

Software-Sicherheit als gesteuerter Zufall

Schwer abschätzbare Zuverlässigkeit von Gesamtsystemen denn
Programmfehler lassen sich nie ausschliessen

Von Max Kleiner

Computer werden immer leistungsfähiger. Ob sie auch sicherer werden, ist allerdings zweifelhaft. Zwar lässt sich die Funktionstüchtigkeit der Hardware gut abschätzen und den Bedürfnissen anpassen. Schwieriger ist die Beurteilung der Zuverlässigkeit von Software. Und noch komplizierter ist es, die Betriebssicherheit von Gesamtsystemen zu evaluieren.

Wer einen Computer kauft, nimmt als selbstverständlich an, dass dieser auch einwandfrei funktioniert. Um so grösser ist der Ärger, wenn dies nicht zutrifft. Den Schaden trägt aber nicht nur der enttäuschte Kunde, sondern letztlich auch der Hersteller, denn schlechte Nachrichten machen rasch die Runde und verunsichern potentielle Käufer.

Dass Qualität zum Geschäftserfolg eines Unternehmens beiträgt, haben zumindest die Hardwarefabrikanten im Laufe der Jahre gelernt - wenn auch oft auf schmerzliche Art und Weise. Hingegen scheinen viele Softwarehersteller diese einfache Regel immer noch nicht zu kennen.

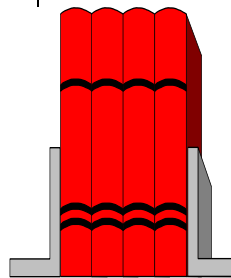
Was aber letztlich zählt, ist die Zuverlässigkeit des ganzen Systems. Fachleute verstehen darunter die Wahrscheinlichkeit, dass dieses im Betrieb während einer bestimmten Zeit nicht versagt. Das sollte nicht etwa eine Frage banger Hoffens sein, sondern auf gesichertem Wissen beruhen. Als Mass für die Zuverlässigkeit dient entweder die Anzahl Fehler, die pro Zeiteinheit durchschnittlich auftreten, oder die statistisch errechnete Zeitspanne, in der kein Fehler auftreten wird.

Risikofaktor Software

Für die Hardware kann man solche Werte relativ einfach ermitteln. Die Ausfallrate elektronischer Bauteile sollte jedem seriösen Hersteller bekannt sein. Wenn er zum Beispiel weiss, dass ein gewisser Kondensator durchschnittlich einmal in 1000 Stunden ausfällt, wird er wahrscheinlich zwei

dieser Bauteile für die gleiche Funktion einbauen, womit an dieser Stelle des Systems eine nach Wahrscheinlichkeitsrechnung pannenfreie Betriebsdauer von einer Million Stunden gewährleistet ist (1000 mal 1000 Betriebsstunden).

Allerdings darf man die Bauteile eines Systems nicht isoliert betrachten, sondern muss damit rechnen, dass defekte Komponenten andere Komponenten negativ beeinflussen. Mit dem Bereitstellen zusätzlicher Redundanz sollte aber auch dieses Problem lösbar sein.



Damit lässt sich praktisch jede gewünschte Hardware-Zuverlässigkeit erreichen.

Nun bestehen aber moderne Systeme nicht aus Hardware allein, sie hängen auch von Software ab. Wenn diese versagt, nützt das beste Gerät nichts mehr. Die Hauptsorge des Systemlieferanten müsste also

weniger der Ausfall physischer Bauelemente sein, sondern ein Versagen der Software.

Überraschenderweise denken aber viele Hersteller gar nicht an Programmfehler, wenn sie die Zuverlässigkeit eines Systems taxieren sollen. Weshalb? Ein Grund dafür könnte sein, dass die Software-Zuverlässigkeit oft nach sehr subjektiven Kriterien beurteilt wird - zum Beispiel dem schieren Glauben eines Programmentwicklers an die Korrektheit seines Codes.

Während also die Fehlerrate der Hardware statistisch berechenbar ist, tappt man bei der Bestimmung der Software-Ausfälle noch weitgehend im dunklen. Verschiedene Fachleute glauben zudem, dass nicht nur gewöhnliche Programmfehler zu einem Versagen führen, sondern dass manchmal auch der gesteuerte Zufall eine Rolle spielt. Der Grund: Software ist eben keine so logische, abstrakte Sache, wie man gemeinhin annimmt.

Im Betrieb ist alles anders

Viele Software-Fehler treten nur unter ganz bestimmten und meist ungewöhnlichen Umständen auf, zum Beispiel bei einer ganz speziellen Kombination von Input, Systemzustand und Ausführungspfad im Programm. Ich bezeichne diese Eventualität als **Magic Logic**.

Schon bei relativ einfachen Aufgaben gibt es da so viele Möglichkeiten, dass niemand voraussehen kann, welche Kombinationen allenfalls zu einem Fehler führen könnten.

Trotz dieser Gefahr wird die Betriebstauglichkeit von Software auch heute noch mit ziemlich hölzer-

nen Methoden eruiert: zum Beispiel, indem die Programmierer ihre Produkte mit standardisiertem Input füttern und dann messen, wie schnell und zuverlässig die Daten verarbeitet werden.

Für die tägliche Praxis sind solche Tests leider nur bedingt aussagekräftig, weil die Betriebssicherheit eines Programms auch von den äusseren Umständen abhängt.

Man müsste also das System in möglichst wirklichkeitsnahen Situationen testen. Bei einer so vielseitigen Maschine wie dem Computer kann man aber nur einen winzigen Bruchteil aller denkbaren Fälle ausprobieren. Fazit: Die Gewissheit, dass kein Software-Fehler auftreten wird, hat man nie.



Am 15. Januar 1990 führte ein kleiner Programmfehler zum Zusammenbruch des gesamten AT&T-Telefonnetzes in den USA. Ein Vermittlungscomputer meldete korrekt einen Fehler. Automatisch wurden alle Gespräche auf benachbarte Netzwerkzentralen umgeschaltet.

Nur, als der Computer wieder richtig aufgeschaltet war, fand er nicht die Programmzeile, dies den anderen Netzwerken zu melden. Er sandte weiter Umschaltbefehle, bis sämtliche Telefonzentralen überlastet zusammenbrachen. Für neun Stunden war der kontinentale und der internationale Telefonverkehr unterbrochen.

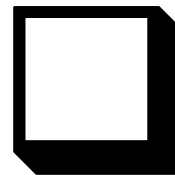
Wenn in der Testphase der Software-Entwicklung Fehler entdeckt und ausgemerzt werden, sollte die Zuverlässigkeit des Programms eigentlich zunehmen.

Man muss das tatsächlich so vorsichtig formulieren, denn die Voraussetzung dazu ist, dass sich durch die Korrekturen nicht wieder neue Fehler einschleichen, was leider, auch aus eigener Erfahrung sehr häufig vorkommt.

Statistik als Qualitätsförderung

Wie viele Fehler im System stecken, mag wohl eine interessante Zahl sein, aber isoliert betrachtet sagt sie wenig aus. Ein System mit vielen, selten auftretenden Fehlern kann nämlich durchaus zuverlässiger sein als eines mit wenigen Fehlern, die dafür umso häufiger vorkommen. Die Erfahrung zeigt, dass die erste Variante eher der Realität entspricht als die zweite: Selbst nach sprichwörtlich Tausenden von Betriebsjahren (man untersuchte gleichzeitig viele Geräte) fördern komplexe Systeme immer wieder neue Fehler zutage; dazwischen arbeiten sie zu voller Zufriedenheit der Anwender.

Sollen denn die Hersteller angesichts all dieser Schwierigkeiten gar keine Statistiken mehr betreiben? Das wäre eine total falsche Konsequenz. Solche Untersuchungen werden zwar nie die Fehlerfreiheit eines Systems beweisen, aber sie können sehr viel zur Verbesserung seiner Qualität beitragen.



Das alleine rechtfertigt schon den grossen Aufwand Fehlerstatistiken zu führen und den Lizenznehmern in regelmässigen Abständen Fragebögen zukommen zu lassen, die einerseits im Hinblick auf erweiterte Funktionen, andererseits auf schnellere Routinen hin von Nutzen sein können.

Allgemeine Zwecke von Fehlerstatistiken können sein:

- ⇒ Rationalisierung von Updates
- ⇒ Optimierung von Programmen
- ⇒ allg. Information

In der Regel werden Fehler dadurch erkannt, dass bei der Arbeit mit einem Programm Abweichungen von der erwarteten Funktion festgestellt werden. In diesem Fall ist zunächst einmal intensiv abzuklären, ob diese unerwarteten Ergebnisse nicht auf Grund einer *Fehlbedienung* zustande gekommen sind.

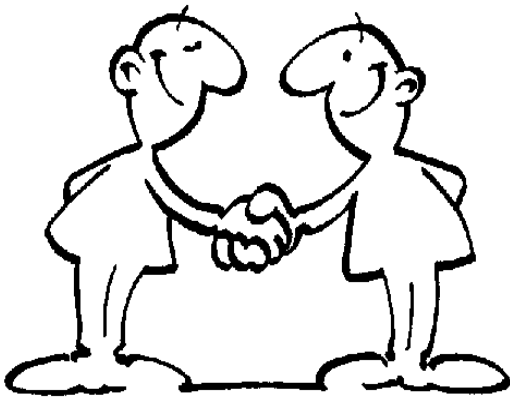
What about "Software-Redundanz"?

Die Computerindustrie und andere, die mit Hardware-Redundanz arbeiten, weisen hinsichtlich der Sicherheitsrisiken gern auf die bereits erwähnte Multiplikationsregel hin:

$$p(A \text{ und } B) = p(A) \cdot p(B).$$

Hardware-Redundanz hat freilich ihre Tücken. Parallel geschaltete Sensoren oder Aktoren können von derselben Fehlerquelle und (oder) zum gleichen Zeitpunkt gestört werden. Ähnliche Misslichkeiten können bei Software-Redundanz auftreten. Selbst dann, wenn sich ein Kontrollautomatismus auf das Mehrheitsvotum verschiedener Computerprogramme stützt, die von unabhängig arbeitenden Teams gestrickt worden sind - *selbst dann* scheitert das System zuweilen an demselben Programmierfehler, der in sämtlichen Programmen parallel auftaucht.

Wie alle Menschen neigen auch Programmierer immer wieder zu ganz bestimmten Irrtümern. Ausserdem muss der Ablauf der Teilprogramme koordiniert werden, eine Programmieraufgabe, die wiederum zur Fehlerquelle werden kann.



In anderen Fällen sind die Programmteile in Ordnung, aber das Programm fürs "Voting" spielt verrückt - so geschehen 1981 beim ersten Countdown der US-Raumfähre. Nun, auch das Redundanz-Konzept hat seine Schwachstellen und wo ich gerade dabei bin: Sprache ist super-redundant, hbn S ds schn gmrkt?

Als abschliessendes Fazit gilt zu sagen, dass zwar das Testen eines Programms auf Fehlerfreiheit von der Logik her ein unlösbares Problem darstellt, dieses aber durch eine transparente und offene Kommunikation zwischen Softwarehäusern und Lizenznehmern mehr als gemildert werden kann.