NAOZAJ VIEME ČO JE TESTOVANIE?

Uvedomenie si relatívnosti skutočnosti je kľúčom k pochopeniu podstaty.

Pavol Perdík

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

Abstrakt. *Kvalita softvérového produktu zohráva nesmierne dôležitú úlohu v rámci softvérového projektu. Kvalitou zvyčajne označujeme mieru splnenia požiadaviek zákazníka. V tejto eseji sa zaoberám rôznymi možnosťami sledovania plnenia požiadaviek zákazníka prostredníctvom rôznych metód testovania. Predovšetkým ale uvažujem o definícii pojmu testovanie a o jeho cieli. Všímam si výhody a nevýhody rôznych jeho metód a stratégií. Poukazujem na základné princípy testovania a uvažujem o ich dôležitosti či nevyhnutnosti. Všímam si problémy, ktoré môžu pri testovaní nastať a tiež za akých podmienok tieto testy zlyhávajú. V závere eseji pripájam aj vlastné skúsenosti s testovaním na rôzne rozsiahlych projektoch.*

Kľúčové slová: *kvalita, testovanie, princípy testovania, black-box, white-box, code inspections*

Pojem kvality v softvéri

V súčasnosti si veľkú pozornosť vo vývoji softvéru a softvéru ako takom získava kvalita. Kvalita je miera splnenia požiadaviek klienta, miera jeho spokojnosti. V dnešných dňoch je práve kvalita rozhodujúcou pre klientov pri výbere produktov. Preto je v záujme softvérových spoločností poskytovať čo najkvalitnejšie služby a produkty.

Keď sa povie kvalita, väčšinu z nás napadne správnosť programu. Nepochybne je to jeden z najdôležitejších atribútov, aby spĺňal požiadavky presne a správne, aby nemal žiadne anomálie a podobne. Treba si však uvedomiť, že to nie je ani z ďaleka jediný

atribút. Ďalšími dôležitými atribútmi sú bezpečnosť, použiteľnosť, prenosnosť, spoľahlivosť a iné [1].

V tejto eseji by som sa chcel zamerať hlavne na proces, ktorý sa týka merania kvality a ním je testovanie.

Čo je a čo nie je testovanie?

Keď sa povie testovanie, každý si pod týmto pojmom niečo predstaví. Zároveň má každý pocit, že viac menej presne vie o čom hovorí, keď hovorí o testovaní. Literatúra všeobecne definuje testovanie ako proces demonštrujúci správnu funkcionalitu softvéru alebo ako proces demonštrujúci neexistenciu chýb v systéme. Podľa [3] to však je úplne inak. Autor ostro namieta. Tvrdí, že testovanie je proces práce so softvérom, cieľom ktorého je naopak nájdenie chýb.

Myslím si, že je to iba hra so slovami. Obe, aj keď zdanlivo protichodné definície, sú správne. Totiž aj testovanie, ako proces hľadania chýb je uskutočňovaný s cieľom vyladiť program, aby mal tých chýb čo najmenej. Preto sú obe definície z veľkej časti správne. Dôležitejšie je však psychologické hľadisko, ktoré [3] vyzdvihuje.

Psychologické hľadisko testovania a motivácia

Problém definície testovania ako procesu nie je sám o sebe dôležitý z hľadiska poznania presnej definície. Dôležitý je vedľajší efekt, ktorý plynie z tejto definície, pretože definícia testovania je pre testerov vlastne jeho cieľom. Skúsme sa zamyslieť nad takouto definíciou testovania: *Testovanie je proces overujúci neprítomnosti chýb v systéme*. Tím testerov stojí pred ťažkou úlohou ako tento cieľ splniť. Sú na to dva dôvody:

- v každom softvéri sú chyby – existuje totiž paradox, že s pribúdajúcim počtom nájdených chýb sa zvyšuje rýchlosť a množstvo nájdených nových [3], preto je tento cieľ prakticky nedosiahnuteľný už z definície
- čím viac sa tím snaží, tým viac chýb nájde, a systém sa tak javí ako čím ďalej, tým viac chybný, naopak ak sa snaží menej, nájde chýb menej (v extrémnom prípade žiadnu) a systém sa javí ako funkčný, preto je pre neho podvedome lepšie nájsť chýb čo najmenej (čo je samozrejme nežiaduce)

Preto pri tejto definícii neexistuje správna a skutočná motivácia pre testerov odhaľovať chyby. Práve motivácia je jedna z najdôležitejších aspektov pri určitých metódach testovania.

Ak si ale ako definíciu stanovíme: *Proces overujúci program s cieľom nájdenia čo najväčšieho množstva najzávažnejších chýb* tak motivácia pre tím môže byť oveľa vyššia. Cieľom nie je nájsť všetky chyby (pretože je to prakticky nemožné). Zohľadňujúc ekonomické hľadisko, cieľom nie je nájsť najviac chýb, ale najmä tie najpodstatnejšie z nich. Je to nesmierne ťažká úloha, avšak oveľa ľahšie splniteľná, ako pri prvej definícii.

Kreativita

Jedným z najvýznamnejších a najpotrebnejších vlastností, ktoré sa očakávajú od vynikajúceho testera je kreativita. Hoci na jednej strane, testovacie tímy využívajú overené testovacie postupy, riadia sa tabuľkami požiadaviek, tabuľkami známych chýb, objavenie veľkej množiny chýb ostáva na testerovi. On musí vedieť výborne odhadnúť a vymyslieť nové vstupné parametre, odhadnúť slabiny softvéru pri rôznych metódach testovania, o ktorých si povieme ďalej.

Metódy testovania

Existuje množstvo rôznych metód testovania, ktoré vieme deliť podľa rôznych kritérií. Dve najzákladnejšie stratégie testovania sú *Black-box testing* a *White-box testing*.

Black-box testing

Ide o tzv. *data-driven* testovanie. Na softvér sa pozeráme v zásade z vonku ako na samostatnú funkčnú jednotku, samostatný systém. Na vstup postupne privádzame rôzne dáta a kontrolujeme výstup. Nezaujíma nás čo sa deje vo vnútri softvéru, chápeme ho ako „čiernu skrinku". K tomuto pohľadu by sme mohli zaradiť aj akceptačné testy, čo sú testy, ktoré musia spĺňať požiadavky zákazníka (dovtedy nemôžme prehlásiť, že softvér pracuje správne). Tieto akceptačné testy sú to najmenšie minimum, ktoré softvér musí splniť.

Výhodou tohto typu testovania je „odosobnenie sa od kódu". Myslím si, že tester sa pozerá na program oveľa viac z používateľského hľadiska, nie je ovplyvnený očakávaným správaním sa systému. Vie objektívne navrhovať vstupy.

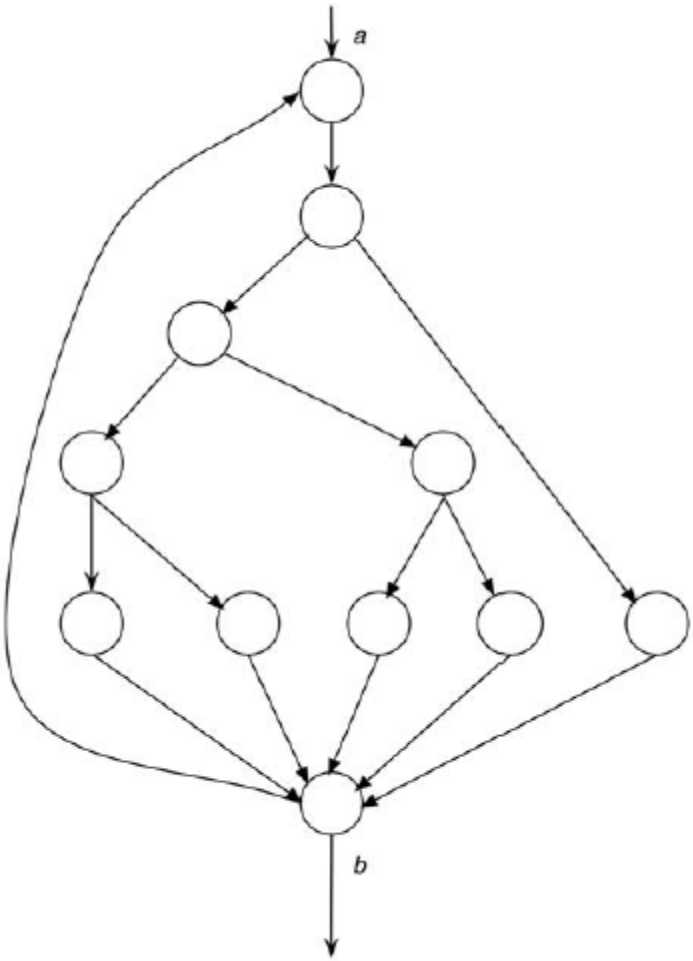
Nevýhodou tohto typu testovania je, že spadá do kategórie „exhaustive", teda vyčerpávajúcich. Podstatou je, že na dôkladné otestovanie je potrebná nesmierne veľká množina vstupov a výstupov. Dôvodom je skutočnosť, že pre niektorú množinu vstupov sa má softvér správať inak. Táto množina však môže byť malá a pri malom množstve testovacích vstupov ju teda nemusíme vôbec zohľadniť. Najčastejším druhom týchto „iných" vstupov sú neplatné, nevalidné vstupy, na ktoré by mal systém adekvátne zareagovať (napr.: pri úlohe výpočtu tan(x), vstup x má určité obmedzenia, softvér by mal byť schopný vziať tieto obmedzenia do úvahy a adekvátne na tieto vstupy reagovať). Veľa krát, ale ide aj o platné vstupy, ktoré sú málo časté. Je preto veľká pravdepodobnosť, že neboli zapracované pri návrhu a pre tieto platné vstupy dáva program neplatný resp. nesprávny výsledok. Pri tomto type testovania je takmer nemožné odhaliť tieto chyby.

Podľa môjho názoru je toto častým reálnym problémom počas vývoja softvéru. Na základe špecifikácie požiadaviek a vypracovaného návrhu (hlavne tzv. „use case") sú vytvorené akceptačné testy, ktoré presne testujú každú požiadavku klienta. Výsledný softvér teda musí úspešne zvládnuť aspoň všetky tieto akceptačné testy. Tieto testy sú teda vopred pripravené. Problém je, že spoločnosti často využívajú tieto testy ako jediný prostriedok testovania a preto výsledný softvér týmito testami prejde, aj keď je do istej miery chybný. Týmito testami prejde určite, pretože práve nimi je ohraničený vývoj softvéru, ktorý nemôže skončiť, dokým týmito testami softvér neprejde. Netvrdím, že tieto testy nemajú význam. Nesmú však byť jediným prostriedkom na overenie správnosti a teda jednej z najdôležitejších kritérií kvality softvéru.

White box testing

Pri tejto stratégii testovania ide o pozerania sa na systém zvnútra. Prechádzame systémom a snažíme sa logicky otestovať každú funkcionalitu softvéru. Ide teda o *logic-driven* testovanie.

Ani táto stratégia nie je ideálna. Problémom je obrovské množstvo kombinácií, ako môže byť program vykonaný. Na obrázku č. 1 môžete vidieť control-flow graf jednoduchého programu. Z grafu je vidieť, že už aj úplne jednoduché programy majú tendenciu mať zložitý strom kombinácií akými môže byť program vykonaný. Pri väčších programoch sú tieto stromy obrovské.



Obr. 1: Control-flow graf jednoduchého programu [3]

Obe stratégie majú svoje nevýhody. Akú stratégiu teda voliť? Ideálnym riešením je vziať si z oboch čosi.

Základné princípy testovania

Autor [3] uvádza prehľad základných pravidiel testovania, ktoré som usporiadal do nasledujúcej tabuľky.

Tab. 1: Základné princípy testovania

	Princíp
1	Dôležitá časť testovacej podmienky je definícia očakávaných výsledkov
2	Programátor by sa mal vyhýbať testovaniu svojich vlastných programov
3	Softvérová spoločnosť by nemala testovať svoje vlastné programy
4	Dôkladne preskúmať výsledky každého testu
5	Testovacie podmienky musia testovať aj neplatné, nevalidné a neočakávané vstupy, nie len tie „správne“
6	Je potrebné nie len hľadať to, čo program nerobí, ale aj to, čo robí a nemal by
7	Vyhýbať sa zahadzovaniu testov pokiaľ program nie je určený na odovzdanie
8	Nenavrhovať testy s cieľom dokázať bezchybnosť programu
9	Pravdepodobnosť nájdenia chýb v určitej sekcii programu priamo závisí na počte už objavených chýb
10	Testovanie je nesmierne kreatívna úloha

Myslím si, že nie všetky tieto základné princípy testovania sú na rovnakom stupni dôležitosti. Preto zhodnotím tie, podľa môjho názoru, najdôležitejšie.

Prvým najdôležitejším princípom je princíp číslo 2: Programátor by sa mal vyhýbať testovaniu svojich vlastných programov. Tento princíp sa odvíja od skutočnosti, že programátor vidí svoj program úplne inými očami. Vie takmer presne určiť, načo slúži ktorý kus kódu bez toho, aby ho presnejšie čítal, čím ho vlastne netestuje. Aj z psychologického hľadiska je ťažké hľadať chybu vo vlastnom kóde. Po dokončení programu má totiž programátor často krát pocit, že program pracuje správne a z princípu nechce prechádzať svoj kód odznova, zadávať testovacie podmienky a kontrolovať reakciu na vstupy.

Na základe osobnej skúsenosti som presvedčený, že tento princíp je naozaj jedným z najdôležitejších. Strávil som dlhé hodiny testovania vlastného programu, ktorý som odovzdal s vynikajúcim pocitom. Po krátkom testovaní mojím nadriadeným, ktorý ma vyviedol z omylu som už opäť sedel nad svojím programom. Bohužiaľ, prístup sa nezmenil, a opäť som ho musel testovať len ja. A znova som to nebol ja, ktorý našiel najviac chýb.

Druhým závažným problémom je časté porušovanie princípu číslo 7: Vyhýbať sa odhadzovania testov pokiaľ program nie je určený na odovzdanie. Princíp si vysvetlime na jednoduchom príklade. Majme určitú sériu testov. Niektorý z nich nájde nejakú chybu, program sa odovzdá na revíziu. Po nej je samozrejme nutné otestovať predtým zlyhané testy. A to je práve ta chyba. V skutočnosti je nutné vykonať všetky doposiaľ vykonané testy, teda aj tie, ktoré nenašli žiadnu chybu (tento krát ich nazvime „úspešné“). Zmenou funkcionality programu (opravou predošlých chýb) môže totiž dôjsť k prejaveniu sa chyby v inej časti programu, ktorú ošetruje práve jeden z týchto – predtým „úspešných“ – testov.

Human vs. Computer testing

Okrem dnes veľmi známeho testovania známych vstupov a výstupov s použitím počítača (napríklad Unit Testing), veľmi rozšíreným typom testovania je aj tzv. Human testing. Ide o testovanie programu ľuďmi priamou prácou s programovým kódom.

Medzi najvýznamnejšie patria Code inspections a Walktroughs. Ich podstatou je sedenie s viacerými testermi, ktorí spolu prechádzajú kód a diskutujú na vzniknuté problémy. Ďalšími metódami sú Desk checking a Peer reviews.

Asi tým najbežnejším je práve Code inspection. Pri Code inspections sedia typicky štyria ľudia. Jeden je moderátor, ktorý riadi beh sedenia. Moderátor by mal byť skúsený programátor, ale nemal by byť autorom testovaného programu. Ďalšími členmi sú zvyčajne programátor daného programu, návrhár (alebo analytik, dizajnér) a tester špecialista. Moderátor poskytne materiály vopred dostatočný čas pred sedením, aby si ich mohol zvyšok tímu naštudovať. Počas sedenia, obvykle trvajúcom približne 90 minút, tím prechádza riadok po riadku programu, sledujúc jeho logiku. Ostatní členovia pri nejasnostiach dávajú otázky. Okrem toho sledujú a porovnávajú jednotlivé časti kódu s obrovským zoznamom historicky známych chýb. Nájdené chyby sa zaznamenávajú, ale neriešia sa ďalej. Úlohou moderátora je spisovať chyby a po ukončení sedenia podať chyby programátorom, ktorý nájdu príčinu chýb a odstránia ich.

Myslím, že tento spôsob Human testing je jedným z najlepších spôsobov testovania, z hľadiska kvality testovania. Je síce najpracnejší. Vyžaduje si zapojiť relatívne veľké množstvo ľudí a vyžaduje veľmi dlhý čas. Za jedno sedenie je tím schopný prejsť približne 150 riadkov [3]. Ak si vezmeme ako príklad program s rádovo tisíc riadkami, je potrebných aspoň sedem takýchto sedení, každé z nich zapája štyroch ľudí. Do celkového času ešte musíme prirátať prípravu všetkých členov na sedenie. Celkový čas strávený testovaním takéhoto programu by teda mohol trvať približne 7 sedení * 4 členovia * (1,5 čas sedenia + 2 čas prípravy, odhad), teda 98 hodín. Od toho sa teda odvíja aj ekonomická otázka testovania.

Naproti tomu jednoduché testovanie samotným autorom programu sa zdá byť lacnejšie. Chyba v myšlienkovom experimente je však v tom, že autor „auto-testovaním“ nájde oveľa menej chýb, a testovanie aj po viacerých cykloch nachádza stále dosť veľa chýb. Skupinové Code inspections sú oveľa účinnejšie. Myslím si, že ak zrátame celkovú sumu človeko-hodín pri testovaní autorom programu, testovanie nebude lacnejšie než pri testovaní skupinou. Inými slovami, som presvedčený, že Code inspections je aj po ekonomickej stránke výhodnejší spôsob testovania.

Najzávažnejšie problémy testovania

Ako som už spomenul, základným, veľmi častým problémom testovania, ktorý ma priamy psychologický a v konečnom dôsledku aj ekonomický dopad je jeho definícia. Uvedomiť si, že testovaním sa pokúšame nájsť čo najviac chýb v programe.

Je známe, že nájsť všetky chyby je nemožné. Každý program ich má. Dokonca je známy paradox, že s pribúdajúcim počtom nájdených chýb v danom programe, je ten program náchylnejší na existenciu ďalších chýb. Keďže testovanie má tiež svoje ekonomické limity, rozhodujúce je teda nájsť práve tie najzávažnejšie chyby. To je úloha

pre testera alebo tím testerov. Navrhnúť, vymyslieť a použiť také postupy a metódy, ktoré odhalia práve tieto najzávažnejšie chyby. Podstatnou vlastnosťou je preto kreativita testerov, ktorý musia byť zároveň profesionálmi v danej oblasti (v danom jazyku, prostredí a podobne).

Ďalším problémom je nedodržiavanie, nedostatočné uvedomovanie si základných princípov. Nedodržiavaním niektorých z nich môže byť testovanie nie len neproduktívne, ale v podstate kontraproduktívne.

Problémom je aj testovanie programu jeho autorom. Je to porušenie základného princípu. Takéto zdanlivo ekonomickejšie testovanie však neprináša reálny úspech. Testovanie je neúspešné (nenachádza žiadne alebo neobjavuje tie najzávažnejšie chyby, samozrejme, to, že testovanie bolo neúspešné sa zistí až pri ďalšom testovaní). Dôležité je preto testovanie v tíme. Viac hláv viac rozumu.

Veľkým problémom je tiež všeobecná nedôvera v testovanie. Najmä menšie a stredné firmy podľa môjho názoru neprikladajú dostatočnú váhu testovaniu. Je to logické, že ak chcú konkurovať väčším koncernom, vynechanie alebo zanedbanie testovania je zdanlivým ekonomickým prínosom (môžu predávať lacnejšie a tým pádom konkurencieschopnejšie produkty). Vedľajší efekt sa však odrazí na ich povesti o kvalite ich produktov, z pochopiteľných dôvodov. To má za následok naspäť nižšiu konkurencieschopnosť.

Podobnú skúsenosť mám aj zo svojich školských či pracovných projektov. Na žiadnom projekte som nemal k dispozícii externého programátora a môžem potvrdiť psychologický efekt pohľadu na svoj program. Čisto zo zaujímavosti sme s kolegom vyskúšali, nech otestuje môj kód a vzišlo z toho hneď niekoľko závažných chýb. Tieto chyby nevyplývali z nedostatku mojich znalostí, napriek tomu som ich tam mal, prosto som ich tam „nevidel".

Pri mojej bakalárskej práci som využil aj automatizované testovanie pomocou Unit testov. Tieto testy boli vynikajúcou pomôckou pri vývoji. Výhodou bolo najmä automatické splnenie niekoľkých princípov testovania a to aj jedného z najdôležitejších – zachovať aj testy, ktoré nenašli žiadne chyby pri ďalších iteráciách testovania. To sa mi naozaj vyplatilo. Veľa krát, keď som opravoval nejakú chybu, ktorú odhalil napríklad test č. 7, po jej oprave „vyskočila" chyba inde. Nezachovaním týchto testov, by bol proces opravy chýb buď zbytočný alebo nekonečný.

Záver

Je nesmierne dôležité uvedomiť si čo je testovanie a čo je jeho cieľom. Jeho cieľom je nájsť v programe čo možno najviac a hlavne najzávažnejších chýb. Tento cieľ ma psychologický efekt na postoj k testovaniu a v konečnom dôsledku ho do značnej miery zefektívňuje.

Nájsť všetky chyby v programe je nemožné. Preto je nutné zapojiť kreativitu testerov s programátormi odborníkmi a využiť existujúce metódy na postupné nájdenie najhlavnejších chýb. Je úlohou vývojového tímu tieto chyby odstrániť.

Pri testovaní je nutné dodržiavať isté pravidlá, princípy. Ich nedodržiavanie je častým problémom, najmä u menších firiem.

Je nevyhnutné uvedomiť si všetky tieto postoje k testovaniu. Ak si ich uvedomíme, vieme väčšou mierou zvyšovať kvalitu našich produktov. Nie je predsa nič lepšie ako spokojný klient.

Použitá literatúra

1. Bieliková M.: Softvérové inžinierstvo - Princípy a manažment. Vydavateľstvo STU, Vazovova 5, Bratislava. 2000
2. Gill, N. S. Factors affecting effective software quality management revisited. SIGSOFT Softw. Eng. Notes 30, 2 (Mar. 2005), 1-4.
3. Myers, G. J.: The Art of Software Testing. John Wiley & Sons, Inc., New Jersey. 2004

Annotation

Do we really know what testing is?

Quality of software product plays very important role in software project. If we say quality, we often mean how we keep client requirements and client specification. In this essay I deal with different kinds of follow client requirements using different kinds of methods. First of all, I debate about the definition of testing and about its purpose. I note advantages and disadvantages of different kind of methods and strategies of testing. I point at basic testing principles and I debate about their importance. I mention problems that can occur during testing or why the testing sometimes fails. I also add my experiences with testing on different projects.