XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 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

Organisatorisches 1/2

- LVA Info
  - VU 2.0, 184.169
  - Wahlfach: 033 522, 066 922, 066 933, 066 937
- Vorlesungsteil
  - 7 VO Termine
  - 15.03., 22.03, 29.03., 19.04., 26.04., 10.05., 24.05.
  - Jeweils Donnerstag, 16:15 - 17:45, pünktlich!
  - EI 4 Reithoffer Hörsaal

Organisatorisches 2/2

- Übungsteil
  - 1 Übungsbeispiel
  - Gruppenarbeit
  - Abschlusspräsentation in der Xion
- Prüfung
  - Donnerstag, 14. Juni 2007, 16:15 – 17:15, EI 4
- VU Web Page
  - http://www.infosys.tuwien.ac.at/Teaching/Courses/SWE/swe.html

Maintenance

Wartung der Anwendungen verbraucht 83% des IT Budget

"Too much money for maintenance. With just 27% of 2004 spending planned for new development, maintenance costs are choking IT productivity."

Gartner, Executive Summary, September 2004

60% des IT-Budget geht auf für Legacy Wartung

"Between 60 and 80 percent of an average company's IT budget is spent on maintaining existing mainframe systems and applications."

Gartner Group, Analyst Report, March 2002

Legacy

Legacy gibt es (fast) immer und überall

- **70% der weltweiten Geschäftsdaten werden mit COBOL prozessiert**
- **Es gibt über 200 Milliarden COBOL "Lines of Code"**
- **30 Milliarden COBOL Transaktionen werden täglich prozessiert – mehr als alle Zugriffe auf Webseiten**

Legacy Modernisierung ist eine strategische Notwendigkeit

"It is clear that new technology will no more replace legacy in total than legacy technology alone will satisfy the needs of today's Web-savvy users. The truth is that modernized legacy applications play a crucial role."

Giga, Analyst Report, September 2002

Inhalt der Vorlesung: Überblick

- Was versteht man unter Software Wartung / Software Evolution / Wartbarkeit?
- Was sind die speziellen Probleme?
- Welche adäquaten Technologien, Prozesse und Tools gibt es, um diesen zu begegnen?
- Was sind die Best Practices der Software Wartung?
- Wie managt man Software Wartung?

Inhalt der Vorlesung: Themen

- *(1) Software Wartung*: Motivation, Definition, Arten, Probleme
- *(2) Software Wartung*: Aspekte, Aktivitäten, Wartungskrise, Legacy Systeme, *Re*verse Engineering
- (3) *Re*structuring, *Re*-Engineering, *Organisation der Wartung*: Software Life Cycle Modelle
- (4) *Tool Demos*, *Organisation der Wartung*: Defect Tracking, Software Configuration Management, Produktivstellung
- (5) *Software Evolution: E*-type Programs, Laws of Software Evolution Explained, Change Patterns
- *(6) Spezielle Kapitel der Software Wartung*: Program Comprehension, Change Impact Analysis, Qualitätsmerkmal „Wartbarkeit“
- (7) Best Practices: Design for Change, MDA und Wartung, Software Wartung im unternehmerischen Kontext: Rollen, Gewährleistung, Software-Wartungsverträge

Lecture 1

- Inhalte
  - Motivation: Software Wartung und Evolution
  - Abgrenzung der Begriffe Software Wartung und Software Entwicklung
  - Definition „Software Wartung“
  - Arten der Software Wartung
  - Probleme der Software Wartung

Software Engineering vs. Software Maintenance

- Suche nach dem Begriff „Software Engineering“ bei amazon.de, Kategorie “Englische Bücher”
  - Treffer: 8405 (2006: 1510; *2005*: 1405; *2004*: 1078)
    - Top Treffer: “Agile Web Development with Rails“ von D. Thomas et al. (2007)
    - Treffer 2: “Design Patterns – Elements of Reusable Code” (GoF; 1995)
    - Treffer 3: “Refactoring” (M. Fowler; 1999)
- Suche nach „Software Maintenance“
  - Treffer: 374 (2006: 90; *2005*: 92; *2004*: 92)
    - Top Treffer 2007: “Macs for Dummies” (Baig; 2006)
    - Top Treffer 2006: “Troubleshooting your PC” (1994)
- Search for „Software Evolution“
  - Treffer: 81 (2006: 23; *2005*: 20; *2004*: 15)
    - Top Treffer 2007: „The Old New Thing. Practical Development Throughout the Evolution of Windows” (Chen et al.; 2007)
    - Top Treffer 2006: „Software Engineering: Evolution And Emerging Technologies: 130“ von K. Zielinski (26. Dez. 2005!)
    - Treffer 7: “Successful Evolution of Software Systems” (Yang et al.; 2003)

Annäherung an den Begriff „Software Wartung“

- Software Wartung hat mit dem geläufigen Begriff der technischen Wartung wenig gemein
  - Software hat keine Verschleißteile
  - Software zeigt keine Abnutzungserscheinungen („wear-out“)

Annäherung an den Begriff „Software Wartung“

- Software gilt im Gegensatz zu physischen technischen Artefakten als „leicht“ änderbar

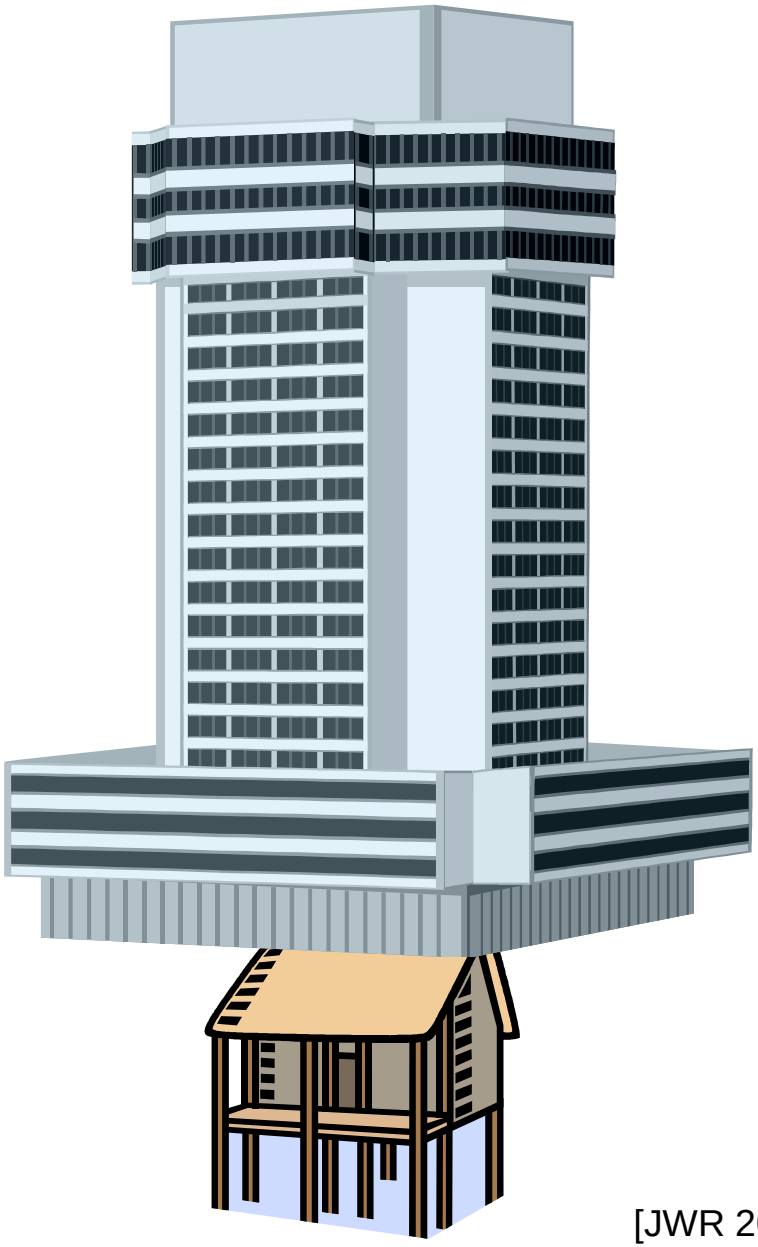
Software ist leicht änderbar?

```
/**
 * Wrong name: should be getRechnungToAuftragIDAndKeyAccountID
 */
 public static Rechnung getRechnungToAuftragIDAndPersonID (Integer iAID,
    Integer iKAID) throws ExceptionPersist {

   Vector v;
   Rechnung r = new Rechnung();
   r.setAuftragID(iAID);
   r.setKundenID(iKAID);
   v = r.search();

   [...]
```

Software ist leicht änderbar?

```
for(int j=0;j<vtz.size();j++) {

    tzkbez = ((DtZeile)vtz.elementAt(j)).getDtKurzBezeichnung();
    iZeichPos = tzkbez.indexOf(':');

    if (tzkbez.substring(0,iZeichPos).equals(tzkbez.substring(iZeichPos+1)))
      ivSortDt = 1;
    else if (tzkbez.substring(0,iZeichPos).compareTo(tzkbez.substring(iZeichPos+1))>0)
      ivSortDt = 0;
    else
      ivSortDt = 2;

    if (ivSortDt == 0) iZeichPos = 0;
    else iZeichPos++;

    int i;
    for(i=0;i<vSortDt[ivSortDt].size();i++) {
      if
       (((DtZeile)vSortDt[ivSortDt].elementAt(i)).getDtKurzBezeichnung().substring(iZeichPos).compareTo(tzkbez.substrin
       g(iZeichPos))>0)
        break;
    }

    if (i<vSortDt[ivSortDt].size())
      vSortDt[ivSortDt].insertElementAt(vtz.elementAt(j),i);
    else
      vSortDt[ivSortDt].add(vtz.elementAt(j));
}
```

Software ist leicht änderbar?



[JWR 2002]

Annäherung an den Begriff „Software Wartung“

- „Programs, like people, get old“
- „Software aging will occur in all successful products“
  - David Lorge Parnas in „Software Aging“ [Parnas 1994]
- Es ist einsichtig, dass, was altert, irgendwie up-to-date gehalten werden muss.
- Die Frage ist: Wie altert Software und warum?

Two Causes of Software Aging

- The first is caused by the failure of the product‘s owners to modify it to meet changing needs
  - „Lack of movement“
- The second is the result of the changes that are made
  - „Ignorant surgery“

[D. L. Parnas, 1994]

Symptoms and Costs of Software Aging

- Inability to keep up
  - "As software ages, it grows bigger"
    - More code to change
    - More difficult to find routines that must be changed
- Reduced performance
  - More machine resources are needed
  - Poor design causes performance bottlenecks
- Decreasing reliability
  - „As the software is maintained, errors are introduced"

[D. L. Parnas, 1994]

Preventive Medicine

- Design for success (aka „Design for change“)
  - Information hiding, abstraction, separation of concerns (SOC), data hiding, object orientation, …
- Documentation
  - Problem: Most documentation is ignored because not being accurate
- Second opinions – Reviews
  - Reviews often are neglected because of time pressure

[D. L. Parnas, 1994]

„Software Geriatrics“

- Stopping deterioration
  - Requires techniques and resources!
- Retroactive documentation
  - Lack of formal basis and influence of short-term interests
- Retroactive incremental modularization
- Amputation
- Restructuring

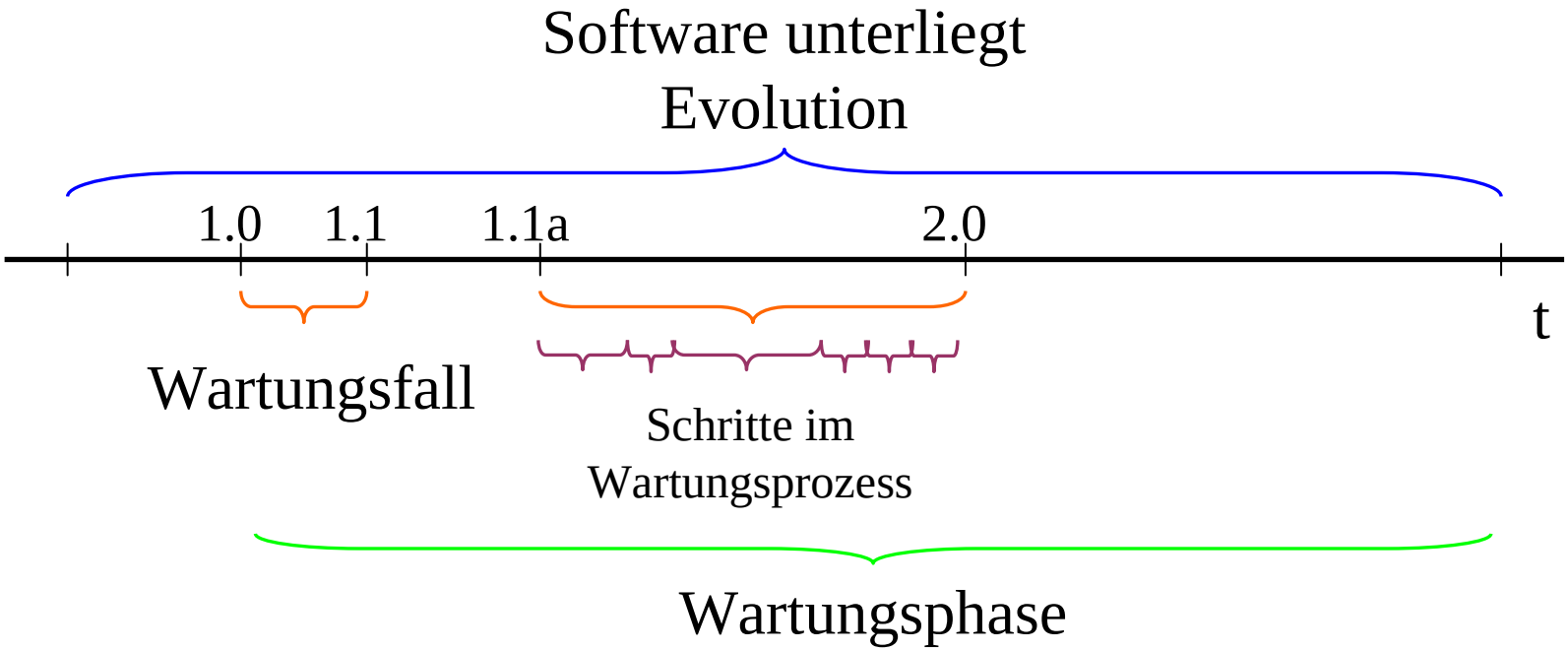
[D. L. Parnas, 1994]

Software Wartung vs. Software Evolution

- Software Wartung
  - bezeichnet als *Tätigkeit* eine Änderung an einem Softwaresystem nach dessen Auslieferung (die Durchführung einer Änderung bezeichnet man als „Wartungsfall“)
  - bezeichnet als *Prozess* die Schritte, die in einem Wartungsfall sequentiell durchzuführen sind
  - bezeichnet als *Phase* den Abschnitt des Lebenszyklus eines Softwaresystems von dessen Auslieferung bis zur Stilllegung
- Software Evolution
  - bezeichnet den *Prozess* der Veränderung eines Softwaresystems von der Erstellung bis zur Stilllegung
  - umfasst: Entwicklung, Wartung, Migration, Stilllegung

[JWR 2002]

Software Wartung vs. Software Evolution



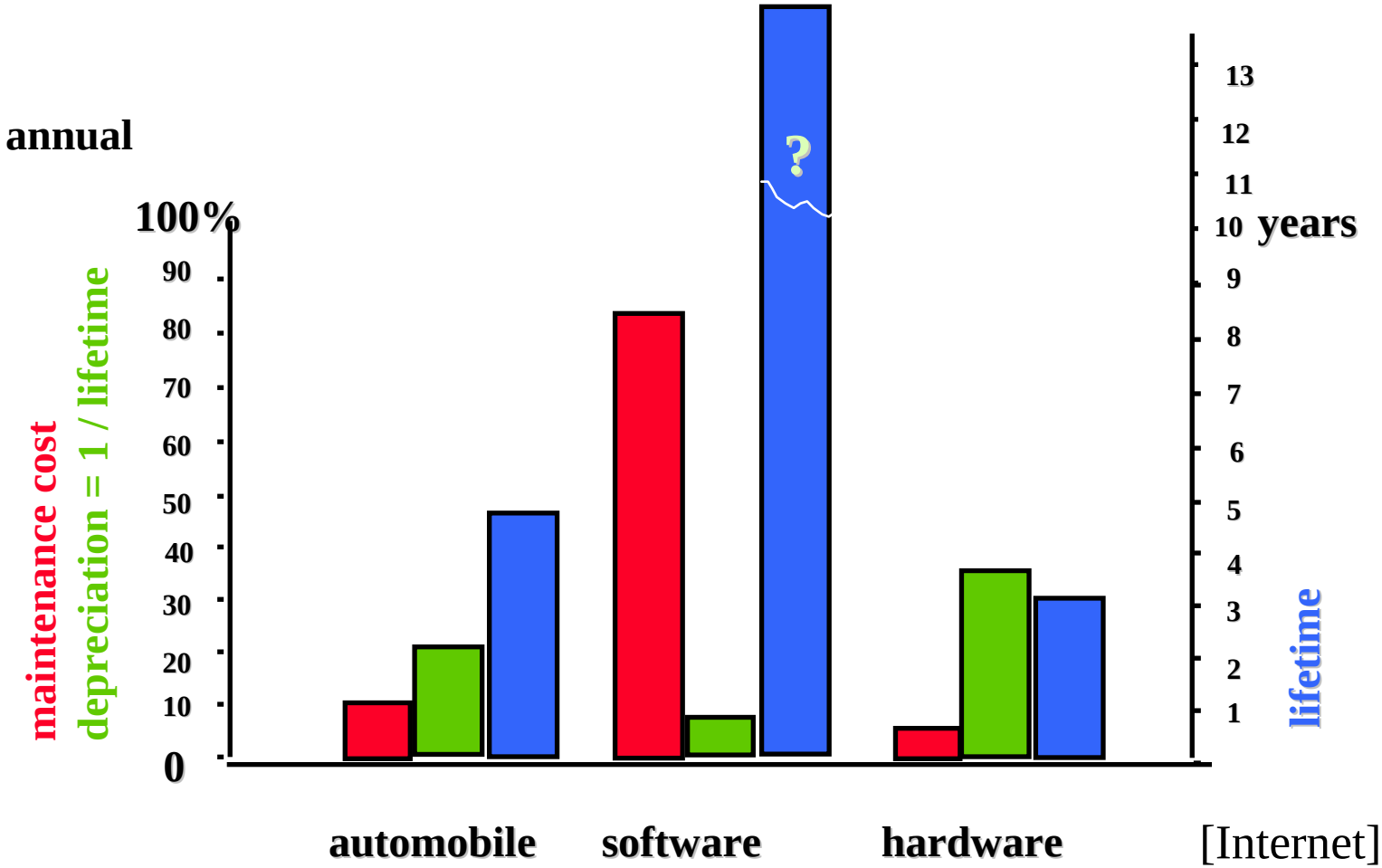
Warum Evolution?

- Viele Software Systeme bilden Geschäftsprozesse der realen Welt nach
- Geschäftsprozesse unterliegen ständigen Änderungen
  - passiv durch Adaption an neue Gegebenheiten (neue rechtliche Gegebenheiten, Marktsituation, Euro, Basel II, …)
  - aktiv durch die Einführung neuer Produkte, neuer Prozesse (Business Process Reengineering (BPR))
- Daher muss die Software laufend an die sich ändernden Geschäftsprozesse angepasst werden

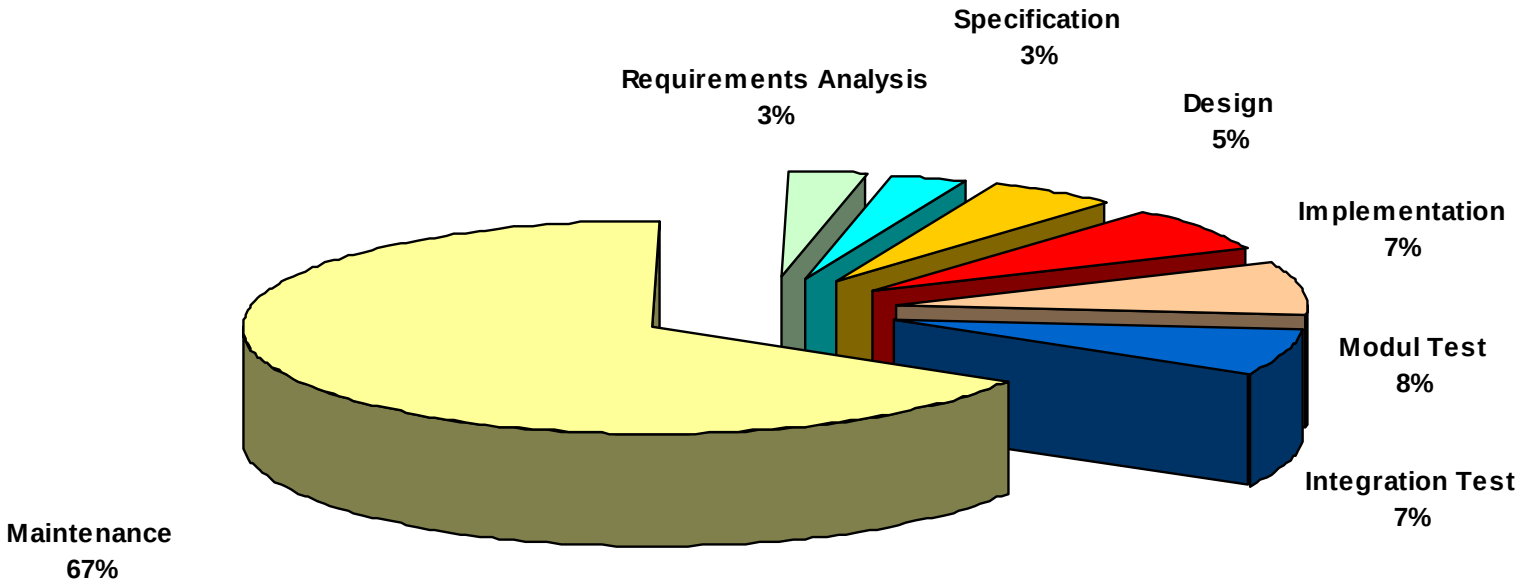
Warum Beschäftigung mit Software Wartung/Evolution?

- „Nevertheless, the industrial track record raises the question, why, despite so many advances, [...]
  - satisfactory functionality, performance and quality is only achieved over a *lengthy evolutionary process*,
  - software maintenance *never ceases* until a system is scrapped
  - software is still generally regarded as the *weakest link* in the development of computer-based systems“.
- [Lehman et al., 1997]

Software Wartung im Vergleich



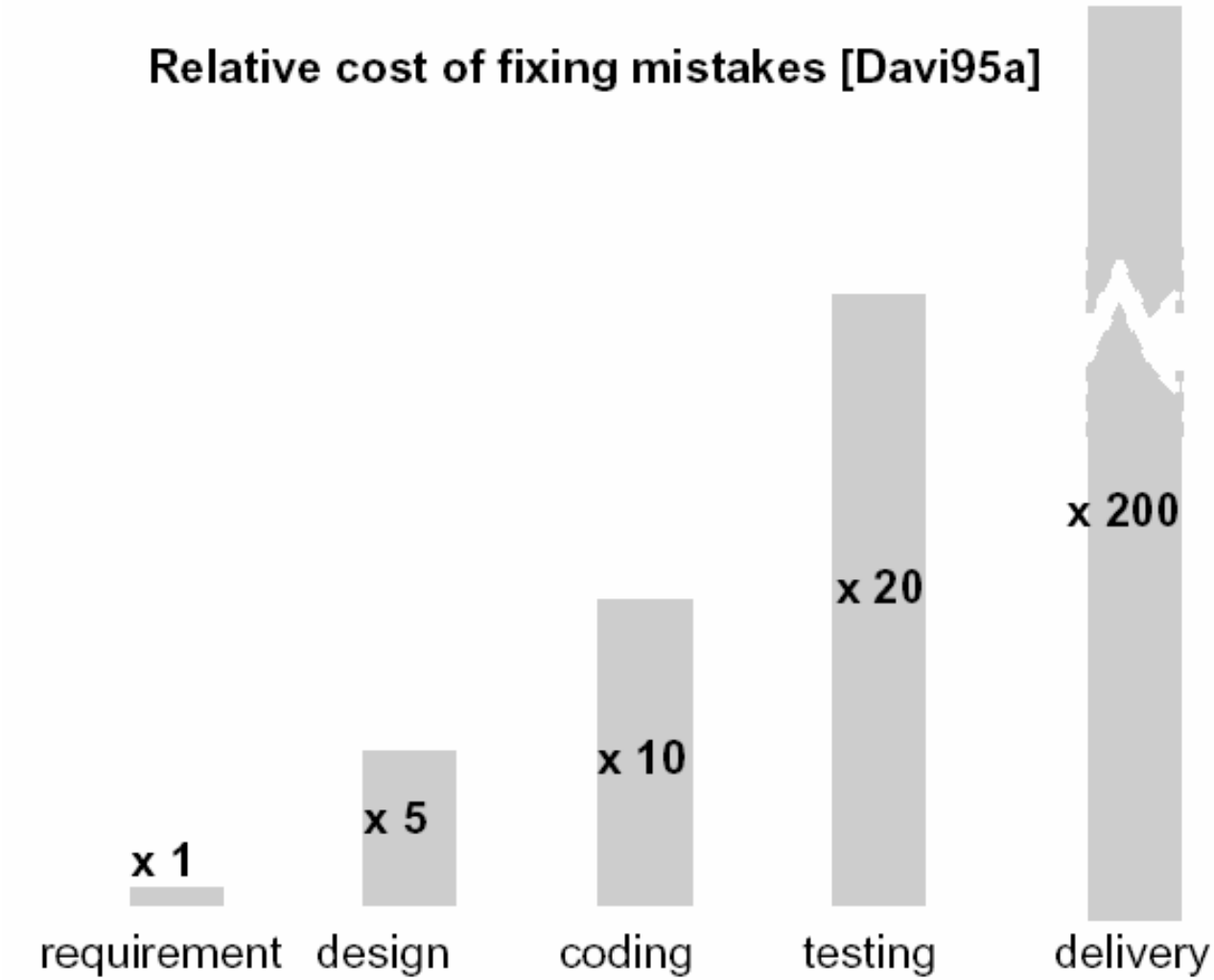
Cost Distribution in the Software Life-Cycle



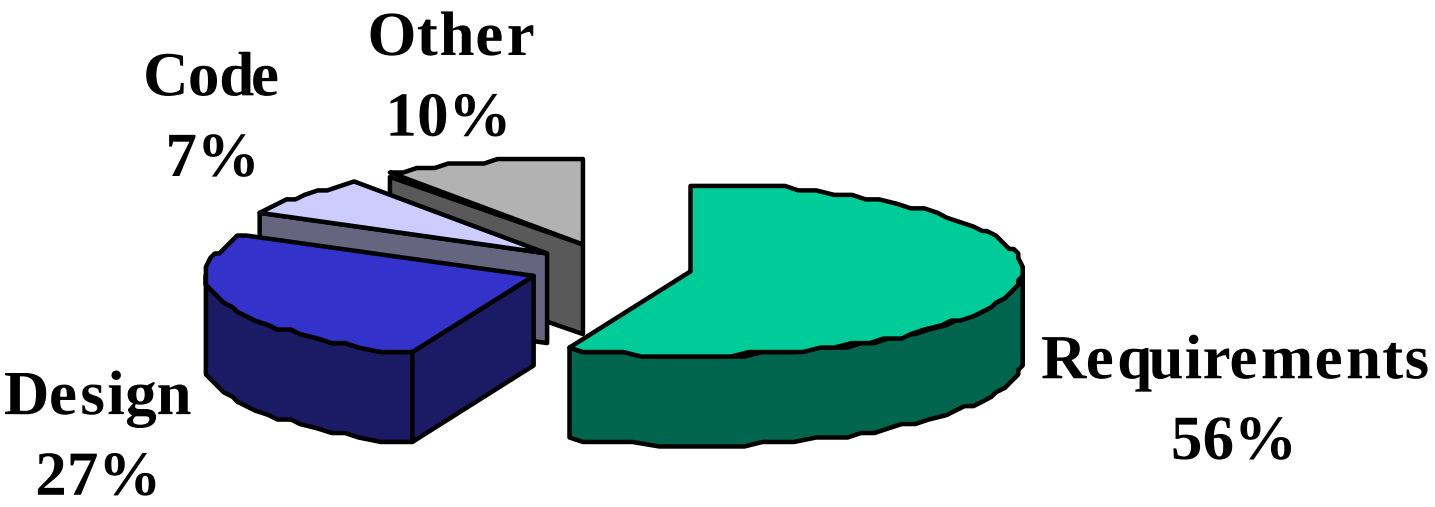
Cost Distribution in the Software-Life-Cycle

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

Cost of fixing bugs per phase



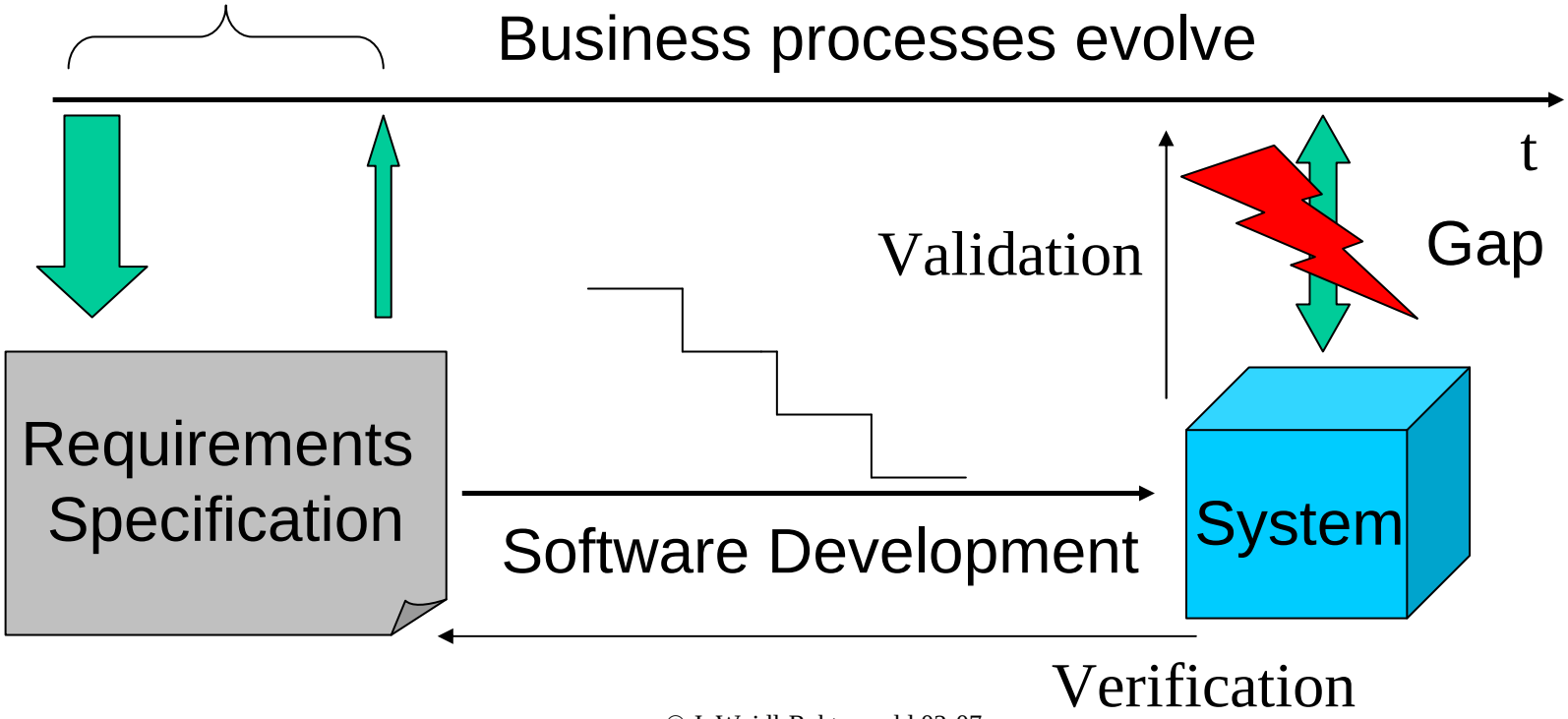
Distribution of Bugs



Source: Klösch/Gall, „Objektorientiertes Reverse Engineering“, Springer

Classic Approach to Software Development

„Requirements Fixing“



Difference software maintenance vs. software development

- Maintenance is similar to software development
  - Some unique skills and processes are employed:
    - Have intimate knowledge of system structure and content
    - Perform impact analysis and know the ripple effect
    - Problem solving skills
      - „Programmers have become part historian, part detective, and part clairvoyant.“ (Corbi 1989)
    - Track and control changes
    - Maintenance Outsourcing
    - Maintenance Cost Estimation

[Internet]

Software Wartung: Definition

Definition: Software Wartung

- *Nach IEEE Std. 610.12-1990 bzw. IEEE Std. 1219-1998*
- Software Wartung ist die Modifikation eines Software-Produktes oder einer Komponente, nach der Auslieferung, mit dem Zweck
  - der Fehlerkorrektur
  - der Verbesserung der Performance oder anderer Systemattribute
  - der Adaptierung an eine geänderte Umgebung

Definition: Software Wartung

- *Nach Barry Boehm*
  - “The process of modifying existing operational software while leaving its primary function intact”
- *Abstrakt*
  - „Preserve the value of software over time“
- Center for Software Maintenance
  - **Software maintenance** is the set of activities, both technical and managerial, that ensures that software continues to meet organizational and business objectives in a cost-effective way.

Allgemeinere Definition [SWEBOK]

- Software Maintenance
  - The totality of activities required to provide cost-effective support to a software system.
  - Activities are performed during the predelivery stage as well as the postdelivery stage.
  - Predelivery activities include planning for the postdelivery operations, supportability, and logistics determination.
  - Postdelivery activities include software modifications, training, and operating a help desk.

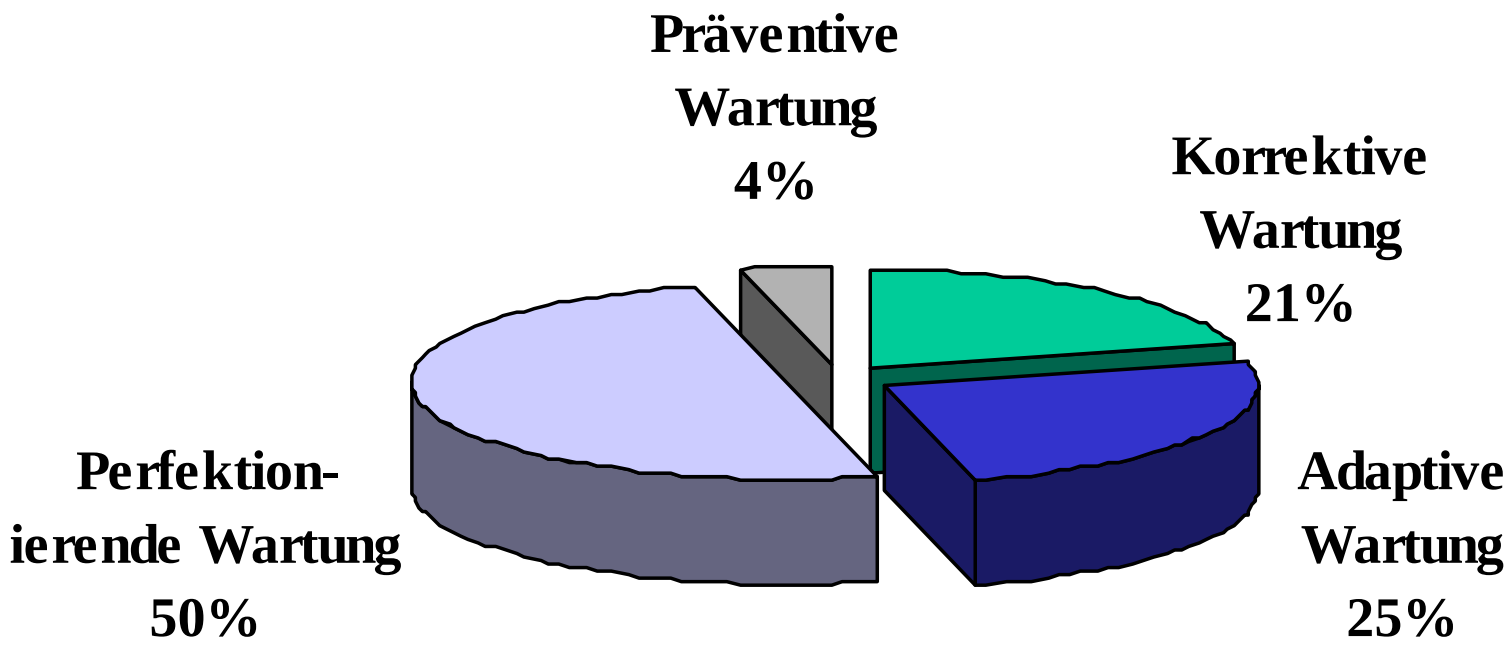
Source: Software Engineering Body of Knowledge, http://www.swebok.org/

Arten der Software Wartung

- ❶ Korrektive Wartung
  - „Bug fixing“; reaktive Natur
- ❷ Präventive Wartung
  - Finden von latenten Fehlern, bevor sie effektive Fehler werden
- ❸ Adaptive Wartung
  - Neue Hardware, Betriebssysteme etc.; neue Anforderungen
- ❹ Perfektionierende Wartung
  - Verbesserungen von Performance und Wartbarkeit (Restructuring, Reverse Engineering, Dokumentationspflege, etc.)
- ❶ + ❷: Corrections /❸ + ❹: Enhancements

Source: Klösch/Gall, „Objektorientiertes Reverse Engineering“, Springer

Arten der Software Wartung



Source: Klösch/Gall, „Objektorientiertes Reverse Engineering“, Springer

Warum ist Software Wartung nicht-trivial?

Motivation

Typische Eigenschaften von Software

- Programme sind nur selten in sinnvolle Modulstrukturen aufgeteilt, und wenn, ist diese Aufteilung meist sehr willkürlich
- Redundanzen in Daten bzw. Funktionen sind ein steter Bestandteil eines Programms
- Die Sichtbarkeitsbereiche von Daten und Funktionen sind meist weiter ausgedehnt aus für das Programm notwendig bzw. überhaupt sinnvoll

Typische Eigenschaften von Software

- Trace Ausgaben sind spärlich, haben keinen Timestamp und keine Quellenangabe
- Durch Änderungen am Code verändert sich das Laufzeitverhalten durch Gleichzeitigkeitsprobleme („race conditions“) nichtdeterministisch
- Dieselbe Methode wird durch ein Flag für unterschiedliche Berechnungen genutzt
- Methoden bleiben bei der Klasse für die sie ursprünglich geschrieben wurden, bei einer Änderung der Funktionalität werden sie nicht zur am Besten geeigneten Klasse verschoben

Typische Eigenschaften von Software

- Variablennamen sind semantisch wertlos
  - int work = 0;
- Variablen werden kontext-abhängig verwendet
  - Fall 1: work speichert Kundennummer
  - Fall 2: work speichert Fakturensumme
- Die Dynamik des Programmdurchlaufs ist während der statischen Code Inspektion nicht oder nur schwer ableitbar

Schwierigkeiten in der Software Wartung

- Missing
  - Development environment (tools, scripts, etc.)
  - Build environment
  - Source code
  - Documentation
  - Design Decisions
  - Domain knowledge
  - Original programmer team / analysts

Schwierigkeiten in der Software Wartung

- Wartung ist ereignisgesteuert (planbar?)
- In der korrektiven Wartung wird nach Meldung eines Fehlers verlangt, die Ursache so schnell als möglich zu finden und den Fehler zu beseitigen (also „quick fix“!), trotz des weiterlaufenden Tagesgeschäftes
- Das Wartungspersonal steht daher meist unter Zeitdruck und das Management und die Dokumentation des Wartungsfalls werden vernachlässigt

Schwierigkeiten in der Software Wartung

- Laufende Wartung erhöht die „Software Entropie“
  - Verklärt
    - Architektur
    - Design
    - Modularisierung
  - Erhöht
    - Abhängigkeiten („Coupling“)
  - Vermindert
    - Orthogonale Trennung („Cohesion“)

Schwierigkeiten in der Software Wartung

- Änderungen an einem Software System sind zwar operativ leicht durchführbar, die Schwierigkeit ist aber, die richtige Änderung durchzuführen - und **nur** diese.

Schwierigkeiten in der Software Wartung

- Viele Probleme der Software Wartung treten erst im Zusammenhang mit großen, alten, komplexen Software Systemen auf, so genannten „*Legacy Systemen*“.
- Diese wurden mit Methoden konzipiert und in Sprachen erstellt, die heute kaum mehr benutzt (und noch weniger gelehrt) werden
  - Strukturierte Analyse, proprietäre Analyseansätze
  - Mumps, FORTRAN, PL/I

State-of-the-Art: „7x24“

- 7 Tage in der Woche 24 Stunden online
- Hardware Hersteller werben mit 99,999 Prozent Verfügbarkeit ihrer Systeme (entspricht 5 Minuten 20 Sekunden Downtime im Jahr)
- Wann werden dann neue Versionen eingespielt?
- Trend zu Applikationsservern, die Wartung zur Laufzeit unterstützen
  - 2. Instanz mit adaptierter Software fährt hoch
  - Die 2. Instanz übernimmt zur Laufzeit
  - Die ursprüngliche Instanz geht außer Betrieb
  - Problem der Datenmigration!

Geschichte der Software Wartung

- Die Wartung von Programmen galt von jeher als *unbeliebte* Tätigkeit im Software Life-Cycle
  - Historisch die Arbeit von neu rekrutierten bzw. nicht so erfahrenen Programmierern
- Das heißt
  - **wenig Know-How**
  - **keine Werkzeuge**
- Folgen
  - **„quick fix“ Modell wird angewandt**
  - **Wartung führt zu noch mehr Fehlern**
  - **Wartung kann aufgrund von steigender Komplexität gar nicht mehr durchgeführt werden**

Evolutionäre Probleme der Software Wartung

- Die Komplexität wächst mit jedem Wartungseingriff
- Daher vergeht zwischen den Wartungseingriffen immer mehr Zeit
- Die Produktivität der Wartungseingriffe sinkt, die Kosten pro Eingriff steigen
- Dies alles geschieht nach Gesetzmäßigkeiten („Laws of Software Evolution“)

Das Pareto Prinzip im Software Engineering bzw. in der Wartung

- 20% der Requirements bedingen 80% der Komplexität
- 80% des Systems sind in 20% der Zeit fertig gestellt
- 20% des Codes beinhalten 80% der Fehler
- 80% der Fehler werden in 20% der Zeit behoben
- Nach Vilfredo Pareto (1848-1923)
  - Italienischer Ökonom und Gesellschaftstheoretiker
  - Dieses Prinzip besitzt auch in vielen anderen Bereichen Gültigkeit (z.B. Zeitmanagement, Verkauf)

POEM - David H. H. Diamond

- The fellow who designed it,
  Is working far away;
  The spec’s not been updated,
  For many a livelong day.

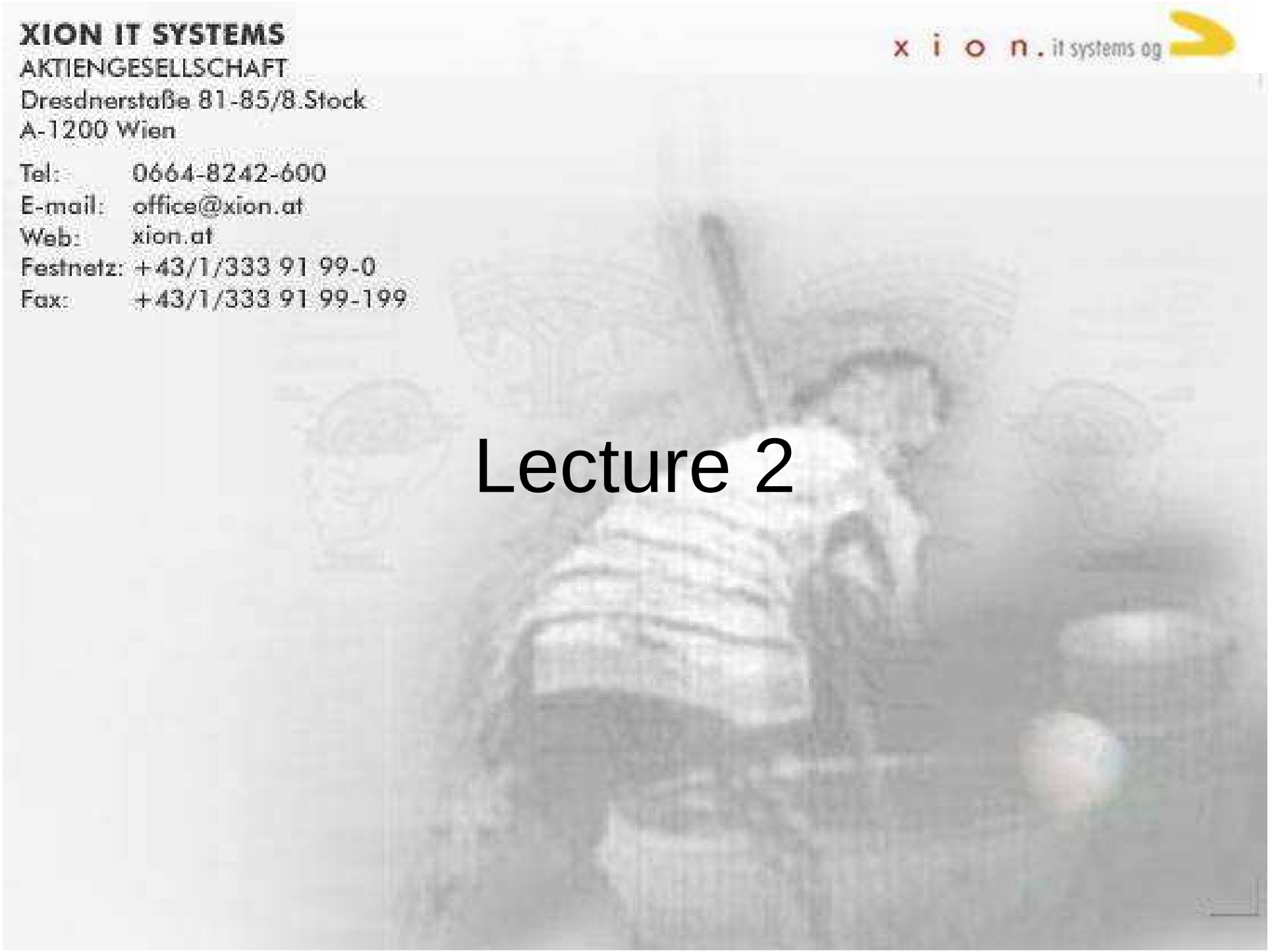
- The guy who implemented it is
  Promoted up the line;
  And some of the
  enhancements
  Didn’t match to the design.

- They haven‘t kept the flowcharts,
  The manual‘s a mess,
  And most of what you need to
  know
  You‘ll simply have to guess.

- We do not know the reason,
  Why the bugs pour in like rain,
  But don‘t just stand here gaping,
  Get out there and MAINTAIN.

XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 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 og



Lecture 2

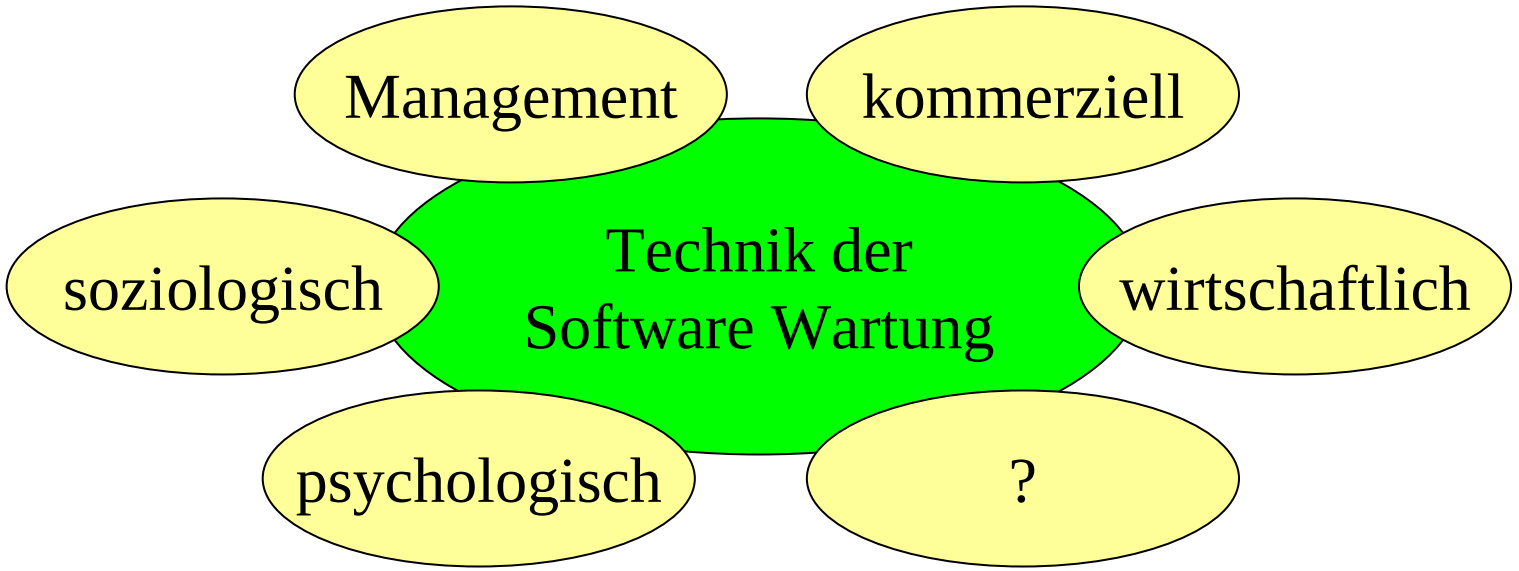
Lecture 2

- 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?

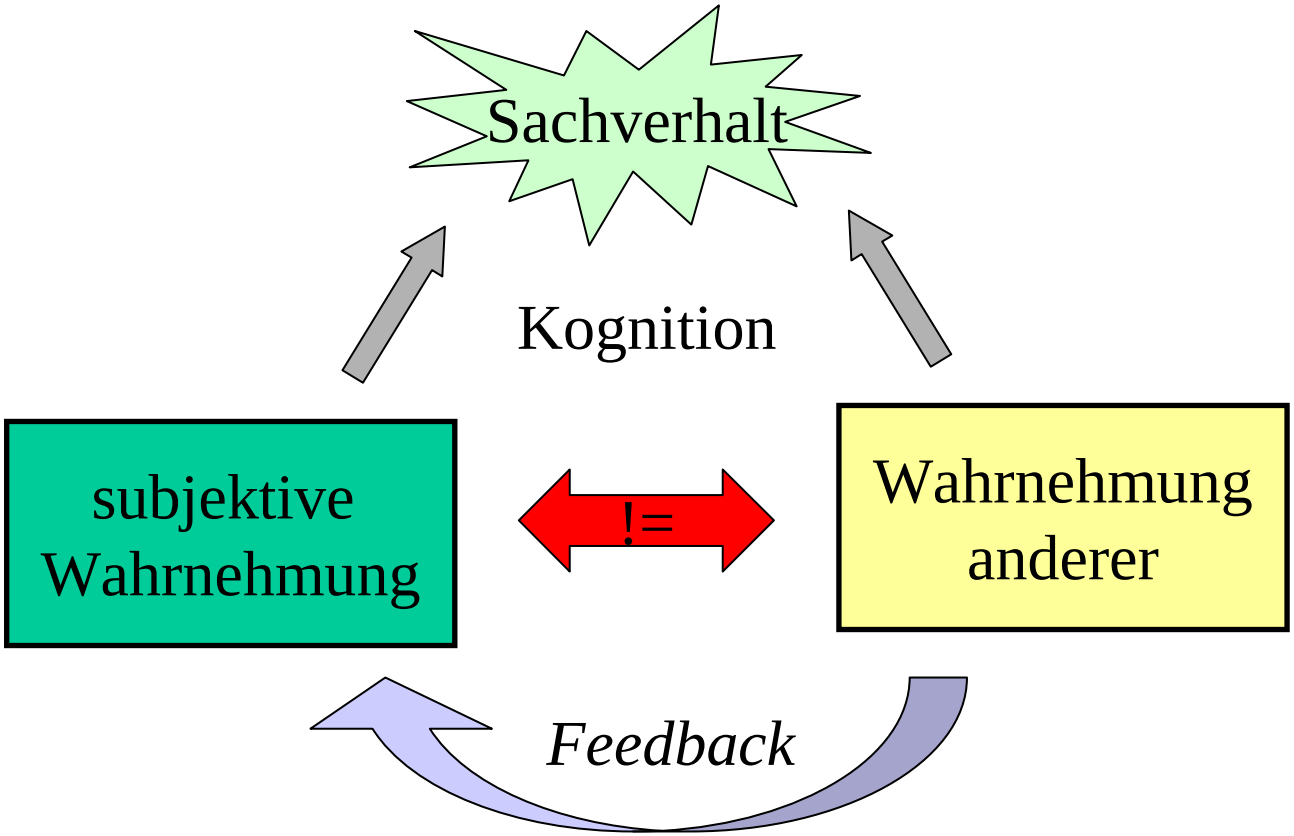
Aspekte der Software Wartung

- Sociological
  - Second hand / apprentice‘s work
  - Maintaining morale
- Psychological
  - Hesitate to touch a huge work of art
  - Fear to damage a working system constantly
  - „Kognitive Dissonanz“

Kognitive Dissonanz

- **Definition (nach** L. FESTINGER)
  - K. D. ist die Bezeichnung für einen unangenehm erlebten Zustand, der aus widersprüchlichen Erfahrungen oder Einstellungen in Bezug auf den gleichen Gegenstand hervorgeht und nach einer Veränderung verlangt.
  - Um den als unangenehm erlebten Zustand aufzulösen, müssen neue, spannungslösende Schritte eingeleitet werden. Dies lässt sich auf mehrere Arten erreichen, z.B. durch die Suche nach neuen **Informationen**, durch eine Einstellungsänderung, durch ein Handlungsänderung usw.
  - Die Theorie der k. D. beschäftigt sich mit der Verarbeitung von wichtigen Informationen, nachdem eine Entscheidung gefällt wurde.
    - Die Theorie besagt, dass nach einer getroffenen Entscheidung vorzugsweise die Informationen ausgewählt werden, die die Entscheidung als richtig erscheinen lassen, und dass gegenteilige Informationen nicht mehr beachtet werden.

Kognitive Dissonanz



Kognitive Dissonanz: Beispiel

- Sachverhalt: **Die Software funktioniert nicht so, wie sie sollte**
- Wahrnehmung Programmierer: „Ich bin mit dem Programm fertig. Es funktioniert.“
- Wahrnehmung Quality Assurance: „Dieser Testfall funktioniert nicht.“
- Feedback QA: „In ihrem Programm ist ein Bug“
- -> kognitive Dissonanz: „Ich bin fertig und das (weniger das Ergebnis, sondern der Zustand) gefällt mir“ **versus** „Du hast Mist gebaut!“
- Auflösung der KD:
  - „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

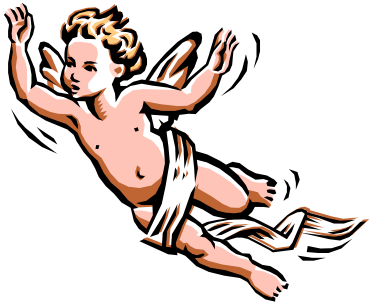
Egoless Programming

- 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!

Beispiel für den psycho-sozialen Aspekt

- aus Petr Kroha „Softwaretechnologie“ [Kroha97]
- Windows of Opportunity [Yourdon92] - beschreiben günstigsten Zeitpunkt zur Neuentwicklung eines Softwaremoduls:
  - „der zuständige Programmierer geht in Rente oder kündigt
  - wenn die Komponente extrem gravierende Fehler aufweist und der Programmierer die Suche aufgegeben hat
  - *wenn es endlich möglich ist, den Widerstand leistenden Programmierer zu entlassen.*“

Kommerzieller Aspekt



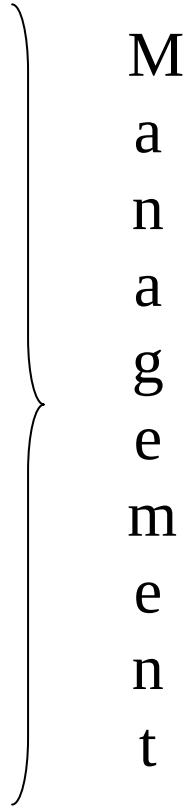
Budget, Time-to-market

Software Qualität

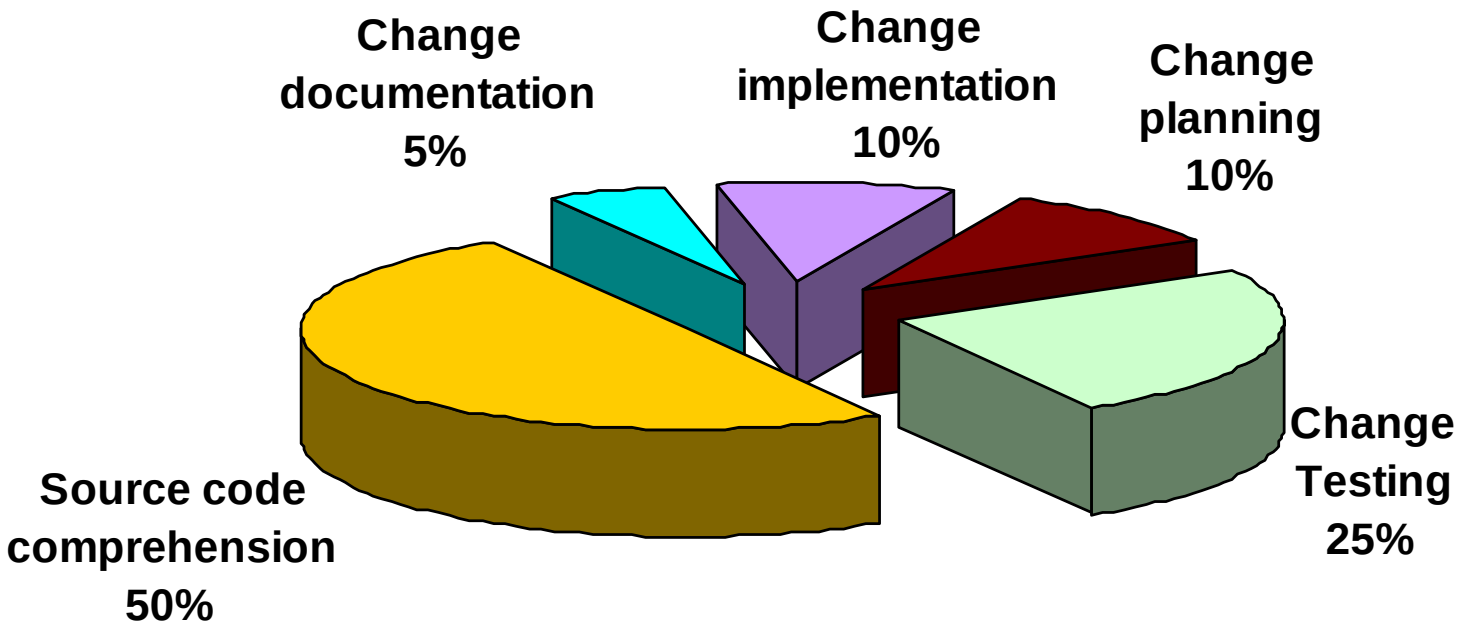
Aktivitäten der Software Wartung

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



Activities in Software Maintenance



Activities in Software Maintenance

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

Software Wartungskrise

Software Wartungskrise

- Unter der „Software Wartungskrise“ versteht man die Unangemessenheit der Prozesse und Werkzeuge, um den Vorgang der Software Wartung auch für komplexe, umfangreiche Systeme qualitativ hochwertig (ökonomisch und technisch) durchführen zu können.

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
  - 7x24 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, ESB)
- Komplexität
  - Neue Anforderungen können im System nicht mehr verwirklicht werden
  - Unzufriedenstellende Performanz (z.B. Transaktionsrate)

[Kroha97]

Probleme bei der 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 Techniken verwenden (Datenbank, 3-Tier, Objektorientierung, Middleware, AOP, SOA, ESB, Clustering, etc.)
- 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

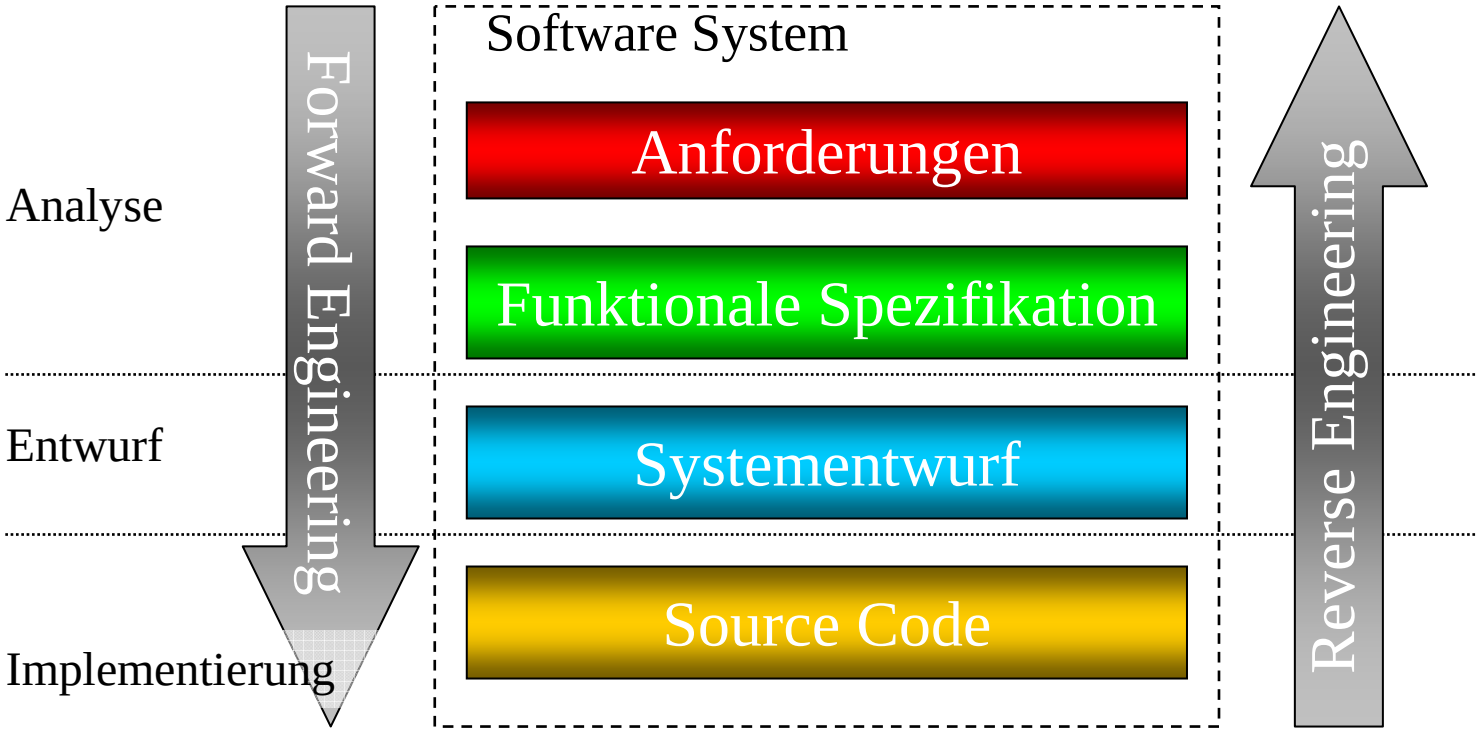
Reverse Engineering

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.

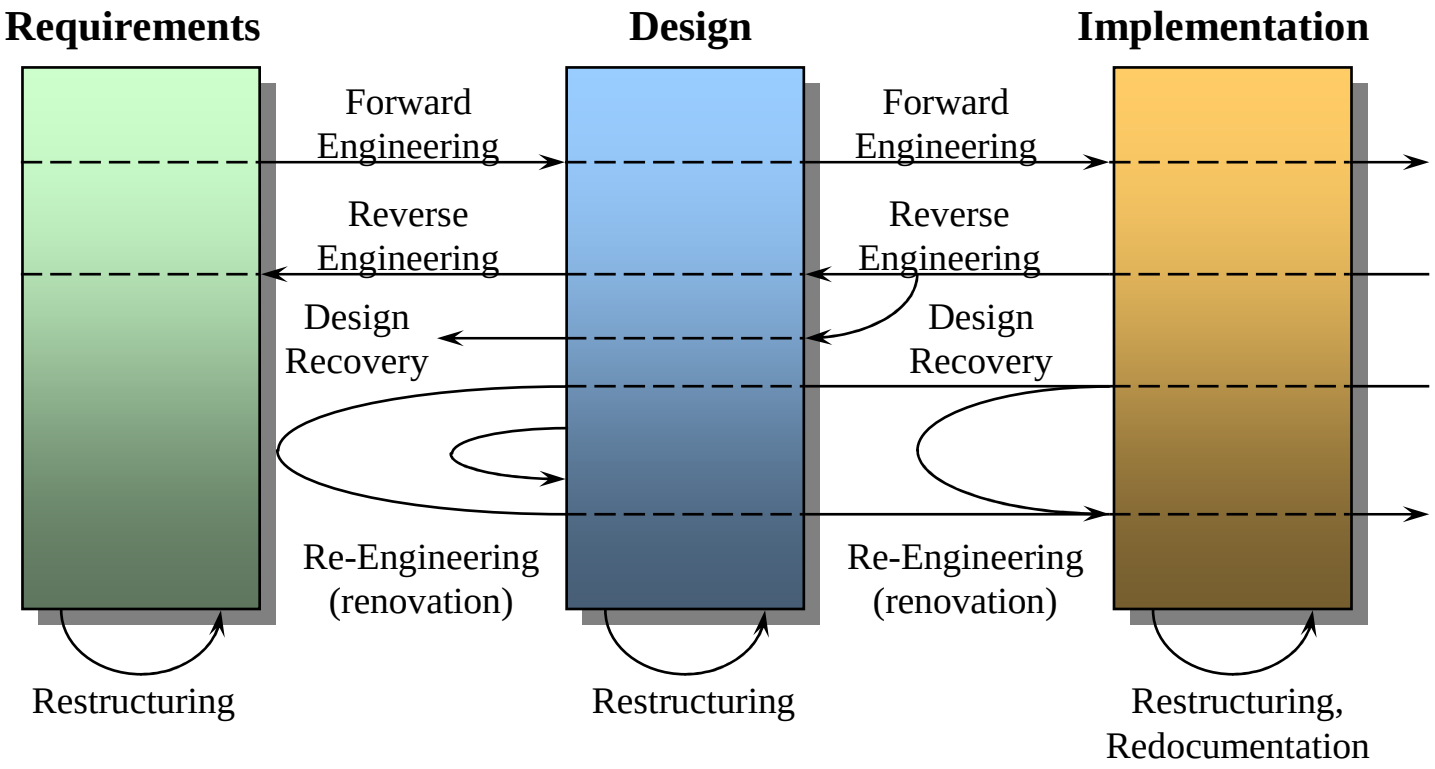
[Chikofsky/Cross]

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

Difficulties

- Dimensions
  - application domain (what) and program (how)
- Information
  - Specific (machine-level) and abstract (design-level)
- Formality
  - Formal structure of programs and informal human understanding

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, meist 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, ...

  } *Angereichert durch informale Information*

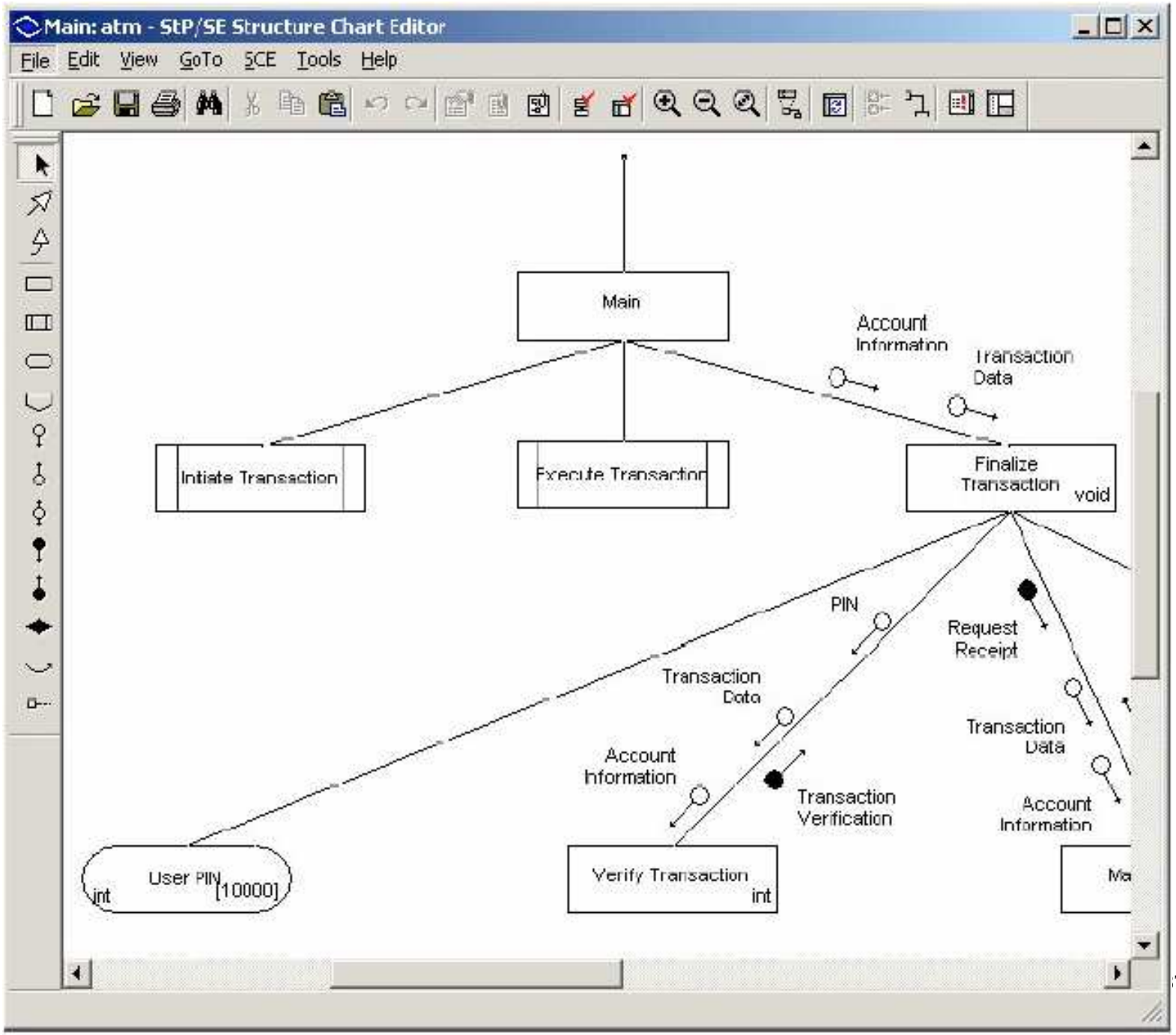
  - Informale Beschreibungen des Software Systems
    - Text
    - Diagramme

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

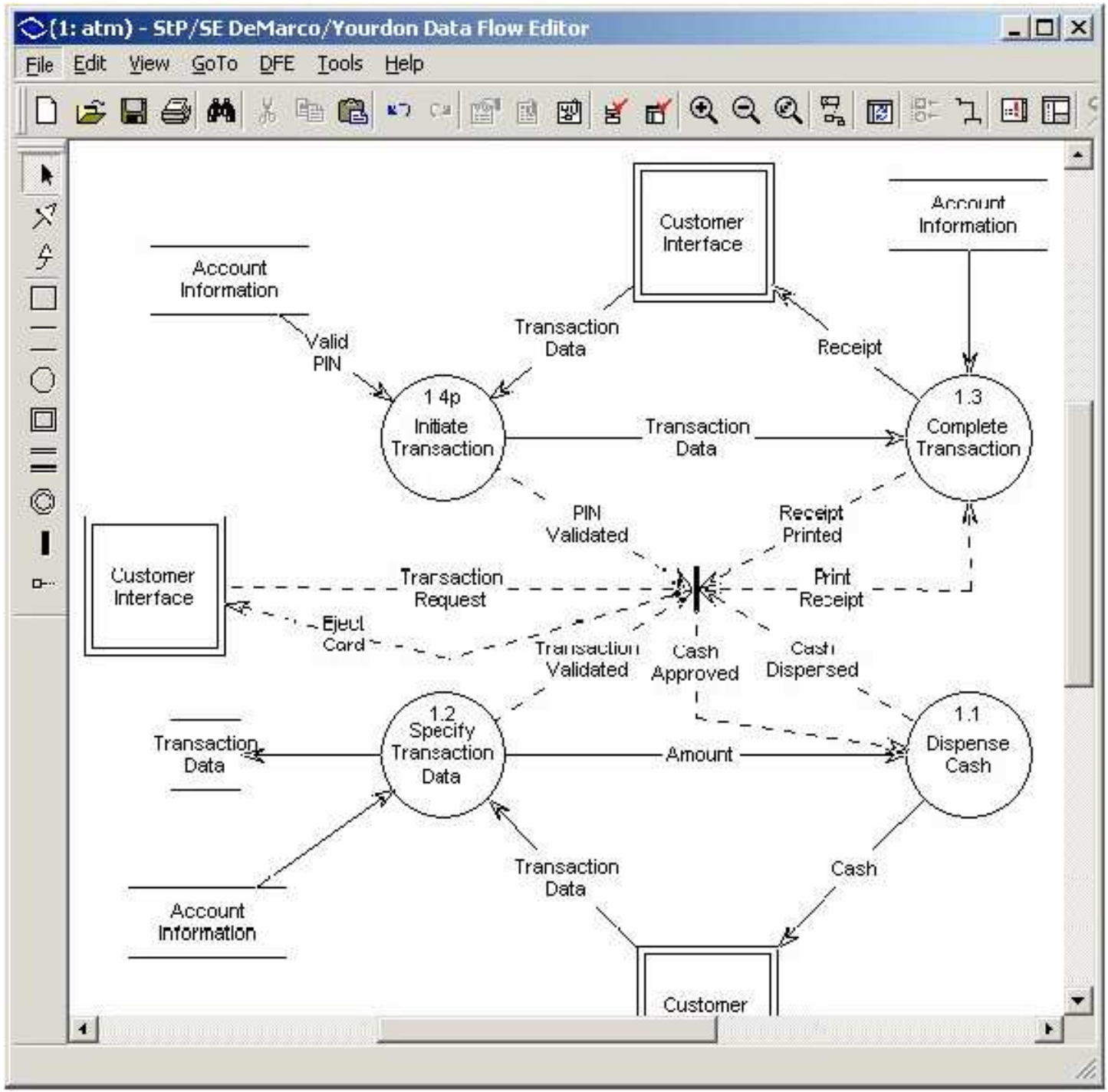

Main: atm - StP/SE Structure Chart Editor
File Edit View GoTo SCE Tools Help
Main
Account Information
Transaction Data
Initiate Transaction
Execute Transaction
Finalize Transaction
void
PIN
Request Receipt
Transaction Data
Transaction Data
Account Information
Transaction Verification
Account Information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

Nested Trees

- Stellen den impliziten Datenaustausch über globale Variablen dar
- Knoten
  - Repräsentieren Module
- Kanten
  - Repräsentieren die Definition eines Moduls in einem anderen
- Nested Tree ermittelt
  - Verschachtelung der Deklarationen von Prozeduren und Funktionen
  - Kann Gültigkeitsbereich für jede Variable im System darstellen

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
Trarsaction Request
Print Receipt
Ejeut Card
Transaction Validated
Cash Approved
Cash Dispersed
1.2 Specify Transaction Data
Amount
1.1 Dispense Cash
Transaction Data
Account Information
Transaction Data
Cash
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 deals 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 bzw. Code Comprehension Tools

- Imagix4D (Imagix Corp.)
  - www.imagix.com
- Software Refinery, Refine/C (Reasoning Systems Inc.)
- Sniff+ (Wind River)
  - http://www.windriver.com/products/development_tools/ide/sniff_plus/
- Source Navigator
  - http://sourcenav.sourceforge.net
- Rational Rose
  - http://www.ibm.com/software/rational
- Software through Pictures SE (Aonix: StP/SE)
  - http://www.aonix.com/stp_se.html
- Sotograph
  - www.software-tomography.com

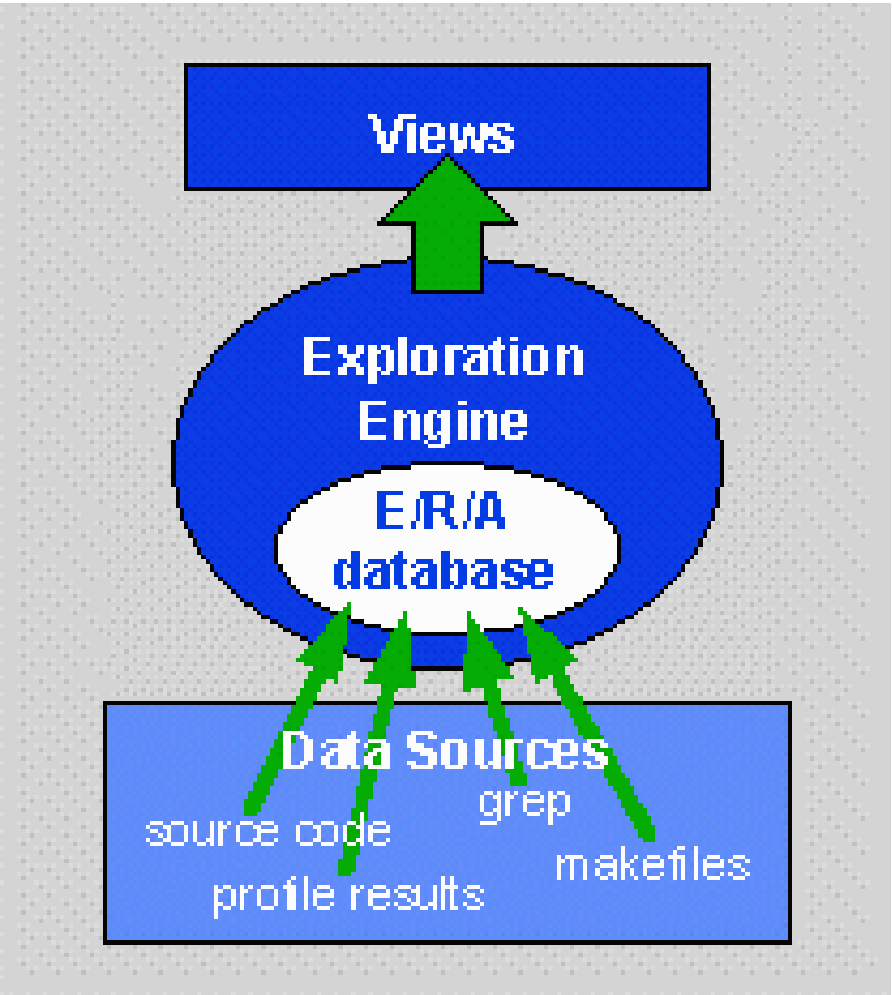
Die Tool Falle

- *“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

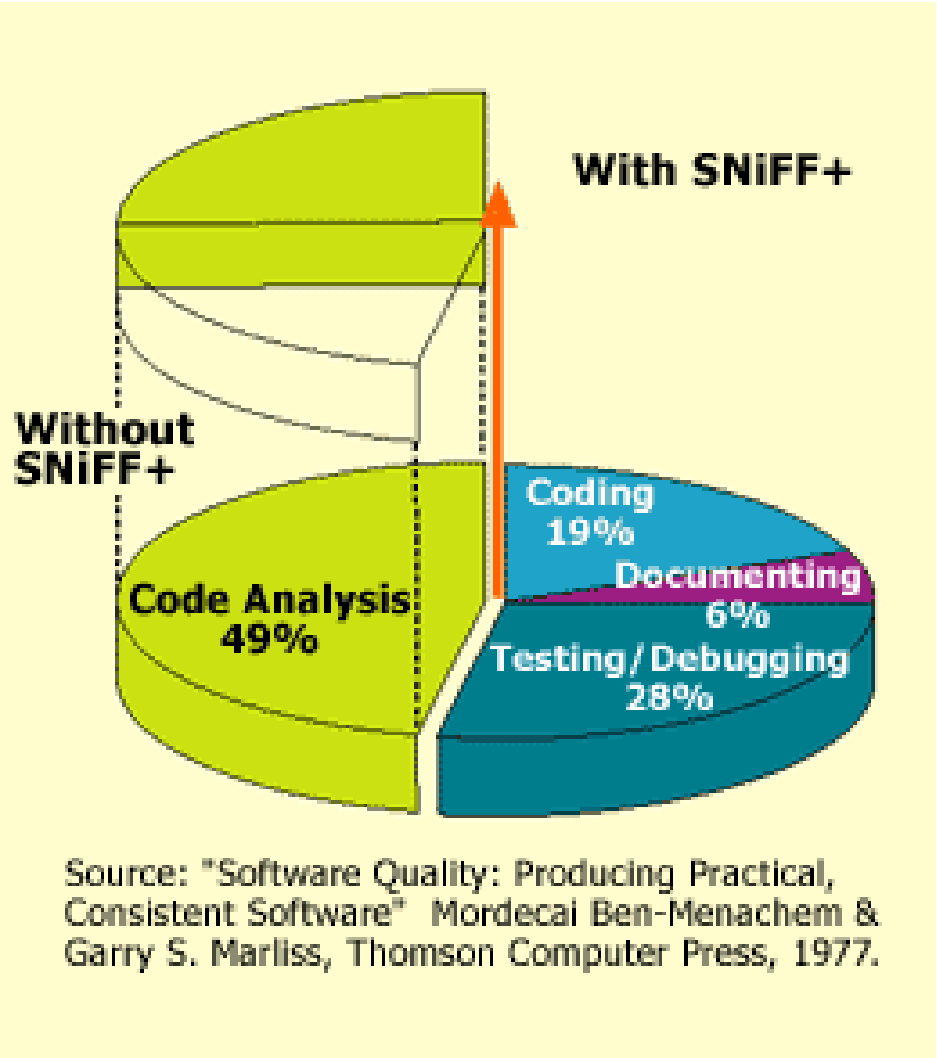
Imagix 4D: Funktionen

- Only for C/C++
- Provides graphical 3D output of analysed source
- Control Flow Analysis
  - Generation of augmented call graphs
- Annotated Flow Charts
  - Shows the logic flow for complex constructs
- Use Browser
  - Shows where and how a symbol is used
- Automated generation of HTML documentation

Imagix 4D: Toolkit



Sniff+



Source: "Software Quality: Producing Practical, Consistent Software" Mordecai Ben-Menachem & Garry S. Marliss, Thomson Computer Press, 1977.

Sniff+: Funktionen

- Source code editor
- Class browser
- Inheritance hierarchy browser
- Xref – Cross reference browser (uses/used)
- Include browser (Java: imports)
- Retriever – search for symbol names
- Grep
- Interface to debugger, build environment, revision control

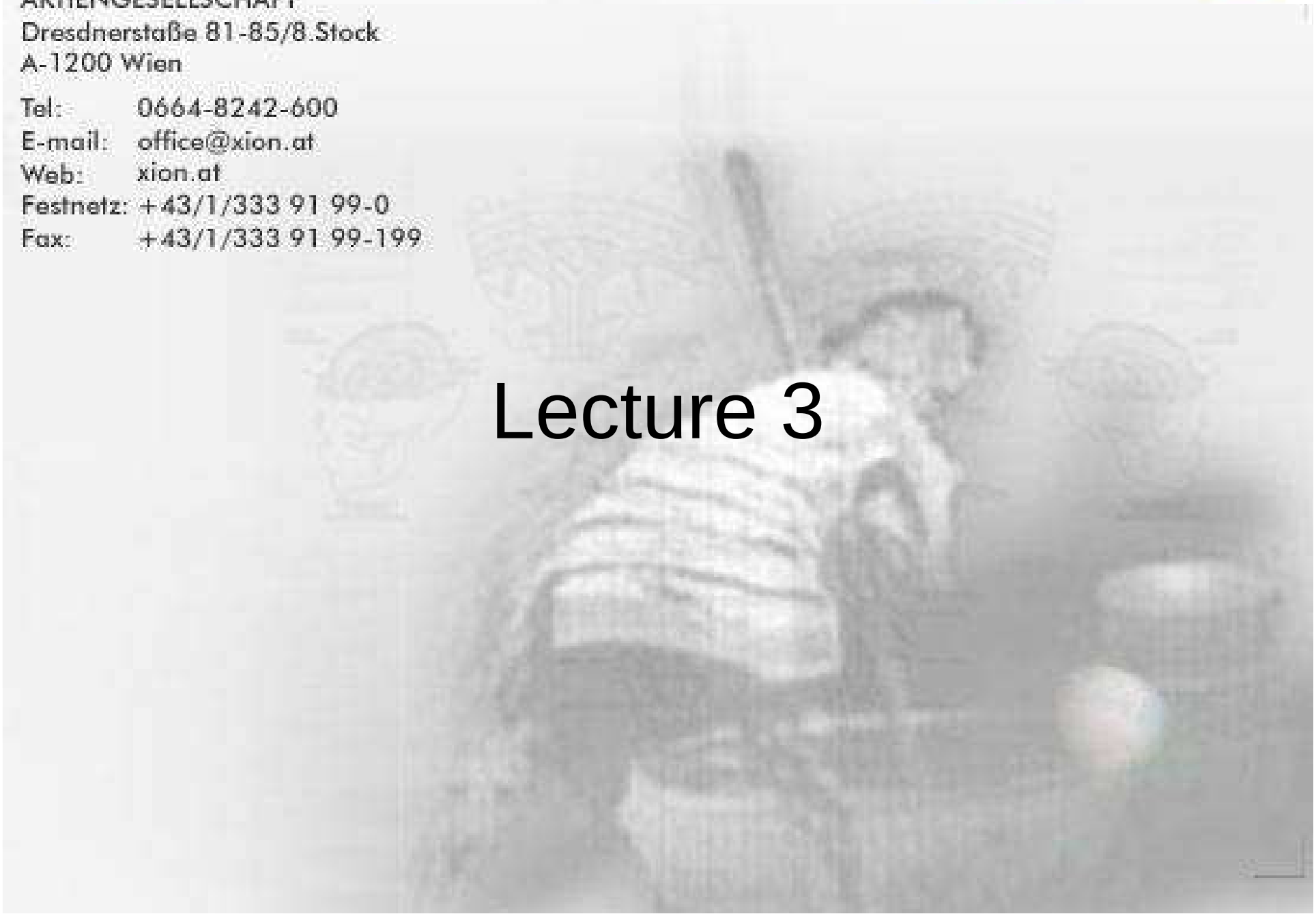
Software Refinery

- Programming-language specific parsers (Fortran, C, Cobol, Ada)
- Standard analyzers (xref, def/use, call tree)
- OO Repository for holding results of analyses
- Wide-spectrum language for writing custom analyses
- Extensions: dialect (parser), intervista (GUI)

XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 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 og

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).

[Chikofsky/Cross]

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

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]

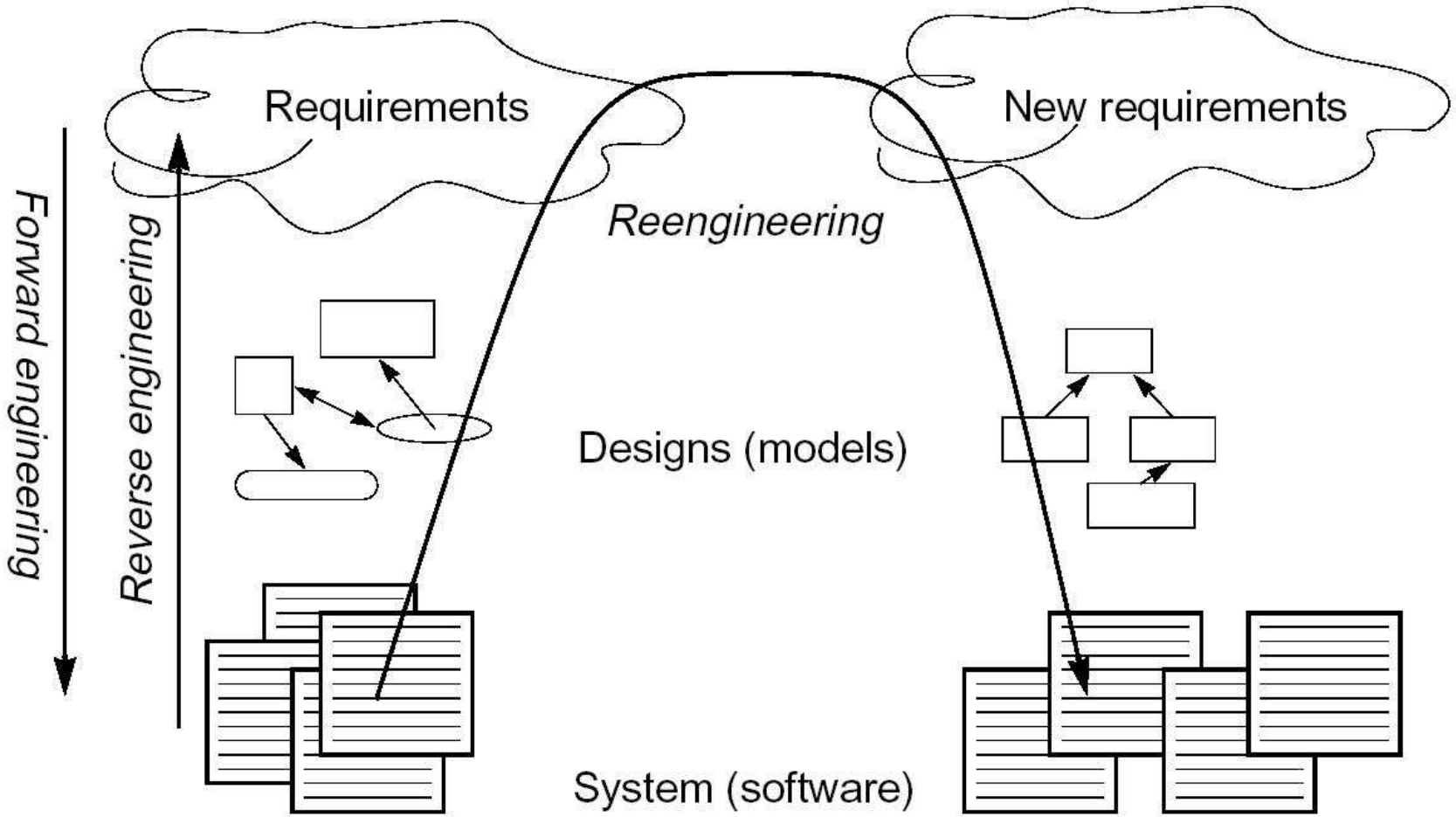
Re-Engineering

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

[Chikofsky/Cross]

Reverse and Reengineering



Re-Engineering: Probleme

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

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**

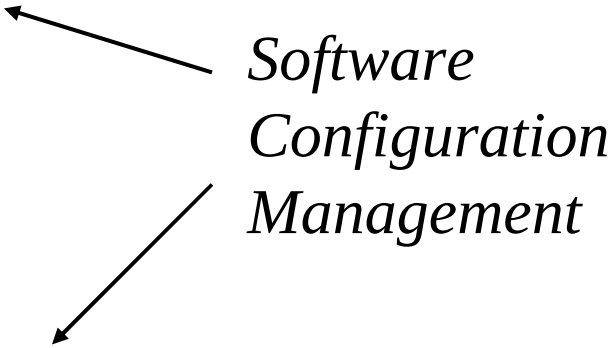
Aktivitäten des Wartungsfalles (vgl. Lecture 1)

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

Management

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



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.)

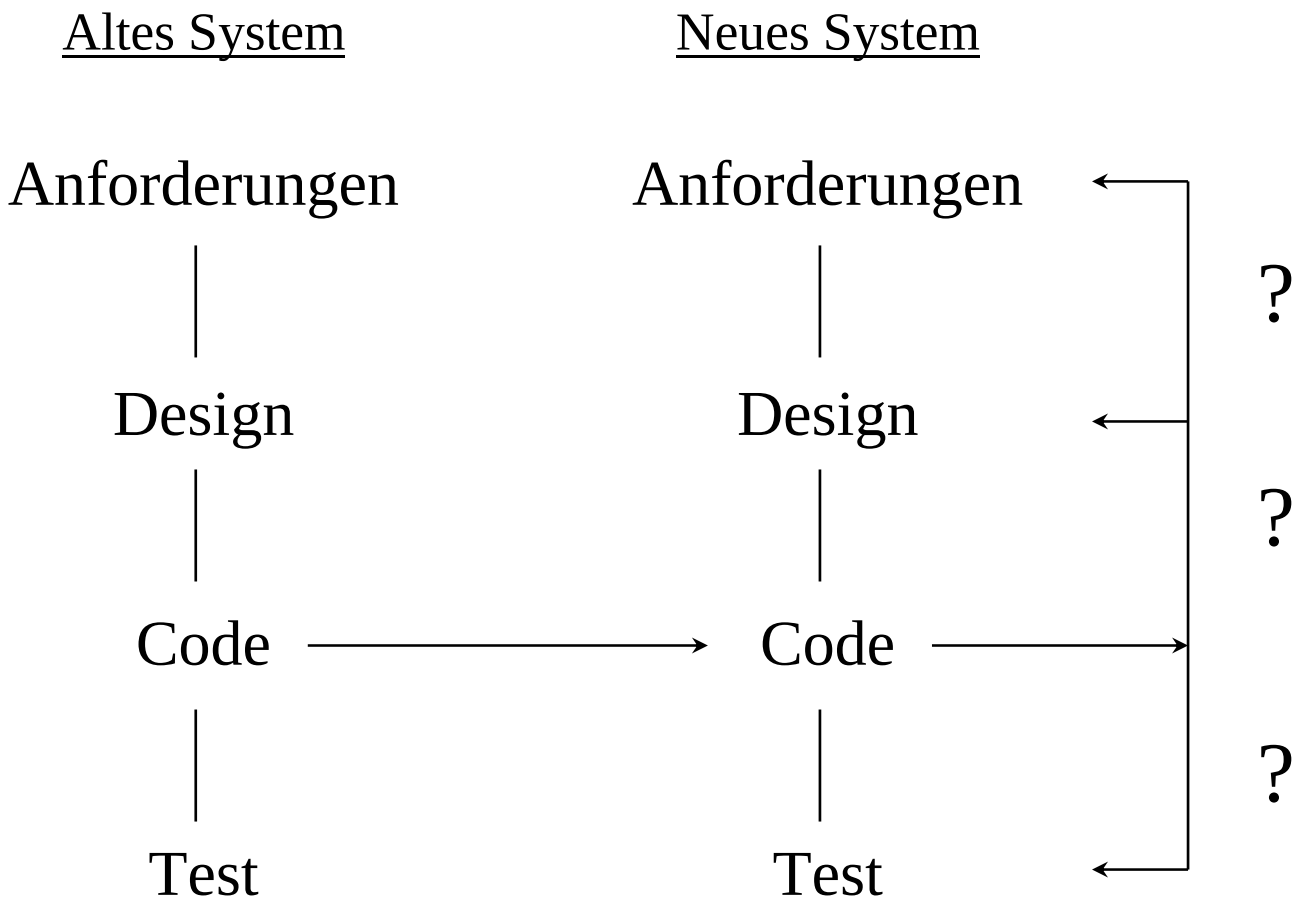
Life Cycle Modelle der Software Wartung

- 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

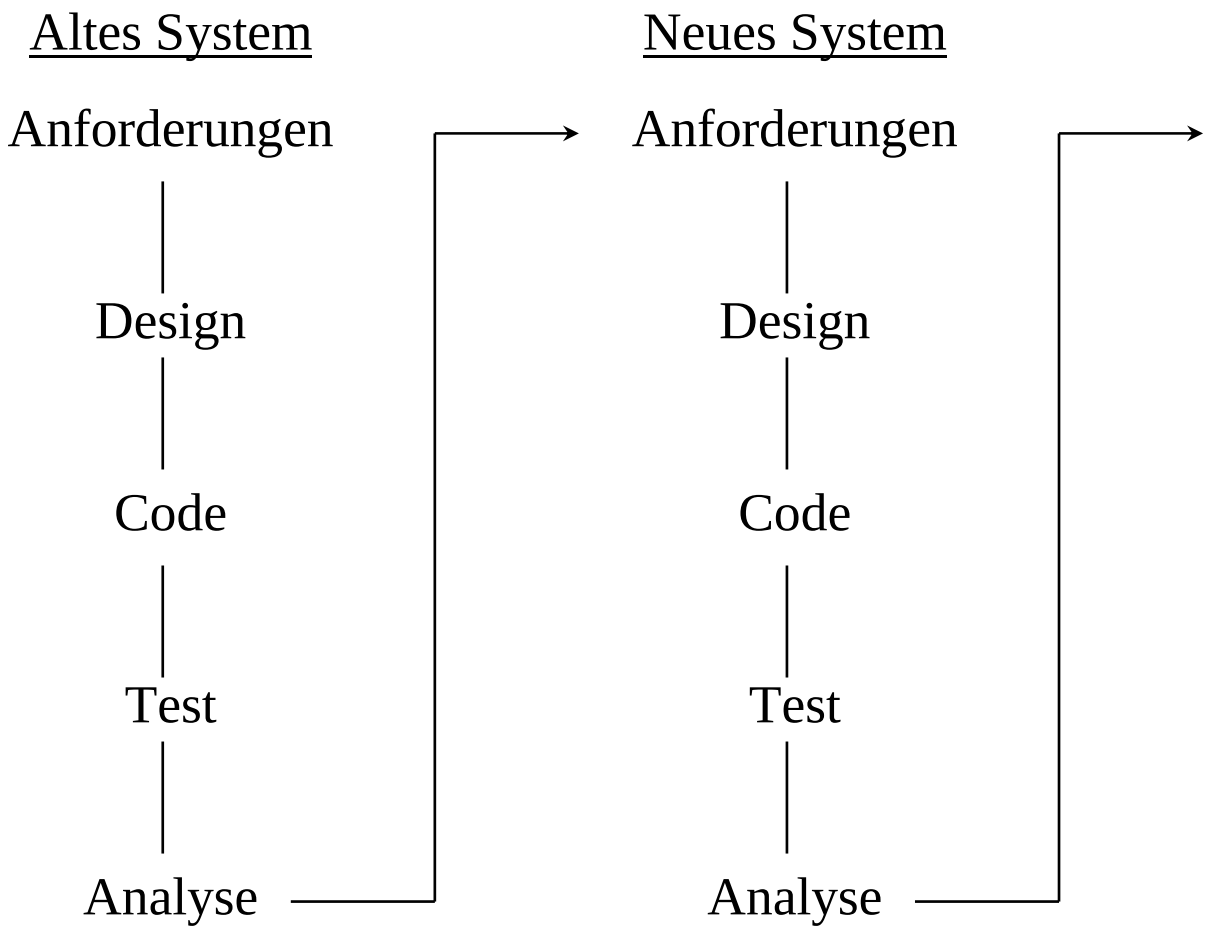
Quick Fix Model



Quick Fix Model: Folgen

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

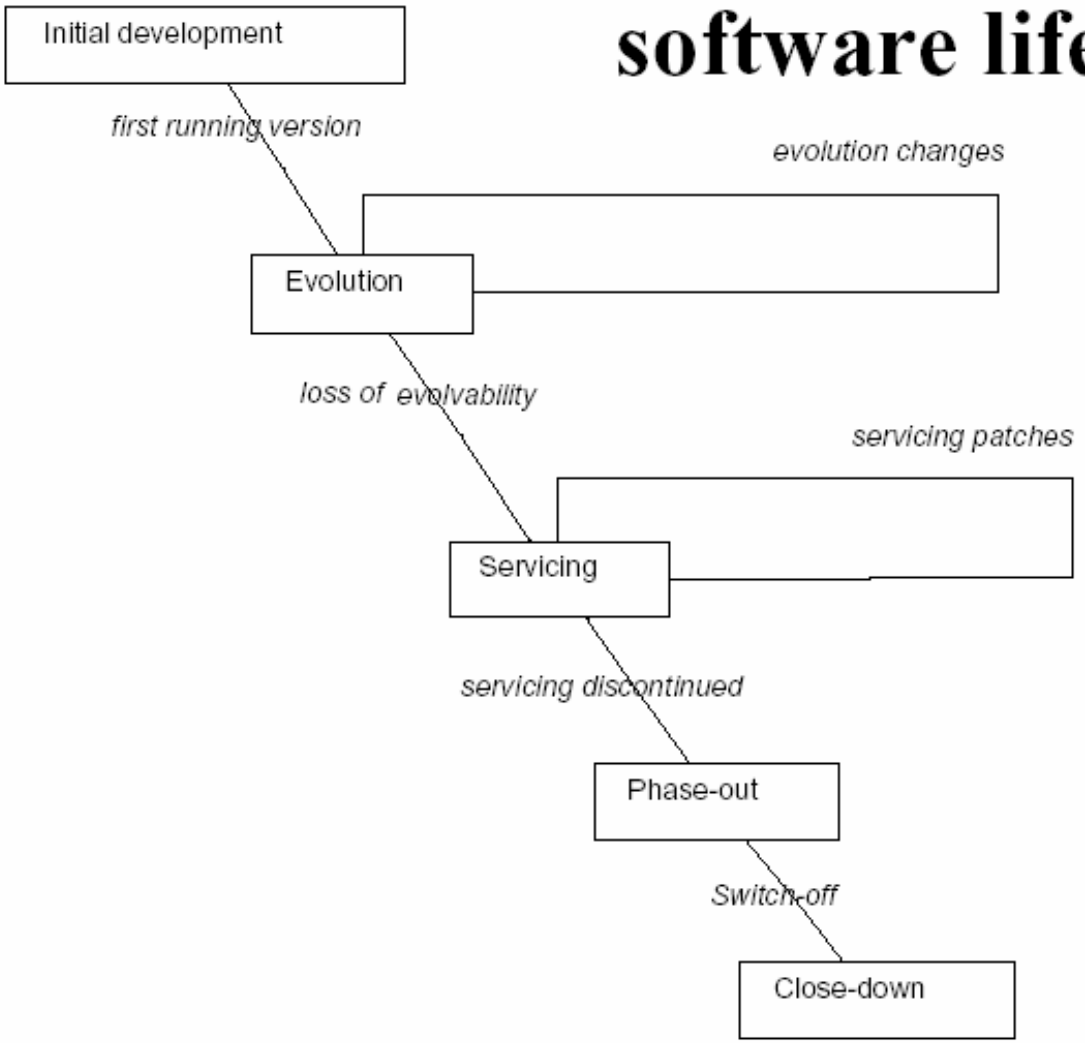
Iterative Enhancement Model



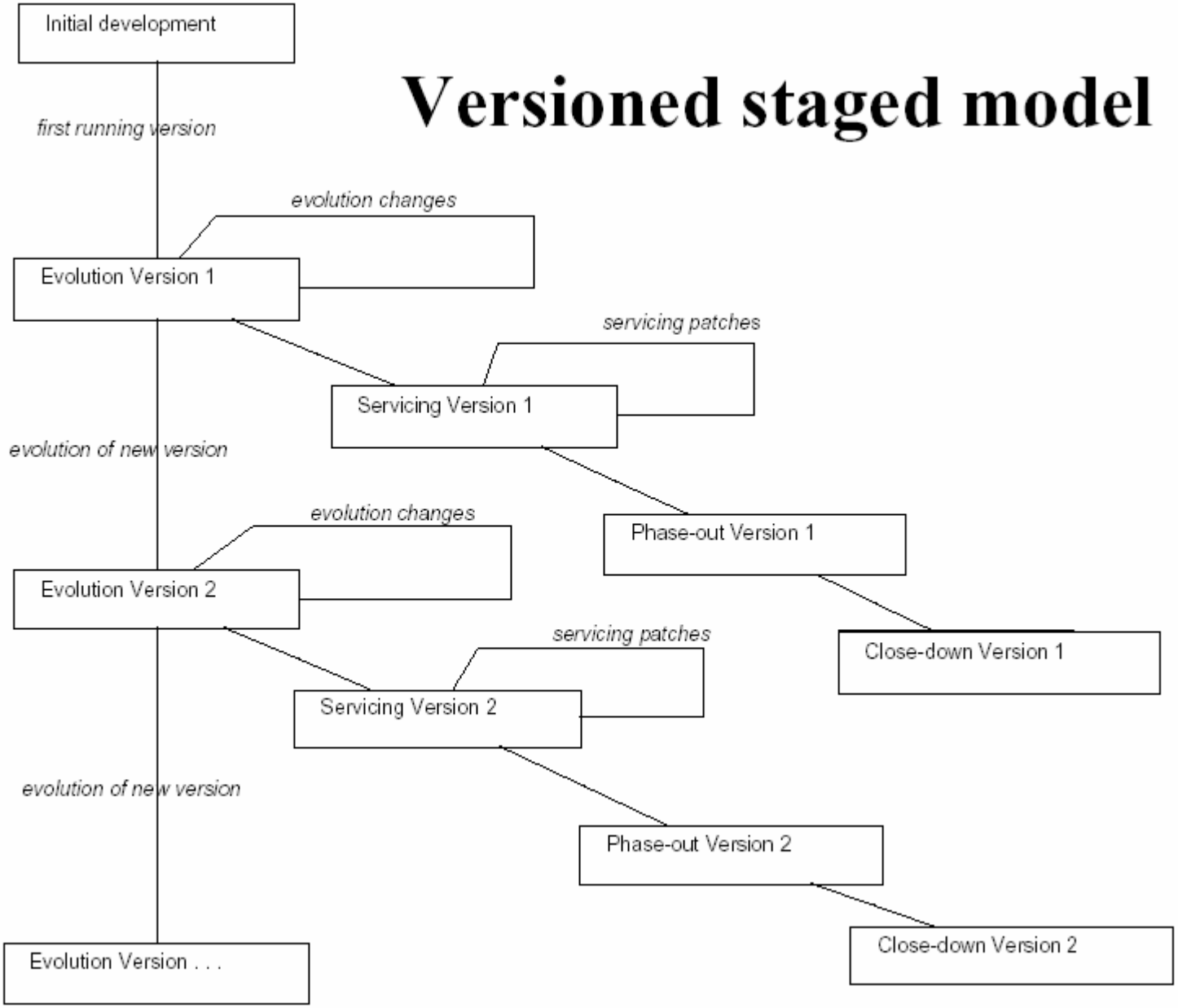
Staged Life Cycle Model

- 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“

Staged model of software lifecycle



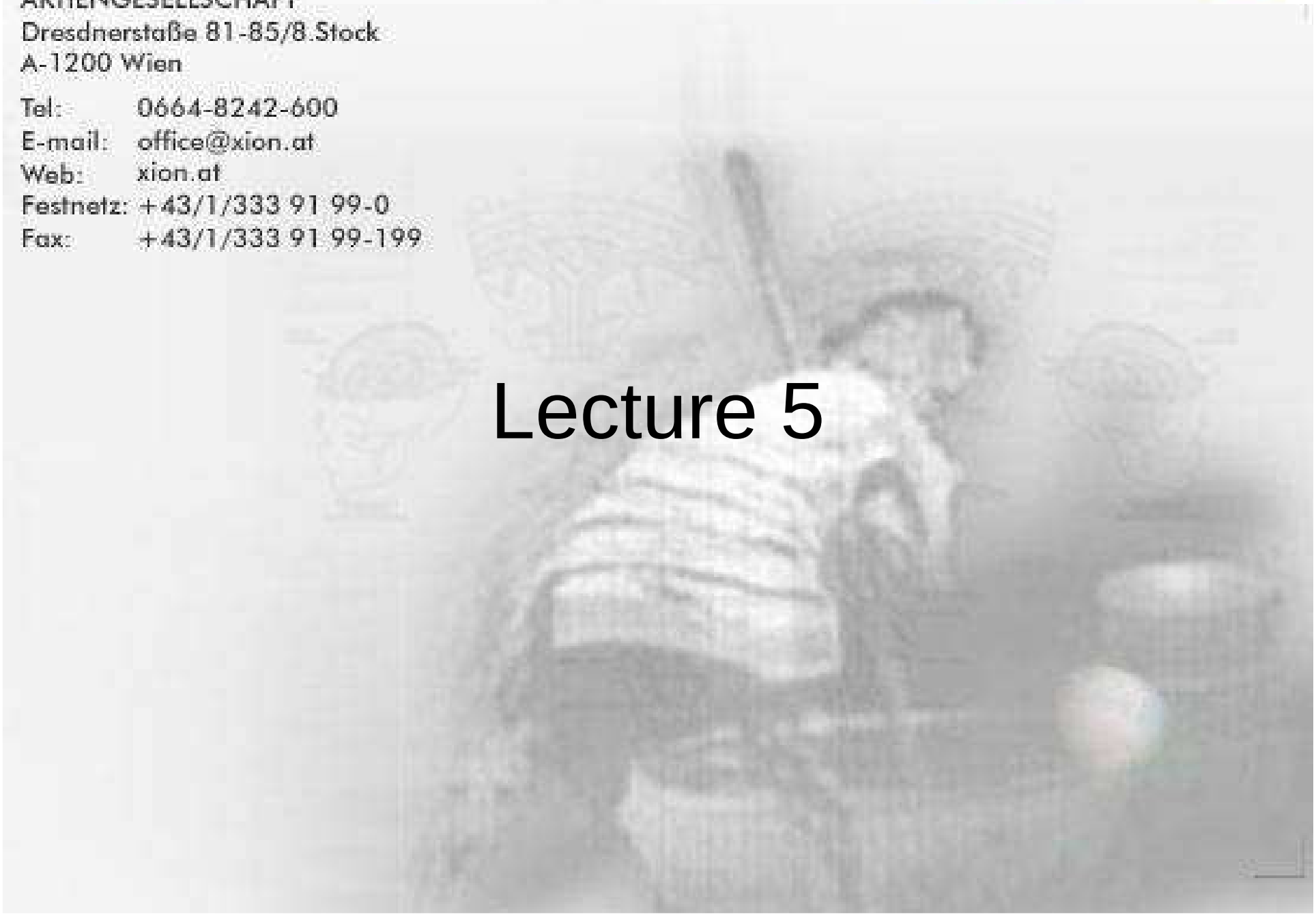
Versioned staged model



XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 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 og

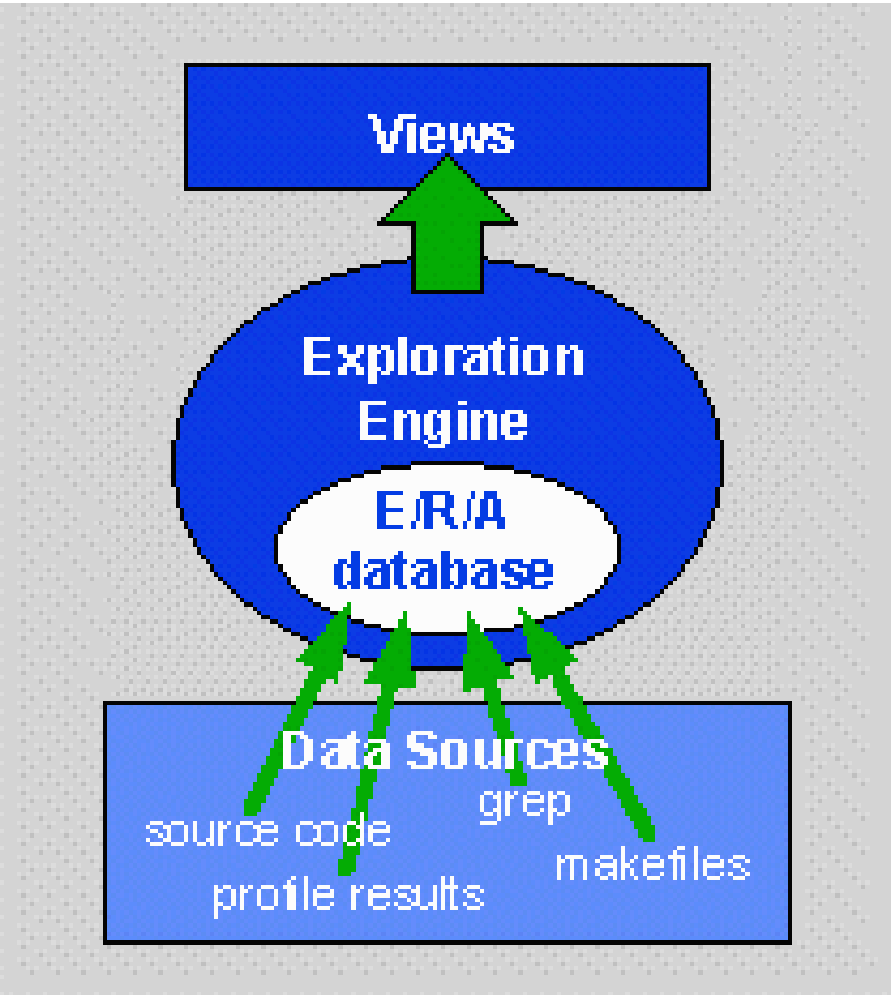
Lecture 5



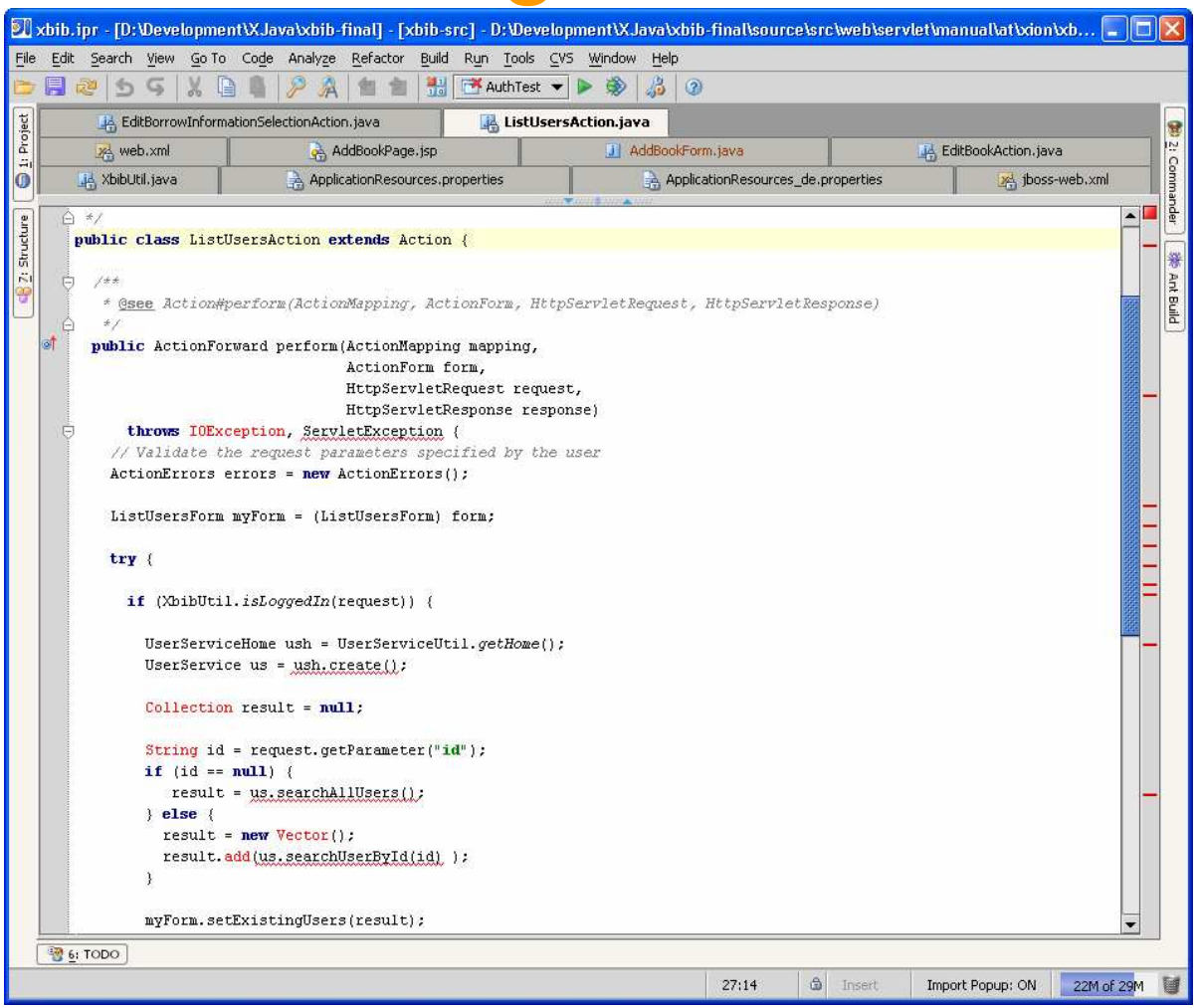
Lecture 5

- Inhalte
  - Tool Demo
    - Reverse Engineering mit Imagix4D
    - Refactoring mit IntelliJ IDEA
    - Architektur- und Evolutionsanalyse mit dem Sotograph
    - Eclipse mit dem CREOLE Plugin
  - Organisation der Wartung
    - Software Configuration Management
      - Defect Tracking und Change Tracking
      - Software Artifact Management
    - Produktivstellung

Reverse Engineering: Imagix 4D



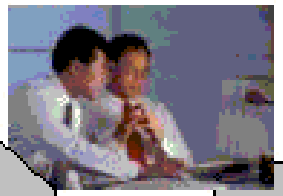
Refactoring: IntelliJ IDEA

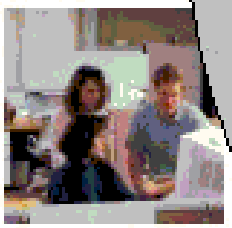


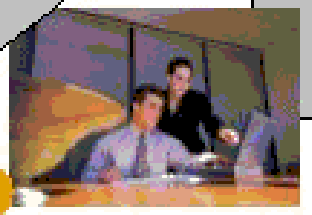
Sotograph

auch für sehr
große Projekte

10^2 Subsysteme
10^3 - 10^4 Klassen
10^6 - 10^7 LOC,
10 – 100 Entwickler







CREOLE

- Eclipse plugin by „The Chisel Group“
  - Computer Human Interaction & Software Engineering Lab
  - [http://www.thechiselgroup.org/creole]
- „Creole is the term used to describe our plug-in to the Eclipse platform which integrates SHriMP with the Eclipse platform's Java Development Tools (JDT).“
- SHriMP = **S**imple **H**ierarchical **M**ulti-**P**erspective

Software Configuration Management (SCM)

Software Configuration Management: Begriffsdefinition

- Software Artifact Management (SAM)
  - Management von versionierten Software Komponenten
- Software Configuration Management (SCM)
  - SAM in Verbindung mit Defect und Change Tracking

SAM Features

- Version Control und Release Management
- Workspace Management
- Build Management
- Process Configurability
- Parallel Development
- Distributed Development

SCM Features

- Defect Tracking
  - Management von Software Bugs
    - Erfassung, Life Cycle Management, Reporting
- Change Tracking
  - Management von Change Requests
    - Erfassung, Life Cycle Management, Reporting
- Essentiell in der Software Wartung!

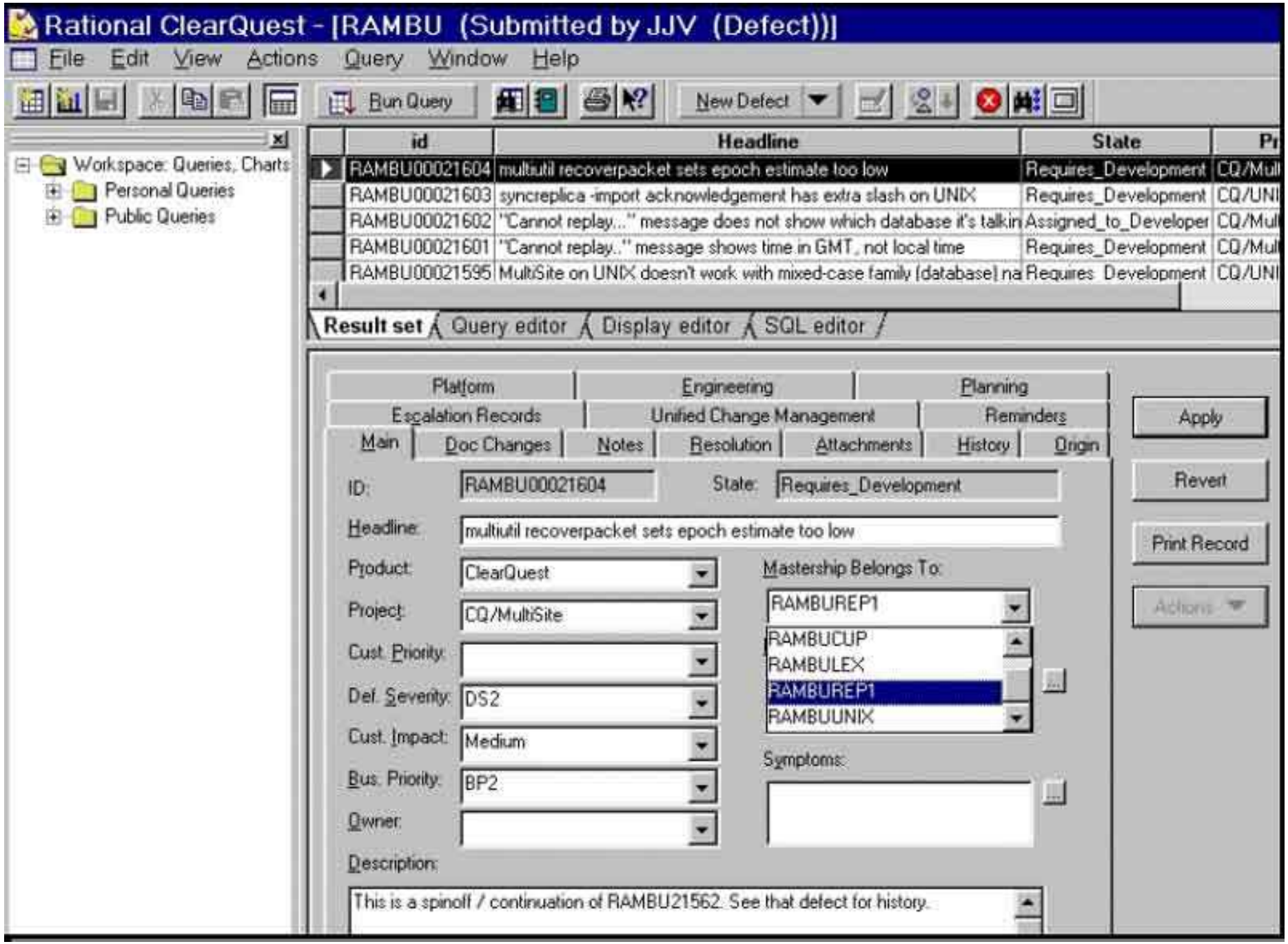
Defect Tracking und Change Tracking

Fehlermeldung/Änderungsantrag

- Erfassung über
  - Call Center
  - Bug Tracking Tool
  - Email
  - Telefon, Fax
  - ...
- Das professionelle Management von Fehlermeldungen wird als *Defect Tracking* bezeichnet
- Das professionelle Management von change requests wird als *Change Tracking* bezeichnet
- Defect Tracking und Change Tracking werden umfassend als *Change Management* bezeichnet

Spezifikation eines Defects

- Kurzbeschreibung
- Zeitstempel des Auftretens
- Submitter (Name, Organisation)
- Längere Beschreibung zum Nachvollziehen der Fehlersituation durch den Ingenieur
- Komponente, in der der Fehler aufgetreten ist (Server, Client, GUI, Connectivity, ...)
- Priorität


Rational ClearQuest - [RAMBU (Submitted by JJV (Defect))]
File Edit View Actions Query Window Help
Run Query
New Defect
Workspace: Queries, Charts
Personal Queries
Public Queries
id
Headline
State
RAMBU00021604 multiutil recoverpacket sets epoch estimate too low Requires_Development
RAMBU00021603 syncreplica -import acknowledgement has extra slash on UNIX Requires_Development
RAMBU00021602 "Cannot replay..." message does not show which database it's talkin Assigned_to_Developer
RAMBU00021601 "Cannot replay.." message shows time in GMT, not local time Requires_Development
RAMBU00021595 MultiSite on UNIX doesn't work with mixed-case family (database) na Requires_Development
Result set
Query editor
Display editor
SQL editor
Platform
Engineering
Planning
Escalation Records
Unified Change Management
Reminders
Main
Doc Changes
Notes
Resolution
Attachments
History
Origin
Apply
Revert
Print Record
Actions
ID:
RAMBU00021604
State:
Requires_Development
Headline:
multiutil recoverpacket sets epoch estimate too low
Product:
ClearQuest
Project:
CQ/MultiSite
Cust. Priority:
Def. Severity:
DS2
Cust. Impact:
Medium
Bus. Priority:
BP2
Owner:
Mastership Belongs To:
RAMBUREP1
RAMBUCUP
RAMBULEX
RAMBUREP1
RAMBUUNIX
Symptoms:
Description:
This is a spinoff / continuation of RAMBU21562. See that defect for history.

Life Cycle eines Defects

- Submission (z.B. per mail, Web, etc.)
- Assignment
  - Der Verantwortliche Manager trifft eine Zuteilung des Defects an einen Wartungsingenieur (Asignee)
- Eventuell Reassignment
  - Der Asignee weist den Defect begründet zurück
- Resolution
  - Der Defect wird mit einer Fehlerbehebungsbeschreibung als gefixt (oder resolved) klassifiziert
- Verificiation
  - Die Behebung wird als erfolgreich klassifiziert
- Closing

Eigenschaften von Defect Tracking Tools

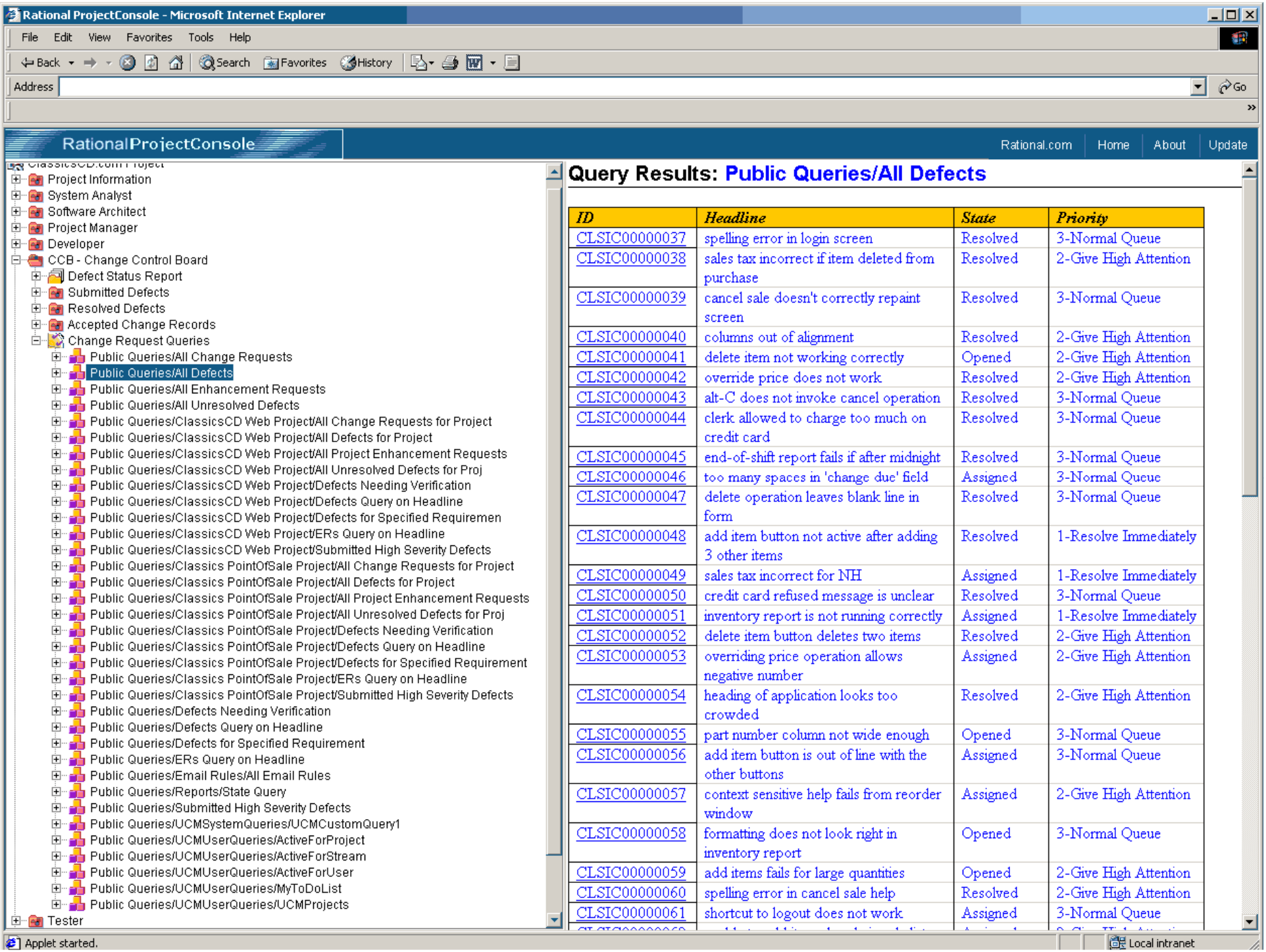
- Bilden den komplette Life Cycle von der Meldung bis zur Schließung ab
- Speichern Defect Reports in Datenbanken
- Liefern umfassende (auch grafische) Reports
  - Defects per Programmer (assigned, fixed)
  - Defects per Month
  - Defects per Module etc.
- Daraus können Meßgrößen für den Zustand der Software und die Effizienz der Organisation abgeleitet werden
- Trend: Defect Tracking Tools entwickeln sich zu Change Management Tools

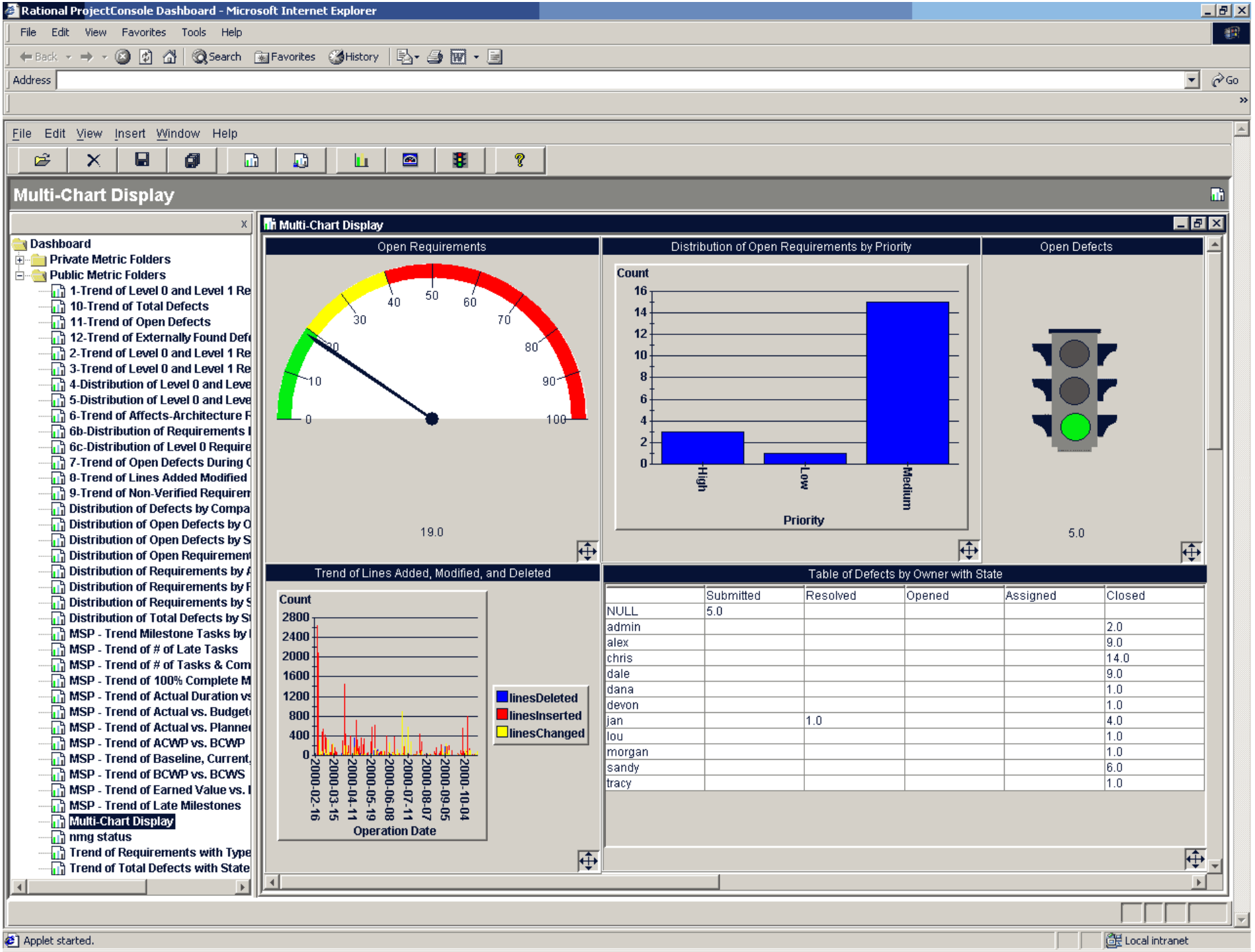
Probleme von Defect Tracking Tools

- Mangelnde Integration
  - in die Entwicklungsumgebung
    - Link von einer Fehlermeldung direkt in den Source Code des Moduls
    - Konfiguration des Debuggers nach der Fehlermeldung (Setzen von Breakpoints, Watches, etc.)
  - in Management Tools
  - in Tools, die andere Entwicklungsphasen betreffen
    - z.B. Aufnahme aller change requests in ein Anforderungsmanagement Tool

Evaluierung von Defects

- Wird meist toolunterstützt
  - Low end
    - z.B. Excel
  - High End
    - z.B. Rational Produkte
- Gewinnung von Kenndaten als Input für das (Projekt-)controlling


Rational ProjectConsole - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Go
RationalProjectConsole
Rational.com Home About Update
Project Information
System Analyst
Software Architect
Project Manager
Developer
CCB - Change Control Board
Defect Status Report
Submitted Defects
Resolved Defects
Accepted Change Records
Change Request Queries
Public Queries/All Change Requests
Public Queries/All Defects
Public Queries/All Enhancement Requests
Public Queries/All Unresolved Defects
Public Queries/ClassicsCD Web Project/All Change Requests for Project
Public Queries/ClassicsCD Web Project/All Defects for Project
Public Queries/ClassicsCD Web Project/All Project Enhancement Requests
Public Queries/ClassicsCD Web Project/All Unresolved Defects for Proj
Public Queries/ClassicsCD Web Project/Defects Needing Verification
Public Queries/ClassicsCD Web Project/Defects Query on Headline
Public Queries/ClassicsCD Web Project/Defects for Specified Requiremen
Public Queries/ClassicsCD Web Project/ERs Query on Headline
Public Queries/ClassicsCD Web Project/Submitted High Severity Defects
Public Queries/Classics PointOfSale Project/All Change Requests for Project
Public Queries/Classics PointOfSale Project/All Defects for Project
Public Queries/Classics PointOfSale Project/All Project Enhancement Requests
Public Queries/Classics PointOfSale Project/All Unresolved Defects for Proj
Public Queries/Classics PointOfSale Project/Defects Needing Verification
Public Queries/Classics PointOfSale Project/Defects Query on Headline
Public Queries/Classics PointOfSale Project/Defects for Specified Requirement
Public Queries/Classics PointOfSale Project/ERs Query on Headline
Public Queries/Classics PointOfSale Project/Submitted High Severity Defects
Public Queries/Defects Needing Verification
Public Queries/Defects Query on Headline
Public Queries/Defects for Specified Requirement
Public Queries/ERs Query on Headline
Public Queries/Email Rules/All Email Rules
Public Queries/Reports/State Query
Public Queries/Submitted High Severity Defects
Public Queries/UCMSystemQueries/UCMCustomQuery1
Public Queries/UCMUserQueries/ActiveForProject
Public Queries/UCMUserQueries/ActiveForStream
Public Queries/UCMUserQueries/ActiveForUser
Public Queries/UCMUserQueries/MyToDoList
Public Queries/UCMUserQueries/UCMProjects
Tester
Query Results: Public Queries/All Defects
ID Headline State Priority
CLSIC00000037 spelling error in login screen Resolved 3-Normal Queue
CLSIC00000038 sales tax incorrect if item deleted from purchase Resolved 2-Give High Attention
CLSIC00000039 cancel sale doesn't correctly repaint screen Resolved 3-Normal Queue
CLSIC00000040 columns out of alignment Resolved 2-Give High Attention
CLSIC00000041 delete item not working correctly Opened 2-Give High Attention
CLSIC00000042 override price does not work Resolved 2-Give High Attention
CLSIC00000043 alt-C does not invoke cancel operation Resolved 3-Normal Queue
CLSIC00000044 clerk allowed to charge too much on credit card Resolved 3-Normal Queue
CLSIC00000045 end-of-shift report fails if after midnight Resolved 3-Normal Queue
CLSIC00000046 too many spaces in 'change due' field Assigned 3-Normal Queue
CLSIC00000047 delete operation leaves blank line in form Resolved 3-Normal Queue
CLSIC00000048 add item button not active after adding 3 other items Resolved 1-Resolve Immediately
CLSIC00000049 sales tax incorrect for NH Assigned 1-Resolve Immediately
CLSIC00000050 credit card refused message is unclear Resolved 3-Normal Queue
CLSIC00000051 inventory report is not running correctly Assigned 1-Resolve Immediately
CLSIC00000052 delete item button deletes two items Resolved 2-Give High Attention
CLSIC00000053 overriding price operation allows negative number Assigned 2-Give High Attention
CLSIC00000054 heading of application looks too crowded Resolved 2-Give High Attention
CLSIC00000055 part number column not wide enough Opened 3-Normal Queue
CLSIC00000056 add item button is out of line with the other buttons Assigned 3-Normal Queue
CLSIC00000057 context sensitive help fails from reorder window Assigned 2-Give High Attention
CLSIC00000058 formatting does not look right in inventory report Opened 3-Normal Queue
CLSIC00000059 add items fails for large quantities Opened 2-Give High Attention
CLSIC00000060 spelling error in cancel sale help Resolved 2-Give High Attention
CLSIC00000061 shortcut to logout does not work Assigned 3-Normal Queue
Applet started.
Local intranet


Rational ProjectConsole Dashboard - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address
Go
File Edit View Insert Window Help
Multi-Chart Display
Dashboard
Private Metric Folders
Public Metric Folders
1-Trend of Level 0 and Level 1 Re
10-Trend of Total Defects
11-Trend of Open Defects
12-Trend of Externally Found Defe
2-Trend of Level 0 and Level 1 Re
3-Trend of Level 0 and Level 1 Re
4-Distribution of Level 0 and Leve
5-Distribution of Level 0 and Leve
6-Trend of Affects-Architecture F
6b-Distribution of Requirements I
6c-Distribution of Level 0 Require
7-Trend of Open Defects During C
8-Trend of Lines Added Modified
9-Trend of Non-Verified Requirem
Distribution of Defects by Compa
Distribution of Open Defects by O
Distribution of Open Defects by S
Distribution of Open Requirement
Distribution of Requirements by A
Distribution of Requirements by F
Distribution of Requirements by S
Distribution of Total Defects by St
MSP - Trend Milestone Tasks by
MSP - Trend of # of Late Tasks
MSP - Trend of # of Tasks & Com
MSP - Trend of 100% Complete M
MSP - Trend of Actual Duration vs
MSP - Trend of Actual vs. Budget
MSP - Trend of Actual vs. Planne
MSP - Trend of ACWP vs. BCWP
MSP - Trend of Baseline, Current,
MSP - Trend of BCWP vs. BCWS
MSP - Trend of Earned Value vs. I
MSP - Trend of Late Milestones
Multi-Chart Display
nmg status
Trend of Requirements with Type
Trend of Total Defects with State
Multi-Chart Display
Open Requirements
0
10
20
30
40
50
60
70
80
90
100
19.0
Distribution of Open Requirements by Priority
Count
16
14
12
10
8
6
4
2
0
High
Low
Medium
Priority
Open Defects
5.0
Trend of Lines Added, Modified, and Deleted
Count
2800
2400
2000
1600
1200
800
400
0
2000-02-16
2000-03-15
2000-04-11
2000-05-19
2000-06-08
2000-07-11
2000-08-07
2000-09-05
2000-10-04
Operation Date
linesDeleted
linesInserted
linesChanged
Table of Defects by Owner with State
Submitted Resolved Opened Assigned Closed
NULL 5.0
admin 2.0
alex 9.0
chris 14.0
dale 9.0
dana 1.0
devon 1.0
jan 1.0 4.0
lou 1.0
morgan 1.0
sandy 6.0
tracy 1.0
Applet started.
Local intranet

Defect Tracking als statistisches Orakel

- Durch statistische Auswertungen der Defect Daten können „malicious modules“ entdeckt werden
- Das sind Module, die besonders oft und/oder von vielen verschiedenen Personen geändert werden
- Deutet hin auf mangelhaftes Design, zu komplexe Programmierung, bis hin zu organisatorischen Schwächen in den Geschäftsprozessen der Organisation

Software Artifact Management – Warum?

- Nachvollziehbarkeit / Traceability
  - Wer hat wann was verändert? (Historie, Versionsvergleich)
  - Warum? (Version Comments)
- Parallele Entwicklung
  - Mehrere Entwickler arbeiten auf demselben File auf verschiedenen *private branches (aka streams)*
  - Mehrere Versionen eines Files werden zu einem gemeinsamen Nachfolger *gemerged*
- Möglichkeit der Reproduktion aller ausgelieferten Versionen
  - verschiedene Kunden
  - verschiedene Zielplattformen
  - verschiedene Feature-Zusammenstellung

Versionsmanagement

- Versionen entstehen durch
  - Bug fixes
  - Anpassung an neue Hardware, Betriebssysteme, Libraries
  - Erweiterungen
  - ...
- Eben durch Einflüsse von Wartung und Evolution

Versionsmanagement

- Typische Funktionen eines SCM Tools
  - Check in / check out
  - Labeling
  - Textual Difference
  - Branching
  - Merging
  - Views
  - Search
  - Reporting
  - History

Versionsmanagement

- Unter Versionsverwaltung stehen
  - Source Code
  - Script files (batch files / shell scripts)
  - html, xml files
  - Javascript files
  - Makefiles, Projektfiles
  - Libraries
  - System Drivers
  - Dokumentation (!)

Versionsmanagement

- Was wird **nicht** versionsverwaltet
  - Alle generierten Artefakte
    - Ausführbare Programme (exe, class files, binaries)
    - Generierte Dokumentation (z.B. javadoc)
    - Temporäre Files

Versionsmanagement Tipps

- SCM sollte vor Beginn eines Projektes aufgesetzt und stabil sein
- Es sollten keine ableitbaren Artefakte eingecheckt werden (Executables, JavaDoc)
- Jede Version, die an den Kunden geht, muss vollständig gelabelt sein (Reproduktion)!
- SCM Repository muss ins Backup!

Build Management

- „Build“
  - Aus den versionierten Komponenten wird eine Produktionsversion erzeugt
  - Probleme
    - Welche Komponenten müssen aufgrund von Abhängigkeiten neu erzeugt werden?
    - Welche Komponenten können wieder verwendet werden?
    - Welche Komponenten in welchen Versionen fanden in welchen Builds Verwendung?
      - „Stückliste“

Release Management

- Ein „Release“ ist ein auslieferbares Produkt definierten Umfanges
  - Sollte jederzeit reproduziert werden können
    - Was befand sich in dieser Release?
  - Dokumentiert durch Release Notes
    - Kann durch Tools (semi-)automatisch geschehen
  - Einpflegen der Release in Dokumentation
    - Release für wen? Wann ausgeliefert? Durch wen? Wann abgenommen? Warum ausgeliefert? Welche Release wurde ersetzt?

Tools

- SCCS (Source Code Control System)
- RCS (Revision Control System)
- CVS (Concurrent Versions System)
- Microsoft Visual Source Safe (VSS)
- Merant PVCS
- Rational ClearCase
- uvm.

Produktivstellung

Wartungsfenster

- Ein Wartungsfenster ist ein begrenztes Zeitintervall, in dem ein Produktionssystem für Wartungsarbeiten außer Betrieb geht
  - Das System muss definiert außer Betrieb genommen werden (Ankündigung und Wartungsmeldung)
  - Die Wartungsarbeiten müssen genau geplant werden (Projektmanagement)
  - Definition des „Point of no return“
    - Ab diesem Zeitpunkt kann nicht mehr auf das alte System zurückgestellt werden (außer durch Einspielen eines Backups)
  - Definition des Wiederanlaufes

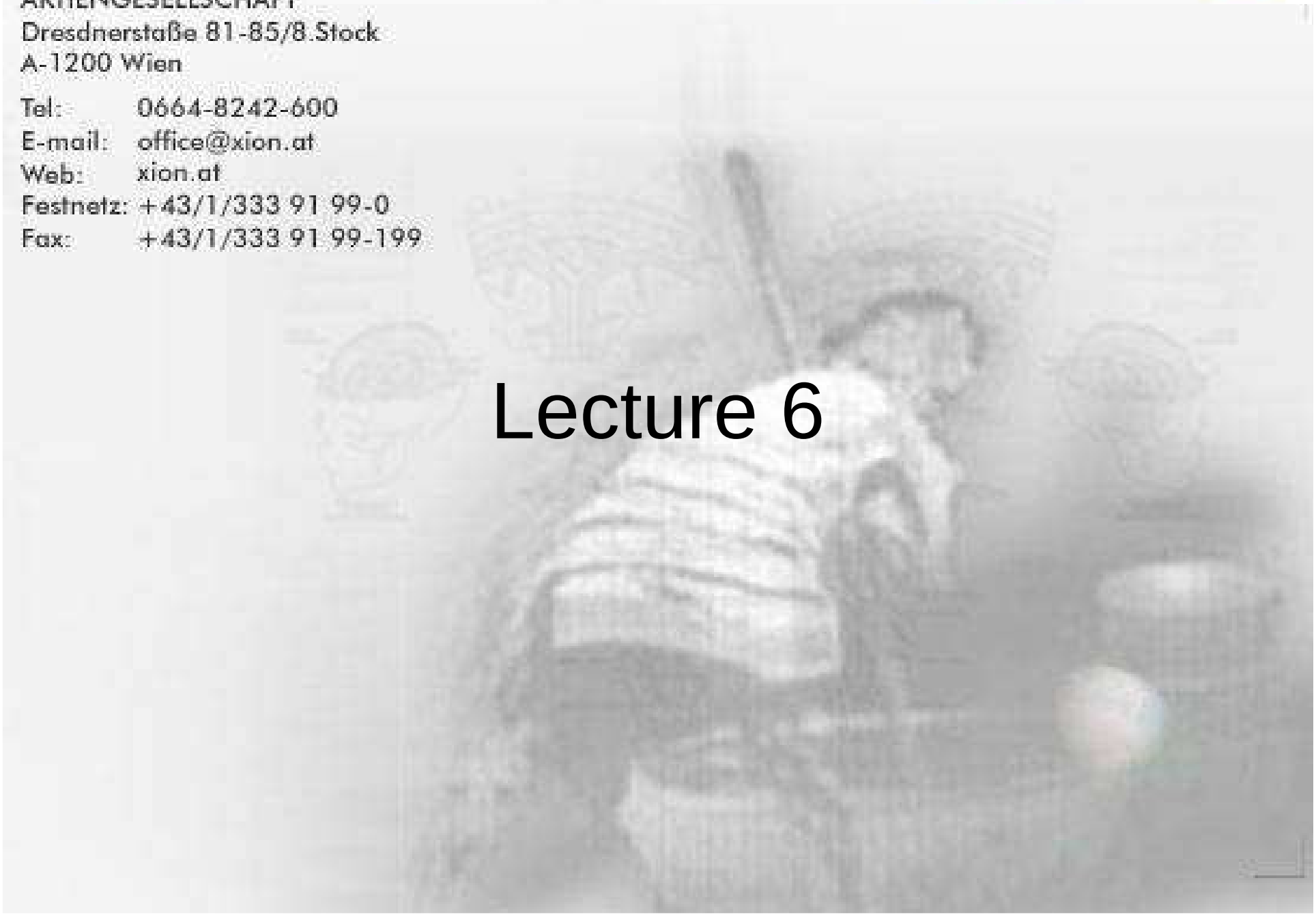
Checkliste Wartungsfenster

- Voraussetzung: Freigabe der neuen Produktions-Release
- Projektplan erstellen
  - Ressourcen/Rollen festlegen
  - Detaillierter Ablaufplan (Work Breakdown Structure), Definition von Go/No-Go bzw. Rollback Punkten
  - Dauer definieren
- Projektplan durch Tests verifizieren
- Termin mit dem Systemverantwortlichen bzw. den Benutzern (intern, eventuell extern) vereinbaren
- Verteilen des Projektplans an die Akteure (Betriebsführungspersonal, Wartungspersonal, etc.)

XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 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 og

Lecture 6



Lecture 6

- Inhalt
  - Software Evolution
    - Definition
    - Types of Programs
    - Laws of Software Evolution
    - Change Patterns and Evolutionary Narratives

Fallbeispiel: „Adaption auf der falschen Abstraktionsstufe“

- Gegeben: Datenbankmodell
  - Datenbankfeld: „SA_EINZEL“
    - Vermutete Semantik
      - SA ... Sportart
      - EINZEL … Einzelsportart
  - Typ: char(1), keine Constraints
  - Wertbelegung in der Datenbank
    - „J“ ... Folgerung: Ja, Sportart ist eine Einzelsportart
    - „N“ ... Folgerung: Nein, Sportart ist keine Einzelsportart
    - **„T“ ... Überraschung!: Sportart ist eine „Tennissportart“** ☹

Software Evolution

Evolution: General Definition 1/2

- Evolution is the process of *progressive change* over time in characteristics, attributes, properties of some material or abstract, natural or artificial, entity or system or of a sequence of these
  - Changes are progressive when they result in a definable trend of, for example, increasing value, growing precision or better fit to a changing domain or context

Evolution: General Definition 2/2

- Entities include objects or collections of objects (e.g. population) such as natural species, societies, cities, artefacts, concepts, theories, ideas or systems of these
- Change process will, in general, be continual with relatively slow rate of change, or discrete with individual incremental changes, small relative to entity as a whole

- Source: Lehman and Ramil 2001

Software Evolution: Definition 1/2

- Keine genormte Definition
- Nach Lehman/Ramil
  - Software Evolution is the process of continual fixing, adaption, enhancement to maintain stakeholder satisfaction
    - In response to changes in domains, needs, expectations
- Nach Bennet/Rajlich
  - Maintenance means general post-delivery activities
  - Evolution refers to a particular phase in the staged model where substantial changes are made to the software

Software Evolution: Definition 2/2

- Nach Godfrey
  - Evolution is what happens while you are busy making other plans
    - Maintenance is the *planned* set of tasks to effect changes
    - Evolution is what actually happens to software

Types of Programs

- *Nach Lehman, Belady 1980, pp. 1060-1076*
- S-type Programs („Specifiable“)
  - Problem can be stated formally and completely
  - Acceptance: Is the program correct according to its specification?
  - This software does not evolve
    - A change to the specification defines a new problem, hence a new program

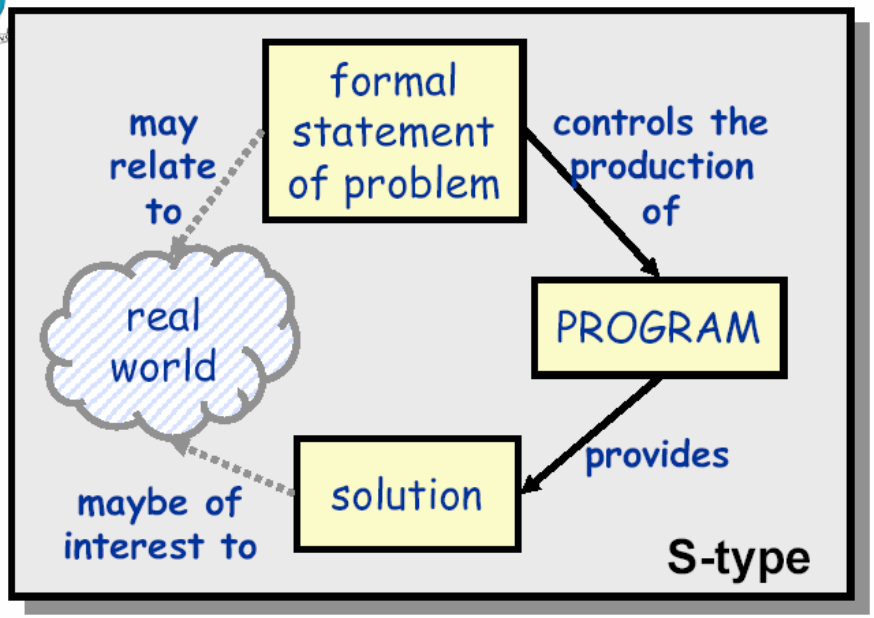
Types of Programs

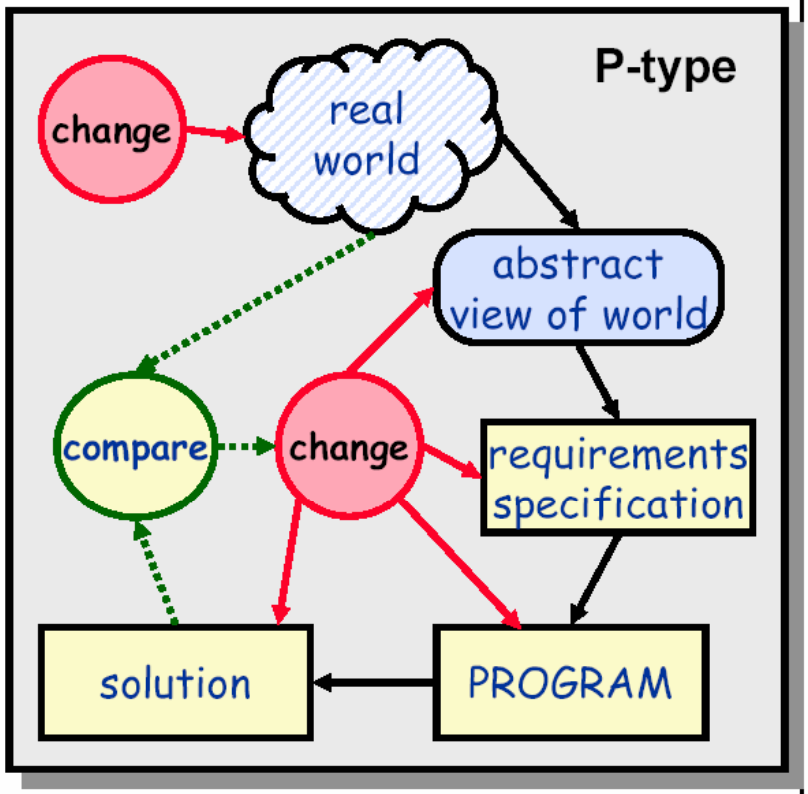
- P-type Programs („Problem-solving“)
  - Imprecise statement of a real-world problem
  - Acceptance: Is the program an acceptable solution to the problem?
  - This software is likely to evolve continuously
    - Because solution is never perfect, and can be improved
    - Because the real-world changes and hence the problem changes

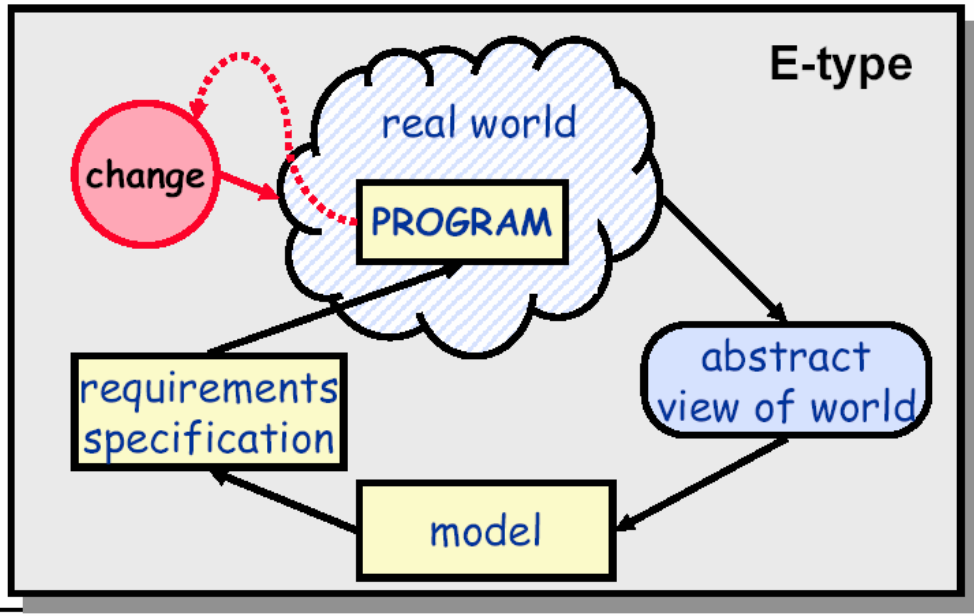
Types of Programs

- E-type Programs („Embedded“)
  - A system that becomes part of the world it models
  - Acceptance: Depends entirely on opinion and judgement; criterion is the satisfaction of stakeholder needs
  - This software is *inherently* evolutionary
    - Changes in the software and the world affect each other

Source: *Adapted from Lehman 1980, pp1061-1063*

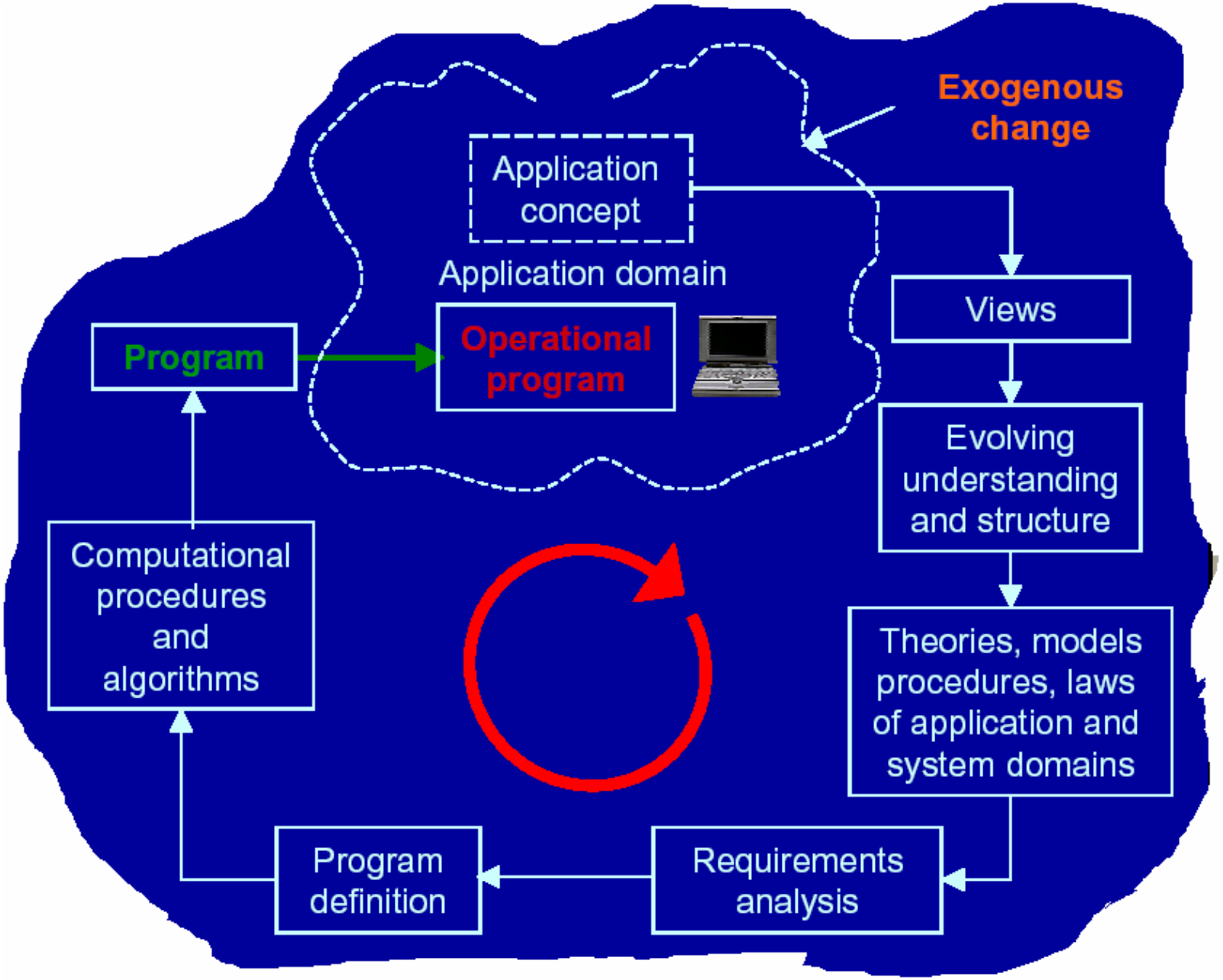




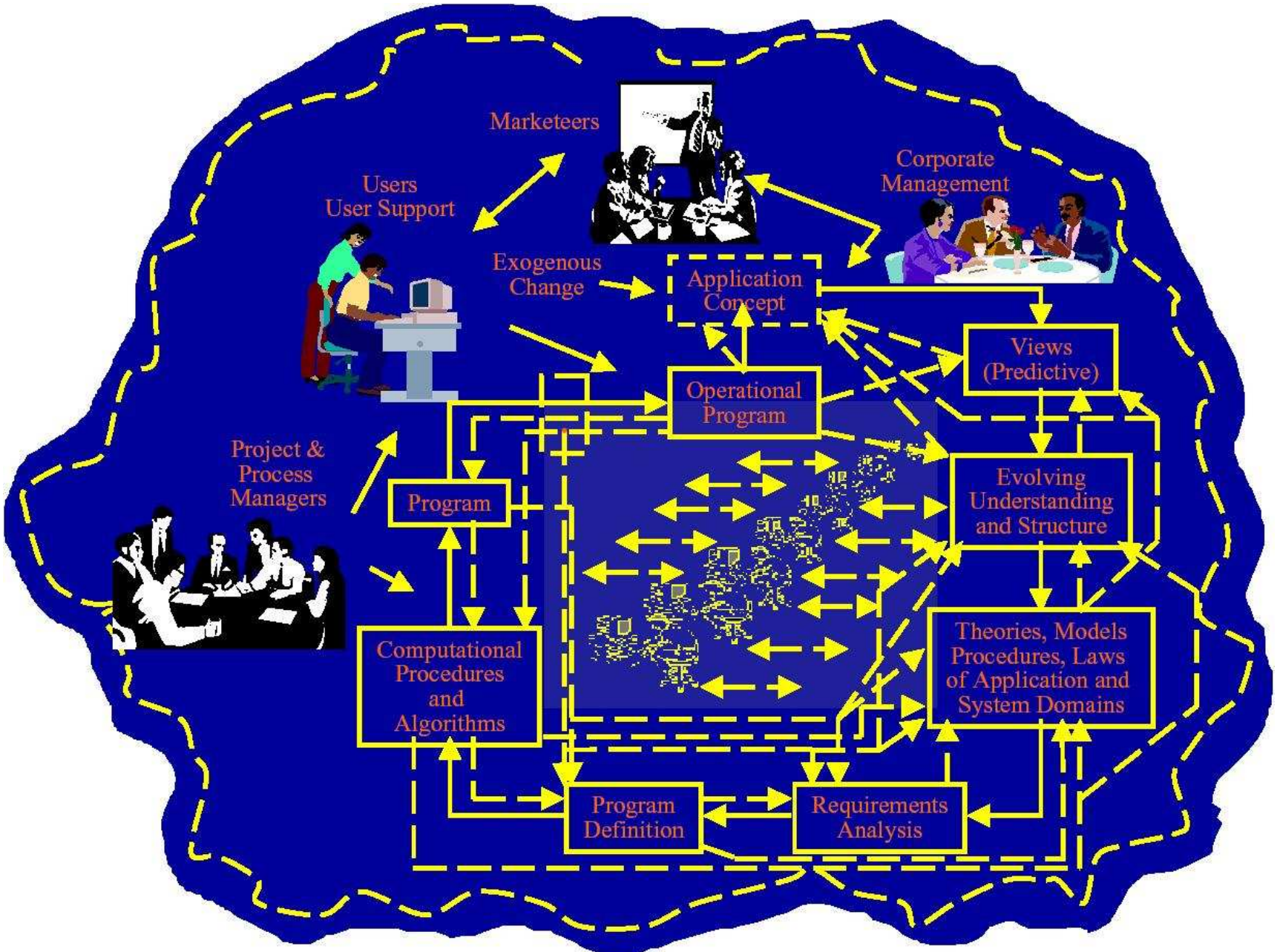


Software Systeme als komplexe Feedback Systeme

- Der Entwicklungs- und Evolutionsprozess eines Software Systems wird von Lehman als
  - Multi-level
  - Multi-loop
  - Multi-agent
  - Feedback System bezeichnet.
- Feedback technisch: Die Rückführung eines (transformierten) Teils des Ausgangssignals als Eingangssignal in ein System
  - („Feedback: The return of a portion of the **output**, or processed portion of the output, of a (usually active) device to the **input**“)



[Lehman, Ramil]

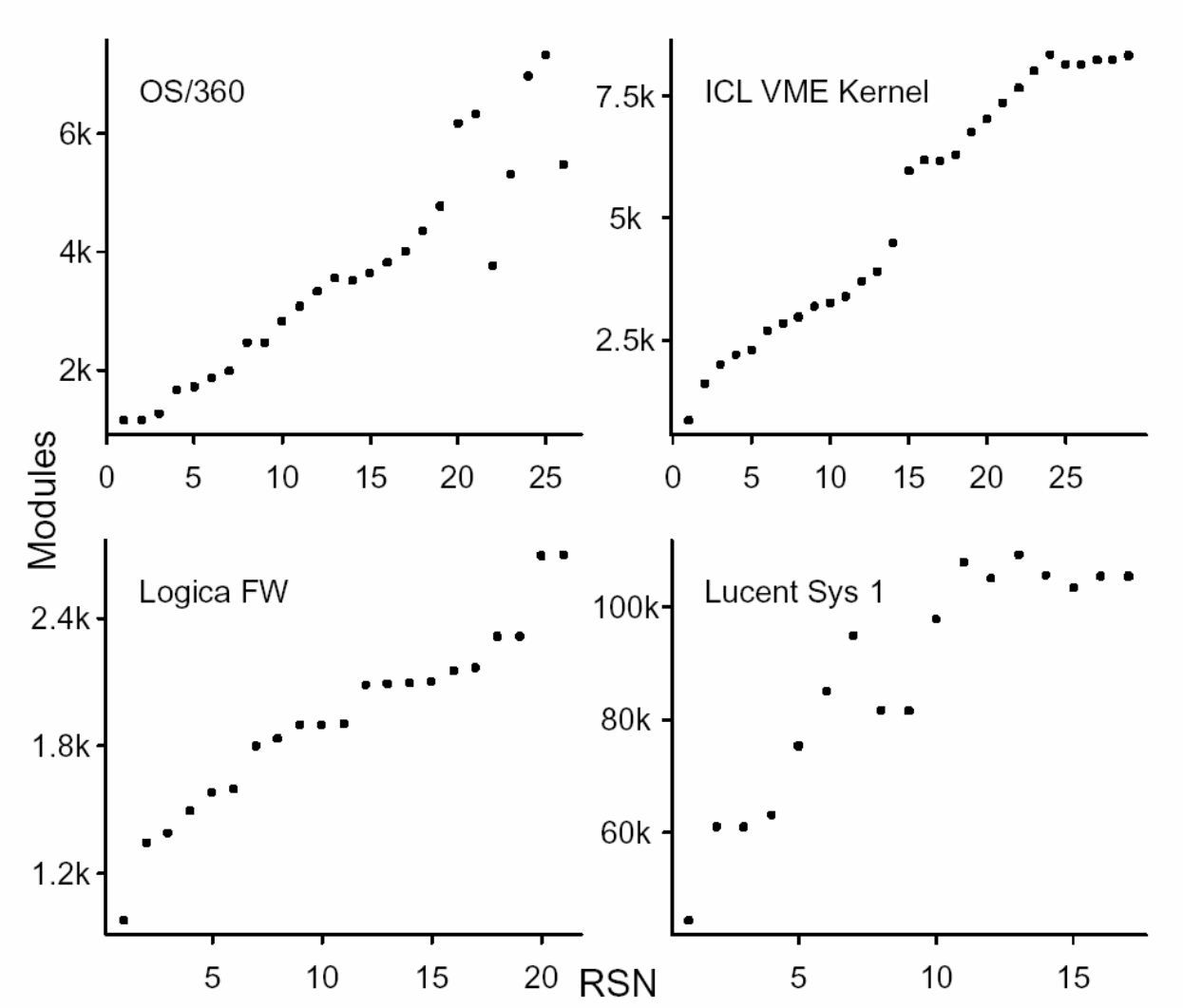


[Lehman, Ramil]

Laws of Software Evolution

- In den späten sechziger Jahren untersuchen *Lehman und Belady* die *Release History Daten* von IBM OS/360 mittels bestimmter Metriken und stoßen auf Eigenschaften im Evolutionsprozess, die bei anderen Systemen in späteren Untersuchungen ebenfalls nachvollzogen werden können
- Diese Eigenschaften scheinen Gesetzmäßigkeiten zu folgen und wurden als „Laws of Software Evolution“ postuliert
- Die „Laws of Software Evolution“ ergeben sich aus der Beobachtung von *E-type programs*

Laws of Software Evolution



Laws of Software Evolution

- Warum „Gesetze“?
  - Die entdeckten Phänomene der Evolution werden als Gesetze bezeichnet, da sie technologie- und prozessunabhängige Mechanismen bezeichnen

Laws of Software Evolution

- *Nach Lehman, Belady 1980, pp. 1061-1063 und spätere Publikationen*
- 1) Law of continuing change
  - "A system that reflects some external reality undergoes continuing change or becomes progressively less useful
    - The change process continues until it becomes more economical to replace it by a new or restructured system."
- 2) Law of increasing entropy (or: complexity)
  - "The entropy of a system increases with time unless specific work is executed to maintain or reduce it."

Laws of Software Evolution

- 3) Fundamental law of software evolution
  - Software evolution is self-regulating with statistically determinable trends and invariants
- 4) Conservation of organisational stability (invariant work rate)
  - During the active live of a software system the *average effective global activity rate* is roughly constant

Laws of Software Evolution

- 5) Conservation of familiarity
  - In general, the average incremental growth rate (growth rate trend) tends to decline
    - As an E-type system evolves all associated with it, developers, sales personnel, users, for example, must maintain mastery of its content and behaviour to achieve satisfactory evolution. Excessive growth diminishes that mastery.
- 6) Continuing growth
  - The functional content of E-type systems must be continually increased to maintain user satisfaction

Laws of Software Evolution

- 7) Declining quality
  - The quality of E-type systems will appear to be declining unless they are rigorously maintained and adapted to operational environment changes
- 8) Feedback System
  - E-type evolution processes constitute multi-level, multi-loop, multi-agent feedback systems and must be treated as such to achieve significant improvement over any reasonable base

Lehmans Approach: Formal

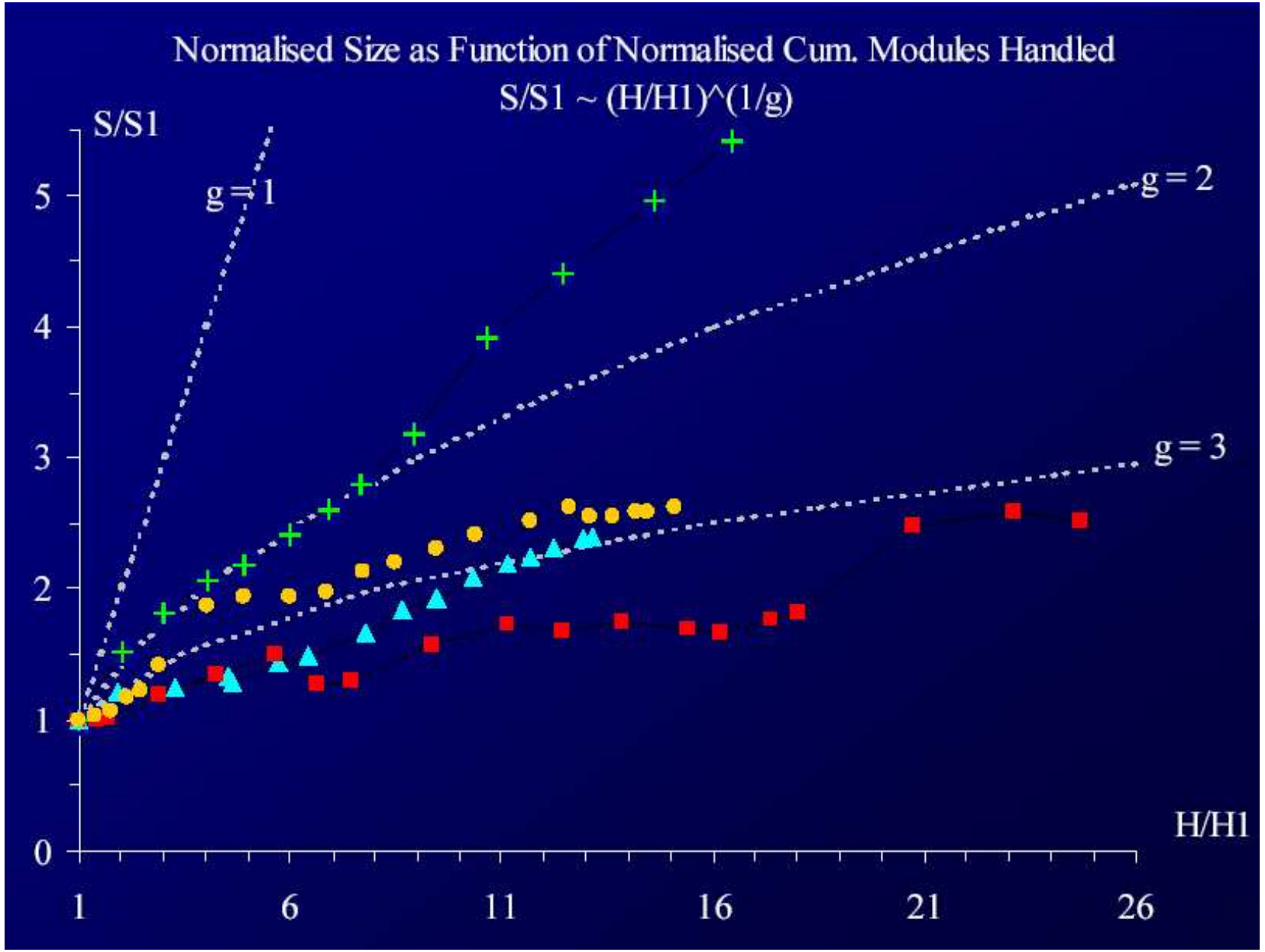
- Lehman describes software evolution on a formal level
  - Based on observations
    - Empirical generalisations are made
    - They provide basis for axioms in a formal theory
  - Possible inferences are proposed
    - Derived from the formal models
    - Basis for potential theorems in formal theory
  - Try to fully prove theorems

Formal Models of Software Evolution: Growth

- Inverse Square Model *[Turski 1996]*
  - $\underline{S_1} = S_1$
  - $\underline{S_i} = \underline{S_{i-1}} + e / (\underline{S_{i-1}})^2$
    - *S ... Size (often number of modules)*
    - *i ... Release sequence number (1..n, n = max release nr.)*
    - *e ... Model parameter*
    - *S_i and $\underline{S_i}$ stand for actual and predicted size at release i*
- *Other model: Normalised size as a function of the normalised work rate [Lehman 2001]*
  - $\underline{S_i} / S_1 = (H_i / H_1)^{1/g'}$ *for* $i >= 1$
    - *H ... Work rate as indirect effort indicator (e.g. elements handled)*
    - *g´ ... Model parameter*

Formal Models: An Example

- Next slide shows the *normalised size as a function of the normalised work rate*
  - size measured in number of modules
  - work rate measured in modules handled
  - For four industrially evolved systems
  - Three different organisations
  - Three different applications domains
- Data taken from release data history


Normalised Size as Function of Normalised Cum. Modules Handled
S/S1 ~ (H/H1)^(1/g)
S/S1
g = 1
g = 2
g = 3
H/H1
0
1
2
3
4
5
1
6
11
16
21
26

Formal Models: Use?

- Formal Models provide means for
  - Evolution planning
    - Simulation, visualisation, release planning
  - Process Management and Control
  - Long term prognosis
  - Overall process improvement
  - Tools

Research Areas in Software Evolution

- The driving force guiding the work will be the search for formally supported techniques:
  - logic-based declarative description and reasoning techniques
  - formal models for software evolution based on rewriting systems
  - software metrics
  - visualisation techniques
  - generation of design documents and source code
  - extraction of design and analysis documentation
  - migration to component-based and web-based systems
  - the use of metamodels as a general integration technique

Change Patterns and Evolutionary Narratives

Change patterns and evolutionary narratives

- Cathedral style [Raymond]
  - careful control and management
  - debugging done before committing code
  - evolution is slow, planned, rarely undone

- Bazaar style
  - lots of low-level changes, frequent fixes
  - lots of “building around” rather than wholesale changing, occasional redesigns
  - creeping feature-itis, “complete” dependency graph

[Godfrey 2001]

Change patterns and evolutionary narratives

- Band-aid evolution (just add a layer)
  - quick & dirty way to add new functionality, esp. if system is not well understood
    e.g. Y2K fixing, adding portability, new features
- "Vestigial features"
  - design artifact persists after rationale dies
    e.g. whale fin bone structure resembles hand

[Godfrey 2001]

Change patterns and evolutionary narratives

- "Adaptive radiation" [Lehman]
  - when conditions permit, encourage wild variation for a while
  - later, evaluate and let "best" ideas live on.
    e.g. Linux kernel evolution

- "Convergent evolution"
  - compare similar systems to reference architecture (or to each other)
    e.g. everyone grows an XML generator in response to market pressure

[Godfrey 2001]

Change patterns and evolutionary narratives

- Radical redesigns (localized and global)
  - aka “refactoring”
  - little new functionality added, but structure changes significantly, legacy cruft dissipates
  - likely “goodness” (design metrics) improves
- Migration patterns
  - look out for known translation idioms, especially if migration is not one big bang

    e.g. procedural-to-OO idioms

[Godfrey 2001]

Change patterns and evolutionary narratives

- Reuse patterns
  - components are (re)used in different systems

    e.g. build COTS interface, throw out homebrew DB

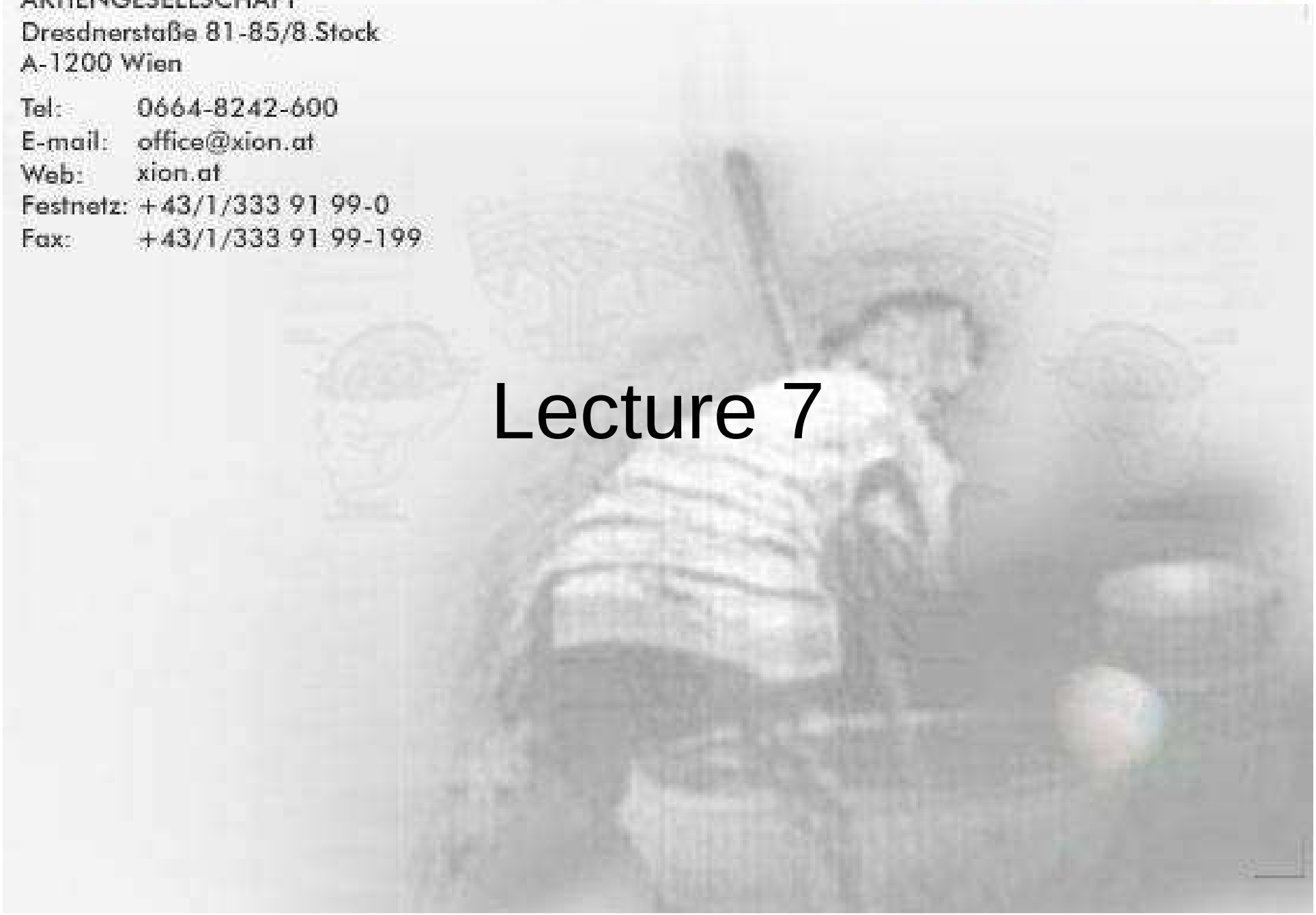
[Godfrey 2001]

Computer science...

- Computer Science is the discipline that believes all problems can be solved by adding one more layer of indirection.
  - *Dennis DeBruler*

XION IT SYSTEMS
AKTIENGESELLSCHAFT
Dresdnerstaße 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 og



Lecture 7

Lecture 7

- Inhalt
  - Spezielle Kapitel der Software Wartung
    - Program Comprehension
    - Change Impact Analysis
    - Change Propagation
    - Wartungsdokumentation
  - Maintainability (Wartbarkeit)
    - Definition
    - Ensuring Maintainability
      - Design for Change: Ausgewählte Kapitel
        - » Cohesion, Coupling, Model Driven Architecture (MDA)
  - Software Wartung im unternehmerischen Kontext

Spezielle Kapitel der Software Wartung

Aktivitäten im Wartungsfall

- Fehlermeldung bzw. Änderungsantrag
  - Dokumentieren/Einpflegen
  - Life Cycle Management
  - Evaluierung/Reporting
- **Analyse bzw. Planung der Änderung**
  - **Program Comprehension**
  - **Change Impact Analysis**
- **Implementierung der Änderung**
  - Restructuring
  - **Change Propagation**
  - Verwalten der Artefakte
- **Verifikation und Validierung**
- **Re-Dokumentation**
- Produktivstellung der Änderung

Spezielle Kapitel der Software Wartung

- Program Comprehension
- Change Impact Analysis
- Change Propagation
- Wartung und Test
- Wartungsdokumentation

Program Comprehension

Program Comprehension

- Definition
  - **Program Comprehension**
    - Research into how software engineers understand existing systems
  - by Malcolm Munro

Program Comprehension

- [R. Brooks 1983]
  - Programmer creates assumptions or hypotheses based on both acquired or existing knowledge to arrive at an understanding
  - Hypotheses are checked against the source code to prove their validity
  - *Beacons* are places in the source code that prove or falsify a hypotheses

Program Comprehension - Approaches

- Top-down approach
  - Taken by the original developer
    - The software engineer takes a top-down approach starting with the most abstract concepts, refines them, and transforms them into a computer program
- Bottom-up approach
  - Taken by the maintenance engineer
    - Programmers learn by focusing on small pieces of code and later combine this information together

Span of Understanding

- Span of Understanding
  - Nennt man die Zeitspanne, die der Programmierer zum Verstehen eines definierten Programmstückes benötigt
- Ziel der Forschung: Wie kann man den *Span of Understanding* verkürzen bzw. minimieren

Change Impact Analysis

Change Impact Analysis

- Die Change Impact Analyse versucht, den so genannten *Ripple-Effekt* erschöpfend zu beschreiben
  - Ripple Effekt einer Änderung
    - Effekt der sequentiellen Programm-Inkonsistenzen aufgrund einer initialen Änderung
- Der sogenannte „Change Impact“ kann dann
  - Abgeschätzt bzw. quantifiziert werden
- Damit werden
  - Änderungen planbar und alternative Vorgehensweisen abwägbar
  - Änderungen in der Software konsistent und vollständig durchführbar

Change Impact Analysis

- Definition
  - **Impact Analysis**
    - Research into the development of a method for predicting the effect of a change to an existing system as early as possible in the change cycle. The research is investigating how changes can be represented and methods for the automatic propagation of changes.
  - by Malcolm Munro

Change Impact Analysis

- Mehrere Ansätze, z.B.
  - analytisch
  - graphbasiert (Vaclav Rajlich)
- Input
  - Meist Abstract Syntax Tree (AST)
- Tools
  - Es gibt heute sehr leistungsfähige high-level Cross-Referencer, die detaillierte Change Impact Analysen erlauben

Change Propagation

Change propagation

- Definition
  - Change propagation ist der Prozess der sequentiellen Behebung von Programm-Inkonsistenzen aufgrund einer initialen Änderung
- Input
  - Korrektes Programm vor der Änderung
  - Change Impact Analysis
- Output
  - Korrektes Programm nach Durchführung der Änderung

Change Propagation spreads to Documentation

- Technical documentation has to be updated whenever a change to dataflow, design, architecture, module procedure, or any other related artefact is made
- Inaccurate documentation can be worse than no documentation
- Entire documentation should be previewed prior to re-release of the software

Wartungsdokumentation

Wartungshandbuch

- Beschreibt für die Wartung relevante Inhalte
  - Einführung in Architektur, Design, Systemumgebung, ... für den Wartungsingenieur
  - Wie ist im Wartungsfall vorzugehen?
  - Was sind die relevanten Dokumente für die verschiedenen Wartungsfälle?
  - Welche Einstiegspunkte in den Code gibt es?
  - Wie ist die Produktivstellung zu planen, was ist dabei zu berücksichtigen?
  - Wie wird die Wartung dokumentiert?

Projekttagebuch Wartung

- Der Wartungsmanager führt ein Projekttagebuch, das die einzelnen Wartungsfälle dokumentiert und zueinander in Beziehung stellt
- Management Summaries für das Top-Management können daraus periodisch produziert werden
- Das Projekttagebuch liefert Daten für die Prozessoptimierung der Wartung und die langfristige Nachvollziehbarkeit

Maintainability (Wartbarkeit)

Definition: Maintainability

- Maintainability is the ease of maintenance
- Can be decomposed as
  - Repairability
    - Ability to correct defects in reasonable time
  - Evolvability
    - Ability to adapt software to environment changes and to improve it in reasonable time

Maintainability - Design for Change

Design for Change

- Problem
  - Wie entwirft und implementiert man Software, sodass zukünftige absehbare bzw. nicht absehbare Änderungen möglichst leicht einzuarbeiten sind?
- Lösung
  - Es gibt mannigfaltige Ansätze in Forschung und Industrie
  - Aber keine „Silver Bullet“ Lösung

Design for Change: Ausgewählte Kapitel

- Design Metriken
  - Cohesion und Coupling
- Model-driven Architecture

Cohesion

- Beschreibt den Grad der logischen Abhängigkeiten innerhalb eines Software Moduls
- Je größer die Cohesion desto besser das Software Design
- Hinter einem Interface sollte die Cohesion maximal sein
- Große Cohesion erlaubt die einfache Wiederverwendung von Software Modulen

Coupling

- Coupling beschreibt den Grad der logischen Abhängigkeiten zwischen verschiedenen Software Modulen
- Je größer das Coupling desto schlechter das Software Design
- Das Coupling über Interfaces hinweg sollte minimal sein
- Großes Coupling verhindert die einfache Wiederverwendung von Softwaremodulen

Model Driven Architecture (MDA) und Software Wartung / Evolution

MDA - Motivation

- Technologie entwickelt sich in kurzen Zyklen weiter
  - Betriebssysteme, Datenbanken, Middleware, Komponentenmodelle, (GUI) Bibliotheken, Programmiersprachen
- Es gibt zu jedem Zeitpunkt mehr als eine adäquate Technologie, um ein System umzusetzen
  - z.B. CORBA, J2EE, .NET
- Die Geschäfts- oder Fachlogik ist beständiger als Technologien
  - Trotzdem wird die Fachlogik bei einem Technologiewechsel in der Regel neu implementiert

Was ist die MDA?

- Vorgehensmodell
  - Das *Platform Independent Model* (PIM) modelliert die Fachdomäne
  - Das *Platform Specific Model* (PSM) beschreibt die Anwendung in der konkreten Implementierungstechnologie
  - Transformation PIM -> PSM -> Code erfolgt im Idealfall voll automatisch
- MDA basiert auf Ideen der modellgetriebenen und generativen Softwareentwicklung

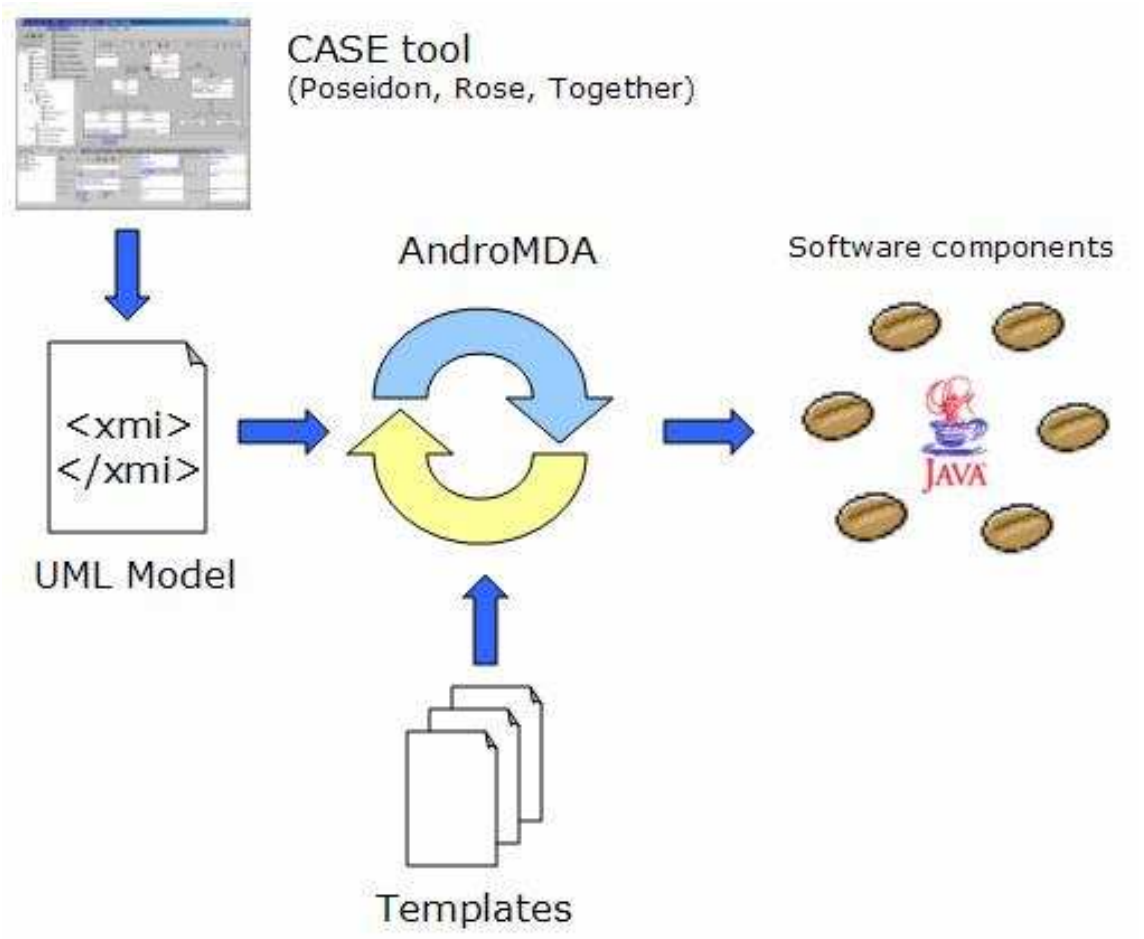
Technologien der MDA

- Unified Modeling Language (UML)
  - OCL für Pre-/Postconditions
  - Action Language für Semantik
- UML Profiles
  - Tailor the language to particular areas of computing (such as EDOC) or particular platforms (such as EJB or CORBA)
- Meta Object Facility (MOF)
  - Defines a standard repository for Meta-Models
  - Used to define information models for particular domains
- XML Metadata Interchange (XMI)
  - The XML-UML standard
- Common Warehouse Metamodel (CWM)
  - Forms the MDA mapping to database schemas

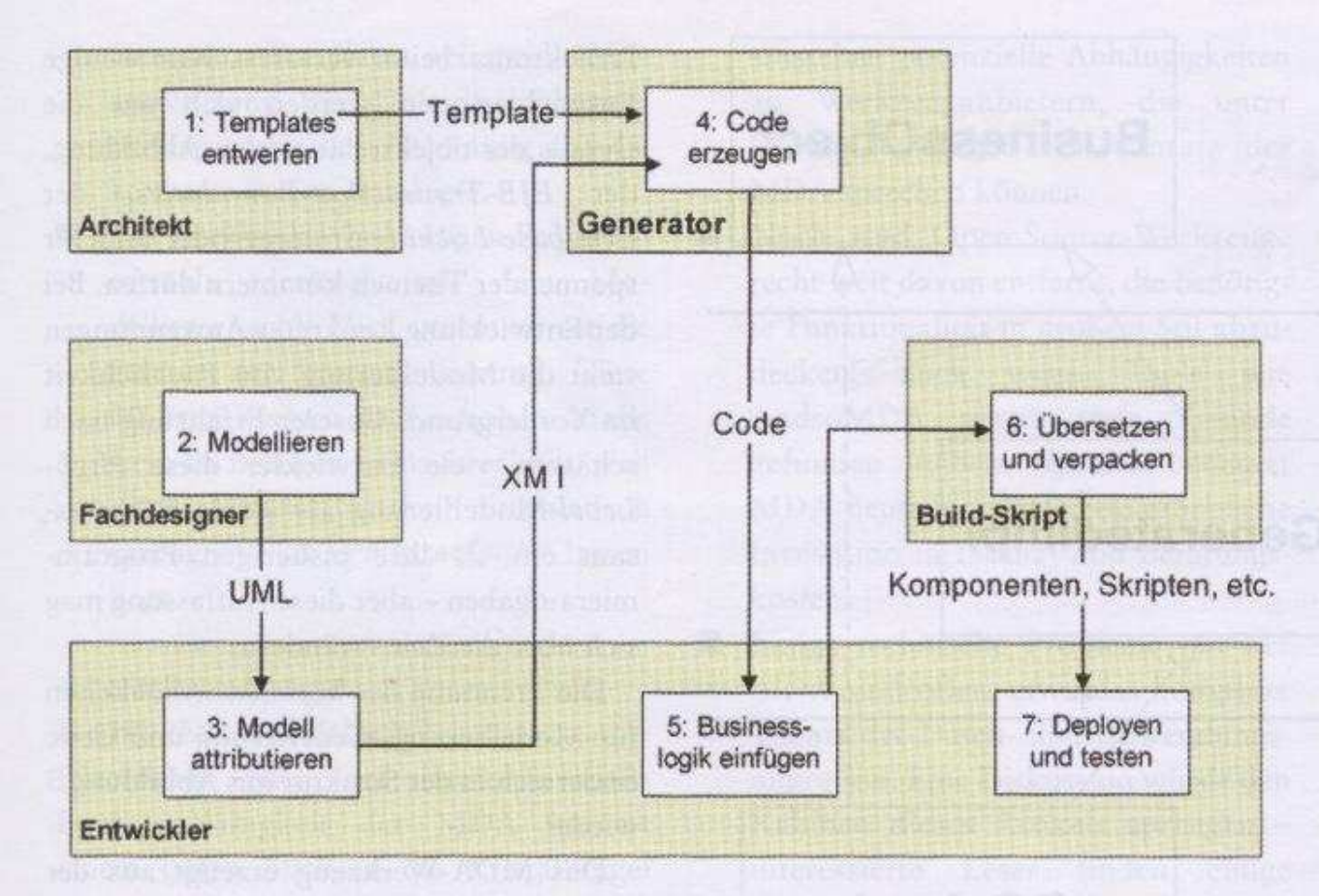
AndroMDA: „MDA light“

- Open Source Tool (www.andromda.org)
- PIM wird in UML modelliert
- Auf ein eigenständiges PSM wird verzichtet
- Template basierter Codegenerator
  - IN: UML Modell (fachlich, technisch attributiert)
  - OUT: Softwarekomponenten (J2EE, etc.)

AndroMDA: Konzept



AndroMDA: Vorgehensmodell



AndroMDA: Was wird generiert?

- Entity Beans (inkl. CMP, allen Interfaces, Utility Klassen, Implementation Stubs)
- Session Beans (alle Interfaces, Implementation Stubs)
- Struts Komponenten (JSP Page Stubs, alle Forms, Action Stubs, Config File!)
- Value Objects
- Deployment Deskriptoren (viele!)
- .ear File (Deployment Package)

AndroMDA: Womit wird generiert?

- Apache Komponenten
  - Ant (XML Build Engine)
  - Jakarta
    - Commons (allgemeine Utility Bibliotheken)
    - Taglibs (Tag Libraries)
    - Struts (MVC Web Application Framework)
    - Velocity (Template Sprache)
  - XML (xerces)
- Xdoclet (Code generation engine)
  - Reads meta data from javadoc tags

MDA: Bottom-line Benefits

- The benefits of MDA are significant to business leaders and developers alike
  - Reduced cost throughout the application life-cycle
  - Reduced development time for new applications
  - Improved application quality
  - Increased return on technology investments
  - Rapid inclusion of emerging technology benefits into their existing systems

[http://www.omg.org/mda/]

Probleme der MDA

- Stabilität des Datenmodells
  - Änderungen im Modell bedingen Datenmigration
- Entfremdung des Entwicklers vom Code
  - Generierung von zig Implementierungsklassen
- Unzulänglichkeiten in den Tools
  - z.B. Ändern eines UML Klassennamens
- Inkrementelle Generierung
- Debugging auf unteren Schichten / Error Reporting

Was erspart mir die MDA nicht?

- Software Artifact Management
  - Auf Modellebene!
- Verifikation / Test
  - Nur auf Modellebene?
- Optimieren
  - Nur auf Modellebene?
- Software Wartung und Evolution(splanung)

MDA und Wartung

- Generierter Code muss nicht gewartet werden
  - Fall-Beispiel „Xion xbib“ (AndroMDA)
    - J2EE Bibliotheksverwaltung im Applikationsserver Cluster (Failover, Load Balancing, Skalierbarkeit, CMP O/R Mapping)
      - Gesamt: 4308 SLOC
      - Manuell: 1815 SLOC (42,13%)
      - Generiert: 2493 SLOC (57,86%), 60% der Klassen, 80,79% der Methoden
- Ohne Code keine quick-fix Wartung möglich (?)
- Wartung quasi nur auf Modellebene möglich
  - Hier sind qualitativ hochwertige Informationen vorhanden
    - Anforderungen, Design, Design Decisions, Dokumentation, …
  - Damit „automatisch“ Iterative Enhancement Wartung

MDA und Evolution

- Three problems of evolution
  - How to express the change?
  - How to propagate the change?
  - How to manage/analyse the change?
- For software evolution to be automated, design and dependency information must be preserved.
- The MDA codifies design information in high-level instances, and dependency information in transformations.
- Different propagation techniques can be applied to the MDA to achieve automated software evolution.

[David Hearnden]

Software Wartung im unternehmerischen Kontext

Software Wartung im Unternehmen

- Stakeholder („beteiligte Parteien“)
  - Hardware/Systeme
  - Softwareentwicklung
  - Testabteilung
  - Betriebsführung
  - Rechenzentrum
  - Call Center
  - Benutzer
  - Kunden

Wartungsphilosophien

- „Throw it over the wall“ – someone else is responsible for maintenance
  - Investment in knowledge and experience is lost
  - Maintenance becomes a reverse engineering challenge
- „Mission orientation“ – development team makes a long term commitment to maintaining the software

Software Wartung im Unternehmen

- Linienorganisation
  - CTO
  - Bereichsleiter
  - Abteilungsleiter
  - Teamleiter
  - Ingenieur/Techniker
- Rollen
  - Planung/Management/Projektleitung
    - Wartungsmanager
  - Operative Durchführung
    - Wartungsingenieur

Wartungsmanager

- Aufgaben
  - Ganzheitliche Planung des Wartungsprozesses
  - Evaluierung, Machbarkeit, Kostenschätzung von Change Requests
  - Beauftragung und Coaching der Wartungsingenieure
  - Controlling und Qualitätssicherung der Durchführung
  - Planung von Wartungsfenster und Produktivstellung
  - Planung von Schulung
  - Dokumentation

Wartungsingenieur

- Aufgaben
  - Umsetzung der Wartungsaufträge (vgl. „Aktivitäten im Wartungfall“)
    - Analyse
    - Design
    - Implementierung
    - Dokumentation
    - Test
  - Durchführung der Systemmodifikation im Wartungsfenster
  - Pflege des Wartungshandbuches

Software Wartungs-Verträge

Gewährleistung

- Modalitäten der Gewährleistung (GWL)
  - Dauer der GWL
    - Standardsoftware (zeitlich unbegrenzte Überlassung)
      - GWL-Bestimmungen des Kaufrechtes
    - Individualsoftware
      - GWL-Bestimmungen des Werkvertrages (wenn nicht anders vereinbart: 6 Monate)
    - Überlassung auf bestimmte bzw. unbestimmte Zeit
      - GWL-Bestimmungen des Mietrechts
  - Wann beginnt die GWL-Frist
    - Mit dem Datum des Abnahmeprotokolls, wenn nicht anders vereinbart
  - Wo sind die GWL Bedingungen geregelt
    - Oft in den Allgemeinen Geschäftsbedingungen, ansonsten (bzw. Ausnahmen und Erweiterungen) im Vertrag

nach [Wolff01]

Gewährleistung

- Modalitäten der Gewährleistung (GWL)
  - Definition der Modalitäten der Behebung von Mängeln
    - Vor allem Reaktionszeit, Zeitdauer bis zur Behebung
  - Abgrenzung der Mängel, die unter GWL fallen
    - Keine GWL z.B. für Mängel, die aufgrund der Hardware-Konfiguration bzw. besonderer Beschaffenheit von Fremdsoftware auftreten
    - Klassifikation von Mängeln (z.B. betriebsverhindernd, betriebsbehindernd, nicht betriebsbehindernd)
- Die Rechtsfolgen der GWL treten nicht schon mit dem Vorhandensein der fehlerhaften Beschaffenheit ein, sondern müssen vom Erwerber **vor Gericht** durch Klage geltend gemacht werden

Software Wartungs-Verträge

- Obwohl Software ohne sachgemäße Pflege schnell unbrauchbar werden kann, wird diese vom Lieferanten nicht automatisch mit der Überlassung geschuldet
  - Kauf: Mängelbeseitigung während der GWL-Frist
  - Miete: Mängelbehebung während der Vertragsdauer
- Die erbrachten Leistungen der Software Wartung (korrektiv nach Ablauf der GWL, adaptiv, perfektionierend, präventiv) können
  - Teil der Leistung gemäß des Überlassungsvertrages sein (meist bei wiederkehrenden Zahlungen)
  - gemäß eines Programmwartungs-Vertrages gegen gesondertes Entgelt erbracht werden

nach [Wolff01]

Inhalt von Software Wartungs-Verträgen

- Herkömmlich
  - Korrektive, adaptive Wartung, perfektionierende Wartung, präventive Wartung
- Zusätzlich
  - Überlassung von neuen Programmversionen
  - Einweisung von Personal in neue Programmversionen
  - Technische Hilfe (z.B. telefonisch, über ein Portal)
  - Beratung beim Einsatz der Software
  - Aufklärung von Bedienungsfehlern
  - Beseitigung der Auswirkungen von Bedienungsfehlern
  - Nachrecherchieren von ungerechtfertigen Mängelrügen

nach [Wolff01]

Arten von Software Wartungs-Verträgen

- Gekoppelte Software Wartung als Nebenleistung im Rahmen eines Hardware-Miet oder –Wartungsvertrags
  - z.B. für Betriebssystem
- Nebenleistung im Rahmen eines Softwarelizenzvertrages
  - Mit periodischen Lizenzgebühren einschließlich Wartung
- Softwarewartung während der GWL-Frist eines Softwareentwicklungsvertrags bzw. eines Softwarelizenzvertrages
  - Mit pauschaler Gebühr, wobei das Entgelt Teil der Kosten der Entwicklung bzw. der Lizenz ist und sich die Leistungen auf die korrektive Wartung beschränken
- Selbständige Leistung im Rahmen eines Softwarewartungsvertrages
- Softwareunterstützung nach Aufwand
  - Wenn der Anwender aus irgendeinem Grund auf die dauernde Wartung nicht angewiesen ist

nach [Wolff01]

Mitwirkung des Anwenders

- Ist der Anwender verpflichtet, die letztgültige Programmversion einzusetzen?
  - Was passiert, wenn er dies nicht tut?
- Ist der Anwender zur Mitteilung der Änderung der Einsatz- und Betriebsbedingungen verpflichtet?
- Der Anwender soll die gemeinsam festgelegten Richtlinien für Fehlermeldungen einhalten
- Wie weit muss der Anwender bei der Analyse, Test, und Rekonstruktion der Fehlerbedingung mitwirken, welche Ressourcen muss er zur Verfügung stellen?

nach [Wolff01]

Software Wartungs-Verträge – Weitere Vertragsbestandteile

- Abgrenzung von Wartung und GWL
- Upgrade Policy bei neuen Versionen
- Übertragung der Software Wartung (z.B. bei Veräußerung des Systems)
- Infrastruktur für die Wartung (Lieferant und Kunde)
- Zeitpunkt der Ausführung von Fehlerkorrekturen, Produktivstellung
- Reaktionszeiten und -modalitäten und Eskalationsverfahren bei Nicht-Reagieren
- Sorgfaltspflicht in Bezug auf Wahrung von Datensicherheit und Datenschutz bei der Ausführung von Wartungsarbeiten
- Entgelt und Zahlungsbedingungen
- Vertragsdauer

nach [Wolff01]

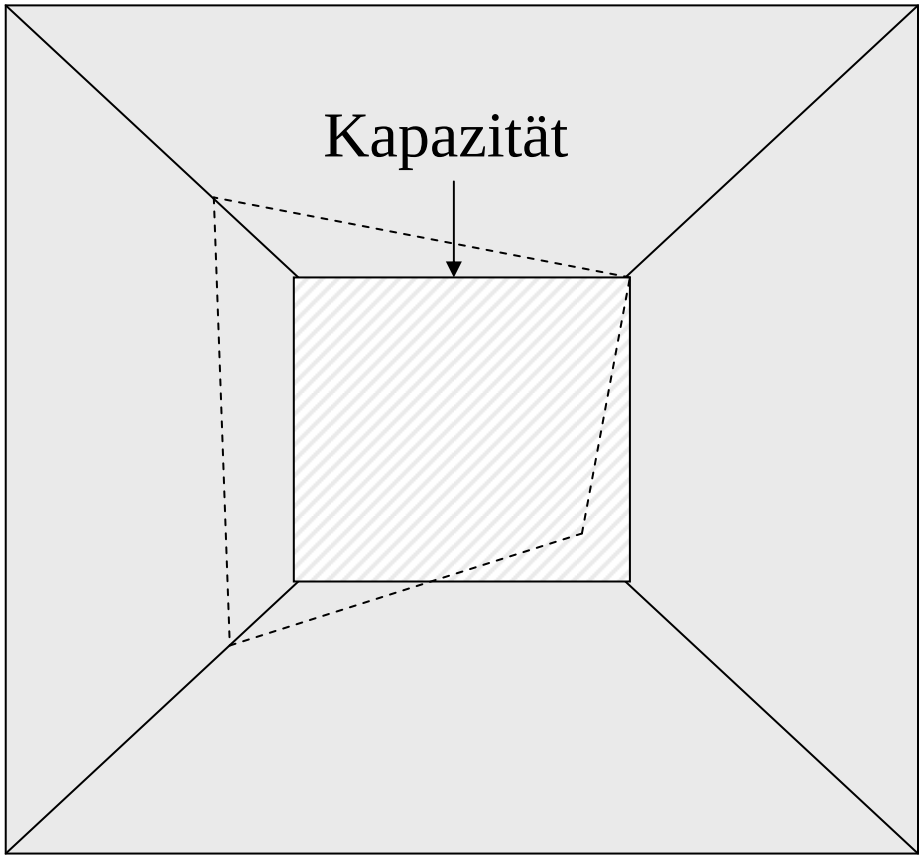
Software Wartungs-Verträge

- Weiterführende Literatur
  - „Das Vertragsrecht der Computer Software“
    - Skriptum zur Vorlesung „EDV Vertragsrecht“ an der Abteilung für Verteilte Systeme
    - Dr. Arthur Wolff, 2001
  - Jaburek, Handbuch der EDV-Verträge

Teufelsquadrat

Qualität

Ressourcen



Projektdauer

Wirtschaft
-lichkeit