

Hochschule Darmstadt

Wissenschaftliches Arbeiten in der Informatik

Therac-25 - Probleme mit der Nebenläufigkeit

Max Stümpfl und Vera Schmitt
SS 2016
15.6.2016

Betreuung: Inge Schestag, Fachbereich Informatik, Hochschule Darmstadt

Eidesstattliche Versicherung

Hiermit erklären wir an Eides Statt, dass wird die vorliegende Arbeit selbstständig und nur unter Zuhilfenahme der ausgewiesenen Hilfsmittel angefertigt haben. Sämtliche Stellen der Arbeit, die im Wortlaut oder dem Sinn nach anderen gedruckten oder im Internet verfügbaren Werken entnommen sind, haben wir durch genaue Quellenangaben kenntlich gemacht.

Ort, Datum

Vorname Nachname

In unserer schriftlichen Ausarbeitung zum Thema „Therac 25 – Problem mit der Nebenläufigkeit“, werden wir in folgenden Abschnitten untersuchen, wie es bei dem medizinisch eingesetzten Gerät Therac-25 zu den schlimmsten Zwischenfällen in 35 Jahren Strahlentherapie kommen konnte. Hierbei werden wir sowohl auf technischen, als auch auf menschliche Versäumnisse, die während der Entwicklung des Geräts, sowie während seiner Inbetriebnahme gemacht wurden eingehen. Unser Ziel wird sein, auf benötigte Qualität von Software sowohl beim Entwickeln als auch beim Prüfen aufmerksam zu machen. Unser Schwerpunkt wird auf der Software, besonders auf dem Thema Multithreading und Entwicklung liegen.

Inhaltsverzeichnis

1	Einführung in Therac-25	1
1.1	Therac-25	1
1.2	Die Hardware	1
1.3	Die Software	1
2	Einführung in die Nebenläufigkeit	4
2.1	Nebenläufigkeit	4
2.2	Threading	4
2.3	Speichermodelle	4
2.4	Zustände	5
2.5	Probleme	5
3	Die Fehlfunktionen von Therac-25	6
3.1	Bekannte Bugs	6
3.1.1	Texas-Bug	6
3.1.2	Washington Bug	6
3.2	Hintergründe	7
3.3	Versäumnisse	7
3.4	Reaktion auf die Unglücke	8
4	Fazit	8
5	Literaturverzeichnis	

1 Einführung in Therac-25

1.1 Therac-25

Therac 25 ist ein medizinisch eingesetzter Linearbeschleuniger, zur Behandlung von Karzinomen, der zur dritten Reihe der Therac-Serie, der kanadischen Firma AECL (Atomic Energie of Canada Limited) gehört. Er wurde 1984 freigegeben und von 1985 bis Januar 1987 gebaut und eingesetzt. Softwarefehler und besonders fahrlässiges Verhalten bei der Fehlersuche und deren Behebung führten zu sechs Todesfällen, die zu den schlimmsten Zwischenfällen in 35 Jahre Strahlentherapie zählen. Insgesamt wurden elf Geräte der Therac-25-Serie in Kanada und den USA installiert.[3][1]

1.2 Die Hardware

Therac-25 ist ein Dual-Mode Gerät, welches erlaubt Patienten sowohl mit Röntgen-, als auch mit Elektronenstrahlung zu bestahlen. Für die Bestrahlung wird ein Drehteller mit einer Bleiplatte verwendet, mit deren Hilfe Strahlung gebündelt und je nach Behandlungsart, gezielt eingesetzt werden kann. Der Drehteller nimmt eine von drei Positionen, Röntgen-, Elektronen-, oder Standbyposition, ein, welche mit Hilfe von drei Sensoren gemessen und von einem Computer geprüft und verarbeitet wird. Auf mechanische Sicherheitsmechanismen, sogenannte „Hardware Interlocks“, die es bei den Vorgängermodellen, Therac-6 und Therac-20, gab, wird bei Therac-25 komplett verzichtet.[1]

Das Gelingen der Strahlentherapie hängt von der richtig gewählten Strahlendosis des Geräts ab. Um eine ausreichende Dosis während der Therapie mit Elektronenstrahlen durch die Bleiplatte sicherstellen zu können, muss eine um ein vielfaches höhere Strahlendosis erzeugt werden. Tritt währenddessen ein Fehler auf, bei dem der Drehteller auf dem Röntgen- oder Standby-Modus steht, hat dies schwerwiegende Folgen für den Patienten, der dann mit einer extremen Überdosis bestrahlt wird. Tritt der Zwischenfall während der Röntgenbestrahlung ein, kommt es bei dem Patient zu einer Unter- oder leichten Überdosis. Beides gefährdet das erfolgreiche Abschließen der Therapie und führt im schlimmsten Fall zum Tod des Patienten.[3]

1.3 Die Software

PATIENT NAME	:TEST				
TREATMENT MODE	:FIX	BEAM TYPE: X	PENERGY(MeV): 25		
UNIT RATE/MINUTE					
MONITOR UNITS		ACTUAL	PREScribed		
TIME (MIN)		0	200		
		50 50	200		
		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	:T25V02-R03	REASON :OPERATOR	COMMAND		

In Anlehnung an: [9]

Als Basis der Software dient ein selbst-programmiertes Echtzeitbetriebssystem „32K PDP 11/23“, welches präemptives Scheduling von kritischen und nicht kritischen Tasks unterstützt. Die Tasks werden nach einem bestimmten Algorithmus gewichtet und anschließend seriell ausgeführt. Auf Synchronisationsoperationen wurde verzichtet. Die Software ist in PDP-11 Assembler geschrieben und verfügt über folgende Komponenten:

- Daten (Kalibrierungstabellen, Einstellungen der Maschine, Patientendaten)
- Scheduler (Ablaufplan)
- Satz von kritischen und unkritischen Aufgaben
- Interrupted Services (Fehlerbehandlung und Initialisierung des Systems)

Die Aufgaben der Software bestanden aus:

- Überwachung des Maschinenstatus
- Vergleichen von eingestellten Patienten- mit eingegebenen Daten
- Einstellung der Maschine
- Abbruch der Bestrahlung nach Ende der Behandlung oder bei Fehlermeldung
- Überwachung der Sicherheitssperre (Interlocks)

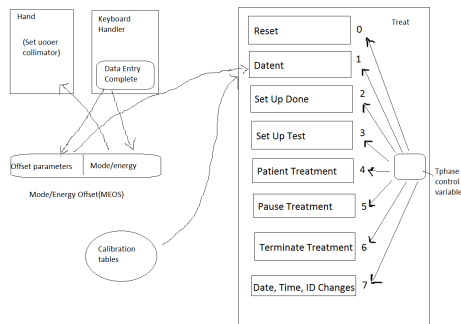
Die Unterteilung in kritische und unkritische Tasks und deren Ausführung haben maßgeblich zu den Unfällen beigetragen. (Mehr dazu in Kapitel „Bugs“ und „Hintergründe“). Zu den kritischen Tasks mit deren Aufgaben von Therac-25 zählen:

- Treatmentmonitor (Treat):
Er dirigiert und überwacht die Platzierung und Behandlung des Patienten in 8 Operationsschritten, welche als Subroutinen aufgerufen werden. Durch eine Variable „TPhase“ wird eine jeweilige Phase und die dazu gehörende Subroutine bestimmt. Nachdem eine Subroutine ausgeführt ist, wird die Variable „TPhase“ inkrementiert und es folgt ein rekursiver Aufruf des Treats.
- Servo Task:
Er kontrolliert die Funktion der Strahlenkanone, die Symmetrie der Strahlungsspannung sowie die Motorik der Maschine
- Housekeeper Task:
Er gibt Fehlermeldungen an das Display weiter und kontrolliert die Fehlermonitore

Zu den unkritischen Tasks gehören:

- Treatment Console Keyboard Processor:
Er bildet die Schnittstelle zum Eingabemedium
- Treatment Console Screen Processor:
Er stellt die Schnittstelle zum Ausgabemedium dar

Die Bedienung von Therac-25 erfolgt über eine Konsole, in der die verschriebenen Werte eingegeben bzw. aus der Patientenkartei geladen werden können. Anschließend werden alle Werte, sowohl die eingegebenen Patientendaten, als auch die gemessenen Werte des Geräts mit Hilfe des Computers geprüft. Sind sie korrekt, werden diese als „VERIFIED“ markiert und die Bestrahlung wird freigegeben. „Bild: Menü“ Die Bedienung des Geräts startet, indem die Taste „SELECT PATIENT“ gedrückt wird und die Liste aller Patienten



In Anlehnung an: [8]

aufgerufen wird. Zum Wählen der Therapie wird „ENTER“ gedrückt was die verschriebenen Einstellungen anzeigt. Für den Beginn der Bestrahlung wird „START“ gedrückt, dies wird so lange fortgesetzt, bis die gewünschte Dosis erreicht wird. Um den einmal ausgeschalteten Stahl wieder in Betrieb nehmen zu können müssen bestimmte Setupprozeduren durchgeführt werden um die Sperrung wieder zu entsperren. Mit „STOP“ kann die Behandlung unterbrochen werden. Wird ein Fehler entdeckt, führt dies ebenfalls zur sofortigen Abschaltung [5].

2 Einführung in die Nebenläufigkeit

In dem folgendem Kapitel wird man einen kleinen Einblick bekommen von der Nebenläufigkeit die eine große Rolle spielt bei den Therac-25 Unfällen, aber dazu mehr in Kapitel 3.

2.1 Nebenläufigkeit

Die Nebenläufigkeit beschreibt in der Informatik die Eigenschaft eines Systems, das mehrere Anweisungen, Berechnungen oder Befehle zur selben Zeit, ohne eine bestimmte Reihenfolge ausführen kann. Sind es mehrere Programme, die zeitgleich arbeiten, so sind diese voneinander abgeschottet und können nicht miteinander kommunizieren. Dies nennt man auch Multitasking. Sind es jedoch mehrere Threads, die zeitgleich arbeiten, spricht man von **Multithreading**, dabei können Threads vom selben Prozess untereinander kommunizieren. Dahingegen sind Threads unterschiedlicher Prozesse, weiterhin voneinander getrennt. [10]

2.2 Threading

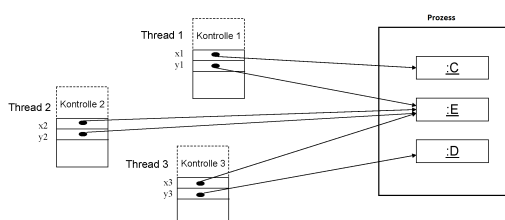
Ein Thread (auch Aktivitätsträger oder leichtgewichtiger Prozess genannt) ist eine Ausführungsreihenfolge eines Prozesses. Ein Prozess kann mehrere Threads haben, wobei alle Threads auf den gemeinsamen Hauptspeicher des Prozesses zugreifen können, um gegenseitig Daten austauschen zu können. Damit die Konsistenz der Daten gewährleistet wird, müssen diese jedoch aktiv verwaltet werden. Sie besitzen weiterhin einen eigenen Programmstack und einen eigenen Befehlszähler, was ihnen den Namen **leichtgewichtiger Prozess** verleiht. [11][13]

Wie bereits in Kapitel 2.1 erwähnt, gibt es auch Multithreading, wobei mehrere Threads nebeneinander arbeiten. Um allerdings echtes Multithreading zu betreiben zu können, benötigt man bestimmte Prozessoren (CPU) die dieses Verfahren unterstützen, was allerdings heute von fast jeder CPU getan wird. Nur wenn eine CPU dieses Verfahren kann, spricht man von *simultanes Multithreading*, da die verschiedenen Threads auf mehrere Kerne der CPU ausgelagert werden und somit parallel arbeiten. Wohin gegen bei älteren CPUs, die nur einen Kern hatten oder Multithreading nicht konnten, jeder Thread ein Zeitfenster bekommt um die jeweilige Anweisung auszuführen. Ist die Zeit abgelaufen und die Anweisung nicht vollendet, wird einfach der nächste Thread ausgeführt, solange wie die Prozesse andauern. Der Wechsel zwischen den Threads passiert so schnell, dass es den Anschein macht für den Benutzer, dass es Multithreading ist, obwohl nur eine Illusion davon erzeugt wird. [14]

Außerdem gibt es noch das sogenannte **Hyper-Threading**, was die Rechenleistung und die Anzahl der zeitgleich ausführenden Operationen nochmals beschleunigen soll. Hierzu wird ein Prozessor in zwei virtuelle Prozessoren geteilt, die wie beim Multithreading unabhängig voneinander zur selben Zeit Threads bearbeiten können. Somit werden auf einem physikalischen Prozessor fast die ganzen Ressourcen benutzt, da fast alle Ausführungseinheiten arbeiten können. Hyper-Threading ist sozusagen die virtuelle Lösung von Intel um simultanes Multithreading zu bewerkstelligen. [15]

2.3 Speichermodelle

In der unten stehenden Abbildung 1 wird auf eine einfache Art veranschaulicht, wie die verschiedenen Threads, beim Multi-threading, eines Prozesses auf die selben Ressourcen zugreifen. Dabei greifen die Threads eins bis drei auf die Variablen/Elementen des Prozesses zu.



In Anlehnung an: [17]

Abbildung 1: Speichermodell

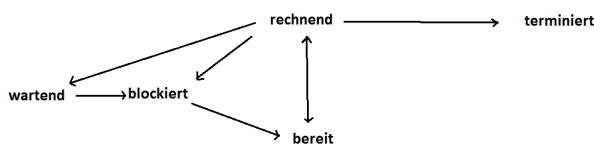
2.4 Zustände

Ein Thread muss mehrere Zustände besitzen, da mehrere Threads sich miteinander synchronisieren müssen. Im Vergleich dazu besitzt ein Prozess nur 3 Zustände, was hier nicht genauer erläutert wird.

rechnend	Der Thread wird im Prozess ausgeführt.
bereit	Der Thread ist rechnen bereit, hat aber nicht Zugriff auf den Prozessor.
blockiert	Der Thread braucht ein Betriebsmittel, das nicht vorhanden ist.
wartend	Der Thread hat ein Betriebsmittel aufgegeben und wartet.
terminiert	Der Thread ist beendet.

Tabelle 1: Zustände

Ein Thread beginnt nachdem er erzeugt wurde im Zustand *bereit* und geht dann durch die 3 anderen Zustände durch, sobald er die Anweisungen erhält. Wird er nicht mehr benötigt, so wird der Thread *terminiert*



In Anlehnung an: [17]

Abbildung 2: Zustände

2.5 Probleme

Ein großes Problem bei Multithreading besteht darin, die Konsistenz zu gewährleisten, da die Threads alle auf die gleichen Objekte im Heap zugreifen. Dies passiert, da man oft mehrere Daten gleichzeitig verändern muss, um von einem konsistenten Zustand in den Nächsten zu gelangen.

Vor allem existiert ein Problem, was auch beim Therac-25 auftrat, wie es dazu kam, wird in Kapitel 3 näher erklärt. Das Problem ist unter dem Namen **race condition** bekannt. Dabei wird von mindestens 2 Threads oder Prozessen zur selben Zeit auf eine Variable zugegriffen. Um dies zu verdeutlichen, wird es anhand eines kleinen Beispiels erklärt. Nehmen wir an, dass Thread A und Thread B zu selben Zeit auf die Variable $X = 0$ zugreifen und Thread A auf X eins addiert und mit der Rechnung innerhalb einer Sekunde fertig ist. X ist nun eins. Thread B addiert allerdings auf X zwei drauf und braucht für diese Berechnung etwas länger, erfährt aber nicht, dass X bereits verändert wurde, also überschreibt Thread B das von Thread A erzeugte Ergebnis mit seinem. Dadurch entsteht ein falsches Ergebnis, was zu Fehlern führt.

Man kann zum Beispiel dafür sorgen, dass nur ein Thread die Möglichkeit hat, über eine gewisse Zeit auf dem Datensatz zu arbeiten, damit dieser in einem konsistenten Zustand bleibt, bevor ein anderer Thread darauf zugreift. Es gibt noch einige weitere Probleme mit Multithreading, die aber nicht von Bedeutung sind für die Zwischenfälle beim Therac-25, daher werden diese nicht erläutert. [12][16]

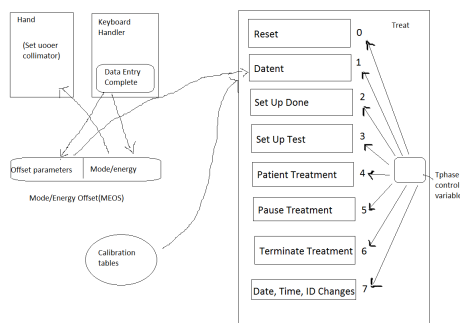
3 Die Fehlfunktionen von Therac-25

Nun werden wir darauf eingehen, welche Funktionen damals beim Therac-25 fehlerhaft agierten, es wird sich aber jedoch hauptsächlich auf die massiven Software Fehler bezogen. Und genauer gesagt, nur auf den Texas- und Washington-Bug, da diese die gefährlichsten Bugs waren, die sich wiederholten und zum Teil Tode verursachten. Die Fehler die nicht so schwerwiegend oder Hardware spezifisch waren, werden nicht behandelt.

3.1 Bekannte Bugs

Die zwei Bugs traten zwischen den Jahren 1985 und 1987 auf und verursachten insgesamt sechs Unfälle mit massiven Überdosen, wobei fast die Hälfte aller Unfälle tödlich endete. Zu Beginn wurden die Vorfälle von der Regierung und von dem Hersteller nicht untersucht, da es ausgeschlossen wurde, dass das Gerät Fehlfunktionen hat. Man kann diese Vorfälle in zwei Gruppen unterteilen, den Texas-Bug und den Washington-Bug.

3.1.1 Texas-Bug



In Anlehnung an: [1]

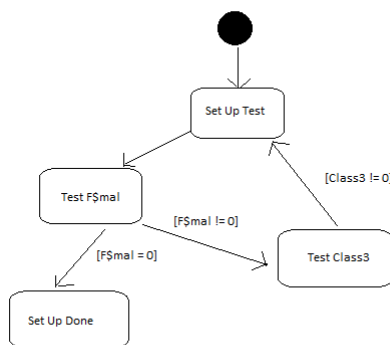
Abbildung 3: Softwaredesign

Ganz besonders dieser Bug zeigt die Schwächen des Softwaredesigns. Bei der Eingabe kommuniziert das *Datent* mit dem *Keyboard Handler* über die gemeinsame Variable *Data entry complete*. Damit wird ein Flag gesetzt, dass die Eingabe beendet ist. Dies geschieht nur durch die Ausrichtung des Cursors, der am rechten unteren Rand des Terminals, siehe Abbildung 1.3, sein muss und nicht durch das Bestätigen eines Buttons. Sobald das Flag gesetzt ist, wird vom *Datent* eine Unterfunktion aufgerufen namens *Magnet*, die die Magneten ausrichten soll, das dauert ungefähr 8 Sekunden. Beim Aufruf von *Magnet* wird ein weiteres Flag gesetzt namens *bending magnet*, dieses soll gelöscht werden, wenn alle Magneten ausgerichtet sind. Um das Ausrichten der einzelnen Magnete zu überprüfen, ruft die Funktion *Magnet* so viele Unterfunktion namens *ptime* auf, die so oft wie die Anzahl an auszurichtenden Magnete aufgerufen wird. Außerdem soll *ptime* überprüfen ob währenddessen noch Benutzereingaben vorgenommen werden. Am Ende von *ptime* soll das *bending magnet* Flag wieder gelöscht werden, welches allerdings bereits beim ersten *ptime* gelöscht wird und nicht nach dem letzten. Dadurch wird nicht die ganze Zeit überprüft, ob Änderungen an den Einstellungen vorgenommen wurden. Die neuen Einstellungen werden nur gespeichert aber nicht berücksichtigt, deswegen kann es passieren der falsche Modus benutzt wird. Hier tritt nun das Problem aus Kapitel 2.5 auf, die *race conditions*, da es wie bereits erklärt wurde es kritische und unkritische Prozesse gibt, wurde das Ausrichten der Magnete als kritisch eingestuft und das verändern der Einstellungen als unkritisch. Das führte dazu das auf dem Display die veränderte Einstellung übernommen wurde, aber die Umsetzung der Änderung an der Maschine blieb allerdings aus. So konnte es dazu kommen das anstatt des Elektronenmodus der Röntgenmodus eingestellt war, obwohl es auf dem Display korrekt angezeigt wurde.[1][3]

3.1.2 Washington Bug

Bei dem Washington-Bug handelt es sich um einen Bug, der nur unter ganz bestimmten Voraussetzungen auftrat. Bei Beginn der Behandlung stellt die bedienende Person die gewünschten Parameter am Terminal

ein. Die Parameterliste im Terminal zeigt währenddessen „UNVERIFIED“, anschließend können noch Feineinstellungen vorgenommen werden. Danach wechselt der Status zu „VARIFIED“ und der Operator wird aufgefordert „SET“ zu drücken. Hat er das Gerät in die richtige Position gebracht, führt das Gerät die Funktion „Set Up Test“ durch. Bei jedem Aufruf wird eine Variable „Class 3“ erhöht. Diese Variable gibt an, ob die Position des Gerätes korrekt ist. Ist die Variable ungleich null herrscht Inkonsistenz (Widerspruch), das Programm springt in die Interrupt-Routine, woraufhin die Behandlung nicht fortgeführt wird. Ist „Class 3“ gesetzt, wird noch eine weitere Variable „F\$mal“ gesetzt, welche der Fehlerbehandlung dient. Tritt eine Fehlfunktion auf, so ist die Variable „F\$mal“ ungleich null und die Funktion „Set Up Test“ wird erneut aufgerufen. Der eigentliche Bug entsteht, wenn die Variable „Class 3“ überläuft. Während der Einstellung des Geräts wird die Funktion „Set Up Test“ mehrere 100-mal zur Kontrolle des Geräts aufgerufen, wobei die 1 Byte große Variable „Class 3“ jedes Mal erhöht wird. Steigt der Wert von „Class3“ über 255(1 Byte), kommt es zu einem „Overflow“ und die Variable erhält wieder den Wert null, trotz, dass ein Fehler vorliegt springt das Programm nicht in seine Interrupt-Routine, prüft nicht die Position des Drehtellers bzw. des Strahlungsarms und erkennt deshalb nicht, dass ein Fehler vorliegt, was das System dazu veranlasst trotz, dass durch „F\$mal“ eine Fehlermeldung vorliegt, die Behandlung fortzusetzen. Drückt die behandelnde Person zu dieser Zeit auf „SET“, erleidet der Patient eine Überdosis, da sich das Gerät noch an der falschen Stelle befindet. [3] [7]



In Anlehnung an: [7]

3.2 Hintergründe

Die Software von Therac-25 hatte den gleichzeitigen Zugriff auf gemeinsame Variablen erlaubt und „Test“ und „set“ waren keine atomaren Anweisungen, diese Implementation führte zu Race Conditions (Siehe Kapitel „Multithreading“) die eine zentrale Rolle bei den Unglücken (Siehe Kapitel „Texas-Bug“ und „Washington-Bug“) spielten. Die Fehlerbehandlung bei Therac 25 konnte die Behandlung auf zwei verschiedene Arten unterbrechen: Zum einen ein „treatment suspend“, welcher das System zurücksetzen und neu starten lies, die zweite Art war „treatment pause“, welches sich mit einer Taste „p“ (proceed) fortsetzen lies, jedoch nur fünf Mal, dann wurde ein Neustart ausgeführt. Die Fehlermeldungen bestanden aus der Bezeichnung „Mal-function“ in Kombination mit einer Zahl von 1 bis 64, diese hatten aber nur in seltenen Fällen mit der Patientensicherheit zu tun, zudem war eine Fehlerhäufigkeit von über 40 Fehlermeldungen an einem Tag sehr häufig.[1]

3.3 Versäumnisse

Die Therac-25 Software wurde von einem einzigen Programmierer über mehrere Jahre entwickelt. Dokumentation, sowie Testverfahren waren nur unzureichend bis gar nicht dokumentiert. Dies machte sowohl die Fehlersuche als auch die Wartung äußerst schwierig. Bei der Entwicklung wurde auf die Suche nach einem ausgereiften, bereits vorhandenes Betriebssystem verzichtet und stattdessen auf ein eigens primitives für den Therac-25 in der eigenen Firma entwickeltes zurückgegriffen. Der Softwaretest bei Therac-25 beschränkte sich zudem nur auf einen Systemtest, der weitgehend beim Kunden durchgeführt wurde und bezog sich fast ausschließlich auf die Hardware [3]. Diverse Fehlermeldungen, die sehr häufig auftraten, ein schlechtes Handbuch[5], sowie ignoranten und naives Verhalten seitens des Konzerns, der alle

Verantwortung und Vorwürfe von sich wies, führten letztendlich zu sechs Todesopfern [6]. Kanadische, sowie US-Amerikanische Behörden stimmen der Zulassung zu, wobei eine Risikoanalyse 1983 ebenfalls ohne Software durchgeführt wurde [3]. Zudem wurden mechanische, bewährte, jedoch benutzerunfreundlichere Sicherheitsmechanismen der Vorgängermodelle gegen potenziell fehleranfälligere softwarebasierte ausgetauscht.

Zusammengefasst:

- Benutzung eines primitiven, eigenen Betriebssystems
- Softwareentwicklung, sowie wichtige Entscheidungen, getroffen durch einen einzigen Programmierer
- Softwaretest beschränkte auf einen Systemtest beim Endverbraucher
- Bequemlichkeit der Endanwender steht über der Sicherheit der Patienten
- Unzureichendes Prüfen und Zulassen des Geräts von Seiten der US-amerikanischen und kanadischen Behörden

[6]

3.4 Reaktion auf die Unglücke

Nach den ersten Vorfällen von Therac-25 wurde ein Techniker von Atomic Energie of Canada Limited(AECL) ins Krankenhaus geschickt um die Sachlage zu untersuchen. Er konnte den Fehler nicht reproduzieren und kam dann zu dem Entschluss, das ein Microswitch defekt sei und meinte, dass alle Benutzer das Gerät vor dem Gebrauch inspizieren sollen. Daraufhin verkündigte AECL, dass die Hardware angepasst wird und, dass das Überprüfen der Software nicht notwendig sei. Nachdem sich ein weiterer Unfall im Jahre 1985 ereignete, reagierte die AECL erst nicht auf jegliche Meldungen und verweigerte eine Aussage. Nach weiteren Unfällen, mit eingeschlossen der Washington Unfall, kam es wieder zur Aussage, dass das Gerät in Ordnung sei. Dies ging, bis eine Technikerin der Tyler-Klinik in Texas eigenständig Tests durchführte und alle Unfälle reproduzieren konnte. Sie schickte die Ergebnisse an die FDA¹, die daraufhin das Therac-25 als schadhaft erklärte und Vorschlag, die Benutzung zu vermeiden. AECL lieferte eine verbesserte Version von Therac-25 beim FDA vor und sicherte zu das alle Fehler behoben seien. Einige Monate später ereignete sich allerdings wieder ein Unfall und der zu begutachtende Techniker behauptete, dass es ein anderer Fehler sein muss. Dieses Verhalten zog sich bis in den Februar des Jahres 1987, bis die Benutzung von Therac-25 eingestellt wurde. Später stellte sich heraus, dass es täglich über 40 Fehlermeldungen gab. Diese seien aber kryptisch gewesen, daher ignorierten die Techniker diese. Außerdem hatten die Techniker ein veraltetes Testverfahren verwendet, welches die Wahrscheinlichkeiten für die Unglücke und deren jeweiligen Ursachen berechnet.[1]

4 Fazit

Man sollte unbedingt aus solchen Fehlern lernen und nicht wie bei den Therac-25 Ereignissen vieles vernachlässigen. Denn es wurden viele Fehler getan, zu aller erst wurde die Software nicht ausgiebig getestet, da der Entwickler es nicht für nötig hielt eine angemessene Testphase durchzuführen. Außerdem wurde komplett auf Hardware-Überwachungskomponenten verzichtet und durch Software-Komponenten ersetzt. Obwohl Hardware basierte Überwachungsmedien in der Regel deutlich stabiler und zuverlässiger sind als die Software-Lösung. Es können sich nämlich immer Fehler einschleichen oder zu *race conditions* kommen. Weiterhin lässt sich sagen, dass man die Fehlermeldungen ,wenn sie auftreten, passend benennen soll oder sie dementsprechend im Handbuch auflisten soll, mit ausreichender Erklärung. Beides war bei Therac-25 nicht der Fall und so wurden die Fehlermeldungen, wovon täglich fast 40 Stück auftraten, einfach ignoriert, denn niemand wusste was damit anzufangen war, nicht einmal die Techniker der Herstellerfirma. Das darf natürlich nicht passieren, was aber entsteht wenn man eine neue Software nicht entsprechend Testet und dokumentiert. Die neue Software entstand aus der alten Software de Vorgängermodellen und wurde mit neuem Code erweitert, welche natürlich zu neuen Bugs führt, die man davor nicht kannte. Diese Vorfälle haben aber letztendlich dazu beigetragen, dass die Gesetzte für das Freigeben von neuen Geräten in dieser Branche verschärft wurden und das in der Zukunft mehr Aufmerksamkeit auf das Testen der Software gelegt wird. Dies trägt dazu bei wie wir heute in der Softwareentwicklung agieren.

¹U.S. Food and Drug Administration, Department of Health and Human Services

5 Literaturverzeichnis

- [1][http :
//formal.iti.kit.edu/beckert/teaching/Seminar – Softwarefehler – SS03/Ausarbeitungen/pfeifer.pdf](http://formal.iti.kit.edu/beckert/teaching/Seminar-Softwarefehler-SS03/Ausarbeitungen/pfeifer.pdf)
- [2][http : //www.inf.fu – berlin.de/inst/ag – se/teaching/V – AWS – 2011/22Sicherheit.pdf](http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-AWS-2011/22Sicherheit.pdf)
- [3][http : //www5.in.tum.de/lehre/seminare/semsoft/unterlagen₀2/therac/website/](http://www5.in.tum.de/lehre/seminare/semsoft/unterlagen02/therac/website/)
- [4][https : //www.youtube.com/watch?v = nxX – aAvZbmM\(1.4.16,15 : 20\)](https://www.youtube.com/watch?v=nxX-aAvZbmM(1.4.16,15:20))
- [5][http : //www4.in.tum.de/lehre/seminare/ps/WS0203/desaster/Herbers – Hasan – Therac – Ausarbeitung –
18 – 11 – 02.pdf](http://www4.in.tum.de/lehre/seminare/ps/WS0203/desaster/Herbers-Hasan-Therac-Ausarbeitung-18-11-02.pdf)
- [6][https : //dpunkt.de/leseproben/1736/Kapitel₁.pdf](https://dpunkt.de/leseproben/1736/Kapitel1.pdf)
- [7][http : //www.erenkrantz.com/Geeks/Therac – 25/Figure.4.gif](http://www.erenkrantz.com/Geeks/Therac-25/Figure.4.gif)
- [8][http : //www5.in.tum.de/lehre/seminare/semsoft/unterlagen₀2/therac/website/swskizze.gif](http://www5.in.tum.de/lehre/seminare/semsoft/unterlagen02/therac/website/swskizze.gif)
- [9][http : //www5.in.tum.de/lehre/seminare/semsoft/unterlagen₀2/therac/website/interface.gif](http://www5.in.tum.de/lehre/seminare/semsoft/unterlagen02/therac/website/interface.gif)
- [10][http : //www.hs – augsburg.de/meixner/prog/html/nebenlaeu figkeit/Nebenlaeu figkeit.html#Situation](http://www.hs-augsburg.de/meixner/prog/html/nebenlaeufigkeit/Nebenlaeufigkeit.html#Situation)
- [11][http : //www.scalingbits.com/java/javakurs2/programmieren2/threads](http://www.scalingbits.com/java/javakurs2/programmieren2/threads)
- [12][http : //www.drdoobs.com/tools/avoiding – classic – threading – problems/231000499](http://www.drdoobs.com/tools/avoiding-classic-threading-problems/231000499)
- [13][http : //www.nt.th – koeln.de/fachgebiete/inf/diplom/proc_sync/index2.html](http://www.nt.th-koeln.de/fachgebiete/inf/diplom/proc_sync/index2.html)
- [14][http : //www.itwissen.info/definition/lexikon/Multithreading – multithreading.html](http://www.itwissen.info/definition/lexikon/Multithreading-multithreading.html)
- [15][http : //www.itwissen.info/definition/lexikon/Hyperthreading – HT – hyper – threading.html](http://www.itwissen.info/definition/lexikon/Hyperthreading-HT-hyper-threading.html)
- [16][http : //www.oreilly.de/german/freebooks/linuxdrive2ger/charrace.html](http://www.oreilly.de/german/freebooks/linuxdrive2ger/charrace.html)
- [17][https : //www.pst.ifi.lmu.de/lehre/SS06/infoII/folien/Folien15Threads1SS06.pdf](https://www.pst.ifi.lmu.de/lehre/SS06/infoII/folien/Folien15Threads1SS06.pdf)