

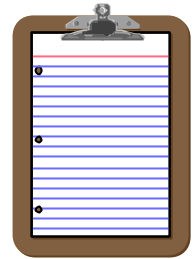
# **Organisation und Projektmanagement der IV**

## **06 Qualitätsmanagement; Stand: 2002-12-14**

**Prof. Dr. Dirk Stelzer**



**Technische Universität Ilmenau**  
Fakultät für Wirtschaftswissenschaften  
Fachgebiet Informationsmanagement



# Gliederung

- Begriffsklärung
- Qualitätsmerkmale für Software
- Produkt- und prozessorientierte Perspektive des Qualitätsmanagements
- Dynamische Qualitätssicherung: Testen
- Übung: Dreieck 3000
- Testgeneratoren und Testsysteme
- Statische Qualitätssicherung: Inspektionen
- Management der Softwareprüfung

# Lernziele



- Sie können den Begriff Qualität definieren.
- Sie kennen wesentliche Qualitätsmerkmale für Software
- Sie können einfache Softwaretests selber durchführen.
- Sie können die Prüfung komplexer Software planen.
- Sie kennen wesentliche Verfahren der statischen und der dynamischen Qualitätssicherung.
- Sie können eine Inspektion planen und steuern.

# Warum Qualitätsmanagement?

**Wenn etwas schief gehen kann,**

**dann wird es auch schief gehen ...**

**und zwar zum ungünstigsten Zeitpunkt.**

(frei nach Murphy)



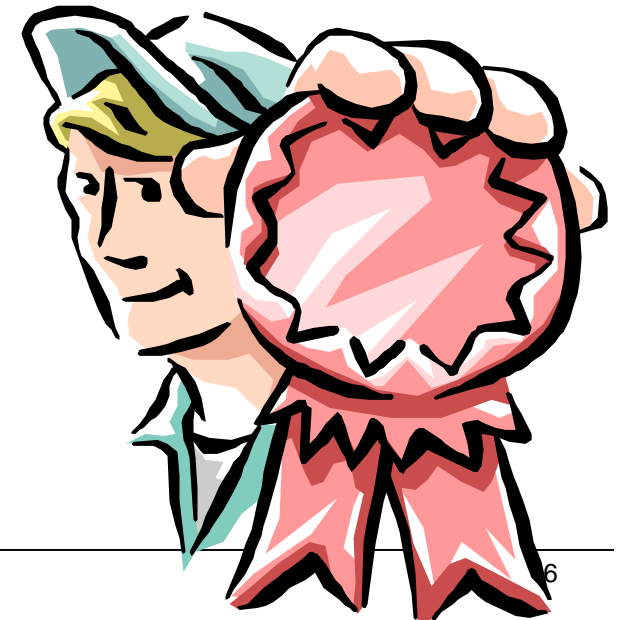


## Einige spektakuläre Fälle

- Venussonde fliegt 1979 am Ziel vorbei.
  - Ursache: Verwechslung eines Punktes mit einem Komma.
- Jagdbomber drehen sich bei Testflügen 1983 über Äquator auf den „Rücken“.
  - Ursache: Vorzeichenfehler.
- Salomon Brothers, Inc. erleidet 1992 einen erheblichen Verlust.
  - Ursachen:
    - » Angestellter erteilt einen Verkaufsauftrag über 11 Mio. Aktien, statt über Aktien im Wert von US\$ 11 Mio.
    - » Ein zweiter Angestellter versäumt, den Auftrag zu prüfen;
    - » das Informationssystem macht zu spät auf einen möglichen Fehler aufmerksam.

# Was bedeutet Qualität?

- Wann würden Sie ein Programm als qualitativ hochwertig bezeichnen?
- Von welchen Kriterien sind Ihre Überlegungen abhängig?
- Qualität als 'Güte' oder als Erfüllung von Anforderungen? (Crosby)
- Produktqualität versus Prozessqualität (ISO 9000, CMM, SPICE, ...)



# Auszug aus einem fiktiven Ausschreibungstext

**„Das zu erstellende System sollte benutzerfreundlich und hochperformant sein. Der Auftragnehmer verpflichtet sich, Entwicklungsmethoden einzusetzen, die dem state of the art der Anwendungsentwicklung entsprechen.“**



# Der Begriff 'Qualität'

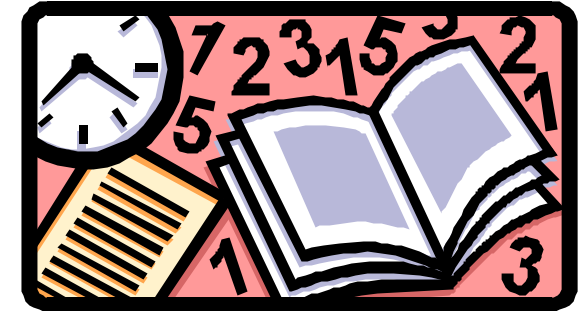
- Abgeleitet aus dem lateinischen Wort 'qualitas': Beschaffenheit, Eigenschaft; Eignung eines Gutes für bestimmte Verwendungszwecke (Zweckeignung).
- Gesamtheit von Merkmalen einer Einheit bezüglich ihrer Eignung, festgelegte und vorausgesetzte Erfordernisse zu erfüllen.

## Software-Produkt-Qualität:

- Gesamtheit von Eigenschaften eines Software-Produkts, die sich auf die Eignung zur Erfüllung gegebener Erfordernisse beziehen.  
(Schmitz, Bons, van Megen /Software-Qualitätssicherung/ 14; in Anlehnung an DIN 55 350)

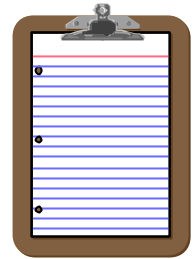


# Operationalisierung der Qualität

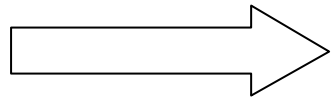


- Qualität (Gesamtheit unterschiedlicher Qualitätsmerkmale)
- Qualitätsmerkmal (ein Aspekt der Qualität)
- Qualitätsmaß / -kennzahl (metrics)  
(objektive, messbare Größe, die in Bezug auf unterschiedliche Ausprägungen eines oder mehrerer Qualitätsmerkmale sensitiv reagiert)
  - Qualitätsziel (angestrebter Wert eines Qualitätsmaßes)
  - Qualitätsmaßzahl (erreichter Wert eines Qualitätsmaßes)

(in Anlehnung an Schmitz, Bons, van Megen /Software-Qualitätssicherung/ 20)



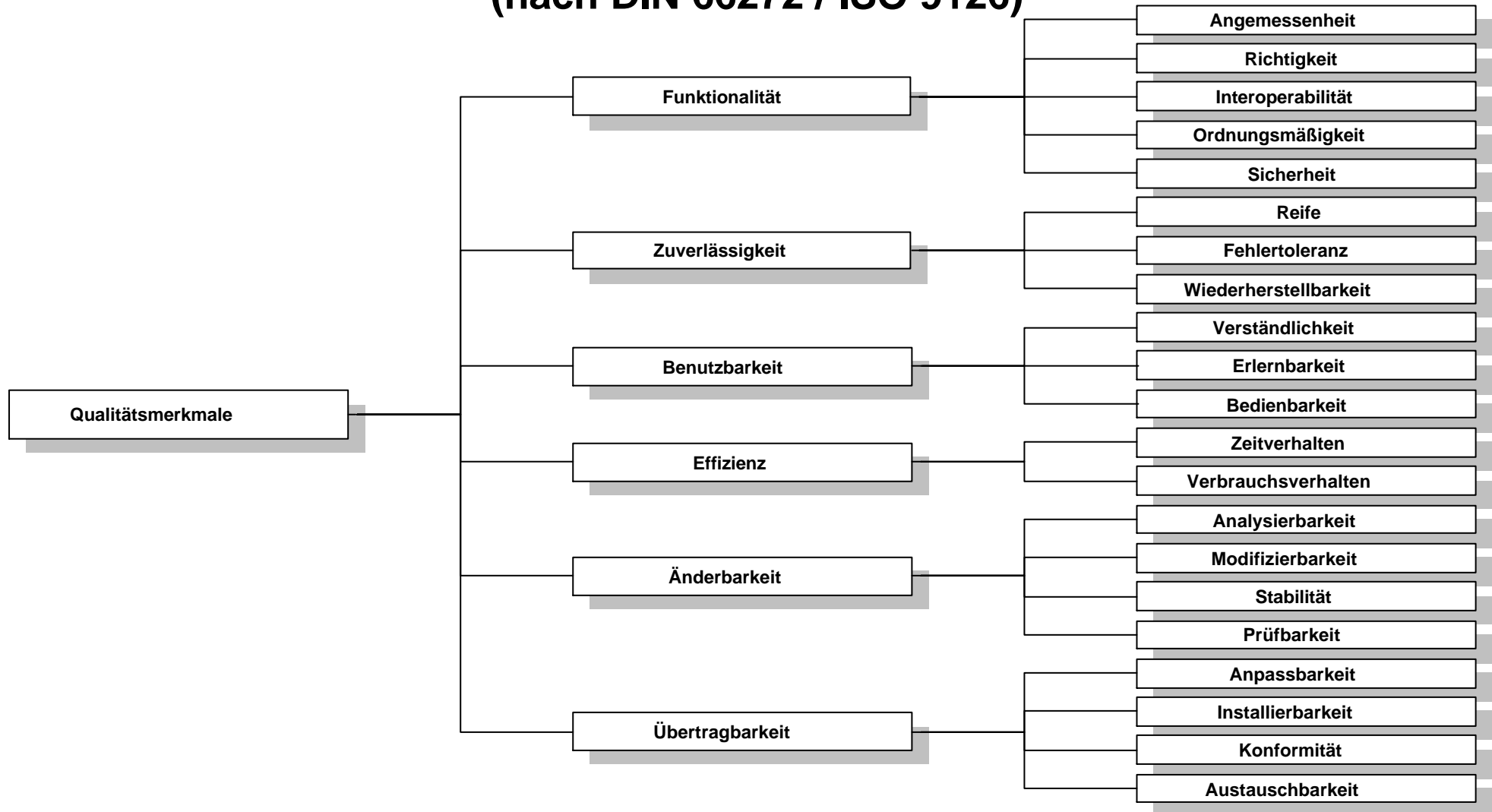
# Gliederung



- Begriffsklärung
- Qualitätsmerkmale für Software
- Produkt- und prozessorientierte Perspektive des Qualitätsmanagements
- Dynamische Qualitätssicherung: Testen
- Übung: Dreieck 3000
- Testgeneratoren und Testsysteme
- Statische Qualitätssicherung: Inspektionen
- Management der Softwareprüfung

# Qualitätsmodell für Software

## (nach DIN 66272 / ISO 9126)



# Qualitätsmerkmale für Software (1)

(nach DIN 66272 / ISO 9126)

**Softwarequalität kann anhand verschiedener Merkmalen bewertet werden:**

- **Funktionalität**  
... Vorhandensein einer Menge von Funktionen,  
die festgelegte oder vorausgesetzte Erfordernisse erfüllen
- **Zuverlässigkeit**  
... Fähigkeit der Software, ihr Leistungsniveau unter festgelegten  
Bedingungen über einen festgelegten Zeitraum zu bewahren
- **Benutzbarkeit**  
... Aufwand, der zur Benutzung der Software erforderlich ist,  
sowie die individuelle Bewertung einer solchen Benutzung durch eine  
festgelegte oder vorausgesetzte Gruppe von Benutzern

# Qualitätsmerkmale für Software (2)

(nach DIN 66272 / ISO 9126)

- **Effizienz**  
... Verhältnis zwischen dem Leistungsniveau der Software und dem Umfang der eingesetzten Betriebsmittel unter festgelegten Bedingungen
- **Änderbarkeit**  
... Aufwand, der zur Durchführung vorgegebener Änderungen notwendig ist
- **Übertragbarkeit**  
... Eignung der Software, von einer Umgebung in eine andere übertragen zu werden

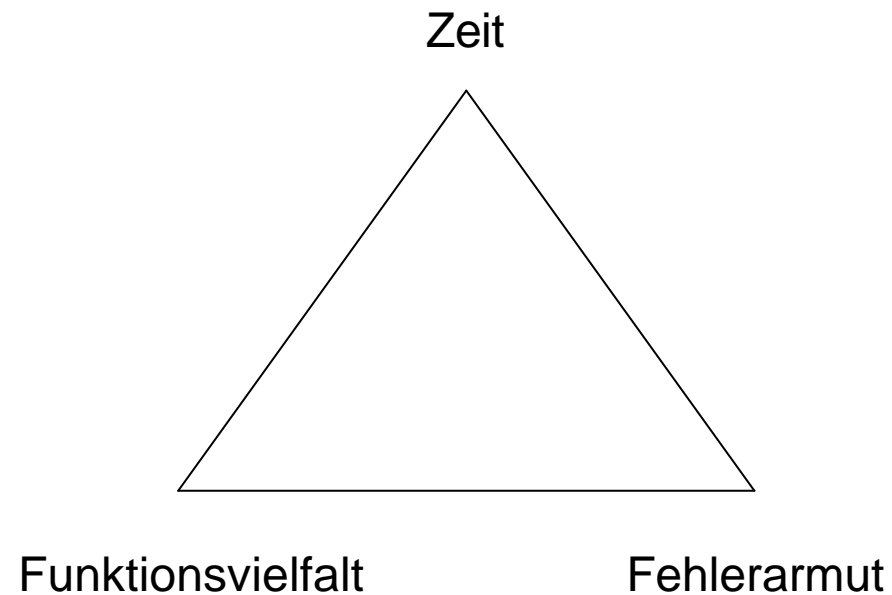
# Qualität - aus verschiedenen Perspektiven betrachtet

**Tester bei Microsoft werden angehalten,  
die Produkte aus folgenden Perspektiven zu prüfen:**

- **Perspektive des Benutzers:** Wie verwendet ein gewöhnlicher Benutzer das Testobjekt?
- **Internationale Perspektive:** Sind Formate und Sprachen für die verschiedenen internationalen Märkte korrekt implementiert?
- **Hardware:** Ist das Testobjekt mit relevanten Hardware-Plattformen kompatibel?
- **Software:** Ist das Testobjekt mit relevanten Software-Paketen kompatibel?
- **Spezifikation:** Ist das Testobjekt mit der ursprünglichen Spezifikation kompatibel?
- **Stabilität:** “Are we ready to ship the product?”

Michael A. Cusumano, Richard W. Selby: Microsoft Secrets: How the world's most powerful software company creates technology, shapes markets, and manages people. New York 1995, S. 86

# “Good Enough” Software

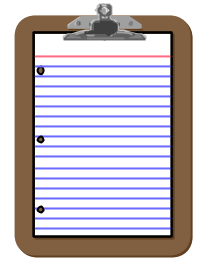


(nach Yourdon /"Good Enough" Software/)

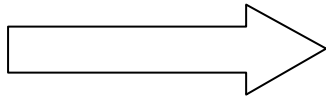
# Qualitätsmanagement

- bezeichnet die Planung, Steuerung und Kontrolle von Qualität.
- „Qualität“:  
Ausmaß, in dem ein Gegenstand geeignet ist, Anforderungen zu erfüllen.
- „Gegenstände“ des Qualitätsmanagements können
  - Produkte,
  - Prozesse oder auch
  - Organisationen sein.





# Gliederung



- Begriffsklärung
- Qualitätsmerkmale für Software
- **Produkt- und prozessorientierte Perspektive des Qualitätsmanagements**
- Dynamische Qualitätssicherung: Testen
- Übung: Dreieck 3000
- Testgeneratoren und Testsysteme
- Statische Qualitätssicherung: Inspektionen
- Management der Softwareprüfung

# Zwei verschiedene Perspektiven des QM

## Qualitätsmanagement hat zum Ziel, dass ...

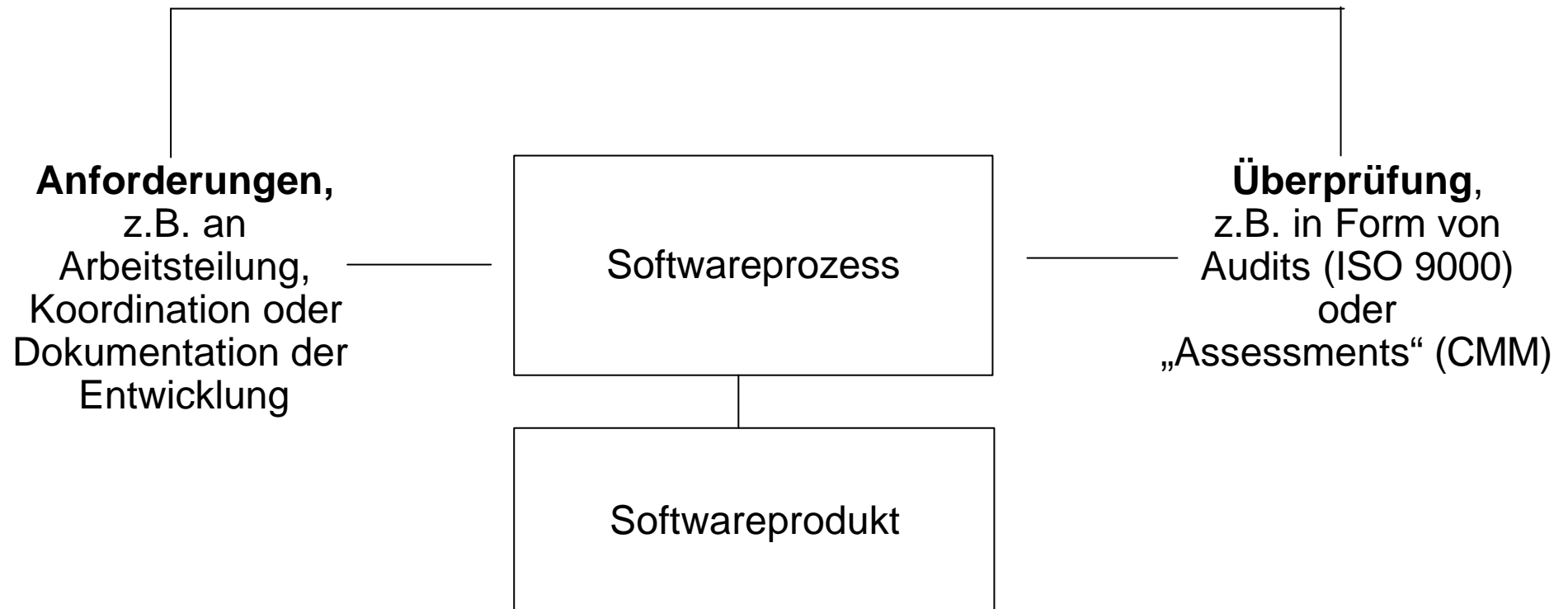
- das entwickelte Softwareprodukt die Anforderungen erfüllt, die an dieses Produkt gestellt werden
- der Softwareprozess die Anforderungen erfüllt, die an diesen Prozess gestellt werden
  - nicht nur Produktqualität, sondern auch
  - (Entwicklungs-)Kosten
  - Zeit(dauer der Entwicklung)

# Produktorientierte Perspektive

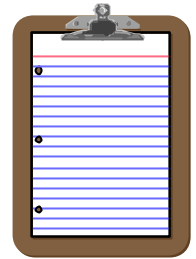


Im Rahmen der Qualitätssicherung wird überprüft, inwiefern das entwickelte **Produkt** diese Anforderungen erfüllt.

# Prozessorientierte Perspektive

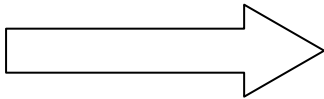


**Annahme: ein qualitativ hochwertiger **Prozess** (das heisst ein Prozess, der die an ihn gestellten Anforderungen erfüllt) bringt mit einer hohen Wahrscheinlichkeit auch ein qualitativ hochwertiges Produkt hervor.**

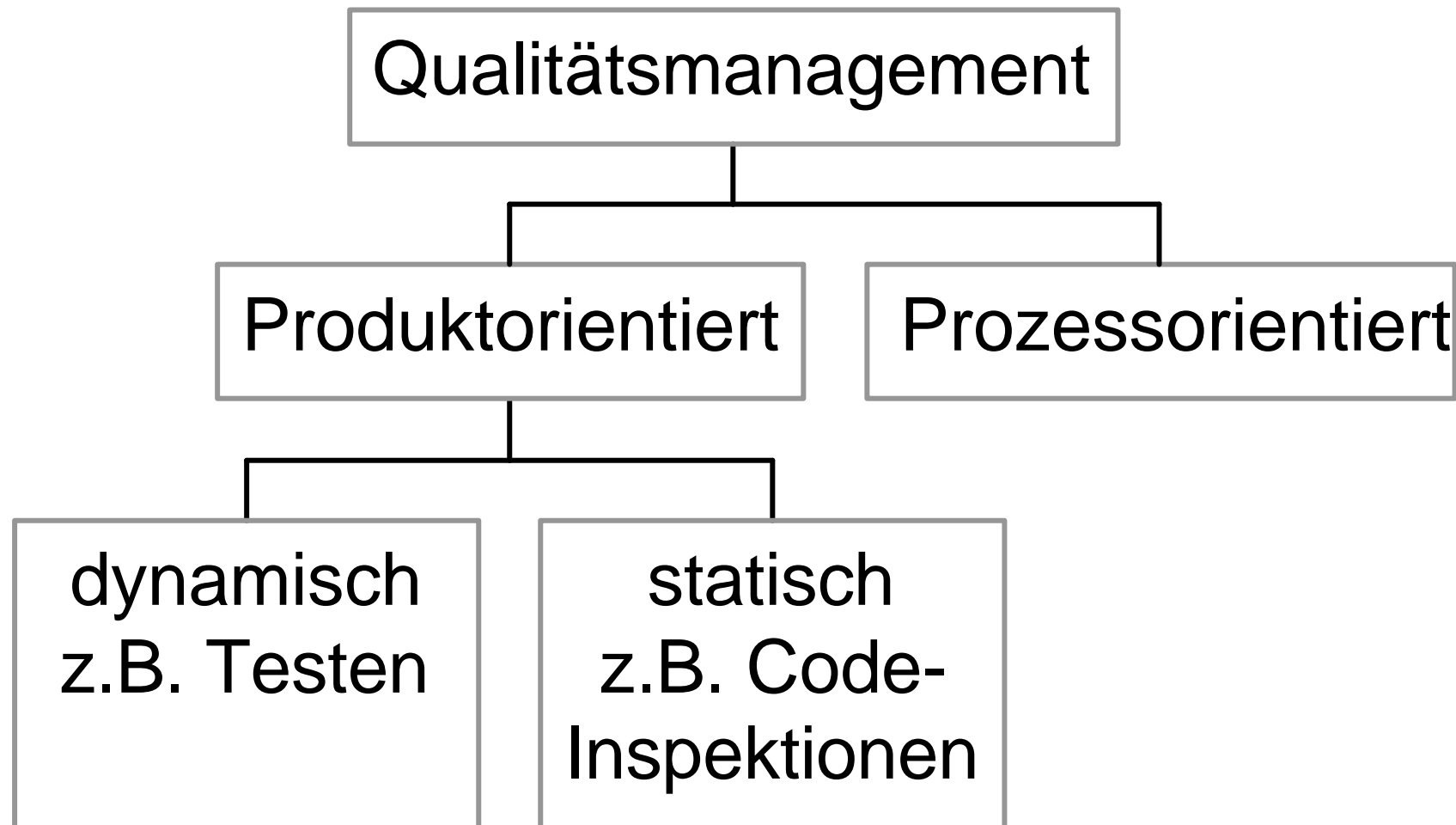


# Gliederung

- Begriffsklärung
- Qualitätsmerkmale für Software
- Produkt- und prozessorientierte Perspektive des Qualitätsmanagements
- **Dynamische Qualitätssicherung: Testen**
- Übung: Dreieck 3000
- Testgeneratoren und Testsysteme
- Statische Qualitätssicherung: Inspektionen
- Management der Softwareprüfung



# Übersicht über die verschiedenen Perspektiven des QM



# Dynamische und statische Qualitätssicherung

## Dynamische Qualitätssicherung

- Prüfen von Software durch Ausführung (= Testen)
- Bsp: Funktionales Testen



## Statische Qualitätssicherung

- Prüfen von (Zwischen)Ergebnissen der Softwareentwicklung, ohne diese auszuführen
- kann sich auf alle Produkte der Softwareentwicklung (Anforderungsspezifikation, Datenmodelle, Quellcode, ...) beziehen
- Bsp: Code-Inspektionen

# Diskutieren Sie folgende Definitionen des Begriffs Softwarefehler

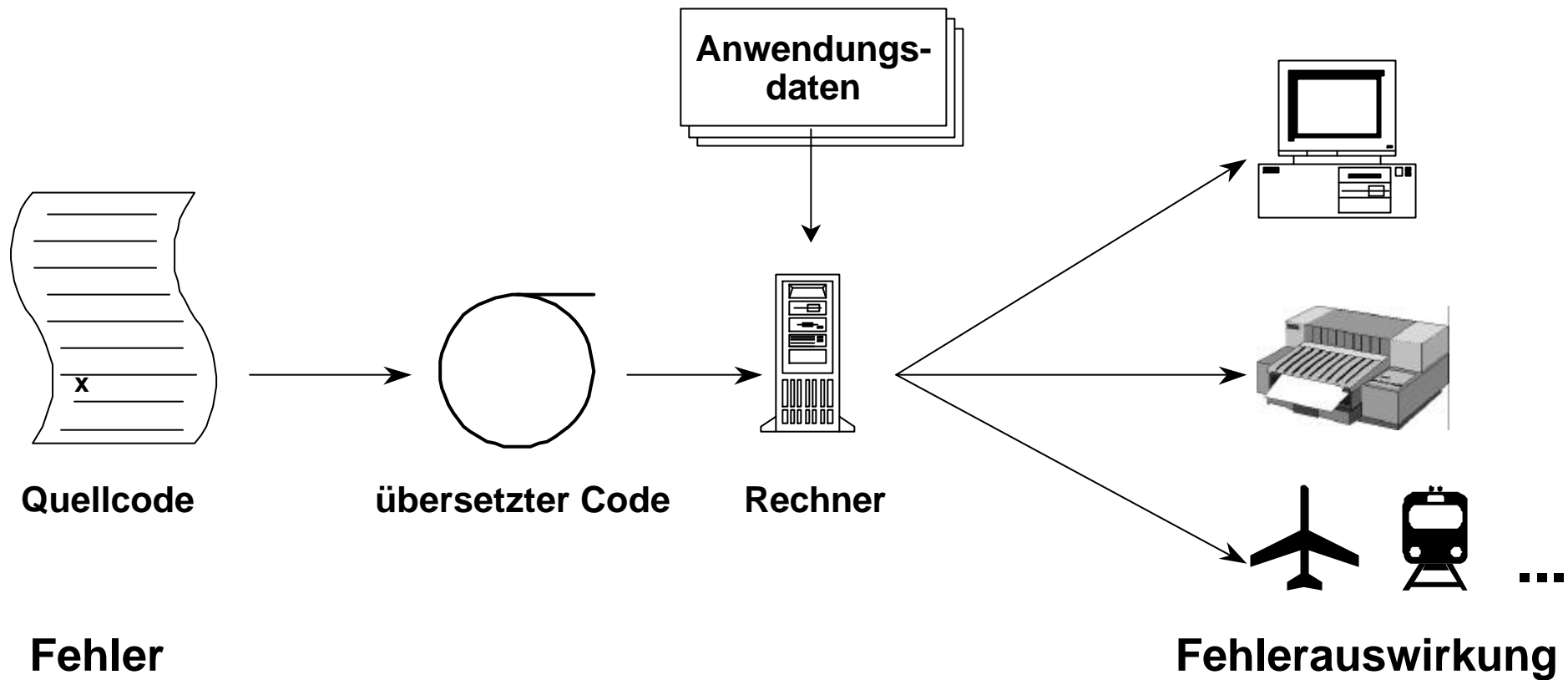
- Ein Fehler ist jede Inkonsistenz der Implementation zur Spezifikation.  
(Liggesmeyer /Modultest/ 31)
- Ein Fehler ist ein strukturelles Merkmal, das ein fehlerhaftes Verhalten des Programms verursacht. (Liggesmeyer /Modultest/ 32)
- Nichterfüllung einer Forderung (DIN /DIN 55 350, Teil 11/)
- Jede Abweichung der tatsächlichen von der vorgesehenen bzw. explizit geforderten Ausprägung einer Eigenschaft eines Produktes der Softwareentwicklung  
(nach Liggesmeyer /Modultest/ 31 ff.)



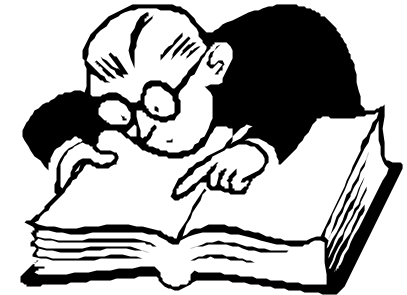


## Inhalt und Struktur der Software

## Verhalten der Software



# Prüfen und Testen



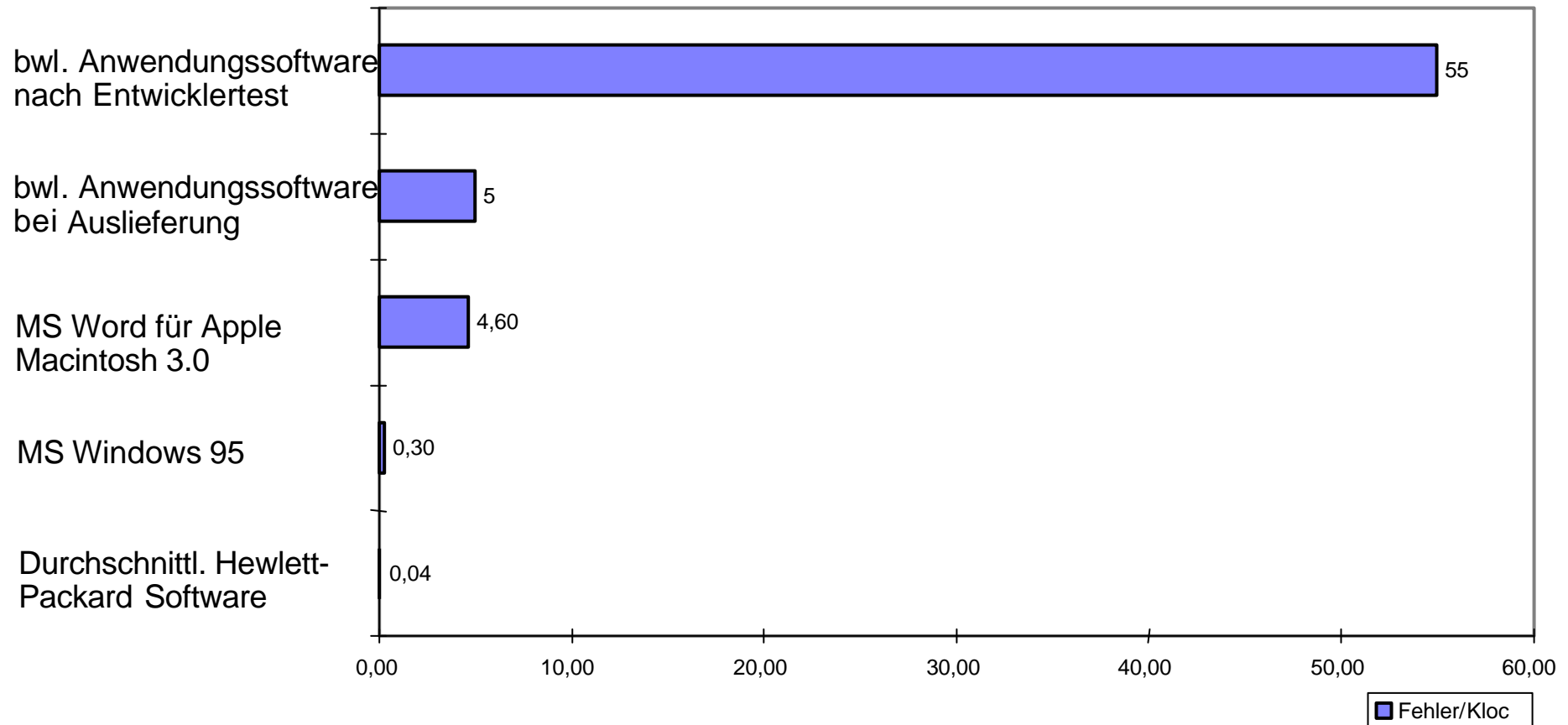
## Prüfen

- Analyse eines Gegenstandes (durch Vergleich mit den vorgesehenen bzw. geforderten Eigenschaften), mit dem Ziel, Fehler zu finden.
- Oberbegriff für verschiedene Methoden zur Aufdeckung von Fehlern in Produkten der Softwareentwicklung
- umfasst statische und dynamische QS-Maßnahmen

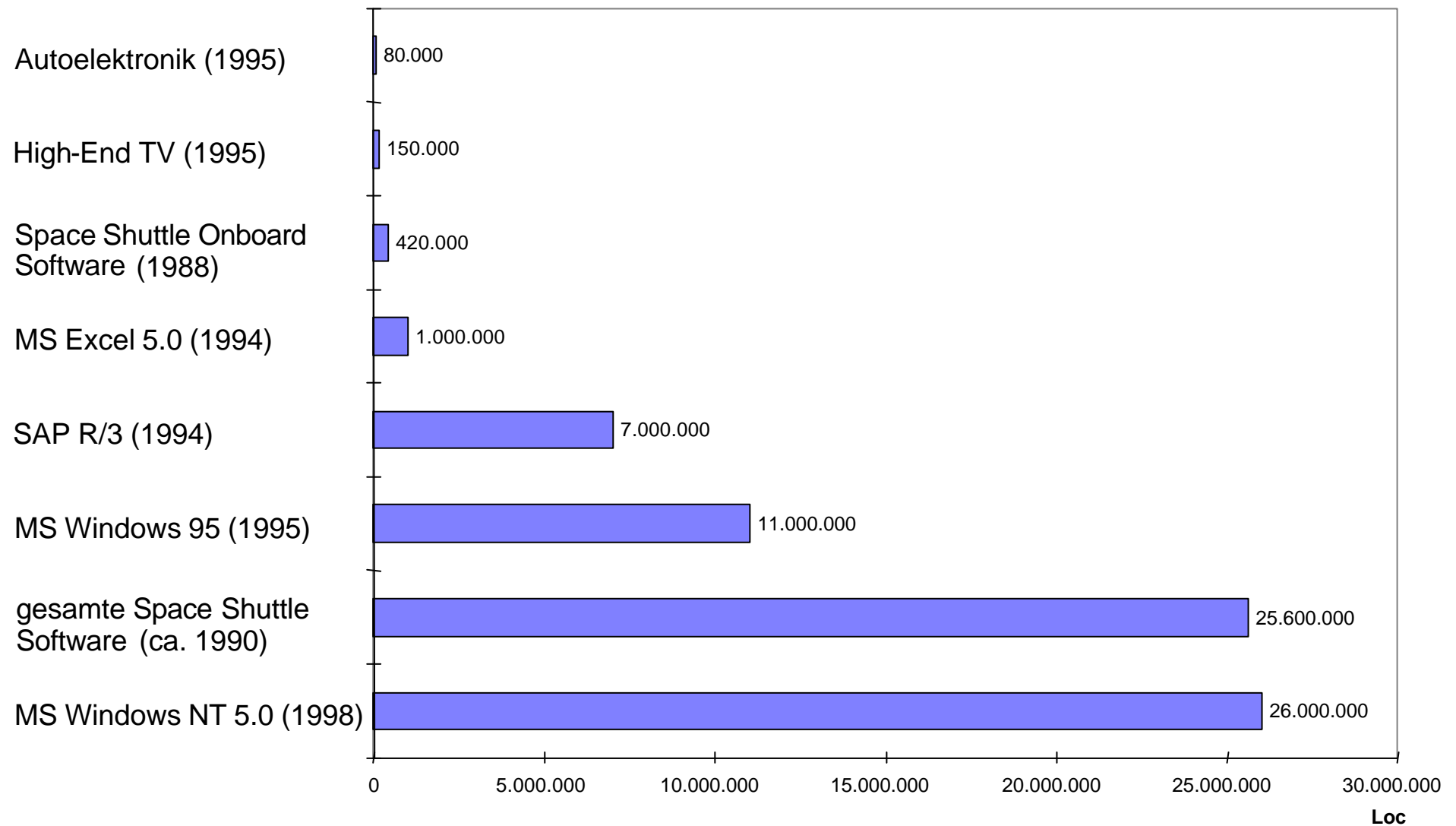
## Testen (Spezialfall des Prüfens)

- bedeutet, ein Programm mit der Absicht auszuführen, Fehler zu finden.
- Dynamische Qualitätssicherung

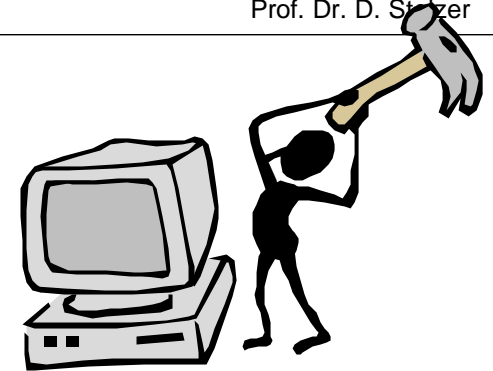
# Beispiele für den Fehleranteil in Software



## Beispiele für den Umfang von Software



# Fehlerkategorien



- Kategorie 1 (Softwareabsturz):** • Fehler erfordert Neustart der Software.
- Kategorie 2 (Funktionsversagen):** • Fehler verhindert Ausführen einer Funktion oder führt zu falschem Ergebnis; eine alternative Lösung ist nicht möglich.
- Kategorie 3:** • wie Kategorie 2, aber eine alternative Lösung ist möglich.
- Kategorie 4 (Schönheitsfehler):** • Fehler hat untergeordnete Bedeutung.

## Auftreten von Fehlersymptomen

**davon Neustart des Rechners  
nötig in % aller Fälle**

- UNIX: ca. 1/Jahr (alle 96.000 Minuten) • 25 %
- Mac/ OS: ca. 510/Jahr (alle 188 Minuten) • 56 %
- MS Office  
unter Win95: ca. 2285/Jahr (alle 42 Minuten) • 28 %

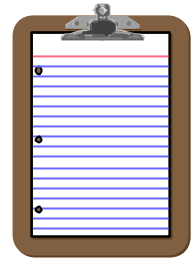


(Quelle:Litzba/Testing Stand 1996; Bei 200 Nutzungstagen/Jahr und 8 Stunden Betriebsdauer/Tag)

# Testen

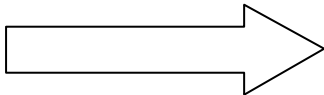


- **Testen:**  
Überprüfen von Software durch Ausführen eines Testobjektes (z. B. eines Softwaremoduls) mit Testdaten.
- **Ziel des Testens:** Fehler finden
- **Fehler:** Abweichungen von den Anforderungen
- **Achtung:**  
Softwaretests sind nicht geeignet, die Sicherheit, Korrektheit oder Qualität eines Softwareproduktes nachzuweisen.  
Softwaretests können lediglich Abweichungen von den Anforderungen (= Fehler) aufzeigen.



# Gliederung

- Begriffsklärung
- Qualitätsmerkmale für Software
- Produkt- und prozessorientierte Perspektive des Qualitätsmanagements
- Dynamische Qualitätssicherung: Testen
- Übung: Dreieck 3000
- Testgeneratoren und Testsysteme
- Statische Qualitätssicherung: Inspektionen
- Management der Softwareprüfung





# Spezifikation Dreieck 3000

Das Programm Dreieck 3000 liest drei ganzzahlige Werte ein.  
Die drei Zahlen werden als Längen von Dreiecksseiten interpretiert.  
Dreieck 3000 gibt daraufhin eine Meldung aus,  
ob das Dreieck ungleichseitig, gleichschenkelig oder gleichseitig ist.

## Aufgaben

- Ermitteln Sie auf der Basis der Spezifikation Testfälle, mit denen Sie überprüfen können, ob Dreieck 3000 einen Fehler enthält. Geben Sie für jeden Testfall eine Begründung. (Was wollen Sie mit dem Testfall erreichen?)
- Machen Sie Annahmen über mögliche Entwicklungsfehler (und über unterschiedliche Realisierungsalternativen des Programms Dreieck 3000) und ergänzen Sie gegebenenfalls Ihre Testfälle.
- Leiten Sie aus den Testfällen Testdaten(kombinationen) ab.

# Grundstruktur eines Softwaretests

## Definition von Testfällen

Ein Testfall beschreibt eine Menge von Eingabedaten, mit denen ein bestimmter Aspekt des Verhaltens eines Testobjekts geprüft werden soll.

## Auswahl einer Testdatenkombination (TDK)

Testdaten sind die Eingabewerte, die bei der Testdurchführung verwendet werden.

Eine Testdatenkombination ist eine Kombination von Testdaten, die gemeinsam für eine Ausführung des Testobjekts verwendet werden.

## Definition des erwarteten Ergebnisses

## Ausführung des Testobjekts mit der TDK

Ein Testverfahren bezeichnet eine begründete Vorgehensweise zur Aufdeckung einer bestimmten Klasse von Fehlern.

Unterschiedliche Testverfahren führen zur Auswahl unterschiedlicher Testdaten bzw. Testdatenkombinationen.

## Vergleich des erwarteten mit dem tatsächlichen Ergebnis

## Testdokumentation für Dreieck 3000

| <b>Testfall</b>                                   | <b>Testdaten</b>   | <b>erwartetes Ergebnis</b>  | <b>Testergebnis</b>   |
|---|--|---|---|
| Welche Eigenschaft des Programms will ich testen? | Welche Daten will ich zur Überprüfung der Eigenschaft verwenden? | Zu welchem Ergebnis soll die Ausführung des Programms mit den Testdaten führen? | Zu welchem Ergebnis hat die Ausführung des Programms mit den Testdaten tatsächlich geführt? |

# Kategorisierung von Testverfahren

## Funktionale Testverfahren

(Testfälle werden aus der Spezifikation abgeleitet.)

- Intuitive Testfallermittlung
- Funktionsabdeckung
- Äquivalenzklassen-Analyse
- Grenzwertanalyse
- Ursache-Wirkungs-Analyse

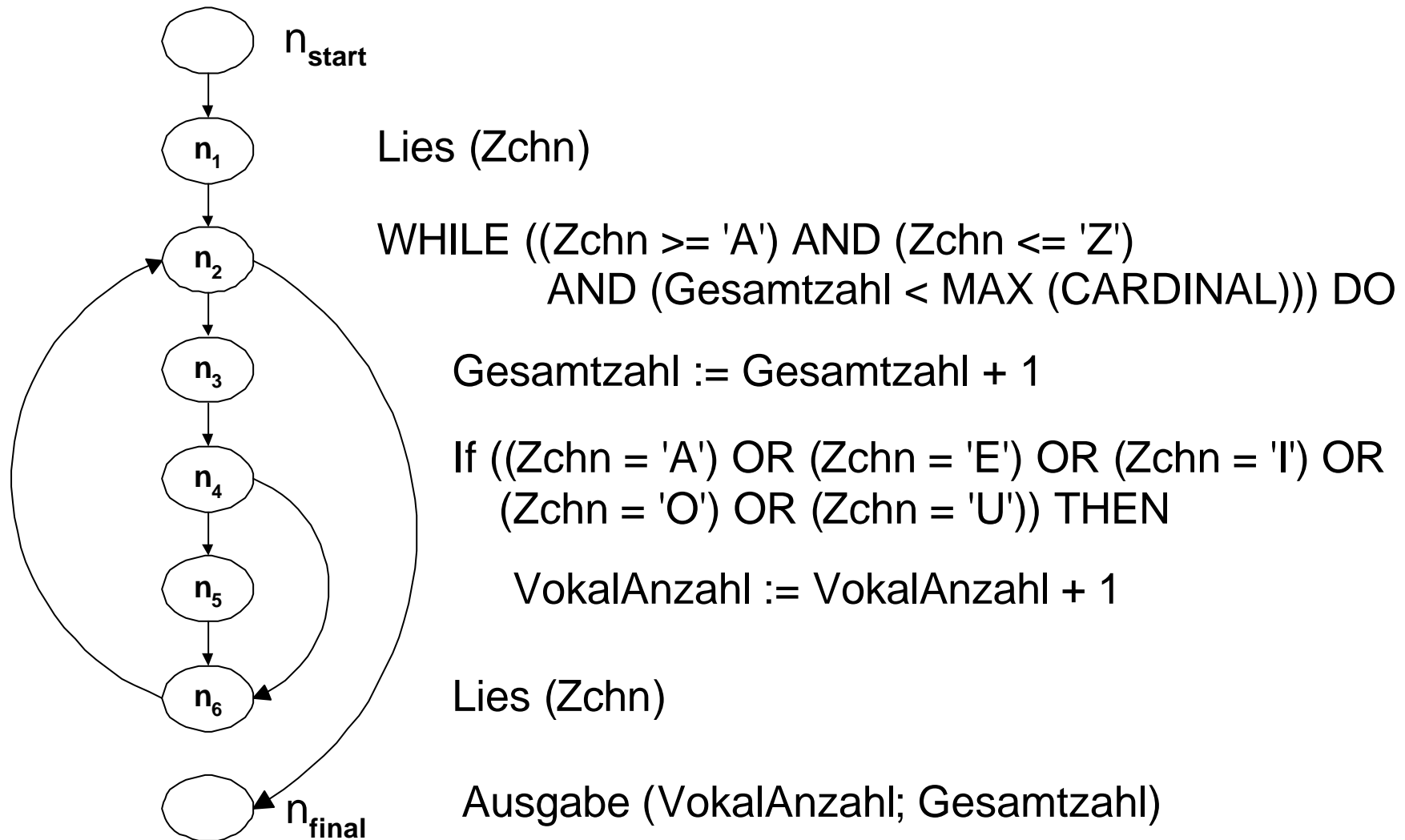


## Strukturelle Testverfahren

(Testfälle werden aus der Struktur des Quellprogramms abgeleitet.)

- kontrollflussorientierte Testverfahren  
zur Erzeugung von Testfällen werden Strukturelemente (z. B. Anweisungen, Zweige, Bedingungen) verwendet
- datenflussorientierte Testverfahren  
(zur Erzeugung von Testfällen werden Zugriffe auf Variablen verwendet)

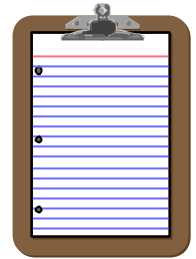
# Kontrollflussgraph ZaehleZchn



(Quelle: Liggesmeyer /Modultest/ 42)

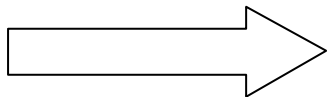
# Abdeckungskenngrößen kontrollflussorientierter Testverfahren

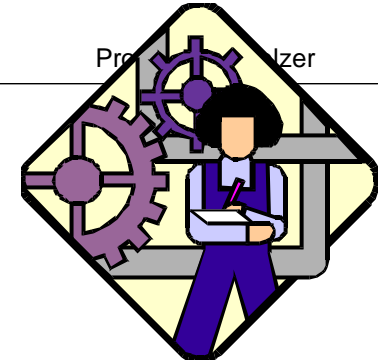
- C0 Anweisungsabdeckung:  
Verhältnis von Anzahl der mit Testdaten durchlaufenen **Anweisungen** zur Gesamtanzahl der Anweisungen
- C1 Zweig- / Entscheidungsabdeckung:  
Verhältnis von Anzahl der mit Testdaten durchlaufenen **Zweige** zur Gesamtanzahl der Zweige
- C2 Bedingungsabdeckung:  
Verhältnis von Anzahl der mit Testdaten durchlaufenen **Prädikate** (Terme innerhalb von Entscheidungen) zur Gesamtanzahl der Prädikate
- C3 Abdeckung aller **Bedingungskombinationen**:  
Verhältnis von Anzahl der mit Testdaten durchlaufenen Bedingungskombinationen zur Gesamtanzahl der Bedingungskombinationen
- C4 Pfadabdeckung:  
Verhältnis von Anzahl der mit Testdaten durchlaufenen Pfade zur Gesamtanzahl der **Pfade**



# Gliederung

- Begriffsklärung
- Qualitätsmerkmale für Software
- Produkt- und prozessorientierte Perspektive des Qualitätsmanagements
- Dynamische Qualitätssicherung: Testen
- Übung: Dreieck 3000
- **Testgeneratoren und Testsysteme**
- Statische Qualitätssicherung: Inspektionen
- Management der Softwareprüfung





# Testgeneratoren und Testsysteme (1/3)

sind Werkzeuge zur Unterstützung und (teilweisen) Automatisierung des Testens.

- *Vorbereitung* des Testens: Testfallerstellung und Testdatengenerierung
  - Testfallgeneratoren, die auf der Grundlage einer formalen Spezifikation automatisch Testfälle ableiten
  - Unterstützung von speziellen Testverfahren (z.B. Ursache-Wirkungs-Analyse, Grenzwertmethode oder Äquivalenzklassenanalyse)
  - Testfallermittlung bei kontrollflussorientierten Testverfahren: Werkzeuge, die Kontrollflüsse im Programm erkennen, graphisch anzeigen, nicht abgedeckte Strukturelemente aufdecken und Vorschläge für zusätzliche Testfälle ableiten.
  - Testdatenerstellung: Werkzeuge, die automatisch Testdaten aus Testfällen ableiten.



## Testgeneratoren und Testsysteme (2/3)

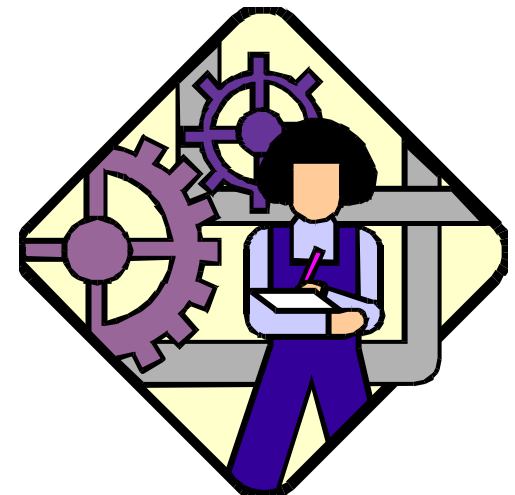
*Durchführung* des Testens:

Ausführung eines Testobjektes mit Testdaten und  
Vergleich der Ausgaben mit den erwarteten Ergebnissen

- Werkzeuge zur Erstellung von Platzhaltern und Treibern
- Capture-Replay-Werkzeuge
- Werkzeuge zur Abdeckungsmessung

*Auswertung* von Testaktivitäten

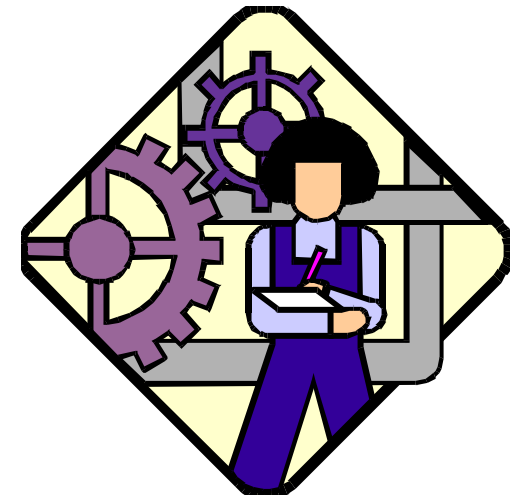
- z.B. Vergleich von Soll- und Ist-Ergebnissen

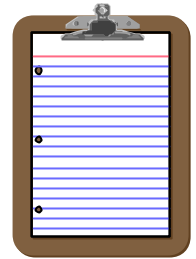


# Testgeneratoren und Testsysteme (3/3)

## *Organisation von Tests*

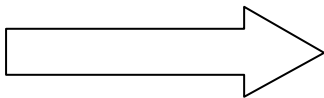
- Werkzeuge zur Verwaltung der Testfälle, Testdaten, Testskripte, Platzhalter und Treiber sowie der Ergebnisse
- Fehlerverfolgung: Verfolgen des Status eines gefundenen Fehlers (z.B. Fehler ausgewertet, lokalisiert, behoben, Modul erneut getestet)



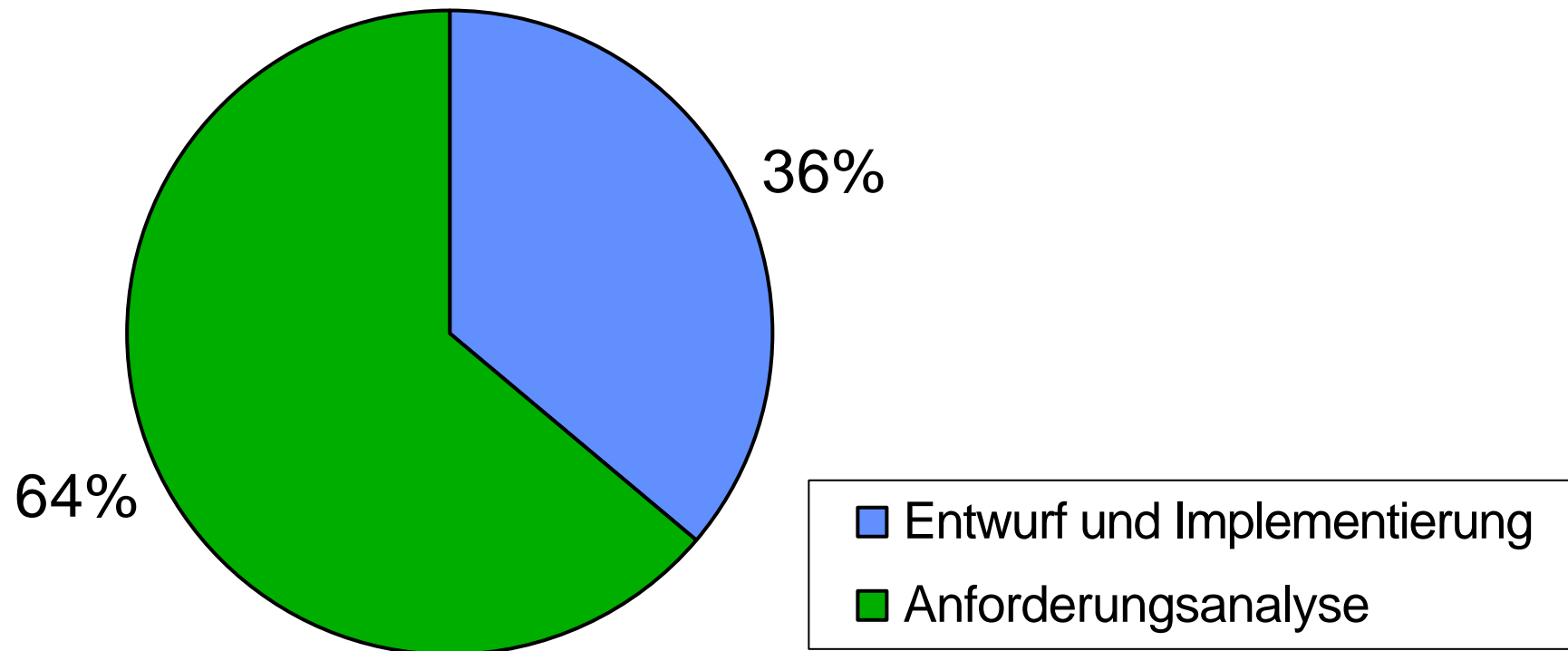


# Gliederung

- Begriffsklärung
- Qualitätsmerkmale für Software
- Produkt- und prozessorientierte Perspektive des Qualitätsmanagements
- Dynamische Qualitätssicherung: Testen
- Übung: Dreieck 3000
- Testgeneratoren und Testsysteme
- **Statische Qualitätssicherung: Inspektionen**
- Management der Softwareprüfung

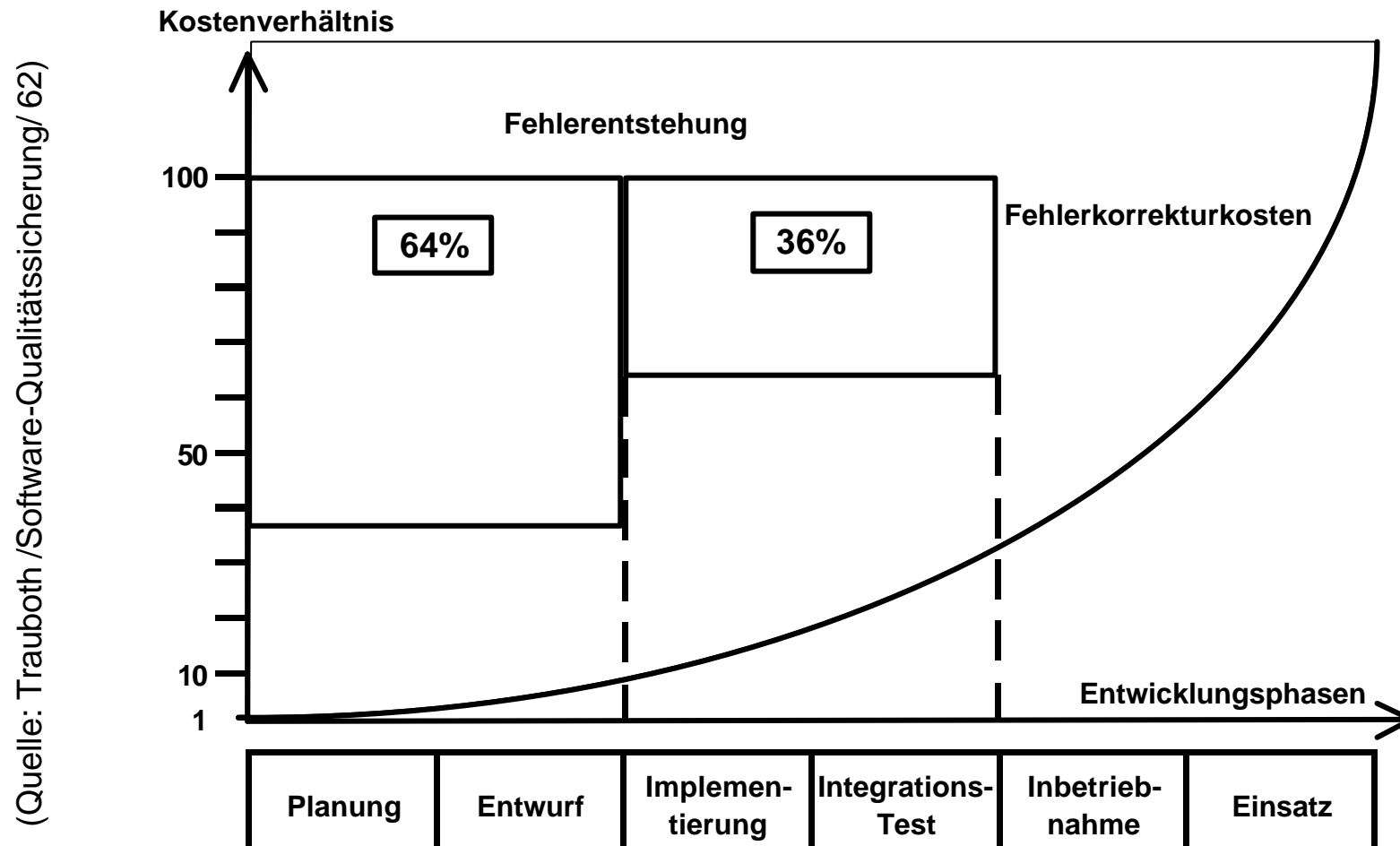


## In welchen Phasen der Softwareentwicklung entstehen wie viel Prozent der Fehler?



(Erfahrungswerte aus verschiedenen Softwareentwicklungsprojekten  
Quellen: Maus /Testmanagement/ 8; Trauboth /Software-Qualitätssicherung/ 62)

# Fehlerentstehung, -entdeckung und -behebungsaufwand



# Statische Qualitätssicherung

- ... umfasst alle Aktivitäten, die Informationen über ein Prüfobjekt bereitstellen, ohne es dynamisch auszuführen
- Beispiele
  - Inspektionen
  - Walkthroughs
- Prüfobjekte können z. B. Anforderungsdokumente, Datenmodelle oder Softwarecode sein.



## Beispiel: Inspektionen

- Ein Team versucht, durch gemeinsames Lesen des Prüfobjekts und mit Hilfe von Checklisten Fehler zu entdecken
- Ziel: Fehlerentdeckung, nicht Fehlerbehebung



# Rollen / Aufgaben der Teilnehmer einer Inspektion

- Moderator
  - verantwortlich für den korrekten Ablauf
  - leitet, koordiniert und steuert die Sitzungen
  - benötigt kein detailliertes Wissen über das Prüfobjekt
- Autor / Entwickler erläutert das Prüfobjekt
- Zwei bis fünf 'Inspektoren' sind für das Auffinden der Fehler verantwortlich:
  - Anwender / Endbenutzer
  - Tester
  - Wartung, Installation, Systemunterstützung







# Ablauf einer Inspektion nach Fagan

(Rollen bei Fagan-Inspektionen: Moderator, Designer, Programmierer, Tester)

- Planung (Ziel: Voraussetzungen für Inspektion schaffen)
- Überblick (Ziel: Informierung der Teilnehmer)
- Individuelle Vorbereitung (Ziel: Verständnisvermittlung)
- Inspektions-Sitzung (Ziel: Fehler finden)
- Fehlerbehebung / Überarbeitung (Ziel: Fehler beheben)
- Nacharbeit  
(Ziel 1 : Sicherstellen, dass alle Fehler korrekt behoben worden sind  
Ziel 2 : prozessverbesserung)

(Fagan /Inspections/ und Fagan /Design and Code Inspections/)



# Die Inspektionssitzung

- Dauer: ca. 90 bis 120 Minuten
- Wie umfangreich sind die Prüfobjekte?
  - Myers: 150 Anweisungen pro Stunde
  - Trauboth: 50-200 loc in 2 Stunden
  - Fagan: 125 loc pro Stunde
- Ziele:
  - Qualität des *Produktes* verbessern  
Nicht: Überprüfung der Leistungsfähigkeit des *Programmierers*
  - Fehlerentdeckung  
Nicht: Diskussion von Fehlerbehebungsmöglichkeiten

# Beispiele für Prüfkriterien

- Stimmt das Objekt mit der Spezifikation überein?
- Ist das Objekt vollständig?
- Sind alle Funktionen korrekt implementiert?
- Enthält das Objekt nur die Funktionen / Inhalte, die es enthalten soll?
- Sind relevante Richtlinien, Normen und Standards eingehalten worden?
- Sind Korrekturen aus vorhergehenden Prüfungen korrekt implementiert?
- Kann das Produkt im verbleibenden Zeit- und Kostenrahmen fertiggestellt werden?



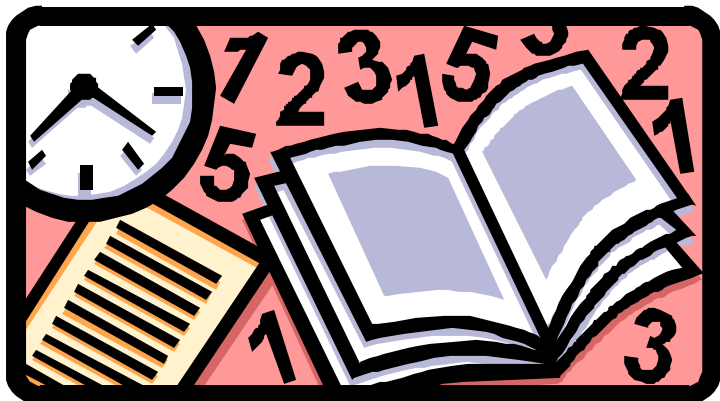
## Einige Vorurteile

- “Inspektionen bringen nichts.”
- “Inspektionen dauern zu lange.”
- “Inspektionen sind zu teuer.”
- “Wer ordentlich testet kann sich Inspektionen sparen.”
- “Unsere Entwickler kennen ihre Programme am besten.  
Eine Inspektion durch andere Mitarbeiter wäre zu aufwendig.”



# Maße für die Effizienz von Inspektionen

- gefundene Fehler / Gesamtaufwand für Inspektionen
- error detection efficiency:  
errors found by an inspection /  
total errors in the product before inspection \* 100



# **Testen oder Inspektionen?**

## **Einige Erfahrungswerte**

### **Vergleich, ohne Fehlerbehebung zu berücksichtigen**

- Test: ca. 0,2 gefundene Fehler pro Personenstunde Gesamtaufwand
- Inspektion: ca. 1 gefundener Fehler pro Personenstunde Gesamtaufwand

Quelle: Northern Telecom

### **Vergleich unter Berücksichtigung der Fehlerbehebung**

- Inspektionen ca. 12-20 x effizienter als Testen

Quellen: IBM, Siemens

# Anteil der gefundenen Fehler pro Entwicklungsschritt

| Entwicklungsschritt      | Gefundene Fehler (%) | Schwankungsbreite |
|--------------------------|----------------------|-------------------|
| Funkt. Spezifikation (I) | 3                    | (2 - 5 %)         |
| Grobentwurf (I)          | 12                   | (10 - 16 %)       |
| Feinentwurf (I)          | 20                   | (17 - 22 %)       |
| Code (I)                 | 28                   | (20 - 32 %)       |
| Modul-Test               | 16                   | (12 - 17 %)       |
| Funktionen-Test          | 12                   | (10 - 14 %)       |
| Komponenten-Test         | 7                    | (8 - 14 %)        |
| System-Test              | 2                    | (0,5 - 7 %)       |

(Quelle: Schnurer /Programminspektionen/ 313 f.)

# Vorteile von Inspektionen und Walkthroughs (1)

- Fehler können direkt - und nicht nur anhand ihrer Auswirkungen wie beim Testen - erkannt werden können
- bereits in den frühen Phasen der Softwareentwicklung können Zwischenprodukte systematisch überprüft werden:
  - Kosten werden reduziert
  - Testaufwand wird reduziert
- Ermöglicht wirksame Projektkontrolle (Frühwarnfunktion)
- Intensiviert Kommunikation zwischen Projekt-Mitarbeitern und Auftraggeber
- Vereinfachung der Programme, Vereinheitlichung von Entwurf, Design und Programmierstil





## Vorteile von Inspektionen und Walkthroughs (2)

- Besser verständlicher Programmcode, bessere Dokumentationen
- Erhöhte Termintreue, verbesserte Aufwandsschätzungen
- Lerneffekte bei allen Beteiligten
- Trotz Fehlern in einzelnen Modulen kann die Prüfung fortgesetzt werden
- Erfahrungswerte: 30 bis 90% aller bis zur Fertigstellung des Produkts entdeckten Fehler sind durch Inspektionen gefunden worden



# Problembereiche bei Inspektionen und Walkthroughs

- Erfolg in hohem Maße von den beteiligten Personen abhängig
- Missverständnis: Beurteilung des Programmierers
- Bei großen Projekten ist es häufig schwierig, Mitarbeiter mit der nötigen kritischen Distanz zum Prüfobjekt zu finden
- In der Praxis häufig vernachlässigt, da Aufwand unangemessen erscheint



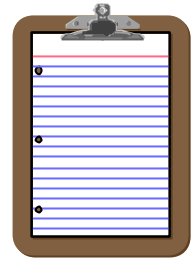
## Fazit (Russell)

- "labor-intensive and low-tech"
- "pays off with faster detection of more errors"
- Aufwand für Inspektionen lässt sich durch Nutzen rechtfertigen:
  - Inspektionen ersparen ein Vielfaches an Fehlerbehebungsaufwand
  - Inspektionen sind effizienter als Testen
- Inspektionen sollen Testen nicht ersetzen
- Testaufwand kann drastisch gesenkt werden, wenn vorher Inspektionen durchgeführt werden

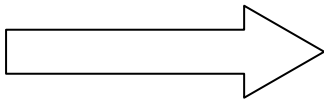


(Quelle: Russell /Inspection/ 28)

# Gliederung



- Begriffsklärung
- Qualitätsmerkmale für Software
- Produkt- und prozessorientierte Perspektive des Qualitätsmanagements
- Dynamische Qualitätssicherung: Testen
- Übung: Dreieck 3000
- Testgeneratoren und Testsysteme
- Statische Qualitätssicherung: Inspektionen
- **Management der Softwareprüfung**
  - Auswahl (und Kombination) von Prüfverfahren
  - Kriterien für die Beendigung des Testens
  - Teststufen
  - Organisation des Testprozesses (Testphasen)
  - einige Aspekte der Testpraxis



# Auswahl (und Kombination) von Prüfverfahren

**Myers und Wallmüller empfehlen folgende Vorgehensweise zum Prüfen von Modulen:**

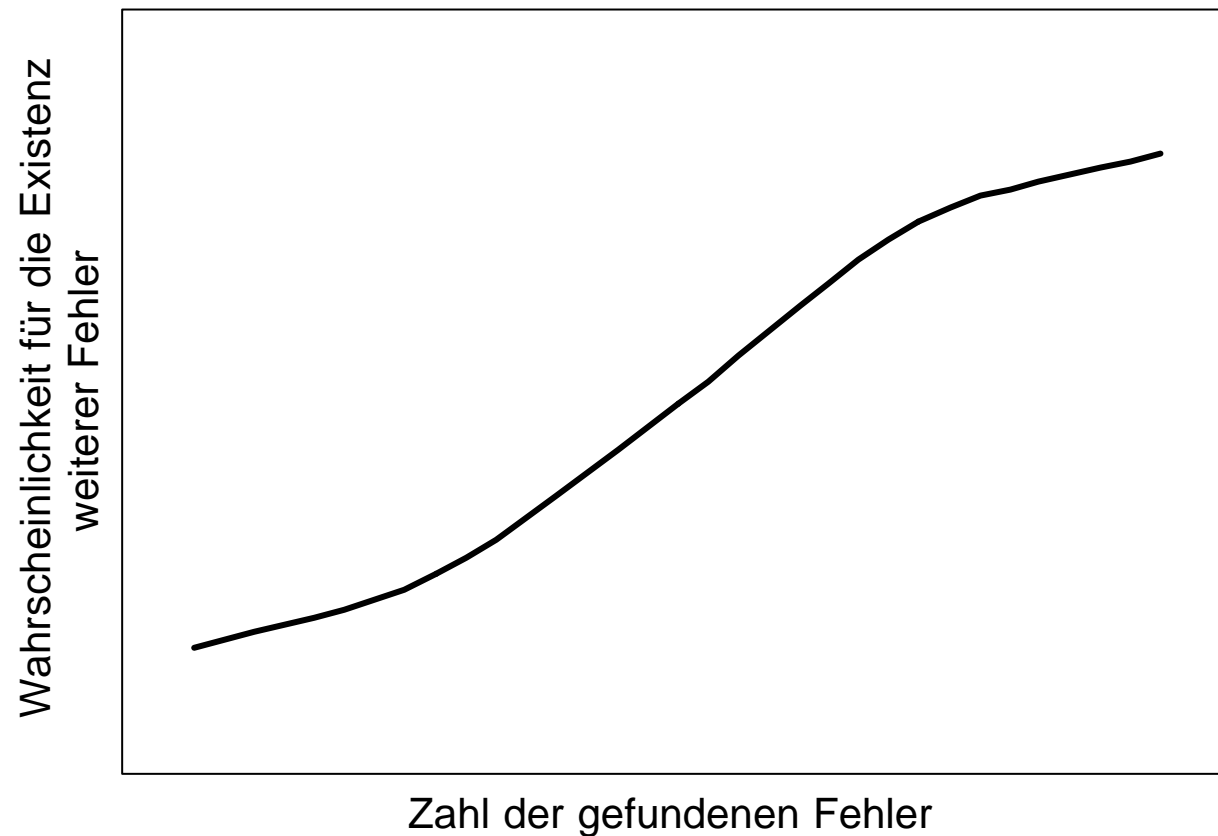
- Prüfen der Spezifikation, z. B. mit der Ursache-Wirkungs-Analyse
- Code-Inspektion
- Definition von Testfällen mit der Äquivalenzklassenmethode
- Auswahl von Testdaten mit der Grenzwertanalyse
- Ergänzung der Testfälle durch intuitive Testfallermittlung (error guessing)
- Überprüfung der erreichten Testabdeckung (z.B. mit C1)
- Ergänzung der Testdaten mit Hilfe struktureller Testverfahren

(in Anlehnung an Myers /Testen/ 75 und Wallmüller /Software-Qualitätssicherung/ 182)

# Kriterien für die Beendigung des Testens

- Vorgegebene Anzahl der zu findenden Fehler
- Beobachtung der Fehlerquote im Verlauf der Testzeit
- Erreichen eines bestimmten C-Maßes /  
einer Testabdeckungskenngröße

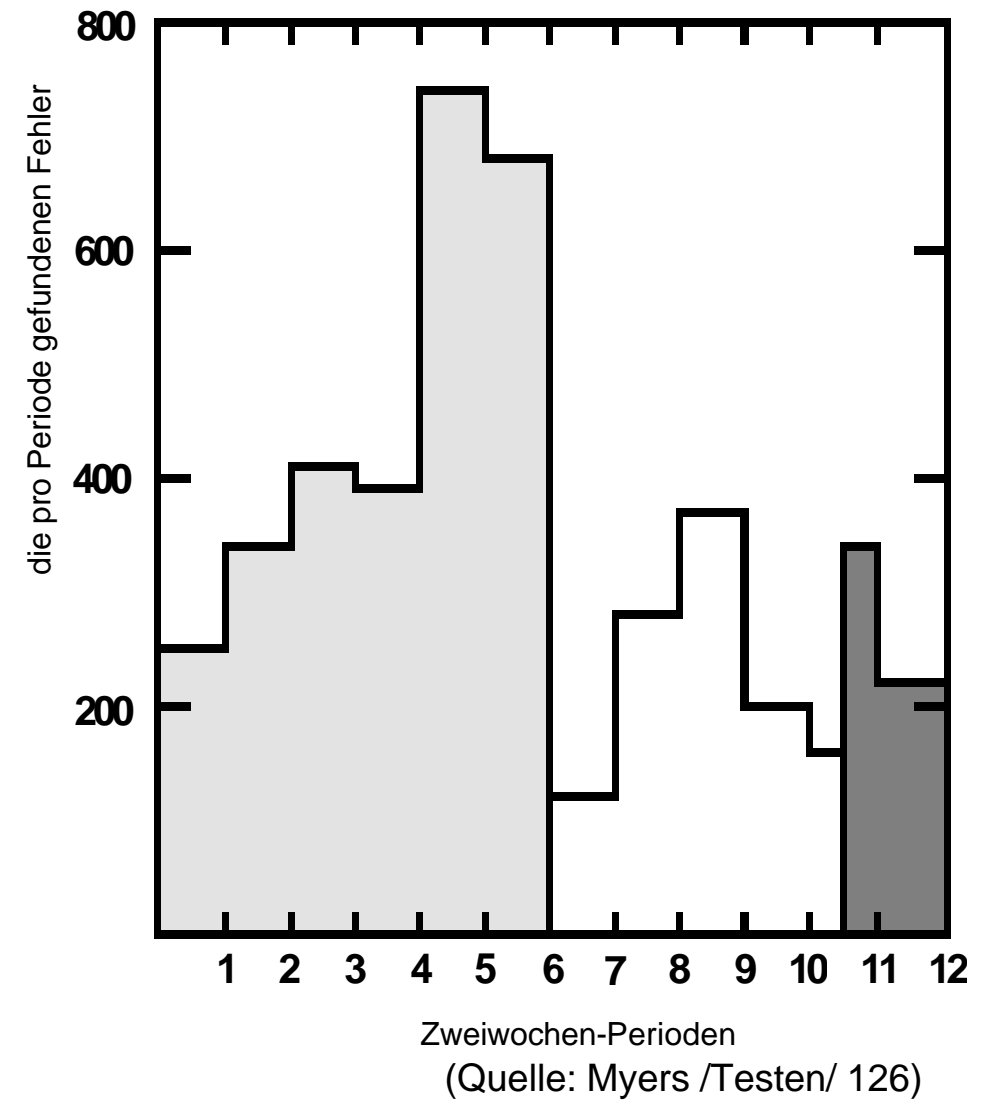
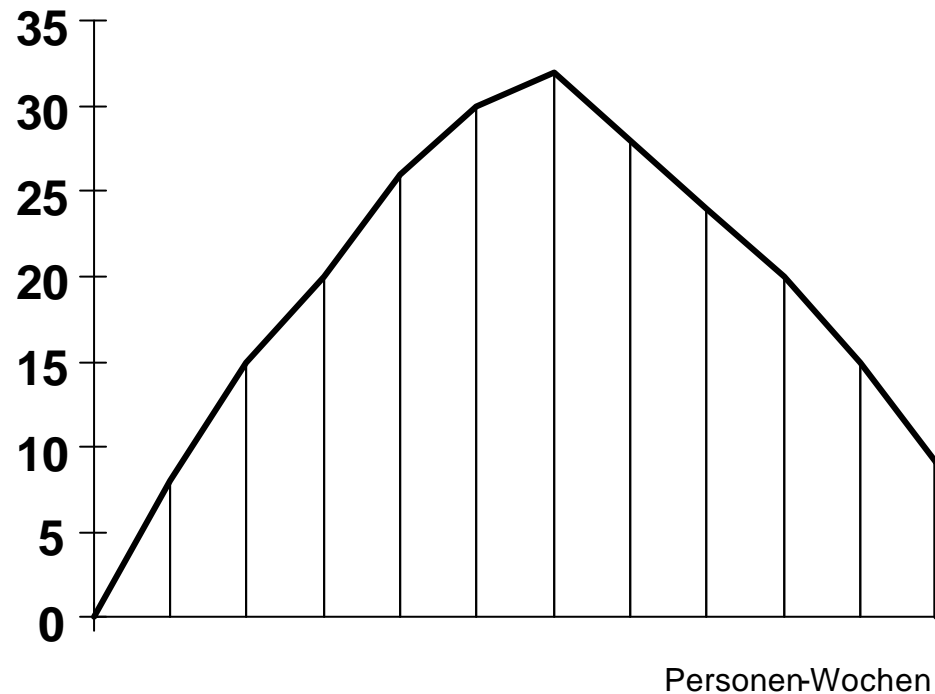
# Beziehung zwischen gefundenen und verbleibenden Fehlern (pro Modul)



(Quelle: Myers /Testen/ 14)

# Fehleranzahl pro Zeiteinheit als Test-Endekriterium

Anzahl gefundener Fehler pro Woche



(Quelle: Wallmüller /Software-Qualitätssicherung/ 190)

(Quelle: Myers /Testen/ 126)



# Testabdeckungskenngrößen als Test-Endekriterium

## Erreichen eines bestimmten c-Maßes / einer Testabdeckungskenngröße

- Typische Beispiele aus der Praxis sind
  - Anweisungsabdeckung (C0) von 95% und
  - Zweigabdeckung (C1) von 85%.
- Praktische Erfahrungen sagen, dass mit C1 = 85% durchschnittlich 90% aller Fehler entdeckt werden.

(Quelle: Trauboth /Software-Qualitätssicherung/ 217)

# Checkliste zur Bestimmung des Auslieferungszeitpunktes von MS Excel 5.0 (1)

- Automatisierte Tests wurden ohne Fehler ausgeführt.
- Manuelle Testfälle wurden ausgeführt.
- Alle spezifizierten Funktionen wurden getestet.
- Ad-hoc-Tests (intuitive Prüfung durch Tester) abgeschlossen.
- Alle gefundenen Fehler behoben, Regressions-Tests abgeschlossen
- Die 200 schwerwiegendsten Fehler wurden einem 2. Regressions-Test unterzogen
- Alle Komponenten (ausser Excel.exe) wurden einen Monat vor dem Produktionstermin “eingefroren”.
- Umfrage bei Testern ergab, dass diese den Eindruck haben, das Produkt könne ausgeliefert werden.

# Checkliste zur Bestimmung des Auslieferungszeitpunktes von MS Excel 5.0 (2)

## Fehlererkennungs-/Fehlerbehebungs-Daten:

- Die Fehlererkennungsquote ist rückläufig und steigt auch bei Regressionstests nicht wieder an.
- Der Anteil der “leichteren” an den gefundenen Fehlern nimmt zu, der Anteil der “schwereren” Fehler nimmt ab.
- Alle gefundenen Fehler wurden daraufhin evaluiert, ob sie in der aktuellen oder erst in der Folgeversion korrigiert werden müssen.
- In der Woche vor dem Produktionstermin werden trotz intensiver Tests keine Fehler mehr gefunden, die unbedingt behoben werden müssen. 321)

# Teststufen

- Modultest
- Integrationstest / Subsystemtest
- Systemtest
- Abnahmetest
- Migrationstest
- Installationstest

(vertiefende Literatur: Trauboth /Software-Qualitätssicherung 2. Aufl./ 231-234; Myers /Testen/ 77-127)

# Organisation des Testprozesses

- Testplanung
- Testentwurf
- Testreview
- Testvorbereitung
- Testausführung
- Testnachbereitung
- Fehlersuche und Fehlerbehebung
- Testinspektion
- Wiederholung vorhergehender Testphasen  
oder  
Freigabe des Testobjekts

(vertiefende Literatur: Trauboth /Software-Qualitätssicherung 2. Aufl./ 235-238)

# **Durchschnittlicher Aufwand einzelner Testaufgaben am gesamten Test-Aufwand**

## **Testfallermittlung vs. Testdaten- und Testprozedurenerstellung vs. Testdurchführung**

- 30-50%: Testfallermittlung
- 20-40%: Testdaten- und Testprozedurenerstellung
- 20-50%: Testdurchführung

## **Vorbereitung vs. Durchführung**

- Testvorbereitung ca. 60 - 80 %
- Testdurchführung und Testauswertung ca. 20 - 40 %

(Quelle: SQS Gesellschaft für Software-Qualitätssicherung mbH)

# Wer sollte Prüfungen durchführen?

## Entwicklung und Prüfung ...

- als Aufgaben einer Rolle
- als Aufgaben unterschiedlicher Organisationseinheiten
- als unterschiedliche Rollen

# **Empirische Ergebnisse zum Softwarequalitätsmanagement (1/3)**

**Schriftliche Befragung von 463 europäischen Unternehmen aus 16 Ländern,  
die sich für das Forschungsprogramm ESSI der Europäischen Kommission  
beworben hatten (1996)**

Dutta, Kulandaiswamy, van Wassenhove /Software Management Best Practices/



## Empirische Ergebnisse zum Softwarequalitätsmanagement (2/3)

- In 75 % der Unternehmen werden Fehler, die nach der Implementierung auftauchen, dokumentiert, analysiert und ihre Behebung kontrolliert.
- In 62 % der Unternehmen existieren Verfahren, mit denen Änderungen am Code kontrolliert werden.
- In 60 % der Unternehmen wird sichergestellt, dass die Konfiguration, die für den abschließenden Abnahmetest verwendet wurde, auch die Konfiguration ist, die beim Kunden installiert wird.
- In 60 % der Unternehmen existieren Verfahren zur Steuerung von Änderungen der Anforderungen, dem Design und der Dokumentation der Software.
- In 52 % der Unternehmen beginnt die Testplanung nach der Formulierung der Kundenanforderungen und nach dem Grobentwurf, aber vor der Programmierung.
- In 51 % der Unternehmen werden vor der Inbetriebnahme der Software Tests durch Benutzer unter Anleitung von Qualitätssicherungsbeauftragten durchgeführt.
- In 51 % der Unternehmen existieren Verfahren, mit denen sichergestellt wird, dass alle erforderlichen Funktionen der Software geprüft worden sind.

## Empirische Ergebnisse zum Softwarequalitätsmanagement (3/3)

- In 45 % der Unternehmen werden regelmäßig Walkthroughs und Inspektionen in jeder Phase der Softwareentwicklung durchgeführt.
- In 42 % der Unternehmen ist eine von den Projekten organisatorisch unabhängige Qualitätssicherung etabliert.
- In 35 % der Unternehmen werden Softwarewerkzeuge verwendet, um den Status von in der Entwicklung befindlichen Modulen und Subroutinen zu verfolgen.
- In 28 % der Unternehmen werden Softwarefehler analysiert, um deren Ursachen bekämpfen zu können.
- In 25 % der Unternehmen werden Werkzeuge verwendet, um die Abdeckung der Software durch Testdaten zu überprüfen.
- In 25 % der Unternehmen werden Regressionstests durchgeführt, sobald Änderungen an bereits implementierten Softwaremodulen vorgenommen wurden.
- In 19 % der Unternehmen werden Softwarewerkzeuge verwendet, um die Anforderungsänderungen vom Design bis zur Codierung zu dokumentieren.
- In 10 % der Unternehmen werden Aufzeichnungen über die Effizienz des Testens in allen Phasen der Softwareentwicklung geführt und analysiert.

# Kontroll- und Verständnisfragen



- Definieren Sie den Begriff Softwareproduktqualität.  
Beschreiben Sie drei verschiedene Qualitätsmerkmale von Software.
- Diskutieren Sie die folgende Thesen:
  - Das Ziel des Testens besteht darin zu zeigen, dass das Testobjekt die in der Spezifikation beschriebenen Funktionen korrekt ausführt.
  - Die Entwickler eines Softwaremoduls sollten dieses nicht testen.
- In verschiedenen empirischen Studien wurde ermittelt, dass Code-Inspektionen effizienter sind als Softwaretests. Welche Gründe gibt es dafür?



# Literaturhinweise

- Michael E. Fagan: Design and Code Inspections to Reduce Errors in Program Development. In: IBM Systems Journal. Nr. 3, 1976, S. 182-211
- Michael E. Fagan: Advances in Software Inspections. In: IEEE Transactions on Software Engineering. Nr. 7, 1986, S. 744-751
- Peter Liggesmeyer: Modultest und Modulverifikation. State of the Art. Mannheim - Wien - Zürich 1990
- Werner Mellis, Georg Herzwurm, Dirk Stelzer: TQM der Softwareentwicklung. Mit Prozeßverbesserung, Kundenorientierung und Change Management zu erfolgreicher Software. 2. Aufl., Braunschweig - Wiesbaden 1998
- Glenford J. Myers: Methodisches Testen von Programmen. 6. Aufl., München - Wien 1999.
- Glen W. Russell: Experience with Inspection in Ultra-Large Scale Development. In: IEEE Software. Nr. 1, 1991, S. 25-31
- Paul Schmitz, Heinz Bons, Rudolf van Megen: Software-Qualitätssicherung - Testen im Software-Lebenszyklus. 2. Aufl., Braunschweig - Wiesbaden 1983
- Karl Ernst Schnurer: Programminspektionen. Erfahrungen und Probleme. In: Informatik-Spektrum. Nr. 6, 1988, S. 312-322
- Ed Yourdon: Good Enough Software. In: Guerrilla Programmer. Nr. 4, 1995, S. 1-3
- Ed Yourdon: Good-Enough Software. In: Application Development Strategies. Nr. 1, 1996, S. 1-13