Software Engineering I

Musterlösungen zur Hauptklausur vom 05.08.2000

Aufgabe 1

a) Abb. 1.1 zeigt ein ER-Diagramm, das zur Beantwortung der Anfragen in der Aufgabenstellung ausreicht:

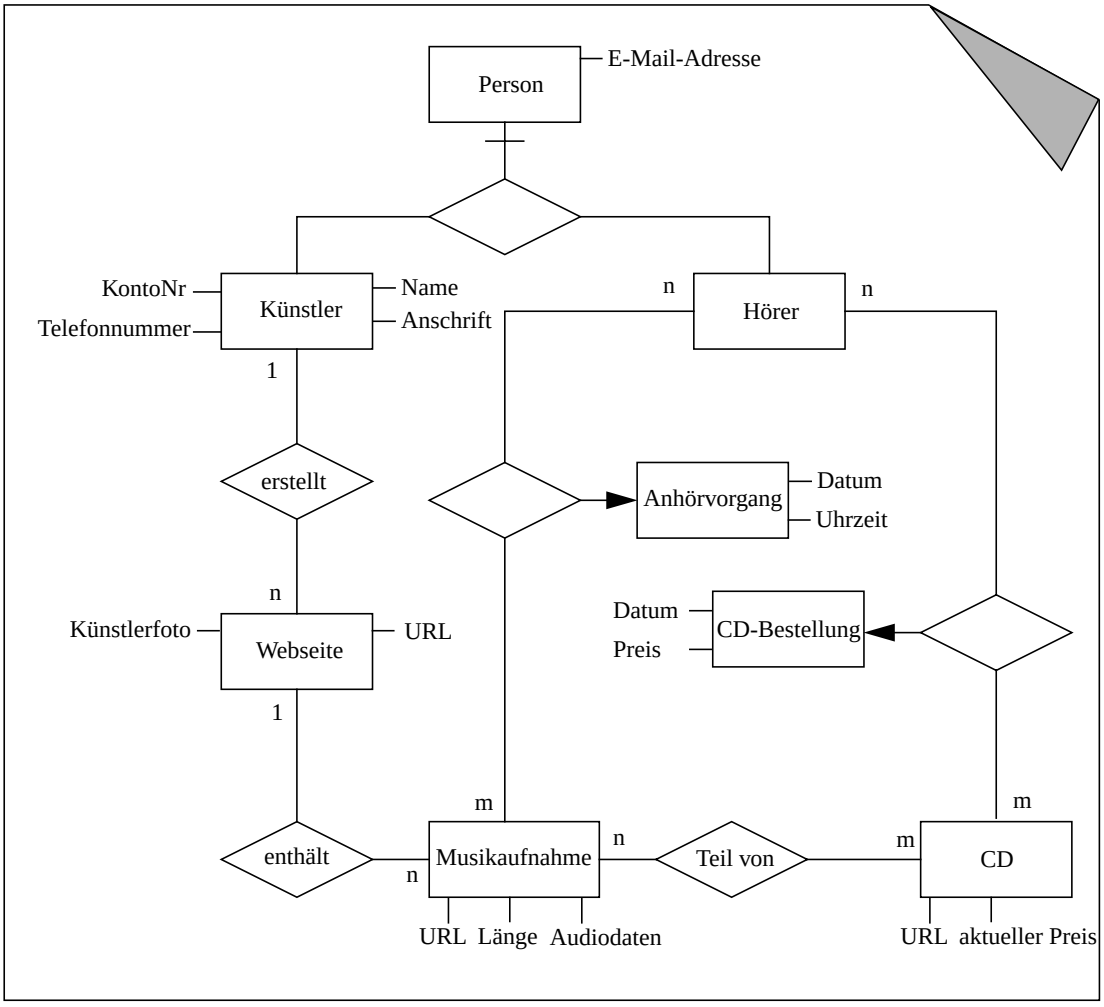

Abb. 1.1: Das ER-Diagramm des E-Commerce-Unternehmens

Aufgabe 2

a) Da die Entitäten „*Schulangehöriger*“, „*Abiturient*“ und „*Lehrer*“ analog zu Abb. 15.20 im Kurstext nur in ein einziges Teilsystem Schulangehörige transformiert werden, besteht die durch die Transformation entstehende modulare Architektur aus drei Teilsystemen für Entitäten, drei Teilsystemen für assoziative Beziehungen und vier Funktionsmoduln zur Verwaltung der vier Beziehungen:

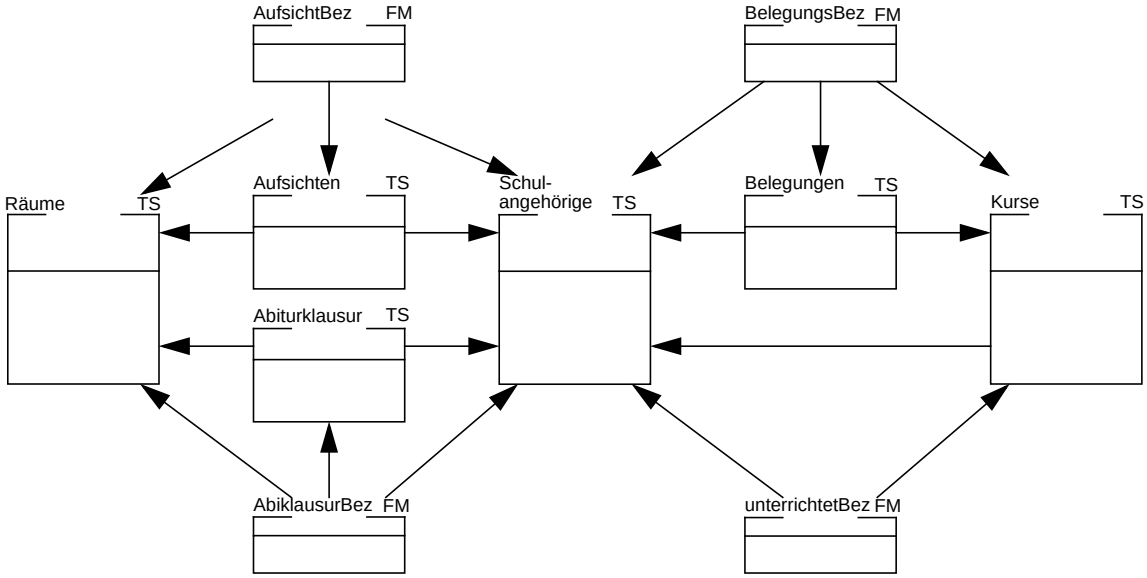

Abb. 2.1: Die durch Transformation des ER-Modells entstehende Architektur (1. Version)

b) Bei der zweiten Variante gibt es statt des Teilsystems Schulangehörige die Teilsysteme Abiturienten und Lehrer:

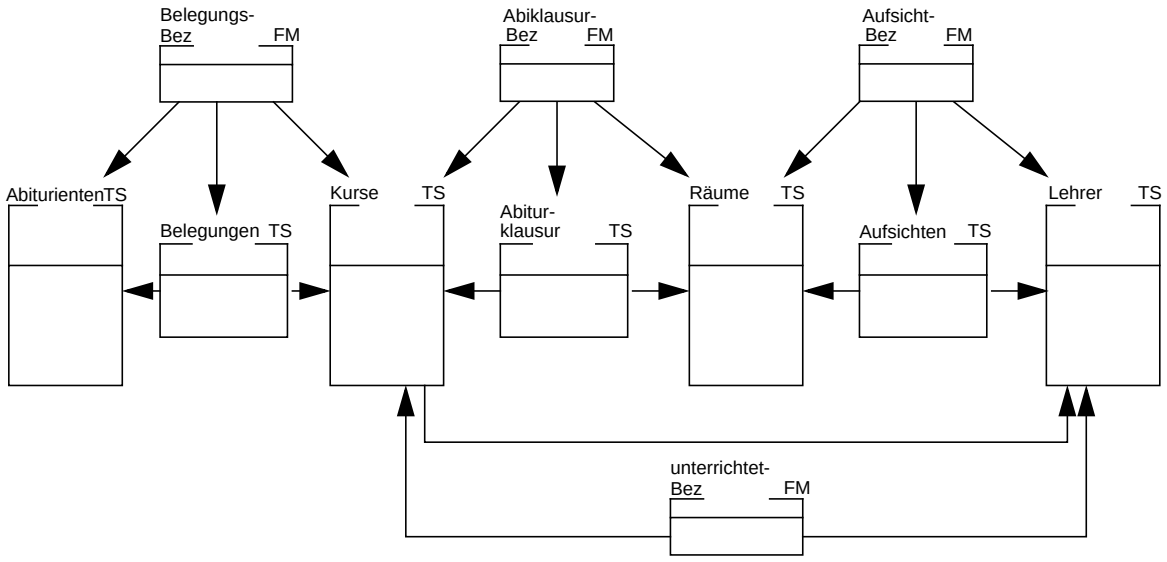

Abb. 2.2: Die durch Transformation des ER-Modells entstehende Architektur (2. Version)

c) Die erste Variante hat den Vorteil der geringeren Redundanz, denn wenn z.B. von Schulangehörigen zusätzlich noch die E-Mail-Adresse gespeichert werden soll, so sind die erforderlichen Änderungen auf das Teilsystem Schulangehörige beschränkt.

Die zweite Variante hat den Vorteil, dass die inhaltlich stark verschiedenen Entitäten „*Abiturient*“ und „*Lehrer*“ auch in der Architektur deutlich separiert sind. Dadurch wird auch wesentlich klarer, mit welchen Entitäten die verschiedenen Beziehungen verbunden sind.

In diesem Fall ist die zweite Variante vorzuziehen, da die klarere Architektur den Nachteil der höheren Redundanz überwiegt. Wenn die Entitäten „*Abiturient*“ und „*Lehrer*“ jedoch wesentlich mehr gemeinsame Eigenschaften — z.B. gemeinsam verwendete Typen oder Beziehungen zu anderen Entitäten — besitzen würden, oder wenn es Operationen gäbe, die auf allen Schulangehörigen (Lehrern und Abiturienten) arbeiten, dann könnte der Vorteil der Redundanzvermeidung überwiegen, was für die erste Variante spräche.

Aufgabe 3 (7 + 15 Punkte)

a) Die Operationen des ADO-Moduls Berechtigung unterscheiden sich von den Operationen des ADT-Moduls Berechtigungsschablone dadurch, dass der Parameter ZB vom Typ TBerechtigung wegfällt. Statt dessen wird ein „Modulgedächtnis“ in Form der Variablen BerechtigungsschablonеDS verwendet.

```
MODULSPEZIFIKATION ADO Berechtigung;
  SCHNITTSTELLENSPEZIFIKATION
    IMPORTE
        IMPORTIERE TKontoNr, TKundenID;
    EXPORTE
      DATENTYPEN
        TKontoNr, TKundenID;
      OPERATIONEN
        Create,
        IsFull: BOOLEAN,
        IsEmpty: BOOLEAN,
        Einfügen (IN KtoNr : TKontoNr; KID : TKundenID),
        Entfernen (IN KtoNr : TKontoNr; KID : TKundenID),
        IstBerechtigt (KtoNr : TKontoNr; KID : TKundenID): BOOLEAN,
        ErsterBerechtigter (IN KtoNr : TKontoNr; OUT KID : TKundenID) : BOOLEAN,
        NächsterBerechtigter (IN KtoNr : TKontoNr; INOUT KID : TKundenID) : BOOLEAN,
        ErstesVerfügbaresKonto (IN KID : TKundenID; OUT KtoNr : TKontoNr) : BOOLEAN,
        NächstesVerfügbaresKonto (IN KID : TKundenID; INOUT KtoNr : TKontoNr)
                                        : BOOLEAN;
    SEMANTIK
        Die Semantik der Operationen kann von den entsprechenden
        Operationen des ADT-Modul Berechtigungsschablone übertragen werden.
(*****************************************************************************)
  RUMPFSPEZIFIKATION
    IMPORTE
        IMPORTIERE Berechtigungsschablone;
    VAR BerechtigungsschablonеDS: Berechtigungsschablone.TBerechtigungsschablone;
    Prozessspezifikation Create
      BerechtigungsschablonеDS := Berechtigungsschablone.Create
    END Create;
    Prozessspezifikation NächstesVerfügbaresKonto
    Parameter:KID : TKundenID; (* -- IN *)
              KtoNr : TKontoNr; (* -- INOUT *)
      RETURN Berechtigungsschablone.NächstesVerfügbaresKonto
                                    (BerechtigungsschablonеDS, KID, KtoNr)
    END NächstesVerfügbaresKonto;
SPEZIFIKATIONSENDE MODUL Berechtigung.
```

b) Die Schnittstellenspezifikation entspricht der der Objektliste, jedoch muß bei der Delete-Operation das zu löschende Objekt als IN-Parameter übergeben werden:

```
MODULSPEZIFIKATION FM Berechtigungsverwaltung;

...
  RUMPFSPEZIFIKATION

    IMPORTE
      AUS Kunden IMPORTIERE
          TKundenID, ErsterKunde, NächsterKunde, DisposeKundenID;
      AUS Konten IMPORTIERE
          TKontoNr, ErstesVerfügbaresKonto, DisposeKtoNr;

  Prozessspezifikation AnzahlKundenOhneBerechtigung
    lokale Variable:  KID : TKundenID,
                      KtoNr : TKontoNr,
                      weitersuchen : BOOLEAN,
                      KontoVorhanden : BOOLEAN,
                      Anzahl : CARDINAL;
    Anzahl := 0
    weitersuchen := ErsterKunde(KID)
    WHILE weitersuchen DO
      KontoVorhanden := ErstesVerfügbaresKonto(KID, KtoNr)
      IF NOT KontoVorhanden THEN
        Anzahl := Anzahl + 1
      ELSE
        DisposeKtoNr(KtoNr)
      END
      DisposeKundenID(KID)
      weitersuchen := NächsterKunde(KID)
    END
    DisposeKundenID(KID);
    RETURN Anzahl
  END LöscheKundenOhneBerechtigung

END MODULSPEZIFIKATION Berechtigungsverwaltung.
```

Aufgabe 4

a) Das naheliegende Kriterium zur Definition von Äquivalenzklassen ist (von der Ausgabe her betrachtet) die Anzahl der Nullstellen, bzw. (von der Eingabe her betrachtet) der Wert der Determinanten $(p/2)^2-q$ (positiv, Null, oder negativ). Dementsprechend definieren wir die folgenden drei Äquivalenzklassen:

Klasse 1: $(inP/2)^2-inQ > 0$ bzw. `outAnzahl = 2`

Klasse 2: $(inP/2)^2-inQ = 0$ bzw. `outAnzahl = 1`

Klasse 3: $(inP/2)^2-inQ < 0$ bzw. `outAnzahl = 0`

b)

Klasse 1: Eingabe: (`inP=20, inQ=101`)
Ausgabe: (`outAnzahl=2`,
`outErsteNullstelle=-11`,
`outZweiteNullstelle=-9`)

Klasse 2: Eingabe: (`inP=20, inQ=100`)
Ausgabe: (`outAnzahl=1`,
`outErsteNullstelle=-10`,
`outZweiteNullstelle=0`)

Klasse 3: Eingabe: (`inP=20, inQ=99`)
Ausgabe: (`outAnzahl=0`,
`outErsteNullstelle=0`,
`outZweiteNullstelle=0`)

c) Die gewöhnliche Implementierung der Operation `NullstellenBestimmen` sieht wie folgt aus:

```
PROCEDURE NullstellenBerechnen(inP: REAL;
                               inQ: REAL;
                               outAnzahl: CARDINAL;
                               outErsteNullstelle: REAL;
                               outZweiteNullstelle: REAL);
(* berechnet die Nullstellen der quadratischen Gleichung x^2+px+q=0
   in Abhängigkeit von p und q.
   outAnzahl enthält die Anzahl der Nullstellen.
   outErsteNullstelle enthält die erste Nullstelle, falls es mind. eine gibt,
   sonst 0.
   outZweiteNullstelle enthält die zweite Nullstelle, falls es zwei gibt,
   sonst 0. *)

VAR
      Determinante: REAL;
```

```
BEGIN
   Determinante = -inP*inP/4.0 - inQ;
   IF Determinante > 0 THEN
      OutAnzahl := 2;
      OutErsteNullstelle := -inP/2.0 + sqrt(Determinante);
      OutZweiteNullstelle := -inP/2.0 - sqrt(Determinante);
   ELSIF Determinante = 0 THEN
      OutAnzahl := 1;
      OutErsteNullstelle := -inP/2;
      OutZweiteNullstelle := 0.0;
   ELSE
      OutAnzahl := 0;
      OutErsteNullstelle := 0.0;
      OutZweiteNullstelle := 0.0;
END;
```

Als Kontrollflussgraph ergibt sich entsprechend den 3 Fallunterscheidungen ein „3-Wege-Kontrollflussgraph“ wie z.B. der in Abb. 4.1:

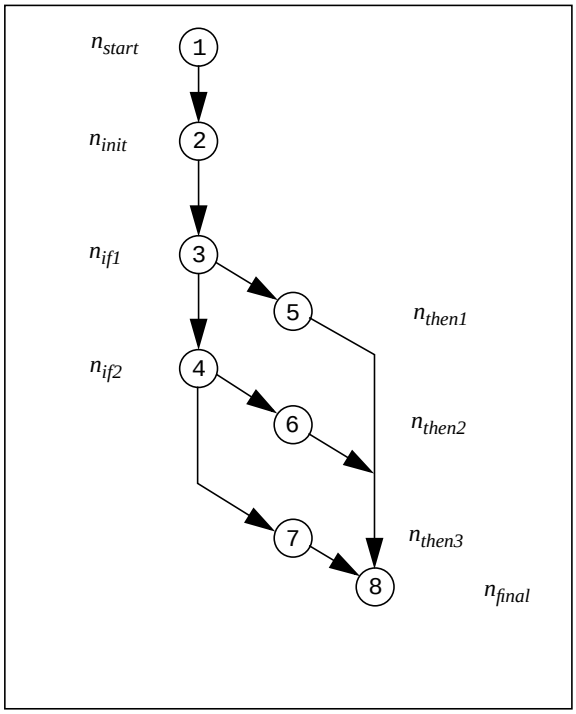

Abb. 4.1: Kontrollfluißgraph der Prozedur NullstellenBestimmen

d) Die unter b) ermittelten Testdaten reichen für Anweisungs- und Zweigüberdeckung aus, da von jedem Testdatum jeweils einer der drei möglichen Pfade durchlaufen werden:

Eingabe: (`inP=20, inQ=101`)
Knotenfolge: 1, 2, 3, 5, 6, 7, 8

Eingabe: (`inP=20, inQ=100`)
Knotenfolge: 1, 2, 3, 4, 6, 7, 8

Eingabe: `(inP=20, inQ=101)`
Knotenfolge: 1, 2, 3, 4, 7, 8

Damit sind sämtliche Knoten und Kanten durchlaufen.

e) Das Testeingabedatum (inP=20000.0, inQ=100000001.0) führt zu einem Fehler durch Rechenungenauigkeit. Rechnet man dezimal mit 8 Stellen Genauigkeit, so ergeben sich folgende Werte:

inP = $2.0000000 * 10^5$
inQ = $1.0000000 * 10^8$ (Rundungsfehler)
Determinante = $1.0000000.0 * 10^8 - 1.0000000.0 * 10^8 = 0.0000000$ (Folgefehler)
outAnzahl = 1 (falsch)

Anmerkung:

Da Computer in Wirklichkeit binär rechnen, kann es sogar zu unerwarteten Rundungsfehlern kommen: Da z.B. die Zahl 0.2 im Binärsystem unendlich viele Nachkommastellen hat, entstehen Rundungsfehler, die im Dezimalsystem nicht passieren würden. Zur Entdeckung solcher Fehler sind funktionale Tests und Kontrollflusstests nicht ausreichend.