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

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

Lecture 6

Lecture 6

- Inhalt
  - Keine zusätzlichen Folien

© J. Weidl-Rektenwald 02-06

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
    - Wartung und Test
  - Maintainability (Wartbarkeit)
    - Definition
    - Ensuring Maintainability
      - Design for Change
  - 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

Cognitive Elements

- Cognitive elements influence the comprehension process
  - Spatial layout of code
    - Vertical
    - Horizontal
  - Font
  - Syntax highlighting
  - Modularisation
  - Inline documentation

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

Murphy‘s Law of Change Propagation

- „If a change to a source statement can introduce an error, it will.“
  - Subprogram is deleted or changed
  - Statement label or identifier is deleted or modified
  - Changes are made to improve execution time
  - File open or close is modified
  - Logical operators are modified
  - Changes are made to boundary test

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

Wartung und Test

Wartung und Test

- Jeder noch so kleine Eingriff an einem Softwaresystem erfordert im Prinzip einen neuerlichen vollständigen Test des Systems, einen so genannten Regressionstest
- Durch Information Hiding und Komponentenorientierung werden Regressionstests oft auf einzelne Module beschränkt
- Automatische Unterstützung des Regressionstests führt zu dramatischen Qualitätsverbesserungen

xUnit

- Aus dem Extreme Programming Ansatz nach Kent Beck
- Zuerst wird der Testcode geschrieben, danach das Modul bzw. die Klasse
- Jedes Modul kann automatisiert und im Bulk getestet werden
- Erleichtert und automatisiert den Regressionstest nach Änderungen
- Liefert aber im Normalfall keine modulübergreifenden Aussagen
- xUnit existiert für mehrere Programmiersprachen (siehe http://www.xprogramming.com/software.htm)

JUnit

- www.junit.org
- JUnit is *Open Source* Software
- "Never in the field of software development was so much owed by so many to so few lines of code" (Martin Fowler)

```
public static void main (String[] args) {
   junit.textui.TestRunner.run (new
     TestSuite(MyTestClass.class));
   }
```

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

Ensuring Maintainability

- During Requirements Analysis
  - Minimal set of requirements documents
  - Standardize documentation
  - Automate documentation
  - Use patterns (see for example “Analysis patterns” by M. Fowler)
  - Review areas of future enhancement and potential revision, portability, and system interfaces
  - …

Ensuring Maintainability

- During Design
  - Limit complexity (see XP: “keep it simple”)
  - Use abstraction and information hiding
  - Use design patterns
  - Document assumptions and decisions
  - Maintain traceability to requirements (!)
  - Require reviews (data design, architecture, …) for ease of modification
  - …

Ensuring Maintainability

- During Implementation
  - Focus on:
    - Modularity
    - Reuse
    - Readability, Style, and Documentation
    - Consistency
  - Isolate dependencies
    - e.g. hardware and software
      - > use layering, minimize coupling
  - …

Ensuring Maintainability

- During Testing
  - Build a reusable test plan (or even reusable tests)
  - Build a test bed for maintenance
  - Automate the regression test as far as possible
  - Review for clues to potential maintenance problems
  - …

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

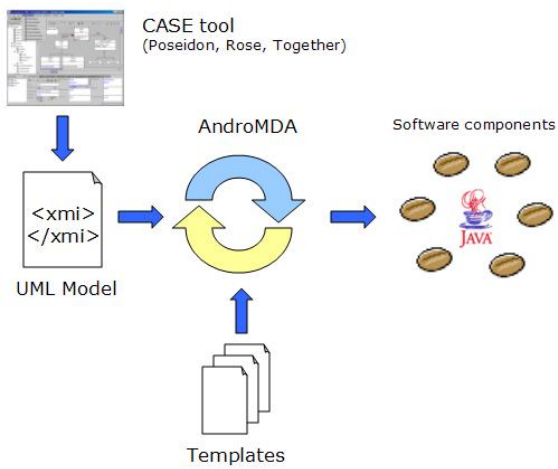
Technologien der MDA

- 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

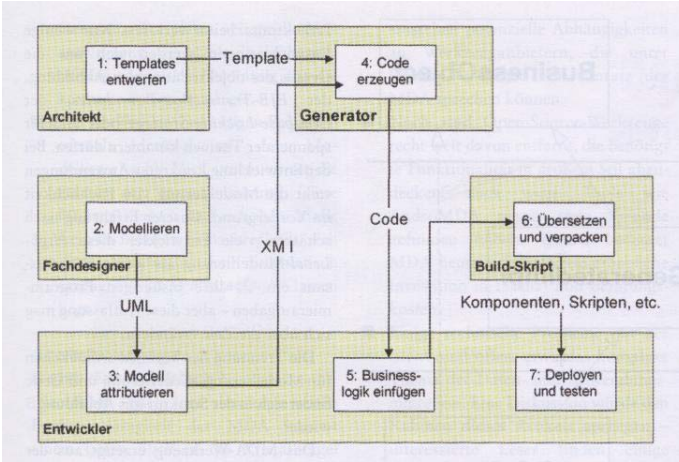
AndroMDA: „MDA light“

- 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.

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

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

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

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

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?

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

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