Software Engineering I

Prof. Dr. Martin Glinz

Fallstudie:

Ariane Flug 501

Universität Zürich
Institut für Informatik

Was geschah mit Flug 501?

So hätte es aussehen sollen...

...und so sah es tatsächlich aus

Chronik der Ereignisse – 1

- Die Software für das Trägheitsnavigationssystem wird unverändert von der Ariane 4 übernommen. Ein Test dieser Software unterbleibt daher.
- Die übrigen Systeme der Rakete werden komponentenweise gründlich getestet. Ein gemeinsamer Test der gesamten Steuerungssoftware der Rakete unterbleibt aus Kosten- und Machbarkeitsgründen.
- In der Software für das Trägheitsnavigationssystem gibt es eine Abgleichsfunktion, deren Werte eigentlich nur sinnvoll sind, solange die Rakete noch nicht fliegt. Diese Funktion arbeitet programmgemäß bis ca. 40 s nach H0 weiter, weil das bei der Ariane 4 im Fall eines Countdownabbruchs kurz vor dem Abheben sinnvoll war.
- Flug 501 startet am 4. Juni 1996. Die Triebwerke zünden um H0=9:33:59 Ortszeit. Die ersten 36 Sekunden des Flugs verlaufen normal.

Chronik der Ereignisse – 2

- Da die Ariane 5 eine andere Flugbahn hat als die Ariane 4, berechnet die Abgleichsfunktion einen Wert, der wesentlich größer ist als erwartet.
- Bei der Konvertierung dieses Werts von einer 64 Bit Gleitkommazahl in eine 16-Bit Festkommazahl tritt ein Überlauf ein; der Rechner erzeugt eine Ausnahmebedingung.
- Die Ausnahmebedingung wird nicht behandelt (obwohl dies in der verwendeten Programmiersprache Ada möglich wäre).
- Der Trägheitsnavigationsrechner setzt eine Fehlermeldung an den Steuerrechner der Rakete ab und schaltet sich 36,75 s nach H0 ab.
- Das Trägheitsnavigationssystem ist aus Sicherheitsgründen doppelt ausgelegt. Ein Umschalten auf das zweite System schlägt fehl, da dieses System das gleiche Problem gehabt und sich vor 0,05 s ebenfalls abgeschaltet hat.

Chronik der Ereignisse – 3

- Die Software des Steuerrechners ist auf den Ausfall beider Trägheits-navigationssysteme nicht ausgelegt und interpretiert die gemeldeten Fehlercodes als Flugbahndaten.
- Dies führt zu völlig unsinnigen Berechnungen und als Folge davon zu unsinnigen Stellbefehlen an die Steuerdüsen der Rakete: Diese werden bis zum maximal möglichen Anstellwinkel ausgeschwenkt.
- Aufgrund der resultierenden Scherkräfte zerbricht die Rakete, worauf der Selbstzerstörungsmechanismus ordnungsgemäß anspricht. Dieser sprengt Rakete und Nutzlast und verhindert damit, dass größere Trümmerteile auf den Boden fallen.

Untersuchung der Ursachen

- Die Untersuchungskommission hat Glück: aus den gefundenen Trümmern lässt sich die Unfallursache exakt rekonstruieren:

  Lions, J.L. (1996). *ARIANE 5 Flight 501 Failure*. Report by the Inquiry Board. Paris: ESA. http://ravel.esrin.esa.it/docs/esa-x-1819eng.pdf

Schaden

- 4 Satelliten verloren: 400-500 M Euro
- 2 Jahre Verzug im Entwicklungsprogramm: > 500 M Euro
- 2 zusätzliche Erprobungsstarts

- bei Gesamtkosten des Projekts von 1987 bis 1998 von 6700 M Euro

- ➪ Was können wir daraus lernen für Software Engineering?

Spezifikation und Entwurf – 1

- **Spezifikation:** Bestehende Software darf nicht unbesehen für eine neue Aufgabe *wiederverwendet* werden. Vorher muss geprüft werden, ob ihre Fähigkeiten den Anforderungen der neuen Aufgabe entsprechen.
- **Dokumentation:** Die *Fähigkeiten* einer Software sowie alle *Annahmen*, die sie über ihre Umgebung macht, müssen *dokumentiert* sein. Andernfalls ist die Prüfung auf Wiederverwendbarkeit extrem aufwendig.
- **Design by Contract:** Kooperieren zwei Software-Komponenten miteinander, so müssen eindeutige *Zusammenarbeitsregeln* definiert, dokumentiert und eingehalten werden: Wer liefert wem was unter welchen Bedingungen.

Spezifikation und Entwurf – 2

- **Fehlerbehandlung:** Jede potentielle *Fehlersituation* in einer Software muss entweder *behandelt* werden oder die Gründe für die Nichtbehandlung müssen so *dokumentiert* werden, dass die Gültigkeit der dabei getroffenen Annahmen überprüfbar ist.
- **Software ≠ Hardware:** Mehrfache identische Auslegung von Systemen hilft nicht gegen logische Fehler in der Software.
- **Sicherheit:** Bei Störungen in sicherheitskritischen Systemen ist *Abschalten* nur dann eine zulässige Maßnahme, wenn dadurch wieder ein sicherer Zustand erreicht wird.

Qualitätsmanagement

- **Test:** Bei der *Prüfung* von Software, die aus mehreren Komponenten besteht, genügt es *nicht*, jede Komponente *nur isoliert* für sich zu prüfen.

  Umfangreiche Systemtests unter möglichst realistischen Bedingungen sind notwendig.

- **Review:** Jedes Programm muss – neben einem sorgfältigen Test – durch kompetente Fachleute *inspiziert* werden, weil insbesondere die Erfüllbarkeit und Adäquatheit von Annahmen und Ergebnissen häufig nicht testbar ist.

- **Effektivität:** Software, die nicht benötigt wird, sollte auch nicht benutzt werden.

Projektführung

- **Risikomanagement:** Die *Risiken* erkennen, angemessene technische Maßnahmen (siehe oben) planen, durchsetzen und überprüfen.