Software Engineering II

11 Evolution und Wartung von Software

SS 2020

Prof. Dr. Dirk Müller

Überblick

- Motivation
- Terminologie
- *Lehmans* Systemarten und *Lehmans* Gesetze
- SW-Evolutionsprozesse
- SW-Wartung
- Zusammenfassung

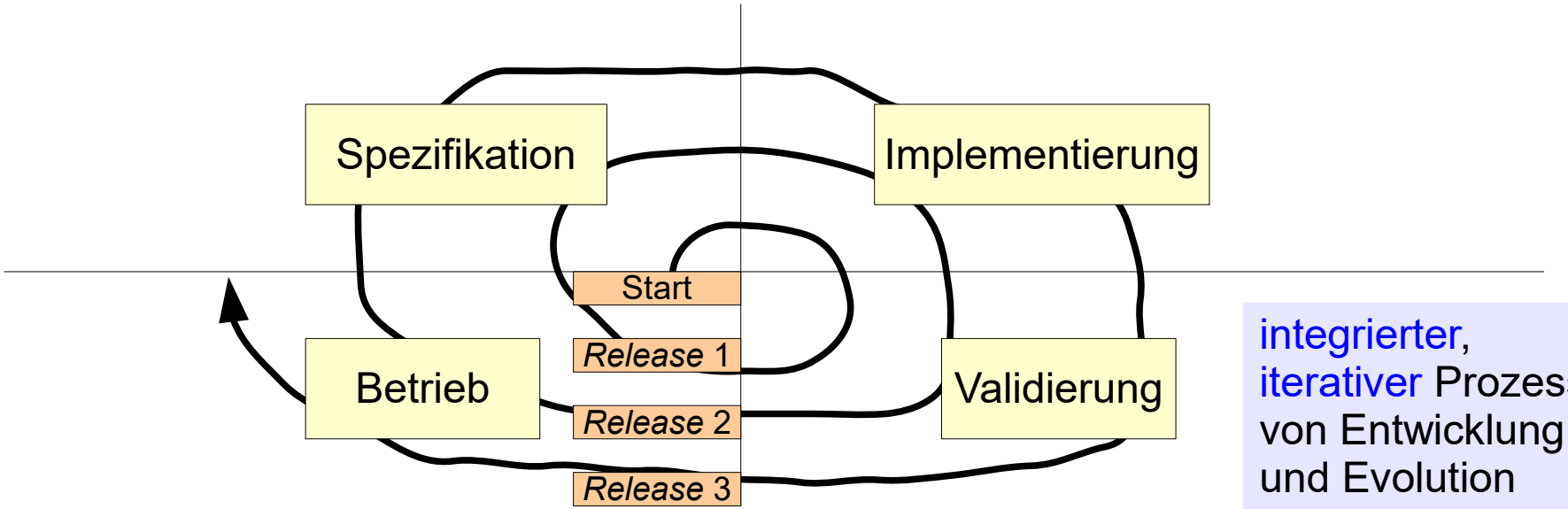
Motivation

- Softwarelebenszyklus mit wichtigem Ereignis: Auslieferung

- naiver Ansatz: Annahme, dass Entwicklung nun endet
- Entscheidung für SW-System als Langzeitentscheidung
- SW-System als wesentlicher Unternehmenswert
  - Investition in deren Anpassung, um ihren Wert zu erhalten
  - Sich ändernde Anforderungen und Debugging hören beide nicht mit der Auslieferung auf. => Entwicklung geht weiter.
- Schätzungen der Evolutionskosten: von 65% bis 90%
- typische Lebensdauern
  - Infrastruktur-Systeme wie Flugverkehrskontrolle: 30 Jahre
  - Business-Systeme: 10 Jahre

Quelle: [Som10], p. 235

SW-Evolution vs. SW-Wartung



integrierter, iterativer Prozess von Entwicklung und Evolution

- Spiralmodell impliziert, dass ein und dieselbe Organisation für SW-Entwicklung und SW-Evolution zuständig ist.
  - typisch für Standardsoftware (COTS)

SW-Evolution

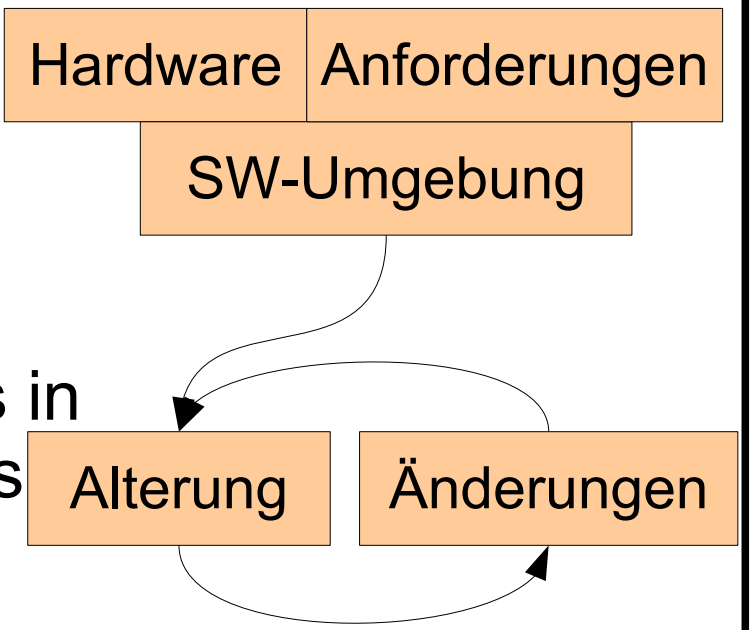
- zweite Organisation nach Auslieferung zuständig
  - typisch for Individualsoftware
  - komplizierter, z. B. Programmverständnis zusätzlich

SW-Wartung

Quelle: [Som10], p. 236

SW- vs. HW-Wartung

- kein Verschleiß bei SW
- HW-Wartung: repariere oder ersetze Komponenten, keine Änderung der Architektur oder der Nutzerschnittstelle
- SW-Wartung: Entwurfsänderungen, neue Features => neue Nutzerschnittstelle
- Relative Alterung von SW bzgl. SW-Umgebung, HW und Anforderungen
- Fixes bei alten Features oder Bugs in neuen Features können neue Fixes oder Änderungen nötig machen
- SW-Wartung sehr herausfordernd



Quellen: [1], p. 17; [2]

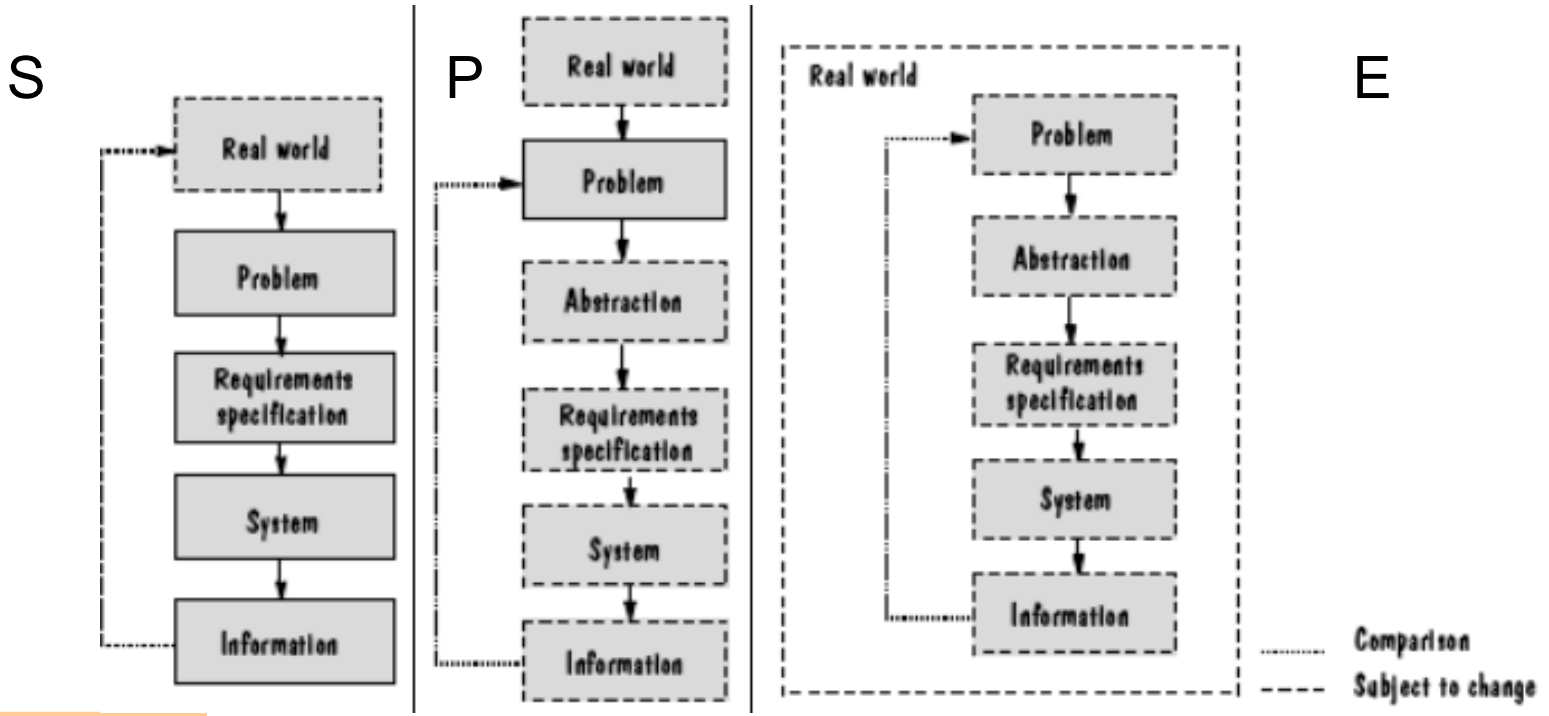
Dynamik der Programm-Evolution

- Studie zur Entwicklung von Systemen
- richtungsweisende Studien von *Lehman* und *Belady* in den 1970er and 1980er Jahren führten zu *Lehmans* Gesetzen
- besserer Begriff: Schlüsselbeobachtungen
- SW-Evolution:
  Änderung der Software, die durch Umgebungsänderung veranlasst wurde
- *“If you don't go forwards you go backwards.”*
  (Stillstand ist Rückschritt)

Quelle: [Som10], p. 240 f.

Systemarten nach *Lehman*

- S-System: formal definiert, ableitbar aus einer Spezifikation
- P-System: Anforderungen basierend auf der näherungsweisen Lösung eines Problems, aber Problem stabil; Ziel: Lösung des Problems
- E-System: eingebettet in die reale Welt, somit Veränderungen so, wie die Welt sich verändert; Ziel: Zufriedenstellung des Nutzers



Quelle: [3], p. 9

Lehmans Gesetze

	Law	Description
1	Continuing change	A program that is used in a real-world environment necessarily must change or become progressively less useful in that environment.
2	Increasing complexity	As an evolving program changes, its structure tends to become more complex. Extra resources must be devoted to preserving and simplifying the structure.
3	Large program evolution	Program evolution is a self-regulating process. System attributes such as size, time between releases and the number of reported errors is approximately invariant for each system release.
4	Organisational stability	Over a program's lifetime, its rate of development is approximately constant and independent of the resources devoted to system development.
5	Conservation of familiarity	Over the lifetime of a system, the incremental change in each release is approximately constant.
6	Continuing growth	The functionality offered by systems has to continually increase to maintain user satisfaction.
7	Declining quality	The quality of systems will appear to be declining unless they are adapted to changes in their operational environment.
8	Feedback system	Evolution processes incorporate multi-agent, multi-loop feedback systems and you have to treat them as feedback systems to achieve significant product improvement.

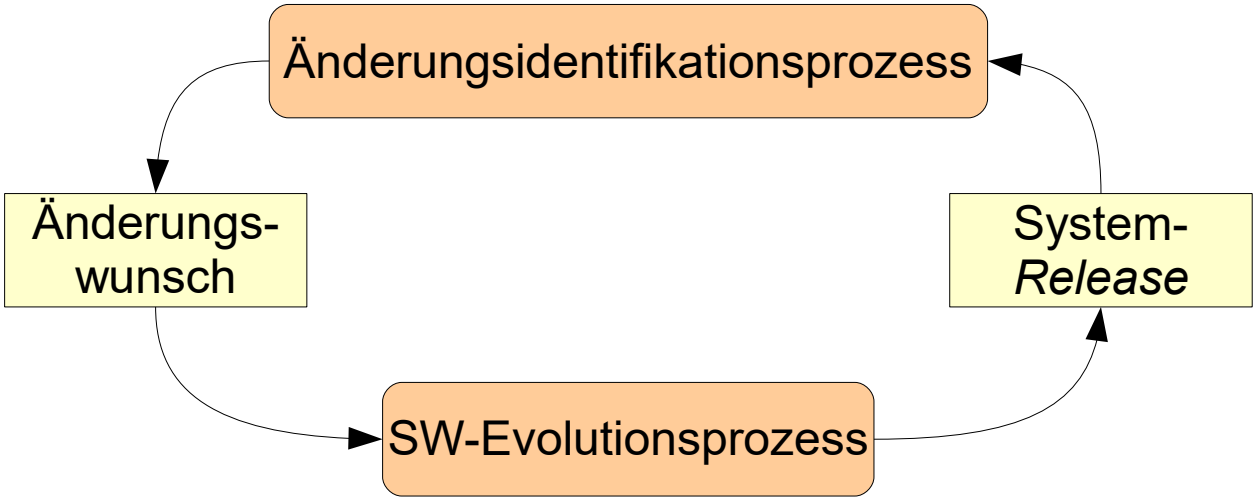
anhaltender Wandel

zunehmende Komplexität

Anwendbarkeit der Gesetze von *Lehman*

- *Lehman* beschränkte Gültigkeit auf Systeme des Typs E
- gültig für monolithische, proprietäre SW
- nur teilweise für *Open-Source*-SW gültig
- am besten auf große Systeme, die von großen Organisationen entwickelt werden, anwendbar
- Studie [4] von 1997 mit klarer Bestätigung für 5 der 8 Gesetze
  - Indizien, dass Gesetze 3 und 5 auch richtig
  - keine Datenbasis zur Beurteilung von Gesetz 7

Evolutionsprozesse



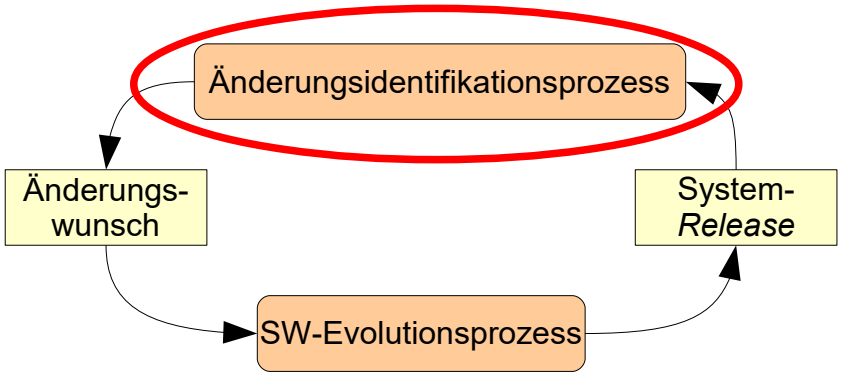
- Änderungswünsche als Triebkraft der SW-Evolution
- Bandbreite von informell (Gespräche) bis zu formalen Prozessen (Änderungsmanagement) abh. von
  - Typ und Größe der SW
  - eingesetzte Entwicklungsprozesse
  - Fähigkeiten und Erfahrungen der beteiligten Mitarbeiter

Quelle: [Som10], p. 237

Änderungsidentifikationsprozess

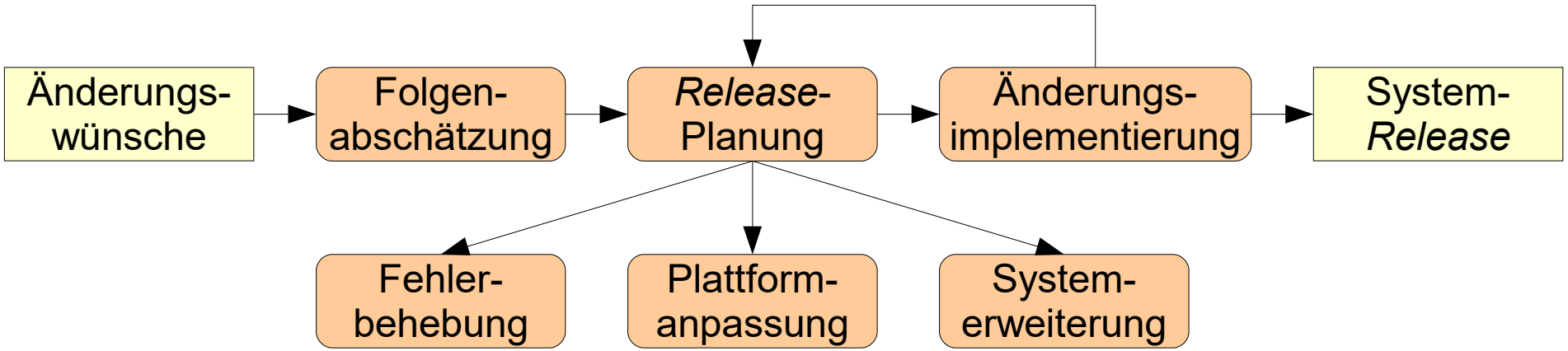
Gründe für Änderungswünsche

- Abweichung zwischen Anforderungen and Implementierung
- neue Anforderungen
- *Bug Reports* von Beteiligten
- Ideen zur SW-Verbesserung vom Systementwicklungsteam

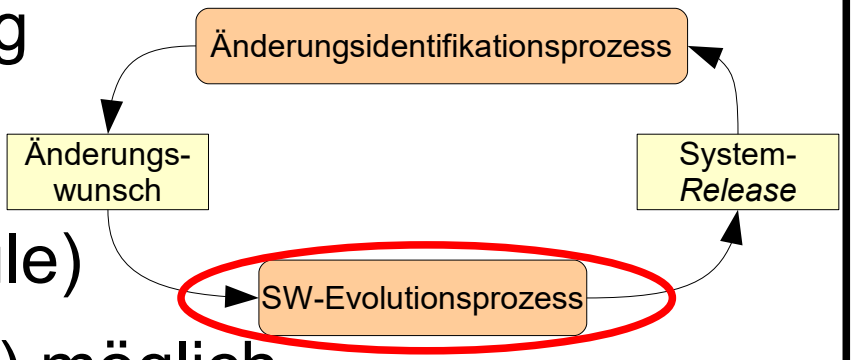


Quelle: [Som10], p. 237

Software-Evolutionsprozess

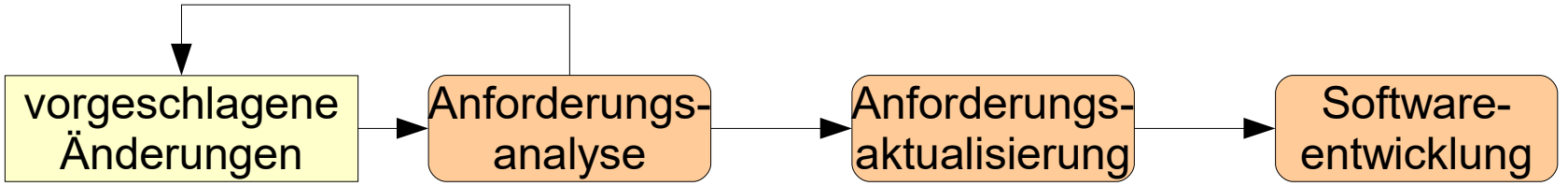


- Folgenabschätzung: Beurteilung der mit der Änderung verbundenen Risiken (Ressourcen, Aufwand, Schedule)
- negatives Ergebnis (Ablehnung) möglich
  - zu teuer
  - kein Nettonutzen: mehr neue Probleme als gelöste

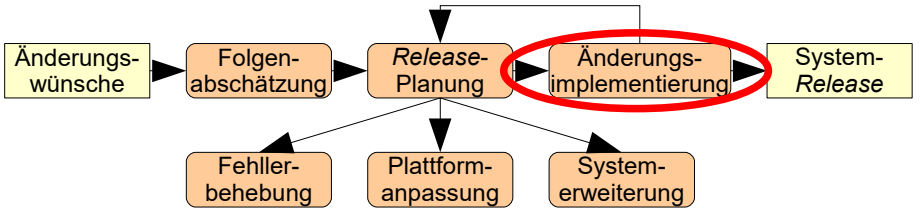


Quelle: [Som10], p. 238

Änderungsimplementierung



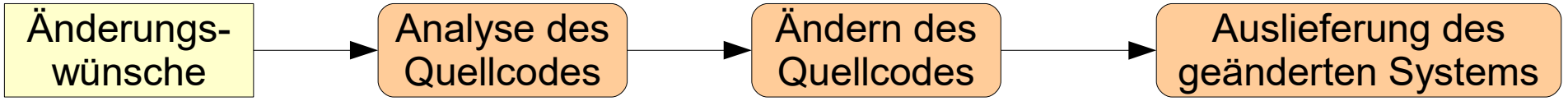
- konsistente Änderung der gesamten SW als Ziel
  Spezifikation, Entwurf, Implementierung, Dokumentation, Testfälle, etc.



- *Prototyping* als Option für schnelle Machbarkeitsstudien
- SW: Programmverständnis als weitere Herausforderung
  - Wie ist das Programm strukturiert?
  - Wie wird die Funktionalität bereitgestellt?
  - Wie könnte die vorgeschlagene Änderung das Programm beeinflussen?

Quelle: [Som10], p. 238 f.

Dringende Änderungswünsche: Notreparatur

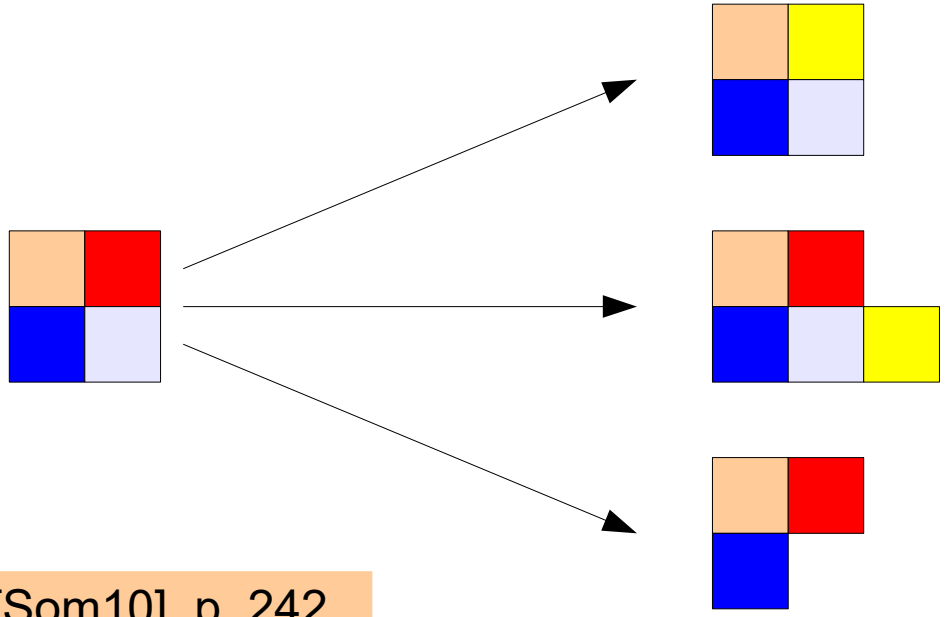


- Notlösung, sofortiger Fix wichtiger als Konsistenz
- typische Ursachen
  - ernster Fehler, der einen normalen Betrieb unmöglich macht
  - Änderungen an der Betriebsumgebung des Systems (z. B. Betriebssystem-Upgrade, also neue Funktionalität), die einen Normalbetrieb unmöglich machen
  - *Business*-Änderungen (z. B. neuer Konkurrent oder neue Gesetzgebung)
- hohes Risiko, dass das System inkonsistent bleibt/wird
  - Alterung der SW wird beschleunigt
  - zukünftige Änderungen komplizierter
  - Wartungskosten steigen

Quelle: [Som10], p. 238 f.

Softwarewartung

- Änderung eines Systems nach Auslieferung
  - typische Anwendung auf Individualsoftware
- Umfang der Änderungen mit großer Bandbreite
  - kleine, lokale Fixes
  - globale Änderungen an der gesamten Architektur
- Komponenten
  - Änderung bestehender
  - Ergänzung neuer
  - Entfernung überflüssiger



Quelle: [Som10], p. 242

Arten der Softwarewartung

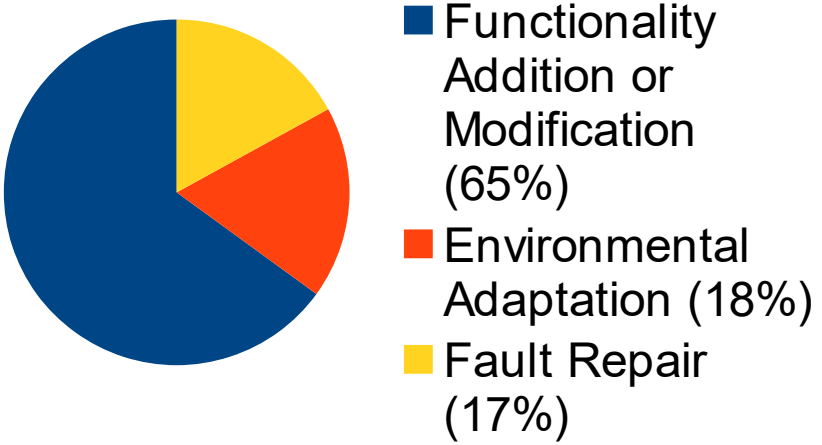
- Fehlerbehebung: korrektiv
  - Codierungsfehler: typischerweise lokal in einer Komponente
  - Entwurfsfehler: mehrere Komponenten betroffen
  - Anforderungsfehler: umfangreicher Neuentwurf des Systems
- Umgebungsanpassung: adaptiv
  - Hardware, Betriebssystem, unterstützende SW
- Funktionalitätserweiterung: perfektiv
  - *Business*- oder organisatorische Änderungen
- Vorbeugung: präventiv
  - Erkennung und Korrektur latenter Fehler bevor sie effektive Fehler werden: z. B. durch *Refactoring*

Quelle: [Som10], p. 243

Wartungsaufwand/-kosten

- 65%-90% (2/3) der IT-Budgets sind Wartungskosten
- domänenspezifische Variation
  - eingebettete, Echtzeit 80:20
  - *Business*-Systeme 50:50
- Implementierung neuer Anforderungen teurer als Fehlerbehebung
- größere Investition in gute Struktur (Verständlichkeit und Änderbarkeit) während der Entwicklung zahlt sich gewöhnlich langfristig aus

Design for Change



Quelle: [Som10], p. 244

Kostenfaktoren der Wartung

- Teaminstabilität
  - *Overhead* für Einarbeitung neuer Mitarbeiter gespart
- getrennte Vertragszuständigkeiten
  - eigene Verträge für Entwicklung und Wartung als Nachteil
  - kein Anreiz fürs Entwicklungsteam, gut wartbare SW zu schreiben
  - keine ganzheitliche Perspektive
- unzureichende Fähigkeiten der Mitarbeiter
  - Wartung mit schlechtem Ruf unter Softwaretechnikern
  - Nachwuchs mit geringer Erfahrung und unzureichendem Wissen bzgl. älterer Programmiersprachen
- Programmalterung und -struktur
  - Änderungen verschlechtern Struktur eines Programms
  - auf Effizienz getrimmt, keine order schlechte Dokumentation
  - kein oder schlechtes Konfigurationsmanagement

Quelle: [Som10], p. 245 f.

Wartungsvorhersage

- Schätzungen zur Wartung
  - Quantität (Wie viele Änderungswünsche wird es geben?)
  - Kosten (Wie hoch sind die jährlichen/totalen Wartungskosten?)
  - lokale Verteilung der Quantität und der Kosten
    (Welche Teile sind wahrscheinlich von Änderungswünschen betroffen?, Welche Teile werden von den höchsten Wartungskosten betroffen sein?)
- Quantität
  - basiert auf Anzahl und Komplexität der Relationen zur Umgebung
  - S-System: keine, P-System: wenig, E-System: viele
- Änderbarkeitsmaße und -indikatoren
  - statisch: Komplexität der Kontroll- und Datenstrukturen, Methoden- und Modulgröße
  - Prozessmetriken: negative Indikatoren falls steigende
    - Anzahl von korrektiven Änderungswünschen (z. B. 1 *Bugfix*, 2 neue *Bugs*)
    - durchschnittliche Zeit für Folgenabschätzung (mehr Komponenten betroffen)
    - durchschnittliche Zeit für Änderungsimplementierung
    - Anzahl der noch zu bearbeitenden Änderungswünsche

Quelle: [Som10], p. 246 ff.

Reengineering vs. *Refactoring*

Reengineering

- typisch für Altsysteme
- nach der Wartung eines Systems bei steigenden Wartungskosten
- gute Werkzeugunterstützung
- zurück bis zum Entwurf und zur Spezifikation

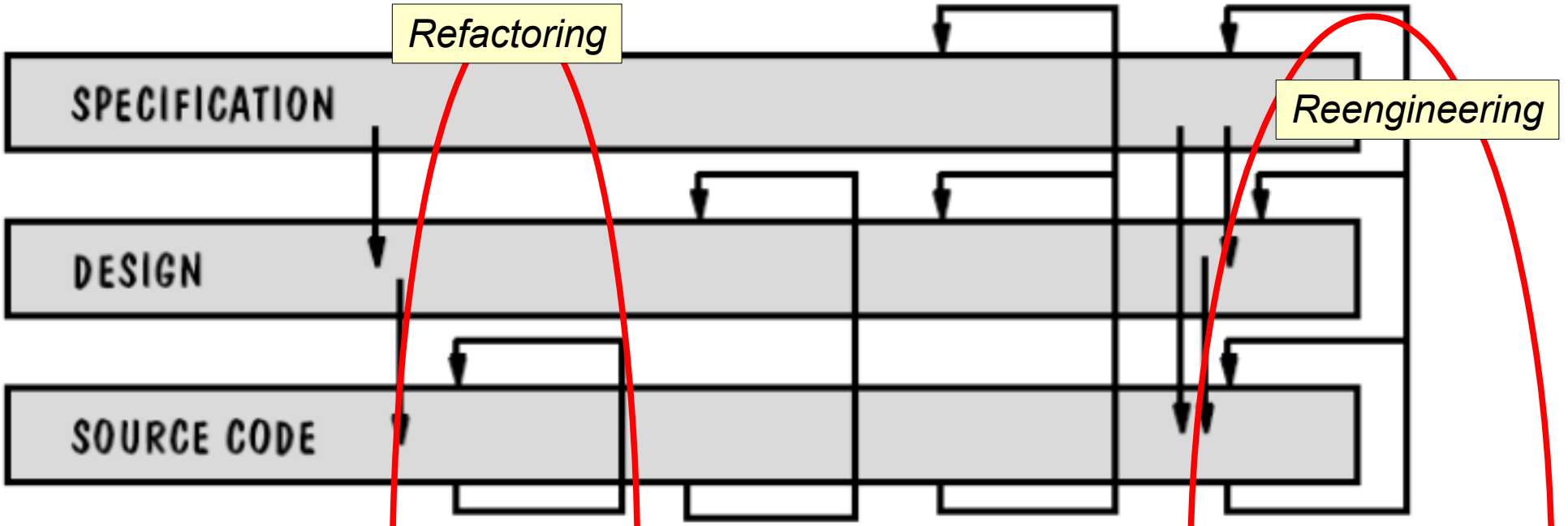
Refactoring

- vorbeugende Wartung
- kontinuierlicher Prozess der Code-Verbesserung in Entwicklung und Evolution
- angestoßen von sogenannten *Code Smells*
- Kataloge und Werkzeuge (z. B. in *Eclipse*) verfügbar
- bleibt auf der Code-Ebene

gemeinsames Ziel: Änderbarkeit verbessern,
dann SW leichter zu verstehen und zu ändern,
Rejuvenation

Quelle: [Som10], p. 250 f.

Software Rejuvenation



Forward Engineering
- forward progression through process

Restructuring
- from code
- internally represent
- iteratively simplify structure and eliminate dead code
- regenerate code

Redocumenting
- from code
- static analysis reports on structure, complexity, volume, data, etc.
- not based on software methods

Reverse Engineering
- from code
- produces design and specification based on accepted software methods
- manages representation

Reengineering
- from code
- reverse-engineer code
- forward-engineer: complete and modify representation regenerate code

Quelle: [3], p. 48

Zusammenfassung

- SW-Entwicklung und -Evolution am besten als ein integrierter, iterativer Prozess repräsentiert durch ein Spiralmodell zu sehen
- Individualsoftware: 1/3 Entwicklungskosten & 2/3 Wartungsk.
- SW-Evolutionsprozess mit Änderungswünschen, Folgen-abschätzung, *Release*-Planung und Änderungs-implementierung
- *Lehmans* Gesetze mit etwas Pessimismus, aber gültig für große monolithische proprietäre SW, die von großen Organisationen entwickelt wird
- 3 Typen von SW-Wartung: Fehlerbehebung, neue Umgebung, *Business*-Wechsel
- *Reengineering* mit *Reverse*/*Forward Engineering*:Altsysteme
- *Refactoring* als kontinuierlicher Prozess der Umstrukturierung von Code ohne sein externes Verhalten zu ändern

Literatur

[1] Frederick P. Brooks, Jr.: “The Mythical Man-Month: Essays on Software Engineering”, slides, 1997, Download am 17.06.2014,
http://www.researchgate.net/profile/Frederick_Jr/publication/220689892_The_mythical_man-month_-_essays_on_software_engineering_(2._ed.)/file/9fcfd50d5e8c033c54.pdf

[2] Parnas, D. L.: “Software Aging”, In: Int'l Conf. on Software Engineering. IEEE Computer Society Press, Sorrento, Italy 1994, p. 279–287

[3] Martin Glinz, Harald Gall: “Kapitel 12, Software Evolution und Reengineering”, lecture slides Software Engineering, WS 2005/06, Institut für Informatik, Universität Zürich,
Download am 19.06.2014,
https://files.ifi.uzh.ch/rerg/amadeus/teaching/courses/software_engineering_ws0506/Kapitel_12_EvolReeng.pdf

[4] M M. Lehman, J F. Ramil, P D. Wernick, D E. Perry, and W M. Turski. 1997. Metrics and Laws of Software Evolution - The Nineties View. In Proceedings of the 4th International Symposium on Software Metrics (METRICS '97). IEEE Computer Society, Washington, DC, USA, 20-.