CASE

Pavel Drbal

Zkratka CASE znamená Computed Aided Software Engeneering a označuje určitý obor činnosti, který by se snad mohl nazvat „Integrovaná tvorba programů pomocí programových prostředků". Jako každý módní výraz je používán v různých situacích a k označování různých jevů, ať již právem nebo neprávem. Jako „CASE", přesněji „CASE–tools" lze označit skoro všechny prostředky, kompilátory libovolných programovacích jazyků počínaje a úzce specializovanými prostředky konče.

Prvý dojem je, že to je kaskáda prostředků různé úrovně, které na sebe navazují. Podobně jako to je třeba u systému Turbo C++, kde existuje překladač z jazyka C++ do jazyka C, z jazyka C do Turboassembleru, z TASM do tvaru cílových modulů atd. Analogicky lze usuzovat, že existuje jakási úroveň A, ve které vyjádříme daný problém. Toto vyjádření se automaticky převede do úrovně B, kde to doplníme o některé podrobnosti, pak je naše řešení převedené do úrovně C atd., až je vyjádřeno v některém běžném programovacím jazyce. Tento dojem není nejpřesnější.

Jiný dojem je ten, že prostředky CASE jsou nezávislé prostředky různých úrovní, určené pro různá použití – že je mezi nimi vztah jako mezi Pascalem, Cobolem a C – každý je pro jiný druh uživatelů a úloh, jiné úrovně abstrakce, ale v podstatě téhož druhu. Také tato představa není úplně přesná. Samozřejmě, produkty různých firem se od sebe liší – ale základní výrazové prostředky (funkční diagram, datový model) si jsou velmi podobné – vycházejí totiž z několika málo metodik rozšířených v literatuře.

Pro popis prostředku CASE se velmi hodí následující indická historka. Přivedli tři slepé muže ke slonu, aby získali vlastní představu o tom, co to slon je. Prvý muž se dotkl chobotu a říká „Slon je had". Druhý nahmatal nohu a říká „Slon je sloup". Třetí stál u břicha a říká „Slon je teplý balón". Při řešení úlohy jsme v podobné situaci jako tito slepci. Nevíme ještě, jaké bude řešení úlohy, ale víme (resp. můžeme zjistit), jak by řešení úlohy mělo z jednotlivých pohledů vypadat.

Použití prostředků CASE umožňuje následující kroky:

- formulace řešeného problému z jednotlivých hledisek (dále jsou vyjmenována která),
- zjištění rozporů mezi jednotlivými hledisky,
- odstranění těchto rozporů.

Dá se to také říci tak, že prostředky CASE slouží k vytvoření několika modelů řešení. Tyto modely se vytvářejí podle různých hledisek (např. datové, funkční). Podstatou pomoci prostředků CASE je to, že umožňují tyto modely porovnat, odstranit rozpory a vytvořit jeden společný model, který vyhovuje ze všech použitých hledisek. Tento společný model slouží k zadání programů, které vytvoří řešení úlohy.

Jaká hlediska jsou používána:

- funkční
- datové
- komunikační
- řídicí
- organizační
- presenční
- konstrukční

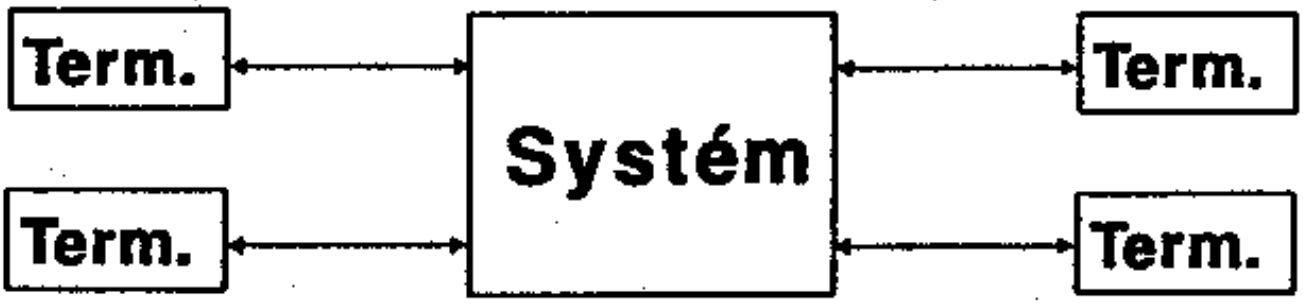
Charakterizujme je poněkud podrobněji.

Modely

Funkční model

Ve funkčním modelu je systém určen komunikací se svým okolím prostřednictvím datových toků. Vstupem tohoto modelu jsou jasně vymezené prvky okolí, které komunikují se systémem, a datové toky, kterými je tato komunikace realizována.

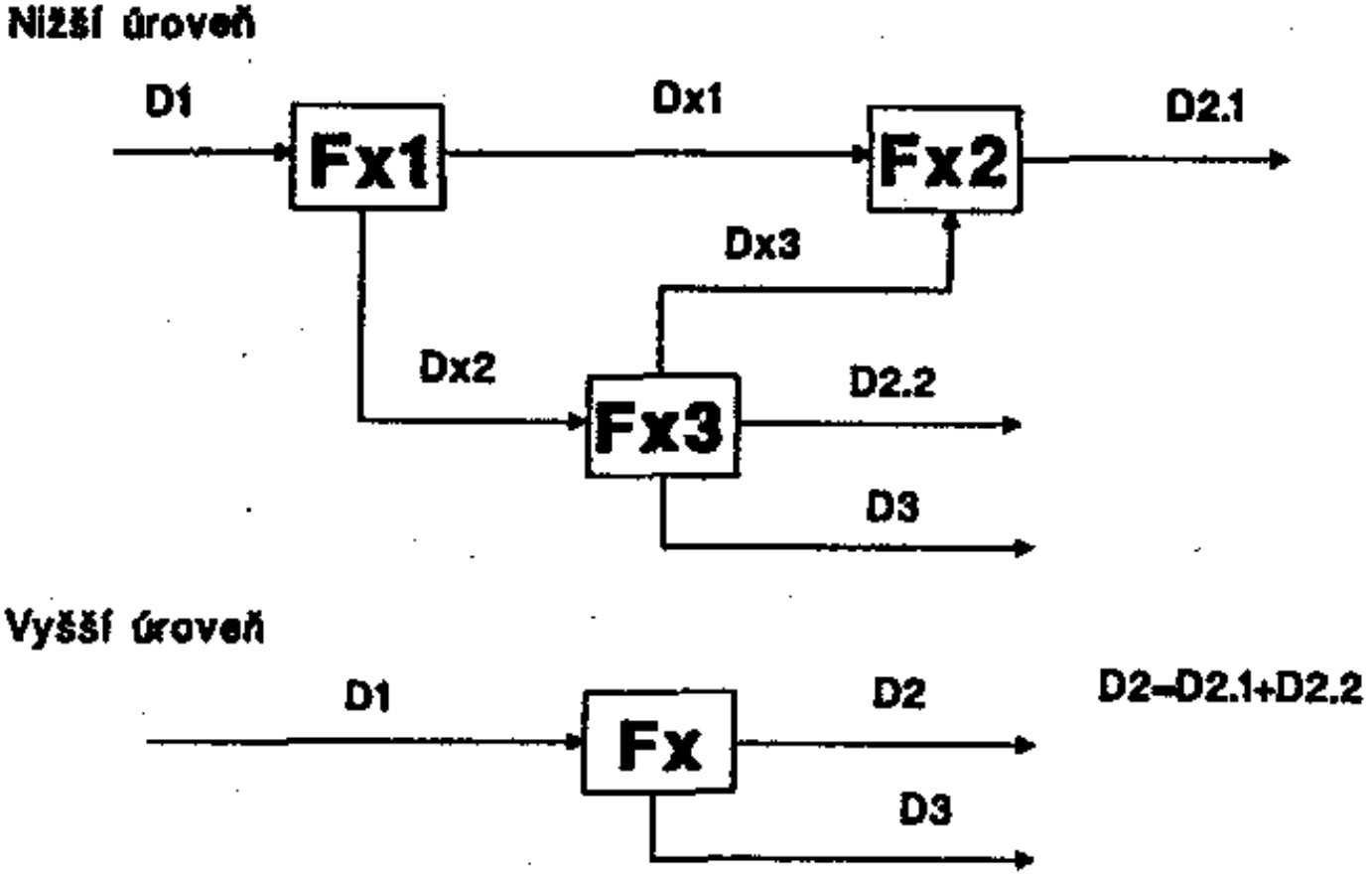
„Term." označuje terminátory, prvky okolí, „Systém" označuje vytvářený model systému.



Komunikace může být jednosměrná (do systému, ze systému) nebo oboustranná, komunikace může být realizována i více datovými toky jedním směrem.

Za předpokladu vyjasněnosti komunikace s okolím, můžeme model systému rozkládat na funkce (tj. podsystémy) a určovat potřebnou komunikaci mezi nimi. Tyto podsystémy

prvé úrovně lze zase rozložit na podpodsystémy (druhé úrovně) atd... Takto získáme model systému v několika úrovních podrobnosti. Komunikace mezi funkcemi (tj. podsystémy) je zase prostřednictvím datových toků. Je samozřejmé, že funkce vyšší úrovně je na nižší úrovni rozložena na několik funkcí a že datový tok vyšší úrovně buď odpovídá datovému toku nižší úrovně nebo je sjednocením několika datových toků nižší úrovně. Viz následující obrázek.



Prvky modelu jsou:

- Funkce – transformují datové toky. Označují se kolečkem. V textu je použit obdélník, protože použitý editor nemá možnost nakreslit kolečko.
- Datové toky – přenášejí informace (data), označují se šipkami.
- Data story (česky snad sklady) slouží k úschově dat – nemění data. Může z nich vycházet datový tok jen s těmi údaji, které jsou v datových tocích směřujících dovnitř. Označují se horizontálně položenými obdélníky.

Při tvorbě funkčního modelu se klade důraz na jasné vymezení funkcí a jejich návaznosti, méně již na frekvenci jejich použití a přesný časový průběh.

Datový model

Datový model je soubor všech dat (údajů) potřebných k řešení. Údaje jsou členěny podle vzájemné závislosti. Tvorba datového modelu má tři základní cíle:

- uvést všechny údaje,
- omezit redundanci, tj. neuvést žádný údaj dvakrát,
- zachytit nejdůležitější vztahy mezi daty.

Datový model vyšel z představ relačních databází, jeho použití je však širší (skutečná realizace nemusí být relační), používá však příslušnou terminologii. Jednotlivé údaje se nazývají atributy a jsou seskupovány do skupin nazývaných entity. Určitá skupina atributů jedné entity se považuje za klíč. Hodnoty atributů klíče jednoznačně určují jeden výskyt entity, tj. jednu instanci entity.

Datový model si lze představit i souborově. Entita representuje popis záznamu jednoho souboru, klíč entity je přístupový klíč souboru. V záznamu jsou popsány všechny údaje, které jsou tímto klíčem určeny.

Přehled základních pojmů:

Entita

- je popis záznamů jednoho souboru,
- je popis řádků jedné tabulky.

Instance entity

- je jeden záznam souboru,
- je jeden řádek tabulky.

Atribut

- je jeden údaj záznamu,
- je jedna položka řádku tabulky,
- je jedna proměnná.

Klíč entity

- je atribut (skupina atributů), jehož hodnoty jednoznačně identifikují (rozlišují) jednotlivé instance entity.

Různé entity mají různé klíče, tj. skupiny identifikujících atributů jsou různé – jeden atribut se může vyskytovat v několika klíčích různých entit.

Vztahy mezi entitami se realizují tím, že klíč jedné entity je uveden jako neklíčový atribut jiné entity.

Příklad

Máme například entitu FIRMA, která zahrnuje všechny obchodní partnery. Jejím klíčem je IČO, dále obsahuje další atributy, jako název, adresu ap. Máme také entitu FAKTURA, jejímž klíčem je číslo faktury. Vztah „faktura je určena firmě" je realizován tím, že mezi atributy FAKTURY je také atribut IČO (který zde není klíčem).

Jiný příklad

Mějme entity POPULACE a FIRMA. Instancí entity POPULACE jsou údaje o jednom člověku, klíčem je rodné číslo. Klíčem entity FIRMA je IČO. Vztah mezi těmito entitami – „člověk je zaměstnancem firmy" je realizován další entitou – ZAMĚSTNANEC, která má klíč ze dvou atributů: IČO a rodného čísla. Klíče všech tří entit jsou různé, i když mají společné části. Vztah mezi entitami ZAMĚSTNANEC a FIRMA je dán atributem IČO v entitě ZAMĚSTNANEC, který určuje vztah „je zaměstnancem které firmy". Tento atribut není klíčem, klíč se skládá ze dvou atributů, IČO je pouze jeho část.

Datový model má i grafickou representaci. Entity jsou znázorňovány čtverečky, vztahy mezi entitami jsou znázorněny čarami, které tyto čtverečky spojují.

Datový model zpracováváme tak, že jej normalizujeme, integrujeme a přizpůsobujeme funkčním požadavkům.

Normalizace je odstranění redundancí, integrace je spojení entit se stejnými klíči. Pak mohou následovat další úpravy, které v podstatě znamenají zvyšování redundance (tj. zdvojování dat), pokud to je vyžadováno z funkčních či časových důvodů. Příkladem zvyšování redundance je lokální číselník, který sice vyžaduje náročnější pravidelnou aktualizaci centrálního číselníku, může však podstatně zhospodárnit provoz, protože vyloučí časté odvolávky na centrální číselník.

Komunikační model

Také se mu říká „prototyping" nebo „slupkový". Je tvořen návrhem všech obrazovek, tj. je to model komunikace s uživatelem. Je podobný jazyku 4GL. Má tři hlavní složky:

- formátování obrazovek,
- výčet použitých dat,
- strukturu návaznosti obrazovek.

Formátování obrazovek znamená určit jakými obrazovkami bude systém komunikovat s uživatelem, jak na nich budou rozmístěny vysvětlující texty a vstupní a výstupní data.

Při umisťování vstupních a výstupních dat na pole (okénka) obrazovky se tato data charakterizují způsobem, který je využitelný v datovém modelu.

Také se určuje struktura návaznosti obrazovek, tj. jak obrazovky po sobě následují – například která podmenu následují po hlavním menu.

Komunikační model má dvě funkce. Jednak slouží k určení způsobu styku uživatele se systémem, případně i k oficiálnímu potvrzení, že systém je řešen způsobem, který objednávající vyžaduje. Druhou funkcí je prověření, zdali se na nic nezapomnělo při vymezování funkcí vytvářeného systému. Když si jednotliví referenti modelují styk se systémem, mohou se jejich požadavky poněkud lišit od abstraktního vymezení funkcí systému.

Řídicí model

Prostředky pro vyjádření řízení v systému nejsou úplně ustáleny, často se jich v jednom prostředku CASE používá více typů. Používají se:

- diagram řídicích toků,
- stavový diagram,
- diagram návaznosti obrazovek.

Diagram řídicích toků se graficky podobá diagramu funkčního modelu, aby se odlišily, používají se čárkované čáry místo plných. Uzly sítě – čárkovaná kolečka – označují rozhodovací algoritmy, čárkované čáry určují tok signálů – ne dat. Signál je podnět k výkonu funkce (tj. tvorby nebo zpracování dat). Diagram řídicích toků určuje časový průběh a další podmínky provádění funkcí. Oba způsoby znázornění (datových i řídicích toků) lze použít v jednom diagramu. Plná kolečka určují funkce nad daty, plné čáry tok dat. Čárkované čáry znázorňují signály. Čárkovaná čára (šipka) může vést do plného kolečka, pak určuje, kdy (za jakých podmínek) se má provádět funkce nad daty. Čárkovaná šipka z plného kolečka do čárkovaného kolečka znázorňuje, že zpracování dat dosáhlo určitého stavu (např. jsou – nejsou k disposici data, bylo dosaženo určitého limitu) a vydá o tom signál. Čárkovaná kolečka znázorňují zpracování signálů.

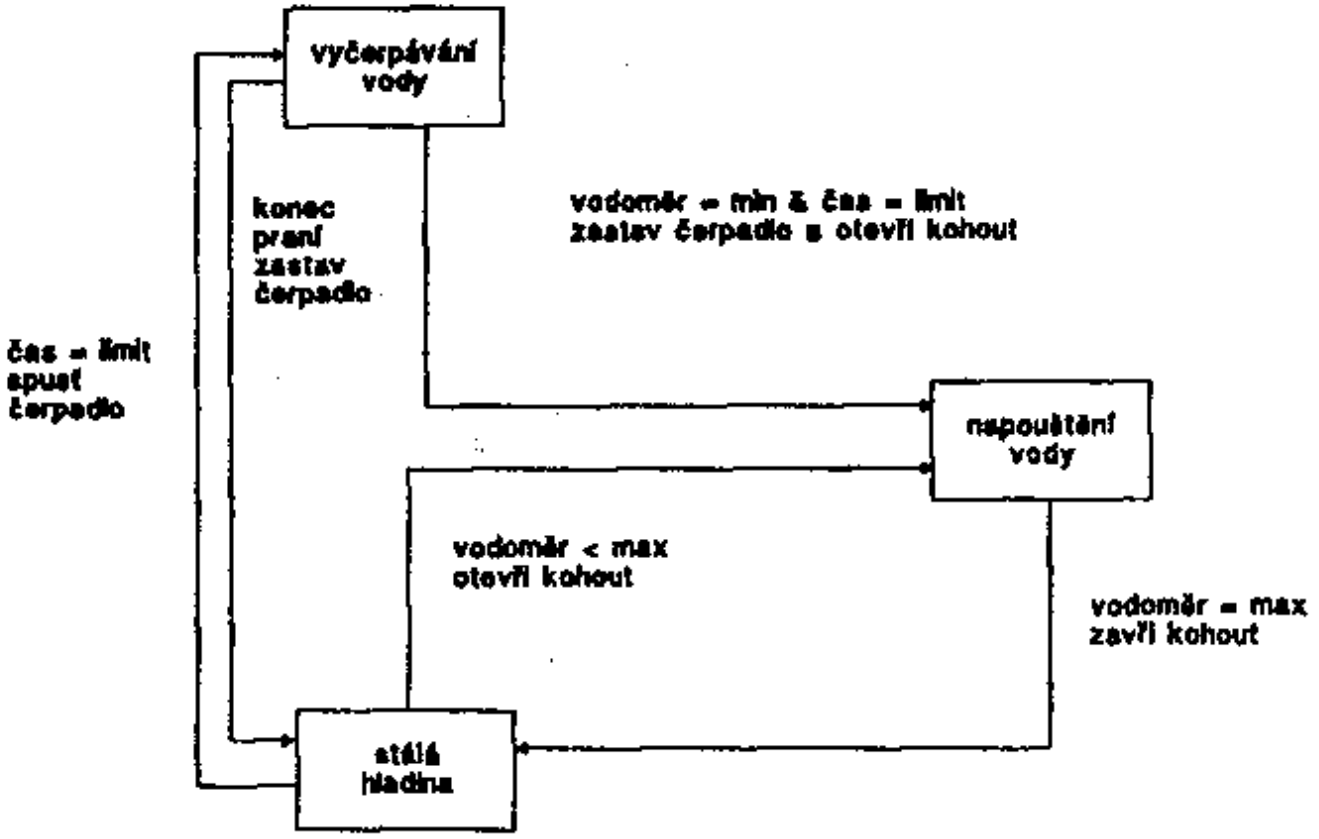
Takový diagram popisuje systém velmi dobře, málokdy však bývá přehledný a názorný, proto se často používají obě representace odděleně.

Stavový diagram popisuje stavy a přechody mezi nimi. Stav je časově „delší" činnost (nebo nečinnost) systému nebo jeho některé složky. Přechody mezi stavy jsou znázorněny šipkami. Přechod je charakterizován „zlomkem", který má ve svém „čitateli" podmínku přechodu a v „jmenovateli" akci přechodem vynucenou. Jako příklad stavového diagramu použijeme část popisu automatické pračky (na další straně).

V některých prostředcích CASE se diagram návaznosti obrazovek označuje jako diagram řídicích toků. To je určité zjednodušení, které je smysluplné jen pro tvorbu relativně malých systémů.

Organizační model

Tento model není nic jiného než „pavouk" organizace, tj. struktura vztahů podřízenosti v organizaci. V modelování má ten smysl, že poskytuje úplný výčet všech pracovišť



včetně jednotlivých referátů a že umožňuje výčet funkcí požadovaných jednotlivými pracovišti. Vazba na funkční i komunikační model je zřejmá.

Presenční model

S nádechem ironie je také nazýván „šéfovský diagram". Má dvě role.

Umožňuje presentovat navrhovaný systém lidem, kteří nejsou a nechodlají být odborníky přes jednotlivé modely. Prvky systému jsou znázorněny piktograficky, tj. uživatel jako schematizovaná postavička, terminátor jako obrazovka a klávesnice, paměť jako válec (tj. svazek disků) ap. Takových piktogramů bývá ku dvacetí.

Dalším smyslem tohoto modelu je lokalizace komponent vytvářeného systému. Rozlišuje se dálkové a lokální spojení, znázorňuje se lokalizace počítačů a přiřazení databank.

Konstrukční model

Konstrukční model popisuje strukturu vytvářeného systému. Používá se jako cílový model, tj. ostatní modely slouží ke kvalifikovanému vytvoření konstrukčního modelu, který se předává programátorům k realizaci. Používají se dvě grafické representace, obě se označují jako strukturní diagramy (structure chart, structure diagram).

Prvý z nich (structure chart) je graf (strom) hierarchie volání procedur. Na vyšších úrovních hierarchie to jsou moduly – tj. konstrukční komponenty vytvářeného systému, na nejnižších úrovních to jsou funkce (procedury realizující nejnižší – tj. již dále nerozkládané – funkce z funkčního modelu) nebo také operace s daty (s data story). V jakém pořadí a za jakých podmínek se jednotlivé komponenty provádějí je popsáno slovně.

Druhá reprezentace vychází z JSP (Jackson Structured Programming). Používá zavedené značky (hvězdička pro iteraci, kolečko pro alternativu), listy strukturogramu však nejsou jednotlivé operace, ale volání jiných podřízených modulů nebo realizace funkcí (rozumí se funkcí z funkčního modelu). Tato reprezentace tedy popisuje strukturu řídících modulů.

Poznámka: Funkční modul je ten, který realizuje požadované funkce systému, řídicí modul je ten, který určuje kdy a za jakých podmínek se funkční moduly provádějí.

Vazby mezi modely

Vytvoření na sobě nezávislých modelů nemá samo o sobě příliš velký smysl. Musíme mít na mysli, že jednotlivé modely jsou jen pomůcky pro tvorbu **modelu vytvářeného systému.** Tímto modelem je souhrn všech vytvořených dílčích modelů, které jsou navzájem sladěny.

Důležitou roli zde hraje centrální databáze prostředku CASE, nazývá se „encyklopedie" nebo „repository". V ní se uchovávají všechny informace o vytvářeném systému, tj. jsou tam uloženy všechny modely. V repository jsou uloženy všechny grafické reprezentace jako jednotlivé výkresy, ale také jednotlivé prvky. Operacemi nad repository můžeme získat seznam nekonzistencí mezi jednotlivými modely.

Tvorba jednotlivých modelů je nutná etapa tvorby systému, je to ale jen sbírání informací. Je to pouze faktografická hra. Skutečná analytická a návrhářská práce je ve sladění všech těchto „pohledů" tak, aby tvořily pohled na **jediný** systém, a to ten vytvářený. Všechny modely musí být konzistentní, tj. nesmějí si odporovat. Teprve když jsou všechny modely konzistentní, lze říci, že to jsou pohledy na jeden a tentýž systém.

Důležité místo mezi modely zaujímají funkční a datový model. Jejich sladění se provádí tím, že se ztotožní „data story" z funkčního modelu s „entitami" z datového modelu. Pak funkční model pracuje pouze s úplnými normalizovanými a integrovanými daty.

Další příklady konzistencí:

Všechny funkce v organizačním modelu musí odpovídat funkčnímu modelu. Jinými slovy: Všechny vyžadované funkce musí být realizovány, nemá smysl realizovat funkce, které nikdo nepotřebuje.

Všechny údaje z komunikačního modelu se musí vyskytovat v datovém modelu. Jinými slovy: Systém může komunikovat s uživatelem jen prostřednictvím těch údajů, které má „uvnitř", tj. těmi, které zpracovává.

Konzistence se netýká pouze vztahů mezi modely, ale i uvnitř jednotlivých modelů mezi jednotlivými úrovněmi rozpracování. Tak například jsme se u funkčního modelu zmiňovali o rozkladu funkcí v podfunkce. Funkční model se vlastně skládá z podmodelů s různou úrovní podrobností. Také zde je nutno prověřit konzistenci.

Požadavek konzistence však není ultimativní (nebo by neměl být ultimativní). Nekonzistence jsou dvou druhů, nechtěné a chtěné.

Nechtěná nekonzistence je projevem toho, že model ještě není dokonalý. Mohou však existovat i nekonzistence, které jsme do modelu zavedli úmyslně, abychom si zjednodušili některé situace. Například konzistence ve funkčním modelu vyžaduje, aby na vyšších úrovních byly uvedeny všechny datové toky, které se vyskytují na podrobnějších úrovních. Někdy je však vhodné, aby na vyšších úrovních nebyly zobrazovány úplně všechny. Důvodem je větší přehlednost těchto vyšších úrovní. Musí však být jasné, které datové toky nezobrazujeme a proč.

Výše uvedený, více méně povrchní popis nekonzistencí musí čtenáři připadat triviální. Copak to je něco těžkého, nezapomenout na něco? Nebo je úplně samozřejmé, že data použitá na obrazovce (v komunikačním modelu) musí být uvedena i v popisu databáze (v datovém modelu). Problém není v jednotlivostech, ale v jejich množství. Jsou desítky různých druhů vazeb, které má smysl kontrolovat. Jestliže se počet prvků vyskytuje v desítkách, tak vazeb mezi nimi jsou stovky, ne-li tisíce. Prostředek CASE na nic nezapomene.

Poučným je následující příběh ze školení. Jeden můj kolega šel na školení Excelerátoru spíše ze zvědavosti. Viděl již hodně mesiášů, kteří slibovali spasit programování. Ostatně je docela zajímavé podívat se na úspěšný programový systém, a kreslit si obrázky datových toků je zábavná činnost. Tak si kreslil obrázky jednoduchého příkladu pro školení – ve třech úrovních podrobnosti. Školení šlo dále, ke zjišťování nekonzistencí, a kurzanti na své výtvory použili kontrolní funkce. Zmiňovaný kolega je dosti sebevědomý a rozhodně neočekával, že by mu mohl někdo v programování radit. Excelerátor mu v tomto jednoduchém příkladě našel 17 nekonzistencí – a měl pravdu. Vztah mého kolegy k prostředkům CASE se výrazně změnil. Uvědomil si totiž, kolik práce by ho stálo, kdyby na tyto nekonzistence přišel až při programování. Změnit obrázek je daleko jednodušší, navíc všechny změny prvků a vztahů se automaticky promítnou do ostatních modelů. Kdežto měnit již napsaný programový kód je pracné. Každá změna příkazů v jedné části implikuje změny v jiných částech – to je navíc zdroj chyb vnášených do systému.

Každou nekonzistenci v etapě návrhu zaplatíme buď reklamací uživatele nebo špatnou funkcí systému, která zase vede k nespokojenosti uživatele.

Životní cyklus systému

Známý pojem životního cyklu a jeho etapy lze zhruba popsat takto:

Analýza

Vstupem je nějaký cíl uživatele, který má nový programový systém splnit. Výstupem je zadání nového programového systému, nebo změny v existujícím, nebo rozhodnutí, že tvorba nového systému by nebyla efektivní.

Návrh

Vychází ze zadání systému, což je výstupem předchozí etapy, a určuje strukturu systému, tj. alokuje požadované funkce do jednotlivých prvků (modulů) systému. Určuje komunikaci mezi těmito prvky (moduly).

Implementace

Realizuje celý systém (jednotlivé moduly a jejich vazby) způsobem zadaným v etapě návrhu. Zde je vlastní programování.

Zavedení

Uvedení nového systému do provozu u uživatele. Sem patří například také školení.

Provoz

Tato etapa ovlivňuje tvorbu programového systému dvěma způsoby. Jsou to inovace nižšího řádu (opravy chyb, úpravy systému) a inovace vyššího řádu, které vyústí ve tvorbu nového systému (viz následující etapu).

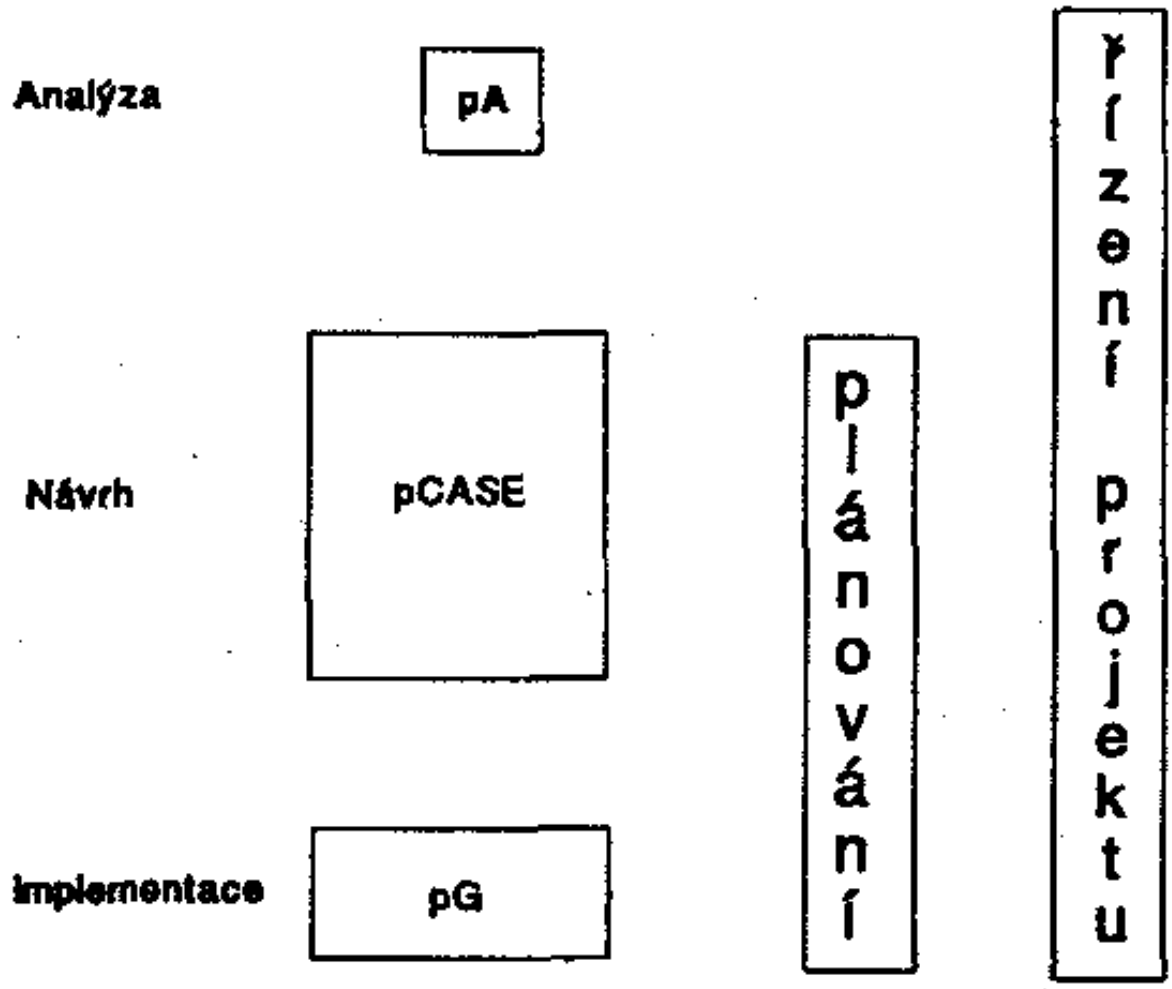
Hodnocení

Vyhodnocení zkušeností s provozovaným systémem je vlastně část analýzy nového systému. Jeden životní cyklus se uzavírá a pokračuje se tvorbou nového systému další generace.

V tomto článku se věnujeme prostředkům CASE v užším smyslu, tj. určeným pro první dvě etapy, částečně i pro třetí. Prostředky CASE se nekryjí přesně s uvedenými etapami. Schématicky to ukazuje obrázek na další straně.

Řízení projektu systému

Plánuje a sleduje celý životní cyklus programového systému. Softwarové firmy mají velmi propracované řízení projektu, kde je životní cyklus programu rozepsán do desítek činností (jednou z těchto činností je naprogramování tohoto nového systému). Hlavní myšlenka řízení je tato:



Je lacinější administrativně sledovat všechny projekty, než dopustit, aby se jeden produkt ukázal nefunkčním.

Tvorba projektu sestává z desítek na sebe navazujících činností, zčásti prováděných paralelně. Patří sem například nejen tvorba nových programů, ale i výběr hardwarových prostředků a rozhodnutí o nákupu software. Je třeba si uvědomit, že paralelně s tvorbou nového systému je třeba vyvinout testy na prověření jeho funkcí. Dále je potřeba hned trojí dokumentace, kromě autorské (která vzniká při tvorbě systému) je nutno vytvořit uživatelské příručky a školící pomůcky.

Plánování tvorby systému

Pod plánováním se nerozumí „české plánování" , tj. splnil jsi úkol, dostaneš prémie, nesplnil jsi úkol, stejně napíšeme, žes ho splnil a dostaneš prémie. Při spolupráci více lidí se plánuje návaznost jejich prací. Cílem je získat reálné informace o termínech a nákladech, ale hlavně mít možnost zjistit, kde dochází k potížím a ihned je minimalizovat.

Podpora analýzy (pA)

Podporu analýzy tvoří spíše metodické předpisy než programové prostředky. Z velké části zabíhá i do psychologie. Důležitou oblastí je umění interviewu.

Z programových prostředků tam patří koincidenční matice a organizační a funkční model. O některých prostředcích nelze jednoznačně prohlásit, zdali podporují analýzu nebo návrh – záleží na přístupu. Například mohu pomocí funkčního modelu popisovat stávající organizaci a teprve pak se rozhodnout, co bude automatizováno – a to pak popisovat funkčním modelem.

Podpora návrhu (pCASE)

Zde se plně uplatňují prostředky CASE v užším smyslu, např. Excelerátor, CASE 4.0, SDW a další. O těchto prostředcích lze říci, že podporují pozdní fáze analýzy a celý návrh.

Podpora implementace (pG)

Cílem návrhu programového systému je určit strukturu dat a strukturu programů, ne programovat. To je úlohou implementace. Přesto z určitých modelů mohou být získány části programů, proto bývají CASE vybaveny prostředky pro generaci.

Celkově lze říci: Na závěr etapy návrhu využijeme všeho, co lze automaticky generovat, a zbytek se doplní ručně. U jednoduchých úloh může být etapa implementace úplně automatizovaná. Obvykle se generují následující části:

Popis databáze

Z datového modelu lze přímo generovat popis relační databáze v jazyce SQL nebo v jazyce konkrétní databáze. Tuto komponentu má každý prostředek CASE, některé jsou schopny generovat i pro desítku různých databázových systémů. Ovšem popis dat může být vytvořen i pro Cobol nebo C.

Popis obrazovek

Často je možno z komunikačního modelu generovat popisy obrazovek – buď pro určitý databázový systém nebo jako deklarace v obecném programovacím jazyce. U prostředků CASE navazujících na jazyk typu 4GL může být tento prostředek velmi dokonalý, někdy je komunikační model přímo ztotožněn s jazykem 4GL.

Struktura systému „ve velkém"

Na některých prostředcích CASE je možno z konstrukčního modelu generovat síť vzájemně se volajících modulů, to vlastně znamená síť vzájemně se volajících podprogramů. Vyžaduje to ovšem mít k dispozici speciální jazyk pro programování „ve velkém". Úlohou implementace je pak pouze naprogramovat těla těchto procedur.

Reinženýring

To je speciální technika, velmi populární v poslední době, která svazuje etapy implementace a návrhu v opačném směru. Obvykle vytváří z již hotového programu (staršího)

konstrukční model. Je tím umožněno moderním způsobem upravovat již existující programy s nedostatečnou dokumentací.

Použití prostředku CASE

Proč se používají prostředky CASE je zřejmé, zlevňují výrobu velkých programových systémů, lze dokonce říci, že umožňují existenci velkých programových systémů.

Nejdříve vznikly metodiky (spojené se jmény Chen, DeMarco, Yourdon, Bachmmann). Metodiky vnášejí řád do procesu tvorby systému, samy o sobě jsou ale velmi pracné. Odhaduji, že ručně prováděný návrh je několikanásobně pracnější než vlastní vytvoření systému.

Prostředky CASE usnadní velmi podstatně návrh. Snižují pracnost ve dvou hlavních směrech: ve „štábní kultuře" pracovních materiálů a ve vyloučení přehlédnutí.

Ručně lze vytvořit krásné grafy a přehledy – ovšem ihned po svém vytvoření se zaplňují vsuvkami, změnami, poznámkami. Denodenně něco překreslovat nebo přepisovat – to člověk nedokáže, počítač ale ano. Jakýkoliv zásah do grafu (třeba datových toků) se okamžitě promítne překreslením celého obrázku. A současně se doplní (opraví) seznam příslušných komponent. S použitím prostředku CASE máme pracovní materiály ihned a dobře nakreslené.

Jeden projekt obsahuje desítky druhů komponent, stovky exemplářů a tisíce vztahů mezi komponentami. Člověk není schopen vzít při každé změně v úvahu všechny důsledky – důvodem je příliš velké množství vztahů.

Ve vztahu metodik návrhu systémů a prostředků CASE lze vysledovat dva základní směry. Jeden z nich je přizpůsobit prostředek CASE konkrétní metodice. Prostředek CASE kontroluje, zdali uživatel dodržuje pravidla metodiky a jejich porušení považuje za nekonzistenci či je jinak znemožňuje.

Druhý směr spíše poskytuje prostředky používané v jednotlivých metodikách a způsob jejich použití, pořadí použití a které nekonzistence se kontrolují, to vše nechává na uživateli.

Oba směry mají své chyby a přednosti, rozhodnutí mezi nimi závisí na stylu práce lidí, kteří systémy navrhují.

Nejpoužívanější způsob práce je tato základní posloupnost modelů:

- funkční model,
- datový model,
- funkční model,
- konstrukční model.

Prvým krokem je tvorba čistě funkčního modelu, tj. modelu funkcí a datových toků, bez „skladů" (data store). V případě striktního dodržování metodiky je přípustná pouze metodou shora–dolů.

Pak následuje podrobný popis datových toků. Tím se získá datová základna, ze které se vytváří datový model. Normalizované entity datového modelu se zařadí jako sklady do funkčního modelu.

Tímto druhým funkčním modelem definitivně skončila analýza úlohy a pokračuje se již konstrukčním návrhem vytvářeného systému, tj. tvorbou konstrukčního modelu. Primitivní funkce, tj. ty, které nejsou již dále rozloženy na podfunkce, jsou považovány za základní programované jednotky. Jsou seskupovány do skupin – modulů. Pro každou takovou skupinu je navržen řídicí modul, který řídí pořadí provádění funkcí. Tyto moduly jsou zase seskupovány do skupin – modulů vyšší úrovně.

Takto vytvořený konstrukční model je předán programátorům, kteří realizují řídicí i funkční moduly. Ostatní modely hrají pomocnou úlohu, jejich pomocí se kontroluje správnost jednotlivých kroků výše uvedeného postupu.

Tento postup je vhodný pro úlohy s převahou funkční složitosti, tj. pro takové úlohy, kde vztahy mezi daty nejsou příliš složité a těžiště práce je v efektivním zajištění akcí systému.

Pro systémy s velkou datovou složitostí je vhodnější následující postup:

- komunikační model,
- datový model,
- funkční model,
- konstrukční model.

Pomocí předchozí analýzy (koincidenčních matic) a komunikačního modelu se „posbírají" všechna data připadající v úvahu. Hlavní práce je ve tvorbě datového modelu, ale ne v celkem mechanické normalizaci, ale v doplnění vztahových entit, které vyjadřují složitější vztahy mezi daty. K tomu je zapotřebí dobře rozumět problému, což je věc citu. Mechanické prostředky a návody zde příliš nepomohou.

K hotovému datovému modelu se vytvoří funkční model, ve kterém se samozřejmě popisují požadované funkce, ale, což je důležitější, také funkce zajišťující integritu dat. Další postup je podobný jako u předchozího postupu.

Je-li úloha celkem jednoduchá, lze použít kratší postup:

- komunikační model,
- datový model,
- řídicí model.

Tento postup je možno použít tehdy, když jsou funkce systému převážně svázány přímo s daty, tj. jestliže vyžadované funkce pouze zpřístupňují a aktualizují data. Pod řídicím modelem se většinou rozumí návaznost obrazovek. Pro takové úlohy jsou výborným realizačním prostředkem jazyky typu 4GL.

Jestliže použitý prostředek CASE je vybaven dobrým systémem generace a řešená úloha je vskutku jednoduchá, tvorba těchto tří modelů může stačit k vytvoření celého systému – celý systém se realizuje generací z těchto tří modelů. Při propagaci prostředků CASE se nejčastěji setkáte s úlohami tohoto typu.

Zkušenější programátoři (nebo snad starší) se neradi nechávají svazovat nějakými publikovanými metodikami. Je to logické, na základě svých zkušeností si vybudovali své vlastní metodiky. Přeučovat se je namáhavé. Má to své opodstatnění. Mnohdy se ukazuje efektivním začít od prostředka, ne předepsaným způsobem shora–dolů. Mnohdy reálný pohled na systém lépe vznikne složením požadavků jednotlivých referentů než z podkladů vrcholového řízení. To se týká hlavně funkčního a konstrukčního modelu. Na druhé straně, na velkých systémech musí pracovat více lidí a není možné, aby každý z nich používal vlastní metodiku. Důležitější než nákup nějakého drahého a geniálního CASE je to, aby pracovní skupina měla svou vlastní metodiku, všem členům důvěrně známou. Samozřejmě při vytváření takové metodiky je dobré vzít v úvahu i to, co vymysleli jiní lidé (možná i chytřejší).

Zavedení prostředku CASE v jednom podniku

Programátoři v tomto podniku byli většinou specializováni na tvorbu systémových prostředků, zkušenosti s řešením aplikačních úloh bylo málo, i když zkušenosti přímo s uživateli byly bohaté.

Během roku 1990 postupně převládalo vědomí, že výrobou

operačních systémů se mnoho lidí v naší republice neuživí. Slovně to uznával každý, ovšem mezi lidskou činností a verbálním projevem je podstatný rozdíl.

Dělaly se první aplikační úlohy a ukázalo se, že jsou sice zdánlivě jednodušší, ale velmi pracné. Na rozmezí roku 90 a 91 byly vytvořeny dvě skupiny. Jedna sbírala informace o prostředcích CASE, druhá se zajímala o metodiku řešení aplikačních úloh, hlavně o etapu analýzy. Seznámení s metodikou bylo převážně svépomocí. Členové skupiny četli literaturu a navzájem si o ní referovali. Nebyla to tedy návštěva přednášek, ale lidé se k té metodice propracovávali sami. Ne všichni získané znalosti později použili, ale podstatně to pomohlo změnit atmosféru.

Další etapou bylo řešení reálných úloh (analýzy) pouze s určitou znalostí metodiky, bez podpůrných prostředků. To přesvědčilo o vhodnosti metodiky (v podstatě Yourdonovy) a ukázalo nutnost podpůrných prostředků. Některé jednodušší jsme si vyrobili sami

(udržování koincidenčních matic, o kterých je zde přednesen referát). Přesvědčivě se ukázalo, že nejlacinější je koupit si již osvědčený prostředek.

Byl koupen Excelerátor, hlavním důvodem výběru byla cena. Na skutečné zhodnocení jeho přínosu je ještě brzy, ale předchra jeho pořízení ukazuje, že je velmi malá pravděpodobnost, že by se neosvědčil.

Zajímavý byl vztah pracovníku k CASE.

1. etapa – je to drahá hračka pro šéfova oblíbence.

2. etapa – cosi bych potřeboval, ale mám příliš mnoho práce, než abych měl čas na takové hlouposti jako je školení.

3. etapa, když té práce bylo skutečně mnoho – přesně to potřebujeme.

4. etapa, po týdenním školení – to je krásný prostředek, s tím bude práce jedna báseň.

Autor: RNDr. Pavel Drbal, CSc.
VŠE - katedra informačních technologií
Nám. W. Churchilla 4
130 00 Praha 3
tel. (02) 2125, kl. 437