Software Wartung und Evolution

Prof. Dr. Harald Gall

Dr. Emanuel Giger

Universität Zürich, Institut für Informatik

http://www.ifi.uzh.ch/seal/teaching/courses/SWEvo13.html

http://tinyurl.com/seal-evolution

Software Wartung und Evolution

Teil 2: Wartungsaspekte, Reverse Engineering

Harald Gall

Institut für Informatik

Universität Zürich

seal.ifi.uzh.ch

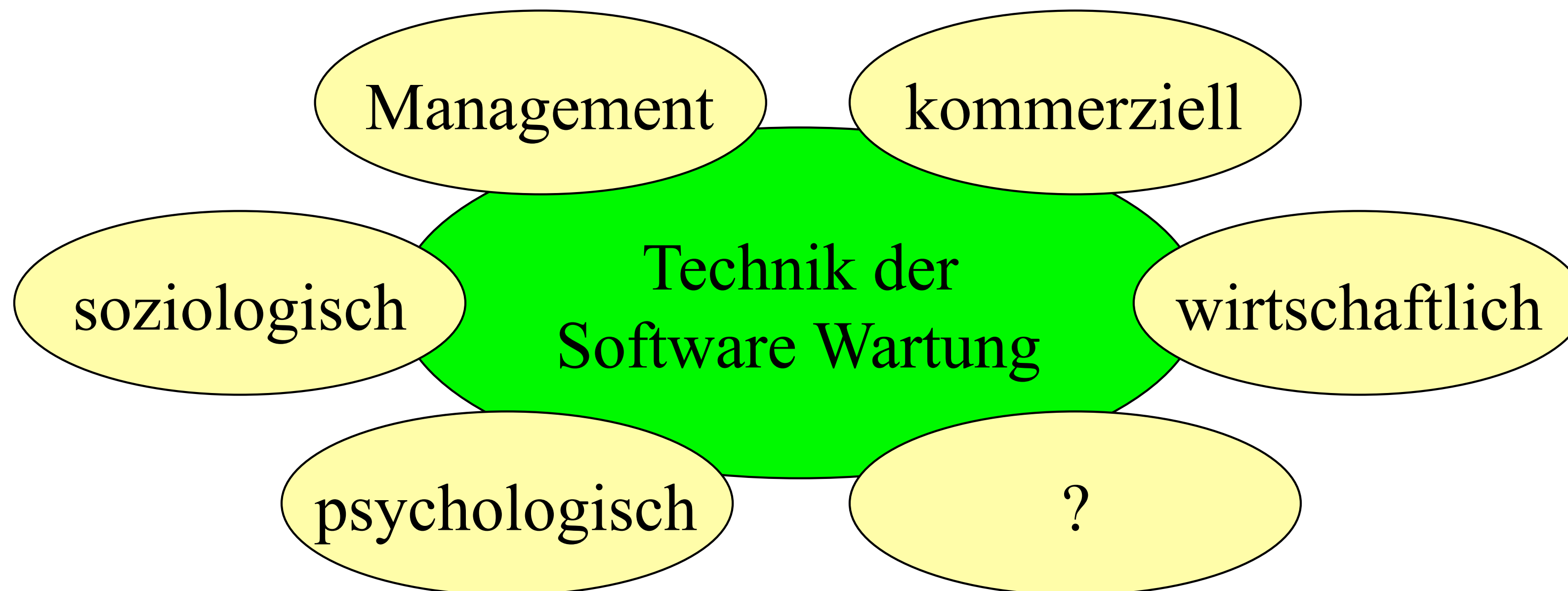
Teil 2

- Inhalte
  - Aspekte der Software Wartung
  - Aktivitäten der Software Wartung
  - Software Wartungskrise
  - Legacy Systeme
  - Reverse Engineering
    - Redocumentation
    - Design Recovery

Aspekte der Software Wartung

... und das war´s?

Aspekte der Software Wartung



- Berücksichtigung aller Aspekte führt zur *holistischen* Betrachtung von Software Wartung

Aspekte der Software Wartung

- Technical: Understanding existing code
- Managerial: Reactive work context
- Economic: Justifying remedial work
- Commercial: How to estimate the effort of a change request?
- Sociological
  - Apprentice‘s work
  - Maintaining morale
- Psychological
  - Hesitating to touch a huge work of art
  - Fear to damage a working system
  - „Kognitive Dissonanz“

Kognitive Dissonanz: Beispiel

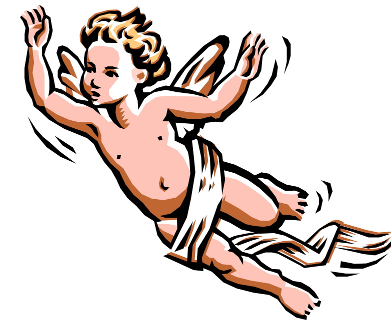
- Sachverhalt: Die Software funktioniert nicht so, wie sie sollte
- Wahrnehmung Programmierer: „Ich bin mit meiner Arbeit zufrieden. Ich finde mich gut.“ (ignoriert)
- Wahrnehmung QA: „Das funktioniert nicht.“
- Feedback QA: „In ihrem Programm ist ein Bug“
- -> kognitive Dissonanz: „Ich bin gut – Du hast Mist gebaut“
- Auflösung:
  - „Der Bug liegt sicher nicht bei mir! Fragen sie mal Kollegen X.“
  - „Es steht aber genau so in den Anforderungen! Lesen sie die mal!“
  - „Das ist kein Bug, die User haben es mir so erklärt.“
  - „Sie wissen ja nicht einmal, was ein Bug ist!“

- Kognitive Dissonanz führt also zu einer Wirklichkeitskonstruktion, die versucht, die Dissonanz aufzulösen.

Egoless Programming

- Über-Identifikation mit den Artefakten der Arbeit ist ein unerwünschter Nebeneffekt
- -> „Egoless Programming“
- Modern: Extreme Programming mit dem Leitfaden „Embrace Change“ [Kent Beck]
  - Ist psychologisches Problem, nicht rein technisches!

Kommerzieller Aspekt



Budget, Time-to-market

Software Qualität

University of Zurich[UZH]

s.e.a.l. software evolution & architecture lab

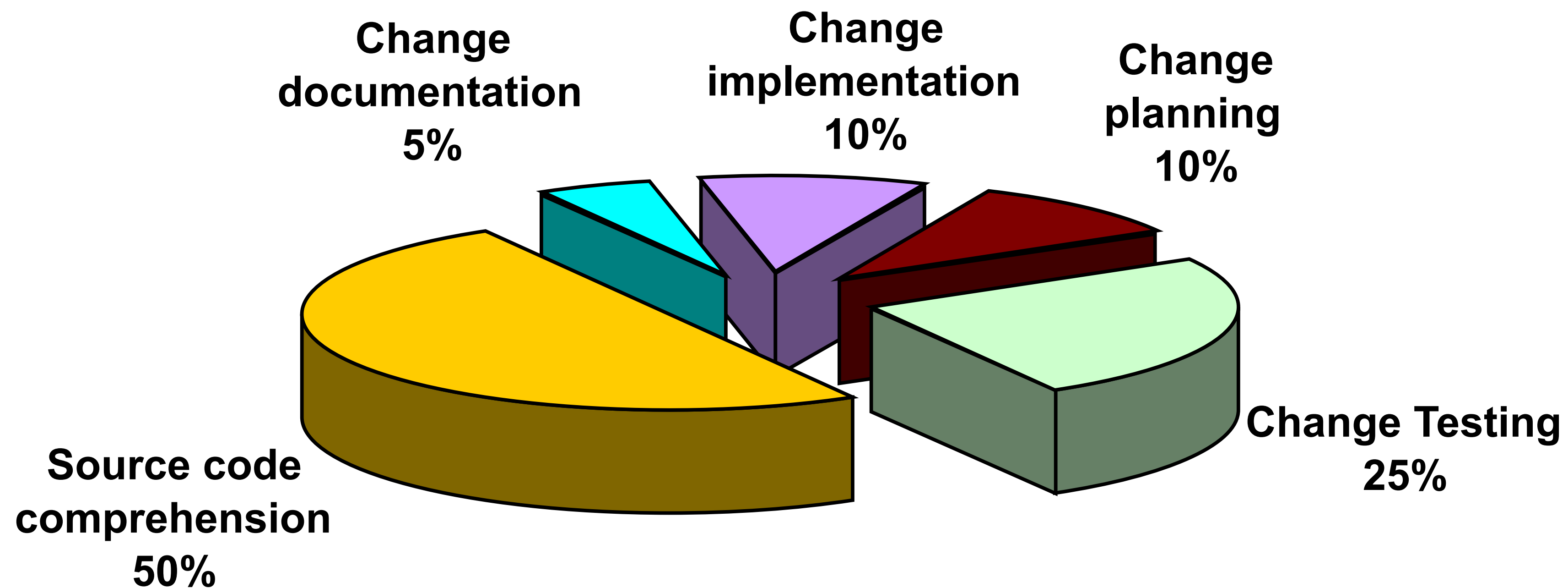
Aktivitäten der Software Wartung

Aktivitäten der Software Wartung

- Analyse bzw. Planung der Änderung
  - Verstehen des Systems, der „Architektur“
  - Source Code Verstehen, Bilden von Hypothesen
  - Verifizieren von Hypothesen
  - Change Impact Analysis
- Implementierung der Änderung
  - Restructuring
  - Change Propagation
- Verifikation und Validierung
- Re-Dokumentation

} Management

Activities in Software Maintenance



Activities in Software Maintenance

Source: Principles of Software Engineering and Design, Zelkovits, Shaw, Gannon 1979

Legacy Systeme

Eigenschaften von Legacy Systemen

- Größe: >1 Mio. LOC
- Alter: > 10 Jahre
  - Verwendung veralteter Programmiersprache wie COBOL, FORTRAN, PL/I, ...
  - Verwendung veralteter Datenspeicherung, z.B.
    - Flat Files
    - Hierarchische Datenbanken
  - Dokumentation veraltet
- Strategische Bedeutung
  - Bilden kritische Geschäftsprozesse ab – Unternehmen ohne System nicht vorstellbar
  - 24x7 Verfügbarkeit

[Kroha97]

Eigenschaften von Legacy Systemen

- Kosten
  - Wartungskosten bei Legacy Systemen typischerweise über 95% der Gesamtkosten
- Systemumgebung
  - Limitierte Hardwareressourcen
  - Begrenzte Anbindungsmöglichkeiten an Kommunikationsprotokolle (Middleware, z.B. CORBA)
- Komplexität
  - Neue Anforderungen können im System nicht mehr verwirklicht werden
  - Unzufriedenstellende Performanz (z.B. Transaktionsrate)

[Kroha97]

Probleme bei Ablöse von Legacy Systemen

- Das neue System muss funktional äquivalent zum alten System sein
- Das neue System muss zusätzlich alle aktuellen Anforderungen implementieren
- Die Daten der Legacy Applikation müssen übernommen werden
- Die neue Applikation soll state-of-the-art Technologien verwenden (Datenbank, 3-Tier, Objektorientierung, Middleware, ...)
- Die Ausfallszeit durch die Ablöse soll minimal sein

[Kroha97]

Gründe für das Scheitern von Legacy Neuimplementierungen

- Um die Mittel bewilligt zu bekommen, müssen umfangreiche Erweiterungen direkt mit der Migration versprochen werden
- Neuentwicklung dauert lange – in der Zwischenzeit ändern sich die Anforderungen
- Es existieren versteckte Abhängigkeiten von vielen Programmen zur Legacy Applikation
- Politische Einflüsse verhindern eine Fertigstellung
- Management von Software Großprojekten ist im Allgemeinen schwierig

[Kroha97]

Migration von Legacy Systemen

- Meist inkrementell auf per-Modul Basis
  - Reverse Engineering
  - Restrukturierung und Reuse
  - Entwicklung neuer Komponenten
  - Integration
  - Datenmigration
  - Inbetriebnahme (Installation, Schulung, ...)

[Kroha97]

Beispiel:
Technologien Erste Bank

- 1990 waren bei der Ersten Bank Systeme in folgenden Technologien im Einsatz
  - Assembler
  - PL/I
  - Cobol
  - Fortran-TRX
  - TRX-GEN 1/2
  - „Entscheidungstabellen, DELTA, TIMESHARING, DMS1100“
    - Aus: Spezielle Aspekte der Informationsverarbeitung in der Wirtschaft, W. Konvicka 1995

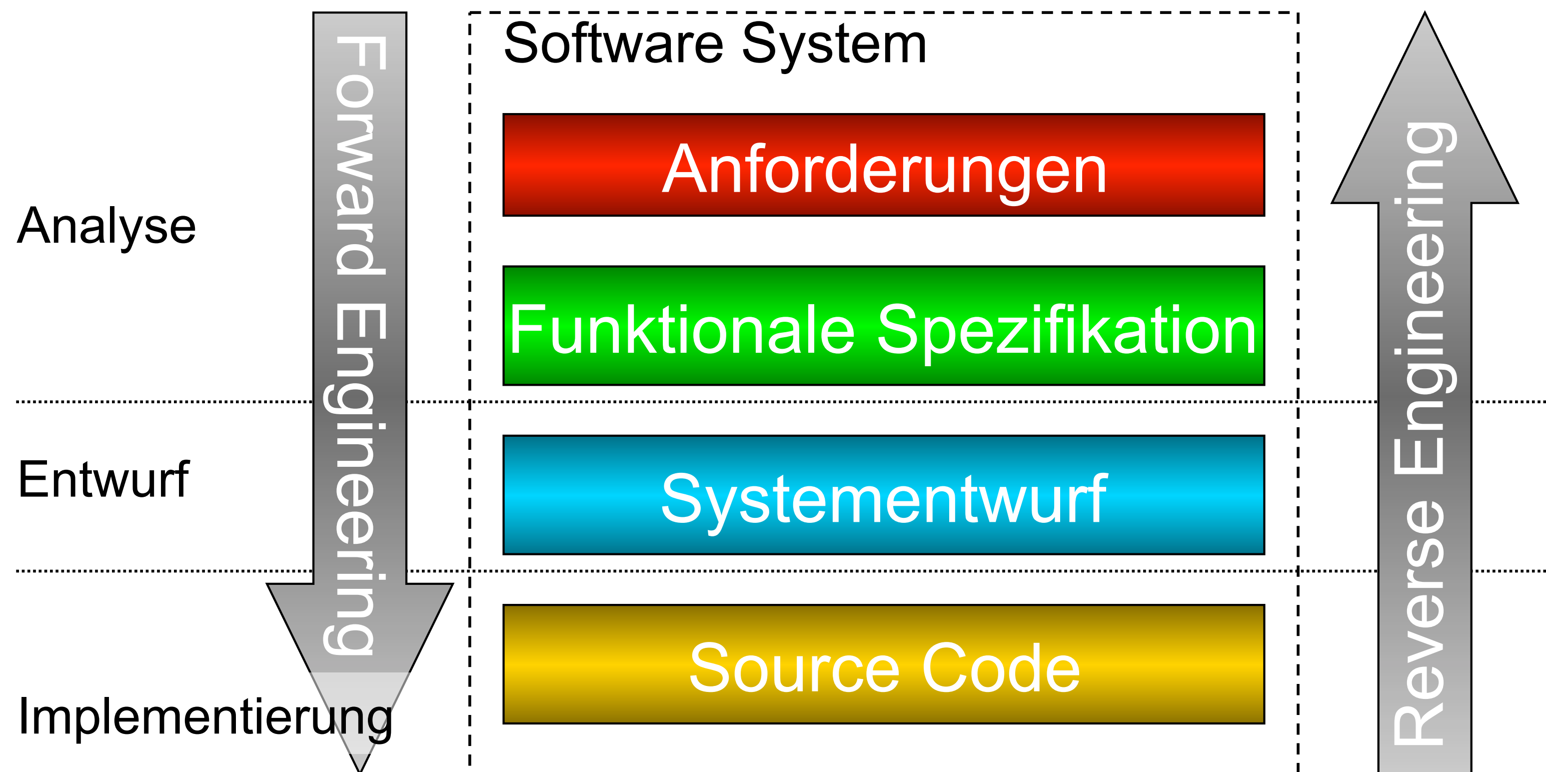
Reverse Engineering

Reverse Engineering: Definition

- Unter Reverse Engineering versteht man den Prozess der Analyse eines bestehenden Systems, mit dem Zweck
  - der Identifikation von Systemkomponenten und deren Beziehungen untereinander, sowie
  - der Erzeugung von Darstellungen des untersuchten Systems auf unterschiedlichen, höheren Abstraktionsstufen.

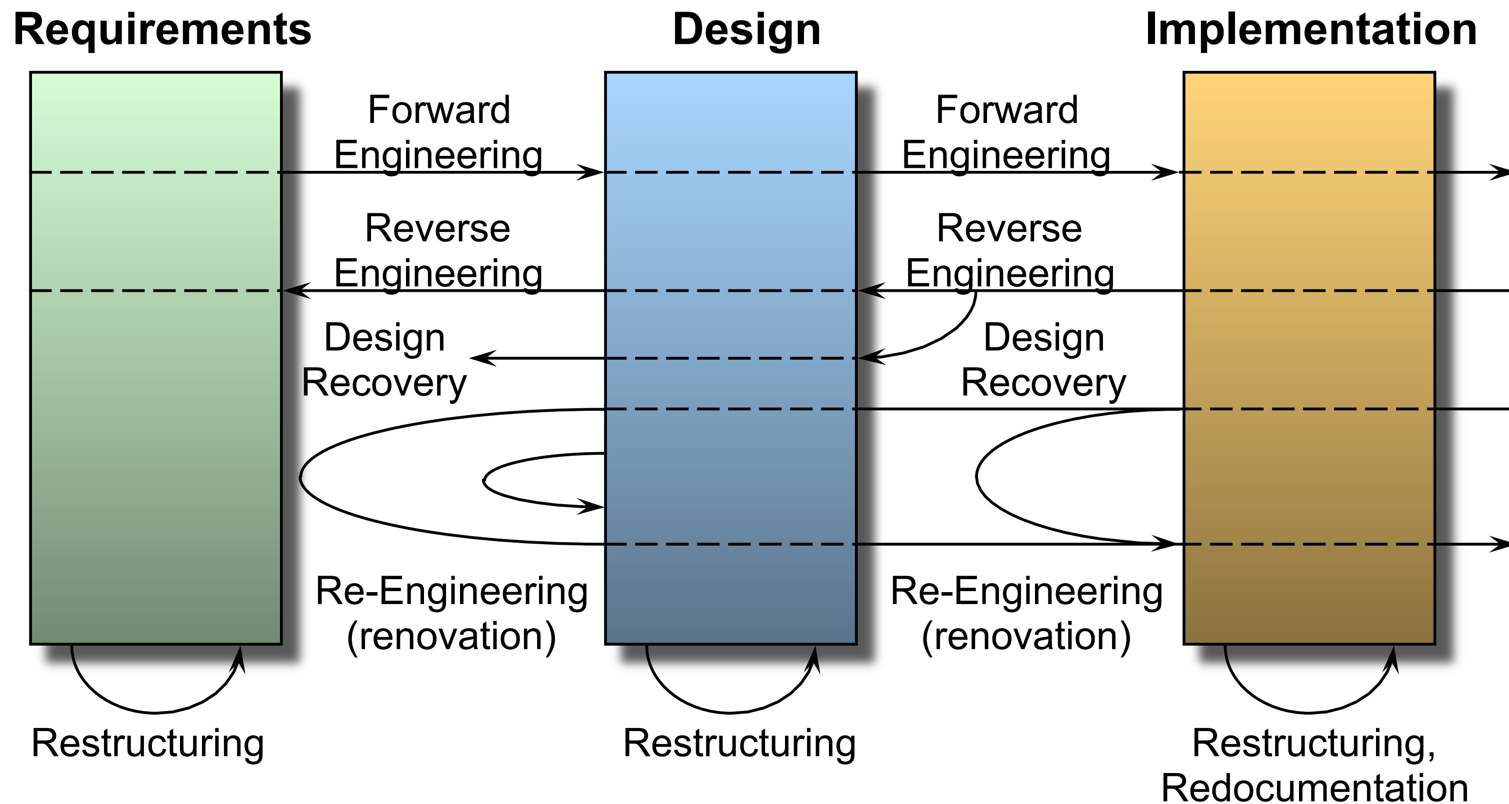
[Chikofsky/Cross]

Forward- und Reverse Engineering



[Klösch/Gall95]

Reverse Engineering Terminologie



[Chikofsky/Cross90]

Motivation

- Ca. 75% of software development time and expense goes toward maintenance
- Of this, ca. 50% goes to understanding the software and the bug/enhancement
- Consequently, we want to devise tools and techniques to support improved understanding of software

Reverse Engineering verlangt Wissen

- Programmiersprache
  - Syntax
  - Semantik
- Programmierung (Algorithmen, Datenstrukturen, Patterns, ...)
- Application Domain („Domänenwissen“)

Rerverse Engineering: Subprozesse

- Redocumentation
- Design Recovery

Redocumentation

- Erzeugung oder Überarbeitung von semantisch äquivalenten Repräsentationen des Systems innerhalb desselben Abstraktionsniveaus
  - Datenfluss- und Kontrollflussdiagrammen, Cross Reference Listings, etc.
  - Automatisch generiert ohne zusätzliche Informationen

Design Recovery

- Ist ein Prozess, in dem zusätzliche Information verwendet wird, um Abstraktionen des Systems zu generieren u. zw. auf höheren Abstraktionsstufen
  - Wissen über Anwendungsdomäne, informale Beziehungen
  - Wissen von Applikationsexperten über Architektur, Modulstruktur, etc.

Design Recovery: Repräsentationen

- Repräsentationen durch Design Recovery
  - Structure Charts
  - Nested Trees
  - Datenflussdiagramme
  - Kontrollflussdiagramme
  - Entity Relationship, ...
  - Informale Beschreibungen der Software
    - Text
    - Diagramme

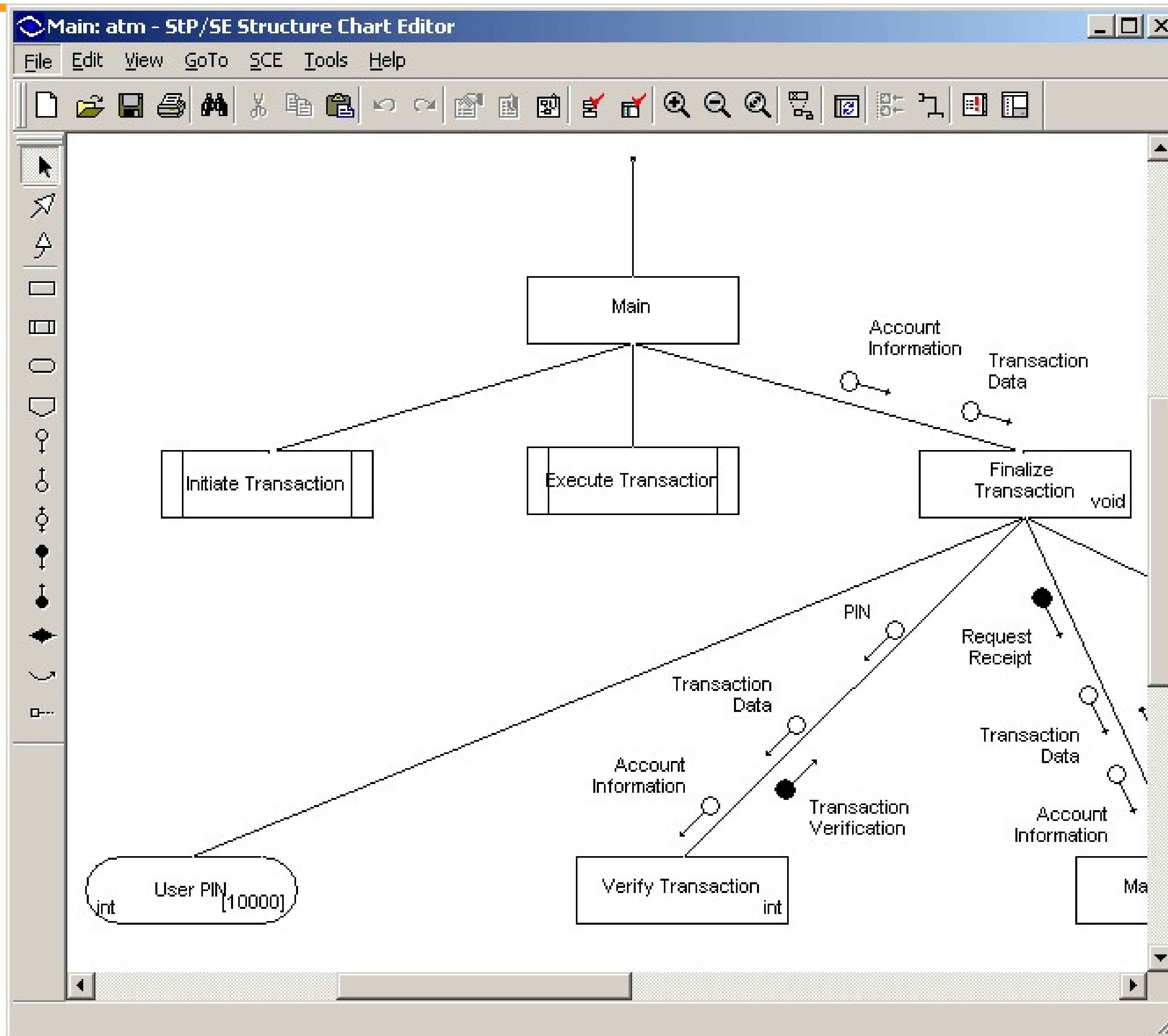
Angereichert durch informale Information

Arten von Design Recovery

- Modellbasiert
  - Desire von Biggerstaff [Biggerstaff89]
- Wissensbasiert
  - Basierend auf zentraler Wissensbasis (Repository)
  - Extrahierung von Programmclichés
    - z.B. Sortieralgorithmen, Listen, Bäume
- Formale Methoden
  - Transformation von v.a. COBOL in Z oder Z++
  - siehe ESPRIT Projekt „REDO“
- Objektorientiert
  - Objekt Identifizierung

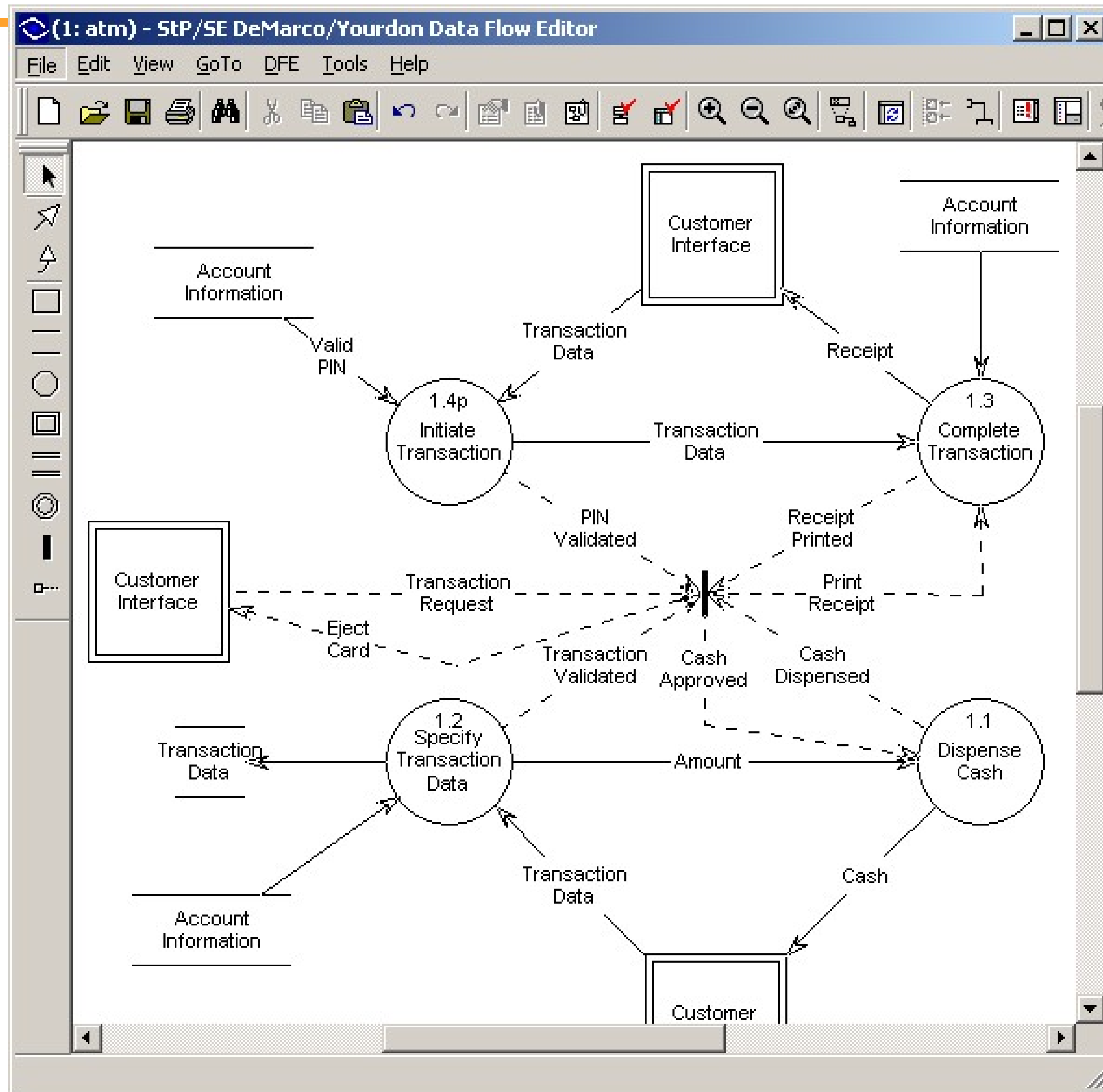
Structure Charts

- Aus dem Strukturierten Design [Coad/Yourdon79]
- Stellen die Modulstruktur nach der funktionalen Dekomposition [Parnas72] in einer hierarchischen Form dar
- Beinhalten Information, welche Daten zwischen den Modulen ausgetauscht werden
- Black box Ansicht von Modulen, Verhalten wird durch Input/Output beschrieben
- In UML: Klassen + Event Trace Diagramme


Main: atm - StP/SE Structure Chart Editor
File Edit View GoTo SCE Tools Help
Main
Account Information
Transaction Data
Initiate Transaction
Execute Transaction
Finalize Transaction
void
PIN
Request Receipt
Transaction Data
Account Information
Transaction Verification
Transaction Data
Account Information
User PIN
int
[10000]
Verify Transaction
int
Ma

Datenflussdiagramme

- Stellen Datenfluss und Datentransformation dar
    - Datentransformation: Ersetzen der formalen Parameter von Prozeduren durch die aktuellen Parameter
- Verschiedene Ansätze zur „bottom-up“ (i.e. reverse) Generierung finden sich in der Literatur
    - z.B. Benedusi, Cimitile und De Carlini [BCD89]


(1: atm) - StP/SE DeMarco/Yourdon Data Flow Editor
File Edit View GoTo DFE Tools Help
Customer Interface
Account Information
Account Information
Valid PIN
Transaction Data
Receipt
1.4p Initiate Transaction
Transaction Data
1.3 Complete Transaction
PIN Validated
Receipt Printed
Customer Interface
Transaction Request
Print Receipt
Eject Card
Transaction Validated
Cash Approved
Cash Dispensed
1.2 Specify Transaction Data
1.1 Dispense Cash
Transaction Data
Amount
Transaction Data
Cash
Account Information
Customer

Kontrollflussdiagramme

- Visualisierung des Kontrollflusses
  - Zwischen Prozeduren
    - Call tree
  - Innerhalb einer Prozedur
    - „Logische Wege“ durch eine Prozedur werden visualisiert
    - Visualisieren die zyklomatische Komplexität (McCabe Metrik)

Design Decisions

- During design and implementation decisions are made according to specific design rationales
  - Formal representation: Design Decision Tree (DDT)
- Tools for making design decisions persistent during the development process are only in experimental stage
  - Therefore, most of the design decisions almost always have to be extracted when examining existing code
- Reverse engineering design decisions deal with automated decision extraction and injection, knowledge repositories, knowledge management

Ziele von Reverse Engineering

- Beherrschung der System Komplexität
- Erzeugen von fehlender oder alternativer Dokumentation
- Wiedergewinnung verlorener Information
- Erkennung von Seiteneffekten und Anomalien
- Migration auf eine andere Hardware/Software Plattform bzw. Integration in eine CASE Umgebung
- Erleichterung der Software-Wiederverwendung

[Klösch/Gall95]

Reverse Engineering Kandidaten

- Schlecht strukturierter Source Code
- Umfassende korrektive Wartung
- Veraltete Dokumentation
- Design Infos fehlen oder sind unvollständig
- Module sind unüberschaubar komplex
- Migration auf eine neue Plattform
- System soll durch ein neues abgelöst werden

[Klösch/Gall95]

Reverse Engineering: Vorteile

- Kosteneinsparung in der Software Wartung
- Ermöglichen weiterer Software Evolution
- Qualitätsverbesserung
- Wiederverwendung von Software Komponenten
- Vorteile im Wettbewerb
- Investitionssicherung

[Klösch/Gall95]

Reverse Engineering: Probleme

- Umfangreiches Source Code Volumen
- Mangelndes Wissen über das Programm
- Inkonsistente Entwicklungsstandards
- Nicht aktuelle oder fehlende Dokumentation
- Hohe Redundanz

[Klösch/Gall95]

Reverse Engineering Tools

- Imagix4D (Imagix Corp.)
    - www.imagix.com
- SonarGraph
    - http://www.hello2morrow.com/products/sonargraph
- Understand (Scitools)
    - http://www.scitools.com
- Source Navigator
    - www.redhat.com
- Rational Rose (IBM)
    - www.rational.com/rose
- uvam.

Die Tool Falle

- *A fool with a tool is still a fool*
  - English proverb
- Give software tools to good engineers. You want bad engineers produce less, not more, poor-quality software.
  - [Davis95]: Principles of Software Development