EXTRÉMNÍ PROGRAMOVÁNÍ A METODICKÝ PŘÍSTUP

Pavel Drbal

Vysoká škola ekonomická, nám. W. Churchilla 4, 130 67 Praha 3, ČR, E-mail: drbal@vse.cz, stránky: http://nb.vse.cz/~DRBAL

1. Abstrakt

Podstatou článku je vysvětlení, proč jsou nutné metodiky. Na rozboru pojmu extrémního programování se ukazují výhody i meze nemetodického přístupu a důvody vzniku metodik.

2. Extrémní programování

V poslední době se objevují články (například [1],[2]), které ospravedlňují nesystematické programování, které se cudně přejmenovalo na extrémní programování. Vždy je s tím spojen určitý nádech údivu: „Podívejte se, ono to funguje, i když to v metodické literatuře zavrhují. A jak to je efektivní!“

Myslím, že mám právo psát o této problematice, protože jsem část svého života prožil ve skupině, která by se teď označovala jako používající extrémní programování. Navíc vidím na studentech a absolventech obojí, jak přirozený přístup k programování (nesystematický), tak i profesní přístup. Čím se vyznačuje extrémní programování?

- Neprovádí zjevnou analýzu a systémový design,
- Neudržuje dokumentaci.

Musí se odlišit extrémní programátoři a začátečníci. Vnějškově se jejich činnost podobá (nedělají formální analýzu a neudržují dokumentaci), je však mezi nimi podstatný rozdíl – programátorům programy hladce fungují, začátečníkům nefungují nebo fungují kostrbatě.

Na extrémní programování lze pohlížet z mnoha stran, některým se budeme věnovat.

2.1 Vnější pohled

Programátorovi se zadá úloha, ten se několika kontrolními otázkami přesvědčí, že dobře porozuměl, sedne si k počítači a začne psát příkazy programu. Za několik dnů (někdy i týdnů – podle velikosti) přinese program, který funguje a i dělá něco navíc.

Vytvořený program je jiný, než bylo požadováno, programátor však přesvědčivě vysvětlí, že tak to je lepší – šéfovi nezbude než jeho vysvětlení akceptovat, protože má pravdu.

Ovšem, takový programátor je ve skupině jen jeden, ostatní nejsou schopni těchto výkonů. Důsledkem bývá, že ve skupině produkující programy je jen jeden nebo dva výkonní programátoři, kteří jsou přepracovaní, a ostatní dělají nedůležité programy a závidí jim společenské a platové zařazení.

2.2 Řeč čísel

Kdysi dávno byl tento fenomén statisticky zkoumán. Ukázalo se, že rozdíl v produktivitě programátorů je neporovnatelný s jinými profesemi. Rozdíl v produktivitě nejhoršího a nejlepšího programátora byl dvacetinásobek. Jinými slovy, špičkový programátor nahradí 20 málo výkonných programátorů – přičemž i ti neméně výkonní programátoři stáli zaměstnavatelům za to, že je platili (tato statistika vznikla v USA).

2.3 Známá východiska

- Autoři výše uvedené statistiky došli k závěru: „**Lepší žádný programátor než špatný programátor!**“

To ovšem není konstruktivní řešení.

- **Zaměstnávat jen špičkové programátory**.

To také není všeobecně použitelné řešení. Špičkových programátorů je málo, v okresním městě rozhodně nepostavíte tým z pěti špičkových programátorů, prostě proto, že jich v tom městě tolik není.

Další důvod proti takovému špičkovému týmu je to, že vytvoříte situaci „**mnoho kohoutů na jednom smetišti**“. Špičkoví programátoři bývají výrazné a svérázné osobnosti (lidsky řečeno, nikdo se s nimi nesnese[1]), velmi špatně se zařadí jako řadoví členové do týmu. (Když se podíváte po jejich osudech, většinou končí jako systémáci, správci sítě nebo databáze – prostě v takové posici, kdy si určují práci sami a nemají ani podřízené.)

- „**Chirurgický tým**“

Tento programátorsky tým je nazván podle analogie z medicíny, ta jeho vznik inspirovala.
Špičkový programátor je šéfem týmu, je jediný, který rozhoduje. Ostatní členové týmu jej zbavují všech rutinních prací. Kódovači kódují rutiny, které jim šéf přikázal naprogramovat, testeři rutiny a průběžné verse prověřují, dokumentátor vytváří dokumentaci a podobně. Všichni členové týmu musí být poslušní a přesně vykovávat šéfem přidělenou práci.

Teoreticky má být v týmu asistent šéfa – člověk stejných kvalit, který je schopen v případě potřeby převzít funkci šéfa (je mladší a učí se na šéfa). *Neviděl jsem programátorský chirurgický tým, který by měl stálého asistenta.*

Je to výborný způsob práce, který má úspěch, když firma dokáže zajistit šéfovi autoritu (přirozeně takové týmy nevznikají, respektive velmi brzy se rozpadnou).
Zkušenost ukazuje, že šéf chirurgického týmu ani nemusí umět programovat, stačí, když to je výborný analytik a má cit pro architekturu programů.

Chirurgický tým se rozpadne, když kódovači získají sebevědomí a chtějí se stát odpovědnými programátory. Milostné vztahy také uspíší konec.

- **Podpůrná technika**

Nejdříve je nutno uvést určité objasnění.

[1] Nikdo se s nimi nesnese – kromě obdivuhodných žen – jejich manželek.

Problém programování je problémem zvládnutí složitosti úlohy. Složitost se špatně měří, nakonec se jako míra složitosti používá počet řádků zdrojového kódu (není to moc dobré měřítko, ale lepší takové než žádné[2]).
Mluvíme-li o mezích programátora, mluvíme o tom, kolik řádků zvládne tak, aby si udržel přehled, byl schopen opravovat a vkládat změny. Každý z nás má určitou mez, někdo zvládne 200 řádek, jiný tři tisíce a další deset tisíc. Z hlediska zvládnutí je jedno, v čem jsou ty řádky – může to být assembler nebo třeba Cobol. Ovšem tři tisíce řádků v assembleru je spíše malý program než střední, ale tři tisíce řádků v Cobolu je už docela slušný systémek.

Jedna z cest jak zvýšit výkon programátora je dát mu k disposici (programovací) jazyk a jiné podpůrné prostředky skutečně vysoké úrovně. Proto tedy se prodávají různé buildry, databáze, testovací systémy a podobně.

Nakoupíme-li výkonnou podpůrnou programovací techniku, svede jeden výkonný člověk nahradit práci celého týmu v lacinějším vývojovém prostředí.
Vlastně to je taková modifikace chirurgického týmu, jen místo kódovačů a testerů máme podpůrné programové systémy.

2.4 Samodokumentace

Existuje směr v programování, který tvrdí: „**nejlepší dokumentací je program sám (nejlépe ve strojovém kódu)**“. Je na tom hodně pravdy. Program podléhá nejrůznějším změnám, opravám, úpravám, rozšířením. Občas se tyto změny z nejrůznějších příčin nepromítnou do dokumentace nebo se promítnou nedůsledně. Expresivně lze říci, že dokumentace lže!

Zápis algoritmu ve vyšším programovacím jazyce také nemusí plně odpovídat realizaci. Uvědomme si, že překladač je jenom program, který byl dělán lidmi. Sice zřídka, ale přece jenom, se stane, že vygenerované strojové instrukce neodpovídají tomu, co bylo zamýšleno ve zdrojovém zápisu. *(Je to druh sportu, hledat chyby v překladačích. Vyšší druh sportu je hledat chyby v logickém návrhu počítače – podařilo se mi to dvakrát v životě.)*

Na samodokumentaci je hodně správného, ovšem rozebírat strojový kód programových systémů současnosti je prakticky nemožné.

„Teorie“ samodokumentace se projevuje jako trend, projevuje se čitelností zdrojových programů – to je neodmyslitelná zásada nově vznikajících jazyků. (*Ovšem je více trendů, které formují programovací jazyk a které jsou navzájem rozporné – typický je rozpor mezi čitelností a stručností.*)
Extrémní programátoři vyznávají samodokumentaci.

2.5 Inteligence

Psychologové říkají, že u lidí existuje asi sedm druhů inteligence (někteří autoři uvádějí 12 či 20). Tyto inteligence jsou dosti různé, například empativní, představivost, umělecké cítění (*nejsou to vlastnosti ale ucelené komplexy vlastností*). Každý z nás je od přírody či boha namíchán svou vlastní směsí těchto inteligencí – každou z nich máme v určité míře. A právě naše originální směs inteligencí určuje, zdali jsme vhodní pro jednotlivé role ve společnosti.

[2] Trochu problém zjednodušuji, přistupte, prosím, na toto zjednodušení.

Těmito tématy se zabývá několik věd, pro nás z jejich poznatků vyplývá, že dobrý programátor musí mít od přírody namíchány inteligence v určitém poměru – s převahou té, kterou bych nazval „**racionální inteligence**“.

Pokud dospělý člověk nemá racionální inteligenci, žádná školení ji v něm nevypěstují. Lze trénovat inteligenci určitého druhu – tak jako můžeme cvičením rozvíjet svou svalovou hmotu, můžeme i cvičit inteligenci určitého druhu. Bohužel vývoj inteligence je vázán na určitý věk; při úrovni našeho školství (škola přestává učit žáky učit se) bude lidí s racionální inteligencí stále méně. Nedá se nic dělat, musíme nají způsob jak tuto situaci zvládnout.

S danou situací se lze vyrovnat dvojím způsobem:
- ⇒ Rozvinout procesy jak vyhledat a přeplatit lidi s racionální inteligencí.
- ⇒ Vytvořit procedury vývoje systémů (programování) tak, aby je zvládali lidé širšího spektra inteligencí.

2.6 Strukturované programování

Správný extrémní programátor se pozná podle těchto projevů:
- ⇒ Říká: „Lepší než sahat do cizího programu je to vše znovu napsat!“
- ⇒ Vyjadřuje hluboké opovržení ke všem metodikám a způsobům psaní dobře strukturovaných programů.
- ⇒ Své programy píše rychle a velmi dobře se v nich dokáže orientovat.
- ⇒ Je vždy velmi zaměstnán a když na něj promluvíte, utrhuje se.

Samozřejmě musí mít velmi dobrou schopnost abstrakce, musí mít ve své hlavě zásobník se 4 či 5 rozpracovanými úrovněmi řešené úlohy, mezi kterými umí přepínat, to musí umět každý dobrý programátor. Trochu uniká, že musí mít ve své hlavě databanku struktur pro řešení jednotlivých problémů.

Zkušený programátor neřeší jednotlivé problémy své úlohy, ale pouze vyhledává ve své paměti struktury, které se mu v minulosti osvědčily. Někdy narazí na nový problém, věnuje čas na jeho řešení a strukturu řešení zařadí do své databanky v paměti. Čili takový programátor při psaní programu nic neřeší, jen používá jeden mustr za druhým. Kvalitu programátora určuje to, jak umí příslušnou strukturu vybrat a kolik jich má ve své paměti vhodných pro řešení předložené úlohy. Tím je vysvětlena rychlost psaní programu i rychlá orientace v něm – stačí v programu rozpoznat mustr a tím je řečeno co se zde řeší a i jaké jsou podrobnosti (u jednoho mustru to je vždy totéž).

Podstatným problémem je to, že ty mustry nejsou sdělné. Každý si je vytvoří podle svých zkušeností, vědomostí a zálib. Proto se jeden programátor nemůže orientovat v programu někoho jiného. (*K mustru patří nejen struktura řešení, ale také konvence jmen, výběr příkazů, členění řádku s příkazy. Například znám programátora, který všechny cykly píše jako věčné a ukončuje je zásadně pouze pomocí „break“.*)

Tyto lidské vlastnosti vysvětlují většinu projevů extrémních programátorů.

Dobře strukturované programy

Z předchozího textu lze vytknout dva hlavní problémy:
- ⇒ dlouhá doba výuky profesionála,

⇒ nesdělnost struktur.

Oba tyto problémy řeší „Strukturované programování“[3]. Strukturované programování podstatně zkrátí dobu získávání zkušeností tím, že nabídne sadu struktur vhodnou pro většinu možných situací. Není nutné získávat zkušenosti pomocí chyb a omylů. Tyto struktury jsou zveřejněny, všichni se učí tytéž struktury a obraty, takže se spolu dobře domluví.

Strukturované programování není všelék, je výhodné jen pro řešení těch úkolů, pro které bylo vytvořeno.

Strukturované programování je prostředek pro zvládnutí jednotlivých procesů.

Strukturované programování je vhodné pro psaní klasických programů, tj. programu s jednorázově zadanými vstupy (dávkovými vstupy) a jednorázově poskytnutým výstupem. U interakčních programů je vhodné k zachycení jednotlivých komunikací, nepopíše interakční systém jako celek (pro to je specializován objektový přístup). Ve schématech strukturovaného programování je vzorec pro konstrukci víceuživatelského přístupu.

Strukturogram popisuje proces bez souvislosti s daty.

Strukturogram zachytí jeden proces, byť velmi složitý, není to prostředek pro popis paralelně běžících procesů.

Strukturogram má ekvivalenty v jiných prostředcích, jako jsou „Přechodové diagramy“, „Akční diagramy“, „Diagramy datových toků“; je však mezi nimi jeden podstatný rozdíl. Výše zmínění prostředky popisují obecný proces jako obecnou síť. Jsou to prostředky pro popis reálných procesů. Strukturogram je oproti tomu prostředek pro popis vytvářených sítí, a to vytvářených tak, aby byly lidmi dobře zvládnutelné. Strukturogram popisuje serioparalelní síť. Důvod je zřejmý, program bez GOTO a s dodržením zásady jediného kontextu je serioparalelní síť.

Postup užití strukturogramů (tj. strukturované programování) je ten, že zamýšlený program (který si obvykle představujeme jako obecnou síť) popisujeme strukturogramem – což ovšem znamená, že zamýšlenou síť transformujeme na serioparalelní – a tu je již snadné programovat (je to 1:1).

Převést obecnou síť na serioparalelní (vytvořit strukturogramy) je vždy možné – to je matematicky dokázáno.

Postup převodu obecné sítě na strukturu je například v knize “Object-Oriented Modeling and Design” [3], kde se nejdříve činnost objektu popíše pomocí přechodového diagramu, pak se převede do strukturogramu a nakonec se předá programátorům. Česky je tento postup uveden ve skriptech „Objektově orientované metodiky a technologie“ [4]. Jiný příklad omezení konstruovaných sítí na serioparalelní je v Jávě, multithreading je sestrojitelný jen jako serioparalelní síť.

Jak pracovat s extrémními programátory

[3] „Strukturované programování“ je zkrácený název, plný text zní „Tvorba dobře strukturovaných programů“; může totiž existovat i špatná struktura programu.

Dosud jsme diskutovali vlastnosti extrémních programátorů, které umožňují jejich výkonnost. Nyní budeme mluvit o organizaci výrazných osobností. V podstatě jsou dvě cesty:

⇒ isolované úkoly,

⇒ stálý kolektiv.

2.7 Isolované úkoly

To je nejobvyklejší způsob zaměstnání extrémního programátora. Programátorovi se svěřují jednotlivé úlohy v tom rozsahu, který zvládá. Ideální způsob zadání úlohy je tento:

Šéf: „Bylo by třeba vyřešit to a to. Myslíš, že to zvládneš?“

Programátor: „Ano“ (Nemůže říci „ne“, to mu hrdost nedovolí.)

Šéf: „Bylo by to třeba zvládnout do tehdy a tehdy. Dokážeš to?“ (Naznačí velké tlaky z vnějšku.)

Programátor: „To je těžké, je to spousta práce, lepší by byl dvojnásobný termín.“

Šéf neustává v nátlaku, buď naznačí, že za svou práci bere plat, nebo slíbí prémii, nebo (a to je nejlepší) pochválí jeho inteligenci a řekne, že jiný člověk není tento úkol absolutně schopen zvládnout.

S takovými lidmi je jedna potíž. Jsou hodně pracovití (neboť své zaměstnání mají za koníčka) a vymýšlejí si pro sebe práci, kterou vylepšují firemní systém – takže každý oficiální úkol je pro ně vlastně práce navíc. Šéf musí tyto věci vědět a buď je podporovat nebo je zamezit, šéf musí umět získat informace o pracovní aktivitě a musí být schopen rozpoznat její důležitost.

Další nebezpečí je ve studiu. Programátor–individualista může být tak ješitný, že se přestane vzdělávat a stane se po čase bezcenným.

Takto se obvykle zachází se systémovými programátory. Jestliže šéf přestane programátorovi lichotit, tento se dostane do stresu, začne mít pocit, že je zneuznáván a pošilhává po jiném místě, byť z lenosti hned neodejde. (Jiná možnost je, že se stáhne do ulity a čeká, až mu vymění šéfa. Většinou se do čtyř let dočká.) Peníze zde obvykle nehrají nejdůležitější roli.

2.8 Kolektiv programátorů

Jiná věc je kolektiv programátorů. Tam je velmi důležité, aby přidělování úkolů běželo po neoficiální (přirozené) hierarchii.

Každá stálejší skupina lidí má dvě řídicí hierarchie. Jedna je oficiální, určená pracovním řádem (šéf, zástupce, šéfovi oblíbenci). To je cesta, po které jsou přidělovány peníze a úkoly. Druhá hierarchie je přirozená. Ve skupině se po čase vynoří člověk, který má největší autoritu, pak se vytvoří skupinka, s jejichž členy vůdce diskutuje a radí se, ti jsou obklopeni lidmi, kteří jsou touto skupinou bezprostředně řízeni, a nakonec je skupina pouze ovlivňovaná. Informace o takových hierarchiích najdete v každé skupině sociologie (psychologie práce). Autor tohoto textu se necítí být sociologem, zná však několik takových programátorských skupin.

Takový kolektiv programátorů pracuje velmi účinně, vytváří produkty efektivně a dokáže být na špičce vývoje. Musí však být splněno několik podmínek.

- Je to dlouhodobá záležitost. Skupině musí být dán čas, aby se neformální hierarchie vyvinula. Nemá smysl vytvářet takovou skupinu pro úkol s termíny počítanými

v měsících. Kolektiv je vlastně organismus, po svém vzniku nějakou dobu dospívá, pak má dlouhou etapu produktivní činnosti a nakonec začne skomírat.

- Členství v kolektivu je dlouhodobá záležitost. Takový kolektiv nemůže existovat při výrazné fluktuaci. Členství v kolektivu musí být doživotní (rozumí se profesionální produktivní život, ne fyzický život). Musí existovat takové podmínky, které nedovolí odchody lidí. Může to být zprostředkováno penězi, jsou však i jiné možnosti.
- Šéfovat takovému kolektivu je velmi nevděčné, protože šéf vlastně jen zprostředkovává styk přirozeného vůdce a jeho skupinky s oficiální hierarchií. Takový člověk se hledá těžce, normální šikovný vedoucí vyrazí všechny lidi převyšující jej inteligencí a autoritou a je pak neohrožený vedoucí oddělení hlupců. Na druhé straně přirozený vůdce kolektivu nebývá ochoten ztrácet čas diskusemi s různými řediteli a presidenty, takže roli oficiálního šéfa nemůže převzít. (U programátorů to platí jednoznačně, porada u ředitele je pro programátora neospravedlnitelná ztráta času. Po čtyřicítce, když už vlastně nemůže být programátorem, to pak funkce bere.)
- Dlouhodobá pracovní náplň. Kolektiv (podobně jako člověk) se stane specialistou na řešení úloh z určité oblasti. Náhle změnit oblast řešených úloh se rovná likvidaci kolektivu a tvorbě nového.

Když srovnáte výše uvedené podmínky se současnou realitou, žádný kolektiv programátorů nemůže existovat. Takové kolektivy existovaly za socialismu, existuje však jedna výjimka, kdy takový kolektiv existuje i dnes, a to jsou začínající firmy.

Jestliže skupinka mladých lidí založí firmu tak, že přirozený vůdce se stane i oficiálním vedoucím (majitelem nebo ředitelem), pak má tento kolektiv velkou šanci na úspěch. Je to kolektiv, který má jeden cíl – prosperitu firmy – a je ochoten tomuto cíli hodně obětovat bez ohledu na peníze. Kolektiv má úspěchy, firma se rozrůstá, až převýší určitou mez a již ji nelze řídit kamarádskými domluvami. Nastává krize firmy a záleží na kvalitách vedení, jak se touto fází projde.

Jestliže čtete životopisy nebo autobiografie úspěšných lidí v hospodářské oblasti, kteří začínali od píky (např. [5]), vždy je tato fáze vývoje popisována.

3. Krize programování

Když se vytvářel velký systém v krátkém čase, tak to skoro vždy dopadlo špatně. Ve velké části případů práce nebyly dokončeny, pokud byly dokončeny, tak to bylo s mnohonásobně většími náklady než se plánovalo a s mnohonásobně překročeným termínem dohotovení. Jen malé procento bylo dokončeno včas a za rozumné peníze. Tento jev konstatovalo velké množství článků, vedlo se kolem toho mnoho diskusí, které hledaly východiska. Východisko bylo nalezeno, nejdříve se však podívejme na důvody této krize.

Důvod je zřejmý. Je to podobně jako se zedníky. Jestliže jeden zedník postaví zeď za tři dny, tři zedníci postaví zeď za jeden den, pak není pravda, že 480 zedníků postaví zeď za tři minuty. Jinými slovy, hledala se organizace práce, ovšem organizace v širším slova smyslu, ne pouze organizace lidí, ale hledaly se i pracovní postupy, pomůcky apod.

Musíme si přiznat, že velké informační systémy jsou v určitém smyslu to nejsložitější, co se lidstvo snaží vytvářet. Samozřejmě existují složitější věci než informační systémy, například mozek, lidské tělo, lidská společnost, život vůbec, vesmír. Tyto systémy se ale snažíme pouze poznat, máme ještě daleko k tomu, abychom je mohli vytvářet. Existují systémy, jejichž

vytváření je svou obtížností srovnatelné s informačními systémy – například samotné počítače – ale zde jsou obtíže jiného charakteru, nelze si z nich vzít poučení. U všech systémů, které mají fyzickou representaci, tvoří logická stavba systému jen menší část obtíží, větší část obtíží tvoří fyzikální vlastnosti použitého materiálu. U stavby budov to je statika, pevnost a tuhost, u stavby přehrad hydrodynamika, u stavby počítačů vodivost, indukce, kapacita. Tvorba programů však není omezena materiálem, jeho fyzikálními vlastnostmi a jeho stárnutím.

Opravím tvrzení ze začátku předchozího odstavce. Všechny velké (úspěšné) projekty jsou v podstatě stejně obtížné, ale pouze u informačních systémů veškerá složitost je v logice konstrukce.

Právě z tohoto důvodu si programování muselo vyvinout vlastní postupy, jen v malé míře mohlo přebírat zkušenosti jiných oborů.

Když se lidstvo snaží ovládnout nějakou novou oblast, je postup zvládnutí vždy v podstatě stejný. Probíhá přibližně v těchto etapách:

- **Doba géniů**. Inteligentnímu člověku přeje náhoda a on objeví nový poznatek. Je to náhodný proces, který proběhne několikrát v dějinách a ujme se teprve tehdy, když je společnost připravena jej přijmout. (Například již staří Egypťané uměli využít páru k mechanické práci, ale práce otroků a nevolníků byla lacinější až do Wattovy doby.)
- **Doba umělců a kouzelníků**. Jen zvlášť vybavené osoby dokáží problém zvládnout.
- **Doba diletantů**. Kdekdo se v dané oblasti exponuje, ale jen málo je úspěšných.
- **Doba pravidel**. Pro zvládnutí problémů se vypracují pravidla, která jsou vzata ad hoc ze zkušenosti. Vznikají rituály, pověry a mystika. Úspěšné sady pravidel jsou profesním nebo rodinným tajemstvím. Severoameričané mají pro tuto etapu velmi užitečné přísloví: „Špatná pravidla jsou lepší než žádná.“
- **Doba metodik**. Prověrka jednotlivých sad pravidel praxí přináší poznání vnitřní logiky oblasti a umožní vydělit obecná pravidla, která zvyšují pravděpodobnost úspěchu. Vznikají metodiky, které sestávají z jednotlivých technik a podpůrných prostředků a z postupů, jak tyto používat. To vše je doplněno obecnými pravidly. Metodiky nejsou homogenní, některé části jsou rozpracovány více, jiné obsahují jen relativně obecná doporučení. Doplňováním a upřesňováním přerůstají do další fáze.
- **Doba kuchařek** – technologických předpisů. Technologický předpis (na rozdíl od metodiky) neponechává prostor lidské iniciativě. Obsahuje podrobné popisy toho, co musí jednotliví pracovníci provést, aby výrobek (například informační systém) byl úspěšně vyroben. Technologický předpis celý výrobní proces rozkládá na posloupnost akcí úzce zaměřených specialistů.

Uvedený postup vývoje metodik je samozřejmě schématický, skutečnost je složitější. Jednotlivé etapy se překrývají, postup se větví, existují slepé uličky a cykly. Proces se rozbíhá do šíře, vznikají nová odvětví s vlastními postupy. Tak například středověké kovářství se rozvinulo v hutní výrobu (prvovýroba), těžké a lehké strojírenství.

Metody a technologie jsou navzájem svázány. Je to tak, že metodika je relativně nekonkrétní, ale obecná – pokrývá širokou oblast. Technologický předpis je naopak konkrétní, ale zaměřen na úzkou oblast použití. Jedna metodika se transformuje na několik technologických předpisů. Vlastně každá aplikace metodiky při tvorbě informačního systému znamená vytváření technologického předpisu pro jeden konkrétní případ (protože například začít analýzu problému aniž je jasné, kdo a kdy bude systém implementovat, je poněkud hochštaplerské).

Krize programování představuje nutnou etapu vývoje programování. Je to etapa hledání a ověřování pravidel – tedy tvorba metodik. Tato etapa je v podstatě dokončena, myslím si, že potřebné metodiky byly nalezeny a nyní jsme v etapě tvorby technologických předpisů.

4. Metodiky

Nebudeme se zde zabývat historií, přesto však zde zmíním několik „objevů“, ze kterých metodiky těží.

Prvním z nich byl „rozděluj a panuj“, který vyústil v hierarchický rozklad – to je všeobecně používaný přístup používaný i v běžném civilním životě.

Druhým byl princip „dobře strukturovaných programů“, který vyústil v paradigma programovacích jazyků (tj. každý jej nevědomky používá; například příkazy „continue“ a „try“ jsou aplikace strukturovaného programování).

Třetím principem (to pořadí je pouze pořadím odstavců, jednotlivé principy se hledaly a rozvíjely paralelně) je několikanásobný pohled na vytvářený produkt. Přišla s tím metodika „strukturované projektování“, která zavedla dva pohledy – datový a procesní. Nyní se počet používaných pohledů blíží desítce.

Čtvrtý přístupem jsou objekty. Asi dvacítku let cestovaly v odborném tisku a po konferencích jednotlivé zásady (například utajování informací, rodiny programů apod.), které se nakonec dočkaly shrnutí ve formulaci objektových principů. Nejdříve se zavedlo objektové programování, po jeho úspěchu vznikl objektově orientovaný přístup k projektování. Všechny moderní metodiky jsou objektové.

Nakonec se objevila třídimensionální struktura metodik:

1. granulita (systém – podsystém – objekt – metoda),
2. časové členění (etapy: rozbor zadání, analýza, systémová architektura, design, implementace),
3. paralelní pohledy (hlavně statický a dynamický).

Podrobněji viz [6] a [7].

Myslím si, že bouřlivý vývoj metodik je ukončen a na pořadu dne je tvorba technologických předpisů (kuchařek). Za prvý takový předpis s širokou publicitou lze považovat RUP [8].

Vývoj metodik není ukončen, v současné době se formulují metodiky pro internetové aplikace, avšak zásadní metodické přístupy jsou již známy.

Podíváme-li se na věc s nadhledem, vidíme následující pyramidu:

1. Vrcholek pyramidy (nejvíce abstraktní)¨jsou principy boje se složitostí – to jsou všeobecně uznávané principy (viz například [9]).
2. Střední vrstvu tvoří metodiky, každá pokrývá určitou oblast.
3. Základnu pyramidy tvoří technologické předpisy. To jsou vlastně metodiky aplikované na konkrétní prostředí.

5. Závěr

Celkový trend vývoje je zřejmý. Směřuje od „umělecké“ činnosti kouzelníků a mágů k pásové výrobě ve velkém kolektivu „dělníků“ – stejný trend mají všechny lidské činnosti zaměřené

na produkci výrobků. V intencích tohoto trendu je také to, že na začátku jsou zapotřebí lidé s vysokou inteligencí a velkými tvůrčími schopnostmi. Nároky se postupně snižují, zvyšuje se zastupitelnost pracovníků, až to končí s relativně nekvalifikovanými pracovníky u pásu. Původní požadavek tvůrčího nadání a všeobecné inteligence je nahrazen požadavky na zvládnutí jednoduchého technologického předpisu pro omezenou oblast jedné role pracovníka. Potřeba kvalifikovaných lidí nemizí, ale přesouvá se na jiné, ještě komplexnější pole.

Pro současnou praxi vyplývá, že máme k disposici, schématicky vzato, tři způsoby výroby informačních systémů:

- ⇒ Máme k disposici soukromé technologie umístěné v hlavách svých extrémních programátorů (to se vyplatí tehdy, jestliže programy mají krátkou životnost).
- ⇒ Podle obecné metodiky specializovaní pracovníci navrhnou technologii vhodnou pro konkrétní řešenou úlohu (to je užitečné při unikátní úloze).
- ⇒ Podle koupené (vytvořené) technologie se hromadně vyrábějí systémy téhož typu (vyplatí se to velkým firmám).

Literatura:

1. Berger J. Štěpánek P.: Extrémní programování v praxi, proceeding of the OBJEKTY'2001, Praha, 2001, ISBN 80-231-0829-X.
2. Biggsová m.: Extrémní praktiky programátorů, COMPUTERWORLD #36, IDG Czech, Praha19-25.10.2001, ISSN 1210-9924
3. Rumbaugh J.: Object-Oriented Modeling and Design, Prentice-Hall 1991
4. Drbal P. a spolupracovníci: OOMT Objektově orientované metodiky a technologie, skripta VŠE, Praha, 1997, 300s. ISBN 80-7079-740-1
5. Brown,M.: Richard Branson, Euromedia Group 2000, ISBN 8-2-42-0319-7
6. Drbal P.: Metodika klasifikace metodik (The Method of Methods Classification), proceeding of the TVORBA SOFTWARE'2001, Ostrava, 2001, ISBN 80-85988-59-3
7. Drbal P.: Jak vytvořit a zkontrolovat vlastní metodiku (The Creating and Testing of The Special Method), minicourse, proceeding of the OBJEKTY'2001, Praha, 2001, ISBN 80-231-0829-X
8. Jacobson I., Booch G, Rumbau J.: The Unified Software Development Process, Addison-Wesley, 1999. ISBN 0-201-57-169-2
9. http://nb.vse.cz/~drbal