SOFTWARE ENGINEERING

Elite Graduate Program

Projektmanagement: Qualitätsmanagement

Martin Wirsing

Institut für Informatik
Ludwig-Maximilians-Universität München

WS 2006/07

Ziele

- Grundlegende Begriffe von Softwarequalität und Methoden und Vorgehensweisen des Software-Qualitätsmanagement kennen lernen
- Grundarten des Testens kennen lernen

Qualität

- Der Begriff Qualität klingt häufig langweilig und wird oft mit schlechter Qualität in Verbindung gebracht
- Qualität wird oft synonym genutzt mit
  - **Dauerhaftigkeit, Stabilität, Fehlerfreiheit**
    - **z. B. für Kleidung, Möbel, Autos**
- Qualität ist aber eigentlich ein hoch spannendes Thema:
  - **Gute Produktqualität führt zu *zufriedenen Kunden***
  - **Gute Produktqualität führt zu *zufriedenen Mitarbeitern***
- Es macht überhaupt keinen Spaß, schlechte Qualität abzuliefern!

Praxisbeispiel: Qualitätsmanager

- In der Softwarefirma XY wurde der Mitarbeiter Müller zum Qualitätsmanager ernannt.
  - **Die Gründe: er war derjenige, auf den man im Projektgeschäft am einfachsten verzichten konnte.**
  - **Er stellt ein Monster an Formalismen zusammen und nennt es Qualitätsmanagementsystem.**
- Man beauftragt eine kleine (von ihren Auftraggebern abhängige) Firma mit der Zertifizierung nach ISO 9000 und erreicht sie tatsächlich auch.
- Und was ist mit der Software? Die ist immer noch so schlecht wie vorher.

Der Begriff Qualität

- **Qualität** (ANSI/ASQC A3-1978), ISO 8402, DIN55350/1, IEEE-Norm):
  “Qualität ist die Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht”
  - Mit anderen Worten, Qualität ist an den Benutzeranforderungen zu messen (e.g. Korrektheit, Verlässlichkeit, Verwendbarkeit, ...)
  - Software altert nicht, aber:
    - Es ist davon auszugehen, dass Software eine lange Lebensdauer hat und dabei immer wieder adaptiert wird
    - Es gibt somit auch am System selbst orientiere Facetten der Qualität e.g. Wartbarkeit, Portierbarkeit, Testbarkeit, ..
- In den folgenden Folien lösen wir uns bei der Motivation vom Begriff Qualität und reden von: „so richtig guter Software“.
- Fragen:
  - **Was macht denn Software so richtig gut?**
  - **Was ist Ihre Lieblingssoftware und was gefällt Ihnen daran?**
  - **Welche Software gefällt Ihnen nicht? Warum?**

Gute Software ist z. T. subjektive Empfindung

- Hängt ab von der Erwartungshaltung des Nutzers
- Hängt ab von sonstigen Eigenschaften und Kontext
  - **Spiel, das öfter mal abstürzt**
  - **Office-Programm, das öfter mal abstürzt**
- Aber: es gibt auch klar messbare Kriterien
- Die Zufriedenheit des Kunden hängt aber nicht nur von der Software ab (vgl. Projektinitialisierung – Kundenbefragung)

SW-Qualität nach DIN ISO 9126

- **Funktionalität:**
  - Vorhandensein von Funktionalität entsprechend den Anforderungen
  - Richtigkeit, Angemessenheit, Interoperabilität, Ordnungsmäßigkeit, Sicherheit
- **Zuverlässigkeit:**
  - Fähigkeit der Software, ihr Leistungsniveau unter festgelegten Bedingungen über einen festgelegten Zeitraum zu erbringen
  - Reife, Fehlertoleranz, Wiederherstellbarkeit
- **Benutzbarkeit:**
  - Aufwand für Benutzung, Beurteilung von Benutzergruppen
  - Verständlichkeit, Erlernbarkeit, Bedienbarkeit
- **Effizienz:**
  - Verhältnis zwischen Leistungsniveau der SW und Umfang der eingesetzten Betriebsmittel
  - Zeitverhalten, Verbrauchsverhalten
- **Änderbarkeit:**
  - Aufwand für Korrekturen, Verbesserungen, Anpassungen
  - Analysierbarkeit, Modifizierbarkeit, Stabilität, Prüfbarkeit
- **Übertragbarkeit:**
  - Eignung zur Übertragung in andere SW oder HW Umgebung
  - Anpassbarkeit, Installierbarkeit, Konformität, Austauschbarkeit

Wie schreibt man richtig gute Software?
Praxisbeispiele

- **Beispiel 1 (Wiederholung):**
  - **PL: Wir liefern am 4. Juli aus.**
  - **Mitarbeiter: Das schaffen wir nicht.**
  - **PL: Das ist mir egal. Wenn ich sage, wir liefern aus, dann liefern wir aus. Strengen Sie sich an.**
- **Beispiel 2:**
  - **Software wurde entwickelt**
  - **Software wird an Kunden übergeben**
  - **Das Programm, das nachts Daten vom Vortag verarbeiten soll, läuft 46h lang.**

Maßnahmen

- Was könnte man in solchen Situationen tun?
  - **Ein gutes Ergebnis erzielt man, wenn man die richtigen Dinge tut.**
  - **Ein gutes Ergebnis erzielt man nicht durch Abnahmeprüfungen.**
- Prüfungen stellen sicher, dass keine Fehler passiert sind. Sie sind eine letzte Sicherheitsmaßnahme.
- Tests und Prüfungen des Messung des existierenden Qualitätsniveaus sind Beispiele für *analytische* Qualitätssicherung.
- *Konstruktive* Qualitätssicherung verbessert den Software-Erstellungsprozess, der im Projekt gelebt wird:
  - **Methoden, Sprachen, Werkzeuge, Richtlinien, Checklisten, die dafür sorgen, dass das Produkt bzw. der Erstellungsprozess bestimmte Eigenschaften besitzt.**

Wer sorgt für Qualität?

- In der Praxis oft gehörter Satz:
  - **„Die Frau Schmidt ist für die Qualität zuständig.“**
  - **Übersetzt hieße das: „Unser Projekt schreibt so richtig gute Software. Das macht die Frau Schmidt“.**
  - **Klingt nicht nur unplausibel, es ist auch so.**
- Damit richtig gute Software rauskommt, muss sich jeder im Projekt anstrengen.
- Übersetzt: **Qualität geht alle an.**

Fazit

- **Qualität geht jeden an**. Jeder einzelne im Projekt sorgt dafür. Ein **Qualitätsbeauftragter unterstützt** dabei, mehr aber auch nicht.
- Es gibt **konstruktive und analytische Maßnahmen** zur Qualitätssicherung.
  - **Mehr Qualität erzeugen kann man nur mit konstruktiven Maßnahmen.**
  - **Analytische Maßnahmen stellen nur sicher, dass ein Qualitätsstandard erreicht ist.**
- Man kann keine Qualität in Software hineinprüfen. **Qualität** kann man nur in Software hinein***konstruieren***.

Qualitätsmanagement

„Qualitätsmanagement umfasst alle Tätigkeiten der Gesamtführungsaufgabe, welche die Qualitätspolitik, Ziele und Verantwortungen festlegt sowie diese durch Mittel wie Qualitätsplanung, Qualitätslenkung, Qualitätssicherung und Qualitätsverbesserung im Rahmen des Qualitätsmanagementsystems verwirklichen.“

[DIN ISO 8402]

Qualitätsmanagement umfasst alle Tätigkeiten, um die Qualität von **Prozessen und Produkten** sicherzustellen.

- **Qualitätsplanung:** Festlegung von überprüfbaren Qualitätszielen und Planung von Maßnahmen zur Erreichung dieser Ziele (dokumentiert im QS-Plan)
- **Qualitätslenkung:** konstruktive Maßnahmen
- **Qualitätssicherung (analytische Maßnahmen):** Umsetzung der Maßnahmen des Qualitätssicherungs-Plans
- **Qualitätsverbesserung:** Prozessverbesserungsmaßnahmen

Prinzip der Qualitätszielbestimmung

- **Festlegung von Qualitätszielen vor Beginn der Entwicklung**
  - das Team weiß, wohin es arbeitet
  - der Kunde weiß, worauf er sich einläßt

- **Die Qualitätsziele werden im QS-Plan festgelegt**
  - die Kritikalität des Systems ist ein wichtiger Anhaltspunkt für die Festlegung der Ziele

Prinzip der quantitativen Qualitätssicherung

- Ingenieurmäßige Qualitätssicherung ist undenkbar ohne die Quantifizierung von Soll- und Istwerten (Rombach, 93).
- Qualitätsziele müssen überprüfbar und messbar sein

Beispiel:

- *Kein überprüfbares Ziel:*

  Das System muss performant sein.

- *Überprüfbar:*

  Die Verarbeitungszeit eines Vertrages durch das System muss unter 8 Sekunden liegen

Prinzip der maximal konstruktiven Qualitätssicherung

Der Aufwand für die analytische Qualitätssicherung soll durch eine möglichst gute konstruktive Qualitätssicherung gering gehalten werden

- Besser vorbeugen als heilen
- Fehler, die nicht gemacht werden können, brauchen auch nicht behoben zu werden

Beispiel

Beim Einsatz einer Programmiersprache mit statischer Typprüfung können Typfehler während der Laufzeit nicht auftreten

Prinzip der frühen Fehlererkennung und -behebung

- **Fehler in den frühen Projektphasen sind die teuersten**
  - Anforderungen des Kunden wurden nicht richtig verstanden
  - Inkonsistenzen im Anforderungsdokument
  - Eine verzögerte Fehlerentdeckung führt zu einem exponentiellen Kostenanstieg.

- **Aufmerksamkeit in die frühen Phasen der SW -Entwicklung investieren**
  - Fehler durch konstruktive Maßnahmen verhindern
  - Fehler, die dennoch gemacht werden, durch sorgfältige Prüfungen der Dokumente in den frühen Phasen rechtzeitig erkennen

Prinzip der unabhängigen Qualitätssicherung

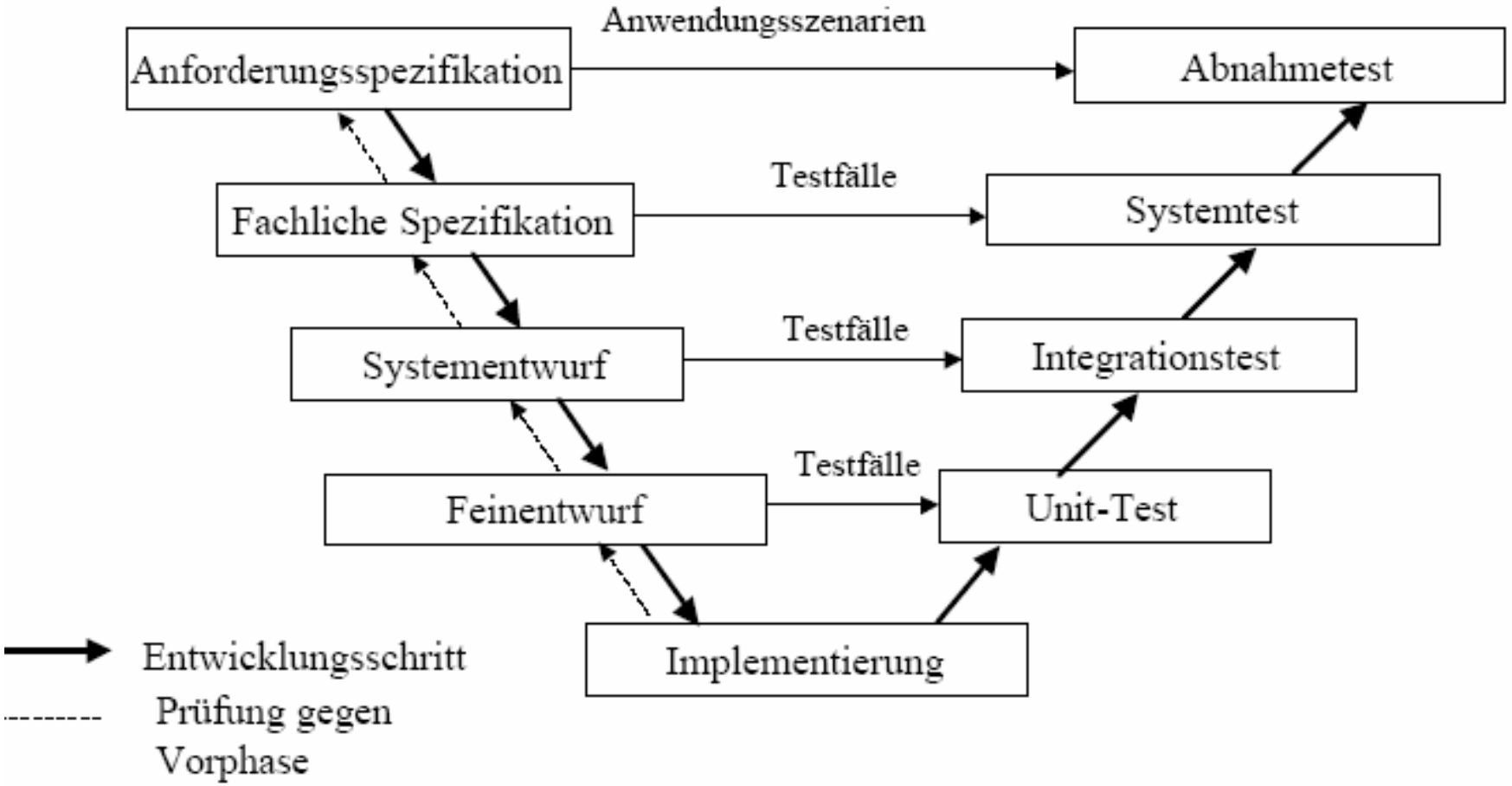
- Ziel der analytischen Qualitätssicherung ist es, Fehler und Mängel aufzudecken.
- Myers: „Testing is a destructive process, even a sadistic process“
- Entwickler können nicht gleichzeitig konstruktiv und destruktiv denken.
  - **Die Entwickler eines Teilprodukts sollten nicht die analytische QS für dieses Teilprodukt vornehmen**
  - **Ausnahme: Test-driven Development**

    (siehe XP - Extreme Programming)

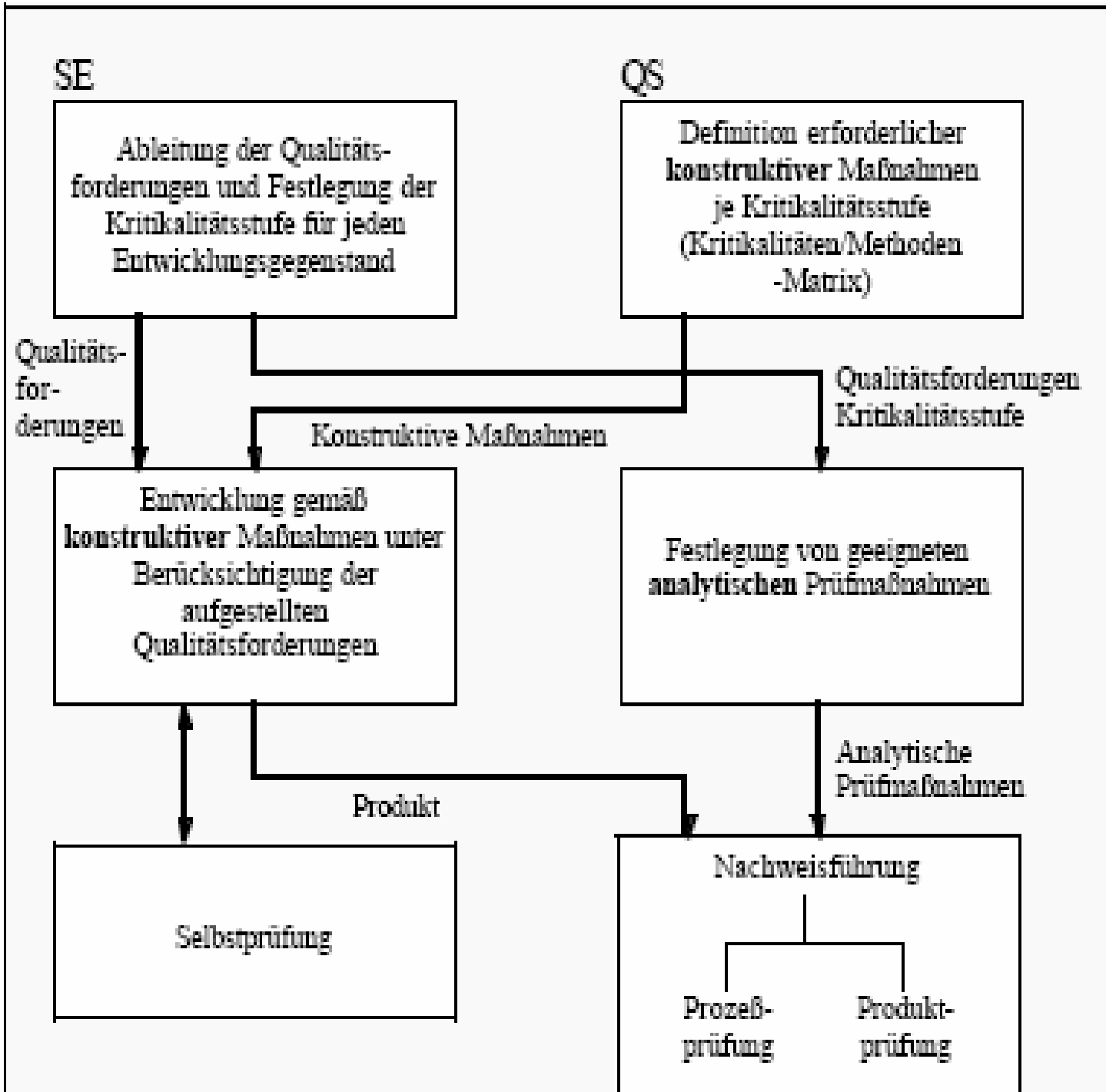
Prinzip der entwicklungsbegleitenden Qualitätssicherung

- **Einbettung der QS in den organisatorischen Ablauf der SW -Entwicklung**
  - Ein Teilprodukt steht der nächsten Phase erst dann zur Verfügung, wenn eine bestimmte Qualität erreicht ist.
- **Beispiel**
  - **V-Modell**

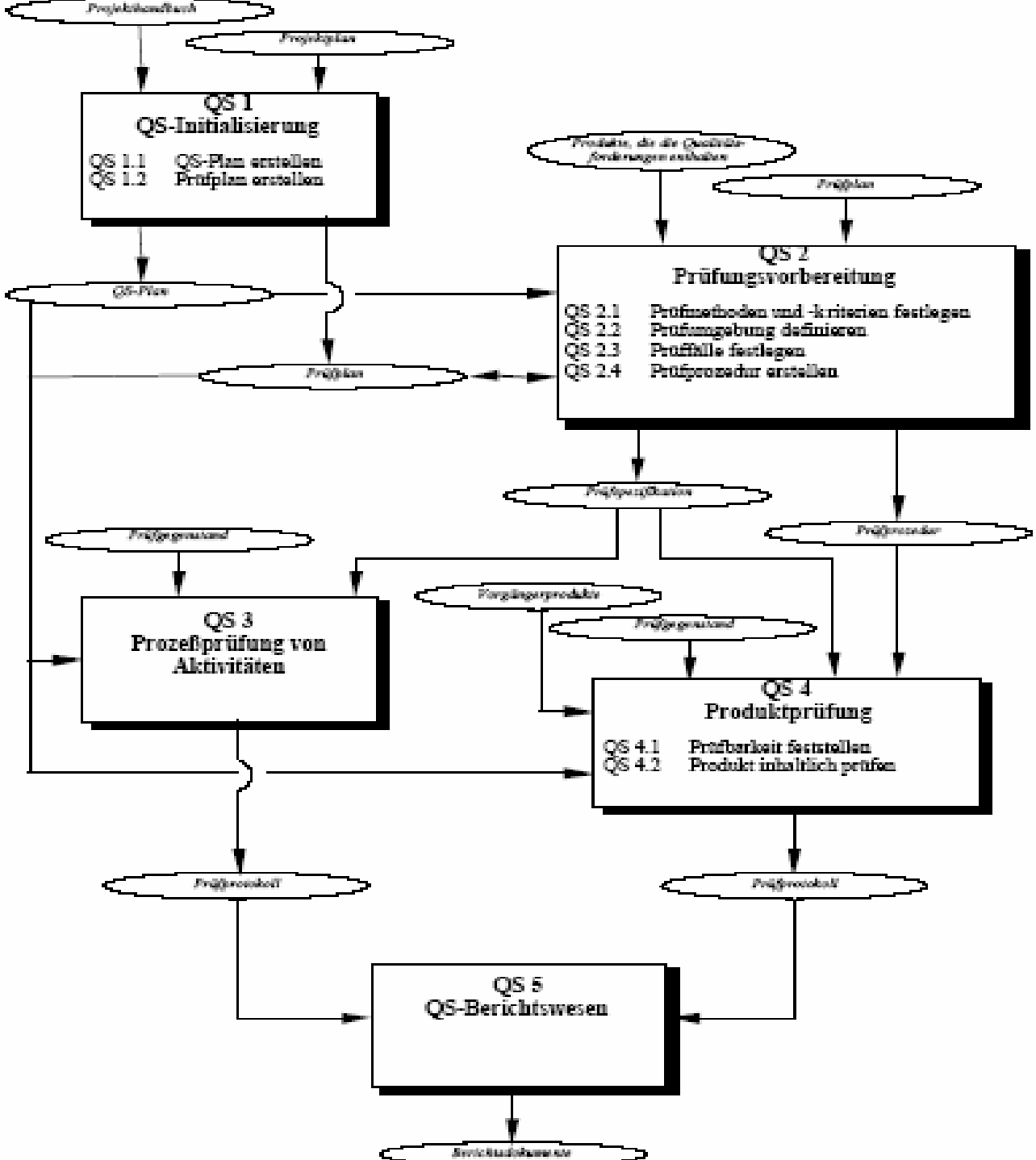
V-Modell:
Einbettung der QS in das Wasserfallmodell



- Qualitäts-sicherung vs.
- SW-Entwicklung



- **V-Modell: Qualitätssicherung**
- **Planung:**
  - QS-Initialisierung: Konstruktiv
- **Prüfung (analytisch):**
  - Vorbereitung
  - Prozessprüfung („Vorgaben eingehalten“)
  - Produktprüfung
- **Leitung (administrativ):**
  - QS-Berichtswesen



Praktische Techniken der QS im Projekt

- Qualitätsmanagementplan
- Qualitätsplanung bei Projektinitialisierung
- Maßnahmen im Projekt
- Ausgewählte Beispiele
  - **Tests**
  - **Audits**

Generelle Herangehensweise

- Qualitätssicherung ist (genau wie Projektleitung oder Projektplanung) ein Prozess.
- Ganz abstrakt sind dies drei Schritte
  - **Festlegen, was die Software so richtig gut macht**

    Qualitätsziele (später: ISO 9126-1)
  - **Festlegen, wo die Software besonders gut sein muss**

    Kritikalität
  - **Festlegen, was dazu getan wird (konstruktiv und analytisch)**

    Qualitätssicherungs-Maßnahmen
- Aufschreiben dieser Festlegungen
  - **Qualitätsmanagement-Plan**

Qualitätsmanagementplan

- Qualitätssicherungs-Plan (Prüfplan)
  - **Definition Aufgabe**: **Was** ist zu tun?
    - Qualitätssicherungsmerkmale identifizieren/Metriken auswählen
    - Zu sichernde Produkte identifizieren
  - **Definition Vorgaben/Hilfsmittel**: **Wie** ist es zu tun?
    - Konstruktive Vorgaben (z.B. Style Guides, Vorlagen)
    - Analytische Vorgaben (Verfahren, Werkzeuge)
  - **Definition Termine**: (Bis) **Wann** ist es zu tun?
    - Einordnung in den Projektplan
  - **Definition Verantwortung**: **Wer** hat es zu tun?
    - Definition von Rollen (Qualitätsmanager, Prüfer, Ersteller)
    - Zuordnung von Verantwortlichen

Qualitätssicherung ist zu Projektbeginn wichtig

- In der Projektinitialisierung werden die wesentliche Weichenstellungen für das Projekt vorgenommen:
  - **Das Team wird zusammen gestellt**
  - **Ziele werden fixiert**
  - **Die Projektplanung wird festgelegt**
- Hier ist der **Qualitätssicherer** der **Sparringspartner des Projektleiters**.
- Falls niemand für die Qualitätssicherung benannt ist/werden soll, sollte der Projektleiter hier zumindest ein **Gespräch mit einem Qualitätsberater** führen.

Maßnahmen im Projekt (1)

- Generell: als Qualitätssicherer muss man kreativ sein!
  - **Reviews (analytisch)**
    - Audits
    - Code Reviews
    - Dokumentenreviews
    - Workshops, zu denen externe Experten eingeladen werden. Zwischenergebnisse werden vorgestellt und diskutiert
  - **Tests (analytisch)**
    - Funktionale Tests, auch als Regressionstests
    - Strukturtests
    - Lasttests, etc.
  - **Durchstich**
    - fachlich
    - technisch

Maßnahmen im Projekt (2)

- **Aufbau einer insgesamt gut nutzbaren Entwicklungsumgebung**
  (Editoren, Build-Werkzeuge, Code-Analyzer, Test-Frameworks, Anbindung an das Konfigurationsmanagement, Modellierungswerkzeuge, Generatoren, Debugger, etc.)
- **Ausbildungsmaßnahmen für Mitarbeiter**
  - **Schulungen**
  - **Workshops**
  - **Vorträge**
  - **Team-Rotation**
- **Verbesserung der Kommunikation**
  - **Spezielle Meetings, Statusrunden**
  - **Vorträge über Fachthemen**

Maßnahmen im Projekt (3)

- **Guidelines, Richtlinien**
  - **Coding-Richtlinien**
  - **Styleguides**
  - **gemeinsame Glossare**
  - **Richtlinien zur Erstellung von Dokumenten**
  - **Entwickeln von Checklisten, Templates**
    - **Operationalisieren die Richtlinien/Guidelines**
- **Vorsicht! Nicht übertreiben! Keine Schrankware produzieren. Richtlinien müssen noch anwendbar bleiben.**

Beispiel: Coding Standards (NASA)

2 Naming Conventions

2.1 Descriptive Names

Names should be readable and self-documenting. Abbreviations and contractions are discouraged. Shorter synonyms are allowed when they follow common usage within the domain.

2.2 Valid Characters

All names should begin with a letter. Individual words in compound names are generally differentiated by capitalizing the first letter of each word. The use of special characters (anything other than letters, digits and underscores) is discouraged.

2.4 Function Names

Function names should preferably be an action verb. Boolean-valued functions may be clearest with the "is" prefix as in "isEmpty()".

3 Style Guideline

3.2 Comments

3.2.1 Automated Documentation Comments

Many different tools use different conventions for flagging comments for it to automatically use. Since a tool hasn't been found (and one looked at seriously does not use the /** convention) there is no reasonable convention to adopt as of yet. When such a tool is chosen, this document should be updated.

(Java) Code comments should use the single line comment delimiter //.

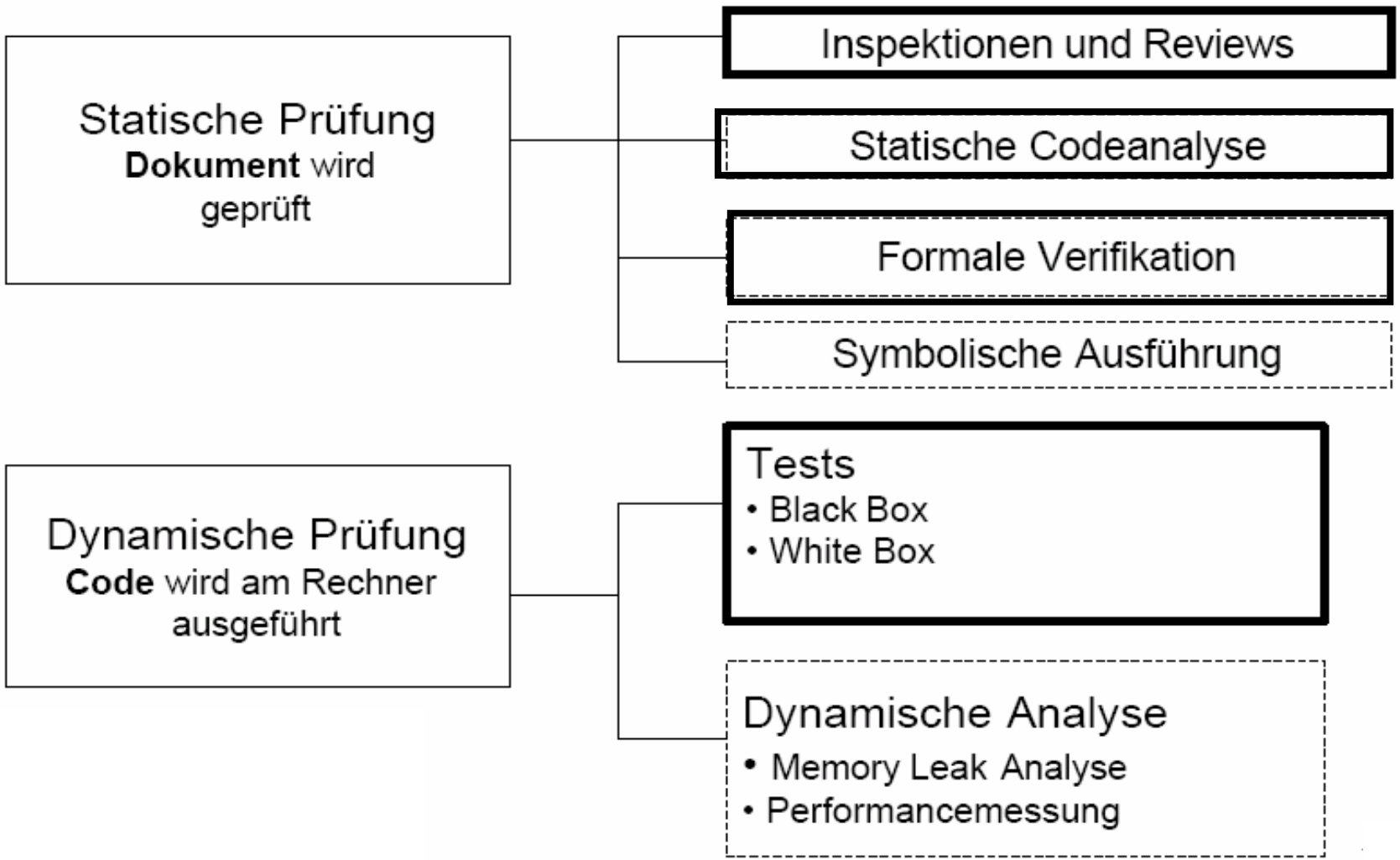
3.2.3 Blank Lines

In source files, use two blank lines to separate each function definition, or in some way make it easy to tell where the new function begins.

Maßnahmen im Projekt (4)

- **Projektablage (Verzeichnis oder Konfigurations-Mgmt) organisieren**
  - **Dafür sorgen, das man Informationen wieder finden kann:**
    - **Konventionen für die Ablage von Dokumenten**
    - **Beschaffen eines Such-Tools**
    - **Händischer Index (z. B. HTML) über die wichtigsten Dokumente**
    - **…**

Verfahren der analytischen Qualitätssicherung



Analytische Verfahren

- **Ansatz:**
  - Keine Qualität per se:
    - Feststellen der Qualität nach der Realisierung
    - Korrektur zur Qualitätssteigerung
  - Meist: Funktionalität, Zuverlässigkeit, Änderbarkeit
- **Verfahren:**
  - Analysierend = statische Überprüfung:
    - Statische Analyse
    - Review (Inspektion, Audit, Walkthrough)
    - Verifikation
  - Testend = dynamische Überprüfung (Überprüfen in der Ausführung):
    - Funktionaler Test
    - Strukturtest

Analytische Verfahren: Statische Analyse

- **Bewerten des Produkts mittels Metriken**
  - Schwerpunkt: Zuverlässigkeit, Änderbarkeit
  - Verschiedene Metriken:
    - **Komponenten:**
      - Umfang: LOC
      - Innere Struktur: Kontrollflusskomplexität
      - Schnittstelle: Anzahl Methoden/Klasse, Schnittstellenbreite
    - **Systeme:**
      - Umfang: LOC
      - Kopplung: Anzahl Aufrufe in/aus Komponente
      - OO-Metriken
- **Statische Überprüfung von Eigenschaften von Modellen oder Code**
  - Klassisch: Code, heute: Modelle
  - Typprüfung, Konsistenzprüfung, Deadlockfreiheit, …

Beispiele: Metriken

- **Komponentenmetriken: „algorithmische“ Komplexität**
  - Halstead: Anzahl (verwendeter) Operatoren/Operanden
  - McCabe: Anzahl Knoten/Kanten Kontrollflussgraph
- **Strukturmetriken:**
  - Komponenten/Subkomponenten
  - Schnittstellenkomplexität (Anzahl, Breite, Struktur)
- **Weitere:**
  - OO-Metriken: Anzahl Vererbungsstufen, Attribute, Methoden
  - Fokus auf Implementierungsgüte

Auditierung

- Audit:
  - **Intensive Untersuchung eines Projekts**
  - **In der Regel durch externe Auditoren**
  - **Sichtung von Unterlagen**
  - **Führen von Interviews**
  - **Kann sowohl formale Kriterien (einfach) als auch Projektinhalte und Vorgehensweise (schwierig) prüfen.**
- Ergebnisse:
  - **Bewertung des Projektstatus, z. B. in Form einer Projektampel**
  - **Liste vorgeschlagener Maßnahmen**

Sinnhaftigkeit von Audits

- Hat ein Auditor überhaupt eine Chance herauszubekommen, was im Projekt los ist?
- Vorab: Ein **Audit ist hilfreich**
  - Die Vorbereitung auf ein Audit selbst ist schon hilfreich
  - Offenheit hilft, weil man als Auditierter selbst ein offenes Feedback bekommt und man das Projekt verbessern kann.
  - Ein Audit als bloße Kontrollveranstaltung mit anschließender Bestrafung verfehlt seinen Zweck.
- **Einschätzung** der Projektsituation **ist schwierig**:
  - gerade bei kurzen Audits (wenige Stunden).
  - wenn die Auditierten sich selbst und die Projektsituation falsch einschätzen und diese Meinung überzeugend vertreten.
  - Trotzdem zeigt die Praxis, dass der Auditor schon nach kurzer Zeit einen guten Eindruck von der Projektsituation bekommt.
- Bei richtiger Handhabung des Werkzeugs Audit sind **„Verschwörungen“** eines Projekts **gegen den Auditor nicht nötig** und finden nicht statt.

Ablauf eines Audits

- Vorab werden die wichtigsten Projektunterlagen an die Auditoren verteilt: Projekthandbuch, Projektauftrag, Projektplanung, QS-Plan
- Auditoren bewerten die Unterlagen und führen Interviews. Interviewed werden: Projektleiter, Qualitätsbeauftrager, ggf. Auftraggeber und Projektmitarbeiter
- Auditoren erstellen einen Auditbericht mit einer Bewertung und vorgeschlagenen Maßnahmen.
- Nach einiger Zeit (z. B. 4-6 Monaten) erfolgt ein Nachaudit oder eine Nachbesprechung, um die Wirkung der Maßnahmen zu überprüfen und den Status neu zu bewerten.

Fragestellungen im Audit

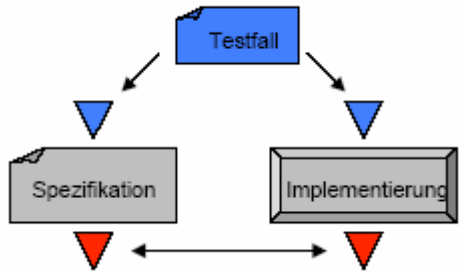
- **Generell:** Im Audit kann man die Dinge abprüfen, die "so ganz einfach und selbstverständlich" klingen. (Dies sind die bisher in der Vorlesung vorgestellten Inhalte.)
- **Projektauftrag:** Ist der Projektauftrag klar definiert? Wo ist er? Ist er unterschrieben? Sind die Ziele klar formuliert? Ist klar, welche Ergebnisse das Projekt liefern soll?
- **Risiken:** Was sind die 5 Top-Risiken im Projekt? Hat dazu jeder im Projekt die gleiche Meinung?
- **Planung:** Gibt es eine aktuelle Planung? Kann der PL diese erklären? Ist die Planung im Team bekannt?
- **Projekthandbuch:** Gibt es ein aktuelles Projekthandbuch?
- **Projektvorgehen:** Ist mit dem Auftraggeber das Abnahmeverfahren geklärt? Ist das Change Request-Verfahren mit dem Auftraggeber abgesprochen? Ist das Verfahren allen bekannt?
- **Organisatorisches:** Ist das passende Know-how im Team vorhanden? Kennt jeder im Projekt das Eskalationsverfahren?
- **Projektstruktur:** Wie sieht die Projektstruktur aus? Ist das Projekt gut gegliedert oder ein unstrukturierter Klumpen?
- **Qualitätssicherungsplan:** Welche QS-Maßnahmen sind geplant?

Analytische Verfahren: Verifikation

- **Ziel: Vollständige Überprüfung der Korrektheit**
- **Technik:**
  - Formalisierung (Teil-)Spezifikation
  - Formalisierung Implementierung
  - Mathematische Überprüfung
- **Verfahren:**
  - Interaktiv (Theorembeweiser)
  - Automatisch (Model Checker)
- **Einschränkungen:**
  - Unter Umständen aufwendig und nur von Spezialisten durchführbar
  - Überprüfung der Formalisierungen

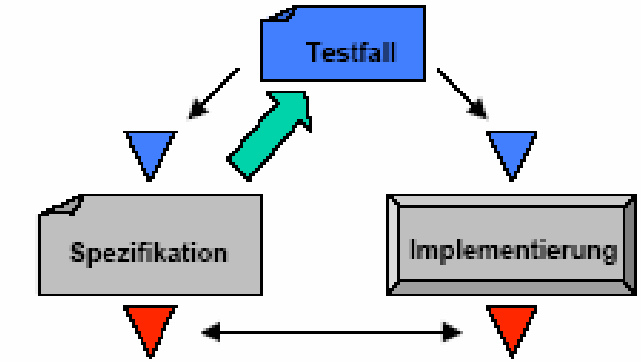
Analytische Verfahren: Testen

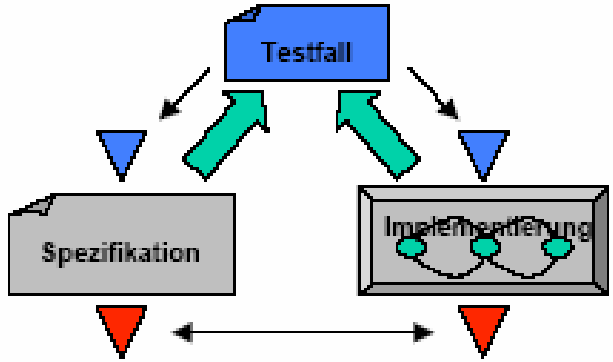
- **Testen** ist der Prozess, ein SW-Produkt durch manuelle oder automatisierte Hilfsmittel zu bewerten, um damit die Erfüllung der speziellen Anforderungen nachzuweisen
- **Testziele:**
  - Funktionalität: Richtigkeit
  - Zuverlässigkeit: Fehlertoleranz, Reife
- **Testprinzip:**
  - Überprüfung Implementierung vs. Spezifikation
  - Überprüfungsmittel: Testfälle
  - Testgüte: Überdeckung des Testraums
  - Spezialfall: Regressionstest (Änderungstest)



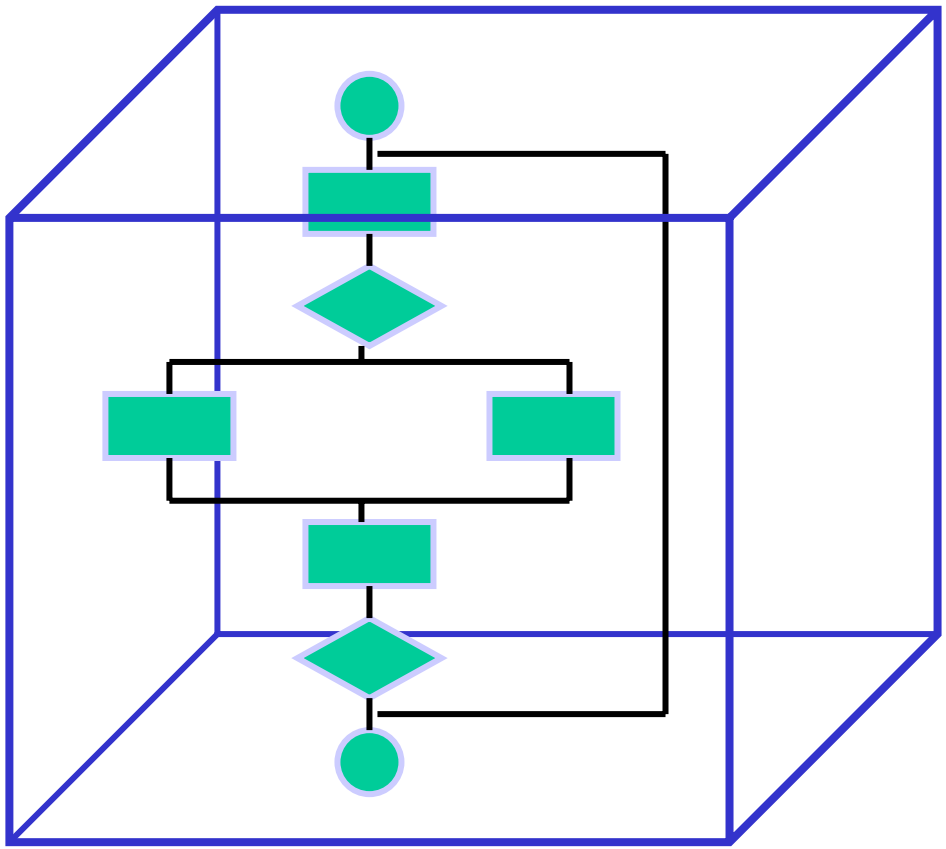
Testen: Formen und Verfahren

- **Ansätze:**
  - **Strukturtest (White-Box-Test):**
    - Kontrollflussorientiert
    - Datenflussorientiert
  - **Funktionaler Test (Black-Box-Test)**
    - Äquivalenzklassentest
    - Testen spezieller Werte
    - Zufallstest
    - Test von Automaten



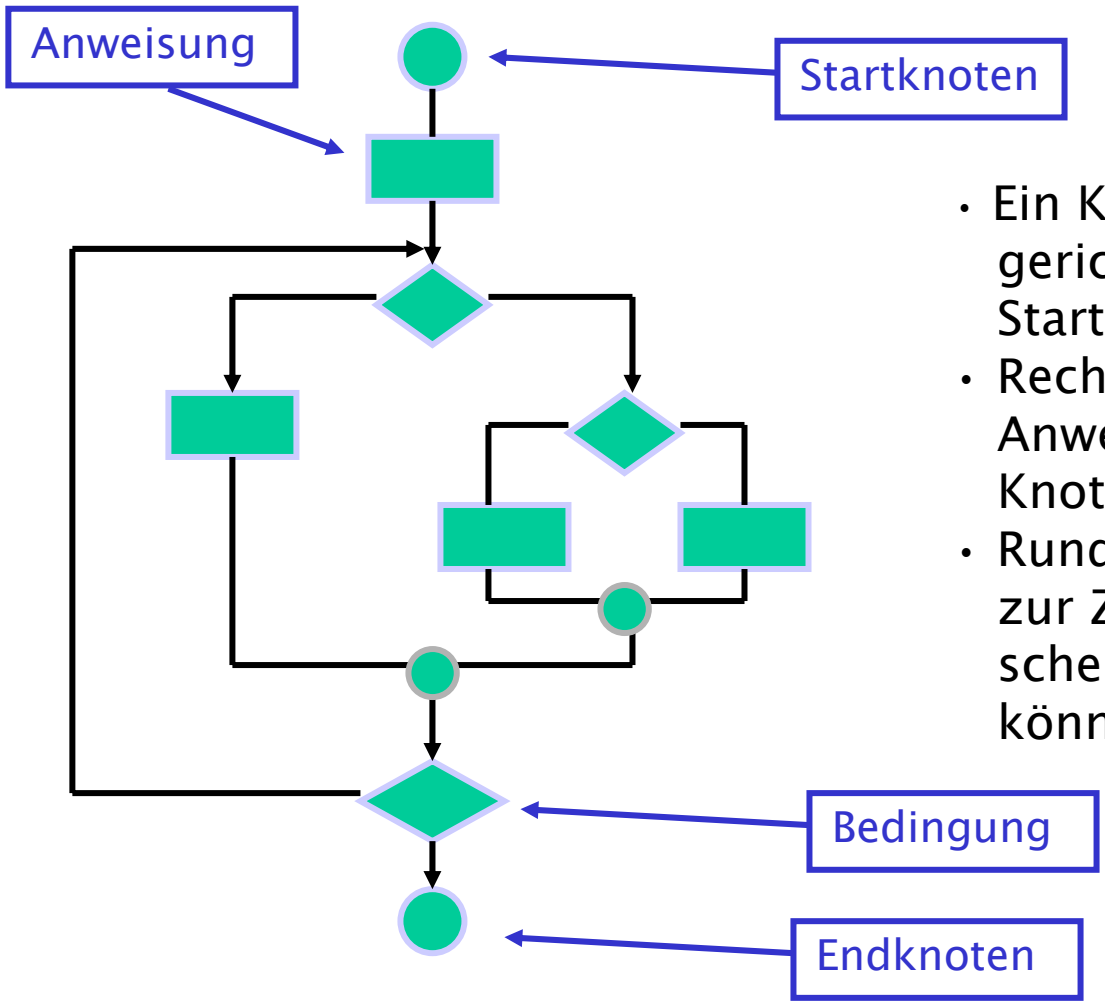


White-Box-Test



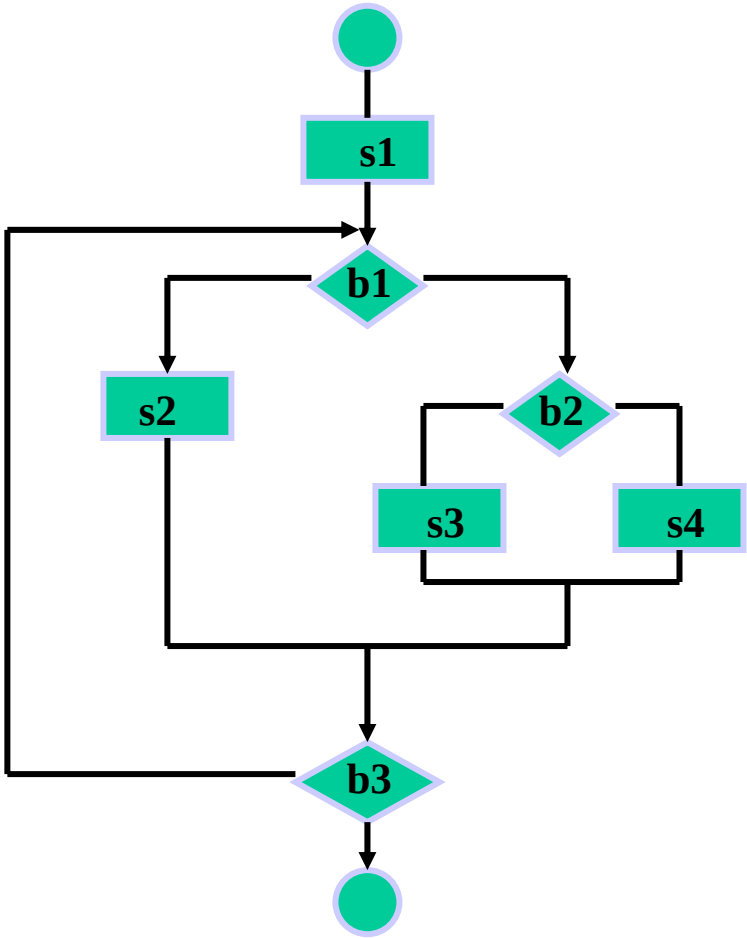
… Ziel ist, dass alle Anweisungen, Bedingungen und Pfade mindestens einmal ausgeführt werden…

Kontrollflussgraph



- Ein Kontrollflussgraph ist ein gerichteter Graph mit genau einem Start- und genau einem Endknoten.
- Rechteckige Knoten stellen Anweisungen dar, rautenförmige Knoten Bedingungen.
- Runde Knoten im Inneren dienen nur zur Zusammenführung von Fallunterscheidungen und können weggelassen werden.

Beispiel Kontrollflussgraph




```
s1;
do {
   if ( b1 ) s2;
   else {
          if ( b2 ) s3;
          else      s4;

}
while (b3)
```

Kontrollflussgraph Programmsegment

Kontrollfluss-orientierte Testverfahren

- **Kontrollfluss-orientierte Testverfahren**
  orientieren sich am Kontrollflussgraphen des Programms

- Man unterscheidet folgende Typen von Verfahren:
  - Überdeckung von Anweisungen (C0)
  - Kantenüberdeckung (C1)
  - Bedingungsüberdeckung (C2, C3)
  - Pfadüberdeckung (C4)

Beispiel: Pfadüberdeckungsverfahren

Kriterium:

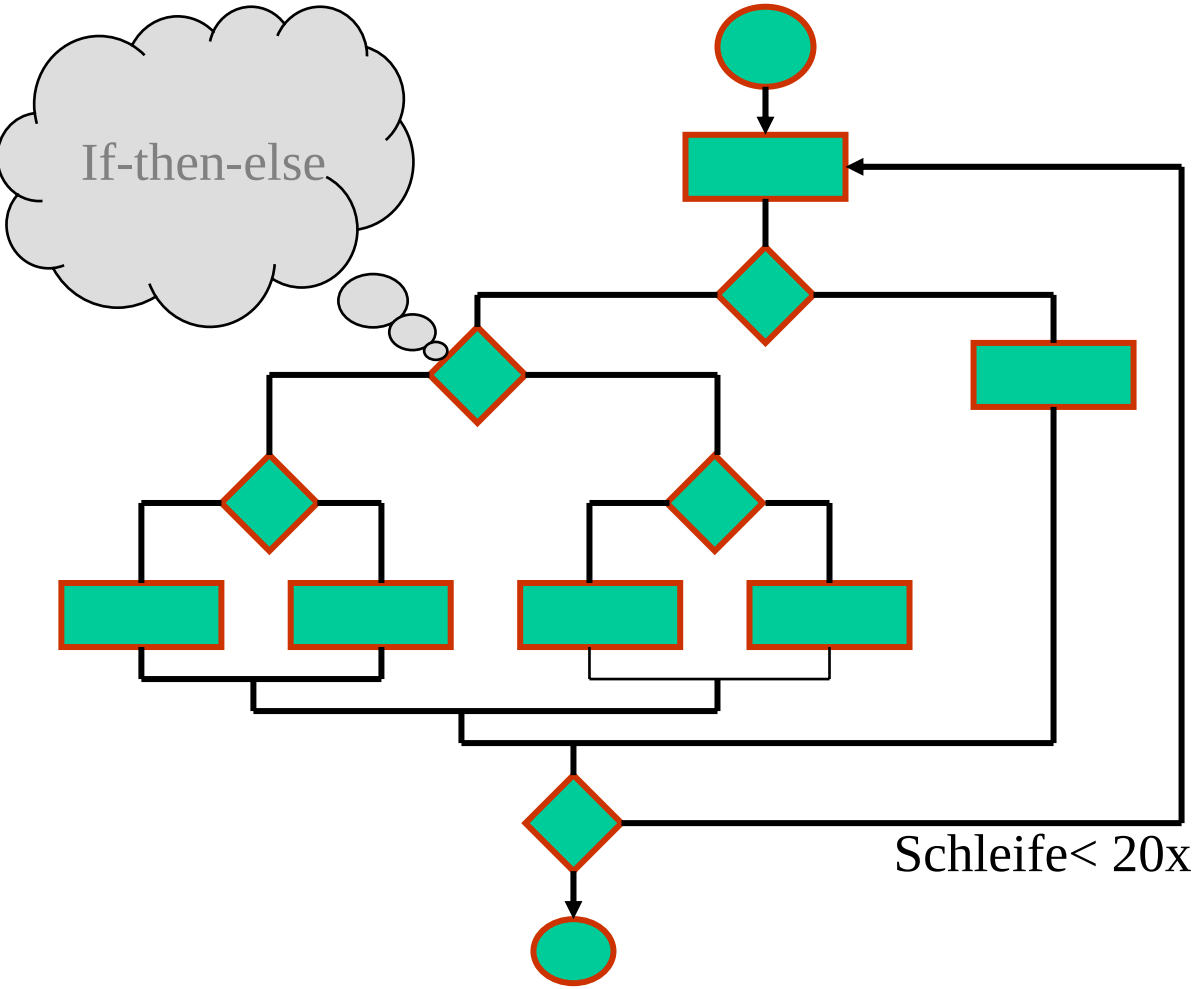
Wähle eine Testmenge, die alle Pfade vom Eingangsknoten bis zum Endknoten durchläuft:

- Nicht realistisch für while-Programme,
- Boundary-interior Pfadtest:
  Die Anzahl der Wiederholungen in Schleifen wird eingeschränkt.

Warum alle Pfade überdecken?

- Logische Fehler und inkorrekte Annahmen sind umgekehrt proportional zu der Wahrscheinlichkeit der Ausführung eines Pfads
- Entwickler glauben häufig, dass ein bestimmter Pfad nicht ausgeführt wird; die Realität verhält sich häufig anders.
- Tippfehler sind zufällig; sehr wahrscheinlich enthalten ungetestete Pfade Tippfehler.

Garantierte Überdeckung: Alle Pfade



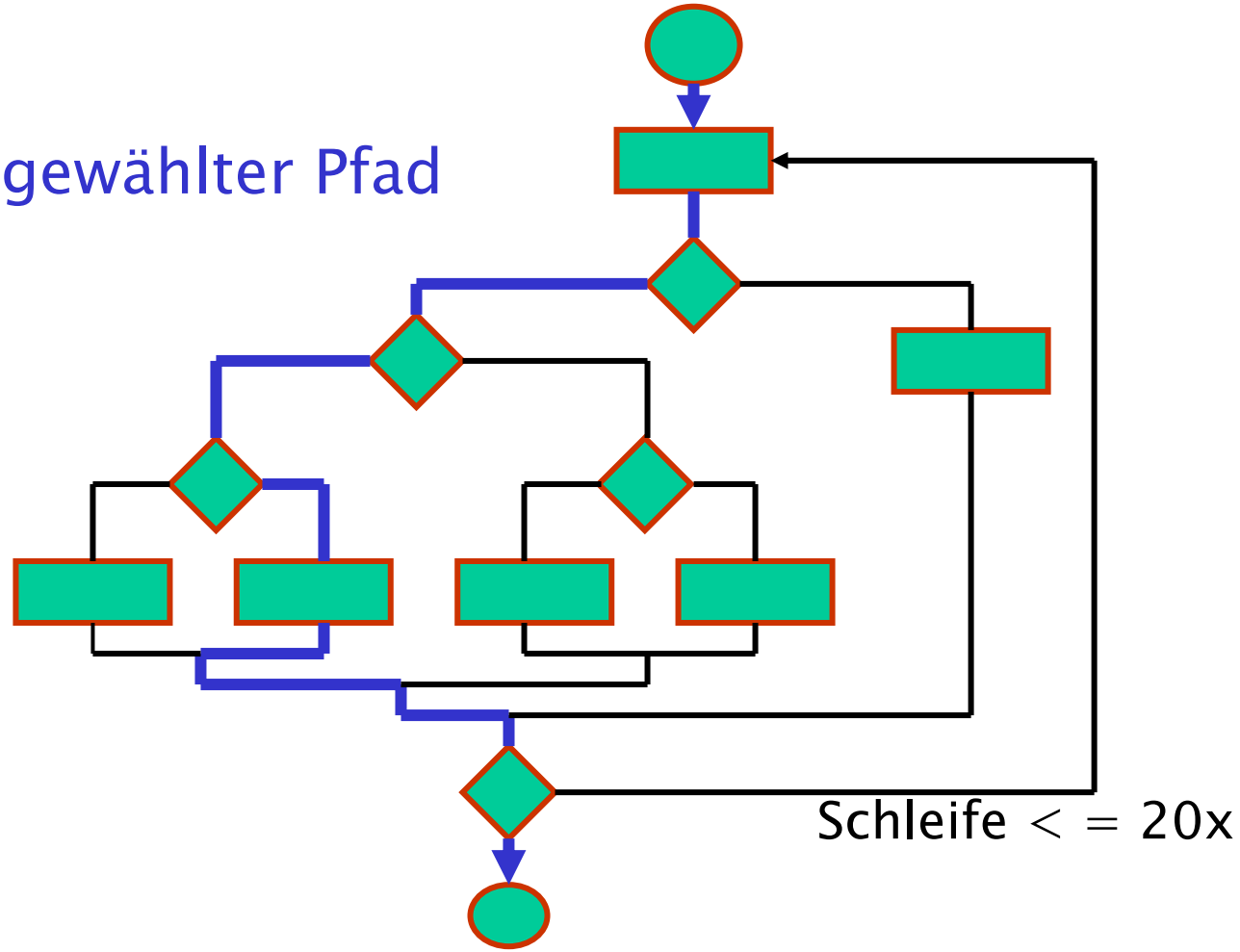
Aber:
Es gibt ca.
$5^{20}=10^{14}$
mögliche Pfade!

Wenn möglich, setze **Modellprüfung** ein – z,.B. zusammen mit Abstraktion.

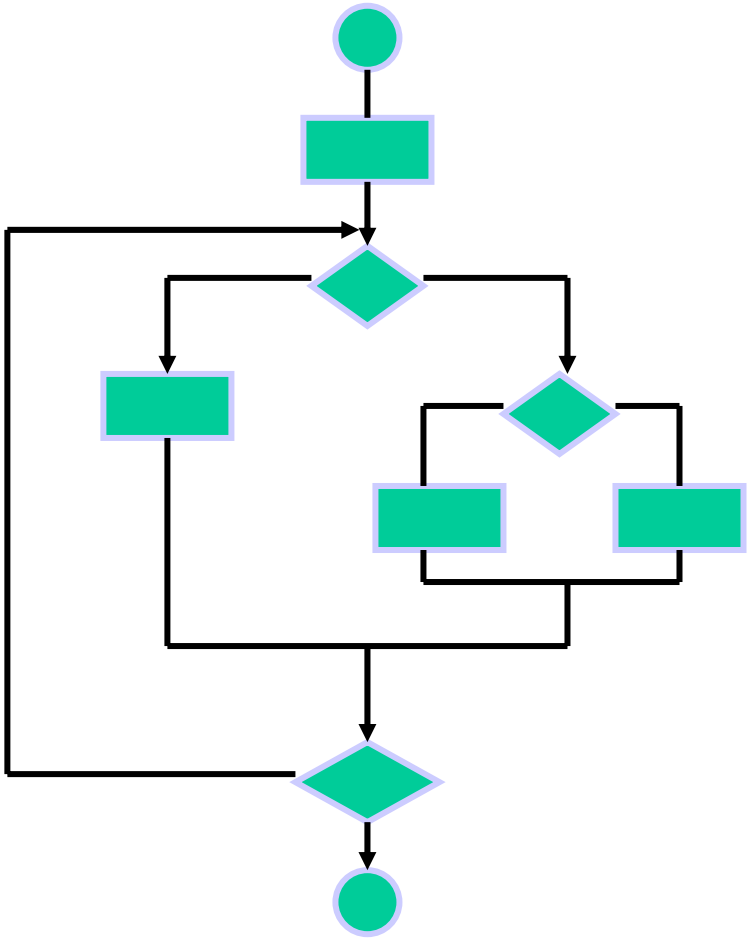
Oder wenn dies **nicht** erfolgreich:

Selektives Testen

Ein ausgewählter Pfad



Zyklomatische Komplexität: Ein Maß für Pfadüberdeckungstesten



Berechnung der zyklomatischen Komplexität V(G) :

Anzahl der Transitionen – Anzahl der Knoten +2

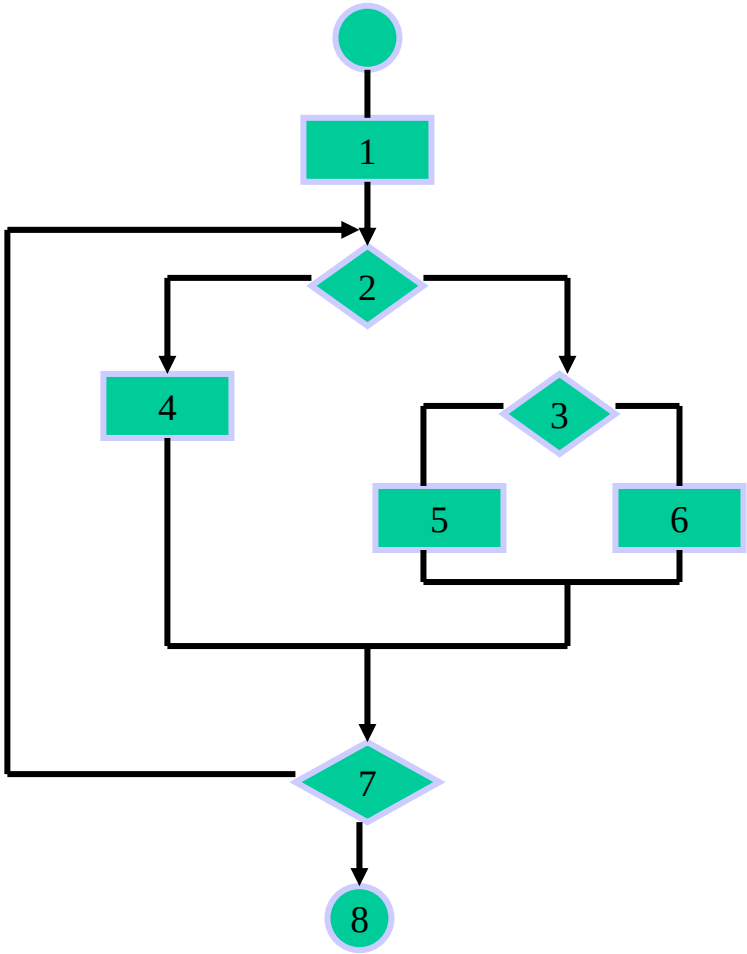
bzw. Anzahl der Bedingungen +1

In diesem Fall, V(G) = 4

V(G) gibt die Anzahl der linear unabhängigen Zyklen von G an (wobei Start- und Endknoten miteinander verbunden werden).

V(G) liefert eine obere Schranke für die Anzahl der Testfälle, die nötig sind, um alle Anweisungen zu überdecken.

Pfadüberdeckungstesten: Beispiel



Wg V(G) = 4 gibt es 4 Pfade:

Pfad 1: 1,2,3,6,7,8

Pfad 2: 1,2,3,5,7,8

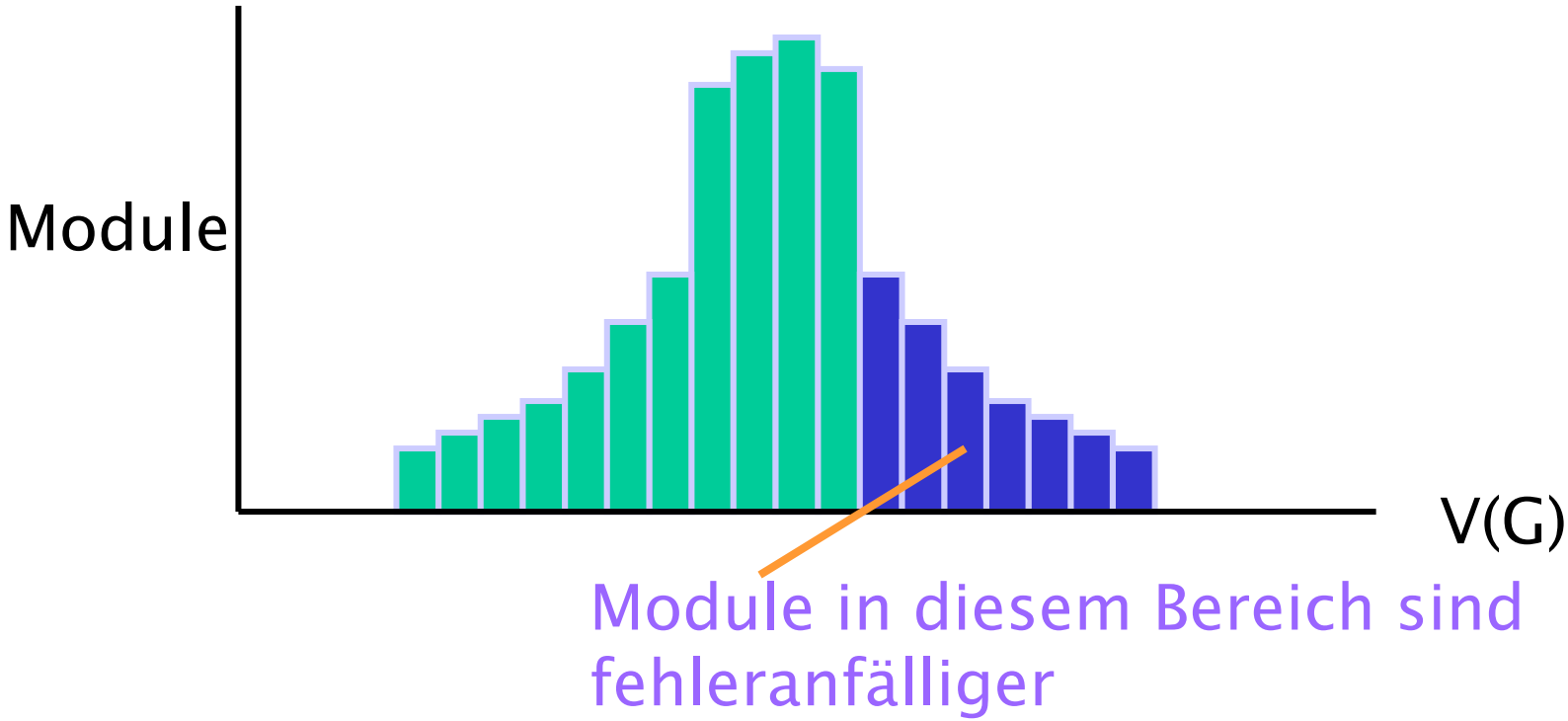
Pfad 3: 1,2,4,7,8

Pfad 4: 1,2,4,7,2,4…7,8

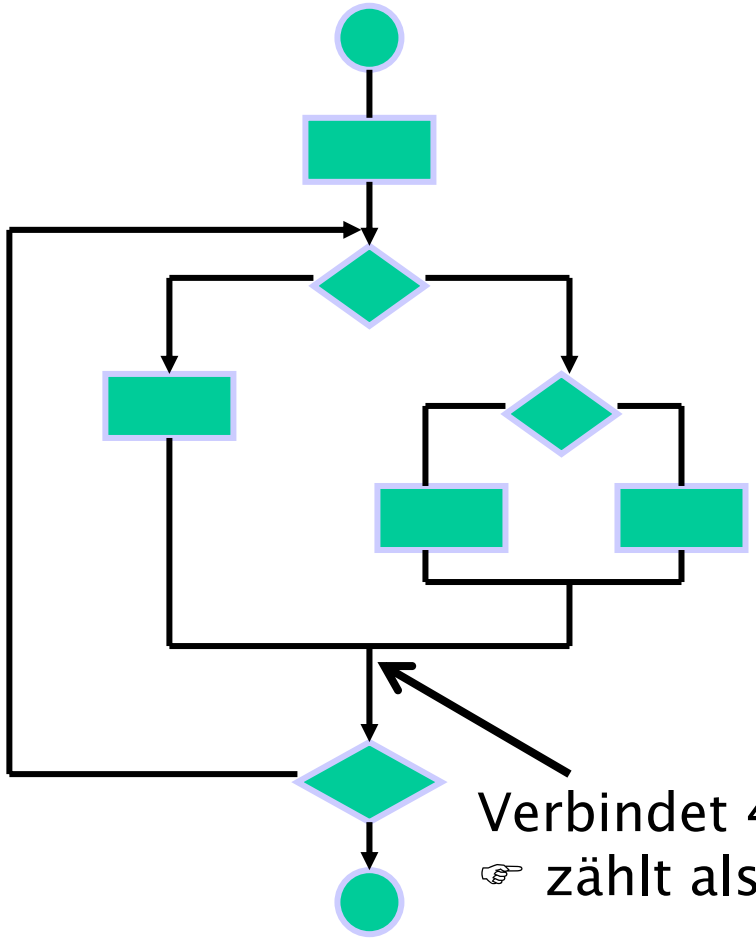
Entwickle daraus Testmengen durch Angabe von geeigneten Inputs.

Zyklomatische Komplexität

Eine Reihe von Industriestudien haben gezeigt, dass die Fehlerwahrscheinlichkeit bei großem V(G) ansteigt.



Pfadüberdeckungstesten: Bemerkungen



- Pfadüberdeckungstesten sollte bei kritischen Modulen angewendet werden.
- Kontrollflussgraphen sind nicht notwendig, aber hilfreich um die Pfade zu definieren.
- Jede Verbindung zwischen den Kästchen zählt als eine Transition.

Testen: Schritte im Prozess

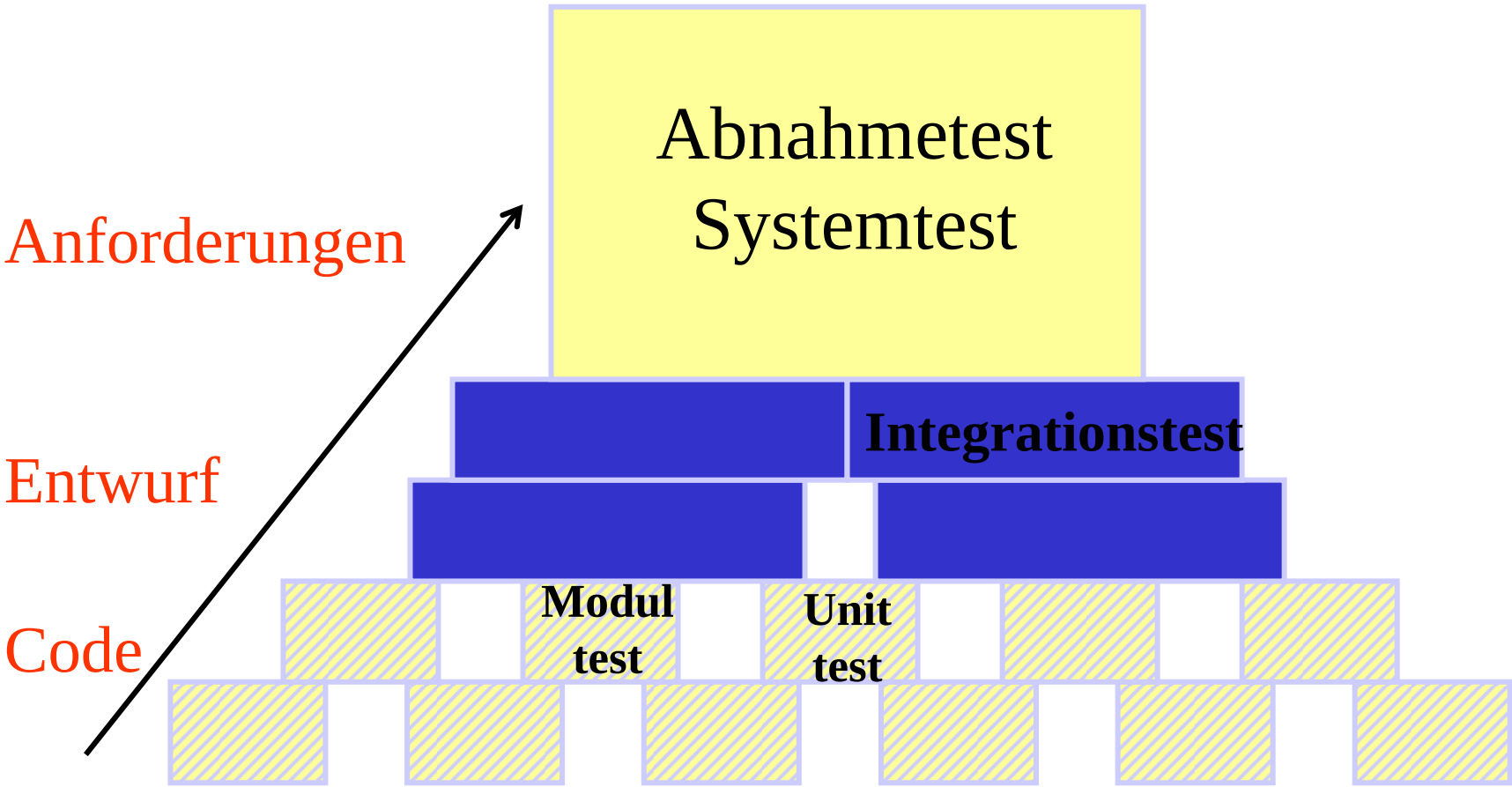
- Test: konstruktive und analytische Anteile
- Testen im Prozess:
  - Konstruktive Maßnahmen:
    - Anforderungsanalyse: Spez. von Abnahme-/ Belastungstest
    - Entwurf: Spez. von Modultest/Integrationstest, Instrumentierung
    - Umsetzung: Instrumentierung
    - Test: Testbettvorbereitung
  - Analytische Maßnahmen
    - Modultest
    - Integrationstest
    - Abnahmetest, Belastungstest

Arten von Tests

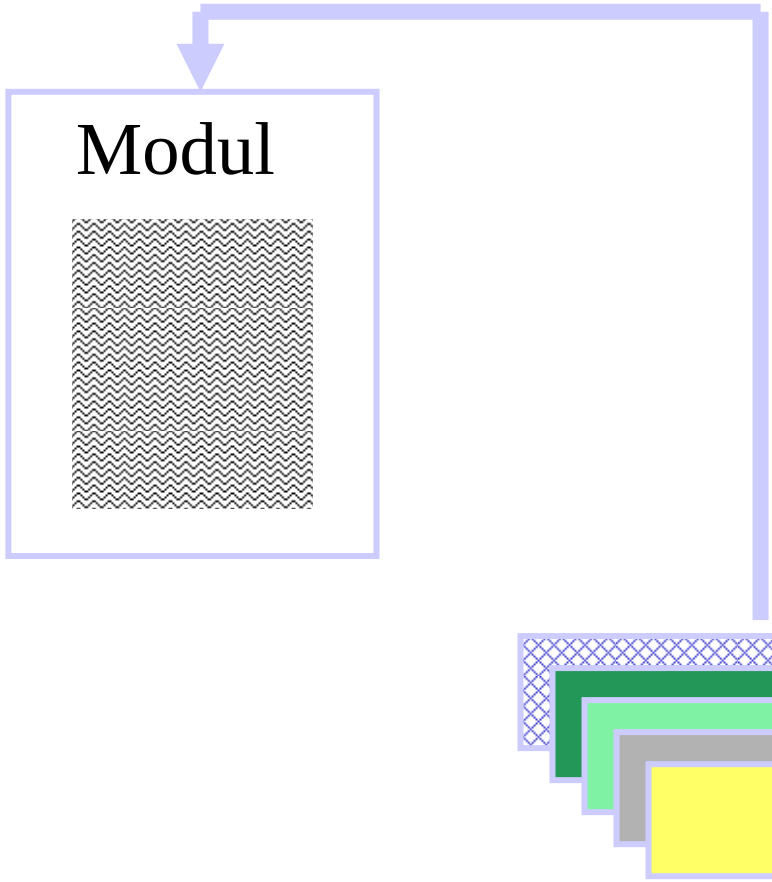
Man unterscheidet beim Testen u.a.

Unit-Test	Test der einzelnen Methoden einer Klasse (White- u. Blackbox-Test)
Modul-Test	Test einer Menge von Klassen mit einer bestimmten Aufgabe
Integrationstest	Test der Integration mehrerer Module
Systemtest	Test des Gesamtsystems (im Labor)
Abnahmetest	Test des Gesamtsystems (durch den Benutzer)
Feldtest	Test während des Einsatzes

Teststrategie: Hierarchie

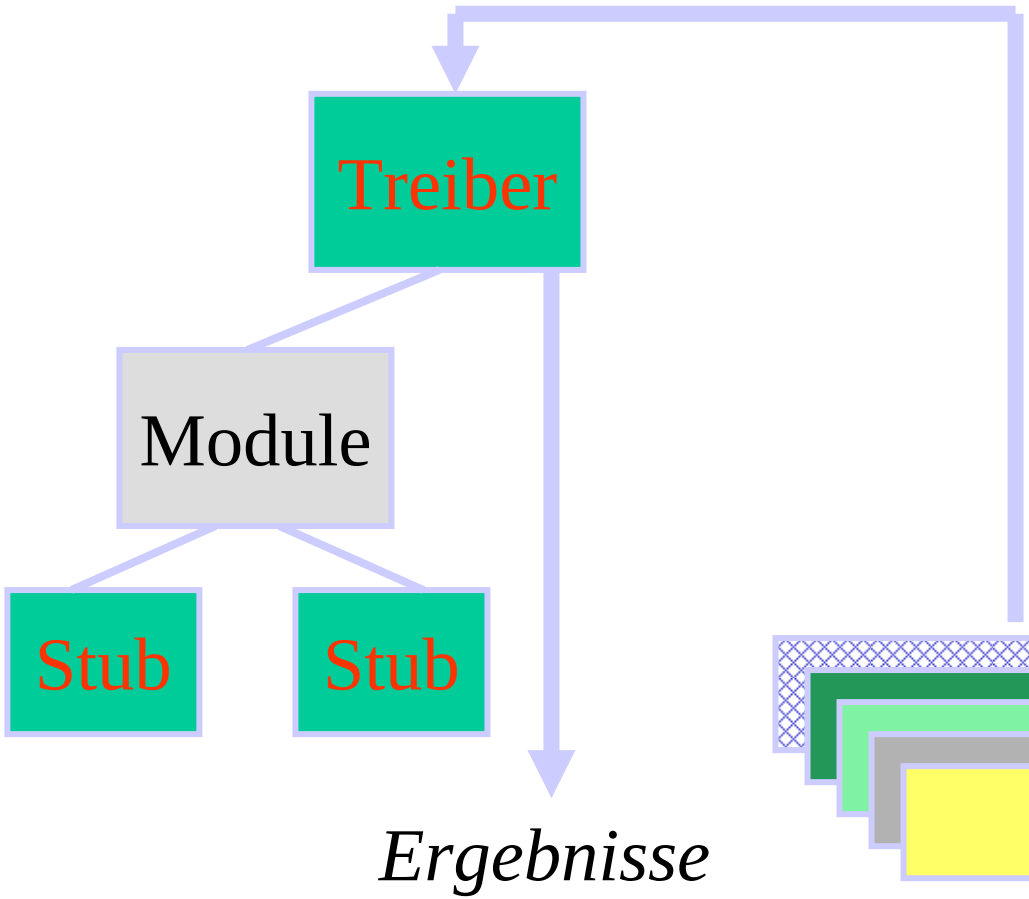


Modultest/Unittest



- Schnittstelle
- lokale Datenstrukturen
- Grenzwerte
- unabhängige Pfade
- fehlerbehandelnde Pfade

Modultest-Umgebung



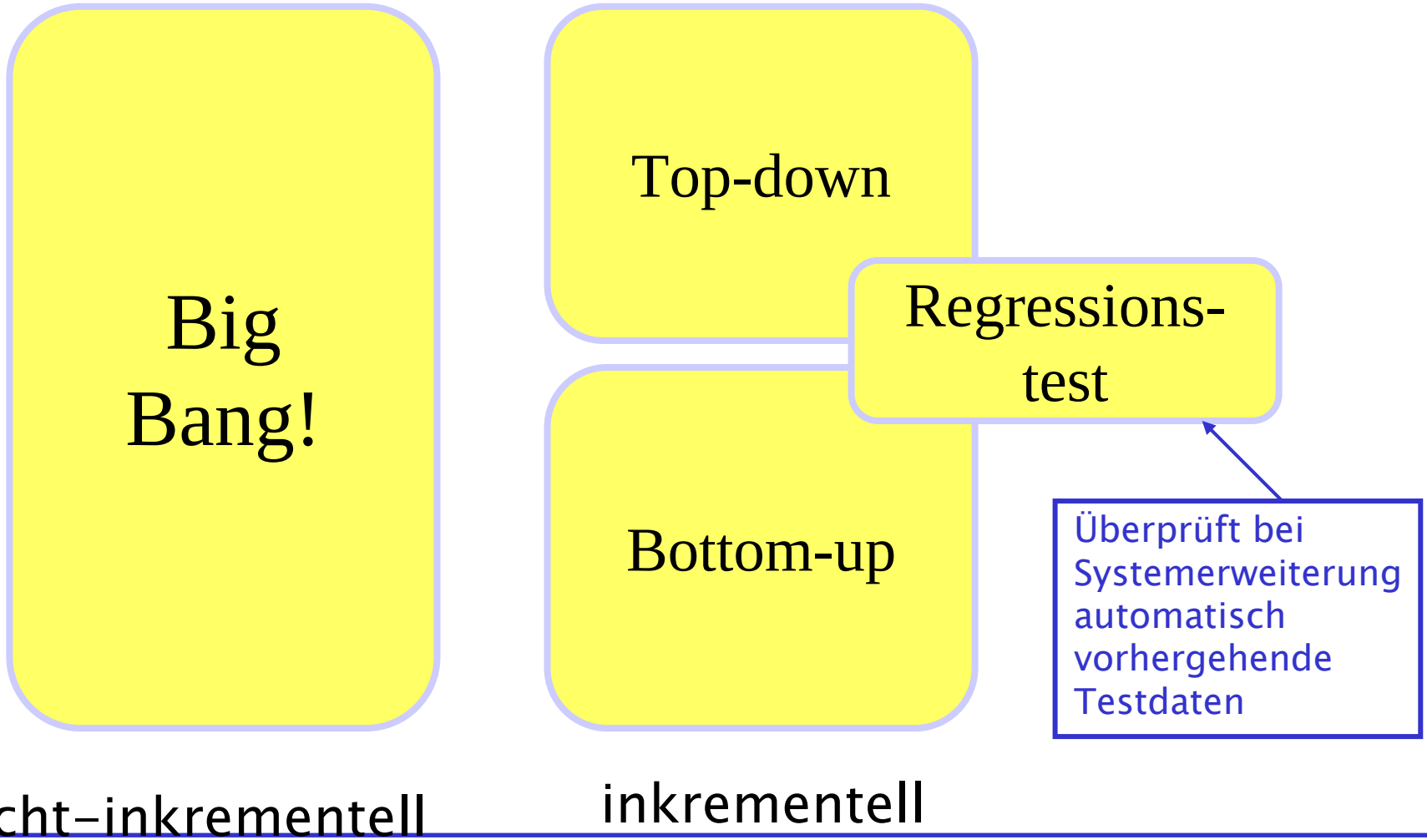
- Schnittstelle
- lokale Datenstrukturen
- Grenzwerte
- unabhängige Pfade
- Fehlerbehandelnde Pfade

Warum Integrationstest?

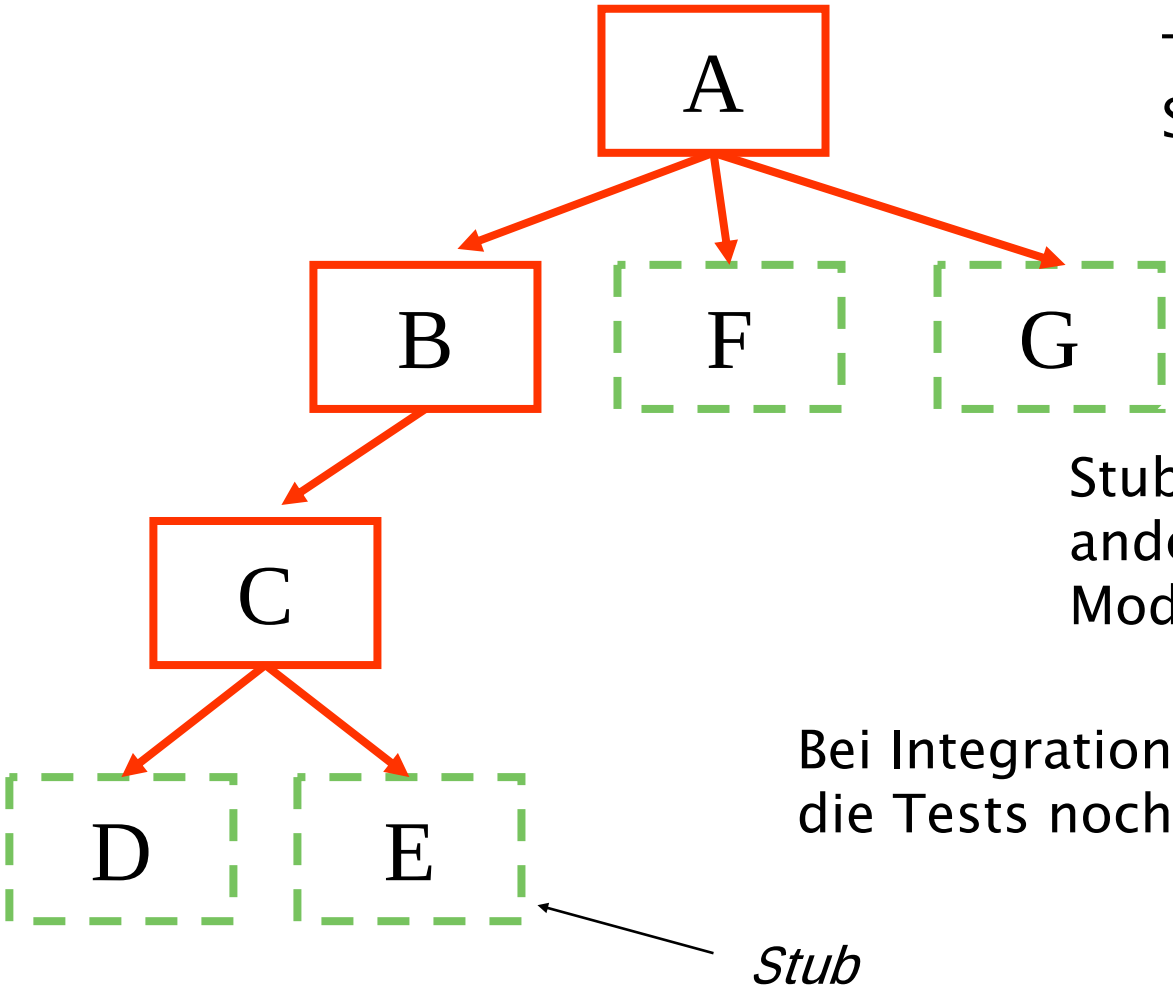
- **Wenn alle Module korrekt sind,**
  warum sollen sie dann nicht zusammenpassen?
- **Was ist mit**
  - undokumentierten Seiteneffekten,
  - unterschiedlicher Interpretation von Operationen,
  - Betriebssystemfeatures , wie z.B. Speicherverbrauch,
  - Einführung von Nebenläufigkeit (race conditions),
  - Verzögerungen der Kommunikation?

☞ **Einheiten/Module zu integrieren erfordert**
Schnittstellen und deren Überprüfung

Integrationstest: Vorgehensweisen



Top-down-Integration

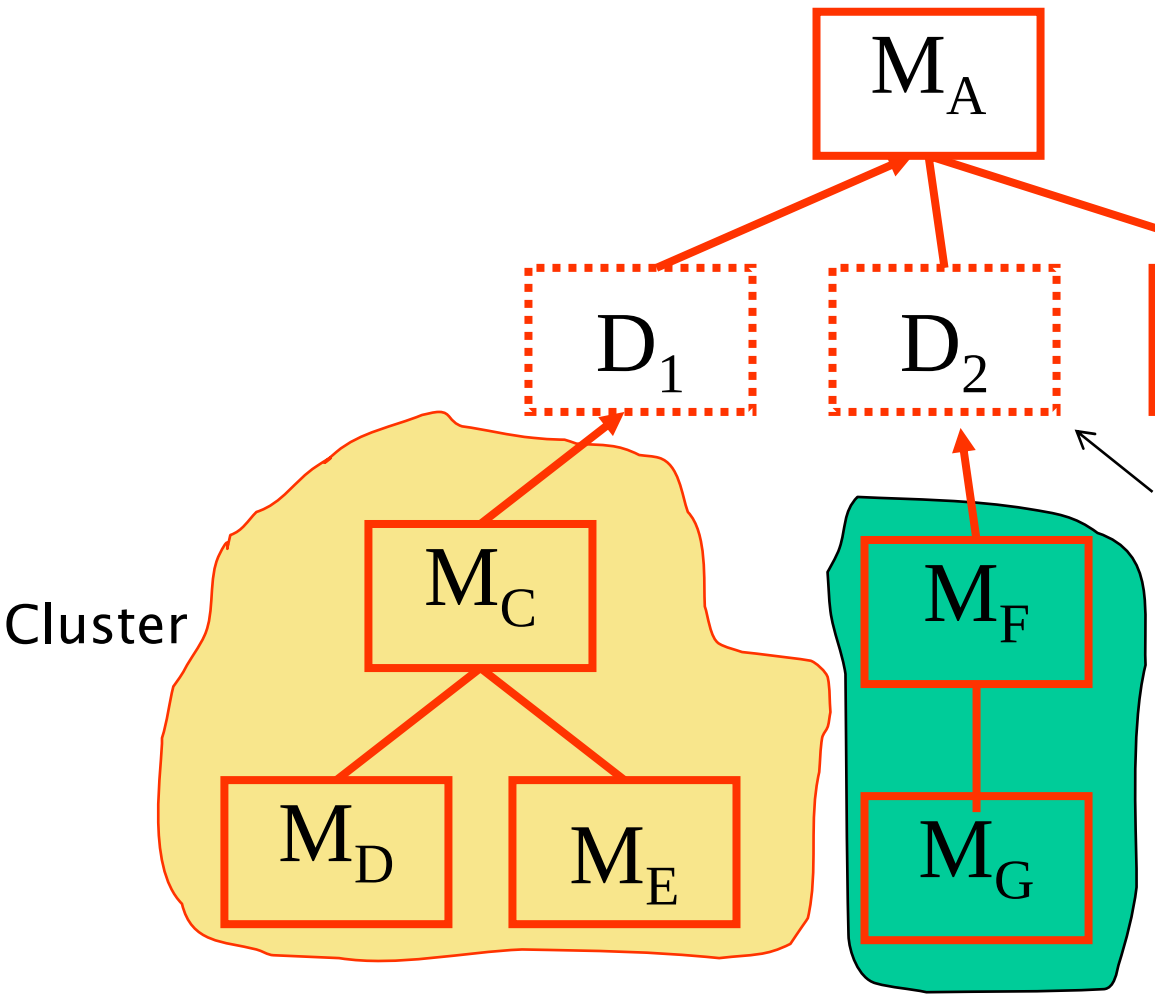


Top-Modul wird mit Stubs getestet.

Stubs werden einer nach dem anderen durch vollständigen Modul ersetzt.

Bei Integration neuer Module werden die Tests noch einmal ausgeführt.

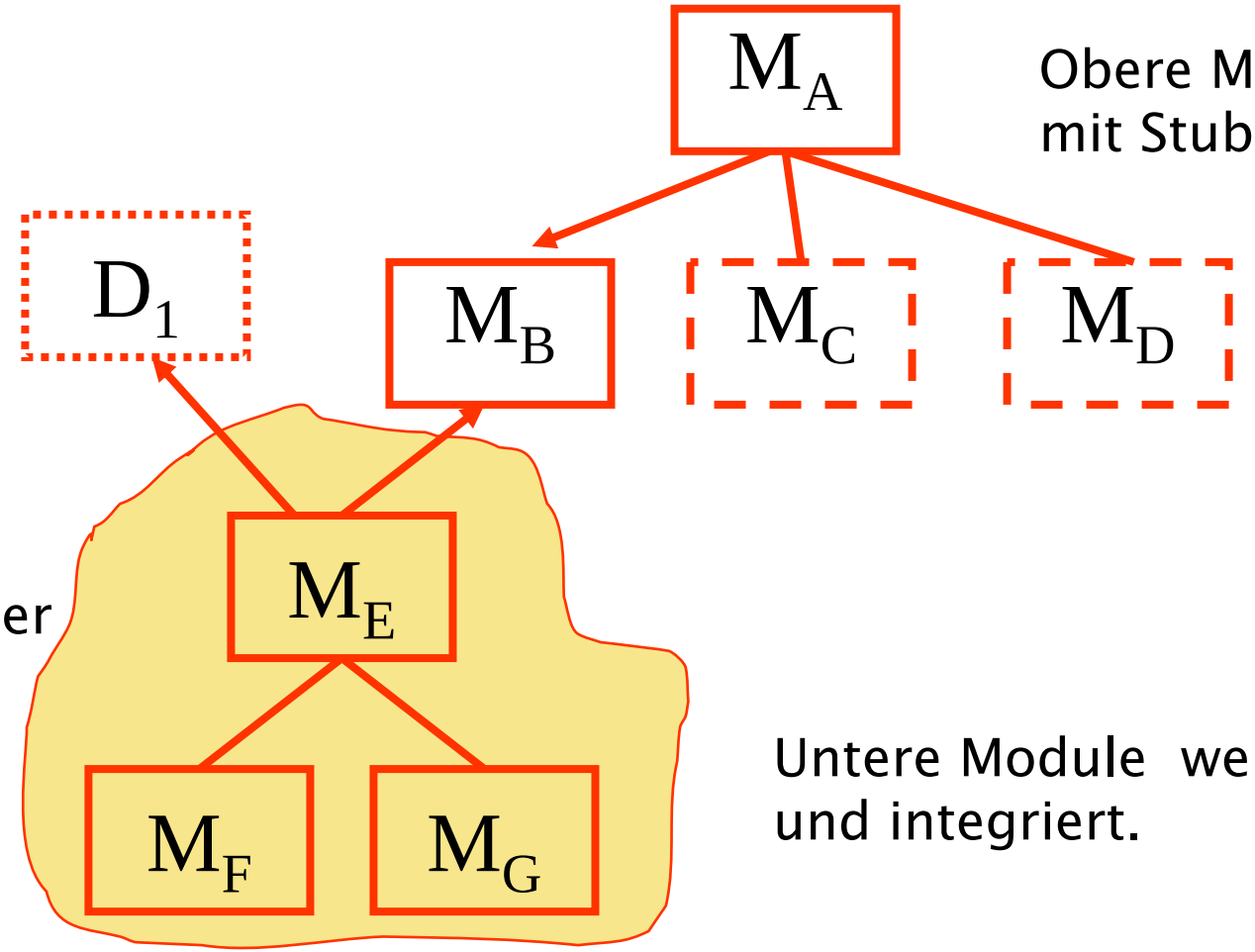
Bottom-up-Integration



Die Treiber werden einer nach dem anderen ersetzt.

Module werden gruppiert und integriert.

Sandwich- Test



Cluster

Obere Module werden mit Stubs getestet.

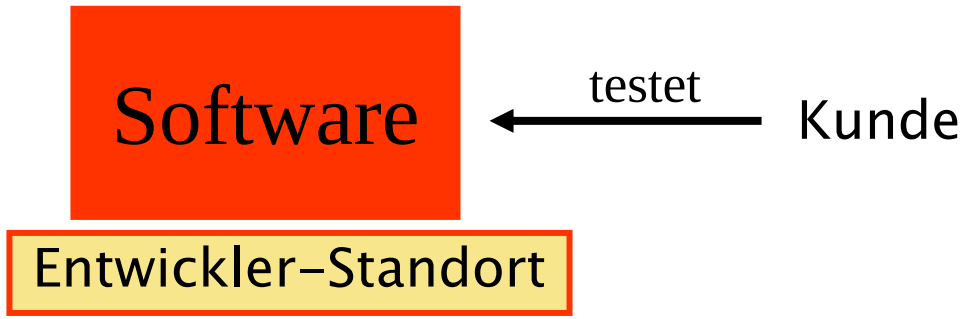
Untere Module werden gruppiert und integriert.

Höhere Tests

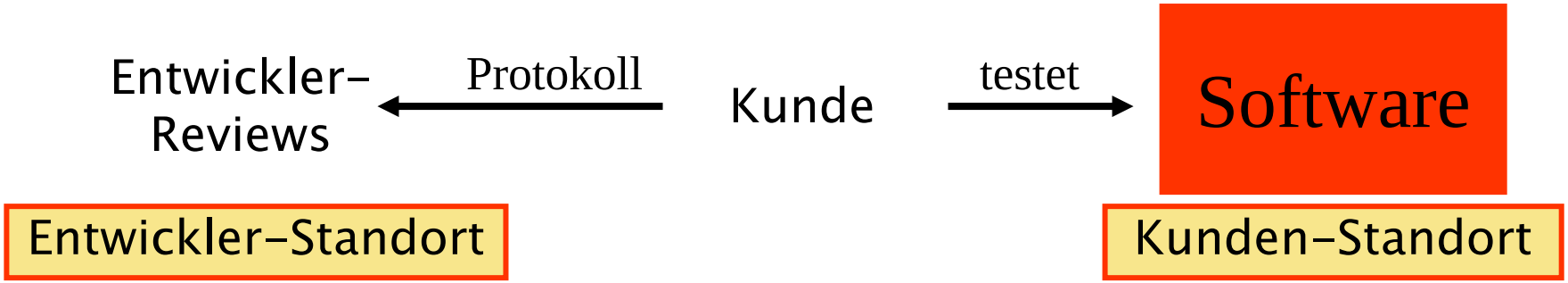
- **Abnahmetest (Validierungstest)**
  - Alpha- und Beta-Test
- **Systemtest**
- **Andere spezialisierte Testarten**
  - Performanztest
  - Sicherheitstest
  - ...

Alpha & Beta-Test

Alpha-Test



Beta-Test



Konstruktive Verfahren

- **Ansatz:**
  - A-priori-Absicherung der Produktqualität
  - Vorgabe von Konstruktionstechniken
  - Präventiv (keine Behebung erforderlich)
- **Techniken:**
  - Methoden (Beispiel: RUP)
  - Sprachen (Beispiel: Java statt C++)
  - Richtlinien (Beispiel: NASA Coding Standard)
  - Checklisten, Vorlagen

Konstruktive Verfahren: Methoden

- **Ziel: Systematische Vorgehensweise**
- **Technik: Vorgabe von Zwischenprodukten**
  - Vorgaben von Modellen (z.B. Objektorientierte Vorgabe von Einzelschritten, etwa OOA mit fachl. Klassenmodell, Use Case-Modellierung)
  - Vorgabe von Erstellungsmittel (z.B. Klassendiagramm, Objektdiagramm, Use Case Diagramm)
- **Ansatz:**
  - RUP
- **Zusätzlich:**
  - Änderbarkeit
  - Werkzeugunterstützung

Konstruktive Verfahren: Richtlinien

Richtlinien

- **Ziel: Produkteigenschaften a-priori festlegen**
- **Technik:**
  - Vorgaben von Checklisten, Schablonen
  - Überprüfung der Richtlinien
- **Ansätze:**
  - Analyse: Strukturierung (z.B. SCR-Tabellen)
  - Design:
    - Architektur (z.B. Pattern)
    - Beschreibungen (z.B. Automaten)
  - Umsetzung: Code (z.B. NASA Coding Standard)
- **Zusätzlich: Änderbarkeit, Übertragbarkeit**

Zusammenfassung (1)

- **Qualität** ist die Gesamtheit von Eigenschaften und Merkmalen eines Produkts oder einer Tätigkeit, die sich auf deren Eignung zur Erfüllung gegebener Erfordernisse bezieht (ISO 8402)
  - In Bezug auf Software sind dies: **Funktionalität, Zuverlässigkeit, Benutzbarkeit, Effizienz, Änderbarkeit, Übertragbarkeit**
- **Qualitätsmanagement** umfasst alle Tätigkeiten, um die Qualität von **Prozessen und Produkten** sicherzustellen, inkl.
  - **Qualitätspolitik, Qualitätsplanung**,
  - **Qualitätslenkung (konstruktive Maßnahmen** wie Methoden, Sprachen, Werkzeuge),
  - **Qualitätssicherung (analytische Maßnahmen**, wie statische Analyse, Verifikation und Testen) und
  - **Qualitätsverbesserung**
- **Qualitätsmaßnahmen** im Projekt umfassen
  - **Reviews, Tests, Durchstich,**
  - **Aufbau einer insgesamt gut nutzbaren Entwicklungsumgebung,**
  - **Ausbildungsmaßnahmen für Mitarbeiter, Verbesserung der Kommunikation,**
  - **Guidelines, Richtlinien, Organisation der Projektablage**
- Ein **Audit** ist eine intensive Untersuchung eines Projekts, bei der in der Regel durch externe Auditoren **formale Kriterien** (Timesheets, Ausgaben, …) als auch **Projektinhalte und Vorgehensweise** geprüft werden.

Zusammenfassung (2)

- Beim **Testen** unterscheidet man zwischen Unit-Test/ Modul-Test, Integrationstest, Systemtest, Abnahmetest und Feldtest.
- Das **Black-Box-Testen** geht von der **Spezifikation** aus und die Interna des Testobjekts sind nicht bekannt. Das **White-Box-Testen** ergänzt das Black-Box-Testen durch systematisches Explorieren der **Struktur des Testobjekts**.
- Wichtige Methoden des Black-Box-Testens sind
  - die **Äquivalenzklassenmethode,**
  - die **Methode der Grenzwertanalyse und**
  - der **Test von Zustandsautomaten.**
- Wichtige Techniken des White-Box-Testens sind **kontrollfluss-orientierte Verfahren** wie etwa
  - Anweisungsüberdeckungsverfahren,
  - Kantenüberdeckungsverfahren,
  - Bedingungsüberdeckungsverfahren,
  - **Pfadüberdeckungsverfahren** sowie datenflussorientierte Verfahren.