Ing. Alexander Kirschner
PVT, Bratislava

VYUŽITEĽNOSŤ VÝVOJOVÝCH BLOKOV PRI NÁVRHU PROGRAMOVÝCH MODULOV

Príspevok sa zaoberá vývojovými blokmi, ktoré vznikli rozvinutím diagramov Nassiho a Shneidermana. Po rozobratí typických problémov návrhu logiky programových modulov sa uvádza genéza vývojových blokov. Tretia časť príspevku je venovaná popisu vývojových blokov a ich aplikačných možností, ktoré sú na konkrétnom príklade ilustrované vo štvrtej časti. Zvlášť sa v príspevku poukazuje na súčasnú aplikáciu techniky vývojových blokov so štrukturovaným návrhom programových systémov.

V historicky nie dlhom období existencie výpočtovej techniky, hlavne po prieniku počítačov do sféry riadenia a správy, celkom prirodzene vystúpila do popredia otázka ich /aplikačného/ programového vybavenia, ktoré určuje využiteľnosť počítačov v jednotlivých oblastiach. S tým úzko súvisí úloha návrhu programov /programových systémov/.

V súčasnosti vykryštalizoval problém návrhu programov /keď odhliadneme od fázy analýzy problému, funkčnej špecifikácie programov resp. od analýzy požiadaviek/ do dvoch hlavných samostatných činností, a to

- modulového návrhu programu /programového systému/,
- návrhu modulu.

Problém modulového návrhu programov /programových systémov/ bol okrem iného rozobraný aj na minuloročnom seminári Programování '80 v príspevku [1]. Je tam popísaný jeden z možných prístupov, ktorí jeho pôvodní autori pomenovali súhrným návrhom /composite design/.

My sa v tomto príspevku zameriame na problém návrhu programového modulu za predpokladu, že je už vykonaný modulový návrh programu /programového systému/ s tým, že návrh modulu má priamo podporovať modulový návrh programu, má byť medzi nimi priamo dokumentovateľný vzťah a ďalej tento návrh modulu má byť v čo najtesnejšej väzbe so zdrojovým kódom modulu. Jednou takouto technikou je technika vývojových blokov, ktorá vychádza z diagramov Nassiho a Shneidermana.

Pod návrhom modulu budeme v tomto príspevku rozumieť návrh jeho logiku, čo je východisko pre kódovanie zdrojového textu modulu.

1. K návrhu programových modulov

V minulosti sme boli svedkami viacerých techník, ktoré sa používali na návrh programov. Snáď najtradičnejšou a najstaršou technikou sú aj dnes široko používané vývojové diagramy. Z iných manuálnych techník /ostatné techniky si tu všímať nebudeme/ len náznakou spomeňme napríklad metódu rozhodovacích tabuliek, Chapinove diagramy [2], Wittyho dimenzionálne diagramy [3] či Jacksonovu metódu návrhu programov / a jej analógie/, popísanú napr. aj v [4].

Nie je tu priestor pre detailnejšie porovnávanie možností jednotlivých techník. Skôr sa pokúsime sformulovať niektoré požiadavky, ktoré by mala istá metóda návrhu programu /modulu/ v súčasných podmienkach v prostredí riešenia úloh hromadného spracovania dát /ale nielen jeho/ spĺňať. S uvážením

dnes široko forsírovaných /a propagovaných/ "štruktúrnych technológií", ale hlavne imperatívu účinnosti a jednoduchosti návrhu modulu, sme toho názoru, že taká technika by mala okrem iného spĺňať nasledovné kritériá:

- podporovať metódy modulového návrhu programu /programového systému/,
- umožňovať tvorbu programového vybavenia princípom "zhora nadol",
- sledovať koncepciu štrukturovanosti programov,
- umožňovať jednoduchú dokumentovateľnosť,
- "zviditeľňovať" logiku modulu,
- byť nástrojom prípadného delenia práce navrhovateľa programu /modulu/ a kódovača, prípadne testovača.

Pre potreby tohoto článku si zavedieme niekoľko základných príkazov pseudokódu, aby naše zápisy boli čo najvšeobecnejšie. Okrem príkazu priradenia /=/ budeme používať príkazy vetvenia /if ... then ... else ...; select .../, ďalej príkazy cyklu /loop ... exitif; while ...; repeat ... until.../ a príkazy vstupu a výstupu /read; write/.

2. Vznik a rozvoj vývojových blokov

Technika vývojových blokov má základ v diagramoch Nassiho a Shneidermana /ďalej NS-diagramy/, ktorí svoj prístup formulovali v roku 1973 v článku [5]. Podstata ich návrhu spočívala vo voľbe štyroch základných blokov /ktoré oni pomenovali symbolmi/ logiky programu, ktoré zodpovedajú základným riadiacim štruktúram štrukturovaného programovania /kódovania/. Zaviedli procesný symbol, symbol rozhodovania, iteračný symbol a symbol BEGIN-END, ktoré sú zobrazené na obr. 1 zľava

doprava.

obr. 1

Do procesného bloku /symbolu/ sa vyznačuje jeden alebo viac sekvenčných príkazov. V bloku vetvenia označenie podm znamená /a aj ďalej bude znamenať/ podmienku vetvenia, F označuje nesplnenie podmienky, T označuje jej splnenie. V bloku iterácie označuje DO nejaký iteračný príkaz.

Tieto symboly možno do seba ľubovoľne vkladať, písať do nich príkazy jazyka, čím vzniká diagram logiky programu /modulu/. Ako je viedieť, tieto symboly nemožno deliť, čo vyžaduje modularizáciu programu, rovnako v nich neexistuje symbol odovzdania riadenia /GO TO/.

Autori vo svojom článku uviedli i návrh doplnkových symbolov pre iteráciu typu UNTIL, viacnásobné vetvenie a pre zobrazenie paralelizmu, no tie sa ujali menej. Svoju techniku pomenovali jazykom vývojových diagramov /flowchart language/, niekedy sa tieto diagramy označujú ako lineárne vývojové diagramy /linear flowchart/, prípadne ako NS-diagramy /NS-charts/. Myslíme, že posledné označenie je najvhodnejšie.

Zverejnenie [5] nevyvolalo žiadnu polemickú reakciu. Pripisujeme to jednoduchosti a bezospornosti ich schém, nie bezvýznamnosti príspevku autorov. Až v roku 1978 publikoval P. Grouse rozpracovanie tejto techniky [6], v ktorom išlo o jej generalizáciu a aktualizáciu, pričom pre analogické schémy

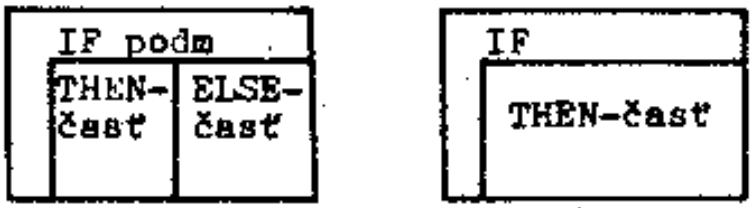
použil označenie flowblocks . Myslíme, že je vhodné prekladať tento termín ako vývojové bloky, keďže v týchto schémach sa zobrazuje predovšetkým logika programu a každý zo symbolov je akýmsi "stavebným blokom" programu /modulu/.

Nasledovná časť je venovaná popisu vývojových blokov a ich niektorých rozšírení a možností.

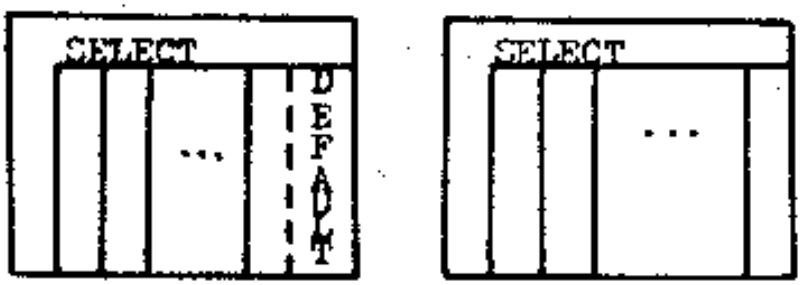
3. Popis a základné charakteristiky

Základnými stavebnými prvkami vývojových blokov sú opäť blok sekvenčný, selektívny a riadený. Sekvenčný blok zodpovedá procesnému bloku v NS-diagramoch. Blok selektívny a riadený majú vo všeobecnosti rovnakú podobu /obr. 2/. Konkrétne môžu byť nasledovné. V rámci selektívnych blokov môže ísť buď o IF-blok, zobrazený na obr. 3 /s ELSE-časťou alebo bez nej/, alebo o SELECT-blok na obr. 4 /s implicitnou DEFAULT-časťou alebo bez nej/.

obr. 2



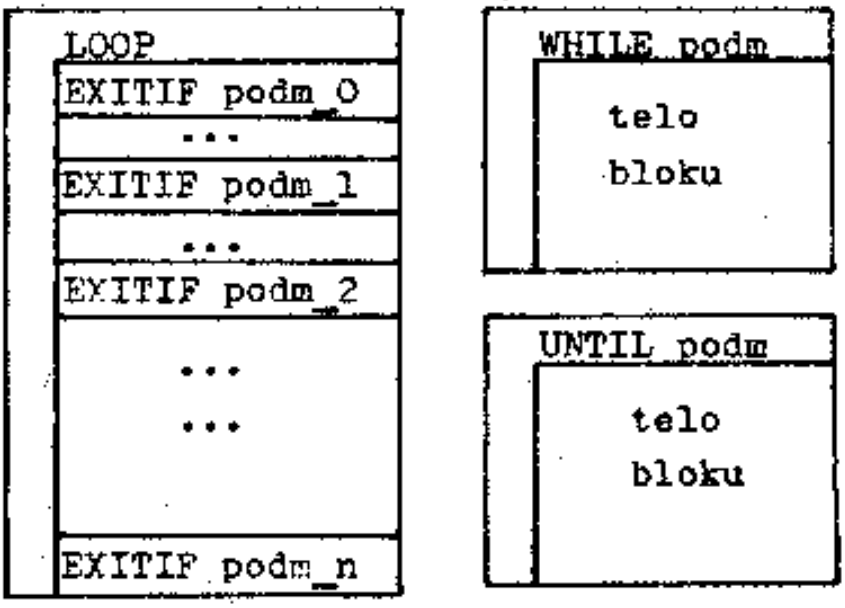
obr. 3



obr. 4

V rámci riadených blokov môžu byť ich symboly ako na obr. 5. Za východiskový riadený blok možno považovať LOOP-blok, z ktorého

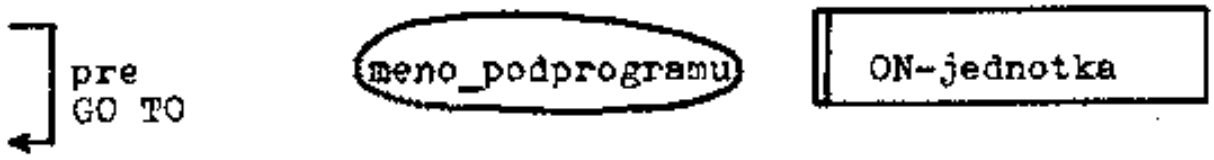
možno odvodiť dva základné iteračné bloky, a to WHILE-blok /existenciou len EXITIF podm_0 v LOOP-bloku/ a UNTIL-blok /existenciou len EXITIF podm_n v LOOP-bloku/. Pre WHILE-blok a UNTIL-blok nie je potrebné znázorňovať podmienky ukončenia cyklu v ich "tele", tie sa zapisujú hneď ku riadiacim slovám.



Treba poznamenať, že kľúčové slová THEN, ELSE, DEFAULT a EXITIF sa môžu vyskytovať len v rámci im prislúchajúceho druhu bloku, nie samostatne. A ďalej to, že takto špecifikované diagramy možno syntakticky definovať [6].

obr. 5

Programovanie úloh hromadného spracovania dát - hlavne s ohľadom na najrozšírenejšie a najpoužívanejšie jazyky, a to COBOL a PL/I - si však vyžaduje zaviesť aj ďalšie symboly pre praktické potreby. Máme na mysli predovšetkým symboly vnútromodulového odovzdania riadenia, volania podriadených modulov a symboly pre isté špecifikované situácie. V prostredí programovacieho jazyka PL/I môžu byť tieto symboly napr. ako na obr. 6.



obr. 6

Samozrejme, takéto a podobné úpravy nie sú nijakým spôsobom predpísané, môže si ich sám zvoliť používateľ, trebárs s ohľadom na programovací jazyk, ktorý používa.

Je vidieť, že osvojenie si týchto schém a štandardov je veľmi prosté. Dajú sa z nich odhadnúť ich základné možnosti. Príklad aplikácie týchto schém je uvedený vo 4. časti. Tu len upozorníme na niektoré výhody, ktoré môže priniesť použitie tejto techniky návrhu modulov.

Predovšetkým je táto technika v priamom vzťahu ku zdrojovému textu programu. Stavebnými prvkami, s ktorými pracuje programátor, sú nie jednotlivé príkazy /ako napríklad pri vývojových diagramoch/, ale celé bloky. Pri porovnaní s vývojovými diagramami odpadá niekedy komplikovaný prevod ich "slučiek" na výkonné príkazy jazyka, čo sa zvykne riešiť napr. neprehľadnými prepínačmi.

Za druhé dôležité rozhranie pre návrh modulu považujeme modulový návrh programu. Technika vývojových blokov mu úplne prirodzene zodpovedá. Parciálne je to naznačené vo 4. časti. Vzhľadom na to, že sa nepoužívajú konektory, priam to programátora núti deliť program do častí.

Podľa nášho názoru je pri aplikácii tejto techniky program hneď pri návrhu vhodne dokumentovaný. Je v ňom vidieť oblasť platnosti zložených príkazov, hraníc cyklov, vetvení ap. No a v neposlednom rade prináša do programovania istú disciplínu, a to vo fáze vývoja logiky programu, ladení, prípadne i testovaní.

Podobne ako u iných techník, i tu si však treba uvedomiť hranice jej platnosti a použiteľnosti. V prípade vývojových blokov tu treba napr. uviesť, že sú účinnejšie pri modernejších /"štrukturovaných"/ jazykoch ako Pascal, PL/I či Ada, než vo FORTRANe , čo je však vec konštrukcie jazyka a nie sa-

motnej techniky. Analogická je situácia pri programovacom jazyku COBOL. Málo vhodné sú vývojové bloky v jazykoch na úrovni assembleru /pravda, pokiaľ nie sú doplnené o štruktúrne makroinštrukcie/.

Vývojové bloky neposkytujú schémy pre návrh paralelizmu. Teda ak by išlo o jeho použitie, treba si zvoliť nejakú notáciu.

V konkrétnom prípade sa môže stať, že ich použitie bude pri riešení niektorých situácií náročnejšie, spravidla však len pri ich prvom výskyte. Ide o cvik a istú zbehlosť.

4. Príklad aplikácie

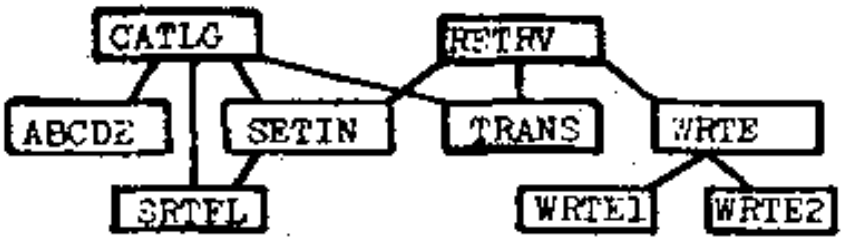
Pre ilustráciu uvádzame príklad použitia techniky vývojových blokov. Pôjde v ňom jednak o poukázanie na niektoré aspekty návrhu programov, ukážku vývojového bloku a niektorých ich ďalších možností. V príklade pôjde len o naznačenia schémy riešenia.

V našej úlohe išlo v zásade o dva problémy.

1. Pre kmeňový súbor, v ktorého dátach sú isté polia smerodatné, sa má vytvoriť jemu prislúchajúci katalóg, v ktorom sú uchované údaje smerodatných polí, každý z údajov podľa toho, z ktorého poľa pochádza, je kódovaný jednoznačným prefixom.

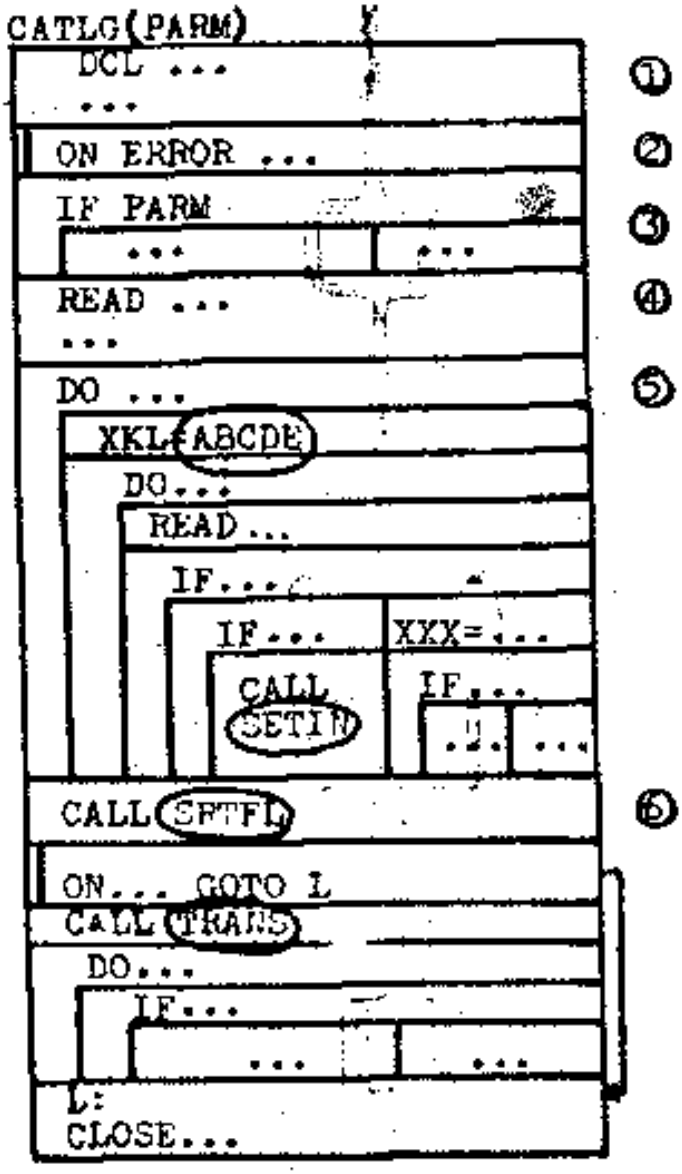
2. Má sa zabezpečiť možnosť vyhľadávania v takomto katalógu.

Úloha bola riešená dvoma hlavnými modulmi, ktoré využívali dva spoločné podriadené moduly. Schéma medzimodulových vzťahov riešenia je na obr. 7. Keďže na tomto mieste si nevšímame problém modulového návrhu programov, na obr. 7 nie sú vyznačené objekty, ktorými sú jednotlivé moduly viazané.



obr. 7

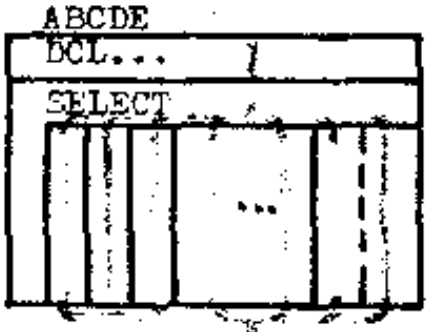
Vzhľadom na ilustratívnosť príkladu tu nebudeme špecifikovať funkcie jednotlivých modulov. Uvedieme len schému programového riešenia modulu CATLG, ktorá je na obr. 8. Úloha bola riešená v PL/I, a tak sa v schéme vyskytujú torzá príkazov tohoto jazyka. Z objektov deklarovaných v module sme v schéme modulu uviedli len parameter hlavnej procedúry PARM, návestie L a dve programové premenné XKL a XXX. Vpravo od diagramu sme očíslovali niektoré jeho bloky.



obr. 8

Na rozdiel od vývojového diagramu, ktorý sa kreslí spravidla od začiatku, diagram tohoto modulu vznikol počínajúc blokom ⑤ a blokmi v ňom vloženými. Iteratívne s tým vznikal blok ① s deklaráciami a potom bloky ②, ③, ④ a ⑥. Druhou fázou bol zápis blokov zostávajúcich.

Schéma podriadeného modulu ABCDE je v najhrubšom priblížení zobrazená na obr. 9. Štruktúra SELECT, použitá v schéme, je pre jazyk PL/I /F/ opísaná napr. v [7].



obr. 9

Chceme upozorniť na jeden aspekt použitia vývojových blokov. A to na možnosti, ktoré poskytujú pre ladenie a testovanie programov. Priamo v schéme modulu možno totiž vyznačiť všetky logické vetvy programu a na ich základe hneď pri návrhu modulu plánovať jeho testy. Tak napr. pre moduly CATLG a ABCDE sú všetky ich logické vetvy znázornené v ich diagramoch tieňovane.

5. Záver

Pokúsili sme sa v základe naznačiť možnosti, ktoré vývojové bloky poskytujú pri návrhu programov. Veľa z toho, čo tu bolo uvedené, si používateľ môže prispôsobiť a zmeniť podľa vlastných potrieb. Sme toho názoru, že okrem uvedených možností sú vývojové bloky vhodným nástrojom aj pre výuku programovania, pretože vedú k návrhu programov od ich najvyššej úrovne.

Literatúra

[1] Čimbura V., Tvrdík J.: Problémy návrhu modulárnich programů. In: Programování '80, DT Ostrava 1980, str. 1 - 37.
[2] Chapin N.: New format for flowcharts. Software - practice and experience, vol. 4/1974/, pp. 341 - 357.
[3] Witty R. W.: Dimensional flowcharting. Software - practice and experience, vol. 7/1977/, pp. 553 - 584.
[4] Vondráček B. a kol.: Technologie strukturovaného programování. In: Programování '80, DT Ostrava 1980, str. 38 - 78.
[5] Nassi I., Shneiderman B.: Flowchart techniques for structured programming. SIGPLAN notices, vol. 1973, No. 8, pp. 12-26.
[6] Grouse P.: "Flowblocks" - a technique for structured programming. SIGPLAN notices, vol. 1978, No. 2, pp. 46 - 56.
[7] Kirschner A.: Implementácia doplnkových riadiacich štruktúr v jazyku PL/I pomocou jeho predprocesora. In: Programování '80, DT Ostrava 1980, str. 118 - 127.