Naozaj dokážeme vytvoriť kvalitný softvér?

BC. MICHAL OKRESA

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

Abstrakt. Zvyšovať kvalitu softvéru počas vývojového cyklu môžeme splnením dvoch základných požiadaviek. Tou prvou je používanie metodík, ktoré zabezpečia vývoj vysoko kvalitného softvéru. Druhá požiadavka súvisí s nasadením techník, ktoré zaručia požadované atribúty kvality v existujúcich artefaktoch. Oba prístupy musia byť kombinované, ak má byť vývojový tím efektívny a úspešný.

V tejto eseji opisujeme procesy verifikácie, validácie a testovania a ich použitie v rámci životného cyklu softvéru. Približujeme čitateľovi metódy a postupy, ktoré sa používajú vo fáze špecifikácie požiadaviek, analýzy, návrhu, implementácie a počas nasadenia softvéru v prevádzke. Venujeme sa porovnaniu troch najpoužívanejších metodík vývoja softvéru: RUP, MSF a XP, z hľadiska zabezpečenia kvality.

Pohľad na realitu

Organizácia Standish Group každoročne potvrdzuje fakt, že úspešný softvérový projekt je dnes skôr vzácnosťou. Do tejto skupiny patrí len menej ako tretina [**Error! Reference source not found.**] zo všetkých projektov. Nedostatočné zapojenie koncových používateľov, komplikované definovanie požiadaviek, zle definované procesy a množstvo iných faktorov ovplyvňuje úspech projektu. Problémom sa dokonca nevyhnú ani najväčší softvéroví giganti. Príkladom zo súčasnosti je spoločnosť Microsoft, ktorá mešká niekoľko rokov s uvedením operačného systému novej generácie alebo neschopnosť Unixových operačných systémov presadiť sa na poli domácich počítačov.

Otázka z nadpisu tejto eseje je teda skutočne na mieste. Sme naozaj na tom tak zle, že nedokážeme vytvoriť kvalitný softvér? A čo je to vlastne tá kvalita?

Manažment v softvérovom inžinierstve, október 2006, s. 1-11.

Kvalita

Kvalitu softvéru môžeme definovať ako súhrn atribútov, ktoré ovplyvňujú schopnosť uspokojovať stanovené alebo predpokladané požiadavky zákazníka. Na základe tejto voľnej definície môžeme chápať kvalitný softvér ako: [**Error! Reference source not found.**]

- Spoľahlivý
  - Adekvátny
    - Korektný
    - Kompletný
    - Konzistentný
  - Robustný
- Testovateľný
  - Porozumiteľný
    - Štruktúrovaný
    - Stručný
    - Samo-opisný
  - Merateľný
- Použiteľný
- Efektívny
- Prenositeľný
- Udržiavateľný

Jednotlivé body si môžu navzájom odporovať alebo sa vylučovať. Napríklad efektívnosť je mnohokrát v rozpore s prenositeľnosťou a porozumiteľnosťou. Takáto situácia môže nastať, ak sú pri vývoji použité optimalizačné techniky nízko-úrovňového jazyka. Preto je dôležité, aby každý softvérový projekt mal definované priority a relatívnu dôležitosť pre zlepšovanie atribútov kvality.

Zvyšovať kvalitu softvéru môžeme používaním vhodných metodík a nasadením techník počas vývoja, ktoré dohliadnu na atribúty kvality.

Verifikácia, validácia a testovanie

Pre každý softvérový projekt musia byť definované požiadavky na validáciu hneď na začiatku projektu. Tie ovplyvňuje veľkosť projektu, unikátnosť, nasadenie v kritickom prostredí, straty pri výpadku a rozpočet. Po stanovení požiadaviek môžeme pristúpiť k výberu špecifických techník pre validáciu, verifikáciu a testovanie.

V nasledujúcich kapitolách budeme rozoberať niekoľko možných prístupov, ale ešte predtým si zadefinujeme niektoré pojmy, aby nedochádzalo k nedorozumeniam: [**Error! Reference source not found.**]

- *Verifikácia* – preukázanie konzistencie, kompletnosti a správnosti softvéru v každej etape a medzi etapami vývojového cyklu.

- *Validácia* – určenie korektnosti výsledného softvéru s prihliadnutím na potreby a požiadavky používateľa. Validácia sa zvyčajne vykonáva verifikáciou každej fázy vývojového cyklu.
- *Testovanie* – overenie funkčnosti spustením programu s testovacou vzorkou dát.

V prípade, že jedným z cieľov projektov je dosiahnutie najvyššej kvality za minimálnu cenu, musí byť verifikácia zahrnutá do každej fázy vývoja. Pretože objavenie chyby v ranných štádiách znižuje náklady na opravu. Ak by sme sa spoliehali len na validáciu, tak by sme čoskoro narazili na problém testovania: testovaním dokážeme, že sme nenašli žiadne chyby nie, že je program bezchybný.

Verifikácia počas životného cyklu softvéru

Tab. 1 súhrne opisuje činnosti verifikácie v rámci životného cyklu softvéru. V nasledujúcich kapitolách budú tieto aktivity popísane bližšie.

Fáza životného cyklu	Verifikácia
Špecifikácie požiadaviek	Výber metód verifikácie. Určenie primeranosti požiadaviek. Generovanie testovacích údajov pre funkčné testovanie.
Návrh	Overenie konzistencie návrhu s požiadavkami a primeranosti návrhu. Generovanie testovacích údajov pre funkčné a štrukturálne testovanie.
Implementácie	Overenie konzistencie implementácie s návrhom a primeranosti implementácie. Generovanie testovacích údajov pre funkčné a štrukturálne testovanie . Aplikácia testovacích dát.
Prevádzka a údržba	Verifikácia v súvislosti s prípadnými zmenami softvéru.

Tab. 1 Verifikácia v jednotlivých fázach vývoja softvéru

Špecifikácia požiadaviek

V tejto fáze by mala byť formulovaná všeobecná stratégia testovania spolu s výberom vhodných metód a kritérií pre vyhodnocovanie testov. Rovnako je dôležité vytvorenie plánu testov. V prípade, že projekt je rozsiahly alebo je kritického charakteru, môže byť vytvorený samostatný testovací tím.

Počas procesu formulácie požiadaviek by mali vznikať testy, ktoré zabezpečia, že požiadavky budú testovateľné.

V tejto fáze by mali byť formulované pravidla pre zabezpečenie kvality (QA – Quality Assurance) a pre tvorbu dokumentácie z testov.

Návrh

Počas detailného návrhu by mali vzniknúť testovacie procedúry, testové prípady a testovacie údaje na overenie návrhu. Vlastnosti štruktúry systému a interakcie podsystémov môžu byť overené simuláciou.

Preskúmanie návrhu môže byť vykonané testovacím tímom, ktorý sa zameria na odhalenie chýbajúcich prípadov použitia, chybnej logiky, problémov s medzi-modulovými komunikáciami, nekonzistencie dátových štruktúr a nevhodnosti rozhraní pre používateľa.

Implementácia

Počas tejto fázy vývoja dochádza ku spúšťaniu vytvoreného kódu nad množinou testovacích dát. Táto etapa je veľmi dobre pokrytá rôznymi nástrojmi pre statickú a dynamickú analýzu, ktoré tiež podporujú testovanie samostatných modulov aplikácií. Je vhodné kombinovať tieto techniky spolu s formálnymi prehliadkami kódu. Počas testovania je veľmi dôležitá kontrola výsledkov testov, preto tie by mali byť kategorizované a archivované.

Prevádzka a údržba

Po finančnej stránke, práve údržba softvéru zaberá veľkú časť nákladov. Počas obdobia, kedy je softvér používaný, je mnohokrát potrebný zásah kvôli oprave chýb alebo doplneniu funkcionality. Po každej takejto modifikácii je potrebné opätovne otestovať softvér, či nevznikli chyby. Takéto testovanie sa nazýva *regresné testovanie*. Cenu regresného testovania do značnej miery ovplyvňuje kvalita testovej dokumentácie. Ak boli prípady testov dôkladne kategorizované a popísané, tak sa minimalizuje duplicitná práca a tým sa zefektívni celý proces testovania.

Techniky validácie, verifikácie a testovania

Metódy testovania softvéru môžeme rozdeliť do dvoch základných tried: *statické* a *dynamické*. Dynamická analýza vyžaduje spúšťanie programu nad množinou testovacích dát pomocou pripravených postupov. Výsledky testov sú následne vyhodnocované, či sa zhodujú s očakávanými. Na druhej strane statické testovanie nevyžaduje spúšťanie programu. Základné techniky analýzy sú vykonávané prekladačom zdrojových kódov, napríklad syntaktická a typová kontrola.

Dôkladná verifikácia v každej fáze vývojového cyklu musí zahŕňať otestovanie každého elementu domény. Táto technika sa volá *úplné* alebo aj *vyčerpávajúce testovanie*. Nie je praktická, pretože problémová doména môže byť veľmi veľká alebo dokonca nekonečná a preto vytvorenie testov by bolo nereálne. Na overovanie správnosti sa využíva podmnožina alebo inak nazývaná aj *testovacia množina* údajov. S pomocou tohto pojmu môžeme formalizovať testovanie takto: [**Error! Reference source not found.**]

Ak existuje konzistentné, spoľahlivé, správne a kompletné kritérium pre výber testovacej množiny pre program P a ak testovacia množina vyhovuje kritériám – teda hlavne, ak všetky testy prebehnú úspešne, tak program P je správny.

Nanešťastie bolo dokázané, že neexistuje algoritmus na nájdenie konzistentného, spoľahlivého, správneho a kompletného testovacieho kritéria. To dokazuje, že testovanie a hlavne vyčerpávajúce testovanie je veľmi zložitý proces. Napriek tomu kombináciou rôznych techník môžeme zvýšiť kvalitu softvéru. Táto kombinácia techník silne závisí od povahy problému. Môžeme využiť neformálne spôsoby testovania, keď vývojár kontroluje po sebe alebo po kolegovi vyprodukovaný kód až po podstatne prepracovanejšie techniky ako je tímová prehliadka projektu. V niektorých prípadoch môžeme využiť aj simuláciu pre určenie výkonnosti a náročnosti algoritmov. Simulácia dokáže odhaliť aj na prvý pohľad skryté chyby, ak sa softvér testuje nad veľkou množinou vstupných údajov.

Prehliadka projektu a manuálna simulácia

Z množstva rôznych techník sme vybrali pre bližší opis tímovú prehliadku projektu a manuálnu simuláciu. Aj keď obe vychádzajú z neformálnych techník, tak ich disciplinované postupy sa snažia odbremeniť programátora od zodpovednosti za verifikáciu. Obe techniky sú založené na čítaní výstupov z jednotlivých etáp (požiadavky, špecifikácia prípadne programový kód), v tíme vedenom moderátorom. Rozdiel medzi nimi je vo vedení stretnutí.

Prehliadka projektu zahŕňa kontrolu produktu krok po kroku s kontrolou každého kroku voči preddefinovaným kritériám. Tieto kritériá obsahujú kontrolu na známe historické chyby, dodržiavanie štandardov, konzistenciu so špecifikáciou prípadne ďalšie. Zvyčajne vývojár opisuje práve preberanú časť, pričom odôvodňuje spôsob riešenia. Týmto procesom a následnou diskusiou sú odhalené chyby, ktoré by sa mohli prejaviť až v neskorších fázach projektu.

Manuálna simulácia sa odlišuje od prehliadky tým, že vývojár neopisuje preberanú časť. Moderátor, programátor alebo iná osoba z tímu poskytne testovacie údaje a vedie proces manuálnej simulácie systému. Priebežné výsledky simulácie sa zaznamenávajú napríklad na tabuli. Účelom manuálnej simulácie je vyvolať diskusiu a tým objaviť chyby pri rozoberaní rozhodnutí vývojárov.

Vo fáze analýzy problému môžu byť tieto techniky použité na overenie, či požiadavky spĺňajú mieru testovateľnosti a primeranosti.

Počas detailného návrhu môže byť prehliadka alebo manuálna simulácie použitá na overenie testovateľnosti a primeranosti modulov a rozhraní. Akákoľvek zmena vyplývajúca z tejto fázy, vyvoláva opakovanie procesu verifikácie skorších fáz projektu a overovanie konzistencie medzi týmito fázami.

V priebehu implementácie môžu byť použité techniky verifikácie na analýzu separátnych modulov a ich následnú integráciu do celého systému.

Najvýznamnejšie metodiky vývoja softvéru

V predchádzajúcich kapitolách sme sa venovali všeobecným odporúčaniam a metódam na zabezpečenie atribútov kvality softvéru. Danou problematikou sa podrobne zaoberajú v literatúre [**Error! Reference source not found.**]. V nasledujúcom texte sa pokúsime načrtnúť ako sú tieto všeobecné pravidlá zaradené v troch najvýznamnejších metodikách vývoja softvéru RUP, XP a MSF. Okrem toho sa budeme venovať aj iným pravidlám, ktoré sa postupom času rozvinuli a vo významnej miere ovplyvňujú celkovú kvalitu softvéru.

Z rôznej kategorizácie prístupov ku softvérovej kvalite môžeme vybrať štyri najhlavnejšie, použitím ktorých ju môžeme zlepšovať [**Error! Reference source not found.**]: lepšie vyhodnocovanie kvality, lepšie meranie, lepšie procesy a nástroje. Na procesnej úrovni v [**Error! Reference source not found.**] odporúčajú použitie štandardov ako Total Quality Management (TQM), ISO 9001, Capability Maturity Model (CMM) alebo Personal Software Process (PSP).

CMM alebo ISO 9001 formujú proces zabezpečenia kvality v zmysle splnenia potrebných požiadaviek, ale nevenujú sa samotnej kvalite procesu. Lepším riešením je zahrnutie vyhodnocovania kvality priamo do vývojového procesu. Pre úplnosť ešte uvádzame, že RUP spĺňa požiadavky CMM úrovne 2 a 3 a XP je s CMM kompatibilné.

Agilné metódy (ako napríklad XP) definujú praktiky, ktoré zabezpečujú okrem vývojového procesu aj riadenie kvality: tvorba unit testov, priebežná integrácia a akceptačné testovanie.

Základné praktiky riadenia kvality

Nasledujúce praktiky boli vybrané z modelov vývoja (XP, RUP, MSF) metódou zdola nahor [**Error! Reference source not found.**].

Iteratívny vývoj softvéru

Pre zlepšenie kvality softvéru by sa mal vývoj riadiť iteratívnymi a inkrementálnymi pravidlami. Iteratívny vývoj je charakterizovaný zjemňovaním a zlepšovaním artefaktov v niekoľkých cykloch. Inkrementálny vývoj znamená, že na začiatku začíname so šablónou artefaktu, ktorá sa počas niekoľkých iteračných cyklov špecializuje a zjemňuje.

Iteračný cyklus zahŕňa všetky aktivity, teda analýzu, návrh, implementáciu, testovanie a nakoniec nasadenie softvéru. Samotný softvér týmto prístupom nadobúda na flexibilite a skorá spätná väzba od zákazníka pomáha zlepšovať kvalitu. Avšak, aby sme využili všetky možnosti iteratívneho vývoja softvéru, musí byť sprevádzaný manažmentom rizík a aktívnou účasťou koncového zákazníka.

Tento prístup do veľkej miery využíva extrémne programovanie (XP), ktoré vyžaduje tvorbu denných zostáv komponentov. To limituje čas potrebný na nájdenie a opravu chýb.

Kvalita ako cieľ

Pre zvýšenie celkovej kvality softvéru je dôležité, aby bola definovaná ako jeden z najdôležitejších cieľov. Ciele musia byť podrobne zadefinované zapojením projektového tímu a koncového zákazníka. To pomôže zaručiť jej dosiahnuteľnosť a merateľnosť.

Microsoft Solution Framework (MSF) definuje ciele kvality na začiatku a zdôrazňuje ich splnenie ako dôležitú časť projektu.

Nepretržitá verifikácia kvality

Každá zmena by mala byť podrobne zdokumentovaná. Nepostačujú správy o stave projektu, ale na identifikáciu problémov sú potrebné aj ohodnotenia súčasných aktivít a možných zmien.

Nepretržitá verifikácia kvality zahŕňa aj podrobné testovanie. Okrem interných testov sú požadované aj externé akceptačné testy pri zákazníkovi, aby bolo zaručené, že produkt spĺňa všetky potreby a požiadavky.

Požiadavky zákazníka

Proces vývoja softvéru je postavený na jasnej štruktúre a metodológií, aby zaistil a zdokumentoval požiadavky zákazníka. Analýza požiadaviek je jedna z najzložitejších disciplín v rámci softvérového inžinierstva. Potreby a požiadavky zákazníka, ktorý zvyčajne nemá hlboké technické znalosti problematiky, musia byť zdokumentované, aby mohli vývojári vytvoriť produkt na základe týchto informácií. Implementácia požiadaviek musí byť sledovaná počas celého vývojového cyklu.

Okrem komunikácie so zákazníkom musia prebiehať aj stretnutia s koncovým používateľom. Vývojový proces musí vyprodukovať procedúry pre zaškoľovanie koncových používateľov.

Zameranie na architektúru

V modernom softvérovom vývoji má architektúra významný vplyv na celkovú kvalitu produktu. Dôležitou časťou dnešného vývoja je integrácia do existujúcich systémov a prostredí. Význam opätovného využívania komponentov systému rastie so stúpajúcim tlakom na znižovanie nákladov.

Význam tímu

Tím môžeme charakterizovať ako skupinu ľudí, ktorí sú zodpovedný za úspech a kvalitu projektu. Ak je zodpovednosť za neúspech priradená len niekomu z tímu, tak sa úspech projektu ohrozí. Upriamenie pozornosti na tímovú prácu zvyšuje motiváciu členov, pretože každý bude mať pocit rovnakej dôležitosti. V konečnom dôsledku sa

jednotliví členovia stotožnia s projektom, budú viac motivovaní a tým pádom ich práca bude podstatne kvalitnejšia. Projekt musí mať definovanú jasnú štruktúru tímu vrátane efektívneho prideľovania úloh a pravidiel komunikácie.

Programovanie v páre

Programovanie v páre bolo veľmi dlho podceňované a spochybňované v súvislosti so zlepšovaním kvality softvéru.

XP demonštruje ako sa môžu dvaja vývojári navzájom dopĺňať. Napríklad jeden vývojár implementuje aktuálnu metódu, pričom druhy pracuje na jej integrácii. To šetrí čas a minimalizuje chyby, keďže dvaja ľudia majú odlišný pohľad na problém a teda sa môžu navzájom dopĺňajú pri jeho riešení.

Prispôsobovanie s mierou

Proces vývoja softvéru musí byť definovaný a formalizovaný s ohľadom na aplikáciu v širokom spektre projektov v rámci organizácie. Proces má byť prispôsobiteľný pre rôzne projekty založené na jeho základných elementoch. Veľký počet základných elementov automaticky nezaručuje vyššiu kvalitu finálneho produktu. Tvorba produktu musí byť podporená zo strany procesu postupmi pre úpravu procesu pri aplikácii pre aktuálny typ projektu a jeho veľkosť.

Manažment verzií a zmien

Existencia manažmentu verzií a zmien má veľký dopad na udržiavanie produktu a jeho budúce rozširovanie. Preto je dôležité, aby počas vývojového procesu boli definované pravidlá pre tvorbu dokumentácie, archivovanie zdrojových kódov a prípadne ďalších aktivít súvisiacich s manažmentom verzií a zmien.

Analýza rizík

Analýza rizík umožňuje včasné odhalenie a potlačenie hrozby. Je kritériom pre akýkoľvek softvérový proces, ktorý sa snaží zlepšiť kvalitu produktu. Analýza rizík by mala byť súčasťou každodenných mítingov aj v období, keď je tím zamestnaný inými problémami. Inak sa projekt stáva náchylný na rizikové faktory.

Podpora kvality v RUP, MSF a XP

Kritérium	MSF	RUP	XP
Iteratívny vývoj softvéru	**Áno**	**Áno**	**Áno**
Kvalita ako cieľ	**Áno**	Nie	Nie
Nepretržitá verifikácia kvality	**Áno**	**Áno**	**Áno**
Požiadavky zákazníka	**Áno**	**Áno**	**Áno**
Zameranie na architektúru	**Áno**	**Áno**	**Áno**
Význam tímu	**Áno**	**Áno**	**Áno**
Programovanie v páre	Nie	Nie	**Áno**
Prispôsobovanie s mierou	**Áno**	**Áno**	Nie
Manažment verzií a zmien	Nie	**Áno**	Nie
Analýza rizík	**Áno**	Nie	Nie

Tab. 2 Podpora kvality v RUP, XP a MSF

Podpora kvality v Microsoft Solution Framework (MSF)

Z Tab. 2 je zrejmé, že MSF poskytuje väčšinu praktík pre zabezpečenie kvality vo vývojovom procese. Kvalita je formálne definovaná ako cieľ. Táto vlastnosť odlišuje dosť výrazne MSF od ostatných metodík. Manažment zmien a programovanie v páre nie sú súčasťou MSF, ale je ich možné jednoducho integrovať do existujúcich procesov. Podobne ako RUP aj MSF začína v prípravnej fáze projektu s definíciou požiadaviek a s úvodným návrhom architektúry a prototypov.

Metodika MSF môže byť upravovaná s ohľadom na potreby projektu, ale kľúčové elementy procesov ostávajú nemenné, čo zaručí základnú úroveň kvality. Analýza rizík je definovaná vlastným modelom, ktorý je považovaný za najkomplexnejší zo všetkých modelov používaných pri vývoji softvéru [**Error! Reference source not found.**].

Podpora kvality v Rational Unified Process (RUP)

Kvalita v RUP nie je definovaná ako cieľ, ale je vyhodnocovaná v priebehu všetkých fáz projektu a po každej iterácii. Proces je rovnako ako MSF zameraný na architektúru. V počiatočných fázach projektu vzniká jej prvá predbežná verzia. Upravovanie RUP na mieru je pomerne zložitý proces, aj keď je metodika RUP oproti MSF a XP ako jediná predávaná ako samostatný produkt s podpornými nástrojmi. Našťastie existuje množstvo predpripravených modifikácií pre rôzne projekty, ktoré vytvorili samotní vývojári RUP. Analýza rizík je prepracovanejšia ako pri XP, ale nie je natoľko sofistikovaná ako v MSF. Vlastný manažment zmien sleduje zmeny a ich dopady na projekt. Programovanie v pároch nie je súčasťou RUP, ale je možné ho jednoducho zaradiť do vývojového procesu.

Podpora kvality v Extreme Programming (XP)

Extrémne programovanie nie je formalizované do takej miery ako predchádzajúce dva procesy. XP je orientovaný na tvorbu malých a stredne veľkých projektov, preto nemá definované náročné podmienky pre manažment zmien a analýzu rizík. Krátkym časom pre vytvorenie verzií, programovanie v pároch, pevné začlenenie zákazníka do vývojového procesu a silné zameranie na tímovú prácu spôsobuje, že XP posilňuje výrazne tvorbu kvalitného softvéru. Najvhodnejším prístupom sa javí zakomponovanie niektorých praktík XP do procesov MSF alebo RUP.

Záver

Najväčšou prekážkou pri nasadzovaní praktík uvedených vyššie je, že nevieme presne merať efektívnosť validačných techník a ani nevieme povedať nakoľko správny má byť softvér. Samozrejmosťou je snaha o vytvorenie čo najlepšieho softvéru, ale dokonalosť nikdy nedosiahneme. V konečnom dôsledku, to čo môžeme považovať za dokonalé a nakoľko to je dôležité sa zásadne líši od typu projektu. Napríklad riadiaci softvér linky na spracovanie rudy má silnejšie požiadavky na kvalitu ako aplikácia na editáciu fotografií. Ak sa pokúsime vyvodiť riešenie z tejto situácie, tak dospejeme k tomu, že "dokonalosť" podchytíme definovaním vhodných atribútov kvality na začiatku projektu. A jej dosiahnutie zabezpečíme konkrétnou metodológiou vývoja softvéru. Pre malé a stredne veľké projekty môžeme odporučiť metodiku XP. S rastúcou veľkosťou projektu je vhodné siahnuť napríklad po metodike RUP a MSF v kombinácii s XP.

Podľa [**Error! Reference source not found.**] sú de facto štandardom pre zabezpečenie kvality v procese vývoja softvéru tieto princípy: *iteratívny vývoj softvéru*, *nepretržitá verifikácia kvality*, *sledovanie požiadaviek klienta*, *zameranie sa na architektúru* a *zdôrazňovanie významu tímu*. Z verifikačných metód sa prikláňame k používaniu formálnych prehliadok projektu.

Odpoveď na otázku z úvodu teda znie: dokážeme vytvoriť kvalitný softvér, ale musíme si správne stanoviť ciele a atribúty kvality, ktoré budeme vhodnou metodikou vývoja softvéru a manažmentom kvality napĺňať.

Použitá literatúra

1. KOULOUMBIS M. IT riaditelia by mali pritvrdiť. In: *Infoware,* September 2006, Číslo 9, s. 30-31.
2. ADRION R., BRANSTAD M., CHIERNIAVSKY J. Validation, Verification, and Testing of Computer Software. In: *Computing Surveys.* June 1982, Vol. 14, No. 2, p. 160-192.

3. ZUSER W., HEIL S., GRECHENIG T. Software Quality Development and Assurance in RUP, MSF and XP – A Comparative Study. In: *3-WoSQ.* May 2005, Vol. 30, Issue 4.

Annotation

Are we really able to create quality software?

The support of software quality in a software development process may be regarded under two aspects: first, by providing techniques, which support the development of high quality software and second, by providing techniques, which assure the required quality attributes in existing artifacts. Both approaches have to be combined to achieve effective and successful software engineering.

In this essay we describe processes used to implement validation, verification and testing during software development life-cycle. We bring closer look at methods and procedures which are used throughout requirements specification, analysis, design, and implementation and within application deployment. We compare three of the most industrially relevant software development process models: RUP, MSF and XP regarding their software quality support.