METODIKA FEATURE-DRIVEN DEVELOPMENT NEOPOUŠTÍ MODELOVÁNÍ A PROCESY, A PŘESTO PŘINÁŠÍ VÝHODY AGILNÍHO VÝVOJE

ing. Alena Buchalcevová, Ph.D

Katedra informačních technologií VŠE Praha
nám. W.Churchilla 4, Praha 3 E-mail: buchalc@vse.cz

Abstrakt
V příspěvku je charakterizována jedna z agilních metodik, metodika Feature-Driven Development (FDD), která dle názoru autorky účelně spojuje modelování s principy agilního vývoje. Jsou popsány nejdůležitější principy a praktiky, na kterých je tato metodika založena, zejména definice užitných vlastností (features) a vývoj řízený těmito užitnými vlastnostmi. Důležitou úlohu v FDD hraje také složení vývojového týmu a odpovědnosti jednotlivých členů, monitorování procesu vývoje a kontrolní mechanismy.

1 ÚVOD

V současnosti můžeme sledovat dva hlavní proudy v metodických přístupech k budování informačních systémů, které jsou označovány jako rigorózní metodiky a agilní metodiky. Hlavním kritériem, které tyto dva proudy odlišuje, je kritérium označované jako Váha metodiky [Buchalcevová,2005]. Rigorózní metodiky vycházejí z přesvědčení, že procesy při budování IS/ICT lze popsat, plánovat, řídit a měřit. Podrobně a přesně definují procesy, činnosti a vytvářené produkty, a tak bývají velmi objemné. Rigorózní metodiky jsou zpravidla založeny na sériovém (vodopádovém) vývoji, ale některé jsou postaveny i na iterativním a inkrementálním vývoji. Jako příklad rigorózních iterativních metodik lze uvést metodiky OPEN, Rational Unified Process (RUP), Enterprise Unified Process (EUP), V rámci rigorózních metodik tvoří samostatnou kategorii metodiky pro hodnocení softwarových procesů (Software Process Assesment), z nichž nejznámější je Model zralosti (Capability Maturity Model) [Buchalcevová,2005].
Turbulentní změny ekonomického prostředí i informačních a komunikačních technologií, spolu s požadavky na rychlé zavedení informačního systému vyžadují změny přístupu k vývoji software, tedy změny metodik. To se projevilo na přelomu tisíciletí vznikem tzv. agilních metodik, jejichž primárním cílem je vytvoření fungujícího software, který přináší hodnotu zákazníkovi. Aby tohoto cíle dosáhly, maximálně „odlehčují“ proces vývoje software pokud jde o meziprodukty, modely, dokumentaci apod. Zaměřují se především na principy a praktiky, na lidi a přímou komunikaci mezi nimi [Buchalcevová,2002]. Mezi agilní metodiky patří také metodika Feature-Driven Development (FDD), jejímiž autory jsou Jeff De Luca a Peter Coad. Tato metodika patří na pomyslné stupnici váhy metodiky k těm spíše „těžším“ a formálnějším agilním metodikám zejména proto, že zachovává procesní řízení, i když ve velmi odlehčené podobě, a zdůrazňuje úlohu modelování při vývoji. To je také důvodem, proč je úspěšně používána i na větších projektech a projektech vývoje softwaru kritického pro poslání organizace.

2 CHARAKTERISTIKA METODIKY FDD

Metodika FDD je založena na iterativním vývoji, který je řízen užitnými vlastnostmi produktu (feature-driven). FDD je iterativní proces s krátkými iteracemi založený na modelování. Začíná vytvořením celkového modelu a pokračuje posloupností dvoutýdenních iterací, ve kterých se provádí návrh i realizace pro jednotlivé užitné vlastnosti. Užitná vlastnost (feature) je malý výsledek užitečný z pohledu zákazníka, je srozumitelný, měřitelný a realizovatelný

v rámci dvoutýdenní iterace. FDD bylo poprvé uvedeno v knize Java Modeling in Color with UML [Coad,1999]. FDD můžeme charakterizovat jako agilní, adaptivní proces vývoje softwaru, který:

- je vysoce iterativní,
- zdůrazňuje kvalitu v každém kroku,
- dodává časté, hmatatelné a fungující výsledky na každé úrovni projektu od vývojáře po vedoucího projektu,
- s minimální zátěží pro vývojáře poskytuje přesné a smysluplné informace o stavu projektu,
- je oblíbený u klientů, kteří vidí výsledky včas a mohou sami posoudit postup projektu, manažerů, kterým poskytuje přesné a smysluplné informace o stavu projektu, i vývojářů, kteří rádi pracují v malých týmech a v krátkých iteracích.

3 PRAKTIKY FDD

Podobně jako jiné úspěšné metodiky vývoje softwaru je FDD postaveno na klíčové skupině praktik, které úspěšně aplikovali v praxi Peter Coad a Jeff de Luca. Tyto praktiky nejsou nové, ale přínos FDD spočívá ve správné kombinaci těchto technik a praktik, které se navzájem doplňují a ovlivňují. Celkový výsledek je pak mnohem dokonalejší než prostý souhrn částí. Plný přínos FDD tedy dosáhneme jen při aplikaci celého procesu FDD. Praktiky, z nichž se skládá FDD jsou [Palmer,Felsing,2002]:

- doménové objektové modelování (Domain Object Modeling),
- vývoj podle užitných vlastností (Developing by Feature),
- vlastnictví tříd (Individual Class Ownership),
- týmy pro užitné vlastnosti (Feature Teams),
- inspekce (Inspections)
- pravidelné buildy (Regular Builds),
- řízení konfigurací (Configuration Management),
- reporting/viditelnost výsledků (Reporting/Visibility of Results).

3.1 Doménový objektový model

Doménový objektový model je tvořen diagramem tříd UML, který zahrnuje nejdůležitější typy objektů v problémové doméně a vztahy mezi nimi. Pro zachycení chování objektů bývá užitečné doplnit diagram tříd o sadu sekvenčních diagramů na vysoké úrovni abstrakce, které ukazují, jak objekty pro splnění svých odpovědností komunikují. Vytvoření doménového objektového modelu má zabránit vzniku nekonzistencí a nedorozumění mezi jednotlivými členy týmu. Doménový objektový model představuje celkový rámec, do kterého se zasazuje funkcionalita jednotlivých užitných vlastností. To pomáhá držet konceptuální integritu systému. Vedeni doménovým modelem týmy pro užitné vlastnosti vytvářejí podrobnější návrh pro každou skupinu vlastností. Doménový objektový model je formou dekompozice objektů. Problém je rozdělen na podstatné objekty. Návrh a implementace každé třídy představuje menší problém k řešení. Doporučená technika pro doménové objektové modelování je „modeling in color“ [Coad,1999], která využívá 4 barevné archetypy tříd, komunikující definovaným způsobem. Používání barev přidává modelu vizuální informaci a umožňuje velmi rychle vytvořit flexibilní a rozšiřitelný objektový model. Doménový objektový model je stručným a relativně dostupným způsobem uložení a komunikace informací, které každý člen týmu potřebuje.

3.2 Vývoj podle užitných vlastností

Výsledkem řady projektů vývoje softwaru je systém, který nenaplňuje požadavky uživatelů. Hlavní problém spočívá v tom, že ve funkční specifikaci se mísí funkce uživatelského rozhraní, uložení dat, síťové komunikace s byznys funkcemi. To pak často vede k tomu, že vývojáři tráví velkou část své práce řešením technologických charakteristik systému místo vývoje byznys funkcionality. Zákazník těžko ocení projekt, který dodá systém se skvělým perzistentním ukládáním objektů, ale bez funkcí na podporu podnikových procesů. FDD nabízí řešit tento problém tak, že do seznamu funkčních požadavků jsou zaznamenány jen ty, které mají hodnotu pro zákazníka. Tyto požadavky musí být vyjádřeny jazykem, kterému uživatel rozumí. FDD nazývá takové požadavky „funkce s hodnotou pro zákazníka“ nebo „užitné vlastnosti“(features). Jakmile jsou užitné vlastnosti systému jednou identifikovány, řídí další vývoj softwaru. Dodávka části infrastruktury může být důležitá někdy i kritická pro projekt, ale nemá žádný význam pro zákazníka. Ukazujeme-li postup projektu na vlastnostech, kterým zákazník rozumí, může jim přiřadit hodnotu a prioritu.

Užitná vlastnost (feature) je malá funkce s hodnotou pro zákazníka vyjádřená ve formátu <akce> <výsledek> <objekt>. Podívejme se podrobněji na jednotlivé části této definice. Užitná vlastnost je **malá funkce** - tak malá, aby byla realizovatelná během dvou týdnů. Pokud je funkce složitější, musí být rozložena na více užitných vlastností. Tím, že se udržují užitné vlastnosti malé, vidí zákazník měřitelný pokrok. To posiluje důvěryhodnost projektu a poskytuje včasnou zpětnou vazbu. Užitná vlastnost je funkce **s hodnotou pro zákazníka** a je mapována k určité činnosti v rámci podnikového procesu.

Příklady vlastností: vypočítat celkovou sumu prodeje, stanovit výkon prodejce, kontrolovat heslo uživatele. Užitná vlastnost je funkce **vyjádřená ve formátu** <akce> <výsledek> <objekt>. Přesný formát vlastností usnadňuje přechod k implementaci. Vlastnost vypočítat celkovou sumu prodeje předpokládá, že bude implementována metoda *vypoctiSumu* ve třídě *Prodej*.

3.3 Vlastnictví tříd/kódu

Vlastnictví tříd určuje, kdo, která osoba nebo role, je odpovědná za obsah třídy. Existují dva obecné přístupy k vlastnictví kódu. Individuální vlastnictví a kolektivní vlastnictví prosazované například v Extrémním programování. Protože současné objektově orientované jazyky využívají konceptu tříd, má smysl definovat třídu jako nejmenší jednotku, ke které je přiřazen vlastník. FDD prosazuje individuální vlastnictví kódu a spoléhá na jeho výhody, mezi které patří zejména definovaná odpovědnost za konceptuální integritu určité části kódu. Pokud je přidávána funkcionalita, vlastník třídy ověří, že je zachován účel třídy a změny jsou realizovány správně. Důležitý je také fakt, že existuje expert, který je schopen objasnit, jak pracuje určitá část kódu a je schopen implementovat změny rychleji. Naopak se FDD snaží pomocí dalších praktik eliminovat nevýhody individuálního vlastnictví, zejména kapacitní problémy vývojářů při potřebě upravovat kód pro více funkcí a riziko ztráty znalostí o třídě při odchodu vlastníka z projektu.

3.4 Týmy pro užitné vlastnosti

Vytvořením doménového objektového modelu identifikujeme klíčové třídy v problémové doméně. Podle praktiky individuálního vlastnictví tříd přiřadíme třídám vlastníky. Současně ale chceme vyvíjet podle užitných vlastností. Přiřadíme tedy každé užitné vlastnosti vlastníka, který bude zodpovědný správnou implementaci této vlastnosti. Protože implementace vlastnosti zpravidla vyžaduje spolupráci několika vlastníků tříd, je vytvořen tým pro užitnou vlastnost, ve kterém jsou vlastníci tříd dočasně seskupeni. Týmy pro užitné vlastnosti jsou díky malé velikosti užitných vlastností malé (3 – 6 vývojářů).

3.5 Inspekce

FDD spoléhá na inspekce, které zajišťují vysokou kvalitu návrhu a kódu. Součástí většiny metodik jsou různé formy kontrol, inspekcí, prověrek, které jsou většinou chápány jako ztráta času. Inspekce však mohou podstatně snížit počet chyb jejich včasnou detekcí. Jak uvádí Jones [Jones,1985], průměrná míra odhalených chyb je pro jednotkové testování jen 24%, pro funkční testování 35%, pro integrační testování 45%. Naproti tomu průměrná efektivita inspekcí návrhu je 55% a inspekcí kódu 60%. Primárním účelem inspekcí v FDD je odhalení defektů. Pokud se inspekce provádějí dobře, můžeme získat ještě dodatečné přínosy. Prvním je přenos znalostí. Inspekce jsou prostředkem rozšiřování znalostí a zkušeností. Tím, že zkušený vývojář prochází kód, vysvětluje použité techniky, mohou se méně zkušení vývojáři velmi rychle naučit osvědčené praktiky. Druhým vedlejším přínosem inspekcí je dodržování standardů, které je naplňováno jen pokud se v rámci inspekcí kontroluje.
Při provádění inspekcí je klíčovou otázkou jejich organizace. Každý musí chápat inspekce především jako nástroj pro odhalování chyb a případně jako nástroj učení. Inspekce se vhodně doplňují s praktikou malých týmů pro užitné vlastnosti. Pokud vývoj užitné vlastnosti neovlivňuje jiné týmy, probíhají inspekce pouze v rámci jednoho týmu.

3.6 Pravidelné buildy

V pravidelných intervalech (týdně, někdy i denně) se integruje kód všech hotových užitných vlastností do jednoho celku. Tato praktika umožňuje odhalit integrační problémy včas. Kromě toho je možné předvést část systému zákazníkovi.

3.7 Řízení konfigurací

Častým omylem bývá přesvědčení, že předmětem řízení konfigurací je pouze zdrojový kód. Opak je pravdou. Je velmi důležité uchovávat verze dokumentu specifikace požadavků, protože bývá předmětem smlouvy mezi zákazníkem a dodavatelem. Stejně tak je třeba udržovat verze analytických a návrhových artefaktů, testovacích případů, dokumentace apod.

3.8 Reporting/viditelnost výsledků

Řízení projektu je úzce svázáno s konceptem viditelnosti – schopností určit skutečný stav projektu. Jak říká Steve McConnell: „ Fungující software dává mnohem přesnější zprávu o stavu projektu než jakákoli papírová zpráva“ [McConnell, 1998].
Standardním limitovaným zdrojem v softwarovém projektu je čas. FDD se snaží minimalizovat takové činnosti jako sběr informací o stavu projektu, porady o stavu projektu, nepotřebné formuláře a neproduktivní komunikaci pokud možno automatizovaným a jednoduchým způsobem sběru těchto informací. FDD poskytuje jednoduchou metodu sběru přesných a spolehlivých informací o stavu projektu a nabízí několik formátů zpráv o stavu projektu.

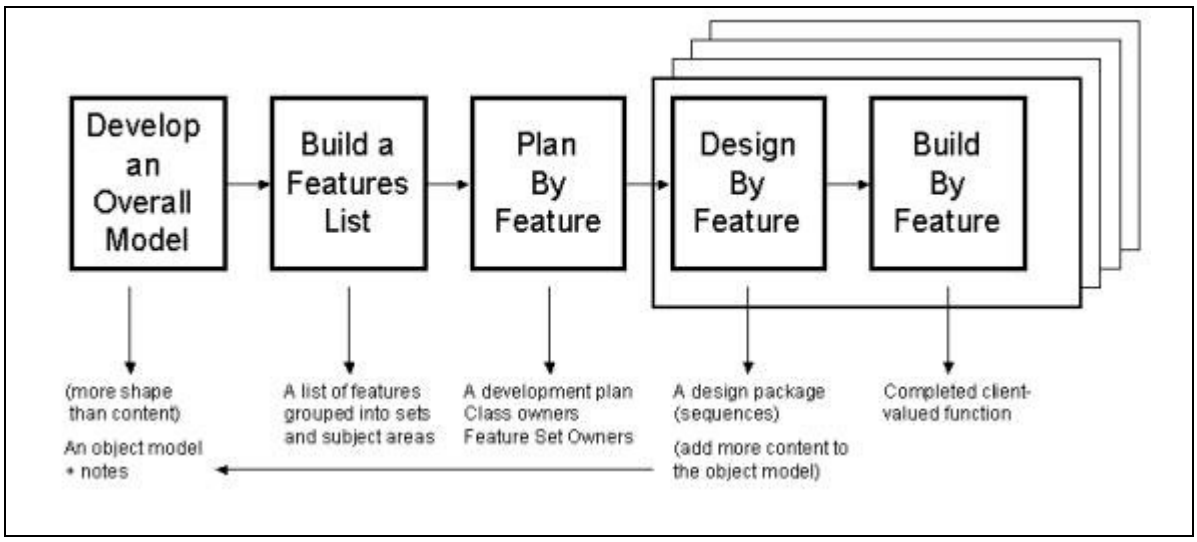
4 PROCESY FDD

FDD na rozdíl od ostatních agilních metodik popisuje postup při vývoji softwaru ve formě procesů. Význam procesů však nepřeceňuje, neboť cílem není splnění předepsaného procesu, ale vytvoření fungujícího softwaru splňujícího požadavky zákazníka. FDD definuje "lehké" procesy, každý z nich je popsán jen na 2 stránkách. Takto definovaný proces umožňuje škálovat metodiku i na větší projekty a rychleji zapojit nové zaměstnance. Popis procesu má být co nejkratší, doporučuje se použít vzor ETVX: Entry, Task, Verification, eXit, to znamená u každého procesu:

- specifikovat jasně vstupní kritéria,
- vyjmenovat úkoly, každý úkol má - název, kdo se jej účastní, zda je povinný, popis úkolu,

- nástroje verifikace,
- výstupní podmínky procesu.

Na obrázku 1 je uveden přehled procesů definovaných v metodice FDD, které jsou dále charakterizovány.



Obrázek 1 : Procesy FDD dle [Palmer,Felsing,2002]

Proces 1 Vypracování celkového modelu

Náplní tohoto procesu je hrubý model celé domény. Vývojáři i pracovníci věcné oblasti pracují společně pod vedením hlavního architekta. Pracovníci věcné oblasti prezentují úvodní představy o systému a jeho kontext. Vývojáři i pracovníci věcné oblasti vytvoří kostru modelu, potom uživatelé podrobněji specifikují použití systému. Vývojáři i pracovníci věcné oblasti pracují v malých týmech, které se zaměřují na určitou část. Výsledky jsou potom integrovány. Výstupem tohoto procesu je diagram tříd a alternativy řešení.

Proces 2 Sestavení seznamu užitných vlastností

Proces definování užitných vlastností není v podstatě nic jiného než funkční dekompozice problémové domény se zaměřením na hodnotu pro zákazníka. Výsledkem je formální, kategorizovaný seznam užitných vlastností, ve kterém je každá užitná vlastnost hierarchicky přiřazena do jedné množiny užitných vlastností (feature set) a ta je zařazena do určité hlavní množiny užitných vlastností (major feature set).

Proces 3 Plánování užitné vlastnosti

Plánování i realizace jsou řízeny seznamem užitných vlastností. Je vytvořen plánovací tým (vedoucí projektu, vedoucí vývoje a hlavní programátoři), který plánuje pořadí vývoje užitných vlastností, přiřazuje hlavního programátora ke každé užitné vlastnosti, určuje termín dokončení.

Proces 4 Návrh užitné vlastnosti

Tento proces řídí hlavní programátor, kterému je užitná vlastnost přiřazena. Pro danou vlastnost vybere třídy, které ji podporují a kontaktuje odpovídající vlastníky tříd. Tento tým vypracuje návrhový balíček, který obsahuje detailní sekvenční diagram, rozšíření tříd a metod v diagramu tříd a opět alternativy návrhu.

Proces 5 Realizace užitné vlastnosti

Každý vlastník třídy realizuje metody, vytváří testovací případy pro svou třídu a provádí jednotkové testy . Po provedení testů vlastník třídy vloží třídu do systému pro správu verzí.

5 ZÁVĚR

Rigorózní a agilní metodiky představují dvě skupiny metodik, které vycházejí z odlišných předpokladů a odlišného chápání procesu vývoje softwaru. To má za následek jiný obsah a zaměření těchto metodik a určuje i projekty, na které je vhodné tyto metodiky aplikovat. I když jsou východiska, obsah, přístupy i použití rigorózních a agilních metodik na první pohled velmi rozdílné a jejich zastánci vystupují zpravidla antagonisticky, je možné oba přístupy určitým způsobem kombinovat. Rigorózní metodiky je možné odlehčit a aplikovat v jejich rámci některý z agilních přístupů. Na druhé straně, pokud potřebujeme použít agilní metodiky na větší projekty či projekty s větší důležitostí, je třeba je více formalizovat, zařadit více dokumentace apod. Metodika Feature-Driven Development je možná takovou zlatou střední cestou. Ač se svými principy řadí mezi agilní metodiky, definuje procesy, byť odlehčené, a zdůrazňuje nutnost modelování předem. Je formálnější než ostatní agilní metodiky, a tak ji lze úspěšně použít i na větší projekty.

LITERATURA

[Buchalcevová,2002] Buchalcevová, A.: *Agilní metodiky*, In: Objekty 2002, ČZU Praha, 2002, ISBN 80-213-0947-4.

[Buchalcevová,2003] Buchalcevová, A: *Návrh metodického rámce IS/ICT*, doktorská disertační práce, VŠE Praha, 2003

[Buchalcevová,2005] Buchalcevová, A.: *Metodiky vývoje a údržby informačních systémů*, Grada, 2005, ISBN 80-247-1075-7.

[Coad,1999] Coad, P. et al.: *Java Modeling in Color with UML*, Prentice Hall, 1999

[Cockburn,1998] Cockburn, A.: *The Methodology Space*, 1998. Dostupný z WWW: http://alistair.cockburn.us/crystal/articles/ms/methodologyspace.htm

[Highsmith,2002] Highsmith, J.: *Agile Software Development Ecosystems*, Addison-Wesley, 2002, ISBN 0-201-76043-6.

[Jones,1985] Jones, C.L.: *A Proces-Integrated Approach to Defect Prevention*, IBM Systems Journal, 1985, str.150-167

[McConnell, 1998] McConnelll, S.: *Software Project Survival Guide*, Redmond, Washington, Microsoft Press, 1998

[Palmer,Felsing,2002] Palmer, S.R, Felsing,J.M.: *A Practical Guide to Feature-Driven Development*, PrenticeHall, 2002,ISBN 0-13-067615-2