Jiří P E Š K A , prom. mat.

ZVT - OKR

Uživatelský software

V tomto článku jsem se pokusil zobecnit zkušenosti, které jsme nabyli při vytváření uživatelského software v Závodě výpočetní techniky OKR.
Poznámka: Namísto termínu uživatelský software jsem chtěl použít nějaký ryze český termín, ale nakonec jsem resignoval.

1. Co rozumíme pod pojmem uživatelský software.

Jsou to uživatelem vytvářené moduly, programy a systémy, které nemají pevně určeny vstupní nebo výstupní soubory, případně způsob transformace mezi nimi. Existuje zde jistá obecnost.

Jako protějšek uživatelskému software můžeme postavit moduly, které pracují s konkrétními vstupními daty. Takové moduly jsou používány v agendách. "Agendový" modul může většinou pracovat s užším rejstříkem vstupních dat a vykonává převážně činnosti, které jsou pevně naprogramovány a dají se jen nepatrně ovlivňovat zvenčí.

Protože míra obecnosti softwarových programů může být různá, je těžko stanovit přesnou hranici mezi agendovými a softwarovými programy. To ani není nijak nutné, neboť řada agendových programů by jen prospělo, kdyby splňovaly nároky

kladané na software.

Kdy vůbec vytváříme uživatelský software:

- jestliže výrobce počítače příslušný software nedodává,
- jestliže neznáme nebo šetříme prostředky na zakoupení software,
- jestliže si myslíme, že to dokážeme lépe než výrobce.

Ponévadž u III. generace počítačů by první důvod v zásadě neměl existovat, nabízí se otázka:

2. Je uživatelský software potřebný?

Ačkoli to tak nevypadá, často je ze třetího důvodu - ledacos můžeme udělat lépe než výrobce. Ten totiž má zkušenosti především se svými potřebami a výrobcovy potřeby nebývají vždy naše potřeby. Tak například firma IBM má znamem lepší ty části software, které si sama používá (linkage editor), než ty, které zřejmě sama příliš nepoužívá (generování ladicích dat).

Používání software v Závodě výpočetní techniky OKR ukazuje tabulka 1. Pro srovnání jsou uvedeny i údaje o využívání software výrobců. Sledování se provádělo za poslední čtvrtletí roku 1975 pro počítače ICL 1905 a IBM 370/145.

ICL				IBM			
% programů		% času		% kroků		% času	
1	2	1	2	1	2	1	2
14,6	13,6	14,3	11,1	13,7	50,8	13,3	29,7

Tabulka 1.

Vysvětlení k tabulce 1.

- Ve sloupcích označených 1 jsou údaje týkající se uživatel-

ského software.

- Ve sloupcích označených 2 jsou údaje pro firemní software.
- U firemního software ICL nejsou zahrnuty údaje o třídicím programu. Naproti tomu v software IBM jsou zahrnuty a tvoří zde nejvýznamnější položku.
- Tabulka zachycuje používání jednotlivých programů. Nezahrnuje tedy důležitou část software tvořenou moduly a rutinami začleňovanými do programů. Nic také neříká o významu generovaných modulů.
- Jako 100% programů (kroků) byl brán počet všech programů (kroků) zaznamenaných v počítačích v posledním čtvrtletí roku 1975.
- Jako 100% času byl vzat součet časů všech programů za IV. čtvrtletí roku 1975. U IBM nebyly zahrnuty časy systémových vstupů a výstupů.

Komentář k tabulce 1.

Z tabulky vidíme, že pro oba počítače z každé stovky programů patří 13 až 14 k uživatelskému software. Obdobný výsledek dostaneme i pro čas spotřebovaný uživatelským software.

Mimo to stojí za povšimnutí značný nárůst použití firemního software u třetí generace (IBM) proti druhé generaci (ICL). Poměr sice není přesný (u IBM je proti ICL zařazen třídící program, který sám přispívá asi 9,5%), ale přesto napovídá o rostoucím vlivu software. Veškerý používaný software u IBM patří k operačnímu systému. Na zvýšení se tedy nepodílí žádný nakupovaný software. Zvýšení je z velké části způsobeno tím, že u IBM je daleko více různých "pomocných" prací, jako je katalogizování, rušení a rezervování prostorů na discích a pod.

Závěr: Údaje v tabulce 1 v našem případě do značné míry potvrdily, že má smysl vytvářet vlastní software.

Přirozeně by bylo možno vyslovit řadu námitek. Například, že některé z věcí, pro něž byl vyvinut vlastní software, by

bylo možno zajistit i prostředky, které poskytuje výrobce.

Touto a dalšími otázkami se budeme zabývat v následující kapitole.

3. Jaký uživatelský software.

Pokud výrobce nedodává některé zcela běžné části software (např. třídicí program) a naše síly na tento úkol stačí, pak je odpověď snadná.

Uživatelský software se však vytváří i u počítačů vybavených rozsáhlým firemním software.

Tady hrají svou roli především individuální uživatelovy podmínky. V ZVT ČKD bylo například nutné programově zajistit datovou kompatibilitu mezi počítači ICL a IBM a vytvořit programovou podporu pro zpracování děrné pásky, které IBM nevěnuje pozornost.

Nyní se dostáváme k otázce ze závěru předchozího odstavce. Proč vlastní software, když jde použít firemní?

Pokud pominememe extrémní případy, jež skutečně nemají nárok na existenci, jako je kupříkladu snaha vytáhnout se s programem, udělat něco "extra", je nadhozená otázka podobná otázce: Proč FORTRAN, když máme assembler?

Zkrátka proto, že existující prostředky nejsou adekvátní vyskytujícím se problémům. Občas zjistíme, že dokážeme za daných podmínek udělat něco lepšího než výrobce i když to nebude mít patřičnou univerzálnost a celosvětovou platnost.

Často nám může právě přílišná universálnost a celosvětová platnost nejvíce vadit.

Nejčastějším problémem při hromadném zpracování dat je získat z dat rychle informace, o nichž jsme dříve netušili, že je budeme chtít, natožpak rychle. Mám na mysli svláště nejobvyklejší situaci, kdy se hledá chyba v posloupnosti za sebou jdoucích programů. Chyba se objeví ve výstupní tiskové sestavě a nikdo se dobrovolně nehlásí jako její původce. Pak je třeba prohledat předchozí datové soubory a zjistit, kde

k chybě došlo.

V takových situacích plných nervozity není nejvhodnější cesta psát a kompilovat vlastní programy. Rovněž dotazovací systémy, které nabízejí firmy, jsou pro aplikace tohoto druhu příliš těžkopádné. V ZVT se velmi osvědčily parametrické programy (ZVMP, ZEUS, ZVYPISY), které dokáží podle zadaných klíčů vyhledávat, vypisovat, vylučovat, opravovat jednotlivé záznamy v souborech dat.

To, že nejužívanější jsou programy pro manipulaci se soubory, ukazuje následující přehled:
(Pro srovnání jsou uvedeny i firemní programy)

ICL

Uživatelský software

	%programů	%času
1. ZVMP (vyhledávání, výpisy výběr, kopie, opravy magnetických pásek)	5,5	[illegible]
2. ZANY (generování tiskových sestav)	2,8	2,9
3. ZQMR (výpis magnetické pásky)	2,2	0,5
4. ZOFF (offline tisk sestav)	0,8	4,0

Firemní software

1. XQMM (kopírování magnetických pásek)	8,2	5,6
2. Kompilátory	-	-

IBM

Uživatelský software

	%programů	%času
1. ZEUS (vyhledávání, výběr výpisy, kopie, opravy sekvenčních souborů, tisky sestav)	2,6	2,0
2. ZVYPISK (výpisy souborů)	1,8	0,2
3. ZG.. (programy pro údržbu knihoven programů)	-	-
4. ZGED (vytváření souborů pro ladění)	0,9	0,6
5. IBMGPF (offline tisk sestav)	0,4	5,8

Firemní software

1. IERCOoo (třídění)	9,5	9,6
2. IEWL (linkage editor)	8,2	2,5
3. IEBGENER (kopie souborů)	5,4	3,4
4. IKFCBLoo (kompilátor COBOL)	4,7	3,4

Přehled nezachycuje význam té části software, která je mimo samostatně fungující programy; tedy různé subrutiny a moduly, jež bývají začleňovány do vyšších programových jednotek. Význam těchto modulů je značný. Nejen šetří práci programátorů, ale hrají důležitou roli při standardizaci a sjednocování pracovních postupů. Přitom jde většinou o nejjednodušší typ software.

Dalším krokem při rozvíjení vlastního software jsou samostatné programy a nejvyšším jsou generátory. (Jejich význam rovněž není v přehledu zachycen.)

Dále ve výčtu nezacházím, neboť nepředpokládám, že by si jednotliví uživatelé dělali kompilátory a operační systémy.

Závěr: Vytváříme takový vlastní software, který vyplňuje me-

zery a nedostatky firemního software, doplňuje jej ve směrech majících pro nás největší význam. Dbáme na to, aby vytvářený software byl úměrný našim silám, možnostem a potřebám.

4. Jak vytvářet uživatelský software.

Na tomto místě chci případné čtenáře informovat o tom, že v tomto článku se nic objevného nedozví.

Objevy v programování se dějí zřídka a navíc je neděláš já, ani je nemám z první ruky. Věci, o nichž chci psát, jsou známé. Tím více však na nich zaráží to, jak se jich nedbá.

Mohli bychom tedy už téměř udělat závěr: Uživatelský software (subrutiny, programy, generátory) vytváříme podobně jako agendové programy, ale lépe.

Které jsou to věci, na něž musíme klást zvýšený důraz.

A. Vymezení funkce programu.

Máme-li vybrán problém, musíme stanovit, v jaké šíři jej chceme řešit, tedy musíme určit, co všechno bude program dělat, případně, co by ještě mohl dělat v budoucnosti.

Ujistíme se, že míra obecnosti programu nedosáhne při zadávání parametrů a obsluze takové komplikovanosti, že zastraší případného uživatele.

Uvědomíme si přitom, že uživatele nezajímá, jak je program krásný uvnitř, ale co do něho musí investovat a co za to dostane. Uživatel chce investovat málo a dostat hodně. Chcete-li svým programem konkurovat jinému podobnému programu, rozhodne výsledek právě poměr

$$\frac{\text{jak se musím namáhat}}{\text{co za to dostanu}}$$

Nejlépe, je-li tento poměr blízek nule. Je-li větší než

jedna, program neděláme. V ušetřeném čase si raději uděláme pořádek v zásuvkách svého pracovního stolu a v dokumentaci starších programů.

Budeme počítat s tím, že skutečnost bude poněkud jiná (a to spíše horší), než si ji představujeme. Proto budeme předpokládat, že po zkušenostech se zavedením budeme program upravovat a vylepšovat.

Spočítáme si, za jak dlouho budeme s programem hotovi. Vyjdou-li nám dva roky a více, raději toho necháme. Při menším potřebném času si to ještě jednou rozmyslíme a pak se do toho pustíme.

To je ovšem přístup pro individualisty. Rozsáhlejší systémy se vytvářejí v týmech. To již dávno praktikují profesionální výrobci software a u nich se musíme učit, hlavně pokud se týká organizace práce.

B. Plánování vstupů a výstupů.

Vstupy

Tady je nejlépe si uvědomit, pro koho program děláme a podle toho se pak zařídit.

Zvláštní pozornost věnujeme vstupním parametrům. Budeme rozlišovat poziční a klíčové parametry.

Poziční parametry jsou takové, u nichž záleží na jejich pořadí nebo pozici. Například víme, že číslo podniku je za třetí oddělovací čárkou nebo že typ údaje je vždy za každou čtvrtou čárkou nebo údaj musí být v určitých sloupcích děrného štítku. Parametry s oddělovacími znaménky používáme, není-li jich mnoho. Vypadnutí jednoho oddělovacího znaménka způsobí posun všech následujících údajů a bolení hlavy uživatelů.

Klíčové parametry používají nějakého klíčového slova. Nezáleží u nich na pořadí. Např. PODNIK=05.

U klíčových parametrů uvážíme, zda jejich tvar nemůže být shodný s něčím, co již okruh potenciálních uživatelů

programů ovládá. Samozřejmě kromě toho, že jsou parametry stejné, musí mít i obdobnou funkci, nechceme-li vnést zmatek do řad přátel i nepřátel.

Nemůže-li být parametr zcela shodný s něčím známým, ale pouze podobný, uvážíme, zda právě tato podobnost nemůže být spíše na závadu. Jak známo, podobné věci si lidé pletou mnohem více než věci různé.

Při určování formy parametrů je vhodné dodržovat konvence, které platí na našem pracovišti. Ty by pak neměly být v rozporu s konvencemi firmy, jejíž počítač používáme. Zvláště způsoby předávání parametrů mezi moduly mají jasně stanovená pravidla. U IBM se předává v registru 1 ukazatel (pointer) parametrů, v registru 13 adresa oblasti pro uschování registrů, v registru 14 je adresa, odkud je modul vyvoláván, v registru 15 adresa vyvolávaného modulu. Konec vektoru parametrů je označen obsazením prvního bitu v posledním slově.

Výstupy

Musí odpovídat řešenému problému. Snažíme se automaticky uživateli dodat maximum vhodných informací. Údaje, jako jsou počty vstupních a výstupních vět, počet chyb, typy chyb a další, přijdou většinou uživateli vhod.

Zvláště nesmíme zapomínat na zprávy a kódy o chodu programu. To ale už souvisí s diagnostikou.

C. Diagnostika

Účinná a standardní diagnostika je jeden z hlavních činitelů rozhodujících o úspěšnosti programu. Ty je známá skutečnost a přitom se snad nikde neprojevuje lenost programátorů výrazněji než právě v této oblasti. Tady si programátoři nejvíce "šetří" práci.

Pokud je program se špatnou diagnostikou zaveden do praxe, nelze jej používat bez neustálého dohledu a komentování jeho tvůrce. Jestliže takový program chce od nás náhodou koupit jiné výpočetní centrum, je třeba spolu s ním prodat i

otce programu. A je to tak lepší.

Pomněme, že úspěšný je takový program, který se užívá a tvůrce o tom neví - program funguje i bez něho.

Každý modul musí při skončení předávat návratový kód, který uživatele zpraví o úspěšnosti chodu modulu.

Nemůže-li modul pokračovat, musí být zřejmé, kde skončil a proč.

Nešetřeme na zprávách. Ty by měly být pokud možno samovysvětlující. Manuály, kde jsou zprávy popsány, se totiž nejčastěji ztrácejí, je jich málo, jsou zastaralé.

Samozřejmě i zprávy musí zachovávat standardy. Nepopsaná či nerozluštitelná zpráva z programu je programátorský zločin, který by se měl trestat vsunutím zločinné ruky do mechanického snímače děrných štítků.

O technice diagnostiky se nejlépe poučíme u operačního systému IBM nebo RJAD.

D. Odolnost vůči parametrům a vstupním datům.

To je další důležitá vlastnost, kterou musí program mít. Parametry a vstupní data nesmí dokázat porušit program. Ze vstupních parametrů a dat si vybereme ty, u nichž by špatný tvar nebo rozsah mohl zavinit destrukci či chybu programu. Ty si, pokud nejsou dříve řádně zkontrolovány počítačem, dobře ohlídáme a nevěříme, že budou v pořádku, ani kdyby nám je psali vlastní rodiče.

Data, u kterých špatná hodnota nemůže ovlivnit náš program, si musí ohlídat uživatel. (Pokud ovšem není toto hlídání součástí funkce programu.)

Při ověřování parametrů existují dvě cesty.

I. Striktně odmítat vše, co se vymyká z limitu, trvat na vyplňování všech hodnot. Při nesplnění vydat zprávu a program ukončit.

II. Zachránit, co se zachránit dá.

Používáme předpokládané (default) hodnoty. To může být pohodlné při kontrolách. Hodnoty nastavíme na začátku a

pokud nejsou přepsány dodanými parametry, máme jistotu, že jsou v pořádku.

O tom, že byly použity předpokládané hodnoty, by mělo být mezi programem a uživatelem jasno.

Tento způsob můžeme uplatňovat tam, kde jsme si jisti, že tím nenastropíme vážnější škody.
Závěr: Byl na začátku.

5. Jak zavádět uživatelský software.

Když je program vypracován (k čemuž použijeme metod modulárního či strukturálního programování, podle toho, na co věříme), je více než řádně odladěn, popsán a dokumentován, začneme jej zavádět do praxe.

Existuje různý vztah uživatelů k programům a k modulům začleňovaných do programu. Ty druhé mají horší postavení.

Programové systémy nebývají na začátku své dráhy odladěny tak, aby si mohl být programátor jist, že tam nebude chyba. U některých programů má spíše tušení hraničící s jistotou, že tam nějaká chyba je. Jediná otázka je kde. Programátor v takové situaci není naladěn na provádění experimentů. Musí dostávat k užívání pouze spolehlivě fungující věci. Můžeme si být jisti, že i tak bude sledovat každý cizí prvek ve svém programu s nedůvěrou. Skončí-li program špatně, přijde se vás zeptat, zda jste to nezavinil vy se svou rutinou. Řídí se přitom heslem: Když je někde chyba, tak ji udělal někdo jiný.

Tady se budete muset (nejlépe s pomocí účinné diagnostiky) obhájit a dokázat, že jediná správně fungující část programu je vaše rutina.

Bohužel nepodaří se to vždy dokázat. Zde se uplatňuje pravidlo: S každým novým uživatelem - nové potíže.

Tyto potíže nejčastěji pramení z nedostatků popisu. Člověk až žasne, co při trochu komplikovanějších parametrech může uživatel zadat, aniž byste mu mohli něco vytknout. Příčinou jsou nejasně definované pojmy, nesprávná terminologie,

neúplnost a nejednoznačnost popisu.

Jak již bylo řečeno, existují situace, kdy programátor není nakloněn experimentům. Proto programátor, který si osvojil nějaký způsob práce, není ochoten se ho okamžitě vzdávat. To znamená, že softwarový program, který přijde až v době, kdy se lidé naučili řešit problém jiným způsobem, má zmenšenou šanci na úspěch.

Dobře můžeme programátory otrávit, rozhodneme-li se program zlepšit a při té příležitosti změníme některou funkci. Do zavedeného programu by se měly dodávat pouze nové funkce a staré se mohou rozšiřovat a zpřesňovat. Nikdy by však nemělo přestat fungovat to, co se dříve užívalo.
Závěr: Zavádění software není válka s uživatelem. Software má být řádně popsán a má fungovat (aspoň většinou).

6. A co dál?

Investice do vlastního software mohou být značné a proto si myslím, že by dále ještě něco mělo být.

Software, pokud za to stojí, by se měl rozšiřovat do dalších výpočetních center. Přes všechny snahy se tak děje málo. Největší úspěch v tomto směru zaznamenávají kluby utvářející se kolem jednotlivých typů počítače. Lidé se tam poznávají, více si navzájem věří, dochází často k různým neoficiálním výměnám.

Při oficiálním prodeji se objevují zádrhele. Vznikají diskuze kolem předávání dokumentace, změn.

Za systémový program o 1 000 instrukcích vypracovaný pro více uživatelů se podle ceníku platí 4 400 Kčs. Přitom může kupující požadovat tolik služeb, že se to prodávajícímu nevyplatí.

Otázky provádění změn u cizích uživatelů a udržování jejich dokumentace, jsou velmi závažné a nevím, jak by je organisace, která má dost starostí sama se sebou, mohla řešit.

Na druhé straně nemůže kupující platit za něco, co mu

nikdo není schopen udržet v pořádku.

Fakt je, že uživatelský softwere, který se prodává, musí být proti tomu, na co jsme zvyklí, o stupeň kvalitnější. Provedení, otestování, dokumentace chce více úsilí.

Závěr: Budeme se muset zlepšit.

Literatura

/1/ Firemní manuály IBM, EJAD

/2/ Walsh, Dorothy A.: A Guide for Software Documentation. Mc GRAW-HILL BOOK COMPANY

/3/ Ceník velkoobchodních cen obor 873 - Práce a služby výpočetní techniky. Federální cenový úřad.

/4/ Edice počítače, Alfa Bratislava.