Eseje o manažmente v softvérovom inžinierstve

Slovenská technická univerzita
Fakulta informatiky a informačných technológií

Eseje o manažmente s softvérovom inžinierstve

Edícia a grafická úprava
Bc. Michal Šefara

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
2006

Editor

Šefara Michal

Prispievatelia

Beňo Jozef	Májek Ján
Darula Martin	Tatranský Tomáš
Fiflík Andrej	Škovran Ivan
Komara Martin	Velický Dušan
Kriška Jozef	Wagner Jozef

Publikácia neprešla jazykovou úpravou

Predslov

Publikácia, ktorá sa dostala do vašich rúk, sa zaoberá riešením problémov súvisiacich s riadením softvérových projektov a ľudí podieľajúcich sa na vývoji softvéru. Pozostáva z esejí študentov prvého ročníka inžinierskeho štúdia roku 2005/2006 na odbore Softvérové inžinierstvo Fakulty informatiky a informačných technológií Slovenskej technickej univerzity v Bratislave. Tieto dokumenty, rovnako ako zborník v ktorom sú zhrnuté, tvoria súčasť semestrálnej práce ku predmetu Manažment v softvérovom inžinierstve.

Našou snahou je priblížiť potrebu manažmentu a riadenia ako systémových činností a z viacerých pohľadov rozobrať aplikovanie manažmentu na tvorbu softvéru. V jednotlivých častiach tohto zborníka sú diskutované problémy ako plánovanie, riadenie ľudských zdrojov, kvalita softvérového výrobku či riadenie konfliktov v tíme. Pre spísanie takejto práce bolo potrebné preštudovať viacero dokumentov a zozbierať mnoho podkladov, slúžiacich ako základ pre rozvíjanie našich myšlienok. Literárne zdroje súvisiace s danou témou sú uvedené v samostatnej časti na konci každej kapitoly.

Veríme, že v tomto zborníku nájdete práve tú informáciu, ktorú hľadáte, a že po jeho prečítaní vás obohatíme prinajmenšom o zaujímavý zážitok.

Michal Šefara

Obsah

Úvod

Práca v ľudskom kolektíve patrí medzi tie najobtiažnejšie činnosti vykonávané človekom. Nie je samozrejmosťou, že ju dokáže robiť ktokoľvek. Mnohí z nás takúto prácu vykonávajú, no len malé množstvo ľudí má z práce v kolektíve naozaj potešenie. Stáva sa skutočnosťou dnešných dní, že jednotlivec sám mnoho nezmôže, preto sa vytvárajú pracovné skupiny, ktoré majú spoločný cieľ snaženia. Pokiaľ chceme nejaký skupinový cieľ dosiahnuť a chceme ho dosiahnuť efektívne, musíme postupovať systematicky a uvážene. Pre systematický postup skupiny ľudí je teda potrebná koordinácia a riadenie, iným slovom manažment.

V odbornej literatúre sa stretávame s rozličnými definíciami pojmu manažment. Z nášho pohľadu je najvýstižnejšia definícia Mary Parker Follerrovej, ktorá hovorí že: „Manažment je umenie dosiahnuť stanovený cieľ prostredníctvom iných ľudí.“ Úlohou manažéra je teda plánovanie, organizácia, vedenie ľudí a koordinácia činností pre dosiahnutie určitého cieľa. Plánovanie ako také sme pochopili ako dynamický proces, v priebehu ktorého manažér rozhoduje aktivitách, ktoré je potrebné uskutočniť v blízkej alebo vzdialenejšej budúcnosti. Organizácia je činnosť manažéra zameraná na tvorbu štruktúry vzťahov medzi podriadenými, pričom tvorí jeden z rozhodujúcich predpokladov efektívneho využívania zdrojov riadenej skupiny. V rámci vedenia ľudí nehovoríme iba o usmerňovaní, ale i o motivácií jednotlivých členov skupiny, tak aby ich konanie smerovalo k dosiahnutiu stanovených cieľov.

Nastolenému trendu sa nevyhneme ani pri vytváraní softvéru. Len máloktorý úspešný softvérový produkt, je ako celok vytváraný jediným človekom. Takmer vždy sa pracuje v tíme, pričom tímy mnohokrát narastajú do gigantických rozmerov. Pri takýchto projektoch vzniká potreba hierarchizácie spolupracovníkov, ako i hierarchizácie riadenia. V hierarchii má každý jednotlivec presne stanovenú pozíciu a pracovnú zodpovednosť. Ukázalo sa, že i pri malých tímoch je vhodné presne stanoviť kompetencie jednotlivcov, aby sa v čo najväčšej miere predišlo konfliktom a stretu záujmov.

V našej práci sme populárnym spôsobom zhrnuli niekoľko kľúčových tém, súvisiacich s tvorbou softvérových produktov. Na tomto mieste sa ich pokúsime stručne priblížiť, aby ste získali prehľad čo v nej môžete nájsť.

V prvej časti sa zaoberáme témou, ako vytvoriť a riadiť úspešný tím počas celého trvania softvérového projektu. Táto časť pojednáva najmä o jednotlivých etapách vývoja tímu riešiaceho softvérový projekt. Snaží sa analyzovať vývoj z pohľadu manažmentu a identifikovať niektoré jeho aspekty. Zaoberáme sa i možnými dopadmi rozhodnutí manažmentu na vývoj tímu v jednotlivých fázach. Identifikujú sa najmä jednotlivé etapy vývoja softvérového tímu a analyzujú sa aspekty zostavovania tímu vzhľadom na odborné a charakterové vlastnosti jeho potenciálnych členov ako aj vzhľadom na veľkosť tímu.

V ďalšej časti sa zaoberáme odhadovaním softvérových projektov. V tejto oblasti je softvérové inžinierstvo stále veľmi neisté. Pri veľkých projektoch je skôr výnimkou, ak sa odhad čo len približne podobá výslednému úsiliu. Pri menších projektoch je situácia lepšia. V práci uvádzame niektoré možnosti zlepšenia odhadovania pre projekty, ktorých členmi sú aj menej skúsení riešitelia. Ide o použitie kontrolného zoznamu, ktorý slúži na pripomenutie dôležitých častí produktu a aktivít projektu, aby neboli vynechané pri tvorbe odhadu rozsahu a potrebného úsilia. Často sa stáva, že pri plánovaní softvérových projektov predpokladáme priebeh podľa navrhnutého plánu. Takmer pri každom projekte však narážame na nejaké neočakávané udalosti, ktoré zásadným spôsobom ovplyvňujú priebeh projektu. Prieskumy poukazujú na to, že veľa neočakávaných udalostí je predvídateľných a preto sa manažment rizík ktorý sa týmito udalosťami zaoberá stáva neoddeliteľnou súčasťou úloh manažmentu. Zaoberáme sa otázkou dôležitosti rizík a ich kategorizáciou.

Vplyv koordinácie procesu vývoja softvéru na produktivitu softvérového tímu hovorí o tom, že dokonca aj dnes, problémy so softvérovými systémami sú bežné a často publikované. Hlavným dôvodom je, že s narastajúcou zložitosťou projektu sa koordinácia aktivít stáva komplikovanejšou. V tejto časti dokumentu sme preskúmali úlohu formálnej a neformálnej komunikácie pri koordinovaní práce na softvérových projektoch. Vysoká zložitosť softvérových systémov viedla ku vzniku veľkého množstva prístupov a metód, ktoré sa snažili o systematický prístup k tvorbe softvéru. Napriek tomu len málo systémov bolo dodaných do prevádzky za vopred dohodnutých podmienok. V situácii, keď sa zvyšuje tlak, ktorý núti vytvárať kvalitný softvér za menej peňazí a za kratší čas, vznikli nové, tzv. agilné metódy. Agilné metódy vznikli zo skorších prístupov k rýchlemu vývoju aplikácií (RAD), ktoré rozpoznali, že spomedzi obmedzení projektu tvorenými časom, cenou a rozsahom má práve rozsah najvyššiu mieru neurčitosti. Preto sa agilné metódy na rozdiel od tých tradičných nepokúšajú určiť rozsah projektu v úvodných fázach, ale pracujú s používateľskými požiadavkami oveľa flexibilnejšie. Tento dokument teda v časti agilné metódy vývoja softvéru a rozsah projektu rozoberá nové zaujímavé postupy na určovanie rozsahu projektu pri použití agilných metód vývoja softvéru.

Každý softvér sa počas svojho vývoja musí vysporiadať so vznikom chýb. Vznik chýb je neodvratný pri každej ľudskej činnosti, preto je nutné zaoberať sa problémom vzniku chýb a snažiť sa o ich odstránenie. V tomto dokumente sa zaoberáme rozborom kritických miest softvéru, pri ktorých často vznikajú chyby. Zaoberáme sa i možnosťami manažmentu ako predchádzať vzniku chýb a ako zabezpečiť úspešnosť softvérového projektu.

V časti kvalita, výsledok plánovania a riadenia sa venujeme riadeniu a plánovaniu kvality v životnom cykle softvéru. Vysvetľujeme, čo je to kvalita, aký rôzny význam má pre zákazníka a výrobcu a aký jej význam v postupe času. Venujeme sa dvom stratégiám riadenia pričom jedna metóda je určená veľkým a druhá malým tímom.

Medzi časté javy v tímovej spolupráci patrí pravidelný výskyt konfliktov medzi členmi tímu. Aby sa zamedzilo eskalácii konfliktu je potrebné riešiť hádky rýchlo

a otvorene. Štatistiky hovoria, že väčšina manažérov si uvedomuje existenciu nezhôd v tíme a zároveň absolvovali tréning v oblasti riešenia medziľudských vzťahov no iba zriedkakedy dávajú vyššiu prioritu riešeniu týchto konfliktov. Preto je potrebné aby jednotliví členovia tímu ovládali techniky pre riešenia konfliktov a priamo ich aplikovali medzi sebou. V tejto práci sú objasnené príčiny, priebeh a niektoré zo spôsobov riešenia konfliktov v tímoch. Pre riešenie konfliktov je veľmi dôležitá komunikácia. Komunikácia vo vnútri lokalizovaných, ako aj distribuovaných tímov je i súčasťou manažmentu a jedným z kľúčových faktorov úspechu pri tvorbe softvérových produktov. Tento dokument sa v časti manažment komunikácie zaoberá časťou plánovania komunikácie a distribúcie informácií, pričom hlavný dôraz kladie na definovanie vhodných spôsobov komunikácie pre jednotlivé úlohy, činnosti a role v tímoch. Dokument približuje teóriu mediálnej bohatosti a jej praktické testy, ktoré boli v posledných rokoch uskutočnené a ktoré potvrdili jej správnosť. Pri vzájomnej komunikácií jednotlivých členov tímu môžu vzniknúť kolízie a nedorozumenia. Je preto vhodne vytvoriť komunikačný manuál. Medzi jeho výhody určíte patri stanovenie komunikačnej matice v rámci hierarchie o ktorej sme písali na úvod tejto kapitoly. Komunikačná matica teda tvorí nariedenie, kto s kým môže v rámci tímu komunikovať, akými prostriedkami, na akej úrovni, a či má dotyčný na túto komunikáciu vôbec oprávnenie.

Pokiaľ ste našli v tomto zhrnutí niečo, čo vás zaujalo, neváhajte a pokojne preskočte na kapitolu, ktorá má pre vás osobne najväčšiu informačnú hodnotu.

Ako vytvoriť a riadiť úspešný tím počas celého trvania softvérového projektu

JÁN MÁJEK

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

Abstrakt. Esej pojednáva o jednotlivých etapách vývoja tímu riešiaceho softvérový projekt. Snaží sa analyzovať tento vývoj z pohľadu manažmentu a identifikovať niektoré aspekty tohto vývoja. Pojednáva o možných dopadoch rozhodnutí manažmentu v jednotlivých fázach na vývoj tímu.

Autor sa zaoberá najmä identifikáciou jednotlivých etáp vývoja softvérového tímu, analyzuje aspekty zostavovania tímu vzhľadom na odborné a charakterové vlastnosti jeho potenciálnych členov, ako aj vzhľadom na veľkosť tímu. Ďalej sa autor zaoberá potrebou vybudovať dobrú komunikačnú štruktúru v rámci tímu, problematikou motivácie členov tímu, ako aj budovania dobrých medziľudských vzťahov. Nakoniec sa autor zaoberá problematikou organizačných zmien v rámci tímu.

Úvod

Úspešný vývoj softvéru už dávno nezávisí len na odborných schopnostiach vývojárov. Potreba vyvíjať rozsiahlejšie aplikácie a systémy si vynútila celý rad požiadaviek na riadenie softvérového projektu, okrem iného najmä v oblasti manažmentu ľudských zdrojov. Dobré zvládnutie tejto činnosti v každej etape životného cyklu softvérového projektu tvorí jeden z hlavných predpokladov úspešnosti projektu ako celku.

Schopnosti projektového manažéra zostaviť dobrý tím, zabezpečiť vhodný spôsob komunikácie, dostatočne motivovať členov tímu, či riešiť konflikty medzi nimi zásadným spôsobom vplývajú na výsledok projektu.

Softvérový projekt prechádza niekoľkými štádiami vývoja, rovnako tak aj členovia tímu. Osobnostné a odborné vlastnosti jednotlivých členov tímu, ich vzájomná komunikácia a vzájomné vzťahy prechádzajú počas práce na softvérovom projekte neustálymi zmenami. Závisí od manažéra, či dokáže včas a vhodne reagovať na tieto zmeny a správne ich usmerňovať.

Etapy vývoja tímu

Softvérový tím zväčša prechádza niekoľkými etapami vývoja. Tieto sú podľa [1] uvedené v tabuľke 1 spolu s charakteristikou každej etapy.

Etapa	Charakteristika
Formovanie	– identifikácia úloh a cieľov tímu – analýza úloh a možností riešenia – odhady prácnosti, rozdelenie práce – vytvorenie podtímov
Kryštalizácia	– vymedzovanie a spresňovanie jednotlivých čiastkových úloh a cieľov (a diskusie o nich) – diskusie a spory o spôsoboch riešenia – mocenské zápasy členov tímu o postavenie v tíme – formovanie podtímov podľa rozsahu prác
Vyjasnenie	– prijatie zásad riešenia – prijatie noriem a pravidiel práce – návrh štruktúry tímu – prijatie cieľov členmi tímu a ich súhlas – vytváranie súdržnosti v tíme
Realizácia	– definitívne stanovanie štruktúry tímu a podtímov – definitívne prijatie rolí – vlastné vykonanie úloh
Ukončenie	– ukončenie úloh (často súvisí s ukončením projektu)

Tabuľka 1 - Etapy vývoja softvérového tímu

Nie pri každom projekte je jednoznačné takéto delenie na etapy. V konkrétnych prípadoch sa často krát jednotlivé fázy navzájom prelínajú a z hľadiska manažmentu im často predchádzajú prípravné fázy.

Fáza budovania tímu

Veľmi dôležitým aspektom je, či projekt má alebo nemá vlastného projektového manažéra. Ukázalo sa, že tento fakt zásadným spôsobom môže ovplyvniť úspešnosť celého projektu. Podľa prieskumu zhuba 5% projektov nemá projektového manažéra. Zväčša ide o menšie tímy (do siedmich členov). Jeden z väčších projektov (zhruba 100 pracovníkov), ktorý zlyhal, nemal projektového manažéra [2].

Najmä v malých firmách sa môže stať, že tímy sú tvorené systémom "ad-hoc" (podľa potreby) a ak je určená firemná hierarchia, zamestnanec v roli projektového manažéra má často aj iné pridelené role. Niekedy je dokonca sám členom tímu v pozícií analytika, návrhára, programátora alebo inej. To samo o sebe nemusí predstavovať problém, najmä pri malých projektoch a tímoch. Môže to byť dokonca výhoda v tom zmysle, že manažér "žije s tímom", takže dokáže zachytiť a riešiť prípadné problémy či konflikty už pri ich vzniku. Myslím si však, že najmä pri väčších projektoch by mal byť určený projektový manažér, ktorý sa venuje len tejto činnosti a má všeobecný prehľad o dianí v tíme a v projekte.

Podľa prieskumu [2] by dobrý projektový manažér mal mať všeobecný prehľad o problematike, na druhej strane nie je nutné, aby bol odborníkom v niektorej konkrétnej oblasti riešeného problému. Mal by mať dobre rozvinuté komunikačné schopnosti a asertívne vystupovanie. Tak dokáže konflikty riešiť, nie vyvolávať. Dobrý projektový manažér by mal mať vytvorenú víziu o cieľoch a spôsobe napredovania projektu. Jeho postoj by mal byť prevažne optimistický, aby dokázal motivovať jednotlivých členov tímu k požadovaným výkonom.

Ideálny tím je zložený z členov, ktorí sa vzájomne dopĺňajú v oblasti osobnostnej aj odbornej. Vhodná voľba členov tímu ešte vo fáze plánovania je veľmi dôležitý faktor úspechu celého projektu. Na to má nemalý vplyv firemná politika v oblasti získavania ľudských zdrojov. V čase začiatku projektu je už neskoro hľadať správnych ľudí do tímu. Noví zamestnanci spravidla potrebujú aspoň zopár mesiacov na to, aby sa zžili s odbornými témami a postupmi v rámci nového pôsobiska, nehovoriac o "zapadnutí" do celkovej firemnej atmosféry v rámci medziľudských vzťahov.

Tímový manažér by mal dostatočne poznať individuálne charakteristiky a schopnosti potenciálnych členov tímu. Ide rovnakou mierou aj o odborné aj o osobnostné vlastnosti. V dobrom (a teda aj úspešnom) tíme by mali byť ľudia s viacerými, navzájom sa dopĺňajúcimi charakterovými črtami. Niektoré základné využiteľné role uvádzam v tabuľke 2, niektoré nežiaduce role v tabuľke 3, ostatné môže čitateľ nájsť napríklad v [1]. Je zrejmé, že jednotliví členovia tímu disponujú kombináciu týchto charakterov. Navyše,

niektoré charakterové črty sa môžu prejaviť až po určitom čase. Manažér sa musí pokúsiť jednotlivé osobnosti v tíme vhodne zladiť, čo väčšinou býva veľmi náročné, často až nemožné.

Rola	Charakteristika
Koordinátor	Dáva veci do širších súvislostí, objasňuje vzťahy, zhŕňa znalosti, dokáže koordinovať činnosti, ktoré navzájom súvisia a tiež dokáže takéto činnosti definovať.
Navigátor	Schopný hodnotiť, či sa riešenie neodchyľuje od cieľov a ide správnym smerom.
Ťahúň	Formuje spôsob, akým tím napne sily, dokáže vybudiť aktivitu, zameriava pozornosť na ciele.
Harmonizátor	Dokáže znižovať napätie v tíme, podporovať rozvoj dobrých vzťahov, ukludňovať spory a hľadať vhodné kompromisy.
Realizátor	Z konceptov a plánov robí praktické pracovné postupy, efektívne a systematicky vykonáva dohovorené plány.
Ukončovač	Má zmysel pre naliehavosť, nepostrádateľný pri ukončovaní a kompletizácií projektu.

Tabuľka 2 -Niektoré využiteľné role a ich charakteristika

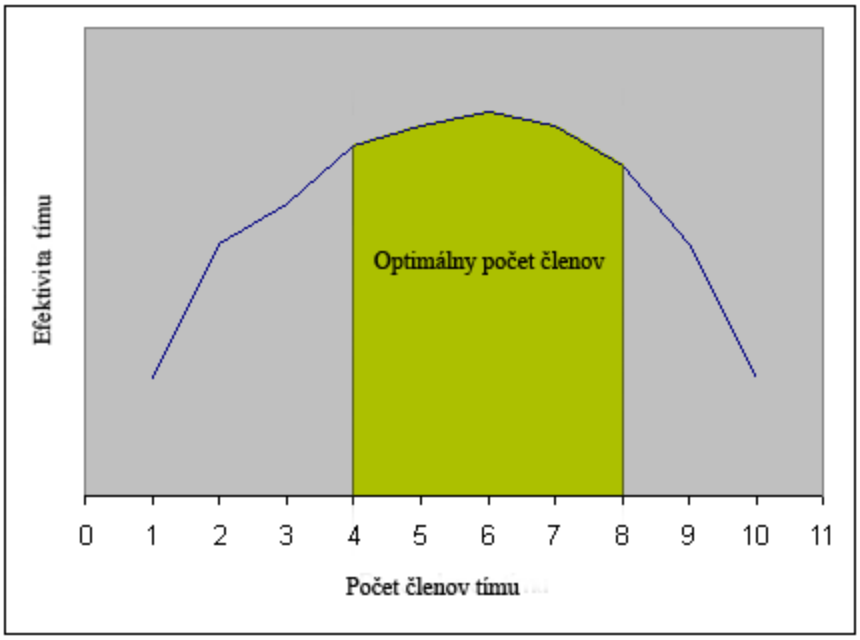
Rola	Charakteristika
Agresor	Závidí, deštruktívne nesúhlasí, bezohľadne útočí na kolegov v pracovných aj osobných záležitostiach.
Negativista	Cieľom je zápor za každú cenu, spochybňuje dohodnuté.
Kecal	Neustále rozpráva a zdržuje.
Vládca, diktátor	Autoritatívny, často nedodrží slovo, ohovára ostatných.

Tabuľka 3 - Niektoré nežiaduce role a ich charakteristika

Vo väčších firmách je dobré, ak sú v informáciách o zamestnancoch zaznamenané aj charakterové črty (zamestnanci často už pri prijímacom pohovore absolvujú psychologické testy). Manažér má potom pri zostavovaní nového tímu uľahčenú robotu. Nesmieme však podliehať predstave, že každú osobu, resp. osobnosť možno zaradiť do jednej kategórie podľa určitej šablóny a tím potom zostaviť na len základe týchto informácií. Takéto informácie by mali mať len pomocnú úlohu pri zostavovaní tímu. Dobrý manažér by mal svojich ľudí poznať a vedieť vhodne využiť ich vlastnosti (tak technické ako aj charakterové) vo svoj prospech a tým pádom aj v prospech celého tímu.

Je výhodou, ak je možné sformovať tím z ľudí, ktorí už predtým spolupracovali na nejakom projekte. Takýto ľudia sa navzájom poznajú a vedia, čo môžu od seba navzájom očakávať. Navyše, v ak tím už predtým dosahoval dobré výsledky, je pravdepodobné, že sa "obrúsili" najostrejšie hrany a tím nebude brzdený počas práce na projekte tým, že sa u niektorého člena prejaví neočakávaná (negatívna) vlastnosť.

Formovanie tímu závisí do istej miery od veľkosti tímu. Niektoré zdroje (napríklad [1]) uvádzajú, že ideálny počet členov tímu je štyri až osem. Tento údaj bol získaný empiricky a podľa mňa do určitej miery závisí od charakteru a veľkosti samotného projektu. Ide o takzvaný „mýtus človeko-mesiaca", ktorý hovorí, že pridávaním ľudí do projektu nad určitú hranicu klesá efektivita tímu a trvanie projektu sa predlžuje [4]. Závislosť efektivity tímu od počtu členov je znázornená na obrázku 1. Pri projektoch, ktoré by vyžadovali väčší tím, môže projektový manažér siahnuť k vytvoreniu hierarchie podtímov (napríklad podľa špecializácie jednotlivých členov a vývojových fáz projektu: analytický tím, návrhový tím, tím programátorov, testovací tím a dokumentačný tím).



Obrázok 1 - Závislosť efektivity tímu od počtu členov

Dôležité pri formovaní tímu je aj to, či jednotlivý členovia budú pracovať výlučne na danom projekte, alebo budú zdieľať prácu na viacerých projektoch. Vo väčšine najmä malých firiem je bežnou praxou, že jeden zamestnanec pracuje na rôznych projektoch,

ktoré sa riešia súčasne. V takom prípade je dobré, ak je vytváranie tímu konzultované na vyššej úrovni manažmentu aj s ostatnými projektovými vedúcimi, najmä s ohľadom na očakávané vyťaženie jednotlivých potenciálnych členov tímu prácou na ostatných projektoch.

Komunikácia v tíme

V súčasnosti nie je výnimkou, ak zamestnanci (členovia tímu) nepracujú na jednom fyzickom mieste (napríklad každý žije v inom meste, alebo dokonca štáte). V takom prípade vznikajú zvýšené nároky na zabezpečenie dobrej komunikácie, pretože prieskumy ukazujú, že tá zaberá pomerne veľkú časť pracovného času [1].

Samozrejme, dobrý projektový manažér sa nesnaží komunikáciu potlačiť, keďže sám by mal vedieť, že tá je základom, ale mal by sa snažiť minimalizovať zbytočnú komunikáciu. Pri menších tímoch je bežné, že komunikuje každý s každým, no takáto komunikácia pri väčších tímoch, zvlášť ak existujú aj rôzne podtímy, nie je vhodná (ide o jeden z dôsledkov "mýtusu človeko-mesiaca").

Manažér by mal niekedy v čase "vyjasnenia" zadefinovať komunikačné kanály v rámci tímu. Jednak by mal určiť vhodné médiá na elektronickú komunikáciu (e-mail, komunikácia v reálnom čase cez niektorú IM službu, telefón, prípadne VoIP) a určiť časové body, kedy sa bude tím stretávať osobne. Ďalej je potrebné určiť jasné pravidlá, kto s kým má komunikovať a ktorí členovia sa musia zúčastňovať osobných stretnutí a aké ďalšie úlohy z toho pre nich vyplývajú (typicky ide o kľúčových členov podtímov).

Motivácia a koordinácia tímu

Veľmi podstatná vlastnosť dobrého manažéra je jeho schopnosť vhodne motivovať ľudí tak, aby podávali dobré výsledky. Motivačné stimuly môžu byť pozitívne alebo negatívne. Za pozitívne stimuly považujeme také, ktorých cieľom je získanie niečoho, čo má pre konkrétneho človeka význam. Môže ísť napríklad o dosiahnutie prestíže či uznania, no môžu to byť napríklad aj spoločné stretnutia. Za negatívne stimuly považujeme také, ktoré v človeku vyvolávajú potrebu vyhnúť sa niečomu nepríjemnému. Je na manažérovi, aby počas vývoja softvérového tímu dokázal nachádzať vhodné motivačné stimuly pre jednotlivých členov tímu. Treba však dať pozor, aby nedošlo k premotivovaniu, zvlášť negatívnymi stimulmi, ktoré bývajú nepríjemné a nesú so sebou riziko motivačnej krízy [2].

Ďalšou dôležitou vlastnosťou projektového manažéra je jeho schopnosť koordinácie tímu. Táto potreba v rámci tímu výrazne závisí od schopnosti tímu učiť sa.

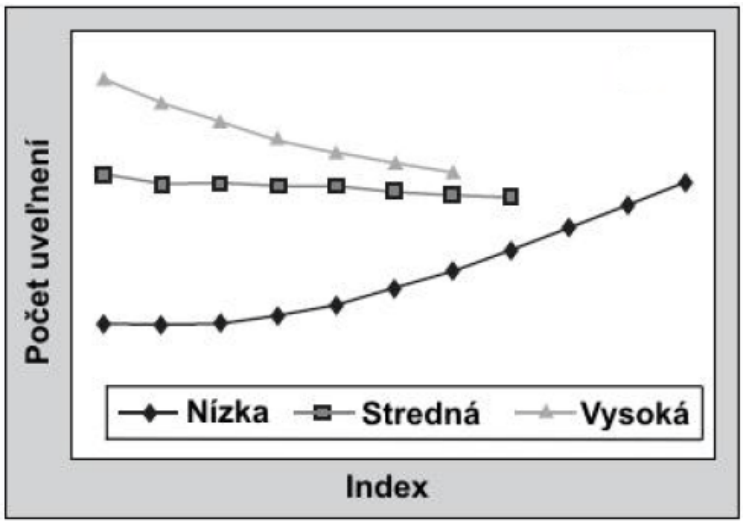
Produktivita tímu závisí aj skúseností jeho členov. Niektoré štúdie preukázali, že je až desaťnásobný rozdiel v produktivite neskúsených programátorov začiatočníkov

a skúsených expertov [3]. Vlastnosťou dobrých vývojárov je ich schopnosť získavať znalosti z projektovej domény a tým sa vyhnúť zbytočnému prepracovaniu a to dokonca aj v neznámom projektovom prostredí.

Tímy, ktoré dokážu stabilizovať moduly veľmi rýchlo, nemusia podliehať príliš častej koordinácií už v skorších fázach projektu. Naopak, s rastúcou voľnosťou je potrebná zvýšená koordinácia, až pokiaľ nie je dosiahnutá dostatočná stabilita a spoľahlivosť. Z toho vyplýva, že neskúsený tím vytvára potrebu častejšej koordinácie a dosahuje stabilitu pomalšie. Tento proces učenia tímu sa prejavuje tým, že vývojový tím stabilizuje novo vyvíjané moduly efektívnejšie ako predošlé, čím dochádza k zvyšovaniu schopností tímu a celkovému rýchlejšiemu pokroku v projekte.

Úroveň koordinácie tímu však závisí aj od iných faktorov. Významne na túto potrebu vplýva napríklad zložitosť samotného systému. Platí, že čím je vyvíjaný projekt zložitejší, tým vyššia by mala byť koordinácia tímu zo strany manažmentu. Táto závislosť však nie je priamo úmerná. V zložitejších systémoch sa chyby jednotlivých modulov navzájom ovplyvňujú a ich oprava je rádovo náročnejšia. Dobrá koordinácia pomáha prechádzať týmto problémom a umožňuje udržať výskyt chýb pod určitou kritickou hranicou.

Obrázok 2 znázorňuje optimálnu krivku koordinácie pre rôzne úrovne zložitosti systému. Vyplýva z neho tiež fakt, že ak projekt čelí nesprávnemu načasovaniu alebo obmedzeným zdrojom, projektový manažéri zvyknú mať tendenciu preskočiť, či aspoň uponáhľať návrh systému a venovať viac času implementácií. Skúsenosti však ukazujú, že aj v takomto prípade je lepšie venovať viac času prvotnému návrhu a predísť tak častému a drahému prerábaniu systému v čase implementácie.



Obrázok 2 - Optimálna krivka koordinácie pre rôzne úrovne zložitosti systému

Riešenie konfliktov v tíme

Počas práce na projekte aj v dobrom tíme vznikajú konflikty medzi jednotlivými členmi. Úlohou manažéra je týmto konfliktom v čo najväčšej možnej miere predchádzať, prípadne ich podchytiť a riešiť už v zárodku. Je dôležité, aby si manažér zachoval nestrannosť.

Takisto nie je vhodné kritizovať zamestnanca za jeho chyby pred ostatnými členmi tímu. Môže to mať dokonca negatívny vplyv aj na ostatných členov tímu, pretože manažér tak v ich očiach stráca prirodzenú autoritu, najmä ak tým začnú spochybňovať jeho metódy práce a rozhodnutia, či začnú podvedome sympatizovať s kritizovanou osobou. Je oveľa vhodnejšie vydiskutovať si problém s členom tímu osamote medzi štyrmi očami. No aj tu by sa mal manažér zdržať čistej kritiky. Mal by vystupovať asertívnym spôsobom a ponechať priestor na vyjadrenie aj druhej strane. Mal by sa snažiť pochopiť dôvody, ktoré viedli k vzniku problému, motivovať zamestnanca k ich odstráneniu a navrhnúť spôsoby ako vzniku takýchto chýb predísť v budúcnosti.

Budovanie dobrých medziľudských vzťahov a zmena organizačnej štruktúry

To, že dôležitosť budovania a vylepšovania dobrých vzťahov nie len v rámci tímu, ale v celofiremnej miere už dnes pochopila väčšina firiem. Je bežné, že firma organizuje pre svojich zamestnancov rôzne športové a spoločenské aktivity, ktoré majú za úlohu práve navodiť priateľskú atmosféru a budovať medziľudské vzťahy v kolektíve. Takéto aktivity vyvíjane firmou sa označujú ako „team-building“ aktivity a okrem už spomínaného majú aj dobrý reprezentačný efekt pre firmu voči svojim zamestnancom. Ľudia totiž radi pracujú pre firmy, ak majú pocit, že nie sú firmou iba „využívaní“ a že okrem odmeny vo forme platu sa snaží pre nich vyvíjať aj ďalšie aktivity.

Posledným faktorom, ktorý vo svojej eseji rozoberiem je zmena organizačnej štruktúry tímu. Asi najkritickejšia je zmena tímového manažéra. Výskum ukázal, že takáto zmena sa deje zhruba v 16% softvérových projektov a väčšina členov tímu ju vníma ako výrazne negatívnu. Organizačné zmeny na ostatných pozíciách neohrozujú úspešnosť celého projektu, no manažment by si mal dať pozor pri zvýšenej fluktuácií pracovníkov [2].

Záver

Vybudovať a viesť tím, ktorý úspešne zvládne vyriešenie softvérového projektu nie je triviálna úloha a predstavuje výzvu pre dobrého manažéra. V tejto eseji som sa snažil poskytnúť prehľad niektorých faktorov, ktoré podľa mňa výrazne vplývajú na úspešnosť projektového manažéra a aj celého tímu.

Je potrebné venovať úsilie a dostatočnú pozornosť najmä vytváraniu tímu s prihliadnutím na individuálne vlastnosti potenciálnych členov a to nielen po odbornej, ale aj charakterovej stránke. Rovnako dôležité je určiť dobrú komunikačnú štruktúru tímu a neustále posilňovať dobré medziľudské vzťahy a tímového ducha. Dobrý manažér sa snaží konfliktom predchádzať, no ak vzniknú musí vedieť ako ich riešiť. Rovnako si treba dať pozor na dobré zvládnutie prípadných organizačných zmien v tíme.

Esej nemala za cieľ poukázať na všetky aspekty, ktoré vplývajú na úspešnosť tímu pri riešení softvérového projektu. Medzi niektoré oblasti, ktoré by si mal dobrý projektový manažér taktiež osvojiť patria napríklad sledovanie zviazanosti a súdržnosti tímu, či rôzne ekonomické a technologické aspekty tímových projektov.

Použitá literatúra

1. Bieliková, M.: *Softvérové inžinierstvo. Princípy a manažment.* Bratislava : Vytavateľstvo STU, 2000. 220 s. ISBN 80-227-1322-8
2. Verner, J. M., Evanco, W. M.: *In-House Software Development: What Project Management Practices Lead to Success?* In: IEEE Software, vol. 22, January/February 2005, no. 1, pp. 86-93
3. Chiang, R. I., Mookerjee, V. S.: *Improving Software Team Productivity.* In: Communications of the ACM, vol. 47, May 2004, no. 5, pp. 89-93.
4. Brooks, F. P.: *The Mythical Man-Month: Essays on Software Engineering*, Anniversary Edition, Addison-Weasley, Reading, PA 1995

Annotation

How to build and lead a successful team during the whole software project

Development of a team solving a software project and the stages of development the team is undertaking are the main topics of this essay. It describes particular problems of this development from the point of the view of the management. It also talks about various aspects of this development and about the effects of the decisions of the project management.

Author in the essay focuses mainly on identifying of the stages of team development, analyzing several aspects of team creation with observing individual technical and personal features of potential team members and also the number of team members as well. Author also discusses the need of creating good communication structure between the team members and the team-building aspects. At last, the essay focuses on the organizational changes in the team.

Odhadovanie v softvérových projektoch

Keď na veľkosti záleží

JOZEF BEŇO

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

Abstrakt. Odhadovanie softvérových projektov je oblasť, kde je softvérové inžinierstvo stále veľmi neisté. Pri veľkých projektoch je skôr výnimkou, ak sa odhad čo len približne podobá výslednému úsiliu. Pri menších projektoch je situácia lepšia. V práci uvádzam niektoré možnosti zlepšenia odhadovania pre projekty, ktorých členmi sú aj menej skúsení riešitelia. Ide o použitie kontrolného zoznamu, ktorý slúži na pripomenutie dôležitých častí produktu a aktivít projektu, aby neboli vynechané pri tvorbe odhadu rozsahu a potrebného úsilia. Ako druhú spomeniem techniku Deplhi, ktorej cieľom je uľahčiť zdieľanie znalostí medzi členmi tímu, a tak prispieť ku kvalitnejšiemu odhadu.

Úvod

Ako inak začať úvahu o odhadovaní v softvérových projektoch, než hrozivými číslami zo široko citovaných správ Standish Group. Od roku 1995, keď sa podľa ich výskumu v USA zrušilo pred dokončením až vyše 31% softvérových projektov, sa situácia síce zlepšila, ale aj podľa reportu z tretieho kvartálu roku 2004 ich stále zlyhalo nepríjemných 18%. V tabuľke (Tab. 1) sú uvedené štatistiky z niekoľkých rokov (zo zdrojov na stránkach Standish Group: www.standishgroup.com).

(v %)	1995	2000	2004
Úspešné	16	28	29
Problémové	53	49	53
Zrušené	31	23	18

Tab. 1. Vývoj úspešnosti softvérových projektov

Za úspešné projekty sa akceptujú len tie, ktoré boli dokončené načas, za plánovaný rozpočet a obsahujú všetky funkčnosti, ktoré boli požadované. Problémové sú tie projekty, ktoré boli dokončené a odovzdané do používania, ale nesplnili minimálne jedno kritérium úspechu, t.j. boli odovzdané neskôr a/alebo ich vývoj stál viac než bol rozpočet a/alebo neobsahovali všetky pôvodne požadované funkcie. Zrušené projekty sú tie, ktoré boli zastavené pred dokončením výsledku alebo boli vyradené z prevádzky krátko po nasadení.

V roku 2001 uskutočnila britská počítačová spoločnosť podobný výskum na vzorke viac ako 1000 IT projektov zo Spojeného kráľovstva. Zistili, že až 90% úspešných projektov malo trvanie do 12 mesiacov a 47% trvalo menej ako 6 mesiacov. Ide však o celkové trvanie a nie o rozsah. K podobnému záveru dospeli aj v Standish Group, keď ako optimálne odporúčajú projekt štyroch ľudí počas štyroch mesiacov, t.j. celkovo 16 človeko-mesiacov (ČM).

Aj tieto zistenia podnietili vznik a rozšírenie agilných metód vývoja softvéru. Jedným z ich nosných princípov je inkrementálny vývoj, keď sa funkčnosť do výsledku dopĺňa postupne po menších častiach. Každá jedna časť je vedená ako samostatný projekt, ktorý je lepšie manažovateľný a, ako ukázali štatistiky, aj spoľahlivejší.

Aj pri malých projektoch je však možné zlepšiť odhadovanie. Najviac problémov s odhadom veľkosti projektu a potrebným úsilím majú menej skúsení členovia tímu. Pre zlepšenie ich orientácie v projekte, ktorá vedie aj k zlepšeniu odhadov, sa používajú aktivity, z ktorých uvediem techniku Delphi a použitie kontrolného zoznamu.

Čo je odhad?

Magne Jörgensen v krátkom článku [1] upozorňuje na skutočnosť, že pojem odhadovania sa používa nejasne. Na príklade s plánovaním rozpočtu dovolenky identifikoval tri rôzne typy odhadov:

- Najpravdepodobnejší
- Minimalizujúci riziko
- Minimalizujúci náklady

Najpravdepodobnejší odhad vyjadruje predpokladanú mieru úsilia potrebného na vykonanie projektu. Môže sa získať pomocou rôznych techník alebo analógiou s predchádzajúcimi podobnými projektmi na základe skúseností.

V druhom prípade je najdôležitejšie vziať do úvahy všetky možné riziká spojené s projektom, aby aj v prípade, že došlo k ich uskutočneniu, nebol odhadnutý rozpočet prekročený.

Tretí typ je spojený s cieľom minimalizovať náklady tým, že sa obmedzí rozpočet a účastníci projektu ho musia pri riešení jednotlivých častí brať do úvahy, t.j. musia si „strážiť“ náklady.

Nejednoznačnosť v odhadovaní môže spôsobovať problémy. Často sa stáva, že odhad, ktorý je vyhotovený ako najpravdepodobnejší, chápe zákazník ako odhad, ktorý už má započítané všetky riziká, a z toho usudzuje, že celkové náklady neprekročia poskytnutý odhad.

Problémy s nejednoznačnosťou sa však týkajú aj porovnávania projektov. Ak sa vyhodnocujú odhady vytvorené s rôznymi cieľmi (najpravdepodobnejší odhad, bezpečný odhad), tak výsledky sú nutne nepresné a neposkytujú relevantné údaje pre ďalší výskum a nevedú ani k spresneniu nových odhadov. Problém sa, podľa Magneho, týka aj najznámejšieho výskumu, vyššie spomenutého reportu Chaos od Standish Group, keď vo vstupných formulároch nie je uvedené, aký typ odhadu majú prispievatelia zadávať. Možno aj preto bolo v ich správe z roku 1995 priemerné prekročenie počiatočného odhadu až 89%.

Odhadovanie pre veľké projekty

Pre veľké systémy boli vypracované rôzne metodiky, ktoré sa odvádzali na základe skúseností z predchádzajúcich projektov. Pomocou určenia veľkosti systému na základe informácií, ktoré sú k dispozícii po fáze špecifikácie a návrhu, sa snažia ďalej odhadnúť koľko úsilia, vyjadreného v človeko-mesiacoch, je potrebné na realizáciu daného projektu.

Tieto metodiky majú definovaný vzťah medzi úsilím a veľkosťou projektu v tvare

$$\acute{U} := a + bV^c \tag{1}$$

Kde Ú je veľkosť úsilia a V je veľkosť systému, čo môže byť údaj v počte riadkov kódu (LOC) alebo veľkosť môže byť určená pomocou funkčných bodov (FP).

Hodnoty premenných a, b a c je pre každú metodiku rozdielna. V tabuľke sú uvedené niektoré metódy a hodnoty ich parametrov (Tab. 2).

Model	Faktor b	Faktor c	Veľkosť
Walston-Felix	5.2	0.91	LOC
COCOMO (organic)	2.4	1.05	LOC
Herd		1.06	LOC
COCOMO (semi-detached)	3.0	1.12	LOC
Bailey-Basili	5.5	1.16	LOC
Frederic		1.18	LOC
COCOMO (embedded)	3.6	1.20	LOC
Jones		1.40	LOC
Halstead	0.7	1.50	LOC
Schneider		1.83	LOC
COCOMO II	2.9	1.10	LOC
Albrecht and Gaffney	-13.39	1	FP

Kemerer	60.62	3	FP

Tab. 2. Porovnanie jednotlivých modelov odhadovania [6]

Rozdiely, ktoré sú medzi jednotlivými metódami, vyplývajú z množiny dát, z ktorých sa štatistickými metódami získavali hodnoty parametrov a, b a c. Tento vzťah je ovplyvňovaný množstvom faktorom a je takmer pre každú firmu alebo dokonca projekt osobitý. Určenie veľkosti projektu je zložité a citlivé na nastavenia. Napríklad COCOMO má až 15 „driverov“, ktorými sa opisuje vývojové prostredie. V metóde Use Case Points [8] sa používa 13 faktorov opisujúcich technickú zložitosť. Je zrejmé, že chybným určením niektorého parametra sa výrazne zmení odhad veľkosti výsledku a teda aj odhad potrebného úsilia. Podľa niektorých výskumov boli pri používaní COCOMO evidované priemerné odchýlky odhadov od uskutočneného úsilia až v rádoch stoviek percent.

Malé projekty

Kathleen Peters z firmy „Software Productivity Center” uvádza v [5], že za malý projekt sa považuje ten, ktorý vykonáva tým veľkosti jednej až dvoch osôb a je naplánovaný na dobu do šesť mesiacov. Malý projekt je tak najviac v rozsahu 12 človeko-mesiacov.

Literatúry, ktorá by sa venovala projektom uvedeného rozsahu, nie je veľa. Čiastočne to môže vyplývať aj z toho, že malé projekty majú lepšie výsledky a je pre ne ťažké uskutočniť dostatočne presné merania. Podrobné zdôvodňovanie odhadov a meranie činností v projekte by malo vplyv na celkové vynaložené úsilie, a teda by nastal, fyzikom dobre známy, efekt, keď meranie ovplyvňuje namerané hodnoty.

V univerzitnom prostredí je možné venovať sa aj malým projektom. Ak projekt trvá jeden školský rok a zúčastní sa ho šesť študentov, je celková veľkosť projektu približne 200 človeko-dní, t.j. je už dostatočne veľký, aby malo zmysel odhadovať jeho veľkosť, resp. jeho náklady. Projektom v tomto rozsahu sa venovali práce [2], [3] a [4], z ktorých budem čerpať niektoré postrehy.

Odhadovanie pre malé projekty

Pre malé týmy je nepraktické strácať čas komplikovanými definíciami rôznych premenných a následnými výpočtami údajov, ku ktorým nemajú dôvod, v konečnom dôsledku, mať dôveru.

Ak by chceli na odhad prácnosti použiť napr. metódu „Use Case Points“, ktorú navrhol Gustal Karner, museli by definovať množstvo informácii [8]:

- počet a zložitosť prípadov použitia
- počet a zložitosť hráčov so systémom
- rôzne nefunkcionálne požiadavky (*prenosnosť*, *udržiavateľnosť* atď.)
- prostredie, v ktorom sa projekt vyvíja (*použitý jazyk*, *motivácia tímu* atď.)

Pre menšie projekty by bolo priradenie váhy každému parametru vstupujúcemu do výpočtov veľmi náročné a, v končenom dôsledku, by výsledná hodnota nemusela byť ani približne správna. Chybné určenie váhy pre niektorý parameter (napr. *jednoduchosť používania* alebo *zložitosť výpočtov*) môže výrazne zmeniť odhad veľkosti, a teda aj potrebného úsilia.

Najvhodnejším spôsobom ostáva ten najpriamočiarejší, a to nechať odhad na členov tímu, aby každý zvážil koľko úsilia bude potrebovať na realizáciu projektu. Pomocou metód pre veľké projekty nie je možné nasimulovať vlastnosti malého projektu a členov riešiteľského tímu. Expertný odhad je pre malé projekty aj najviac odporúčaným spôsobom v literatúre.

Problémy môžu mať menej skúsení vývojári. Najmä pre nich je vhodné použitie niektorých jednoduchých techník, z ktorých sa budem krátko venovať použitiu kontrolných zoznamov a technike Delphi.

Kontrolný zoznam

Kontrolné zoznamy pomáhajú členom tímu nezabúdať na niektorú oblasť riešenia. Zvyčajne sú odvodené z Breakdown Structure a obsahujú najpodstatnejšie aspekty projektu. Zoznam môže byť definovaný produktovo alebo procesne, t.j. môže obsahovať to, čo sa má dodať (*triedy*, *obrazovky*, *dokumentácia*) ale aj aké aktivity sú spojené s vývojom (napr. *tréning používateľov*, *testovanie*).

Najmä pre neskúsených vývojárov je kontrolný zoznam užitočný pri odhadovaní potrebného úsilia, keďže je pre nich charakteristické, že sa sústredia len fázu tvorby kódu (konštrukčná fáza v Unified Process) a na ostatné si nerezervujú čas.

Podľa výsledkov výskumu na univerzite Bournemouth [3] študentov najprv nechali odhadnúť veľkosť a potrebné úsilie. V druhom kole odhadovania im poskytli kontrolné zoznamy pre veľkosť a aj pre proces. Pri procesom zozname použili aktivity definované v Rational Unified Proces.

Výsledok porovnania odhadov v prvom a druhom kole potvrdil predpoklad, že neskúsení, resp. menej skúsení riešitelia zabudnú na niektorú časť produktu alebo procesu. Väčšina študentov zväčšila svoj odhad veľkosti a takmer všetci zvýšili odhad potrebného úsilia, čo svedčí o tom, že v prvom kole venovali málo pozornosti iným aktivitám ako samotné programovanie.

Skúsenejší členovia tímu sú potrební na to, aby zadefinovali obsah kontrolného zoznamu, resp. na to, aby schválili použitie existujúceho zoznamu z podobného projektu.

Metóda Deplhi

Metóda Deplhi je založená na tímovej diskusii, kde si jednotliví členovia tímu vymieňajú názory týkajúce sa projektu. Význam majú najmä poukazovania na možné riziká, kde sa spoločnou diskusiou môže zlepšiť odhad jeho miery a času potrebného na jeho riešenie.

Metóda Delphi je iteratívna metóda a je zameraná na eliminovanie skupinového efektu tým, že odhady sú vykonávané anonymne. Zistilo sa, že je to vhodná metóda na minimalizovanie podľahnutiu väčšinovému názoru. Debata členov je otvorená a nie je v nej vyžadovaný všeobecný súhlas diskutujúcich.

V prípade výskumu [3] nie je jednoznačne možné určiť vplyv metódy Deplhi na odhadovanú veľkosť resp. potrebné úsilie. V niektorých prípadoch novo získané vedomosti umožnili znížiť odhad, keďže niektoré predtým problematické otázky sa v diskusii ukázali byť jednoduchšie na riešenie. Na druhej strane sa niekedy ukázalo, že študenti zabudli na niektoré aktivity alebo časti produktu, a preto výsledkom diskusie bolo zvýšenie odhadov.

Metóda Delphi však mala pozitívny efekt v tom, že riešitelia po diskusii mali bližšie odhady celkového úsilia potrebného na riešenie projektu. Tiež sa zvýšila aj dôvera k vytvorenému odhadu.

Záver

Pre odhadovanie malých projektov je najčastejšou a najkvalitnejšou metódou expertný odhad. Malým tímom, kde ani jeden člen nemá dosť skúseností s odhadovaním, nepomôžu žiadne metodiky, keďže na ich zvládnutie je potrebné mať veľa znalostí a ich výstupy sú veľmi citlivé na vstupné parametre, a teda prakticky nepresné.

V prípade, že sa použije kontrolný zoznam a metóda Deplhi, dôjde k zlepšeniu odhadu, keďže dochádza k zdieľaniu poznatkov. K presnosti odhadov však prispieva aj Parkinsov princíp, podľa ktorého sa práca vždy rozširuje až dovtedy, kým sa nenaplní čas, ktorý je pre ňu vyhradený. Takže, ak expert pri odhadoch aplikuje bežné riadenie rizika tak, že pôvodný odhad vynásobí osvedčenou konštantou, je veľká pravdepodobnosť, že projekt bude ukončený včas a za odhadnuté náklady.

Uvedený spôsob odhadovanie projektov je na Slovensku dominantný a používa sa pri všetkých projektoch, od malých až po tie najväčšie. Súvisí to s bežnou praxou vo firmách, ktoré podceňujú význam úvodných fáz projektu a snažia sa čím skôr dostať k implementácii. Bez dôkladne zvládnutej špecifikácie požiadaviek nie je možné kvalitne odhadovať, keďže ako vraví múdrosť štatistikov: Zo zlých údajov sa zlou metódou môžu získať dobré výsledky, ale ak máme dokonalú metódu, tak zlé vstupy nutne vedú k zlým výstupom.

Použitá literatúra

1. Jörgensen, M.: How Much Does a Vacation Cost? Or What is a Software Cost Estimate. *ACM Software Engineering Notes,* Vol. 28, Issue 6 (Nov. 2003) 30.
2. Johanson, L., Lungdren, M: *Software Project Planning Light: Applying project planning techniques on small software projects.* Växjö, 1999.
3. Passing, U., Shepperd, M.: An Experiment on Software Project Size and Effort Estimation. International Symposium on Empirical Software Engineering 2003 (ISESE'03).
4. Jörgensen, M, Sjöberg, D.I.: Impact of Effort Estimates on Software Project Work.
5. Peters, K.: *Software Project Estimation.* Kathleen Peters, Software Productivity Center Inc., 1999 (www.spc.ca)
6. Johnson, K.: *Software Cost Estimation: Metrics and Models.* Department of Computer Science, University of Calgary.
7. Shepperd, M., Schofield, C.: Estimating Software Project Effort Using Analogies. *IEEE Transactions On Software Engineering*, Vol. 23, No. 12 (Nov. 1997) 736 –743.
8. Cohn, M.: Estimating With Use Case Points. *Methods & Tools*, Vol. 13, No. 3 (2005) 3-13

Annotation

Estimating in Software Projects

For estimating software projects great differences between small ones and huge ones exist. From several studies it is known that huge projects are more likely to fail than small ones. But there are still opportunities for improving estimates. Effort and size estimate accuracy depends on team members' capabilities and therefore less experienced people tend to underestimate their estimate. In this essay I am briefly introducing two techniques, check list and Delphi, which could help them improve their estimates.

Manažment rizík v softvérovom projekte

JOZEF KRIŠKA

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
jozef.kriska@pobox.sk

Abstrakt. Pri plánovaní softvérových projektov ako veční optimisti často predpokladáme, že všetko bude prebiehať podľa navrhnutého plánu. Opačný pohľad ponúka otázku: Prečo vytvárať podrobné plány, keď aj tak nemôžeme predpovedať všetky udalosti? Uvedené prístupy mnohokrát vedú ku vzniku neočakávaných udalostí. Dôsledky týchto udalostí sú väčšinou nežiadúce a prispievajú k zlyhaniu softvérových projektov. Prieskumy však poukazujú na to, že veľa neočakávaných udalostí je predvídateľných, a preto sa manažment rizík stáva neoddeliteľnou súčasťou úloh manažmentu. Práca opisuje základné princípy manažmentu rizík, tiež sa snaží poskytnúť odpovede na dve základné otázky: Aké faktory vnímajú softvéroví projektoví manažéri ako riziká, a ktoré považujú za najdôležitejšie? Ako môžeme kategorizovať riziká?

Úvod

„Čo sa môže pokaziť, to sa aj pokazí“, tak znie jeden z Murphyho zákonov. A teda, podobne ako v mnohých iných oblastiach, tak aj v oblasti tvorby softvéru, sa objavujú zlyhania projektov. Môžeme konštatovať, že softvér je jednou z najproblematickejších technológií tejto doby. Veľa projektov má vysokú pravdepodobnosť, že budú predčasne ukončené alebo pozastavené, čo však nie je len „dôsledkom“ spomínaného Murphyho zákona, ale hlavne dôsledkom zanedbania analýzy a riadenia rizík manažmentom softvérových projektov. Príznaky toho, že organizácia vyvíjajúca softvérový produkt zanedbáva manažment rizík, sú zväčša nasledovné: projekt je neustále v stave nestability; opakujú sa sklzy v navrhnutom pláne činnosti, ktoré sú dôsledkom opakujúcich sa zhmotnení daných rizík; pracovníci pracujú v ustavičnom strese a pod.

V publikácii [3] je uvedené, že zo všetkých projektov jedna tretina bola predčasne ukončená a viac ako polovica nespĺňala všetky požiadavky. Približne len jedna šestina projektov bola ukončená načas a neprekročila limit rozpočtu. Z týchto údajov je alarmujúce, že len veľmi málo manažérov softvérových projektov sa zaoberá

manažmentom rizík v softvérovom projekte a preto je potrebné tejto problematike venovať väčšiu pozornosť.

Riziká a manažment rizík

Riziko je definované ako *možnosť utrpieť poškodenie, stratu alebo zničenie* [4]. Je charakterizované dvom intuitívnymi vlastnosťami: *neurčitosť* a *strata*. Zahŕňa možnosti vzniku problému, ktorého dôsledky môžu mať dopad na cenu, plánovanie alebo celkový úspech projektu. Je zrejmé, že čím je projekt komplexnejší, tým zachytáva väčšiu škálu rizík. Uvedená závislosť je znázornená na obrázku 1.

Obr. 1 Vzťah medzi zložitosťou systému a rizikami.

Manažment rizík zahŕňa *techniky a návody pre proces identifikovania možných rizík projektu*, *ich analýzy* a *eliminovania.* Kladie dôraz na splnenie nasledovných troch cieľov:

- Prevencia rizík
- Zmiernenie rizík
- Zaistenie potrebných činností pri zlyhaní

Pre dosiahnutie spomínaných troch cieľov sa musí manažment rizík opierať o základné princípy [2], ktorých formulácia je nasledovná:

- Spoločná vízia výsledného produktu
  - Spoločná vízia produktu založená na spoločnom cieli, spoločné vlastníctvo, kolektívna povinnosť
  - Sústredenie sa na výsledok
- Tímová práca
  - Spoločné úsilie pre dosiahnutie spoločného cieľa
  - Zdieľanie zručností a skúseností v rámci tímu
- Pohľad dopredu

    - „Myslenie na zajtrajšok“, identifikovanie rizík, očakávanie možných dopadov identifikovaných rizík
- Otvorená komunikácia
    - Tok informácii medzi všetkými úrovňami v rámci projektu
    - Formálna a neformálna komunikácia v rámci tímu
- Ucelený manažment
    - Vnímanie manažmentu rizík ako neoddeliteľnú súčasť celého manažmentu projektu
- Plynulý proces
    - Udržiavanie stálej ostražitosti voči rizikám

Jedným z cieľov manažmentu rizík je predchádzať nepredvídaným udalostiam protiopatreniami, ktoré obmedzia riziká v projekte, alebo znížia ich dopad. Manažment rizík pracuje s možnosťami výskytu takých udalostí, ktoré nie sú „štandardné“ alebo všeobecne očakávané. Keďže medzi riziká zahŕňame aj tie riziká, ktoré neboli identifikované na začiatku, je potrebné zdôrazniť, že *proces manažmentu rizík sa musí vyvíjať*, prispôsobovať na nové a zmenené poznatky v procese vývoja projektu.

Manažment rizík sa týka predovšetkým procesu, ale už menej konečného produktu. Hlavnou úlohou je preto zaistenie integrity procesu vývoja softvéru. Ak tento proces prebieha dobre, existujú len minimálne nepredpokladané nežiadúce dopady na riadenie vývoja projektu a teda aj na výsledný produkt. Všetky identifikované rizikové faktory by preto mali byť pod efektívnou kontrolou manažmentu.

Manažment rizík nemôže byť vnímaný len z metodologického pohľadu (techniky a návody pre proces identifikácie, analýzy...), ale je potrebné do neho zahrnúť aj ďalšie aspekty týkajúce sa jeho komplexnosti, ktoré rozširujú manažment rizík o dva ďalšie rozmery: *časový rozmer* a *ľudský rozmer*. Pohľad na časový rozmer nám poskytuje dve vízie:

- Dlhodobá vízia reprezentujúca globálnu perspektívu
- Krátkodobá vízia predstavujúca pohľad projektových manažérov

Ľudský rozmer reprezentuje viac menej intelektuálny rozmer – najviac kritický, keďže vývoj softvérového produktu vyžaduje intelektuálne aktivity, hlavne v procese vývoja. Pri ľudskom rozmere zväčša identifikujeme štyri základné aspekty:

- Jednotlivec
- Tím
- Manažment
- Osoby zahŕňajúce zákazníka a klienta

Prečo formálny prístup v manažmente rizík ?

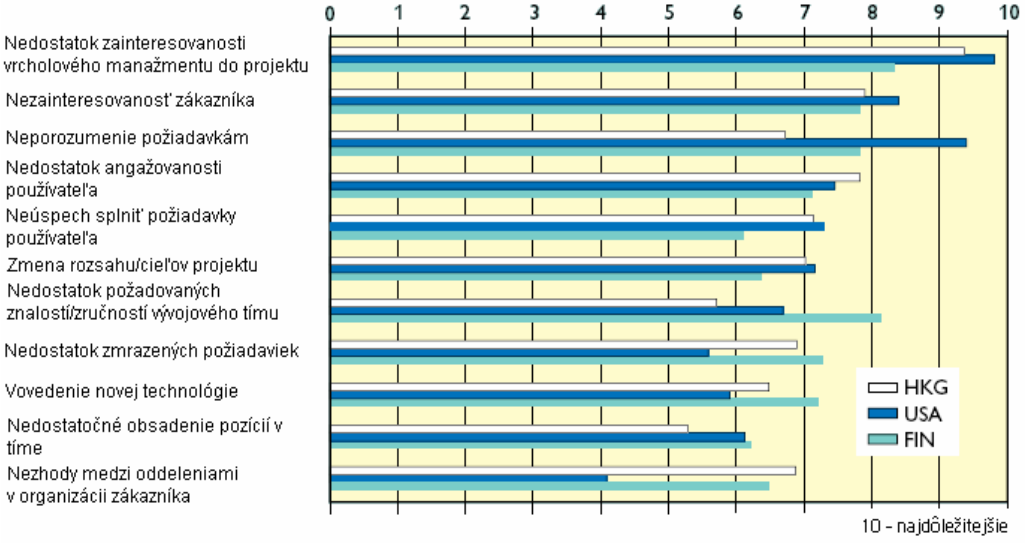
Formálny prístup v manažmente rizík poskytuje projektovému tímu mnohé výhody. Umožňuje použitie štruktúrovaného mechanizmu pre identifikáciu a analýzu rizík, ktoré

ovplyvňujú úspešnosť projektu. Zvážením potenciálnych dopadov každého rizika môžeme poukázať na tie, ktoré majú najvyššiu prioritu a teda budú medzi prvými zaradené pod kontrolu manažmentu. Tímový prístup umožňuje viacerým zúčastneným projektu poukázať na rôzne spoločné riziká a tak priradiť zodpovednosť na znižovaní daných rizík najvhodnejším osobám v rámci tímu. Bez formálneho prístupu nemôžeme zaistiť včasné spustenie protiopatrení definovaných manažmentom rizík, ich ukončenie podľa navrhovaného plánu a taktiež nemôžeme zaistiť ich efektívnosť.

Hlavné príčiny zlyhania softvérových projektov

Zoznam rizík, ktoré sa môžu vyskytnúť v oblasti tvorby softvérového produktu je veľmi rozsiahly. Riziká sú zväčša identifikované ako výsledky analýz možných príčin zlyhania softvérových projektov, alebo sú získané z predchádzajúcich projektov, ktorých ukončenie bolo zapríčinené zhmotnením niektorého z možných rizík.

Zodpovedanie otázky: Ktoré faktory vnímajú softvéroví projektoví manažéri ako riziká pri tvorbe softvérového produktu a ktoré považujú za najdôležitejšie? je možné vyčítať z grafu na obrázku 2. V grafe sú zachytené skúsenosti softvérových projektových manažérov z troch oblastí sveta: Hon Kong, USA a Fínsko. Projektovým manažérom bola položená najprv otázka ohľadom identifikovania rizík, s ktorými sa počas svojej praxe stretli. Druhá otázka bola smerovaná na zachytenie dôležitosti jednotlivých identifikovaných rizík.



Obr. 2 Riziká a ich dôležitosť podľa projektových manažérov z troch rôznych oblastí sveta. [7]

Zo štúdie, ktorej výsledky sú znázornené na obrázku 2 vyplývajú zaujímavé výsledky. V zmysle identifikovania a určenia dôležitosti jednotlivých rizík je vidieť, že mnohé riziká sú spoločne označované za dôležité z pohľadu projektových manažérov zo všetkých troch oblastí. Toto pozorovanie poukazuje na existenciu univerzálnych množín rizík v globálnom meradle, ktoré nám umožňujú vytvárať kategórie jednotlivých rizík na základe ich predpokladanej závažnosti. Možné prístupy kategorizácie rizík sú načrtnuté v kapitole Kategorizácia rizík.

Zaujímavé je, že len jedno riziko zo všetkých uvedených rizík označovaných za dôležité je ovplyvnené technológiou. Je to riziko zavedenia novej technológie do procesu tvorby softvéru. Tomuto riziku neprikladajú projektoví manažéri veľkú dôležitosť v porovnaní s ostatnými rizikami. Jedným z dôvodov tohto pozorovania môže byť pocit manažérov, že môžu kontrolovať riziko spôsobené zavedením novej technológie do vývojového procesu, čo vyplýva aj z poznámky jedného projektového manažéra: „Očakávam, že odhady rizík týkajúcich sa technológie sú známe, a že sú zachytené v projektovom pláne.“ [7].

Snahou nasledovných podkapitol je priblížiť niekoľko rizík, ktoré patria medzi najkritickejšie.

Nedostatok zainteresovanosti vrcholového manažmentu

Nedostatočná zainteresovanosť vrcholového manažmentu do projektu patrí medzi najkritickejšie riziká softvérových projektov (viď. obrázok 2). Je zrejmé, že aktívna účasť vrcholového manažmentu hrá jednu z kľúčových rolí vo všetkých etapách projektu - od jeho inicializácie až po etapu implementácie.

Nezainteresovanosť zákazníka

Riziko predstavujúce nezahrnutie záväzku zákazníkov hlavne do procesu špecifikácie požiadaviek. Napr.: Používatelia vyvíjaného softvéru sú primárni zákazníci... potom, ak nie sú používatelia viazaní k spoločnému úsiliu – k spolupráci, napr. v procese špecifikácie požiadaviek, môže vzniknúť riziko toho, že používateľ sa nebude zapájať do tohto procesu, a teda môže dôjsť k mylnému predpokladaniu jeho funkčných požiadaviek na výsledný softvérový produkt.

Neporozumenie požiadavkám

Neporozumenie požiadavkám patrí tiež k hlavným rizikám procesu vývoja softvérového produktu. Môžeme konštatovať, že požiadavky vlastne riadia celý projekt. Proces špecifikácie a analýzy požiadaviek môže byť veľmi zložitý a časovo náročný. Avšak bez podrobnej špecifikácie a analýzy požiadaviek vzniká možnosť vyvíjania softvérového systému, ktorý nikto nechcel, preto je potrebné klásť dôraz na vytvorenie podrobnej množiny požiadaviek.

Nedostatočná angažovanosť používateľov

Ďalšie veľmi kritické riziko. Toto riziko môže vzniknúť ako dôsledok zhmotnenia rizika nedostatočnej väzby na zákazníka. A teda, ak nie sú používatelia dostatočne zahrnutí do procesu vývoja softvéru, najmä do procesu špecifikácie, viac menej je isté, že sa nebudú vývojovým tímom identifikované požiadavky prekrývať s požiadavkami používateľa.

Neúspech splniť požiadavky používateľa

Riziko nesplnenia požiadaviek používateľa je tiež pomerne dôležité. Vo veľkej miere prispievajú k úspechu či neúspechu projektu očakávania používateľov, ktoré sú odrazom ich požiadaviek. V mnohých prípadoch je preto nutné opakovane vyhodnocovať splnenie požiadaviek v porovnaní so skutočnou funkčnosťou systému.

Nedostatok požadovaných znalostí vývojového tímu

V prípade nadhodnotenia zručností a znalostí jednotlivých členov tímu môže vznikať riziko nezvládnutia technológií použitých pri vývoji softvérového produktu. Takíto pracovníci sa potom dopúšťajú kritických a rozsiahlych chýb, ktoré znižujú celkovú produktivitu. Je potrebné poukázať nielen na riziko nezvládnutia technológií členmi vývojového tímu, ale aj na riziko nedostatočných znalostí projektových manažérov. Pre veľké projekty sa očakávajú zručnosti v plánovaní, organizovaní a komunikácii, ktoré pracovník so zručnosťami v oblasti technológií nemusí mať. Zručnosti vývojových pracovníkov a projektových manažérov preto zväčša tvoria dizjunktné množiny.

Kategorizácia rizík

Jedným z najdôležitejších procesov manažmentu rizík je ich analýza. Základným a najpoužívanejším prístupom v procese analýzy rizík je určenie pravdepodobnosti a možného dopadu pre každé z identifikovaných rizík. Použitím matice pravdepodobnosti a možného dopadu rizika kategorizujeme riziká od tých kritických až po tie, ktoré sú pre daný projekt okrajové. Je zrejmé, že kritické riziká sú charakterizované vysokou pravdepodobnosťou a vysokým dopadom, zatiaľ čo okrajové riziká budú predstavovať riziká s malou pravdepodobnosťou a malým dopadom. Príklad takejto kategorizácie rizík je znázornený tabuľkou 1. Riziká sú klasifikované do troch tried: kritické, stredne kritické a okrajové.

		Pravdepodobnosť		
		Vysoká	Stredná	Nízka
Dopad	Vysoký	Kritické	Kritické	Stredne kritické
	Stredný	Kritické	Stredne kritické	Okrajové

	Nízky	Stredne kritické	Okrajové	Okrajové

Tab. 3 Príklad kategorizácie rizík prostredníctvom matice pravdepodobnosti a dopadu.

Predchádzajúci prístup nám poskytuje spôsob, ktorým môžeme kategorizovať riziká na základe ich pravdepodobnosti a predpokladaného dopadu. Do tejto kategorizácie však nie je zahrnutý poznatok, že mnohí projektoví manažéri poukazujú na dôležité riziká nielen z hľadiska ich pravdepodobnosti a možného dopadu, ale hlavne z faktu, že tieto riziká nie sú pod ich priamou kontrolou. Mnohé riziká sú preto označované z pohľadu projektových manažérov za dôležitejšie ako tie ostatné z toho dôvodu, že majú nad nimi slabú alebo žiadnu kontrolu.

Poňatie dôležitosti kontroly nad danými rizikami v spojení s ich dôležitosťou umožňuje vytvorenie prístupu pre iný spôsob kategorizácie rizík. Riziká môžu byť mapované do jednej zo štyroch kategórií, ako to znázorňuje tabuľka 2. Jeden aspekt pôsobiaci na zaradenie do danej kategórie predstavuje dôležitosť rizika, druhý predpokladanú úroveň jeho kontroly projektovým manažérom. Pre definovanie dôležitosti rizika je vhodné použiť predchádzajúci prístup (tabuľka pravdepodobnosti a dopadov), ktorý nám umožňuje určiť kritické a stredne kritické riziká. Úrovne kontroly predstavujú možnosti projektových manažérov predchádzať svojimi činnosťami zhmotneniu jednotlivých rizík alebo minimalizovať ich dopady.

		Úroveň kontroly	
		Nízka	Vysoká
Dôležitosť rizika	Vysoká (kritické riziká)	**I** Väzba na zákazníka	**II** Rozsah a požiadavky
	Stredná (stredne kritické riziká)	**IV** Prostredie	**III** Realizácia

Tab. 4 Kategorizácia rizík zohľadňujúca úroveň kontroly daných rizík. [7]

Do kategórie I patrí pomerne veľa rizík. Riziká v tomto kvadrante sú charakteristické väzbou na zákazníka, resp. používateľa. Možnosti kontroly týchto rizík bývajú zväčša pomerne nízke. Konkrétne do tohto kvadrantu zahŕňame riziká: nezainteresovanosť zákazníka, nedostatok angažovanosti používateľa a taktiež nedostatočné zainteresovanie vrcholového manažmentu do projektu. Pre potlačenie rizík z tejto kategórie je potrebné zabezpečiť nielen dobré vzťahy so zákazníkom, ale aj väzbu zákazníka na daný projekt.

Riziká spojené s nedostatočným odhadom rozsahu projektu ako aj nepresným špecifikovaním požiadaviek na vyvíjaný systém patria do kategórie II. Do kategórie II zaraďujem zväčša riziko neporozumenia požiadaviek používateľa a nedostatok

zmrazených požiadaviek – veľa požiadaviek, ktoré sa menia počas vývoja. Ako bolo spomenuté v predchádzajúcej kapitole, požiadavky riadia celý projekt, a preto je potrebné venovať väčšiu pozornosť špecifikovaniu a zmenám požiadaviek. Pre špecifikovanie rozsahu projektu je vhodné najprv určiť tie prvky, ktoré projekt nemá nezahŕňať.

Kategória III obsahuje riziká, ktoré majú vplyv na realizáciu projektu. Tieto riziká sú zväčša spojené so zlyhaniami manažmentu projektu: nedostatočné obsadenie pozícií v tíme, nevhodná metodológia v procese vývoja softvéru. Medzi opatrenia pre prevenciu rizík zachytených kategóriu III patria: presné definovanie rolí a zodpovedností v tíme, použitie disciplinovaného procesu vývoja softvéru; zahrnutie metodológií, ktorých použitím môžeme rozložiť projekt na menšie, a tak lepšie kontrolované časti.

Riziká prostredia, v ktorom sa projekt vyvíja, ale aj prostredia mimo organizáciu vyvíjaného projektu patria do kategórie IV. Predstavujú udalosti, ktoré môžu nastať vo vnútri organizácie alebo pôsobením vonkajších vplyvov. Patria sem riziká: zmena rozsahu/cieľov projektu a nezhody medzi oddeleniami v organizácii zákazníka. Zhmotnenie rizika kategórie IV je zväčša málo pravdepodobné, ale keď nastane, jeho dôsledky pôsobia na projekt veľmi zničujúco, preto je vhodné vytvárať postupy pre činnosti v takýchto situáciách.

Záver

Manažment rizík je považovaný za jednu z dôležitých oblastí v riadení softvérových projektov, keďže existuje mnoho spôsobov ich zlyhania. Táto práca poskytuje vniknutie do tejto problematiky. Prezentuje najčastejšie zlyhania softvérových projektov, ako aj dva prístupy kategorizácie rizík. Poukazuje na možnosť prepojenia spomínaných dvoch prístupov kategorizácie. Prvý umožňuje extrahovať kritické a stredne kritické riziká na základe ich pravdepodobnosti a dopadov. Takto klasifikované riziká môžu byť následne využité v druhom prístupe, ktorý do kategorizácie zahŕňa aj možnosť ich kontroly projektovým manažérom.

Použitá literatúra

1. Boehm, B. W.: Software Risk Management: Principles and Practices, 1991.
2. Higuera, R. P., Haimes, Y. Y.: Software Risk Management, 1996.
3. May, L. J.: Major causes of software project failures. *The Journal of Defense Softrware Engineering (1998)*.
4. Bieliková, M.: Manažment v softvérovom inžinierstve, 1999.

5. Webster, K. P. B., Oliveira, K. M., Anquentil, N.: A Risk Taxonomy Proposal for Software Maintenance. *Proceedings of the 21st IEEE International Conference on Software Maintenance (2005) 1063 – 1072.*
6. Addison, T., Vallabh, S.: Controlling software project risks – an empirical study of methods used by experienced project managers. *Proceedings of SAICSIT*, (2002) 128-140.
7. Keil, M., Cule, P., Lyytinen, K., Schmidt, R.: A framework for identifying software project risks. *Commun (Nov 1998) 76 – 83.*

Annotation

Software project risk management

In the planning process of software projects as eternal optimists, we often assume that everything will go exactly as planned. In opposite view: What's the point of making detailed plans, when we can not accurately predict what's going to happen ? These approaches are directed to origin of unpredictable events. Impacts of these events damage software projects. Researches reveal that many of that problems encountered were in fact predictable, therefore risk management presents important process of project management. This paper focuses on the basic principles of risk management, and address two basic questions: What are the factors that software project managers perceive as risks and which of these factors do they consider most important ? How can be risk factors categorized ?

Vplyv koordinácie procesu vývoja softvéru na produktivitu softvérového tímu

JOZEF WAGNER

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
wagner00@student.fiit.stuba.sk

Abstrakt. Už od počiatku bol softvérový priemysel v kríze. Softvér bol a stále je nespoľahlivý, dodávaný po stanovených termínoch, pružne nereagujúci na meniace sa potreby, neefektívny a drahý. Dokonca aj dnes, problémy so softvérovými systémami sú bežné a často publikované. Hlavným dôvodom je, že koordinácia aktivít sa stáva komplikovanou s rozsahom a narastajúcou zložitosťou projektu a preto sa problémy pochopiteľne častejšie vyskytujú pri vývojoch rozsiahlych softvérových systémov.

Preskúmame úlohu formálnej a neformálnej komunikácie pri koordinovaní práce na softvérových projektoch. Väčšina súčasných podporných nástrojov pre koordináciu používa procedúry formálnej komunikácie, ale ukazuje sa potreba zapojenia aj procedúr neformálnej komunikácie.

Existuje niekoľko prístupov koordinácie problémov pri vývoji softvérových systémov. Opíšeme tri súčasné prístupy, ktoré sa používajú: veľký tresk, častá integrácia a periodická synchronizácia, a chybami poháňaná koordinácia. Prístupy sa líšia v rozdielnom načasovaní a intenzite koordinácie. Taktiež sa budeme venovať faktorom, ktoré ovplyvňujú intenzitu koordinácie.

Záver eseje patrí motivácií ako ďalšiemu prostriedku na zvýšenie produktivity.

Úvod

Ako je známe, už od počiatku bol softvérový priemysel v kríze. Softvér je nespoľahlivý, dodávaný po stanovených termínoch, pružne nereagujúci na meniace sa potreby, neefektívny a drahý, a bol počas predchádzajúcich tridsiatich rokov. Dokonca aj dnes, problémy so softvérovými systémami sú bežné a často sú publikované v prestížnych článkoch na internete.

Problémy sa pochopiteľne častejšie vyskytujú pri vývojoch rozsiahlych softvérových systémov, pretože koordinácia aktivít sa stáva komplikovanou s rozsahom a narastajúcou zložitosťou projektu. Ich následky sa prejavujú vo forme meškania projektov, prekročenia nákladov a nespokojnosti zákazníkov.

V jednej ankete, ktorá ma zaujala, sa mali projektoví manažéri meškajúcich projektov vyjadriť, čo podľa ich názoru bola hlavná príčina, že nedodržali časový plán celého projektu. Až 50 % opýtaných respondentov uviedlo, že produktivita ich softvérového tímu nebola taká vysoká ako sa pôvodne očakávalo. Bolo to však naozaj spôsobené zníženou produktivitou softvérového tímu [1]?

Koordinácia softvérového tímu

Koordinácia [2] bola zadefinovaná ako usmernenie vynaloženého úsilia jednotlivcov k dosiahnutiu spoločných, explicitne predložených cieľov a integrácia alebo vzájomné poprepájanie rozdielnych častí organizácie za účelom splnenia kolektívnej množiny úloh.

Koordinácia v procese softvérového vývoja znamená, že rozliční ľudia pracujú na spoločnom projekte, pričom si vymieňajú informácie a ich aktivity sa vzájomne preplietajú. Musia mať však rovnaký pohľad na vyvíjaný softvér najmä z hľadiska jeho budúcej funkcionality, organizácie a prepojenia s už existujúcimi softvérovými systémami. Aby ho postavili efektívne, musia zdieľať podrobnú špecifikáciu návrhu a informácie o vývoji jednotlivých softvérových modulov. Aby som to zhrnul, musia skoordinovať svoju prácu takým spôsobom, aby sa čo najrýchlejšie úspešne ukončila, nič sa nerobilo zbytočne a vo výsledku všetko do seba zapadalo.

Charakteristika softvérového vývoja

Dosiahnutie vyhotovenia úspešného softvérového systému vyžaduje dôslednú koordináciu celého vynaloženého úsilia v cykle softvérového vývoja. A práve táto koordinácia sa nedosahuje vždy najľahšie.

Základnou črtou väčšiny softvérových systémov je práve ich rozsiahlosť, ktorá zabraňuje jednotlivcovi alebo malej skupine ich vytvorenie alebo dokonca ich dokonale pochopenie. Ak by softvérový systém bol malý, celý vývoj by mohol byť efektívne koordinovaný, pretože jednotlivec alebo malá skupina by mohli riadiť svoju prácu a zamerať sa na všetky detaily implementácie. Často rozsiahle projekty sú oveľa úspešnejšie, ak jedna, často výnimočná osoba so znalosťami z aplikačnej domény ako aj znalosťami softvérového riadenia koordinuje projekt. Avšak toto je nereálne pre rozsiahle softvérové projekty, kde veľkosť systému sa meria v miliónoch alebo desiatkach miliónoch riadkov kódu a životnosť projektu v rokoch.

Problémy koordinovania rozsiahlych projektov doplňuje miera neistoty. Pod neistotou môžeme rozumieť nepredvídateľnosť softvéru ako aj výsledkov úloh softvérových inžinierov. Na rozdiel od výroby, vývoj softvérového produktu nie je rutinná aktivita. Veľa softvérových systémov sú jedinečné projekty s neexistujúcimi prototypmi aplikácií alebo systémami, ktoré by sa jednoducho modifikovali alebo zmenili. S vývojom softvéru i miera neistoty narastá, pretože špecifikácia funkcionality softvéru sa v čase mení, ako sa menia potreby používateľov a vyvíjajú konkurenčné systémy.

Veľký rozsah a neistota v softvéri by bol skoro zanedbateľný problém, keby softvérový produkt nevyžadoval precíznu integráciu zo svojich softvérových komponentov. Veľa softvérov je zložených z tisícok modulov, ktoré musia do seba zapadať, aby softvérový systém fungoval korektne. Slabá koordinácia medzi podskupinami produkujúcimi softvérové moduly môže viesť k chybám pri integrovaní samotných modulov.

V softvérom inžinierstve sa ukázalo, že praktické skúsenosti tímov z prechádzajúcich projektov nevyriešili problémy koordinácie vo veľkých softvérových projektoch.

Na potlačenie softvérovej krízy sa najčastejšie ponúkajú tri prístupy: (1) technické nástroje - od nových pracovných staníc k syntaxou riadením editorom až po vyššie programovacie jazyky, na zvýšenie produktivity jednotlivých vývojárov, (2) stavebnicovosť - technická ako napr. objektovo–orientované programovanie a manažérska, ako delenie do tímov podľa požiadaviek a (3) formálne procedúry - technické ako CASE nástroje, špecifikačné nástroje a manažérske ako testovacie plány, rozpisy odovzdaní a dokumenty požiadaviek.

Aj keď tieto techniky nepochybne prispeli k miernemu rastu v produktivite softvéru za posledných dvadsať rokov, zaoberali sa iba čiastočne problémami koordinácie vo vývoji softvéru. Nástroje na zvýšenie produktivity programátorov ako jednotlivcov nevyriešili koordinačné problémy. Podobne, vrstvové architektúry a štruktúrované programovacie techniky môžu zredukovať množstvo rozhraní medzi modulmi, avšak ľudia z rozdielnych tímov sa stále musia dohadovať, čo sa postaví a musia vhodne skladať moduly softvéru.

Komunikácia ako prostriedok zlepšenia koordinácie

Popredné výskumy ukazujú, že formálna a neformálna komunikácia sa najlepšie hodí pre rôzne typy aktivít. Pod formálnou komunikáciou rozumieme písomnú komunikáciu alebo štandardné stretnutia. V prípade softvérového vývoja, formálna komunikácia zahŕňa techniky ako napr. písanie dokumentov špecifikácie, formálne špecifikačné jazyky a automatizované ohlasovanie a stopovanie chýb v programe. Tieto techniky kontrastujú s neformálnou komunikáciou, pod ktorou rozumieme osobnú a interaktívnu komunikáciu. Formálna komunikácia je užitočná pre koordinovanie transakcií v skupinách a organizáciách, avšak často zlyháva tvárou v tvár neistote, ktorá je typická pre väčšinu

práce so softvérom. Za týchto podmienok môže byť neformálna komunikácia potrebná pre koordináciu.

Práve kvôli vzájomnej previazanosti jednotlivých skupín pracujúcich na vývoji softvérového projektu, neformálna, medziľudská komunikácia by mohla byť cenná metóda dosiahnutia potrebnej koordinácie. Ale v rozsiahlejších tímoch, neefektívnosť vzájomnej komunikácie každej dvojice ľudí môže zabrániť používaniu neformálnej komunikácie ako praktickej techniky na riešenie problémov koordinácie.

Koordinácia problémov a prístupy

Mnoho softvérových organizácií po celom svete by mohli z veľkej časti eliminovať koordinačné problémy, keby si dokázali správne odpovedať na otázku: „Kedy je najlepší čas na koordináciu?“ Odpoveď na túto otázku je veľmi dôležitá, pretože zdržiavanie procesu koordinácie od určitého bodu môže viesť k drahému prepracovávaniu softvéru, avšak predčasná koordinácia môže byť kontraproduktívna, pretože môže narušovať prácu na vývoji softvéru.

V súčasnosti sa používajú tieto tri prístupy [3]:

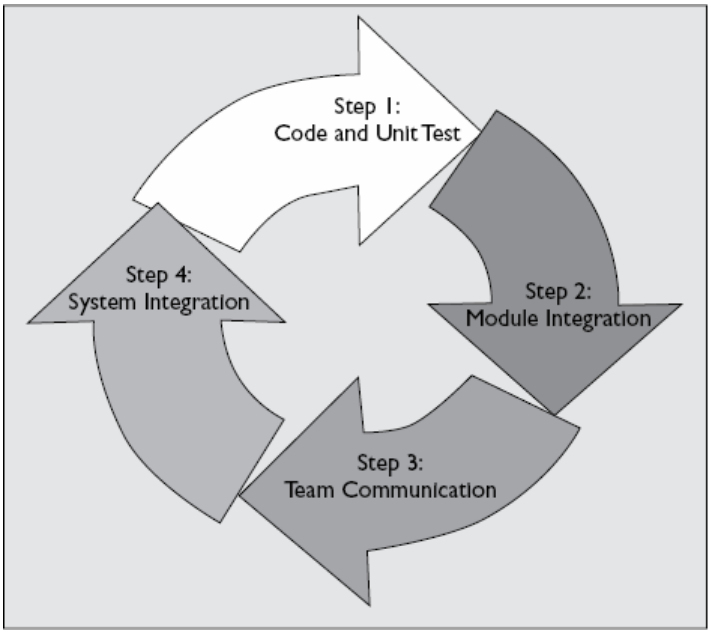
1. veľký tresk
2. častá integrácia a periodická synchronizácia
3. chybami poháňaná koordinácia

Tieto prístupy sa od seba odlišujú rozdielnym spustením procesu koordinácie.

Veľký tresk (Big Bang)

Ide o prístup, kde celá koordinácia nastáva na konci projektu. V tomto prístupe sú kroky 2, 3 a 4 z Obr. 1 pozdržané, až pokiaľ nie je skompletizovaný krok 1. *Big Bang* nasleduje vodopádový model a tak nejde o inkrementálnu vývojovú techniku. Od načasovania koordinácie už sama o sebe nie je aktívne organizovaná, a preto má tento prístup mimoriadne nízke nároky na organizáciu projektu a je veľmi vhodný pre malé tímy pracujúce na dobre rozpracovanom projekte.

Hlavným nedostatkom tejto koordinácie je rozšíriteľnosť. Softvérové komponenty v systéme môžu mať vzájomne komplikované vzťahy. Ak sa tieto vzťahy nezjednodušia dostatočne skoro, projekt sa stane komplikovaným a horšie modifikovateľným v budúcnosti. Takéto vedľajšie efekty robia prístup *Bing Bang* veľmi nákladným pre väčšie projekty.



Obr.1. Činnosti počas konštrukčného cyklu. (zdroj [3])

Častá integrácia a periodická synchronizácia

Softvérové organizácie v tomto prístupe teraz uskutočňujú integráciu modulov (krok 2 v Obr. 1) oveľa častejšie, často denne. Ide o dobre zdokumentovaný a často publikovaný prístup prebraný softvérovým gigantom, firmou Microsoft. Takzvaný „*Daily Build and Smoke Test*“ bol použitý pri vývojoch mnohých projektov v tejto firme. Popri častej integrácii modulov, prebieha komunikácia v tímoch a systémová integrácia, aby sa zabezpečila kvalita softvérového produktu počas celého vývoja.

Zatiaľ čo častá koordinácia na úrovni modulov aj systému pomáha zmierňovať vedľajšie efekty (ako napr. komplikované rozhrania modulov), dôležitá otázka zostáva nezodpovedaná. Ako dlho má trvať vývoj v každom cykle ?

V mnohých organizáciách si môžeme všimnúť, že základná koordinačná politika prebieha nasledovne. Koordinácia je najintenzívnejšia na začiatku projektu, počas neho poľavuje a blízko konca projektu znovu stúpa. Tento trend môže byť vysvetlený ako výsledok dvoch faktorov: tímového učenia sa a systémovej stability.

Chybami poháňaná koordinácia

Zatiaľ čo, koordinačné politiky založené na čase majú jasné organizačné prínosy, nejavia sa plne schopnými pokryť dynamiku vyvíjajúceho sa projektu. Všeobecne platí, že

koordinácia je oveľa naliehavejšia, keď sa zdá, že systém stráca synchronizáciu, inak je vhodné nechať prácu na vývoji softvéru pokračovať. Z tohto vyplýva, že koordinačné rozhodnutia by mali byť nejako späté s aktuálnym stavom systému.

S pokročilým vývojom a nástrojmi riadenia projektu, je teraz možné získať údaje o chybách v systéme a ďalšie príbuzné metriky v skorých fázach projektu. Projektoví manažéri s použitím týchto údajov môžu plánovať koordináciu vždy, keď priemerné náklady na opravu chýb začnú rásť.

Tento typ koordinácie je mimoriadne prínosný pre zhustené časové plány, teda také, kde pomer požadovanej práce k dostupnému času je vysoký.

Faktory ovplyvňujúce koordináciu

Intenzita koordinácie je pomer úsilia vynaloženého na koordináciu v jednom konštrukčnom cykle. Faktory ovplyvňujúce intenzitu koordinácie by sa dalo rozčleniť do štyroch skupín:

- projekt
- tím
- systém
- technológia

Projekt

Kritickým faktorom ovplyvňujúcim produktivitu tímu je prípustný čas vývoja softvérového systému. Pre danú množinu požiadaviek, kratší čas vývoja softvéru sa dosiahne, ak bude na ňom pracovať väčší tím, avšak za cenu drahšej koordinácie medzi členmi tímu. Na druhej strane, zväčšovanie veľkosti tímu znižuje výslednú produktivitu na jedného člena.

Tento fenomén naznačuje, že neuvážené pridávanie ďalších členov do tímu môže od istej hranice skutočne predĺžiť trvanie projektu. Jediným rozumným východiskom pre veľké tímy je ich rozdelenie na relatívne menšie skupiny a zavedenie hierarchickej komunikačnej štruktúry. Takto sa tímy vývojárov môžu sústrediť na dobre prepojiteľné komponenty systému.

Skúsenosť tímu a efekt učenia sa

Mnohé štúdie ukázali až desaťnásobný rozdiel v produktivite medzi nováčikom a skúseným vývojárom. Jenou z čŕt prvotriednych vývojárov je ich schopnosť získavať projektovo-špecifické vlastnosti a vyhnúť sa väčšiemu prepracovávaniu, dokonca v pre nich neznámej projektovej doméne. Dôkazom tohto procesu učenia je, že vývojový tím implementuje a optimalizuje novo vyvíjané moduly oveľa efektívnejšie ako predošlé a tak

dochádza k zlepšeniu schopností tímu a k rýchlejšiemu a kvalitatívne lepšiemu pokroku v projekte.

Softvérové charakteristiky systému

Čím väčšia je zložitosť systému, tým väčšia by mala byť intenzita koordinácie. Tento pomer však v žiadnom prípade nie je priamoúmerný. V zložitých systémoch sa každý nesúlad v jednotlivých moduloch ťažko opravuje a patrične zvyšuje náklady. Intenzívnejšia koordinácia preto pomáha udržiavať chybovosť v rozumnej miere. Náklady na celkovú koordináciu taktiež narastajú so zložitosťou systému.

Pokiaľ nie je na projekt vyčlenených dostatok finančných, materiálnych aj ľudských zdrojov alebo je projekt v časovom sklze, projektoví manažéri sú často v pokušení preskočiť fázu návrhu, aby sa viac času venovalo produktívnej implementácií. Často by pre nich bolo lepšie investovať ešte nejaký čas do prvotriedneho návrhu, aby predišli časovo náročnému a drahému prerábaniu systému v priebehu konštrukcie.

Technológie a nástroje

Zmeny v produktivite možno merať tým ako ľahko členovia tímu vedia skĺbiť svoje pracovné prostredie s vývojom a koordináciou. Iným meradlom môžu byť skúsenosti z danej problémovej oblasti. Jednoducho povedané, ak tím presne vie, čo má robiť, ako, kedy a kde, produktivita bude vyššia, ako keby si najskôr naštudoval celú problémovú doménu.

V organizácii so sofistikovanou CASE podporou je tím vedený ku koordinácií oveľa častejšie. Technologické inovácie, ktoré ponúkajú nové nástroje, sa často krát považujú za pomôcky na zrýchlenie vývoja celého projektu. V prípade správneho použitia dokážu odbúrať nadbytočnú koordináciu. Organizácie zaoberajúce sa vývojom softvéru si v súčasnosti môžu vybrať z pestrej palety nástrojov a technológií.

Je preto úlohou dobrého a efektívneho manažmentu projektového procesu rozhodnúť sa, či finančné prostriedky radšej investuje do nových a kvalitných technológií, alebo do ďalších zamestnancov.

Motivácia ako prostriedok zlepšenia produktivity

Osobne si myslím, že najlepší spôsob, ako zvýšiť produktivitu v ktorejkoľvek inštitúcií je vhodným spôsobom motivovať jej zamestnancov. To znamená vytvoriť im také pracovné prostredie, v ktorom sa budú cítiť príjemne a nebude ich pri práci nič vyrušovať. Predsa len, ak zamestnanec trávi jednu tretinu pracovného dňa na nejakom pracovisku, začne toto miesto nazývať druhý domov. Zároveň by mali byť za svoju prácu adekvátne finančne ohodnotení.

V mnohých softvérových firmách sa osvedčil prémiový spôsob odmeňovania zamestnancov. Pracujúci dostávajú primeraný mesačný plat a na konci roka sa im podľa zisku firmy a ich pracovnej činnosti vypočítavajú prémie. Tie tvoria u tých najúspešnejších zamestnancov niekoľko násobok ich mesačného platu. Prémie sa samozrejme rozdávajú iba, ak bola daná firma počas roku úspešná a zisková. Takto sú zamestnanci motivovaní a spoločnými silami sa snažia o to, aby podnik čo najlepšie prosperoval.

Ďalším výborným spôsobom zvýšenia produktivity zamestnancov je, podľa môjho názoru, umožniť im ďalej sa vzdelávať, rozvíjať svoje schopnosti, ale hlavne im umožniť úspešný postup v povolaní. Je zrejmé, že ak sa zamestnanec vo firme cíti dobre a vie, že to niekam môže dotiahnuť, pracuje usilovnejšie, aby si to jeho nadriadení všimli.

Na predchádzajúcich riadkoch som uviedol niekoľko spôsobov pozitívneho motivovania pracovníkov v podniku. Avšak motivácia ako prostriedok na zlepšenie produktivity môže byť aj negatívna.

Typickým príkladom negatívneho motivovania zamestnancov sú praktiky, ktoré prevádzkuje nielen manažment slovenskej verejnoprávnej televízie. Hlavnou myšlienkou tejto motivácie je navodiť u zamestnancov strach o stratu zamestnania. Slová personálneho manažmentu sú prosté: „Ak sa vám u nás nepáči, tak môžete odísť“. Neviete si predstaviť, aký obrovský účinok majú tieto slová najmä na starších pracovníkov tohto podniku, pretože vedia, že čím je človek starší, tým ťažie si nájde novú prácu. Personálny manažment toho patrične využíva a okrem množstva práce, ktorú stále kladie na plecia takýchto zamestnancov, nechce im zvyšovať plat. A tak sa pýtam, ako je možne, že vyštudovaný inžinier, ktorý pracoval tri desiatky rokov v tejto televízii, má sotva taký mesačný plat, ako nastupujúci ekonóm ?

Záver

Správne načasovaná koordinácia celého procesu vývoja softvéru umožní firmám dodržiavať dohodnuté termíny. Zákazníci budú spokojní a firmy takto ušetrený čas a finančné prostriedky, ktoré by inak padli na „hasenie“ projektu, môžu investovať práve do motivácie svojich zamestnancov a nových technológií. Tým dosiahnu ešte väčší nárast produktivity a za ten istý čas dokážu zvládnuť a úspešne dokončiť viac projektov.

Celkom na záver by som si dovolil odpovedať na otázku položenú v úvode. Aký bol hlavný dôvod meškania projektov? Podľa môjho názoru išlo vo väčšine prípadov práve o pochybenie projektového manažmentu, pretože neodhadli korektne plán projektu a neskoordinovali správne celý proces vývoja softvéru.

Použitá literatúra

1. Scacchi, W.: Understanding software productivity. *Advances in Software Engineering and Knowledge Engineering*, D. Hurley (ed.), Volume 4, (1995) 37-70.
2. Kraut, E.R., Streeter A.L.: Coordination in software development. *ACM Press*, Vol.38, No. 3 (March 1995) 69-81.
3. Chiang, R.I., Vijay S., Mookeriee.: Improving software team productivity. *ACM Press*, Vol.47, No. 5 (May2004) 69-81.

Annotation

Coordination's affect within software development process on the software team productivity

Since its inception, the software industry has been in crisis. Software has been unreliable, delivered late, unresponsive to change, inefficient and expensive. Even today, problems with software systems are common and highly-publicized occurrences. Major contributor is the problem of coordinating activities while developing large software systems, therefore coordination becomes much more difficult as project size and complexity increases.

In particular, we examine the respective roles of formal and informal communication mechanism in coordinating work on software projects. Most of the existing coordination support tools have used formal communication procedures, but there is a need for informal communication procedures as well.

Many approaches of software development coordination problems exist. We describe three current approaches being used: Big Bang, Frequent integration and periodic synchronization and Fault-driven coordination. We describe factors that affect coordination intensity.

The end of essay belongs to motivation as another remedy for productivity increase.

Agilné metódy vývoja softvéru a rozsah projektu

MARTIN KOMARA

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

Abstrakt. Vplyv softvérových systémov na náš život je obrovský a rastie každým dňom. Softvér sa stal nevyhnutným pre fungovanie ľudskej spoločnosti. Softvérové systémy patria k najzložitejším výtvorom ľudstva vôbec, čo spôsobuje problémy pri ich tvorbe a prevádzke. Vysoká zložitosť softvérových systémov viedla ku vzniku veľkého množstva prístupov a metód, ktoré sa snažili o systematický prístup k tvorbe softvéru. Napriek tomu len málo systémov bolo dodaných do prevádzky za vopred dohodnutých podmienok. V situácii, keď sa zvyšuje tlak, ktorý núti vytvárať kvalitný softvér za menej peňazí a za kratší čas, vznikli nové, tzv. agilné metódy. Agilné metódy vznikli zo skorších prístupov k rýchlemu vývoju aplikácií (RAD), ktoré rozpoznali, že spomedzi obmedzení projektu tvorených časom, cenou a rozsahom má práve rozsah najvyššiu mieru neurčitosti. Preto sa agilné metódy, na rozdiel od tých tradičných, nepokúšajú určiť rozsah projektu v úvodných fázach, ale pracujú s používateľskými požiadavkami oveľa flexibilnejšie. Tento príspevok rozoberá nové zaujímavé postupy na určovanie rozsahu projektu pri použití agilných metód vývoja softvéru.

Úvod

Odkedy bol vymyslený tento svet, stretávajú sa jedinci ľudského rodu s najrôznorodejšími obmedzeniami. Najskôr išlo o rieku, ktorá zabraňovala prapredkom dostať sa k zdrojom tukov, bielkovín a cukrov, pokojne sa pasúcim na druhej strane. Potom to bol oceán, ktorý sa stal poslednou zastávkou mnohých dobrodruhov (traduje sa napríklad príbeh istého Janovčana, ktorý neumrel na mori od hladu a smädu len vďaka tomu, že sa medzi Európou a Áziou, kam sa dobrodruh pôvodne chcel dostať, zhodou okolností vyskytol ešte jeden dovtedy neznámy kontinent). Neskôr ľudstvo vypliešťalo oči, keď zistilo, že sa aj tak nedostane nikam, pretože sa nachádza na jednej veľkej guli. A aby sme náhodou neboli nezdravo optimistickí po objave prvej kozmickej rýchlosti, veľmi rýchlo bolo ľudstvo

oboznámené s drobným obmedzením nazývaným hraničná rýchlosť vo vesmíre. Existujú jedinci, ktorí sa rozhodli bojovať s obmedzeniami, ktoré život prináša. Títo sa dajú rozpoznať podľa toho, že ich hroby bývajú vytvorené ad-hoc, navŕšené z dostupných materiálov. My ostatní zbabelí, túžiaci po dôstojnom mieste posledného odpočinku, sa musíme naučiť s týmito obmedzeniami žiť.

Taký softvérový projekt, napríklad. Na softvérový projekt sa vzťahujú najmä tri obmedzenia: Čas, cena a rozsah.

Diabolská trojica

Pre mnohé projekty je čas absolútne rozhodujúcim kritériom. Je to najmä v prípadoch, keď je tento dátum daný externými okolnosťami, na ktoré nemá organizácia nijaký vplyv. Príkladom by mohla byť banka, ktorá musí upraviť svoj informačný systém tak, aby spĺňal nové legislatívne požiadavky pred tým, než legislatívna úprava nadobudne účinnosť. Iným príkladom takéhoto projektu môže byť tvorba eseje, ktorú musí študent odovzdať predtým, ako prof. Bieliková zruší možnosť vkladania do redakčného systému.

Cena je pre projekt mimoriadne dôležitá. Ako inak, veď ide o peniaze. Výsledná cena pozostáva z dvoch častí: nákladov a zisku. Náklady sú tvorené nákladmi na softvérové a hardvérové vybavenie (vrátane údržby), nákladmi na školenia, nákladmi na pracovnú silu a nákladmi na réžiu. Zatiaľ čo náklady na softvér a hardvér sa dajú dobre odhadnúť, náklady na pracovnú silu závisia od času, koľko bude daný projekt napokon trvať. Ako teda vidno, cena závisí od času.

Funkcionalita systému je tretie obmedzenie. Neschopnosť dodať prisľúbenú funkcionalitu v dohodnutom rozsahu znamená nedodržaný prísľub, čo sa prejavuje na obchodných vzťahoch a má nepriaznivé dopady na ďalšiu spoluprácu. Väčší rozsah projektu si samozrejme vyžaduje viac času a/alebo zdrojov. Z toho vyplýva, že čas a cena projektu sa odvíja od jeho rozsahu.

Tradičné metódy, tradičné problémy

Tradičný model manažmentu softvérového projektu sa zameriaval na zafixovanie rozsahu, z ktorého sa potom vychádzalo pri dohode so zákazníkom o cene a čase. Je tu však malý háčik. Ak je najskôr definovaný rozsah, vývojári sú nútení spraviť odhad úsilia, ktoré bude potrebné vykonať na uvedenie daného produktu do života. Problém, ktorý iste nepoteší, je, že spomedzi času, ceny a rozsahu má práve rozsah najväčšiu mieru neistoty. Neurčitosť vykonaných odhadov na základe takto určeného rozsahu je veľmi vysoká a môže vyústiť do situácie, že skutočné a odhadnuté množstvo potrebných zdrojov a času sa líši až o 200%.

V takejto situácii vznikol veľký tlak na manažérov, aby vytvárali presnejšie odhady. Vznikli nové techniky, ktorých využitie malo vyústiť do lepších, presnejších odhadov. Využívanie týchto techník však znamenalo stále väčšie a väčšie množstvo papierovania, s čím súvisela potreba veľkého množstva času na vytváranie, čítanie, konzultáciu a schvaľovanie. O škodách na lesoch ani nehovoriac. A to bol v skratke príbeh, ako sa zrodili tzv. ťažkotonážne, tradičné metódy vývoja softvéru.

Tradičné metódy vývoja softvéru definujú rozsah produktu pomocou rozpisu práce, počnúc vysokoúrovňovými požiadavkami, ktoré boli ďalej dekomponované na špecifickejšie požiadavky [3]. Tento postup dáva manažérovi k dispozícii presnejší odhad času a potrebných zdrojov (inými slovami výslednej ceny) celého projektu, pretože čím presnejšie je špecifikovaný projekt a potrebné činnosti, tým sú jednotlivé odhady lepšie. Akonáhle je vytvorený takýto rozpis prác, úlohou manažmentu rozsahu projektu ostáva sledovanie, aby nedošlo k zmene rozsahu, najmä formou nekontrolovanej zmeny požiadaviek.

Tieto metódy fungujú dobre, pokiaľ nie je produkt príliš zložitý a požiadavky sú konzistentné a zlučiteľné. Bohužiaľ, v praxi bývajú produkty príliš zložité na to, aby ich bolo možné úplne definovať, čo vyúsťuje do skutočnosti, že výstupy z týchto metód bývajú nespoľahlivé a zavádzajúce.

Dôkaz, že pod tlakom sa pracuje mimoriadne ťažko, zverejnila v roku 1994 The Standish Group, keď uverejnila štúdiu, podľa ktorej iba 16,2% softvérových projektov bolo dodaných načas, so špecifikovanou funkcionalitou, a nebol prekročený plánovaný rozpočet. Pri 31,3% projektov nebol dodržaný dohodnutý čas, cena alebo rozsah. Zvyšných 52,7% projektov bolo zrušených. Ako najčastejším dôvodom, prečo došlo k zrušeniu projektu, uviedli respondenti nekompletnú špecifikáciu požiadaviek [4].

V takejto situácii sa akiste nemožno čudovať snahám nahradiť tieto tradičné metódy novými, efektívnejšími metódami vývoja softvéru, ktoré by umožnili vyhnúť sa nutnosti dopredu vytvárať odhady, ktoré sa napokon aj tak ukážu ako nepresné.

Agilné metódy na scénu

Agilný znamená schopný pohybovať sa rýchlo, flexibilne, s vysokým stupňom zručnosti a kontroly. Agilné metódy sú navrhnuté tak, aby boli flexibilnejšie. Zatiaľ čo pri tradičných metódach prevládalo presvedčenie, že pri dostatočnom úsilí sa nám podarí vytvoriť kompetnú sadu požiadaviek na finálny systém, ktoré sa už v ďalších fázach životného cyklu nebudú meniť, agilné metódy pokladajú zmenu požiadaviek za niečo, čo nastane bez ohľadu na úsilie, ktoré vynaložíme na to, aby sme tejto zmene zabránili.

Za kľúčovú sa považuje komunikácia v tíme, ktorú nedokážu nahradiť ani moderné komunikačné prostriedky, ako sú napr. email, videokonferencie či mobilné telefóny. Kontakt zoči-voči stále zostáva najpriamejšou a najefektívnejšou formou komunikácie.

Ako už bolo uvedené, jednou z najčastejších príčin zlyhania softvérového projektu je nedostatočné zapojenie zákazníka do procesu vývoja. Zákazník, ktorý je k dispozícii vývojovému tímu alebo je priamo jeho súčasťou, je kľúčovou požiadavkou všetkých agilných metód.

Zákazník by mal vedieť správne zodpovedať všetky otázky vývojového tímu, mať právo robiť záväzné rozhodnutia a rozhodovať sa správne a na rozdiel od tradičných metód vývoja softvéru by mal byť k dispozícii nielen v úvodnej fáze projektu, ale počas všetkých fáz životného cyklu.

Vzhľadom na to, že zapojenie zákazníka je hlavným cieľom agilných metód vývoja softvéru, najčastejšia technika získania požiadaviek na systém je rozhovor, ktorý je zdrojom priamych znalostí o probléme. Je známe, že reťazové predávanie si informácií vedie k nedorozumeniam. Preto všetky agilné metódy zdôrazňujú priamu komunikáciu so zákazníkom. V prípade, že vznikne nejasnosť alebo sú požiadavky definované príliš povrchne, člen vývojového tému by mal kontaktovať priamo zodpovednú osobu a vyhnúť sa komunikácii cez tretiu osobu.

Dôležitou vlastnosťou agilných metód je, že zmena sa, na rozdiel od tradičných metód, považuje za prirodzený jav v rýchlo sa meniacom prostredí a nepokladá za potrebné vyvíjať úsilie na jej elimináciu. Agilné metódy sa zameriavajú na vytvorenie takého návrhu, aby bolo možné zmeny jednoducho realizovať [2].

Rozsah

Ako sa teda určuje rozsah pri použití agilných metód? Treba povedať, že určovanie rozsahu funguje diametrálne odlišne ako pri tradičných metódach. V prvom rade sa výsledný čas projektu neodvíja od rozsahu, ale práve naopak. Urobí sa toľko, na koľko je dostatok času, pričom zákazník má plnú kontrolu nad tým, ktorá časť sa bude implementovať.

Agilné metódy sú charakteristické iteratívnym vývojom. Plán, čo sa bude v danej iterácii vykonávať, sa určí na jej začiatku v spolupráci so zákazníkom. Dĺžka trvania iterácie je obyčajne krátka, od dvoch do šiestich týždňov. Počas každej iterácie dôjde k implementácie niekoľkých cieľov.

Požiadavky by mali smerovať k výsledkom, ktoré sú pre koncového zákazníka nejakým spôsobom užitočné. Požiadavky, ktoré sa majú implementovať, sú uložené v rade s prioritou. Túto prioritu určuje zákazník v spolupráci s vývojovým tímom. Bežnou praxou je najprv implementovať požiadavky s najvyššou prioritou, ktoré prinášajú najvyššiu hodnotu pre zákazníka. Počas vývoja sa zlepšuje porozumenie problému a nové požiadavky na systém sú pridávané. Priorizácia je opakovaná často počas celého procesu vývoja, aby sa zabezpečilo, že všetky požiadavky sú aktuálne.

Je potrebné si uvedomiť, že priorita jednotlivých požiadaviek sa mení v čase. Napríklad priorita požiadavky, ktorá smeruje k splneniu legislatívnej normy, sa bude zvyšovať s približujúcim sa termínom účinnosti danej normy.

Preto je dôležité získať prioritu pre každú požiadavku z rozsahu meniacu sa v čase. Manažér by sa mal dohodnúť so zákazníkom na rozsahu, ktorý bude obsahovať niekoľko požiadaviek s nižšou prioritou, ktorá sa bude postupom času zvyšovať. Vývojový tím zahrnie tieto požiadavky do plánu, avšak zákazník súhlasí s tým, že niektoré z nich sa nepodarí dokončiť načas v prípade neočakávaných problémov.

Ako sa zlepšuje pochopenie problému počas jeho vývoja, môže sa stať, že sa zmení priorita niektorých požiadaviek, alebo dôjde k jej úplnému vypusteniu, nakoľko sa zistí, že splniť danú požiadavku nie je potrebné pre zabezpečenie obchodných cieľov.

Dôležitou vlastnosťou v súvislosti s radom požiadaviek je možnosť dohodnúť sa na rezerve pre danú iteráciu. Najprv by malo byť odsúhlasené celkové množstvo požiadaviek, ktoré sa budú implementovať v danej iterácii, na základe odhadu potrebného úsilia a dátumu dodania. Výsledkom je celkový počet požiadaviek, ktoré sa budú implementovať v danej iterácii. Potom sa dohodne rezerva, čo znamená, že požiadavky sa rozdelia do troch kategórii: na tie, ktoré je absolútne nevyhnutné implementovať, tie, ktoré by bolo potrebné implementovať a tie, ktoré by bolo dobré implementovať.

Snaha o kategorizovanie požiadaviek do týchto kategórií však naráža na neochotu zákazníka označiť akúkoľvek požiadavku pre danú iteráciu za inú ako nevyhnutne implementovanú. Zákazník bude predpokladať, že takto kategorizované požiadavky nebudú nikdy realizované. Tento predpoklad je založený na zlej skúsenosti so spoločnosťami, ktoré nezrealizujú čo prisľúbia, a celkom určite nikdy nezrealizujú, čo neprisľúbia. V takomto prípade sa nedá dohodnúť na rozsahu projektu, čo znamená riziko pre celý projekt [1].

Záver

Agilné metódy vývoja sú výzvou pre softvérových manažérov a klientov, ktorí sa musia vzdať závislosti na objemnej dokumentácii a medzivýsledkoch, tvorených výstupmi z jednotlivých fáz životného cyklu programu. Nástroje, ktoré sa používajú namiesto objemnej dokumentácie a rozpisu prác sú najmä zlepšená komunikácia so zákazníkom, priorizácia požiadaviek a krátke trvanie iterácie, na ktorej konci je fungujúci softvér.

Agilné metódy opúšťajú snahu zadefinovať rozsah vytvorením kompletného zoznamu požiadaviek na začiatku projektu, nakoľko odhad celkového úsilia potrebného na dokončenie projektu na základe rozpisu prác má v sebe veľkú mieru neurčitosti.

Agilné metódy umožňujú prispôsobovať rozsah projektu meniacim sa podmienkam prostredia a zohľadňujú vývoj toho, čo zákazník požaduje na základe dojmov z práce so softvérom dodaným v predchádzajúcich iteráciách.

Používateľ má možnosť v spolupráci s vývojovým tímom zmeniť svoje požiadavky na začiatku každej iterácie, ako aj zmeniť prioritu už existujúcich požiadaviek tak, aby dokázal čo najlepšie naplniť svoje obchodné ciele. Agilné metódy týmto vnášajú do rozsahu projektu flexibilitu, aká nebola možná pri použití tradičných metód tvorby softvéru.

Použitá literatúra

1. Anderson, D.J., Schragenheim, E.: Agile Management for Software Engineering: Applying the Theory of Constraints for Business Results. Prentice Hall, PTR, New York, 2003.
2. Beck, K., Andres, C.: *Extreme Programming Explained: Embrace Change, Second Edition*. Addison Wesley Professional, 2004.
3. Bieliková, M.: *Manažment v softvérovom inžinierstve*. 1999.
4. *Chaos Report, Few IS Projects Come in on Time, on Budget*. Computer World 12, 1995.

Annotation

Agile software development methods and project scope

Impact of software systems on everyday life is ample, and continues to grow day by day. Software became inevitable for the existence of human society. Software systems are accounted to be among the most complex creations of mankind at all, causing troubles with their creating and maintaining. High degree of complexity led to the creation of great number of approaches and methods for systematic approach to software system manufacturing. Despite this effort, very few systems have been delivered to customer on time, on budget, and within scope. In a situation where the pressure to create quality software with thinner budget and shorter time to market is constantly growing, new so called agile methods have emerged. They are based on earlier rapid application development (RAD) approaches, which recognised that among project constrains represented by time, cost and scope is the scope one single constraint with the greatest measure of uncertainty. It is for this reason why agile methods unlike traditional methods try not to settle project scope in the early phases of the project; rather they introduce greater flexibility to coping with user requirements. This paper covers new and interesting approaches for determining project scope using agile software development methods.

Chyby a manažment softvérového projektu

TOMÁŠ TATRANSKÝ

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

Abstrakt. Každý softvér sa počas svojho vývoja musí vysporiadať so vznikom chýb. Vznik chýb je neodvratný pri každej ľudskej činnosti, preto je nutné zaoberať sa problémom vzniku chýb a snažiť sa o ich odstránenie. Tento dokument sa zaoberá rozborom kritických miest softvéru, pri ktorých často vznikajú chyby. Spomína aj rozšírenia softvéru ako systém pluginov a podpora skriptovania, ktoré môžu byť zdrojom chýb, prípadne môžu tvoriť bezpečnostné riziko. Ďalej sa zaoberá možnosťami manažmentu ako predchádzať vzniku chýb a ako zabezpečiť úspešnosť softvérového projektu. Na záver článok diskutuje súčasný stav v zodpovednosti za chyby a škody spôsobené chybami softvéru. Uvádza argumenty za aj proti zavedeniu zodpovednosti za chyby.

Úvod

Tvorba softvérových produktov je oblasť, v ktorej sa chyby vyskytujú v každej fáze vývoja projektu. Preto je veľmi dôležité brať do úvahy všetky možnosti vzniku chýb a čo najviac sa im snažiť vyvarovať. Chyby môžu nastať a vznikajú aj za ideálnych podmienok, ak je však manažment projektu zanedbaný, veľa vzniknutých chýb môže ostať neodhalených a šanca, že vzniknú nové chyby sa zväčšuje. Ak sa na zásadnú chybu narazí až v neskoršej fáze vývoja projektu, veľmi často sa stáva, že pokus o jej odstránenie hrubo naruší celý vývoj a môže projekt odsúdiť k zániku.

Preto je veľmi dôležité zaoberať sa pri manažmente softvérového projektu otázkou minimalizácie chýb v projekte na strane tvorby samotného kódu, ale rovnako aj na strane manažmentu softvérového projektu. V ďalšom texte podrobnejšie predstavíme oblasti, v ktorých často vznikajú chyby. Následne popíšeme postupy softvérového inžinierstva, vďaka ktorým môžeme vznik chýb znížiť a zároveň zabezpečiť skoršie odhalenie vzniknutých chýb. V poslednej časti sa zaoberáme zodpovednosťou za škody spôsobené chybami v softvéri a uvádzame argumenty za aj proti zavedeniu takejto zodpovednosti.

Časté oblasti vzniku chýb

Vznik chýb v softvérových projektoch je hlavne vďaka veľmi krátkej, iba niekoľko desaťročnej histórií softvérového inžinierstva bežná vec. Podľa prieskumu z roku 2003 [3] je priemerný počet chýb v softvéri na 1000 riadkov rovný 0.15, pričom najviac chýb softvér vykazuje, ak pochádza zo Spojených štátov amerických (až 0.4) a najmenej, ak pochádza z Japonska (0.02). Tieto čísla, pri zohľadnení faktu, že každý väčší produkt obsahuje aj viac ako milión riadkov kódu znamenajú, že počet chýb sa môže vyšplhať až k stovkám chýb v jednom produkte.

Ale aj napriek týmto nepriaznivým zisteniam sa čoraz viac darí pomocou zavádzania nových postupov odhaľovať vzniknuté chyby. Darí sa to najmä vďaka testovacím nástrojom ako je JUnit, statickou analýzou kódu, či používaním typovo bezpečných jazykov ako C# alebo Java. Čo sa nedarí znižovať je počet chýb, ktoré vznikajú pri tvorbe kódu, a to rovnako u začínajúcich programátorov, ako aj u skúsených programátorov s dlhoročnou praxou [4]. Zaujímavý je poznatok, že počet chýb u začínajúcich programátorov je približne rovnaký ako u tých skúsenejších, čo autori štúdie [4] odôvodňujú tým, že skúsenejší programátori pracujú na zložitejších úlohách a robia aj zložitejšie a ťažšie odhaliteľné chyby.

Veľkým problémom pri testovaní kódu môžu byť chyby v samotných testoch. Takéto chyby sa obyčajne rýchlo prejavia pri pokusoch testovať správny kód. Ak sa ale chyba v testovaní prejaví tak, že chybný test skončí úspešne pre správny aj chybný testovaný kód, môže to spôsobiť značné nepríjemnosti pri neskoršom testovaní produktu. Ďalším veľkým zdrojom nových chýb býva samotné odstraňovanie chýb. Na zamedzenie zavádzania nových chýb pri oprave starých chýb sa odporúča dôkladné testovanie všetkých dotknutých funkcií a modulov softvéru.

Manažment kvality softvéru sa okrem zabezpečenia zhody výsledného softvéru so špecifikáciou zaoberá aj kvalitou s ohľadom na počet chýb v softvéri. Z tohto druhého pohľadu je základným problémom kvality softvéru to, že programátori sú ľudia a ako takí aj oni robia chyby. Množstvo zanesených chýb do softvéru je ovplyvnené najmä zložitosťou a komplexnosťou systému. Čím je systém zložitejší, tým sa stáva menej prehľadný. Veľa skrytých chýb zostáva v softvéri aj po testovaní a navonok sa môžu prejaviť až pri dlhodobom nasadení, kedy môžu spôsobiť veľké škody. Chyby vznikajú aj počas prvotných fáz vývoja produktu, napríklad nepochopením požiadaviek zákazníka alebo zlým návrhom, ktorý je v rozpore so špecifikáciou. Tieto chyby stoja za mnohými predčasnými ukončeniami projektov a ich odstránenie je kľúčové pre úspešné dokončenie projektu. Úspešne vytvorený produkt zväčša obsahuje už iba chyby zanesené programátormi pri implementácií kódu. Väčšinou ide o preklepy, chyby z nepozornosti a chyby, ktoré vzniknú zanedbaním ošetrenia chybových stavov.

Bezpečnostné slabiny rozšírení softvéru

Bezpečnostné slabiny často vznikajú vďaka nedostatočne zabezpečeným rozšíreniam softvéru, ako je napríklad systém pluginov alebo podpora skriptovania v rámci softvéru. Záškodnícky kód, ktorý využíva takýto model, býva ťažko rozlíšiteľný od užitočného kódu a môže nastať situácia, kedy takéto rozšírenie prináša viac škody ako úžitku. Postupov ako sa chrániť pred záškodníckym kódom interpretovaným v rámci softvéru je viacero: analýza, prepísanie, monitorovanie a ďalšie. Analýza prebieha pred spustením kódu a v prípade zistenia nebezpečnosti je kód zamietnutý. Prepísanie je spôsob, pri ktorom modifikujeme nebezpečné časti kódu tak, aby nebol kód škodlivý. Monitorovanie prebieha počas vykonávania kódu a v prípade zistenia škodlivého kódu nastáva jeho zastavenie.

Možnosti zlepšenia kvality softvéru

V článku [1] autor uvádza množstvo poznatkov, ktorých aplikovaním môžeme skvalitniť proces vývoja softvéru. Ak budeme aplikovať tieto pravidlá, je pravdepodobné, že sa nám okrem skvalitnenia procesu vývoja podarí zároveň znížiť počet chýb v softvéri. Tu je niekoľko zásad pre vývoj softvéru, nielen z článku [1]:

Testovacie podmienky

Hneď ako je možné uzavrieť špecifikáciu podmienok, zmluvné strany by mali uvažovať ako bude možné otestovať splnenie požiadaviek. V skutočnosti mali nad týmto uvažovať už vo fáze tvorby špecifikácie, pretože je ľahké skĺznuť k testovacím podmienkam, ktoré sa nedajú reálne otestovať. Napríklad: „Program by mal uspokojivo spracovať vstup na výstup.“

Ak sa nám podarí správne špecifikovať podmienky pre testovanie, možnosť že chybu v programe neodhalíme je podstatne nižšia. Rovnako môžeme zabrániť vzniku chýb pri návrhu a implementácií vyplývajúcich z nepochopenia požiadaviek zákazníka.

Podrobná špecifikácia rozhraní

Presný popis jednotlivých formátov, špeciálne požiadavky na komunikáciu, ako napríklad kedy je možné prenášať niektoré typy dát. Všetky tieto detaily musia byť dohodnuté všetkými zúčastnenými stranami a spísané na papier. Táto podmienka v podstate patrí k tej predchádzajúcej. Ak má byť zabezpečená široká kompatibilita aj komunikácia pomocou implementovaných rozhraní, musí byť testovaná.

Skoré plánovanie a špecifikácia

Ak manažment pristúpi k tvorbe softvéru bez hlbšej analýzy a podrobného plánu, výsledkom bude odkladanie vytvorenia kompletnej špecifikácie a zdesenie, keď

nevytvorenie jedného programu oneskorí vývoj celého systému. Zákonite potom vzniká časový stres a počet chýb vytváraných vystresovanými programátormi sa zvyšuje. Čo môže mať za následok ďalšie oneskorenia pri vývoji alebo spôsobiť, že dodaný softvér bude nekvalitný a plný chýb.

Množstvo zamestnancov v počiatočných fázach

Analytici by nemali byť zaťažovaní povinnosťou zamestnať každého zamestnanca len preto, aby mal čo robiť. Kvantita nie je náhrada za kvalitu, môže veci iba zhoršiť. Je nutné, aby na návrhu systému pracovalo čo najmenej ľudí. O to viac by títo ľudia mali komunikovať medzi sebou, ale aj so zákazníkom.

Zodpovedná voľba jazyka

Výber programovacieho jazyka je rovnako ako výber celoživotného partnera ťažkou a dôležitou voľbou, ktorá sa nedá tak ľahko vziať späť. Ak už je programovací jazyk vybraný, stáva sa jeho zmena problematickou a projekt by to mohlo predražiť alebo úplne potopiť.

Plánovanie integrácie a nasadenia

Častá chyba pri plánovaní nasadenia softvéru do prevádzky je podcenenie času potrebného na integráciu softvéru po tom čo bol naprogramovaný. Pri integrácií vznikajú problémy aj pri veciach, kde by to nikto nečakal. Málo času vyhradeného na integráciu potom spôsobuje ďalší stres a prehliadanie menších nedostatkov na úkor zásadných chýb a problémov.

Skoré plánovanie testov

Akceptačné testy by mali byť určené čo najskôr, tak aby bolo možné testovať okamžite po implementácií. Mrhanie času pri vývoji softvéru sa nevypláca. Obyčajne sa totiž v takýchto prípadoch začnú objavovať problémy vtedy, keď už ich nikto nečakal a nie je na ne čas.

Testom riadené programovanie

Testy píšeme pred samotným písaním implementácie vlastností. Cyklus písania testov a písania funkčného kódu sa opakuje v malých iteráciách a často. Každá funkcia musí mať test ešte pred napísaním jej kódu. Testom je pokrytý každý riadok kódu programu. Funkčný testovací kód predstavuje okamžitú odozvu pri návrhu nových funkcií. Každý píše vlastné testy pre vlastný kód. Vývojové prostredie musí zabezpečiť rýchlu kompiláciu a služby pre testovanie. Návrh musí pozostávať z vysoko súdržných, voľne previazaných komponentov tak, aby bolo testovanie jednoduché.

Zodpovednosť za chyby

Prístup softvérových spoločností k zodpovednosti za škody spôsobené chybami softvéru je v súčasnosti žalostný. Ak si v dnešnej dobe inštalujete ľubovoľný softvér, musíte väčšinou odsúhlasiť text zmluvy, v ktorej sa autor zbavuje akejkoľvek zodpovednosti za chyby v softvéri. Takéto zbavenie zodpovednosti je logické a akceptovateľné ešte v prípade, že softvér je používateľovi ponúkaný zadarmo, bez nároku na odmenu, ak je však softvér poskytnutý za úplatu, je takéto správanie v slušnej spoločnosti nanajvýš podozrivé. Tieto praktiky sú dnes široko používané a väčšina používateľov ich nakoniec akceptuje, ale takéto správanie je v iných oblastiach podnikania neprípustné. Tak prečo by práve oblasť softvérového inžinierstva mala byť výnimkou? Ukazuje sa, že aj napriek textom zmlúv, ktoré musia používatelia akceptovať pri inštalácií, niektoré súdy majú na zodpovednosť za škody spôsobené chybami v softvéri iný názor a prisudzujú postihnutým používateľom odškodné [2]. Aj z tohto dôvodu sa softvérové spoločnosti začínajú viac venovať dôkladnému testovaniu vlastného softvéru pred uvoľnením na trh. Podobné zmluvy však určite v dohľadnej dobe nezmiznú. Ide hlavne o problém previazanosti rôznych softvérových výrobkov medzi sebou a fakt, že takéto zmluvy používajú súčasné operačné systémy, a preto softvér, ktorý sa vykonáva na týchto operačných systémoch nemôže vziať na seba zodpovednosť, ktorej sa zbavili tvorcovia operačných systémov.

Argumenty tvorcov softvéru

Tvorcovia softvéru sa bránia proti zavedeniu zodpovednosti za chyby a škody nimi spôsobené. Snažia sa presvedčiť verejnosť, že v súčasnosti nie je v ľudských silách zabezpečiť bezchybnosť softvéru, nie je možné úplne eliminovať chyby, že by tým mohlo dôjsť k spomaleniu vývoja softvéru, zvýšili by sa náklady na vývoj softvéru, a tým by došlo ku krachom viacerých spoločností, softvér by sa predražil. Často môže byť veľmi ťažké určiť pôvodcu chyby. Argumentujú, že o kvalitu softvéru sa postará samotné trhové prostredie a že súčasná podpora je dostatočná.

Argumenty používateľov

Medzi hlavné argumenty používateľov softvéru patrí skvalitnenie softvéru, čo je dôležitejšie ako rýchlosť vývoja nových verzií. Ďalej argumentujú stále vyššou mierou používania softvéru v bežnom živote ľudí a tým pádom stále väčšiu závislosť ľudí od softvéru. Softvér sa používa v podstate v každom odvetví ľudskej činnosti, od riadenia výroby v továrňach až po ovládanie rakiet s nukleárnymi hlavicami. Preto treba donútiť tvorcov softvéru k vyššej zodpovednosti, a tým zvýšiť kvalitu ich produkcie. Ďalším argumentom používateľov sú súčasné zákony pre ostatné oblasti ľudského obchodu. Zákazník je chránený obchodným zákonníkom 24 mesačnou zárukou, ktorá však na základe výnimky neplatí pri softvérových výrobkoch, pričom softvér je rovnaký produkt ako čokoľvek iné a nie je preto dôvod obrať zákazníka o jeho spotrebiteľské výhody.

Záver

Chyby sú súčasťou vývoja softvéru od vzniku softvérového inžinierstva a budú realitou aj v budúcnosti. Najväčší pokrok sa v súčasnosti deje v oblasti testovania kódu, pomocou ktorého je možné odhaľovať viac chýb ako v minulosti. Ďalší pokrok je možný v oblasti špecifikácie a návrhu softvéru a v oblasti dodržiavania zásad vývoja bezpečného softvéru. Najväčšie rezervy stále ostávajú na poli zodpovednosti za chyby v softvéri, kde si tvorcovia softvéru stále nedokážu priznať vlastnú zodpovednosť.

Použitá literatúra

1. Barry W. Boehm: Software Engineering As It Is, *4th International Conference on Software Engineering*, September 1979, pp. 11-21.
2. Michael A. Cusumano: Who is liable for bugs and security flaws in software? *Communications of the ACM*, Vol. 47, No. 3 (March 2004), 25-27
3. Michael A. Cusumano et al.: Software development worldwide: The state of the practice. *IEEE Software*, (Nov.-Dec. 2003).
4. Gerard J. Holzmann: The Logic of Bugs. *ACM SIGSOFT Software Engineering Notes*, Vol. 27, No. 6 (November 2002), 81-87

Annotation

Errors and management of software project

Every software development process needs to count with the emergence of errors. The emergence of errors is inevitable in every human activity, therefore it is necessary to deal with this problem and try to remove the errors. In this document we talk about critical places of software, where errors are common. We talk about plugin and script systems and their impact on the control of error behaviour and the security of software. Next we describe options in management to prevent emerging of errors in software projects. In the end we discuss recent progress in responsibility for errors and damage caused by errors in software.

Kvalita, výsledok plánovania a riadenia

ANDREJ FIFLÍK

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
fiflik01@student.fiit.stuba.sk

Abstrakt. V tejto práci sa venujem hlavne riadeniu a plánovaniu kvality v životnom cykle softvéru. Na začiatku sa pokúsim vysvetliť, čo je to vlastne kvalita a aký rôzny význam má pre zákazníka a výrobcu. Ďalej rozoberiem jej význam v postupe času. V riadení sa zameriam na základe predstavy TQA na dve stratégie riadenia, a to SQA pre veľké tímy a Podporný tím pre malé tímy a opíšem ich výhody i nevýhody. V plánovaní sa zase zameriam na dôležitosť plánovania a vyzdvihnem hlavne prvé fázy projektu.

Kvalita a jej miesto v softvérovom projekte

Čo je to vlastne kvalita? Nízka cena? Dizajn? Trvácnosť? Pre každého niečo iné. Čo však môžme povedať je, že kvalita je vlastnosť. Vlastnosť, ktorá svojim spôsobom hodnotí produkt.

Táto vlastnosť však môže byť dosť subjektívna. Ak mi totiž záleží na príťažlivom vzhľade ťažko poviem, že zverák má kvalitný dizajn. Avšak, ak mi záleží na praktickosti produktu, bude pre mňa zverák veľmi kvalitný produkt. Z toho vyplýva, že na kvalitu môžeme hodnotiť rôzne. V čom sa však zhodneme, je to, že kvalitu hodnotíme hlavne podľa toho, na čo je konkrétny produkt určený. Spomínaný zverák nebudeme hodnotiť podľa dizajnu ale skôr podľa praktickosti. No pri kúpe z väčšieho množstva zverákov si ťažko vyberieme ak budú mať všetky rovnakú praktickosť. Tu už zohľadníme i ostatné kvalitatívne aspekty napríklad i spomínaný dizajn. Zákazník teda môže, podľa môjho názoru, kvalitu považovať za určitý súbor vlastností s rôznou prioritou. Pričom dôležitejšie vlastnosti sa odvíjajú hlavne od toho na čo je produkt určený. Priorita zvyšných vlastností závisí už na konkrétnom zákazníkovi. Pre výrobcu však kvalita znamená niečo iné. Výrobca sa v prvom rade snaží predať svoj produkt. Ak bude vyrábať nekvalitné produkty, získa si negatívne meno a nikto si už jeho výrobky nekúpi. No nemôže ani vyrábať titánový zverák, pretože bude síce veľmi kvalitný a trvácny ale jeho cena bude privysoká. Výrobca musí pri svojej výrobe nájsť istú rovnováhu kvality a ceny. Musí si určiť cieľovú

skupinu odberateľov a na jej základe si stanoviť úroveň kvality. Predsa len skôr predá tisíc železných zverákov po päťsto korún ako jeden titánový za stotisíc korún. Odborná definícia vraví, že kvalita je definovaná ako charakteristický znak alebo vlastnosť uspokojujúca potrebu alebo určenie. Je to veľmi zaujímavá definícia. Z pohľadu zákazníka je kvalita určená ako istá vlastnosť uspokojujúca potrebu s cieľom ju získať. Z pohľadu výrobcu, ktorý sa snaží predať svoj výrobok, je táto vlastnosť tiež veľmi dôležitá, no cieľom je ju dosiahnuť. Musí si však uvedomiť do akej miery je schopný danú vlastnosť dosiahnuť, respektíve za akú cenu. Pomôckou pri určovaní akú kvalitu vyrábať môže byť pre výrobcu napríklad to, pre koho má byť daný produkt určený. Je totiž rozdiel či bude vyrábať napríklad kalkulátor pre širokú verejnosť alebo pre NASA. Taktiež môže konzultovať so zákazníkom o jeho predstavách alebo porovnať s už existujúcimi výrobkami. Dôležité je teda uvedomiť si pre koho daný výrobok vyrába a aké sú jeho potreby na kvalitu. A ešte dôležitejšie je uvedomiť si to včas.

V softvérových produktoch sa nároky na kvalitu softvéru oproti minulosti značne zmenili. Na začiatku sa softvér prispôsoboval hardvéru, keďže jeho cena bola ďaleko nižšia ako cena hardvéru. Aj kvalita sa v tomto období zamerala viac na efektívne využitie hardvéru. V súčasnosti však softvér svojou cenou prekonal hardvér, preto sa i kvalita zamerala viac na softvérový produkt ako taký a jeho životný cyklus.

Softvérové produkty sa v súčasnosti zapájajú do ľudského života veľmi vysokou mierou. Zapájajú sa i do takých činností ako je bankovníctvo, jadrová energetika a medicína, kde musia poskytovať maximálnu kvalitu a spoľahlivosť. Kvalita má teda v softvérových produktoch svoje pevné a dôležité miesto.

„Čo si zasadíš to budeš i žať“

V poľnohospodárstve známy výrok. Nedá sa zvýšiť kvalita obilia po žatve. Kvalita produkcie je determinovaná pestovateľom od okamihu kedy vyberie odrodu a produkčný systém. Po zasiatí sa na kvalitu už len dohliada. V softvérových produktoch je to rovnako, úroveň kvality determinujeme už na začiatku procesu a počas celého života produktu ju len udržujeme. Hneď na začiatku projektu si teda treba jasne zvoliť ako kvalitný bude výsledný produkt. Z toho si totiž identifikujeme i najdôležitejšie vlastnosti, ktorými sa budeme riadiť pri vyvíjaní. Tieto vlastnosti nazývame tiež normami kvality projektu. Odborná literatúra odporúča zaviesť si takzvaný Plán akosti. Tento plán stanovuje, ktoré vlastnosti sú najdôležitejšie pre konkrétny produkt. Definuje metriky použité na hodnotenie stupňa dosiahnutej kvality. Jeho časťou je i definícia procesu zabezpečenia kvality [4]. Práve táto časť hovorí o tom ako zabezpečiť aby bol vyrábaný výrobok kvalitný

Riadenie kvality

Ako zabezpečiť kvalitný výrobok? Najznámejšou a najvšeobecnejšou metódou pri výrobe na zaistenie kvality je tzv. Total Quality Management (TQA). TQA je stratégia manažmentu spoločnosti, ktorá kladie dôraz na zapracovanie požiadaviek na kvalitu do všetkých organizačných procesov. V praxi to znamená, že ak budeme dbať na kvalitu na každom stupni výroby, tak i výrobok bude kvalitný. Táto stratégia bola úspešne použitá v priemyselnej výrobe, vzdelaní, vláde a službách [1]. Pre každé odvetvie sa postup ako dosiahnuť požadovanú kvalitu líši.

Zamerajme sa však konkrétne na softvérové inžinierstvo. Softvérová výroba sa značne líši od klasických spôsobov výroby, a to hlavne preto, že softvér je sám o sebe nehmotný a bez iných výrobkov, ako sú počítače alebo dátové médiá, nedokáže vôbec existovať. V praxi existuje viacero možností ako zachovať stanovenú kvalitu produktu počas jeho vývoja. Závisí to od toho ako veľký je výsledný produkt a ako veľký tím na ňom pracuje. Stratégia Software Quality Assurance (SQA) je určená pre veľké projekty a veľké tímy. SQA spočíva vo vytvorení samostatnej skupiny na zabezpečenie kvality. Táto skupina dohliada nad dodržiavaním kvality na všetkých úrovniach v spoločnosti. Efektívna SQA vykonáva tieto štyri činnosti [2]:

- trénovanie a plánovanie
- audity, inšpekcie a posudky
- štandardy a procedúry
- metriky

V rámci trénovania SQA zabezpečuje všeobecné povedomie o kvalite a tiež potrebné vedomosti zamestnancov. SQA musí stáť na začiatku každého projektu a práve ona naplánuje aké činnosti bude treba vykonať aby sa dosiahla požadovaná kvalita a aké postupy budú použité pri vytváraní produktu. SQA tieto postupy zároveň vydáva, okrem nich navyše vytvára interné štandardy kvality. Ich dodržiavanie kontroluje pomocou auditov a inšpekcií aby tak udržala úroveň kvality v spoločnosti. Audity však môže vykonávať i externý audítor. SQA odľahčuje projektových manažérov od dohliadania nad kvalitou, no počas života projektu s nimi intenzívne spolupracuje. Výhodou takejto skupiny je, že fyzicky predstavuje kvalitu. Nie je to len interná norma prijatá v spoločnosti za účelom presadzovania kvality v procese výroby ale fyzicky dohliada na dodržiavanie prijatých ustanovení, dopĺňa a mení ich. Často sa totiž ku kvalite pristupuje dosť neformálne. Väčšina ľudí si myslí, že ak sa budú snažiť, kvalita príde sama. Kvalita však nie je len otázkou prirodzenej inklinácie. Treba k nej pristupovať profesionálne. Vo veľkých spoločnostiach je práve SQA veľkým prínosom.

Naproti tomu v malých spoločnostiach s malými tímami je síce SQA dobrým riešením, avšak úplne stačí vytvorenie podporného tímu. Takýto tím sa osvedčil napríklad v Letnom inštitúte lingvistiky v Dallase [1].

Podporný tím sa skladá z troch manažérov, ktorých hlavná úloha je odstraňovať prekážky realizačnému tímu. Podporný tím sa nesnaží riadiť realizačný tím, to necháva na vedúceho tímu. Radšej sa zameriava na zabezpečenie tímu a vytvára isté rozhranie medzi realizačným tímom a zvyškom organizácie. Realizačný tím vďaka podpornému tímu môže zabudnúť na firemnú politiku. Tento príklad som vybral hlavne preto, lebo práve pracujeme na projekte v malom tíme a práve takýto podporný tím nám vytvárajú naši pedagógovia. Podporný tím stojí na začiatku každého projektu. Dokonca skôr ako sa samotný projekt rozbehne. Vyberá vhodných členov realizačného tímu, identifikuje projektové ciele a pripravuje tzv. programové prehlásenie podniku. Odmlčí sa až po skončení projektu. V našom projekte stál podporný tím na začiatku a vyberal jednotlivých členov do projektu. Jeden podporný tím sa staral o štrnásť projektov. Avšak u nás sa po pridelení členov do jednotlivých projektov podporný tím rozdelil. Na každý projekt ostal jeden člen podporného tímu a tak vytvoril tzv. „jednočlenný podporný tím“. Jeho funkcia však zostala nezmenená. Podporný tím vykonáva hlavne tieto činnosti:

- zabezpečí potrebné zdroje
- monitoruje pokrok v projekte
- dohliada nad kvalitou produktu
- zabezpečuje zodpovednosť voči zvyšku organizácie pre projektový tím

Keďže podporný tím blízko spolupracuje s realizačným tímom, môže tak dohliadať na dodržiavanie kvalitatívnych noriem. Pomáha tak znížiť zaťaženie vedúceho projektu. Práve tu vidím výhodu podporného tímu v malých tímoch. V našom prípade sme teda mali jednočlenný podporný tím. Vytvára totiž most medzi realizačným tímom a zvyškom spoločnosti. Umožňuje mu tak plne sa sústrediť na prácu a pritom ho smerovať podľa firemnej politiky. Pri dozeraní na stav a kvalitu projektu je ďaleko objektívnejší, pretože nie je súčasťou tímu. Pri akýchkoľvek potrebách sa realizačný tím nemusí obracať zakaždým na vedenie firmy ale na svoj podporný tím. Vhodné by bolo, aby členovia podporného tímu boli vo firemnej hierarchii na dosť vysokom poste aby tak mohli pružne reagovať na tieto potreby.

„Kvalita sa neprihodí, kvalitu treba naplánovať“

Plánuj projekt! Naplánovanie projektu skôr ako doňho skočíme robí prácu ďaleko efektívnejšou a radostnejšou. Toto pravidlo by mal mať na pamäti projektový manažér. Nech už ide o akýkoľvek projekt, veľký či malý, plánovaním sa projekt značne sprehľadňuje a umožňuje kontrolu kvality v ktorejkoľvek etape jeho vývoja. Naplánovať celý projekt naraz je však veľmi náročné, takže si treba rozdeliť prácu spôsobom zhora nadol, najprv určiť míľniky, a potom konkretizovať jednotlivé obdobia medzi míľnikmi. No nie vždy je možné hneď na začiatku konkretizovať všetky činnosti medzi míľnikmi, je

však nutné, aby práve prebiehajúca činnosť bola definovaná presne po najbližší míľnik. Ak sa má nejaká činnosť vykonávať veľmi dlho, je nutné ju rozdeliť na také časti, aby jedna časť trvala maximálne pol roka a mala i svoj výstup. Jednotlivé činnosti predstavujú tzv. moduly. Každý modul musí mať svoj výstup aby bolo možné overiť jeho kvalitu. Modul by však nemal zaberať viac ako dva týždne práce pre jedného človeka. Každý modul musí mať definovaný termín začatia i ukončenia. Práca na moduloch prináša ďaleko častejšie ovocie ako práca na veľkých projektoch, ktorých výsledky sa objavia až po dlhom čase. Takto sa i veľmi ľahko motivuje každý člen tímu (zlepšujeme morálku a chuť členov tímu ďalej pracovať) [1].

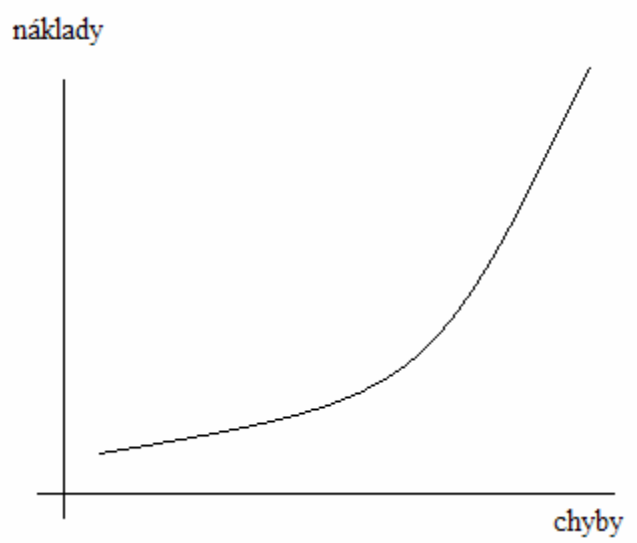
„Opíš čo robíš a rob to tak ako si to opisuješ“

Dôležitá vec pri plánovaní je spraviť si pri práci poriadok. Vytvorením presných postupov pre jednotlivé činnosti sa celý proces vývoja softvéru sprehľadní. Navyše takto umožníme optimalizovanie činností, lepšiu kontrol kvality procesu i jednoduchšie plánovanie. Áno, môže sa zdať zbytočné opisovať každú činnosť, ale treba si uvedomiť, že práve tak sa predchádza zbytočným a opakovaným chybám. Preto vytvárať postupy do každého stupňa výroby je dôležité. Pripomeňme si teraz známu stratégiu TQA, ktorá vraví práve o tom, že ak zavedieme kvalitu do každého stupňa výroby, tak i výrobok bude kvalitný. Softvér má však trochu inú stratégiu výroby ako klasické výrobky. Tie sa po vyrobení považujú za hotové a ich ďalší život je už mimo výrobcu. Softvér po vyrobení ďalej udržiava kontakt s výrobcom a jeho život pokračuje použitím priamo v prevádzke. Zaniká až jeho vyradením. Výrobca počas prevádzky poskytuje aspoň minimálne záruky v prípade poruchy. Navyše je pre softvér špecifická možnosť zmeny alebo doplnenia funkcionality, aj po ukončení výroby. Aby sme teda zachovali stanovenú kvalitu podľa TQA, zavedieme jej princípy do každej úrovne vývoja softvéru. Pre každý projekt bude kvalita v manažmente spoločnosti rovnaká. V životnom cykle bude však závislá od konkrétneho projektu. Životný cyklus softvérového produktu môžeme rozdeliť podľa použitého vývojového modelu. Aby som bol konkrétnejší, predstavme si softvér vyvíjaný na vodopádovom modeli. Životný cyklus takéhoto softvéru môžeme rozdeliť do týchto fáz:

1. Štúdium vhodnosti
2. Analýza požiadaviek
3. Návrh produktu
4. Detailný návrh
5. Implementácia a testovanie modulov
6. Systémová integrácia
7. Údržba a podpora

Jednotlivé fázy, ak chceme kontrolovať kvalitu, musia mať i svoj náležitý výstup. Na začiatku si naplánujeme celý priebeh projektu a stanovíme si kvalitatívne kritériá.

Dôležitou súčasťou plánovania nie je len naplánovanie jednotlivých etáp a činností v projekte, ale i rozdelenie zdrojov práve pre jednotlivé etapy projektu. A tu si treba uvedomiť dôležitosť, ktorú som tak kládol na stanovenie si úrovne kvality. Ak je potreba vytvoriť produkt v čo najkratšom čase bez ohľadu na kvalitu, musia byť zdroje pridelené hlavne tej etape, ktorá trvá najdlhšie. Uvedomme si ale jeden fakt. Ak sa vyskytne chyba pri takto rýchlo tvorenom softvéri, je ťažko ju vôbec zaregistrovať. A chyba ostane nepovšimnutá, kým nespôsobí iný problém alebo škodu. Preto si treba dobre zvážiť do ktorých etáp a ako veľa zdrojov pridelíme. Závisí to od stanovenej úrovne kvality.

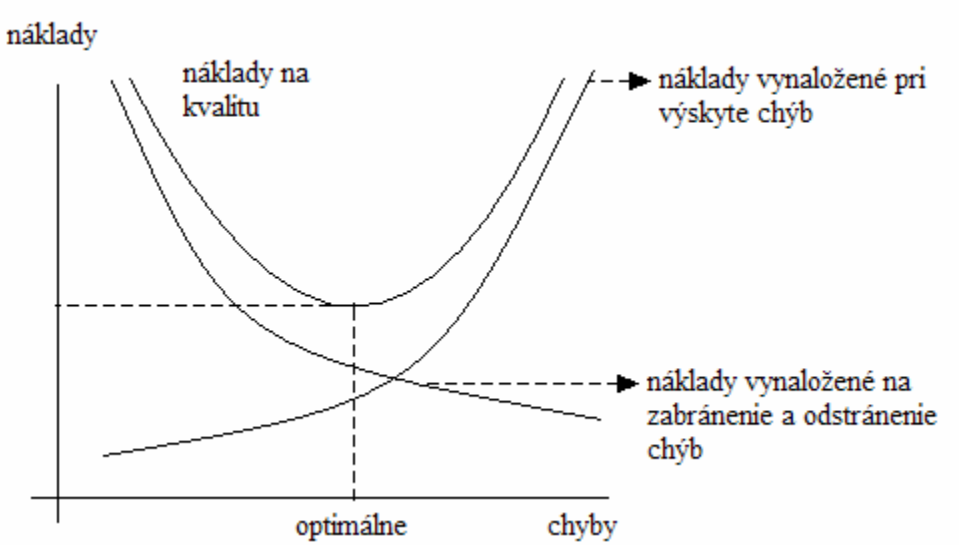


Obr. 3. Závislosť nákladov vynaložených na opravu chyby počas života softvéru

„Kvalita je meraná zlyhaniami. Zlyhaniami v ktorejkoľvek fáze života projektu“

Ale ktoré etapy sú pre kvalitu tie najdôležitejšie? Pomôžem si obrázkom (Obr. 3). Na obrázku je znázornená závislosť nákladov vynaložených pri výskyte chýb počas života softvéru. Ako vidno, chyby objavené na začiatku projektu si vyžadujú ďaleko nižšie náklady, ako chyby z neskorších etáp. Predstavme si teraz dva extrémne prípady. Chyba objavená v analýze požiadaviek sa veľmi jednoducho opravuje. V prípade nejasnosti sa konkrétna požiadavka dodefinuje a zahrnie do dokumentácie. Zvyčajne si vyžaduje len minimálne náklady. Na druhej strane, ak by bola chyba objavená až pri odovzdávaní produktu zákazníkovi, môže to mať ďaleko väčšie následky. Tu už totiž nie je možné jednoduché prepísanie dokumentácie. Zvyčajne si takáto chyba vynúti prepisovanie zdrojového kódu, ak nie prepracovanie celého projektu odznova. Preto dôležitosť prvých fáz života softvéru netreba podceňovať. Väčšina zdrojov by mala byť pridelená práve do

fáz analýzy a návrhu, aby sa tak odhalilo čo najväčšie množstvo chýb [3]. Avšak nemôžme prideliť všetko do zabezpečenia kvality. Treba nájsť také optimálne riešenie, aby bola zachovaná stanovená kvalita ale nemrhalo sa zbytočne zdrojmi iba na kvalitu. Takéto optimálne riešenie nám predstavuje nasledujúci obrázok (Obr. 4). Optimálne riešenie predstavuje minimálny súčet nákladov vynaložených na zabránenie chybám a odstránenie pri ich výskyte. Práve tieto náklady by mali byť investované na udržanie kvality.



Obr. 4. Závislosť vynaložených nákladov na zabránenie a odstránenie chýb

V ďalších fázach života softvéru viac-menej nad kvalitou dohliadame. Vo fázach implementácie a systémovej integrácie je to dosť jednoduché, stačí ak samotné testovanie bude úspešné. Najdlhšou fázou je údržba a podpora. Softvér je dokončený a zavedený do prevádzky. Počas tejto fázy sa dohliada na jeho správne fungovanie a v prípade ak má zákazník záujem môžu sa urobiť aj dodatočné zmeny v softvéri. Treba si však uvedomiť, ako som už spomínal, že zmeny vykonané po dokončení softvéru bývajú dosť nákladné. Ak sa však v tejto fáze objaví chyba, je nutné ju opraviť minimálne tak, aby nespôsobovala škody. Ak sa dá, je najlepšie ju úplne odstrániť. Ako som už spomínal, v tejto fáze je však oprava chýb značne nákladná, preto sa treba dohodnúť so zákazníkom na ďalšom postupe.

Záver

Kvalita predstavuje dôležitý článok v každom úspešne vyvíjanom produkte. Súčasný zákazník vyžaduje od softvéru čoraz väčšiu kvalitu. A ak výrobca nechce stratiť kontakt so zákazníkom, musí sa mu prispôsobovať. Bez plánovania a riadenia kvality to však nejde. Plánovanie a riadenie nám umožní nielen udržať si počiatočne stanovenú úroveň kvality, ale poskytuje nám dobré informácie pre prácu na ďalších projektoch.

Metódy riadenia SQA a Podporný tím predstavujú možné riešenia riadenia kvality v praxi. Zatiaľ čo SQA je priamo určená na sledovanie kvality a sleduje ju na každej úrovni vo firme, Podporný tím je viac-menej istým organizačným článkom medzi vedením a realizačným tímom. Dohliada, mimo iného, aj na dodržiavanie kvalitatívnych noriem firmy. Je však lepším riešením pre menšie tímy.

Plánovanie kvality je vynikajúcim nástrojom ako si udržať v projekte prehľad o jej stave. Plánovanie sa skladá aj z plánovania zdrojov vynaložených na jej udržanie. Plánovanie netreba podceňovať hlavne v prvých fázach projektu, aby sa predišlo neskorším problémom.

Kvalita je súčasťou života každého softvérového produktu a je len taká veľká, ako veľká pozornosť sa jej venuje.

Použitá literatúra

1. Rettig, M., Simons, G.: A project planning and development process for small teams. *Communications of the ACM*, Vol.36, No.10 (1993).
2. Wheeler, S., Duggins, S.: Improving software quality. *Proceedings of the 36th annual Southeast regional conference table of contents.* ACM Press, New York (1998), 300 – 309.
3. Krishnan, M.S.: Cost, Quality and User Satisfaction of Software Products: An Empirical Analysis. *Proceedings of the 1993 conference of the Centre for Advanced Studies on Collaborative research: software engineering - Volume 1*. IBM Press, Toronto (1993), 400 – 411.
4. Bieliková, M.: Manažment v softvérovom inžinierstve. Bratislava, 1999.

Annotation

Quality, result of planning and management

In this paper I describe quality management and planning in software life cycle. Firstly, I will try to explain basic quality concepts and then explain how quality is related to customer and producer. I will describe importance of relation between quality and time passing. In the management part I will focus on basic ideas of TQA, specifically two strategies of management, SQA for teams with many

members and Support team for smaller teams. I will describe their advantages and disadvantages. In the planning I will focus on importance of planning and underline mostly first phases in project lifecycle.

Manažment konfliktov v softwarovom tíme

DUŠAN VELICKÝ

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

Abstrakt. Hlavnou výhodou tímu oproti jednotlivcovi je rôznorodosť zdrojov, poznatkov a myšlienok. Nevýhodou tejto rôznorodosti je pravidelný výskyt konfliktov medzi členmi tímu. Aby sa zamedzilo eskalácii konfliktu, je potrebné riešiť hádky rýchlo a otvorene. Štatistiky hovoria, že väčšina manažérov si uvedomuje existenciu nezhôd v tíme a zároveň absolvovali tréning v oblasti riešenia medziľudských vzťahov no iba zriedkakedy dávajú vyššiu prioritu riešeniu týchto konfliktov. Preto je potrebné, aby jednotliví členovia tímu ovládali techniky pre riešenie konfliktov a priamo ich aplikovali medzi sebou. V tomto článku sú objasnené príčiny, priebeh a niektoré zo spôsobov riešenia konfliktov v tímoch.

Úvod

Konflikty vznikajú z rozdielov. Keď jednotlivci prichádzajú pracovať do tímov, ich rozdiely ako sila, hodnoty, postoje a sociálne faktory prispievajú ku vzniku konfliktov v tíme.

Keďže čím ďalej tým viac spoločností delí jednotlivé projekty medzi tímy vývojárov, narastá potreba oboznamovať jednotlivých členov s metódami riešenia konfliktov. Konflikt v tíme nie je vždy deštruktívny, v prípade že je úspešne zvládnutý, je prínosom pre celý tím.

Štatistiky hovoria, že väčšina manažérov si uvedomuje existenciu nezhôd v tíme a zároveň absolvovali tréning v oblasti riešenia medziľudských vzťahov, no iba zriedkakedy dávajú vyššiu prioritu riešeniu týchto konfliktov. Preto je potrebné, aby jednotliví členovia tímu ovládali techniky pre riešenie konfliktov a priamo ich aplikovali medzi sebou.

Príčiny konfliktov

Často je ťažké odhaliť príčiny konfliktov, no podľa Varneyho [3] ich možno deliť do troch kategórií:

- Komunikačné
- Štrukturálne
- Personálne

Konflikty ktorých príčinou je komunikácia, sú najzávažnejšie a zároveň najfrekventovanejšie. Pod zlou komunikáciou je možné chápať nedostatočné počúvanie toho druhého, nedostatočné zdieľanie informácií, rozdiely medzi interpretáciou a vnímaním informácie a ignorovanie neverbálnej komunikácie.

Štrukturálne dôvody konfliktov tkvejú zväčša vo veľkosti organizácie alebo tímu, dĺžke obratu informácií, zainteresovanosti jednotlivých členov organizácie, systému finančného ohodnocovania a v množstve závislostí medzi členmi tímu.

Personálne dôvody zahŕňajú veci ako sebaúctu, osobné ciele jednotlivcov, ich hodnoty a potreby. K úspešnému zdolaniu konfliktu je potrebné, aby si manažéri a členovia tímu uvedomovali nepredvídateľnosť týchto potrieb, ich dopad na jednotlivcov a na tím ako taký.

Typy konfliktov

Konflikt v tíme nemusí byť vždy deštruktívny, ba dokonca môže viesť k novým nápadom, postupom v rámci organizačných postupov a zvýšeniu zainteresovanosti jednotlivcov k riešeniu problémov. Konflikt sa stane negatívnym, keď je ponechaný k eskalácii, k bodu, keď sa ľudia cítia porazenými a je vytvorená atmosféra nedôvery, v ktorej sa účastníci konfliktu len podozrievajú[1].

Nelson [2] varuje, že negatívne konflikty veľmi rýchlo vedia zničiť tím a ich príčinou je zväčša zlé plánovanie. Ponúka taktiež zoznam vysoko pravdepodobných oblastí z ktorých môžu vzniknúť negatívne konflikty.

- Administratívne procedúry
  V prípade, že tím nemá základy pre to čo robí, členovia tímu nie sú schopní koordinovať svoju prácu.
- Ľudské zdroje
  Nedostatok ľudských zdrojov má za následok, že niektorí členovia tímu budú znášať príliš vysokú záťaž, preto by mal manažment prispôsobovať plány projektu pri každej zmene požiadaviek tak, aby bolo možné pri „normálnej“ záťaži dokončiť projekt v požadovanom termíne.

- Prekročenie rozpočtu
  Veľakrát nevyhnutné, stáva sa problémom práve vtedy, keď je pri plánovaní použitá zlá mierka alebo granularita. Je potrebné, aby si tím uvedomoval, že v prípade vzniku tohto problému sa v najhoršom prípade bude musieť chýbajúci čas nájsť v rámci tímu alebo spoločnosti. Každá vyspelejšia spoločnosť má motivačné prostriedky, ktoré riešia práve tento problém.
- Plánovanie
  Všetky informácie ohľadne plánovania by mali byť dostatočne viditeľné pre každého zamestnanca tímu.
- Zodpovednosť
  Každý zamestnanec by mal mať prehľad o tom, kto je začo zodpovedný. Jednotlivé zodpovednosti za dôležité celky by mali byť rozdelené na členov tímu.

Riešenie negatívnych konfliktov

V prípade výskytu negatívnych konfliktov je možné využiť jeden z päť všeobecne akceptovaných postupov: priamy postup, vyjednávanie, dozor, ústup a deemfáza.

- Priamy postup
  Toto je možno najlepší postup zo všetkých. Riešenie problému je sústredené na vedúceho, ktorý vykonzultuje problém spolu so zainteresovanými. Je potrebné, aby zo strany vedúceho bola zachovaná objektivita, v prípade, že je použitá kritika, musí byť cielená a konštruktívna.
- Vyjednávanie
  Toto je vynikajúca technika v prípade, že obe strany majú nápady, ale nevedia nájsť spoločnú reč. Zväčša je potrebná tretia osoba, vedúci tímu, ktorý pomôže nájsť kompromis. Kompromis je o ubratí na oboch stranách, preto sa táto metóda končí zväčša kompromisom a odchodom oboch stále nespokojných strán.
- Dozor / Dodržiavanie pravidiel v tíme
  Je dôležité, aby bola táto metóda použitá iba vtedy, keď je jasné, že pracovník nechce byť členom tímu, alebo úplne odmieta pracovať s ostatnými členmi tímu. Možno je v tomto prípade lepšie pre obe strany, keď si pracovník nájde iný, nový tím.
- Ústup
  Metóda sa používa vtedy, keď konflikt nie je vlastne konfliktom. Jednoduchým obchádzaním alebo zamedzením vzniku problému vie vedúci tímu dostatočne zdržiavať, aby sa obe strany „schladili“. V prípade, že je táto technika použitá vo vhodnom prostredí a skúseným vedúcim, je možné pomôcť predchádzať malým

incidentom, ktoré normálne vedú niekoho k prežitiu „zlého dňa“ namiesto toho, aby spôsobili problémy, ku ktorým nikdy nemalo prísť.

- Deemfáza
  Táto metóda je formou vyjednávania, pri ktorej sa dôraz kladie na miesta súhlasu oboch strán. Zväčša sa stáva, že keď strany pochopia, že sú navzájom v zhode, otvorí sa nový, konštruktívny smer diskusie.

Typické príklady každodenných konfliktov v soft. tíme

Táto kapitola ukazuje typické príklady konfliktov medzi jednotlivými členmi soft. tímu. Keďže posledných pár rokov pôsobím v soft. tíme, ktorého hlavnou náplňou je návrh, implementácia a údržba webových IS, sústredím sa v tejto práci hlavne na takýto typ soft. tímov. Avšak väčšina týchto problémov postihuje aj tímy, ktoré majú úplne odlišné zameranie.

Nekonečný boj: programátor VS tester

Medzi najčastejšie dôvody konfliktov medzi programátormi a testermi patria hlavne:

- Nedostatočná analýza
  Projekt je akceptovaný bez uváženia o jeho zameraní a doménovej oblasti. Vypracované dokumenty analýzy dostatočne nepokrývajú celé správanie systému.
- Nedostatok znalostí z doménovej oblasti
  Programátori začnú programovať a testeri začnú testovať bez toho, aby mali podrobné znalosti z doménovej oblasti.
- Zlá organizácia práce
  Časové hranice sú príliš tesné, programátori pracujú deň-noc, aby dodržali termíny. Zdrojový kód sa dostáva k testerom bez dostatočného testovania na úrovni programátorov.
- Chybný testovací nástroj
  Nič nie je horšie ako defektný testovací nástroj alebo zlé nastavenie parametrov tohto nástroja, nakoniec na to doplatia ako programátori, tak i testeri, pričom vzniká riziko prešvihnutia termínu alebo rozpočtu.

Riešením týchto problémov je pravidelná osveta v doménovej oblasti v rámci stretnutí a dobré plánovanie v rámci manažmentu.

Programátor VS dizajnér VS zákazník

Zákazník má záujem o produkt, prebehne analýza a implementácia samotnej aplikácie + dizajnu. Pri priebežných odovzdávaniach je alebo dizajnér alebo zákazník príliš kreatívny

až tak, že čiastočne mení v malých množstvách už implementované časti produktu. Zväčša sa jedná o prezentačnú vrstvu. V prípade, že je prezentačná vrstva príliš spojená s aplikačnou, sú tieto zmeny kritické a určite vedú k negatívnym konfliktom, a k omeškaniu alebo prekročeniu rozpočtu projektu.

Táto skutočnosť je zväčša problémom manažérskym, pretože programátor a aj dizajnér sú závislí na kompromise/dohode medzi zákazníkom a vedením. V prípade, že sú dohody medzi vedením a zákazníkom zle postavené, alebo je zvolená zlá komunikácia so zákazníkom, doplatí na to ako projekt, tak aj tím, ktorý na ňom pracuje.

Riešením je vhodne zvolená komunikácia so zákazníkom, vhodné poradie pri zisťovaní informácií od zákazníka, podrobné a hlavne reálne plánovanie prác na projekte.

Programátor VS analytik

Niekedy sa stane, že analytik dodá materiál, ktorý dostatočne detailne neopisuje celý systém, resp. popisuje časti systému tak, ako to nikto zo strany tímu nepotrebuje.

Špeciálnym prípadom je, keď analytika zastupuje človek, ktorý pozná len doménovú oblasť, no nepozná vôbec hranice technológie, ktorou bude projekt, IS realizovaný. Táto situácia môže nastať vtedy, keď je nemožné dopraviť ku klientovi analytika/programátora (časová tieseň, nedostupnosť klienta, finančná náročnosť).

Rovnako programátora poteší aj fakt, keď analytik nevie pri zbieraní požiadaviek klienta usmerniť tak, aby sa dali použiť v rámci projektu už vyvinuté komponenty, resp. skomplikuje implementáciu tým, že navrhne univerzálne riešenie, ktoré klient nepotrebuje a nikdy potrebovať nebude.

Na obranu analytikov treba povedať, že na vzdory programátorov je skoro nemožné spraviť model, s ktorým bude programátor úplne spokojný. Vždy sa tam nájdu nezrovnalosti, na ktoré môže prísť len programátor, pretože vidí do konkrétnej časti najhlbšie zo všetkých zainteresovaných.

Riešením týchto problémov je analytik, ktorý bol v minulosti na pozícii programátora a súčasne detailne pozná hranice a možnosti technológie, v ktorej bude projekt implementovaný. V prípade, že predošlý prípad nie je možné dosiahnuť, riešením je prítomnosť najskúsenejšieho programátora pri analýze a pri „kritických“ stretnutiach s klientom.

Manažment/projektový manažér VS zvyšok tímu

Je všeobecne známe, že v rámci trhu IT nie je ľahké uspieť v tvrdej konkurencii ostatných firiem. Dôsledkom toho je, že kontrakty na projekty sa robia s minimálnou časovou rezervou, nie je do nich premietnutý čas na testovanie a iné potrebné činnosti, ktoré sú nevyhnutné pre úspešné zvládnutie projektu. Dôvodom je cena výsledného projektu a to najpodstatnejšie: „deadline“, teda dátum odovzdania produktu do užívania koncovému zákazníkovi.

Taktiež je všeobecne známe, že softwarové firmy v jednom časovom intervale participujú na viacerých projektoch, ktoré sú rozdistribuované medzi tímy spoločnosti. Nie je vylúčené a je skôr pravidlom, že niektorí členovia tímu sa podieľajú na viacerých projektoch.

V prípade, že dôjde k časovej tiesni na jednotlivých projektoch, je isté, že programátor bude znášať vysokú záťaž, alebo je veľmi pravdepodobné, že sa jeho časť práce oneskorí, pretože sa bude musieť na poslednú chvíľu rozdeliť medzi iných členov tímu.

Vedenie projektov zväčša odpovedá na otázky typu „Kedy to má byť hotové?“ zázračným slovom ASAP (as soon as possible, v preklade čo možno najskôr) a priority pozná až v hraničných situáciách.

Záver

Momentálny stav trhu trpí nedostatkom odborníkov v oblasti informačných technológií. Každá zo softwarových spoločností je hlavne o ľuďoch, preto si vedenie firiem musí uvedomovať potreby svojich zamestnancov a venovať konfliktom na pracovisku dostatočnú pozornosť. V prípade, že tak nespraví, je vysoko pravdepodobné, že nespokojných členov tímu stratí.

Použitá literatúra

1. Bowditch, J. L., Buono, A. F.: A primer on organizational behavior (4th ed.). New York, NY: JohnWiley & Sons., 1997.
2. Nelson, M. Interpersonal team leadership skills. Hospital Material Management Quarterly, 16 (4), 53 - 63., 1995.
3. Varney, G. H.: Building productive teams: An action guide and resource book. San Francisco, CA: Josey-Bass, Inc., 1989.

Annotation

Conflict management in software teams

A major advantage a team has over an individual is its diversity of resources, knowledge, and ideas. However, diversity also produces conflict. Conflict arises from differences, and when individuals come together in teams, their differences in terms of power, values, and attitudes contribute to the creation of conflict. To avoid the negative consequences that can result from disagreements, most methods of resolving conflict stress the importance of dealing with disputes quickly and openly. Statistics proved that although most managers are aware of disagreements and have received

training in conflict resolution, they seldom assign a high priority to solving conflict problems. With this in mind, it is critical that team members possess skills to resolve conflict among themselves.

Manažment komunikácie pre lokalizované a distribuované softvérové tímy

MARTIN DARULA

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

Abstrakt.

Komunikácia vnútri lokalizovaných, ako aj distribuovaných tímov je veľmi dôležitou súčasťou ich manažmentu a jedným z kľúčových faktorov ich úspechu pri tvorbe softvérových produktov. Tento dokument sa zaoberá časťou plánovania komunikácie a distribúcie informácií, pričom hlavný dôraz kladie na definovanie vhodných spôsobov komunikácie pre jednotlivé úlohy, činnosti a role v tímoch. Dokument približuje teóriu mediálnej bohatosti a jej praktické testy, ktoré boli v posledných rokoch uskutočnené a ktoré potvrdili jej správnosť. Tiež ukazuje, že distribuované tímy sú schopné vykonávať zadané úlohy s aspoň takým dobrým výsledkom ako tímy lokalizované, a teda že výhoda osobnej komunikácie sa s postupným vytváraním a zlepšovaním jednotlivých foriem komunikácie a hlavne s ich úspešným a efektívnym ovládnutím členmi tímu, stiera.

Úvod

Dávno sú preč časy, keď na vytvorenie profesionálneho softvérového systému stačil jeden človek. Samozrejme, aj dnes sa nájdu projekty, na ktoré nie je potrebný veľký počet ľudí, avšak tieto sú vždy malého rozsahu, určené pre veľmi špecifickú úlohu, často z vedeckej oblasti iba na spracovanie výsledkov, prípadne na vytvorenie malých pomocných nástrojov alebo malého softvéru tvoreného nadšencom v jeho voľnom čase, pri nedefinovanom čase dokončenia. Vo veľkej väčšine sa jedná o softvér poloprofesionálny alebo amatérsky. Naproti tomu sa oveľa väčšia časť nielen softvérových produktov vytvára v tímoch, čiže za aktívnej účasti od niekoľkých, cez desiatky až po stovky ľudí, ktorí sa môžu nachádzať na jednom mieste, v relatívne blízkej vzdialenosti jeden od druhého, ale v poslednom čase sa v stále väčšej miere vytvárajú tímy distribuované, kde členovia nielen že nie sú sústredení v jednej budove, ale nemusia sa nachádzať ani v tom istom štáte. V oboch prípadoch – v lokalizovaných, ako i distribuovaných tímoch – je jedným z kritických bodov úspešnosti projektu vhodné zvládnutie komunikácie [4]. Podľa [1] sa proces manažmentu

komunikácie rozdeľuje na 4 hlavné časti: plánovanie komunikácie, distribúcia informácií, referovanie o výkone a administratívne uzavretie. Tento dokument sa bude venovať predovšetkým častiam plánovania komunikácie a distribúcie informácií a zameria sa hlavne na priblíženie komunikačných módov čitateľovi, ich prepojenie s definovanými rolami v tíme a informovať o vhodných módoch pre jednotlivé roly.

Teória mediálnej bohatosti

Teória mediálnej bohatosti [6] (media richness theory) je jednou z teórií venujúcich sa problémom použitia médií pre komunikáciu medzi ľuďmi. Podľa [2],[3] je jej základom myšlienka potreby použitia rôznych druhov médií pre rôzne druhy úloh a účely komunikácie kvôli odlišným charakteristikám jednotlivých médií. Medzi hlavné charakteristiky, podľa ktorých sa médiá rozdeľujú, patria:

1. Schopnosť spätnej väzby – určuje ako rýchlo môžu komunikujúce strany žiadať o vysvetlenie a ako rýchlo ho dostanú. Podľa tohto kritéria sa delia médiá na synchrónne a asynchrónne. Medzi synchrónne patria napr. osobný kontakt, telekonferencia alebo online chat, zatiaľ čo medzi asynchrónne patria napr. e-mail, internetové fóra alebo web.
2. Dostupnosť viacerých kanálov – týka sa počtu rôznych dostupných komunikačných kanálov, ktoré môžu komunikujúce strany využiť pri komunikácii pomocou konkrétneho média. Ide napr. o fyzickú prítomnosť, výrazy tváre, tón hlasu, gestá, slová, čísla a grafiku. Medzi médiá s viacerými komunikačnými kanálmi patria napr. osobný kontakt, web, konferenčný softvér, naproti tomu typickým médiom disponujúcim jediným komunikačným kanálom je textový chat.
3. Rozmanitosť jazyka – hovorí o tom, aké rôzne formy jazyka médium podporuje. Napr. čísla podporujú presnosť vyjadrovania, zatiaľ čo prirodzený jazyk podporuje lepšie pochopenie širšieho rozsahu predstáv a myšlienok. Médiá, ktoré využívajú písanú formu jazyka poskytujú väčšiu presnosť vyjadrovania, zatiaľ čo hlasové médiá podporujú prirodzený jazyk a prirodzené vyjadrovanie spojené s jednoduchším pochopením predmetu komunikácie.
4. Zameranie sa na človeka – týka sa stupňa pozornosti a zaujatosti komunikujúcej osoby, ktorý médium umožňuje dosiahnuť, a tiež emocionálneho obsahu, ktorý správy obsahujú.

Celkovo je možné medzi komunikačné módy zahrnúť osobné stretnutia, e-mail, telefón, pero a papier, tabuľu, chat, počítačové fóra, počítačovú konferenciu, telefónnu konferenciu, fax, poštu a podobne.

Teória okrem definovania vlastností pre médiá tiež určuje typy situácií, ktoré pri komunikácii nastávajú a stupeň zvládnutia ktorých odráža tiež vhodnosť použitia média pre daný typ komunikácie:

1. Nejasné situácie – ide o situácie, pri ktorých dochádza k výmene nejednoznačných a viacvýznamových informácií, vďaka ktorým môže dôjsť k zmäteniu subjektu alebo k potrebe bližšieho a dodatočného vysvetlenia predchádzajúcej komunikácie. Môže ísť tiež o výmenu viacerých subjektívnych a rozporných pohľadov na diskutovanú problematiku. Jasné situácie sú naproti tomu také, kde dochádza k výmene presných, jasných a objektívnych informácií podporených všeobecnými referenciami. Podľa teórie mediálnej bohatosti sú pre nejasné situácie vhodné bohaté médiá oplývajúce viacerými komunikačnými kanálmi a schopnosťou rýchlej spätnej väzby.
2. Nerozhodné situácie – sú situácie, kde nedostatok informácií spôsobuje problém správne sa rozhodnúť, z čoho vyplýva potreba získania dodatočných informácií. Naproti tomu rozhodné situácie oplývajú dostatkom informácií a môžu vyriešiť nerozhodnosť. Podľa spomínanej teórie pomalé, asynchrónne médiá s malým počtom komunikačných kanálov sú vhodné pre obe, nerozhodné aj rozhodné, situácie.
3. Vymieňané správy môžu mať osobnú alebo neosobnú povahu. Podľa teórie mediálnej bohatosti sú pre osobnú komunikáciu vhodnejšie bohaté médiá s viacerými kanálmi a krátkou odozvou.

V reálnych situáciách sa však človek môže nachádzať v jednom momente vo viacerých situáciách, a tak rozhodnutie o vhodnosti konkrétneho média pre daný moment nie je úplne triviálne. Podľa [2] je možné definovať 3 všeobecné kategórie komunikácie v tímoch, ktoré zodpovedajú teórii mediálnej bohatosti, a všeobecne zahŕňajú viaceré situácie, ktoré pri komunikácii môžu nastať:

1. Osobná komunikácia – vytvára sociálne vzťahy medzi členmi tímu a udržuje tím súdržný. Tiež pomáha zvýšiť morálku v tíme, produktivitu a spokojnosť s prácou [5]. Pod osobnú komunikáciu spadajú napr. pozdravy, neformálne rozhovory alebo vtipy. Komunikácia zahŕňa emocionálny obsah a podľa teórie sú pre ňu vhodné synchrónne médiá s väčším počtom komunikačných kanálov.
2. Tímový manažment – oproti osobnej komunikácii udržiava tím súdržný po organizačnej stránke. Medzi komunikáciu tímového manažmentu patria napr. rozhovory o časoch a miestach stretnutí, rozdeľovanie úloh a potrebného materiálu. Kategória vyžaduje prechod od nerozhodných k rozhodným situáciám a preto sa pre ňu podľa teórie hodia médiá asynchrónne, s menším počtom kanálov.
3. Komunikácia počas práce na úlohe – týka sa priamo pracovnej činnosti riešenia problému, na ktorom tím pracuje. Patrí sem napr. tzv. brainstorming, prezentácia výsledkov, tvorba nových nápadov a rozhodovanie o nich. Tieto činnosti

vytvárajú väčšie množstvo protichodných a nejasných názorov, ktoré je potrebné konzultovať a rozhodovať o správnych riešeniach, preto táto kategória vyžaduje bohaté médiá, s viacerými komunikačnými kanálmi a rýchlou odozvou.

Ďalším významným faktorom pri porovnávaní vhodnosti jednotlivých komunikačných riešení v tímoch je aj konkrétna rola člena tímu, ktorá určuje jeho potreby a spôsob práce. Rôzne sú aj spôsoby komunikácie medzi jednotlivými rolami navzájom. Podľa [1] a [3] je možné definovať nasledovné role, ktoré v tíme vznikajú:

1. Iniciátor – iniciuje nové myšlienky, zásady riešenia, zmeny
2. Vyšetrovateľ – prináša do tímu informácie a myšlienky z oblastí mimo tímu, vytvára vonkajšie kontakty tímu
3. Koordinátor – líder tímu, vyhodnocuje kvality jednotlivých členov tímu, definuje a koordinuje činnosti, objasňuje vzťahy a smeruje tím k úspechu
4. Ťahúň – oplýva energiou implementovať nové myšlienky a rozhýbať projekt, dokáže vybudiť aktivitu a zameriava pozornosť na ciele
5. Navigátor – kriticky posudzuje návrhy, zabraňuje tvorbe chýb a monitoruje postup tímu, či ide tím správnym smerom
6. Realizátor – poskytuje neformálnu komunikačnú sieť a podporu, ktorá pokračuje aj po skončení stretnutí, efektívne a systematicky vykonáva dohodnuté úlohy
7. Organizátor – vytvára z plánov manažovateľné úlohy
8. Ukončovač – dohliada na včasné ukončenie projektu a kompletizáciu úlohy

Podobne ako pri kategóriách komunikácie je možné podľa teórie vytvoriť mapovanie medzi jednotlivými rolami a požiadavkami na médium, vhodné pre jednotlivé role.

Realita

Na potvrdenie názorov poskytnutých teóriou mediálnej bohatosti bolo vypracovaných viacero štúdií. V [2] bola skúmaná komunikácia medzi distribuovanými tímami študentov. Závery štúdie potvrdili pravdivosť teórie, pričom pre osobnú komunikáciu si študenti vybrali ako primárne médium chat, pre tímový manažment e-mail a pre komunikáciu počas prác prevládal chat a zdieľané aplikačné prostredie. V štúdii sa konštatuje, že je výhodné sprístupniť viacero typov médií pre komunikáciu v tímoch a tímy budú využívať rôzne médiá pre rôzne typy komunikácie podľa toho, čo im lepšie vyhovuje, pričom sa ukázal súlad s teóriou. Vyplynulo tiež, že používatelia majú tendenciu viac využívať bohatšie médiá oproti chudobnejším.

E-mail sa ukázal ako výhodný pre jeho asynchrónnu povahu a jednoduchosť použitia, čo umožňuje voľne a presne formulovať myšlienky, roztiahnuť konverzáciu v čase a obísť potrebu dohadovania času pre komunikáciu. Podstata e-mailu tiež umožňuje stručné a presné vyjadrovanie myšlienok.

Zdieľané aplikačné prostredie umožňuje viackanálovú komunikáciu medzi subjektami a hlavnou výhodou je okamžitá spätná väzba, ktorá umožňuje spresňovať a vysvetľovať. Nevýhodou je vyššia zložitosť použitia a potreba synchronizácie členov tímu.

Mapovanie rolí na požiadavky na médiá a konkrétne módy komunikácie uskutočnila napr. štúdia [3]. Z nej vyplýva, že osobné hodnotenie niektorého z komunikačných módov závisí od role danej osoby. Ako najhodnotnejšie médium je všeobecne považovaný osobný kontakt, teda lokalizované tímy majú výhodu oproti distribuovaným, inak je vhodné uskutočňovať pravidelné osobné stretnutia členov tímu, čo potvrdzuje aj [7]. Avšak pri roli iniciátor boli ako najhodnotnejšie označené módy pero a papier, tabuľa a počítačová konferencia, ktoré uprednostňuje pred ostatnými. Realizátor uprednostňoval osobný kontakt, e-mail a počítačovú konferenciu. Ostatné role neboli vyhodnotené ako uprednostňujúce určitú špeciálnu formu komunikácie, pre role navigátor, organizátor a ukončovač boli významnými okrem osobného kontaktu e-mail a počítačová konferencia. Pre komunikáciu medzi rolami boli označené ako najhodnotnejšie médium osobný kontakt a telefón. Pre realizátora je na komunikáciu s ostatnými rolami významná aj tabuľa. Avšak pre komunikáciu iniciátor – vyšetrovateľ a iniciátor – koordinátor sa neodporúča pero a papier. Z tejto štúdie však vyplýva, že nie je možné vybrať konkrétne médium, ktoré bude všeobecne používané na celkovú komunikáciu, čo potvrdzuje teóriu mediálnej bohatosti.

Väčšina štúdií sa zameriava na klasické metódy vývoja softvéru. Avšak v [8] bola rozobratá možnosť aplikovania virtuálnych tímov pri agilných technikách vývoja softvéru. Tu sa už prejavujú nedostatky distribuovaných tímov, a to hlavne osobného kontaktu, ktorý je pri týchto technikách podstatný. Ukazuje sa, že osobný kontakt vedie k lepšej produktivite a kvalite výsledného produktu, ako aj vyššiemu uspokojeniu členov tímu z vykonanej práce ako v distribuovaných tímoch. Avšak na druhej strane nedostatok osobného kontaktu je možné zmierniť výborným zvládnutím komunikačných technológií, bližším spoznaním sa s partnermi, vzájomnou dôverou, motiváciou a aktivitami tvorby a utužovania tímu.

Záver

Manažment komunikácie pre tímy je veľmi dôležitou súčasťou úspešného vedenia tímu k výborným výsledkom. Je dôležité poznať obmedzenia a preferencie pre jednotlivé role členov tímu, ako aj činnosti, pre ktoré je potrebný konkrétny typ komunikačného média. Pri plánovaní komunikácie je potrebné brať tieto obmedzenia a informácie na zreteľ, aby bolo možné vhodným spôsobom zabezpečiť adekvátne komunikačné prostriedky a vyčleniť na ne vhodnú časť rozpočtu tímu.

Ukazuje sa tiež, že aj keď lokalizované tímy majú výhodu možnosti priamej osobnej komunikácie, táto výhoda nie je až taká markantná ako by sa mohlo zdať. Dokonca je

možné úspešne viesť virtuálne tímy pracujúce technikami agilného programovania, čo by sa zdalo takmer nemožné. Na druhej strane to však vyžaduje veľké odhodlanie nielen manažmentu, ale aj dobrú motiváciu jednotlivých členov tímu. Je však tak či tak nutné organizovať čas od času aj osobné stretnutia, bez ktorých sa (zatiaľ) vývoj softvéru nezaobíde.

Použitá literatúra

1. Bieliková, M.: *Softvérové inžinierstvo. Princípy a manažment.* Slovenská technická univerzita v Bratislave, Bratislava, 1971.
2. Graveline, A., Geisler, C., Danchak, M.: Teaming Together Apart: Emergent Patterns of Media Use in Collaboration at a Distance. In: *Proceedings of IEEE professional communication society international professional communication conference and Proceedings of the 18th annual ACM international conference on Computer documentation: technology & teamwork*, IEEE Educational Activities Department, Piscataway (2000), 381 – 393.
3. Michailidis, A., Rada, R.: Organizational Roles and Communication Modes in Team Work. In: *34th Annual Hawaii International Conference on System Sciences (HICSS-34)-Volume 8*, IEEE Educational Activities Department, Piscataway (2001), 8068 – 8077.
4. Amponsah, K.: Patterns of Communication and the Implications for Learning among Two Distributed-Education Student Teams. In: *Proceedings of the 21st annual international conference on Documentation*, ACM Press, New York (2003), 20 – 27.
5. Javed, T., Maqsood, M., Durrani, Q.: A Survey to Examine the Effect of Team Communication on Job Satisfaction in Software Industry. In: ACM SIGSOFT Software Engineering Notes, Volume 29 , Issue 2, ACM Press, New York (2004), 6 – 13.
6. Morris, M., Ogan, C.: Internet jako masové médium. In: *Journal of Communication 46 (1)*, Oxford University Press, Oxford (1996), 39 – 50.
7. Mark, G.: Building Virtual Teams: Perspectives on Communication, Flexibility and Trust. In: *ACM SIGGROUP Bulletin, Volume 19 , Issue 3*, ACM Press, New York (1998), 38 – 41.
8. Domino, M., Hevner, A., Collins, R.: Applying Agile Software Development Processes to Global Virtual Teams: A Study of Communication Modalities. In: *Proceedings of the 2002 ACM SIGCPR conference on Computer personnel research*, ACM Press, New York (2002), 76 – 78.

Annotation

Communication Management for Collocated and Distributed Software Teams

Communication within collocated, as well as distributed teams is one of the most important parts of their management and one of the key factors of their success at creating a good software product. This document concerns communication planning and information distribution as parts of communication management whereas it aims at definition of appropriate means of communication for tasks and roles in teams. This document describes the media richness theory and its practical application and tests, which were executed in recent years and confirmed it. Also it shows that distributed teams are capable of doing their tasks at least as well as collocated teams and that the advantage of personal communication is slowly diminishing by the creation and improvement of new communication means and taking full advantage of them successfully.

Vzťah zákazník a vývojár v softvérových projektoch

IVAN ŠKOVRAN

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

Abstrakt. Počas celého trvania vývoja softvérového systému dochádza ku komunikácii medzi zákazníkom a vývojárom. Podmienkou úspešnej špecifikácie požiadaviek, návrhu systému, predvádzania prototypu, či testovania je dobrý a fungujúci vzťah medzi zákazníkom a vývojárom. Tvorba softvérového systému nie je jednoduchá úloha, preto by sa obidve strany mali snažiť spraviť čo najviac pre úspešnú spoluprácu. Úlohou zákazníka je čo najlepšie, najpresnejšie a najzrozumiteľnejšie vysvetliť vývojárovi svoje požiadavky na softvérový systém. Na druhej strane, vývojár musí vedieť počúvať a pochopiť, čo sa mu zákazník snaží vysvetliť. Na základe získaných informácií musí vývojár navrhnúť softvérový systém, ktorý bude vyhovovať požiadavkám používateľa. V tejto práci sa snažíme zachytiť základné aspekty vo vzťahu zákazník a vývojár pri tvorbe softvérového systému. Najprv oboznámime čitateľa s problémovým prostredím. Potom uvedieme hlavné prekážky a problémy vo vzťahu zákazníka a vývojára. V poslednej kapitole sa venujeme rôznym metódam a nástrojom používaným pri tvorbe softvérového systému, so zameraním na zákazníka a vývojára.

Úvod

Proces získavania požiadaviek pri návrhu softvérového systému je nerozlučne spätý so získavaním najrozmanitejších informácií od zákazníka. V tomto procese sa vývojár snaží za pomoci rôznych nástrojov a postupov čo najlepšie porozumieť požiadavkám zákazníka. To, aký úspešný bude návrh softvérového systému, a v konečnom dôsledku aj aký úspešný bude celý softvérový projekt, závisí na komunikácii a spolupráci zákazníka a vývojára. Problém získavania požiadaviek so sebou nesie okrem aspektov spätých s technickou odbornosťou aj aspekty personálne, kultúrne, či dokonca psychologické. V každom prípade sa nejedná o triviálny problém. Aby bol vývojár schopný porozumieť

požiadavkám a potrebám zákazníka, potrebuje vedieť, ako so zákazníkom pracovať a komunikovať tak, aby ho úspešne zapojil do procesu tvorby softvéru.

Keď hovoríme o komunikácii medzi používateľom a vývojárom, máme na mysli tak komunikáciu medzi tímom vývojárov a komunitou používateľov, ako aj osobnú komunikáciu medzi jedným vývojárom a jedným používateľom. Vďaka osobnej a bezprostrednej komunikácii s jednotlivými zákazníkmi a sledovaniu a porozumeniu práce zákazníka je vývojár, a prostredníctvom neho celý vývojový tím, schopný dokonale porozumieť požiadavkám kladeným na softvérový systém. Samozrejme, komunikácia medzi nimi môže prebiehať aj nepriamo, prostredníctvom rôznych nástrojov na to určených, alebo prostredníctvom tretej osoby. Je to však práve osobný rozhovor tvárou v tvár, ktorým sa dá najlepšie odhaliť skutočná podoba požiadaviek zákazníka. Dobre vedený osobný rozhovor dokáže často odhaliť vývojárovi a dokonca aj zákazníkovi to, čo sa vlastne od systému očakáva, a ktoré črty systému sú pre zákazníka najpodstatnejšie.

V tejto práci sa bližšie pozrieme na súčasné možnosti a trendy v procese získavania požiadaviek. Zhrnieme základné metódy a postupy a pozrieme sa na hlavné funkcie a úlohy osôb, ktoré sú potrebné pre úspešné získavanie požiadaviek. Podrobnejšie sa budeme venovať problematike úzkej spolupráce medzi zákazníkom a vývojárom a možným spôsobom ich komunikácie. Možno sa pokúsime aj osloviť vývojárov a zákazníkov a poradiť im, ako zlepšiť ich vzájomnú spoluprácu, ktorej cieľom bude vytvorenie softvérového produktu, s ktorým budú obe strany spokojné.

Problémové prostredie

Prvým krokom pri vývoji softvérového systému je zistiť, na čo má byť systém určený, čo presne má robiť. Proces získavania požiadaviek má niekoľko fáz. Najskôr je potrebné identifikovať požiadavky od používateľa. Ďalším krokom je analýza týchto požiadaviek a ich následné zdokumentovanie za účelom špecifikácie a verifikácie používateľom. Získavanie požiadaviek môžeme klasifikovať ako úspešné, ak má zákazník pocit, že vývojár porozumel všetkým jeho potrebám a požiadavkám. Z pohľadu vývojára je získavanie požiadaviek úspešné, ak je pomocou nich možné dobre navrhnúť informačný systém.

Na vytváraní špecifikácie pre informačný systém väčšinou pracuje a spolupracuje väčší počet ľudí. Preto je potrebné pre každého člena zapojeného do projektu konkrétne špecifikovať jeho funkcie a role. Podľa [4] môžme role a funkcie klasifikovať nasledovne:

- *Kupec* je človek, ktorý má na starosti zmluvy, účty a peniaze. Kupec sa priamo nezúčastňuje na tvorbe projektu, jeho úloha je zviazaná skôr s manažmentom procesu.

- *Koncový používateľ* bude vytváraný softvérový systém používať. Títo ľudia by sa mali aktívne zúčastňovať na tvorbe softvérového systému, hlavne vo fáze špecifikácie požiadaviek.
- *Odborník z doménovej oblasti* najlepšie rozumie prostrediu v ktorom bude softvérový systém nasadený, hlavne z pohľadu systémových požiadaviek.
- *Údržbár* je zodpovedný za riadenie zmien a implementáciu zmien v softvérových produktoch, ktoré sa už v praxi používajú.
- *Programový manažér* je človek zodpovedný za obchodné a trhové záležitosti a vykonáva dozor nad vývojom softvéru. Často ide o ľudí, ktorí priamo komunikujú so zákazníkom.
- *Inžinier požiadaviek* je vlastne systémový inžinier a jeho úlohou je identifikácia požiadaviek a ich zdokumentovanie.
- *Softvérový inžinier* je expertom na návrh systému, vytvorenie prototypu a ďalšie technické záležitosti.
- *Tester* je zodpovedný za vývoj a vykonávanie testov, ktoré majú overiť správnosť a preukázať schopnosti systému. Ich činnosť je dosť dôležitá, pretože zákazníkovi musíme nejako preukázať, že systém spĺňa jeho požiadavky.

Prekážky vo vzťahu zákazník a vývojár

Proces získavania požiadaviek a s tým spojená komunikácia zákazníka a vývojára so sebou prináša rôzne problémy. Tu sa pokúsime zhrnúť základné problémy, ktoré sa pri tomto zložitom procese môžu vyskytnúť.

Slabá komunikácia

Vďaka komunikácii o cieľoch, úlohách, plánoch, obmedzeniach a prioritách nášho systému môžme vyvinúť softvérový systém, ktorý bude vyhovovať potrebám používateľa, no nemusí splniť všetky jeho očakávania. Pri komunikácii totižto musíme pozerať nie len na to o čom komunikujeme, ale aj na to, ako komunikujeme.

Najlepším a najefektívnejším spôsobom komunikácie je rozhovor tvárou v tvár. Pri tomto spôsobe komunikácie musia zúčastnení vedieť vhodne verbálne vyjadriť svoje požiadavky a musia byť samozrejme odborníkmi v téme rozhovoru. Inak je takmer isté, že nastanú problémy. Ak pri rozhovore medzi vývojárom a zákazníkom jedna strana zistí, že druhá nepočúva a nejaví o rozhovor záujem, znižuje sa tak efektivita tejto spolupráce a komunikácia viazne. Naopak, ak je vývojár dobrým poslucháčom, povzbudený rozprávač si to všimne, čo môže viesť len ku zlepšeniu komunikácie. Niektoré z príznakov, že s komunikáciou nie je niečo v poriadku:

- osoba na to, čo hovoríte, vôbec nereaguje
- ak s vami ten druhý stále súhlasí (za predpokladu, že nie ste jeho šéf :)
- ak druhá osoba nie je nikdy spokojná s tým, čo poviete.

Medzi vývojárom a zákazníkom funguje aj nepriama komunikácia, tj. komunikácia pomocou rôznych nástrojov alebo prostredníctvom tretej osoby. Je takmer vylúčené, aby medzi oboma stranami existoval len tento spôsob komunikácie. Tretia osoba často nepozná presné potreby a požiadavky zákazníka, a tak často dochádza k ich prekrúteniu a skresleniu. Nepriama komunikácia je vhodná len ako doplňujúca, a to v prípade, že osobný kontakt nie je možný.

Nezrozumiteľnosť a neodbornosť

Častou prekážkou pri komunikácii na technickej úrovni je používanie odborných názvov, termínov, ktorým niektorá zo strán nerozumie. Preto by sa mala strana, ktorá používa tieto termíny, snažiť svoje vyjadrovanie prispôsobiť úrovni druhej strany a snažiť sa zrozumiteľným jazykom vysvetliť daný problém.

Ďalším problémom, ktorý s touto tematikou súvisí je to, keď sa vývojár správa ku zákazníkovi ako ku hlupákovi, ktorý sa „do toho“ vôbec nerozumie. Takýmto prístupom nič nedosiahneme, toto často urážlivé správanie nikam nevedie. Vývojár by mal naopak zákazníkovi pomôcť porozumieť problematike ktorej nerozumie. Mal by pochopiť, že aj človek, ktorý sa vďaka stlačenému Caps Lock-u, nevie prihlásiť do systému, môže byť vo svojej oblasti skutočným odborníkom, a že nemá najmenší dôvod sa mu za to posmievať.

Rozdielne oblasti pôsobenia

Väčšina vývojárov má pri vývoji softvérového systému len veľmi málo, resp. žiadne skúsenosti z oblasti, pre ktorú je softvérový systém určený. Vývojárom chýbajú praktické skúsenosti a znalosti, ktoré sa dajú získať len prácou v danom odbore. Tieto skúsenosti má tvorcovi softvérového systému priblížiť samotný zákazník, používateľ. Zo skúsenosti vieme, že vo veľkom množstve vytvorených softvérových systémov, kde sa táto spolupráca podcenila, dostal zákazník hotový produkt, ktorý síce bol vytvorený podľa jeho požiadaviek, ale nenaplnil všetky jeho očakávania.

Na druhej strane je pre zákazníka a používateľa tvorba softvérového systému veľkou neznámou. Zákazník teda nevie, ako musí svoje požiadavky špecifikovať a aké obmedzenia si potrebuje stanoviť, aby sa jeho systém vôbec dal reálne navrhnúť a implementovať. Preto je práve na vývojároch, aby poučili zákazníka o tom, čo je, a čo nie je možné za daných podmienok navrhnúť a vytvoriť. Vývojár musí vedieť ako usmerniť zákazníka správnym smerom, ale tak, aby pôvodné požiadavky zákazníka zostali v najväčšej možnej miere zachované.

Metódy a nástroje

V tejto časti sa budeme venovať rôznym metódam a použitiu rôznych nástrojov, ktorých cieľom je zlepšiť vzťah a komunikáciu medzi zákazníkom a vývojárom. Zároveň uvedieme niektoré príklady z praxe, kde sa takéto techniky a nástroje používajú.

Získavanie informácií za pomoci komunikačných zručností

Ak ako vývojár dokážeme dať zákazníkovi najavo, že ho pozorne počúvame, pomôže to obom stranám pri získavaní informácií a celkovo pri budovaní ich vzťahu. Prejavy ako očný kontakt, poloha tela a výraz tváre pri počúvaní prezradí veľa o tom, či v skutočnosti dávame pozor a počúvame. To ako reagujeme na rozprávajúceho je aspoň tak dôležité ako to, ako rozprávame. Samozrejme, nemali by sme nechať zákazníka rozprávať monologicky. Pýtanie sa otázok, utvrdenie sa v problematike a písanie si poznámok dá jasne najavo, že sme v komunikácii aktívni. Samozrejme, všetkého veľa škodí. S kladením otázok by sme to nemali preháňať a nemali by sme rozprávajúceho stále prerušovať.

Ak zákazník nevie presne čo potrebuje, resp. nevie to definovať, je na vývojárovi, aby mu v tom pomohol. Zákazník používateľ ani nemusí byť schopný presne popísať to, čo potrebuje. Stačí, ak vývojárovi ukáže svoju každodennú činnosť, na ktorú sa viaže vyvíjaný informačný systém. Je úlohou vývojára pomocou týchto informácií navrhnúť pre používateľa konkrétne riešenie.

Naomi Kartenová v knihe Managing Expectation [2]ponúka zopár rád, ako získavať informácie od používateľa:

9. Nepredpokladajte čo zákazník myslí. Opakujte otázky, ktoré ste sa už pýtali, preformulujte ich. Dostanete na nich rôzne odpovede z rôznych pohľadov.
10. Žiadajte si vysvetlenie. Buďte si istý, že rozumiete tomu, čo vám zákazník hovorí. Nebojte sa priznať, že niečo neviete. Zdôraznite zákazníkovi, že vaše otázky sú nevyhnutné k tomu, aby ste plne pochopili jeho očakávania.
11. Zbierajte informácie z viacerých zdrojov. Pýtaním sa rovnakých otázok rôznych ľudí získate širší pohľad na ich potreby. Odpovede od viacerých ľudí vám pomôžu vyplniť diery v informácii získanej od jedného človeka.
12. Dávajte si pozor na nepresnosti. Zvážte zručnosti a reakcie zákazníkov. Odpovedajú vám na otázky z ich vlastného pohľadu, alebo sa vám len snažia povedať to, čo chcete počuť?

Návrhár ako zákazníkov učeň

V práci Apprenticing With the Customer [1] autori hľadajú odpoveď na otázku, aký vzťah by sa mal vytvoriť medzi zákazníkom a vývojárom, aby vývojár čo najlepšie porozumel požiadavkám zákazníka. Ich odpoveď na túto otázku znie: „Tak, ako sa učeň učí zručnosti od svojho majstra, tak sa aj návrhár chce naučiť niečo od zákazníka sledovaním jeho práce. Základné aspekty tohto vzťahu môžme zhrnúť nasledovne:

- Z pohľadu zákazníka, majstra, nie je schopnosť učiť potrebná. Majster neučí svojho učňa ako klasický učiteľ, ale učí ho vykonávaním svojej práce a komentovaním toho, čo robí. Zákazník teda len vykonáva svoju každodennú činnosť a komunikuje s učňom. Učeň počúva.
- Pozorovanie práce majstra odhalí to, čo je podstatné. Niektoré činnosti sú výsledkom skúseností, iné sa stali zvykom. Najlepšie sa o tom čo, ako, a prečo robím hovorí priamo, ak to v tom okamihu skutočne robím.
- Pozorovanie práce majstra odhalí všetky detaily. Komentovanie priamo pri vykonávaní práce zabráni zovšeobecňovaniu zo strany majstra. Ak majster hovorí o svojej činnosti, a popri tom ju nevykonáva, často sa stane, že si nepamätá všetky detaily, mieša činnosti dokopy, a neopisuje činnosti v správnom poradí.
- Odhalenie štruktúry práce zákazníka. Vývojár sa učí z pozorovania majstra, sleduje jeho metódy a stratégie a snaží sa im porozumieť. Keď im porozumie, začne vytvárať možné návrhy pre systém.
- Učeň sa učí zo skúseností svojho majstra. Majster vie pomocou vlastnej skúsenosti presne povedať, čo ide robiť, čo pri tejto práci potrebuje, aká je postupnosť krokov ktorú vykonáva, čo pomáha učňovi lepšie porozumieť jeho práci.

Medzi učňami a vývojármi však existuje jeden podstatný rozdiel. Učeň sa len naučí vykonávať majstrovu činnosť, vývojár musí navyše navrhnúť systém, ktorý majstrovi zjednoduší jeho činnosť.

- Návrhár musí odhaliť dôležité štruktúru v práci majstra. Návrhár hľadá v jeho práci postupy, obmedzenia a vzory a pod.
- Návrhár si musí ujasniť, či problematike skutočne rozumie. Pokiaľ je návrhár len pasívny, len pozoruje a nekomunikuje, nemôže navrhnúť dobrý systém. Návrhár si musí u používateľa overiť, či problému dobre rozumie.
- Prácou návrhára je zlepšiť prácu majstra. Návrhár musí stále rozmýšľať nad spôsobmi, ako vylepšiť a zefektívniť jeho činnosť.
- Návrhár vie majstra usmerniť. Návrhár je odborníkom na zovšeobecňovanie a vie odhaliť štruktúru práce, a preto musí vedieť majstrovi poradiť a vhodne ho usmerniť.

Prototypovanie

Dobrá spolupráca medzi vývojárom a zákazníkom je dôležitou podmienkou pre úspešné zvládnutie prototypovania. Prototypovanie znamená vytvorenie jednoduchého modelu systému za účelom lepšieho porozumenia systému a požiadavkám. Prototypovanie znižuje riziká a pomáha už na začiatku vývoja systému zabrániť chybám, ktorých odhalenie

a riešenie v neskorších fázach vývoja by bolo oveľa náročnejšie a nákladnejšie. Prototypovanie by malo prebiehať nasledovne:

1. Navrhni používateľské rozhranie, len so základnými funkciami.
2. Čo najrýchlejšie vytvor „hrubý“ systém. Systém nemusí byť spoľahlivý, má však mať čo najlepšie vytvorené používateľské rozhranie.
3. Nechaj používateľa „pohrať sa“ s prototypom, čím od neho získaš hneď na začiatku odozvu, kladnú, či zápornú.
4. Pozoruj používateľa. Všimni si, ako používateľ na systém reaguje.
5. Zisti názor používateľa na prototyp. Pomocou vhodných otázok zisti od viacerých používateľov ich názor, požiadavky na zmeny v prototype a pod.
6. Na základe spätnej väzby od používateľov prerob vytvorený prototyp a vráť sa k prvému kroku.

Pri prototypovaní dochádza k častému a opakovanému kontaktu medzi vývojárom a zákazníkom. Pre úspešný priebeh prototypovania je preto výhodné využiť niektoré zo skôr uvedených informácií o osobnej komunikácii.

Spôsoby výmeny informácií

Autori Mark Keil a Erran Carmel sa problematike vzťahu zákazníka a vývojára venujú z trocha iného pohľadu. Vo svojej práci Customer-Developer Links [3] skúmajú, ako závisí úspešnosť softvérových projektov od počtu a rôznorodosti komunikačných kanálov a techník (liniek) medzi vývojármi a zákazníkmi.

Autori porovnávajú výhody a nevýhody tých ktorých liniek používaných vo vývoji generického softvéru a softvéru na zákazku. Linky rozdeľujú na priame a nepriame. Priame linky sú tie, pri ktorých dochádza ku priamemu kontakty zákazníka a vývojára. Komunikácia cez tretiu osobu, nepriamo, je zasa charakteristická pre nepriame linky. Z prieskumu, ktorý bol vykonaný vyplynuli nasledovné závery:

Čim viac liniek, tým lepšie, ale ...

Projekty, v ktorých bolo použité väčšie množstvo liniek boli spravidla úspešnejšie. Pri úspešných projektoch stredná hodnota počtu použitých liniek bola 5,4. Menej úspešné projekty mali strednú hodnotu počtu liniek na úrovni 3,2.

Pravidlo čím viac liniek, tým lepšie, neplatí úplne. Mať v projekte príliš veľa liniek taktiež nie je optimálne. Výdavky spojené s implementáciou veľkého počtu liniek by mohli prevýšiť výhody, ktoré väčšie množstvo liniek prináša, čo by bolo kontraproduktívne.

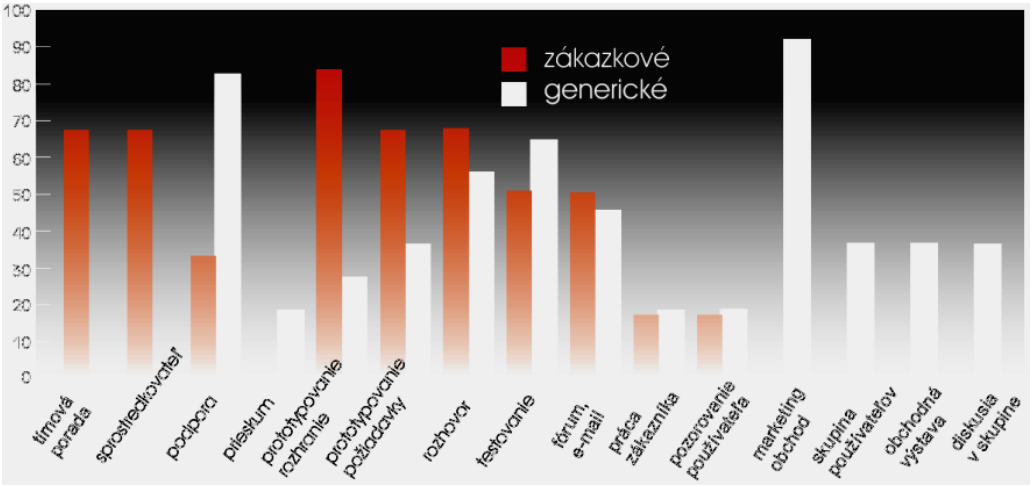
Nespoliehajte sa na nepriame linky

Na základe výsledkov prieskumu je jasné, že projekty, v ktorých boli vo väčšej miere zastúpené priame linky dopadli úspešnejšie, ako tie s malým počtom priamych liniek. Desať zo štrnástich menej úspešných projektov malo medzi zákazníkmi a vývojármi len

jednu, alebo žiadnu priamu linku. Dôvody sú zrejmé. Komunikácia cez tretiu osobu je „zašumená“, sprostredkovateľ nemusí informácie podať tak, ako mu boli zdelené.

Zvážte použitie rôznorodých liniek

Pri vývoji softvéru by manažéri mali zvážiť aj použitie liniek, ktoré nie sú v ich prostredí typicky používané, ale sú vnímané ako vcelku efektívne. Na nasledujúcom obrázku môžeme vidieť, ktoré linky a v akej miere sa používajú pri vývoji generického softvéru a softvéru na zákazku.



Obr.1. Percentuálne zastúpenie liniek pre zákazkové a generické softvérové systémy

Záver

Vzťah zákazník a vývojár v softvérových projektoch. Tento názov reprezentuje problematiku, ktorá v sebe zahŕňa množstvo zaujímavých a rozmanitých úloh. Snaha o ich úspešné riešenie stojí pred každým, kto to myslí s tvorbou softvérových systémov vážne. To, že táto problematika nie je triviálna, si po prečítaní tejto eseje určite každý uvedomil. Zachytáva skutočne široké spektrum tém a úloh, ktorých zvládnutie si vyžaduje veľa odborných znalostí od softvérového inžinierstva, cez komunikačné schopnosti, až po psychológiu.

V eseji som sa snažil načrtnúť, čo všetko táto bohatá téma obsahuje. Podrobne sa zaoberať všetkými jej aspektmi by bolo v takejto krátkej práci nemožné. Preto som sa snažil len zhrnúť o čo vlastne ide, vytvoriť o problematike ucelený obraz. Niektoré časti som rozvinul podrobnejšie, iné som len okrajovo spomenul. Dúfam, že sa aj čitateľovi,

ktorému softvérové inžinierstvo veľa nehovorí, podarilo porozumieť aspoň čo-to zo zložitého procesu tvorby informačného systému.

Použitá literatúra

1. Beyer, R. H, Holzblatt, K.: Apprenticing With the Customer. *Communications of the ACM,* Vol. 38, No. 5 (May 1995) 45-52.
2. Karten, N., *Managing Expectations*, Dorset House Publishing, New York, NY, 1994.
3. Keil, M., Carmel, E.: Customer-Developer Links in Software Development. *Communications of the ACM,* Vol. 38, No. 5 (May 1995) 33-44.
4. Saiedian, H., Dale, R.: Requirements engineering: making the connection between the software developer and customer. In: *Information and Software Technology,*Elsevier Science B. V. (2000), 420-428.

Annotation

Customer-Developer Relationship in Software Projects

For the whole software development process it is typical that there is active communication between customer and software developer. Functioning relationship between customer and developer is the key aspect of successful development process including requirements engineering, designing of the system, prototyping and testing. It is not an easy task to design and develop software system. Therefore they have to try hard to create a successful relationship full of collaboration. Basic task of customer is to explain to a developer, as good as it gets, how to fulfill his needs and requirements. On the other side, developer's task is to listen carefully and to understand customer's needs. After that, he must design software system, with the use of gathered information. In this essay we try to explain basic aspects in customer-developer relationship. In the first chapter, we explain to the reader background of this relationship. The next one deals with difficulties. The last chapter is dedicated to techniques and tools for enhancing the elicitation process.

Záver

V tejto chvíli ste sa dopracovali na samí záver našej práce. Veríme, že jej čítanie bolo pre vás poučné a bol to prínos pre vašu intelektuálnu či odbornú stránku. Diskutované témy boli analyzované viac či menej podrobne. Ku niektorým však boli iba načrtnuté základné otázky, no jadro problematiky ostalo nedotknuté. Pokiaľ by ste sa o niektorej z uvedených tém chceli dozvedieť niečo viac, ako je uvedené v tomto dokumente určite vám odporúčame nahliadnuť do niektorej z publikácií uvedenej v literárnych zdrojoch.