

Software Management

11. Sanierung

Prof. Dr. K.-P. Fähnrich

02.07.2009

Übersicht der Vorlesung

- 1. Grundlagen**
- 2. Planung**
- 3. Organisation: Gestaltung**
- 4. Organisation: Prozess-Modelle**
- 5. Personal**
- 6. Leitung**
- 7. Innovationsmanagement**
- 8. Kontrolle: Metriken, Konfigurations- und Änderungsmanagement**
- 9. CASE**
- 10. Wiederverwendung**
- 11. Sanierung**

Gliederung

1. Zur Problematik

2. Konzepte und ihre Terminologie

3. Technik

3.1. Verpacken von Altsystemen

3.2. Verstehen von Altsystemen

4. Kosten/Nutzen der Sanierung

5. CARE-Werkzeuge

Begleitliteratur: Helmut Balzert, Lehrbuch der Software-Technik
Quelle der Grafiken und Tabellen: Helmut Balzert, Lehrbuch der Software-Technik,
wenn nicht anders angegeben

1. Zur Problematik

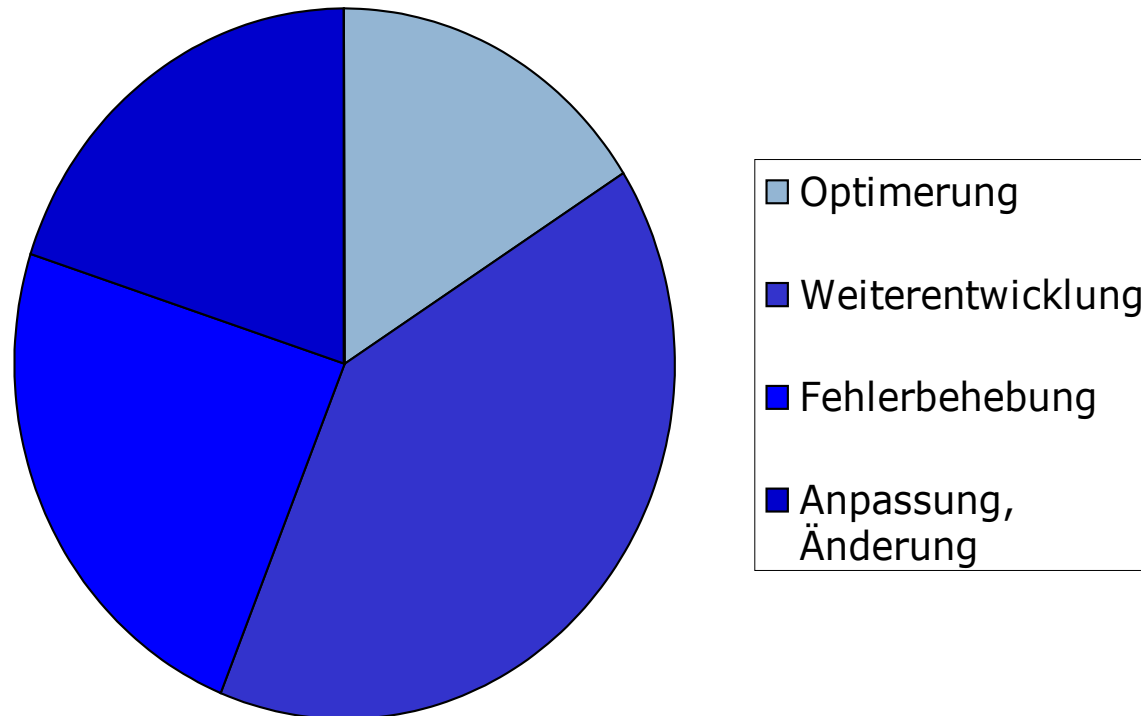
Fakten (nach [Sneed 92a])

- Die Wartungskosten eines durchschnittlichen Anwenderunternehmens liegt zwischen 50 und 75% des gesamten DV-Budgets.
- Nur 30% sind individueller, problemspezifischer Natur. Der Rest ist softwaretechnisch identisch und ließe sich mit entsprechenden Standards abdecken.
- Ein typischer Anwender in den USA hat durchschnittlich 2200 Programme mit 1,15 Millionen Anweisungen, die 40 bis 50 Anwendungen abdecken.
- Ein typisches Programm in den USA ist meistens in COBOL geschrieben, lebt 5 bis 7 Jahre und enthält 700 Anweisungen.
- Ein Programm, das ein hierarchisches oder netzwerkorientiertes DBS benutzt, wird jährlich zwischen 10 und 20-mal angepasst.

1. Zur Problematik

Verteilung des Wartungsaufwandes (nach [Sneed 92 a])

Wartungsaufwand



1. Zur Problematik

Fakten (nach [Sneed 92a]) (2)

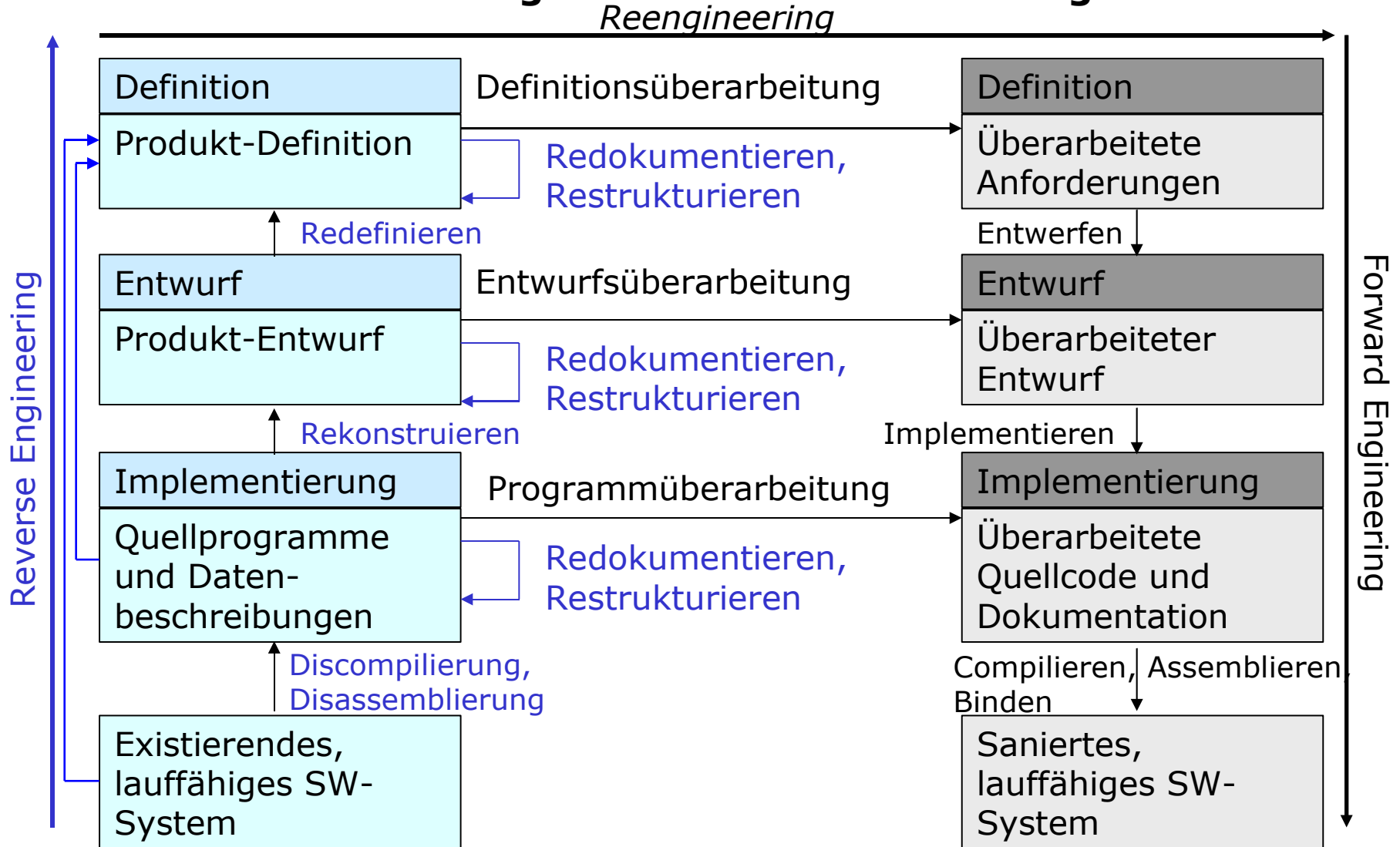
- Deutsche Cobol-Programme sind zu 80% monolithisch, zu 77% unstrukturiert und enthalten zu 93% überflüssige, redundant gehaltene Daten.
- Ein Cobol-Programmierer verwaltet ca 1100 Data-Division-Anweisungen, 2000 Procedure-Division-Anweisungen und 270 Sprunganweisungen.
- Die Komplexität eines unstrukturierten Programms erhöht sich nach einer Korrektur um 4%, nach einer Änderung um 17% und nach einer Erweiterung um 26%.
- Ein Wartungsprogrammierer benötigt 47% seiner Zeit für die Programmanalyse, 15% für die Programmierung, 28% für den test und 9% für die Dokumentation.
- Die Wartungskosten je geänderte Zeile sind bei kleinen Anwendungen am höchsten.

Gliederung

- 1. Zur Problematik**
- 2. Konzepte und ihre Terminologie**
- 3. Technik**
 - 3.1. Verpacken von Altsystemen**
 - 3.2. Verstehen von Altsystemen**
- 4. Kosten/Nutzen der Sanierung**
- 5. CARE-Werkzeuge**

2. Konzepte und ihre Terminologie

Terminologie der Software-Sanierung



2. Konzepte und ihre Terminologie

Definitionen

Forward Engineering: stellt den normalen Entwicklungsablauf bei der Neuentwicklung von Softwaresystemen dar.

Reverse Engineering: Hierbei wird versucht, aus einem fertigen Produkt, z.B. einem Prozessor oder einem Schaltkreis, die Produktspezifikation abzuleiten.

Re-engineering (Renovierung): Umfasst alle Aktivitäten zur Änderung von Software-Altsystemen, um sie in einer neuen Form wieder implementieren zu können.

Sanierung: Umfasst alle notwendigen Reverse Engineering; Reengineering und Forward Engineering-Maßnahmen, um ein saniertes lauffähiges System zu erhalten, das definierte neue Ziele erfüllt.

Gliederung

- 1. Zur Problematik**
- 2. Konzepte und ihre Terminologie**
- 3. Technik**
 - 3.1. Verpacken von Altsystemen**
 - 3.2. Verstehen von Altsystemen**
- 4. Kosten/Nutzen der Sanierung**
- 5. CARE-Werkzeuge**

Begleitliteratur: Helmut Balzert, Lehrbuch der Software-Technik
Quelle der Grafiken und Tabellen: Helmut Balzert, Lehrbuch der Software-Technik,
wenn nicht anders angegeben

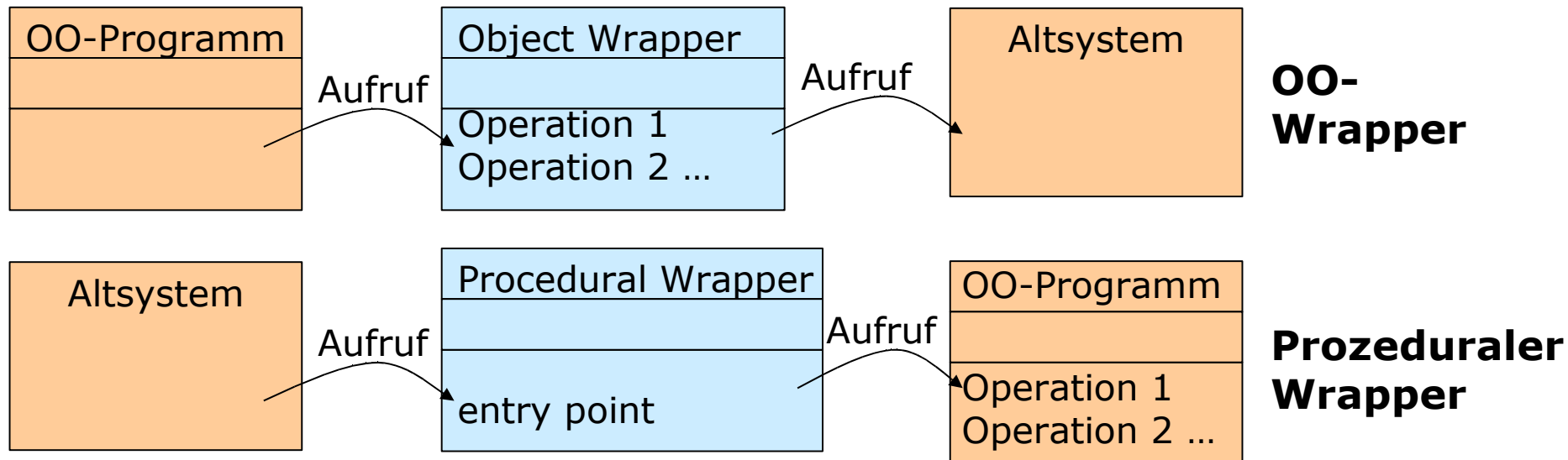
Verfahren der Sanierung

- Kategorien von Verfahren zur Realisierung der verschiedenen Konzepte der Sanierung:
 - Verpacken (*wrapping*),
 - Verstehen und
 - Verbessern von Altsystemen.
- Hat man mittels Reverse Engineering ein ausreichendes Verständnis und eine angemessene Dokumentation der jeweiligen Abstraktionsebene erreicht, dann kann auf dieser Grundlage das Altsystem verbessert werden.
- Eine Konvertierung eines Altsystem in eine neue Programmiersprache ohne Veränderung der Systemarchitektur erfordert nur die Transformierung des Codes in eine neue Programmiersprache

3.1. Verpacken von Altsystemen

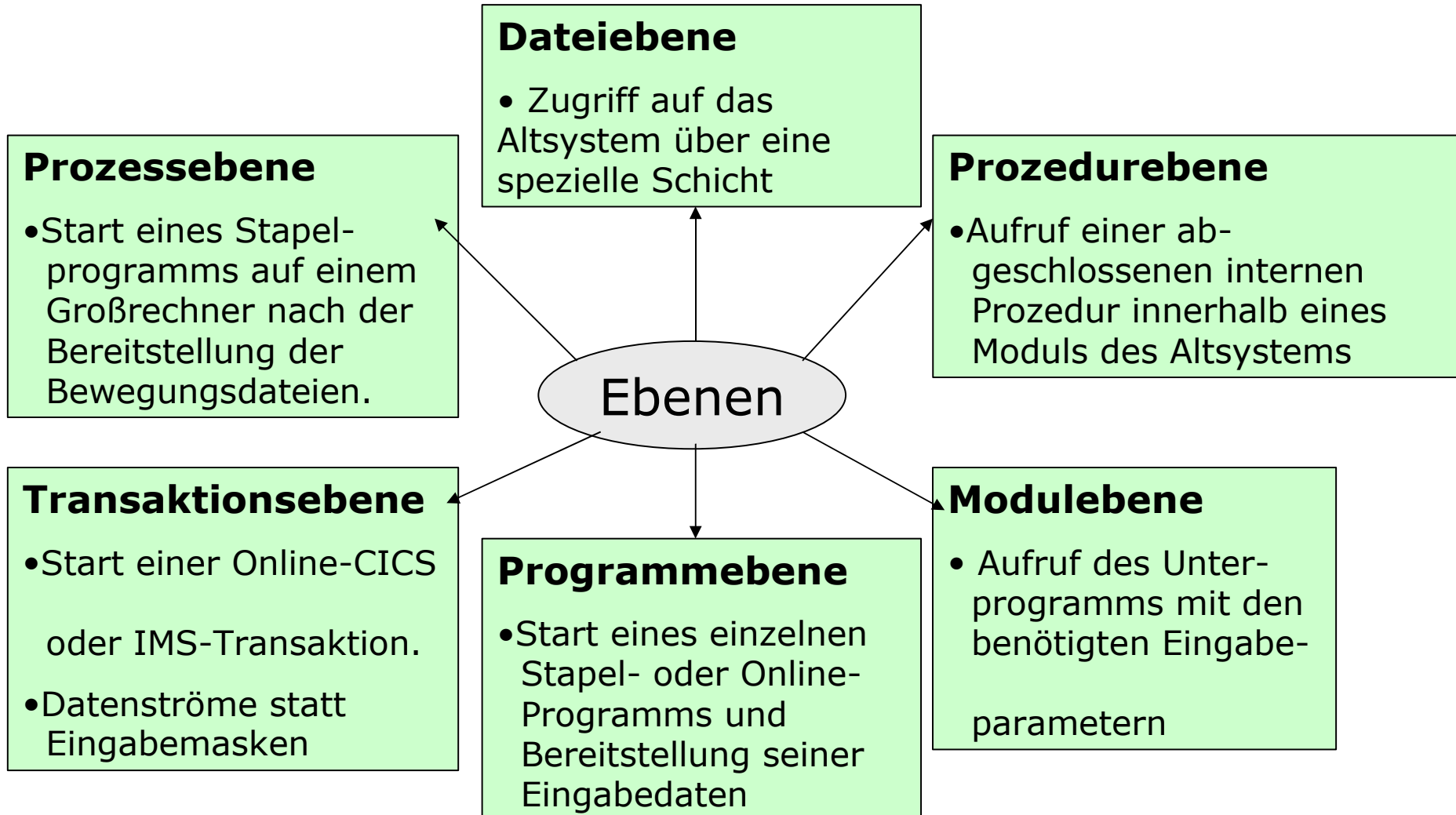
Wrapping

- Entstanden im Rahmen der objektorientierten Entwicklung.
- Ist sowohl für die Sanierung als auch für die Migration von Altsoftware geeignet.
- **Grundidee:** Verpacken der Altsoftware oder ihrer Teilsysteme durch eine Software-Schicht nach außen hin, so dass eine OO-Benutzung möglich wird.



3.1. Verpacken von Altsystemen

Verpackungsebenen konventionaler Altsysteme



3.2. Verstehen von Altsystemen

Möglichkeiten

- Prinzipiell gibt es 3 Möglichkeiten:
 - Verstehen des Systems als Black box,
 - Verstehen des Systems als White box,
 - Mischformen der ersten beiden Möglichkeiten.
- Das Verstehen des Systems als Blackbox ist in vielen Fällen ausreichend. Dazu werden 3 Engineering-Formen verwendet:
 - Reverse Engineering: Erstellen eines OOA-Modells des Altsystems
 - Reengineering: Überarbeitung, Verbesserung und Erweiterung des entstandenen OOA-Modells.
 - Forward Engineering: Entwicklung eines neuen Systems ausgehend vom OOA-Modell.

Gliederung

- 1. Zur Problematik**
- 2. Konzepte und ihre Terminologie**
- 3. Technik**
 - 3.1. Verpacken von Altsystemen**
 - 3.2. Verstehen von Altsystemen**
- 4. Kosten/Nutzen der Sanierung**
- 5. CARE-Werkzeuge**

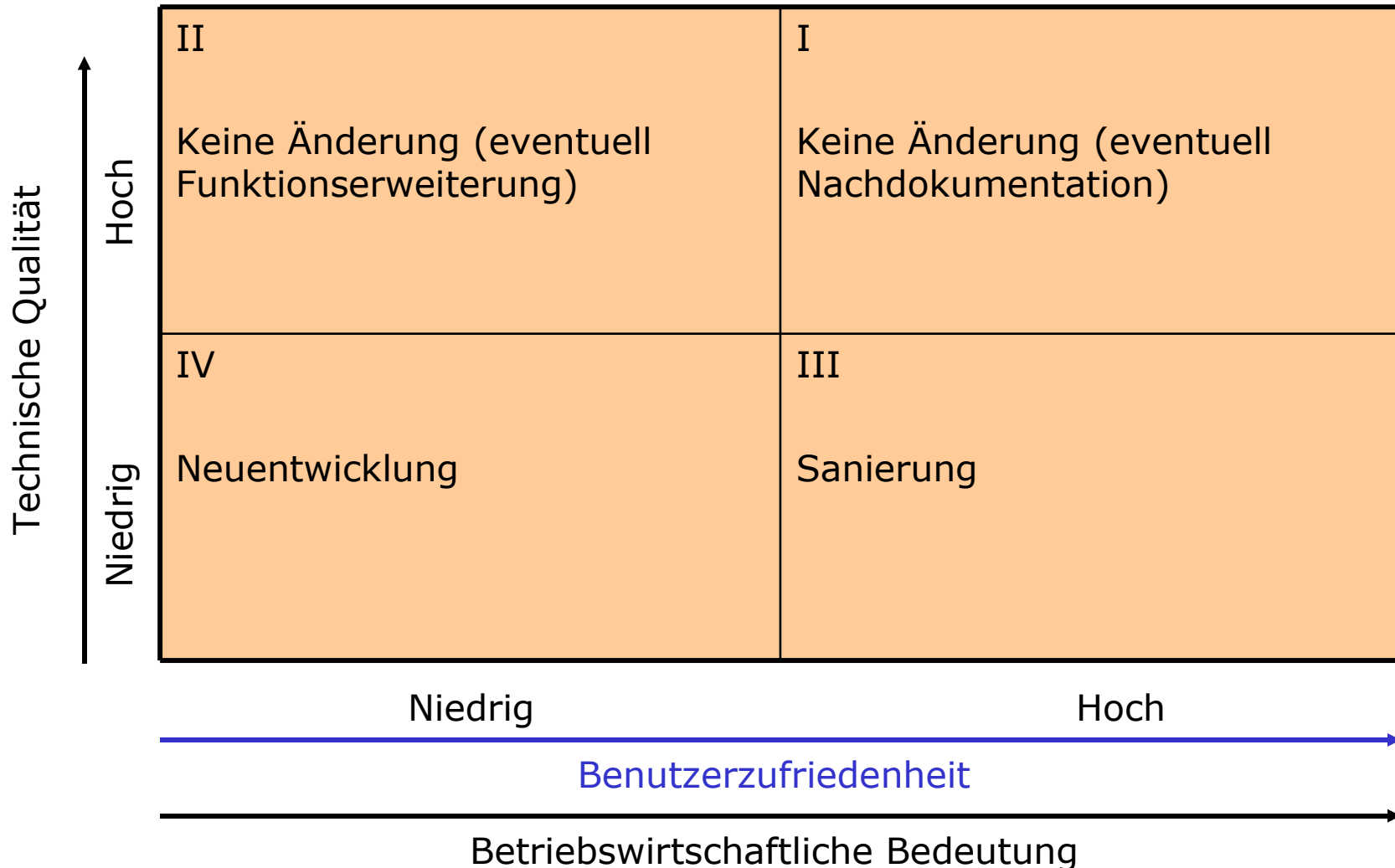
4. Kosten/Nutzen der Sanierung

Möglichkeiten

- Folgende Möglichkeiten der Kosten/Nutzenbetrachtung stehen zur Auswahl:
 - Weiterführung der korrektiven und adaptiven Wartung,
 - Sanierung,
 - Neuentwicklung,
 - Einsatz einer Standardsoftware.
- Einen ersten Anhaltspunkt für eine mögliche Sanierung gibt eine Portfolioanalyse.
- Alte Systeme werden nach 2 Kategorien bewertet:
 - der technischen Qualität und
 - der Benutzerzufriedenheit oder der betriebswirtschaftlichen Bedeutung.

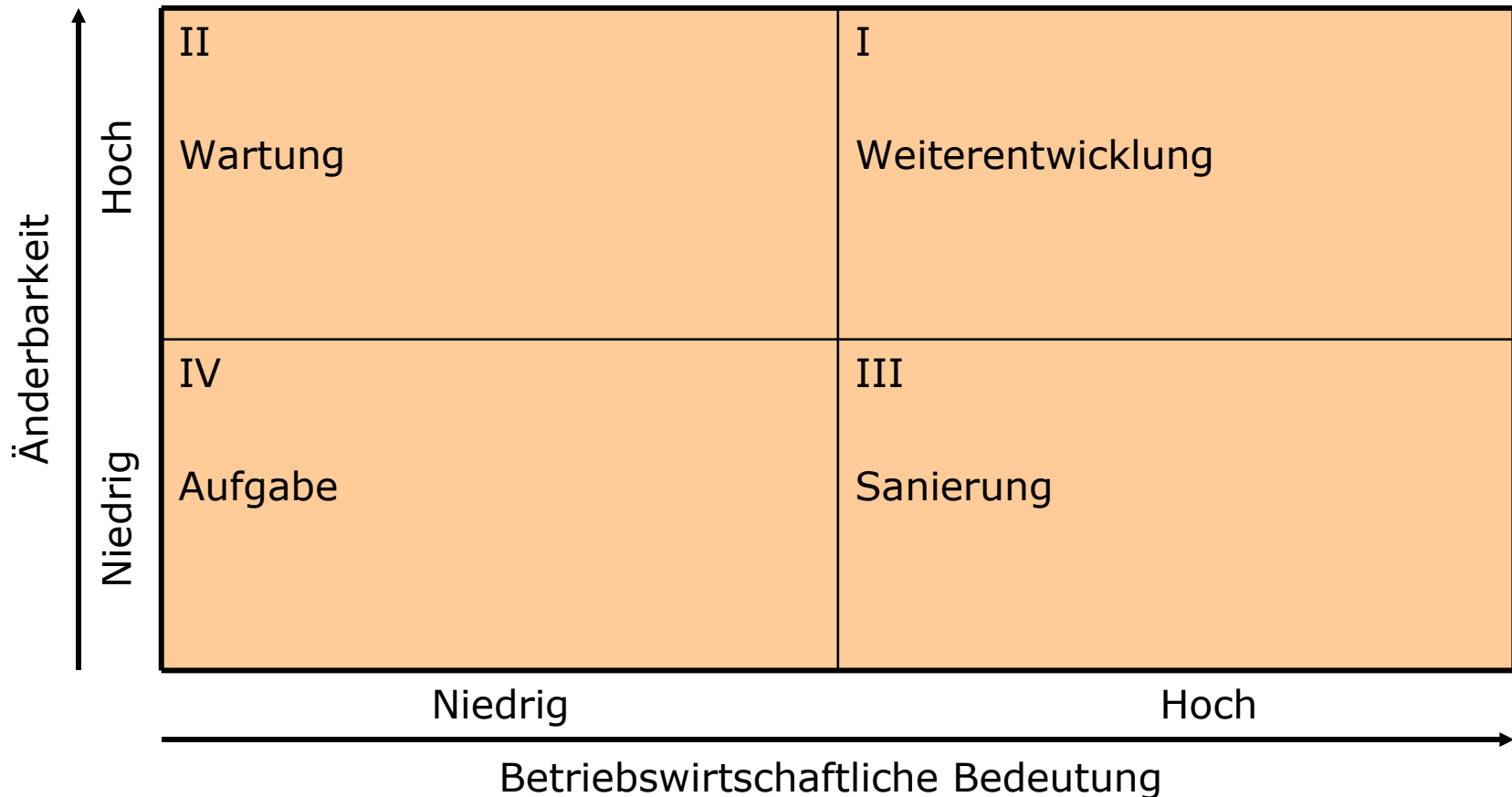
4. Kosten/Nutzen der Sanierung

Portfolioanalyse der Altsysteme (nach [Sneed 92a, 95])



4. Kosten/Nutzen der Sanierung

Portfolioanalyse der Altsysteme (nach [Jacobson, Lindström 91])



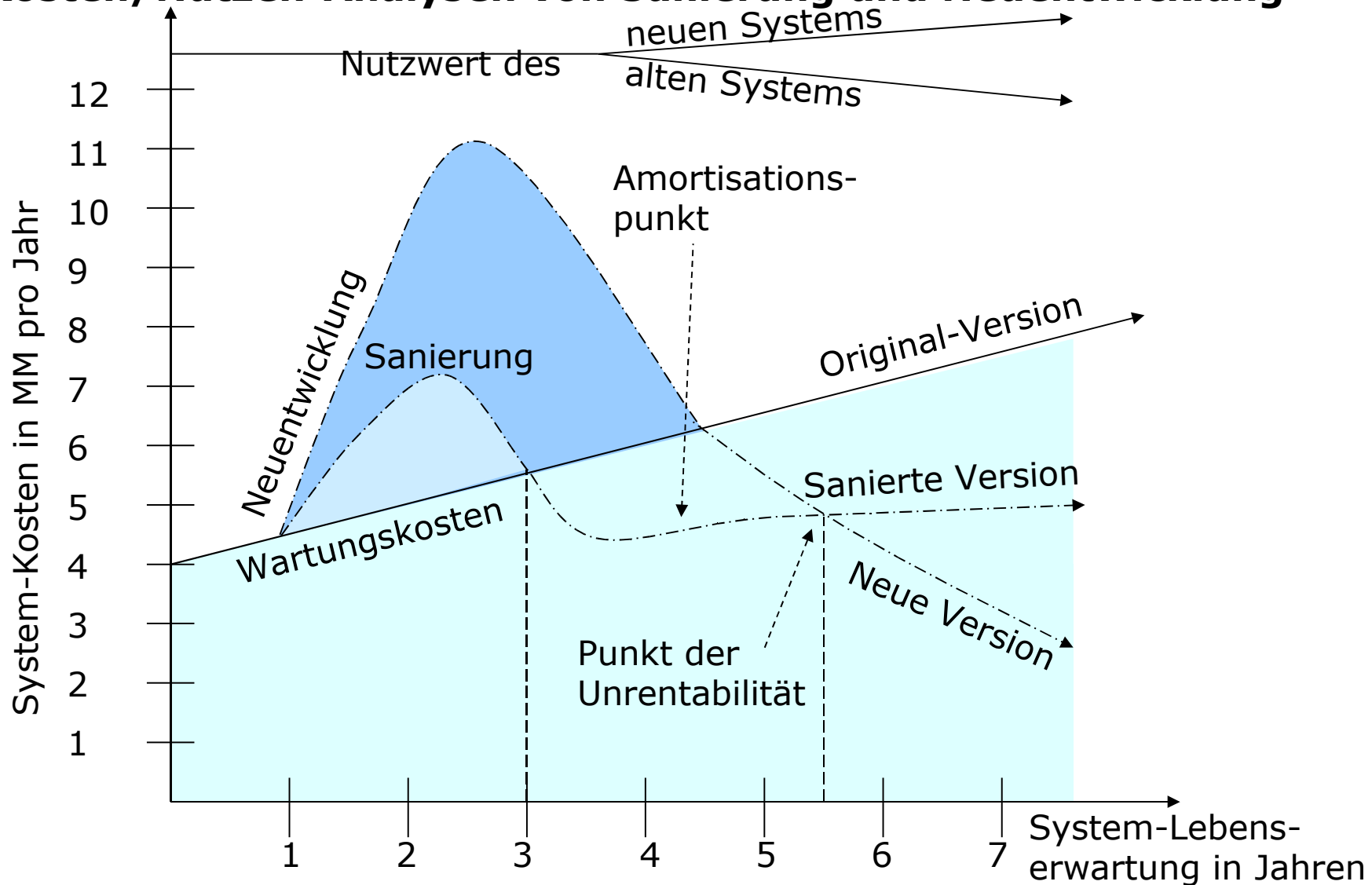
4. Kosten/Nutzen der Sanierung

Kosten/Nutzen-Analysen

- 3 Ursachen der Sanierung (nach [Sneed 92a]):
 - Das Altsystem ist technisch überholt und muss ersetzt werden.
Sani-Nutzen = (Alt-Wert - [Sani-Nutzen * Sani-Risiken]) -
(Neu-Wert - [Entw-Kosten * Entw-Risiken])
 - Das Altsystem hat schwerwiegende technische Mängel, die die Wartung erschweren und die Leistung beeinträchtigen.
Sani-Nutzen = (Alte-Wart-Kosten - Sani-Wart-Kosten)
+ (Alt-Wert - [Sani-Nutzen * Sani-Risiken])
- (Neu-Wert - [Entw-Kosten * Entw-Risiken])
 - Das Altsystem sollte überarbeitet werden, um die Wartungskosten zu reduzieren und die technische Qualität zu verbessern.
Sani-Nutzen = (Jährliche Kostenersparnis durch Sanierung)
- (Sani-Kosten * Sani-Risiken)

4. Kosten/Nutzen der Sanierung

Kosten/Nutzen-Analysen von Sanierung und Neuentwicklung



4. Kosten/Nutzen der Sanierung

Nutzen einer Sanierung (nach [Figliolo 89])

- Niedrige Wartungskosten durch ersparten Aufwand pro Wartungsauftrag.
- Niedrige Wartungskosten durch die Ablösung von höher qualifiziertem Personal durch jüngeres, weniger qualifiziertes Personal.
- Niedrigere Produktionskosten durch eine Reduzierung der Systemabbrüche.
- Niedrige Verluste wegen „missed opportunities“ durch Freisetzung gebundener Kapazitäten für neue Aufgaben.

4. Kosten/Nutzen der Sanierung

Risiken

- Das Risiko einer Neuentwicklung ist 2 bis 3 mal höher als das einer Sanierung unter der Voraussetzung, dass die Datenstrukturen unverändert bleiben.
- Die Kosten einer Sanierung betragen normalerweise nur $\frac{1}{4}$ der Kosten einer Neuentwicklung.
- Die Sanierung ist immer dann eine günstige Alternative, wenn Funktionalität und Informationsstruktur der Software konstant bleiben.
- Eine günstiger Zeitpunkt für eine erfolgreiche Sanierung liegt vor, wenn
 - die Wartungsmitarbeiter ausscheiden oder versetzt werden,
 - das System in eine andere technische Umgebung konvertiert werden,
 - das System funktional überarbeitet wird.

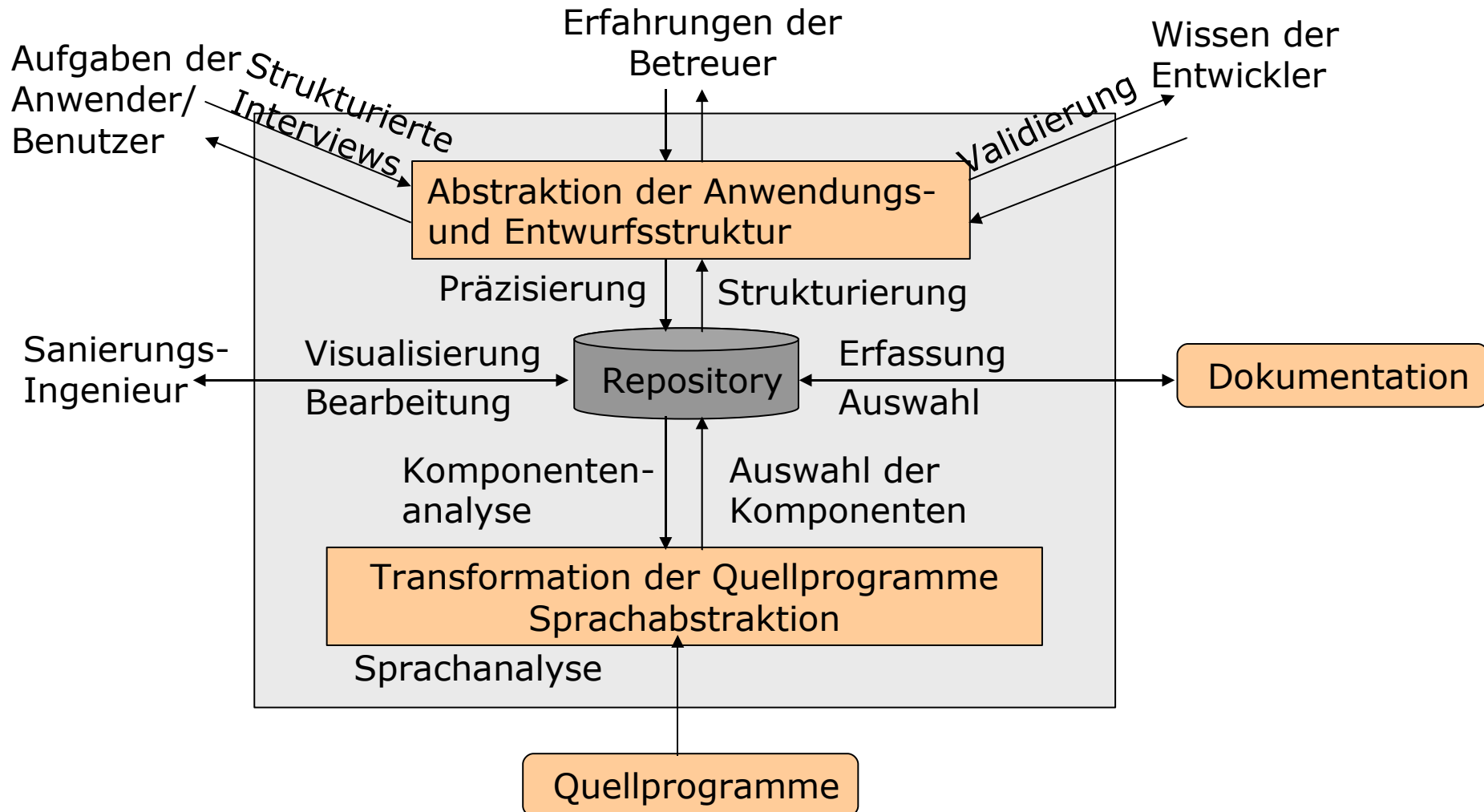
Gliederung

- 1. Zur Problematik**
- 2. Konzepte und ihre Terminologie**
- 3. Technik**
 - 3.1. Verpacken von Altsystemen**
 - 3.2. Verstehen von Altsystemen**
- 4. Kosten/Nutzen der Sanierung**
- 5. CARE-Werkzeuge**

Begleitliteratur: Helmut Balzert, Lehrbuch der Software-Technik
Quelle der Grafiken und Tabellen: Helmut Balzert, Lehrbuch der Software-Technik,
wenn nicht anders angegeben

5. CARE-Werkzeuge

Beispiel einer CARE-Umgebung (nach [Witschurke 95])



5. CARE-Werkzeuge

Aussagen

- CARE-Werkzeuge für die Reformatierung und Nachdokumentation sind sehr umfassend und hilfreich für die Wartung.
- Der Nutzen von Werkzeugen zur Code-Restrukturierung und zur Modularisierung muss noch nachgewiesen werden.
- Eine vollautomatische Programmmanierung mit Rekonstruktion und Redefinition ist gegenwärtig nur eingeschränkt möglich.
- CARE-Werkzeuge müssen in eine CASE-Umgebung integriert oder an sie anbindbar sein.
- Mit der Unterstützung von CARE-Werkzeugen lässt sich die Lebenszeit von Altsystemen so verlängern, dass die vorzeitig nicht entsorgt werden müssen.
- Durch den gezielten Einsatz von CARE-Werkzeugen kann der Wartungsaufwand um etwa 20 bis 25% verringert werden.

5. CARE-Werkzeuge

Aussagen (2)

- Durch verbesserte CARE-Werkzeuge konnte die Anzahl der restrukturierten, redokumentierten und konvertierten Anweisungen pro Entwickler und pro Tag von 70 Anweisungen (1983) über 350 Anweisungen (1986) auf 2000 Anweisungen (1990) gesteigert werden.
- Die Verbreitung und Anwendung von CARE-Werkzeuge entsprechen nicht ihrer Leistungsfähigkeit. Gründe für die Stagnation liegen in den relativ hohen Kosten, fehlende Schnittstellen sowie einer Vorliebe für Neuentwicklung und Standorte.
- Die Weiterentwicklung der Werkzeuge bezogen auf den Funktionsumfang und den Bedienungskomfort hält sich in Grenzen.
- Das Angebot für die Sprachen C und C++ steigt.

- [Figliolo 89]
Figliolo R., benefits of Software Reengineering, in Proc. Of Software Maintenance Association Conference
- [Jacobson, Lindström 91]
Jacobson I., Lindström F., Re-engineering of old systems to an object-oriented architecture, OOPSLA
- [Sneed 92 a]
Sneed H., Softwaresanierung, Rudolf Müller Verlag
- [Sneed 95]
Sneed H., Wann Softwaresanierung wirtschaftlich ist, in online 3/92
- [Witschurke 95]
Witschurke R., Verständnisvoll - Interaktives Reverse Engineering, in iX 6/95