SUBJEKTIVNÍ FAKTOR V PROGRAMOVÁNÍ

aneb
Co dnes dělá opravdový programátor?

Petr KOUBSKÝ[1]

Niklaus Wirth, tvůrce Pascalu, byl při jedné své přednášce dotázán, jak vyslovovat své jméno. Odpověděl: Můžete mne volat jménem (Wert) nebo hodnotou (Worth). Z této poznámky je zřejmé, že N. Wirth je pojídač koláčů. Opravdový programátor uznává jediný mechanismus předávání parametrů: volání hodnotou po návratu, tj. referencí, jak je to implementováno v kompilátorech IBM/370 FORTRAN G a H. Opravdový programátor nepotřebuje ke své práci všechny tyto abstraktní pojmy, je úplně šťastný, má-li děrovač štítků, kompilátor FORTRAN IV a pivo.

-Ed Post: Opravdoví programátoři neužívají Pascal

Určitě jste ho všichni znali: Postova *opravdového programátora*, který neužívá Pascal, na večírcích debatuje o bezpečnosti operačních systémů a jak ji obejít, nezdržuje se psaním dokumentace a nikdy by nehovořil k počítači s myší. Svého času [1] posloužil citovaný článek jako úvod bouřlivé panelové diskuse na SOFSEMu, a později koloval mezi lidmi od počítačů v nejrůznějších formách samizdatu, od trpělivého opisování přes xerox po magnetické pásky a diskety. Myslím, že takovou oblibu nezískala snad jen proto, že je neobyčejně vtipná (jako že je), ale především proto, že se dotkla citlivých problémů, aktuálních dodnes. Proto jsem se rozhodl sfouknout z ní prach a podívat se, jak se do dneška, případně do zítřka oboru promítá dilema věčného sporu mezi pojídači koláčů na jedné straně a těmi, kdo umějí psát fortranské programy v jakémkoli jazyce na straně druhé. Že jsem se přitom dostal do úplně jiných končin, za to může víc naše bouřlivá doba, ve které se toho tolik děje, než já sám. Předem slibuji, že jsem si Posta vypůjčil jen jako uvedení do problému, a že se rozhodně nebudu snažit napsat jeho článek podruhé. Rovněž se chci omluvit všem, kdo snad očekávali, že opět půjde o nějakou legraci.

Vyhoďte ho dveřmi, vrátí se vám oknem

Kdykoli je dnes řeč o programovacím stylu, vždy se diskutující nakonec dostanou ke strukturovanému programování, anebo vůbec k metodám softwarového inženýrství. Nic proti nim.

---

[1]Ing. Petr Koubský, CSc., závod software, JZD AK Služovice
Software, PO BOX 226, 111 21 Praha 1

Strukturované programování je výborná metodika (nebo možná přesněji: filozofie řešení problémů na počítači). A rozpoznání softwarového inženýrství jako samostatné disciplíny umožnilo exaktnější pohled na zákonitosti tvorby software, což v důsledcích vedlo a vede ke zvýšení efektivity programátorské práce. Je pošetilé tyto přístupy zamítat, a nebudu-li jim dnes nijak zvlášť dělat advokáta, pak hlavně proto, že v roce 1990 to už považuji za zbytečné.

Neméně pošetilé je ovšem absolutizovat možnosti a dosah inženýrského a strukturovaného přístupu. K oblíbeným iluzím systémových analytiků patří představa, že popíšou strom, přidají slova "krát deset tisíc" a dostanou tak použitelný popis lesa. Systémům se říká systémy právě proto, že jsou čímsi víc, než souhrnem svých částí. A počítačové systémy jsou tvořeny nejen hardwarem a softwarem, ale také lidmi a navíc složitými vazbami, které propojují všechny tyto složky.

V analogii k matematice, přírodním vědám a některým technickým disciplínám se klasický inženýrský přístup k programování snaží vyloučit ze svých úvah subjekt programátora. Programátor je prostě jen dalším procesorem, který se chová optimálně, píše programy v souladu s určitými apriorními zásadami a do výpočetního systému nepatří – je vně. Takový přístup je naprosto pochopitelný a oprávněný z historického hlediska. Inženýrský přístup přece vznikl proto, aby čelil krizi software, aby zvýšil produktivitu práce programátorů, aby ze softwarové manufaktury udělal solidní průmyslovou velkovýrobu. Což se, s jistými výhradami, všechno povedlo. Ale inženýrskému přístupu přitom zbylo jen málo prostoru k vlastnímu rozvoji. Stále víc lidí začíná cítit kánony strukturovaného programování jako dogma; subjektivní pohled na tvorbu software, vyhozený z teorie a metodik dveřmi výpočetního střediska, se začíná vracet zpět oknem Norton Commanderu na obrazovce ťáťéčka.

Ve svém příspěvku si chci všimnout několika otázek, které spolu souvisejí velmi volně. Lze je však považovat za různé aspekty téhož problému: úloha subjektivního faktoru v programování a ve využití počítačů, nebo obecněji: vztah člověka a výpočetní techniky. Mezi lidskou duševní prací a počítači existuje neobyčejně intenzivní *oboustranné* propojení. Uvažujme tedy o charakteru tohoto propojení, o jeho důsledcích, a to jednak pro programování a aplikace výpočetní techniky jako disciplínu, jednak pro člověka – jednotlivce a konečně pro vývoj společnosti jako celku.

Estetika programování

Nemůže pracovat dobře ten, kdo na předmět a výsledek své práce nehledí jako na estetický objekt. To platí pro popeláře i pro programátora. V programování lze rozlišit [2] *vnější* a *vnitřní estetiku*. První z nich se týká vstupů, výstupů, komunikace s uživatelem, tisků, tedy souhrnně řečeno, je to estetika uživatelského rozhraní programu. Vnitřní estetika souvisí s programovacím stylem.

Elegance vnějších projevů programu je v podstatě elegancí průmyslového designu, a dobrý design musí především ctít funkčnost výrobku. Estetická měřítka pak odvozujeme právě od tohoto hlediska: hezké je to, co je užitečné. Karel Čapek ústy svého přednosty stanice v Obyčejném životě [3] o tom říká: "*To jsou sta malých problémů, jak upravit službu, jak předělat pořádek věcí, kde parkovat prázdné vozy a kdesi cosi; nedělám své krásné nádraží z kytiček jako starý pán, ale z vlakových souprav, z krásného pořádku, z hladké a tiché cirkulace. Každá věc je krásná, je-li na svém místě; ale takové místo je vždycky jenom jedno a není každému dáno jej nalézt. A najednou je tu jakoby větší a volnější prostor, věci mají svůj čistší obrys a nabývají čehosi jako vznešenosti; tak, teď je to to pravé.*" Je to tak, a my si musíme dát pozor, abychom nedělali své programy krásné z kytiček, z animovaných šestnáctibarevných obrázků a dvojhlasých melodií, ale naopak z *hladké a tiché cirkulace*: z přehledné obsluhy, z jasného rozlišení funkčních stavů, z rozumných chybových hlášení, ze zvýraznění důležitých věcí a vynechání nedůležitých.

K takové estetice má inženýrství mnoho co říci. Ovšem spíše inženýrství jako filozofie, jako solidní a poctivý přístup ke světu, než speciálně inženýrství softwarové. Kdo se chce naučit psát dobré programy pro uživatele, tomu může citovaná pasáž z Čapka dát víc než celá kapitola z Dijkstry či Wirtha. Mám pocit, že je zde potřeba - kromě programátorské řemeslné zručnosti, kromě zkušeností odborné - také jistá životní zkušenost a lidská vyzrálost. Mám pocit, že nejlepší uživatelská rozhraní píšou vyrovnaní, rozvážní lidé s vysokou sociální inteligencí, lidé vnímaví, se schopností empatie, vcitování do druhého. Mám pocit, že cholerický, šílený programátor s ASCII tabulkou v hlavě a zbytky jídla ve vousech může napsat geniální program, ale nikdy k němu nedokáže udělat geniální ovládání, nikdy nezvládne dialog programu s uživatelem, nikdy se nestrefí do toho, co vlastně zadavatel chtěl - i kdyby se o to všechno třeba snažil. Věřím, že právě zde hraje subjekt programátora velice významnou roli.

A pak je zde vnitřní estetika programu. Ne tedy, jak se program tváří navenek, ale jak je naprogramován. Softwarové inženýrství tuto otázku obvykle ignoruje, a pokud se jí přece dotkne, pak ji opět plně ztotožňuje s funkčností programu. Pečlivá volba řídicích a datových struktur, dekompozice, přehledná grafická úprava programu, použití samovysvětlujících se konstrukcí a názvů, dostatek komentářů, jasná vazba programu na dokumentaci.

Problém je však v tom, že mnoho programátorů - a často těch nejlepších - podobná estetická měřítka neuznává, nebo uznává jen v omezené míře. Mnoha lidem působí daleko silnější estetický zážitek napětí mezi těžko srozumitelným zdrojovým textem a perfektně fungujícím programem než odsazené cykly a obsažné komentáře. Je to zcela pochopitelný jev. Jsou-li tvůrčí práci vnuceny - byť třeba v sebelepším úmyslu - umělá omezení, začne velmi brzy zkoumat způsoby, jak tato omezení obejít nebo překonat; neučiní-li tak, ztratí svůj tvůrčí charakter.

Podívejme se na malý příklad. Máme banální zadání: napsat v jazyku FORTRAN logickou funkci, která zjistí, zda je dané číslo sudé nebo liché. Nejběžnější řešení uvádí každá učebnice:

```
LOGICAL FUNCTION ODD (K)
ODD = K/2*2.NE.K
RETURN
END
```

Funkce využívá jednak vlastností celočíselného dělení, jednak toho, že operace stejné priority se provádějí zleva doprava. Je součástí programátorské abecedy, je to jeden z těch obratů, kterým se učíme spolu se samotnými základy jazyka - jakýsi *fortranský idiom*, chcete-li. Je přehledná, nepotřebuje komentář.

Nyní si ukážeme další možnost řešení:

```
LOGICAL FUNCTION ODD (K)
ODD = K.AND.1
RETURN
END
```

Jaká je tato verze? Úderná, stručná, překvapivá a tajemná; kdo nezná principy vnitřního zobrazení celých čísel, nemůže jí rozumět. Jde až za hranice jazyka na území nikoho: norma FORTRANu logické operace s operandy typu INTEGER ani nezakazuje, ani nepovoluje. Z hlediska strukturovaného programování je taková verze samozřejmě úplně špatná. Obsahuje dvojitou implementační vazbu; přestože lichá čísla většinou mají v řádově nejnižším bitu jedničku a přestože výraz K.AND.1 většina fortranských překladačů přeloží v souladu s úmyslem zde uvedeným, nebude to určitě platit všude a vždy. Ubezpečuji vás však, že naprosto nezanedbatelné procento programátorů bude druhou variantu považovat za elegantnější než první a pro některé z nich to bude dostatečným důvodem, aby ji použili - bez ohledu na rizika, jež to s sebou nese.

Argumentuji tím pro záměrné zatemňování smyslu, pro vytváření samoúčelných šarád a rébusů? Snažím se vytvářet iluzi hloubky tam, kde žádná není? Neřekl bych. Spíš si myslím, že jde o přirozenou reakci na strnulost málo inspirativních kánonů strukturovaného programování. Architekta s tvůrčími ambicemi taky určitě nebaví projektovat panelové domy podle jednoho ze sedmi či kolika typizovaných vzorů. Programátor navíc, na rozdíl od architekta a koneckonců na rozdíl od inženýra v jakémkoli jiném oboru, pracuje s nekonečně pružným a tvárným materiálem: s jazykem. Programovací jazyky mají překvapivě mnoho společného s jazykem přirozeným, tedy s tím, v němž píšeme - občas - básně a milostné dopisy, a ovšem také - častěji - úřední korespondenci a příspěvky na různé konference. O druhé verzi funkce ODD jsem napsal: úderná, stručná, překvapivá, tajemná, překračující hranice jazyka do území nikoho. To by klidně mohla být charakteristika dobré moderní básně! Úplně soukromě si myslím, že právě zde je původ síly, které nutí některé programátory psát poněkud temně; podvědomě cítí, že jazyk, jenž je jejich

nástrojem a prostředím pro jejich myšlenky, má na víc, než k čemu je obvykle používán. Ale tohle tvrzení berte, prosím, už spíš jako metaforu než doslova.

Linie fronty je tedy vytyčena. Na jedné straně realismus - solidní, spolehlivý, účelný, bezpečný, trochu rozvláčný a často nudný, a na druhé straně romantika - přitažlivá, magická, inspirující, nebezpečná a zrádná. V umění bývá romantismus reakcí na realismus a naopak. Snad je tomu podobně i v programování.

Viry, hackeři a otázka subkultury

Robert Morris, třiadvacetiletý student v džínsech, tričku a teniskách, zamával za provázek od sítě - a svět se zachvěl. Tedy přinejmenším ta část světa, která počítá dočasný výpadek svých výpočetních systémů v milionech dolarů. Morris dokázal stořádkovým programem v REXXu zastavit 6000 počítačů po celém území USA a škoda údajně činila $100,000,000, nemluvě o ztrátě důvěry zákazníků ve služby federálních počítačových sítí. Padl tak mýtus o nenarušitelnosti významných počítačových systémů, pokud vůbec kdy existoval. Pentagon sice záhy přispěchal s ujištěním, že ve vojenských sítích se nic podobného stát nemůže, ale věrohodnost takového tvrzení je nevelká.

Na světě je pořád víc hardware, a navíc jednotlivé systémy jsou stále více navzájem propojeny. Vzniká tak souvislé elektronické či informační prostředí: prostředí celosvětových počítačových sítí. Stát se účastníkem sítě je snadné: potřebujete jen péécéčko, modem a telefon. Po připojení je už jedno, zda zrovna sedíte v Heidelbergu, Islámabádu či v Sydney. Počítač sám o sobě je ohromná hračka, intelektuální magnet vysoké přitažlivosti. Ale co potom říci o síti? Neobyčejně vydatný přístup k informacím, neomezené možnosti komunikace, ničím neohraničená kapacita zpracování údajů - to jsou podstatné vlastnosti zapojení počítačů do sítí. Taková technologie nutně inicializuje rozsáhlé změny ve společnosti. Je ostatně známo, že stále klesá podíl pracujících, kteří se zabývají přímou materiální výrobou (v USA už jen 12%), a naopak roste podíl těch, kteří se tak či onak živí zpracováním informací.

Někde na okraji informační společnosti pak stojí ti, kteří parazitují na jejích funkcích nebo je poškozují. V dnešní době se nevyplácí přepadat dostavníky a poštovní vlaky. Informace mají cenu hotových peněz; a kdo je umí nelegálním způsobem získat, ten ony peníze vydělá. Hackeři, většinou mladiství a často vynikající programátoři od pánaboha, však málokdy pracují pro peníze. Jejich motivací je síť sama, sám výpočetní systém. Snaha proniknout, zorientovat se v zašifrovaných tajemstvích, hrát nebezpečnou hru, přelstít anonymního a mocného protivníka. Za *napichováním* systémů cítím prastaré mýty o boji hrdiny s čarodějem na jeho vlastním území. Třaskavá směs logického a magického, mýtu a reality, to je intelektuální motivace nejlepších hackerů. Ale kousek tohoto pohledu na svět je zároveň obsažen v každém opravdu tvůrčím programátorovi.

Podobně je tomu i s počítačovými viry. Knížka [4] uvádí výčet patnácti alternativních důvodů, proč se viry píšou a rozšiřují: recese, provokace, pomsta, vytváření umělého trhu, ochrana zájmů, teror atd. Někde pod tím je ale téměř vždy schována subjektivní motivace programátora, jeho vzrušení nad možností něco takového vůbec naprogramovat. Znám programátory, kteří otevřeně přiznali, že z autorství *dobrého* (rozuměj: účinného, úspěšného) viru by měli větší radost než z normálního produktu.

Je třeba otevřeně říci, že v prostředí počítačů vznikla svérázná subkultura, která se vyznačuje vlastní filozofií, vlastním slangem, vlastními hodnotovými měřítky a morálními kategoriemi. Tato subkultura se zviditelnila masovým rozšířením osobních počítačů. Programátoři jí často podléhají a jsou - ať už si to uvědomují či nikoli - vázáni jejími principy. Počítače dnes přitahují nejlepší mozky generace, stejně jako třeba v padesátých letech jaderná fyzika. Celkem běžně se již používá termín *počítačová inteligence* pro označení těch, kdo spojují svou duševní práci s výpočetní technikou. Většinou to jsou mladí lidé; odborně brilantní, rychlí, pohotoví, často se zřetelně porušenou hodnotovou orientací a s minimálním zájmem o nepočítačový svět. Tvoří širokou škálu, na jejímž jednom konci nacházíme fanatické hackery, hráče a hitové bastlíře, na konci druhém pak třeba špičkové odborníky na umělou inteligenci, postulující člověka jako speciální případ kognitivního stroje.

Rozlehlé systémy a fatalismus

V každé úvodní knížce o počítačích najdete důrazné upozornění na jejich naprostý determinismus. Počítač se ve své činnosti slepě řídí programem, provádí posloupnost instrukcí vytvořenou programátorem a nic jiného dělat nemůže - asi takhle se to obvykle dočtete. Snad je v tom zdůrazňování i podvědomá snaha ospravedlnit se před veřejností: Golem je naprosto bezpečný, v každém okamžiku ho máme pod kontrolou. Ale je to pravda? I deterministický systém může být obtížný k pochopení, je-li složitý. A pokud jeho složitost přeroste určitou mez, pak z našeho subjektivního hlediska zanikne rozdíl mezi deterministickým a stochastickým chováním [5]. Jestliže necháme soustavu deterministických pravidel růst, pak se dříve či později stane natolik obsáhlou, že ji už nebudeme schopni úspěšně použít k predikci chování systému. A to je právě případ velkých programů.

Ona dnes již není plně adekvátní ani představa provádění posloupnosti instrukcí, kdy je programátor železničářem nastavujícím výhybky ve složitém kolejišti logických cest. Taková analogie sice zůstává v platnosti pro většinu počítačů a způsobů programování, ovšem na druhé straně přibývá těch, které se - každý po svém - vymykají: neimperativní programovací jazyky jako PROLOG a Lisp, paralelní systémy a asociativní paměti, systolické sítě a neuropočítače, do určité míry i expertní systémy a jiné produkty umělé inteligence.

Je třeba si uvědomit, že rozsáhlé systémy dnes nepíšou jednotlivci, ale týmy. Programy často vznikají postupně po dobu několika let, složení týmu se přitom mění, někdy se v průběhu řešení vyvíjí i celková koncepce. Důsledkem je, že neexistuje žádný jednotlivý programátor, který by měl globální přehled o fungování programu. Jednoho dne tady program prostě je, trochu podobný neznámému zvířeti nebo kamenu z Měsíce. Můžete ho používat, můžete zakládat svá rozhodnutí na výsledcích, které produkuje; ale nesnažte se zjistit, jak funguje. Nemáte nejmenší šanci.

Jakže?, řeknete. Vždyť přece existuje dokumentace. Vždyť jsou k dispozici výpisy zdrojových textů. Na discích jsou zdrojové i přeložené moduly, a pokud ani to není dost důvěryhodné, pak můžeme udělat disassembler zaveditelných programů nebo dump paměti nebo zavést trasování nebo...

Ano, ovšem. To všechno se dá udělat; a pokud jste šikovní a trpěliví, můžete se tímto způsobem dovědět mnoho o lokálních vlastnostech některých částí programu. Ale co vám to řekne o celku? Přečíst řekněme dvě stě tisíc fortranských zdrojových řádků a vytvořit si na jejich základě konzistentní a pokud možno úplný obraz o fungování programového systému, to nedokáže nikdo. To je prostě úloha nad lidské síly, úloha, se kterou se může vypořádat zas jen počítač. Díky specifice počítačů a díky specifice týmové práce jsme nabyli schopnosti *vytvářet mnohem složitější programové celky, než jaké dokážeme pochopit*. Jakmile počet vnitřních pravidel a zákonitostí přesáhne určitou mez, přestává být programátor suverénním vládcem; vytvořený svět se začíná řídit vlastními zákony, začíná si žít vlastním životem. Ze stvořitele a šéfa se stává udivený a zmatený divák.

Není to mnoho povyku pro nic? Vždyť od toho máme počítače, aby za nás řešily úlohy: ať je tedy řeší, ať to dělají, jak je jim libo, nás zajímají výsledky. Desítky let si přece vtloukáme do hlavy, že dobrý program se musí tvářit jako černá skříňka, že uživatel musí být odstíněn od implementačních detailů, aby se mohl soustředit na svou vlastní problematiku. Každý své ctí dbalý informatik vám vlídně vysvětlí, že to jinak nejde, pokud mají být počítače pro společnost přínosem: nebudete přece paní z místenkové pokladny na Hlavním nádraží nutit, aby se na stará kolena učila dvojkovou soustavu a ASCII kód.

Podle Josepha Weizenbauma [6] je hlavní důvod k obavám tento: programy, které používáme, jsou v podstatě dvojího druhu. Ty první vycházejí z určité teorie - dá se říci, že jsou jejím odrazem či modelem. Ty druhé jsou heuristické a s žádnou uzavřenou teorií nesouvisejí. Důležité je, že programů prvního druhu je překvapivě malý počet. Patří sem řekněme překladač Pascalu (pokud syntax a sémantiku Pascalu prohlásíme za teorii, což v dané souvislosti můžeme). Výsledky jeho činnosti lze konfrontovat s teorií a odhalit tak případnou chybu v překladači. Norma jazyka jasně definuje, jak má fungovat konstrukce if ... then ... else; funguje-li jakkoliv jinak, je v překladači chyba a jeho produkty jsou nepoužitelné.

U programů druhého druhu však podobná konfrontace není možná, neboť odpovídající teorie neexistuje - přinejmenším ne v takové podobě, aby poskytovala na jednoznačnou otázku jednoznačnou odpověď. Běžným, každému z nás blízkým příkladem je téměř libovolný operační systém; ale dá se sem zařadit většina rozsáhlejších softwarových projektů. Systémy řízení v reálném čase. Velké databáze. Expertní systémy a spolu s nimi téměř všechny aplikace umělé inteligence. Překvapivě velká část úloh zpracování hromadných dat. A ovšem problémy kosmické techniky a zbraňových systémů .Všechny tyto programy mají důležitou společnou vlastnost: *jsou jedinou existující definicí sama sebe.* Nejsou konzistentní s žádnou konkrétní teorií. Tím není řečeno, že by nepoužívaly výsledků nejrůznějších vědních disciplín; na rozdíl od programů prvního druhu však neexistuje - ani v principu - alternativní způsob získání výsledků. Programy druhého druhu nejsou reprezentací nějaké teorie, ale slepencem heuristických přístupů, izolovaných vědeckých poznatků a ad hoc pravidel, která to všechno lépe či hůře drží pohromadě. Nepleťte se: není řeč o programovacím stylu, ale o obsahu programu!

Docházíme tudíž k následujícímu závěru: systémy, které jsme nazvali programy druhého druhu (a víme, že k nim patří většina prakticky využívaných velkých programových celků), nemůžeme žádným způsobem účinně kontrolovat. Analýza jejich činnosti je prakticky nemožná. A srovnat jejich výsledky s teoreticky očekávanými hodnotami také nemůžeme, neboť neláme příslušnou teorii - to jsme si ukázali zde.

Tak vzniká ve společnosti přístup, který můžeme nazvat *počítačovým fatalismem.* I mezi vzdělanými a poučenými uživateli je často hlavní příčinou důvěry ve výsledky výpočtů ta skutečnost, že jsou tištěny tiskárnou počítače. Kdo ten počítač programoval, jak kvalitní program vznikl, jaké jsou meze jeho použitelností - to jsou otázky, které si neklade téměř nikdo. Současně se vytrácí pocit zodpovědnosti za rozhodnutí učiněná s počítačovou podporou. Už i v češtině zdomácněly obraty jako *počítač rozhodl*; ale co to, proboha, znamená? Nic jiného, než že nějaký vedoucí pracovník dobrovolně degradoval sám sebe do funkce pošťáka mezi výpočetním střediskem a výkonnými složkami. Mnoho lidí se domnívá, že se strojem nelze polemizovat, že závěry počítače jsou *a priori* dokonalejší a hlubší než úsudek člověka. To platí bez výhrad pouze u programů prvního druhu!

Nechce-li se tudíž uživatel - a v moderním světě jsme nebo budeme uživateli počítačů víceméně všichni - stát hříčkou v rukou anonymních tvůrců velkých systémů, musí vědět o existenci základních dvou druhů programů. Dostanete-li z počítače řešení soustavy lineárních algebraických rovnic, není celkem o čem diskutovat: je to jednoznačná úloha s ověřitelným řešením, starostí lidí od počítače je zajistit, aby program fungoval bezchybně - a co je důležité, je v jejich silách to zajistit. Pokud však dostanete - vytištěný touž tiskárnou a na témže sugestivním perforovaném papíru - třeba marketingovou expertizu, odhad volebních výsledků anebo literární rešerši na předem dané téma, pak je třeba zbystřit pozornost.

Jak to spolu všechno souvisí

Na začátku jsem prohlásil, že chci hovořit o různých aspektech jediného problému. Ptáte se jistě, kde ten jediný problém vidím, jakou spojitost nalézám ve výše uvedených útržkovitých poznámkách. Pokusím se o odpověď, přestože ji musím formulovat spíše intuitivně než exaktně. Zde je: zdá se mi, že všechny výše uvedené problémy, od backerů po počítačový fatalismus, nejsou marginálními obtížemi růstu výpočetní techniky, ale naopak jeho zásadní a integrální složkou. Že to, s čím se setkáváme, je dáno podstatou počítačů a podstatou lidské povahy, že z logiky věcí přímo vyplývá vznik podobných problémů.

Zpětná intelektuální stimulace od počítačů směrem k lidem je příliš intenzivní, než aby ji bylo lze zanedbat. Technický, inženýrský přístup k věci si láme hlavu nad tím, co mohou a nemohou dělat lidé a počítači. Už brzy by měl ale vážně někdo začít studovat problém, co dělají počítače s lidmi.

Co tedy dnes dělá opravdový programátor?

Všechno možné. Především nijak důsledně nelpí na tezích Postova článku. Mnozí opravdoví programátoři opustili výpočetní střediska, IBM/370 i OS; mnozí, i když zdaleka ne všichni. Ti kteří tak učinili, se většinou naučili jazyk C a s radostí zaznamenali, že se v něm dají dělat ještě daleko zamotanější hádanky než ve FORTRANu. Po překonání počátečního odporu si zvykli na péčka, neboť v nich správně rozpoznali jen trochu zdokonalený děrovač štítků.

Mnoho opravdových programátorů se již ekonomicky postavilo na vlastní nohy, nebo to hodlají učinit v nejbližší době. Soukromé podnikání se zdá být jejich světu blízké, neboť dává člověku značnou autonomii; ale jen málokdo z nich se projeví jako dobrý obchodník. Jiným plně vyhovuje postavení námezdní síly bez jiných starostí kromě assemblerských; ale kde dneska takové místo najít?

Pro opravdového programátora je dále charakteristické to, že nezakrývá opovržení nad starostmi, které jsem se v tomto článku pokusil zveřejnit. Jejich obvyklá reakce na podobné teze je *a)* je to všechno nesmysl, *b)* kdo z počítačů zblbnout nechce, ten z nich nezblbne, *c)* ten chlap jakýkov počítač neviděl, a když, tak nějaké pošahané péčéčko (to ale protestuji! Na OS/VS1 jsem sice už skončil, a přiznám se, že rád, ale syntax řídicích štítků pro utilitu IEHMOVE si ještě pamatuju. A pro SUPERZAP taky. -P.K.), *d)* nemám čas, nechodí mi program.

Přátelé opravdoví programátoři, já vás chápu. Ale zkuste se na věc podívat zvenčí. Nemusíte se mnou přece souhlasit. Stačí, když si uvědomíte, že jsou lidé, kteří tyto zdánlivé malichernosti považují za problém, kteří uvažují o počítačích v kategoriích jak kladných, tak záporných důsledků. Podobně jako jiní o automobilu, o jaderné energetice nebo o insekticidech.

Pokusme se o závěr. Opravdoví programátoři dnes, stejně jako v minulosti, představují jeden pól oboru. Na tom druhém - o tom jsme vůbec nehovořili - je matematická informatika se svými B-stromy, nekonečnými automaty a Turingovými stroji. Uprostřed mezi oběma póly je hustá džungle počítačového a softwarového průmyslu: žádní vědci, prakticky žádní opravdoví programátoři, ale obchodníci, marketingoví experti, lidi od reklamy, konzultanti, větší a menší šéfové. Víceméně čistý show-business. Pokud se má softwarové inženýrství jako praxeologická disciplína (tedy nikoli taková, která lidem předepisuje, jak to mají dělat, ale taková, která popisuje, jak to dělají ve skutečnosti) vyvíjet vpřed, musí se snažit obsáhnout celé toto rozpětí, od vědy přes kšeft až k počítačovým šílencům, které v podstatě nezajímá ani jedno, ani druhé. Všude se lze naučit něco užitečného.

Literatura

[1] Post E.: Opravdoví programátoři neužívají Pascal. Konference SOFSEM'84, teze pro panelovou diskusi (překlad, původní zdroj neuveden). Otištěno též v: Mikrobáze 5/88, str. 2-4.

[2] Koubský P.: Cesty moderního programování. JZD AK Slušovice 1990, str. 200-204.

[3] Čapek K.: Spisy, svazek VIII. Československý spisovatel Praha 1985, str. 324.

[4] Lamačka P.: Počítačové viry a iné druhy infiltrácií. JZD AK Slušovice 1989, str. 12-14.

[5] Jako 2, str. 330-334.

[6] Weizenbaum J: Computer Power and Human Reason. Ruský překlad: Vozmožnosti vyčislitel'nych mašin i čelovečeskij razum. Radio i svjaz' Moskva 1982.