P R O V O Z N Í S P O L E H L I V O S T A P L I K A Č N Í H O S O F T W A R E

Ing. Dušan Streit

Provozní spolehlivost aplikačního software bývá jedním z nejvíce podceňovaných aspektů při budování ASŘ. Přitom právě tato spolehlivost a odolnost programového zabezpečení je limitujícím činitelem při provozování ASŘ a podstatnou měrou se podílí na jejich efektivitě. Svědčí o úrovni programového vybavení a pozornosti, jaká je jeho tvorbě věnována. Jedná se o komplexní pojem, který v rámci systému zpracování dat musí být zajištěn soustavou standardů; sestává z těchto komponentů:

- správnost funkcí (programů, úloh, subsystémů)
- provozní odolnost (proti chybným datům, nesprávné obsluze, výjimečným stavům, haváriím)
- efektivnost řešení (provozní i uživatelská)
- jednoduchost a jednoznačnost obsluhy (operátorský komfort, provozní dokumentace)
- zabezpečení a ochrana dat a programů
- přístupnost změnám
- přenositelnost a typovost řešení.

V příspěvku bude toto téma rozebráno vzhledem ke zpracování hromadných dat, a to s akcentem na programové zabezpečení. Pozornost však bude obrácena ne k programu, ale jejich soustavě (úloze, subsystému), se zvláštním důrazem na odolnost software. V jednotlivých kapitolách budou rozvedeny prostředky, kterými je možno pozitivně provozní spolehlivost ovlivnit.

1. Systémový přístup - standardizace

Zde bych použil negativní argumentace; nastíním situaci, která je protikladem systémového přístupu:
Úlohy se řeší agendově, izolovaně a bez integračních vazeb. Každý řešitel se stará pouze o svůj problém. V něm se dobře orientuje, a proto nevytváří žádnou projektovou dokumentaci. Sám se účastní programování a hlavním kriteriem v této oblasti je pro něj termín ukončení. Proto se nastaví o metodiku programování a uplatňování typových prvků - ostatně nic z toho v takových podmínkách se ani

nerozvíjí. Do provozu se "předávají" polotovary bez řádné dokumentace. Bez osobní účasti autorů nelze agendu zpracovat, stále se něco dolaďuje a řeší jako provizorium. Každé zpracování je "nervák" a autoři jsou trvale vázáni agendou. Dochází ke skluzům zpracování a to právě tehdy, když je termín nejžhavější. V nejméně očekávaném okamžiku se ve výsledcích objevují "záhadné" chyby. Agenda je odsouzena k zániku také tím, že ji nelze přizpůsobovat objektivní realitě a pokud se ještě zpracovává, nikdo ji nepotřebuje. Trpí lidé i naše lesy.

Tato situace je dosti častá, i když z hlediska řešitelů záměrně nadsazené (uživatelé by však tu nadsázku už tak nepociťovali). Konkrétní příčiny tohoto stavu mohou být různé, v zásadě však existuje jediná základní příčina - špatná organizace práce (z hlediska vedoucích) doprovázená nedostatečnou pracovní kázní (z hlediska podřízených). V této obecné rovině existuje také jediný "lék" - systémový přístup. Zastřešuje také veškeré prostředky uvedené v dalších kapitolách, avšak v kontextu této kapitoly je třeba zdůraznit tyto předpoklady:

a) V souladu s metodickými pokyny pro budování ASŘ je třeba vytvořit a soustavně rozvíjet <u>komplex standardů</u>, které systémový přístup uvádějí do praxe v konkrétních podmínkách. Standardy musí zabezpečovat celý systém od analýzy až po provoz.

b) <u>Organizace</u> musí být upravena tak, aby byli určeni pracovníci oprávnění standardy vydávat (nejlépe systémoví programátoři). Na druhé straně závazné dodržování těchto standardů se musí stát součástí pracovní kázně.

c) <u>Strategické cíle</u> musí v každém okamžiku stát nad dočasnými, dílčími a vedlejšími cíli. Blížící se termín se nesmí stát zaklínadlem pro obcházení zásad. "Ušetřený" čas nám potom pochybí v nejméně vhodném okamžiku.

d) Na časové ose analýza - provoz platí, že při <u>investicích řešitelských kapacit</u> jsou tyto investice tím účinnější, čím více jsou směřovány do včasnějších etap řešení. Platí zde zásady: "nejdřív zasej, pak můžeš sklízet", "jak si usteleš, tak si lehneš", "dvakrát měř, jednou řež".

2. Dekompozice systému zpracování dat - parametrické programy

Zde bych chtěl obrátit pozornost na specializaci v programovém zabezpečení, a to specializaci technologickou, nikoliv agendovou. Na základě této specializace a strukturalizace lze zobecnit a v konečném důsledku parametrizovat obdobné problémy. Takováto řešení se mohou stát ověřenými prefabrikáty pro konkrétní aplikace a je možno se o ně opírat počínaje analýzou. Navíc se tak omezí řešení nesourodých činností v jednom programu, které byly častou příčinou chyb při programování.

Z analytického hlediska to znamená především oddělení vstupů a výstupů od algoritmického zpracování, rovněž některé manipulace s daty a informacemi (např. sdružování a výběry) lze řešit jako obecný problém. Jako prostředek mohou být využity parametrické programy. Podrobněji k této problematice je v teoretické rovině pojednáno v literatuře /1/, příklady praktických aplikací jsou dokumentovány v literatuře /2/ a /3/. Hrubé blokové schéma technologické dekompozice systému zpracování dat je znázorněno v obr. 1.

3. Projektová dokumentace - předprojektová příprava

Dokumentace má zásadní a v praxi nedoceněný význam v celém procesu budování ASŘ. V kontextu tohoto příspěvku mám na mysli především její pracovní poslání, které kladu nad archivní účely. Její úloha je dostatečně zakotvena v metodických pokynech, proto chci pouze připomenout:

Technický projekt je vyvrcholením práce v etapě systémové analýzy a slouží jako podklad pro programování; subsystém nebo úlohu nelze považovat za připravenu k programování, pokud není zodpovědně zpracován a dokončen. Provádějící projekt, jak známo, je možno chápat jako souhrn programové a provozní dokumentace plus uživatelská příručka; vytváří podmínky pro bezkonfliktní předání úlohy do užívání.

Přestože metodické pokyny vymezují závazně obsah těchto dokumentů, vyplácí se blíže specifikovat jejich náležitosti, formu a interní zásady jejich tvorby, nejlépe v technickém projektu systému. Navíc bych chtěl na základě praktických zkušeností doporučit, aby ve stadiu předprojektové přípravy se jako úvodní zadání znovu

používal ideový projekt. Měl by být stručným zhodnocením analýzy současného stavu a formulací úlohy vzhledem k její automatizaci a pouhým ideovým nástinem řešení. Je nedocenitelným prostředkem pro komunikaci s uživatelem, protože technický projekt je chápán spíše ve vztahu k programovému řešení a přesahuje možnosti uživatele se v něm orientovat. Můžeme však od uživatele vyžadovat oponenturu k ideovým projektům, přizvat jej tak ke spolupráci a z toho v dalším řešení těžit.

K základním metodám řešení bych chtěl v této souvislosti vyslovit svůj osobní názor: V etapě analýzy současného stavu je třeba preferovat metodu zdola nahoru, v etapě syntézy nového systému je třeba přednostně postupovat metodou shora dolů.

4. Integrované datové základna - interaktivní přístup

Za základnu projektového řešení pokládám informační systém. Proto zmapování všech dat a informací, které jsou předmětem nebo výsledkem automatizace, považuji za východisko technického projektu. Jako povinné součást TP se v našem VS zpracovává tzv. tabulka vstupně výstupních vztahů, která je vlastně seznamem položek v této formě: označení položky - slovní název - zdroj - užití. Zdrojem položky může být vstupní soubor nebo algoritmus, užitím pak algoritmus, výstupní soubor nebo sestava. Označení položek, souborů, algoritmů a sestav v této tabulce je závazné pro využití v celém technickém projektu a měla by se přenášet až na úroveň jednotlivých programů (např. ve formě identifikátorů). Ideálem je jednotná báze dat s vyloučením multiplicit a redundancí. Nemusí to vždy znamenat databankové systémy, je možno zachovat souborovou organizaci doplněnou o tzv. systémový adresář dat a podporu parametrických programů. Praktický příklad aplikace je možno vidět např. v ČKD Praha (tzv. systém SAD) nebo v JZD Kelč (tzv. systém GIN), který umožňuje oddělení popisů datové báze od programů. O vztahu mezi integrovanou datovou základnou a možností interaktivního přístupu pojednává literatura /4/.

5. Rozhodovací tabulky - strukturované projektování

Vyčlenění datových vazeb do samostané kapitoly v TP vytváří

předpoklady pro efektivní uplatnění rozhodovacích tabulek v algoritmické části. Je možno se abstrahovat od zdroje a užití jednotlivých položek a zaměřit se na "čisté" algoritmy - podmínky a činnosti. V této souvislosti je třeba zdůraznit, že jakákoliv formalizace je lepší než volná forma TP ve stylu slohových cvičení. U rozhodovacích tabulek navíc ještě převládá funkční hledisko nad procedurálním (ve srovnání s vývojovými diagramy), což má velký význam pro jejich využití jakožto komunikačního prostředku. V kontextu tohoto příspěvku je podstatné, že poskytují nástroj pro plné vyjasnění problému a umožňují sledovat úplnost, kontradikci či redundanci pravidel a vyvarovat se tak zbytečných chyb už při analytickém řešení. Nehledě na to, že se dají optimalizovat a automatizovaně překládat a využít tak i při programovém řešení. V oblasti automatizovaného překladu však nemám praktické zkušenosti, mohu pouze odkázat na literaturu /5/ a upozornit na to, že algoritmy ve formě RT lze jakožto číselníky ukládat na soubory a interpretovat při exekuci programů; některé algoritmy tak lze aktualizovat nezávisle na překladu, jak je využíváno např. v PŘÍVT Praha v rezortu zemědělství.

Závěrem bych chtěl poukázat na to, že i rozhodovací tabulky by se měly aplikovat strukturalizovaně a hierarchicky, tedy tak, jak je známo např. z uplatnění HIPO diagramů nebo z modulárního programování. Principy strukturovaného programování by se měly využívat už v etapě analytického řešení, které umožní jejich prosazování na vyšší hierarchické úrovni; tak je možno hovořit o strukturovaném projektování.

6. Manipulace se sekvenčními soubory - tvorba vět

Nejfrekventovanější manipulací s daty je vedle třídění jejich výběrové sdružování. U sekvenčních (sériových) souborů to představuje jejich synchronizaci dle klíčů. Tento problém lze řešit na úrovni jednotlivých programů metodou normalizovaného programování. V souladu se strukturovaným řešením jej však také lze velmi efektivně řešit na úrovni manipulace s daty a filosofie práce se soubory. Na základě funkcí syntézy (sdružování) a selekce (výběrů) je možno předem připravit soubor, který obsahuje všechny potřebné údaje pro další zpracování. Znamená to synchronizaci periodických

a matričních souborů tak, aby před vlastním zpracováním úlohy byl vytvořen integrovaný soubor, v jehož větě jsou sdruženy všechny potřebné položky bez ohledu na jejich zdroj. Na základě těchto zásad vstupuje do úlohy každý matriční soubor pouze jedenkrát a najednou předá všechna využitelná data. Při vlastním zpracování pak lze údaje čerpat pouze z jednoho souboru, což zjednodušuje programové řešení.

Tento problém byl obecně vyřešen jako parametrický systém "selektivní syntéza", jak je popsán v literatuře /2/ a implementován v n. p. Slezan F-M na počítači ODRA 1305 a v Kooperačním sdružení ASŘ a VT zemědělských organizací okr. F-M na počítačích SM-3/20 a SM-4/20.

Tato filosofie vychází z návrhu integrované věty. I když se tato věta postupně "obaluje" dalšími daty, její jednotná forma by měla prostupovat celým zpracováním úlohy. Integrovaná věta by měla mít pevnou délku a v rámci úlohy i jednotný popis tak, aby při zkušebním zpracování nebo výjimečných stavech bylo možno programy a soubory různě kombinovat.

V této spojitosti bych chtěl upozornit na efektivní systém třídění, kdy vstupní soubor existuje v bázi pouze jedenkrát a kolik je třídicích hledisek (parametrů třídění), tolik je vytvořeno k němu přístupových katalogů. Je to umožněno prostou indexací vět v souboru, v katalogu pak jsou uloženy sekvenční indexy (volitelně i s klíči), které odkazují na příslušnou větu v souboru v závislosti na daném setřídění. Šetří se tak místo na vnějších pamětech a čas třídění, protože se "nepřehazují" celé věty, ale pouze indexy. Takový systém je vytvořen a využíván v Kooperačním sdružení ASŘ a VT zemědělských organizací okr. F-M na systému SMEP.

7. Kontrolní chod - odolnost programů

Kontroly mají zásadní vliv na provozní spolehlivost úlohy, avšak úzce také souvisí s odolností programů proti chybným datům. V zásadě se dá říct, že to, co kontroly propustí, nesmí způsobit v dalším zpracování žádnou havárii ani výjimečný stav. Záruky analytika, že k určité konstelaci nemůže dojít, jsou bezpředmětné, pokud nejsou garantovány kontrolami, jinak si program musí ošetřit

vše sám.

Proto kontroly musí být řešeny komplexně už na úrovni technického projektu a nejlépe v samostatné kapitole. V souladu s typovou dekompozicí systému zpracování dat (viz obr. 1) je třeba kontroly zabezpečovat dvoustupňově:

- kontroly formální (syntaktické),
- kontroly logické (sémantické).

Formální kontroly se zabezpečují v rámci vstupů dat (konverzního chodu). Tyto kontroly zásadně chybné věty ze zpracování vylučují s možností víceúrovňových oprav, kdy opravy se vždy znovu kontrolují. Vyloučené věty mohou být v konverzních souborech pouze blokovány a fyzicky přístupné interaktivním opravám např. z terminálu. Při on-line pořizování vstupních dat je žádoucí zpracování alespoň formátových kontrol v reálném čase, takže systém upozorňuje na chyby hned při pořízení. Formální kontroly musí zabezpečovat:

- kontrolu numeričnosti,
- kontrolu rozsahu (rozmezí od - do),
- kontrolu přípustnosti (výslovné hodnoty z číselníku),
- kontrolu nenulovosti,
- kontrolu nezápornosti,
- kontrolu modulem (kontrolní znak),
- kontrolu součtovou (kontrolní číslo) apod.

Tyto kontrolní funkce mohou být řešeny kontrolními podprogramy.

Logické kontroly mají individuální charakter a je proto vhodné je řešit specifickými kontrolními programy nebo chody. Není však vhodné kontroly tříštit do jednotlivých programů úlohy. Vhodně navržený systém pro manipulace s daty umožní položky integrovat tak, aby se tyto kontroly prováděly komplexně. V zásadě se jedná o vazební kontroly, které mohou být např. relačního nebo logického charakteru s akcentem na položky z matričních souborů. Při návrhu je třeba analyzovat způsob reagence na chybné věty; existují tyto možnosti:

- chybné věty se vylučují ze zpracování; opravy se zabezpečují novým vstupem,
- chybné věty se ponechávají, nutno je však opravit buď přepisem chybných hodnot na principu editace nebo systémem opravných vět (např. dočítování ke mzdám),

- "chybné" věty se ponechávají, slouží pouze jako upozornění pro uživatele, že ve výstupních informacích může být rozpor (nesmí však způsobit chybnou práci v programu!),
- chybné věty se automaticky opravují předpokládanou nebo náhradní "default" hodnotou.

Default hodnoty mají překvapivě široké spektrum uplatnění. Při důkladné analýze jejich odvozování mohou ve více než 90 % případů odpovídat realitě. Jejich využití je efektivní především tam, kde by jinak bylo nutno zablokovat další zpracování a hodnoty jsou třeba pouze "kosmetického" charakteru. Využití náhradních hodnot by mělo být dokumentováno s ponechánim možnosti dodatečné opravy.

Důležitým momentem při analytickém zabezpečení kontrol je vyřešení opravných chodů. Znamená to, že v určitém bodě se musí zpracování přerušit a počkat na opravy (většinou od uživatelů) nebo zpracování provádět vícefázově. Obecně musí platit zásada, že po kontrolách a případných opravách chyb už nesmí do syatému zpracování dat vstupovat další data, která by už kontrolami neprocházela.

8. Správnost programů - strukturované programování

Správnost programu má zásadní význam na provozní spolehlivost úlohy. Dokazování této správnosti a teoretické nástroje jsou rozebrány např. v literatuře /6/. Program však není izolovaný prostředek, proto je nutno správnost programu chápat v kontextu celého zpracování. Chtěl bych se zde zaměřit na některé praktické nástroje s širšími důsledky.

Především se jedná o strukturované řešení - strukturované projektování a strukturované programování. Mám na mysli jeho širší pojetí než třeba schemata dle Jacksona či Myerse; v přehledu se tím rozumí:

- hierarchická dekompozice,
- přístup shora dolů (top - down approach),
- modularita (vstupy - černá skříňka - výstupy),
- zabezpečení elementárních struktur formalizovanými schematy (sekvence, testování, cyklus, volání subfunkce),

- formální přehlednost zdrojového textu (omezení GOTO, samovysvětlující identifikátory, COMMENT, rozsahová přehlednost modulů),
- dělba práce (týmová práce),
- dokumentace jako vedlejší produkt pracovních materiálů (RT, HIPO apod.).

V přehledu byly vyjmenovány pouze obecné nástroje. Skutečná správnost řešení však musí být zajišťována při konkrétním řešení každého jednotlivého programu (řetěz je tak silný, jak je silný jeho nejslabší článek). Nejsem optimistou v tom, že dokazování správnosti programu by proniklo do běžné praxe, přesto však je na místě určité "teoretizování" před tím, než přistoupíme ke kódování programu. I při využití vyšších programovacích jazyků a podpůrných ladicích prostředků tvrdím, že těžiště práce programátora a logiky programu tkví ve výstavbě vývojových diagramů, strukturních diagramů, blokových schemat, RT a v simulaci; kdo si tohle neuvědomí a honosí se tím, že tvoří programy přímo "ze vzduchu" není vůbec programátor, ale jenom kódovač a neví "o čem vůbec programování je".

Ke správnosti programů bych chtěl přispět přehledem nejčastějších elementárních chyb, na které je nutno se především zaměřit:

- přetečení indexu mimo rozsah tabulky (nutno kontrolovat!),
- zacyklení programu (UNTIL není možno stavět na vnějším předpokladu!),
- aritmetické, formátové přetečení (kontrola rozsahu!),
- dělení nulou (nutno testovat dělitele!),
- nedefinované počáteční hodnoty (nutno definovat nebo přiřadit,pozor hlavně při restartech!),
- čtení po nalezení konce souboru (je-li ve druhém souboru vyšší klíč),
- záznam je mimo rozsah přímého souboru (řešit algoritmem pro výpočet indexu nebo klíče!),
- ztráta poslední věty (po uzavření vstupních souborů je opomenuto zpracování resp. zápis),
- neadekvátní typ parametrů pro podprogramy a funkční procedury,

- chyby v synchronizaci souborů a testech změn klíčů (řešit obecnými prostředky!).

Závěrem této kapitoly bych chtěl uvést několik námětů a zásad, jimž se v těchto souvislostech vyplatí věnovat pozornost:

a) Odolnost proti chybným datům
   - je třeba si uvědomit, že data nejsou parametry a tudíž nesmějí mít zásadní vliv na vnitřní logiku programů - - vnitřní logika nesmí předpokládat jejich správnost.

b) Kontinuita zpracování
   - program by měl pokud možno doběhnout do konce; zpráva operátorovi, že např. nulou dělit nelze, nic neřeší; v programu je třeba využít např. default hodnot a po jeho ukončení provést analýzu výjimečných stavů nebo alespoň "post mortem" výpis.

c) Kontrola setříděnosti
   - program by měl kontrolovat, zda všechny soubory jsou správně setříděny a hlásit operátorovi, jestliže se opomenulo třídění určitého souboru.

d) Jednoduchost a jednoznačnost obsluhy
   - je základem filosofie zpracování z hlediska operátorského; nelze připustit zjednodušování programu na úkor operátorského komfortu.

e) Zásady komunikace s počítačem
   - jedná se o sjednocení zásad komunikace s počítačem jak ve vazbě program - operátor, tak operátor - program; nelze připustit soukromý "rozlet" programátorů - nutno omezit jak "povídání", tak zprávy typu ERR OVRFL IND J.

f) Programová a provozní dokumentace
   - nutno apelovat na aktuálnost a věnovat pozornost výjimečným stavům a restartům.

g) Srozumitelnost programů
   - program je třeba budovat už s perspektivou údržby a vývoje; co často podléhá změnám, musí být zohledněno v řešení.

h) Zkušební příklad

- jedná se o prověření správnosti a návaznosti všech programů; reprezentativní data jsou vhodná pro ladění jednotlivých programů, pro zkušební zpracování úlohy jsou nejlepším kvalifikovaným souborem reálná data většího rozsahu; při vyhodnocování má velký význam soustava kontrolních čísel a vazeb.

i) Knihovny

- jedná se o jednotný kompilační a knihovní systém se souladem programů na zdrojové a zaváděcí knihovně; využít generací a kopií.

9. Technické a systémové programové prostředí - závěry

Hardware a dnes hlavně operační systém vytvářejí atributy filosofie zpracování na počítači a musí tedy být i jiný konkrétní přístup při zabezpečování provozní spolehlivosti programů. Tyto rozdíly je třeba vidět např. v porovnání systému JSEP a SMEP. Přestože špičková technika SMEP, např. minipočítače SM-52/11, jsou co do rychlosti procesoru a kapacity pamětí srovnatelné s malými a středními počítači, např. s EC 1027, nelze přehlížet původní určení těchto systémů; na jedné straně k interaktivnímu zpracování a na druhé straně ke hromadnému zpracování dat (v těchto intencích je třeba budovat i sítě počítačů). Na adresu systémů SMEP je třeba uvést, že mají vzhledem k hromadnému zpracování dat nedostatečně zabezpečenou systémovou programovou ochranu dat a programů; kontrola se provádí pouze na hardwarové úrovni, nejsou využity například kontrolní sumy bloků či jiné softwarové prostředky. Totéž lze říci o dynamickém přidělování zdrojů výpočetního systému (hlavně periférií) a zabezpečování restartů. Na druhé straně zde existuje velmi pružný terminálový systém a předpoklady pro zpracování v reálném čase. To všechno se samozřejmě musí promítnout v interních zásadách při zabezpečování aplikačního softwaru včetně jeho provozu. Znamená to dotváření systémového software v souladu s určením výpočetního systému a vytvoření tomu odpovídající organizace při zabezpečování a ochraně dat a programů (systém kopií, generací, restartů, multiprogramování apod).

Jestliže jsou provozovány sítě různých počítačů s off-line převodem magnetických médií, chtěl bych upozornit na jednu zásadu.. Převody kódů a systémové struktury je nutno vždy provádět na tom systému, do kterého se soubor přebírá. Na tomto systému je nutno číst soubor na fyzické úrovni s protokolem o tom, které bloky resp. věty bylo možno převést a které nikoliv (např. v důsledku fyzické nekompatibility). Zásadním omylem je simulovat na jednom počítači strukturu souboru jiného počítače a na tomto druhém počítači už předpokládat mateřskou formu souboru.

Závěrem tohoto příspěvku by mohlo být, že provozní spolehlivost a odolnost aplikačního software je tvořena celým komplexem činitelů. Přesto však existují určité obecné zásady, na které jsem se pokusil upozornit.

Literatura

/1/ Streit, D.: K problematice parametrických programů. MAA 7/81

/2/ Streit, D.: Systém DMG pro manipulování s daty a výstup informací. MAA 1/83

/3/ Streit, D.: Zkušenosti se standardizací v oblasti konverzí a prvotních kontrol vstupních dat. Sborník Metody programování počítačů 3. generace, DT ČVTS Ostrava, 1977

/4/ Streit, D.: Integrovaná informační základna a interaktivní přístup v "nedatabankovém" dávkovém prostředí. Sborník Programování '82, DT ČSVTS Ostrava, 1982

/5/ Jiříček, P. a kol.: Problémy využívání rozhodovacích tabulek. Sborník Programování '78, DT ČVTS Ostrava, 1978

/6/ Honzík, J.: Dokazování programu. Sborník Programování '83, DT ČSVTS Ostrava, 1983

obr. 1: Technologická dekompozice systému zpracování dat