Máme vhodné prostriedky na zabezpečenie kvality a testovanie aj malom projekte? Alebo nestrieľajme kanónom na vrabce.

MICHAL KURŤÁK

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

Abstrakt. Vývoj softvérových systémov je náročná úloha, ktorej sa obyčajne zúčastňuje veľký počet ľudí. Zabezpečenie kvality takto vyrábaných produktov sa preto nezaobíde bez systematického riadenia kvality. Táto esej rozoberá existujúce štandardy riadenia kvality, zamýšľa sa najmä nad ich použiteľnosťou v malých projektoch a tímoch, pričom vychádza aj z osobných skúseností. Esej porovnáva tieto odporúčané a pomerne zložité procesy riadenia kvality s inými odľahčenými metodikami. V práci sa ďalej zamýšľam nad úlohou a dôležitosťou testovania pri zabezpečení kvality. Na testovanie je možné pozerať z rôznych uhlov pohľadu. V dokumente sa pokúšam konfrontovať teóriu s možnosťami, ktoré máme v praxi. Tieto možnosti sú väčšinou dané podmienkami kontraktov so zákazníkmi a z toho vyplývajúcich obmedzení. Opäť sa snažím porovnať štandardné metódy tvorby plánov a riadením dokumentácie s agilnými metódami testovania a zamyslieť sa nad akýmsi prienikom týchto dvoch prístupov, ktorý bude vhodný pre tím menšieho rozsahu.

Prečo sa zaoberať kvalitou softvéru?

Skôr ako sa začnem zamýšľať nad zabezpečením kvality v softvérovom projekte je potrebné pojem kvality definovať.

„Pod kvalitou rozumieme súhrn vlastností a charakteristík výrobku procesu alebo služby, ktoré preukazujú jeho schopnosť splniť určené alebo odvodené potreby.“ (ISO 8402)

Na prvý pohľad je zrejmé, že takáto definícia kvality je pomerne abstraktná a len málo výrobkov s takouto definíciou kvality vystačí. Napríklad skrutka je výrobok, ktorý má pevne definované požiadavky. Ak zákazník požaduje skrutku, ktorá je vyrobená z určitého materiálu a „v ťahu“ udrží stanovený počet kilogramov, vie povedať, ktorá skrutka spĺňa jeho základné potreby, a ktorá jeho potreby nespĺňa.

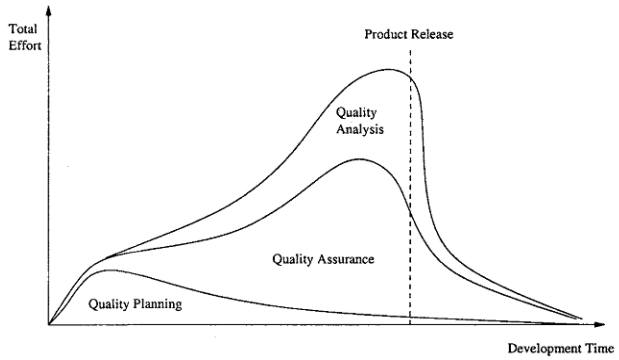
Manažment v softvérovom inžinierstve, október 2006, s. 1-3.

Ak sa pokúsime takúto jednoduchú úvahu uplatniť na softvérový výrobok, narazíme na viacero problémov. Spomeniem dva najzávažnejšie:

- požiadavky zákazníka na softvérový výrobok sú veľmi málo jasné a zrejmé
- softvér je „živý organizmus“, po odovzdaní softvérového výrobku vzniká skoro okamžite požiadavka na jeho zmenu alebo na novú funkcionalitu

Uvediem prípad, v ktorom sa ukáže závažnosť druhého problému v spojitosti s kvalitou. Softvérová firma odovzdá objednaný softvérový výrobok na čas, pričom výrobok spĺňa, dokonca v niektorých častiach prevyšuje požiadavky zákazníka. Zákazník je nadmieru spokojný, ale používatelia systému zistia, že by bolo vhodné, keby dodaný program obsahoval ešte ďalšiu funkcionalitu. Z pohľadu zákazníka, ide o malú zmenu, preto kontrakt o dodaní softvéru je podpísaný s relatívne krátkym časovým intervalom na modifikáciu výrobku. Dodanie sa však natiahne o pomerne dlhú dobu, zákazník nie je spokojný, označí výrobok ako nekvalitný a výrobca softvéru môže na takomto projekte utrpieť stratu. Vzniká otázka, ktorý výrobok bol nekvalitný? Prvý výrobok alebo neskoršie odovzdaný modifikovaný výrobok?

Z uvedeného vyplýva, že kvalita v softvérovom projekte je komplexný pojem a zasluhuje si potrebnú pozornosť v závislosti od veľkosti projektu. Celkovú kvalitu v softvéri podľa [4] ovplyvňuje kvalita softvérového procesu, kvalita zdrojov, technológia vývoja, náklady, rozvrh práce a samozrejme kvalita samotného výrobku. Samotný manažment kvality je zložitý proces, ktorý sa skladá z troch subprocesov: plánovanie kvality, zabezpečenie kvality a riadenie kvality (Obr. 1 Množstvo práce vynaloženej na manažment kvality v závislosti od času).



Obr. 1 Množstvo práce vynaloženej na manažment kvality v závislosti od času

ISO/IEC 9126, ISO 9000 a CMMI

Zabezpečenie spomínaných procesov manažmentu kvality je pokryté viacerými normami, z ktorých som vybral tri najvýznamnejšie.

Norma ISO/IEC 9126 definuje model slúžiaci na špecifikáciu a ohodnotenie kvality softvérového produktu. Kvalitu hodnotí systémom šiestich základných charakteristík, ktoré sa rozdeľujú na ďalšie subcharakteristiky. Jednotlivé subcharakteristiky odrážajú rôzne aspekty kvality softvéru a sú určené atribútmi. ISO/IEC 9126 poskytuje návod pre zostrojenie metrík jednotlivých atribútov, čiže definuje spôsob ako hodnotiť kvalitu softvéru viac-menej absolútnou mierou. Ľahko si vieme predstaviť, že dôkladná analýza charakteristík a ich atribútov, nemusí priniesť očakávané výsledky. S veľkosťou projektu rastie aj komplexnosť charakteristík, ťažkosti s ich meraním a je viac ako pravdepodobné, že táto norma by bola prekážkou v riadení malých projektov.

Norma ISO 9000 a projekt CMMI sa vzťahujú najmä na manažment kvality z hľadiska riadenia, zabezpečenia a plánovania kvality pomocou riadenia procesov. ISO 9000 je pomerne komplikovaná, všeobecná norma, ktorá rýchlo podlieha byrokratickému prístupu ([4]). Z tohto dôvodu je vhodná najmä pre rozsiahlejšie projekty a spoločnosti. V porovnaní s modelom CMM v [1] v súvislosti s manažmentom softvérových procesov sa ukázalo ISO 9000 ako menej použiteľné, resp. jeho princípy sú ukotvené vo vhodnejšom CMM.

CMM je model, ktorý klasifikuje vyspelosť a úroveň procesu. Z hľadiska softvérového inžinierstva je dôležitý projekt CMMI, ktorý si dal za úlohu integrovať CMM modely súvisiace so softvérovým inžinierstvom. V súčasnej dobe existuje v rámci CMMI 25 procesných oblastí rozdelených do štyroch skupín, ktoré definujú stupne vyspelosti (stupňovitá reprezentácia CMMI). CMMI definuje 6 (0 až 5) úrovní pre „capability“(úroveň procesu) a 5 (1 až 5) úrovní pre „maturity“(vyspelosť procesu).

CMM vyspelosť úrovne 2, kam sa podela kvalita?

Je možné, že aj napriek dôkladnému riadeniu procesov nie je dosiahnutá požadovaná kvalita softvérového výrobku? Uvediem príklad situácie, kedy aj napriek snahe vedenia nebolo možné požadovanú kvalitu dosiahnuť. Spoločnosť mala vytvorenú SQA skupinu (skupina zodpovedná za zabezpečenie softvérovej kvality). Táto skupina mala na starosti riadenie dokumentov, školenia, interné audity a podobné náležitosti. Tím, v ktorom som pracoval sa skladal zo 4 vývojárov, projektového manažéra, 2 nezávislých testerov a jedného zamestnanca zodpovedného za kvalitu. Úroveň 2 modelu CMM hovorí najmä o opakovateľnosti, avšak vyrábané softvérové výrobky boli skoro nemodifikovateľné. Problém celej situácie bol v tom, že „kvalitár“ sledoval najmä kvalitu dokumentov, pretože mal na starosti kvalitu veľkého počtu projektov, do ktorých nevidel. V projekte chýbali metriky, pričom moje osobné snahy stroskotávali na zlých časových odhadoch v kontraktoch a neochote o aspoň malé posúdenie kódu („review“). Testy prebiehali iba na systémovej úrovni, pričom sa takto neodhalilo množstvo negatívnych scenárov, ktoré vyšli na povrch v „ostrej“ prevádzke u

zákazníka, pretože chýbali jemnejšie stupne testovania. Kvalita práce jednotlivých zamestnancov bola meraná časom stráveným nad úlohou. Po určitom čase sa produkt dostal do tak nevyhovujúceho stavu, že vznikla požiadavka na jeho úplnú prestavbu. V tomto projekte bolo hlavným problémom nepochopenie kvality ako takej, vynaloženie nákladov na silné prostriedky (CMM, skupina SQA, vysoké výdavky na riadenie dokumentov) a nesprávne rozloženie síl. Ak si v príručke prečítame ako zostaviť tím a ako dosiahneme kvalitu, neznamená to, že sa to hodí na náš projekt. Na dosiahnutie kvality, by v takomto menšom tíme postačovali vybrané časti zo spomínaných metodík a noriem, jeden skúsený pracovník, ktorý by mal okrem inej práce na tom istom projekte na starosti podporu v oblasti kvality.

Ako je to s kvalitou v malých projektoch

Predchádzajúca úvaha na tému štandardov, ktoré definujú kvalitu a poskytujú systémy riadenia kvality nám ukázala, že tieto štandardy nie sú schodná cesta pre malé projekty a tímy. Potvrdilo sa však, že bez riadenia kvality sa nezaobíde ani projekt, na ktorom pracuje 5 až 7 ľudí.

Prvú možnosť ako riadiť kvalitu malom projekte nám ukazuje článok [2], ktorý sa zaoberá zúžením princípov CMM na malý projekt použitím tzv. PSP (Personal Software Process – Osobný Softvérový Proces). Podstata tohto princípu spočíva vo výbere iba určitých procesných oblastí a pre každú túto oblasť definovaním cieľov podľa úrovne CMM. Procesná oblasť Plánovanie softvérového projektu na úrovni 2 definuje tieto ciele:

1. Softvérové odhady sú plánované a zaznamenávané v softvérovom projekte
2. Aktivity a príspevky týkajúce sa softvérového projektu sú plánované a zaznamenávané
3. Všetky osoby a skupiny, ktoré prispievajú do projektu súhlasia s týmito činnosťami v softvérovom projekte

Takto presne definovanými cieľmi sa zbavíme ťažkopádnosti CMM a naviac PSP definuje aj konkrétne princípy, ktoré nás k spomínaným cieľom dovedú. Je však dôležité poznamenať, že kvalita v softvérových projektoch nie je samostatný pojem a je závislá od spôsobu riadenia celého projektu. Metodiky podobné alebo súvisiace s klasickým PMBOK (Process Management Body of Knowledge) ako RUP (Rational Unified Process) alebo CMMI môžeme stotožniť so spomínaným zjednodušeným CMM modelom, no pri menších projektoch často uvažujeme o použití agilných metodík vývoja softvéru.

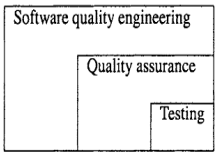
Agilné metodiky programovania a riadenia projektov samostatne nedefinujú procesy manažmentu, ale sústreďujú sa na prísne zachovanie určitých princípov, v ktorých sú tieto procesy zahrnuté. Spomeniem niekoľko princípov, ktoré by som zaradil medzi princípy ovplyvňujúce kvalitu, a ktoré môžeme považovať za vhodné princípy pri akejkoľvek metodike riadenia projektu:

1. Všetci, ktorí prispievajú do projektu pracujú spolu na jednom mieste. Ak s nami zákaznícka strana pracuje priamo na projekte, nemusíme venovať toľko úsilia špecifikácii a požiadavkám. Všetci poznajú projekt do hĺbky.
2. Plánovanie projektu.
3. Častá až neustála integrácia produktu. Snažiť sa vytvárať jednotlivé časti aplikácie tak, aby mohli byť samostatne integrovateľné. Oveľa rýchlejšie vieme posúdiť aktuálny stav projektu a následne odhadovať nové činnosti bez namáhavého a podrobného plánovania.
4. Časté posúdenia kódu („reviews“). Nie je priamo súčasťou agilných metodík, ale z môjho pohľadu vhodnejšie ako nesplniteľné párové programovanie. Ak si zavolám pri uložení do úložiska kolegu na posúdenie toho, čo som spravil spotrebujeme málo projektového času, ale veľa získame. S týmto princípom súvisí aj princíp spoločného vlastníctva kódu.

V uvedených bodoch som nespomínal testovanie softvéru, pretože sa mu venuje nasledujúca časť dokumentu. Môžeme zhrnúť, že riadenie kvality v malom projekte môže spočívať v definovaní cieľov a aktivít na ich dosiahnutie tak ako v [2] s dohliadnutím na to, aby boli v čo najvyššej miere dodržované spomínané princípy agilného programovania.

Testovanie a jeho miesto v manažmente kvality

Doteraz som hovoril najmä o riadení kvality a o meraní kvality prostredníctvom metrík. Neoddeliteľnou súčasťou riadenia kvality je validácia a verifikácia produktu a jeho súčastí. Existujú dve základné techniky validácie a verifikácie a to statické techniky a dynamické techniky. Keďže statické techniky validácie a verifikácie nie sú predmetom tohto dokumentu, nebudem sa nimi ďalej zaoberať, ale rozoberiem bližšie metódy dynamického overovania, ktoré sa týkajú procesu testovania softvéru. [5] uvádza, začlenenie testovania do procesu kvality tak, ako je to na Obr. 2 Hierarchia procesov kvality (Pod „Software quality engineering“ rozumieme nám známy pojem



Obr. 2 Hierarchia procesov kvality

manažment kvality).

Pod testovaním softvéru podľa [6] rozumieme „*proces, alebo sériu procesov navrhnutých preto, aby sme sa uistili, že program robí to na čo bol navrhnutý a nespráva sa nijak nečakane.*“. Proces testovania je podľa [5] rozdelený na 3 základné činnosti:

1. Plánovanie a príprava testov – zahrňuje vytýčenie cieľov, stanovenie stratégie testovania, príprava testovacích prípadov a testovacích procedúr
2. Vykonávanie testov – vykonanie samotných testov, sledovanie a meranie správania produktu
3. Analýza a vyvodenie dôsledkov

Podobne aj ako v iných oblastiach manažmentu kvality si môžeme pri vykonávaní týchto aktivít pomôcť štandardom. Riadením testovacej dokumentácie, teda činnosťami zahrnutými v bodoch jeden a dva sa zaoberá štandard IEEE 807 z roku 1998. Štandard vysvetľuje význam testovacích prípadov, procedúr, štruktúru plánu testov aj výstupy testovania.

Aké techniky testovania máme k dispozícii?

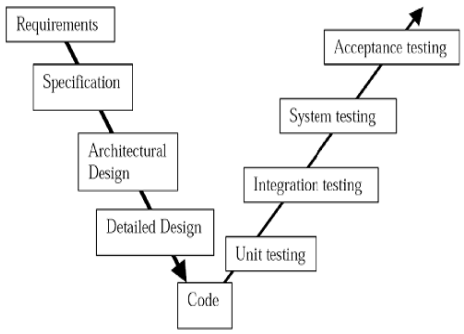
Na základe definície testovania uvedenej v predchádzajúcej časti vystupuje do popredia najmä jeden dôležitý poznatok. Ak chceme testovať, musíme vedieť čo program robí a na čo bol navrhnutý. Z toho vyplýva, že vieme ako sa má program správať a aké sú jeho výstupy pri všetkých vstupoch. Predchádzajúca laická úvaha nás vedie k tomu, že softvérový produkt, príp. jeho súčiastky môžeme testovať minimálne dvoma metódami.

Prvou skupinou testov z tohto pohľadu je funkcionálne testovanie. Účelom funkcionálneho testovania je najmä zistenie, či vstupno-výstupné správanie vyhovuje špecifikácii ([4]). Príkladom konkrétnej metódy testovania je metóda „black box“. Jej základom je absencia znalostí o akejkoľvek vnútornej štruktúre testovaného komponentu, preto sa niekedy táto metóda označuje aj ako „data-driven“.

Opačným prístupom k funkcionálnemu testovaniu je testovanie štrukturálne. Opäť sa budem zaoberať iba dynamickými technikami štrukturálneho testovania. Štrukturálne testovanie, niekedy označované aj ako „white-box“, prípadne „glass-box“ vychádza z vnútornej štruktúry programu. Tri najdôležitejšie techniky štrukturálneho testovania sú:

- pokrytie ciest – každá cesta (tok riadenia) v programe sa vykoná aspoň raz
- pokrytie vetvení – každá vetva v programe sa prejde aspoň raz
- pokrytie príkazov – každý príkaz v programe sa vykoná aspoň raz

Tieto dve metodiky testovania nám však ešte nehovoria ako a čo testovať. Techniky štrukturálneho testovania nie je možné vo väčších projektoch úplne pokryť a naopak funkcionálne testovanie nám nedá úplnú odpoveď na otázku, či je softvér správny. Z tohto dôvodu si potrebujeme zvoliť správnu stratégiu testovania počas jednotlivých fáz životného cyklu softvéru. V-model tvorby softvéru definuje stratégie testovania tak ako na Obr. 3 Fázy testovania vo V-modeli vývoja softvéru (zdroj [5]).



Na uvedenom modeli sa nachádzajú rôzne úrovne testovania softvéru:

- Unit testovanie – Testovanie jednotiek je testovanie na úrovni implementácie. Viacero metodík kladie dôraz na vytvorenie testov ešte pred samotným implementovaním jednotiek a takýmto spôsobom vytvoriť špecifikáciu jednotiek.
- Integračné testovanie – Integračné testovanie je metóda testovania, ktorá sa zaoberá najmä integrovaním rôznych komponentov (jednotiek), pričom sa zameriava na spoluprácu medzi nimi. V niektorých procesoch predchádza integračnému testovaniu ešte testovanie komponentov, pretože vo väčšine prípadov netvorí komponent iba jedna jednotka. Testovanie komponentov, však zdieľa princípy s testovaním integrácie, preto nie je rozoberané do väčšej hĺbky. V zásade máme 4 možnosti testovania integrácie komponentov ([4]) a to: zhora-nadol, zdola-nahor, jednofázové testovanie a sendvičové testovanie.
- Systémové testovanie – Pod systémovým testovaním sa chápe testovanie programu vzhľadom na jeho pôvodné ciele, ktoré má spĺňať a má za úlohu demonštrovať ako program. Systémové testovanie sa skladá z viacerých prípadov: testovanie prostriedkov, testovanie kapacity, záťažové testovanie, použiteľnosti, bezpečnosti, výkonnosti, konfigurácií, kompatibility, spoľahlivosti, opraviteľnosti, dokumentácie a testovanie procedúr.
- Akceptačné testovanie – Akceptačné testovanie je testovanie pripravené ešte pred spustením. Môžeme ho chápať ako určitý druh kontraktu medzi zákazníkom a objednávateľom. Ak akceptačné testovanie dopadne pozitívne, softvérový produkt je pripravený na odovzdanie.

Silné nástroje a prostriedky, ale ako to použiť

Pri testovaní softvéru a celkovom manažmente kvality môžeme naraziť dva zásadné a v dnešnej dobe často spomínané problémy tzv. „over-engineering“(vynaloženie priveľkého, nepotrebného inžinierskeho úsilia) a „under-engineering“(vynaloženie

veľmi malého inžinierskeho úsilia). Príklad tohto problému som už raz načrtol v časti venovanej manažmentu kvality. V prípade testovania sa nachádzame v podobnej situácii, máme k dispozícii silné prostriedky, ale problém je zvoliť správnu mieru použitia týchto prostriedkov.

Testovací plán podľa IEEE 807 pokrýva aj fázy vývoja softvéru a definuje ich ako takzvané „test levels“. Problémom môže byť výber vhodnosti testovacích stratégií v jednotlivých fázach. Riešenie ponúka autor [5] na Tab. 1 Spôsoby testovania vo fázach životného cyklu a prislúchajúce roly.

Fáza	Perspektíva	Rola
testovanie jednotiek	white-box	programátor
integráčné testovanie	white-box	tester
systémové testovanie	black-box	tester
akceptačné testovanie	black-box	nezávislý tester/zákazník

Tab. 1 Spôsoby testovania vo fázach životného cyklu a prislúchajúce roly

Už pojednanie o riadení kvality ukázalo, že pre projekt menšieho rozsahu nie je výhodné zaoberať sa štandardom, pretože zvyčajne adresuje veľké projekty a pre malé projekty a tímy je prakticky nepoužiteľný až škodlivý. V predchádzajúcej časti som spomenul niektoré princípy agilného programovania, ktoré sa mi ukázali ako vhodné pre takýto projekt. Uvediem ešte ďalšie dva princípy, zovšeobecnené tak, aby boli použiteľné aj v iných prístupoch riadenia projektu:

1. TDD (Test Driven Development). Testom riadený vývoj znamená vytvorenie testu ešte pred implementovaním prípadu použitia, triedy alebo metódy. TDD môžeme zovšeobecniť tak, že test (minimálne jeho špecifikáciu) vytvárame postupne zhora nadol ešte pred začatím fázy vývoja. Naviac unit testovanie nás núti do určitého spôsobu implementácie tak, aby bol kód testovateľný a minimálny – spĺňal „iba“ testy. Ak by sa nám podarilo testovať pomocou unit testov do čo najvyšších úrovní (integračné a systémové testovanie) vývoja, môžeme riadiť zdrojový kód presne podľa špecifikácie z týchto úrovní už v skorej fáze vývoja softvéru. Je samozrejmé, že na týchto vyšších úrovniach nie sme schopní otestovať všetky scenáre, ale takýto spôsob testovania nám môže ušetriť množstvo práce. Určité postupy ako testovať pomocou unit testov aj na systémovej a integračnej úrovni uvádza [3], kde sú spomenuté aj princípy unit testovania pomocou metódy Right-BICEP. Táto metóda vysvetľuje akým spôsobom a čo testovať, pretože klasické metódy white-box testovania sú v praxi často nepoužiteľné alebo časovo nesplniteľné.
2. Akceptačné testovanie. Princíp akceptačného testovania bol už skôr vysvetlený, agilné metodiky kladú dôraz na to, že tieto testy sú akýmsi druhom kontraktu so zákazníkom.

Záver

Správny manažment kvality pri riadení softvérového projektu nie je jednoduchá úloha, ktorá je však o to potrebnejšia rovnako v malých a veľkých projektoch. Zabezpečenie a riadenie kvality je citlivá činnosť, ktorá je závislá od viacerých faktorov. Množstvo štandardov, odporúčaní a noriem, ktoré sa sústreďujú na kvalitu softvérových produktov a aj napriek tomu nie sú jednoducho a priamočiaro použiteľné iba potvrdzuje tento uzáver. Stratégia vhodná pre malý projekt je často iba kompromisom medzi zúženou štandardnou stratégiou a princípmi agilných metodík. Pomer v tomto kompromise je určený povahou konkrétneho projektu. Podobná situácia je aj v časti zabezpečenia kvality, a to v testovaní softvéru. Je vhodné zaviesť plán testovania a výstupné dokumenty testov, v kombinácii s technológiou TDD a testovaním na úrovni fáz vývoja softvéru.

Kvalita a testovanie sú súčasťou softvérových výrobkov, ktorá sa nesmie v žiadnom prípade podceniť, no súčastne neplatí, koľko prostriedkov na kvalitu vynaložíme, taký kvalitný produkt bude.

Použitá literatúra

1. Paulk, M.C.: A Comparison of ISO 9001 and the Capability Maturity Model for Software. *Software Engineering Institute, Carnegie Mellon University*, July 1994.
2. Saedian, H., Carr, N.: Characterizing a Software Process Maturity Model for Small Organizations. In: *ACM SIGICE Bulletin*, Vol. 23, No. 1, Júl 1997
3. Hunt, A., Thomas, D.: Pragmatic Unit Testing in Java with JUnit. *The Pragmatic Programmers*, September 2003. ISBN 0-974-51401-2.
4. Bielikova, M.: Softvérové inžinierstvo, Princípy a manažment. *Slovenská technická univerzita*, Bratislava, 2000
5. Tian, J.: Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement. *Wiley-IEEE Computer Society Press*, Február 21, 2005. ISBN 0-471-71345-7.
6. Myers, G.J. et. al..: The Art of Software Testing, Second Edition. *John Wiley & Sons*, Jún 21, 2004. ISBN 0-471-46912-2.

Annotation

Do we have sufficient ways and means for quality assurance and testing in small project?

Development of software systems is a complicated task, usually contributed by many people. Therefore quality assurance of this kind of pruducts does not get along without systematic quality management. This essay analyzes existing standards for software quality, contemplates about their usability in small projects and teams and bears also on personal experiences. Essay compare these recommendations and relatively complicated quality management processes with

other lightweight methodologies. Further, I have focused on meaning and importance of testing in quality assurance. It's possible to look at software testing from various views. This article tempts to confront theory with real possibilities. These possibilities are generally given by contract conditions and limitations. Again I tempt to compare standard methods of testing and control of test documentation with agile methodology testing and contemplate about some intersection of theses two accesses, which would be suitable for small team.