Agilné metódy vývoja softvéru

Škálovanie agilných metód

IGOR BERTA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
berta01@student.fiit.stuba.sk

Abstrakt. Agilné metódy sa čoraz častejšie objavujú ako alternatíva ku „klasickým" metódam vývoja softvéru. Dôvodov ich úspechu je niekoľko, pričom jednotlivo sú rozoberané v prvej časti tejto eseje. V tejto časti sú ďalej rozoberané základné charakteristiky agilných metód. Jednou z najznámejších agilných metód je Extrémne programovanie, ktoré je bližšie rozoberané v druhej kapitole. Ďalšia časť je venovaná škálovaniu agilných metód. To znamená použitiu agilných metód pre veľké a distribuované projekty. Sú v nej rozoberané skúsenosti jednotlivých tímov a ich odporúčania spolu s experimentmi vykonanými v tejto oblasti.

Úvod

Agilné metódy vývoja softvéru sa napriek tomu, že sa jedná o relatívne nový prístup k vývoju softvéru, tešia stále väčšej obľube, a to najmä zo strany vývojových pracovníkov. Prečo je tomu tak? Čím to je spôsobené? Sú agilné metódy naozaj všeliekom? Je možné použiť agilné metódy v hocakom projekte bez ohľadu na typ projektu, rozsah projektu, zákazníka a podobne? Práve na takéto otázky sa pokúsim poskytnúť odpoveď na nasledujúcich pár stranách.

V prvej časti eseje je uvedený stručný prehľad samotných agilných metód, princípov a postupov, na ktorých sú tieto metódy založené. Okrem samotného prehľadu sú v tejto časti vysvetlené niektoré pojmy agilných metód pomocou krátkej charakteristiky jednej z najznámejších agilných metód, a to Extrémneho programovania.

Druhá časť eseje sa venuje problému, ktorý vyšiel na povrch so samotným rozširovaním agilných metód. Tento problém bol nazvaný škálovanie agilných metód a jedná sa o problém rozsahu projektu a tým vlastne aj rozsahu tímu riešiteľov projektu, pre ktorý je nie len vhodné, ale vôbec možné použiť agilné metódy vývoja softvéru.

Agilné metódy vývoja softvéru, január 2006, s. 7-22.

Agilné metódy vývoja softvéru

Oficiálny vznik samotných agilných metód vývoja softvéru sa datuje k februáru roku 2001, kedy skupina 17-tich softvérových inžinierov, iniciátorov vzniku týchto metód, vydala takzvaný Manifest agilného vývoja softvéru [8]. V tomto manifeste je zhrnutých, okrem iného, aj 12 základných princípov agilného vývoja softvéru [8].

Predtým ako sa budem venovať samotným princípom agilných metód, by som rád bližšie analyzoval, prečo sú agilné metódy úspešné v projektoch, kde takzvaný klasický, alebo plánom riadený vývoj zlyháva. Jednou z príčin je dynamický vývoj v disciplíne softvérového inžinierstva, ktorý je spôsobený aj prudkými zmenami na poli informačných technológií. Princípy, ktoré boli v čase ranného vývoja softvérového inžinierstva správne a im boli prispôsobené aj jednotlivé metódy vývoja softvéru, sa v súčasnosti ukazujú byť, najmä pri niektorých typoch projektov, nepoužiteľné.

Jedná sa najmä o predpoklad *presného definovania* a *špecifikovania* požiadaviek zo strany zákazníka v počiatočných fázach projektu, v samotnej fáze *analýzy*. Tento predpoklad bol správny v časoch, kedy bol softvér vyvíjaný väčšinou pre presne špecifikovaného zákazníka, platformu a v niektorých prípadoch dokonca aj samotný hardvér. V súčasnosti takýto prístup však už nie je možné s úspechom aplikovať pri veľkej väčšine vyvíjaných softvérových systémov. Prečo? Odpoveď je jednoduchá. Dynamická zmena požiadaviek [7]. V dnešnej dobe sa požiadavky na vyvíjané softvérové systémy menia *dynamicky*, či už zmenou technológie, požiadaviek trhu, legislatívy a podobne. Preto každý jeden vývojový tím by mal byť schopný pružne reagovať na *zmenu* a *vznik nových požiadaviek* .

Mnoho projektov, používajúcich práve klasické metódy vývoja softvéru stroskotalo práve na tomto probléme, kedy napriek tomu, že vývojárom sa podarilo splniť všetky definované a špecifikované požiadavky presne podľa špecifikácie, systém bol už v čase uvedenie na trh zastaraný. Systém bol zastaraný, pretože požiadavky sa menili počas vývoja systému, ale tieto zmeny už nebolo možné zapracovať do systému. Alebo bolo možné, ale len za cenu zlyhania časového plánu, prekročenie rozpočtu a podobne.

Aj klasické prístupy sa snažia o zohľadnenie možných zmien požiadaviek, prípadne vzniku nových požiadaviek. Avšak snažia sa ich zohľadniť *naplánovaním* tejto zmeny či vzniku [7]. To je však možné len s určitou mierou úspešnosti, ktorá závisí od viacerých faktorov. Jedným z nich sú aj skúsenosti pracovníka ktorý vytvára plán, ale v konečnom dôsledku aj od náhody. V jednom medznom prípade, kedy by neprišlo k žiadnej zmene požiadaviek, by prišlo k plytvaniu časom a zdrojmi. Naopak v druhom medznom prípade, ktorý je nanešťastie častejší, kedy by prišlo k takej veľkej zmene požiadaviek, ktorá nebola naplánovaná, by prišlo k prekročeniu rozpočtu a časového plánu.

Tento problém sa snažia agilné metódy riešiť tým, že *vítajú zmeny požiadaviek* a rátajú s nimi už v samotnej svojej podstate. Samotná zmena požiadaviek v každej fáze vývoja softvéru predstavuje základný princíp agilných metód. Ja v tejto eseji nebudem uvádzať všetkých 12 princípov, ako boli sú uvedené v Manifeste agilných metód, pretože všetky vychádzajú zo štyroch základných oporných pilierov agilných metód, ktorými sú [8]:

1. kladenie dôrazu na ľudí, ako na najdôležitejšiu časť vývojového reťazca
2. kladenie dôrazu na funkčný softvér
3. kladenie dôrazu na úzku spoluprácu so zákazníkom
4. schopnosť prispôsobenia sa novým požiadavkám

Z týchto 4 základných pilierov agilných metód sa postupom času, ako sa jednotlivé agilné metódy vyvíjali, vytvorili ďalšie charakteristiky spoločné pre jednotlivé agilné metódy. Medzi tieto charakteristiky patria:

- inkrementálny a iteratívny spôsob vývoja
- vysoký dôraz na komunikáciu medzi jednotlivými členmi tímu
- vysoká miera spolupráce medzi jednotlivými členmi tímu
- zapracovanie testov do skorých fáz procesu vývoja softvéru
- silné zapojenie zákazníka do procesu vývoja softvéru
- zníženie objemu dokumentácie na najnižšiu možnú mieru
- časté vydávanie funkčných verzií
- zníženie viditeľnosti
- redukcia nadbytočnej práce

V tomto zozname je uvedených len zopár spoločných znakov agilných metód. Samotné praktické využitie týchto znakov sa môže od jednej metódy k druhej líšiť. Napríklad pri Vývoji riadenom pomocou čŕt je presne stanovené časový harmonogram vydávania nových funkčných verzií systému, a naproti tomu pri Extrémnom programovaní nič podobné nie je.

Ak by sme sa chceli bližšie pozrieť na jednotlivé charakteristiky agilných metód, nájdeme tu najmä dva typy.

Jedny sa snažia o splnenie motta „Softvér vytvárajú ľudia pre ľudí“ [7]. To znamená, že sa kladie dôraz viac na *ľudí*, ktorí sú zapojení do samotného procesu vývoja softvéru ako na samotný *proces* [7]. V agilných metódach sa nikdy neuprednostňuje dodržanie časového plánu, ktorý vlastne ani neexistuje, aspoň nie vo forme a hĺbke známej z klasických metód, pred samotnými vývojármi. Agilné metódy sa snažia o vytvorenie tímu, ktorý bude spolupracovať dlhí čas, preto je kladený dôraz na ľudí. Je snaha o vytvorenie čo *najužšieho vzťahu* medzi jednotlivými vývojovými pracovníkmi, ako napríklad ich umiestnením v jednej miestnosti, ktoré patrí k základným odporúčaniam skoro všetkých agilných metód. Ďalším prostriedkom sú časté *stretnutia*, napríklad v prípade metódy Skrumáž, sú tieto stretnutie každé ráno [7]. Tieto stretnutia predstavujú vo všetkých agilných metódach najdôležitejšiu formu *komunikácie*. Je dokázané, že najefektívnejším spôsobom komunikácie je komunikácia *tvárou v tvár*, to znamená komunikácia, kedy sú ľudia fyzicky v jednej miestnosti a nekomunikujú pomocou telefónu, Internetu a podobne [3,6,7,8].

Druhý typ sa snaží o to, čo bolo spomínané už viackrát a to umožnenie, dokonca o *uvítanie zmien* v každej fáze vývoja softvéru. Na to, aby toto bolo umožnené, je potrebná veľmi *úzka spolupráca so zákazníkom*. Úzka spolupráca v prípade niektorých agilných metód, ako je napríklad Extrémne programovanie, predstavuje začlenenie predstaviteľa zákazníka do tímu, ako regulárneho člena, ktorý sa zúčastňuje všetkých tímových stretnutí a podobne [4]. Ďalej je to *časté vydávanie funkčných verzií*, na ktorých zákazník otestuje samotné svoje požiadavky, zistí, nakoľko sú ešte aktuálne, aké funkcionality je potrebné ešte implementovať, čím vlastne vznikajú nové požiadavky. Týmto je podmienený aj samotný spôsob vývoja, ktorý je *inkrementálny* a *iteratívny*. Ďalšou charakteristikou, ktorá sa dá zaradiť do tejto kategórie, je *Redukcia nadbytočnej práce*, kedy vývojár nevyvíja časti, ktoré nie sú takpovediac *na rade*. To znamená, že ak vývojár vytvorí triedu číslo, tak nebude implementovať operáciu sčítanie, pretože jednoducho ešte nie je *na rade*, aj keď si je istý, že ju bude robiť v ďalšej iterácii. Nebude ju implementovať, pretože nie je zadaná zákazníkom a schválená ostatnými členmi tímu. A najmä preto, lebo zákazník ešte stále môže zmeniť svoju požiadavku na operáciu plus.

Samotné agilné metódy nepredstavujú presné *smernice*, ako sa má alebo ako by mal vyvíjať softvér, skôr obsahujú sadu *odporúčaní* [7], ktorú je možné prispôsobiť, je možné použiť len vyhovujúce časti a podobne prispôsobiť potrebám konkrétneho projektu.

V tejto časti som sa snažil aspoň nahrubo opísať základné princípy a spoločné črty agilných metód, ktoré sa budem teraz snažiť demonštrovať na krátkom opise jednej z agilných metód, konkrétne na Extrémnom programovaní.

Extrémne programovanie

Extrémne programovanie som si ako reprezentanta agilných metód vybral z viacerých príčin. Prvou z nich je už samotný názov tejto metódy, ktorý pôsobí, ako dvojsečná zbraň. Z jednej strany táto metóda priťahuje záujem, pretože je predsa *extrémna*, ale zo strany druhej odrádza od použitia, skutočného použitia pri vývoji softvéru, veď ktorá organizácia, by chcela mať svoj softvér vyvíjaný *extrémnymi* metódami?

Obidva prípady sú však tak trošku zavádzajúce. Pre samotných vývojárov, ale aj zákazníkov, ktorý sa bližšie oboznámia s touto metódou vývoja softvéru je však jasné, že slovíčko *extrémne* v názve metódy sa vzťahuje skôr na prístupy, ktoré zavádza. Jedným z najvýraznejších reprezentantov tohto prístupu je Programovanie v pároch.

Druhý dôvod, prečo som si vybral práve Extrémne programovanie, sa týka druhej časti eseje, kde väčšina zdokumentovaných pokusov v oblasti škálovania agilných metód bola vykonaná práve s pomocou Extrémneho programovania.

Čo je základom Extrémneho programovania

Samotný autor metodológie Extrémneho programovania Kent Back stanovil štyri základné pojmy Extrémneho programovania [4]:

1. jednoduchosť
2. komunikácia

3. spätná väzba
4. odvaha

Jednoduchosť spočíva v pravidle robiť všetko tak jednoducho ako sa len dá. Urobiť len to, čo zákazník naozaj *chce* a nerobiť to čo by zákazník *mohol eventuálne chcieť* v ďalšej iterácii, aj keď vieme, že to bude chcieť. Napríklad, ak zákazník chce už spomínanú triedu číslo a operáciu sčítania, tak toto pravidlo nám káže implementovať len operáciu sčítania a nie operácie násobenia alebo delenia, aj keď sme si istý, že v ďalšej iterácii ich bude zákazník chcieť implementovať.

Komunikácia patrí pri tejto metodike k silným podmienkam úspešnosti, preto aj samotná metodika predpokladá isté podmienky, ako je umiestnenie tímu v jednej miestnosti a podobne.

Spätná väzba je v tejto metodike zabezpečená silnou participáciou zákazníka na vývoji, odporúča sa dokonca, aby sa reprezentant zákazníka stal členom tímu. Ďalej je zabezpečená tým, že sa v podstate jedná o *vývoj riadený testami*, ako uvidíme v ďalšom. A v neposlednom rade preto, že sú stanovené *krátke vývojové iterácie*, pričom na konci každej vzniká nejaká *doručiteľná* a *otestovateľná* funkcionalita.

A posledný z pojmov, *odvaha,* naráža na odvahu jednotlivých členov tímu. Odvahu pre zmenu, pretože samotný vývoj sa začína od jednoduchého návrhu, ktorý sa postupne zlepšuje *refaktorizáciou*.

Základné charakteristiky Extrémneho programovania [4]

- *Tím ako celok* – všetci ľudia zúčastnení na projekte sú súčasťou jedného tímu, pričom jedným z členov tímu je aj zástupca zákazníka. Takto je zabezpečený neustály kontakt so zákazníkom a samotný členovia tímu nemusia na jednej strane vytvárať a na strane druhej študovať podrobné špecifikácie.
- *Plánovanie hrou* – plánujú sa len najbližšie udalosti, plány sú jednoduché. Neplánujú sa jednotlivé činnosti, stanovujú sa len najbližšie ciele a približný čas ich naplnenia.
- *Krátke iterácie* – krátke iterácie zabezpečujú lepšiu viditeľnosť zo strany zákazníka, umožňujú včasné testovanie, dávajú zákazníkovi do rúk ďalšie a ďalšie výsledky, ktoré môže prehodnotiť, nakoľko spĺňajú jeho požiadavky a nakoľko nie.
- *Zákaznícke testy* – na základe požiadavky od zákazníka, jej špecifikácie, sú stanovené testy, ktoré musí implementácia úspešne absolvovať, na to aby mohla byť akceptovaná zákazníkom.
- *Jednoduchý návrh* – táto charakteristika je zhrnutá v známej skratke extrémneho programovania YAGNI – „you ain't gonna need it", to nebudeš potrebovať a bola vysvetlená už v úvode tejto kapitoly
- *Programovanie v pároch* – táto charakteristika je pravdepodobne jedna z tých, ktoré priniesli Extrémnemu programovaniu jeho meno. Programovanie prebieha v pároch, čo sa ukázalo byť ako, aj keď sa to môže zdať

neuveriteľné, efektívnejšie ako programovanie dvoch samostatných programátorov. A okrem iného podporuje aj ďalšiu myšlienku Extrémneho programovania a tou je spoločné vlastníctvo kódu.

- *Vývoj riadený testami* – pomocou Zákazníckych testov je zabezpečená neustála spätná väzba medzi zákazníkom a vývojovým tímom. Ako bolo spomenuté aj vyššie, najskôr sa stanoví a implementuje test a až následne sa implementuje samotná požiadavka, funkcionalita. Dôležité je spomenúť, že tento test musia úspešne absolvovať aj všetky ďalšie vydania, to znamená že finálne vydanie určite absolvuje všetky zákaznícke testy.
- *Zlepšovanie návrhu* – tento pojem bol už vysvetľovaný v prvej časti tejto kapitoly, jednoducho ak je potrebné urobiť zmenu v návrhu, tak sa tento zmení – refaktoring, teda zmena kódu, jeho zhutnenie, zovšeobecnenie,...
- *Neustála integrácia* – keďže samotný systém je vyvíjaný inkrementálne a iteratívne, je potrebná neustála integrácia nových *inkrementov*, to znamená výsledkov jednej iterácie
- *Spoločné vlastnenie kódu* – táto charakteristika je taktiež veľmi dôležitá, súvisí aj s Programovaním v pároch. Všetci členovia tímu vlastnia všetok kód a môžu robiť úpravy v hocktorej časti kódu, ak v nej nájdu chybu a podobne. Tento prístup pomáha riešiť aj problémy s odchodom člena tímu, prípadne s jeho ochorením. Výsledkom je aj vyššia kvalita kódu, keďže na ňu nedohliada len autor, ale celý tím.
- *Dohodnutý štandard písania kódu* – hneď dve charakteristiky sú podmienené vznikom štandardu pre písanie kódu a to Programovanie v pároch a Spoločné vlastnenie kódu. Táto charakteristika znamená, že sa členovia tímu, aj keď len neformálne dohodnú na štandarde písania kódu, poznámok, pomenovávania tried, premenných a podobne.
- *Obraz systému* – počas práce na projekte si každý člen tímu vytvorí vlastný obraz, dalo by sa povedať model, systému. Tým, že celý tím spolupracuje v čo možno najširšej miere, komunikuje denne o svojich problémoch, nápadoch a ideách, mali by sa *Obrazy systému* jednotlivých členov tímu čo najviac podobať.
- *Udržateľné tempo* – z praxe je známe, že každý vývojár je schopný podávať určitý výkon, sú známe odchýlky až 30%. Každý vývojár najlepšie pozná svoje tempo a vie zhodnotiť, kedy ešte má význam pracovať a kedy je už lepšie nepokračovať a ísť si radšej napríklad oddýchnuť. To znamená, pracuje sa vtedy, keď to má ešte význam a nie vtedy, keď to prikazuje zmluva.

Aj z tohto krátkeho opisu Extrémneho programovania je jasné, že sa zaraďuje medzi agilné metódy vývoja softvéru, pričom niektoré charakteristiky agilných metód dovádza až do *extrému*, ako napríklad *spoluprácu* a *komunikáciu* vo forme Kolektívneho vlastníctva kódu a Programovania v pároch alebo *jednoduchosť* vo forme známeho akronymu YAGNI.

Aké projekty sú vhodnými kandidátmi na použitie agilných metód? Je veľmi ťažké stanoviť presnú hranicu a presne povedať, pri ktorom projekte je vhodné použitie agilných metód a pri ktorom nie. Vo všeobecnosti sa dá však povedať, že to je najmä v tých projektoch, ktoré spĺňajú požiadavky, na základe ktorých prišlo k vzniku agilných metód. To znamená v prípade *nejasných* a často sa *meniacich požiadaviek*, v prípade, kedy je *zákazník* ochotný participovať na vývoji a v krajnom prípade sa dokonca stať členom vývojového tímu, a podobne. To znamená, že nie sú stanovené pevné hranice, kedy použiť *agilné* a kedy *klasické* metódy vývoja softvéru. A už vôbec nie sú dané pravidlá, kedy použiť ktorú agilnú metódu. Tieto rozhodnutia záležia na skúsenostiach a schopnostiach jednotlivých členov tímu v prípade agilných metód alebo človeka za to zodpovedného, v prípade klasických metód.

Akých ľudí potrebujeme do tímu vyvíjajúceho agilnými metódami? Sú známe rôzne testy, ktoré nám povedia, aký je kto typ človeka a ešte jednoduchšie je zistiť, ako kto vie kvalitne programovať. Čo je však problém agilných metód a špeciálne Extrémneho programovania, že sa nehľadajú nadpriemerní programátori (bez ďalších požadovaných vlastností). Hľadajú sa ľudia, ktorí budú schopní:

- programovať v pároch
- zdieľať kód – Kolektívne vlastníctvo kódu

Tieto vlastnosti sa ťažko odhaľujú pomocou testov. Programovanie v pároch je vec, ktorá je za normálnych okolností pre vývojára dosť ťažko predstaviteľná. A ak aj je vývojár schopný programovať v páre, určite nie je každý schopný programovať v páre s každým. Takisto kolektívne vlastníctvo kódu. Je známych veľa prípadov z praxe, keď vývojár jednoducho potreboval *vlastniť* určitú časť kódu. Pre takýto typ vývojárov sú vhodnejšie iné agilné metódy, ako napríklad Vývoj riadený pomocou čŕt.

Ako ďalej vidieť, agilné metódy a Extrémne programovanie ako ich reprezentant sú určené pre malé tímy. Čo však znamená *malé tímy*? Tento pojem sa odlišuje od metódy k metóde. V hrubom zovšeobecnení by sa dalo povedať, toľko koľko sa zmestí do jednej miestnosti. To samozrejme nie je pravda, pretože miestnosťou je aj letecký hangár. A preto, že samotné sedenie v jednej miestnosti nič nerieši. V agilných metódach sa obvykle jedná sa o typ komunikácie každého s každým, čo by bolo pri väčšom počte ľudí neúnosné a réžia by rástla rýchlejšie ako produktivita tímu. Pri extrémnom programovaní sa hovorí ako o optimálnom počte členov tímu o čísle 12, ale sú samozrejme zdokumentované a úspešné projekty až do 25 členov [4,7,8].

Z uvedeného vyplýva, že projekty ako napríklad Tímový projekt, ktorý momentálne absolvujeme na fakulte, sú ideálnymi kandidátmi na použitie agilných metód. Pri bližšom porovnaní narazíme na viacero spoločných charakteristík, ktoré naznačujú, že použitie agilných metód v Tímovom projekte by mohlo priniesť lepší výsledok, ako použitie klasických metód vývoja softvéru:

- *malý tím vývojárov* - typicky sa v tíme nachádza 6 členov
- *úzka spolupráca so zákazníkom* - ak berieme do úvahy vedúceho projektu ako predstaviteľa zákazníka, tak sa spĺňa aj podmienka úzkeho kontaktu so zákazníkom, kedy vedúceho tímu možno považovať v niektorých prípadoch aj za člena tímu

- *nejasne špecifikované požiadavky, prípadne vznik nových* - požiadavky od vedúceho tímu ako predstaviteľa zákazníka prichádzajú obvykle počas oboch semestrov.
- *časté stretnutia tímu* – tím sa stretáva minimálne raz týždenne, čo by nebolo dostačujúce pravdepodobne ani pre jednu z agilných metód, ale typicky sa tím stretáva takmer denne v škole, kde sa preberajú problémy a nápady

Škálovanie agilných metód

Po tom, čo bolo uvedené v závere predchádzajúcej časti, by sa mohlo zdať zvláštne vôbec rozprávať o škálovaní agilných metód. Navyše ak sa sám zakladateľ agilných metód a jeden z autorov Manifestu agilného vývoja Martin Fowler, vyjadril [6]: „Škálovanie agilných metód a extrémneho programovania je pravdepodobne posledná vec, ktorú by som chcel robiť“. Napriek tomu sa väčšina účastníkov konferencie, na ktorej toto uviedol, vyjadrila, že vývojári *budú* škálovať agilné metódy, aj napriek silným varovaniam [6].

Škálovanie agilných metód neznamená len zmenu v rozsahu a *veľkosti projektu*, a teda aj tímu. Znamená škálovanie aj v zmysle *geografickom*, to znamená *distribuovanie* členov tímu. Nie všetci členovia tímu sú umiestnení v jednej miestnosti, nemusia byť v jednej budove, dokonca ani v jednom štáte.

Škálovanie agilných metód je pomerne nová iniciatíva, ktorá logicky prišla po uznaní kvalít agilných metód vývoja softvéru pre *malé tímy* a malé projekty. Samotné uznanie kvality a funkčnosti agilných metód nebolo urobené matematicky, ani inak podobne. Bolo urobené len empirickým skúmaním, to znamená porovnaním *úspešnosti projektov*, ktoré používali *klasické metódy* vývoja a projektov, ktoré používali *agilné metódy*. Tu narážame na najväčší problém škálovania agilných metód, ktorým je nedostatok empirických výsledkov.

Príčin je hneď niekoľko. Jedná sa o pomerne novú iniciatívu, a nie všetci zákazníci veľkých projektov sú ochotní pristúpiť na vývoj pomocou agilných metód, nie všetky tímy používajúce agilné metódy zdokumentujú svoj postup a podobne. Preto musíme naše zistenia opierať skôr o rôzne experimenty s Distribuovaním programovaním v pároch a podobne.

Od začiatku myšlienky škálovania agilných metód sa objavilo viacero myšlienkových prúdov. Jeden z nich staval na agilných metódach ako takých, bez zmeny alebo kombinácie s inými, klasickými metódami. Druhý z nich sa pokúšal práve o vytvorenie akéhosi *hybridného* prístupu, kedy by sa skombinovalo to *dobré* z agilných metód a to, čo robí klasické metódy fungujúce pre *veľké a distribuované* projekty, z klasických metód. Posledný z prístupov sa snaží o vytvorenie úplne *nových* metód, ktoré by vznikli na základe *agilných* metód, ale boli by orientované pre *veľké a distribuované* projekty.

V súčasnosti sa ukazuje byť najlepšou druhá alternatíva. Prvá alternatíva, ktorá striktne stavia na dodržiavaní agilných metód, naráža na neustále problémy, najmä s charakteristikami agilných metód ako je napríklad komunikácia, jej kvalita a

podobne. Tretia alternatíva je tiež úspešná, ale načo vyvíjať nové metódy, ak máme metódy, ktoré sú už otestované, funkčné, a vývojári sú na ne pripravení. Okrem toho tretia alternatíva bola použitá a zdokumentovaná len v jednom konkrétnom prípade a to vo firme Nokia, kde boli využité princípy Adaptívneho vývoja softvéru a vytvorenie takzvaných Komunít podľa skúseností (Community of Practice) [5].

Distribuovaný vývoj softvéru sa v posledných rokoch stáva stále bežnejším najmä vo veľkých spoločnostiach [3]. Jedným z dôvodov je snaha veľkých spoločností ušetriť peniaze na veľkých projektoch najímaním rovnako zdatných programátorov z krajín, kde je lacnejšia pracovná sila, pričom ale časť tímu je stále vytvorená z kmeňových pracovníkov firmy na centrále. Ďalším z dôvodov je napríklad dištančné vzdelávanie alebo on-line vzdelávanie, kedy je potrebná kooperácia jednotlivých študentov pri práci na projektoch, pričom ale oni sami sú umiestnený v geograficky rozdielnych oblastiach.

Charakteristiky agilných metód vývoja pre veľké projekty

Kľúčovými charakteristikami, na ktoré je potrebné sa zamerať pri škálovaní agilných metód sú nasledujúce štyri [3]:

1. *komunikácia* – komunikácia musí prebiehať nielen v rámci tímov, ale aj medzi jednotlivými tímami pracujúcimi na projekte. Podľa viacerých odborníkov z oblasti manažmentu je nevyhnutné vytvoriť nielen jeden komunikačný kanál a to z dôvodu, že by prišlo skoro k jeho preťaženiu. Preto je potrebné vytvoriť viacero komunikačných kanálov a používať ten, ktorý sa na plánovanú činnosť najlepšie hodí.

2. *zapojenie zákazníka do procesu* – problémom pri veľkých projektoch býva aj zapojenie zákazníka do procesu vývoja. Je to najmä z dôvodu, že v prípade veľkých projektov obvykle nebýva zákazník jeden, ale je to celá skupina zákazníkov, v niektorých prípadoch dokonca konkurentov. V takom prípade je nemožné zvoliť jedného hovorcu zákazníka, pretože sami zákazníci môžu mať protichodné požiadavky. Problémom môže byť aj vyvíjanie generického softvéru, kedy je zákazníkom celý segment trhu a zákazník je neviditeľný.

3. *plánovanie* – najmä v prípade veľkých projektov je neobvyklé plánovať projekt oproti výsledkom. Typicky je plán zameraný na míľniky, ktoré nie sú zoradené podľa priority alebo biznis hodnoty pre zákazníka. Preto je často ťažké, prejsť od tohto typu plánovania k plánovaniu používanému agilnými metódami, ako je napríklad *plánovanie hrou*. Z tohto dôvodu je občas potrebná kombinácia oboch prístupov pre vytvorenie kompromisného plánu projektu.

4. *integrácia* – väčšina agilných metód obsahuje ako jednu zo svojich charakteristík *neustálu integráciu*. Tento prístup je však často v prípade veľkých tímov alebo veľkého počtu tímov veľmi ťažké.

V tomto zozname sú uvedené len niektoré charakteristiky agilných metód, ktoré sú zvyčajne problémové v prípade *škálovania* agilných metód pre *veľké* projekty.

Samozrejme zoznam sa môže líšiť projekt od projektu, v závislosti od typu projektu, distribuovanosti alebo lokálnosti tímov, skúseností jednotlivých vývojárov s agilnými metódami a podobne.

Riešenie týchto problémov je taktiež individuálne pre každý projekt. Napriek tomu existujú určité *odporúčania*, pre niektoré z agilných metód vývoja. Cieľom týchto odporúčaní je pomoc vyrovnať sa s hore uvedenými a aj mnohými ďalšími *problémami*, s ktorými sa tím môže stretnúť počas vývoja veľkého projektu použitím tejto metódy. Príkladom je Extrémne programovanie, pre ktoré vznikol aj pojem Distribuované extrémne programovanie a jednotlivé odporúčania sa uvedené v kapitole Extrémne programovanie a distribuovaný vývoj.

Charakteristiky distribuovaného agilného vývoja [3]

1. *Zníženie objemu a kvality komunikácie* – prieskumy v tejto oblasti ukázali, že so vzdialenosťou sa znižujú obidva tieto ukazovatele. Je to spôsobené aj tým, že je zložitejšie viesť neformálnu diskusiu, ktorá je neoddeliteľnou časťou každého softvérového projektu.
2. *Oneskorenia v rozhodnutiach a distribúcii informácií* – distribuované projekty môžu mať členov, ktorí sú z rôznych časových pásiem, tým pádom pracujú v rozličných časoch, je zložitejšie usporadúvať on-line stretnutia a podobne.
3. Väčšina distribuovaných projektov je aj rozsahovo a počtom členov tímu veľkých
4. *Kultúrne rozdiely* – pretože softvér je vyvíjaný ľuďmi, je vývoj softvéru silne viazaný na ľudskú povahu, ktorá je silne ovplyvnená aj kultúrou. Preto sa kultúrne rozdiely stali jedným z faktorov ovplyvňujúcich vývoj softvéru.

Ako bolo spomenuté v prvej časti tejto eseje, Manifest agilných metód obsahuje 12 základných princípov. Z týchto 12 sú najmä nasledujúce tri ovplyvnené distribuovaným vývojom [3]:

1. *Individuality a interakcie* – nedostatok komunikácie opísaný vyššie robí ťažším najmä výber správnych individualít, ktoré by boli schopné udržovať efektívnu komunikáciu medzi sebou
2. *Spolupráca so zákazníkom* – blízke vzťahy so zákazníkom sú taktiež ovplyvnené distribuovanosťou projektu, čo vyúsťuje opäť do menšieho objemu a efektivity komunikácie
3. *Odpoveď na zmenu* – pretože samotné distribuované prostredie prináša extra zdržania, predlžuje sa čas potrebný na zmenu

Extrémne programovanie a distribuovaný vývoj

V tejto časti sa pokúsim o modifikáciu metódy Extrémne programovanie tak, aby zodpovedal podmienkam distribuovaného vývoju. Týmto vzniká takzvané

Distribuované extrémne programovanie. Charakteristikami Extrémneho programovania, ktoré sú ovplyvnené distribuovanosťou projektu sú [3]:

- *plánovanie hrou* – s pomocou videokonferencií môžu byť všetci členovia tímu a zákazníci zapojený do procesu vývoja
- *neustála integrácia* – jednotlivý členovia tímu zdieľajú časti kódu nad ktorými vytvárajú a spúšťajú testy
- *programovanie v pároch* – Vzdialené programovanie v pároch, kedy sú jednotlivý členovia vývojového páru umiestený v rozdielnych lokalitách, je riešené zdieľaním pracovnej plochy vývojového prostredia
- *zapojenie zákazníka* – z dôvodu čo najužšej spolupráce so zákazníkom sa usporadúvajú videokonferencie so zákazníkom a členmi vývojového tímu.

Na to, aby sme mohli vyvíjať metódou Distribuovaného extrémneho programovanie potrebujeme minimálne tieto prostriedky [3]:

- členovia tímu musia byť schopní hovoriť jedným jazykom
- potrebujeme prostriedok na výmenu informácií, stanovovanie rozvrhov, to znamená nejaký komunikačný nástroj
- prostriedok pre manažment kódu a kolektívne vlastníctvo kódu
- nástroj umožňujúci programovanie v pároch, napríklad nástroj pre on-line zdieľanie pracovnej plochy
- a v neposlednej miere nástroje pre on-line komunikáciu, napríklad audio a video konferencie

Okrem problémov s distribuovanosťou projektu, je potrebné sa zamerať aj na veľkosť projektu, to znamená na rozsah projektu a veľkosť tímu. Najdôležitejšími charakteristikami ovplyvnenými rozsahom projektu sú [3]:

- ústna dokumentácia
- kolektívne vlastníctvo

Škálovanie Extrémneho programovania naráža najmä na problém *ústnej dokumentácie*, kedy väčšina dokumentácie je vypovedaná len ústne a uložená v hlavách jednotlivých vývojárov. Z tohto dôvodu nie je potrebné vytvárať zložitú dokumentáciu, ktorá by nebola dodaná zákazníkovi a aj tak by slúžila len pre vnútorné potreby tímu. Problémom sa však stáva, ak pre veľkosť tímu, nie je možné umiestniť tím v jednej miestnosti, podobne ako v prípade distribuovaného vývoja.

Tento problém je možné riešiť opačným prístupom, kedy sa upustí od skratky YAGNI a vytvorí sa úloha *správcu dokumentácie*, pričom táto úloha sa bude považovať jednoducho za ďalšiu z požiadaviek, aj keď nie od zákazníka. Tento správca dokumentácie je zodpovedný za *vytvorenie dokumentácie*, pričom stále platí zásada urobenia len toho, čo je potrebné. Táto dokumentácia je v presne takom *rozsahu* a na takej úrovni *abstrakcie*, aby pomohla všetkým členom tímu pochopiť ciele

a požiadavky stanovené v projekte a umožňovala vytvorenie ústnej dokumentácie v rámci tímu.

Distribuovaná komunikácia a Extrémne programovanie [1]

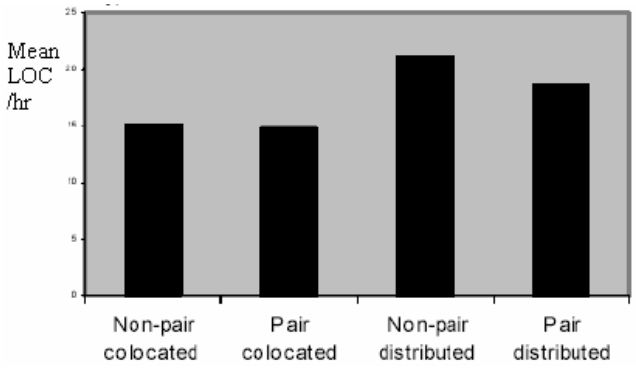
Ako bolo už niekoľko krát spomenuté, komunikácia má kľúčovú úlohu vo všetkých agilných metódach. Rovnako bolo spomenuté, že Extrémne programovanie a konkrétne jeho metóda Programovania v pároch predstavuje jednu z *komunikačne najnáročnejších* aktivít v rámci agilných metód. Dôvodom je úzka spolupráca dvoch ľudí, ktorí pracujú na rovnakých častiach projektu, dokonca kódu. Avšak výsledkom ich snaženia je len jedna implementácia tejto časti, na ktorej sa musia dohodnúť a spoločne ju naprogramovať. Preto sa zovšeobecnene dá povedať, že ak funguje Programovanie v pároch aj distribuovane, tak fungujú aj ostatné praktiky používané v agilných metódach.

Z tohto dôvodu bol vykonaný zaujímavý experiment. Tohto experimentu sa zúčastnilo 134 študentov, ktorí boli rozdelení do štyroch tímov :

1. 9 skupín, ktoré neboli distribuované a nepoužívali programovanie v pároch
2. 16 skupín, ktoré neboli distribuované a používali párové programovanie
3. 8 skupín, ktoré boli distribuované a nepoužívali párové programovanie
4. 5 skupín, ktoré boli distribuované a používali párové programovanie

Jednotliví členovia tímov sa vyberali sami spomedzi študentov a cieľom bolo naprogramovať jednoduchá hru v Jave.

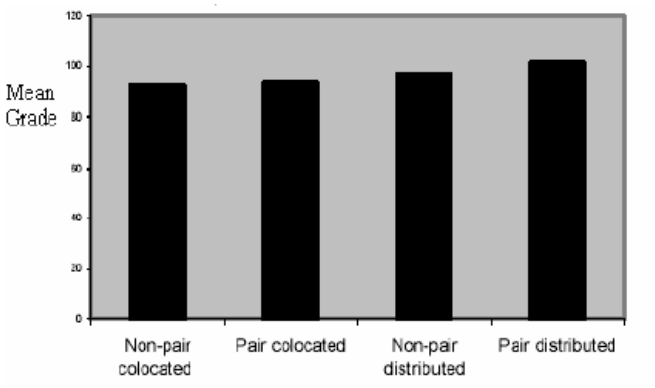
Na meranie produktivity bol použitý parameter Počet riadkov kódu za hodinu a pre meranie kvality softvéru bol použitý parameter získanej známky za tento projekt.



Obr. 1. Porovnanie produktivity jednotlivých skupín

Ako je možné vidieť na obrázku **Obr. 1**, distribuované páry boli produktívnejšie ako nedistribuované, pričom ale distribuovaný jedinci, to znamená tí, ktorí nepoužívali párové programovanie, boli produktívnejší ako tí, ktorí programovali v pároch. Avšak

autori tohto experimentu upozorňujú, že dáta nie sú štatisticky signifikantné, z dôvodu veľkého rozptylu.



Obr. 2. Porovnanie kvality softvéru vytvoreného jednotlivými skupinami

Ako vidieť z obrázku **Obr. 2**, distribuované skupiny používajúce programovanie v pároch získali najlepšie hodnotenie. Opäť autori upozorňujú na malé rozdiely, ktoré neboli dostatočné na to, aby mohli byť výsledky štatisticky signifikantné.

Napriek použitiu zjednodušených kritérií hodnotenia kvality a produktivity tímov a štatistickej nejednoznačnosti výsledkov experimentu sa dá na základe výsledkov experimentu povedať, že distribuované programovanie v pároch nie je menej produktívne alebo produkujúce horší softvér, ako programovanie v pároch na jednom mieste alebo programovanie individuálne.

Rozsahovo a veľkosťou tímu veľké projekty

Doteraz diskutované metódy boli síce určené pre distribuované projekty, ale neriešili sa v nich problémy vznikajúce pri rozsahovo veľkých projektoch. Napriek tomu, že v súčasnosti je ešte stále známych málo výsledkov projektov vyvíjajúcich na tomto princípe, z výsledkov jedného z nich, od firmy ThoughtWorks je možné vypozorovať nasledovné [3]:

- neustála integrácia ako prostriedok zamedzenia problémov s integráciou medzi lokálnymi a vzdialenými tímami
- je zavedená úloha analytika, ktorý predstavuje akýsi medzistupeň medzi zákazníkom a vývojármi v čase neprítomnosti zákazníka
- sú zavedené úlohy lokálnych analytikov, umožňujúcich lepšie pochopenie aplikačnej logiky
- vývojári vo vzdialených tímoch píšu testy, aby im bolo umožnené lepšie pochopenie požiadaviek

- zákazníci testujú jednotlivé vydania softvéru, z dôvodu otestovania nových funkcionalít a poskytnutia spätnej väzby
- uskutočňujú sa pravidelné stretnutia, kde sa preberá aktuálny stav projektu
- je nevyhnutné vytváranie väčšieho množstva dokumentácie ako kompenzácia zníženého objemu a kvality komunikácie

Z vyjadrení jednotlivých vývojárov a firmy ThoughtWorks, je možné usúdiť, že s úspechom použili agilné metódy aj v projekte s veľkým rozsahom, s dodržaním hore uvedených prispôsobení. Ďalej možno usúdiť, že distribuované agilné metódy nie sú síce natoľko efektívne ako lokálne agilné metódy, ale sú stále efektívnejšie ako klasické metódy, či už lokálne alebo distribuované.

Samozrejme, že nie je možné na základe jednej úspešnej aplikácie distribuovaných agilných metód, vytvoriť pravidlá pre úspešné aplikovanie agilných metód vo veľkých geograficky rozľahlých projektoch. Na základe pozorovaní viacerých distribuovaných projektov vznikli nasledujúce *odporúčania* [9]:

- počiatočné, takzvané „kick-off" stretnutie by malo byť uskutočnené na jednom mieste a nie on-line, z dôvodu lepšieho spoznania jednotlivých členov tímu
- je nevyhnutné zabezpečenie konferenčného serveru zabezpečujúceho komunikáciu n-n, z dôvodu on-line stretnutí tímu
- wiki[1] je vhodnou alternatívou k použitiu klasických indexových kartičiek
- pri distribuovanom programovaní v pároch je nevyhnutná vôľa jednotlivých vývojárov komunikovať
- každodenné stretnutia sú nevyhnutnou súčasťou vývoja, podobne ako pri metóde Skrumáž

Ako vidieť napriek tomu, že neexistuje dostatočné množstvo empirických dát na vyvodenie jednoznačného záveru, je možné povedať, že agilné metódy môžu byť *úspešné* aj v distribuovaných projektoch. Samozrejme, ako som spomenul už na začiatku tejto kapitoly, nie je možné snažiť sa o presadzovanie *čistých* agilných metód, pretože tie sú presne určené, čo bolo uvedené aj v samotnom Manifeste agilného vývoja, pre *malé tímy* vývojárov, umiestnené na *jednom mieste*. Preto je potrebné využitie ďalších prostriedkov, z ktorých niektoré sú známe z klasických prístupov. Príkladom môže byť kompenzácia nedostatku a zníženej kvality komunikácie pomocou zvýšeného objemu dokumentácie.

Rovnako je možné vyvodiť z uvedeného experimentu, že Distribuované programovanie v pároch *neprináša* menšiu produktivitu alebo menej kvalitný kód, ako lokálne programovanie v pároch. Samozrejme tento predpoklad nemusí platiť *všeobecne*, a záleží na type projektu, na používaných prostriedkoch pre distribuované programovanie v pároch a samotnú komunikáciu a v neposlednej rade aj na samotných

[1] wiki – je špeciálny typ webovej stránky, ktorá dovoľuje používateľom jednoducho pridávať a editovať svoj obsah

vývojároch, na tom ako sú schopní vysporiadať sa s týmto novým prístupom, ktorý opäť prináša niečo nové, ako je len samotné programovanie v pároch.

Ako som spomenul, agilné metódy sú z jedným z kandidátov na to, akým spôsobom pracovať v rámci Tímového projektu. Ale prečo sa potom zaoberať ich škálovaním? Veď predsa tím na tímovom projekte je vždy maximálne 8 ľudí. Odpoveď je jednoduchá a je viditeľná aj z tejto časti. Snažil som sa orientovať skôr na problém distribuovanosti projektu ako na jeho škálovanie, čo je podľa mňa častejší prípad, najmä v akademickom prostredí. A rovnako je to aj prípad nášho tímu, pretože dvaja z členov nášho tímu odchádzajú študovať do zahraničia a preto v prípade použitia niektorej z agilných metód sa z ich distribuovaním stretneme na vlastnej koži.

Zo všetkých uvedených experimentov, skúseností a odporúčaní je možné vyvodiť niekoľko záverov. Prvý z nich je, že agilné metódy *možno*, napriek silným varovaniam zo strany autorov agilných metód, škálovať. Ďalší z nich je, že agilné metódy možno škálovať, ale len vtedy, ak sa *vzdáme*, poprípade *zvoľníme* niektoré z charakteristík agilných metód.

V neposlednom rade je potrebné poznamenať, že aj samotné agilné metódy predstavujú *nový prístup*, čo samozrejme platí pre Škálované agilné metódy dvojnásobne. Preto je v tejto oblasti publikovaných veľmi málo prác, a ak aj boli, neboli v nich presne opísane metodologické postupy používané pri vývoji. A práve z dôvodu nedostatku informácii, empirických výsledkov o tom, ako skončili distribuované projekty a veľké projekty využívajúce agilné metódy vývoja projektu, nie je tieto možné v súčasnosti *objektívne* zhodnotiť. V každom prípade agilné metódy vývoja softvéru predstavujú *zaujímavú alternatívu* ku klasickým metódam vývoja softvéru, najmä v prostredí nejasných a meniacich sa požiadaviek. Ale pravdepodobne potrvá ešte istý čas, pokiaľ budú vývojárske a zákaznícke spoločnosti schopné akceptovať vývoj pomocou agilných metód aj v prípade *veľkých a distribuovaných* projektov.

Záver

Agilné metódy od svojho vzniku poskytujú zaujímavú alternatívu ku *klasickým* metódam vývoja softvéru. Trvalo istý čas, kým sa tieto metódy dostali do povedomia, začali byť používané v reálnych projektoch a osvedčili sa v praxi . Až samotná prax ukázala, s akými problémami je potrebné sa v prípade týchto metód vyrovnať. A samozrejme, aj aké sú najväčšie výhody týchto metód. Taktiež prax preukázala to, čo bolo deklarované už v samotnom Manifeste agilných metód. A to, že tieto metódy sú primárne určené a funkčné pre malé a lokálne tímy.

Potom, ako sa agilné metódy osvedčili v oblasti malých projektov a lokalizovaných tímov, vznikla otázka, či môžu byť úspešné aj v prípade veľkých projektov a distribuovaných tímov. Napriek tomu, že samotní autori agilných metód sa stavali k tomuto novému prístupu negatívne, boli už aj v tejto oblasti publikované výsledky. Jedná sa najmä o výsledky experimentov, ale v poslednom čase čím ďalej tým viac, aj o výsledky konkrétnych projektov. Na základe týchto výsledkov je možné predpokladať, že agilné metódy *je* možné škálovať, to znamená že ich *je* možné použiť

pre *veľké* projekty a *distribuované* tímy. Avšak pravdepodobne potrvá ešte istý čas, kým sa začnú *agilné metódy,* či už škálované alebo nie, používať vo všetkých tých projektoch, kde môžu priniesť *úsporu* času, peňazí a *kvalitnejší* výsledný produkt.

Použitá literatúra

1. Baheti, P., Williams, L., Gehringer, E., Stotts, D.: *Exploring Pair Programming in Distributed Object-Oriented Team Projects*, OOPSLA Educator's Symposium 2002, Seattle, WA., November 4 – 8, 2002
2. Beck, Kent: *Extreme Programming Explained: Embrace Change,* 1999, Addison - Wesley
3. Hoang, B., Khatkar, B., Momoh, J., Tu, W.: *Distributed Development and Scaling of Agile Methods,* USA, University of Calgary, 2003
4. *Jeffries, Ron: Manuals in Extreme Programming, 2001* http://www.xprogramming.com/xpmag/manualsInXp.htm
5. Kaehkoenen, T.: Agile Methods for Large Organizations–Building Communities of Practice, *Proceedings of the Agile Development Conference (ADC'04)*, Vol. 0, 2004, 2-11
6. Reifer, DJ., Maurer, F., Erdogmus, H.: Scaling Agile Methods*, IEEE Software,* Vol. 20, No. 4 (2003), 12-14
7. Tím Hobiti: *Vývoj softvéru v treťom tisícročí*, Slovensko, FEI STU Bratislava, 2002
8. Tím Kyklop: *Vývoj softvéru v treťom tisícročí*, Slovensko, FEI STU Bratislava, 2002
9. Wills, A.: *Dispersed Agile Software Development and Dispersed eXtreme Programming* , fastnloose, 2003, http://www.fastnloose.com/cgi-bin/wiki.pl/dad

Annotation

Agile methods of software development and their scaling

Agile methods have being established as an alternative to the classic methods of software development. Reasons of their success are discussed in the first section of this paper. In the same section are discussed basic characteristic of agile methods also. One of the most famous agile methods is Extreme programming, which is discussed in the next section. Second half of this paper is dedicated to scaling of agile methods, using agile methods in large and distributed projects. Published experiences and recommendations are discussed in these section beside experiments in this area.