SHORA-DOLŮ NEBO ZDOLA-NAHORU ?

RNDr. Ing. Ivan Lexa CSc, OZS Břeclav

1. Úvod

"Pochopitelně že jedině shora-dolů", slyšíme při každé příležitosti a týž dojem získáváme pročítáním většiny odborné literatury z posledních pěti let. Pokud se semtam někdo najde, kdo má jiný názor, často si již netroufá jej vyslovit nebo dokonce publikovat, aby nevypadal jako nevzdělanec. Tento příspěvek si klade za cíl ukázat, že oba postupy se vzájemně nevylučují a že jeden bez druhého by dokonce neměl ani smysl.

Podstata věci přesahuje rámec programování a týká se vlastně jakéhokoliv tvůrčího procesu. Z toho důvodu a vlastně proto, že v jiných oborech se postupu zdola-nahoru dostává více cti než je tomu /momentálně/ v programování, jsou v příspěvku schválně uváděny názorné příklady z různých oborů.

2. Základní pojmy a definice

Abychom mohli srozumitelně vyjadřovat složitější myšlenky, potřebujeme především vhodnou slovní zásobu. Rekapitulujme si proto podstatu postupů shora-dolů a zdola-nahoru a tím, že přitom vytvoříme /pro potřeby tohoto příspěvku/ některé užitečné pojmy a označení.

Oba postupy jsou symbolicky zachyceny na obr. 1. Čtverečkem je zde zobrazen dílčí výtvor, budeme mu říkat prvek. Prvkem může být například

v programování : program, podprogram, modul, příkaz, dat.typ,
ve strojnictví : šroub, ložisko, motor, převodovka,
v hudbě : skladba, motiv, akord, tón,

Ke každému prvku můžeme přistupovat dvojím způsobem. Z vnějšího pohledu budou zajímavé užitné vlastnosti prvku, tedy to, co nám prvek může poskytnout, k čemu ho lze použít. Naopak z vnitřního pohledu jde o to, jak je prvek vytvořen, realizován, tedy jeho struktura.
Čára spojující na obrázku dva prvky vyjadřuje skutečnost, že výše zakreslený prvek je realizován za pomocí níže zakresleného prvku. Tak např. spojnice na obr. 2 říkají, že stěna je zkonstruována z cihel a malty, malta pak z písku, vápna a vody. Vztah dvou prvků, symbolizovaný spojnicí, nazývejme stručně vazba. Zdůrazněme zde jakési pravidlo, že v rámci jedné vazby jsou u nižšího prvku rozhodující jeho vnější vlastnosti /tedy co/, kdežto u vyššího prvku naopak jeho vnitřní struktura /tedy jak/. V hierarchii prvků a vazeb se tedy jednotlivý prvek jakoby obrací svým vnějškem směrem nahoru a svým vnitřkem směrem dolů /při stavbě stěny nás zřejmě bude zajímat jaká je hustota malty, ale nebude nás zajímat kolik vody do ní dali, aby tu hustotu dosáhli/.

Vraťme se však znovu k obr. 1. Postup shora-dolů /dále budeme používat označení ⇩ / začíná vždy jediným prvkem, u kterého známe vnější vlastnosti /= zadání/, ale neznáme jeho strukturu. Tedy víme co, ale zatím nevíme jak. V obrázku je tento prvek označen číslem 1 a nazývejme ho cíl /může to být například sídliště, které máme postavit/. Nyní podrobíme prvek analýze, přičemž se vědomě držíme stěžejních věcí a vyhýbáme se detailům. Z analýzy nám vyplynou požadavky na nižší prvky /v obr. 1 označeny čísly 2, 3, 4/ a struktura prvku 1 /která vyjadřuje jak jsou prvky 2, 3, 4 v jeho konstrukci použity/. V případě našeho sídliště skončí tato etapa například mapou /v které budou zakresleny jen obrysy domů/ a popisem vnějších /užitných/ vlastností předpokládaných typů domů.

Prvky 2, 3, 4 nás však staví znovu do stejné situace - známe vnější vlastnosti, ale zatím neznáme strukturu. Stačí tedy pro každý z nich zopakovat obdobný analytický postup jako s prvkem 1 /v našem příkladě dekomponovat domy dejme tomu na byty/. Dalším a dalším opakováním nás ⇩ vede ke stále většímu počtu prvků, složitost přibývajících prvků však neustále klesá. V kaž-

dém stádiu tohoto postupného "zjemňování" máme nějaké prvky, jejichž vnější vlastnosti i způsob využití již známe, zatímco jejich strukturu a způsob realizace ještě ne. Množinu těchto prvků nazývejme plán.

A nyní přirozená otázka : kdy a jak vlastně ⇩ skončí ? Odpověď : Aby ⇩ měl vůbec smysl, musíme již při jeho zahájení mít jistotu, že existuje množina základních prvků, nazvěme ji jednoduše základna. Prvky základny není třeba tvořit, protože již jsou vytvořené a kdykoliv je potřebujeme, můžeme je použít. Abychom tedy projekt úspěšně završili, musíme ⇩ cílevědomě "směrovat" k základně a skončíme tehdy, když poslední prvek plánu naváže na základnu.

Přirozenou základnou pro jakýkoliv obor lidské tvůrčí činnosti jsou prvky, které nevytvořili lidé. Říkejme jim suroviny. Dávno se přišlo na to, že taková základna je příliš nízká a neumožňuje stavět si vysoké cíle. Lidé si proto začali vytvářet vyšší, umělé základny, a to postupem opačným než je ⇩ .

Postupem zdola-nahoru /dále budeme používat označení ⇧ / vždy navažeme na existující základnu a vytváříme prvky nové, vyšší základny. Nová základna nemusí vždy tvořit oddělenou hladinu /jak to napovídá obr. 1/. Některé prvky staré základny totiž nemusí ztratit aktuálnost a mohou se stát i prvky nové základny. Tak například vytvořením integrovaných obvodů neztratilo smysl používat diskrétních tranzistorů. Občas se také stává, že dosavadní základna nestačí na to, abychom na ní vybudovali základnu novou, takže se musíme zčásti nebo zcela opřít o základny nižší./například tranzistory nebylo možné vytvořit na základě elektronek/.

Dá se říci, že ⇧ vždy neprobíhá podle přesné logické šablony /jako je tomu u ⇩ / a cesta ke kvalitativně vyšší základně může být někdy dost dlouhá a klikatá. Souvisí to s tím, že nová základna si svou důvěru a postavení získá teprve opakovaným úspěchem projektů, které se o ni opíraly.

Současnou základnu spolu se všemi nižšími základnami, na kterých je postavena, nazvěme báze /obr. 1/. Jak asi vypadá báze současného programování, to nastiňuje stručně obr. 3.

Ještě než přejdeme k dalším úvahám, chtěl bych upozornit

na charakteristiku ⇩ a ⇧ v/1/, která je tak brilantní, že by bylo škoda, aby upadla v zapomenutí.

3. Porovnání obou postupů

	⇩	⇧
podstatou postupu je	analýza	syntéza
nový prvek je nám	prostředkem	cílem
že píšeme nový prvek nebo danou základnou vytvořit	nevíme jistě, spíše předpokládáme	víme jistě
jaké bude mít nový prvek vlastnosti	přesně víme	to přizpůsobíme možnostem základny
jak bude nový prvek zkonstruován	vyřešíme později	tím se musíme zabývat hned
konkrétní použití nového prvku	známe	nezajímá nás (může být libovolné)
předpokl. aplikace	v 1 projektu	v mnoha projektech
návratnost vynaložené námahy	brzká, jednorázová	opožděná, postupná
hlavní riziko	že požadujeme něco co nepůjde (nebo nebude efektivní)	že vytvoříme něco, co se nebude dostatečně využívat
aplikace postupu je spíše záležitostí	jednotlivce nebo řešitelského týmu	kolektivní až celospolečenskou

4. Nevýhody obou postupů, vzájemná podpora

Největší úskalí ⇧ je riziko špatného odhadu poptávky. Není-li nový prvek nebo pracně vytvořená celá základna v tvůrčí

praxi přijata a běžně využívána, vyjde vynaložená námaha nazmar. Historie zná namálo pokusů o vytvoření nových základen, které neuspěly. Nejhorší na tom je to, že často se ani pořádně neví proč. Zdá se, že kromě kvality základny rozhodují o jejím úspěchu ještě další silné, ale málo prozkoumané faktory. Tak například uměle vytvořený jazyk Esperanto se neujal, ačkoliv podle názorů jazykovědců je to velmi kvalitní jazyk /zaručeně lepší než například angličtina/. Jiný pozoruhodný ale také skoro neúspěšný jazyk je ALGOL 68.

Hlavní nevýhodou ⇩ je multiplicita tvůrčí práce. Pravidelně se totiž stává, že v různých projektech se nezávisle řeší stejné nebo velmi podobné problémy a tak se vlastně plýtvá tvůrčím potenciálem lidí. U rozsáhlejších projektů /zejména jsou li řešeny týmem s větším počtem pracovníků nebo dokonce pracovišť/ dochází k multiplicitě i uvnitř projektu. Na první pohled by se mohlo zdát, že multiplicita "mezi projektová" a "vnitroprojektová" mají spolu málo společného. Stačí si však uvědomit, že například sérii 15 projektů, vyřešených na jednom pracovišti za dva roky, lze také chápat jako jediný projekt o 15 částech, který se řešil postupně. Z toho vidíme, že nemá valný smysl rozlišovat oba druhy multiplicit a že jejich společnou příčinou je příliš velký rozsah prací ⇩, který již nedovedeme organizačně zvládnout. Vzhledem k tomu, že rozsah prací ⇩ je určen "propastí" mezi zadáním projektů a úrovní základny, dá se z toho udělat dílčí závěr, že ⇩ není ekonomicky pro dosahování cílů příliš vysokých, tj. neodpovídajících stavu současné základny.

Vidíme, že ⇩ i ⇧ má své nedostatky. Ukažme dále, že oba postupy se navzájem podporují, tj. jeden druhému pomáhá jeho nedostatky zmírnit.:

a/ Pomocí ⇧ budujeme nové, vyšší základny. Pokud se nová základna podařila", bude pravidelně využívána a přinese to výrazný ekonomický efekt. Každý další projekt, řešený ⇩ bude totiž jednodušší a tedy méně pracný. Navíc prvky základny mohou být daleko pečlivěji vytvořeny a jejich lepší kvality se projeví "lepšími parametry" dosahovaných cílů pomocí ⇩.

b/ Sledování multiplicitních prvků v projektech ⇩ nám umožňuje vytipovat, co nám chybí v současné základně. ⇩ tedy poskytuje cenné podněty pro ⇧ a snižuje tak již zmíněné riziko špatného odhadu poptávky.

5. Kdy volit ⇩ a kdy ⇧

Když máme zvládnout konkrétní projekt /projekty/ a máme k tomu k dispozici vyhovující /tj. především dostatečně vysokou/ základnu, je zřejmě nejvhodnější ⇩. Znovu však zdůrazněme /jak již bylo dříve naznačeno/, že táž základna může být vyhovující pro jeden projekt a přitom nevyhovující, máme-li takových /navzájem podobných/ projektů dělat mnoho.

Pokud pro požadovaný projekt /nebo projekty/ vhodnou základnu nemáme, ale víme, jaká by měla být, bývá ekonomičtější pomocí ⇧ dobudovat stávající nebo vytvořit novou základnu. Přirozeně je vhodné na tento úkol podle možností soustředit prostředky /případně i kapacity/ ze všech pracovišť, které jsou v podobné situaci a mají zájem vytvořenou základnu využívat.

V některých případech může být vhodné vytvářet nové základní prvky tak vysoké, že přímo nenavazují na stávající základnu. Pak je na místě kombinace obou postupů: pomocí ⇧ stanovíme vnější vlastnosti nového prvku a pak ho realizujeme projektem ⇩.

Někdy se nám může stát, že hned po prvních krocích cítíme nejistotu. Nevíme totiž, zda některé prvky vzniklé dekompozicí budou vůbec realizovatelné. Cítíme, že kdesi hodně hluboko poblíž základny nutně narazíme na prvek, od jehož vlastností bude záviset úspěch celého projektu nebo přinejmenším způsob dekompozice hned v prvních krocích nahoře. Jinými slovy by se dalo říci, že máme v základně "díru" a přitom máme důvod se domnívat, že vlastnosti tohoto chybějícího prvku základny mají klíčový význam. Striktně trvat na ⇩ v této situaci je asi mírně řečeno nerozumné. Zřejmě je vhodné "na chvíli" opustit ⇩ a pomocí ⇧ klíčový prvek napřed vytvořit.

Existuje ještě jedna typická situace, kdy není vhodné vystačit s ⇩: když dostaneme nepřesné /mlhavé/ zadání a přesto

máme něco vytvořit. Pravidelně se to stává v oborech, kde teprv probíhá základní výzkum a neschopnost přesně formulovat zadání, nelze nikomu zazlívat. Praxe ukazuje, že čím je cíl nepřesněji stanoven, tím hůře jsme schopni ho dekomponovat. Pokud se nám dekompozice povede, bývá pravidlem přenos nepřesnosti na dekompozicí vzniklé prvky. Tímto způsobem se může nakonec znehodnotit celý projekt do takové míry, že ztrácí smysl.

Jak v takové situaci efektivně postupovat ? Snažme se představit si množinu všech přesných zadání, která by se mohly za nepřesným zadáním "skrývat". Pak vytvořme takovou novou /dá se říci účelovou/ základnu, která by mohla posloužit libovolnému zadání z této množiny a přitom byla co možná nejvyšší.
Tak získáme možnost ekonomicky realizovat řadu jednotlivých projektů na základě postupně upřesňovaných zadání. Situace a způsob vhodného počínání jsou tedy obdobné, jako bychom měli za úkol realizovat množství příbuzných projektů.

Nakonec kardinální kriterium : V dané situaci volíme z ⇩ a ⇧ ten postup, který se jeví jako ekonomičtější, tj. který slibuje menší riziko zbytečně vynaložené tvůrčí práce.

6. Závěr pro programování

Napřed si položme otázku : je současná základna programování dostatečná ? Odpovíme NE a dokážeme to nepřímo. Kdyby byla dostatečná, docházelo by k multiplicitám programátorských prací jen v ekonomicky nevýznamné míře. Potřebujeme tedy novou základnu, nebo lépe řečeno nové základny /žádná základna není totiž definitivní/. Na této potřebě nic nemění ani různost názorů na to, to, jaké formy by nové základny měly mít /zda to budou rozpáhlé knihovny modulů /3/, nějaké ještě vyšší programovací jazyky nebo snad dialogové programovací systémy/.

A nyní se rozhlédněme kolem sebe. Většina profesionálů bere existenci základen jako samozřejmý fakt a problémem jejich vzniku nebo rozvoje se vůbec nezabývá. I mezi těmi, kteří potřebu rozvoje základen chápou, je rozšířen názor, že k jejich rozvoji dochází jakési samovolně, protože nové základní prvky vznika-

jí jako vedlejší produkt ⇩/stačí prvek, který vznikl v rámci projektu, trochu zobecnit a zařadit do knihovny/. Tento názor je nejen mylný, ale i nebezpečný, protože odsuzuje základnu k živoření z milodarů, které jsou nesourodé a jejichž kvalita a obecnost/pro projekt vyhovující/ většinou neodpovídá poslání základny. Jen ojediněle se setkáváme s hlasy, které spojují vznik základny výhradně s ⇧ /například /2//.

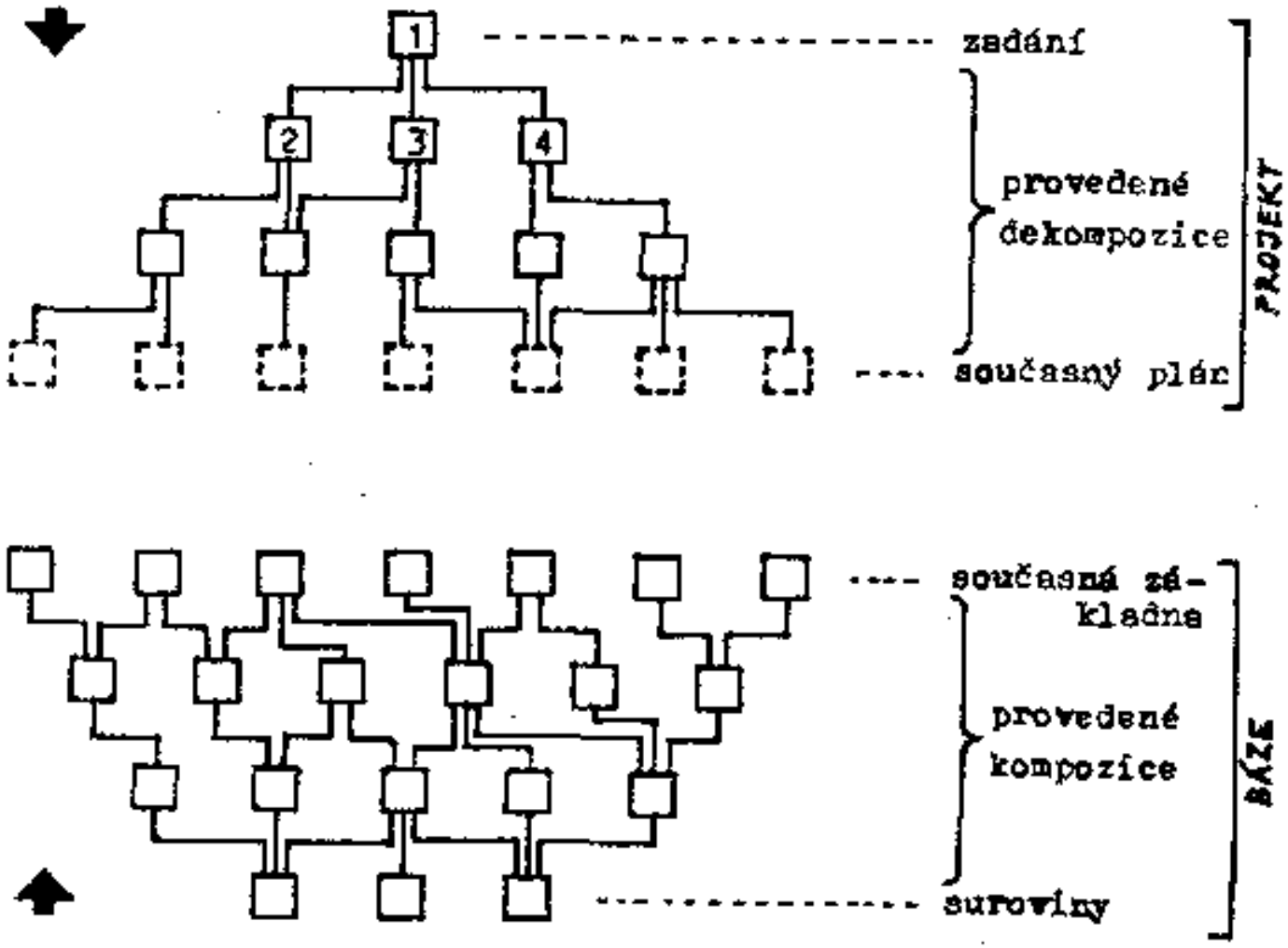
A tak stále stojíme před paradoxem :Na jedné straně by základny, s ohledem na jejich klíčový význam, zasluhovaly systematickou, organizovanou a cílevědomou péči značné části profesionálních programátorů /navíc té "lepší" části/. Na druhé straně jsme však svědky toho, jak je celá nová generace programátorů jednostranným zdůrazňováním ⇩ od péče o základny vlastně odrazována.

7. Použitá literatura

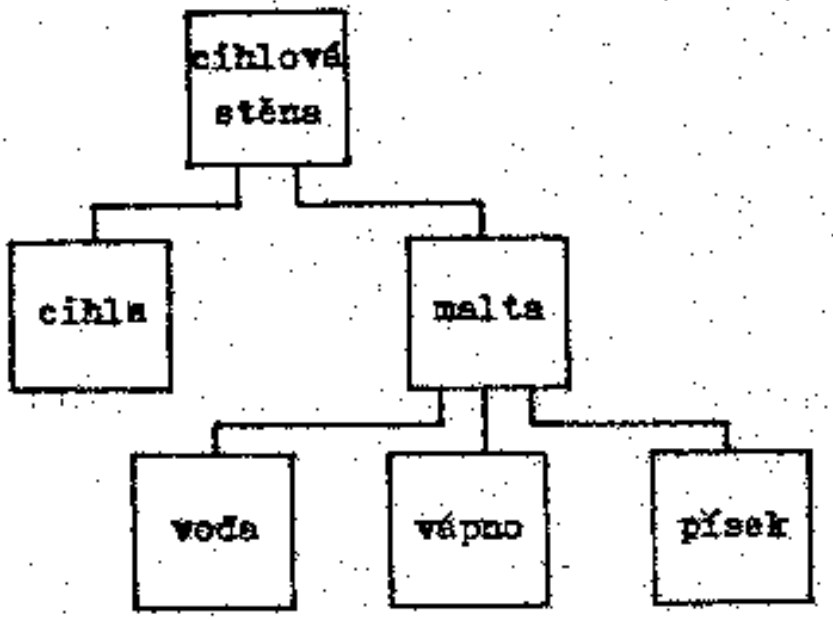
/1/ Suchomel J.: Technologie strukturovaného programování programátor - profesionál ?.Sborník semináře Programování 83, strana 50, řádek 5 až 19.

/2/ tentýž příspěvek, strana 51 dole.

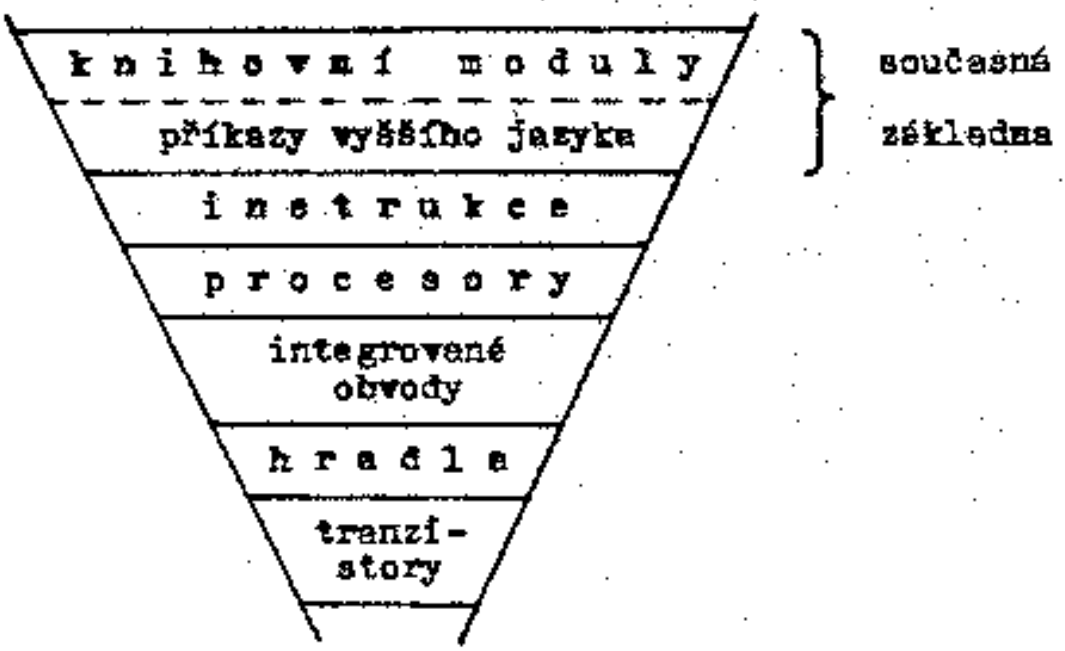
/3/ Bébr R.: Přežije programátor rok 2000 ? Sborník semináře Programování 85, strana 8 dole.



Obr. 1 - znázornění struktury projektu a báze



Obr. 2 - příklad vazeb mezi prvky



Obr. 3 - nástin báze dnešního programování