Verlässliche Echtzeitsysteme

Einleitung

Fabian Scheler

Friedrich-Alexander-Universität Erlangen-Nürnberg
Lehrstuhl Informatik 4 (Verteilte Systeme und Betriebssysteme)

`www4.informatik.uni-erlangen.de`

15. April 2013

Softwarefehler und ihre Konsequenzen

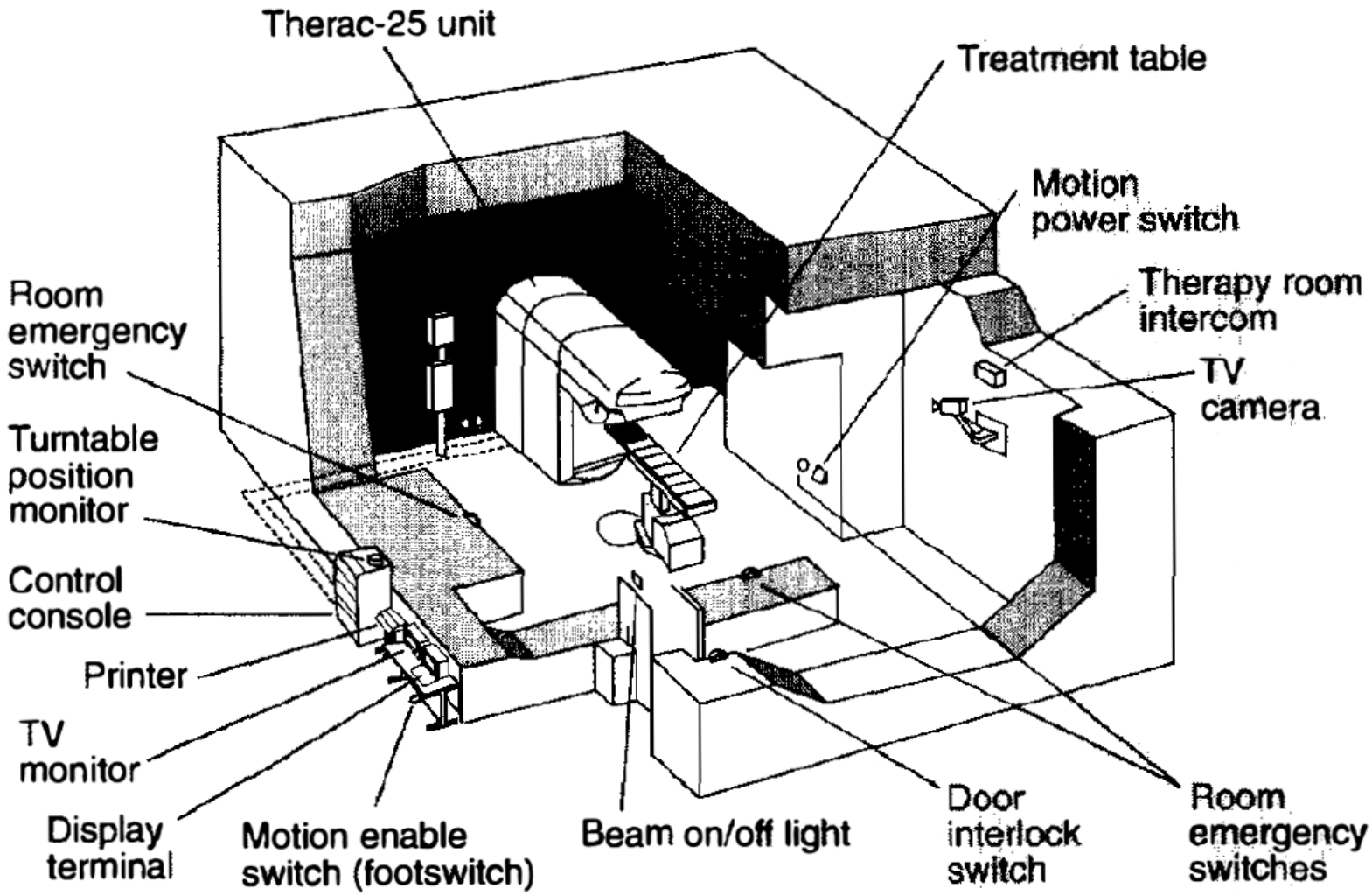
- Echtzeitsysteme sind häufig in unser tägliches Leben eingebettet
  - sie interagieren vielfältig und häufig mit anderen Systemen und Menschen
  - Fehlfunktionen können katastrophale Folgen haben
    - Gefahr für Leib und Leben, finanzieller Schaden, ...
  - ihr Einsatz erfordert großes Vertrauen in die verwendete Technik
  - Beispiele: Automobile, Industrieanlagen, medizinische Geräte, Luftfahrt

☞ sicherheitskritische Systeme (engl. *safety-critical systems*)
  - mit hohen Anforderungen an die funktionale Sicherheit (engl. *functional safety*)

- die korrekte Funktion zu garantieren ist eine große Herausforderung
  - und gelingt leider nicht immer ...
  - Linearbeschleuniger Therac-25 ↦ II/3 ff.
  - Trägerrakete Ariane 5
  - Mars Climate Orbiter

Therac-25

Linearbeschleuniger für den Einsatz in der Strahlentherapie



(Quelle: Nancy Leveson [2])

Entstehungsgeschichte

frühe 70er Kooperation von AECL und CGR

Therac-6 6 MeV, Röntgenstrahlung

Therac-20 20 MeV, Röntgenstrahlung und Elektronenstrahlen

- basierend auf „Neptune“ bzw. „Sagittaire“ von CGR
  - erweitert um einen Kontrollrechner (DEC PDP11)
  - Sicherungssysteme waren allesamt mechanisch

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)
  - mechanische Sicherheitssysteme wurden durch Software ersetzt

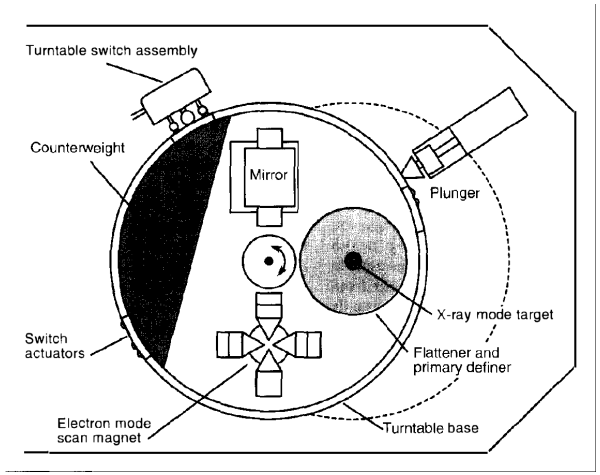
1976 erster Prototyp ohne Steuerung durch den Kontrollrechner

1982 - 1985 Fertigung und Auslieferung

- Installationen in elf amerikanischen und kanadischen Kliniken

Handhabung/Funktionsweise

- verschiedene Modi per Drehrad einstellbar
  - 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 [2])

- Behandlungsablauf: der Operateur . . .

1. im Behandlungsraum
   - Patienten $\rightsquigarrow$ Behandlungstisch
   - stellt Strahlengang etc. ein
2. verlässt den Behandlungsraum
3. am Bedienterminal
   - Eingabe der Behandlungsparameter
   - Behandlungsart, Energieniveau, . . .
4. Steuerrechner überprüft Eingabe
   - Freigabe im Erfolgsfall

Betriebssoftware/Firmware

- baut auf der Firmware des Therac-6 auf (Entwicklungsbeginn 1972)
  - ein Entwickler, implementiert in PDP11-Assembler, Portierung ab 1976
- in Software implementierte Aufgaben

  Systemüberwachung Behandlung verhindern/pausieren/abbrechen
  Parameterprüfung für manuelle Eingaben des Operateurs
  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, . . .

East Texas Cancer Center – 21. März 1986

- geplante Behandlung: 22 MeV-Elektronenstrahl, 180 Rad, 10 cm x 17 cm, insgesamt 6000 Rad über 6,5 Wochen
  - Operateur verlässt den Raum, gibt Behandlungsparameter ein
    - Eingabefehler: `x` anstelle von `e` (Röntgen- statt Elektronenstrahl)
    - schnelle Korrektur des Fehlers mit der Cursor-Taste
    - die übrigen Parameter wurden mit der Return-Taste bestätigt
    - Parameter werden bestätigt
  - die Behandlung wird durch einen Druck auf `b` gestartet (1)
  - die Behandlung wurde mit der Meldung „Malfunction 54“ pausiert
    - Bedeutung: „dose input 2“ - die Strahlendosis ist zu hoch/niedrig
    - Behandlung wurde durch einen Druck auf `p` fortgesetzt (2)
    - um sofort wieder mit derselben Fehlermeldung pausiert zu werden
- Aussage des Patienten zu dem Vorfall
  1. wie ein „elektrischer Schlag“ und „heißer Kaffee“ auf seinem Rücken
  2. „elektrischer Schlag“ am Arm und seine Hand „verließe seinen Körper“
- der Patient verstarb 5 Monate später an den Folgen der Überdosis
  - schätzungsweise 16500-25000 Rad, 1 cm x 1cm, 1 Sekunde

East Texas Cancer Center – 11. April 1986

- geplante Behandlung: 10 MeV-Elektronenstrahl, 7 cm x 10 cm, selber Operateur wie am 21. März 1986
  - Hergang gleicht dem Vorfall am 21. März 1986
    - Eingabefehler (`x` statt *e*) $\leadsto$ Korrektur mit Cursor- und Enter-Taste
    - Beginn der Behandlung durch Einschalten des Strahls (Taste `t`)
    - Behandlung wird nach der Fehlermeldung „Malfunction 54“ pausiert
  - durch die Gegensprechanlage sind laute Geräusche zu hören
  - Operateur stürmt ins Behandlungszimmer
- Aussage des Patienten zum Vorfall
  - fühlt „Feuer“ an der Seite seines Gesichts (auf Nachfrage)
  - etwas traf ihn an der Seite des Gesichts und er sah einen Lichtblitz
  - ein zischenedes Geräusch, das an kochende Eier erinnnert
    - Sättigung der Ionenkammer zur Detektion einer Überdosis
    - Ionenkammern waren bereits bei 804 Rad gesättigt
- der Patient starb am 01. Mai 1986 an den Folgen der Strahlendosis
  - AECL schätzte nach eigenen Messungen eine Dosis von ca. 25000 Rad

Softwarefehler 1: Dateneingabe wird u. U. ignoriert . . .

Rekonstruktion [2] basiert auf Information von AECL, ist 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` $\rightsquigarrow$ geteilte Variable `DataEntComplete`
    - `DataEntComplete == 1` $\rightsquigarrow$ `Tphase = 3`: Dateneingabe abgeschlossen
    - sonst: `Tphase` bleibt unverändert, `DataEnt` wird erneut ausgeführt
  - `DataEntComplete == 1` garantiert, dass die Endposition erreicht wurde
    - nicht, dass der Cursor noch dort ist $\rightsquigarrow$ spätere Eingaben gehen u. U. verloren
    - Dateneingabe wird u. U. beendet, bevor alle Änderungen eingegeben wurden
- Tastaturbehandlung sichert Modus/Energieniveau $\mapsto$ Variable `meos`
  - Byte 0 $\mapsto$ Position der Drehscheibe je nach Betriebsmodus
  - Byte 1 $\mapsto$ weitere Betriebsparameter (Konsistenz zu Byte 0 ist wichtig!)

. . . aber nur wenn der Operateur schnell genug ist?

```
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();
}
```

- die Routine `DataEnt` . . .
  - setzt Betriebsparameter ($\rightsquigarrow$ siehe `meos`)
  - initialisiert die Ablenkmagnete ($\rightsquigarrow$ `Magnet`)
  - aktualisiert ggf. `Tphase`
- die Routine `Magnet` . . .
  - initialisiert Magnet für Magnet
    - angezeigt durch das Flag `MagnetFlag`
  - wartet mit `Ptime` eine Zeitspanne ab
    - ca. 1 Sekunde je Ablenkmagnet

  $\rightsquigarrow$ insgesamt ca. 8 Sekunden für 8 Magnete
- die Routine `Ptime`
  - wartet die Verzögerung aktiv ab
  - setzt `MagnetFlag` zurück
    - Eingaben werden nur beim 1. Aufruf erkannt
    - die weiteren Aufrufe führen diese Überprüfung nicht durch

Softwarefehler 1: Auslösung & Behebung

Auslösung: Fehleingabe durch den 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/9 und Folie II/10)

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

Yakima Valley Memorial Hospital – 17. Januar 1987

- geplante Behandlung: Filmüberprüfung (3 Rad, 22 cm x 18 cm und 4 Rad, 35 cm x 35 cm) und Photonenbestrahlung, 79 Rad
  - nach der Filmüberprüfung kontrolliert der Operateur die korrekte Position des Patienten auf dem Behandlungstisch
    - Drehteller $\rightsquigarrow$ Position „Lichtkegel/Spiegel“ mit der manuellen Konsole
  - anschließend wurde der Drehteller in die Behandlungsposition gebracht
    - durch drücken der Set-Taste oder durch die Eingabe eines Set-Kommandos
  - startet die Behandlung durch einen Druck auf die Taste `b`
    - die Maschine pausiert $\rightsquigarrow$ keine Dosis oder Dosisrate wird angezeigt
    - Operateur setzt die Behandlung fort (Taste `P`), Maschine pausiert erneut
    - über die Gegensprechanlage ist der Patient zu hören
- Aussage des Patienten zu dem Vorfall
  - er verspüre ein Brennen im Brustbereich
  - später: Verbrennung der Haut im gesamten bestrahlten Bereich
- der Patient verstarb im April an den Folgen der Strahlendosis
  - AECL schätzte anhand von Messungen eine Dosis von 8000-10000 Rad

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;
}
```

- die Variable `Class3` wird gesetzt, wenn der „Lichtkegel/Spiegel“(-Testmodus) aktiviert wird
- die Routine `Setup_Test`
  - inkrementiert `Class3` im Testmodus
  - fragt `F$mal` ab, um den Kollimator zu prüfen
- die Routine `Lmtchk`
  - ruft `Chkcol` auf, falls `Class3` gesetzt ist
- die Routine `Chkcol` prüft die Kollimatorposition
  - und setzt ggf. Bit 9 der Variable `F$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
  - ⇝ die 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
  - und zwar genau dann, wenn `Class3` überläuft
- die Fehlstellung des Kollimators wird nicht überprüft/erkannt
  - die Variable `F$mal` hatte den Wert 0 (`Chkcol` wurde nicht augerufen)

  $\rightsquigarrow$ der Elektronenstrahl wurde mit 25 MeV aktiviert

Fehlerbehebung: die Variable `Class3` wird nicht inkrementiert

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

Therac-25: Resümee

- 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
  - ...

Gliederung

Weitere berühmte Softwarefehler

- Fehlfunktion einer MIM-104 Patroit Abwehrrakete [1]
  - 25. Februar 1991, Dhahran - Saudi Arabien (während des Irak-Kriegs)
  - eintreffende Scud-Rakete wurde nicht erfasst, 28 Soldaten starben
  - Ursache: Rundungsfehler (Konvertierung 0,1 $\mapsto$ 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 '0' ein
  - Ursache: die folgende „Division durch 0“ verursachte einen Totalabsturz
- Auflistung weiterer berühmter und berüchtigter Softwarefehler
  - `http://de.wikipedia.org/wiki/Programmfehler`
  - `http://en.wikipedia.org/wiki/List_of_software_bugs`

Literaturverzeichnis

[1] Carlone, R. ; Blair, M. ; Obenski, S. ; Bridickas, P. :
Patriot Missile Defense: Software Problme Led to System Failure at Dhahran, Saudi Arabia / United States General Accounting Office.
Washington, D.C. 20548, Febr. 1992 (GAO/IMTEC-92-26). –
Forschungsbericht

[2] 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