PODPORA ICT VE VELKÝCH FIRMÁCH A OBJEKTOVĚ ORIENTOVANÝ PŘÍSTUP

Vojtěch Merunka

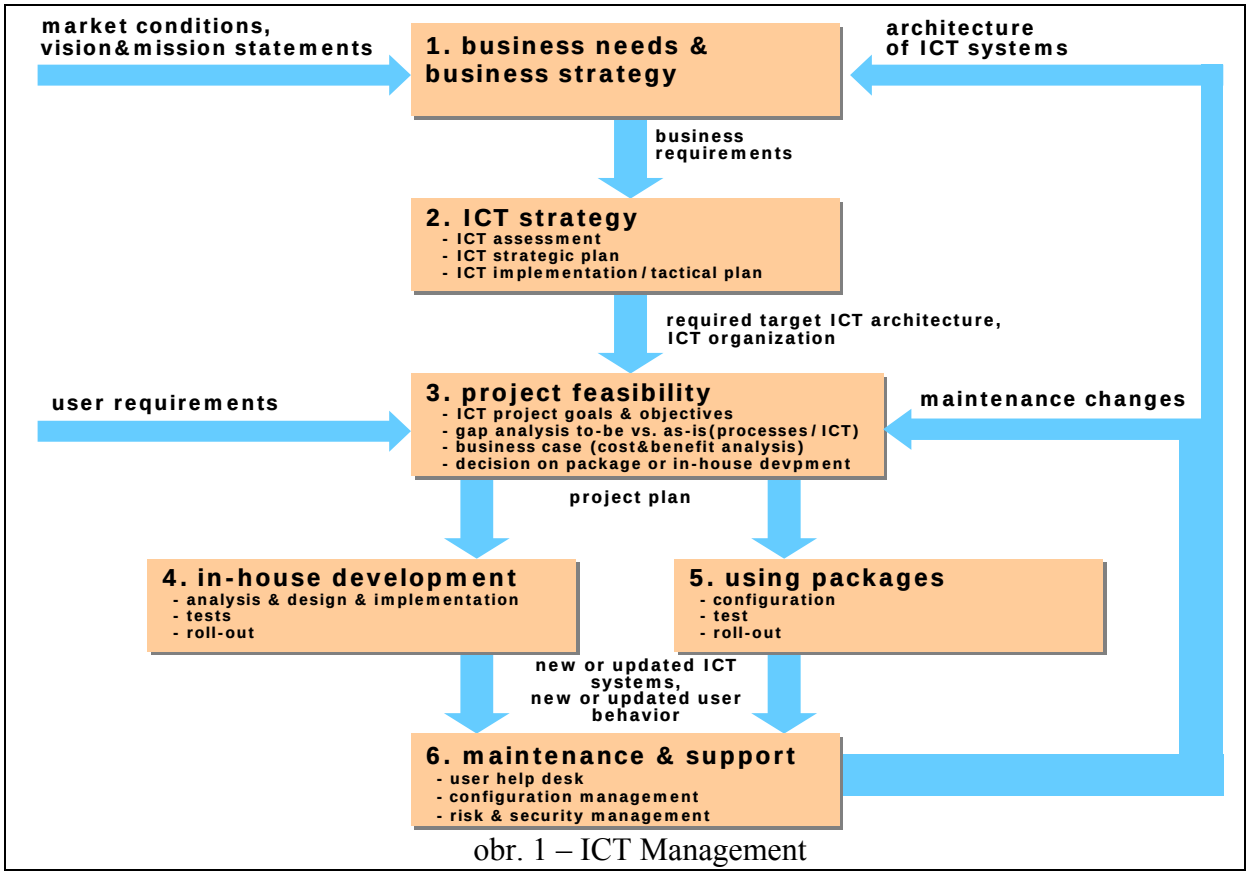
Katedra informačního inženýrství, PEF ČZU v Praze
merunka@pef.czu.cz, http://kii.pef.czu.cz/~merunka

Abstract
Tento článek se zabývá tvorbou softwaru s využitím objektově orientovaného přístupu ve velkých firmách z pohledu projektového manažera. V článku popisovaná metodika vychází z prací Scotta Amblera a praxe v poradenské a konzultační firmě Deloitte&Touche.

Klíčová slova: objektově orientovaný přístup, řídící a podpůrné procesy, organizační řád, pracovní týmy, ICT Management, Object-Oriented Process Pattern, RUP.

ICT Management

V minulosti se na firemní oddělení výpočetní techniky nahlíželo jinak než dnes – jen jako na jeden z řady podpůrných útvarů, které musely plnit příslušné úkoly, ale od kterých se neočekávala aktivní účast na rozhodování o řízení firmy.



obr. 1 – ICT Management

V dnešních firmách, ať se zabývají nejrůznějším předmětem podnikání, tvoří informační a komunikační systémy integrální součást samotných firem. Zároveň zde vlivem

technologického rozvoje dochází k postupné konvergenci informačních a komunikačních technologií. Proto se oblast řízení těchto technologií označuje „ICT Management“. Jedná se o soubor činností, které zároveň realizují a zároveň mají vliv na vlastní podnikovou strategii. Jeden z možných přístupů je uveden na obrázku č. 1, který je převzat z knihy Jima Halla [6].

Na tomto schématu jsou následující bloky činností:

1. **Formulace podnikové strategie firmy** – nejen na základě potřeb trhu, ale i na základě možností firemních informačních technologií.
2. **Formulace informační strategie firmy** – jako rozpracování formulované podnikové strategie na detail ICT.
3. **Posuzování projektů** – zde se rozhoduje o „spouštění“ jednotlivých ICT projektů jako postupná realizace cílů formulovaných v informační strategii s ohledem na potřeby uživatelů a znalosti z údržby stávajících systémů.
4. **Tvorba projektů vlastním vývojem** – jako jeden způsob realizace projektu.
5. **Použití hotového softwaru** – jako druhý způsob realizace projektu.
6. **Údržba a podpora** projektem vytvořeného či koupeného systému.

Pro řízení jednotlivých softwarových projektů, které je na schématu náplní bloků 3-4-6 nebo 3-5-6, máme zatím k dispozici dvě metodologie. První z nich je „Object-Oriented Software Process Pattern" od Scotta Amblera a druhou je RUP-„Rational Unified Process". Ve světě i u nás je více rozšířen RUP, ale Amblerův přístup má podle mého názoru následující významné přednosti:

1. … je jednodušší a je důsledně procesně orientovaný ve srovnání s RUPem, na který lze nahlížet jen jako na obrovskou sadu jednotlivých nástrojů a technik, ze kterých si projektový manažer musí umět správně vybrat.
2. … zabývá se i fází údržby a provozu již vyrobeného systému, čímž dobře do procesu začleňuje správu softwarové konfigurace, podporu uživatelům a vytváří předpoklady pro zpětnou vazbu na zahájení nových projektů. RUP se postimplementačními fázemi nezabývá.
3. … lépe podporuje tvorbu v čistých objektových jazycích a databázích (důraz na znovupoužitelnost, refactoring, komponentový přístup, …) a lépe do celkového procesu začleňuje využívání metrik. RUP je šitý na míru konzervativnější technologii hybridních programovacích jazyků (např. C++ nebo Java) a relačních databází.

Mýty a realita objektového přístupu u velkých projektů

V dnešní době není třeba obhajovat přednosti objektového přístupu, mezi které v první řadě patří znovupoužitelnost, komponentový přístup a celkově snadnější programování. Objektově orientované programování (OOP) již přestává být doménou nadšenců a již se stalo hlavním stylem tvorby softwaru. Určitě bychom dnes dokázali pojmenovat nemálo oblastí, kde bez využití OOP by nebylo možné software v rozumném čase a s rozumnými náklady ani sestavit.

Jenomže právě proto, že se s OOP dnes setkáváme u doopravdy velkých projektů, tak dochází k problémům. Tyto potíže vycházejí především ze slepé víry nadšenců (většinou čerstvých absolventů našich vysokých škol) ve všemocnost nových verzí objektových programovacích jazyků a v samospasitelnost UML. V žádném případě nezpochybňuji příznivé vlastnosti OOP, kterým se zabývám již přes 10 let a sám jsem nadšeným propagátorem Smalltalku a objektových databází. Jde ale o to, že u velkých projektů nelze pasivně očekávat, že se výhody objektového modelování a programování projeví „samy od sebe“. Taková

zjednodušení vedou k velkým rozčarováním. Dokonce se v praxi stává, že krach přehnaných očekávání a následný neúspěch projektu vede u firem k odsouzení OOP jako pro ně nevhodného přístupu. Velké projekty je totiž potřeba kvalifikovaně plánovat a řídit tak, aby se přednosti OOP doopravdy projevily a ne aby se staly ohrožením projektu. Problémy, o kterých hovořím, lze shrnout jako:

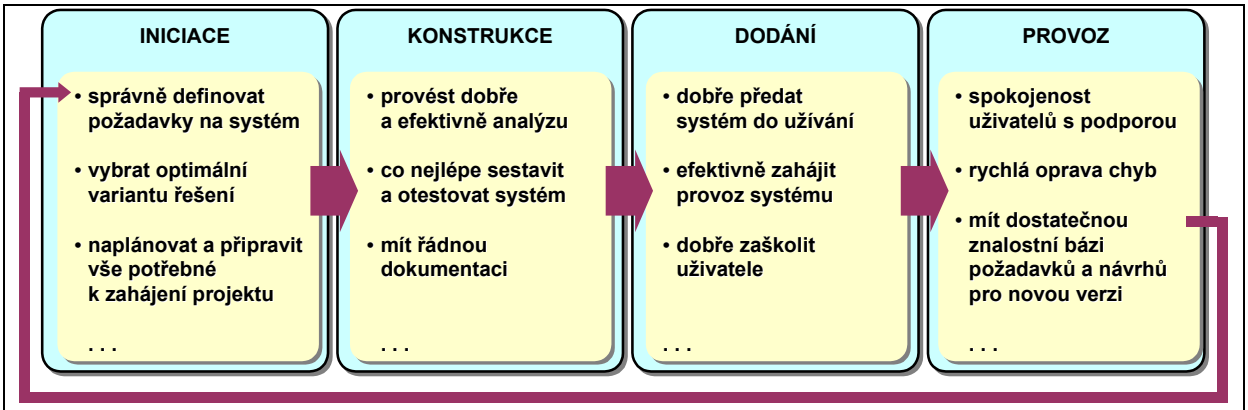
- **Mýtus iterativního přístupu**. Představa, že sekvenční alias vodopádový model tvorby softwaru je překonaná záležitost, je jednoduše mylná. Sám jsem pod vlivem populární literatury o OOP také dříve podlehl tomuto mámení. Pro menší projekty nebo samostatné práci nad komponentami je iterativní (či až dokonce spirální) přístup ideální, ale u velkých projektů není možný jiný přístup k jejich plánování a řízení než právě sekvenční.
- **Mýtus lepšího a kvalitnějšího programování**. OOP samozřejmě je novým a lepším paradigmatem tvorby softwaru – jenomže dnes neprogramujeme aplikace na zelené louce, ale tvoříme komponenty, kde se potýkáme s problémy napojení na stávající rozhraní a stávající datové struktury. K tomu si přidejme požadavky uživatelů. Představa, že kvality OOP zázračným způsobem sníží množství nezbytných testů, je v tomto kontextu nesprávná. Bohužel u velkých projektů testujeme více než kdykoliv předtím.
- **Mýtus znovupoužitelnosti**. Znovupoužitelnost je nepochybně výborná věc. Populární literatura uvádí, že může dosahovat až 80%. To mohu sám potvrdit i z vlastní zkušenosti. Jenomže musíme mít CO znovupoužívat. Komponenty a data v předchozích systémech jsou pro analytiky a vývojáře spíše noční můrou než úlevou. A pokud si chceme „vyrábět“ vlastní součástky k příštímu znovupoužití, tak nám takto vynaložená práce a čas zatěžuje projekt, který právě probíhá.
- **Mýtus nejdůležitějšího programátora**. Na problematiku „stavění" softwaru není možné nahlížet jen pouze z perspektivy „zedníků" byť jakkoli kvalifikovaných. Klíč k úspěšným velkým projektům spočívá v umění podívat se na tvorbu softwaru také očima „architekta" či „stavbyvedoucího". Poměr analytických profesí (doménových specialistů, softwarových architektů a dalších) se blíží na úkor ryze programátorských až k hodnotě 1:1. Na jejich znalostech a především schopnostech komunikace velké projekty často závisejí více, než na mistrovství programátorů.

ASDM – Advanced Software Development Model

Toto řešení je modifikací Amblerova přístupu, které se používá ve firmě Deloitte&Touche Praha. ASDM je model životního cyklu projektu obsahující čtyři hlavní fáze. Pro každou fázi jsou identifikovány dva až čtyři dílčí procesy a k nim soubor charakteristických činností a souvisejících pracovních rolí a odpovědností.

Tento přístup nahlíží na tvorbu softwaru zvenčí - očima „stavbyvedoucího" - a ne očima „zedníka". To znamená, že zde popisujeme něco trochu jiného, než klasické metody analýzy a návrhu, které jsou v tomto chápání metodami „pro zedníky". (Při určité míře zjednodušení to znamená, že například metoda „extrémního programování" nebo OMT může být chápána jako konkrétní náplň jediné fáze konstrukce tohoto přístupu).

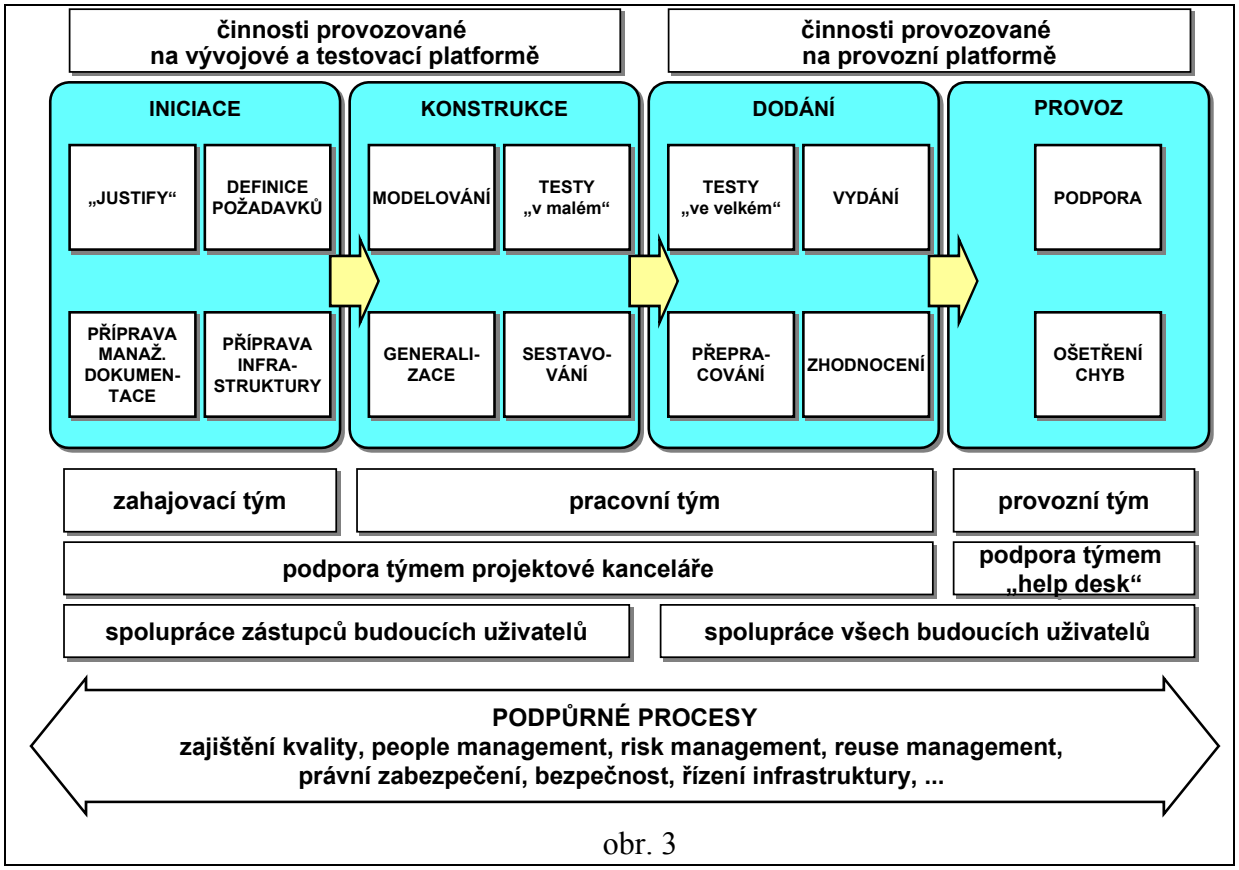
Celý model životního cyklu projektu je členěn na čtyři fáze pojmenované „iniciace", „konstrukce", „dodání" a „provoz". Z pohledu projektového managementu je každá fáze ASDM zodpovědná za plnění specifických výkonnostních požadavků (obr. 2).



obr. 2 – oblasti klíčových výkonnostních požadavků ASDM

Čtyři hlavní fáze ASDM by se měly provádět sériově po sobě. Protože na konci každé fáze dochází ke změnám týmů a platforem, tak je doporučováno, aby manažer projektu neopomněl svolat pracovní setkání, kde prezentuje dosavadní postup projektu, provede kontrolu dokumentace a dalších dosud vytvořených výstupů a seznámí pracovní skupinu s novými členy týmu.

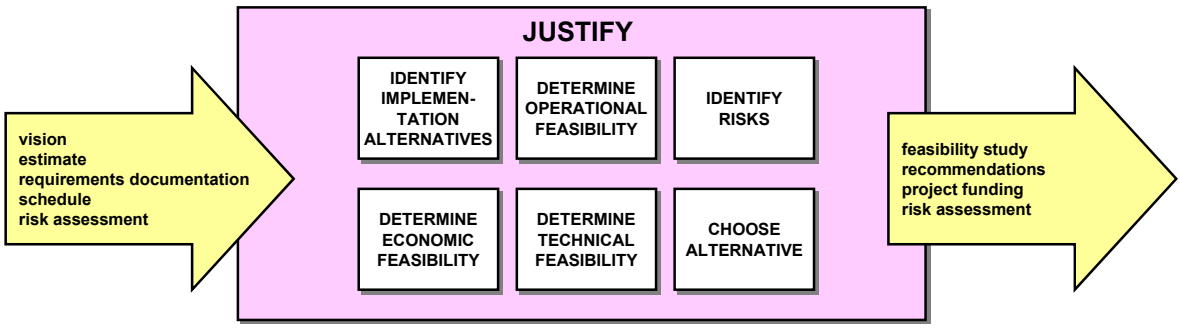
Každá z uvedených čtyř fází ASDM se skládá ze dvou až čtyř dílčích procesů. Dílčí procesy uvnitř fází je ale možné provádět opakovaně. (Chceme-li, tak na ASDM lze nahlížet jako na kombinaci vodopádového a spirálního přístupu.)



obr. 3

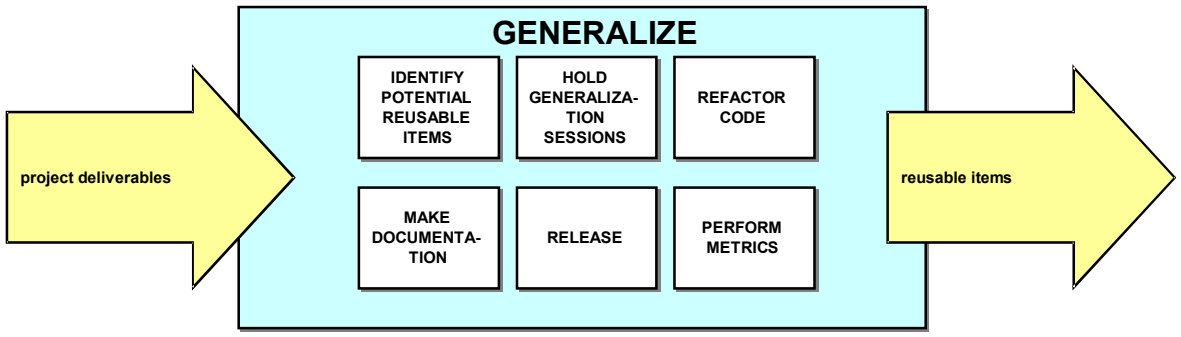
Každý z dílčích 14 procesů je možné v konkrétních podmínkách modelovat ve formě procesní mapy, kde lze vyznačit podrobný výčet potřebných aktivit a vymezení rolí účastníků procesu. Na obrázcích 4 a 5 jsou dva příklady podrobnějšího obsahu dílčích procesů:

The purpose is to determine whether or not an application should be built. It is a reality check to determine whether or not a project makes a sense.



obr. 4

This is the recognition that the short-term pressures of software development result in the temptation for developers to settle for specific, non-reusable solutions. In this process, application specific items are identified and then reworked to be reusable by other development teams.



obr. 5

Fáze a procesy jsou následující:

Proces **INICIACE** (initiation) slouží k zajištění všech přípravných prací, nezbytných k zahájení tvorby software. Jeho cílem je připravit vše potřebné k zahájení projektu. Skládá se z procesů:

1. **Definice požadavků** (define requirements). Prvotní sběr požadavků na nový systém, ke kterému dochází na zahajovací schůzce účastníků projektu.
2. **Příprava manažerské dokumentace** (prepare management documents). Příprava plánu čerpání zdrojů a časového plánu celého projektu. Ten zahrnuje požadavky na kapacity, technické a finanční zdroje včetně sestavení časového harmonogramu řešení.
3. **Příprava infrastruktury** (prepare infrastructure). Je to příprava podkladů, informačních zdrojů, případně i instalace dílčích systémů, které budou potřeba pro tvorbu systému.
4. **Proces zmocnění** či **narovnání** (justify). Dílčí proces, během kterého je třeba vypracovat rizika celého projektu, právního zabezpečení a počítačové bezpečnosti. Zároveň by zde mělo dojít k nalezení alternativ řešení a výběru optimální varianty. Během této fáze se také může na základě zjištěných okolností rozhodnout systém neimplementovat.

Proces **KONSTRUKCE** (construct) slouží k vytvoření požadovaného systému. Vlastní tvorba začíná teprve v této fázi. Vzhledem ke specifickým vlastnostem objektového software se předpokládá iterativní opakování těchto fází, protože zde může až v rámci prvotní implementace docházet ke zpřesnění a správnému pochopení zadání. Počet iterací se pohybuje okolo 2 až 3:

1. **Modelování** (model). Dílčí proces, ve kterém dochází k podrobné definici zadání a úplnému získání podkladových materiálů. Během tohoto procesu by měly být modelovány a vyzkoušeny (simulovány) cílové uživatelské procesy. Vše by mělo být dokumentováno.
2. **Sestavování** (program). Proces, kdy dojde k vytvoření (naprogramování) vlastního software. Tento podproces se odehrává na vývojové platformě. Kromě samotného software by měla být pro potřeby budoucích úprav a oprav vytvořena systémová dokumentace. Zároveň by měla být zahájena práce na uživatelské dokumentaci.
3. **Testy v malém** (tests in small). Proces, který slouží k prověření funkčnosti. Tento podproces se odehrává na testovací (nikoli provozní) platformě a účastní se ho speciálně určení členové vývojového týmu, tedy nikoliv skuteční uživatelé a na vývoji nezúčastnění zadavatelé.
4. **Generalizace** (generalize). Proces, kde dojde k optimalizaci vytvořeného software, jeho podrobnému dokumentování a identifikaci potenciálně znovupoužitelných a nahraditelných artefaktů. Tato „úklidová" fáze si neklade za cíl vylepšení funkčnosti vytvářeného systému, ale jeho rozčlenění po technické stránce. Bez tohoto pořádku, který je nutné do systému vnést, jsou pozdější úpravy, opravy, rozšiřování nebo opakované využití komponent ve tvorbě dalších programů komplikované.

Proces **DODÁNÍ** (deliver) slouží k uvedení nového systému do provozu na provozní platformě. Tvoří jej dílčí procesy, které mohou běžet i souběžně.

1. **Vydání** (release) je proces, který zajišťuje přechod na provozní platformu, řešení problémů s instalací na této platformě a přípravu provozní infrastruktury. Jeho součástí jsou i podpůrné a pomocné činnosti, jako například zveřejnění systému koncovým uživatelům (propagace) a organizace nezbytných zaškolení, vydání příruček atp.
2. **Testy ve velkém** (tests in large) jsou testy, prováděné zástupci cílových uživatelů.
3. **Přepracování** (rework) je opravný proces, který reaguje na výsledky testů. Je podobný konstrukci v malém a probíhá na provozní platformě. Nejde jenom o software, ale může obsahovat zásahy nebo změny do dokumentace atd.
4. **Zhodnocení projektu** (assessment) je závěrečný proces pracovního týmu a týmu projektové kanceláře, který slouží k jejich vnitřním potřebám. Je okamžik, kdy je možno poučit se z chyb, ocenit iniciativu účastníků, zhodnotit rizika, kvalitu, vyhodnotit metriky apod. Výsledky, alespoň rámcové, by měly být zveřejněny a vždy archivovány.

Proces **PROVOZU** pokrývá časový úsek používání systému v praxi. Jeho součástí jsou dva dílčí procesy:

1. **Podpora** (support), která reprezentuje množinu aktivit, prováděných v týmu „help desku". Jedná se o různé rady uživatelům, týkající se ovládání a instalace systému, péče o průběžné doškolování a zaškolování (např. nově příchozích uživatelů, kteří nebyli zaškoleni). Dále sem patří řešení problémů, spojených s poruchami hardware, případně sběr podnětů, vedoucích ke zlepšení chodu systému.
2. **Údržba** (maintenance), které představuje operativní odstraňování chyb v systému. Zde je důležité rozlišit alespoň podněty ke zlepšení systému, který je plně funkční, a

hlášení chyb, které nedovolují pokračovat v chodu systému nebo chyb, které ovlivňují chod systému nesprávným způsobem.

Manažer projektu by se měl také postarat v průběhu projektu o řádné zajištění kvality. Jako jednoduchý a účinný prostředek jsou doporučovány kontrolní seznamy (check-lists), kterými lze ověřovat, zda můžeme příslušnou fázi zahájit, zda vykonáváme co je právě potřeba a nebo zda můžeme fázi prohlásit za ukončenou (ukázka pro proces generalizace na obr. 6).

GENERALIZE
entrance conditions checklist

- ☑ project deliverable
- ☑ experienced reuse engineers are available
- ☑ organizational support for reuse exists
- ☑ team members have been given the appropriate training

to be performed checklist

- ☑ potential reusable items have been identified
- ☑ generalization sessions were held
- ☑ potentially reusable items were refactored
- ☑ reusable items were documented
- ☑ examples of how to reuse reusable items were documented
- ☑ reusable items were released into the repository and made accessible to all developers
- ☑ risk assessment document has been updated
- ☑ decisions (both made and forgone) were documented into group memory
- ☑ metrics have been collected

exit conditions checklist

- ☑ generalized items have been submitted to the reuse repository
- ☑ all developers have been made aware of new items

obr. 6

Jednotlivé týmy v procesech tvorby softwaru

Stejně jako fáze životního cyklu nelze zužovat pouze na triviální schéma analýza / návrh / implementace, tak i problematiku týmů nelze omezovat pouze na zadavatele, uživatele, řešitelský tým a jeho manažera. Pro správný chod popisovaných procesů potřebujeme následující:

Tým projektové kanceláře. Je to stálý tým pro všechny projekty, které se v softwarové firmě či oddělení provádějí. Jeho úkoly jsou tyto:

1. Organizuje a zahajuje tvorbu nového systému počínaje výběrem a jmenováním pracovního týmu.
2. Sbírá, archivuje a poskytuje nezbytné informace a dokumentaci vývojovým pracovníkům. K tomu je vhodné používat softwarový nástroj. Tato „knihovna znovupoužitelných znalostí" je anglicky označována jako „group memory".
3. Organizuje testování systému ve fázi dodání.
4. Zajišťuje školení.
5. Hodnotí projekty.
6. Organizuje ukončení činnosti pracovního týmu a předání jeho výsledků do provozu.
7. Řídí chod „help desku".

Tým „help desku". Je to stálý tým s těmito úkoly:

1. Je kontaktním místem pro hlášení chyb, organizuje a zajišťuje jejich nápravu, provádí vyrozumění o způsobu nápravy chyby podle zásady: kdo chybu ohlásil, ten je vždy informován o způsobu řešení.
2. Sbírá podněty ke zlepšení od uživatelů.

Zahajovací tým. Je to ad-hoc tým, sestavený projektovou kanceláří pro nastartování nového projektu. Jeho úkoly jsou:

1. Spolupracuje s týmem projektové kanceláře při jmenování pracovního týmu.
2. Připravuje manažerské podklady pro projekt.
3. Vyhodnocuje rizika, aspekty bezpečnosti, práva, varianty řešení, …
4. Sestavuje a zodpovídá za plán projektu, obsahujícího časové odhady, požadavky na zdroje a kapacity.
5. Koordinuje spolupráci se zástupci uživatelů.

Pracovní tým. Je to ad-hoc tým, sestavený projektovou kanceláří pro vypracování projektu. Mezi hlavní odpovědnosti a pravomoci patří:

1. Zodpovídá za projekt a jeho výsledek v osobě manažera projektu.
2. Produkuje všechny dohodnuté výstupy (software, dokumentace, příručky, rozčlenění na opakovatelně použitelné prvky, sběr hodnot příslušných metrik aj.).

V souvislosti s týmy je třeba se také lehce zmínit o problému optimální alokace lidských zdrojů. Pro čtyři fáze naší metodiky se doporučují optimální poměry alokace pracovní síly 1:3:2:1. Klasickou chybou nezkušených projektových manažerů je plýtvání pracovní silou při zahájení projektu. Autor článku dokonce zná případy z praxe některých českých firem, kdy původní mnohočlenný tým ze zahajovacího meetingu, kde vystupoval i charismatický šéf firmy a přizvaní „členové týmu“ s dlouhými akademickými tituly, do fáze konstrukce degradoval pouze na jednoho až tři přetížené studenty-programátory.

Závěr

Slepá víra ve výhody objektového přístupu může ve světě „velkého programování“ dokonce vést i ke zhoršení kvality tvorby softwaru. Bez aktivního manažerského zajištění nelze očekávat, že se projeví výhody pro OOP charakteristických vlastnosti jako je znovupoužitelnost, komponentový přístup a nové paradigma programování. V popisované metodice ASDM k tomu slouží především proces „justify“ z fáze „iniciace“, proces „generalizace“ z fáze „konstrukce“ a specifické požadavky na projektovou kancelář.

Literatura:

1. Merunka V., Polák J., Carda A.: Umění systémového návrhu, Grada 2003, ISBN 80-247-0424-2
2. Merunka V.: Řídící a podpůrné procesy v objektově orientované tvorbě softwaru, ve sborníku konference Objekty, listopad 2002 Praha, ISBN 80-213-0947-4
3. Drbal P.: Extrémní programování a metodický přístup, ve sborníku konference Tvorba softwaru 2002, ISBN 80-85988-74-7
4. Molhanec M.: UML – několik kritických poznámek, ve sborníku konference Tvorba softwaru, květen 2002, Ostrava, ISBN 80-85988-74-7
5. Molhanec M.: Objektové metodologie - jejich užití a výklad, v tomto sborníku konference Tvorba softwaru, květen 2003, Ostrava

6. Hall J.et al.: Accounting Information Systems 3rd edition, South-Western Publishing, 2003, ISBN 0538877960
7. Ambler, Scott W.: Process Patterns - Buiding Large-Scale Systems Using OO Technology, Cambridge University Press - Managing Object Technology Series 1998, ISBN 0-521-64568-9
8. Ambler, Scott W.: More Process Patterns - Delivering Large-Scale Systems Using OO Technology, Cambridge University Press - Managing Object Technology Series 1999, ISBN 0-521-65262-6
9. Taylor, David A.: Business Engineering with Object Technology, John Wiley 1995 ISBN: 0471045217