P. Fischer
VVS - Bratislava

BPS - INTEGROVANÝ PROGRAMOVACÍ SYSTÉM

1. ÚVOD

Bratislavský programovací systém je komplexný programový systém, ktorý integruje a automatizuje niektoré činnosti spojené s vytváraním softwarových produktov. BPS je koncipovaný ako integrovaný, no pritom dostatočne modulárny a otvorený systém. Svojou štruktúrou je BPS veľmi podobný interpretačným systémom. Najtypickejšou vlastnosťou spoločnou jednak BPS, či interpretačným systémom, je okrem iných najmä prítomnosť bázy dát, prístupnej prostredníctvom administrátora bázy dát.

Báza dát, resp. systém riadenia bázy dát predstavuje jadro systému BPS. Služby tohoto jadra systému využívajú viaceré skupiny automatizačných prostriedkov (jazykové, dokumentačné, optimalizačné, linkovacie, testovacie, ladiace, udržiavacie a modifikačné, vyhodnocovacie, atď.), ktoré sú sprístupnené užívateľom pomocou triviálneho užívateľského jazyka.

BPS má slúžiť jednak počas doby implementácie programových systémov a potom i pri ich údržbe a modifikáciách. Jedná sa o programový systém pokrývajúci širokú etapu tvorby programov, v ktorej skoro v každom kroku vďaka formalizácii jednotlivých činností (formálne jazyky jednotlivých prostriedkov systému) existuje pomerne silná spätná väzba (kompilátory týchto jazykov), ktorá reguluje správnu čin-

nosť procesu a poskytuje množstvo informácií (báze dát) pre jeho ďalšie vylepšovanie. Integrujúcim článkom systému BFS je metóda modulárneho programovania. Z tejto metódy vychádzajú všetky programovacie prostriedky systému.

Modulárne programovanie

Prakticky všetky nové progresívne metódy tvorby programov sa zakladajú na systematickej aplikácii známeho a v praxi osvedčeného princípu "rozdeľuj a panuj". V programovaní ho aplikujeme tak, že ak stojíme pred riešením zložitého problému postupujeme tak, že sa ho snažíme "rozbiť" na niekoľko jednoduchších problémov, ktoré sú síce nejakým spôsobom navzájom závislé, ale ktorých zložitosť riešenia je menšia. V rozvíjaní väčších problémov na menšie problémy pokračujeme dovtedy, až jednotlivé podproblémy sú danými programovacími prostriedkami jednoducho a efektívne realizovateľné. Ak program P je riešením daného problému, potom riešením jeho jednotlivých podproblémov sú časti programu, ktoré nazývame <u>modulmi</u>. Vzájomným závislostiam jednotlivých podproblémov odpovedajú <u>spojenia</u> medzi modulmi. Pod spojením dvoch modulov rozumieme predpoklady, resp. požiadavky, ktoré jeden modul kladie na modul druhý.

Formálny popis hore uvedenej metódy je nasledovný:

Nech P je problém, ktorý máme riešiť pomocou počítača M a jazyka L, potom pod procesom tvorby programu P, ktorý daný problém P rieši rozumieme proces vytvárania postupnosti programov

$$P_0, P_1, \ldots\ldots, P_n, \text{ takých, že}$$

1. P_0 je program pozostávajúci iba z jediného modulu m_0, ktorý daný problém rieši. Modul m_0 však na riešenie problému môže použiť i služby iných modulov, tým že si s nimi vytvorí spojenia. Tieto moduly nemusia existovať (kompilátor jazyka L a počítač M ich nemusia

poskytovať) o tvorcu modulu m_0 o nich predpokladá iba to, že budú realizovať požiadavky v spojeniach naň kladených.

2. Program P_i získame z programu P_{i-1}, tak, že niektorý z modulov, s ktorými program P_{i-1} má spojenie, a ktoré nie sú ešte vytvorené - deklarujeme (realizujeme). Vždy sa snažíme vybrať taký modul, o ktorého spojeniach predpokladáme, že sú úplné, resp. je malá pravdepodobnosť, že sa v budúcnosti zmenia.

Prechod od programu P_{i-1} k programu P_i nazývame i-tým krokom procesu tvorby programu P.

Nech m_i je modul, ktorý sme v i-tom kroku deklarovali, potom

$$P_i = P_{i-1} \cup \{m_i\} \quad (*)$$

S. Proces končí programom $P = P_n$, v ktorom sú už všetky moduly deklarované.

Z (*) plynie, že ak je program P_{i-1} korektný, t.j. robí to čo sa od neho očakáva a ak deklarácia modulu m_i je korektná, t.j. realizuje spojenia programu P_{i-1} s ním potom aj program P_i je korektný. Táto úvaha, respektíve princíp stojí v pozadí úspechu všetkých progresívnych metód tvorby programov.

Intelektuálne najnáročnejšou činnosťou celého procesu tvorby programov je tvorba spojení medzi modulmi a kontrola či daný modul správne realizuje s ním vytvorené spojenia. Spojenia medzi modulmi definujú na množine modulov čiastočné usporiadanie modulov, ktoré zároveň určuje i hierarchickú štruktúru programu.

Hovoríme, že postupnosť modulov $m_0, m_1, \ldots, m_n$ je <u>hierarchicky usporiadaná</u> ak pre ľubovoľné dva moduly m_i a m_j v danej postupnosti platí, že ak m_i má spojenie s m_j potom m_i je v danej postupnosti pred modulom m_j t.j. $i < j$.

Hovoríme, že program P bol vytvorený metódou zhora-nadol ak jeho jednotlivé moduly boli vytvárané v hierarchickom poradí. V prípade, že jednotlivé moduly programu boli vytvorené v opačnom poradí, potom hovoríme o metóde tvorby programov zdola-nahor.

Dôležitosť spojení medzi modulmi programu okrem iného dokumentuje i tá skutočnosť, že sú to práve oni čo určujú štruktúru celého programu. Pod štruktúru nejakého objektu sa chápe jeho čiastočný popis; v prípade programu sú to moduly programu a spojenia medzi nimi. Ide tu o čiastočný popis programu, pretože abstrahuje od detailov jednotlivých modulov programu. Pre vytvorenie lepšej predstavy o štruktúre programu definujeme si graf štruktúry programu.

Pod grafom štruktúry programu rozumieme trojicu
$GŠP = (N,S,n_0)$, kde

N - je konečná množina modulov programu, ktoré tvoria vrcholy grafu.

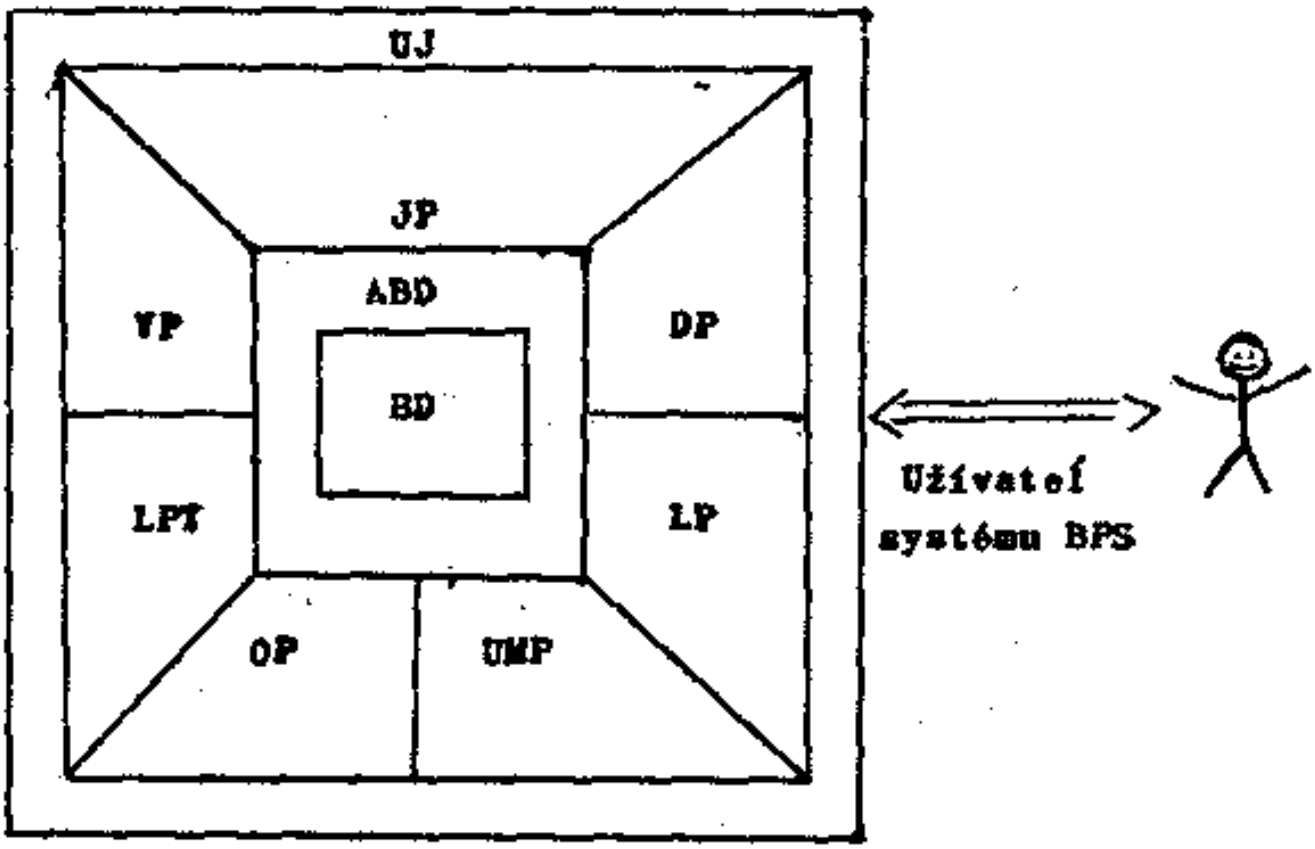
S - je konečná množina spojení medzi modulmi a reprezentuje množinu orientovaných hrán v grafe. Z vrcholu m vedie orientovaná hrana do vrcholu n práve vtedy ak modul m má spojenie s modulom n.

n_0 - je počiatočný vrchol grafu a odpovedá modulu m_0.

Je zrejmé, že v každom kroku procesu tvorby programov možeme vytvoriť takýto graf a to automaticky. To je veľmi dôležité pretože graf štruktúry programu je v pozadí prakticky všetkých programovacích nástrojov systému BPS.

2. Štruktúra BPS.

Jadro systému predstavuje báza dát, obsahujúca programy, vzťahy medzi programami a rozne informácie o programoch a procese ich tvorby. Štruktúra systému BPS charakterizuje obr. 1.



obr. 1

BD - báza dát
ABD - administrátor bázy dát
JP - jazykové prostriedky
DP - dokumentačné prostriedky
LP - linkovacie prostriedky
VP - vyhodnocovacie prostriedky
OP - optimalizačné prostriedky
UMP - prostriedky pre údržbu a modifikáciu programov
LXP - ladiace a testovacie prostriedky
UJ - užívateľský jazyk systému BPS

Okolo bázy dát sú sústredené rôzne programovacie prostriedky, ktoré majú cez administrátora bázy dát prístup k informáciam v báze dát. Styk užívateľa s jednotlivými prostriedkami sprostredkováva užívateľský jazyk systému BPS. Systém BPS umožňuje užívateľovi pracovať s ním od začiatku procesu tvorby programov, počas jeho údržby i v priebehu používania programu.

2.1 Báza dát

Báza dát systému BPS pozostáva z knižníc, ktoré si užívateľ systému vytvára sám pomocou príkazov pre manipuláciu s knižnicami. Každá knižnica pozostáva z viacerých menších častí, ktoré v ďalšom budeme nazývať deckami. Obsahom decku môže byť modul programu a to v rozných tvaroch - textový, binárny alebo absolútny, rôzne informácie o programoch, vzťahoch medzi jednotlivými programami a pod.

Užívateľ má možnosť užívať obsah knižnice, meniť a prípadne zrušiť celú knižnicu a podobne je s deckami v rámci knižnice. Najmenšou manipulačnou jednotkou je decK knižnice. Niektoré decky v rámci jednotlivých knižníc si budú môcť vytvárať jednotlivé zložky (prostriedky) systému BPS, ku ktorým užívateľ nebude mať priamy prístup.

S knižnicami sa bude dať pracovať spôsobom on-line cez komunikačné terminály alebo spôsobom off-line cez štandardné vstupno-výstupné médiá.

Podľa obsahu knižnice a frekvencie jej používania jej fyzické uloženie môže byť na médiu s priamym prístupom (disk) alebo na médiu so sekvenčným prístupom (mg. páska) . Prístup do bázy dát bude realizovaný cez administrátora bázy dát pomocou prístupových mechanizmov bázy dát. Súčasťou systému budú i prostriedky pre ochranu knižníc pred ich znehodnotením.

2.2. Jazykové prostriedky

Jazykové prostriedky tvoria kompilátory tých jazykov, ktoré systém BPS dovoluje jeho užívateľom používať. Ide predovšetkým o kompilátory jazykov jednotlivých programovacích prostriedkov systému BPS, ďalej kompilátor univerzálneho programovacieho jazyka vyššej úrovne, ktorý umožňuje modulárne programovanie, kompilátor jazyka TWS, t.j. jazyka pre písanie kompilátorov a prípadne kompilátory ďalších jazykov akonáhle sa to ukáže potrebné.

Pri návrhu univerzálneho programovacieho jazyka sa vychádzalo z nového jazyka prof. Wirtha - 'MODULA'(Wi). Ide tu o programovací jazyk, ktorý umožňuje modulárne programovanie, tak ako to popisuje kapitola o modulárnom programovaní. Základnou kompilačnou jednotkou v tomto jazyku je modul. Modul je vlastne najväčšou abstraktnou štruktúrou, ktorá sa v tomto jazyku dá vytvoriť. Modul pozostáva z ľubovoľného počtu objektov, ktoré v module deklaruje jeho tvorca. Objektom modulu môže byť konštanta, typ, premenná alebo operácie nad premennými modulu. Operácie modulu majú tú istú štruktúru ako procedúry. Môžu mať vlastné objekty, ktoré sú im lokálne. V rámci procedúr však nemožu byť deklarované iné procedúry - hniezdenie procedúr nie je možné. Ide teda o fortranovskú štruktúru procedúr. Objekty modulu môžu byť dvojakého druhu:

- globálne, ktoré až na niektoré obmedzenia môžu používať iné moduly ak o to explicitne požiadajú (majú s daným modulom spojenie),
- lokálne, ktorých použitie nepresahuje rámec modulu.

Pri deklarácii objektov daného modulu môžeme použiť i (globálne) objekty iných modulov (ktoré ešte nemusia byť úplne deklarované - iba ich čiastočné špecifikácie), ktoré sú uvedené v zozname použitých modulov.

Ak modul m používa niektoré z globálnych objektov modulu n hovoríme, že modul m je nadriadený modulu n a modul n

zase podriadený modulu m. Podriadený modul môže použiť niektoré (premenné alebo konštanty) objekty nadriadeného modulu iba vtedy, ak mu ich nadriadený modul pošle ako parameter v niektorých z operácií podriadeného modulu.

Jazyk TWS sa do jazykových prostriedkov zaradil preto, že systém BPS je určený predovšetkým pre tvorbu veľkých programových systémov a každý "slušný" programový systém by mal obsahovať jazyk, ktorý ho jednak popisuje, a ktorým sa daný systém ovláda.

2.3 Dokumentačné prostriedky

Z praktických skúseností vieme, že v rôznych štádiách tvorby programu (návrh, implementácia, ladenie, testovanie, údržba, modifikácia a pod.) sa vyžaduje iný druh dokumentácie. Rozne druhy dokumentácie slúžia rozným skupinám ludí pre rozne účely.

Cielom dokumentačných prostriedkov je poskytovanie dokumentácie programu, ktorá bude slúžiť tým čo sa na jeho tvorbe a údržbe bezprostredne podieľajú. Ide tedz o programovú dokumentáciu, ktorá bude slúžiť počas tvorby a údržby programu. V ďalšom sú uvedené niektoré podmienky, o ktorých sa domnievame, že by každá dokumentácia mala spĺňať. Dobrá dokumentácia by mala byť:

a) prístupná - kus popísaného papiera na stole programátora nemože byť považovaný za dokumentáciu

b) jednotná - musí splniť nejaký štandard čo do obsahu i formy

c) úplná a pravdivá - užívateľa dokumentácie nezaujíma, ktorý algoritmus sa včera vybral pre riešenie daného problému, ale ten, ktorý sa použil a ako sa implementoval

d) prehľadná - hierarchicky usporiadaná, umožňujúca rýchlu orientáciu v programe

e) aktívna - pri hľadaní určitých informácií o nejakom objekte v programe má upozorniť i na doplňujúce informácie.

Je zrejmé, že zoznam uvedených podmienok, ktoré dokumentácia má spĺniť nie je úplný. Žiaľ, už tieto kritéria vylučujú človeka z procesu na jej tvorbe a to bolo účasi i cieľom - poukázať na to, že dokumentácia s horeuvedenými vlastnosťami musí byť generovaná automatickými prostriedkami, čo možno najmenej závislými od človeka.

Pri analýze programov sa zistilo, že pre pochopenie programu je najdôležitejšia jeho štruktúra a tá je zasa daná tokom riadenia a tokom dát v programe.

Tok riadenia v programe určuje postupnosť príkazov programu v akej sa realizujú počas výpočtu programu. Tok dát v programe určuje zmeny hodnot premenných programu počas jeho výpočtu. Tok dát a tok riadenia v programe jednoznačne určujú dynamické vlastnosti programu.

Cieľom dokumentačných prostriedkov je zistenie potrebných informácií o toku riadenia a toku dát v programe a vo vhodnom tvare ich prezentovať užívateľovi.

Vstupom pre dokumentačné prostriedky bude program v povodnom tvare a ich základ budú tvoriť - syntaktický, analyzátor toku dát a analyzátor toku riadenia v programe.

2.4 Linkovacie prostriedky

Spoľahlivosť softwaru je bezosporu najdôležitejším kritériom jeho kvality. Jedným z možných spôsobov ako spoľahlivosť softwaru dosiahnuť je vyvíjať ho takým spôsobom, aby pravdepodobnosť chyby v ňom bola čo najmenšia. Dosiahnuť sa to dá dvoma spôsobmi:

a) znížením pravdepodobnosti vzniku chýb
b) zvýšením pravdepodobnosti detekcie chýb.

Nutným nie je však postačujúcim predpokladom vytvorenia spoľahlivého (spoľahlivejšieho) softwaru je profesio-

nálne zvládnutie tak riešeného problému ako i programovacích nástrojov, pretože nedostatočná znalosť problému a nesprávne ovládanie programovacích nástrojov sú najčastejšími zdrojmi chýb. Túto skutočnosť si na druhej strane musíme uvedomiť i tvorcovia programovacích nástrojov, aby programovacie nástroje neboli príliš rozsiahle (Algol 68, PL/I/ a intelektuálne náročné (axiomatické špecifikácie , formálny dôkaz korektnosti programu), a aby správnosť a oprávnenosť ich použitia v jednotlivých fázach procesu tvorby softwaru sa dala automaticky kontrolovať (syntaktický a sémantický analyzátor v kompilátore). A práve linkovacie prostriedky majú plniť úlohu takéhoto "kontrolóra" pre jazykové prostriedky.

Prečo potrebujeme "kontrolóra" pre programovací jazyk, akým je Pascal, keď túto úlohu vykonáva jeho kompilátor?

Praktickým dôsledkom aplikácie nových progresívnych metód tvorby programov, ako je štruktúrované programovanie, modulárne programovanie a pod. je rozdelenie programu na množstvo menších podprogramov - modulov. Riešenie zložitejšieho problému sa tak redukuje na riešenie väčšieho počtu menších problémov. V rozdeľovaní väčších problémov na menšie pokračujeme dovtedy, až jednotlivé podproblémy sú danými prostriedkami jednoducho a efektívne realizovateľné. Jednotlivé podproblémy sú v programe riešené modulmi. Moduly programu sa v procese tvorby programu používajú rovnakým spôsobom ako každý iný príkaz programovacieho jazyka. Modul je teda akýsi problémovo orientovaný príkaz. Rozdiel medzi príkazom jazyka a modulom je ten, že zatiaľ čo syntax a semantiku príkazu jazyka definuje jeho tvorca a realizuje kompilátor jazyka, syntax a sémantiku modulu si definujeme a realizujeme sami v priebehu riešenia problému.

Na tvorbe väčších programov sa podieľa viac ľudí a je bežné, že ten kto daný modul vytvára, nepozná všetkých jeho užívateľov a opačne, ten kto daný modul používa sa nezaujíma o jeho vnútornú štruktúru. Je len samozrejmé, že programátor nebude poznať jednotlivé moduly tak dobre ako základné príkazy jazyka. Chyby spôsobené nesprávnym použitím modulu (nekonzistentnosť skutočných a formálnych parametrov), nesprávnou komunikáciou medzi modulmi (nesprávna postupnosť volaní) a pod. sú veľmi nepríjemné a v programovaní časté. Žiaľ, pri separátnej kompilácii modulov a súčasnej štruktúre kompilátorov (bez bázy dát) je detekcia takýchto chýb kompilátorom nemožná.

A práve z horeuvedených dôvodov sme do systému BPS zaradili linkovacie prostriedky, ktorých cieľom je detekcia a lokalizácia chýb súvisiacich s modulmi (procedúrami). Ide tedy o akýsi metakompilátor pre moduly a pre "interface" medzi modulmi. Nie je vylúčené, že sa v budúcnosti pokúsime o vytvorenie jazyka pre komunikáciu medzi modulmi a synchronizáciu modulov. Vznik vhodného jazyka pre tieto účely by značne skvalitnil celý proces tvorby programu ako i výsledné programy. Niektoré teoretické výsledky tohoto druhu sa už objavujú.

Okrem horeuvedených činností linkovacie prostriedky budú prevádzať i tú činnosť, ktorú doteraz prevádzajú "Loader" (CDC) a "LINKAGE EDITOR" (IBM) odtiaľ je ich názov.

Základ linkovacích prostriedkov bude tvoriť syntaktický analyzátor, analyzátor toku riadenia a analyzátor toku dát medzi modulmi (procedúrami).

2.5. Prostriedky pre modifikáciu a údržbu programu

Proces tvorby programov je proces iteratívny, v ktorom neustále dochádza k modifikácii programu a to buď za

účelom vylepšenia jeho funkčných prípadne prevádzkových vlastností (efektívnosť).

Z množstva požadovaných modifikácií sa prevážná väčšina vyžaduje na programe v povodnom tvare. Pri modifikácii programu ide v podstate o manipuláciu s textom (programu):

- pridanie novej časti textu na presne špecifikované miesto,
- zámenu špecifikovanej časti textu za inú časť textu,
- vyradenie špecifikovanej časti textu a pod.

Cieľom prostriedkov pre údržbu programov bude umožňovat "spúšťanie" programov a to buď jednorázové na požiadanie, alebo pravidelné viacnásobné, ktoré sa prevádza pravidelne za určitých okolností ako napr.:

- pri každej modifikácii programu sa dajú do činnosti testovacie, dokumentačné, linkovacie a prípadne iné prostriedky systému BPS,
- pravidelné "dumpy" programov a knižníc programov za účelom ochrany programov pred ich prípadným poškodením,
- pravidelné denné, týždenné alebo mesačné spúšťanie programov a pod.

Ďalej ide o údržbu niekoľkých verzií tých istých programov. S prostriedkami pre modifikáciu a údržbe programov bude pracovať nielen užívateľ programu, ale predovšetkým jeho tvorca a to buď spôsobom on-line alebo off-line.

Na týchto prostčiedkoch sa v skupine už začalo pracovať a dospeli sme aj k realizácii niektorých častí ako je BTS a Edit. Bližšie informácie o týchto prostriedkoch čitateľ nájde v užívateľských príručkách BTS-u a Editu-u.

3.6. Optimalizačné prostriedky

Efektívnosti programov a hľadaniu optimálnych programov sa v poslednej dobe venuje veľká pozornosť. Ukazuje sa, že to má zhruba nasledujúce príčiny:

- snaha nájsť čo najlepší algoritmus pre riešenie daného problému je konkrétnym prejavom prirodzenej potreby robiť to, čo sa má robiť čo najlepšie;
- aj malé zefektívnenie tých algoritmov a programov, ktoré sa realizujú (bežia) často, môže priniesť výrazný časový a teda aj ekonomický efekt. Zvlášť je to dôležité pri základnom softwaro a pri softwaro pracujúcom s veľkým množstvom dát;
- nájdenie efektívnejšieho programu často umožňuje riešiť predtým prakticky neriešiteľné úlohy a môže priniesť viac ako niekoľkonásobné zrýchlenie operačnej rýchlosti počítačov.

Ukazuje sa, že z celého radu možných kritérií efektívnosti programov má praktický a tiež teoretický význam časová a pamäťová zložitosť programov, ktorá udáva ako rastie čas potrebný na výpočet programu resp. kapacita použitej pamäte.

Efektívnosť programu sa dá v podstate dosiahnuť dvoma spôsobmi:

1. výberom efektívneho algoritmu,
2. efektívnou implementáciou algoritmu.

Výberom efektívnych algoritmov sa venuje v poslednej dobe veľká pozornosť, hoci zatiaľ sa nepodarilo nájsť univerzálnu metódu tvorby efektívnych algoritmov (a algoritmov vôbec). Potešiteľná je však tá skutočnosť, že v poslednej dobe sa pomaly vynárajú princípy konštrukcie efektívnych algoritmov a programov, ktoré majú všeobecnejšiu platnosť ako napr. voľba štruktúr dát, "divide et impera", balancovanie, rekurzia, dynamické programovanie, programovanie s návratom a pod. Je len samozrejmé, že nie sú to úplné a jediné techniky, dá sa očakávať, že v budúcnosti sa objavia ďalšie.

Cieľom optimalizačných prostriedkov nie je ani výber vhodného algoritmu ani jeho efektívna implementácia, ale

cieľ ovela skromnejší - zefektívnenie (vylepšenie) daného konkrétneho programu ekvivalentnými úpravami textu programu. V súčasnej dobe je známe množstvo ekvivalentných úprav textu programu, ktorý zefektívňuje daný program pre ktoré sú už známe i algoritmy, ktoré ich realizujú. Okrem známych štandardných optimalizácií programu (pod optimalizáciou sa rozumie vylepšovanie programu), optimalizačné prostriedky vykonávajú optimalizáciu, ktorú si vynucuje samotná metodológia programovania. Ako už bolo spomenuté, dosiahnutá aplikácia nových metód programovania značne zvyšuje počet modulov a tým i počet komunikácií medzi modulmi (každý modul môže potenciálne komunikovať s každým iným modulom programu ba dokonca i sám so sebou v prípade rekurzie, a to viackrát).

Z hľadiska efektívnosti programu je pre nás dôležitá tá skutočnosť, že každá komunikácia vyžaduje:

a) pamäť pre uloženie mechanizmu (časť programu), ktorý komunikáciu realizuje - v prípade volania procedúry sú to inštrukcie pre uloženie obsahov registrov, adresy návratu, prepojenie skutočných formálnych parametrov a pod.

b) čas potrebný na realizáciu - výpočet uvedených inštrukcií.

Neberúc do úvahy tieto okolnosti, by mohlo mať za následok, že z ekonomického hľadiska by čas výpočtu programu nebol vobec vyhovujúci.

A práve okrem štandardných druhov optimalizačných informácií programu počítame i s transformáciami, ktoré horeuvedené problémy značne redukujú. Ide predovšetkým o:

- nahradzovanie volania procedúry telom (otvorené procedury),
- eliminácie zbytočných výpočtov,
- "garbage collection" v čase kompilácie,
- podmienené generovanie kódu,
- odstránenie rekurzie, zámenenie cyklov a pod.

Základom optimalizačných prostriedkov je opäť analyzátor programu, predovšetkým globálna analýza toku dát a toku riadenia v programe.

2.7. Ladiace a testovacie prostriedky

Mnohé štatistické údaje z praxe ako i niekoľko teoretických štúdií poukazujú na to, že testovanie softwaru reprezentuje 30 až 50 % hodnoty celého softwaru. Navyše nespoľahlivosť programu, ktorá je dôsledkom nesystematického prístupu pri testovaní a ladení programu spôsobuje ďalšie a v mnohých prípadoch katastrofálne finančné straty. So vzrastajúcimi aplikáciami počítačov prakticky v každej ľudskej sfére, s veľkosťou a dôležitosťou riešených problémov rastie i záujem o hľadanie nových spôsobov ako zvýšiť kvalitu testovania a súčasne znížiť jeho náklady. Význam pojmov ladenie a testovanie programov nie je vždy jasný a častokrát si ich vysvetľuje každý po svojom.

Testovanie - je proces, v ktorom sa snažíme odhaliť chybu tým spôsobom, že "spúšťame" program pre rôzne (ale pritom dovolené sú) vzorky vstupných dát a porovnáme dosiahnuté výsledky s očakávanými. Prípadné nezhody pri porovnávaní dosiahnutých a očakávaných výsledkov znamenajú detekciu chyby v programe.

Ladenie (debugging) je proces lokalizácie a eliminácie chýb testovaním objavených.

Testovaním a ladením programu vlastne overujeme spoľahlivosť programu a z toho dôvodu budeme v ďalšom proces testovania a ladenia programu nazývať tiež procesom overovania programu alebo skrátene overovaním programu.

Niektorí "experti" vo svojich prejavoch a článkoch hlásajú, že testovanie a ladenie je záležitosť čisto mechanická, ktorú navyše môže pravdaže iná skupina ľudí než tá, čo daný program vytvorila a od ktorej sa nevyžaduje veľká intelektuálna náročnosť. Zdá sa, že pravý opak je pravdou, a že testovanie a ladenie programu zo daného stavu

programovaných prostriedkov je proces intelektuálne náročný, vyžadujúci profesionálnu znalosť štruktúry programu, jeho špecifikácií a jazyka, v ktorom je program napísaný, že jedinou mechanickou činnosťou je opakované "spúšťanie" programu pre rozne vzorky vstupných dát, ich porovnávanie a sledovanie toku riadenia a toku dát v programe. Problémom číslo jedna pri testovaní programu, je určenie takej množiny vzoriek vstupných dát, ktorá by úplne overila spoľahlivosť celého programu a minimálna v tom zmysle, že každá iná taká množina, ktorá by bola úplná, bola by väčšia (vyššej kardinality). O tomto probléme je známe, že je algoritmicky neriešiteľný, čo je na druhej strane známkou problému intelektuálne náročného. (Pri programovaní vôbec sa stále stretávame s problémami algoritmicky neriešiteľnými). Nutným nie však postačujúcim predpokladom vytvorenia takejto množiny je dokladná znalosť štruktúry programu. Ako je známe, tá je výsledkom intelektuálnej činnosti tvorcov programu (ktorej navyše presná definícia chýba)a nedá sa jednoduchou analýzou programu získať. Problémy s presnou definíciou štruktúry programu len poukazujú na to, že súčasťou štruktúry programu sú i jej tvorcovia (v umení to tak býva a programovanie je občasti umením).

Ďalším problémom pri testovaní je určenie očakávanej vzorky výstupných dát pre každú vzorku vstupných dát. Podobne ako predošlý aj tento problém je algoritmicky neriešiteľný a vyžaduje úplnú znalosť špecifikácii programu. Ako je známe, problém jednoznačného zápisu a kontroly na úplnosť a konzistentnosť špecifikácii zostáva zatiaľ neriešiteľný. Vyriešením problému špecifikácií sa do značnej miery vyriešia i niektoré problémy testovania.

O procese lokalizácií a eliminácií chýb nemožno taktiež hovoriť ako o mechanickom procese. Zistiť, ktorou vetvou výpočet programu prešiel (tok riadenia) je síce

mechanická záležitosť, ale to k lokalizácii a odstraneniu chyby nestačí, k tomu je potrebná analýza dát takto získaných a to zatiaľ automatizovať žiaľ nevieme.

Z horeuvedenej analýzy čitateľ môže mať dojem, že pri testovaní a ladení sa toho veľmi nedá vylepšiť. Nie je to tak beznádejné, pretože tá činnosť, ktorá je mechanická, je časovo veľmi náročná a jej "komputerizácia" môže značne znížiť náklady na testovanie a ladenie programu. O ladiacich a testovacích prostriedkoch sa dá predpokladať, že umožnia overovať program v dvoch etapách.

V prvej etape pojde iba o čiastočné overovanie systému (časti systému), ktoré sa bude vykonávať priebežne s vývojom programu. Za předpokladu "top-down" prístupu pri tvorbe programu vyžaduje sa v tejto etape simulácia abstraktných (ešte neimplementovaných) častí programu. Pojde tu samozrejme iba o jednoduchú simuláciu (prázdne telá nedeklarovanej procedúry, deterministický alebo nedeterministický automat simulujúci činnosť daného objektu, generátor náhodných dát z daného rozsahu a pod.). Tento spôsob sa bude postupne zdokonaľovať s vývojom techník pre špecifikáciu softwaru.

V druhej etape pojde o dôsledné testovanie už kompletného programu. Pojde tu v podstate o systematické testovanie stále väčších a väčších častí programu, pričom sa zachováva zásada, že vždy sa testuje tá časť programu, ktorá je na iných častiach nezávislá, alebo je závislá na tých častiach programu, ktoré už boli testované.

Súčasťou overovacích prostriedkov budú prostriedky pre:

- simuláciu nerealizovaných objektov programu na základe známych špecifikácií,
- sémantickú dekompozíciu programu na jednotlivé časti vhodných pre testovanie,
- identifikáciu a kódovanie jednotlivých ciest v programe,

- generovanie vzoriek vstupných dát pre jednotlivé cesty v programe,
- spúšťanie programu pre dané vzorky vstupných dát a to buď na požiadanie alebo automaticky pri každej modifikácii programu,
- prostriedky pre sledovanie toku riadenia a toku dát v programe (symbolický dump, stepovanie) a prostriedky pre analýzu toku dát a toku riadenia v programe.

Interaktívny spôsob práce s overovacími prostriedkami môže značne zvýšiť celkový efekt procesu testovania a ladenia softwaru.

2.8. Vyhodnocovacie prostriedky

Proces tvorby programov je proces živelný, v ktorom existuje iba veľmi slabá spätná väzba. Charakteristickou vlastnosťou takéhoto procesu je to, že sa zle riadi, kontroluje a poskytuje málo informácií pre jeho zlepšenie. Živelnosť procesu tvorby programov sa potom spätne prenáša i na tvorcov programovacích prostriedkov, pretože nemajú spoľahlivé informácie o tom, čo všetko sa v skutočnom programovaní deje. Tvorcom programovacích prostriedkov potom nič iné nezostáva než spoliehať sa na vlastnú intuíciu a improvizovať. A preto sa niekoho nemôže diviť, že mnohé nové a "nádejné" produkty prinášajú jeho tvorcom sklamanie a do výroby softwaru ďalšie zmätky. Len tak si totiž môžeme vysvetliť dlhotrvajúcu krízu softwaru, o ktorej vieme, že nevznikla z nadvýroby, ale zo zaostalosti procesu výroby softwaru.

Spoľahlivé informácie o užitočnosti existujúcich programovacích nástrojov, o chybách v procese vzniklých, o náročnosti detekcie, lokalizácie a odstraňovaní chýb, ďalej informácie o tvorcoch softwaru a využití jednotlivých prostriedkov by mali slúžiť ich "manažerom". Napríklad informácie o chybách (jednoduchá štatistika chýb), ktoré vzni-

kajú pri používaní daného prostriedku, informácie o tom kde, kedy, ako a kým daný prostriedok bol použitý sú nesmierne dôležité pre tvorcu daného prostriedku a tiež pre "managera" programátorov. Tvorca na základe týchto informácií zistí, dobré a slabé miesta prostriedku a môže previesť nápravu, "manager" zase môže zabezpečiť školenie alebo stiahnuť prémie.

Potreba spoľahlivého zdroja informácie je o to naliehavejšia, že softwarové inžinierstvo okrem toho, že prežíva krízu je mladou vednou (inžinierskou) disciplínou. Z histórie vieme, že každá "rozumná" vedná disciplína sa vo svojich začiatkoch najviac opiera o empiriu (existujú aj výnimky, ale tých je málo). Integrácia procesu tvorby softwaru, formalizmus a štruktúra systému BPS sú zárukou spoľahlivého zdroja požadovaných informácií.

Cieľom vyhodnocovacích prostriedkov je zber konkrétnych dát z procesu tvorby programov, ich analýza a vyhodnocovanie. Zdrojom informácií pre vyhodnocovacie prostriedky sú programy, jednotlivé prostriedky systému BPS a taktiež užívatelia systému BPS. Problémy, ktoré s tým súvisia sa týkajú prevážne:

- výberu vhodných dát,
- metód zberu a uloženia dát,
- kódovanie chýb,
- metódy analýzy dát a pod.

V štruktúre systému BPS vyhodnocovacie prostriedky sú síce uvedené explicitne ako samostatný celok, ale je zrejmé, že sú súčasťou ďalších zložiek systému.

3. Záver

Cieľ, ktorý sme systémom BPS sledovali bol:

- Podrobiť metódu modulárneho programovania,

- Prinútit tvorcov programu buď priamo alebo nepriamo k aplikácii danej metódy pri tvorbe programu.
- Zaviesť do procesu tvorby programu silnejšiu spätnú väzbu a to všade tam, kde je to len možné a tým poskytnúť lepšiu kontrolu nad celým procesom tvorby programu.
- Mechanizovať a pokiaľ je to len možné i automatizovať tie činnosti procesu tvorby programu, ktoré môže vykonávať stroj a to omnoho rýchlejšie a spoľahlivejšie. Ide predovšetkým o činnosti mechanické pre programátora nezaujímavé, ale ktoré na druhej strane sú zdrojom častých chýb (modifikácie, optimalizácia, testovanie, dokumentácia a pod.).
- Poskytnúť tvorcom programu, užívateľom programu, manažerom softwarových projektov a tvorcom programovacích prostriedkov a techník dostatok pravdivých informácií pre analýzu ich činnosti, ktorej dôsledkom by mali byť podnety pre ďalšie vylepšenie ich činnosti.

Do akej miery sa nám tento cieľ podarí splniť ukážu aj praktické aplikácie systému BPS.

Literatúra

[Du] Duplinský, J.: Niektoré aspekty tvorby programových systémov, VP 107, VVS Bratislava, 1976

[Pe] Parnas, D.L.: A Technique for Software module Specification with example. CACM, vol 15, Number 5, May 1972

[Wi] Wirth, N.: Modula: A language for modular multiprogramming. TR.18 ETH Zurich, Jun 1976

[BTS] Bratislavský terminálový systém. Pracovný materiál VVS Bratislava.