-Kompatibilität

Softwareentwicklungsprozesse mangelhaft

- Fehlende Peer-Reviews von Code
- Kein Bug-Tracking
- Fehlender Source-Code (von Funktionalität auf überwachender CPU)
- Fehlendes Wissen bei Entwicklung nach V-Modell

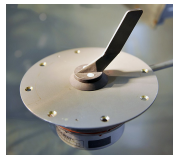
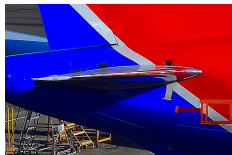
Watchdog-Design unzulänglich

- Mechanismus: regelmäßiges Zurücksetzen von Timer
- Ziel: Überlastsituationen, Aufgaben ohne Fortschritt erkennen
- Camry L4: fehlerhafte Watchdog-Verwendung
 - Überlast-Situationen bis 1,5 Sekunden
 - Zurücksetzen von Watchdog in Timer-Interrupt

- 1 Therac-25
- 2 Ariane 5
- 3 Mars Climate Orbiter
- 4 Toyota Camry
- 5 Boeing 737 Max**
- 6 Weitere berühmte Softwarefehler

- 2018, 189 Opfer, Indonesien
- 2019, 157 Opfer, Äthiopien
- *Boeing Was 'Go, Go, Go' to Beat Airbus With the 737 Max [7]*
- Konkurrenzprodukt: Airbus A320 neo
- Boeing: effizientere, aber größere Turbine an existierende 737-8
- Korrektur der Flugeigenschaften: Maneuvering Characteristics Augmentation System (MCAS)

Boeing 737 Max – MCAS



- Flugverhalten: Flugzeug neigt standardmäßig nach oben
- Ziel: Flugverhalten von 737 nachahmen
- Alternative: Schulungen der Piloten
- MCAS: Korrektur der Fluglage
- Angle-of-Attack-Sensor (Neigungswinkel) überschreiben
- Kritische Bruchstelle: Sensor

<https://aviationweek.com/aerospace/boeing-737-max-mcas-explained>

https://en.wikipedia.org/wiki/File:Southwest_Airlines,_Boeing_737-8_MAX,_N8709Q_horizontal_stabilizer.jpg

https://en.wikipedia.org/wiki/File:Angle_attack_sensor_hg.jpg

- 1 Therac-25
- 2 Ariane 5
- 3 Mars Climate Orbiter
- 4 Toyota Camry
- 5 Boeing 737 Max
- 6 Weitere berühmte Softwarefehler**

Weitere berühmte Softwarefehler

- Fehlfunktion einer MIM-104 Patriot Abwehrrakete [3]
 - 25. Februar 1991, Dhahran - Saudi Arabien (während des Irak-Kriegs)
 - Eintreffende Scud-Rakete wurde nicht erfasst, 28 Soldaten starben
 - **Ursache:** Rundungsfehler (Konvertierung 0,1 → Fließkommazahl)
- Stromausfall im Nordosten der USA, 14. August 2003
 - Ein lokaler Stromausfall wurde übersehen
 - **Ursache:** Race Condition im Überwachungssystem von GE
- „Smart Ship“ USS Yorktown manövrierunfähig, 21. September 1997
 - Ein Besatzungsmitglied tippte direkt eine 'o' ein
 - **Ursache:** die folgende „Division durch 0“ verursachte einen Totalabsturz
- Auflistung weiterer berühmter und berüchtigter Softwarefehler
 - [Http://de.wikipedia.org/wiki/Programmfehler](http://de.wikipedia.org/wiki/Programmfehler)
 - [Http://en.wikipedia.org/wiki/List_of_software_bugs](http://en.wikipedia.org/wiki/List_of_software_bugs)

[1] Barr, M. :

Bookout V. Toyota – 2005 Camry L4 Software Analysis.

https://web.archive.org/web/20140228020145/http://www.sddt.com/files/BA_SLIDES.pdf, 2013

[2] Board, M. C. O. M. I. ; Laboratory, J. P. ; NASA, U. S.:

Mars Climate Orbiter Mishap Investigation Board: Phase I report / Jet Propulsion Laboratory.

1999. –

Forschungsbericht. –

ftp://ftp.hq.nasa.gov/pub/pao/reports/1999/MCO_report.pdf

[3] Carlone, R. ; Blair, M. ; Obenski, S. ; Bridickas, P. :

Patriot Missile Defense: Software Problems Led to System Failure at Dhahran, Saudi Arabia / United States General Accounting Office.

Washington, D.C. 20548, Febr. 1992 (GAO/IMTEC-92-26). –

Forschungsbericht

[4] CBS News:

Toyota “Unintended Acceleration” Has Killed 89.

<https://www.cbsnews.com/news/toyota-unintended-acceleration-has-killed-89/>, 2010

[5] Le Lann, G. :

An analysis of the Ariane 5 flight 501 failure – a system engineering perspective.

In: *Proceedings of International Conference and Workshop on Engineering of Computer-Based Systems (ECBS 1997)*.

Washington, DC, USA : IEEE Computer Society, März 1997. – ISBN 0-8186-7889-5, S. 339-346

[6] Leveson, N. ; Turner, C. :

An investigation of the Therac-25 accidents.

In: *IEEE Computer* 26 (1993), Jul., Nr. 7, S. 18-41.

<http://dx.doi.org/10.1109/MC.1993.274940>. –

DOI 10.1109/MC.1993.274940. –

ISSN 0018-9162

[7] New York Times:

Boeing Was 'Go, Go, Go' to Beat Airbus With the 737 Max.

<https://www.nytimes.com/2019/03/23/business/boeing-737-max-crash.html>, 2019

[8] Stroustrup, B. :

Software Development for Infrastructure.

In: *IEEE Computer* 45 (2012), Jan., Nr. 1, S. 47–58.

<http://dx.doi.org/10.1109/MC.2011.353>. –

DOI 10.1109/MC.2011.353. –

ISSN 0018–9162