Vorlesung + Übung Praxis des Programmierens

Ort: Di, 14-16, HS 002, Geb. 45.

Übungen: Ab nächste Woche, Wo??

Dozent: Dr. Peter Sanders, Geb. 46, Raum 315, Tel. 0681 9325 115, `sanders@mpi-sb.mpg.de`

Vorkenntnisse: Etwas Stoff von Informatik I/II, Programmieren in C

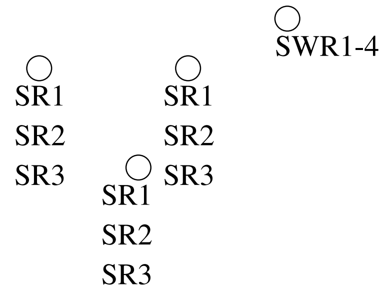
Scheinvergabe: Projekt $\wedge$ Programmieraufgaben $\wedge$ Übungsaufgaben

1 Einführung und Überblick

1.1 Projekt: Frequenzzuteilung für Radiosender

Eingabe und Ausgabe:

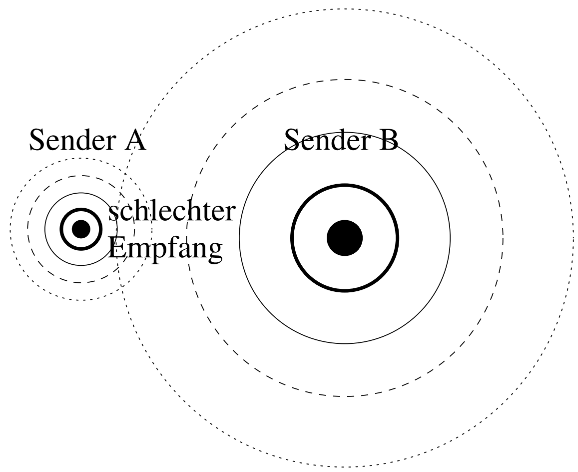
- Sender (Position, Reichweite,Anzahl gewünschter Kanäle ...)
- Partielle Zuteilung (für spezielle Benutzerwünsche, Zwischenspeichern, ...)
- Endergebnis: Zuordnung Programm→Kanal (pro Sender)



Schlüsselfeatures

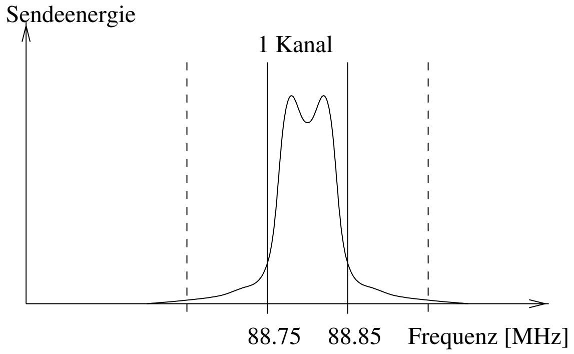
- Interaktiv
- Grafische Benutzeroberfläche
- Automatische Planung
- Flexibel: Frequenzbereiche, verschiedene potentielle Anwendungen:
  - Rundfunk
  - Fernsehen
  - Richtfunk
  - Mobiltelefon
  - Satelliten
  - XXXXXXXXX (top secret)
  - ...

Reichweiten



$d(A,B) <$ Reichweite(A) + Reichweite(B)
$\Rightarrow$ Störungsmöglichkeit

Kanalabstände



- "Benachbarte" Sender dürfen nicht den **gleichen** Kanal benutzen (für keinen unterstützten Sender)
- Übersprechen auf Nachbarkanäle ⇒ "Benachbarte" Sender dürfen nicht auf **aufeinanderfolgenden** Kanälen senden
- Senderbeschränkung (co-site constraints) Mehrere Kanäle Abstand bei Kanälen, die von gleichen Sender verwendet werden.

Ziel: Minimiere Zahl benötigter Kanäle

1.2 Vorlesungsinhalte

- Heute: **Mehr Überblick**
- Projekt und softwaretechnische Minimalausstattung
- 3 Vorlesungen: Eine Teilmenge von C++ und LEDA, nützliche Datenstrukturen
- Algorithmen-Know-How für das Projekt
- 2 Vorlesungen Softwaretechnik: Planung und Modellierung
- 2–3 Vorlesungen Programmiertechnik: Erhöhung der Code-Qualität, Fehlersuche, Werkzeuge
- Effiziente Programme
- Java

+ Einschübe: Projekt, C++

Übungsblätter

1.3 ISO C++

- Ein besseres C
- Objektorientierung = Klassen + Vererbung
- Generische Datentypen (templates)
- Ausnahmebehandlung (hier nicht)
- Namespaces, Varianten von `cast`, ... (hier nicht)
- Eine Standardbibliothek (hier wenig→LEDA)

1.4 Java statt C++? Pro & Contra

- Wichtig vs. wichtiger? (Ada, Pascal-Familie, Eiffel, Smalltalk, funktional, logisch, ...)
- Häßlich vs. häßlicher (Syntax, Felder, Includes)

+ Java lässt sich vollständiger vermitteln

– Java entbehrt wichtige Features, z.B. Generizität.

– Java ist *de facto* und z.T. *inhärent* ineffizient (z.B. zu viele Referenzen).

+ Einfachere Speicherverwaltung (LEDA gleicht das z.T. aus)

+ Bibliotheken (Benutzeroberfläche)

– Kein LEDA

- Unterschiedliche Specialties: z.B. Internet vs. Hardwarenähe

1.5 LEDA

Library of **E**fficient **D**ata **T**ypes and **A**lgorithms

Das „**E**“ könnte aber auch für „**E**legant“ oder „**E**infach benutzbar“ stehen; oder für „korr**E**kt“.

- Grundlegende Datenstrukturen (Listen, Wörterbücher, . . .)
- Graphen
- Geometrie
- Grafische Benutzeroberflächen
- Diverses (Langzahlarithmetik, . . .)

1.5.1 Ressourcen

- Homepage: `http://www.mpi-sb.mpg.de/LEDA/leda.html`
- Holen: `http://www.mpi-sb.mpg.de/LEDA/download/research.html`
  - **Solaris**, AIX, Irix mit `g++,egcs` oder Hersteller Compiler
  - **LINUX** mit `g++,egcs`
  - Windows95/WindowsNT: nur Objektcode, nur für Borland, MSVisualC++, WatCom
- Dokumentation: `http://www.mpi-sb.mpg.de/LEDA/information/`
- Handbuch: `http://www.mpi-sb.mpg.de/LEDA/MANUAL/MANUAL.html` Ältere Version auch auf Papier
- Leda-Buch: `http://www.mpi-sb.mpg.de/~mehlhorn/LEDAbook.html`

1.6 Literatur

1.6.1 Ein C++ Buch

- S. B. Lippmann and J. Lajoie. *C++ Primer.* Addisson Wesley, **3rd** edition, 1998. Basiert auf ISO-Standard, über 1200 Seiten, deutsche Version?
- B. Stroustrup. *The C++ Programming Language.* Addison Wesley, **3rd** edition, 1998. Kurz vor ISO-Standard, über 900 Seiten, deutsche Version verfügbar.

Wie liest man Telefonbücher?

1. Einführende Teile lesen
2. Rest querlesen bis durchlesen (je mehr Erfahrung man hat, desto *genauer*)
3. Erfahrungen sammeln
   - Programmieren
   - Buch als Nachschlagewerk benutzen
   - Erklärende und weiterführende Bücher lesen
4. Zu einzelnen Kapiteln zurückkehren

1.6.2 Objektorientierung allgemein

Bertrand Meyer.
Objektorientierte Softwareentwicklung.
Hanser, München, 1990.
Gut zu lesen. Programming by contract. Objektorientierung am Beispiel der Programmiersprache Eiffel. Englisch: OO Software Const. bei Prentice Hall.

1.6.3 Software-Engineering

W. S. Humphrey. *Introduction to the Personal Software Process.* SEI series in Software Engineering. Addison Wesley, 1997.
Quantitative Ansätze zur Verbesserung des Softwareentwicklungsprozesses. Für Studienanfänger geschrieben. Zeitaufwendig aber vielleicht lohnend.

Grady Booch. *Object-oriented analysis and design with applications.* Addison-Wesley, Menlo Park, CA, 2nd edition, 1994.
Gut geschrieben. Viele Beispiele.

P. Jalote. *An integrated approach to software engineering.* Springer, 2nd edition, 1997.
Für Studienanfänger geschrieben. Leicht lesbar aber etwas langweilig.

Helmut Balzert. *Lehrbuch der Software-Technik : Software-Entwicklung [Band 1].* Spektrum Akademischer Verlag, 1996.
gut strukturiert, $>$ 1000 Seiten.

1.7 Softwareentwicklungsprozesse

1.7.1 Systemgrößenstufen [Humphrey95]

0 Programmfragmente, z.B. ein dutzend Zeilen, einfache Prozuduren. Entwurf wäre Verschwendung

1 Kleine Programme und Module, einige hundert Zeilen. Typische Größe in Programmierübungen. Ein Programmierer. Design informell z.T. im Kopf.

2 Komponenten, tausende von Zeilen. Kleine Teams. Abstraktion, Schnittstellenspezifikationen und Programmiertechnikkonzepte werden wichtig.

3 Systeme. Beliebig groß, 10^4–10^7 Zeilen. Hierarchie von Teams. Unser Projekt–Betriebssystem (Solaris, ...)

4 Multisysteme. Nicht mehr zentral kontrollierbar.

1.7.2 Ziele

- Qualität: Korrektheit, Benutzbarkeit, Wartbarkeit
- Zeitplan
- Personal effektiv einsetzen
- Viele Benutzer (und $\neq$ Hersteller)
- Manchmal: Portabilität, Tuning, Ausfallsicherheit, Echtzeitanforderungen, verteilt, parallel, extern, ...

$\Rightarrow$ grosse Programm sind (pro Zeile Code) eine Größenordnung teurer als kleine „selbstgestrickte“ Programme.

1.7.3 Wasserfallmodell

Phasen:

Planung:
Machbarkeitsanalyse/Anforderungsanalyse + Systemspezifikation (Schnittstellen nach außen)

Entwurf: Aufbau → Schnittstellen (intern) → Detailentwurf (top down)

Implementierung: Codierung (mit Modultest)

Abnahme und Einführung: Integrationstest, Abnahmetest (vom Kunden) Probebetrieb, Installation

Wartungsphase: Fehlerbeseitigung, **Funktionserweiterung**

In dogmatischer Form unrealistisch oder nur bei Routineprojekten geeignet.

1.7.4 Prototyping

Während der Machbarkeitsanalyse wird ein Teil der Funktionalität versuchsweise realisiert.

Ziel: Mehr über das Problem lernen. Wichtig bei schwierigen Problemen und unentschlossenen Kunden

- Effizienz, Korrektheit, Dokumentation sind zweitrangig.
- Potemkinsche Dörfer OK
- Simulationen OK

⇒ Prototyp wegwerfen!

Akademischer Beitrag

ggf. Theorie statt Prototyping

1.7.5 Inkrementelle Entwicklung

Schrittweise Erweiterung einer Grundfunktionalität

(Entwurf Implementierung Test)*

Probleme:

- Darf nicht in Hacken ausarten
- Enge Zeitpläne erzwingen parallele Entwicklung von Features
- Wann hat System einen konsistenten, testbaren Zustand? (gutes Design kann das mildern)

1.7.6 Was wir machen könnten

1. Anforderungsanalyse
2. Entwurfsprototyp
3. Aufteilung in Komponenten
4. Spezifikation der Schnittstellen: (nach außen und zwischen Komponenten) Nebenprodukt: Teilentwurf der Komponenten
5. Optional Prototypen der Komponenten
6. Inkrementelle parallele Entwicklung der Komponenten. Für jeden inkrementellen Schritt:
   - Detailentwurf
   - Implementierung
   - Integrationstest mit einer stabilen Variante der anderen Komponenten (Anfangs Prototypen oder „Stubs“)
7. Abnahmetest

1.8 Teamstruktur

1.8.1 Demokratisch

Gruppenleiter ist bestenfalls erster unter Gleichen

+ Gutes Arbeitsklima
+ Kompetenz aller Mitglieder kommt zur Geltung
– Aber die kompetenteren Mitglieder kommen ggf. nicht genug zur Geltung
– Anzahl Kommunikationswege wächst quadratisch
– Wer ist verantwortlich?

⇒ Am ehesten für kleine Teams mit ähnlicher oder sich ergänzender Kompetenz.

Im Projekt:

- Für Module und kleinere Komponenten
- Für Optimierungsalgorithmen

1.8.2 Chefprogrammiererteam

Chefprogrammierer trifft alle wichtigen Entscheidungen, entwirft, implementiert kritische Teile

Assistent übernimmt administrative Aufgaben, vertritt Chefprogrammierer

Sekretär Verwaltet Dokumente etc.

Tester testet

Werkzeugmacher schreibt Teile, die alle Programmierer benötigen

Programmierer ach ja

Angeraten wenn

- Optimale Ausnutzung eine einzelnen „Gurus“
- Teilweise nur Anfangssetting
- Sensitive aber einfache Anwendungen (z.B. Finanzbereich)
- Bei uns am ehesten bei sehr engagiertem Bremser

1.8.3 Hierarchisch

+ Bei grossen Teams unvermeidbar
– Seitlicher Informationsfluss problematisch
– Gute Programmierer werden befördert ... (→ Bezahlung und Level entkoppeln?)

- Flach!
- Teamstruktur ≈ Produktstruktur