

**XION IT SYSTEMS**

AKTIENGESELLSCHAFT

Dresdnerstraße 81-85/8.Stock  
A-1200 Wien

Tel: 0664-8242-600

E-mail: office@xion.at

Web: xion.at

Festnetz: +43/1/333 91 99-0

Fax: +43/1/333 91 99-199

x i o n . it systems ag



# Software Wartung und Evolution

*Dipl.-Ing. Dr. techn. Johannes Weidl-Rektenwald*  
*Xion IT Systems AG*

**XION IT SYSTEMS**

AKTIENGESELLSCHAFT

Dresdnerstraße 81-85/8.Stock  
A-1200 Wien

Tel: 0664-8242-600

E-mail: office@xion.at

Web: xion.at

Festnetz: +43/1/333 91 99-0

Fax: +43/1/333 91 99-199

x i o n . it systems ag



# Lecture 3

## Lecture 3

- Inhalte
  - Restructuring
    - Refactoring
  - Reengineering
  - Dimensionen der Neuentwicklung

## Restructuring

## Restructuring: Definition

- Restructuring is the transformation of
  - one representation form to another
  - at the **same relative abstraction level**,
  - while preserving the subject system's **external behavior** (functionality and semantics).

## Restructuring

- Abstraktionsniveau bleibt erhalten
- Code-to-code transformation
  - Beispiele
    - Ersetzung von goto statements
    - IF zu CASE Transformation
    - Verbesserung der Modularisierung
- Data-to-data transformation
  - Datennormalisierung (z.B. Transformation in 3. Normalform)
- Ziele
  - Verbesserte Struktureigenschaften, Modularisierung
  - Dadurch erhöhte Lesbarkeit, Testbarkeit, Effizienz

## Restructuring: Historie

- 1966 Boehm/Jacobini
  - Jeder synchrone Ablauf eines Algorithmus kann mit 3 Konstrukten a) Sequenz b) Selektion c) Iteration logisch äquivalent zu hergebrachten mit goto programmierten Formen dargestellt werden
- 1968 Dijkstra (*goto* statement considered harmful)
  - Programme ohne *goto* sind übersichtlicher und besser lesbar
- 1972 Ashcroft und Manna
  - Alle Kontrollstrukturen in einem Programm können durch einen Algorithmus in while-Strukturen ersetzt werden

## Restructuring: Historie

- 1975 Erste Restructuring Tools kommen auf den Markt
  - „Structured Engine“ für Fortran
  - Neater/2 für PL/I
- 1980 Belady et al
  - „A graphic representation of structured programs“
  - Bedeutsam für die grafische Unterstützung des Restrukturierungsprozesses
- Modern
  - Refactoring (entspricht Restructuring im OO Paradigma)

# Refactoring

## Refactoring: Motivation and Definition

- “Any fool can write code that a computer can understand. Good programmers write code that humans can understand.”
- Definition “Refactoring”
  - “Improving the design of a program after it has been written”
  - A change made to the internal structure of software to make it **easier to understand** and **cheaper to modify** without changing its observable behaviour
- Refactoring is
  - Perfective maintenance
  - Object-oriented restructuring
  - Functionality preserving

[Fowler99]

# Refactoring

- Refactoring basiert auf Arbeiten von Ward Cunningham, Kent Beck und William Opdyke
- Es existiert eine große Anzahl von vordefinierten Refactoring Maßnahmen
  - z.B. Extract Method, Move Method, Move Field, Remove Parameter, Collapse Hierarchy, Remove Middle Man, ...
    - M. Fowler, "Refactoring – Improving the design of existing code", Addison-Wesley, 1999
    - [www.refactoring.com](http://www.refactoring.com)

# Refactoring: Advantages

- Advantages of Refactoring
  - Improves the Design of Software
    - Without refactoring, the design of software will decay
  - Makes Software Easier to Understand
    - Easier understanding means lower maintenance costs
  - Helps You Find Bugs
    - As you refactor, you better understand code
  - Helps You Program Faster
    - Based on the assumption that good design allows rapid development

[Fowler99]

## When to Refactor

- „The Rule of Three“
  - Refactor when you **add a function**
  - Refactor when you **need to fix a bug**
  - Refactor as you do a **code review**

[Fowler99]

## When to Refactor

- Duplicate Code
  - Extract method, Pull up field
- Long Method
  - Extract method, Introduce parameter object
- Large Class
  - Extract class, Extract subclass
- Switch Statements
  - Replace type code with subclasses/state
- ...

[Fowler99]

## Refactoring: Problems

- Databases
  - Flexibility in refactoring depends on the [level of indirection](#)
  - Changing the database schema forces you to [migrate the data](#)
- Changing Interfaces
  - Many refactorings change the [interface](#) of a class
  - Public interfaces vs. published interfaces
  - Solution: leave the old interface unchanged and publish a new one
- Design changes that are difficult to refactor
  - Boils down to the question: Can everything be solved by refactoring?

[Fowler99]

## Refactoring: Tools

- IDE embedded
  - Borland JBuilder
  - IntelliJ IDEA
  - Eclipse etc.
- Plugins
  - RefactorIt (NetBeans, Forte, JDeveloper, JBuilder)
  - JafaRefactor (jEdit)
- Tools are available for Java, Smalltalk, .NET, Visual Basic, Python, Self, ...
- [[www.refactoring.com](http://www.refactoring.com)]

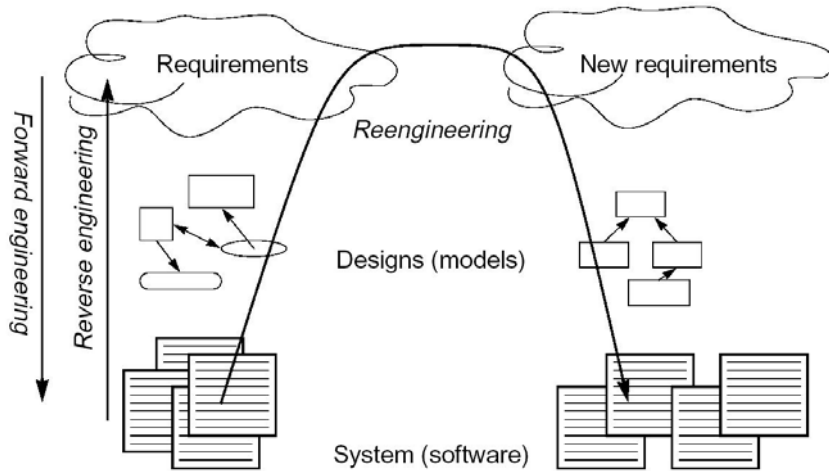


# Re-Engineering

## Re-Engineering: Definition

- Re-Engineering ist
  - die **Untersuchung und Änderung** eines bestehenden Systems
  - mit dem Ziel, das System in einer neuen, geänderten Form zu implementieren
    - **Strukturiert in Reverse Engineering und Forward Engineering Phase**

## Reverse and Reengineering



© J. Weidl-Rektenwald 02-06

119

## Re-Engineering: Probleme

- Komplexe Systeme können meist nur **schrittweise migriert** werden
  - Anbindung über *forward-* und *reverse-gateways*
- Reverse Engineering Ergebnisse sind unter Umständen dürrtig
  - Fehlendes Know How, Werkzeuge
  - Hohe Komplexität der untersuchten Systeme
  - Zeitdruck
  - Im durch Forward Engineering erstellten System fehlt daher oft Funktionalität

© J. Weidl-Rektenwald 02-06

120

## Reengineering: Examples

- Data migration
  - Flat Files Data Repository to Database
- Programming language migration
  - e.g. 3GL to 4GL (Cobol to C++ or Java)
- User interface migration
  - Command line to GUI
- Procedural to object-oriented paradigm
  - „Re-architecting“

## Dimensionen der Neuentwicklung

- Neuentwicklung
  - From scratch / Auf der grünen Wiese
  - Durch Anforderungsanalyse, typisches Forward Engineering
- Re-Engineering
  - Reverse Engineering von
    - Design
    - Architektur
    - Anforderungen
  - Anschließendes Forward Engineering
- Transformation
  - Intra-paradigmatisch vs. inter-paradigmatisch

# Organisation der Wartung

## „Organisation“

- *Organisation* als *Tätigkeit* („organisieren“) heißt, einem zielorientierten, sozio-technischen System eine *dauerhaft wirksame Struktur* zu geben. Diese Struktur entsteht durch formalisierte (verbindliche) *generelle Regelungen*, in dem die *Beziehungen* von *Aufgabenträgern, Informationen und Sachmitteln* bei der Aufgabenerfüllung festgelegt sind.

## Organisation und Management

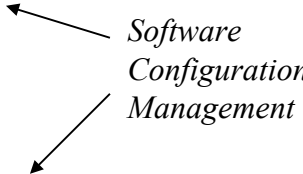
- *Organisation* als *Ergebnis* des Organisierens ist die **dauerhaft wirksame Struktur** von zielorientierten sozio-technischen Systemen.
- Gegenstand der Managementlehre ist die Gestaltung von **Organisationen**

## Aktivitäten des Wartungsfalles (vgl. Lecture 1)

- Analyse bzw. Planung der Änderung
  - Program Comprehension
  - Change Impact Analysis
- Implementierung der Änderung
  - Restructuring
  - Change Propagation
- Verifikation und Validierung
- Re-Dokumentation

M  
a  
n  
a  
g  
e  
m  
e  
n  
t

# Management des Wartungsfalles

- Fehlermeldung bzw. Änderungsantrag
    - Life Cycle Management (Defect and Change Tracking)
    - Evaluierung/Reporting
  - Analyse bzw. Planung der Änderung
    - Program Comprehension
    - Change Impact Analysis
  - Implementierung der Änderung
    - Restructuring
    - Change Propagation
    - Verwalten der Artefakte (Software Artifact Management)
  - Verifikation und Validierung
  - Re-Dokumentation
  - Produktivstellung der Änderung
- Software Configuration Management*
- 

# Life Cycle Modelle der Software Wartung

# Life Cycle Modelle

- Life Cycle Modelle
  - beschreiben den zeitlichen Ablauf von Teil-Prozessen in einem Gesamtprozess
  - bieten einen Top-Level View auf einen Gesamtprozess

# Life Cycle Modelle der Software Entwicklung

- Wasserfall Modell [W.W. Royce 1970]
- Spiralmodell [Boehm]
- Rapid prototyping
- OO
  - Fountain Model
  - Inkrementell-iterative Entwicklung (vgl. Rational Unified Process - RUP)
  - Extreme Programming (Kent Beck et al.)

## Life Cycle Modelle der Software Wartung

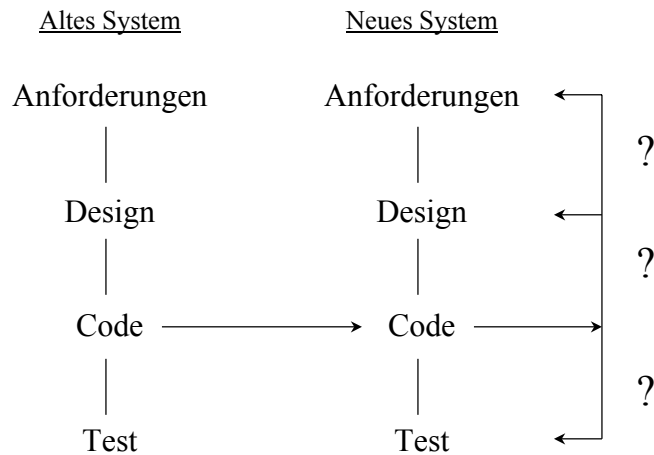
- Quick-fix Model [Basili90]
  - Änderung erfolgt **sofort** auf der Code Ebene
  - Änderung wird in Dokumentation typischerweise **nicht** nachgezogen
- Iterative Enhancement Model [Basili90]
  - Beginnt eine Änderung bei den **Anforderungen** und zieht Änderungen definiert bis in die **Testphase** nach

## Life Cycle Modelle der Software Wartung

- Full-reuse Modell [Basili90]
  - Reengineering Ansatz
  - Klassisches Forward-Engineering beginnend mit den Requirements und Wiederverwendung von möglichst großen Teilen des Altsystems
  - Voraussetzung
    - **Altsystem ist in wieder verwendbare Teile strukturiert**



## Quick Fix Model



© J. Weidl-Rektenwald 02-06

133

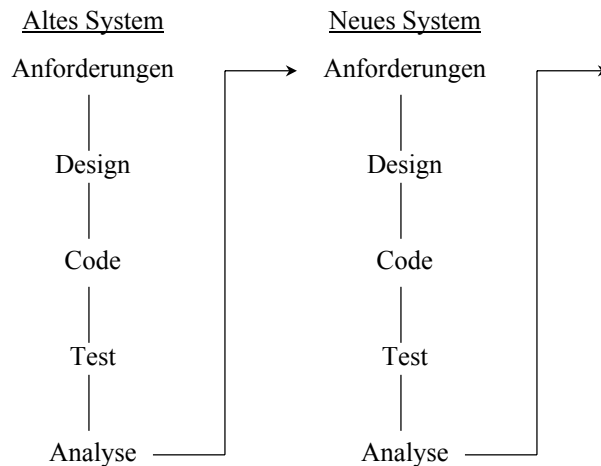
## Quick Fix Model: Folgen

- Führt zu **unübersichtlichen** Systemen
- Mit jeder Änderung werden weitere Änderungen **erschwert**
  - Dokumentation zu den vorangegangenen Änderungen (warum?, warum hier?, warum nicht anders?) ist nicht vorhanden

© J. Weidl-Rektenwald 02-06

134

## Iterative Enhancement Model



© J. Weidl-Rektenwald 02-06

135

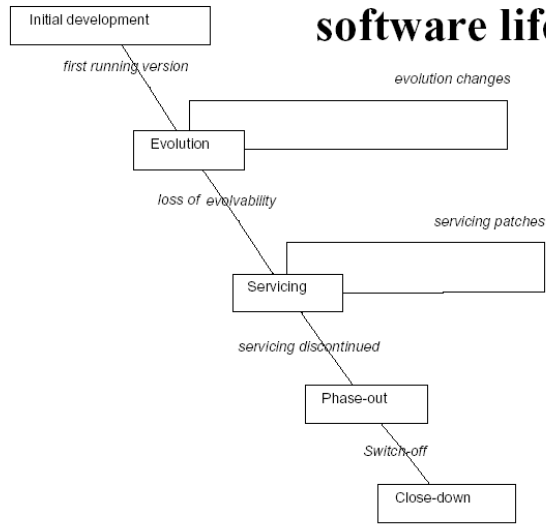
## Staged Life Cycle Model

- Nach [Bennett/Rajlich]:
- „If changes can be anticipated at design time
  - They can be built in by parameterisations etc.“
    - (i.e. they can be planned for)
- „However, 40 years of hard experience confirms:
  - Many changes cannot even be **conceived** by the original designers
  - **Inability** to change software quickly and reliably means that business opportunities are lost
  - Our solution: base the software life cycle on the fact that many changes cannot be predicted“

© J. Weidl-Rektenwald 02-06

136

# Staged model of software lifecycle



# Versioned staged model

