M O D U L Á R N Ě N E B O S T R U K T U R O V A N Ě ?

Josef Tvrdík, KNS Ostrava

1. Úvodní poznámka

V celém příspěvku je strukturované programování (dále SP) chápáno v "širším" smyslu , zejména jako přítomnost vhodných řídících struktur - viz např. /1,2/, nikoliv ve smyslu Jacksonovy technologie /3/. Modulárního programování (dále MP) se týká práce /4/, dokazování programů práce /5/. Většina pojmů a tvrzení v tomto příspěvku patří nyní už spíše k programátorskému folklóru. Pouze pro pořádek jsou uvedeny odkazy na články v minulých sbornících semináře Programování. Tam zájemce může nalézt rozsáhlejší přehled literatury.

2. Záměrná nedorozumění

V souvislosti s programovacími technikami dochází občas k dlouhým a většinou bezcílným diskusím o vztahu modulárního a strukturovaného programování - viz např. i diskuse na semináři Programování 84. Někdy dokonce stoupenci jedné techniky (nebo spíš přístupu) popírají právo na existenci přístupu druhému. Příčiny těchto nedorozumění jsou zejména v tom, že se
- mluví o stejných věcech různými slovy
- mluví o různých věcech stejnými slovy
- zdůrazňují drobné a malicherné rozpory
- opomíjejí podstatné shody.

A tak se vedou řeči o dobře strukturovaných programech, o nutnosti programování bez GOTO, o nutnosti programování s GOTO a pod. Někdy zastánci SP vycházejí z názoru, že dobré programování je synonymem SP. Doporučují, aby se všechny dobré a prakticky osvědčené metody a postupy v tvorbě programů označily (byť třeba dodatečně) jako součásti SP. Pozor - výsledek tohoto přístupu může být pochybný - viz /6/. Leckdy argumentace pro SP vyznívá tak, jako by všechny rozumné praktiky v programování (postupná dekompozice úlohy, zvládnutelný rozsah zdrojového textu programového modulu, přehledný zápis zdrojového programu, komentáře,

mnemotechnická jména atd.) byly až důsledkem aplikace vhodných (např. Dijkstrových) řídících struktur.

Naopak odpůrci SP často využívali chyb v propagaci SP k tomu, aby vyvolávali dojem, že SP je možné zcela zatratit. Např. postupným zjemňováním řešení se přibližujeme strojovému kódu, takže o programování bez GOTO nemůže být ani pomyšlení. Ostatně i běžně rozšířené vyšší programovací jazyky typu Fortran, Cobol, Basic nutně vyžadují využití bohatějších řídících struktur než dovoluje SP, a že tedy ani na této úrovni nelze o programování bez GOTO uvažovat. I velmi silný argument pro SP, totiž potenciální možnost automatizovaného dokazování správnosti programů, lze snadno zpochybnit. Jednak tím, že je to pro nejbližší budoucnost možnost opravdu jenom potenciální, jednak tím, že značná část "provozních" chyb programu je ovlivněna nároky na prostředky výpočetního systému a důkaz správnosti programu v těchto situacích mnoho nepomůže.

V dalším odstavci se pokusíme pohlédnout na MP a SP z poněkud smířlivějších pozic.

3. Některá východiska, cíle a principy SP a MP

SP a MP mají řadu společných vlastností. Především to, že ani SP ani MP není ucelenou technologií tvorby programového vybavení. Neposkytují řešiteli přesný návod, jak postupovat v každém kroku řešení jednotlivých fází vývoje programového vybavení. Spíše jenom z pozic "zdravého rozumu" poukazují na možné důsledky volby různého přístupu na vlastnosti výsledného produktu. SP i MP jsou převážně názorem na řešení programového vybavení, nikoliv návodem. Společný je i základní princip SP a MP, t.j. dekompozice problému na části, mezi kterými je zjevné hierarchické uspořádání a jsou jasně definována rozhraní. Velmi podobné jsou i cíle - bezchybné, přehledné, modifikovatelné, efektivní programy (u SP navíc dotazatelná správnost programů). SP i MP jsou podobné v tom, že nesvazují řešitele přísnými technologickými postupy a poskytují mu značnou volnost v uplatnění jeho invence. Proto jsou vhodné zejména tam, kde je tato invence žádoucí, t.j. při řešení rozsáhlých nestandardních úloh.

Rozdílná pro SP a MP jsou východiska a motivace. U SP je to motivace převážně matematická, opěrnými body i vznešeným cílem je ucelená axiomatická teorie. U MP jsou východiska a přístupy téměř výhradně technické (inženýrské). Inspirace vychází z jiných technických oborů, např. strojírenství, elektroniky, stavebnictví. Cílem je stavebnicově sestavený výrobek s možností opakovatelného využití modulů v různých propojeních (kontextech). Rozdíly přístupu a výsledků MP oproti jiným technickým oborům jsou způsobeny kratší historií softwarového inženýrství, zvláštní povahou programu jeho výrobku (program se opakovaným používáním neopotřebovává) a snad větší složitostí programových produktů (viz např. /7/).

Další rozdíl mezi SP a MP je v převládajícím způsobu uvažování. V MP naprosto převažuje myšlení funkční. Funkční dekompozice probíhá do takové podrobnosti, aby procedurální složka řešení (vyřčení logiky modulu) byla už téměř triviální. SP je soustředěno především na algoritmus úlohy, uvažování je převážně procedurální.

Nejen společné vlastnosti SP a MP, ale především rozdíly mezi nimi poskytují vhodný základ pro kombinované využití přístupů SP a MP ve vývoji programového vybavení. Pro takové komplementární využití SP a MP není podstatné administrativní rozhodnutí o vzájemné nadřazenosti a podřízenosti pojmů SP a MP, ale uvážlivý přístup k řešení, otevřenost vůči praktickým požadavkům, zkušenostem a vůči novým teoretickým poznatkům.

4. Příběh o zápočtovém příkladu

Čtyři studenti - René, Wilhelm, Gabo a Standa - skládali zápočet z předmětu "Praktické strukturované programování". Jejich úkolem bylo napsat ve Fortranu (F4P na počítači SMEP) část programu podle následujícího zadání.

Zadání: Napište část programu pro výběr z nabízených možností (menu). Požaduje se, aby program vypsal menu na obrazovku; přečetl uživatelovu volbu; zkontroloval, zda uživatelova volba je dovolená; při nedovolené volbě vypsal upozornění a umožnil opravu; při dovolené volbě předal řízení do úseku programu,

ve kterém se provede zvolená funkce; po vykonání této funkce ukončit běh programu.

Pokyn pro řešení: Předpokládejte, že menu má 3 položky. Jim odpovídající moduly (podprogramy) F1, F2, F3 jsou již implementovány a nemají žádné vstupní/výstupní parametry.

Každý student se chopil tužky a papíru a začal rychle zapisovat algoritmus v pseudokódu.

René usoudil, že charakter případného opakování čtení uživatelovy volby lze nejlépe vystihnout řídící strukturou REPEAT UNTIL (tělo cyklu se provádí nejméně jednou). Algoritmus zapsal stručně, přehledně a jednoduše bez použití GOTO - viz obr. 1R.

```
piš menu
REPEAT UNTIL i > 0 & i < 4
  čti i
  IF i < 1 nebo i > 3
    THEN piš upozornění!
  ENDIF
ENDREPEAT
CASE i OF S1, S2, S3
S1: F1
S2: F2
S3: F3
ENDCASE
```

Obr. 1R: Zápis s REPEAT UNTIL

Wilhelm během semestru na cvičení hodně chyběl a tak znal pouze 3 základní řídící struktury - sekvenci, větvení IF THEN ELSE a cyklus WHILEDO. Po krátkém zamyšlení a drobných obtížích s nutností nastavit předem hodnotu proměnné (podmínky cyklu) i on napsal algoritmus zadané úlohy bez použití GOTO, viz obr. 1W.

```
piš menu
i = 0
WHILE i < 1 nebo i > 3 DO
  čti i
  IF i < 1 nebo i > 3
    THEN piš upozornění
  ENDIF
ENDWHILE
CASE i OF S1, S1, S3
S1: F1
S2: F2
S3: F3
ENDCASE
```

Obr. 1W: Zápis s WHILE DO

Gabo usoudil, zadaná úloha je tak jednoduchá a průhledná, že zadání je vlastně téměř řešením. Použití příkazu skoku přirozeně vyplývá ze zadání a správnost takového programu jistě by někoho nenapadlo dokazovat formálními prostředky - GOTO použil s naprostým klidem - viz obr. 1G.

Standa si vzpomněl, že na nějakém cvičení probírali řídící strukturu SELECT (nebo CASE), která obsahovala i větev ELSE. Usoudil, že využití této struktury a ošetření nedovolené volby z menu právě ve větvi ELSE je nejpřirozenějším a strukturovaným řešením zadané úlohy. Při zápisu algoritmu se dostal do nesnází, ale nakonec je vyřešil použitím GOTO ve větvi ELSE - obr. 1S.

Všichni čtyři studenti pak navržené algoritmy přepsali na terminálu do Fortranu a otestovali správnou funkci svých programů. Zvláštní shodou náhod napsali všichni studenti první část programu (výpis menu na obrazovku) naprosto stejně (obr.2), druhá část jejich řešení je uvedena na obr. 3R, 3W, 3G a 3S.

```
        piš menu
L1:     čti i
        IF i < 1 nebo i > 3
          THEN piš upozornění
               GO TO L1
        ENDIF
        CASE i OF S1, S2, S3
S1:     F1
S2:     F2
S3:     F3
        END CASE
```

Obr. 1G: Zápis s GO TO

```
        piš menu
L1:     čti i
        CASE i OF S1, S2, S3, ELSE
S1:     F1
S2:     F2
S3:     F3
ELSE:   piš upozornění
        GO TO L1
        ENDCASE
```

Obr. 1S: Zápis s CASE (SELECT) a ELSE

```
      TYPE 1000
1000  FORMAT (' PRIKLAD O VYBERU Z MENU:'
     1         / 5X,'1 ... FUNKCE 1'
     2         / 5X,'2 ... FUNKCE 2'
     3         / 5X,'3 ... FUNKCE 3')
2000  FORMAT (x,' VOLBA ? ')
3000  FORMAT (' *** NEDOVOLENA VOLBA, ZKUS ZNOVA')
```

Obr. 2: Úvodní část programu

```
   5  TYPE 2000
      ACCEPT *, I
      IF (I.GE.1 .AND. I.LE.3)  GO TO 8
        TYPE 3000
        GO TO 5
   8  GO TO (10,20,30) I
  10  CALL F1
      GO TO 50
  20  CALL F2
      GO TO 50
  30  CALL F3
  50  STOP
```

Obr. 3R, 3G: Druhá část programu podle 1R, 1G

```
      I = 0
   5  IF (I.GE.1 .AND. I.LE.3)  GO TO 8
      TYPE 2000
      ACCEPT *, I
      IF (I.GE.1 .AND. I.LE.3)  GO TO 5
        TYPE 3000
        GO TO 5
   8  GO TO (10,20,30) I
  10  CALL F1
      GO TO 50
  20  CALL F2
      GO TO 50
  30  CALL F3
  50  STOP
```

Obr. 3W: Druhá část programu podle 1W

```
   5  TYPE 2000
      ACCEPT *, I
      GO TO (10,20,30) I
      TYPE 3000
      GO TO 5
  10  CALL F1
      GO TO 50
  20  CALL F2
      GO TO 50
  30  CALL F3
  50  STOP
```

Obr. 3S: Druhá část programu podle 1S

Vyučující prohlédnul výsledná řešení a zkontroloval, zda René a Gabo od sebe neopisovali (programy 3R, 3G jsou shodné). Naštěstí žádný z nich dosud nevyhodil zápis řešení v pseudokódu - obr. 1R, 1G, a tak mohli prokázat původnost svých přístupů. Pak studenti i asistent prodiskutovali společně všechna čtyři řešení, studenti dostali zápočty, asistent pochválil jako nejjednodušší a nejstručnější řešení Standovo - 3S, a do karty ke zkoušce mu zapsal plus. Wilhelmovo řešení (3W) označil za kostrbaté a do karty mu zapsal mínus. Všichni se rozešli, každý s nějakým námětem k přemýšlení:

R: Moje řešení bylo lepší než 3S, to příliš využívá možností tohoto kompilátoru Fortranu. Navíc vzniklo ze špatně strukturovaného návrhu.

W: Zbytečná složitost a nepřehlednost programů není způsobena jen užitím příkazu skoku. Je potřeba užívat ty řídící struktury, které odpovídají řešenému problému. Jak to ale vždycky rozpoznat ?

G: Renému bez GOTO nakonec vyšel stejně dobrý jako mně s GOTO. Na tom strukturovaném programování bez GOTO přece jenom asi něco bude.

S: Měl jsem štěstí, že jsem znal detailně způsob překladu přepínače tímto kompilátorem. Při použití jiného kompilátoru by řešení nemuselo být tak jednoduché. Samotná dobrá znalost programovacího jazyka nemusí vždycky dostačovat k přehlednosti a jednoduchosti řešení.

Vyučující: Ta hodnocení (plus a mínus) při zkoušce nemohu brát příliš vážně, jsou spíše vyjádřením okamžitého subjektivního pocitu než objektivním měřítkem. Na jednom jednoduchém příkladu nelze mnoho poznat. Hlavně, aby se naučili rozumně řešit i složitější problémy.

Vypravěč fiktivního příběhu : Ještě, že vyučující zadal tak jednoduchý příklad. Jinak by asi ten příběh vůbec neměl konec.

Čtenář (?): Spojení přístupů SP a MP v řešení vývoje programového vybavení je velmi prosté a přirozené, vždyť studenti s tím neměli žádné potíže.

Literatura:

1. Brzický J., Strukturové programování a prvé skušenosti s jeho použitím v jazyce PL/1, sborník Metody programování počítačů III. generace, DT ČSVTS Ostrava 1975

2. Hořejš J., 30 let strukturovaného programování, sborník Programování 84, DT ČSVTS Ostrava 1984

3. Vondráček B., Kretschmer M., Drbal P., Suchomel J., Technologie strukturovaného programování, sborník Programování 80, DT ČSVTS Ostrava 1980

4. Čimbura V., Tvrdík J., Problémy návrhu modulárních programů, sborník Programování 80, DT ČSVTS Ostrava 1980

5. Honzík J., Dokazování programu, sborník Programování 83, DT ČSVTS Ostrava 1983

6. Čapek J., Povídání o pejskovi a kočičce, str. 79-88, SNDK Praha, 1968

7. Bloudil J.M., O programátorství, Výběr informací z organizační a výpočetní techniky č. 4, str. 545, 1980