

# UML efektivně a prakticky

**Jiří Svačina**

Unicorn a. s.  
V Kapslovně 2767/2  
130 00 Praha 3  
e\_mail: jiri.svacina@unicorn.cz

**Klíčová slova:** architektura IS, objektově-orientované modelování, RUP, UML, vývoj IS

***Abstrakt:** Tento příspěvek se zabývá úlohou UML a vizuálního modelování v oblasti tvorby a implementace informačních systémů. Modelování je obecně vnímáno jako důležitá součást vývoje systémů, nicméně řada týmů teprve hledá správné využití modelovacích technik. Problematika nových platforem, objektového vývoje, internetových systémů apod. klade zvýšené nároky na kvalitu a užitečnost vytvářených modelů. Příspěvek nejprve ukazuje možné využití UML v procesu vývoje systému a dále shrnuje některé klíčové praktiky a poučení, které společnost Unicorn nabyla v rámci svého dlouhodobého působení v oblasti projekce, konstrukce a provozu informačních systémů.*

## 1. Úvod

Unified Modeling Language (UML) je dnes patrně nejrozšířenějším všeobecným standardem pro objektově-orientované modelování a obecně pro specifikaci a dokumentaci softwarových systémů. Samotný jazyk UML je formálně definován a spravován konsorciem OMG. UML je běžně vyučováno na vysokých školách, případně prostřednictvím komerčních ICT firem.

Určitou výzvou a problémem pro vývojové týmy nicméně stále zůstává schopnost skutečně efektivního a užitečného využití UML v praxi. Přece jen jazyk definující množství diagramů, symbolů, vazeb a rozšiřujících mechanismů není tak jednoduché použít jako například "staré dobré" vývojové diagramy.

UML samo podává jen málo informací o tom, jak jej konkrétně prakticky využít při tvorbě software. Vývojový tým by tedy kromě samotného UML měl aplikovat i určitou metodologii zohledňující kromě jiného také využití vizuálního modelování.

UML nabízí analytikům či vývojářům velkou variabilitu výrazových prostředků. Metodologie vývoje by měla stanovit, které z těchto prostředků jsou relevantní, a jak je třeba je využít. Bez ohledu na to, kterou z moderních metodologií tým využije, měl by se v oblasti využití UML soustředit na praktičnost a smysluplnost vizuálního modelování. To neznamená vytvořit co největší množství diagramů a modelů co nejpřesněji odpovídajících poslední specifikaci UML a co nejpřesněji popisujících každou maličkost, ale spíše využití modelování pro rychlou a přiměřeně přesnou komunikaci mezi členy týmu. Praktické modelování se tedy bude soustředit hlavně na řešení aktuálních problémů softwarového vývoje nebo systémové integrace, nikoli pouze na "tvorbu modelů".

Z pohledu využívání UML je možné dnešní vývojové týmy rozdělit na tři základní skupiny - týmy, které modelování nepraktikují nebo považují za zbytečné, týmy, které modelování glorifikují nebo s ním stráví převážnou část (převážně neúspěšných) projektů a pak týmy, které používají vizuální modelování jako jeden z praktických nástrojů pro tvorbu software a znají jeho výhody a omezení. Pro poslední skupinu pak není modelování přítěží, ale konkurenční výhodou.

## 2. Využití UML pro zadání informačního systému

Zadáním informačního systému rozumíme specifikaci požadavků, omezení, podmínek a pravidel definujících požadovaný cílový stav informačního systému. Zadání systému reprezentuje vizi nebo představu zadavatele systému (zákazníka), tedy specifikaci kvality a kvantity projektu, konfrontovanou s technickými a ekonomickými možnostmi řešitelského týmu. Zadání tedy definuje rozsah projektu.

Informaci popisujících požadavky na systém bude typicky velké množství. Jako vhodné se jeví zvolit určitou kategorizaci požadavků, např. pomocí metodiky FURPS+ (viz [1]). Metodika FURPS+ definuje jako kategorie požadavků funkčnost, použitelnost, spolehlivost, výkon, provozovatelnost a technologická kritéria, kterým je nutno vyhovět. Všechny tyto oblasti mohou za určitých okolností samozřejmě využívat výhod vizuálního modelování.

### 2.1 Modelování požadavků pomocí případů užití

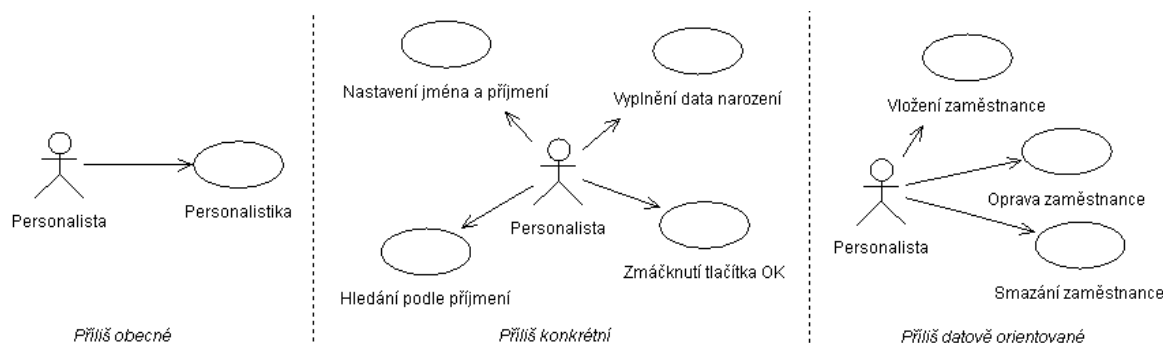
Naše zkušenosti ukazují, že pro problematiku informačních systémů spadá nejvíce objemu informací, které je třeba zjistit a zaznamenat, do kategorie funkčnosti. V této oblasti nabízí UML podporu především prostřednictvím tzv. diagramů případů užití (use case diagramů).

Např. podle [1] je případ užití "sekvence akcí, pomocí kterých systém poskytuje určitému aktérovi (externí entitě) službu nebo výsledek určité hodnoty". Laicky řečeno je případ užití konkrétní funkčnost systému, která z pohledu jeho uživatele "dává smysl", tedy jedná se např. o konkrétní pracovní postup, pomocí kterého lze dosáhnout určitého smysluplného cíle.

Případy užití jsou popsány ve formě textové specifikace definující cíl a způsob použití systému. Je-li těchto textových specifikací hodně (v řádu desítek či stovek), je výhodné vytvořit jeden či více diagramů případů užití, sloužících jako určitá "mapa" pro navigaci funkcností systému. Vývojový tým se může rozhodnout využít diagramů případů užití jako primárního prostředku řízení komunikace se zákazníky. Při modelování případů užití je potom třeba řešit hlavně následující problémy:

- Co je vlastně skutečně případem užití (problém granularity případů užití)
- Kdy přejít od kreslení diagramů k textové specifikaci případů užití (problém priority případů užití)
- Případy užití nepokrývají všechny požadavky (problém opomenutí dalších rozměrů systému)

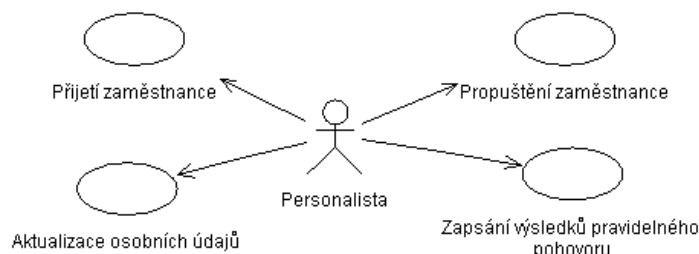
Problém granularity případů užití souvisí obvykle s nepochopením účelu modelu případů užití (snaha o hierarchickou dekompozici pomocí případů užití, snaha o modelování navigace uživatelského rozhraní, zjednodušení problému na vložení/opravu/smazání datové položky apod.), viz např. obr. 1.



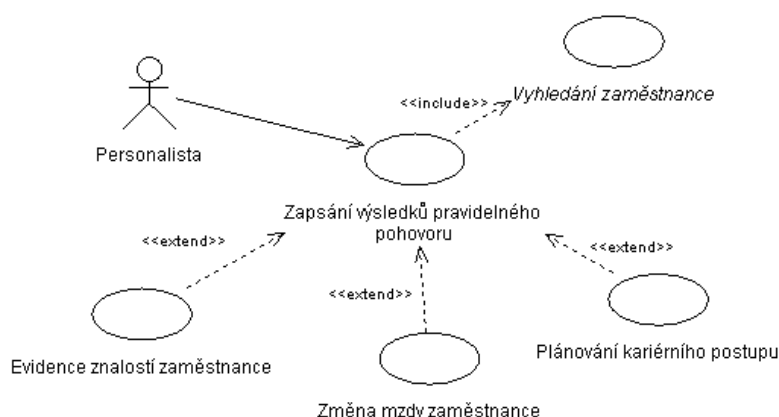
Obr 1. - Nevhodně identifikované případy užití (příklad)

Takto nevhodně pojaté případy užití vedou buď k nepochopení ze strany zákazníka (jsou moc technické) nebo naopak k nejednoznačnému zadání systému (přílišné zevšeobecnění).

"Správně" strukturovaný model případů užití bude mít podle našich zkušeností dvě úrovně podrobnosti. První úroveň podrobnosti (viz obr. 2) se soustředí na klíčové pracovní postupy, které má systém z pohledu aktérů zachycovat, druhá úroveň podrobnosti (určitá forma dekompozice) potom na specifitější nebo "nezajímavé" pracovní postupy, doplňující ty klíčové (viz obr. 3).



**Obr. 2 - Klíčové pracovní postupy v diagramu případů užití (příklad)**



**Obr. 3 - Zpodrobnění klíčového případu užití za použití vazeb mezi případy užití (příklad)**

Samotný diagram případů užití samozřejmě nedává žádné konkrétní informace o zadání systému, je to jen vizualizace "nadpisů" jednotlivých specifikací. Vývojový tým musí vyhodnotit prioritu jednotlivých případů užití z pohledu zákazníka a soustředit se na specifikaci (popis) důležitých funkcí co nejdříve. Další požadavky (URPS+) je potom třeba specifikovat a vyhodnotit zároveň s případy užití.

Řízení projektu iterativním způsobem pak umožní vytvářet funkční software již v situaci, kdy např. některé případy užití ještě nejsou plně specifikovány (neboť z pohledu řízení představují malé riziko).

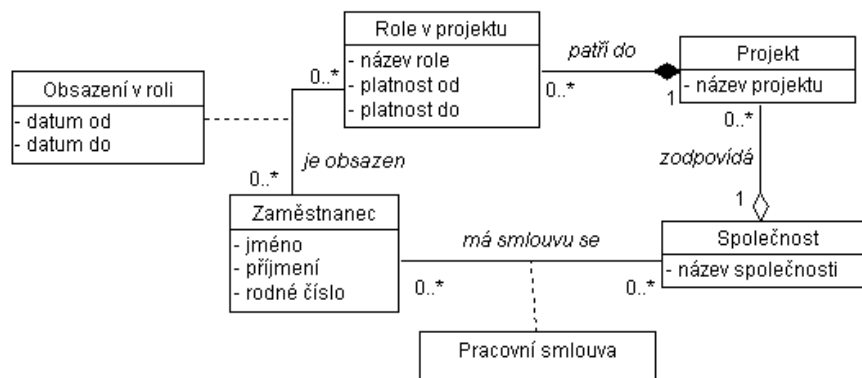
## 2.2 Modelování problémové domény

Model případů užití je zaměřen na specifikaci požadované funkčnosti informačního systému. Součástí procesu pochopení zadání systému je i specifikace pravidel, která systém musí dodržovat (business rules).

Zkušenost ukazuje, že pro zachycení významné množiny business rules je účelné studovat informace, které má systém zpracovávat, jejich strukturu, vztahy a omezení těchto vztahů.

Tyto skutečnosti se obtížně zachycují prostřednictvím případů užití, především proto, že je obvyklé, aby skupina případů užití pracovala nad společnou množinou informací - potom stojíme před problémem, kam vlastně pravidla zapsat.

Z pohledu vizuálního modelování se pro zachycení struktury informací, se kterými systém pracuje, jeví efektivní využití diagramů tříd.



**Obr. 4 - Diagram tříd, definující informace zpracovávané v systému (příklad)**

Takovéto diagramy je možno považovat za ekvivalent konceptuálních datových modelů, známých z jiných metodik analýzy a návrhu systému. Diagram tříd má ovšem bohatější výrazové prostředky (více typů vazeb, přesná specifikace kardinality vazeb přímo v diagramu, jazyk omezení OCL apod.).

### 2.3 Další techniky UML využitelné pro zachycení požadavků na systém

Případy užití je možné doplnit o diagramy aktivit ukazující tok událostí případu užití. Diagramy tříd ukazující řešenou problémovou doménu je možné dále doplnit o objektové diagramy ukazující instance tříd a vztahy mezi nimi nebo o stavové diagramy ukazující životní cyklus entit v systému.

Z praktického hlediska je třeba poznamenat, že UML dnes v oblasti zadání systému postrádá především

- ekvivalent data flow diagramů (DFD), využitelných pro dekompozici procesů a specifikaci datových toků a transformací pro části systému bez výrazné uživatelské interakce,
- prostředky pro modelování vzhledu uživatelských rozhraní a navigace uživatelským rozhraním.

Tyto nedostatky nutí vývojový tým v případě potřeby využít další modelovací techniky.

## 3. Využití UML pro analýzu a návrh informačního systému

Techniky uvedené v předchozí kapitole nemají v zásadě nic moc společného s objektově-orientovanou analýzou a návrhem. Soustředí se na specifikaci požadavků na softwarový systém, tedy na zodpovězení otázky "CO" máme vlastně vyrobit.

Další oblastí zájmu vývojového týmu bude softwarová architektura, tedy specifikace toho, "JAK" je nebo bude informační systém postaven z technického hlediska. Pro další účely budeme předpokládat, že vývojový tým použije alespoň pro část řešení objektově-orientované technologie - v opačném případě mohou být úvahy o UML nevhodné.

### 3.1 Softwarová architektura

Softwarová architektura definuje vnitřní organizaci systému z různých pohledů, např. (viz [1]). Všechny tyto pohledy by měly vyhovovat požadavkům kladeným na systém. Všechny tyto pohledy je také možné dokumentovat prostřednictvím UML.

*Pozn.: Každý softwarový systém má nějakou architekturu. UML a vizuální modelování je možné obecně použít pro definici (vytvoření) architektury nebo její dokumentaci, případně pro obojí.*

To, že vizuální modelování a UML je možné použít, ještě neznamená, že to je vždy vhodné. Vývojový tým se musí rozhodnout, které aspekty architektury má smysl "analyzovat" a/nebo "navrhovat" pomocí UML, a které aspekty je možné pokrýt jiným, efektivnějším způsobem, např. zdrojovým kódem.

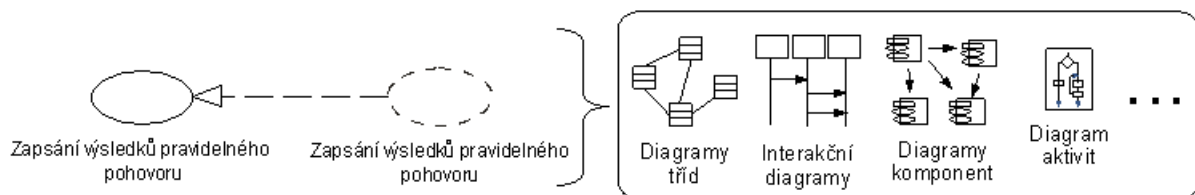
Co je třeba analyzovat a navrhovat pomocí modelování a kolik tohoto modelování je třeba udělat dříve, než začneme implementovat (programovat) je dáno riziky, kterým projekt musí čelit.

Jestliže zadání systému je definováno v kategoriích FURPS+, potom vývojový tým musí vzít všechny požadavky ze všech kategorií a zvážit, jak velké potenciální riziko daný požadavek pro tým představuje. Architekturu potom budujeme tak, abychom co nejdříve vyřešili klíčová rizika.

### 3.2 Realizace případů užití

Iterativní řízení projektu vyžaduje, aby byl informační systém vytvářen postupně a postupně také validován a verifikován (schválen a testován). Jestliže případ užití definuje funkčnost "užitečnou" z pohledu uživatele, potom je logické implementovat systém postupně s ohledem na případy užití.

Pro každý případ užití potom UML umožňuje definovat tzv. realizaci případu užití neboli kolaboraci.



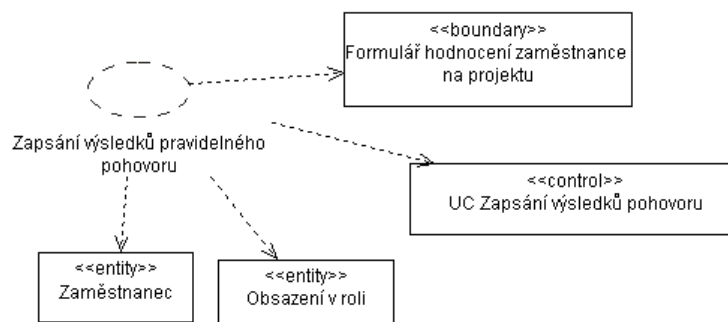
Obr. 5 - Realizace případu užití

Realizace případu užití představuje analýzu nebo návrh implementace dané funkčnosti. Definuje použité třídy či komponenty, jejich zodpovědnost a vzájemnou spolupráci vedoucí k implementaci požadavku. Samotná realizace potom v modelu slouží jako místo, ve kterém jsou pohromadě různé diagramy ukazující způsob realizace daného případu užití.

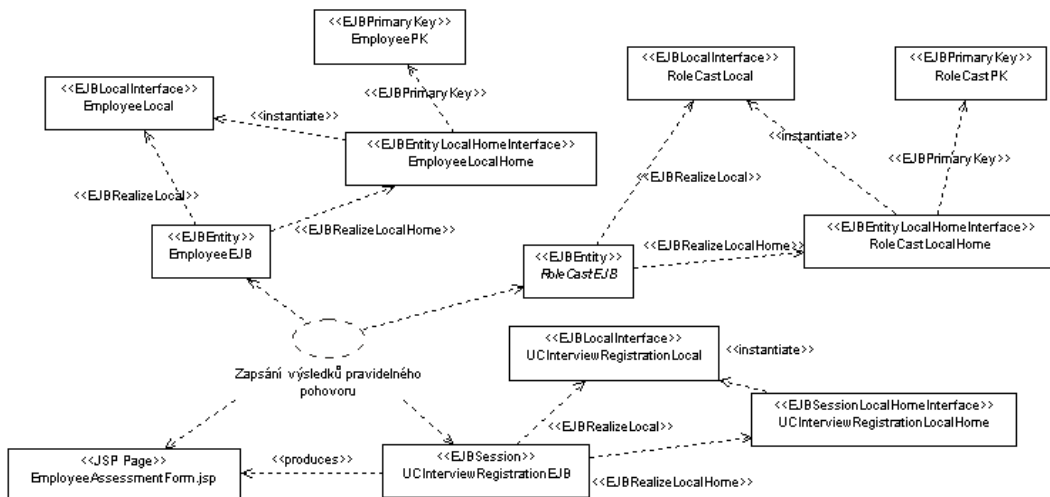
### 3.3 Analytický a návrhový model

Cílem objektově-orientované analýzy a návrhu je vytvořit modulární systém složený z prvků s vysokou soudržností (high cohesion, prvek zodpovídá za dobře definovanou a srozumitelnou sadu činností) a slabou provázaností (low coupling, prvky nejsou provázány každý s každým, ale komunikují podle dohodnutých pravidel). Například pro interaktivní případ užití zvolíme obvykle určitou (v rámci podobných případů užití v daném systému pokud možno stejnou) variaci návrhového vzoru model-view-controller (MVC).

Objektový model potom má např. dvě úrovně abstrakce - mluvíme o analytickém a návrhovém modelu.

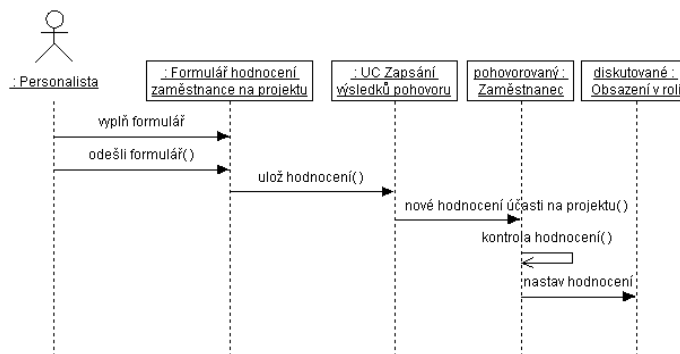


Obr. 6 - Analytický model pracuje s abstrakcemi přímo na úrovni MVC (příklad)



**Obr. 7 - Návrhový model pracuje s abstrakcemi přímo mapovatelnými na zdrojový kód (příklad)**

Zde uvedené diagramy tříd ještě nepředstavují objektovou analýzu nebo návrh, protože statický diagram tříd neukazuje spolupráci objektů. Za tímto účelem je třeba používat interakční diagramy, jako je sekvenční nebo kolaborační diagram (viz obr. 8).



**Obr 8 - Interakční diagram ukazuje spolupráci tříd (příklad)**

Programátor nebo softwarový architekt využívající UML pro modelování realizací případů užití musí samozřejmě dostatečně dobře znát jak model systému (aby mohl příslušné třídy/komponenty využívat opakovaně), tak cílové implementační prostředí (aby model měl hlavu a patu a byl implementovatelný).

### 3.4 Aplikační framework a design patterns

Postup popsany v předchozí podkapitole předpokládá, že programátor nebo softwarový architekt "vědí", jak danou funkčnost realizovat a v zásadě řeší především konkrétní detaily. Za tímto účelem musí vývojový tým sdílet společnou technickou vizi daného systému. Tato vize je realizována tzv. aplikačním frameworkem. Framework definuje softwarovou architekturu požadovaného systému.

V informačních systémech lze z pohledu architektury nalézt určité typické opakující se situace. Aplikační framework pak využije tzv. architektonické design patterns, ukazující typické řešení takovýchto situací. Příkladem architektonických design patterns mohou být systémové problémy, jako je autorizace, autentifikace, auditing, pooling a caching objektů, řízení transakcí apod., nebo typické situace z pohledu uživatele, např. práce s formulářem, průvodce, obsluha jednoduchého číselníku nebo práce se sestavami. Takovéto design patterns je vhodné vyhledat, popsat a opakovaně používat.

Klíčovou otázkou z pohledu modelování potom je, zda je třeba tyto konkrétní situace odpovídající nějakému design pattern znovu modelovat nebo "jen" odkázat na existující známé řešení. V takovéto situaci se jedná především o ekonomické rozhodnutí související s rozporem mezi náklady na projekt a touhou po co nejpodrobnější a nejpřesnější dokumentaci softwarového řešení.

Nelze obecně říci, kdy pro analytiku a návrháře má smysl modelovat formou kopírování již existujících myšlenek a kdy je výhodnější pouze odkázat na obecné řešení - záleží na celkové situaci projektu.

## 4. Praktické zkušenosti s modelováním

Naše dlouhodobá praxe ve tvorbě informačních systémů nás vedla k formulování určitých klíčových praktik souvisejících s modelováním systémů. Tyto praktiky jsou orientovány na modelování, které je efektivní a přináší projektům konkurenční výhodu a vysokou přidanou hodnotu. Určitým potvrzením správnosti úvah, ke kterým jsme dospěli, bylo vydání knihy o tzv. agilním modelování (viz [3]), jejíž závěry podivuhodně dobře korespondují s našimi zkušenostmi.

Následující myšlenky jsou zjednodušeným souhrnem postupů a pravidel, která s ohledem na modelování informačních systémů považujeme za klíčové.

### 4.1 Primárním cílem je tvorba software

Primárním cílem projektů informačních systémů je vytvořit a dodat požadovaný software. Úspěch projektu je charakterizován splněním čtyř klíčových proměnných projektu - kvality, kvantity, termínu a rozpočtu. Kvalitu pak chápeme jako "schopnost produktu být efektivně použit", tedy jako vlastnost související s návratností investice zákazníka do softwarového projektu.

Model jako takový není z pohledu zákazníka softwarem - není to tedy primární cílový produkt. Existuje celá řada projektů, jejichž výstupem jsou hlavně modely softwarových systémů (studie proveditelnosti, dokumentace existujících systémů apod.), nicméně u těchto projektů je samotná přidaná hodnota velmi sporná, pokud nenásleduje úspěšný projekt vedoucí k vytvoření nebo rozvoji informačního systému.

Z tohoto pohledu je třeba modely chápat jako artefakty víceméně dočasné, sloužící jako jeden z podkladů k vytvoření software. Jejich význam je pak především v komunikaci mezi lidmi.

### 4.2 Dalším cílem je zajištění pokračování života softwarového systému

Myšlenka předchozí podkapitoly naráží často na námitky typu "a co technická dokumentace?". Modely mohou představovat užitečný zdroj informací pro provoz, údržbu a rozvoj systému - obsahují znalosti a informace, které není bezpečné udržovat pouze formou know-how existujícího v myslích členů vývojového týmu.

Správnou reakcí na takovouto námitku je korektně určit, které z vytvářených modelů mohou pomoci v zajištění kontinuity života softwarového systému.

Jedná se opět o ekonomické rozhodnutí. Bylo by krásné mít vždy plně aktuální verzi všech modelů ukazující systém z celé řady pohledů. Praxe ale ukazuje, že takovýto stav je jen těžko dosažitelný. Aktualizace všech modelů je pro tým nákladná (zabere to hodně času) a je také špatně vymahatelná (je obtížné v případě změny motivovat členy týmu k zásahům do mnoha dokumentů). Vývojový tým se se zákazníkem musí domluvit, které modely je třeba udržovat a které je možno zneplatnit.

Například společnost Unicorn Services, zabývající se provozem a rozvojem informačních systémů, vyžaduje od dodavatele aktuální a dostatečně přesné modely dokumentující funkčnost systému (model případů užití), databázi (datové modely), technickou implementaci (implementační model, komponenty, jejich zodpovědnosti, provázanost, moduly systému apod.) a model softwarové architektury.

### 4.3 Úloha UML a CASE nástrojů v modelování

UML je komplexní a rozsáhlý jazyk pro modelování systémů. Nicméně UML samotné není obecně postačujícím prostředkem pro efektivní modelování - postrádá celou řadu technik, namátkou datové modelování, procesní modelování (DFD), modelování uživatelských rozhraní a navigace systémem atd. Analytik si těchto nedostatků musí být vědom a v případě potřeby využít jinou techniku.

Stejně tak bývá často pro modelování přeceňována úloha CASE nástrojů. Jestliže považujeme modelování za prostředek komunikace, pak mnohdy není efektivní používat pro modelování počítačový nástroj - lepší bude tužka a papír nebo tabule a digitální fotoaparát. Tým pak musí zvážit, zda je třeba taktické informace zachycené touto formou překreslit do formálního modelu nebo prostě zahodit.

### 4.4 Modelování v kontextu životního cyklu projektu

Iterativní projekt předpokládá, že nastanou změny. Tyto změny se vývojový tým musí naučit chápat jako nedílnou součást životního cyklu projektu. Jelikož model není software, jeví se jako nesprávná praktika snažit se nejprve technické řešení celého systému podrobně namodelovat a následně teprve implementovat. Klíčovým aspektem je zde nedostatek zpětné vazby, který modelování přináší - málokdo má schopnost se v rozsáhlém modelu skutečně dobře orientovat, pokud si nemůže "sáhnout" na skutečný software.

Stejně jako vše ostatní v iterativním projektu by i modelování mělo být orientováno na rizika projektu. Vytvoření kompletního podrobného technického modelu celého systému dříve, než začneme vyvíjet rizika spíše zvyšuje než snižuje - proto jej nepovažujeme za efektivní a užitečnou techniku.

Jinými slovy, pokud někdy vývojový tým definuje svou činnost pomocí formulací typu "v této fázi projektu se zabýváme modelováním toho a toho", pak takový projekt pravděpodobně nedopadne dobře.

## 5. Závěr

Problematika vizuálního modelování v kontextu tvorby a implementace informačních systémů zůstává jednou z klíčových technik vedoucích k úspěchu v našem oboru. Nicméně jako každá jiná technika i vizuální modelování má svá silná a slabá místa a zůstává úkolem softwarových týmů snažit se nalézt co nejvhodnější způsob jeho využití.

Věřím, že naši společnosti se to daří a že se z našich zkušeností mohou poučit i ostatní.

## Literatura

- [1] Rational Unified Process 2002.05, IBM; 2002, <http://www.rational.com/products/process.jsp>
- [2] OMG Unified Modeling Language Specification 1.5, OMG; 2003, <http://www.omg.org>
- [3] Agile Modeling, Scott Ambler; John Wiley & Sons, 2002, ISBN: 0-471-20282-7

## Abstract

This paper discusses the UML and visual modeling in the context of implementation and development of information systems. Visual modeling is considered as an important part of system development, yet there are many teams still looking for effective and correct utilization of these techniques. The issues like new technologies, object-orientation, Internet based systems etc. emphasizes the quality and usability of our models. The paper shows some techniques we have found useful for software development and then discusses some of the key practices and lessons learned by the Unicorn company during the many years we are in the business of projecting, constructing and maintaining the information systems.