Realizace principů softwarového inženýrství v programovacích jazycích

Jarmila Holcová

1. Úvod

Softwarové inženýrství je disciplína, která vznikla za účelem překonat softwarovou krizi. Zabývá se zkoumáním a vývojem metod, technik a nástrojů, které podporují efektivní tvorbu kvalitního softwaru. Přitom kvalitním softwarem rozumíme nejen software správný (tj. splňující systémovou specifikaci) ale také dostatečně spolehlivý, uživatelsky vhodný a pohodlný, udržovatelný, efektivní a přenositelný. Z etap životního cyklu softwaru (analýza požadavků, systémová specifikace, návrh, implementace, testování, údržba) kvalitu ovlivňuje zejména etapa návrhu.

Existuje řada prostředků (systémy CASE - Computer Aided Software Egineering), které dodržují určitou metodu nebo kombinaci metod a usnadňují návrh (grafické prostředí) a automatizují implementaci (generují kód). Většinou však pokrývají pouze určitou aplikační oblast (technologické procesy, ekonomické systémy, ...) nebo speciální část softwaru (uživatelské obrazovky, výstupní sestavy). Avšak v mnoha případech nebo lépe řečeno, obecně, prostředníkem pro transformaci návrhu do tvaru, který je srozumitelný počítači, zůstává programovací jazyk.

V souladu se základními principy softwarového inženýrství jsou na současný programovací jazyk kladeny stále větší nároky.

2. Programovací jazyky

Existují stovky programovacích jazyků a nové stále vznikají. Většina z nich vede nevýznamný život v okolí svých tvůrců a počítačová veřejnost o nich ani neví. Aby byl programovací jazyk obecně přijat, musí vycházet z konkrétní potřeby, protože jinak není ospravedlněno úsilí vynaložené na přeškolení programátorů. Existence jazyka a jeho kompilátoru musí být zabezpečena po dobu 20 let a více, spolehlivé kompilátory musí být dostupné na všech důležitých cílových strojích a musí být zabezpečen jejich vývoj a údržba.

Až dosud pouze Fortran, Cobol, PL/1, Basic, Pascal a C splňují tyto primární ekonomické požadavky.

3. Pascal, C, Modula-2

Programovací jazyky pro univerzální účely, které se staly všeobecně uznávané, používané a rozšířené jsou jazyky Pascal a C. Jazyk, který by se mohl dostat do stejné situace, je jazyk Modula-2. Společný rys těchto jazyků je to, že jejich syntaktické a sémantické vlastnosti podporují jisté základní principy softwarového inženýrství.

Programovací jazyk Pascal byl vyvinut v 70. letech jako jazyk pro výuku strukturovaného programování, jako tehdy nové metody softwarového inženýrství.

Jeho následníkem je jazyk Modula-2, poprvé definovaný Wirthem v roce 1982, který podporuje další metodu sofwarového inženýrství a to modulární programování. Obsahuje navíc konstrukty umožňující vyjádřit problémy, na které Pascal nestačí, např. reprezentaci paralelních procesů.

Jazyk C byl původně vyvinut pro vývoj operačních systémů a další systémový software s cílem zajistit co největší přenositelnost a efektivnost. Postupně se začal používat i v jiných aplikačních oblastech (systémy reálného času, vestavěný software, ...). Dnešní oficiální standard je ANSI C z roku 1988.

Realizace základních principů softwarového inženýrství v těchto jazycích je předmětem následujících odstavců.

4. Princip abstrakce

Univerzálním principem softwarového inženýrství je princip abstrakce. Dle slovníku cizích slov je abstrakce myšlenkový proces, resp. jeho výsledek, v němž odlučujeme odlišnosti a zvláštnosti, abychom zjistili obecné vlastnosti a vztahy. V našem případě to znamená uvažování o problému, aniž se zajímáme o způsob budoucí realizace (implementace), tj. přesné formulování CO má být řešeno, nikoli JAK to má být řešeno. Pojem abstraktní se vztahuje k tomu, co v dané chvíli považujeme za konkrétní. Nejvyšší úrovní abstrakce je systémová specifikace používající problémově orientovanou terminologii. Nejnižší úrovní je reprezentace problému pomocí řídicích a datových struktur implementačního jazyka, tedy použití implementačně orientované terminologie.

Princip abstrakce se uplatňuje ve Wirthově metodě postupného zjemňování, v metodě označované jako modulární programování a v použití abstraktních datových typů. Všechny tři metody ke svému vyjádření používají koncept programová jednotka neboli modul.

V programovacích jazycích se abstrakce realizuje v případě řídicích struktur pomocí pojmenování posloupnosti příkazů (podprogram, procedura, funkce) - procedurální abstrakce. Pojmenováním činnosti ukrýváme způsob zpracování. Vyšší úrovní abstrakce je pojmenování podproblému, ucelené činnosti, která může sestávat z několika procedur. V tomto případě musí jazyk umožňovat implementaci modulů.

V případě datových struktur je vazba na konkrétní technické vlastnosti počítače silnější. Moderní programovací jazyky podporují datové struktury pole, záznam, množina,... a mechanismus dynamického přidělování paměti. Skutečná datová abstrakce však opět něco ukrývá, a to reprezentaci dat. To vede k pojmu abstraktní datový typ, k jehož realizaci stejně jako v případě procedurální abstrakce slouží modul.

5. Postupné zjemňování a modularizace

Metoda postupného zjemňování (poprvé popsaná N. Wirthem) spočívá v následujícím:
(1) Rozlož problém na podproblémy.
(2) Uvažuj o každém problému samostatně a rozlož jej opět na podproblémy.
(3) Pokračuj tak dlouho, dokud nebudou problémy tak jednoduché, že může být formulován algoritmus jejich řešení.

Každý další rozklad je zjemněním předchozího rozkladu a vnáší do formulace další detaily až do okamžiku, kdy může být řešení formulováno v požadovaném implementačním jazyce. Velký programový systém je rozkládán na stále menší moduly podle principu abstrakce, takže

algoritmus ani reprezentace zpracovávané informace podproblému nejsou na dané úrovni zjemnění podstatné.

Modul je soubor činností (funkcí) a dat, které řeší jistý uzavřený problém, modul komunikuje se svým okolím (tj. s ostatními moduly) pouze prostřednictvím dobře definovaného rozhraní a jeho začlenění do většího programového systému může být provedeno bez znalosti jeho vnitřní struktury a jeho správnost může být ověřena bez úvah o začlenění do většího programového systému. Z pohledu principu abstrakce můžeme rozhraní modulu chápat jako specifikaci modulu. Obsahuje všechny informace týkající se toho, CO modul provádí.

Je zřejmé, že modulární návrh softwaru
- redukuje složitost (dekompozice na podproblémy),
- zrychluje implementaci a testování (dobře navržený modul je při dodržení rozhraní nezávislý na ostatních modulech)
- usnadňuje modifikaci (modul soustřeďuje spolu související prvky).

V programovacím jazyce Modula-2 je koncept modulu realizován rozdělením na dvě části - definiční modul, který určuje rozhraní, tj. vše, co je zvenčí viditelné, a implementační modul, který je popisem realizace. Definiční a implementační modul tvoří logickou jednotku (mají stejné jméno), ale jsou samostatně kompilovatelné. V definičním modulu jsou navíc explicitně v seznamu EXPORT vyjmenovány nabízené identifikátory a v seznamu IMPORT vyjmenovány identifikátory používané z jiných modulů (datové typy, procedury, proměnné). Definiční modul umožňuje popsat rozhraní přesně a přitom dostatečně abstraktně. Syntax definičního modulu z jazyka Modula-2 lze použít i během návrhu bez ohledu na to, v jakém jazyce bude potom programový systém implementován.

V jazyce Pascal se modul vyjadřuje pomocí jednotky - unit, která je součástí některých implementací jazyka (např. Turbo Pascal fy Borland). Unit se skládá ze sekce INTERFACE, která obsahuje rozhraní modulu (tj. export) včetně vyjmenování používaných unitů (tj. import, ale bez explicitního vyjmenování), a ze sekce IMPLEMENTATION, která obsahuje detailní realizaci.

V jazyce C je modulem soubor definicí funkcí a proměnných překládaný do souboru s verzí .OBJ. Specifikaci modulu je odpovídající hlavičkový soubor obsahující seznam úplných funkčních prototypů.

Přesný syntaktický zápis modulů v jednotlivých jazycích je uveden v konkrétním příkladu v části 6.

6. Abstraktní datový typ

Pomocí modulů se dá přirozeně realizovat abstraktní datový typ (ADT). Abstraktní datový typ má své jméno a je určen operacemi, které lze aplikovat. Jeho konkrétní reprezentace je skryta.

Jako příklad je použit abstraktní datový typ zásobník s typickými operacemi.

V jazyce Modula-2 je abstraktní datový typ specifikován pomocí definičního modulu:

```
DEFINITION MODULE ADT_Zasobnik;
(* specifikace abstraktního datového typ Zásobník*)
```

```
 EXPORT QUALIFIED typ_zas,NovyZas,Push,Pop,Prazdny
 type typ_zas; (*nezřetelný (opaque) typ*)
 PROCEDURE NovyZas(VAR zas:typ_zas);
   (*generuje novou instanci struktury zásobník*)
 PROCEDURE Push(zas:typ_zas; prv:INTEGER; VAR ind:BOOLEAN);
   (*vkládá do zásobníku zas prvek prv,
   ind=False, je-li zásobník plný, ind=True jinak*)
 PROCEDURE Pop(zas:typ_zas; VAR prv:INTEGER; VAR ind:BOOLEAN);
   (*načítá ze zásobníku zas poslední prvek do prv,
   ind=False, je-li zásobník prázdný, ind=True jinak*)
 PROCEDURE Prazdny(zas:typ_zas):BOOLEAN;
   (*vrací hodnotu True, je-li zásobník zas prázdný,
   jinak False*)
END ADT_Zasobnik;
```

Abstraktní datový typ má jméno typ_zas a je definován jako nezřetelný (opaque) datový typ, což znamená, že jeho reprezentace zůstává skryta, ale přitom jej lze po přeložení definičního modulu používat v celém programovém systému. Nezřetelný datový typ má tu nevýhodu, že v době kompilace definičního modulu nejsou známy paměťové požadavky. Proto je v jazyce Modula-2 abstraktní datový typ vždy implementován jako typ ukazatel. Jiná datová struktura musí být generována dynamicky. Implementace je sice trochu méně efektivní, ale zato celá realizace je soustředěna do implementačního modulu.

```
IMPLEMENTATION MODULE ADT_Zasobnik;
(*implementace abstraktního datového typu Zásobník*)
 FROM Storage IMPORT ALLOCATE;
 CONST maxpocet  = 20;
 TYPE typ_zas    = POINTER TO typ_zasins;
     typ_zasins = RECORD
                  pocet:CARDINAL;
                  prvek:ARRAY[1..maxpocet] of INTEGER;
 PROCEDURE NovyZas(VAR zas:typ_zas);
 BEGIN
  NEW(zas);
  zas^.pocet:=0;
 END NovyZas;
 PROCEDURE Push(zas:typ_zas; prv:INTEGER; VAR ind:BOOLEAN);
 BEGIN
  IF zas^.pocet<maxpocet
    THEN
      zas^.pocet:=zas^.pocet+1;
      zas^.prvek[zas^.pocet]:=prv;
      ind:=TRUE;
    ELSE
      ind:=FALSE;
  END;
 END Push;
 PROCEDURE Pop(zas:typ_zas; VAR prv:INTEGER; VAR ind:BOOLEAN);
 BEGIN
  IF zas^.pocet>0
```

```
    THEN
      prv:=zas^.prvek[zas^.pocet];
      zas^.pocet:=zas^.pocet-1;
      ind:=TRUE;
    ELSE
      ind:=FALSE;
    END;
 END Pop;
 PROCEDURE Prazdny(zas:typ_zas):BOOLEAN;
 BEGIN
  RETURN zas^.pocet=0;
 END Prazdny;
END ADT_Zasobnik.
```

V jiném modulu může být abstraktní datový typ zásobník použit následujícím způsobem:

```
IMPLEMENTATION MODULE X;
 FROM ADT_Zasobnik IMPORT typ_zas, NovyZas, Push, Pop, Prazdny;
 ...
 VAR z1,z2,z3: typ_zas;
 ...
 BEGIN
  NovyZas(z1); NovyZas(z2); NovyZas(z3);
  ...
 END X.
```

V jazyce Pascal je jak specifikace tak implementace modulu v jednom unitu. Zvláštní unit (TypZasU) je však použit k vyjádření nezřetelného typu jazyka Modula-2.

```
unit ADT_Zasobnik;
INTERFACE
 uses TypZasU;      {definuje typ_zas}
 procedure NovyZas(var zas:typ_zas);
 procedure Push(zas:typ_zas; prv:Integer; var ind:Boolean);
 procedure Pop(zas:typ_zas; var prv:Integer; var ind:Boolean);
 function  Prazdny(zas:typ_zas):Boolean;
IMPLEMENTATION
 procedure NovyZas(var zas:typ_zas);
 begin
  New(zas);
  zas^.pocet:=0;
 end{NovyZas};
 procedure Push(zas:typ_zas; prv:Integer; var ind:Boolean);
 begin
  if zas^.pocet<maxpocet then
  begin
   zas^.pocet:=zas^.pocet+1;
   zas^.prvek[zas^.pocet]:=prv;
   ind:=True;
  end
```

```
  else
    ind:=False;
 end{Push};
 procedure Pop(zas:typ_zas; var prv:Integer; var ind:Boolean);
 begin
  if zas^.pocet>0 then
  begin
    prv:=zas^.prvek[zas^.pocet];
    zas^.pocet:=zas^.pocet-1;
    ind:=True;
  end
  else
    ind:=False
 end{Pop};
 function  Prazdny(zas:typ_zas):Boolean;
 begin
   Prazdny:=zas^.pocet=0;
 end{Prazdny};
begin
end.
```

Unit TypZasU definuje používané typy. Kvůli jednotnosti příkladu je ADT také implementován jako ukazatel, ale v Pascalu to není podmínkou. Změna realizace ADT znamená modifikaci dvou unitů - TypZasU a ADT_Zasobnik.

```
unit TypZasU;
INTERFACE
const maxpocet = 20;
type typ_zasdins = record
                     pocet: Byte;
                     prvek: Array[1..MAXPOCET] of Integer;
                   end;
type typ_zas  = ^typ_zasins;
IMPLEMENTATION
begin
end.
```

Použití abstraktního datového typu zásobník, znamená uvedení obou unitů v sekci uses, oproti Modula-2 nejsou explicitně vyjádřeny používané identifikátory:

```
uses TypZasu,ADT_Zasobnik;
 ...
 var z1,z2,z3: typ_zas;
 ...
 begin
   NovyZas(z1); NovyZas(z2); NovyZas(z3);
   ...
 end.
```

V jazyce C podobně jako v Pascalu jsou použité typy soustředěny do samostatné části, v tomto případě hlavičkového souboru typzash.h. Vzhledem k tomu, že v C jsou všechny parametry předávány hodnotou, jsou zde definovány i ukazatelové typy pro realizaci výstupních parametrů. Z důvodů jednotnosti příkladu jsou použity funkce bez návratové hodnoty (typ void).

```
/*ADT_Zas.h - Specifikace
 *          abstraktního datového typu zásobník
 */
#include "typzash.h"          //specifikace použitých typů
void NovyZas(typ_zas_p zas_p);
void Push(typ_zas zas, int prv, int_p ind_p);
void Pop(typ_zas zas, int_p prv_p, int_p ind_p);
Prazdny(typ_zas zas);
```

```
/*ADT_Zas.C - Implementace operací
 *          abstraktního datového typu zásobník
 */
#include <stdlib.h>
#include "typzash.h"          //specifikace použitých typů
void NovyZas(typ_zas_p zas_p)
{
 *zas_p = (typ_zas)malloc(sizeof(typ_zasins));
}/*NovyZas*/
void Push(typ_zas zas, int prv, int_p ind_p)
{
 if (zas->pocet < MAXPOCET)
 {
  zas->pocet++;
  zas->prvek[zas->pocet] = prv;
  *ind_p = 1/*TRUE*/;
 }
 else
  *ind_p = 0/*FALSE*/;
}/*Push*/
void Pop(typ_zas zas, int_p prv_p, int_p ind_p)
{
 if (zas->pocet > 0)
 {
  *prv_p = zas->prvek[zas->pocet];
  zas->pocet--;
  *ind_p = 1/*TRUE*/;
 }
 else
  *ind_p = 0/*FALSE*/;
}/*Pop*/
Prazdny(typ_zas zas)
{
 return (zas->pocet == 0);
}/*Prazdny*/
```

```
/*typzash.h - specifikace použitých typů
 *          abstraktního datového typu zásobník
 */
#define MAXPOCET 20
typedef struct{              //typ instance zásobník
      int pocet;
      int prvek[MAXPOCET];
    }typ_zasins;
typedef typ_zasins *typ_zas;  //typ zásobníku jako ukazatele
                              //na typ instance zásobník
typedef typ_zas *typ_zas_p;   //typy výstupních parametrů
typedef int     *int_p;
```

7. Závěr

Konstrukce modulů a abstraktního datového typu je jazykem Modula-2 přímo podporovaná, specifikace a implementace jsou jasně odděleny, provádí se typová kontrola při překladu. Definiční modul svým explicitním popisem importních a exportních identifikátorů může být i dobrou notací návrhu a dokumentace. V Pascalu je modul vyjadřován zprostředkovaně pomocí unitu, provádí se typová kontrola při překladu. V jazyce C lze specifikaci modulu realizovat hlavičkovým souborem, rozsah typové kontroly záleží na programátorovi.

Vývoj programovacích jazyků s přímou podporou abstraktních datových typů vede k objektově orientovaným jazykům (Smalltalk, C++).

Literatura

[1] Herout, P.: Učebnice jazyka C, nakladatelství KOPP, 1993
[2] Honzík, M.J.: Vybrané kapitoly z programovacích technik, skripta, Fakulta elektrotechnická, Vysoké učení technické, Brno, 1991
[3] Pomberger, G.: Software Engineering and Modula-2, Prentice Hall, 1986
[4] Standish, T.A.: Data Structures, Algorithms and Software Principles, Addison Wesley, 1994

Autor: RNDr. Jarmila Holcová
katedra informatiky a počítačů
PřF, OU Ostrava
Bráfova 7, Ostrava 1
tel.: (069) 222808 kl.230
e_mail: holcova@oudec.osu.cz