
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

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
class dict {
public:
    dict();           //constructor for dictionary
    ~dict();         //destructor for dictionary
    void insert (char* c, int i);
    int lookup(char* c);
    void remove (char* c);
private:
    struct node {
        node* next;
        char* name;
        int id};
    node* root;
};
```

FIGURE 5.1 Definition of a dictionary module in C++

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
package Dictionary is  
  procedure Insert (C:String; I: Integer);  
  function Lookup(C:String): Integer;  
  procedure Remove (C: String);  
end Dictionary;
```

FIGURE 5.2 Package specification in Ada

```
package body Dictionary is  
  type Node;  
  type Node_Ptr is access Node;  
  type Node is  
    record  
      Name: String;  
      Id: Integer;  
      Next: Node_Ptr;  
    end record;  
  Root: Node_Ptr;  
  procedure Insert (C:String; I: Integer) is  
  begin  
    --implementation...  
  end Insert;  
  function Lookup(C:String) return Integer is  
  begin  
    --implementation...  
  end Lookup;  
  procedure Remove (C: String) is  
  begin  
    --implementation...  
  end Remove;  
begin  
  Root := null;  
end Dictionary;
```

FIGURE 5.3 Package body in Ada

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
with Dictionary; use Dictionary;  
procedure Main is  
    Code: Integer;  
begin  
    Insert ("volleyball", 1);  
    Insert ("basketball", 2);  
    Insert ("football", 3);  
    ...  
    Code := Lookup ("basketball");  
    ...  
end;
```

FIGURE 5.4 Fragment of a client of the dictionary package of Figure 5.2

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
Unit A  
  export routine X (int, int);  
  ...  
end A
```

```
Unit B  
  import from Unit A  
  ...  
  call X (...);  
  ...  
end B
```

FIGURE 5.5 Sketch of a program composed of two units

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
program program_name (files);  
declarations of constants, types, variables, procedures and functions;  
begin  
    statements (no declarations)  
end.
```

FIGURE 5.6 Structure of a Pascal program

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

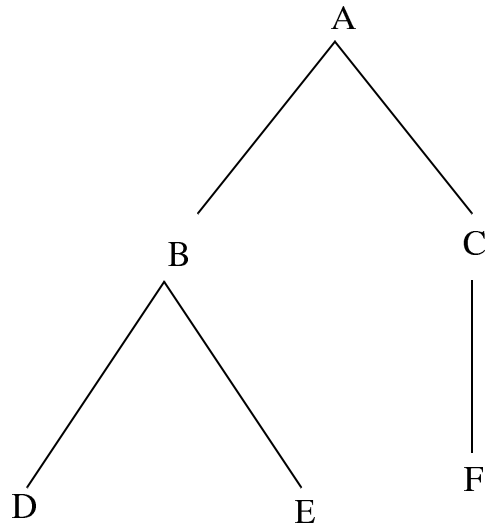


FIGURE 5.7 Static nesting tree of a hypothetical Pascal program

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

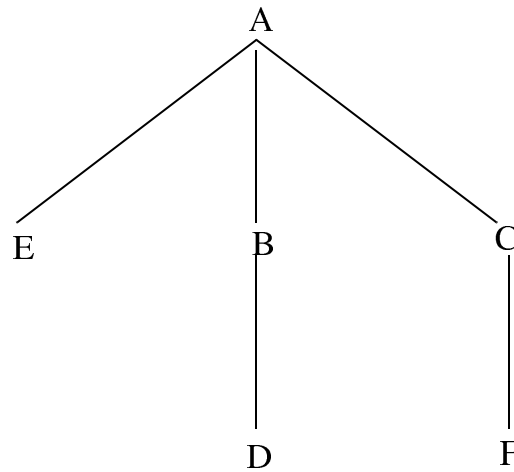


FIGURE 5.8 A rearrangement of the program structure of Figure 5.7.

```

/* file stack.h */
/* header (include) file exporting declarations to clients*/
typedef struct stack {
    int elements[100]; /* stack of 100 ints */
    int top;           /* number of elements*/
};
void push(stack, int);
int pop(stack);
/*****-----end of file      *****/

```

```

/*file stack.c */
/*implementation of stack operations*/
#include "stack.h"
void push(stack s, int i) {
    s.elements[s.top++] = i;
};
int pop (stack s) {
    return s.elements[--s.top];
};
/*****-----end of file      *****/

```

```

/* file main.c */
/* a client of stack*/
#include "stack.h"
void main(){
    stack s1, s2;           /* declare two stacks */
    s1.top = 0; s2.top = 0; /* initialize them */
    int i;
    push (s1, 5); /* push something on first stack */
    push (s2, 6); /* push something on second stack*/
    ...
    i = pop(s1); /* pop first stack */
    ...
}

```

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
#include ...various files...
global declarations
function definitions
void main (parameters)
{
...one main function needed
...in a program
}
```

FIGURE 5.10 Structure of a C module

```

// ***** file stack.h
//header file containing declarations exported to clients
class stack {
public:
    stack();
    void push(int);
    int pop();
private:
    int elements[100]; // stack represented as array
    int top = 0;      // number of stored elements
};
// *****end of file stack.h
// *****file stack.c
// the implementation of stack member functions
#include "stack.h"
void stack::push(int i) {
    elements[top++] = i;
};
int stack::pop() {
    return elements[--top];
};
// *****end of file stack.c
// ***** file main.c
// a client of stack
#include "stack.h"
main(){
    stack s1, s2; // declare two stacks
    int i;
    s1.push(5); // push something on first stack
    s2.push(6); // push something on second stack
    ...
    i = s1.pop(); // pop first stack
    ...
}
// *****end of file main.c

```

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
class complex {  
public:  
    complex(double r, double i ){re = r; im = i;}  
    friend complex operator+ (complex, complex);  
    friend complex operator- (complex, complex);  
    friend complex operator* (complex, complex);  
    friend complex operator/ (complex, complex);  
private:  
    double re, im;  
};
```

FIGURE 5.12 Illustration of the use of friend declarations in C++

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;
```

```

procedure X ( ... ) is                                --unit specification
  W: Integer;
  package Y is                                         --inner unit specification
    A: Integer;
    function B (C: Integer) return Integer;
  end Y;
  package body Y is separate;                          --this is a stub
begin                                                  --uses of package Y and variable W
  ...
  ...
  ...
end X;
-----next file-----
separate (X)                                         --subunit's body
package body Y is
  procedure Z (...) is separate;                    --this is a stub
  function B (C: Integer) return Integer is
  begin                                              --use procedure Z
    ...
    ...
    ...
  end B;
end Y;
-----next file-----
separate (X.Y)
procedure Z (...) is
begin
  ...
end Z;

```

FIGURE 5.13 The use of stubs in Ada

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
package Dictionary is  
  type Dict is private;  
  procedure Insert (D: in out Dict; C:String; I: Integer);  
  function Lookup(D: Dict; C: String) return Integer;  
  procedure Remove (D: in out Dict; C: String);  
private  
  type Node;  
  type Node_Ptr is access Node;  
  type Node is  
    record  
      Name: String;  
      Id: Integer;  
      Next: Node_Ptr;  
    end record;  
  type Dict is new Node_Ptr;  
end Dictionary;
```

FIGURE 5.14 An Ada package specification exporting a (private) type

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
with Dictionary;  
use Dictionary;  
procedure Main is  
    People, Things: Dict;  
    ...  
begin  
    Insert (Things, "Table", 20);  
    Insert (People, "Bill", 45);  
    ...  
end;
```

FIGURE 5.15 A client of package Dictionary

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;

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
structure Dictionary =
struct
  exception NotFound;

  val create: (string * int) list = nil; (*create an empty dictionary*)

  (* insert (c, i, D) inserts pair <c,i> in dictionary D*)
  fun insert (c:string, i:int, nil:(string * int) list) = [(c,i)]
    | insert (c, i, (cc, ii)::cs) =
      if c=cc then (c,i)::cs
      else (cc, ii)::insert(c,i,cs);

  (* lookup (c, D) finds the value i such that pair <c,i> is in dictionary D *)
  fun lookup(c:string, nil:(string * int) list) = raise NotFound
    | lookup (c, (cc,ii)::cs) =
      if c = cc then ii
      else lookup(c,cs);
end;
```

FIGURE 5.16 Dictionary module in ML

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
val D = Dictionary.create; (*create an empty dictionary *)  
val newD = Dictionary.insert ("Mehdi", 46, D); (*insert a pair*)  
...  
Dictionary.lookup("Mehdi", newD); (*produces value 46*)
```

FIGURE 5.17 Fragment of client of Dictionary structure of Figure 5.16

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
signature DictLookupSig = sig
  exception NotFound;
  val lookup : string * (string * int) list -> int
end
```

FIGURE 5.18 A signature definition for a specialized dictionary

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
structure LookupDict: DictLookupSig = Dictionary;  
val L = LookupDict.create; (* not allowed by the LookUp interface *)  
val D = Dictionary.create; (*create a new empty dictionary *)  
val newD = Dictionary.insert ("Mehdi", 46, D);(* insert something *)  
val age = LookupDict.lookup("Mehdi", newD);  
val L2 = LookupDict.insert("Carlo", 50, newD); (*error, insert not available*)
```

FIGURE 5.19 A structure constrained by a signature

Generische Funktionen in C++

```
template <class T>                int i, j;
void swap(T& a, T& b)            char x, y;
{                                  pair<int,string> p1, p2;
    T temp = a;                  ...
    a = b;                       swap(i,j);    // swap integers
    b = temp;                     swap(x,y);    // swap characters
}                                  swap(p1,p2);  // swap pairs
```

Generische Funktionen in Ada

```
generic
  type T is private;
procedure
  Swap(in out 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 T is
begin
  if X < Y then return Y;
  else return X;
  end if
end Max;

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

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
template <class T1, class T2>
class pair {
    public:
        T1 first;
        T2 second;
        pair (T1 x, T2 y) : first(x), second(y) { }
            // first(x) initializes first with x;
            // second(y) initializes second with y;
};
...
pair<int, int> intint(2, 1456);
pair<string, int> stringint("Mehdi", 46);
pair<employee_t, employee_t> employees(jack, jill);
    //pair of user-defined type employee_t
```

FIGURE 5.20 Definition and use of a generic pair data structure in C++

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
structure Dictionary =
struct
  exception NotFound;
  val create = nil; (*create an empty dictionary*)
  (* insert (c, i, D) inserts pair <c,i> in dictionary D*)
  fun insert (c, i, nil) = [(c,i)]
    | insert (c, i, (cc, ii)::cs) =
      if c=cc then (c,i)::cs
      else (cc, ii)::insert(c,i,cs);
  (* lookup (c, D) finds the value i such that pair <c,i> is in dictionary D *)
  fun lookup(c, nil) = raise NotFound
    | lookup (c, (cc,ii)::cs) =
      if c = cc then ii
      else lookup(c,cs);
end
```

FIGURE 5.21 A polymorphic Dictionary module in ML

©1998 Ghezzi and Jazayeri, from Programming Language Concepts, Third Edition, published by John Wiley.

```
template<class Iter, class T>
Iter find (Iter first, Iter last, T x)
{
    while (first != last && *first != x)
        ++first;
    return first;
}
```

FIGURE 5.22 A generic algorithm find