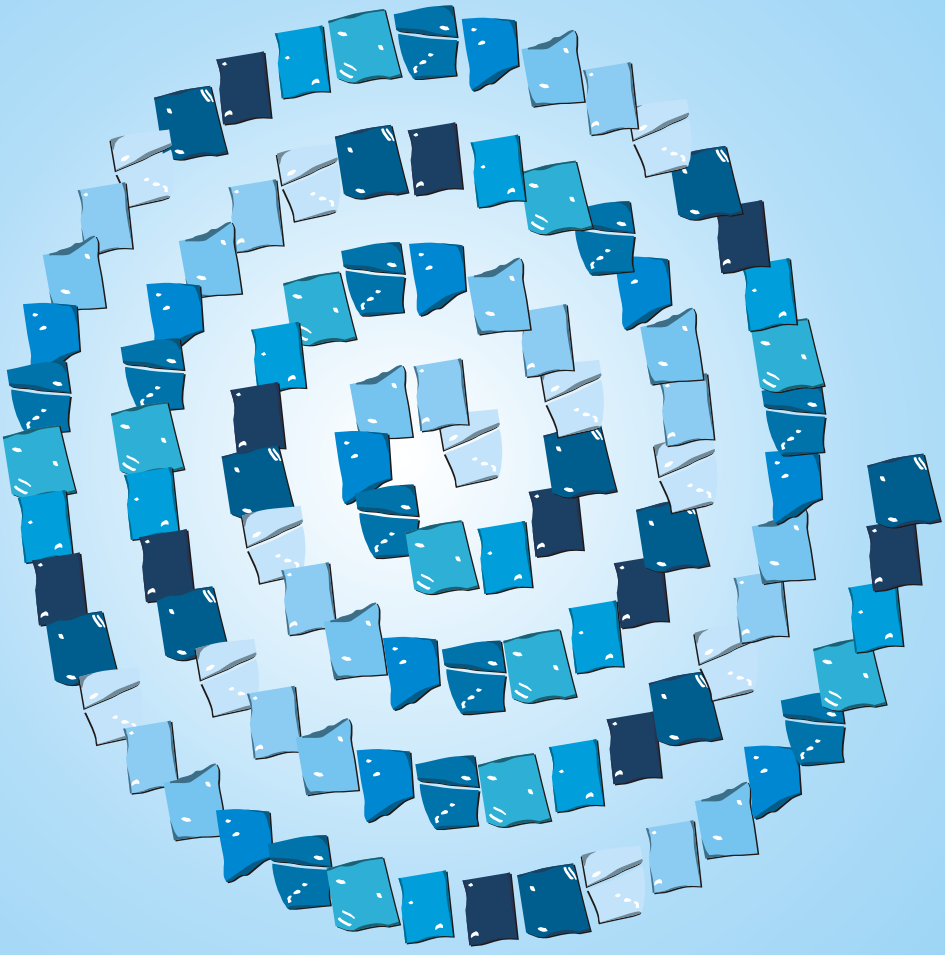
Manažment v softvérovom inžinierstve

Zborník esejí 2006



Mário Lenický (Ed.)

Fakulta informatiky a informačných technológií
Slovenská technická univerzita

Manažment v softvérovom inžinierstve

Zborník odborných esejí 2006

1. vydanie

Fakulta informatiky a informačných technológií
Slovenská technická univerzita v Bratislave

Manažment v softvérovom inžinierstve, január 2006, s. 1-128.

MANAŽMENT V SOFTVÉROVOM INŽINIERSTVE
Zborník
január 2006, 128 strán

Editori
Mário Lenický, Ivan Blanárik

Obálka
Peter Sýkora

Autori príspevkov
Igor Berta, Ivan Blanárik, Pavol Dragúň, Andrej Ďurica, Ľuboš Fazekaš, Andrej Janžo, Michal Jemala, Peter Kasan, Ladislav Kočiš, Attila Kotrba, Lukáš Kročka, Peter Sýkora, Tomáš Vanderka, Vojtech Szöcs

FAKULTA INFORMATIKY A INFORMAČNÝCH TECHNOLÓGIÍ
SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
Ilkovičova 3, 842 16 Bratislava

Predslov

V súčasnosti sa v čoraz vyššej miere dostáva na povrch marketingovej sféry pojem softvérového inžinierstva. Softvérové inžinierstvo je skloňované vo všetkých pádoch ako moderná technologická a manažérska disciplína týkajúca sa produkcie a údržby softvérových produktov, ktoré sú vyvíjané a udržiavané v určitom časovom horizonte pri použití určitých rozpočtových a časových zdrojov.

Práce v zborníku sú určené pre širší okruh manažérov, počnúc vedúcimi projektov až po vrcholových manažérov. Aj napriek faktu, že zborník neposkytuje hlboký náhľad do problematiky, môže byť nápomocný pri riešení jednoduchých problémov spojených s riadením projektov a vývojárskych tímov, prípadne osvetlí čitateľovi určitú časť problematiky, v ktorej nemá markantné skúsenosti, prípadne sa v nej necíti byť „doma“.

Napriek faktu, že manažment softvérového inžinierstva zohráva majoritnú úlohu pri tvorbe softvérového produktu a je zásadným činiteľom ovplyvňujúcim jeho kvalitu, ostáva úloha manažmentu softvérového inžinierstva na Slovensku naďalej značne podceňovaná, najmä v prostredí malých a veľkých firiem.

Projekt s kvalitným manažérskym prístupom vedie nepochybne k väčším výnosom za použitia menšieho množstva vynaložených zdrojov, a preto oblasť manažmentu softvérového inžinierstva odporúčame čitateľovi do pozornosti.

Prehľad zborníka

Časť orientovaná na špecifiká softvérových projektov

1. Berta, Igor: Agilné metódy vývoja softvéru
2. Blanárik, Ivan: Manažment softvérového systému a vplyv na manažment softvérového projektu
3. Ďurica, Andrej: Plánovanie a odhadovanie v softvérových projektoch
4. Fazekaš, Ľuboš: Nové technológie a nástroje a vplyv na manažment softvérového projektu
5. Janžo, Andrej: Manažment rizík v softvérovom projekte
6. Jemala, Michal: Sledovanie postupu softvérového projektu a manažment
7. Kočiš, Ladislav: Vzťah „zákazník a vývojár“ v softvérových projektoch
8. Sýkora, Peter: Manažment kvality a vplyv na výsledok projektu
9. Vanderka, Tomáš: Chyby a manažment softvérových projektov

Časť orientovaná na softvérové tímy

10. Dragúň, Pavol: Zlepšovanie produktivity softvérových tímov
11. Kasan, Peter: Manažment konfliktov v tíme
12. Kotrba, Attila: Vývoj tímu v softvérovom projekte a vplyv na manažment
13. Kročka, Lukáš: Manažment softvérových projektov a organizácia softvérových tímov
14. Szöcs, Vojtech: Manažment komunikácie pre lokalizované a distribuované softvérové tímy

Obsah

Úvod 6
Agilné metódy vývoja softvéru 7
Manažment softvérového systému a vplyv na manažment soft. projektu 23
Plánovanie a odhadovanie v softvérových projektoch 29
Nové technológie a nástroje a vplyv na manažment softvérového projektu 37
Manažment rizík v softvérovom projekte 45
Sledovanie postupu softvérového projektu a manažment 55
Vzťah zákazník a vývojár v softvérových projektoch 63
Manažment kvality a vplyv na výsledok projektu 69
Chyby a manažment softvérových projektov 77
Zlepšovanie produktivity softvérových tímov 83
Manažment konfliktov v tíme 91
Vývoj tímu v softvérovom projekte a vplyv na manažment 101
Manažment softvérového projektu a organizácia softvérových tímov 109
Manažment komunikácie pre lokalizované a distribuované softvérové tímy 117
Zhrnutie 127

Úvod

Napriek dlhoročnému výskumu a úsiliu mnohých odborníkov v sfére manažmentu softvérových projektov ostáva táto pomerne moderná disciplína softvérového inžinierstva ešte značne neprebádaná. Súčasný vývoj problematiky riadenia softvérových projektov sa ešte stále odohráva v zmysle hľadania otázok a odpovedí.

Nasledujúce riadky ponúkajú zborník odborných esejí, ktorý vznikol za účelom vytvorenia kolekcie študentských prác z oblasti manažmentu softvérového inžinierstva v rovnomennom študijnom predmete pod vedením doc. Ing. Márie Bielikovej, PhD. Publikácia zoskupuje štrnásť odborných esejí, pričom každá z esejí je reflexiou osobných myšlienok, úvah a príspevkov autora k aktuálnej téme. Prezentovaním svojich názorov a príspevkov v zborníku esejí priložia autori malý ale hodnotný úsek dláždenia na dlhej a náročnej ceste poznania metód a postupov manažmentu softvérových projektov.

Príspevky v tejto publikácii sa teda nesú jednak v zmysle osvieženia známej problematiky, ale aj v zmysle nových myšlienok tlmočených prispievateľmi. Autori sa zamerali na aspekty, ktoré sú pri riadení softvérového projektu kľúčové.

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]

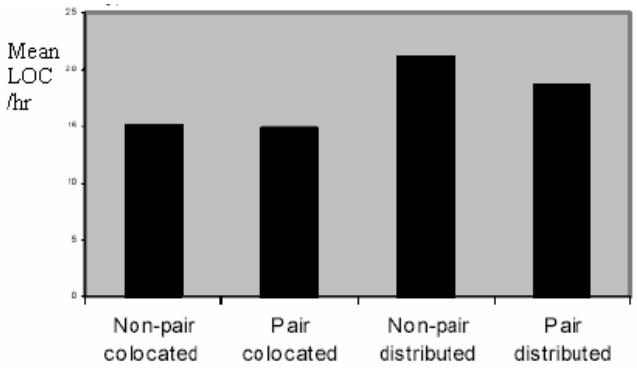
Ako bolo už niekoľko krát spomenuté, komunikácia má kľúčovú úlohu vo všetkých agilných metódach. Rovnako bolo spomenuté, že Extrémne programovanie a konkrétne jeho metóda Programovania v pároch predstavuje jednu z *komunikačne najnáročnejších* aktivít v rámci agilných metód. Dôvodom je úzka spolupráca 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.

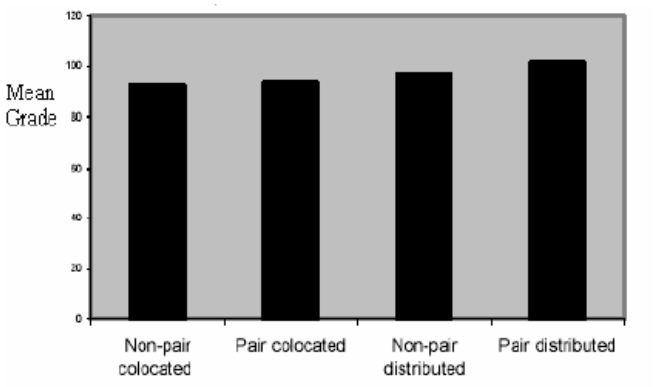
Na meranie produktivity bol použitý parameter Počet riadkov kódu za hodinu a pre meranie kvality softvéru bol použitý parameter získanej známky za tento projekt.



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.

Manažment softvérového systému a vplyv na manažment softvérového projektu

IVAN BLANÁRIK

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
ivan_blanarik@yahoo.com

Abstrakt. Vytvorenie softvérového systému už dávno nie je len otázkou programovania. Stále dôležitejšia sa ukazuje byť úloha manažmentu softvérového systému, ktorého cieľom je zosúladiť všetky prebiehajúce procesy. Vyvíjané softvérové produkty sa stávajú čoraz rozsiahlejšie a komplexnejšie. Na ich vývoji obyčajne spolupracuje niekoľko tímov, pričom pre úspešné vyriešenie projektu je nutné ich činnosť neustále kontrolovať a koordinovať. Bežným sa stáva aj priebežné upresňovanie a dopĺňanie požiadaviek kladených na výsledný produkt, pričom tie je nutné neustále aktualizovať. Tento článok sa zaoberá významom manažmentu softvérových systémov pri riešení týchto problémov. Vysvetľuje hlavné úlohy manažmentu konfigurácii a manažmentu zmien, ktoré sú jeho súčasťou. Nástroje používané pri riadení softvérových systémov uľahčujú vývoj nových produktov. V niektorých prípadoch však môže ich nasadenie viesť k zníženiu efektivity. V tomto článku sú uvedené niektoré aspekty, ktoré by mali byť zvážené pred prvým nasadením softvérového nástroja.

Úvod

V posledných rokoch sa na trh dostáva obrovské softvérových produktov. Zákazníci kladú čoraz väčšie a väčšie požiadavky na funkcionalitu softvéru a ten sa následne stáva rozsiahlejším a zložitejším. Na jeho vývoji je nutná zväčša spolupráca viacerých tímov navzájom, pričom ich činnosť treba nejakým spôsobom vhodne koordinovať, aby ich práca bola efektívna a rýchla.

Dnešným trendom sa zároveň stáva neustále skracovanie životného cyklu vývoja nového produktu. Na jeho skrátenie vplývajú viaceré faktory, či už sú požiadavky od zákazníkov, ktorí chcú vyvíjaný softvér čo najskôr používať, alebo samotné konkurenčné prostredie trhu, kde by zdĺhavý vývoj nového produktu mohol viesť k strate potencionálnych zákazníkov. Často sa preto stáva, že počiatočné požiadavky na softvér bývajú nekompletné, nekonzistentné a musia sa ujasňovať počas celého životného cyklu jeho vývoja. V minulosti boli požiadavky na softvér analyzované a pomerne presne špecifikované už v začiatočných fázach projektu. V dôsledku rýchleho

Manažment softvérového systému a vplyv na manažment softvérového projektu, január 2006, s. 23-28.

vývoja je teraz nutné starať sa o zapracovanie zmien do požiadaviek kedykoľvek je to nutné.

Manažment softvérových systémov, ako jedna z disciplín softvérového inžinierstva, je nevyhnutný pre úspešné dokončenie projektu. Jeho úlohou je práve starať sa o zosúladenie činností jednotlivých tímov pracujúcich na projekte, uchovávanie všetkých verzii vyvíjaného produktu a riadenie zmien požiadaviek počas projektu s cieľom zvyšovať efektivitu práce.

Pri manažmente softvérových systémov možno účinne využívať rôzne špeciálne nástroje a tým automatizovať nutné úkony spojené s riadením. Všeobecne ich používanie prináša lepšie výsledky a veľké projekty sa už bez nich ani nezaobídu. Ich efektívnosť však závisí od množstva faktorov a niektorých prípadoch môže dokonca viesť k zníženiu produktivity.

Manažment konfigurácii

Riadenie vývoja softvérových systémov so sebou prináša rad problémov, ktorých príčinami môžu byť napríklad aj:

1. modifikácie zdrojového kódu – programátori môžu kedykoľvek pozmeniť kód programu, resp. pozmeniť jeho funkcionalitu
2. závislosti medzi modulmi – jednotlivé moduly sa často vyvíjajú paralelne, pričom bývajú medzi sebou funkčne zviazané, a teda modifikácia jedného z nich môže ovplyvniť celý systém
3. vzájomné ovplyvňovanie práce – pretože softvér vyvíjajú tímy, ktoré pracujú na rovnakej časti, činnosť jedného pracovníka môže ovplyvňovať prácu ostatných členov tímu [6]

Manažment konfigurácii je časťou manažmentu softvérových systémov. Zameriava sa na riadenie činností vykonávaných počas vývoja – kontrolovaním zmien sa snaží zabezpečiť stabilný vývoj softvérového produktu, resp. predísť chaosu spôsobeného množstvom korekcii, rozšírení a prispôsobení funkcionality softvéru.

Pre malé tímy je ešte reálne riadiť vývoj softvéru komunikáciou každého s každým. Akonáhle sú však tímy väčšie, nie je únosné, aby bol vývoj riadený takýmto spôsobom. Namiesto toho sa používajú procedúry ako napr. správy a oznámenia, ktoré všetkých členov tímov oboznamujú s aktuálnym stavom vývoja. Takéto a ďalšie procedúry majú zabrániť nežiaducim zmenám v softvéri, ktoré by mohli nepriaznivo ovplyvniť prácu ostatných.

Manažment zmien

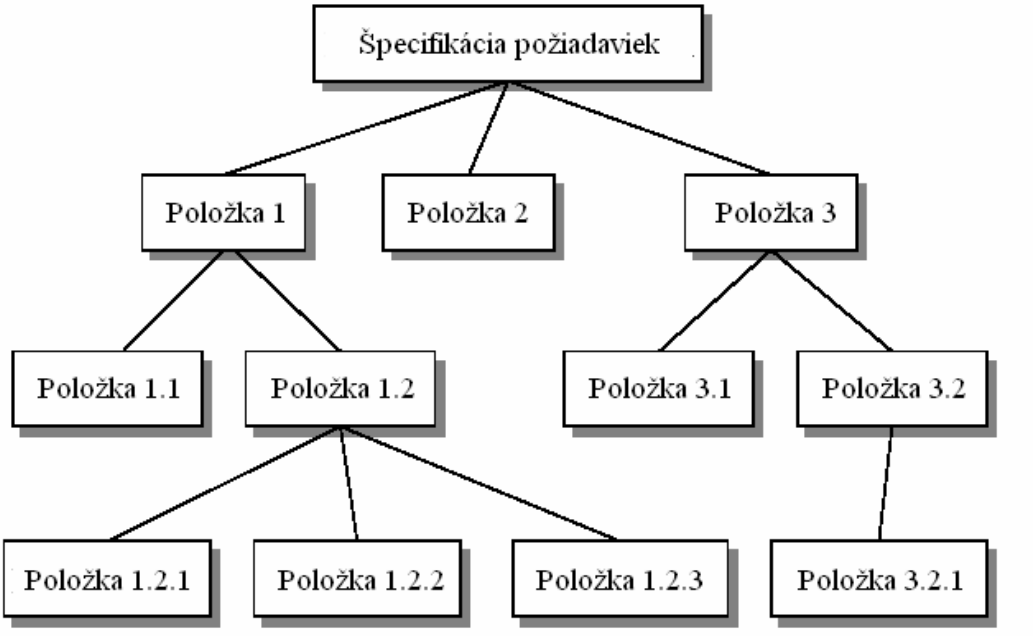
Ako bolo vyššie spomenuté, požiadavky sa výsledný produkt bývajú často nekompletné a preto je nutné ich postupne upravovať a dopĺňať. Proces ujasňovania požiadaviek sa teda vykonáva počas celého životného cyklu vývoja, obzvlášť pri

použití vývojového modelu ako sú napr. inkrementálne vyvíjanie, prototypovanie a pod. [3]

Pri riadení zmien požiadaviek vystupujú do popredia nasledujúce problémy:

4. nesprávne zaobchádzanie s požiadavkami počas návrhu a implementácii softvéru
5. neadekvátne pochopenie zmien vykonaných v požiadavkách počas vývoja systému
6. nedostatočná kontrola nad tým, ktoré požiadavky sa práve implementujú, a ktoré sú už splnené
7. neadekvátna validácia a verifikácia výsledného systému voči požiadavkám naň kladených

V prípade, že je špecifikácia požiadaviek uchovávaná v rámci jedného dokumentu, môžeme síce manažovať ich rôzne verzie, ale iba ako celok – nemáme prístup k jednotlivým požiadavkám zvlášť. Z tohto dôvodu sa definuje napr. virtuálny dokument pozostávajúci z položiek špecifikácie požiadaviek organizovaných do štruktúry znázornenej na **Obr. 1**. Najvyššia úroveň v ňom predstavuje základné požiadavky špecifikácie, ktoré sú ďalej spresňované na nižších úrovniach. Jednotlivé položky môžu byť popísané buď prirodzeným jazykom alebo grafmi, formálnou špecifikáciou a pod.



Obr. 1. Špecifikácia požiadaviek

S každou položkou je späté jej číslo, názov a ďalšie potrebné atribúty prislúchajúce danej verzii požiadaviek, ako sú číslo verzie položky, meno jej autora a dátum vzniku, stav spracovania a pod.

Akonáhle sú požiadavky rozdelené na samostatné entity, môžeme spravovať ich jednotlivé verzie. Môžeme pridávať nové položky, modifikovať ich alebo odstrániť staré, čím dostaneme novú verziu požiadaviek. V každom okamihu je teda k dispozícii aktuálna verzia požiadaviek spolu so stavom, v akom sa nachádza ich realizácia. To umožňuje udržiavať si dokonalý prehľad o tom, aká časť systému je už hotová, a ktoré požiadavky sa ešte musia implementovať.

Nástroje pre manažment softvérových systémov

Riadenie vývoja softvérového systému bez akýchkoľvek podporných nástrojov by bolo pre drvivú väčšinu projektov len ťažko predstaviteľné. Vo svete preto existuje množstvo rôznych programov uľahčujúcich manažment softvérových systémov. Niektoré z nich sú voľne dostupné, väčšina je však spoplatňovaná niekedy až do rádovo tisícok dolárov.

Nástroje pre manažment softvéru majú množstvo spoločných vlastností a odlišujú iba niektorými špecifickými črtami. Medzi ich základné vlastnosti patrí napríklad podpora viacerých používateľov, intuitívne používateľské rozhranie, škálovateľnosť, flexibilita pri integrácii s inými nástrojmi, ľahké nastavenie atď. Ich spoločným cieľom je čo najviac automatizovať nutné úkony spojené s riadením projektu.

Tieto nástroje síce uľahčujú manažment projektu a zefektívňujú prace na vývoji softvéru, avšak nemožno očakávať, že iba ich zakúpenie a nasadenie do prevádzky prinesie obrovský krok vpred. Bez oboznámenia sa s daným nástrojom a vytvorením firemnej politiky jeho používania bude proces vývoja softvéru aj naďalej prebiehať neorganizovane [1].

Často nepomáhajú tieto nástroje v takej miere, ako by mohli. Dôvodom môžu byť neexistujúce mechanizmy na vykonávanie bežných úkonov alebo ich použitie je príliš komplikované. Programátori sa chcú plne sústrediť na svoju prácu - na vytváranie softvéru a čokoľvek ostatné považujú za zbytočne vynaložené úsilie. Nástroje na manažment sa používajú každodenne a preto by ich použitie malo byť čo najmenej citeľné, resp. obťažujúce. Naopak, čo by si mali programátori povšimnúť, je ich užitočnosť pri práci. Pokiaľ si programátori uvedomia, že im tieto nástroje pomáhajú zlepšiť ich pracovný výkon, budú ich používať aj z vlastnej vôle.

Nástroje pre manažment softvérových systémov sústredia svoju pozornosť hlavne na podporu stredných a veľkých projektov, kde je ich využitie už skutočne nevyhnutné. Môžu byť samozrejme nasadené aj v rámci menších projektov. Pri nich však už treba zvažovať, či ich prínos v projekte bude prevyšovať vedľajšie efekty vyplývajúce z ich použitia. Používanie každého nástroja totiž zaberá určité množstvo času a v malých tímoch môže byť vzájomná komunikácia ich členov efektívnejšia. Samozrejme svoju rolu pri nasadení nejakého nástroja zohráva aj nutnosť oboznámiť sa s jeho prostredím a rozhodne aj cena, ktorá častokrát býva nezanedbateľná.

Ďalším z faktorov, ktoré treba zvážiť pri výbere vhodného nástroja, je platforma, na ktorej bude projekt prebiehať. Taktiež výkonnosť nástroja je dôležitá, resp. či sa bude nástroj starať o niekoľko megabajtov alebo niekoľko stoviek. Iný nástroj sa bude pri riadení a kontrole softvéru pozostávajúceho z niekoľko sto súborov a iný pri tisícoch súborov. Rozhodujúcim môže byť aj podpora, akú predajca k výrobku poskytuje.

Záver

Manažment softvérového systému je neoddeliteľná súčasť riadenia projektu. Bez neho by bol vývoj akéhokoľvek väčšieho systému len ťažko predstaviteľný. Jeho úlohou je totiž koordinovanie prác jednotlivých tímov a členov tímu, a tak zabraňovať nežiaducim kolíziám pri tvorbe softvéru. Zároveň sa tiež stará o riadenie zmien v špecifikácii požiadaviek a kontrolovanie stavu ich plnenia.

Na manažment softvérového systému sa s úspechom používajú rôzne špecializované softvérové nástroje. Na trhu sa ich vyskytuje niekoľko desiatok, pričom sa na prvý pohľad odlišujú jeden od druhého len nepatrne. Pri výbere nástroja, ktorý bude najviac vyhovovať požiadavkám firmy, je nutné brať do úvahy množstvo faktorov. Popri finančných aspektoch je treba zobrať v úvahu aj rozsah projektov, pri ktorých má byť nástroj nasadený.

Je nutné zvoliť si nástroj odpovedajúci potrebám projektu. Ten však nie je všetko, aj keď vo väčšine prípadov znamená krok vpred. Nesprávne používanie nástroja bez nejakej firemnej stratégie môže prinášať rovnaké alebo ešte horšie výsledky ako jeho nepoužívanie.

Použitá literatúra

1. Berczuk, S.: Pragmatic Software Configuration Management. IEEE Software, 2003
2. Chan, A. K. F., Hung, S.: Software Configuration Management Tools. *8th International Workshop on Software Technology and Engineering Practice*, IEEE Press, 1997
3. Crnkovic, I., Funk, P., Larsson, M.: Processing Requirements by Software Configuration Management. *25th Euromicro Conference (EUROMICRO '99),* Vol. 2, 1999
4. Estublier, J.: Software configuration management: a roadmap. *Proceedings of the Conference on The Future of Software Engineering*, ACM Press, 2000
5. Estublier, J., Leblanng, D., Clemm, G., Conradi, R., Hock, A., Tichy, W., Wiborg-Weber, D.: Impact of the Research Community for the Field of Software Configuration Management. *24th International Conference on Software Engineering*, ACM Press, 2002

6. Grinter, R. E.: Using a configuration management tool to coordinate software development. *Proceedings of conference on Organizational computing systems*, ACM Press, 1995

Annotation

Software system management and influence on software project management

Management of software system is an essential part of development of a new software product. It‘s hard to imagine how could we control and coordinate a large project, which includes many teams, without any software system management tool. However, simply buying a tool a putting it in a place doesn‘t mean automatically success in project. In this article are mentioned main goals of software management and I offer some aspects, which should be considered before deciding for some tool.

Plánovanie a odhadovanie v softvérových projektoch

ANDREJ ĎURICA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
durica01@student.fiit.stuba.sk

Abstrakt. Pri súčasnom trende vývoja softvérových výrobkov sa dostáva do popredia, ako jedna veľmi dôležitá činnosť, práve manažment projektu. Skladá sa z viacerých častí, ktoré sa snažia zabezpečiť organizáciu, plánovanie projektu a rozvrhnutie hlavných vykonávaných činností. Jednou z týchto častí je práve plánovanie a odhadovanie, ktorou sa podrobnejšie zaoberá tento dokument. Dôraz je tu kladený na objasnenie procesu plánovania, a to z viacerých pohľadov. Ide v podstate o určitú postupnosť jednoduchých krokov, ktoré sa môžu aj niekoľkokrát opakovať. Iný pohľad hovorí o rozdelení na základné procesy, čiže samotné vytvorenie rozvrhu a rozpočtu, a na podporné procesy, kam by sme mohli zahrnúť napríklad analýzu a manažment rizík. Odhadovanie je rozdelené na štyri kroky: odhadovanie veľkosti projektu, potrebného ľudského úsilia, rozvrhu a ceny projektu. Nasledujú možnosti spôsobu odhadovania projektu, pričom najlepšie je vychádzať z predošlých skúseností, ak sú k dispozícii. Ak nie, môžu sa použiť zaužívané postupy, alebo matematické úvahy a jednoduché vzorce. Na záver sú uvedené možné dôvody a dôsledky podcenenia plánovania a odhadovania, kam patria narýchlo zostrojené slabé plány, podcenenie mnohých faktorov, ako je zložitosť systému, množstvo ľudských zdrojov apod.

Úvod

V súčasnosti, keď hovoríme o vývoji softvéru, ide o zložitý a často dlhý proces, kde je potrebné koordinovať veľké množstvo činností, na ktorých sa zúčastňuje značný počet ľudí. Preto sa za posledných pätnásť rokov stále viac dostáva do popredia pojem manažment softvérových projektov.

Manažment softvérových projektov zahŕňa viacero činností. Medzi hlavnými činnosťami by sme mohli vyhradiť zopár pojmov, ktoré najlepšie charakterizujú ich podstatu. Hovoríme teda o organizovaní, plánovaní projektu a vytváraní rozvrhu plánovaných činností.

Medzi prvoradé zámery manažmentu softvérových projektov patrí zabezpečenie odovzdania softvéru načas a s daným rozpočtom, pričom musí spĺňať požiadavky spoločnosti, ktorá ho vyvíja, ale hlavne tej, pre ktorú je vyvíjaný.

Plánovanie a odhadovanie v softvérových projektoch, január 2006, s. 29-36.

Keďže proces vývoja softvéru nie je štandardizovaný, pri vývoji ide vždy nanovo o zložitý proces, na ktorom sa podieľa väčšinou veľké množstvo ľudí. Takýto vývoj obvykle trvá dlhšiu dobu (dokonca až niekoľko rokov). Preto je potrebné čo najlepšie rozvrhnúť, plánovať a celkovo manažovať všetky faktory, ktoré vplývajú na vývoj softvéru (finančné prostriedky, ľudské zdroje, čas,…).

Manažovanie projektu však nemusí hneď zaručiť, že budú splnené všetky požiadavky kladené na vývoj softvéru a na softvér samotný. Dokonca aj veľmi dobre manažovaný projekt môže niekedy zlyhať. Avšak vôbec nemanažované projekty sú doslova odsúdené na zlyhanie. Preto je manažovanie projektov dôležité. Aj keď v ojedinelých prípadoch nemusí zaručiť požadovaný výsledok, je však výrazne vyššia pravdepodobnosť, že sa to podarí.

Medzi činnosti vykonávané v manažmente patria hlavne: písanie ponuky, resp. návrhu, výpočet nákladov, plánovanie a odhadovanie, monitorovanie a zhodnotenie projektu, výber a oceňovanie personálu, písanie výkazov, resp. zápisov a prezentácií. Táto odborná esej sa bude zaoberať jednou zo spomínaných častí, a to práve plánovaním a odhadovaním projektov.

Plánovanie a odhadovanie v softvérových projektoch

Plánovanie v softvérovom projekte vlastne zahŕňa všetky oblasti odhadovania, analýz rizík a rozvrhovania. Akokoľvek, v kontexte prostriedkov, resp. zdrojov, plánovanie obsahuje odhadovanie, t.j. pokus o určenie, koľko peňazí, úsilia, ľudských, ale aj iných zdrojov a koľko času bude stáť vytvorenie špecifického softvérového systému. Zároveň tu môžeme identifikovať samotné plánovanie a rozvrhovanie činností a predvídateľných okolností projektu.

Plánovanie

Pri plánovaní ide pravdepodobne o činnosť, ktorá zaberá v manažovaní projektu najviac času. Predstavuje spojitú aktivitu, ktorej intenzita by sa mala držať stále na rovnakej úrovni od prvotného konceptu až po dodanie systému. Plán totiž nie je nič statické. Práve naopak, stále sa mení. V začiatočnej fáze projektu sa mení vždy pri získaní nových poznatkov o projekte a v ďalších fázach sa tiež neustále zjemňuje a aktualizuje informácie.

Plánovaním sa vlastne snažíme redukovať neurčitosť výsledku projektu, zvyšovať prehľad o budúcich požadovaných činnostiach, čím sa zároveň zvyšuje výkonnosť [1].

Proces plánovania

Sommerville [3] identifikoval proces plánovania projektov pomocou jednoduchého algoritmu zapísaného nasledovným spôsobom:

Stanoviť obmedzenia projektu
Urobiť počiatočné odhady parametrov projektu
Určiť míľniky projektu a doby doručenia
Pokiaľ sa projekt neskončil alebo nebol zrušený, *slučka*

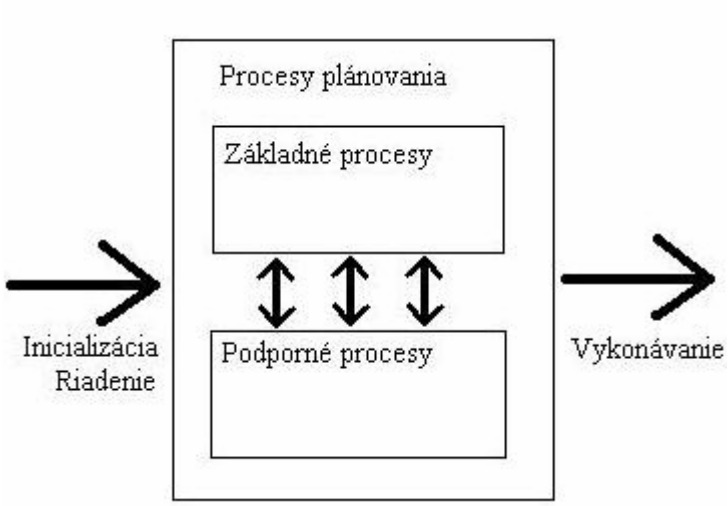
Vystav projektový rozvrh
Iniciuj činnosti podľa rozvrhu
Čakaj (chvíľu)
Zhodnoť vývoj (resp. pokrok) projektu
Zreviduj odhady parametrov projektu
Aktualizuj rozvrh projektu
Prejednaj míľniky a doby doručenia projektu
Ak (nastali problémy) *tak*
Iniciuj technické zhodnotenie a možnú revíziu
Koniec ak
Koniec slučky

Tento algoritmus ponúka náhľad na to, ČO by sa malo robiť pri plánovaní projektu a v akom poradí. To AKO a aj ostatné podrobnosti však už zostávajú ponechané na jednotlivca, keďže každý projekt si bude vyžadovať vlastné charakteristické prístupy. Nasledujúce poznámky charakterizujú zopár dôležitých činností pri procese plánovania, ktoré je potrebné zvážiť pri zostavovaní konkrétnejšieho plánu:

- stráviť dostatočný čas pochopením samotného problému.
- odhadnúť množstvo potrebného úsilia, čo je obvykle veľmi ťažké, keďže sa má na základe pár informácií pomerne podrobne určiť vývoj projektu a predpovedať možné problémy, ktoré ho môžu skomplikovať. Tu je zároveň potrebné určiť základné funkcie systému a ich zložitosť.
- vyvinúť rozvrh so zahrnutím určitej „bezpečnostnej siete“, t.j. samotný odhad ešte zväčšiť o určitú hodnotu danú nejakým faktorom. Zároveň by bolo vhodné stanoviť záchranný plán, ktorý by sa nasadil v krízových situáciách. Na konci je žiadané ešte raz skontrolovať reálnosť rozvrhnutých činností.
- zrevidovať rozvrh pri každom väčšom pokroku v pochopení projektu, alebo pri získaní nových informácií.

Rozdelenie procesov plánovania

Podľa [1] sa procesy plánovania softvérového projektu delia na *základné* a *podporné procesy* (pozri **Obr. 1**).



Obr. 1. Delenie procesov plánovania softvérového projektu podľa [1]

Základné procesy

Základné procesy plánovania sa sústreďujú najmä na vytvorenie rozvrhu a rozpočtu projektu, pričom tieto sa vykonávajú v určitom rozsahu v každom projekte. Dôležité je poznamenať, že plánovanie nie je exaktná veda, teda dva rôzne tímy pravdepodobne vytvoria rôzne plány pre ten istý projekt. Neexistujú žiadne presné postupy, iba návody.

Podporné procesy

Podporné procesy plánovania viac závisia od skutočnej povahy projektu. Ich vykonávanie by malo byť v nejakej miere zahrnuté v plánovaní každého projektu, ale ich presnejšie časové určenie a potrebný rozsah sa zvyčajne ukáže až v priebehu plánovania (napr. riziká sa na začiatku plánovania môžu zdať minimálne, avšak po vytvorení rozvrhu projektu sa ukáže, že tento je veľmi agresívny a teda vyžaduje analýzu vzniknutého rizika).

Medzi hlavné charakteristiky sem patrí riziko a s ním spojený manažment, ako je identifikácia rizík, ich analýza a zapracovanie do plánu. Ďalším dôležitým faktorom je organizácia vrátane personálneho manažmentu a komunikácia v projekte.

Plánovanie a odhadovanie

Plánovanie a odhadovanie sú tímovo alebo skupinovo orientované činnosti. Dôvod tohto je tak jednoduchý ako aj vynikajúci. Dal by sa vyjadriť pomocou známeho porekadla „Dve hlavy sú viac ako jedna“, čiže keď sa na činnosti podieľa viacero ľudí, je menej pravdepodobné, že sa urobí veľké množstvo chýb a celkovo bude výsledok plánu a odhadu jasnejší a bude vystihovať podstatu.

Keďže plánovanie je vykonávané na rôznych úrovniach organizácie, čo zahŕňa obchodné strategické plánovanie, programové a projektové plánovanie, operačné plánovanie a ďalšie, je v úlohe plánovania zahrnutý širší rozsah funkčných zodpovedností..

Odhadovanie

Efektívne odhadovanie v softvérovom projekte je jedna z najnáročnejších a najdôležitejších činností pri vývoji softvéru. Správne projektové plánovanie a riadenie nie je možné bez správneho a spoľahlivého odhadu. Softvérový priemysel, ako celok, neodhaduje projekty veľmi dobre a náležite ich nevyužíva. Projekt a aj jeho účastníci potom trpia pod takouto chybou oveľa viac, ako by museli, a následne je potrebné zamerať určité úsilie na zlepšenie aktuálnej situácie, čím sa vlastne len dobieha to, čo sa predtým vlastnými chybami zameškalo.

Podľa [2] zahŕňa odhadovanie v softvérových projektoch nasledujúce štyri základné kroky:

- odhadnúť veľkosť vyvíjaného produktu, čo sa dá merať rôznymi mierkami (napr. počet riadkov programu, počet funkčných modulov…).
- odhadnúť potrebné úsilie v človeko – mesiacoch alebo inej mierke zobrazujúcej časové využitie ľudských zdrojov.
- odhadnúť rozvrh v kalendárnych mesiacoch.
- odhadnúť cenu projektu v príslušnej mene.

Odhadovanie veľkosti

Precízny odhad veľkosti vytváraného softvéru je základ efektívneho odhadu. Východisko informácií ohľadom oblasti projektu by sa malo, pokiaľ je to možné, začať formálnym opisom požiadaviek, napr. špecifikácia požiadaviek systému zákazníka, systémová špecifikácia… Ak je projekt znova odhadovaný v neskorších fázach životného cyklu projektu, dokumenty opisujúce návrh môžu byť použité na poskytnutie ďalších detailov. Nedostatok formálnej východiskovej špecifikácie by nemal odradiť od prvotného odhadu projektu. Niekedy je k dispozícii len slovný opis alebo matné vyčlenenie obrysov. V každom prípade je nevyhnutné konzultovať úroveň rizika a neistoty odhadu so všetkými zainteresovanými osobami a po objavení sa nových východiskových informácii prerobiť odhad.

Odhadovanie úsilia

Keď je hotový odhad veľkosti projektu, je možné z toho odvodiť odhad potrebného ľudského úsilia. Tento prepočet z veľkosti softvéru na úplné vynaložené úsilie je možné vykonať iba ak je definovaný vývojový životný cyklus softvéru a vývojový proces, ktorého sledovaním sa špecifikuje, navrhne, implementuje a otestuje softvér. Vývoj softvérového projektu zahŕňa oveľa viac ako len jednoduché naprogramovanie softvéru. V skutočnosti programovanie je často najmenšia časť celkového úsilia. Písanie a zhodnocovanie dokumentácie, implementácia prototypov, návrh jednotlivých

čiastočne dodávaných častí, prehodnocovanie a testovanie programu zaberá väčšiu časť celého úsilia projektu. Odhad úsilia na projekte si vyžaduje identifikáciu, odhadnutie a následné zhrnutie všetkých činností, ktoré je potrebné vykonať na vytvorenie produktu odhadovanej veľkosti.

Odhadovanie návrhu

Tretím krokom v odhadovaní vývoja softvérového projektu je určenie projektového rozvrhu z odhadu úsilia. Toto vo všeobecnosti zahŕňa odhadovanie počtu ľudí, ktorí sa budú na projekte zúčastňovať, na akých činnostiach sa budú zúčastňovať, kedy začnú svoje aktivity na projekte a kedy ich ukončia. Keď sú tieto informácie zozbierané, je potrebné ich zapísať do plánovacieho kalendára.

Odhadovanie ceny

Je veľa faktorov, ktoré je potrebné zvážiť pri odhadovaní celkovej ceny projektu. Tu musia byť zahrnuté náklady na ľudskú prácu (mzdy), zakúpený, resp. prenajatý softvér a hardvér, výdavky na cestovanie za účelom stretnutí a testovania, telekomunikačné náklady atď. Konkrétne, ako bude odhadnutá končená cena projektu, závisí na rozdeľovaní finančných prostriedkov samotnej firmy, ktorá projekt realizuje. Často projektový vývojový manažér odhaduje iba náklady na mzdy a identifikuje prídavné projektové náklady považované organizáciou za mimoriadne výdavky. Najjednoduchšie, ako sa dajú získať náklady na mzdy, je vynásobiť odhadom úsilia projektu s nejakou všeobecnou hodnotou práce. Konkrétnejšie hodnoty by sa dali získať použitím špecifických hodnôt pre jednotlivé pozície a príslušný počet ľudí na danej pozícii, ktorí sa budú na projekte zúčastňovať. Tu je ale potrebné vedieť rozdeliť celkové úsilie medzi jednotlivé pozície.

Spôsob odhadovania

Pri odhadovaní, a to platí aj pre jednotlivé spomínané kroky odhadovania, je najlepšie vychádzať z predošlých skúseností. Hovoríme teda o odhadovaní analógiou. Ak už bol vopred odhadovaný, resp. vyvíjaný podobný softvérový projekt, je viac ako vhodné využiť tieto poznatky pri odhadovaní nových projektov. Je tu potrebné stanoviť určitý pomer medzi predošlými projektmi a súčasným vyvíjaným, aby sa dala podľa daného pomeru aplikovať podobnosť odhadu.

Ak neexistujú žiadne skúsenosti z predošlých podobných projektov, je vhodné použiť nejakú inú metódu. Môže to byť jednoduchá matematika a zdravý rozum, napríklad pre odhadovanie veľkosti projektu by to bol počet požadovaných funkcií spolu s ich zložitosťou. Pri odhadovaní úsilia je možné použiť dnes už pomerne vyspelé a všeobecne uznané algoritmické postupy, ako je napr. COCOMO model (pozri [1]), ktoré boli odvodené na základe mnohých vykonaných projektov. Ďalšou možnosťou, ako je tomu pri odhadovaní rozvrhu, je možné použiť nejaký matematický model (pozri [2]).

Podcenenie plánovania a odhadovania

Plány sú často narýchlo a neadekvátne zostrojené, čo sa deje primárne kvôli ich nekorektnému navrhnutiu. Toto môže vyústiť v nedostatočnom vyhradení času na splnenie jednotlivých úloh a v niektorých prípadoch môžu byť aj potrebné činnosti úplne odignorované.

Kvôli slabým plánom a odhadom môže vzniknúť prílišné prepracovanie projektu, zhusťovanie činností, podcenenie potrebných zdrojov (ľudských aj iných) a napokon premeškané dátumy odovzdania alebo nízka kvalita odovzdaného projektu. Projekt potom za následok stojí viac ako mal, alebo trvá dlhšie ako bol naplánovaný. Toto zase spôsobí oneskorenie dodania v tomto projekte využitých ľudských zdrojov na ďalší projekt.

Pre zaručenie efektívneho plánovania a odhadovania musia byť jasne stanovené ciele a zamýšľané výsledky. Bez týchto alebo bez angažovanosti manažmentu je nepravdepodobné vytvorenie užitočných plánov. Nedostatok rozumných plánov má za následok vznik programu alebo projektu, ktorý nedosiahne požadované ciele a už vôbec nie predpokladané zisky.

Záver

Dokument sa zameral na časť manažmentu softvérového projektu. Hlavným cieľom bolo priblížiť čitateľovi činnosti a charakteristiky plánovania a odhadovania. Pri plánovaní bol kladený dôraz na proces plánovania, pričom boli uvedené dva rôzne postoje k tejto problematike: algoritmický postup zobrazujúci plánovanie ako postupnosť krokov a plánovanie rozdelené medzi dve skupiny, akúsi hlavnú a podpornú (nie však menej cennú). Pri odhadovaní boli opísané štyri hlavné kroky, ktoré je potrebné vykonať na vytvorenie dobrého odhadu softvérového projektu (odhadovanie veľkosti projektu, potrebného úsilia, rozvrhu a ceny projektu). Ďalej sú zobrazené možné spôsoby odhadovania, kde je vyzdvihované odhadovanie na základe predošlých skúseností. Na záver sú uvedené možné dôsledky a dôvody podcenenia plánovania a odhadovania a tým je zdôraznená ich nevyhnutnosť v procese manažmentu softvérových projektov.

Použitá literatúra

1. Bieliková, M.: Softvérové inžinierstvo – Princípy a manažment, STU Bratislava, 2000. 220 s. ISBN 80-227-1322-8
2. Peters, K.: Software project estimation. Software Productivity Ceneter Inc., 1999, http://www.spc.ca/downloads/resources/estimate/estbasics.pdf (11.12.2005)
3. Sommerville, I.: Software engineering, 5th edition, kap. 3, Addison-Wesley 1996

Annotation

Planning and estimating in software projects

In the current software project development trend as one of very important activities project management is getting into spotlight. It contains several pieces trying to ensure the organization, planning of projects and to allocate the main executed parts. Planning and estimating is one of this pieces which is this document dedicated to. The stress is put on clarify the planning process from more points of view. In fact it consists of some series of simple steps which can be repeated several times. Another view of this is dividing the process into main processes and supporting processes. Main processes consist of the creation of schedule and budget estimation. Supporting processes include for example risk analysis and management. Estimating is divided into four steps: estimating the size of project, the effort, the schedule and project cost. Next are the opportunities of project estimating. The best way is to use the know-how from previous projects, if there are any. If not, there can be made use of often used practices, or mathematical formula. At the end possible reasons and effects of underestimating projects are shown. There could be many factors like in haste made weak plans, underestimating the size or effort of the project.

Nové technológie a nástroje a vplyv na manažment softvérového projektu

Riziká zavádzania nových technológií a nástrojov do procesu vývoja softvéru a ich manažment

ĽUBOŠ FAZEKAŠ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
nespokojny@atlas.sk

Abstrakt. Ak chce softvérová firma prežiť na súčasnom trhu, musí zvládnuť proces prechodu na nové technológie. Táto potreba vyplýva z toho, že sa neustále objavujú nové technológie, ktoré je potrebné využiť vo vývojovom procese, či už preto, že si to vyžaduje samotný riešený problém alebo zákazník. Zvládnutie tohto procesu nie je triviálna záležitosť. Vyplývajú z neho mnohé riziká a je potrebné uplatniť špecifické nástroje na ich elimináciu. Táto esej rozoberá príčiny vzniku potreby novej technológie vo vývojovom procese a hlavné riziká, ktoré z jej nasadenia vyplývajú. Potom postupne predstavuje jednotlivé nástroje, ktoré môže projektový manažér použiť na elimináciu týchto rizík.

Úvod

Vývoj softvéru je veľmi komplikovaný proces a jeho manažment si vyžaduje veľké množstvo znalostí, ale hlavne skúseností. Navyše tento proces prebieha v neustále meniacich sa podmienkach, ktorým sa manažér musí vedieť flexibilne prispôsobovať.

Niektorí autori definujú manažment ako minimalizáciu rizík. Manažér v oblasti vývoja softvéru sa snaží eliminovať tri hlavné riziká:

- projekt neskončí načas
- náklady na projekt prekročia určený rozpočet
- výsledný produkt nebude spĺňať požadovanú funkcionalitu

Takýto pohľad na projekt predstavuje vysoko úrovňové nazeranie, ktoré pre riadenie projektu veľa neprináša. Každý manažér si musí položiť otázku, aké sú faktory, ktoré vplývajú na termín ukončenia projektu, čerpanie zdrojov a výslednú funkcionalitu. Dostane sa tým k tomu, že každé z týchto rizík pozostáva z viacerých, už konkrétnych rizík, ktoré majú rôznu váhu. Tieto konkrétne riziká možno rozčleniť do viacerých oblastí, pričom niektoré sa môžu nachádzať aj vo viacerých definovaných

Nové technológie a nástroje a vplyv na manažment softvérového projektu, január 2006, s. 37-44.

oblastiach. Jednou z oblastí sú riziká vyplývajú z nasadenia nových technológií a nástrojov do procesu vývoja softvéru.

Vývoj softvéru je mladá a prudko sa rozvíjajúca disciplína, v ktorej dochádza častým zmenám, ktoré ju zvyčajne posúvajú vždy o krok dopredu. Tieto zmeny sa najčastejšie týkajú vzniku nových technológii a nástrojov. Za posledné polstoročie možno za najvýznamnejšiu zmenu považovať postupný prechod od programovacích jazykov strojovej úrovne k jazykom štvrtej generácie. Aktuálne možno spomenúť najmä veľký boom jazyka Java, platformy J2EE a špecifikácie WebService.

Nové technológie a nástroje v projekte

Z projektového hľadiska sa však slovné spojenie „nová technológia“ chápe trochu inak. Ak máme vývojový tím pracujúci výhradne v jazyku C, bude pre nás realizácia projektu v jazyku C++ predstavovať použitie novej technológie, hoci jazyk C++ je tu už dávno. Ak sme zvyknutý vyvíjať aplikácie s grafickým užívateľským rozhraním pre systém Windows, bude pre nás vývoj grafického rozhrania aplikácie pre systém Linux dotyk s novou technológiou. Ak sme doteraz vytvárali programy písaním zdrojového kódu a prejdeme na čisto vizuálnu tvorbu programu, bude to pre nás predstavovať použitie nového nástroja.

Vidno teda, že na pojem „nová technológia“ sa dá nazerať dvoma spôsobmi. V prvom ho treba chápať ako niečo nové v rámci celého kontextu vývoja softvéru. V druhom ako technológiu, s ktorou doteraz nemáme v našich projektoch žiadnu skúsenosť. Pritom je zrejmé, že ak je technológia úplne nová, zvyčajne s ňou ani nemáme žiadne skúsenosti. Prvý prípad teda zvyčajne implikuje druhý. Ďalej sa budeme zaoberať hlavne druhým prípadom, nakoľko tento je omnoho častejší.

Ak ideme riešiť projekt a nie sme niečím obmedzovaný, zvyčajne si zvolíme také technológie, s ktorými máme dostatok skúseností. To znamená, že máme ľudí, ktorí tieto technológie ovládajú a majú potrebné know-how. Zvyčajne už pre ne máme pripravené potrebné vývojové a testovacie prostredia, vypracované metodiky (napr. pravidlá písania kódu), testovacie stratégie a vlastné podporné prostriedky. V dôsledku množstva skúseností už dokážeme robiť dobré odhady, čím sa výrazne znižujú tri hlavné spomínané riziká. Lebo riziko vyjadruje vlastne mieru neurčitosti výstupu akcie a teda čím viac informácií máme, tým je riziko menšie.

Často krát sa však stane, že musíme použiť technológiu alebo nástroj, s ktorým nemáme žiadne skúsenosti. Prečo je to tak ?

Dôvody použitia nových technológií a nástrojov

Dôvodov použitia alebo prechodu na novú technológiu alebo nástroj môže byť viacero. Najčastejšou príčinou je priama požiadavka zákazníka na použitie konkrétnej technológie. Zákazník môže mať na to viacero dôvodov.

Najčastejším je, že zákazník má svoje ostatné systémy postavené na danej technológii a teda má pre ňu už vytvorenú potrebnú infraštruktúru. Pod infraštruktúrou

sa chápu v tomto prípade najmä existujúce servery (aplikačné, databázové, ...), nakúpené licencie alebo uzavreté strategické partnerstvá s dodávateľmi danej technológie. Tiež môže takáto požiadavka súvisieť so strategickými plánmi firmy do budúcna, alebo jednoducho s dobrou skúsenosťou zákazníka s danou technológiou. Občas sa tiež stane, že zákazník požaduje jej využitie jednoducho preto, že je momentálne populárna a jemu to zvyšuje imidž.

Iným dôvodom na použitie novej technológie alebo nástroja je to, že si to vyžaduje riešený problém sám zo svojej podstaty. Príkladom sú napr. webové aplikácie, ktoré si vyžadujú použitie iných technológii ako pri vývoji desktopových aplikácii.

Ďalším dôvodom môže byť závislosť na tretej strane. Ak sú napríklad treťou stranou dodávané softvérové súčiastky realizované ako COM objekty, musíme použiť technológiu, ktorá nám s nimi umožňuje pracovať.

Ak sa pozrieme na tento problém z hľadiska biznisu, tak hlavným dôvodom je potreba flexibility firmy z hľadiska používaných technológií a nástrojov. Ak firma nemá alebo nechce mať takúto flexibilitu, prichádza o veľké množstvo potenciálnych zákaziek. Tým sa rapídne redukuje jej zisk, množstvo zákazníkov a dôsledkom môže byť až zánik firmy. Schopnosť adaptovať sa na nové technológie je teda nevyhnutnou vlastnosťou firmy, ktorá chce prežiť na súčasnom trhu.

Riziká použitia nových technológií a nástrojov

V úvode boli spomenuté tri hlavné riziká, ktoré sa snaží každý manažér eliminovať. Tieto riziká spolu úzko súvisia a nemožno ich striktne oddeliť. Prekročenie časového harmonogramu zvyčajne znamená aj prekročenie rozpočtu projektu. Ak chceme eliminovať riziko, že projekt nebude poskytovať požadovanú funkcionalitu, zvyčajne dôjde k prekročeniu časového harmonogramu.

Aby sme však mohli riziká analyzovať, musíme ich istým spôsobom kategorizovať a spomenuté členenie sa ukazuje ako pomerne vhodné. Každé z týchto rizík má svoje príčiny a tiež dôsledky, ktoré vyplývajú z jeho naplnenia.

Prekročenie plánu

Hlavným dôvodom je, že vývoj postupuje pomalšie, ako sme naplánovali. Je to dôsledok toho, že vývojový tím sa musí novej technológii „priúčať“ priamo počas vývoja a teda vývoj produktu napreduje pomalšie. Zvyčajne sa tiež vyskytuje väčšie množstvo chýb, ktoré treba opravovať. Občas sa stane, že vývoj zastane na probléme, s ktorým sa vývojový tím nevie okamžite vysporiadať. Riešenie musí zisťovať napr. prieskumným prototypovaním, čo však zaberá čas. Takisto môže dôjsť k potrebe opravy väčšieho množstva kódu v dôsledku neskoršieho zistenia nesprávneho pochopenia filozofie technológie alebo použitia neoptimálnych prístupov v doterajšom vývoji. S novou technológiou zvyčajne súvisí aj používanie nového nástroja. Adaptácia vývojárov na nové vývojové prostredie tiež trvá istú dobu. Až po tejto dobe ho vedie využívať naplno.

Dôsledky prekročenia plánu môžu byť rôzne a ich závažnosť závisí hlavne od konkrétneho kontextu projektu. Ak má zákazník pripravenú alebo už rozbehnutú marketingovú kampaň súvisiacu s vyvíjaným projektom, budú dôsledky omeškania ukončenia projektu určite väčšie, ako keď sa jedná o projekt pre internú potrebu vyvíjajúcej firmy.

Dôsledky vyplývajúce z prekročenia plánu možno rozdeliť na priame a nepriame. Priamym dôsledkom je zvyčajne penalizácia zo strany zákazníka, čo znamená zníženie plánovaného zisku z projektu a pri väčších omeškaniach môže viesť až k stratovosti projektu. Takáto penalizácia je v súčasnosti zapracovaná v takmer každej dodávateľskej zmluve. Nepriamym dôsledkom je strata dôvery zákazníka a tým aj strata možných budúcich zákaziek, čo opäť vedie k zníženiu zisku firmy. Môže tiež dôjsť k lavínovému efektu, kedy ľudia, ktorí pracujú na projekte boli plánovaní v danej dobe už na iné projekty a v dôsledku toho dôjde k oneskoreniu týchto projektov. Takúto lavínu treba vedieť zastaviť v jej počiatočných štádiách, inak sa firma môže dostať do závažných problémov.

Prekročenie rozpočtu

Prekročenie rozpočtu súvisí hlavne so snahou eliminovať riziko prekročenia plánu. Ak projekt postupuje pomalšie ako by mal, zvyčajným manažérskym riešením je pridanie ďalších ľudí do projektu. Vieme, že nie vždy je to najšťastnejšie riešenie. Ak však aj pomôže, nových ľudí treba zaplatiť. Zvyčajne je však už za behu projektu ťažké zohnať ľudí a často krát je si potrebné vypomôcť outsourcovanými zamestnancami, ktorí sú zvyčajne niekoľko krát drahší ako vlastní zamestnanci.

Ak sa v dôsledku veľkých problémov na projekte rozhodneme využiť služby externých konzultantov alebo konzultačných firiem, opäť musíme na to vynaložiť značné množstvo finančných prostriedkov.

Dôsledkom prekročenia rozpočtu je zníženie zisku z projektu až jeho prípadná stratovosť. Najmä pri malých firmách môže takéto niečo výrazne zakolísať s existenciou celej firmy. V neposlednom rade nás to môže stáť naše miesto projektového manažéra vo firme.

Nesplnenie požadovanej funkcionality

Ak nie je možné z nejakých dôvodov hýbať s termínom ukončenia projektu, zvyčajne musíme najviac manažovať riziko, že vytvorený produkt nebude poskytovať požadovanú funkcionalitu. Príčin môže byť viacero. V priebehu projektu sa môže ukázať, že realizácia niektorých funkcií je s použitím danej novej technológie omnoho obtiažnejšia, než sa predpokladalo. Ak sa tak stane pri konci projektu, zvyčajne už nie je dosť času na ich realizáciu. Občas sa tiež stáva, že požadovaná funkcionalita jednoducho nie je s danou technológiou realizovateľná. To sa samozrejme zvyčajne zistí, až keď už je podpísaná zmluva a projekt je v plnom prúde.

Dôsledkom nesplnenie požadovanej funkcionality môže byť v najhoršom prípade až nepodpísanie preberacieho protokolu. To znamená, že zákazník nám za projekt nezaplatí a celý jeho vývoj bol realizovaný z našich zdrojov. V lepších prípadoch zákazník chápe dodaný produkt ako neúplný a vyplatí iba časť peňazí za projekt.

V dôsledku nekompletne dodaného produktu môžeme získať v očiach zákazníka nálepku amaterizmu.

Vyhodnotenie rizík

Spomenuté riziká sú veľmi veľké a ako bolo naznačené, ich naplnenie môže pre firmu znamenať veľké problémy. Navyše sa v našom prostredí, na Slovensku, šíria informácie veľmi rýchlo. O neúspechu projektu sa zvyčajne veľmi rýchlo dozvedia aj naši ostatní aktuálni ale aj potenciálni zákazníci. Preto musí manažér, hneď ako sa prijme rozhodnutie využívať na projekte novú technológiu, analyzovať z toho vyplývajúce riziká a hľadať spôsoby ich eliminácie.

Eliminácia rizík

Na elimináciu spomínaných rizík je potrebné zapojiť viacero nástrojov a vhodne ich koordinovať. Výber konkrétnych nástrojov a mieru ich použitia musí zvážiť, po konzultáciách s technologicky vyspelými zamestnancami alebo externými konzultantmi, projektový manažér. Ten tiež musí zabezpečiť ich podporu a používanie všetkými zúčastnenými.

Školenia

Prvý prostriedok, ktorý každého hneď napadne, sú školenia. Školenia pre danú technológiu môžu zamestnancom poskytnúť základné informácie o nej a tvoria odrazový mostík pre jej ďalšie spoznávanie. Ak sa jedná o projekt dlhšieho trvania, zvyčajne je vhodné naplánovať sériu školení s postupne odbornejšími témami.

Otvorene treba povedať, že úroveň školení je rôzna. Často krát sa zamestnanec vráti zo školenia, za ktoré si školiaca organizácia vypýtala veľké peniaze, len málo obohatený alebo dokonca zmätený. Často sa stáva, že na školení je veľmi veľký priestor venovaný marketingovým informáciám, ktoré sú však pre tím takmer nepodstatné. Tento fakt si musia uvedomiť manažéri, ktorý zvyčajne chodia na školenia typu „asertivita“ alebo „ako karhať zamestnancov“, pri ktorých je dosť ťažko posudzovať kvalitu. Preto, najmä pri plánovaní série školení, je veľmi vhodné získavať spätnú väzbu od zamestnancov, aby bolo využitie školení čo možno najefektívnejšie.

Ďalším častým problémom je, že manažéri sa mylne domnievajú, že ich pracovníci sa zo školenia vrátia ako experti na danú technológiu. To býva často aj dôvod, prečo sa pracovníci neradi nechávajú školiť na nové technológie. Ak sa vyskytne akýkoľvek problém, manažér zvyčajne očakáva, že ho bude vyškolený zamestnanec vedieť riešiť, hoci ten sa s daným problémom nikdy na školení nestretol.

Vidíme teda, že len samotné školenia nepostačujú, hoci sú veľmi dôležitým prvkom.

Najatie nových zamestnancov

Ak vieme, že budeme realizovať projekt pomocou novej technológie a naše zdroje sú naplno vyťažené, je vhodné prijať nových zamestnancov, ideálne ešte pred začatím projektu. Pritom je potrebné hľadať takých ľudí, ktorí majú s danou technológiu bohaté alebo aspoň nejaké skúsenosti a majú veľkého tímového ducha. Takýchto ľudí je potom vhodné použiť na programovanie kritických vecí, pri ktorých sa vyžaduje veľké množstvo praktických skúseností. Takisto ich môžeme použiť na realizáciu interných školení a tým preniesť ich know-how na našich zamestnancov.

Najatie nových zamestnancov bez skúseností zvyčajne neprináša až tak veľký úžitok, najmä ak ich prijímame do už rozbehnutého projektu. Snaha riešiť problém „hrubou silou“, dodávaním nových ľudí do projektu, zvyčajne pri vývoji softvéru nevedie k úspechu.

Najatie externých konzultantov

Jedným z najvýznamnejších nástrojov boja proti rizikám, vyplývajúcim z použitia novej technológie, je najatie externých konzultantov alebo konzultačných firiem. Aj keď je tento nástroj veľmi drahý, pri dobrom použití je jeho prínos pre firmu omnoho väčší ako vynaložené náklady.

Počet konzultantov závisí hlavne od veľkosti tímu. Spočiatku je pritom dobré, ak je konzultant prítomný na pracovisku stále, aby mohol okamžite zodpovedať a riešiť „nováčikovské“ problémy. Neskôr, keď už stála prítomnosť nie je potrebná, môže prichádzať na pravidelné stretnutia, kde odpovedá na otázky a rieši problémy, ktoré vznikli od posledného stretnutia. Intervaly stretnutí sa môžu postupne zväčšovať, rozhodujúca je však potreba tímu.

Technologické stretnutia

Pravidelné technologické stretnutia predstavujú ďalší nástroj na efektívne šírenie know-how v rámci firmy. Ide o stretnutia, kde sa spoločne riešia problémy alebo sa ukazujú riešenia problémov. Najčastejšie pritom expert ukazuje žiadané riešenia problémov, zamestnanec predvádza, ako vyriešil problém alebo sa spoločne hľadá riešenie problému. Tieto stretnutia sú veľmi dôležité, nakoľko zamestnanci pri novej technológii často narážajú na rovnaké alebo podobné problémy. Ideálne je, ak sa tieto stretnutia realizujú spočiatku na dennej báze, neskôr sa tento interval môže predĺžiť, zohľadňujúc pritom potreby tímu. Počas prvých týždňov vývoja sú dokonca možné okamžité stretnutia (flash meeting), ktoré sa realizujú hneď, ako sa narazí na významný problém.

Báza znalostí

Báza znalostí slúži na zhromažďovanie nadobudnutých poznatkov, ktoré spolu vytvárajú firemné know-how. Zamestnanci sem zapisujú informácie, popisy riešení problémov, postupy, útržky kódov, ktoré nadobudli alebo vytvorili a mohli by sa zísť ostatným členom tímu. Tým dochádza k šíreniu know-how medzi členmi tímu. Tu zapísaným poznatkom nehrozí, že ich zamestnanec zabudne a tým sa úplne stratia. Pri

odchode zamestnanca z firmy sa týmto spôsobom pre firmu uchová značná časť jeho znalostí. Do bázy znalostí tiež môžu byť zapisované všeobecné informácie alebo napr. dohodnuté pravidlá.

V takomto systéme musí byť hlavne možné jednoducho vyhľadávať, aby bolo možné ľahko nájsť požadovanú informáciu alebo zistiť, že sa takáto informácia v báze znalostí nenachádza. Tiež musí byť jednoduché zadávať nové informácie a prepájať ich už s existujúcimi. Systém, ktorý takúto funkcionalitu poskytuje je napr. wikipedia. Môže sa však jednať aj o vlastnú aplikáciu, zvyčajne postavenú nad databázou. Prinajhoršom pomôže aj vhodne štruktúrovaný Word dokument.

Literatúra a zdroje

Vývojovému tímu je potrebné zabezpečiť rýchly prístup k zdrojom informácií, ktoré môžu pri vysporiadavaní sa s novou technológiu potrebovať.

Veľké množstvo informácií je možné v súčasnosti nájsť na Internete, kde existujú dokonca celé webové sídla špecializované na riešenie problémov v konkrétnej technológii. Zvyčajne sú realizované formou diskusného fóra, kde užívateľ môže zadať otázku a iný užívatelia mu môžu na ňu odpovedať. Často krát je možné nájsť riešenie problému bez zadávania otázky, nakoľko sa už skôr na tento problém niekto pýtal a dostal odpoveď. Preto, ak aj firma má vnútornú stratégiu, ktorá neumožňuje zamestnancom prístup na Internet, je ju v tomto prípade vhodné prehodnotiť.

Takisto je dobré zabezpečiť pre tím dostatok kvalitnej odbornej literatúry. Z nej je zvyčajne možné systematicky naštudovať potrebné oblasti. Veľmi vhodné je, ak je literatúra (aj) v elektronickej podobe, nakoľko v tom prípade je v nej možné aj veľmi rýchlo vyhľadávať potrebné informácie.

Zohľadnenie v pláne

Veľmi dôležité je zohľadniť to, že sa bude používať nová technológia už v projektovom pláne. Je potrebné odhadnúť dobu potrebnú na realizáciu jednotlivých činností, čo je však vzhľadom na to, že nie sú žiadne doterajšie skúsenosti pomerne ťažké. Tu môže pomôcť experti z externého prostredia. Navyše treba rátať s tým, že vývoj bude spočiatku výrazne spomalený potrebou zamestnancov naučiť sa a prispôsobiť novej technológii. Do plánu musíme tiež započítať dobu potrebnú na realizáciu jednotlivých tu spomínaných nástrojov (realizácia školení, technologických stretnutí, ...).

Záver

Použitie nových technológií a nástrojov v softvérovom projekte prináša tiež nové riziká. Tieto súvisia najmä so schopnosťou tímu rýchlo sa adaptovať a preniknúť do novej technológie. Dobrý projektový manažér musí odhaliť tieto riziká už na začiatku projektu a rozhodnúť o uskutočnení opatrení a použití nástrojov na ich eliminovanie. Výber týchto nástrojov, miera ich uplatňovania a ich vzájomná koordinácia patrí k schopnostiam, ktoré projektový manažér zvyčajne nadobudne dlhodobou

skúsenosťou. O to dôležitejšie je pre mladých projektových manažérov štúdium týchto skúseností a ich možnosti uplatnenia v nimi riadených projektoch.

Použitá literatúra

1. Thomas, R.: Industry Experience in Migrating to Object Technology. *Technology of Object-Oriented Languages and Systems, 1998* 65-75
2. Lam, W., Vickers, A.J.: Managing the Risk of Component-Based Software Engineering. *5th International Symposium on Assessment of Software Tools,* 1997 123-133.

Annotation

New technologies and tools and impact on software project management

To survive on the contemporary market, a software company has to master a process of transition towards the use of new technologies and tools. This need is consequence of constant emergence of new technologies, which must be used in the development process, either they are required by the problem we deal with in the project, or are required by customer. Management of this process is not trivial. There are many risks associated with it. To eliminate them, we need to apply specific procedures and tools. This essay deals with causes of need of new technologies and tools and main risks, which are associated with transition towards using them. Later on, it presents particular management tools, that project manager can use to deal with risks.

Manažment rizík v softvérovom projekte

ANDREJ JANŽO

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
janzo00@student.fiit.stuba.sk

Abstrakt. Analýza a manažment rizík v softvérových projektoch je stále na okraji záujmu manažmentu. Vysoký pomer neúspešných projektov dokazuje, že manažment by mal tento nezodpovedný prístup k rizikám vo vlastnom záujme prehodnotiť. Táto práca sa venuje manažérskemu systému ProRisk, ktorý popisuje postup pri manažmente rizík v softvérových projektoch a pokrýva celý životný cyklus projektu. Práca prechádza jednotlivými fázami systému pri tvorbe modelu rizík projektu, od identifikácie rizík, cez konštrukciu modelu, jeho kalibráciu, výpočet rizikových hodnôt, až po vytvorenie akčných plánov na zamedzenie výskytu rizík a monitorovací a udržovací proces systému. Následne sa venuje zhodnoteniu systému a jeho výhod a nevýhod pre manažérov a organizáciu.

Úvod

V súčasnej dobe nazývanej aj informačnej, sa vyvíja veľké množstvo softvérových produktov. Napriek tomu, že sa softvér vyvíja už niekoľko desaťročí, manažment sa rizikami nezaoberá dostatočne. Podľa vykonaných prieskumov len jedna šestina všetkých softvérových projektov bola dokončená načas a v rámci plánovaného rozpočtu, jedna tretina projektov bola zrušená a vyše polovica bola odmietnutá [6]. Štúdia [3] ukázala, že 35 % zrušených projektov bolo vo fáze implementácie. Tento veľmi zlý pomer neúspešnosti projektov bol hlavne z dôvodu, že manažment sa nezaoberal varovaniami pred rizikami [5].

Na základe týchto prieskumov je zrejmé, že práca manažmentu bola veľmi slabá v oblasti analýzy a manažmentu rizík. Dobrá analýza a manažment rizík by mala byť základnou požiadavkou na manažment v každom väčšom softvérovom projekte. Väčšina softvérových projektov sa uskutočňuje v nepredvídateľnom prostredí, v ktorom existuje veľa pascí, ktoré môžu ovplyvniť úspešný výsledok projektu. Poprojektové analýzy odhaľujú, že veľa problémov, ktoré sa vyskytli boli v skutočnosti predpovedateľné.

Projekt sa považuje za úspešný, ak spĺňa požiadavky (merané funkcionalitou, spoľahlivosťou, udržovateľnosťou, prenositeľnosťou, efektívnosťou, integráciou a prevádzky schopnosťou), je dodaný na čas a v rámci rozpočtu [7].

Manažment rizík v softvérovom projekte, január 2006, s. 45-54.

Manažment rizík sa zaoberá identifikáciou, čo môže ísť zle a ako môžu tieto prípady dopadať na projekt. Formálna analýza rizík je založená na pravdepodobnostnej teórii. Väčšina softvér vyvíjajúcich projektov zahrňuje veľký počet rizikových faktorov. Na rozdiel od triviálnych malých prípadov, je extrémne ťažké spraviť detailný odhad potrebných pravdepodobností. Preto je veľmi náročná úloha vymyslieť vhodnú metodológiu, ktorá môže byť aplikovateľná na praktické problémy.

Vývoj softvéru má relatívne krátku históriu skúseností na ktorých by sa dalo stavať. V iných disciplínach, starších (špeciálne väčšinou tradičné inžinierske oblasti), majú dobre zavedenú bázu znalostí, praktické manuály a profesionálne štandardy. Softvérové inžinierstvo, ako disciplína, ešte stále len prichádza k týmto termínom s otázkami. Ako výsledok toho je stále potrebné sa učiť ako majú byť riziká modelované a riadené v softvérových projektoch.

Riziko v softvérovom projekte

Riziko znamená možnosť utrpieť stratu, poškodenie, nevýhodu alebo zničenie. Riziko sa samozrejme týka neistoty vzniku známych prípadov, ale aj prípadov, ktoré nie sú na počiatku identifikované ako dopadajúce na projekt [4]. Manažment rizík sa preto musí vyvíjať a učiť, adaptujúc sa na nové meniace sa znalosti ako postupuje projekt.

Hodnota rizika je zvyčajne definovaná ako efekt dopadu rizikovej udalosti a pravdepodobnosť, že udalosť nastane. Môžeme ho vyjadriť nasledovným vzorcom

$$V(A) = P(A)*C(A) \tag{1}$$

kde P(A) je pravdepodobnosť, že nastane udalosť A, C(A) vyjadruje cenu dopadu alebo efektu rizika a V(A) je hodnota rizika pre udalosť A. Hodnota rizika je interpretovaná ako priemerná očakávaná strata nastania udalosti. Z takejto formy vyjadrenia rizika môžeme intuitívne prísť na to, že hodnota rizika je vyššia ak je vyššia pravdepodobnosť alebo cena dopadu. Ako výsledok, riziká s nízkou cenou dopadu ale vysokou pravdepodobnosťou môžu mať rovnakú hodnotu, ako riziká s vysokou cenou ale nízkou pravdepodobnosťou.

Pre malý počet rizikových udalostí je tento model dostatočne jednoduchý a poskytuje užitočnú podporu pre hodnotenie rizikových udalostí. Ako rastie komplexnosť je nutné zvážiť nezávislosť udalostí a kombinovať tieto udalosti na umožnenie efektívneho manažmentu, odhadu a riadenia. Musí sa zistiť veľký počet pravdepodobnostných hodnôt alebo rozdelenie pravdepodobností od manažmentu tímu. Tiež je potrebné odhadnúť podmienené pravdepodobnosti medzi udalosťami, dané tým, že niektoré nebudú úplne nezávislé od ostatných. Takáto úloha je veľmi komplexná, ak nie nemožná pre praktické projekty.

Ak je možné spraviť niektoré predpoklady o podmienených pravdepodobnostiach medzi rizikovými faktormi, potom je možné navrhnúť niektoré funkčné stratégie.

Napríklad, kombinovaný dopad skupiny rizikových faktorov môže byť odhadnutý pomocou vzorca

$$V_I = \sum_{i \in I} w_i \cdot V_i \tag{2}$$

pre prípady, ktoré sú v podstate nezávislé, alebo

$$V_I = \prod_{i \in I} w_i \cdot V_i \tag{3}$$

pre prípady, ktoré sú v podstate navzájom vylučujúce sa, alebo

$$V_I = \max_{i \in I} \{w_i \cdot V_i\} \tag{4}$$

pre fuzzy model zjednotenia následkov udalostí. Váhový koeficient w sa berie ako rizikový efekt vyskytnutej udalosti.

Model rizík v softvérovom projekte

Taxonómia rizík softvérových projektov bola vyvinutá Inštitútom softvérového inžinierstva (SEI – Software Engineering Institute). Táto taxonómia definuje stromovú štruktúru hierarchie rizikových oblastí, náležite klasifikovanú na definovanie klastrov rizikových faktorov. Najvyššie tri levely sú zobrazené v tabuľke 1. Štvrtý level odpovedá skupine otázok, ktoré vyšetrujú potencionálne riziko a tak definujú rizikové faktory pre model. Príspevky každého klastra rizikových faktorov projektu môžu byť odhadované akumuláciou rizikových hodnôt na vrchole stromu. Relatívne príspevky každého faktora a klastra sú opísané váhovými faktormi, ktoré merajú každý efekt zvlášť. Tieto váhové koeficienty boli odhadované z veľkého počtu prípadových štúdií a môžu poskytovať štartovací bod pre projektový rizikový model.

Root Node [Level 0]	[Level 1]	[Level 2]
Project Risk	Sofware Development Risk	Requirements
		Design
		Code and Unit Test
		Integration and Test
		Engineering Specialities
	Development Environment	Development Process
		Development System
		Management Process
		Management Methods
		Work Environment
	Program Constraints	Resources
		Contract
		Program Interfaces

Tab. 1 SEI Taxonómia

Taxonómia SEI je komplexná a potrebuje detailnú analýzu projektu a organizácie prevádzky. Z tejto analýzy procesov sa získajú potrebné detailné poznatky a hodnotenia rizík.

D.W.Karolak [2] navrhol viac formálny model, podobný taxonómii SEI, ale poskytujúci viac kvantitatívny a viac detailný prístup. Tento SERIM (Software Engineering Risk Management) model ponúka manažmentu rizík štyri prepojené stromy založené na 81 rizikových faktoroch. Tie odkazujú na rizikové perspektívy, ktorého tri najvyššie úrovne sú v tabuľke 2.

Rovnako ako SEI model, štvrtá úroveň je vyjadrená sadou otázok, ktoré vyšetrujú rizikové faktory podľa pravdepodobnosti výskytu. Rizikové faktory sú zdieľané vo všetkých perspektívach, rozdiely sú len vo váhových faktoroch. SERIM model je oproti SEI modelu komplexnejší, sú to v skutočnosti štyri modely, každý zodpovedajúci inej perspektíve.

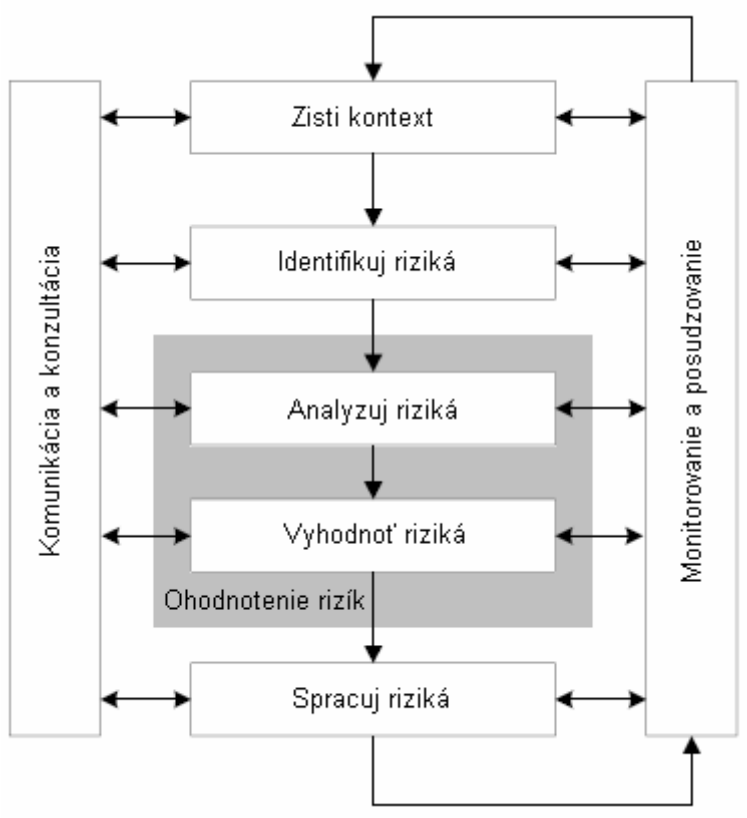
SERIM model potrebuje od používateľa (manažéra), aby poskytol odhady pravdepodobností pre každý rizikový prípad. Následne sa spočíta kombinovaná riziková hodnota pre každú úroveň stromu. Rizikový manažment je potom schopný porovnať príspevky rizík pre každú časť stromu medzi perspektívami a identifikovať kľúčové rizikové faktory.

Root Node [Level 0]	[Level 1]	[Level 2]
Risk Elements	Technical	(a) Organization
		(b) Estimation
		(c) Monitoring
		(d) Development Methodology
		(e) Tools
		(f) Risk Culture
		(g) Usability
		(h) Correctness
		(i) Reliability
		(j) Personnel
	Cost	(a) to (j) as above
	Schedule	(a) to (j) as above
Risk Categories	Process	(a) to (j) as above
	Product	(a) to (j) as above
Development Phases	Pre-requirements	(a) to (j) as above
	Requirements	(a) to (j) as above
	Design	(a) to (j) as above
	Coding	(a) to (j) as above
	Testing	(a) to (j) as above
	Delivery and Maintenance	(a) to (j) as above
Risk Activities	Identification	(a) to (j) as above
	Strategy and Planning	(a) to (j) as above
	Assessment	(a) to (j) as above
	Mitigation and Avoidance	(a) to (j) as above
	Reporting	(a) to (j) as above
	Prediction	(a) to (j) as above

Tab. 2. SERIM rizikové perspektívy

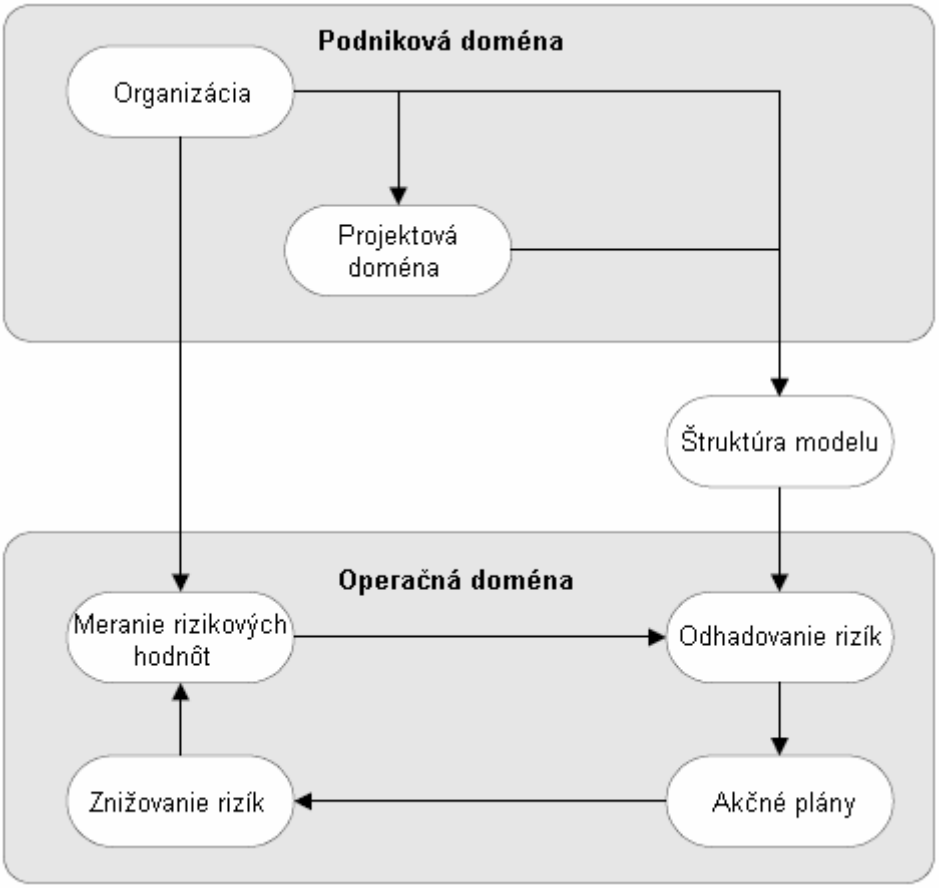
Systém pre manažment rizík

Manažment rizík musí byť integrovanou súčasťou štruktúry projektového manažmentu, ak má byť efektívny. Kroky zobrazené na obrázku 1 musia byť obalené v procese, ktorý náležite identifikuje roly a zodpovednosti medzi organizáciou a projektovým týmom.



Obr. 1. Systém manažmentu rizík

ProRisk manažérsky systém postupuje podľa týchto krokov a rozširuje ich. Schéma tohto systému v kontexte typického softvérového projektu je na obrázku 2. Tento systém sa zameriava na primárne zložky projektu, podnikovú doménu a operačnú doménu. Podniková doména poskytuje možnosti na identifikáciu ekonomického prostredia, vedomostí a skúseností organizácie v ktorej sa projekt vyvíja. Operačná doména obsahuje modely na meranie, identifikovanie, odhadovanie, opisovanie a implementovanie plánov a obaľuje ich do spojitých cyklických procesov, ktoré majú byť aplikované počas projektu.



Obr. 2. Manažérsky systém ProRisk

Identifikácia podielnikov (stakeholders)

Podielnici v projekte sú jednotlivci, skupina ľudí alebo organizácií, ktoré majú úžitok z výsledku projektu (a dedukciou, utrpia stratu, ak projekt zlyhá). Majú jednu alebo viac perspektív a tieto perspektívy sú merané v ich vlastných jednotkách. Na efektívnu reprezentáciu ich záujmov, metodológia musí podporovať aj viac podielnikov aj rôzne spôsoby vyjadrenia záujmu.

Identifikácia rizikových faktorov

Identifikácia podielnikov pomáha v identifikácii rizikových faktorov, ale aj tak je potrebné vedieť viac informácií. Sami podielnici, nie sú schopní vytvoriť kompletnú sadu rizikových faktorov, ktoré opíšu ich vlastnú perspektívu záujmu. Cieľom je identifikovať všetky veci, ktoré sa môžu zle vyvíjať a toto je úloha pre rizikový manažment. Na zistenie kompletnej sady rizikových faktorov je možné začať od nuly

a použiť brainstorming alebo Delphi metódu. Toto môže byť proces, ktorý pravdepodobne spotrebuje veľa času, ale v niektorých prípadoch to môže byť oprávnené. Získaním sady rizikových faktorov je rozumné začať s jedným z modelov a prispôsobiť ho potrebám vlastného projektu.

Konštrukcia modelu stromu rizík

Konštrukcia kompletného modelu stromu rizík je iteratívny a učiaci proces vyžadujúci zainteresovanosť všetkých podielnikov a manažmentu rizík, ktorí budú zodpovední za celkový model. Z počiatočného zoznamu rizikových faktorov, je ako prvé potrebné vytvoriť rizikové klastre. Všetky rizikové faktory v klastri by sa mali týkať podobných aspektov projektu. Kompletný strom rizík bude vytvorený ďalšou agregáciou klastrov pokiaľ nebude mať každá perspektíva jeden koreňový uzol, reprezentujúci celý projekt. Kompletný projekt môže byť reprezentovaný jedným alebo viac stromami alebo perspektívami.

Kalibrácia modelu

Kalkulácia rizikovej hodnoty vyžaduje pravdepodobnosti rizikových udalostí a ich cena/dopad pre každú perspektívu ku ktorej patrí. Získavanie týchto hodnôt nie je triviálne. Kalibrácia modelu začína odhadovaním váhových faktorov. Každá riziková perspektíva musí byť spracovaná samostatne:

Cenová perspektíva: váhové faktory budú napríklad v eurách a budú vyjadrovať priemerné zvýšenie nákladov projektu, ak prípad nastane.

Rozvrhová perspektíva: váhové faktory budú v časových jednotkách (dni, týždne, atď.) a budú vyjadrovať priemerné zdržanie projektu, ak prípad nastane.

Ostatné rizikové perspektívy (napr. kvalita, reputácia) musia byť tiež merané ak sa použijú.

Odhadovanie pravdepodobností rizikových udalostí

Toto je posledný krok vo fáze tvorby modelu a nutný krok pred tým ako sa použije model na čokoľvek užitočné. Každý rizikový faktor musí byť odhadnutý na jeho pravdepodobnosť výskytu. Obvykle bývajú tieto hodnoty reprezentované v rozsahu 0 až 1. Reprezentácia pomocou slov (XLow, VLow, Low, Med, High, VHigh, XHigh) môže používateľom umožniť myslieť vo viac opisných výrazoch ako číselné vyjadrenie.

Výpočet kombinovaných rizikových hodnôt

ProRisk manažérsky systém poskytuje viacero kombinácií modelu:

- Sumácia – kombinuje rizikové hodnoty sumáciou hodnôt pre každý klaster podľa vzorca (2)
- Súčin – kombinuje rizikové hodnoty súčinom hodnôt na najnižšej úrovni klastra podľa (3) a potom sumáciou najvyšších úrovní podľa (2)

- Maximum – založené na Murphyho zákone, ktorý hovorí, že ak sa niečo pokazí, tak to bude prípad, ktorý má najväčšiu rizikovú hodnotu, berúc tak do úvahy najhorší prípad.

Ak je model vytvorený a nakalibrovaný, môže sa použiť pri identifikácii kľúčových rizikových faktoroch.

Vytvorenie akčných plánov

V ProRisk systéme, akčné plány poskytujú podporu pre dokumentáciu, manažment a na prehodnotenie rizikových prípadov počas vývoja projektu. Akčné plány vyžadujú:

- Delegovanie a zodpovednosť, t.j. kto je zodpovedný za ich vykonanie
- Opis, čo bolo spravené na redukciu dopadov rizikových faktorov.
- Rozmiestnenie zdrojov na umožnenie dokončenia práce.
- Časový rozsah, opisujúci kedy má byť práca dokončená.

Monitorovanie vývoja

Ako projekt postupuje, rizikové hodnoty sa menia z času na čas. Aby bolo možné primerane manažovať projekt, toto prechodné správanie sa musí zaznamenávať, aby bolo možné monitorovať vývoj rizikových vlastností. ProRisk manažérsky systém umožňuje časom vyvíjať model rizík, udržiavať snímky stavu modelu v časových epochách, umožňujúc prechodným zmenám byť zaznamenané a monitorované.

Záver

Softvér vyvíjajúce projekty sú náročné na analýzu rizík, pretože obsahujú široké množstvo rizikových faktorov, ktoré sa dopredu ťažko odhadujú. Väčšina súčasne dostupných nástrojov nie je dostatočne škálovateľná na väčšie problémy alebo sú náročné na množstvo informácií potrebných na nastavenie a kalibráciu modelu.

Ponúknutý ProRisk systém môže byť aplikovaný aj na malé aj na komplexnejšie projekty. Umožňuje používateľom prispôsobiť si model a špecializovať ho pre vlastné potreby. Vyžaduje však tiež detailnú analýzu organizácie a projektovej domény, čo zaberie dosť času a môže byť náročné najmä pre neskúsených, začínajúcich manažérov. Organizácia by preto pri zavádzaní nových postupov a po prijatí nového manažéra mala robiť dôsledné školenia. Na dobrý odhad pravdepodobností a kalibráciu modelu však treba aj určité skúsenosti v danej oblasti, a školenie je preto len začiatkom. Dobré výsledky v odhade rizík sa dajú robiť až po skúsenostiach z viacerých projektoch.

Dobré výsledky ProRisk systému môžu byť zaručené iba za nasledovných predpokladov:

- Rizikové faktory sú nezávislé alebo navzájom sa vylučujúce.
- Na výpočet zložených rizikových hodnôt sa použitie vhodná metóda.

- Rizikové perspektívy sú vytvorené z odpovedajúcich rizikových hodnôt.
- Extrémne prípady sú z modelu vyňaté.

Problém môže byť v prvom predpoklade, keďže nie všetky riziká sú nezávislé a môžu ovplyvňovať iné riziká. S ďalšími troma predpokladmi sa dá súhlasiť, ak použijeme zlú metódu, tak je zrejmé, že dostaneme zlé výsledky, alebo nie je možné v modeloch počítať s extrémnymi prípadmi, ktorých dôsledky by boli katastrofálne ale majú aj extrémne nízku pravdepodobnosť výskytu.

Skúseným manažérom môže ProRisk systém pri manažovaní rizík značne pomôcť, pretože pokrýva kompletný životný cyklus projektu a umožňuje analyzovať riziká a zároveň sa venovať konvenčným projektovým manažérskym aktivitám.

Použitá literatúra

1. Addison, T., Vallabh, S.: Controlling software project risks – an empirical study of methods used by experienced project managers. *Proceedings of SAICSIT*, (2002) 128-140.
2. D. W. Karolak, Software Engineering Risk Management, *IEEE Computer Society*, 1997
3. Ewusi-Mensah, K., Przasnyski, Z.H.: On information systems abandonment: An exploratory study of organisational practise. *MIS quarterly*, Vol. 15, No. 1 (1991) 67-88.
4. Geoffrey G. Roy, A Risk Management Framework for Software Engineeering Practice, *aswec, p. 60, 2004 Australian Software Engineering Conference* (ASWEC'04)
5. Keil, M., Cule, P.E., Lyytinen, K., Schmidt, R.: A framework for identifiing software project risks. *Communication of the ACM*, Vol. 41, No. 11 (1998) 77-83.
6. May, L.J.: Major causes of software project failures. http://stsc.hill.af.mil/crosstalk/1998/jul/causes.asp. 25.6.2002
7. Powell, P.L., Klein, J.H.: Risk management for information systeme development. *Journal of information technology*, Vol. 11 (1996) 309-319.

Annotation

Risk management in software projects

Risk analysis and management in software projects is still on marginal interests of management. High failure rate of projects proves, that management should revaluate this irresponsible access to risk. This paper presents ProRisk Management framework, that describes risk management method in software projects and covers whole life-cycle of project. Paper goes through individual phases of the framework, from risk identification, through model construction, his calibration, computing risk values, till developing action plans to prevent risk occurrence and monitoring and maintaining process of the framework. Thereafter devotes to valuation of this framework and his benefits and disadvantages for managers and organization.

Sledovanie postupu softvérového projektu a manažment

MICHAL JEMALA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
jemala01@student.fiit.stuba.sk

Abstrakt. Mnoho softvérových projektov sa končí odlišne, ako bolo pôvodne naplánované. Bežné je oneskorenie, prekročenie rozpočtu, prípadne nesplnenie požadovanej funkcionality. Sledovanie postupu projektu, môže manažérov upozorniť na prichádzajúce problémy a umožniť im tak prijať včas potrebné opatrenia. Kľúčovým faktorom pri sledovaní postupu projektu je zabezpečiť jeho viditeľnosť a to v ľubovoľnom okamihu riešenia projektu, v najlepšom prípade kontinuálne a automatizovane.

Úvod

Sledovanie postupu softvérové projektu predstavuje integrálnu súčasť procesu referovania o výkone, ktorý predstavuje subproces riadenia softvérového projektu. Pravidelné sledovanie a vyhodnocovanie postupu projektu predstavuje veľmi dôležitý faktor z hľadiska úspešnosti projektu. Niektoré štúdie dokonca uvádzajú, že softvérové firmy v USA premrhali v roku 1995 až 59 mld. dolárov len kvôli predĺženiu projektu a 81 mld. dolárov kvôli úplnému zrušeniu projektu [2]. Je zrejmé, že včasné odhalenie príčin umožní projektovým manažérom prijať potrebné opatrenia, ktoré zmiernia resp. úplne odstránia ich negatívne dôsledky. Nedodržanie plánu nepredstavuje jedinú hrozbu ovplyvňujúcich úspešnosť projektu. Je nutné sledovať aj prekročenie rozpočtu, kvalitu samotného vývojového procesu, efektívnosť použitých metód a prostriedkov a ostatné podporné činnosti projektu.

V tejto rozprave sa budeme venovať sledovaniu softvérových projektov. Predstavíme si zaujímavý model vývoja softvéru, ktorý umožňuje veľmi efektívne sledovať a hlavne prehľadne vizualizovať postup projektu. Ďalej rozoberieme prístup využívajúci pri sledovaní softvérového projektu princípy telemetrie. Na záver si uvedieme príklad sledovania postupu softvérového projektu priamo z akademickej pôdy.

Sledovanie postupu softvérového projektu a manažment, január 2006, s. 55-61.

Princípy sledovania postupu softvérového projektu

Základným princípom pri sledovaní postupu je zabezpečiť viditeľnosť projektu. Teda je nutné čo najpresnejšie určiť aktuálny stav projektu. Stav projektu je možné určovať periodickým zaznamenávaním a následne vhodným vyhodnocovaním špecifických charakteristík projektu. Avšak aké charakteristiky sledovať a ako ich vyhodnocovať, aby sme mohli exaktne určiť stav projektu? Neexistuje jednoznačná odpoveď na túto otázku. Jednotlivé projekty, aj keď sa týkajú rovnakej oblasti (v tomto prípade vývoja softvérových produktov), sú značne rozdielne. Je nutné zohľadňovať aj špecifické aspekty týchto projektov, aby sme mohli zodpovedne vyhodnocovať ich postup. Našťastie existuje viacero overených odporúčaní, ktoré výrazne uľahčia proces sledovania, pre ľubovoľný projekt [1]:

- *Plán projektu.* Je značne komplikované, ak nie nemožné, sledovať postup softvérového projektu, ak nie je stanovený akýsi referenčný model. Tým je v tomto prípade plán projektu, obsahujúci časové i finančné predpoklady. V stanovených bodoch, ktoré definuje samotný plán, je možné vyhodnotiť stav projektu, určiť prípadne odchýlky a prijať potrebné opatrenia.
- *Začiatok a koniec každej činnosti.* Dôkladné definovanie termínov pre jednotlivé činnosti a úlohy, na čo najmenšej úrovni abstrakcie, umožní v pláne identifikovať kritické miesta, ktoré treba obzvlášť dôkladne sledovať (metóda kritickej cesty).
- *Zdroje pre každú činnosť.* Ak sú v pláne dôkladne definované zdroje potrebné pre vykonanie danej činnosti alebo úlohy, a zároveň sa sledujú aj skutočne spotrebované zdroje, je možné stanoviť ukazovatele, umožňujúce nielen presne vyhodnotiť stav projektu, ale aj predpovedať ďalší postup projektu.
- *Výsledky činností.* Azda najťažšou úlohou, je sledovanie výsledkov jednotlivých činností. Výsledkom každej etapy v životnom cykle vývoja softvéru má byť určitý výstup. Ak chceme efektívne sledovať postup, je nutné vyhodnocovať aj tieto výstupy. Nástroje na vyhodnocovanie výstupov jednotlivých etáp v životnom cykle vývoja softvérov poskytuje manažment akosti a meranie v softvérovom projekte.

Sledovanie postupu a modely životného cyklu vývoja softvéru

V [3] sa uvádza, že súčasné modely životného cyklu vývoja softvéru podporujú sledovanie postupu projektu len málo alebo vôbec nie. Zdôrazňujú, že tento proces manažmentu projektu, je podstatný a je vhodné ho zakomponovať priamo do modelu životného cyklu vývoja softvéru. V článku sa demonštruje Rekurzívny multivláknový model (RMT).

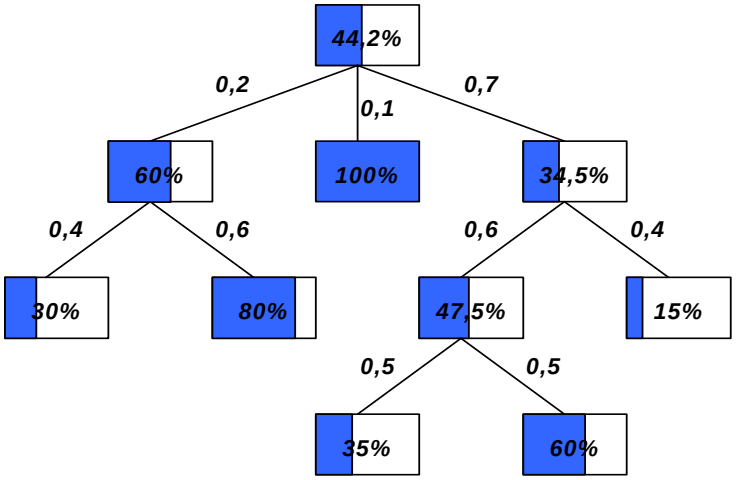
RMT je založený na dekompozícií vyvíjaného softvérového systému do tzv. vlákien. Vlákno predstavuje činnosť – implementáciu softvérového komponentu. Každé vlákno pozostáva z niekoľkých fáz (analýza požiadaviek, plánovanie, analýza,

návrh, implementácia, testovanie). Jednotlivé vlákna sa priradia manažérom a tímom vývojárov, ktorí sú zodpovedný za ich realizáciu. Vlákna je možné jednak dekomponovať a vytvárať tak hierarchiu (rekurzia vlákien), a jednak plánovať ich verzie (iterácia vlákien). Pričom každé vlákno, ktoré je potomkom, má priradenú váhu zodpovedajúcu percentuálnemu podielu úsilia potrebného na vykonanie jeho rodiča (suma váh všetkých potomkov daného rodiča je 100%). Taktiež každá iterácia je ohodnotená percentom z celkového úsilia potrebného na realizáciu daného vlákna. Takto štruktúrovaný projekt je možné následne veľmi efektívne sledovať. Vyhodnotenie stavu projektu má potom 2 fázy:

1. *vyhodnotenie listov* – $Effort^{leaf} = \sum_{j=1}^{k-1} e_j + p \cdot e_k$, pričom $Effort^{leaf}$ – percento vyjadrujúce vykonané úsilie v danom liste, e_j – percento prislúchajúce *j*-tej iterácií, e_k – percento prislúchajúce aktuálne rozpracovanej iterácií, p – percento vyjadrujúce stav aktuálne rozpracovanej iterácie.
2. *vyhodnotenie uzlov* – $Effort^i = \sum_{j=1}^{J} w_j \cdot Effort_j^{i-1}$, pričom $Effort^i$ – predstavuje úsilie vykonané na *i*-tej úrovni, $Effort_j^{i-1}$ – predstavuje úsilie vykonané na (*i*-1)-vej úrovni pre *j*-teho potomka, w_j – predstavuje podiel úsilia potomka potrebného na ukončenie rodiča.

Autori tohto modelu implementovali uvedené koncepty do jednoduchého CASE nástroja s názvom RMT Tool. RMT Tool umožňuje veľmi efektívne a prehľadne zobrazovať postup v softvérovom projekte. Príklad vizualizácie vytvorenej počas riešenia projektu je uvedený na **Obr. 1**.

Každý uzol umožňuje zobraziť doplňujúce informácie (stav prislúchajúcich fáz a zoznam riešiteľov). Informácie, poskytované pomocou nástroja RMT Tool, sú prístupné jednak pre vývojárov, pre projektových manažérov, ale najmä pre tzv. manažérov postupu (angl. progress managers). Manažéri postupu majú na starosti analyzovanie a vyhodnocovanie existujúcich dát (generovanie rôznych reportov) a doplňanie nových dát. Takéto sledovanie a zaznamenávanie postupu projektu umožňuje predpovedať ďalší vývoj, odhadovať ukončenie jednotlivých vlákien a aj celého projektu.



Obr. 1 Vizualizácia stavu softvérového projektu vytvorená pomocou RMT Tool

Uvedený prístup predstavuje zaujímavý spôsob sledovania postupu v softvérom projekte, avšak má podľa môjho názoru niekoľko nedostatkov. Prvým nedostatkom je, že vôbec neuvažuje fakt, že ukončenie všetkých potomkov nemusí znamenať zákonite aj ukončenie prislúchajúceho rodiča. Keď si uvedomíme, že vlákno predstavuje implementáciu softvérového komponentu, potom je jasné, že úspešným implementovaním všetkých komponentov nedostávame hneď aj fungujúci modul alebo podsystém. Analogicky implementovaním všetkých modulov alebo podsystémov nedostaneme hneď fungujúci systém. V prípade softvérových projektov, v tejto situácií, určite neplatí jednoduchá rovnosť 1+1=2. Riešením by mohlo byť naplánovanie a vytvorenie ďalšieho vlákna, ktoré by predstavovalo činnosť spojenú s integráciou prislúchajúcich komponentov, modulov a podsystémov do väčšieho celku. Ďalším nedostatkom je fakt, že výhody takéhoto modelu sledovania postupu softvérového projektu sa strácajú v tom prípade, ak sa aplikuje na rozsiahly projekt. Takýto projekt môže mať tisícky vlákien a uvedený graf postupu by bol enormne rozsiahly. Sami autori priznávajú, že tento model je vhodný pre malé a stredne veľké projekty. Teda mohol by byť využitý napríklad aj pri riešení tímového projektu. Posledný nedostatok vidím v tom, že tento prístup nedáva návod ako sledovať jednotlivé činnosti, aké charakteristiky sledovať a ako ich vyhodnotiť tak, aby sme mohli doplniť do grafu údaj, že toto vlákno je ukončené napríklad na 33%. Pri výpočte vynaloženého úsilia pre listové uzly sa sumujú jednotlivé iterácie, podľa môjho názoru nie je možné dostatočne exaktne dopredu určiť počet iterácií a ešte aj prislúchajúce percento z celkového úsilia potrebného na ukončenie daného vlákna. Okrem toho vo vzorci vystupuje ešte parameter *p* vyjadrujúci postup aktuálne rozpracovanej iterácie. Ako teda určiť skutočne objektívnu hodnotu tohto parametra?

Telemetria v softvérovom projekte

V predchádzajúcej časti sme prestavili model vývoja softvéru založenom na precíznej dekompozícií vytváraného systému a podrobnom pláne, na základe ktorého sa sleduje postup projektu. Pričom uvedený plán sa vytvára väčšinou na základe skúseností z predchádzajúcich projektov a s využitím nejakého zaužívaného modelu. V [4] sa diskutuje prístup so zaujímavým názvom Telemetria v softvérovom inžinierstve. Pod pojmom telemetria sa chápe výrazne automatizovaný proces vzdialeného merania a zhromažďovania dát. Pričom dôraz sa kladie práve na to, aby sa dáta zbierali vzdialene a automatizovane, t.j. aby boli vývojári a vedúci pracovníci odbremenení od periodického ručného zhromažďovania dát. Ďalším kľúčovým faktorom tohto prístupu je okamžitý prístup k dátam vyplývajúci z toho, že dáta sa zbierajú kontinuálne a nie v diskrétnych časových okamihoch. Aké dáta sa vlastne sledujú? Autori uvádzajú nasledovnú kategorizáciu:

1. *telemetria pri vývoji* – sleduje sa počet editovaných súborov, čas strávený v danom vývojovom prostredí, zmeny vykonané v jednotlivých častiach systému (artefakty), počet riadkov kódu a poradie vykonávaných príkazov.
2. *telemetria pri zostavovaní* – sledujú sa výsledky z kompilácie, linkovania a unit testov.
3. *telemetria pri vykonávaní* – sleduje sa správanie systému počas behu (angl. runtime) a výsledky rôznych záťažových testov (angl. load and stress tests).
4. *telemetria pri používaní* – sleduje sa správanie používateľa a jeho interakcia so systémom.

Ako súvisí takáto telemetria so sledovaním postupu v projekte? Telemetria poskytuje, v kombinácií s klasickými metódami, veľmi efektívny nástroj ako včas určiť anomálie a rôzne odchýlky v projekte a podporuje okamžité rozhodovanie o nasledujúcom postupe. Analógiou môžu byť preteky formuly 1. Ak by sa technici a stratégovia stajne dozvedeli požadované dáta o stave monopostu iba počas naplánovanej zastávky v boxoch, mohli by zasiahnuť do monopostu a reagovať tak na špecifické situácie až počas ďalšej zastávky. A vtedy už môže byť neskoro. Avšak vzhľadom na to, že monitorujú monopost kontinuálne, môžu okamžite odhaliť prichádzajúce problémy, obmienať plán alebo predpovedať ďalší priebeh pretekov. Takýto prístup umožňuje veľmi flexibilný spôsob riadenia projektu.

Samotné zbieranie dát sa vykonáva pomocou tzv. senzorov monitorujúcich príslušný program. V aktuálnej verzii systému HackyStat[2], ktorý implementuje tento prístup, existujú senzory pre bežne používané vývojové prostredia ako: Eclipse, Emacs, JBuider, Vim, Visual Studio, pre kancelárske balíky Word, Excel, pre zostavovacie nástroje ako sú Ant, Make, pre monitorovanie práce s CVS a unixovským príkazovým riadkom, a nakoniec aj pre automatizované testovanie pomocou JUnit a meranie pomocou nástroja JMeter. Architektúra systému umožňuje jednoducho

[2] www.hackystat.org

pridávať ďalšie senzory pre špecifické nástroje a tiež definovať aké dáta sa majú sledovať.

Z uvedeného je zrejmé, že takýto prístup vedie k enormnému množstvu nazbieraných dát. Autori systému HackyStat prišli na tento problém počas využívania tohto systému priamo pri jeho vývoji. Riešením bolo vybudovanie kontrolného centra, ktoré zahŕňalo 9 monitorov kontinuálne zobrazujúcich rôzne sledované charakteristiky. Skúsenosť s takýmto využitím telemetrie bola priaznivá, samotní vývojári sa - pri prechádzaní okolo - zastavovali v kontrolnom centre. Mohli sledovať ako napreduje ich projekt a taktiež získavali potrebný nadhľad nad celým projektom. Zaujímavé by bolo sledovať, či by sa takýto sústavný monitoring ujal v externom prostredí, či by pracovníci pracovali vo firme, kde by boli sledovaní pri každej činnosti. Napriek týmto otvoreným otázkam si myslím, že sa jedná o veľmi zaujímavý a perspektívny prístup, ktorý má potenciál ujať sa napríklad v rámci agilných prístupov k vývoju softvérových systémov.

Sledovanie postupu a tímový projekt

Ako zakomponovať uvedené prístupy do riešenia tímového projektu? Prvý prístup, si vyžaduje vytvoriť precízny plán. Po vytvorení podrobného plánu, ktorý zahŕňa dostatočnú úroveň detailov pre jednotlivé činnosti, je možné aplikovať princíp modelu RMT. Využitie telemetrie je menej vhodné. Tímový projekt sa nerieši dostatočne dlhé obdobie a neprebieha tak intenzívny vývoj, aby bolo možné počas jedného semestra využiť jej výhody. Oba prístupy však majú spoločné to, že sa snažia automatizovať sledovanie postupu. Podobný spôsob sa aplikuje aj na Arizona State University v USA. Kde študenti počas bakalárskeho štúdia riešia veľmi podobný projekt ako študenti FIIT v rámci tímového projektu. Na tejto univerzite je vybudovaný čiastočne automatizovaný spôsob vyhodnocovania postupu v projekte, ktorý slúži inštruktorom jednotlivých tímov pri ich kontrole, ale hlavne študentom samotným. Pomáha im sledovať aktuálny stav projektu, na základe ktorého môžu modifikovať ďalší plán. Keďže je možné prehliadať aj progres ostatných tímov, zvyšuje sa zároveň úroveň súťaživosti, čo motivuje študentov k lepším výsledkom.

Nástroj predstavuje webovú aplikáciu, pričom podstatou sledovania postupu je zadávanie týždenných reportov. Tieto sa v systéme spracovávajú a následne sa generujú rôzne štatistiky a prehľady. Napríklad čas strávený pri riešení projektu celkovo, ale aj v jeho jednotlivých fázach alebo čas strávený na tímových stretnutiach. Pričom sa dá vyhodnocovať tím ako celok, ale aj jednotliví členovia tímu.

Uvedený prístup má avšak jednu slabinu. Ako zaručiť, že študent vloží do systému ozaj pravdivé dáta? V tomto mieste sa nám kruh uzatvára, dostávame sa opäť na začiatok. Teda aké objektívne merateľné charakteristiky sledovať a ako ich vyhodnocovať, aby sme mohli exaktne určiť stav projektu? Mohlo by sa zdať, že uvedené prístupy nám neprinesú žiadaný recept. Ale nie je to tak. Môžu výrazne prispieť k zlepšeniu celkového procesu, ale nikdy nemôžu nahradiť ľudský faktor. Dobrý projektový manažér, ktorý participoval pri riešení desiatok projektov, má „zabudovaný“ vlastný senzor, ktorý nadobudol skúsenosťami a určite aj tým, že sa

poučil na vlastných chybách. Neznamená to avšak, že treba zanevrieť na uvedené princípy a postupy, treba ich aplikovať, skúšať a modifikovať, ale s rozvahou.

Záver

Ak sa chceme ocitnúť v pozícií manažéra úspešne zakončeného softvérového projektu, je nevyhnutné kontinuálne sledovať jeho postup, reagovať na rôzne odchýlky a anomálie, modifikovať plán, opäť sledovať, reagovať a modifikovať... Pri tomto procese nám môžu veľmi pomôcť viaceré metódy, odporúčania a nástroje, ktoré automatizujú proces sledovania. Ich výsledky treba vyhodnocovať opatrne a hlavne zo zreteľom na ľudí, ktorých sa týkajú.

Použitá literatúra

1. Bieliková, M.: *Softvérové inžinierstvo*. Princípy a manažment. Vydataveľstvo STU, Bratislava, 2000.
2. Collofello, J. S., Hart, M.: Monitoring Team Progress in a Software Engineering Project Class. In: *29th ASEE/IEEE Frontiers in Education Conference*. November 10 - 13, 1999 San Juan, Puerto Rico.
3. Conception, A. I., Lin, S., Simon, S. J.: The RMT Tool: A Computer Aided Software Engineering Tool for Monitoring and Predicting Software Development Progress. In: *Proceedings of the 21st International Conference on Software Engineering* (ICSE'99).
4. Johnson, P. M., Kou, H., Paulding, M., Zhang, Q., Kagawa, A., Yamashita, T.: Improving Software Development Management through Software Project Telemetry. *IEEE Software*, Vol. 22, No. 4 (2005), pp. 76-85.

Annotation

Monitoring progress of software project and management

Many software projects face difficulties because their monitoring relies too strongly on subjective information from the team members. Deviations from planning are often recognized too late. Automated project monitoring can alert managers on coming problems and allow them to take essential precautions. Therefore the key point during monitoring progress is visualizing real project state automatic and anytime.

Vzťah zákazník a vývojár v softvérových projektoch

LADISLAV KOČIŠ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
kocis01@student.fiit.stuba.sk

Abstrakt. V súčasnosti sa väčšina softvérových projektov končí neúspechom. Je to spôsobené hlavne zlými vzťahmi medzi vývojármi a zákazníkmi, jedná sa hlavne o komunikačné vzťahy. V eseji sú popísané spôsoby komunikácie medzi zákazníkom a vývojármi, ktoré by mohli zvýšiť šancu projektu na úspech.

Úvod

V mnohých literatúrach sa odporúča, aby zákazník spolupracoval pri vývoji systémov. V praxi je táto spolupráca často obmedzená a dokonca v niektorých extrémnych prípadoch chýba úplne. V takých prípadoch musia vývojári pracovať naslepo. Sú známe aj také prípady, kedy vývojári po špecifikovaní požiadaviek na systém pracujú samostatne, bez spolupráce zákazníka. Zákazník uvidí systém až po nejakom čase a to už hotový systém. V takýchto prípadoch je veľká šanca, že zákazník dodaný systém neakceptuje.

V nasledujúcej časti popíšem vzťahy medzi zákazníkom a vývojármi podrobnejšie. Neskôr rozoberiem rôzne výhody a nevýhody, ktoré plynú z takejto komunikácie.

Vzťah medzi zákazníkom a vývojármi

I keď je používateľská zainteresovanosť problematická skoro vo všetkých informačných systémoch [1], situácia sa zhoršuje neustálym vývojom informačných technológií. Rozdielne spoločenské postavenie používateľov/zákazníkov mení vzťahy medzi používateľmi a vývojármi. Kým v tradičných vývojoch informačných systémov bol používateľ na podobnom organizačnom stupni alebo ešte nižšom, nový používatelia majú omnoho vyššie organizačné postavenie vo firmách. Toto má veľmi výrazný dopad na potenciál možnej spolupráce. Väčšina firiem nie je ochotných poskytnúť zamestnanca, ktorý by sa podieľal na vývoji s vývojárskym tímom a to buď

Vzťah zákazník a vývojár v softvérových projektoch, január 2006, s. 63-68.

z neochoty, alebo jednoducho preto, že ten zamestnanec je potrebný pre chod firmy. V takýchto prípadoch majú vývojári sťažené podmienky.

Čo sa myslí pod používateľskou spoluprácou? Používateľská spolupráca je definovaná ako spolupráca pri vývoji informačných systémoch [1], čo je dosť široký pojem. Takáto spolupráca môže mať veľa výhod, používateľ môže pomôcť zlepšiť proces návrhu a pomáhať pri vzniknutých problémoch, ktoré sa vyskytnú počas implementácie. V procese návrhu by sa mohlo jednať hlavne o identifikovanie požiadaviek na systém alebo odobrenie rôznych dizajnových návrhov. V časti implementácie by to mohli byť akceptačné testy, predídenie rôznym konfliktom a neustála podpora.

Mnohé empirické štúdie sa snažili demonštrovať vzťahy používateľskej spolupráce v závislosti od úspechu projektu a vzťahoch s používateľom [2]. Vznikli rôzne metodológie špeciálne navrhnuté tak, aby umožnili používateľovu spoluúčasť na projektoch. Napriek širokému akceptovaniu dôležitosti a prínosu používateľovej spoluúčasti a dostupnosti metodológii, v praxi sú vzťahy medzi používateľom a vývojármi menej úspešné.

Podľa mňa takáto spolupráca v praxi zlyháva najmä z týchto dôvodov:

- slabá motivácia zákazníka. Zákazník nechce byť zahrnutý do procesu vývoja, chce aby mu vývojári dodali systém podľa jeho predstavy.
- časové problémy. Zákazník má vlastnú prácu, ktorá ho oberá o čas a nemusí ho mať dostatok aj pre vývojárov.

Ak sú používatelia angažovaní v čo najskoršej fáze vývoja projektu, tak projekt má väčšiu šancu uspieť. Mali by pravidelne konzultovať dizajnové rozhodnutia s vývojármi, ešte lepšie by bolo, keby boli plne integrovaní do vývojárskeho tímu. Toto podľa mňa nie je väčšinou možné, lebo informačné systémy používajú aj ľudia vo vyšších firemných štruktúrach, ako napr. riaditelia, manažéri, atď. Pre nich je takáto spolupráca nemožná. Očakávajú však, že systém bude dizajnovaný tak, aby splnil ich požiadavky, ale nemajú čas a možno ani záujem byť zahrnutí v detailoch vývoja. Väčšinou sú neochotní spolupracovať s vývojármi a môžu byť citliví aj na svoju technickú zdatnosť.

V takýchto situáciách sa vývojári nemôžu spoliehať na kooperatívnu spoluprácu so zákazníkom, ale na svoje schopnosti prekonať tieto obmedzenia. Vývojár musí ukázať značnú vynachádzavosť v hľadaní ciest ku odhadnutiu používateľových požiadaviek. V ďalšej časti popíšem rôzne spôsoby komunikácie medzi vývojármi a zákazníkom.

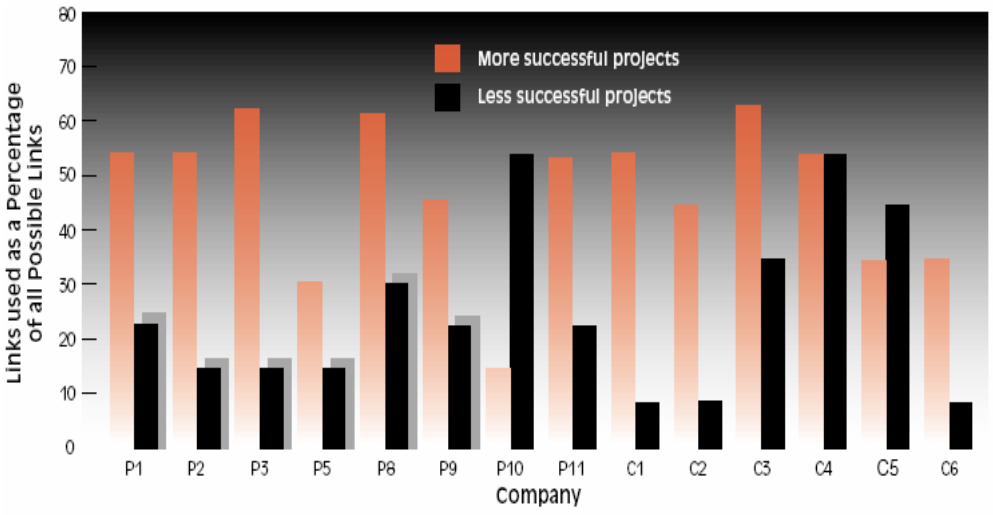
Spôsoby komunikácie

Prečo je tak dôležitá komunikácia medzi zákazníkom a vývojármi? Myslím si, že preto, lebo zákazník má najväčšiu znalosť oblasti, kde sa bude používať daný systém. A teda môže byť zdrojom veľmi užitočných informácií a bola by škoda, keby to vývojári nevyužili.

Získanie takýchto užitočných informácii vyžaduje zavedenie jednej alebo viacerých liniek medzi zákazníkom a vývojárom. Pod linkou sa myslí technika alebo

kanál, ktorý umožní komunikáciu medzi zákazníkom a vývojárom. V dnešných časoch existuje takýchto liniek alebo spôsobov komunikácie veľké množstvo, z ktorých si vývojári môžu vyberať. Technologický vývoj a zníženie cien ich sprístupnilo prakticky každému. Získať vstupy od zákazníka nikdy nebolo ľahšie ako teraz.

Veľké množstvo dostupných liniek dáva príležitosť a znamená výzvu pre softvérových manažérov. Manažéri musia rozhodnúť, ktoré linky a koľko sa ich vyberie na komunikáciu. Na obrázku je znázornená závislosť úspešnosti projektov od percenta použitých komunikačných kanálov z určitého maximálneho počtu podľa jednej štúdie[2]



Obr. 3. Závislosť úspešnosti projektov od počtu použitých komunikačných kanálov

Z obrázku je vidieť, že projekty, kde sa komunikovalo viacerými kanálmi, boli úspešnejšie. Pre predstavu uvádzam v nasledujúcej tabuľke linky, ktoré autori brali do úvahy pri štúdii:

Linka	**Popis**
Pomocný tím	Nápomocný, štruktúrovaný tím, zvyčajne sa používa na zistenie požiadaviek na projekt.
Zákaznícka podpora	Zákaznícka podpora, pomáha zákazníkom s každodennými problémami.
Dizajnové prototypovanie	Zákazníkovi je odkryté demo/prototyp programu v ranej fáze projektu na odhalenie prípadných počiatočných problémov.
Prototypovanie požiadaviek	Zákazníkovi je predvedené demo programu v ranej

	fáze projektu za účelom odhalenia požiadaviek na systém.
Interview	Vývojár komunikuje so zákazníkom za účelom získania ďalších informácií.
Testovanie	Môžu vzniknúť nové požiadavky a spätná odozva od zákazníka pri testovaní projektu.
Email	Zákazník posiela problémy, otázky a návrhy na zlepšenie.
Používateľské laboratórium	Špeciálne skonštruované laboratórium na sledovanie zákazníka pri práci.
Štúdia	Zákazníci informujú vývojárov o tom čo robia, za účelom pochopenia problémovej oblasti.
Marketing a predaj	Pravidelné stretávanie s aktuálnymi a potenciálnymi zákazníkmi za účelom zistenia návrhov a potrieb trhu.
Používateľské skupiny	Zákaznícke skupiny, ktoré diskutujú o softvéri a možných zlepšeniach.
Prezentácia produktu	Zákazníkovi je odprezentovaný produkt a zistená jeho odozva na prezentáciu.
Záujmová skupina	Malá skupina zákazníkov diskutuje s vývojármi o softvére.

Výsledky tejto štúdie sa mi zdajú nepresné. Nikde sa v nej neuvádza, aké typy projektov brali autori do úvahy. Hodnotenie úspešnosti projektu vychádzalo len zo subjektívnych názorov manažérov. Číselne vyhodnotiť úspešnosť projektu môže byť ťažké a nepresné. Je zrejmé aj to, že nie pre každý projekt sa dá použiť alebo je vhodné použiť ľubovolnú linku uvedenú v tabuľke.

V nasledujúcej časti sa pokúsim popísať výhody a nevýhody, ktoré môžu vzísť z komunikácie medzi zákazníkom a vývojárom.

Výhody komunikácie medzi zákazníkom a vývojárom

Výhody spolupráce zákazníka s vývojármi sú zrejmé. Zlepšuje sa proces získania požiadaviek na systém, zákazníci sú viac informovaní o postupe vývoja, čo zas vedie k väčšej spokojnosti zákazníkov a k vyššej kvalite výsledného systému a nakoniec aj k väčšej použiteľnosti systému.

Podľa mňa, zákazníci, ktorí sú zahrnutí v počiatočných fázach projektu, alebo dokonca počas celého vývoja systému a teda vidia, že systém sa vyvíja podľa ich prianí a požiadaviek, sú viac náchylní akceptovať a používať daný systém. Takáto spolupráca je teda výhodná pre obidve strany. Zákazník si môže všimnúť viac detailov, ktoré môžu vývojári ľahko prehliadnuť, keďže nie sú odborníci z danej oblasti. Nakoniec takáto spolupráca môže byť výhodná aj finančne, keďže vďaka tomu, že zákazník je

k dispozícii vývojárom takmer okamžite, tak sa vývoj môže aj urýchliť a tým znížiť náklady výsledného produktu.

Pre vývoj použiteľného systému by mala byť zákaznícka podpora nevyhnutná. Teda platí, že čím skôr sa zákazník zúčastní na vývoji produktu, tým bude vývoj lacnejší kvôli neskorším výdavkom na zmenu systému z dôvodu nepochopenia požiadaviek a zlej špecifikácie. Výhody takejto komunikácie sú:

- výsledná kvalita systému je lepšia, lebo vďaka neustálej spolupráci zákazníka vývojári lepšie a presnejšie pochopia požiadavky kladené na systém
- môže sa predísť neskorším kritickým zlyhaniam systému
- výrazne zlepšenie šance na akceptovanie systému zákazníkom
- zákazník lepšie pochopí systém a bude ho vedieť používať efektívnejšie

Keďže zákazník je odborníkom z oblasti, kde sa bude vyvíjaný systém používať, má možnosť výrazne ovplyvniť vývoj systému. V počiatočných fázach projektu môže pomôcť pri dizajnových problémoch. V neskorších fázach dáva návrhy na zlepšenie, komentáre k riešeniu špecifických problémov alebo kritizuje dodané riešenie. Vývojári majú teda okamžitú odozvu od zákazníka, čo nakoniec povedie ku kvalitnejšiemu systému. Samozrejme, čím viac sa zákazník zapája do vývoja, tým je väčšia šanca, že finálny projekt bude úspešný, či už z hľadiska predajnosti alebo spokojnosti používateľov systému.

Nevýhody komunikácie medzi zákazníkom a vývojárom

Čo ak zákazník zle identifikuje alebo špecifikuje požiadavky na systém? Toto je zrejme najhoršia možnosť, ktorá môže vyplynúť z komunikácie medzi zákazníkom a vývojármi. Alebo ak si zákazník myslí, že pozná všetky odpovede na otázky a chce, aby sa všetko robilo ako on chce. Toto môže byť ďalší problém, na ktorého riešenie sa ťažko hľadá odpoveď.
Ďalšie nevýhody:

- veľa času zaberie zhromažďovanie dát
- zákazník sa bráni byť pozorovaní pri práci, pozorovatelia majú dojem, že pracovník nepracuje, keď ho sleduje
- ťažkosti pri dohodnutí pozorovania koncového používateľa pri práci

Záver

Je vhodné zakaždým angažovať zákazníka do vývoja systému? Sú situácie, kedy je to nevhodné? Odpovede na tieto otázky sa hľadajú ťažko. Každý vyvíjaný systém je iný, preto sa ani nedá vytvoriť nejaký ideálny model spolupráce medzi zákazníkom

a vývojárom. Isté je len to, že ak chcú manažéri zvýšiť šancu svojho projektu na úspech, mali by vytvoriť čo najviac komunikačných kanálov so zákazníkom.

Použitá literatúra

1. Joe Nandhakumar, Matthew Jones: Designing in the dark: The changing user-developer relationship in information systems development, (1997) 75-88.
2. Mark Keil, Erran Carmel: Customer-developer links in software development, Communications of the ACM, Vol. 38, Issue 5 (May 1995) 33-44
3. Yasemin Salihoğlu: User Involvement in Software Project Development: A Review of Models, Software Project Management Research Paper, 12s.

Annotation

Relations between customer and programmer

In present times the most software projects are not finished with success. It is dependant on bad relation between customer and programmer, mostly on communication relations. In this essay are characterized the ways of communication between customer and programmer, which can increase the chance of project success.

Manažment kvality a vplyv na výsledok projektu

PETER SÝKORA

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

Abstrakt. Kvalita je rôznymi odborníkmi na kvalitu definovaná rôzne. Dokument sa zaoberá problémom, akým je vnímanie kvality z pohľadu zákazníka. Zaoberá sa aj používanými metrikami pri meraní kvality a potrebou kvantifikovať kvalitu. Ďalej uvádza prezentáciu podprocesov procesu manažmentu kvality (Juranova trilógia) a hlavné nástroje kvality požívané na zvyšovanie kvality. Pokúšam sa odpovedať na otázku, prečo je potrebné neustále zlepšovať kvalitu a ako vplývajú certifikáty kvality na koncový produkt. Na konci sú rozobraté účinky manažmentu kvality na výsledok projektu, t.j. aké prínosy z toho vyplývajú a aké prostriedky je nutné vynakladať na zabezpečenie kvality.

Úvod

Cieľom asi každého manažéra softvérového projektu je vytvoriť produkt čo najvyššej kvality. Kvalita však nie je dielom náhody a musí byť plánovaná [2]. Dnes už síce existuje mnoho príručiek a prístupov, ktoré vytvorili experti na kvalitu a profesionáli z praxe, ale žiadny s týchto nepredpisuje presný postup na zvyšovanie kvality. Zvyšovanie kvality je zabezpečené hlavne manažmentom, ktorý má za úlohu plánovanie, riadenie a zlepšovanie kvality.

Kvalita verzus zákazník

Každý zákazník má na kvalitu svoj vlastný názor. Podľa ISO 9000:2000 je kvalita miera akou súbor vlastných charakteristík spĺňa požiadavky. Teda v prípade, že niekto dodá zákazníkovi produkt, ktorý lepšie spĺňa jeho požiadavky ako môj produkt, tak má tento produkt pre neho vyššiu kvalitu.

Môj názor je, že zákazník posudzuje kvalitu softvéru aj podľa ostatných programov a informačných systémov, ktoré používa. Kvalita nie je statická, pretože potreby ľudí a situácia sa mení a podľa toho by sa mal meniť aj program ak si chce zachovať kvalitu.

Manažment kvality a vplyv na výsledok projektu, január 2006, s. 69-75.

Nie je možné vytvoriť kvalitný produkt bez poznania hodnotenia ľuďmi, ktorí ho budú používať v danom čase a na danom mieste.

Meranie kvality

Merania sú dobré pre rozpoznanie úspechov, nielen pre odhalenie problémov.

Požiadavky zákazníka sú väčšinou subjektívne. Je potrebné ich previesť do objektívnych merateľných parametrov. Napríklad zákazník má požiadavku, aby bol výsledný softvérový systém spoľahlivý. Túto požiadavku je napr. možné previesť na množstvo porúch za časové obdobie. Presný postup prevodu nie je pevne daný a je potrebné čo najlepšie poznať potreby zákazníka.

Kan v *Metrics and Models in Software Quality Engineering* [3] identifikoval niekoľko kategórií metrík. Rozdelil metriky kvality softvéru do dvoch tried: metriky kvality koncového produktu a metriky kvality procesu.

Metriky procesu by mali podľa neho obsahovať hlavne metriky množstva a časov príchodov chýb počas testovania. Tieto metriky pre svoje použitie potrebujú veľmi dobre prepracovaný manažment testovania.

Metriky koncového produktu sú často jediné metriky, ktoré zaujímajú zákazníka. Kategórie metrík koncového produktu obsahujú: metriky softvérovej spoľahlivosti, metriky hustoty chýb, metriky hustoty problémov u zákazníka a metriky spokojnosti zákazníka. Podľa mňa nie sú tieto metriky pre aktuálny projekt veľmi dôležité, lebo málokedy vedú k rapídnej zmene projektu, pretože takéto úpravy sú veľmi nákladné. Veľmi vhodné sú však pre poučenie z predchádzajúcich chýb v ďalších projektoch. Dokonca sú veľmi dôležité, ak sa organizácia snaží o neustále zlepšovanie kvality.

Manažovanie pre kvalitu

Pre dosiahnutie kvality je dobré začať stanovením „vízie“ organizácie spolu so stratégiami a cieľmi. Konverzia cieľov na výsledky sa deje cez manažérske procesy (postupnosť aktivít, ktoré produkujú zamýšľané výsledky). Manažovanie pre kvalitu zahŕňa v zvýšenej miere tieto tri manažérske procesy [2]:

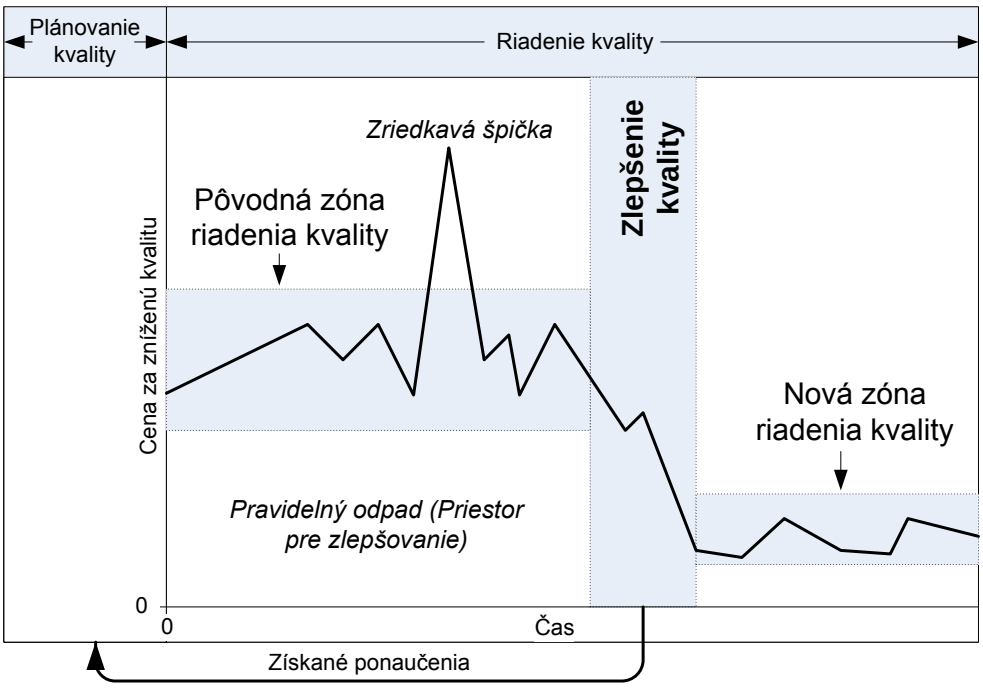
- plánovanie kvality,
- riadenie kvality,
- zlepšovanie kvality.

Plánovanie kvality súvisí s inicializáciou projektu. Pri inicializácii projektu sa treba hlavne zamyslieť nad cieľmi kvality. Ďalej je potrebné identifikovať, kto sú zákazníci, aké sú ich potreby. Nakoniec treba vyvinúť proces tvorby softvéru a spôsob riadenia kvality tak, aby sa v čo najväčšej miere minimalizovali nedostatky.

Benchmarking [2] je pomenovanie pre pre postup stanovenia cieľov založený na cieľoch dosiahnutých ostatnými. Štandardným cieľom je požiadavka, aby spoľahlivosť nového produktu bola aspoň na takej úrovni, ako produkt, ktorý ide nahradiť, a aspoň

na takej úrovni, ako najlepší konkurenčný produkt. Výhodou použitia tohto prístupu je poznatok, že výsledné ciele nie sú nedosiahnuteľné, pretože už boli dosiahnuté inými. Hlavný postup pri použití benchmarkingu je identifikovať konkurenciu, identifikovať porovnávané charakteristiky a spôsob porovnávania týchto charakteristík. Toto sa dá použiť aj prípade porovnávania softvérového procesu aj porovnávaním produktov.

V prípade, že sú kladené vysoké požiadavky na spoľahlivosť softvéru je dobré pri plánovaní použiť diagram príčin a následkov(tzv. Ishikawov Diagram)[2]. Ako následok sa zvolí nespoľahlivý program (popr. smrť človeka) a identifikujú sa možné príčiny. Výhodou tohto postupu je podľa mňa možnosť identifikovať možné poruchy systému ešte skôr, ako nastanú.



Obr. 1. Juranova trilógia (obrázok prebratý z [2])

Riadenie kvality je vlastne proces, ktorý zabezpečuje dodržiavanie postupov a udržanie charakteristík na určitej úrovni, ktoré boli definované v čase plánovania kvality. Tento proces sa skladá zo zbierania a analyzovania údajov. V prípade, že charakteristiky presiahnu nastavené medze je potrebné použiť protiopatrenia. Medzi protiopatrenia by som zaradil napríklad zvýšenie motivácie zamestnancov.

Obr. 1 je použitý na opísanie vzťahov medzi plánovaním kvality, riadením kvality a zlepšovaním kvality: potom ako boli vykonané zlepšenia procesu, dosiahol sa nový stupeň vykonávania. Teraz je potrebné vytvoriť nové kontrolné mechanizmy, aby sa predišlo chátraniu procesu k predchádzajúcej alebo ešte k horšej úrovni.

Diagram na **Obr. 1** je dosť všeobecný, aby mohol popisovať proces v rôznych odvetviach. Ja pod pravidelným odpadom v procese tvorby softvéru rozumiem prácu, ktorá by sa v prípade dobrého plánu kvality nemusela vykonávať. Táto redundantná práca môže nastať napríklad v prípade, keď je potrebné zahodiť určitú časť projektu, pretože neboli pochopené požiadavky zákazníka. Riešenie tohto problému by mohlo spočívať napríklad zlepšením postupu analýzy potrieb zákazníka. Ďalším príkladom by mohlo byť riešenie vzniknutých problémov v prípade nedostatkov procesu manažmentu verzií zdrojových kódov. Použitím vhodných nástrojov by sa tieto problémy dali odstrániť a dosiahli by sme zlepšenie procesu tvorby softvéru.

Zlepšenie kvality softvérového procesu sa dá dosiahnuť rôznymi prostriedkami. Veľmi jednoduchým príkladom je zavedenie manažmentu zmien. Pri zlepšení procesu sa získajú skúsenosti, ktoré môžu byť použité pri nasledujúcom plánovaní kvality. Napríklad, keď plánovaným zlepšením kvality sa dosiahne zhoršenie, je nanajvýš vhodné sa z toho poučiť a identifikovať dôvody zníženia kvality.

Modely kvality

CMM (the Capability Maturity Model for Software) je referenčný model [1] pre hodnotenie zrelosti softvérového procesu a ako normatívny model poskytujúci návod softvérovým organizáciám ako vyvíjať softvérový proces od ad hoc chaotického k zrelému disciplinovanému softvérovému procesu.

Jedným zo spôsobov, ako sa presadiť na trhu je získať certifikát kvality. Zákazník má istú mieru istoty, lebo certifikát do určitej miery zaručuje používanie postupov, ktoré sú overené auditmi. Niektoré veľké spoločnosti dokonca požadujú určitý druh certifikátu.

Štandardy rodiny ISO 9000 poskytujú požiadavky na vlastnosti systému manažmentu organizácie, ale nepredpisujú ako tieto vlastnosti implementovať.

Certifikát ISO 9000 nie je vôbec jednoduché získať. Veľa spoločností žiada o registráciu, ale Kan [3] tvrdí, že 60-70% neprejde úspešne ani prvotným auditom.

Niektoré spoločnosti majú oddelenie zabezpečovania kvality softvéru (angl. Software quality assurance department). Takéto oddelenie predpisujú oba spomínané modely. V prieskume, ktorý robili Wheeler a Duggings [5] na otázku: „Má vaša organizácia oddelenie na zabezpečenie kvality softvéru, ktoré stanovuje a dohliada na dodržiavanie štandardov, ktoré definujú spôsob dosiahnutia kvality?“ 47 z 80 respondentov odpovedalo kladne. Tento výsledok však neznamená, že sa všetkým týmto spoločnostiam podarilo proces manažmentu kvality zvládnuť. Môj osobný názor je, že vytvorením takéhoto oddelenia sa v niektorých prípadoch zvýši len množstvo byrokracie.

Spôsoby zvyšovania kvality

Medzi najčastejšie používané nástroje kvality patria porovnávanie s konkurenciou, Paretova analýza a diagram príčin a následkov.

Paretova analýza [2] vychádza z princípu, že 80 percent negatívnych dôsledkov je spôsobených 20-timi percentami najzávažnejších príčin. Teda odstránením menšej časti príčin sa dá dosiahnuť odstránenie väčšej časti negatívnych dôsledkov. Napríklad, v prípade potreby zníženia počtu kritických chýb softvéru je možné vykonať analýzu, aby sa zistilo v ktorých častiach procesu tvorby vzniká väčšina chýb, potom identifikovať príčiny a snažiť sa ich odstrániť.

Diagram príčin a následkov [2] sa používa, keď je potrebné analyzovať dôsledok a odhaliť jeho príčiny. V prvom kole sa zisťujú hlavné príčiny dôsledku. V druhom kole sa za dôsledky zoberú tieto príčiny a analyzuje sa možnosť ich vzniku (teda znova príčiny).

Benchmarking [2] je jeden z najdôležitejších nástrojov v oblasti kvality. Hľadanie a napodobňovanie najlepšieho môže posilniť motiváciu každého zainteresovaného, často vyprodukovaním prelomových výsledkov. Cieľom benchmarkingu je dosiahnuť výhodu oproti konkurencii.

Potreba zlepšovania

Zavedením postupov na zlepšenie kvality procesu sa práca manažmentu nekončí. Aj v prípade, že organizácia získa certifikát kvality je nutné sa zlepšovať, pretože jednou z podmienok udržania si certifikátu je nutnosť sa stále zlepšovať (ISO 9000). Inak by pri pravidelných auditoch organizácia o certifikát prišla.

Očakávania zákazníka časom rastú a preto nie je možné sa spoľahnúť, že zákazník bude vždy považovať za kvalitné to isté. Časom rastie aj kvalita konkurencie, preto zlepšovanie je aj jedna z podmienok udržania postavenia na trhu.

Cena za kvalitu

Podľa Jurana [2] je celková cena za kvalitu = cena prevencie + cena posúdenia + cena interných chýb + cena externých chýb.

Cena prevencie je cena aktivít, ktoré sú špecificky navrhnuté na prevenciu pred nízkou kvalitou. Napríklad nízka kvalita môže zahŕňať programátorské chyby, chyby v návrhu, nedostatky v používateľskom manuály, alebo neudržiavateľný kód.

Cena posúdenia zahŕňa cenu za aktivity, ktoré sú určené na odhalenie problémov. Pre oblasť softvérového inžinierstva by som sem zaradil napríklad posúdenie návrhu, posúdenie prototypu a používané testy softvéru.

Cena interných chýb je cena odstránenia zistených nedostatkov ešte predtým, ako sa produkt dostane k zákazníkovi.

Cena externých chýb zahŕňa náklady vzniknuté po odovzdaní produktu do používania koncovému zákazníkovi (napr. cenu za vykonaný servis, cenu za vytváranie a distribuovanie záplat).

Cieľom pri tomto prístupe je celkovú cenu za kvalitu minimalizovať. Nie je zaručené, že zvýšením výdavkov na prevenciu klesne množstvo výdavkov ako si veľa ľudí myslí, aj keď je všeobecne známe, že čím skôr sa v softvérovom projekte chyba

odhalí, tým menej nákladné je jej odstránenie. Najdôležitejšie je aby postupy prevencie boli dobre naplánované, pretože drahšia prevencia nie vždy znamená lepšia.

Prínosy manažmentu kvality

Jedným z dôvodov, prečo som za použitie manažmentu kvality v organizácii, je výchova zamestnancov a poučenie sa z projektov. Ako vyplýva z výsledkov prieskumu, ktorý vykonali Kautz a Ramzan [4], manažment kvality softvérového projektu zvyšuje uvedomelosť o vývoji softvéru a jeho problémoch.

V prieskume, ktorý zverejnili Wheeler a Duggings [5], autori zistili, že 41% organizácií s oddelením na zabezpečenie kvality vydalo po oficiálnom uvedení záplatu za kratšiu dobu ako 30 dní z dôvodu kritickej chyby. V prípade spoločností bez tohto oddelenia to bolo len 33%. Z tohto v závere vyvodili, že neexistuje žiadna závislosť medzi rýchlosťou objavenia kritickej chyby a existenciou takéhoto oddelenia. Autori však neuviedli veľkosť spoločností ani rozsah projektov a ešte keď zoberiem do úvahy, že respondentov bolo len 80, tak sa z toho podľa mňa nedá vyvodiť žiaden záver.

Záver

Úloha zlepšovania kvality softvéru je najmä manažérska úloha, nie technická. Vyžaduje si v čo najväčšej miere zainteresovanosť manažmentu. Problém vývoja softvéru väčšinou nie je v tom, že nebola použitá najnovšia, „výborná“ technika vývoja, ale to že organizácia nevedela dobre odhadnúť obrovské problémy v produkovaní kvalitného softvéru a nevenovala dostatočnú pozornosť praktickým detailom tvorby softvéru, ktorý robí presne to, čo zákazník od neho požaduje.

Organizácie zaoberajúce sa vývojom softvéru občas zabúdajú na to, že najdôležitejšou metrikou kvality je spokojnosť zákazníka, ktorá nie je u každého zákazníka rovnaká. Často si zamieňajú kvalitu s existenciou oddelenia pre zabezpečenie kvality. Kvalitu dokonca nezaručuje ani certifikát kvality. Certifikát kvality len do určitej miery zaručuje používanie a vlastnosti manažérskych postupov, pre ktoré je známe, že vo väčšine prípadov vedú k zlepšeniu kvality.

Proces manažmentu kvality sa skladá z plánovania kvality, riadenia kvality a zlepšovania kvality. Zlepšovaním kvality sa dá dosiahnuť zníženie nákladov na zabezpečenie kvality.

Cena za kvalitu zahŕňa náklady na prevenciu, cenu posúdenia, cenu interných chýb a cenu externých chýb. Za predpokladu, že cena externých chýb (chýb odhalených až u zákazníka) je najvyššia, zlepšením prevencie má tendenciu celková cena za kvalitu klesať.

Medzi prínosy manažmentu kvality patrí hlavne výchova zamestnancov, pretože manažment kvality poskytuje lepšie možnosti pochopiť vlastnosti a problémy vývoja softvéru.

Použitá literatúra

1. Herbsleb, J., Zubrow, D., Goldenson, D., Hayes, W., Paulk, M.: Software Quality and the Capability Maturity Model. *Communications of the ACM*, Vol. 40, No. 6 (1997), 30-40.
2. Juran, J. M., Godfrey, A. B.: *Juran's Quality Handbook (5th Edition)*, McGraw-Hill, 2002.
3. Kan, S. H.: Metrics and Models in Software Quality Engineering, Addison-Wesley, 1995.
4. Kautz, K. and Ramzan, F.: Software Quality Management and Software Process Improvement in Denmark. In: *Proceedings of the 34th Annual Hawaii international Conference on System Sciences (Hicss-34)-Volume 5*. HICSS. IEEE Computer Society, Washington, DC (2001), 5014.
5. Wheeler, S. and Duggins, S.: Improving software quality. In: *Proceedings of the 36th Annual Southeast Regional Conference* ACM-SE 36, ACM Press, New York (1998), 300-309.

Annotation

Quality management and its effect on project result

Quality definition differs from one quality expert to another. Document discuss the problem: what means quality for customer. Quality measurement metrics and the need of quality measurement are noted here. Then subprocesses of quality management process (Juran's trilogy) and essential quality tools used for increasing quality are presented. I am trying to answer the question: Why are quality improvements needed and how certificates are influencing end-product quality. At the end are effects on project result analyzed which consists of quality income and quality costs.

Chyby a manažment softvérových projektov

TOMÁŠ VANDERKA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
tomas.vanderka@ynet.sk

Abstrakt. Všetci dobre vedia že ľudia robia chyby, robia ich neustále, ale nik nevie dopredu povedať s určitosťou kedy a aké. Už niekoľko desaťročí ľudia vyvíjajú techniky na zredukovanie chýb a zvýšenie efektivity vývoja softvéru. Napriek tomu softvérové projekty stále zlyhávajú, aj napriek pokrokom v technológii. Tento článok sa snaží v krátkosti popísať a vysvetliť niekoľko dôvodov prečo sa tak deje, a načrtnúť čo môžeme robiť aby sme tieto problémy zvládli.

Úvod

Softvérové inžinierstvo je relatívne nová oblasť, aspoň v porovnaní s inými oblasťami. Pri vývoji softvéru ide zväčša o tímové úsilie vytvoriť niečo podľa špecifických potrieb zákazníka. Na celý tím sú vytvárané tlaky, či už zvnútra alebo zvonku spoločnosti pre ktorú pracujú, ale aj úplne nezávislé vplyvy ako rodinné problémy, a množstvo ďalších. Všetky tieto tlaky prispievajú k vzniku chýb v riadení, rozhodovaní, programovaní atď. Je nemožné brať do úvahy všetky tieto vplyvy pri manažovaní projektu, keď ich ani nedokážete všetky vymenovať.

Napriek tomu všetkému, existuje veľké množstvo prístupov k manažovaniu takýchto projektov, ktoré znižujú riziko problémov. Žiadny z nich ale výskytu chýb nemôže zabrániť úplne. Pretože nech sa použije ktorýkoľvek z nich, vždy existuje šanca že niekto spraví chybný krok. Jediné čo môžete spraviť je snažiť sa znížiť jeho dopady, poučiť sa z neho a pokračovať ďalej.

Preto sa nebudem venovať konkrétnym metódam manažovania projektov, ale skôr bežným chybám, ktoré môžeme spraviť nezávisle od použitej techniky. Budem sa snažiť popísať dôvody ich vzniku, ich dopady a poskytnúť rady ako sa ich vyvarovať.

Najbežnejšie chyby

Nasledujúci zoznam zhŕňa najbežnejšie chyby, ktorých sa môžete dopustiť pri manažovaní softvérového projektu. Ich poradie nijak neurčuje ich závažnosť.

Chyby a manažment softvérových projektov, január 2006, s. 77-81.

- Vytýčenie nerealistických cieľov
- Zlé odhady potrebných zdrojov
- Negatívna motivácia v prípade tiesne
- Nedostatočné pochopenie požiadaviek
- Slabé sledovanie pokroku práce
- Slabá komunikácia
- Použitie neoverenej technológie
- Podľahnutie komerčným tlakom
- Nedostatočné testovanie

Ku každej spomenutej chybe popíšem v krátkosti jej následky a možné riešenie.

Prehnané ciele projektu

V rámci komunikácie obchodníkov so zákazníkom sa môže ľahko stať že sa ponúkaný systém zvrhne na kolos, ktorý bude riešiť všetky používateľove najtajnejšie sny. To neskôr ovplyvní aj jeho pohľad na riešenie, ktoré v skutočnosti potrebuje. Projektový manažér by si mal dať pozor aby projekt ostal v hraniciach možností. Ak sa rozhodne implementovať projekt, na ktorý nemá dostatok prostriedkov tak ho nedokončí ani keby na to mal nekonečno času.

Zlé odhady prostriedkov a času

Odhady sú samé o sebe zložitá vec. Nemôžem povedať čo je dobrý a čo zlý odhad, pretože až na konci sa ukáže či bol dobrý alebo nie. Odhady môže robiť iba skúsený manažér, ktorý už má za sebou viacero podobných projektov. Nikdy by sme nemali strieľať odhady len tak z hlavy, mali by byť vždy podložené nejakými reálnymi skúsenosťami.

Najdôležitejšie ale je tieto odhady neskracovať podľa želania zákazníka, alebo iných nadriadených. Takto sa nedá nič získať, projekt bude trvať vždy rovnako dlho. Dokonca ak je odhad príliš mimo môže projekt trvať ešte dlhšie, pretože neskôr nebude možnosť dynamicky zmeniť plán a pribrať nových ľudí. O tom hovorí aj staré známe pravidlo od Brooksa [1], ktoré hovorí že to spomalí projekt ešte viac.

Ďalej je vždy lepšie použiť pesimistické odhady ako tie optimistické, pretože v prípade optimistických sa môžeme dostať iba do negatívneho sklzu.

Negatívna motivácia

Niektorí manažéri majú pod tlakom tendenciu prenášať tento tlak na ostatných členov tímu. Väčšina ľudí ale na tlak reaguje negatívne, hlavne ak ide o kreatívnu prácu akou tvorba softvéru určite je. Takýto tlak iba zvýši nervozitu, napätie a zníži vynaliezavosť, čiže dosiahne sa pravý opak požadovaného výsledku. Nesymetricky pozitívna

motivácia má kladný efekt [7], a úspešné smerovanie projektu treba dať pocítiť aj členom tímu, napríklad malými oslavami k ukončeniu určitého míľnika.

Samozrejme že problémy treba riešiť, ale konštruktívne. Je možné skúsiť napríklad brainstorming a odhaliť nejaké nové nápady. Alebo si na chvíľu dať s problémom pokoj, a neskôr sa riešenie objaví veľmi rýchlo, keď sa na problém pozrieme z inej strany. V prípade negatívnej motivácie nastáva aj útlm nových nápadov, ktoré by mohli priniesť riešenie, pretože v strese sa každý správa negativisticky a skôr hľadá prekážky a nedokáže vidieť pozitíva.

Nepochopenie požiadaviek

Celý projekt je v podstate implementácia požiadaviek. Ak tieto nie sú jasné hneď na začiatku, môže sa ľahko stať že sa projekt začne uberať zlým smerom, a tento smer už neskôr nebude možné jednoducho zmeniť. Preto treba dbať na to aby boli požiadavky poriadne vyjasnené so zadávateľom čo najskôr na začiatku projektu. Niektorí sa bránia zmenám požiadaviek v priebehu projektu pomocou kontraktov so zákazníkom, to ale niekedy môže mať za následok že klient súhlasí s niečím, čo sa mu na začiatku možno páči, ale neskôr všetci prídu na to že chcel niečo iné. Ak aj firma má kontrakt s funkčnými požiadavkami, klient nemôže namietať, ale nakoniec nebude spokojný.

Nikdy sa netreba spoliehať na to že klient vie čo od systému očakáva. Z doterajších procesov môže mať zafixované určité rutiny, ktoré sa mu zdajú byť jediným správnym spôsobom ako dosiahnuť cieľ. Takéto predsudky automaticky blokujú akékoľvek iné možné riešenia.

Preto sa treba na začiatku čo najviac zamerať na pochopenie čo budúci používateľ naozaj potrebuje, a nie na to čo si sám pýta [7]. Na to aby vznikla dohoda je ale nutné s ním všetko konzultovať až sa dospeje k obojstranne jasnému výsledku.

Niekedy je vhodné keď sa požiadavky menia aj počas realizácie projektu, v úzkej spolupráci s klientom pri každej fáze projektu. Samozrejme v obmedzenom rozsahu, inak každá zmena príliš spomaľuje vývoj. Na druhej strane sa ale viac približuje k požadovanému výsledku.

Nedostatočné sledovanie postupu prác

Niektorí ľudia sa v prípade problémov zvyknú tváriť že všetko je v poriadku, ale skutočnosti nevedia ako ďalej. Takáto situácia môže trvať dlhší čas, až nakoniec vyjde najavo, že v skutočnosti projekt stojí. Vtedy je to ako by ste šliapli na mínu. Z ničoho nič nastane zmätok, reorganizácia, zmena plánov, časový sklz.

Práve dostatočné reportovanie a kontrola reálneho stavu projektu môže zabrániť podobným situáciám. Niekedy nie je postačujúce uspokojiť sa s tým čo nám iný ľudia povedia, treba veci overiť dôkladne. To je efektívne možné iba keď sú procesy v rámci projektu dostatočne transparentné. Napríklad projekt je vhodnejšie rozčleniť na malé časti, ktoré je jednoduchšie kontrolovať. Takisto sa môže využiť sledovanie verzií, automatické testovanie a reportovanie.

Dôležité je, že to netreba preháňať. Každá práca vývojára pomimo jeho hlavnej úlohy v projekte ho môže značne spomaliť. Aj pár minútové prerušenie počas produktívnej fázy ho dokáže vyviesť zo sústredenia a spomaliť aj o niekoľko hodín.

Preto sa treba zamerať najmä na automatizované zaznamenávanie a sledovanie pokroku v projekte, aby sa tým čo najmenej obmedzovala kreatívna práca vývojárov.

Slabá komunikácia v tíme aj mimo neho

S rozsahom projektu sa zväčšuje okruh ľudí, ktorí musia navzájom komunikovať pri riešení úloh. Nie je dokonca zriedkavé, že na projekte pracuje viacero oddelených tímov z viacerých spoločností.

Ak je táto komunikácia nedostatočná, ľudia spolu nehovoria, niektoré problémy sa nedostanú do uší ľuďom, ktorí by ich mohli vyriešiť. Takto sa postupne nazbiera masa skrytých prekážok, valiacich sa spolu s projektom. A až nakoniec niektorá vypláva na povrch, jej odstránenie vyžaduje veľké úsilie. Tiež môže nastať situácia, keď viac ľudí pracuje na tom istom iba preto, lebo o tom nik iný nevie. Takto sa zbytočne plytvá ich časom.

Tímový manažér by mal priebežne sledovať čo sa v tíme deje, usporadúvať stretnutia, na ktorých by sa riešili všetky problémy. Raz za čas je dobré usporiadať stretnutie kde každý člen z tímu spomenie niečo na čo narazil ale nepočul o tom hovoriť nikoho iného. Niektoré veci sa totiž zdajú byť zrejmé a jasné určitým ľuďom a vôbec ich nenapadne žeby o tom ostatní nevedeli. Takto je možné pomerne skoro odhaliť nové skryté problémy, alebo riešenia na tie už známe.

Použitie novej a neoverenej technológie

Vždy keď sa objaví nejaká nová aspoň trochu úspešná technológia, začne sa okolo nej tvoriť veľký záujem. To že o niečom všetci naokolo hovoria ešte neznamená že je to použiteľné a vhodné pre konkrétny projekt. Ak chcete takúto novinku použiť mali by ste si dobre rozmyslieť jej pozitíva, negatíva a aj prípadné riziká, ktoré môže priniesť ak sa neosvedčí.

Záujem o nové technológie sa väčšinou riadi podľa známej „hype“ krivky. Po období veľkého záujmu nastáva útlm, a až neskôr sa situácia stabilizuje. Bezpečný spôsob je najskôr nechať iných overiť jej zrelosť. Určite sa nájdu takí čo naskočia na kolotoč a nie všetkým sa to vyplatí. Treba si brať skúsenosti z iných projektov, ktoré sa ju snažili implementovať, či už sa im to podarilo alebo nie. Takto môžete získať prehľad o prípadných problémoch, na ktoré môžete s touto technológiou naraziť.

Podľahnutie komerčným tlakom

Pri každom komerčnom projekte je prirodzený tlak na minimalizáciu nákladov a času potrebného na ukončenie projektu. Netreba sa nechať dotlačiť k stene a prijať podhodnotené odhady a rozpočet. Ak si necháte vnútiť niečo čo nie je možné splniť, raz musíte naraziť na hranicu možností, a potom sa situácia skomplikuje, čo nakoniec povedie k celkovému predraženiu a zníženiu kvalitu softvéru. Čo je samozrejme horšie ako hneď na začiatku povedať *nie*, aj keď sa to viacerým ľuďom nemusí páčiť.

Najlepšie je úplne sa odpútať od finančných implikácií projektu, jedine tak môžete mať nenarušený pohľad na skutočné nároky. Potom je možné znížiť požiadavky v prípade že klient si nemôže dovoliť kompletný systém, treba pritom ale

pamätať na to, že je veľká pravdepodobnosť dopracovania odstránených častí v budúcnosti.

Nedostatočné testovanie

Toto úzko súvisí so sledovaním postupu projektu. Zo zväčšovaním implementovaných častí projektu sa zväčšuje aj pravdepodobnosť zanesenia rôznych chýb. Čím neskôr sa na chybu príde tým viac úsilia si vyžaduje jej odstránenie. Je to tým že čím ďalej tým viac častí projektu je touto chybou ovplyvnených. Neskôr je pri odstraňovaní chyby nutné upraviť všetky takéto časti.

Rovnako ako včasné reportovanie zaostávajúcich činností je dobré priebežne sledovať a testovať stav plnenia funkčných požiadaviek na výsledok projektu. Čím skôr sa odhalia problémy a budú sa riešiť, tým lepšie. Dobrá protipožiarna ochrana je vždy lepšia ako schopní požiarnici.

Netreba sa báť použitia automatických systémov testovanie, ktoré už momentálne sú na dostatočne vysokej úrovni a ponúkajú veľmi dobrú podporu projektu [4]. Hlavne s využitím nástrojov na sledovanie verzií sú nevyhnutnou súčasťou každého väčšieho projektu, pretože zvyšujú transparentnosť a pomáhajú rýchlo odhaliť chyby a presnú zmenu, ktorá k nim viedla. Manuálne testovanie má stále svoju úlohu, ale je vhodné skôr na testovanie užívateľského rozhrania. Čo sa týka manuálnych testov tak by ich určite nemali robiť tímoví vývojári, ktorí sú ovplyvnení svojím pohľadom, ale špeciálna skupina, ktorá je oboznámená s požiadavkami používateľa.

Záver

V tomto článku som zhrnul a popísal niekoľko chýb, ktorých sa môže dopustiť projektový manažér, a ktoré môžu skomplikovať, prípadne úplne zastaviť softvérový projekt. Dúfam že tento krátky prehľad niekomu pomôže sa zamyslieť, možno aj predísť chybám a úspešne zvládnuť projekt.

Použitá literatúra

1. Brooks, F.: The mythical man-month: Essays on Software Engineering, 20th Anniversary Edition. Addison Wesley, 1995. (ISBN: 0-20-183595-9)
2. Charette, R.N: Why software fails, In: IEEE Spectrum, Sep, 2005
3. Gaitros, D.A.: Common Errors in Large Software Development Projects, In: *Crosstalk, The journal of defense software engineering*, Mar, 2004.
4. Hurley, F.: Seven Tips for Keeping Software Development Projects Healthy, In: *IT Professional*, Jul-Aug, 2002, 60-63.
5. Marasco, J.: Software Development Edge, The: Essays on Managing Successful Projects, Addison Wesley, April, 2005, (ISBN: 0-32-132131-6)
6. May, L.J.: Major Causes of Software Project Failures, In: *Crosstalk, The journal of defense software engineering*, Jul, 1997.

7. Verner, J.M., Cerpa, M.: Australian Software Development: What Software Project Management Practices Lead to Success?. In: *Proc. of the 2005 Australian Software Engineering Conference,* 2005.

Annotation

Errors and software project management

Everybody knows that people tend to make mistakes, but no one can predict when and what exactly will it be. During last few decades new techniques are proposed to reduce errors and increase effectiveness of software development. Despite all efforts, many software projects still fail. This article tries to shortly describe and explain few reasons why, and gives a summary of what can we do to cope with.

Zlepšovanie produktivity softvérových tímov

PAVOL DRAGÚŇ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
dragun01@student.fiit.stuba.sk

Abstrakt. Ak chce firma uspieť v dnešnom konkurenčnom prostredí, je potrebné, aby sa snažila dosiahnuť minimálne rovnako kvalitný výsledok, ale za kratší čas a menšie náklady ako konkurencia. Táto práca sa zaoberá problematikou zvyšovania produktivity softvérových tímov, ktorej cieľom je dosiahnuť práve lepšiu efektívnosť práce a kvalitnejšie výsledky. Snaží sa načrtnúť základné faktory, ktoré ovplyvňujú produktivitu a odhaľuje spôsoby a možnosti ako ju zvyšovať. Venuje sa hlavne koordinácii prác, pričom bližšie analyzuje dôvody vedúce k potrebe dobrej koordinácie a naznačuje spôsoby riešenia tohto problému. S koordináciou súvisí aj komunikácia v tíme, ktorá hrá tiež dôležitú úlohu z pohľadu produktivity. Práca sa okrem toho zaoberá aj faktormi vplývajúcimi na výkon jednotlivca, nakoľko výkon a produktivita celého tímu závisí aj od výkonov jeho členov. Výkon jednotlivca je ovplyvnený jeho zodpovednosťou, prístupom k práci, schopnosťami, podmienkami, v ktorých pracuje a motiváciou.

Úvod

Technológia a jej prienik do všetkých oblastí nášho života je už nevyvrátiteľným faktom. Ľudia pomaly prekonali prvotný strach z neznámeho a uvedomujú si, že technológia im môže výraznou mierou uľahčiť život. Umožňuje im rýchlejšie získavať informácie, odbremeňuje ich v istej miere od manuálnej práce, vyrovnáva trhové rozdiely spôsobené zemepisným rozmiestením, zefektívňuje procesy v podnikaní a tiež zvyšuje cenovú transparentnosť trhu. Dôkazom zvýšeného záujmu o softvér a technológie ako také je nárast zákazníckych požiadaviek smerom k producentom a zvýšený dopyt po ich výrobkoch a službách.

Proces vývoja softvéru je zložitý a dôležitú úlohu v ňom hrajú ľudia a ľudská kreativita, ktorú zatiaľ nie je možné v plnej miere nahradiť výpočtovou technikou. Poznáme isté prístupy napríklad generovanie kódu alebo celých programov, ktorých snahou je automatizovaná tvorba softvéru, no ich použitie je v súčasnosti obmedzené. Mnohé projekty si navyše vyžadujú istú dávku nielen kreativity ale aj inovácie, a nie je možné v plnej miere uplatniť len skúsenosti a poznatky z minulosti. To je jedným

Zlepšovanie produktivity softvérových tímov, január 2006, s. 83-89.

z dôvodov, prečo softvérové časti systému neustále spôsobujú predlžovanie projektov, zvyšovanie nákladov a v konečnom dôsledku aj nespokojnosť zákazníka.

Zákazníci požadujú stále lepšie a komplexnejšie riešenia za nižšiu cenu a kratší čas. Tvorba takýchto rozsiahlych riešení si vyžaduje nasadenie veľkých tímov ľudí. Ak chce firma uspieť v dnešnom konkurenčnom prostredí je potrebné, aby sa snažila dosiahnuť minimálne rovnako kvalitný výsledok, ale za kratší čas a menšie náklady ako konkurencia. Keďže v dnešnej dobe už táto práca nie je len záležitosťou jedného – dvoch ľudí, je o to dôležitejšie vedieť, ako riadiť, koordinovať a viesť tím k tomu, aby sa vývoj softvéru čo možno najviac zefektívnil. Jednou z možností ako dosiahnuť efektívnu prácu a kvalitnejšie výsledky je zvyšovanie produktivity softvérových tímov.

Piliere softvérového manažmentu

Zlepšenie produktivity je postavené na troch základných pilieroch softvérového manažmentu [3]:

- technológia,
- ľudia a
- proces.

Technológia

Zlepšovaniu technológií používaných pri tvorbe softvéru sa v posledných rokoch venovala veľká pozornosť a nemalé prostriedky. Výsledky tohto snaženia sa už dostavili v podobe kvalitných kompilátorov, ktoré zrýchľujú a zefektívňujú programovanie. Robustné pomocné nástroje umožňujú jednoduché nachádzanie, odstraňovanie chýb a ladenie systému. Boli tiež vyvinuté riešenia pre manažment projektov a pre podporu komunikácie v tíme. Kvalita procesu vývoja sa výrazne zlepšila vďaka zavedeniu nových noriem a metrík pre meranie kvality procesov.

Ľudia

Ako som už v úvode spomenul, pri tvorbe softvéru má nezastupiteľnú úlohu ľudský faktor. Žiadna z výhod ponúkaných modernými technológiami alebo kvalitným procesom vývoja nemôže byť plne využitá bez kompetentnej pracovnej sily.

Investovanie do ľudských zdrojov si vyžaduje dlhodobé plánovanie a výsledok sa dostaví až po dlhšej dobe. Často krát sa stáva, že zamestnanci po čase odídu a vynaložené úsilie a peniaze na ich zaškolenie sa zamestnávateľovi nevrátia. Firmy sa preto snažia zamestnávať ľudí s bohatými skúsenosťami v danej oblasti, do ktorých už nie je potrebné toľko investovať. Inou formou prevencie je písomné zaviazanie sa zamestnanca, že zostane v danej spoločnosti minimálne vopred určenú dobu. Táto doba je stanovená tak, aby dokázal svojou prácou v istej miere kompenzovať náklady firmy spojené s jeho zaškolením. V prípade, že by chcel odísť skôr, musel by ich všetky sám uhradiť.

Proces

Tretím spôsobom ako dosiahnuť stúpanie produktivity je zjemnenie a zušľachtenie samotného vývojového procesu. Bez správneho vývojového procesu bude softvérový tím vytvárať riešenia chaoticky a neefektívne, čo má za následok nízku produktivitu a slabú kvalitu systému. Proces môže byť zlepšený vhodným výberom procesného modelu, ktorý je závislý od veľkosti a typu projektu.

Koordinácia v tíme

Ak chceme zvyšovať produktivitu opierajúc sa o druhý spomenutý pilier [3], musíme identifikovať faktory, ktoré vplývajú na prácu ľudí v tíme. Na prvom mieste je potrebné určite spomenúť koordináciu, s čím veľmi úzko súvisí aj komunikácia v tíme.

Cieľom koordinácie je efektívne využívanie času jednotlivých členov tímu tak, aby žiaden z nich nemusel čakať na výsledky práce ostatných kolegov. Medzi najznámejšie prístupy riešenia koordinácie v tíme patria [1]:

- Big Bang,
- častá integrácia a pravidelná synchronizácia a
- koordinácia riadená poruchami.

Big Bang

Pri tomto prístupe nastáva celá integrácia na konci projektu a preto je typický pre vodopádový model vývoja softvéru. Každý člen tímu vykonáva svoju úlohu, resp. časť projektu s tým, že integrácia jednotlivých častí sa uskutoční až na záver. Tento prístup je vhodný pre malé a dobre definované projekty. Pri veľkých projektoch môžu pri integrácii nastať nečakané problémy spojené s nežiaducou interakciou softvérových komponentov. Použitie prístupu Big Bang pre veľké projekty je preto veľmi nákladné.

Častá integrácia a pravidelná synchronizácia

Tento prístup sa vyznačuje častým integrovaním modulov. Tím je nútený komunikovať nielen o integrácii, ale aj o všetkých problémoch, ktoré nastanú. Častá integrácia modulov, periodická tímová komunikácia a systémová integrácia sú uskutočňované tak, aby zabezpečili kvalitu produktu už počas jeho tvorby. Tento prístup je využívaný vo veľkých firmách, kde je kladený veľký dôraz na kvalitu.

Zaujímavou môže byť otázka časovania integrácie. Mnohé spoločnosti riešia túto otázku *ad hoc*. Na začiatku projektu bývajú však intervaly kratšie, postupne sa predlžujú a na konci projektu sa opäť skracujú.

Koordinácia riadená chybami

Tento prístup je sofistikovanejší oproti dvom spomenutým vyššie. Jeho snahou je vykonať integráciu vtedy, keď je to pre daný projekt najvhodnejšie. Poskytuje tak väčšiu flexibilitu a zachytáva dynamiku procesu vývoja produktu.

Tak ako prechádza projekt rôznymi štádiami, mení sa aj potreba integrácie a koordinácie. Niekedy je potrebné kvôli odstráneniu vzniknutých problémov organizovať koordinačné stretnutia častejšie, inokedy je vhodné nechať vývoj nerušene pokračovať. S použitím vyspelých nástrojov podporujúcich vývoj a riadenie projektov je možné získať relevantné metriky a pomocou nich tak určovať aktuálny stav projektu. Na základe zisteného počtu chýb a niektorých iných metrík môžu manažéri naplánovať integráciu na najvhodnejší čas. Projekt je v raných štádiách alebo verziách omnoho viac náchylný na výskyt chýb než v neskorších fázach. Tieto vlastnosti sú však závislé napríklad od veľkosti a zložitosti projektu, od počtu ľudí v tíme, od ich skúseností a znalostí, alebo od vývojového prostredia a platformy.

V praxi sa tento prístup oproti periodickému ukazuje ako efektívnejší, no neprodukuje také jasné a prehľadné administratívne výsledky. Je vhodné ho použiť pre triedu systémov, s vývojom ktorých už majú pracovníci dostatok skúseností.

Komunikácia v tíme

Pri zlepšovaní produktivity softvérového tímu ide ruka v ruke spolu s koordináciou aj komunikácia [4]. Tím môžeme charakterizovať ako skupinu ľudí, ktorí sa podieľajú na dosiahnutí spoločného cieľa. Z tohto pohľadu je nevyhnutné, aby každý člen tímu mohol otvorene vyjadriť svoj názor. Často krát sa stáva, že noví členovia tímu majú problém sa do neho začleniť, nakoľko ich straší a skúsenejší kolegovia im nedajú dostatok priestoru na prezentáciu svojich názorov. Cítia sa byť preto nechcenou súčasťou tímu, začínajú mať komplexy a ich produktivita automaticky klesá.

Každý, kto sa zúčastňuje projektu, musí byť pripravený prijímať, ale aj vysielať informácie a musí chápať spôsob komunikácie, do ktorej je zapojený individuálne alebo ako člen tímu [2]. Na efektívnosti komunikácie sa podieľa odosielateľ informácie ale aj jej prijímateľ. Odosielateľ by sa mal prostredníctvom spätnej väzby snažiť zistiť, či prijímateľ informáciu pochopil. Preto je dôležité, aby zvolil adekvátny tón a spôsob vyjadrovania. Musí byť dostatočne zdvorilý, empatický a mal by byť pripravený na to, že prijímateľ ho nemusí pochopiť hneď na prvý krát. Na druhej strane by mal prijímateľ preukázať rešpekt voči odosielateľovi. Nesmie ho prerušovať a musí prejaviť záujem o porozumenie predávanej informácie. Ak jej neporozumie hneď, mal by to dať vhodnou formou najavo odosielateľovi.

Brainstorming

Vhodnou formou, ako riešiť nejaký problém alebo ako spojiť rôzne uhly pohľadu členov tímu na predostretú vec, je *brainstorming*. Jedná sa o riadenú diskusiu, ktorej cieľom je v slobodnej atmosfére a za účasti všetkých členov tímu vyprodukovať čo najviac myšlienok. Vedením diskusie je poverený jeden člen tímu, moderátor.

Postupne dáva rovnaký priestor všetkým zúčastneným, aby mohli vyjadriť svoj postoj. Žiaden názor pri tom nie je hodnotený, ani kladne ani záporne. Člen tímu poverený zapisovaním zapisuje všetky myšlienky. Na konci spolu prejdú všetkým, čo vyprodukovali a začnú jednotlivé postoje a názory spolu hodnotiť a vyberať tie najlepšie. Tým, že má každý právo sa vyjadriť a nikto ho pritom nekritizuje, nemôže sa stať, že by sa niekto cítil byť v tíme menejcenný.

Výkon jednotlivca

Softvérový tím pozostáva z viacerých ľudí, individualít. Ak chceme zvýšiť produktivitu tímu, je potrebné zamerať sa aj na výkon jednotlivcov [2], ktorí tento tím tvoria.

Výkon jednotlivca je ovplyvnený najmä:

- jeho prístupom k práci,
- jeho schopnosťami,
- pracovnými podmienkami a
- motiváciou.

Prístup k práci

Koordinácia ako taká nemá veľký zmysel, pokiaľ nebudú členovia tímu pristupovať k práci zodpovedne. Vyžadovanie zodpovednosti od členov tímu nezahŕňa len kvalitné a profesionálne odvedenie práce na pridelených úlohách, ale aj plnenie termínov tak, aby ostatní členovia nemuseli čakať na výsledky týchto úloh.

Každý človek má vlastný prístup k práci, ktorú sa snaží vykonávať viac či menej zodpovedne. Zodpovedný prístup je možné dosiahnuť napríklad kontrolou medzivýsledkov práce. Tu však treba vycítiť správnu mieru kontroly, aby zamestnanec nedostal pocit, že je striktne kontrolovaný, ba až sledovaný. Pri kontrole medzivýsledkov sa otvára priestor na prípadné usmernenie, či správne mienenú radu, ktorá by napomohla rýchlejšiemu dokončeniu úlohy, ako aj zvýšeniu kvality celkovej práce.

Schopnosti

Každý človek má nejaké vrodené schopnosti, potenciál, ktorý môže ďalej rozvíjať. Počas života získava ďalšie schopnosti a znalosti, teoretické štúdiom a praktické prácou.

Firma môže zlepšovať schopnosti a znalosti svojich zamestnancov napríklad prostredníctvom školení a kurzov. Takáto investícia do ľudských zdrojov je však veľmi drahá a trvá dlhý čas kým sa spoločnosti vráti späť. Firmy sa preto snažia vyberať ľudí, do ktorých sa oplatí investovať. S týmto zámerom konajú fyzicky aj psychicky náročné pohovory, kde dokonale preveria schopnosti, znalosti, komunikatívnosť, empatiu, sebavedomie či vytrvalosť uchádzačov. Pri takto vybratých zamestnancoch má firma

istotu, že majú dobré predpoklady pre profesionálny rast a tak môžu byť pre ňu neskôr prínosom.

Podmienky

Pracovné podmienky sú ďalším dôležitým faktorom ovplyvňujúcim výkon jednotlivca. Je dôležité, aby firma zabezpečila pokojné a kreativitu podporujúce prostredie. Je rozdiel, či človek pracuje v príjemných kancelárskych priestoroch, ktoré sú skrášlené napríklad kvetmi, alebo v deprimujúcom neupravenom prostredí bez okien.

Okrem prostredia ako takého je dôležité, aká atmosféra v ňom vládne. Zo skúseností je zrejmé, že v priateľskom prostredí sa pracuje omnoho lepšie, ako v prostredí nabitom nevraživosťou. Zamestnávateľ by sa mal preto starať o utužovanie vzťahov medzi zamestnancami napríklad prostredníctvom firemných večerí, kde sa môžu všetci lepšie spoznať.

Motivácia

Koontz a Weihrich definujú motiváciu nasledovne [2]: *„Motiváciu možno chápať ako určitý reťazec nadväzných reakcií: pocit potreby dáva vzniknúť zodpovedajúcim prianiam alebo cieľom, ktoré vytvárajú určité napätie (z dôvodu nesplnených cieľov) a vedú k vzniku aktivít smerujúcich k dosiahnutiu cieľov. Konečným dôsledkom tohto procesu je uspokojenie“*.

Znalosť motívov umožňuje pochopenie konania človeka. Každý človek má v zmysle Maslowovej teórie isté potreby, ktoré sa snaží uspokojiť. Môže ísť o fyzické, sociálne potreby, potreby uznania alebo sebarealizácie. Potreby sú podnetom konania človeka s cieľom ich uspokojenia a je ich možné využiť aj v oblasti pracovných pomerov [4]. Niekoho motivuje finančné zabezpečenie, možnosť postupu a kariérneho úspechu, iného môže motivovať pocit uznania za dobre vykonanú prácu. Je úlohou dobrého manažéra, aby poznal jednotlivých členov tímu, aká je ich pyramída potrieb a čo ich môže motivovať k lepšej a zodpovednejšej práci. Motivácia jednotlivcov môže mať zároveň pozitívny vplyv na celý tím.

Motivácia je zároveň prevenciou proti „odfláknutiu“ práce, problémom s dochádzkou do zamestnania, zmeškaniu termínov a v konečnom dôsledku proti problémom s produktivitou a kvalitou práce.

Záver

Vývoj softvéru je zložitý proces, ktorého neodmysliteľnou súčasťou sú ľudia. Ak chce firma uspieť v dnešnom konkurenčnom prostredí, je potrebné, aby sa snažila dosiahnuť minimálne rovnako kvalitný výsledok, ale za kratší čas a menšie náklady ako konkurencia. Jednou z možností ako dosiahnuť lepšiu efektivitu práce a kvalitnejšie výsledky je zvyšovanie produktivity tímov.

Medzi hlavné faktory ovplyvňujúce produktivitu softvérových tímov patria koordinácia prác v tíme, komunikácia ale aj výkon jednotlivých členov. Výkon

jednotlivca je pritom závislý od jeho zodpovednosti, prístupu k práci, schopností, podmienok, v akých pracuje a v neposlednom rade od motivácie.

Táto práca si nekládla za cieľ vyčerpať celú tému. Jej cieľom bolo hlavne načrtnúť tieto základné faktory vplývajúce na produktivitu a zároveň poukázať na spôsoby a možnosti jej zvyšovania.

Použitá literatúra

1. Chiang, I. Robert, Mookerjee, Vijay S.: Improving Software Team Productivity. In Communications of the ACM, Vol. 47, No. 5 (May 2004) 89-93.
2. Bieliková, M.: *Softvérové inžinierstvo. Princípy a manažment*. Bratislava, Slovensko, 2001.
3. Landis, L., McGarry, F., Waligora, S., et. al.: Manager's handbook for software development (Revision 1). NASA Software Engineering Laboratory, SEL-84-101, (Nov. 1990).
4. Unger, B., et. al.: Improving Team Productivity in System Software Development. In Proceedings of the fifteenth annual SIGCPR conference, Virginia, 104-115.

Annotation

Improving software team productivity

In order to beat the competitive firms is necessary to produce qualitatively comparable results, but in shorter time and with lower costs. This paper is concerned with improving software team productivity in order to achieve better work effectiveness and qualitatively better results. It is pointing out some basic productivity affecting aspects and discovers some ways how to improve it as well. Therefore we give team coordination due attention and we describe some ways how to solve coordination problems. With coordination hangs together communication in team as well. This paper deals also with team member productivity and with factors that affect it. The team member productivity influences the whole team productivity. It depends on his conscientiousness, skills, working conditions and motivation.

Manažment konfliktov v tíme

PETER KASAN

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava

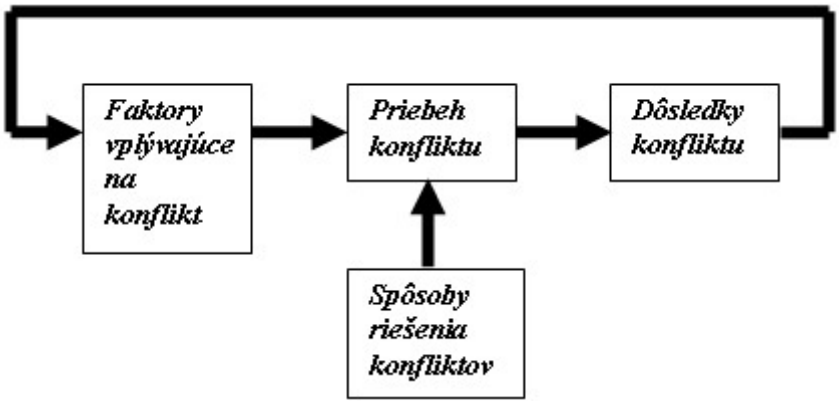
Abstrakt. Problematika tímov a vzájomné vzťahy ich členov sú v dnešnej „tímovej" dobe stále viac aktuálne a skúmané. Vzťahy v skupine a ich dynamika síce patria do oblasti skupinovej psychológie, ale každý dobrý manažér alebo tím líder by sa mal s nimi oboznámiť. Jedným s dôležitých prvkov týchto vzťahov, na ktorý treba upriamiť viac pozornosti, je konflikt. Je dôležité pochopiť čo to je, ako vzniká, aké ma fázy a ako ho treba riešiť, prípadne mu úplne predchádzať. Tieto otázky sú riešené v prvej časti tejto eseje. Druhá časť sa sústreďuje na spôsoby minimalizovania faktorov, ktoré spôsobujú konflikt a to špecificky pre dve roly v tíme – programátorov a testerov. Sú tu popísané niektoré situácie, ktoré vytvárajú napätie medzi spomínanými skupinami a tiež možný postup ako im predchádzať.

Úvod

V dnešnej dobe, kedy sú oblasti znalostí príliš rozsiahle a kedy začínajú prevládať projekty využívajúce mnohé tieto oblasti naraz, nie je možné aby boli takéto projekty zvládané jednotlivcami. Preto vznikajú tímy ľudí, kde sa každý špecializuje na určitú oblasť znalostí alebo má pridelenú úlohu v rámci tímu. Keď sa spomína pojem tím treba ho bližšie popísať. Tímy sú dynamicky interagujúce entity, ktoré sa skladajú z navzájom sa ovplyvňujúcich častíc. Akokoľvek sa zvonku zdá tím ako kompaktná entita, stále je zložená z častíc, jednotlivcov, ktorí majú svoje potreby, povahu a ciele, pričom tieto sa veľmi často nezhodujú s potrebami iných častíc, jednotlivcov. A ako to v prírode býva, rozdiely často vyvolávajú interakciu. Táto interakcia môže byť pozitívna, ale veľmi často je negatívna a prerastá do konfliktov.

Konflikt

Konflikt možno definovať ako nezhodu v cieľoch alebo záujmoch jednotlivcov či skupín. Taktiež by sa dal definovať ako proces, kedy jedna strana vníma, že jej záujmy alebo ciele sú v protiklade, alebo sú negatívne ovplyvňované záujmami a cieľmi druhej strany [1]. Možno ho znázorniť ako cyklus, v ktorom majú jednotlivé položky stanovenú svoju úlohu (pozri **Obr. 2:** Konflikt ako cyklus [1]).



Obr. 2: Konflikt ako cyklus [1]

Každý kto sa už dostal s niekým do akéhokoľvek konfliktu alebo vystupoval len ako nezúčastnená strana vie, že tento nevzniká len tak sám od seba a bez príčiny. Vždy na začiatku existujú nejaké faktory, ktoré vo väčšej alebo menšej miere podkurujú pod kotlom. A ako to býva, problémy nenechajú na seba dlho čakať a konflikt je na svete. Tento nemôže trvať večne aj keď často trvá veľmi dlho (prípad storočnej vojny). Nakoniec sa skôr či neskôr síce vyrieši, ale záleží len od spôsobu jeho riešenia aké veľké škody napácha (prípadne aký prínos z neho vznikne). Toto je dôsledok konfliktu, ktorý zase spätne ovplyvňuje faktory vzniku. Žiarivým príkladom je nikdy nekončiaci kruh „oko za oko, zub za zub“. Každopádne, spôsoby riešenia a minimalizovanie počiatočných faktorov môžu znížiť počet obehov po tomto bežeckom okruhu.

Faktory vplývajúce na konflikt môžu byť rozdelené do viacerých kategórií. Medzi najdôležitejšie z nich patria nasledujúce:

1. ***Faktory individuality***: osobnosť - povaha, vzdelanie , skúsenosti, potreby, záujmy, ciele individuálneho člena tímu.
2. ***Faktory tímu***: veľkosť tímu, vedenie tímu, história predchádzajúcich konfliktov, pozície v tíme.
3. ***Faktory projektu***: charakteristika vyvíjaného systému a jeho dôležitosť, dostupnosť zdrojov, časový tlak a obmedzenia na projekt.

Samotný konflikt môže vznikať na dvoch úrovniach: Na úrovni ľudí (osobností) a na úrovni rolí (úloh v tíme).

Na úrovni ľudí vznikajú konflikty najmä v dôsledku rôznych osobností v tíme. Vo veľkom množstve prípadov sa v skupine nájdu ľudia (jeden alebo viac), ktorí majú z hľadiska ostatných členov konfliktnú povahu aj keď si to veľmi často neuvedomujú. Jedná sa napríklad o ľudí, ktorí majú pocit, že všetko vedia najlepšie a vždy musia mať pravdu. S takýmito ľuďmi sa nielen ťažko diskutuje, ale aj pracuje. Veď prečo by ste mali vôbec vymýšľať nejaké riešenie, keď to ich je to najlepšie? Takisto veľmi „obľúbené“ povahy sú ľudia, ktorí už z princípu musia všetko a všetkých kritizovať a musia mať vo všetkom posledné slovo. Týchto pár príkladov tvorí len malú časť možných osobností (povahových čŕt), ktoré majú vysoký podiel na vzniku konfliktov

medzi členmi tímu. Zriedkavé prípady nezhôd môžu vznikať aj na úrovni sympatií medzi členmi tímu, kedy si vyslovene niektorí členovia navzájom nesadnú a sú si navzájom nesympatický. Tento prípad je ale zriedkavý a týka sa väčšinou len dvoch individualít.

Na úrovni rolí alebo úloh v tíme vznikajú konflikty najmä z hľadiska odlišného pohľadu na projekt, z hľadiska dostupných zdrojov, časového tlaku a tiež z hľadiska sledovaných cieľov pri projekte. Možno tu identifikovať konflikty medzi testermi a programátormi, medzi manažérmi a analytikmi, medzi návrhármi a programátormi, atď. Ďalšia časť (Hľadanie faktorov vplývajúcich na konflikty medzi testermi a programátormi a ich riešenie) sa bude sústreďovať podrobne práve na identifikáciu možných faktorov vplývajúcich na jeho vznik a ich minimalizovanie.

Medzi spôsoby ako sa vysporiadať s búrlivou situáciou medzi členmi tímu patria postupy a metódy, ktorých aplikovaním môžeme získať rozriešenie konfliktu. Patria sem:

1. ***Vnútenie názoru*** – je to spôsob riešenia konfliktu, kedy sa individuálny člen alebo skupina tímu snažia vnútiť (presadiť) svoj názor alebo riešenie druhej strane. Je to riešenie výhra–prehra. Takýto spôsob riešenia môže, ale nemusí viesť k negatívnym dôsledkom konfliktu, kedy porazená strana môže prepadnúť znechuteniu vedúcemu k zníženiu efektivity práce.
2. ***Prispôsobenie*** – je to opačný spôsob ako predchádzajúci. V tomto prípade dochádza k obetovaniu svojich potrieb a požiadaviek v záujme uspokojenia druhej strany. Takisto sa jedná o riešenie výhra–prehra.
3. ***Kompromis*** – jedná sa o riešenie výhra–výhra, kedy sa obidve konfliktné strany snažia dospieť k určitému kompromisu v spornej oblasti. Tento spôsob je asi najvhodnejší z hľadiska dôsledkov konfliktu, nakoľko majú obidve strany pocit istej výhry. Je to najvhodnejší spôsob v situáciách, kedy nie je možné uspokojiť všetky požiadavky zúčastnených strán.
4. ***Riešenie problému*** – je to spôsob, ktorý sa snaží nájsť možnosť, ako plne uspokojiť záujmy všetkých zúčastnených strán. Jedná sa o riešenie výhra–výhra. Používa sa v prípadoch, keď existuje možnosť, že také riešenie existuje a treba ho len nájsť. Tento spôsob je tiež veľmi vhodný z hľadiska dôsledkov konfliktu.
5. ***Vyhýbanie sa konfliktu*** – spôsob, kedy sú individuality alebo skupina ľahostajné k záujmom druhej strany a odmietajú sa podieľať na konflikte, čím sa vzdávajú zodpovednosti za riešenie[1]. Tento spôsob je najmenej vhodný zo všetkých spomenutých, nakoľko sa problém nerieši a dochádza k zníženiu efektivity práce. Použitie tohto spôsobu vedie veľmi často k negatívnym dôsledkom.

Dôsledky konfliktov možno rozdeliť na pozitívne a negatívne. Pozitívne dôsledky vznikajú vtedy, keď bola použitá vhodná metóda riešenia sporu. Vtedy sa dá povedať, že tento spor bol prínosom pre projekt, na ktorom pracuje tím. Negatívne dôsledky vznikajú pri nevhodnom riešení konfliktu a majú nepriaznivý vplyv na projekt a postup prác na ňom. Medzi pozitívne dôsledky patrí:

1. ***Prínos k projektu***: vylepšenie procesu, spokojnosť členov tímu, zníženie nákladov na riešenie, naplánovanie ďalšieho postupu.
2. ***Prínos k vyvíjanému systému*** – zvýšenie kvality a použiteľnosti.
3. ***Prínos k individualite (osobnosti)*** – zvýšenie efektívnosti, sebauspokojenie.

Medzi negatívne dôsledky patria najmä:

1. ***Negatívny prínos k projektu***: nekvalitný proces riešenia, nespokojnosť a rozčarovanosť členov tímu, zvýšenie nákladov na riešenie, nedostatočné naplánovanie ďalšieho postupu.
2. Negatívny prínos k vyvíjanému systému – zníženie kvality a použiteľnosti.
3. ***Negatívny prínos k individualite (osobnosti)*** – zníženie efektívnosti, rozčarovanie, apatia.

Dôsledky kladné alebo záporné spätnou väzbou ovplyvňujú faktory, ktoré sú zdrojom možných problémov. V prípade, ak sa nezhoda medzi zúčastnenými správne vyrieši, môže ovplyvniť vzťahy v tíme k lepšiemu a zredukovať pravdepodobnosť vzniku ďalšieho konfliktu. V prípade, ak sa vyrieši nesprávnym spôsobom, môže sa napätie v skupine prehlbovať a spory sa stanú častejšími. Preto je dôležité používať správny prístup ako zvládať vzniknutú krízovú situáciu.

Možnosťou ako zredukovať konflikty v budúcnosti nie je len aplikácia správneho riešenia sporu, ale aj minimalizovanie faktorov, ktoré ho môžu podporovať. Ak napríklad tím líder vidí, že má v tíme problematickú osobu, ktorá nevie vychádzať s ostatnými členmi skupiny bez toho aby v nich vždy nevyvolala búrku negatívnych pocitov, môže to riešiť či už vymenením tejto osoby alebo pristúpením k iným opatreniam riešiacim tento problém (napríklad sa môže porozprávať s danou osobou a upozorniť ju na existujúci problém).

Aspektom predchádzania vzniku konfliktov by som sa rád podrobnejšie venoval v ďalšej časti, pretože podľa môjho názoru existuje väčšie množstvo negatívnych sporov ako pozitívnych. A práve dopad tých negatívnych, aj keď v mnohých prípadoch vhodne riešených, má nepriaznivý vplyv na celý tím. Odhliadnuc od možných škôd, ktoré takéto spory môžu napáchať, vždy stojí menej úsilia a času odstrániť na začiatku neželané faktory, ako neustále riešiť konflikty, ktoré vyvolávajú.

Predchádzanie konfliktov: Prínos alebo strata?

Táto otázka je na mieste, nakoľko často je konflikt práve tou hybnou silou, ktorá posúva riešenie problému vpred. Zoberme si situáciu, že v tíme vznikol problém. Na jeho riešenie sa zíde celý tím a všetci členovia sa snažia nájsť cestičky a úskalia jeho riešenia. Skupina sa skladá z rôznych ľudí s rozdielnym spôsobom myslenia a prístupom k veciam, pričom každý z nich bude hľadať vlastné cestičky. Jeden sa bude snažiť prekážku preliezť, druhý podliezť a tretí ju jednoducho obíde. Práve

v tomto bode vzniká konflikt záujmov, ktorý treba riešiť. Vhodným spôsobom (napríklad usmerňovanou civilizovanou diskusiou, pri ktorej ľudia po sebe nehádžu okolitými predmetmi) sa dospeje k záveru, ktorý môže, a často aj je prínosom k riešeniu problému. Nakoľko pri spore má každá zo strán svoje argumenty často práve z týchto vyskočia potenciálne „kamene na ceste". Pokiaľ sa tieto identifikujú, zníži sa šanca, že sa tím na ceste k problému potkne.

Na druhej strane často existujú spory, ktoré nemajú vôbec žiaden prínos k akémukoľvek problému a len vytvárajú zlú atmosféru medzi členmi tímu. Často to bývajú úplne bezvýznamné problémy, ktoré vyplývajú z povahy a zvykov jednotlivých ľudí (ako spor o to, či má byť kávovar položený pri dverách alebo v strede kancelárie na stole) až po vážnejšie problémy, ktoré už môžu mať vážne následky. Takéto negatívne konflikty treba identifikovať a snažiť sa ich odstrániť už v zárodku.

Úlohou dobrého tím lídra je preto odlíšiť pozitívne konflikty od negatívnych. Tie ktoré sú pozitívne, treba nechať prepuknúť a vhodnými metódami riadiť ich priebeh, pričom záporné treba zastaviť, kým sa vôbec stačia rozvinúť, a to hlavne minimalizovaním faktorov, ktoré ich podnecujú. Preto odpoveď na otázku 'Prínos alebo strata?' nie je jednoznačná a záleží len na okolnostiach.

Ďalšou kapitolou by som rád nadviazal na túto problematiku a špeciálne na identifikovanie a odstraňovanie faktorov spôsobujúcich vznik negatívnych konfliktov medzi programátormi a testermi.

Hľadanie faktorov vplývajúcich na konflikty medzi testermi a programátormi a ich riešenie

Ako už bolo spomenuté, softvérový tím sa skladá z členov, ktorí majú pridelenú rolu (úlohu) v tíme, na základe ktorej majú rozdielny pohľad na projekt a sledujú rôzne ciele pri projekte. Tieto faktory nevyhnutne vedú k prepuknutiu konfliktu medzi členmi tímu majúcimi rozdielne úlohy. Z dlhodobého pozorovania boli vytipované niektoré viac alebo menej konfliktné páry rolí. Medzi tie viacej náchylné na konflikt patria aj dvojica programátor–tester. V rámci konfliktov rolí existujú ďalej rôzne úrovne (a k nim prislúchajúce faktory vzniku), na ktorých môže k týmto konfliktom dochádzať. Na základe otázok boli identifikované tieto tri základné úrovne [2]:

1. **Konflikty na úrovni procesu – faktory projektu**: nedostatok času, rôzne technické požiadavky.
2. ***Konflikty na úrovni ľudí – faktory individuality***: rozdielne charakterové črty, naštrbenie perfektnosti kódu.
3. Konflikty na úrovni organizácie – faktory tímu: pozícia v tíme.

Ako si možno všimnúť, rozdelenie konfliktu do vrstiev [2] identifikuje pre každú vrstvu faktory vzniku, ktoré možno zaradiť do členenia, ktoré bolo popísané v kapitole Konflikt, a vychádza z [1]. Môj názor je taký, že všeobecný model a identifikácia atribútov [1] je postavený na podobných konkrétnych štúdiách ako [2] a vystihuje reálnu situáciu, ktorá panuje v oblasti konfliktov v tíme. Zároveň ukazuje, že všetky

reálne spory v tímoch medzi ktorýmikoľvek rolami majú rovnakú štruktúru ale rozdielne parametre.

Konflikt na úrovni procesu – faktor: *nedostatok času*

Čas tvorí jeden z troch kľúčových zdrojov pri akomkoľvek projekte, či už sa jedná o softvérový projekt alebo iný. V dnešnej dobe keď sa softvérové projekty týkajú systémov, ktoré majú byť značne komplexné, bezporuchové, efektívne, udržovateľné a musia spĺňať mnoho ďalších vlastností, sa čas pri projektoch týkajúcich sa týchto systémov stáva čoraz vzácnejším artiklom. Projekt treba preto podrobne naplánovať a jednotlivým fázam prideliť čas, za ktorý by mali byť zrealizované. Často sa pritom stáva, že fáza testovania je podhodnotená a počíta sa na ňu s minimálnym možným časom. Lepší prípad nastane, keď sa tento (síce krátky) čas na testovanie dodrží. Horšie je to už ale v prípade, keď má niektorá z predchádzajúcich fáz oneskorenie a potom sa termíny dobiehajú na úkor testovania, ktoré je väčšinou na konci procesu.

A tu sa nachádza kameň úrazu, na ktorom začínajú konflikty medzi testermi a programátormi. Nakoľko väčšinou fáza implementácie (tvorby kódu) tesne predchádza testovaniu, sú väčšinou testeri závislí od programátorov. Môžu začať testovať až vtedy, keď im programátori dodajú potrebné výstupy (moduly, komponenty, atď.). Veľmi často sa stáva, že programátori si nechávajú dodanie ich výstupov na poslednú chvíľu a testeri sú potom časovo tlačení, aby stihli zrealizovať testovanie do požadovaného termínu. Faktom tiež je, že programátori ako nie koncový článok reťaze procesu, nie sú priamo vystavení koncovému termínu odovzdania (tak ako testeri) a preto u nich nie je až také silné nutkanie dodržať pridelený čas. Často sa stáva, že sa fáza implementovania systému natiahne na úkor práve testovacej fázy. Tieto časové prešľapy vyvolávajú u testerov pocit naliehavosti, čím rastie napätie a stres a konflikt je na svete.

Minimalizovanie faktorov spôsobujúcich takýto typ konfliktov sa môže realizovať viacerými spôsobmi. Jedným z nich je pridelenie dostatočného času na fázu testovania. Manažér alebo tím líder si musí uvedomiť dôležitosť testovania v celom procese a adekvátne tomu vyčleniť aj potrebné množstvo času. Druhou možnosťou je plánovať priebežné testovanie, ktoré overuje systém cyklicky. Takto sa odhalí veľká časť chýb už počas vývoja systému a fáza testovania neostane na koniec.

Možnosťou ako riešiť časové prešľapy implementačnej fázy na úkor testovacej, je vyčleniť istý časový balík, ktorý nebude pridelený žiadnej fáze projektu ale bude uložený v záložnom rezervoári pre prípad potreby. Je to vhodnejšie ako rozdeliť tento čas medzi fázy projektu, nakoľko je známe, že ktorákoľvek projektová fáza trvá minimálne toľko času koľko ho dostala prideleného. Preto ak existuje záložný čas a niektorá fáza presiahne pridelený časový interval môže jej byť stále pridelený čas z rezervoára. Takto nebude dochádzať k využívaniu testovacej fázy na dobiehanie časových prešľapov v implementačnej fáze a testeri budú mať dostatočné množstvo času na testovanie. Nebudú tlačení časom a nebude rásť stres spôsobujúci konflikty medzi nimi a programátormi.

Konflikt na úrovni procesu – faktor: *používateľské vs. technické požiadavky*

Druhou príčinou konfliktov na úrovni procesu je odlišný pohľad programátorov a testerov na proces. Programátori sa sústredia na kreatívne postupy, snažia sa nachádzať nové možnosti riešenia a nové cesty ako riešiť zadaný problém. Sú orientovaní prevažne na technickú stránku systému a tomu podriaďujú aj svoje ciele. Na rozdiel od nich testeri majú za úlohu testovať systém tak ako sa bude používať. Znamená to, že sa musia čiastočne postaviť do roly používateľa a pozerať sa na systém jeho očami. Z toho vyplýva, že testeri sú orientovaní na funkcionálnu stránku systému a tomu podriaďujú svoje ciele. A práve toto je príčinou vzniku konfliktu. Veľmi často sa totiž stáva, že programátori použijú nový prístup alebo inováciu, ktorá sa nezhoduje s používateľskými požiadavkami, čo nakoniec spôsobí problémy testerom. Z pohľadu programátora je to vylepšenie riešeného problému, ale z pohľadu testera komplikovanie jeho roboty.

Jedným z možných riešení je zadefinovanie spoločného cieľa pre jednotlivcov aj pre skupiny. Cieľ musí byť dostatočne motivujúci pre všetky skupiny, aby sa pre tento cieľ nadchli všetci a podriadili mu svoju činnosť. Nemal by to byť len cieľ, ktorý kladie dôraz na včasné dokončenie vyvíjaného systému. Ciele by mali byť motivujúce ako napríklad vyvíjanie kvalitného systému, ktorý bude bezporuchový a bude ho možné nasadiť v niektorej prestížnej oblasti a podobne. Druhou možnosťou, ako predchádzať konfliktom na tejto úrovni, je poskytnúť testerom a programátorom opačný pohľad na proces. Tento spôsob zahŕňa príležitostné výmeny rolí, medzi programátormi a testermi, čo im umožní vyskúšať si ako vidia proces ich kolegovia.

Konflikt na úrovni ľudí – faktor: *rozdielne charakterové črty*

Pri popise konfliktu bolo spomenuté, že tieto nevznikajú len na úrovni rolí v tíme (pri procese) ale tiež na úrovni ľudí. Každý človek má iné myšlienkové pochody a iné charakterové črty, pričom tieto môžu byť ovplyvnené (a často aj bývajú) profesiou alebo špecializáciou toho ktorého človeka. Takisto bolo dokázané, že ľudia si často vyberajú profesiu podľa ich charakteru (osobnosti). Napríklad, testeri sú často popisovaní ako pedanti sústredení na detaily, a skupiny, ktoré vytvárajú, sú popisované ako značne súdržné. Naopak programátori sa často popisujú ako ľudia kreatívni, temperamentní, individualistickí (nevytvárajúci pevné skupiny)[2]. Je to celkom pochopiteľné nakoľko si to vyžaduje ich špecializácia. Tieto charakterové črty pravdaže nemusia byť pravidlom (vždy sa nájdu výnimky). Dokonca by sa dalo povedať, že v dnešnej dobe individualita u programátorov ustupuje. Je to dané rôznymi tréningovými kurzami na vysokých školách, kde je aktuálny trend smerovať študentov k spolupráci v tíme. No nech je to ako chce, rozdielne charakterové črty často spôsobujú nedorozumenia, ktoré môžu prerásť až do konfliktu.

Prvou z možností ako minimalizovať negatívne konflikty na danej úrovni je trénovanie členov tímu (či už ide o programátorov, testerov ale aj ostatných) v metódach riešenia konfliktov. Tieto metódy rozvíjajú schopnosti členov tímu v oblasti komunikácie, riešenia problémov a mnohé ďalšie. Takéto schopnosti u členov tímu pomôžu značne minimalizovať negatívne dôsledky sporu.

Druhou možnosťou ako ich minimalizovať je posilovať vzťahy medzi jednotlivými skupinami tímu. Veľmi často sa napríklad stáva, že programátori a testeri spolu komunikujú len v prípade, ak nastane niekde problém alebo chyba. Takýto prístup nie je najvhodnejší z hľadiska budovania dobrých vzťahov. V záujme zlepšovania vzťahov medzi skupinami tímu (napríklad medzi programátormi a testermi) je vhodné realizovať spoločné akcie, kde sa členovia týchto skupín môžu stretnúť, zoznámiť sa a porozprávať sa. Dôležité je aby išlo o aktivity, ktoré členov tímu spájajú (či už ide o športové aktivity alebo posedenie pri pive).

Konflikt na úrovni ľudí – faktor: *naštrbenie perfektnosti kódu*

Je dané povahou človeka, že je hrdý na svoje výtvory. Z vlastnej skúsenosti viem, že u programátorov môže byť tento pud značne posilnený. Sú hrdí na svoj kód a často naň nedajú dopustiť. Na druhej strane úlohou testerov je hľadať chyby a upozorňovať na ne. Z povahy ich práce vyplýva, že sú hrdí na schopnosť odhaľovať chyby a sú radi keď nejakú odhalia. A tu je lokalizované epicentrum konfliktu. Perfektný kód verzus jeho spochybňovanie. Programátori často vnímajú snahu testerov nájsť chyby v ich kóde ako spochybňovanie ich schopností (aj keď to tak často nie je). Podľa môjho názoru by mali byť skôr radi, ak sa podarí odhaliť chyby ktoré spravili, aby sa z nich poučili a aby sa ich kód pri najbližšej príležitosti konečne priblížil „dokonalosti“. Bohužiaľ, často to tak vnímané nie je, a preto s tým treba niečo robiť. Faktor vzniku konfliktu je ten, že tester musí hľadať chyby v kóde a oznámiť to programátorovi. Otázkou je, akou formou mu to oznámi respektíve ako poukáže na nájdené chyby. Často sa reportovanie chýb podobá diplomatickému jednaniu.

Spôsobom minimalizovania vzniku sporu v tomto prípade je trénovanie testerov v diplomatickom prístupe (spôsobe), ako oznámiť programátorom kde a aká chyba nastala. Pokiaľ bude oznámenie podané vhodným spôsobom, slušnou formou a do istej miery diplomaticky, pravdepodobnosť vzniku konfliktu by sa mala značne znížiť.

Konflikt na úrovni organizácie – faktor: *pozícia v tíme*

Veľmi často sa v tíme stáva, že niektorý členovia považujú svoju úlohu pri projekte za tú najdôležitejšiu. Je to často dané hlavne povahou danej osoby a často aj vplyvom okolia. Ak je neustále kladený dôraz na niektorú úlohu v tíme a tá je neustále vyzdvihovaná do popredia pred ostatné úlohy, môže osoba na danej pozícií získavať pocit nadradenosti a nenahraditeľnosti. Naopak, ak je niektorá pozícia prehliadaná, môže osoba ktorá ju vykonáva, nadobudnúť dojem menejcennosti. Takéto pôsobenie prostredia nemá vplyv len na osoby vykonávajúce dané úlohy, ale aj na vnímanie osôb v okolí. Ak napríklad členovia tímu vidia, že tím líder prehliada alebo podceňuje niektorú úlohu (rolu), môžu ju aj oni začať podceňovať a prehliadať.

Veľmi často sa v softvérových skupinách stáva, že úloha testerov je podceňovaná a testeri potom musia neustále bojovať o uznanie a rešpekt ostatných členov. Z praxe je tiež známe, že úloha programátorov je často neúmerne zvýrazňovaná, čo u nich môže viesť k pocitu nadradenosti a môžu začať nazerať na testerov zhora. Títo si to často uvedomujú, čo vedie k zvyšovaniu napätia medzi nimi. Toto napätie často prerastá až do konfliktu.

Riešením tejto situácie môže byť rovnocenný prístup zo strany tím lídra (manažéra) ku všetkým úlohám v tíme. Dôležité je, aby neprehliadal žiadnu úlohu a naopak aby vyzdvihoval členov nie na základe ich úlohy v skupine, ale na základe dobre odvedenej práce a prínosu k projektu.

Záver

Softvérové tímy (ale aj tímy vo všeobecnosti) sa skladajú z nesúrodej skupiny osobností, ktoré zastávajú určité úlohy. Pretože jednotliví členovia majú rozdielne povahy a úlohy, ktoré v skupine predstavujú, sa veľmi často ovplyvňujú, vznikajú rozpory v záujmoch a cieľoch, čo vedie k vzniku konfliktov. Úlohou tím lídra je vopred identifikovať a následne minimalizovať faktory vplývajúce na vznik konfliktov, a tým zabrániť aby vôbec vznikli. V prípade, že už konflikt vznikne, aj napriek snahám mu predísť, musí byť tím líder schopný identifikovať a použiť vhodné metódy na jeho vyriešenie. Použitie vhodnej metódy mu umožní minimalizovať jeho negatívny dopad a niekedy mu dokonca použitie správnej metódy v pravý čas poskytne nástroj, ako obrátiť tento konflikt v prospech projektu a tímu.

Použitá literatúra

1. Barki, H., Harvick, J.: Interpersonal conflict and its management in information system development. *MIS Quarterly 25*, 2 (June 2001), 195-228
2. Cynthia F. Cohen, Stanley J. Birkin, Monica J. Garfield, Harold W. Webb: Managing Conflict in Software Testing. In Communications of the ACM, Vol. 47, No. 1 (January 2004), 76-81.

Annotation

Conflict management in team

In this „Team era“, problematics of the teams and the mutual relations of their members are currently being the object of a lot of research. The interactions in a group and its dynamics belong to the area of group psychology but a good manager or a team leader should be aware of this topic. One of the most important part of these relations that we should focus on, is conflict. It is very important to understand what is a conflict, how does it occur, what phases does it have and how solve these situations or how to avoid them. These questions are handled in the first part of this essay. The second part is focused on the means of minimizing of the factors, which cause the conflicts, namely two roles in the team – the programmers and the testers. There are also described some situations, which make pressure between the two mentioned groups and some possible tactics of minimalizing them.

Vývoj tímu v softvérovom projekte a vplyv na manažment

ATTILA KOTRBA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
attila.kotrba@centrum.sk

Abstrakt. Softvérový tím je skupina ľudí, ktorý majú spoločný cieľ, a to vyvinúť produkt, podľa požiadaviek zákazníka. Tím ľudí, či už malý, alebo veľký, v priebehu projektu nie je stály, ale mení sa. Na zmenu tímu vplýva mnoho faktorov, ktoré spôsobujú jeho zmenu. Ľudia sa počas práce na vývoji menia, tak isto sa menia aj kompetencie a role jednotlivých vývojárov. Manažment tohto procesu je dôležitý z hľadiska dokončenia projektu a pre manažéra je manažment ľudských zdrojov jednou z najdôležitejších činností. Esej uvažuje o vývoji tímu a jeho vplyve na manažment.

Úvod

Vývoj softvéru je zložitý technický proces. Ale nielen technický, počas vývoja zo sebou musia interagovať a komunikovať desiatky ľudí. Ľudia v tímoch musia používať tak isto rôzne technické pojmy, metódy, ako aj technológie súvisiace s informatikou. Na softvérový tím sa môžeme pozerať z hľadiska technického, ako zoskupenie špecialistov v obore, ale aj z hľadiska spoločenského, z ktorého uvidíme skupinu ľudí, ktorí majú svoje potreby a požiadavky, a počas vývoja svoje názory menia tak isto samozrejme, ako sa menia ich požiadavky na kolegov v tíme. Teda počas cyklu vývoja produktu sa tím vyvíja. Potreby tímu z hľadiska technického sú iné na začiatku fázy vývoja, ako počas vývoja, alebo na konci vývojového cyklu. Tím je zložený zo špecialistov, ktorý sa zaoberajú rôznymi oblasťami vývoja. Vývoj softvéru ale nevyžaduje počas svojho cyklu stále konštantný počet ľudí na určitú činnosť. Zvládnutie manažmentu tímu je jedným z predpokladov úspešného vývoja softvéru.

Tím a jeho vznik

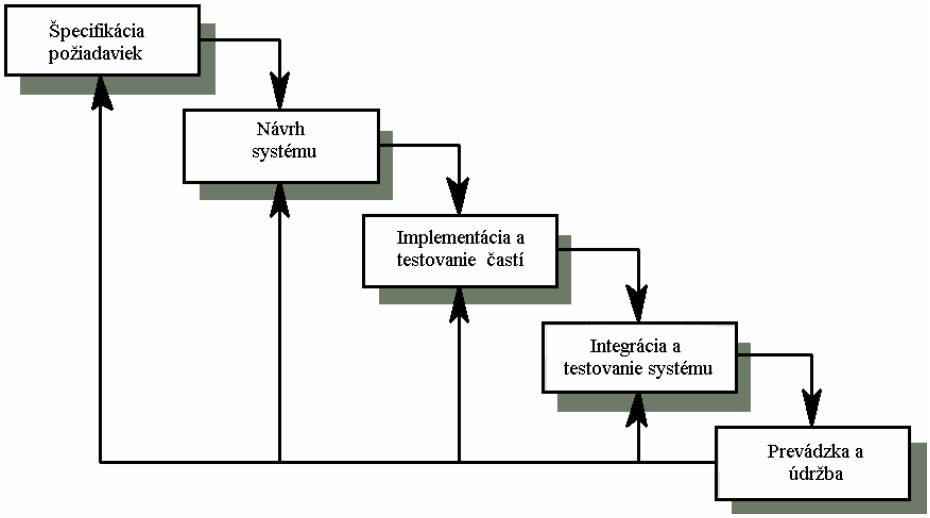
Na začiatku vývoja softvéru sa musí naplánovať postup jeho vývoja. Pri plánovaní tohto postupu je nutné zahrnúť do plánu aj požiadavky na ľudské zdroje. Tím môže byť vytvorený viacerými spôsobmi, a to výberom z interných pracovníkov firmy, alebo

Vývoj tímu v softvérovom projekte a vplyv na manažment,
január 2006, s. 101-107.

prijatím nových zamestnancov. Ktorý spôsob je všeobecne vhodnejší, sa nedá povedať. Myslím ale, že vhodnou a citlivou kombináciou týchto dvoch postupov sa dá vytvoriť vhodný tím na zvládnutie úlohy.

Tím vytvorený výlučne z interných zdrojov má veľkú výhodu v ľuďoch, ktorý majú prax v danej firme, navzájom sa poznajú (určite nie všetci, podľa veľkosti spoločnosti), a majú zvládnuté postupy a procesy vo vnútri firmy. Pravdepodobnosť, že zamestnanec sa rozhodne odísť v priebehu projektu, je nízka, a omnoho nižšia, ako pri pracovníkoch z získaných z externých zdrojov. Nákladné môže ale byť ich preškoľovanie na inú platformu, ak s ňou nepracovali. Takýchto členov tímu ale nie je jednoduché zaobstarať, keďže v spoločnosti nie je naraz toľko voľných ľudí, ak budú zaťažený s viacerými projektmi naraz, existuje tu možnosť ohrozenia pôvodného projektu. Ďalšia nevýhoda spočíva v relatívnej stability zamestnanca, ktorý nemá nové nápady, a všetko sa bude snažiť riešiť zaužívanými postupmi, aj keď by boli inovárorské postupy vhodnejšie. Pri výbere zamestnancov z externých zdrojov je v prvom rade náročné získať viacero, resp. až veľké množstvo softvérových pracovníkov naraz. Nábeh na vnútorné procesy a postupy vo firme je pomalší, tak isto aj rýchlosť práce nových ľudí a ich efektívnosť je vo všeobecnosti nižšia. Na druhej strane nový ľudia môžu byť vhodným a žiadaným „osviežením“ vo firme a môžu byť expertmi v technológiách, ktorých sú interný zamestnanci neznalí. Môžu ponúknuť inovátorské postupy, ktoré budú viesť k rýchlemu a úspešnému ukončeniu projektu. Nebezpečenstvo ale spočíva v možnom rýchlom odchode nových zamestnancov, ktorým spoločnosť nebude vyhovovať, a tým tím stráca ľudí. Vhodnou kombináciu ale podľa mňa možno dosiahnuť, že tím bude obsahovať ľudí, ktorý dobre poznajú postupy v spoločnosti a dokážu viesť a motivovať nových zamestnancov k vysokému výkonu.

Proces vytvorenia softvérového systému zahŕňa viacero fáz. Uvažujme o klasickom vodopádovom modeli vývoja softvéru (pozri **Obr. 1.**):



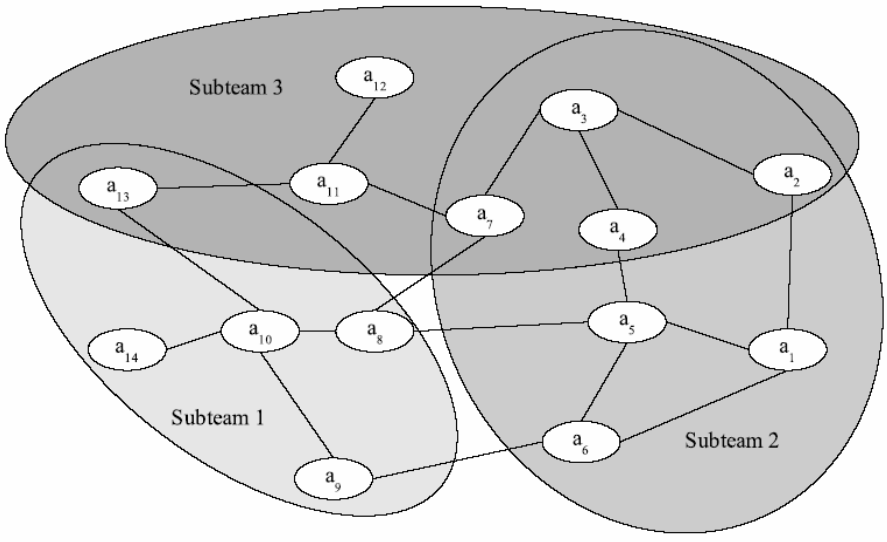
Obr. 1. Vodopádový model tvorby softvéru

Vo všetkých týchto fázach je potreba vo vývojovom tíme určitého počtu rôznych vývojárov. Na začiatku vo fáze špecifikácie a návrhu systému sa v tíme musí nachádzať určite dostatočné množstvo analytikov a návrhárov, ktorý vytvárajú a udržujú návrh systému. Pri implementácii musia byť nasadení programátori, ktorí implementujú systém. Zároveň je nutné systém testovať, čo na úrovni modulov a celého systému vykonávajú testeri. Po nasadení aplikácie do prevádzky sa vykonáva aj technická podpora aplikácie, čo vykonávajú znova iný ľudia. Hranice týchto činností nie sú pevné a jednotlivé etapy sa prelínajú, takže napr. časť programátorov je vyťažená aj v etape návrhu, alebo testovania, a naopak.

Rozdelenie tímu

Keďže v tíme sa nachádza ale určité množstvo analytikov, programátorov, a rôznych iných pracovníkov, ktorý nie sú naplno vyťažení počas celého životného cyklu softvéru, naskytá sa otázka ako ich využiť. Odpoveďou je možnosť práce aj na iných projektoch. Myslím si, že nemá zmysel držať niektorého člena tímu len na jednom projekte, kde nie je využitý. Počas práce na softvéri je nutné tím manažovať dostatočne dobre, vyžaduje si teda skúseného manažéra, ktorý vie určiť vyťaženie svojich ľudí a určiť im prácu.

Rozdelenie do podtímov je jednou z možností, ako zlepšiť komunikáciu medzi tímami. Určite nie je vhodné vo väčších tímoch nad 10 ľudí umožniť komunikáciu „každý z každým“, pretože počet komunikačných kanálov je veľmi veľký. Informácie by sa nemuseli dostať ku všetkým členom tímu, ktorý to potrebujú, a komunikácia by bola neefektívna. Rozdelenie do podtímov umožňuje efektívnu komunikáciu medzi tímami, a medzi členmi jednotlivých tímov. Tým vytvára aj určitú úsporu času na komunikáciu. Rozdelenie tímu na podtímy je na **Obr. 2**.



Obr. 2. Rozdelenie tímu do podtímov

Toto rozdelenie umožňuje aj komunikáciu viacerých členov z jedného podtímu s viacerými členmi z iného podtímu. Jedna osoba môže byť členom aj viacerých podtímov, čo umožňuje voliť určitú dynamiku v softvérovom tíme. Je tu tiež možnosť počas vývoja softvéru tímy meniť. Oproti tomuto rozdeleniu existuje aj rozdelenie, kde komunikácia medzi tímami prebieha cez konkrétny kanál. Toto rozdelenie ale neumožňuje dynamiku predchádzajúceho modelu. Preto sa nazdávam, že možnosť koexistencie člena tímu vo viacerých podtímoch je určite vhodná, a dokonca žiadaná.

Vývoj tímu

V roku 1965 Bruce Tuckman zaviedol štvorfázový model vývoja tímu, ktorý sa stal všeobecne akceptovaným modelom, ako sa tímy vyvíjajú. Úspešný a skúsený team-leader a manažér by mal vedieť, v ktorej fáze sa jeho tím nachádza. Myslím, že túto znalosť môže efektívne využiť pri riadení tímu.

Po vytvorení tímu nastáva fáza formovania tímu (forming). Predstavte si spoločnosť, kde vás prijmú, zaškolia a po zaškolení sa stretnete prvý krát s ľuďmi, ktorých ste nikdy nevideli, a manažér vám oznámi, že idete spolu robiť projekt. Myslím si že v takejto situácii sa noví pracovníci môžu ocitnúť dosť často, čo som aj ja zažil. Keďže ľudia sa medzi sebou nepoznajú, navonok sa síce tvária slušne, ale zachovávajú si odstup od ostatných ľudí. Vedúci tímu musí ľudí zoznámiť, jeho úlohou je vytvoriť priateľskú atmosféru, v ktorej dokážu všetci ľudia efektívne pracovať. Vedúci tímu má v tejto fáze rozhodujúci vplyv na jednotlivých ľudí, ľudí musí uviesť do spôsobu práce tímu a procesov vo vnútri spoločnosti. V tejto fáze sa ľudia zoznamujú, a na konci fázy by sa mala skupina ľudí zmeniť na tím. Myslím že v tejto fáze by si mali ľudia začať pomaly dôverovať a určite sa vyselektujú ľudia, ktorý nevedia pracovať tímovo, takéto charakteristiky sa poznajú už na začiatku procesu formovania. Nemožno povedať presne, koľko by mala táto fáza trvať, ale myslím, že pri ľuďoch, ktorí sa poznali, ale so sebou nespolupracovali by mala byť omnoho kratšia ako pri ľuďoch, ktorý sa navzájom nepoznali.

Tím ďalej vstupuje do fázy kryštalizácie tímu (storming). Táto nastáva, ak si jednotlivý členovia tímu začínajú viac dôverovať. Toto však neznamená, že ich názory sú rovnaké, práve naopak, v tejto fáze už každý člen tímu osvojuje vlastné názory na projekt, na ktorom sa pracuje a aktívne sa svoje názory snaží „vnútiť“ tímu. Skúsený ľudia vedia, že v tejto fáze sa musí tím zorientovať a vyriešiť názorové problémy, určiť, ktoré spôsoby sú lepšie, a využiť ich v projekte. Týmto integrácia tímu stúpa, a takisto dôvera členov tímu medzi sebou. V tejto fáze môžu nastať aj nebezpečné situácie, keď sa niektorý člen s dohodnutým riešením nezmieri, a bude sa stále snažiť aplikovať svoje názory. Týmto pravdepodobne budú vznikať rôzne poruchy v tíme a tieto by sa mali identifikovať a vyriešiť čo najskôr. V tejto fáze by bolo možno vhodnejšie ľudí, ktorý sa nevedia integrovať do tímu, z tímu preradiť a vymeniť ich. Toto rozhodnutie asi dosť výrazne ovplyvní celý zvyšok tímu, je určite veľmi vážne, ale myslím že môže zabrániť neskorším problémom, a možno až neúspechu celého projektu.

Postupne sa celý tím zjednocuje v použitých metódach a názoroch, vstupuje do fázy normovania (norming). Členovia tímu používajú spoločné pracovné metódy, celý tím už toto akceptuje. Svoje názory členovia tímu menia a zjednocujú na čo najlepšie pre potreby tímu. Členovia tímu ale podľa mňa vo všeobecnosti niektoré názory a metódy nemusia považovať za najlepšie možné, ale akceptujú rozhodnutie tímu, takže pracujú tímovo a majú predpoklad úspešnej tímovej práce. Snažia sa o použitie dohodnutých metód v čo najväčšej miere na danom projekte, aj keď sami by určite rozhodli inak. Komunikácia v tíme je vyrovnaná a nezhody v zásadných otázkach už nevznikajú.

Tím vstupuje do jeho finálnej fázy, realizácie (performing). V tejto fáze tím spolupracuje na riešení úloh, ktoré vyplývajú z projektu. Spolupráca tímu sa čím , tým zlepšuje, členovia tímu si veria navzájom, vzťahy medzi jednotlivými členmi tímu sú vyprofilované, čiastočne sa ale menia aj v tejto fáze. Tím je schopný pracovať na najťažších problémoch, jeho výkonnosť je v tejto fáze najväčšia. Komunikácia medzi členmi tímu je otvorená a všetky väčšie problémy sa riešia na úrovni celého tímu, menšie rozhodnutia rieši aj jeden člen tímu.

Táto fáza môže viesť do rôznych ďalších fáz. Ak sa projekt skončí úspešne, tým finalizuje svoju činnosť a jeho členovia sú nasadený na iné tímy. Ak sa tím osvedčil, môže sa použiť aj na iné projekty v danom zložení. Do týmu môžu tiež vstúpiť nový ľudia, resp. aj odísť existujúci, čím sa tím znova presúva do fáze formovania. Za daných okolností, ak sa úspešne projekt ukončí, si myslím že najvhodnejším riešením pre spoločnosť je pokračovať na novom projekte s daným tímom. Počas projektu tím nadobudol skúsenosti, ktoré sa pri novozaloženom tíme budú musieť znova vyvinúť. Ak má firma ďalšie projekty, ktoré sa charakteristikou blížia existujúcemu, odporúčam použiť daný tím na riešenie nového projektu. Možno by sa daný tím mohol obohatiť o nových ľudí, ak je projekt väčší, resp. ak sa spoločnosť rozhodla prijať nových pracovníkov. Títo získajú nové skúsenosti a neutrpí výkon tímu. Myslím si že zohraný a vyformovaný tím je veľmi cenný pre firmu, a môže jej priniesť omnoho viac úžitku ako nový tím, u ktorého existujú mnohé riziká.

Riziká a problémy

Prácu tímu môžu prerušiť viaceré udalosti. Ak tím opustí jeden alebo viacerý ľudia, je to citeľná strata, aj pri väčších tímoch. Využitie zvyšného potenciálu a riešenie úbytku pracovných síl si vyžaduje schopnosti dobrého manažmentu. Takéto neplánované zásahy do tímu bolia najviac, ak tím stratí vedúcich pracovníkov, alebo tých najlepších ľudí, resp. je vo fáze realizácie (performing). Ich nahradenie nie je len jednoduchým získaním iných ľudí a ich zapojením do tímu. Aj keď manažér nájde vhodného kandidáta, so správnymi znalosťami a skúsenosťami, jeho úplné zapojenie do už existujúceho tímu si vyžaduje dlhší čas, počas ktorého sa musí nový pracovník adaptovať, a nabehnúť na prácu tímu. Pri strate zamestnanca utrpí súdržnosť tímu, ktorá sa dlho sceľuje, pri novom zamestnancovi. Nemožno zanedbať ani osobné konflikty ľudí v tíme, môže sa stať, že nová pracovná sila z tímu vypadne už krátko po nástupe, čo môže byť spôsobené aj osobnými konfliktami, alebo neschopnosťou, či

neakceptovaním spôsobu práce existujúceho tímu novým pracovníkom. Ak nový zamestnanec nabehne na prácu tímu a vykazuje dobré výsledky, je nevyhnutnou úlohou manažéra, aby udržiaval tím v konzistentnom stave, pretože v závislosti od veľkosti tímu, a samozrejme aj projektu, ďalší výpadok práce tímu môže viesť k neúspešnému projektu. V niektorých prípadoch by bolo možno aj vhodnejšie pracovať s existujúcim počtom ľudí, isto všetci poznáme Brookov zákon: „Adding manpower to a late software project makes it later". Či už na omeškaný projekt, ale aj na neomeškaný, myslím že treba veľmi citlivo zvážiť, či sa oplatí do projektu pridať ľudí, alebo dokončiť projekt s nižším počtom ľudí.

Zo stratou zamestnanca je nutné rátať aj stratu vedomostí, ktoré mal zamestnanec. Top zamestnanci majú spoločnosti najväčší osoh, a preto ich stratou odíde aj množstvo vedomostí, ktoré sa musia nahradiť dostatočne kvalitným novým zamestnancom. Dokumentácia, aj keď jej úroveň môže byť vynikajúca, nikdy nenahradí vedomosti a skúsenosti pracovníka, ktorý firmu opustil. Nabehnutie nového zamestnanca na prácu tímu môže byť problematické v prípade, ak zamestnanec musí väčšinu znalostí študovať z dokumentácie, lebo ostatný zamestnanci o daných veciach nevedia nič, alebo sú ich znalosti výrazne obmedzené.

Úlohou manažmentu je ale udržiavať tím konzistentný už od začiatku vývoja, a nie len v prípade, ak v tíme začínajú vznikať problémy, rozpory. Dôležitou úlohou manažérov aj z tohto pohľadu je vhodne motivovať ľudí k práci a k dobrým výkonom. Lebo konkurencia nespí, a zamestnanec sa rozhoduje aj podľa odmien za prácu, ktoré dostane, a v prípade výrazne vyšších u konkurencie je len otázkou času, kedy zamestnanec odíde.

Záver

Zloženie tímu na začiatku projektu je dôležitá činnosť, ktorá sa vyžaduje od manažérov, ktorý poznajú ľudí, s ktorými spolupracujú, a vedia čo od nich môžu očakávať. V procese vývoja tímu však vznikajú rôzne vzťahy medzi jednotlivými členmi tímu, ktoré sa na začiatku projektu ťažko odhadujú. Manažment musí zvládnuť túto úlohu a počítať s rôznymi možnosťami pri vývoji tímu. Neúspešný projekt môže byť pre mnohé spoločnosti ten posledný, ktorý robili, a neúspech projektu môžu navodiť aj negatívne zmeny vo vývoji tímu. Domnievam sa, že oblasť manažmentu tímu a ľudských zdrojov je veľmi dôležitá pre každú spoločnosť na trhu. Stabilné tímy si spoločnosti musia udržiavať, znalosť a know-how kvalitných zamestnancov je vlastnosť, ktorú si spoločnosti nemôžu dovoliť stratiť.

Použitá literatúra

1. Brooks, F.: *The mythical Man-Month.* Addison-Wesley, 1975.
2. Sawyer, S.: Software development teams. *Communications of the ACM*, Vol. 47, No. 12 (2004)

3. Reed, A.: Why developers leave (And what you can do about it) DeveloperDotStar.com, 2005.
4. Scerri, P., Xu, Y., Liao, E., Lai, J., Sycara, K.: *Scaling teamwork to very large teams.* AAMAS'04, New York, 2004.
5. Cushing, J., Cunningham, K., Freeman, G.: *Towards best practices in software teamwork.* Consortium for computing sciences in colleges. 2003.
6. Barnum, C.M.: Building a team for user-centered design. IEEE. 2000

Annotation

Team process in software project and effect to management

Software team is a group of people, which has common goal, and it is to develop a product, by the customer's requirements. Team of people, it can be small or large, is not only group of permanent people in the process of project developing, it is a dynamic formation, it makes changes. On team changing, many factors are applying, which causes team changes. People in the team are changing too. Competencies and roles of people in the team has changing in development process. Management of this process is very important in viewpoint of finishing the project successfully and for manager is management of human resources one of the most important activities. Essay is thinking about team process and his effect to management.

Manažment softvérového projektu a organizácia softvérových tímov

LUKÁŠ KROČKA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
krocka01@student.fiit.stuba.sk

Abstrakt. Výber vhodných ľudí do projektu výrazne ovplyvňuje jeho úspešnosť. Úlohou manažéra projektu je zanalyzovať všetky potrebné informácie a vytvoriť vhodnú štruktúru tímu, aby bol schopný riadiť projekt do úspešného konca. V eseji popisujem niektoré faktory, ktoré ovplyvňujú rozhodovanie projektového manažéra pri organizovaní tímov.

Úvod

V súčastnosti je na trhu mnoho softvérových firiem, ktoré sa zaoberajú vývojom softvéru. Na to, aby sa firme podarilo preraziť v tomto konkurenčnom prostredí, je potrebné riadiť celý proces tvorby softvéru. Riadenie je veľmi dôležité z hľadiska dodržania termínov, rozpočtu a na zabezpečenie požadovanej kvality softvéru. Bez ohľadu na veľkosť firmy, táto sa v danom čase väčšinou nevenuje iba jednému projektu. Vo firme sa pracuje na viacerých projektoch súčasne, ktorých výstupy (celý softvér alebo jeho časť) bývajú určené rôznym zákazníkom. Tieto výstupy projektov sú výsledkom tímovej práce ľudí, ktorí sa na nich podieľajú. Úlohou manažmentu projektu je okrem iného aj prideľovať ľudí na jednotlivé projekty a určiť ich zodpovednosti.

Cieľom tejto eseje je uviesť čitateľa do problematiky manažmentu softvérového projektu , zhrnúť úlohy a zodpovednosti projektového manažéra a pokúsiť sa uviesť základné faktory ovplyvňujúce manažéra projektu pri organizovaní tímov.

Projekt

Skôr, ako začnem písať o manažmente projektu a organizácii softvérových tímov, treba vysvetliť, čo to projekt je. Ako je uvedené v [4]: „Projekt je jedinečný proces zložený z rady koordinovaných a riadených činností s dátumom zahájenia a ukončenia, vykonávaný pre dosiahnutie cieľa, ktorý vyhovuje špecifickým požiadavkám, vrátane obmedzení daných časom, nákladmi a zdrojmi“. Voľne povedané, projekt je

transformácia vstupov na požadované výstupy (jedinečné ciele), pričom proces tejto transformácie je dočasný a riadený.

Softvérové projekty sú známe tým, že sú ukončené neskoro, s väčším rozpočtom, ako bolo plánované a často aj s tým, že nespĺňajú požiadavky používateľa [3]. Takéto projekty asi nemôžeme vyhlásiť za úspešné. Výskumní pracovníci odhadujú, že ročne sa investuje 80 až 500 miliárd dolárov do IT projektov, ktoré nikdy nie sú implementované [3]. Čím je to spôsobené? Najčastejšia odpoveď je slabý manažment projektu. Efektívny manažment projektu je aspoň tak dôležitý, ako úspešný softvérový vývoj.

Tiež si myslím, že na úspešnosť projektu má významný vplyv aj jeho povaha. Ak sa firma zaoberá tvorbou generického softvéru, asi sú jej projekty častejšie odsúdené na neúspech, ako firmy, ktorá sa zaoberá tvorbou softvéru na zákazku. Generický softvér je taký, ktorý sa predáva ľubovolnému záujemcovi a na trh sa dostáva až po jeho dokončení (teda pri jeho tvorbe nie sú uzatvorené zmluvy so zákazníkmi).

Manažment softvérového projektu

Metódy a nástoje manažmentu projektu sú čoraz dôležitejšie vďaka tomu, že súčasné organizácie dosahujú svoje ciele vďaka tímom, ktoré sú často zložené aj z ľudí rôznej kultúry a z ľudí, ktorí majú rôznu špecializáciu [1]. Úlohou manažmentu projektu však nie je výber nových zamestnancov do firmy, či organizácie, ale manipulácia s ľudskými zdrojmi, ktoré má k dispozícii. Rastúca rôznorodosť skúseností, vedomostí, kultúry a perspektívnosti projektových tímov môže mať pozitívny i negatívny vplyv na skupinové procesy a výsledky. Kultúrna rozmanitosť tímov môže spôsobiť problémy pri fungovaní tímových procesov, ale tiež môže priniesť aj isté výhody. Takýmito výhodami môže byť napríklad lepšia kreativita tímu, lepšia schopnosť inovácie a riešenia problémov v tíme.

Vytváranie tímov na základe kultúrnej rozmanitosti prináša nové výzvy oproti klasickému prístupu k manažmentu projektu [1]. Napriek tomu si myslím, že výber nových ľudí do firiem sa v praxi nerobí na základe ich „pôvodu“, ale na základe ich osobnosti a skúseností, ktoré sú požadované. Manažment ľudských zdrojov teda značnou mierou ovplyvňuje prácu manažmentu projektu a výber vhodných ľudí nie je ľahká záležitosť.

Vráťme sa však k manažmentu softvérového projektu. Metódy manažmentu projektu sú určené na maximalizáciu výhod tímovej práce pri riešení projektu (a tým prispieť k vyšším ziskom). Cieľom manažmentu softvérového projektu pri prideľovaní ľudí na jednotlivé projekty nie je len výber najvhodnejších kandidátov, ale aj prevencia pred nedorozumeniami, nesúhlasom a konfliktami medzi členmi tímu. Časté konflikty v tíme spôsobujú stratu v podniku.

Zodpovednosť manažéra projektu

Manažér projektu je zodpovedný za všetko, čo je nevyhnutné pre úspešné ukončenie projektu, či už priamo alebo nepriamo [5]. Je v centre všetkého, čo súvisí s projektom.

Projektový manažér má napríklad nasledovné zodpovednosti:

- tvorí hlavný spojovací bod s ďalšími oddeleniami organizácie
- má priamu zodpovednosť za aktivity jednotlivých členov tímu, ktorí sa zúčastňujú na projekte, za jednotlivé úlohy v projekte a za všetky výstupy.

Je zodpovedný za to, že ciele sú realistické. Pre každý projekt by mali byť vhodne definované: ciele, rozpočet, meranie výkonnosti, zodpovednosti a časový rozsah. Aby mal manažér nejaký stupeň dôvery k výstupu projektu, potrebuje nasadiť do projektu správnych ľudí so správnou kombináciou skúseností. Tí by mali pracovať s procesmi a nástrojmi, ktoré majú odskúšané a s ktorými majú dobré skúsenosti. Pre efektívne riadenie projektu je potrebné pochopiť: účel, ciele, oblasť, financovanie a mandát.

Na pozíciu manažéra softvérového projektu sa človek pravdepodobne nedostane hneď po prijatí do nového zamestnania. Podľa môjho názoru sa musí softvérový inžinier na túto pozíciu vypracovať, pretože každý softvérový projekt je jedinečný a každá softvérová firma používa vlastné postupy a metodiky na dosahovanie svojich cieľov. V tom sa softvérové firmy líšia od ostatných firiem. Manažéri z iných odvetví ako IT majú často problémy uchytiť sa na pozícii IT manažéra. Naopak IT manažérovi takýto prechod nerobí až také problémy.

Štruktúra a organizácia tímu

Spôsob, akým je tím štruktúrovaný, môže hrať podstatnú úlohu v jeho fungovaní [5]. Rozdielne druhy tímov majú rozdielne charakteristiky. Opatrnosť pri zostavovaní tímu a určenovaní vzťahov môže viesť k rozdielnym výsledkom. Rozličné roly v tíme závisia na povahe projektu. Manžér projektu teda musí zostaviť z dostupných zdrojov taký tím, ktorý je najvhodnejší na realizáciu daného projektu. Keďže vo firmách sa často pracuje na viacerých projektoch súčasne, výber ľudí na konkrétny z nich je ovplyvnený zaťažením týchto ľudí v iných projektoch.

Je bežné, že zdroje a štruktúra projektu sa postupom času menia. Často na začiatku jeden lepší tím definuje biznis riešenie, potom nejaký všestrannejší tím ho dodá a nakoniec prevádzkový tím ho zavádza do prevádzky. Štruktúra tímu sa pravdepodobne bude prispôsobovať v každej fáze, aby vyhovela vyvíjajúcej sa povahe projektu. Správna štruktúra pre menší tím pravdepodobne nebude pracovať pre veľký vývojový tím.

V malej začínajúcej firme, v ktorej pracuje napríklad 5 ľudí, sa štruktúra tímu pravdepodobne nebude meniť. Samozrejme, za istých okolností (napríklad zdravotné problémy niektorého pracovníka) sa štruktúra zmení, avšak v takomto prípade bude potrebné urobiť isté opatrenia, aby sa vôbec projekt vyriešil načas. Výpadok zamestnanca v malej firme asi výraznejšie ovplyvní jej chod, ako výpadok v nejakej veľkej organizácii.

Štýl tímu

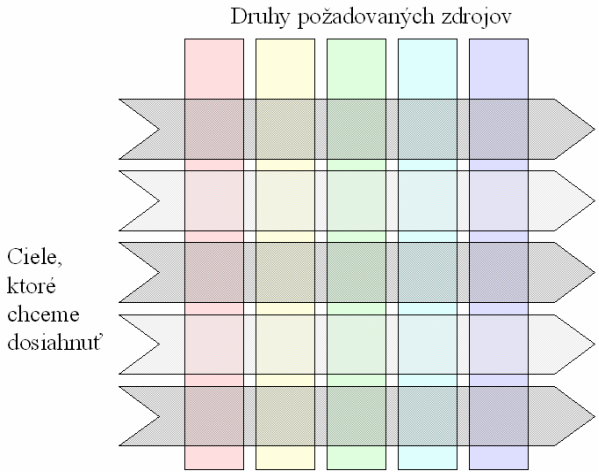
Existujú dva hlavné štrukturálne rozmery projektového tímu [5]. Tieto rozmery umožňujú pozerať sa na tím z dvoch pohľadov:

- na základe toho, aké sú typy zdrojov v tíme

– na základe toho, čo dodávajú

Napríklad návrhár webovej stránky môže pracovať s biznis manažérmi a sieťový špecialista vytvárať webové rozhranie, zatiaľ čo ďalší webový návrhár pracuje s inými biznis manažérmi ale možno ten istý sieťový špecialista na Intranet aplikácii pre prezentáciu vnútorných informácií manažmentu – obe ako súčasť toho istého projektu. Otázkou teda je, či je lepšie mať tím vývojárov, tím manažérov a tím sieťových špecialistov alebo je výhodnejšie mať tím pre webové rozhranie a tím pre samotný systém, ktorý poskytuje informácie pre manažment.

Pri hladaní odpovede na vyššie uvedenú otázku môžeme vnímať projektový tím ako maticu. Táto je znázornená na nasledujúcom obrázku.



Obr. 1. Vnímanie tímu ako matice [5].

Členovia rôznych tímov (zdroje) potrebujú v týchto tímoch spolupracovať, aby mohli zdielať vedomosti a zabezpečiť konzistentné riešenie. Ľudia, ktorí spolu pracujú na rôznych procesoch alebo funkčných aspektoch riešenia, budú rovnako potrebovať spoluprácu. Každý takýto podtím, či už horizontálny alebo vertikálny, potrebuje svojho vedúceho. Členovia tímu potrebujú poznať svoje individuálne úlohy, aby mohli efektívne pracovať.

Ďalšou často kladenou otázkou je, ako tím štruktúrovať vzhľadom na riadenie. V literatúre sa uvádza niekoľko základných pravidiel a postrehov, ktoré môžu pomôcť pri rozhodovaní o štruktúre tímu [5]:

– Ľudia, ktorí spolu pracujú v tíme, zvyčajne vnímajú svojich spolupracovníkov ako „ľudí, ktorí sú na ich strane“. Títo spolu bežne spolupracujú a pomáhajú jeden druhému za účelom dosiahnutia ich spoločného cieľa.

– Umiestnenie ľudí do rovnakého projektu vytvára spoluprácu, zdieľanie vedomostí a prenos skúseností

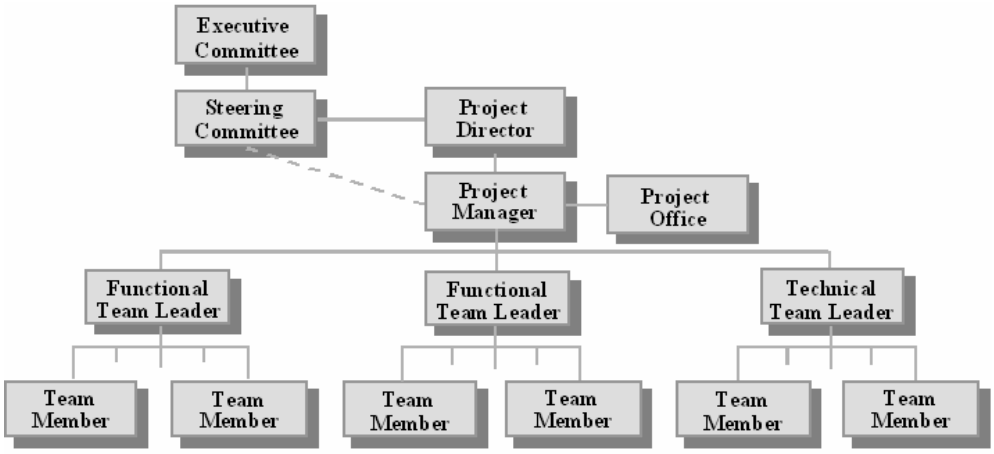
- Vytvorenie dobrého, efektívneho tímu je rozhodujúce o úspešnosti projektu. Štruktúra tímu má dopad na správanie sa tímu ako celku. Cieľom je vytvoriť spolupracujúci tím, kde jednotlivci zdieľajú vedomosti, spolupracujú, podporujú jeden druhého a navzájom sa motivujú za účelom dosiahnutia cieľov.
- Vyššie uvedené vlastnosti majú vplyv na technickú uskutočniteľnosť projektu, na efektívnosť a výkonnosť tímu.
- Chápanie, vedomosti a schopnosti ľudí pracujúcich v ďalších tímoch sú zriedkavo stopercentne využité.
- Skutočnosť, že ľudia pracujú v ďalších tímoch, je často chápaná ako nepríjemnosť, pretože tým negatívne vplývajú na vývoj tímu.
- Umiestnenie veľkého počtu ľudí do jedného tímu spôsobuje skôr vzájomné pôsobenie a spomaluje vývoj tímu.

Pri manažmente projektu je dôležité okrem identifikácie typu osobnosti myslieť aj na úroveň schopností a sily. Ak potrebujeme niekoho, kto má vykonávať biznis rozhodnutia, musíme vybrať tú správnu osobu. Ak potrebujeme niekoho na bežnú prácu, nemali by sme plytvať časom pre nás cennejšieho pracovníka.

Na základe vyššie uvedených skutočností by som asi ťažko vytváral štruktúru tímov, ak by som nemal isté predstavy o schopnostiach a znalostiach svojich podriadených pracovníkov. Jedinečnosť každého projektu núti projektového manažéra učiť sa na vlastných chybách a odhadnúť najvhodnejšiu štruktúru tímu pre každý projekt. V nasledujúcich podkapitolách uvádzam možné spôsoby organizovania tímov.

Funkcionálna organizácia tímov

Na nasledujúcom obrázku je znázornený príklad funkcionálnej štruktúry tímu.



Obr. 2. Príklad funkcionálnej organizácie tímov [5].

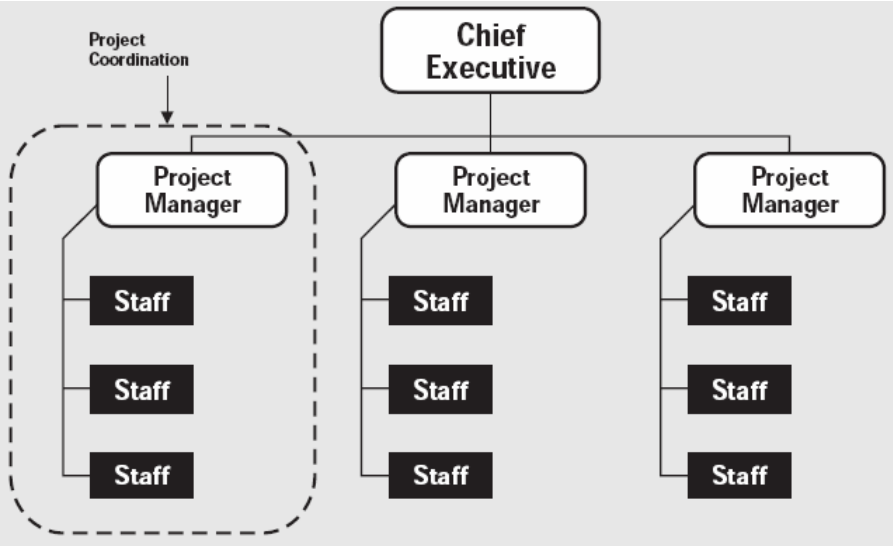
Klasická funkcionálna štruktúra poskytuje hierarchiu, kde každý zamestnanec má jasného nadriadeného [2]. Personál je zoskupený podľa osobitostí, ako je napríklad výroba, marketing, inžinierska činnosť a účtovníctvo na vrchnej úrovni. V tejto štruktúre sú teda pracovníci združení na základe funkcionálnej oblasti. Jednotlivé tímy sú zložené z ľudí tak, že do jedného tímu sú zahrnutí ľudia, ktorí majú jednu funkciu v organizácii (dostaneme tak nejaké oddelenia v podniku).

Pre softvérovú firmu môže byť vytvorený zvlášť tím pre analýzu a návrh architektúry, tím pre návrh a implementáciu, tím pre testovanie, tím pre technickú podporu a pod. Jednotlivé oddelenia v podniku pracujú nezávisle od iných oddelení, avšak ich vzájomná spolupráca a koordinácia je samozrejme nevyhnutná. Táto je vykonávaná na úrovni funkcionálnych manažérov (na obrázku uvedené pod názvom Functional Team Leader). Spolupráca a komunikácia medzi tímami je vykonávaná na vyššej úrovni hierarchie. Pri tejto štruktúre tímov sú na projekt vyberaní ľudia z jednotlivých oddelení [5].

Zvoliť takúto štruktúru tímov je asi vhodnejšie v organizáciách, ktoré pracujú na veľkých systémoch podobného charakteru.

Projektová organizácia tímov

Príklad takejto organizácie tímov je znázornený na nasledujúcom obrázku.



Obr. 3. Príklad projektovej organizácie tímov [2].

Členovia tímov sú zoskupení podľa jednotlivých projektov, na ktorých sa podieľajú. Projektoví manažéri majú medzi sebou pomerne veľkú nezávislosť [2]. Aj táto organizácia tímov vytvára rôzne oddelenia. Tím reportuje priamo projektovému manažérovi a ako celok poskytujú služby pre rôzne projekty.

Takúto štruktúru tímov môže byť vhodnejšie použiť vo firmách, ktoré pracujú na mnohých projektoch rôzneho charakteru.

Maticová organizácia tímov

Kombináciou predchádzajúcich dvoch je maticová organizácia. Táto kombinácia môže byť realizovaná na rôznej úrovni. Výsledkom je slabá, vyvážená alebo silná maticová organizácia [2]. Takáto organizácia tímov často spôsobuje komplikovanejší manažment projektu.

Hoci takáto organizácia tímov prináša so sebou mnoho komplikácií do manažmentu projektu, myslím, že je asi najčastejšie používaná v praxi, pretože poskytuje najlepšiu flexibilitu pri výbere ľudí na jednotlivé projekty.

Záver

Každá z vyššie uvedených organizácii tímov má svoje výhody i nevýhody. Dobrý manažér projektu musí byť schopný prideliť tých správnych ľudí na jednotlivé projekty. Samozrejme, veľkú ulohu pri jeho rozhodovaní zohráva aj charakter projektu. Keďže každý projekt je jedinečný, je tažké vytvoriť nejaké najlepšie postupy pre manažéra projektu. Ten musí byť schopný zanalizovať všetky relevantné informácie a na ich základe organizovať tímy tak, aby sa dosiahli požadované ciele. Je v jeho kompetencii dotiahnuť projekt do úspešného konca tak, aby efektívne využíval dostupné ľudské zdroje.

Použitá literatúra

1. Beise, C.M.: IT Project Management and Virtual Teams. Salisbury, USA, 2004.
2. Duncan, W.R.: A Guide to the Project Management Body of Knowledge. PA, USA, 1996
3. Fox, T.L., Spence, J.W.: The Effect of Decision Style on the Use of a Project Management Tool: An Empirical Laboratory Study.
4. Staníček, Z., Kajkr, J.: Riadenie projektov zavádzania IS do organizácií. Brno, 2005 (cz)
5. Wallace, S.: The ePMbook. 2004

Annotation

Software project management and software team organization

The choice of appropriate people into the project has an large impact to its success. The task of project manager is to analyze all needed information and to make appropriate team structure to be able to manage the project into successful end. In this essay I describe some factors which affect the decision of project manager in case of team organization.

Manažment komunikácie pre lokalizované a distribuované softvérové tímy

Význam, komunikačné technológie a dôležité faktory vplývajúce na výkon distribuovaného softvérového tímu

VOJTECH SZÖCS

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
Xeno.Morph@gmx.de

Abstrakt. S požiadavkou na vývoj neustále zložitejších softvérových systémov sa model distribuovaného vývoja systému, resp. jeho súčiastok stal v súčasnosti štandardným riešením. Distribuované softvérové tímy stoja nielen pred úlohou samotného vývoja danej súčiastky systému, ale predovšetkým tiež pred úlohou vzájomnej koordinácie a komunikácie s úsilím úspešného dosiahnutia stanoveného cieľa. Ukazuje sa, že aspekty ako komunikácia, koordinácia a informačné povedomie majú zásadný vplyv na celkový výkon distribuovaného softvérového tímu. Dobre zvolená komunikačná technológia má pritom výrazný pozitívny vplyv na uvedené aspekty. Z hľadiska koordinácie sa v praxi osvedčila koncepcia koordinačnej autority, ktorá výrazne zefektívňuje proces koordinácie tímu a jej užitočná hodnota je citeľná najmä v problémových situáciách, do ktorých sa projekt môže časom dostať pod vplyvom rôznych okolností. Tento príspevok prezentuje zistené poznatky z oblasti manažmentu komunikácie distribuovaných softvérových systémov s dôrazom na komunikačné technológie a dôležité faktory ovplyvňujúce celkový výkon softvérového tímu. Pozornosť je venovaná tiež analýze koncepcie distribuovaných softvérových tímov najmä z hľadiska jej významu a charakteristických vlastností.

Úvod

Vo svete neustále sa rozvíjajúcich informačných technológií možno pozorovať trend, že nové systémy vo všeobecnosti sú čoraz komplexnejšie, sofistikovanejšie a modulárnejšie. Vývoj systémov, resp. ich komponentov už dávno nie je záležitosťou jednej osoby, čím vzniká potreba manažmentu softvérových tímov hlavne z hľadiska ich efektívnej koordinácie a komunikácie s ohľadom na dopad týchto faktorov na celkový výkon vývojových tímov.

Softvérové tímy možno z globálneho hľadiska rozdeliť na lokalizované a distribuované. Lokalizovaný softvérový tím je charakteristický tým, že všetci

Manažment komunikácie pre lokalizované a distribuované softvérové tímy, január 2006, s. 117-125.

členovia tímu sú priestorovo (a tiež časovo) lokalizovaní na jednom mieste. Toto miesto môže tvoriť napríklad oddelenie v organizácii (rôzne pracovné priestory v rámci tohto oddelenia) alebo v ideálnom prípade spoločný pracovný priestor.

V kontraste s touto koncepciou je distribuovaný softvérový tím, ktorého členovia sa nachádzajú na rôznych miestach a často krát aj v rôznych časových pásmach. Technológia CSCW [4] zahŕňa dvojrozmerný model kolaboratívnej práce distribuovaných softvérových tímov:

- *Rovnaký čas – rovnaké miesto*.
- *Rovnaký čas – rôzne miesto*.
- *Rôzny čas – rovnaké miesto*.
- *Rôzny čas – rôzne miesto*.

Z hľadiska distribuovaných softvérových tímov je najzaujímavejšia kombinácia *Rovnaký čas – rôzne miesto*, ktorá predstavuje zároveň hlavné zameranie tohto príspevku. Zvyšné kombinácie modelu kolaboratívnej práce majú potenciál v určitej miere využiť znalosti a skúsenosti nadobudnuté štúdiom uvedenej kombinácie [2].

Bandinelli et al. [5] poukazuje, že vzhľadom na kooperatívnu povahu vývoja softvérových systémov je úspech na projekte závislý „na kvalite a efektívnosti komunikačných kanálov vo vývojovom tíme“. Komunikácia v distribuovanom softvérovom tíme je bezpochyby kľúčovým aspektom, ktorý výrazne ovplyvňuje koordináciu a teda aj celkový výkon tímu. Potreba kvalitnej a efektívnej komunikácie je pritom obzvlášť dôležitá v projektoch, ktoré sa vyznačujú veľkou mierou zmien v čase [1]: meniace sa prostredie alebo požiadavky na systém, slabo alebo nejasne definované kritériá na výstupy alebo systém, atď.

Význam, príčiny vzniku a dôsledky

Distribuované softvérové tímy predstavujú vo všeobecnosti dôsledok nevyhnutných zmien v organizačnej štruktúre daného projektu [2]. Charakter týchto zmien vysoko závisí od rozsahu projektu: zatiaľ čo pri malých projektoch (zhruba 1 tím na projekt) môžu príčiny vzniku distribúcie tímu predstavovať nevyhnutnú priestorovú separáciu členov tímu (napríklad rôzne miesta pobytu členov), pri stredných alebo veľkých projektoch zahŕňajúcich niekoľko desiatok až stoviek tímov sú príčiny najčastejšie typu prispôsobenie sa požiadavkám klienta či nevyhnutná prítomnosť vysoko kvalifikovaných členov na strategických miestach.

Nie je možné jednoznačne určiť pozitívny alebo negatívny charakter distribúcie členov tímov vzhľadom na projekt, nakoľko koncepcia distribuovaných aj koncepcia lokalizovaných softvérových tímov majú špecifické výhody a nevýhody. Vlastnosti, výhody a nevýhody oboch koncepcií sú spravidla komplementárne – napríklad nevýhody distribuovaných softvérových tímov predstavujú výhody pri lokalizovaných tímoch a naopak.

Vhodnosť tej-ktorej koncepcie pre daný projekt je vysoko podmienená charakterom projektu, a to najmä z hľadiska jeho „veku“ a komplexnosti. Spoločnosť

Microsoft napríklad odporúča, aby sa vývoj nového systému lokalizoval podľa možností na jedno miesto, čo poskytuje výhodu okamžitých konzultácií vývojárov pri vzniku problémov alebo nejasností počas vývoja systému [6]. Možno tiež predpokladať, že počas vývoja naozaj veľkých systémov bude vhodné využiť distribuovaný model vývoja jednotlivých komponentov systému v štýle jeden tím resp. organizácia – jeden komponent, čo hovorí naopak v prospech koncepcie distribuovaných softvérových tímov. V takomto prípade je distribúcia predpokladom pre organizačne optimalizovaný, nezávislý a kolaboratívnou kultúrou riadený vývoj komponentov systému s vysokou úrovňou kvality výstupov, dokumentácie a vzájomnej komunikácie.

Dôsledky distribúcie softvérového tímu sú opísané jeho výhodami, nevýhodami, vlastnosťami a v neposlednej rade aj dôležitými faktormi. Uvedené aspekty sú bližšie vysvetlené v ďalších častiach.

Výhody a nevýhody

Na vytvorenie si dobrého obrazu o distribuovaných softvérových tímoch v kontexte nadväzujúcej analýzy a úvahy o komunikačných technológiách je potrebné poznať výhody a nevýhody tejto koncepcie. Výhody aj nevýhody (pozri [2]) sú pre zachovanie stručnosti uvedené v odrážkach.

Pozitíva

Medzi hlavné výhody distribuovaných softvérových tímov patria:

- Zvýšenie úrovne a formálnosti dokumentov, výstupov a komunikácie. Tento aspekt je dôsledkom toho, že členovia tímu nemajú navzájom vzhľadom na ich distribúciu nadviazané nežiadúce, príliš neformálne komunikačné alebo iné väzby.
- Členovia tímu sú viac nútení samostatne pracovať a študovať. Otázkami a relatívne malými problémami, ktoré môžu vyriešiť aj sami, nie sú zaťažovaní iní, väčšinou v danej oblasti vysoko kvalifikovaní členovia tímu.
- Separácia jednotlivých úloh v tíme je dobrá pre nemožnosť ovplyvniť iné časti alebo procesy systému (napríklad testovanie).
- Po čase môže medzi členmi tímu vzniknúť zdravá konkurencia v snahe vybudovať alebo zlepšiť si dobré meno či postavenie v tíme.
- Potreba štandardnej kolaboratívnej kultúry komunikácie a koordinácie v tíme prospieva k zvýšeniu celkového výkonu softvérového tímu.

Negatíva

Ako negatíva možno uvažovať nasledujúce aspekty:

- Nemožnosť poriadať „živé“ stretnutia (meetings), ktoré sú často vitálne pre úspešný priebeh projektu. Alternatívou je v tomto prípade nevyhnutná komunikácia s využitím vhodných komunikačných technológií.
- Distribúcia členov tímu dáva menšie predpoklady pre vybudovanie dobrých, neformálnych pracovných vzťahov v tíme. Nie je tajomstvom, že dobré vzťahy a pracovná atmosféra nemalou mierou prispievajú k výkonu jednotlivých členov tímu.
- Veľký dôraz na komunikáciu môže v krajnom prípade vyústiť do stavu, keď členovia tímu strávia viac času samotnou komunikáciou, než užitočnou prácou. Tento jav sa zvykne označovať ako komunikačné preťaženie.
- Veľký dôraz na informačné povedomie (prehľad o stave projektu a jeho komponentoch) členov tímu môže vyústiť do stavu zahltenia daného člena často krát preňho nepodstatnými informáciami. Tento jav sa zvykne označovať ako kognitívne preťaženie.

Dôležité faktory

Medzi dôležité faktory ovplyvňujúce výkon distribuovaného softvérového tímu patria koordinácia, komunikácia, informačné povedomie a komunikačné a kognitívne preťaženie.

Koordinácia

Koordinácia je kľúčový aspekt na dosiahnutie úspechu v projekte [1]. Je potrebné zabezpečiť určitú autoritu (napríklad tímový líder), ktorá bude túto funkciu plniť najmä v zmysle prideľovania úloh a zodpovedností, plánovania činností na projekte a kontroly plnenia plánu (nevyhnutná je dobrá viditeľnosť čiastkových výsledkov projektu).

V prípade väčších projektov, resp. vývoja komplexnejších systémov alebo ich komponentov možno pod koordináciou vnímať tiež proces integrácie jednotlivých súčiastok systému do fungujúceho celku. V takomto prípade je dôležitým faktorom informačné povedomie jednotlivých členov tímu, resp. viacerých vývojových tímov. Z globálneho hľadiska by činnosť koordinačnej autority mala byť výrazne integratívna, v kontraste so separatívnou činnosťou jednotlivých vývojových jednotiek (členovia tímu zodpovední za vývoj, resp. rôzne vývojové tímy) [2].

Bez prítomnosti koordinačnej autority je pravdepodobnosť úspechu distribuovaného tímu veľmi nízka, nakoľko jej funkcia je nevyhnutne distribuovaná na jednotlivých členov tímu (v určitej miere). Decentralizácia, resp. absencia koordinačnej autority takmer vždy spôsobuje vnesenie problémov súvisiacich s jej funkciou a má vážne následky na celkový výkon distribuovaného softvérového tímu.

Koordinácia úzko súvisí s komunikáciou a informačným povedomím členov tímu. Zároveň s týmito väzbami existuje nevyhnutné prepojenie koordinácie

s komunikačným a kognitívnym preťažením – extrém týchto faktorov má väčšinou na koordináciu rozsiahly negatívny dopad.

Komunikácia

Komunikácia v kontexte distribuovaných softvérových tímov sa týka predovšetkým schopnosti využitia vhodných komunikačných technológií a stratégií, najmä z hľadiska kvality a efektívnosti komunikačných ciest medzi jednotlivými členmi tímu. Voľba metód a technológií na zabezpečenie komunikácie má pritom zásadný vplyv na koordináciu distribuovaného softvérového tímu.

Komunikačné technológie

S postupom času a rozvojom nových technológií sa vo svete distribovaného vývoja softvéru objavuje čoraz viac komunikačných technológií, ktoré sa vzťahujú na určité komunikačné stratégie. Všetky tieto technológie majú za cieľ efektívnym a podľa možnosti najmenej invazívnym spôsobom prezentovať danému členovi tímu preňho relevantné informácie o všetkých aspektoch projektu.

Dôležitými faktormi v tomto kontexte sú asynchrónnosť (relevantné informácie sú pre daného člena k dispozícii v čase preňho vhodnom – tzv. „as needed" prístup) a úroveň agregácie relevantných informácií [1]. Agregáciu informácií tu možno vnímať ako prítomnosť určitého centrálneho informačného úložiska, v ktorom sa pre každého člena tímu nachádzajú preňho relevantné informácie o projekte. Inými slovami, jedná sa o myšlienku personalizácie informačného úložiska pre jednotlivých členov distribuovaného softvérového tímu.

Elektronická pošta (e-mail)

Táto technológia je vzhľadom na jednoduchosť jej použitia a vysokú rozšírenosť v súčasnosti najpoužívanejšou komunikačnou technológiou v distribuovaných softvérových tímoch. Niektorí autori publikácií o problematike komunikačných technológií a stratégií dokonca hovoria, že predstavuje najväčší prínos pre distribuované softvérové tímy [1].

Jedná sa o asynchrónnu a neinvazívnu technológiu, ktorá vykazuje aj určitý stupeň agregácie informácií v podobe poštovej schránky a možností jej personalizácie pre daného člena tímu. Ukazuje sa tiež, že v porovnaní s bežnými osobnými stretnutiami (meetings) má táto komunikačná technológia menší vplyv na neželané zvyšovanie kognitívneho preťaženia [1].

V kontexte informačného povedomia (informovanosť daného člena tímu o celkovom stave projektu) možno pri tejto technológii hovoriť o tzv. pasívnom informačnom povedomí [1]. Jedná sa o stav, keď má daný člen tímu možnosť čítať nielen relevantnú poštu preňho určenú, ale aj poštu určenú iným členom, resp. iným vývojovým tímom. Nepredpokladá sa, že daný člen bude pozorne študovať každú správu súvisiacu s projektom, no fakt, že má túto možnosť, môže výrazne prispieť k vybudovaniu si dobrého celkového obrazu o projekte a postupe prác na ňom.

Užitočnú hodnotu tejto technológie zvyšuje aj jej možnosť práce s ľubovoľným typom informácií v podobe príloh. Vzhľadom však na to, že koncepcia klasickej

elektronickej pošty už dávno dosiahla svojich hraníc (ohraničenia v súvislosti s prílohami, bezpečnosťou atď.) je viac ako pravdepodobné, že čoskoro vznikne nová technológia, ktorá bežnú elektronickú poštu nahradí a posunie jej hranice.

Nepretržitá komunikácia (instant messaging)

Čoraz viac sa v kontexte komunikačných technológií dostáva do popredia technológia nepretržitej komunikácie. Táto technológia pôvodne vznikla so všeobecným zámerom možnosti posielania správ v reálom čase, no čoskoro sa ukázal jej potenciál v oblasti manažmentu komunikácie distribuovaných softvérových tímov. Napriek jej výhodám zostáva v súčasnosti nosnou komunikačnou technológiou elektronická pošta a nepretržitá komunikácia sa využíva ako jej rozšírenie.

Vo všeobecnosti možno nepretržitú komunikáciu rozdeliť na privátnu a skupinovú. Privátna komunikácia zahŕňa výmenu správ dvoch osôb, bez možnosti zdieľania obsahu správ iným osobám. Možno ju prirovnať k technológii elektronickej pošty, ale v reálnom čase. Tento typ nepretržitej komunikácie je ideálny pre cielené a užšie zamerané rozhovory medzi dvoma členmi tímu.

Skupinová komunikácia zahŕňa na rozdiel od privátnej výmenu správ medzi viacerými osobami, ktoré sú v určitej skupine, navzájom. Všetci členovia danej skupiny majú možnosť zapájať sa do diskusie a nepretržite pozorovať jej priebeh. Tento prístup možno výstižne prirovnať k „informačnej videokonferencii“. Vhodný je predovšetkým v situáciách, keď sa jedná o širšie zámery, dôležité rozhodnutia či riešenie závažných problémov.

Napriek viditeľným pozitívam prináša táto komunikačná technológia so sebou aj výraznú mieru vplyvu na komunikačné preťaženie. Fakt, že členovia tímu majú možnosť nepretržite diskutovať o danom probléme, môže viesť (a často krát aj vedie) k odchýleniu sa od pôvodného zámeru diskusie – členovia tímu začínajú tráviť viac času komunikáciou, ako samotnou užitočnou prácou na projekte (komunikačné preťaženie).

Výmena súborov

Výmena súborov, resp. dokumentov alebo čiastkových výsledkov projektu je v súčasnej dobe už menej využívanou komunikačnou technológiou. Tvorí stupeň vývoja komunikačných technológií pred masívnym rozvojom elektronickej pošty. Dôvodom ústupu tejto technológie je najmä rozvoj technológií podporujúcich výmenu informácií (vrátane súborov) v reálnom čase.

Hybridný prístup s inovatívnymi prvkami

Vzhľadom na neustále narastajúci význam komunikácie v distribuovaných softvérových tímoch možno v súčasnosti vidieť snahy podporovať proces komunikácie vývojom softvérových nástrojov, ktoré integrujú princípy viacerých komunikačných technológií a zavádzajú inovatívne prvky pre zvýšenie efektívnosti podpory tohto procesu. Tieto nástroje majú vo väčšine prípadov zameranie vyššej úrovne ako úroveň manažmentu komunikácie distribuovaných softvérových tímov – často krát sa ich zameranie týka podpory manažmentu softvérového projektu ako celku (pre prehľad o takýchto nástrojoch pozri [7]).

Informačné povedomie

Nezanedbateľný vplyv na koordináciu a výkon jednotlivých členov distribuovaného softvérového tímu má úroveň ich informačného povedomia o stave projektu ako celku. Jedná sa najmä o aktuálny stav projektu, plnenie projektového plánu a postup a dosiahnuté výstupy iných členov tímu – predovšetkým tých, ktorých úlohy súvisia s daným členom. Dôležitosť informačného povedomia sa navyše umocňuje faktom, že tvorí kľúčový aspekt pri integrácii čiastkových výsledkov jednotlivých členov tímu (informácie, ako ďaleko je daná úloha, či aké sú priebežné výstupy). Pre podrobnejšie informácie o informačnom povedomí pozri tiež [1].

Podporné nástroje

Na dosiahnutie požadovanej miery informačného povedomia členov tímu je vhodné okrem bežných metód (napríklad metóda aliasov elektronickej pošty) využiť aj pre tento účel určené podporné nástroje. Fussell et al. [1] prezentuje vo svojej práci myšlienku dvoch takýchto nástrojov (tzv. awareness devices):

- *archív elektronickej pošty* (podobnosť s personalizovanou elektronickou nástenkou),
- *monitor projektových aktivít* (sledovanie tímových a osobných údajov a aktivít).

Kľúčovými faktormi pri návrhu takýchto podporných nástrojov sú predovšetkým asynchrónnosť (zníženie miery invazívnosti), agregácia informácií s využitím určitej miery personalizácie a pasívnosť informačného povedomia [1].

Pasívne informačné povedomie

Pasívne informačné povedomie predstavuje stav, keď má daný člen tímu možnosť sledovať informácie súvisiace s projektom popri jeho bežnej komunikácii, podľa možností v čo najmenej rušivej podobe.

Metóda aliasov elektronickej pošty

Z metód, ktorými sa budovanie, udržiavanie a zlepšovanie informačného povedomia u členov distribuovaného tímu realizuje, možno spomenúť často používané aliasy elektronickej pošty. Alias je v tomto prípade adresa elektronickej pošty, ktorá reprezentuje nie jednotlivca, ale skupinu jednotlivcov. Veľkosť a zloženie tejto skupiny, resp. viacerých skupín závisí najmä od distribúcie úloh a zodpovedností v danom tíme. Daný člen tímu má možnosť čítať poštu určenú skupine, do ktorej patrí, čím si vytvára jasnejší obraz o stave a postupe vývojových prác skupiny. V extrémnom prípade tvorí skupinu celý vývojový tím.

Komunikačné preťaženie

Komunikačné preťaženie predstavuje extrém v procese komunikácie, ktorý nastáva vtedy, keď je daný člen tímu zahltený komunikáciou na úkor užitočnej práce na

projekte. Príčinu dosiahnutia tohto extrému možno vo všeobecnosti rozdeliť na projektovo-orientovanú alebo osobnú.

Projektovo-orientovaná príčina súvisí so vznikom problémov v projekte. Najčastejšie sa jedná o nejasnosť požiadaviek na vytváraný systém a nejasnosť väzieb a interakcií medzi jednotlivými súčiastkami systému. Túto príčinu možno efektívne odstrániť budovaním informačného povedomia a zabezpečenia kvalitnej a efektívnej komunikácie.

Osobné príčiny v súvislosti s komunikačným preťažením sa stali závažnými najmä s príchodom nástrojov podporujúcich nepretržitú komunikáciu. V tomto prípade problém spočíva v členovi tímu, ktorý sa odkloní od pôvodného zámeru projektovej komunikácie a stráca čas irelevantnou diskusiou.

Kognitívne preťaženie

V súvislosti s informačným povedomím predstavuje extrémny stav kognitívne preťaženie, keď je daný člen tímu zahltený zväčša irelevantnými informáciami. Keď takáto situácia nastane, je nutné zvážiť rovnováhu potreby informovanosti daného člena tímu najmä z hľadiska jeho úloh, zodpovedností a zainteresovanosti na projekte. Pre podrobnejšie informácie o kognitívnom preťažení pozri tiež [3].

Vo všeobecnosti platí, že asynchrónne komunikačné prístupy (napríklad elektronická pošta) majú oproti synchrónnym (napríklad bežné osobné stretnutia či porady) oveľa menší vplyv na rast kognitívneho preťaženia. Tento fakt je dôsledkom toho, že daný člen tímu má schopnosť efektívnejšie „absorbovať“ relevantné informácie vtedy, keď sú mu tieto informácie poskytnuté podľa jeho potrieb a možností. V prípade osobných stretnutí môže napríklad dôjsť k prepočutiu dôležitých informácií alebo strate koncentrácie v prípade dlhších stretnutí. Technológia elektronickej pošty v kontraste s osobným stretnutím umožňuje sledovať relevantné informácie v takom tempe a v takej intenzite, ktorá vyhovuje danému členovi tímu.

Záver

Na základe prezentovaných informácií možno povedať, že aspekt komunikácie je pre úspešnú činnosť distribuovaných softvérových tímov viac než nevyhnutný. Dobre stanovená komunikačná stratégia, resp. dobre zvolená komunikačná technológia má výrazný pozitívny vplyv na koordináciu vývojového tímu a teda aj na jeho celkový výkon a úspešnosť v projekte.

V prípade komunikácie a koordinácie zohráva dôležitú úlohu kolaboratívna kultúra tímu. Dobre stanovené komunikačné procedúry pre možné (často krát problémové) situácie počas vývoja systému v tíme znižujú pravdepodobnosť vzniku komunikačného chaosu a preťaženia jednotlivých členov tímu.

Úroveň potreby koordinácie tímu je priamo úmerná distribúcii členov tímu a rozsahu projektu, na ktorom sa daný distribuovaný softvérový tím podieľa. Prítomnosť koordinačnej autority výrazne zefektívňuje proces koordinácie tímu a jej

užitočná hodnota je citeľná najmä v problémových situáciách, do ktorých sa projekt môže časom dostať pod vplyvom rôznych okolností.

Použitá literatúra

1. Fussel, S.R., Kraut, R.E., Javier Lerch, F., Scherlis, W.L., McNally, M.M., Cadiz, J.J.: Coordination, Overload and Team Performance: Effects of Team Communication Strategies. In Proc. of Computer Supported Cooperative Work, ACM, Seattle, Washington (1998).
2. French, A., Layzell, P.: A Study of Communication and Cooperation in Distributed Software Project Teams. In Proc. of International Conference on Software Maintenance, IEEE, Bethesda, Maryland (1998).
3. Franz, H.: The Impact of Computer Mediated Communication on Information Overload in Distributed Teams. In Proc. of Hawaii International Conference on System Sciences, IEEE, Maui (1999).
4. Marca, D., Bock, G.: Groupware: Software for Computer Supported Cooperative Work (Chapter 4)., IEEE Computer Society Press, 1992.
5. Bandinelli, S., Di Nitto, E., Fuggetta, A.: Supporting Cooperation in the SPADE-1 Environment. IEEE Transactions on Software Engineering, Vol. 22, No. 12 (1996) 841-865.
6. Cusamano, M., Selby, R.: Microsoft Secrets: How the worlds most powerful software company creates technology, shapes markets, and manages people. New York Free Press, 1995.
7. Sauter, V.L.: Comparison of the Project Management Software. http://www.umsl.edu/~sauter/analysis/488_f01_papers/surijamorn_files/48801.htm (10.12.2005).

Annotation

Communication management for localized and distributed software teams. Relevance, communication technologies and essential factors affecting the performance of a distributed software team.

With a demand for increasing complexity during the design of software systems, the distributed system or component design model has become a standard solution. Distributed software teams are confronted not only with the design phase, but also with mutual coordination and communication with the effort of successfully achieving their goal. It turns out, that the aspects like communication, coordination and informational awareness have significant effect on the overall performance of a distributed software team. A wisely chosen communication technology has a strong positive impact on the aspects stated. In terms of team coordination the concept of coordination authority has proven to be essential with a major effect on the coordination process efficiency increase. The actual value of coordination authority is notable especially in problem situations, which the project may encounter during the course of various circumstances. This paper presents the results and conclusions from a study of communication management of

distributed software teams with emphasis on communication technologies and essential factors affecting performance of a distributed software team. Attention is also paid to the analysis of the distributed software team concept, particularly its relevance and characteristics.

Zhrnutie

V prvej časti publikácie bola zdôraznená potreba vymedzenia rozsahu a rizík softvérového projektu. Riziká definujú aj samotný manažment ako snahu o ich minimalizáciu a elimináciu. Základné riziká predstavuje neskoré ukončenie projektu, prekročenie rozpočtu na projekt a nedostačujúca funkcionalita riešenia. Niektoré riziká môžu byť na sebe závislé, t.j. ovplyvňujú sa navzájom. V prípade závislých rizík je metodika ich manažmentu náročnejšia ako v druhom prípade.

Autori sa v tomto bloku esejí navyše zaoberali otázkami manažmentu kvality, manažmentu chýb, sledovania postupu softvérového projektu, a v neposlednom rade aj procesmi plánovania a odhadovania.

Projektoví manažéri sa pri svojej náročnej práci často dopúšťajú chýb. Pri vytýčení si nerealistických cieľov je spochybnená samotná podstata projektu. V prípade, že dôjde k nesprávnemu odhadnutiu najmä časových zdrojov na riešenie projektu, dochádza k nestabilnej situácii. Isto nečerpám len z vlastnej skúsenosti, ak poznamenám, že v súčasnosti väčšina najmä menších firiem v snahe vyhovieť akýmkoľvek časovým nárokom zákazníka, aj keď zjavne nereálnym, zaťaží svojich projektových inžinierov do takej miery, že to ohrozí samotný beh projektu. Tlak síce za určitých podmienok vytvára zrýchlenie produktivity, ale za obrovskú cenu zníženia kvality. Odhadovanie zdrojov je veľmi zložité a najmä nepresné, a preto je dobré preferovať pesimistické odhady pred optimistickými. Pri riadení projektov je podobne potrebné vyvarovať sa použitiu neoverenej technológie, ak je to možné. Z pohľadu priamej komunikácie s pracovníkmi tímu je závažnou chybou, ak v projekte nevládne pozitívna motivácia, ba dokonca dochádza zo strany manažmentu k negatívnej motivácii v prípade stavu tiesne.

Cieľom každého softvérového projektu je dosiahnutie kvality v najvyššej možnej miere. Kvalita produktu nie je zaručená vytvorením oddelenia kvality, a nemôže ju s určitosťou zaručiť ani certifikát kvality. Podľa mienky autora eseje z oblasti manažmentu kvality, ktorá je súčasťou tohto zborníka, a nezávislého testu, na ktorý sa autor odvoláva, u korporácií so stabilným oddelením kvality dochádza k uvoľneniu verzií produktov s kritickými chybami dokonca častejšie ako u firiem bez tohto oddelenia.

Každý softvérový projekt má určitý postup, ktorý treba sledovať. Toto sledovanie je veľmi dôležitým faktorom ovplyvňujúcim úspešnosť projektu, pričom jeho základom je zistenie aktuálneho stavu, v ktorom sa projekt nachádza, s čo možno najväčšou presnosťou za predpokladu určenia všetkých kľúčových charakteristík. Pri sledovaní je možné použiť určité hotové nástroje a metódy, avšak treba mať na zreteli, že nie všetky sú presné.

Druhá časť zborníka sa venovala problematike základnej funkčnej jednotky, ktorá zohráva najdôležitejšiu úlohu pri riešení projektu, a síce tímu. V každom tíme, a to nielen v tíme riešiacom problémy z oblasti softvérového inžinierstva, je dôležitá

organizácia a komunikácia jednotlivých štruktúrnych jednotiek tímu. V zmysle efektivity tímovej práce bol osvieženým pojem zvyšovania produktivity tímu a manažmentu konfliktov v tíme. Tím bol v priereze tejto časti zborníka chápaný ako ucelená funkčná jednotka v zmysle určitej vlastnej evolúcie, ktorá je neoddeliteľnou súčasťou jeho existencie.