Unterstützung der Modularität

Softwareengineering erfordert Module:
* Aufteilung eines Projekts auf Mitarbeiter
* Lokale Fehler in einem Modul können lokal behoben werden
* Korrektness des Programms hängt von jedem einzelnen Modul und dem Zusammenspiel der Module ab

Kapselung: Ein Service pro Modul

Data hiding = Trennung Schnittstelle / Implementierung:
* Clients brauchen nur relevante Aspekte eines Moduls kennen
* Implementierung ohne Einfluß auf Clients änderbar

Module unterstützen unabhängige oder getrennte Übersetzung

Module sollen mit Namenskonflikten und Zusätzen in neueren Versionen umgehen können

Namensräume in C++

```
namespace XYZCorp {
   typedef turbodiesel ...;
   void start(turbodiesel);
}

XYZCorp::turbodiesel t;

using XYZCorp::turbodiesel;
turbodiesel t;
XYZCorp::start(t);

using namespace XYZCorp;
turbodiesel t;
start(t);
```


Schnittstellen in Ada

```
with X;
package T is
   C: Integer;
   procedure D(...);
end T;

package body T is
   ...
   ...
   ...
end T;
```

```
with T;
procedure U(...) is
   ...
   ... T.D ...
   ... T.C ...
end U;

with T;
use T;
procedure U(...) is
   ...
   ... D ...
   ... C ...
end U;
```


Bibliothekspakete in Ada

```
package Root is
   ... -- Spezifikation der Bibliothek Root
end Root;

package Root.Child is
   ... -- Spezifikation der Bibliothek Root.Child
end Root.Child;

package body Root.Child is
   ... -- Implementierung der Bibliothek Root.Child
end Root.Child;
```

Generische Funktionen in C++

```
template <class T>
void swap(T& a, T& b)
{
   T temp = a;
   a = b;
   b = temp;
}
```

```
int i, j;
char x, y;
pair<int,string> p1, p2;
...
swap(i,j);    // swap integers
swap(x,y);    // swap characters
swap(p1,p2);  // swap pairs
```


Generische Funktionen in Ada

```
generic
   type T is private;
procedure Swap(X,Y: T) is
   Temp: T := X;
begin
   X := Y;
   Y := Temp;
end Swap;
```

```
generic
   type T is private;
   with function "<"(X,Y: T)
         return Boolean is <>;
function Max(X,Y: T)
         return Boolean is
begin
   if X < Y then return Y;
   else return X;
   end if
end Max;

function Int_Max is
         new Max(Integer);
```