O JEDNOM RIZIKU POSTUPU SHORA - DOLŮ

RNDr. Ing. Ivan Lexa CSc. , OZS Břeclav

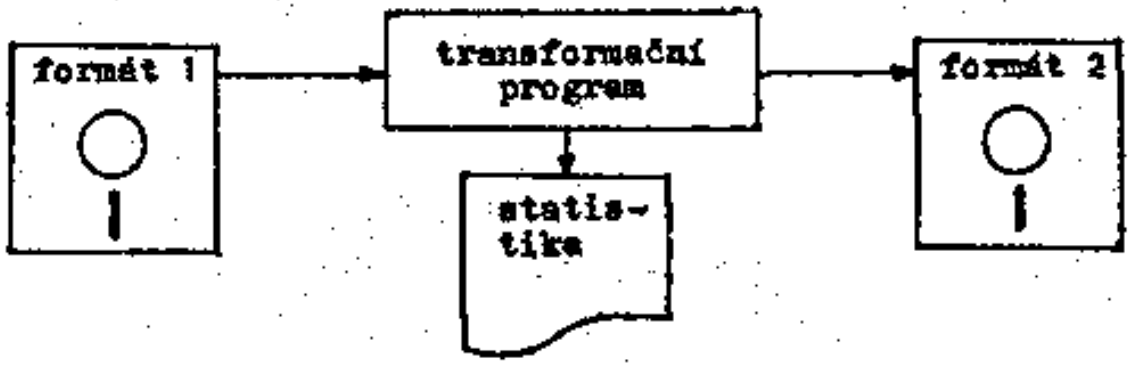
Shora-dolů jako po másle

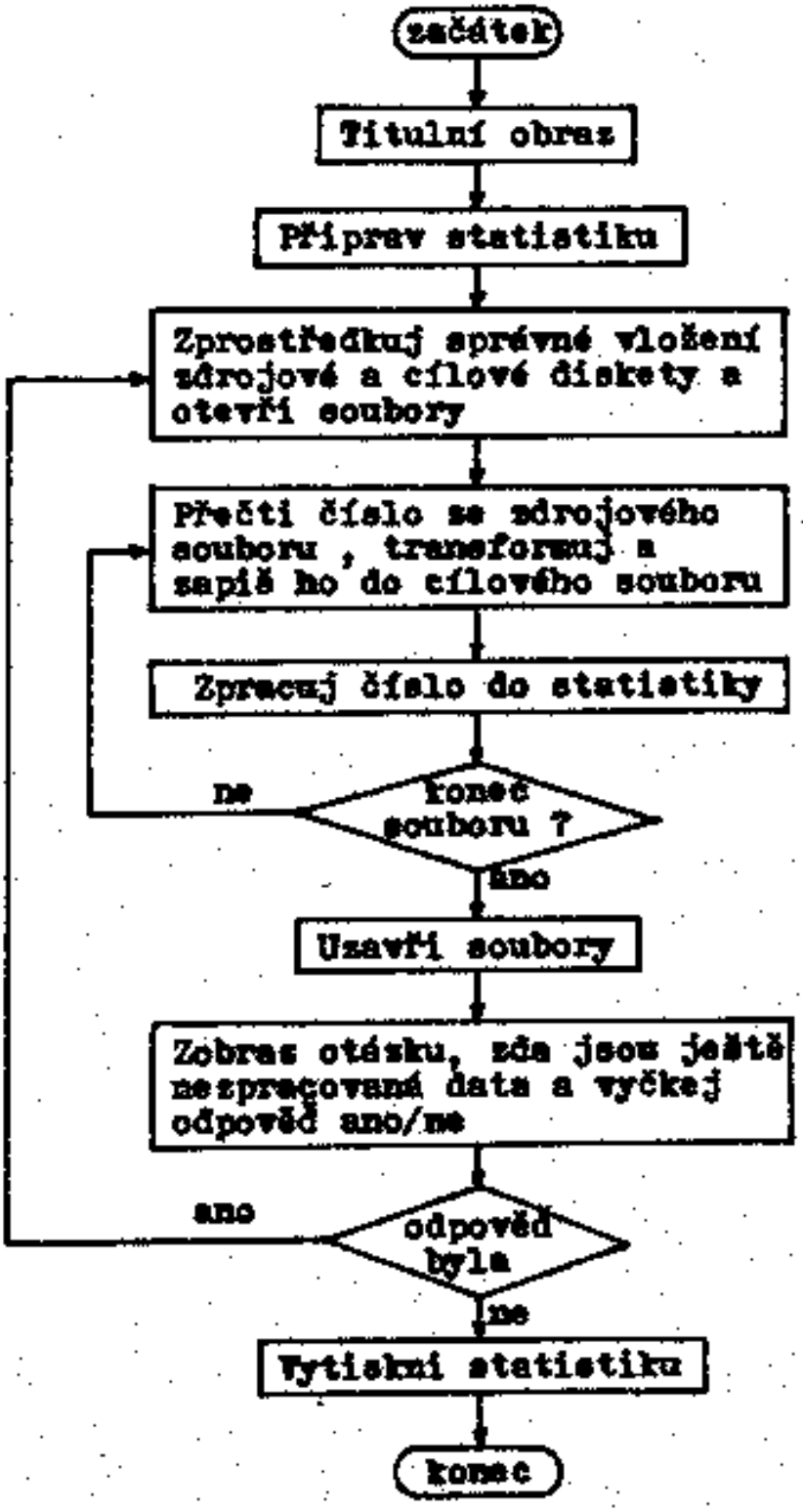
Důsledný postup shora-dolů má obrovskou přednost: Složitý úkol převedeme na několik jednodušších a ty předložíme řešitelům. Řešitelé pracují podle dílčích (již jednodušších) zadání a nejsou zatěžováni informacemi o celém problému. Složitý dílčí úkol lze další dekompozicí převést na ještě jednodušší úkoly. Řešitelé ručí pouze za splnění dílčích úkolů a pokud úkoly splnili, je tím automaticky docíleno vyřešení celého úkolu.

Postup shora-dolů nám tedy umožní udělat ze složitého jednoduché a z nezvládnutelného zvládnutelné. Takové "dekompoziční opojení" se dnes značně rozšířilo, zejména u studentů a čerstvých absolventů škol. Ukažme dále na třech příkladech jedno úskalí, které tato lákavá idea v sobě skrývá.

Příklad 1 - transformace dat

Jak to začalo :
Autor X dostal udělat program na transformaci dat z jednoho formátu do druhého (vždy jeden soubor na jedné disketě). Vedlejším produktem měl být tištěný protokol o statistických vlastnostech přetransformovaných číselných údajů ze všech zpracovaných disket. Postupem shora-dolů se dostal až do následujícího stupně rozpracování úlohy





Vzhledem k tomu, že se statistikou měl málo skušeností, rozhodl se (správně), že příslušné algoritmy přenechá svému podřízenému Y. Zadal mu tedy úkol vytvořit v jazyce Pascal tři procedury

```
procedure PripravStatistiku
procedure ZpracujCislo ( Cislo : integer )
procedure TiskStatistiky
```

i s potřebnými globálními deklaracemi pro tyto procedury.
X i Y měli pocit, že zadání je zcela jasné.

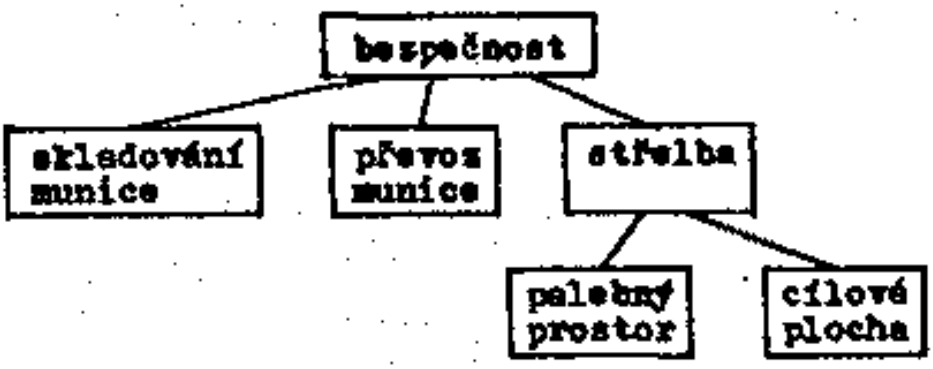
Jak to dopadlo :
Asi za měsíc se Y chlásil, že mu statistika perfektně funguje a ukázal několik "sestav", vytvořených na základě simulovaných posloupností čísel. V té době měl i X svůj program již odladěn bez statistiky (3 procedury byly nahrazeny atrapami).

Při pokusu o skloubení obou výtvorů však došlo ihned k rozčarování. Y potřeboval pro zvládnutí úkolu tak velké pole, že se do operační paměti použitého mikropočítače nemohlo vejít a použil (zcela logicky) disketového souboru. Protože X takové možnost vůbec nenapadla, nepovažoval za potřebné informovat Y o tom, že v obou disketových jednotkách (více jich počítač neměl) se během práce programu několikrát vymění datové diskety a že tedy pracovních souborů používat nesmí.

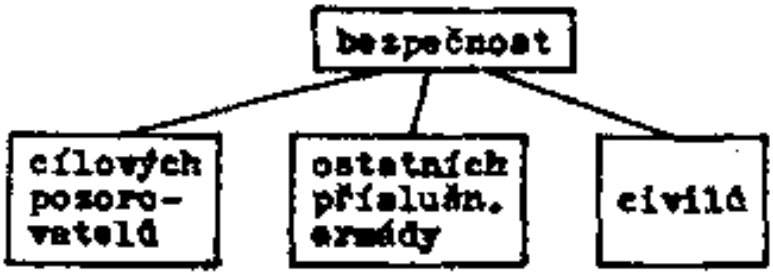
Dílčí závěr :
X podcenil náročnost svého úkolu a nemaje dostatečné zkušenosti stanovil pro Y nepřesné zadání.

Příklad 2 - bezpečnost naživo

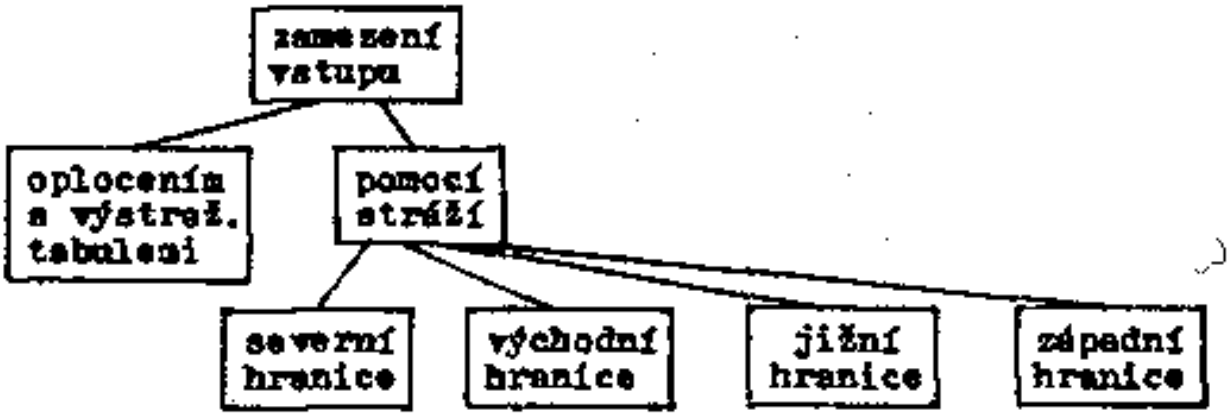
Jak to začalo :
Velitel dělostřelecké střelnice řešil komplexně bezpečnost jejího provozu. V duchu vojenských tradic postupoval výhradně shora-dolů a první kroky jeho dekompozice vypadaly takto



Čtyřmi "dílčími bezpečnostmi" pak pověřil příslušné důstojníky. Důstojník X, který byl zodpovědný za bezpečnost na cílové ploše, postupoval v dekompozici dále



Bezpečností civilů pověřil svého podřízeného Y, přičemž jeho úkol formuloval jednoduše: zamezit přístup civilů na cílovou plochu. Y pak opět použil postup shora-dolů takto



a pověřil své podřízené dílčími úkoly ... . Jednou z posledních větví postupné dekompozice byl rozkaz pro vojína Z : V době od 12.00 do 14.00 zamezit proniknutí osob přes svěřený úsek plotu.

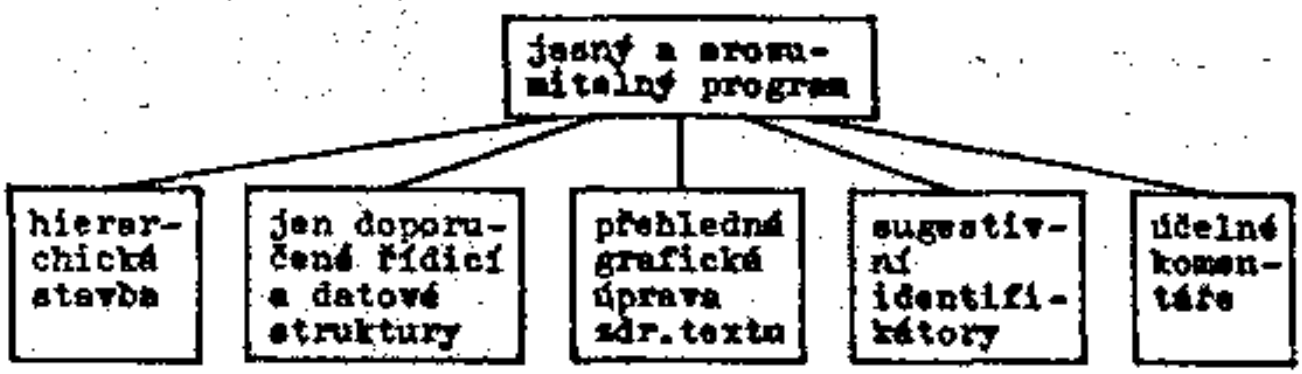
Jak to dopadlo :
Ve 13.24 zastřelil vojín Z civilní osobu, která prolezla pod plotem a ignorovala všechny předpisem stanovené výstrahy. Vyšetřováním se pak zjistilo, že občan byl zcela hluchý, že šel do objektu sbírat houby a že vojín Z neměl žádnou konkrétní informaci o tom, proč je plot hlídán a že je tedy (postupoval přesně podle předpisů) nevinen.

Dílčí závěr :
Přestože všechny kroky dekompozice vypadaly předem zcela správně, došlo k paradoxu - občan byl zastřelen vlastně proto, aby na cílové ploše nepřišel k úrazu! Při dodatečném prozkoumání postupného rozkladu nalezneme příčinu: Důstojník X svůj díl dekompozice nezvládl. Neuvědomil si, že výbuch dělostřeleckého granátu není jediným nebezpečím cílové plochy a formuloval úkol pro Y nesprávně.

Příklad 1 - recept na dobré programy

Jak to začalo :

Na první stadium revoluce, zvané "strukturované programování" lze nahlížet jako na vážný pokus vyřešit postupem shora-dolů obecný úkol "dělat dobré programy". První dekompoziční krok vypadal nejčastěji takto



Tento nepochybně záslužný čin v metodice programování se postupně proměnil v široké hnutí. Představa, že striktním dodržováním několika pravidel dokážeme dělat samé dobré programy byla natolik lákavá, že si získávala stále další přívržence a nadšence.

Jak to dopadlo :

Dnes se již ukázalo, že uvedená dekompozice není přesná (v metodických pracích se již nemluví o pravidlech, ale spíše o zásadách a doporučeních a připouští se výjimky). Navíc přesnou dekompozici už ani nikdo neslibuje. To však nebrání existenci armády dogmatiků, kteří jsou schopni například trestat každé použití příkazu skoku v programu. Ukázka v příloze (procedura v jazyce Pascal-Turbo) názorně ukazuje, že striktní trvání na absenci příkazu skoku může přehlednost programu podstatně zhoršit, což je opak toho oč jsme původně usilovali. (Zřejmě "kolekce" doporučených struktur zatím není úplná a chybějící "vhodné" struktury si musíme zatím pomocí skokového příkazu vytvářet). Další příklady lze nalézt v [1].

Dílčí závěr :

Aplikován na samotnou metodiku programování je postup shora-dolů tak náročný, že jeho hlavní výhodu (t.j. úplnou přeměnu prvotního cíle na dílčí cíle) zatím nemůžeme využít.

Zobecňující druhy

1. Jeden dekompoziční krok postupu shora-dolů můžeme symbolicky vyjádřit grafem

Je-li tento krok vykonán zcela správně, pak můžeme v dalším postupu vyhlásit prostředky 1 až n za nové cíle. Jinak řečeno, nutností úspěšného postupu shora-dolů je systematická přeměna prostředků v cíle a možnost na původní cíle "zapomenout".

A nyní pohleďme na trošku věc z úplně jiné stránky. V historii lidstva bylo až příliš častou příčinou neúspěchu různých snažení, hnutí, organizací i celých ideologií právě to, že prostředky byly postupně povýšeny na cíle a na původní cíle se "zapomělo"!

Jak se vypořádat s tímto paradoxem ? Je jediné logické východisko: narazíme-li s plným úspěchem povýšit prostředky na cíle, vyplývá z toho, že příslušný dekompoziční krok není korektní! To konec-konců potvrdily i tři uvedené příklady.

Tak se dostáváme k následujícímu názoru: Na řešení obtížných úkolů budou nejúčinnější právě první dekompoziční kroky. I sebemenší nepatrná (a předem těžko předvídatelná) chyba v tomto rozkladu může mít později zdrcující následky v nesplnění až popření původního cíle!

2. V dekompozičním kroku musíme obecně respektovat dva druhy vazeb: vazby mezi cílem a prostředky a vazby mezi prostředky navzájem. Největší riziko při dekompozici je právě v tom, že ve složitějších situacích nějakou vazbu opomeneme uvažovat nebo zatím nevíme o její existenci. V příkladě 1 byl takovou vazbou omezený počet diskretových jednotek, v příkladě 2 to bylo ohrožení bezpečnosti sousedních stránek.

V programování mohou být opomíjeny zejména tyto druhy vazeb :
- ovlivňování obsahu společných proměnných nebo registrů
- ovlivňování obsahu společné vnější paměti

- ovlivnění stavu sdílené periferie
- skrytá vazba mezi objekty řízeného procesu
- skrytá vazba mezi různými uživateli téhož počítačového systému
- vyčerpání kapacity společného zdroje (operační nebo vnější paměť, počet periferních jednotek, přidělovaný čas, ...)

3. Pokud celý úkol řeší jeden člověk, je riziko opomenutí některé vazby menší (většinou ho to "trkne"). Také bývá pravidlem, že na opomenutí přijde poměrně brzy, takže negativní dopad je obyčejně únosný. Horší je již situace v malých kolektivech. I zde se však častými poradami a vzájemným informováním o stavu prací dá předejít "příliš drahým" omylům.

Nejhorší předpoklady má úkol, na jehož řešení se podílí větší množství pracovníků nebo dokonce odlehlých pracovišť. V takové situaci bývá skrytá chyba v prvních dekompozičních krocích odhalována až s velkým zpožděním, takže ztráty ze zbytečně vynaložené práce mohou celý projekt neúnosně prodražit a zdržet.

Ve shodě s [2] můžeme tedy očekávat výrazný růst rizika s přibývajícím počtem řešitelů úkolu.

Shora-dolů střízlivě a zodpovědně

Cílem příspěvku bylo ukázat, že ačkoliv je postup shora-dolů všeobecně považován za jedinečný prostředek, který ze složitého dělá jednoduché, přesto dekompozice složitých zodpovědných úkolů nepatří do rukou začátečníka. Na provedení konkrétní dekompozice neexistuje žádný "recept" (máme pouze velmi obecná doporučení) a jsou to nakonec zkušenosti s podobnými úkoly, které nás chrání před vážnými omyly.

Chybějící zkušenosti pracovníka, který vede řešení rozsáhlejšího úkolu, se dají částečně kompenzovat častou výměnou informací a názorů mezi jednotlivými řešiteli nebo pracovišti a to jak v horizontálním, tak i ve vertikálním směru (rozpor v zadáních dílčích úkolů se tak objeví daleko dříve). Takový přístup sice chrání celý projekt před katastrofami, na druhé straně však snižuje jeho efektivnost, protože dílčí řešitel se nemůže plně soustředit pouze na řešení "svého" úkolu.

Použitá literatura

[1] Honzík J. a kol.: Programovací techniky ,
Slušovice 1986. /strana 268/

[2] Sokol J.: Kompletování a údržba velkých programových systémů, sborník Programování 87. /strany 13,14/

Příloha

Stavba procedury s vhodným využitím příkazu skoku :

```
procedure Zpracování_s_kontrolou ;
    label Chyba ;
    begin
        Zpracování_1 ;
        if Podmínka_1 then goto Chyba ;
        Zpracování_2 ;
        if Podmínka_2 then goto Chyba ;
        Zpracování_3 ;
        if Podmínka_3 then goto Chyba ;
        Zpracování_4 ;
        if Podmínka_4 then goto Chyba ;
        Zpracování_5 ;
        exit ;
              Chyba : Chybové_opatření_1 ;
                      Chybové_opatření_2 ;
                      Chybové_opatření_3
    end ;
```

a též procedura bez použití příkazu skoku :

```
procedure Zpracování_a_kontrolou ;
    procedure Chyba ;
        begin
          Chybové_opatření_1 ;
          Chybové_opatření_2 ;
          Chybové_opatření_3
        end ;
    begin
       Zpracování_1 ;
       if Podmínka_1
           then Chyba
           else begin
                  Zpracování_2 ;
                  if Podmínka_2
                     then Chyba
                     else begin
                            Zpracování_3 ;
                            if Podmínka_3
                               then Chyba
                               else begin
                                      Zpracování_4 ;
                                      if Podmínka_4
                                         then Chyba
                                         else Zpracování_5
                                    end
                          end
                end
    end ;
```