Software Engineering

Informatik II.

7. Software-Entwicklung

– Realisierung –

Dipl.-Inform. Hartmut Petters

Vorwort – was ich noch zu sagen hätte ...

Basis dieser Vorlesung sind vor allem die folgenden Ausarbeitungen

- Vorlesungsskript „Software Engineering“
  von Prof. Dr. Martin Glinz Universität Zürich
  http://www.ifi.unizh.ch/groups/req/courses/ses/
- Vorlesungsskript „Informatik II – Software Engineering“
  von Frau Prof. Dr. Kühn FH Karlsruhe FB W
  http://www.home.fh-karlsruhe.de/~kuin0001/inhalt.htm
- Das Buch „Software Engineering“ 6. Auflage/2001
  von Prof. Ian Sommerville University of Lancaster (UK)
  Addison Wesley ISBN 3-8273-7001-9

Konkret entnommene Beiträge sind i.d.R. mit einem Quellen-Verweis gekennzeichnet – sollte dieser fehlen bitte ich um Nachsicht.

Den „roten Faden“ durch die Vorlesung habe ich dem Skript der Vorlesung von Prof. Dr. Martin Glinz entnommen und um eigene Beiträge erweitert bzw. aus den beiden anderen Quellen ergänzt.

Für die Möglichkeit der Verwendung der wesentlichen Inhalte möchte ich mich an dieser Stelle bei den Autoren herzlich bedanken.

Möglichkeiten der Realisierung

- Optionen
  - *Entwerfen* und *Codieren*
  - *Entwerfen* und *Generieren*
  - *Konfigurieren* vorhandener Komponenten
- Aber immer
  - *Integrieren* und *prüfen*
- Wenn möglich
  - *Vorhandene Komponenten* wiederverwenden

Detailentwurf + Codierung

- *Entwurf*
  - Entwurf des Programm-Ablaufs
    - Logische Abfolge der Befehle (schrittweise)
    - Logisches Ablaufdiagramm
  - Entwurf der Algorithmen + Datenstrukturen
  - Präzisierung des Lösungskonzepte
- *Codierung*
  - Umsetzung des Detailentwurfs in Programm-Code
  - Aufbau des Testbetts zur Überprüfung
  - Überprüfen + Verifizieren

Code-Generierung

- Entwurf
  - Erstellung eines *formalen* oder *teilformalen Entwurfsdokuments*
  - *Syntax* muss den Anforderungen des Code-Generators genügen
  - Dient als *Grundlage* für die Generierung
- Beispiele: Generierung von
  - *Datenbank-Schemata* (z.B. durch ARIS-Toolset)
  - *Einfache Anwendungen* (z.B. Symantec Café)
  - *Benutzeroberfläche* (z.B. Rational Rose)
  - *Dialogen*
  - *Codierungs-Rahmen* mit Übergabeparametern

Systematische Programmierung

- Geeignete Wahl von *Algorithmen* und *Datenstrukturen*
- Verwendung geschlossener Ablaufkonstrukte
- Saubere Gliederung der Software in Prozeduren und Module
- Verwendung *symbolischer Konstanten* und *abstrakter Datentypen*
- Verwendung *sprechender Bezeichnungen* + *selbstdokumentierende Namen* für Prozeduren, Variablen, Konstanten und Typen
- *Einhalten* vorgegebener *Programmierrichtlinien* für Formatierung, Namensgebung, Bezeichnungen + Abläufe
- Dokumentation aller Definitionen durch Kommentare
  - Datei-Header mit Beschreibung des logischen Ablaufs sowie der Ein- und Ausgabeparameter
  - Zeilenkommentare für logischen Blöcken + schwer verständliche Zeilen
  - Wirkung der Prozeduren bzw. eingesetzten Methoden
  - Bedeutung der Variablen + Konstanten
- *Defensives* Programmieren – lieber ausführlicher
- Vermeidung von *Trickprogrammierung* und Programmierung mit *Seiteneffekten*

Werkzeuge zur Programmierung

- Geschlossene Entwicklungsumgebung
  - Visual-C / C++
  - Visual-Java
  - Visua-Basic,
  - ...
- Einzelwerkzeuge

Editoren	Code-Erstellung
Übersetzer / Compiler	Zwischencode-Erzeugung (relativ übersetzt)
Linker	Programm-Erzeugung
Testumgebung	Laufzeit-Umgebung mit Testmöglichkeiten
Make-Utilities + Makefiles	Übersetzungshilfen mit Beschreibung der Abhängigkeiten

Prinzipieller Programm- / Funktions-Aufbau

- Header-Files — Funktionen / Deklarationen
  - Allgemeine Deklarationen
  - Makro-Definitionen
  - Globale Variable
- Bibliotheken — vorübersetzte Programmteile
  - Zugekaufte Funktionen
  - Funktionen anderer Team-Mitglieder
  - Eigene vorübersetzte Funktionen
- Deklarationen — Bekanntgabe wichtiger Elemente
  - Globale Variable / Funktionen
  - Lokale Variable / Funktionen
- Funktionsdeklarationen — Übergabe- / Ergebnis-Parameter
  - Funktionsart / Ergebnisart
  - Funktions-Parameter
- Programm- / Funktions-Rumpf — Funktionsprogrammierung
  - Codierung
  - Kommentare
  - Ergebnisübergabe

Programm – Funktion - Routine

- Begriffs-Festlegung
  - Programm
    selbständig lauffähiges Software-Produkt, dass alle Teilfunktionen enthält, so dass es direkt gestartet werden kann.
  - Funktion
    Teilprogramm, dass eine *bestimmte Funktion* ausübt und beim Aufruf die erforderlichen *Parameter übergeben* bekommt.
    Nicht ohne Testrahmen lauffähig.
  - Routine
    Teilprogramm, dass eine *definierte Funktion* ausführt, aber *keinerlei Parameter* benötigt, sondern nur Veränderungen in dem Programmkontext vornimmt.
    Nicht ohne Testrahmen lauffähig.
    Programmierung über *Seiteneffekte*!

Basis-Kenntnisse

- *Grundaufbau*
  - Bibliotheken
  - Deklarationsdateien
  - Hauptprogramm
  - Unterprogramme
- *Variable + Konstanten*
  - Lebensdauer
  - Gültigkeit
- *Funktionen / Routinen*
  - Aufgaben
  - Übergabewerte / Rückgabewerte
  - Datenbereiche / Datenstrukturen
  - Testumgebung / Testbett

Programmentwurf - Vorgehen

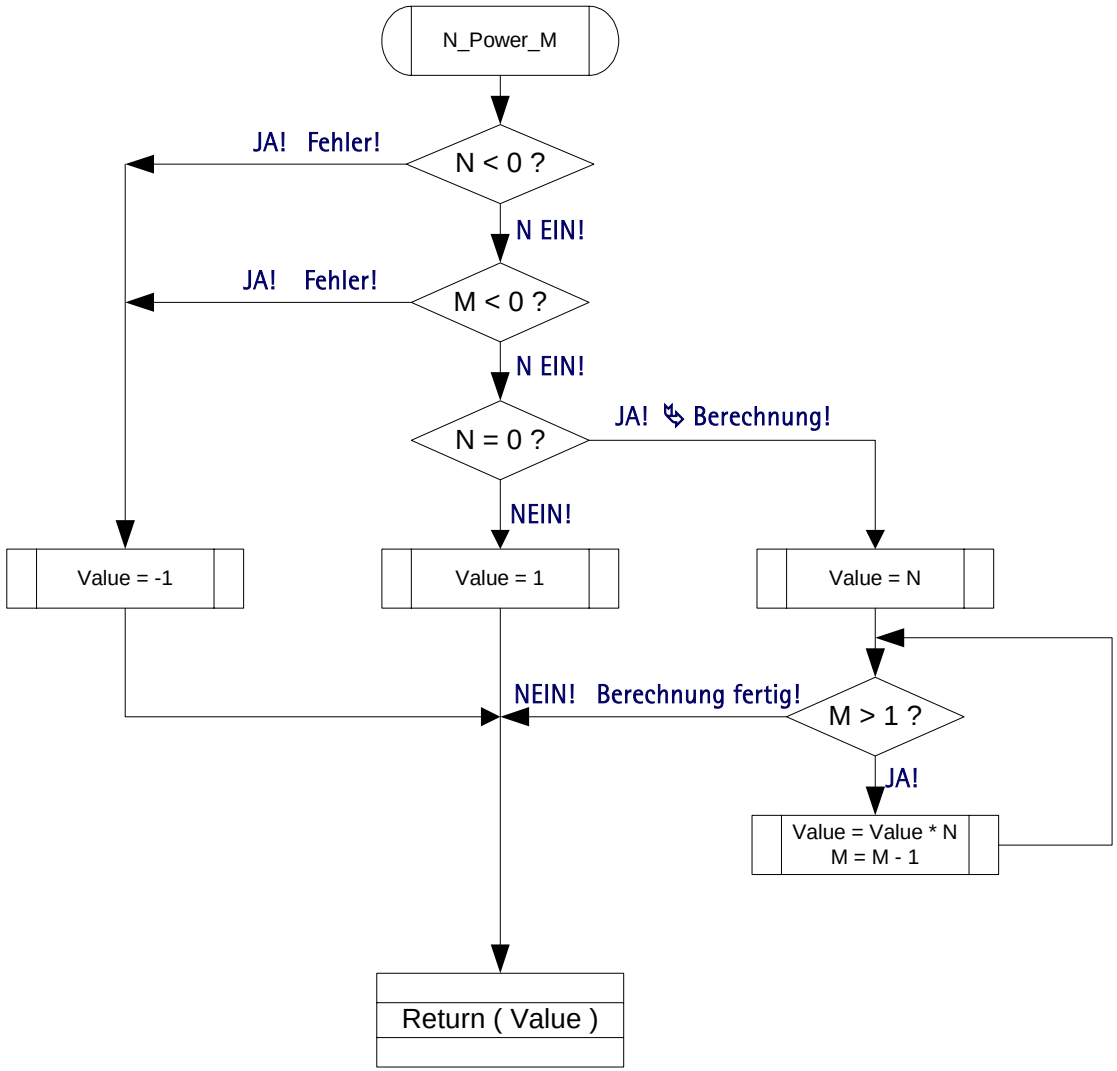
- Spezifikation der Aufgaben / Detail-Entwurf
- Darstellung des Ablaufs
  - Teilformal (Stichpunkte, Pseudo-Programmcode)
  - Ablaufdiagramme (Nassi-Schneidermann, o.ä.)
  - Datenstrukturen + Informationsfluss (SSAD, Entity-Relationship-Diagramme)
- Festlegung der Typen
  - Funktionsdeklarationen (void, int, char *, ...)
  - Übergabe-Parameter
  - Interne Variablen + Konstanten (Lebensdauer, Gültigkeit)
  - Rückgabewerte / Art der Rückgabe
- Basis-Dokumentation im Programm- / Funktionskopf
- Testrahmen
  - Einbettung in das „Main"-Programm
  - Testfälle (Standard, Sonderfälle, Eingaben, Ergebnisausgaben, ...)

Beispiel: n_power_m – (n^m)-Berechnung

- Funktion- und Variablen-Deklaration
  - 2 Übergabe-Parameter: n, m (Typ: Integer)
  - Ergebniswert der Funktion: Return-Wert (Typ: Integer)
  - Laufvariable: i (Typ: Integer)
  - Ergebnis-Variable: Ergebnis (Typ: Integer)
- Funktionsablauf
  - Prüfung der Übergabe-Parameter, ob negativ
    - JA: Fehler Ergebnis = -1
  - Sonderfälle
    - m = 0 Ergebnis = 1
    - n = 0 Ergebnis = 0
  - Reguläre Berechnung
    - Wahl der Bearbeitungsschleife
      - While ... Do
      - Do ... While
      - For ...
    - Ergebnis = n

```
for (i=1; i<m; i=i+1)        /* statt „i=i+1“ kann auch „i++“ geschrieben werden */
        {
        Ergebnis = Ergebnis * n;
}
return (Ergebnis);
```

Ablaufdiagramm – N-Power-M-Berechnung



Allgemeine Richtlinien

- Grundsätzlich bestehende *Programmier-Richtlinien* einhalten
- *Keine Verwendung* des Befehls *„goto"* und Sprungmarken
- *Keine Einsprünge* in Funktionen oder Routinen
- *Keine* Verwendung *„globaler Variablen"*
- Konstante, abstrakte Datentypen und Makros in *„Header-Files"* deklarieren
- Bearbeitungs-*Funktionen* eines *„abstrakten Datentyps" zusammenfassen* (Information Hidding)
- Möglichst nur *einen einzigen Rücksprung* aus einer Funktion
  - Einzige Ausnahme eventuell „Fehlerausgang"
- Möglichst *sprechende Namen* verwenden
- Möglichst den Code einer *Funktion nicht größer ale 1-2 A4-Seiten*
- Besser *ausführlicher programmieren* als komprimiert und unübersichtlich (keine Effizienzsteigerung, da Compiler optimiert)
- *Übergabe-Parameter*, allgemeinen *Ablauf* (Formel, etc.), *Ergebnisübergabe* etc. im Funktionskopf *beschreiben*
- Zeilenkommentare nur wo *sinnvoll* als *Block-Kommentar*

Beispiel: Datei-Header aus Java-Funktion

```
/*****************************************************************************
Autor:          Martin Arnold
Projekt:        Beispiel Vorlesung SW-Engineering
Sprache:        Symantec Café, Projectmanager 8.2b3
Copyright:      Forschungsgruppe Requirements Engineering
                Institut für Informatik, Universität Zürich
Klasse:         MeasurementList
Version:        1.01 von M. Glinz am 18.12.1996
-----------------------------------------------------------------
Zweck:
Die Objekte der Klasse MeasurementList modellieren Messreihen.

Verantwortlichkeiten
Speicherung, Skalierung, Filterung und Abfrage von Messreihen
*****************************************************************************/

Import java.awt.List;
```

Beispiel – Header einer einfachen Funktion

```
/*
 *      Funktion:           Zwei Text-Strings zusammenkopieren
 *      Autor:              Hartmut Petters
 *      Aufrufparameter:    Char * s1, char *s2          (Beispiel: S1=„ABC“, S2=„DEF“)
 *      Funktion:           2 Text-Strings zusammenkopieren     (Beispiel: S1:“ABCDEF“)
 *      Rückgabewert:       int anz                      Länge des neuen Strings
 *      letzte Änderung:            16.11.2003 hjp
 */

int cpy_2_to_1(char * s1, char * s2)
{
        char    *       p_s1;
        int             anz = 0;

        p_s1 = s1;
        while (*p_s1++)
                anz++;
        while (*s2)
        {
                *p_s1++ = *s2++;
                anz++;
        }
        *p_s1 = '\0';
```

Beispiel – Funktion „square"

```
/*
 *    function square
 *    Berechnet das Quadrat des übergebenen Wertes und liefert das Ergebnis
 *    als Return-Wert zurück
 */
      int      square ( num )                    /* function square     */

      int      num;                              /* formal parameter    */
      {
               int      result;
               int      value;

               result  = num * num;

               return ( result ) ;
      }
```

Beispiel - Testbett

```
#include    <stdio.h>                          /* I/O-header-file          */
#include    <stdlib.h>                         /* function-declarations    */

main ( int argc, char **argv )                 /* start-function           */
                                               /* argc = argument-count    */
                                               /* argv = arguments         */
{
      extern int      atoi();                  /* conversion ascii to int  */
      extern int      square();
      int             result;
      int             value;

      value  = atoi(*(argv+1));                /* read 1st argument        */
      result  = square(value);                 /* square-function-call     */

      printf("SQUARE ( %d ) = %d\n", value, result);   /* print result    */

      exit ( 0 ) ;
}
```

Beispiel - Trickprogrammierung

- Ein Unterprogramm in C zur Verkettung zweier Zeichenketten (Strings)

```
void strcat (char *s, char *t)
{
        while (*t++); for (t--;*t++=*s++;        /* t <- (t concat s)  */
}
```

- Ein Unterprogramm zum kopieren einer Zeichenkette

```
void strcpy (char *s, char *t)
{
        for (;*t++=*s++;)
}
```

- Besonderheiten dieser Art der Programmierung
  - Beide Male sind die eigentlichen *Schleifen leer*
  - Fragen hierzu
    - Macht diese Funktion / dieses Programm nichts?
    - Doch! Aber die *Funktionalität* ist in den Seiteneffekten versteckt!
  - *Elegant* + knapp zu *programmieren*
  - Schwer zu verstehen und zu prüfen!

Test + Integration

- Prüfung jeder Einzelkomponente
  - *Syntaktische* Richtigkeit (durch Compiler)
  - *Semantische / Inhaltliche* Richtigkeit
    - *Selbstinspektion* des Codes durch Autor(in)
    - *Formale Inspektion* durch Dritte
    - *Komponententest*
- Schrittweise Integration der Komponenten
  - *Aufwärts*- oder *Abwärtsintegration*
  - *Integrationstests*
- Systemtest
- Abnahme (Code-Inspektion / Code-Review)

Zusammenfassung

- Nur *kurzer Überblick* bzgl. Realisierung / Umsetzung
- Umsetzung *systematisch* angehen
  - Grundsätzliche *Gedanken zum Lösungsansatz*
  - Ablauf er Funktionen skizzieren / *Ablaufdiagramme*
  - *Übergabewerte + Ergebniswerte* festlegen
  - *Header-Beschreibung* mit Funktionsbeschreibung
  - *Codierung* der Funktionen (evtl. mit Testausgaben)
  - *Testbett* erstellen, um lauffähigen Rahmen zu schaffen
  - Einbettung der Funktion in *reale Modul-Umgebung*
- Jede *Integrations-Stufe* getrennt *testen*
- Endabnahme durch Dritte

Literatur – Software Engineering

- Skript Informatik II — Prof. Dr. Kühn / Fb W FH Karlsruhe
  http://www.home.fh-karlsruhe.de/~kuin0001/inhalt.htm
- Skript Software Engineering — Prof. Dr. Martin Glinz Universität Zürich
  http://www.ifi.unizh.ch/groups/req/courses/se_I/
- Skript Software Engineering II — Bernd Kahlbrandt FH Hamburg
  http://www.informatik.fh-hamburg.de/~khb/st4se2/
- Software Engineering
  Ian Sommerville — (ISBN3-82737-001-9)
- Software Engineering
  - Grundkurs für Praktiker –
  Roger S. Pressman — (ISBN 3-89028-163-X)
- Software Entwurf
  - Methoden und Werkzeuge –
  A. Schulz — (ISBN 3-486-21608-2)
- Software Engineering und Prototyping
  Thorsten Spitta — (ISBN 3-540-17542-3)
- CASE
  Helmut Balzert — (ISBN 3-411-03224-3)
- Software-Qualitätssicherung
  Ernest Wallmüller — (ISBN 3-446-15846-4)

Software Engineering

Informatik II.

8. Software-Entwicklung

– Qualitäts-Management –