Softwarefehler

Informatik und Gesellschaft
Universität Zürich FS2011

Autoren: Jonas Hofer, Robin Engbersen, Simon Kaeser

Inhalt

Einleitung

Auf Softwarefehler trifft man oft im täglichen Leben. Computer stürzen ab, Programme verabschieden sich mit oder ohne Fehlermeldungen und Mobiltelefone wecken uns zu falschen Zeitpunkten. Doch es gibt nicht nur solche, in den meisten Fällen relativ unbedeutende Softwarefehler wie diese. Die Geschichte zeigt uns diverse Beispiele, welch verheerende Folgen Fehler in Computerprogrammen, sogenannte Bugs, haben können. Diese Arbeit fasst in kurzem Rahmen die Arten und die Entstehung von Softwarefehlern zusammen, illustriert die möglichen Folgen von Softwarefehlern anhand von drei berühmten Beispielen und zeigt Möglichkeiten zur Vorbeugung von Bugs auf.

Arten und Entstehung von Softwarefehlern

Softwarefehler, umgangssprachlich auch *Bugs* genannt treten aufgrund von menschlichem Versagen auf. Sie bezeichnen ein fehlerhaftes Verhalten in Computersystemen, was meistens zu unerwarteten Ergebnissen oder sogar zum Absturz des gesamten Systems führt. Softwarefehler können auch bei erfolgreicher Kompilierung des Quelltextes auftreten. (DATACOM GmbH, B. (Hrsg.), 2006) Das Wort Bug stammt aus dem 19. Jahrhundert. Damals wurde irrtümlicherweise das Knistern und Rauschen in der Telefonleitung damit erklärt, dass kleine Tiere, sogenannte *Bugs* die Leitung anknabbern. Im Jahre 1945 wurde der erste tatsächliche Fund einer Motte in einem Relais eines Computers dokumentiert. Diese Motte führte zu einer Fehlfunktion. Aus diesem Zwischenfall resultiert vermutlich die heutige Verwendung des Wortes Bug in Zusammenhang mit Computer. (Wikipedia.org, 2011)

Es gibt viele Möglichkeiten, Softwarefehler in Gruppen einzuteilen. Die gängigste Einteilung unterscheidet Logikfehler und Laufzeitfehler. Logikfehler charakterisieren sich durch falsche Ansätze in der Problemlösung für eine Software. Es kann sich aber auch um eine Fehlinterpretation eines Algorithmus handeln, der dann in die Software mit eingebaut wird und aufgrund falscher Berechnungen falsche Ergebnisse liefert.

Laufzeitfehler treten erst bei der Ausführung der Software auf. In den meisten Fällen liegt die Ursache von Laufzeitfehlern an der Programmlogik. Aber

auch an Komptabilitätsproblemen von Software mit fremden Hardware- oder Softwareumgebungen. (Wikipedia.org, 2011)

Folgen von Softwarefehlern

Folgen von Softwarefehler können in ausgewählten Fällen extremen Schaden anrichten. Daher ist es bei der Entwicklung wichtig, das Risiko richtig einzuschätzen, damit für unvorhergesehene Softwarefehler bereits im Vorfeld präventive Massnahmen ergriffen werden können. So können Auswirkungen der Fehler in der Software bis zu einem gewissen Grad kontrolliert werden.

Ariane 5

Vorgeschichte

Die Ariane Raketen werden von der European Space Agency (ESA) gebaut und genutzt. Die erste Ariane Rakete, die Ariane 1 wurde an Heiligabend 1979 in den Orbit geschossen. Es folgten weitere Modelltypen, wie die Ariane 2 und Ariane 3. Beide hatten grundsätzlich das Ziel, einen – für Europa – unabhängigen Zutritt zum Weltall zu verschaffen. Als der Hauptkonkurrent NASA das Spaceshuttle weiterentwickelte und die Nutzlast erhöhte, blieben die Ariane Raketen auf der Strecke. Damit die ESA auf dem internationalen Weltraummarkt bestehen und konkurrenzfähig bleiben konnte, mussten sie ebenfalls neue, bessere Raketen entwickeln. Dadurch entstanden die Ariane 4, wie auch die Ariane 5 Raketen. Als die ESA die Ariane 5 entwickelte übernahm sie eine grosse Menge der Softwaremodule von der alten Ariane 4. Dies wurde beim Erstflug der Ariane 5 am 4. Juni 1996 zum Verhängnis. 40 Sekunden nach dem Start des Jungfernfluges der Ariane 5 änderte sie abrupt den Kurs um etwa 20 Grad. Dies führte zum Auseinanderbruch der Rakete und schlussendlich zur automatischen Selbstzerstörung. (ESA, 2004)

Ursache

Obwohl die Ariane 5 grösser, schwerer und einiges leistungsfähiger als die Ariane 4 war, übernahmen die Entwickler viele Softwareteile von der alten Ariane 4. Bei der Unfallanalyse der Ariane 5 stellte sich heraus, dass beim Konvertieren einer 64-Bit Gleitkommazahl in eine 16-Bit Ganzzahl im Lenksystem ein Fehler auftrat, der von keiner Exception aufgefangen wurde. Dies führte dazu, dass das Lenksystem dem Navigationscomputer falsche Ergebnisse lieferte. Daher kam es zu einer ungewollten 20 Grad Neigung der Rakete, die schlussendlich – aus Sicherheitsgründen – zur Selbstzerstörung der Rakete führte. Die finanziellen Auswirkungen des Absturzes beliefen sich auf 500 Mio. Euro. (Weyand, 2003)

Pentium-FDIV-Bug

Abbildung 1 - Intel Pentium 66 (P5) [Wikipedia.org, 14. März 2011]

Entdeckung

Professor Dr. Thomas Ray Niceley, welcher am Lynchburg College im Staat Virginia unterrichtete, entdeckte im Zuge seiner Forschungstätigkeiten eine Unregelmässigkeit in seinen Berechnungen. Gewisse von ihm, mit einem neuen Computer berechnete Zahlen, stimmten nicht mehr mit den alten Testresultaten überein. Während mehreren Monaten suchte Dr. Niceley nach der Ursache dieses Fehlers. Nachdem er sämtliche von ihm geschriebene Software überprüft und als Ursache ausgeschlossen hatte, konnte er den Fehler schliesslich nur noch auf die CPU zurückführen. Mehrere von ihm getätigte Untersuchungen bestätigten diese Vermutung. Am 24. Oktober 1994 meldete er sich telefonisch bei Intel und berichtete vom gefunden Fehler. Als sich Intel schliesslich nicht mehr zurückmeldete, verbreitete er seine gefundenen Erkenntnisse an weitere Forschungsinstitutionen und Industrieanlagen des Landes. Durch den dadurch entstehenden Druck von Seiten Forschung und Industrie sowie der Presse, bestätigte Intel am 28. November 1994 offiziell den Fehler in den Typen Pentium 60/66 (P5), Pentium 90/100 (P54C) und mobile Pentium 75 (P54C). Intel wertete den Fall allerdings als unwichtig ab, weil der Fehler nach eigenen Statistiken bei normaler Verwendung von Office Programmen nur zirka alle 27000 Jahre einmal auftritt. (Wikipedia.org, 2010) (Knieschewski, 2004)

Ursache

Der Bug zeigt sich darin, dass der Prozessor bei gewissen Divisions-Berechnungen mit sehr hohen Zahlen oder vielen Nachkommastellen falsche Resultate zurückliefert. Diese Gegebenheit hat seine Ursache im SRT-Divisionsalgorithmus, welcher bei einer Berechnung auf eine Matrix zurückgreift, die in der Programmable Logical Area gespeichert ist. Diese Tabelle besteht aus 1066 Zellen, von welchen 5 eine fehlerhafte Zahl beinhalten. Greift der Algorithmus nun bei einer Berechnung wie zum Beispiel 824.633.702.441 / 824.633.702.441 auf eine solche fehlerhafte Zelle zu, so resultiert ein falscher Wert (hier: 0,999999996274709702 anstatt 1). Diese fehlerhaften Zellen entstanden aus einem Fehler in der for-Schleife des C-Programmes, welches die einzelnen Werte in die Matrix auf der Programmable Logical Area des Prozessors geschrieben hatte. Anstatt einer 2 wurde kein Wert initialisiert, wodurch eine 0 in der Tabelle stehen blieb. (Jackson, 1998) (Knieschewski, 2004)

Folgen

Auf Grund des grossen Drucks der Öffentlichkeit willigte Intel schliesslich ein, sämtliche CPU auszutauschen sofern der alte, vom Fehler betroffene Prozessor an Intel zurückgesendet wird. Intel verkaufte über zwei Millionen fehlerhafte Pentium Prozessoren, wovon zirka 10% der an Industrie und Forschung ausgelieferten Prozessoren, ausgetauscht wurden und zirka 2% aus den restlichen Kundensegmenten im amerikanischen Markt. Weiter musste Intel um die zwei Millionen CPUs verschrotten, welche noch an Lager waren oder von Resellern und Produzenten zurückgesendet wurden. So entstand für Intel ein Schaden von 475 Millionen US-Dollar. (Knieschewski, 2004) (Wikipedia.org, 2010)

Patriot Missile System Failure

Am 25. Februar 1991 wurde eine irakische Scud-Rakete vom Patriot-Flugkörper-Abwehrsystem aufgrund eines Softwarefehlers nicht abgefangen und

schlug in eine Armeeunterkunft ein. Dabei wurden 28 US-Soldaten getötet, weitere 98 wurden verletzt.

Abbildung 2 – Patriot Missile System [Lum, A., 15. März 2011]

Hintergrund

Das Patriot System ist ein defensives Boden-Luft Raketensystem, das vom amerikanischen Rüstungskonzern Raytheon in Zusammenarbeit mit der US-Army entwickelt und gebaut wurde.

Das Waffenkontrollsystem hatte 3 Hauptaufgaben:

- Suche nach Objekten mit Scud-Charakter (unter anderem Geschwindigkeit, Azimut und Höhe).
- Vorausberechnung der zukünftigen Position der angreifenden Rakete durch das "Range-Gate" System im Radarcomputer, um die Flugbahn zu verfolgen.
- Abfangen einer in Reichweite befindlichen feindlichen Rakete durch eine Patriot Abwehrlenkwaffe. (Lum)

Ursache

Der Softwarefehler, der zur Zerstörung der amerikanischen Armeeunterkunft führte, tritt bei der Berechnung der zukünftigen Position der angreifenden Rakete auf.

Die Berechnung erfolgt aufgrund der Geschwindigkeit und der Zeit der letzten Radar-Erfassung. Die Geschwindigkeit wird als ganze Zahl mit Nachkommastellen gespeichert. Die Zeit hingegen als fortlaufende ganze Zahl in Zehntelsekunden, wobei der Algorithmus für die Berechnung der zukünftigen Position die beiden Inputs als reelle Zahlen benötigt.

Tabelle 1 - Abweichung Range-Gate [Lum, A., 15. März 2011]

h	s	Berechnete Zeit	Fehler [s]	Abweichung Range-Gate
0	0	0	0	0 m
1	3600	3599.9966	.0034	7 m
8	28800	8799.9725	.0025	55 m
20	72000	71999.9313	.0687	137 m
48	172800	172799.8352	.1648	330 m
72	259200	259199.7528	.2472	494 m
100	360000	359999.6667	.3433	687 m

Der Computer des Patriot-Systems hat lediglich ein 24-bit fixed-point-register. Weil die Zeit in 1/10sec gemessen wird, was eine nicht-terminierende binäre Repräsentation darstellt, wird bei jedem Hochzählvorgang die addierte Zahl nach 24bits hinter dem Dezimalpunkt abgeschnitten. Statt 0.0001100110011001100110011001100... wird nur 0.000110011001100110011001100 addiert. Der dadurch verursachte Fehler wird also mit zunehmender Einsatzzeit immer grösser und die Auswirkung des Fehlers ist direkt proportional zur Raketengeschwindigkeit. (Arnold, 1996)

Beim Design des Patriot-Systems wurden als primäre Ziele sowjetische Flugzeuge und Cruise-Missiles angenommen, die eine Geschwindigkeit von etwa MACH 2 erreichen. Damals wurde eine Einsatzdauer von nur ein paar Stunden propagiert, da das System zur erschwerten Aufspürung stets weiterverschoben werden sollte.

Im Golfkrieg wurden die Systeme als statische Verteidigungssysteme eingesetzt (Daueroperation) und mussten Scud-Raketen mit Geschwindigkeiten von MACH 5 abfangen.

Die Rakete, die am 25.Feb 1991 28 Soldaten tötete, hätte von einem Patriot-System mit einer Laufzeit von bisher 100 Stunden abgefangen werden müssen. Jedoch wächst der Zeitfehler nach 100 Stunden kontinuierlicher Benutzung auf 0.34 Sekunden. Bei einer Geschwindigkeit der Scud-Rakete von zirka 1.7km/sec ergibt dies eine Abweichung von mehr als einem halben Kilometer. Diese führte dazu, dass das Zielverfolgungsradar die Rakete an einer falschen Position erwartete und deswegen nicht abfangen konnte.. (Blair, 1992)

Vermeidung & Behebung von Softwarefehlern

Debugging

Durch Debugging können Syntaxfehler und Laufzeitfehler mit hoher Sicherheit ausgeschlossen werden. Zusätzlich übernehmen während dem Programmieren die gängigen Editoren wie Eclipse die Fehlersuche nach Syntaxfehlern automatisch. Während der Kompilierung des Programms werden auch Zirkelbezüge und Typsicherheit überprüft. Denkbar ist der Einsatz von Tools wie Findbugs, Lint oder Splint zur spezifischen Suche nach Bugs.

Testverfahren

Testverfahren müssen entwickelt und spezifiziert werden, bevor ein Programm überhaupt geschrieben wird. Damit wird erreicht, dass der Test nicht genau so geschrieben wird, dass er präzise zum geschriebenen Programm passt, sondern, dass das Programm zu den gemachten Anforderungen kompatibel ist. Die Testfälle sollten ebenfalls schon vor Beginn der Implementierungsphase bestimmt werden. Dabei wird zwischen Unit-Tests (Komponententests) und Integrationstests unterschieden.

Das Testen muss durch Annahme von für den Anwendungsfall erwarteten Werten erfolgen. Die Tests sollten immer auch Werte mit einbeziehen, die an den Grenzen der Definitionen liegen. Diese Grenzen sollten nicht zu restriktiv gesetzt werden. Im Jahre 1985 zum Beispiel wurden von einem Satelliten, der die Gaszusammensetzung der Atmosphäre analysierte, die zu tiefen Ozonwerte über der Antarktis als Messfehler verworfen. Schliesslich stellten sich diese Werte jedoch als das mittlerweile allbekannte Ozonloch heraus. (Uherek, 2004) Zusätzlich zu möglichst umfassenden Definitionsgrenzen sollte immer eine gewisse Sicherheitsmarge eingebaut werden, (vgl. Ariane) um bei Überschreitung der vorgängig gemachten Anforderungen die Software nicht unkontrolliert terminieren zu lassen. Grosses Augenmerk sollte auf Benutzereingaben gelegt werden, da Benutzer eine grosse Fehlerquelle darstellen. Illustrierend: Ein US-Soldat gab auf dem Steuerungscomputer eines US-Amerikanischen Kriegsschiffes aus Versehen eine Null ein, anstatt eines Kursparameters. Ergebnis: Der Steuerungscomputer des Schiffs stürzte aufgrund einer Division durch null ab und das Schiff musste ins Dock zurück geschleppt werden. (Wired, 1998)

Saubere Anforderungsspezifikationen

Die Anforderungen an eine Software müssen vor Beginn der Programmierarbeiten abschliessend definiert und untrennbar mit der geschriebenen Software verknüpft werden. Bei nachträglichen Änderungen an der Software muss die ursprüngliche Spezifikation unbedingt mit einbezogen werden um folgenschwere Fehler zu vermeiden. (Glinz, 2003)

Wäre dieses Vorgehen auch beim Patriot-Missile-System angewendet worden, hätte der Fehler auffallen müssen, als man die neuen Anforderungen an die Software mit den ursprünglich gemachten Annahmen verglich.

Ebenfalls auf ein solches Problem zurückzuführen ist der Ariane Flug 501, bei dem die Rakete ausser Kontrolle geriet, da Software der alten Ariane 4 wiederverwendet wurde, ohne zuvor die Anforderungen zu vergleichen.

Öffentliche Beta-Tests

Öffentliche Beta-Tests können natürlich nur in Fällen angewendet werden, wo dies möglich und zweckmässig ist. Wer hat schon einen Mars-Rover zuhause, um die dafür geschriebene Software zu testen. Hingegen ist es bei Microsoft-Anwendersoftware mittlerweile Standard, dass vor der Veröffentlichung der so genannten Release Candidates Beta-Tests durchgeführt werden, um vor allem Betriebssystem- und Maschinenspezifische Programmprobleme minimieren zu können.

Design by Contract

Dies bezieht sich auf die Interaktion von Programm-Modulen. Die Metapher eines Vertrags bezieht sich auf Bedingungen, die vom aufgerufenen Programm gegenüber dem aufrufenden Programm erfüllt werden müssen. Es wird „vertraglich“ zugesichert, dass das angebundene Modul die Anforderungen präzise erfüllt. (Hausherr, 2006)

Pre- & Postconditions

Bevor ein Abschnitt oder eine Funktion zur Ausführung gelangt, wird überprüft ob die der Berechnung zugrunde liegenden Daten spezifischen Eingangsbedingungen genügen. Falls diese verletzt sind, wird eine Fehlerbehandlung aufgerufen, die statistisch ermittelte Werte annimmt oder eine neue Berechnung anordnet. Genauso wird mit der Postcondition verfahren, denn das bearbeitende Programm weiss selbst am besten, welche Werte als Ausgabe in Frage kommen. (Buettner, 2008)

Fazit

Softwarefehler sind Folgen von menschlichen Fehlern. Sie sind nicht gänzlich zu vermeiden und treten immer wieder auf. Speziell in sensibler Umgebung, in welcher Menschenleben vom korrekten Funktionieren einer Software abhängen, muss aber unbedingt auf stabile und fehlerfreie Softwareentwicklung Wert gelegt werden. Jegliche Nichtberücksichtigung von vormals getroffenen Annahmen oder ungenügendes Testen kann zu verheerenden Katastrophen führen und viele Menschenleben gefährden. Aber auch im Alltag trifft man immer wieder auf Softwarefehler, welche sich durch abstürzende Betriebssysteme oder nicht mehr antwortende Programme zeigen. Mittels konzentrierter Arbeit und ausführlichem Testen können solche Fehler aber so gut wie möglich minimiert werden, um ein fehlerfreies Funktionieren der Software bestmöglich zu garantieren.

Literaturverzeichnis

Arnold, D. N. (24. September 1996). *Patriot Missile Failure*. Abgerufen am 15. März 2011 von University of Minnesota: http://www.ima.umn.edu/~arnold/455.f96/disasters.html

Blair, M. (4. Februar 1992). *Federation of American Scientists*. Abgerufen am 15. März 2011 von United States - General Accounting Office: http://www.fas.org/spp/starwars/gao/im92026.htm

Buettner, F. (17. April 2008). *Validating Pre- and Postconditions - Uni Bremen*. Abgerufen am 15. März 2011 von http://www.db.informatik.uni-bremen.de/projects/USE/prepost.html

DATACOM GmbH, B. (Hrsg.). (13. Juni 2006). *itwissen.info*. Abgerufen am 15. März 2011 von http://www.itwissen.info/definition/lexikon/Programmfehler-bug.html

ESA. (14. Mai 2004). *European Space Agency*. Abgerufen am 2011. März 15 von http://www.esa.int/esaM/Launchers_Home/SEMN2E67ESD_0.html

Glinz, M. (2003). *Requirements Engineering.* Abgerufen am 15. März 2011 von www.ifi.uzh.ch/req/ftp/ses/kapitel_09.pdf

Hausherr, M. (15. November 2006). *Seminar Enterprise Computing.* Abgerufen am 15. März 2011 von http://www.gruntz.ch/courses/sem/ws06/DBC.pdf

Jackson, T. (1998). Inside Intel. In T. Jackson, *Inside Intel.* Hoffmann und Campe.

Knieschewski, S. (8. Januar 2004). *Universität Koblenz.* Abgerufen am 14. März 2011 von http://www.uni-koblenz.de/~beckert/Lehre/Seminar-Softwarefehler/Folien/knieschewski.pdf

Lum, A. (kein Datum). *Patriot Missile Software Problem*. Abgerufen am 15. März 2011 von Verification – Software Bug Report: http://sydney.edu.au/engineering/it/~alum/patriot_bug.html

Uherek, E. (5. Mai 2004). *MPI for Chemistry, Mainz*. Abgerufen am 15. März 2011 von http://www.atmosphere.mpg.de/enid/Basis/2__Ozonloch_1mg.html

Weyand, C. (10. 11 2003). *Universität Koblenz.* Abgerufen am 15. März 2011 von http://www.uni-koblenz.de/~beckert/Lehre/Seminar-Softwarefehler/Ausarbeitungen/weyand.pdf

Wikipedia.org. (7. Dezember 2010). *Wikipedia, Die freie Enzyklopädie.* Abgerufen am 14. März 2011 von http://de.wikipedia.org/w/index.php?title=Pentium-FDIV-Bug&oldid=82401223

Wikipedia.org. (2. März 2011). *Wikipedia, Die freie Enzyklopädie.* Abgerufen am 15. März 2011 von http://de.wikipedia.org/wiki/Softwarefehler

Wired. (24. Juli 1998). Abgerufen am 14. März 2011 von http://www.wired.com/science/discoveries/news/1998/07/13987