Software-Paradigmen – Was programmiere ich da eigentlich?

Dipl.-Ing. Frank Listing, MicroConsult GmbH

Ein SW-Paradigma ist ein grundsätzlicher Programmierstil. Also die Art und Weise, wie ein Programmierer seinen Code schreibt.

Die Unterschiede der Paradigmen liegen im Umgang mit den Daten und ihrer Repräsentation bzw. in der Darstellung des Kontrollflusses.

Die verschiedenen Paradigmen schließen sich nicht grundsätzlich gegenseitig aus. Sie können in vielen Fällen untereinander kombiniert werden.

Bei der Bezeichnung der Paradigmen herrscht oft Verwirrung. Einige Namen werden falsch verwendet, bei anderen herrscht Unklarheit, wo neben der großen Ähnlichkeit nun der Unterschied besteht. In diesem Artikel werden einige der heute gebräuchlichen SW-Paradigmen aufgezeigt und erläutert.

Überbegriffe

Grundsätzlich werden zwei verschiedene Arten der Programmierung unterschieden: Die imperative Programmierung und die deklarative Programmierung.

Bei der imperativen Programmierung wird in der Programmiersprache der Lösungsweg eines Problems beschrieben. Bei der Abarbeitung des Programmes wird das Problem auf diese Weise gelöst.

```
procedure quicksort(l,r : integer);
var x,i,j,tmp : integer;
begin
  if r>l then
  begin
    x:=a[l]; i:=l; j:=r+1;
    repeat
      repeat  i:=i+1 until a[i]>=x;
      repeat  j:=j-1 until a[j]<=x;
      tmp:=a[j]; a[j]:=a[i]; a[i]:=tmp;
    until j<=i;
    a[i]:=a[j]; a[j]:=a[l]; a[l]:=tmp;
    quicksort(l,j-1);
    quicksort(j+1,r)
  end
end;
```

Listing 1: Beispiel imperative Programmierung – Quicksort in Pascal

Die deklarative Programmierung hingegen setzt auf eine detaillierte Beschreibung des Problems. Der Compiler ist damit in der Lage, einen Lösungsweg zu generieren.

```
quicksort [] = []
quicksort (x:xs) = quicksort [n | n<-xs, n<x] ++ [x] ++
    quicksort [n | n<-xs, n>=x]
```

Listing 2: Beispiel deklarative Programmierung – Quicksort in Haskell

Paradigmen

Prozedurale Programmierung

Die prozedurale Programmierung zerlegt ein Programm in überschaubare Teile. Dafür werden Prozeduren und Funktionen verwendet. Daten werden entweder als Parameter übergeben oder global bereitgestellt.

Objektorientierte Programmierung

Die objektorientierte Programmierung orientiert sich am menschlichen Alltag. Der Mensch nimmt seine Umwelt in Form von Objekten wahr. Dieses Prinzip wird in die Programmierung übernommen. Grundbausteine sind Abstraktion - Daten und Funktionen werden zusammengefasst (in eine Klasse), Kapselung - Daten werden nach außen hin gekapselt, nicht zum Objekt gehörende Methoden dürfen nicht auf diese zugreifen, Vererbung - abgeleitete Klassen erben Daten und Funktionen der Basisklasse.

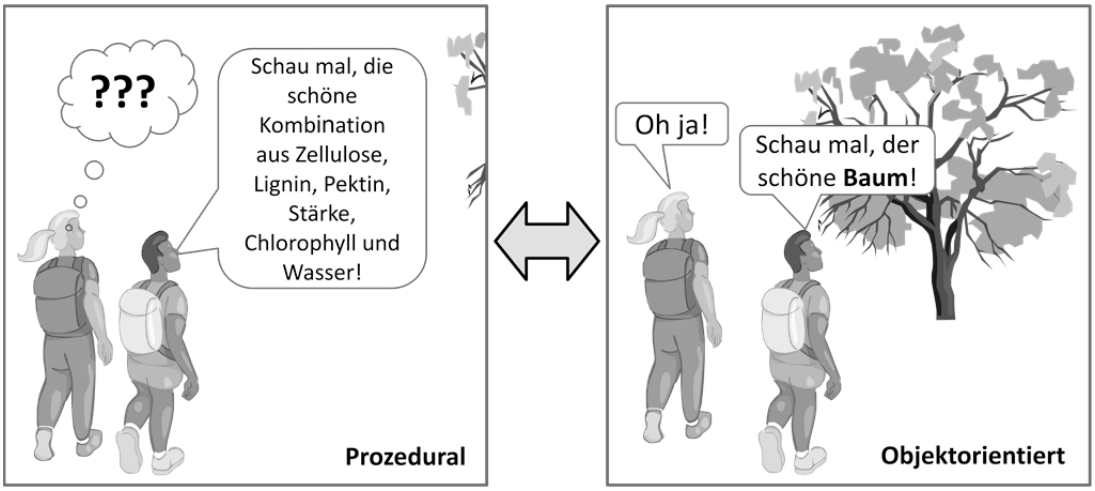

Bild 1: Prozeduralen Programmierung vs. Objektorientierte Programmierung

Im Gegensatz zur prozeduralen Programmierung ist bei der objektorientierten Programmierung ein Zusammenhang zwischen Daten und Funktionalität klar erkennbar.

Strukturierte Programmierung

Die strukturierte Programmierung basiert auf einer baumartigen Zerlegung des Programmes in Teilprogramme (Prozeduren). Sie ist damit eine Erweiterung der prozeduralen Programmierung. Die Besonderheit der strukturierten Programmierung ist die Beschränkung auf drei Kontrollstrukturen:

1. Sequenz (Anweisungen, die hintereinander ausgeführt werden),

2. Verzweigung (Entscheidung) und Wiederholung (Schleife),

3. Vermeidung von goto.

Die Anwendung der strukturierten Programmierung ist heute selbstverständlich, viele neuere Programmierparadigmen bauen darauf auf.

Modulare Programmierung

Die modulare Programmierung unterstützt das Verständnis von großen Programmen. Diese werden in Teilblöcke (Module) aufgeteilt. Der Vorteil liegt in der Senkung der Komplexität. Module können einzeln entworfen, entwickelt und getestet werden. Weiterhin wird die Wiederverwendung von Modulen verbessert bzw. überhaupt ermöglicht. Durch saubere Schnittstellen zwischen den Modulen können diese durch Testtreiber und Teststubs einzeln getestet werden. Das erleichtert die Fehlersuche enorm.

Aspektorientierte Programmierung

Die aspektorientierte Programmierung erweitert vor allem die objektorientierte Programmierung um die Möglichkeit generische Funktionen über mehrere Klassen hinweg zu verteilen. Das geschieht unabhängig von vorhandenen Klassenhierarchien. Typische Beispiele für Aspekte sind Protokollierung (Logging), Fehlerbehandlung und Persistenz.

Funktionale Programmierung

Die funktionale Programmierung basiert auf dem mathematischen Funktionsbegriff. Ein Programm wird als Funktion betrachtet, die wiederum aus Teilfunktionen zusammengesetzt wird. Als historische theoretische Grundlage dient das Lambda-Kalkül von Alonzo Church. Jeder Ausdruck und jeder Wert wird als auswertbare Funktion betrachtet. Das bedeutet, dass ein wesentlicher Bestandteil der Programmierung die Übergabe von Funktionen als Parameter an Funktionen ist. Auch die Rückgabewerte von Funktionen sind oft wieder Funktionen. Variablen haben immer denselben Wert. Es gibt weder Zuweisungen noch Schleifen. Als Ersatzmechanismus werden Rekursionen eingesetzt.

Programmiersprache und Paradigma

Verschiedene Programmierparadigmen wurden lösungsbezogen entwickelt und sind bei falschem Einsatz unproduktiv, z.B. imperativ vs. deklarativ. Andere Paradigmen bauen aufeinander auf und ergänzen sich gegenseitig, wie z.B. prozedurale und strukturierte Programmierung. Die objektorientierte Entwicklung ist heutzutage Stand der Technik und wird in vielen Projekten eingesetzt.

Auswahl einer Programmiersprache

Die Auswahl einer Programmiersprache sollte möglichst auf Basis objektiver Kriterien gefällt werden. Hobbies oder Vorlieben einzelner Entwickler spielen hierbei eine untergeordnete Rolle.

1. Zielsystem

Auf welcher Plattform soll die Applikation laufen (embedded Device, Windows-PC, Linux-PC, …). Soll die Applikation auf mehreren Plattformen laufen? Das Zielsystem schränkt in vielen Fällen die Auswahl der Programmiersprachen stark ein. Besonders für embedded Systeme sind oft nur C- oder C++ - Compiler verfügbar.

2. Randbedingungen

Welche zusätzlichen Randbedingungen müssen beachtet werden? Gibt es Echtzeitanforderungen? Welche Bibliotheken passen am besten, um die Aufgabe umzusetzen und an welche Programmiersprache sind diese gebunden? Ist ein guter Support vorhanden oder gibt es aktive Communities auf die im Problemfall zurückgegriffen werden kann?

3. Team

Wie ist der Ausbildungsstand des Teams? Welche Programmiersprache wird am besten beherrscht? Gibt es Vorlieben bei den Entwicklern? Hier sollte ein größtmöglicher Konsens gefunden werden. Wenn eine neue Programmiersprache eingeführt werden muss, ist die Motivation der Entwickler zu beachten. Die unbegründete Anweisung, eine neue Programmiersprache einzusetzen, wird zum Scheitern des Projektes führen.

Fazit

Es gibt nicht DIE Programmiersprache! Es muss immer das größtmögliche Optimum angestrebt werden. Oft wird die Auswahl durch die Plattform stark eingeschränkt (z.B. embedded Programmierung). Es ist immer sinnvoll, eine auf der Zielplattform etablierte Sprache zu verwenden – Exoten sind Risikobehaftet.

Quellen:

http://de.wikipedia.org/wiki/Programmierparadigma
http://de.wikipedia.org/wiki/Prozedurale_Programmierung
http://de.wikipedia.org/wiki/Objektorientierte_Programmierung
http://de.wikipedia.org/wiki/Strukturierte_Programmierung
http://de.wikipedia.org/wiki/Modulare_Programmierung
http://de.wikipedia.org/wiki/Aspektorientierte_Programmierung
http://de.wikipedia.org/wiki/Funktionale_Programmierung

Autorenprofil:

Dipl.-Ing. **Frank Listing** ist seit 2002 Trainer und Projektcoach bei der MicroConsult GmbH mit dem Schwerpunkt Microsoft-Plattformen, objektorientierte Programmierung und Testen von Embedded Systemen und u.a. fachlich für das Thema .NET verantwortlich. Sein Wissen gibt er immer wieder auch in Publikationen und Fachvorträgen weiter.