I. Murphy's Gesetze (Murphy's laws)

Urheber:

1949 von Air Force Captain
Edward A. Murphy

Anlass:

Fehlerhafte Messwertaufnahme bei Raketenschlittenexperiment

Kommentar zum veranwortlichen Techniker:

"If there is any way to do it wrong, he'll find it."

Abb.: Der Testpilot wird auf den Raketenschlitten geschnallt

Murphy's Law: Top Five

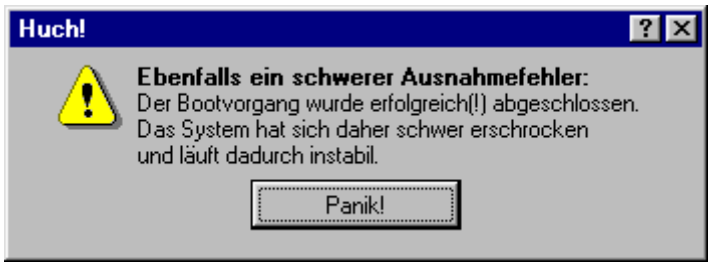
1. If anything can go wrong, it will

2. If there is a possibility of several things going wrong, the one that will cause the most damage will be the one to go wrong

3. If anything just cannot go wrong, it will anyway

4. If you perceive that there are four possible ways in which something can go wrong, and circumvent these, then a fifth way, unprepared for, will promptly develop

5. The hidden flaw never stays hidden for long



II. Die teuersten Softwarefehler

Der Vorreiter: die US-Raumsonde Mariner 1

Start der ersten amerikanischen Venussonde am 22.Juli 1962

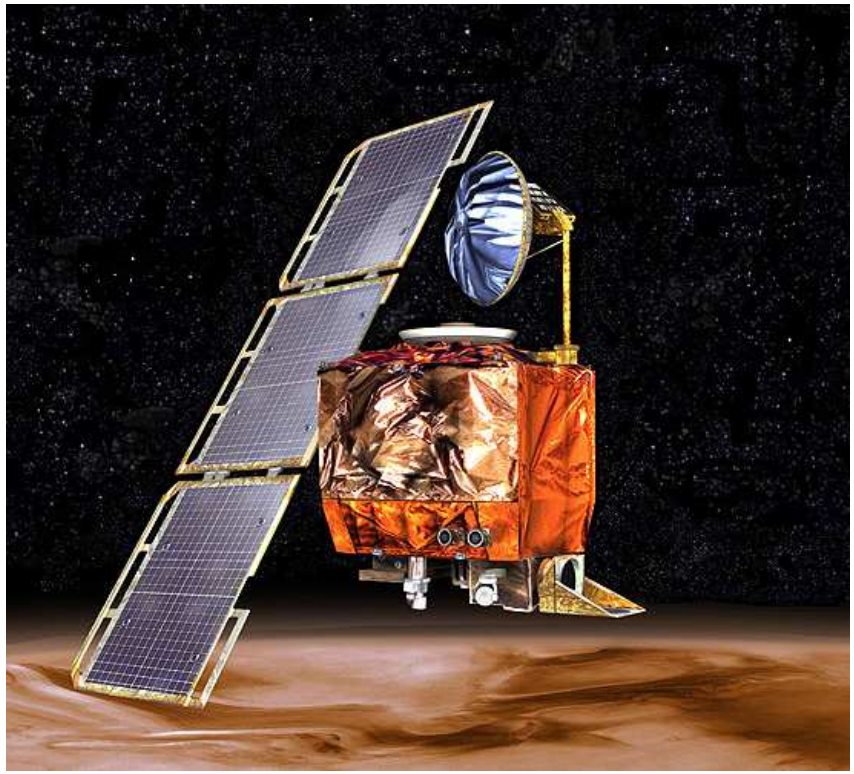
Absturzursache:
Ausschnitt aus dem FORTRAN-Programm zur Steuerung der Flugbahn:

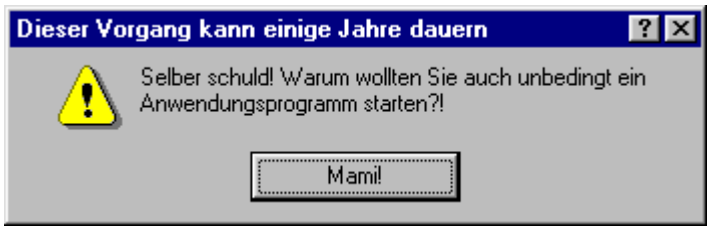
```
DO 5 K = 1. 3
   ...
5 CONTINUE
```

Entscheidender Fehler: Punkt statt Komma!

Vom Compiler interpretiert als: `DO5K = 1.3`
Nicht erkannt als Schleife, sondern als Variablenzuweisung

Der jüngste Nachfolger: Mars Climate Orbiter





Start am 11.09.1998

Ziele:

Erreichen der Mars-Umlaufbahn

Kartograhpieren der Mars-Oberfläche

Absturzursache:

Angesteuerte Umlaufbahn war 170 km tiefer als geplant

vorher

Das teuerste Feuerwerk der EU-Raumfahrtsgeschichte: Die Trägerrakete Ariane 5

Jungfernflug der europäischen Trägerrakete am 4. Juni 1996

Mit an Bord: 4 Cluster-Satelliten

Absturzursache:

Überlauf bei Umwandlung einer Fliesskommazahl in eine Ganzzahl

Auszug aus dem Ada-Programm des Trägheits-Navigationssystems:

```
...
declare
  vertical_veloc_sensor: float;
  horizontal_veloc_sensor: float;
  vertical_veloc_bias: integer;
  horizontal_veloc_bias: integer;
  ...
```

```
begin
  declare
    pragma suppress(numeric_error, horizontal_veloc_bias);
  begin
    sensor_get(vertical_veloc_sensor);
    sensor_get(horizontal_veloc_sensor);
    vertical_veloc_bias := integer(vertical_veloc_sensor);
    horizontal_veloc_bias := integer(horizontal_veloc_sensor);
    ...
  exception
    when numeric_error => calculate_vertical_veloc();
    when others => use_irs1();
  end;
end irs2;
```

Sprengung 39 Sekunden nach der Zündung

Geschätzte Gesamtkosten:
1700 Millionen DM

nachher

Automatisierungstechnik bei der Bundesbahn:

Das Eisenbahn-Stellwerk Altona

Absturz des Systems 13.3.1995

Stellwerk muss geschlossen werden

Das Denver-Koffer-Debakel: Denver Airport Gepäcksystem

Eröffnung um 16 Monate verspätet

Zerquetschte und verlorene Koffer

Sortierung von Hand notwendig

Zurück ins Mittelalter: Der US-Stromausfall von 2003

Acht Staaten sind für fünf Tage ohne Strom

Ursache:
Fehler in Managementsoftware führt zu Überlastung der benachbarten Stromnetze

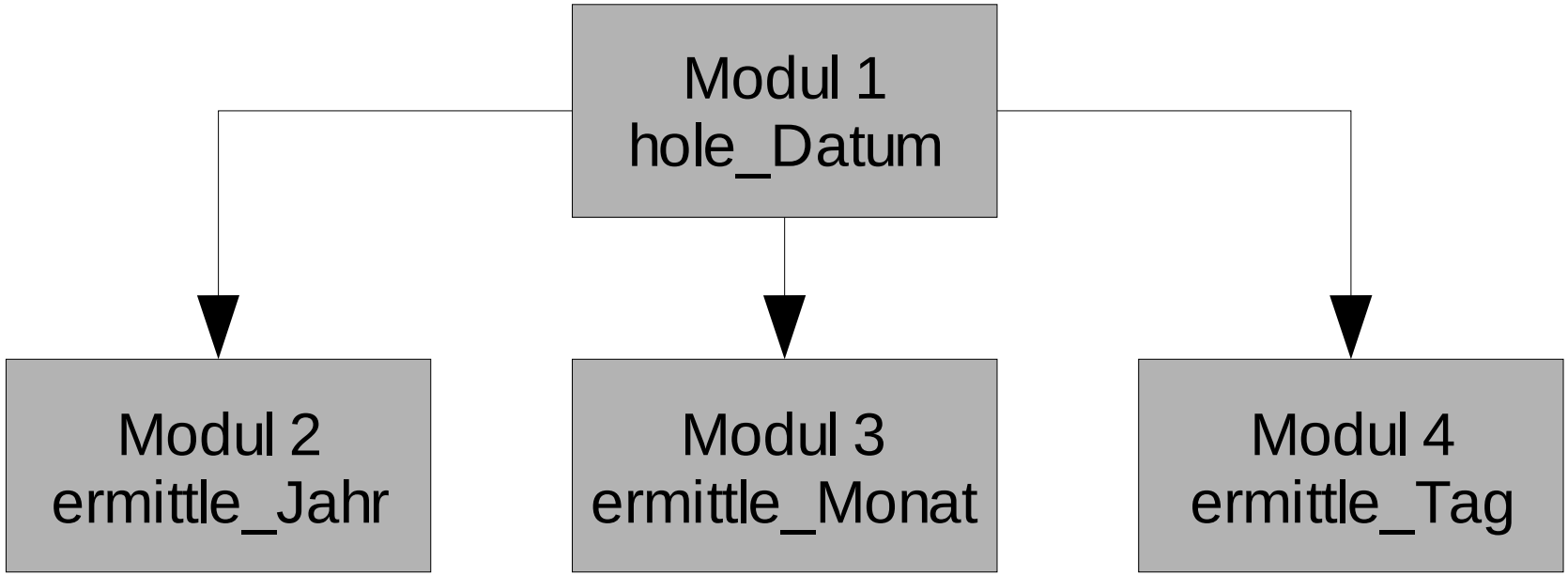
Kommunikationsprobleme:

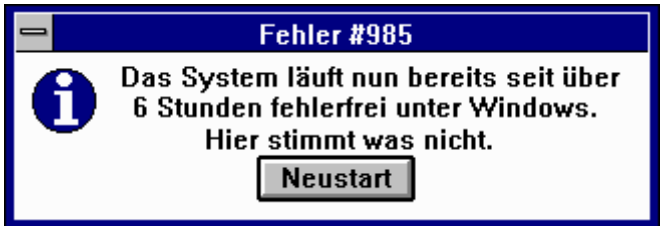
Ausfall des AT&T-Telefonnetzes am 15.01.1990

Neun Stunden lang kann ein Grossteil der Ferngespräche nicht vermittelt werden

III. Die theoretischen Grundlagen

Einfaches Beispiel: Ermittlung von Tag, Monat und Jahr anhand des Systemdatums (z.B. „Montag, der 28.Juni 2004“).

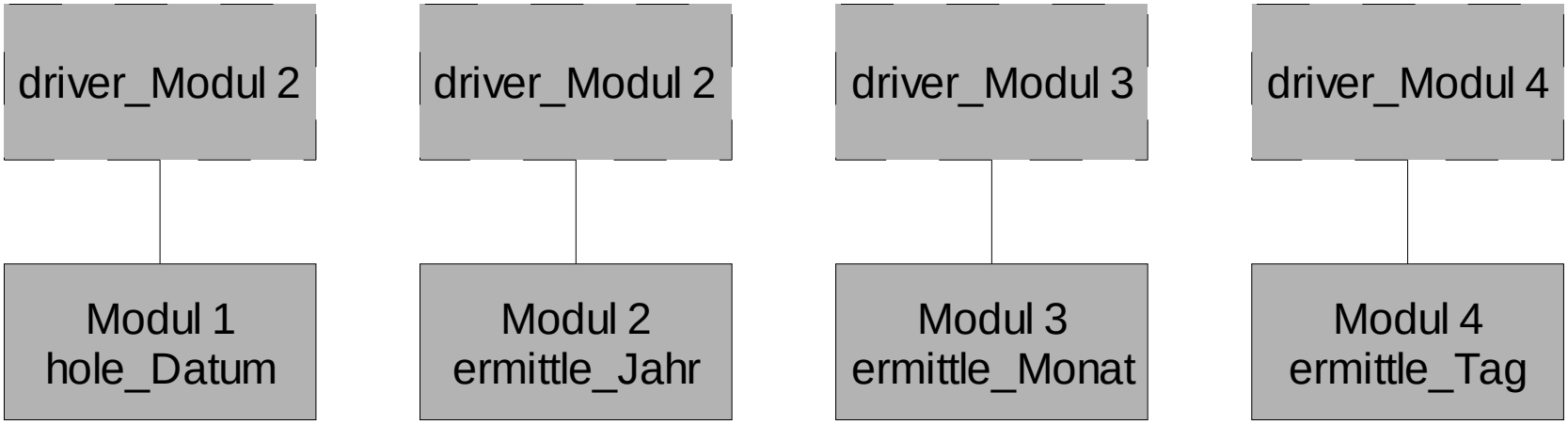




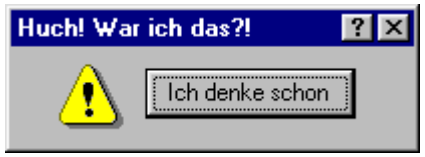
1. Top-down versus Bottom-up-Strategie beim Modultest

1.1. Bottom up

Module werden isoliert getestet
Treiberprogramme (driver) werden benötigt

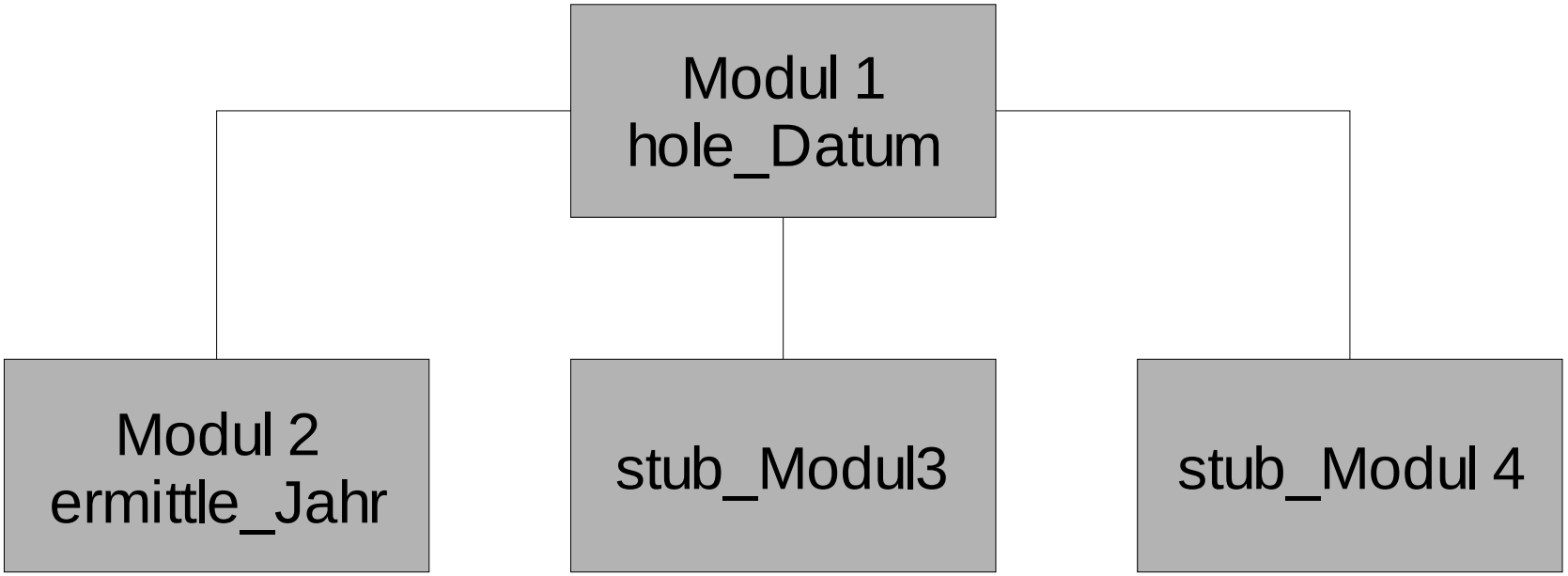


\+ Beobachtung der Ergebnisse einfach
\- Programm wird nicht als Ganzes geprüft



1.2. Top down

Untermodule werden zunächst durch Platzhalter ersetzt und inkrementell wieder eingefügt



\+ Fehler im obersten Modul werden besser erkannt
\- Beobachtung der Ergebnisse schwieriger

2. Der White-Box-Test

Der gesamte Programmcode ist dem Tester bekannt

Testkriterium:
Ausführung möglichst jeden Programmpfades beim Testen

Liste der Testabdeckungen:

C0	Alle Anweisungen in einem Modul 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 be jeder logischen Bedingung zumindest einmal ausgeführt wird.
C2	Alle Programmsegmente werden wenigstens einmal ausgeführt. Bei Schleifen muss so getestet werden, dass Schleife 1. nicht ausgeführt wird, 2. mit einem niederen Wert des Schleifenzählers geteststst 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

3. Der Black-Box-Test

Der Tester beurteilt nur die Funktion, ohne den Aufbau zu kennen

1. *Ein Programmierer sollte nie versuchen, sein eigenes Programm zu testen*

2. *Das Testen von Software ist ein Experiment*

3. *Das Testen ist definiert als die Ausführung eines Programms mit der erklärten Absicht, Fehler zu finden*

4. *Definition der erwarteten Ergebnisse vor dem Beginn des Tests*

5. *Die Wahrscheinlichkeit, in der näheren Umgebung eines bekannten Fehlers weitere zu finden, ist überproportional hoch*

4. Der Gray-Box-Test

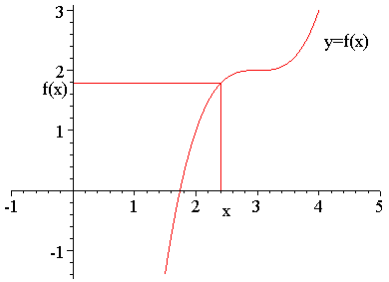
Mischung aus White und Black Box

Tester kennt in groben Zügen die Impementierung

Vorteile bei Kompatibilitätstests, grösserer Überblick

5. Die wichtigsten Testausprägungen

5.1. Der Funktionstest



5.2. Der Volume-Test

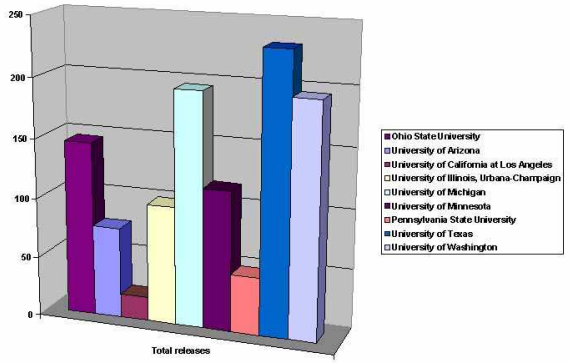
5.3. Der Stress-Test

5.4. Speicherverbrauch und Auslastung des Prozessors

5.5. Recovery Testing

5.6. Der Mutationstest

5.7. Benchmarks

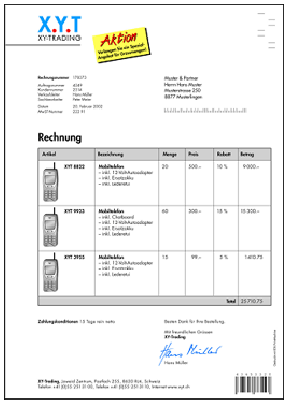


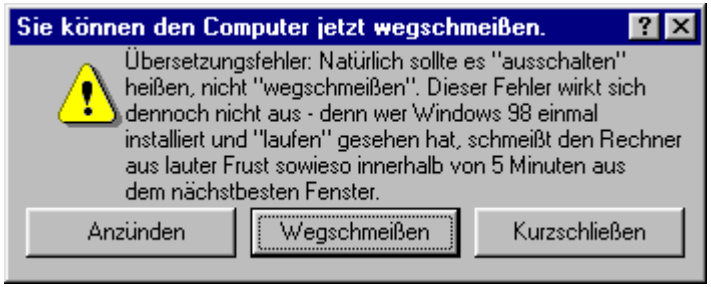
5.8. Der Test von Prozeduren und Verfahren

5.9. Configuration Testing

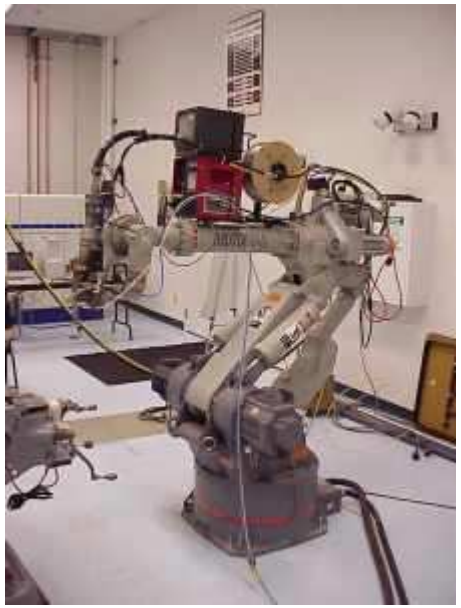
5.10. Usability Testing

5.11. Überprüfung von Dokumenten





5.12. Der Systemtest



6. Test-Automation

Vermeidung von für den Benutzer langwierigen und ermüdenden Tests

Automatisierte Testabläufe ohne Benutzereingriff möglich

Testläufe lassen sich auf verschiedenen Rechnerkonfigurationen wiederholen

Quellmaterial:

1. Georg Erwin Thaller – Software-Test: Verifikation und Validation

2. Murphy's Laws: http://www.murphys-laws.com/

3. Mars Climate Orbiter:
   http://wwwzenger.informatik.tu-muenchen.de/lehre/seminare/semsoft/unterlagen_02/erdfern/website/mco.html

4. Kleine BUGs, grosse GAUs:
   http://wwwzenger.informatik.tu-muenchen.de/~huckle/bugs.html

6. Software Reliability:
   http://www-aix.gsi.de/~giese/swr/allehtml.html