# Einführung in Software Engineering

**Barbara Paech, Marcus Seiler**

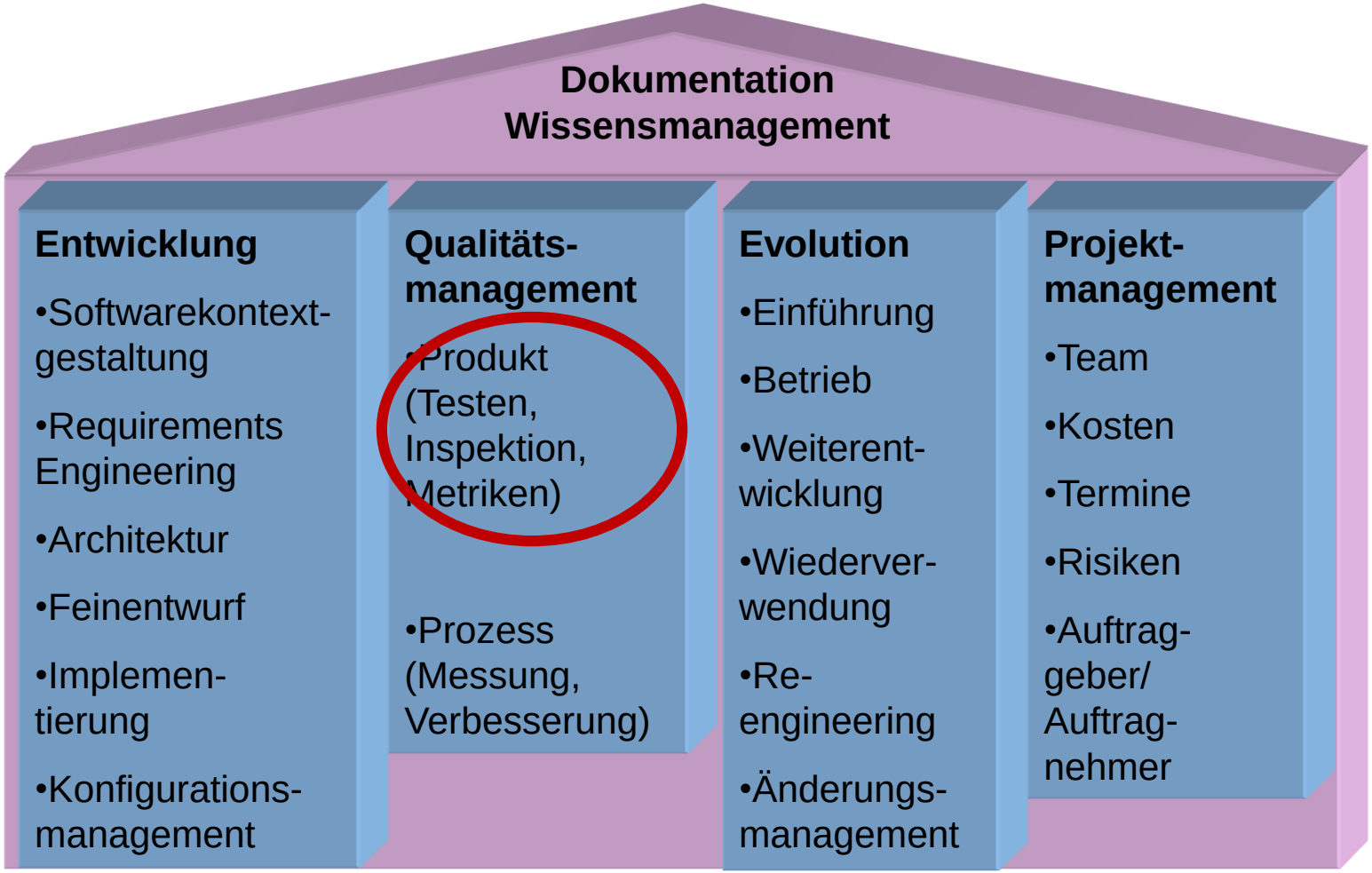
Institute of Computer Science
Im Neuenheimer Feld 326
69120 Heidelberg, Germany
http://se.ifi.uni-heidelberg.de
paech@informatik.uni-heidelberg.de

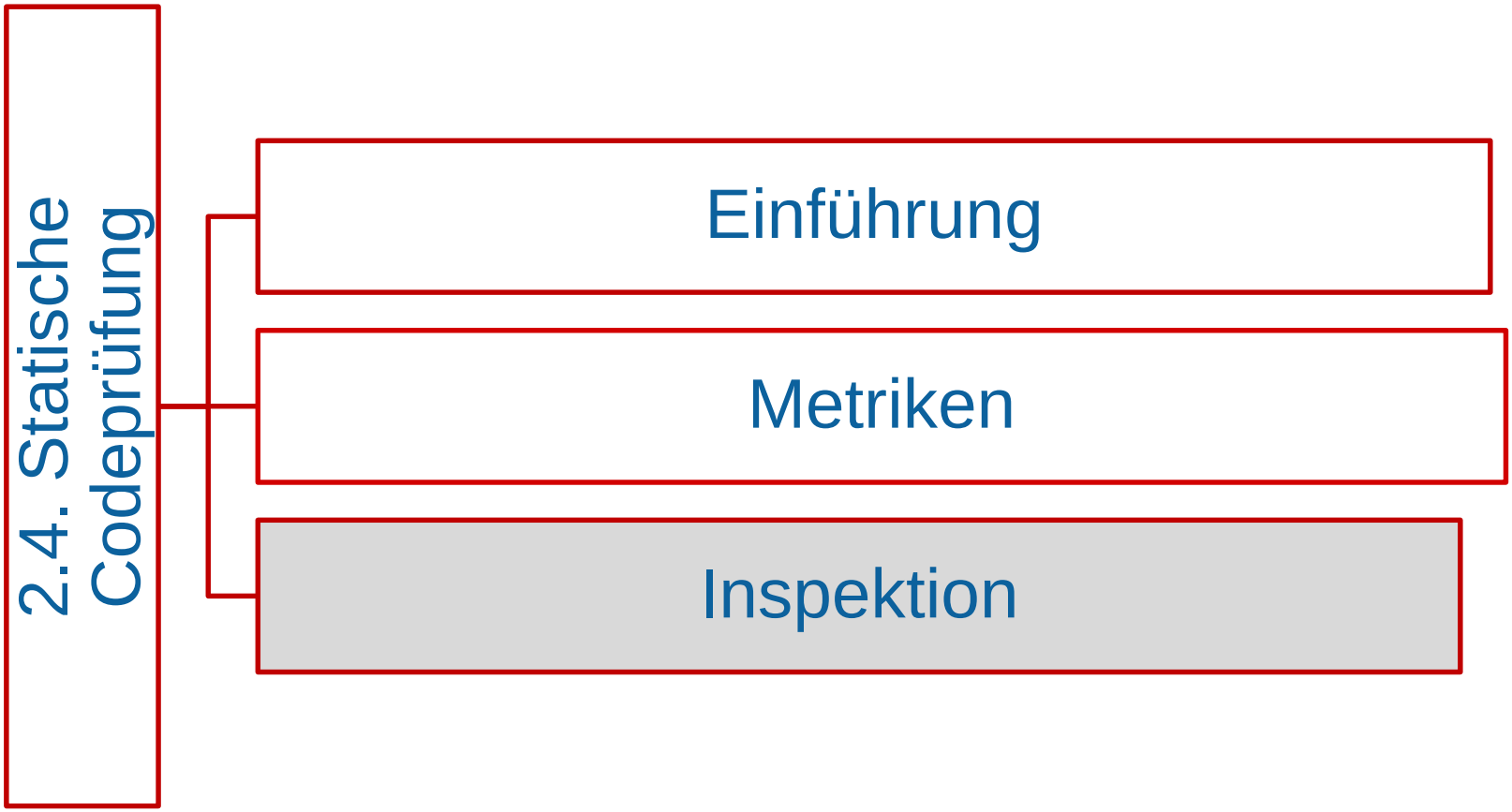

# Kapitel 2:
# SWE im Kleinen

# 2. SWE im Kleinen (2.Teil)

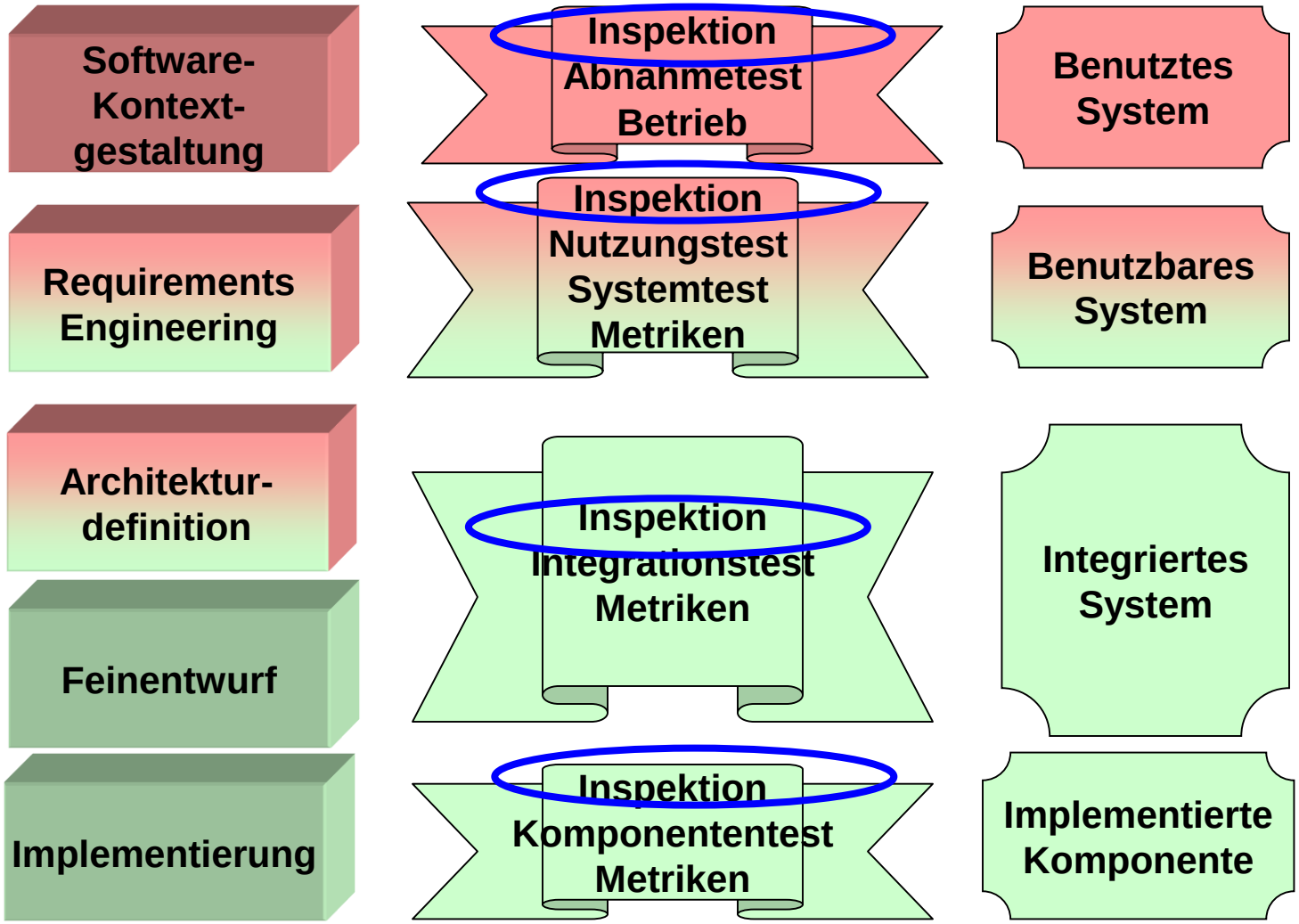
- 2.1. Vorgehen
- 2.2. Einführung Qualitätssicherung
- 2.3. Organisatorische Qualitätssicherung
- 2.4. Statische Codeprüfung
  - Einführung
  - Metriken
  - Inspektion
- 2.5. Umgang mit Fehlern
- 2.6. Testen (Dynamische Codeprüfung)

# Aufgabenbereiche des Software Engineering





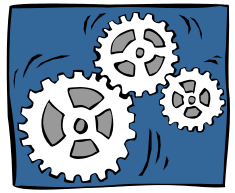
# Aktivitäten und Ergebnisse der Entwicklung und Qualitätssicherung



# Manuelle vs. automatisierte Prüfung

- Betrachtung des Prüfobjekts durch Mensch oder Werkzeug
- Reviews / Inspektionen
  - Manuelle Prüfungen durch eine oder mehrere Personen
  - Menschliche Analyse- und Denkfähigkeit wird genutzt, um komplexe Sachverhalte zu prüfen und zu bewerten
  - Kann bei allen Dokumenten durchgeführt werden, die während des Softwareentwicklungsprozesses erstellt oder verwendet werden (z. B. Vertrag, Anforderungsspezifikation, Entwurfsspezifikation, Quelltext, Testkonzepte, Testspezifikationen, Testfälle, Testskripte oder Anwenderhandbücher)

- Statische Analyse
  - Automatisierte Prüfungen durch entsprechende Werkzeuge
  - Nur bei Dokumenten mit formaler Struktur (z.B. Programmtext, UML-Diagramme)



# Möglichkeiten und Prinzipien der Prüfung

- Eine mechanische Prüfung setzt voraus, dass der Prüfling einer mechanischen Analyse zugänglich sind.
- Darum kommt ganz überwiegend (nur) die Inspektion in Frage.

| Das Dokument … ist prüfbar durch | Inspektion | Metriken | Test |
|---|---|---|---|
| Lastenheft | X | (x für nat. Sprache) | |
| Pflichtenheft | X | (x) | |
| Systementwurf | X | (x) | |
| Benutzerhandbuch | X | (x) | |
| Testdaten | X | (x) | |
| Definition der Daten und Algorithmen | X | | |
| Code | X | x | X |
| Anleitungen etc. | X | (x) | |

# Inspektionsverfahren

- Es gibt viele Verfahren, Software zu inspizieren.

1. die **Durchsicht** (der „Schreibtisch-Test“)
2. die **Stellungnahme**
3. das **Technische Review**
4. der **Walkthrough**
5. die **Design- and Code Inspections**

- **Gesprächsrunden** – vor allem zur Klärung der Anforderungen – werden ebenfalls als Reviews bezeichnet werden.
  - Wir sprechen in diesem Fall von **Workshops**. Sie sind sehr sinnvoll, haben aber eine andere Zielsetzung als die Reviews.

# Technisches Review

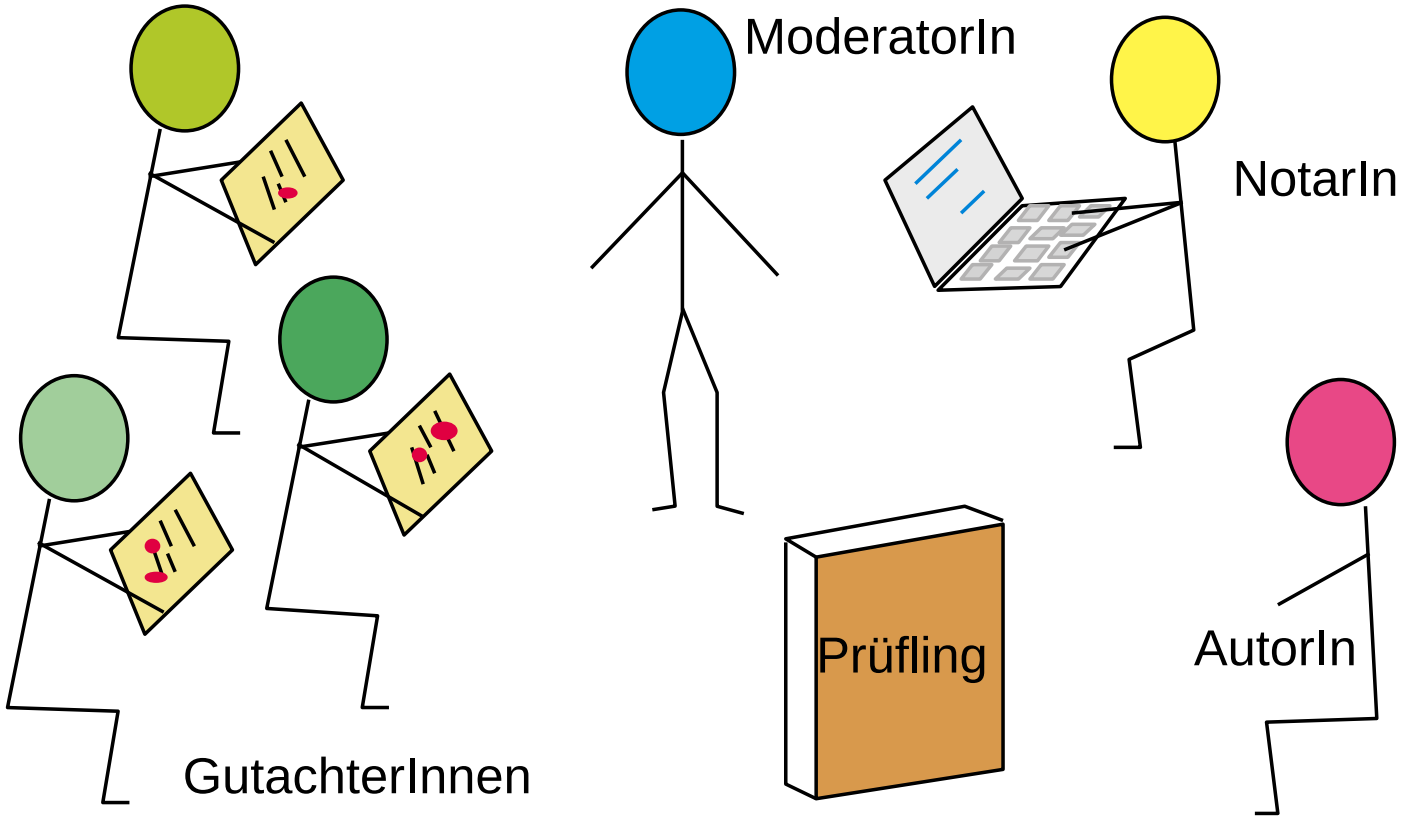
- Nachfolgend wird das Technische Review beschrieben. In der Praxis gibt es zahlreiche Varianten. Zwischen diesen Formen gibt es keine simple Qualitätsordnung!
- Eine Software-Einheit wird (dezentral) von mehreren GutachterInnen inspiziert; in einer gemeinsamen Sitzung werden die Mängel zusammengetragen und dokumentiert.
- Ziel des Reviews ist es, Fehler zu finden, nicht, die Fehler auch zu beheben.
- Prüfling kann jeder in sich abgeschlossene, für Menschen lesbare Teil von Software sein — ein einzelnes Dokument, ein Codemodul, ein Datenmodul.
- Zur Prüfung werden Referenzunterlagen benötigt (Vorgabe oder Spezifikation, Richtlinien, evtl. auch Fragenkataloge).

# Rollen im Technischen Review

- Zuständigkeiten und Verantwortlichkeiten sind im Review klar durch folgende Rollen definiert:
  - ModeratorIn leitet das Review, ist also für den ordnungsgemäßen Ablauf verantwortlich.
  - NotarIn führt das Protokoll.
  - GutachterIn sind KollegInnen, die den Prüfling beurteilen können.
  - AutorIn ist der/die Urheber Indes Prüflings oder einE RepräsentantIn des Teams, das den Prüfling erstellt hat.
  - ManagerIn (Linienvorgesetzte oder ProjektleiterIn) hat den Auftrag zur Erstellung des Prüflings gegeben und somit auch die Verantwortung für die Freigabe des Prüflings. Er/Sie sollte (vor allem in den ersten Versuchen) nicht am Review teilnehmen!

# Das Review-Team

- Das Review-Team bilden alle TeilnehmerInnen des Reviews außer dem/der AutorIn.



# Aufträge an die GutachterInnen - 1

- GutachterInnen erhalten im Technischen Review immer konkrete Aufträge. Diese haben einen doppelten Sinn:
  - Fehler, die immer wieder vorkommen, werden sehr wahrscheinlich entdeckt.
  - EinE GutachterIn, die/der nach bestimmten Mängeln sucht, ist insgesamt aufmerksamer als ohne speziellen Prüfauftrag.

**Fragenkatalog für GutachterInnen - Art des Dokuments: Spezifikation**

Bereiten Sie die Review-Sitzung vor, indem Sie den Prüfling speziell unter den Ihnen in der Einladung zugeordneten Aspekten aus der folgenden Liste prüfen:

**Aspekt "Form":**

*Ist die Darstellung im Dokument sinnvoll?*

1. Sind alle Anforderungen erkennbar, d. h. von Erklärungen unterscheidbar?
2. Sind alle Anforderungen eindeutig referenzierbar?
3. Ist die Spezifikation jeder Anforderung eindeutig?
4. Sind alle Anforderungen überprüfbar formuliert?

# Aufträge an die GutachterInnen - 2

**Aspekt "Schnittstellen":**

*Sind alle Schnittstellen eindeutig spezifiziert?*

5. Sind alle Objekte der Umgebung (NutzerIn, andere Systeme, Basis-Software etc.) sowie alle Informationsflüsse von und nach diesen Objekten spezifiziert?
6. Sind alle Nutzerklassen des Systems (DauernutzerIn, gelegentliche NutzerIn, System-AdministratorIn etc.) identifiziert?
7. Ist die Benutzungsschnittstelle für jede der Nutzerklassen festgelegt?
8. Ist die Bedienphilosophie einheitlich?
9. Ist das beschriebene Bedienkonzept den Vorkenntnissen der NutzerInnen angemessen?

...

# Wie prüft man ein Dokument?

- Begriffserläuterung Lesetechniken
  - Lesetechniken sind Hilfsmittel für GutachterInnen. Sie sollen die Überprüfung der Qualitätseigenschaften von Softwareprodukten erleichtern.

- Welche Lesetechniken gibt es für Inspektionen?
  - Ad-hoc (also gar keine besondere Technik)
  - Checklistenbasiertes Lesen
  - Fehlerklassenbasiertes Lesen (meist besondere Art von Checkliste)
  - Perspektivenbasiertes Lesen (vor allem für Dokumente, siehe später)

# Einordnung der Lesetechniken

- Checklisten-basiertes Lesen
  - schnellere, objektivere und wiederholbare Analyse
  - wichtige Ressource eines Unternehmens, die den gegenwärtigen Stand von Erfahrungen reflektieren
  - Für Entdeckung semantischer Mängel nicht so gut geeignet

- Perspektiven-basiertes Lesen
  - Familie von Techniken für unterschiedliche Ziele, unterschiedliche Artefakte, verschiedene Sichten auf Artefakte
  - systematischer als Checkliste
  - Besonders geeignet für Entdeckung semantischer Mängel
  - *Siehe später*

# Typische Checkliste für JAVA-Code

- Variable, Attribute, and Constant Declaration Defects
  - e.g. Could any non-local variables be made local?
- Method Definition Defects
  - e.g. Is every method parameter value checked before being used?
- Class Definition Defects
  - e.g. Can the class inheritance hierarchy be simplified?
- Data Reference Defects
  - e.g. For every object or array reference: Is the value certain to be non-null?
- Computation/Numeric Defects
  - e.g. Are there any computations with mixed data types?
- Control Flow Defects
  - e.g. Are all exceptions handled appropriately?
- Input-Output Defects
  - e.g. Have all files been closed after use?
- Module Interface Defects
  - e.g. If an object or array is passed, does it get changed, and changed correctly by the called method?
- Comment Defects
  - e.g. Do the comments and code agree?
- Layout and Packaging Defects
  - e.g. For each compile module: Is no more than about 600 lines long?
- Modularity Defects
  - e.g. Are the Java class libraries used where and when appropriate?
- Storage Usage Defects
  - e.g. Are arrays large enough?
- Performance Defects [Optional]
  - e.g. Are there tests within a loop that do not need to be done?

# Anwendung von Checklisten

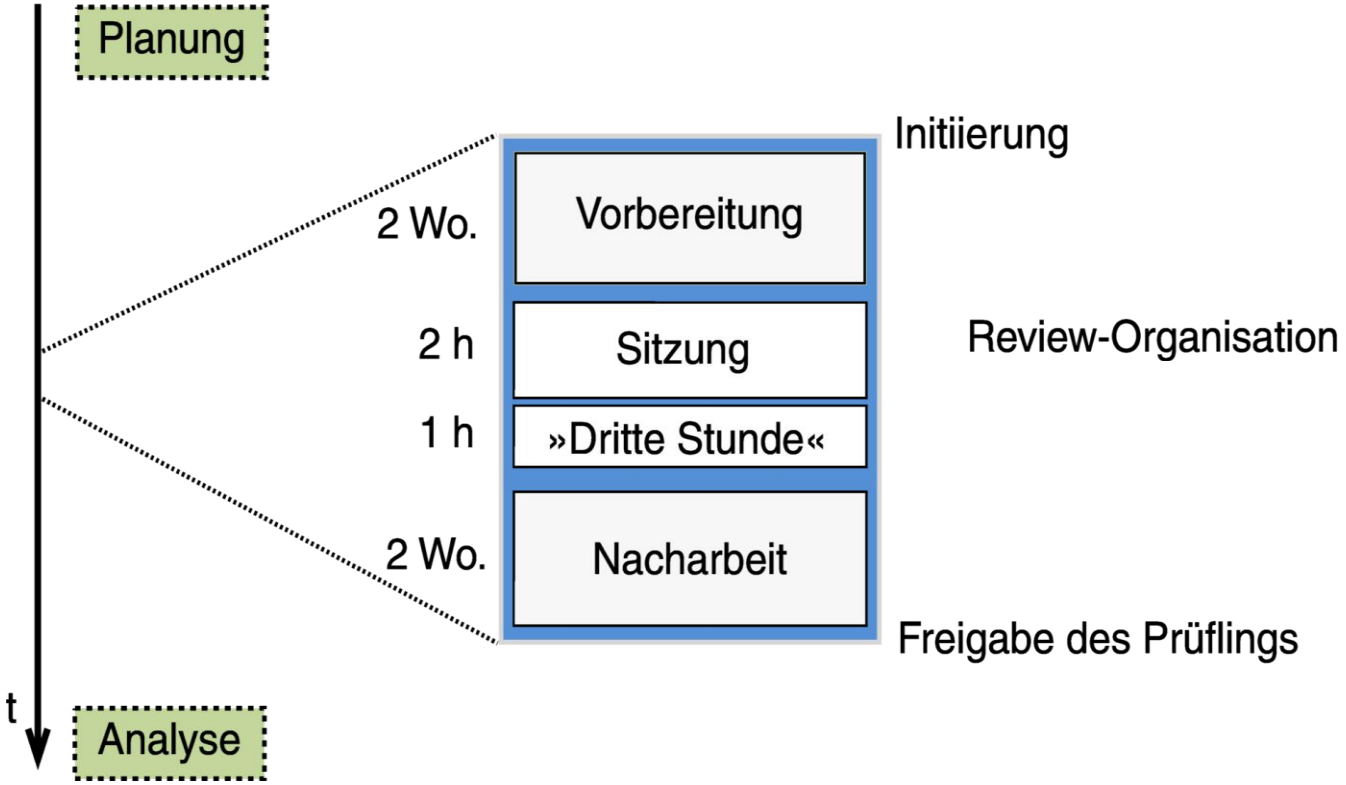
- Allgemeine Checkliste aus der Literatur auswählen
- Checkliste unternehmens- und projektspezifisch erweitern
- Checkliste ist nie vollständig, daher nur Gedankenstütze !

- Jeder Punkt der Checkliste muss für jedes einzelne Element (bei Code, z.B. Klasse, Methode) geprüft werden. Zwei Vorgehensweisen:
  - Einen Punkt der Checkliste für alle Elemente prüfen, dann nächster Punkt
  - Ein Element auf alle Punkte der Checkliste prüfen, dann nächstes Element
- Probleme:
  - Alle GutachterInnen suchen das Gleiche
  - Durchgehen der Checkliste ist „ermüdend“

- Mögliche Abhilfe:
  - GutachterInnen betrachten unterschiedliche Teile des Prüfobjektes
  - GutachterInnen bearbeiten unterschiedliche Teile der Checkliste
  - GutachterInnen sollen kreativ mit dem Produkt umgehen => Perspektiven-basiertes Lesen

# Organisation und Ablauf - 1

- Die eigentliche Review-Sitzung ist eingerahmt von je etwa zwei Wochen Vorbereitung und Nacharbeit;
unmittelbar nach der Sitzung kann die „Dritte Stunde“ folgen.



# Organisation und Ablauf - 2

- Anstoß: von der Konfigurationsverwaltung. ModeratorIn wird informiert, verteilt die Einladungen an die GutachterInnen; Prüfling und Prüfaufträge sind Teile der Einladung.
- Vorbereitung: Die **GutachterInnen lesen** das zu prüfende Dokument und prüfen es nach den ihnen zugeteilten Gesichtspunkten.
- Review-Sitzung: Die GutachterInnen tragen die in der Vorbereitung entdeckten Mängel vor. Gemeinsam erheben, gewichten und protokollieren sie die Befunde.
- Resultat: Liste der Schwächen mit Empfehlung, welche Arbeiten vor der Freigabe des Prüflings durchgeführt werden sollten.
- Dritte Stunde: Die Gutachter Innen und die/der AutorIn reden ohne Regeln und ohne Protokoll.
- Nacharbeit: Sache der AutorInnen; **Umfang** legt ManagerIn fest.

- Planung und Analyse finden lange vor/nach dem Review statt.

# Review-Regeln - 1

Was ist zu beachten, damit der Review ein gutes Ergebnis hat?

1. ModeratorIn organisiert das Review, lädt dazu ein und führt es durch. (Zweckmäßig: fester Review-Termin)
2. Die Review-Sitzung ist auf 2h beschränkt. Falls nötig wird eine weitere Sitzung, frühestens am nächsten Tag, einberufen.
3. ModeratorIn sagt oder bricht die Sitzung ab, wenn die Sitzung nicht erfolgreich durchgeführt werden kann. Der Grund des Abbruchs ist zu protokollieren.
4. Der Prüfling, nicht AutorInnen stehen zur Diskussion. Das heißt:
   - GutachterInnen müssen auf Sprache und Ausdrucksweise achten.
   - AutorIn darf weder sich noch den Prüfling verteidigen.
5. Die Rollen werden nicht vermischt. Insbesondere darf ModeratorIn nicht gleichzeitig als GutachterIn fungieren.
6. Stilfragen (außerhalb der Richtlinien) werden nicht diskutiert.

# Review-Regeln - 2

7. Problemlösen ist keine Aufgabe des Review-Teams. Befunde werden nicht als Anweisungen an AutorIn protokolliert.
8. GutachterInnen bekommen Gelegenheit, ihre Befunde angemessen zu präsentieren.
9. Konsens der GutachterInnen zu jedem Befund wird laufend protokolliert.
10. Die einzelnen Befunde werden gewichtet als:
    - Kritischer Fehler, Hauptfehler, Nebenfehler
    - Gut (kein Fehler festgestellt)
11. Das Review-Team gibt eine Empfehlung über die Annahme des Prüflings ab:
    - Akzeptieren ohne Änderungen
    - Akzeptieren mit Änderungen
    - Nicht akzeptieren
12. Am Schluss unterschreiben alle TeilnehmerInnen das Protokoll.

# Das Review als soziales Experiment

- Wo Reviews eingeführt werden, bedeuten sie für die Beteiligten eine große Veränderung.
- Zuerst werden die negativen Folgen sichtbar:
  - Die Aussicht, als AutorIn im Review zu sitzen, macht Angst.
  - Viele EntwicklerInnen sind Autodidakten!
- Regeln, die die Anfangsprobleme entschärfen sollen:
  - der Ausschluss der Vorgesetzten
  - das Verbot, über Stilfragen zu diskutieren
  - die Leitung des Reviews durch erfahreneN ModeratorIn
  - die Dritte Stunde, die den Emotionen Auslauf gibt
  - der regelmäßige Rollentausch
  - ein objektives, technisches Kriterium, welche Resultate einem Review unterzogen werden.

# Lohnt sich denn die ganze Mühe?

- Klares Ja!
  - Viele Fehler werden zu insgesamt akzeptablen Kosten entdeckt. Es wäre weit teurer, die Fehler nicht (oder nicht so) zu suchen.
  - Die Zusammenarbeit der EntwicklerInnen wird besser, das Vertrauen zwischen den Software-Leuten deutlich höher; niemand fürchtet sich davor, die anderen um Rat zu fragen. Richtlinien werden nicht belächelt, sondern gelebt.
  - Die Beteiligten entwickeln ein sachlich fundiertes Selbstbewusstsein und einen begründeten Stolz auf ihre Arbeit.

- Wer einmal an Reviews gewöhnt ist, möchte auf keinen Fall mehr darauf verzichten.

# Ein Erste-Hilfe-Kasten - 1

- Reviews sind weltweit seit langem als wichtigste Technik der Software-Prüfung anerkannt
- Ihr Nutzen ist klar nachgewiesen und von allen, die regelmäßig und systematisch Reviews durchführen, bestätigt.

- Trotzdem gibt es viele Fälle, in denen die Einführung von Reviews gescheitert oder zu einer ohne Überzeugung betriebenen Formalübung degeneriert ist.

1. Für die Vorbereitung oder sogar für die Review-Sitzung selbst fehlt die Zeit → Reviews in der Planung berücksichtigen
2. Die Reviews finden in einer bis zur Unkenntlichkeit reduzierten Form statt → Standard (Minimalreview) vorgeben
3. Gute ModeratorInnen fehlen → Leute aussuchen und ausbilden

# Ein Erste-Hilfe-Kasten - 2

4. Bezugsdokumente fehlen → suchen, anpassen, bereitstellen
5. EntwicklerInnen haben Angst → Erste Reviews gründlich vorbereiten (und auf die Ängste eingehen, selbst wenn sie geleugnet werden)
6. Reviews beißen sich an Äußerlichkeiten fest → ihre Bedeutung klarstellen, dann weiter. Beim zweiten Review aber diesen Punkt erneut betonen, nicht schleifen lassen.
7. Bezugsdokumente sind ungeeignet → diskutieren u. verbessern
8. Zeitdruck sabotiert Prüfungen → Klare Entscheidung der Prioritäten
9. Geprüfte Dokumente werden verändert ⇒→ ChangeManagement verbessern
10. Interesse an Reviews sinkt ⇒→ mit Statistiken Erfolg nachweisen

# Wann sind Reviews (noch) nicht sinnvoll?

- Nur in wenigen Situationen ist von der Review-Einführung abzuraten:

a) wenn ein Projekt praktisch bereits gescheitert ist und die Verantwortlichen nach einem Wunder Ausschau halten,

b) wenn keinerlei Bezugsdokumente vorliegen,

c) wenn die EntwicklerInnen in feindliche Lager gespalten sind.

- Was tun?

a) Retten, was zu retten ist, z.B. zentrale Dokumente (Spezifikation, Planung für das Rest-Projekt) in akzeptablen Zustand bringen.

b) Versuchsreview durchführen, Defizite erkennen, Arbeit an Bezugsdokumenten anstoßen.

c) Reviews zunächst separat weiterlaufen lassen, GutachterInnen zwischen den Lagern austauschen

# Weitere Inspektionstechnik: Durchsicht

- Eine Durchsicht führt einE EntwicklerIn allein durch.
- Es ist zweckmäßig, dazu den Bildschirm zu verlassen und das (Teil-) Resultat in Ruhe, aus einer gewissen Distanz, zu überprüfen.

- Die Durchsicht sollte selbstverständlich und obligatorisch sein!
- Sie wird durch Reviews o.ä. nicht überflüssig.

- In Zeiten der (reinen) Stapelverarbeitung war die Durchsicht einfach notwendig.
- Heute wissen wir (u.a. durch Humphreys PSP), dass die Durchsicht effizienter ist als jeder (spontane) Test.

# Weitere Inspektionstechnik: Stellungnahme

- Eine Stellungnahme ist ein „off-line“-Review unter der Regie eineR AutorIn.
- Vorteile
  - geringer Organisationsaufwand
- Erhebliche Nachteile
  - ManagerIn ist nicht beteiligt. Der Aufwand für die Stellungnahmen ist nicht geplant.
  - AutorIn ist gleichzeitig ModeratorIn und wählt GutachterInnen aus, deren Engagement ungewiss bleibt.
  - Das Dokument ändert sich während der Ausarbeitung der Stellungnahmen, ein Teil der Kommentare geht ins Leere.
  - Ein Protokoll über die Befunde gibt es nicht. Die Nachbearbeitung geschieht nach dem Gutdünken des/der AutorIn und wird nicht kontrolliert.

# Weitere Inspektionstechnik: Structured Walkthrough

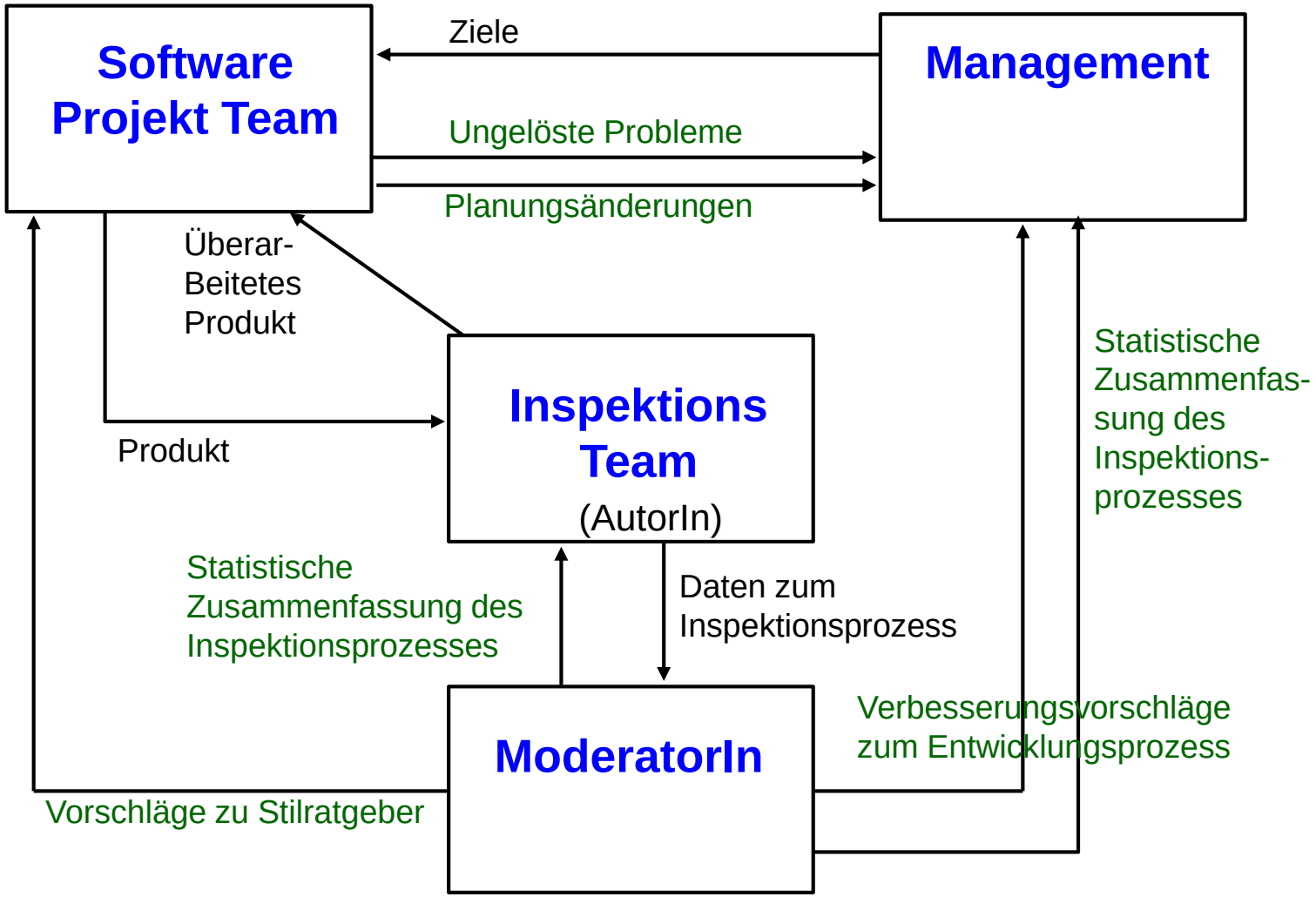
- Der Structured Walkthrough ist die Billig-Variante des Reviews
  - AutorIn ist ModeratorIn
  - AutorIn stellt ihr/sein Arbeitsergebnis Schritt für Schritt vor, GutachterInnen werfen (vorbereitete oder spontane) Fragen auf und versuchen so, mögliche Probleme zu identifizieren. Die Probleme werden protokolliert.
  - AutorIn kompensiert durch Präsentation die Einsparung (oder Reduktion) der Vorbereitung.
  - Während Vorbereitung entdeckt AutorIn selbst viele Fehler.
- Varianten mit oder ohne Vorbereitung der GutachterInnen
- Typische Anwendung: Programmcode.
- Die Effizienz ist niedriger als beim Review!

# Weitere Inspektionstechnik: Design and Code Inspection

- Design and Code Inspection wurde eingeführt von Michael Fagan, IBM

- Die Edel-Variante der Reviews
  - Mit Einführungssitzung. Dort werden der Prüfling und sein Umfeld vorgestellt, die Ziele des Reviews werden allen Beteiligten nochmals deutlich gemacht.
  - GutachterInnen-Notizen, die abgegeben werden.
  - VorleserIn liest den Prüfling Seite für Seite vor, bevor dann zu dem vorgelesenen Teil die Befunde erfasst werden.
  - Entscheidungskompetenz für die Freigabe des geprüften Arbeitsergebnisses
  - Erhebung von Metriken

# Verwendung der Reviewergebnisse (Follow–Up)



# Zusammenfassung Inspektionen

- Inspektionen sind eine gut verstandene und leicht einsetzbare Qualitätssicherungs-Maßnahme, die schnell konkrete Verbesserungen bringt

- Zusätzliche Vorteile: Wissens- und Kenntnistransfer in den Treffen durch die Checklisten / Perspektiven und durch die Ergebnisse

# 2.5. Umgang mit Fehlern

# Schema von Prüfungen

- Alle Prüfungen folgen grundsätzlich dem gleichen Schema.

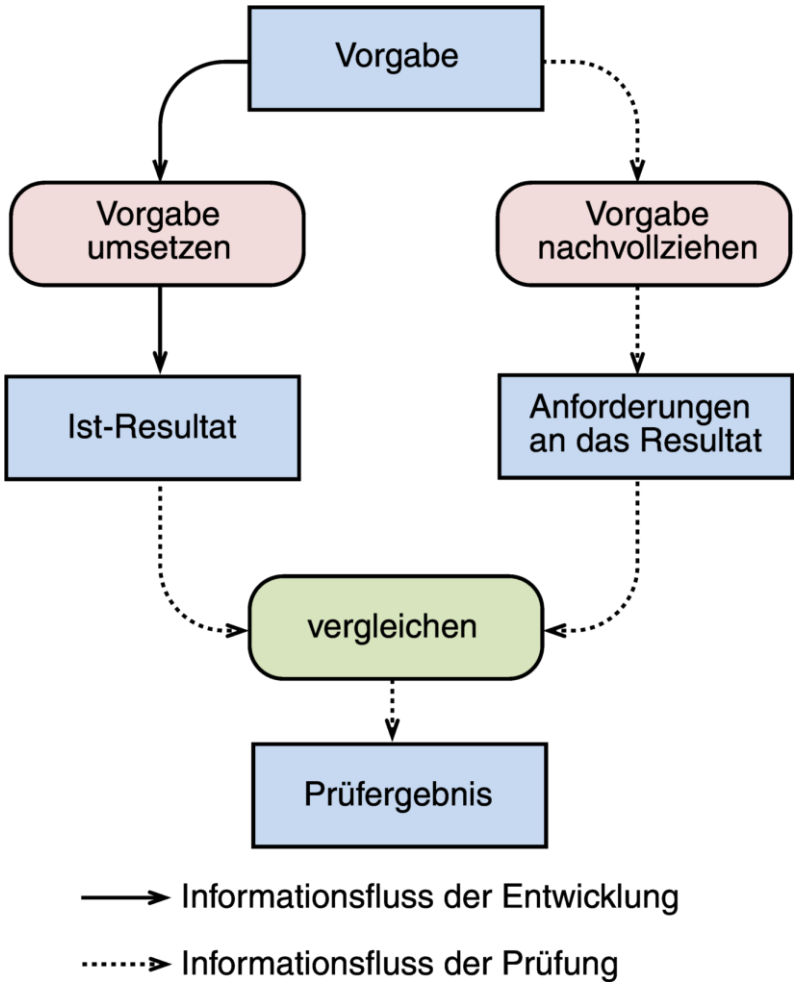
- In jeder Prüfung wird das
  - Resultat (der Ausbildung, der Entwicklung, ...) mit den
  - Erwartungen an das Resultat

verglichen.

- Eine dabei festgestellte **Diskrepanz** weist auf einen Fehler hin

- Sie zeigt aber nicht, wo der Fehler liegt, im Prüfling oder sonstwo.

# Das Prinzip der Prüfung




- Jede Prüfung beruht auf dem Prinzip der Zweigleisigkeit (oder Mehrgleisigkeit).
- Um die (angebliche) Lösung zu prüfen, wird sie
  - mit einer vermutlich richtigen verglichen
  - oder an Kriterien gemessen, die aus den Anforderungen abgeleitet sind.

# Feststellungen für alle Prüfungen

- Alles, was gefordert wurde, muss auch geprüft werden.
- In jeder Prüfung gibt es zwei Pfade von der Vorgabe zum Resultat, die eigentliche Produktion und den Prüfpfad.
- Die Prüfung kann nur Fehler aufdecken, die auf einem der beiden getrennten Pfade liegen, aber keine Fehler, die in den gemeinsamen Teilen liegen. Die Vorgabe muss also anders geprüft werden.
- Werden Fehler entdeckt, so ist das Prüfresultat positiv.
  Liegt der Fehler (nur) im Prüfzweig, so ist es falsch positiv.
- Werden Fehler nicht entdeckt, so wurde entweder nicht nach ihnen gesucht, oder das Prüfresultat war falsch negativ. Solche Resultate entstehen durch zufällige Fehlerkompensation oder (sehr viel öfter) durch unzulässige Verallgemeinerung negativer Resultate oder durch Fehler in den gemeinsamen Teilen.

# Positive und negative Prüfresultate

- Prüfung führt zu einem Befund: Prüfergebnis ist positiv (!)
  - Vgl. die ähnliche Terminologie in der Medizin, wo man von einem positiven oder negativen Befund spricht.
- Wenn der Grund für den Befund nicht in einem Fehler des Prüflings liegt, ist das Prüfergebnis falsch positiv.
  - Mögliche Ursachen: Falsch bestimmte Sollresultate, falscher Prüfling, Fehler beim Vergleich
- Falsch positive Resultate machen unnötig Arbeit, sind aber sonst harmlos.
- Viel schlimmer: falsch negative Resultate, bei denen ein tatsächlich vorhandener Fehler nicht entdeckt wird. Sie suggerieren falsche Sicherheit!
- Daher ist es bei jeder Prüfung weit wichtiger, falsch negative als falsch positive Resultate zu vermeiden.

# Umgang mit Fehlern

- Fehlermanagement (oft auch Bug-Tracking, Issue-Tracking genannt) soll 2 Ziele sicherstellen:
  - Nachvollziehbarkeit / Finden: Wie beschreibe ich Fehlerwirkungen, so dass jemand anderes den Fehlerzustand finden kann?
  - Beseitigung: Wie stelle ich sicher, dass jede gefundene Fehlerwirkung beseitigt wird?

- Debugging
  - Wie komme ich systematisch von der Fehlerwirkung zum Fehlerzustand?

➢ Mehr zur Werkzeugunterstützung dafür in der Zentralübung

# Fehlermanagement

- Fehlermanagement umfasst Übermittlung und Verwaltung von Fehlermeldungen

- Üblicherweise ist eine zentrale Fehlerdatenbank vorhanden, in der alle Probleme, Mängel oder Fehlerwirkungen, die während der  Prüfung (oder auch später im Betrieb) entdeckt werden, erfasst und verwaltet werden

- Die Fehlerdatenbank gibt ein einheitliches Schema zur Meldung von Fehlern vor.

# Schema zur Fehlermeldung

- Nachverfolgbarkeit / Finden
  - EntdeckerIn, Datum, Betroffenes Produkt, ausgeführte QS-Aktivität
  - Beschreibung: Defekt und ggf. Verbesserungsvorschlag

- Einordnung zur weiteren Behandlung/ Beseitigung:
  - Schwere der Auswirkung (Fehlerklasse)
  - Dringlichkeit/Priorität der Behebung (Fehlerpriorität)
  - Status (z.B. offen, zurückgestellt, abgewiesen, Duplikat, Korrektur, erledigt)
  - Änderungshistorie, welche Tätigkeiten von welchen Projektteammitgliedern zur Korrektur der Abweichung durchgeführt wurde (Isolation der Fehlerursache, Änderung der betreffenden Code-Stellen, Fehlernachtest)

- Verbesserung des Entwicklungsprozesses:
  - Schlussfolgerungen und Empfehlungen (aus dem gefundenen Fehler, z.B. Vermeidung von Mehrfachvererbung)
  - Globale Probleme, z.B. wenn andere Bereiche durch die Korrektur der Abweichung beeinflusst werden könnten

# Fehlerklasse – Auswirkung des Fehlers im Betrieb

| Klasse | Bedeutung |
|---|---|
| 1 | Systemabsturz mit ggf. Datenverlust; der Prüfling ist in dieser Form nicht einsetzbar |
| 2 | Wesentliche Funktion ist fehlerhaft; Anforderung nicht beachtet oder falsch umgesetzt; der Prüfling ist nur mit großen Einschränkungen einsetzbar |
| 3 | Funktionale Abweichung bzw. Einschränkung (»normale« Fehler); Anforderung fehlerhaft oder nur teilweise umgesetzt; Prüfling kann mit Einschränkungen genutzt werden |
| 4 | Geringfügige Abweichung; Prüfling kann ohne Einschränkung genutzt werden |
| 5 | Schönheitsfehler (z.B. Mangel im Maskenlayout); Prüfling kann ohne Einschränkung genutzt |

# Fehlerpriorität – Dringlichkeit für die Behebung

Auswirkung im Betrieb nicht zwangsläufig gleich der Dringlichkeit.
Für Planung der Behebung auch Berücksichtigung des Korrekturaufwands oder der QS-Planung insgesamt.

| Priorität | Bedeutung |
|---|---|
| 1 PATCH | Der Arbeitsablauf bei den KundInnen ist blockiert oder die laufenden Prüfungen/Tests können nicht fortgesetzt werden. Das Problem muss unmittelbar, ggf. provisorisch, behoben werden; ein Patch ist zu erstellen |
| 2 NÄCHSTE VERSION | Die Fehlerkorrektur erfolgt mit der nächsten regulären Produktversion oder der nächsten Prüfobjektlieferung |
| 3 GELEGENTLICH | Die Fehlerkorrektur erfolgt, sobald die betroffenen Systemteile ohnehin überarbeitet werden |
| 4 OFFEN | Korrekturplanung ist noch zu erstellen |

# Issue Tracking

- Heute oft Verwaltung von Fehlermeldungen und neuen Feature in einem Werkzeug (Issue Tracker)
- Erlaubt verteilte Erfassung von Issue (Fehlermeldung oder Feature)
- Erlaubt Verteilung der Arbeit und Verfolgung, wer was gemacht hat, um Fehler zu beheben oder Feature zu implementieren
- In Open Source Entwicklung immer eingesetzt
- Beispiele siehe Übungsblatt 4

➢ **Mehr dazu in der Zentralübung**

# Debugging

- Debugging dient der Suche nach dem Fehlerzustand (Fehlerursache)


- Oft sehr ad-hoc

- **Systematisches Vorgehen**
  - Reproduziere die Fehlerwirkung und beschreibe sie (Fehlermanagement)
  - Suche Fehlerzustand durch schrittweise Vereinfachung
    - Entwicklung einer Theorie ala Mastermind
  - Verbessere Fehlerzustand und lerne daraus für die Zukunft

# 2.6. Testen

- Einführung
- Testdurchführung
- Testfallspezifikation
- Black-Box-Komponententest

# 2.6. Testen

- Einführung
- Testdurchführung
- Testfallspezifikation
- Black-Box-Komponententest

- Siehe auch Kapitel 19.1-19.4. bei Ludewig/Lichter (Hausaufgabe 2.5.)
- Nachfolgend ein paar Folien dazu
  - Hier teilweise auch Synonyme zu Begriffen genannt, z.B. Testrahmen statt Testgeschirr, White-Box statt Glass-Box
  - Zusätzlich: wichtige Unterscheidung logische und konkrete Testfälle

# Begriffe (Hausaufgabe)

- Grundlegende Begriffe
  - Test, systematischer Test, Regressionstest, Testfall, Testeingabe, Soll-Resultat, Ausführung, IST-Resultate, Vergleich
- Testarten
  - Vorbereitungsaufwand: Laufversuch, Wegwerftest, Systemat. Test
  - Komplexität (Teststufen): Einzeltest, Modultest (oft synonym Komponententest), Integrationstest, Systemtest
  - Getestet Eigenschaften: Funktionstest, Installationstest, Wiederinbetriebnahmetest, Verfügbarkeitstest, Last-und Stresstest. Regressionstest
  - Beteiligte Rollen: Alpha- und Betatest, Abnahmetest
- Testauswahl
  - Black-Box-Test
  - White-Box-Test (bei LL wird das Glass-Box-Test genannt)

# Begriffe (Hausaufgabe)

- Testdurchführung
  - Testvorbereitung, Testausführung, Testauswertung, Testanalyse, Testgeschirr, Platzhalter, Testtreiber, Testprotokoll, Testbericht, Testendekriterium,
- Testwerkzeuge
  - Instrumentierung, Überdeckung, Metriken, automatische Testdurchführung, Capture/Replay, Testverwaltung, Spezialtests
  - Siehe auch Test-Framework JUnit (in der Zentralübung)

# Vor- und Nachteile des Testens (Hausaufgabe)

- Vorteile des Testens
  - Natürlich, sichtbares Systemverhalten
- Vorteile des systematischen Testens
  - reproduzierbar, wiederverwendbar, Prüfung auch der Testumgebung
- Nachteile des Testens
  - Nicht vollständig möglich
  - nicht alle Anwendungssituationen testbar,
  - Code-Eigenschaften wie Wartbarkeit nicht zu testen
  - liefert nur Fehlerwirkung, nicht Fehlerzustand/ursache
  - Testen prüft nur Code

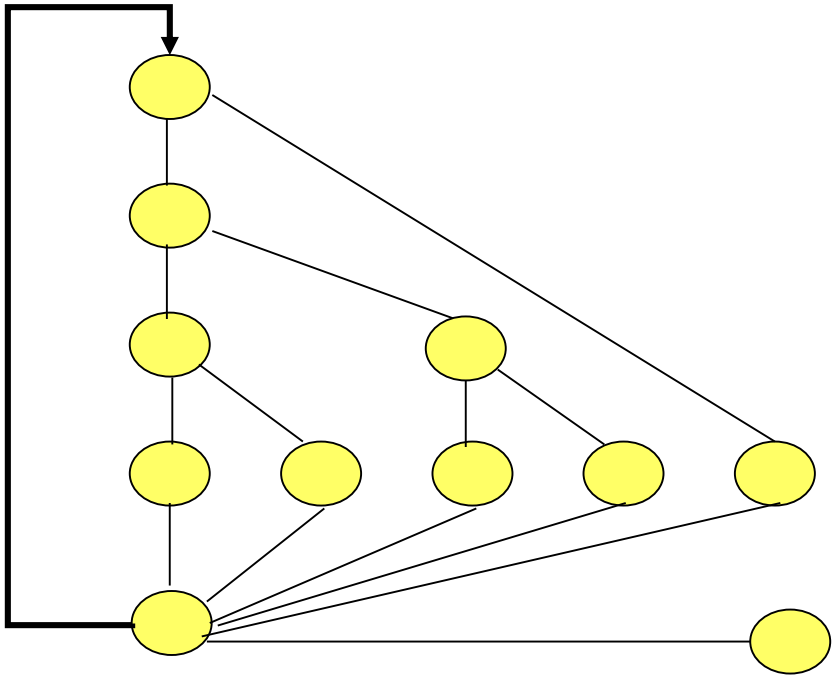
# Einführung Testen

- **Testen** ist die - auch mehrfache – Ausführung eines Programmes auf einem Rechner mit dem Ziel, Fehlerwirkungen zu finden
- Systematischer Test
  - Randbedingungen definiert oder präzise erfasst
  - Eingaben systematisch ausgewählt
  - Ergebnisse dokumentiert und nach Kriterien beurteilt, die vor dem Test festgelegt wurden

> "All code is guilty, until proven innocent."
>
> Anonymous

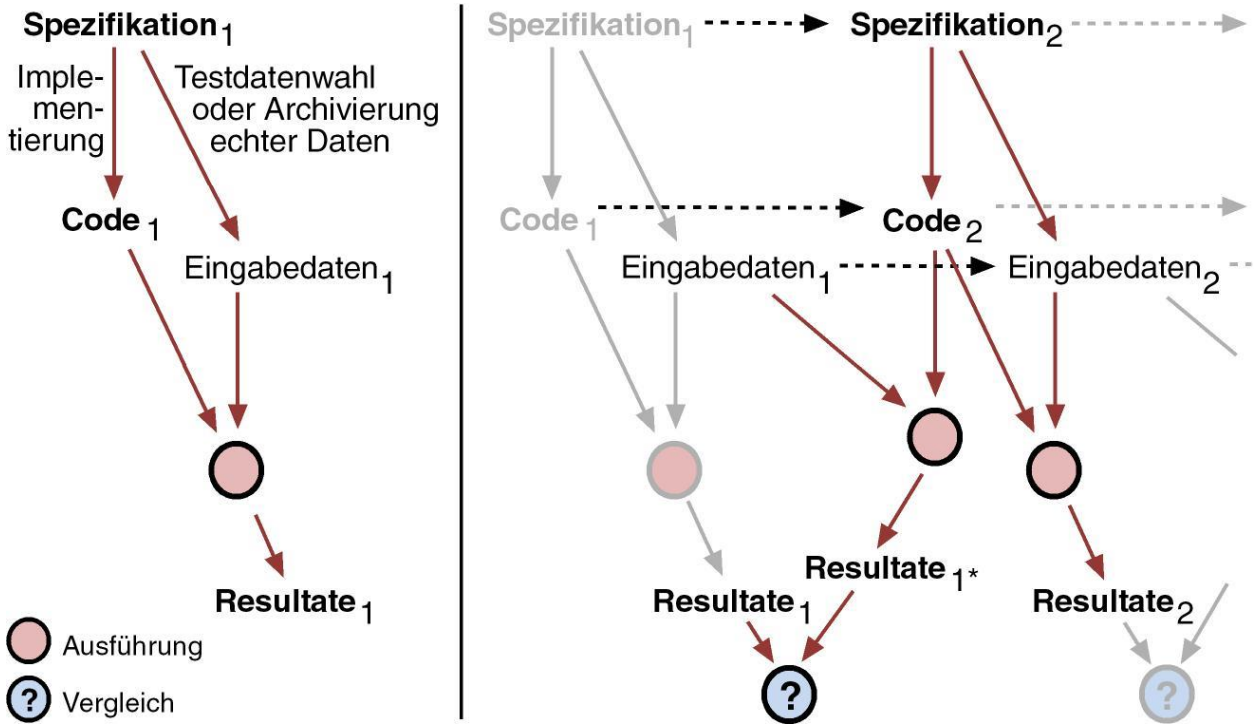
# Testaufwand

- Vollständiges Testen fast nie möglich
  - Wieviele Abläufe sind zu testen für einen vollständigen Test bei maximal x Schleifendurchläufen?
  - Wieviele sind es für x = 20?

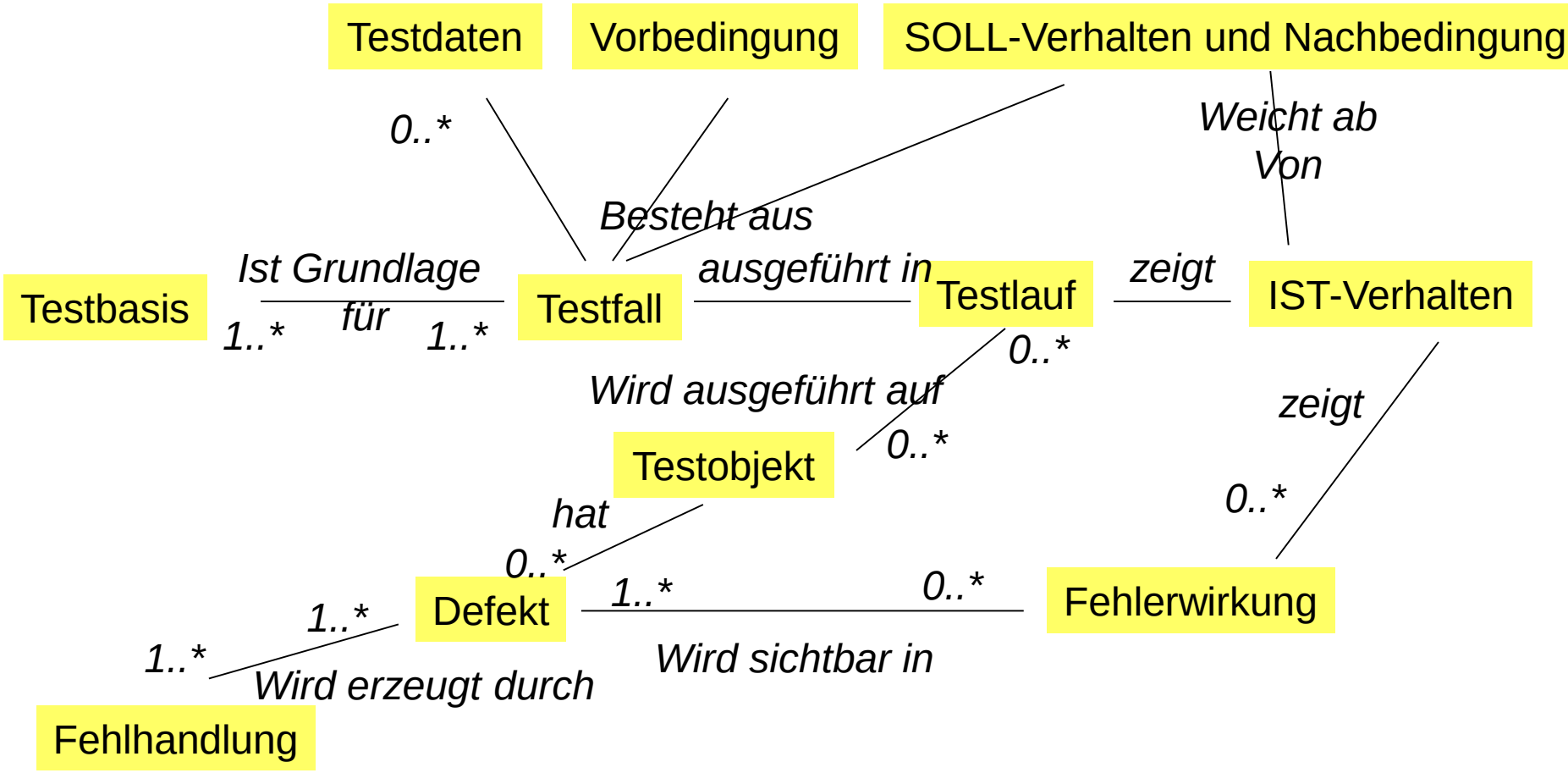


# Regressiontest

- Software muss nach Änderungen wieder getestet werden => Regressiontest prüft, ob Änderungen nicht unbeabsichtigte Effekte haben, und damit ob die Spezifikation nach der Änderung noch erfüllt wird.



# Begriffe im Umfeld Testen



# Testbegriffe (1)

- Testbasis
  - Alle Dokumente, aus denen die Anforderungen ersichtlich werden, die an eine Komponente oder ein System gestellt werden, bzw. die Dokumentation, auf der die Herleitung oder Auswahl der Testfälle beruht
- Testfall
  - Menge von Eingabewerten, den für die Ausführung notwendigen Vorbedingungen und Randbedingungen, der Menge der erwarteten Ergebnisse (Sollwerte) und den erwarteten Nachbedingungen
- Vorbedingung
  - Zustand des Testobjekts (und/oder der Umgebung), der gegeben sein muss, um einen Testfall oder eine Testsuite ablaufen zu lassen
- Nachbedingung
  - Zustand des Testobjekts (und/oder der Umgebung), in dem sich das Testobjekt (oder die Umgebung) nach Ausführung eines Testfalls oder einer Testsuite befindet

# Testbegriffe (2)

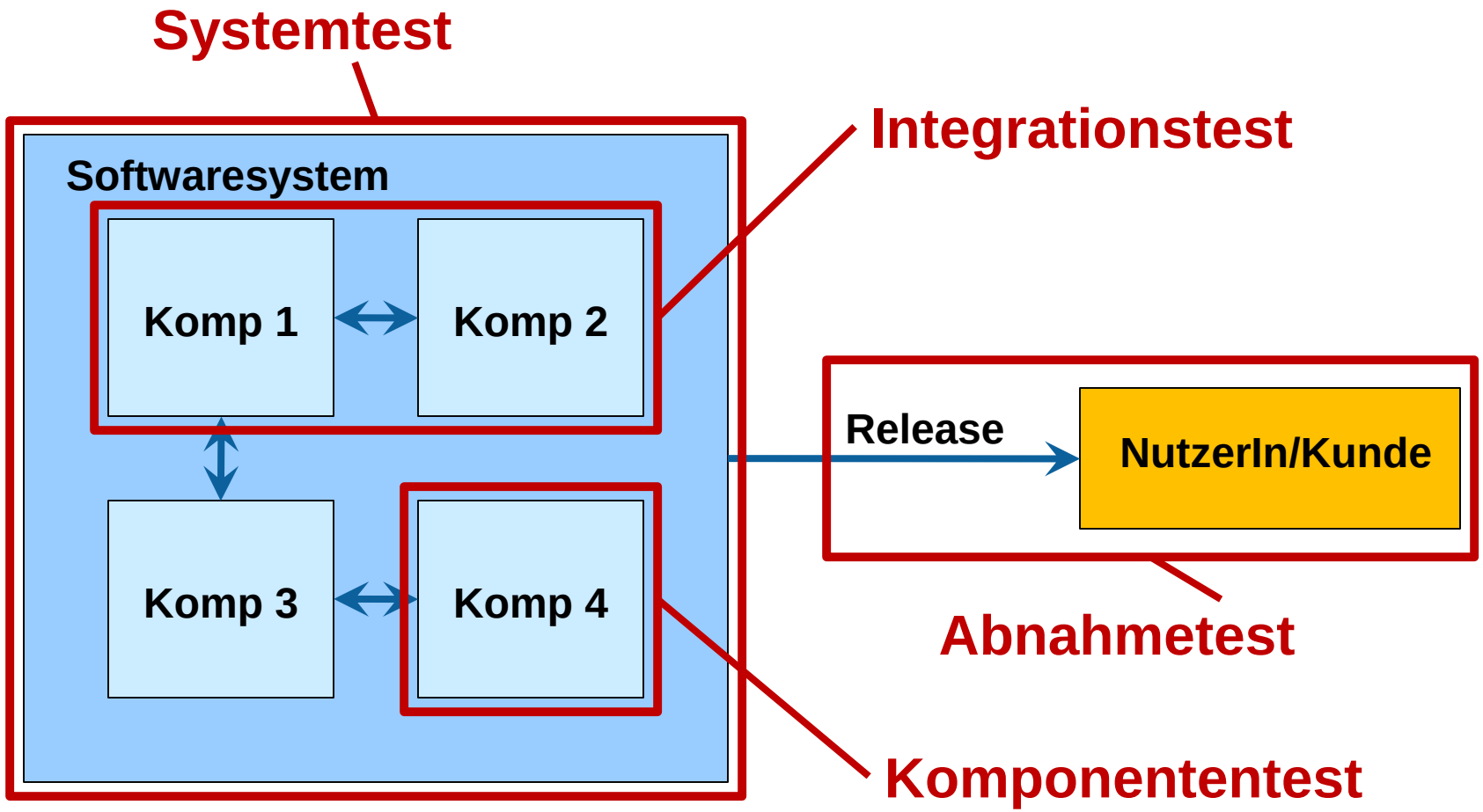
- Testlauf
  - Ausführung eines oder mehrerer Testfälle oder Testsuiten mit einer bestimmten Version des Testobjekts
- IST-Verhalten (Testobjektreaktion)
  - Die Folge der im Testlauf erzeugten Ist-Ausgaben (interne Zustände, Reaktionen, Ausgaben) des Testobjekts; die Testobjektreaktion sollte mit der spezifizierten Soll-Reaktion leicht abgeglichen werden können (im Idealfall automatisch)

[Ludewig/Lichter 2010 Kapitel 19]

# Testarten

- Tests werden nach mehreren Kriterien unterschieden
  - Wie systematisch ist der Test (Vorbereitungsaufwand)?
    - Laufversuch, Wegwerftest, Systemat. Test
  - Was ist das Testobjekt (Teststufen)?
    - Einzeltest, Modultest (oft synonym Komponententest), Integrationstest, Systemtest
  - Welche  Eigenschaft wird getestet?
    - Test von Funktion oder Qualität: Funktionstest, Installationstest, Wiederinbetriebnahmetest, Verfügbarkeitstest, Last-und Stresstest. Regressionstest
  - Wer testet wann (Beteiligte Rollen)?
    - Beim Hersteller vor Auslieferung: Alphatest
    - Bei ausgewählten KundInnen vor Auslieferung:  Betatest
    - Bei KundInnen als Teil der Auslieferung: Abnahmetest (Hersteller demonstriert, dass Anforderungen erfüllt; KundInnen haben ggf. eigene Testfälle)

# Teststufen



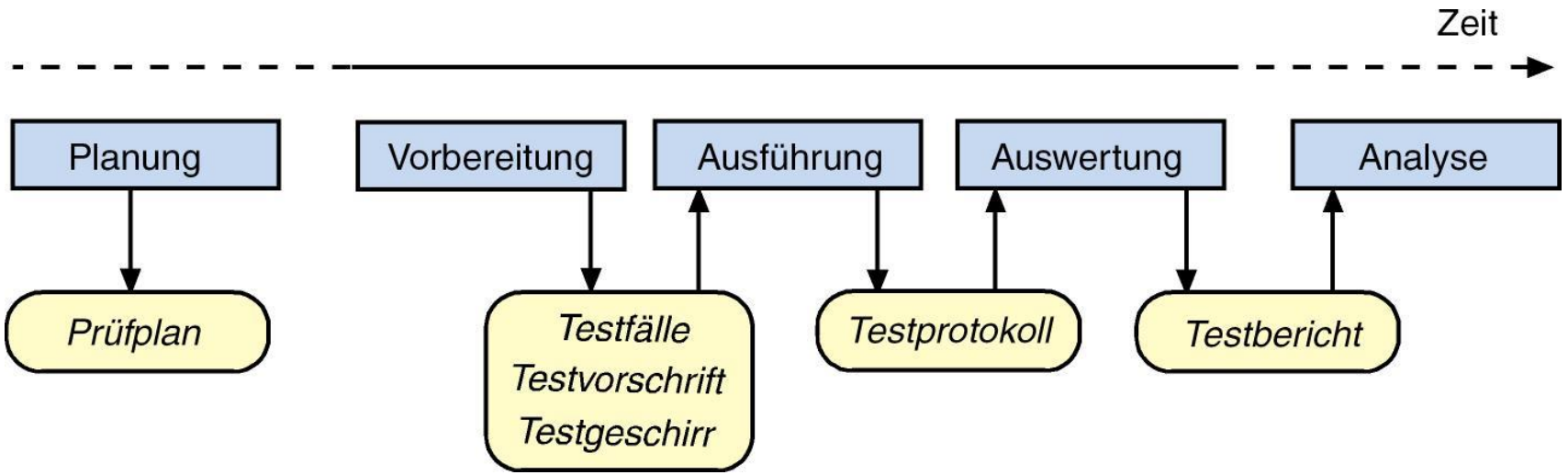

Alle Stufen sind wichtig!

# 2.6. Testen

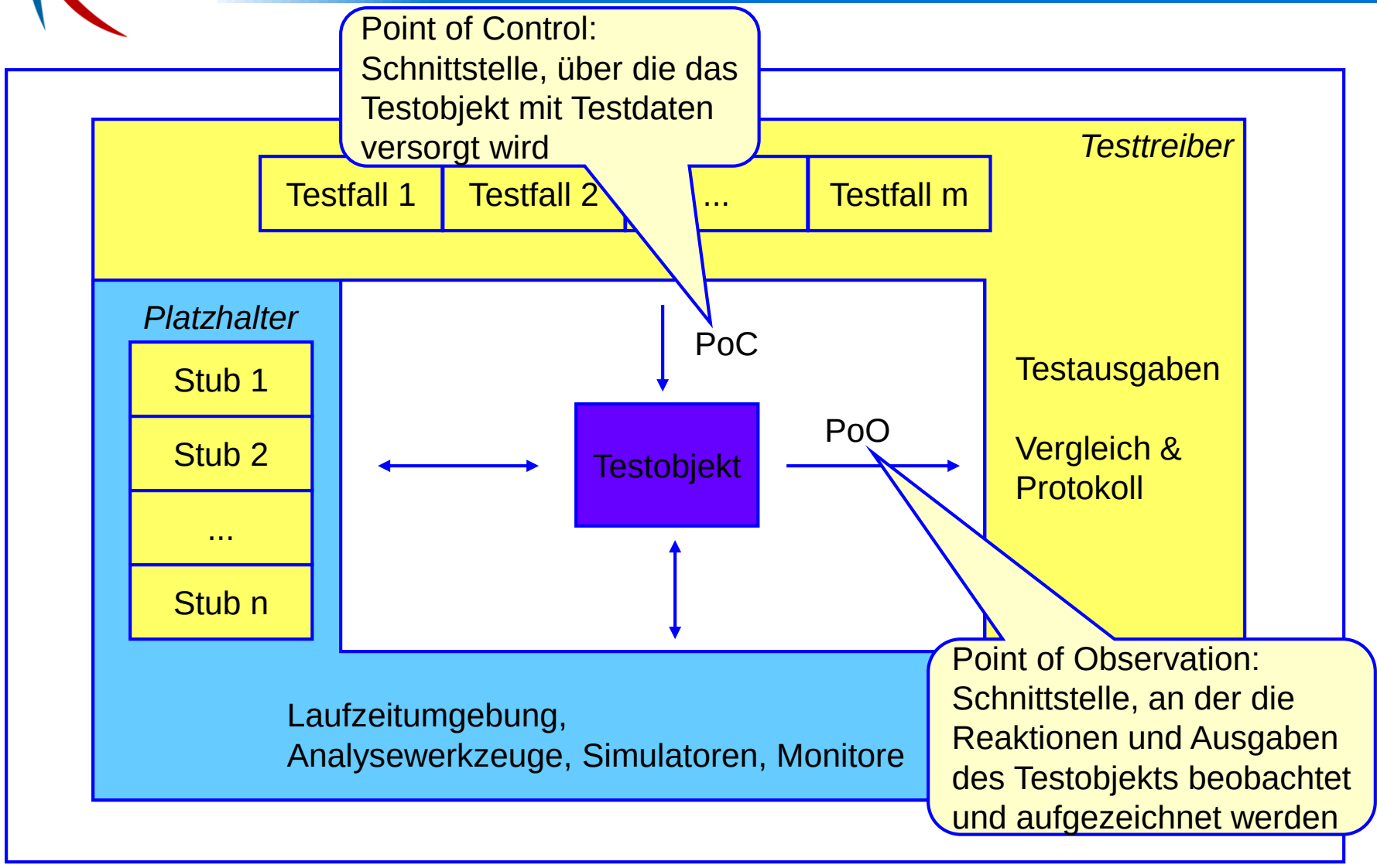
- Einführung
- Testdurchführung
- Testfallspezifikation
- Black-Box-Komponententest

# Testdurchführung

- Testumgebung (synonym: Testgeschirr bei LL oder Testrahmen) muss vorbereitet werden, beinhaltet Testtreiber und Platzhalter
- Bei der Testausführung sollte gelten
  - Keine spezielle Testvariante des Prüflings
  - Kein Abbruch des Tests, wenn Fehler erkannt
  - Keine Modifikation des Prüflings im Test
  - Kein Wechsel zwischen Test und Debugging



# Aufbau eines Testrahmens



# Testbegriffe (3)

- Testrahmen (test bed)
  - Sammlung aller Programme (u. a. Testtreiber und Platzhalter), die notwendig sind, um Testfälle auszuführen, auszuwerten und Testprotokolle aufzuzeichnen
- Testumgebung
  - Gesamtheit aller Hardware- und Softwarekomponenten (auch der Testrahmen), die notwendig sind, um Testfälle durchzuführen
- Platzhalter (stub)
  - Platzhalter werden beim Komponenten- und Integrationstest benötigt, um noch nicht implementierte Komponenten für die Testdurchführung zu ersetzen bzw. zu simulieren
- Testtreiber
  - Programm, das ermöglicht, ein Testobjekt ablaufen zu lassen, mit Testdaten, zu versorgen und Ausgaben/Reaktionen des Testobjektes entgegenzunehmen.

# Testbegriffe (4)

- Testspezifikation
  - Festlegung der Testobjekte und ihrer Testfälle (unter Verwendung der festgelegten Testtechniken) : was soll getestet werden?
  - sowie der Testendekriterien: wie lange soll getestet werden?
- Testendekriterium
  - Legt fest, wann der Test als abgeschlossen betrachtet wird
  - Beispiele:
    - X% der Testfälle müssen erfolgreich durchlaufen sein
      - Siehe Abdeckungsüberlegungen bei der Systematik zur Erstellung der Testfälle
      - Oft auch zusätzliche Priorisierung wichtiger Testfälle
    - X Stunden/Aufwand verbraucht
    - X Stunden kein Fehler mehr gefunden
    - Fehlerdichte X (Anzahl der gefundenen Fehler im Verhältnis zur Größe des Testobjekts) wird überschritten

# Testautomatisierung

- Verschiedene Arten von Testwerkzeugen für
  - Testmanagement: Erfassung, Katalogisierung, Verwaltung (inkl. Status), Priorisierung von Testfällen; typischerweise verbunden mit Werkzeugen zum Fehlermanagement, Konfigurationsmanagement und Anforderungsmanagement
  - Testspezifikation: insbes. Unterstützung bei der Testdatengenerierung (aus Datenbank, Code, Schnittstelle, Spezifikation)
  - Testdurchführung: umfassen Testrahmen, also Testtreiber, Eingabe von Testdaten, Monitoring, Ergebnisvergleich, Stuberzeugung; auch Capture-Replay für Oberflächentest
- Testdurchführung für Komponententest: JUnit

# JUnit

- Framework zum Testen von Komponenten (Klassen, Paketen, Schichten ...)
- Speziell für Java
- Stellt Möglichkeit zur Erzeugung eines Testrahmens zur Verfügung, der
  - die notwendige Umgebung für die zu testende Komponente bereitstellt
  - die einzelnen Operationen der Komponente aufruft
  - den vorgefundenen Programmzustand mit dem erwarteten vergleicht
  - Abweichungen von Erwartungen meldet (mögliche Fehler)

➢ **Mehr dazu in der Zentralübung**

# 2.6. Testen

- Einführung
- Testdurchführung
- Testfallspezifikation
- Black-Box-Komponententest

# Was beinhaltet ein Testfall?

- Ein Testfall legt das SOLL-Verhalten fest anhand eines Testorakels basierend auf
  - Anforderungsspezifikation
  - Benutzungshandbuch
  - Ausführbarer Prototyp (formale Spezifikation)
  - Alte Versionen

- Man beschreibt Testfälle auf 2 Abstraktionsebenen
  - Logische Testfälle (Wertebereich für Ein/Ausgabe)
    - Z.B. Festlegung durch Äquivalenzklassen
  - Konkrete Testfälle (spezifische Ein/Ausgaben)
    - Z.B. Festlegung konkreter Repräsentanten aus den Äquivalenzklassen

# Beschreibung eines Testfalls

- **Testfallbeschreibung**
  - Name (identifiziert den untersuchten Fall)
  - Getestete Anforderung (Bezug zur Anforderung)
  - Typ (Komponenten, Integration, System…)
  - Vorbedingung (welche Daten liegen im System vor)
  - Nachbedingung (wie wurden die Daten verändert)
  - Testinfrastruktur (Benötigte Testsoftware, z.B. auch Stubs, Treiber)
  - (Ggf. unvollständig) ausgefülltes Testprotokoll, das für jeden Schritt angibt
    - Eingabe
    - erwartete Ausgabe
    - erwartete Ausnahmen
- Bei logischen Testfällen nur Angabe von Wertebereichen, bei konkreten Testfällen konkrete Werte für Ein- und Ausgabe und Ausnahmen.
- Bei konkretem Testfall steht nach Ausführung im Testbericht, wie sich die Funktion verhalten hat (unerwartete Ausgaben?) und der Teststatus zeigt an, ob der Testfall erfolgreich durchgelaufen ist => Testprotokoll

# Testfallbeschreibung für Operationstest

- Name
- Getestete Anforderung
- Typ Komponententest
- Vorbedingung
  - Bedingung zu relevanten Komponentendaten und Einschränkung der Eingaben
- Nachbedingung
  - Veränderte Komponentendaten
- Testschritte
  - Eingabe: Eingabewerte für Operation
  - erwartete Ausgabe: Ausgabewerte der Operation
  - erwartete Ausnahmen

# Testspezifikation und Testskripting

- Testspezifikation (für funktionale Anforderungen und Qualitätsanforderungen)
  - Definition von Testendekriterien
  - Ableitung und Dokumentation von logischen Testfällen

- Testskripterstellung
  - Definition von konkreten Testfällen und Testdaten
  - Implementierung der Testskripts (Vorbedingungen, Testschritte, Set-up und Tear-down Routinen, Testdatengenerierung, Testorakelermittlung)

# 2.6. Testen

- Einführung
- Testdurchführung
- Testfallspezifikation
- Black-Box-Komponententest

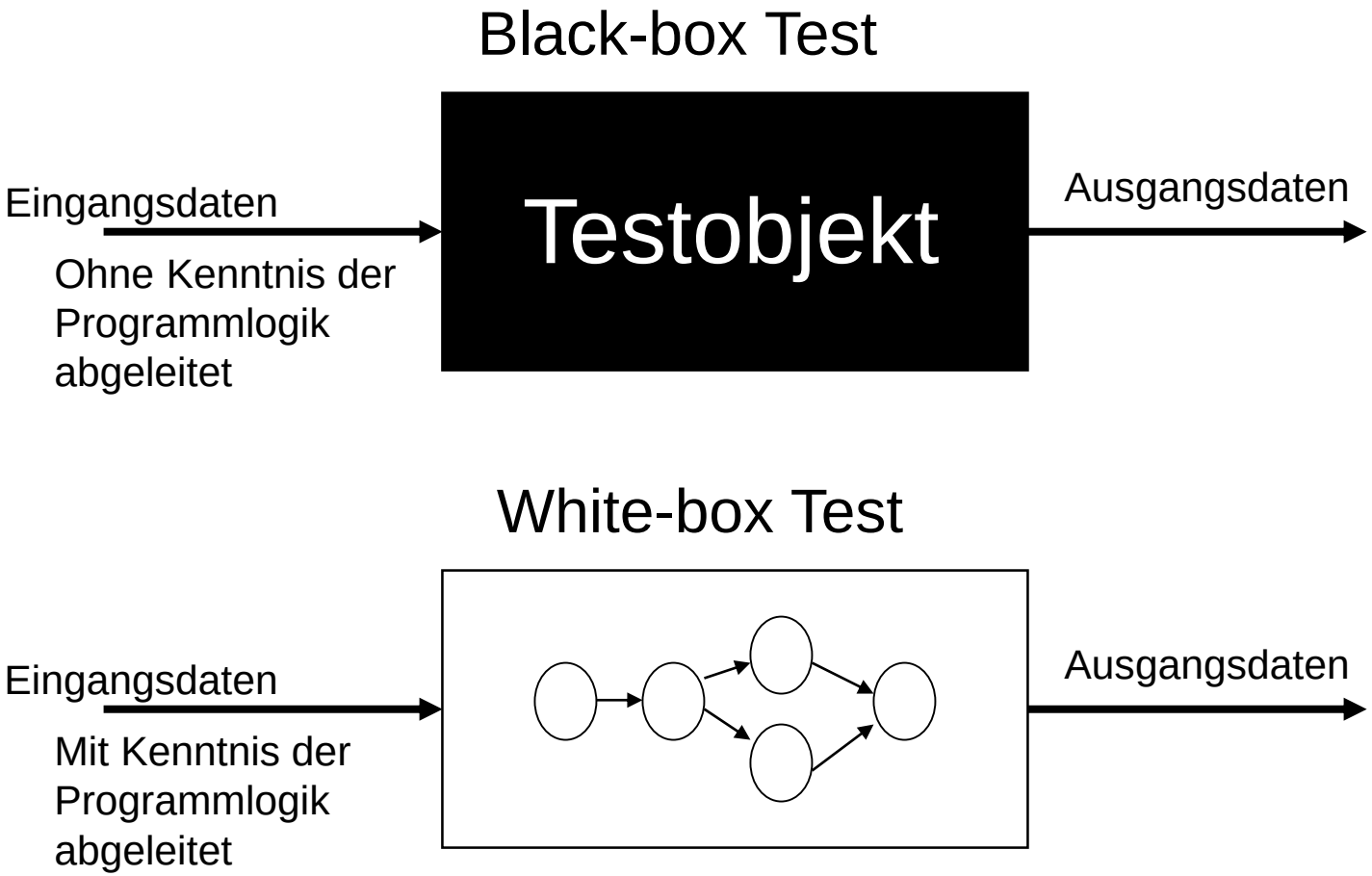
# Auswahl der Testfälle

- Ein Testfall ist gut, wenn er mit hoher Wahrscheinlichkeit einen noch nicht entdeckten Fehler aufzeigt, also
  - Repräsentativ (stellvertretend für viele andere Testfälle)
  - Fehlersensitiv (hohe Wahrscheinlichkeit)
  - Redundanzarm (prüft nicht, was andere Testfälle prüfen)
- Typische Ansätze zur Auswahl
  - Elemente überdecken
    - Elemente der Spezifikation, z.B. Funktionen => Black-Box-Test
    - Elemente des Codes, z. B. Anweisungen => White-Box
  - Kritische Pfade identifizieren
    - Teilweise sowohl bei Black-Box als auch White-Box, insbes. auch intuitiver Test
  - Daten, die statistisch die Nutzung widerspiegeln
    - Statistischer Test (hier nicht behandelt)

# Verfahren zur Testfallerstellung

- **Black-Box-Verfahren:**
  - Testen die Außenwirkung des Testobjekts
  - Keine Steuerung des Ablaufs des Testobjekts
  - Nutzen Kenntnisse über die Schnittstelle, Spezifikation
  - Beispiele: Äquivalenzklassen, Grenzwertbezogener Test, Zustandsbezogener Test
- **White-Box-Verfahren:**
  - Testen gezielt die verschiedenen Bestandteile und Abläufe des Testobjekts
  - Nutzen Kenntnisse über den inneren Aufbau (Code)
  - Beispiele: Überdeckung der Anweisungen, der Zweige, der Bedingungen, der Pfade
- **Intuitiver / erfahrungsbasierter Test**:
  - Beruht auf Kenntnis typischer Fehlerzustände
  - Sollte immer **zusätzlich** zu systematischen Verfahren durchgeführt werden

# Black-box Test vs. White-box Test

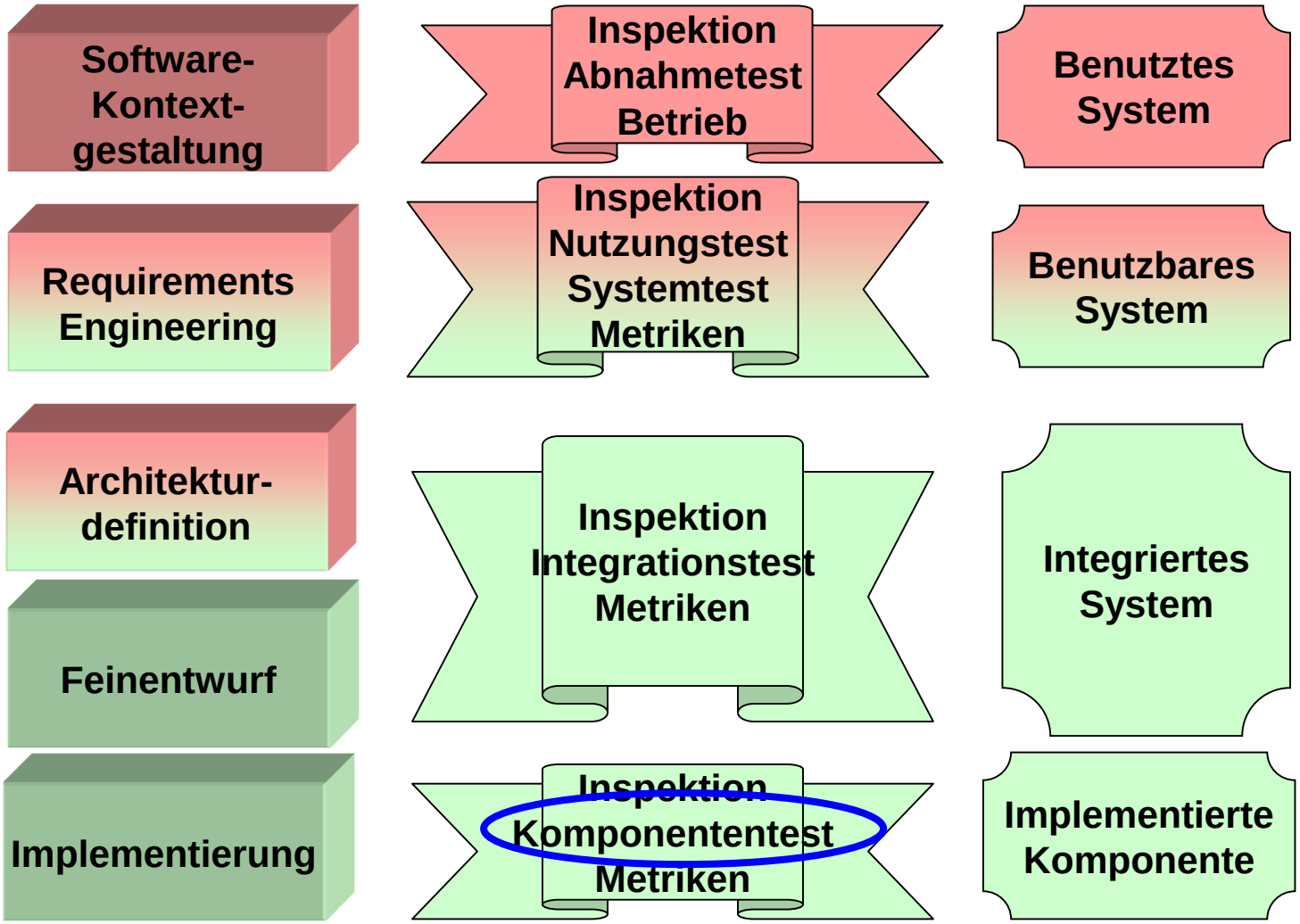


# Wichtige Unterscheidung

- **Positiv-Test:**
  - korrekte Eingaben (erwartet korrekte Ergebnisse)

- **Negativ-Test:**
  - unzulässige Eingaben (erwartet sinnvolle Ausnahmebehandlung)

# Aktivitäten und Ergebnisse der Entwicklung und Qualitätssicherung



# Komponententest

- **Komponente** = abgeschlossene Code-Einheit, z.B. Klasse, Funktion, Modul

- Typische Fehlverhalten einer Komponente:
  - Nichtterminierung
  - Falsches oder fehlendes Ergebnis
  - Unerwartete oder falsche Fehlermeldung
  - Inkonsistenter Speicherzustand
  - Unnötige Ressourcenbelastung
  - Unerwartetes Ausnahmeverhalten (z.B. Absturz)

# Black-Box-Komponenten-Test

- Leitet Testfälle aus der Beschreibung des Ein/Ausgabeverhaltens ab
- Systematik zur Erstellung der Testfälle ist Abdeckung
  - der Ein/Ausgabewerte und
  - Operationen und
  - spezifizierten Ausnahmen
- Prüft insbesondere, ob alle Anforderungen an die Komponente umgesetzt wurden
- Kann Fehlerzustände, die keine Fehlerwirkungen hervorrufen, NICHT aufdecken
  - Insbesondere unerreichbarer Code

# Äquivalenzklassentest (1)

- Äquivalenzklasse: Teilmenge der möglichen Eingabewerte
- Annahme: Programm reagiert für alle Werte aus der Äquivalenzklasse prinzipiell gleich
- Testfälle decken alle Äquivalenzklassen ab: mindestens ein Vertreter pro Klasse
- Grenzwerte: falls Werte in der Äquivalenzklasse geordnet, teste an jedem Rand den exakten Grenzwert, sowie die beiden benachbarten Werte (innerhalb bzw. außerhalb)

- Typische Äquivalenzklassen
  - Zulässige/unzulässige Datenbereiche (insbesondere bei komplexen Formaten)
  - Grenzwerte
  - Unterteilung der Eingaben so, dass die unterschiedlichen Ausgabewerte abgedeckt sind (ggf. auch Äquivalenzklassen der Ausgabewerte)

# Äquivalenzklassentest (2)

- Testfallableitung bei **mehreren** Eingaben:
  - Kombiniere alle gültigen Äquivalenzklassen der verschiedenen Eingaben
  - Kombiniere jede ungültige Äquivalenzklasse mit anderen gültigen Äquivalenzklassen

- Vereinfachung:
  - Nur häufige Kombinationen
  - Nur Testfälle mit Grenzwerten
  - Nur Abdeckung paarweiser Kombinationen

- Minimal:
  - Jede gültige Äquivalenzklasse kommt in einem Testfall vor

# Beispiele: Welche Testfälle?

- Funktion Binäre Suche:

int **search** (int[] a, int k)

**pre**: a.length > 0
**post**:

(result >= 0 **and** a[result] ==k) or

(result == -1 **and**
(not exists i. i>=0 **and** i< a.length **and** a[i] == k)
)

# Beispiel: Äquivalenzklassen (1)

- Funktion:
  int **search** (int[] a, int k)
  **pre**: a.length > 0
  **post**:
  (result >= 0 **and** a[result] ==k) or
  (result == -1 **and** (not exists i. i>=0 **and** i< a.length **and** a[i] == k))

Wieviele Testfälle nötig?

- Gültige Äquivalenzklassen:
  - Param1: a.length > 0 (gültige Eingabe),
  - Verfeinerung von Param1: a)  a.length = 1, b) a.length > 1 (Grenzwert)
  - Param2: k in a, k nicht in a (unterschiedliche Ausgabe)
  - Verfeinerung von Param2: a) k ist erstes, b) mittleres oder c) letztes Element von a oder d) k nicht in a (Grenzwerte für k in a)

- Ungültige Äquivalenzklasse:
  - Param1: a.length = 0 (Ungültige Eingabe)

# Beispiel: Äquivalenzklassen (2)

- Testfälle:

| Feld a | Element k | Ergebnis (result) |
|---|---|---|
| [] (a.length = 0) | x | ungültig |
| [x] (a.length = 1) | x (x in a) | 1 |
| [y] (a.length = 1) | x (x not in a) | -1 |
| [..,x,..] (a.length > 1) | x (x Mitte) | x |
| [...] (a.length > 1) | x (x not in a) | -1 |
| [x,....] (a.length > 1) | x (x Anfang)) | 1 |
| [...,x] (a.length > 1) | x (x Ende) | n |

# Weitere Übung

- Siehe Vorlesungsarbeitsblatt
  - Äquivalenzklassen zu Eingaben
  - Äquivalenzklassen zu Ausgaben
  - Funktionsüberdeckung

- Siehe auch Kapitel 19.8 bei Ludewig/Lichter

# Erste Zusammenfassung Test

- Test dient der Aufdeckung von Fehlerwirkungen
- Test ist auf verschiedenen Ebenen nötig
- Typische Verfahren für die Erstellung von Testfällen für Komponentenebene und Systemebene sind Black-Box- und White-Box-Test.
- Eine typische Methode für Black-Box-Test ist Äquivalenzklassenbasierter Test.