1. Murphy’s Law – Murphy’s Gesetz

- US Air Force Captain Edward A.Murphy entwickelt Ausrüstung zur Messung der Beschleunigung
- Wird von Testpiloten getragen, um zu messen, wieviel Beschleunigung der menschliche Körper aushält (USAF Projekt MX981)
- Messwandler (transducer) war mit mehreren Sensoren bestückt
- Ausrüstung wird an Raketenschlitten angebracht, Experiment wird ein Fehlschlag
- Murphy untersucht Anschluss aller Sensoren, zwei Arten möglich, nur eine richtig, natürlich genau falsch angeschlossen
- Murphy verflucht verantwortlichen Techniker:
  "If there is any way to do it wrong, he'll find it."

- Projektmanager führt eine Liste von Gesetzen und fügt dieses hinzu, er nennt es Murphy’s Gesetz
- Beteiligter Mediziner behauptet in einer Pressekonferenz, dass die geringen Unfallzahlen des Projekts nur durch den festen Glauben an Murphy’s Law und an die Notwendigkeit, diese zu umgehen, erreicht werden konnten.
- Luftfahrtindustrie setzt Murphy’s Law in der Werbung ein, die Legende ist geboren

Die persönlichen Top Five:

1. **If anything can go wrong, it will**
   Was schiefgehen kann, geht schief
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
   Wenn verschiedene Dinge schiefgehen können, wird immer das kostenträchtigste schiefgehen
3. If anything just cannot go wrong, it will anyway
   Wenn etwas einfach nicht schiefgehen kann, wird es trotzdem schiefgehen
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
   Wenn man an vier Fehlermöglichkeiten denkt und diese umgeht, dann wird sich spontan ein fünfer unabgedeckter Fehlerfall entwickeln
5. The hidden flaw never stays hidden for long
   Die verborgenen Fehler bleiben nicht lange verborgen

2. Die teuersten Softwarefehler

2. 1. Der Vorreiter: Die US-Raumsonde Mariner 1

- Start der ersten amerikanischen Venussonde am 22.Juli 1962

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

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

Entscheidender Fehler: Punkt statt Komma!
Compiler erkennt Programmcode zu:

```
   DO5K = 1.3
```

Wertzuweisung an eine nicht deklarierte Variable
Kein Durchlauf der (nicht vorhandenen) Schleife

- Abweichung der Trägerrakete von der vorgesehenen Flugbahn
- Zerstörung der Rakete nach 290 Sekunden

Ursache:

- Fortran erlaubt Leerzeichen in Namen und Zahen
- Variablen-Deklaration nicht notwendig
- Strukturierte Schleife (END DO) nicht möglich

Geschätzte Kosten: 18,5 Mill. US-Dollar

2.2. Der jüngste Nachfolger: Mars Climate Orbiter

- Start am 11.09.1998
- Ziele: Erreichen der Mars-Umlaufbahn, Unterstützung des Mars Polar Lander, Kartographieren der gesamten Oberfläche
- Angesteuerte Umlaufbahn 170 km tiefer als geplant
- Dadurch Absturz oder Abprall
- Grund: zwei Gruppen der Nasa waren am Projekt beteiligt, die eine rechnet in Fuss, die andere in Meter

2.3. Das teuerste Feuerwerk der EU-Raumfahrt: Die Trägerrakete Ariane 5

- Jungfernflug der europäischen Trägerrakete am 4. Juni 1996
- Mit an Bord: 4 Cluster-Satelliten

Ausschnitt 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;
```

Der Ablauf der Katastrophe:

- 37 Sekunden nach Zündung erreicht Ariane 5 in 3,7 km Flughöhe eine Horizontal-Geschwindigkeit von 32768 Einheiten, etwa fünfmal mehr als die Ariane 4
- Umwandlung in eine Ganzzahl führt zu einem Überlauf, der jedoch nicht abgefangen wurde

- Ersatzrechner hatte das Problem schon 72 ms vorher und schaltete sich ab.
- Diagnose-Daten wurden zum Hauptrechner geschickt, die dieser als Flugbahndaten interpretierte
- Unsinnige Steuerbefehle werden an die seitlichen Feststoff-Triebwerke und später auch an das Haupttriebwerk geschickt, um die grossen Flugabweichungen auszugleichen
- Rakete druht auseindanderzubrechen und wird gesprengt (39 Sekunden nach Zündung)

Die Gründe für den übersehenen Bug:

- Nur bei 3 von 7 Variablen wurde ein Überlauf geprüft
- Für de anderen 4 Variablen existierten mathematische Beweise, dass die Werte klein genug blieben (bezogen auf Ariane 4)
- Beweise galten nicht für die Ariane 5, sie wurden hierfür gar nicht nachvollzogen
- Beim Programm-Design ging man nur von Hardware-Fehlern aus
- Ersatzrechner hatte identische Software, Rechner sollte sich im Fehlerfall abschalten und an Ersatzrechner übergeben, da Neubestimmung der Flughöhe zu aufwendig
- Da Ersatzrechner schon vorher ausgefallen war, wurden nur die Diagnosedaten gesendet

Grundlegendes Problem der Wiederverwendbarkeit von Software

Die Kosten:

250 Mill. DM Startkosten
850 Mill. DM für Cluster-Satelliten
600 Mill. DM für nachfolgende Verbesserungen
Verdienstausfall für 2 bis 3 Jahre

2.4. Automatisierungstechnische Bemühungen der Bundesbahn: Das Eisenbahn-Stellwerk Altona

- 1995 soll altes Stellwerk mit 50 Beschäftigten durch Intel 486-Echtzeit-System ersetzt werden
- es wären nur noch 10 Personen zum Betrieb notwendig
- Nach Inbetriebnahme am 13.3.95 Absturz des Rechners, Stellwerk muss geschlossen werden, daraus resultiert Chaos im gesamten Bundesbahnverkehr
- Ursache: System versucht bei starkem Verkehr 4 kByte Stack (Kellerspeicher) anzulegen, obwohl nur 3,5 kByte benötigt und vorhanden sind

2.5. Auch Flughäfen geht es nicht besser: Das Denver-Koffer-Debakel

- Bei Neueröffnung des Flughafens in Denver soll automatisiertes Gepäcksystem verwendet werden
- 300 Computer, Laserscanner, Photozellen, Ethernet-Netzwerk
- Eröffnung durch Gepäcksystem um 16 Monate verspätet
- Bei Inbetriebnahme Netzwerküberlastung, Sortier-Anweisungen kamen nicht rechtzeitig
- Resultat: zerquetschte und verlorene Koffer
- Gepäck-Sortierung per Hand notwendig
- Geschätzte Kosten: 3.2 Milliarden US-Dollar

2.6. Zurück im Mittelalter: Der US-Stromausfall 2003

- Im August 2003 bleiben acht Staaten im Nordosten der USA für fünf Tage ohne Strom, insgesamt waren 50 Millionen Menschen betroffen
- Softwarefehler im Managementsystem zur Steuerung der Stromnetze
- Kombination von Ereignissen und Alarmen führt zum Ausfall des Systems, Backup-Server mit Anzahl der Fehlermeldungen überfordert
- Die Fehlermeldungen werden nicht an das Bedienpersonal weitergeleitet, eine Stunde lang bleibt der Kontrollbildschirm eingefroren
- Dadurch keine Abkopplung des schadhaften Stromnetzes von den anderen
- Dadurch Überlast in den Nachbar-Stromnetzen

2.7. Kommunikationsprobleme: Ausfall des AT&T-Telefonnetzes

- am 15.01.1990 können 70 von 138 Millionen Ferngesprächen innerhalb der USA 9 Stunden lang nicht vermittelt werden
- eine Schaltzentrale in New York versetzt sich nach Fehlfunktion in Reset-Modus
- Es erfolgt Ausfall-Meldung an alle anderen Zentralen, nach Reset erfolgt Meldung OK, werden Ferngespräche weiter vermittelt
- Bei drei Schaltzentralen kommen kurz nach der OK-Meldung neue Gespräche an, Daten werden zerstört, Rechnerausfall
- Schneeball-System folgt, 9 Stunden lang werden viele Zentralen lahmgelegt, 50% davon befinden sich im Reset-Modus und verschicken wieder OK-Meldung

- Notlösung: OK-Meldungen werden nicht mehr verschickt
- Ursache: break-Befehl von C wurde falsch umgesetzt
- Kosten: 75 Mill. US-Dollar bei AT&T
  mehrere 100 Mill, US-Dollar bei den Telefonkunden

3. Die Theorie

3.1. Top-down versus bottom-up-Strategie beim Testen von Modulen

einfaches Beispiel: Ermittlung von Tag, Monat und Jahr anhand des Systemdatums
Grundsätzlicher Aufbau des Programms

3.1.1. Bottom up

- Jedes Modul wird für sich alleine getestet
- Es werden Testprogramme, sogenannte Treiber oder Driver benötigt
- Nachteil, da das Programm als Ganzes nicht existiert
- Vorteile, wenn sich schwerwiegende Fehler in den Modulen am Boden der Struktur verbergen
- Beobachtung der Ergebnisse einfach

3.1.2. Top down

- zunächst wird oberstes Modul getestet, die anderen Module werden durch Platzhalter (stubs oder dummies) ersetzt
- nach und nach werden die anderen Module hinzugenommen (Incremental Testing)
- Nachteile, da stubs komplizierter zu programmieren sind als driver

- Vorteile, wenn sich schwerwiegende Fehler in der Spitze der Struktur verbergen
- Beobachtung der Ergebnisse schwieriger

3.2. Der White Box Test

- der innere Aufbau der Module und der Kontrollfluss ist bekannt
- Code liegt offen vor, wie gläserner Behälter
- Kriterien für den White Box Test ist die Ausführung möglichst jedes Programmpfades beim Testen

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 geteststt 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

Schwächen:

1. Intensives Prüfen beweist nicht, dass das Programm mit seiner Spezifikation übereinstimmt
2. Das Programm oder das Modul kann einige spezifizierte Funktionen nicht enthalten.
3. Fehler in den Daten werden häufig nicht aufgedeckt.

-> *externe Testgruppe benötigt*

3.3. Der Black Box Test

- Tester beurteilt Programmcode lediglich nach Funktion, die eigentliche Implementierung ist nicht bekannt
- Das detaillierte Lastenheft ist ein Leitfaden hierzu
- Keine Kriterien für die Testabdeckung wie beim White Box Test möglich
- Tester braucht viel Erfahrung und Geschick

Die wichtigsten Grundsätze:

1. *Ein Programmierer sollte nie versuchen, sein eigenes Programm zu testen*
   - eigenes Werk muss kritisch betrachtet werden
   - auch nicht bedachte Testfälle müssen durchgeführt werden
   - Programmierer kann Lastenheft falsch verstanden haben
2. *Das Testen von Software ist ein Experiment*
   - Entwurf der Testfälle ist kreative und herausfordernde Tätigkeit
   - Nie annehmen, dass doch keine Fehler gefunden werden
   - Ergebnis kann vom erwarteten Resultat abweichen

3. *Das Testen ist definiert als die Ausführung eines Programms mit der erklärten Absicht, Fehler zu finden*
    - Zahl der gefundenen Fehler ist der Indikator für die Güte des Tests

4. *Definition der erwarteten Ergebnisse vor dem Beginn des Tests*
    - Schwäche des Programms kann nicht als wünschenswerte Eigenschaft dargestellt werden

5. *Die Wahrscheinlichkeit, in einem bestimmten Segment des Programmcodes in der näheren Umgebung eines bekannten Fehlers weitere zu finden, ist überproportional hoch*
    - Fehler sind nicht zufällig verteilt
    - Bestimmte Module sind schwieriger als andere, Konzentration des Programmierers ist nicht konstant hoch

3.4. Der Gray Box Test

- Mischung aus White und Black Box Test
- Tester kennt in groben Zügen die Implementierung und kann Schwachpunkte in der Entwicklungsphase leichter erkennen, grösserer Überblick
- Vorteile beim Kompatibilitätstests, z.B. von Druckern

3.5. Die wichtigsten Testausprägungen

3.5.1. Der Funktionstest

- Funktionen der Software werden auf Modulebene getestet
- Schliesst Probleme an Schnittstellen nicht aus, Erkennung derselben aber durch fehlerfreie Module leichter

3.5.2. Der Volume Test

- die Grenzen der Belastbarkeit eines Programms werden ausgelotet
- Programmcode muss grosse Mengen von Daten bearbeiten
- So ausgelegt, dass normalerweise Grenzwerte in der Praxis nie erreicht werden

3.5.3. Der Stress-Test

- ähnlich wie Volume Test
- Programm wird innerhalb eines kurzen Zeitraums mit hoher Belastung ausgesetzt
- Gut zum Erkennen von Fehlern in Buslast, Puffern, etc.

3.5.4. Speicherverbrauch und Auslastung des Prozessors

- besonders wichtig bei Echtzeitsystemen
- sowohl Rechenzeitverbrauch als auch Speicherauslastung müssen Grenzwerte einhalten

3.5.5. Recovery Testing

- Ausfall eines Computers der eines Programms darf nicht zu Datenverlusten oder Leistungsminderung führen

3.5.6. Der Mutationstest

- die Testfälle werden einem Test unterzogen
- der Programmcode wird mutiert und die Testfälle für den unmutierten Programmcode angesetzt
- sind die Resultate unverändert, sind die Testfälle nicht ausreichend
- sind die Resultate verändert, wurden Testfälle gut gewählt

3.5.7. Benchmarks

- Überprüfung der Leistungsfähigkeit der Software
- Ausführungszeit/Verarbeitungsgeschwindigkeit wird gemessen

3.5.8. Der Test von Prozeduren und Verfahren

- Anwendungsfall wird auf mögliche Missverständnisse/Fehlerquellen und Effizienz geprüft

3.5.9. Configuration Testing

- Programm läuft in Entwicklungsumgebung einwandfrei, versagt aber am Markt
- Der durchschnittliche Benutzer verfügt nicht über einen ebenso leistungsstarken Computer
- Programm wurde in der speziellen Hardwarekonfiguration noch nie getestet und lässt sich nicht oder nur eingeschränkt verwenden

3.5.10. Usability Testing

- Programm/Website wird auf seine Bedienbarkeit hin überprüft
- Interaktion des Menschen mit dem technischen System
- Setzt den Fokus auf den Anwender/Kunden

3.5.11. Überprüfung von Dokumenten

- Dokumentation wird auf gültigen und vollständigen Inhalt hin untersucht

3.5.12. Der Systemtest

- Entwicklungsschritte werden vom Kunden abgenommen, Validation der Ergebnisse in regelmässigen Abständen
- Auch nach Abschluss der Entwicklung werden die Programme nach Kundenwunsch gewartet/erweitert

3.6. Test-Automation

- Bei jedem neuen Release muss Programm wieder getestet werden, Anpassung der alten Testfälle ist leichter als Neuerstellung
- Der Test kann ohne menschlichen Tester ablaufen, Zeitersparnis
- Es können mehrere Testläufe gleichzeitig ausgeführt und somit mehrere Benutzer simuliert werden
- Langwierige und ermüdende Tests lässt Mitarbeitern Zeit für die Erarbeitung kreativer Tests
- Testläufe lassen sich auf verschiedenen Rechnerkonfigurationen wiederholen