Plánovanie a agilné metodológie vývoja softvéru

MAREK TOMŠA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
marek.tomsa@gmail.com

Abstrakt. O zmysle a nutnosti plánovania pri riadení akéhokoľvek projektu je asi zbytočné diskutovať. Aj keď sú známe všeobecné metódy vytvárania a manažovania plánov, konkrétna aplikácia metódy by mala vždy zohľadňovať špecifiká projektu. Metódy tvorby plánov sa ukazujú byť škálovateľné pre rôzne veľkosti projektu, avšak v prípade softvérových projektov, a navyše špecifických softvérových projektov so slabou znalosťou problémovej oblasti, prípadne nemožnosťou vytváraný produkt dopredu plne špecifikovať, prichádza nutnosť nielen prispôsobiť veľkosti projektu a podmienkam samotný proces tvorby softvéru, ale aj proces tvorby plánov a ich riadenia. Táto esej rozoberá problematiku plánovania v kontexte projektov aplikujúcich dnes moderné agilné metodológie a diskutuje osobitosti pri plánovaní ťažko špecifikovateľných rýchlo sa meniacich a vyvíjajúcich softvérových projektov. Na príklade agilnej metodológie riadenia projektu Scrum uvádzame špecifické aspekty prispôsobenia tradičných metód tvorby a manažmentu plánov pre malé agilné tímy.

Úvod

Nikto určite nepochybuje o tom, že bez plánovania sa žiadny projekt netriviálneho rozsahu nemôže zaobísť. Závislosť rýchlosti ukončenia projektu od počtu ľudí v ňom nasadených je neodškriepiteľná. Do tejto závislosti však vstupuje aj schopnosť organizácie úsilia účastníkov projektu, a vo väčšine prípadov ju nemožno označiť za lineárnu. Ak by prebiehala práca desiatky inžinierov v projekte tak, že každý by čakal na výsledok práce toho, kto robí pred ním, nielenže by nedosiahli ukončenie projektu za desaťkrát menší čas, ale pravdepodobne by sa stalo, že projekt by bol dokončený neskôr ako keby na ňom robil jeden človek. Prínos vyplývajúci z počtu ľudí podieľajúcich sa na projekte vyplýva v prvom rade z možností ich činnosti paralelizovať, a tiež zo schopnosti manažéra tieto činnosti naplánovať tak, aby boli s ohľadom na zdroje naozaj vykonateľné paralelne.

Manažment v softvérovom inžinierstve, október 2006, s. 1-8.

Čím je tím väčší, tým väčšia je réžia na jeho riadenie a najmä na komunikáciu jeho členov. Malé tímy, pozostávajúce z približne piatich ľudí, majú predpoklad podať pri správnom prístupe najvyšší výkon na jedného pracovníka. Ak berieme do úvahy straty spôsobené nutnou komunikáciou a riadením, mali by dvaja ľudia spolu spraviť menej ako je súčet práce, ktorú by spravil každý sám. V ideálnych tímoch by to takto nemalo byť. Synergický efekt plynúci zo spolupráce, determinovaný často kvalitami členov tímu, spôsobujúci, že dvaja spolu spravia viac ako je súčet ich práce, by mal byť vyšší ako straty, ktoré priniesla nutnosť komunikovať a riadiť týchto ľudí. Na oba faktory má vplyv spôsob riadenia projektu. Ak sú vytvorené podmienky na maximalizáciu synergického efektu a minimalizáciu strát, bol splnený prvý predpoklad na to, aby práca na projekte bola správne efektívna.

Kvalitatívne aspekty riadenia, spočívajúce v schopnosti manažéra projekt naplánovať a najmä zmanažovať tak, aby k spomínanému synergickému efektu za správnych okolností zloženia tímu došlo, stavajú najmä na jeho osobných kvalitách a skúsenostiach. V tejto eseji sa sústredíme v prvom rade na procesné aspekty plánovania, a to v prostredí malých tímov aplikujúcich agilné metodológie riadenia a vývoja softvéru.

Plánovanie

Každý projekt má dostupný istý konečný objem zdrojov. Vo svojej podstate má svoju množinu dostupných zdrojov každý z nás, danú minimálne tým, ako dlho žijeme. Pri projektoch bývajú zdroje často krát obmedzené oveľa viac ako len dĺžkou ľudského života. Do hry vstupujú faktory ako financie, čas, požiadavky a obmedzenia obchodného prostredia, ale aj osobné nasadenie účastníkov projektu. Plánovanie môžeme chápať ako tvorbu rozvrhu, ktorým ohraničenia prostriedkov odhadujeme a deklarujeme, ako budú tieto obmedzené prostriedky v priebehu projektu využívané tak, aby priniesli želaný výsledok.

Z pohľadu oblastí riadenia projektu je možné plánovať rozsah projektu, náklady, časovú náročnosť a následnosť činností, či pridelenie ľudských zdrojov k jednotlivým činnostiam. Čo však nie je možné naplánovať sú choroby členov tímu, živelné pohromy, zmeny v legislatíve, politický prevrat, zvýšenie cien materiálu a všetky ostatné udalosti, ktoré tu nie je možné vymenovať, pretože sú vo svojej podstate neznáme. Alternatívy môžu zahrnúť pesimistický a optimistický odhad. Ani pesimistický odhad však nemôže zahŕňať alternatívy, ktoré môžu byť dôsledkom týchto nepredvídateľných udalostí. K plánovaniu projektu je však nutné pristupovať tak, aby nás ani neplánované udalosti neprekvapili. Toto je možné dosiahnuť tým, že tvorca plánu bude pripravený tento plán aktualizovať a bude mať dopredu premyslené, čo robiť v prípade nepredvídateľnej udalosti.

Ďalšie dôvody plánovania sú zrejmé – v každom projekte je nutné kontrolovať jeho progres, tento referovať smerom k ľuďom, ktorí majú na ňom záujem – v prípade softvérových projektov často zákazníci, vysoký manažment firmy alebo investori. Občas je nutné projekt zhodnotiť a rozhodnúť sa, či je vôbec výhodné v ňom

s ohľadom na dostupné zostávajúce zdroje pokračovať. Toto by nebolo možné bez dopredu vypracovaného plánu.

Nielen spomínané nepredvídateľné udalosti, ale aj prirodzená chyba odhadu, ktorá nutne sprevádza každý plán a môže nadobudnúť veľkosť celého jedného rádu, spôsobujú, že reálny progres na projekte sa skôr či neskôr odchýli od plánu. Vtedy treba plán aktualizovať. Toto môže byť ťažká úloha, pokiaľ pre ňu nie sú vytvorené predpoklady vyplývajúce zo samotného softvérového procesu. O problémoch tradičných modelov vývoja softvéru a ich možnom riešení v agilných prístupoch pojednáva nasledujúca kapitola.

Tradičné vs. agilné metodológie vývoja softvéru

V 21. storočí informačné technológie zasiahli do každej oblasti ľudského života. Softvérové systémy sa netvoria už len pre vedecké účely, a už dlho tiež nie len pre kancelárske použitie, ale dostávajú sa do všetkých oblastí ľudského života. Dnes takmer každý vlastní mobil, ktorý má procesor výkonnejší ako boli počítače veľkosti troch miestností pred pár desiatkami rokov. Otázka realizácie konceptov, ktoré posúvajú ľudstvo dopredu, čím ďalej tým menej naráža na technické problémy a stále viac sa opiera na schopnosti softvérových inžinierov potrebné riešenie vymyslieť, navrhnúť a zrealizovať. V tejto oblasti bol dosiahnutý obrovský pokrok.

Obchodné prostredie si zároveň nepriamo vyžaduje aby bol proces vývoja softvéru efektívny. Neefektívny proces má takmer zákonite za následok zvýšenie nákladov, oneskorenie projektu, alebo dokonca neschopnosť dodať požadovaný produkt. Zatiaľ čo technické a implementačné prostriedky prešli v posledných rokoch búrlivým vývojom a navrhnúť a implementovať systém veľkého rozsahu je vďaka nim stále ľahšia úloha, ukazuje sa, že najväčšou slabinou býva práve proces riadenia životného cyklu softvéru. Stále sa hľadá, ale nenachádza, tzv. „silver bullet“ riešenie, ktoré vyrieši všetky problémy so softvérovým procesom. Ukazuje sa, že proces vývoja softvéru je príliš komplexný a diverzný na to, aby existovalo jediné takéto riešenie [2].

Tradičným modelom vývoja softvéru je vodopádový model. Je realizáciou lineárneho prístupu k vytvoreniu produktu, kedy sú jednotlivé fázy na seba naviazané v postupnosti. Po analýze a špecifikácii problému prichádza hrubý a po ňom detailný návrh celého riešenia a nasleduje implementácia a testovanie výsledného produktu. Takýto model má jednu zásadnú nevýhodu. Vo svojej podstate predpokladá dokonalú znalosť problémovej oblasti, dokonalú špecifikáciu systému, návrh, od ktorého sa dá priamočiaro prejsť k implementácii, a výsledok, ktorý je jednoducho otestovateľný ako celok. Odmieta akékoľvek zmeny v špecifikácii vo fáze implementácie. Rigorózne dbá na dokumentovaní jednotlivých fáz vývoja, s čím prichádza jeho hlavná a asi aj jediná výhoda - relatívne vysoká miera viditeľnosti a zdokumentovania procesu, pretože každá fáza životného cyklu poskytuje dokument o výsledku, ktorý je vstupom do nasledujúcej fázy.

Modernejším prístupom je iteratívny špirálový model. Tento čiastočne stiera ťažkopádnosť vodopádového modelu, ale stále je len opakovaním niekoľkých malých lineárnych vodopádov. Pružnosť reakcie na zmenu v požiadavkách je vyššia, avšak,

ako uvádza Schwaber v [4], rovnako ako vodopádový model, aj špirálový model sa spolieha na chybný predpoklad, že softvérové procesy sú definované a predikovateľné procesy. Aj bez prihliadnutia na nevhodnosť lineárneho prístupu a s obmedzením negatívnych dopadov tohto prístupu dokonalou znalosťou problémovej oblasti a bezchybnou špecifikáciou, ktorá naozaj vystihuje to, čo zákazník požaduje, nie je možné vylúčiť výskyt nepredvídateľných udalostí. Rigoróznosť sprevádzajúca všetky tradičné modely vývoja softvéru bráni efektívnej možnosti flexibilne sa vysporiadať s týmito nepredikovateľnými udalosťami.

V poslednej dobe sa stávajú čoraz populárnejšími agilné metodológie vývoja softvéru. Tieto zahŕňajú prístupy ako Adaptívny vývoj softvéru, eXtreme Programming, Crystal Clear, Scrum a niekoľko ďalších. Agilné metodológie vznikli v prvom rade ako pokus o odľahčenie ťažkopádneho tradičného rigorózneho vodopádového modelu. Vyzdvihuje sa komunikácia medzi jednotlivými členmi tímu namiesto dokumentovania. Dbá sa na výsledok vo forme funkčného softvéru namiesto vzornej dokumentácie niečoho, čo síce funguje presne podľa špecifikácie, ale nie je vo výsledku tým, čo zákazník skutočne chcel. V neposlednom rade prívrženci agilného prístupu hlásajú, že je vždy lepšie byť pripravený reagovať na zmenu, ako byť schopný nasledovať a dokonale naplniť plán.

Scrum a plánovanie

Je prirodzené, že ani agilné metódy nemôžu preskočiť proces získavania iniciálnych požiadaviek od zákazníka, a tiež nie je možné vyvíjať systém bez toho, aby bola definovaná jeho architektúra na vyššej úrovni.

Metodológia Scrum rozdeľuje proces vývoja softvéru na tzv. predohru, hru a dohru. Predohra zahŕňa plánovanie a návrh architektúry produktu na vysokej úrovni. Hra predstavuje postupnosť periodických „šprintov“ (sprints) v trvaní preddefinovanej doby, zvyčajne jedného až štyroch týždňov. Každý šprint zahŕňa analýzu danej časti problému, návrh zapracovania zmien a samotnú implementáciu zmien. Na konci šprintov sa jednotlivé časti integrujú do dovtedajšieho riešenia, spraví sa prehľad a definuje sa nová funkcionalita vo forme tzv. balíčkov (packets). Každý člen vývojového tímu prezentuje výsledok svojej práce a diskutujú a riešia sa problémy, na ktoré sa narazilo. Zároveň sa vytvárajú nové balíčky a zakladajú sa do tzv. knihy nedokončených prác (backlog), ktorá predstavuje množinu balíčkov, čakajúcich na pridelenie. Na konci šprintu sa organizuje tzv. recenzia (review) s účasťou zákazníka, vysokého manažmentu, prípadne marketingu alebo iných osôb. Je prezentovaná funkčná verzia systému, revidujú sa požiadavky na ďalší šprint a zavádzajú sa nové položky do knihy nedokončených prác.

Udržiavaním knihy nedokončených prác má tím neustále prehľad, koľko práce ostáva nedokončenej z minulého šprintu. Každý tím (ak ich je viac) dostane pre daný šprint množinu balíčkov, ktoré je potrebné do konca ďalšieho šprintu implementovať. V tíme sa jednotlivé balíčky ďalej rozdeľujú podľa skúseností a špecializácie konkrétneho vývojára. Balíčky bývajú rozlíšené funkčne (orientované na konkrétnu

vyžadovanú vlastnosť produktu) alebo systémovo (orientované na konkrétnu vrstvu aplikácie).V momente kedy sa manažment rozhodne, že produkt je po aktuálnom šprinte pripravený na vydanie, sa pristúpi k dohre, ktorá zahŕňa intenzívne testovanie systému ako celku, kompletizáciu používateľskej dokumentácie a prípadnú prípravu marketingových a tréningových materiálov.

Aj keď sa môže zdať, že agilné prístupy sú výmyslom nedisciplinovaných programátorov, snažiacich sa vyhnúť sa neobľúbenej dokumentačnej práci, opak je pravdou. Agilné prístupy vyžadujú od členov tímu disciplínu v komunikácii, ktorej nedostatok sa nielen prejaví oveľa rýchlejšie ako nedostatok v dokumentovaní, ale práca na projekte si ho inherentne vyžaduje. Nie je možné, aby tím spolu vôbec nekomunikoval a nikto si to nevšimol. Je však dosť pravdepodobné, že sa tímu podarí vytvárať nepoužiteľnú dokumentáciu len z dôvodu aby nejaká bola. Vďaka vyzdvihovaniu komunikácie sa produkt stáva kolektívnym dielom, ktorého vlastníkom sú všetci členovia tímu.

Čo toto všetko znamená pre tradičný model plánovania? Potrebujú vôbec agilné tímy plány? Odpoveď je prirodzene kladná. Dokonca je možné aplikovať tie isté prístupy k plánovaniu ako v projektoch vyvíjajúcich softvér tradičnými metodológiami.

Iniciálna fáza plánovania v procese Scrum zahŕňa definovanie prvej verzie obsahu knihy nedokončených prác, definovanie rozsahu funkcionality pre jednotlivé verzie produktu, výber najvhodnejšej verzie pre prvú fázu vývoja, namapovanie balíčkov na jednotlivé položky knihy nedokončených prác, a prirodzene naplánovanie rozpočtu. Ak by sme skončili pri iniciálnom plánovaní, v zásade by sa plánovanie v Scrume nelíšilo od plánovania v akomkoľvek inom softvérovom procese. Rozdiely prichádzajú v realizácii procesov riadenia plánov v priebehu projektu.

V prípade zjemňovania plánov je to vo frekvencii tohto zjemňovania, v prípade odhadu potrebného úsilia je rozdiel vo veľkosti práce ktorú treba odhadnúť. Rozdelením projektu na šprinty je nielen možné každú z fáz vývojového cyklu (fáz v zmysle definovaného balíčka funkcionality) z nich lepšie pochopiť, ale prirodzene aj naplánovať. Jednoduchý model komunikácie v tíme toto zjemňovanie plánov ďalej zjednodušuje, keďže na to, aby sa zistilo, že je nutné vyriešiť nejaký neočakávaný problém a prispôsobiť sa mu, alebo zmeniť prístup v implementácii konkrétnej časti projektu, nie je často potrebné čakať na pokyn od vedenia, zapracovanie novej požiadavky do špecifikácie, odsúhlasenie zákazníkom, a súhlas návrhára s modifikáciou architektúry. Stiera sa rozdiel medzi projektovým manažérom, programátorom, testerom a údržbárom kódu. Tím získava schopnosť riadiť sám seba, vývojári určujú ako je nutné ďalej napredovať, pretože práve oni to vedia najlepšie.

Je známym faktom, že v prípade preplánovania času využijú ľudia vždy všetky dostupné časové prostriedky. Scrum so svojim princípom šprintov nediktuje vývojárom dokedy majú konkrétnu vec spraviť, hovorí len o tom, kedy má byť hotová ďalšia iterácia prototypu s pridanou funkčnosťou, pripravená na prezentáciu. Jej určenie býva dosiahnuté kolektívnou dohodou. Toto čiastočne vylučuje situáciu, kedy tvorca plánu stanoví termín splnenia čiastkovej úlohy na mesiaca a pracovník ho, tak ako to býva, stihne s minimálnou rezervou napriek tomu, že úloha mu v skutočnosti zabrala týždeň.

Highsmith [3] opisuje projekty využívajúce agilné metodológie ako komplexné adaptívne systémy (CAS). CAS sú nelineárne, otvorené a dynamické. V nelineárnych systémoch nie je hodnota celku zhodná súčtu jeho častí. Otvorený systém komunikuje s okolím a poskytuje okoliu výstupy. Dynamická povaha systému sa prejavuje tým, že systém sa v závislosti od vstupov mení. Poriadok doň vnášajú vyvíjajúce sa väzby medzi jeho časťami. Mnohokrát je takýto systém riadený malým počtom jednoduchých pravidiel. Hojne používaným príkladom modelu opísaného ako CAS je kolónia mravcov. Je zrejmé, že kolóniu mravcov nikto nekontroluje. Každý jedinec funguje podľa svojej jednoduchej sady pravidiel, a predsa sú schopní spoločne vytvárať veľmi komplexné diela a vykazujú zložité vzory správania sa. Snahou manažéra teda nemá byť maximalizovať mieru kontroly, ale priniesť a dbať na dodržiavanie dohodnutých pravidiel a v prípade, že ich niekto nedodržiava, zistiť dôvod a prispôsobiť podmienky tak, aby dodržaniu pravidla nič nebránilo. Čiže manažér v agilnom projekte nie je kontrolór, ale strážca poriadku a stability v projekte.

Odhadovanie ťažko špecifikovateľného projektu

Tradičné postupy odhadovania projektov zahŕňajú analógiu na základe podobnosti s iným projektom, ktorého rozsah je známy, alebo sa postupuje na základe analýzy špecifikovaných vlastností produktu. Existujú rôzne metriky, uvažujúce počet podsystémov, odhad počtu tried, modulov, alebo metód, a tiež počet formulárov aplikácie alebo komplexnosť dátového modelu. Takéto odhady je možné použiť len na špecifický druh systémov, pretože vôbec neuvažujú zložitosť, ktorá je zákonite jedinečná pre každý neprebádaný problém.

V prípade prelomových projektov, nasadzujúcich riešenia, ktoré ešte neboli v danej oblasti použité, alebo aj pri projektoch z pre tím úplne neznámej oblasti, je proces odhadu projektu na základe analýzy špecifikovaných vlastností výsledného produktu ťažký. Jeho náročnosť navyše rastie s veľkosťou riešenej úlohy. Prirodzeným riešením je dekompozícia problému na podproblémy až pokým sa nedospeje k problému, ktorého náročnosť vieme ľahko odhadnúť.

Scrum prináša mierne odlišný prístup k odhadovaniu rozsahu. Na začiatku projektu sa určí približný termín, v ktorom by sa mala vybrať finálna verzia produktu. Proces sám o sebe predpokladá nedodržanie termínu a produkt je vyvíjaný tak, aby na konci každého šprintu bol pripravený na vydanie. Nepostupuje sa od činností k časovému rozvrhu, ale časový rozvrh (šprinty) sa napĺňajú činnosťami. Na konci každého šprintu sa na základe analýzy knihy nedokončených prác rozhodne o špecifikovaní dodatočných požiadaviek do ďalšieho šprintu. Toto nerieši problém nemožnosti odhadnúť rozsah neznámeho projektu na jeho začiatku, ale do istej miery eliminuje riziko, keďže proces zabezpečuje projekt s istou mierou funkčnosti na konci každého šprintu.

Nesprávne odhadnutý projekt, akým je každý projekt netriviálneho rozsahu, pretože dokonalý odhad zo zásady nie je možné spraviť, sa môže, a väčšinou aj začne, po čase vzďaľovať od plánu. V prípade projektu, ktorý je na začiatku ťažké úplne

špecifikovať, nastanú odchýlky od plánu ešte rýchlejšie. Navyše pri projektoch, kde sa nasadzujú neoverené technológie alebo je potrebné navrhnúť nové prístupy a metódy, často nie je možné bez dôkladnej analýzy ani približne odhadnúť rozsah. Avšak nejaký odhad je potrebné urobiť vždy. Ak je už na začiatku jasné, že je nesprávny, tak predpoklady majú nevyhnutne za následok odchýlenie sa od plánu.

Pravidelné stretnutia a vykazovanie činností má okrem iného za cieľ takéto odchýlky odhaliť. Pri zistení, že skutočnosť nezodpovedá plánu, alebo že vývoj ukazuje, že nie je možné, aby sa budúci plán naplnil, sa pristupuje k revidovaniu plánu. Okrem pravidelných, a viac menej očakávaných revízií plánu pri dosiahnutí významných plánovaných udalostí projektu, spočívajúcich často v zjemnení existujúcich plánov, je dôležité byť pripravený aj na priebežné upresňovanie plánu. Tomuto zodpovedá jeden z bodov „manifestu pre agilný vývoj softvéru“ [1]: (voľne preložené) „[vyššie si ceníme] reakciu na zmenu ako nasledovanie plánu“, ktorý pri extrémnom pojatí takmer až popiera význam vytvárania veľkých plánov.

Čím sú iterácie väčšie (a stretnutia menej časté), tým ťažšie býva odhadnúť úsilie pre ďalšiu iteráciu. Pri procese Scrum majú typické šprinty dĺžku jeden týždeň, maximálne mesiac a stretnutia k projektu, teda akési mikroiterácie, prebiehajú na začiatku každého dňa. Náplňou každého stretnutia je sumarizácia činností od posledného stretnutia, identifikácia problémov, na ktoré jednotliví členovia narazili, a určenie činností do ďalšieho stretnutia. Toto prispieva k zvýšeniu efektivity tímu, k rýchlemu vyriešeniu problémov hneď ako nastanú, a tiež ku zvýšeniu pracovnej morálky všetkých členov tímu. Je známy fakt, že ak má človek pravidelne vykazovať činnosti, tak mu svedomie nedovolí nerobiť a nevykázať nič. Ak má človek referovať o postupe každý deň, je zaručené, že každý deň sa bude práci na projekte aspoň minimálne venovať. Vo väčšine tímov však problém nepracovania na projekte netreba riešiť a každodenné stretnutia pôsobia najmä ako faktor zvyšovania pocitu súdržnosti v tíme a zjednodušovania riešenia problémov.

Schwaber v [4] sleduje ako rýchlosť, tak aj zrýchlenie v projektoch využívajúcich Scrum. Ukazuje sa, že projekty majú na začiatku nízku rýchlosť, a tiež zrýchlenie. Keď sa vybuduje dostatočná infraštruktúra a rámce a objekty majú už základnú funkcionalitu, zrýchlenie a prirodzene aj rýchlosť sa zvyšujú. V pokročilých šprintoch projektu sa pozoruje zníženie zrýchlenia, zatiaľ čo rýchlosť ostáva vysoká. Toto poznanie prináša predpoklad, že odhad rozvrhu projektu takmer zákonite prináša veľkú chybu, keďže zrýchlenie spôsobuje nelinearitu v priebehu projektu. Ako uvádza Schwaber, zatiaľ nie sú známe a je potrebné vyvinúť metriky pre odhadovanie týchto procesov.

Záver

Plánovanie tvorí dôležitú časť riadenia softvérového projektu. V tejto eseji sme analyzovali špecifiká plánovania na prípade tímu využívajúceho agilnú metodológiu vývoja softvéru Scrum. Vyzdvihli sme prínosy tejto metodológie pre riadenie životného cyklu projektu a pozitívny vplyv procesu na plánovanie. Na príklade projektu s ťažko špecifikovateľným rozsahom dotýkajúceho sa prelomových a málo

prebádaných oblastí sme ukázali vhodnosť nasadenia agilnej metodológie vývoja softvéru z pohľadu tvorby a manažmentu plánov.

Použitá literatúra

1. Beck, K. et al.: *Manifesto for Agile Software Development. http://agilemanifesto.org/ (22.10.2006)*
2. Brooks, F.P., Jr.: *No silver bullet: essence and accidents of software engineering.* IEEE Computer Society Press, Los Alamitos, 1987.
3. Highsmith, J.: *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems.* Dorset House, New York, 2000.
4. Schwaber, K.: *SCRUM Development Process.* http://jeffsutherland.com/oopsla/schwapub.pdf (22.10.2006)

Annotation

Planning and agile software development methodologies

It is without doubt that any project needs proper planning in its management process. Even if general methods of plan creation and management are known, particular method application should always reflect all specific features of the particular project. Plan creation methods seem to be scalable for wide range of project sizes, but in case of software projects, especially specific projects where the problem domain is not fully understood, or it's not possible to specify the product fully in advance, there comes the need for adaptation of the software process as well as the process of plan creation and management itself. In this essay we analyze problems with planning in the context of projects which apply agile development methodologies and we discuss special aspects of planning in projects that can be described as hard to specify, quickly developing and changing software projects. On an example of agile software development methodology "Scrum" we discuss specific aspects to be taken into account when adapting traditional plan management methods in small agile teams.