Software-Produktmanagement

Ein Vortrag von

Harry M. Sneed

SoRing Kft., Budapest

für die TU Dresden

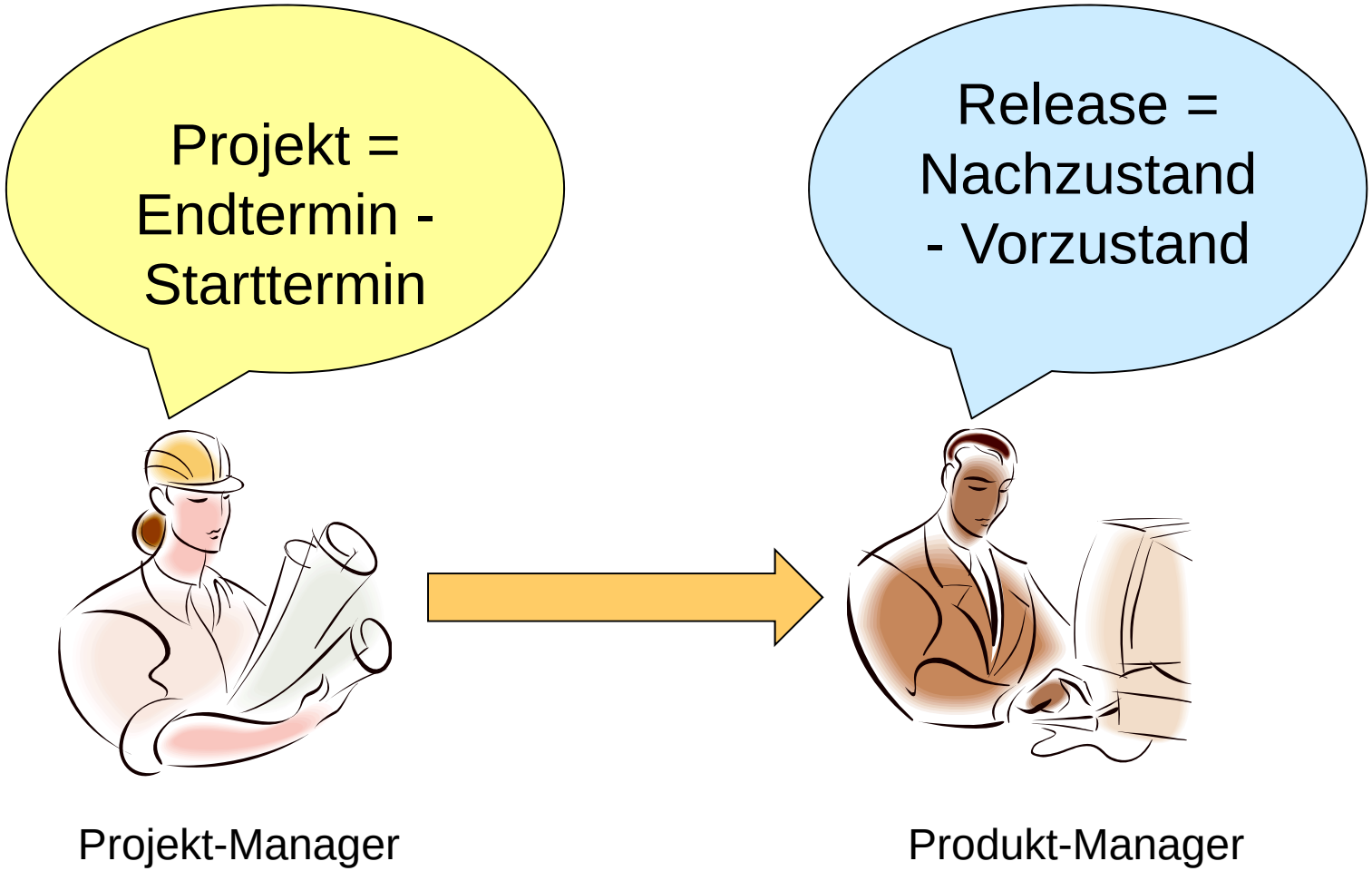
11. Januar 2015

Vom Projektmanagement zum Produktmanagement

- In der klassischen Softwareentwicklung wurde zu sehr auf das Projekt fokussiert. Alles drehte sich um die Aktivitäten – Phasen, Aufgaben, Funktionen, Rollen, usw. Vorgehensmodelle wie Wasserfall, V-Modell, RUP, SPICE und evolutionäre Entwicklung stand im Mittelpunkt der Betrachtung. Das Objekt der Tätigkeit wurde zu oft vernachlässigt.

- Neuerdings steht wieder das Produkt im Vordergrund, bzw. das Objekt. Produkte entstehen, wachsen, reifen und sterben irgendwann. Das Produkt muss konzipiert, spezifiziert, modelliert, kodiert, integriert, getestet und gewartet werden. Das alles zu steuern und das Produkt nach außen zu vertreten ist Aufgabe des Produktmanagements.

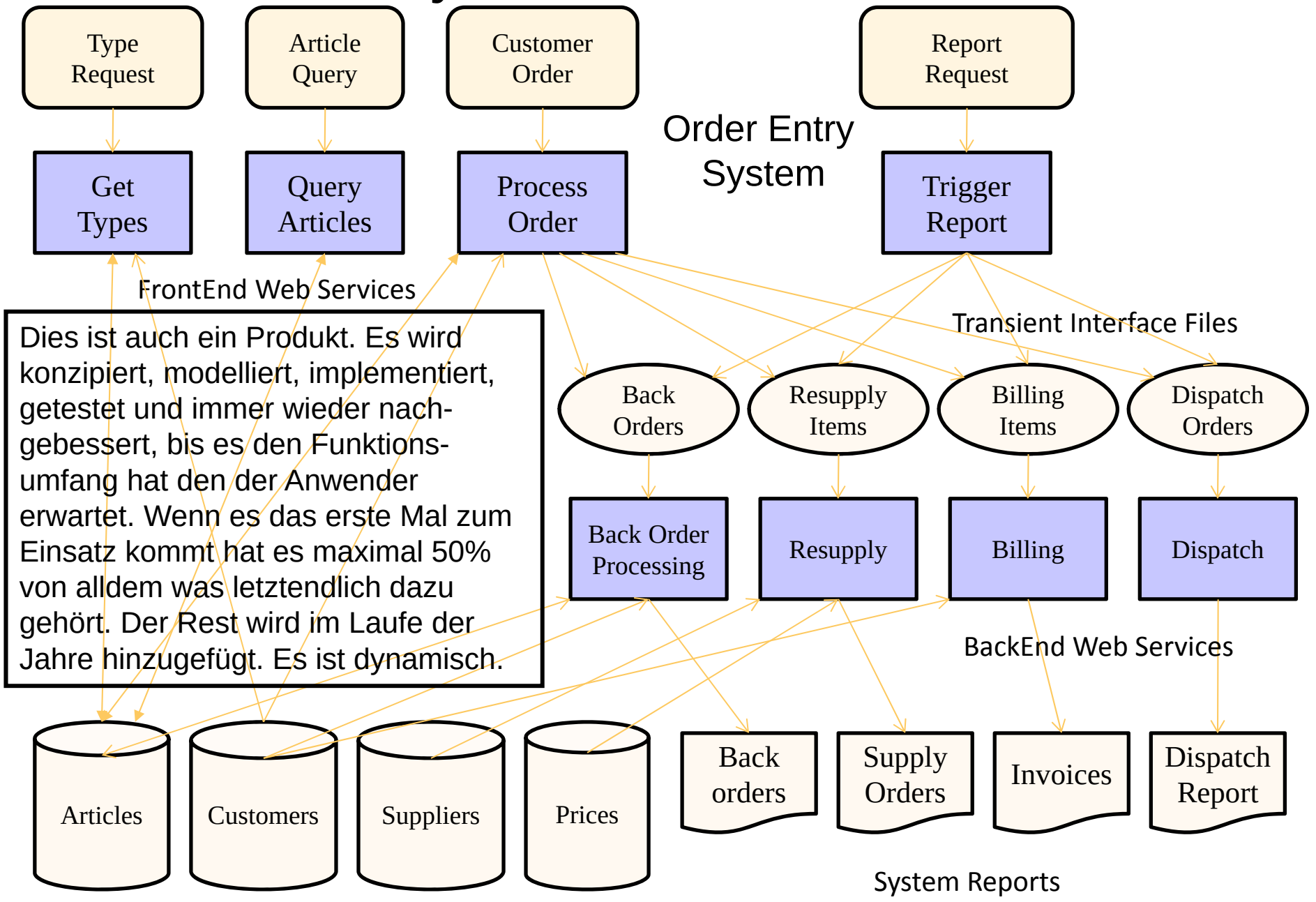
Weg vom Projektmanagement Hin zum Produktmanagement



Ein statisches Produkt

Dies ist ein Produkt. Es wird konzipiert, modelliert, simuliert, ausprobiert, gebaut und getestet. Später wird es repariert, nachgebessert und gewartet. Wenn es das erste Mal zum Einsatz kommt hat es schon mindestens 95% von alldem was letztendlich dazu gehört. Man weis vom Anfang an, wozu es benutzt wird und kann den Funktionsumfang bestimmen. Die Wartung dient dazu den original Zustand zu erhalten und geringfügig nachzubessern. Es ist ein statisches Produkt.

Ein dynamisches Produkt

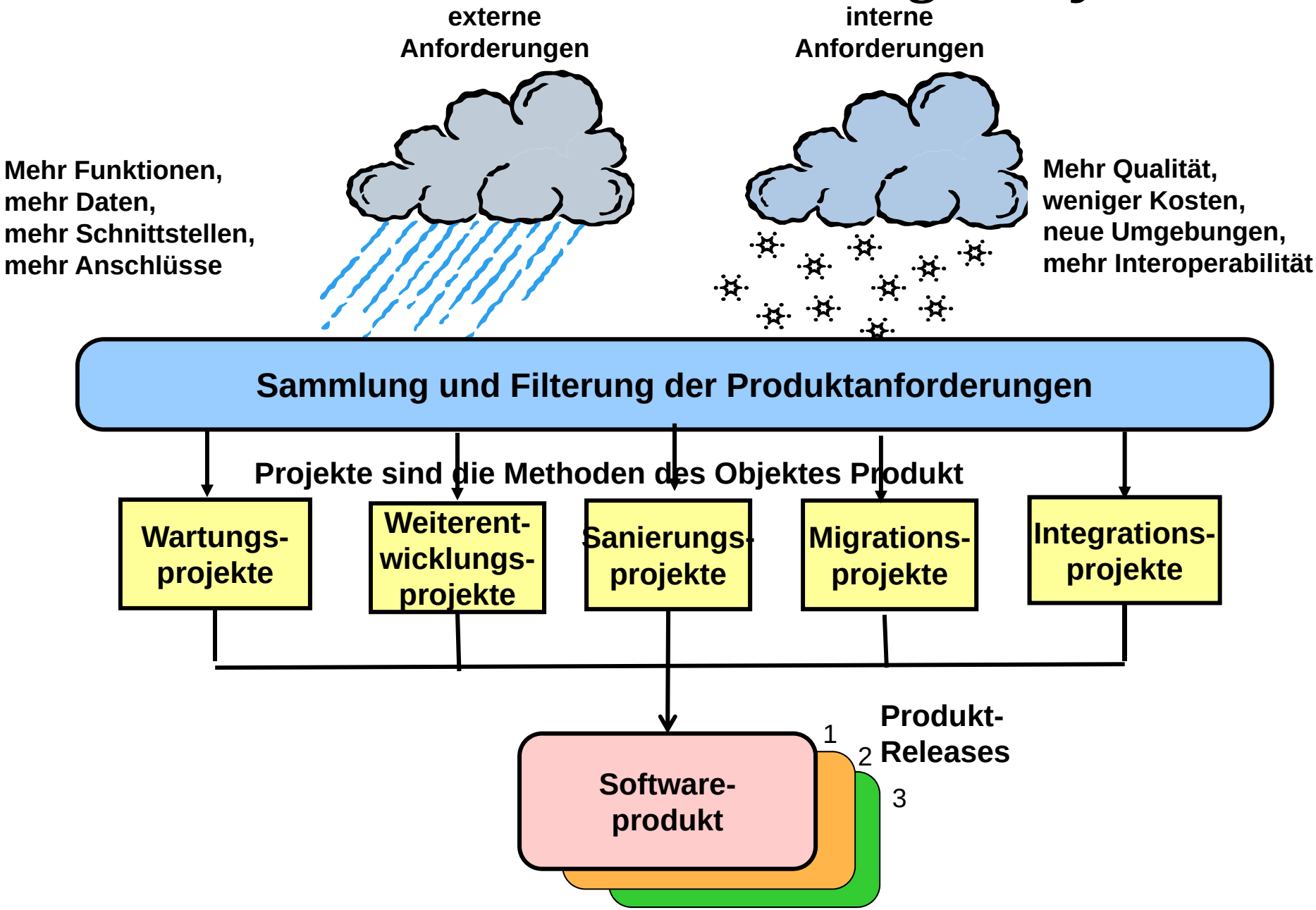


Dies ist auch ein Produkt. Es wird konzipiert, modelliert, implementiert, getestet und immer wieder nach-gebessert, bis es den Funktions-umfang hat den der Anwender erwartet. Wenn es das erste Mal zum Einsatz kommt hat es maximal 50% von alldem was letztendlich dazu gehört. Der Rest wird im Laufe der Jahre hinzugefügt. Es ist dynamisch.

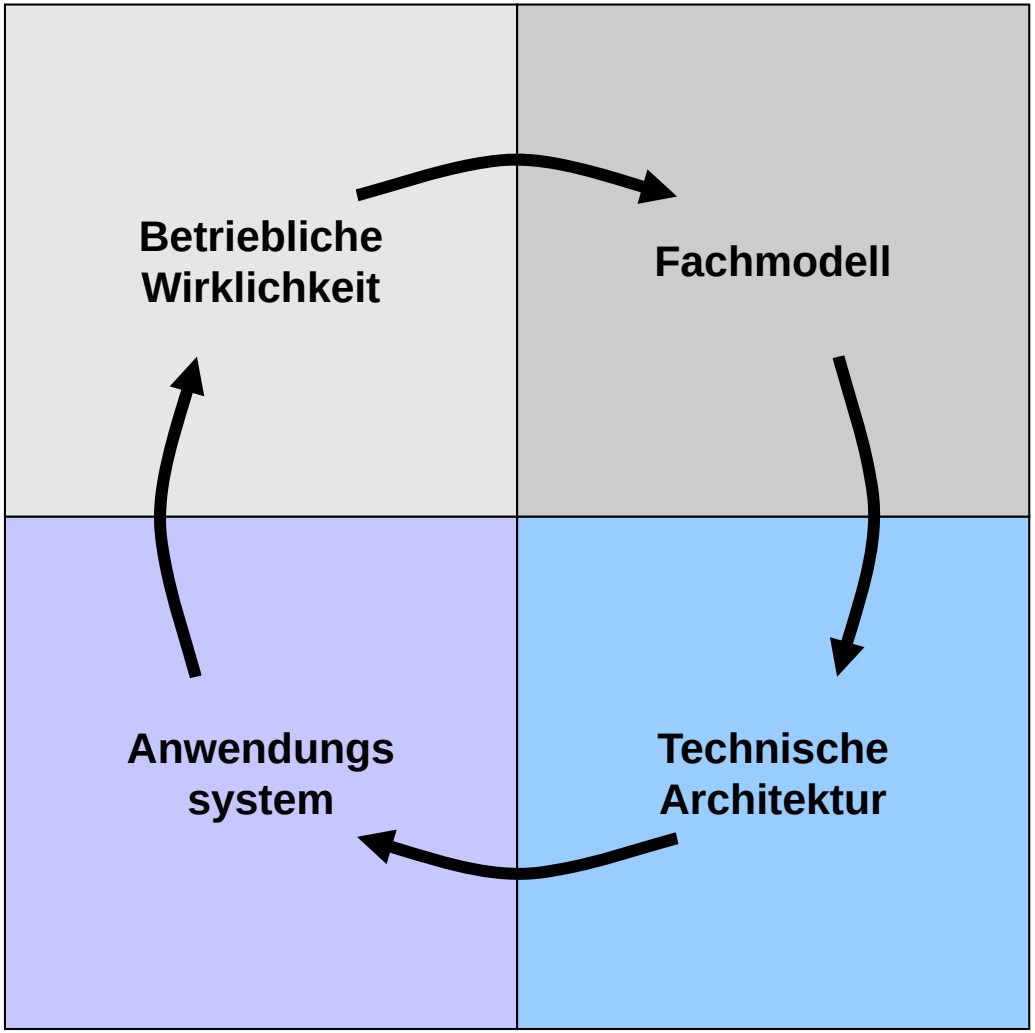
Software Produktmanagement ist das Management dynamischer Produkte

- IT-Softwaresysteme sind im Gegensatz zu embedded Realtime, bzw. eingebaute, Gerätesteuerende Systeme höchst dynamisch.
- Sie müssen deshalb vom Anfang an für dauerhafte Evolution konzipiert und konstruiert werden.
- Es muss immer möglich sein bestehende Komponente zu ändern bzw. auszutauschen ohne andere Komponente zu beeinflussen.
- Es muss auch möglich sein neue Komponente jederzeit einzubauen ohne negative Auswirkung.
- Ein dynamisches Produkt ist vom Anfang an auf Wandlung ausgerichtet.
- Deshalb ist das dynamisches Produktmanagement grundsätzlich anders.

Ein Produkt hat viele nebenläufige Projekte



Produktevolution als permanente Nachbesserung



Urmodell des Software-Lebenszyklus

Gott der Erhaltung

Vishnu
der Maintainer

Gott der
Schöpfung

Brahma
der Developer

Evolution

Erhaltung

Geburt

Tod

Wiedergeburt = Recycling

Entwicklung

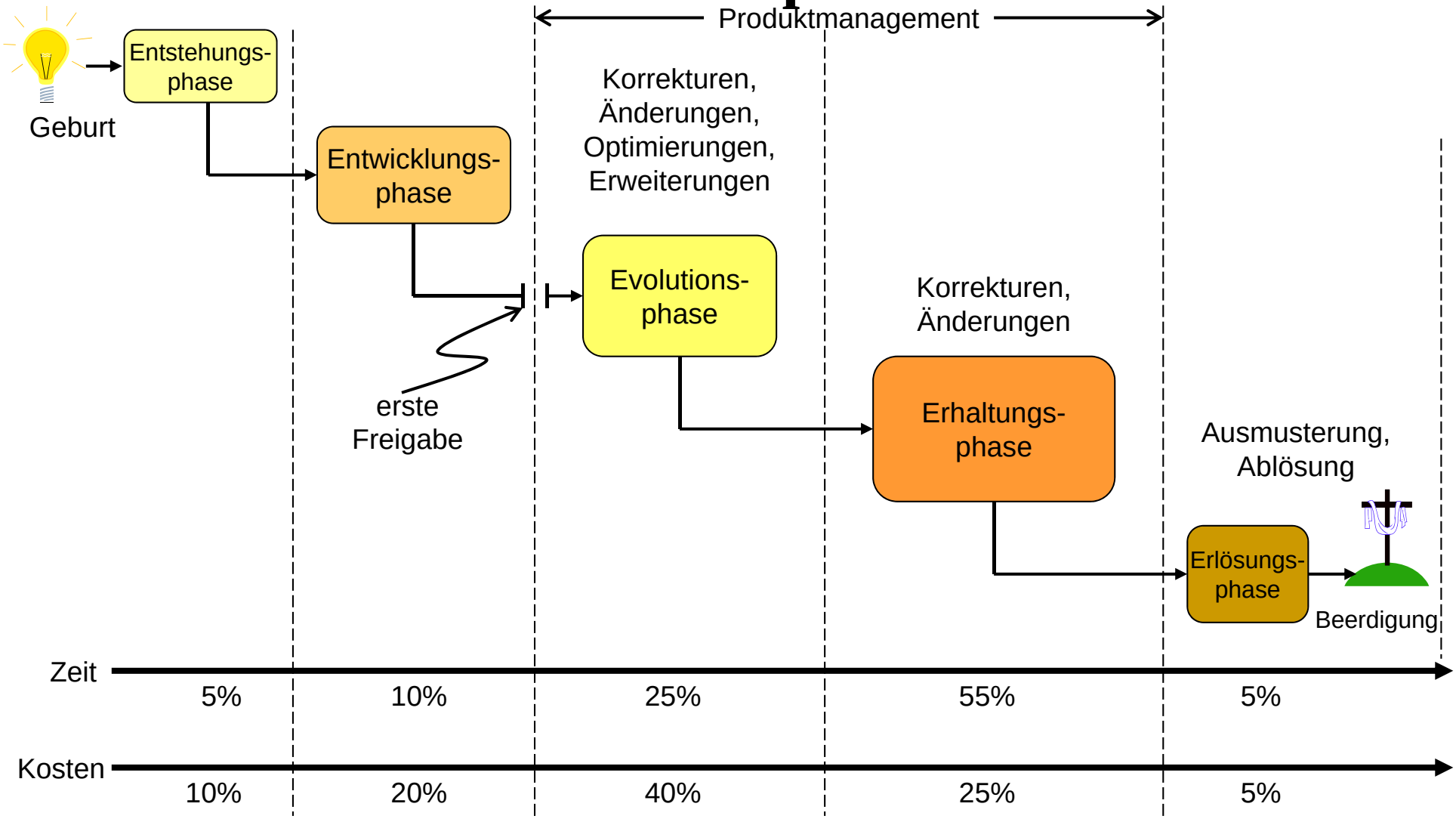
Erlösung

Gott der
Wiederverwendung

Shiva
der Reengineer

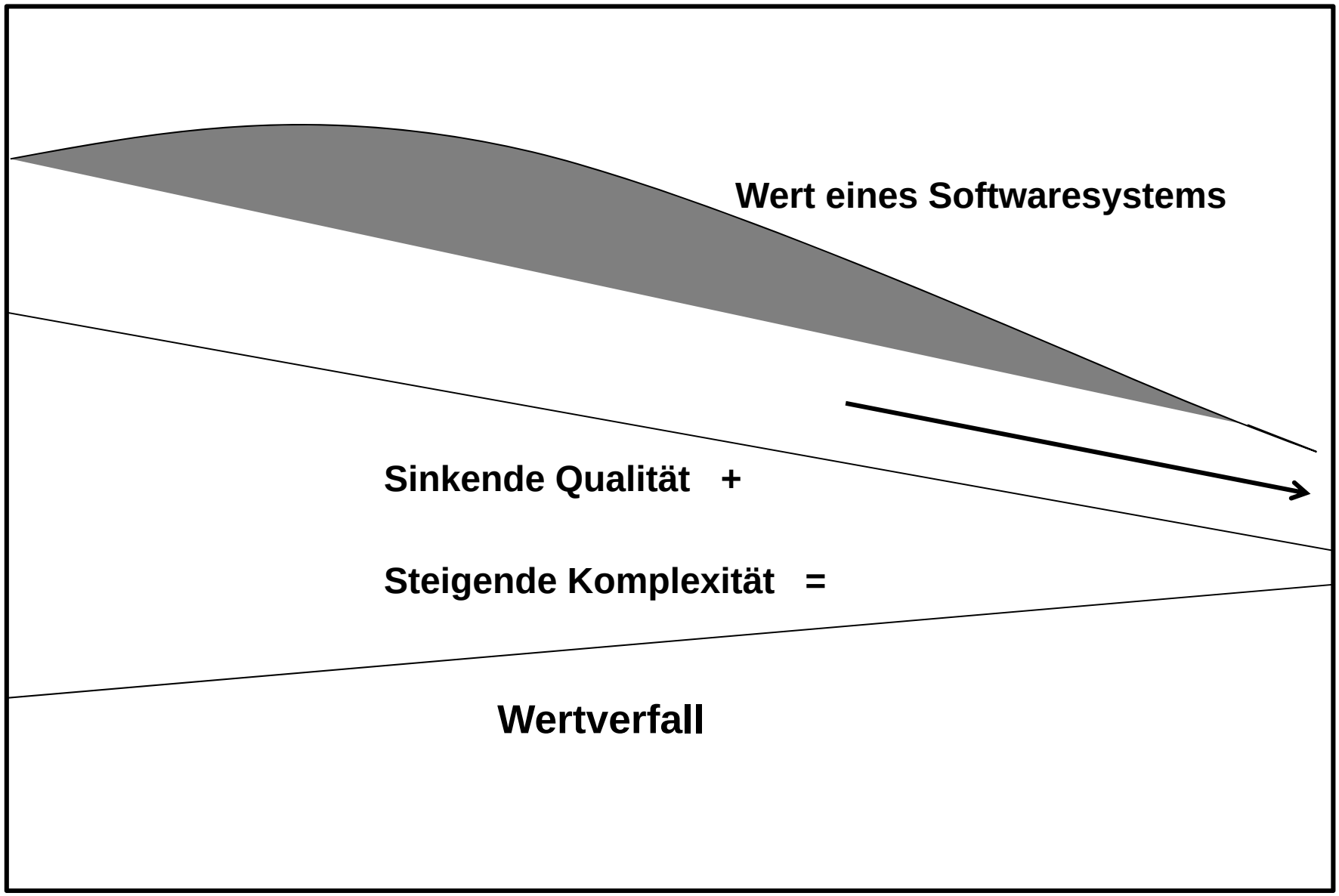
Hindu-Dreieinigkeitslehre

Fünf Phasen im Lebenszyklus eines Softwareproduktes



Nach Bennett und Rajlik

Wertverfall eines Softwareproduktes



Ziele eines Software Produktmanagers

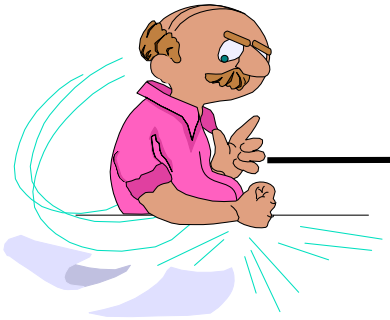
- **Gewährleistung der Betriebsbereitschaft**
  - durch sofortige Problembehebung
  - durch Backup-Lösungen
- **Stetige Weiterentwicklung**
  - durch die Annahme und Abarbeitung neuer Anforderungen
  - durch neue Releases mit zunehmender Funktionalität
- **Verhinderung des Wertverfalles**
  - durch ständige Überwachung des Produktzustandes
  - durch regelmäßige Sanierung (Refactoring)
  - durch wiederholte Regressiontests
  - durch Fortschreibung der Anforderungsdokumente, der Entwurfsmodelle und der Testfälle (Konsistenz)

Ziel 1 = Gewährleistung der Betriebsbereitschaft

Trouble Shooting Team

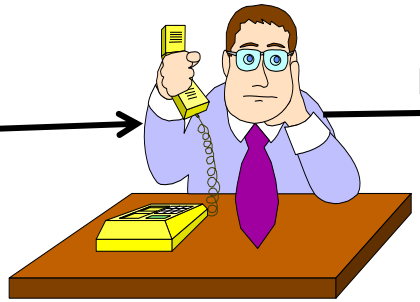
Repariert letztes Release

Problem Management



Benutzer

Incident Report



Help Desk

Probleme

Fehler Daten bank

Problem Historie

Problem Report

Vorfallmeldungen werden vom Help Desk in Fehlermeldungen umgesetzt und an das Trouble Shooting Team weiter gegeben

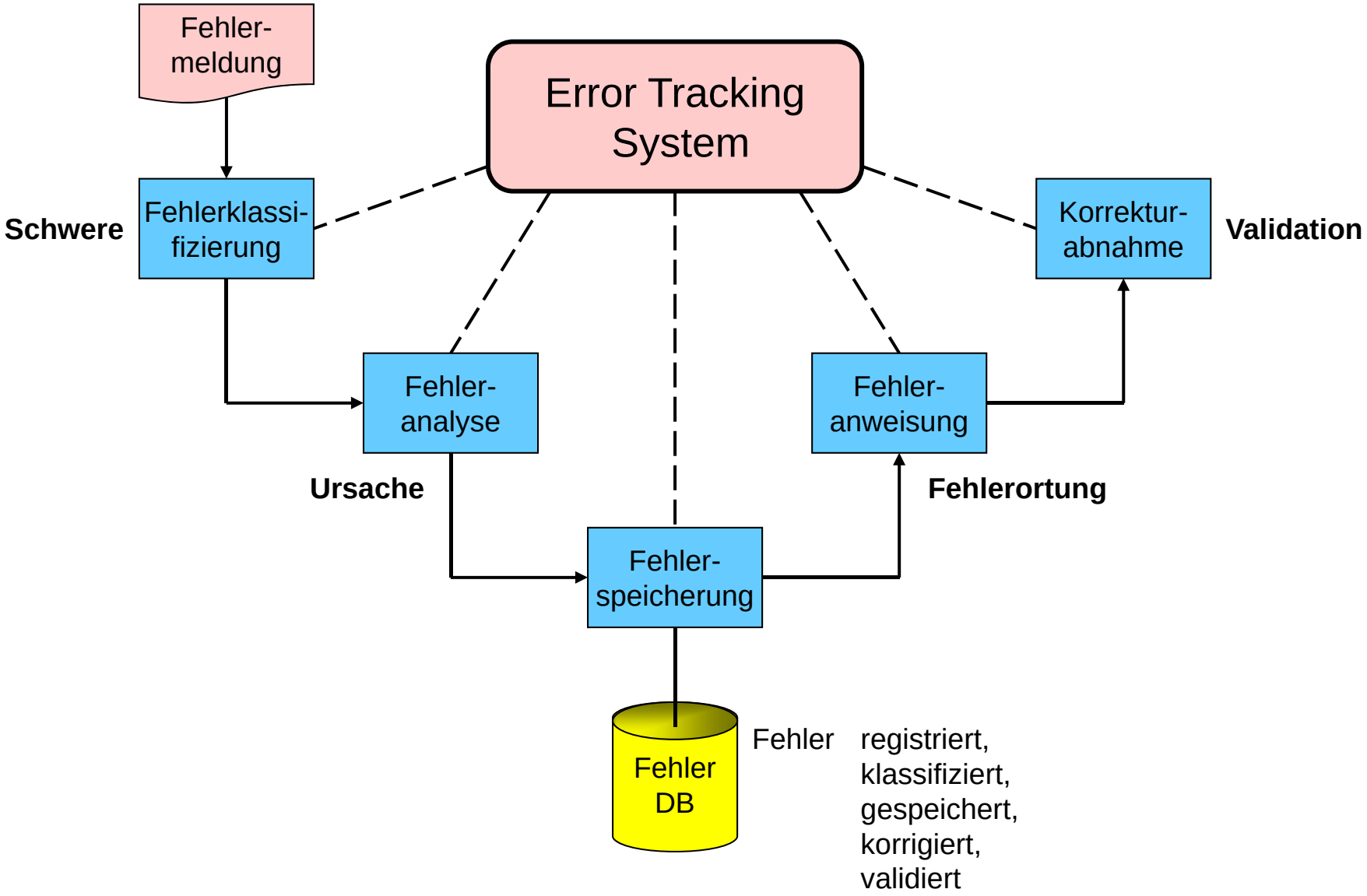
Produktmanager

Errors

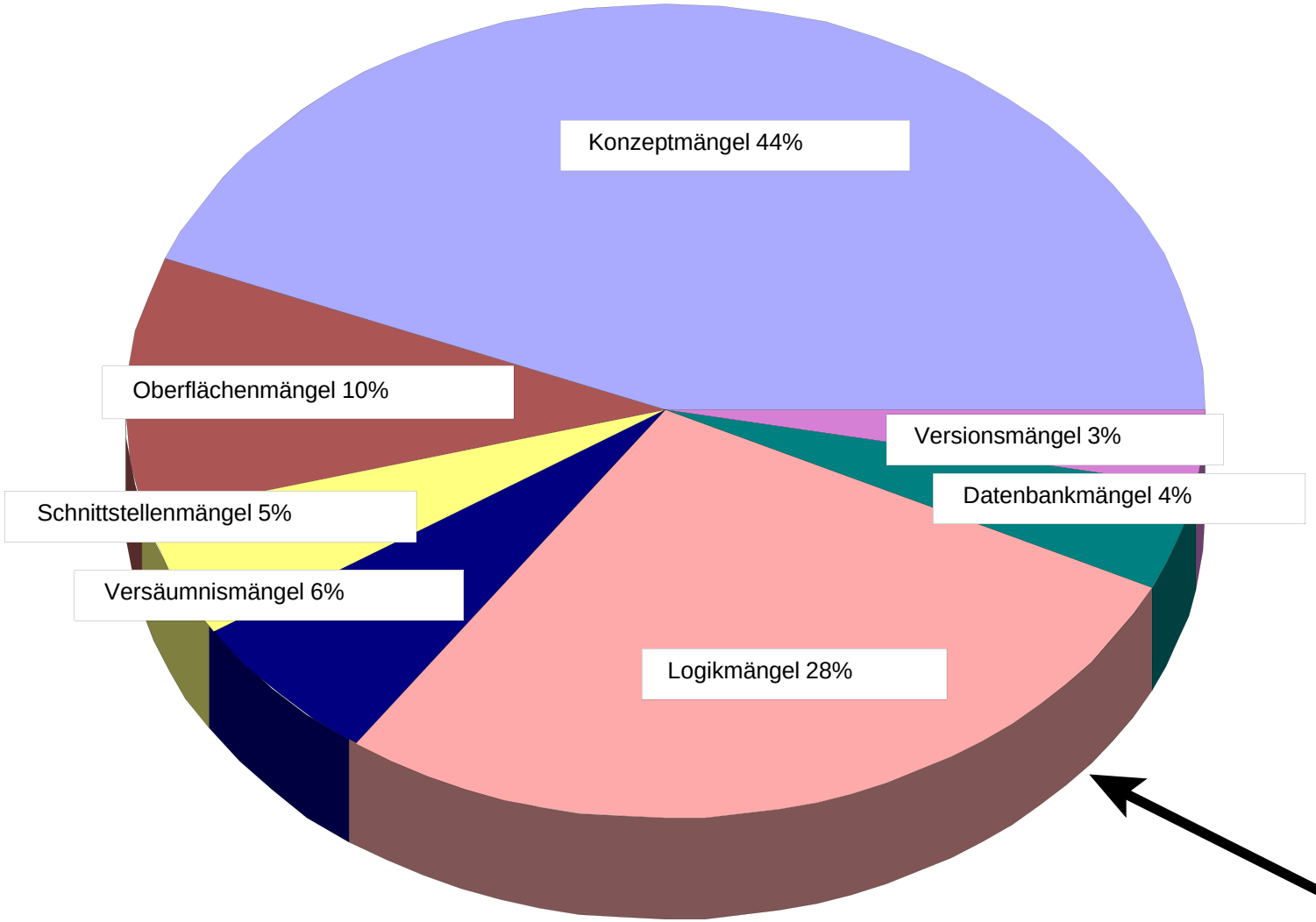
Korrektur bericht

Maintenance Engineer

Fehlerverfolgung

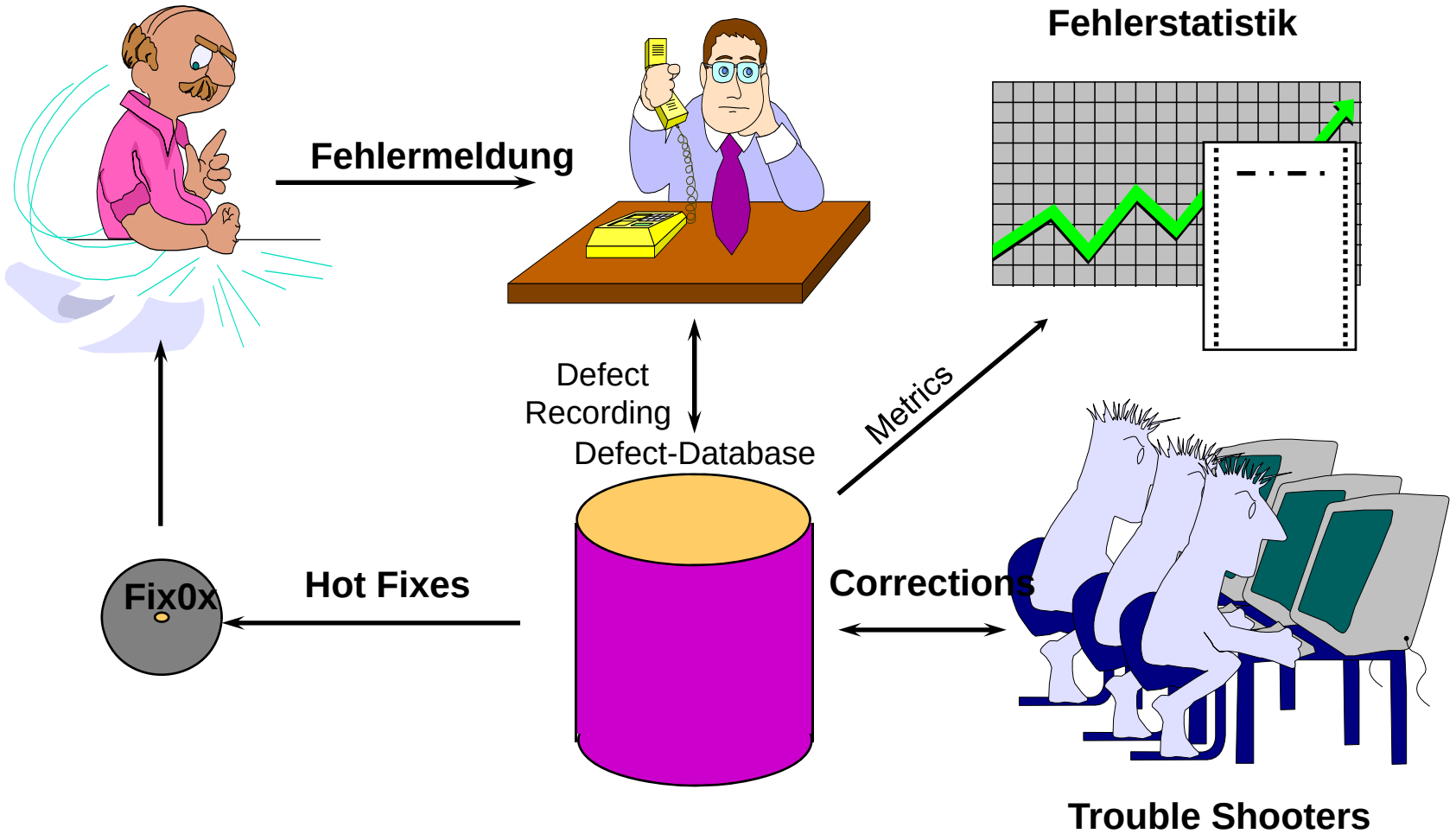


Fehleranalyse



Fehlerverteilung nach Fehlertyp

Fehlervoraussage



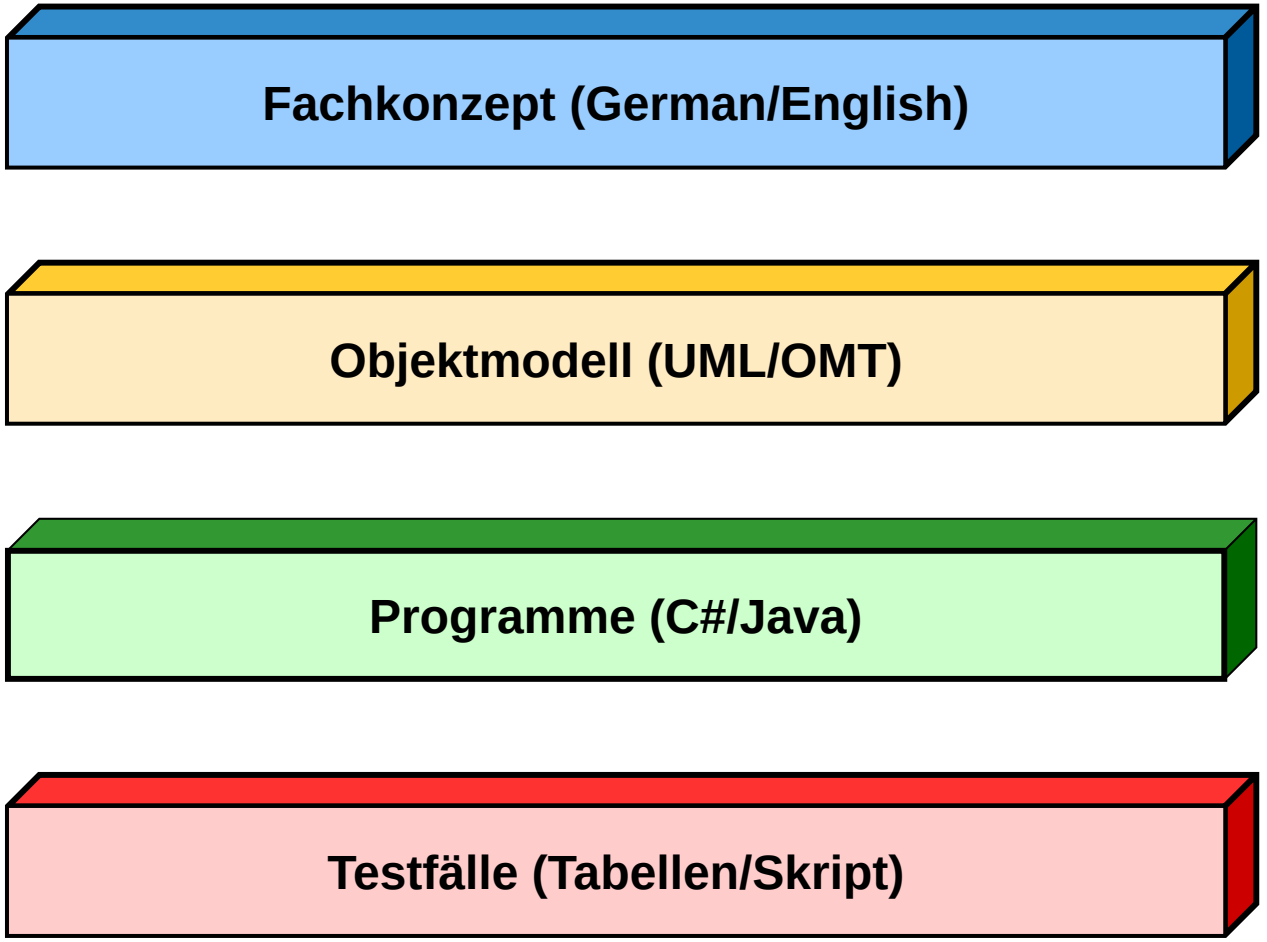
Erwartete Fehler = Letzte Fehlerdichte * Neue Systemgröße * (1- Testüberdeckung)

Ziel 2 = Stetige Weiterentwicklung

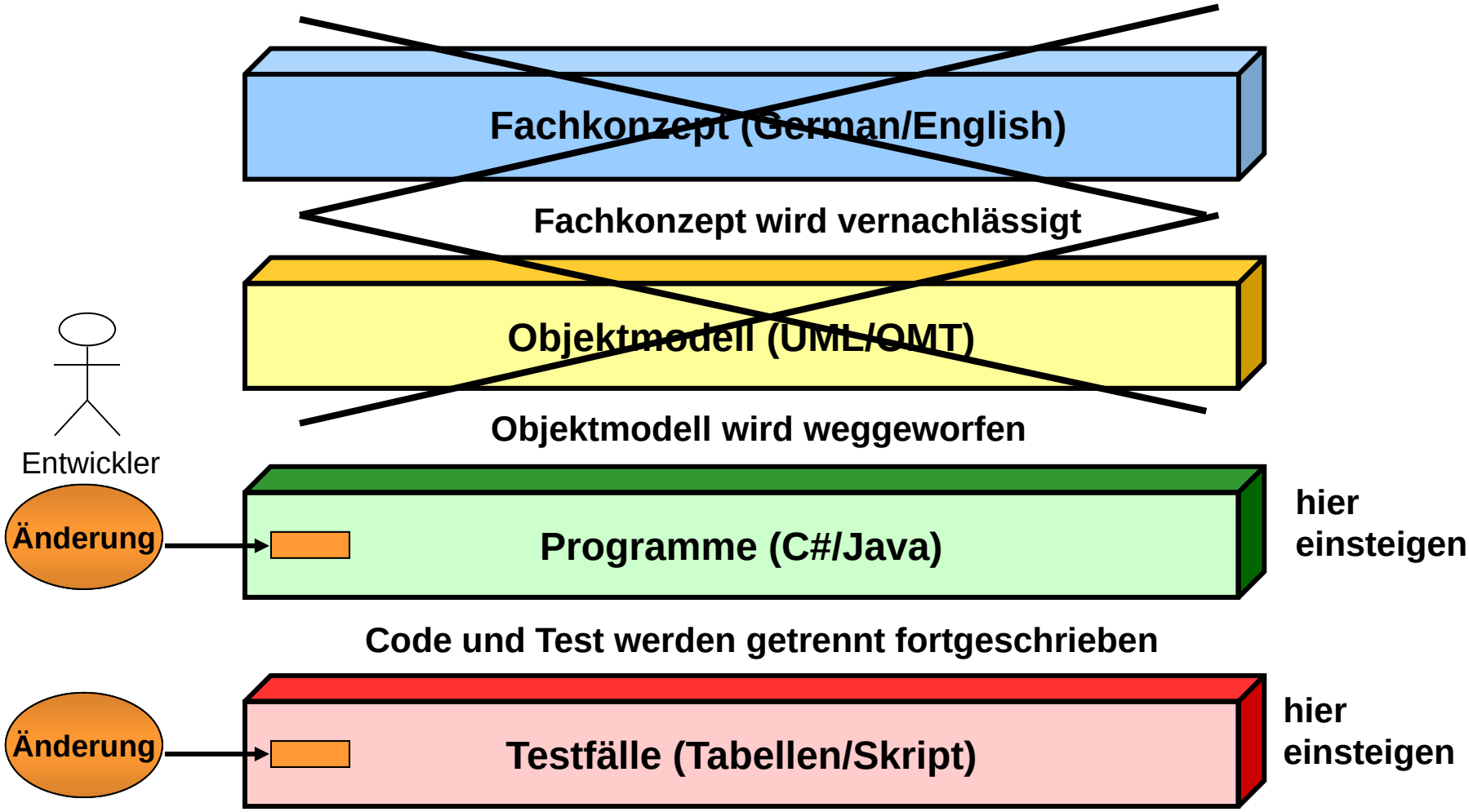
Development Team

Bereitet nächstes Release vor

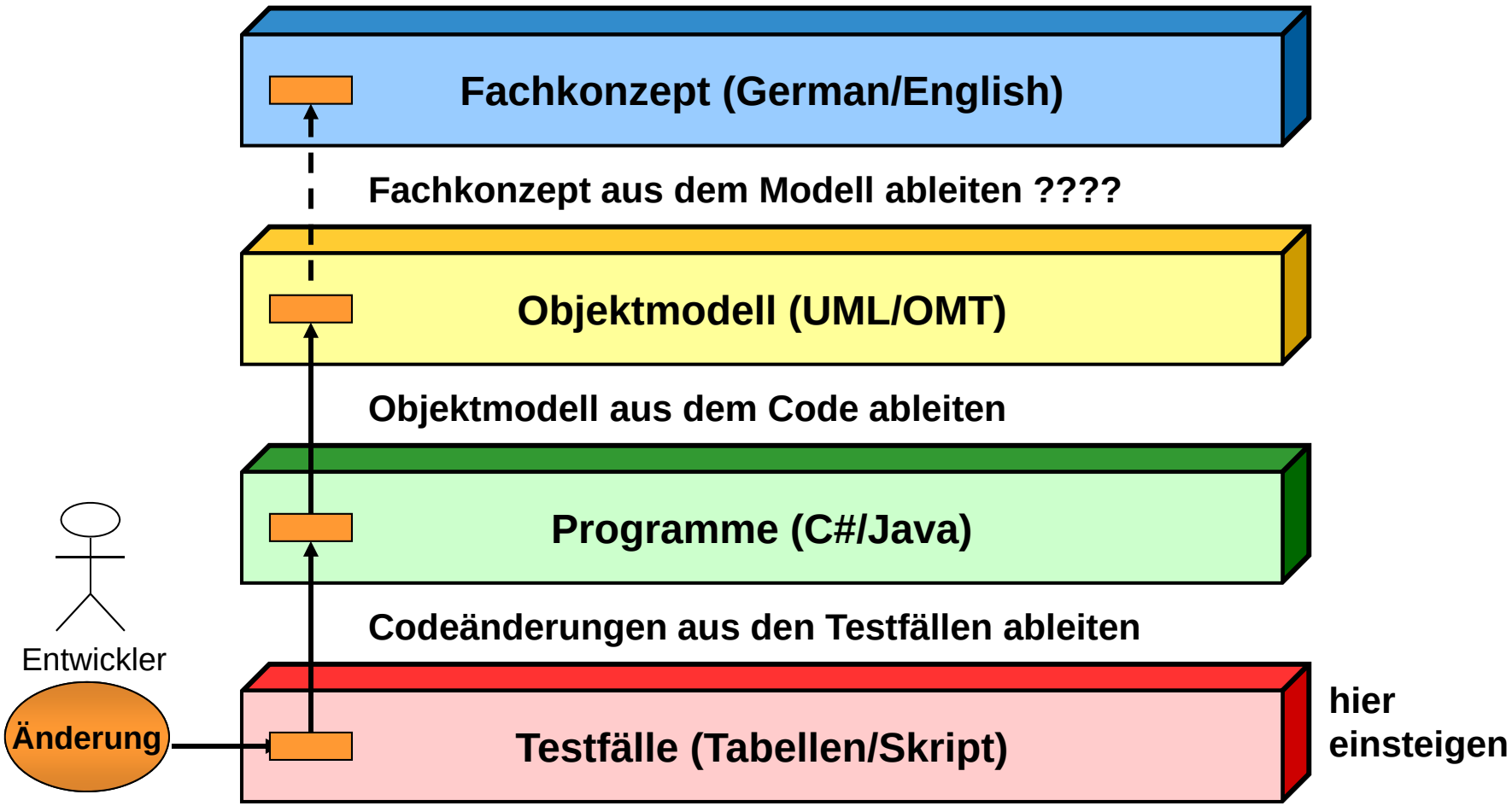
Semantische Ebenen eines Softwareprodukts



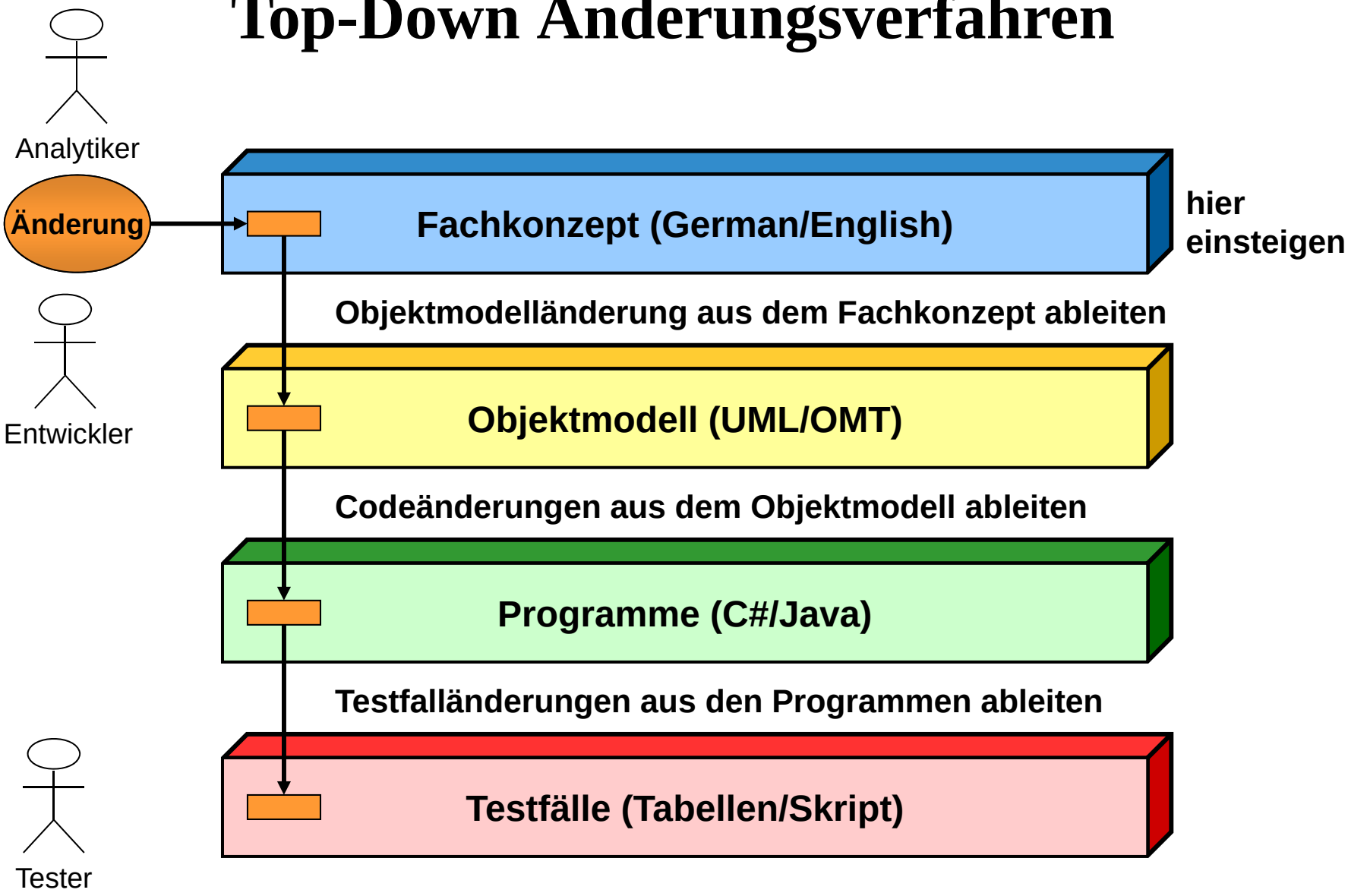
Konventionelles Änderungsverfahren



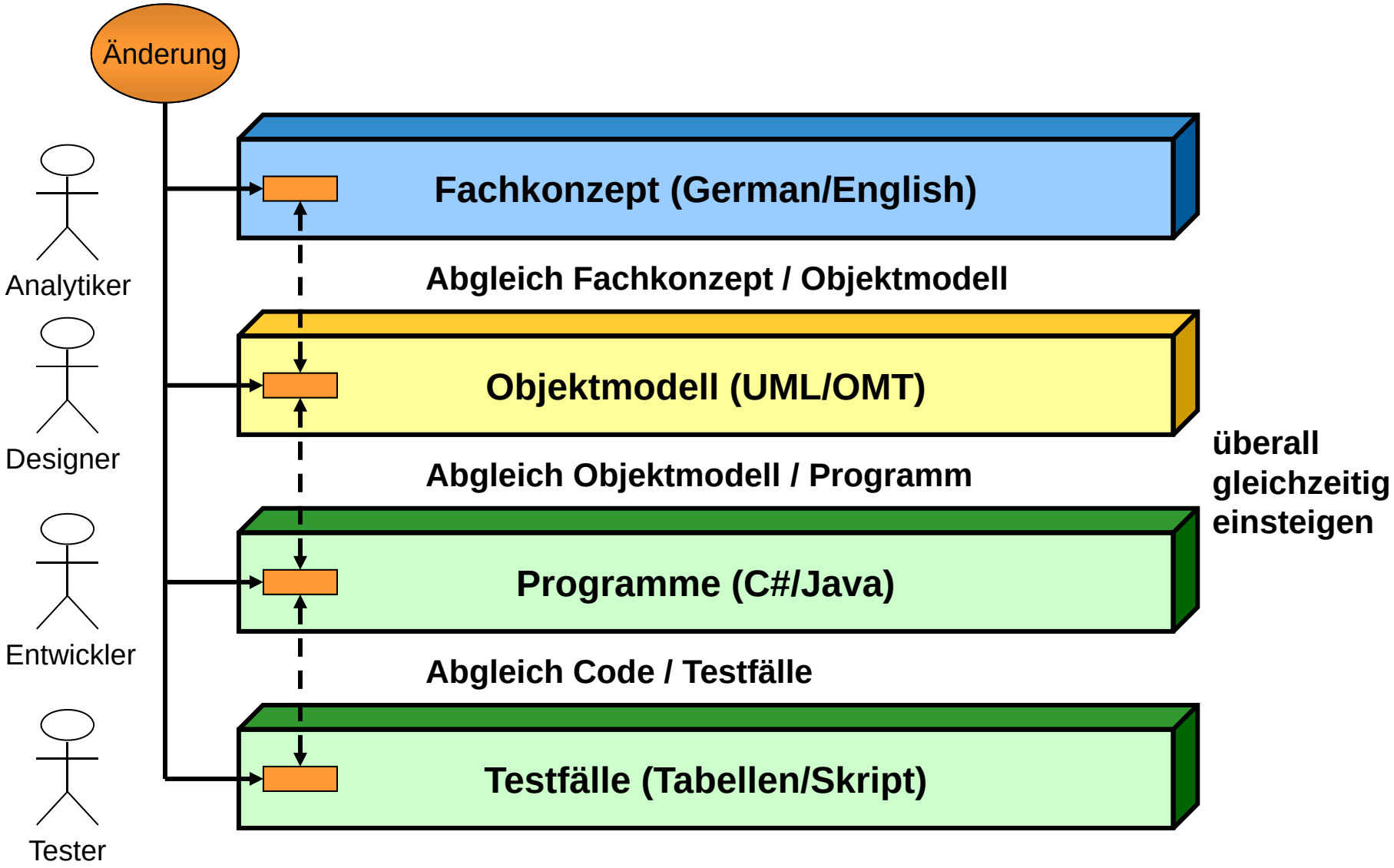
Bottom-Up Änderungsverfahren



Top-Down Änderungsverfahren



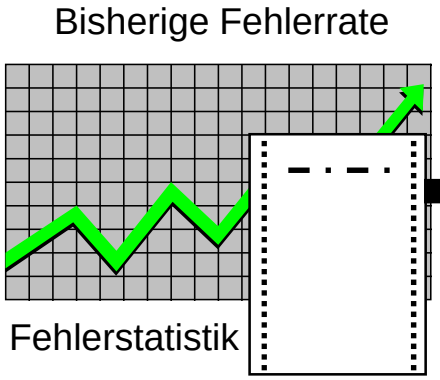
Paralleles Änderungsverfahren



Alternative Ansätze zur Steuerung der Software Evolution

- **Der Top-Down modellgetriebener Ansatz**
  generiert neuen Code aus dem geänderten Modell
- **Der Bottom-Up Codegetriebener Ansatz**
  generiert neues Modell aus dem geänderten Code
- **Der Testgetriebener Ansatz**
  ändert den Code aufgrund der neuen Testfälle
- **Der anforderungsgetriebener Ansatz**
  steuert gleichzeitige Fortschreibung des Codes und des Tests

Schätzung neuer Releases



Schätzung der Reparaturkosten aufgrund der Fehlerstatistik

€ **Kosten der Fehlerbehebung**

Bestehende Codemenge

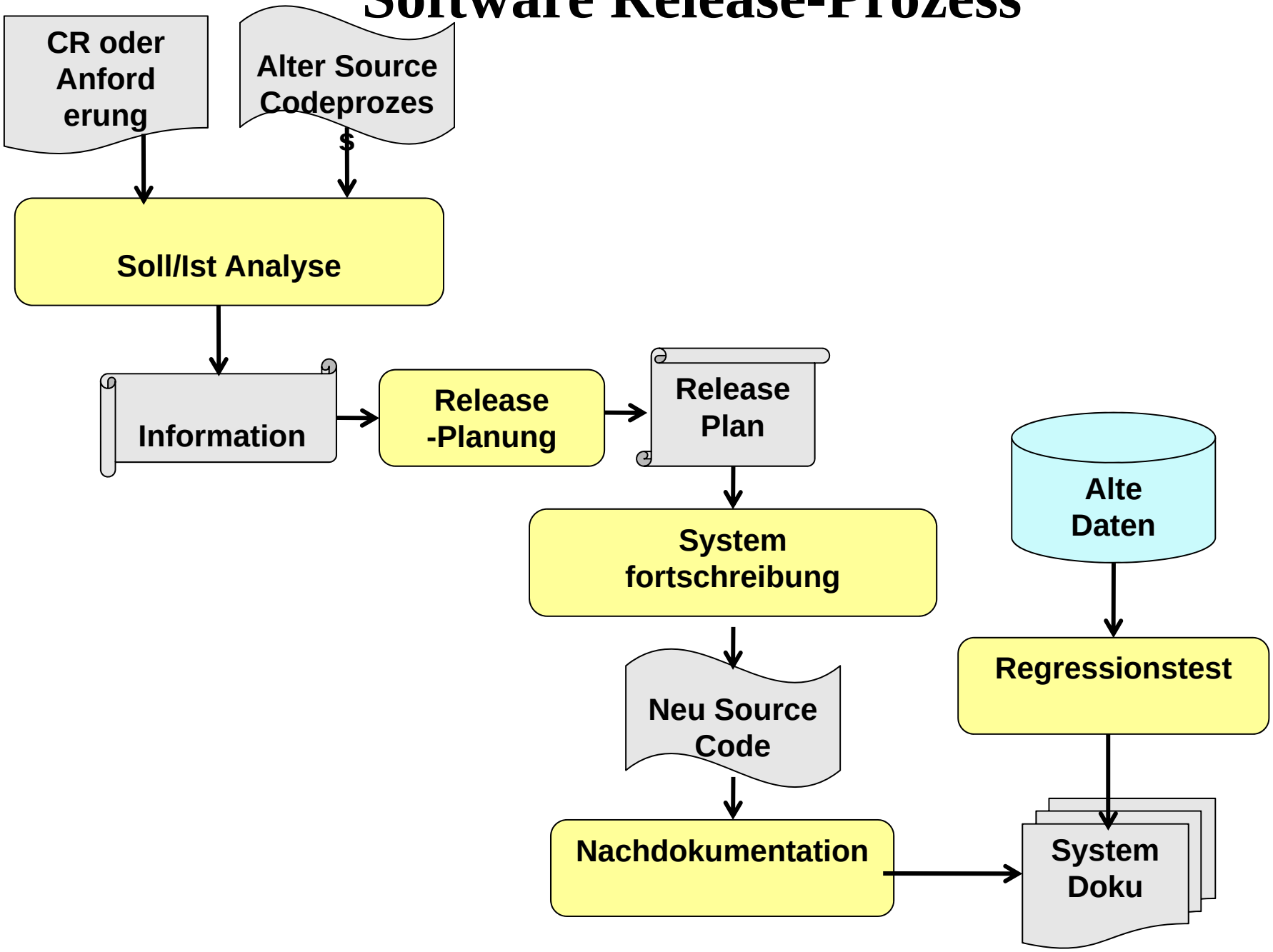
Bestehender Source Code

Schätzung der Weiterentwicklungs kosten aufgrund der Sourceanalyse

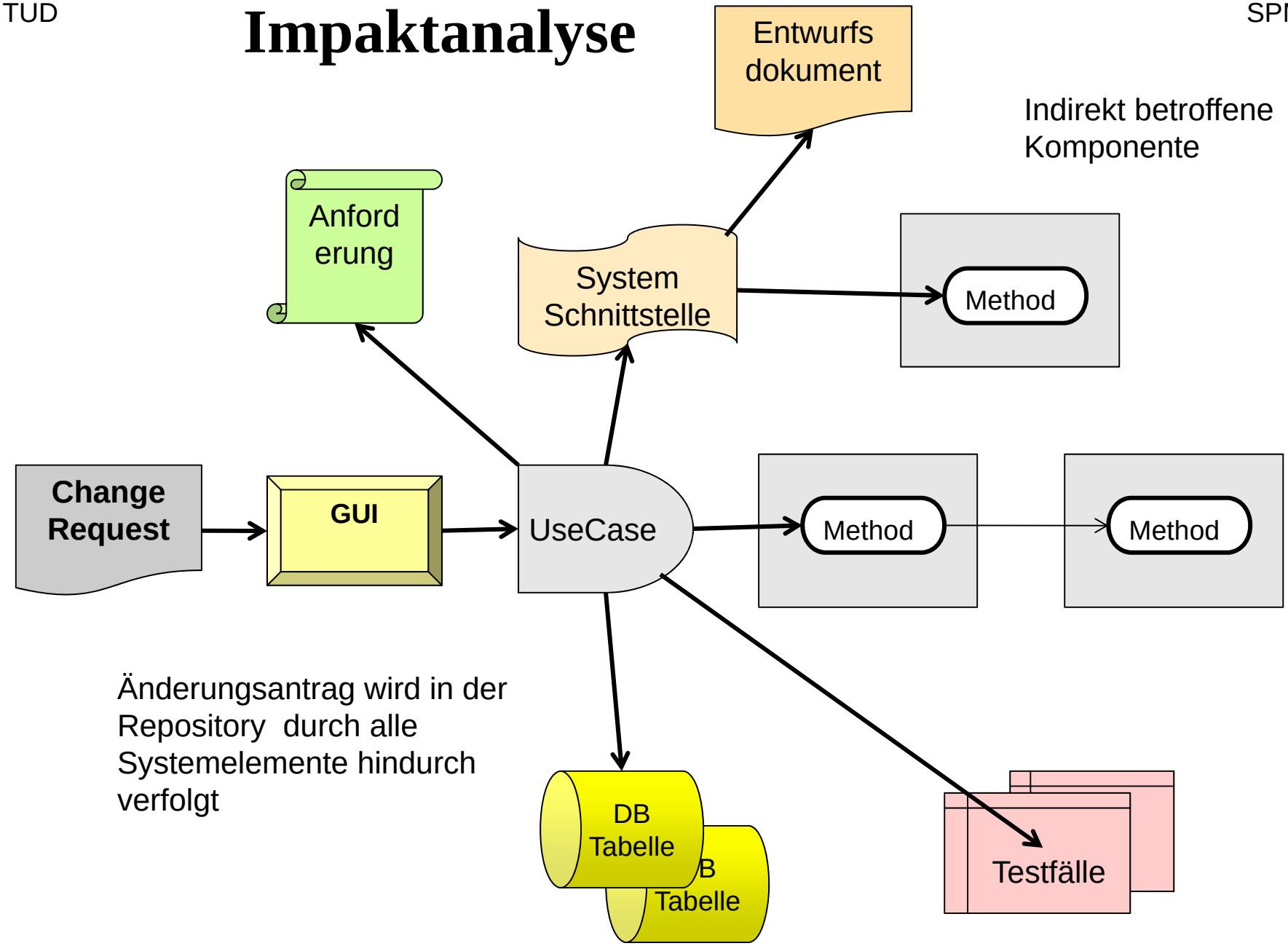
€ **Kosten der Weiterentwicklung**

Kosten der nächsten Releaseperiode =
Kosten der Fehlerbehebung des letzten Releases +
Kosten der Weiterentwicklung des nächsten Releases +
Produktmanagementkosten

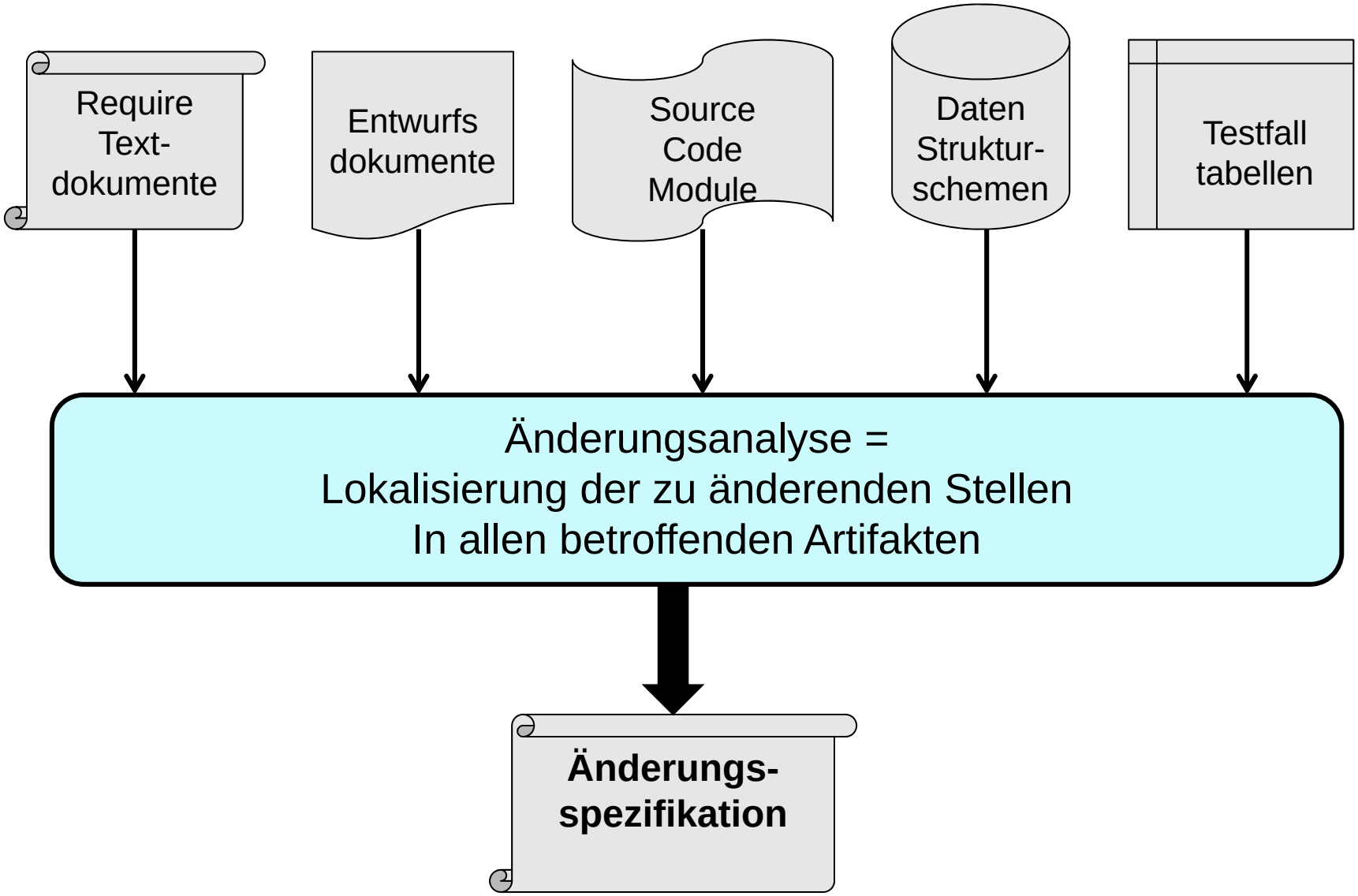
Software Release-Prozess



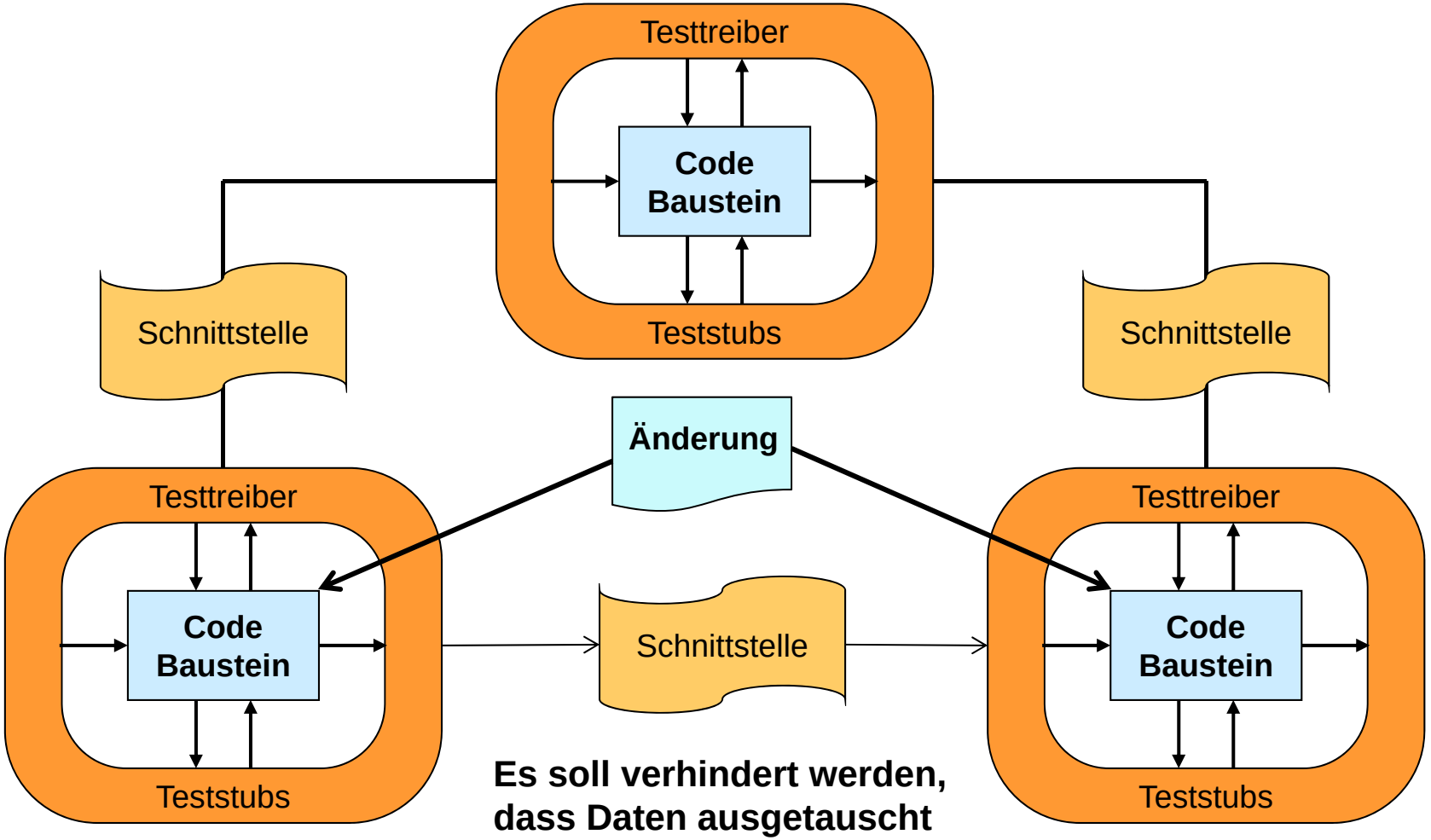
Impaktanalyse



Wirkungsbereich einer Systemänderung



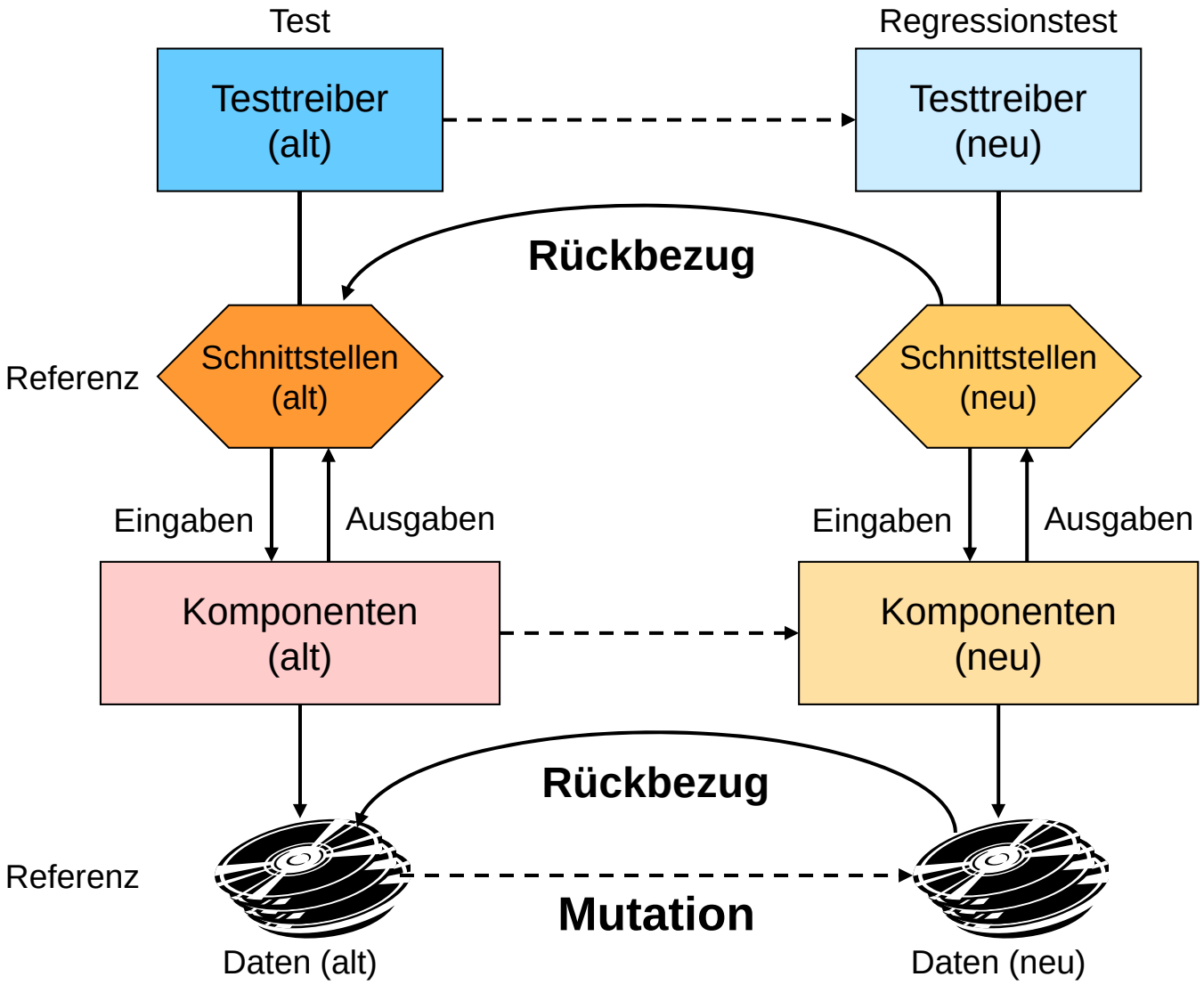
Abschottung der Code-Bausteine zwecks der Veränderung



Es soll verhindert werden, dass Daten ausgetauscht werden bis die Änderung abgesichert ist.

System-Regressionstest

Rückbezug auf den letzten Test

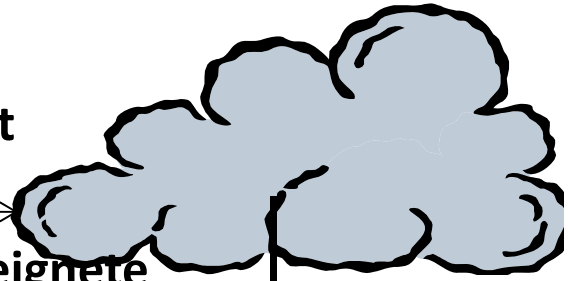


Tester treiben die Entwicklung voran

Product Owner

Verhandelt mit dem Architekt ueber die geeignete Loesung

System Architekt



Sucht geeignete Services

Tester-driven Ansatz

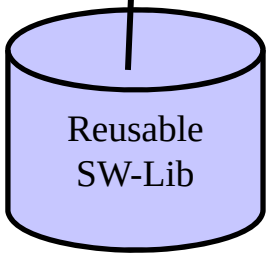
Tester als Integrator

integriert & testet

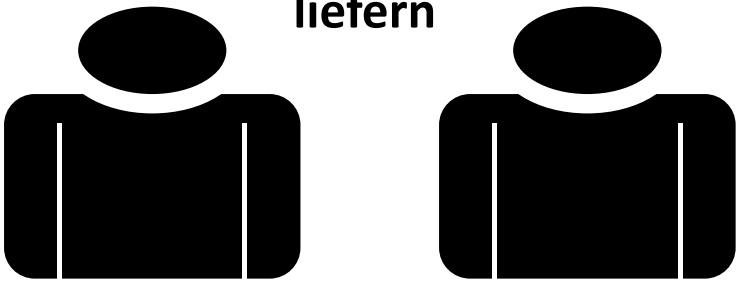
Standard Komponente

selbstgebaute Software

Standard Services



liefern



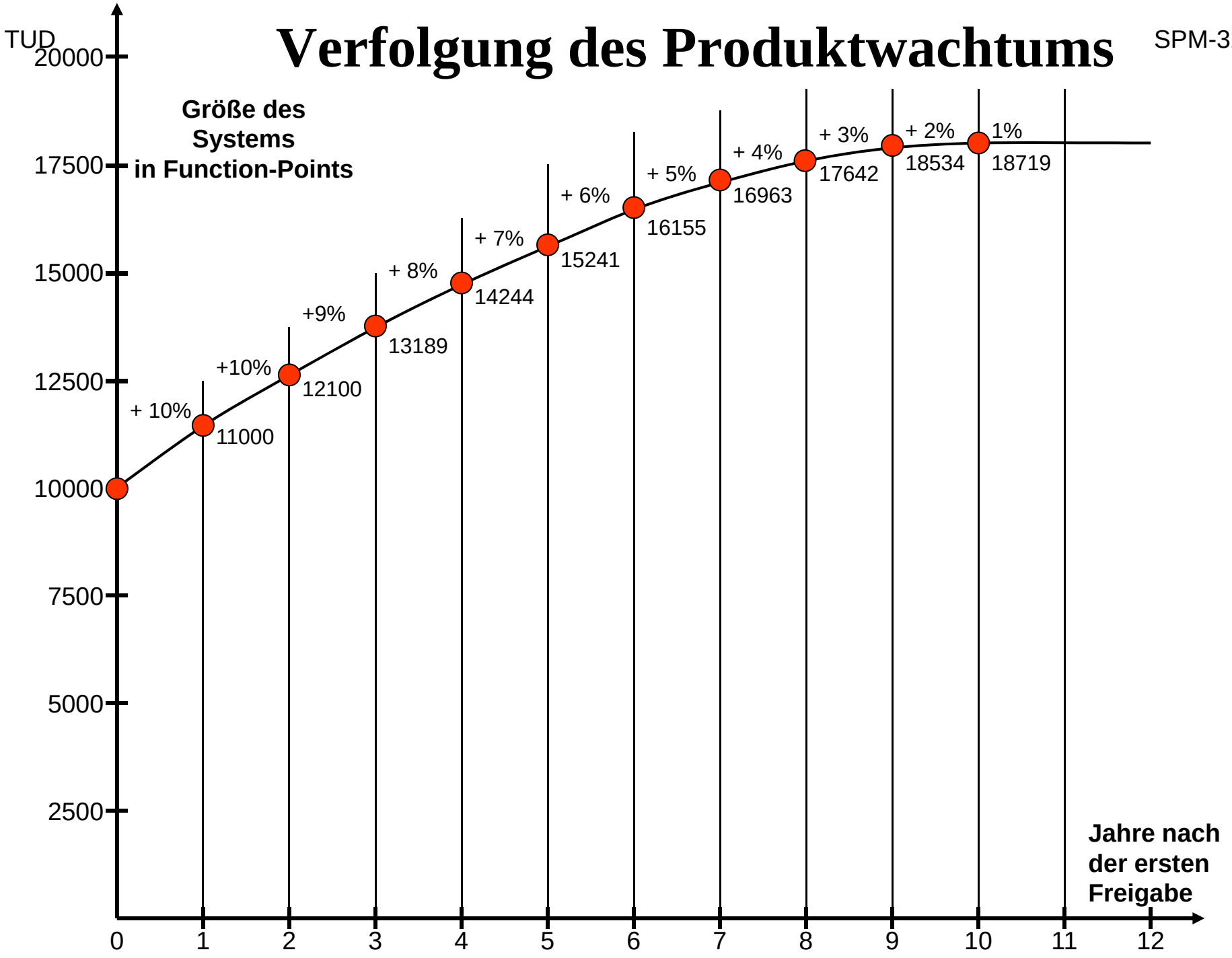
Entwickler

Ziel 3 = Verhinderung des Wertverfalles

Produktmanagementeam

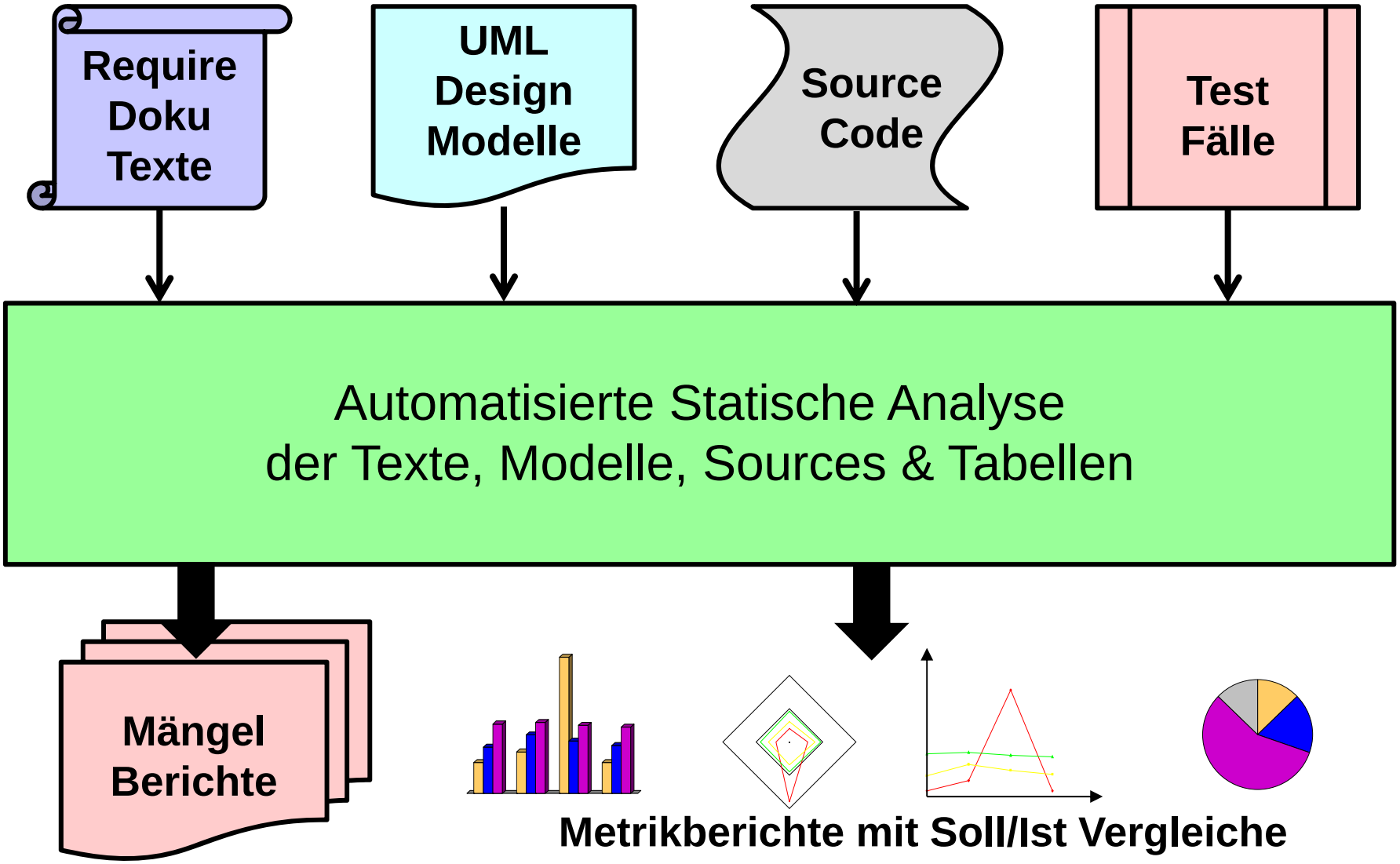
steuert die Produktevolution

Verfolgung des Produktwachtums

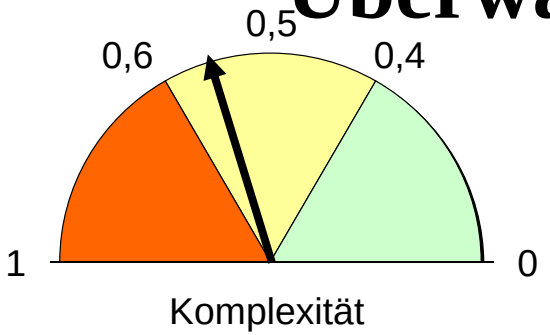


Prüfung & Messung der Produktartifakte

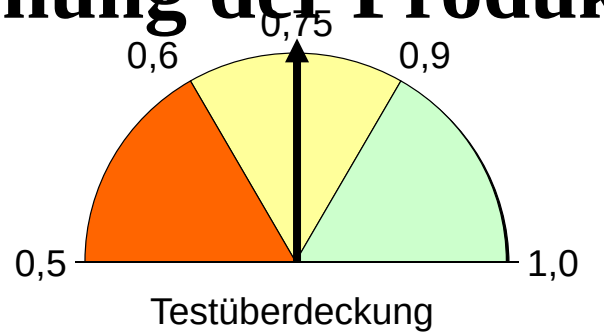
Aus den aktuellen Produkt-Bibliotheken



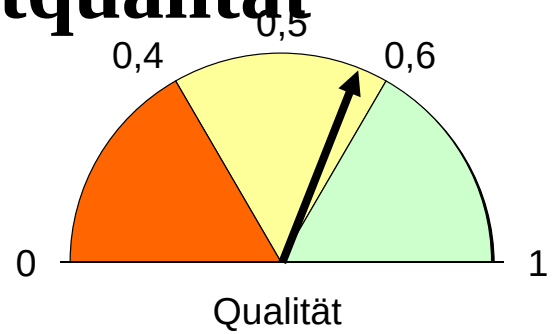
Überwachung der Produktqualität



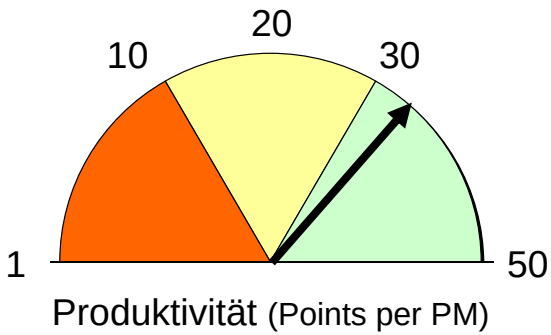
Komplexität



Testüberdeckung

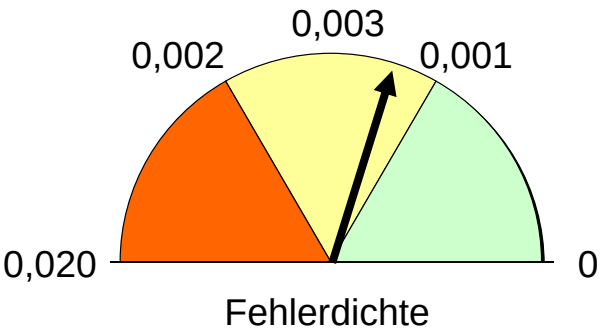


Qualität

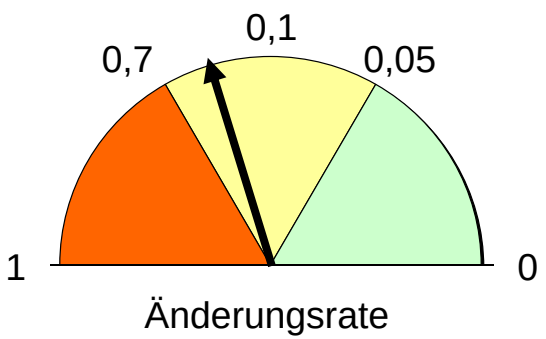


Produktivität (Points per PM)

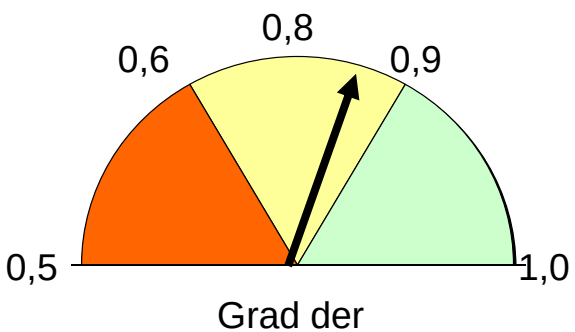
Der Produktmanager‘s Dashboard



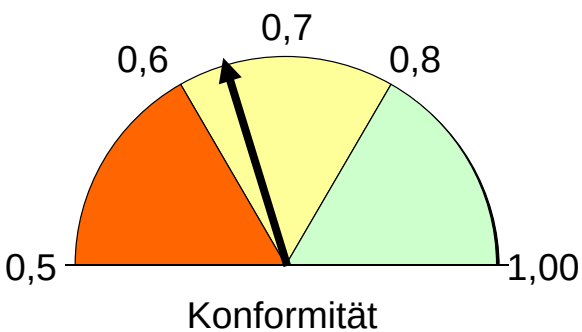
Fehlerdichte



Änderungsrate



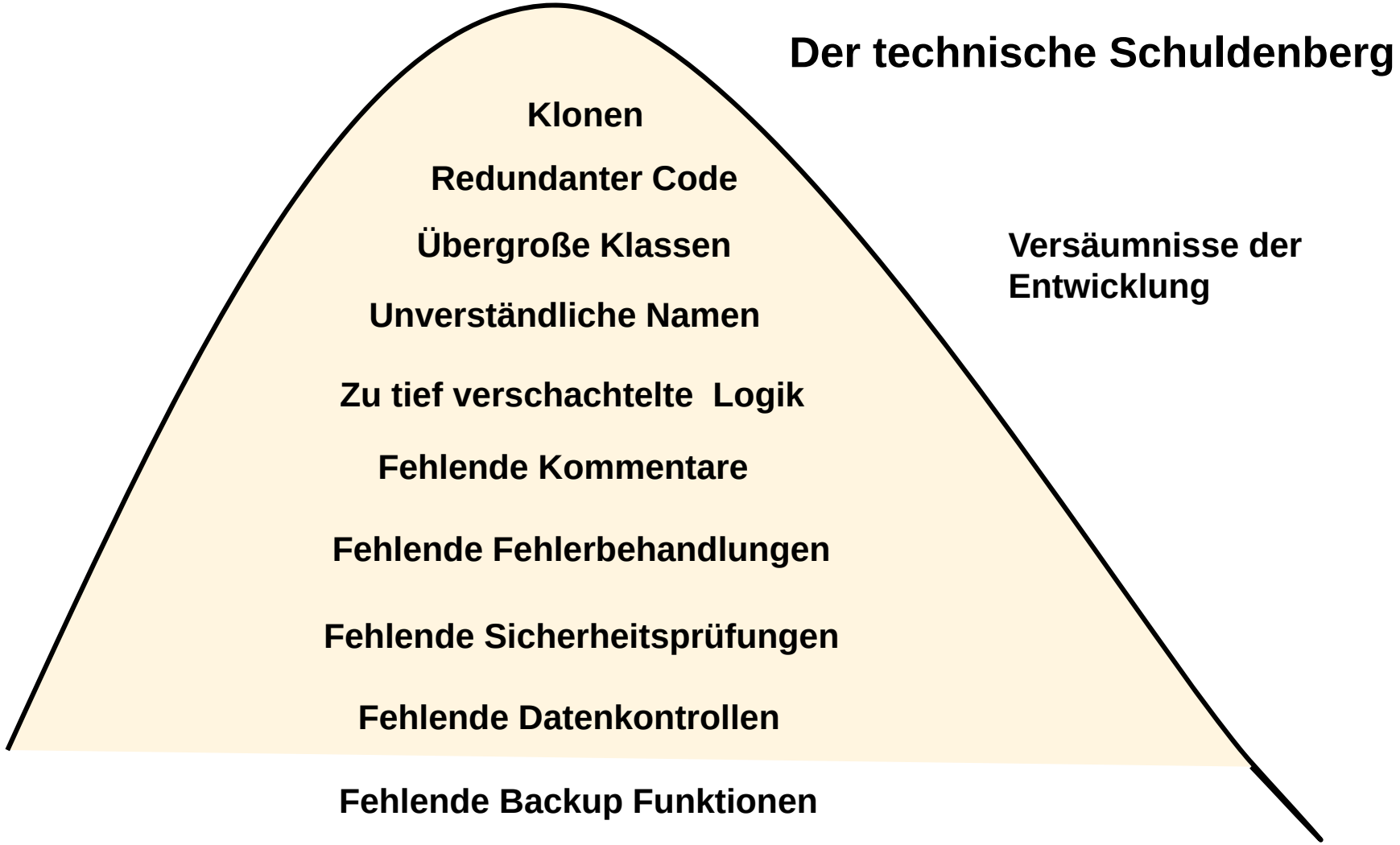
Grad der Anforderungserfüllung



Konformität

Produktzustand wird an Hand Qualitätsindikatoren überwacht

Verfolgung des Qualitätsverfalls

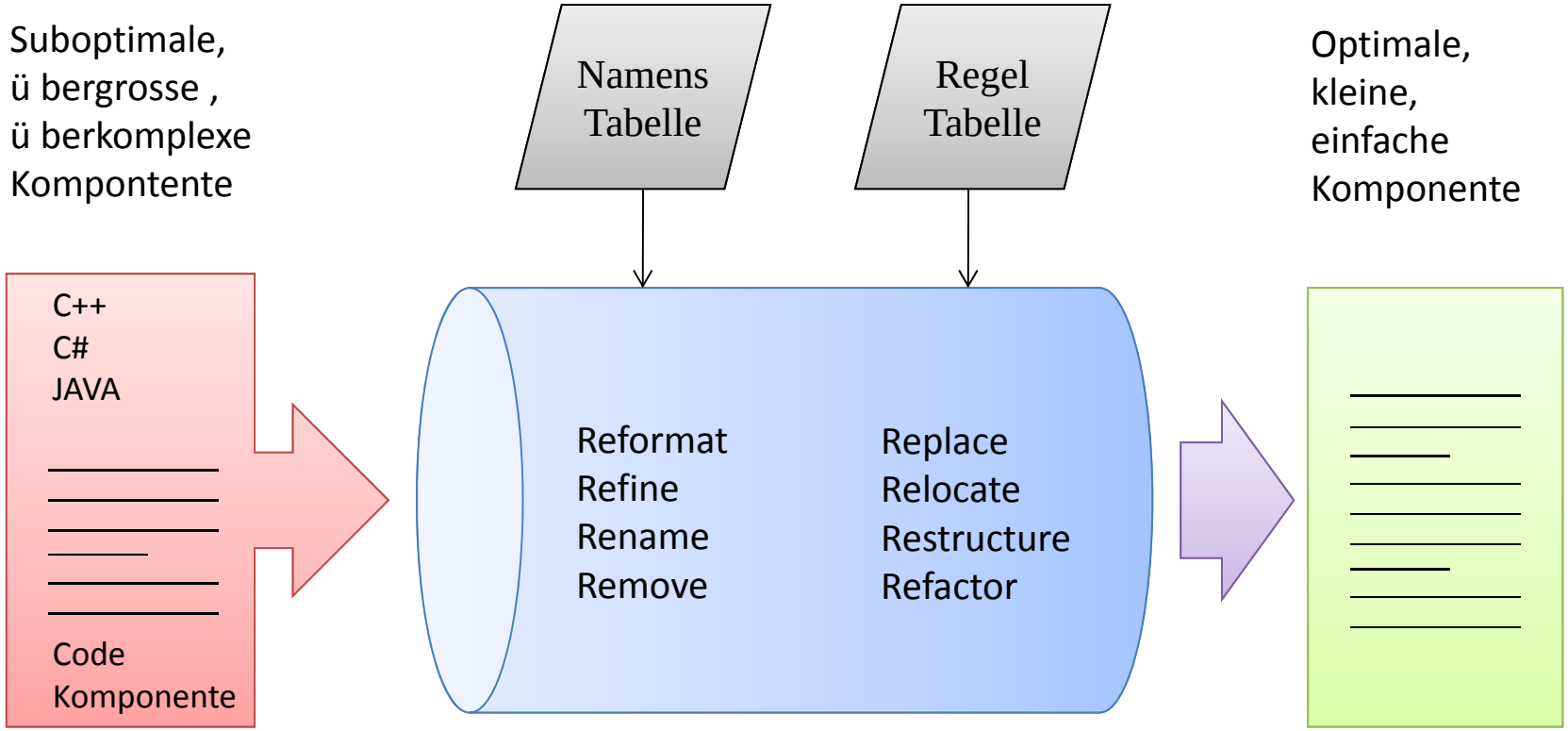


Technische Schulden im Code

Major deficiences sorted by number of occurences	**NachbesserungsAufwand(Stunden)**	
(10) Return Value is not controlled after Method Call 2435	x 0,5	= 1217
(22) Public Variables should be avoided in Classes 2280	x 1	= 2280
(01) IO-Operations are not in a try block 913	x 1,5	= 1370
(25) Class is not derived from a Superordinate Class 219		
(14) Control logic exceeds maximum allowed nesting level 962	x 2	= 1924
(15) Missing Final clause in method declaration 856	x 0,5	= 428
(04) Data Casting should be avoided 692	x 2	= 1384
(18) Check on incoming public request is missing 535	x 4	= 2140
(26) Nested Classes are not allowed 376	x 4	= 1504
(27) Returning a Function my cause an endless loop 239	x 6	= 1434
(11) Conditions should not contain an Assignment 150	x 1	= 150
(17) Try and Catch clauses do not match 112	x 1	= 112
(13) Default is missing in Switch Statement 85	x 1	= 85
(12) Case block should contain a Break statement 77	x 2	= 154
(08) Method Invocation with Array is not in a try Block 31	x 3	= 93
(07) External Variables are not allowed 29	x 2	= 58
(06) Standard IO Functions are prohibited 21	x 4	= 84
(09) There should be no global Data Definitions in C# 5	x 2	= 10
(02) Two Dimensional Arrays violate 1. Normal Form 2	x 8	= 16
Sanierungsaufwand		= 1**4.443**

Produktsanierung

Suboptimale,
ü bergrosse ,
ü berkomplexe
Kompontente



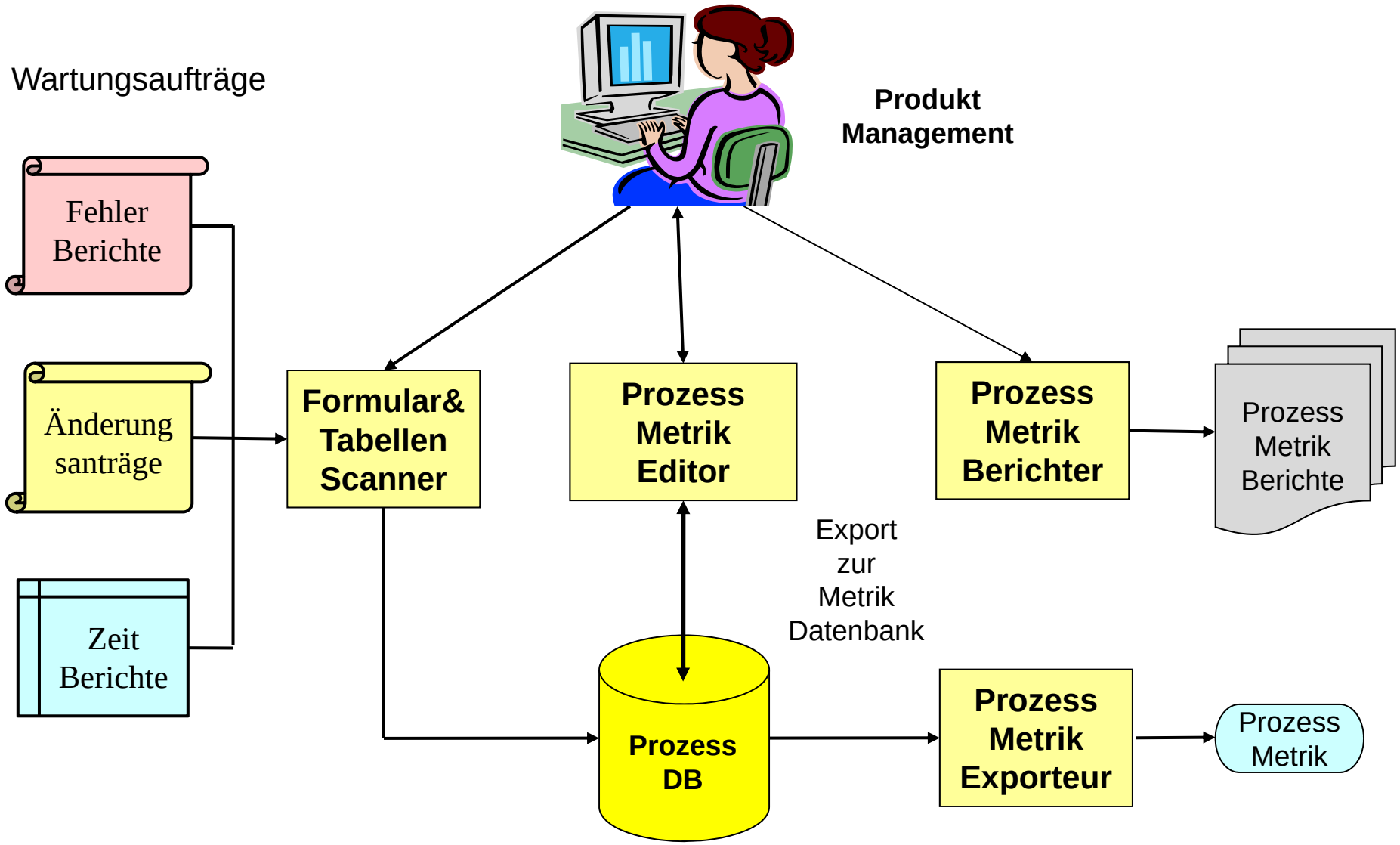
Optimale,
kleine,
einfache
Komponente

Code Refactoring

Zu grosse Klassen
Zu grosse mehrzweckige Methoden
Zu tiefe Logikverschachtelung
Zu viele Fallunterscheidungen
Zu viele redundante Methoden
Zu viele redundante Daten
Zu breite Schnittstellen
Nicht wiederverwendbar
Kaum kommentiert

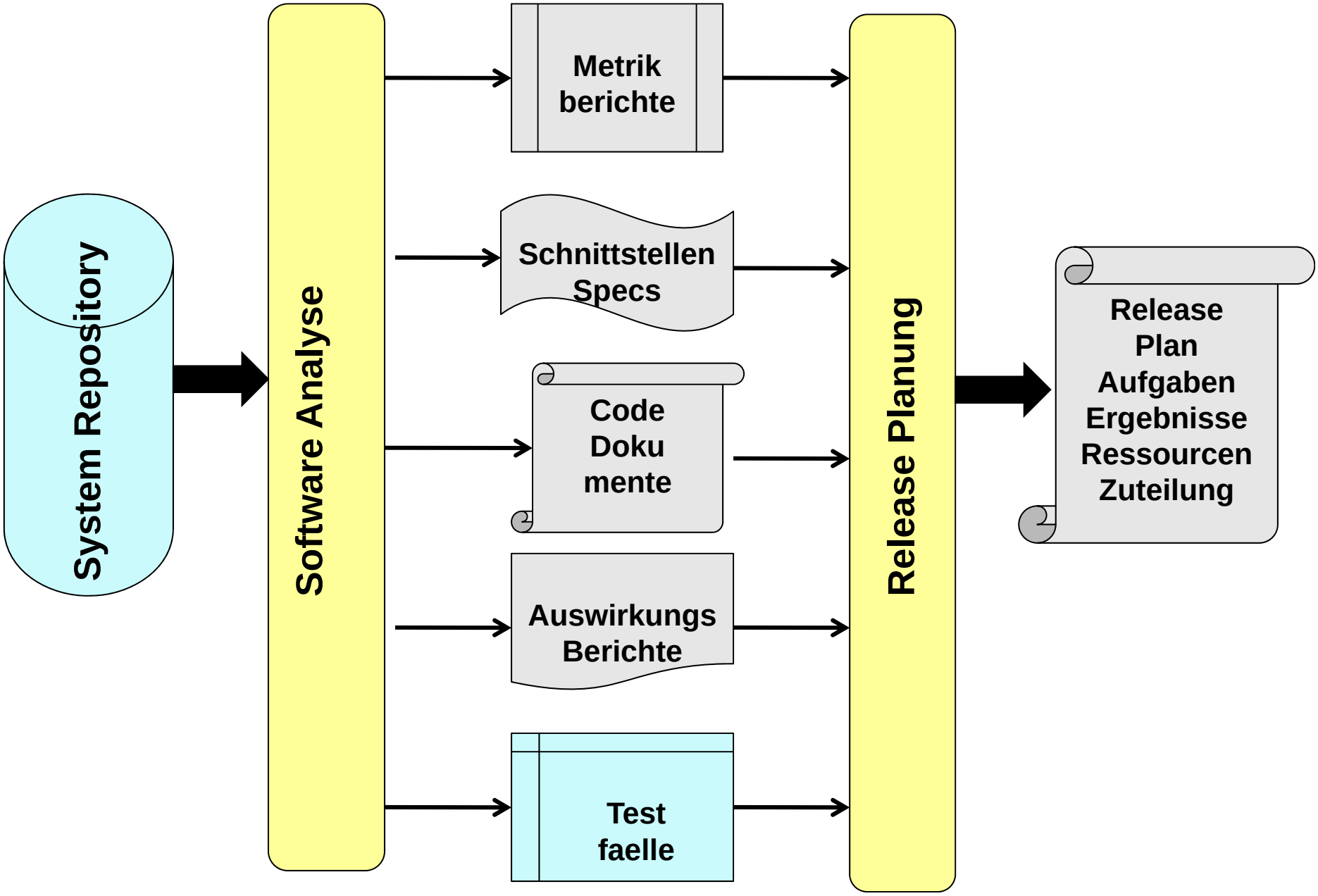
Kleine Klassen
Kleine einzweckige Methoden
Flache Ablauflogik
Keine Fallunterscheidungen
Keine redundante Methoden
Keine redundante Daten
Schmale Schnittstellen
Wiederverwendbar
Wohl kommentiert

Steuerung der Produktevolution



Prozessdaten werden mit Scanning Werkzeugen aus den Wartungsaufträgen gewonnen

Software Releaseplanung



Zusammenfassung

- Befreiung vom konventionellen Projektbegriff
- Projekte haben ein Anfang und ein Ende
- IT-Systementwicklung hat kein Ende
- Sie ist ein fortwährender Evolutionsprozess
- Ein IT-Produkt wird ständig korrigiert, erweitert und nachgebessert.
- Code, Design, Anforderungsdokumente und Testware sind parallel fortzuschreiben um konsistent zu bleiben.
- Produktverfall durch Erosion ist zu verhindern.
- Code, Design, Anforderungen und Testware sind deshalb regelmäßig zu sanieren (Reengineering)
- Projektleiter werden zum Produktmanager.