PREDISPOZÍCIE VÝVOJOVÝCH METODOLÓGIÍ Z HĽADISKA KVALITY

V dynamickej dobe sa zmena stala pravidlom.

Miroslav Hruška

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
Autor[zavináč]mail[.]com

Abstrakt. *Proces vývoja softvéru musí zohľadňovať oblasť z reálneho sveta, ktorú modeluje. Problematickosť vyjadrenia špecifikácie spoločne s jej priebežnými zmenami sa stali hlavnou bariérou jednoznačne určeného cieľa pre softvérový produkt. Vývoj softvéru tak bežne musí počítať s určitou mierou neistoty a uspôsobiť plán natoľko, aby dokázal dané zmeny pokryť. Nehovoríme však o okrajovom pokrytí neočakávanej zmeny, ale o jej plnohodnotnom spracovaní pre dosiahnutie požadovanej úrovne kvality. Správne zapracovanie zmeny do systému musí byť konzistentné s doterajším stavom a naďalej musí umožňovať jeho rozširovanie prípadným, vopred málo očakávaným smerom. Rozdiely medzi pohľadmi na vývoj medzi vodopádovým modelom, iteratívnou/ inkrementálnou metodológiou a agilnými metódami vývoja sú vhodnými kandidátmi na analýzu povahy kvality. Záverečné zistenia ustanovujú, že zmeny v procese vývoja softvéru môžu byť efektívne zachytené a spracované. Dokonca samotné zabezpečenie kvality má tendenciu byť úspešnejšie u procesov, ktoré sú na zmenu stavané.*

Kľúčové slová: *kvalita, softvér, vodopádový model, iteratívny a inkrementálny vývoj, agilné metódy*

Kvalita

Bez výhrad sa zhodneme na tom, že pod vysokou kvalitou rozumieme pozitívny atribút označovaného subjektu. Pod mojím pohľadom na kvalitu rozumiem takú mieru kombinácie atribútov veci, v akej sa pre účely svojho použitia nachádza v pozitívnych vzťahoch. Vo filozofickom popise pojmu sa dozvieme, že je to vlastnosť veci, ktorá pojednáva o tom, ako dobrá, alebo zlá je.

Z pohľadu softvérového inžinierstva, ako inžinierskej disciplíny sa presunieme hlavne do pragmatickej roviny. V nej môžeme jednoducho hovoriť o kvalite ako o miere splnenia požiadaviek. Na druhú stranu, takto vymedzený pojem je o ochudobnený voči predchádzajúcemu. Keď si uvedomíme, že hranice požiadaviek sú konečné, pridaním racionálnej veci neuvedenej v špecifikácií, môžeme dosiahnuť kvalitnejší produkt.

Samozrejme, práca nad rámec zvyšuje zaťaženie zdrojov. Som však presvedčený o tom, že existujú prípady, kedy malá nadpráca nepomerne zvýši kvalitu produktu. Ako príklad uvediem softvérovú spoločnosť, ktorá vyvíja komplexný softvér na správu určitého typu dokumentu. Z hľadiska používania softvéru, cieľovú skupinu budú prevažne tvoriť stredne zruční používatelia. Softvér je dostatočne komplexný a generický do takej úrovne, že je možné (a bežné) používať viacero jeho funkcií s malou obmenou za sebou. V takejto situácii by sa zišla akási elementárna forma automatizácie. Tú môžeme reprezentovať pomocou makier. Ak by bol softvér implementovaný aspektovo-orientovaným spôsobom, reprodukcia použitej funkcionality by bola jednoduchou, ba triviálnou záležitosťou.

Zmysel kvality

Vymedzením pojmu *kvalita* sa ujasnil priestor, v rámci ktorého možno uvažovať nad jej celkovými dôsledkami. Je dobré uvedomiť si, že produkty, ktorým pripisujeme kvalitu nie sú pasívne existujúce. Pod pasívnou existenciou rozumieme také bytie entity, ktoré by nemalo dôsledky vo svete.

V prípade kvality je možné z širšieho hľadiska použiť známe príslovie: reťaz je tak silná, ako jej najslabší článok. Bežne sa môžeme stretnúť s konkrétnou aplikáciou daného príslovia v oblasti bezpečnosti. Ak však vychádzame z predstavy fyzickej reťaze, treba mať na pamäti, že všetky jej články sú rovnaké a plnia rovnaký účel. V prípade abstraktnej reťaze produktov a procesov s nimi spätých to nemusí tak byť, pretože sa tam môžu nachádzať články, ktoré si explicitne nevyžadujú zvýšenú mieru snahy na to, aby plnili svoju úlohu. A napriek tomu môžu dosahovať vysokú kvalitu. Rozdiel s fyzickou reťazou sa teda stáva zrejmý.

Zamyslime sa nad dôsledkami existencie menej kvalitného článku v reťazi. Pri fyzickej reťazi je kvalita celku znížená, použiteľnosť zmenšená. Ak však prenesieme pojem reťaze inam, dospejeme k záveru, že do istej miery môže nasledujúci článok pokryť nedokonalosť predchádzajúceho. To je výrazne rozdielna charakteristika. Takéto riešenie však v sebe s veľkou pravdepodobnosťou skrýva zníženú ekonomickosť. Je to v skutočnosti analógia prevencie a riešenia. Cena riešenia problému často výrazne prevyšuje náklady spojené s prevenciou. Z uvedených dôvodov, možnosť pokrývania predchádzajúcich chýb robustným článkom reťaze je prospešná, ale signalizuje však

problém, ktorý by sa mal riešiť. A to najmä z toho dôvodu, že vysoké konkurenčné prostredie si vyžaduje kvalitu neustále zvyšovať.

Závislosť produktu od procesu

Práce významných expertov v oblasti kvality sú založené na premise [3], ktorá tvrdí, že „kvalita systému alebo produktu je vysoko ovplyvnená kvalitou procesu použitého na jeho vytvorenie a udržiavanie." V tomto prípade je podľa môjho názoru dobré si uvedomiť, že uvedení experti sa zaoberajú dosahovaním kvality všeobecne, teda nie konkrétne pre proces vývoja softvéru. V dobe, kedy ich práce začali nadobúdať hodnotu sa jednalo o technické procesy, ktorých výsledkom bol hmotný produkt. Analógiou preniesli experti poznatky z uvedenej oblasti do procesu vývoja softvéru. Dospeli k záveru, že „aj keď sa produkčné linky (ktoré sú orientované na stroje, opakujúce sa a ich výsledkom je množstvo výrobkov) a vývoj softvéru (orientovaný na ľudí, intelektuálny a výsledkom je jeden systém) rozlišujú, prechod princípov kvality je odôvodnený".

Keďže je to vážne a pomerne radikálne tvrdenie, je dôležité, aby bolo správne chápané v prípade, že o jeho pravdivosti budú pochybnosti. Prirodzene sa vynára otázka, ako súvisí kvalita výsledného *softvérového* produktu s odbornosťou vývojárov, ktorý ho vytvárajú. Štúdia [4], ktorá sa zaoberala vývojom 26 prevažne malých softvérových projektov, odpovedá na uvedenú otázku a tvrdí, že schopnosti vývojárov v *signifikantnej* miere ovplyvnili kvalitu návrhu a implementácie.

Právom sa vynára otázka, či z uvedených výsledkov vzniká rozpor. Zabezpečenie kvality pojednáva o pravidlách a smerniciach, ktoré sú vynikajúcimi odporúčaniami a majú potenciál vniesť do produktu kvalitu. Ak však v organizácii bude nedostatočne kvalifikovaných odborníkov, zrejme ani rafinované systémy na zabezpečenie kvality nepomôžu. Určitá rozporuplnosť tu podľa môjho názoru existuje a to najmä z toho dôvodu, že softvérový proces je proces týkajúci sa ľudí a tento faktor nemožno zanedbať ani do tej miery, ako bolo uvedené v [3].

Vývoj softvéru

Proces vývoja, vzhľadom na povahu softvéru, je proces, ktorý v mnohých prípadoch musí byť takpovediac šitý na mieru. Tvrdenie sa dá ešte upresniť tak, že proces vývoja musí byť šitý na mieru *zmenám*, očakávaným aj neočakávaným a najmä ich potenciálnemu rozsahu.

Z princípu konkurenčného prostredia vyplýva snaha byť najlepším pre každý subjekt v ňom zainteresovaný. Byť najlepším môže v sebe zahŕňať splnenie mnohých vlastností, ale je nám jasné, že medzi nimi na prvých miestach nebude chýbať kvalita. Polemika na tému samotného prvenstva sa môže odohrávať na rozmedzí rôznych definícií kvality a jej aplikovateľnosti z iných pohľadov. Dôležité však je, že kvalita tam *rozhodne* patrí.

Úspešné softvérové spoločnosti pri vykonávaní zákazky v závislosti na požiadavkách zákazníka musia vedieť aplikovať aspoň niekoľko *výrazne iných* princípov a postupov pri vývoji softvéru. V predchádzajúcom odseku bolo naznačené, že vývoj softvéru musí byť adekvátny konkrétnej situácií. V [2] je delenie metodológií založené na miere rigoróznosti, čo je podľa môjho názoru veľmi dobré a vhodné delenie. Dobré je nielen z dôvodu, že dokáže vystihnúť podstatu procesu vývoja softvéru. Dá sa aplikovať aj na prostredie, ktoré softvér modeluje. Teda rigoróznosť prostredia, s ktorou má byť softvér spätý odráža do

akej miery rigorózny model vývoja softvéru použiť. Podľa uvedeného pravidla sa delia typy procesu vývoja softvéru na:

- rigorózne
- iteratívne rigorózne
- agilné

Podrobný výpis konkrétnych metód vývoja je možné nájsť v [2, 9]. Z početného množstva metód je vhodné zvoliť tie, ktoré vo vysokej miere reprezentujú danú skupinu. Z uvedených dôvodov sú medzi ne postupne zaradené: *vodopádový model*, *inkrementálny a iteratívny* a agilná metodika *Scrum.*

Vplyv vývojových metodológií na kvalitu

Vodopádový model je lineárny, sekvenčný model, v ktorom je najväčšej miere prítomná rigoróznosť. Je to plánom riadený vývoj a v ortodoxnom poňatí neobsahuje možnosť vrátenia sa do predchádzajúcej fázy.

Inkrementálny a iteratívny vývoj je z hľadiska plánovania taký model, ktorý je z *rozumných* dôvodov rozdelený do *niekoľkých* častí. Na takýto proces vývoja sa dá pozerať ako na viacnásobne uskutočnený vodopádový model.

Treťou metodikou je agilná metodika Scrum, ktorú po hrubých zjednodušeniach môžme prirovnať k inkrementálnemu vývoju takým spôsobom, akým bol inkrementálny vývoj prirovnaný k vodopádovému.

Vezmime do hypotetickej úvahy, že všetky tri typy vývoja sú len rovnaké princípy rozmiestnené do časovej sféry rozličných rozmerov. Potom, prirodzene, ich jediný rozdiel vyplýva zo štrukturalizácie času. Znamená to toľko, že miera korelácie časových intervalov (v ktorých sa môže diať zmena, či požiadavka na funkčný produkt) modelovaného biznis prostredia s vývojovým procesom by mala byť maximálna. Z veľmi hodnotných informácií zo štúdií [6, 7], sa okrem samotnej štatistiky, dajú identifikovať nasledovné faktory, ktoré majú podľa môjho názoru *hlavný* vplyv na kvalitu produktu:

- úroveň špecifikácie
- skúsenosti vývojárov, resp. zložitosť vyvíjaného systému
- čas medzi začiatkom vývoja a začiatkom testovania
- produktivita, komunikácia v rámci tímu a frekvencia odovzdávania projektu

Ak by sme zotrvali pri úvahe rozdielu v rozmere času, tak rozdiel medzi vodopádom a metodikou Scrum sú dva rády. Dôsledky takéhoto *výrazného* rozdielu spočívajú už v celkom iných princípoch riadenia. Kým v iteratívnom a inkrementálnom vývoji sú postupy viazané na plán, v metodike Scrum je plánovanie minimalizované.

Úroveň špecifikácie

Vynikajúca špecifikácia produktu, nielen softvérového, je pre jeho úspešnosť a kvalitu fundamentálnou požiadavkou. Potrebu vynikajúcej špecifikácie môžeme vzhľadom na uvažovanú hypotézu rozdeliť pre jednotlivé metódy z časového hľadiska. Vodopádový model *musí* mať vynikajúcu špecifikáciu s presne danými požiadavkami na začiatku.

Konkrétne pri vodopáde ako sekvenčnom, lineárnom prístupe je táto požiadavka pre správny proces absolútne kritickou.

Rozumnú mieru neurčitosti je schopný pokryť iteratívny a inkrementálny vývoj. Môžeme o ňom uvažovať ako o princípe zhora-nadol, kedy postupnými iteráciami sa od abstraktného produktu s nižším množstvom konkrétnej funkcionality dostávame postupne k jeho plne funkčnej verzii. Z uvedeného vyplýva, že špecifikácia produktu *musí* byť jednoznačná iba pri prechode do ďalšej iterácie. Teda rozložením vývoja do iterácií sa skresala aj potreba komplexnej a presnej špecifikácie v zmysle zníženia jej ďalekosiahlosti. Riziko spočíva v možnosti nekonzistentnej špecifikácie s doterajším vývojom.

Scrum s filozofiou založenou na agilných princípoch musí vytvárať softvér, ktorý vychádza z modulárnej architektúry. Výskum [1] preukázal, že viac ako 90 percent spoločností praktizujúcich agilný spôsob vývoja, víta dynamické zmeny špecifikácie aj v *neskorých* fázach implementácie projektov. Z toho vyplýva, že dizajn projektov musí byť flexibilný a rozpracovaný na vysokej úrovni abstrakcie.

Zhodnotenie

Požiadavky na špecifikáciu sa z uvedeného zreteľne líšia. Plánovanie, ktoré následne vychádza zo špecifikácie bolo analyzované v štúdii [8]. Tá sa zaoberala úspešnosťou odhadnutia zdrojov. Vo výsledkoch sa uvádza, že zdroje boli odhadnuté správne pri 20% sekvenčných procesov a 50% pri flexibilných. Dôvodom sú v prípade sekvenčných procesov podľa mňa najmä nečakané „prekvapenia" v neskorých fázach implementácii a testovaní (tam, kde sa to nerobí pravidelne), ktorým sa budem venovať v podkapitole o produktivite.

Z tohto pomerne jasne vyplýva, že sekvenčné procesy majú potenciál byť úspešne a ekonomicky výhodné viac-menej v známych podmienkach. Pri iteratívnom vývoji je výsledkom iterácie spustiteľný produkt. Procesy sú však formálne, len delené do kratších časových úsekov. Treba dbať na to, aby v začiatku bol blízky kontakt so zákazníkom. V agilných metódach sa zákazník stáva *súčasťou* tímu a jeho spoluúčasť umožňuje preklenúť dva rozdielne svety – svet modelovaného prostredia a IT svet. Špecifikácia sa vyvíja v čase a keďže je zákazník prítomný v procese vývoja, tak aj špecifikácia naberá *správnu* formu. Pod správnou formou myslím takú forma, ktorá vyhovuje vývojárom.

Zložitosť systému, resp. skúsenosti vývojárov

Podľa môjho názoru je zložitosť systému inverznou funkciou k skúsenostiam vývojárov. Na skúsenosti sa pozerám ako na dlhodobú pamäť vývojárov, ktorá zaradená v kontexte systému umožňuje v ňom nachádzať vyššiu mieru súvislostí.

Vodopádový model a inkrementálny si vyžadujú vyššiu mieru skúsených vývojárov v konkrétnej oblasti. Na druhú stranu, agilné metódy vyzdvihujúce do popredia ľudí a podporujúce komunikáciu v tíme, umožňujú túto mieru znížiť vďaka zvyšovaniu produktivity. Produktivite sa budem venovať aj v nasledujúcej podkapitole.

Zhodnotenie

Pri agilných metódach je dôležité, aby boli používané správne, k čomu je potrebné zabezpečiť školenia. Agilné metódy však môžu mať problémy s vývojom veľkých

projektov. Myslím si, že decentralizácia organizácie na jednotlivcov je už pri veľkom tíme kontraproduktívna. Viem si predstaviť akúsi predfázu vývoja, kedy sa veľký projekt rozdelí do maximálne nesúvisiacich častí tak, aby zodpovedali optimálnej veľkostí jednotlivých tímov pre agilný vývoj. Narúša to však samotný princíp agilnosti. Na druhú stranu si myslím, že ak by pri takomto vývoji bol v každom tíme zákazník a robili by sa denné stretnutia aj v rámci zákazníkov a hlavných vývojárov, tak by sa opäť dali dosiahnuť výborné výsledky. V takomto prípade by som pri agilnom vývoji odporúčal, aby v prvotných fázach bol na komunikáciu medzi tímami kladený vysoký dôraz.

Z opačnej strany sa však naskytá otázka, aká je šanca, že vo veľkom projekte budú očakávané veľké zmeny s podmienkou, že na pravidelnej (a častej) báze prichádza funkčný softvérový produkt. Ak uvážime, že veľký projekt reprezentuje veľkú časť reality, tak sa podľa môjho názoru také veľké zmeny v realite neudejú. Teda v takomto prípade by som agilný vývoj nahradil plánovaným. Predsa len si myslím, že nie je úplne možné, aby obrovský projekt nemal v *rozumnej* miere určené hranice.

Produktivita, komunikácia v rámci tímu a frekvencia odovzdávania projektu

Uvedená kategória vplyvu na kvalitu má veľa spoločného, aj keď sa to na prvý pohľad môže javiť nejasne. Pri vodopádovom modeli môže byť s produktivitou vývojárov problém z toho hľadiska, že produkt sa odovzdáva „niekedy neskôr“. Psychologické dôvody na pokusy o riešenie problémov svojpomocne (aj napriek vysokej časovej neefektívnosti) môžu výrazne degradovať produktivitu. Absencia neformálnosti v tíme neumožňuje prirodzené pýtanie sa a riešenie problémov.

V iteratívnom vývoji sa problém zmenšuje, keďže funkčný produkt odchádza v pravidelných intervaloch. V ponímaní produktivity je iteratívny prístup kompromisom.

Pri agilnom vývoji, kedy sa musí pravidelne prezentovať dosiahnutý pokrok, je potenciál produktivity nastavený správnym smerom. Štúdia [6] sa zaoberala prechodom projektu z iteratívneho spôsobu na Scrum. Po ukončení projektu boli uskutočnené rôzne interview. Tím sa zhodol na tom, že veľké množstvo krátkodobého plánovania (v rámci denných stretávok) a spätná väzba *prospešne* vplývala na zjednotenie spoločného chápania systému a pomáhala *včas* predchádzať chybám. V prípade problémov pri vývoji platilo, že každý môže sám stráviť riešením problému maximálne desať minút. V štúdii sa naďalej uvádza, že 68% respondentov považuje Scrum za produktívnejší v zmysle, že predchádza nevyužitiu času.

Zhodnotenie

Myslím si, že ľudia, ich pamäť a schopnosti myslenia za predpokladu efektívnej práce inklinujú ku kratším časovým plánom. Pretože konkrétne činnosti je možné veľmi jednoducho zmerať a jednoznačne posúdiť pokrok. Z nielen osobnej skúsenosti zároveň viem, že sledovanie vlastného progresu má pozitívny psychologický dopad. Preto oblasť produktivity a pravidelné dodávky softvéru mi prídu ako jednoznačne dobré riešenie.

Taktiež si myslím, že keď sa vytvorí línia v akej zákazník dostáva softvér, tak zároveň začína na určitej úrovni rozumieť procesom v spoločnosti. Myslím si preto, že jeho nasledujúce požiadavky sú v menšej miere odtrhnuté od IT reality. To je podľa mňa dôležité, pretože dochádza ku konsenzu medzi tým, čo už je vytvorené a akým spôsobom

to vytvorené je, a tým, čo bude zákazník žiadať. Myslím, že aj to je dôvod vysokého percentuálneho ohodnotenia (viac ako 90%), ktoré pojednávalo o vítaní dynamickej zmeny v neskorej fáze implementácie v agilných procesoch. Z tohto dôvodu z môjho pohľadu vychádzajú ako *výrazne lepšie* z hľadiska kvality pre malé a stredné projekty agilné procesy.

Čas medzi začiatkom vývoja a začiatkom testovania

Na skúsenosti vývojárov som navrhoval pozerať sa ako na ich dlhodobú pamäť. Podobným spôsobom by som chcel zasadiť do kontextu projektu „krátkodobú" pamäť, ktorá má možnosť prejaviť sa v čase od začiatku vývoja po začiatok testovania. Inak povedané času, ktorý je venovaný implementácií.

Myslím si však, že nedokonalosť ľudskej pamäte v tomto prípade nie je jediným faktorom, ktorý ovplyvňuje zmysel pravidelného testovania. Myslím si totiž, že pravidelným testovaním (a najmä integračným) môžeme redukovať návrhové chyby, ktoré by sa aj pri perfektnej ľudskej pamäti riešili neskôr veľmi ťažko. Vývojovou metódou, ktorá má najčastejšiu dodávku softvéru je z uvádzaných v eseji metodika Scrum.

Prirodzene vzniká otázka, či platí, že čím skôr sa bude testovať, tak tým kvalitnejší produkt dostaneme. Extrémnym prípadom je testom riadený vývoj, ktorého efektivitou sa zaoberali v štúdiách [5, 10]. Výsledky sa zhodujú v tom, že sa *zvýšila* kvalita kódu a produktivita. Na druhú stranu kvalita návrhu ostala na rovnakej úrovni a priemerne sa čas vývoja softvéru daným spôsobom zvýšil o približne 20%. Myslím si, že výhodou testom riadeného vývoja je, že má pokryté funkcionálne časti kódu. Z toho aj vyplýva, že tvorba novej funkcionality má odozvu a testy ukážu, či jej implementácia neporušila predchádzajúce väzby v kóde.

Zhodnotenie

Medzi implementáciou a testovaním by mal byť krátky časový úsek, pretože kým si vývojári pamätajú kód, dokážu s ním efektívnejšie pracovať. V prípade veľkých projektov (kde je problém s agilnými metódami a čas medzi testovaním a implementáciou je dlhý) by bolo potrebné vytvoriť efektívny systém na uvedenie vývojárov do obrazu.

Osobne by som navrhoval vytvoriť vizualizáciu zdrojového kódu na základe anotácií. Anotáciami by boli označované špecifické funkcionálne vlastnosti metód (napr. transformačná funkcia, zabezpečujúca dáta, a pod.). Následne by sa mohla vygenerovať vizualizácia kódu s tým, že by špeciálnym spôsobom boli označované podmienky a cykly spoločne s funkciami, ktoré sa v nich nachádzajú. Anotované metódy by podľa typu boli organizované do vrstiev. Vrstvy by sa dali vypínať a zapínať. Racionálnym vytvorením typov funkcionálnych vlastnosti metód (t.j. typov anotácií) by sa dal kód organizovať na vyššej úrovni. Práve táto vyššia úroveň by umožnila skrze vizualizáciu rýchlejšie predstavenie si kódu. Rovnako by sa dala vizualizácia použiť aj na ladenie.

Záver

Identifikovaním hlavných faktorov pre dosiahnutie kvality bolo možné zhodnotiť pre jednotlivé vývojové metódy úroveň, v akej ich ponúkajú. Štúdie, ktoré obsahovali

množstvo štatistických údajov boli cennými zdrojmi poznatkov na analýzu vývojových metód. Štúdie takmer jednohlasne vyzdvihovali agilné metódy, pričom ich hlavnou limitáciou je veľkosť tímu. Taktiež bola upravená predstava o zabezpečení kvality z rýdzo manažérskeho pohľadu, kde v štúdii bolo preukázané, že odbornosť vývojárov je signifikantný faktor. Kým vo vodopádovom a inkrementálnom modeli to môže spôsobiť vážne dôsledky, agilný vývoj sa s tým dokáže skrze silnú komunikáciu vyrovnať. Dôležité však je, aby agilné metódy boli aplikované v spoločnosti správne, pretože ich nepochopenie môže priniesť problémy. Vidíme, že pomocou menej zdatného tímu je možné skrze agilný vývoj a správne pochopenie princípu tvoriť kvalitnejšie produkty ako v ostatných dvoch typov vývoja. Agilné metódy teda definitívne patria do malých a stredných projektov.

Nakoľko v menších projektoch je agilná metóda veľmi prospešná, bolo by zaujímavé analyzovať možnosti zavedenia agilných metód do veľkých organizácií. V prípade, že by sa podarilo vytvoriť rafinovaný systém, bolo by to určite na osoh celej informatickej spoločnosti.

Použitá literatúra

1. Ahmed A. et al.: Agile software development: Impact on productivity and quality. In: 2010 IEEE International Conference on Management of Innovation & Technology Conference on Management of Innovation & Technology. 2010, s. 287-291.
2. Ambler W. Scott. 2009. Choose the Right Software Method for the Job. http://www.agiledata.org/essays/differentStrategies.html
3. Baker R.E. et al.: Handbook of software quality Assurance. Fourth Edition. Artech House, Norwood. 2008, s. 260.
4. Beaver, J.; Schiavone, G.: The Effects of Development Team Skill on Software Product Quality ACM SIGSOFT Software Engineering Notes. Software Engineering Notes, Vol. 31, No. 3. 2006, s. 1-5.
5. Bhat, T.; Nagappan, N.: Evaluating the efficacy of test-driven development: industrial case studies. In: Proceedings of the 2006 ACM/IEEE. 2006, s. 356-363.
6. Li, J.; Moe, N. B.; Dybå, T.: Transition from a Plan-Driven Process to Scrum – A Longitudinal Case Study on Software Quality. In: Science And Technology, 2010.
7. Mitchell, S. M.; Seaman, C. B.: A comparison of software cost, duration, and quality for waterfall vs. iterative and incremental development: A systematic review. In: 3rd International Symposium on Empirical Software Engineering and Measurement. 2009, s. 511-515.
8. Moløkken-Østvold, K.J; Jørgensen, M.: A Comparison of Software Project Overruns – Flexible vs. Sequential Development Models. IEEE Transactions on Software Engineering, Vol. 31, No. 9. 2005, s. 754-766.
9. Ruparelia, N. B.: Software development lifecycle models. ACM SIGSOFT Software Engineering Notes, Vol. 35, No. 3. 2010, s. 8-13.

10. Siniaalto, M.; Abrahamsson, P.: A Comparative Case Study on the Impact of Test-Driven Development on Program Design and Test Coverage. In: First International Symposium on Empirical Software Engineering and Measurement. 2007, s. 275-284.

Annotation

Predispositions of software development methodologies in terms of quality

Software development process must take into account the area of the real world it models. Problems with expression of specification together with the interim changes have become a major barrier to clearly describe target software product. Software development typically must rely on some degree of uncertainty and adapt a plan so that it can meet those changes. However, we are not talking about marginal cover of unexpected change, but its full-fledged processing to achieve the required level of quality. Proper incorporation of changes to the system must be consistent with the actual state and must continue to allow its expansion potential, few expected direction in advance. Differences between views on the development of the waterfall model, iterative / incremental methodology and agile development methods are good candidates to analyze the nature of quality. We can conclude that the change in the software development process can be efficiently captured and processed. Even the mere provision of quality tends to be more successful in processes that are built for change.