



Vorlesung Softwaretechnik I (SS 2024)

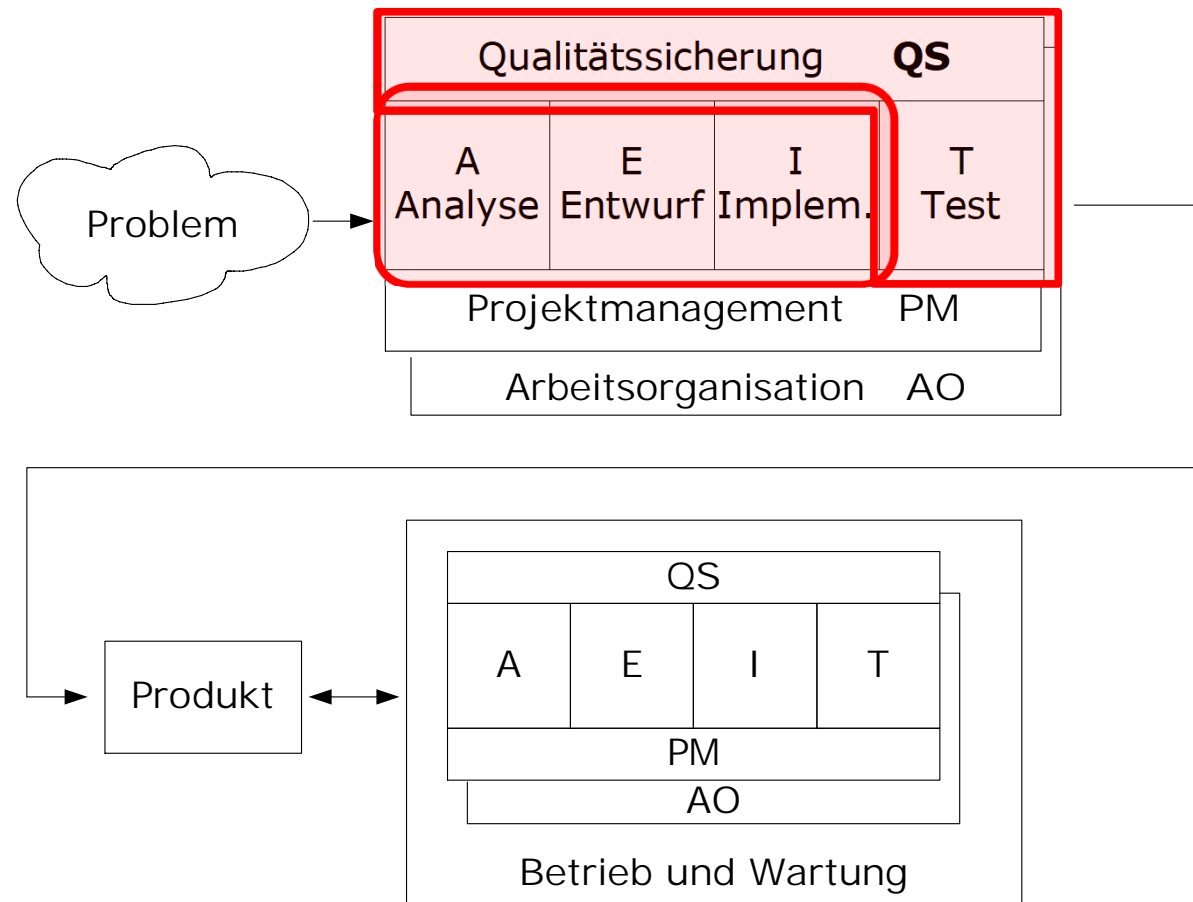
10. Software Qualität & Qualitätssicherung

Prof. Dr. Jens Grabowski

Tel. 39 172022

grabowski@informatik.uni-goettingen.de

Worum geht es in diesem Kapitel?





Inhalt

- Software Qualität
- Qualitätssicherung
- Qualitätsaspekte für Softwareprojekte
- Lernziele



Inhalt

■ **Software Qualität**

- **Was ist Software Qualität?**
- Qualitätskriterien
 - Korrektheit von Software
 - Zuverlässigkeit von Software
 - Robustheit von Software
 - Vertrauenswürdigkeit von Software
 - Effizienz
 - Wartbarkeit von Software
 - Portierbarkeit von Software
 - Kompatibilität von Software
- Was ist Software Qualität? (revisited)
- Teufelsquadrat nach Sneed



Was ist Software Qualität?

```
program SORT;
var    a, b: file of integer;
      Feld: array [ 1 .. 10 ] of integer;
      i, j, k, l : integer
begin
    open ( a, '/usr/jens/Zahlen/Datei');
    i := 1;
    while not eof ( a ) do begin
        read ( a, Feld [ i ] ); i := i + 1
    end;
    l := i - 1;
    open ( b, '/usr/jens/Zahlen/Datei');
    for i := 2 to l do begin
        for j := l downto i do
            if Feld [ j - 1 ] > Feld [ j ] then begin
                k := Feld [ j - 1 ]; Feld [ j - 1 ] := Feld [ j ]; Feld [ j ] := k
            end;
        write ( b , Feld [ i - 1 ] )
    end
end SORT;
```



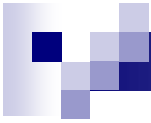
Was ist Software Qualität?

- Bewertung des Sortierprogramms:
 - ☐ Wie funktioniert das Programm?
 - ☐ Lesbarkeit?
 - ☐ Robustheit?
 - ☐ Portabilität?



Ziel der Software-Technik ...

- ... ist die effiziente Entwicklung messbar qualitativ hochwertiger Software, die
 - **korrekt** bzw. **zuverlässig** arbeitet,
 - **robust** auf Fehleingaben, Hardwareausfall, usw. reagiert,
 - **effizient** (ökonomisch) mit Hardwareressourcen umgeht,
 - eine **benutzerfreundliche** Oberfläche besitzt und
 - gut **wartbar** und einfach **wiederverwendbar** ist.

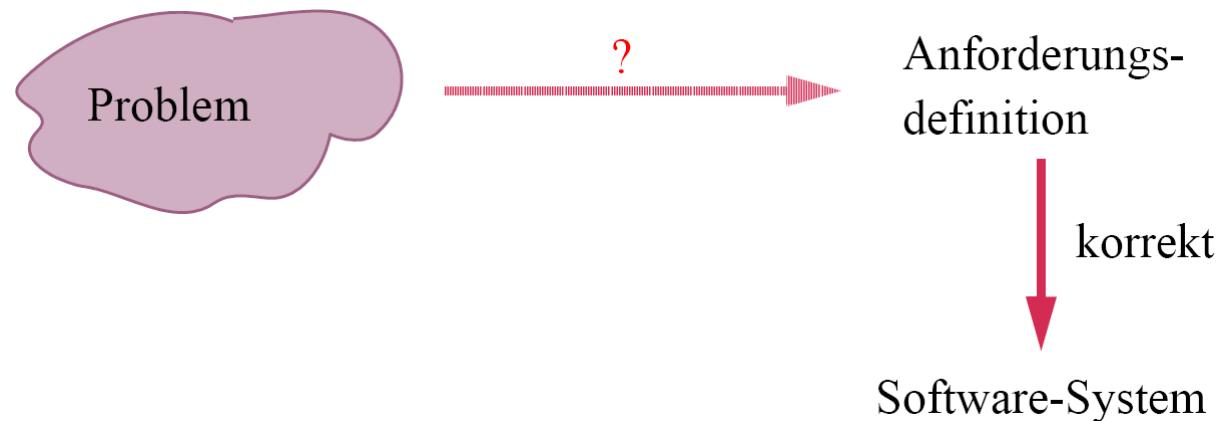


Inhalt

- Software Qualität
 - Was ist Software Qualität?
 - **Qualitätskriterien**
 - **Korrektheit** von Software
 - **Zuverlässigkeit** von Software
 - **Robustheit** von Software
 - **Vertrauenswürdigkeit** von Software
 - **Effizienz**
 - **Wartbarkeit** von Software
 - **Portierbarkeit** von Software
 - **Kompatibilität** von Software
 - Was ist Software Qualität? (revisited)
 - Teufelsquadrat nach Sneed

Korrektheit von Software

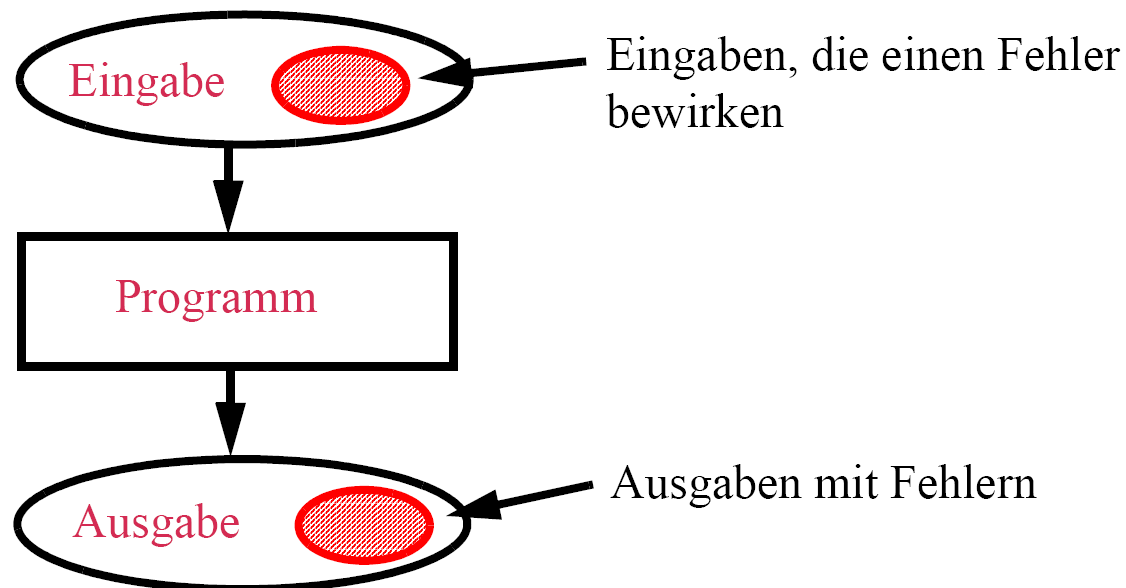
- Ein Software-System ist (funktional) korrekt, wenn es sich genauso verhält, wie es in der **Anforderungsdefinition** festgelegt wurde.



- Problem:
 - Anforderungsdefinitionen sind häufig informal und eigentlich nie ganz widerspruchsfrei, vollständig, ...

Zuverlässigkeit von Software

- Zuverlässige Software funktioniert meistens, Zuverlässigkeit ist also ein Maß für die Wahrscheinlichkeit, dass ein Software-System sich in einem bestimmten Zeitraum so verhält, wie es von ihm erwartet wird.
- Korrekte Software = 100% zuverlässige Software



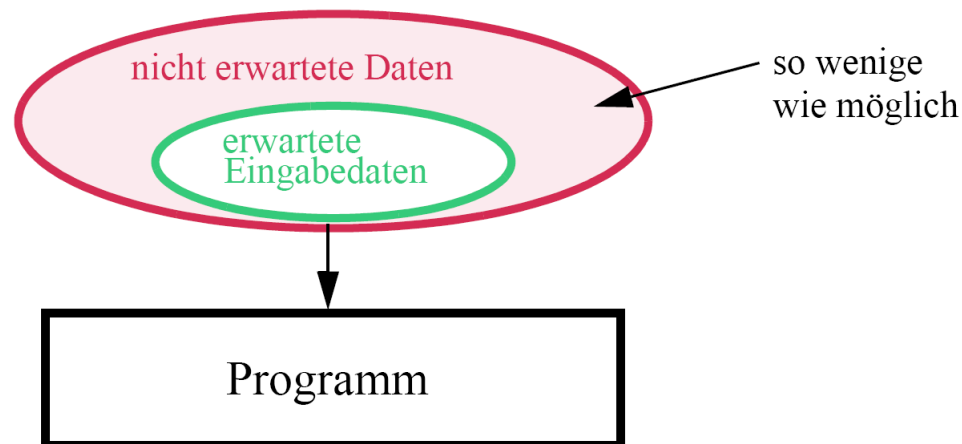


Zuverlässigkeit von Software

- Beispiele für (un-)zuverlässige Software:
 - zuverlässige Software:
 - Textverarbeitungssystem, das manchmal Schmierzeichen auf den Bildschirm schreibt.
 - unzuverlässige Software:
 - Textverarbeitungssystem, das in unregelmäßigen Abständen aus unerfindlichen Gründen abstürzt.
- Metriken für Bewertung der Zuverlässigkeit:
 - *rate of failure occurrence* (ROFOC):
 - Häufigkeit von nicht erwartetem Verhalten, z.B. 2 Fehler pro 100 operationellen Zeiteinheiten.
 - *mean time between failure* (MTBF):
 - mittlerer Zeitabstand zwischen zwei Fehlern (also von unerwartetem bzw. unerwünschtem Verhalten).
 - *availability* (AVAIL):
 - mittlere Verfügbarkeit der Software, z.B. 998 von 1000 Zeiteinheiten war das System benutzbar.

Robustheit von Software

- Ein Software-System ist robust, wenn es auch unter **unvorhergesehenen Umständen** funktioniert bzw. vernünftig reagiert (z.B. auf zufällige oder beabsichtige Angriffe).



- Anmerkung:
 - Auch der Software-Entwicklungsprozeß sollte robust sein, z.B. gegen
 - Ausfall von Mitarbeitern
 - unvermeidlicher Hardware-/Betriebssystemwechsel



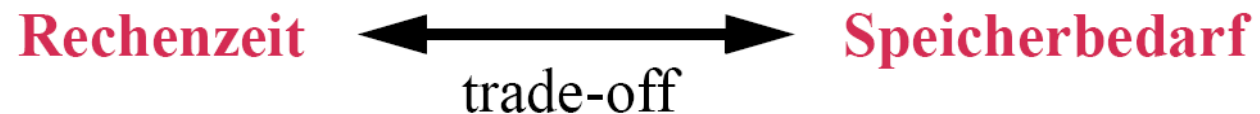
Vertrauenswürdigkeit von Software

- Vertrauenswürdige Software verursacht (auch) im Fehlerfall **keine Katastrophen**:
 - inkorrekte Software kann vertrauenswürdig sein (z.B. wenn die Software keinen wirklichen Schaden anrichten kann),
 - korrekte Software muss nicht vertrauenswürdig sein (z.B. bei Fehlern in der Anforderungsdefinition).
- Beispiele (nicht) vertrauenswürdiger Software:
 - Joystick-Steuerungen in Spielekonsolen sind von Natur aus vertrauenswürdig (können keinen echten Schaden anrichten).
 - Joystick-Steuerung im Los Alamos National Laboratory:
 - Am 26. Februar 1998 führte die Fehlfunktion einer Joystick-Steuerung dazu, dass sich zwei Uranstücke nicht langsam, sondern mit maximaler Geschwindigkeit aufeinander zu bewegen. Der Operator drückte rechtzeitig einen Notaus-Knopf (angeblich war die Summe der Massen unterkritisch).
[ACM SIGSOFT Software Engineering Notes, vol. 23, no. 4 (1998), S. 21]

Effizienz von Software

■ Effiziente Software

- Effiziente Software nutzt Hardware-Ressourcen (hinreichend) ökonomisch (so dass die Software auf der verfügbaren Hardware benutzbar ist).

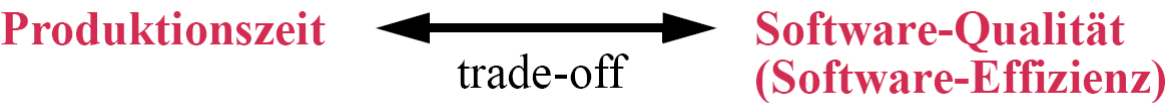


■ Wie ermittelt man Effizienz:

- theoretische Komplexitätsanalyse
- Simulationsmodelle
- Messungen am realen System (Profiling, ...)

Effizienz des Software Entwicklungsprozesses

- Effizienz des Software-Erstellungsprozesses:
 - Effiziente Software-Herstellung = **produktive** Software-Herstellung, also mit möglichst wenig Mitarbeitern in möglichst kurzer Zeit möglichst gute Software herstellen.
- Problem:

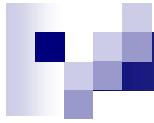


Produktionszeit \longleftrightarrow **Software-Qualität (Software-Effizienz)**
trade-off
- Lösung:
 - Wiederverwendung von Software-Komponenten, Vorgehensweisen, ...
 - Einsatz von hohen Programmiersprachen, Codegeneratoren, ...
 - ...
- Wie misst man Produktivität?



Wartbarkeit von Software

- Software-Wartung:
 - Änderungen an der Software aufgrund von Fehlern, geänderter Anforderungen oder neuer Randbedingungen (z.B. neue Hardware, neue Betriebssystemversion). Gut wartbare Software ist leicht korrigierbar, modifizierbar und erweiterbar.
- Wartbarkeit wird unterstützt durch:
 - Einhaltung von Programmierrichtlinien,
 - Einhaltung der Prinzipien des Architekturentwurfs,
 - „Aktuelle“ Dokumentation (muss bei Wartung aktualisiert werden),
 - Versionskontrolle und Änderungsmanagement,
 - usw.
- Anmerkung:
 - Jede Wartung reduziert die Wartbarkeit, solange bis die Software nicht mehr änderbar ist (Gesetz der zunehmenden Entropie).



Portierbarkeit von Software

■ Portierbare Software:

- Ein System ist portierbar, falls es in **verschiedenen Umgebungen** (ohne großen Änderungsaufwand) läuft.
- **Umgebung** = Hardware und Basissoftware

■ Portierbarkeit erreicht man durch:

- Dokumentation von Systemvoraussetzungen und Einschränkungen.
- Verwendung von Hochsprachen mit Standardbibliotheken.
- Unabhängigkeit von Hardwareeigenschaften.
- Portierbarkeit schon im Software-Design mit berücksichtigen.
- usw.



Kompatibilität von Software

- Kompatible (interoperable) Software
 - Kompatible Software kann leicht mit anderer Software kooperieren (oder durch Konkurrenzprodukte ersetzt werden).
- Beispiele:
 - Grafiken eines Zeichenprogramms oder Tabellen einer Spreadsheet-Anwendung können in Textverarbeitung übernommen werden.
 - Verschiedene Software-Entwicklungsumgebungen, CAD-Programme, ... können (fast) ohne Semantikverlust ihre Daten austauschen.
 - Integrierte Anfragen, Datenmanipulationen auf mehreren Datenbanken.
- Wichtig:
 - **Standards** für Schnittstellen, Austauschformate, ...
 - **offene Systemen** mit austauschbaren Komponenten



Inhalt

- Software Qualität
 - Was ist Software Qualität?
 - Qualitätskriterien
 - Korrektheit von Software
 - Zuverlässigkeit von Software
 - Robustheit von Software
 - Vertrauenswürdigkeit von Software
 - Effizienz
 - Wartbarkeit von Software
 - Portierbarkeit von Software
 - Kompatibilität von Software
 - **Was ist Software Qualität? (revisited)**
 - Teufelsquadrat nach Sneed



Was ist Software Qualität? (revisited)

■ Definition Qualität allgemein

- Qualität ist die **Gesamtheit von Eigenschaften und Merkmalen** eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht. (DIN 55350 Teil 11)

DIN 55350 Teil 11
Begriffe der Qualitätssicherung und Statistik;
Grundbegriffe der Qualitätssicherung, 08.95

■ Software Qualitätsmerkmale - (DIN ISO 9126)

- Funktionalität
- Zuverlässigkeit
- Benutzbarkeit
- Effizienz
- Änderbarkeit
- Übertragbarkeit

ISO/IEC 9126:1991
Bewerten von Softwareprodukten, Qualitätsmerkmale
und Leitfaden zu ihrer Verwendung
(identisch mit DIN 66272, 94)



Was ist Software Qualität? (revisited)

- Software Qualität definiert sich aus der Erfüllung von Qualitätszielen und der Umsetzung von Qualitätsmerkmalen.
- (informell) Visualisiert
 - Gesamt-Softwarequalität :=

<i>Bewertung</i>	Funktionalität (u.a. Korrektheit)	+
<i>Bewertung</i>	Zuverlässigkeit	+
<i>Bewertung</i>	Benutzbarkeit (u.a. Robustheit , Vertrauenswürdigkeit)	+
<i>Bewertung</i>	Effizienz	+
<i>Bewertung</i>	Änderbarkeit (u.a. Wartbarkeit)	+
<i>Bewertung</i>	Übertragbarkeit (u.a. Portierbarkeit , Kompatibilität)	
 - Erläuterung: Die **Blau/Türkis** hervorgehobenen Begriffe wurden im Abschnitt Qualitätsmerkmale detaillierter diskutiert.

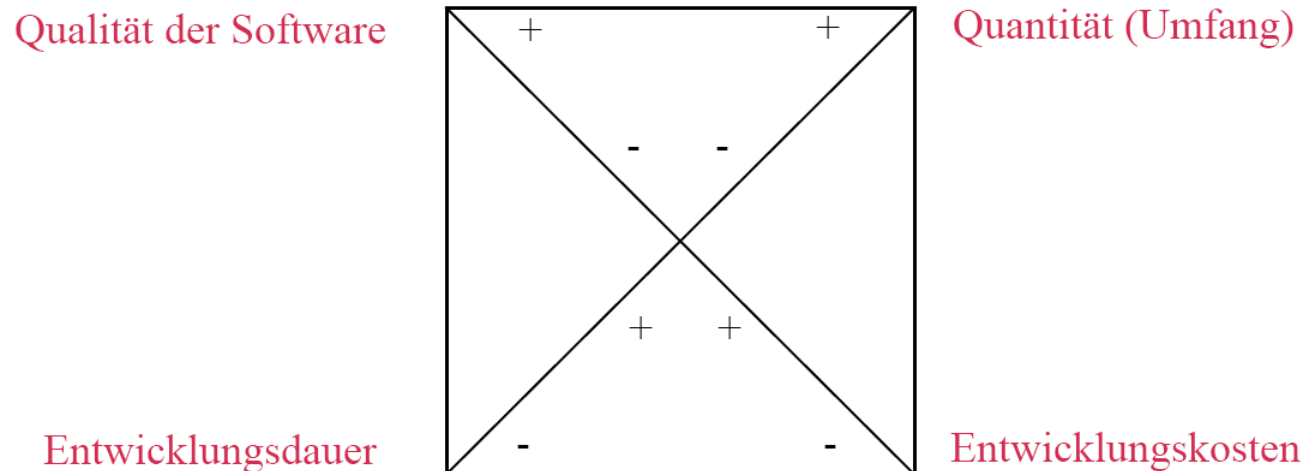


Inhalt

- Software Qualität
 - Was ist Software Qualität?
 - Qualitätskriterien
 - Korrektheit von Software
 - Zuverlässigkeit von Software
 - Robustheit von Software
 - Vertrauenswürdigkeit von Software
 - Effizienz
 - Wartbarkeit von Software
 - Portierbarkeit von Software
 - Kompatibilität von Software
 - Was ist Software Qualität? (revisited)
 - **Teufelsquadrat nach Sneed**

Teufelsquadrat nach Sneed

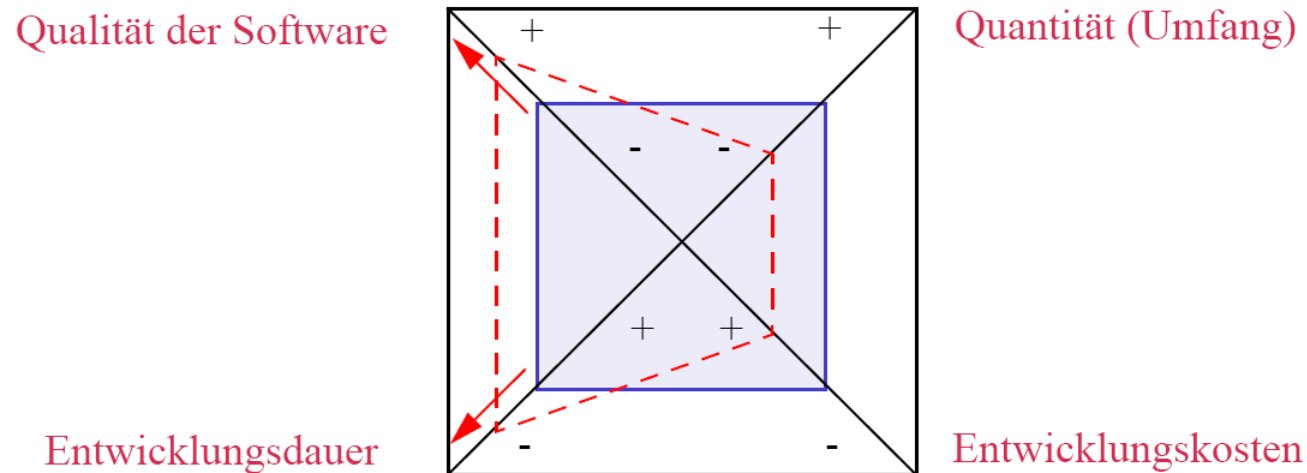
[H.M. Sneed: Software-Management, Müller GmbH (1987)]



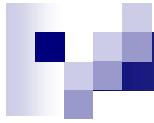
- Erläuterungen zum Teufelsquadrat:
 - Für Software-Entwicklungsprozess werden Qualität, Quantität, Entwicklungsdauer und Entwicklungskosten der Software gemessen.
 - Die Messwerte werden auf den vier Skalen (von den Ecken zur Mitte verlaufend) eingetragen und die Punkte miteinander verbunden

Teufelsquadrat nach Sneed

[H.M. Sneed: Software-Management, Müller GmbH (1987)]



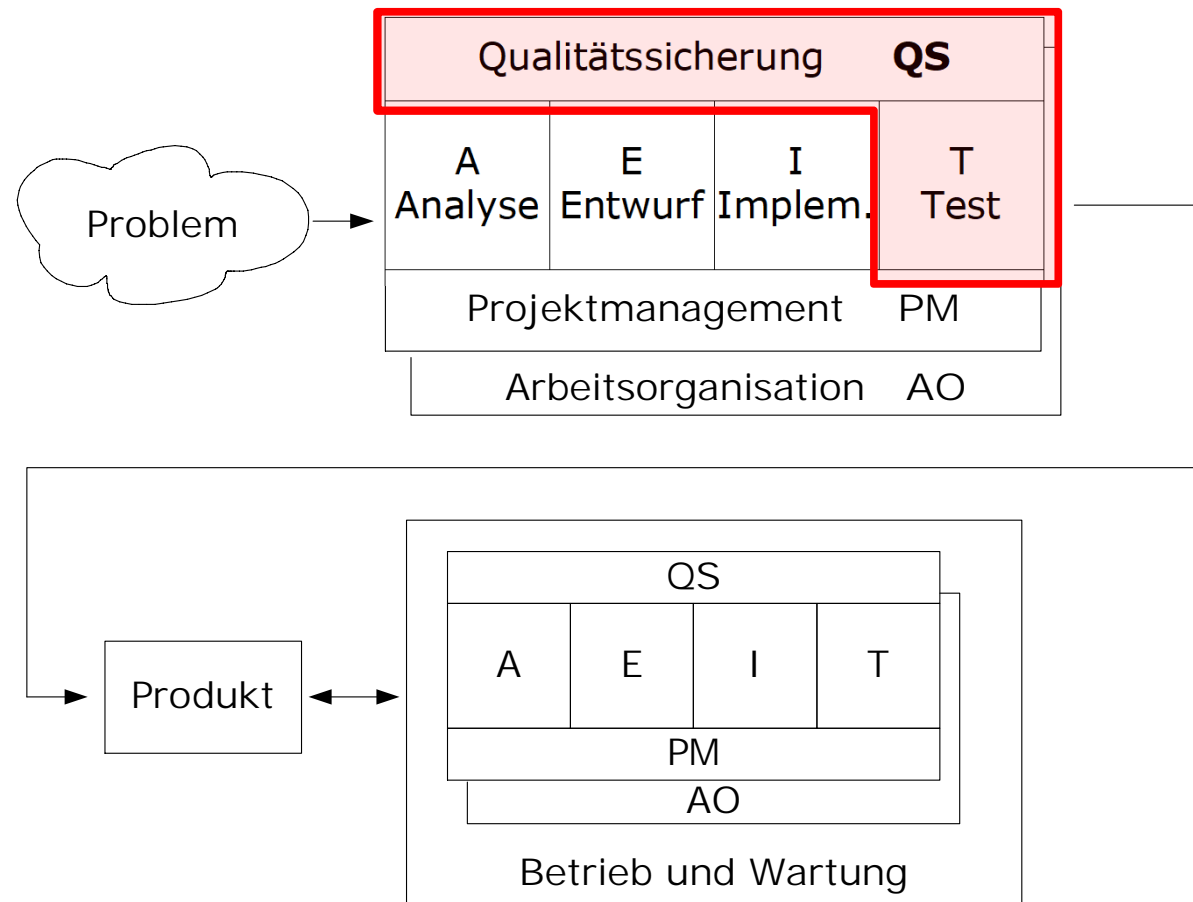
- Erläuterungen zum Teufelsquadrat:
 - Fläche des Quadrats = **Produktivität** ist invariant (Veränderung ggf. durch Mitarbeiterschulung, besseres Vorgehensmodell, ...)
 - Erhöhung der Qualität und Reduktion der Entwicklungsdauer geht nur mit Reduktion des Produktumfanges und/oder Erhöhung der Entwicklungskosten



Inhalt

- Software Qualität
- **Qualitätssicherung**
- Qualitätsaspekte für Softwareprojekte
- Lernziele

Worum geht es in diesem Kapitel?

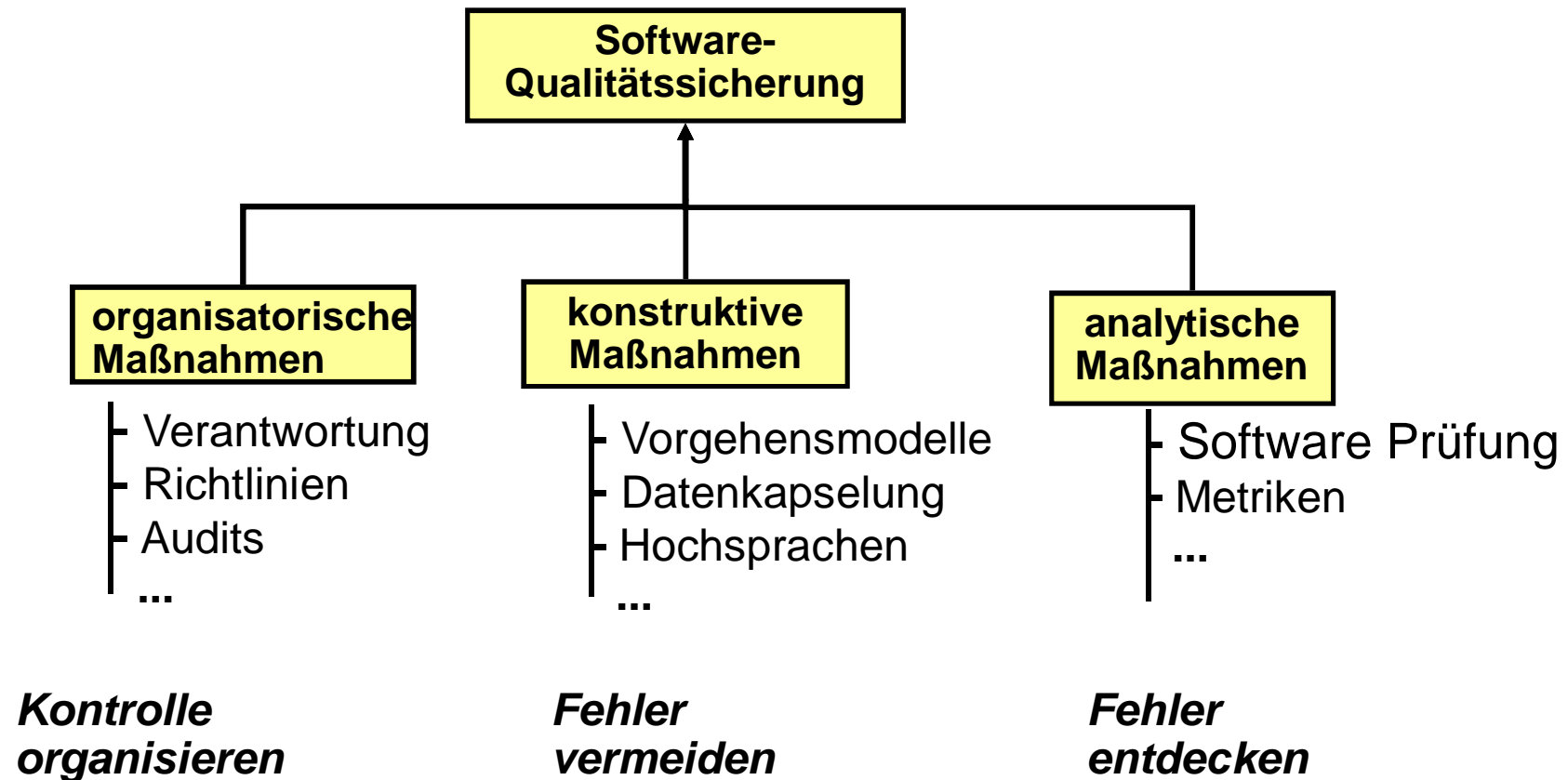




Inhalt

- **Qualitätssicherung**
 - **Qualitätssicherung im Überblick**
 - Statische Prüfungen
 - Review
 - Statische Analysen
 - Kontrollflussanalyse
 - Datenflussanalyse
 - Dynamische Prüfungen
 - Glass-Box Testing
 - Black-Box Testing
 - Testen im Software-Erstellungsprozess

Qualitätssicherung im Überblick (1)





Qualitätssicherung im Überblick (2)

- Organisatorische Maßnahmen


- Überprüfung und Verbesserung der Qualität der Softwareentwicklungsprozesse in einer Organisation
 - ISO 9000 ff Zertifizierung bestätigt geregelte Vorgehensweise (trifft nahezu keine Aussage über Qualität der eingesetzten Vorgehensweisen).
 - Capability Maturity Model Integration (CMMI) unterscheidet verschiedene Reifegrade bei Softwareentwicklungsprozessen und schlägt Verbesserungsmaßnahmen vor.

- Konstruktive Maßnahmen zur Fehlervermeidung

- Vorgehensweisen zur Entwicklung von vornherein qualitativ hochwertiger Softwareprodukte

- Analytische Maßnahmen zur Fehlerentdeckung & Fehlerelimination

- Nachträgliche Überprüfung der Qualität entwickelter Produkte oder Dokumente



Prinzipien für die konstruktive und analytische Qualitätssicherung

■ Qualitätszielbestimmung

- Auftraggeber und Auftragnehmer legen vor Beginn der Softwareentwicklung gemeinsames Qualitätsziel mit nachprüfbarem Kriterienkatalog fest (als Vertragsbestandteil des Pflichtenheftes, z.B. durch Abnahmetests).

■ Quantitative Qualitätssicherung

- Einsatz automatisch ermittelbarer Metriken zur Qualitätsbestimmung (objektivierbare, ingenieurmäßige Vorgehensweise).

■ Integrierte, frühzeitige konstruktive Qualitätssicherungsmaßnahmen

- Verwendung geeigneter Methoden, Sprachen und Werkzeuge (Sprachen mit vernünftiger Syntax, strenger Typisierung, ...)

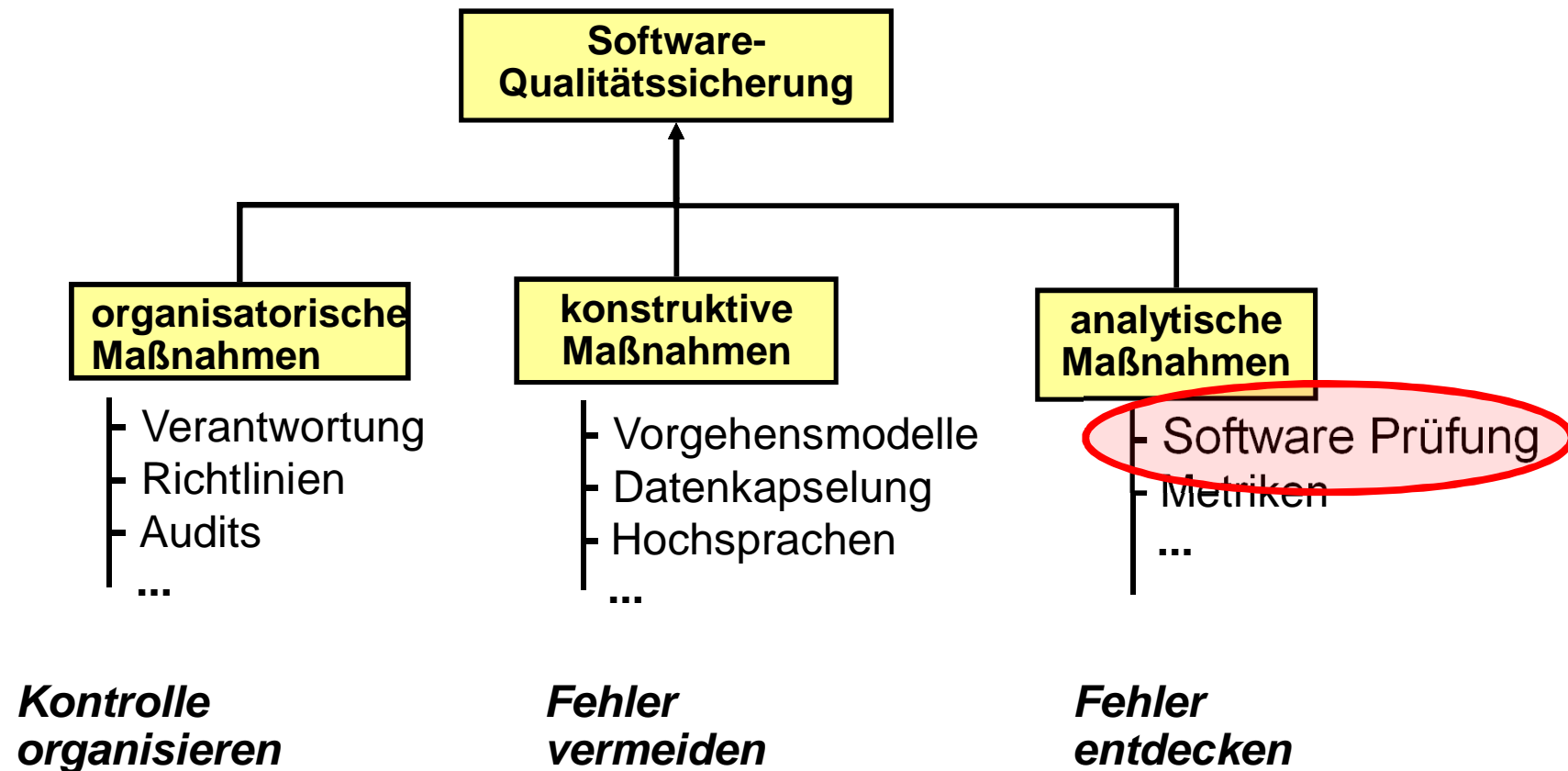
■ Integrierte, frühzeitige analytische Qualitätssicherungsmaßnahmen

- Nicht nur fertiges Softwareprodukt testen, sondern alle erzeugten Dokumente wie Analyse- und Designmodelle sowie einzelne Softwarekomponenten.

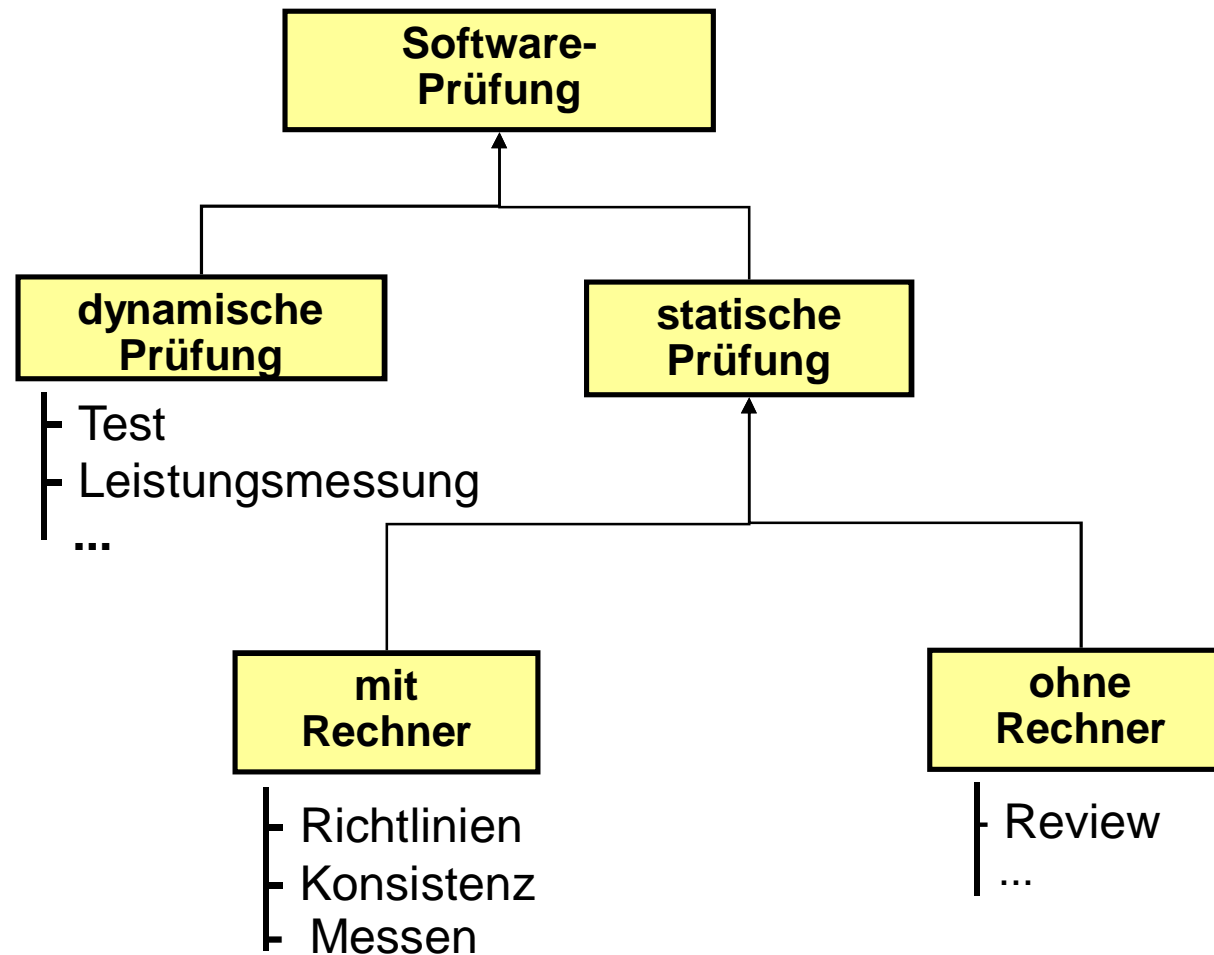
■ Unabhängige Qualitätssicherung

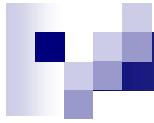
- Entwicklungsprodukte werden durch eigenständige Qualitätssicherungsabteilung überprüft und abgenommen (verhindert u.a. Verzicht auf Testen zugunsten Einhaltung des Entwicklungszeitplans)

Qualitätssicherung im Überblick (1)



Qualitätssicherung im Überblick (3)

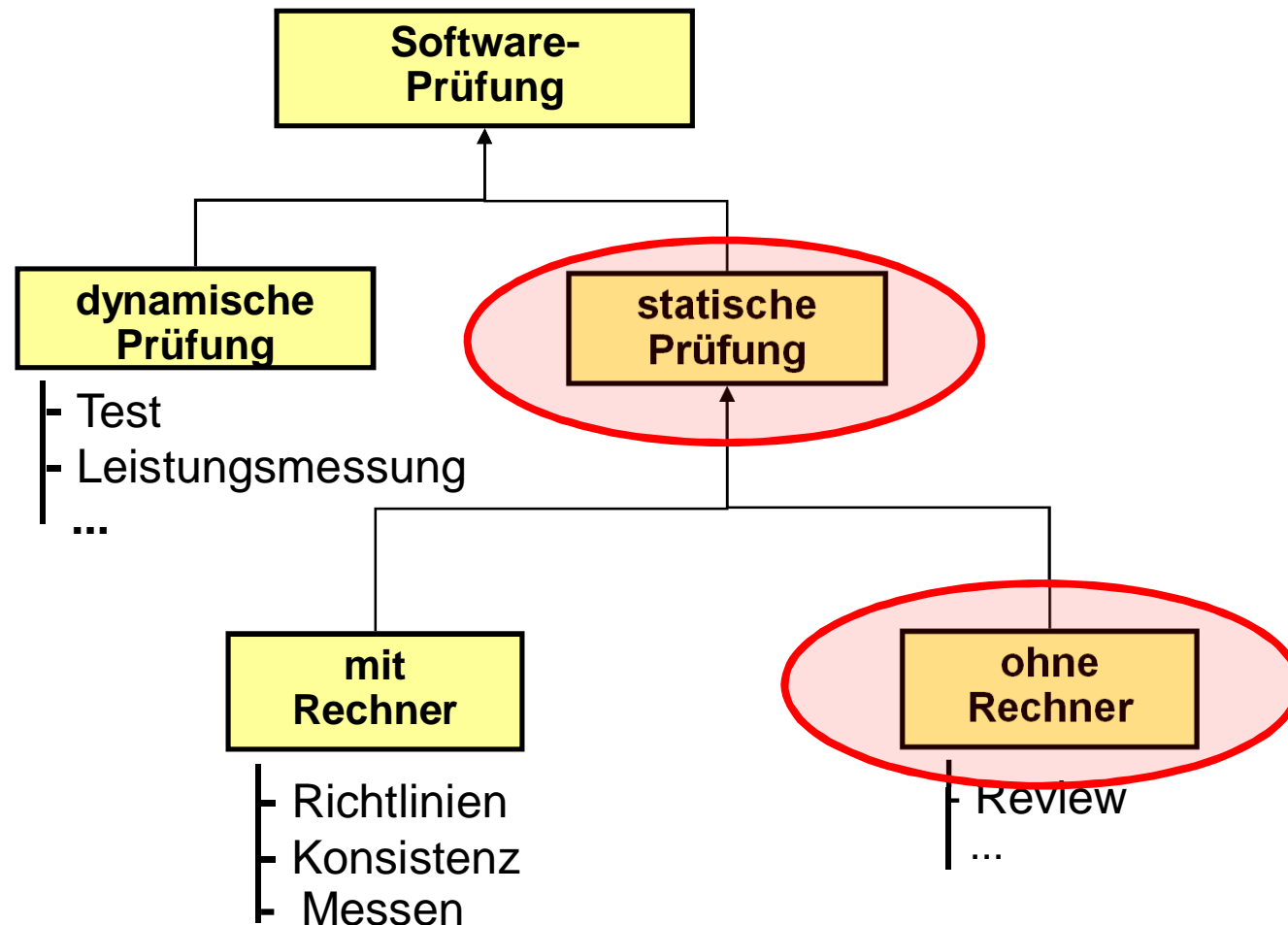




Inhalt

- Qualitätssicherung
 - Qualitätssicherung im Überblick
 - **Statische Prüfungen**
 - **Review**
 - Statische Analysen
 - Kontrollflussanalyse
 - Datenflussanalyse
 - Dynamische Prüfungen
 - Glass-Box Testing
 - Black-Box Testing
 - Testen im Software-Erstellungsprozess

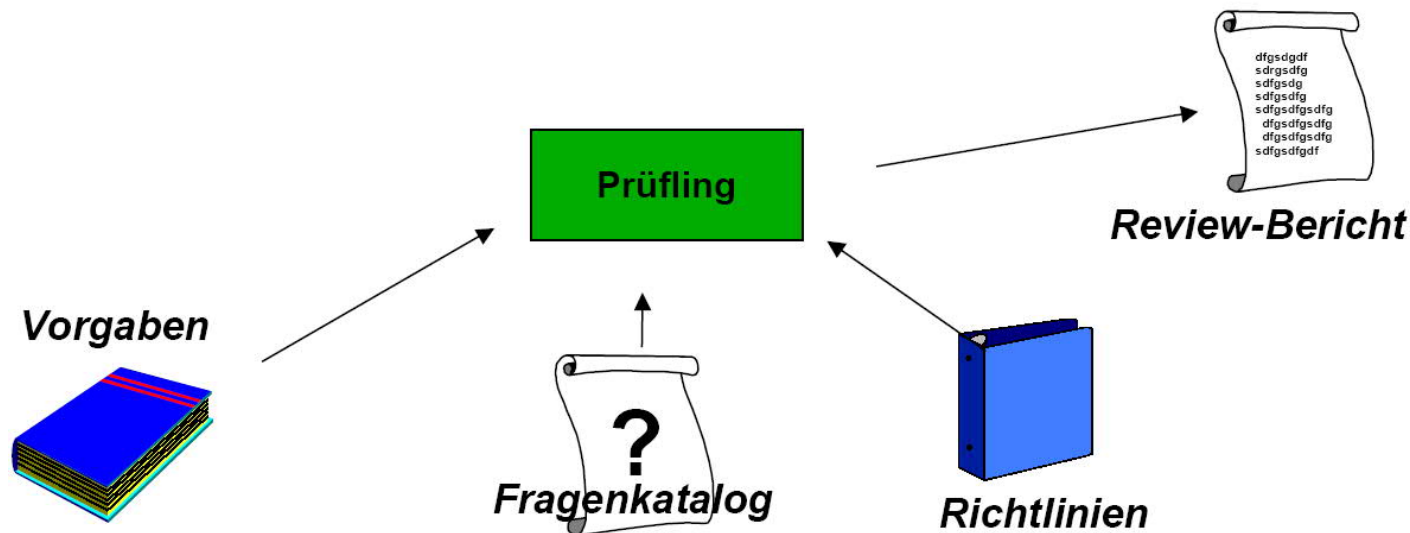
Qualitätssicherung im Überblick (3)

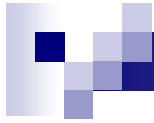


Review - Definition

■ Review

- Ein Review (Inspection) ist die Prüfung eines Dokuments (Prüfling) gegenüber **Vorgaben** und **Richtlinien** mit dem Ziel, **Fehler und Schwächen** des Prüflings aufzuzeigen, aber auch **positive Merkmale** zu würdigen.

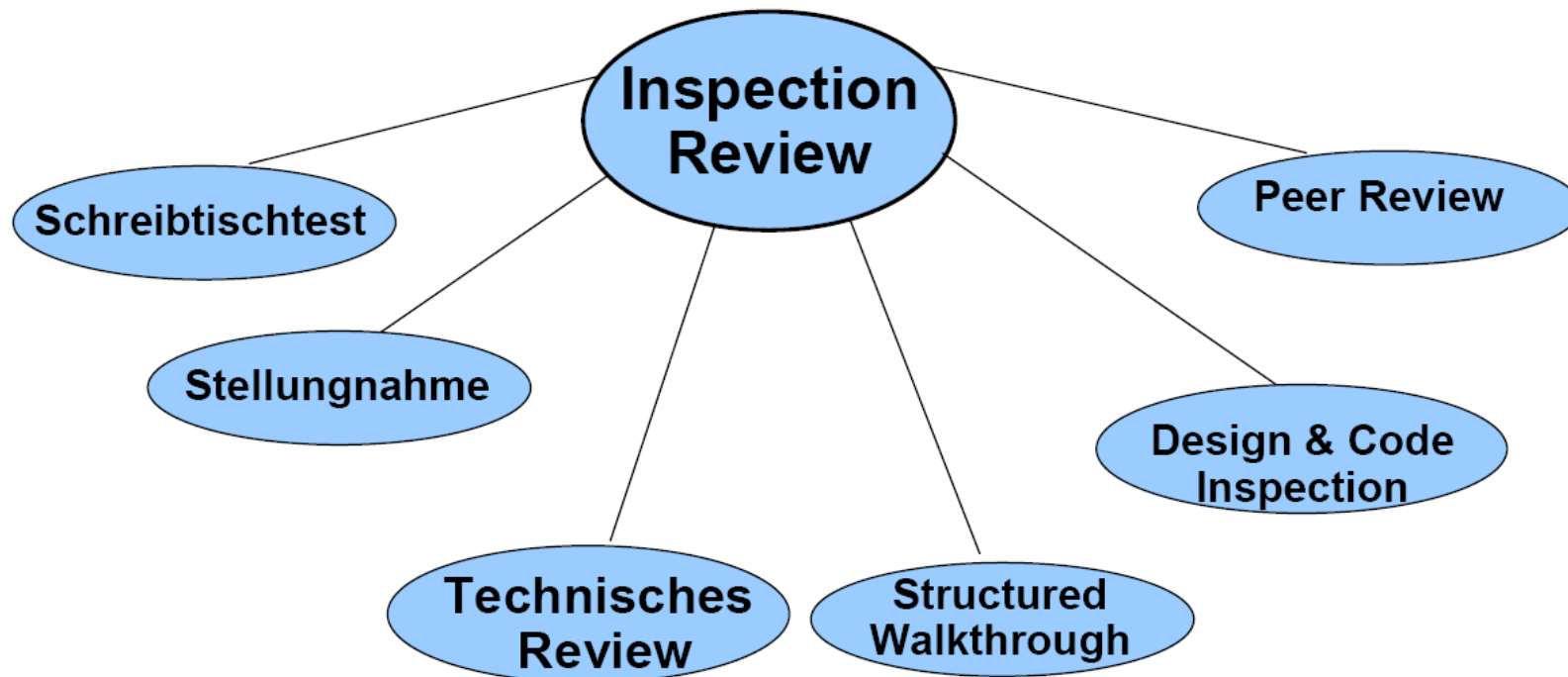




Reviews

- Alle Dokumente können einem Review oder einer Inspektion unterzogen werden, z. B.
 - ☐ Anforderungsbeschreibung,
 - ☐ Entwurfsspezifikationen,
 - ☐ Programmtexte,
 - ☐ Testpläne,
 - ☐ Benutzungshandbuch,
 - ☐ usw.
- Oft sind Reviews die einzige Möglichkeit, die Semantik solcher Dokumente zu überprüfen.

Klassifikation von Reviews

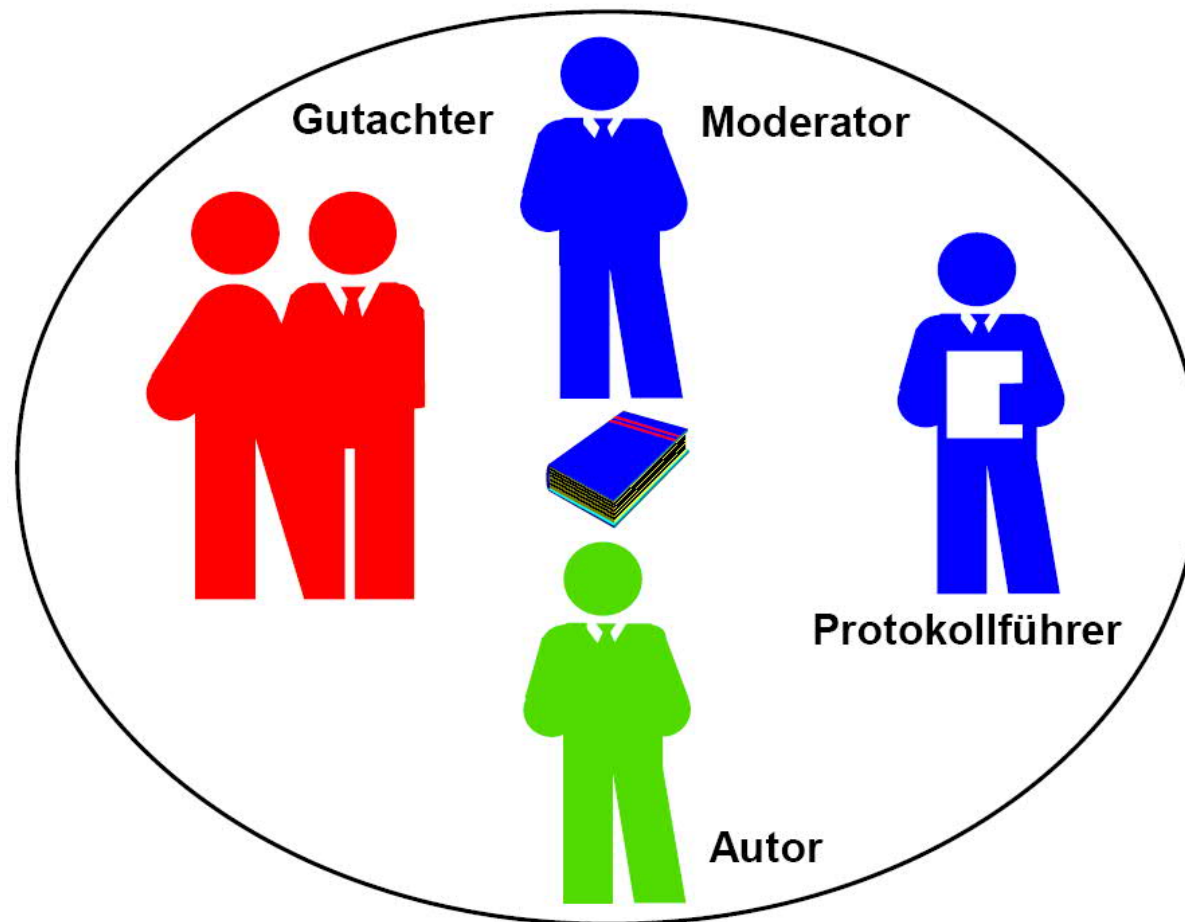


■ Ziel

- Fehler zu **entdecken**, nicht Fehler zu beheben.

Rollen beim Review

1(3)





Rollen beim Review

2(3)

■ Moderator

- plant und leitet das Review, bestimmt mit dem Autor (und dem Manager) das Review-Team.
- sollte fachkompetent und neutral sein.

■ Autor

- stellt den Prüfling bereit, ist verantwortlich für die Prüfbereitschaft des Prüflings.

■ Protokollführer

- dokumentiert die festgestellten Mängel (neutral).
- verfasst den Review-Bericht.



Rollen beim Review

3(3)

■ Gutachter

- ☐ kann als Experte den Prüfling **beurteilen**.
- ☐ ist verantwortlich für seine Vorbereitung.
- ☐ mindestens 2 , maximal 5 Gutachter pro Review.

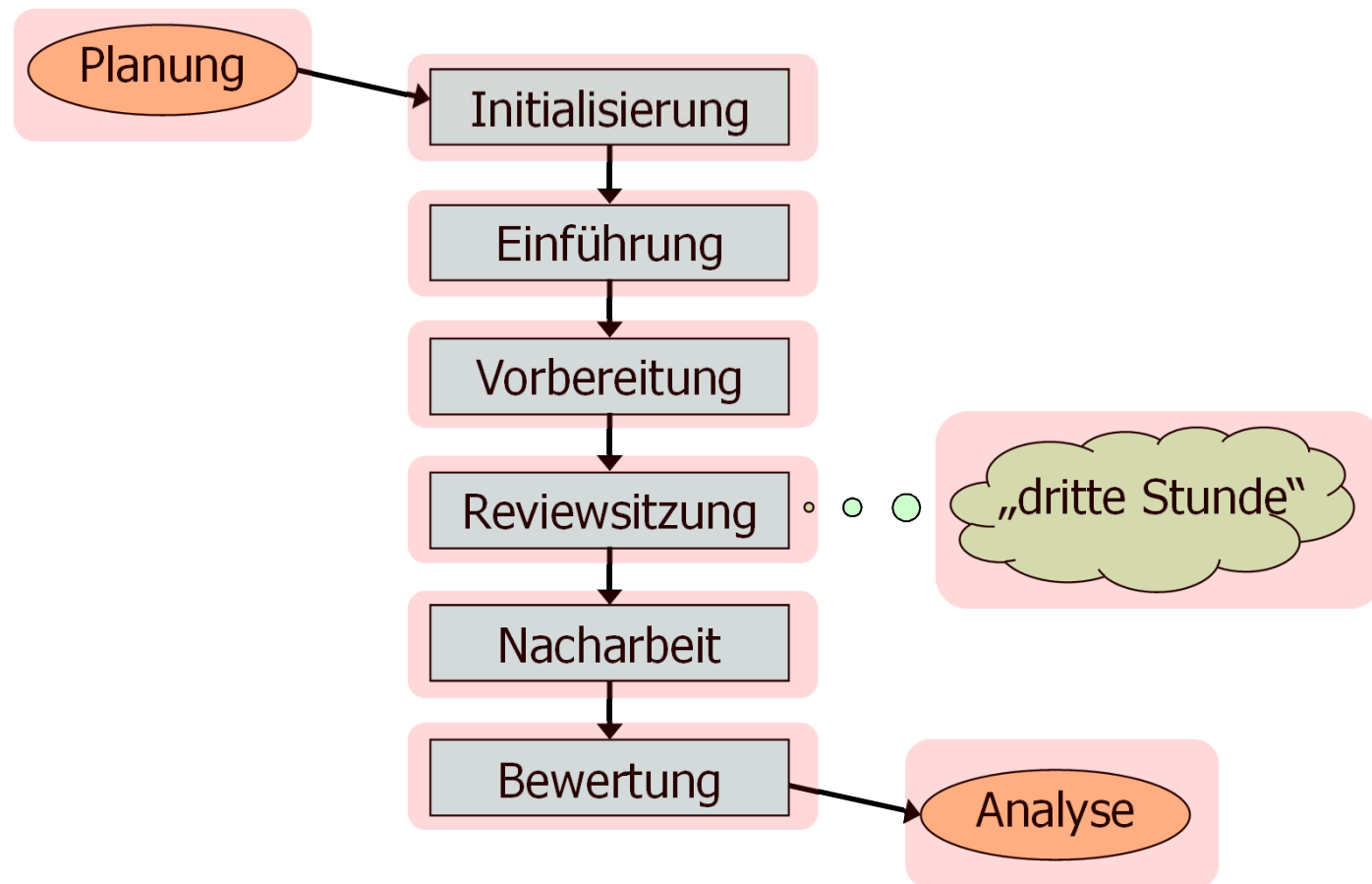
■ Review-Team

- ☐ Moderator, Gutachter, Autor, Protokollführer.
- ☐ verantwortlich für die **Bewertung/ Empfehlung**.

■ Manager

- ☐ hat Auftrag zur Entwicklung des Prüflings gegeben.
- ☐ muss Review und **Ressourcen** dazu einplanen.
- ☐ nimmt nicht am Review teil.
- ☐ **entscheidet** über Nachbearbeitung oder Freigabe.

Technisches Review – Ablauf





Technisches Review – Planung

- Der **Manager** ist für die Planung verantwortlich
 - Er plant das Review bei Vergabe des Arbeitspakets mit ein (10 - 20% des Gesamtaufwandes).
 - Er dokumentiert das Review im Projektplan.
 - Er bestimmt Autor und Moderator.
 - Er wählt die Aspekte (zusammen mit dem Autor), die im Review geprüft werden.
 - Er kümmert sich um die Freistellung der Gutachter.



Technisches Review - Initialisierung

- Autor meldet dem Manager die Fertigstellung des Prüflings.
- Manager und Moderator beurteilen die Prüfbereitschaft des Prüflings.
 - ☐ Prüfling von Autor entsprechend Richtlinien geprüft.
 - ☐ Prüfling im Versionsmanagement System, Bearbeitung gesperrt.
- Moderator legt Zeit und Ort für das Review fest.

Technisches Review - Einführung

- Versorgung der am Review beteiligten Personen mit allen benötigten Informationen.
- Schriftliche Einladung oder sofortiges erstes Treffen des Reviewteams, um über Bedeutung, Sinn und Zweck der durchzuführenden Reviewsitzung zu informieren.
 - Sind die beteiligten Personen mit dem Umfeld des zu prüfenden Dokumentes wenig vertraut, kann auf dem Treffen neben der kurzen Vorstellung des Prüfdokuments auch seine Einbettung in das Anwendungsgebiet dargestellt werden





Technisches Review - Einführung

- Neben dem Dokument, das einem Review unterzogen werden soll, müssen den beteiligten Personen weitere Unterlagen zur Verfügung stehen:
 - Dokumente, die herangezogen werden müssen, um zu entscheiden, ob eine Abweichung, ein Fehler oder eine korrekte Beschreibung des Sachverhalts vorliegt, also die Dokumente (z.B. Pflichtenheft, Richtlinien oder Standards), gegen die geprüft wird
 - Darüber hinaus sind Prüfkriterien (z.B. in Form von Checklisten) sehr sinnvoll, die ein strukturiertes Vorgehen unterstützen



Technisches Review - Vorbereitung

- Gute Vorbereitung der Gutachter ist zentrale Voraussetzung für ein erfolgreiches Review.
- Der Moderator
 - muss sicherstellen, dass die Gutachter Zeit zur Vorbereitung haben (Absprache mit dem Manager)
 - muss das Review abbrechen, wenn die Gutachter sich nicht vorbereiten können.
- Gutachter prüfen den Prüfling nach den zugeteilten Aspekten:
 - Kleinere Mängel werden im Dokument direkt markiert (Tippfehler, Abweichungen von Richtlinien etc.).
 - Alle anderen Mängel werden notiert.
 - Der Autor muss den Gutachtern für Fragen zur Verfügung stehen.



Technisches Review – Review-Sitzung

- Der Moderator leitet die Sitzung und **verhindert** durch das Einhalten von Regeln **negative Effekte**.
- Der Prüfling wird **Schritt für Schritt** durchgegangen.
- Gutachter bringen ihre Anmerkungen ein (präzise)
 - Feststellungen und Beobachtungen,
 - **keine Lösungen** oder Belehrungen.
- Protokollführer notiert die Anmerkungen für alle sichtbar für den **Review-Bericht**.
- Empfehlung an Manager ist die **gemeinsame Meinung** von Gutachtern und Autor.

Technisches Review – Review-Sitzung

■ Empfehlung im Review-Bericht

akzeptieren

akzeptieren nach Überarbeitung

nicht akzeptieren





Technisches Review – Review-Sitzung

- Neue Review-Sitzung kann verlangt werden, wenn die Empfehlung **nicht akzeptieren** heißt.
- Nachprüfung kann auf ein Mitglied des Review- Teams delegiert werden.
- Generell gilt:
 - ☐ maximal 2 Stunden Dauer.
 - ☐ konstruktive Grundeinstellung notwendig.
 - ☐ Probleme werden nicht gelöst oder Lösungen diskutiert.
 - ☐ Autor ist passiv; er räumt grobe Missverständnisse aus.



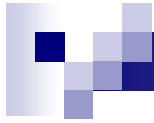
Technisches Review – „Dritte Stunde“

- Damit das Review effektiv und effizient ist, sind **Lösungsdiskussionen nicht gestattet.**
- Zweck der dritten Stunde (Review-Nachlese)
 - ☐ Gute Ideen und Lösungen sollen nicht verloren gehen.
 - ☐ Entspannte Diskussion mit dem Autor ist möglich.
 - ☐ Manöverkritik kann geübt werden:
 - Was war richtig/gut?
 - Was war falsch/schlecht?



Technisches Review – Nachbearbeitung/Bewertung

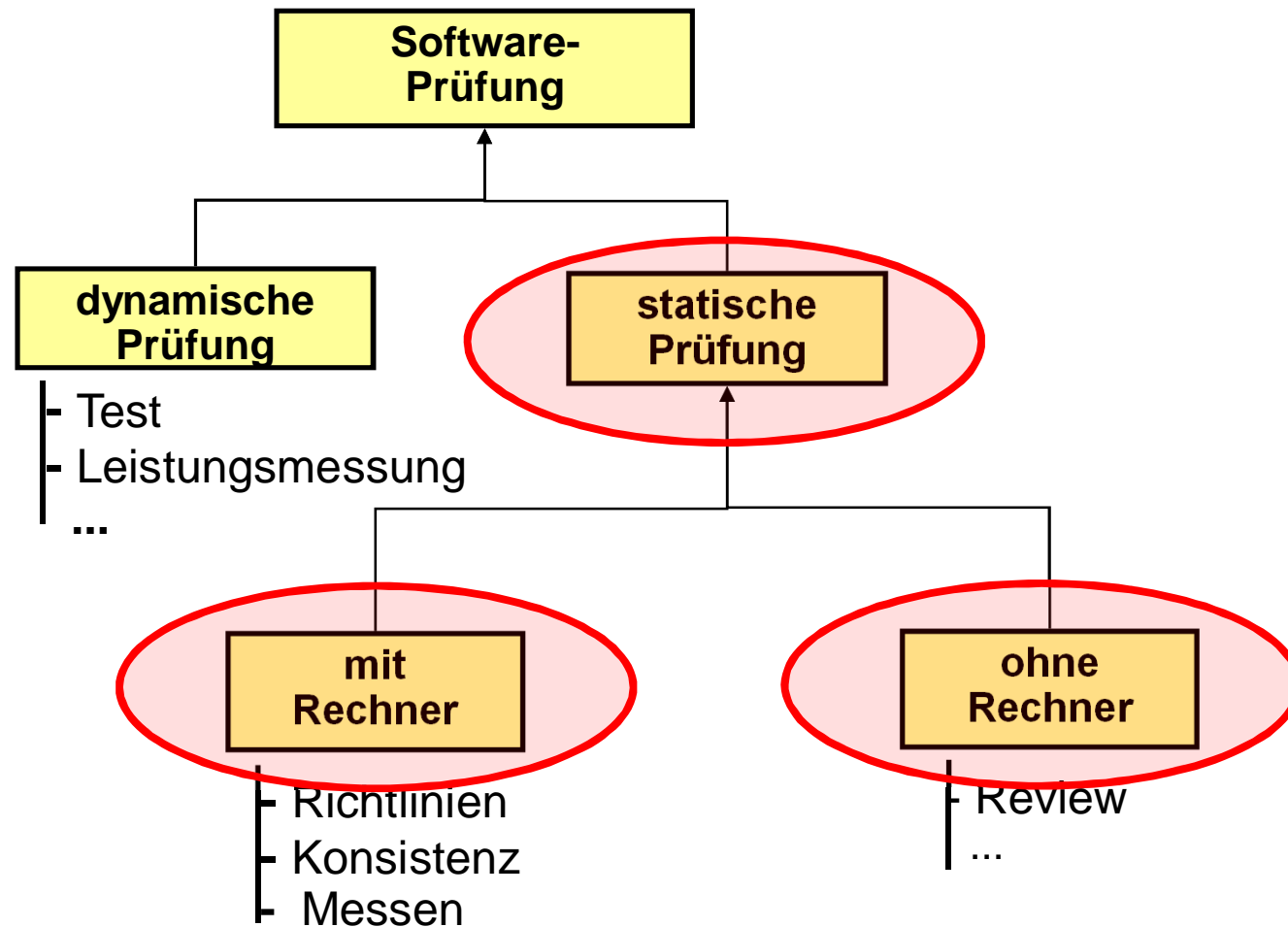
- Manager **entscheidet** aufgrund des Review-Berichts
 - Überarbeitung oder Freigabe.
 - Bei Überarbeitung muss er notwendige Ressourcen zur Verfügung stellen.
 - Falls er den Prüfling freigibt, und auf eine notwendige Überarbeitung verzichtet → seine Verantwortung.
- Autor überarbeitet das Dokument gemäß der Befunde
 - nur das verändern, was **gefordert** wurde.
 - keine sonstigen Verbesserungen.
- Moderator überprüft Review-Bericht:
 - Kleinere Mängel → Rückgabe an den Manager.
 - Größere Mängel → Einberufung einer neuen Review-Sitzung.



Inhalt

- Qualitätssicherung
 - Qualitätssicherung im Überblick
 - **Statische Prüfungen**
 - Review
 - **Statische Analysen**
 - **Kontrollflussanalyse**
 - Datenflussanalyse
 - Dynamische Prüfungen
 - Glass-Box Testing
 - Black-Box Testing
 - Testen im Software-Erstellungsprozess

Qualitätssicherung im Überblick (3)



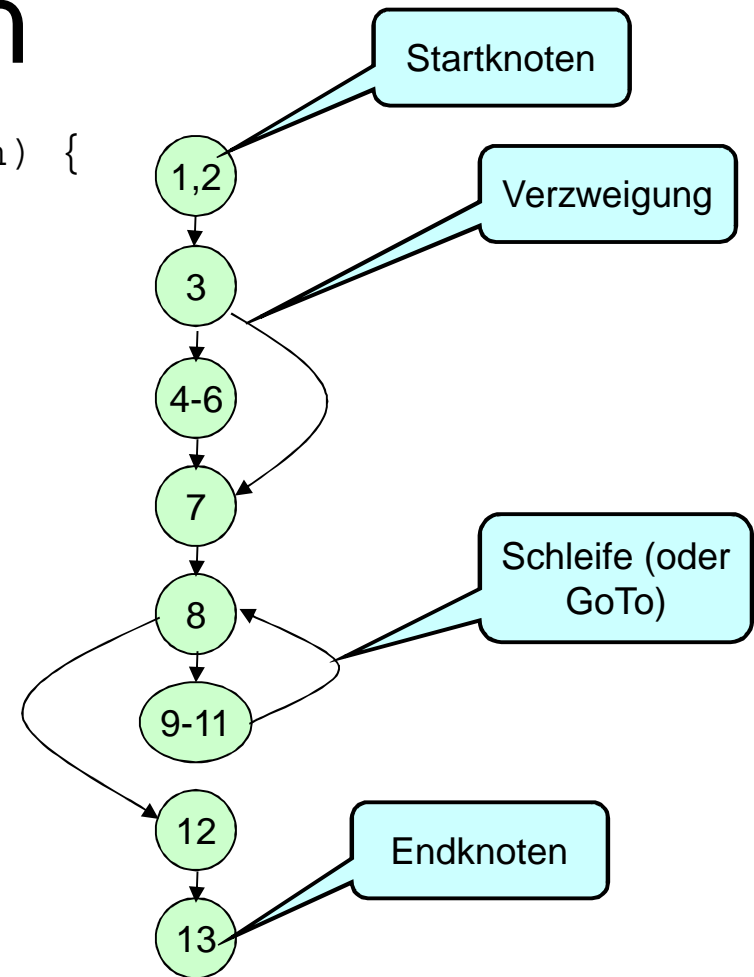


Statische Analyse von Programmcode

- Werkzeuge:
 - ☐ Compiler,
 - ☐ Analysatoren.
- Beispiele für Fehler bzw. fehlerträchtige Situationen, die mit der statischen Analyse im Programmcode nachgewiesen werden können:
 - ☐ Verletzung der Syntax,
 - ☐ Implizite Typ-Anpassungen (Casting), unbekannte (externe) Namen,
 - ☐ Abweichungen von Konventionen und Standards,
 - ☐ Kontrollflussanomalien,
 - ☐ Datenflussanomalien,
 - ☐ etc.

Beispiel eines Kontrollflussgraphen

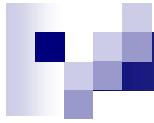
```
1.  public int ggt-euclid (int m, int n) {  
2.      int r;  
3.      if (n > m) {  
4.          r = m;  
5.          m = n;  
6.          n = r;  
7.      }  
8.      while (r != 0) {  
9.          m = n;  
10.         n = r;  
11.         r = m % n;  
12.     }  
13.     return n;  
14. }
```





Kontrollflussanalyse

- Durch die Anschaulichkeit des Kontrollflussgraphen lassen sich die Abläufe durch ein Programmstück leicht erfassen.
 - Sind Teile des Graphen oder der ganze Graph sehr unübersichtlich und die Zusammenhänge und der Ablauf kaum nachvollziehbar, sollte eine Überarbeitung des Programmtextes erfolgen, da komplexe Ablaufstrukturen oft fehlerträchtig sind
- **Kontrollflussanomalie:** Statisch feststellbare Unstimmigkeit beim Ablauf des Testobjekts (z.B. nicht erreichbare Anweisungen):
 - Sprünge aus Schleifen heraus
 - Sprünge in Schleifen hinein
 - Programmstücke mit mehreren Ausgängen
- Diese Unstimmigkeiten müssen keine Fehler sein, widersprechen aber den Grundsätzen der strukturierten Programmierung.
- **Voraussetzung:** Graph ist nicht manuell, sondern von einem Werkzeug erstellt, so dass eine eins-zu-eins–Abbildung zwischen Programmtext und Graph gewährleistet ist.

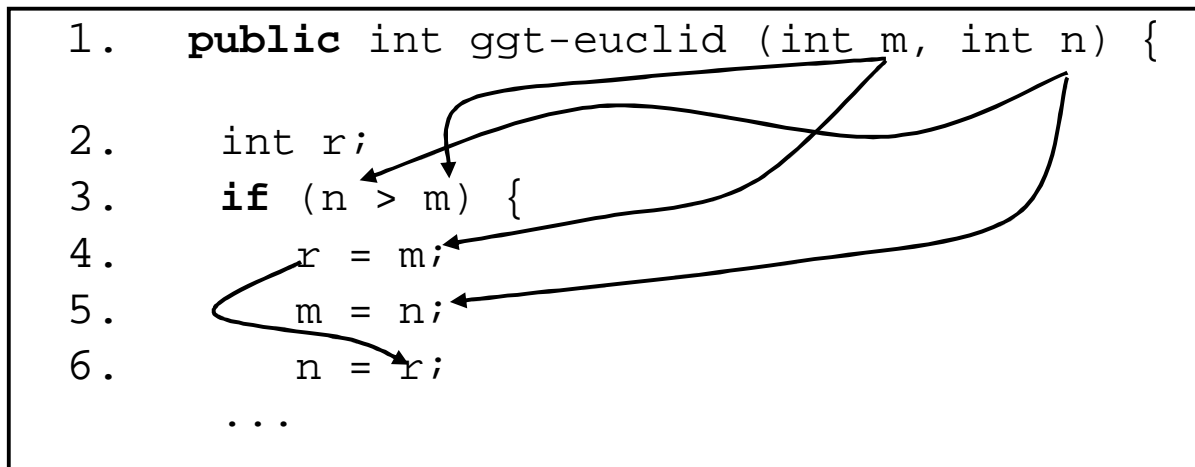


Inhalt

- Qualitätssicherung
 - Qualitätssicherung im Überblick
 - **Statische Prüfungen**
 - Review
 - Statische Analysen
 - Kontrollflussanalyse
 - **Datenflussanalyse**
 - Dynamische Prüfungen
 - Glass-Box Testing
 - Black-Box Testing
 - Testen im Software-Erstellungsprozess

Datenflussanalyse

- Verwendung von Daten auf »Pfaden« durch den Programmcode.



- Nicht immer können Fehler nachgewiesen werden, oft wird in diesem Zusammenhang dann ebenfalls von Anomalien oder Datenflussanomalien gesprochen. Zum Beispiel:
 - ☐ Lesen einer Variablen ohne vorherige Initialisierung oder
 - ☐ Nicht-Verwendung eines zugewiesenen Wertes einer Variablen



Datenfluss-Anomalien 1(2)

- Analysiert wird die Verwendung jeder einzelnen Variablen.
- Drei Verwendungen oder Zustände der Variablen werden unterschieden:
 - **Undefiniert (u)**
 - Eine Variable hat keinen definierten Wert (z.B. am Programmanfang oder wenn sie freigegeben oder ihr Gültigkeitsbereich verlassen wird).
 - **Definiert (d)**
 - Der Variablen wird ein Wert zugewiesen.
 - **Referenziert (r)**
 - Der Wert der Variablen wird gelesen bzw. verwendet.



Datenfluss-Anomalien 2(2)

- Drei Arten von Datenflussanomalien
 - **ur-Anomalie**
 - Ein undefinierter Wert (**u**) einer Variablen wird auf einem Programmpfad gelesen (**r**).
 - **du-Anomalie**
 - Die Variable erhält einen Wert (**d**) der allerdings ungültig (**u**) wird, ohne dass er zwischenzeitlich verwendet wurde.
 - **dd-Anomalie**
 - Die Variable erhält auf einem Programmpfad ein zweites Mal einen Wert (**d**), ohne dass der erste Wert (**d**) verwendet wurde.
- Zusammengefasst ist eine Datenfluss-Anomalie die
 - Verwendung einer Variablen ohne vorherige Initialisierung, oder die
 - die Nicht-Verwendung des Wertes einer Variablen.

Beispiel: Datenfluss-Anomalien

```
1 void Tausch (int min, int max) {  
2     int hilf;  
3     if (min > max) {  
4         max = hilf;  
5         max = min;  
6         hilf = min;  
7     }  
8 }
```

// u(hilf) (in Java 0)
// r(min, max)
// d(max), r(hilf)
// d(max), r(min)
// d(hilf), r(min)
// u(hilf)

dd (max)

ur (hilf)

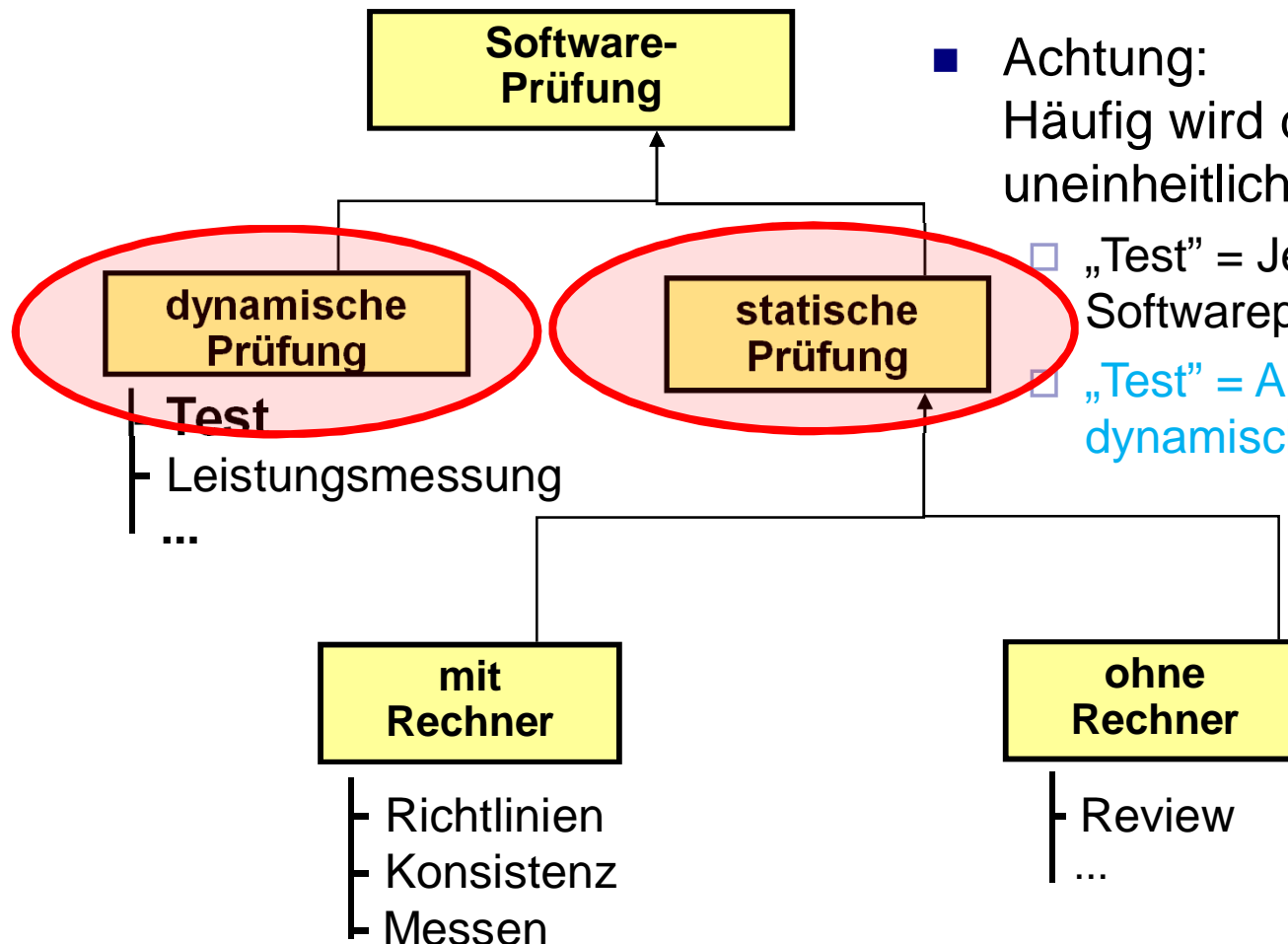
du (hilf)



Inhalt

- Qualitätssicherung
 - Qualitätssicherung im Überblick
 - Statische Prüfungen
 - Review
 - Statische Analysen
 - Kontrollflussanalyse
 - Datenflussanalyse
 - **Dynamische Prüfungen**
 - Glass-Box Testing
 - Black-Box Testing
 - Testen im Software-Erstellungsprozess

Dynamische-Prüfungen (Tests) von Software



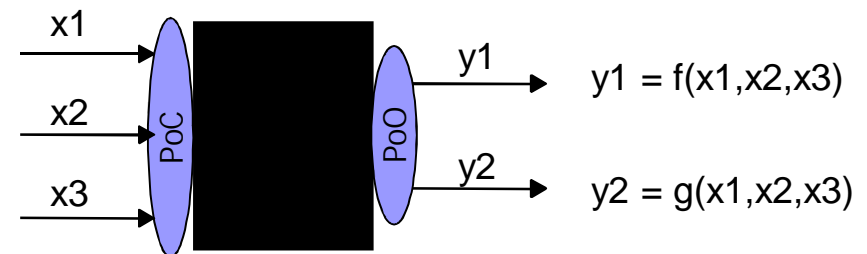
- Achtung:
Häufig wird der Begriff „Test“
uneinheitlich verwendet:

- „Test“ = Jede Art von
Softwareprüfung,
- „Test“ = **Ausschließlich**
dynamische Prüfung (hier).

Grundsätzliche Testverfahren

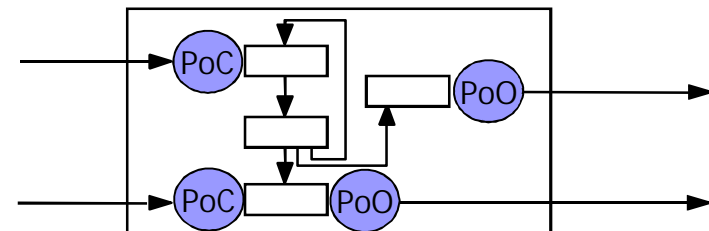
■ Funktionsorientiert (**Black-Box-Test**):

- Testfall-Auswahl aufgrund der Spezifikation.
- Programmstruktur (Quelltext) kann unbekannt sein.
- PoC und PoO liegen außerhalb des Testobjekts

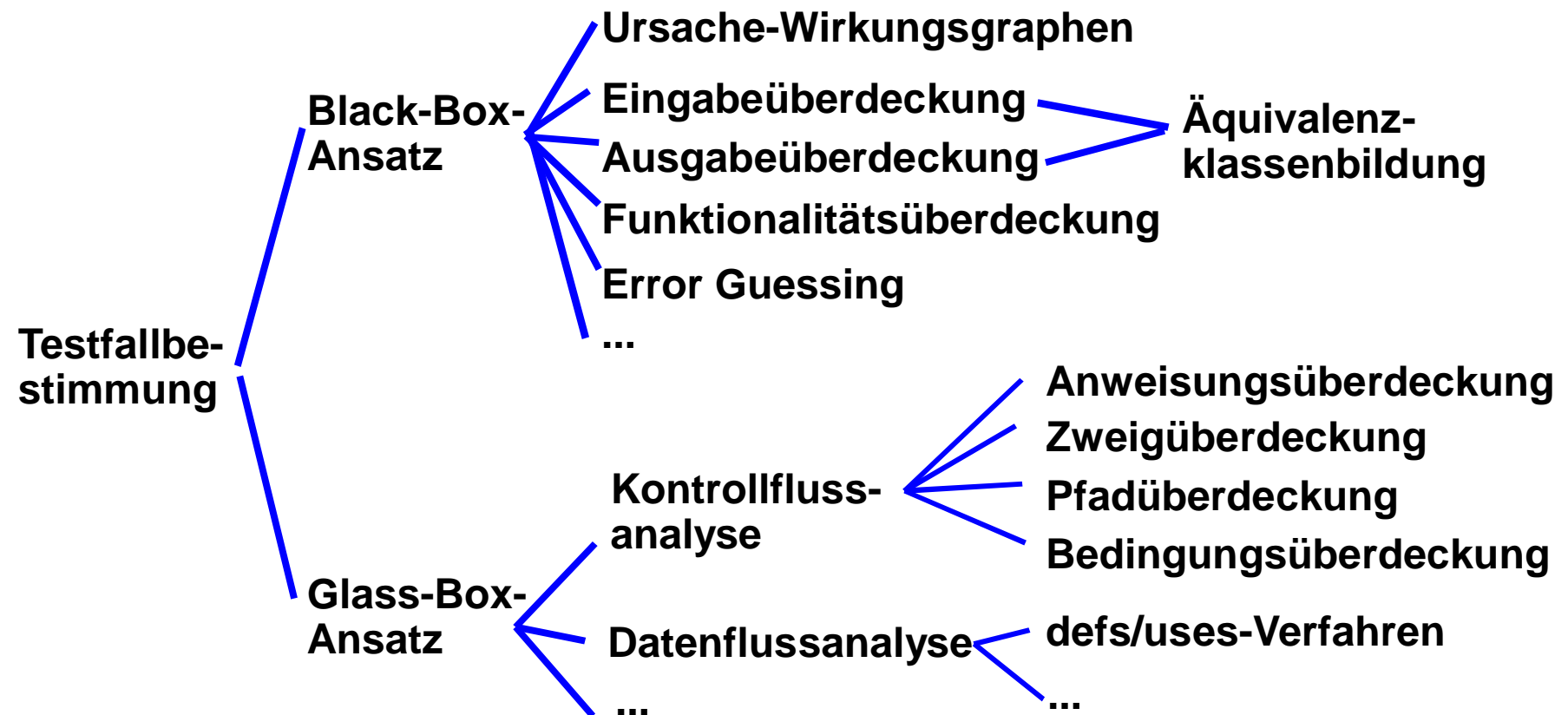


■ Strukturorientiert (**Glass-Box-Test, White-Box-Test**):

- Testfall-Auswahl aufgrund der Programmstruktur.
- Quelltext muss bekannt sein.
- Während der Testläufe wird der innere Ablauf des Testobjekts analysiert (PoO liegt innerhalb des Testobjekts)
- Eingriffe in den Ablauf des Testobjekts sind möglich (PoC kann innerhalb des Testobjekts liegen)



Bestimmung von Testfällen

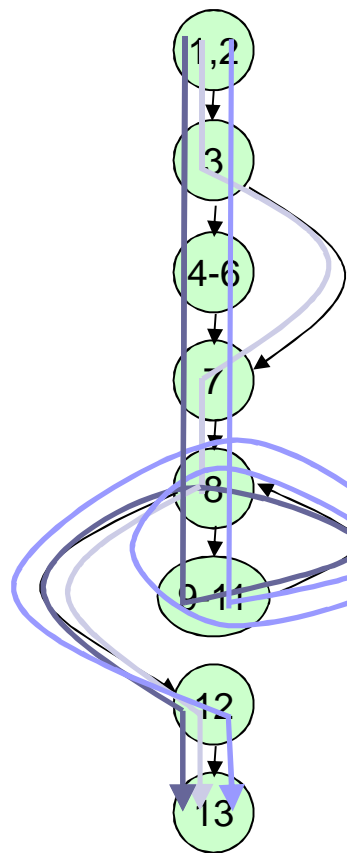




Inhalt

- Qualitätssicherung
 - Qualitätssicherung im Überblick
 - Statische Prüfungen
 - Review
 - Statische Analysen
 - Kontrollflussanalyse
 - Datenflussanalyse
 - **Dynamische Prüfungen**
 - **Glass-Box Testing**
 - Black-Box Testing
 - Testen im Software-Erstellungsprozess

Beispiel für Glass-Box-Ansatz: Kontrollfluss-basierte Testauswahl

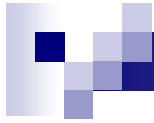


```

1.  public int ggt-euclid (int m, int n) {
2.      int r;
3.      if (n > m) {
4.          r = m;
5.          m = n;
6.          n = r;
7.      }
8.      r = m % n;
9.      while (r != 0) {
10.         m = n;
11.         n = r;
12.         r = m % n;
13.     }
14.     return n;
15. }

```

- Pfad 1 (Anweisungsüberdeckung):
 - 1, 2, 3, 4-6, 7, 8, 9-11, 8, 12, 13
- Testfall 1:
 - m = 4, n = 6
- Pfad 2 (+ Pfad 1 = Zweigüberdeckung):
 - 1, 2, 3, 7, 8, 12, 13
- Testfall 2:
 - m = 6, n = 3
- Pfad 3 - ...: (Pfadüberdeckung)
 - 1, 2, 3, 4-6, 7, 8, 9-11, (8, 9-11)⁺, 8, 12, 13
- Testfall 3:
 - m = 15, n = 36
- Testfall 4:
 - m = 10, n = 36
- ...

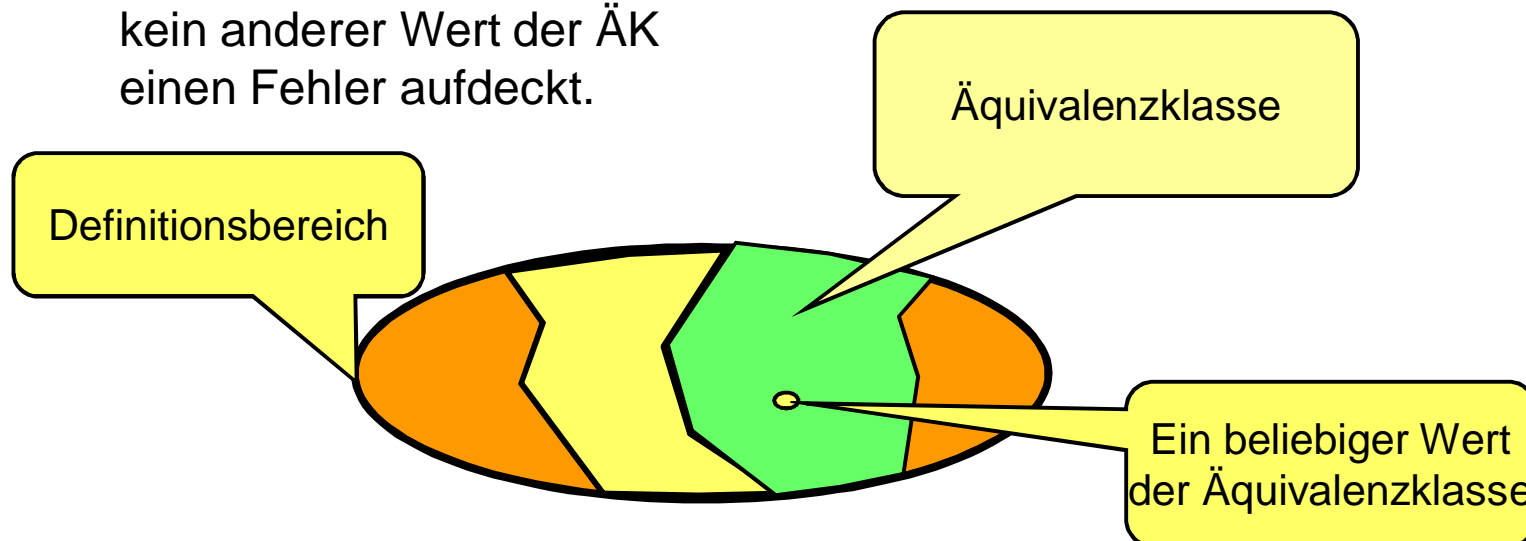


Inhalt

- Qualitätssicherung
 - Qualitätssicherung im Überblick
 - Statische Prüfungen
 - Review
 - Statische Analysen
 - Kontrollflussanalyse
 - Datenflussanalyse
 - **Dynamische Prüfungen**
 - Glass-Box Testing
 - **Black-Box Testing**
 - Testen im Software-Erstellungsprozess

Beispiel für Black-Box-Ansatz: Äquivalenzklassenbildung

- Die Definitionsbereiche der Eingaben werden so in Äquivalenzklassen (ÄK) partitioniert, dass alle Werte einer Klasse äquivalentes Verhalten des Prüflings ergeben.
 - Wenn ein Wert der ÄK einen Fehler aufdeckt, wird erwartet, dass auch jeder andere Wert der ÄK diesen Fehler aufdeckt.
 - Wenn ein Wert der ÄK keinen Fehler aufdeckt, wird erwartet, dass auch kein anderer Wert der ÄK einen Fehler aufdeckt.





Beispiel für Black-Box-Ansatz: Äquivalenzklassenbildung

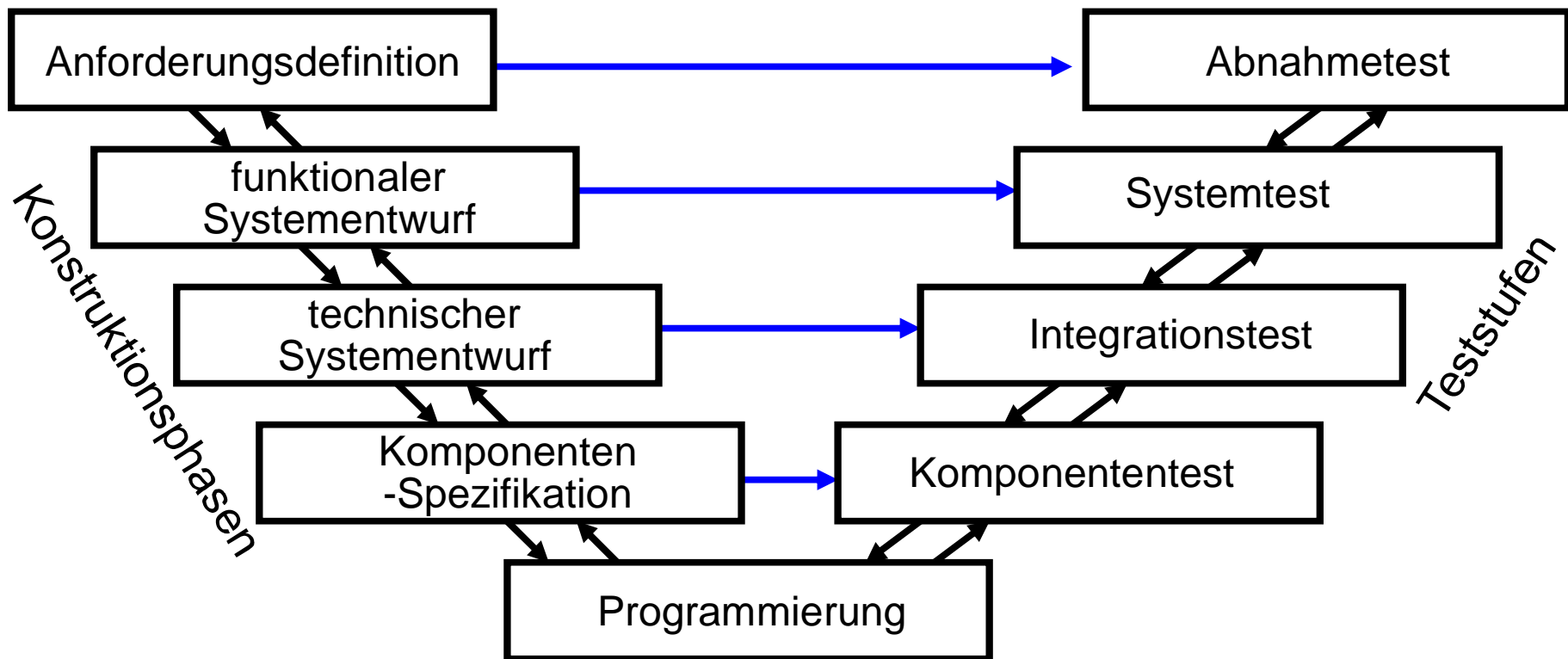
- Schritt 1: **Aufstellung von Eingabebedingungen**
 - Schritt 2: **Bildung von Äquivalenzklassen**
 - Schritt 3: **Äquivalenzklassen identifizieren**
 - Schritt 4: **Testfälle definieren (gültige Ä-Klassen)**
 - Schritt 5: **Testfälle definieren (ungültige Ä-Klassen)**
- *Anwendung für DAMPF Beispiel auf separaten Folien.*



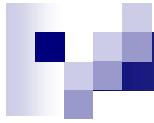
Inhalt

- Qualitätssicherung
 - Qualitätssicherung im Überblick
 - Statische Prüfungen
 - Review
 - Statische Analysen
 - Kontrollflussanalyse
 - Datenflussanalyse
 - **Dynamische Prüfungen**
 - Glass-Box Testing
 - Black-Box Testing
 - **Testen im Software-Erstellungsprozess**

Testen im V-Vorgehensmodell



Testfallauswahl basiert auf den entsprechenden Dokumenten →



Inhalt

- Software Qualität
- Qualitätssicherung
- **Qualitätsaspekte für Softwareprojekte**
- Lernziele



Qualitätsaspekte für Software-Projekte

- Prozessqualität

- ☐ sagt etwas über die Voraussetzungen für ein Projekt aus.

- Projektqualität

- ☐ sagt etwas über ein konkretes Projekt aus.

- Produktqualität

- ☐ sagt etwas über das entwickelte Produkt aus.

- ☐ gliedert sich in

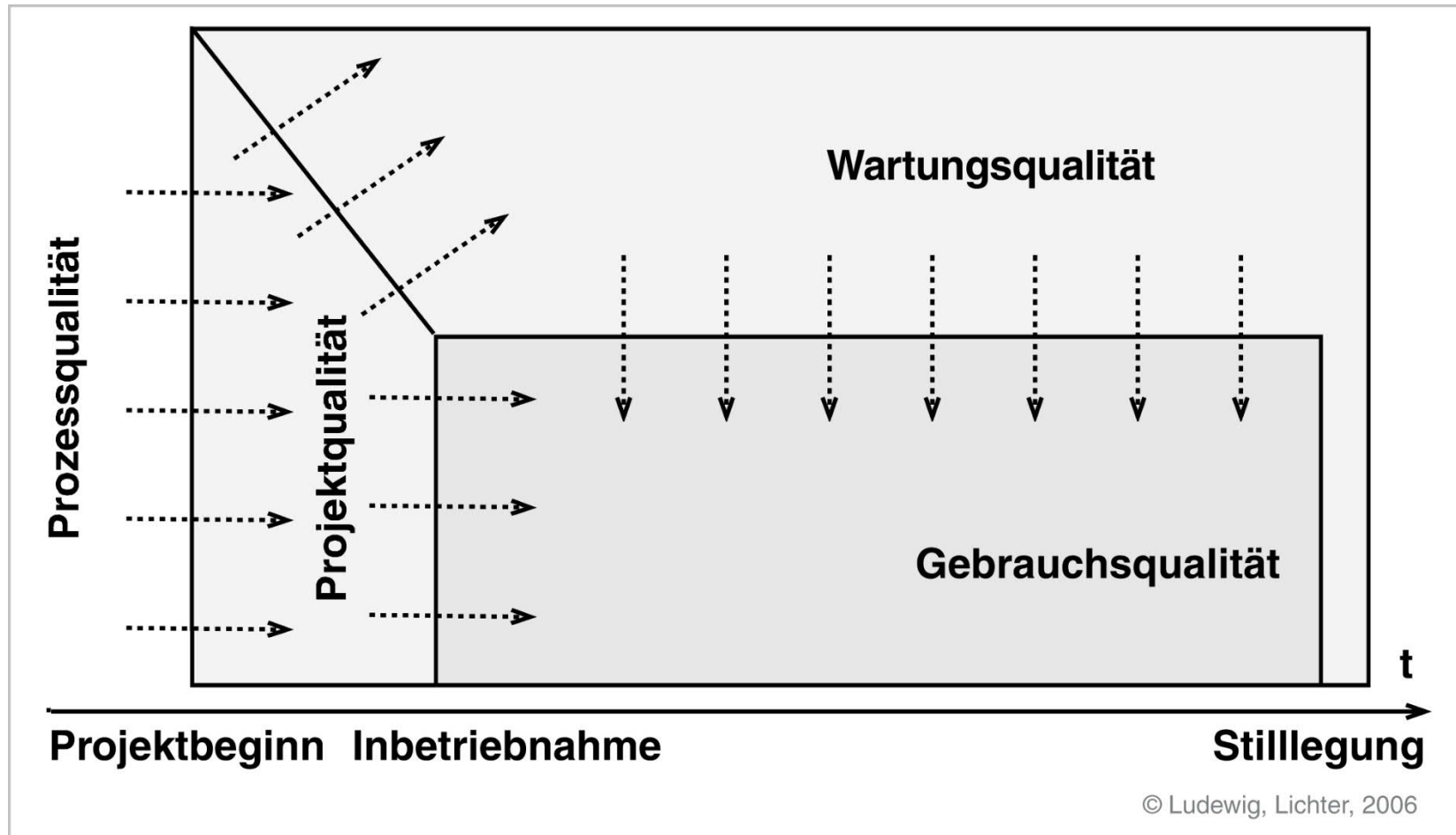
- Interne Qualität (auch Wartungsqualität)

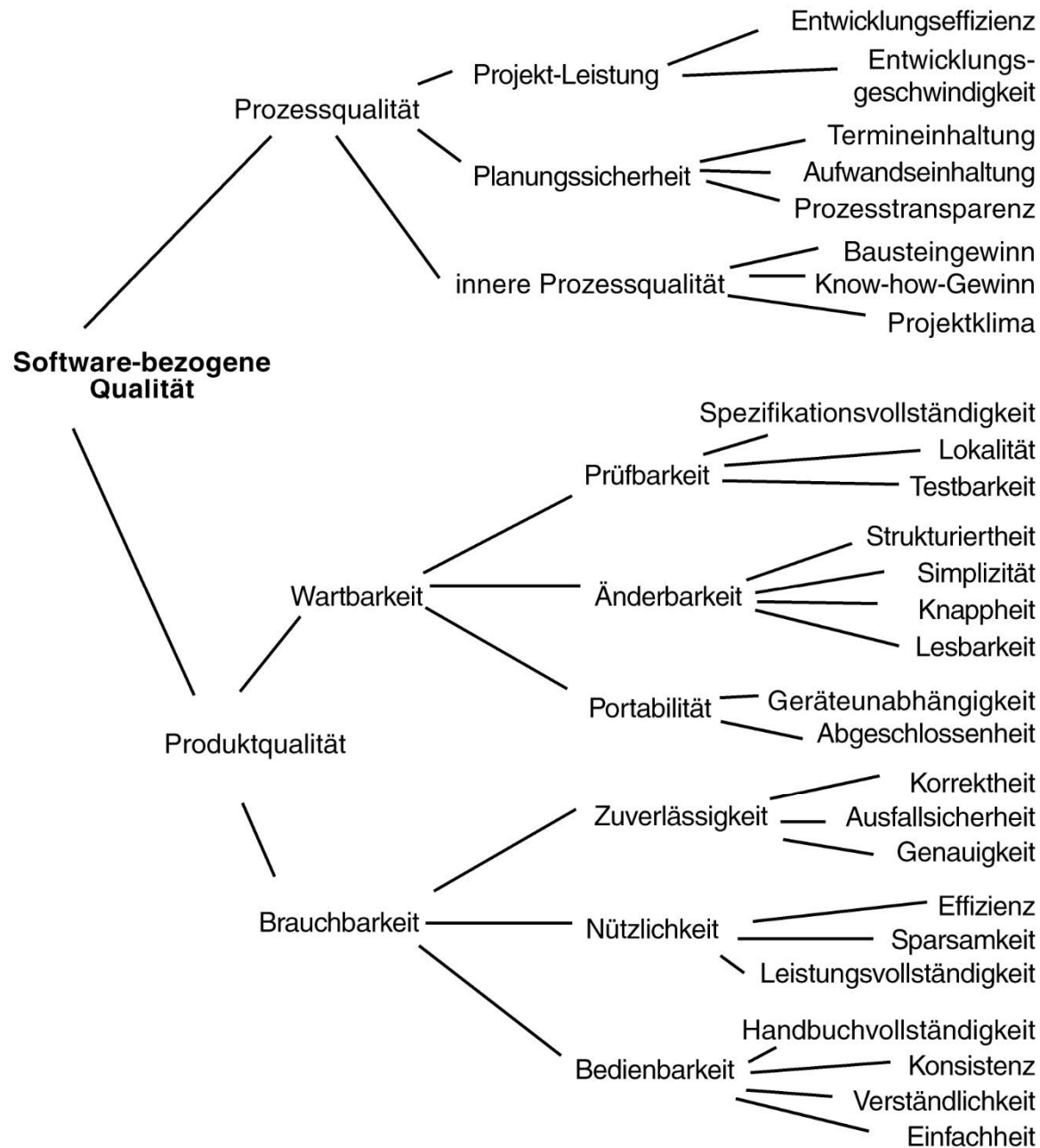
- ☐ sichtbar für Entwickler des Softwaresystems

- Externe Qualität (auch Gebrauchsqualität)

- ☐ sichtbar für die Benutzer des Softwaresystems

Zusammenhang zwischen den Qualitätsaspekten





Qualitätskriterien für Software und Softwareprojekte



Inhalt

- Software Qualität
- Qualitätssicherung
- Qualitätsaspekte für Softwareprojekte
- **Lernziele**



Lernziele

1(3)

■ Software Qualität

- ☐ Was ist Software Qualität?
- ☐ Welche typischen Qualitätskriterien gibt es für Software Qualität?
- ☐ Was drückt das Teufelsquadrat nach Sneed aus?



Lernziele

2(3)

■ Qualitätssicherung

- ☐ Welche drei prinzipiellen Qualitätssicherungsmaßnahmen gibt es?
- ☐ Was ist der Unterschied zwischen statischer und dynamischer analytischer Qualitätssicherung?
- ☐ Was ist ein Review und wie wird es durchgeführt?
- ☐ Was ist Kontrollflussanalyse und wie funktioniert sie?
- ☐ Was ist Datenflussanalyse und wie funktioniert sie?
- ☐ Welche zwei prinzipiellen Testverfahren kennen sie?
- ☐ Wie funktioniert die Kontrollflussbasierte Auswahl von Testfällen beim Glass-Box Testen?
- ☐ Wie funktioniert die Auswahl von Testfällen basierend auf der Äquivalenzklassenbildung?
- ☐ Wo wird im Softwareentwicklungsprozess getestet?



Lernziele

3(3)

- Welche Qualitäten existieren neben den produktbezogenen Qualitäten in der Softwareentwicklung?