Tomáš M A R E K prom. mat.

ZVT - OKR Ostrava

Formální standardisace programů

1. Růst objemu programovacích prací

Jsme svědky neobyčejně prudkého rozvoje nasazování výpočetní techniky do všech oblastí společenského života. Zdá se, že neexistuje VHJ a dnes již i podnik, kteří by o nasazení výpočetní techniky alespoň neuvažovali. Stále více agend se na počítačích realizuje a řeší a rozvoj techniky i teorie umožňuje stále další a další agendy na počítači realisovat a přitom složitost a komplexnost realizací neustále roste.

Samočinné počítače doznaly takového zásadního zlepšení nejen svých technických parametrů, ale rovněž podstatně i své prvotní (po instalaci) užitné hodnoty dané úrovní firemního systémového a mnohdy i náročného aplikačního software, a to v tak krátké lhůtě, že lze ztěží nalézt obdoby v jiném oboru. Avšak ani tento pokrok nás nezbavuje a zřejmě ani v budoucnosti nezbaví nutnosti vytvářet vlastní uživatelské programy. Přitom počet programů potřebných k ekonomickému vytížení počítače neustále roste. Zatímco skupinka programátorů (tehdy ještě současně analytiků) byla schopna svými programy vytížit počítač 1. generace v plném 24 hodinovém provozu, nestačily by jejich programy obstarat dnešnímu počítači práci ani na 2 hodiny denně. Takže, i když se bude podíl firemního aplikačního software na vytížení počítačů zvyšovat, absolutně bude stále ještě

potřebe uživatelských programů růst.

A to jsme zřejmě stále ještě na začátku "počítačové" exploze. Kolik integrovaných systémů, systémů a subsystémů bude vyprojektováno? Kolik programů, podprogramů, routin a subroutin bude napsáno a kolik práce a námahy bude s tím spojeno? Jaké ekonomické náklady a kolik lidského času a někdy i zdraví to všechno bude stát?

2. Vlastnosti programů a jejich kvalifikace

Nejčastěji požadované vlastnosti programů lze shrnout do následujících bodů:

1) provozní spolehlivost
2) provozní efektivnost
3) rychlost realizace programu
4) odolnost proti změnám
5) přenosnost mezi programátory

Priorita jednotlivých vlastností se samozřejmě od případu k případu mění. Jinak bude posuzován softwarový program, nebo dokonce systémová routina než aplikační program s nízkou frekvencí zpracování. Jiný názor na tuto otázku bude mít zákazník, než analytik a další názor může mít programátor. Omezíme-li se na aplikační programy běžného typu, zdá se, že priority jednotlivých vlastností jsou obvykle uvedeny ve shodném pořadí jako zde. Zastávám názor, že při dnešní úrovni výpočetní techniky nebude hodnocení podle takových priorit správné (je totiž beze změny převzato z období počítačů nižších generací), a že bychom se měli snažit tyto priority změnit. Proberme si nyní jednu po druhé výše uvedené vlastnosti.

Provozní spolehlivost. Pod tímto termínem nutno rozumět nejen spolehlivost programu při bezvadně fungujícím hardwarovém zařízení, softwarovém vybavení, za předpokladu správných vstupních dat a ostatních okolností. Spolehlivý program by se měl "chovat rozumně" i při nesplnění předpokládaných podmínek a měl by umožňovat minimalizovat ztráty vzniklé mimo něho (kontroly, restarty a pod.). Jako vlastnost programu patří provoz-

ní spolehlivost nesporně v naprosté většině případů na prvé místo a neměla by se vlastně v našem seznamu vyskytovat. Je to totiž takové základní kriterium programu, že bez jeho splnění nelze vůbec o programech hovořit. Programy, které ho podcení, způsobují značné ekonomické ztráty v provoze a mnohem větší morální škody u uživatelů.

Provozní efektivnost. Zde si nejčastěji představujeme efektivní práci programu co se týče spotřeby strojového času a přiměřené nároky na zdroje, především na vnitřní paměť počítače. Ostatní požadavky na zdroje jsou většinou dány specifikací úlohy a nelze je v etapě programování ovlivnit. Provozní efektivnost je parametr, který za dnešního stavu techniky nemá tak klíčové postavení jako dříve. Rostou rychlosti a rozsahy pamětí počítačů a klesají náklady na jednu instrukci a na jeden byte paměti. Kromě speciálních často prováděných, nebo dlouho v paměti setrvávajících routin by měla být provozní efektivnost zařazena na nižším místě.

Rychlá realizace programu. S tímto požadavkem se zřejmě budeme stále častěji u uživatelů, resp. zákazníků, setkávat, neboť mnohdy výrazným způsobem ovlivňuje úspěch celého rozsáhlého projektu. Důsledný rozvoj softwarové podpory (generátory, aj.) jakož i metodiky programování (strukturované, normované, modulární programování, aj.) mohou výrazným způsobem zkrátit termíny realizací. Ani zde však nepřeceňujme význam tohoto kriteria, i když mu náš nadřízený i zákazník přiděli prioritu nejvyšší. Je třeba si uvědomit, kolik energie se na tvorbu nových programů vynakládá - dle seriozních statistik je to méně než 20% programátorské práce. Hlavní objem programovacích prací spočívá v provádění změn ve starých programech!

Odolnost proti změnám. Moderní koncepce programu, komplexně splňující současné i budoucí požadavky uživatele je jakousi "preventivní" nebo "aktivní" odolností programu proti změnám. Je však více závislá na analytikovi, a proto zde o ní nebudeme hovořit. Druhým typem je "pasivní" odolnost proti změnám, či spíše proti jejím důsledkům. Program, u kterého nepatrná změna způsobí neočekávané potíže a je nutno je pečlivě a dlouho

ladit, nebo u kterého zdánlivě nevelká změna způsobí rozklad celé logiky, je špatný program a stává se postrachem svého okolí. Nejhorší na tom je, že nedostatky programu v tomto směru se mnohdy projeví neočekávaně při větší metodické změně systému. Čas na provedení takové změny bývá neúměrně kratší než lhůta, která byla vyhrazena na realisaci tehdy nového systému. Program nelze z časových důvodů přepracovat a termín routiny, který se nezadržitelně blíží, je ohrožen. Protože nelze reálně očekávat pokles požadavků na změny (spíše naopak), a protože již nyní, jak bylo dříve řečeno, spočívá v provádění změn naprostá většina programovacích prací, je nutno tomuto kriteriu přiřadit daleko větší prioritu, než tomu dosud je.

Přenositelnost mezi programátory. Zatím bylo častým zvykem, že autor programu měl na starosti i jeho běžnou údržbu, dokud byl k dostižení. Vztah tohoto druhu mnohdy překlenul i značné organizační mezery a programátor se svého dílka nezbavil ani po převodu do jiného útvaru, jistě k oboustranné škodě. V budoucnosti bude nutno přejít k novým formám práce. Řešitelský tým realizuje projekt, bude převeden na další problém a na údržbu bude vyčleněna minimální kapacita. Autora programu již nebude vždy možné údržbou zatěžovat. Vzhledem k rozvoji našeho oboru dojde k zvýšení počtu programátorských míst a k zvýšení počtu programátorů. Nutně se zvýší fluktuace a kritérium přenositelnosti programu bude patřit mezi nejdůležitější. Jeho podcenění může vést k obdobným havarijním situacím, jaké byly popsány při posuzování kritéria odolnosti proti změnám.

Rychlosti a výkony počítačů rostou rychleji, než jejich ceny. Řada logických operací je implantována v hardware počítače a účelně zrychluje zpracování programu. Ekonomické náklady propočtené na jednotku práce klesají.

Roste komplexní nasazení počítačů a důležitost jejich výpočtů. Případné nedostatky způsobují značné ekonomické a mnohdy nenapravitelné morální škody.

Rostou požadavky na kvalifikaci programátorů a na programátory vůbec. Celkově se zvyšuje hodnota lidské práce a roste nedostatek pracovních sil.

Proto jsem přesvědčen, že již přišla doba, kdy by program měl být posuzován dle následujícího žebříčku vlastností:

1) provozní spolehlivost
2) přenosnost mezi programátory
3) odolnost proti změnám
4) rychlost realizace
5) provozní efektivnost

3. Metodika programování

Rozvoji metodiky programování se začíná věnovat pozornost větší měrou, než tomu bylo doposud, ale horší je to již s publicitou, či zaváděním dosažených výsledků do praxe. Programovací kurzy a výukou v tomto směru vůbec nepočítají, ale omezují se pouze na popis jednotlivých výroků programu. O tom, jak řadit jednotlivé výroky za sebe tak, aby tvořily logicky ucelené bloky a aby výsledný program nejen správně fungoval, ale splňoval i náročná dříve uvedená kritéria, se nelze v příručkách dočíst. Mezi praktickými programy a školními programy z kurzů je propastný rozdíl a je na programátorovi samém, jak se s prvotními potížemi vyrovná. Postupem času si každý vypracuje svůj styl, jehož největší nevýhodou je individualita, neboť při tvorbě se nemůže programátor od opřít. Ztratí s tím spoustu času, energie a elánu.

V následujícím bych chtěl obrátit pozornost na otázku standardizace programování a programů, která jistě výrazně zvýší přenosnost programů. Standardizace je obecný rys evoluce a dříve nebo později se musí v každém oboru po prvotním období bouřlivého a živelného rozvoje dostavit. Mám za to, že dozrál čas, abychom v oblasti programování začali o této otázce uvažovat a abychom začali formovat pravidla, kterými by se mohl (a musel) programátor při své práci řídit tak, jako se například konstruktér musí řídit normou. Je pravděpodobné, že někteří budou standardizaci chápat jako omezování jejich tvůrčí činnosti či dokonce osobnosti, ale stěží naleznou pádné argumenty pro obhajobu dnešního individualismu. I v rámci standardů zůstane programátorovi bohatý prostor pro tvůrčí činnost při

vlastní programátorské práci.

Standardizace programování by měla začít již u systémové analýzy, kde dochází k rozčlenění systému na jednotlivé programy a moduly. Měla by zahrnovat volbu programovacího jazyka a metody a to podle typu programu, způsobu jeho užívání a řady dalších kritérií. A nemělo by se zapomínat ani na formální standardizaci, tj. na standardizaci vnější formy a úpravy programu. Pravidla formální standardizace by měla doplňovat metodické standardy (normalizované a modulární programování) tak, jak tomu je u strukturového programování. Vzhledem k technické nenáročnosti, účinnosti a bezprostředních výsledků standardů tohoto druhu lze doporučit jejich včasné zavedení na všech, především rozsáhlejších, pracovištích.

V následujícím vás chci seznámit s hlavními rysy standardu formální úpravy programů používaným v našem závodě již od roku 1972 a zaktualizovaným začátkem roku 1975.

4. Standard formální úpravy programu ZVT OKD

Náš standard vychází z programovacího jazyka COBOL a z metody normalizovaného programování. Všechna pravidla však, v případě nutnosti přiměřeně modifikovaná, platí i pro další programovací jazyky, popřípadě i pro programy neužívající metody normalizovaného programování.

4.1 Zásady grafické úpravy

Hlavní důraz klademe na samodokumentaci zdrojového programu a dostatkem vysvětlivek a komentářů. Minuty, které se ušetří jejich vynecháním, stojí v budoucnu často hodiny a někdy i dny. Zdrojový program musí především obsahovat základní identifikační údaje jako jméno autora, datum vzniku a stručný popis programu, ale taky popis všech větších nebo důležitějších změn spolu s jejich datem a jménem provádějícího programátora.

Dalším požadavkem je přehlednost a čitelnost programu. Doporučujeme opticky oddělovat logické bloky a úseky. Datové struktury se mají zapisovat tak, aby naznačovaly svou logickou stavbu (každá datová položka na jeden řádek, podřízené da-

tové položky se zapisují odsazené - např. o 2-3 sloupce, datové charakteristiky začínají jednotně v pevně určených sloupcích - např. ve 30, 40 a 55 sl., atd.). Výkonné výroky (v oddíle procedur) začínají vždy od 12. sloupce a píší se po jednom na řádek. Výroky, jejichž provádění je podmíněné (např. výroky obsažené v klausulích IF, AT END, WHEN), se píší odsazeně o 3-4 sloupce. Klademe značný důraz na přehledný zápis zejména složených výroků (IF---ELSE, READ---AT END, SEARCH---WHEN, PERFORM---VARYING, atd.).

Na příkladech 1 a 2 chci demonstrovat dvě krajní možnosti při popisech datových struktur, resp. při zápisu výkonných výroků. Prvním zápisem v každém příkladu sice ušetříme více než 50% děrných štítků vzhledem ke druhému způsobu zápisu; tato úspora je však jednorázová, celkové délky zápisů (v počtu vyplněných znaků) jsou však shodné a ztráta přehlednosti je tak značná, že by se podobné případy neměly vůbec vyskytnout.

Příklad 1: Úsporný a logicky přehledný popis téže datové struktury:

```
01 E1-SRAZKA. 05 E1-CITAC PIC S9(9) COMP. 05 E1-ROKMES.
10 E1-ROK PIC XX. 10 E1-MES PIC XX.
05 E1-IDENT. 10 E1-PODZAV. 15 E1-POD PIC XX. 15 E1-ZAV PIC XX.
10 E1-CZN PIC S9(9) COMP.
05 E1-DRUH PIC X(3). 05 E1-ZNAK PIC X.
05 E1-KCS PIC S9(9) COMP OCCURS 2 TIMES.
```

```
01 E1-SRAZKA.
   05 E1-CITAC      PIC S9(9) COMP.
   05 E1-ROKMES.
      10 E1-ROK     PIC XX.
      10 E1-MES     PIC XX.
   05 E1-IDENT.
      10 E1-PODZAV.
         15 E1-POD  PIC XX.
         15 E1-ZAV  PIC XX.
      10 E1-CZN     PIC S9(9) COMP.
```

```
05 E1-DRUH    PIC X(3).
05 E1-ZNAK    PIC X.
05 E1-KCS     PIC S9(9) COMP    OCCURS 2 TIMES.
```

Příklad 2: Úsporný a přehledný zápis téhož složného výroku:

```
IF I = J THEN ADD 1 TO I ADD 2 TO J
MOVE 12 TO L GO TO CTENI ELSE ADD 2 TO I
MOVE ZERO TO J MOVE 1 TO L GO TO ZAPIS.
```

```
IF I = J
   ADD  1          TO I
   ADD  2          TO J
   MOVE 12         TO L
   GO TO CTENI
ELSE
   ADD  2          TO I
   MOVE ZERO       TO J
   MOVE 1          TO L
   GO TO ZAPIS.
```

4.2 Členění programu a volba jmen návěští.

Programy jsou členěny do bloků (sekcí) dle metody normalisovaného programování a toto členění je závazné i pro programy, které ze závažných důvodů této metody neužívají. Jsou to následující bloky:

A	blok úvodních operací
B	blok čtení souborů
C	blok výběru věty
D	blok testů změn klíčů
E	blok zpracování věty
F	blok závěrečných operací

G,H,I a T bloky subroutin a pomocných tiskových subroutin.

Odstavce (paragrafy) patřící k témuž bloku se píší bezprostředně za sebou, jsou očíslovány a dle těchto čísel vzestupně uvnitř bloku seřazeny. Bloky se píší v pořadí dle výše uvedeného seznamu. Názvy návěští obsahují část identifikační

a část mnemotechnickou a mají následující strukturu:

s-mnemotechnické jméno-m,

kde s znamená předznačení bloku písmenem (A-I,T) dle výše uvedeného seznamu a

m je vzestupné číslování návěští v rámci bloku (s krokem 5 nebo 10).

Odpovídající blok vývojového diagramu je označen s-m a dle téže identifikace s-m jsou všechna návěští v programu seřazena. Mnemotechnická část návěští zajišťuje srozumitelnost a identifikační část návěští zajišťuje přehlednost programu, přičemž délka jména návěští zůstává stále na přijatelné úrovni.

Příklady:

A-NULUJ-TAB1-20,
G-SUMA-ZAVOD-100,
T-HLAVA-50, atp.

4.3 Jména souborů a identifikátory dat.

Každému souboru v programu je jednoznačně přiděleno jednopísmenné předznačení předcházející mnemotechnický název. Z hlediska tohoto pravidla se sekce vnitřní paměti (WORKING-STORAGE SECTION), resp. sekce konstant (CONSTANT SECTION), resp. spojovací sekce (LINKAGE SECTION) považují rovněž za datové soubory a jsou jim přidělena předznačení W, resp. C, resp. L.

Příklady:
D-STITKY
K-KMEN
P-CHYBY
T-TISK

Identifikátory datových položek označují příslušnost k oblasti dat v programu a věcný význam datové položky. Je z nich patrno, v které části programu lze nalézt její deklaraci a jaký bude její obsah během chodu programu. Identifikátory dat mají následující strukturu:

si-mnemotechnické jméno,

kde a je předznačení souboru a

i je pořadové číslo věty (rekordu) v rámci souboru či sekce.

Rekordy jsou uvnitř souboru či sekce seřazeny vzestupně dle svých pořadových čísel. Mnemotechnické jméno (zkratku) volíme tak, aby co nejsrozumitelněji vyjadřovala obsah datové položky a aby nebyla příliš dlouhá. Tyto dva protichůdné požadavky je nutno řešit zdravým kompromisem a pro nejrozšířenější datové položky zavádíme celozávodně závazný seznam mnemotechnických zkratek.

Příklady:

F1-PODZAV	číslo podniku-závodu z prvního rekordu souboru F
C12-TARIF	tarif ze dvanácté tabulky v sekci konstant
W3-PODZAV	číslo podniku-závodu z třetího rekordu v pracovní paměti
L1-OKRES	číslo okresu z prvního rekordu spojovací sekce

Při volbě mnemotechnického jména vycházíme z logických a ne fyzických charakteristik datové položky, které se mohou snadno změnit (tedy např. volíme CITAC, DELKA, INDEX místo SLOVO, BUNKA, BIT, ZNAK). Vyhýbáme se nesrozumitelným, matoucím, nic neříkajícím a pozornost odvádějícím jménům. Snažíme se rovněž respektovat alespoň nejzákladnější lingvistická pravidla.

Výjimky z výše uvedených pravidel pro volbu identifikátorů datových položek platí pro kvalifikovaná data, subskripty, indexy, výhybky, pomocná pole, klíčové oblasti, aj.

5. Závěr.

Situace v našem závodě a zejména fluktuace a obtížná zastupitelnost programátorů nás mutí mimo jiné formálně standardizovat zdrojové programy. Je to prostý, ale účinný způsob, jak podstatně zvyšovat schopnost přenosu programů mezi programátory, což považujeme vedle spolehlivosti za nejdůležitější

parametr programu. Před definitivní formulací závazného standardu jsme trpělivě diskutovali jeho zásady s vedoucími programovacích týmů, neboť jsme potřebovali, aby kvalitu vnější úpravy programů považovali za svou věc. Bez jejich kladného přístupu ke standardům by všechny snahy o zlepšení v tomto směru ztroskotaly, neboť by nebyly ničím více, než administrativními opatřeními. Je ovšem pravdou, že pádným argumentem napomáhajícím našim snahám byla objektivní nutnost provádět mnoho značných změn do cizích programů za nepřetržitého rutinního provozu.

Každému novému programátorovi u nás jsou v nástupním kursu programování vštěpovány zásady metody normalizovaného programování a standardu formální úpravy zdrojových programů. Dnes můžeme říct, že jsme v odboru programování přesvědčili všechny pracovníky o jejich účelnosti a že je všechny nové programy respektují. Předávané programy přestaly být postrachem a jejich údržba se zkvalitnila a zrychlila. Horší je situace již v odboru systémové analýzy a čeká nás tam ještě spousta osvětové práce.

Jsem přesvědčen, že podobná situace nastala, nebo musí nastat v nedaleké budoucnosti ve všech výpočtových střediscích a proto jim vřele doporučuji se touto problematikou zabývat a nepodceňovat ji. Věřím, že isolované vznikající podnikové standardy se stanou základem pro vznik obecnějších, například oborových norem.

Na závěr se chci ještě zmínit o politováníhodném nešvaru, který se ještě v našich publikacích někdy vyskytuje. Někteří autoři považují za nutné "prošpikovat" své ukázky programů vtipnými identifikátory typu PES, KOČKA, KOZA, TELE atp. Neuvědomují si, že špatný příklad se snadno následuje a špatný zvyk se špatně odstraňuje. Snažme se využít každé příležitosti k propagaci dobrých snah a tento nešvar brzo vymýtit.

Literatura:

1. Kalhous,O: Normované programování aneb jak každý programátor podle svého blaženosti dojde, Mechanizace a automatiza-

be administrativy č. 2, ročník XI/1971

2. Průša,K: Lze programování racionalizovat?, Mechanizace a automatizace administrativy č. 1, ročník 12/1972

3. Metzl,K: Metoda normalizovaného programování ve zpracování hromadných dat, seminář Metody programování počítačů III. generace, 1975

4. Čevela Vlastimil: Uspořádaná logika a organizace programování, Mechanizace a automatizace administrativy č. 2, ročník XV/1975

5. Brzický Josef: Progresívní směry v programovací technice, Mechanizace a automatizace administrativy č. 8, ročník XV/1975

6. Standard č. 3 - formální úprava zdrojových programů, Závod výpočetní techniky OKD, 1975