PROGRAMOVÁNÍ BEZ PROGRAMÁTORŮ

Josef Tvrdík, Karel Metzl
KNS Ostrava, OKR-AŘ Ostrava

Abstrakt: Práce se zabývá prostředky pro vývoj aplikačních programů koncovým uživatelem. Jsou uvedeny příklady užití dotazovacích jazyků ON-LINE ENGLISH, QUERY-BY-EXAMPLE a DATATRIEVE a programovacích jazyků APL a NOMAD. Diskutovány jsou technologické a organizační změny potřebné pro účelné využití těchto nových programovacích prostředků.

1. Úvod

Podnětem k napsání tohoto příspěvku byly dvě práce /1,2/, které mají řadu společných vlastností: autoři obou jsou uznávané autority v oblasti programování a aplikací výpočetní techniky, obě práce se zabývají otázkami blízké budoucnosti využívání počítačů a oba autoři ukazují na nutnost kvalitativních změn v technologii programování a celém přístupu k vývoji software. Obě práce byly vydány v r. 1982. Nelze samozřejmě ve všem srovnávat několikastránkový záznam přednášky /1/ s více jak třísetstránkovou knihou /2/, navíc autoři se liší v hodnocení hlavních zdrojů kvalitativních změn v programování v blízké budoucnosti. Proto dobře dostupnou práci /1/ doporučujeme k zevrubnému přečtení a dále v příspěvku se budeme zabývat především obtížněji dostupnou knihou /2/.

Některé zajímavosti týkající se budoucnosti aplikací výpočetní techniky a programování byly prezentovány již v panelové diskusi o programovacích technologiích na semináři Programování 85. Nyní se zaměříme především na prostředky, které (i u nás) mohou přispět ke zkrácení doby vývoje aplikačního programového vybavení a podstatně přiblížit vývoj programového vybavení koncovému uživateli.

2. Nutnost kvalitativních změn v programovací technologii

I podle velmi střízlivých odhadů se předpokládá, že v průmyslově vyspělých státech stoupnou v nastávajícím desetiletí výpočetní kapacity nejméně stonásobně. Uvedený nárůst bude zajištěn

jednak přibýváním nových počítačů, ale i většími rychlostmi a kapacitami pamětí. Vytvořit běžně používanými prostředky programové zabezpečení pro takovou hardwarovou sílu je nad možnosti analytiků a programátorů. Představa, že bylo nutno jejich stavy stonásobně zvýšit, působí komicky. Vždyť např. v USA by to znamenalo 28 milionů programátorů. Je zajímavé, že s podobným technickým problémem se společnost již setkala. Na začátku století, kdy došlo k rychlému zavádění telefonů, se také zdálo, že třetina lidstva bude muset být zaměstnána jako přepojovatelky v místních i meziměstských sítích. A přece se našlo řešení - zavedení automatických ústředen v počátcích aspoň do nejfrekventovanějších místních sítí a později i do meziměstského a mezistátního styku.

Nemusíme se ale dívat do příliš vzdálené budoucnosti, abychom zjistili, že zárodky blížících se potíží se projevují již dnes. Průběhem mnohaletého vývoje se ustálil následující model organizace analyticko-programátorských prací. Uživatel předloží své požadavky, analytik podle nich vytvoří specifikaci subsystému navrhne vstupy, výstupy i datové soubory. Hotové dílo předloží pak uživateli ke schválení a podpisu. Ke slovu se pak dostávají programátoři, kteří podle předložených specifikací zakódují programy v COBOLu nebo PL/I a odladí je. Zdánlivě je vše v pořádku. Podívejme se ale na úskalí, která se v tomto postupu skrývají. Hned v začátku, když se sejde uživatel s analytikem, jde o setkání dvou zcela odlišných kultur. Každý z nich mluví jiným jazykem a třeba i pod stejnými pojmy vidí každý něco jiného. Když analytik vypracuje své objemné dílo, uživatel mu ho klidně podepíše. Ne proto, že ho prostudoval, rozuměl mu a souhlasil s ním, ale protože ví, že bez podpisu by se nezačalo s programováním. Ani analytik nepřikládá podpisu příliš velkou váhu. Zkušenost ho poučila, že uživatel uvidí první výsledky, přijde se změnami a často vše postaví na hlavu. Typický příklad rozdělení chyb a nákladů na jejich odstranění je uveden v následující tabulce.

	Rozdělení chyb	Náklady na odstranění
Chyby v požadavcích	56 %	82 %
Chyby v návrhu	27 %	13 %
Chyby v kódu	7 %	1 %
Jiné chyby	10 %	4 %

Tradiční postup ještě jakž takž vyhovoval u dávkových agend, u nichž se daly poměrně snadno určit požadavky uživatele. Se zaváděním databázových systémů, displejových terminálů a při snaze využívat počítače k rozhodování a řízení se ukázal tento přístup jako zcela nevyhovující. Doba vývoje je příliš dlouhá a výsledek přes vynaložené náklady není pro praxi přínosem.

Jediná cesta, jak se dostat z popsaných potíží a připravit se na nástup nové výkonější techniky, je zvýšit řádově produktivitu práce při vývoji aplikací a zapojit do tohoto vývoje i koncové uživatele. Obojí vyžaduje nový podpůrný software a novou organizaci práce.

3. Prostředky pro vývoj programového vybavení bez tradičního programování.

3.1 Typy softwarové podpory

Martin rozlišuje sedm typů softwarových prostředků pro vývoj aplikací bez tradičního programování:

- Jednoduché dotazovací prostředky pro výpisy vět ze souborů v přijatelném formátu. Tyto prostředky existují od zavedení prvních diskových pamětí. Sem patří např. některé utility a služební programy.
- Komplexní dotazovací jazyky dovolují uživateli formulovat i složitý dotaz na obsah databáze. Dotaz se může týkat i více vět a jejich vztahů. Jazyky tohoto typu jsou zpravidla určeny jen pro jednu organizaci datové základny, ať již databázovou nebo souborovou. Řada dotazovacích jazyků tohoto druhu dovoluje nejen čtení, ale i aktualizaci datové základny.
- Generátory sestav jsou prostředky, které dovolují výběr specifikovaných dat z datové základny a jejich výstup ve formě zpráv. Dobré generátory sestav dovolují základní numerické zpracování (průměry, součty, četnosti hodnot ap.) a měly by být přirozeným rozšířením dotazovacího jazyka snadno použitelným pro zkušenějšího uživatele.
- Grafické jazyky jsou vlastně dalším rozšířením možností výstupu předchozích dvou typů softwarových prostředků. U nás jsou prozatím důmyslnější grafické prostředky pro zpracování dat známy

spíše z prospektů nebo z programového vybavení dovezených mikropočítačů (VISIPLOT s.p.). Martin upozorňuje na užitečnost grafických jazyků ve spojení se simulačními modely.

- Generátory aplikačních programů umožňují automatizované generování programů nebo modulů "na míru" podle požadavků uživatele. Vygenerovaný kód lze doplňovat moduly vytvořenými tradiční technologií.
- Programovací jazyky velmi vysoké úrovně jsou procedurální jazyky dovolující podstatně kratší zápis programu než v tradičních jazycích COBOL, FORTRAN nebo PL/I. Při tom jazyky mají být tak jednoduché, aby je zvládnul koncový uživatel, jehož hlavní pracovní náplní není programování. Počet příkazů programu v jazyku velmi vysoké úrovně má být asi 20 až 40 krát menší ve srovnání s COBOLem.
- Parametrické balíky aplikačních programů jsou určeny pro určitou oblast zpracování dat u různých uživatelů, často i na různých počítačích. Právě vhodná úroveň parametrizace a neprocedurální řídící jazyk umožňují snadné využití takových programů koncovými uživateli v mnoha kontextech zpracování dat. Příkladem parametrických balíků jsou programové systémy pro statistickou analýzu dat /3,4/.

Domníváme se, že z uvedených typů softwarové podpory vývoje aplikačních programů bez tradičního programování jsou pro nás aktuální nebo aspoň inspirující především dotazovací jazyky, programovací jazyky velmi vysoké úrovně a parametrické aplikační balíky. Parametrickým programům byl věnován seminář Programování 82 - viz sborník /5/. Příklady dotazovacích jazyků a programovacích jazyků velmi vysoké úrovně budou uvedeny v následujících odstavcích.

3.2. Dotazovací jazyky

3.2.1 ON-LINE ENGLISH

Jako příklad dotazovacího jazyka velmi blízkého přirozenému jazyku anglosaského uživatele uvádí Martin ON-LINE ENGLISH. Tento jazyk využívá rozsáhlého slovníku základních slov, který je součástí definice jazyka a je společný pro všechny aplikace a slovníku problémově orientovaného, který je definován správcem databáze a může být rozšiřován (zatím ne automaticky) na základě "učení se"

z uživatelovy komunikace. Do základního slovníku patří slova jako WHAT, WHEN, IN, COUNT, STATISTICS, HOW MANY, ALL, FIELD NAME a.p. Druhá část slovníku obsahuje pojmy daného problémového okruhu, např. EMPLOYEE, TITLE, SALARY atd. Pro "přirozenou" komunikaci je důležité, aby pojmy ve slovníku bylo možno vyjádřit i synonymy, např. WORKER, JOB, PAY atd.

Uživatelovy dotazy na soubor zaměstnanců pak mohou probíhat asi takto (U - uživatel, T - terminál):

```
U: WHAT ARE THE NAMES OF ACCOUNTANTS IN NEW YORK
T: PRINT NAME OF ALL EMPLOYEES WITH
   CITY=NEW YORK
   AND JOB=ACCOUNTANT
```

Po schválení se na terminálu vypíše počet vět vyhovujících podmínce, nadpis a seznam jmen všech účetních pracujících u firmy v N.Y. Uživatel může dále svůj dotaz zpřesnit:

```
U: WHICH ONES EARN MORE THAN 20000
T: PRINT NAME OF ALL EMPLOYEES WITH
   CITY=NEW YORK
   AND JOB=ACCOUNTANT
   AND SALARY > 20000
```

a pak opět následuje nadpis a seznam jmen.

Program nemusí vždy dotazu rozumět, pak může komunikace probíhat např. takto:

```
U: WHAT ARE THE NAMES OF ACCOUNTANTS IN POUGHKEEPSIE
T: I'M NOT FAMILIAR WITH THE WORD "POUGHKEEPSIE"
   IF IT'S A WORD YOU EXPECT TO FIND IN THE DATABASE HIT THE
   RETURN KEY.
   OTHERWISE EITHER FIX ITS SPELLING OR ENTER A SYNONYM FOR IT
U: stiskne klávesu RETURN
T: WHAT FIELD SHOULD IT APPEAR IN?
U: CITY
```

a pak se na terminálu objeví interpretace dotazu a vybrané informace z datové základny podobně jako v předchozích příkladech. Uživatel má možnost vyptávat se na názvy ze slovníku a tak postupně formulovat svůj dotaz.

Dotaz ve "skoro" přirozeném jazyce může jistě udělat velký dojem na občasného uživatele. Užívání takového dotazovacího jazyka má však svá úskalí. Přirozenost jazyka je pouze předstíraná a každé předstírání stojí úsilí. V tomto případě údržbu slovníku, dosti složitou interpretaci dotazů, vyhledávání ve slovníku a ošetření stavů vzniklých z nedorozumění nebo z drobných překlepů. I Martin upozorňuje na otíže způsobené nejednoznačností a nestrukturovaností přirozeného jazyka.

Další obtíže se ukáží, když se pokusíme jednoduché a srozumitelné věty uvedeného příkladu přeložit do ON-LINE češtiny. Čeština má proti angličtině volnější pravidla větné stavby, složitější skloňování, ohebné slovní druhy atd. Většina současných terminálů nemůže využívat celou českou abecedu. O některých otázkách české komunikace s počítačem se lze dočíst v /7/. Dotazovací jazyk blížící se přirozené češtině není patrně nejschůdnější cestou účelné komunikaci uživatele - neprogramátora s počítačem.

3.2.2 QUERY-BY-EXAMPLE

QUERY-BY-EXAMPLE uvádí Martin jako příklad dotazovacího jazyka vhodného pro počítačově nevzdělaného uživatele, který nechce nic vědět o formálním popisu dotazu a datových strukturách. Komunikace uživatele v tomto jazyku probíhá vyplňováním formulářů (tabulek), které jsou zobrazeny na displeji. Uživatel vyplňuje tabulku požadovanou funkcí (P. - zobraz, U. - aktualizuj, I. - vlož, D. - vymaž), hodnotou nebo příkladem hodnoty. Nabízené tabulky odpovídají vazbám mezi datovými položkami v datové základně.

Na př. odpověď P. v prázdné tabulce

P.		

znamená výpis všech dostupných tabulek. V následujících tabulkách je to, co píše počítač, uvedeno malými písmeny a text zapsaný uživatelem velkými písmeny.

Zadání

ZAMESTNANCI	P.	P.	P.	P.

vypíše tabulku věty ZAMESTNANCI s názvy položek,

zamestnanci	cis-zam	jmeno	plat	oddeleni
		P.		KONSTRUKCE

výše uvedená odpověď specifikuje výpis seznamu jmen zaměstnanců v oddělení KONSTRUKCE. Podobně jen odpověď P. v posledním sloupci (oddělení) znamená výpis názvů oddělení. Zadání

zamestnanci	cis-zam	jmeno	plat	oddeleni
P.				KONSTRUKCE

vypíše úplnou tabulku s údaji o zaměstnancích v oddělení konstrukce. Dotaz v prázdné tabulce

P.	PLAT	

znamená požadavek výpisu názvu všech tabulek, které obsahují položku PLAT.

Uživatel může při zadání dotazu užívat relačních operátorů $>, <, \geq$, = atd. Kromě toho může algebraicky formulovat příkaz pro jistý objekt nebo skupinu objektů:

zamestnanci	cis-zam	jmeno	plat	oddelení
U.			500xl.1	KONSTRUKCE

Tento příkaz znamená aktualizaci položky PLAT, a to zvýšení platu u všech zaměstnanců oddělení konstrukce o 10 %. Výraz 500 znamená symbolické označení položky, na číselné hodnotě nezáleží, je to symbolické vyjádření příkladem nějaké možné hodnoty této položky.

Podobně dotaz

zamestnanci	cis-zam	jmeno	plat	oddeleni
		P. NOVAK		

znamená výpis jmen všech zaměstnanců, jejichž jméno začíná na N. Pomocí takto příkladem formulovaných dotazů lze jednoduše zadávat velice "pálené" požadavky týkající se více než jedné tabulky. Lze využívat standardních funkcí SUM, AVG (průměr), CNT (počet), MAX, MIN. Tak např. průměrný plat v oddělení konstrukce se zjistí dotazem

zamestnanci	cis-zam	jmeno	plat	oddeleni
			P. AVG	KONSTRUKCE

Za přednosti jazyka QUERY-BY-EXAMPLE se považuje:
- tabulková forma popisu dat i výsledků
- neužívají se klíčová slova (s výjimkou příkazů P.,U.,I.,D. a názvů několika standardních funkcí)
- uživatel dostává explicitní popis dat se kterými pracuje
- jazyk lze přirozeně rozšiřovat
- dialog se snadno naučí i počítačově naivní uživatel.

Martin uvádí příklad sekretářky, která nikdy předtím nepracovala s počítačem, a přesto po půlhodině výuky QUERY-BY-EXAMPLE pracovala rozumně s datovou základnou.

3.2.3 DATATRIEVE

V dlouhém seznamu dotazovacích jazyků vhodných pro koncového uživatele uvádí Martin i DATATRIEVE firmy DEC. Podrobněji se jím nezabývá, ale vzhledem k tomu, že je to jazyk dostupný a užívaný i u nás (na počítačích SMEP pod názvem DTS), považujeme za účelné se o něm zmínit.

DATATRIEVE je určen k interaktivní práci s datovou základnou. Postupně interpretuje a provádí z terminálu zadané příkazy pro operace s daty. Syntaxe příkazů je velmi volná, využívá se anglických klíčových slov, operandy jsou výrazy tvořené z datových objektů (logických názvů souborů, datových položek) definovaných v uživatelově slovníku, konstant, proměnných a běžných logických a aritmetických operátorů.

Rozsah funkcí DATATRIEVu pokrývá možnosti uváděné v předchozím odstavci u jazyka QUERY-BY-EXAMPLE. Navíc jeho součástí je i generátor sestav a některé další prostředky pro náročnější a

racionální využívání (editor definic ve slovníku, možnost definice procedur, definice přístupových práv různých uživatelů k různým souborům atd.).

Uživatelova komunikace může probíhat např. takto (údaje zadávané uživatelem jsou podtrženy):

```
DTR > SET DICTIONARY OSOB
DTR > READY ZAMESTNANCI
DTR > FIND ZAMESTNANCI WITH ODDELENI="KONSTRUKCE"
DTR > (10 RECORDS FOUND)
DTR > SORT BY JMENO
DTR > PRINT ALL JMENO
```

Touto sekvencí příkazů se provede připojení dříve definovaného slovníku OSOB, který obsahuje definice vět a logických souborů (případně procedury a další definice). Dále se zpřístupní logický soubor ZAMESTNANCI pro čtení, vytvoří se přechodný soubor zaměstnanců oddělení konstrukce, ten se abecedně setřídí podle jmen zaměstnanců a setříděný seznam jmen se vypíše na terminál.

Podobným jednoduchým způsobem lze formulovat i podstatně složitější dotazy a další požadavky na zpracování dat (aktualizace, transformace, přidávání nových vět, vytváření nových souborů atd. Výpisy lze směrovat na různá výstupní zařízení nebo soubory. Styl dialogu a syntaxe příkazů navazuje na pravidla známá z obvyklých programovacích jazyků. Diskutabilní je, zda DATATRIEVE nebo jeho část je opravdu jazyk vhodný pro koncového uživatele. Zkušenosti z našeho pracoviště ukazují, že někteří uživatelé zvládnou dotazovací příkazy. Definice slovníkových objektů, t.j. popisy vět a souborů, procedury a ovládání generátoru sestav však přece jen vyžaduje znalosti, které po koncovém uživateli nelze požadovat. Věty jsou stromové datové struktury popisované pomocí PICTURE podobně jako v COBOLu nebo PL/I. Definování věty tedy vyžaduje i znalosti o možnostech uložení dat v počítači. Rychlost zpracování dotazu zvláště u rozsáhlejších dat velmi podstatně závisí na organizaci datových souborů (sekvenční nebo index-sekvenční, sekundární klíče). Proto je asi vhodné, aby koncovému uživateli bylo ponecháno jenom dotazování. Údržbu slovníků, ostatní činnosti pro vytváření prostředí pro práci koncového uživatele a řešení neobvyklých požadavků by měl vykonávat kvalifikovaný pracovník - správce datové

základny. Ostatně i obsáhlý manuál /6/ je určen především čtenáři programátorovi nebo administrátorovi databáze. U manuálu DTS /8/ pro operační systém DOS RV je velmi obtížné určit jakým čtenářům je určen - snad detektivům, neboť tolik nepřesností a rozporných formulací se v počítačových příručkách vyskytuje jen vyjímečně.

3.3. Programovací jazyky velmi vysoké úrovně

3.3.1 APL

APL uvádí Martin jako jeden z jazyků vhodných pro koncového uživatele. Za největší přednosti APL je považováno:

- základy jazyka a jeho použití lze zvládnout velmi rychle (Martin uvádí 5 minut pro základní kalkulačkové funkce)
- snadnost vektorových a maticových operací
- stručnost zápisu programu
- přirozené rozšiřování o subrutiny současně s rozvojem znalostí uživatele
- firemně (IBM) dodávané doplňky APL pro zpracování dat.

V jazyku APL je realizována značná část programů pro zpracování interních dat firmy IBM. Zkušenosti IBM ukazují, že čas potřebný na vývoj programu v APL je 4 - 10 krát kratší než v PL/I nebo v COBOLu. V APL může i dosti složitou aplikaci programovat jeden člověk, zatímco v COBOLu je pro tutéž práci nutný tým řešitelů.

Silné i slabé stránky APL asi nejlépe ve stručnosti ilustruje těchto několik příkladů:
Součet zadaných prvků vektoru, maximum, minimum

```
U: A←12 14 16 18 20      RETURN
   +/A                   RETURN
T: 80
U: ⌈/A                   RETURN
T: 20
U: ⌊/A                   RETURN
T: 12
```

Poněkud neobvyklé je, že v APL jsou výrazy vyhodnocovány zprava, takže příkaz U: -/A vyvolá odpověď T: - 40,
neboť (((20-18)-16)-14)-12)=-40

Příkazy vstupu a výstupu mají jednoduchý tvar:

```
U: A←□          RETURN
T: □
U: 23 42 56 78 12 RETURN
U: □←A
T: 23 42 56 78 12
```

Průměr lze vypočítat jednořádkovým příkazem

```
U: +/A ÷⍴A      RETURN
```

(⍴ A je funkce APL, která znamená počet prvků vektoru A)

Dokonce i výpočet směrodatné odchylky lze zadat na jeden řádek:

```
U: +(+/((A-(+/A)-⍴A)*2)÷⍴A←□)*.5    RETURN
T: □
U: 10 12 14 16 18 20 22 24          RETURN
T: 4.582575695
```

Tak nepřehledný zápis je možný, ale není nutný. V následujícím příkladu definice subrutiny jsou příkazy pro výpočet směrodatné odchylky rozepsány na více řádků

```
T:      U:
        ∇      SD
(1)             A←□                 RETURN
(2)             M←(+/A)÷⍴A          RETURN
(3)             B←(A-M)*2           RETURN
(4)             C←+/B               RETURN
(5)             D←(C÷⍴A)*0.5        RETURN
(6)             'PRUMER=';M         RETURN
(7)             'SM.ODCHYLKA=';D    RETURN
(8)     ∇
```

Subrutinu lze jednoduše volat:

```
U: SD                               RETURN
T: □
U: 10 12 14 16 18 20 22 24          RETURN
T: PRUMER=17
   SM.ODCHYLKA=4.582575695
```

V APL je možno psát interaktivní programy, podobně jako v jiných jazycích a přitom využívat maticové prostředky jazyka pro výpočty a pro relační operace nad malými datovými základnami. APL lze využívat i v rámci jiných softwarových prostředků dodávaných IBM - APLDI a ADRS. APLDI je dotazovací jazyk pro inter-

aktivní práci s databází, ve kterém jednou z voleb je zpracování vyhledaných dat pomocí APL. ADRS je generátor sestav, ve kterém k numerickému zpracování dat je možné užít příkazů APL.

Martin uvádí, že svět se dělí na nadšené příznivce a nebo na zásadní odpůrce jazyka APL. Příznivci jsou spíše z řad matematicky založených uživatelů než z řad programátorů. Ti jen údajně s oblibou soutěží v napsání co nejrozsáhlejšího programu na jeden řádek. Právě nepřehlednost zápisu příkazů a nepodobnost běžným programovacím jazykům představuje podle našeho soudu hlavní překážky v rozšíření jazyka APL k praktickému využití koncovými uživateli.

3.3.2 NOMAD

Zatím co jazyk APL byl navržen pro složité výpočty, byl NOMAD původně určen pro ukládání dat a manipulaci s databankou. Stejně jako jiné jazyky čtvrté generace používá NOMAD svůj vlastní databázový systém, generátor zpráv i grafiky a programovací jazyk velmi vysoké úrovně. Jazyk obsahuje procedurální i neprocedurální prvky. Základní podmnožina jazyka je složena hlavně z neprocedurálních výroků a je snadno pochopitelná i uživatelům, kteří nikdy neprogramovali. Tím, že jazyk pracuje na předem definovaných databázích, odpadá uživateli nutnost starat se o data t.j. o jejich vstupní i výstupní formáty, způsob uložení atd.

Velmi jednoduchým zápisem např.

```
LIST CUSTNAME ADDRESS LASTDATE ACCOUNTTOTAL
```

získá uživatel seznam zákazníků spolu s dalšími požadovanými údaji. Navíc NOMAD sám seznam vhodně rozvrhne na obrazovku, stránkuje, opatří hlavičkami nad sloupci, v položce LASTDATE oddělí lomítky dny, měsíce, roky atd. Přeje-li si uživatel mít seznam setříděný podle zákazníků, stačí výrok doplnit na

```
LIST BY CUSTNAME .....
```

Přidá-li výrok

```
LIST AVERAGE (ACCOUNTTOTAL)
```

získá navíc průměr předcházejících hodnot.
Stejně jednoduchým způsobem se zapisuje výrok PLOT, který dokáže názorně zakreslit závislosti jednoduchým grafem.

Ačkoliv NOMAD lze prospěšně využívat již s malým počtem neprocedurálních snadno použitelných příkazů typu LIST a PLOT, je kompletní jazyk velmi obsažný a jeho možnosti jsou srovnatelné např. s COBOLem. Kromě řady neprocedurálních výroků obsahuje i běžné procedurální prvky jako IF-THEN-ELSE, DO bloky, výrok FOR pro smyčky a ON pro vyjimečné stavy.

Charakteristickým rysem NOMADu je, že pracuje s vlastním typem databázové organizace, která umožňuje ukládat data relačním způsobem, ale také hierarchicky pomocí směrníků automaticky dodávaných do záznamů. Vzhledem k tomu, že do databází může přistupovat každý koncový uživatel, má NOMAD zabudovanou běžnou ochranu heslem, výběrovým omezením přístupu buď k aktualizaci nebo ke čtení s možností kryptografického zašifrování některých údajů. Důmyslné jsou i kontroly vstupujících dat. Např. popisy položek

```
ITEM VENDNUM AS MEMBER 'VENDORS';
ITEM INVNUM AS A6 MASK 'AAX599';
ITEM QTY AS 9999 LIMITS (0:400);
```

udávají, že v položce VENDNUM musí být platné číslo dodavatele, jednoho z našeho seznamu dodavatelů. Položka INVNUM musí být šestiznaková, přičemž první dva znaky musí být abecední, další znak je alfanumerický a následuje třímístné číslo menší než 600. V položce QTY musí být zadané množství v rozsahu 0 až 400.

NOMAD umožňuje měnit hromadně celé sloupce údajů nebo polí jednoho typu jediným výrokem. Např.:

```
CHANGE SALARY = SALARY x 1.07 WITHIN DATA BASE
```

zvýší platy všech zaměstnanců v uvedené databázi o 7 %. Předřadí-li se výrok

```
SELECT DEPT = 'ACCOUNTING'
```

bude se zvýšení týkat pouze zaměstnanců účtárny.

Uvedený způsob hromadných změn je zvláště výhodný pro modelování dotazů typu "Co by se stalo, kdyby....", které často potřebují vedoucí pracovníci pro kvalifikované rozhodování. Význačným rysem NOMADu v této oblasti je možnost uchovat původní údaje a po skončeném modelování vrátit databázi do původního stavu.

Jinou výhodnou vlastností NOMADu je možnost jednoduchého zařazení dat z jiných databází nebo datových souborů do vnitřních

databází NOMADu. Pomocí neprocedurálního výroku LOAD lze při tom zajistit i potřebné konverze a úpravy do požedovaného tvaru.

V praxi se NOMAD využívá dvojím způsobem. Buď ho používají jednotliví pracovníci, kteří si budují vlastní databáze a používají je pro organizaci své "bezpapírové" kanceláře t.j. k budování kartoték, vyřizování pošty a případně i pro modelování. Druhý způsob představuje hromadné nasazení pro více uživatelů. Tento způsob je výhodnější, ale vyžaduje pomoc ze strany oddělení pro zpracování dat, které se musí postarat o administraci dat a o šíření vyvinutých aplikací i mezi ostatní uživatele.

4. Organizační předpoklady zavádění nového software

Zavádění nového software čtvrté generace a soustavné zapojování koncových uživatelů do vývoje aplikací si vyžádá i změny v náplni práce systémových analytiků, programátorů a v jejím řízení. Systémový analytik, který byl dříve prostředníkem mezi uživatelem a programátorem a jehož hlavní náplní bylo psát objemné specifikace pro obě strany, se stane spolu s koncovým uživatelem přímým tvůrcem aplikací. Předpokládá se, že většinu času bude trávit mezi koncovými uživateli a zacvičovat je v používání nových nástrojů. Tam, kde nepůjde z důvodů strojové efektivity aplikaci přímo vygenerovat, měl by analytik vytvořit ve spolupráci s uživatelem a na jeho terminálu alespoň t.zv. prototyp. T.j. navrhnout zprávy, rozvrhy obrazovek, dialogy tak, aby si mohl uživatel na modelu vyzkoušet chování aplikace. Pak teprve poslouží prototyp k vygenerování (nejlépe automatickému) skutečné aplikace a jejímu provoznímu doladění.

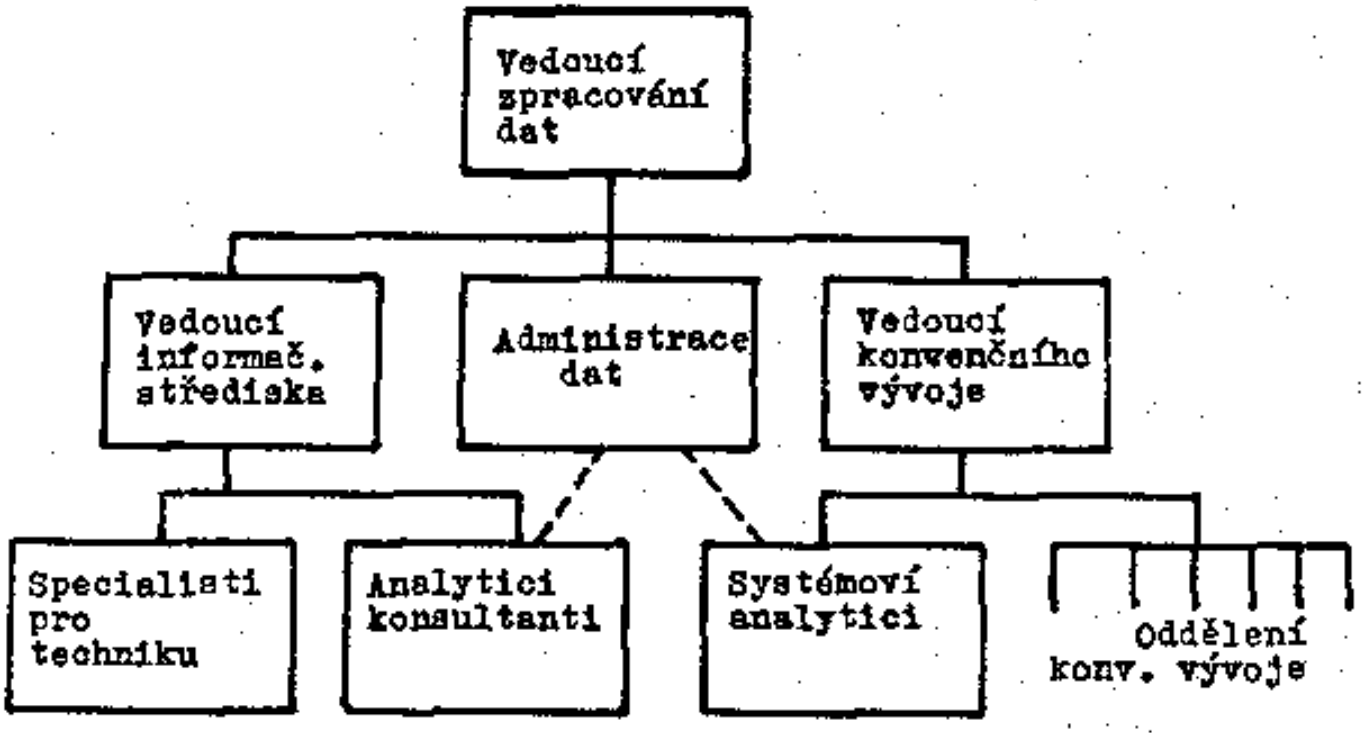
Mezi analytiky vznikne nová specializace - analytik pro techniku. Do jeho náplně bude spadat výběr nejvhodnějšího podpůrného software, jeho mistrné zvládnutí, jednání se softwarovými firmami, sledování využití systémů a plánování budoucích zdrojů, doladování a reorganizace aplikací za účelem lepší strojové průchodnosti atd.

A jaká budoucnost kyne nám programátorům? Zdá se, že dojde k určité diferenciaci. Na jedné straně stoupne zájem o zručné programátory v assembleru pro vývoj různých softwarových nástrojů čtvrté generace a na druhé straně část programátorů posílí analytiky, kde zvláště výše zmínění analytici pro techniku se budou určitě získávat hlavně z řad bývalých programátorů. Časem by měl

poklesnout zájem o dnes nejrozšířenější a nejžádanější programátory v COBOLu a PL/I.

Je třeba si uvědomit, že přechod na novou technologii nelze uskutečnit naráz. Vždyť změna mezi software druhé a třetí generace trvala přibližně 10 let. Pro přechodnou dobu navrhuje Martin organizační strukturu podle následujícího obrázku.

Vývoj aplikací je rozdělen do dvou větví: na konvenční vývoj a t.zv. informační středisko. Obě mají společnou administraci dat, která hraje důležitou standardizační roli u dat, která se mezi oběmi oblastmi předávají. Personální obsazení závisí na



stupní využívání nových metod. Zpočátku malé informační středisko se může postupně rozšiřovat a infiltrovat nové metody vývoje aplikačních programů i do oddělení konvenčního vývoje.

5. Závěr

Příklady softwarových prostředků přibližujících aplikační programování koncovému uživateli ukazují na jednu z možných cest budoucího vývoje v oblasti využívání výpočetní techniky. Důležité je, že některé prostředky této úrovně nejsou ani pro nás vzdálenou utopií, ale jsou dostupné prakticky okamžitě.
Rozvoj a především účelné široké využívání takových softwarových prostředků vedou k naplnění myšlenky obsažené v názvu článku /1/. Současné problémy aplikace výpočetní techniky nelze vyřešit pouhým masovým programováním jednoduchých úloh na mikropočítačích. (Tento dojem může leckdo nabýt z popularizačních pořadů naší televize). Připomínáme slova A. Jeršova /1/: " ... gramotnost není jen schopnost číst, ale i výchova inteligentního člověka " a " ... druhá gramotnost není jen schopnost psát příkazy, ale současně i výchova rozhodného i předvídavého člověka ".

Literatura:

1. Jeršov A.P., Programování - druhá gramotnost, Pokroky matematiky, fyziky a astronomie 27, 325-335 (1982)
2. Martin J., Applications Development without Programmers, Prentice - Hall, 1982
3. Dixon W.J. (editor), BMDP Statistical Software, Univ. of California Press, 1983
4. Nie N. et al, Statistical Package for Social Science, Mc Graw Hill, 1975
5. Sborník Programování 82, DT ČSVTS Ostrava, 1982
6. DATATRIEVE - 11 V2.0, Users´Guide, DEC, 1980
7. Novák V., Zdebski S., Čeština mezi člověkem a počítačem, in sborník Programování 83, DT ČSVTS Ostrava, 1983
8. Dotazovací systém DTS, Datasystém Bratislava, 1982