Ing.Ivan Herkeľ
VÚD Žilina

O MOŽNOSTI KVALITATÍVNEHO HODNOTENIA PROGRAMOV CEZ KVANTITATÍVNE VELIČINY

ÚVOD

Pri posudzovaní procesu programovania musíme vziať do úvahy, že existuje viac kvalitatívne odlišných foriem a prostredí, v ktorých sa program môže nachádzať. Podstata je pritom jedna a tá istá - je ňou algoritmus istej zložitosti. Hoci je podstata pri prechode rôznymi stavmi taká istá, súčasný stav teórie ani praxe neumožňuje vyvodiť vlastnosti všetkých stavov z tejto jedinej podstaty. Preto treba každú vrstvu, v ktorej sa môže program nachádzať, skúmať zvlášť a [illegible] aj výsledky experimentálne overovať. V ďalšom texte by som sa zmienil o niektorých otázkach procesu tvorby programov v týchto vrstvách:

- tvorba programu individuálnym programátorom,
- tvorba systému programov,
- tvorba programov organizáciou.

PRIRODZENÉ VLASTNOSTI TVORBY PROGRAMOV

Programy sú istým zápisom algoritmov a ich tvorba má charakter manipulácie so symbolmi. Z praktického hľadiska nás zaujímajú tieto otázky:

1. ktoré veličiny zmerané z konkrétnych programov máme zvoliť za základné,
2. ak máme zvolené základné stavové veličiny, potom aké sú vzťahy pre výpočet ďalších veličín z týchto základných.

Analógiou s termodynamikou sa navrhlo a experimentálne overovalo, že program má tieto dve nezávislé základné stavové veličiny:

- počet jedinečných operátorov použitých v programe,
- počet jedinečných operandov použitých v programe.

Z týchto sa potom vyčleňuje ešte jedna špecifická podskupina, a to počet jedinečných vstupno - výstupných operátorov a operandov v programe.

Na začiatku programovania je program v tvare "čiernej skrinky" / Black Box - BB /, ktorá je pripojená k okoliu prostredníctvom vstupno - výstupných parametrov, pričom operátory sú iba dva, a to operátor "BB" a operátor pripojenia vstupno - výstupného parametra. Nahradzovaním operátorov programami sa nám program vo "vyššom" jazyku transformuje do programu v "nižšom" jazyku, až dospejeme k zdrojovému textu, ktorý bude mať N_1 celkových použití, n_1 jedinečných operátorov, N_2 celkových použití a n_2 jedinečných operandov, spolu N celkových použití a n jedinečných veličín. K problému určenia hodnoty N z hodnôt n_1 a n_2 sa môžme postaviť kombinatoricky, vypočítaním strednej hodnoty N pre všetky možné programy, ktoré môžme dostať z n_1 jedinečných operátorov a n_2 jedinečných operandov /za určitých obmedzujúcich predpokladov/. Avšak už pre malé hodnoty n_1, n_2 je stredná kombinatorická hodnota N tak veľká, že je zrejmé, že návrh programu s využitím tejto kombinatorickej variety prebieha iba pri najmenších základných úsekoch programov. Určitou dedukciou možno odvodiť nasledujúci vzťah pre celkový počet použití operátorov a operandov:

$$N = n_1 . \lg n_1 + n_2 . \lg n_2 \quad ,$$

kde lg je logaritmus pri základe 2. Tento vzťah je výsledkom štruktúrneho návrhu programu a treba ho brať v štatistickom zmysle, ako základnú tendenciu.

V tých prípadoch, kedy už máme pred sebou hotový program, / tak, že z neho môžme určiť skutočný celkový počet jedinečných operátorov a operandov N_{sk}, počet jedinečných operátorov a operandov n_1 a n_2, počet vstupno - výstupných jedinečných operandov n_2'/ sme postavení pred problém takého ohodnotenia, ktoré by nám umožnilo kvantitatívne ohodnotiť isté vnútorné kvality. Podkladom pre takéto ohodnotenie sú ďalšie dve odvodené veličiny:

- objem programu V
- energia programu E

Objem programu definujeme ako ohodnotenie aproximácie výberového procesu pri tvorbe programu binárnym prehľadávaním N nenáhodných výberov zo zoznamu n položiek. Teda:

$$V = N_{sk}.\lg /n_1 + n_2/$$

Pre počiatočný tvar programu v tvare "BB" si môžme tiež definovať počiatočný objem V´ ako:

$$V' = /2 + n_2'/ .\lg /2 + n_2'/$$

Energiu programu /danú jeho charakterom, ako manipulácie so symbolmi/ definujeme úsilím daným celkovým počtom elementárnych rozlíšení potrebných na zostavenie programu a aproximujeme množstvom všetkých možných binárnych porovnaní všetkých možných výberov /daných objemom V/ medzi sebou, pričom túto veličinu normujeme úsilím na vytvorenie počiatočného "BB" tvaru. Teda:

$$E_{min.} = \frac{V^2}{V'}$$

Táto energia sa musí dodať nepretržitým uvoľňovaním mentálnych rozlíšení programátorom a platí:

$$E_{men} \geq E_{min.}, \qquad E_{men} = S.T$$

kde T je doba programovania a S je tzv. Strud-ovo číslo, o ktorom platí, že $5 \leq S \leq 20$ za sek.

Na vyjadrenie ťažkostí pri tvorbe programu nám slúži ďalšia odvodená veličina, ktorou je úroveň programu L. Úroveň programu je nepriamo úmerná ťažkostiam pri tvorbe programu. Je definovaná koeficientom expanzie programu z počiatočného tvaru o objeme V´ do konečného tvaru o objeme V, čo znamená expanziu energie z hodnoty E´ /na vytvorenie objemu V´/ na hodnotu E /na vytvorenie objemu V/. Teda:

$$L = \frac{V'}{V}$$

Isté maximálne úsilie na vytvorenie programu o objeme V z počiatočného tvaru o objeme V´ si môžme aproximovať binárnym porovnávaním všetkých možných výberov v počiatočnom tvare a všetkých možných výberov v konečnom tvare, teda V.V´; pomer tohoto maximálneho úsilia k skutočnému, danému energiou

E, nazývame jazykovou úrovňou programu J, teda:

$$J = \frac{V.V'}{E} = \frac{V'^2}{V}$$

Hodnota J sa ukázala pozoruhodne konzervatívnou vzhľadom na rôzne veľkosti programov, a blízka hodnote 1 pre väčšinu terajších programovacích jazykov:

assembler CDC	J = 0,88
Fortran	J = 1,14
Algol 68	J = 1,21
PL/I	J = 1,53
technická angličtina	J = 2,16

Samozrejme, že tieto hodnoty sú priemerné a orientačné. Pri uvoľňovaní mentálnych rozlíšení jednorázovým aktom bolo zistené, že je možné v priemere uvolniť iba istý počet rozlíšení bez toho, abysme urobili chybu. Priemerný počet chýb B vnesených do programu objemu V manipuláciou so symbolmi by mal byť rovný pomeru objemu V programu a istého objemu V_1', ktorý sme schopní urobiť vnesením iba jednej chyby. Na druhej strane, pri návrhu počiatočného tvaru "BB", by mal byť počet chýb úmerný $\frac{V}{E_0}$, kde E_0 bolo experimentálne odhadnuté číslom 3000. Teda?

$$B = \frac{V}{E_0} = \frac{V}{3000}$$

Ak ako "bezchybný" z hľadiska symbolickej manipulácie definujeme program, pre ktorý je $B \leq 0,5$ potom pre "bezchybné" programy platí $V \leq 1500$. Tomu zhruba odpovedajú programy s počtom jedinečných operátorov a operandov menším ako 56 a celkovým N menším ako 250. Ak však dopredu nevieme úplne nič o budúcich vytvorených programoch chceme určite vytvoriť bezchybný program, potom môžme zobrať za základ odhadu kombinatorickú strednú hodnotu N. Potom sú napr. tieto možné hodnoty n_1 a n_2:

$n_1 = 1;2$	$n_2 = 2$ až 8
$n_1 = 3;4;5$	$n_2 = 2$ až 7
$n_1 = 6;7$	$n_2 = 6$ až 8

Ak teda chceme vytvoriť bezchybný konečný program, musíme

ho z počiatočného tvaru "BB" rozvíjať tak, aby každý čiastkový operátor bol rozvinutý do bezchybného programu.
V predchádzajúcom texte uvedené vzťahy však možno aplikovať aj na iné typy schém, ako programové texty, napr. rádiotechnické schémy, vývojové diagramy, systémové schémy atď. Môžme ich zobrať za základ popisu tvorby schém individuálnym tvorcom, pričom sa jedná o jednu celistvú schému. Vzťahy sú štatistickej povahy a mali by byť v konkrétnych podmienkach otestované.

VÝVOJ SYSTÉMU PROGRAMOV

Vývoj veľkého software systému sa deje zložitým procesom tvorby jeho jednotlivých častí v čase. V priebehu vývoja sa jeho jednotlivé moduly rozširujú, niektoré zanikajú iné vznikajú. Programátori prichádzajú a odchádzajú. Preto sa na tvorbu rozsiahleho software systemu musíme pozerať skôr na makrosystém. V literatúre /2/ je podaná pomerne uspokojivá teória, experimentálne overená. Jej základom je Norden-Rayleigh model vývoja rôznych častí projektov v čase. Jeho podstata môže byť vyjadrená vetou - rýchlosť dokončovania práce je úmerná pracovnej časovej jednotke /krok/ a množstvu práce, ktorá ešte zostáva k urobeniu. To vedie k určitej diferenciálnej rovnici, v ktorej vstupné parametre sú:

t_d - vývojový čas t.j. čas, za ktorý sa dosiahne plná operačná schopnosť systému, a ktorý /empiricky dokázané/ odpovedá času, v ktorom sa dosiahne maximálne úsilie vo vývoji

K - celkové úsilie v človekorokoch, potrebné na prevedenie celého cyklu života systému

C - konštanta, miera stavu technológie, mení sa skokom

Vzniknutú diferenciálnu rovnicu je možné riešiť numericky a v priebehu riesenia meniť parametre, a tak odhadovať vplyv zmeny podmienok riešenia na budúce možnosti vývoja. Pre dôveryhodne použitie však v konkrétnych podmienkach

treba empiricky odhadnúť parametre K a t_d. V literatúre /2/ je tiež ukázaný spôsob, ako zo štatistických nezávislých premenných /počet file, počet správ, počet aplikačných subprogramov/ získať spoľahlivé odhady K a t_d. Riešením diferenciálnej rovnice a jej integrovaním je možné odvodiť nasledujúci pozoruhodný vzťah:

$$S_c = C \cdot K^{\frac{1}{3}} \cdot t_d^{\frac{4}{3}}$$

kde S_c je počet zdrojových procedúr vytvorených počas vývoja programu. Pri experimentálnom overovaní bolo zistené, že hodnota výrazu $\frac{K}{t_d^2}$ bola malá, ak boli systémy ľahké k riešeniu, a bola t_d veľká, ak boli systémy ťažké k riešeniu v termínoch programátorského úsilia a čase k jeho produkovaniu. Tiež sa empiricky zistilo, že gradient veličiny $\frac{K}{t_d^2}$ je určený výlučne veličinou $\frac{K}{t_d^3}$ a je konštantný v širokých medziach, t_d pričom konštanta je určená statusom systému /t.j. či je úplne nový, prebudovávaný, zložený a pod./. Výraz $\frac{K}{t_d^2}$ bol nazvaný ťažkosť systému k riešeniu D. Význam gradientu t_d je tento, čiara konštantného gradientu v súradniciach S, t_d nám určuje spodnú hranicu pre možné hodnoty K, t_d, S_c. Metodologický zmysel horeuvedených vzťahov je tento:

- produktivita práce pre rôzne projekty bude tá istá, ak ťažkosť D bude tá istá
- ak určitý projekt potrebuje určité množstvo výstupného produktu /S_c/, potom čas na vývoj t_d je stlačiteľný iba k určitej hodnote /danej gradientom/, ináč sa systém stáva neúnosne ťažkým k riešeniu /pre zvyšujúcu sa cenu ľudí, zvyšujúcu sa zložitosť vzájomnej komunikácie medzi väčším počtom ľudí, a pod./
- potreba ľudí /človekoroky/ nie je a nemôže byť v priebehu vývoja konštantná.

Z hľadiska nákladov na vývoj to potom vyzerá nasledovne:

náklady na vývoj systému = 0,4.K.cena človekoroku

Okrem praktických možností využitia v predchádzajúcom stručne uvedenej teórie z nej vyplýva dôležitý heuristický záver:

čas vo vývoji software systému nie je voľnou veličinou, ale veľmi nákladnou položkou. Zdola je čas vývoja pre veľké systémy obmedzený hodnotou 2 roky. O obmedzeniach shora sa zmienim v nasledujúcich odstavcoch.

ZROVNANIE TU UVEDENÝCH EMPIRICKÝCH TEÓRIÍ

Spojením vzťahov z oboch teórií môžme odvodiť pre úroveň programu L z 1.teórie vzťah:

$$L = \text{konštanta} \cdot \frac{1}{D^{\frac{3}{2}}}$$

kde D je veličina z druhej teórie majúca význam ťažkosti, čo odpovedá určeniu významu L v 1.teórii. To opravňuje k domnienke, že medzi nimi existuje rozumný vzťah, z ktorého možno vyťažiť metodologické poznatky. Nech určitý problém si vyžaduje určitú veľkosť konečného zdrojového kódu. Zistime si pomer $E_{symb.}$ ku $E_{ment.}$, spojením oboch teórií za predpokladu, že premenná bude iba t_d. Potom môžme odvodiť:

$$\frac{E_{symb.}}{E_{ment.}} = \text{konštanta} \cdot t_d^{\frac{5}{3}}$$

kde konštanta je v podstate závislá od stavu technológie programovania a od psychických schopností programátora. Vidíme, že s rastom t_d sa pomer medzi energiou nutnou k symbolickej manipulácii a energiou uvoľňovanou programátorom zväčšuje. Aj keď v skutočnosti je hodnota symbolickej energie nižšia oproti teoretickej, vďaka rôznym obmedzeniam na programovací proces, nesporný trend nerovnosti je zrejmý. To má tento metodologický význam:
pri príliš dlhej dobe vývoja vzniká v projekte nedostatok mentálnej energie, avšak keďže množstvo kódu v konečnom dôsledku musí byť a je naprogramované, energetický deficit ide na úkor prudkého zvýšenia intenzity mentálnej činnosti / v istom zmysle však fiktívnej/, čo sa nutne musí prejaviť na kvalite vyvinutého programového systému /vznikajú chyby naviac k hodnote B, ktorá je prirodzenou/.

Z tohoto hľadiska by tvorba software systémov, ktorých veľkosť je nad hranicou danou stavom technológie /techniky, programovací jazyky, spôsob programovania, organizácia práca atď./, vôbec nemala byť pripustená.

DYNAMIKA VÝVOJA SOFTWARE V ORGANIZÁCII

Software je vyvíjaný v organizácii, ktorá sama má určitú štruktúru a je systémom. Programovací proces, ktorý v nej prebieha, má správanie samostabilizujúceho sa systému so spätnou väzbou, ktorý je rušený zo strany vonkajších požiadaviek. Organizácia má tendenciu programovať s konštantnou rýchlosťou. Pokus zvýšiť rýchlosť nad hodnotu danú vnútornou kvalitou organizácie vyvoláva samovoľnú reakciu s tendenciou vrátiť rýchlosť na pôvodnú hodnotu. V procese práce organizácie na rôznych systémoch dochádza k rôznej koncentrácii úsilia na ich rôzne časti. V organizácii, v ktorej je vývoj software temer úplne neriadený, je práca organizácie rovnomerne a nezávisle rozložená medzi všetky časti vyvíjaných systémov. Stupeň koncentrácie úsilia na jednotlivé časti vyvíjaných systémov môže byť takto mierou kvality práce organizácie. Vyvíjaný systém má prirodzenú tendenciu zvyšovať svoju neurčitosť /entropiu/ v čase. V priebehu jeho života sa stáva stále ťažším ho modifikovať, pridávať k nemu nové funkcie a vôbec meniť ho. Software systém má teda svoj vlastný život:

- zrodenie
- obdobie dospelosti
- úmrtie

Aby tento proces bol riadený, organizácia musí cieľavedome vynakladať úsilie na znižovanie entropie software systému /e seba samej/, ktorý je jej produktom. Z toho je zrejmé, že je žiaduce, aby doba vývoja systému bola menšou časťou celkového žitia systému. Vzhľadom na rýchle zmeny technológie dĺžka života software nieje príliš veľká a je skutočne žiaduce, aby vývoj nebol dlhší, ako predpokladaný život sys-

tému. Bližšie viď literatúre /3/.

ROZDIEL MEDZI MALÝMI A VEĽKÝMI SOFTWARE SYSTÉMAMI A MALÝMI A VEĽKÝMI ORGANIZÁCIAMI PRODUKUJÚCIMI SOFTWARE

Pod veľkým systémom rozumieme zhruba systém , ktorého vývoj trvá viac ako 2 roky pri celkovom úsilí prevyšujúcom 5 až 10 človekorokov. Pri malých systémoch vzniká jedno nebezpečie, a to, že programovací proces /ako sa nám javí cez štatistické veličiny/ je prekrytý "šumom", a je ťažké vystihnúť v ich vývoji určujúcu tendenciu a posúdiť ich kvalitu. To isté možno v istom zmysle povedať aj o malých organizáciách vyvíjajúcich software. Okrem toho je pre nich neúnosné vyvíjať prostriedky pre odhad a riadenie vlastného programovacieho procesu, je neúnosné dvihať stav technológie /čo jediné zvyšuje veľkosť systémov, ktoré je únosné riešiť/, nemôžu dosiahnúť vhodné rozdelenie práce programátorov. Preto je žiaduce, aby podstatná časť produkcie software bola zústredená do menšieho počtu väčších organizácií produkujúcich software na ekonomickom základe.

Na záver by som rád podčiarkol, že v článku je pod pojmom programátor myslený "priemerný" programátor t.j. programátor v štatistickom zmysle. Preto hlavná starostlivosť vo vývoji software systémov musí byť venovaná tomuto všednému a často nezaslúžene obchádzanému programátorovi.

Použitá literatúra:

1. Maurice H. Halstead: Technical Reports N° 66,67,69,72,190, 229; Purdue University
2. Lawrence H. Putnam : A General Empirical Solution to the Macro Software Sizing and Estimating Problem; IEEE TRANS. ON SOFT. ENG. - 1978
3. M.M. Lehman, F.N. Parr: Program Evolution and its Impact on Software Engineering; Imperial College; London - 1976