AGILNÍ METODIKY A SPRÁVA POŽADAVKŮ

ing. Alena Buchalcevová, Ph.D

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

ABSTRAKT:
Jedním z nejvýraznějších rozdílů mezi agilním a tradičním vývojem softwaru je způsob přístupu ke správě požadavků. Zatímco tradiční přístupy zavádějí pro správu požadavků „těžké“ procesy, definují formální náležitosti dokumentu specifikace požadavků, vytvářejí modely funkčních požadavků, matice trasovatelnosti a používají komplexní softwarové nástroje, agilní přístupy nabízejí jednoduché techniky a nástroje, které v maximální míře podporují komunikaci vývojářů se zákazníkem a využívají faktu, že zákazník je členem týmu. Příspěvek podrobněji rozebírá odlišnosti obou přístupů ke správě požadavků.

KLÍČOVÁ SLOVA:
správa požadavků, agilní metodiky, RUP, Extrémní programování, Scrum, FDD

1 ÚVOD

Změny technologií a ekonomického prostředí, ke kterým v posledních letech dochází, vyvolávají potřebu rychlého zavedení IS/ICT a jeho přizpůsobování měnícím se podmínkám. Tradiční rigorózní metodiky vývoje softwaru přestávají v takových podmínkách vyhovovat a začínají se prosazovat agilní metodiky. Jedná se o různé metodiky, které vznikaly od druhé poloviny 90. let. Jejich primárním cílem je vytvoření fungujícího softwaru, který přináší hodnotu zákazníkovi. Aby tohoto cíle dosáhly, maximálně „odlehčují“ proces vývoje softwaru z hlediska množství vytvářených meziproduktů, modelů, dokumentace a dalších artefaktů. Zaměřují se především na principy, praktiky a přímou osobní komunikaci [4]. Každá z agilních metodik je svým způsobem specifická, ale všechny jsou postaveny na stejných hodnotách, které byly v roce 2001 definovány v Manifestu agilního vývoje softwaru a jsou propagovány zejména díky Alianci pro agilní vývoj softwaru.
Agilní metodiky se v poslední době stále více používají na projektech v zahraničí. Jedná se o plné nasazení určité agilní metodiky, kombinace různých agilních metodik a nebo aplikaci agilních přístupů v rámci tradičních metodik. Situace v ČR je poněkud jiná. Na konferencích se sice setkáváme s příspěvky o agilních metodikách a odborná česká literatura se snaží této oblasti věnovat [5], [13], ale přesto se v české praxi zatím agilní metodiky nedočkaly většího rozšíření [7].
Úspěšnost projektů vývoje informačních systémů není uspokojivá. Podle výsledků průzkumů je více než 30% všech softwarových projektů zrušeno před dokončením. Přes 70% ze zbývajících projektů nedodá požadovanou funkcionalitu. V průměru softwarový projekt dosahuje 189% plánovaných nákladů a je dokončen za 222% času [10]. Jako příčiny tohoto stavu se uvádí:

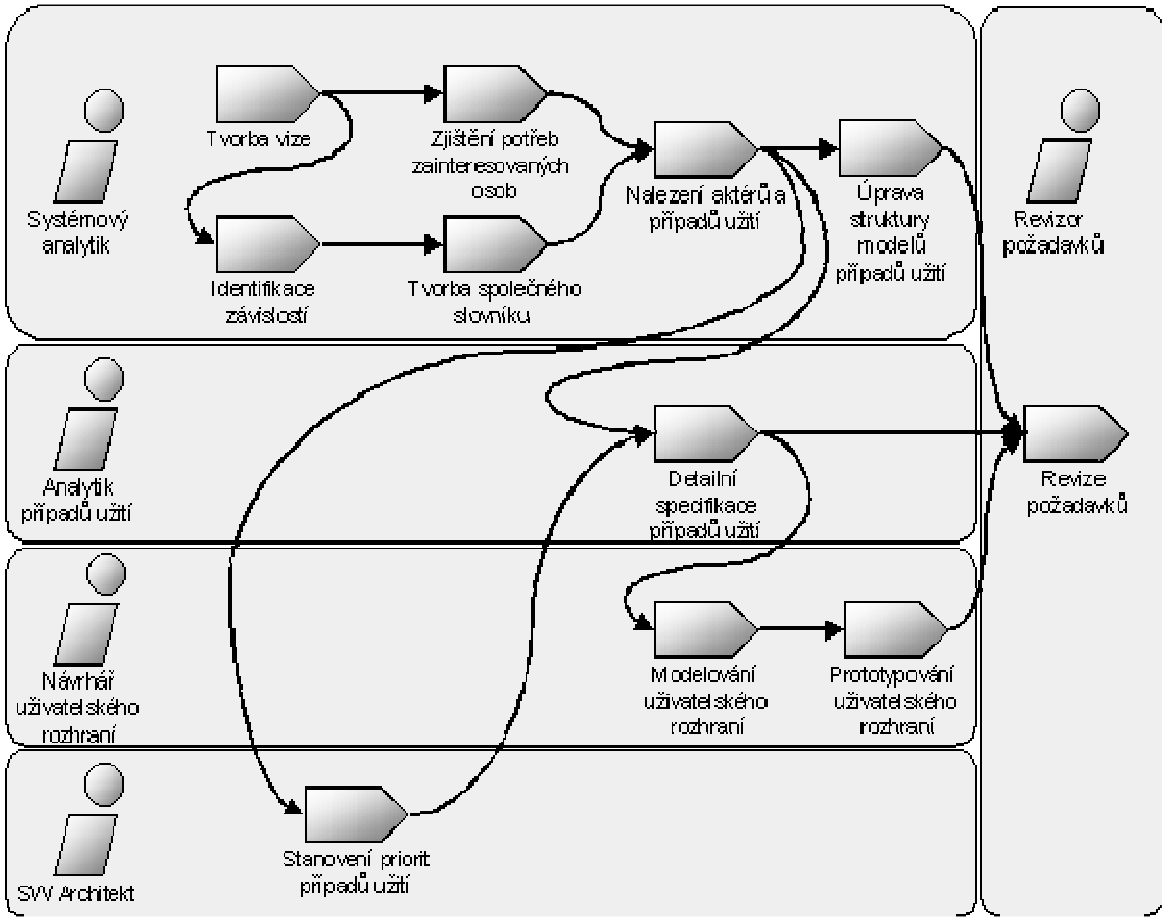
- špatná správa požadavků – vyvíjí se bez vstupu a odezvy uživatelů a bez pochopení problému, který má být řešen,
- špatná správa změn – změny požadavků a dalších produktů jsou nevyhnutelné, ale zřídka jsme schopni je trasovat a poznat jejich dopad,
- špatné řízení kvality – máme špatné metriky pro kvalitu, málo znalostí o procesech ovlivňujících kvalitu,
- špatné řízení plánu a nákladů – přesné plánování je výjimkou.

Klíčovou příčinou je správa požadavků. Pokud nepochopíme a nespecifikujeme dobře chování navrhovaného systému, nemůžeme systém dobře navrhnout, verifikovat, že splňuje požadavky, měřit dosaženou kvalitu ani definovat očekávané přínosy.

Požadavek můžeme definovat jako schopnost nebo vlastnost aplikace, kterou uživatel potřebuje k vyřešení problému a dosažení cíle. Správa požadavků je systematický přístup k identifikaci, organizaci, komunikaci a řízení měnících se požadavků na softwarovou aplikaci. Primárním cílem správy požadavků je specifikace požadavků, která definuje a dokumentuje chování budovaného systému [8]. Chyby v požadavcích mají velké dopady. Náklady na opravy chyb v požadavcích jsou zpravidla více než 10 krát vyšší než na opravu jiných chyb. Chyby v požadavcích jsou přitom nejčastější, představují více než 40% všech chyb softwarových projektů [10].
Role požadavků při vývoji softwaru je nezpochybnitelná a uvědomují si ji jak představitelé tradičních, tak agilních metodik. Na způsob, jak se s problémem požadavků vypořádat, mají však odlišný pohled.

2 SPRÁVA POŽADAVKŮ V TRADIČNÍCH METODIKÁCH

Správa požadavků je v tradičních metodikách založena na standardizovaných procesech, které definují pravidla pro zadávání, zpracování a sledování požadavků. Správa požadavků je chápána jako základní proces na úrovni zralosti 2 (podle CMMI) a jeden z prvních kroků, který by organizace měla udělat pro zlepšení zralosti svých procesů. Podívejme se na postup správy požadavků podle metodiky Rational Unified Proces, který je znázorněn na obrázku 1.
.



Obrázek 1 : Správa požadavků dle RUP zdroj [16]

Požadavky jsou zaznamenávány předepsaným způsobem. Potom se kontrolují, seskupují podle souvislosti, odstraňují se duplicity a posuzují z hlediska realizace požadavku a jeho dopadu na existující systémy. Odhaduje se termín realizace a potřebné zdroje. Po posouzení požadavků a odsouhlasení jejich realizace následuje vytvoření vize budoucího softwaru. Vize shrnuje klíčové potřeby zadavatele a popisuje požadované funkční i systémové vlastnosti softwaru. V průběhu správy požadavků se vytváří společný slovník, který má za úkol sjednotit používanou terminologii a pomáhá odstranit možné duplicity způsobené rozdílnou formulací téhož problému. V této fázi vzniká úvodní model případů užití, který obsahuje klíčové funkce systému. Model případů užití je dále zpřesňován. Snahou je identifikovat všechny aktéry a případy užití. Pro případy užití jsou definovány priority a rizika. Na základě klíčových případů užití je definována architektura systému. Případy užití jsou pak detailně popsány (vstupní i výstupní podmínky, scénáře případu užití). Výsledkem je detailní specifikace případů užití. Požadavky dle RUP je třeba řídit v rámci celého životního cyklu požadavků včetně řízení jejich změn. Kvalitní správa požadavků je velice složitý proces, při němž se doporučuje využívat softwarových nástrojů, které umožňují efektivněji organizovat a centralizovat procesy a zároveň přispívají k flexibilnější komunikaci mezi všemi účastníky

3 AGILNÍ SPRÁVA POŽADAVKŮ

Jedním z nejvýraznějších rozdílů mezi agilním a tradičním vývojem softwaru je právě způsob přístupu k požadavkům. Podle Manifestu agilního vývoje softwaru [9] dávají agilní metodiky přednost *spolupráci se zákazníkem* před sjednáváním kontraktu a mění tak zásadně koncept specifikace požadavků. Východiskem je přesvědčení, že není možné dohodnout a podepsat požadavky na začátku projektu. Proto se dle agilních metodik definují na začátku jen hrubé požadavky, které se potom na základě každodenních jednání s uživateli zpodrobňují a dokonce i mění. To zdůrazňuje spoluúčast uživatelů na definování požadavků a přenesení odpovědnosti za projekt na uživatele.

Správa požadavků v tradičním pojetí je o dokumentech. Otázkou ale je, jak dobře můžeme zachytit znalost v dokumentu bez jejího zkreslení. Dokumenty mají několik omezení – jsou jednosměrné (pouze od autora k čtenáři), autor píše jen ze svého pohledu, nejednoznačné, mohou být vágní. Proti dokumentům staví agilní metodiky osobní komunikaci, která je daleko efektivnější, a doplňují ji jednoduchými technikami pro záznam požadavků. Vycházejí z toho, že modely a dokumenty jsou jen vedlejší produkty při vývoji softwaru a primárním artefaktem je spustitelný kód a testy. Proto doporučují „just enough requirements management“. Cílem vývojářského týmu není totiž napsat perfektní dokument specifikace požadavků, ale dodat včas a efektivním způsobem fungující software, který uspokojí potřeby zákazníka.

Agilní metodiky mají také specifický přístup k otázce trasovatelnosti požadavků. Trasovatelnost požadavků umožňuje jednak propojit požadavky mezi sebou (horizontální trasovatelnost), jednak je propojit s dalšími artefakty při vývoji – modely, kódem, testy (vertikální trasovatelnost). Tradiční metodiky (např. CMMI) doporučují vytvářet matici trasovatelnosti (traceability matrix), která udržuje metadata o vztazích mezi artefakty. Udržovat tuto matici ale vyžaduje hodně práce a použití nástrojů na správu požadavků (například Rational Requisite Pro, Borland CaliberRM nebo Telelogic DOORS). Na tuto činnost je třeba vyhradit nejméně jednoho člověka a je velmi náročné udržet matici v aktuálním stavu. Agilní vývojáři si uvědomují, že trasovatelnost požadavků je důležitá, ale neplýtvají časem na vytváření a udržování matice trasovatelnosti. Místo toho používají nástroje a techniky, které umožňují trasovat požadavky přímo. Například každému požadavku přiřadí jedinečné číslo, které se pak uvádí u testů a ve zdrojovém kódu. Nebo ještě lépe, pokud se definují požadavky ve formě akceptačních testů, existuje jen jediný zdroj, se kterým se pracuje [1].

Dalším poměrně revolučním přístupem agilních metodik ke správě požadavků je myšlenka spravovat požadavky stejně jako hlášení o chybách. Hlášení o chybě je chápáno jako požadavek. A opačně každý požadavek je vlastně chybové hlášení typu chybějící chování. Požadavky a hlášení o chybách jsou udržovány v jednom nástroji a je s nimi nakládáno stejným způsobem. Příkladem takového postupu je například v současnosti největší agilní projekt Eclipse, kdy jsou požadavky spravovány v Open Source nástroji na sledování chyb Bugzilla. Při společném nakládání s požadavky a chybami se ale můžeme dostat do problému u projektu s fixní cenou, kdy je třeba zvlášť udržovat původní dohodnuté požadavky a požadavky na změny [2].
V následujících odstavcích se podíváme detailněji na koncept požadavků u některých agilních metodik.

3.1 Požadavky v metodice Extrémní programování

Extrémní programování (XP) je velmi „lehká“, ale přesto disciplinovaná metodika, která zavádí specifické praktiky jako párové programování, refaktorizace, testy před kódováním a další. Požadavky v XP se definují ve formě uživatelských příběhů (user story), které jsou náhradou dokumentu specifikace požadavků. Uživatelské příběhy píší zákazníci a vyjadřují v nich své potřeby, které má systém podpořit. Podobají se scénářům užití, ale nepopisují jen uživatelské rozhraní, ale i hodnotu pro zákazníka. Popis je stručný, cca 3 věty až odstavec, a používá se terminologie zákazníka. XP poskytuje formalizovaný návod jak strukturovat popis uživatelského příběhu v angličtině [17]:

As a **[Type of User]**, **[Function to Perform]** *so that* **[Business Value]**

Například

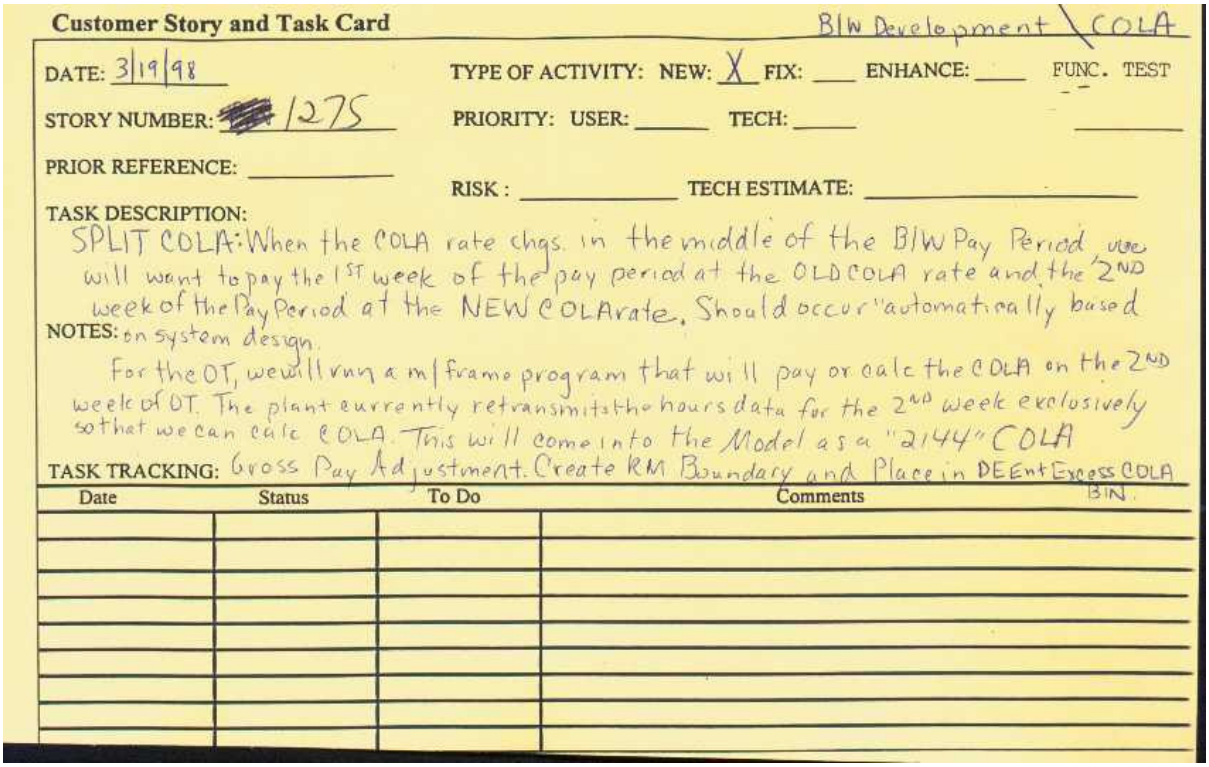
- As a sales person, I want to add a new contact so that I can follow up later with prospects.
- As a sales manager, view new contacts added by salesperson, so I can track leads
- As a system administrator, add a new sales person so they can access the system

V češtině bychom mohli definovat formát následovně:

Jako **[role uživatele]**, **[funkce, která má být provedena]** *a tak* **[byznys hodnota]**

Není bezpodmínečně nutné používat tento formát pro popis, ale je dobré jej mít jako myšlenkový koncept, který pomáhá orientovat se na byznys hodnotu požadavku pro zákazníka, nikoli na technologie.
Po sepsání uživatelských příběhů (user stories) je zákazníci kategorizují podle toho, jakou pro ně mají hodnotu (nutné, důležité, „bylo by dobré“). Vývojáři pak pro každý příběh odhadují čas potřebný pro implementaci, přičemž ne všechny příběhy mohou být v daném okamžiku odhadnuty bez dalších podrobností. Zákazníci potom vyberou rozsah a datum další dodávky (release) a vývojáři výběr potvrdí. Dodávka je rozdělena na několik iterací o délce maximálně 3 týdny. Výsledek iterace není dodáván zákazníkovi.
Hlavní rozdíl mezi uživatelskými příběhy a tradiční specifikací požadavků je v úrovni podrobnosti. Uživatelské příběhy mají takovou úroveň podrobnosti, aby bylo možné odhadnout rizika a dobu implementace (1, 2 až 3 týdny). Teprve až bude příběh vybrán pro implementaci, vývojáři si vyjasní podrobnosti na základě osobní komunikace se zákazníkem. Uživatelské příběhy mají 3 důležité aspekty, které XP vyjadřuje jako CCC (Card, Conversation, Confirmation) - karta, konverzace, potvrzení.
Uživatelské příběhy se píší na karty (viz obrázek 2). Karta neobsahuje všechny informace, které požadavek zahrnuje, ale je symbolem požadavku – obsahuje právě tolik textu, aby byl požadavek identifikován. Používá se pro plánování. Zapisují se na ni poznámky týkající se nákladů a priority. Karty jsou velice flexibilní, je možné je mít v libovolném pořadí, je možné

je snadno předávat. Například když je příběh vybrán pro implementaci, předává se karta programátorům a po dokončení implementace se vrací zpět k zákazníkovi.
Požadavek jako takový je komunikován mezi zákazníkem a programátorem prostřednictvím konverzace. Konverzace probíhá opakovaně, nejprve při plánování dodávky a potom znovu při plánování iterace. Konverzace je hlavně ústní, ale může být doplněna dokumenty. Konverzace ale nestačí, potřebujeme nějaký způsob potvrzení zjištěných faktů. Potvrzení je realizováno definováním akceptačních testů. Pro každý uživatelský příběh musí být vytvořen jeden nebo více akceptačních testů, které potom ověří, že byl příběh správně implementován.


Customer Story and Task Card

B1W Development \ COLA

DATE: 3/19/98

TYPE OF ACTIVITY: NEW: X FIX: ___ ENHANCE: ___ FUNC. TEST

STORY NUMBER: 1275

PRIORITY: USER: ___ TECH: ___

PRIOR REFERENCE: ___

RISK: ___ TECH ESTIMATE: ___

TASK DESCRIPTION:
SPLIT COLA: When the COLA rate chgs. in the middle of the B1W Pay Period user will want to pay the 1ST week of the pay period at the OLD COLA rate and the 2ND week of the Pay Period at the NEW COLA rate. Should occur automatically based on system design.

NOTES:
For the OT, we will run a m/frame program that will pay or calc the COLA on the 2ND week of OT. The plant currently retransmits the hours data for the 2ND week exclusively so that we can calc COLA. This will come into the Model as a "2144" COLA

TASK TRACKING: Gross Pay Adjustment. Create RM Boundary and Place in DEEntExcessCOLA BIN.

Date	Status	To Do	Comments

Obrázek 2 : Karta pro zápis uživatelského příběhu zdroj [15]

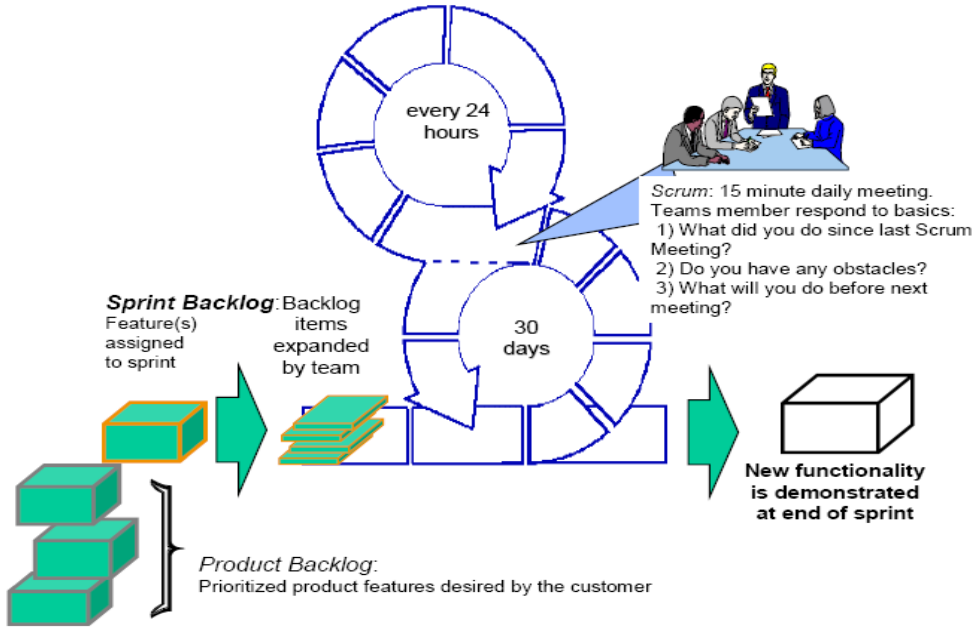
3.2 Definice užitných vlastností v FDD

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. Metodika Feature Driven Development (FDD) nabízí řešit tento problém tak, že do seznamu funkčních požadavků jsou zaznamenány jen ty požadavky, 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.
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*. Výsledkem specifikace požadavků dle FDD je formální, kategorizovaný seznam užitných vlastností, ve kterém je každá užitná vlastnost hierarchicky přiřazena do množiny užitných vlastností (feature set) a hlavní množiny užitných vlastností (major feature set).
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 a termín dokončení. Každé užitné vlastnosti přiřadí vlastníka - hlavního programátora, který je zodpovědný za správnou implementaci této vlastnosti. Návrh a implementace probíhá také podle užitných vlastností. Tento proces řídí hlavní programátor, kterému je užitná vlastnost přiřazena. 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) [6].

3.3 Scrum a požadavky

Přístup Scrum je založen na přesvědčení, že vývoj softwaru není definovaný proces, jak rigorózní metodiky předpokládají, ale empirický proces, a proto vyžaduje úplně odlišný styl řízení. Název Scrum byl vybrán podle skrumáže (mlýna) v rugby proto, aby zdůraznil, že metodika Scrum je podobně jako hra rugby adaptivní, rychlá a samoorganizující. Metodika Scrum je zaměřena především na řízení projektu. Vývoj softwaru probíhá v rámci 30 denních iterací nazývaných Sprint, na jejichž konci je dodána vybraná množina užitných vlastností. Klíčovou praktikou metodiky Scrum je používání každodenních 15 minutových porad (*Scrum Meetings)*, které slouží pro koordinaci a integraci prací. Scrum definuje jen 3 role - Product Owner, Team, ScrumMaster. Product Owner spravuje seznam požadavků (Product Backlog) tak, aby maximalizoval hodnotu projektu. Team je kros-funkční skupina lidí, kteří se sami řídí tak, aby v každém sprintu dodali fungující SW. ScrumMaster zodpovídá za Scrum proces, jeho správnou implementaci a maximalizaci užitku.



Obrázek 3 : Postup dle Scrum [18]

Sprint je 30 denní iterace, na jejímž začátku se koná Sprint Planning Meeting. Jeho cílem je definovat seznam požadavků, které mají být realizovány v rámci sprintu (Sprint Backlog). Do sprintu se vybírají požadavky s nejvyšší prioritou ze seznamu požadavků produktu (Product Backlog).

4 ZÁVĚR

Chyby v požadavcích jsou hlavním důvodem toho, že softwarové projekty převyšují rozpočet, jsou dodány mnohem později, než bylo plánováno, je redukován rozsah řešení, jsou špatné kvality, nejsou po dodání používány a nebo jsou předčasně ukončeny. Přitom chyby v požadavcích jsou nejčastější chyby při vývoji softwaru a také chyby, jejichž odstranění je nejnákladnější. To dokumentuje i analýza rizik uvedená v [14]. O zlepšení stavu v oblasti požadavků se snaží jak tradiční, tak agilní metodiky. Způsob, jaký doporučují pro zlepšení správy požadavků, je však diametrálně odlišný.

Tradiční metodiky chápou správu požadavků jako jeden z prvních kroků pro zlepšení zralosti softwarových procesů. Zavádějí „těžké“ procesy pro správu požadavků, definují formální náležitosti dokumentu specifikace požadavků, vytvářejí modely funkčních požadavků, matice trasovatelnosti a používají komplexní nástroje na správu požadavků. Agilní přístupy nabízejí jednoduché techniky a nástroje, které v maximální míře podporují komunikaci vývojářů se zákazníkem a využívají faktu, že zákazník je členem týmu.

LITERATURA

1. Ambler, S.W: Don’t Enter the Matrix, SD Agile Modeling Newsletter, October 2005
2. Ambler, S.W: Rethinking how you view requirements management, Dr. Dobb’s Agile Modeling Newsletter, January 2007
3. Beck, K. Extrémní programování, Praha: Grada Publishing 2002, 160 stran, ISBN 80-247-0300-9
4. Buchalcevová A. Agilní metodiky, Objekty 2002

5. Buchalcevová, A. Metodiky vývoje a údržby informačních systémů, Grada publishing, 2005, ISBN 80-247-1075-7
6. Buchalcevová, A.: Metodika feature-driven development neopouští modelování a procesy, a přesto přináší výhody agilního vývoje. In: Tvorba softwaru 2005. Ostrava : Tanger, 2005, s. 25–30. ISBN 80-86840-14-X
7. Buchalcevová, A., Leitl, M.: Průzkum používání agilních metodik v ČR. In: Objekty 2006. Praha : PEF ČZU, 2006, s. 125–136. ISBN 80-213-1568-7
8. Davis, A.M, Leffingwell, D. A: Using requirements management to Speed Delivery of Higher Quality Applications, Rational Software Corporation
9. Fowler, M. The Agile Manifesto: where it came from and where it may go, http://martinfowler.com/articles/agileStory.html
10. Frankl, A. King, T.: How to detect requirements errors, White paper, n8 Systems, July 2005
11. Jeffries, R.: Essential XP: Card, Conversation, Confirmation, dostupný z http://www.xprogramming.com/xpmag/expCardConversationConfirmation.htm
12. Johnson, J.–Boucher K.D– Connors K.– Robinson J.: Collaborating on Project Success, Software Magazine, February/March 2001
13. Kadlec, V. Agilní programování: Metodiky efektivního vývoje softwaru, Computer Press, 2004, ISBN 80-251-0342-0
14. Lacko, B.: Aplikace metody RIPRAN v softwarovém inženýrství. In: Tvorba softwaru 2001, TU- VSB Ostrava 2001, str.97-103
15. http://www.extremeprogramming.org/rules/userstories.html
16. http://objekty.vse.cz/Objekty/Rup4
17. http://www.extremeplanner.com/resources/Agile-Requirements.html
18. http://www.controlchaos.com/