Softwaresicherheit

Ausarbeitung zum Überfachlichen Seminar
Medizinische Informatik

Inhaltsverzeichnis

1 Bekannte Softwarefehler

1.1 Harmlosere Fehler

1.1.1 Was für Raketen gut ist, kann doch für ein Flugzeug nicht schlecht sein

Ein Programmierer der US Air Force wollte in seinem Programm, das für eine Rakete bestimmt war, Platz sparen, Das stellte er so an: Wenn immer die Rakete den Äquator überquerte, sollte sie einfach ihre bisherigen Flugkoordinaten folgen, allerdings mit umgekehrten Vorzeichen. Dieses Vorgehen führte zwar dazu, dass die Rakete beim Überfliegen das Äquators um die eigene Achse rollte. Das störte allerdings niemanden.

Als das Programm jedoch unverändert in den Autopiloten des Jägers F-18 übernommen wurde, flog der Jet beim überfliegen des Äquators plötzlich mit dem Cockpit nach unten weiter. Einfach Pech für den Piloten, nicht wahr?

1.1.2 Umstellungsproblem

Bei der Umstellung auf den Euro ist es bei der Bank 24, einer Tochter der Deutschen Bank, zu Fehlern gekommen. Bei der Abrechnung für das erste Quartal fanden Kunden auf ihren Konto Beträge, die um mehrere Billionen DM falsch waren. Am 6. und 7. April hatten Kunden von Bank 24 plötzlich Millionenbeträge in 13stelliger Höhe auf ihren Konten, etwa ein Betrag von –7 902 343 433 862,49 DM. Andere wurden kurzzeitig zu Millionären.

Bank 24 behauptete, dass es sich um Einzelfälle handele, gab aber keine detaillierte Stellungnahme zur Ursache des Fehler ab.

1.2 Schäden durch Softwarefehler

1.2.1 Umrechnungsfehler

Am 24. September 1999 ging die Raumsonde Mars Orbiter der NASA, die auf dem roten Planeten landen und die Oberfläche erkunden sollte, kurz vor dem Ziel verloren. Das war um so erstaunlicher, als die Sonde bis zu diesen Zeitpunkt ohne Probleme funktioniert hatte.

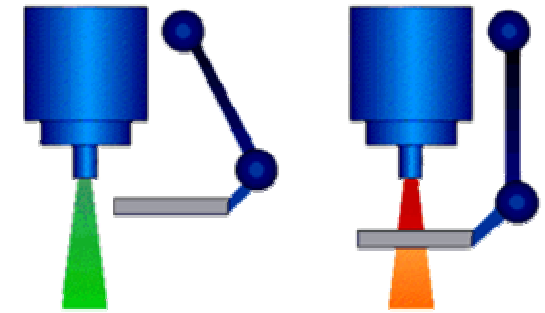
Als Ursache des Fehler stellte sich heraus, dass die Umrechnung verschiedener Maßeinheiten nicht funktioniert hatte. Der Fehler lag in einer Umrechnungstabelle in der Support Software der Kontrollstation in den USA. Dabei ging es darum, den Kurs der Sonde auf dem weg zum Mars durch Korrekturmanöver zu berichtigen. Die am Jet Propulsion Laboratory (JPL) in Kalifornien eingesetzte Software erwartete für den Schub die Einheit Newton, währen man beim Hersteller Lockheed Martin offensichtlich mit Pfunden rechnete. Die Folge war, dass bei den Steuermanövern der Schub während der gesamten Flugzeit um 22 Prozent zu gering ausfiel.

Der Fehler war während des gesamten Fluges zum 220 Millionen Kilometer weit entfernten Mars gegenwärtig, wurde aber offensichtlich nicht bemerkt. Üblicherweise sind die Bahnen derartiger Sonden auf 10 Kilometer genau. Kritisch wurde es erst bei der Annäherung an den roten Planeten. Die Sonde kam der Oberfläche um 100 Kilometer (62 Meilen) zu nahe. Beim Mars Orbiter ist nach den Angaben von Lockheed Martin zunächst der Treibstofftank explodiert, die Trümmer sind dann auf der Marsoberfläche zerschellt. Die Kosten der Sonde werden mit 125 Mill. US$ angegeben.

1.2.2 Krebsklinik

In der zweiten Hälfte der achtziger Jahre wurde in den USA und Kanada eine Reihe von Unfällen mit einer Maschine zur Krebstherapie, der THERAC-25, bekannt. Dieses Gerät war eine Weiterentwicklung der THERAC-23 durch die kanadische Firma Atomic Energy of Canada Limited (AECL). Das Unternehmen war im Besitz der kanadischen Regierung und befasste sich hauptsächlich mit dem Bau von Kernkraftwerken.
Bevor ein neues Medizinisches Geräte in der USA verwendet werden darf muss dieses erst von der FDA (Food and Drug Administration) abgenommen werden. Wenn diese das Gerät als unbedenklich einstufen wird es zugelassen. Allerdings konnten diese nicht alle Fehler finden und so wurde die THERAC-25 zugelassen.

Die Weiterentwicklung THERAC-25 war insofern bemerkenswert, als mit ihr sowohl Röntgenstrahlen (X rays) als auch Elektronenstrahlen verabreicht werden konnten. Außerdem war die Benutzungsführung verbessert worden.
Der *Elektronenmodus* wird zur Behandlung von Geweben verwendet, die nahe an der Körperoberfläche liegen. Um auch tiefergelegene Gewebe effizient und „schonend“ behandeln zu können, wurde das Gerät um einen Röntgenmodus ergänzt, der die Patienten mit Photonen anstelle von Elektronen bestrahlt.



Um tiefer gelegene Gewebe punktuell bestrahlen zu können, ohne großen Schaden an Nachbargeweben zu verursachen, muss die Energie durch eine Bleiplatte gebündelt werden. Da die Strahlen erst einmal die Platte durchdringen müssen, muss eine viel höhere Energie aufgewendet werden (die ca. 100fache Energie).
Es ist offensichtlich, dass es **fatal** ist, wenn ein Patient mit Photonen (Röntgenmodus) bestrahlt wird, ohne dass sich die Platte zwischen Gerät und Patient befindet.
Die Unfälle betrafen im Einzelnen die folgenden Kliniken:

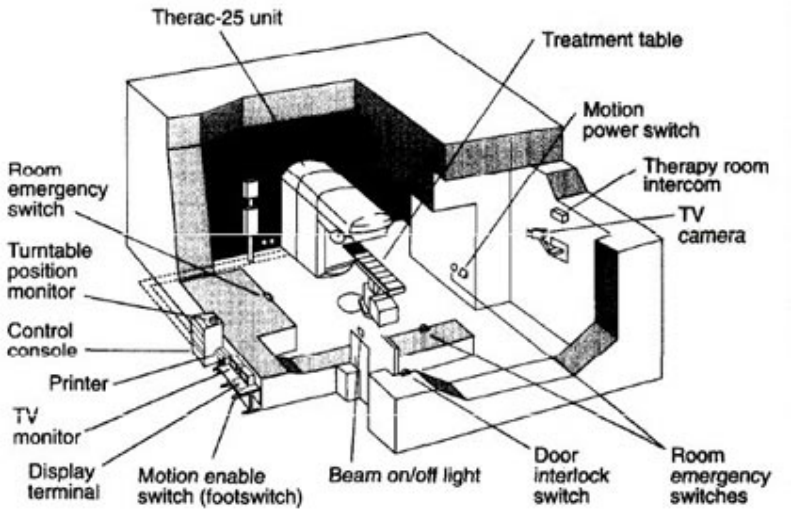
- Maritta Georgia, am 3. Juni 1985: Eine einundsechzigjährige Patientin erhielt eine Dosis von 15 000 bis 20 000 rads, was zu erheblichen Verbrennungen führte. Normale Dosen liegen bei etwa 200 rad. Sie konnten Arm und Schulter nach der Behandlung nicht mehr gebrauchen. Ihre Brüste mussten operativ entfernt werden.
  Es gab lange keine Meldung des Unfalls an die FDA. Gesetzeslage: Hersteller mussten schwere Unfälle melden; Anwender (Krankenhäuser) nicht

- Hamilton, Ontario, Kanada, am 26. Juli 1985: Eine vierzigjährige Frau erhielt eine Strahlendosis im Bereich des Nackens, die zwischen 13 000 und 17 000 rads lag. Sie starb am 3. November 1985 an Krebs.
- Yakima, Staat Washington, Dezember 1985: Eine Patientin erhielt eine zu hohe Strahlendosis, was zu abnormaler Rötung ihrer rechten Hüfte führte.
- Tyler, Texas, 21. März 1985: Ein Mann erhielt eine Dosis zwischen 16 500 und 25 000 rads. Er konnte seinen rechten Arm nicht mehr gebrauchen und starb fünf Monate später an der Folgenden Überdosis.
- Der Unfall im East Texas Cancer Center (ETCC) Der Therac-25 war schon zwei Jahre vor dem Unfall im *East Texas Cancer Center* (ETCC) in Betrieb, mehr als 500 Patienten wurden bis dahin behandelt. Am 21 März 1986 kam ein Patient zu seiner neunten Behandlung ins ETCC, für die Fortsetzung einer vorgeschriebenen Serie von Bestrahlungen zur Entfernung eines Tumors auf seinem Rücken.
  Die geplante Behandlung sollte eine 22 MeV-Behandlung von 180 rads am Oberrücken links von der Wirbelsäule sein, als Teile einer Gesamtbestrahlung von 6,000 rads über sechseinhalb Wochen. Er wurde in den Behandlungsraum genommen und mit dem Gesicht nach unten auf den Behandlungstisch gelegt. Die Operateurin verließ den Behandlungsraum, schloss die Tür, und setzte sich ans Kontrollterminal.
  Da die Operateurin routiniert im Umgang mit der Konsole war, durchlief sie die Folge von Eingabefenster relativ schnell. Bei dieser Behandlung jedoch vertippte sie sich und gab statt „e"-Mode (für Elektron) „x"-Mode (für X-ray) ein. Nachdem sie schon bestätigt hatte, bemerkte die den Fehler und kehrte mit der ↑ Taste zurück zur Mode-Eingabe, wo sie erneut e-Mode auswählte.
  Weil die anderen Parameter korrekt waren, drückte sie lediglich die Eingabetaste und ließ die restlichen Werte unverändert. Der Bildschirm zeigte die Werte der Parameter als überprüft (VERIFIED) an - das Terminal zeigte, wie erwartet, BEAM READY an. Durch die B-Taste für *beam on* wurde die Bestrahlung gestartet. Die Maschine schaltete sich sofort aus und die Konsole zeigte MALFUNCTION 54 (Funktionsstörung) an. Das Blatt, das auf dem Rand der Maschine hing, erklärte, dass dies der Code für einen „Menge Eingabe 2" (dose input 2) Fehler war. Es waren keine anderen Informationen erhältlich, die die Bedeutung des MALFUNCTION 54 erklären hätten könnten.
  Später stellte ein AECL Techniker fest, das MALFUNCTION 54 der Code für eine zu hohe oder zu niedrige Bestrahlung war. Die Meldung war dafür gedacht, lediglich bei firmeninternen Entwicklungsarbeiten aufzutreten.

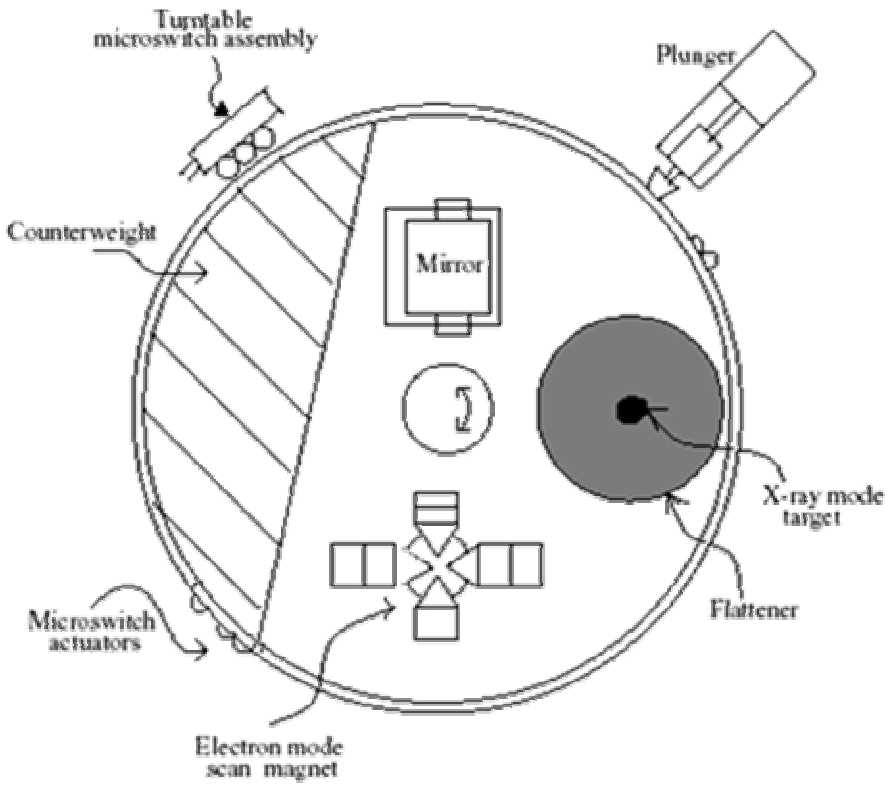
Auf der Bildschirmanzeige zeigte die Maschine eine starke Unterdosierung an: 6 Bildschirmeinheiten waren verabreicht worden, während die Operateurin 202 Einheiten angefordert hatte. Es war keine Seltenheit, dass die Maschine die Behandlung stoppte oder verzögerte. Sie reagierte wie immer wenn die Maschine stoppte und drückte wie gewohnt einfach die P-Taste (proceed) um mit der Behandlung fortzufahren. Wieder schaltete sich die Machine mit derselben Fehlermeldung aus.
Nach dem Behandlungsversuch spürte der Patient enorme Schmerzen wie nach einem Stromschlag in seinem Arm, und hörte ein Summen aus dem Gerät. Ein Arzt untersuchte ihn und stellte bloß eine Errötung der Behandlungszone fest, konnte aber nichts anderes als einen Stromschlag diagnostizieren. Der Patient wurde mit der Anweisung, zurückzukommen falls Komplikationen auftreten sollten, entlassen. Auch der zuständige Physiker des Krankenhauses konnte nicht feststellen, was die Funktionsstörung zu bedeuten hatte. Die Maschine wurde für den Rest des Tages zur Behandlung anderer Patienten benutzt.
In Wirklichkeit, was niemandem zu diesem Zeitpunkt bekannt war, erhielt der Patient jedoch eine massive, im Zentrum der Behandlungszone konzentrierte Strahlenüberdosis (16 500 bis 25 000 rads).
In den nächsten Wochen hatte der Patient weiterhin Schmerzen an der Schulter und am Hals.
Schließlich wurde er mit dem Verdacht auf eine durch Bestrahlung verursachte Rückenmarksentzündung ins Krankenhaus eingeliefert. Mittlerweile waren sein linker Arm und seine Beine ebenso wie seine Stimmbänder gelähmt. Außerdem waren auch sein Darm, seine Harnblase, sein linker Lungenflügel und seine Diaphragma betroffen. Er litt auch an einer Herpesinfektion. Fünf Monate später verstarb er an den Komplikationen der Überdosis.

- Tyler, Texas, 11. April 1986: Ein Mann wurde am Gehirn bestrahl. Anstatt, der vorgesehenen relativen kleinen Dosis erhielt er 4 000 rads. Der Patient starb am 1. Mai an der Folgenden Überdosis.
- Yakima, Staat Washington, Januar 1987: Ein Mann erhielt zwischen 8000 und 10 000 rads. Vorgesehen hatte der Arzt ganze 86 rads. Der Patient starb im April 1987 an Komplikationen, für die zu hohe Strahlenbelastung ursächlich war.

Verbrennungen durch Strahlung



Therac-25



Der Drehteller - Sicht von unten (aus Sicht des Patienten)

1.2.2.1 Drehteller-Positionierung

Entscheidend für eine erfolgreiche Behandlung eines Krebspatienten ist die richtige Postionierung des Drehtellers. Eine falsche Positionierung kann fatale Auswirkungen haben.
Wenn sich der Drehteller in der Position für den Röntgenmodus befindet, also sich die Bleischeibe zwischen Gerät und Patient befindet, das Gerät aber mit der Dosis einer Elektronenbehandlung bestrahlt, resultiert eine Unterdosis, die aber keine größere Auswirkung hat, als dass der Patient unnötig beunruhigt wird.
Wenn sich aber das Gerät in der Position für eine Elektronenbestrahlung befindet, die Maschine aber mit der Energie einer Röntgenbehandlung bestrahlt, kommt es zu einer ungeheuren Überdosis, die ca. das 100fache der für den menschlichen Körper verträglichen Dosis beträgt.
Die Positionierung des Drehtellers wird von einem Computer durchgeführt, der durch bestimmte Sensoren die Drehung der Scheibe kontrolliert. Insgesamt werden die Daten von 3 Sensoren aufgezeichnet.
Die gemessenen Werte werden dann anhand einer Software verglichen und überprüft. Dadurch wird eine mechanische Verriegelung, wie sie noch die beiden Vorgängermodelle Therac-6 und Therac-20 hatten, überflüssig.

1.2.2.2 Die Software

Die Sicherheitssysteme, die vorher aus elektrischen Stromkreisen und Hardware bestanden, wurden bei dem Therac-25 von einem Computer übernommen. Das Computerprogramm, das aus immerhin über 20 000 Anweisungen bestand, wurde von einem einzigen Programmierer in mehreren Jahren geschrieben. Es wurde nicht bekannt, wer dieser Programmierer war, welche Ausbildung er genossen hatte, oder welche Qualifikationen er besaß. Lediglich sein Ausscheiden aus der AECL 1986 wurde bekannt gegeben. Das Programm war nur unzureichend

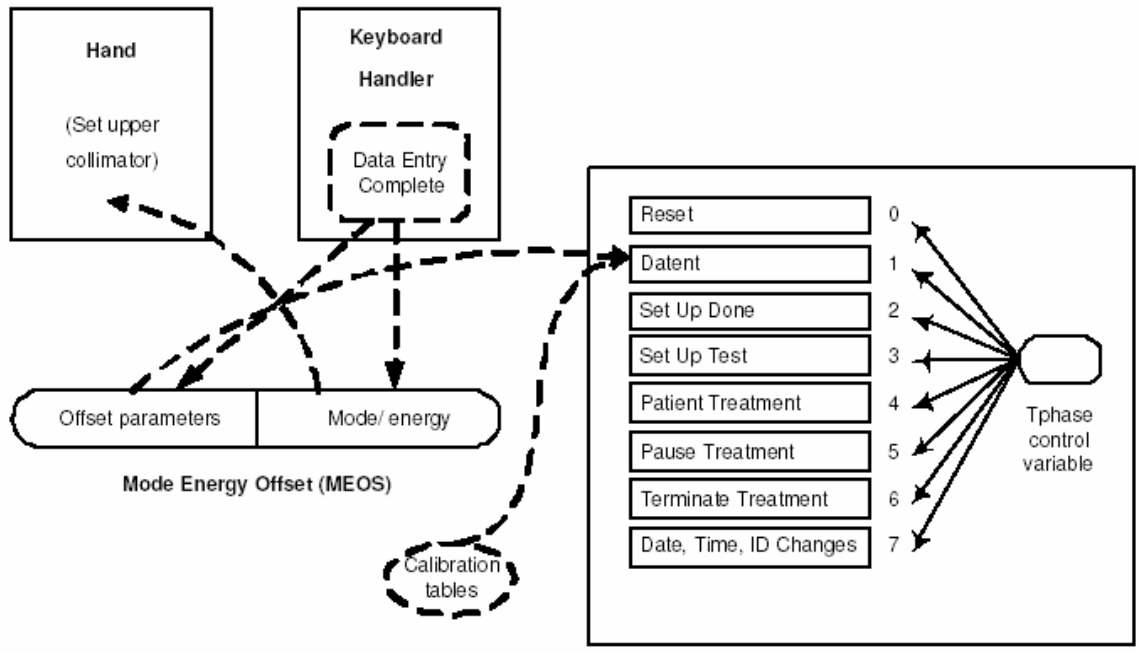
dokumentiert, und es gab noch nicht einmal Hinweise darauf, dass das Programm getestet wurde, bevor es zum Einsatz kam.
Die Therac-25 Software verwendete kein kommerzielles Betriebssystem. Das eigene Betriebssystem war ein Standalone-Realtime-Betriebssystem. Es wurde extra für das Therac-25 geschrieben und lief auf einem 32K PDP-11/23.
Die Software, geschrieben in einer PDP-11 Assembler Sprache, hatte vier Hauptbestandteile: die gespeicherten Daten, einen Scheduler, kritische und nicht kritische Aufgaben und eine Interruptroutine. Die Daten bestanden aus der Kalibrierung und den Patienten- bzw. Behandlungsdaten. Der Scheduler kontrollierte die Events und die gleichzeitigen Prozesse, sofern kein Interrupt vorlag. Kritische Aufgaben der Software waren:

- der Behandlungsmonitor (Treat), der das Patienten Setup und die Behandlung kontrolliert und anzeigt. Dazu verwendet er acht Subroutinen in Abhängigkeit vom Wert der Tphase Kontrollvariablen. Die Treats interagieren mit dem Keyboard Handler, der die Kommunikation zwischen Techniker und Konsole regelt (siehe Abbildung: Texas Bug)
- die Servo Task, der die Bestrahlungsdosis und die Behandlungsparameter verwaltet.
- die Housekeeper Task, der den Systemstatus kontrolliert, verwaltet und auf dem Display anzeigt.

Die unkritischen Aufgaben bestanden unter anderem aus dem Keyboard Prozessor, dem Screen Prozessor und dem Service Keyboard Prozessor (dient der Kommunikation zwischen Behandlungssystem und Techniker).
Die Software hatte den gleichzeitigen Zugriff auf gemeinsame Variablen erlaubt und „Test“ und „set“ waren keine atomaren Anweisungen. Aus dieser Implementation des Multitasking resultierten Race Conditions (Wenn das Ergebnis mehrerer Ereignisse von deren Reihenfolge abhängt und diese Reihenfolge zeitlich nicht garantiert werden kann, dann spricht man von einer Race Condition. Der Name Race Condition leitet sich davon ab, dass mehrere Prozesse eine Art Rennen um die betroffene Ressource austragen, welches nur einer gewinnen kann), die bei den Unglücken eine wichtige Rolle spielten.
Das Therac-25 konnte auf zwei verschiedene Arten die Behandlung im Falle eines Fehlers unterbrechen. Auf der einen Seite gab es einen so genannten treatment suspend, der das System zurücksetzen und neu starten ließ, und der zweite Weg, die Behandlung zu unterbrechen, war die weniger schlimme treatment pause, die aber nur eine einzige Taste (p für proceed) benötigte, um die Behandlung fortzusetzen. Dies konnte man aber nur maximal fünf Mal ausführen, bis das System einen Neustart initiierte.
Die Fehlermeldungen waren kryptisch und bestanden aus dem Wort Malfunction gefolgt von einer Zahl von 1 bis 64. Die Fehlermeldungen hatten aber nur in den seltensten Fällen etwas mit der Sicherheit der Patienten zu tun. Laut Berichten eines Technikers kam es bis zu mehr als 40 Fehlermeldungen an einem Tag. Außerdem wurde dem technischen Personal vermittelt, dass es unmöglich sei, einem Patienten eine Überdosis zu verabreichen.

1.2.2.3 Die Fehlfunktion

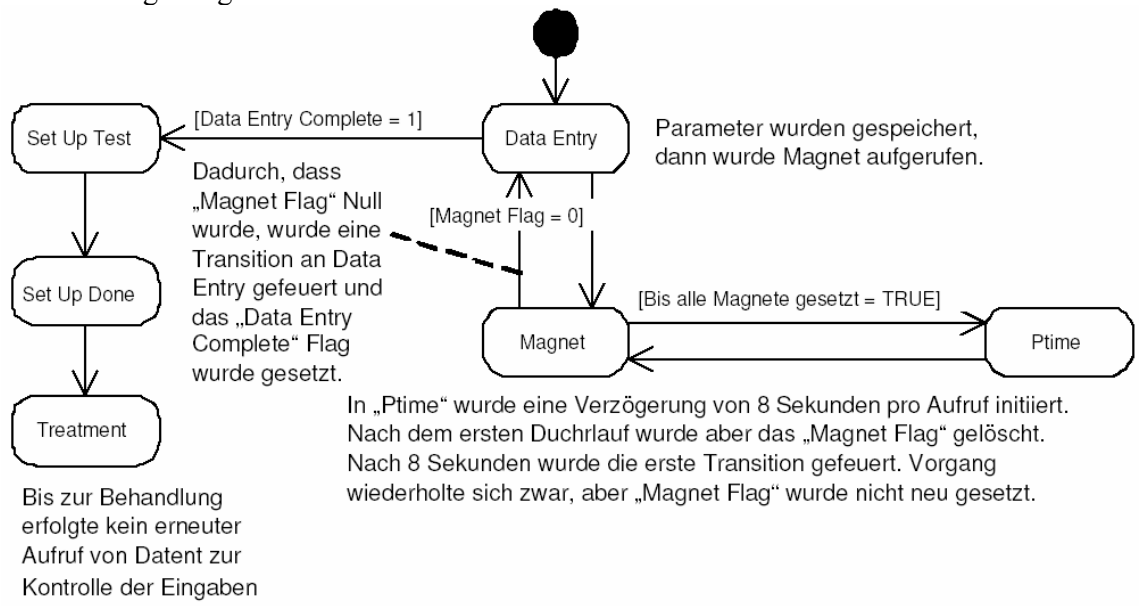
Für die Überdosis der sechs Fälle waren menschliches Fehlverhalten und zwei völlig verschiedene Softwarefehler verantwortlich.



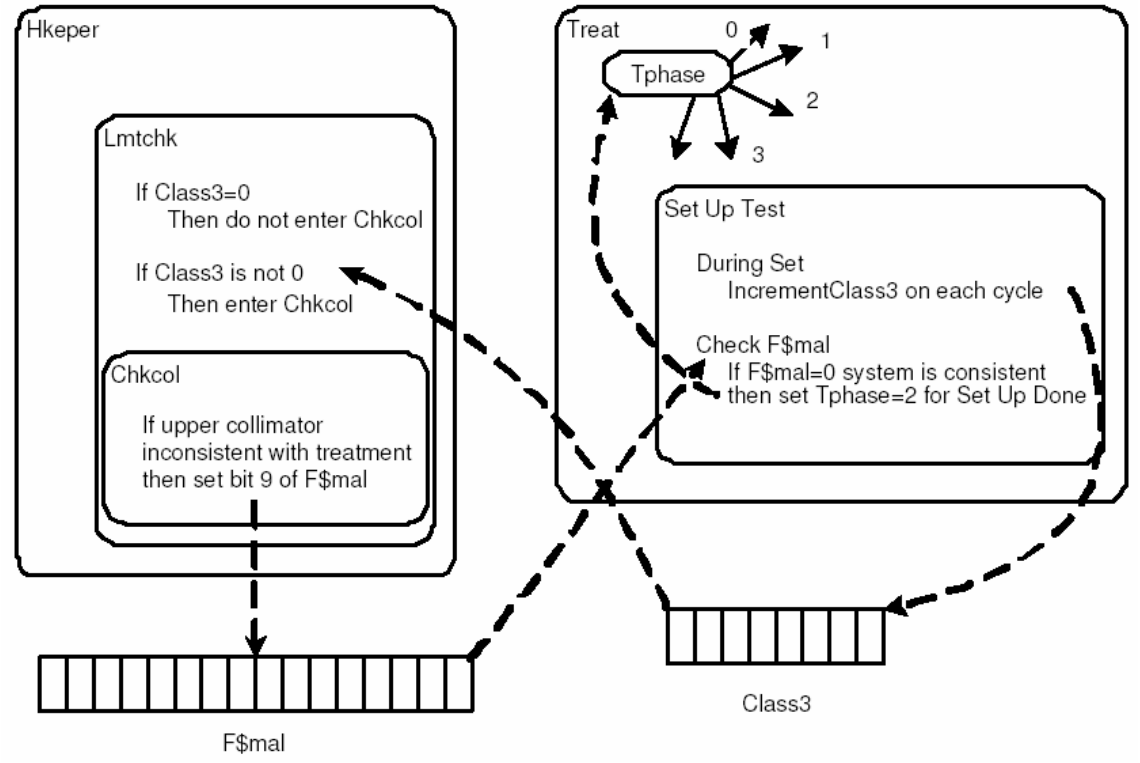
Texas-Bug

Der Texas-Bug: Ganz besonders der Software Fehler, der für die Texas Unglücke verantwortlich war, zeigt die Schwächen des Designs der Software auf. Der Behandlungsmonitor (Treat) kontrollierte die verschiedenen Phasen der Programmabarbeitung in den acht Subroutinen. Die Variable Tphase zeigte auf die Subroutine, die abgearbeitet werden sollte.
Eine Subroutine, Datent (data entry), kommunizierte zwischen dem Keyboard Handler und dem System. Dazu griff Datent auf gemeinsame Variablen zu. Der Keyboard Handler erkannte, ob die Dateneingabe vollständig war und setzte dann die „Data Entry Complete" Flag und Tphase wurde von 1 (Datent) auf 3 (Set Up Test) gesetzt, die dann ausgeführt wurde. Wurde das Flag nicht gesetzt, wurde auch Tphase nicht verändert und das Programm ging wieder in die Hauptroutine.
Die Daten wurden durch einen Cursor eingegeben. Sobald der Cursor alle Eingabemöglichkeiten durchlaufen hatte und am rechten unteren Rand des Displays ankam, wurde das „Data Entry Complete" automatisch gesetzt. Das Unglück ereignete sich durch einen Eingabefehler der Technikerin, da sie die Möglichkeit hatte, die Daten unverändert zu lassen und durch Bewegen des Cursors zur nächsten Eingabemöglichkeit zu gelangen. Sie hatte nun aus Routine ein x für Röntgenstrahlung eingegeben, obwohl der Patient eine Elektronenstrahlenbehandlung bekommen sollte. Sie bemerkte den Fehler und korrigierte die Eingabe, war aber schon bis an den rechten unteren Rand des Displays gekommen, und ging dann mit dem Cursor zurück, um aus dem x ein e zu machen. Anschließend gab sie die Strahlenintensität ein.
Der Keyboard Handler speicherte die Daten über Modus und Strahlenintensität in einer gemeinsamen Variablen, der zwei Byte großen Mode/ Energy Offset Variablen (MEOS). Das niedrigere Byte wurde auch noch von „Hand" verwendet, um die richtige Stellung des Drehtellers zu setzen. Wenn aber jetzt die „Data Entry Complete" durch den Keyboard Handler gesetzt war, wurden Veränderungen nicht mehr erkannt.
Für den Mode/ Energy Offset waren zwei nebenläufige Tasks zuständig, der nicht-kritische Keyboard Handler und der kritische „Task treatment processor". Das Einstellen der Ablenkmagneten dauerte acht Sekunden. Daher wurde ein Flag gesetzt, dass anzeigte, dass sich die Magnete positionierten und eine Unterfunktion wurde initialisiert, die eine zeitliche Verzögerung bewirkte. Diese Unterfunktion konnte mehrmals aufgerufen werden, bis alle Magnete positioniert waren. Ferner sollte, solange das Flag gesetzt war, überprüft werden, ob die Daten verändert wurden. Diese Flag wurde aber bei dem ersten Durchlauf der zeitverzögernden Unterfunktion gelöscht und somit konnten keine Veränderungen mehr erkannt werden, die nach

acht Sekunden, nach Erreichen des Cursors am rechten unteren Rand des Displays, vorgenommen wurden. Die Veränderungen wurden also im System gespeichert, aber sie wurden beim Einstellen des Modus und der Intensität nicht mehr beachtet. Dies führte zu einer ca. 100fachen Überdosis.
Der Texas-Bug wäre sehr einfach zu beheben gewesen. Man hätte das Flag nicht innerhalb der zeitverzögernden Unterfunktion löschen dürfen, somit wären dann zu jedem Zeitpunkt Veränderungen registriert worden.



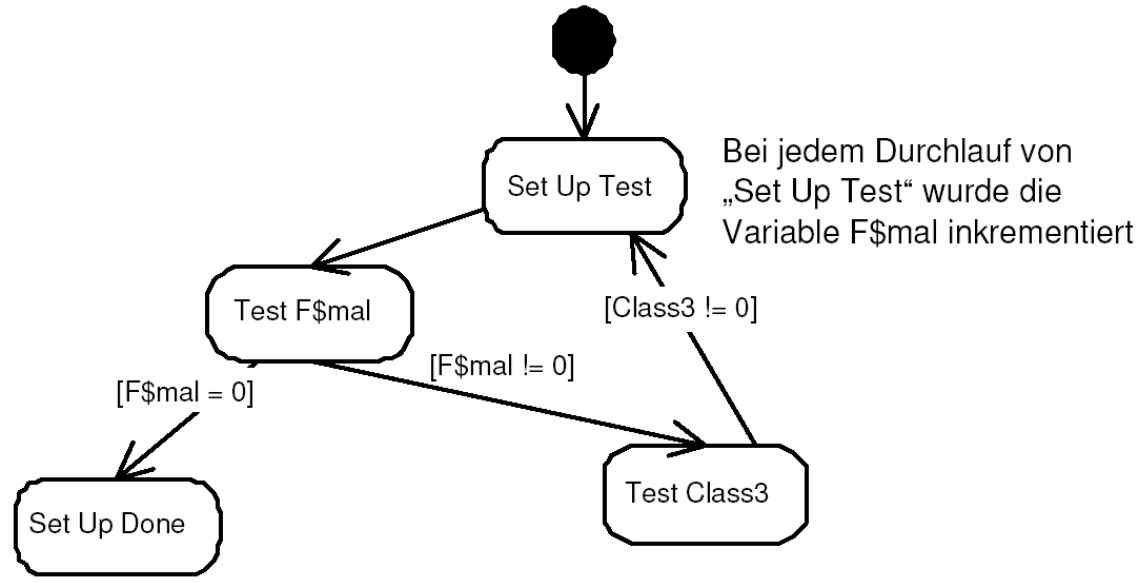
Darstellung der Subroutine Data Entry



Washington Bug

Der Washington-Bug: Es gab einen zweiten Bug, der für den zweiten Unfall in Washington und wahrscheinlich auch für den Unfall in Ontario verantwortlich war.

Normalerweise lief eine Behandlung so ab, dass der Patient auf dem Behandlungstisch platziert wurde, der Operator verließ daraufhin den Raum und stellte alle Parameter außerhalb des Behandlungsraumes am Terminal ein. Das Therac-25 besaß aber die Möglichkeit, den Patienten anhand eines Positionierungslichtes ganz genau zu platzieren. Zu dieser Feineinstellung befand sich der Operator im Behandlungsraum und stellte die Position direkt am Gerät ein. Dadurch veränderte sich der Status der Parameter von Unverified zu Verified. Anschließend gab es die Meldung Press Set Button und der Bestrahlungsarm brachte sich in die richtige Stellung.
Nachdem die Behandlungsparameter eingegeben wurden und durch die Datent Routine verifiziert wurden, wurde die Set-Up-Test-Routine aufgerufen. Bei jedem Durchlauf der Set-Up-Test-Routine wurde eine gemeinsame Variable Class 3 inkrementiert. Diese Variable gab an, ob die Stellung des Gerätes verifiziert wurde. Wenn diese Class 3 ungleich null war, lag eine Inkonsistenz vor und die Behandlung wurde nicht fortgesetzt. War die Variable null, dann bedeutete dies, dass die Parameter mit denen der Behandlung übereinstimmten und die Behandlung konnte begonnen werden. Falls die Class 3 Variable ungleich null war, führte Set-Up-Test weitere Kontrollen des Systems durch Abfragen einer anderen gemeinsamen Variablen durch: F$mal. Wenn diese Variable ungleich null war, gab es eine Fehlfunktion im System, und das Set-Up-Test wurde erneut aufgerufen. Bei null wurde die Subroutine Set-Up-Test beendet und Tphase auf 2 gesetzt (Set-Up-Done-Subroutine) und die Behandlung konnte fortgesetzt werden.
Die Set-Up-Test-Routine konnte aber mehrere hunderte Male aufgerufen werden, bis das Gerät bereit war. Das Problem dabei war, dass bei jedem Aufruf die Variable Class 3 inkrementiert wurde, die aber nur ein Byte groß war. Das bedeutet, dass die Variable maximal den Wert 255 annehmen konnte, was wiederum dazu führte, dass bei jedem 256 Aufruf von Set Up Test Class 3 gleich null war, obwohl F$mal noch Fehlfunktionen signalisierte. Dadurch wurde aber nicht mehr überprüft, in welcher Stellung sich der Drehteller bzw. der Strahlungsarm befand, der sich aber noch in der Stellung für das Positionieren des Patienten befand. Die Software initiierte dann die ganzen 25 MeV, obwohl der Drehteller nicht in der richtigen Position war. Ein hoch konzentrierter Elektronenstrahl war die Folge.
Die Lösung dieses Bugs wäre nicht schwieriger gewesen als die Lösung des Texas-Bugs. Man hätte der Variable Class 3 bei jedem Aufruf von Set-Up-Test nur einen festen Wert ungleich null zuweisen müssen ohne Inkrementierung, so dass Inkonsistenz hätte signalisiert werden können, ohne einen Overflow zu bekommen.



Set Up Test

1.2.2.4 Ablauf einer Behandlung und Bedienung des Gerätes

Ein typischer Ablauf einer Behandlung begann damit, dass der zu behandelnde Patient auf dem Behandlungstisch Platz nahm und der Operator ihn positionierte. Anschließend wurden vom Operator alle notwendigen Parameter eingestellt; in erster Linie der Behandlungsmodus (Elektronen- oder Röntgenmodus), die Drehung des Armes und das Bestrahlungsfeld. Beim Röntgenmodus wurde für die Intensität der Bestrahlung ein Default-Wert von 25MeV verwendet, der nicht geändert werden musste; wenn der Operator aber den Elektronenmodus einstellte, musste er manuell die Intensität einstellen (in der Regel ca. 200rad).
Anschließend wurde der Drehteller positioniert, bei Bedarf die Platte zwischen Patient und Gerät gebracht und nach Belieben weitere Filter eingefügt. Danach wurden die Magneten eingestellt.
Daraufhin verlies der Operator den Behandlungsraum und gab am Terminal die Patientenspezifischen Daten (einschließlich Behandlungsmodus, Energieniveau, Dosis und Zeit), das Bestrahlungsfeld, die Drehung des Armes und den Behandlungsmodus ein. Um Daten, die sich zwischen Behandlungen nicht verändert hatten, beizubehalten, brauchte der Techniker nur Return auf der Tastatur zu drücken
Das Gerät (die Software) überprüft anschließend, ob die eingestellten Werte mit den tatsächlichen des Gerätes übereinstimmen. Ist dies der Fall, wechselt der Status (der Werte) auf dem Display von *UNVERFIED* zu *VERIFIED*. Nun konnte die Behandlung beginnen.

1.2.2.5 Überwachung des Patienten

Der Patient konnte während der Behandlung auf 2 Arten beobachtet werden. Durch ein Audio-System, welches akustische Signale aus dem Behandlungsraum an den Operator-Raum sendet, und durch ein Video-System, so dass der Patient jederzeit auf einem Monitor beobachtet werden konnte. Außerdem konnte der Operator durch die beiden Systeme direkt mit dem Patienten kommunizieren.

1.2.2.6 Verfahren bei Komplikationen

Sollte es während der Behandlung zu Fehlfunktionen des Geräts kommen, so gab es verschiede Unterbrechungsverfahren. Beim ersten Auftreten eines Fehlers erschien auf dem Display des Operators eine Fehlermeldung („Treatment Pause“). Durch Drücken der Taste „P“ konnte die Behandlung aber in der Regel fortgesetzt werden, ohne am Gerät etwas zu ändern. Nach fünfmaligen Drücken der P-Taste schaltete das Gerät schließlich in den „Treatment Suspend“-Modus, der die Behandlung unterbrach und einen Neustart der Behandlung erzwang.

1.2.2.7 Fehlermeldungen

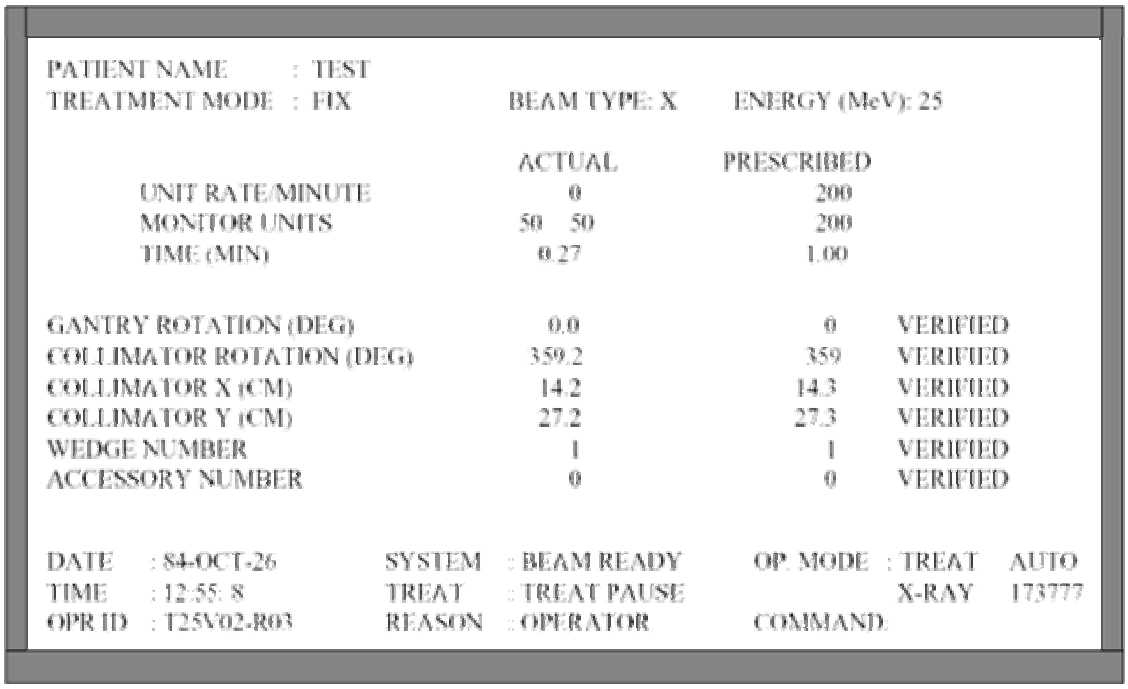
Bei Fehlfunktionen oder Inkonsistenzen des Gerätes erschienen auf dem Display der Operators Fehlermeldungen. Leider legten die Systementwickler keinen großen Wert auf Fehlermeldungen, so dass keine vernünftige Fehleranalyse durchgeführt werden konnte. Insbesondere waren diese Fehlermeldungen sehr kryptisch (z.B. „Malfunction 54“) und an keiner Stelle dokumentiert, so dass man nur wusste, dass ein Fehler auftrat, nicht aber warum. Eine andere Schwachstelle war, dass Fehlermeldungen (ohne erkennbaren Grund) sehr häufig auftraten. Durch ein anschließendes Drücken der P-Taste konnte die Behandlung ohne Schwierigkeiten fortgeführt werden.
Da Fehler sehr häufig auftraten und es auch selten eine erkennbare Ursache gab, waren Operatoren sehr schnell an diese Fehlermeldungen gewöhnt und schenkten diesen keine besondere Aufmerksamkeit mehr.

Nicht selten kam es zu über 50 Fehlern pro Tag. Das fast unermessliche Vertrauen in die eingebauten Sicherheitsmechanismen (die aber fast ausnahmslos softwareseitig implementiert waren), trug ebenfalls dazu bei, Fehlern keine ausreichende Beachtung zu schenken. So war eine Überbestrahlung aus Sicht der Operatoren und des Herstellers ausgeschlossen.

1.2.2.8 Häufigste Fehler der Therac-25

Der häufigste Fehler lag darin, das bei einer Korrektur am Terminal nur die anzeige am Terminal geändert wurde, aber die Werte wurden nicht übernommen.

Bildschirmterminal



Darstellung des Benutzer-Interfaces

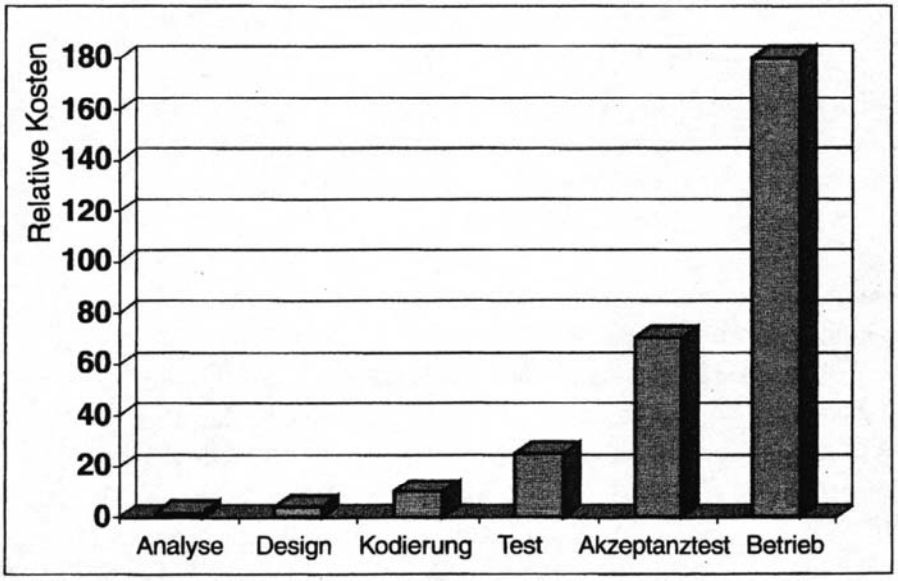
2 Wie kann man Software sicherer machen?

2.1 Wann ist eine Verifikation sinnvoll?

2.1.1 Frühe Verifikation

Am besten ist es wenn man schon im Anfangsstadion der Entwicklung eines neuen Softwareprodukts mit der Verifikation beginnt. Denn umso später ein Fehler erkannt wird, umso teurer wird es diesen zu beheben.

Abb. 2-13 S. 33



2.1.2 Verifikation ohne Computer

Es ist sinnvoll vor beginn der Programmierung in das Design der Software zeit zu investieren. Am ende jeder Phase sollte immer eine Verifikation geplant sein, hierbei kann es sich z.B. von einer Spezifikation, einen Schnittstellen-Dokument, einen Software-Entwurf oder um einen Programmcode handeln.

2.2 Welche Möglichkeiten der Fehlersuche gibt es?

2.2.1 Fagan Inspection

Hier handelt es sich um eine Revision eines Softwareprodukts, die so abläuft:

- Eine Gruppe von Kollegen in der Regel nicht mehr als vier oder fünf, studieren unabhängig voneinander das Softwareprodukt.
- Die untersuchten Produkte sind relativ klein, bilden aber eine abgeschlossene Arbeit.
- Die Prüfer verwenden jeweils ein bis vier Stunden um das Produkt zu untersuchen.
- Die Fagan Inspection ist in sechs Phasen eingeteilt:

  1.1.1. Planungsphase:

  Ist ein Software-Produkt fertig, kann eine Fagan Inspection beginnen. Als erstes wird ein Moderator benötigt, welcher für die erfüllte Eingangsvorhaussetzung und

für die Durchführung sorgt. Die Eingansvoraussetzung besteht darin, dass sich der Code einwandfrei Kompilieren lässt, alle Mitglieder ihre Rolle zugewiesen wurde, Verteilung einer Kopie des Produkts und gegebenenfalls einer Dokumentation.

1.1.2. Einführung zur Fagan Inspection:
Diese Einführung dient dazu, dass Mitarbeiter die noch keine Erfahrung mit der Fagan Inspection gemacht haben eine Einführung der Methoden erhalten. Falls alle ausreichend mit der Methode vertraut sind, kann die Einführung auch entfallen.

1.1.3. Vorbereitende Arbeiten:
Die Inspektoren bereiten sich individuell auf die Sitzung vor, dies dauert im Normalfall nicht länger als eineinhalb Stunden. Wenn ein Mitarbeiter zum ersten mal diese Inspektion durchführt kann dieser mehr Zeit benötigen. Zur Aufdeckung häufiger Fehler, kann auch ein Fragebogen eingesetzt werden.

1.1.4. Gemeinsame Sitzung:
Hier treffen sich die Inspektoren, der Moderator und der Autor (Programmierer) untersuchen das Software-Produkt gemeinsam. Vor dem Beginn überprüft der Moderator, ob sich die Inspektoren auf die Sitzung vorbereitet haben. Während der Sitzung muss eine Person das Produkt vorstellen, wie das geschieht ist dieser Person überlasen, allerdings muss das Produkt vollständig abgedeckt werden. Alle anderen suchen während der Vorstellung Fehler, der Autor selbst greift höchstens ein, wenn ein bestimmter Punkt nicht verstanden wurde, seine Anwesenheit ist aber nicht notwendig. Während der Sitzung werden die Fehler nur dokumentiert und nach ihrer Schwere klassifiziert, es werden keine Korrekturen vorgeschlagen. Wegen der intensiven geistigen Anstrengung dauert eine Sitzung nicht länger als zwei Stunden. Am Ende wird entschieden ob das Produkt akzeptiert wird, ob der Autor die gefundenen Fehler beseitigen wird. Die Beseitigung der Fehler wird nur vom Moderator überwacht, welcher auch für das weitere vorgehen verantwortlich ist.

1.1.5. Nacharbeit:
Der Autor beseitigt alle Fehler, die in der Sitzung identifiziert und dokumentiert wurden.

1.1.6. Überprüfung:
Der Moderator prüft, ob der Autor alle Fehler beseitigt hat. Falls der Moderator mit der Nacharbeit einverstanden ist, wird das Produkt in der Regel als fertig betrachtet. Alternativ dazu kann eine zweite Inspektion durchgeführt werden.

Mit der Fagan Inspection kann ein erheblicher Teil der Fehler in einem Software-Produkt gefunden werden, allerdings werden nicht alle gefunden, womit weitere Fehlersuchmaßnahmen notwendig sind.

2.2.2 Walkthroughs

Diese Methode ähnelt in vielerlei Hinsicht der Fagan Inspection.
Die Teilnehmer beim Walkthrough sind die folgenden:

- Der Autor oder Programmierer
  Der Autor muss hier unbedingt anwesend sein.
- Der Tester
  Der Tester muss sich intensiv mit dem Quellcode befassen.
- Ein Sekretär oder Schreiber
  Der Schreiber muss gefundene Fehler notieren, und übergibt nach der Sitzung die Liste dem Programmierer.
- Der Moderator

Der Moderator spielt seine Rolle wie bei der Fagan Inspection. Er bereitet das Treffen vor, benachrichtigt die Teilnehmer, verteilt das notwendige Material und sorgt für einen ruhig gelegenen Raum.

Ein entscheidender unterschied zur Fagan Inspection ist, dass die Teilnehmer die Aufgabe haben, vor der Sitzung kleine überschaubare Testfälle zu kreieren. Diese Testfälle werden dann Schrittweise durchgegangen, allerdings nicht mit Hilfe eines Computers, sondern sie werden von Hand durchgespielt. Mit den Testfällen kann man zwar nicht alle Möglichkeiten abdecken, aber sie dienen dazu das Team in Fahrt zu bringen und die Annahmen des Programmierers und die Logik des Quellcodes zu hinterfragen. In vielen Fällen findet der Programmierer, durch die Fragen seiner Kollegen, oft selbst einen erheblichen Anteil der Fehler, weil logische Fehler und unklare Punkte im Programm aufgedeckt wurden.

Kurze fragen sind:

1. Gibt es Endlosschleifen?
2. Kann es dazu kommen, dass das Programm seine Ausführung niemals endet?
3. Gibt es genauso viele geöffnete wie geschlossene Klammern? (Compiler)
4. Kann bei einem *case*-Statement das Programm mit einem Programmteil fortfahren, in dem es nie kommen sollte?
5. Hängt eine Entscheidung von Bedingungen ab, die auf Gleichheit prüfen? (Compiler)

Fehleranfällig sind auch die Bereiche der Schnittstellen sowie Ein- und Ausgabe.

Das Team sollte aus weniges als einen halben Dutzend Experten bestehen darunter zum Beispiel:

- ein Erfahrener Programmierer,
- ein Fachmann für eine bestimmte Programmiersprache, der mit ihren Tücken vertraut ist,
- ein neu hinzugekommener Programmierer, der dieser Sache wahrscheinlich unvoreingenommen gegenübersteht,
- ein Entwickler, der das fertige Programm eines Tages warten muss.

Nach dem Walkthrough erfolgt die Beseitigung der Fehler in ähnlicher weise wie bei der Fagan Inspection. Der Entwickler arbeitet die Änderungen ein und der Moderator ist dafür verantwortlich, dies zu überprüfen. Es besteht auch die Möglichkeit ein zweites Walkthrough anzusetzen.

2.2.3 Modultest als White Box Test

Bei einem White Box Test ist der Quellcode bekannt, somit hat man die Möglichkeit einzelne Module davon genauer zu betrachten und nach ihrer Richtigkeit zu prüfen. Dazu ist es sinnvoll ein Testprogramm zu erstellen, womit man die Richtigkeit der ausgaben für einzelne eingaben überprüfen kann. Hier gibt es zwei Möglichkeiten:

1. Man lasst sich die Ergebnisse ausgeben und überprüft diese, in dem man diese von Hand nachrechnet oder nachschlägt. Der Nachteil hierbei ist, dass man bei einer Wiederholung des Test, zum Beispiel bei einem Fehlerfall, alle Ergebnisse wieder kontrollieren muss.
2. Es ist auch möglich die Ergebnisse von dem Testprogramm selbst überprüfen zu lassen. Allerdings müssen die Ergebnisse dem Testprogramm erst mitgeteilt werden, indem man sie einprogrammiert oder von einem anderen Programm berechnen lässt, hier ist es aber notwendig, dass dieses auch korrekt arbeitet. Der Vorteil dieser Methode ist, dass man dem Test immer wieder wiederholen kann und das ohne großen Aufwand.

Der Vorteil eines White Box Test ist, dass man das Programm gezielt überprüfen kann. Zum Beispiel an Stellen wo eine Verzweigung stattfindet und an anderen Randstellen. Hier hat

man auch die Möglichkeit, dass man alle Pfade testet, was auch notwendig ist. Denn bei einen White Box Test sollte man alle Pfade mindestens einmal durchlaufen haben.
Nachteilig ist, dass uns der Code bekannt ist und somit manche Eingaben nicht getestet werden, weil man der Meinung ist, dass diese sowieso richtig bearbeitet werden. Es ist aber leider so, dass man manche Reaktionen des Programms nicht richtig interpretiert, da man als Programmierer die geplante Funktionsweise des Programms kennt, kommt man nicht auf die Idee das bei bestimmten Eingaben ein Fehler auftritt. Da es auch manchmal so ist, dass ein Programm nicht wie geplant reagiert, so wie man es als Programmierer annimmt. Solche Fehler sind mit Hilfe des White Box Test nur schwer zu finden, falls diese nur bei vereinzelten Eingaben auftreten.

2.2.3.1 Testabdeckung

Da der Programmcode offen vor uns liegt, können wir den Kontrollfluss des Programms verfolgen. Wir sehen die Daten und die Auswirkung beim setzen gewisser Werte. Wir verstehen auch die Logik des Programms und können uns vorstellen, wie sich das Programm in einer bestimmten Situation verhält. Das können wir ausnutzen und gezielte Tests durchführen.
Maß der Testabdeckung:

C0 Alle Anweisungen in einem Modul oder Programm werden ausgeführt.

C1 Alle Segmente eines Moduls oder Programms werden mindestens einmal ausgeführt. Es werden also alle Pfade durch das Programm wenigstens einmal durchlaufen.

C1+ Alle Programmsegmente werden wie bei C1 mindestens einmal ausgeführt. Zusätzlich wird bei Schleifen mit den Extremwerten getestet.

C1p Alle Programmsegmente werden mindestens einmal ausgeführt. Darüber hinaus wird bei jedem logischen Ausdruck so getestet, dass der Code bei jeder logischen Bedingung zumindest einmal ausgeführt wird.

C2 Alle Programmsegmente werden wenigstens einmal ausgeführt. Bei Schleifen muss so getestet werden, dass die Schleife 1. nicht ausgeführt wird, 2. mit einem niedrigen Wert des Schleifenzählers getestet wird und 3. mit einem hohen Wert des Schleifenzählers getestet wird.

Cik Alle Programmsegmente werden mindestens einmal ausgeführt. Darüber hinaus werden alle Schleifen in dem Modul oder Programm für die Werte i bis k ausgeführt, wobei i = 1,2,3 ... k.

Ct Die Kombination aller möglichen Pfade wird durch White Box Test abgedeckt.

Heutzutage wird das Maß C0 als nicht ausreichen betrachtet. Hundert Prozent Testabdeckung nach C1 ist der heutige Standard und das Kriterium C2 ist in vielen Fällen erreichbar.

2.2.3.2 Incremental Testing

Nachdem die einzelnen Module ausgetestet worden sind, gibt es die Möglichkeiten:

- alle Aufeinmahl miteinander zu verbinden und zu Testen oder
- eines nach dem anderen anzufügen und zu Testen.

2.2.3.3 Schwächen des White Box Tests

Mit diesem Test kann man einige Fehler finden, aber er hat auch nachteile:

1. Intensives Überprüfen eines Programms mittels White Box Test beweist nicht, dass das Programm mit seiner Spezifikation übereinstimmt.
2. Trotz White Box Test kann das Programm oder das Modul einige spezifizierte Funktionen nicht enthalten.
3. Beim White Box Test werden Fehler in den Daten häufig nicht aufgedeckt (z. B. < für <=)

2.2.4 Black Box Test

Hier wird der Test nicht vom Programmierer durchgeführt sondern von einer Gruppe, die den Quellcode nicht kennt. Diese Gruppe beurteilt den Programmcode lediglich nach seiner Funktion. Die Tester orientieren sich an dem Lastenheft oder an dem Benutzerhandbuch, sofern dieses existiert.
Die Aufgabe der Tester besteht darin die Fehler in einem Programm zu finden. Da Fehler nicht gleichverteilt sind, sondern sich auf bestimmte bereiche, wie komplizierte Module konzentrieren, testen die Tester diese bereiche genauer. Sie müssen auch überprüfen ob das Programm gemäß den Spezifikationen arbeitet.
Der Vorteil einen externen Testgruppe ist, das diese gewillt sind Fehler zu finden, im Gegensatz zu einem Programmierer, welcher bevorzugt wenn ein Programm „fehlerfrei“ ist.

3 Wie kann man die Anzahl der Fehler bestimmen?

3.1 Wie bekommt man die zu erwartende Zahl der Fehler?

Bei jedem Software-Projekt fragt man sich natürlich, wie hoch die zu erwartende Zahl der Fehler sein könnte. Wenn man keine Metriken aus dem eigenen Unternehmen besitzt, sind Zahlen aus der Fachliteratur als erste Annäherung ganz nützlich.

3.2 Gleichungen zur Bestimmung der Fehleranzahl

Gleichungen zur Bestimmung der Fehleranzahl:

- Basil gibt für die Zahl der zu erwartenden Fehler die folgende Gleichung an:

  $F = 4{,}04 + 0{,}0014\ LOC^{4/3}$

  wobei

  F: Erwartete Fehleranzahl

  LOC: Programmlänge in Lines of Code

  Für diese Formel gilt die folgende Einschränkung:

  $340 < LOC < 12\ 541$
- Gleichung von zwei japanischen Praktikern:

  $F_r = C_1 + C_2\ (SCHG/KLOC) - C_3\ ISKL - C_4\ (DOCC/KLOC)$

  wobei

  F_r: Fehlerrate pro 1000 Lines of Code (LOC)

  C_1: Konstante 67,98

  C_2: Konstante 0,46

  C_3: Konstante 9,69

  C_4: Konstante 0,08

  SCHG: Änderung am Lastenheft während der Entwicklung

  KLOC: Codeumfang in 1000 LOC

  ISKL: durchschnittliche Erfahrung des Entwicklungsteams mit einer Programmiersprache (in Jahren)

  DOOC: Ausgereiftheit des Entwurfs, d. h. Anzahl der geänderten oder neu erstellten Seiten in Designdokumentation

4 Kriterien für das Ende der Tests

4.1 Error Seeding

Hier benötigt man eine dritte Gruppe von Entwicklern oder erfahrene Testern, welche von der Entwicklergruppe der Software sowie von der Testgruppe organisatorisch unabhängig ist. Die Aufgabe dieser Gruppe besteht darin, zusätzliche Fehler in die Software einzubauen, die nur sie kennen. Erst nach diesem Schritt wird die zu testende Software an die Testgruppe weitergeleitet.

Dem Verfahren liegt die Annahme zu Grunde, dass eine gute Testgruppe beide Arten von Fehlern finden wird, die von den Entwicklern unbewusst gemachten Fehler und die von der dritten Gruppe absichtlich in den Programmcode eingefügten Fehler. Testende ist folglich dann, wenn alle zusätzlichen Fehler von der Testgruppe gefunden wurden.
Den Erfolg der Testgruppe kann man seitens des Managements daran messen, wie viele der eingebauten Fehler sie bereits gefunden hat. In mathematischer Notation lässt sich das Verfahren wie folgt ausdrücken:

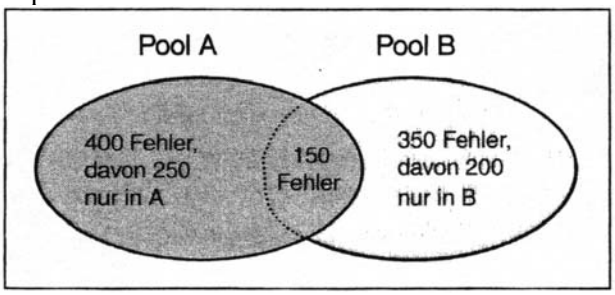
$$Defects_{total} = (Defects_{seeded}/Defects_{seeded + found}) \times Defects_{found}$$

wobei

$Defects_{totoal}$:	Gesamtzahl der erwarteten fehler
$Defects_{seeded}$:	Zahl der künstlich hinzugefügten Fehler
$Defects_{seeded + found}$:	Zahl der hinzugefügten Fehler, die bereits gefunden wurden
$Defects_{found}$:	Gesamtzahl der bisher gefundenen Fehler.

4.2 Zwei Testgruppen

Hier setzt man zwei voneinander unabhängige Testgruppen ein, die beide denselben Code testen. Aus der Zahl der von beiden Gruppen gefundenen Fehler im Programm kann man Schlüsse auf die verbleibenden Fehler ziehen.
Bsp: *Abb. 5-7 S. 168*



In diesem Bild sind 933 Fehler zu erwarten
Formel für Gefundene fehler:

$$Defects_{unique} = Defects_A + Defects_B - Defects_{A+B}$$

wobei

$Defects_{unique}$:	Zahl der einmaligen Fehler
$Defects_A$:	Zahl der Fehler in Pool A
$Defects_B$:	Zahl der Fehler in Pool B
$Defects_{A+B}$:	Zahl der Fehler in Pool A und Pool B

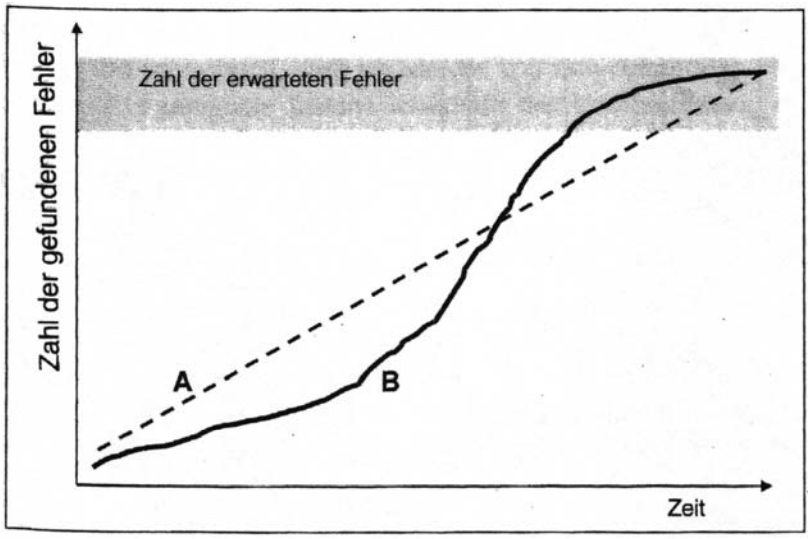
Zahl der Insgesamt zu erwartenden Fehler:

$$Defects_{totoal} = (Defects_A \times Defexts_B)/Defects_{A+B}$$

4.3 Graphische Darstellung

Da die beiden vorherigen Lösungen nicht in kleinere Firmen eingesetzt werden können bietet sich hier eine Graphische Lösung an. Hier trägt man die Gefundenen Fehler pro Zeiteinheit auf, etwa Fehler pro Woche oder bei großen Projekten Fehler pro Arbeitstag. Als Alternative bietet es sich an, die kumulierte Zahl der Fehler über eine Zeitachse aufzutragen. Mit der letzten Darstellung ergibt sich das Folgende Bild

Abb. 5-8 S. 169



Theoretisch ergibt sich für den Anstieg der Fehler über der Zeit mit ‚A' markierte Kurve. In der Praxis zeigt sich allerdings häufig, dass am Anfang der Testphase weniger Fehler gefunden werden als zunächst erwartet. Das liegt oft daran, dass die Entwickler einen Vorsprung an Know-how gegenüber der Testgruppe besitzen. Haben sich die Tester allerdings erst richtig mit der Materie vertraut gemacht, finden sie umso mehr Fehler. Die Kurve steigt dann steil an und zeigt einen Verlauf wie unter ‚B' gezeichnet. Auch bei Fehlern in der Software ist es in der Regel so, dass die schweren Fehler erst gefunden werden, wenn die Testgruppe so richtig warm geworden ist. Das ist auf der anderen Seite nicht verwunderlich, denn wären die Fehler offensichtlich hätten diese die Entwickler schon gefunden.

4.4 Wann kann ein Programm freigegeben werden?

Hier ist es sinnvoll schon am Anfang des Projekts die Freigabebedingungen festzulegen. Die Bedingungen könnten wie folgt aussehen:

1. Ist auch nur ein Fehler der Fehlerklasse I bekannt, stellt dies ein Hindernis für die Freigabe dar. Software mit einem Fehler (oder Mehreren) der Klasse I wird generell nicht an Kunden ausgeliefert. Fehler der Klasse I sind ein Kriterium für die Sperre der Software.
2. Wenn die Fehler der klasse II in großen Mengen auftreten, so dass bezweifelt werden muss, dass Kunden und Anwender mit der Software so arbeiten können, wie sie das erwarten, dann ist auch bei Fehlern der Klasse II eine Sperre möglich. Die Entscheidung darüber liegt beim Qualitätsmanagement.
3. Fehler der Klasse III stellen kein Hindernis für die Freigabe dar. Sie sind schnellstmöglichst zu beseitigen.

5 Literaturverzeichnis

- Georg Erwin Thaller: Software-Test – Verifikation und Validation.
  - aus 2: Prozessbegleiten, Fagan inspections, Code walkthroughs
  - aus 3, 4: White Box Test, Black Box Test
  - aus 5: Metriken
  - aus 6: sicherheitskritische Software …, Qualitätsverbesserung: DIN EN ISO
- A. Gemmer: Risk Management: Moving bayond process. IEEE Computer 30(5):33-43, 1997.
- N. G. Leveson, C. S. Thurner: An investigation of the Therac-25 accidents. IEEE Computer 26(7):18-41, 1993.
- Vorlesung Sicherheit: Therac-25 Prof. Dr. Lutz Prechelt Freie Universität Berlin, Institut für Informatik http://www.inf.fu-berlin.de/inst/ag-se/
  aufgerufen am 25.11.2004:
  http://www.inf.fu-berlin.de/inst/ag-se/teaching/V-AWS-2004/22_Sicherheit.pdf
- Henning Herbers & Tamer Hasan - Der Therac Unfall
  aufgerufen am 25.11.2004:
  http://www4.informatik.tu-muenchen.de/lehre/seminare/ps/WS0203/desaster/Herbers-Hasan-Therac-Ausarbeitung-18-11-02.pdf
- Hauptseminar Prof. Huckle - Therac-25
  aufgerufen am 25.11.2004:
  http://www5.in.tum.de/lehre/seminare/semsoft/unterlagen_02/therac/website/#sonstiges
- Martin Pfeifer Universität Koblenz - Seminar Dr. Beckert Berühmt berüchtigte Softwarefehler Therac-25
  aufgerufen am 25.11.2004:
  http://www.uni-koblenz.de/~beckert/Lehre/Seminar-Softwarefehler/Ausarbeitungen/pfeifer.pdf