VARIL TESTER KAŠIČKU, NAŠIEL V NEJ CHYBIČKU

Len vytrvalí sa dostanú až ku koreňu problému.

Filip Hlaváček

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

Abstrakt. *Testovanie zohráva významnú úlohu pri zabezpečovaní kvality softvérového projektu. Včasným odhalením chýb možno vo vývoji softvérového produktu rýchlo a efektívne napredovať. Napriek tomu sa testerom často nedostáva patričného ocenenia a sú tŕňom v oku každého developera. V ideálnom prostredí by tester nachádzal rôznorodé chyby vznikajúce v kritických a ojedinelých situáciách. Avšak realita sa výrazne líši a tester denno-denne zápasí s množstvom včerajších problémov, chybami, ktoré vznikajú v dôsledku zlej interpretácie požiadaviek, či analýz, prípadne kontrolovaním „údajne" vyriešených chýb. To všetko má negatívny vplyv na efektivitu práce testera predovšetkým pri prieskumnom testovaní. Táto esej vrhá svetlo na prácu testera z nevšedných uhlov s cieľom poukázať na faktory, ktoré majú dopad na efektivitu práce testerov, predovšetkým pri vykonávaní manuálnych testov.*

Kľúčové slová: *test, tester, efektivita, chyba*

Úvod

Každý úspešný softvérový projekt, či malý, alebo veľký musí rýchlo a efektívne napredovať. To sme schopní dosiahnuť len v prípade, že sa budeme držať existujúcich štandardov, procesov a procedúr. Zabezpečenie softvérovej kvality si kladie za úlohu vybrať tie správne štandardy, procesy a procedúry pre daný projekt a následne overiť či boli korektne implementované. Projekt bez zabezpečenia softvérovej kvality môže síce rýchlo napredovať, ale v neskorších fázach môže jedna zabudnutá „chybička" spustiť lavínu, ktorá nenávratne pochová celý projekt.

V mojej eseji sa zameriavam práve na hľadanie chýb, teda testovanie, ktoré je hlavnou náplňou práce testera. K samotnému testovaniu existuje kvantum štúdii rozoberajúcich kedy, kde a prečo testovať, prípadne aké sú výhody testovaním riadeného vývoja, manuálneho, alebo automatického testovania. Ja sa zameriam na veľmi špecifickú oblasť, ktorou je „voľné“, na prvý pohľad náhodné a ničím neriadené testovanie.

Prieskumné testovanie

Napriek tomu, že ho niektorí označujú ako prežitok, zohráva manuálne testovanie nenahraditeľnú úlohu. Aj keď automatizované testovanie poskytuje efektívne spôsoby ako napríklad preveriť zaťažiteľnosť systému, otestovať veľké množstvá vstupov, jeho hlavnou úlohou je len uvoľniť čas pre manuálne, teda kreatívne testovanie, nie nahradiť ho. Manuálne testovanie je najčastejšie spájané s vykonávaním testovacích prípadov písaných podľa špecifikácie. Je množstvo článkov, príspevkov, návodov ako správne postupovať pri tvorbe manuálnych testov, na čo sa zamerať, a iné. Napriek tomu písanie testov si vyžaduje isté skúsenosti a hlavne prax, kde tester postupne zisťuje akým chybám sa treba vyhýbať. Tieto manuálne testy formou testovacích prípadov však nedokážu pokryť celý systém. Sú vhodné na testovanie malých jednotiek, alebo konkrétnych vlastností systému, ale pokryť nimi každú možnú situáciu je takmer nemožné, respektíve nanajvýš neefektívne.

Do hry prichádza spomínané „voľné“ testovanie. V tejto forme testovania tester intuitívne a na základe získaných vedomostí a schopností skúma určitú oblasť v snahe nájsť chyby a nedostatky. Nazvime ho *prieskumné testovanie* (PT). Táto forma testovania je bežne zaužívaná, ale nepripisuje sa jej veľká dôležitosť. Často je odstavená na bočnej koľaji a na scénu prichádza až v prípade, že sa prejdú všetky testovacie prípady a ostane čas nazvyš. Avšak z vlastných skúseností viem s istotou povedať, za istých okolností má minimálne rovnako veľký potenciál ako samotné testovanie podľa vopred pripravených testovacích prípadov. Skúsim stručne charakterizovať prieskumné testovanie do niekoľkých bodov:

- Pri PT tester využíva predovšetkým svoju intuíciu a nadobudnuté vedomosti. Preto nie je vhodné aby tester-nováčik začal práve s PT.
- Testovanie môže prebiehať bez toho aby existovali testovacie prípady. Stačí aby tester mal základné vedomosti o testovanom subjekte z analýz a špecifikácii požiadaviek.
- PT nenasleduje žiaden vopred vytýčený plán a tester má voľnosť skúmať ktorékoľvek zákutia testovaného systému, avšak ľahko sa môže stať, že „zablúdi“. Šikovný tester si rýchlo uvedomí, že pokiaľ si nezvolí stratégiu „ako“ bude postupovať pri skúmaní systému, začne efektivita PT prudko klesať.
- Jedným z problémov je monitorovať, prípadne dokumentovať postup PT. Hlásenie aktuálneho stavu PT by testera zbytočne zaťažovalo a efektivita testovania by prudko poklesla. Sám tester vie len hrubým odhadom určiť stav testovania, keďže nevie do ktorého kúta ho najbližší krok zavedie. Vzhľadom na túto skutočnosť, je vhodné PT zamerať na menšie subjekty, čo umožní súčasne viacerým testerom testovat jeden systém. Alternatívou je striedať testerov na PT,

keďže voľnosť, ktorú PT poskytuje, sa prejaví aj na odlišných postupoch jednotlivých testerov, čo môže viesť k odhaleniu nových chýb.

– Počas PT nemá tester presne definované hranice a môže sa stať, že v rozrobenom systéme narazí na časti systému, ktoré ešte nie sú nasadené. V tom prípade sa je treba vopred oboznámiť s aktuálnym stavom systému, prípadne sa daným oblastiam vyhýbať.
– Jednou z najdôležitejších požiadaviek na testera pri PT je „čistota mysle". PT testovanie a testovanie pomocou testovacích prípadov si dovolím prirovnať ku kontrole slepačích vajíčok. Testovací prípad sa dá prirovnať napríklad ku hľadaniu praskliny v škrupinke vajíčka a PT testovanie ku hľadaniu nedostatkov na vajíčkach, ktoré sú umiestnené na bežiacom páse. V oboch prípadoch sa musí tester sústrediť čisto na prácu. Každé rozptýlenie, respektíve akýkoľvek externý podnet môžu viesť k prehliadnutiu chyby. V prvom prípade však tester vie aké chyby hľadá. Stačí sa len sústrediť na daný detail a stále dookola si klásť „tú istú otázku". V prípade PT je takéto priamočiare zameranie skôr nevýhodou.
  V prípade, že sa tester priveľmi sústredí na konkrétnu vec, ľahko prehliadne iné chyby. Samozrejme, v prípade, že sa snažíme overiť funkcionalitu konkrétnej oblasti, orientujeme sa len na obmedzenú časť systému, avšak napriek tomu je dôležité byť v stave, keď vieme pohotovo sledovať správanie systému a odhaliť chyby.

Chcem byť najlepší!

Efektivita PT sa dá maximalizovať eliminovaním negatívnych faktorov pôsobiacich na testera a uvedomením si zaužívaných postupov pri PT. Predpokladám, že tak ako aj ja, tak sa aj každý tester, ktorý už PT praktizoval nájde v nasledujúcich bodoch charakterizujúcich všeobecné postupy v PT.

Vidieť nevidené

Vychádzajúc zo štúdie zaoberajúcej sa práve spomínanou tematikou [2] môžeme definovať dve stratégie postupovania pri širokospektrálnom PT - stratégia postupu skúmaním a stratégia postupu podľa dokumentácie. V prvom prípade tester skúma, teda prehľadáva systém vopred určeným spôsobom. Každý tester sa stretol minimálne s tým že pri PT postupoval podľa užívateľského rozhrania, kde testoval funkcionalitu jednotlivých sekcií podľa rozmiestnenia tlačidiel, alebo záložiek v danom okne. Ďalšie často využívané postupy, ktoré do tejto stratégie patria, sú napríklad skúmanie slabých oblastí, aspektovo orientované testovanie, funkcionálne testovanie zhora nadol, simulácia používania reálne nasadeného systému a náhodné testovanie založené na intuícii a skúsenostiach testera. V stratégii kde sa tester pri postupe drží dokumentácie je jeho skúmanie usmerňované napríklad testovacími prípadmi, zdokumentovanými zmenami v systéme a požiadavkami na systém.

V prípade, že sa pri PT má tester zamerať na špecifickú vlastnosť systému, alebo riešiť konkrétnu časť, teda testovať „detaily", vieme postupy kategorizovať tromi spôsobmi. Postup skúmaním je vhodný pri testovaní extrémov a abnormálnych situácii,

alternatívnych spôsobov manipulácie so systémom, testovaní starej funkcionality, alebo spätnej kompatibility a vzájomnej kompatibility jednotlivých prvkov systému. Postup porovnávaním si nájde uplatnenie hlavne pri testovaní komplexných ale nie striktne definovaných prvkov. Napríklad porovnávame funkcionalitu jedného prvku nasadeného do rôznych oblastí, alebo porovnávame vytvorené riešenie s iným existujúcim riešením, prípadne inou verziou. Pri PT orientovanom na vstupy, testujeme hraničné situácie, limity a obmedzenia. Ak si tester uvedomí spomínané súvislosti, dokáže rýchlo skrotiť PT a môže vykazovať lepšie výsledky ako testeri pracujúci s testovacími prípadmi.

Nie je chyba ako chyba

Eliminovať všetky negatívne vplyvy na pracovisku je nemožné. Dajú sa len obmedziť, a treba ich vedieť promptne riešiť keď už raz nastanú. Najintenzívnejší dopad môže mať na psychiku testera stres z práce kvôli nedostatku času, alebo negatívne reakcie vývojárov. Tie sa stupňujú každou nahlásenou chybou. Preto je celkom na mieste, keď si tester položí otázku „a čo s touto chybou“? Z vlastnej skúsenosti skúsim charakterizovať najčastejšie dôvody ktoré vedú k nejasnostiam, ohľadom spracovania chýb.

Testy nie sú aktuálne.

Predpokladám, že tento problém sa vyskytuje prevažne v menších spoločnostiach, kde sa najaktuálnejšie úlohy a informácie často predávajú ústne. V prípade, že došlo k zmene a neboli o nej všetci zodpovední informovaní, dochádza k zníženej efektivite v napredovaní daného projektu, k nedorozumeniam, miestami až ku konfliktom. Predstavme si, že po konzultácii so zákazníkom analytik upraví špecifikáciu, ale zabudne o tom informovať testerov. Tí následne počas testovania odhalia chyby, ktoré sú v skutočnosti len dôsledkom nových zmien. V tomto prípade stačí len aktualizovať testy a zopakovať testovanie. Stávajú sa ale aj prípady, keď vývojári pracujú s neaktuálnymi verziami špecifikácie. Následne keď im testeri komplet celú iteráciu vrátia ako zlú, dochádza zbytočne k stupňovaniu napätia medzi týmito dvoma stranami.

O chybe sa už vie, len nie je zaevidovaná v systéme.

S týmto problémom sa opäť stretávame v projektoch s menším počtom ľudí. Na spomínanej chybe sa už údajne pracuje ale nie je nikde evidovaná. Následne je tester „huckaný“ aby to nechal tak, veď ono sa to čoskoro opraví. Problém nastane v situácii, keď na rovnakú chybu narazí nezávisle iný tester, alebo sa na danú chybu zabudne.

Tester vie, že už danú chybu nahadzoval.

Situácia je pomerne zriedkavá. Manažér dá požiadavku na pretestovanie istej časti systému, ale nespomenie „detail“, v horšom prípade o ňom ani nevie, že v danej časti ešte nebola nasadená najnovšia verzia s opravenými chybami. Vývojári sa totiž väčšinou hrajú každý na svojom piesočku, a sem tam zabudnú, že aj ostatní sa chcú pohrať.

Tester sa „obáva“ reakcie vývojára.

Jeden z najčastejších konfliktov, ktoré sa v projektovom tíme objavujú sú práve konflikty medzi testermi a vývojármi. Tester je tŕňom v oku každého vývojára, čo je pochopiteľné,

keďže jeho úlohou je hľadať nedostatky vo vývojárovej práci. Veľmi rýchlo tester pochopí, že hlásenie chýb, nie je také jednoduché ako sa na prvý pohľad zdá. Samozrejme v prípade, že sa má pretestovať prihlasovanie sa do systému a používateľ nemá možnosť prihlásenia, je jednoduché vývojára presvedčiť, že tam niečo nie je v poriadku. No ak ide o chybu pri ktorej má byť, napríklad ľavé tlačidlo posunuté o dva pixle doprava, nie je jednoduché obhájiť, že máte pravdu, obzvlášť, keď jeho pozícia nie je presne špecifikovaná.

Zle interpretované analýzy, prípadne špecifikácie zo strany testera, alebo vývojára.

Aj keď je dokumentácia na základe ktorej testeri pripravujú testy a vývojári tvoria samotný produkt na vysokej úrovni, zrozumiteľná a detailná, vždy sa nájde časť ktorá sa dá interpretovať po svojom. Predstavme si situáciu kde vývojár dostane špecifikáciu podľa ktorej má naprogramovať časť systému. Bez dlhšieho zamýšľania sa nad problematikou danej oblasti, sa pustí do práce, ktorej výsledkom je program, ktorý v podstate spĺňa všetky kritéria špecifikácie. Počas toho na základe tej istej špecifikácie tester pripravuje manuálne testy. Ale v okamihu, keď sa tester pustí do testovania objaví podľa daného testu chybu, ktorú pošle vývojárovi, aby ju opravil. Ten ale chybu nechce uznať, lebo je presvedčený, že správanie sa systému je korektné. Tu nastáva zásadný problém keďže tak ako vývojár nedá dopustiť na svoj kód, tak bude tester brániť korektnosť svojho testu. Vo veľkých spoločnostiach, kde tester a vývojár nemajú možnosť si takúto situáciu vydiskutovať zoči-voči môže jej riešenie zabrať dlhé hodiny, až dni, keďže „hádať" sa s niekým písomnou formou komunikáciu je zdĺhavé.

Ticho pred búrkou

Skúsme si predstaviť nasledovnú štandardnú situáciu: „Termín ukončenia sa blíži, všetci makajú, sú nervózni a hrešia. Len v kúte testerov vládne kľud. Dobre vieme, že je to len ticho pred búrkou, neostáva nič iné, len si ho užiť. V momente, keď zavládne ticho odvedľa, a do miestnosti začnú chodiť od ucha k uchu vysmiati vývojári, vieme koľkú nám bije. Rad za radom sa začnú valiť do poštovej schránky úlohy, na ktoré je vymedzené minimum času. Treba pretestovať celý systém, ale nech to je už do zajtra, aby mali vývojári čas to opraviť."

Vo väčšine spoločností je práca testera veľmi nárazová. Manažéri zodpovední za plán, často zabúdajú na testerov, ktorí sú na chvoste produkčného cyklu. To by v zásade nebol problém, keby každý nemyslel len na seba. Málokedy berú ľudia ohľad na to čo bude po nich. Či už ide o znečisťovanie životného prostredia budúcim generáciam, alebo o smeti, ktoré ostanú na zemi po koncerte, ľudská povaha sa nezaprie ani v softvérovej produkcii. Vývojár neberie ohľad na to, že tým, že na úlohe bude robiť „o niečo" dlhšie, skracuje čas ostatným, ktorí čakajú na výsledky jeho práce. Následne sa uberie z času testerov, namiesto toho aby sa celý časový plán posunul. Pracovať pod časovým nátlakom môže byť pomerne efektívne, ale dovolím si tvrdiť, že to tak nie je v oblasti testovania. Na automatizované testy to vplyv nemá, na mysli mám manuálne testy. Je veľa spôsobov ako manuálne testovať, avšak všetky vyžadujú koncentráciu testera. Celý systém sa v podstate dá pretestovať za akýkoľvek čas, otázna je však miera efektívnosti. V prípade, že inú možnosť nemáme, treba si vedieť stanoviť priority a mať istú metodiku práce.

Pokoj na duši

Ako možno vidieť z predchádzajúcich bodov, pri všetkých vyššie spomínaných problémoch s ktorými sa môže tester stretnúť je príčinou práve ľudský faktor. Dobrý manžér sa snaží problémom predchádzať, ale nebuďme naivní, úplne eliminovať sa nedajú. Tak ako v každom programe sa nájde chyba, tak skôr či neskôr bude treba čeliť rôznym problémom. V skratke skúsim rozobrať ako sa spomínaným problémom dá predchádzať, prípadne ich riešiť.

Sú dve hlavné oblasti, ktoré musíme riešiť. Prvou ľudia, druhou je plánovanie (stratégia). V článku [1] sú tri oblasti a to ľudia, proces a organizácia. Dovolím si tvrdiť, že kategórie organizácia a proces sa dajú prerozdeliť do mnou spomínaných dvoch skupín, lebo podstatou sú opäť ľudia a ich rozhodnutia.

Ľudia

Ako aj v iných oblastiach, aj tu často zlyháva ľudský faktor. Každý kalich raz pretečie a aj tá najstabilnejšia povaha sa raz vykoľají. Konflikty medzi vývojarmi a testermi nie sú zriedkavé. Zastávam názor, že tieto problémy sa vyskytujú prevažne vo väčších firmách, kde sú možnosti komunikácie medzi zamestnancami rôznych oddelení, resp. tímov značne obmedzené. Napríklad v prípade, že vzniknú nejasnosti ohľadom nájdenej chyby, tester s vývojárom majú len obmedzené možnosti konzultovať vzniknutú situáciu a nájsť rozumné riešenie. Neosobná komunikácia formou, či už cez elektronickú poštu, alebo dokonca video hovorov nikdy úplne nenahradí komunikáciu zoči-voči, ktorá v tomto kontexte stavia malé spoločnosti do veľkej výhody. Predstava, že by som nemal možnosť nezrovnalosti konzultovať priamo s analytikom, alebo vývojárom ktorý na danom subjekte pracuje, ma desí. Vysvetliť zložitý problém je jednoduchšie keď máme k dispozícii všetky spôsoby vyjadrovania sa. Z toho vyplýva, že malé tímy, ktoré majú možnosť spolu verbálne komunikovať dokážu promptnejšie reagovať v krízových situáciach. Takáto „voľná“ forma komunikácie so sebou nesie však aj isté nevýhody, ako napríklad nekontrolované dohody, keď sa napríklad tester dohodne s vývojarom na kompromise, ktorý sa však nemusí zhodovať so zákaznikovými požiadavkami. Viacero takýchto rozhodnutí, hlavne v malých tímoch môže viesť k zlému smerovaniu celého projektu. Preto netreba podceniť schopnosť vyjadrovať sa formálne, lebo každé rozhodnutie musí byť riadne zdokumentované. Každá chyba musí byť čo najpresnejšie definovaná a zadaná v systéme.

V dobrom testerovi by mala byť štipka psychológa, aby vedel odhadnúť reakcie vývojara, aby vedel ako podať zlú správu spôsobom takým, aby vývojara nepodráždil. Veľa testerov sa učí priamo v praxi, na vlastných chybách ako pristupovať k podávaniu „zlých“ správ. Nie každý má však v povahe tú snaživosť a schopnosť prispôsobiť sa ostatným. V týchto prípadoch je vhodné ak spoločnosť ponúka školenia v riešení konflikov.

Dôležitú úlohu zohrávajú práve slávne team-building akcie, ktorých cieľom je utužovať vzťahy medzi zamestnancami. Priznám sa, že keď som sa mal po prvý krát zúčastniť takejto akcie, očakával som len prednášky o jednotlivých funkciách členov tímu, o vzájomnej spolupráci a vytýčených cieľoch. Akcie typu spoločné hostiny, výlety do hôr a raz týždenne hodina futbalu boli príjemným prekvapením. A až teraz, keď sa nad tým

všetkým zamýšľam, vidím aký pozitívny vplyv majú na prácu v tíme. Členovia tímu sa majú možnosť vzájomne spoznať a eliminovať, prípadne aspoň oslabiť pozíciu tej zafixovanej ilúzie, ktorú si jeden o druhom vytvoria v práci. Veľkou výhodou je, že testeri a vývojari majú napríklad možnosť rozberať veci z normálneho života a nemusia opäť viesť negatívne orientovanú debatu ohľadom najnovších chýb. Vhodne zvolené športové aktivity, ako futbal, tenis ponúkajú možnosť jednotlivým členom výbúriť sa, dostať zo seba to čo by normálne dusili, resp. by ich to trápilo.

Ďalším spôsobom ako predchádzať konfliktom medzi testermi a vývojármi, je utužiť ich spoluprácu už v skorých etapách projektu. Pokiaľ už vo fáze pred samotným vývojom/programovaním komunikujú testeri s vývojarmi môžu viesť nezainteresovanú debatu, keďže ešte nejde o nič osobné. To následne vedie k spolupráci aj v neskorších fázach projektu. Problém vzniká aj v dôsledku inak vytýčených cieľov a preto je vhodné nechať obe zúčastnené strany podieľať sa na ich formulácii. Vývojar sa bude snažiť implementovať najnovšie technológie, bez ohľadu na to aký to bude mať dopad na používateľské požiadavky, a tester práve naopak kladie dôraz na prívetivosť systému. V prípade, že sa však obe strany zúčasnia definovania cieľov v ranných etapách projektu, majú motiváciu uberať sa neskôr rovnakým smerom.

Plánovanie

Ako som vyššie spomínal, situácia kedy je na testera vyvíjaný nátlak aby dohnal to čo iný prepískli je pomerne častá. Riešením tejto situácie je priebežné testovanie a fexibilné plánovanie. V prípade priebežného testovania nenastane situácia, keď tester prvý krát vidí daný systém tesne pred jeho odovzdaním. Práve naopak, tester systém už dobre pozná, vie na čo sa má zamerať, kde hľadať chyby a na čo sa ma sústrediť. Preto aj v časovej tiesni, vie správne určiť priority a efektívne využiť obmedzený čas. No napriek tomu je vhodné plánovať tak, že sa ráta s omeškaním a pred samotným termínom ukončenia sa počíta s rezervou. Rezerva však musí byť dostatočne veľká aby sa s oneskorením vývoja mohol celý časový plán posunúť, bez toho aby sa skracoval vyhradený čas ostatným fázam, ako je napríklad akceptačné testovanie.

Záver

Esej v prvej časti v skratke pojednáva o prieskumnom testovaní, o jeho výhodách ale aj rizikách. Druhá časť sa zameriava na spôsoby ako zefektívniť prácu testera. Mojim cieľom je dať podnet na základe spomínaných faktov, či už testerom, vývojárom, alebo manažérom, na zamyslenie sa nad jednoduchou otázkou:

„Koľkí cestou z práce sa pýtame, mohol som niečo spraviť inak?“

Použitá literatúra

1. Cohen, C. F., Birkin, S. J., Garfield, M. J., and Webb, H. W. 2004. Managing conflict in software testing. *Commun. ACM* 47, 1 (Jan. 2004), 76-81. DOI= http://doi.acm.org/10.1145/962081.962083

2. Itkonen, J., Mantyla, M. V., and Lassenius, C. 2009. How do testers do it? An exploratory study on manual testing practices. In *Proceedings of the 2009 3rd international Symposium on Empirical Software Engineering and Measurement* (October 15 - 16, 2009). Empirical Software Engineering and Measurement. IEEE Computer Society, Washington, DC, 494-497. DOI= http://dx.doi.org/10.1109/ESEM.2009.5314240

Annotation

The tester was cooking the porridge,
there he found a bug

Testing plays an important role in software quality management. Prompt bug discovery helps the project to keep up with the schedule. Still, testers do not get enough respect from developers, who think of them as of a thorn in their eye. In an ideal environment testers would only find bugs in rare critical and extreme situations. However, in our real world, testers have to face new bugs and error, that emerge due to misunderstanding or underestimation each day. This essay points out different views at the work of testers with the goal to highlight unusual factors on tester efficacy, especially on manual testing.