Lerninhalte und -ziele im Informatikunterricht der gymnasialen Oberstufe

Christian Maurer

Freie Universität Berlin
Institut für Informatik
Takustraße 9
D-14195 Berlin

URL www.inf.fu-berlin.de/~maurer
E-Mail maurer@inf.fu-berlin.de

Vorwort

Im Laufe des letzten Jahrzehnts war ich häufig in Gespräche mit Lehrerinnen und Lehrern verwickelt, in denen es darum ging, was denn nun eigentlich die Invarianten der Informatik im Schulunterricht seien.

Diese Frage liegt auf der Hand, denn die Informatik ist erst wenige Jahrzehnte alt – im Gegensatz zu den etablierten Schulfächern des gleichen Aufgabenfeldes wie der Mathematik, die seit Jahrtausenden gelehrt, und der Naturwissenschaften, die seit Jahrhunderten betrieben werden. Sie hat sich aber mittlerweile in ihrem Kern insoweit gesetzt, als sich – weltweit vergleichbare – Standards in der akademischen Lehre herauskristallisiert haben.

Anläßlich aktueller Diskussionen möchte ich hier versuchen, die eingangs gestellte Frage in einer Synopse zu beantworten.

Gerade wegen der sich förmlich überschlagenden Entwicklungen in vielen Bereichen, die den Blick auf's Wesentliche zu trüben drohen, und der entsprechenden Vielschichtigkeit fachdidaktischer Debatten scheint es mir dringend notwendig – wenn nicht längst überfällig – die informatischen Invarianten für den Fachunterricht an der allgemeinbildenden Schule zu identifizieren und – generisch, aber präzise – zu benennen. Es geht gewissermaßen darum, einen Meilenstein in der Entwicklung der Informatik in der Schule zu fixieren.

Anlaß der Ausarbeitung war, herauszufinden, welche Themenkreise sich für einen Profilkurs eignen – einem fakultativen Addendum zum Unterricht im ersten Jahr der gymasialen Oberstufe an der Berliner Schule. In ihm sollen ergänzende Themen behandelt werden, die – im Sinne des lokalen Ordnens von Freudenthal – in sich geschlossene Aufgabenbereiche bilden, ohne auf spätere Kursinhalte vorzugreifen. Sie sollten durch markante fachtypische Inhalte gekennzeichnet sein, um den Schülerinnen und Schülern eine fundierte Entscheidung zu ermöglichen, ob sie zu ihrer ins Auge gefaßten Wahl von Informatik als Leistungsfach im Kurssystem stehen wollen.

Eine derartige Untersuchung kann selbstverständlich nur in den Kontext klarer Vorstellungen über die inhaltlichen Fundamente eines Gesamtcurriculums für den Informatikunterricht in der gymnasialen Oberstufe eingebettet sein – alles andere wäre – wegen der Bedeutung des Profilkurses für die Schüler – unverantwortbares und in seinen Folgen nicht kalkulierbares Stückwerk.

Im Mittelpunkt aller Überlegungen stehen die folgenden Rahmenbedingungen:

Ein Versuch der Beschreibung der Invarianten ist letztlich äquivalent zum Postulat nach weitestgehender Stabilität gegen aktuelle Ausprägungen. Selbstverständlich muß Schule als allgemeinbildende Institution auf aktuelle Entwicklungen flexibel reagieren; das impliziert aber keineswegs, daß sie unreflektiert jede aktuelle Strömung unmittelbar in Schulunterricht umgießen muß – ihre Aufgabe ist vielmehr, die zentralen gedanklichen zeitinvarianten Kerninhalte herauszupräparieren und zu lehren.

Daneben muß Schule einen Beitrag zum Verständnis liefern, wie sich unsere Welt durch den massiven Einsatz von Rechnern ändert, die durch Programme gesteuert werden, die nichts anderes als von Menschen verfaßte Texte sind – mit allen damit verbundenen Risiken.

Unabdingbare Voraussetzungen für die Abbildung wissenschaftlicher Erkenntnisse auf Schulunterricht sind realistische Einschätzungen der Machbarkeit im Kontext des Bildungs- und Erziehungsauftrages der Schule – im Klartext: scharf durchdachte didaktische Reduktionen wesentlicher fachlicher Inhalte.

Was wäre wohl besser dazu geeignet, als die explizite Benennung von Lerninhalten? Ihre Angabe in Form operationalisierter Lernziele dient lediglich der Klärung ihrer curricularen Abhängigkeiten und der Festlegung von Intentionen. Additive Niederschriften von Begriffen überlassen ihre Behandlung dagegen der vollständigen Beliebigkeit der Interpretation – bis herunter zur ihrer unverbindlichen Erwähnung; meine Absicht ist aber, den Rahmen für die Erkenntnisse *hinter* den Begriffen zur Debatte zu stellen, ohne längliche Details auszubreiten. Als konzise Methode zu diesem Zweck ist die Operationalisierung von Lernzielen unübertreffbar. Man extrahiere aus ihnen alle Substantive und die zugehörigen Adjektive – damit sind die Lerninhalte in einer feineren Struktur definiert als jeweils zu Beginn ausgewiesen.

Die Lerninhalte und Lernziele sind bewußt auf einem abstrakten Niveau formuliert, das den Einsatz unterschiedlicher programmiersprachlicher Paradigma und unterrichtlicher Schwerpunktsetzungen, Akzentuierungen und inhaltlicher Ausgestaltungen erlaubt – unter Berücksichtigung konkreter schulischer Rahmenbedingungen und mit dem Blick auf aktuelle Entwicklungen; insbesondere unter Gewährleistung einer weitestgehenden Methodenvielfalt.

Um es informatisch auszudrücken:

Es handelt sich um eine Spezifikation, deren Implementierung den Absprachen der Fachkonferenzen an der jeweiligen Schule überlassen bleibt. Insbesondere muß anhand der lokalen Gegebenheiten entschieden werden, welche Programmiersprachen im Unterricht eingesetzt werden sollen.

Diese Frage wird hier ausgeklammert, weil sowohl das funktionale wie das imperative Paradigma für die Umsetzung der Lerninhalte gut geeignet ist. Daneben ist es durchaus möglich, Teile des Unterrichts auf deklarative Sprachen, insbesondere Datenbankanfragesprachen, abzustützen oder Objektorientierung stärker zu betonen.

Vorrangig ist die Frage nach den Inhalten:

Für den Informatik-Unterricht in der gymasialen Oberstufe sind in erster Linie die Kerngebiete des Grundstudiums relevant; ferner gesellschaftliche Fragen, soweit sie nicht Gegenstand des Unterrichts in Sozialkunde oder Politischer Weltkunde sind, sowie die exemplarische Beschäftigung mit einem komplexeren System – insbesondere zur Ausprägung der Eindrücke, was „Programmieren“ im Grunde heißt.

Damit ist der Pflichtbereich abgesteckt:

- Algorithmen und Datenstrukturen,
- Rechnersysteme,
- Automaten und formale Sprachen und
- Gesellschaftliche Aspekte der Informatik.

Die Einteilung der entsprechenden Kapitel entspricht dabei den vier Kurshalbjahren der Jahrgangsstufen 11 und 12.

In einem Wahlpflichtbereich – etwa in der Jahrgangsstufe 13 – kommen Gebiete in Frage, deren fachliche Hintergründe nichts anderes als diese Grundlagen voraussetzen und sich nachweislich didaktisch auf Schulniveau reduzieren lassen, wie z. B.

- ein Softwareprojekt und
- Datenbanksysteme oder
- Nichtsequentielle Programmierung.

Die strikte Trennung dieser Gebiete im Unterricht ist keineswegs zwingend; eine teilweise Integration kann durchaus sehr sinnvoll sein: Beispielsweise können Automaten gut im Kontext der Rechnersysteme behandelt werden; Gesellschaftliche Aspekte finden an verschiedenen Stellen im Unterricht ihren Raum und ein Softwareprojekt kann in Verbindung mit einem Wahlpflichtbereich sehr an Reiz gewinnen.

Manche Themen – wie bedeutend auch immer sie für andere Schulfächer sein mögen – sind dagegen deutlich von Informatik-Unterricht abzugrenzen:

Die *Anwendung* von Informatik-Systemen hat nur *dann* etwas mit den Inhalten des Schulfaches Informatik zu tun, wenn es sich um Werkzeuge für den Unterricht oder um informatische Themen handelt, die schülergerecht aufbereitet werden können; der Rechnereinsatz in anderen Schulfächern ist eine Angelegenheit eben dieser Fächer; auch die bloße Nutzung des weltweiten Netzes definiert keine ernsthaften Lernziele des Faches Informatik, kann demzufolge ebenfalls nicht Gegenstand des Informatik-Unterrichts sein.

Ansonsten ist festzuhalten, daß Komplexität und Schwierigkeitsgrad der meisten übrigen Gebiete der Informatik erheblich zu hoch sind, als daß sie sich – abgesehen von einfachsten Grundbegriffen – auf Schulunterricht abbilden lassen.[1] Dazu gehören Gebiete wie z. B. Betriebssysteme (Vor.: Nichtsequentielle Programmierung, Rechnerorganisation), Verteilte Systeme (Vor.: Betriebssysteme), Expertensysteme (Vor.: Deklarative Programmierung und Datenbanksysteme) oder Künstliche Intelligenz (Vor.: Expertensysteme u. v. a. m.), die daher nicht in den Kanon des „normalen“ informatischen Schulstoffs aufgenommen werden können.

Dank gebührt meinem Kollegen P. Bartke; diverse Grundideen in dieser Zusammenstellung wären ohne seine vielfältigen Anregungen weniger klar. Darüberhinaus hat er das Kapitel über Rechnersysteme maßgeblich beeinflußt.

Die vorgelegte Arbeit ist noch nicht als vollständig zu betrachten: sie soll mit einem Kapitel über Datenbanksysteme fortgesetzt werden.

Über jede Anregung und jeden Hinweis auf Widersprüche freue ich mich im Interesse der erstrebenswerten Normalität des Schulfaches Informatik und seines wichtigen Beitrags zur Allgemeinbildung.

Berlin, im März 2000 Christian Maurer

---

[1] Der Mathematik- bzw. Physikunterricht befaßt sich ja z. B. auch nicht mit Algebraischer Topologie oder Quantentheorie: deren Voraussetzungen (Topologie, Homologische Algebra und Garbentheorie bzw. Theoretische Mechanik, Elektrodynamik, Differentialgleichungen und Funktionalanalysis) liegen am Ende oder jenseits des Grundstudiums.

Inhalt

Literatur

Dewdney, A. K.: *Der Turing-Omnibus*
Eine Reise durch die Informatik in 66 Stationen
Springer-Verlag, 1995

Nievergelt, J.: *„Roboter programmieren“ – ein Kinderspiel*
Bewegt sich auch etwas in der Allgemeinbildung?
Informatik-Spektrum 22 (1999), 364–375

Rechenberg, P.: *Mythen und Fetische des Computer-Zeitalters*
LOG IN 19 (1999), Heft 2, 28–33

Rechenberg, P., Pomberger, G. (Hrsg.): *Informatik-Handbuch*
2., aktualisierte und erweiterte Auflage
Carl Hanser Verlag, 1999

Hinweis

zur Bedeutung der Markierungen bei der Angabe der Lernziele:

- • Lernziele in Basis- bzw. Grundkursen *und* in Profil- bzw. Leistungskursen
- ∘ Alternativen für vertiefende Lernziele *nur* in Profil- bzw. Leistungskursen.

Algorithmen und Datenstrukturen

1. Elementare Programmstrukturen

Lerninhalte:

Atomare Datentypen und arithmetisch-logische Operationen, elementare algorithmische Konstrukte und Typkonstruktoren, Rekursion als Sprachmittel.

Lernziele:

- Einfache Programme zur Berechnung arithmetischer und boolescher Ausdrücke unter Verwendung der zugehörigen Zugriffsoperationen (Rechen-, Vergleichs- und logische Operationen) und entsprechenden Grundgesetzen erstellen können.
- Mit atomaren Datentypen und algorithmischen Grundkonstrukten im verwendeten Programmierparadigma – *z. B. funktional*: Definitionen und Signaturen von Funktionen, Auswertungen, Musteranpassung *oder imperativ*: Variablenkonzept und Wertzuweisungen, einfache Prozeduraufrufe, Kontrollstrukturen – sicher umgehen
- und dabei inhärent rekursive Probleme als solche erkennen und mit (linearer) Rekursion – *z. B. funktional*: rekursive Funktionen *oder imperativ*: rekursive Prozeduraufrufe – bearbeiten und ihre Terminierung begründen können.
- Elementare Typkonstruktoren im verwendeten Programmierparadigma – *z. B. funktional*: Listen, Tupel, Variantentypen, elementare Polymorphie *oder imperativ*: Felder, Verbunde, Teilbereichs- und Aufzähltypen – problemgerecht anwenden können.
- Diese Grundtechniken zur Lösung kleinerer Probleme im Kontext unterschiedlicher Anwendungen einsetzen können wie z. B.
  - Zählprozesse, Wertermittlungen und Wertetabellen
  - Verarbeitung von Zeichenketten, Wortspiele
  - Umwandlungen von Darstellungen, Codes und Einheiten
  - Wort- und Zahlenrätsel
  - logische Fragestellungen, Logeleien
  - numerische, geometrische und kombinatorische Berechnungen
  - Lösungen von Packproblemen
  - graphische Darstellungen einfacher Tabellen, Blockgraphik
  - Simulationen von einfachen Automaten oder Modellrobotern

- Rekursion als Sprachmittel zur Bearbeitung anspruchsvollerer kleiner Probleme einsetzen können wie z. B.
  - Konstruktion der natürlichen Zahlen
  - primitiv rekursive Definitionen arithmetischer Operationen
  - rekursive Definitionen syntaktischer Grundbegriffe der verwendeten Programmiersprache (z. B. Ausdrücke, Anweisungsfolgen)
  - Bearbeitung einfacher rekursiver Strukturen (z. B. Listen, Terme, Operator- und Codebäume)
  - Nachweis der Gleichheit von Funktionen durch strukturelle Induktion
  - Anwendungen generativer Grammatiken (z. B. Klammergebirge, Palindrome, hierarchische Strukturen, fraktale Kurven, Lindenmayer-Systeme)

2. Elementare algorithmische Abstraktionsprinzipien

Lerninhalte:

Funktionstypen bzw. Prozedurkonzept, Parametrisierung, Lokalität, lineare Strukturen und elementare Zugriffsoperationen, Rekursion versus Iteration, naiver Komplexitätsbegriff.

Lernziele:

- Algorithmische Muster für einfache Berechnungen unter Einsatz der im verwendeten Programmierparadigma dafür vorgesehenen Abstraktionskonzepte – *z. B. funktional*: Funktionstypen, Funktionen als Parameter *oder imperativ*: Prozedurkonzept und Parametermechanismen – entwickeln und die Gründe für ihren Einsatz (z. B. Strukturierung, Verallgemeinerung, Wiederverwendbarkeit) benennen können.
- Diese Abstraktionen syntaktisch korrekt – *z. B. funktional*: durch Angabe der Signatur *oder imperativ*: durch Parametrisierung mit Typangaben und ggf. Angaben der Parameterart – und semantisch präzise, d. h. vollständig und widerspruchsfrei (z. B. durch Angabe von Voraussetzungen und Funktionswerten bzw. Effekten) spezifizieren können.
- Aktual- von Formalparametern unterscheiden und die Abarbeitung funktionaler bzw. prozeduraler Abstraktion – *z. B. funktional*: der Auswertung einer Funktion *oder imperativ*: des Aufrufs einer parametrisierten Prozedur – protokollieren können („Trockentest").

- Diese Mechanismen schachteln – *z. B. funktional*: Komposition von Funktionen, lokale Definitionen *oder imperativ*: lokale Prozeduren und Prozeduren mit Funktionswerten als aktuellen Parametern – können.
- *Nur im funktionalen Paradigma*:
  Rekursive Funktionen mit geeigneten Funktionalen rekursionsfrei darstellen können.
- *Nur im imperativen Paradigma*:
  Die bei der Verwendung von Prozeduren auftretenden Lokalitätsprobleme (Gültigkeits- und Sichtbarkeitsbereiche von Variablen) beherrschen können.
- *Nur im imperativen Paradigma*:
  Ggf. unterschiedliche Parametermechanismen (insbesondere Ein- und Ausgabeparameter) sachgerecht verwenden und das Prinzip ihrer Funktionsweise erläutern können.
- Rekursion von Iteration – *z. B. funktional*: Endrekursion mit Akkumulatortechnik *oder imperativ*: Entrekursivierung durch Umwandlung in Schleifen – abgrenzen können.
- Lineare Darstellungen ungeordneter und geordneter Mengen – *z. B. funktional*: als Listen *oder imperativ*: als Felder – mit elementaren Verfahren (lineare Suche und elementare Sortierverfahren) bearbeiten und begründen können, daß das mit linearem bzw. quadratischem Berechnungsaufwand verbunden ist.
- Diese Techniken zur Lösung weiterer kleinerer Probleme aus unterschiedlichen Anwendungsbereichen einsetzen können wie z. B.
  - Einfügen, Suchen und Ändern von Einträgen in Verzeichnissen
  - Kalenderberechnungen
  - Simulationen einfacher Spiele
  - Erstellung kleiner Rechnungen oder Tabellen
  - Konstruktion von Zahlbereichserweiterungen
  - Simulation von Zufallsexperimenten
  - Anwendungen zellulärer und selbstreproduzierender Automaten
  - Auswertung und Darstellung von Meß- oder statistischen Daten
- Einfache Beispiele nichtlinearer Rekursion (z. B. Türme von Hanoi, Binomialkoeffizienten, Fibonacci-Zahlen) kennen und begründen können, daß das mit exponentiellem Berechnungsaufwand verbunden ist.

3. Datenabstraktion und höhere Programmstrukturen

Lerninhalte:

Abstrakte Datentypen und -objekte als Programmkomponenten, Spezifikationen als Schnittstellenbegriff; „Teile & Herrsche“-Prinzip (höhere Such- und Sortierverfahren), Rückverfolgungsstrategien.

Lernziele:

- Den Objektbegriff als Leitlinie zur Konstruktion von Programmkomponenten in Form abstrakter Datentypen und -objekte herausarbeiten und (z. B. im Hinblick auf Zuverlässigkeit und Wiederverwendbarkeit) bewerten können.
- Objekt- und Metasprache am Beispiel von Implementierung und Spezifikation sauber trennen können.
- Spezifikationen grundlegender abstrakter Datentypen (z. B. Zahlbereiche, Zeichenketten, geordnete Mengen, geometrische Objekte, Kellerspeicher, Warteschlangen, Kalenderdaten) mit Standardoperationen angeben, ausgewählte Teile auf der Basis einfacher Repräsentationen implementieren, ihre Funktionsweise durch sorgfältige Inspektion des Programmtextes nachvollziehen und damit ihre Korrektheit begründen können,
- insbesondere Zugriffsoperationen auf Mengen mit Ordnungsrelationen, die durch lineare Strukturen repräsentiert sind, mit höheren Verfahren (binäre Suche, Quicksort, Sortieren durch Verschmelzen) implementieren und begründen können, daß das mit logarithmischem bzw. linear-mal-logarithmischen Berechnungsaufwand verbunden ist,
- und dabei das „Teile-&-Herrsche“-Prinzip als informatisches Konzept herausarbeiten können.
- Das Prinzip von Rückverfolgungsalgorithmen auf – *z. B. funktional*: Listen von Listen *oder imperativ*: doppelt indizierten Feldern – angeben können (z. B. Acht-Damen-Problem, Springer-Problem).
- Die behandelten Datentypen bzw. -objekte und Algorithmen in verschiedenen Anwendungsbereichen wie in 2. anwenden können.
- Gründe dafür angeben können, daß Tests nur zum Nachweis der Existenz von Fehlern, aber nicht zum Beweis ihrer Abwesenheit taugen.
- ◦ Die Korrektheit von Implementierungen in geeigneten einfachen Beispielen halbformal oder formal nachweisen können.

- *Nur im imperativen Paradigma*:
  Prozedurtypen zur Entwicklung generischer algorithmischer Muster benutzen können.
- Aufwendigere Datentypen mit den zugehörigen komplexeren Algorithmen im Kontext kleiner Anwendungssysteme implementieren können wie z. B.
  - Warteschlangen als Ringpuffer
  - Irrgärten als zweidimensionale Felder
  - Verzeichnisse als Tabellenmengen (Felder von Verbunden)

4. Komplexere abstrakte Datentypen und effiziente Algorithmen

Lerninhalte:

Spezifikation und Implementierung komplexerer linearer und verzweigter Datenstrukturen mit ausgewählten Zugriffsoperationen.

Lernziele:

- Elemente in einseitig (oder effizient beidseitig) traversierbare Folgen – *z. B. funktional*: (Paare von) Listen *oder imperativ*: einfach (oder doppelt) verkettete Listen – einfügen und aus ihnen entfernen sowie die Folgen traversieren können.
- *Nur im imperativen Paradigma*:
  Persistente Mengen durch sequentielle Dateien repräsentieren, Elemente einfügen und suchen sowie die Mengen traversieren können.
- Geordnete Mengen durch binäre Suchbäume repräsentieren, Elemente einfügen und suchen sowie die Mengen traversieren können.
- *Nur im imperativen Paradigma*:
  Vor- und Nachteile statischer bzw. dynamischer Repräsentationen abstrakter Datentypen (Felder versus Geflechte) angeben können.
- Mindestens ein Beispiel für alternative Implementierungen eines abstrakten Datenobjekts oder -typs kennen und sie hinsichlich ihrer Effizienz für mögliche Verwendungszwecke vergleichend bewerten können.
- Die behandelten Datentypen zur Lösung angemessen dimensionierter Probleme aus unterschiedlichen Bereichen anwenden können wie z. B.
  - Benutzung von Karteien, Adreßbüchern, Inventarverzeichnissen
  - Textbearbeitung, Editieroperationen

  - Verwaltung von Finanzen
  - Auswertung von Turnieren
  - Symbolisches Rechnen (z. B. Grundzüge elementarer Computeralgebra)
  - Konstruktion fehlerkorrigierender Codes
  - Verschlüsselung durch kryptographische Verfahren
  - Anwendungen von Spielbäumen
- Halden über ihre Invarianten charakterisieren, Elemente unter Erhalt der Invarianten einfügen und aus ihnen entfernen sowie Halden zum Sortieren und zur Konstruktion von Prioritätsschlangen verwenden können.
- Geordnete Mengen durch bestimmte Klassen ausgeglichener Bäume repräsentieren, ihre Invarianten charakterisieren, Elemente suchen und unter Erhalt der Invarianten einfügen und entfernen können.
- Numerische Ausdrücke mit Hilfe von Operatorbäumen darstellen und die Ausdrücke per Traversierung der Bäume auswerten können.
- Daten in Codebäumen komprimiert ablegen und effizient restaurieren können (Quadtrees, Huffman-Bäume u. ä.).
- Elemente per Direktzugriff in Mengen einfügen und suchen können (Repräsentation in Feldern, Zugriffe über Streuspeicherfunktionen, Überlaufbehandlung).
- Elemente und kürzeste Verbindungen in Graphen suchen können (Tiefen- und Breitensuche, kürzeste Wege, minimale spannende Bäume).
- Ausgewählte Teile von umfangreichen abstrakten Datentypen spezifizieren und implementieren und zur Lösung komplexer Anwendungsprobleme einsetzen können.

Rechnersysteme

1. Darstellungen von Daten und Zahlen, Grundrechnungsarten

Lerninhalte:

Zeichen- und Zifferncodes, Binärcodes, Darstellung von Zahlen, Dualzahlarithmetik, Gleitkommazahlen.

Lernziele:

- Beispiele für Codes für Zeichen (z. B. ASCII, Telegrafencodes) und Ziffern (ausgewählte Tetradencodes) angeben können.
- Einfache Beispiele fehlererkennender Codes (Paritätskontrollcodes, Prüfzifferverfahren) angeben können.
- ◦ Fehler an Beispielen von Codes mit Hammingabstand > 2 erkennen und korrigieren und die Funktionsweise des entsprechenden Verfahrens erläutern können.
- Natürliche Zahlen in Stellenwertsystemen unterschiedlicher Basis darstellen und diese Darstellungen (z. B. mittels Horner-Schema) ineinander umwandeln können.
- Dualzahlen addieren und multiplizieren und dabei die Rolle des Übertrags herausarbeiten können.
- ◦ Dualzahlen dividieren, die Dualdarstellung von Brüchen entwickeln und den Wert einfacher periodischer Dualbrüche mit geometrischen Reihen bestimmen können.
- Beschränkte Dualzahlen mittels Darstellung im Zweierkomplement subtrahieren und die Korrektheit des Verfahrens (z. B. am Zahlenring) unter Berücksichtigung des Überlaufs erläutern können.
- Die Grundidee der Darstellung von Gleitkommazahlen und ihren Bezug zur „wissenschaftlichen Notation“ darlegen können.
- ◦ Ein Format zur Darstellung reeller Zahlen in seinen Grundzügen entwickeln und die Gleitkommaarithmetik einschließlich der Problematik von Rundungsfehlern in Ansätzen umreißen können.

2. Schaltfunktionen und Schaltnetze

Lerninhalte:

Gesetze der Schaltalgebra, Grundgatter, Schaltfunktionen, einfache Schaltnetze, Minimierungsverfahren.

Lernziele:

- Gesetze der Schaltalgebra tabellarisch angeben und die entsprechenden Grundgatter grafisch darstellen können.
- ◦ Grundzüge der technischen Realisierung von Grundgattern (z. B. Transistorschalter, NAND-Gatter) nachvollziehen können.
- Den Begriff der Schaltfunktion definieren und Normalformen von Schaltfunktionen zur Lösung einfacher Probleme angeben können.
- Schaltfunktionen durch Termumformung oder grafische Methoden (KV-Diagramme) minimieren können.
- Einfache Schaltnetze (z. B. Arithmetik- und Codierschaltungen) entwickeln und mit Grundgattern darstellen können;
- ◦ auch im Rahmen von Rechner-Simulationen oder Experimenten.
- ◦ Einen ausgewählten Aspekt der Konstruktion von Schaltnetzen (z. B. Implementierungen mit programmierbaren Grundbausteinen, Minimierung mit algorithmischen Verfahren) an Beispielen bearbeiten können.

3. Schaltwerke und Rechnermodell

Lerninhalte:

Gatterschaltungen mit Rückkopplung, Zustandsübergangstabellen und Entwurf von Schaltwerken, Komponenten des v. Neumann-Rechners.

Lernziele:

- Das Grundprinzip der Realisierung von Speicherelementen durch bistabile Gatterschaltungen erklären können.
- Zu einfachen Problemstellungen einen Zustandsgraphen entwerfen und daraus eine Zustandsübergangstabelle ableiten können.
- Schaltwerke von Schaltnetzen abgrenzen und das Schaltwerkskonzept nach Moore und Mealy erläutern können.
- Die Funktionsweise einfacher Schaltwerke (z. B. Flipflops, Schieberegister, Zählschaltungen) z. B. anhand von Zustandsübergangstabellen nachvollziehen können.

- ◦ Zur Erkennung von Bitmustern einen systematischen Schaltwerksentwurf mit D-Flipflops durchführen können.
- • Ein Blockschaltbild eines v. Neumann-Rechners skizzieren, seine Komponenten benennen und ihre Wechselwirkungen beschreiben können.
- ◦ Ausgewählte Komponenten des v. Neumann-Rechners zu Registermodellen auf Schaltwerksebene verfeinern und ihre Funktionsweise auf der Registertransferebene beschreiben können.

4. Maschinennahe Programmierung

Lerninhalte:

Hol- und Ausführungszyklus, Ausführung von Maschinenbefehlen, Adressierungarten, Maschinenprogrammierung.

Lernziele:

- • Einfache Beispiele imperativer Anweisungen (mit einem realen oder einem Modell-Assembler, möglichst in einer Zweiadreßsprache) in Maschinenbefehle umsetzen, ihren Ablauf darstellen
- • und dabei die Rolle der beteiligten Busse herausarbeiten und den v. Neumann'schen Flaschenhals identifizieren können.
- ◦ Den Hol- und -Ausführungszyklus am Beispiel des Ablaufs einfacher Maschinenbefehle in Form von Registertransferoperationen erläutern können.
- • Direkte und indirekte Adressierung bei Maschinenbefehlen unterscheiden und den Zusammenhang mit imperativen Konstrukten erklären können.
- ◦ Die Rolle des Stapelspeichers im Kontext von Prozedurkonzept, Parametermechanismen und rekursiven Aufrufen aufzeigen und an einfachen Beispielen nachvollziehen können.
- ◦ Ausgewählte imperative Programmfragmente auf Maschinenebene implementieren und ihren Ablauf verfolgen können.

Automaten und formale Sprachen

1. Algorithmusbegriff

Lerninhalte:

Naiver Algorithmusbegriff.

Lernziele:

- Die Grundeigenschaften eines Algorithmus benennen können.
- Umgangssprachlich formulierte Handlungsanweisungen auf ihre algorithmischen Aspekte hin untersuchen können.
- Die Forderung nach genügender Präzision bei umgangssprachlichen Formulierungen von Algorithmen begründet befürworten können.

2. Berechenbarkeit

Lerninhalte:

Maschinenmodelle, Berechenbarkeitsbegriff.

Lernziele:

- Die Semantik der Grundkonstrukte der Sprache eines geeigneten Maschinenmodells (z. B. Registermaschine, Turingmaschine) wiedergeben können.
- Ausgewählte kleine Probleme (z. B. Zählprozesse, Kopieren von Variablenwerten, arithmetische Grundoperationen) unter Einsatz dieser Sprache algorithmisch lösen können.
- ◦ Komplexere Probleme (z. B. Wertvergleiche, Berechnungen aufwendigerer arithmetischer Ausdrücke) in der Sprache des eingesetzten Maschinenmodells lösen und die Korrektheit der Lösungen begründen
- ◦ oder Entwürfe für komfortable Erweiterungen der Sprache des Modells entwickeln und zur Lösung geeigneter Probleme anwenden können.
- Aus dem Modell einen Berechenbarkeitsbegriff ableiten und den Zusammenhang zum naiven Algorithmusbegriff herstellen können.
- Den Grundgedanken der Church'schen These wiedergeben können.
- ◦ Einen alternativen Berechenbarkeitsbegriff (z. B. `WHILE`-Berechenbarkeit, μ-Rekursion oder λ-Kalkül) skizzieren und Plausibilitätsbetrachtungen für deren Äquivalenz nachvollziehen können.

3. Endliche Automaten und reguläre Sprachen

Lerninhalte:

Grammatik- und Sprachbegriff, endliche Automaten.

Lernziele:

- • Den Grammatik- und Ableitungsbegriff und die von einer Grammatik erzeugte Sprache definieren und Beispiele angeben können.
- • Endliche Automaten durch Zustandsgraphen veranschaulichen und zur Beschreibung einfacher Konstrukte oder Abläufe einsetzen und
- • entscheiden können, ob ein Wort von einem gegebenen endlichen Automaten erkannt wird.
- • Zu einer regulären Grammatik einen (ggf. nicht-)deterministischen endlichen Automaten konstruieren können, der die von ihr erzeugte Sprache erkennt.
- • Reguläre Ausdrücke charakterisieren und (z. B. zur Suche von Mustern) anwenden können.
- ◦ Die Äquivalenz der Ausdrucksstärke der genannten Konzepte an geeigneten einfachen Beispielen plausibel machen können.
- ◦ Ausgewählte Probleme mit Turing-Maschinen lösen können.
- ◦ Sprachklassen charakterisieren und die Chomsky-Hierarchie entwickeln können.

4. Grenzen der Berechenbarkeit

Lerninhalte:

Existenz schwer lösbarer und nicht berechenbarer Probleme.

Lernziele:

- • Einen Kompexitätsbegriff formulieren und an Beispielen erläutern können, daß Algorithmen exponentieller Komplexität praktisch undurchführbar sind.
- • Eine heuristische Begründung dafür angeben können, daß es algorithmisch formulierbare Probleme gibt, die nicht berechenbar sind.
- ◦ Das Halteproblem formulieren und seine Unentscheidbarkeit auf einer einfachen Ebene nachvollziehen können.
- ◦ Ein schwer lösbares Problem (z. B. Rucksackproblem, Problem des Handlungsreisenden) nennen und plausibel machen können, daß es schwer lösbar ist.

Gesellschaftliche Aspekte der Informatik

1. Geschichte der Informatik

Lerninhalte:

Ausgewählte Aspekte der Entwicklung von Rechenmaschinen und Programmen.

Lernziele:

- Grundzüge der Funktionsweise eines historischen Rechenapparates erklären und dieses Beispiel in den Kontext seines historischen Umfeldes stellen *oder*
- ausgewählte Aspekte der technischen Entwicklung elektronischer Rechenmaschinen wiedergeben können.
- An einem historischen oder einem aktuellen Beispiel den Einfluß von Wissenschaft, Technik oder Wirtschaft auf die Entwicklung von Rechnern und Programmsystemen aufzeigen können.
- Einen groben Überblick über die Entwicklung des Preis-Leistungs-Verhältnisses von Mikrorechnern in den letzten zwei Jahrzehnten geben können.

2. Datenschutz

Lerninhalte:

Datenschutzrecht und informationelle Selbstbestimmung.

Lernziele:

- Den Begriff der „informationellen Selbstbestimmung" im Kontext der Rechtsprechung erläutern können.
- Grundbegriffe des Datenschutzes aus den Rechtsgrundlagen (z. B. Bundesdatenschutzgesetz, Schuldatenverordnung) extrahieren und an kleinen Fallstudien identifizieren können.
- Das Postulat nach angemessener Sorgfalt beim Umgang mit Daten anhand aktueller Szenarios vertreten können.
- Rechtskonzepte in Entwürfen für geeignete Datenstrukturen (z. B. einen Datensatz „Schülerin") modellieren können.

Softwareprojekt

1. Der Lebenszyklus komplexer Programme

Lerninhalte:

Programmlebenszyklus, Prinzipien der Programmierung komplexer Systeme, Bedeutung der Projektdokumentation.

Lernziele:

- Ein typisches Modell eines Programmlebenszyklus darstellen, seine Phasen voneinander abgrenzen und ihre logischen Abhängigkeiten erläutern können.
- „Programmieren“ als ganzheitliches Konzept der systematischen Bearbeitung aller Phasen definieren können.
- Die Bedeutung einer vollständigen und widerspruchsfreien Dokumentation der Ergebnisse jeder Phase als Arbeitsgrundlage für die jeweils nächste Phase darlegen können.
- Schnittstellen systematisch entwickeln und nutzen können.
- Eigene Interessen gemeinsamen Zielen unterordnen und dabei ggf. projektbedingte Spannungsfelder kooperativ abbauen können.

2. Systemanalyse

Lerninhalte:

Untersuchung und Beschreibung des Ist-Zustandes eines Systems.

Lernziele:

- Datenflüsse und funktionelle Abläufe in einem System angemessener Komplexität modellhaft erfassen und charakteristische Objekte herauspräparieren können.
- Geeignete Sprachebenen zur Bewältigung von Diskrepanzen zwischen Sacherfordernissen und informatischen Möglichkeiten finden können.
- Die Möglichkeit von Rückwirkungen des Rechnereinsatzes auf ein System in dessen Analyse einbeziehen können.
- Die Ergebnisse der Analyse zur Eingrenzung und Präzisierung der Aufgabenstellung verwerten können.
- ◦ Schwierigkeitsgrad und Zeitbedarf der Programmierung komplexer Systeme realistisch einschätzen können.

3. Anforderungsdefinition

Lerninhalte:

Ein-/Ausgabeverhalten von Systemen.

Lernziele:

- Die Benutzeroberfläche als Gesamtheit der Interaktionen zwischen Benutzern und Rechnerperipherie definieren können.
- Bildschirm- und ggf. Druck- und Dateiformate von Objekten sowie typische Teile einer Systemsteuerung sorgfältig beschreiben
- und dabei weitestgehend von den technischen Rahmenbedingungen der vorgesehenen Zielplattform abstrahieren können.
- Ausgewählte Teile einer Anforderungsdefinition als Grundlage für die Spezifikation des geplanten Systems und als Benutzerhandbuch niederlegen können.
- ◦ Ergonomische Aspekte bei der Gestaltung einer Benutzeroberfläche einbeziehen können.

4. Spezifikation

Lerninhalte:

Komponentenbegriff, Identifikation und Spezifikation von Komponenten, Prinzipien des Entwurfs einer Systemarchitektur.

Lernziele:

- Sachgerechte Kriterien an die Zerlegung eines Systems in möglichst orthogonale Komponenten als Kapseln für Entwurfsentscheidungen benennen und daraus folgern können, daß die Komponenten nur über ihre Schnittstellen kommunizieren dürfen.
- Die Eignung programmiersprachlicher Konzepte danach bewerten können, inwieweit sie textuelle Trennung und getrennte Übersetzbarkeit von Spezifikation und Implementierung unterstützen.
- In der Anforderungsdefinition ermittelte Objekte und Zugriffe auf sie als Komponenten identifizieren und daraus ihre Spezifikation (z. B. als abstrakte Datentypen oder -objekte) entwickeln können.
- Komponenten nach strukturellen Abhängigkeiten ihrer Attribute hierarchisieren und daraus eine Systemarchitektur ableiten können.
- ◦ Nachweisen können, daß ein objektbasiertes Entwurfsraster sowohl die Verständlichkeit der Spezifikationen der Komponenten als auch ihre Wartbarkeit und Wiederverwendbarkeit sicherstellt.

5. Implementierung

Lerninhalte:

Implementierungen abstrakter Datentypen/-objekte, Nutzung von Schnittstellen, Test, Systemintegration.

Lernziele:

- Die Repräsentationen der abstrakten Datentypen bzw. -objekte in Komponenten in problemangemessenem Feinheitsgrad modellieren und die spezifizierten Zugriffsoperationen implementieren und dabei die Schnittstellen anderer Komponenten nutzen können.
- Implementierungen von Komponenten in kleine Testumgebungen einbetten und umfassend gegen ihre Spezifikation testen können.
- Aufgedeckte Unstimmigkeiten nicht durch eigenmächtige Interpretationen, sondern durch komponentenübergreifende Korrektur von Fehlern früherer Phasen im Projektteam beseitigen können.
- ◦ Die Qualität der Arbeitsergebnisse der einzelnen Phasen anhand der Systemintegration der implementierten Komponenten beurteilen können.

6. Wartung und Risiken komplexer Systeme

Lerninhalte:

Revision des Programmlebenszyklus, (In-)Stabilitäten komplexer Programmsysteme.

Lernziele:

- Aus der Benutzung des Systems Verbesserungs- und Erweiterungsvorschläge erarbeiten und mit einer Vertiefung der Systemanalyse einen erneuten Durchlauf der Phasen des Programmlebenszyklus initiieren können.
- Ansätze für eine Revision des Systems durch Verfeinerung seiner Strukturen und Operationen entwickeln können.
- Ein Programmsystem auf adäquate Invarianz gegen die Wirkungen gezielter Veränderungen untersuchen und daraus auf die Qualität seines Entwurfs und seiner Konstruktion schließen können.
- ◦ Die mit allen Problemen menschlicher Kommunikation behaftete Transformation von Sachanalysen in formal korrekte Programmtexte als maßgeblichen Faktor grundsätzlicher Risiken komplexer Programmsysteme ausmachen können.

Nichtsequentielle Programmierung

Vorbemerkung

Verbindlich sind die Themenkreise 1 und wahlweise 3 oder 4, für Leistungskurse zusätzlich 2. Darüberhinaus sollte – in Grundkursen wenigstens in Ansätzen – ein weiterer Themenkreis behandelt werden; empfohlen wird die Auswahl aus 5 oder 6.

1. Nebenläufigkeit und Prozessbegriff

Lerninhalte:

Grundbegriffe der Nebenläufigkeit, Prozeßbegriffe, Konflikte beim Zugriff auf gemeinsame Daten, Sperrsynchronisation.

Lernziele:

- Die Begriffe „determiniert“, „deterministisch“ und „sequentiell“ definieren und ihre logischen Abhängigkeiten darlegen können.
- Anwendungsklassen, bei denen Nebenläufigkeit eine Rolle spielt, nennen und diese Rolle herausarbeiten können.
- Mit den Grundkonstrukten der verwendeten Programmiersprache zum Aufruf nichtsequentieller Anweisungsfolgen umgehen und den statischen Prozeßbegriff vom Prozedurbegriff abgrenzen können.
- Den dynamischen Prozeßbegriff erläutern, die wichtigsten Prozeßzustände nennen und ihre Übergänge charakterisieren können.
- An einfachen Beispielen typische Lese-Schreib-Konflikte bei nebenläufigen Zugriffen aufzeigen können.
- Die Forderung nach der Unteilbarkeit gewisser Anweisungsfolgen als Methode zur Lösung solcher Probleme begründen und daraus die Begriffe „kritischer Abschnitt“ und „gegenseitiger Ausschluß“ ableiten können.
- Verklemmungen beschreiben und die Notwendigkeit ihrer Vermeidung rechtfertigen können.
- Das Konzept der Sperrsynchronisation durch Angabe von Eigenschaften von Sperrprotokollen herausarbeiten können.

2. Betriebssystemaspekte

Lerninhalte:

Prozeßverwaltung, Unterbrechungen, Prozeßwechsel.

Lernziele:

- ◦ Grundaufgaben einer Prozeßverwaltung schildern können.
- ◦ Den Zuteiler als besonderen Prozeß zur Prozeßverwaltung charakterisieren und seine Aufgaben beschreiben können.
- ◦ Die Rolle von Unterbrechungsbehandlungen für die Prozeßverwaltung erklären können.
- ◦ Prozeßwechsel in das Modell des v. Neumann-Rechners einordnen können.
- ◦ Grundideen der Implementierung einer Prozeßverwaltung skizzieren können.

3. Schlossvariable

Lerninhalte:

Sperrprotokolle auf Maschinenebene, Schloßalgorithmen.

Lernziele:

- • Die Arbeitsweise von Sperrprotokollen mit Schloßvariablen erläutern können.
- • Eine Schloßvariable als abstraktes Datenobjekt spezifizieren können.
- • Die Spezifikation eines geeigneten Maschinenbefehls wiedergeben, zur Implementierung eines Sperrprotokolls verwenden und deren Korrektheit begründen können.
- • Lösungsansätze für Schloßalgorithmen auf korrekte Funktionsweise überprüfen können.
- • Zu einem vorgelegten einfachen Schloßalgorithmus für zwei Prozesse begründen können, daß er gegenseitigen Ausschluß garantiert und verklemmungsfrei ist.
- • Die Verwendung von Maschinenbefehlen und Schloßalgorithmen vergleichend bewerten und das Prinzip des aktiven Wartens problematisieren können.
- ◦ Ansätze zur Verallgemeinerung von Schloßalgorithmen auf mehrere Prozesse nachvollziehen können.

4. Semaphore

Lerninhalte:

Binäre und allgemeine Semaphore, Anwendungen.

Lernziele:

- Die Grundidee der Implementierung von Sperrprotokollen mittels binärer Semaphore formulieren können.
- Semaphore als abstrakte Datentypen spezifizieren können.
- Das Konzept binärer Semaphore auf allgemeine Semaphore erweitern können.
- Spezifische Konflikte beim Zugriff auf gemeinsame Daten in einfachen Anwendungen (z. B. beschränkter und unbeschränkter Puffer, erstes Leser-Schreiber-Problem, Links-Rechts-Problem) herausarbeiten, die Probleme mit Hilfe von Semaphoren lösen und die Korrektheit der Lösungen begründen können.
- Die Verwendung von Semaphoren zur Lösung von Nebenläufigkeitsproblemen kritisch reflexieren können.
- ◦ Komplexere Anwendungen (z. B. zweites Leser-Schreiber-Problem, Problem der speisenden Philosophen, Zuteilung von Ressourcen, Barrierensynchronisation) mit Hilfe von Semaphoren entwickeln können.
- ◦ Ansätze zur Implementierung von Semaphoren unter Rückgriff auf die Prozeßverwaltung skizzieren können.

5. Monitore

Lerninhalte:

Monitorkonzept, Bedingungsvariable, Anwendungen.

Lernziele:

- Monitore als abstrakte Datenobjekte mit impliziten Synchronisationseigenschaften charakterisieren und zur Lösung unbedingter Synchronisationsprobleme verwenden können.
- Bedingungsvariable mit Operationen zum Blockieren und Signalisieren spezifizieren können.
- Monitore mit Bedingungsvariablen zur Lösung bedingter Synchronisationsprobleme (z. B. Sperrsynchronisation, elementare Ressourcenverwaltung) konstruieren, insbesondere den semantischen Bezug

zwischen Bedingungsvariablen und booleschen Ausdrücken herstellen können.
- Spezifische Konflikte beim Zugriff auf gemeinsame Daten in einfachen Anwendungen (z. B. beschränkter und unbeschränkter Puffer, erstes Leser-Schreiber-Problem, Links-Rechts-Problem) herausarbeiten, die Probleme mit Hilfe von Monitoren lösen und die Korrektheit der Lösungen begründen können.
- ○ Komplexere Anwendungen (z. B. zweites Leser-Schreiber-Problem, Problem der speisenden Philosophen, Zuteilung von Ressourcen, Barrierensynchronisation) unter Einsatz von Monitoren entwickeln können.
- ○ Ansätze zur Realisierung von Monitoroperationen auf der Ebene der Prozeßverwaltung skizzieren können.

6. Botschaftenaustausch

Lerninhalte:

Kanalkonzept, bewachtes selektives Warten, Anwendungen.

Lernziele:

- Kanäle als abstrakte Datentypen mit Sende- und Empfangsoperationen spezifizieren und als Mittel zur Synchronisation in verteilten Systemen charakterisieren können.
- Asynchronen Botschaftenaustausch als Rendezvous charakterisieren und zur Konstruktion einfacher Filter anwenden können.
- Die Semantik des bewachten selektiven Empfangens von Botschaften erläutern und als Lösungsschema für die Kunden-Anbieter-Architektur einsetzen können.
- Spezifische Konflikte beim Zugriff auf gemeinsame Daten in einfachen Anwendungen (z. B. beschränkter und unbeschränkter Puffer, erstes Leser-Schreiber-Problem, Links-Rechts-Problem) herausarbeiten, die Probleme mit Hilfe von Botschaftenaustausch lösen und die Korrektheit der Lösungen begründen können.
- ○ Komplexere Anwendungen (z. B. zweites Leser-Schreiber-Problem, Problem der speisenden Philosophen, Zuteilung von Ressourcen, Barrierensynchronisation) auf der Basis von Botschaftenaustausch entwickeln können.
- ○ Bezüge zu einem Synchronisationsmittel herstellen können, das gemeinsamen Speicher voraussetzt.