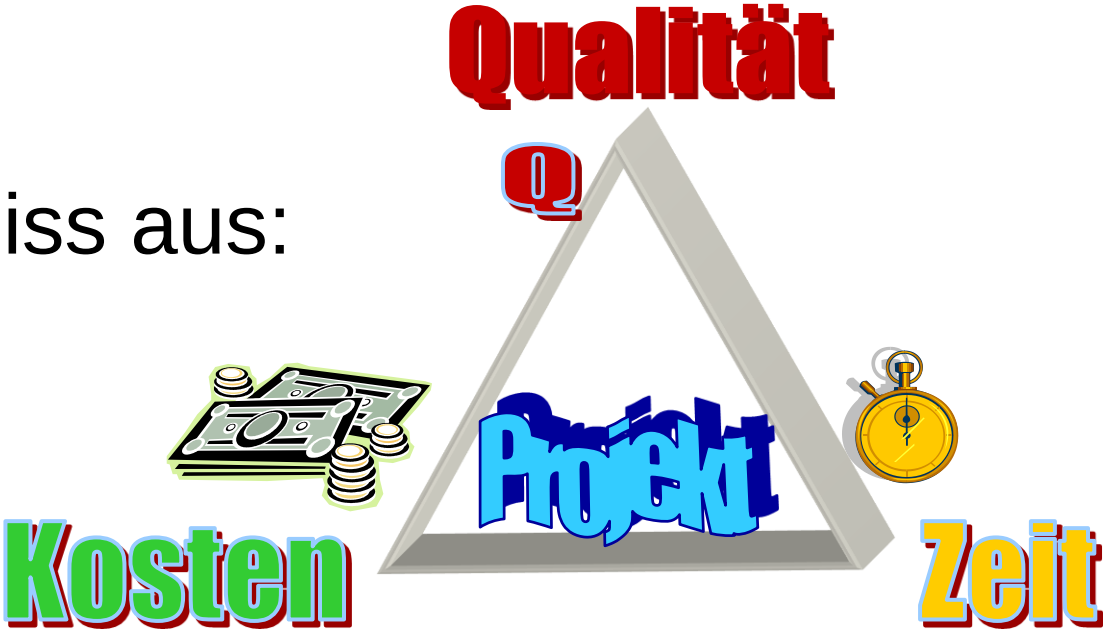
# Gesunde Software

**Frank Listing**
**f.listing@microconsult.com**

Typische **Herausforderungen** in **Software-Projekten**:

- Sehr **knapp bemessene Zeit**
- Die Software soll auf **mehreren Plattformen** eingesetzt werden.
- Der Kunde stellt nur **unvollständige Anforderungen** zur Verfügung.
- **Änderungswünsche** des Kunden im laufenden Projekt

Kompromiss aus:

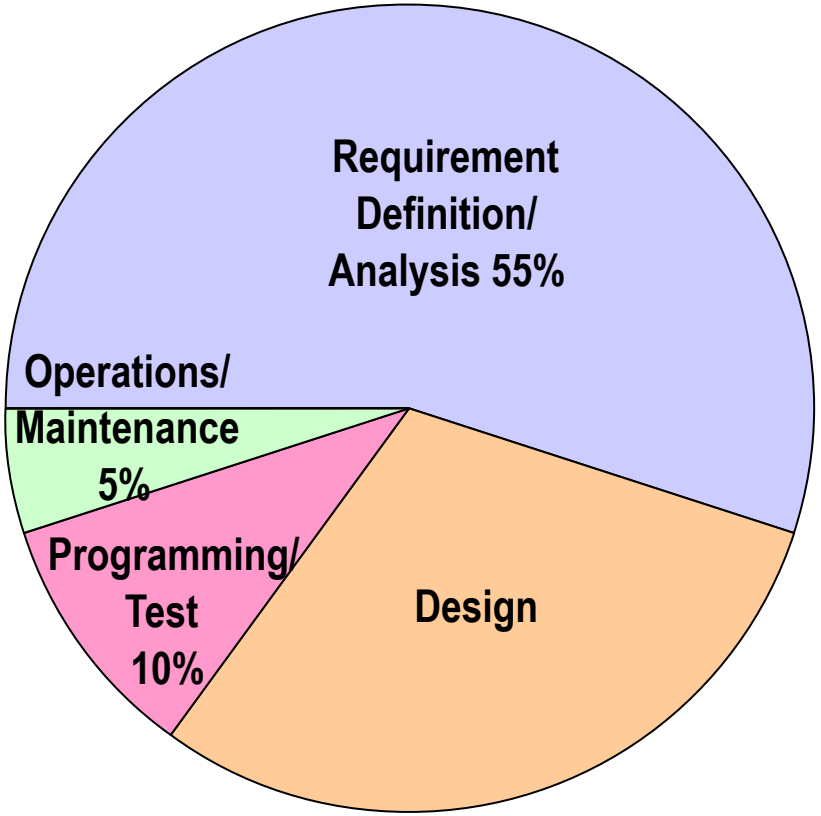


# Wie kann ein SW-Projekt auf den harten Alltag vorbereitet werden?

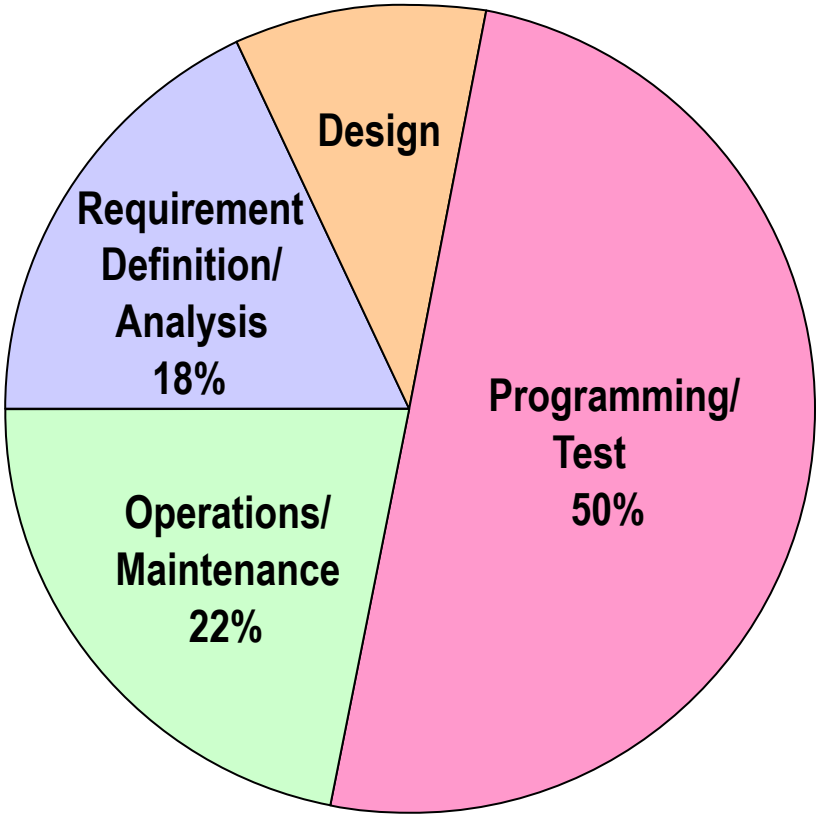
# Vorbeugen ist besser als Heilen

Wo sind die meisten Fehler und wann werden sie gemacht?

## Errors Introduced




## Errors Found




From: 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.

## **Erkenntnisse** aus der **Projektentwicklung**:

- **Ausreichend Zeit** in den **Projektstart** investieren.
- Aktiv mit dem Kunden zusammenarbeiten, um möglichst früh **vollständige Anforderungen** zu haben.
- Die Software so gestalten, dass sie mit den Anforderungen mitwächst und nachträgliche Änderungen einfach einzubringen sind.

Hier gilt - wie im richtigen Leben - das **Vorsorgeprinzip**:
Wird von Anfang an auf ein gesundes System hingearbeitet, reduziert sich der notwendige Aufwand und die Gesundheit der Software ist am stabilsten.

**Etwas mehr Aufwand beim Projektstart bringt große Zeiteinsparungen in den späten Projektphasen.**

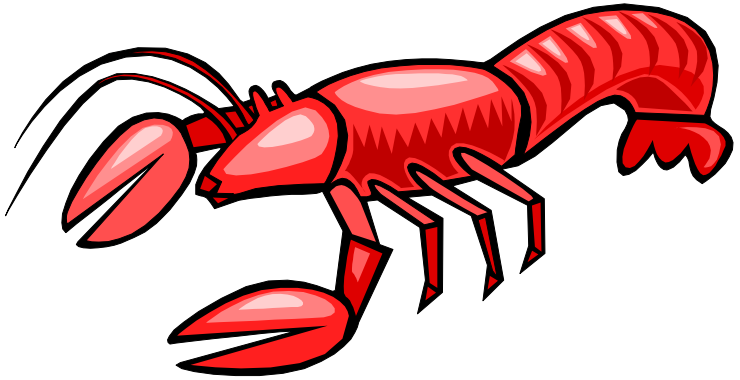
# Software-Struktur – Nachteil monolitischer Projekte

Ältere monolithische Projekte haben die Struktur eines Schalentiers.

Doch so ein Krebs ist hier ein schlechtes Vorbild:

Wird der Körper zu groß, muss ein neues Gehäuse her.

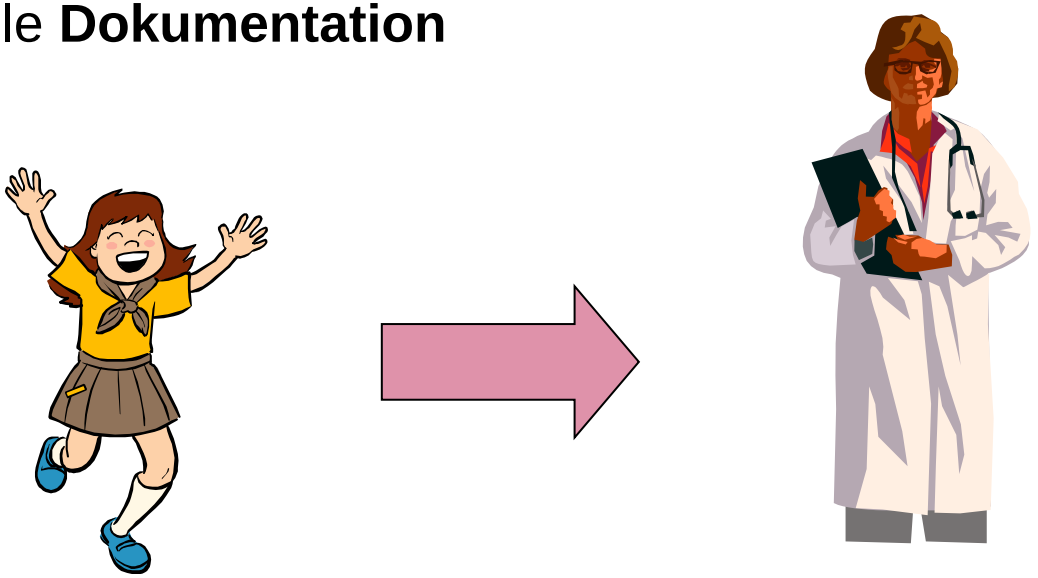
Für die Software heißt es:
**Bei Anforderungsänderungen muss der Code neu erstellt werden.**



# Gesundes Wachsen von Software

Vorbild Mensch: Das Skelett ist so gebaut, dass die Applikation mit den Anforderungen mitwächst.

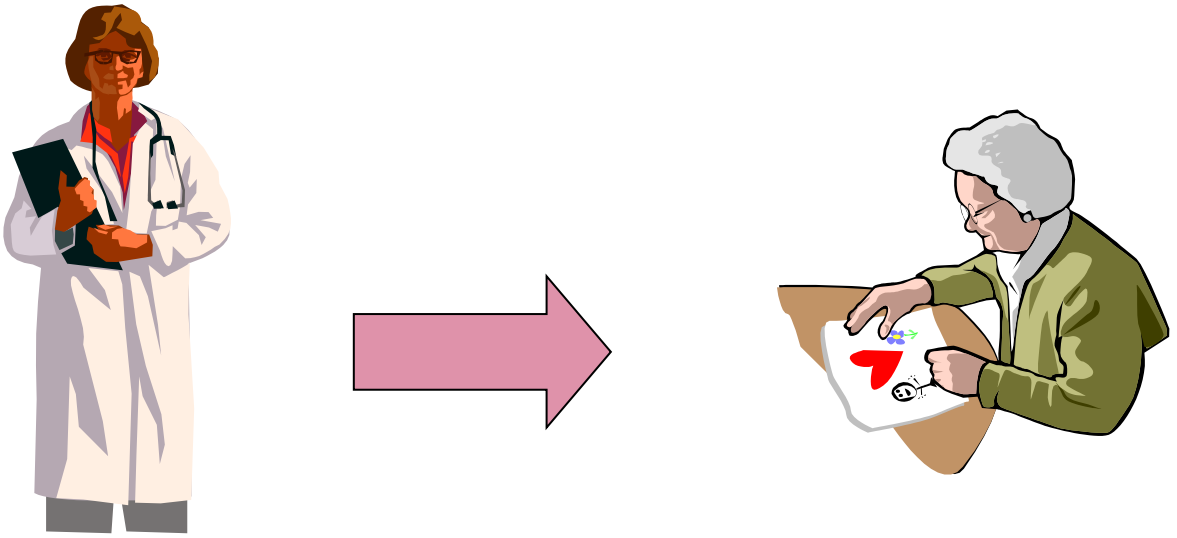
- **solide Architektur**
- Komponenten mit **definierten Schnittstellen**
- gute **Strukturierung** des **Codes**
- sinnvolle **Dokumentation**



# Gesund altern – regelmäßige Wartung der Software

Durch kontinuierliche Pflege und Gesundheitschecks wird die Software lange gesund gehalten.

- **Einsatz** von **Codier-** und **Dokumentationsrichtlinien**
- **regelmäßige Prüfungen**, ob gegen Regeln (Architektur, Codierung) verstoßen wurde



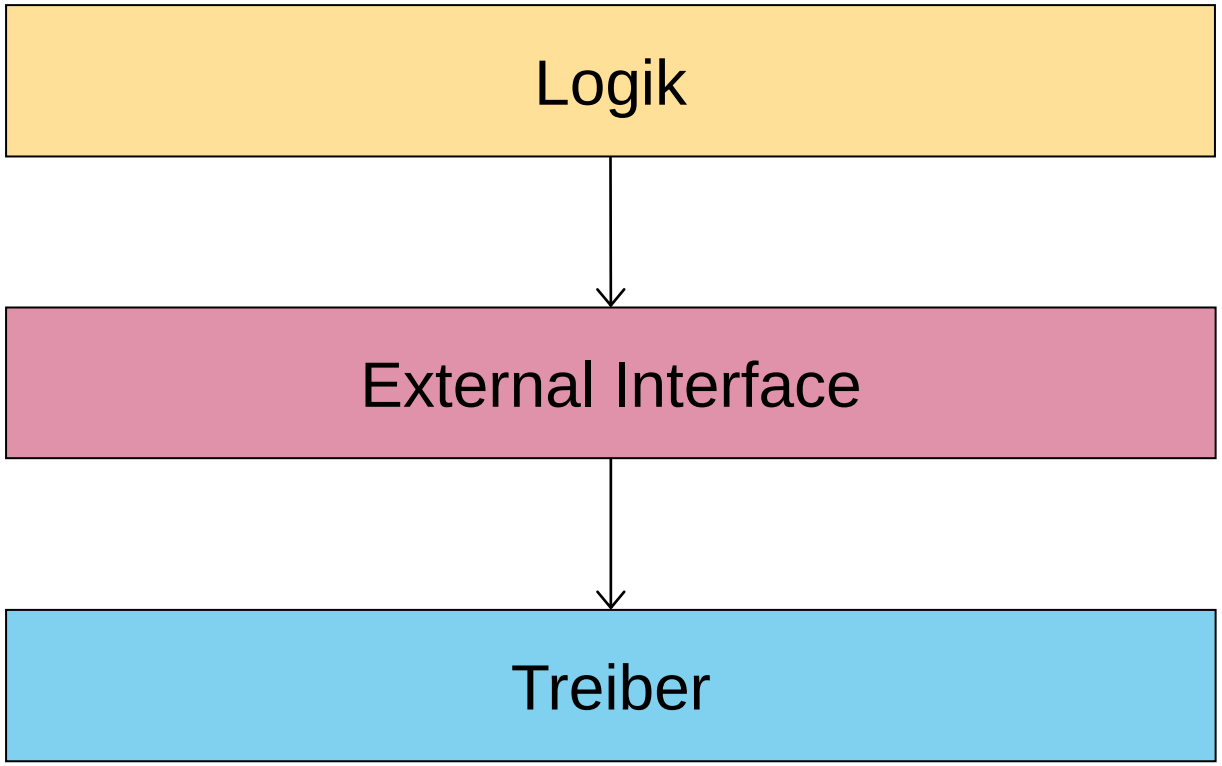
# Gesunder Körper – gute Software-Architektur

Wenige Regeln helfen schon, die Grundlagen für eine gute Architektur zu schaffen:

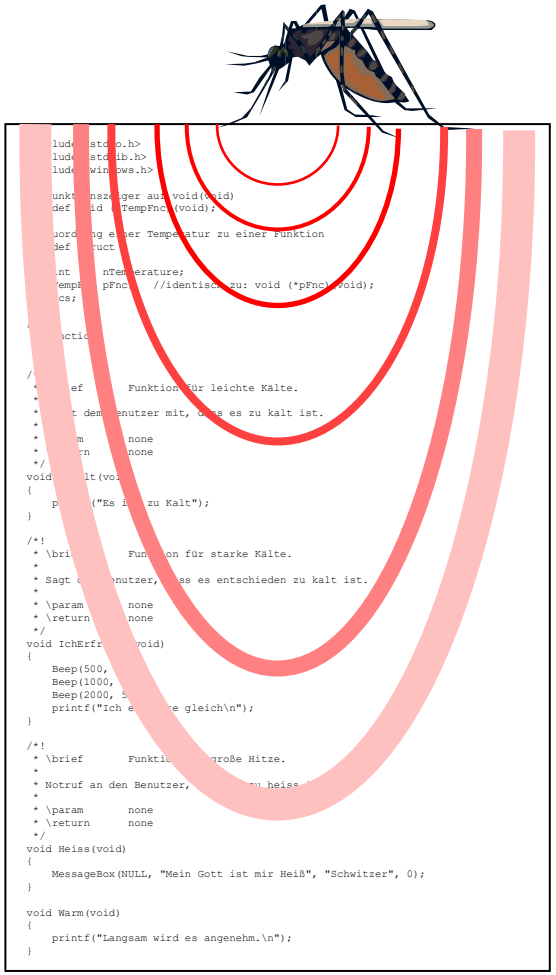
- **Modularisierung**
  - Einsatz von Schichtenmodellen
  - Bildung von Komponenten
- **Lose Kopplung**
  - Die Komponenten sind untereinander über wenige wohldefinierte Schnittstellen verbunden.
- **Hohe Kohäsion**
  - Enge Verzahnung von Codeteilen gibt es nur innerhalb der Komponenten.
- **Anwendung des Prinzips der "heilen" Welt**
  - Die Komponenten können problemlos aus dem Gesamtverbund gelöst und mittels geeigneter Treiber und Stubs einzeln getestet werden.

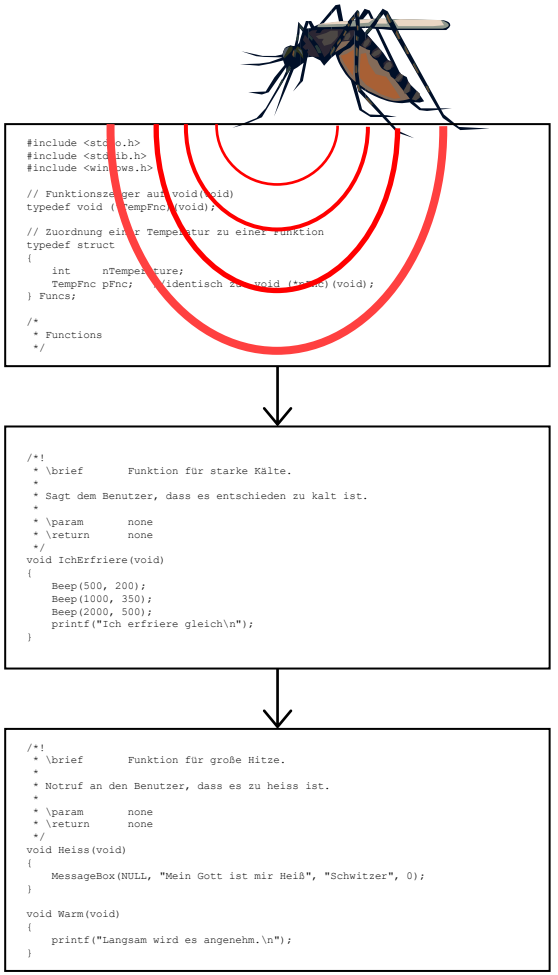
# 3-Schichten-Modell im Embedded Bereich

Die Komponenten werden den einzelnen Schichten zugeordnet.



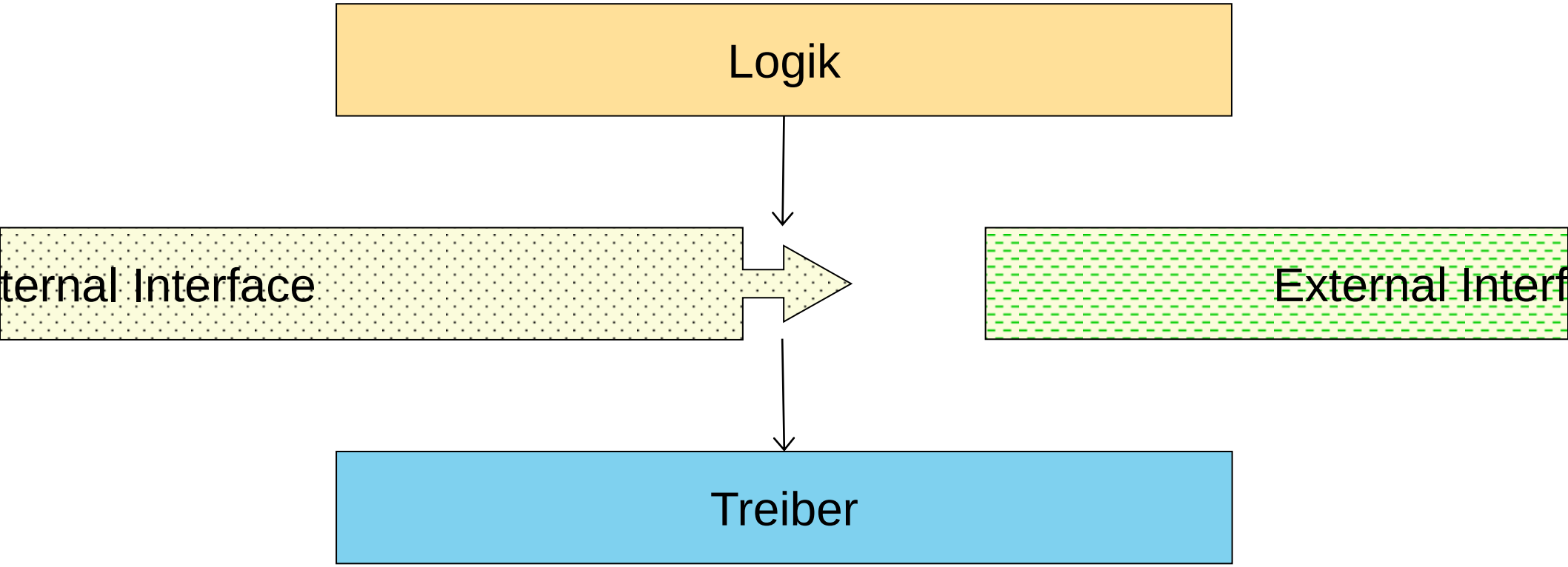
Durch **abgeschlossene Komponenten/Module** wirken sich Fehler meist nur lokal aus und nicht in der gesamten Applikation.





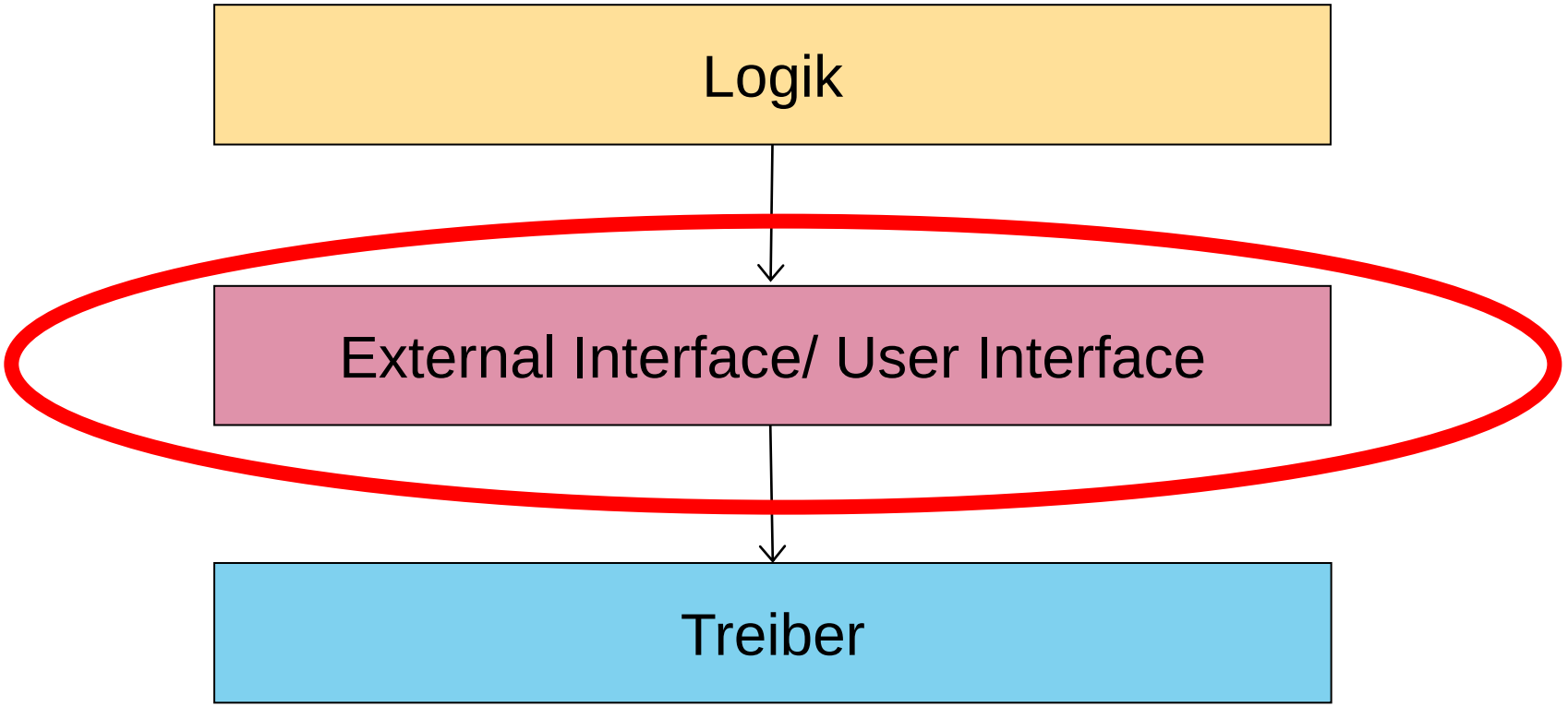
## **Lose Kopplung von Softwaremodulen**:

- bewirkt eine **einfache Austauschbarkeit der Komponenten** bzw.
- ermöglicht eine **Wiederverwendung** in Parallel- oder Folgeprojekten.



## **Hohe Kohäsion innerhalb eines Moduls**:

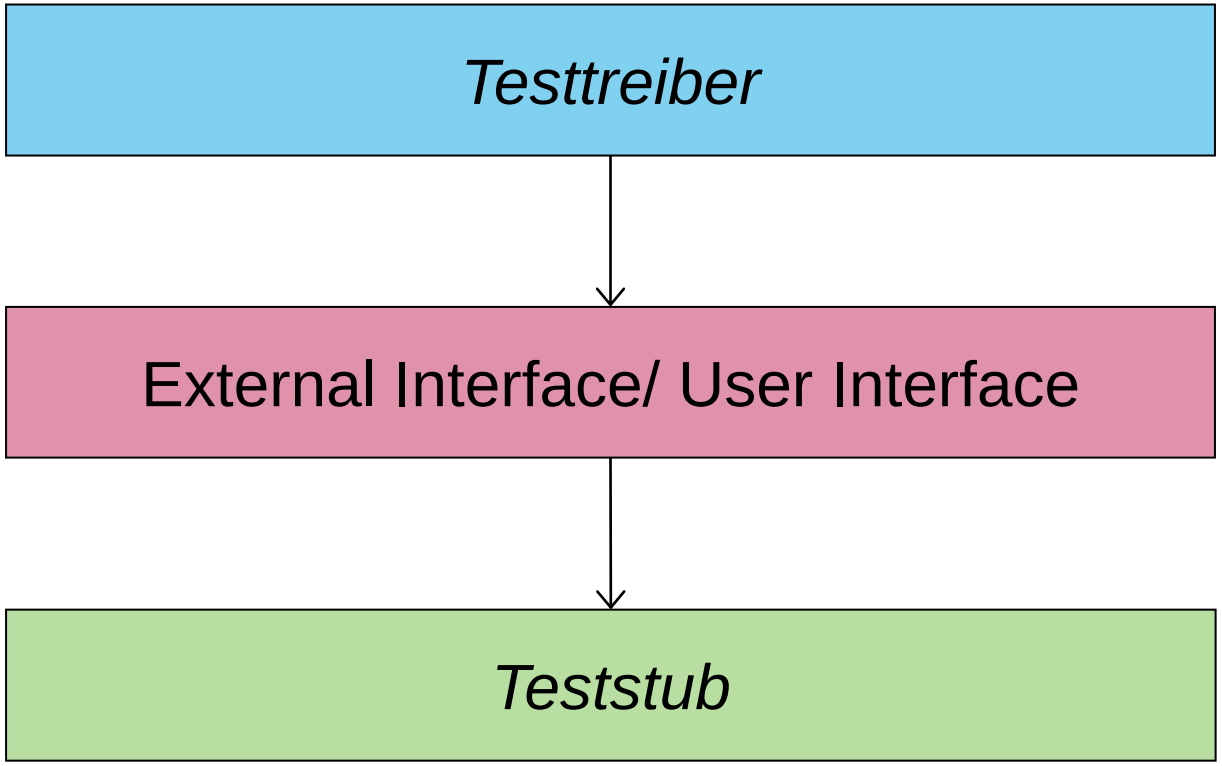
- **starke Verzahnung** von Code gibt es nur **innerhalb** des Moduls
- **nach außen** gibt es nur wenige und **einfache Schnittstellen**.



## Prinzip der "heilen" Welt:

→ **einzelne Software-Komponenten** sind **einzeln testbar**

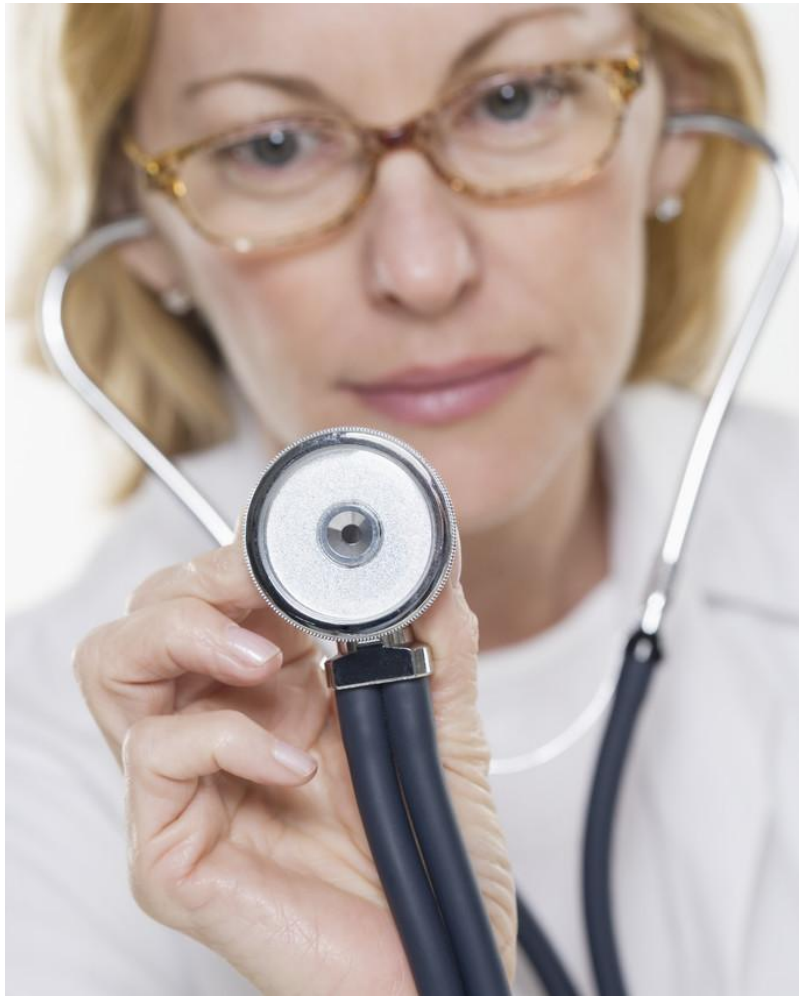
→ es muss nicht immer bis zum Systemtest gewartet werden.



# Gesunde Organe – sauberer Code

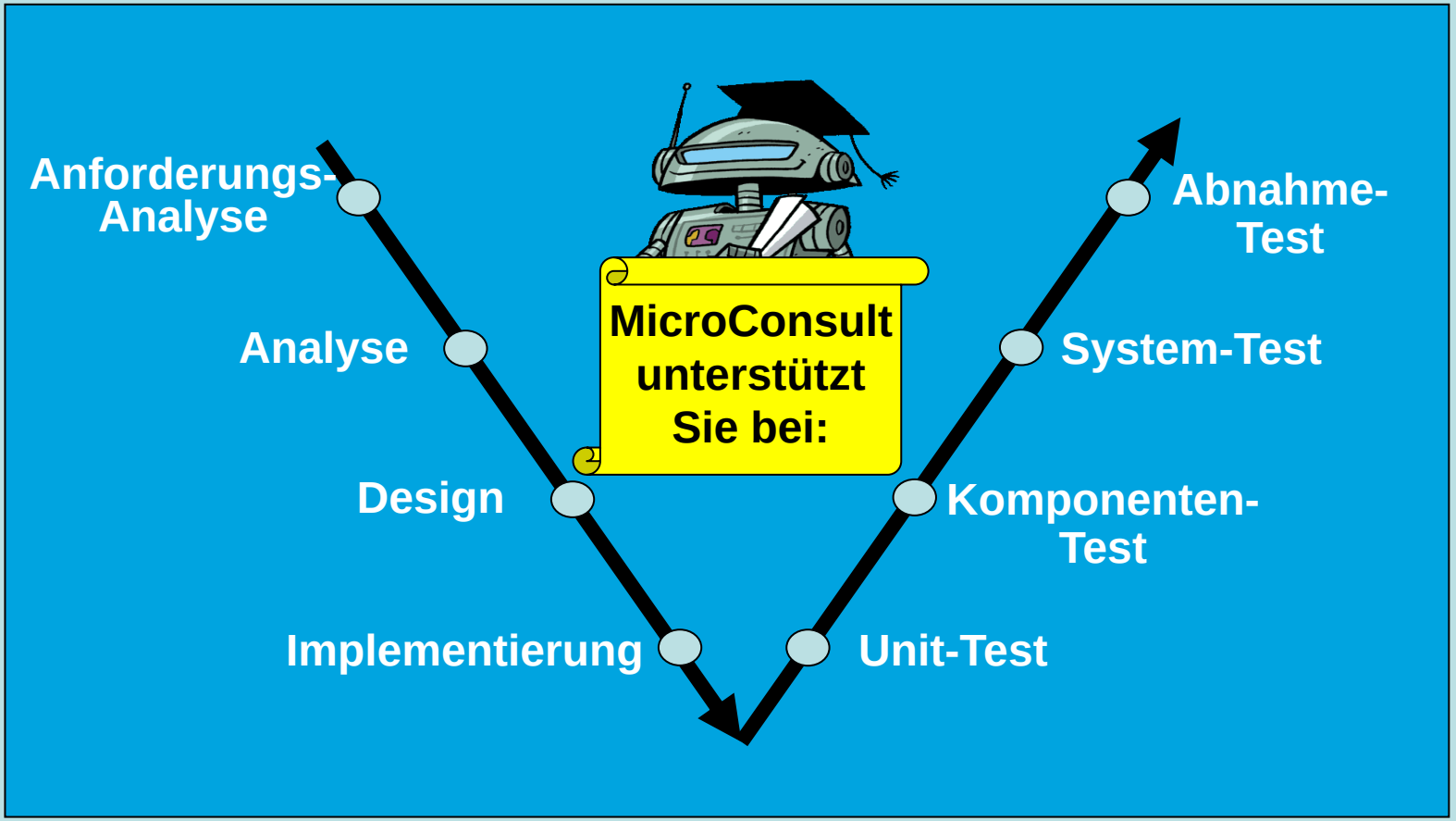
Durch den Einsatz von **Codier-Richtlinien** kann ein großer Schritt in Richtung gesunder Software getan werden.

- **Verbot kritischer Konstrukte**
  - Senkung der Fehleranfälligkeit
- **einheitliche Strukturierung**
  - Ordnung im "Kleinen" – Fehler werden schneller gefunden und können korrigiert werden.
- **einheitliche Namensgebung**
  - einfachere Fehlersuche in "fremdem" Code
  - schnellere Reviews
- **Dokumentationsrichtlinien**
  - schnellere Einarbeitung neuer Mitarbeiter
- **Einsatz von Dokumentations-Tools**
  - aktuelle Dokumentation nach jedem Build



- Eine Software ist gesund, wenn Körperbau (**Architektur**) und Organe (**Code**) gesund sind.
- **Regelmäßige Checks** erhalten die Gesundheit.
- Alle Mitarbeiter müssen auch mit den **Richtlinien** vertraut sein, um die benötigte **SW-Qualität** zu gewährleisten.

**Beratung, Training, Workshops, Coaching, Consulting**




**HW-/SW-Technologien, Tools, Methoden, Prozess, Team**

**Vortrag per E-Mail anfordern**: info@microconsult.de
Betreff: „Vortrag embedded world 2012“