P R O G R A M Á T O R A J E H O S V Ě T

Ing. Richard Bébr

Příspěvek se zabývá některými psychologickými a sociologickými problémy programátorů jakožto společenské skupiny. Z rozboru vztahů, projevů a situací se příspěvek pokouší nalézt závěry, použitelné v programátorské praxi.

Motto:

"... a hardware kráká: not - and - or !"
Ada C. Forth: úryvek z poemy "Upsibyte".

1. Systém "programátorství"

1.1 Základní pojmy:

Kdekdo neustále hovoří o "systémovém přístupu" a tak by se zdálo, že základy teorie systémů a systémového inženýrství jsou převážné většině řečníků i posluchačů běžně známy. V praxi to nebývá tak docela pravda a tudíž autor předpokládá, že laskavý čtenář nejprve osvěží své znalosti z tohoto oboru (případně i nahlédnutím do lit. /1/ a /2/).

Jak známo, systémem může být takřka cokoliv (automobil, mraveniště, spolek chovatelů drobného zvířectva, člověk, počítač a pod.). Pokusme se tedy považovat za systém celou množinu programátorů.

Pokud přísný systémový teoretik namítne, že systém je charakterizován společným zájmem neb cílem, můžeme se omezit na množinu programátorů, kteří se účastní semináře Programování v Ostravě; tato množina je dozajista reprezentativním vzorkem, takže poznatky, které získáme můžeme bez obav aplikovat i na jiné programátorské skupiny (systémy).

1.2 Systém a okolí:

Připomínám, že pojem "programátor" chápeme na seminářích v Ostravě vždy poměrně obecně (a zahrnujeme sem tedy i na př. analytiky a jiná podezřelá individua).

Systém "programátrstvíˇ se skládá z prvků:

- programátorů
- programů

a ze všech vzájemných vazeb mezi nimi (programátor - programátor, program - program, programátor - program).

Okolí systému je podle definice tvořeno takovými prvky, jejichž vzájemné vazby nás nezajímají a u nichž jsou důležité jen vztahy k prvkům zkoumaného systému. V našem případě tvoří okolí

- počítače (s technickým i programovým vybavením)
- lidé (vedoucí, provozní pracovníci, uživatelé a v širším smyslu celá společnost).

Takto vymezený systém podrobíme nyní zkoumání. Metodika nám ukládá analyzovat prvky a pak jejich vztahy. Prvek "programátor" je dostatečně popsán, rozebrán a zhodnocen (viz na př. lit. /3/ a /4/). O tom, co je to vlastně program se dohadujeme na seminářích "Programování" již dvanáctý rok. Budeme se tedy nadále zajímat pouze o vztahy mezi prvky systému navzájem a o vztahy prvků systému k okolí.

Jedna z milých definic praví: "Systémové inženýrství je aby se na něco nezapomělo"; v tomto duchu zaměříme průzkum na ta hlediska, která obvykle unikají naší pozornosti. Z rozboru by měly vyplynout náměty pro (alespoň drobná) zlepšení funkce systému.

2. Rozbor vztahů uvnitř systému

2.1 Vztah programátor - programátor (komunikace):

Důležité vazby mezi prvky našeho systému se realizují mluvenou a psanou komunikací. Vyjděme z modelové situace: pan Programátor jedná s jakýmsi hostem (na př. uživatelem) a tu vstoupí něžná dívenka s dotazem, co že se má dělat s programem - řekněme - EROS (evidence rozpočtů staveb). Host zbystří sluch a Programátor plynně pronese:

"Včera prolezla syntax, tak to naloudujte, zlinkujte a exekněte a kdyby to kleklo, vydampujte bednu ! ".

Host se psychicky hroutí, dívenka září nad jasnými pokyny a spěchá je provést; systémový odborník konstatuje, že intersystémová komunikace používá specifický jazyk, nesrozumitelný okolí. Víme, že většina profesí má svůj odborný slang, avšak programátoři v tomto ohledu dosahují úrovně nikdy předtím nevídané (či spíše neslýchané).

Jiným příkladem je "motto", uvedené v záhlaví tohoto příspěvku, které něco řekne pouze dobrému odborníku-programátorovi (který je současně znalcem světové poezie).

Prapůvodní příčinu výlučnosti programátorského "jazyka" nalezne systémový odborník v historii: programátorství je prvou profesí v dějinách techniky, ve které pracovník komunikuje se svým technickým nástrojem (strojem) slovní formou se širokou pojmovou bází.

Popsaná skutečnost má závažné důsledky. Obecně platí, že čím víc se určitá skupina odlišuje formou komunikace (jazykem) od obecné normy, tím se stává výlučnější se všemi dopady včetně negativních. Když se na př. v hospodě baví o své práci šoféři, fotbalisté, stavaři, umělci atd., dá se to poslouchat; jakmile začnou programátoři, lidé s hrůzou a nevolí odsedají.

Uvnitř systému má programátorský slang své výhody: je stručný, přesný, jednoznačný. Nikdy jej však nemáme používat ve styku s okolím; posluchače v lepším případě otrávíme, v horším popudíme.

Specifický jazyk proniká někdy i do písemných projevů. To už je ovšem hrubá chyba, která má za následek na př. nepoužitelnost dokumentace. Někdy se v podivuhodných variacích objeví slang i na výstupu počítače (na sestavách a obrazovkách). I tam zbytečně děsí nebo nervuje konzumenty (lit. /3/).

Zajímavým faktem je, že v jistých situacích programátor opouští svůj jazyk a stává se mile sdílným, srozumitelným a příjemným; to se děje na př. při tvorbě počítačových her (lit. /3/), při komunikaci programátora s dětmi nebo při tvorbě t. zv. sexuálně podbarvené dokumentace (určené pracovnicím, se kterými má programátor jisté úmysly - viz též odst. 3.3).

2.2 Vztah programátor - program (programování):

Program jakožto prvek systému má vazby k některým jiným programům. Tyto vazby jsou exaktně specifikovány (pokud ne, tak "to nechodí") a netřeba se jimi zabývat. Zajímavější je vztah programu a programátora. Tak jako se osobnost člověka jasně projeví při řízení auta nebo při hře v šachy, tak osobnost programátora se vždy promítne do jeho programu.

Uveďme příklad : znalec studuje jakýsi program v Cobolu; zočiv zápis

```
COMPUTE Q(I) = R1 / Z(I).
COMPUTE R1 = R1 - Q(I) * Z(I).
```

zaraženě a prohlásí: to psal zběh z oblasti VTV, který donedávna pracoval v Basicu, teď se pokouší o agendu a je zcela bezohledný k spolupracovníkům. Luštění významu zabere znalci drahný čas a vyžaduje použití vědeckých pomůcek, zejména magie, rumu, jasnovidectví a hádání z kávové sedliny.

Naproti tomu zápis téhož algoritmu ve tvaru

```
DIVIDE CASTKA BY PLATIDLO(I) GIVING POCET-PLATIDEL(I)
REMAINDER CASTKA.
```

rozjasní tváře znalce, který radostně zvolá: hle - výčetka platidel a psal ji starší, usedlý, života znalý, ohleduplný a velmi sympatický šuhaj. Nebo cobolské paragrafy

```
CHOBOTNICE. PERFORM CHAPADLO THRU CHAPADELKO.
KOLECKO. PERFORM RAFEK THRU OSICKA.
```

(vzato ze skutečnosti, autor si nic nevymýšlí !) prozrazují poeticky laděného, dovádivého, leč zcela nepraktického a poněkud přitroublého písaře.

Závěry si pozorný čtenář už vytvořil jistě sám (i bohatá literatura sborníků "Programování '75 až '85" k nim hodně přináší). Systémové pravidlo říká "účel světí prostředky" a účelem programu není jen "chodit", ale i být srozumitelný, čitelný a tedy i opravitelný a doplnitelný. Nadstavbou je pak program elegantní; tento pojem se sice vymyká teoretickým úvahám, pro praktika je však jasný a jednoznačný.

2.3 Vztah program - programátor (ladění):

Tak jako psaní programu realizuje vztah programátor-program, tak ladění je vztahem program-programátor. Systémový odborník zde s údivem konstatuje, že v tomto vztahu se program chová jako prvek živý, nadaný ďábelským důvtipem a mnohými intelektuálními schopnostmi. Při psaní programu vítězí programátorův duch nad reálnou problematikou, při ladění bojuje programátor proti intelektu svého výtvoru.

Známe dnes již klasické příklady úskoků, které zmátly i vynikající programátory. Uveďme ukázky:
Ve Fortranském programu předepsal autor cyklus

```
DO 20 I = 1 , 10
```

a děrovačka místo čárky ťukla tečku. Kompilátor odpáral mezery a zpracoval přiřazovací příkaz

```
DO20I = 1.10
```

a protože Fortran povoluje implicitní deklarace, nebyla hlášena chyba (proměnná DO20I je přípustná). Při běhu programu nastaly rozkošné zmatky, cyklus se neprováděl a pracovalo se s hodnotami I "od loňska"; v žurnálu se chyba přehlédla, neboť většina rychlotiskáren poněkud "rozmazává" a čárku od tečky prakticky nerozlišíme.

Nebo jiný nešťastník napsal podprogram

```
SUBROUTINE MALER (I,J,K)
REAL J
K = I * SQRT (J) + 0.5
RETURN
```

a v hlavním programu bylo volání

```
CALL MALER (3,5,K)
```

Chování při spuštění bylo depresivní a programátor pochyboval o svém zdravém rozumu dokud neopravil volání na správný tvar

```
CALL MALER (3,5.,K)
```

(Fortranisté jásají, ostatní se tváří nechápavě, autor se tlumeně pochechtává; není-li to někomu jasné, nechť se poptá u znalých kamarádů).

Takřka tragickým se stal případ podprogramu

```
      SUBROUTINE PRUS (I,J,K,L)
      L = 0
      DO 10 M = I , J , K
   10 L = L + M
      K = L / 5
      RETURN
```

který jeden ubohý chasník zavolal

```
      CALL PRUS (1,5,1,L)
```

a pak trávil osobní volno u stroje, proklínaje své povolání. Bystrý čtenář rychle odhalí, že zrádný podprogram mění i parametr K, volaný omylem konstantou 1. Tak se v hlavním programu změnila jednička na něco jiného, cykly chodily s podivuhodným krokem, N = N + 1 přičítalo jakási ďábelská čísla atd.

Autor zná případ, kdy program pro testování chyb ve vstupních datech byl "odladěn" na bezchybných údajích i případ, kdy

```
      SUBROUTINE BLB (A,B,C,D,E)
```

byla volána

```
      CALL BLB (X,5.4,Y,4.5)
```

s výsledkem na pomezí šílenství (v uvedených příkladech šlo o slovový stroj, kde se parametry předávají sekvencí adres).

Systémový odborník je nyní na rozpacích: jak zhodnotit vztah program-programátor při ladění? Dozvídá se, že existují metody prověrky programů (zaručeně probereme všechny větve programu, zaručeně použijeme všechny kombinace vstupních dat atd.), které však výše popsané chyby neodhalí. Zarazí se i nad případem, kdy zlomyslný uživatel přidělával kleštičkami otvory do vstupních štítků (devadesátek blahé paměti) a zmátl tím perfektně odladěné programy. Zhodnotí tedy vztah "ladění" jako exaktně nedeterminovaný na pomezí rutiny, intuice a umění. Dodá, že každý dobrý nápad by měl být v této oblasti respektován a počká si na další semináře "Programování XX", zda k tomuto zajímavému problému něco užitečného řeknou.

2.4 Zprostředkovaný vztah programátor - údaje:

Data sice nejsou prvkem systému, ale programy zprostředkují jejich zpracování. Programátor nástrojem = programem obrábí (zpracovává) materiál = data. Můžeme tedy údaje gnoseologicky chápat jako součást programových řešení (zde vidíte jeden z mnoha mazených obratů systémového inženýrství). Vztahem programátor-program se realizuje i ovlivnění dat. Jsou zde nějaká neobvyklá úskalí a potíže?

Samozřejmě. Příklad objasní jednu skupinu problémů:
do programu vstoupí data o finančním plánu a o jeho plnění jednotlivými útvary; výstupem má být procento plnění na jedno desetinné místo. Systémově myslící programátor si uvědomí, že
- bude-li plán 30 000 Kče
- budou-li 3 útvary
- a každý útvar splní 10 000 Kčs

musí se vytisknout

UTVAR1	10 000 KCS	33.3 %
UTVAR2	10 000 KCS	33.3 %
UTVAR3	10 000 KCS	33.3 %
SOUCET	30 000 KCS	100.0 %

a uživatel se okamžitě ozve, že součet ve sloupci 3 nesouhlasí. Není matematické metody, která by závadu odstranila a neexistuje způsob, jak uživatele přesvědčit, že výsledek je správný. Programátor se dostává do t. zv. "informačního squeezu" (pojem, známý hráčům bridže), kdy každý možný výsledek je špatný. Systémová věda může stav pouze konstatovat, nikoliv však řešit; musí ale důrazně upozornit na další hledisko, které problém ještě více komplikuje: plnění plánu se v hodnocení ekonomů opírá vždy o číslo 100; co je více, je překročení, co je méně (tedy i 99.999) je neplnění! Z pohledu ekonomických úředníků může uvedená výstupní sestava značně zkreslit závěry o plnění plánu!

Jinou skupinu problémů představují číselníky. Moudří myslitelé ve snaze umožnit "komputerizaci" stanovili různé platné a závazné pokyny. Nepříliš systémově myslící programátor nyní páchá agendu MTZ a uvažuje:

- existuje číselník měrných jednotek (směrnice .....)
- musím tedy donutit skladníky, aby místo "kg" psali "150"
  (za to mě proklejí)
- v programech musím držet tabulky, abych na sestavě vytiskl místo "150" údaj "kg" (což mě i počítač obtěžuje).

Nakonec si uvědomí, že příslušný výnos byl zřejmě míněn jako univerzální, umožňující řešit MTZ třeba i na Cellatronu. Slabší povaha podlehne a nechá vážit jablka na stopadesátky, silnější se vzepře, nerespektuje autority a usnadní skladníkům (i sobě) práci. Systémová teorie nesoudí, ale radí: buďme vždy rozumní!

Problematika dat je rozsáhlá a zasloužila by si samostatný příspěvek. Na závěr tedy uveďme alespoň inspirativní výrok nejmenovaného specialisty, který vždy hlásal fortranistům: "řekni mi, co dáváš do COMMON a já ti řeknu, jaký jsi". V cobolských programech pak zvláště studoval OCCURS a REDEFINES.

Systémový odborník shrne: řekni, jak řešíš datové struktury a já ti řeknu, jaký jsi programátor (a jak úspěšné budou tvé programy).

3. Okolí systému

3.1 Vztah programů k okolí:

V našem "psycho" přístupu (nemá nic společného s horrorem téhož názvu) jsme dokázali, že v programu je promítnuta osobnost programátora. Proto se v dalších odstavcích nezabýváme vztahem programů k okolí, neboť tyto vztahy jsou jen projekcí vztahů programátora k příslušným prvkům okolí.

3.2 Počítače:

Mezi programátorem a počítačem se realizuje klasický typ vztahu "člověk - stroj", ozvláštněný jistými mimořádnostmi.

Povšimněme si na př. problematiky generační:
S generacemi techniky se vyvíjejí i generace programátorů. Programátory I. generace u nás prakticky nemáme, kdežto programátorů, kteří začínali na II. generaci je u nás poměrně dost. Jsou to muži (někdy se vyskytne i žena) středního věku (zhruba od 101000 do 111111 let), ošlehaní děrnými i magnetickými páskami,

duševně buď zcela vyrovnaní nebo zcela nenormální. Vzpomínají často na zašlé časy a vyprávějí neuvěřitelné historky o krocení Cellatronů, ZPA šestistovek, Minsků a prvních Packardů. Jejich vztah ke III (a třiapůlté) generaci je dvojího typu: ti, kteří se nemohou rozloučit s bity, sekvenčními soubory na páskách a assemblerováním ("bitoví žongléři" - viz diskuse na "Programování '85") byli zahnáni mezi systémové programátory, kde se vyžívají tvorbou zcela neužitečných maker; jen občas jsou prospěšní celku (jde-li na př. o pomoc při hledání v rozsáhlých výpisech paměti). Druhá skupina se bez potíží přizpůsobila nové generaci (a přizpůsobí se i všem dalším), váží si toho, že programy už nejsou taková dřina a své přebohaté zkušenosti dokáže perfektně využívat. Na mládež se dívá s tichou a vyrovnanou skepsí. Programy těchto "převychovaných" programátorů II. generace poznáme podle některých drobných, ale neklamných známek (na př. instinktivní šetření pamětí). Typické bývá pro tyto pracovníky dodržování "starých dobrých zásad", na příklad v Basickém programu

```
10  A = A + 1
20  IF  A ..... 10 THEN GO TO 100
30  .....
    .....
90  GO TO 10
```

napíše moderní mladý muž III. generace jednoduše

```
20  IF  A = 10 THEN GO TO 100
```

(a samozřejmě to chodí), kdežto programátor - klasik určitě

```
20  IF  A >= 10 THEN GO TO 100
```

Podobně nulování celkového součtu zajišťuje mládež

```
77  SOUCET PIC 9(8) VALUE 0.
```

kdežto stařešinové

```
77  SOUCET PIC 9(8).
.....
.....:  MOVE ZERO TO SOUCET.
```

Jsou to jemné rozdíly, nepodložené žádnou exaktní úvahou. Systémový odborník po dlouhém bádání odhalí, že jde o reflexy zážitků z pravěku, uložených v podvědomí; je to třeba strach (u starších strojů a operačních systémů oprávněný), že 2 + 1 nebude zcela přesně 3 a podobně.

Mládež III. generace (zhruba pod hexa 28 let) je - samozřejmě - sebejistá, drzá a nevychovaná, historkám starších kolegů se skrytě nebo zjevně vysmívá a odmítá dobře míněné rady; to však konec konců není problém úzce programátorský. V programech je mládež velkorysá, o 10 kBytů sem nebo tam se nestará, diví se, proč starý pán píše A * A * A , když se dá napsat A ** 3 a podobně. Jen občas ve slabší romantické chvilce vrhá mládež potají obdivné pohledy na staršího kolegu, který kdysi dostal do 4 kB celou agendu a to si k tomu musel sám udělat logaritmus a odmocninu.

Hloubkovým průzkumem odhalí systémový odborník další generační zajímavost: máme generace I,II,III,III a půl a nyní se očekává nástup generace V. Kde je IV. generace? Je to na způsob hotelových pokojů 11,12,12a,14 nebo je to něco jiného? Systémová věda nenalézá řešení; domníváme se však, že vážnou Computer Science zde ovlivnil nějaký rozdováděný novinář nebo reklamní textař. Je štěstí, že se to stalo (zatím) jen v otázce číslování generací !

3.3 Šéfové:

Programátor má svého šéfa, ten zase vyššího nadřízeného a tak to jde dále až ke cobolské úrovni 01. Vztah k vedoucím je formálně přesně stanoven, rozebereme tedy obsahovou stránku.

Vytvořme dvoubitové označení šéfa

```
X Y
|  └──→ znalost programování (0=ne,1=ano)
└─────→ znalost řízení (0=ne,1=ano)
```

a máme 4 typy šéfů:

Typ 01 se hodí pouze na nejnižší řídící úroveň (šéfanalytik, vedoucí týmu). Na vyšších úrovních se neosvědčuje, protože se neustále vrací k programování (které však léty sezení na šéfovské židli zapomíná), trapně šťourá do programů a místo aby se staral o chod útvaru, tajně si v šuplíku prtá soukromé prográmky, ze kterých čiší nostalgie a časem i fušeřina. Šéf 01 je pověstný tím, že si sám udělá roztodivný program pro výpočet premií, přičemž mu nezbyde čas na vyhádání dostatečně vysoké částky odměn pro útvar.

Typ 10 je vynikající a lze ho všude doporučit. Stará se o podří-

zené, promptně řeší jakýkoliv problém a nezdržuje programátory zasahováním do jejich práce. Je až s podivem, jak dobří reprezentanti typu 10 dovedou bez odborných znalostí odhadnout odvedenou práci a jak si tito borci umějí vybírat kvalitní odborně vyspělé spolupracovníky, kterým mohou plně důvěřovat (a na této důvěře založí způsob šéfování).
Typ 11 se vyskytuje vzácně; měl by být nasazován na špičkové řízení špičkových úkolů. Díky vyšším šéfům typu 00 se to však zpravidla neděje. Vedoucích typu 11 se nadřízení bojí a tak je raději izolují a uklidí na místa, kde se typ 11 nemůže projevit. Typ 00 každý jistě dobře zná. Ze systémového hlediska není na těchto vedoucích nejhorší to, že nic neumějí ani to, že sami sebe vždy považují za typ 11. Mnohem vážnější je skutečnost, že podvědomě svou kvalifikaci 00 cítí a z toho se u nich vytváří komplex, který se šéf 00 snaží kompenzovat úsilovným předváděním svých "schopností" ve všech směrech a situacích; takové křečovité demonstrace vedou k debaklům, které komplex prohlubují a šéf 00 se ocitá ve zpětnovazební smyčce, ze které není úniku. Vnější efekt tohoto psychického stavu je pak otravování podřízených a ve svém souhrnu velice nízká kvalita i kvantita díla.

Ideální je tedy hierarchie: vedoucí týmů 01, jim nadřízen 11, všechny další vyšší stupně obsazeny 10. K praktické realizaci tohoto ideálu nemůže ovšem autor podat žádné rozumné rady.

3.4 Provoz:

Vztah programátora k provozu se realizuje dvojím způsobem:

a) přímo (osobním kontaktem)
b) nepřímo (respektováním provozních potřeb a problémů v programech a jejich dokumentaci).

Přímý vztah je neúprosným pokrokem nemilosrdně mýcen. Ladí se buď v close-shopu nebo z terminálů. Doby, kdy se na sále neustále potulovali programátoři, besedovali se směnaři, poplácávali operátorky a MP jednotky, slídili kolem tiskáren, hrabali se ve štítcích a mezitím ladili, jsou nenávratně pryč. Z hlediska primárních funkcí je to v pořádku, ale co do sekundárních efektů je to vlastně škoda. Programátor se od provozu stále více vzdaluje, respektování provozních požadavků je nutno vynucovat předpisy, směrnicemi a jinými lejstry. Dokud byl "provoz" v programátorově podvědomí představován tu sympaťákem směnařem, tu milou tvářičkou

operátorky neb děrovačky, tu ráznou avšak nobleení vedoucí VVK, dotud nemohl programátor (podvědomě) příliš komplikovat těmto příjemným lidem život. Jestliže se však provoz stal institucí, reprezentovanou jen strohými a nepříjemnými spisy, směrnicemi a příkazy, stává se ve vrcholné fázi až nepřítelem programátora, který s nechutí plní různé nesmyslné požadavky a (více či méně podvědomě) se je snaží obejít, případně zcela ignorovat.

Řešení tohoto stavu je obtížné a u velkých středisek prakticky nemožné, má-li být založeno pouze na úředně formální bázi. Vyšší vedoucí by si však měli tento stav uvědomit a věnovat mu pozornost. Protože jde o problém převážně sociologický, musí konkrétní opatření vycházet z individuálních podmínek dané situace.

3.5 Uživatel:

Na téma "vztah programátora k uživateli" již bylo napsáno hodně úvah (na př.lit /3/). Základní pojmy byly podrobně prozkoumány, existují recepty, návody, předpisy, směrnice.

V pojetí tohoto příspěvku upozorníme na hledisko psychologické. Tak jako kvalitní šachista odhadne soupeřovy psychologické slabiny a podle nich volí způsob hry, měl by dobrý programátor odhadovat psychologické rozpoložení uživatele - ne proto, aby nad ním zvítězil, ale naopak aby uživateli co nejvíce usnadnil jeho těžký život.

Uveďme příklad: v jistém podniku zavedli personální agendu; vzhledem k naprostému vzájemnému nepochopení uživatele a řešitele bylo řešení špatné a po několika měsících se zpracování zastavilo, neboť jinak by se pracovnice personálního útvaru zhroutily. Nejmenovaný řešitel dostal nedávno za úkol vyřešit a zavést v tomto podniku novou personální agendu. Drastická zkušenost, zapsaná do vědomí i podvědomí pracovnic uživatele vytváří tvrdou psychickou bariéru, takže je předem jasné, že i to nejlepší řešení narazí na zoufalý odpor uživatele. Hrubou chybou by bylo využít nyní předpisů, příkazů a směrnic a uživateli zavedení agendy vnutit (nařídit). Řešitel se rozhodl pro poněkud neobvyklý postup: nejprve se naprogramují a zavedou ty úlohy, které budou poskytovat uživateli zajímavé a užitečné výsledky, nebudou však

vyžadovat žádnou aktivní činnost uživatele (na př. vkládání dat). Stojí to sice trochu přemýšlení, ale jde to (především využitím jiných již provozovaných agend a vstupů "odjinud"). Bližší detaily může autor popsat v kuloárních diskusích. Věříme, že drobné užitečné úlohy postupně obnoví důvěru uživatele.

Jiným příkladem je řešení jednoduché agendy pro podnik, který si zakoupil personální počítač. S výpočetní technikou nemá zkušenosti, ale chtěl by své pracovníky pro ni získat. Při programovém řešení bylo plně využito možností daného hobby-počítače, zvláště pak grafiky a akustického výstupu. Uživatelská činnost se podobá hře, akce jsou doprovázeny příjemnými melodiemi, chybové vzkazy jsou graficky zdůrazněny a podloženy signálem "hoří". Před analytickým návrhem proveděl však řešitel průzkum, zda uživatel toto "vylehčení" psychicky vydrží; šlo o to, zda se v uživatelských řadách nevyskytují ponuří úředníci známého sucharského typu, kteří by hravé řešení považovali za hrubé znevažování a zesměšňování vážné úřední práce. Bylo zjištěno milé a příjemné pracovní klima bez zkostnatělých elementů a tak bylo popsané řešení realizováno.

4. Závěr

Správný systémový přístup se vždy snaží o harmonické skloubení všech myslitelných hledisek, neboť jen touto cestou vytvoříme systémy vyvážené, elegantní, úspěšné a po všech stránkách užitečné.

Literatura

/1/ Štech,J: Základy teorie systémů. SNTL,1982.

/2/ Haluda,M: Systémové inženýrství. VADAS, 1980.

/3/ Bébr,R: Příspěvky ze seminářů Programování
- Počítačové hry a jejich význam. 1981.
- Uživatel a jeho svět. 1982.
- Přežije programátor rok 2000 ? 1985.

/4/ Demner,J: Typologie programátora. Příspěvek semináře SOFSEM, též VTM, č.19, roč.39/1985, Kladá fronta.