Zborník esejí

Manažment
v softvérovom inžinierstve

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Bratislava
2006

Manažment v softvérovom inžinierstve – zborník esejí

Editor:
Peter Židek

Autori:
Ľubomír Hromádka, Vladimír Krivuš, Peter Ledňa, Matej Macháč, Tomáš Minčeff, Andrej Neczli, Peter Prikryl, Pavol Prikryl, Miloš Radošinský, Ján Suchal, Peter Vojtek, Ondrej Žáry, Peter Židek

Obálka:
Ján Suchal

Titulné strany:
Tomáš Minčeff

CD:
Matej Macháč

Webová stránka:
Ondrej Žáry

Gramatická kontrola:
Tomáš Matúšek, Pavol Prikryl, Peter Prikryl

Technická a organizačná podpora:
Ľubomír Hromádka, Peter Ledňa, Vladimír Krivuš, Andrej Neczli, Miloš Radošinský, Peter Vojtek

Publikácia je zbierkou esejí skupiny autorov vytvorených v rámci predmetu Manažment v softvérovom inžinierstve realizovaného na Fakulte informatiky a informačných technológií Slovenskej technickej univerzity v Bratislave.

Prvé vydanie
Dátum vydania: 12. január 2006
Počet výtlačkov: 1

Obsah

Úvod

Cieľom tohto zborníka je oboznámiť čitateľa s niektorými známymi, ale aj menej známymi témami z oblasti manažmentu v softvérovom inžinierstve. Autori ponúkajú základné fakty o zvolenej téme a rozširujú ju o svoje poznatky a skúsenosti a snažia sa ju aplikovať na predmet Tvorba softvérového systému v tíme. Keďže sa nachádza v zborníku väčšie množstvo tém, uvediem ich stručný prehľad a charakteristiku.

Téma manažmentu rizík v softvérovom projekte sa snaží poukázať na spôsoby ako predchádzať rizikám a prípadne minimalizovať ich škody na projekte. Autor uvádza informácie o jednotlivých fázach procesu. Sem patria identifikácia rizík, analýza rizík, plánovanie manažmentu rizík a riadenie rizík.

Agilné metódy programovania sú v súčasnosti používaným postupom hlavne pri malých projektoch. Autor sa zameriava na extrémne programovanie ako hlavného predstaviteľa agilných metód. Vysvetľuje základné informácie a postupy a tiež sa snaží poukázať na možnosti použitia tejto metódy pre väčšie projekty.

Virtuálna softvérová spoločnosť je témou ďalšej časti tohto zborníka. Jej autor sa snaží oboznámiť čitateľa s definíciou Virtuálnej Softvérovej Spoločnosti a tiež s nástrojmi, ktoré využívajú pri komunikácií. V druhej časti sa zaoberá možnými problémami virtuálnych tímov a porovnáva ich s lokálnymi.

Kvalita softvéru je dôležitým faktorom pri hodnotení softvéru, ale aj spoločnosti, ktorá ho vytvorila. Esej, ktorá sa venuje téme kvality, sa snaží nahliadať na kvalitu softvéru v kontexte manažmentu kvality. Jednotlivé znaky kvality hodnotí z hľadiska dôležitosti pre zákazníka a tvorcu. Tiež sa zaoberá pojmom chyby a na záver ponúka pohľad na manažment kvality ako na systém riadenia.

Pri téme manažment softvérového systému a vplyv na manažment softvérového projektu sa autor snaží oboznámiť so základnými konceptami systémov pre manažment softvéru. Tiež sa zaoberá jeho vplyvom na manažment projektu.

V téme zlepšovanie produktivity softvérových tímov sa autor zaoberá jednotlivými faktormi, ktoré na ňu vplývajú a v druhej časti sa zaoberá najvýraznejším z nich – schopnosťami tímu. Tu sa venuje osobnostnému zloženiu menšieho tímu a snaží sa poukázať na vhodnosť rôznych typov osobností na rôzne pozície v tíme. Okrajovo sa zaoberá aj psychológiou osobnosti a jej typológiou MBTI (Mayers-Briggs Type Indicator).

Odhadovanie v softvérových projektoch je ďalšou témou v tomto zborníku. Autor v nej definuje základné fázy procesu odhadovania a poukazuje na potrebu skúseností pri odhadoch. Tiež sa zameriava na najvážnejšie problémy pri odhadovaní a ponúka k nim možné riešenia. Na záver pridáva pár užitočných rád a pripomienok, ktoré majú uľahčiť a spresniť odhady.

Ďalšou témou v tomto zborníku je Chyba a manažment softvérového projektu. Jej autor sa zaoberá príčinami vzniku chýb v softvérovom projekte. Uvažuje nad chybami z hľadiska vplyvu na zákazníka. Tiež sa zaoberá testovaním ako spôsobom

minimalizácie chýb. Na záver naznačuje akú cenu môže mať chyba a jej závislosť na viacerých faktoroch.

Vzťah medzi zákazníkom a vývojárom je dôležitou časťou celého procesu vývoja softvéru, a preto je zaradený ako jedna z tém tohto zborníka. Článok sa venuje problémom v komunikácií medzi zákazníkom a vývojárom ako aj úlohe zákazníka pri vývoji softvéru. V druhej časti sa zameriava na agilné metódy a komunikáciu zákazníka s vývojárom pri ich použití. Na záver ponúka niekoľko riešení tejto komunikácie, ktoré môžu zefektívniť vývoj softvéru.

Sledovanie postupu softvérového produktu a manažment je téma, ktorá sa zaoberá plánovaním projektu. Autor sa zameriava na možnosti plánovania projektu a možnosti kontroly dodržiavania plánu. Tiež uvádza metódy pre meranie postupu v projekte. Na záver sa tiež zaoberá dôvodmi, prečo je vhodné plánovať a kontrolovať dodržiavanie plánu.

Na vývoj tímu v softvérovom projekte a vplyv na manažment je možné nazerať z viacerých strán. Autor si vybral tri modely vývoja tímu: Bruce Tuckmanov model, Herseyho a Blanchardov situačný vodcovský model a Tannenbaumovo a Schmidtovo kontinuum. Tiež sa zaoberá úlohami manažéra vo vývoji tímu v súvislosti s uvedenými modelmi.

V súčasnosti sa objavuje množstvo nových technológií a nástrojov, ktoré sa snažia zjednodušiť vývoj softvéru. Novými technológiami a nástrojmi a ich vplyvom na manažment softvérového projektu sa zaoberá autor eseje s rovnakým názvom. Autor sa zameriava hlavne na nástroje, ktoré uľahčujú tvorbu dokumentácie k projektu a tiež na tie, ktoré zjednodušujú komunikáciu medzi členmi tímu. Tiež sa zaoberá kritériami výberu vhodného nástroja.

Vyvíjanie softvéru v tíme so sebou prináša aj negatíva. Jedným z nich sú konflikty, ktoré môžu v tíme vzniknúť. Touto témou sa zaoberá esej s názvom Manažment konfliktov v tíme. Autor sa venuje možným dôvodom vzniku konfliktu a následne sa snaží ponúknuť riešenia vzniknutého konfliktu.

Manažment ako celok

Úvod

Vzhľadom na množstvo informácií týkajúcich sa manažmentu vývoja softvéru, ktoré sú uvádzané v tomto zborníku, rozhodol som sa venovať manažmentu ako univerzálnemu nástroju na riadenie. Manažment sa skladá z viacerých odvetví a činností. Jednou z nich je práve manažment v softvérovom inžinierstve. Pokiaľ sa však pozeráme na manažment ako univerzálny nástroj, obsahuje aj viaceré časti, ktoré sa softvérového inžinierstva netýkajú.

Manažment ako celok

Manažment je možné chápať vo viacerých podobách. V jednej je to činnosť, ktorej úlohou je organizovanie skupiny pracovníkov s cieľom dosiahnuť výsledky, ktoré nedokáže vytvoriť jednotlivec. Druhým variantom je pohľad na manažment ako na skupinu ľudí, ktorej úlohou je:

- Plánovanie
- Organizovanie
- Personálne zabezpečenie
- Vedenie
- Kontrola

Plánovanie

Plánovanie je jednou zo základných úloh manažmentu. Jeho cieľom je vytvorenie predstavy o budúcnosti podniku a tiež prinášať lepšie finančné výsledky firmy. Úlohou plánovania je aj rozhodnúť o tom, ktoré aktivity firmy sa budú rozvíjať a ktoré naopak ustúpia do úzadia. Prostredníctvom plánovania je možné použiť doterajšie výsledky na predvídanie a dosahovanie lepších výsledkov, rovnako ako doterajšie problémy využiť na predchádzanie ďalším problémom. Plánovanie ako také je možné chápať aj ako nástroj na prispôsobovanie vlastných zdrojov okoliu. Bez ohľadu na spôsob nazerania na plánovanie, jeho hlavnou a najdôležitejšou úlohou je vypracovanie cieľov.

Plánovanie je nutné založiť na reálnych informáciách získaných z prostredia, kde firma pôsobí, ale tiež musí brať do úvahy podmienky a zdroje samotnej firmy. V prípade informácií o okolitom prostredí by to malo byť najmä zistenie informácií o možnej konkurencií, potenciálnych dodávateľoch a odberateľoch, čiže zákazníkoch. Dôležitými faktormi sú aj rôzne ekonomické, politické alebo prírodné činitele, kam sa radia napríklad príjmy potenciálnych zákazníkov alebo právne predpisy a normy. Medzi zdroje firmy, ktoré je nutné brať do úvahy, sa radia finančné zdroje, personálne možnosti firmy ako aj organizačné faktory.

Organizovanie

Cieľom organizovania je rozdelenie podniku na časti a štruktúry, ktoré by boli samostatné a nezávislé. Činnosti väčšiny oddelení sa odlišujú aspoň minimálne, avšak môže existovať viacero oddelení, ktorých činnosti sú totožné alebo sa niektoré ich oblasti prekrývajú. Význam organizovania je snaha o vytváranie trvalých vzťahov, ktoré sa využívajú pri plnení konkrétnych opakovaných úloh, tiež sa snaží zaviesť právomoci a zodpovednosti pre všetky činnosti a vytvárať produktívny systém v spoločnosti.

Vo firme môžu existovať rôzne organizačné štruktúry. Pre malé firmy s minimálnym počtom pobočiek to je jednoduchá organizácia, kde jedna osoba riadi všetky činnosti. Pre väčšie podniky je táto štruktúra nevhodná a nepoužiteľná, preto sa

pristupuje k rozdeleniu firmy na funkčné úseky. Tu sú už zamestnaní špecialisti, ktorí danú oblasť riadia. Druhou možnosťou je rozdelenie firmy na objektovú štruktúru. V tomto prípade sa firma delí na časti, ktoré sa tvoria na základe geografického rozloženia alebo podľa zamerania.

Personálny manažment

Personálny manažment sa snaží riadiť a organizovať pracovníkov tak, aby podávali čo najlepšie pracovné výkony. Na úspešné dosiahnutie tohto cieľa vplýva niekoľko faktorov. Asi najdôležitejším faktorom je samotný človek, jeho osobnosť a predpoklady pre danú prácu. Podobným faktorom je aj vzťah človeka k práci, pretože aj keď má predpoklady, nemusí ho tá práca zaujímať. Vplyv na výkonnosť pracovníka má aj správna analýza jeho práce a podmienok, v ktorých pracuje. Nemálo vplýva na zamestnanca aj jeho sociálna a ekonomická situácia rovnako ako aj jeho potenciálne zlepšovanie a vzdelávanie sa v pracovnej oblasti.

Spôsob ako zabezpečiť vysokú výkonnosť zamestnancov je správna motivácia. Motiváciou by sme mohli nazvať záujmy alebo motívy zamestnanca, ktoré sú hlavným dôvodom jeho konania. Cieľom manažmentu je v tomto prípade snaha o pochopenie, vysvetlenie, predvídanie a dokonca aj ovplyvnenie.

Medzi úlohy personálneho manažmentu sa tiež radí:

- výber zamestnancov, ktorý sa začína pri vypísaní ponuky na pracovné miesto až po prijatie zamestnanca do pracovného pomeru,
- zaúčanie a adaptácia do nového pracovného prostredia,
- zisťovanie príčin odchodu zamestnancov a snaha zabrániť tomuto javu.

Vedenie

Vedenie ľudí nie je len motivovanie ich k lepším výkonom. Je nutné zabezpečiť spokojnosť a pohodu zamestnancov nielen v práci, ale aj mimo nej. Vedením ľudí sa snažíme, aby pracovali dobrovoľne a s radosťou. Tiež máme záujem na tom, aby sa plne využívala a naďalej rozvíjala ich iniciatíva. Veľmi pozitívny vplyv na výkon pracovníka má aj dobrý vzťah medzi nadriadeným a podriadeným, ktorý podporuje spoluprácu a tiež ak môže pracovník plne využívať svoju kvalifikáciu a naďalej si ju rozvíjať.

Pre úspešné vedenie potrebujeme mať zamestnaných úspešných a kvalitných vedúcich. Vedúci pracovník musí okrem znalosti oblasti, v ktorej je vedúcim mať aj množstvo iných schopností a znalostí. Dôležitou znalosťou je sociológia, psychológia a pedagogika z pohľadu správania sa ľudí a pôsobenia na nich. Vedúci pracovník takisto potrebuje mať dostatočné množstvo skúseností s vedením, a teda nemôže bez skúseností nastúpiť ako vedúci päťdesiatčlenného tímu. Vzhľadom na veľké množstvo činností spadajúce do jeho kompetencií potrebuje mať dostatok času na výkon svojej funkcie, a teda nemôže zastávať iné posty vo firme.

Medzi riadiace činnosti sa radí najmä prideľovanie úloh, prideľovanie právomocí a kompetencií, hodnotenie pracovníkov a samozrejme motivovanie k lepším výkonom.

Kontrola

Úlohou kontroly je porovnávanie dosahovaných výsledkov s naplánovanými. Kontrola sa môže vzťahovať na rôzne činnosti – kontrola vedenia, kontrola riadiacej činnosti alebo kontrola výkonnej činnosti. Kontrola ako celok sa dá rozdeliť na etapy, ktoré vyjadrujú postup pri jej vykonávaní.

1. Presné definovanie cieľov a noriem, ktoré sa majú dodržiavať (napr. výkonnosť alebo predaj).
2. Porovnanie reálnej situácie s naplánovanou. V náväznosti na zistené rozdiely je nutné vykonať analýzu príčin, a to aj v prípade, že výkonnosť prekročila očakávania.
3. V prípade nedostatočného plnenia plánu je nutné zaviesť opatrenia, ktoré sa pokúsia tento stav napraviť.

Cieľom kontroly môže byť ľubovoľná časť podniku alebo činnosti. Vo všeobecnosti však môžeme kontrolu rozdeliť podľa niekoľkých kritérií. Prvým kritériom je pôvod kontroly, a teda či je kontrola vykonávaná osobami, ktoré do firmy nepatria (externá), alebo ju vykonávajú zamestnanci firmy (interná). Druhým pohľadom, podľa ktorého je možné rozdeliť kontrolu, je úroveň riadenia, ktoré kontrolu inicializovalo, a teda či je to vrcholová úroveň riadenia alebo nižšie riadenie. Kontrola môže byť zameraná na celý podnik alebo len na niektorú jeho časť alebo časti, a preto delíme kontrolu aj na všeobecnú a špecifickú. Posledným spôsobom delenia kontroly je podľa času jej vykonávania voči kontrolovanému objektu. Kontrola teda môže byť predbežná, priebežná alebo následná. V prípade správneho kombinovania jednotlivých typov kontrol môžeme dosiahnuť kvalitnú kontrolu, a teda vieme zabezpečiť kvalitnejší beh firmy.

Manažment rizík v softvérovom projekte

ĽUBOMÍR HROMÁDKA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
hromadka01@student.fiit.stuba.sk

Abstrakt. Manažment rizík v softvérovom projekte hrá dôležitú úlohu počas celého behu projektu. Jeho úlohou je včas rozpoznať a ošetriť nepredvídané situácie v projekte tak, aby vôbec nevznikli alebo spôsobili čo najmenšie škody. Manažment rizík pozostáva z identifikácie rizík, analýzy rizík, plánovania manažmentu rizík a riadenia rizík. Esej dáva ucelený pohľad na jednotlivé fázy manažmentu rizík, poskytuje prehľad najzávažnejších rizikových faktorov identifikovaných skúsenými projektovými manažérmi a snaží sa poskytnúť informácie o manažmente rizík v rámci tímového projektu počas štúdia na FIIT STU.

Úvod

Výdavky na softvérové projekty každoročne celosvetovo vzrastajú. S rastúcim počtom softvérových projektov rastie aj počet neukončených alebo neúspešných projektov. Alarmujúce sú čísla informujúce o výdavkoch na neúspešné projekty. Zo štúdie zverejnenej v roku 1998 [4] vyplýva, že už pred desiatimi rokmi dosahovali výdavky na neúspešné projekty v Spojených štátoch desiatky miliárd dolárov a predstavovali asi tretinu (!) celkových výdavkov na softvérové projekty.

Zlyhanie projektu má často na svedomí nedostatočný manažment rizík. V našej eseji venovanej manažmentu rizík sa pokúsime voviesť čitateľa do problematiky manažmentu rizík. Rozsah témy je značný, a preto nemôžeme podrobne rozobrať každý aspekt manažmentu rizík. Podrobnejšie sa venujeme identifikácii rizík pomocou rámca vytvoreného na základe panelovej diskusie projektových manažérov [4] a [6].

V eseji sa tiež snažíme prihliadať na manažment rizík v súvislosti s projektmi riešenými v rámci štúdia v približnom rozsahu tímového projektu na FIIT STU.

Vymedzenie pojmu riziko

Riziko softvérového projektu je možnosť výskytu nejakej udalosti alebo zhody nepriaznivých okolností, ktoré majú negatívny vplyv na projekt, jeho ciele (čiastkové

alebo finálne), plán alebo náklady. V [3] je riziko definované ako „možnosť, *nebezpečenstvo straty, neúspechu, škody“*.

Z hľadiska manažmentu softvérového projektu je niekedy za riziko považovaná nielen možnosť výskytu udalosti, ktorá, ak sa vyskytne, má negatívny dopad na ciele projektu (hrozba), ale aj možnosť výskytu udalosti, ktorej dopad na projekt je pozitívny (príležitosť). Vo všeobecnosti pre prístup k „pozitívnemu riziku“ platí opak než pri „negatívnych rizikách“, t.j. napríklad ak pri hrozbách sa snažíme minimalizovať pravdepodobnosť a dopad ich zhmotnenia, tak pri príležitostiach sa snažíme zvýšiť šance a dopad ich uskutočnenia. Slovo riziko budem ďalej používať len v jeho častejšie používanom, t.j. negatívnom, význame.

Pojem rizika sa uvažuje s ohľadom na možnosť utrpenia straty, pričom sa vyhodnocuje pravdepodobnosť jeho výskytu a rozsah prípadne spôsobených škôd. Manažment rizík v softvérovom projekte sa zameriava na elimináciu týchto dvoch faktorov, t.j. na zníženie pravdepodobnosti nastatia negatívnej udalosti, prípadne na minimalizáciu ujmy spôsobenej zhmotnením rizika. Manažment rizík sa nezaoberá výskytom udalostí, ktoré sú očakávané a v projektoch bežné – na ne sa zameriava „štandardný“ manažment.

Manažment rizík treba vykonávať s prihliadnutím ku konkrétnemu projektu. Jeho dôležitosť závisí od povahy a rozsahu projektu, nikdy ho však nemôžeme vynechať, pretože výskyt nejakej neštandardnej udalosti počas projektu je viac ako pravdepodobný a riešenie mimoriadnej udalosti spôsobenej zhmotnením neuvažovaného rizika si môže vyžiadať veľmi veľa úsilia a často má za následok v lepšom prípade predĺženie trvania projektu alebo zvýšenie nákladov na projekt, v horšom prípade neúspech celého projektu.

Procesy manažmentu rizík v softvérovom projekte

Manažment rizík v softvérovom projekte zahŕňa procesy spojené s identifikáciou a analýzou rizík a s následným plánovaním opatrení, ktoré treba prijať v súvislosti s identifikovanými rizikami. Podľa [5] môžeme procesy manažmentu rizík rozdeliť nasledovne:

- Identifikácia rizík – určenie, ktoré riziká sú pre daný projekt významné a zdokumentovanie ich charakteristiky.
- Analýza rizík – vyhodnotenie jednotlivých rizík a ich vzájomného pôsobenia s cieľom vyhodnotiť ich závažnosť, stanoviť priority a analyzovať možné reakcie na zhmotnenie rizika.
- Plánovanie manažmentu rizík – prijatie konkrétnych opatrení na ošetrenie rizík.
- Riadenie rizík – sledovanie rizikových faktorov a reakcia na zmeny v rizikách počas životného cyklu projektu.

Jednotlivé procesy bližšie opíšeme v nasledujúcom texte.

Identifikácia rizík

Proces identifikácie rizík pozostáva z určenia významných rizík, u ktorých je pravdepodobné, že ovplyvnia projekt, a z dokumentovania typických charakteristík identifikovaných rizík.

Na prvý pohľad sa môže zdať, že identifikácia rizík prebieha jednorazovo v počiatočných fázach projektu pri jeho plánovaní. V skutočnosti je potrebné, aby sa identifikácii nových rizík venovala pozornosť pravidelne počas celého projektu.

Identifikácia rizík spočíva v zistení, aké nebezpečné nepredvídané situácie môžu v projekte nastať a čo vedie k týmto situáciám (takzvané spúšťače udalostí). Pre každé riziko je potrebné určiť pravdepodobnosť výskytu udalosti a vykonať odhad rozsahu škôd pri jej nastatí. Vyjadrením pravdepodobnosti výskytu udalosti i jej dôležitosti (z pohľadu rozsahu spôsobených škôd) nemusí byť presné číslo, väčšinou sa namiesto toho používa stupnica vyjadrujúca relatívne porovnanie hodnôt (napr. nízka, stredná, vysoká). Riziká spojené s výstupom projektu sú často opísané na základe ich predpokladaného dopadu na rozpočet a časové aspekty projektu.

Klasifikácia rizík

Pri identifikácii rizík má zmysel zamyslieť sa nad povahou rizík.

Z hľadiska riadenia rizík rozlišujeme interné riziká a externé riziká. Interné riziká sú také, ktoré môže projektový tím riadiť alebo ovplyvňovať, a preto je možné prijať náležité opatrenia, aby sa tieto riziká minimalizovali. Externé riziká sú mimo pôsobnosti projektového tímu, ktorý na ne nemá dosah. Ak sa ďalšou analýzou externého rizika zistí, že potenciál jeho výskytu je vysoký a jeho dopad môže byť závažný, tak je vecou vrcholového manažmentu, aby na také riziko zareagoval, napr. jeho zohľadnením v celkovej cene vytváraného softvérového produktu alebo aj prípadným odstúpením od rizikovej zákazky.

Ďalšou kategorizáciou rizík je ich rozdelenie na všeobecné riziká týkajúce sa väčšiny softvérových projektov a špecifické riziká súvisiace s konkrétnym riešeným projektom. Systematickým prístupom k analýze rizík je možné ušetriť prácu znovupoužitím poznatkov o všeobecných rizikách.

Faktory vstupujúce do identifikácie rizík

Veľký význam pri identifikácii rizík zohráva povaha vytváraného produktu, výstupy plánovania projektu, ale aj skúsenosti projektového tímu a informácie o predošlých projektoch.

Povaha výstupu softvérového projektu má podstatný vplyv na identifikované riziká. Projekty založené na známych technológiách a postupoch nesú so sebou zvyčajne menšiu mieru rizika než projekty vyžadujúce nasadenie nových technológií.

Pri identifikácii rizík môžu byť nápomocné informácie z predošlých riešených projektov. Tieto údaje môžu byť vo forme záznamov v projektových dokumentoch, alebo to môžu byť spomienky a skúsenosti členov tímu s predchádzajúcimi projektmi, ktoré sú však vo všeobecnosti menej spoľahlivé než zdokumentované výsledky.

Výstupy procesov z iných oblastí manažmentu projektu sú tiež dôležitým zdrojom informácií, na základe ktorých sa identifikujú riziká. Príkladom môžu byť odhady nákladov a odhady časových požiadaviek projektu. Agresívne odhady a odhady vykonané na základe limitovaného množstva informácií prinášajú so sebou vyššie riziko. Iným príkladom je plán využitia pracovníkov. Ak je projektový tím postavený na niekoľkých skúsených členoch so špecifickými znalosťami, tak treba prihliadať na to, že môže byť ťažké nahradiť členov projektového tímu v prípade ich náhleho výpadku alebo z dôvodu ich vysokého vyťaženia na iných úlohách s vyššou prioritou, ktoré musia riešiť na iných projektoch v rámci organizácie.

Špecificky, v rámci tímového projektu riešeného na našej fakulte treba hlavne pri zostavovaní časového plánu projektu myslieť na to, že riešitelia – spolužiaci majú množstvo iných povinností aj v iných predmetoch a v diplomovom projekte, ktorých termíny navyše zvyknú byť „nakopené“ (napríklad ku koncu semestra). Preto je nutné plán projektu vytvoriť tak, aby výstupy projektu odovzdávané v jednotlivých kontrolných bodoch riešenia projektu boli pripravené na revíziu skôr než na poslednom stretnutí tímu pred dátumom odovzdania, pretože potom nemusia mať členovia tímu dostatok času na ich záverečné úpravy a v prípade časového sklzu sa nevyhnutne odovzdá produkt slabšej kvality.

Závažným problémom v rámci tímového projektu je, keď niektorý člen tímu z tímu predčasne odíde. Toto riziko by mal skúsený vedúci tímu včas identifikovať (väčšinou študent neukončí spoluprácu zo dňa na deň, ale postupne si prestane plniť svoje povinnosti a prestane mať záujem na fungovaní tímu a dosahovaných výsledkoch) a prijať opatrenia, aby sa odchod jedného člena z tímu významne neprejavil na dosiahnutých výsledkoch. Toto sa dá dosiahnuť napr. dôsledným vyžadovaním hmatateľných priebežných výstupov od každého člena tímu, aby odchodom nedisciplinovaného člena nevznikla potreba dorobiť jeho polsemestrovú prácu. V prípade, že člen tímu už odišiel, je možné situáciu riešiť určením menej ambicióznych cieľov pre zmenšený tím.

Techniky identifikácie rizík

Existuje viacero techník, ktoré pomáhajú identifikovať možné riziká. Patria sem:

- Zoznamy rizík – sú typicky organizované podľa zdroja rizika. Realizujú sa často vo forme dotazníkov alebo klasifikačných schém pre zdroje rizík.
- Dekompozícia – uľahčuje pochopenie rizík, napomáha pri identifikácii rizikových častí systému.
- Analýza predpokladov – skúma predovšetkým optimistické predpoklady, ktorých nesplnenie môže predstavovať riziko.
- Interview – diskusia všetkých zúčastnených strán (nielen členov vývojového tímu) môže viesť k odhaleniu neuvažovaných rizík.

Dobrý projektový vedúci by mal vedieť identifikovať nové hroziace riziko aj počas behu projektu.

Výstupy identifikácie rizík

Dôležitou súčasťou identifikácie rizík je zdokumentovanie identifikovaných rizík. Ide o zachytenie nasledovných poznatkov:

- Zdroje rizík – úplný zoznam rizík bez ohľadu na ich závažnosť alebo pravdepodobnosť ich zhmotnenia. Zoznam by mal pre každé riziko obsahovať odhad pravdepodobnosti jeho zhmotnenia, rozsah možných následkov, určenie časových súvislostí a predpokladanú frekvenciu výskytu.
- Identifikácia možných spúšťačov rizika.
- Potreba ďalšej aktivity v inej oblasti manažmentu projektu v súvislosti s identifikáciou rizík.

Analýza rizík

Analýza rizík zahŕňa vyhodnotenie rizík a ich vzájomných súvislostí. Jej cieľom je priorizovať riziká a určiť, ktoré riziká si vyžadujú nejakú reakciu. Analýza rizík musí zohľadniť viaceré faktory, medzi ktoré patria:

- Tolerancia zúčastnených strán voči rizikám – závisí od konkrétnej organizácie (či už ide o zákazníka alebo dodávateľa), napr. pravdepodobnosť prekročenia rozpočtu o 10% môže byť pre niektorých zákazníkov nepodstatná a pre iných môže byť považovaná za neprípustnú. V tímovom projekte a podobných projektoch počas štúdia je napríklad dôležité zamyslieť sa nad tým, či hrozí nedodržanie termínu a aké z toho plynú následky. Pri čiastkových termínoch nemusí ísť o závažné riziko (závisí od dohody s pedagógom), ale nedodržanie termínu záverečného odovzdania práce môže znamenať nezískanie zápočtu.
- Identifikácia rizík – spracovaná v predchádzajúcej fáze manažmentu rizík.
- Odhady nákladov a trvania činností – v rôznych situáciách sú zúčastnené strany ochotné vynaložiť rôzne množstvo prostriedkov na samotný manažment rizík.

Techniky vyčíslenia rizík

Medzi metódy a techniky analýzy rizík patria:

- Vyčíslenie očakávanej peňažnej hodnoty ako súčinu pravdepodobnosti zhmotnenia rizika a očakávanej spôsobenej škody pri zhmotnení rizika. Získané číselné údaje treba správne interpretovať, napr. je ťažké porovnať riziko s nízkou pravdepodobnosťou a vysokou spôsobenou škodou s rizikom s vysokou pravdepodobnosťou, ale nízkou spôsobenou škodou. Číselné vyjadrenie rizík tiež nezohľadňuje ďalšie faktory, napr. výskyt udalostí spôsobujúcich škodu v skupinách. Táto technika sa preto väčšinou používa ako základ pre ďalšiu analýzu (napr. pomocou rozhodovacích stromov).

- Simulácia – využíva model systému za účelom analýzy jeho správania. Simulácia rozvrhu projektových činností sa väčšinou vykonáva s využitím metódy Monte Carlo.
- Rozhodovací strom – zachytáva vzťah medzi prijatými rozhodnutiami a neurčitosťou výsledku. Uzly stromu reprezentujú rozhodnutie (zobrazujú sa štvorčekom) alebo neurčitosť (zobrazujú sa krúžkom). Listami stromu je vyčíslenie očakávanej hodnoty rizika.
- Expertný odhad – možno aplikovať nezávisle od ostatných techník.

Výstup analýzy rizík

Výstupom analýzy rizík je zoznam udalostí, ktoré môžeme ignorovať (akceptácia rizika) a zoznam udalostí, ktorými sa treba ďalej zaoberať.

Plánovanie manažmentu rizík

Plánovanie manažmentu rizík zahŕňa definovanie krokov spojených s konkrétnym rizikom. Reakcie na hrozby spadajú do nasledovných kategórií:

- Vyhnutie – eliminácia určitého rizika odstránením jeho príčin.
- Zmiernenie očakávanej peňažnej hodnoty škody znížením pravdepodobnosti jej výskytu alebo zmenšením hodnoty spôsobených škôd (napr. poistením).
- Akceptácia dôsledkov škody – môže byť aktívna (pri ktorom je vypracovaný plán, ktorý sa vykoná pri výskyte udalosti spôsobujúcej škodu) alebo pasívna (prijatím rizika nižších ziskov pri výskyte danej udalosti).

Konkrétnou technikou manažmentu rizík je napríklad naplánovanie činností, ktoré treba vykonať pri zhmotnení niektorého z identifikovaných rizík, vypracovanie alternatívnych stratégií, poistenie alebo zabezpečenie subdodávky rizikovej časti systému od inej spoločnosti, ktorá má skúsenosti s novou technológiou.

Výstupom plánovania manažmentu rizík môže byť vypracovanie plánu manažmentu rizík, úprava zmluvy so zákazníkom alebo vyčlenenie dostatočných časových a peňažných rezerv pre prípad potreby ošetrenia udalosti spôsobujúcej škodu.

Riadenie rizík

Riadenie rizík je vlastne vykonávaním vypracovaného plánu manažmentu rizík počas vykonávania projektu. Táto etapa manažmentu rizík zahŕňa sledovanie projektu s cieľom včasného rozpoznania rizikovej situácie a reakcie na zmeny v projekte, ktoré si môžu vyžadovať opätovnú identifikáciu, analýzu a plánovanie manažmentu rizík. Riadenie rizík tiež zahŕňa vysporiadanie sa s nastatím situácie spôsobujúcej škodu.

Rámec pre identifikáciu rizík v softvérových projektoch

V osemdesiatych rokoch minulého storočia vypracoval Boehm zoznam 10 „top“ rizík softvérových projektov [2]. Táto štúdia identifikuje hlavne faktory, ktoré môže projektový manažment kontrolovať. Od čias jej publikovania sa pohol softvérový priemysel významne dopredu. Nové zaujímavé výsledky poskytuje štúdia [4] vypracovaná Delphi metódou na základe panelových diskusií skúsených projektových manažérov v troch krajinách: vo Fínsku, v Hongkongu a v Spojených štátoch. Projektoví manažéri, ktorí sa zúčastnili tejto štúdie, odpovedali na otázky, aké faktory považujú za rizikové a ktoré z nich považujú za najdôležitejšie. Štúdia viedla k zaujímavým výsledkom:

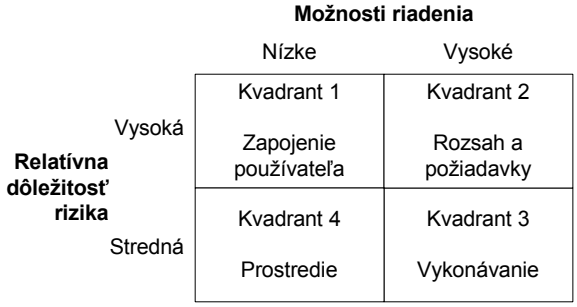
1. Aj napriek rozličným podmienkam vo zvolených krajinách (mentalita ľudí, podnikateľské prostredie, ...) identifikovali respondenti vo všetkých krajinách rovnakú skupinu asi tucta rizík, ktoré považovali za dôležité, aj keď s rôznou úrovňou dôležitosti.
2. Na rozdiel od Boehmovej štúdie [2] je väčšina najzávažnejších identifikovaných faktorov mimo možností priamej kontroly projektového manažéra.

Vypracovaný zoznam rizík získaných na základe štúdie je uvedený v tabuľke 1. Táto tabuľka môže slúžiť ako kontrolný zoznam pri identifikácii rizík projektu.

1.	Nedostatok *zainteresovanosti* vrcholového manažmentu do projektu (podpora od vrcholového manažmentu nestačí)
2.	Neúspech v zainteresovaní zákazníka
3.	Neporozumenie požiadavkám
4.	Nedostatok adekvátnej zaangažovanosti používateľa
5.	Neúspech splniť očakávania koncových používateľov
6.	Zmena cieľov alebo rozsahu projektu
7.	Nedostatok potrebných znalostí/zručností projektového tímu
8.	Nedostatok zmrazených požiadaviek
9.	Zavedenie novej technológie
10.	Nedostatočné alebo nevhodné obsadenie pozícií v tíme
11.	Konflikt medzi organizačnými jednotkami zákazníka

Tab. 1. Zoznam rizík získaných na základe štúdie [4]

Autori štúdie následne vypracovali rámec pre identifikáciu, analýzu a plánovanie manažmentu rizík, ktorý namiesto sústredenia sa na konkrétne riziko ponúka vysokoúrovňový pohľad na skupiny rizík a stratégie na ich odstránenie. Riziká boli zoskupené podľa dvoch charakteristík: možností riadenia projektovým manažérom a relatívnej dôležitosti rizika v porovnaní s ostatnými rizikami. Rámec je zobrazený na obrázku **Obr. 1.** Z hľadiska manažmentu projektu je kritické hlavne zvládnutie rizík patriacich do kvadrantov 1, 2 a 3.



Obr. 1. Rámec pre kategorizáciu rizík (podľa [4])

Z uvedenej štúdie môžeme vychádzať aj v tímovom projekte, kde môžeme identifikovať nasledovné analógie v rizikách:

- Neporozumenie požiadavkám na tímový projekt – riziko spojené s orientovaním úsilia členov tímu nesprávnym smerom, keď sa tím sústredí na vytvorenie dokonalého programu ako hlavnej súčasti softvérového produktu a pritom zanedbá ostatné dôležité súčasti projektu (napr. náležité zdokumentovanie vytvoreného produktu a aj tímovej práce samotnej; slabé prezentovanie výsledkov, ktoré tím dosiahol). Riešením môže byť priebežné vytváranie všetkých potrebných výstupov.
- Neúspech v zainteresovaní zákazníka – zákazníkom je v tomto prípade pedagogický vedúci projektu. Ak tím nezíska od neho dostatočnú spätnú väzbu (napríklad vinou viaznucej vzájomnej komunikácie alebo nepripravenosťou členov tímu na tímové stretnutia), tak pravdepodobne celkom nesplní jeho očakávania.
- Nedostatok potrebných znalostí/zručností tímu – pri určovaní preferencií na výber projektu by sa mali zohľadniť schopnosti jednotlivých členov tímu, aby tím nemusel vynaložiť priveľké úsilie na získanie potrebných znalostí, čím stratí veľa času, ktorý môže byť využitý lepšie.
- Nevhodné obsadenie pozícií v tíme – tím (a hlavne jeho vedúci) by mal zvážiť, kto je vhodný na ktorú pozíciu v tíme, aby napr. dohľad nad dokumentáciou zabezpečoval ten člen tímu, ktorý má najlepšie skúsenosti s dokumentovaním. Tiež je dôležitá voľba vedúceho tímu, pretože táto osoba musí svojou autoritou tím vhodne usmerňovať a mala by mať dostatok na to potrebných skúseností a zručností.

Záver

Zvládnutie manažmentu rizík v softvérovom projekte môže byť kľúčovou otázkou úspechu celého projektu. Dobrý projektový manažér musí s rizikami v projekte rátať,

vedieť ich predvídať a včas odhaliť blížiacu sa rizikovú udalosť. Dôležitú úlohu hrá nielen jeho teoretická pripravenosť, ale aj cit a skúsenosti nadobudnuté praxou.

Použitá literatúra

1. Bieliková, M.: *Softvérové inžinierstvo. Princípy a manažment.* Vydavateľstvo STU, Bratislava 2000.
2. Boehm, B.W.: Software Risk Management: Principles and Practices. *IEEE Software,* Vol. 8, No. 1 (1991), 32-41.
3. Doruľa, J., Kačala, J., et al.: *Elektronický lexikón slovenského jazyka.* Forma, Bratislava 1998.
4. Keil, M., et al.: A Framework for Identifying Software Project Risks. *Communications of the ACM,* Vol. 41, No. 11 (1998), 76-83.
5. Project Management Institute, Standards Committe: *A Guide to the Project Management Body of Knowledge.* Project Management Institute, Newtown Square 1996.
6. Wallace, L., Keil, M.: Software Project Risks and their Effect on Outcome. *Communications of the ACM,* Vol. 47, No. 4 (2004), 68-73.

Annotation

Software Project Risk Management

Software project risk management is an important part of overall software project management. It has to be performed throughout the entire project lifecycle. Its main objective is to identify and treat unexpected risky events in the project, so as they a) never happen, b) cause minor losses. Risk management consists of risk identification, risk analysis, risk response development and risk response control. This essay offers a complex view on risk management phases and overview of the most important factors identified by experienced project managers. It also tries to cope with risk management in software projects like team project in study at FIIT STU.

Autopsia extrémneho programovania

JÁN SUCHAL

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
johno@jsmf.net

Abstrakt. Agilné metódy programovania, kde sa za hlavného predstaviteľa pokladá práve extrémne programovanie, sú považované za mimoriadne vhodné najmä pre menšie projekty s často sa meniacimi požiadavkami. Vzniká tak však aj mylná predstava o tom, že praktiky, ktoré tvoria ich jadro, sú pre iné projekty nevhodné a spôsobujú ich zlyhania. Dostávajú sa takto do úzadia bez hlbšieho skúmania ich pravej podstaty. V poslednej dobe sa však ukazuje, že niektoré praktiky sú natoľko rozumné, že vhodný výber a ich použite môže mať pozitívny dopad na takmer ľubovoľný projekt. Je preto potrebné spoznať príčiny vzniku ako aj výhody a dôsledky týchto praktík a nezavrhovať ich hneď ako celok.

Tradičné miesto agilných metód

Je všeobecne známe, že agilné metódy programovania sú vhodné najmä pre projekty s často sa meniacimi požiadavkami a menšie tímy. To je síce pravda, avšak dôsledkom tohto tvrdenia vzniká aj mylná predstava o tom, že akonáhle je projekt trochu väčší alebo je špecifikácia projektu dopredu dostatočne jasná, treba na agilné metódy rýchlo zabudnúť. Problém vidím v tom, že agilné metódy sa nesprávne chápu ako nejaký postup práce. Ak potom zlyhal postup, tak ho treba zavrhnúť. Väčšina agilných metód je však definovaná dosť vágne, veľmi často len formuláciou akýchsi základných princípov. O nejakom detailne prepracovanom pracovnom procese sa tu hovoriť však určite nedá.

Extrémne programovanie (XP)

Extrémne programovanie ako agilná metóda programovania sa v súčasnosti pokladá za najrozšírenejšiu a najznámejšiu zo všetkých. V očiach mnohých je XP dokonca akýmsi predstaviteľom a základným stelesnením hlavných princípov či praktík ostatných agilných metód.

Niektoré z týchto praktík sú však natoľko kontroverzné a ich pravý význam a dôsledky natoľko nepochopené, že považujem za dôležité ich tu podrobnejšie rozobrať.

Malé verzie

Akonáhle sa do vyvíjaného produktu pridáva nová funkcionalita dôležitá z hľadiska zákazníka, je potrebné hneď vydať novú verziu. Navyše to treba robiť tak často, ako sa len dá.

Zrejmou výhodou tohto prístupu je to, že čím častejšie vývojový tím vydáva nové verzie, tým má lepšiu spätnú väzbu od zákazníka. Dokáže rýchlejšie reagovať na vznikajúce požiadavky a problémy. Čím skôr sa teda zavedie do systému nová funkcionalita, tým viac času má vývojový tím na jej prípadné opravy.

Tento nápad je vo svojej podstate výborný, avšak má aj svoje úskalia. Čo ak zákazník potrebuje určité kritické funkcie a nová verzia ich ešte všetky neobsahuje? Čo ak je potrebné vykonať pred nasadením náročnú migráciu dát? Nebude častá distribúcia novej verzie v nasadzovanom prostredí príliš nákladná?

Ideálne pre tento spôsob nasadzovania sa však zdajú byť centralizované webové aplikácie. Stačí nasadiť novú verziu na centrálny server a všetci v tom okamihu používajú nový, vylepšený produkt.

Jednoduchý návrh, emergencia architektúry

V XP je systém navrhovaný čo najjednoduchšie, pričom hlavným a jediným cieľom je to, aby fungoval. Akékoľvek komplikácie sú odstraňované hneď ako sa spozorujú a do systému sa zo zásady pridáva nová funkcionalita až vtedy, keď je naozaj potrebná. Tento princíp je známy pod skratkou YAGNI (You *Arent Gonna Need It)* alebo KISS (Keep *It Simple, Stupid)*.

Zdá sa, že YAGNI je v priamom rozpore so zaužívaným spôsobom vývoja softvéru – špecifikácia, analýza, návrh, implementácia. Ak však mne ako programátorovi vznikne potreba použiť nejakú funkcionalitu, ktorú ešte nemám implementovanú, tak vlastne vykonávam akúsi skrytú špecifikáciu. Výhoda je v tom, že presne viem, akú funkcionalitu potrebujem. V klasickom vodopádovom modeli sa môže aj ten najlepší analytik snažiť ako len vie, ale takú jasnú predstavu ako ja bude mať len zriedkavo. Ak bude príliš špecifikovať, tak len pridá programátorom zbytočnú robotu, ale ak bude príliš zjednodušovať, tak bude funkcionalita nakoniec znova chýbať.

Rozdiel oproti klasickému modelu je v tom, že jednotlivé fázy sa nevykonávajú iba raz, ale mnohokrát – v iteráciách. XP predpokladá, že výsledkom veľkého počtu takýchto iterácií začne postupne vznikať (emergovať) aj celková architektúra systému.

To, čomu sa ale snaží XP pomocou YAGNI princípu vyhnúť, je zbytočne veľký návrh (Big *Design Up Front)*. Predpokladá totiž, že funkcionalitu sa nikdy nepodarí odhadnúť úplne presne a navyše sa môžu zmeniť aj požiadavky zákazníka. Čo je však potrebné zdôrazniť, je fakt, že XP úplne nezavrhuje návrh ako taký. Akýsi hrubý návrh sa vytvára pri prvotnom plánovaní funkcionalít obsiahnutých v jednotlivých verziách produktu. Navyše, ak ide o väčšie projekty, je dokonca nevyhnutné vykonať akúsi dekompozíciu systému na menšie celky alebo vrstvy, ktoré už majú jasnejšie definovanú funkcionalitu. [7]

Ak je teda možné nejakú funkcionalita produktu dopredu dostatočne špecifikovať, nie je dodržovanie YAGNI princípu až také nevyhnutné, avšak XP ho odporúča prakticky všade.

Testovanie

V XP sa kladie mimoriadny dôraz na testovanie, pričom existujú dva typy testov:

- Testy jednotiek, ktoré slúžia na testovanie jednotlivých častí kódu, väčšinou na úrovni tried.
- Akceptačné testy, ktoré vychádzajú z takzvaných užívateľských scenárov, pričom testujú očakávanú funkcionalitu produktu ako celku.

Oba typy testov sú však plne automatizované, úplne pokrývajú potrebnú funkcionalitu a je možné ich kedykoľvek spustiť. Ak sú k dispozícii takéto testy, tak otestovanie kompletnej funkčnosti systému trvá väčšinou len zopár minút, na otestovanie triedy stačí maximálne niekoľko sekúnd. Dôsledky tohto prístupu sú však obrovské. Nikto z programátorov sa teda nemusí báť zmeniť akýkoľvek kód. Testy mu slúžia ako akási záchranná sieť a pokiaľ sú naozaj kvalitné, tak nielen chybu odhalia, ale ju dokážu aj veľmi presne lokalizovať. Programátori striktne využívajúci takéto automatizované testovanie dokonca tvrdia, že dôsledkom toho sa z ich programátorského života úplne vytratila fáza hľadania chýb (debugging).

Táto praktika sa v praxi natoľko osvedčila, že vznikla samostatná metóda programovania známa ako testmi riadený vývoj (test *driven development, TDD)*. Je založená na tom, že ešte pred tým, ako sa napíše akákoľvek nová funkcionalita, musí sa napísať automatický test, ktorý túto funkcionalitu overí. Až potom sa môže pristúpiť k samotnej implementácii funkcionality. Akonáhle prebehnú všetky testy úspešne, vykoná sa refaktoring kódu a pokračuje sa vytvorením ďalšieho testu. Tento postup je známy ako *red/green/refactor* a nazýva sa *TDD mantra.*

Táto praktika dokonca nepriamo podporuje aj spomínaný YAGNI princíp, pretože ak by chcel programátor pridať nejakú zbytočnú funkčnosť, musel by pre ňu najprv napísať test a to programátor ako tvor lenivý dobrovoľne robiť nikdy nebude.

Ukazuje sa, že kód vytváraný takýmto prístupom je aj oveľa kvalitnejší a prehľadnejší. Testy dokonca môžu slúžiť ako nepretržite aktuálna dokumentácia funkčnosti, pretože sú väčšinou tvorené ako jednoduché príklady použitia danej časti systému či produktu ako takého. Dôležité je zdôrazniť aj to, že testy samotné musia mať minimálne takú kvalitu ako kód, ktorý testujú.

Ani táto metóda sa však nedá použiť úplne všade. Automatizácia testovania používateľských rozhraní je vo všeobecnosti príliš komplikovaná, aj keď pri webových aplikáciách už sú dostupné vcelku robustné testovacie riešenia. Taktiež testovanie distribuovaných systémov týmto spôsobom je ešte len v plienkach.

Táto metóda sa však dá použiť vždy bez ohľadu na to, o aký rozsah projektu ide.

Párové programovanie

XP sa preslávilo práve svojou koncepciou párového programovania. Základom je to, že zdrojový kód tvoria dvaja ľudia naraz. Jeden monitor, jedna klávesnica, jeden počítač

a dvaja ľudia. Človek, ktorý práve sedí pri klávesnici a píše kód, premýšľa o tom, ako najlepšie implementovať konkrétnu metódu. Ten druhý, čo mu hľadí cez rameno, rozmýšľa globálnejšie. Rozmýšľa o tom, či by sa systém nedal nejako zjednodušiť, či sa neporuší nejaká iná funkcionalita systému. Ak vidí, že partnerovi niečo nejde tak, ako by si predstavoval, bez váhania zoberie klávesnicu a roly sa vymenia.

Výsledky dosiahnuté touto na prvý pohľad značne neefektívnou praktikou sú viac ako pozoruhodné. Štúdie na univerzitách a skúsenosti z praxe ukazujú, že kvalita kódu tvorená práve v pároch je nielen oveľa vyššia, ale čo je takmer nepochopiteľné je fakt, že dvaja ľudia v páre urobia tú istú prácu asi o 20% rýchlejšie ako dvaja ľudia samostatne. [7]

Ďalšou nezanedbateľnou výhodou tohto prístupu je, že programátori pracujúci v pároch, ktorí väčšinou majú rôzne úrovne znalostí či skúseností, sa tak neustále vzdelávajú. Jeden radí tomu druhému, je mu oporou. Na druhej strane sa zvyšuje sebaistota každého z nich, lebo vedia, že ten druhý dáva pozor a nedovolí mu spraviť nejakú hlúposť.

Projektoví manažéri si chvália tento prístup aj z iného dôvodu. Tvrdia, že ak pridajú do tímu nových ľudí, tak čas, ktorý je potrebný na to, aby začali byť produktívni, sa výrazne skracuje. Rádovo z niekoľkých mesiacov na týždne. [7]

Na niekoľkých univerzitách boli dokonca pokusy vyučovať programovanie pomocou XP, ukázalo sa však, že táto praktika predpokladá určité zručnosti v programovaní, a tak výučba programovania ako takého nie je úplne vhodná. Ak však študenti už majú nejaké skúsenosti s návrhovými vzormi a refaktoringom, môžu sa týmto štýlom veľmi efektívne zdokonaľovať napríklad v ich použití.

Ďalšie pokusy hovoria o tom, že párové programovanie je dobré len na riešenie určitých problémov. Tvrdia, že ak ide o zložitý problém s nejasným riešením, je dobré programátorov párovať, naopak pri tvorbe napríklad používateľských rozhraní je to zbytočné plytvanie ľudskými zdrojmi. [8]

Ukazuje sa taktiež, že mentalita programátorov je v súčasnosti natoľko zdeformovaná sólovým štýlom programovania, že adopcia párového programovania väčšinou trvá nejaký ten čas. Netreba sa však na začiatku zľaknúť neefektivity. Väčšina programátorov totiž neskôr hodnotí párové programovanie ako veľmi rozumný a hlavne omnoho zábavnejší spôsob práce.

Spoločné vlastníctvo kódu

XP vraví, že zdrojový kód môže meniť ktokoľvek a kedykoľvek chce. Zodpovednosť za produkt nesie celý tím. Je úplne zrejmé, že bez predpokladu dobrého testovania by sa toto mohlo veľmi rýchlo skončiť katastrofou. To, čomu sa ale týmto XP snaží vyhnúť, je, že danému modulu či triede bude rozumieť len jeden človek. Nemalú váhu v tom nesie práve párové programovanie. Keďže partneri sa v párovom programovaní úplne bežne striedajú, je jasné, že každý programátor sa vo svojej práci stretne s množstvom kódu, ktorý nepísal, ale je nútený ho dobre pochopiť.

Nemôže sa teda stať, že ak odíde či ochorie nejaký programátor, tak po sebe zanechá kód, ktorému nikto nerozumie. Na druhej strane niektoré agilné metódy ako SCRUM zavádzajú presný opak. Tvrdia, že ak je za danú triedu či modul zodpovedný

jeden človek, vytvorí sa medzi ním a zdrojovým kódom akýsi citový vzťah, ktorý ho núti sa o kód starať a neustále ho zlepšovať. Zástanci XP argumentujú tým, že je to veľmi neefektívne, lebo ak je potrebné pridať novú funkcionalitu, tak musia vyhľadať vlastníka triedy a vysvetliť mu, o čo vlastne ide. Stratia pri tom oveľa viac času, ako keby si to sami implementovali.

Zákazník na pracovisku

V XP sa súčasťou tímu počas vývoja stáva i skutočný užívateľ alebo zákazník. Nemyslí sa tým však komunikácia cez e-mail či telefón. Je to človek, ktorý je na plný úväzok „zamestnaný“ vo vývojovom tíme, odpovedá na otázky, pomáha vytvárať akceptačné testy a určuje priority. XP pripúšťa, že takéhoto človeka si nemôže dovoliť uvoľniť každá firma len tak, avšak argumentuje tým, že ak vyvíjaný systém nemá pre firmu ani cenu práce, ktorú by človek tam odpracoval, tak je najlepšie systém ani nezačať vyvíjať.

Kritici XP tvrdia, že firmy budú posielať na takéto miesto neskúsených pracovníkov, aby ušetrili peniaze. Neskúsení pracovníci však nemajú jasnú predstavu o potrebnej funkcionalite produktu a ten teda v konečnom dôsledku nemôže byť kvalitný. Praktické skúsenosti s XP však ukazujú, že tomu tak nie je, pretože zákazníci tento problém veľmi dobre chápu a na takéto miesta posielajú skúsených a kvalitných pracovníkov.

Štandardy písania zdrojového kódu

V XP sa kladie najväčší dôraz na samotný zdrojový kód. Pri vývoji totiž nevzniká prakticky žiadna dokumentácia či špecifikácia. Všetky požiadavky sú viac-menej priamo zapísané kódom do akceptačných testov, väčšina komunikácie prebieha nad zdrojovým kódom, a preto je pre XP mimoriadne dôležité udržiavať ich v dobre čitateľnej, štruktúrovanej a pre všetkých zrozumiteľnej forme. Pri metódach, kde vznikajú dokumentácie, to nie je také kritické, ale vo všeobecnosti sa to veľmi odporúča.

Adopcia agilných praktík

Praktiky, na ktorých je celé XP založené, sú navzájom silno previazané. Kvalitný kód nemôže existovať bez dobrých testov, dobrá architektúra systému nemôže emergovať bez použitia refaktoringu, produkt nemôže byť kvalitný bez spätnej väzby od zákazníka. Zdá sa, že každá praktika v XP má svoje jedinečné a nezastupiteľné miesto.

Problém je v tom, že adoptovať všetky praktiky XP naraz sa pre vývojový tím môže zdať ako veľký skok do neznáma. Je to určite tak a výsledky takto začínajúceho tímu budú asi ďaleko od tých očakávaných. Ak by sa však vynechala alebo zanedbala nejaká kľúčová praktika, tak sa zase požadovaný výsledok nemusí dostaviť vôbec. Tento nedostatok pripúšťajú aj samotní tvorcovia XP, pričom tvrdia, že pre dobre disciplinované tímy nebýva táto fáza až taká kritická.

Ak potom časom tím zistí, že nejaká praktika XP pre ich konkrétne projekty nefunguje, netreba sa báť vyskúšať ju vynechať. Ak sa to osvedčí v praxi, nie je dôvod sa k nej vracať. Treba to však skúšať aj naopak, nie je dôvod zavrhnúť automatické testovanie len preto, že sa používa v agilných metodikách, ktoré ako celok pre projekt možno nie sú vhodné. Zástancovia XP navyše dobre vedia, že reálny život je plný zmien a hovoria, že každý tím by si mal svoju metódu programovania vlastne ušiť na mieru. Toto je práve cesta k uplatneniu XP a agilných metodík v ľubovoľnom projekte.

Použitá literatúra

7. Lan Cao, Kannan Mohan, Peng Xu: How Extreme does Extreme Programming Have to be? Adapting XP Practices to Large-scale Projects. In: *Proceedings of the 37th Hawaii International Conference on System Sciences.*

8. M. Muller, W.Tichy: Case Study: Extreme Programming in a University Environment.

Annotation

Autopsy of extreme programming

Agile methodologies represented mainly by extreme programming are considered as very useful techniques in development of small projects in changing environments. This fact has developed into a misunderstanding that core practices of these methodologies are not appropriate for other projects and are mistakenly considered as sources of project failures. They are being rejected without even knowing their true meaning. Nowadays is being shown that some of these practices are so rational, that they can have some positive effect almost on any project. So it is essential to identify their true meaning and consequences they are causing to be able to use them and not to reject them completely.

Manažment komunikácie pre lokalizované a distribuované softvérové tímy

Virtuálne softvérové spoločnosti

VLADIMÍR KRIVUŠ

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

Abstrakt. Táto esej sa zaoberá predstavením pojmu Virtuálna softvérová spoločnosť a skladá sa z dvoch častí: v prvej sa zaoberám hlavne definíciou pojmu Virtuálna softvérová spoločnosť (Virtuálny distribuovaný tím) a opisom nástrojov, s ktorými VSC pracujú. Pri týchto nástrojoch sa sústreďujem na komunikáciu a nie na zdieľanie dokumentov. V druhej časti sa na základe výsledkov reálnych výskumov v tejto oblasti zaoberám problémami, s ktorými sa VSC stretávajú, aké výhody poskytujú, ale aj ich porovnaním s lokálnymi tímami.

Úvod

Koniec 20. a začiatok 21. storočia sú nepochybne obdobím rozvoja a rozkvetu informačných technológií. Preto snáď nikoho neprekvapí, že priaznivá geopolitická situácia, ktorá podporila globalizáciu a vznik nadnárodných koncernov, zanechala vplyv aj na tomto odvetví.

Tento vplyv je možné vidieť nielen na tom, že jednou zo základných vecí nutných riešiť pri vývoji nového softvéru je podpora lokalizácie, ale aj na vzniku takzvaných Virtuálnych softvérových spoločností (VSC – Virtual Software Corporation).

Vzniku týchto VSC napomohol nielen veľký dopyt po nových softvérových produktoch, pre ktorých implementáciu bolo (a je) nutné získať odborníkov, ktorí sa len zriedka vyskytujú na jednom mieste v dostatočnej koncentrácii, ale hlavne rozvojom Internetu a telekomunikácií ako celku. To umožnilo spojiť kvality rôznych ľudí, ktorých od seba často delia tisíce kilometrov, do spolupráce na jednom projekte.

V tejto eseji sa chcem zaoberať hlavne opisom toho, s akými nástrojmi VSC pracujú, s akými problémami sa stretávajú, aké výhody poskytujú, ale aj ich porovnaním s lokálnymi tímami.

Čo je to VSC

Pri definícii tohto pojmu si pomôžem citátom z [9]: VSC je tímové prostredie, v ktorom všetci jeho členovia pracujú spoločne na dosiahnutí spoločných záujmov alebo cieľov. Typicky sú takéto tímy rozmiestnené na rôznych miestach a dokonca v rôznych krajinách a väzby medzi nimi sú dočasné a sporadické – viazané na špecifické projekty alebo úlohy.

Ako synonymum pojmu VSC je možné chápať tiež pojem Virtuálny distribuovaný softvérový tím.

To, že na projekte môžu pracovať ľudia z rôznych časových pásiem, môže dokonca viesť k tomu, že v čase, keď jedna skupina končí svoju prácu (pracovnú dobu) na projekte, ďalšia začína. Tak sa vlastne na projekte pracuje 24 hodín denne, čo značne urýchľuje jeho vývoj. (Pod prácou na projekte je možné chápať aj testovanie, vytváranie dokumentácie, či iné.)

Manažment VSC

Ako predchádzajúca kapitola napovedá, na jednom projekte môže naraz pracovať dosť nesúrodá skupina ľudí. Už samotná kooperácia ľudí v lokálnych tímoch môže byť veľmi ťažká, pretože je nutné riešiť medziľudské vzťahy, nedôveru či neochotu, antipatie, či až prílišnú náklonnosť. Toto je však takmer vždy možné vyriešiť vzájomným vyjasnením stanovísk a zvolením toho správneho kompromisu.

Práve o túto možnosť jednoduchého spôsobu komunikácie sú členovia VSC (zvlášť tých distribuovaných) ukrátení. Tu vystupuje do popredia manažment, od ktorého kvality môže záležať zlyhanie samotného projektu. Je vhodné, aby každú takúto lokálnu podskupinu (ak existujú a nejde o jednotlivcov) riadil jeden reprezentant, ktorý sa bude starať o akúsi hlavnú komunikáciu a šírenie správ. Tieto správy totiž môžu byť aj všeobecného charakteru a nemusia si nutne vyžadovať osobitnú pozornosť každého člena tímu – zahltenie informáciami je vždy kontraproduktívne.

Každý takýto lokálny manažér by mal v prvom rade mať prehľad o tom, čo sa v jeho podskupine deje, kto na čom pracuje, s akými problémami sa stretáva a ako sa plní plán projektu. V tom by mu mali pomáhať lokálne komunikačné nástroje (viď. ďalšia kapitola). Toto potom tvorí základ pre synchronizáciu tímu ako celku, kedy medzi sebou pravidelne a v dohodnutej forme komunikujú títo manažéri – komunikácia medzi podskupinami. V prípade, že je nutné, aby sa vytvoril nový komunikačný kanál medzi podskupinami, malo by to ísť *cez* manažéra (minimálne jeho vytvorenie). Ten rozhodne, kto zo skupiny je vhodný na spracovanie požiadaviek druhej strany, čo mu umožňujú jeho znalosti podskupiny.

Komunikačné nástroje VSC

Hlavným nástrojom sú produkty telekomunikačného odvetvia, a to hlavne Internet. To je spôsobené tým, že VSC, ako každý tím, musí vo väčšej či menšej miere komunikovať. Analýzou niekoľkých produktov (napríklad [10] produkt ProjectCollaboration), ktoré sú zamerané na podporu VSC a jeho manažmentu, ako aj výsledkov výskumných projektov, ktoré skúmali prácu virtuálnych distribuovaných tímov z psychologického hľadiska [11] či z hľadiska efektivity práce [12], je možné zistiť, že k tým najpoužívanejším patria:

- e-mail,
- interaktívna komunikácia (ICQ, IRC, ...),
- telefón,
- zdieľaná *plocha* (ide o možnosť simultánne písať a kresliť na plochu viacerými používateľmi) – writeboard, fórum, ...,
- hlasová komunikácia cez Internet,
- obrazová (video) komunikácia cez Internet.

Spôsoby komunikácie v tíme

Ako bolo spomenuté v kapitole Manažment VSC, komunikácia musí prebiehať v podskupine ako aj v tíme ako celku. Pre komunikáciu v lokálnej podskupine je najvhodnejšia priama schôdzka všetkých zúčastnených, ktorá je vedená lokálnym manažérom. Tu sa má možnosť každý člen tímu vyjadriť k tomu, na čom pracoval, čo a ako chce robiť ďalej, prípadne aké problémy nastali a aké budú mať tieto problémy vplyv na projekt (napr. plán) – pritom zaujme stanovisko aj manažér.

Netreba však zabudnúť, že aj táto podskupina je súčasťou tímu, a teda treba informovať o problémoch, návrhoch a zmenách aj zvyšok tímu. Na toto je vhodné použiť e-mail či zdieľanú plochu. Pre čo najskoršie upozornenie na problémy je možné použiť aj telefón. Nesmie sa však zabudnúť na vytvorenie zápisu o takejto udalosti – vždy je dobré mať dôležité informácie na papieri. Zdieľanie týchto dokumentov je už záležitosťou nástrojov a manažmentu zdieľania dokumentov, ktorou sa v tejto eseji nezaoberám. Spomeniem iba jeden produkt, s ktorým mám osobnú skúsenosť: MS Visual SourceSafe.

Výsledky reálneho výskumu

Aby som sa dostal z teoretickej hladiny a mojich osobných názorov, zhrniem tu výsledky dvoch skôr spomenutých pokusov, z ktorých každý sa zaoberal iným pohľadom na prácu virtuálnych distribuovaných tímov, a to hlavne pre otázky, ktoré

ich výsledky načrtávajú. Oba tieto výskumy prebiehali na študentoch univerzít, ktoré sa nachádzali vždy v dvoch rôznych krajinách.

New methods for Studying Global Virtual Teams

Tento výskum sa snažil čo najviac sa priblížiť skutočným podmienkam, ktoré môžu nastať, a tak boli aj projekty zadané študentom skutočnými firmami. Výskum prebiehal na 10 skupinách súčasne. Zapojené boli univerzity z USA, Ruska, Číny ako aj z Holandska. Na vypracovanie mali študenti jeden semester (12 týždňov). Študentom bol poskytnutý nástroj, ktorý umožňoval interaktívnu (obdoba ICQ), pasívnu (e-mail), hlasovú (prenos hlasu po Internete), ako aj obrazovú (prenos videa po Internete) komunikáciu – záležalo iba na ich preferenciách. Počas priebehu projektu bola monitorovaná ich vzájomná komunikácia. Na záver mali za úlohu vyplniť dotazník, v ktorom sa mali vyjadriť k vzájomnej dôvere a miere pohodlia pri práci v podskupine, ako aj k druhej časti tímu.

Najlepšie vzájomné ohodnotenie dôvery si vyjadril ten tím, ktorý na komunikáciu používal prevažne formu videokonferencie. Toto mu umožnilo oveľa lepšie sa navzájom spoznať, a tak sa študenti priblížili ku charakteru lokálnej skupiny. To iba dokazuje, aká je komunikácia v skupine dôležitá. Takáto *spoločenská* skupina však vznikla iba jedna. Ostatné tímy po počiatočnom odskúšaní všetkých ponúknutých foriem komunikácie zvolili buď ICQ alebo e-mail. Toto, ako aj skutočnosť, že vzájomné hodnotenia študentov *cez oceán* boli horšie ako hodnotenie študentov v rámci kontinentu, by mohlo viesť k názoru, že vytváranie takýchto distribuovaných tímov je nevhodné. Alebo tu išlo o kultúrne rozdiely? Pred kontaktom s druhou podskupinou v rámci tímu sa jej členovia najskôr poradili a potom vybrali toho, ktorý mal o výsledku informovať druhú stranu. Chceli týmto šetriť čas a vyhnúť sa duplicitnému zdieľaniu informácií alebo sa snažili mať čo najmenší kontakt s druhou skupinou?

Najväčšia nedôvera vznikla medzi študentmi Holandska a USA. Holanďania boli vedomostne nadradení a to vyvolalo nedôveru k skupine z USA. Keď skupina z USA nezdieľala informácie v čase, v ktorom to robila holandská skupina, táto (bez toho, aby to konzultovala s druhou stranou) prešla do určitej formy vzbury a dokumenty s druhou skupinou tiež začala zdieľať iba chvíľku pred vzájomným pravidelným stretnutím. Americká strana si pritom vzniknutého napätia počas celého priebehu výskumu nebola vedomá. Toto poukazuje na ďalšiu nevýhodu, ktorú prináša neexistencia osobného kontaktu, a to prenášanie svojich postojov a názorov na *človeka* na druhej strane *kábla,* ktoré sa nie vždy zhodujú so skutočnosťou.

Paradoxne vznikla situácia, že študenti, ktorí mali doma rýchly Internet, uprednostňovali prácu z pohodlia domova. Bol toto spôsob, ako odbúrať stres vzniknutý komunikovaním s cudzími ľuďmi? Bola istota domova záplatou na tento problém?

I keď tento výskum priniesol veľké množstvo výsledkov, treba priznať, že prebehol na dosť malej vzorke a nebol tu zahrnutý tím, ktorý by tvorili aj dve podskupiny z tej istej krajiny, čo by mohlo napomôcť k zodpovedaniu otázky o zvýšenej neochote spolupracovať s podskupinou inej národnosti.

Team Performance Factors in Distributed Collaborative Software

Pri druhom výskume sa stretli opäť študenti, tentoraz z USA a Švédska. Rovnako sa stretli s projektom, ktorý trval jeden semester. Tentoraz im nebol priamo poskytnutý nástroj, ktorý by mali používať pre komunikáciu – výber bol ponechaný na nich samotných. Na rozdiel od predchádzajúceho výskumu, komunikácia medzi skupinami tímu bola povinná a vzťahovala sa na jednotlivé podúlohy, ktoré museli byť prerokované. Stretnutie viedol vždy jeden človek, ktorého výber podliehal pravidlám striedania.

Za najzaujímavejší výsledok tohto pokusu považujem skutočnosť, že si skupiny opäť zvolili skôr možnosť použiť *anonymné* IRC, e-mail či webové fórum pred možnosťou hlasovej či obrazovej komunikácie.

Tiež sa zistili väzby medzi množstvom komunikácie v tíme a výsledným hodnotením projektu.

Tento výskum podľa mňa nepatril k tým kvalitnejším – autori dospeli k záverom rýchlo a neanalyzovali získané údaje zo všetkých strán (málo kategórií).

Záver

Dúfam, že cieľ tohto dokumentu, ktorým je predstaviť pojem Virtuálna softvérová spoločnosť, bol splnený.

Ako z praktických výsledkov vyplýva, VSC je vhodné voliť iba v krajnom prípade – po vyčerpaní iných možností, pretože napriek postupujúcej globalizácii sú kultúrne a spoločenské rozdiely v komunikácii stále citeľné. Pre VSC je situácia obdobná, i keď je možné vziať do úvahy typ ľudí, ktorí softvér vyvíjajú – introverti – aj pri takomto type projektu je komunikácia potrebná.

Použitá literatúra

9. Technical Research Centre of Finland, VTT Publications 409: Towards virtual software configuration management. http://www.vtt.fi/inf/pdf/publications/2000/p409.pdf, 12.12.2005
10. JDH Technologies: Web-4M for Team/Project Collaboration. http://www.jdhtech.com/pages/whitepapers/ProjectCollaboration.pdf, 12.12.2005
11. IEEE, Published in the Proceedings of the Hawaii's International Conference On System Sciences, January 3-6, 2001: New Methods for Studying Global Virtual Teams: Towards a Multi-Faceted Approach. http://cscw.msu.edu/reports/HICSSweb.pdf, 12.12.2005
12. Hause & Woodroffe, 2. PPIG 2001, Bournemouth UK, University (GVSU) in Michigan, USA: Team Performance Factors in Distributed Collaborative Software Development. http://www.ppig.org/papers/13th-hause.pdf, 12.12.2005

Annotation

Management of Communication for Local and Distributed Software Teams – Virtual Software Companies

This essay deals with introduction of term Virtual Software Company and has two parts: in the first part is defined term Virtual Software Company (Virtual Distributed Team) and are described the tools which are VSCs using. While describing these tools, I focus on those which are used to communicate, not those which are used to share the documents. Context of the second part is composed problems with which are VSCs facing and which are based on results of real researches in this area.

Manažment kvality a vplyv na výsledok projektu

PETER LEDŇA

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

Abstrakt. Kvalita je bezpochyby najdôležitejšou vlastnosťou každého výrobku. Základnou myšlienkou tejto eseje je pozrieť sa hlbšie na pojem kvalita softvéru, a to v kontexte manažmentu kvality. Príspevok podrobne analyzuje jednotlivé znaky kvality. Diskutuje o tom, ktoré z nich sú dôležité z pohľadu zákazníka a z pohľadu výrobcu softvéru. Rozoberie sa tu pojem chyba a taktiež jednotlivé rozpory medzi požiadavkami zákazníka a požiadavkami výrobcu na softvér. Na záver sa opisuje manažment kvality ako systém riadenia kvality.

Úvod

O pojme kvalita výrobku sa hovorí, že je to najstaršia ekonomická kategória. Kvalita nie je len vlastnosť výrobku, ale aj ľudí, ktorí tento výrobok vytvárajú a v praxi aj správne používajú. Z hľadiska pojmu „kvalita softvéru“ sa musíme zamyslieť, čo pojem znamená z pohľadu zákazníka a čo z pohľadu vývojára. Je zrejmé, že tieto dva pohľady sa líšia. Kým zákazník požaduje bezchybný softvér za čo najnižšiu cenu, spoločnosti vytvárajúce softvér musia hľadať rozumnú mieru medzi tým, čo požaduje zákazník a svojimi vlastnými potrebami.

Manažment kvality sa používa vo svete systému riadenia kvality v rôznych oblastiach priemyslu, ale i v tvorbe riadenia kvality softvéru. Manažment kvality zvyšuje kvalitu softvéru, a to takým spôsobom, že neustále vylepšuje proces jeho výroby.

Čo je to kvalita?

Pod pojmom kvalita môžeme rozumieť celkový súhrn vlastností a charakteristík produktu (výrobku, procesu alebo služby), ktoré ovplyvňujú schopnosť uspokojovať stanovené alebo predpokladané požiadavky zákazníka [14]. Kvalita je schopnosť inherentných vlastností výrobku plniť požadované vlastnosti. Ak sa chceme však zamyslieť nad pojmom „požadované vlastnosti“, uvedomíme si, že je to pojem veľmi

obšírny. Na hlbšie pochopenie si zavedieme iné definície pojmu kvalita, ktoré nám objasnia jeho význam z iných uhlov pohľadu. Podľa [14] si zavedieme nasledujúce definície kvality:

- Transcendentné definície kvality výrobku: kvalita výrobku je nedefinovateľná, každý užívateľ má svoj vlastný názor, ktorý je ovplyvnený vlastným individuálnym chápaním.
- Výrobkovo orientované definície kvality: kvalita výrobku závisí od skutočnosti, či má výrobok také vlastnosti, aké mu predpisuje technická dokumentácia.
- Výrobne orientované definície kvality: výrobok je kvalitný vtedy, ak sú pri jeho výrobe dodržané výrobné a technologické postupy uvedené v technickej dokumentácii.
- Hodnotovo orientované definície výrobku: pod kvalitným výrobkom budeme rozumieť výrobok, vyrobený za prijateľnú cenu pre zákazníka a pri výrobných nákladoch prijateľných pre výrobcu.
- Užívateľsky orientované definície kvality: výrobok je kvalitný vtedy, ak má všetky vlastnosti zhodné s požiadavkami zákazníka a požiadavky zákazníka musia byť implementované do návrhu výrobku a technickej dokumentácie.

Ako je vidieť, tak pojem kvalita je komplexný a nemožno ho zhrnúť do jednej myšlienky. Jednotlivé definície kvality nie sú rovnaké a niekedy si aj navzájom odporujú. Požiadavky zákazníka sú často nereálne, pretože sa riadia heslom „chcem všetko za čo možno najnižšiu cenu“. A naopak výrobca (softvéru) chce vytvoriť čo najmenej za čo možno najväčší zisk. Aby sme pochopili hlbšiu podstatu problému, rozdeľme si vlastnosti softvérového produktu na:

- Užitočné vlastnosti: sú také vlastnosti softvéru, ktoré sú funkčne dôležité pre jeho používanie.
- Neužitočné vlastnosti: sú také, ktoré sú pre používanie daného softvéru nežiaduce alebo škodlivé. Sem patria aj chyby (vady), ktorými sa budeme zaoberať neskôr.

Užitočné vlastnosti softvéru delíme do jednotlivých tried, ktoré sa nazývajú znaky kvality. Vzhľadom na špecifickú problematiku pri tvorbe softvérových produktov sú tieto znaky kvality odlišné od iných druhov tovarov. Hlavné delenie je nasledovné [16]:

- Externé: sú dôležité najmä pre zákazníka a priamo ovplyvňujú cenu softvéru na trhu.
- Interné: sú dôležité pre výrobcu softvéru a majú priamy dopad na rozvoj schopností vývojového tímu splniť projektové externé požiadavky kvality, vývoj výdavkových plánov a rozvrhu.

Externé znaky kvality softvéru

Niektoré z týchto znakov sú si podobné, ale všetky majú úplne iný zmysel. Externé znaky kvality softvéru sú [16]:

- Výdavky: peniaze, ktoré použil zákazník, aby vyvinul, kúpil, použil a udržiaval softvérový systém. Často sa tiež nazývajú celkové výdavky na vlastníctvo.
- Dodržanie termínu: miera, do akej bol dodržaný konečný plán dodania alebo miera, na základe ktorej je softvérový systém dodaný v rámci vyžadovaného časového rámca zákazníkovi.
- Úplnosť: miera, na základe ktorej systém implementuje svoj plánovaný rozsah s dôrazom na špecifické požiadavky a dodacie podmienky.
- Správnosť: miera, do akej výrobok spĺňa špecifikáciu.
- Spoľahlivosť: správanie sa výrobku pri výpadku. Výrobok by nemal pri výpadku systému spôsobiť ani fyzické ani ekonomické škody.
- Robustnosť: schopnosť, s akou môže softvér pokračovať v prípade neplatného vstupu alebo neočakávaného problému.
- Použiteľnosť: úsilie, ktoré treba vynaložiť na to, aby sa dal výrobok používať.
- Efektívnosť: splnenie kritérií na využitie zdrojov počítačového systému, na čas potrebný na realizáciu a ďalších kritérií spojených so samotným vývojom výrobku.
- Výkonnosť: rýchlosť alebo priechodnosť, minimalizovanie času medzi udalosťami vstupu a výstupu systémom, maximalizovanie množstva potrebnej vykonanej práce v danom časovom období.
- Kapacita: schopnosť alebo vhodnosť pre podržanie alebo ukladanie dát alebo informácií. Kapacita môže byť obmedzená návrhom alebo požiadavkami a taktiež operačným systémom.
- Merateľnosť: miera, do akej nie je kapacita, efektivita a výkonnosť softvérového systému limitovaná svojim dizajnom, implementáciou a hardvérom, na ktorom sa vykonáva.
- Kompatibilita: miera, do akej bude softvérový systém fungovať a komunikovať správne, spoľahlivo a robustne s inými podobnými systémami, s ktorými zdieľa rovnaké typy dát, súborové systémy a užívateľské rozhrania. Spiatočná kompatibilita sa týka najmä schopnosti softvérových systémov pracovať s predchádzajúcimi verziami, z ktorých bola odvodená.
- Interoperabilita: úsilie, ktoré treba na zabezpečenie spolupráce systému s inými systémami.

- Integrita/bezpečnosť: úsilie, ktoré treba vynaložiť na to, aby bol systém bezpečný.
- Prispôsobivosť: miera, do akej sa systém môže použiť bez modifikácie v iných aplikáciách alebo prostrediach, pre ktoré bol vytvorený.
- Konfigurovateľnosť: miera, do akej môže užívateľ zmeniť operačné parametre systému.
- Presnosť: miera, do akej je vybudovaný systém bezchybný, hlavne čo sa týka kvantitatívnych výstupov. Presnosť sa odlišuje od správnosti; kým presnosť je určenie ako dobre systém pracuje na úlohe, pre ktorú bol navrhnutý, správnosť vyjadruje, či bol systém správne implementovaný.
- Opakovateľnosť: miera, do akej bude systém opakovane produkovať tie isté výstupy, za predpokladu zhodnej množiny vstupov a zhodného operačného systému.
- Budovateľnosť: ľahkosť, s ktorou môže byť softvérový produkt spoľahlivo vybudovaný zo samostatných komponentov.

Interné znaky kvality softvéru

Interné znaky kvality softvéru sú [16]:

- Udržovateľnosť: úsilie, ktoré treba vynaložiť na ďalší vývoj a údržbu výrobku podľa meniacich sa potrieb zákazníka a aj meniaceho sa okolia.
- Pružnosť: úsilie, ktoré treba na modifikáciu výrobku v prevádzke.
- Prenosnosť: úsilie, ktoré treba na prenos výrobku z jednej platformy na inú.
- Znovupoužiteľnosť: miera, do akej možno jednotlivé časti výrobku znovu použiť pri iných podobných aplikáciách.
- Čitateľnosť: ľahkosť, s akou môže vývojár čítať alebo porozumieť zdrojovému kódu alebo technickej dokumentácii systému.
- Testovateľnosť: úsilie, ktoré treba vynaložiť na testovanie vlastností výrobku, napr. či vykazuje požadované správanie.
- Zrozumiteľnosť: ľahkosť, s akou môže niekto pochopiť softvérovému systému na systémovo-organizačnej a tiež na detailnej úrovni. Zrozumiteľnosť v porovnaní s čitateľnosťou súvisí so zviazanosťou a súdržnosťou systému vo všeobecnejšej rovine. Zahŕňa porozumenie nielen toho, čo systém robí, ale prečo to robí.

Chyby a zdroje chýb

Ako už bolo spomínané, chyby patria medzi neužitočné vlastnosti softvérových produktov. S chybami v softvéri sa stretol asi každý človek, ktorý denne pracuje s počítačom. Nie je zriedkavosťou, že aplikácia alebo operačný systém „padne“, pričom to býva v tej najnevhodnejšej chvíli.

Vývoj softvéru prešiel za uplynulé desaťročia mnohými významnými zmenami. Asi najväčší skok z pohľadu redukcie chýb bol zaznamenaný s príchodom tzv. typových jazykov ako je C++, Smalltalk a neskôr Java. Taktiež sa zlepšili aj testovacie či odlaďovacie nástroje. Napriek tomu sa však zdá, že chýb v softvéri neubúda. Čím to je?

Jednou z odpovedí na túto otázku je, že za posledné roky sa zvýšila aj zložitosť softvéru. Tieto chyby, ktoré vyplývajú zo zložitosti vytváraného systému, sú čoraz „zákernejšie“ a ťažšie rozpoznateľné. Zložitosť je zapríčinená najmä užitočnými (ktoré sú potrebné na správne fungovanie systému), ale často aj postrádateľnými (také, ktoré vznikli z „rozmaru“ zákazníka) funkciami systému.

Ďalším významným faktorom, ktorý vplýva na počet chýb je, že zložité systémy nevytvára jeden človek, ale tím ľudí. Tímová práca vnáša do tvorby softvéru chyby, ktoré vyplývajú najmä zo zlej komunikácie medzi jednotlivými členmi tímu. Na veľkých projektoch sa často rozdeľujú jednotlivé úlohy medzi rôzne tímy, pretože nie je možné, aby jedna malá skupinka ľudí daný projekt vytvorila v určenom časovom rozpätí. Pri takýchto veľkých projektoch nastáva podobný problém, a to najmä v zlej komunikácii medzi jednotlivými tímami.

Jeden z najvýznamnejších zdrojov chýb spočíva v zlej, resp. nedostatočnej komunikácii so zákazníkom. Komunikácia so zákazníkom je veľmi dôležitá, pretože si musíme uvedomiť, že daný produkt sa vytvára práve pre zákazníka. Je nepríjemné zistiť pri odovzdávaní produktu, že zákazník takto vytvorený produkt nechcel. V [14] sa uvádza, že až 15% zo všetkých chýb je spôsobených zlou komunikáciou so zákazníkom. Vzhľadom na túto skutočnosť, by sa mal každý výrobca softvéru riadiť heslom „Náš zákazník, náš pán“.

Špeciálnym druhom chýb sú tzv. bezpečnostné chyby. Tieto chyby sa prejavujú až pôsobením tretej strany. Nejde teda o chybu v pravom slova zmysle. Na druhú stranu je však v dnešnej dobe otázka bezpečnosti veľmi dôležitá, pretože žiaden zákazník asi nechce, aby prišiel o dáta, alebo aby sa jeho citlivé údaje dostali do nepovolaných rúk.

Čo je problematické pre softvérové systémy?

Pri stanovovaní priorít pri tvorbe softvérového systému narážame často na problémy. Tie súvisia s rozdielnymi požiadavkami na kvalitu softvéru z pohľadu zákazníka a z pohľadu tvorcu, resp. údržbára systému.

Zákazník často požaduje, aby mal systém čo najviac funkcií, vysokú mieru konfigurovateľnosti a prispôsobiteľnosti. Z pohľadu vývojára takáto „zbytočná“ funkcionalita zvyšuje zložitosť kódu, čo následne prináša vysoké riziko vzniku chýb. Taktiež, čím je zložitejšia funkcionalita systému, tým sa znižuje miera testovateľnosti, pretože je nutné vytvoriť väčšie a komplexnejšie vzorky testovacích vstupov.

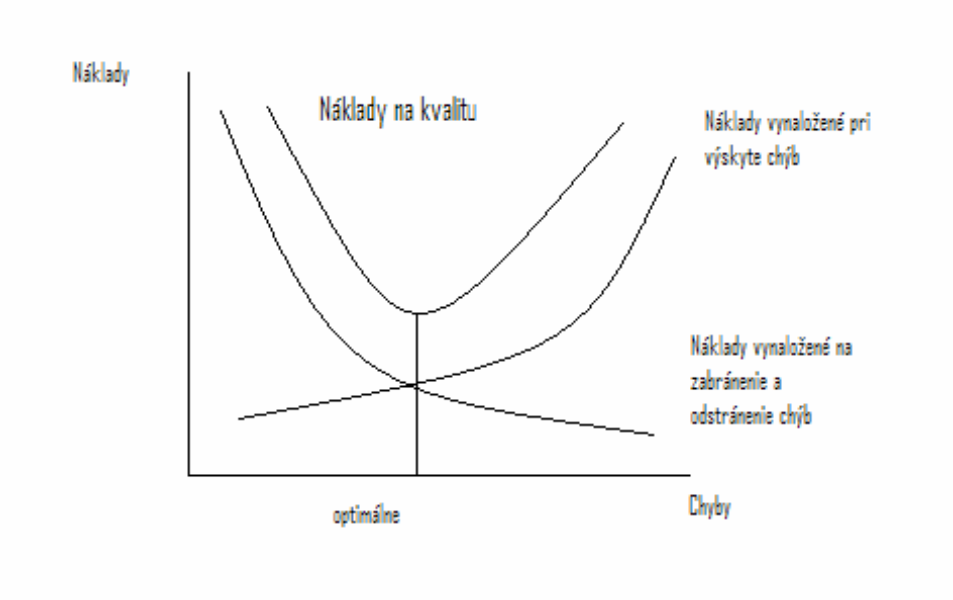
Z pohľadu údržbára sa táto zložitosť prejaví ako zhoršená miera udržiavateľnosti systému.

Ako ďalší príklad môžeme uviesť, že zákazník často požaduje vysokú mieru efektívnosti systému. Z pohľadu vývojára to znamená, že systém je nutné prispôsobiť čo najviac požiadavkám zákazníka a následne upraviť triedy, knižnice, moduly alebo celé podsystémy, ktoré boli použité a prebrané z iných podobných systémov. To zapríčiní nielen nižšiu schopnosť znovupoužiteľnosti (či už existujúcich komponentov alebo komponentov, ktoré sa práve v systéme vyrábajú), ale opäť sa tu vynára vysoké riziko vzniku chýb v už odladených komponentoch.

Zákazníci často požadujú, aby vytváraný systém dokázal spolupracovať s inými už existujúcimi systémami, t.j. požadujú vysokú mieru interoperability. Pre vývojára to ale znamená vytvorenie nových rozhraní a taktiež s tým súvisiace ošetrenie neočakávaných vstupov od iných systémov, t.j. zvýšenie miery robustnosti systému, čo opäť zapríčiní zvýšenie pravdepodobnosti vzniku chýb.

Jeden z najzávažnejších problémov však nastáva, keď zákazník nie je schopný dodať úplnú špecifikáciu vytváraného projektu (čo býva prakticky vždy). Špecifikácia je častokrát neúplná a čo je horšie, nekonzistentná. Zákazník často mení svoju špecifikáciu počas vývoja softvéru, čo je už v rozbehnutom vývoji softvéru veľmi nepríjemné a celý projekt sa tým môže predražiť.

Asi najdôležitejšou požiadavkou zákazníka je, aby bol daný softvérový produkt bezchybný. Takáto požiadavka je však prakticky nerealizovateľná, pretože by to trvalo „nekonečne“ dlho a stálo by to „nekonečne“ veľa peňazí. Preto je nutné hľadať optimálne riešenie, t.j. minimalizovať sumu nákladov vynaložených na odstraňovanie chýb. Súvis medzi týmito hodnotami je na obrázku **Obr. 2.**



Obr. 2. Súvis medzi nákladmi, chybami a kvalitou

Manažment kvality

Systém manažérstva kvality má svoje korene v 50-tych rokoch povojnového Japonska. Za jeho zakladateľa možno považovať W. Edwarda Deminga, ktorého postupy na riadenie kvality boli úspešne aplikované počas obnovy zničeného japonského priemyslu. Počas osemdesiatych rokov boli vydané prvé certifikáty ISO 9000. Dnes je manažment kvality široko používaný v rôznych odvetviach priemyslu, a to najmä v automobilovom a elektronickom.

Pod pojmom manažérstvo kvality sa bude rozumieť koordinačná činnosť všetkých oddelení podniku tak, aby vyrábané výrobky mali vlastnosti zhodné s požiadavkami zákazníka [14].

Použitie manažérstva kvality v softvérovom inžinierstve má určité špecifiká. Podstatný rozdiel medzi tvorbou softvéru a napr. výrobou ložísk je v tom, že tvorba softvéru nie je vo všeobecnosti opakovateľný proces. Inými slovami povedané, pokiaľ vytvárame dva rôzne softvérové produkty, potom proces ich výroby nie je rovnaký. Dôvodom neopakovateľnosti procesu výroby je najmä skutočnosť, že každý (zákaznícky) vytváraný softvér je nejakým spôsobom špecifický a pri jeho realizácii nie je možné opakovať tie isté úkony, ako tomu bolo pri iných.

Ďalším dôležitým aspektom odlišnosti medzi softvérom a inými výrobkami je istá obtiažnosť pri meraní jednotlivých znakov kvality. Kým pri už spomínanom ložisku je pomerne jednoduché zistiť meraním jeho priemeru alebo váhy, či vyhovuje alebo nie, pri stanovovaní znakov kvality softvéru to nie je až také jednoduché.

Manažment kvality vychádza z predpokladu, že zlepšením kvality procesu výroby sa zvýši aj kvalita vyrábaného výrobku. Takýto postup sa používa najmä preto, lebo v softvérovom inžinierstve nie je ľahké merať výstupnú kvalitu pomocou nejakej výstupnej kontroly [15]. Zlepšovaním procesu výroby sa znižujú aj náklady na kvalitu, t.j. pri rovnakej cene za vyššiu kvalitu alebo za rovnakú kvalitu zaplatíme nižšiu cenu.

Aktivity vykonávané pri manažmente kvality

Ako už bolo spomínané, hlavnou úlohou manažérstva kvality je zaistenie kvality pri výrobe softvérových produktov. Hlavné aktivity manažmentu kvality sú [15]:

- Plánovanie kvality: Výber vhodných procedúr a štandardov pre špecifický projekt a ich následné prispôsobenie. Vytvára sa tzv. plán kvality, v ktorom sa jasne stanovuje, ktoré atribúty kvality sú najdôležitejšie pre daný vyvíjaný výrobok. Definuje sa proces zabezpečenia kvality (procedúry zabezpečenia kvality sa dokumentujú pomocou tzv. príručky kvality, tieto treba vytvárať na základe postupov, ktoré sa skutočne používajú). Definujú sa metriky, ktoré slúžia na hodnotenie stupňa dosiahnutej kvality.
- Zabezpečovanie kvality: Koordinácia, usmerňovanie vykonávania všetkých tých plánovaných a systematických činností, ktoré treba na to, aby sa vytvorila dostatočná dôvera, že výrobok alebo služba bude spĺňať určité požiadavky. Kvalitu musia zabezpečovať všetci účastníci projektu.

– Riadenie kvality: Sledovanie výsledkov a identifikácia spôsobov eliminácie nevyhovujúcej kvality. Riadenie sa uskutočňuje na základe analýz meraní. Zabezpečenie riadenia často vykonáva skupina na zabezpečenie kvality (SQA group).

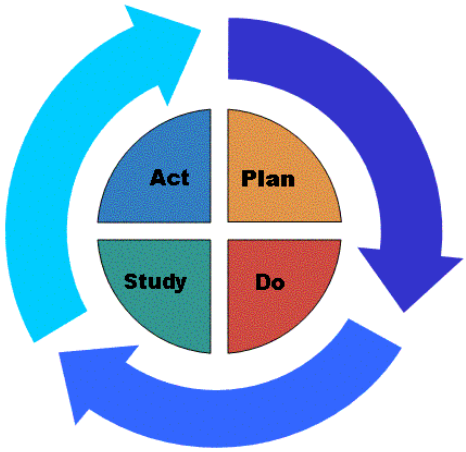
Na celý tento postup sa možno pozrieť aj z alternatívneho hľadiska, a to z pohľadu Demingovho PDSA cyklu. Ako možno vidieť z obrázku **Obr. 3,** Demingov cyklus sa skladá zo štyroch častí, ktoré sa „donekonečna“ opakujú v danom poradí. Význam jednotlivých častí cyklu je nasledovný [14]:

– Plánovanie (Plan): Zaznamenanie stavu procesov, analýza štatistických dát a procesov, určenie vplyvových veličín a navrhnutie vylepšení procesov.

– Realizácia (Do): Vyvodenie opatrení z navrhnutých zmien pri fáze plánovania a ich následné zavedenie a presadenie. Následne dostávame zmenené výsledky.

– Preskúmanie (Study): Porovnanie dokumentovaných výsledkov s východiskovým stavom. Ak nastalo zlepšenie, fáza plánovania správne identifikovala vplyv správnych vplyvových veličín, ak nastalo zhoršenie výsledkov, tak identifikácia bolo nesprávna.

– Konanie (Act): celý cyklus musí prebehnúť ešte raz, aby sa vykonala náprava, alebo aby sa dosiahlo ďalšie zlepšenie.



Obr. 3. Znázornenie PDSA Demingovho cyklu.

Ako merať kvalitu softvéru?

Meranie softvéru je odvodením numerickej hodnoty pre každý atribút softvérového produktu alebo procesu. Softvérová metrika je ľubovolný typ merania súvisiaci

so softvérovým systémom, procesom alebo dokumentáciou [13]. Metrika vlastne umožňuje softvéru a softvérovému procesu byť kvantifikovateľný.

Ak sa teraz zamyslíme nad tým, ako merať jednotlivé znaky kvality, zistíme, že externé znaky kvality softvéru sa merajú ľahšie ako interné. Je to zapríčinené tým, že externé znaky sa často prejavujú na samotnom produkte a sú tým pádom dobre a objektívne merateľné. Pre interné znaky je situácia o niečo komplikovanejšia, pretože ich hodnotenie je veľmi subjektívne, a preto takéto meranie nie je možné pokladať za absolútne. Ako príklad interných znakov kvality môžeme uviesť zrozumiteľnosť, kde rýchlosť pochopenia systému bude závisieť od osobných schopností a skúseností programátora. Ako príklad externých znakov môžeme uviesť jeden z najdôležitejších znakov kvality softvéru a to spoľahlivosť, ktorá je charakterizovaná strednou dobou medzi poruchami (výpadkami) MTBF a strednou dobou opravy MTTR. Pre spoľahlivosť potom platí [14]:

$$Spolahlivost = \frac{MTBF}{MTBF + MTTR}$$

Meranie kvality softvéru predstavuje, resp. meranie v softvérovom projekte je široká oblasť sama o sebe. Cieľom tejto časti eseje nebolo objasniť spôsob merania v softvérovom projekte, ale poukázať na rozdiely v meraní interných a externých znakov kvality.

Využitie manažmentu kvality v malých projektoch

Sústava ISO 9000 predstavuje dobrú skupinu noriem na zabezpečovanie kvality. Bohužiaľ, sú s ňou spojené obrovské náklady na dodatočnú pracovnú silu, dokumentáciu a pod. Pre malé projekty (cca 5 ľudí) si je vhodné z nej vybrať v obmedzenej miere niektoré postupy a normy. Jedným zo zaujímavých postupov je statická previerka kódu, kde sa zistí, či sa programátor drží dokumentácie, alebo sú rozpory medzi implementovaným kódom. Taktiež je vhodné vytvoriť si nejakú metriku na identifikáciu kritických častí kódu a túto metriku postupne zlepšovať a spresňovať.

Odporúčal by som aj založenie dokumentácie manažmentu kvality, kde budú zhrnuté pravidlá pre členov tímu a výsledky previerok a testovania zdrojového kódu a dokumentácie.

Mále projekty sú svojim spôsobom špecifické a nemá v nich veľkú cenu dbať na rozsiahlu dokumentačnú časť riadenia kvality, ako to požaduje ISO 9000. Skôr by bolo vhodné zaviesť akúsi „kultúru kvality“, kde každý člen tímu bude niesť osobnú zodpovednosť za kvalitu.

Záver

Kvalita je neoddeliteľnou súčasťou nášho vnímania a posudzovania výrobkov ako takých. Výrobcovia softvéru by si mali uvedomiť, že kvalita je v skutočnosti to, čo sa predáva, resp. to, čo každý zákazník kupuje. Chybám počas vývoja softvéru sa prakticky nedá vyhnúť. Manažment kvality zvyšuje kvalitu softvéru, a to takým

spôsobom, že neustále vylepšuje proces jeho výroby. Ak by sa proces výroby priebežne nezlepšoval, potom by sa začal zhoršovať. Manažment kvality úzko súvisí aj s manažmentom rizík, pretože predpokladá, že kvalitným procesom výroby softvéru sa dá vyhnúť aj prípadným veľkým škodám pri zmene špecifikácie počas vývoja softvéru.

Použitá literatúra

13. Ian Sommervile: Software Engineering 7th edition, Chapter 27, 2004.
14. Ladislav Hulényi.: Poznámky z predmetu Manažment kvality,(2004)
15. Mária Bieliková.: *Softvérové inžinierstvo – princípy a manažment,* Bratislava, 2000.
16. Richard G. Russell.: *Defining Software Quality,* 2004

Annotation

Quality management and its influence on project result

The quality is with no doubt the most important part of the each product. The basic idea of this essay is to look more deeply into a concept of system quality, in particular, in the context of management of quality. The paper analyses the individual parts of quality characters. It discusses about which of them are important from the customer's view and from the view of the producer of the software. It examines the concept of an error and also contemplates about individual contradictions between the requirements of the customer and that of the software producer. In conclusion it describes the management of the quality as the system of quality control.

Manažment softvérového systému a vplyv na manažment softvérového projektu

MILOŠ RADOŠINSKÝ

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

Abstrakt. Obsahom eseje je stručný úvod do problematiky a oboznámenie sa so základnými konceptmi systémov pre manažment softvéru. V práci je vyhradený tiež priestor problematike vplyvu na manažment projektu.

Úvod

Manažment softvérového systému alebo v angličtine *software configuration management* (SCM) je súčasťou každého softvérového projektu. Jeho úlohou je zaistiť integritu a konzistenciu v softvérovom výrobku počas celého procesu vývoja a údržby. Tieto požiadavky kladené aj na podporné nástroje vyplývajú z podstaty a povahy softvérových systémov.

Softvér je neviditeľný a neuchopiteľný. Na jeho zviditeľnenie používame abstraktné modely ako napr. model štruktúry, funkčné modely a iné. Tieto modely sú súčasťou softvérového systému vo forme *dokumentácie* ako špecifikácia požiadaviek, špecifikácia softvéru a iné. Softvér je zložitý, lebo rieši zložitý problém. To vyžaduje dekompozíciu problému a jeho riešenia na menšie podproblémy a celky. Na najnižšej úrovni delenia softvéru možno rozpoznať *súčiastky* alebo *komponenty,* ktoré sa už nedelia.

Komponenty a dokumenty definované v tomto texte podliehajú ustavičným zmenám. Zvyčajne existujú vo viacerých verziách. Navyše medzi nimi existujú rôzne závislosti (napr. "používa", "je abstrakciou" a iné). Keď k tomu všetkému pripočítame ešte fakt, že softvérový systém môže existovať v niekoľkých variantoch, je nutné zaviesť pravidlá a definovať postupy na udržiavanie konzistencie výrobku. Nie bezvýznamná sa javí podpora spolupráce členov tímu, ktorá môže ovplyvniť celkovú organizáciu tímu.

V tejto práci sa budem venovať podpore nástrojov pre spoluprácu v tíme a organizácii údajov v nástrojoch na podporu manažmentu softvéru.

Základné úlohy v manažmente softvéru

Na podporu manažmentu softvéru treba v projekte plánovať štyri základné úlohy. Tieto úlohy možno rovnako dobre aplikovať v prostredí výrobcu v procese vývoja ako aj v cieľovom prostredí počas údržby. Všetky úlohy by mali byť v rámci organizácie štandardné, ale mali by sa prispôsobovať pre potreby konkrétnych projektov.

Identifikovanie entít

Tu treba definovať časti systému (entity), ktoré budú podliehať kontrole pri zmenách. Tieto entity je nutné jednoznačne identifikovať. To možno docieliť organizovaním systému a jeho častí do hierarchie. Miesto v hierarchii potom môže entitu jednoznačne identifikovať. Identifikované entity môžu byť napr. zdrojový kód programu, plán projektu, akceptačné testy, teda komponenty a dokumentácia systému.

V tomto bode treba definovať aj tzv. základné konfigurácie - *baselines.* Základná konfigurácia (baseline) je dobre definovaný stav výrobku. V projekte možno definovať niekoľko základných konfigurácií. Pre každú z nich treba definovať, ktoré entity majú byť jej súčasťou a v akom stave. Riešenie projektu sa potom plánuje a realizuje s ohľadom na tu definované základné konfigurácie. Možno definovať napr. analytické, návrhové, testovacie základné konfigurácie. V nich sa definujú stavy systému, ktoré budú na konci analýzy, návrhu a testovania. Základná konfigurácia je podobný koncept ako míľnik.

Riadenie zmien

Jeden z najdôležitejších cieľov manažmentu softvéru je zachovať konzistenciu výrobku aj napriek ustavičným zmenám. To sa dosahuje riadením zmien na identifikovaných entitách. Riadenie je súčasťou celého procesu zmeny od zadania až po implementáciu zmeny. Požiadavka na zmenu obyčajne obsahuje tieto informácie: entity, ktoré sa majú zmeniť, pôvodca, dátum, priorita zmeny a popis zmeny. Treba si pritom uvedomiť, že nie je zmena ako zmena. Treba brať na zreteľ stav entity, ktorá sa má zmeniť. Entita, ktorá je súčasťou základnej konfigurácie (baseline), má vyššiu váhu ako entita, ktorá je iba prechodom medzi dvoma základnými konfiguráciami. Toto treba brať do úvahy aj pri posudzovaní zmeny.

Evidovanie stavu systému

Evidovanie stavu systému slúži hlavne manažérom projektu na sledovanie súčasného stavu systému. Bez týchto informácii by neboli schopní posúdiť, či projekt beží v súlade s plánom. Evidované informácie môže využiť napr. komisia na schvaľovanie zmien (CCB) – evidujú sa správy o navrhovaných a schvaľovaných zmenách a správy o problémoch. Komisia CCB môže tiež sledovať stav realizovaných zmien, takže môžu byť rýchlo odhalené a vyriešené potenciálne problémy.

Overovanie stavu systému

Hlavným cieľom tejto úlohy je zabezpečiť kvalitu a udržovať integritu výrobku. Overuje sa tu, či existujúca konfigurácia systému je kompletná, správna a konzistentná. Okrem toho sa overuje, či sprievodné informácie získané v úlohe evidovanie stavu systému sú spoľahlivé a konzistentné s reálnym stavom výrobku.

Podpora spolupráce v tíme

Dnes sa aj tie najmenšie projekty riešia v tíme. Úsilie ľudí v tíme musíme koordinovať tak, aby si vzájomne nezasahovali do práce. Na druhej strane by sme mali vytvoriť také podmienky, aby boli čo najviac produktívni. Koordinácia práce spočíva v riadení zmien. K zmenám nemôže dochádzať náhodne a bez kontroly.

Ak sa komponenty v tíme zdieľajú bez obmedzenia, môže vzniknúť potreba uskutočniť zmeny paralelne. Na to sú používané dve rôzne stratégie: zakázať alebo podporiť paralelné zmeny. Keď sú zakázané, zmeny treba vykonať sekvenčne. Na to stačí zamykať komponenty predtým, ako môžu byť zmenené. Zámok dáva istotu, že iba jedna osoba v čase môže zmeniť komponent. Nástroje, ktoré umožňujú paralelnú zmenu komponentov, jednoducho vytvoria vetvu, aby dali najavo fakt, že sa vykonávajú dve paralelné zmeny, ako v prípade nástroja CVS. Väčšina nástrojov je tiež schopná zlúčiť dve vetvy späť do jednej verzie (vetvy), aby vzniklo zloženie dvoch zmien.

Čo by sme chceli získať, je rozdelenie času na obdobie, kedy vykonávame zmeny izolovane od iných zmien a obdobie, kedy integrujeme vlastné zmeny so zmenami ostatných členov tímu. Na tento účel môžeme použiť tzv. *Pracovný priestor* a tzv. *Centrálne úložisko* (repository). V pracovnom priestore dochádza k zmenám. V centrálnom úložisku dochádza k integrácii zmien. Podrobnejšie sa tomu venujem v kapitole SCM modely.

SCM modely

V nástrojoch na podporu manažmentu softvéru boli podľa [18] vypozorované štyri modely organizácie údajov. Všetky modely majú spoločné to, že používajú jedno centrálne úložisko súborov – *repository*. Jednotlivé modely sa však líšia jeho organizáciou a obsahom údajov. V praxi sa používajú oddelene alebo v kombinácii s malými odchýlkami. Všetky štyri modely realizujú správu konfigurácií a spoluprácu členov tímu rôznym spôsobom.

Check Out/Check In Model

Je základný SCM model, ktorý zavádza koncept centrálneho úložiska (repository), v ktorom sú uložené viaceré verzie jednej súčiastky. Model je zameraný na prácu s individuálnymi súbormi. Nevytvára model systému a nepozná pojem konfigurácie. Verzie súborov sa identifikujú explicitne číslami. Pojmy ako check-out a check-in

označujú operácie výberu a vrátenie súčiastky z a do centrálneho úložiska. Tieto pojmy sa uplatňujú aj pri ďalších modeloch. Do tejto kategórie možno zaradiť systémy Source Code Control System (SCCS) alebo Revision Control System (RCS).

V tomto modeli sa uplatňuje pesimistický model spolupráce. Jednu súčiastku môže modifikovať najviac jeden člen tímu. Ak aj druhý člen chce modifikovať rovnakú súčiastku, musí vytvoriť jej novú vetvu (variant) v grafe verzií. Neskôr sa môžu obidve vetvy spojiť do jednej vetvy pomocou vizualizačného nástroja, ktorý zobrazí ich rozdiely. Ak chce druhý člen tímu obsah súčiastky iba čítať, druhá vetva sa nevytvorí. Systém (model) pritom rozlišuje výber súčiastky s právom na zápis a na čítanie. Právo na zápis dáva možnosť modifikovať a vrátiť súčiastku do úložiska (repository).

Kompozičný Model

Kompozičný model rozširuje SCM z úrovne komponentov na úroveň celého systému. Zavádza model systému a konfiguráciu systému, ktoré popisujú štruktúru systému ako množinu komponentov, verzií komponentov a ich relácií. Model systému tu reprezentuje všetky možné konfigurácie systému. Konfigurácia vzniká z modelu systému aplikovaním výberových pravidiel, ktoré v rámci jednej rodiny komponentov vyberú konkrétnu verziu na základe určitých vlastností verzie. Konfigurácia je potom daná modelom systému a výberovými pravidlami.

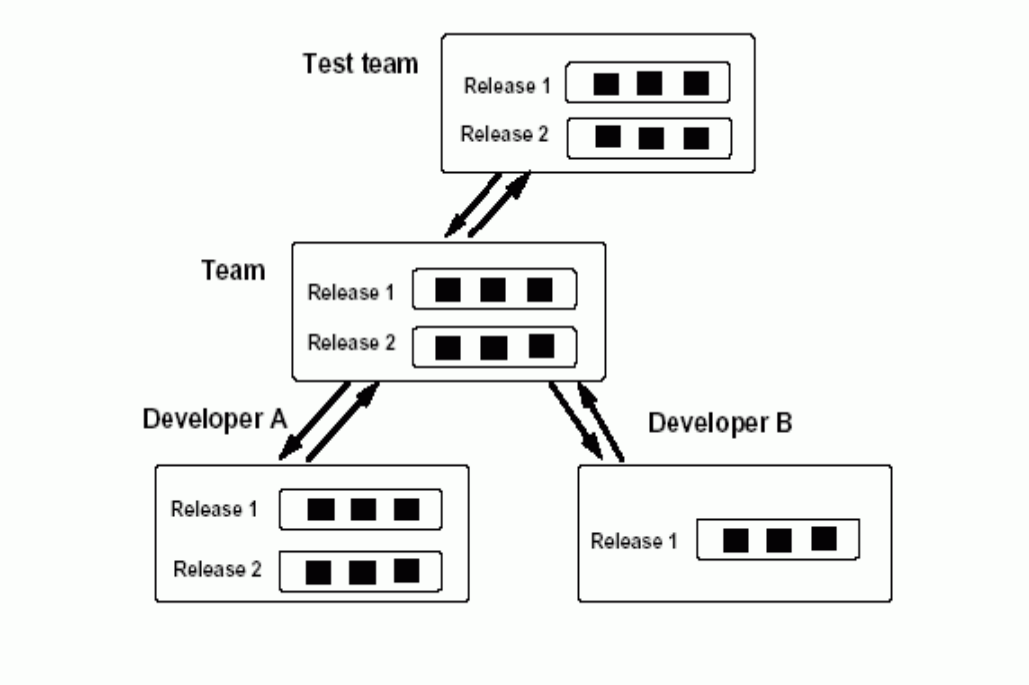
V modeli možno rozpoznať dva typy konfigurácii – viazaná a čiastočne viazaná. Viazaná konfigurácia je v čase jednoznačná. Výberové pravidlá vyberajú konkrétne verzie explicitne a jednoznačne. Táto konfigurácia môže byť pomenovaná názvom. Je trvalo uložená a vedená v centrálnom úložisku. Môžu sa na ňu odkazovať iné konfigurácie. Čiastočne viazaná konfigurácia nie je v čase jednoznačná. Výberové pravidlá v nej vyberajú verzie implicitne, napr. najnovšia verzia. Tento typ konfigurácie je označovaný ako konfiguračný predpis alebo šablóna (template). Používa sa na aktualizáciu pracovného priestoru (working area) vývojára. Rozdielne konfigurácie sa môžu vytvoriť použitím rôznych výberových pravidiel verzií. Na riadenie spolupráce tímu sa používa model Check In/Check Out.

Transakčný Model

Transakčný model chápe evolúciu systému ako postupnosť atomických – transakčných zmien. Tieto zmeny sa vykonávajú nad kópiou pôvodných údajov v tzv. *Pracovnom priestore.* Pracovný priestor je miesto, kde dochádza k izolovaným zmenám systému. Tieto zmeny sú počas transakcie pre pôvodné dáta (systém) neviditeľné. Po ukončení transakcie sa zmeny šíria na miesto pôvodu. Transakciu teda možno chápať ako postupnosť krokov a operácii, ktoré vedú k izolovanej zmene údajov (systému).

Tento koncept sa zatiaľ nijako nelíši od konceptu centrálneho úložiska (repository) v ostatných modeloch SCM. Nie je to celkom tak. Pridanou hodnotou transakčného modelu je pracovný priestor. Oproti iným modelom SCM má pracovný priestor v transakčnom modeli niekoľko zaujímavých vlastností. Pracovný priestor má podobné vlastnosti ako centrálne úložisko. Môže prijímať, uchovávať a posielať nové

konfigurácie systému – môže byť miestom vzniku alebo miestom uloženia konfigurácie systému. Okrem toho sa pracovné priestory môžu organizovať v hierarchii, v ktorej sú definované vzťahy podriadený a nadriadený. Podriadený pracovný priestor je zdrojom konfigurácie (údajov) pre nadradený pracovný priestor. Napríklad pracovný priestor celého tímu je podriadený pracovnému priestoru jedného člena tímu. V každom pracovnom priestore môžu prebiehať transakčné zmeny nad údajmi z podriadeného pracovného priestoru. Po ukončení transakcie sa zmeny šíria na miesto pôvodu – podriadený pracovný priestor.



Obr. 4. Organizácia tímu

Uvediem teraz príklad organizácie tímu podľa obrázku. Na obrázku **Obr. 4** je znázornená hierarchia pracovných priestorov. Každý vývojár má vlastný pracovný priestor, ktorý je nadradený pracovnému priestoru tímu a ten je nadradený testovaciemu tímu. Vývojár vyberá konfiguráciu z pracovného priestoru tímu. Zmena vo vybranej konfigurácii zahájí transakciu. V pracovnom priestore vývojára sa vytvorí pracovná konfigurácia, na ktorej sa robia zmeny. Po dokončení zmien sa otvorená transakcia uskutoční (ukončí). Vzniká nová konfigurácia v mieste pôvodu. Uskutočnené zmeny sa zviditeľnia v pracovnom priestore tímu.

Naznačený postup nebýva vždy tak priamočiary. Komplikácie môžu nastať v situácii, ako je na obrázku, keď na jednej vetve konfigurácie systému pracujú viacerí vývojári. Pracovný priestor tímu je zdieľaný a dochádza v ňom k súbežnej aktualizácii systému. V pracovnom priestore tímu sa zviditeľňujú čiastkové zmeny od jednotlivých

členov tímu. Každá zmena sa zviditeľní v novej konfigurácii v priestore tímu. Tá vznikne zlúčením zmenenej konfigurácie vývojára s najnovšou konfiguráciou v priestore tímu. V pracovnom priestore tímu tak vznikajú nové konfigurácie ako výsledok zlúčenia všetkých čiastkových zmien. Bezkolízne zlučovanie konfigurácií sa deje automaticky. Ak ale systém rozpozná kolíziu, konfigurácie sa musia zlúčiť manuálne. Konfigurácie možno zlučovať aj v pracovnom priestore jednotlivých členov tímu, keď požiadajú o aktualizáciu svojho pracovného priestoru. Po tom, ako sa úspešne zlúčia všetky čiastkové zmeny v pracovnom priestore tímu, môže sa ich transakcia ukončiť. V pracovnom priestore testovacieho tímu sa objaví najnovšia konfigurácia. Po úspešnom otestovaní testovací tím ukončí transakciu a otestovaná konfigurácia sa zapíše do centrálneho úložiska.

V transakčnom modeli používateľ pracuje primárne s konfiguráciou systému. To znamená, že používateľ primárne vidí konfigurácie. Model systému a použité verzie sa odvodia z konfigurácie systému – sú v nej definované. Tento prístup je opačný vzhľadom na kompozičný model SCM, kde používateľ vyberá najskôr model systému (štruktúru systému) a potom konkrétne verzie komponentov (konfiguráciu).

V modeli sú tri kategórie súbežnosti:

- súbežnosť v rámci jedného pracovného priestoru,
- súbežnosť medzi viacerými pracovnými priestormi,
- súbežnosť, nezávislý vývoj.

V prvom prípade sú súbežné zmeny zakázané. Obmedzuje sa buď prístup do pracovného priestoru alebo aktivita na jednu osobu. Okrem toho je tiež možné zamknúť individuálne komponenty. V druhom prípade zmeny v oddelených priestoroch spoločne vyvíjajú systém. Modely spolupráce, ktoré riešia túto súbežnosť, sú buď optimistický alebo pesimistický model. Pesimistický model zamyká zdieľané podriadené pracovné priestory alebo zdieľané centrálne úložisko. Tretí prípad predpokladá, že systém sa vyvíja na nezávislých konfiguráciách – variantoch.

Model založený na zmenách

Ako vyplýva z názvu, tento model sa zameriava na zmeny, nie na verzie. Pri vývoji systému vznikajú požiadavky na rozšírenie systému alebo opravu chýb – *požiadavky na zmenu.* V jednej požiadavke na zmenu sa zvyčajne špecifikuje iba jedna *logická zmena.* Avšak tá môže spôsobiť zmeny vo viacerých komponentoch – *fyzické zmeny.* Teda jedna požiadavka na zmenu môže vyvolať niekoľko fyzických zmien. Orientácia na zmeny má preto oproti orientácii na verzie niekoľko výhod. Vývojári pristupujú a pracujú naraz so skupinou komponentov, ktoré patria k jednej logickej zmene. Požiadavky na zmeny sa môžu jednoducho priradiť k fyzickým zmenám komponentov.

Konfiguráciu systému tu tvorí tzv. *baseline* (model systému) a množina logických zmien. Model systému je v tomto prípade daný množinou komponentov a ich relácií. Pričom každý komponent je v modeli zastúpený práve jednou verziou, na rozdiel od kompozičného modelu systému (súčasťou modelu systému sú všetky verzie).

Rozdielne konfigurácie môžu vzniknúť aplikovaním rozdielnej sady logických zmien na základ (baseline). Avšak nie všetky kombinácie zmien sú konzistentné. Niektoré zmeny sú vzájomne závislé alebo konfliktné. Konfliktné zmeny sa nesmú použiť súčasne. Naopak závislé zmeny sa musia použiť súčasne. Na riadenie súčinnosti tímu sa používa model CheckIn/CheckOut.

Zhrnutie vlastností modelu:

- Každá fyzická zmena komponentu je súčasťou logickej zmeny.
- Každý komponent môže byť súčasťou logickej zmeny.
- Každá konfigurácia môže mať niekoľko logických zmien.

Identifikácia verzií

Podľa [20] existujú tri spôsoby ako rozlíšiť verzie komponentu:

1. Číslovanie alebo značkovanie verzií sa používa na značkovanie variantov alebo číslovanie revízií. Revízie sa číslujú, pretože sú usporiadané do postupnosti a ich poradie je dôležité. Varianty sa značkujú názvom, pretože nie sú usporiadané do konkrétnej postupnosti a názov variantu (vetvy) obyčajne odráža niektoré vlastnosti variantu (vetvy). Čísla a značky sú súčasťou grafu histórie verzií. V tomto grafe má každá revízia svoje číslo a každý variant svoje meno – každá revízia a variant má v grafe svoje miesto.
2. Identifikácia na základe vlastností (atribútov) sa používa na identifikáciu variantov. Pre každý komponent sa najskôr definuje n-rozmerný vektor vlastností. Tento vektor slúži na identifikáciu variantov jedného komponentu. Variant komponentu sa potom vyberá z n-rozmerného priestoru verzií, kde každý rozmer reprezentuje jednu vlastnosť komponentu. Takýto prístup nepoužíva a nie je obmedzený na graf verzií.
3. Identifikácia pomocou zmien sa používa na identifikáciu revízií. Miesto číselného značenia revízie možno použiť značenie na základe logických zmien, pričom každá logická zmena má svoje meno. Revízia sa potom môže identifikovať ako postupnosť pomenovaných zmien, ktoré sa aplikovali na základ (baseline).

Uvedené spôsoby možno kombinovať. To vyplýva z rozdielnej podstaty verzií. Pre revízie možno použiť identifikáciu na základe čísla alebo na základe zmien. Pre varianty možno použiť značkovanie alebo vlastnosti.

Záver

V tomto príspevku som sa snažil podať stručný úvod do problematiky manažmentu softvéru a oboznámiť čitateľa so základnými konceptmi systémov pre manažment softvéru.

Použitá literatúra

17. L. Bendix, A. Dattolo, F. Vitali: Software configuration management in software and hypermedia engineering: A survey. Department of Computer Science, Aalborg University.
18. Peter H. Feiler: Configuration Management Models in Commercial Environments. Technical Report CMU/SEI-91-TR-7 ESD-9-TR-7, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, March 1991
19. Jari Vanhanen: Improving configuration management processes of a software product. Helsinki University of Technology, december 97
20. Andreas Zeller.: Configuration Management with Version Sets A Unified Software Versioning Model and its Applications. Der Technischen Universität Braunschweig, April 1997

Annotation

Software Configuration Management and the impact on Software Project Management

This essay introduces the basic concepts of Software Configuration Management and describes different types of software tools for SCM. The different types of SCM tools are described from technical and from teamwork point of view. It handles the SCM impact on the project management too.

Zlepšovanie produktivity softvérových tímov

PETER VOJTEK

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

Abstrakt. Táto esej sa venuje produktivite softvérového tímu najmä s cieľom pomôcť produktivitu zvýšiť. V prvej časti analyzujeme vplyvy a faktory, ktoré majú dopad na produktivitu tímu. Jednotlivé vplyvy sú kvantitatívne porovnané a najvýznamnejší vplyv – schopnosť tímu – je v ďalšej časti práce hlbšie preskúmaný. V rámci schopnosti tímu ako spôsobilosti tímu plniť zadané úlohy sa pozornosť venuje osobnostnému zloženiu menšieho tímu, pričom do úvahy sa berie vhodnosť rôznych typov osobností na jednotlivé pozície v softvérovom tíme. Táto charakteristika je opísaná pomocou typológie osobnosti MBTI (Mayers-Briggs Type Indicator). Súčasťou práce je aj úvod do psychológie osobnosti a bližší opis typológie MBTI.

Úvod

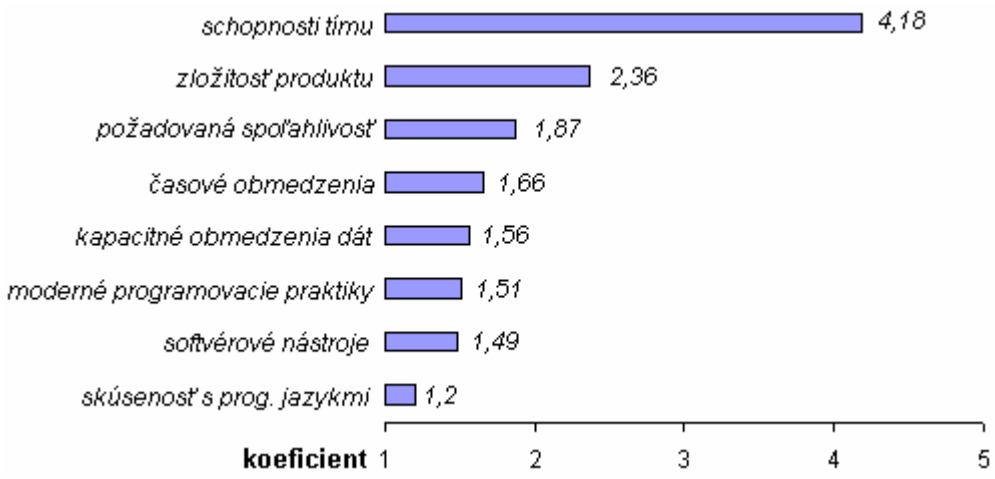
Softvérovým projektom sa príliš často stáva, že prekračujú časové a finančné ohraničenia. Keby sme na ich výsledky nazerali rovnako kriticky ako na iné produkty, mohli by sme mnohé výsledky softvérových projektov považovať za nesplňujúce požiadavky – najmä finančné, časové a kvalitatívne.

Túto skutočnosť zapríčiňuje viac okolností, jednou z nich je aj produktivita tímu. Tejto ľahko opísateľnej ale ťažko merateľnej veličine je venovaná najväčšia pozornosť v tejto práci.

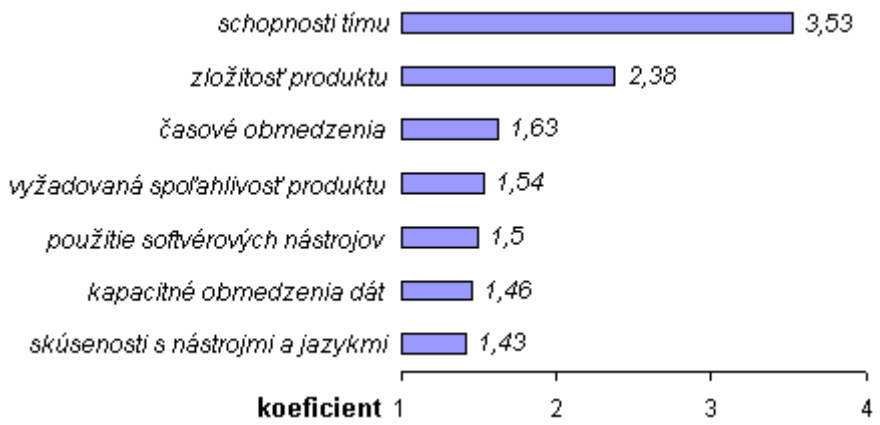
Faktory ovplyvňujúce produktivitu

Na aké aspekty sa sústrediť pri snahe zlepšiť produktivitu a znížiť cenu vývoja softvéru? Možností je na prvý pohľad dosť. Použiť vhodné softvérové nástroje, zamestnať ľudí s expertnou znalosťou programovacích jazykov a skúsenosťami, aplikovať vhodné metódy a prostriedky manažmentu.

Podľa štúdie [21] sú všetky tieto vymenované aspekty dôležité, ale výrazne ich predčí iný faktor – schopnosti tímu. Schopnosťou tímu sa myslí spôsobilosť tímu ako celku plniť dané úlohy. Obrázky **Obr. 5** a **Obr. 6** porovnávajú vybrané faktory v rokoch 1980 a 2000.



Obr. 5. Faktory vplývajúce na produktivitu softvérového tímu, rok 1980.



Obr. 6. Faktory vplývajúce na produktivitu softvérového tímu, rok 2000.

Je očividné, že ľudský faktor ovplyvňuje produktivitu softvérového tímu v omnoho väčšej miere než technológie. Táto skutočnosť sa nezmenila ani pri porovnaní v rozmedzí dvadsiatich rokov. Tento fakt je zaujímavý pri uvedomení si, že dvadsať rokov je obdobie, za ktoré sa iné skutočnosti pri vývoji softvéru zmenili na nepoznanie.

V ďalšej časti sa táto esej sústredí práve na ľudí v softvérovom tíme. Pre ozrejmenie všetkých pojmov je však najprv potrebné objasniť niektoré oblasti z psychológie, presnejšie typológie osobnosti.

Porovnávanie osobností

V psychológii pojem osobnosť označuje súhrn emócií, správania a myšlienok jedinečných pre danú osobu [22]. Porovnávať a hodnotiť osobnosti možno pomocou modelov osobností, ktoré sa delia do troch hlavných tried:

- **Typológie** (typologies) – pri tomto prístupe sa osobnosti delia podľa typov. Príkladom je známe avšak nevedecké rozdeľovanie osobností podľa znamení zverokruhu.
- **Faktory** (factorial models) – prístup vychádza z myšlienky, že osobnosti možno porovnávať v rámci zvoleného priestoru tvoreného rozmermi, dimenziami osobnosti.
- **Cirkumplex** (circumplex) – predstavuje premostenie medzi prvými dvoma triedami modelov rozvíjajúc viac do hĺbky vzťahy medzi rôznymi typológiami alebo faktormi.

V nasledujúcej časti je bližšie opísaný model osobnosti MBTI nazerajúci na osobnosť jedinca pomocou typológie.

MBTI typológia osobnosti

Skratka MBTI znamená Myers-Briggs Type Indicator [23] a označuje psychologický test, pomocou ktorého možno určiť typológiu osoby. Test bol vytvorený ešte počas druhej svetovej vojny a vychádza z prác C. G. Junga.

Profil osobnosti vyplýva z určenia typov, ku ktorým sa človek prikláňa. Tento prístup je v kontraste s inými psychologickými testmi, napr. meraním inteligencie, kde sa určujú povahové črty. Zatiaľ čo črty úzko súvisia a menia sa so schopnosťami človeka, preferované typy sa v zdravom prostredí menia prirodzene, nezávisle od nadobudnutých schopností.

MTBI sa skladá zo štyroch dvojdielností (dichotómií):

- **Extraverzia/introverzia** – určenie smeru, ktorým človek „orientuje“ svoju snahu a energiu. Typ *extraverzia* sa orientuje na svet okolo seba a ostatných ľudí, typ *introverzia* sa sústreďuje na seba a svoje myšlienky.
- **Zmysly/intuícia** – určenie spôsobu, akým človek prijíma informácie. Typ *zmysly* sa spolieha na päť zmyslov, typ *intuícia* berie do úvahy skôr mimovoľné, ťažko opísateľné zdroje informácií.
- **Myslenie/cítenie** – hodnotiace funkcie. Typ *myslenie* používa rozhodovanie blízke matematickej logike „pravda-nepravda“ či podmienené výrazy „ak-tak-inak“. Typ *cítenie* hodnotí pomocou výrazov „viac-menej“ alebo „lepšie-horšie“.
- **Usudzovanie/vnímanie** – ak je človek typ *usudzovanie*, snaží sa o rýchle vyriešenie problému, zvolenie si jednej možnosti. Tento typ sa rozhoduje na základe vonkajších, spoločenských pravidiel. Naopak, typ *vnímanie* rád

necháva všetky možnosti otvorené a rozhoduje sa na základe osobných pravidiel.

Úspešný tím

Je zrejmé, že pri zvyšovaní produktivity softvérového tímu sa „oplatí“ klásť najväčší dôraz na samotný tím. Aspekty, ktoré ovplyvňujú výkonnosť tímu, sú najmä personálne zloženie tímu, vedenie tímu, komunikácia a koordinácia v rámci tímu.

Tieto skutočnosti sa tiež líšia v závislosti od veľkosti tímu [24]. V tejto eseji ďalej venujeme pozornosť len menším softvérovým tímom s tromi až siedmimi členmi. Cieľom je analyzovať nasledovné:

- určiť vzťah medzi typom osobnosti vedúceho tímu a výkonom tímu,
- určiť účinky osobností jednotlivých členov tímu na výkon tímu,
- napokon, analyzovať vplyv rôznorodosti osobností členov tímu na výkon tímu.

Osobnostné zloženie menšieho tímu

Nasleduje opis typov osobností pre tímy s tromi až siedmimi členmi. Analyzované sú MBTI typy osobností vhodné pre jednotlivé pozície členov tímu a tiež pre tím ako celok. Celkové zhrnutie je v tabuľke **Tab. 2**.

Vedúci tímu

Z výsledkov práce [25] vyplýva, že významným ukazovateľom pri tejto role v tíme je najmä dichotómia *zmysly/intuícia*. Tímy s vedúcimi typu *intuícia* ukázali lepšie výsledky než pri vedúcich typu *zmysly*. Intuitívne typy majú lepšiu schopnosť vnímať celkový obraz situácie, takisto sú úspešnejší pri vytváraní a ohodnocovaní alternatívnych riešení.

Ďalej, vedúci tímu je úspešnejší, ak sa na osi *myslenie/cítenie* nachádza bližšie pri type *cítenie*. Dôvodom je silnejšia orientácia na osoby – takýto vedúci svoje rozhodnutia zvažuje s ohľadom na ich dôsledky na jednotlivých členov tímu.

Systémový analytik

Na rolu systémového analytika má významný dopad len dichotómia *myslenie/cítenie*. Analytici typu *myslenie* sú úspešnejší než typ *cítenie*. Znamená to, že analytické schopnosti tejto roly sú dôležitejšie než správanie.

Programátor

Rola programátora vplýva na výkonnosť tímu najmä pomocou spoločenskej interakcie, teda dimenzia *extraverzia/introverzia*. Presnejšie, programátor *extrovertného* typu je lepší než *introvert*. Dôvod tkvie v skutočnosti, že aj programátor musí komunikovať

najmä so systémovým analytikom, inými programátormi. V malých tímoch navyše programátor často zastáva aj úlohu systémového analytika, a tak komunikuje s vonkajším okolím.

Rôznorodosť tímu

Pre produktivitu tímu nie je dôležitá len osobnosť jednotlivých členov tímu, ale aj porovnanie typov osobností medzi členmi. Pre zvýšenie produktivity sú v tomto smere významné dichotómie *extraverzia/introverzia* a *zmysly/intuícia*. Čím je väčší rozdiel medzi vedúcim a ostatnými členmi tímu v týchto typoch, tým je vyššia produktivita tímu.

To, či sú si ostatní členovia tímu (okrem vedúceho) typmi osobnosti blízki alebo vzdialení, nie je pri menších tímoch jednoznačné, pretože v počiatočných fázach vývoja projektu, akými sú analýza a návrh, sú vhodnejšie nesúrodé osobnosti, zatiaľ čo pri implementácii má vyššiu produktivitu homogénny tím.

pozícia v tíme	úloha	výkonnosť tímu
vedúci tímu	analýza informácií	intuícia > zmysly
	rozhodovanie	cítenie > myslenie
systémový analytik	rozhodovanie	myslenie > cítenie
programátor	komunikácia s ostatnými	extraverzia > introverzia
rôznorodosť vedúci-člen	komunikácia s ostatnými	extraverzia ~ introverzia
	analýza informácií	intuícia ~ zmysly
rôznorodosť člen-člen	všetky rozmery	nie je dôležité

Tab. 2. Zhrnutie vplyvu typu osobnosti na produktivitu tímu. Operátor > znamená, že typ osobnosti na ľavej strane znamená vyššiu výkonnosť než typ na pravej strane. Operátor ~ znamená, že pravá a ľavá strana by mali byť odlišné.

Záver

Kvalita softvérového produktu, teda výsledku softvérového procesu, závisí od mnohých okolností. Táto práca bližšie rozoberá len jednu z týchto okolností – produktivitu tímu. Pri bližšej analýze tejto oblasti boli porovnané faktory, ktoré za produktivitou stoja. Z týchto faktorov je ešte bližšie popísaný ten najvýznamnejší – schopnosť tímu ako celku spĺňať zadané úlohy. Tento faktor je analyzovaný hlbšie

pri pohľade na tím z psychologického hľadiska. Analyzované boli štandardné roly členov menšieho tímu a ich obsadenie rôznymi typmi osobností podľa typológie osobnosti MBTI s ohľadom na ich vplyv na produktivitu tímu.

Použitá literatúra

21. Corvus intl.: The Source of Software Productivity, 2005. http://www.corvusintl.com/white_papers.htm
22. Stevens, K.T.: The Effects of Roles and Personality Characteristics on Software Development Team Effectiveness. *Dizertačná práca, Virginia Tech University,* 1998.
23. Čakrt, M.: Typologie osobnosti pro manažery. Kdo jsem já, kdo jste vy? *Praha: Management Press*, 2000.
24. White, K. MIS Project Teams: An investigation of cognitive style implications. *MIS Q*. 8, 2 (June 1984) 85–101.
25. Lam, Y.W., Gorla, N.: Who Should Work with Whom? Building Effective Software Project Teams. *Communications of the ACM*, June 2004/Vol. 47, No. 6.

Annotation

Improving software team productivity

This essay deals with software team productivity, mainly with the aim to increase the productivity. In the first part we analyze the impacts and factors that influence the team productivity. Particular impacts are quantitatively compared and the most significant impact – team capability – is in the next part more deeply reviewed. Within the team capability we pay attention to personality setup of a smaller team, where we take into the consideration suitability of various types of personalities for particular positions in the software team. This characteristic is described by Myers-Briggs Type Indicator (MBTI). Part of the essay is also the introduction into the personality psychology and more detailed description of MBTI typology.

Koľko to bude stáť?

Odhady v softvérových projektoch

TOMÁŠ MATÚŠEK

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
matusek01@student.fiit.stuba.sk

Abstrakt. V tvrdej konkurencii vždy zvíťazí ten, kto dokáže najlepšie odhadnúť situáciu a pružne reagovať na podnety z okolia. Zvlášť to platí v oblasti vývoja softvérových produktov. Príspevok stručne definuje základné fázy procesu odhadovania a poukazuje na dôležitosť využitia skúseností z predchádzajúcich projektov pri jeho realizácii. Ďalej sa pokúša rozoznať najdôležitejšie problémy, ktorým musia tvorcovia odhadov čeliť a ponúka k nim možné riešenia. Nakoniec uvádza cenné rady a pripomienky, ktoré majú prispieť k lepšiemu odhadu, a teda aj k úspešnejšiemu projektu.

Úvod

Odhad a naplánovanie času a úsilia potrebného k vývoju softvérového systému patria medzi úlohy s vôbec najvyšším sklonom k chybovosti v oblasti softvérového inžinierstva. Dôvodov, prečo tomu tak je, existuje množstvo, z nich najdôležitejšie sú [26]:

- Nekvalifikovaný odhad: nemôžeme dostatočne odhadnúť niečo, čomu nerozumieme.
- Neuvažovanie prostredia: všetky techniky odhadu je potrebné upraviť podľa podmienok prostredia, v ktorom ich chceme aplikovať.
- Nejasný proces odhadu: musia byť stanovené a dodržiavané presné procedúry odhadu.
- Neschopnosť realizácie odhadu: podmienky stanovené odhadom nesmú byť porušené.
- Nedostatočná flexibilita odhadu: je potrebné si uvedomiť, že softvérový projekt nie je statická entita, a teda sa vyvíja v čase. Rovnako by sa mal meniť aj náš odhad.

Je všeobecne známe, že teória a prax majú od seba na míle ďaleko. Existuje množstvo prístupov alebo metód odhadovania, ktoré sú podrobne vysvetlené v rôznych

odborných prácach a pomerne ľahko dostupné. Cieľom môjho príspevku je skôr upozorniť na problémy, ktoré sa môžu objaviť v rôznych štádiách odhadovania, chyby, ktorým sa je potrebné vyhnúť a ponúknuť nápady a vylepšenia, ktoré prispejú k lepšiemu odhadu a tým aj k úspešnému projektu. Verím, že úsilie mnou vynaložené nebolo zbytočné a prinesie svoje ovocie.

Prehľad činností

Proces odhadovania softvérových projektov pozostáva z týchto štyroch základných krokov [27]:

1. Odhad rozsahu vyvíjaného produktu.
2. Odhad úsilia (v človeko-mesiacoch alebo človeko-hodinách).
3. Plán činností (v kalendárnych mesiacoch).
4. Odhad ceny projektu (v peniazoch).

Každý z krokov si stručne vysvetlíme, aby sme získali prehľad a lepšie dokázali pochopiť a identifikovať možné problémy.

Odhad rozsahu

Presný odhad rozsahu projektu je prvým krokom k úspechu. Zdroj informácií by mal, pokiaľ je to možné, vychádzať z formálnej špecifikácie požiadaviek. V praxi je však obvyklé, že jediné, čo máme, je slovný opis. Napriek tomu sa nesmieme nechať odradiť a riziká vyplývajúce z nepresného opisu zakomponovať do odhadu a oboznámiť s nimi účastníkov projektu. Odhad je treba prehodnotiť okamžite, ako sú k dispozícii presnejšie údaje.

Odhad sa vykonáva dvoma základnými spôsobmi:

1. Analogicky, t.j. vychádza sa z predchádzajúceho projektu, ktorý mal podobné vlastnosti. Projekt sa rozdelí na niekoľko častí, ktorým sa pridelí percentuálne ohodnotenie voči predchádzajúcemu projektu. Celkovo sa dá dospieť k pomerne presným výsledkom za predpokladu, že projekty sú si dostatočne podobné.
2. Použitím algoritmických postupov, ako napr. metóda funkčných bodov (Function points) [27]. Počítajú a vyhodnocujú sa vlastnosti produktu.

Odhad úsilia

Podarilo sa nám vytvoriť a identifikovať predpokladaný rozsah projektu, takže z neho teraz môžeme odvodiť potrebné úsilie, ktoré treba vynaložiť. Táto konverzia je možná len vtedy, ak máme definovaný vývojový cyklus softvéru a procesy s ním súvisiace. V súvislosti s výberom procesov a vhodného vývojového cyklu treba spomenúť fakt, že menšie projekty sa riadia ľahšie ako veľké [28]. Teda je vhodné si projekt postupne rozdeliť na niekoľko samostatných modulov a venovať sa odhadom pre každý z nich samostatne. Výhody vyplývajúce z takéhoto rozhodnutia sú:

- Plány a odhady sú presnejšie.
- Chyby a nedostatky sa hľadajú rýchlejšie.
- Pokrok vo vývoji sa dá jasne sledovať.
- Pracovná morálka je vyššia (každý má pocit, že prispel k pokroku).

Samotný odhad úsilia si vyžaduje identifikovať a kvantifikovať všetky aktivity potrebné na vytvorenie produktu odhadovanej veľkosti. Na výber máme z dvoch alternatív:

1. Na základe záznamov o priebehu predchádzajúceho projektu môžeme jednoducho stanoviť potrebné úsilie pre aktuálny projekt. Avšak len za predpokladu, že záznamy existujú, jedná sa o projekt porovnateľného rozsahu, využívame rovnaký vývojový cyklus a na vývoji sa budú podieľať tí istí ľudia.
2. Ak z nejakého dôvodu nemôžeme použiť prvý spôsob, riešením sú všeobecne akceptované algoritmické prístupy ako COCOMO alebo Putnam metodológia [27], ktoré vychádzajú zo štúdia množstva úspešne skončených projektov.

Plán činností

Vytvorenie plánu v sebe vo všeobecnosti zahŕňa počet ľudí, ktorí budú pracovať na projekte, náplň ich činnosti a čas, kedy prácu na projekte začnú a kedy ju ukončia. Keď máme tieto informácie, je potrebné ich premietnuť do kalendárneho rozpisu. Znovu je najlepšie využiť skúsenosti nadobudnuté z predchádzajúcich projektov.

Odhad ceny

Pri stanovovaní celkovej ceny projektu existuje veľa faktorov, ktoré musíme brať do úvahy. Je potrebné uvažovať náklady na mzdy, hardvér a softvér, priestory, cestovné, školiace kurzy a podobne. Presné stanovenie ceny závisí od finančnej politiky spoločnosti. Najjednoduchšie stanovenie ceny práce je vynásobenie potrebného úsilia hodinovou sadzbou. Samozrejme, takto jednoducho to v praxi nejde, keďže každá úloha v tíme je spravidla inak finančne ohodnotená. Znovu, najlepší spôsob je vychádzať zo skúseností z predchádzajúcich projektov.

Presnosť odhadu a kompromisy

Vždy, keď sa vytvorí odhad si kladieme otázku, do akej miery zodpovedá skutočnosti. Presná odpoveď neexistuje, aspoň nie do ukončenia projektu. Pochopiteľne, každý si želá, aby bol odhad vychádzajúci z dostupných údajov v každej fáze čo najpresnejší. Dôležité je, aby odhad nevytváral falošný dojem istoty.

Čo to znamená? Výsledok odhadu by mal tvoriť nejaký interval hodnôt, ktoré ohraničujú danú oblasť. Čím skúsenejší je tvorca odhadu, tým je interval užší. Nie je správne používať presné čísla, pretože zvádzajú k domnienke, že aj odhad je presný, čo

samozrejme nie je pravda, pretože v praxi je veľmi málo vecí jednoznačných a presných. Presnosť odhadu ďalej ovplyvňujú nasledovné skutočnosti:

- Presnosť údajov používaných pri odhade.
- Presnosť výpočtov pri odhade.
- Miera, s akou sa zhodujú využívané záznamy z predošlých projektov s aktuálnym projektom.
- Predvídateľnosť vývojového procesu.
- Neočakávané okolnosti.

Po vytvorení odhadu sa môžeme pustiť do riešenia ďalšieho problému – ako skombinovať zistené skutočnosti, aby sme uspokojili seba, manažment a zákazníkov. Táto úloha si vyžaduje dobrú znalosť jednotlivých faktorov a vzťahov medzi nimi. Je dôležité mať na pamäti tieto veci:

- Cena produktu a čas vývoja sú nepriamo úmerné. Predĺžením času vývoja je možné ušetriť, riskujeme však nespokojnosť zákazníka. Je potrebné zvoliť vhodný kompromis, t.j. zistiť, aké omeškanie je pre zákazníka ešte prípustné.
- Urýchliť vývoj sa dá len tromi spôsobmi. Znížením funkcionality systému, zvýšením počtu pracovníkov (ale len pri činnostiach, ktoré je možné vykonávať paralelne) alebo predĺžením pracovnej doby. Funkcionalita by nám neprešla u zákazníkov, takže do úvahy prichádzajú len druhé dve možnosti, čo má však za následok nárast ceny. Navyše, väčší počet ľudí nemusí znamenať viac času. Pri predĺžení pracovnej doby síce produktivita dočasne stúpne, ale po čase klesne, keďže ľudia budú viac vyčerpaní.
- Pre každý projekt existuje tzv. minimálny možný časový plán, ktorý sa treba snažiť nájsť.

Je zaujímavé pozorovať dôsledky niektorých kompromisov na konečnú cenu produktu. Ako príklad je možné uviesť projekt, kde rozdiel medzi najkratším a nominálnym časovým plánom je iba 2 mesiace., ale na to treba zvýšiť počet ľudí o 10 a celkové náklady vzrastú skoro o 870.000$ [27].

Nie v každom projekte je takýto drastický rozdiel medzi najkratším a nominálnym časovým plánom, ale vzťah medzi cenou a počtom zamestnancov podlieha pravidlám, ktoré nie je možné ovplyvniť, a preto je potrebné, aby s nimi každý zúčastnený počítal.

Problémy

Hoci je efektívny odhad jednou z najdôležitejších častí tvorby softvérového produktu, sú s ním najväčšie problémy. Prečo je to také ťažké? Nasledujúce dôvody nám na túto otázku odpovedia [27].

- Odhadnúť rozsah projektu predstavuje veľmi ťažkú úlohu, ktorá sa zvyčajne opomína a prechádza sa priamo k tvorbe rozvrhu. Samozrejme, je to

nesprávny prístup, pretože keď sa nezamyslíme nad tým, čo máme za úlohu vytvoriť, asi sa nám nepodarí vymyslieť dobrý rozvrh a tým pádom úspešne skončiť projekt.

- Zákazníci si zvyčajne neuvedomujú, že vývoj softvéru je proces neustáleho vylepšovania a zmeny a odhady vytvorené na začiatku procesu sú pomerne nepresné. Aj dobré odhady sú iba odhady so svojou dávkou neurčitosti, a predsa sú často pokladané za istotu. Riešením problému by mohlo byť vytváranie odhadov ako intervalov určitej šírky namiesto konkrétnych dátumov. Je treba si dať pozor, aby interval nebol príliš úzky, lebo tým sa nič nerieši. Ďalším možným východiskom je vyjadrovať sa o termínoch s určitou pravdepodobnosťou, napr. je na 80 % isté, že to stihnem do určitého dátumu.
- Spoločnosti zvyčajne nezaznamenávajú údaje o vyvíjaných projektoch, pritom práve tieto významne prispievajú k úspešným odhadom. Je preto potrebné, aby sa zaviedli nejaké metriky, na základe ktorých sa môžu údaje vyhodnocovať.
- Často je ťažké dohodnúť sa na reálnom termíne so zákazníkom alebo manažmentom. Každý chce mať všetko hotové čo najskôr, avšak existuje minimálna možná doba, pod ktorú sa čas vývoja projektu nedá stlačiť. Treba dobre zvážiť, čo je a nie je možné a za akú cenu a neustúpiť pod tlakom zákazníka. Zázraky sa síce stávajú, ale nie tak často, aby sa na ne dalo spoliehať.

Špecifické prípady

Malé projekty

Veľa ľudí pracuje na malých projektoch, ktoré sú definované malým počtom účastníkov (jeden až dvaja ľudia) a krátkym časovým rozsahom (do 6 mesiacov) [27]. Algoritmické postupy založené na veľkých projektoch sa na ne dajú veľmi ťažko aplikovať. Odhady v malých projektoch sú vysoko závislé na vlastnostiach individuálnych riešiteľov, a preto je najlepšie, aby ich vykonávali sami.

Projekty v neznámom prostredí

Čoraz častejšie sa stáva, že riešime projekt z oblasti, v ktorej nikto z tímu nemá žiadne skúsenosti. Takéto projekty so sebou nesú vysokú mieru rizík a je potrebné postupovať mimoriadne opatrne. Riziká treba dôkladne identifikovať a oboznámiť s nimi zákazníkov. Rovnako je nevyhnutné sa vyhnúť záväzkom v podobe pevných termínov dokončenia. Po každom zlepšení orientácie v problémovom prostredí je vhodné prehodnotiť odhad.

Výber vývojového cyklu projektu je často kľúčový krok pre úspech softvéru. Pri projektoch s vysokou mierou neurčitosti sú vhodné najmä iteratívny a špirálový

model, keďže vývoj sa vykonáva po kúskoch a v každej fáze dochádza k identifikácii rizík a je možné prehodnotiť odhad. Naopak, klasický vodopádový model je z jasných príčin nevhodný.

Tipy a vylepšenia

Nasledujúce zásady majú za úlohu vylepšiť proces odhadovania v softvérových projektoch.

- Na realizáciu odhadu si treba rezervovať dostatok času. Neuvážené odhady vedú k zvýšeniu rizík a v konečnom dôsledku k zlyhaniu projektu.
- Všade, kde je to možné, sa využívajú údaje získané z prechádzajúcich projektov.
- Odhady by mali vykonávať ľudia, ktorí v danej oblasti pracujú. Inak sa zvyšuje nepresnosť.
- Nástroje na tvorbu odhadov predstavujú veľkú pomoc. Uľahčujú samotný odhad a starajú sa o to, aby sa na nič nezabudlo.
- Každý odhad vykonáva viacero ľudí a používajú sa rôzne metódy. Porovnaním výsledkov sa dospeje buď k ich konvergencii, t.j. odhad je pravdepodobne dobrý alebo k rozptylu, čiže asi sme niečo prehliadli a je potrebné lepšie pochopiť skúmaný problém.
- Počas životného cyklu projektu je vhodné vykonávať odhady pravidelne. Spolu s produktom sa vyvíjajú aj odhady, ktoré sa postupne blížia k skutočnej hodnote.
- Štandardizovaný proces odhadu, ktorý každý chápe a dodržiava, znamená lepší prehľad a efektívnejšie rozdelenie úsilia.
- Časť úsilia je potrebné venovať aj vylepšovaniu procesu odhadu. Po každom ukončenom projekte sa vyhodnotia výsledky, zistia sa chyby a nedostatky a vykoná sa prípadná náprava.

Zhrnutie

Ľudia sa odjakživa snažili nájsť univerzálne riešenie na všetky problémy. Samozrejme, nikdy neuspeli, pretože svet nie je čiernobiely ani statický, mení sa z minúty na minútu a nie je možné eliminovať neurčitosť, ktorá nás obklopuje.

Neexistuje dokonalý odhad ani žiadna metóda, ktorá by nám ho priblížila. Žiaden algoritmus nikdy nezaručí úspech, nech je akokoľvek skvelý. Preto jediné, čo nám ostáva, je učiť sa a získavať cenné skúsenosti.

Človek sa síce najlepšie učí na vlastných chybách, ale tie so sebou nesú skoro vždy negatívne dôsledky. Kto sa im chce vyhnúť, používa metódu možno menej účinnú, zato bezpečnejšiu. Učí sa na chybách a skúsenostiach druhých. A presne o to som sa pokúsil v mojom príspevku. Priblížiť problematiku odhadovania a upozorniť na chyby a skúsenosti iných. Dúfam, že sa mi to podarilo.

Použitá literatúra

26. Fairley, R.E.: Recent advances in software estimation techniques. *Proceedings of the 14th international conference on Software engineering.* ACM Press, 1992.
27. Peters, K.: Software project estimation, Software Productivity Center, 1999.
28. Rettig, M, Simons, G.: A project planning and development process for small teams. *Communications of the ACM,* Vol. 36, No. 10 (1993).

Annotation

How much?

In today's ruthless competition the winner is always the one who makes the best estimate of the situation and reacts accordingly. This is also true for software engineering. This paper briefly defines basic phases of estimation process denoting the importance of reusing historical data from similar projects. It also tries to identify serious problems, which the estimators must face and proposes alternative solutions. Finally, it presents valuable tips and suggestions, which contribute to better estimates and eventually, successful project.

Chyba a manažment softvérového projektu

Nerozlučiteľný spoločník

TOMÁŠ MINČEFF

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
minceff01@student.fiit.stuba.sk

Abstrakt. Chyba je nerozlučným partnerom akéhokoľvek softvérového projektu. Tento stav je zapríčinený viacerými faktormi. Asi najdôležitejším je osobitosť vývoja softvéru a jeho vlastnosti. Pozrieme sa na tvorcov chýb a ich skúsenosti. Popíšeme chyby z hľadiska ich viditeľnosti a následkov, ktoré z nich vyplývajú. Testovanie patrí k oblastiam, ktoré sa často podceňujú, a preto sa zameriame na dôvody tohto stavu a vzťah testovača a programátora. S chybami priamo súvisí spoľahlivosť softvérového produktu. Ako jednému zo základných modelov správania sa chyby sa budeme venovať modelu Musa. Nakoniec si porovnáme náklady, ktoré platíme za chyby a ako sa mení ich cena v závislosti od viacerých faktorov.

Úvod

Keď sa používatelia rozprávajú o softvéri, s veľkou pravdepodobnosťou sa zaoberajú jeho nedostatkami a najmä chybami. Dôvodom je skutočnosť, že chyby im spôsobujú nejakú škodu, či už ide o čas strávený napr. pri opakovanom zadávaní údajov alebo stratu finančných prostriedkov. Používatelia sa často bezvýhradne spoliehajú na softvér, pretože im uľahčuje prácu. Preto je kvalita a spoľahlivosť softvéru posudzovaná v prvom rade podľa výskytu chýb.

Ani po takmer polstoročí vývoja softvérových produktov neexistuje žiadny postup alebo metodika, ktorá by viedla k bezchybnému softvéru. V tomto príspevku sa pozrieme na príčiny tohto stavu, ako sa postaviť k chybám a možnostiam eliminácie chýb. Neobídeme ani cenu, ktorú platíme za chyby.

Chyba a manažment

Ak by sme chceli explicitne vyjadriť vzťah medzi chybou a manažmentom softvérového projektu, potom cieľom manažmentu projektu je okrem vytvorenia samotného produktu snaha minimalizovať výskyt chýb a zabezpečiť spoľahlivosť

a kvalitu softvéru. Výskyt chýb u používateľa spôsobuje nielen stratu finančných prostriedkov potrebných na ich odstránenie, ale môže poškodiť dobré meno softvérového manažéra a celej spoločnosti.

Chyby nemusia priamo súvisieť len s používateľmi. Niektoré chyby dokonca môžu spôsobiť problémy samotnému dodávateľovi, napr. vtedy ak nie je zabezpečená dostatočná modifikovateľnosť. Z toho dôvodu je potrebné riadiť vývoj takým spôsobom, aby sa chyby v dostatočnom množstve odstraňovali ešte pred uvedením softvérového produktu na trh.

Osobitosť vývoja softvérového produktu

Vývoj softvérového produktu sa výrazne líši od ostatných výrobných činností. V bežných inžinierskych odvetviach, ako napr. stavebníctvo, nie je predstaviteľné, aby sa nejaký projekt dokončil a potom sa otestoval, či nezlyhá a splní svoje funkcie. Hlavnou príčinou je zničenie takmer všetkých zdrojov, ktoré sa použili na jeho vytvorenie. Ďalej ich nie je možné použiť.

Softvérový projekt sa skladá z veľkého množstva netriviálnych modulov, ktoré je možné znovu použiť. V prípade výskytu chyby vo veľkom množstve prípadov stačí upraviť len niektoré jeho časti. Len v malom percente prípadov sa moduly vyradia z činnosti a začnú sa budovať odznovu.

Je možné vytvoriť projekt bez chýb? Takáto možnosť existuje iba v prípade malých systémov, resp. programov, ktorých rozsah je niekoľko funkcií, prípadne tried bez zložitých závislostí. Príkladom je implementácia triviálnych algoritmov, ako sú matematické výpočty, pri ktorých je možné preukázať jeho správnosť.

Autori chýb

Ako povedal Mark Paulk z Univerzity Carnegie Mellon: „Základným problémom s kvalitou softvéru je to, že programátori robia chyby.“ A pritom je úplne jedno, či ide o skúseného programátora alebo o programátora-začiatočníka.

Jedným z hlavných dôvodov, prečo aj skúsený programátor vytvára chyby, je skutočnosť, že je nasadzovaný na riešenie komplikovanejších úloh prinášajúcich so sebou vyššiu zložitosť. Z toho vyplývajú aj vyššie nároky na odhalenie ich príčin ako chyby spôsobené len preklepmi menej skúsených kolegov [29].

No nielen programátori sú zodpovední za chyby, ale aj analytici, návrhári, dokonca aj samotní zákazníci. Mohli by sme povedať, že hlavnou príčinou sú nedostatky v komunikácií a neporozumenie dokumentácii. A to už od samotného špecifikovania požiadaviek. Následkom toho zákazník nedostane to, čo chcel a považuje to samozrejme za chybu. Pritom môžu byť splnené všetky požiadavky vyplývajúce zo zmluvných záväzkov.

Viditeľnosť chyby

Chyba sprevádza softvérový projekt počas celého jeho života. Od samotného zadania projektu cez analýzu, návrh, vývoj, údržbu až po vyradenie z činnosti.

Chyby môžeme z hľadiska viditeľnosti rozdeliť na externé a interné [30]. Externá chyba je ľahko odhaliteľná pre bežného používateľa. Zjednodušene program nevykonáva, čo je definované v jeho špecifikácii a je to jasne viditeľné okamžite po jeho vykonaní (resp. nevykonaní).

Interné chyby, často nazývané skrytými chybami, nemajú efekt pri bežných operáciách. Ležia nepovšimnuté a neodhalené testovaním a dozvieme sa o nich pri vzniku neočakávaných výnimiek alebo pri údržbe produktu, kedy sú odhalené programátorom pri opätovnom prechádzaní kódu.

K interným chybám patrí aj nedodržiavanie štandardov formátovania a pomenovania objektov zdrojového kódu. Takéto chyby spôsobujú problémy pre programátora pri budúcej údržbe softvéru.

Ďalšou formou skrytých chýb je definovanie objektov, ktoré nie sú nikde použité, nesúvisiace komentáre, nevhodné použitie dátových štruktúr, výber neefektívnych algoritmov a pod.

Príklad zo školského tímového projektu

Príkladom projektov, ktoré obsahujú skryté chyby, je riešenie témy RoboCup – nové stratégie na predmete Tvorba softvérového systému v tíme. Cieľom projektov je vytvorenie autonómneho hráča, ktorý je schopný samostatne hrať simulačnú futbalovú robotickú ligu. Problém je v náročnosti vývoja takéhoto hráča, preto sa používajú hráči vytvorení v predchádzajúcich ročníkoch a postupne sa vylepšujú. Napriek tomu, že autori deklarujú modularitu a rozšíriteľnosť, prax je úplne iná.

V snahe dokončiť projekt a odladiť hráča pre turnaj sa programátori sústredili na funkcionalitu. Študenti dobre vedia, že sa s týmto zdrojovým textom už viac nestretnú, preto nemajú motiváciu na lepšie dodržiavanie štandardov. V dôsledku toho, každý nasledujúci tím strávi niekoľko dní až týždňov nad analýzou kódu, aby bol schopný pokračovať v práci.

Pre tieto projekty je typické ponechanie pôvodného kódu v zdrojových textoch, aj keď sa už nepoužíva. Ako argument sa uvádza možnosť znovupoužitia. Priznajme si, že miesto na odkladanie myšlienok nie je zdrojový kód, ale jeho dokumentácia.

Ďalším problémom je nedostatočné používanie komentárov a nezahrnutie posledných a podstatných zmien do finálnej dokumentácie. Za takýchto podmienok je odhalenie chyby veľmi zdĺhavé a menej času sa potom môže venovať zlepšovaniu samotného produktu.

Testovanie

Jednou z hlavných príčin zníženia kvality je podcenenie času potrebného na testovanie. V súčasnej dobe sa zákazník snaží minimalizovať čas potrebný na vývoj softvéru.

Pod týmto nátlakom sa snažia manažéri vyhovieť zákazníkom v snahe získať tieto zákazky. Avšak čas na vývoj nie je možné znižovať pod istú hranicu. Stáva sa skôr pravidlom, že manažéri vidia isté rezervy v dokumentovaní a testovaní. Je pravda, že tieto dve činnosti prakticky priamo neovplyvňujú výsledný softvérový produkt. Toto riešenie je ale veľmi krátkozraké.

Obchádzanie testovania má za následok vypustenie chyby zákazníkovi. Keďže firma poskytuje údržbu pre softvérový produkt, chyba sa vráti naspäť ako reklamácia. Takáto chyba sa napokon musí opraviť. Rozdiel je ale v nákladoch, ktoré v tomto prípade zahŕňajú okrem samotnej opravy aj manažment zmien, manažment verzií a prípadne aj konfigurácií. Tiež si to vyžaduje návrat programátora k staršiemu kódu.

Nedostatočná dokumentácia spôsobuje problémy najmä v etape údržby softvéru. Základným problémom je to, že znalosť systému je uložená iba v mysliach pôvodných autorov. Z toho vyplýva zvýšená komunikácia medzi členmi tímu a ťažšie zaškolenie nových pracovníkov. Takáto situácia je živnou pôdou pre vytváranie nových chýb.

Ďalším možným rizikom je nedostatok pracovníkov vykonávajúcich testovanie a tiež odďaľovanie postúpenia nového kódu na testovanie. V takejto situácií sa môže stať, že vo veľmi krátkom čase je potrebné otestovať príliš veľa funkcionality softvéru. Dôsledkom toho sa stráca podstata testovania, ktorou je odhalenie čo najväčšieho počtu chýb a tým zabezpečiť lepšiu spoľahlivosť. Namiesto toho slúži testovanie len ako súčasť schvaľovacieho procesu.

Netreba podceňovať ani automatizované testovanie, ktorého úloha sa výrazne zvyšuje pri údržbe softvéru. Hlavným dôvodom jeho používania je dosiahnutie vysokej kvality softvéru. Aj pri oprave chýb sa do zdrojového kódu prinášajú nové chyby, ktoré môžu spôsobiť dokonca aj väčšie škody, ako by spôsobila pôvodná chyba.

Úloha testovačov

Okrem samotného času na testovanie hrajú dôležitú úlohu testovači. Táto pozícia sa dlhodobo podceňuje, dokonca títo členovia tímu bývajú považovaní za menejcenných. Je to spôsobené možno aj nižšou kvalifikáciou, najmä v prípade, ak ide o tzv. používateľské testovanie.

Prečo nie sú obľúbení? Ich prácou je hodnotenie softvéru a teda aj samotných vývojárov. Či si to už vývojári uvedomujú alebo nie, k svojmu programu majú istý vzťah. Ak sa hodnotí ich program, potom sú hodnotení aj oni sami. Určite nie je nikomu príjemné počúvať o vlastných chybách. Ak zamestnanca hodnotí riadiaci pracovník, ten ho môže motivovať napr. finančne za dobre odvedenú prácu, prípadne naopak. Vždy je tu možnosť odmeny. Testovač nemá ako motivovať autora kódu a stáva sa tak len „poslom zlých správ“.

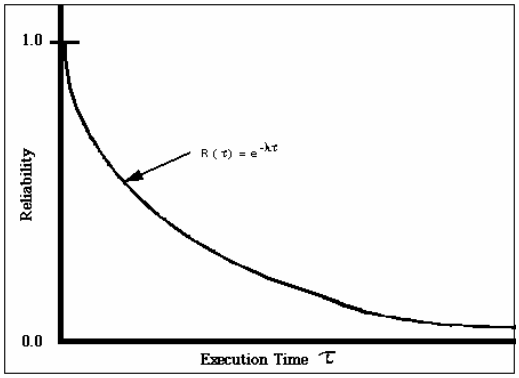
Na druhej strane túto profesiu stále častejšie vykonávajú skúsení programátori, ktorých už vývoj samotných aplikácií omrzel. Nepozerajú sa na softvér ako používatelia, ale ako nezávislí programátori. Majú skúsenosti s vlastnými chybami, poznajú kritické miesta systémov, a preto majú lepšie možnosti odhaliť chyby. Nachádzanie nových chýb sa pre nich stáva „bojovou“ úlohou a odhalenie chyby odmenou.

Spoľahlivosť

Softvérová spoľahlivosť je definovaná ako pravdepodobnosť, že softvér nezapríčiní zlyhanie systému počas špecifikovaného časového úseku pri špecifikovaných podmienkach. Spoľahlivosť je jedna z najstarších metrík používaných pri hodnotení softvéru. Na jej výpočet sa pozrieme pomocou modelovania správania sa chýb.

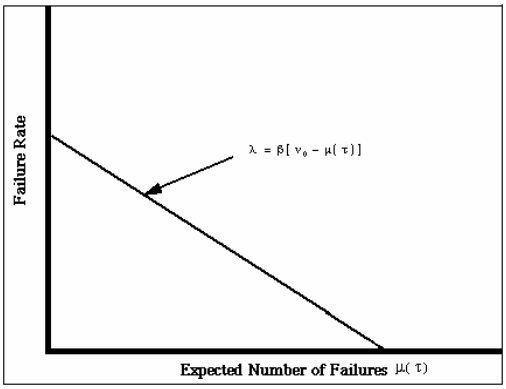
Musa Basic Execution Time Model

Tento model [31] opisuje správanie sa chýb v systéme počas testovania. Pri tomto modeli sa rozlišuje medzi chybou a zlyhaním. Zlyhanie nastáva počas vykonávania programu. Chyba je nedostatok zdrojového kódu, ktorý môže spôsobiť jedno alebo viacero zlyhaní. Predpokladajme, že softvérový systém je nasadený v stabilnom prostredí, pričom sa nemenia používatelia ani nároky používateľov na funkcionalitu systému. Systém sa nemení počas prevádzky. Ak sú tieto požiadavky splnené, potom softvér môžeme modelovať ako systém s konštantnou frekvenciou zlyhania λ, takže množstvo chýb priamo úmerne závisí od času používania systému. Pravdepodobnosť $R(\tau)$, že softvér bude pracovať bez zlyhania, je nepriamo úmerná uvažovanému času. Tento vzťah je možné vidieť na obrázku **Obr. 7.**



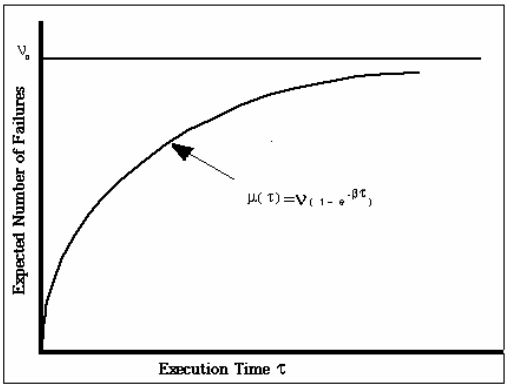
Obr. 7. Závislosť spoľahlivosti od času [31]

Systém sa bude vyvíjať počas testovania za účelom odstránenia pozorovaných chýb, čím sa zvýši jeho spoľahlivosť. Tento model je založený na predpoklade, že všetky chyby prispievajú k chybovosti rovnakým dielom. Takže frekvencia výskytu zlyhaní je klesajúca lineárna funkcia závislá od počtu očakávaných zlyhaní $\mu(\tau)$. Vzťah medzi frekvenciou zlyhania a počtom očakávaných zlyhaní je zachytený na obrázku **Obr. 8.**



Obr. 8. Frekvencia zlyhania v závislosti od očakávaného počtu zlyhaní [31]

Dvoma základnými parametrami tohto modelu sú predpokladaný počet zlyhaní potrebných na odstránenie všetkých chýb – ν_0 a predpokladaný pokles frekvencie zlyhania za zlyhanie – β. Na obrázku **Obr. 9** môžeme vidieť vzťah medzi očakávaným počtom zlyhaní od času.



Obr. 9. Očakávaný počet chýb v závislosti od času [31]

Spoľahlivosť môžeme odhadnúť na základe parametrov, ktoré poznáme z predchádzajúcich etáp vývoja softvéru. Tento odhad je založený na týchto troch parametroch:

- počiatočná frekvencia zlyhaní – λ_0,
- počet chýb na začiatku testovania – ω_0,
- faktor redukcie chýb – *B*.

Frekvencia počtu zlyhaní za chybu ϕ je definovaná ako:

$$\phi = \frac{\lambda_0}{\omega_0} \tag{1}$$

Táto hodnota ϕ je rozdielna od frekvencie zlyhania za zlyhanie β pre vznik chýb počas ladenia programu. Faktor redukcie *B* vyjadruje očakávaný počet odstránených chýb za zlyhanie. Vzťah medzi týmito veličinami sa nachádza v rovniciach 2 a 3.

$$\beta = B\phi = \frac{\lambda_0}{\nu_0} \tag{2}$$

$$\nu_0 = \frac{\omega_0}{B} \tag{3}$$

Keď máme parametre tohto modelu, potom môžeme určiť, ako dlho musí prebiehať testovanie, aby sme dosiahli požadovanú spoľahlivosť. Pre faktor redukcie chýb *B* sa odporúča hodnota 0,955 chyby za zlyhanie.

Náklady na chyby

Už viackrát sme spomínali vzťah medzi chybou a cenou, ktorú za ňu skôr či neskôr zaplatíme. Táto cena závisí od:

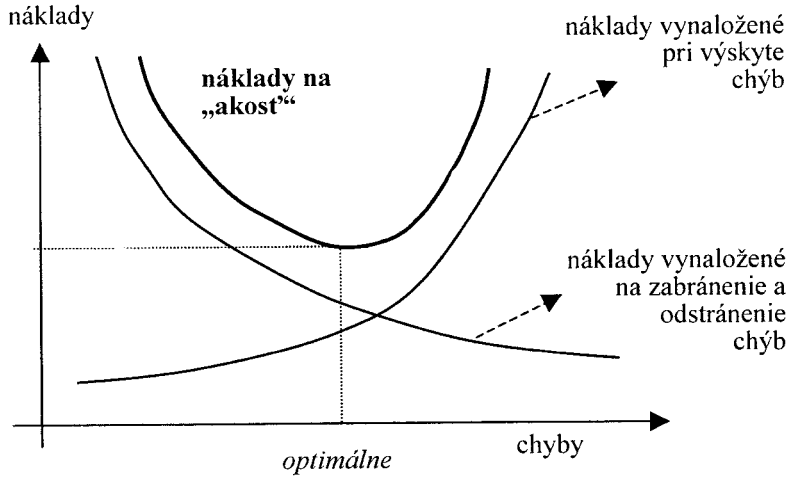
- etapy životného cyklu programu, v ktorom táto chyba vznikla,
- času potrebného na jej odhalenie.

Chyba, ktorá vznikne pri návrhu, môže razantne ovplyvniť nasledujúce etapy projektu najmä v prípade, ak bude potrebné prepracovať doteraz vytvorené dielo. Chyba vytvorená zväčša počas implementácie môže ovplyvňovať produkt ako celok, ale jej odstránenie je zvyčajne iba predmetom jedného modulu.

Čas potrebný na odhalenie chyby opäť spôsobuje zvyšujúce sa náklady na jej odstránenie. Príčinou je napríklad to, že programátor sa venuje modulu, v ktorom nastala chyba, len určitý čas a potom prechádza na riešenie ďalších problémov. Po istom čase si už programátor nepamätá všetky podrobnosti o danom module a je donútený si ich občerstviť, čo stojí nejaký ten čas.

Ďalším aspektom, ktorý treba brať do úvahy, je cena opatrení, ktoré sme vynaložili na elimináciu chýb a efekt, ktorý tieto opatrenia priniesli. Tieto náklady rastú so stúpajúcimi požiadavkami na bezpečnosť, pričom relatívne množstvo odhalených chýb s rastúcimi prostriedkami klesá.

Našou snahou je použitie optimálnych nákladov za podmienky, že zákazník si ich neurčil v zmluve. Optimálne náklady si môžeme odvodiť z grafu na obrázku **Obr. 10** ako maximálny počet odstránených chýb za minimálne investované prostriedky.



Obr. 10. Určenie optimálnych nákladov na kvalitu [32]

Zhodnotenie

Chyba vo všetkých svojich podobách ovplyvňuje nielen naše životy, ale aj softvérové projekty. Aj keď neexistuje žiadny recept na ich odstránenie, nie je vhodné aby sme sa ich báli. Chyby musíme brať ako súčasť riešenia a vytvoriť predpoklady pre ich primerané riadenie s cieľom uspokojiť potreby nielen používateľa, ale aj vývojových pracovníkov, či už ide o programátorov alebo testovačov.

Použitá literatúra

29. Holzmann, G. J.: The Logic of Bugs, ACM SIGSOFT Software Engineering Notes, Vol. 27, No. 6 (2002), 81 – 87.
30. Douce C. R., Layzell P. J.: Evolution and Errors: An Empirical Example, Proc. of ICSM, Oxford, UK (1999), 493 – 498
31. Marciniak J., Vienneau R.: Software Engineering Baselines, Rome, 1996 http://www.dacs.dtic.mil/techs/baselines/reliability.html
32. Bieliková M.: Manažment v softvérovom inžinierstve, 1999

Annotation

Fault and Software Project Management

The fault is inseparable partner of every software project. This state is caused by multiple factors. The most important one is distinctiveness of software development and its characters. We look at authors of faults and their experience. Than describe fault visibility and their consequences. Testing is one of domain, which we often downgrade, so we look at that reason and relationships between tester and programmer. The faults are directly connected with reliability of software product. The Musa is one of the main models of simulation faults in software systems. At the end we compare fault cost and relation between this value and some dependent factors.

Vzťah zákazník a vývojár v softvérových projektoch

ANDREJ NECZLI

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

Abstrakt. Súčasné trendy v oblasti vývoja softvéru naznačujú čoraz väčšiu potrebu existencie vzťahu medzi zákazníkom (zadávateľom projektu) a vývojárom daného softvéru. Potreba zdokonalenia a prehĺbenia komunikácie medzi nimi je len logickým vyústením neustáleho konkurenčného boja firiem v snahe vyvinúť softvérový produkt v čo najkratšom čase. Tento článok sa venuje problémom komunikácie medzi oboma subjektami, úrovniam komunikácie medzi nimi, ako aj samotnej úlohe zákazníka pri vývoji softvéru. Esej ďalej pojednáva o agilných metodikách ako novom trende vývoja softvéru a skúma vzťah zákazníka a vývojára pri extrémnom programovaní, hlavnom predstaviteľovi týchto metodík. Na záver upozorňuje na existenciu liniek medzi zákazníkom a vývojárom, ktoré môžu do značnej miery zefektívniť vývoj softvéru aj v budúcnosti.

Úvod

Medzi zákazníkom a vývojárom existuje krehký, dynamický vzťah, meniaci sa časom, požiadavkami na vyvíjaný softvér, ako aj odlišnými pohľadmi vývojára na zákazníka.

V takomto vzťahu zákazník nemôže mať len pasívnu úlohu na projekte, ale mal by byť aktívnou súčasťou v procese vývoja. Obaja, tak zákazník ako aj vývojár, si musia neustále overovať, či sa správne chápu a skúmať, či všetko podstatné bolo zodpovedané.

Proces komunikácie medzi týmito dvoma subjektami je dosť zložitý a skrýva v sebe mnoho úskalí, ktoré možno prekonať len výberom vhodných liniek medzi zákazníkom a vývojárom spomenutých v závere tejto eseje. Najprv sa však budeme venovať problémom komunikácie, úrovňami komunikácie a predovšetkým využitiu komunikácie pri použití nových vývojových agilných metodík s dôrazom na extrémne programovanie.

Problémy komunikácie so zákazníkom

Okolo každého projektu dochádza z času na čas k ostrejšej diskusii medzi zadávateľom projektu (zákazníkom) a vývojovým tímom. Obe strany by sa mali snažiť riešiť podobné situácie dohodou založenou na racionálnej analýze. Ale v niektorých prípadoch môže vývoj vyústiť do situácie, keď zákazník vyjadruje značnú mieru nespokojnosti a vyhráža sa podniknutím právnych krokov, ako sú penále alebo vypovedanie zmluvy. Niekedy sa môže dodávateľ iba domnievať, čo takúto reakciu zo strany zákazníka vyvolalo.

Hlavná prekážka, ktorú si musíme pri komunikácii so zadávateľom projektu (zákazníkom) a budúcimi užívateľmi vždy uvedomiť, spočíva v tom, že ide o špecialistov z iného odboru používajúcich odlišné termíny a de facto hovoriacich iným jazykom. Väčšinou nebudú informačným technológiám rozumieť do detailov, a preto je treba sa pri jednaní s nimi vyhnúť podrobnostiam. Užívateľa (zákazníka) zaujíma predovšetkým to, čo vidí na obrazovke a ako bude s aplikáciou pracovať. Aj keď má zadávajúca firma svojho správcu informačného systému, musíme počítať s tým, že ani on nebude orientovaný na spomínaný iný typ informácií, aký používa práve vývojový tím. Pre neho bude dôležité skôr to, aký hardvér a softvér aplikácia vyžaduje, ako má celý systém nastaviť a či aplikácia dodržuje všetky bezpečnostné pravidlá. Preto nemôže vývojový tím očakávať, že mu zákazník nejakým tvorivým spôsobom pomôže pri riešení technických problémov. [37]

Úrovne komunikácie so zákazníkom

Komunikácia so zákazníkom prebieha na niekoľkých úrovniach. Pred zahájením projektu je potrebné vyriešiť všetky obchodné a právne otázky. V rámci projektu sa vymenuje takzvaný projektový výbor, čo je grémium pre formálnu komunikáciu zúčastnených strán. Jeho členmi bývajú projektoví manažéri a niekto z ich nadriadených. Obvykle sa schádzajú pravidelne k posúdeniu priebehu projektu, na mimoriadnych stretnutiach sa preberajú prípadné problémy, ktoré sa na úrovni vedúcich projektov vyriešiť nepodarilo.

Najčastejšia komunikácia prebieha medzi vedúcimi projektu oboch strán – na strane zákazníka môže byť jeho skutočná funkcia iná, ale vždy musí existovať niekto, kto má projekt v zadávajúcej organizácii na starosti, napríklad správca alebo manažér informačných systémov. Komunikácia na tejto úrovni býva niekedy formálna, napríklad keď si obe strany vymieňajú zoznam požiadaviek alebo nájdených chýb, ale väčšinou ide o bežnú výmenu informácií o tom, ako sa projekt vyvíja, aké sú názory koncových užívateľov, kedy prebehnú konzultácie špecialistov a podobne. Je prirodzené, že občas dochádza ku sporom, ale obaja manažéri by sa mali snažiť nakoniec nejakým spôsobom dohodnúť. Pokiaľ sa to nepodarí, rieši problém projektový výbor, ale je veľmi nebezpečné, keď k tomu dochádza často a prirodzená komunikácia sa zmení na formálnu výmenu dokumentov.

A na tej poslednej úrovni si informácie vymieňajú odborní pracovníci vo väčšine prípadov úplne neformálne a nepredvídateľne. Niekedy sa projektoví manažéri snažia túto komunikáciu obmedziť, vynára sa však otázka, nakoľko je to účinné. Väčšinou ide o krátke telefonické rozhovory alebo výmenu e-mailov, keď obidvaja pracovníci diskutujú o nejakom detaile, ktorý nie je pre ďalší priebeh projektu významný.

Písomné zápisy komunikácie

Z každého jednania, ktoré môže mať na ďalší priebeh projektu nejaký vplyv, aj keď ide iba o krátky telefonický rozhovor, je potrebné vytvoriť zodpovedajúci písomný zápis a zaslať ho každému, koho sa týka. Pokiaľ sa dvaja technici dohodnú na tom, že na obrazovke číslo 34 bude nové textové pole, stačí poslať krátky e-mail vedúcim projektu a tomu členovi tímu, ktorý má na starosti užívateľské rozhranie. Z jednania projektového tímu bude naopak vytvorený formálny zápis obsahujúci závery všetkých preberaných bodov, ktoré obdržia všetci jeho členovia.

Pokiaľ dôsledky jednania vyžadujú zmenu v oblasti, ktorá je opísaná niektorým dokumentom, ako je napríklad zoznam požiadaviek, schéma databázy, návrh užívateľského rozhrania a podobne, musí zápis dostať taktiež pracovník, ktorý je za udržiavanie tohto dokumentu zodpovedný, aby mohol čo najskôr tieto úpravy do dokumentu zaznamenať. Mal by taktiež uviesť odkaz na jednanie, na ktorého základe sa zmena prevádza, aby v budúcnosti bolo možné vystopovať jej príčinu.

Úloha zákazníka pri vývoji softvéru

Klasické techniky softvérového inžinierstva sa týkajú situácie, ktorá bola platná pred desiatimi, dvadsiatimi rokmi – tejto situácii plne vyhovujú a jej požiadavky bezvýhradne napĺňajú. Softvér bol vyvíjaný skôr väčšími, organizovanými a stabilnými tímami. Dôraz, ktorý sa kládol na kvalitu, bol ešte stále spôsobený obavou z toho, aby sa neopakovalo čosi ako softvérová kríza, ktorá spomalila vývoj programov pred nástupom softvérového inžinierstva. Zákazník si vtedy radšej počkal trochu dlhšie (nič iné mu ani neostávalo, lebo konkurencie bolo málo), ale očakával zato kvalitný výsledok.

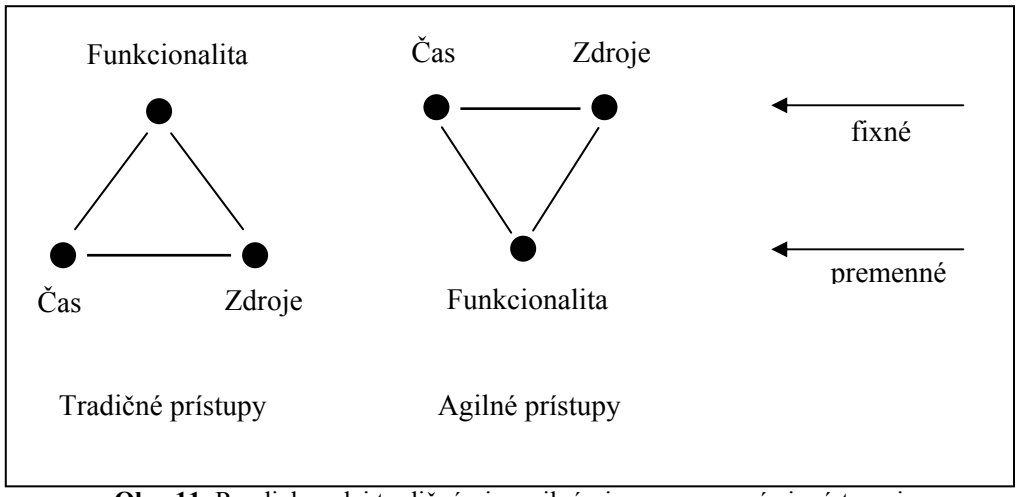
Svet sa však zmenil, a to hneď v niekoľkých smeroch. Predovšetkým, zákazník síce stále očakáva kvalitu, ale už odmieta na ňu dlho čakať. V prípade internetových projektov naviac zákazník často nie je dopredu ani definovaný. Budúci užívatelia nevedia (nebudú vedieť), čo má systém poskytovať. Aj u iných projektov sa však v praxi objavuje zaujímavý paradox – až pozoruhodne často je zákazník spokojný s relatívne nepoužiteľným produktom. Môže to byť spôsobené okrem iného tým, že pri samotnom zadaní projektu ešte ani presne nevie, čo vlastne od produktu požaduje. Agilné metodiky tento problém riešia typicky omnoho užším spojením zákazníka s vývojovým tímom.

Druhým faktom je, že programátorské firmy sa zmenšujú, pretože vzniká obrovské množstvo menších programov. „Firma“ je tak často zložená z piatich ľudí, výnimkou nie sú ani tímy, pozostávajúce z dvoch alebo troch programátorov.

Nakoniec, najdôležitejším parametrom pri vývoji softvéru je rýchlosť. Hlavným cieľom predsa je, aby bola aplikácia čo najskôr nasadená a mohla si získavať nových užívateľov. U internetových aplikácií pritom platí, že funkčnosť možno pridávať „za behu“, teda že je možné spustiť čo najskôr obmedzenú (avšak plne fungujúcu) verziu aplikácie a ďalšie požiadavky na funkcionalitu realizovať postupom času. Dnešné požiadavky na rýchly a flexibilný vývoj aplikácií sa snažia riešiť už vyššie spomenuté agilné metodiky, ktorým bude venovaná nasledujúca časť.

Agilné metodológie – základné princípy

Rozdiel medzi tradičnými prístupmi a agilnými metodikami možno najlepšie pozorovať na obrázku **Obr. 11,** ktorý porovnáva chápanie základných premenných pri vývoji softvéru:



Obr. 11. Rozdiel medzi tradičnými a agilnými programovacími prístupmi

Tradičné metodiky (znázornené ľavým trojuholníkom) vychádzajú z potreby naplniť za každú cenu dokument *Špecifikácia požiadaviek.* Požiadavky – teda funkcionalita – sú fixné, pričom v úlohe premenných vystupujú čas a potrebné zdroje. Pri tradičnom vývoji bolo teda jasné, čo bude program robiť, avšak ťažko sa odhadovalo, koľko to bude stáť a ako dlho to bude trvať.

Agilné prístupy oproti tomu považujú čas a zdroje za fixné (stanovené na začiatku projektu zadávateľom) a mení sa (resp. priebežne sa prispôsobuje) funkcionalita (požiadavky). Na začiatku projektu je teda presne stanovený najdlhší možný čas vývoja a najvyššie možné náklady. Do týchto atribútov sa potom vývoj musí vojsť. V priebehu

prác na projekte tím komunikuje so zákazníkom a priebežne prehodnocuje priority. Zákazník by v ideálnom prípade mal byť členom vývojového tímu, mal by komunikovať s vývojovým tímom, participovať na návrhu a spolurozhodovať o testoch. Agilné metodiky vyžadujú oveľa menší objem formálnych a byrokratických artefaktov. Vychádzajú pritom z pevného presvedčenia, že *základným, jednoznačným, exaktným, nespochybniteľným a vlastne jediným spoľahlivým nositeľom informácie je zdrojový kód.* Uvedená myšlienka je základným filozofickým východiskom aj pre extrémne programovanie (Extreme *Programming,* XP), pravdepodobne najrozšírenejšiu a najznámejšiu agilnú metodiku. Preto si ju prezrime trochu bližšie, hlavne v súvislosti s rolou zákazníka pri XP. [35]

Vzťah zákazníka a vývojára pri extrémnom programovaní

Extrémne programovanie je metodika vhodná pre malé až stredné tímy (zvyčajne sa uvádza počet dvoch až desiatich programátorov), ktorí sa pri vývoji musia vyrovnať s rýchle sa meniacim alebo nejasným zadaním.

Zákazník tu má pomerne dôležitú pozíciu, pretože hneď po programátorovi (vývojárovi) zastáva druhú kľúčovú rolu. Programátor síce dokáže programovať, ale nevie čo. Zákazník to síce neovláda, ale zato vie, čo je potrebné programovať. Preto je zákazník neoddeliteľnou súčasťou vývoja softvéru podľa XP. Na zákazníka sú však taktiež kladené niektoré neštandardné požiadavky: musí sa zžiť s niektorými zásadami XP, ako napríklad naučiť sa písať zadania a testy funkcionality a v neposlednom rade aj prijať možnosť ovplyvňovať projekt a zodpovedať zaň. Skutočný zákazník musí sedávať s vývojovým tímom, byť k dispozícii, odpovedať na otázky, riešiť konflikty a stanovovať drobné priority. Musí to byť niekto, kto bude skutočne používať systém po zavedení do prevádzky.

Formovanie vzťahu zákazník-vývojár pomocou *Pattern Language*

Svet možno vnímať ako systém zložený z malých obrazcov (patterns), pričom nikdy nemožno spozorovať, aby bol čo i len jeden obrazec (pattern) v izolácii. Vždy by mal byť súčasťou nejakej zbierky obrazcov alebo jazyka obrazcov (pattern *language)* [33]. Autor tejto myšlienky John Muir takýto jazyk pre interakciu zákazníka s vývojárom vytvoril. Podľa neho mnohé z týchto obrazcov, ak nie všetky, by sa mali dať aplikovať v agilnom vývojovom prostredí, aj keď neboli napísané vyslovene pre tento účel. Tu sú niektoré z nich (a ďalšie, ktoré sú v príbuzných jazykoch):

- *Ide o vzťahy, nie o predaj:* Rozvíjajte vzťahy so zákazníkom. Sústreďte sa na tento vzťah, nie na aktuálnu transakciu. Využitie: *Pochopenie zákazníka* a Získanie *dôvery.*
- *Pochopenie zákazníka:* Spoznajte zákazníka čo najviac: Využitie: *Efektívne počúvanie, Včasná odpoveď* a Stretnutia *popri stretnutí.*

- *Získanie dôvery:* Každý kontakt so zákazníkom je šancou na získanie dôvery. Využite ju. Využitie: *Efektívne počúvanie, Včasná odpoveď* a Stretnutia *popri stretnutí.*
- *Efektívne počúvanie:* Počúvajte zákazníka so zámerom pochopiť ho. Využitie: *Osobná integrita, Uvedomenie si hraníc, Pomoc zákazníkovi* a Dobré *spôsoby.*
- *Včasná odpoveď:* Ak dostanete od zákazníka nejakú požiadavku, dajte mu vedieť, že ste ju prijali a ako ju mienite riešiť.
- *Stretnutia popri stretnutí:* Príďte na schôdzku dostatočne skoro na to, aby ste sa stretli s ostatnými účastníkmi a strávili s nimi nejaký čas. Po stretnutí venujte trochu času a pohovorte si s ostatnými o bežných záležitostiach.
- *Osobná integrita:* Nezadržiavajte dôležité informácie od zákazníka, ale stanovte si určité hranice.
- *Pomoc zákazníkovi:* Nehádajte sa. Pokúste sa pochopiť, akým spôsobom zákazníkov biznis funguje. Nesnažte sa uchlácholiť zákazníka dávaním sľubov, ktoré nemôžete dodržať. Využitie: *Uvedomenie si hraníc* a Dobré *spôsoby.*
- *Uvedomenie si hraníc:* Berte každú konverzáciu so zákazníkom ako časť rokovania. Nediskutujte o tom, čo nie je súčasťou Vašej zodpovednosti. Využitie: *Dobré spôsoby.*
- *Dobré spôsoby:* Buďte zdvorilý. Oblečte sa adekvátne k tomu, čo očakáva od Vás zákazník. Preukážte rešpekt ku každému, vrátane konkurencie. Buďte opatrní v interakcii s ostatnými pred zákazníkom.

Funkcia liniek medzi zákazníkom a vývojárom

Mnohé z najlepších nápadov pre nové produkty a produktové zlepšenia prichádzajú zo strany zákazníka alebo koncového užívateľa produktu a práve tento fakt si vyžaduje zriaďovanie jednej alebo viacerých liniek medzi zákazníkom a vývojárom (customer-developer *links).* Tieto sú definované ako určité kanály, umožňujúce zákazníkom a vývojárom si vymieňať navzájom informácie. [36]

Dnes je k dispozícii obrovské množstvo liniek, ktoré predstavujú príležitosť ako aj výzvu pre manažérov softvérového vývoja. Príležitosť spočíva v tom, že nebolo nikdy ľahšie získať vstupy od zákazníka než teraz a výzva tkvie v tom, že vývojoví manažéri si v súčasnosti musia vedieť vybrať z veľkého množstva liniek, z ktorých sú mnohé často mätúce. Preto je rozhodnutie o tom, koľko a aké typy liniek sa budú využívať čokoľvek, len nie priamočiara záležitosť. Toto náročné rozhodnutie by mohlo pomôcť uľahčiť vymedzenie hlavných rozdielov medzi nimi, nachádzajúcich sa v tabuľke **Tab. 3**. Rozlišuje sa pritom medzi dvoma vývojárskymi prostrediami, teda či

ide o softvér na zákazku (určený na použitie v rámci zákazníckej firmy) alebo softvérový balík (určený pre komerčné využitie):

Dimenzia vývoja	Softvér na zákazku	Softvérový balík
Cieľ	Softvér, vyvíjaný pre použitie v rámci zákazníckej firmy (zvyčajne nie na predaj)	Softvér, vyvíjaný pre vonkajšie použitie (na predaj)
Moment, kedy je identifikovaná väčšina zákazníkov	Pred zahájením vývoja	Potom, ako sa vývoj ukončí a produkt ide na trh
Počet zákazníkov (organizácií)	Zvyčajne jeden	Veľa
Fyzická vzdialenosť medzi zákazníkom a vývojárom	Zvyčajne malá (zákazníci sídlia v rovnakej budove ako vývojári)	Zvyčajne veľká (zákazníci môžu byť vzdialení od vývojárov aj tisíce kilometrov)
Typ projektu	Projekt nového systému; zlepšovanie údržby	Nové produkty, nové verzie (hlavné a menšie)
Požiadavky na softvérového zákazníka	Užívateľ, koncový užívateľ	Zákazník
Všeobecné ukazovatele úspešnosti	Spokojnosť, priaznivé prijatie	Predaj, podiel na trhu, dobré ohlasy na produkt

Tab. 3. Hlavné rozdiely medzi vývojárskym prostredím softvéru na zákazku a softvérového balíku.

Tabuľka **Tab. 4** potom uvádza príklad, ako môžu byť definované jednotlivé linky medzi zákazníkom a vývojárom vzhľadom na spomenuté vývojové prostredia a ich kombináciu. Inventár týchto liniek tvorí základ koncepcie počítania liniek v rámci daného projektu, pretože čím väčšia je účasť zákazníka na procese vývoja, tým úspešnejší softvérový projekt môže byť.

Linka	Popis	Zvyčajne používané pri: Z=SW na zákazku, B=SW balík
Sprostredkovateľský tím	Sprostredkovaný štruktúrovaný kurz so zákazníkmi, slúžiaci na vylákanie požiadaviek od nich.	Z
MIS sprostredkovateľ	Niekto, kto definuje zákaznícke ciele a potreby pre dizajnérov a vývojárov.	Z
Podporná línia	Oddelenie, ktoré pomáha zákazníkom s každodennými problémami (tzv. zákaznícka podpora, technická podpora, help desk).	Z/B
Prieskum	Anketa vykonaná na vzorke zákazníkov.	Z/B
Prototypovanie užívateľského rozhrania	Zákazníkom je predvedená demo- alebo nejaká ranná verzia, aby sa odhalili problémy s užívateľským rozhraním.	Z/B
Prototypovanie požiadaviek	Zákazníkom je predvedená demo- alebo nejaká ranná verzia, aby sa spresnili požiadavky na systém.	Z/B
Interview	Rozhovor s koncovým užívateľom, otvorený a pološtruktúrovaný.	Z/B

Testovanie	Nové požiadavky a spätná väzba, prameniaca z testovania. Nezahŕňa zisťovanie chýb.	Z/B
E-mail/bulletin board	Dodatočné problémy zákazníkov, otázky a návrhy na bulletin board alebo cez e-mail.	Z/B
Laboratória pre užitočnosť	Špeciálne vybavené laboratória zamerané na identifikáciu vecí, ktoré užívateľ pri práci využíva.	Z/B
Pozorovacie štúdium	Užívatelia sú sledovaní na určité obdobie, aby sa zistilo, čo vlastne robia.	Z/B
Marketing a predaj	Zástupcovia sa pravidelne stretávajú so zákazníkmi (súčasnými aj potenciálnymi), aby si vypočuli ich návrhy a potreby.	B
Užívateľská skupina	Skupiny zákazníkov sa pravidelne schádzajú, aby prediskutovali používanie softvéru a jeho možné zlepšenia.	B
Trade show	Zákazníkom je na trade show predstavený prototyp s cieľom získať od nich spätnú väzbu.	B
Skupina, sústrediaca sa na niečo	Malá skupina zákazníkov a moderátor sa zídu, aby diskutovali o danom softvéri.	B

Tab. 4. Linky medzi zákazníkom a vývojárom (customer-developer *links).*

Záver

Komunikácia medzi zákazníkom a vývojárom je téma na mnoho diskusií. Ja som sa snažil podať náhľad na problematiku hlavne zo strany vývojára, pričom som dal dôraz na využitie komunikácie zákazníka a vývojára hlavne pri nových technikách vývoja softvéru, akými sú agilné metodiky.

Taktiež zaujímavý je pohľad na komunikáciu medzi zákazníkom a vývojárom prostredníctvom takzvaného jazyka obrazcov (pattern *language),* ktorý nachádza využitie nielen v oblasti komunikácie vývojára so zákazníkom, ale môže mať aj všeobecné uplatnenie. Tento jazyk by mal napomáhať do značnej miery pri vytváraní a udržiavaní medziľudských vzťahov ako takých.

No a nakoniec definovanie liniek medzi týmito dvoma subjektami podstatne uľahčí budúci vývoj komunikácie medzi zákazníkom a dodávateľom softvéru a proces vzájomnej výmeny informácií bude omnoho efektívnejší.

Použitá literatúra

33. Fraser, S., Martin, A., Biddle, R., Hussman, D., Miller, G., Poppendieck, M., Rising, L., Striebeck, M.: The role of the Customer in Software Development: The XP Customer – Fad or Fashion? *Companion to the 19th annual ACM SIGPLAN conference on Object-oriented programming systems, languages and applications.* ACM Presss, New York (2004), 148-150.
34. Heiskanen, A., Similä, J.: Gatekeepers in the action structure of software contracting: A case study of the evolution of user-developer relationships. *ACM SIGCPR ComputerPersonnel,* Vol. 14, No. 1-2 (1992) 30-44.
35. Kadlec, V.: *Agilní programování. Metodiky efektivního vývoje software.* Computer Press, Brno, 2004. (CZ)
36. Keil, M., Carmel, E.: Customer-Developer Links in Software Development. *Communications of the ACM,* Vol. 38, No. 5 (1995) 33-44.
37. Paleta, P.: *Co programátory ve škole neučí aneb Softwarové inženýrství v reálné praxi.* Computer Press, Brno, 2003. (CZ)

Annotation

The customer/developer relationship in software projects

The latest trends in software development area suggest even more needs of presence of customer-developer relationship. The need to improve the communication between these two subjects is just a logical ending of resident concurrent struggle of corporations, trying to develop the product in the shortest time. This article is dedicated to communication problems between both of the mentioned subjects, communication levels between them, as well as the role of customer in software development by itself. The essay further deals with the agile methodics as a new trend of software development and explores the customer-developer relationship in extreme programming, the main representative of these techniques. In

conclusion essay urges on existence of the customer-developer links which can in essential way make the software development in the future more effective.

Sledovanie postupu softvérového projektu a manažment

PETER PRIKRYL

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

Abstrakt. Napriek najlepšiemu plánovaniu, účasti najlepšieho tímu a predvídaniu všetkých možných úskalí to vyzerá tak, že softvérové projekty majú dlhoročnú prax vo vyvolávaní nepredvídateľných problémov. Jedna z možností, ako sa môžeme vyhnúť neúspešnému ukončeniu projektu, je sledovanie postupu softvérového projektu. Aby sme vôbec mohli sledovať vývoj a postup v softvérovom projekte, je veľmi dôležité vytvoriť plán. Tiež je potrebné zaviesť vhodné metriky, ktorými budeme merať postup. Metriky by nemali nútiť ľudí dosahovať lepšie výsledky v meraní, ale motivovať ich efektívne dosiahnuť vytýčené ciele projektu. Keď máme plán projektu, je možné pýtať sa, akú časť projektového plánu sa podarilo splniť, odhadovať, koľko práce ešte treba spraviť a porovnávať naplánovaný stav so skutočným stavom. V prípade rozdielov je možné plán skorigovať a nasmerovať úsilie tímu tým správnym smerom.

Úvod

Významným problémom v softvérovom projekte počas fázy vykonávania projektu je jeho kontrola a sledovanie. Napriek najlepšiemu plánovaniu, účasti najlepšieho tímu a predvídaniu všetkých možných úskalí to vyzerá tak, že softvérové projekty majú dlhoročnú prax vo vyvolávaní nepredvídateľných problémov.

Aby sa predišlo možnému neúspechu softvérového projektu, je dôležité sledovať charakteristiky projektu a vývoj týchto charakteristík v čase. Tým, že manažér vidí skutočný stav softvérového projektu, môže v prípade rozdielov medzi plánovaným a skutočným stavom projektu zasiahnuť a usmerniť vývoj ďalšieho postupu v projekte. Vyhodnocovanie vykonanej práce a analýza nákladov a úsilia vynaložených na vykonanú prácu napomáhajú efektívnejšiemu využitiu zdrojov v budúcnosti.

Čo je to postup

Je jednoduché povedať, že niekto pracuje. Ale ako sa dá povedať, že niekto postupuje? Sledovať postup pri jednoduchej práci, ako napríklad maľovanie bytu, je jednoduché. Vždy po vymaľovaní jednej steny alebo miestnosti je postup viditeľný každému. Ale pri zložitejšej práci, akou je napríklad aj tvorba softvérového systému, je ťažšie identifikovať skutočný postup.

Niektoré činnosti vyžadujú premýšľanie a skúmanie, ktoré môže trvať hodiny alebo dni bez viditeľných výsledkov. Iná práca môže zahŕňať toľko rôznych podúloh alebo rozhovorov s ostatnými, že sa nedá žiadnym spôsobom zistiť, ako efektívne je táto práca vykonávaná, alebo či vôbec úsilie vynaložené na túto prácu povedie k postupu v softvérovom projekte. Pri zložitej práci alebo pri práci s veľkým množstvom ľudí je vždy ťažké rozoznať prácu v projekte od postupu v projekte.

Pre softvérový projekt je v prvom rade dôležité vytvoriť plán projektu. Plán nám umožňuje efektívne smerovať svoje úsilie na prácu, ktorú treba vykonať, aby sme splnili vytýčené ciele projektu. Keď máme plán projektu, je možné pýtať sa, akú časť projektového plánu sa podarilo splniť, odhadovať, koľko práce ešte treba spraviť a porovnávať naplánovaný stav so skutočným stavom. V prípade, že dosiahnuté výsledky nezodpovedajú naplánovaným, je však nutné zmeniť plán.

V tímoch, v ktorých je slabé riadenie, vzniká priestor pre to, aby bola vynaložená energia na zbytočnú alebo kontraproduktívnu prácu. Väčšinou je to spôsobené tým, že manažér si to nevšimne, nestará sa, alebo si nenájde čas, aby nasmeroval úsilie ľudí užitočnejším smerom. Keď sa niekto pustí do práce, nezáleží na tom, ako rýchlo v práci postupuje, ak postupuje nesprávnym smerom. Nasmerovanie úsilia môže byť oveľa dôležitejšie než veľkosť samotného vynaloženého úsilia.

Ako určiť postup v softvérovom projekte

Nevyhnutnou súčasťou toho, aby sme mohli určiť, sledovať a kontrolovať postup softvérového projektu, je plánovanie a ohraničenie cieľov, ktoré sa majú prácou v softvérovom projekte dosiahnuť. Ak by som niekomu dal do ruky valček na maľovanie, ukázal mu tri kýble bielej farby a povedal mu, nech vymaľuje celý byt, bolo by jasné, ako má vyzerať postup. Aj pri zložitejšej práci je úloha projektového vedúceho v tíme rovnaká – jeho úlohou je, aby každý vedel, ako má vyzerať výsledok v čase, keď projekt skončí.

To, ako má na konci vyzerať výsledok softvérového projektu, sa môže počas vykonávania projektu s časom meniť a je úlohou projektového vedúceho, aby sa táto predstava menila približne rovnako v celom tíme. Ak v tíme ešte nie je predstava o konečnom výsledku softvérového projektu, prvou úlohou, ktorú môže projektový vedúci v tíme zadať je, aby každý prispel názorom alebo nápadom, ako má konečný výsledok projektu vyzerať.

Ďalšou súčasťou určovania postupu softvérového projektu je dennodenné riadenie tímu správnym smerom. Je úlohou projektového vedúceho tímu, aby neustále

poukazoval na to, ako práca členov tímu prospieva splneniu vytýčených cieľov. Pri každom stretnutí musí projektový vedúci tímu motivovať ľudí tak, aby ich práca posúvala softvérový projekt bližšie k jeho vytýčeným cieľom, a teda aj bližšie k jeho úspešnému ukončeniu.

Projektový vedúci má najlepší prehľad o aktuálnom stave softvérového projektu. O svoj pohľad by sa mal čo najčastejšie deliť s ostatnými členmi tímu. Najlepšie vidí aj to, kedy ľudia svojou prácou postupujú nesprávnym smerom, alebo v horšom prípade znehodnocujú prácu iných členov tímu. Ak sa o svoj pohľad rozdelí s ostatnými členmi tímu, hoci nepozná odpovede na všetky otázky, ľudia jeho názor prijmú a upravia svoju prácu tak, aby maximalizovali postup v softvérovom projekte.

Sledovanie práce v softvérovom projekte

Aby sme mohli sledovať prácu a postup v softvérovom projekte, potrebujeme ich vedieť najprv nejakým spôsobom zmerať. Predstava je taká, že zavedením metriky umožníme ľuďom sústrediť sa na dosiahnutie lepších výsledkov. Efekt takéhoto merania však často býva iný. Ľudia sa sústredia viac na dosiahnutie lepších výsledkov v meraní namiesto dosiahnutia vytýčených cieľov projektu.

V niektorých organizáciách merajú [40]:

- Koľko hodín strávil pracovník v kancelárii.
- Počet napísaných hlásení alebo špecifikácií.
- Počet prezentácií.
- Množstvo telefónnych hovorov.
- Počet napísaných riadkov kódu.
- Počet opravených chýb.
- Množstvo poslaných e-mailov.

Všetky merania využívajúce spomenuté metriky majú pomerne nízku výpovednú hodnotu, pretože zachytávajú iba vykonané aktivity. Napríklad, ak v organizácii, v ktorej pracujem, merajú počet opravených chýb, pre mňa to znamená, že budem opravovať hlavne tie jednoduchšie chyby. Stihnem ich opraviť viac a budem aj lepšie hodnotený. Chyby, ktorých oprava by trvala príliš dlho, v softvérovom projekte zostanú. Podobne aj meranie počtu riadkov v programe vypovedá iba o rozvláčnosti napísaného kódu, ale už nie o jeho kvalite. Každé z týchto meraní má svoje slabé miesta, ktoré sa dajú veľmi ľahko zneužiť. Ale keďže je také jednoduché ich zmerať, často sa v praxi používajú. Projektovému vedúcemu informácie z týchto meraní môžu pomôcť, ale len v prípade, ak sú interpretované s rozumom a k dispozícii sú aj ďalšie informácie.

Sledovanie postupu v softvérovom projekte

Sledovanie postupu v softvérovom projekte už nie je také jednoduché ako sledovanie vykonávania práce. Spôsoby, akými sa meria postup v softvérovom projekte, sú viac subjektívne a vyžadujú viac rozmýšľania. Pre sledovanie postupu v softvérovom projekte je dobré klásť si nasledovné otázky o vykonávanej práci:

- Ku ktorému cieľu prispieva táto práca?
- Ako nám táto práca pomôže dostať sa bližšie k cieľu?
- Aký veľký je to príspevok k dosiahnutiu cieľa? (Dostane nás to bližšie k cieľu o 5 % alebo o 50 %?)
- Je kvalita vykonávanej práce dostatočná?
- Čo nám táto práca umožní ďalej vykonať? Ako nám pomôže v inej práci, ktorú treba tiež vykonať?

Zodpovedanie týchto otázok nám vždy pomôže utvoriť si obraz o postupe v projekte. Samozrejme záleží aj od toho, na akej úrovni podrobnosti sú vytýčené ciele definované. Ak sú ciele definované príliš nepresne, môže byť postupom aj práca, ktorá nemusí priniesť očakávané výsledky. Ak sú však ciele príliš konkrétne a sú definované príliš striktne, nezostáva priestor pre tvorivé riešenie problémov.

Určenie vhodnej úrovne podrobnosti pri definovaní cieľov je umenie a dá sa dobre zvládnuť iba na základe skúseností. Vo všeobecnosti je vhodné určiť ciele so vzrastajúcou úrovňou podrobnosti pre celý projekt, každý tým a ciele pre každého člena tímu.

Pomer práce a postupu

Pri riešení nových problémov je často potrebné vyskúšať viacero návrhov a preskúmať viacero alternatív. Treba nájsť nové spôsoby riešenia alebo skombinovať existujúce myšlienky. Je to proces skúmania a experimentovania. Tvorivá práca je neefektívna. Treba pri nej dlho skúmať a experimentovať, aby sme nakoniec dosiahli nejaký postup.

Projektový vedúci potrebuje určiť, aký pomer práce k postupu pre splnenie daného cieľa sa očakáva. Môže tak pomôcť ľuďom oddeliť úlohy, ktoré budú vyžadovať veľa pokusov, aby ich bolo možné splniť (napríklad návrh novej architektúry softvérového systému) od tých, ktoré sú priamočiare a nevyžadujú veľa rozmýšľania (napríklad naplnenie databázy).

Pomer práce k výkonu vyjadruje, koľko práce (alebo času) je potrebnej pre dosiahnutie jednotky postupu. Napríklad [40]:

- Naplnenie databázy údajmi: 1 k 1.
- Návrh nového vyhľadávacieho algoritmu: 5 k 1.
- Pretvorenie celého používateľského rozhrania: 5 k 1.

– Vytvoriť perpetuum mobile: ∞ k 1.

Manažér môže ľuďom pomôcť, aby ich pomer práce k postupu bol čo najnižší tým, že sa bude po každej úlohe pýtať tieto otázky:

– Čo sme sa naučili z tejto úlohy, čo by nám mohlo pomôcť s ďalšou úlohou?

– Sú nejaké nové otázky k ďalšej úlohe, ktoré by zvýšili veľkosť postupu?

– Potrebujeme ďalšie informácie, ktoré zlepšia ďalší pokus?

– Dajú sa ciele definovať lepšie?

Pomer práce k postupu naznačuje ešte ďalšiu vec: Pracovník, ktorý je skúsenejší alebo šikovnejší, dosiahne lepší pomer práce k postupu, než dosiahne menej skúsený pracovník.

Z pomeru práce k postupu je zrejmé, že rôzna úroveň kvality bude mať rôzny pomer práce k postupu. Nájsť dostatočné riešenie môžeme už po piatich pokusoch, ale až po pätnástich pokusoch môžeme nájsť výborné riešenie. Úlohou projektového vedúceho je teda pomôcť tímu rozhodnúť, aké veľké investície sú vhodné pre dosiahnutie daných cieľov.

Úskalia sledovania postupu v softvérovom projekte

V nástrojoch na riadenie projektu, akým je napríklad aj Microsoft Project, existuje v niektorých prípadoch klam, ktorý hovorí, že zadaná úloha môže byť čiastočne splnená (napr. že úloha môže byť splnená na 10 % alebo 20 %). Takýto stav vzniká pri automatickom dopočítaní stavu úlohy na základe aktuálneho času a času vyhradeného na úlohu.

Ak pod úlohou myslíme splnenie nejakého cieľa, tento klam je očividný – buď cieľ splnený je, alebo cieľ splnený nie je. Je to čierno-biela, binárna logika. Ak je cieľ definovaný neurčito a formulácia zámeru je nepresná, ako napríklad „napíš nejaké vyjadrenie k posudku“, tak je táto úloha splnená v momente, keď ju začneme vykonávať. Akonáhle napíšeme jednu vetu, napísali sme „nejaké vyjadrenie k posudku“.

Na druhú stranu, ak je úloha dobre definovaná a jej výsledky sa dajú odovzdať, potom dosiahneme splnenie úlohy práve vtedy, keď odovzdáme jej výsledok. Napríklad „vytvor kompletnú používateľskú príručku a technickú príručku“. Táto definícia úlohy je oveľa užitočnejšia, pretože jej splnenie sa dá jasne zmerať. Úloha je splnená, iba ak sú dokumenty napísané, boli skontrolované a opravené a sú pripravené pre zverejnenie. Skončili sme, ak sa už nemusia robiť ďalšie zmeny a môžeme sa presunúť na ďalšiu úlohu.

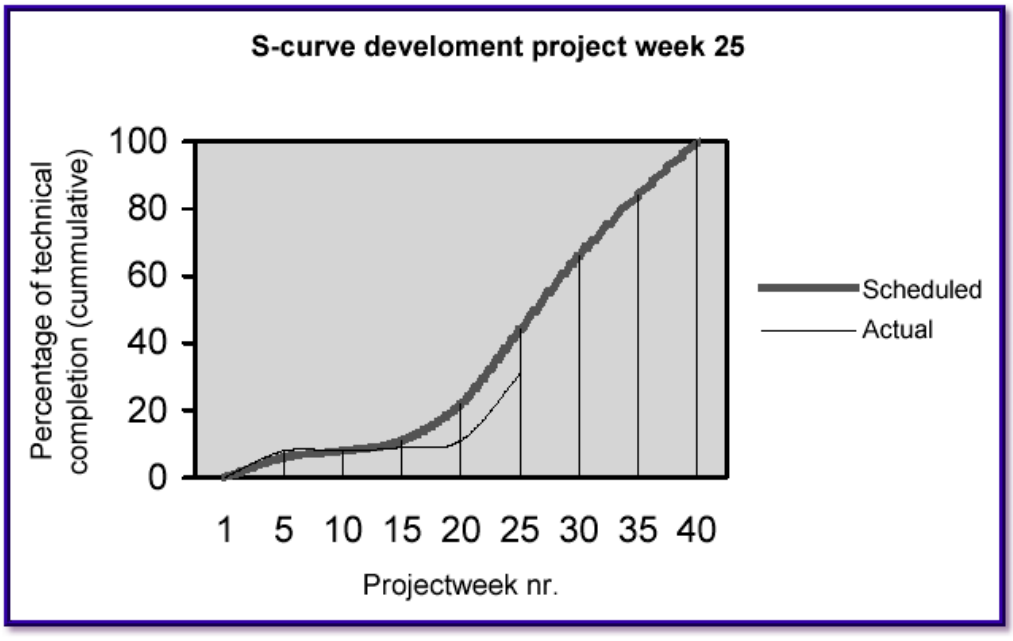
Nebezpečenstvo toho, že uveríme, že úloha môže byť čiastočne splnená je, že dáva falošný pocit bezpečia. Ľudia budú tvrdiť, že splnili úlohu na 50 % v polovici jej vyhradeného času, pretože definovať, ako vyzerá úloha splnená na 50 % môže byť náročné. Aj v prípade, že ešte treba dokončiť 90 % úlohy, budú stále trvať na tom, že keďže prešla polovica času vyhradeného na danú úlohu, musela zostať tiež už iba

polovica práce, ktorú treba vykonať. Túto mylnú predstavu majú obzvlášť ľudia, ktorí veria, že čas je elastický. To znamená, že sa dá nahustiť ľubovoľné množstvo práce na danú dĺžku času, záleží iba na tom, ako tvrdo sa pracuje [38]. Ale v softvérových projektoch často platí, že keď sa zdá, že je 90 % úlohy hotovej, zostáva ešte dokončiť zvyšných 90 %.

Zobrazenie postupu v softvérovom projekte

Je dôležité, aby tím pracujúci na projekte, ale aj samotný zákazník videli postup v softvérovom projekte. Výsledky v softvérovom projekte majú slabšiu viditeľnosť a väčšinou je v projekte zapojených veľa ľudí z rôznych oblastí. Preto treba pri zobrazovaní postupu zvoliť jednoduchú metriku. Tá sa dá zvoliť analýzou práce, ktorú bude treba vykonať. Práca sa potom rozdelí na menšie úlohy a ohodnotí sa ich príspevok k celkovému postupu. Po rozdelení práce a jej naplánovaní máme predstavu o tom, ako bude vyzerať vývoj projektu.

Naplánovaný stav projektu môžeme zakresliť do grafu, kde na x-ovej osi bude čas a na y-ovej osi bude vykonaná práca. Pod vykonanou prácou môžeme rozumieť napríklad počet hotových tabuliek v databáze alebo počet naprogramovaných a otestovaných funkcií v projekte. Do tohto grafu sa bude časom zakresľovať aj skutočný stav projektu. Postup sa nesleduje neustále, ale v pravidelných intervaloch.



Obr. 12. Graficky znázornený naplánovaný a skutočný vývoj projektu [39].

Z obrázku **Obr. 12,** ktorý zachytáva vývoj stavu (teda postup) v softvérovom projekte, sa dajú prečítať nasledovné skutočnosti:

- Z horizontálneho posunu naplánovaného a skutočného stavu projektu sa dá usúdiť, že projekt má dvojtýždňové meškanie.
- V projekte bolo desať týždňov, počas ktorých v projekte nenastal žiadny merateľný postup.
- Koniec naplánovanej krivky v posledných týždňoch je príliš strmý. To naznačuje, že dĺžka trvania projektu sa výrazne predĺži. Pretože v súčasnom týždni (25) je krivka so skutočným stavom paralelná s naplánovanou krivkou, je pravdepodobné, že sa bez zásahu do plánu nestihne dohnať dvojtýždňové meškanie.

Toto nie je jediný spôsob zobrazenia postupu v softvérovom projekte. Často sa používa aj graf, ktorý je opačne orientovaný – zachytáva, koľko toho ešte treba urobiť, namiesto toho, koľko sa už urobilo. To, ktorý z nich sa zvolí, závisí iba od psychológie. Niekoho viac motivuje, keď sa hýbe po krivke smerom nahor, ďalší chce dosiahnuť nulu.

Záver

Ak chceme vidieť v softvérovom projekte nejaký postup, nezaobídeme sa bez plánu. Plán nám umožní sledovať postup v softvérovom projekte. Pomocou plánu vieme zhodnotiť, aký mal byť skutočný stav projektu a ako veľmi sa líši od naplánovaného stavu projektu. Sledovanie vývoja charakteristík softvérového projektu v čase vykonávania projektu umožňuje projektovému vedúcemu riadiť projekt a v prípade rozdielov od naplánovaného stavu včas zasiahnuť a nasmerovať ľudí správnym smerom. Vyhodnocovanie vykonanej práce a analýza nákladov a úsilia vynaložených na vykonanú prácu napomáhajú aj efektívnejšiemu využitiu zdrojov v budúcnosti.

Pri sledovaní softvérového projektu je pritom dôležité rozoznať prácu od skutočného postupu. Zákazníka väčšinou nezaujíma, že na projekte sa pracuje, skôr ho zaujíma, aký bol postup: Čo sa od posledného stretnutia zmenilo, ako sa tím priblížil k dosiahnutiu dohodnutých cieľov. Aj preto je dôležité, aby sme vedeli postup nejakým spôsobom zmerať, prípadne zakresliť do grafu. Zvolený graf má aj psychologický efekt, môže motivovať k ďalšej práci a ďalšiemu postupu. Okrem aktuálneho stavu sa z neho dajú rýchlo vyčítať napríklad aj možné problémy a môže byť aj podnetom pre zmenu plánu.

Použitá literatúra

38. Nick Jenkins: *A Project Management Primer or "a guide on how to make projects work",* 2005.
http://www.nickjenkins.net/prose/projectPrimer.pdf

39. Joop van der Linden: *Monitoring Progress in Software Development,* júl 2003. http://www.stsc.hill.af.mil/crosstalk/2003/07/vanderLinden.html
40. Scott Berkun: *Work vs. Progress,* august 2005. http://www.scottberkun.com/essays/essay45.htm

Annotation

Monitoring progress in software project and management

Despite the best planning, the best team participation and expecting all possible pitfalls, it seems that software projects are experienced to cause unexpectable problems. One possible way how to avoid unsuccessful project termination is to monitor progress of the software project. To be able to monitor development and progress of the software project it is important to create a plan. It is also needed to choose feasible metrics to measure the progress. Choosed metrics should not drive people to achieve better results in a measurement, but it should motivate them to effectively attain identified goals of the project. When the plan is ready, it is possible to ask, which planned tasks are already accomplished, estimate amount of tasks remaining and compare scheduled project status to the real project status. In case of difference, it is possible to correct the plan and aim the effort to the right direction.

Vývoj tímu v softvérovom projekte a vplyv na manažment

Od diktatúry k slobode

MATEJ MACHÁČ

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
machac01@student.fiit.stuba.sk

Abstrakt. V práci je rozoberaný vývoj tímu z hľadiska troch rôznych modelov. Ako prvý je rozoberaný Bruce Tuckmanov model „formovanie, kryštalizácia, vyjasnenie, realizácia“ aj s neskôr autorom pridanou piatou fázou „ukončenie“, ktorá ma špeciálny vplyv na manažment. Ako druhý je popísaný Herseyho a Blanchardov situačný vodcovský model so štyrmi fázami a podobnosť fáz týchto dvoch modelov. Posledným rozoberaným modelom je Tannenbaumovo a Schmidtovo kontinuum. Ďalej sú popísané úlohy manažéra v rôznych fázach vývoja tímu podľa modelu podobného s Bruce Tuckmanovým modelom a je dopodrobna opísaná úloha manažéra a zmeny v riadení pri siedmich stupňoch pridelenej voľnosti v modeli Tannenbaumovho a Schmidtovho kontinua.

Úvod

Bez tímovej práce sa väčšina firiem nezaobíde, a to ani softvérových. Časom, keď softvér vyvíjal jeden človek, už dávno odzvonilo. Zložitosť softvérových produktov stále rastie a kladie stále vyššie nároky nielen na kvalifikáciu, ale aj na tímovú spoluprácu.

Efektívny tím ale nie je akési zavretie skupinky do jednej miestnosti a priradenie im konkrétnej úlohy. Každý tím musí mať svoj cieľ, motiváciu a vedúceho. Vedúcim musí byť manažér schopný vytvoriť a viesť tím, rozhodovať, prideľovať prácu a právomoci a v neposlednej rade aj prebrať zodpovednosť za výsledky celého tímu, ale bez toho, aby obmedzil jeho funkčnosť. A funkčnosť je závislá od vývoja.

V priebehu času prechádza tím vývojom. Je to vývoj od nesúrodej skupinky ľudí s nejasnými cieľmi až po dobre pracujúci efektívny stroj. Od direktívneho rozhodovania, cez presadzovanie rozhodnutí, spolupráce na tvorbe rozhodnutí až po prenechanie tvorby rozhodnutí na tím. Od diktatúry k slobode.

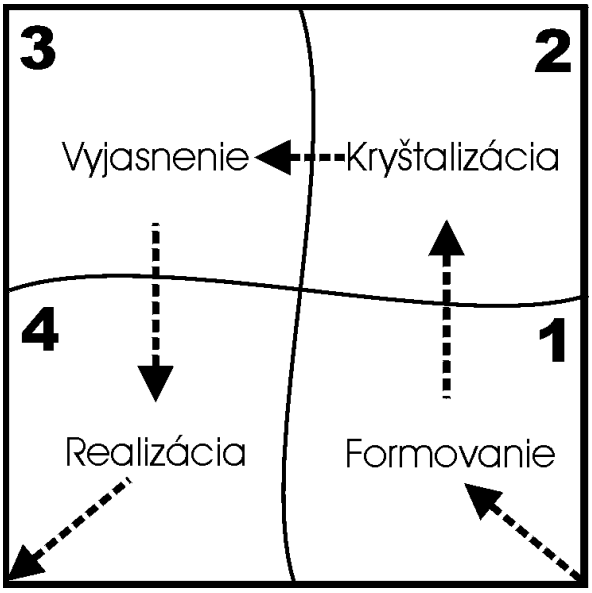
Vývoj tímu

V roku 1965 publikoval Dr. Bruce Tuckman svoj model vývoja tímu „formovanie, kryštalizácia, vyjasnenie, realizácia“ (Forming, Storming, Norming, Performing) a v roku 1970 ho ešte doplnil o fázu ukončenia (Adjourning) [41]. Teória formovania, kryštalizácie, vyjasnenia a realizácie pomáhala elegantne vysvetliť vývoj tímu a správanie. Je podobný s ďalšími modelmi, ako napríklad Tannenbaumovým a Schmidtovým kontinuom (Tannenbaum and Schmidt Continuum) a hlavne s Herseyho a Blanchardovým situačným vodcovským modelom (Hersey and Blanchard's Situational Leadership model), ktorý vznikol zhruba v tom istom čase.

Bruce Tuckmanov model vysvetľuje, ako tím dospieva a získava zručnosti, ako sa vyvíjajú vzťahy a ako sa mení vedenie vedúcim tímu. Začínajúc s príkazovým vedením cez usmerňovanie, spoluprácu až ku prenechaniu vedenia ako znázorňuje i Herseyho a Blanchardov situačný vodcovský model. V tomto bode si tím môže vybudovať následníka vedúceho a pôvodný vedúci môže ísť vytvárať nový tím. Tento pokrok v správaní tímu a štýle vedenia je jasne vidieť v Tannenbaumovom a Schmidtovom kontinuu – ako právomoci a voľnosť dávaná tímu vedúcim rastie, kým kontrola vedúcim sa znižuje. Vo všetkých troch modeloch je vidieť ten istý efekt, ale reprezentovaný tromi odlišnými spôsobmi.

Bruce Tuckmanov model

Fázy Bruce Tuckmanovho modelu a ich súvis možno jasne vidieť na obrázku (**Obr. 13**).



Obr. 13. Bruce Tuckmanov model (podľa [41])

Formovanie

V tejto fáze je typická vysoká závislosť od vedúceho pre vedenie a usmerňovanie. Takmer všetky dohody v tíme vznikli na základe nariadenia vedúcim. Nie sú vyriešené individuálne roly a zodpovednosti, zmysel tímu, cieľ a vonkajšie vzťahy. Členovia testujú toleranciu systému a vedúceho. Procesy sú často zanedbávané.

Kryštalizácia

Vedúci vedie tím. Rozhodnutia v tíme vznikajú ťažšie. Členovia tímu súperia a snažia sa presadiť vo vzťahu k ostatným členom tímu a vedúcemu, ktorý musí odolávať výzvam od členov tímu. Vyjasňuje sa význam, ale stále zostáva dosť nejasností. Vznikajú skupinky a záujmy a môžu sa prejaviť zápasy o moc. Aby sa tím vyhol odvedeniu pozornosti vzťahmi a emóciami, musí sa zamerať na ciele.

Vyjasnenie

S pomocou vedúceho sa v tíme formuje dohoda a súhlas. Roly sú jasné a akceptované. Kým veľké rozhodnutia sa robia dohodou tímu, malé môžu byť pridelené jednotlivcovi alebo malej skupinke v rámci tímu. Tím sa môže zapájať do zábavných a spoločenských aktivít, aby zlepšil vnútorné vzťahy. Platí všeobecný rešpekt voči vedúcemu, ale časť vedenia je viac zdieľaná tímom. Vedúci pomáha a oprávňuje.

Realizácia

Tím má väčší prehľad v stratégii, veľmi dobre vie, prečo robí to, čo robí. Má spoločnú predstavu a je schopný fungovať aj samostatne. Dochádza k novému definovaniu cieľov. Objavujú sa nezhody, ale teraz sú už riešené vnútri tímu. Tím dokáže pracovať na dosiahnutí cieľa a pritom dokáže priebežne dávať pozor na vzťahy, štýl a problémy procesu. Členovia tímu na seba dohliadajú. Tím vyžaduje pridelenie úloh a projektov vedúcim. Už od neho nepotrebuje asistenciu alebo byť inštruovaný. Členovia môžu žiadať od vedúceho pomoc s osobným a medziľudským vývojom.

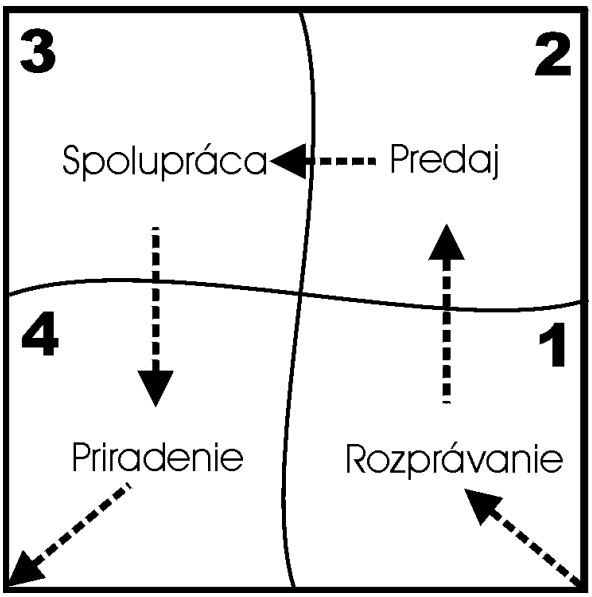
Ukončenie

Bruce Tuckman zdokonalil svoju teóriu okolo roku 1975 a pridal piatu fázu k svojmu modelu – nazval ju ukončenie (taktiež sa označuje aj deformovanie (Deforming) a smútenie (Mourning)). Ukončenie je skôr doplnkom ako rozšírením pôvodných štyroch fáz – pozerá sa na skupinu z perspektívy za účelom prvých štyroch fáz. Táto fáza je určite veľmi dôležitá pre ľudí v skupine a ich blaho, ale nie pre hlavnú úlohu riadenia a vývoja tímu, ktorá je stredobodom pôvodných štyroch fáz.

Tuckmanova piata fáza je rozchod skupiny v priaznivom prípade po úspešnom ukončení úlohy, keď je jej účel splnený a všetci sa môžu rozísť k novým veciam, spokojní s dosiahnutým. Z organizačného hľadiska je v piatej Tuckmanovej fáze nápomocné rozoznanie a citlivosť k slabinám ľudí, hlavne ak členovia tímu boli úzko spätí a cítia pocit neistoty alebo hrozbu z tejto zmeny. Pocity neistoty by boli prirodzené pre ľudí, ktorí si ťažšie zvykajú.

Herseyho a Blanchardov situačný vodcovský model

Fázy modelu a ich súvis môžete vidieť na obrázku (**Obr. 14**). Názvy fáz vychádzajú z anglických názvov „telling“, „selling“, „participating“ a „delegating“. Prvej fáze treba rozumieť ako dávaniu pokynov (rozprávanie), druhá už je dávaním pokynov aj s vysvetlením (predaj), v tretej už vedúci spolupracuje (spolupráca) a v poslednej (štvrtej) len priraďuje úlohy tímu (priradenie).



Obr. 14. Herseyho a Blanchardov situačný vodcovský model (podľa [41])

Klasický situačný vodcovský model manažmentu a vodcovský štýl taktiež ilustruje ideálny vývoj od nezrelosti (fáza 1) až po zrelosť (fáza 4), počas ktorých sa riadenie a štýl progresívne vyvíja od relatívne oddeleného nariaďovania úloh (1) cez viac manažérsky založené úlohy vysvetľovania (2) a spolupráce (3) až po cieľovú fázu relatívne oddeleného prideľovania (4), keď už sa tím v ideálnom prípade vo veľkej miere riadi sám a v lepšom prípade obsahuje aspoň jedného potenciálneho následníka riadenia/vedenia.

Cieľom vedúceho alebo manažéra je teda vyvinúť tím cez štyri fázy a potom sa presunúť k ďalšej role.

Ironické je, že tohto vývoja sa obáva väčšina manažérov. Dobré organizácie ale vysoko hodnotia vedúcich a manažérov, ktorí to vedia dosiahnuť.

Model taktiež ilustruje štyri hlavné riadiace a vodcovské štýly, medzi ktorými je dobrý vodca schopný prepínať v závislosti od situácie (napríklad vyspelosť tímu v súvislosti s podrobnou úlohou, projektom alebo výzvou).

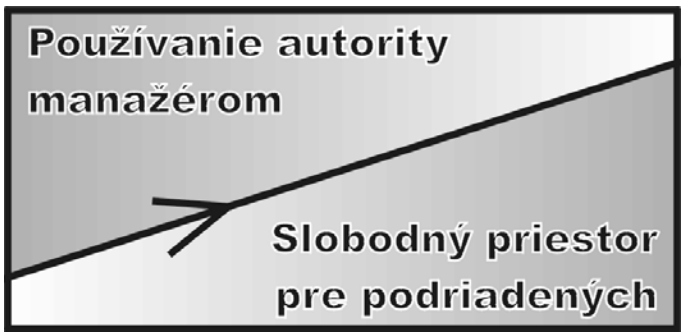
Medzi oboma spomenutými modelmi možno badať spoločné znaky – prekrývanie sa fáz oboch modelov. Prvá fáza Bruce Tuckmanovho modelu „formovanie,

kryštalizácia, vyjasnenie, realizácia“ je podobná fáze Herseyho a Blanchardovho situačného vodcovského modelu „rozprávanie“. Druhá fáza „kryštalizácia“ Bruce Tuckmanovho modelu je podobná fáze „predaj“. Obdobne je tretia fáza podobná fáze „spolupráca“ a štvrtá fáze “priradenie“.

Tannenbaumovo a Schmidtovo kontinuum

Tannenbaumovo a Schmidtovo kontinuum je jednoduchý model, ktorý poukazuje na vzťah medzi stupňom voľnosti, ktorý sa manažér rozhodne dať tímu a stupňom autority, ktorý si ponecháva. Ako sa sloboda tímu zvyšuje, autorita manažéra sa znižuje. To je pre manažéra aj tím pozitívna cesta vývoja.

Diagonála na obrázku zobrazujúcom tento model (**Obr. 15**) je ekvivalentná čiarkovanej čiare v obrázkoch predchádzajúcich dvoch modelov (**Obr. 13**, **Obr. 14**).

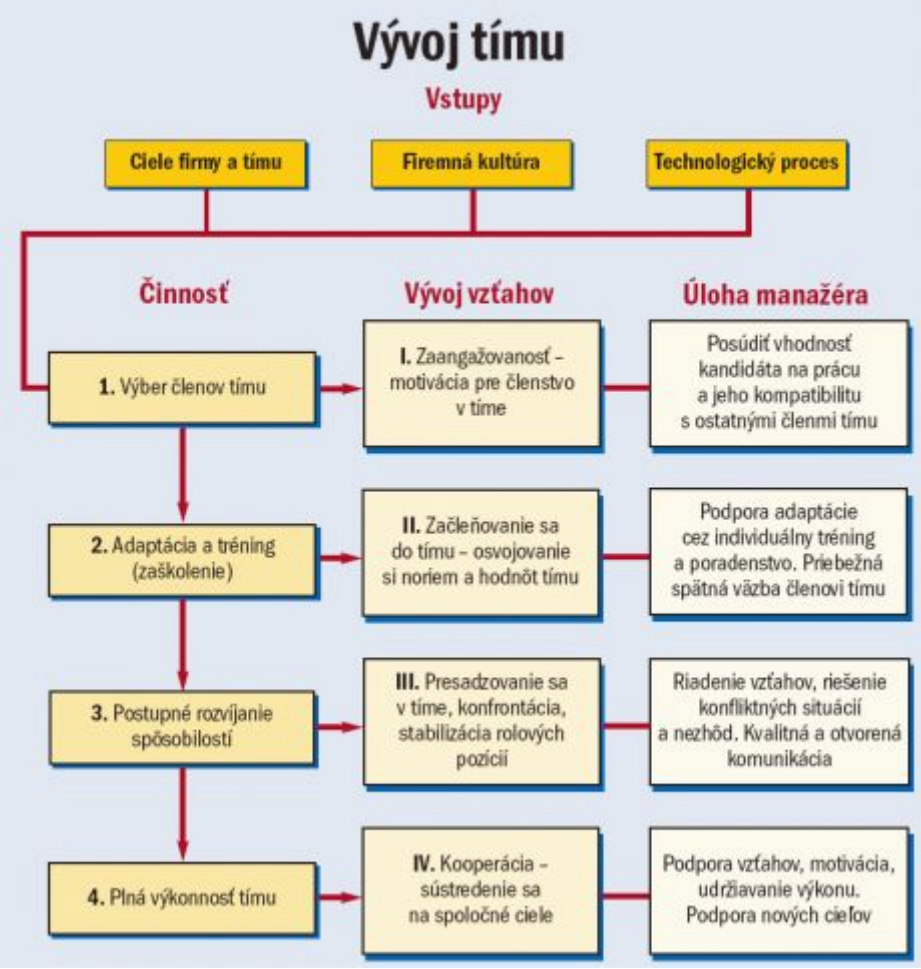


Obr. 15. Tannenbaumovo a Schmidtovo kontinuum (podľa [42])

Vplyv na manažment

Úlohy manažéra sa počas vývoja tímu neustále menia, čo si musí manažér uvedomovať a prispôsobovať tomu patrične svoje konanie. Úlohy manažéra počas jednotlivých fáz vývoja tímu sú uvedené na obrázku (**Obr. 16**). Obrázok má podobnosť s Bruce Tuckmanovým modelom, aj keď sa významom fáz mierne líši. Počas vývoja tímu sa ale pre manažéra objavujú aj iné úlohy, ktoré treba riešiť a taktiež sú kladené požiadavky aj na jeho vedenie. Keď má tím slabé vedenie a manažér tímu nie je schopný implementovať a kontrolovať pravidlá spolupráce v tíme, je jeho rola vodcu potlačená, čím umožňuje silným a ambicióznym jedincom v tíme preberať vedenie. Ale ani autoritatívne vedenie nie je žiadané, keď manažér vnucuje svoje riešenia a chce od tímu, aby pracoval "metódou známeho výsledku". To spôsobuje rozladenosť v tíme, nechuť riešiť problém a pocity nízkej subjektívnej hodnoty pre firmu. Nutné je i dosahovať v tíme zjednotený pohľad na úlohu a ciele. Manažér musí s tímom diskutovať o jeho cieľoch a normách výkonnosti a ľudia musia "ťahať za jeden povraz". I samotný riešený problém musí byť dobre definovaný, aby ho členovia tímu

nevnímali odlišne. Doteraz zmienené problémy treba riešiť hlavne v prvých dvoch fázach Bruce Tuckmanovho modelu. Nakoniec sa nesmie zabúdať ani na diskusiu o vzťahových problémoch a ich riešenie. Toto sa dotýka hlavne tretej a štvrtej fázy Bruce Tuckmanovho modelu.



Obr. 16. Úlohy manažéra počas jednotlivých fáz vývoja tímu (prevzaté z [43])

V prípade Bruce Tuckmanovho modelu hrá ale veľmi dôležitú úlohu aj piaty stupeň vývoja tímu. Manažér musí byť schopný rozoznať pocity členov tímu po skončení úlohy, počas ktorej dlhodobo spolupracovali s úzkou skupinou ľudí. Musí vedieť rozpoznať pocity neistoty a hrozby, ktoré na členov tímu pôsobia zo zmeny a citlivo k nim podľa toho pristupovať.

U modelu Tannenbaumovho a Schmidtovho kontinua je dôraz kladený na úlohu manažéra vyvíjať tím. Mal by prideľovať úlohy a žiadať od tímu, aby robil vlastné

rozhodnutia až do stupňa, ktorý zodpovedá ich schopnostiam. Na tento účel slúži stúpajúca škála pridelenej voľnosti.

Postupom času by sa mal manažér snažiť previesť tím z jedného konca na druhý (ako ukazuje obrázok (**Obr. 15**)) hore po stupnici, kde by sa mal taktiež snažiť vytvoriť jedného alebo viacerých potenciálnych nástupcov, ktorí prevezmú jeho úlohu.

Pri používaní Tannenbaumových a Schmidtových princípov je veľmi dôležité si uvedomiť, že bez ohľadu na množstvo zodpovednosti a voľnosti prenesenej na tím, manažér zostáva zodpovedný za akékoľvek katastrofické problémy, ktoré vzniknú. Pridelenie slobody a zodpovednosti za rozhodnutia tímu ho v žiadnom prípade zodpovednosti nezbavujú. Preto je prideľovanie či už jednotlivcovi alebo tímu úloha pre skúseného manažéra. Ak sa všetko vydarí, tím musí zožať všetku slávu, ak sa všetko pokazí, manažér musí zniesť hanbu. Toto je férové, pretože manažér je zodpovedný za zhodnotenie dôležitosti každej situácie zahŕňajúc aj riziká a za stupeň voľnosti, ktorý môže byť daný tímu. Je rozumné použiť tento pohľad pri Tannenbaumovom a Schmidtovom kontinuu, aj keď nie je jeho súčasťou, pretože by mohlo byť oslabené až kontraproduktívne.

Stupne pridelenej voľnosti Tannenbaumovho a Schmidtovho kontinua [42]:

1. Manažér rozhoduje a oznamuje rozhodnutie – zatiaľ sa tím aktívne nezúčastňuje na tvorbe rozhodnutí.
2. Manažér rozhoduje a potom „predáva“ svoje rozhodnutie tímu (tu vidno podobnosť s druhou fázou Herseyho a Blanchardovho situačného vodcovského modelu) – vysvetľuje dôvody tímu, hlavne pozitívne dôsledky, ktoré sa tímu budú páčiť, vďaka čomu má tím pocit, že si uvedomuje dôležitosť tímu.
3. Manažér prezentuje rozhodnutia s nápadmi, ktoré k nemu viedli a prijíma otázky – diskutuje dôvody vedúce k rozhodnutiu, čo tímu umožňuje pochopiť a prijať rozhodnutia ľahšie ako v stupňoch 1 a 2, pričom tím už je viac zapojený do rozhodovania.
4. Manažér uvádza provizórne rozhodnutie a prijíma diskusiu – zohľadní názory a až potom sa rozhodne, čo už tímu dáva možnosť reálne zasiahnuť do manažérovho konečného rozhodnutia a taktiež ukazuje, že tím má čo ponúknuť (čím sa zvyšuje motivácia).
5. Manažér prezentuje situáciu alebo problém, získa návrhy a potom rozhodne – od tímu sa očakáva, že navrhne nápady alebo nové možnosti a ujasní si dôsledky každej možnej alternatívy. Tento stupeň už predstavuje vysoký a špecifický vývoj tímu vhodný hlavne vtedy, keď už má tím podrobnejšie znalosti a skúsenosti ako manažér, čo tímu pridáva ďalšiu motiváciu a voľnosť.
6. Manažér vysvetlí problém, definuje parametre a žiada od tímu, aby rozhodol – v tomto stupni už manažér úspešne previedol rozhodnutia na tím, aj keď iba v nejakých limitoch. Manažér sa môže rozhodnúť zúčastniť alebo nezúčastniť na rozhodovaní. Aj keď sa zdá, že tento stupeň už dáva tímu obrovskú

voľnosť, manažér môže ešte stále kontrolovať riziko a výstupy v rámci stanovených obmedzení.

7. Manažér dovolí tímu identifikovať problém, vytvoriť možnosti a rozhodnúť sa v rámci priradených obmedzení – toto už je extrémny stupeň voľnosti, pričom tím efektívne vykonáva to, čo manažér robil v stupni 1. Manažér môže byť súčasťou tímu, v tom prípade už ale nemá vyššiu autoritu ako ktorýkoľvek iný člen tímu.

Záver

Ako vidieť, všetky modely vývoja tímu sa zhodujú v jednom – vedenie vedúcim sa od diktatúry postupne mení až po voľnosť. Bez tohto vývoja by nebola možná plná výkonnosť tímu. Nie je však možné začínať hneď s úplnou voľnosťou, ľudia tvoriaci tím sa najprv musia naučiť spolupracovať. Až keď zvládnu riadenie seba samých a pochopia fungovanie tímu, môžu postupne participovať na jeho riadení a získavať skúsenosti. Posledným stupňom je vznik nového potenciálneho vedúceho.

Každý vedúci musí rozumieť svojim úlohám, musí dokázať presadiť svoje rozhodnutia v počiatočných fázach vývoja tímu, rozumne obmedzovať spoluprácu na tvorbe rozhodnutí až obmedzovať už len priestor pre rozhodnutia robené tímom, až nakoniec už len prideľovať úlohy tímu bez potreby bližšieho riadenia.

Popri tom ale musí zvládať i manažment rizík. Netreba totiž zabudnúť, že nakoniec v prípade úspechu bude to zásluhou tímu, v prípade neúspechu dôsledkom vedúceho. Preto veľa šťastia všetkým manažérom a vedúcim (medzi týmito pojmami je rozdiel, pričom ideálnym prípadom je spojenie do jednej osoby [44]).

Použitá literatúra

41. Chapman, Alan: Bruce Tuckman forming storming norming performing model, www.businessballs.com, 1995-2005.
42. Chapman, Alan: Tannenbaum and Schmidt Continuum, www.businessballs.com, 1995-2005.
43. Kališ, Mojmír: Tímová spolupráca vo firme, kariera.hnonline.sk, 3. 11. 2005.
44. Rudy, Ján, Piškanin, Andrej, a kolektív: Základy manažmentu. Univerzita Komenského v Bratislave, 1999.

Annotation

Development of team in software project and the influence on management

In this paper I took a closer look on process of development of a team from three different perspectives. As first view I used Bruce Tuckman „forming, storming, norming, performing“ model, also mentioning the later added fifth stage „adjourning“ which has a special meaning in

terms of management. As second view I used Hersey and Blanchard's Situational Leadership model which also has four stages and I pointed out the similarities of stages of these two models. Tannenbaum and Schmidt Continuum is the last view. As next I described the managing tasks which are needed in different stages of development of team explained on model similar to Bruce Tuckman „forming, storming, norming, performing“ model and I described in detail changes in managing a team using seven stages of freedom delegating in Tannenbaum and Schmidt Continuum.

Nové technológie, nástroje a ich vplyv na manažment softvérového projektu

PAVOL PRIKRYL

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

Abstrakt. Zložitosť riadenia a vývoja v softvérových projektoch so sebou prináša aj rôzne technológie a softvérové nástroje. Pomocou nich je možné napríklad vytvárať komplexné modely, automaticky generovať zdrojové kódy a dokumentáciu alebo komunikovať s inými ľuďmi, ktorí sa na týchto projektoch podieľajú. Z veľkého množstva nástrojov a technológií, ktoré sú už v súčasnosti k dispozícii, som sa zameral na tie, ktoré pomáhajú vytvárať a udržiavať dokumentáciu k softvéru, ako aj tie, ktoré zlepšujú komunikáciu medzi členmi tímu. Pre úspech softvérového projektu je priam nevyhnutné, aby o používaných nástrojoch vedeli a rozhodovali ľudia, ktorí ho riadia. Aké nástroje a technológie použiť? Odpovedať na túto otázku nie je vôbec jednoduché.

Úvod

V dnešnom svete medzinárodného obchodu je nesmierne dôležité dokončiť softvérový projekt načas a súčasne neprekročiť plánovaný rozpočet. Softvérová spoločnosť si jednoducho nemôže dovoliť so svojím projektom meškať, pretože jej zákazník by veľmi rýchlo prešiel ku konkurencii. Práve preto je dôležité zaoberať sa novými technológiami a nástrojmi, ktoré môžu podstatne znížiť čas a prostriedky potrebné pre vývoj softvéru a uvedomiť si, aký vplyv majú na manažment softvérového projektu.

Táto esej sa zaoberá dôležitosťou nástrojov a technológií, ktoré sa používajú pri tvorbe dokumentácie počas práce na softvérovom projekte. Za dokumentáciu softvéru sa považuje akýkoľvek dokument, ktorého cieľom je informovať o softvérovom systéme, ku ktorému patrí. Medzi takúto dokumentáciu patria napríklad dokumenty s požiadavkami, špecifikáciou a architektonickým a podrobným návrhom. Medzi ľudí, ktorí s takouto dokumentáciou pracujú pri vývoji softvéru, patria napríklad manažéri, vedúci projektov, vývojári, ale aj zákazníci.

Ďalšou dôležitou oblasťou, ktorou sa táto esej zaoberá, je komunikácia medzi ľuďmi, ktorí sa zúčastňujú práce na softvérovom projekte. Spôsob komunikácie a zvolená technológia alebo nástroje, ktoré majú zvýšiť jej efektivitu, môžu ovplyvniť

úspech alebo neúspech projektu. Preto by sa manažment mal zamyslieť aj nad možnosťami, ktoré ponúkajú nové technológie a nástroje v oblasti komunikácie.

Nástroje v softvérovom projekte

Jednou z často kladených otázok, ktorú sa pýtajú vývojári, je, aký nástroj alebo nástroje budú používať. Hoci je nástrojov veľké množstvo, často sa oplatí použiť práve tie najjednoduchšie. Na rozdiel od komplexných nástrojov je ľahšie naučiť sa používať, používať a zdieľať s ostatnými jednoduchšie nástroje. Aj komplexné nástroje majú svoje miesto, efektivita jednoduchších však môže byť vyššia.

Pri výbere komplexných nástrojov je potrebné prihliadnuť na investíciu, ktorú treba vynaložiť a na získanú hodnotu, ktorá vznikne pomocou takéhoto nástroja. Určiť vhodnosť použitia softvérového nástroja na základe týchto kritérií nie je jednoduché. Preto by sme mali aspoň približne poznať výhody a nevýhody použitia CASE nástrojov. Niektoré z nich sú uvedené v tabuľke **Tab. 5**.

Použitie CASE nástrojov	
Nevýhody	**Výhody**
– Potreba zacvičenia a vzdelávania ľudí pracujúcich s nástrojom. – Náklady na vývoj a údržbu modelov vytvorených nástrojom. – Cena za aktualizáciu nástroja. – Nedostatočná integrácia s inými nástrojmi. – Potreba upravovať výsledky nástroja. – Vygenerovaný zdrojový kód môže byť príliš jednoduchý alebo obsahovať nadbytočné informácie vyžadované nástrojom.	– Generovanie zdrojových kódov. – Podpora zmeny úrovne abstrakcie (napríklad od analýzy cez návrh až ku zdrojovému kódu). – Testovanie konzistencie a validity vytvoreného modelu. – Synchronizácia modelov so zdrojovým kódom. – Generovanie dokumentácie. – Zefektívnenie práce. – Komunikácia pri vývoji v tíme.

Tab. 5. Možné výhody a nevýhody CASE nástrojov

Dokumentácia softvéru a používané nástroje

Počas práce na softvérovom projekte väčšieho rozsahu vzniká veľké množstvo dokumentácie. Tímy pracujúce v takýchto projektoch potom musia zápasiť s organizáciou a údržbou vytvorenej dokumentácie. Na druhej strane, pri softvérových projektoch malého a stredného rozsahu vzniká málo alebo až vôbec žiadna dokumentácia. Ľudia v takýchto projektoch si často uvedomujú dôležitosť dokumentácie, ktorú však nevytvárajú najmä kvôli časovým a iným obmedzeniam svojich prostriedkov. Ako môžu technológia a nástroje zjednodušiť vytváranie a zvýšiť použiteľnosť dokumentácie softvéru?

V prvom rade si treba uvedomiť, že s nástrojmi pracujú ľudia. Pomocou nástrojov chcú dosiahnuť nejaký cieľ, ale najprv sa musia daný nástroj naučiť používať. A pretože ľudia sú leniví, zvlášť programátori, dopredu si zvážia, koľko svojho času a úsilia sú ochotní venovať štúdiu práce s nástrojom, ktorý ich môže priblížiť k vytúženému cieľu.

Dokumentácia je nedielnou súčasťou každého softvérového systému. Vytváranie a udržiavanie dokumentácie softvéru môže byť pre niekoho nutné zlo, zatiaľ čo niekto iný môže mať z tejto úlohy radosť. Existuje však niekoľko objektívnych dôvodov, prečo vytvárať dokumentáciu softvéru. Jedným z nich je napríklad skutočnosť, že dokumentáciu vyžaduje ten, pre koho je softvér vytváraný.

Ďalším dôvodom pre vytváranie dokumentácie softvéru môže byť komunikácia s inou skupinou vývojárov, ktorá sa na softvérovom projekte podieľa. Nie vždy je možné umiestniť všetkých členov vývojového tímu na jedno miesto. Preto je potrebné nájsť spôsob, akým budú vývojári medzi sebou komunikovať a zdieľaná dokumentácia spolu s kombináciou občasných stretnutí, videokonferencií, e-mailov a iných nástrojov je možným riešením. Dokumentácia však nemôže byť primárnym prostriedkom pre komunikáciu, pretože je veľmi ľahké neporozumieť niečomu, čo je napísané. Napriek tomu je dokumentácia dobrým podporným mechanizmom.

Pre samotného vývojára má dokumentácia význam v tom, že lepšie pochopí niektoré veci. Tým, že človek niečo napíše, sformuluje svoje myšlienky na papier, môže odhaliť niektoré problémy, ktoré by mohli ohroziť úspech projektu. Ak sa riešenie nejakého problému javí ako jednoduché a priamočiare na prvý pohľad, pri pokuse opísať ho podrobnejšie sa môže ukázať ako veľmi komplikované.

Užitočnosť softvérových nástrojov pre dokumentáciu

Zodpovedať na otázku, aké technológie a nástroje sa používajú pri vytváraní a pri údržbe dokumentácie softvéru, nám môže pomôcť prieskum [45], ktorý sa uskutočnil v máji roku 2002. Účastníci prieskumu odpovedali na niekoľko otázok, ktoré sa týkali dokumentácie softvéru. Ich úlohou bolo tiež vymenovať technológie, ktoré považujú za užitočné a ktoré naopak nie.

Jedna z otázok bola položená takto: *„Ktoré softvérové nástroje považujete za NAJVIAC užitočné pri vytváraní / úprave / prehliadaní / generovaní dokumentácie softvéru? (Napríklad textové editory, textové procesory, tabuľkové procesory,*

Javadoc)". Účastníci, ktorí odpovedali na túto otázku, teda mohli vymenovať viacero nástrojov – ich odpovede zachytáva tabuľka **Tab. 6**.

Nástroj	Počet (%)
MS Word (a iné textové procesory)	54
Javadoc a podobné nástroje (Doxygen, Doc++)	51
Textové editory	22
Rational Rose	12
Together (Control Centre, IDE)	7

Tab. 6. Užitočné technológie pre dokumentáciu softvéru

Ako najužitočnejšie nástroje pre prácu s dokumentáciou softvéru označili účastníci prieskumu MS Word a iné textové procesory (54 %) a nástroje na automatické generovanie softvérovej dokumentácie (51 %), akými sú napríklad Javadoc, Doxygen, Doc++ a podobne. V menšej miere boli zastúpené textové editory (22 %) a potom Rational Rose (12 %) a Together (7 %). Ďalšie technológie, ktoré považovali účastníci prieskumu za užitočné, boli ArgoUML, Visio, FrameMaker, virtuálna tabuľa (whiteboard) a digitálne kamery, JUnit a XML editory.

Nasledujúca otázka znela: *„Ktoré softvérové nástroje považujete za NAJMENEJ užitočné pri vytváraní / úprave / prehliadaní / generovaní dokumentácie softvéru?"*. Zaujímavé je, že niekoľko nástrojov, ktoré boli väčšinou účastníkov označené za najužitočnejšie, označilo zopár účastníkov za najmenej užitočné. Medzi tieto nástroje patril MS Word a textové procesory (15 % ich označilo za najmenej užitočné), Javadoc a podobné nástroje (12 %), textové editory (7 %) a Rational Rose (2 %).

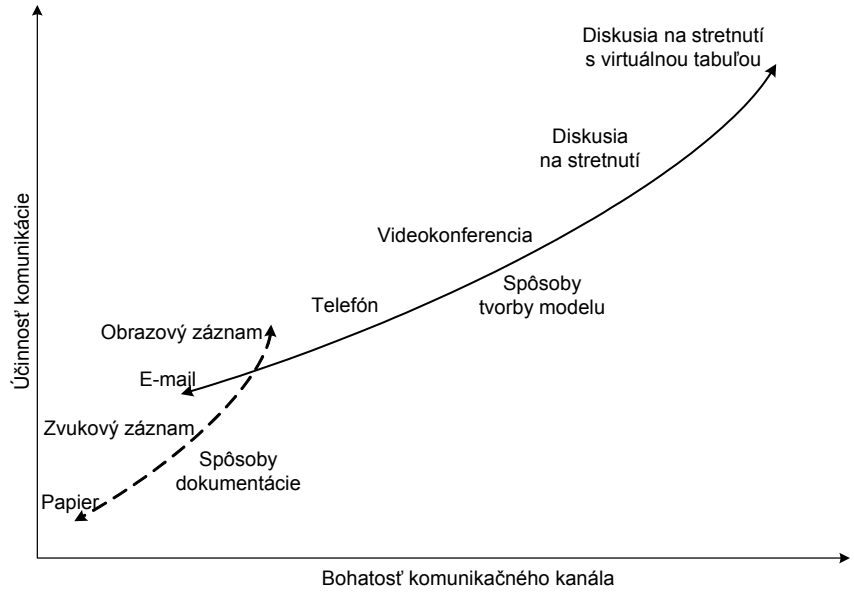
Z uvedeného prieskumu vyplýva, že existujú dva rôzne druhy technológií, ktoré sú užitočné pri vytváraní dokumentácie softvéru:

- Textové procesory – hoci nie sú najúčinnejším spôsobom komunikácie, sú dostatočne flexibilné a vo všeobecnosti jednoduché na používanie.
- Nástroje na automatické vytváranie dokumentácie – tieto nástroje zjednodušujú údržbu dokumentov tým, že ju zabezpečia prakticky samy.

Komunikácia pri vývoji softvéru

Efektívna komunikácia je v podstate základom každého úspešného softvérového projektu. Prečo je vlastne komunikácia taká dôležitá? Vo všeobecnosti si pri komunikácii vymieňajú jednotlivci nejaké informácie. Potreba komunikovať zasahuje prakticky do každej etapy softvérového projektu – počas analýzy, pri spresňovaní špecifikácie, vytváraní návrhu, implementácii a testovaní až po údržbu softvérového systému a technickú podporu pre zákazníka.

Pri spolupráci viacerých ľudí na jednom softvérovom projekte sa môžu uplatniť rôzne spôsoby komunikácie. Efektívnosť rôznych spôsobov komunikácie a bohatosť použitého komunikačného kanála je zobrazená na **Obr. 17**, ktorý bol použitý v eseji zaoberajúcej sa efektívnosťou komunikácie pri použití agilných metód [46]. Pri vytváraní dokumentácie je možné výsledok zapísať alebo vytlačiť na papier, čo v sebe zahŕňa napríklad aj dokumentáciu vo formáte HTML. Použitie rôznych spôsobov pri komunikácii však samozrejme závisí aj od situácie – v niektorom prípade môže byť videokonferencia účinnejšia ako diskusia na stretnutí.



Obr. 17. Spôsoby komunikácie [46]

Účinnosť komunikácie ovplyvňuje niekoľko faktorov. Napríklad fyzická vzdialenosť ľudí, ktorí navzájom komunikujú – so zväčšujúcou sa vzájomnou vzdialenosťou sa znižujú možnosti efektívnej komunikácie. Ľudia môžu napríklad sedieť v jednej miestnosti vedľa seba a za jedným počítačom pri programovaní v dvojici, alebo sa na druhej strane môžu nachádzať v rôznych budovách či dokonca na rôznych kontinentoch. Našťastie, v dnešnej dobe nie je problém komunikovať cez službu instant messaging alebo usporiadať virtuálne stretnutie napríklad na IRC serveri.

Ďalším problémom pri komunikácii môže byť časový faktor – ak napríklad severoamerická firma potrebuje komunikovať s ázijskou alebo európskou spoločnosťou, určite bude musieť vziať do úvahy aj rôzne časové pásma. Tento problém sa však môže prejaviť aj v tímoch menšieho rozsahu, keď spolupracujú ľudia s rôznymi osobnými časovými rozvrhmi.

Najdôležitejší je asi najmä spôsob, akým ľudia medzi sebou komunikujú. Ak človek dokáže počúvať a snažiť sa porozumieť myšlienkam toho druhého a zároveň vie

rozprávať bez toho, aby urážal prácu niekoho iného, tak je veľká šanca, že rozhovor prinesie pozitíva obidvom stranám. Priateľskosť rozhovoru podstatne závisí od dôvery medzi ľuďmi. Na druhú stranu, určite nie je dobré, keď obava z toho, aby sme neurazili svojich kolegov, zabráni akémukoľvek vyjadreniu nesúhlasu alebo poukázaniu na objektívne nedostatky, ktoré sa v projekte vyskytujú.

Ako komunikovať

Výber vhodnej technológie a nástrojov, ktoré sa použijú pri komunikácii počas práce na softvérovom projekte, môže výrazne ovplyvniť jeho budúci úspech. Odpoveď na najvhodnejšiu technológiu nie je vôbec jednoznačná – v niektorých prípadoch to môže byť e-mail, inokedy osobné stretnutie alebo napísaný dokument. Navyše, ak sa má používať technológia efektívne, takmer vždy je výhodné použiť najjednoduchšie nástroje, ktoré sa dajú ľahko naučiť ovládať. Niektoré používané technológie a nástroje pre komunikáciu sú vymenované v tabuľke **Tab. 7**.

Technológia	Opis
Kolaboratívne nástroje pre modelovanie	CASE nástroje, ktoré umožňujú vývojárom súčasne pracovať na jednom alebo viacerých modeloch s aktualizáciou modelu v reálnom čase.
Kolaboratívne nástroje pre písanie	Textové procesory umožňujúce viacerým ľuďom písať jeden dokument v rovnakom čase.
Diskusné nástroje	Nástroje ako napríklad e-mail, diskusné fóra, mailing-listy, služby ako instant messaging a chat umožňujúce prenos textu a prípadne súborov medzi ľuďmi.
Všeobecné modely	Modely využívajúce jednoduché nástroje a techniky, ktoré sa dokážu investori ľahko naučiť.
Video konverzácia	Kamera a softvér nainštalované na počítači zabezpečujúce prenos zvuku a obrazu s druhým človekom.
Nástroje na riadenie verzií	Softvérové nástroje používané pre riadenie verzií rôznych artefaktov softvérového projektu.
Nástroje na virtuálne stretnutia	Nástroje, ktoré zabezpečia komunikáciu medzi viacerými ľuďmi, ktorí sa fyzicky nachádzajú na rôznych miestach.

Tab. 7. Komunikačné technológie

Z hľadiska manažmentu je dôležité vedieť vybrať správne spôsoby komunikácie, ktoré napomôžu úspechu softvérového projektu. Hlavným cieľom efektívnej komunikácie je predovšetkým zdieľanie informácií, ktoré je zvyčajne výhodné pre obidve strany. Komunikácia je najefektívnejšia vtedy, keď obidve strany sledujú rovnaký cieľ, pre splnenie ktorého sú schopné navzájom spolupracovať. V prípade, že ľudia neveria pravdivosti alebo užitočnosti informácií, ktoré dostávajú, je cieľ efektívnej komunikácie takmer nesplniteľný.

Výber komunikačnej technológie sa môže v rôznych softvérových projektoch výrazne líšiť. Môže ísť o krátke konverzácie až po dlhotrvajúce stretnutia, prípadne o dokumenty alebo o časové rozvrhy napĺňané údajmi z databázy, ktoré sú dostupné na intranete. Faktory, ktoré ovplyvňujú výber komunikačnej technológie, môžu byť napríklad [47]:

- *Potreba mať informácie okamžite dostupné* – treba zvážiť, či úspech projektu závisí od často aktualizovaných informácií, alebo napríklad stačí písať pravidelné správy.
- *Dostupnosť technológie* – či sú doterajšie technológie vyhovujúce, alebo ich treba nahradiť novými.
- *Očakávané zaučenie ľudí pre prácu s technológiou* – či majú ľudia, ktorí sa podieľajú na softvérovom projekte, dostatočné znalosti a skúsenosti na používanie danej technológie, alebo ich bude treba naučiť s ňou pracovať.
- *Dĺžka projektu* – aká je pravdepodobnosť zmeny používanej technológie počas trvania softvérového projektu takým spôsobom, aby sa oplatilo používať novšiu technológiu.

Záver

Pri práci na softvérovom projekte sa už v súčasnosti využíva veľké množstvo technológií a nástrojov. Táto esej sa zaoberá najmä tými technológiami a nástrojmi, ktoré súvisia s vytváraním a údržbou dokumentácie softvéru ako aj komunikáciou medzi ľuďmi, ktorí sa na softvérovom projekte podieľajú. Je úlohou manažmentu softvérového projektu rozhodnúť, ktoré z dostupných technológií sa budú používať – toto rozhodnutie môže ovplyvniť aj úspech alebo neúspech celého projektu.

Použitá literatúra

45. Andrew Forward, Timothy C. Lethbridge: *The Relevance of Software Documentation, Tools and Technologies: A Survey,* 2005 http://www.literateprogramming.com/doceng.pdf
46. Scott W. Ambler: *Communication on Agile Software Projects,* október 2005 http://www.agilemodeling.com/essays/communication.htm

47. William R. Duncan: *A Guide to the Project Management Body of Knowledge,* 2005 http://egweb.mines.edu/eggn491/Information%20and%20Resources/pmbok.pdf

Annotation

New technologies, tools and their impact on software project management

Complexity of management and development in software projects requires new technologies and software tools. These can be very helpful when designing complex models and are able to automatically generate source code and documentation, and other tools can help us to communicate with people participating in the software project. This essay focuses especially on those tools and technologies, whose purpose is to create and maintain software documentation or to make communication more effective. Project managers can and should choose among the available tools – this decision may have impact on the success of the software project. Which tools and technologies should be used? There is no simple answer to this question.

Manažment konfliktov v tíme

ONDREJ ŽÁRY

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
zary01@student.fiit.stuba.sk

Abstrakt. Keďže sa stále viac vývoja uskutočňuje v tímoch, vzrastá potreba riešenia konfliktov. Základom pre vznik konfliktu sú rozdiely (názorové ale aj iné). Keď sa jednotlivci stretnú v tíme, rozdiely medzi nimi prispievajú k vzniku konfliktov. Vznik konfliktov tiež podporuje nedostatok zdrojov, čo sa pri vývoji softvéru stáva pomerne často. Pri riešení konfliktov sa považuje za dôležité riešiť spory rýchlo a otvorene, aby sa predišlo negatívnym dôsledkom. Konflikt sa bežne chápe ako negatívny jav – nemusí však byť deštruktívny. Ak je správne riešený, jeho výsledok môže byť pre tím prínosom. Pri vývoji softvéru sa konflikty môžu vyskytnúť napríklad medzi vývojármi a koncovými používateľmi alebo medzi programátormi a testermi.

Úvod

Veľkou výhodou tímu oproti jednotlivcovi sú rozdiely v prostriedkoch, znalostiach a nápadoch. Táto výhoda je však súčasne aj nevýhodou, pretože rozdiely sú základom konfliktov. Keďže sa stále viac vývoja uskutočňuje v tímoch, vzrastá potreba riešenia konfliktov. S konfliktmi sa každý z nás stretáva v každodennom živote. Ich riešenie teda intuitívne zvládame. Pri tímovej práci, akou je napríklad vývoj softvéru v tíme, je však potrebné, aby sme konflikty riešili konštruktívne a takým spôsobom, aby negatívny dopad na tím ako celok bol čo najmenší.

Najprv sa budeme zaoberať konfliktmi samotnými, príčinami ich vzniku, metódami ich riešenia a predchádzaním vzniku konfliktov. Ďalej túto teóriu aplikujeme na oblasť vývoja softvéru v tíme.

Konflikty

S konfliktmi sa bežne stretávame. Ale vieme vlastne, čo to konflikt je, aké sú príčiny jeho vzniku, ako ho čo najlepšie riešiť alebo čo robiť, aby vôbec nevznikol? Na tieto otázky sa pokúsia odpovedať nasledujúce riadky.

Čo je to konflikt

Konflikt je situácia, v ktorej existuje nesúlad medzi dvoma stranami (jednotlivcami alebo skupinami ľudí) v nejakej oblasti, pričom v tejto oblasti tieto dve strany majú spolupracovať.

Konflikt v tíme nemusí byť vždy chápaný ako negatívny jav. Konflikt môže priniesť nové nápady a nové prístupy k organizácií a k riešeniu problémov. Môže pomôcť objaviť dôležité problémy a poskytnúť možnosti na zlepšovanie komunikačných schopností [50].

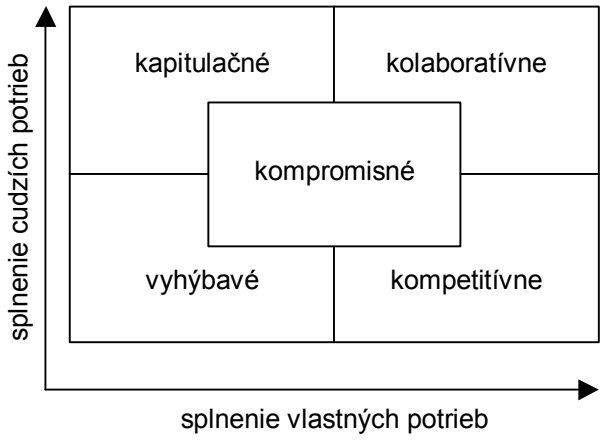
Príčiny vzniku konfliktov

Medzi najčastejšie príčiny vzniku konfliktov v tíme podľa [52] patria:

- nedostatok zdrojov (peniaze, vybavenie, ...),
- rozdiely v názoroch,
- nesúhlasné názory na potreby, ciele a priority,
- zlá komunikácia v tíme,
- zlá organizačná štruktúra,
- chýbajúca tímová spolupráca,
- nejasnosti v rolách a zodpovednostiach.

Riešenia konfliktov

Základom riešenia každého konfliktu je dohoda. Obrázok **Obr. 18** zobrazuje možné spôsoby riešenia konfliktov (dohody) podľa Thomasa Kilmanna [48] a [51].



Obr. 18. Riešenie konfliktov podľa Thomasa Kilmanna [48]

Obrázok ukazuje, ako závisí spôsob riešenia konfliktu od miery dôležitosti vlastných a cudzích potrieb. Žiadny z týchto piatich spôsobov riešenia konfliktov sa nepovažuje za zlý ani za dobrý – voľba najvhodnejšieho spôsobu závisí vždy od konkrétnej situácie.

Podľa [51] sú najproblematickejšie dva druhy dohody – distributívna a integratívna – a preto sa im ďalej venuje:

Distributívna dohoda (kompetitívna, win-lose)

- jedna strana „vyhrá“ a druhá „prehrá“,
- sú pevné zdroje, ktoré sa musia rozdeliť medzi dve strany – čím viac dostane jedna strana, tým menej dostane druhá,
- záujmy jednej strany sú opačné ako záujmy druhej,
- hlavným cieľom je maximalizovanie vlastných záujmov,
- hlavné stratégie: manipulácia, nátlak a zadržiavanie informácií.

Integratívna dohoda (kolaboratívna, win-win, vytváranie hodnoty)

- obe strany môžu vyhrať – zdroje sú variabilné a môžu sa rozdeliť medzi ne,
- hlavným cieľom je maximalizovať spoločný zisk oboch strán,
- hlavné stratégie: spolupráca, zdieľanie informácií a spoločné riešenie problémov,
- tento typ sa tiež označuje ako *vytváranie hodnoty,* pretože cieľom je, aby obe strany odchádzali zo stretnutia s tým, že majú viac ako predtým.

Situácie v reálnom svete však nie sú také jednoduché – často sú kombináciou viacerých druhov. Príkladom môže byť určovanie ceny – predajca a kupujúci majú čiastočne opačné záujmy (predajca chce čo najvyššiu cenu, kupujúci najnižšiu), no aj čiastočne spoločné (obe strany majú záujem, aby sa produkt predal).

Kľúčom k úspešnému riešeniu konfliktu je posunúť situáciu z win-lose na win-win. Takmer každá situácia má isté aspekty win-win riešenia, len je často potrebné ich hľadať. Win-lose riešenie je len posledná možnosť.

Podľa [49] pomáhajú pri prekonávaní konfliktov tieto body:

1. útočiť na problém, nie na osobu,
2. zamerať sa na to, čo je možné urobiť – nie na to, čo možné nie je,
3. podporovať rozdielne pohľady na vec a čestný dialóg,
4. vyjadrovať svoje pocity bez obviňovania druhých,
5. uznať svoj podiel na probléme,
6. počúvať a porozumieť pohľadu na vec druhej osoby pred vyjadrením svojho,
7. rešpektovať pohľad na vec druhej osoby,

8. riešiť problém a pritom budovať vzťahy.

Predchádzanie konfliktom

Najjednoduchšie riešenie je žiadne riešenie. Preto najjednoduchšie riešenie konfliktov je predchádzať im. Predchádzať problémom v tíme je možné pomocou dobrej tímovej spolupráce a dobrou prácou manažéra.

Podpora tímovej spolupráce

Tímová spolupráca je nevyhnutná pre efektívnu prácu tímu ako celku. Tímová spolupráca vychádza zo spoločného cieľa a všetci členovia tímu musia vedieť, aká je ich úloha pri dosahovaní tohto cieľa. Pre dobrú tímovú spoluprácu a minimalizovanie rizika vzniku konfliktu je podľa [52] dôležité, aby všetci členovia tímu:

- zdieľali informácie – informovali včas a korektne všetkých členov tímu o problémoch,
- vyjadrovali, čo očakávajú od iných členov tímu,
- podporovali sa navzájom – oceňovali zásluhy kolegov a povzbudzovali ich k dosahovaniu dobrých výsledkov,
- zlepšovali tím – podporovaním morálky a chránením reputácie tímu,
- riešili potenciálne konflikty – otvorene vyjadrili rozdiely v názoroch.

Konflikty pri vývoji softvéru v tíme

Pri vývoji softvéru je množstvo príležitostí na vznik konfliktu. Ide hlavne o konflikty medzi vývojovým tímom a používateľmi a konflikty medzi programátormi a testermi v rámci tímu.

Konflikty medzi vývojovým tímom a používateľmi

Takéto konflikty vznikajú tak, že vývojový tím má na produkt iný názor ako používatelia tohto produktu. Toto môže vzniknúť ako dôsledok nedostatočnej komunikácie medzi vývojármi a používateľmi – vývojári si myslia, že vedia najlepšie, čo používateľ chce a čo je pre neho dobré. Príkladom vzniku takéhoto konfliktu je zmena používateľského rozhrania v každej novej verzii produktu bez toho, aby na to boli vážne dôvody. V praxi je toto prípad produktu Microsoft Office – v každej z verzií 97, 2000, XP, 2003, 12 sa zmenilo používateľské rozhranie, používatelia si musia zvykať stále na nové prvky a často s nimi doslova bojovať, pretože mnohokrát prácu neuľahčujú, ale naopak znepríjemňujú. Používatelia, zvyknutí na to, že produkt sa ovláda a správa stále rovnako, môžu konflikt vyriešiť veľmi jednoducho – prechodom na konkurenčný produkt (riešenie win-lose) za predpokladu, že taký existuje.

Konfliktom medzi vývojármi a používateľmi je najlepšie predchádzať, pretože nebývajú veľmi konštruktívne a môžu viesť k poškodeniu dobrej povesti softvéru a následným stratám. Predchádzať týmto konfliktom je možné jedine komunikáciou s používateľmi – je potrebné si uvedomiť, že produkt sa vyvíja pre nich.

Konflikty medzi programátormi a testermi

Tieto konflikty vznikajú najčastejšie z nedostatku zdrojov, hlavne z nedostatku času. Testovanie nasleduje po implementácii, takže testeri a programátori sa často musia deliť o spoločný zdroj – čas, ktorého nikdy nie je dosť. Keďže testovanie býva posledná fáza vývoja softvéru pred odovzdaním zákazníkovi, prenášajú sa do nej všetky oneskorenia z predchádzajúcich fáz. V snahe dobehnúť oneskorenie nasleduje tlak na testerov, aby testovanie ukončili čo najskôr. Už tento samotný tlak môže byť príčinou vzniku konfliktu, keďže testeri si uvedomujú dôsledky nedostatočného testovania. Urýchlenie testovania môže mať za následok neobjavenie závažných chýb, ktoré následne môže objaviť zákazník – z toho majú testeri problémy a môže vzniknúť ďalší konflikt – konflikt so zákazníkom.

Dobrý manažér takéto konflikty riešiť nemusí, môže im predchádzať. Jeden možný spôsob je plánovať projekt s dostatočnou časovou rezervou, resp. vyhradiť viac času na testovanie, aby aj v prípade oneskorenia bol dostatok času na kvalitné otestovanie. V prípade, že už k oneskoreniu došlo a času nie je dostatok, môže byť vhodnejšie s týmto problémom oboznámiť zákazníka a vysvetliť mu, že je aj v jeho záujme, aby sa produkt dostatočne otestoval.

Záver

Konflikt môže mať aj pozitívne výsledky – môže nájsť problémy a tiež možnosti ich riešenia. Úlohou manažéra je minimalizovať výskyt konfliktov s negatívnymi výsledkami – buď konfliktom predchádzať alebo ich pretvárať do produktívnej podoby. Tu najviac pomôžu praktické skúsenosti z úspešných aj neúspešných projektov a množstva vyriešených konfliktov.

Použitá literatúra

48. Starrett, E.: Highpoints From the Amplifying Your Effectiveness Conference http://www.stsc.hill.af.mil/crosstalk/2003/02/Starrett.html
49. Swales, Cheri: Overcome Team Conflict http://management.monster.com/articles/conflict/
50. Townsley, Carole A.: Resolving Conflict in Work Teams http://www.rvarmstrong.com/ResolvingConflictInWorkTeamsArticle.htm
51. Wertheim, E.: Negotiations and Resolving Conflicts: An Overview http://web.cba.neu.edu/~ewertheim/interper/negot3.htm

52. Education and Training Unit (ETU): Conflict Management
http://www.etu.org.za/toolbox/docs/building/conflict.html

Annotation

Team Conflict Management

The essay discusses conflict management in general – conflicts, their causes, solutions and prevention. Next, the conflict management theory is applied to team software development and explained on two examples – conflict between developers and end user and conflict between programmers and testers.

Zhrnutie

Tento zborník esejí sa zaoberá rôznymi témami z oblasti manažmentu v softvérovom inžinierstve. Vzhľadom na obmedzené zdroje nedokáže pokryť túto problematiku úplne, avšak snaží sa zachytiť tie najzaujímavejšie a najdôležitejšie časti.

Autori ponúkajú nielen všeobecné fakty o zvolenej problematike, ale snažia sa do nej vniesť aj vlastný názor a skúsenosti. Tiež sa snažia ju aplikovať na tím, s ktorým sa stretávajú v predmete Tvorba softvérových systémov v tíme. V tomto prípade nie je možné aplikovať všetky metódy alebo len v pozmenenej verzií, ale aj napriek tomu môžu byť tieto techniky užitočné aj v tomto prípade.

V dnešnej dobe sa do vývoja softvéru snažia presadiť agilné metódy vývoja na čele s extrémnym programovaním. Preto sa niektoré eseje zamerali aj na vzťah manažmentu a agilných metód alebo na vplyv použitia agilných metód pre nimi zvolenú oblasť manažmentu.

Vzhľadom na rozmery súčasných softvérových systémov je nutné, aby na vývoji spolupracovalo väčšie množstvo ľudí, čiže tím. Niektoré z esejí sa venujú práve tímu, jeho vývoju alebo prípadným problémom v tíme, ako napríklad konflikty medzi jednotlivými členmi.

Je jasné, že kvalita výrobku je dôležitým faktorom výsledného produktu, a preto sa viacero tém či už priamo alebo len okrajovo, zaoberalo kvalitou softvérového produktu. Tu sa snažili identifikovať rôzne vplyvy, ktoré môžu kvalitu produktu zvýšiť alebo naopak znížiť.