Universität Zürich[UZH]

Institut für Informatik

Martin Glinz Thomas Fritz

Software Engineering

Kapitel 12

Software Evolution und Reengineering

12.1 Grundlagen

12.2 Evolution

12.3 Pflege und Änderung

12.4 Reengineering

12.5 Legacy Systeme

Braucht Software regelmäßige Wartung?

Etwa so?

Auszug aus dem Service-Handbuch für die Wartung von Software des Typs E:

- *Schleifen im Code:* Auf festen Sitz prüfen
- *Schnittstellen:* Auf Leichtgängigkeit prüfen, schmieren
- *Bildschirmmasken:* auf Abnutzung prüfen; spätestens nach 10'000 Zugriffen ersetzen

  ...

- Software verschleißt nicht
- Aber: auch Software altert:
  - Maßnahmen zur Erhaltung der Gebrauchstauglichkeit erforderlich
  - Software-Pflege

Software-Pflege (Wartung)

Pflege (Wartung, maintenance) – Modifikation und/oder Ergänzung bestehender Software durch neue Software mit dem Ziel, Fehler zu beheben, die bestehende Software an veränderte Bedürfnisse oder Umweltbedingungen anzupassen oder die bestehende Software um neue Fähigkeiten zu erweitern.

- Systematische Erhaltung der Gebrauchstauglichkeit durch kontrollierte Evolution
- Nicht nur Fehlerbehebung
- Kontinuierlicher Prozess

Evolution

Evolution (evolution) – Eine Sequenz von Veränderungen einer Einheit im Lauf der Zeit, welche bewirkt, dass die Einheit sich an Veränderungen in ihrem Umfeld anpasst. Einheiten sind Gegenstände, Lebewesen, Systeme, Prozesse, etc.

- Technische Artefakte unterliegt einer Evolution
- Software Evolution: Anpassung von Software (Komponenten oder Systemen) an ein verändertes Umfeld mit dem Ziel der Erhaltung der Gebrauchstauglichkeit

Treiber der Software-Evolution

- Aus der Benutzung von Software entstehen neue Anforderungen
- Das Umfeld ändert sich
  - Geschäftsziele und -bedürfnisse
  - Märkte, Konkurrenten
  - Regulatorisches Umfeld
- Die Technik ändert sich
  - Neue Rechner, Betriebssysteme, Treiber
  - Neue Versionen von Basissoftware, verwendeten Frameworks, etc.
  - Geänderte Schnittstellen zu Nachbarsystemen
- Fehler treten auf und müssen repariert werden
- Qualitätsmerkmale (Antwortzeiten, Durchsatz, Zuverlässigkeit, etc.) verschlechtern sich oder genügen veränderten Anforderungen nicht mehr

Veränderung von Software durch Pflege

- Pflege ist notwendige Folge der Software-Evolution
- Durch Pflege wird Software typisch größer und schlechter
- Die Software-Evolution unterliegt einer Eigendynamik
  - Pflege ist nur begrenzt steuerbar
  - Gesetze von Lehman

Drei Klassen von Software [vgl. Kapitel 13]

Klassifikation von Lehman:

- S-Typ: Vollständig durch formale Spezifikation beschreibbar.
  Erfolgskriterium: Spezifikation nachweislich erfüllt
- P-Typ: Löst ein abgegrenztes Problem.
  Erfolgskriterium: Problem zufriedenstellend gelöst
- E-Typ: Realisiert eine in der realen Welt eingebettete Anwendung.
  Erfolgskriterium: Anwender zufrieden

- Software vom S-Typ ist stabil
- Software vom P- und E-Typ ist einer Evolution unterworfen

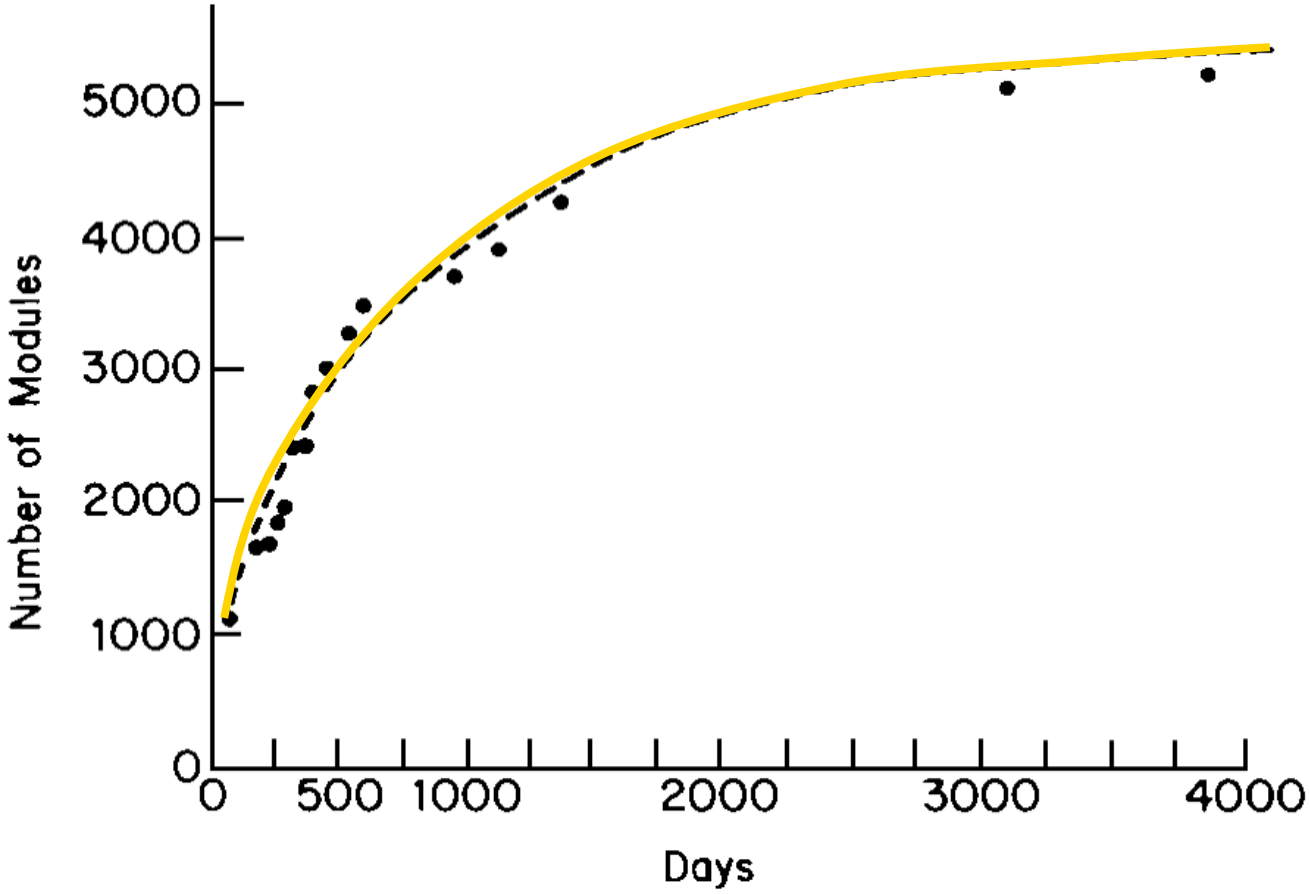
Lehmans Gesetze der Software Evolution

	Name	Brief description
I	Continuing change (1974)	*E*-type systems must be continually adapted else they become progressively less satisfactory.
II	Increasing complexity (1974)	As an *E*-type system evolves, its complexity increases unless work is done to maintain or reduce it.
III	Self regulation (1974)	The evolution process of *E*-type systems is self regulating, with a distribution of product and process measures over time that is close to normal.
IV	Conservation of organizational stability (1980)	The average effective global activity rate in an evolving *E*-type system is invariant over a product's lifetime.
V	Conservation of familiarity (1980)	During the active life of an evolving *E*-type system, the average content of successive releases is invariant.
VI	Continuing growth (1980)	The functional content of an *E*-type system must be continually increased to maintain user satisfaction with the system over its lifetime.
VII	Declining quality (1996)	Stakeholders will perceive an *E*-type system to have declining quality unless it is rigorously maintained and adapted to its changing operational environment.
VIII	Feedback system (1974–1996)	The evolution processes in *E*-type systems constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable baseline.

Quelle: Cook et al. 2005

Kontinuierliches Wachstum (Gesetz 6)

Beispiel: Betriebssystem OS 360/370 von IBM



Quelle:
Lehman und Belady (1985),
p. 307

Kontinuierliches Wachstum (Gesetz 6) – 2

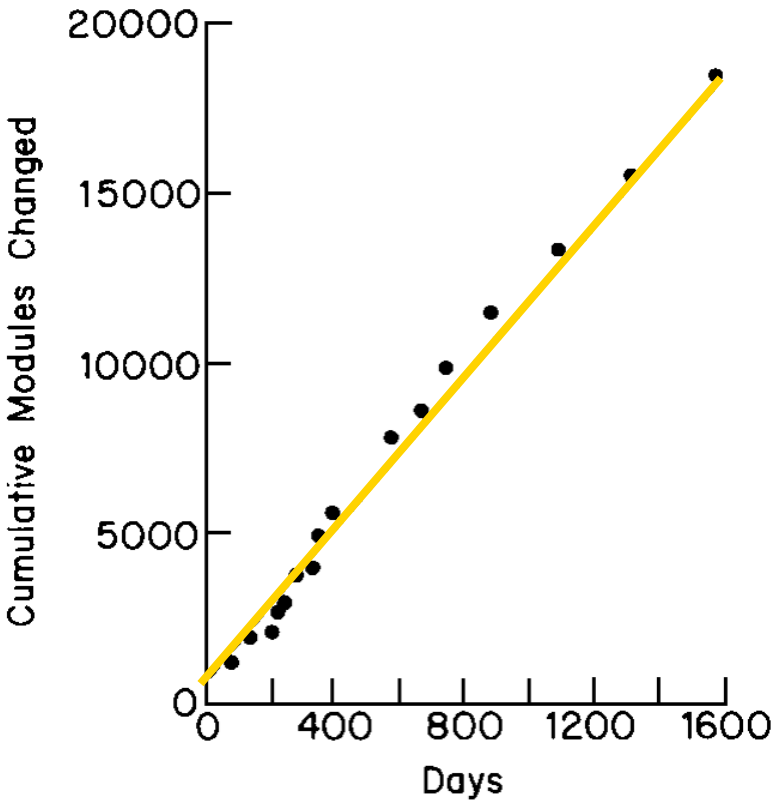
Wachstum des
Linux-Kerns 2005-2009

Kernel Version	Files	Lines
2.6.11	17,090	6,624,076
2.6.12	17,360	6,777,860
2.6.13	18,090	6,988,800
2.6.14	18,434	7,143,233
2.6.15	18,811	7,290,070
2.6.16	19,251	7,480,062
2.6.17	19,553	7,588,014
2.6.18	20,208	7,752,846
2.6.19	20,936	7,976,221
2.6.20	21,280	8,102,533
2.6.21	21,614	8,246,517
2.6.22	22,411	8,499,410
2.6.23	22,530	8,566,606
2.6.24	23,062	8,859,683
2.6.25	23,813	9,232,592
2.6.26	24,273	9,411,841
2.6.27	24,356	9,630,074
2.6.28	25,276	10,118,757
2.6.29	26,702	10,934,554
2.6.30	27,911	11,560,971

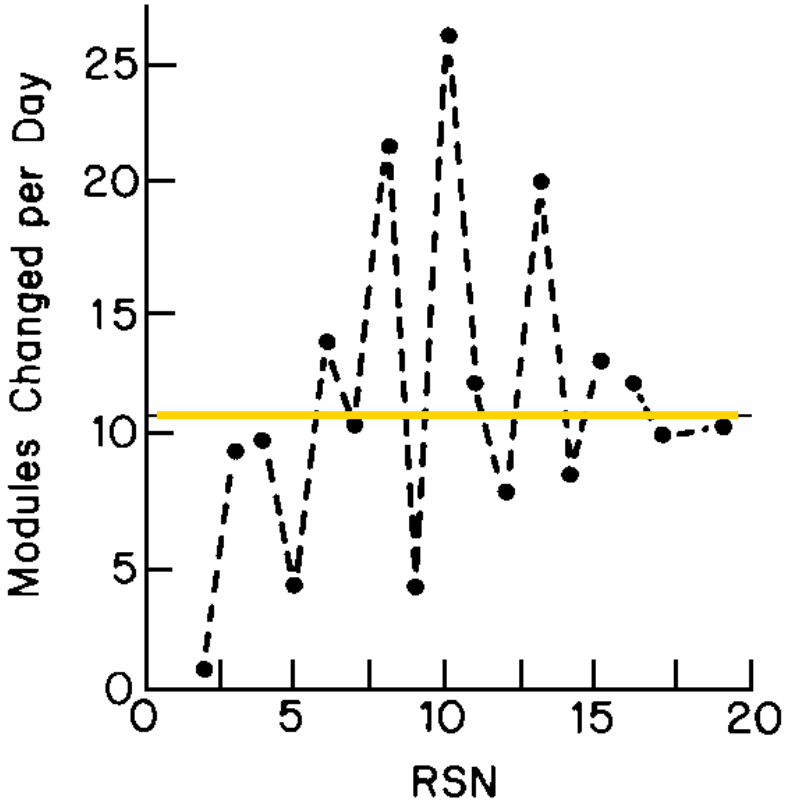
Quelle: Greg Kroah-Hartman, Jonathan Corbet, Amanda McPherson (2009) *Linux Kernel Development*. The Linux Foundation.

Statistische Invarianz (Gesetze 4 und 5)

Kumulierte Anzahl geänderter Module



Änderungsrate über der Folge der Releases

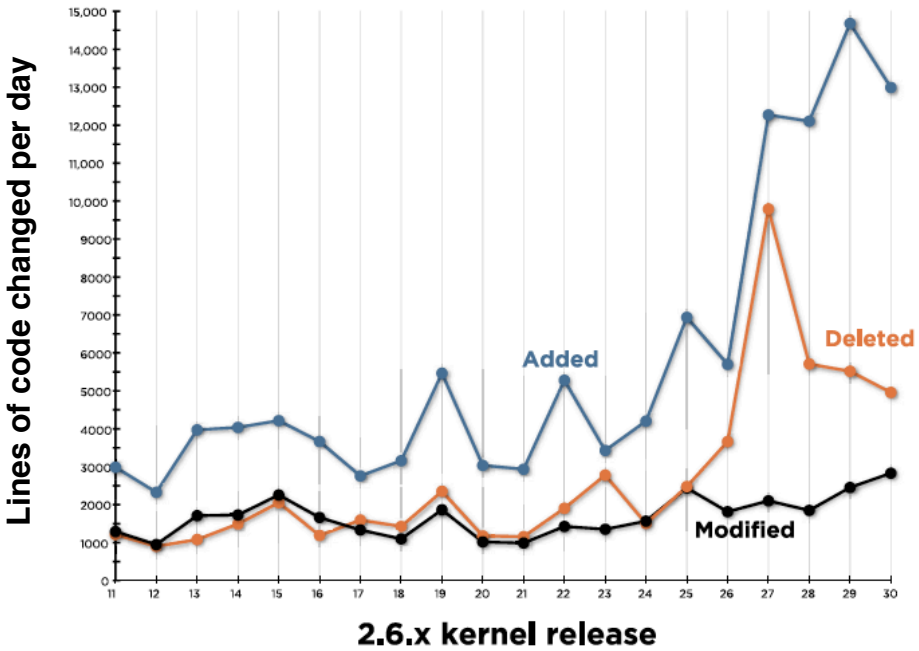


Quelle: Lehman und Belady (1985), p. 416

Statistische Invarianz – 2

Änderungsraten im Linux-Kern 2005-2009

- Über längere Zeit statistisch invariant
- Aber mit Sprungstellen



Kernel Version	Changes (patches)
2.6.11	3,616
2.6.12	5,047
2.6.13	3,904
2.6.14	3,627
2.6.15	4,959
2.6.16	5,369
2.6.17	5,727
2.6.18	6,323
2.6.19	6,685
2.6.20	4,768
2.6.21	5,016
2.6.22	6,526
2.6.23	6,662
2.6.24	9,836
2.6.25	12,243
2.6.26	9,941
2.6.27	10,628
2.6.28	9,048
2.6.29	11,678
2.6.30	11,989

Quelle: Greg Kroah-Hartman, Jonathan Corbet, Amanda McPherson (2009) *Linux Kernel Development*. The Linux Foundation.

Validität der Lehman-Gesetze

Empirische Basis der Lehman'schen Gesetze: große Software-Systeme in den 1960er und 1970er Jahren

Validität aus heutiger Sicht?

I	Continuing change	Beschreiben fundamentale Phänomene der Software-Evolution
VII	Declining quality	
VIII	Feedback system	
II	Increasing complexity	Vor allem bei großen Systemen und Systemlandschaften, gelten nicht universell
VI	Continuing growth	
III	Self-regulation	Nicht als „Gesetze", sondern als häufig beobachtete Phänomene interpretieren
IV	Conservation of org. stability	
V	Conservation of familiarity	

12.1 Grundlagen

12.2 Evolution

12.3 Pflege und Änderung

12.4 Reengineering

12.5 Legacy Systeme

Systematische Pflege (Wartung, Maintenance)

Wie kann bestehende, in Betrieb befindliche Software kontrolliert verändert werden?

- Von welcher Art sind die Pflegemaßnahmen?
- Wie sieht der Änderungsprozess aus?
- Was kostet das Ganze?

Arten von Pflege

- Fehlerbehebung
  - Behebung von im Betrieb aufgetretenen Fehlern
- Anpassung
  - Adaptieren bestehender Software an eine veränderte technische Umgebung
- Erweiterung
  - Hinzufügen neuer Fähigkeiten zwecks Erfüllung veränderter Anforderungen

Klassifikation von Pflegemaßnahmen

[ISO/IEC/IEEE 14764:2006]

- Korrektive Wartung (Corrective maintenance)
  - Korrektur aufgetretener Fehler
- Adaptive Wartung (Adaptive maintenance)
  - Erhaltung der Gebrauchstauglichkeit in einer veränderten Umgebung
- Perfektionierende Wartung (Perfective maintenance)
  - Maßnahmen zur Verbesserung der Gebrauchstauglichkeit
- Präventive Wartung (Preventive maintenance)
  - Suche und Behebung von bisher nicht aufgetretenen Fehlern

Der Pflegeprozess

Kontinuierlicher Prozess mit folgenden Teilaufgaben:

- Erfassung der Bedürfnisse der Interesseneigner (Stakeholder)
  - reaktiv: Problemmeldewesen (vgl. Kapitel 20)
  - proaktiv: wohin geht die Reise (Märkte, Umwelt, Konkurrenz,...)
- Analyse der gemeldeten Probleme / erkannten Bedürfnisse
- Planung der Pflegearbeiten
- Durchführung der anstehenden Arbeiten
- Auslieferung der Ergebnisse (als neue Releases, vgl. Kapitel 20)
- Verwaltung der Artefakte (Problemmeldungen, Arbeitspakete, Lieferungen..., vgl. Kapitel 20)

Rollen in der Software-Pflege

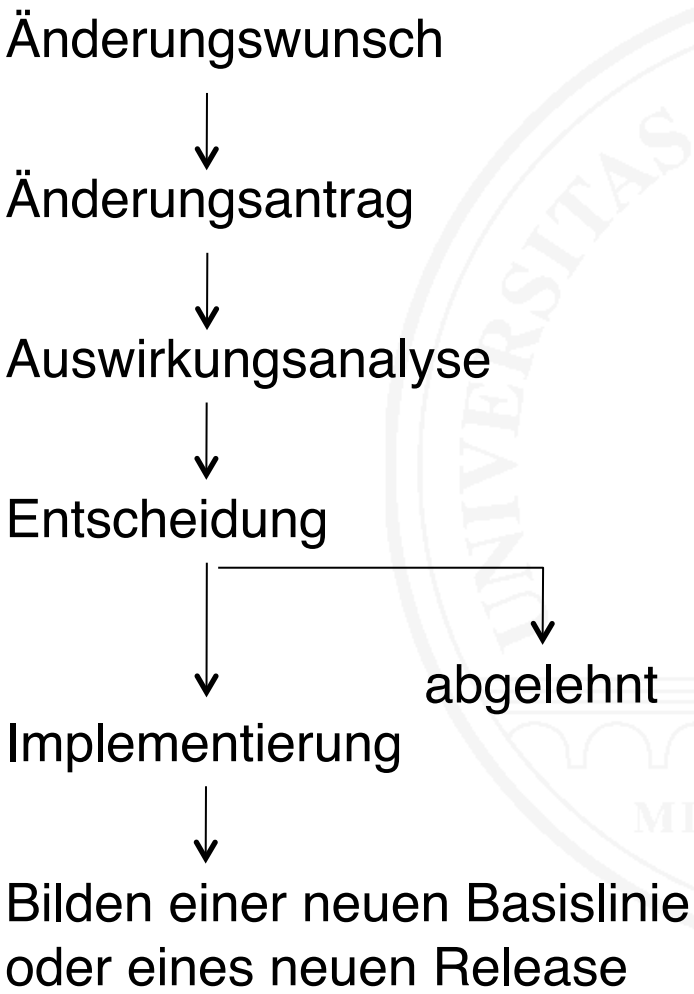
- Produktmanager / Produkteigner
  - Verantwortlich für die Erhaltung der Gebrauchstauglichkeit seines Produkts (reaktiv und proaktiv)
  - Organisiert und koordiniert Pflegemaßnahmen
  - Verantwortlich für die Releaseplanung
- Pflegeteam
  - Führt Pflegemaßnahmen durch
- Interesseneigner (Stakeholder) – Benutzer, Kunden, Betreiber, Entwickler, ...
  - Melden Probleme und Verbesserungswünsche

Aufgaben des Pflegeteams

- Das zu pflegende System verstehen
- Probleme lokalisieren
- Information in bestehender Software und ihrer Dokumentation finden
- Code und Dokumentation ändern und/oder erweitern
- Code und/oder Dokumentation restrukturieren
- Nicht mehr benötigten Code entfernen bzw. stilllegen
- Problemmeldungen und Problembehebung systematisch verfolgen und verwalten

Änderungsprozess in der Software-Pflege

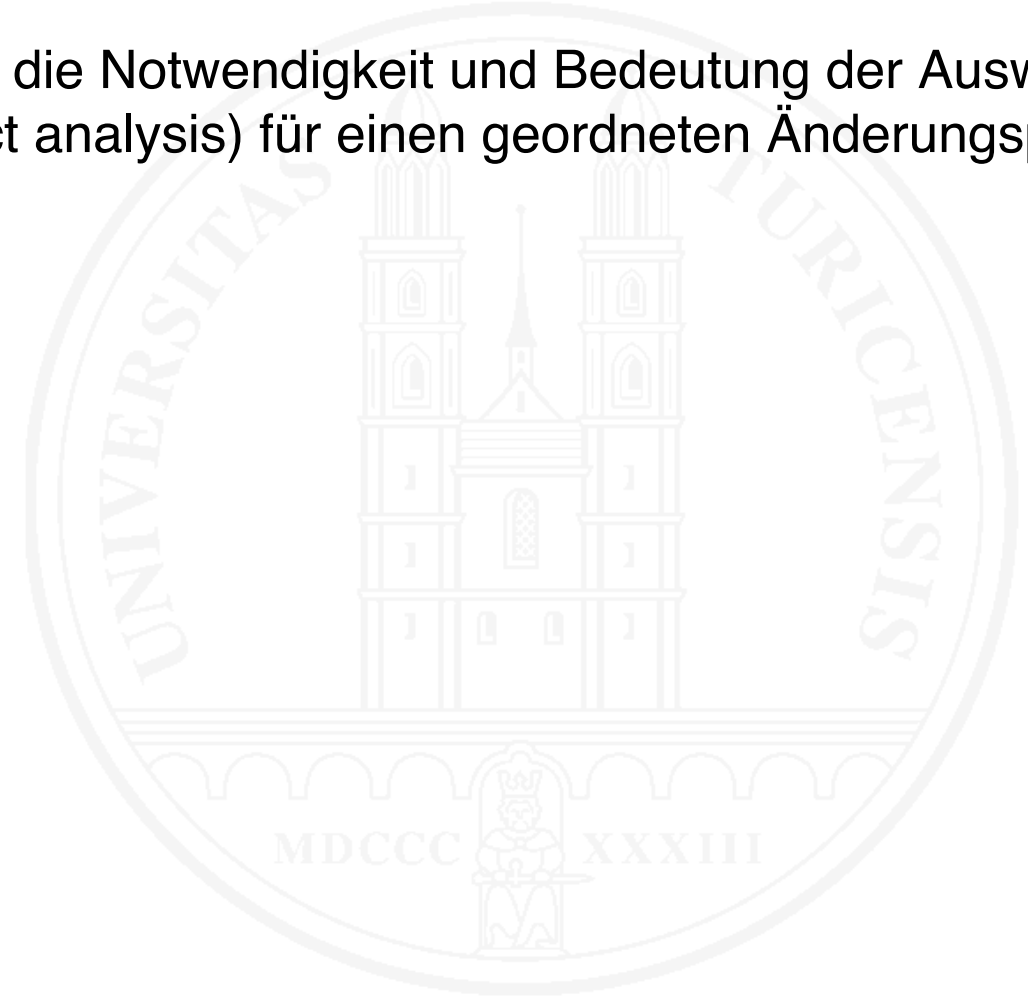
[vgl. Kapitel 20]



- Beispiel: Kunde will eine bestehende Systemfunktion ändern
- Formular auszufüllen
- Machbar? Auswirkung auf vorhandene Artefakte? Kosten und Termine?
- Durch Produktmanager, Produkteigner oder Change Control Board
- Auftrag an Pflegemitarbeiter; ggf. Änderung weiterer Artefakte
- Formeller Abschluss der Änderung

Mini-Übung 12.1

Begründen Sie die Notwendigkeit und Bedeutung der Auswirkungs-analyse (impact analysis) für einen geordneten Änderungsprozess.



Abwicklung von Änderungen

Verschiedene Formen möglich:

- Selbstorganisiert: Pflegeteam arbeitet Liste der Problemmeldungen selbständig ab
- Projektartig: Produkteigner bildet Arbeitspakete und weist diese Mitgliedern des Pflegeteams zur Bearbeitung zu
- Scrum-artig:
  - Produkteigner führt Auftragsbestand (product backlog) auf Basis der Problemmeldungen
  - Abarbeitung in Iterationen fester Länge
  - Planungsspiel zur Festlegung des Inhalts jeder Iteration

Pflegekosten (vgl. Kapitel 15.5)

- Faustregel für die Kostenverteilung von Pflegekosten:
  - 60% Verbesserungen
  - 20% Anpassungen
  - 20% Fehlerbehebung

- Faustregel über die Verteilung der verschiedenen Pflegearten
  - Korrektiv 21%
  - Adaptiv 25%
  - Perfektionierend 50%
  - Präventiv 4%

Messungen im Pflegeprozess

- Fortlaufende Messungen geben Hinweise zur Pflegbarkeit des Systems/Produkts und zur Qualität des Pflegeprozesses
- Typische Messgrößen:
  - Anzahl Fehlermeldungen pro Zeiteinheit (1)
  - Anzahl andere Problemmeldungen pro Zeiteinheit (2)
  - Mittlerer Aufwand zur Behebung eines Fehlers (3)
  - Mittlerer Aufwand zur Behebung anderer Probleme (4)
  - Anzahl offener Problemmeldungen (5)

(1) und (3) geben Hinweise auf sich verbessernde/verschlechternde Pflegbarkeit des Systems im Verlauf der Evolution

(2) gibt Hinweise auf das Tempo der Evolution

(3), (4), (5) geben Hinweise auf die Prozessqualität

Prognose von Pflegeaufwendungen

- Projekt/Produkt-Repository: enthält komplette Änderungsgeschichte des gesamten Produkts
- Durch Repository-Analyse lassen sich Informationen gewinnen, welche für die Prognose zukünftiger Pflege-Aufwendung taugen
- Beispiele:
  - Stabile vs. volatile Module
  - Besonders fehlerträchtige Module
  - Module, die typisch mitzuändern sind, wenn Modul x geändert wird

12.1 Grundlagen

12.2 Evolution

12.3 Pflege und Änderung

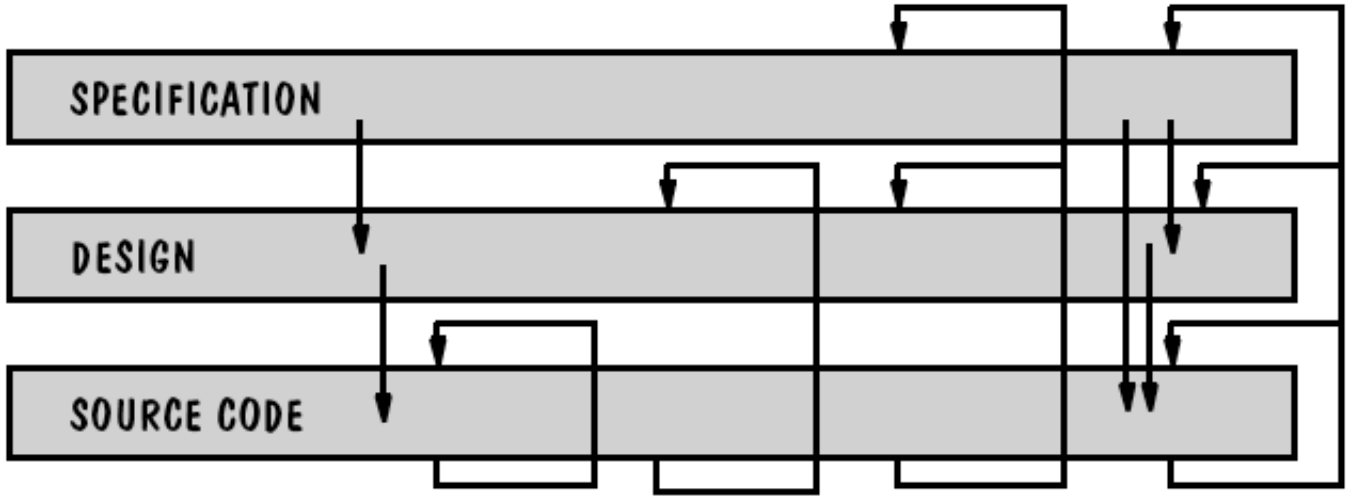
12.4 Reengineering (Folien von Harald Gall)

12.5 Legacy Systeme

Software Rejuvenation

- Redocumentation: static analysis adds more information
- Restructuring: transform to improve code structure
- Reverse engineering: recreate design and specification information from the code
- Reengineering: reverse engineer and then make changes to specification and design to complete the logical model; then generate new system from revised specification and design

Taxonomy of software rejuvenation



Forward Engineering
- forward progression through process

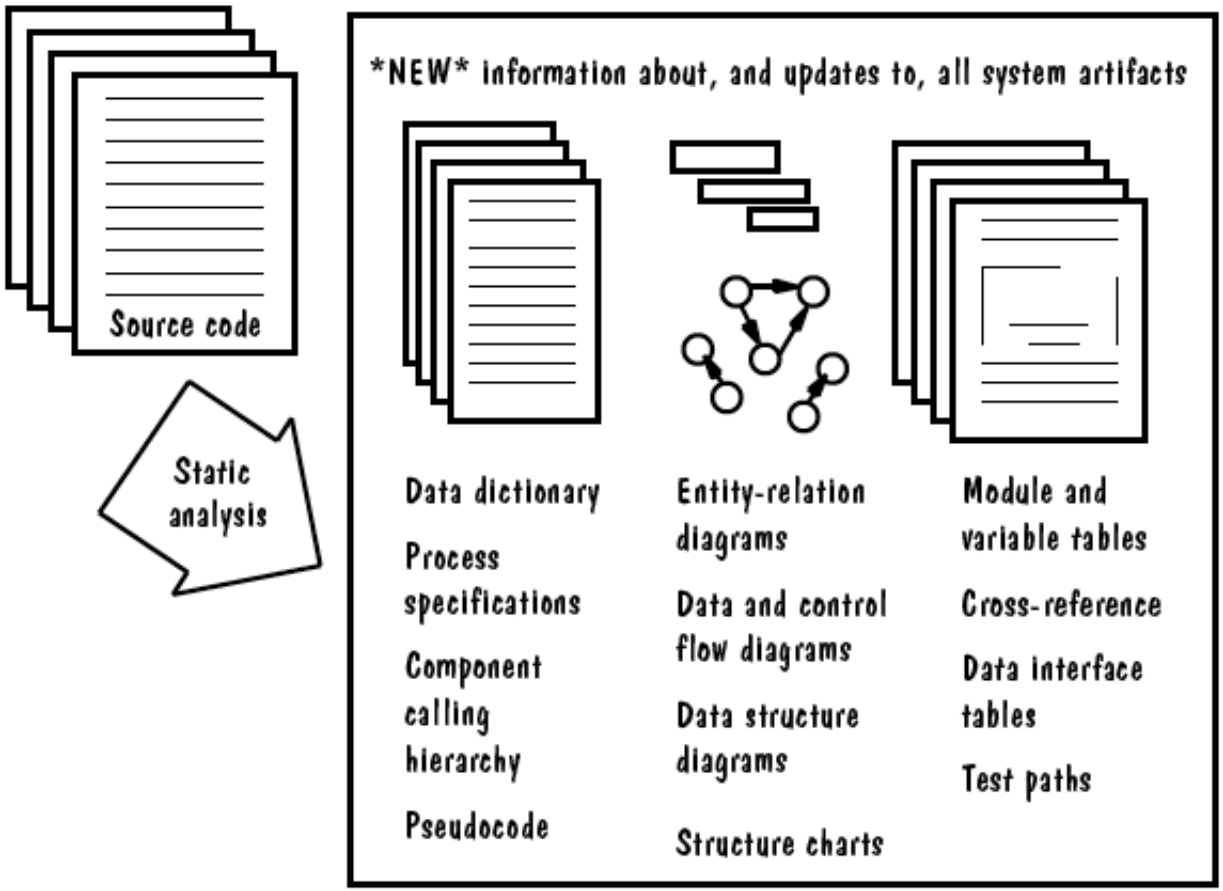
Restructuring
- from code
- internally represent
- iteratively simplify structure and eliminate dead code
- regenerate code

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

Reverse Engineering



Redocumentation

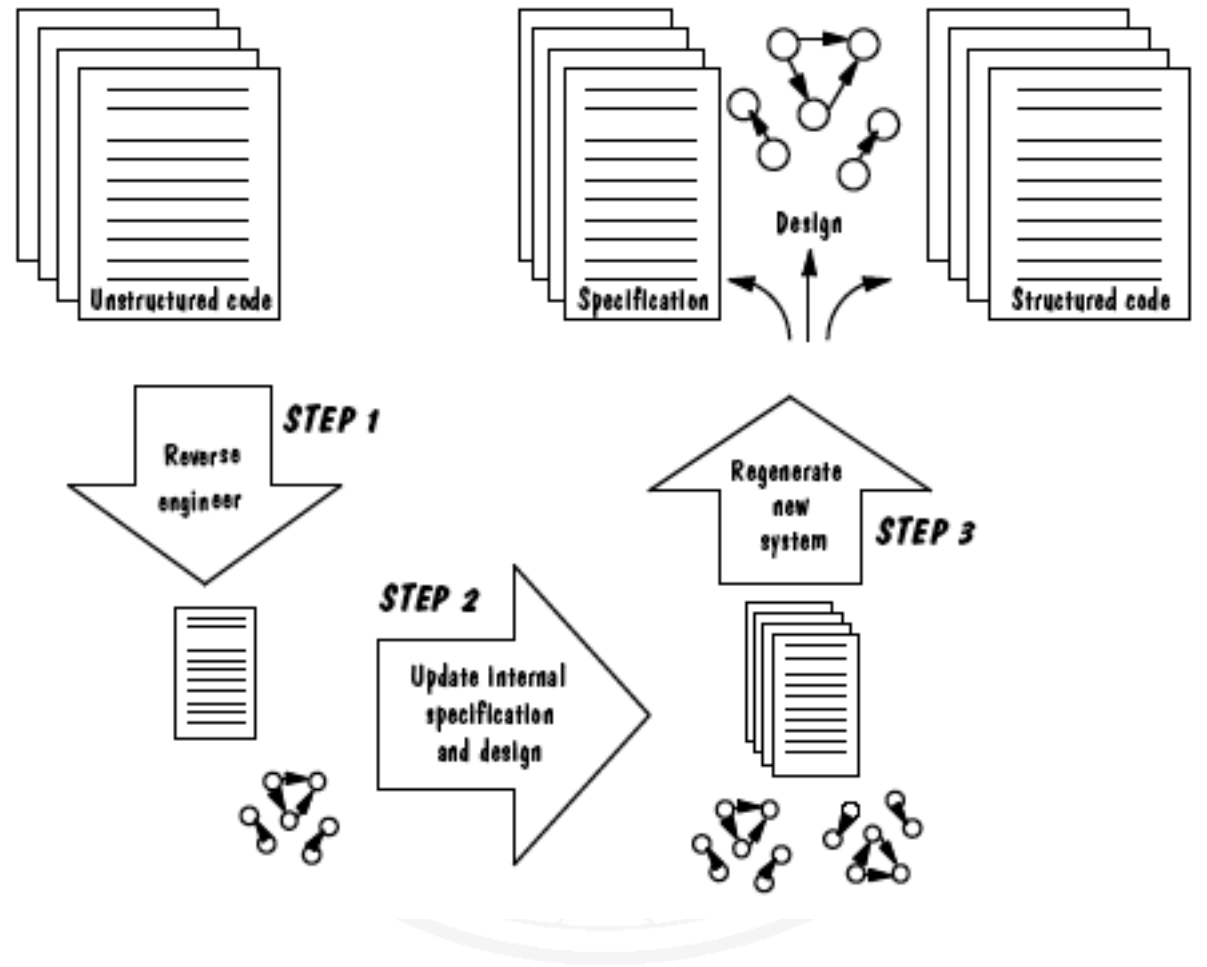
Output may include:

- component calling relationships
- data-interface tables
- data-dictionary information
- data flow tables or diagrams
- control flow tables or diagrams
- pseudocode
- test paths
- component and variable cross-references

Reengineering

- Restructuring or re-writing part or all of a legacy system plus changing its functionality according to new requirements
- Applicable where some but not all sub-systems of a larger system require frequent maintenance.
- Reengineering involves adding effort to make them easier to maintain. The system may be re-structured and re-documented.
- = Reverse Engineering + Delta + Forward Engineering

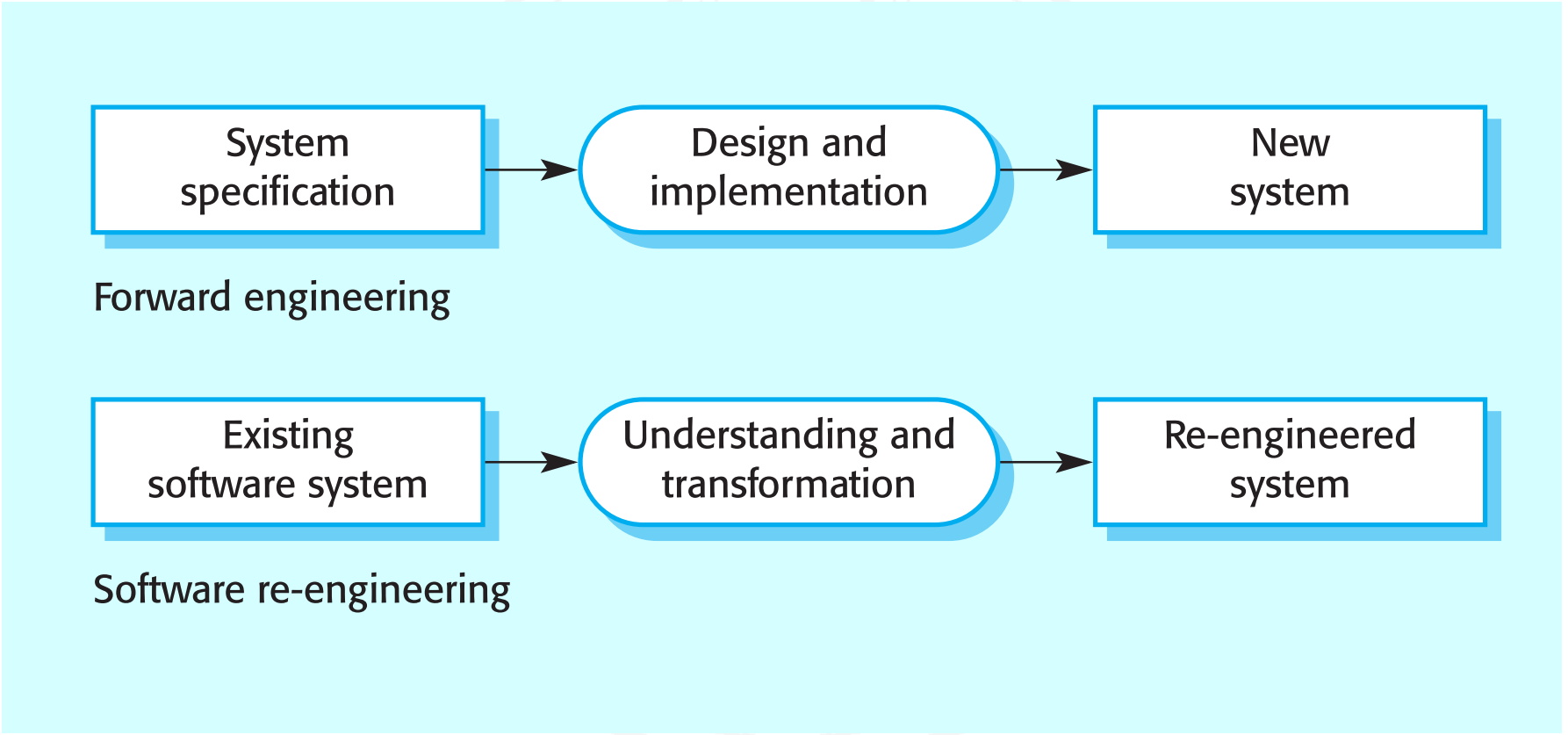
Reengineering



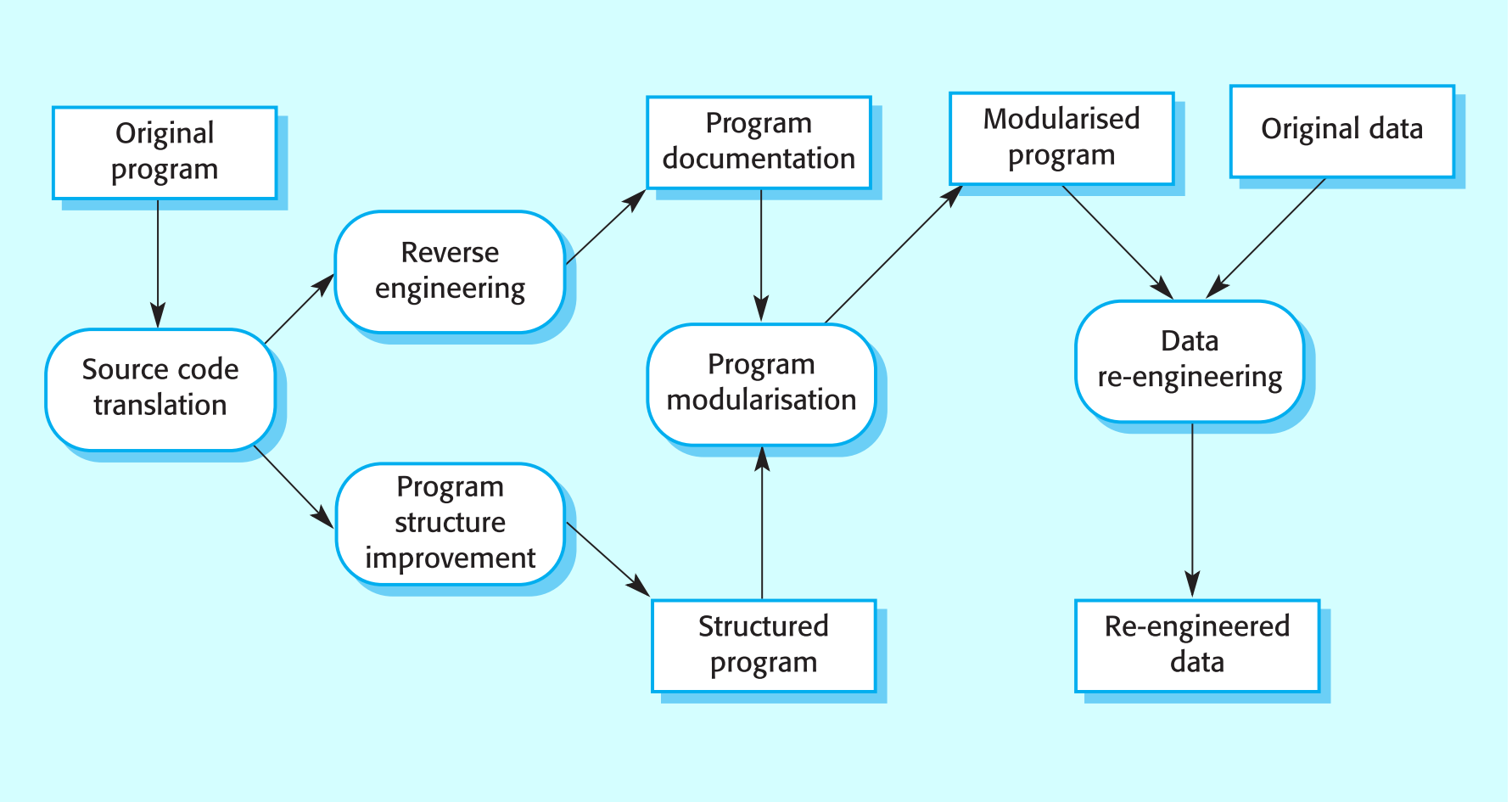
Advantages of Reengineering

- Reduced risk
  - There is a high risk in new software development. There may be development problems, staffing problems and specification problems.
- Reduced cost
  - The cost of re-engineering is often significantly less than the costs of developing new software.
- e.g. Object-oriented Reengineering Patterns

Forward and Re-Engineering



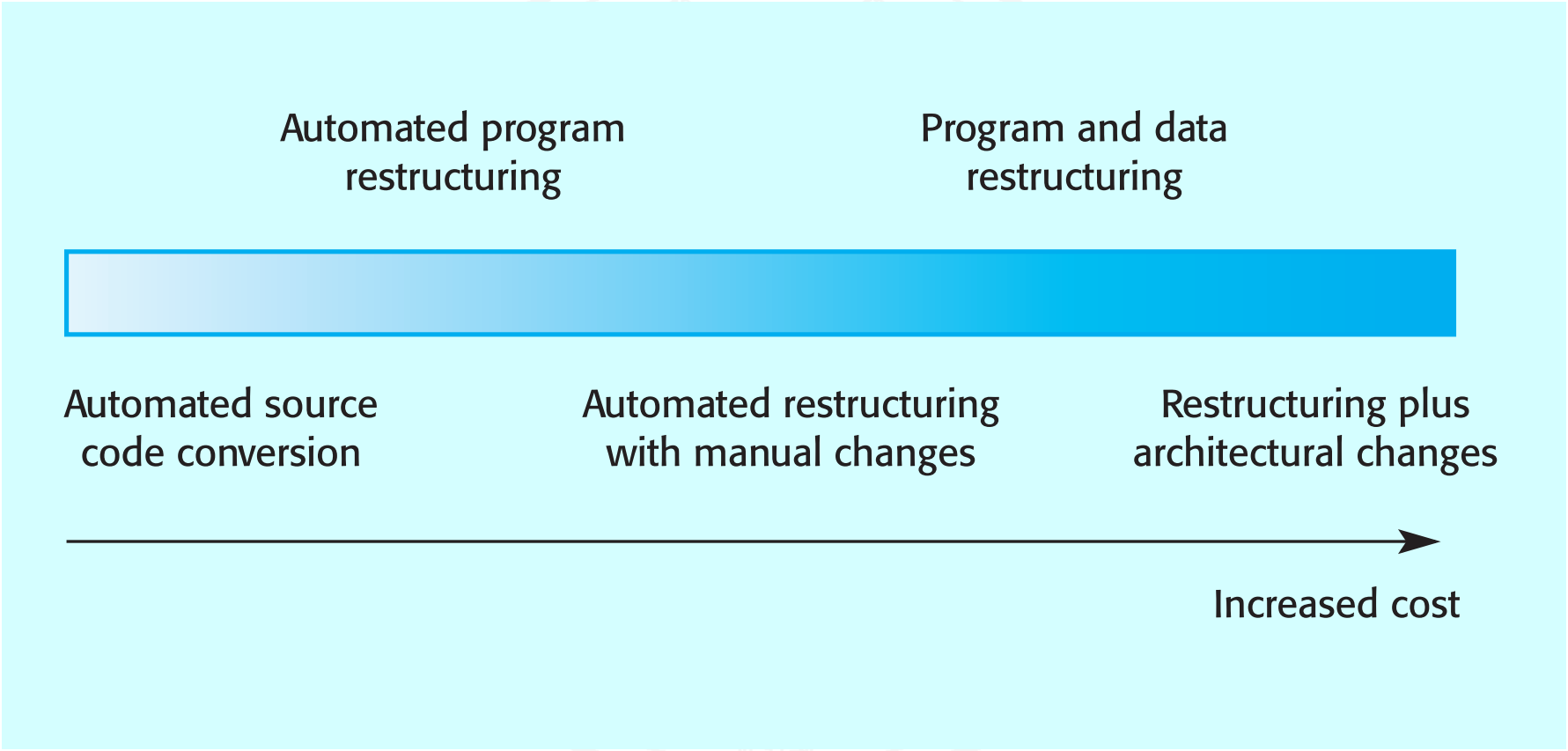
The Reengineering process



Reengineering process activities

- Source code translation
  Convert code to a new language
- Reverse engineering
  Analyze the program to understand it
- Program structure improvement
  Restructure automatically for understandability
- Program modularization
  Reorganize the program structure
- Data reengineering
  Clean-up and restructure system data

Reengineering approaches



Reengineering cost factors

- The quality of the software to be reengineered
- The tool support available for reengineering
- The extent of the data conversion required
- The availability of expert staff for reengineering.
  - This can be a problem with old systems based on technology that is no longer widely used.

12.1 Grundlagen

12.2 Evolution

12.3 Pflege und Änderung

12.4 Reengineering

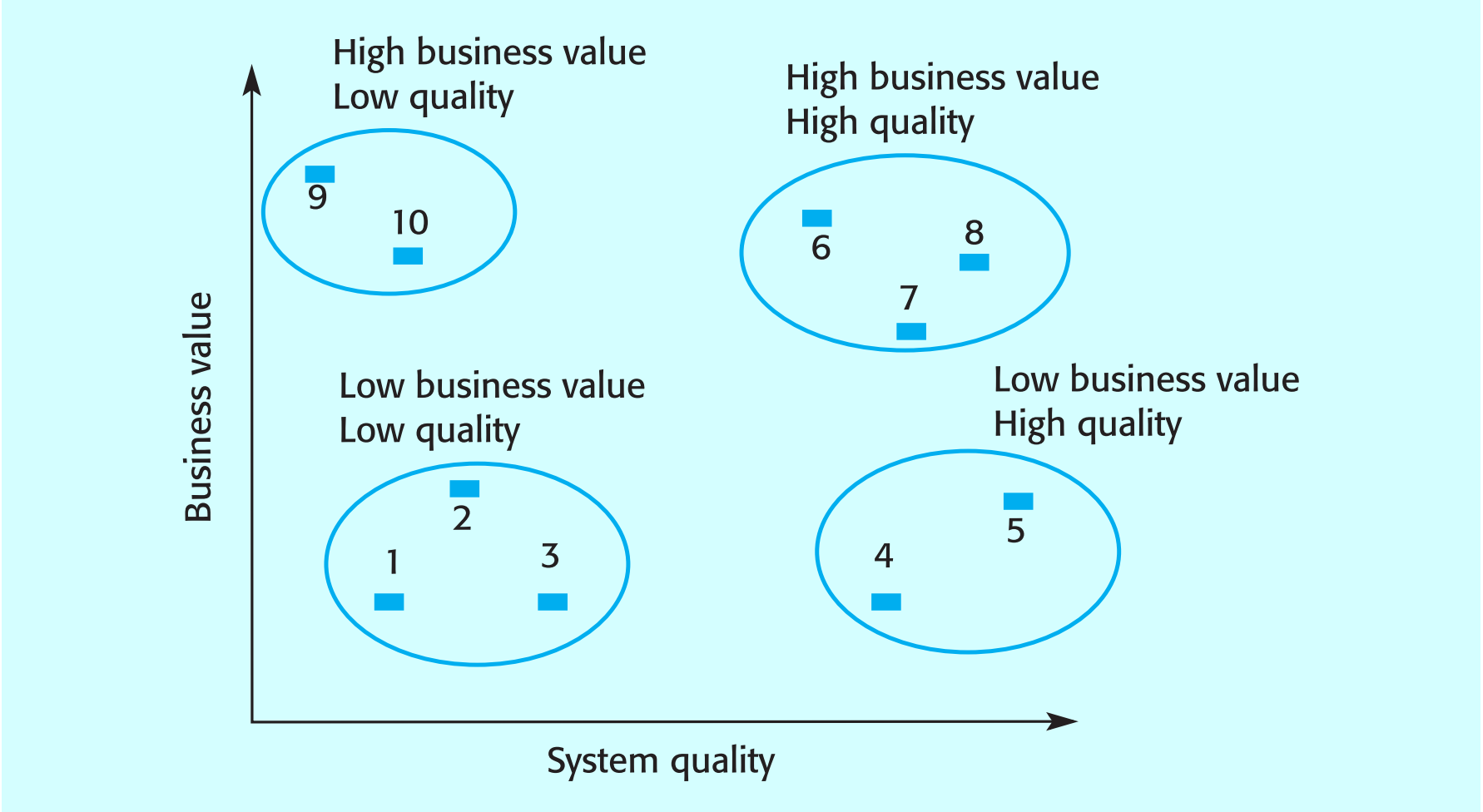
12.5 Legacy Systeme (Folien teilweise von Harald Gall)

Legacy system evolution

Legacy System – An old system, typically outdated with respect to technology and/or supported processes, which is still in operational use

- Maintaining legacy systems is difficult and expensive
- Organisations that rely on legacy systems must choose a strategy for evolving these systems
  - Shutdown: scrap the system completely and modify business processes so that it is no longer required
  - Continue maintaining the system
  - Transform the system by re-engineering to improve its maintainability
  - Replace the system with a new system
- The strategy chosen should depend on system quality and business value

System quality and business value



Strategies for evolving legacy systems

- Low quality, low business value
  - These systems should be scrapped.
- Low quality, high business value
  - These make an important business contribution but are expensive to maintain. Should be re-engineered or replaced if a suitable system is available.
- High quality, low business value
  - Replace with COTS, scrap completely or maintain.
- High quality, high business value
  - Continue in operation using normal system maintenance.

Business value assessment

- Assessment should take different viewpoints into account
  - System end-users;
  - Business customers;
  - Line managers;
  - IT managers;
  - Senior managers.
- Interview different stakeholders and collate results.

System quality assessment

- Business process assessment
  - How well does the business process support the current goals of the business?
- Environment assessment
  - How effective is the system's environment and how expensive is it to maintain?
- Application assessment
  - What is the quality of the application software system?

Business process assessment

- Use a viewpoint-oriented approach and seek answers from system stakeholders
  - Is there a defined process model and is it followed?
  - Do different parts of the organisation use different processes for the same function?
  - How has the process been adapted?
  - What are the relationships with other business processes and are these necessary?
  - Is the process effectively supported by the legacy application software?
- Example - a travel ordering system may have a low business value because of the widespread use of web-based ordering.

Environment assessment

- System supplier
  - Is there a stable supplier providing maintenance for the system?
- Hardware and support software required to run the system
  - How old?
  - Failure rate?
  - Maintenance contract costs?
  - Support software license costs?
  - Amount of local support required?
- Infrastructure and neighboring systems
  - Operating system and other licensed software required to run the system still available and maintained by supplier?
  - Any problems in interfacing the system with other systems?

Application assessment

- Satisfaction of users with respect to
  - functionality?
  - performance?
  - data quality?
- Source code
  - Still available?
  - How understandable and how well documented?
  - Under configuration control?
  - Environment required for making source code changes still available?
  - Tests / test data available?
- Maintainers: How many people...
  - ... have the knowledge to maintain the system?
  - ... know exactly what the system actually does?

Literatur

S. Cook, R. Harrison, M. M. Lehman, and P. Wernick (2006). Evolution in Software Systems: Foundations of the SPE Classification Scheme. *Journal of Software Maintenance and Evolution* **18**(1):1–35.

S. Demeyer, S. Ducasse, O. Nierstrasz (2003). *Object-Oriented Reengineering Patterns*. Morgan-Kaufmann. http://www.iam.unibe.ch/~scg/OORP/

M.W. Godfrey, D.M. German (2013). On the evolution of Lehman's Laws. *Journal of Software: Evolution and Process*. Published online November 2013, DOI: 10.1002/smr.1636.

ISO/IEC/IEEE (2006). Software Engineering – Software Life Cycle Processes – Maintenance. ISO/IEC/IEEE Standard 14764:2006, 2nd edition.

M.M. Lehman (1980). Programs, Life Cycles, and Laws of Software Evolution. *Proceedings of the IEEE* **68**(9):1060–1076. (Nachgedruckt als Kapitel 19 in Lehman und Belady 1985).

M.M. Lehman, L.A. Belady (Hrsg.) (1985). *Program Evolution: Processes of Software Change*. London: Academic Press.

T. Mens, S. Demeyer (Hrsg.) (2008). Software Evolution. Berlin: Springer.