Methoden zur Überprüfung der Ergebnisse

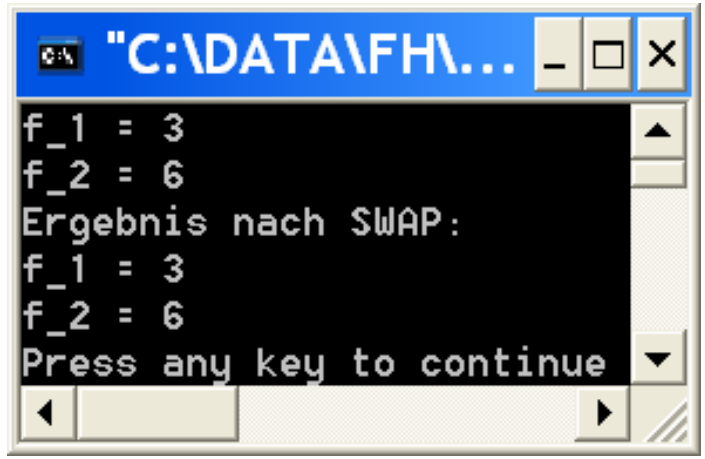
8. Reviews

von
Prof. Dr. Wolfgang Weber

([Wi05] S.17f, S.57f, S.297f, S.421f, S.489f,...)
Folien zur Vorlesung entstammen auch
Präsentationen von Prof. R. Hahn,
Prof. U. Andelfinger und Prof. G. Raffius

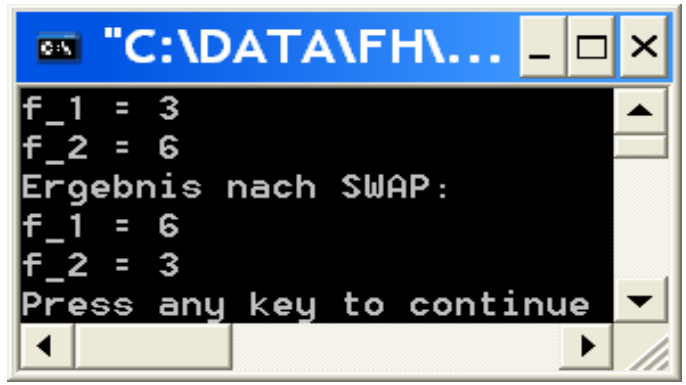
Quiz: Was tut das Programm?

```
#include <iostream.h>
void swap(float *p1, float *p2) {
  float *temp;         // Hilfsvariable
  temp = p1;
  p1   = p2;
  p2   = temp;
}
int main() {
         float f_1, f_2;
         f_1 = 3;   f_2 = 6;
         float *f1, *f2;
          f1 = &f_1;           f2 = &f_2;
          cout << "f_1 = " << *f1 << endl;
          cout << "f_2 = " << *f2 << endl;
          swap (f1, f2);
          cout << "Ergebnis nach SWAP: " << endl;
          cout << "f_1 = " << *f1 << endl;
          cout << "f_2 = " << *f2 << endl;
          return 0;
}
```



Quiz: Jetzt tut das Programm, was es soll ...

```
#include <iostream.h>
void swap(float *p1, float *p2) {
  float temp;          // Hilfsvariable
  temp = *p1;
  *p1  = *p2;
  *p2  = temp;
}
int main() {
         float f_1, f_2;
         f_1 = 3;   f_2 = 6;
         float *f1, *f2;
          f1 = &f_1;           f2 = &f_2;
          cout << "f_1 = " << *f1 << endl;
          cout << "f_2 = " << *f2 << endl;
          swap (f1, f2);
          cout << "Ergebnis nach SWAP: " << endl;
          cout << "f_1 = " << *f1 << endl;
          cout << "f_2 = " << *f2 << endl;
          return 0;
}
```



Quiz: Jetzt tut das Programm, was es soll ...

- Was müssen wir tun, um solche Fehler bei der Programmierung zu vermeiden?
- Was muss ich tun, um Fehler, die in jeder Phase der SW-Entwicklung gemacht werden können, zu vermeiden? (Fehler in frühren Phasen sind besonders teuer!) Beispiel: Absturz Jungfernflug ARIANE 5.
- Was muss ich tun, damit die SW nach der Fertigstellung den geforderte Qualitätsanforderungen entspricht?
- Wie halte ich den Aufwand zum Erreichen dieser Ziele möglichst gering?

Qualitätssicherung

Produktqualität / Prozessqualität:

- **Produktqualität**

  Erreichen der Qualitätsvorgaben durch Anwendung konstruktiver und analytischer Maßnahmen bei der Entwicklung des Produktes (siehe nächste Folien)

- **Prozessqualität**
  - Definition des Entwicklunsprozesses,
  - Finden von Schwachstellen im Entwicklungsprozess,
  - Systematische Verbesserung des Entwicklungsprozesses, so dass beim nächsten Projekt nicht wieder die gleichen Fehler gemacht werden.

  Norm ISO 9000; CMM, CMMI; SPICE

  (siehe Kapitel Prozessqualität)

Qualitätssicherung

Konstruktive QS / Analytische QS:

Konstruktive QS

- Es wird versucht , das Entstehen von Fehlern durch geeignete Maßnahmen während der Entwicklung zu verhindern, d. h.

  Qualität soll nicht nur nachträglich in ein Produkt hineingeprüft werden.

  - Einsatz von
    - Methoden (UML,...)
    - Konstruktionsprinzipien (OO-Entwurf, 4-Schichten-Architektur,...)
    - formalen Verfahren (formale Spezifikation, ...)
    - Werkzeugen (Case Tool, ...)
    - Vorgehensmodellen (Wasserfallmodell, V-Modell, ...)
    - Vorgehensrichtlinien (Einsatz von Prototypen, von Anwendern verstehbare Anforderungsdokumentation, ...)

Qualitätssicherung

Analytische QS

- **statische Maßnahmen**
  - **Validierung:** Are we building the right product? ->Abklärung mit Anwenderseite.
  - **Verifikation:** Are we building the product right? ->Prüfung durch Analytiker, Entwickler.

    z. B. Intra-Modell-Verifikation, Inter-Modell-Verifikation bei der Anfordserungsermittlung und dem Entwurf

    Es wird geprüft:
    - Vollständigkeit: Keine Lücken
    - Eindeutigkeit: Keine Missverständnisse möglich
    - Konsistenz: Keine widersprüchlichen (oder redundanten) Angaben

  Verfahren:
  - Manuelle Prüfverfahren (Reviews)
  - Anwendung von Werkzeuge zur Bestimmung/Erhöhung der Qualität
    - Konsistenz-Checks über dem UML-Modell mit Hilfe des CASE-Tools
    - Kompilierung des Codes zum Finden formaler Fehler (z. B. Syntaxfehler)
    - Codeanalyse mit Werkzeugen zur Bestimmung von Qualitäts-Maßen (siehe Kapitel Metriken)
  - mathematisches Beweisen der Korrektheit von Programmen (Formale Verifikation)
- **dynamische Maßnahmen**
  - Tests

Lernziel Manuelle Prüfverfahren

Sie sollen in diesen Kapitel verstehen,

- warum manuelle Prüfungen notwendig sind
- warum Reviews oft "diplomatische Missionen" sind
- was einen Review ausmacht und wie er abläuft
- dass es verschiedene manuelle Prüfverfahren gibt

Anschließend können Sie ein Review durchführen.

Warum manuelle Prüfmethoden?

- Prüfung von Syntax, Konsistenz und Vollständigkeit
  - werden oft von Werkzeugen automatisiert durchgeführt
- Prüfung der Semantik
  - ist nur manuell möglich!

⇨ Hauptziel der manuellen Prüfung

Was wird geprüft?

- Alle Ergebnisdokumente der einzelnen Phasen

  Frage: welche Phasen?

  (auch Zwischenergebnisse vor Phasenenden)

  z. B.
  - UML-Modelle
  - verbale Beschreibungen
  - Testpläne
  - HW- / SW-Architekturen
  - Code
  - Dokumentation
  - ...

Phasen einer manuellen Prüfung (eines Reviews)

0) Prüfanforderung
   - Autor stößt den Prüfprozess an
1) Vorbereitung
2) Sitzung
3) Nachbearbeitung

1) Vorbereitung

- Teilnehmer festlegen und deren Rollen
  - Rollen: Endbenutzer, fachverwandte Spezialisten, Ahnungslose,...
- Jeder Reviewteilnehmer bekommt folgende Unterlagen:
  - Prüfobjekt (z.B. OOA-Modell)
  - Ursprungsobjekt (z.B. Pflichtenheft)
  - Erstellungsregeln (z.B. Methode zur Erstellung eines OOA-Modells)
  - Fragenkatalog, Checkliste (z.B. "Wurde Regel xxx eingehalten?")
- Individuelle Vorbereitung
  - Jedes Mitglied arbeitet die Unterlagen individuell durch
  - Die Vorbereitung muss bis zur Sitzung abgeschlossen sein.
  - Jeder Prüfer sucht nach rollenspezifischen Defekten (z.B. als Endbenutzer, Wartungsspezialist, Gesamtsystem-Verantwortlicher).
  - Gefundene Defekte sind zu notieren.
  - Die empfohlene Arbeitsgeschwindigkeit ist zu beachten (ca. ein bis wenige Seiten pro Stunde – Qualität vor Quantität!).

2) Sitzung

- Ablauf der Sitzung
  - 3 bis 5 Teilnehmer
  - nicht unterbrechbare Sitzungen von jeweils 1 – 2 Stunden
  - Defekte werden dem Protokollführer laut gemeldet.
  - wie beim Brainstorming
    - ohne Diskussion der einzelnen Punkte
    - keine Lösungsvorschläge (evtl. nach Sitzung im kleinen Kreis diskutieren)
    - keine Kommentierung potenzieller Defekte
- Aussehen des Formulare und des Protokolls zum Review:

Formular zur Durchführung von Reviews

Datum		
Reviewdauer	Start:	Ende:
Moderator		
Prüfobjekt		
Referenzunterlagen		
Reviewprotokoll		

Anz. schwere Defekte (S)	Anz. leichte Defekte (L)	Anz. Verbesserungs-vorschläge (V)	Anz. Fragen (F) an den Autor	Anz. neue Defekte (D) in der Sitzung entdeckt
Reviewergebnis				

Reviewprotokoll

Lfd. Nr.	**Klassifikation**	**Beschreibung**	**Relevante Regel a. Checkliste**
1.			
2.			

3) Überarbeitung

- Der Autor überarbeitet das Prüfobjekt
  - Änderungen werden an Hand des Protokolls eingearbeitet.
  - Im Protokoll wird notiert, welche Aktionen pro Protokolleintrag unternommen wurden.
  - Evtl. müssen Änderungsanträge gestellt werden.
- Freigabe des Prüfobjekts
  - Evtl. erst nach erneuter Überprüfung.

Arten von Reviews

- Inspektion
  - sehr formal
  - sehr in die Tiefe gehend (Vorlesen der einzelnen Absätze des Dokuments)
  - individuelle Vorbereitung auf Sitzung zum festgelegten Termin vor der Sitzung
- eigentlicher Review
- Walkthrough
  - abgeschwächter Review
  - nur Autor (= Moderator) und Gutachter
  - Prüfobjekt wird vom Autor ablauforientiert vorgetragen, Gutachter macht Vorschläge, Autor entscheidet
- Stellungsnahme
  - Autor bittet um Kommentare zu Prüfobjekt bei einem oder mehreren Kollegen

Qualitative Nutzenbetrachtung für Reviews

- Aufwand und Nutzen von Reviews (Erfahrungswerte)
  - Aufwand
    - 15-20% des Erstellungsaufwands des Prüfobjekts
    - Die Trainings- und Einführungskosten fallen pro Team nur einmal an
  - Nutzen
    - 60-70% der Fehler in einem Dokument werden gefunden
    - Reduktion der Fehlerkosten in der Entwicklung von 75% und mehr
    - Nettoeinsparungen für die Entwicklung ca. 20%, für die Wartung ca. 30%

- „Return on Investment“ (RoI)
  - Der RoI ergibt sich aus dem Verhältnis der ersparten Ingenieurstunden dividiert durch die Kosten. Dieser Wert ist viel besser als für viele andere Investitionen.
  - Die Investition zahlt sich schnell aus

Bewertung von Reviews

- Vorteile:
  - praktikabler Aufwand
  - auch für kleine Entwicklungsteams geeignet
  - Wissensverbreiterung
  - Entwicklungszeit wird verkürzt
  - Risiko wird reduziert
- Nachteile:
  - erfordert gewisse Disziplin
- Anwendung:
  - Standardverfahren für „normal-kritische“ Dokumente

Voraussetzungen für manuelle Prüfmethoden

- Planung
  - Feste Einplanung des notwendigen Aufwands und der benötigten Zeit
  - Jedes Mitglied des Prüfteams muss in der Prüfmethode geschult sein
  - Keine Überlastung einzelner Mitarbeiter durch Teilnahme an Prüfungen

- Organisation & Kultur
  - Hohe Priorität der Prüfungen beim Projektleiter / im Management / in der Mannschaft
  - Kurzfristige Durchführung der Prüfung
  - Prüfungsergebnisse werden nicht eingesetzt zur Beurteilung von Mitarbeitern, d. h. Vorgesetzte (und Zuhörer) sollen an den Prüfungen nicht teilnehmen

Der Mensch in manuellen Prüfmethoden

- Wichtige Grundeinstellung:
  - es geht *nicht* darum keine Fehler zu machen!
  - es geht darum, mit möglichst wenigen Fehlern *weiter* zu machen!
  - Ihre Kollegen helfen Ihnen dabei!

  - bei einer Durchsprache / Diskussion treten Missverständnisse zu Tage
  - Menschen mit unterschiedlichem Hintergrund denken an andere Dinge

- Bei allen Vorteilen
  - es wird die (harte) Arbeit eines Menschen kritisiert
  - nicht alle Menschen können damit gut umgehen
  - deshalb: Vorsicht mit der Form der Kritik:
    - immer sachlich bleiben
    - keine persönlichen Angriffe
    - konstruktive Kritik üben

Faktor Mensch

Die Grundeinstellung ist entscheidend!

Wegen der Qualitätssicherung muss ich...

Ich hab's mir genau angeschaut!

Es funktioniert doch!

Die wollen bloß mein Baby schlecht machen!

Ich hab keine Zeit für so etwas...

Ich will gerne..., damit die Qualität stimmt

Viele Augen sehen mehr...

Es gibt immer Fehler!

Die helfen mir meine Arbeit besser zu machen!

Die Zeit hole ich durch die entdeckten Fehler locker raus...

Zusammenfassende Bewertung manueller Prüfmethoden

+ **Oft die einzige Möglichkeit, die Semantik zu überprüfen**
  - Notwendige Ergänzung werkzeuggestützter Überprüfungen
  - Effizientes Mittel zur Qualitätssicherung
+ **Verantwortung für die Qualität der geprüften Produkte wird vom ganzen Team getragen**
+ **Verbreiterung der Wissensbasis der Teilnehmer**
+ **Prüfteam lernt die Arbeitsmethoden seiner Kollegen kennen**
+ **Autoren bemühen sich um eine verständliche Ausdrucksweise, da mehrere Personen das Produkt begutachten**
+ **Unterschiedliche Produkte eines Autors werden von Prüfung zu Prüfung besser**

– **In der Regel aufwendig**
  - Bis zu 20% der Erstellungskosten des zu prüfenden Produkts
– **Autoren geraten u. U. in eine psychologisch schwierige Situation**
  - »Sitzen auf der Anklagebank«
  - »Müssen sich verteidigen«

Kontrollfragen zu manuellen Prüfverfahren

- Warum sind manuelle Prüfungen notwendig?
- Wie läuft ein Review ab?
- Warum ist der Autor in einer Begutachtung in einer schwierigen Situation?

Können Sie jetzt ein Review durchführen?

Anforderungs-Review

Es wird überprüft, ob das in der Anforderungsspezifikation modellierte System später **das tun wird, was es tun soll.**

(Are we building the right product?)

= Abgleich: Anforderungsspezifikation ⇔ reale Anforderungen des Benutzers

Es werden geprüft:

- Use-Cases
- Domainklassen
- Benutzerschnittstellen
- Prototypen
- nichtfunktionale Anforderungen
- falls vorhanden: weitere Dokumente

Anforderungs-Review

- Das Review-Team besteht aus 3 bis 7 Teilnehmern.

  Neben dem
  - Autor und
  - Moderator

  sollen folgende Personen dabei sein:
- Vertreter der Anwenderseite (Stakeholders) wie
  - Auftraggeber,
  - zukünftige Benutzer,
  - (Anwendungsexperten)
- Vertreter der IT-Seite wie
  - Systemanalytiker,
  - Entwickler

Anforderungs-Review

Ablauf Validierung:

- Prüfung der textuellen Beschreibungen der Anwendungsfälle durch Anwender.
  Checkliste mit typischen Problemen wie:
  - Vage, missverständliche Formulierungen
  - Fehlende Wertebereiche
  - Inkonsistente Verwendung (z. B. verschiedene Begriffe für gleiche Sachverhalte / Daten)
- Für jeden Anwendungsfall werden ausgewählte Szenarien durchgespielt
  und betroffenen Domainobjekte auf Vollständigkeit geprüft. (auch: Testfälle werden geprüft.)
- Prüfung der spezifizierten Benutzeroberfläche.
  Empfehlenswert: Explorative Prototypen zur Beurteilung der Benutzeroberfläche (siehe Kap. Prototyping)
- Prüfen Konsistenz zwischen Use-Case-Beschreibungen und Benutzeroberfläche.
- Woher kommen, gehen die Eingabe-/Ausgabefelder?
  Existieren für Eingabe-/Ausgabefelder auch Attribute im Domainklassenmodell?
- Prüfung Domainklassenmodell durch Anwender.
  - Typen der Attribute sind richtig
  - Multiplizitäten sind richtig

Durchsprache nicht-funktionaler Anforderungen.

Anforderungs-Review

Ablauf Verifikation:

- **Intra**-Modell-Verifikation:
  Prüfung auf Vollständigkeit, Eindeutigkeit, Konsistenz eines Modells.
  - Im Anwendungsfall-Modell: Einhaltung von Regeln wie:
    - Jeder Anwendungsfall liefert ein Ergebnis
    - Einfache, richtige Formulierung (SPO)
    - Ist wirklich Black-Box-Use-Case formuliert?
    - Sind Ausnahmen in Alternative Courses / Extend-Use-Cases ausgelagert?
    - Übersichtlichkeit (wenn zu groß: Nur von einem Use-Cases benötigte Extend-/Include-Use-Cases sind als Alternative-Courses bzw. Unconditional-Insertions definiert.)
    - Verbale Beschreibung mit Grafik konsistent. (Beziehungen, Namen etc.)

Anforderungs-Review

Ablauf Verifikation:

- **Intra**-Modell-Verifikation (Fortsetzung):
  - Im Domainklassenmodell: Einhaltung von Regeln wie:
    - Klassen beziehen sich auf materielle / immaterielle Dinge der Problemwelt
    - Klassen haben mind. 1 Attribut
    - Klassen haben in Generalisierungshierarchien keine redundanten Features
    - alle Beziehungen sind mit Multiplizitäten versehen
    - ableitbare Attribute sind als solche gekennzeichnet
    - redundante Daten sind eliminiert
    - Daten / Funktionen etc. mit gleicher Bedeutung kommen nicht mit unterschiedlichen Namen mehrfach vor

Anforderungs-Review

Ablauf Verifikation:

- **Inter**-Modell-Verifikation:
  - Abgleich der verschiedenen Modelle: Einhaltung von Regel wie:
    - Auf jede Klasse muss in mindestens einem Anwendungsfall Bezug genommen werden.
    - Werden Instanzen von Klassen erzeugt, gelesen bzw. verändert und zerstört?
    - Attribute, Beziehungen für einen Anwendungsfall relevant?

      (CRUD-Matrix: Matrix mit Domainklassen und Anwendungsfällen.

      Matrixeinträge: Create, Read, Update, Delete.)

Anforderungs-Review

- Frage: Welche Rollen sind für was zuständig? (Rollen S. 25)
- Wer (in erster Linie) für Validierung?
- Wer (in erster Linie) für Verifikation?

Entwurfs-Review

Es wird überprüft, ob das Produkt so entworfen ist, dass es **das tut, was es auf Grund der Anforderungsdefinition tun soll.**

(are we building the product right?)

= Abgleich: Entwurfspezifikation ⇔ Anforderungsdefinition

d. h. nicht unbedingt Abgleich gegen reale Anforderungen.

Aber:

Falls Unvollständigkeiten oder Inkonsistenzen in Anforderungsdefinition gefunden werden oder

falls nichtfunktionale Anforderungen nicht definiert bzw. definierte so nicht erfüllbar sind:

→ Auch Änderung in Anforderungsdefinition

(nochmalige Validierung der Anforderungen).

Entwurfs-Review

- Die Gruppe für eine Entwurfsinspektion besteht aus 3 bis 5 Teilnehmern
  (Bei kleineren Projekten evtl. nur 2 Teilnehmern)
  - 1. Entwerfer
  - 2. Moderator
  - 3. Erfahrener Programmierer (Protokollführer für entdeckte Fehler)
  - 4. evtl. 2-3 weitere Teilnehmer
    (Codierer, Wartungspersonal, Mitarbeiter aus anderen Projekten)

Entwurfs-Review

Ablauf Verifikation (Vorbereitung auf die Review-Sitzung):

- Anhand der Anforderungsdefinition werden die Fälle mittels des Entwurfs durchgespielt. (Evtl. Definieren und Durchspielen zusätzlicher Scenarien.)
- Die anhand des Entwurfs zu erwartenden Ergebnisse (auch nichtfunktionale Anforderungen) werden daraufhin überprüft, ob sie mit den gemäß Anforderungsdefinition zu erwartenden übereinstimmen. (auch: Testfälle werden geprüft.)
- Prüfung, ob jedes Modell in sich konsistent (Intra-Modell-Verifikation)

  (ähnlich Überprüfung z. B. Domain-Klassenmodell – siehe vorne)
- Prüfung, ob alle Modelle im Zusammenwirken untereinander konsistent (Inter-Modell-Verifikation). Oft durch CASE-Tool unterstützt.
- Sind redundante Daten vermieden.

Entwurfs-Review

Ablauf Review-Sitzung:

- Durchführen der Schritte der Verifikation.
- Es wird ein Protokoll über entdeckte Fehler, Verbesserungsvorschläge und Fragen an den Autor geführt.
- Über kritische Stellen des Entwurfs wird mit dem Entwerfer diskutiert.
  (Ausführliche Diskussionen z. B. auch über Lösungsmöglichkeiten:
  In separater Sitzung im kleinen Kreis – z. B. im Anschluss an die Hauptsitzung)
- Auch "Schwachstellen" werden protokolliert.
- Es wird aber noch keine Fehlerkorrektur vorgenommen ! Fehler werden erst (später durch den Entwerfer bereinigt)

Code-Review

Es wird überprüft, ob **der Code das tut, was er auf Grund der Entwurfsspezifikation tun soll.**

= Abgleich: Code ⇔ Entwurfsspezifikation

Die Mannschaft für eine Codeinspektion besteht aus ca. 4 Teilnehmern :

- 1. Programmierer des vorliegenden Produkts
- 2. Moderator
- 3. Testspezialist
- 4. Entwerfer

Code-Review

Ablauf der Sitzung:

- Der Programmierer erklärt die Programmlogik Schritt für Schritt gemäß der Anweisungen.

  (Evtl. auch nur Durchsprache der vorab im Einzelstudium gefundenen Fehler)
- Eventuell auftretende Fragen oder

  Abweichungen von den Codierungsrichtlinien

  werden verfolgt.
- Der Moderator protokolliert die Fehler.
- Kritische Fälle werden eventuell durchgespielt und

  die Ergebnisse mit denen gemäß der Entwurfsspezifikation verglichen.
- Die vorliegende Programmliste wird anhand einer Checkliste häufiger Fehler analysiert (auch schon vorher von jedem Teilnehmer)
- Der Moderator achtet darauf, dass sich auf das Entdecken von Fehlern beschränkt wird und protokolliert diese.
- Die Korrektur wird später durch den Programmierer vorgenommen.

Checkliste für Code-Review

Rote unterstrichene Nummer bedeutet : Fehler können vom Compiler normalerweise nicht gefunden werden, d.h. diese Punkte sind besonders wichtig für Nachprüfungen.

Variablenbenutzung

1. Werden Variable angesprochen, die nicht initialisiert wurden?
2. Werden die Indexgrenzen für Felder (Arrays) eingehalten?
3. Sind die Indizes vom richtigen Typ?
4. Werden Zeiger richtig verwendet?
5. Stimmen formale Parameter und die Argumente für die Unterprogramme im Typ überein?
6. Ist die Anzahl der Argumente gleich der Argumente der formalen Parameter bei Unterprogrammen?
7. Sind Variablennamen aussagekräftig
   (*nicht: A := B + C / 100 B, sondern: brutto := netto + Mwst_Satz / 100 netto)*
   und gemäß Programmierrichtlinien benannt.

Variablenvereinbarung

1. Sind alle Variablen vereinbart?
2. Werden Felder richtig initialisiert?
3. Gibt es Variable mit ähnlichen Namen?
4. Sind die globalen Variablen an der richtigen Stelle vereinbart?

Berechnungen

1. Werden Berechnungen mit nicht arithmetischen Variablen durchgeführt?
2. Gibt es Berechnungen, an denen Daten unterschiedlichen Typs beteiligt sind?
3. Kann es Über- oder Unterlauf in Zwischenergebnissen geben?
4. Kann irgendwo eine Division durch Null auftreten?
5. Wo können unvertretbare Ungenauigkeiten durch die Dualdarstellung auftreten?
6. Wurde die Priorität der Operationen beachtet?
7. Sind die ganzzahligen Divisionen korrekt?

Checkliste für Code-Review

Vergleiche

1. Werden Werte verschiedenen Typs miteinander verglichen?
2. Sind die Vergleichsoperationen korrekt verwendet?
3. Sind die Boolschen Ausdrücke korrekt?
4. Treten überkomplizierte Boolsche Ausdrücke auf?

Programmablauf

1. Werden bei Fallunterscheidungen alle Fälle berücksichtigt?
2. Wird jede Schleife richtig beendet? (Endlosschleife möglich?)
3. Hört das Programm nach endlich vielen Schritten auf?
4. Werden Schleifen aufgrund der Eingangsbedingung nicht ausgeführt?
5. Ist die Schrittzahl in zählenden Schleifen um eins zu hoch oder zu niedrig?

Schnittstellen

1. Werden Eingabeparameter im Unterprogramm verändert?
2. Werden Konstante auf der Position von Variablenparametern als Argument übergeben?
3. Ist die Benutzung globaler Variablen über alle Module konsistent?
4. Kann auf die Verwendung zumiondest einiger globaler Variablen zugunsten von Parametern/Instanzvariablen verzichtet werden?
4. Sind Parameter und Argumente in Bezug auf Typ und Argument konsistent?

Ein- und Ausgabe

1. Werden die Dateien richtig geöffnet und geschlossen?
2. Sind die Dateien richtig vereinbart?
3. Werden Zeilenende und Dateiende korrekt abgefragt?

Weitere Prüfungen

1. Gibt es nicht angesprochene Variablen im Programm?
2. Erhält man bei der Übersetzung Warnungsmeldungen?
3. Werden die Eingabedaten auf Gültigkeit geprüft?
4. Fehlen bestimmte Funktionen oder Prozeduren?
5. Ist Programm ausreichend kommentiert, sind Kommentare richtig und aussagekräftig
6. Sind Programmierrichtlinien eingehalten?

Formale Verifikation

Formale Verifikation:

Es wird mathematisch bewiesen, dass ein Programm zu zulässigen Eingabewerten die richtigen Ergebnisse (Ausgabewerte) liefert, d.h. richtig ist.

Weiterhin muss bewiesen werden, dass ein Algorithmus in jedem Falle terminiert.

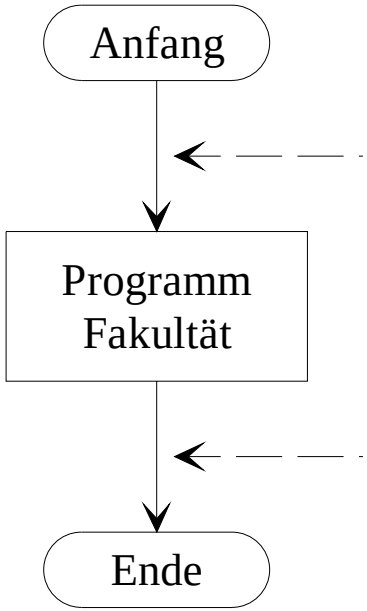
Vorgehen bei der formalen Verifikation

- Jede geschaffene Funktionalität eines Softwaresystems hat die Aufgabe,
  - aus bestimmten Anfangsbedingungen über Daten („precondition“)
  - bestimmte Endbedingungen („postcondition“) herbeizuführen.

  Dies ist wie ein Vertrag zu sehen zwischen demjenigen,
  - der eine Funktionalität benutzt,
  - und dem, der sie erfüllt.
- Die formale Verifikation eines Systems verläuft nun folgendermaßen:

  Durch formales Anwenden von Regeln der mathematischen Prädikatenlogik wird über die Wirkungsweise des Algorithmus der Beweis geführt wird, dass er aus seinen Anfangsbedingungen die Endbedingungen herstellt. Gelingt dieser Beweis, so erfüllt das System exakt die Spezifikation und ist damit korrekt.

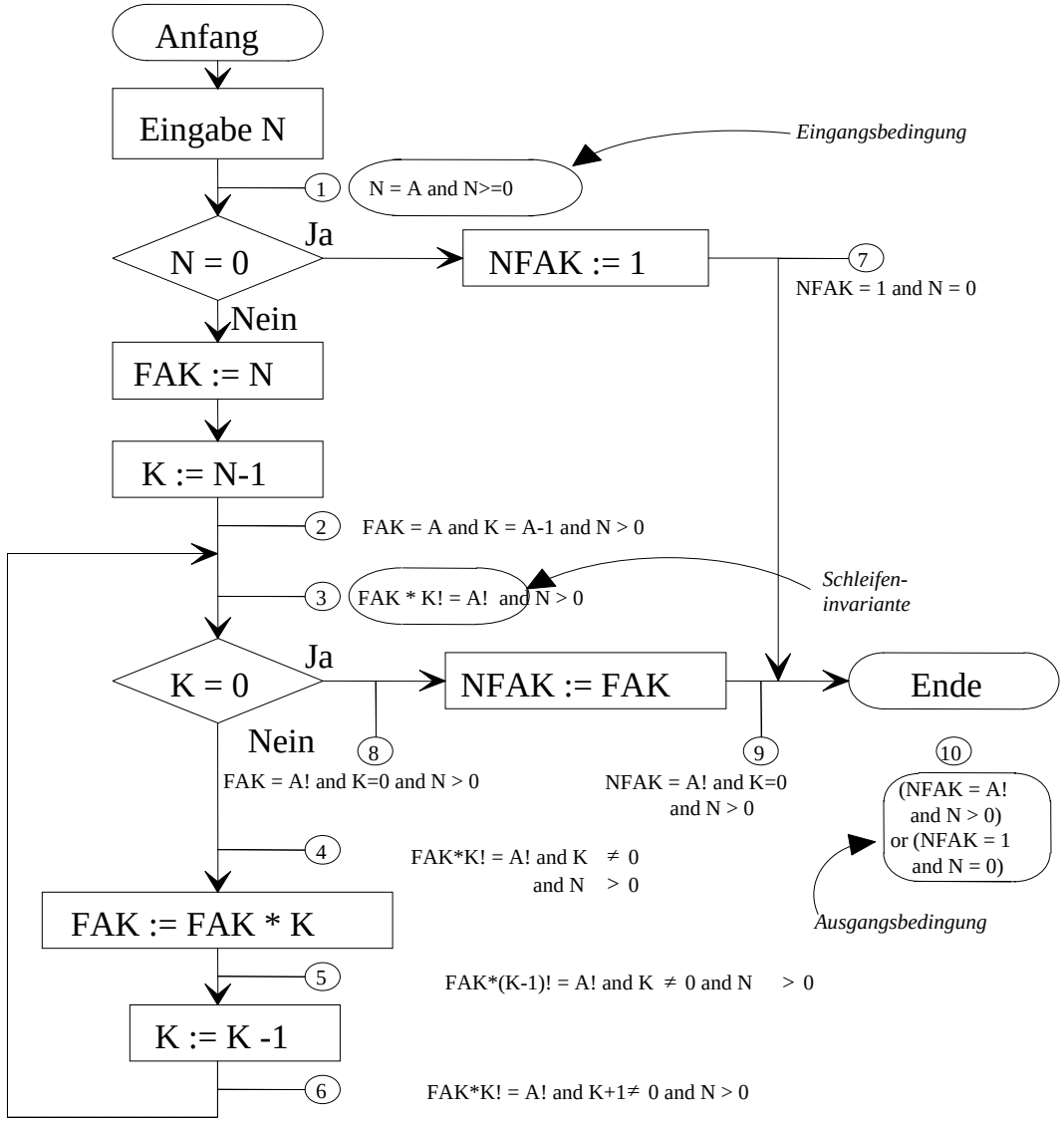
Beispiel "Formale Verifikation der Funktion fakultaet()"



Eingangsbedingung (pre-condition):
N=A AND N >= 0
[A sei ein zulässiger konkreter Wert der Eingabevariable N]

Ausgangsbedingung (post-condition):
(NFAK=A! AND N > 0)
OR
(NFAK=1 AND N=0)
[NFAK sei das Ergebnis der Funktion]

Beispiel "Formale Verifikation der Funktion fakultaet()"



Beispiel "Formale Verifikation der Funktion fakultaet()"

Bemerkung:

- Formale Verifikation sehr aufwendig

  ➔ nur durchführbar bei Programmen mit extrem hohen Anforderungen an die Zuverlässigkeit

  (z. B. Teil der Programme zur Kernkraftwerks-Steuerung)

Abnahme des Gesamtsystems

Es wird überprüft, ob **das System das tut, was er auf Grund der Anforderungsspezifikation tun soll.**

= Abgleich: Systemverhalten ⇔ Anforderungsspezifikation

- Tests durchgeführt von Entwickler + Testspezialist

Nach Abschluss aller Tests:

- Das System wird durch Vertreter der Anwenderseite (Stakeholders) wie
  - Aufraggeber,
  - zukünftige Benutzer,
  - (Anwendungsexperten)

  abgenommen, d. h.
  - Programm wird getestet und nichtfunktionale Anforderungen geprüft,
  - Dokumentation, Benutzerhandbuch wird durchgelesen, … und
  - es wird bestätigt, dass Systemverhalten, Dokumentation, Benutzerhandbuch etc. der Anforderungsspezifikation entspricht.