Ing. Stanislav Lacko

TOS Kuřim

SPOLEHLIVOST PROGRAMŮ

1. Úvod

Prudký vzrůst počtu počítačů přinesl s sebou mimo jiné i nutnost, řešit problémy, spojené se spolehlivostí programů. Nesprávné výsledky z počítačů mají často katastrofální důsledky. Kromě primárních škod se dostavuje navíc ztráta důvěry širší veřejnosti v prospěšnost a možnosti počítačů. Pouhé vyprávění komických situací, které počítače způsobily chybnými výpočty, je nahrazováno postupně vysloveně negativním postojem k zavádění výpočetní techniky do praktického života.

Bylo provedeno několik průzkumů [1], které analyzovaly příčiny chybných výpočtů. Ukázalo se, že většina chyb byla způsobena špatnými programy. Téměř 65 % chyb. Teprve zbývající část zahrnuje technické poruchy a jiné příčiny. Tato skutečnost vrhá kritické světlo na metody vypracovávání programů a posléze na programátory samotné. Vzniklé situace je o to svízelnější, že pro řešení otázek spolehlivosti programů nelze použít běžný aparát klasické teorie spolehlivosti, jak jej známe z technické praxe.

2. Spolehlivost programu

Spolehlivost programu je charakterizována souladem mezi předpokládaným a skutečným chováním programu. Tuto vlastnost je možno vyjádřit dvojím způsobem:

a) Jako pravděpodobnostní veličinou S_{pr}, která udává, jakou máme naději, že výpočet proběhne správně. Je nutno přiznat, že dosud nebyly stanoveny metody, které by analytickými výpočty určily přesnou hodnotu veličiny S_{pr}.
Lze uvažovat, že v budoucnosti budou navrženy prognostické po-

stupy, na jejichž základě budeme moci spolehlivost sestaveného programu předpovědět s určitou mírou nejistoty.

b/ Jako statistickou veličinu S_{st}, která vyhodnocuje dosud provedené výpočty

$$(1) \quad S_{st} = \frac{K}{N} = \frac{\text{počet případů, kdy výpočet proběhl správně}}{\text{celkový počet výpočtů provedených programem}}$$

nebo

$$(2) \quad S_{st} = 1 - \frac{K}{N} = 1 - \frac{\text{počet případů, kdy výpočet proběhl nesprávně}}{\text{celkový počet výpočtů provedených programem}}$$

Za správný výpočet považujeme případ, kdy průběh výpočtu, trvání výpočtu, spotřeba operační paměti a jiných technických zdrojů počítače /periferní zařízení, sekundární paměti, virtuální paměť/ jsou v souladu s předpoklady a obdržené výsledky jsou numericky, formálně i obsahově správné.

Jestliže chování programu nebo jeho výsledky vykazují odchylky od předpokládaných stavů a správných výsledků, došlo při přípravě programu nebo při jeho provádění k jedné chybě nebo více chybám. Chyba je tedy příčina, která způsobuje nesoulad mezi předpokládaným správným a konkrétním chybným průběhem výpočtu. Podle toho, v důsledku jakého jevu chyba vznikla, můžeme rozeznávat různé skupiny chyb a druhy spolehlivosti.

Je samozřejmé, že pro stanovení hodnoty S_{st} musíme zajistit, aby celkový počet výpočtů provedených programů - N byl dostatečně velký.

V souvislosti se zde citovanou tematikou se používá termínu správnost programu. Dále budeme tento termín používat ve smyslu logické správnosti naprogramovaného výpočetního algoritmu. Správnost programu je nutná podmínka pro to, aby program byl spolehlivý. Není však postačující podmínkou. Spolehlivost programu, jako pojem, uvažuje více možností a příčin, které mohou přivodit špatný výsledek

a průběh výpočtu. Zdá se, že omezené použití metod automatického dokazování správnosti programu /viz dále/ je nutno hledat také ve skutečnosti, že vypracované metody provádějí jen důkaz správnosti výpočetního algoritmu.

2.1 Elementární spolehlivost

Programy jsou realizovány prostřednictvím technického vybavení počítače. Jestliže se na některé mechanické součástce nebo elektrickém obvodu vyskytne porucha, může to mít za následek chybnou funkci programu, není-li technicky zjištěno, že se úkon opakuje tak dlouho, až porucha zmizí. Elementární spolehlivost je závislá na spolehlivosti technického zařízení počítače.

2.2 Funkční spolehlivost

Program je vytvářen za tím účelem, aby zajistil na počítači provedení vybrané posloupnosti elementárních operací určitého algoritmu. Jestliže je algoritmus špatně funkčně sestaven, pak i na jinak bezporuchovém počítači získáme špatný výsledek. Funkční spolehlivost ovlivňují všechny logické chyby v programu.

Provedení výpočtového algoritmu je u současných počítačů zajišťováno prostřednictvím programovacích jazyků, které využívají operační systém počítače. Jestliže kompilátor nebo řídící programy operačního systému obsahují chyby, může v důsledku nich i formálně správně naprogramovaný algoritmus ve vyšším programovacím jazyku dávat nesprávné výsledky.

Řada chyb vzniká při výpočtu v důsledku strojového zobrazování čísel v počítači /přeplnění, ztráta přesnosti atd./. Například výrazy (1/2 + A) a (0.5 + A) nemusí být v počítači reprezentovány shodným výsledkem. Jestliže se na tyto skutečnosti v programu nepamatuje, stává se program velmi nespolehlivým.

Z výše uvedeného přehledu vyplývá, že funkční spole-

hlivost můžeme dále rozlišit na:

a/ správnost algoritmu
b/ spolehlivost systémových programů
c/ spolehlivost strojového zobrazení čísel a jiných objektů v programu.

2.3 Manipulační spolehlivost

Správně sestavený algoritmus dá špatné výsledky, je-li nesprávně použit. Do této skupiny poruch patří případy, kdy se např. vyvolá nesprávný program nebo podprogram, obsluha vloží jinou verzi datových souborů nebo programových knihoven, zadají se programu nebo podprogramu nevhodné parametry atd. Jak roste složitost realizace výpočtů na počítači, začíná tento druh spolehlivosti nabývat na významu. Nejznámějším případem manipulačních chyb jsou případy, kdy aplikujeme algoritmus na vstupní data, která obsahují chyby.

V souvislosti s manipulačními chybami je vhodné poznamenat, že mohou nastat v důsledku omylu nebo také tím, že uživatel je k chybnému použití veden nedostatečnou, někdy vysloveně chybnou, dokumentací naprogramovaného algoritmu.

2.4 Implementační spolehlivost

V praxi se často vyskytují případy, kdy se při reklamaci chybného výsledku zjistí, že došlo k chybné interpretaci uživatelova požadavku analytikem, nebo že programátor jinak pochopil analytikovo zadání programu /případně obojí/. Implementační spolehlivost postihuje skutečnost, zda uživatel dostal výsledky, jaké požadoval.

2.5 Komplexní spolehlivost

Při provádění výpočtů se čtyři vyjmenované skupiny chyb vzájemně prolínají. Máme-li posoudit pravděpodobnost správného výsledku výpočtu, musíme vzít v úvahu komplexní charakter spolehlivosti vlivem všech okruhů chyb.

Rozsáhlé výpočty se dnes realizují prostřednictvím více programů. Proto při určování spolehlivosti konečného

výsledku musíme vzít v úvahu spolehlivost všech programů použitých při výpočtu.

Pokud nijak pojmově nezdůrazníme, o který druh spolehlivosti se jedná, pak obecně vždy uvažujeme komplexní spolehlivost programu. To platí i pro vztahy (1) (2) .

2.6 Relativní nespolehlivost

Při rutinním používání programu může uživatel za chybný výsledek označit takový, který odporuje uživatelovým předpokladům. Hledá se chyba v počítači, programu, analýze, postupu při výpočtu, ve vstupních datech. Nakonec se ukáže, že výsledek je správný, ale uživatelovy předpoklady byly mylné a soud o výsledku nepravdivý.

Než-li se označí určitý výsledek za chybný, mělo by se pečlivě prověřit, zda podklady, ze kterých se vychází, a kritéria, která byla použita, jsou správná.

3. Hodnota spolehlivosti programu

Jak bylo dříve uvedeno, zatím nejsou k dispozici metody pro určení hodnoty spolehlivosti. Můžeme však vyslovit některé úsudky o mezních hodnotách spolehlivosti. Následující úvahy platí jak pro veličiny S_{pr}, tak i pro veličiny S_{et}.

U technického zařízení počítače je vždy určitá možnost výskytu poruchy.

Nejsou ještě dnes prakticky dostupné metody, které by dokázaly, že naprogramovaný algoritmus je úplně správný. Běžně používané testovací postupy mohou prokázat existenci určitých chyb, ale nemohou dokázat jejich nepřítomnost.

Při implementaci programů se nepodařilo vyvinout metodu, prostřednictvím níž bychom zabránili nepřesnostem při vzájemné komunikaci mezi uživatelem, analytikem a programátorem.

Realizace výpočtu na počítači vyžaduje vždy přítomnost člověka. U něj je nutno připustit možnost omylu, zapomnětlivosti, špatného pochopení apod.

Uvážíme-li všechny tyto skutečnosti, pak pro veličinu S /komplexní spolehlivost programu/ platí:

$$0 \leq S < 1 \qquad (3)$$

Absolutně spolehlivý program, pro nějž by hodnota spolehlivosti nabyla hodnoty S = 1, nelze nikdy vypracovat.

Jednotlivé příkazy programu /v mnoha případech jsou kompilátorem takto realizovány/. O nich pak můžeme jednotlivě uvažovat jakou mají komplexní spolehlivost celého programu jako součin komplexní spolehlivosti jednotlivých příkazů, které se zúčastní výpočtu v konkrétním případě:

$$S_{prog} = S_{PRIK_1} * S_{PRIK_2} \cdots\cdots * S_{PRIK_L} \qquad 4$$

Protože pro komplexní spolehlivosti příkazů platí:

$$0 \leq S_{PRIK_i} < 1$$

bude zřejmě S_{PROG} tím menší, čím větší bude hodnota L. Rozsáhlejší program má větší možnost výskytu chyb a tím má i menší spolehlivost.

4. Změna spolehlivosti v čase

Komplexní spolehlivost programu je veličina, která se mění počas životnosti programu. Její průběh můžeme znázornit křivkou na obr. 1. Na křivce můžeme pozorovat tři období:

I. období - "počáteční potíže". V tomto období se odstraňují nedostatky programátorské a analytické práce ve spolupráci s uživatelem. Křivka začíná v počátku souřadnic, protože pravděpodobnost, že by program o více než 250 řádcích dával správné výsledky hned napoprvé, je nulová. Toto údobí "dětských nemocí" by se mělo shodovat s dobou zkoušení a ověřování programu /resp. programového systému/. Předání uživateli do rutinního užívání v době kratší než t_1, může mít nepříjemné následky.

II. období - "provozní spolehlivost". Z důvodů, které byly uvedeny v odst. 3, vyskytují se i po důkladném testování programu čas od času chyby, které jsou průběžně opravovány, takže spolehlivost roste. Zdálo by se, že nic nebrání tomu, aby po dostatečně dlouhé době narostla tak, že by se program stal absolutně spolehlivým. Není to však pravda. Období provozní spolehlivosti končí /z důvodů uvedených níže/ v čase t_2. Doba $st = t_2 - t_1$, by měla být shodná s dobou praktického běžného užívání programu pro rutinní výpočty.

III. období - "znehodnocení programu". Závěrečná fáze životnosti programu je způsobena několika příčinami:

- postupným zvyšováním nespolehlivosti stárnoucího počítače
- zastaráváním dokumentace, ve které se neprovedly všechny změny prováděné ve II. období
- kladením požadavků na program, se kterými nebylo při návrhu uvažováno, a snahou přizpůsobit program těmto požadavkům.

Pro mnohé programátory je nepochopitelné, že III. období může nastat. Ale ono existuje! Přijde často ještě dříve, než je vyčerpána životnost technického zařízení počítače, a tím zvýšena jeho provozní nespolehlivost. Způsobuje to zejména poslední ze tří výše uvedených příčin.

Po určité době úspěšného používání programu se téměř vždycky přistoupí k realizaci požadavků, které navrhují:

- rozšíření funkcí programu

- optimalizaci jeho činnosti /rychlost výpočtu, obsazení paměti, využívání sekundárních pamětí/
- odstranění některých nepohodlných vlastností /zadávání vstupních dat nebo řídících parametrů, nepřehledně uspořádané výsledky/.

Realizace vznesených požadavků vždy způsobí výskyt nových chyb. Jejich počet záleží na způsobu sestavení programu, na složitosti programu, na rozsahu prováděných změn, na kvalitě dokumentace a zejména na tom, kdo program upravoval /zda autor či jiný programátor/. Je-li program málo rozsáhlý, přehledně sestavený, zásah jednoduchý a kvalitní dokumentace, můžeme zachytit vývoj počtu chyb diagramem na obr. 2a. Je-li naopak program sestaven bez přihlédnutí k pozdějším změnám, je rozsáhlý a složitý, bude průběh výskytu chyb podle diagramu na obr 2b.

Požadavky uživatelů jsou jak známo nevyčerpatelné. Mohou se proto vyskytnout další návrhy na změny, ke kterým se přistoupí, aniž byly dokonale odstraněny chyby z minulého zásahu. K chybám stávajícím se přidruží další chyby z následovného zásahu do programu. Chyby se vzájemně prolínají a je těžko je vůbec identifikovat. Další zásahy situaci jen zhoršují. Počet chyb lavinovitě roste /viz obr 2c/ a program se zakrátko stává nepoužitelným.

Obrázek č. 1 zachycuje situaci, kdy program byl dokonale navrhnut a ve II. období se prováděly jen drobné opravy prokazatelných chyb. To je zjevně idealizovaný předpoklad. Průběh skutečné spolehlivosti u celé řady běžných programů zachycuje obr. 3.

V určitém případě[1] byla ověřována spolehlivost modifikací rozsáhlých programů pro dávkové zpracování v oblasti vědeckotechnických výpočtů, a byl získán diagram na obr. 4. Na osu y byl zaznamenán procentový podíl úspěšných prvních chodů. Z diagramu je zejména patrné následující:
- pravděpodobnost, že první chod po zásahu do programu bude úspěšný, je asi 50 % i při malém zásahu.

- Při změně jednoho příkazu je pravděpodobnost prvního úspěšného chodu menší než při změně 5 příkazů. Je to způsobeno zřejmě tím, že programátoři optimisticky předpokládají, že změna jednoho příkazu nemůže způsobit chybu a zásah dokonale neprověří.
- Měli bychom se vyvarovat zásahu do programu ihned, jakmile požadavek vznikne. Lépe je kumulovat několik úprav do jednoho termínu. Rozsah úprav by však neměl přesáhnout rozumnou mez 10 - 20 příkazů na jeden zásah do programu. /To samozřejmě neplatí pro opravy prokazatelných chyb nebo zásadní přepracování programu./

5. Ověřování spolehlivosti programů

Prozatím se k tomuto účelu používá ve většině případů zkušebních výpočtů. Programu se předloží pro zpracování vstupní data úlohy, od které známe předem výsledek. Se známým výsledkem srovnáváme řešení, které vyprodukoval program. Je nutné si uvědomit, že uvedeným postupem nelze ověřit chování programu pro všechny kombinace vstupních dat a situací při výpočtu. Již pro dva celočíselné vstupy o 32 bitech by bylo nutno testovat 2^{64} možností. Jestliže by průběh testu trval miliontinu vteřiny, pak by celý test vyžadoval stovky bilionů roků. Testovaná data představují vždy jen malou část z celkového počtu možných variant vstupních dat. Proto tento způsob nemůže prokázat absolutní spolehlivost programu.

Protože při testování programu použijeme vždy neúplný soubor testů, můžeme výše popsaný způsob, označovaný pojmem "ladění programu", charakterizovat takto:

"Testováním zjišťujeme přítomnost určitých chyb. Zjištěné chyby opravujeme a podrobíme program opět testům. Testování ukončíme v okamžiku, kdy se domníváme, že spolehlivost programu je dostatečně vysoká pro rutinní provozování výpočtů. Na hodnotu spolehlivosti usuzujeme ze vztahů (1) resp. (2)."

K ověřování programu se někdy může použít metoda založená na ruční simulaci výpočtu /"ladění u stolu"/. Je však prakticky nezožné simulovat rozsáhlé výpočty, obsahující složité matematické funkce a velké počty proměnných. Navíc se netestuje činnost kompilátoru a operačního systému. Ruční simulování programu slouží proto jen jako pomocný prostředek pro jednoduché algoritmy, které nepřesahují možnosti a schopnosti programátora.

Perspektivnější způsob dokazování správnosti programů spočívá na formálních důkazech matematické indukce. Jeho výhodou je, že může být prováděn automaticky počítačem podle speciálně otestovaného programu. V současné době jsou tyto metody ve stavu intenzivního zkoumání. Zatím však jejich praktickému rozšíření brání malá efektivnost a těžkopádnost. Dokazování i zcela triviálních výpočtů vede k velkému počtu operací. Navíc, jak již bylo uvedeno, testují tyto metody pouze nutnou podmínku spolehlivosti, tj. správnost programu. Proto se věnuje pozornost metodám, které zvyšují účinnost dosavadního ověřování programů výpočtem.

Navrhované zdokonalení je možno seskupit do tří skupin.

a/Zlepšení výběru ověřovacích dat pro testovací výpočet. Analýzou programu lze vybrat soubor vstupních hodnot tak,aby bylo ověřeno co nejvíce příkazů testovaného programu. Podstatnou část takového postupu lze automatizovat, včetně generování souboru zkušebních dat. Vypracované systémy bývají zdokonalovány často tak, že do jednotlivých úseků programu jsou automaticky kompilátorem zařazena počítadla, která indikují po ukončeném výpočtu, kolikrát byl určitý úsek programu použit. Vytištěné informace lze navíc využít při rozboru práce programu a případnou optimalizaci. Viz obr. 5.

b/Využití intuice programátora při zkoušení programu. Tyto metody se snaží dát programátorovi k dispozici lepší možnosti jak sledovat průběh výpočtu, ovlivňovat výpočetní

proces a korigovat program na základě zjištěných skutečností. Navržené systémy využívají interaktivního ověřování programu metodou dialogu prostřednictvím přímo napojeného obrazovkového displeje. Vypracované systémy zajišťují vytváření zvláštních dat o průběhu výpočtu, která se ukládají na magnetickou pásku. Uložená data popisují chod výpočtu a programátor může jejich analýzou /prostřednictvím speciálních systémových programů/ získat potřebná fakta o funkci programu.

V souvislosti s technikami sledování průběhu výpočtu je nutno označit za nevyhovující mnohé dosud používané systémy u počítačů II. a III. generace. Ty interpretují jednotlivé instrukce přiloženého programu a vypisují v oktalovém, resp. hexadecimálním tvaru obsah čítače instrukcí, střadačů, paměťového registru atd. Programátor potom musí pracně podle vytištěných tabulek vyhledávat, jaká je vazba mezi symbolickými objekty ve zdrojovém textu programu a obsahem buněk operační paměti. Měla by platit zásada, že sledovací systémy musí umožnit sledovat programátorovi průběh výpočtu v symbolických názvech, které použil při psaní zdrojového textu programu. Uvedená zásada je např. plně realizována u počítače IV. generace ICL 2900.

Současné techniky sledování výpočtu vyžadují, aby byl program zkompilován zvláštním způsobem, jestliže má být prováděno sledování průběhu výpočtu. Běžně zkompilovaný program takové možnosti neposkytuje. Tímto kompromisem se vyřešil ožehavý problém efektivní práce programu. Program zkompilovaný pro možnost sledování výpočtu pracuje vždy pomaleji a zabírá větší část operační paměti. Předpokládá se, že jakmile programátor považuje program za schopný rutinního výpočtu, zkompiluje jej bez možností sledování výpočtu. Tím značně klesnou náklady na opakující se rutinní výpočty. Potíž však nastane, jestliže program vypočte špatné výsledky pro určitá data. Nejsou detailní podklady o průběhu výpočtu a program není uzpůsoben, aby dovolil takové údaje získat i při opakování výpočtu /těžko se někdy navo-

zují úplně stejné podmínky jako byly při vzniku chyby/, pokud by se nezkompiloval opět pro možnost sledování. Tak, jak se zvětšuje rychlost počítačů, se ukazuje, že bude pro budoucnost výhodnější pracovat i v rutinním provozu s programy, které sledování dovolují při každém výpočtu. To však má zpětně dopad na řešení sledovacích metod, které musí být vypracovány velmi efektivně.

c/Využití myšlenky prevence pro snížení počtu chyb. Jestliže nemůžeme chyby v programech vyloučit, můžeme aspoň udělat řadu opatření předem, která chybu zaregistrují a zabrání jí, aby negativně ovlivnila výpočet.

Má-li naprogramovaný algoritmus poskytovat správný výsledek, musí jednotlivé proměnné nabývat postupně určitých hodnot z určitého příslušného definičního oboru. Např. určitá proměnná nesmí být nikdy nulová, jiná nemůže nabýt záporné hodnoty atd. Jestliže program v průběhu výpočtu odhalí odchylku od předpokládaných hodnot, může signalizovat dosažení nežádoucí hodnoty proměnné a nepřipustit, aby výpočet proběhl až do konce a byl získán chybný výsledek.

Praktickou realizaci uvedené myšlenky lze zajistit sítí kontrol, které programátor umístí na významných místech programu. Objevily se však návrhy a pokusy, zdokonalit v tomto směru dosavadní programovací jazyky. Každá deklarace proměnné by mohla obsahovat nadbytečné informace např. integer a pozitive, b from 100 to 999, c ;
Počítač by mohl kontrolovat při každém přiřazení hodnoty deklarovaným proměnným během výpočtu, zda hodnota proměnné splňuje zadané omezení. Dosavadní kompilátory zajišťují signalizaci jen překročení maximální nebo minimální zobrazitelné hodnoty /a to ještě ne všechny/. Pouze jazyk PASCAL [13] dovoluje v deklaraci definovat skutečnost, že určitá proměnná smí nabývat jen přesně vymezených hodnot. Protože tatáž proměnná může být použita pro různé účely, je nutno pamatovat i na možnost dynamické změny atributů v průběhu výpočtu.

Myšlenku preventivní kontroly lze rozšířit i na zabezpečení povoleného pořadí dynamicky prováděných operací. Neměl by se např. vyskytovat dvakrát po sobě příkaz OPEN pro určitý soubor, pokud nebyl soubor uzavřen příkazem CLOSE apod. Chybné posloupnosti příkazů by měly být odhaleny a ohlášeny.

Na předcházení chybám je nutno pamatovat zejména u realizace kompilátorů jazyků vyšší úrovně. Z dokumentace kompilátorů /a samozřejmě z definic jazyků/ by měly vymizet výroky: "... pak funkce příkazu není určena ...", "... hodnota výrazu není definována ..." atd. Jedná se o případy, kdy např. hodnota přepínače je rovna nule nebo přesahuje počet definovaných návěští, velikost indexu překračuje mez pole atp. Tyto skutečnosti je potřeba v průběhu výpočtu hlásit a zaručit, že chyba nezpůsobí neidentifikovatelný chod programu.

6. Jak navrhovat spolehlivé programy

Programátor může podstatně ovlivnit spolehlivost programu. Každý programátor by si měl uvědomit tuto skutečnost a při návrhu i tvorbě programů si stále klást otázku: " Co mohu udělat pro zvýšení spolehlivosti tohoto programu?" Programátor by měl být spíš pesimista a předpokládat, že okolnosti se mohou zhoršovat neomezeně. Předpokládat, že vstupní data budou bez chyb a všechna dodána k počítači správně seřazená, že obsluha počítače při ručním zásahu do řízení programu neudělá chybu atd., není optimismus, ale lehkomyslnost.

Některé druhy chyb se tak často opakují, že má význam je samostatně analyzovat a naznačit způsoby jejich odstraňování s cílem, zvýšit spolehlivost programů. Opakovaný výskyt stejných chyb program od programu nutí k domněnce, že v našich výpočtových střediscích se věnuje pravděpodobně příliš málo času na postimplementační etapy /v některých střediscích je ani nezavedli/, a že noví programátoři se stále ještě učí programovat prostřednictvím vlastních

omylů a nikoliv pomocí dokonalé metodiky programování.

6.1 Zabezpečení proti chybám hardware

Mezi programátory je rozšířen chybný názor, že poruchy technického zařízení se nemají vůbec vyskytovat, a jestliže se vyskytnou, má je eliminovat a hlásit operační systém. V důsledku těchto nesprávných názorů nevěnují zabezpečení proti chybám hardware pozornost. Je sice pravda, že těžiště ochrany proti tento druhům chyb realizuje operační systém, ale aplikační program musí s možností chyb technického zařízení počítat tím více, čím déle výpočet trvá. Při dlouhotrvajících výpočtech je pravděpodobnost výskytu technické poruchy větší než při krátkých výpočtech. Opakování krátkých výpočtů navíc nezpůsobuje takové ztráty a organizační potíže v provozu jako opakování rozsáhlých několikahodinových výpočtů.

Programátor by měl zejména v programu:

- zajistit dlouhý výpočet metodou orientovaného stopu /check point and restart system, start od libovolného čísla listu sestavy na tiskárně, přerušit a pokračovat u kteréhokoli cyklu iterace atp./;
- reagovat v programu na všechny chyby hardware, které jsou hlášeny prostřednictvím operačního systému tak, aby uvedl výpočet do předem definovaného stavu, který dovolí odstranit následky chyby a zařídí pokračování výpočtu /příkazů v sekci DECLARATIVES zejména příkazem USE AFTER STANDARD ERROR PROCEDURE v jazyku COBOL, nebo pomocí příkazů pro ošetření chyb ON, SIGNAL, REVERT v jazyku PL/I /;
- je-li následkem technické chyby změna v konfiguraci počítače /nepracuje jedna mag. páska, nepracuje tiskárna/ měl by program mít možnost pracovat i při takto omezených podmínkách /změna počtu pomocných pásek, nahrazení jinou symbolickou jednotkou apod./;
- zajistit jiným způsobem kontrolu tam, kde technické ani programové vybavení standardně neposkytují dostatečné záruky detekce chyb /např. u dálnopisného

kódu, kde není kontrola paritou zařadit test kontrolní sumy/.

Počítače IV. generace a některé modernizované počítače III. generace mají podstatně zdokonalený návrh obvodů počítače s ohledem na detekci a odstraňování technických chyb. Spolu s vysokou spolehlivostí obvodů velké integrace a s postupným odstraňováním mechanických částí z jednotek počítače, by se měl stále zmenšovat podíl technických chyb na snižování spolehlivosti programů.

6.2 Chyby operačního systému

Každá verse operačního systému obsahuje určité chyby. Mělo by se stát pravidlem, že jakmile některý programátor ve spolupráci se systémovým programátorem zjistí prokazatelnou chybu:

1/ Okamžitě informují ostatní programátory o její existenci
2/ Oznámí, jak se chybě vyvarovat /jak se např. obejít bez klauzule, která chybu způsobila/
3/ Zajistí zápis příslušného komentáře do dokumentace
4/ Ohlásí chybu dodavateli software, aby ji mohl opravit.

Podle doporučení dodavatele programového vybavení se pak provede oprava chyby. V žádném případě by se nemělo přistoupit k bezhlavému opravování operačního systému. Takové zásahy jen vedou k dalším chybám a mohou naopak znehodnotit celý systémový mag.disk nebo mag. pásku.

Pro každý typ počítače a druh operačního systému by měla být organizována vzájemná informační služba o chybách operačního systému /pokud takovou službu neorganisuje sám dodavatel počítače/. Ušetří se tím mnoho nákladů, protože výpočtové středisko se vyhne chybám, o kterých už má příslušné informace.

6.3 Chyby ve vstupních údajích

Program musí být navržen tak, aby bezpečně určil nesprávná data, která by vedla ke zkroucení štecího programu nebo k chybné funkci některého z následujících programů. Programátor by měl zajistit, aby zpráva o chybě lo-

kalizovala místo a druh chyby. Podle těchto informací musí být schopen uživatel snadněji opravit chybné údaje. Jestliže se na opravě chyb bude podílet více pracovníků, musí být hlášení o chybách naprogramováno tak, aby každý z nich snadno identifikoval ty chyby, které má opravit /nejlépe samostatným protokolem o chybách nebo aspoň zvláštní značkou/.

V mnohých případech je nutno zajistit, aby ve vstupních datech byly obsaženy určité nadbytečné informace, které se využijí pro kontrolu. Jestliže se zvolí za vstupní data jen prostá posloupnost čísel, které jsou odděleny mezerami, těžko lze při chybě na děrné pásce zjistit, která čísla ještě patří k chybné větě, a která již k další větě, a chyba v jednom čísle může znehodnotit zbytek správných dat.

Snímací program je nutno vypracovat tak, aby odhalil co nejvíce chyb na jeden průchod souborem vstupních dat. Pro uživatele je nepraktické, když jsou mu postupně průchod za průchodem hlášeny jednotlivé chyby v téže větě. Řešení tohoto požadavku spadá do problematiky nazývané "zotavení po chybě". Je obtížné chybu identifikovat, ale bývá ještě těžší najít cestu, jak zahladit její následky a obnovit správnou činnost programu. Zásadu, aby následky chyby byly co nejvíce omezeny, a nebyly příčinou dalších jiných chyb lavinovitě se šířících, je nutno dodržovat ve všech případech řešení problému spolehlivosti programů.

6.4 Implementační chyby

Dělba práce mezi uživatele, analytika a programátora při zavádění automatizace skrývá v sobě nebezpečí, že v důsledku vzájemného nedorozumění dostane uživatel program, který nerealizuje jím požadované funkce, nebo je realizuje jinak a s jinými výsledky. Používá-li zadavatel pro formulaci úlohy přirozeného jazyka, analytik vývojových diagramů a programátor příkazů programového jazyka, mohou vzniknout chyby při transformaci z jednoho popisného systému do druhého.

Chybám tohoto druhu lze předcházet přesnou formalizací komunikačních systémů. Jedním z takových postupů je používání rozhodovacích tabulek. Při jejich použití je možné, aby jeden a ten samý systém vyjednávání použili jak zadavatel úlohy, tak analytik i programátor.

Obecně platí, že čím jsou vzájemné dorozumívací jazyky odlišnější, tím je těžší dorozumění mezi oběma stranami, a tím větší je možnost zkreslení předávané informace. Zkreslení se zvětšuje kromě toho i při zvyšujícím se množství na sebe navazujících komunikačních článků. Proto mají tak velký význam problémově orientované jazyky. Prostřednictvím nich uživatel formuluje pouze cíle výpočtu a programové vybavení počítače samo zjistí a provede odpovídající algoritmy pro jejich realizaci.

6.5 Chyby následkem špatné organizace

Příprava programů pro počítač je dnes velmi rozsáhlý a složitý proces, který přesahuje schopnosti a možnosti jednoho člověka. Stává se proto kolektivní činností různých specialistů, jejichž práci je nutno organizovat. Nevhodnou a nedostatečnou organizací vznikají nejrůznější chyby, mající příčiny v nesprávné posloupnosti vykonávaných činností, v nedostatečné výměně informací, ve zmatečních příkazech a specifikacích atd.

Ve většině výpočtových středisek se dosud používá praxe, kdy se soubor požadovaných programů dělí počtem programátorů. Tyto skupiny programů zpracovává každý programátor izolovaně sám. Následky představují úplnou pohromu. Skupiny programů izolovaně vypracovávané nejsou funkčně sladěny a vzájemně nespolupracují. Vzniká navíc duplicita v programátorských činnostech. Narůstá počet nedorozumění a tím i chyb. Prodlužuje se doba ladění, protože je obtížné nalézt, ve kterém programu chyba vzniká. Vyskytuje se řada konfliktních situací mezi programátory navzájem i mezi vedením výpočtového střediska, které není spokojeno s neustálým prodlužováním termínů dokončení programů. Na-

rovnoměrně je čerpán strojový čas na ladění programů. V jednom okamžiku chtějí všichni počítat, v následujících dnech vznikají prostoje, když programátoři studují výsledky testů. Mnozí programátoři musí podávat mimořádné výkony a věnovat zvýšené úsilí, protože jim byly přiděleny ty nejsložitější programy. U jiných programátorů zůstává část jejich pracovní kapacity nevyužita, protože dostali při rozdělování sice stejný počet programů jako první skupina, ale tyto programy jsou triviální.

Výše zmíněnou praxi je potřeba zavrhnout a nahradit ji způsobem organizace programátorské práce, která bývá označována termínem "metoda týmu vedoucího programátora" /v anglosaské literatuře/ nebo názvem "programování metodou chirurgické brigády" /termín používaný v sovětské literatuře/. V podstatě jsou zde řešeny všechny otázky, které se týkají organizace práce uvnitř skupin, pověřených vývojem programů, i styk skupin mezi sebou a s ostatním personálem provozu výpočtového střediska. Prosazuje se rozdělení činností ve skupině programátorů do hierarchicky členěného systému práce, který bývá označován jako "metoda tvorby programů shora dolů" Vedoucí programátor sestaví hlavní kostry programů a podprogramů, rozpracovává jejich klíčové části a rozděluje práci ostatním členům týmu. V týmu je uplatněna průběžná oponentura všech myšlenek a všech naprogramovaných skutečností. Součástí organizace práce jsou speciální prostředky: podpůrná knihovna vývoje a knihovna dokumentace systému. Zavádí se systém do testování programů. Protože se vychází ze zásad strukturovaného programování, je zajištěno, že navržené programy budou přehledné a srozumitelné i jinému programátorovi a později jednoduše modifikovatelné.

Úspěch každé lidské činnosti závisí na její organizaci. V programování vystupuje tato skutečnost jako velmi důležitý faktor. Každé středisko by se mělo zaměřit nejen na organisaci práce programátorských týmů, ale mělo by organizovat i formální stránku programování velmi pečlivě /používané formuláře, předávací protokoly, postup při

opravách a předávání programů apod./.

6.6 Sémantické chyby

Syntaktické chyby ve zdrojovém textu odhalí většinou kompilátor. Jak se dokonalují kompilační techniky, vzrůstá množství takových chyb, které kompilátor sám opraví /případně doplní standardně předpokládaným objektem/ a vypíše varující hlášení a zprávu o tom, jak chybu opravil. Programátoři mnohdy lehkomyslně spoléhají na dokonalost kompilátoru a nevěnují upozorňujícím výpisům patřičnou pozornost. Tak se stane, že chybějící příkazy end jsou doplněny na nesprávná místa, typy proměnných se zvolí nevhodně, ze záporných konstant se stanou kladné apod. Syntaktické chyby jsou takto nahrazeny chybami sémantickými.

Je nutno věnovat pozornost i chybám, vznikajícím ze záměny identifikátorů. Je krajně nevhodné v jednom programu deklarovat současně identifikátory POL a PÓL, protože se programátor může snadno přepsat nebo zapomenout označit rozdílnost znaků nula a velkého písmena "O". Časté je záměna jednoznakových identifikátorů i za j, k za l, e za l, m za n a naopak. Např. v příkazu: for k:= 1 step 1 until a [j, i] do BETA; byly zaměněny znaky i a j při psaní nebo děrování. Správně mělo být omezení cyklu hodnotou a [i, j]. Použije-li se při zkoušení jen situace, kdy i = j, zdá se být program dokonce v pořádku.

6.7 Špatná dokumentace

Mnoho chyb je zviněno v důsledku dokumentace, která:

- obsahuje chyby v textu nebo chybné příklady příkazů,
- není aktualizována pro poslední platnou verzi programu,
- je psána nejednoznačně, nesrozumitelně, nesystematicky.

Moderní racionalizační metody v oblasti programování kladou důraz na samodokumentaci programu svým zdrojovým textem. Tuto myšlenku lze plně využít pro údržbu programu

a jeho používání jiným programátorem než autorem. V praxi však program užívají i uživatelé neprogramátoři. Dokumentace pro ně bývá většinou podceňována a tak vzniká při rutinním provozu celá řada chyb.

6.8 Chyby obsluhy počítače

Mnoho chybných výsledků vzniká v důsledku nesprávné manipulace s programem obsluhou počítače. Čím více ručních zásahů program vyžaduje /výměna mag. pásek, nastavování klíčů na ovládacím pultu apod./, tím je větší pravděpodobnost, že dojde k omylu nebo přehlédnutí a ke vzniku chyby. Programátor by měl navrhnout program tak, aby nevyžadoval ručních zásahů nebo je aspoň minimalizovat. Je to nejen výhodné z hlediska snadného použití programu v tzv. "uzavřeném provozu výpočtového střediska", ale zejména se tím výrazně zvýší spolehlivost programu. Tam, kde jsou ruční zásahy nutné, měly by být naprogramovány v zápětí vhodné kontroly a protokolování provedených zásahů a obsluha by měla být nucena ke kontrole svých akcí.

V řadě výpočtových středisek tyto chyby úzce souvisejí se špatnou organizací průvodních formulářů zadávaných prací. Následkem jsou chybné verze magnetických pásek s programy nebo s daty použité pro výpočet, záměna vstupních dat, nesprávná šířka tabelačního papíru, jiné varianty výpočtů než požadované, záměny řídících štítků atd.

Programátoři využívají skutečnosti, že bezprostředně prokazatelný viník je obsluha počítače a odmítají zahrnovat tyto chyby do hodnocení spolehlivosti programů. Pravdou však zůstává, že programátor má vždy k dispozici programové prostředky, kterými lze předcházet těmto chybám, a je hrubou chybou programátora, jestliže spoléhá na nejméně jistý článek v řetězu postupu zpracování dat - na člověka.

5.9 Chybné postupy testování

Jak bylo již uvedeno, pouhým testováním programu ne-

ní možno dokázat, že je program bez chyb. Přesto je dnes testování nejrozšířenější způsob ověřování programů.

Programátoři často dělají chybu tím, že uvažují výhradně jen o množství kontrolních příkladů, aniž by provedli rozbor, zda se jedná o reprezentativní vzorky. Za správný je považován takový program, který dává správné výsledky v co nejvíce případech. Provedeme-li 10 000 testů jen s jedním druhem transakcí stejné hodnoty, je to mnohdy skoro totéž, jako bychom provedli jen jeden test /pokud tímto testem nesledujeme něco jiného, např. dynamické obsazování oblastí paměti/. Mnohdy se neberou v úvahu extrémní případy, které mohou nastat. Zapomíná se, že je nutno v testech simulovat i kritické situace /chyby v datech, poruchy technického zařízení/.

Zkušební data by měla být pečlivě vybrána a připravena, a způsob testování důkladně předem promyšlen. V praxi to bývá často naopak. Za testovací data se vezmou nahodilé údaje a příležitostně se s nimi provede výpočet a program se považuje za ověřený. Předá se do rutinního užívání provozu počítače, a tak první rutinní zpracování bývá teprve opravdovou zkouškou programu a vyčerpávající zkouškou nervů programátorů a uživatelů.

6.10 Chyby programátorů

Nepozornost programátorů je zdrojem chyb všeho druhu. Soustředěnost při práci by měla být pro programátora samozřejmostí. Bohužel tomuto požadavku neodpovídají pracovní prostory programátorů ve většině výpočtových středisek. Skutečnost, že programátor pro napsání programu potřebuje čistý papír, tužku a gumu, vede mnohé nezasvěcené vedoucí hospodářské pracovníky k domněnce, že programátoři mohou tyto papíry popisovat v jakémkoli prostředí, ve kterém je jim umožněno si sednout. Přeplněnost místností dalšími programátory, kteří mají stoly jednou hranou přiraženou k sobě a sedí v úzkých uličkách, hlučné telefony na každém stole nebo naopak jeden telefón pro velkou kancelář, ke

kterému jsou programátoři vyvolávání ze svých míst, zástupy návštěvníků z jiných útvarů, kteří se přišli poradit proč ten či onen program pracuje tak a ne jinak, hlučný kompresor na rozvod stlačeného vzduchu pod okny atd., to vše vytváří ideální podmínky pro vznik celé řady chyb. Jejich následky a odstraňování vyžadují náklady několikrát převyšující investice do vybudování vhodných pracovních prostor pro tyto náročně duševně pracující zaměstnance.

Někdy je náborem získán do programovacího týmu pracovník, u kterého se dodatečně zjistí, že je nedbalý, neukázněný, nedá si poradit, nemá pocit odpovědnosti ke své práci, nejeví zájem o zvyšování kvalifikace a kvality své práce. Takového pracovníka je nutno převést na jiný druh práce, protože napsáním chybných programů může zaměstnavateli zavinit velké škody.

Mnohé chyby dělají programátoři z nevědomosti. Vybavit programátory dokonalou sadou bezchybných programovacích specifikací a příruček, zajistit jejich dokonalé vyškolení a průběžně je metodicky vést ke zvyšování kvalifikace, musí být nedílnou součástí náplně práce vedoucích výpočtových středisek. Jen tak je možno podstatně snížit chyby při vypracovávání programů.

Často je řada chyb způsobena programátory, kteří tvoří své programy nebo provádějí opravy v časové tísni. Uživatelé přicházejí se svými požadavky pravidelně na poslední chvíli. Každý programátor může uvést bezpočet příkladů, kdy se o nutnosti provést určitou akci za pomoci počítače vědělo měsíce předem.

Požadavek na vypracování programů však přišel do výpočtového střediska týden před termínem provedení výpočtu. Nedokonalá analýza, narychlo sestavené a nevyzkoušené programy pak znamenají stresové situace pro všechny, kdož se na takové akci podílejí. Tuto skutečnost by si měli uvědomit všichni vedoucí pracovníci, kteří často kritizují, že jim počítač dává výsledky pozdě a špatné, a odmítají se seznámit

se základními principy přípravy úloh pro počítače. Provedení výpočtu s programem, který se teprve nachází ve stadiu ladění konec konců situaci ve zpoždění úkolu nevyřeší, ale téměř vždy navíc zkomplikuje. Obdobná situace nastane, když jsou několik dní před rutinním zpracováním nuceni programátoři k rozsáhlým mimořádným úpravám již delší dobu používaných programů.

Na druhé straně jsou velmi rozšířené případy, kdy se do časové tísně dostávají programátoři sami v důsledku nepřiměřeného optimismu a neseriozních předpovědí pracnosti a termínů vypracování programů.

Programátoři někdy zvyšují počet chyb tím, že se snaží navrhnout komplikované, obecné, vše řešící programy velkého rozsahu. Cílem by mělo být vždy nejjednodušší ekonomické řešení a nikoliv logické monstrum.

7. Spolehlivost programů a hledisko nákladů

Vytváří-li programátor od počátku program tak, že řadou testů ověřuje, zda výpočetní proces probíhá podle předpokladů, pak sestaví program,který vykazuje vysokou spolehlivost i při výskytu havarijních incidentů - program je méně citlivý na chyby. Protože instrukce testů zabírají místo v paměti a spotřebovávají strojový čas, bude program pracovat pomaleji, než jiný program, který takové testy nemá. Mnoho programátorů dává raději přednost rychle pracujícímu nespolehlivému programu před spolehlivě pracujícím, ale pomalejším programem. Je to pozůstatek myšlení z doby průkopnického zavádění počítačů, kdy programátoři mezi sebou soutěžili o minimální počet instrukcí a nejkratší dobu práce programů ve strojových kódech. Protože program bez testů je rychlejší a snadněji napsán /navíc chyby se může programátor dopustit právě v instrukcích testu/ je pochopitelné, že mnoho programátorů se dá svést na "cestu nejmenšího odporu". Mnohdy jsou v tomto bludu podporováni vedoucími, kteří z hlediska termínů a nákladů takový způsob psaní progra-

mů nepřímo preferují vytvořenými podmínkami a hodnocením programátorů podle počtu napsaných programů a rychlosti výpočtů jejich programů. Proto se proti zvyšování spolehlivosti programů staví dosti často bariéra efektivnosti a nákladů.

Určité nebezpečí, že při snaze naprogramovat všechny možné /někdy i nepravděpodobné/ testy bude vytvořen několikanásobně rozsáhlejší program, samozřejmě existuje. Je mu možno zabránit tím, že se problém spolehlivosti bude řešit komplexně. Budou-li z hlediska zvyšování spolehlivosti programů navrhovány operační systémy, kompilátory programovacích jazyků, technická zařízení počítačů a prostředky pro testování programů, pak se počet testů přímo vytvářených ve zdrojovém textu programu aplikačním programátorem sníží na nezbytné minimum.

8. Závěr

Situace v některých výpočtových střediscích je značně kritická. Stovky používaných programů, které byly vypracovány starou individualistickou technikou po primitivní analýze, vyžadují tolik času na svou údržbu /odstraňování chyb, modifikace, rozšiřování, úpravy apod./, že zbývá čím dál méně času na tvorbu nových programů. Proto se nové programy vytvářejí v časové tísni, plné těchže chyb a nedostatků, a situace se jen více komplikuje. Programátoři a analytici jsou v takových výpočtových střediscích tolik zatíženi svou špatně vykonávanou prací, že nemají čas se zamyslet, jak pracovat lépe.

Co může odstranit tuto kritickou situaci?

Výsledky analýzy musí poskytovat ucelenější obraz o řešení problému než dosud. Zatím lze analýzu přirovnat k plovoucímu ledovci, který odkrývá jen jednu desetinu své nebezpečné velikosti nad hladinou. Stejně tak i řada výsledků nedokonalých analýz zachytí jen část problematiky. O větší neodhalenou část problémů se později rozbije úsilí programátorů do neefektivní a neproduktivní práce. Zavádění systé-

mové analýzy do praxe výpočetních středisek, nejen vizitkami na kancelářích analytiků, ale především obsahem a výsledky práce, je nutná podmínka pro možnost dobrých výsledků navazující programátorské činnosti.

V oblasti programování je nutno zavést dokonalou metodiku programování, vybudovanou nikoliv na empirickém, ale na vědeckém základě. Současné směry v racionalizaci programování, které jsou prosazovány různými progresivními metodami /normované programování, zlepšené programování, strukturované programování, modulární programování/ je možno charakterizovat jako rozpracování následujících zásad.

1/ Než se přistoupí ke psaní jednotlivých příkazů programu v určitém programovacím jazyku, je nutno nejprve navrhnout celkovou výstavbu programu a stanovit jeho chování z hlediska nejobecnějších funkcí. Tento postup cyklicky opakovat pro jednodušší a jednodušší struktury a funkční vlastnosti až na příkazy programovacího jazyka. Dávat přednost metodě návrhu programu shora dolů před metodou zdola nahoru. Zdá se, že staré přísloví: "Dvakrát měř a jednou řež!", je nutno pro potřeby programování upravit do znění: "Nejprve desetkrát uvažuj, potom pečlivě programuj!"

2/ Kromě prvotních požadavků na správnou problémovou funkci programu, a kromě sekundárních požadavků na efektivní práci programu /rychlost, krátká doba kompilace, malé požadavky na operační paměť/ je důležité zajistit, aby program byl jednoduchý, srozumitelný a snadno modifikovatelný.

3/ Pro ověřování správnosti programu je potřeba použít exaktních metod, které zaručí maximální spolehlivost programu.

4/ Programování se stává kolektivní činností. Proto je nutno věnovat pozornost dobré organizaci práce programátorských týmů. S tím úzce souvisí požadavky na vzájemnou vyměnitelnost programů mezi programátory navzájem, rychlé zaškolování nových pracovníků do stávajících týmů

a všeobecné zvyšování znalostí analytiků a programátorů.

5. Pro maximální zhodnocení programátorské práce a pro zvýšení spolehlivosti, je nutno důsledně prosazovat standardizaci programování ve všech směrech.

V příspěvku nebyly podrobně uvedeny metody, které se zabývají dokazováním správnosti a technikou ladění programů. Tato problematika byla diskutována na celostátním semináři SOFSEM '76 [4] [12]. Referáty obsahují celou řadu ilustrativních příkladů různých systémů, které byly implementovány. V eseji [4] autor navrhl českou terminologii a částečnou klasifikaci pojmů z této oblasti programování. Pozorné prostudování uvedených referátů je možno doporučit všem programátorům.

Nikdy se nenavrhne absolutně spolehlivý program, jako nelze navrhnout a vyrobit absolutně spolehlivý stroj. Je však nutno navrhovat programy tak, aby jejich spolehlivost byla úměrná k účelu aplikace.

Profesor novosibirské univerzity A. Jeršov ve svém pozoruhodném referátu [5] upozornil na důležitou skutečnost. Programování ztrácí již dnes mýtus tajuplné výjimečné činnosti. Programátoři si to musí plně uvědomit. Do budoucna nelze počítat s tím, že se chyby v programech budou jen vysvětlovat a omlouvat, ale že za ně budou muset nést programátoři vůči uživatelům plnou odpovědnost.

Spolehlivost programů je faktor, který významnou měrou může rozhodnout otázku, zda se počítače stanou všestrannými pomocníky člověka nebo postrachem společnosti, jak to líčí někteří spisovatelé povídek science - fiction.

Literatura

1 Boehm B.: Software and its impact: Datamation, roč. 19, 1973, č. 5, str. 48 - 59

2 Ogdin J.: Designing Reliable Software, Datamation r. 18, 1972, č. 7, 71 - 78

3 Huang J.: An approach to Program Testing. Comput. Surv., 1975, č. 3, 113 - 128

4 Hořejš J.: Ladění programů, sborník VVS OSN Bratislava ze semináře SOFSEM'76, 4 - 12

5 Ershov A. P.: Aestetics and the Human Factors in Programming, Datamation, 1972, č. 7, 62 - 67

6 Rouback J.: Software Reliability how it affects system reliability. Microelectron. and Reliab., roč. 14, 1975, č. 2, 121 - 140

7 Novák D.: COBOL - chyby v uživatelských programech, MAA roč. XV. č. 2, 1976, 53 - 55

8 Problematika tvorby vel'kých programových systémov. VVS OSN Bratislava, október 1975

9 ICL 2900 Series: Technical overview, International Computers Limited, London 1974

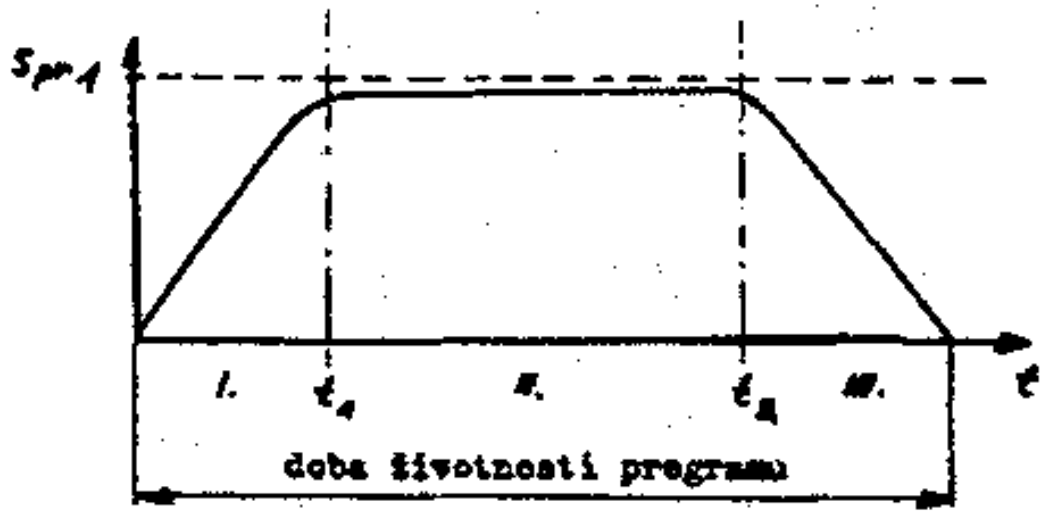
10 Bell D.: Programmer selection and programming errors. Computer Journal, roč. 19, č. 3, 1976, 202 - 206

11 Palme J.: Language for Reliable Software, Datamation, roč. 21, 1975, č. 12, 77 - 80

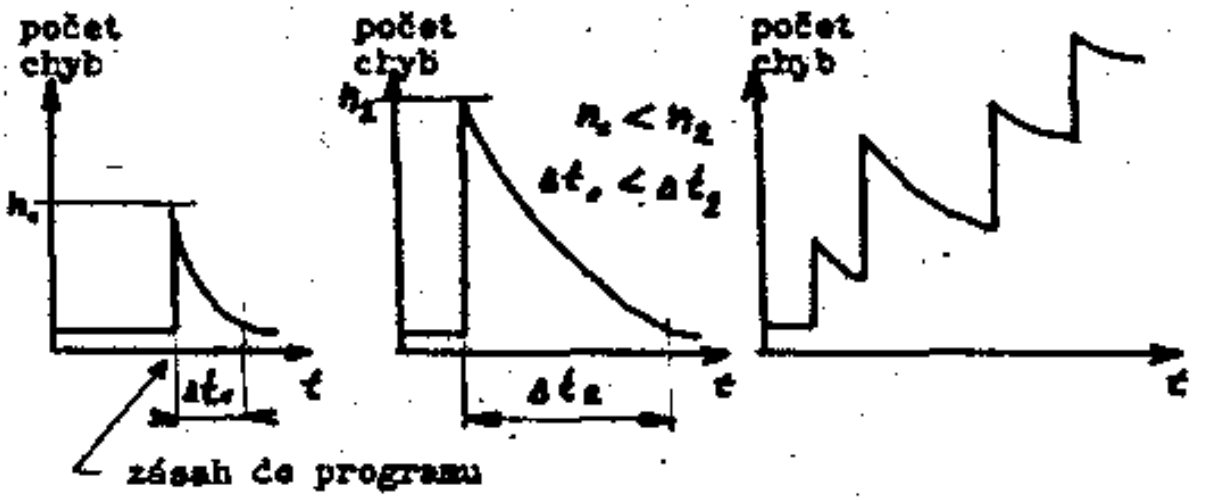
12 Gruska J., Privara I.: Dokazovanie správnosti programov, sborník VVS OSN Bratislava ze semináře SOFSEM 1976, 331 - 376

13 Wirth N.: The programming language PASCAL. Acta informatica 1/1971, 35 - 63.

14 Novák D.: Jak hledat chyby v programech, MAA, roč. XVI, 1976, č, 10, str. 389 - 341

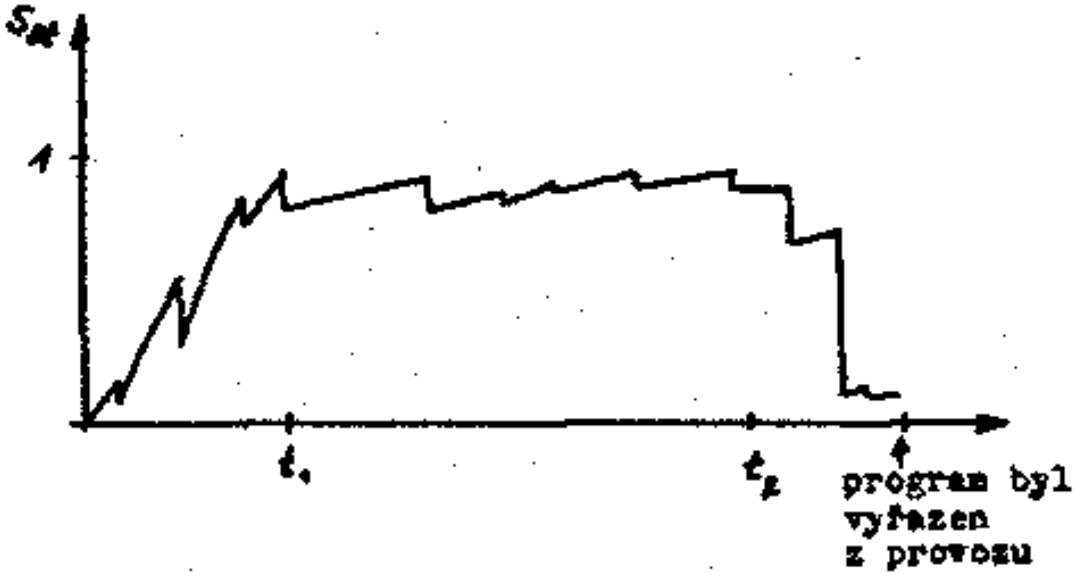


OBR. 1.

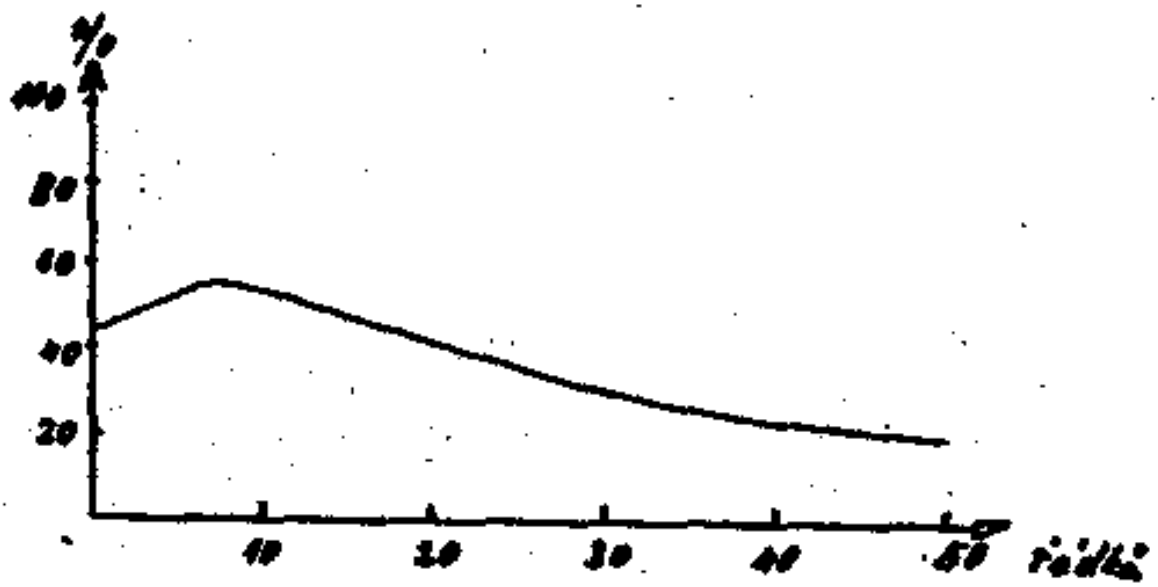


2. a) 2. b) 2. c)

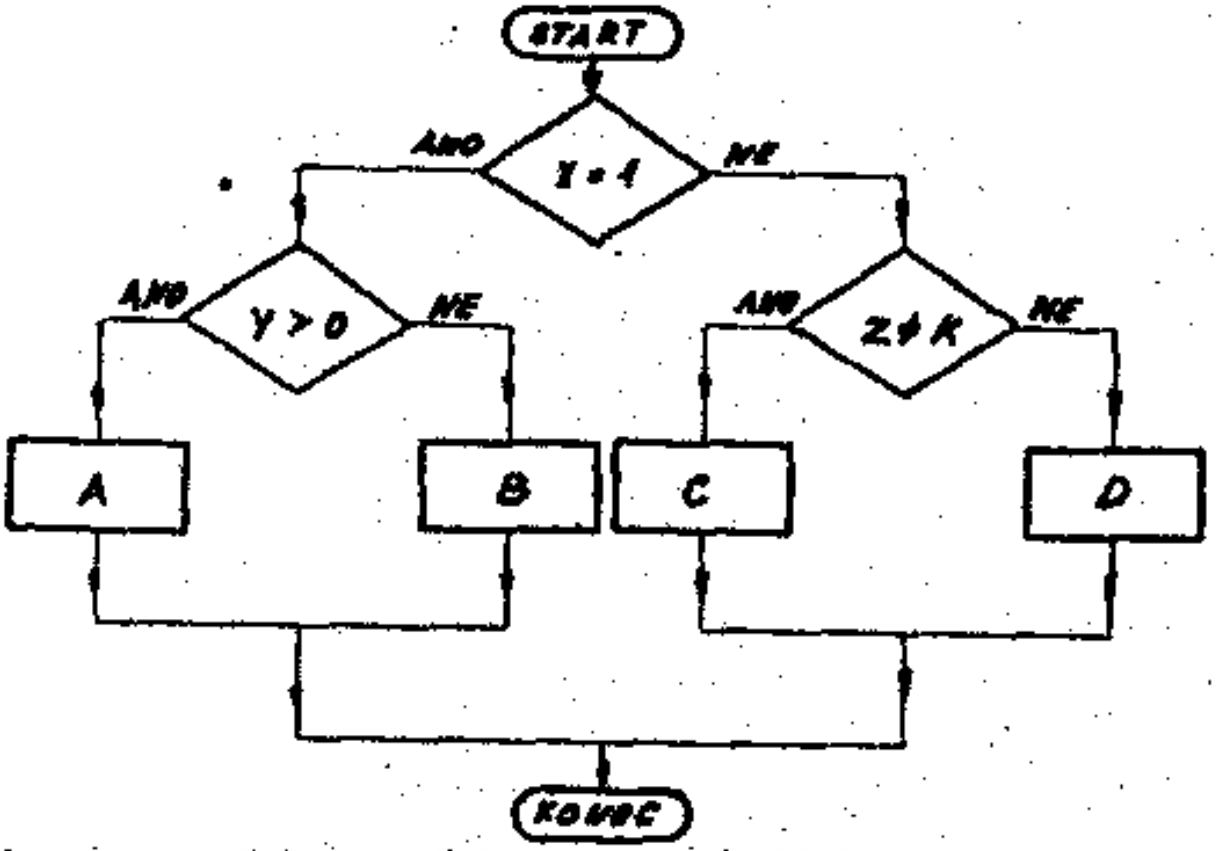
OBR. 2.



OBR. 3



OBR. 4.



1.test: X=1 ∧ Y>0 testuje blok A
2.test: X=1 ∧ Y≤0 testuje blok B
3.test: X≠1 ∧ Z≠K testuje blok C
4.test: X≠1 ∧ Z=K testuje blok D

Soubor minimálních testů, aby byly testovány všechny funkční bloky programu /A,B,C,D/ a všechny testy nejméně jednou.

OBR. 5