Testovanie a kvalita v softvérovom projekte

JOZEF HOPKO

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
jozef.hopko[zavináč]gmail[.]com

Abstrakt. Testovanie má v procese vývoja softvéru svoje pevné postavenie. Môžeme sa naň pozerať ako na jednu z metód na zabezpečenie kvality. Softvér vieme testovať na rôznych úrovniach s využitím rôznych stratégií. V tejto eseji si ukážeme základné princípy testovania, ako aj začlenenie testov do plánovania projektu. Priblížime si testovanie metódou čiernej a bielej skrinky. Zameriame sa aj na kvalitu samotných zdrojových kódov, ich písanie a revízie v menších tímoch. Pozrieme sa na V-model vývoja softvéru a plánovanie testov. Na zabezpečenie kvality softvéru v menších tímoch sa jemne inšpirujeme agilnými metódami.

Úvod

S rastúcimi možnosťami nasadenia softvérových systémov rastú aj požiadavky na ich kvalitu. Softvérové systémy sa rozšírili do všetkých oblastí priemyslu. Softvér nájdeme nielen v počítačoch ale aj hračkách, spotrebnej elektronike, automobiloch, lietadlách atď. V závislosti od typu nasadia softvéru máme aj rôzne požiadavky na jeho kvalitu.

V prípade, že sa vyskytne problém v softvéri hracej konzoly, máme po zážitku z hry, ale relatívne sa nič vážne nedeje. Stačí si nainštalovať novú záplatu a hráme sa ďalej. Nanešťastie, práve takúto situáciu mnohí z nás poznajú z oblasti operačných systémov. Tu je už pokazená nálada to najmenej, čo sa nám môže stať. Situácia sa začne vyostrovať v prípade, že prejdeme do automobilového alebo leteckého priemyslu.

Už tu nám musí byť jasné, že softvér musí spĺňať určité požiadavky a normy predtým, než ho nasadíme do praxe. Mieru splnenia týchto požiadaviek nazývame kvalita [3]. Jednou z metód na zabezpečenie kvality softvéru je práve testovanie.

Kvalita softvéru

Podľa vyššie uvedenej definície je kvalitný softvér teda taký, ktorý spĺňa požiadavky. Požiadavky na softvérový produkt sú väčšinou rôzne a môžme sa na ne pozerať z

rôznych aspektov. Napríklad v bankových systémoch sa kladie dôraz hlavne na bezpečnosť a spoľahlivosť. V priemyselnej oblasti je dôraz kladený aj na reakčný čas systému. Softvér netvorí len fungujúci vykonateľný kód, ale aj dokumentácia, používateľské príručky atď.

Požiadavky, ktoré kladieme na softvér, môžme v zásade rozdeliť na funkcionálne a nefunkcionálne. Funkcionálne požiadavky špecifikujú funkcionalitu, ktorú má systém poskytovať, správanie sa systému. Nefunkcionálne požiadavky popisujú aké vlastnosti má systém mať. Nazývame ich aj kvalitatívne požiadavky [4]. Sem patria napríklad požiadavky na:

- bezpečnosť systému
- výkonnosť systému
- udržiavateľnosť systému
- spoľahlivosť systému

Benito Zychlinsky považuje systém za udržiavateľný, ak je tento systém ľahko pochopiteľný pre programátora určeného na jeho údržbu. Taktiež musí byť jednoducho modifikovateľný v prípade nových požiadaviek na zmeny. Efektívny systém je taký, ktorý spĺňa všetky požiadavky používateľa bez zbytočného plytvania pridelenými zdrojmi [2].

Ako vidieť, niektoré nefunkcionálne požiadavky je ťažké objektívne overiť a väčšinou sú overované subjektívne. Funkcionálne požiadavky vieme testovať objektívne a v prípade jednotlivých funkcií vieme testy aj zautomatizovať.

Čo je testovanie?

Testovanie je činnosť zameraná na určenie kvality, jej zlepšenie a identifikovanie chýb a problémov.

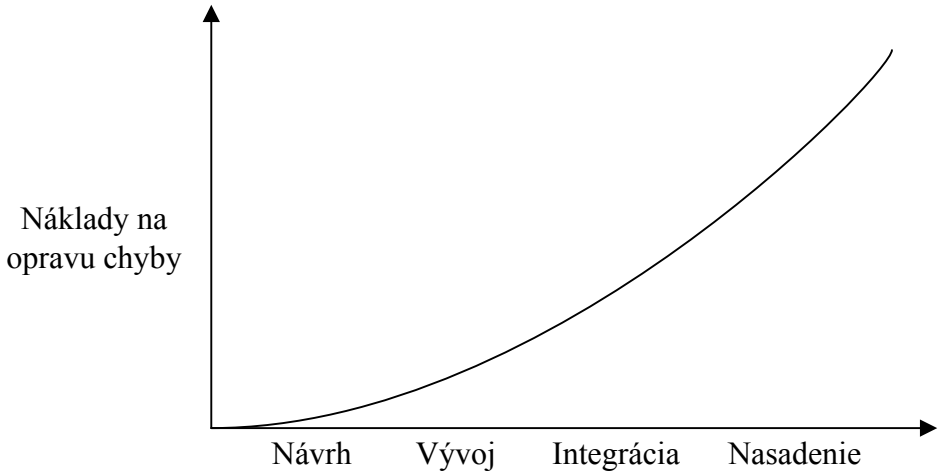
Testovanie softvéru pozostáva z dynamickej verifikácie správania sa programu nad konečnou množinou testovacích prípadov [4].

Prečo testovať?

Motivácia, prečo testovať, nemusí byť hneď jasná. Veď stačí sa len dostatočne sústrediť na to, čo vytvárame a nedať chybám šancu, aby vôbec vznikli. Potom sú testy vlastne zbytočné. Nanešťastie pre nás, situácia nie je taká jednoduchá. Pri tvorbe softvérových systémov sme rôzne limitovaní, či už finančne alebo časovo. Čas od spustenia projektu do uvedenia na trh nie je neobmedzený a ľudia sú tlačení, aby pracovali rýchlejšie. Je prirodzené, že pri tomto procese nastávajú chyby. Dôležité je však, aby sa s chybami dopredu počítalo. Taktiež je dobré mať postupy na hľadanie a odstraňovanie chýb a s tým súvisiace časové, ľudské a iné zdroje.

Ekonomický prínos testovania je taktiež nezanedbateľný. Čo sa nám spočiatku môže zdať ako plytvanie so zdrojmi, sa neskôr väčšinou ukáže ako veľmi dobrá

investícia. Ako vidno z Obr. 1, odhalenie chýb na začiatku projektu je lacnejšie, ako v neskorších fázach projektu.



Obr. 1. Náklady na opravu chýb v rôznych etapách vývoja [1].

Testovaním taktiež overujeme, či spĺňame špecifikované požiadavky a do akej miery. Znamená to, že testovanie a kvalita softvéru spolu úzko súvisia. Na testovanie sa môžme pozerať aj ako na jednu z metód , ktorou možno preukázať kvalitu softvéru.

Úrovne testovania

Z konceptuálneho hľadiska by sme mohli testovanie softvéru rozdeliť do niekoľkých hladín v závislosti od úrovne priblíženia. V zásade rozoznávame štyri úrovne testov: testy modulov, integračné testy, systémové testy a akceptačné testy. Testovanie na všetkých úrovniach má svoje opodstatnenie.

Testy modulov

Testovanie modulov má za cieľ určiť, či jednotlivé časti softvéru spĺňajú špecifikovanú funkcionalitu. V zásade pracujeme na úrovni zdrojového kódu , pričom sa testujú priamo jednotlivé funkcie alebo moduly. Na tejto úrovni vieme testy veľmi dobre automatizovať, nakoľko každá implementovaná funkcia by mala byť testovateľná. Testovanie modulov presnejšie definuje štandard IEEE 1008-87. Odhaľovaním chýb na tejto úrovni sa zaoberajú aj rôzne agilné metódy, ktoré si priblížime neskôr.

Integračné testy

Pri integračných testoch sa posúvame o jednu konceptuálnu úroveň vyššie. Prechádzame od zdrojového kódu ku jednotlivým komponentom systému. Zaujíma nás funkcionalita jednotlivých komponentov systému a ich vzájomná interakcia. Návrh integračných testov sa odvíja od architektúry systému, pričom môžme postupovať zhora - nadol, alebo zdola - nahor.

Cieľom testov je teda otestovať rozhrania jednotlivých komponentov. Taktiež zisťujeme ako komponenty systému navzájom komunikujú, ako sa šíria chybové alebo iné správy v systéme. Na tejto úrovni nás taktiež môže zaujímať vplyv chyby v jednej komponente na ostatné komponenty.

Systémové testy

Systémové testy sa koncentrujú na správanie systému ako celku. Väčšina chýb spojených s funkcionalitou systému by už mala byť odhalená predchádzajúcimi testami. Na tejto úrovni sa sústredíme na nefunkcionálne požiadavky na systém, ako napríklad spoľahlivosť, rýchlosť, bezpečnosť a pod. Taktiež sleduje spoluprácu systému s inými systémami.

Akceptačné testy

Akceptačné testy slúžia na verifikáciu, či daný softvér spĺňa požiadavky špecifikované zákazníkom. Testy sa zameriavajú na funkcionálne a nefunkcionálne stránky systému, pričom operujeme na úrovni biznis cieľov zákazníka. Na tejto úrovni verifikujeme, či sme vytvorili softvér, ktorý zákazník požadoval.

Testy sa vykonávajú v prevádzkovom prostredí zákazníka na základe pripraveného plánu.

Niektoré metódy návrhu testov

Čierna skrinka

V tomto prípade tester nevie nič o tom ako softvér funguje vo vnútri. Pozná len množinu vstupov a očakáva správne. Podobne, ako pri čiernej skrinke nevidíme teda do vnútra. Pri vytváraní testov zohľadňujeme predovšetkým špecifikáciu. Testy sú vykonané z pohľadu používateľa a jednotlivé testovacie scenáre vieme navrhnúť hneď, ako je dostupná špecifikácia [5].

Biela skrinka

Niekedy nazývané aj testovanie sklenej skrinky. Tak, ako pri sklenej skrinke vidíme dovnútra, tak aj tester pozná štruktúru systému. Pokúšame sa otestovať všetky možné cesty alebo prípady, ktoré môžu nastať. Testovaním bielej skrinky vieme odhaliť aj

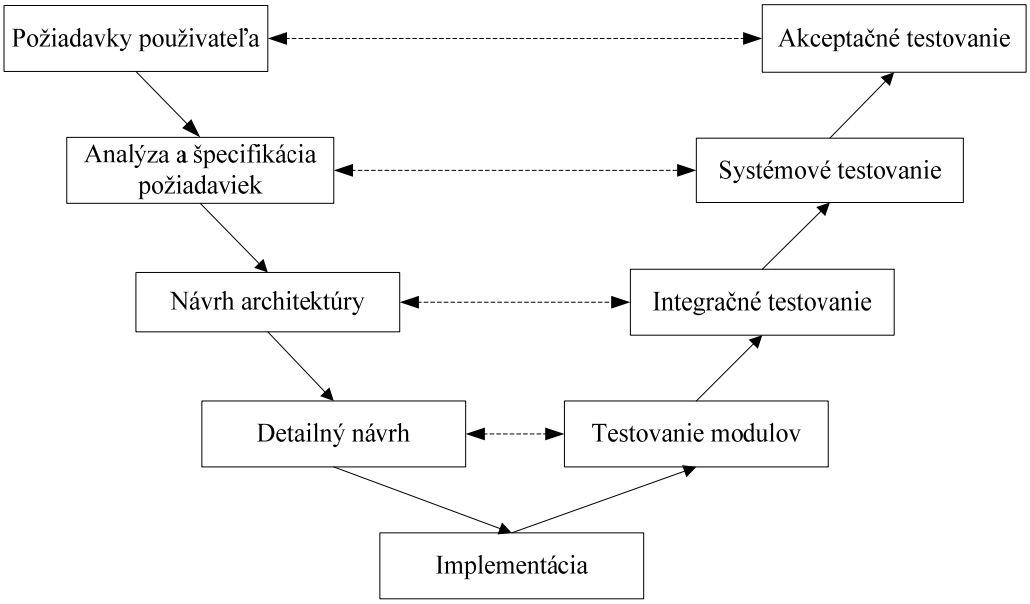
chyby v tzv. „skrytom“ kóde, ktorý sme testami čiernej skrinky nepokryli. Testovanie je však náročnejšie a drahšie [5].

Stresové testovanie

Test simuluje najvyššiu možnú záťaž systému. Je vhodný pre zložité systémy, kde nie sú ľahko predvídateľné výkonnostné charakteristiky [1]. Testujeme, ako sa systém správa napríklad pri zahltení požiadavkami a pod.

V-Model

V-Model nám poskytuje abstrakciu životného cyklu softvéru. Zachytáva jednotlivé etapy vývoja softvéru a ich mapovanie na už spomínané úrovne testovania. Na Obr. 2 možno vidieť súvis medzi jednotlivými etapami životného cyklu a ich následnosť.



Obr. 2. V-Model vývoja softvéru.

Ľavá vetva zobrazuje etapy vývoja a pravá vetva zobrazuje úrovne testovania. Prerušované čiary znázorňujú vzťah medzi etapou vývoja a jej prislúchajúcim testom. Jednotlivé etapy vývoja nie sú pre nás v tomto kontexte dôležité. Dôležité je, že každú etapu vieme otestovať.

Plán testovania

V-Model nám poskytol pohľad na vývoj a testovanie na najvyššej úrovni abstrakcie. Ak sa presunieme o jednu úroveň nižšie, zistíme, že aj jednotlivé úrovne testovania je potrebné naplánovať.

Životný cyklus testovania zahŕňa napríklad nasledujúce aktivity, pričom aktivity sa však môžu líšiť v závislosti od firmy či aplikácie [6]:

1. Plánovanie testov
2. Analýza testov
3. Návrh testov
4. Testovacie cykly
5. Vyhodnotenie testov

Plánovanie testov

V tejto fáze rozhoduje projektový manažér čo je potrebné otestovať, či sú k dispozícií potrebné zdroje a podobne. Vytvorený plán by mal identifikovať jednotlivé funkcie a vlastnosti, ktoré je nutné otestovať. Taktiež definuje typy potrebných testov, osoby zodpovedné za testovanie ako aj časový harmonogram.

Analýza testov

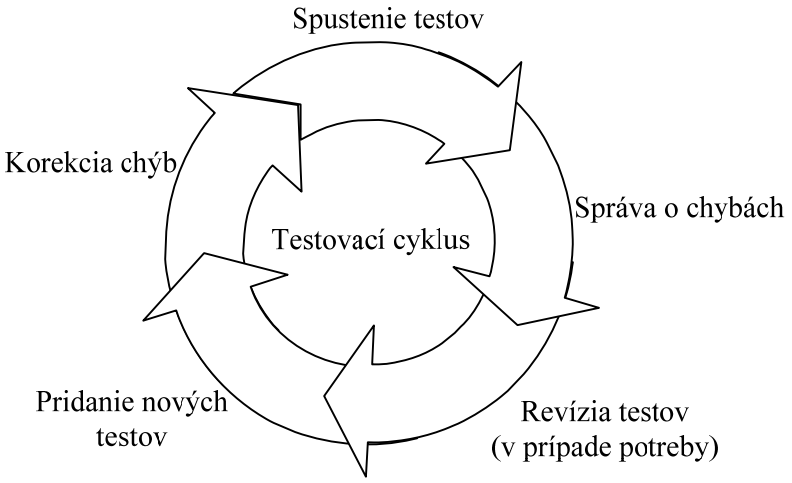
Pri analýze testov sa rozhoduje, ktoré testy vieme resp. nevieme zautomatizovať. Začíname s vytváraním testovacích scenárov. Zisťujeme, či naše testy pokrývajú všetky funkcionálne, nefunkcionálne požiadavky a špecifikácie produktu.

Návrh testov

Testovacie scenáre už máme pripravené a prechádzame ku príprave dát pre jednotlivé testy. Špecifikujeme podmienky splnenia resp. nesplnenia testov pre testovanie jednotlivých modulov.

Testovacie cykly

V tejto fáze spúšťame testy a opravujeme chyby dovtedy, pokým všetky testy neskončia bez chýb. Takto vieme výborne otestovať jednotlivé moduly systému, pričom testy možno spúšťať automaticky. Obr. 3 ilustruje priebeh testovacieho cyklu, pričom začíname spustením testov.



Obr. 3. Testovací cyklus.

Jedným z pravidiel, ktoré tu možné aplikovať, je pravidlo, ktoré nám hovorí, že každá chyba by mala mať testovací scenár. To znamená, že ku každej chybe, ktorú objavíme, vytvoríme nový testovací scenár. Tým si zabezpečíme, že v prípade, že sa chyba niekedy znovu vyskytne, vieme ju odchytiť.

Vyhodnotenie testov

Výsledky testov musia byť na konci vyhodnotené. Vo väčšine prípadov znamená úspešný test že sa softvér alebo jeho časť chovala podľa očakávania. No nie vo všetkých prípadoch sú neočakávané výsledky hneď chybami. Pred tým, ako sa rozhodne, či je potrebné chybu odstrániť, treba zvážiť závažnosť danej chyby a náklady na jej odstránenie.

Problémy testovania

To, že softvér prejde nami navrhnutými testami ešte neznamená, že je kvalitný. Jeden problém s testami je ten, že nám v zásade ukážu len tie chyby a problémy, ktoré chceme vidieť. Testy neodhalia všetky chyby, a sú len také dobré, ako ten, kto ich navrhoval. Niekedy je vecou náhody, že odhalíme určité chyby. Ako už bolo povedané, na zabezpečenie kvality potrebujeme spĺňať aj nefunkcionálne požiadavky. Pravé pri požiadavkách na kvalitu, spoľahlivosť a efektívnosť systému by som sa zastavil a bližšie pozrel na úroveň zdrojového kódu. Podľa mňa sa práve na tomto mieste rozhoduje o úspechu či neúspechu. Odmyslíme si nachvíľu výborne navrhnutú architektúru systému a prenesme sa ku zdrojovým kódom.

Fungujúci kód nestačí

Raz sa mi dostal do rúk zdrojový kód ktorý som mal skontrolovať a zoptimalizovať. Bola to jedna funkcia v jazyku C a mala približné 300 riadkov. Okrem väčšieho počtu riadkov obsahovala aj neuveriteľných 13 vnorených *if-then-else* konštrukcií. Moja motivácia čítať daný kód a zamýšľať sa nad jeho funkcionalitou bola nulová, nakoľko som nevidel koniec a zmysel tohto snaženia. Takže prišla na rad reštrukturalizácia. Výsledná funkcia mala menej ako 100 riadkov, pričom z toho približne 15 riadkov tvorila tabuľka. Algoritmus bol veľmi jednoduchý, obsahoval iba jeden cyklus a žiadne viacnásobne vnorené *if-then-else* konštrukcie.

Na mieste by bola otázka, či vlastne bolo potrebné zmeniť niečo, čo už fungovalo a bolo otestované. Z funkčného hľadiska som nič nové nevytvoril a z ekonomickej stránky by sme to mohli vidieť ako zle investovaný čas. Musím uznať, že je to v určitých prípadoch pravda. Čo sa však stane ak ku požiadavkám na softvér pridáme aj požiadavku udržiavateľnosti kódu? Situácia sa začne meniť v prospech kratšieho a lepšie štrukturovaného kódu. V prípade nových modifikácií sa nám investovaný čas začne rýchlo vyplácať. Nakoľko v pôvodnom algoritme bolo všetko napísané napevno, pri každej zmene bolo potrebne prechádzať 300 riadkov a hľadať zmeny. Daný proces väčšinou zabral minimálne hodinu. V novom kóde stačilo len zmeniť hodnoty v tabuľke a nová zmena bola obslúžená.

To, že programátori niekedy vyprodukujú obrovské funkcie s neprehľadnou štruktúrou ešte nemusí znamenať, že nevedia programovať. Niekedy sa to ani nedá inak spraviť. Problémom je postupné nabaľovanie sa zdrojového kódu s prichádzajúcimi zmenami. S každou zmenou sa štruktúra programu degeneruje. Často nie je čas rozmýšľať ako by sa to dalo spraviť lepšie, nakoľko nás obmedzujú časové a ekonomické faktory. V rozumných intervaloch by však mali byť naplánované revízie zdrojového kódu a s tým súvisiaca reštrukturalizácia. Pravé tieto dve činnosti, pokiaľ sú vykonávané dôsledne, môžu celkovú kvalitu projektu posunúť o krôčik ďalej. K lepším riešeniam niekedy stačí len iný pohľad na problémy. V nasledujúcej časti si revízie a reštrukturalizáciu kódu priblížime.

Revízie zdrojového kódu

Jedným z problémov pri vývoji softvéru je „neviditeľnosť“. Pri špecifikácií nevidíme všetky problémy a úskalia ktoré nás čakajú. Veľa detailov zbadáme až v procese implementácie. Často sú to problémy spojené s technológiou ktorú máme k dispozícií a jej obmedzeniami. V daných prípadoch je nutné implementovať rôzne záplaty, meniť a prispôsobovať už existujúci kód. Ako už bolo naznačené, tieto zásahy degenerujú kód a jeho štruktúru. Preto je vhodné robiť vo vhodných intervaloch revízie kódu. Podľa môjho názoru sú revízie kódu nevyhnutné a majú svoje technické a ekonomické opodstatnenie.

Pri revíziách je dobré okrem funkcionality sústrediť sa aj na zjednodušenie a sprehľadnenie štruktúry a hlavne zredukovanie kódu na nutné minimum. Správny kód

je podľa mňa taký, z ktorého keď zmažeme alebo zmeníme jeden riadok, neprejde testami, to znamená - stratí definovanú funkcionalitu. Prečo sa snažiť, redukovať a zjednodušovať kód? Výhod je hneď niekoľko. Menej kódu a jednoduchšia štruktúra znamená rýchlejšie revízie a jednoduchšie modifikácie. Taktiež sa v mnohých prípadoch urýchli písanie testov. Nezanedbateľná je aj redukcia nákladov spojených s údržbou softvéru, čo je väčšinou tiež jeden z aspektov kvalitného softvéru. Treba mať na pamäti, že to, čo sme vyprodukovali, budú po nás čítať aj iní.

Je výhodné , keď sa revízie zdrojového kódu robia vo dvojici. Každý vníma niečo iné a šanca odhalenia skrytých chyb je väčšia. Pričom odhalené chyby je možné hneď aj konzultovať. Takéto prechádzanie kódu je dobré aj pre programátorov, ktorí sa uistia , že tam je napísané to, čo tam vlastne chceli napísať. Osvedčilo sa mi robenie revízií zdrojových kódov v papierovej forme, nakoľko si človek spraví lepší prehľad o celom module a vzťahoch vo vnútri. Taktiež sa dajú ľahšie odhaliť opakujúce sa časti kódu. Monitor počítača a dostupné periférie predsa len poskytujú obmedzený náhľad a možnosti listovania v zdrojových kódoch a dokumentoch.

Programovanie vo dvojiciach

Táto technika patrí skôr do oblasti extrémneho programovania a nebudeme sa ňou zaoberať podrobne. Skôr by som chcel vyzdvihnúť určité myšlienky v ktorých vidím prínos pre programovanie v menších tímoch.

Ako programátori či návrhári to nemáme jednoduché. Musíme sa vyrovnať s tým, že to, čo robíme, obsahuje chyby. V praxi to znamená, že už nemôžeme veriť ani tomu, čo sami napíšeme. Práve tu prichádza na rad programovanie vo dvojici. Pokiaľ jeden programátor píše a koncentruje sa na syntaktickú správnosť, druhý kontroluje logickú a štrukturálnu stránku napísaného kódu. Prínosom daného prístupu je, že už v tomto momente eliminujeme značné množstvo chýb.

Z vlastných skúseností môžem povedať, že programovanie vo dvojici je výbornou technikou na zlepšenie kvality softvéru. Je to trochu neobvyklý prístup, no dosiahnuté výsledky sú z môjho pohľadu veľmi dobré. Vyžaduje si to však kooperáciu od oboch zúčastnených programátorov. Nie je však vždy nutné, aby obaja programátori sedeli pri jednom počítači a kontrolovali sa. V závislosti od situácie môžu pracovať samostatne, pričom dôležité sú pravidelné kontroly.

Dôležitú úlohu v tomto procese zohráva motivácia a komunikácia. Motivácia je tou hnacou silou, ktorá nás núti znovu skontrolovať napísaný kód a vymýšľať ďalšie testy. V konečnou dôsledku to môže rozhodovať o úspechu či neúspechu celého projektu, prípadne našej profesionálnej kariéry. Veľkú úlohu v párovom programovaní zohráva komunikácia, nakoľko sa úloha rieši spoločne. Všetky zistenia, problémy a riešenia treba medzi sebou konzultovať.

Záver

Testovanie je neoddeliteľnou súčasťou vývojového procesu softvéru. Má dopad na celkovú kvalitu softvéru a preto ho nemožno podceňovať. Na to kedy, ako a čo

testovať, máme definované metódy a štandardy. Testovanie však zo sebou nesie niekoľko problémov. Neodhalí napríklad všetky chyby. Na úrovni zdrojového kódu sa snažíme zvyšovať kvalitu a odstraňovať pre testy neviditeľné chyby - napríklad revíziami zdrojových kódov. Znižovať počet chýb je možné aj programovaním vo dvojici. Programátori sa navzájom kontrolujú a uisťujú sa v tom, že to, čo produkujú je správne a spĺňa to požiadavky.

Použitá literatúra

1. Beck, K.: *Extrémní programování*, Grada, 2002, ISBN 80-247-0300-9
2. Benito Zychlinski Z. and Mario Palomar A.: *A software quality assurance program through reusable code*. ACM Press, ISBN 0-89791-148-2.
3. Bieliková, M.: *Princípy softvérového inžinierstva*. STU Bratislava, ISBN 80-227-1322-8.
4. IEEE Computer Society: *Guide to Software Engineering Body of Knowledge*, SWEBOK, 2004 Version, http://www.swebok.org
5. Risley Crista: *Principles of Software Engineering*, http://www.cse.fau.edu/~maria/COURSES/CEN4010-SE/C13/, Navštívené dna 20.10.2007.
6. *Software Testing Life Cycle*: http://editorial.co.in/software/software-testing-life-cycle.php, Navštívené dňa 20.10.2007

Annotation

Testing and quality in software project

Testing is elementary part of software development process. It provides methods for software quality assurance. We can test the software on different levels with different strategies. In this essay, we learn the basic principles of testing and test planning. We take a look at black box and white box testing. We also concentrate on the quality of source codes and their revisions in smaller teams. We introduce the V-Model of software development. To assure the quality of software in smaller teams we will take some inspiration from agile methods.