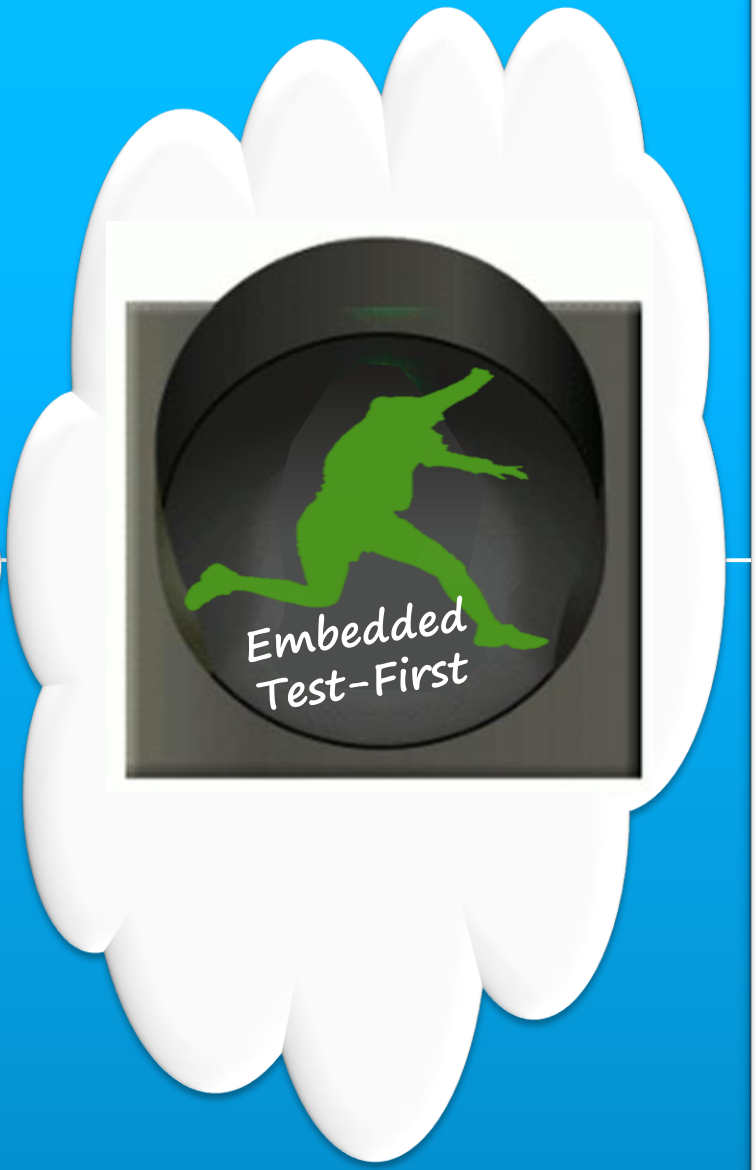
Test First =
Erst Testen, dann Denken?

Test-Driven Development
von Embedded Systemen

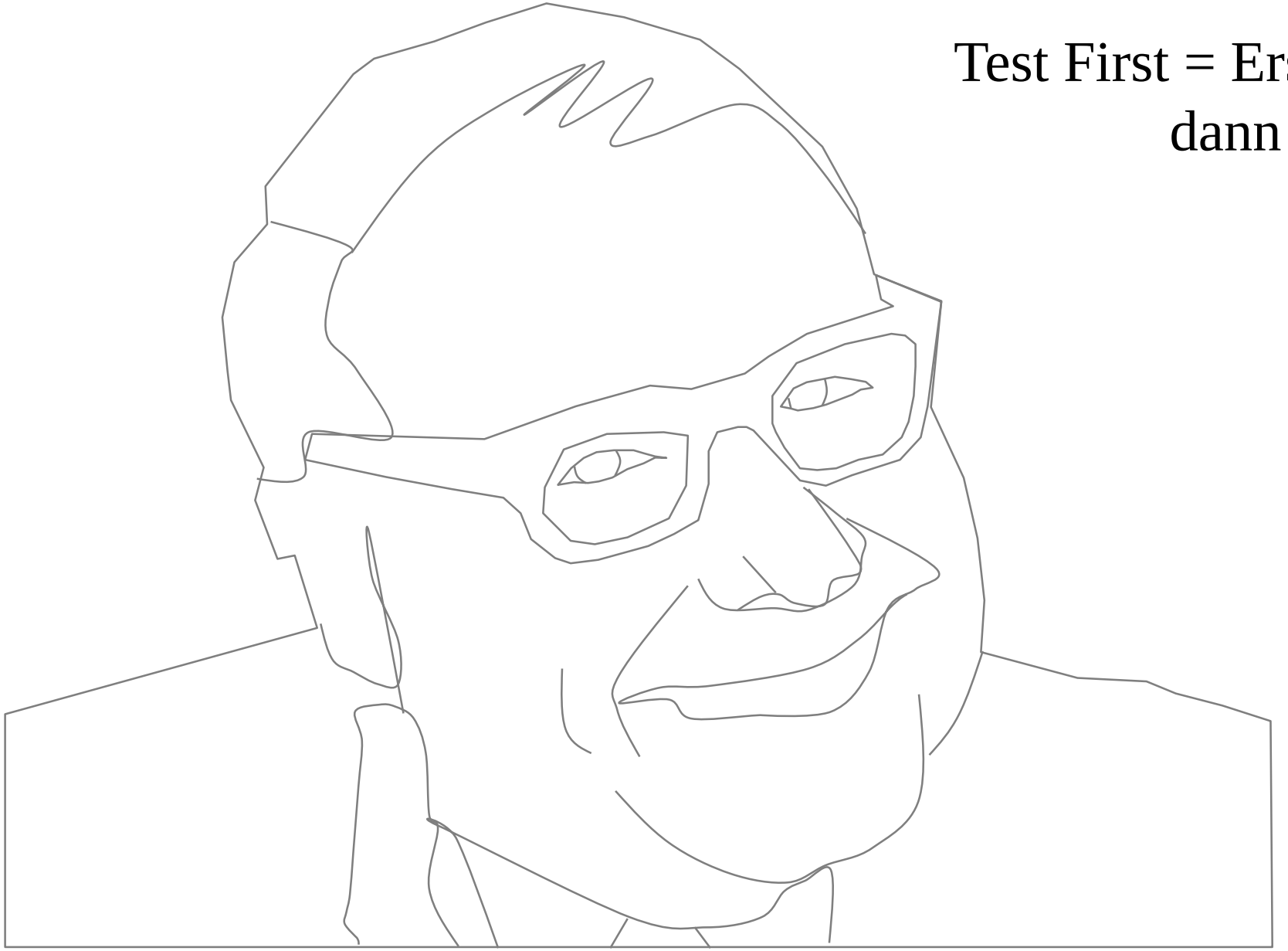


Dipl.-Ing. Univ.
Remo Markgraf

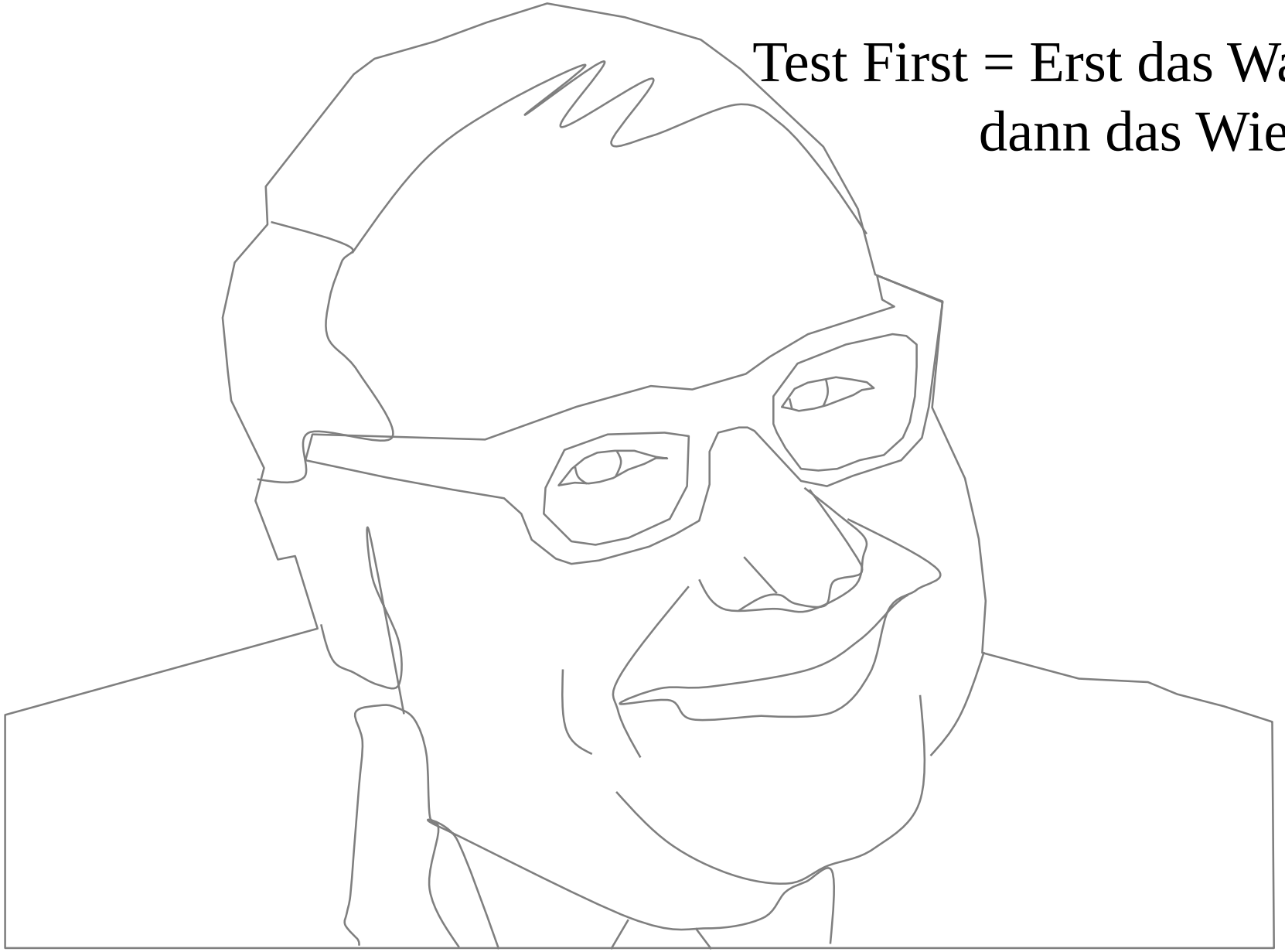
MicroConsult GmbH

Sie dürfen nicht alles glauben,
was Sie denken!

Heinz Erhardt

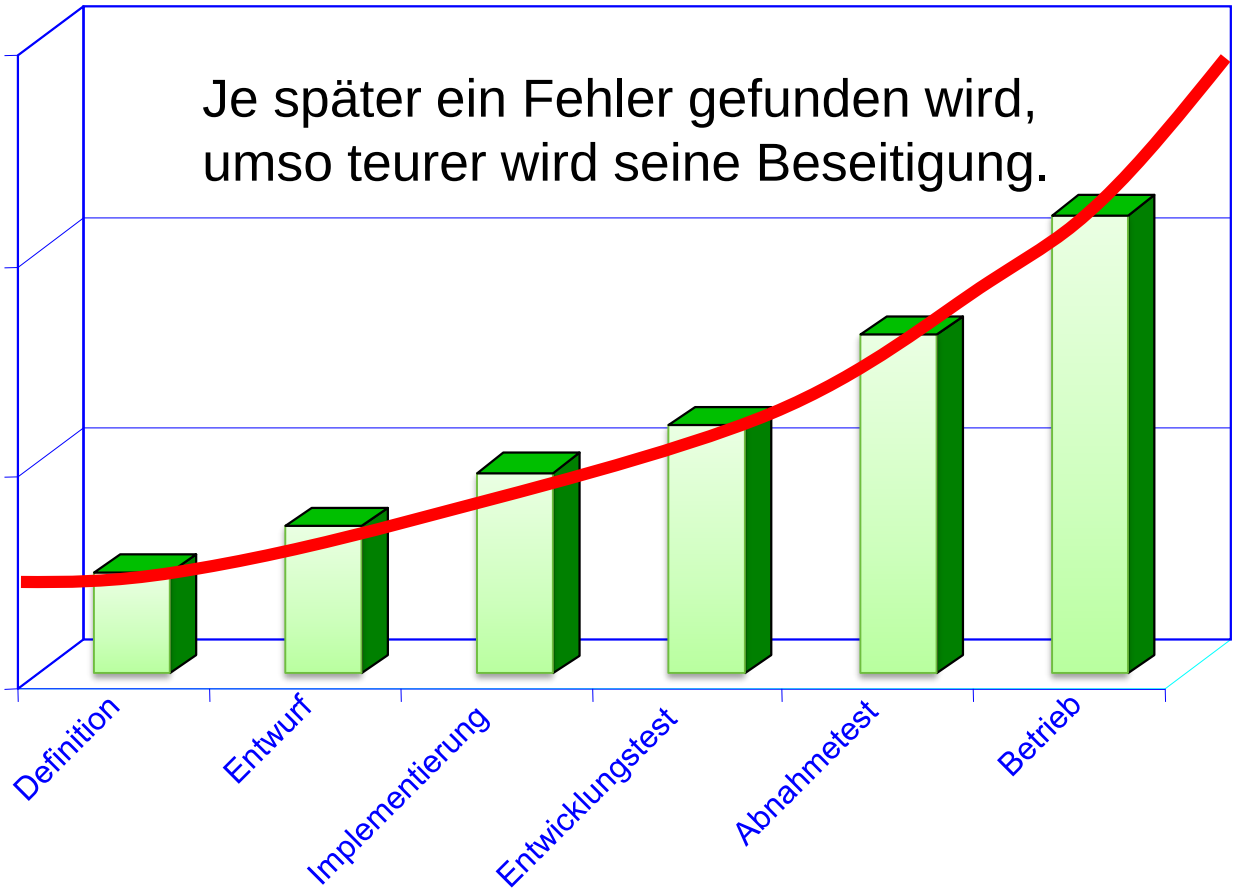


Test First = Erst Testen, dann Denken?



Test First = Erst das Was Testen,
dann das Wie Denken!

Gründe für den Test-First-Ansatz



Fehler vermeiden bzw. früh entdecken und beseitigen

Quelle: H. Balzert, 1998; Boehm, 1976

Test-First-Ansatz

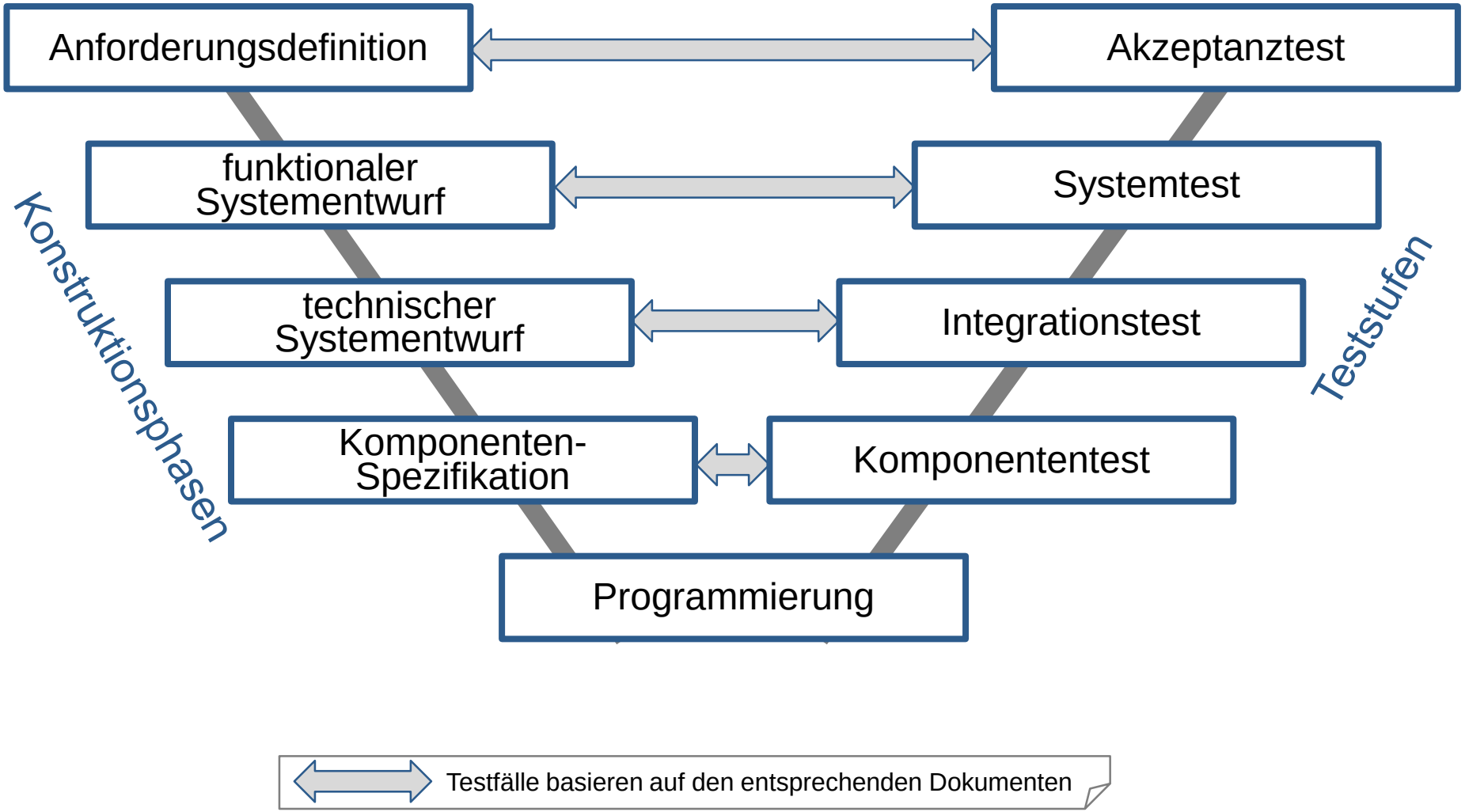
Ziel:

Fehler so früh wie möglich aufdecken

Umsetzung:

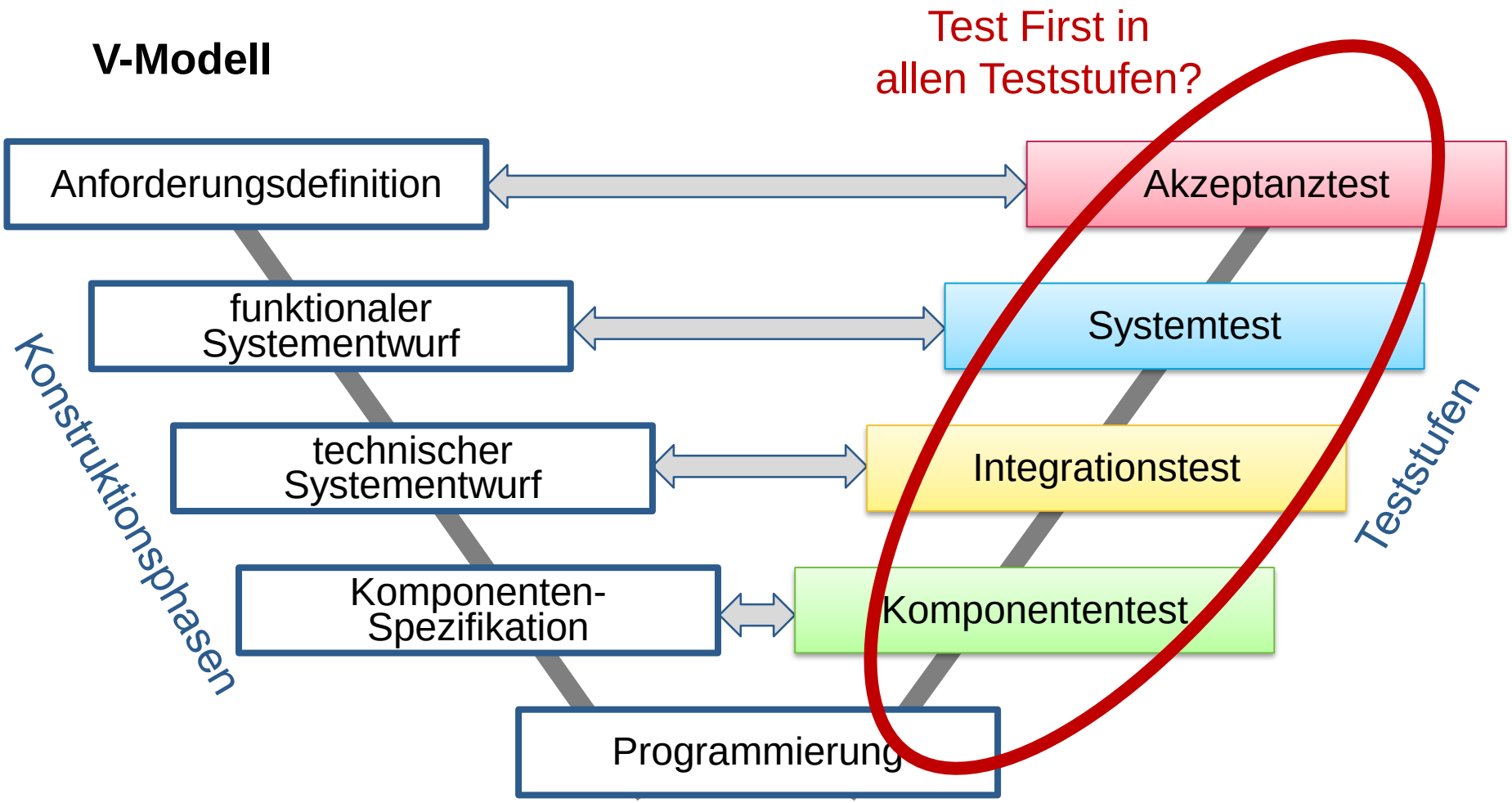
- Tests erstellen bevor der jeweilige Testgegenstand existiert
- Tests möglichst früh ausführen
- Test-First lässt sich in allen Teststufen anwenden

V-Modell



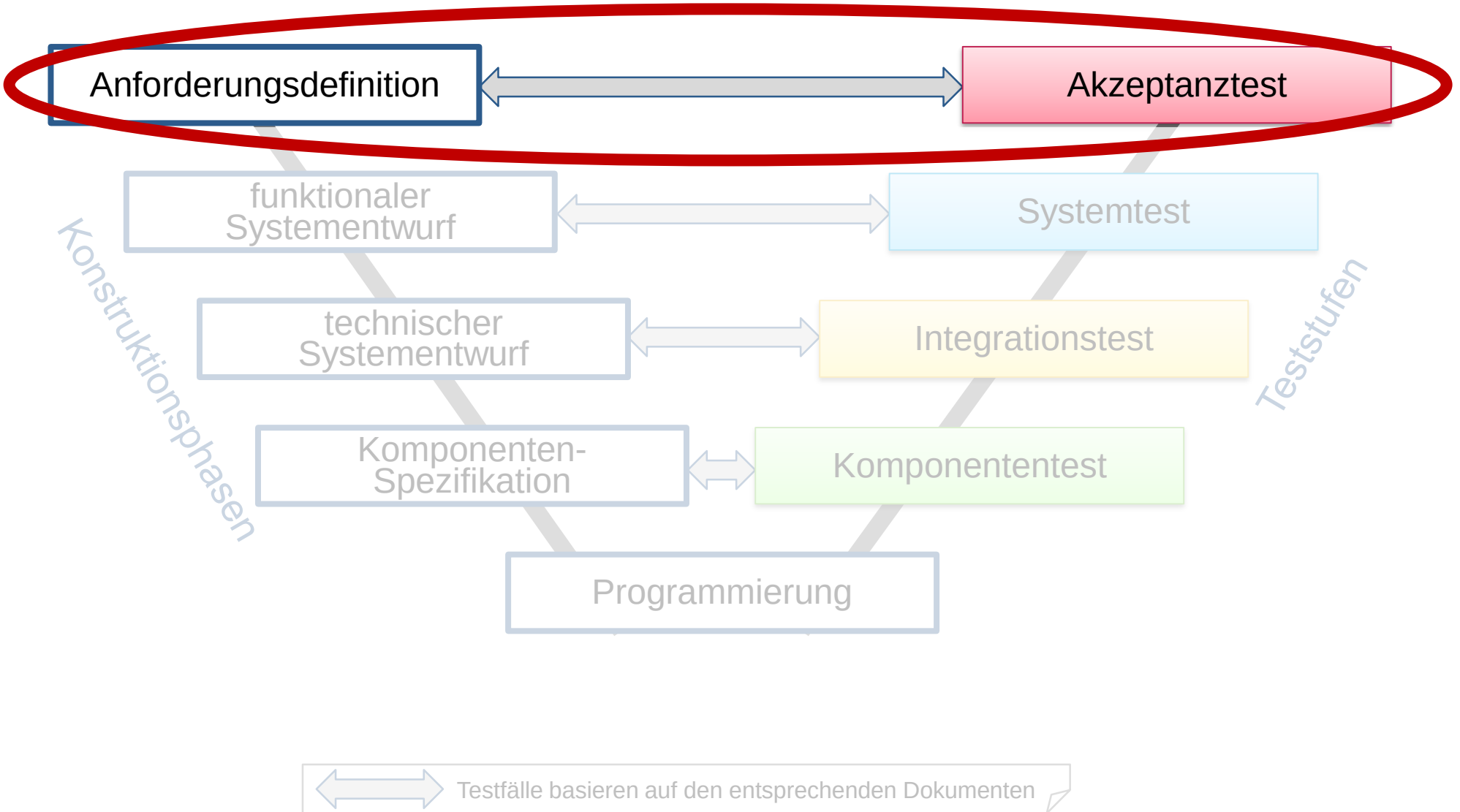
Quelle: V-Modell XT Release 2.3 http://www.v-modell-xt.de/

V-Modell

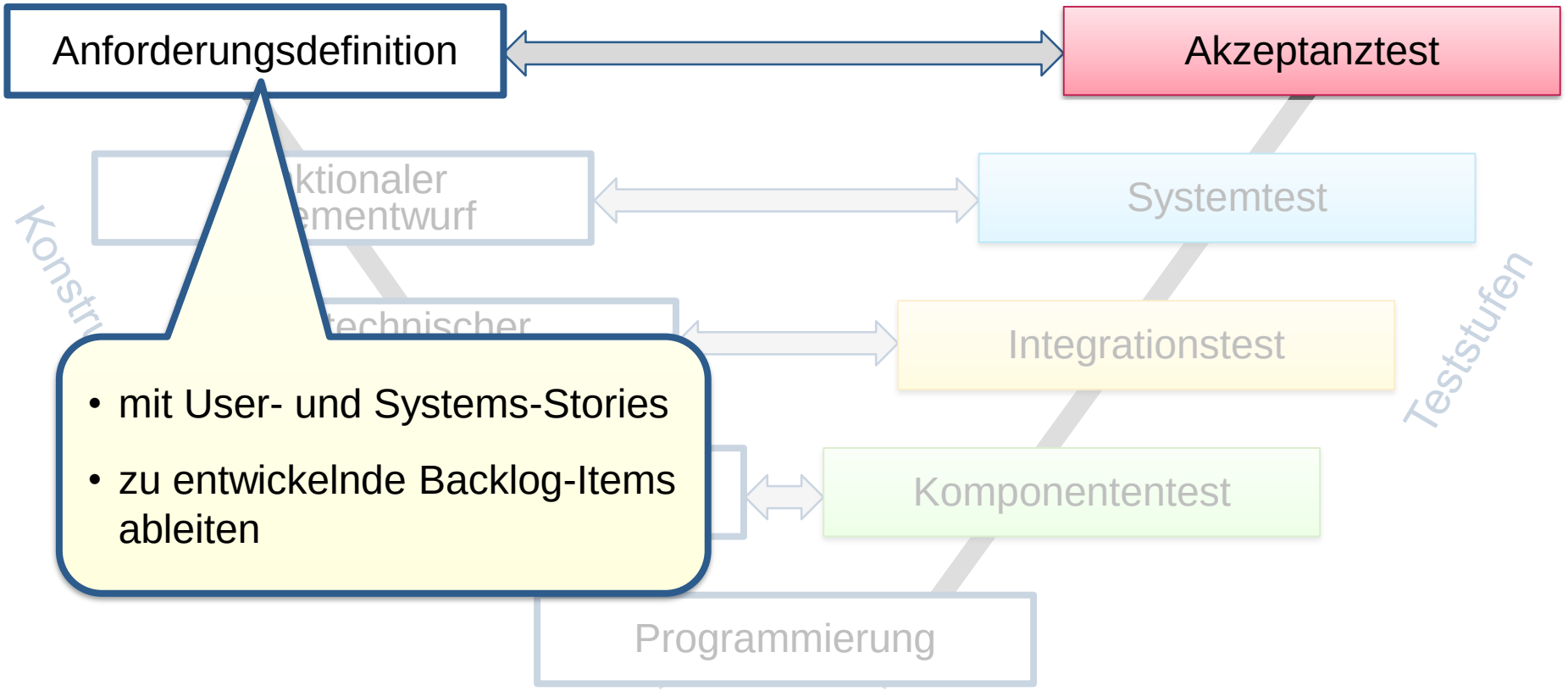


Quelle: V-Modell XT Release 2.3 http://www.v-modell-xt.de/

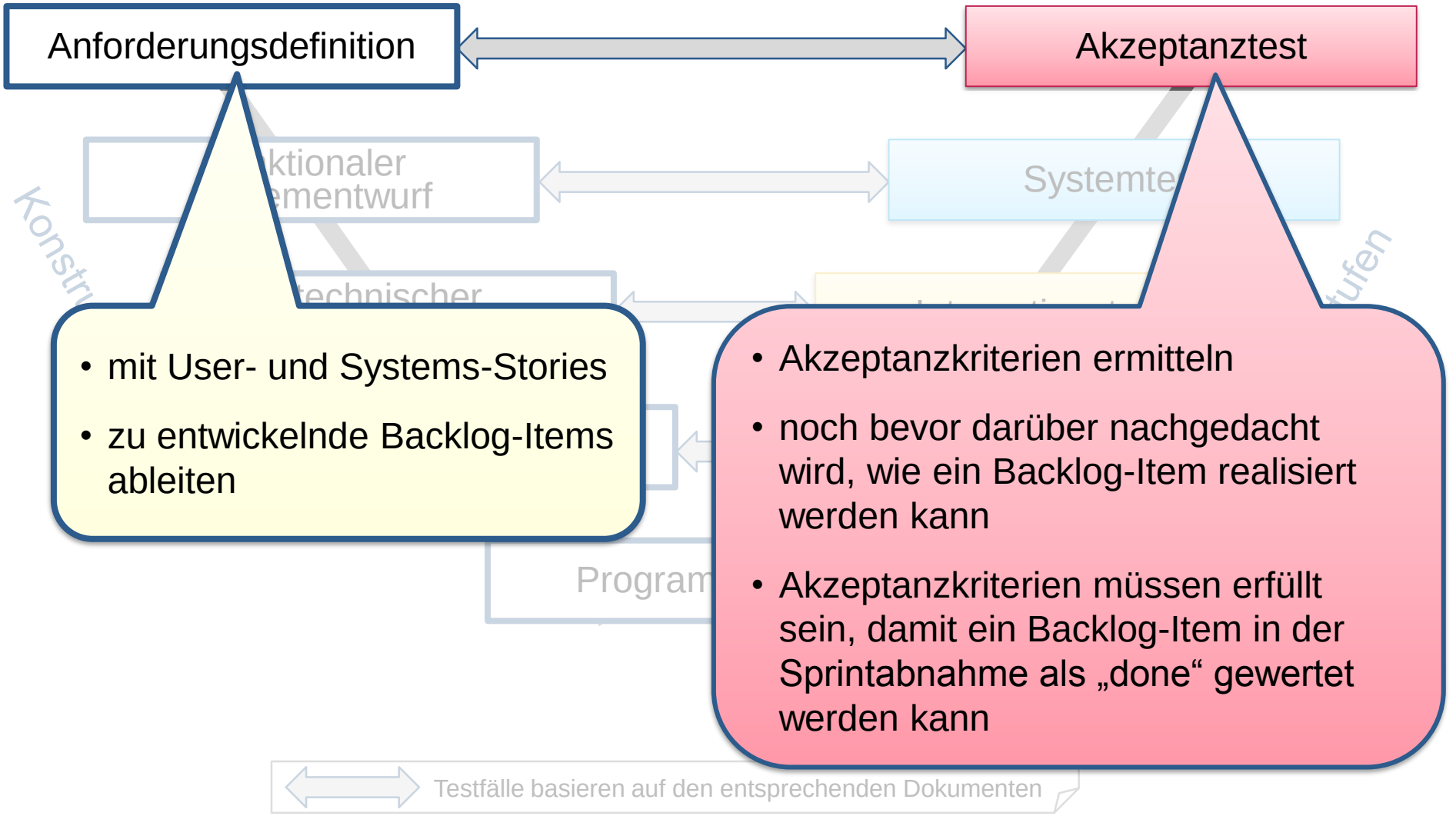
Test-First im Akzeptanztest



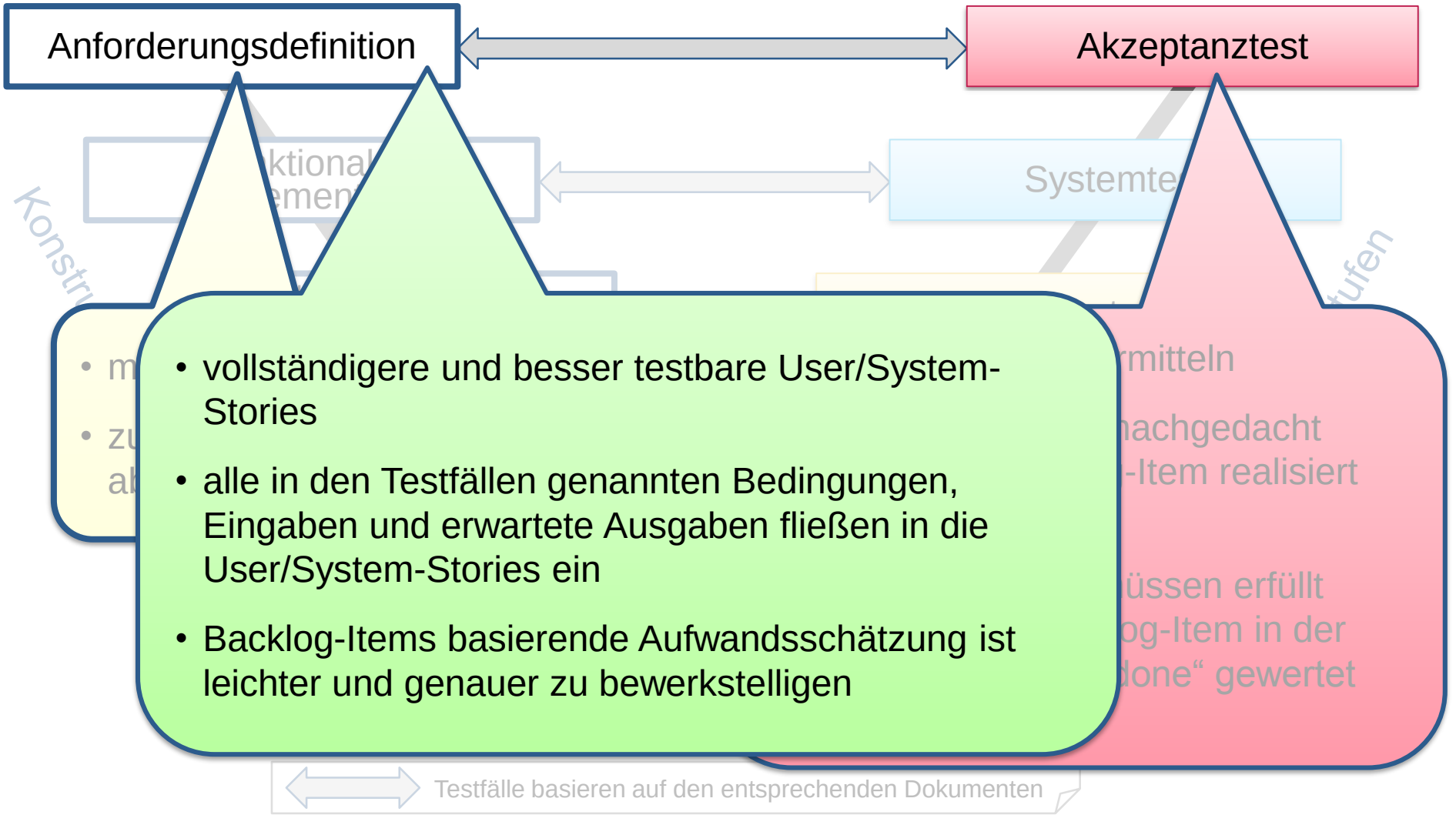
Test-First im Akzeptanztest



Test-First im Akzeptanztest

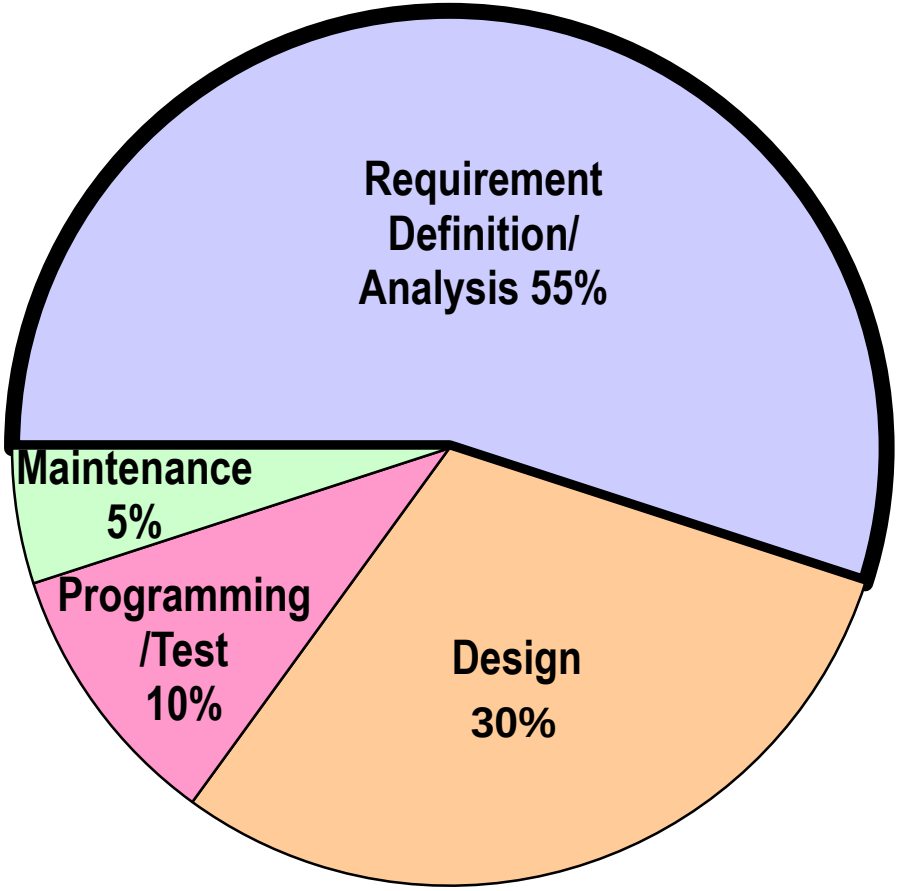


Test-First im Akzeptanztest

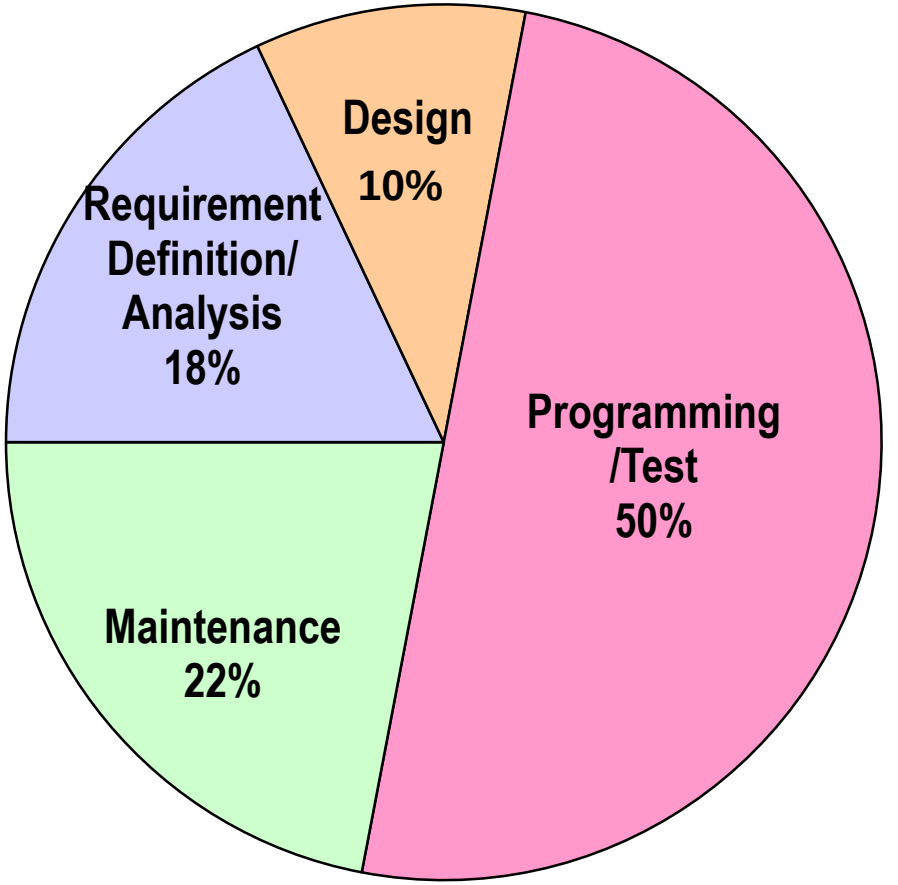


- vollständigere und besser testbare User/System-Stories
- alle in den Testfällen genannten Bedingungen, Eingaben und erwartete Ausgaben fließen in die User/System-Stories ein
- Backlog-Items basierende Aufwandsschätzung ist leichter und genauer zu bewerkstelligen

Errors Introduced

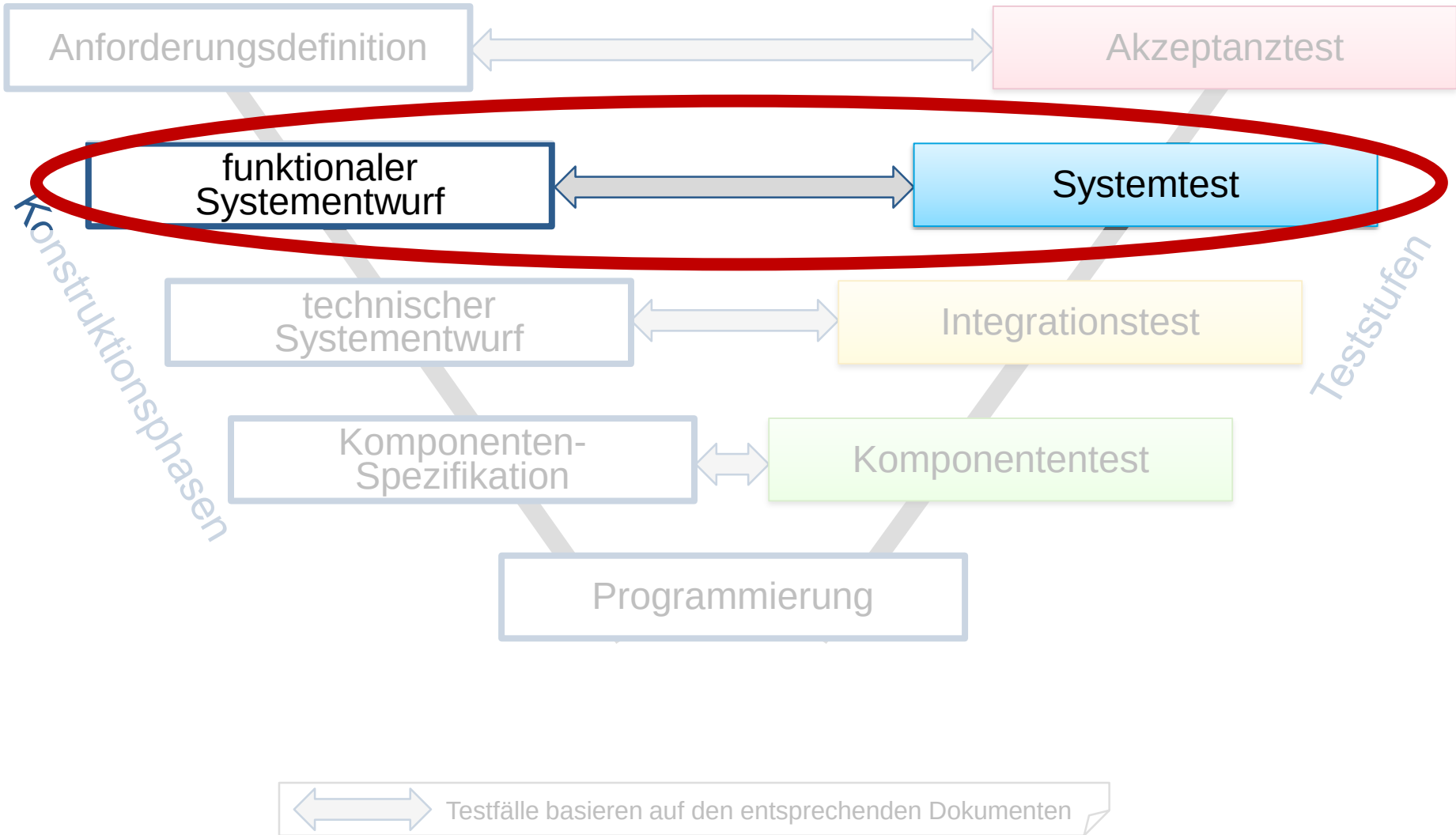


Errors Found

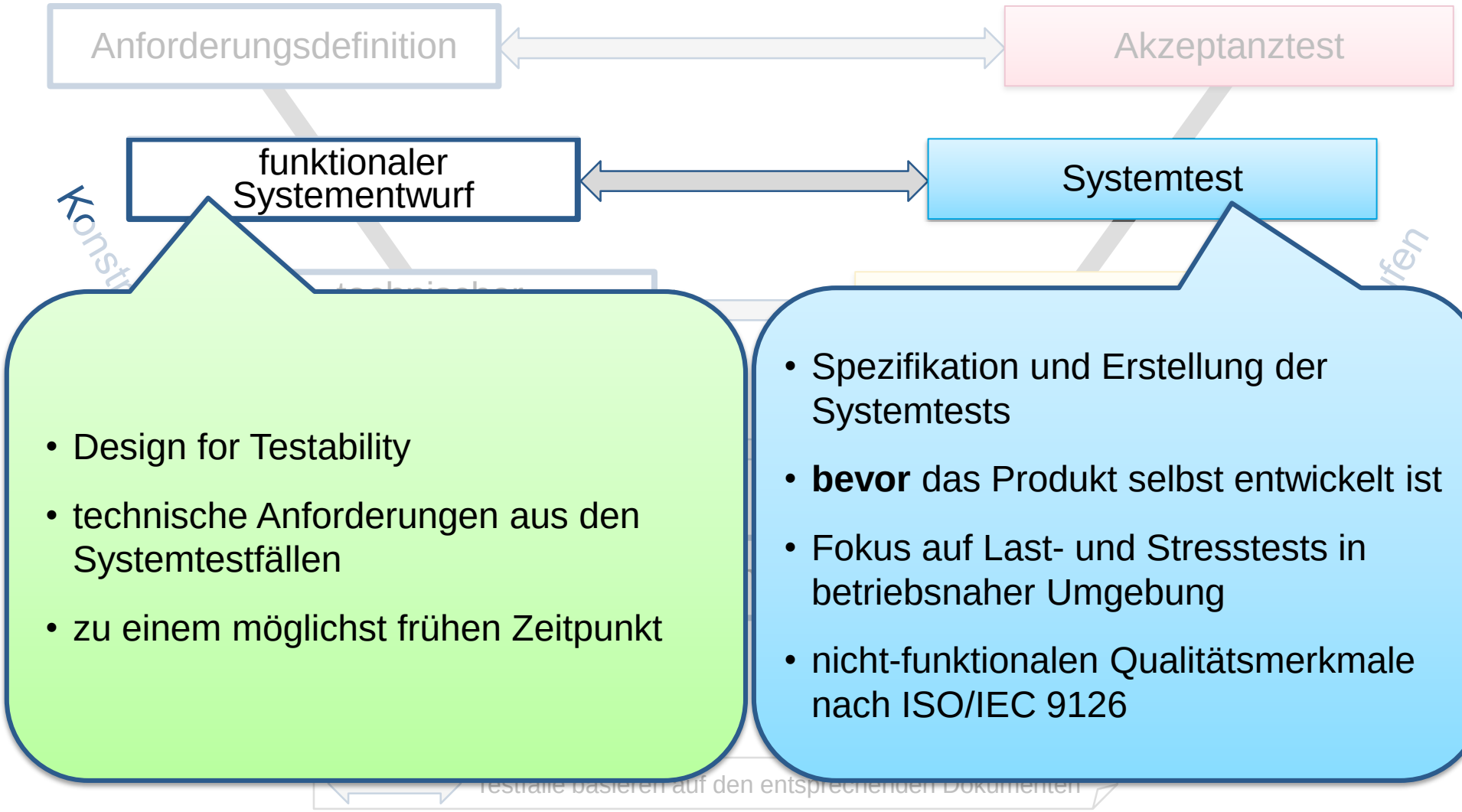


Source: Beltracchi, Leo: Notes on a Means-Ends Requirements Hierarchy. In: Halden Reactor Project HPR-348, Volume I; Loen 1996;
For the data reference is given to: Koss, E.: Developing Reliable Space Flight Software, Las Vegas, Nevada, January 28-30, 1986.

Test-First im Systemtest



Test-First im Systemtest



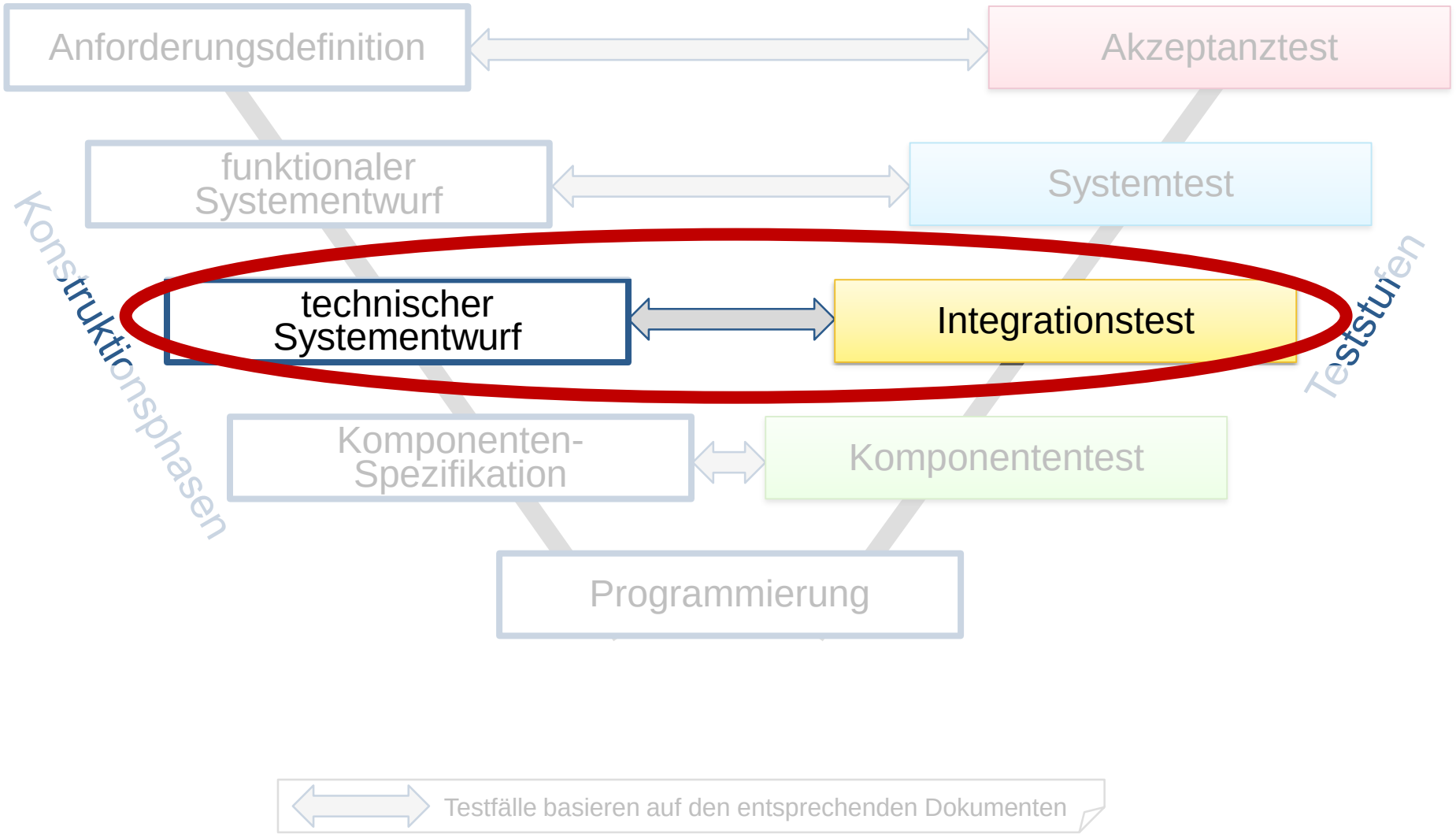
Test-First im Systemtest

Qualitätsmerkmal	Qualitäts-Teilmerkmal
Funktionalität	Angemessenheit, Richtigkeit, Interoperabilität, Ordnungsmäßigkeit, Sicherheit
Zuverlässigkeit	Reife, Fehlertoleranz, Wiederherstellbarkeit
Benutzbarkeit	Verständlichkeit, Erlernbarkeit, Bedienbarkeit
Effizienz	Zeitverhalten, Verbrauchsverhalten
Änderbarkeit	Analysierbarkeit, Modifizierbarkeit, Stabilität, Prüfbarkeit
Übertragbarkeit	Anpaßbarkeit, Installierbarkeit, Konformität, Austauschbarkeit

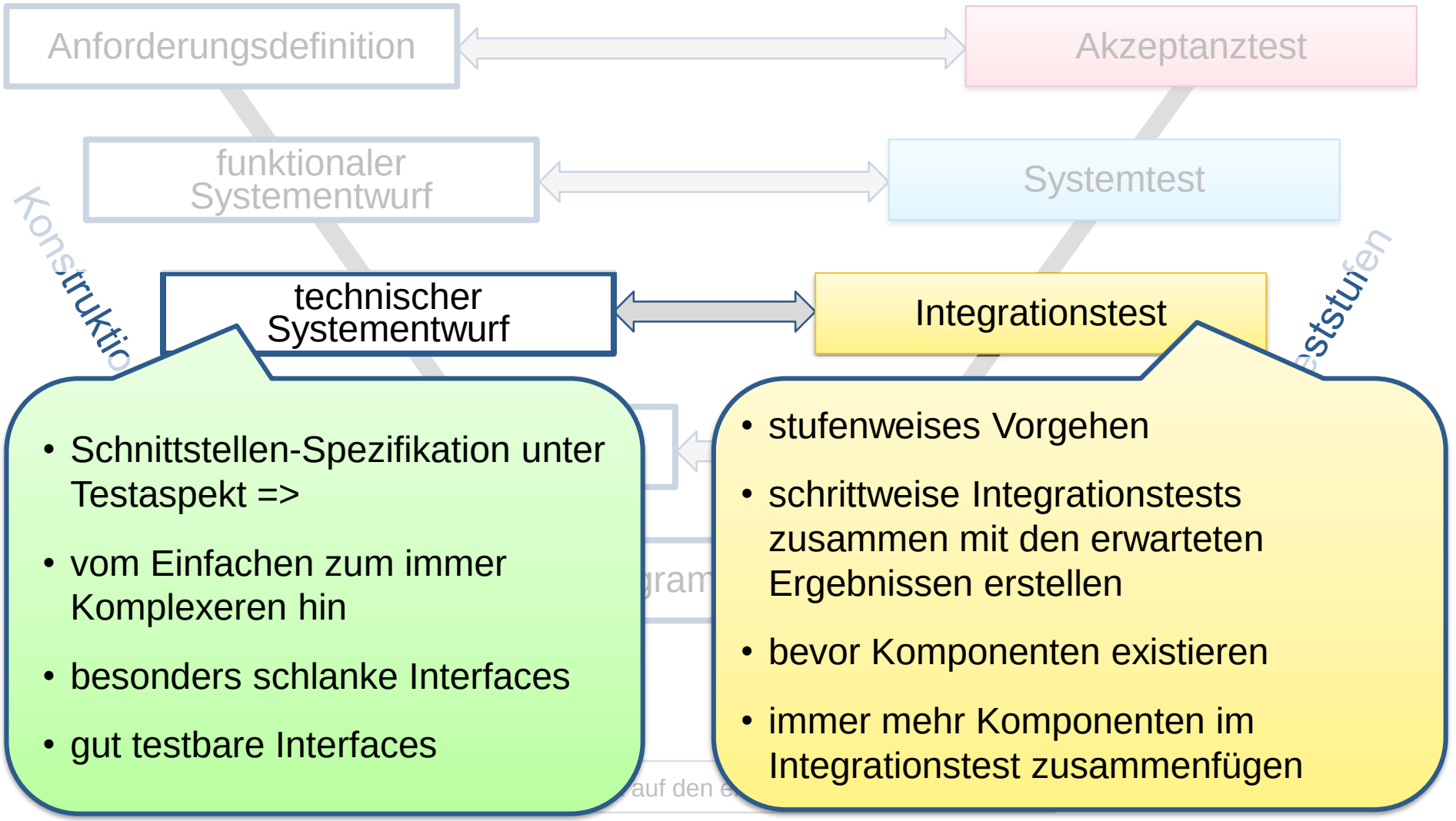
- zu einem möglichst frühen Zeitpunkt

- nicht-funktion… Qualitätsmerkmale nach ISO/IEC 9126

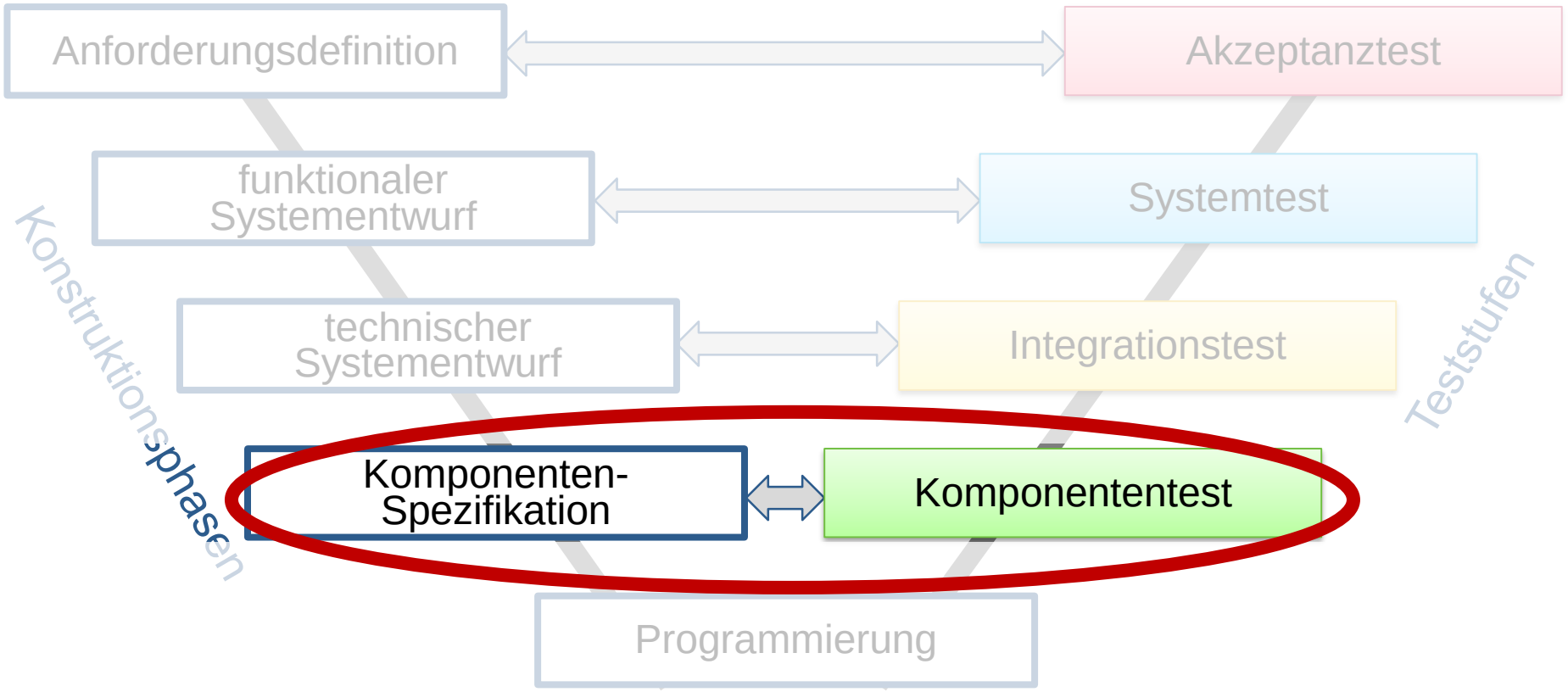
Test-First im Integrationstest



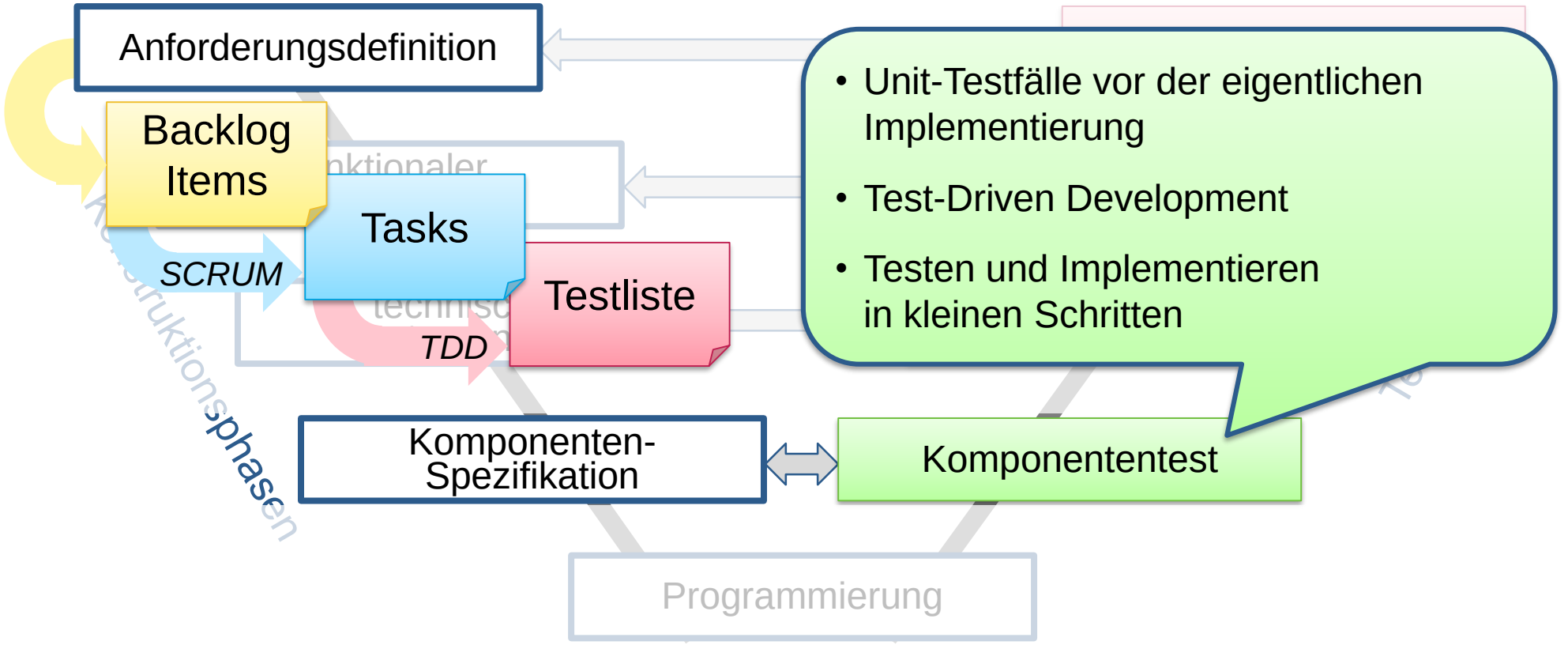
Test-First im Integrationstest



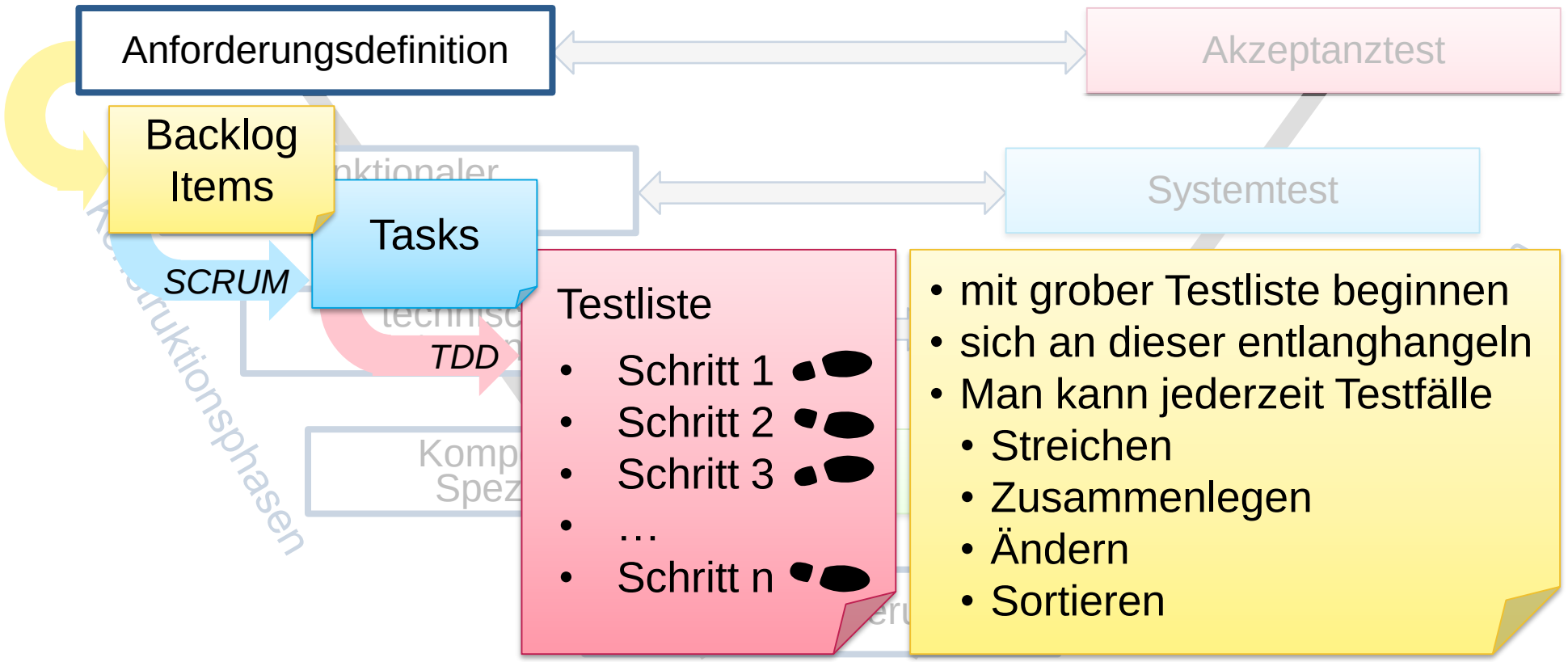
Test-First im Komponententest



Test-First im Komponententest



Test-First im Komponententest



Test-First im Komponententest

Iterative Code Entwicklung anhand automatisierter Testfälle

SCRUM

TDD

Testliste

- Schritt 1
- Schritt 2
- Schritt 3
- …
- Schritt n

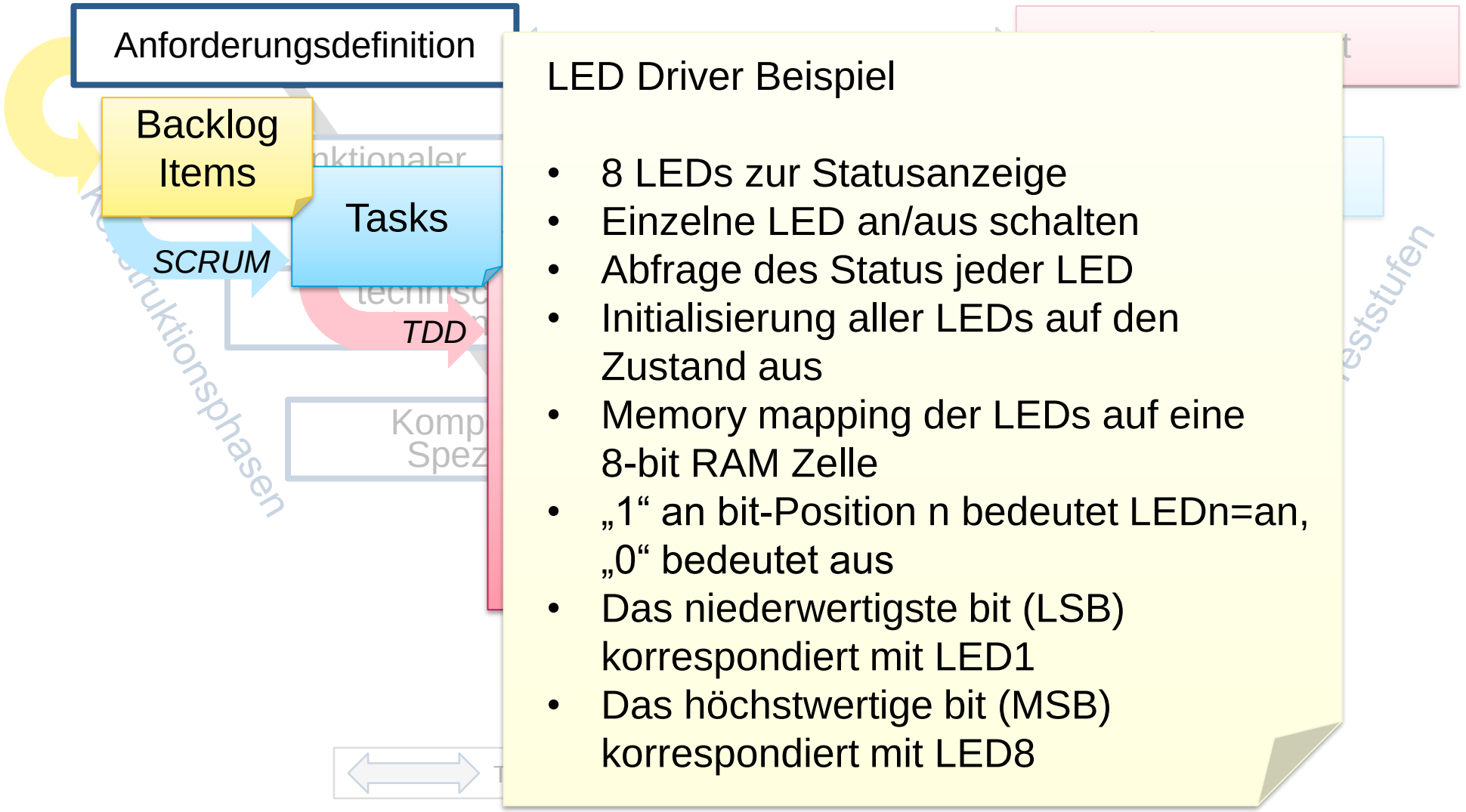
Integrationstest

Komponententest

Konstruktionsphasen

Testfälle basieren auf den entsprechenden Dokumenten

Beispiel Testliste

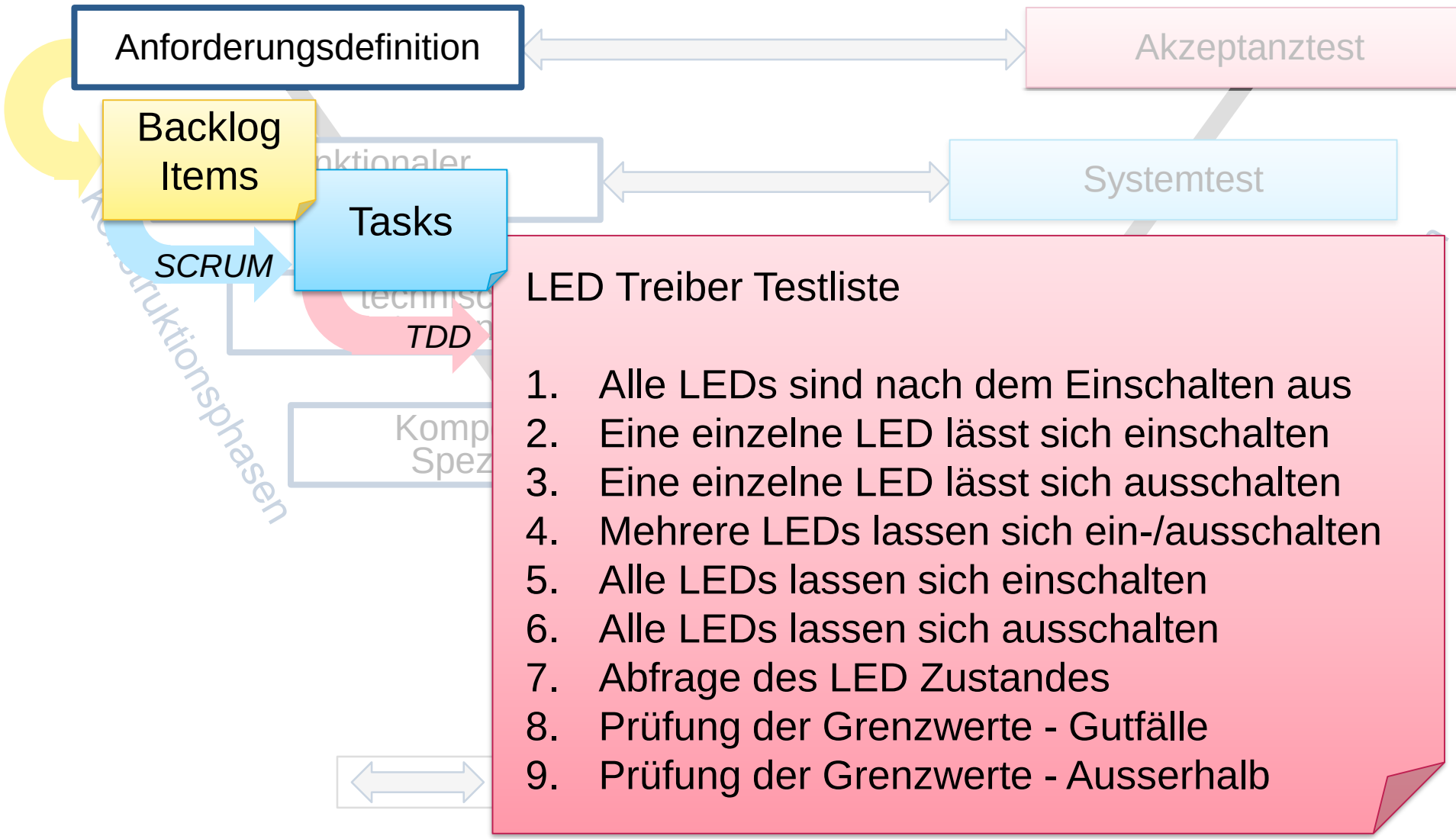


LED Driver Beispiel

- 8 LEDs zur Statusanzeige
- Einzelne LED an/aus schalten
- Abfrage des Status jeder LED
- Initialisierung aller LEDs auf den Zustand aus
- Memory mapping der LEDs auf eine 8-bit RAM Zelle
- „1“ an bit-Position n bedeutet LEDn=an, „0“ bedeutet aus
- Das niederwertigste bit (LSB) korrespondiert mit LED1
- Das höchstwertige bit (MSB) korrespondiert mit LED8

Quelle: James W. Grenning, Test-Driven Development for Embedded C

Beispiel Testliste

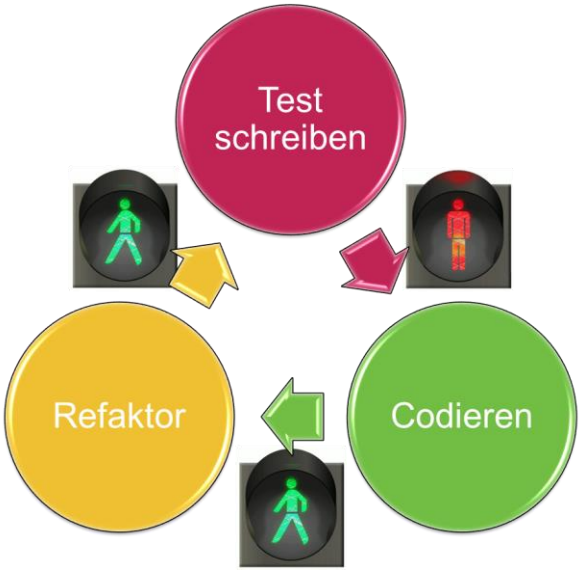


Quelle: James W. Grenning, Test-Driven Development for Embedded C

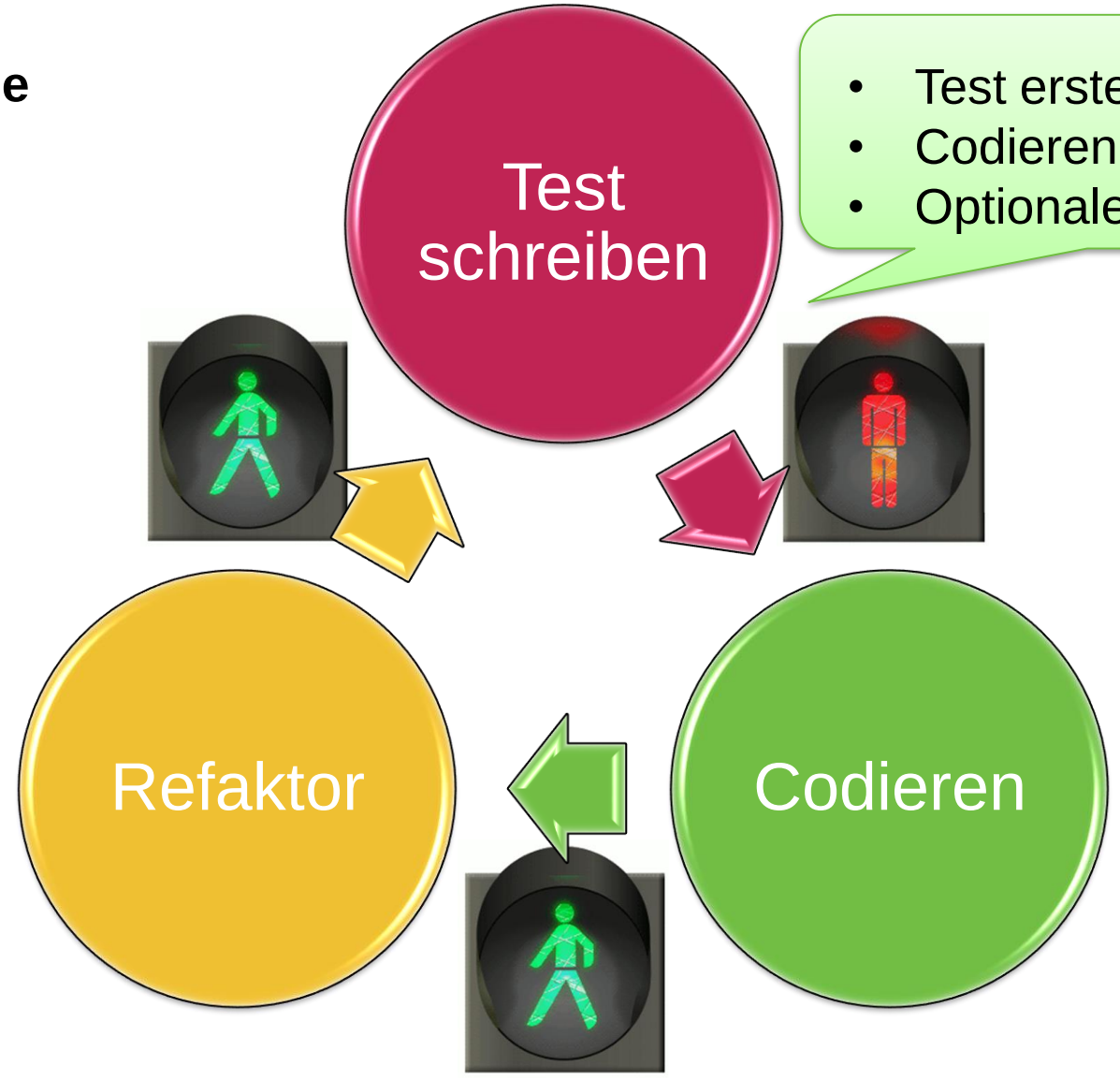
Test-Driven Development

Iterative Code Entwicklung anhand automatisierter Testfälle

Jeder inkrementelle Schritt besteht aus einer Iteration des **TDD-Cycles**:

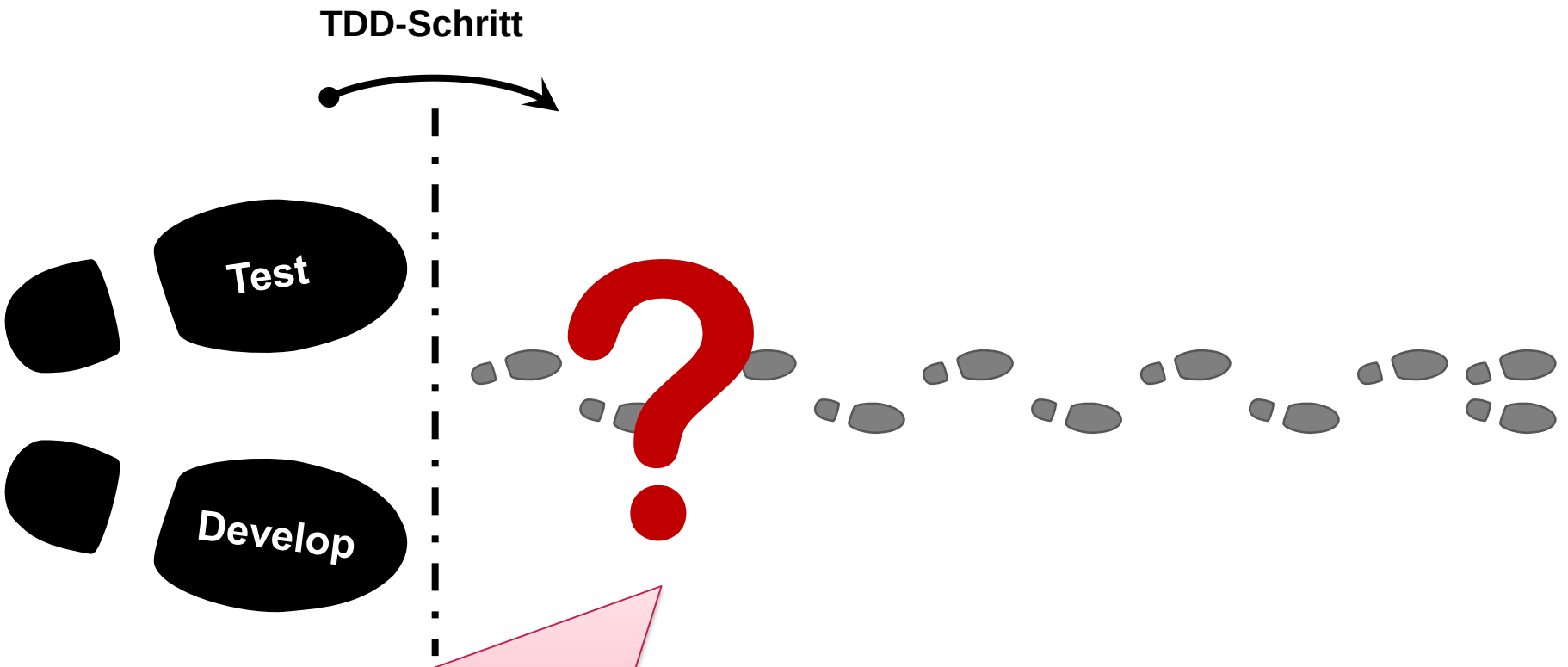


TDD-Cycle



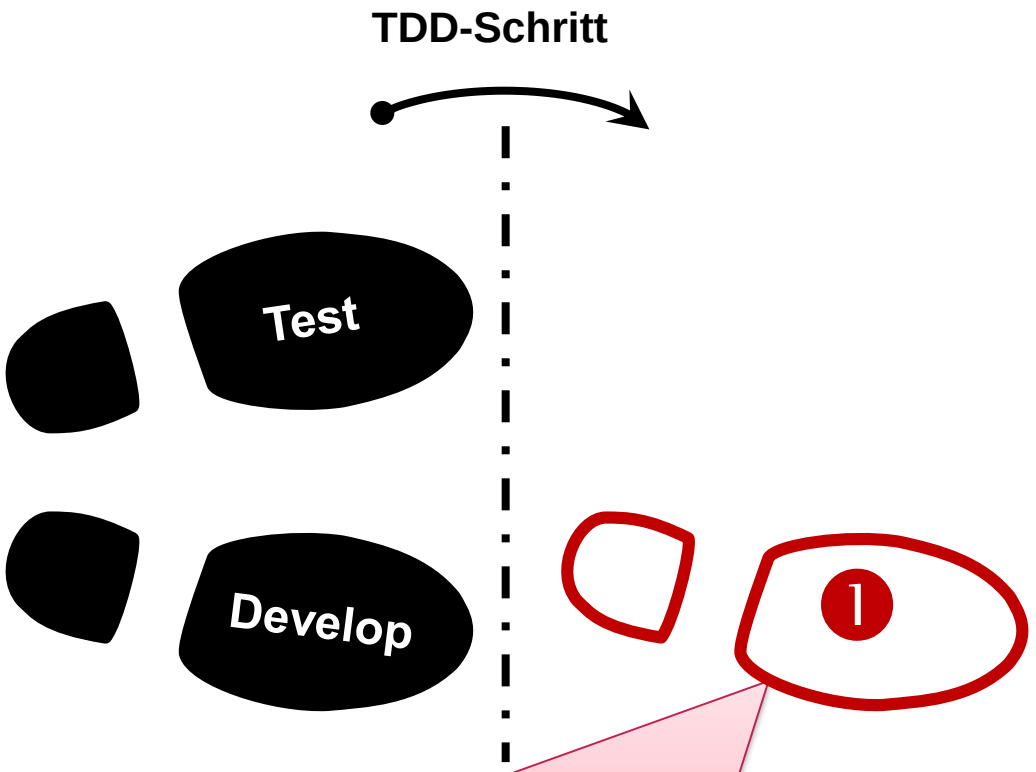
- Test erstellen, Testablauf
- Codieren
- Optionales Refactoring

=> Anzahl der Tests wächst mit jedem Cycle kontinuierlich an

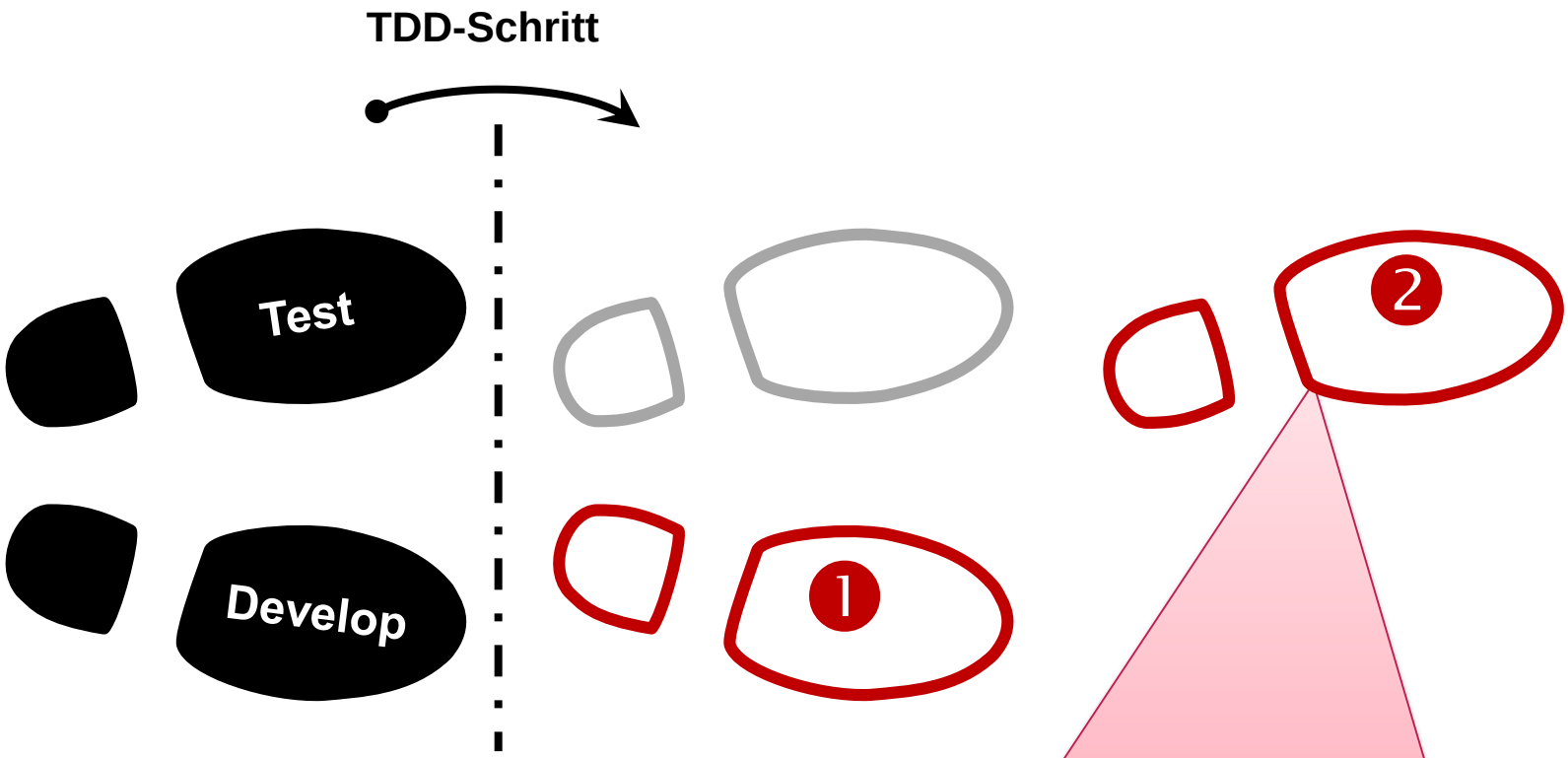


➔ Drei TDD-Regeln nach Bob Martins*

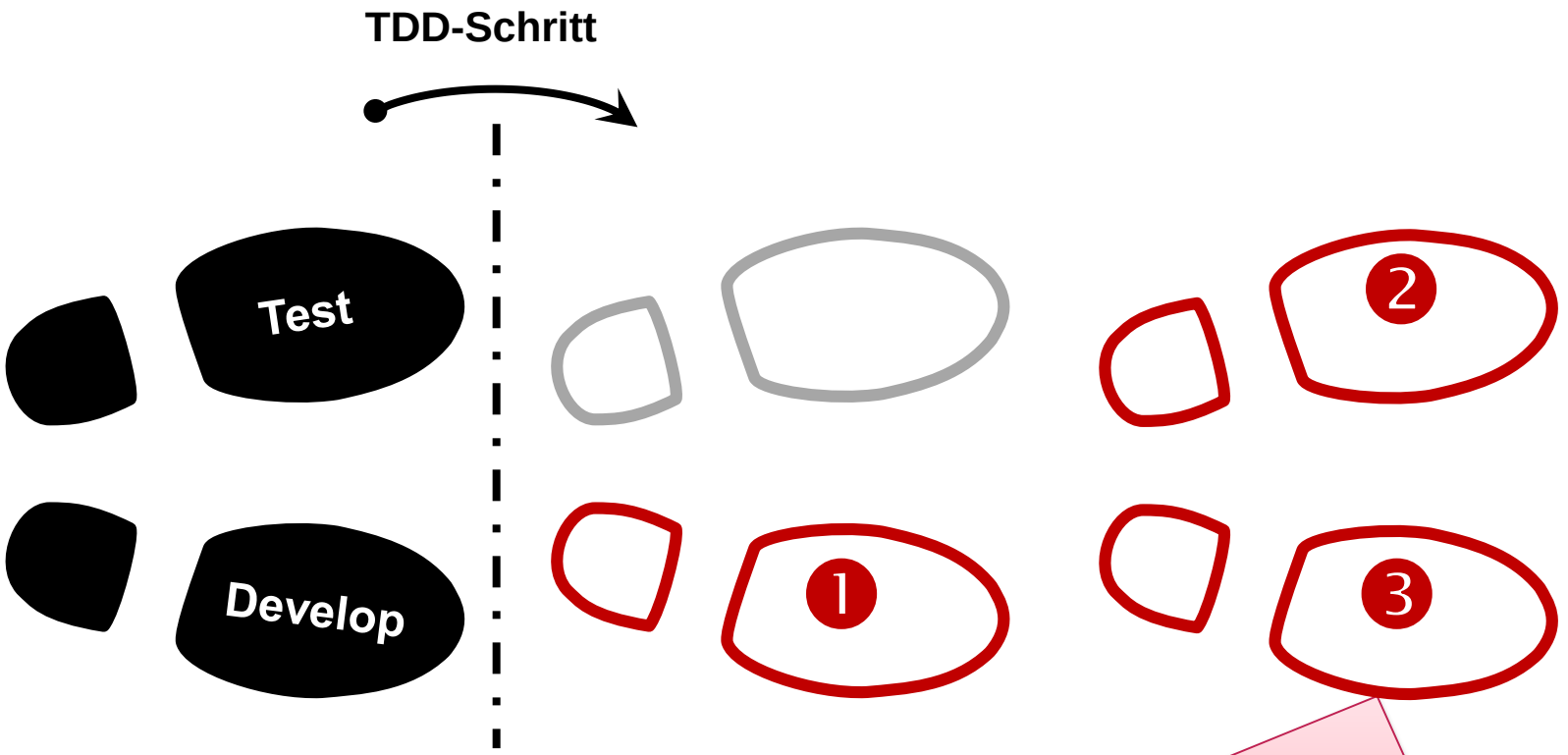
*) http://butunclebob.com/ArticleS.UncleBob.TheThreeRulesOfTdd



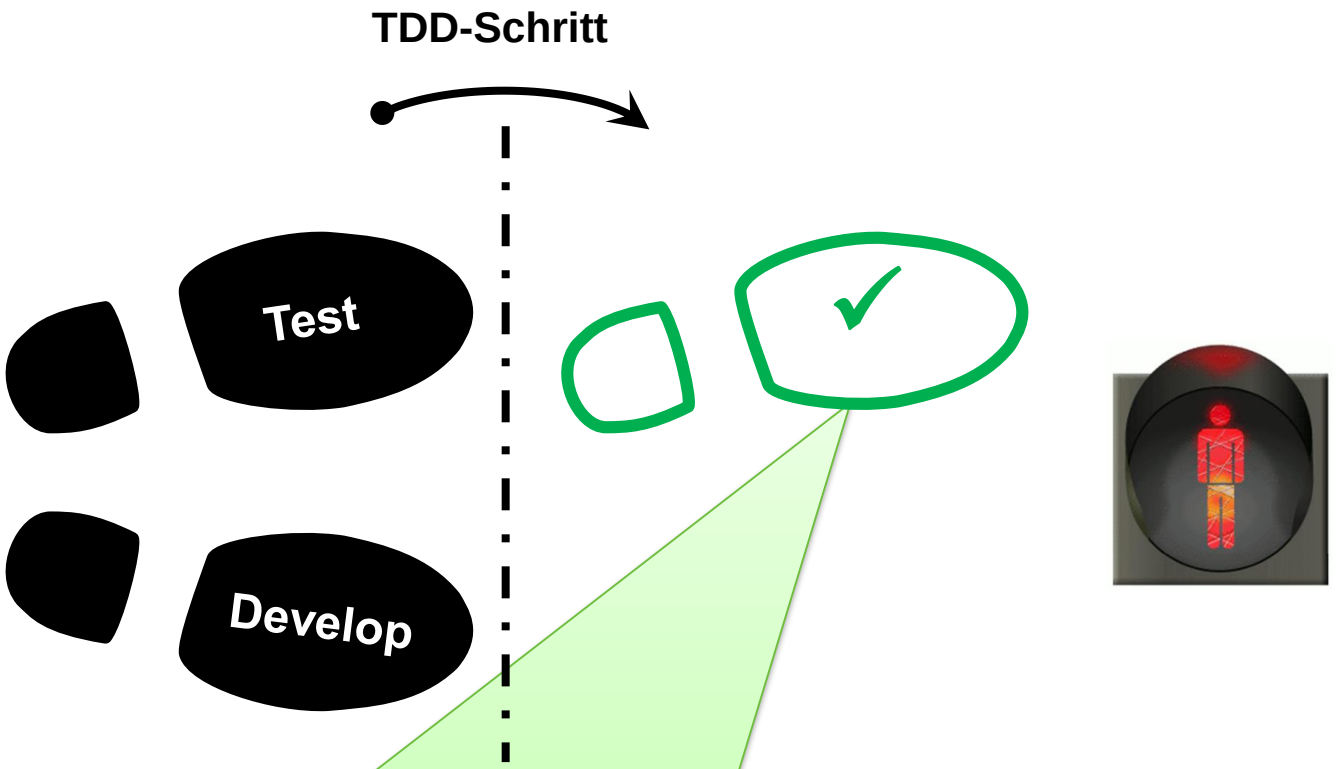
1. Schreibe keinen Code solange dieser nicht zur erfolgreichen Durchführung eines Testfalls erforderlich ist.



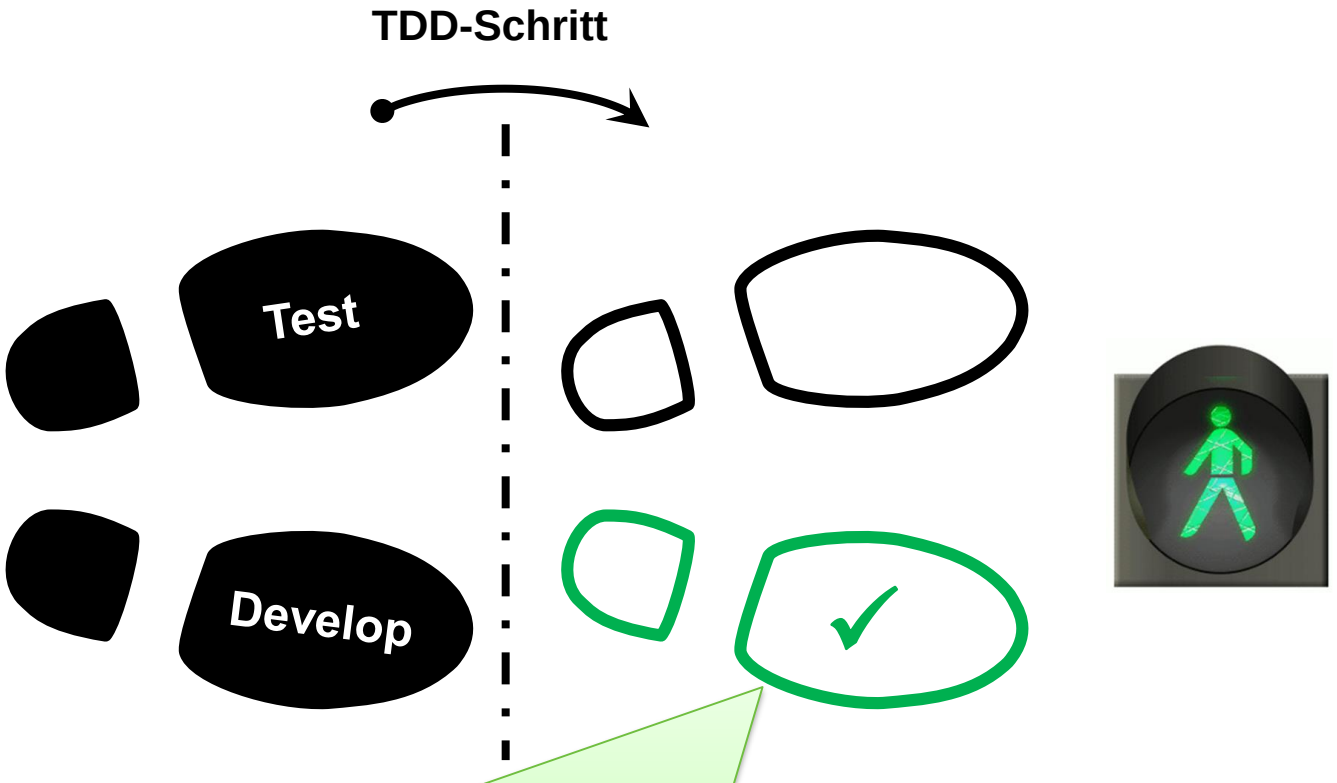
2. Begrenze den Umfang eines Unit-Testfalls darauf fehl zu schlagen. Auch Compiler-/Linker-Fehler sind Fehler.



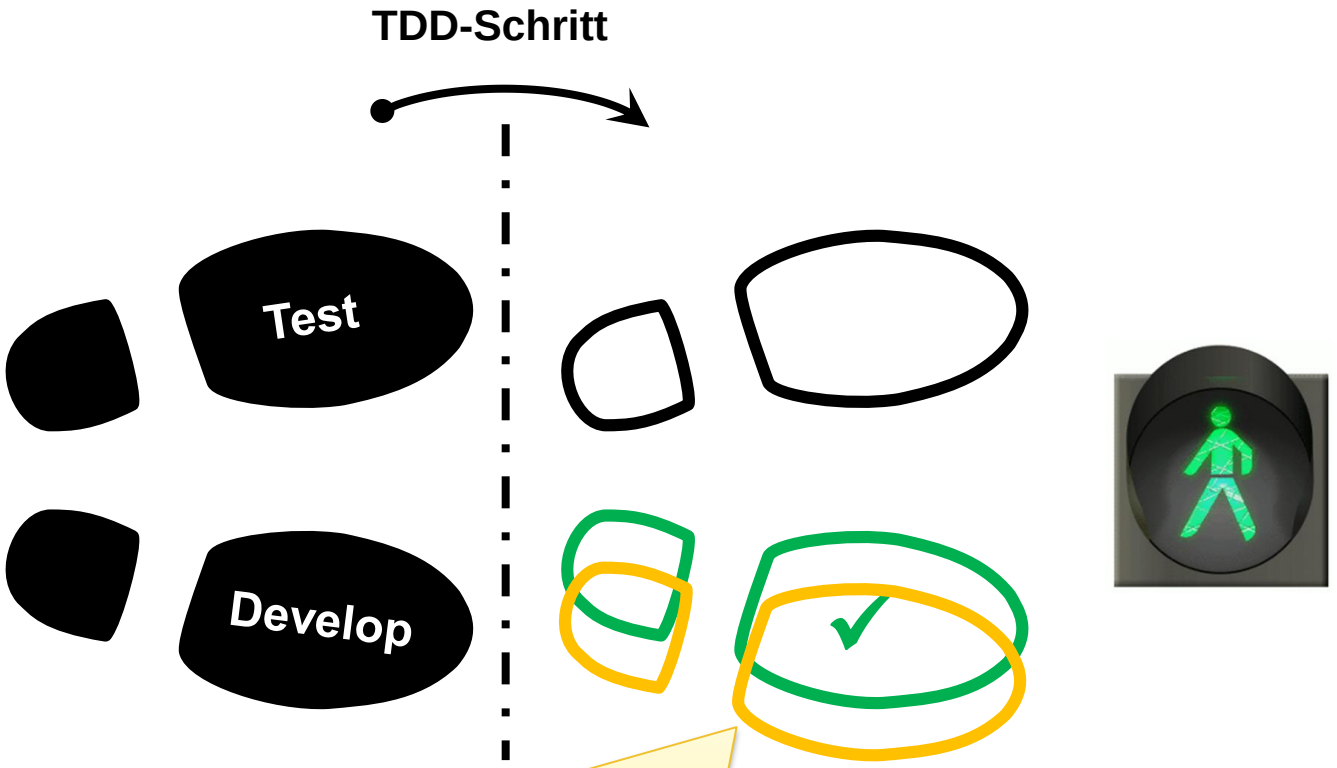
3. Schreibe nicht mehr Code, als erforderlich ist, um den aktuellen Unit-Testfall zu bestehen.



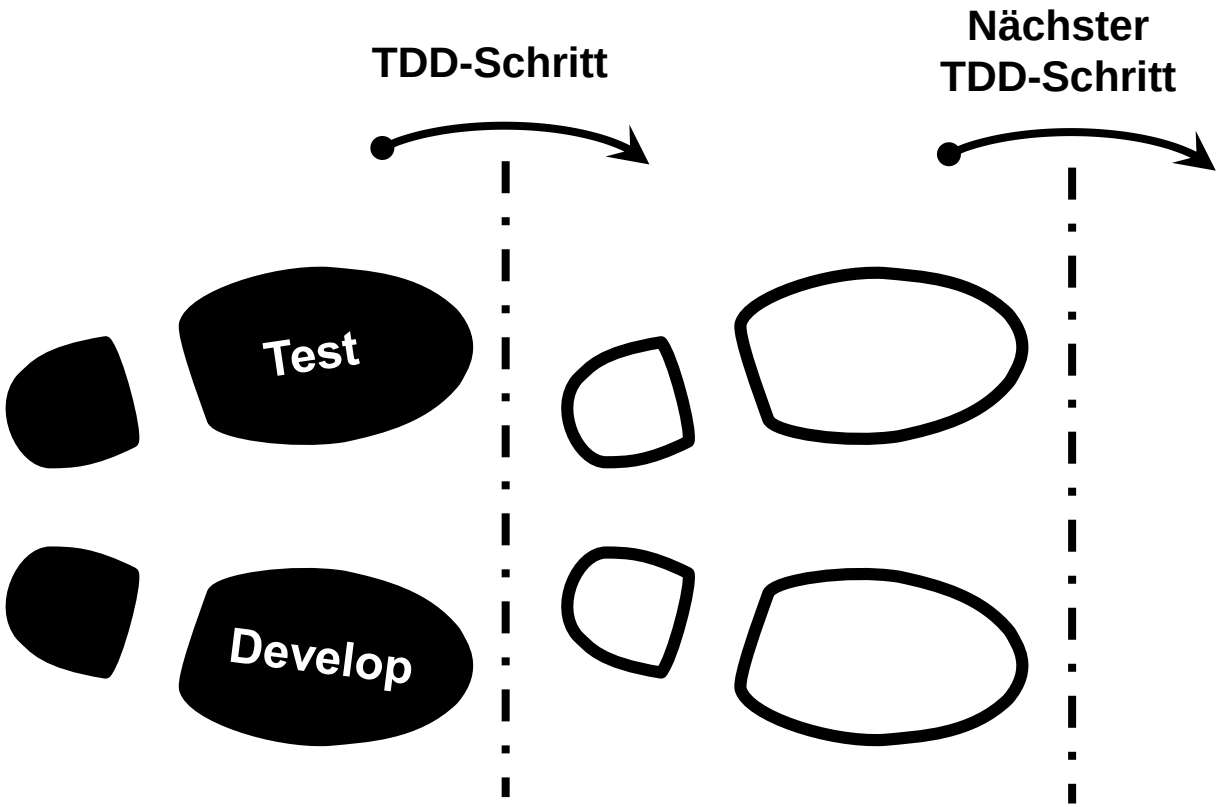
➔ Gültiger erster Schritt: Test schreiben, Test schlägt fehl



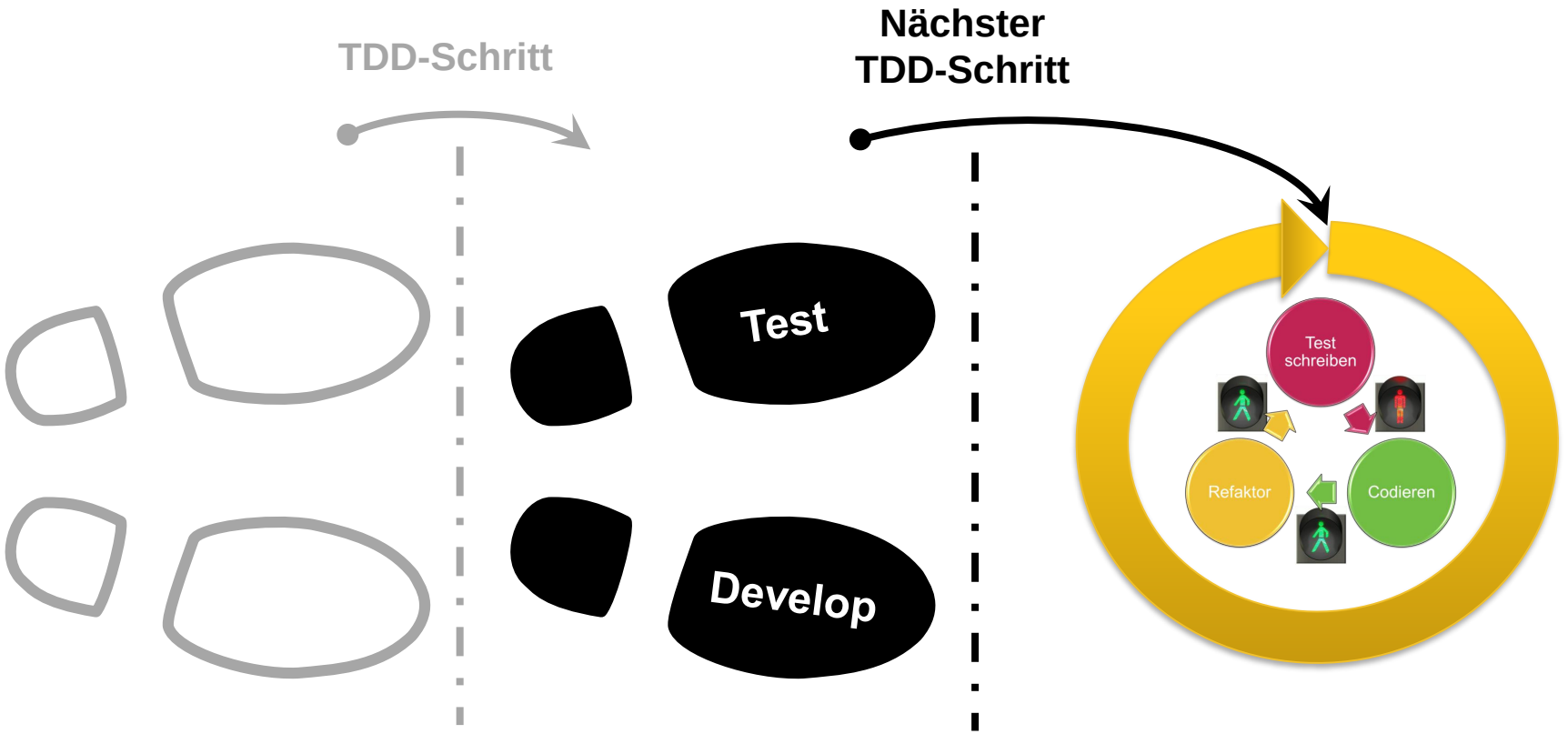
➔ Gültiger nächster Schritt: Codieren bis Test gelingt



➔ Optionaler nächster Schritt: Refaktorieren und alle bisher erstellten Tests MÜSSEN gelingen



➔ Nächste Runde im TDD Cycle starten


TDD-Schritt
Nächster
TDD-Schritt
Test
Develop
Test
schreiben
Refaktor
Codieren

Embedded Test-Driven Development

Test-Driven Development =>

- Viele kleine Änderungen
- Tests erstellen
- Code Stück für Stück inkrementell erweitern
- TDD Cycle: Test-Code-Test-Code……Refactoring

TDD von Embedded Systemen =>
Hindernisse bekannt als **„Das Target Hardware Bottleneck“**

- Test Suiten auf dem Embedded System
- Lange Upload Zyklen
- Knappe Ressourcen
- Späte Verfügbarkeit des Target-Systems

TDD von Embedded Systemen
Spezielle Strategien => Überwindung der Hindernisse

Dual Targeting

- Von Tag 1 an
- Codierung für >= 2 Plattformen
  - Entwicklungs-Plattform
  - Target-Plattform
- Ermöglicht Entwicklung mit konstanterer Geschwindigkeit
- Vermeidet Probleme mit Anhäufung von ungetestetem Code
- Umgehung des Hardware Bottlenecks
- Erlaubt Testen von Code vor Verfügbarkeit der Target-Hardware

Positiver Seiteneffekt:

- Beeinflusst das Design
- Mehr Augenmerk auf klare HW-un-/abhängige Strukturierung

=> saubereres Layering der Software

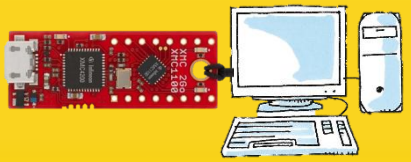
=> später leichter auf andere Systeme portierbar

Embedded TDD Strategien

1. Test on Host

2. Test on Target

3. Remote Testing



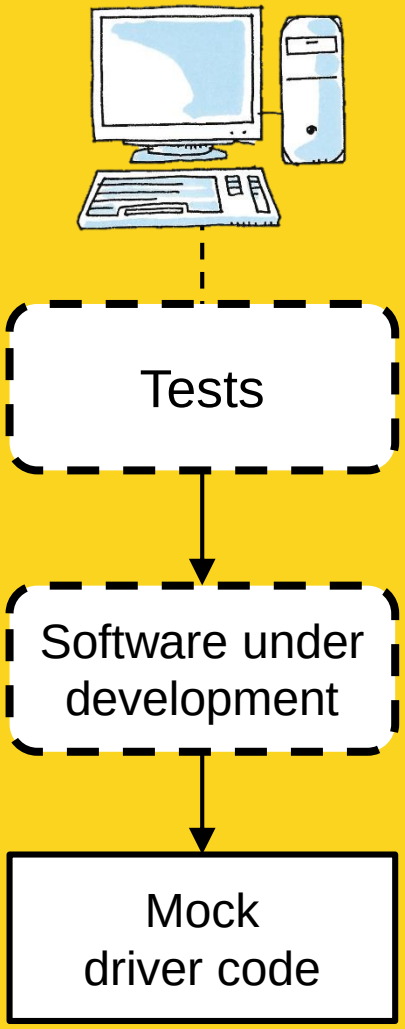
Quelle: Test-Driven Development Strategies applied to Embedded Software,
Cordemans, Piet, Boydens, Jeroen, Van Landschoot, Sille, Steegmans, Eric, 30. Jannuar 2012

1 Test on Host

- Durchführung der Tests auf der Entwicklungsumgebung
- Ersetzen der Hardware Abhängigkeiten durch Mocks
- Verifizieren des Mockverhaltens auf dem Target

3 Remote Testing

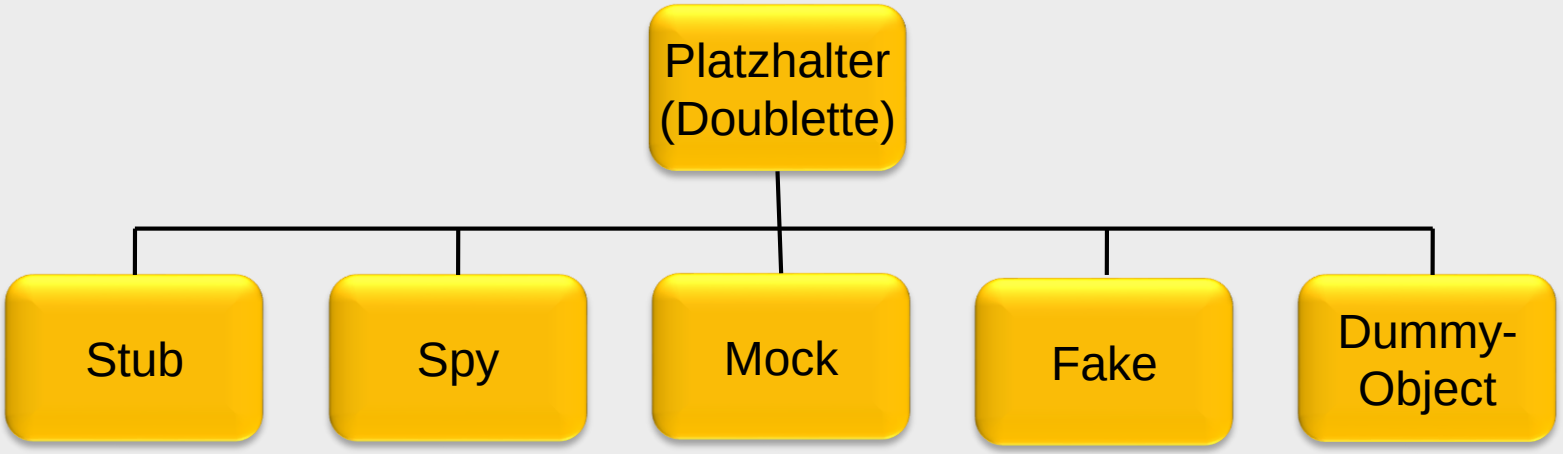
1 Test on Host Strategie



Stubs, Mocks und Dummies

- Möglichst isolierter Test des Testobjektes (Modul, Funktion)
- Ersatz der Abhängigkeiten des Testobjektes durch Platzhalter
- Bereitstellung noch nicht implementierter Teile

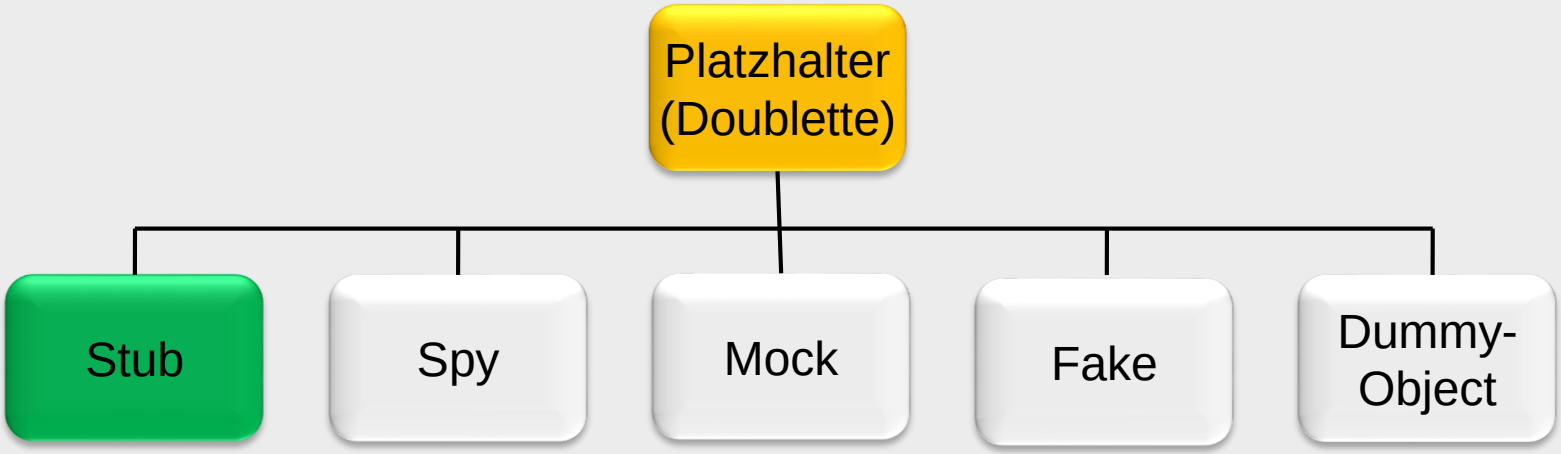
- Verschiedene Platzhaltertypen:



Stubs

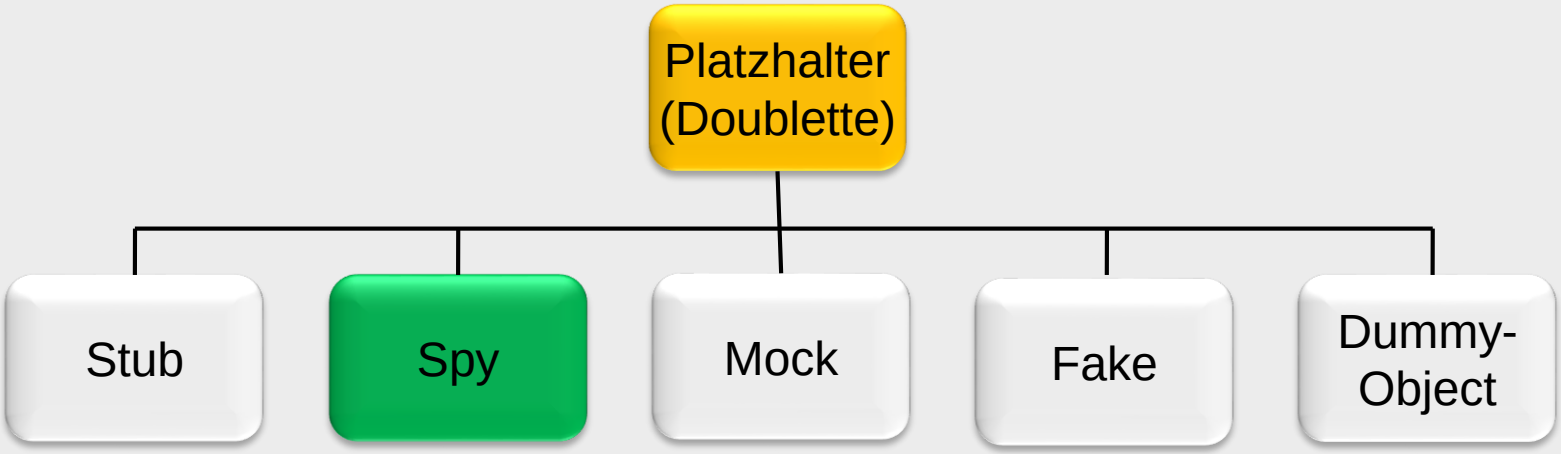
- Identisches Interface
- Reaktionen nach festem, ausgewähltem Muster

- Die Reaktionen werden zu jedem einzelnen Testfall festgelegt
  => zusätzlicher indirekter Test-Input-Parameter (Indirect Input)



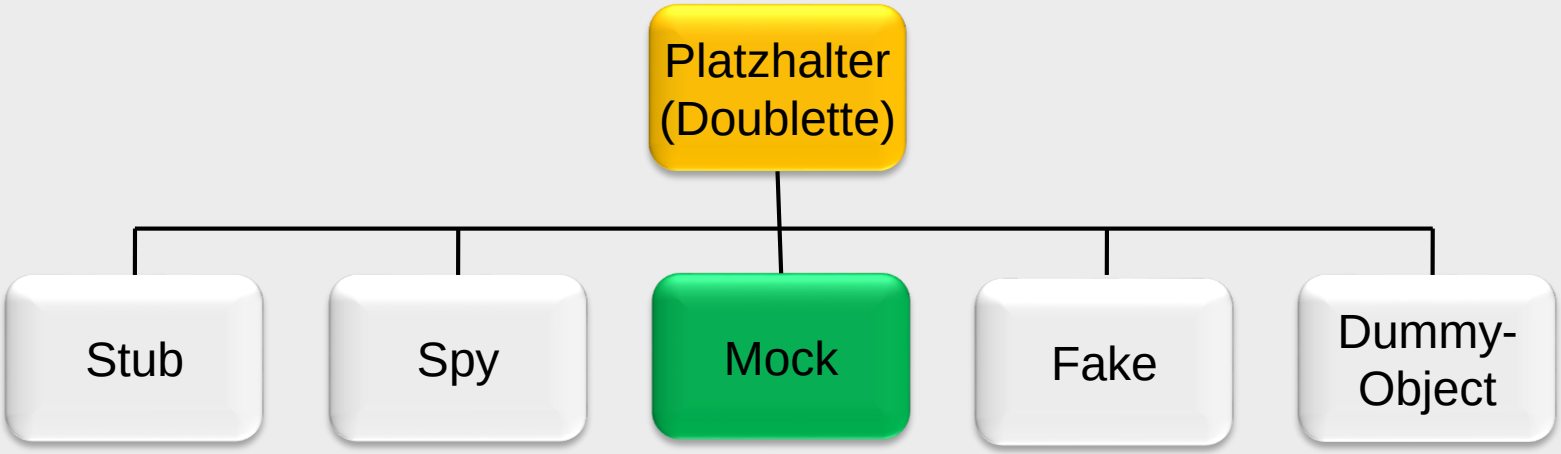
Spy

- Wie Stub, zusätzlich Protokollierung
- „Ausspionieren“ der Aufrufe und übergebenen Daten (Indirect output)
- Protokollierte Daten werden verwendet für
  - Feststellung des Testergebnisses
  - Debugging



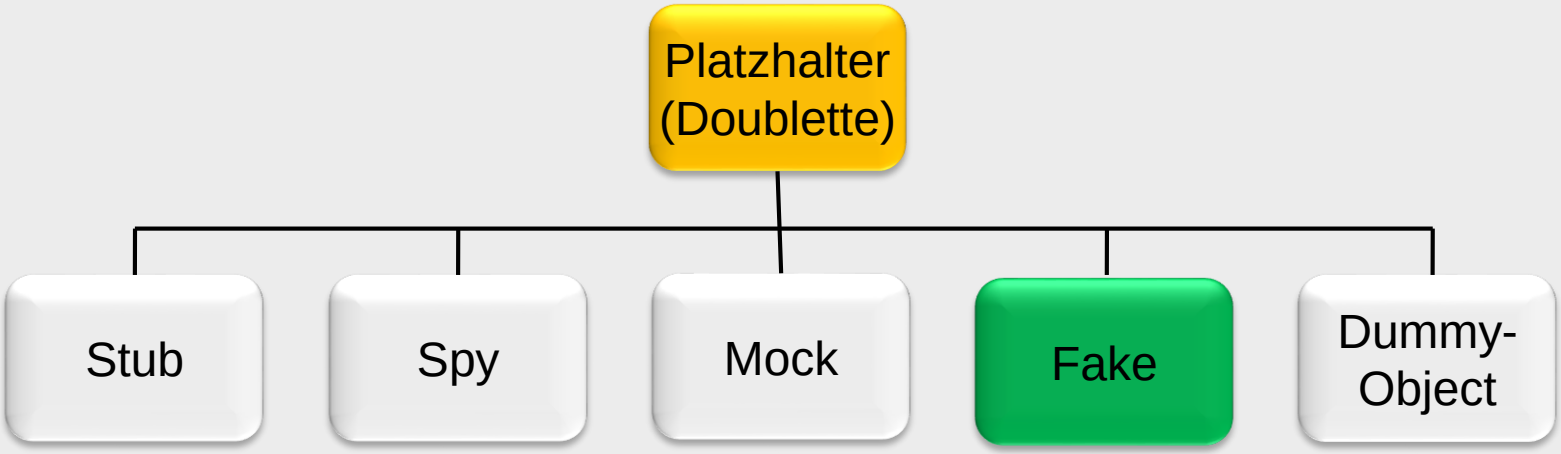
Mock

- Intelligenter Stub
- Auswertung der Aufrufe und übergebenen Daten
- Prüfung auf Zulässigkeit und Korrektheit der Daten
- Reaktion abhängig von der Auswertung der Daten



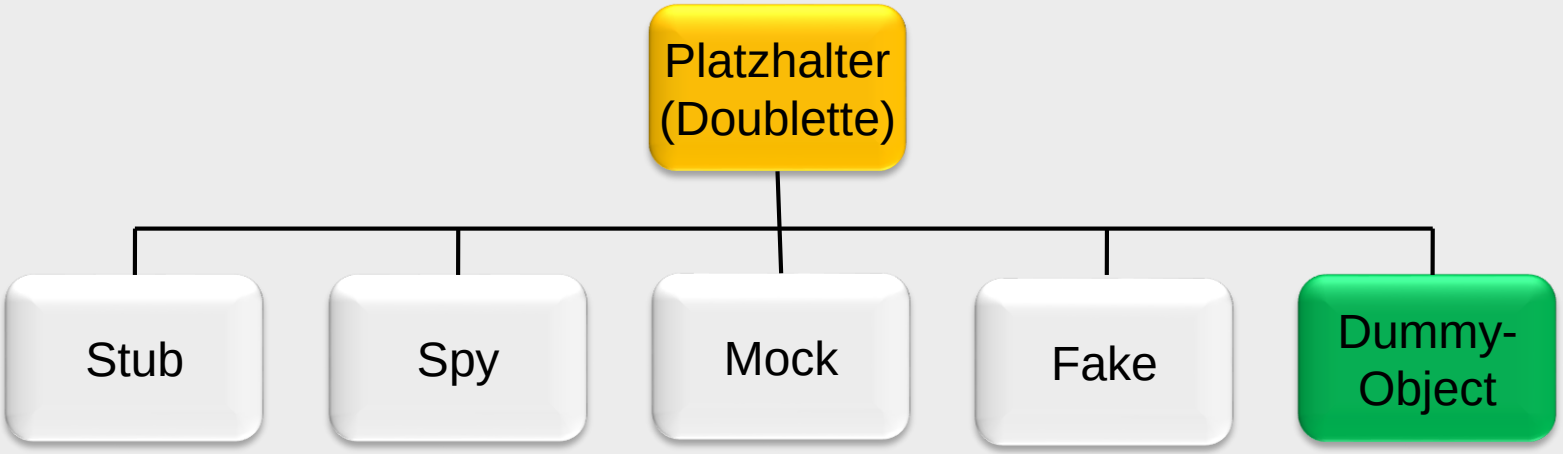
Fake

- Ersetzt eine Komponente von der das Testobjekt abhängig ist
- Stark vereinfachte Implementierung
- Kein weiterer Einfluss auf das Testergebnis



Dummy Object

- Wie Fake nur eben ein Objekt
- Pseudo-Objekt, leeres Objekt, Null-Pointer,….
- Kein weiterer Einfluss auf das Testergebnis



1 Test on Host Strategie

Tests

Software under development

Mock driver code

Verifizieren des Mockverhaltens

Tests

Software under development

Real driver code

1 Test on Host

Vorteile:
- Vermeidet häufige Uploads
- HW-unabhängigen Tests auf dem Host
- „Fake“ des HW-Verhaltens durch Mocks
- Plattform-unabhängige Codierung mit Layer Architektur

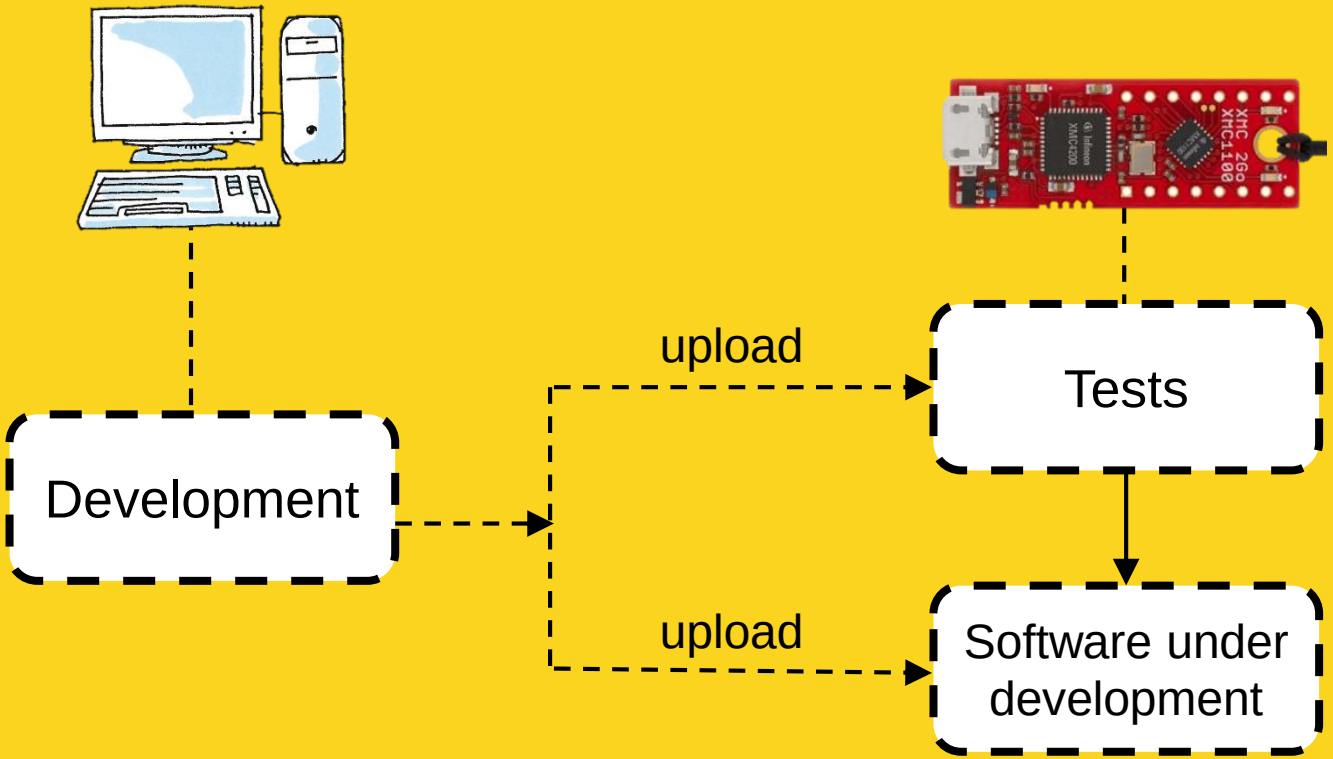
Nachteile:
- Kein Real-Time HW Verhalten wegen Mocks
- Keine Abdeckung von Cross-Plattform Problemen (CrossCompiler, Target Datenhaltung, Mockfehler)
- HW-abhängige Tests auf dem Target erforderlich
- Dual Build für Host und Target erforderlich

1 Test on Host

2 Test on Target

- Tests und Testframework auf dem Target
- Durchführung der hardwareabhängigen Tests auf dem Target

2 Test on Target Strategie



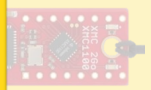
Vorteile:
- Reelle Target Testumgebung
- Test des Real-time Verhaltens möglich
- Einfacher Test zusammen mit Legacy Code
- Komplett Validierung und Verifizierung
  (sofern ausreichend Ressourcen vorhanden sind)

2 Test on Target

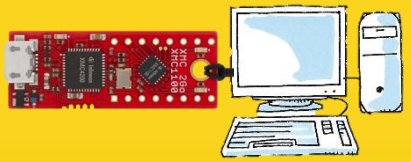
Nachteile:
- Häufige Uploads
- Ressourcen für Testumgebung
- Späte Verfügbarkeit des Target-Systems
- Nur für HW-abhängige Tests praktikabel
- Nur im Zusammenspiel mit zusätzlicher Teststrategie

1 Test on Host

- Inter-Prozess Kommunikation
- Tests und hardwareunabhängiger Code auf dem Host
- Hardwareabhängiger Code läuft auf dem Target



3 Remote Testing



3 Remote Testing Strategie (mit IPC Kommunikation)

Tests

Software under development

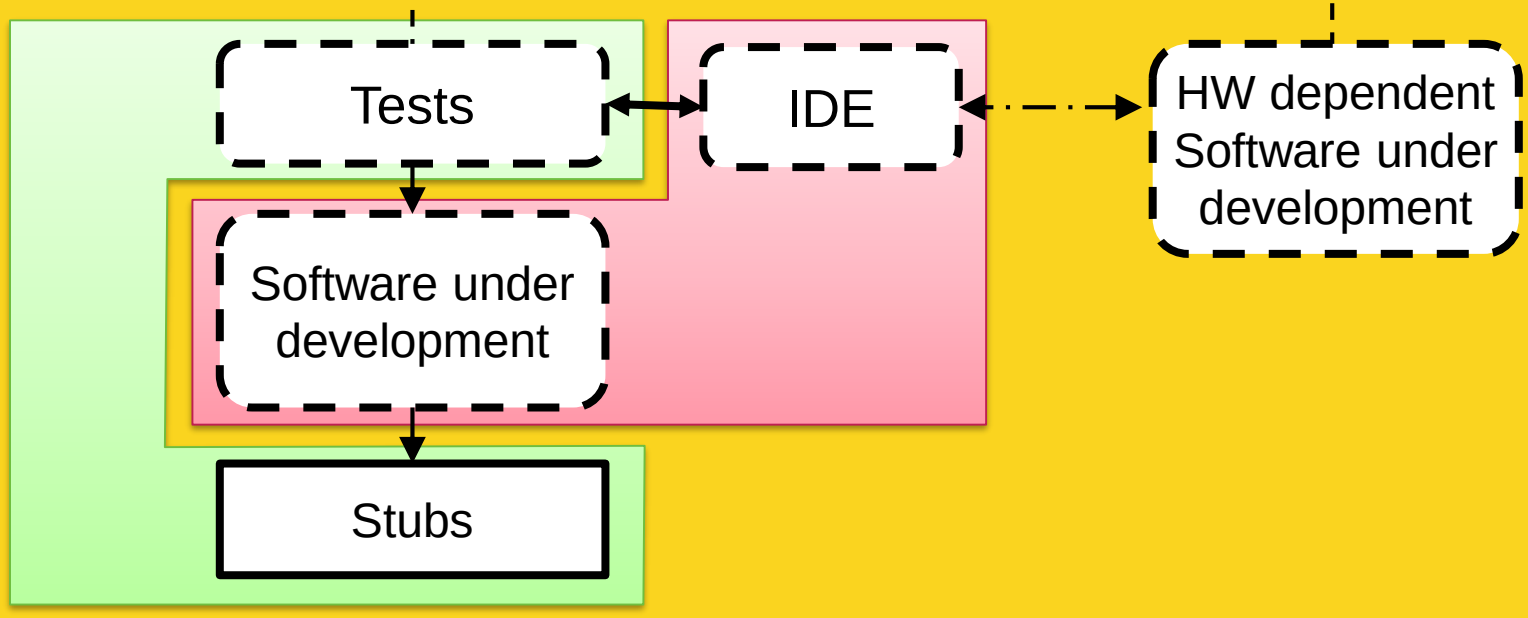
Stubs

Broker on Target

HW dependent Software under development

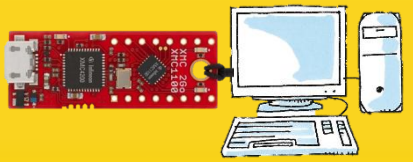
Broker on Host

3 Remote Testing Strategie (mit Test-SW <> IDE Kommunikation)



Vorteile:
- Ressourcen für Testumgebung nur auf dem Host
- HW-unabhängigen Tests auf dem Host
- HW-abhängige Tests auf dem Target
- IPC Kommunikation oder
- Unterstützende Test-Software

3 Remote Testing

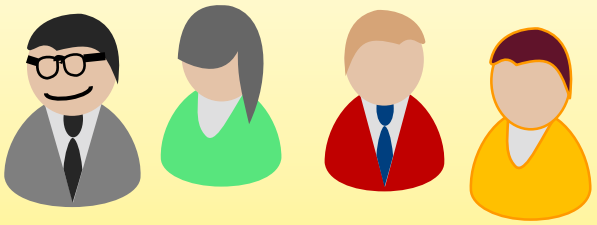


Nachteile:
- Dual Build für Host und Target erforderlich
- Komplexität des Testsystems

Anwendungsmöglichkeit für TDD

Umsetzung individuell:

- TDD
- traditionell



Anwendungsmöglichkeiten:

- Neues Projekt
- Neues Modul
- Erweiterung bestehender Module *(Testliste nicht vergessen!)*

Zusammenfassung Test-First-Ansatz

Ziel: Fehler so früh wie möglich aufdecken

Umsetzung:

- Tests erstellen bevor der jeweilige Testgegenstand existiert
- Tests möglichst früh ausführen
- Test-First lässt sich in allen Teststufen anwenden

- TDD ist die Umsetzung des Test-First-Ansatzes im Komponententest = Inkrementelles Entwickeln der Unit-Tests parallel zur Implementierung
- Die Einhaltung von nur drei Regeln und die richtige Strategie im Umgang mit dem Target-Hardware-Bottleneck ermöglichen TDD auch für Embedded-Systeme

Danke

www.microconsult.de Remo Markgraf, Tel. 089 450617-36, r.markgraf@microconsult.de