Praxis des Programmierens

Vorlesung: Dienstags, 14:15–16:00 Uhr, Gebäude 45, Hörsaal 002.

Übungen: Ab nächste Woche (?)

Dozent:
Uwe Waldmann,
Gebäude 46.1 (MPI), Raum 627,
Tel.: (0681) 9325-227 bzw. 92227,
E-Mail: `uwe@mpi-sb.mpg.de`
WWW: `http://www.mpi-sb.mpg.de`
`/~uwe/lehre/praxprog/`

Skript:
nein, aber Folienkopien auf Webseite

Scheinvergabe:
Übungsaufgaben + Teilnahme + Projekt
(keine groben Fouls!)

Behandelter Stoff

Softwareentwicklung

Objekt-orientiertes Programmieren

Java

Programmierstil/Dokumentation

Programmiertechniken

Entwurfsmuster

Projekt (Aussagenlogische Erfüllbarkeit)

Voraussetzungen

Grundlegende Programmierkenntnisse in C

Zeit

Literatur (SW-Entwicklung)

Ian Sommerville: *Software Engineering.*
Addison-Wesley, 1992.

Helmut Balzert: *Lehrbuch der Software-Technik (2 Bände).*
Spektrum Akademischer Verlag, 1996/1998.

Literatur (Java)

Ken Arnold, James Gosling:
The Java Programming Language.
Addison-Wesley, 1997.

Mary Campione, Kathy Walrath:
The Java Tutorial. Addison-Wesley, 1998.

Bruce Eckel: *Thinking in Java.*
Prentice Hall PTR, 1998.

Norman Hendrich: *Java für Fortgeschrittene.*
Springer-Verlag, 1997.

Ernst-Wolfgang Dieterich: *Java.*
R. Oldenbourg Verlag, 1999.

David Flanagan: *Java in a Nutshell: A Desktop Quick Reference.*
O'Reilly & Associates, 1997.

James Gosling, Bill Joy, Guy Steele:
The Java Language Specification.
Addison-Wesley, 1996.

Literatur (OO-Prog. & Entwurf)

Timothy Budd: *An Introduction to Object-Oriented Programming.*
Addison-Wesley, 1997.

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns.*
Addison-Wesley, 1995.

Grady Booch: *Object-Oriented Analysis and Design With Applications.*
Addison-Wesley, 1994.

1. Softwareentwicklung

Erwartungen an Software (I)

Qualität

Korrekt:

erfüllt Spezifikation/Erwartungen der Benutzer, versagt nicht (bzw. nicht häufiger als erlaubt).

Effizient:

vergeudet keine Ressourcen.

Ergonomisch:

Benutzerschnittstelle angepaßt an die Bedürfnisse, Fähigkeiten und Gewohnheiten der Benutzer.

Wartbar:

Änderungen einfach möglich (geänderte Rahmenbedingungen, erweiterte Anforderungen, Fehlerbeseitigung).

Erwartungen an Software (II)

Herstellung

Zeitplan wird eingehalten.

Personal wird effektiv eingesetzt.

Komponenten werden wiederverwendet/sind wiederverwendbar.

Erwartungen an Software (III)

Eigenschaften können nicht gleichzeitig optimiert werden.

Gewichtung der Eigenschaften je nach System unterschiedlich.

Software-Krise

um 1960:

schnellere und größere Rechner

$\Rightarrow$ viel leistungsfähigere Programme denkbar

aber: bisherige Entwicklungsmethoden nicht skalierbar

$\Rightarrow$ Projekte um Jahre verzögert, Software teuer und unzuverlässig, Wartung schwierig.

Systemgrößen

1. Programmfragmente: ein Dutzend Zeilen, ein Programmierer, kein Entwurf.
2. Kleine Programme und Module: einige Hundert Zeilen, ein Programmierer, Entwurf informell im Kopf.
3. Größere Programme, Komponenten: einige Tausend Zeilen, kleine Teams, Entwicklungskonzepte werden wichtig.
4. Systeme: 10^4–10^7 Zeilen, Hierarchie von Teams.
5. Multisysteme: Nicht mehr zentral kontrollierbar.

SW-Entwicklungstechniken (I)

Wasserfallmodell:

Phasenweise Softwareentwicklung: Planung, Entwurf, Implementierung, Modultest, Integration, Systemtest, Wartung. Jede Phase wird formell abgeschlossen, bevor nächste Phase in Angriff genommen wird.

Inkrementelle Entwicklung:

Entwicklung eines Minimalsystems; schrittweise Erweiterung der Grundfunktionalität.

Prototyping:

Ähnlich der inkrementellen Entwicklung; Prototyp dient im wesentlichen dazu, Anforderungen an das System zu ermitteln, und wird später ersetzt.

SW-Entwicklungstechniken (II)

Formale Transformation:

Entwicklung einer formalen Spezifikation, maschinell unterstützte Transformation der Spezifikation in ein lauffähiges Programm.

Wasserfallmodell (I)

Phasen:

Planung:

Machbarkeitanalyse; Anforderungsanalyse.

Entwurf:

Systemarchitektur; (schrittweise) Zerlegung in Komponenten.

Implementierung und Modultest:

Codierung; Test Modul $\leftrightarrow$ Spezifikation.

Integration, Systemtest:

Zusammenbau der Komponenten;
Test Gesamtsystem $\leftrightarrow$ Spezifikation;
Abnahme durch den Kunden.

Wartung:

System ist in Betrieb; ggf. Fehlerkorrektur, Funktionserweiterung.

Wasserfallmodell (II)

Unterstützt saubere Strukturierung.

Erlaubt (auch hierarchische) Aufteilung bei großen Projekten.

Management-freundlich.

In der Praxis: Phasen überlappen.
Voraussschau notwendig; Fehlentscheidungen einer Phase werden in späterer Phase offenbar.

Zyklen notwendig.

sonst: Designprobleme werden durch Implementierungstricks überbrückt.

Inkrementelle Entwicklung

Schrittweise Erweiterung der Grundfunktionalität:

> Entwurf → Implementierung → Test
> → Entwurf → Implementierung → Test
> → ...

Nützlich, falls präzise Spezifikation des Systems zu Beginn unmöglich (z. B.: Sprachverarbeitung, Mustererkennung, ...).
Hier häufig: Korrektheit $\rightsquigarrow$ Adäquatheit.

Klare Systemstruktur geht leicht verloren.
⇒ Wartung schwierig.

Enger Zeitplan erzwingt parallele Entwicklung.

Nicht Management-freundlich.

Mögliche Auswege?

Erweitere Wasserfall-Modell um Risiko-Analyse:

> Welche negativen Folgen kann eine momentane Entscheidung möglicherweise in einer folgenden Phase haben?
> Wie kann dem Risiko begegnet werden?
> Welche Alternativen gibt es?

Methoden mischen, z. B.:

> erst Prototyp, dann konventionelles Wasserfall-Modell.
>
> einzelne Module erst als Prototyp entwickeln.
>
> einzelne Module inkrementell entwickeln.

Teamstruktur (I)

Demokratisch:

Alle Mitglieder gleichberechtigt.

⇒ Gutes Arbeitsklima.

⇒ Kompetenz aller Mitglieder kommt zur Geltung.

⇒ Ausfall einzelner Mitglieder zu verkraften.

aber: Anzahl Kommunikationswege wächst quadratisch.

Verantwortlichkeit?

⇒ nur für kleine Teams und kleine Projekte bzw. Module.

Teamstruktur (II)

Chefprogrammiererteam:

Ein (oder sehr wenige) Programmierer trifft alle wichtigen Entscheidungen, entwirft und implementiert kritische Teile.

Andere Programmierer steuern Einzelteile bei.

+ Dokumentierer, Tester, ...

⇒ optimale Ausnutzung eines „Gurus“

aber: Ausfall des Chefprogrammierers fatal.

Arbeitsklima?

Team und Projekt dürfen nur so groß sein, daß Chefprogrammierer Übersicht behält.

Teamstruktur (III)

Hierarchisch:

Bei großen Teams/Projekten unvermeidlich.

Teamstruktur $\approx$ Produktstruktur.

Möglichst flache Hierarchie.

Seitlicher Informationsfluß?