Z K U Š E N O S T I Z T V O R B Y P R O J E K T U P R A C U J Í C Í H O S D A T A B A N K O U

Ing. Bohumil Vondráček

V poslední době jsme svědky rostoucího pronikání výpočetní techniky téměř do všech oborů lidské činnosti. S tím samozřejmě roste na jedné straně potřeba rychlého produkování a zavádění nových projektů do praxe, na druhé straně pak potřeba zajištění efektivní údržby a přenositelnosti projektů již využívaných. Proto byly a neustále jsou hledány metody a prostředky (jak technické, tak organizační), které zahrnují téměř celou projekční činnost a které by ve svých důsledcích měly vést k žádoucímu zefektivnění projekčních a programátorských prací. Ve svém příspěvku bych se chtěl zabývat (na příkladu konkrétního projektu) některými programátorskými prostředky, které vedou k zajištění žádoucích výsledků.

1. Popis výchozí situace

Jako výpočetnímu středisku servisního typu nám byl k naprogramování a následně k rutinnímu provozování předán zadavatelem hotový technický projekt obsahující analytické řešení zpracování dovozního obchodního případu jedné organizace zahraničního obchodu. Šlo o dálkově orientovaný projekt z oblasti hromadného zpracování dat. Dokumentace technického projektu (rozsah cca 800 stran) byla sice napsána dle určité osnovy, byla však nesourodá, nesrozumitelná a přestože obsahovala všechny předepsané kapitoly, nebylo možné na základě informací v ní obsažených zahájit programátorské práce.

Technický projekt předpokládal využití banky dat DMS/90 (vychází z doporučení Codasyl a je velmi podobná IDMS) pro zajištění přístupu k datové základně. Pro představu o rozsahu navržené schema obsahovalo 53 typů vět a 42 setů. Datová základna dle předpokladů analytiků měla zabírat cca 150MB diskových kapacit.

Pro implementaci projektu byl k dispozici počítač Univac 90/60 (pod operačním systémem VS/9) vybavený reálnou pamětí 1MB. Uvedený počítač a jeho virtuální operační systém umožňuje

komfortní interaktivní přípravu programátorských produktů. Vzhledem k okolnosti, že uvedený počítač dožívá, bylo by nutno předpokládat, že projekt bude nutno v průběhu rutinního provozu převést pod dosud neznámý počítač a operační prostředí. Termín zavedení do rutinního provozu byl, jak už to obvykle bývá, rovněž krátký. Navíc zadavatelé uplatňovali požadavek budoucího snadného doplnění interaktivních funkcí namísto některých dosavadních dodávkově zpracovávaných transakcí.

Pracovní tým programátorů, který byl k dispozici, byl složen z pracovníků, kteří kromě školení neměli žádné zkušenosti s databankovým zpracováním, měli však bohaté praktické zkušenosti z využívání mětodiky Jacksonova strukturovaného programování a pochopitelně znali zvolené programovací jazyky, operační systém a standardy našeho VS.

2. Cíle autorů prováděcího projektu

Jedním z prvních problémů, které bylo nutno řešit, byl problém srozumitelnosti zadání. Shodou okolností bylo technické řešení vypracováno pracovníky jiného útvaru našeho podniku a podařilo se nám dosáhnout zařazení 3 původních autorů technického projektu (ne hlavních) do týmu řešící prováděcí projekt. Prvním cílem bylo dosáhnout přepisu zadání do srozumitelné pokud možno formalizované formy a tím zajistit jasné informační rozhraní mezi analytiky a programátory.

Některé další cíle byly již programátorsky orientované:

a) zajištění programátorské disciplíny tak, aby programové produkty jak ve fázi návrhu programových struktur, tak při vlastním kódování byly formálně jednotné, tj. nebyly obrazem práce konkrétního programátora,
b) produkování programové dokumentace průběžně (co není dokumentováno je neznámé) a pokud možno automaticky na počítači,
c) urychlení prací na prováděcím projektu a zajištění spolehlivosti a snadné údržby schopnosti výsledných produktů zavedením generátoru zdrojového textu,
d) zajištění přenositelnosti, tj. nezávislosti hotových produktů na operačním systému a disponibilním databankovém systému,
e) oddělení částí zpracování transakcí tak, aby bylo možno v

budoucnu některé komponenty beze změn převzít v případě doplňování interaktivních transakcí,

f) zajištění možnosti programování a testování transakcí bez ohledu na jejich návaznosti.

V dalším textu budou uvedeny některé metody a prostředky, které byly použity k realizaci výše uvedených cílů.

3. Generátor zdrojových textů

Těžištěm projektu bylo zpracování cca 30-ti vstupních aktualizačních transakcí. Na základě rozboru požadavků analytiků na zpracování těchto transakcí byly vytipovány dvě skupiny:

- transakce obsahující pouze jednu vstupní aktualizační větu,
- transakce obsahující jednu řídící vstupní větu, za níž následuje sekvence skupin dalších vstupních aktualizačních vět.

Pokud původní analytická představa byla jiná, bylo možné po nepatrné úpravě struktury a formátu vstupních vět transakce převést na jeden z výše uvedených typů. Výjimku tvořily transakce, které později nebyly jako celek generovány. Uvedeným typům transakcí pak odpovídaly příslušné řídící programové struktury, které však bylo nutno dle požadavků zpracování konkrétní transakce modifikovat.

Výsledkem provedeného rozboru bylo tedy nalezení parametrů potřebných pro modifikaci struktur dle požadavků zpracovávaných transakcí. Parametry byly zobecněny, byla jim přiřazena jména, syntaxe a sémantika a v podstatě daly vznik problémově orientovanému neprocedurálnímu jazyku, který na jedné straně dovoloval analytikům týmu specifikovat formalizovaným způsobem konkrétní funkce příslušné vstupní transakce a zdokumentovat je, na druhé straně umožnil automatickou generaci zdrojového textu.

V rámci parametrů se vyskytovaly parametry identifikační, parametry pomocí nichž byla modifikována řídící struktura modulů, parametry definující prováděné logické kontroly a parametry přístupu k databance. Syntaxe a význam jednotlivých parametrů byl dokumentován v uživatelské dokumentaci generátoru a je nedílnou součástí dokumentace projektu.

Problém nastal v okamžiku volby implementačního jazyka pro

zápis tohoto generátoru. Musel to být takový jazyk, který umožňuje pohodlnou práci s řetězci znaků tak, aby náklady vynaložené na generátor byly poměrně nízké a práce proběhly v rozumně krátkém časovém intervalu. Vzhledem ke skutečnosti, že velký výběr nebyl, byl zvolen jako implementační jazyk ediční interpretační prostředek umožňující zápis procedur. Problémem do budouona však je, že podobný prostředek patrně nebude k dispozici při obměně počítače. V případě přechodu na jiný počítač bude buď nutno tento software přeprogramovat, nebo se spokojit s vygenerovanými texty v cílovém programovacím jazyce a další vývoj generovaných produktů provádět na modulech v cílovém jazyku.

Při návrhu programových struktur generovaných preprocesorem byla použita výlučně metoda Jacksonova strukturovaného programování. Generátor zajišťoval generaci dvou oddělených modulů pro zpracování jedné transakce: modulu logických kontrol a výkonného modulu. V modulu log. kontrol byla vstupní data kontrolována dle specifikovaných parametrů a bylo přitom prováděno pouze čtení datové báze. Generovaný modul log. kontrol zajišťoval správný přístup do datové báze a v případě vzniklé chyby vstupu byly generovány odpovídající formáty vět protokolu o provedených resp. chybných transakcích. Formáty chybových zpráv byly parametrizovány tak, aby při shodných chybách ze dvou různých typů transakcí byl výsledek formálně shodný. Přitom byl používán soubor obsahující parametrizované texty chyb. Tento soubor chybových zpráv je vlastně již součástí uživatelské dokumentace. Při procesu hlášení chyb byl rovněž využíván slovník dat, obsahující mimo jiné též ke každé zkratce jména údaje náhradní uživatelský text. Texty chybových zpráv byly voleny o proměnné délce tak, aby byly koncovým uživatelům co nejsrozumitelnější.

Výkonný modul byl vyvolán v případě, že transakce byla jako celek správná, využíval currency dosažené v modulu log. kontrol a zajišťoval požadovanou aktualizaci datové báze. Byl invertovanou korutinou procesu log. kontrol. Při transakcích obsahujících více než jednu vstupní aktualizační větu byl využíván pracovní mezisoubor jako jednotka pro předání dat mezi oběma procesy.

Rozdělení zpracování transakce do dvou modulů bylo provedeno mimo jiné i proto, aby při předpokládaném doplňování interaktiv-

ních funkcí nebylo nutné výkonné moduly přeprogramovávat. Navíc je možné v budoucnu rozšířit parametry o popis obrazovkových formátů a zajistit analogicky generování modulu logických kontrol i pro interaktivně zpracovávané transakce.

Generátor zdrojových textů generuje v průměru na cca 60 řádek parametrů dva moduly každý s cca 1200 řádek. V hlavní řídící struktuře je automaticky obsloužena celá řada funkcí, o kterých analytik ani řádový programátor nemusí prakticky nic vědět (např. sbírání a výpis statistických informací, zpracování fatálních chyb a pod.).

Výše uváděný průzkum analytických potřeb na parametry generátoru záměrně nezobecnil některé specifické potřeby transakcí. Šlo o funkce, které se vyskytovaly pouze při zpracování jediné transakce a tudíž zobecňovat je a zahrnout je do parametrů a následně dle nich generovat výstupní text by bylo luxusem, který stojí jen čas a náklady při tvorbě generátoru. Pro tyto potřeby byla zvolena jiná strategie. Na vybraná místa generované struktury bylo možno doplňovat části zdrojového textu přímo v cílovém programovacím jazyku. Toto řešení se zdá být nedůsledností, ale z praktických důvodů zcela opodstatněné. V tomto případě se mezi parametry dokumentující vstup pro generátor vyskytl příznak manuálního řešení, který byl dokumentován komentářem s jasným zápisem o jakou funkci jde a byl programátory separátně naprogramován s označením místa, kam mají být příslušné řádky včleněny. Ruční včleňování řádek do vygenerovaného textu není vhodné, neboť by bylo nutné při změnách provádět tuto činnost opakovaně, rovněž tak navržený způsob zajišťuje neporušení generovaného textu. Tím, že existovaly ručně programované části, zbavili jsme se možnosti generovat text v různých jazykových mutacích. Tato ztráta možnosti není zanedbatelná v případě použití cílového jazyka, u nějž přeložený modul je podstatně kratší.

Tvorba generátoru při použití výše zmíněného edičního prostředku si vyžádala 3 člověkoměsíce na specifikaci zadání, 2 člověkoměsíce při vlastním kódování. Zdá se, že se tato práce vyplatila, neboť tímto způsobem vzniklo a bylo otestováno 46 modulů s průměrným počtem 1648 zdrojových řádek v COBOLu v

průběhu 2 měsíců. Pro tyto moduly bylo zapsáno 1768 zdrojových řádek parametrů a 10707 ručně vkládaných řádek v Cobolu, přičemž celkový počet řádek činil 75803.

Vygenerovaný text je bezpečně správný, odpovídá představám dobře strukturovaného čitelného programu, obsahuje všechny standardem předepsané komponenty a je dokumentován jak popisem parametrů, tak vygenerovanými strukturními diagramy (viz dále). Každý modul je opatřen generovaným rámem (viz dále). Dokumentace modulu vzniká průběžně a je automaticky při změnách aktualizována.

Generátor vznikl jako jednoúčelový pouze pro potřeby tohoto projektu, avšak lze ho do budoucna používat i pro jiné projekty, pokud se analytické požadavky přizpůsobí jeho filosofii. Navíc je možné do něj další funkce doplňovat. Opírá se však o zcela konkrétní, částečně modifikovatelné řídící struktury. Proto se nabízí otázka, zda by nebylo vhodné koncipovat generátory tohoto typu poněkud jinak. Mít totiž k dispozici prostředek, který dostane jako jeden vstup formalizovaně popsanou řídící strukturu, jako druhý vstup formáty parametrů a jako třetí vstup vztah parametrů k řídící struktuře. Je samozřejmé, že některé řídící struktury by bylo možné standardizovat a následně je jako části nebo celku využívat k definici prvního vstupu. Otázkou zůstává, zda pokud je toto schůdná cesta, by se neměli řešením zabývat softwarové firmy.

Domnívám se, že podobně jako generátory normovaného programování či rozhodovacích tabulek sloužící pro generaci jisté třídy úloh lze, pokud se to vyplatí (náklady) připravovat podobné jednoúčelové generátory. Je zajímavé, že řadoví pracovníci, kteří měli tento prostředek používat, nepřijali tento způsob práce ze začátku ochotně.

4. Prostředek pro zajištění nezávislosti produktů na konkrétním databankovém systému

Jednou ze snah autorů prováděcího projektu byl pokus o zajištění nezávislosti programových modulů na konkrétním databankovém prostředí a tím zajištění přenositelnosti produktů v tomto smyslu. Obecně se dá říci, že zajištění přístupu programu k datové bázi je implementační problém, neboť např. je-li vydána

programem funkce čti větu, nemusí program nic vědět o způsobu uložení a získání této věty. Programu se tento způsob práce dotýká pouze v tom, že jeho struktura musí zohledňovat i zpracování výjimečných stavů (jako věta nenalezena, stav eof). To znamená, že problém styku s daty lze redukovat až do hierarchické vrstvy. Často se v této souvislosti hovoří o abstraktním datovém typu, přičemž jsou k dispozici funkce s jasným interface nad tímto datovým typem. V případě systému pro zajištění přístupu k datům půjde o to, aby funkce definované nad jednotlivými typy dat měly nezměněný interface hierarchicky směrem vzhůru, směrem dolů budou mít interface různý podle konkrétního disponibilního databankového systému (přeprogramují se). Naší snahou bylo, aby ani funkce na poslední hierarchické úrovni nemusely být při změně databankového systému měněny.

Představa je taková, že je programována nová databanka (naše) a existuje interface modul, přes který se stýkají programové moduly s příslušným databankovým systémem. Tento interface modul musí směrem vzhůru zajistit shodné chování, ať už jsou skutečná data ukládána za použití libovolného databankového systému, nebo v extrému za použití klasických organizací. Interface modul je tedy jediným produktem, který se v případě změny fyzického uložení dat mění.

Byla tedy pro potřeby projektu specifikována nová databanka, která měla řadu funkcí shodných s klasickou databankou dle doporučení Codasyl. Počet funkcí však byl omezen na nezbytné minimum. Přístup k tomuto modulu byl zajišťován pomocí CALL interface.

Dalším záměrem bylo zjednodušení řídících struktur programových modulů, který interface modul využívají a tím snížení jejich složitosti a zajištění lepší přehlednosti a čitelnosti. Používání databanky totiž vyžaduje po každé DML operaci testovat, zda došlo k předpokládanému stavu nebo k chybě. Navržený systém naopak vyžadoval od programátorů předem specifikovat předpokládané výjimečné stavy, vše ostatní bylo považováno za chybu a zpracování těchto chyb bylo řešeno centrálně v interface modulu. Tento způsob zpracování výjimečných stavů pak zajišťuje snadnou kontrolu, zda jsou všechny předpokládané výjimečné stavy v programových

strukturách náležitě ošetřeny a ve svých důsledcích to znamená zvýšení spolehlivosti výsledných programových modulů.

Rovněž tak některé informace poskytované databankovými systémy automaticky, musel programátor při použití tohoto systému explicitně požadovat (jde např. o DBKEY, RECORD-NAME a pod.), což má jednak metodický důvod, jednak umožňuje získat centrálně přehled o využívání těchto informací.

Uvedená koncepce použití interface modulu umožňuje rovněž snížit spotřebu strojového času pro přípravu prováděcího projektu. Je totiž odstraněna potřeba DML preprocesoru, který je jinak nutno provádět před vlastní kompilací. Odstraněním tohoto preprocesoru jsme se však zbavili možnosti odhalit některé chyby oproti kompilátoru cílového jazyka. Dosavadní praxí se však potvrdilo, že k chybám tohoto charakteru prakticky nedochází. Přídavná spotřeba strojového času v interface modulu závisí na tom, jaký databankový systém je pro ukládání dat použit. V případě podobnosti s naší abstraktní databankou je tento čas zanedbatelný.

V průběhu zpracování prováděcího projektu dochází obvykle ke změnám schematu, zejména formátu vět (doplňování některých informací). Při přímém používání databankového systému je s těmito změnami někdy třeba provést novou kompilaci schématu a všech subschémat obsahující měněnou větu. Vypracovaný interface modul realizoval tyto změny bez potřeby následné rekompilace schématu v případě, že se nemění délka věty (tj. je počítáno s rezervou) a kdy se nemění pozice a délka klíčových údajů (CALC klíč a klíč setříděného setu).

Realizace myšlenky oddělení vlastního databankového systému a uživatelských modulů a zabezpečení komunikace těchto modulů přes zprostředkující interface modul má řadu dalších perspektivních výhod. Lze totiž v průběhu rutinního provozu upravit tento modul tak, aby např. získával a vypisoval různé statistické informace o práci s databankou (rychlost, četnost přístupů, typy požadovaných funkcí atp.), pořizoval centrální testovací výpisy, simuloval zápisové funkce, které mění databanku a pod., a to bez potřeby úprav programových modulů, které jej využívají a bez jejich následné kompilace. To je možné právě proto, že veškeré

funkce přístupu k datům jsou realizovány přes jediný modul. Je třeba poznamenat, že některé databankové systémy mají výše uvedené možnosti již přímo zabudovány.

Změna interface modulu k zajištění výše uvedených funkcí se v rutinním provozu může dít toliko výměnou objekt modulu tohoto interface modulu na knihovně. Je třeba pouze zajistit, aby byl interface modul přisestavován dynamicky při spuštění programu.

V našem VS byl vyzkoušen převod programu pod systém EC 1033 a databankový systém IDMS. Díky standardu používání omezeného repertoiru Cobolských příkazů a klausulí uplatňovaném v rámci tvorby projektu stačila pouhá rekompilace modulů, výměna interface modulu a pochopitelně nagenerování databanky. Tento test dopadl velmi úspěšně.

5. Programová dokumentace

Snahou vedení týmu bylo, aby programová dokumentace vznikala průběžně, tedy v okamžiku otestování produktu byla konečná. Dále, aby byla dokumentace získávána z předaných produktů. Např. pro zdrojové moduly sestávala dokumentace z rámu modulu, strukturních diagramů a v případě generovaných modulů ještě z opisu parametrů. Rám modulu a opis parametrů pro generátor byl tištěn v záhlaví modulu formou komentářů.

5.1 Rám modulu

Tento pojem označoval část dokumentace udržovanou ve zdrojovém modulu formou komentářů. Sloužil k jednoznačnému popisu funkce příslušného modulu (ale i copy/macro modulu, JCS procedury a pod.). Povinností každého člena týmu bylo předávaný modul opatřit rámem, aby další člen týmu, který měl modul používat se z něho dozvěděl všechny potřebné informace. Důsledné dodržování tohoto předpisu pak vedlo ke snížení času potřebného pro komunikaci mezi členy týmu.

Členění rámu bylo následující:
- jméno modulu, verze
- autoři: analytik, programátor
- funkce modulu
- volání modulu s uvedením parametrů

- popis vstupních parametrů
- popis výstupních parametrů
- popis pracovních parametrů
- předpoklady nutné k vyvolání modulu
- vedlejší efekty zpracování
- výjimečné stavy
- poznámky
- volané podprogramy
- použité copy/macro moduly
- změnový seznam (verze, autor, datum změny, popis změny).

Při použití rámu v copy/macro modulech a JCS procedurách byl jeho obsah redukován. Moduly vzniklé jako produkt generátoru byly opatřovány rámem automatizovaně.

5.2 Dokumentace struktury modulu

Pro potřeby zajištění vytváření strukturních diagramů modulů byl vyvinut podpůrný prostředek, který na počítači grafickým způsobem znázornil strukturní diagram modulu (jen pro moduly v Cobolu) včetně podmínek v iteracích a selekcích a přiřazených operací (operace pouze odkazem na příslušné řádky zdrojového textu). Pomocí tohoto prostředku byl každý modul před předáním dokumentován a spolu s rámem a vlastním zdrojovým textem tvořil jedinou a postačující dokumentaci modulu.

Podpůrný prostředek zabezpečující generaci strukturního diagramu mohl pracovat v režimu, kdy prováděl pouze formální kontrolu správnosti kódu, takže byl též používán pro automatickou kontrolu dodržování standardu kódování. Kontrolu prováděl každý pracovník týmu sám, neboť jeho povinností při předání hotového produktu bylo odevzdat nejenom hotový modul, ale též i dokumentaci jeho struktury. Byl to tedy jeden z prostředků k zabezpečení dodržování programátorské disciplíny.

5.3. Zásady projektu

Šlo o písemný materiál vypracovaný před skutečným zahájením prací na prováděcím projektu, ve kterém byly prezentovány vydané standardy platné pro celý projekt, sloužil jako uživatelská

dokumentace všech podpůrných prostředků (byly zde uváděny i příklady použití), dokumentoval předepsané pracovní postupy a obsahoval globální informace o strukturách programů (tok dat v programech, okolí programů) tak, aby pracovník, který pracoval na podřízených modulech,mohl získat informace o okolí vytvářených modulů. Do materiálu byly soustředěny všechny globálně platné informace tak, aby po přečtení materiálu mohl pracovník bez další potřeby vysvětlování pracovat.

Materiál byl udržován na počítači pomocí text editoru a byl průběžně aktualisován. Změny byly v materiálu zvláště zvýrazněny šipkami na levém okraji řádku, aby při změně sersa se pracovníci nemuseli zabývat studiem celého materiálu.

5.4 Přehled programů a modulů

Šlo o písemný materiál, který dokumentoval stav rozpracovanosti programů a modulů. Pro každý modul zde bylo sledováno: odpovědný analytik - termín zahájení, předpokl. termín ukončení, skutečný termín ukončení; odpovědný programátor - termíny; programovací jazyk. počet řádek modulu; příznak použití generátoru - počet řádek parametrů, počet řádek ručních vstupů; zkrácený popis funkce.

Tyto informace byly používány jako podklad pro řízení projektu, ale rovněž jako podklad pro odměňování. Materiál byl udržován na počítači.

6. Testovací prostředky

Interaktivní možnosti poskytované firemním systémem byly postačující (interaktivní testování s možností symbolických odkazů).

Byl však vyvinut prostředek sloužící k interaktivnímu plnění a opravám vět databanky a k pořizování formátových výpisů obsahu databanky. Tento prostředek byl nezbytný pro zajištění nezávislosti testování jednotlivých transakcí, tj. členové týmu nemuseli čekat s testováním na okamžik předání programových modulů zabezpečujících aktualizaci vět databanky, se kterými sami pracovali.

Byla vyvinuta ještě dávková forma tohoto prostředku (pro

potřeby testování systému jako celku) umožňující zadání přístupové cesty provést formátový výpis vět a jejich vazeb. Firemní software nenabízel žádnou uspokojivou možnost tyto aktualizace a výpisy databanky provádět.

7. Závěry

V příspěvku jsem se zaměřil na prezentaci některých prostředků a metod, které vedly k zvýšení produktivity programátorských prací a při nichž byl používán počítač jako pomocný prostředek. Domnívám se totiž, že jedna z možností zvýšit efektivnost projekčních a programátorských prací je v automatizování těchto činností. Pomocí námi vypracovaných projektů se snažíme racionalizovat činnosti koncových uživatelů, aniž sami dosud málo využíváme výpočetní techniku k racionalizaci vlastních činností. Je však třeba si uvědomit, že zapojení počítače k automatizování projekční činnosti vyžaduje zavedení a důsledné dodržování standardizace.

Příloha: Příklad vstupních parametrů pro generátor zdrojového textu

```
     LKMNME=KMD43
     VKSNME=KMD44
     VER=000,25/9/84,MACKOVA
     TRANS=DD120,AKTUALIZACE DB VETY IKTG
     DBRECW=IKTG-304,OP-306,JCENY-317,ZKM-300
     ANME=KUBKOVA
     PRME=MACKOVA
     LKNUM=DV-DD120,IPKTG-DD120
     LKPOVI=UZIVA-DD120
     KODOP=10,VYTVORENI NOVE VETY IKTG
(1)     LKPOVI=KIKTG-DD120
        LKPOVI=NIKTG-DD120
        LKCALC2=IKTG-304,KIKTG-DD120
(4)     LKCALC1=CHYBE03,ZKM-300,ZMOBA-DD120
        VKSTORE1=IKTG-304
     KODOP=20,ZMENA POLI DATOVE CASTI VETY IKTG
        LKPOVI=(1)
        LKPOV3=NIKTG-DD120
```

```
(2)   LKCALl=CHYBE05,IKTG-304,KIKTB-DD120
      LKCALC1=(4)
      VKMODIFY1=IKTG-304
   KODOP=80,ZRUSENI VETY IKTG
      LKPOVI=(1)
(3)   LKPOV2=NIKTG-DD120
      LKCALl=(2)
      LKSETl=IKT-OPE
      VKERASEl=IKTG-304
   KODOP=92,VYPIS VSECH IKTG
      LKPOV2=KIKTG-DD120
      LKPOV2=(3)
      VKFETCH2=IKTG-304,AZAHRPA
```

Uvedený příklad dokumentuje parametry pro zpracování nejjednodušší transakce projektu nevyžadující v tomto případě zápis ručně vkládaných programových částí. Na základě uvedených parametrů byly vygenerovány dva moduly (modul log. kontrol - 1092 řádek, výkonný modul - 692 řádek).

Pro porozumění jednotlivým parametrům uvádím orientačně jejich významu parametry PRGNME až PNME jsou identifikačními parametry definující jména programu, modulu, log. kontrol, výkonného modulu, transakce, analytika, programátora, použité DB věty; parametry KODOP uvádějí kódy operací; parametry LKxxx resp. VKxxx definují požadované log. kontroly resp. výkonné akce, je-li transakce správná; pro úsporu zápisu bylo možné použít odkazy na návěští s významem zopakuj akci definovanou u tohoto návěští.