KVANTITA != KVALITA

O konečný výsledok sa najviac pričiníme aj tak my sami.

Matej Pružinský

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

Abstrakt. *Z pohľadu zákazníkov je kvalita softvérového produktu priamo úmerná jeho funkcionalite. Na strane druhej, vývojári vnímajú kvalitu ako komplexnú množinu viacerých aspektov, napríklad aj prehľadnosť a efektívnosť kódu, pretože kvantita nezaručuje automaticky kvalitu. V súčasnosti existuje viacero metód, modelov a druhov testovania, ktoré by mali viesť k znižovaniu chybovosti vyvíjaného softvérového produktu a tým aj k zvyšovaniu jeho kvality. Spomeniem napríklad vývoj riadený testovaním, V-model, systémové testy, akceptačné testy a iné. Názory na ich efektívnosť sa rôznia, avšak dôležitosť testovania je dnes už nespochybniteľná. Aj samotné testovanie má však svoje úskalia. Je potrebné vedieť kedy, ako a čo testovať. Za týmto účelom tak ako pri samotnom vývoji softvérového produktu, tak aj pri testovaní existuje pojem plán. Cielom eseje je poskytnúť čitateľovi komplexnejší pohľad na spomínanú problematiku, zmapovať súčasnú situáciu a načrtnúť možné zlepšenia do budúcnosti.*

Kľúčové slová: *test, kvalita, plán*

Prečo testovať a čo je vlastne kvalita?

Kvalitu softvérového systému môžeme vnímať rôzne. Inak ju vníma programátor, inak manažér, inak zákazník. V najjednoduchšom scenári môžeme kvalitu merať mierou splnenia požiadaviek na systém. Tak ako v iných odvetviach, tak aj v informatike existuje norma umožnujúca definovať kvalitu. Takouto normou je norma ISO 9126. Táto definuje 6 vlastností kvalitného softvérového produktu (tabuľka 1) [3]:

1. Funkčnosť

2. Spoľahlivosť
3. Použiteľnosť
4. Efektívnosť
5. Udržovateľnosť
6. Prenositeľnosť

Tab. 1. Vlastnosti a čiastkové vlastnosti definované normou ISO 9126

Vlastnosť	Čiastková vlastnosť
Funkcionalita	zhoda, vhodnosť, presnosť, interoperabilita, bezpečnosť
Spoľahlivosť	vyzretosť, obnoviteľnosť, tolerancia ku chybám
Použiteľnosť	naučiteľnosť, porozumiteľnosť, funkčnosť
Efektívnosť	tímová spolupráca, využitie zdrojov
Udržovateľnosť	stabilita, analyzovateľnosť,meniteľnosť, testovateľnosť
Prenositeľnosť	inštalovateľnosť, nahraditeľnosť, prispôsobivosť

Softvérové systémy sú dnes už prirodzenou súčasťou každodenného života, uľahčujú a spríjemňujú nám rôzne životné situácie. Je však potrebné si položiť dôležitú otázku. Ako získam kvalitný softvér? Odpoveďou je testovať, testovať a ešte raz testovať. Koncoví zákazníci merajú kvalitu softvéru podľa jeho funkcionality, spoľahlivosti. Funkcionálne správny, kvalitný program by mal byť program bez chýb. Naneštastie na úplne dosiahnutie takejto vlastnosti softvéru (bezchybovosti) dnes neexistuje žiadny nástroj či metóda. Čím väčší softvér, tým viac kódu a tým aj väčšia pravdepodobnosť výskytu chýb. K dokonalému softvéru (softvéru bez chýb) sa môžeme priblížiť práve testovaním.

Vývoj softvéru je komplikovaný proces skladajúci sa z viacerych fáz, to znamená, že samotný softvér sa neustále mení. Každá takáto zmena môže so sebou priniesť nové a nové chyby, avšak nepretržitým testovaním eliminujeme výskyt takýchto chýb na minimum. Dôležité je však aj kedy odhalíme chybu. Včasným testovaním môžeme ušetriť nemalé zdroje, či už finančné alebo iné. Aj malá chyba odhalená príliš neskoro môže v konečnom dôsledku spôsobiť veľké problémy pri vývoji softvéru. Stráca sa čas a iné prostriedky potrebné na nápravu a nie je možné korektne napredovať vo vývoji softvéru. Aj preto sa dnes pri tvorbe kvalitných softvérov nevyužívá iba jedna testovacia technika či metóda, ale ich vhodná kombinácia. Podľa môjho názoru, bez ohľadu na akúkoľvek „vychytenú“ techniku testovania je vždy nutné pravidelne testovať a prehliadať kód. Prehliadky síce môžu zvýšiť náklady o 5 – 15%, ale ich výhody, ktoré z nich plynú sú značné [8].

Kód ako základný „kameň“ kvality

Základom každého softvérového systému je kód. Možno však považovať kód za základný „kameň“ kvailty? Určite áno. Kvalitné softvérové programy sú vytvárané s perspektívou do budúcnosti, s ich možným vylepšením, či prerobením na iný účel. Tejto politike je potrebné prispôsobiť aj kultúru písania kódu.

Z vlastnej skúsenosti viem, aké úskalia so sebou prináša neprehľadný kód. Zamyslime sa napríklad nad funkčným kódom, ktorý má 1000 riadkov. Kód sa dá prepísať na 200 riadkov. Je potrebné urobiť takýto prepis, keď funkcionalitu vlastne nezmením ? Určite áno, kedže pri pôvodnom kóde by nám zmena či vyhľadanie nejakej informácie trvalo niekoľkokrát dlhšie oproti novému, prehľadnejšiemu, efektívnejšiemu, kratšiemu kódu. Môže sa zdať, že čas potrebný na zefektívnenie kódu bol strávený zbytočne, keďže funkcionalita ostala rovnaká, ale opak je pravdou. Každý programátor, dokonca aj ja sám som už neraz ocenil takúto revíziu a reštruktualizáciu kódu, najmä v situaciách, kedy sa neočakávane zmenili požiadavky na celkovú funkcionalitu softvéru. Takéto zmeny prinášajú do kódu vždy niečo nové, ale častokrát sa vďaka týmto zmenám stávajú niektoré časti kódu zbytočné. Pravidelnou revíziou sú tieto časti odstraňované a tým je aj výsledna kvalita kódu vyššia, pretože kód s jednoduchou štruktúrou je ľahké modifikovať, testovať, aj prehliadať. Klasický programátor odhalí testovaním za hodinu 2 až 4 chyby, ale pri prehliadkach kódu je to 6 až 10 chýb. Aj na tejto štúdii je vidieť dôležitosť pravidelných prehliadok kódu [1].

Takéto prehliadky kódu je vhodné vykonávať vo väčších skupinách. Pri programovaní/revíziách sa najčastejšie pod pojmom skupina myslí dvojica/trojica. Každý zo skupiny má iný pohľad na kód, všíma si niečo iné. Pri správnej komunikácii a súhre v rámci skupiny je takýto prístup odhaľovania chýb veľmi efektívny. Množstvo takto odhalených chýb narastá s počtom „prehliadačov“ exponenciálne.

Samotné prehliadky sa však netešia veľkej obľube, najmä na strane autorov programov, kde existuje ešte stále akýsi strach z možných výsledkov prehliadky. Z vlastnej skúsenosti nemôžem povedať nič negatívne na pravidelné prehliadky kódov. Na jednej strane mi pomáhajú neopakovať svoje chyby a na strane druhej zvyšujú moju kultúru písania kódov.

Prehliadky nedokážu odhaliť funkcionálne chyby, ale majú svoj podiel na výslednej kvalite softvéru, taktiež pomáhajú spoznávať kód aj iným osobám ako autorom, a preto by podľa môjho názoru nemali chýbať v žiadnom procese vývoja softvéru.

Rôznorodosť testovania

Testovanie je proces, ktorý určuje a v konečnom dôsledku aj zlepšuje výslednú kvalitu a taktiež pomáha odhaľovať chyby a dalšie problémy v procese vývoja softvéru. Testovacích techník je viacero a len ich vhodnou kombináciou môžeme dosiahnuť najvyššiu kvalitu. Niektoré techniky musia byť špecialnejšie než iné v dôsledku požiadaviek. Napríklad softvér pre obedovú tabuľu v reštaurácii nemusí ukazovať správne číslo menu, kedže sa nejedná až o tak závažný problém. No na strane druhej softvér v nemocnici musí byť 100% v meraní pulzu, inak by mohlo dojsť k neštastiu.

Krok za krokom

Tvorba kódu je zvyčajne „beh na dlhšie trate“. Často sa mení a narastá Pokiaľ je kód ešte malý testovanie nepredstavuje problém. Avšak s narastajúcim kódom je testovanie každej zmeny náročnejšie a náročnejsie a v konečnom dôsledku vysoko neefektívne.

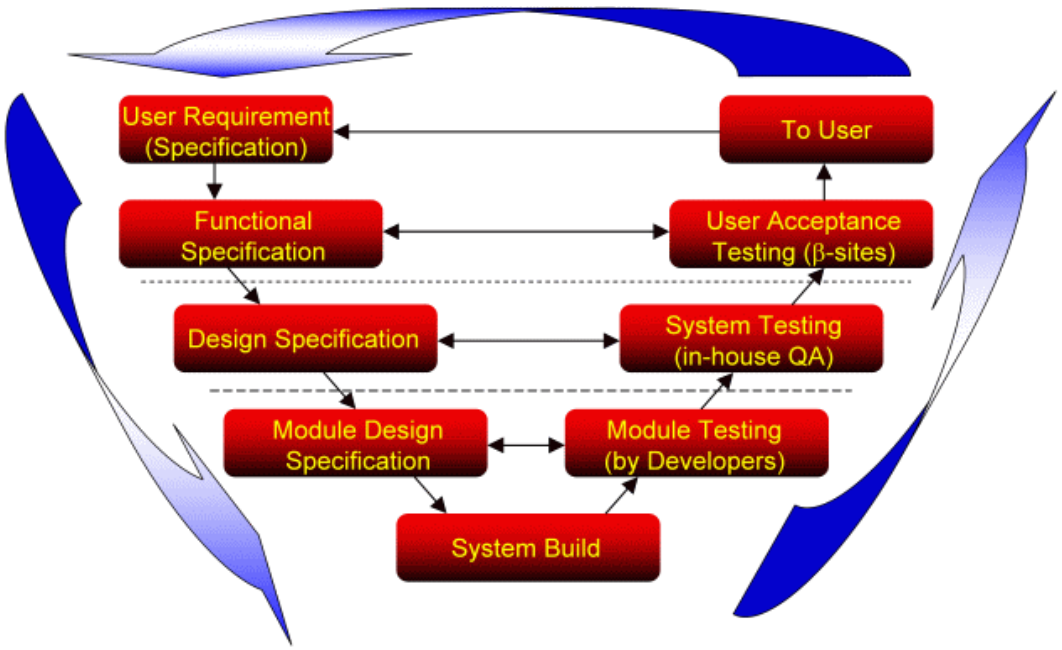
Z tohoto dôvodu vznikajú automatizované čiastkové testy, ktoré môžeme volať aj jednotkové alebo blokové. Známe je aj pomenovanie testovanie modulov. Sú to testy zamerané na jednotlivú časť kódu, jeden blok. Podľa môjho názoru je testovanie pomocou týchto testov veľmi výhodné a efektívne, kedže pri tvorbe kódu sa priebežne krok za krokom výtvárajú testy pre jednotlivé pridávané časti a nakoniec týmto spôsobom získame kompletnú sadu testov, ktoré dostatočne kvalitne testujú funkcionalitu navrhovaného systému. Ak sú jednotlivé testy navzájom dostatočne nezávislé, vieme pri kontrole presne určiť, ktorý test zlyhal, kde sa nachádza chyba. Tak ako prehliadky, tak aj blokové testy nedokážu odhaliť všetky chyby. Ich význam je však jasný, a preto by sa mali v kombinácii s inými testovacími technikami používať. Avšak realita nie je taká ružová. Tieto testy sa stretávaju taktiež s malou obľubou na strane programátorov, ktorým sa zvyčajne „nechce“, prípadne „nie je čas“ na ich tvorbu.

Ak ku spomínanej sade testov na záver pridáme ešte aj nefukcionálne testy na bezpečnosť, rýchlosť a podobne, tak dostaneme celkový systémový test, ktorým testujeme systém ako celok. Aj to je ďalšie plus, ktoré hovorí pre používanie jednotkových testov.

Životný cyklus testovania

Životný cyklus softvéru tak ako aj testovania najlepšie znázorňuje V-model na obrázku 1. Kedže sa predpokladá, že jednotlivé časti softvéru sa budú meniť, prípadne opravovať, tak aj samotné testovanie bude nutné zopakovať. Po uvedomení si týchto faktov, zistíme, že V-model nie je celkom presný a vnímať ho skôr ako kruhový model by bolo vhodnejšie.

Na ľavej strane vidíme jednotlivé etapy vývoja softvéru a na strane pravej k nim prislúchajúce úrovne testovania.



Obr. 1. V-Model vývoja softvéru [7]

Na pravej strane vidíme okrem už spomínaného systémového testovania a testovania modulov aj testovanie akcpetačné.

Takéto testovanie známe aj ako zákaznícke, je testovanie funkcionálnych a nefunkcionálnych stránok softvéru, či daný systém spĺňa zákazníkové požiadavky. Táto úroveň ukazuje, nakoľko sú splnené zákazníkové predstavy o výslednom systéme. Testovanie prebieha v prostredí zákazníka.

Teraz alebo potom ?

Prístup k programovaniu „najskôr kód, potom test“ už dnes nie je jediný smer pre vývojárov. Táto technika naďalej zostáva pre väčšinu ľudí prvoradá, avšak existujú aj iné techniky, ako napríklad TDD (“Test-Driven Development“).

Filozofiou tejto techniky je tvorba testov ešte pred samotným kódom. Na jednej strane je tento prístup považovaný ako pozitívny pre kvalitu, no na strane druhej je považovaný za kontraproduktívny. Tento prístup má určite svoje výhody, V princípe je na začiatku testovania vytvorená nejaká množina testov, ktorá nemá byť na čom testovaná, a preto skončí neúspešne. Ďalším krokom je implementácia testovanej funkcionality a opätovné spustenie testov. Tento cyklus sa opakuje až do úplného skončenia vývoja. Takýmto prístupom opäť získame na konci komplexnú množinu testov funkcionality.

Je ťažké rozhodnúť, ktorá technika písania testov je lepšia. Výber záleží hlavne na programátorovi, ktorá z nich mu viac „sedí“. Osobne by som sa rozhodol pre techniku „najskôr kód, potom test“, avšak toto rozhodnutie je skôr výsledkom zvyku než skúseností.

Ak vytvárame ako prvé testy, strávime veľa času ich návrhom, čo v konečnom dôsledku ovplyvní aj samotnú náročnosť testovania. Či v dobrom alebo zlom to záleží na

nás a našom návrhu. Veľa času však môžeme stratiť aj pri tvorbe testu až po napísaní kódu. Môže totižto nastať situácia, kedy si uvedomíme, že lepšie navrhnutý kód by nám výrazne ušetril čas pri písaní testu. Tento fakt potom vedie k prepracovaniu kódu (strata času), v horšom prípade k nevytvoreniu testu, ktorý môže v budúcnosti chýbať.

Plán testovania

Už samotný V-model ukázal, že aj životný cyklus testovania nie je náhodná záležitosť. Je zrejmé, že aj jednotlivé úrovňe testovania je potrebné naplánovať.

Životný cyklus testovania môžeme rozdeliť do niekoľkých častí [5]:

- Plánovanie testov
- Analýza testov
- Návrh testov
- Samotné testovanie
- Vyhodnotenie testov

V prvej časti má rozhodujúci vplyv spravidla manažér projektu. Ten rozhoduje, čo je potrebné testovať, aké zdroje sú k dispozícii, aké treba zaobstarať a podobne. Výsledný plán presne definuje, ktoré vlastnosti budú testované a aké techniky na testovanie budú použité. Je v ňom zahrnutý aj časový harmonogram testovania a taktiež mená osôb zodpovedných za jednotlivé fázy testovania.

Analýza testov je fáza, kedy sa rozhoduje, ktoré testy možno zautomatizovať. Plánované testy sa v tejto fáze analyzujú aj z hľadiska pokrytia funkcionality systému.

Fáza návrhu testov hovorí sama za seba. Testy už máme naplánované a v tejto časti prechádzame k ich samotnej implementácii.

V testovaní prebieha testovanie celého systému spúšťaním testov do tej doby, než všetky skončia bez chýb. V tejto časti by sme mali zaznamenávať chyby, aby sme pri upravovaní testov zabezpečili budúce vyhnutie sa takýmto chybám.

Ak by na záver testovanie nebolo vyhodnotené, bol by jeho význam nulový. Takéto vyhodnotenie nám poskytuje nie len presný obraz o správnosti funkcionality systému, ale aj prehľad o chybách ako aj o stavoch systému, ktoré zatiaľ neboli vyhodnotené ako chybové.

Čierna a biela skrinka

Testovať možno aj pomocou metódy biela (white ale taktiež glass box) alebo čierna skrinka (black box). V prípade bielej skrinky má tester prístup ku zdrojovému kódu a testuje produkt na základe neho. Vidí nielen to, čo sa deje na povrchu skrinky, ale taktiež vnútorné reakcie systému. Takýmto prístupom stráca pohľad užívateľa, avšak má možnosť lepšie odhadnúť, kde hľadať chyby a kde nie. Naopak pri testovaní pomocou čiernej skrinky sa zameriavame na vstupy a výstupy programu bez znalosti, ako je implementovaný. Systém je čiernou skrinkou, do ktorej sa nedá pozrieť, vidíme iba ako vyzerá a ako sa chová navonok. Zmyslom takejto metódy je analyzovať chovanie softvéru vzhľadom k očakávaným vlastnostiam tak, ako ho vidí užívateľ.

Medzi týmito kategóriami vznikla ešte tretia, testovanie šedej skrinky (gray box), kde máme síce apriornú informáciu o vnútorných algoritmoch produktu, ale nie takú, aby to bolo považované za testovanie bielej skrinky. Napríklad nemáme k dispozícii celý zdrojový kód, ale iba informácie o matematických princípoch použitých v systéme [6].

Testovanie testov

Kedže vývoj riadený testovaním sa čoraz viac presadzuje, vznikajú aj rôzne testovania, ktoré sa snažia preukázať zlepšenie, ktoré takýto prístup prináša, či už z časového hľadiska alebo iného.

V októbri 2002 prišli páni Hagner a Müller s výsledkami svojho experimentu ohľadom porovnania klasického prístupu oproti vývoju riadeného testovaním [4]. Ich experiment prebehol na vzorke 19 ľudí. Vo výsledku sa pozerali najmä na:

- zrozumiteľnosť kódu
- čas potrebný na vývoj
- výslednú kvalitu zdrojového kódu

Vzorka bola rozdelená do dvoch skupín, pričom obe riešili tú istú úlohu. Experimentálny vývoj pozostával z dvoch častí: z implementačnej a akceptačnej. Jedna skupina pristupovala k implementácii klasickým spôsobom „najskôr kód, potom test“, zatiaľ čo druhá skupina najskôr vyrábala testy, a potom nasledovala implementácia. Akceptačné testy Hagnera a Müllera ukázali jeden zaujímavý fakt. Napriek tomu, že čas potrebný na vývoj bol v oboch skupinách približne rovnaký a kvalita kódu tiež, sa ukázalo, že pri snahe znovu použiť existujúci kód pre ďalšie účely mala skupina využívajúca vývoj riadený testovaním značne menej chýb v kóde ako skupina využívajúca klasický prístup.

Je však nutné poznamenať, že vzorka, na ktorej sa uskutočnil experiment, bola príliš malá a skladala sa z absolventov, a preto nemôžeme považovať výsledky za 100% relevantné.

Za relevantnejší môžeme považovať experiment pána George B., ktorý svoj výskum uskutočnil na vzorke profesionálnych vývojárov [2]. 8-členné tímy rozdelil do dvojíc, kde vždy jedna dvojica využívala klasický prístup „najskôr kód, potom test“, zatiaľ čo druhá dvojica využívala vývoj riadený testovaním. Pri tomto experimente bola zároveň využitá aj technika programovanie v dvojici. Na zabezpečenie relevantnosti výstupov bolo použitých 20 testov pomocou metódy „čierna skrinka“. Dvojice využívajúce vývoj riadený testovaním prešli priemerne 18% viac testov ako dvojice využívajúce klasický prístup, avšak pracovali na vývoji o 19% dlhšie. Tento čas však môžeme považovať za „prijateľnú stratu“, kedže dvojice s klasickým prístupom síce pracovali menej, avšak relevantné výsledky v podobe testov dostatočne pokrývajúcich implementovanú funkcionalitu dosiahla iba jedna dvojica. V konečnom dôsledku bol vývoj riadený testovaním opäť úspešnejší. Jednalo sa o kategóriu pokrytie kódu testami. (takzvaný code coverage). Tento ukazovateľ je veľmi dôležitý najmä pre vývojárov z oblasti zdravotníctva, armády, či kozmonautiky, kde sa kladie asi najväčší dôraz na čo možno najkvalitnejší softvér. Štandardne je toto pokrytie niekde v rozmedzí 80% až 90%, avšak dvojice využívajúce moderný prístup dosiahli 97% pokrytie vetvení, 92% pokrytie príkazov a 98% pokrytie metód, čo je naozaj nadštandardný úspech.

Na záver bol vyhotovený ešte krátky prieskum, ktorý sa pýtal na produktívnosť, efektivitu a náročnosť vývoja riadeného testovaním. Výsledky sú znázornené v nasledujúcej tabuľke.

Tab. 2. Výsledky prieskumu o TDD

	% ÁNO	Otázka
Náročnosť	56%	Princípy sú náročné na osvojenie ?
Produktivita	87.5%	Umožňuje lepšie pochopenie požiadaviek ?
	95.8%	Redukuje množstvo riadenia ?
	50%	Znižuje čas potrebný na vývoj ?
Efektívnosť	92%	Prináša vyššiu kvalitu kódu ?
	79%	Podporuje jednoduchší dizajn?
	71%	Je technika zreteľne efektívnejšia ?

Čo si nakoniec vybrať?

Esej sa svojim obsahom venuje rôznym metodám a technikám testovania softvéru. Ako však už bolo spomenuté, zatiaľ neexistuje také testovanie, ktoré by zabezpečilo absolútnu kvalitu, odhalilo všetky chyby, či zamedzilo ich tvorbu. Keď sa však pozrieme do minulosti, je zrejmé, že v tejto oblasti sa už urobil veľký pokrok, ktorý však ešte stále nie je na konci, a podľa môjho názoru ani nikdy nebude. Na začiatku boli systémy, ktoré boli neinteraktívne, nepohodlné a ak sme spravili niečo mimo predpísaného manuálu, mohlo nás čakať nemilé prekvapenie zo strany systému. Dnes už máme systémy, pri ktorých sú chyby už značne zminimalizované a to aj vďaka testovacím technikám a metódam opísaných práve v tejto eseji. A čo bude zajtra? Budúcnosť testovania nie je naplánovaná, preto nikto s určitosťou nevie predpovedať, čo sa bude diať. Možno už zajtra príde niekto s myšlienkou natoľko prevratnou, že doterajšie spôsoby testovania začnú byť vnímané ako „smiešne“. Jedno je však isté, kedže dokonalé testovanie neexistuje neostáva nám nič iné, len vhodnou kombináciou existujúcich techník a metód minimalizovať chybovosť softvérov a tým zvyšovať ich kvalitu.

Testovanie nemožno podceňovať, pretože negatívne dôsledky takejto ľahkovážnosti môžu byť naozaj privysoké. O výslednú kvalitu by sa však nemali starať len špecializované tímy, či jeden na to určený človek. Úsilie o čo možno najkvalitnejši softvér by sa malo začať už na úrovni vedúcich manažérov, ktorí by mali nové techniky včas implementovať do procesu vývoja. Veľmi dôležitým a dnes podľa môjho názoru zanedbávaným aspektom testovania je komunikácia so zákazníkom, pretože len on vie, čo v skutočnosti naozaj chce, aj keď si to občas ani sám neuvedomuje. Avšak aj v takomto

prípade existuje riešenie a to v podobe agilného prístupu k vývoju softvéru. Princíp takéhoto prístupu spočíva v častej komunikácii v tíme a hlavne v častej komunikácii so zákazníkom.

Vo všeobecne je známe, že nové veci sa zvyknú prijímať s nevôľou. Ak sa však manažérom podarí tieto, myslím zbytočné predsudky, prekonať, je tím na najlepšej ceste k vytvoreniu kvalitného softvéru.

Použitá literatúra

1. Erdogmus, H., Morisio, M., Torchiano, M.: *On the effectivness of the test-first approach to programming*. Proceedings of the IEEE Transactions on Software Engineering, 31(1). January 2005
2. George, B.: Analysis and Quantification of Test Driven Development Approach, MS Thesis, North Carolina State University, 2002.
3. http://en.wikipedia.org/wiki/ISO_9126
4. Müller, M. M. and Hagner, O., Experiment about Test-first programming. In: *IEEE Proc. – Software*, Volume 149, Issue 5, October 2002, p. 131 – 136.
5. *Software Testing Life Cycle*: http://editorial.co.in/software/software-testing-lifecycle.php
6. http://cs.wikipedia.org/wiki/Testov%C3%A1n%C3%AD_softwaru
7. http://www.sympatec.com/JP/Quality/images/V-Model.gif
8. Watts, S. Humphrey: *Introduction to the Personal Software Process.* Addison-Wesley, Pearson Education, Inc. 2002:159-163

Annotation

Quality != Quantity

From customer point of view is the quality of software product is directly related to it's functionality. On the other side, developers perceive quality as complex set of more aspects, for example transparency and effectiveness of the code, because quantity does not automatically guarantee quality. These days there are several methods, models and types of testing, which should lead to decrease in error rates of developed software product and with that also to increase of the quality. Opinions on effectiveness differ, but the importance of testing is these days indubitable. However, testing has it's own risks. It is needed to know when testing is needed and how and what to test. For this reason, there is a plan, as for developing itself also for testing. The goal of this essay is to provide the reader with more complex view on mentioned problematics, map current situation and outline possible improvements for the future.