

Willkommen zur Vorlesung  
*Methodische Grundlagen des Software-  
Engineering*

im Sommersemester 2011

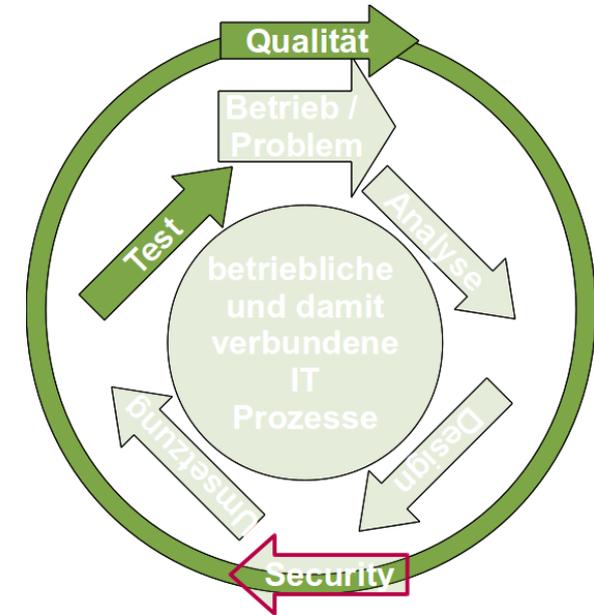
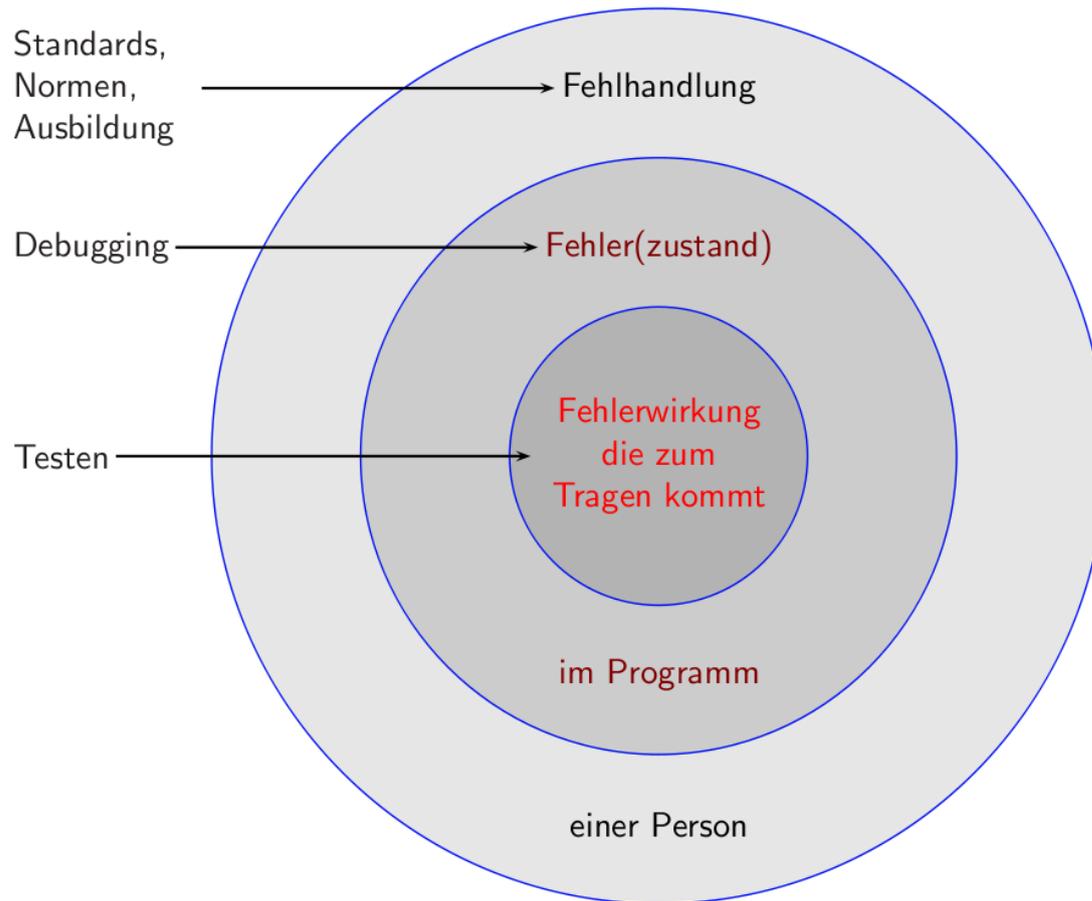
Prof. Dr. Jan Jürjens

TU Dortmund, Fakultät Informatik, Lehrstuhl XIV

# 12. Testen, Grundlagen und Begriffe

[inkl. Beiträge von Stefan Berlik, Universität Siegen]

# Einordnung Grundlagen Testen

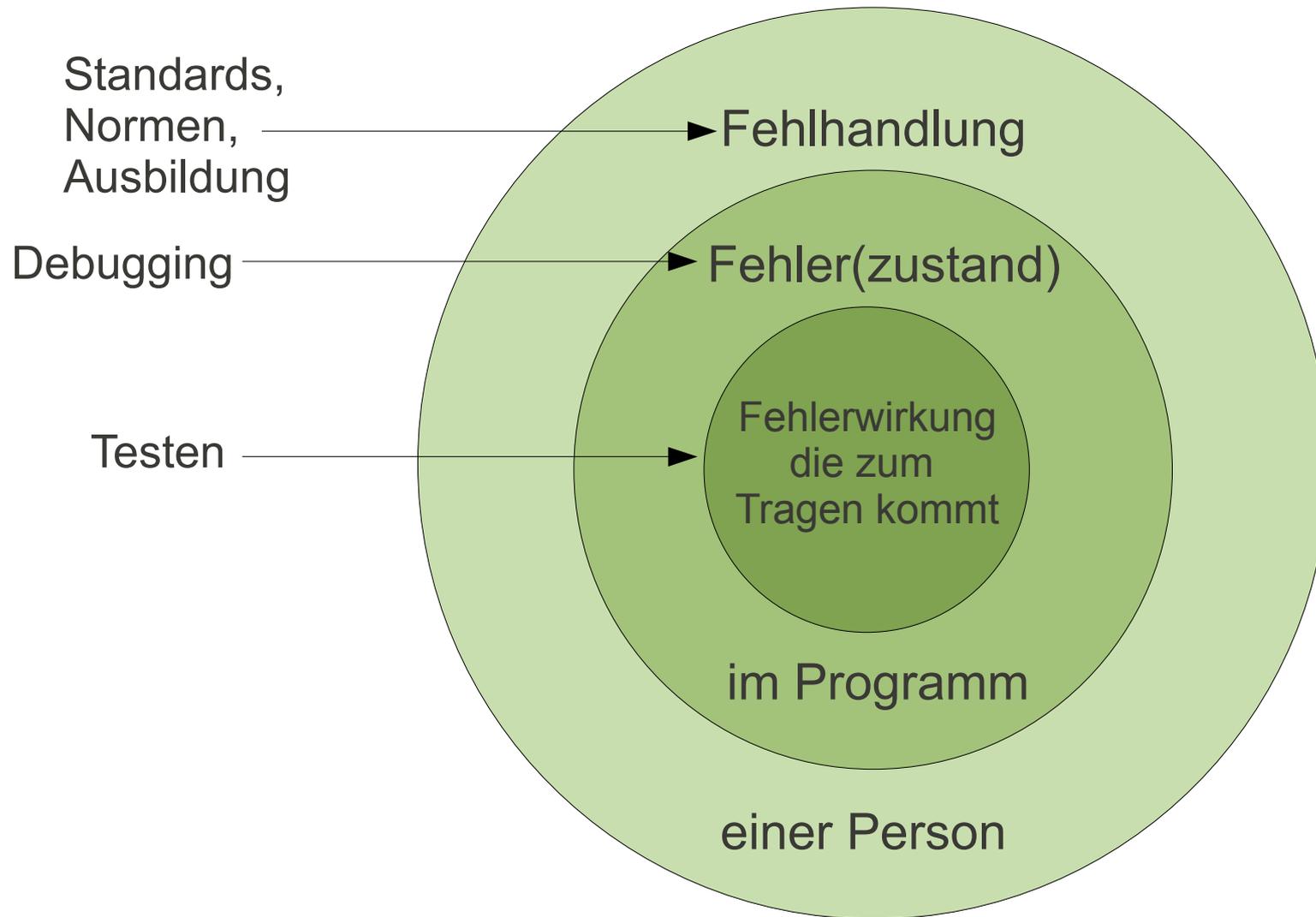


- Business Prozesse
- Qualitätsmanagement
- **Testen**
  - **Grundlagen**
  - Blackbox / Whitebox Testing
  - Beispiele: Junit, Fuzzing(Spike)
- Sicherheit
- Sicheres Software Design

- Wann liegt ein **nicht anforderungskonformes** Verhalten des Systems vor?
- Eine Situation kann nur dann als **fehlerhaft** eingestuft werden, wenn vorab festgelegt wurde, wie die **erwartete korrekte, also nicht fehlerhafte Situation** aussehen soll
- Ein Fehler ist demnach die Nichterfüllung einer festgelegten Anforderung, d.h eine **Abweichung** zwischen dem **Ist-Verhalten**, das während der Ausführung der Tests oder des Betriebs festgestellt wurde, und dem **Soll-Verhalten**, wie es in der Spezifikation oder den Anforderungen festgelegt wurde.
- Ein **Mangel** liegt vor, wenn eine gestellte Anforderung oder eine berechnete Erwartung nicht angemessen erfüllt wird.

- Jeder Software-Fehler oder -Mangel ist seit dem Zeitpunkt der Entwicklung der Software vorhanden. Er kommt jedoch erst bei der Ausführung der Software zum Tragen.  
[Vergleich: Hardware-Fehler können auch erst zur Laufzeit entstehen (z.B. durch Einwirken von Strahlung auf Chips).]
- Dieser Sachverhalt wird als **Fehlerwirkung (failure)** (auch als Fehlfunktion oder äußerer Fehler) bezeichnet.
- Die Ursache hierfür ist ein **Fehlerzustand (fault)** in der Software (auch als Defekt oder innerer Fehler bezeichnet).
- Dessen Ursache liegt in einer **Fehlhandlung (error)** einer Person (z.B. Programmierer).

# Entstehung von Fehlern & Gegenmaßnahmen



- Testen ist eine **stichprobenartige** Prüfung
- Testen ist **nicht** Debugging
- **Fehlerfreiheit** ist durch Testen **nicht mit letzter Sicherheit demonstrierbar** (Testen jedes einzelnen möglichen Systemablaufes im Allgemeinen nicht machbar).
- „Kein umfangreiches Softwaresystem ist fehlerfrei“

- **Frage:** Was ist Software-Qualität?
- **Antwort:** Die Erfüllung Qualitätszielen / Umsetzung von Qualitätsmerkmalen
- „**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)
- **Qualitätsmerkmale:** (DIN ISO 9126)
  - Funktionalität
  - Zuverlässigkeit
  - Benutzbarkeit
  - Effizienz
  - Änderbarkeit
  - Übertragbarkeit

- Testen ist ökonomisch sinnvoll, solange die Kosten für das Finden und Beseitigen eines Fehlers im Test niedriger sind als die Kosten, die mit dem Auftreten eines Fehlers bei Nutzung verbunden sind.“  
[M. Pol, T. Koomen, A. Spillner: Management und Optimierung des Testprozesses. dpunkt-Verlag, 2. Auflage, 2002.]
- Erfolgreiches Testen senkt die **Kosten**

**Validation:** Prüfung, ob ein Entwicklungsergebnis die individuellen Anforderungen bezüglich einer speziellen beabsichtigten Nutzung erfüllt.

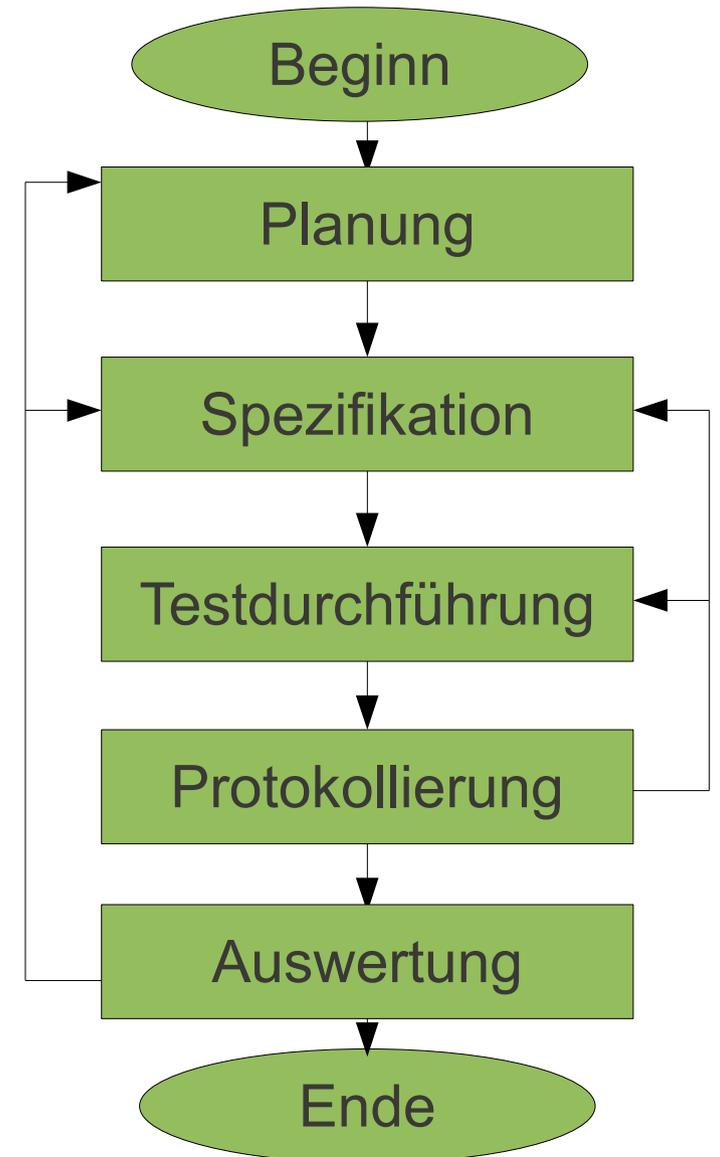
Wurde das **richtige System** realisiert?

**Verifikation:** Prüfung, ob die Ergebnisse einer Entwicklungsphase die Vorgaben der Phaseneingangsdokumente erfüllen.

Wurde das System **richtig realisiert**?

- Was ist der Unterschied zwischen Testen und Debugging?

- Testplanung
- Testspezifikation
- Testdurchführung
- Testprotokollierung
- Auswertung und Bewertung des Testendes



- Planung der Ressourcen
  - Mitarbeiter
  - Werkzeuge
  - Geräte usw.
- Festlegung der Teststrategie
  - Wie **intensiv** sollen welche **Systemteile** getestet werden?
  - Welche **Testmethoden** sollen verwendet werden?
  - Welcher **Überdeckungsgrad** soll erreicht werden?
  - **Priorisierung** der Tests (unter Berücksichtigung des **Risikos** im Fehlerfall)
  - **Werkzeugunterstützung** einplanen (Einsatz, Beschaffung, Einarbeitung)
- Festzulegen im Testkonzept (engl. test plan)

- Unter Anwendung der gewählten Testmethoden sind die entsprechenden **Testfälle** zu **spezifizieren**
- Grundlage sind alle Dokumente, aus denen Anforderungen an das Testobjekt hervorgehen (**Testbasis**)
- Unterscheidung: **logische** Testfälle vs. **konkrete** Testdaten
- **Testdaten** umfassen mehr als nur Eingabedaten:
  - **Vorbedingung** (Ausgangsbedingung)
  - **Randbedingungen**, die einzuhalten sind
  - Vor der Testdurchführung ist festzulegen, welche Ergebnisse bzw. welches Verhalten erwartet wird (**Soll-Verhalten** bzw. -Ergebnis)
  - **Nachbedingung**, die erfüllt sein muss



- Drei Gruppen von Testfälle lassen sich unterscheiden
  - Testfälle zur Prüfung der spezifizierten und vom Testobjekt zu liefernden Ergebnisse und Reaktionen  
**Positiv-Test:** erwartete Eingaben bzw. erwartete Bedienung (engl. normal)
  - Testfälle die die **spezifizierte** Behandlung von Ausnahme-und Fehlersituationen überprüfen  
**Negativ-Test:** erwartete Fehleingaben bzw. erwartete Fehlbedienung (engl. exceptional)
  - Testfälle zur Prüfung der Reaktion des Testobjekts auf ungültige und **unerwartete** Eingaben bzw. Randbedingungen, für die **keine Ausnahmebehandlungen** (engl. exception handling) **spezifiziert** wurden  
**Negativ-Test:** unerwartete Fehleingaben bzw. unerwartete Fehlbedienung (engl. catastrophic)

- Vorab auf Vollständigkeit der zu testenden Teile prüfen
- Vorabprüfung der Hauptfunktionen  
Treten hier bereits Fehlerwirkungen auf, ist ein tiefes Einsteigen in den Test wenig sinnvoll
- Die Durchführung der Tests ist exakt und vollständig zu protokollieren
  - Stellt die Nachvollziehbarkeit auch für nicht direkt beteiligte Personen (z.B. den Kunden) sicher
  - Weist nach, ob die geplante Teststrategie tatsächlich umgesetzt wurde
  - Aus dem Testprotokoll muss hervorgehen, welche Teile wann, von wem, wie intensiv und mit welchem Ergebnis getestet wurden

- Wurde ein Fehler gefunden? Ist eine Fehlerwirkung nachweisbar?

→ Nichtübereinstimmung von Soll-und Ist-Ergebnis!

Allerdings können auch:

- das Soll-Ergebnis
- die Testumgebung
- die Testfallspezifikation

**falsch** bzw. **fehlerhaft** sein

- **Fehlerklasse** bestimmen, **Priorität** für die Beseitigung festlegen

- **Das Testen beenden?**

- Gewünschter Überdeckungsgrad erreicht?
- Problem: Unerreichbarer Programmcode
- Weiterer Aufwand gerechtfertigt?
- Faktoren zur Entscheidung, ob das Testen beendet wird
- In der Praxis: Zeit und Kosten

- Während des Entwicklungsprozesses werden unterschiedliche Dokumente erstellt
  - Informelle Texte
  - Modelle
  - Formale Texte
  - Programmcode
  - ...
- Programme sind **statische** Beschreibungen dynamischer Prozesse
- Dynamische Tests prüfen Prozesse, die aus der ‚Interpretation‘ einer Beschreibung (Testobjekt) resultieren.
- Statische Tests führen das Testobjekt nicht aus, sondern überprüfen es nur ‚als solches‘.

- **Analyse** des Testobjekts (meist eines Dokuments, aber auch Modelle oder Source-Code) durch intensive **Betrachtung**
- **Ziel:** Ermittlung von Fehlerzuständen (Defekten) im Dokument
- **Grundidee: Prävention**  
Fehlerzustände und Abweichungen sollen so **früh wie möglich** erkannt werden - noch bevor sie im weiteren Verlauf der Softwareentwicklung zum Tragen kommen und **aufwändige Nachbesserungen** verursachen

# Manuelle vs. automatisierte Prüfung

- Analyse des Prüfobjekts durch **Mensch** bzw. **Werkzeug**
- **Review**
  - Manuelle Prüfung durch eine oder mehrere Personen
  - Durchführbar mit allen Dokumenten, die während des Softwareentwicklungsprozesses erstellt oder verwendet werden (z.B. auch Verträge)
- **Statische Analyse**
  - **Automatisierte Prüfung** durch entsprechendes **Werkzeug**
  - Nur bei Dokumenten mit **formaler Struktur** möglich (z.B. Modelle, Quellcode)

- Erläutere den Unterschied zwischen statischem und dynamischem Test.



- Nutzen **menschliche Analyse- und Denkfähigkeit**, um komplexe Sachverhalte zu prüfen und zu bewerten
- Methode: Intensives **Lesen** und **Nachvollziehen** der untersuchten Dokumente
- Es existieren verschiedene Varianten, die sich durch die **Intensität**, die benötigten **Ressourcen** (z.B. Personen und Zeit) und die verfolgten **Ziele** unterscheiden

- Bezeichnung für ein **bestimmtes Vorgehen** bei der Prüfung von Dokumenten
- Allgemein auch als **Oberbegriff** für verschiedene, von Personen durchgeführte statische Prüfverfahren gebräuchlich
- Alternativ häufig als **Inspektion** bezeichnet
- Alle Dokumenttypen können einem Review oder einer Inspektion unterzogen werden, z.B.
  - Anforderungsbeschreibung
  - Entwurfsspezifikationen
  - Programmtexte
  - Testpläne
  - Benutzungshandbuch usw.

- Ein Review umfasst fünf Arbeitsschritte
  1. Planung (welche Dokumente (deren Status?), Aufwand, Reviewteam)
  2. Einführung (Vorstellung Prüfdokument, Checklisten)
  3. Vorbereitung (individuell, Beschränkung auf bestimmte Aspekte)
  4. Reviewsitzung
  5. Nachbereitung (Autor: Mängel beseitigen; Manager: überprüfen, ggf. weiteres Review, Auswertung)

- **Manager:** Wählt die Prüfobjekte, stellt das Reviewteam zusammen, leitet sämtliche Planungsaktivitäten
- **Moderator:** Leitet die Sitzung, vermittelt, ist unvoreingenommen
- **Autor:** Ersteller des Dokuments oder Hauptverantwortlicher
- **Gutachter (Reviewer, Inspektor):** Gruppe von Fachexperten
  - Maximal fünf
  - Kennzeichnen mangelhafte Teile des Prüfobjektes
  - Benennen als gut befundene Dokumententeile auch so
- **Protokollant:** Aus pragmatischen Gründen kann es manchmal sinnvoll sein, den **Autor selbst** das Protokoll führen zu lassen

- Vom Moderator geführt
- Üblicherweise auf festen **Zeitraumen** beschränkt (max. 2 Stunden)
  - Falls nötig wird eine weitere Sitzung abgehalten
  - Frühestens jedoch am nächsten Tag
- **Ziel:** Beurteilung des Prüfobjektes in Bezug auf die Einhaltung und Umsetzung der Vorgaben und Richtlinien
- Die **Bewertung** der einzelnen Befunde wie auch die **Gesamtbeurteilung** soll von allen Gutachtern getragen werden

- Moderator hat das Recht, eine Sitzung **abzusagen** oder **abzubrechen**, wenn
  - einer oder mehrere Gutachter nicht erschienen oder ungenügend vorbereitet sind
  - er aus irgendeinem Grund nicht imstande ist, die Sitzung erfolgreich und effizient zu leiten
- Das **Resultat** (also das zu prüfende Dokument, das Prüfobjekt) und **nicht der Autor** stehen zur Diskussion
  - Gutachter müssen auf ihre Ausdrücke und Ausdrucksweisen achten
  - Autor darf weder sich noch das Resultat verteidigen  
Der Autor sollte deshalb keinen *Angriffen* ausgesetzt werden, die ihn zur *Verteidigung* nötigen
  - Rechtfertigung seiner Entscheidungen wird teilweise als legitim und hilfreich angesehen

- **Moderator** darf **nicht** gleichzeitig als **Gutachter** fungieren
- **Keine allgemeinen Stilfragen** außerhalb der Richtlinien werden diskutiert

- Entwicklung und Diskussion von Lösungen ist nicht Aufgabe des Reviewteams.
- Jeder Gutachter muss Gelegenheit haben, seine Befunde angemessen präsentieren zu können.
- Der Konsens der Gutachter zu einem Befund ist laufend zu protokollieren.
- Befunde nicht in Form von Anweisungen an den Autor protokollieren
  - Anweisungen in Form zusätzlicher konkreter Korrekturvorschläge werden allerdings teilweise als durchaus sinnvoll und hilfreich für die Qualitätsverbesserung angesehen.

- Einzelne Befunde werden gewichtet, z.B. als
  - **Kritischer Fehler**  
Prüfling oder Prüfobjekt ist für den vorgesehenen Zweck unbrauchbar, Fehler muss vor der Freigabe behoben werden.
  - **Hauptfehler**  
Nutzbarkeit des Prüflings beeinträchtigt, Fehler sollte vor der Freigabe behoben werden.
  - **Nebenfehler**  
Geringfügige Abweichung, z.B. Rechtschreibfehler oder Mangel im Ausdruck, beeinträchtigt den Nutzen kaum.
  - **Gut**  
Fehlerfrei, bei Überarbeitung nicht ändern.

- Review-Team gibt **Empfehlung** über die Annahme des Prüflings ab
  - **Akzeptieren** → keine Änderungen nötig
  - **Akzeptieren mit Änderungen** → kein weiteres Review nötig
  - **Nicht akzeptieren** → weiteres Review oder andere Prüfmaßnahme erforderlich
- **Protokoll** enthält Liste aller Befunde, die in der Sitzung diskutiert wurden sowie zusätzlich
  - Informationen zum Prüfobjekt und den verwendeten Dokumenten
  - Beteiligte Personen und ihre Rollen
  - Kurze Zusammenfassung der wichtigen Punkte
  - Ergebnis der Reviewsitzung mit der Empfehlung der Gutachter
- Am Schluss unterschreiben alle Sitzungsteilnehmer das Protokoll

- **Manager** entscheidet, ob er der **Empfehlung** folgt oder eine **andere Entscheidung** trifft, für die er dann aber die alleinige Verantwortung trägt
- War das **Resultat** des ersten Reviews **nicht akzeptabel** muss ein **weiteres**, meist jedoch verkürztes **Review** durchgeführt werden
- Im Regelfall werden die **Mängel** durch den Autor auf Grundlage der Review-Ergebnisse **beseitigt**
- Ordnungsgemäße Durchführung der **Überarbeitung** ist zu **kontrollieren**, meist durch den Manager

- „Manuelle, informale Prüfmethode, um Fehler, Defekte, Unklarheiten und Probleme in schriftlichen Dokumenten zu identifizieren. Der Autor präsentiert das Dokument in einer Sitzung den Gutachtern.“  
[Balzert, Lehrbuch der Softwaretechnik, 1998]
- **Sitzung** bildet den Schwerpunkt
- Vorbereitung hat im Vergleich zu den anderen Review-Arten den geringsten Umfang, teilweise kann sogar darauf verzichtet werden
- In der Sitzung übernimmt der **Autor** die Rolle des **Moderators** und ggf. auch die des **Protokollanten**; er stellt das Prüfobjekt den Gutachtern vor (**problematisch**)

- Meist werden typische **Anwendungsfälle** (Benutzungssituationen, Szenarien) **ablauforientiert** durchgespielt
- Alternativ können auch einzelne **Testfälle** nachgespielt werden
- Gutachter versuchen durch meist spontane Fragen, mögliche Fehler und Probleme aufzudecken
- Geeignet für **kleine Entwicklungsteams** und für Prüfungen unkritischer Dokumente
- **Wenig Aufwand**, da Vor- und Nachbereitungen von geringem Umfang bzw. nicht zwingend erforderlich sind
- Für die Nacharbeit ist der Autor verantwortlich, eine **Kontrolle** findet **nicht** statt

- Fehler werden frühzeitig erkannt und beseitigt
  - **Produktivitätssteigerung** in der Entwicklung, da weniger Ressourcen für die Fehlererkennung und -beseitigung benötigt werden
  - **Verringerung** des Aufwands für den dynamischen **Test**, da im Testobjekt weniger Fehlerzustände vorhanden sind
- Es ergeben sich oft **kürzere Entwicklungszeiten**
- **Reduzierte Fehlerhäufigkeit** im Einsatz des Systems
- Durch die geringere Zahl an Fehlern und Ungenauigkeiten ist eine **Kostenreduzierung** während der Produkt-Lebenszeit zu erwarten
  - **Ungenauigkeiten** der Kundenwünsche in den Anforderungen können durch Reviews aufgedeckt und sofort geklärt werden
  - Die Anzahl der **Änderungswünsche** nach der Inbetriebnahme des Softwaresystems kann so verringert werden

- Überprüfungen im Team führen zu einem **Wissensaustausch** unter den beteiligten Personen, was die **Arbeitsmethoden** der einzelnen Personen **verbessert** und damit auch die Qualität der nachfolgenden Produkte dieser Personen
- Da das Review in einer Gruppe stattfindet, ist eine klare und **verständliche Darstellung** der Sachverhalte notwendig
- Oft bringt bereits dieser Zwang den Autor zu **Einsichten**, die er vorher nicht hatte
- Die **Verantwortung** für die Qualität des untersuchten Prüfobjekts trägt das **Team**



- **Kosten** für Reviews betragen etwa 10%-15% des Entwicklungsbudgets
- **Einsparungen** zwischen 14% und 25% sind möglich, die Mehrkosten für die Reviews schon eingerechnet
- Bei konsequenter Nutzung lassen sich bis zu **70% der Fehler** in einem Prüfling finden



- **Ziel:** Aufdeckung vorhandener **Fehlerzustände** oder fehlerträchtiger Stellen in einem Dokument
  - Bezeichnung statische Analyse weist darauf hin, dass auch diese Form der Prüfung **keine Ausführung** der Prüfobjekte (z.B. eines Programms) beinhaltet
- **Beispiele**
  - **Rechtschreibprüfung**  
Führt statische Analyse eines Textes durch, prüft die Einhaltung korrekter Schreibweise
  - **Compiler**  
Führt statische Analyse eines Programmtextes durch, prüft die Einhaltung der Syntax der Programmiersprache

- Zu analysierendes Dokument muss eine **formale Struktur** haben, um durch ein Werkzeug überprüft werden zu können
- **Informeller Text:**
  - Stand der Praxis: Rechtschreibung und elementare Grammatik (vgl. z.B. Word)
  - Darüberhinaus aktuelle Forschung in der Computerlinguistik z.B. kritische Teile eines Anforderungsdokumentes mithilfe Bayescher Filter identifizieren (ähnlich wie Spam-Klassifizierung)
- Statische Analyse ist **nur** mit Werkzeugunterstützung sinnvoll

- Enger Zusammenhang
  - Durch eine statische Analyse eines formalen Dokumentes können bereits einige Fehler und Unstimmigkeiten nachgewiesen werden und damit die Menge der im Review zu berücksichtigenden Aspekte erheblich verringert werden
  - Da statische Analysen werkzeuggestützt durchgeführt werden, ist der Aufwand wesentlich niedriger als beim Review
- Deshalb empfiehlt sich folgender **Ablauf**
  1. Statische Analyse
  2. Review

- **Compiler** führen eine statische Analyse des Programmtextes durch indem sie die Einhaltung der **Syntax** der jeweiligen Programmiersprache prüfen
- Die meisten Compiler bieten darüber hinaus **zusätzliche Informationen**, die durch statische Analysen ermittelt werden
- Neben Compilern gibt es spezielle Werkzeuge, die zur gezielten Durchführung einzelner oder ganzer Gruppen von Analysen eingesetzt werden.

- **Fehler(zustände) bzw. fehlerträchtige Situationen**, die mit der statischen Analyse von Programm Quellcode nachgewiesen werden können, sind beispielsweise
  - Verletzung der **Syntax**
  - Abweichungen von **Konventionen** und **Standards**
  - **Kontrollflussanomalien**
  - **Datenflussanomalien**

- In welchem Zusammenhang stehen Reviews und die statische Analyse?

- Durch die **Anschaulichkeit** des Kontrollflussgraphen lassen sich die Abläufe eines Programmstücks leicht erfassen
  - Bei **unübersichtlichen Graphen** bzw. Teilgraphen mit kaum nachvollziehbarem Zusammenhang oder Ablauf sollte der **Programmtext überarbeitet** werden, da komplexe Ablaufstrukturen oft **fehlerträchtig** sind
- **Kontrollflussanomalie**: Statisch feststellbare Unstimmigkeit im Ablauf des Testobjekts wie 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 wurde durch einem **Werkzeug** erstellt, das eine eins-zu-eins-Abbildung zwischen Programmtext und Graph gewährleistet

- Verfolgung der **Pfade** von Daten durch den Programmcode

## Beispiel

Datenfluss in einfacher Methode (s. Tafel)

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

- Drei Zustände bzw. Verwendungen von Variablen werden unterschieden
  - **Undeniert (u)**: Die Variable hat keinen denierten Wert
  - **Deniert (d)**: Der Variablen wird ein Wert zugewiesen
  - **Referenziert (r)**: Der Wert der Variablen wird gelesen bzw. verwendet
- Drei Arten von Datenflussanomalien werden unterschieden
  - **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: Datenfluss-Anomalie ist die
  - **referenzierende** Verwendung einer Variablen **ohne vorherige Initialisierung**
  - oder die **Nicht-Verwendung** eines Wertes einer Variablen

- **Zwischen den die zu den Anomalie verursachenden Anweisungen können beliebig viele andere Anweisungen stehen**
  - Anomalien sind dann nicht mehr offensichtlich
  - Können bei einer manuellen Prüfung leicht übersehen werden
  - Werkzeug zur Datenflussanalyse deckt die Anomalien auf (z.B. Live- / Avail-Algorithmus bzw. algebraische Methode nach Forman)
- **Nicht jede Anomalie führt direkt zu fehlerhaftem Verhalten**
  - Eine du-Anomalie hat beispielsweise nicht immer direkte Auswirkungen (das Programm kann korrekt laufen)
  - Dennoch lohnt sich meist eine genauere Untersuchung der anomalen Programmstellen, um weitere, semantisch tiefsinnigere Unstimmigkeiten ausfindig zu machen
- **Modellierungs-/Berechnungs-Problem**
  - dynamische Strukturen (Bäume, Listen)
  - Zugriff auf Arrays mit variablem Index

- **Erweiterung** des Begriffs der **Ausführung** eines Programms mit konkreten Werten
  - Statt konkreter Werte werden **symbolische Werte** für die Eingabevariablen verwendet
  - Mit diesen wird dann entsprechend ‚gerechnet‘

## Beispiel

Symbolische Ausführung der Anweisung:

$$C := A + 2 B.$$

Die symbolischen Werte seien mit  $w()$  bezeichnet. Vor Ausführung der Anweisung gelte  $w(A) = a$  und  $w(B) = b$ . Dann ergibt die symbolische Ausführung der obigen Zuweisung:

$$w(C) = w(A) + 2 w(B) = a + 2 b.$$

# Symbolische Ausführung - Vorteile

- Es ist keine formale Programmspezifikation erforderlich
- Ein symbolischer ‚Test‘ deckt eine Vielzahl normaler Testdaten ab
- Insbesondere werden Fehler entdeckt, bei denen für eine Teilmenge der Eingaben die Ergebnisse falsch berechnet werden

# Symbolische Ausführung - Nachteile (1)

- Ergebnisse sind schwieriger als beim konventionellen Testen zu überprüfen, da die symbolischen Ausdrücke mit der Spezifikation verglichen werden müssen
- Die verwendete Programmiersprache muss formal definiert sein, damit ein symbolischer Interpreter arbeiten kann
- Als alleinige Testmethode ist die symbolische Ausführung nicht ausreichend, da ein funktionsorientierter Test fehlt

# Symbolische Ausführung - Nachteile (2)

- Es muss ein Theorembeweiser zur Verfügung stehen, der alle notwendigen Umformungen erzeugen kann (z.B.  $a + 2 \cdot b - a = 2 \cdot b$ ) und korrekt arbeitet
- Für hinreichend mächtige Programmiersprachen existiert kein vollständiger automatischer Theorembeweiser (sonst wäre die Äquivalenz von Programmen entscheidbar)
- Daher kann es vorkommen, dass der Theorembeweiser weder  $w(B) = \text{true}$  noch  $w(B) = \text{false}$  bei einer Verzweigung mit Prädikat  $B$  beweisen kann und daher beide Verzweigungsausgänge gewählt werden, obwohl tatsächlich einer nicht ausführbar ist



- Lesson Learned
  - Grundlagen Testen
  - Testprozess
  - Testarten
  - Review und Reviewprozess
  - Weitere Testansätze