Akí dobrí by mali byť testeri?

FILIP VANEK

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
xvanekf[zavináč]is[.]stuba[.]sk

Abstrakt. Testovanie patrí medzi najpodstatnejšie etapy vývoja softvérových systémov. Jeho význam ešte narastá pri tzv. iteratívnom spôsobe vývoja, kde v praxi fáza testovania jedného cyklu prebieha simultánne s fázou implementácie nasledovných cyklov. Schopnosti testujúcich tak neovplyvňujú len rýchlosť a kvalitu samotného testovania, ale aj celého vývoja systému. Na jednej strane môže byť dobrý tester pre programátora obrovskou pomocou, na strane druhej je neskúsený tester schopný v značnej miere spomaliť nielen testovanie, ale aj ostatné fázy vývoja tým, že pridáva prácu ostatným zainteresovaným osobám. Príliš veľké znalosti o systéme môžu rovnako viesť k menšej kvalite testov, preto je všeobecným pravidlom, že tester by mal do vnútra systému vidieť čo najmenej. V tejto eseji sa budem venovať práve otázke správnej miery znalostí testerov o testovaných systémoch, ako aj ich schopnosti efektívne používať rôzne podporné nástroje.

Úvod

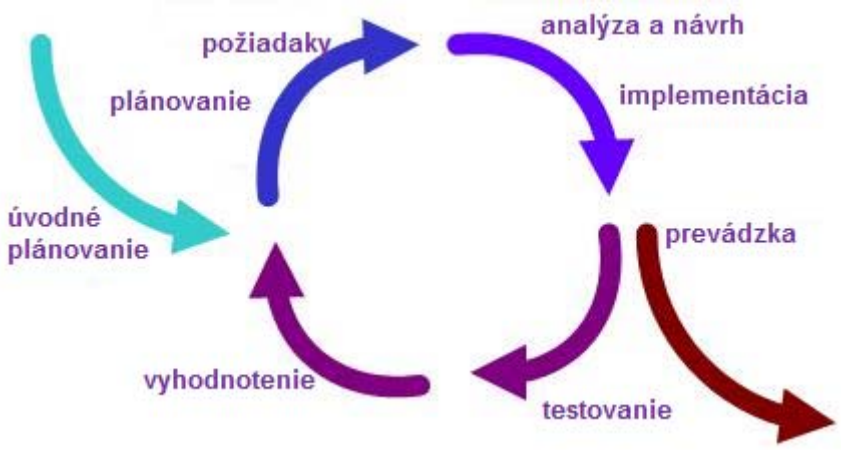
„Zasa tí testeri…“. Veta ktorú si neraz pri pohľade na svoju e-mailovú schránku povie nejeden programátor vyvíjajúci nový softvér často poriadne nahlas a s patričnými titulmi na ich osobu. Na druhej strane každý programátor potvrdí, že dobre vyškolený a spoľahlivý tester ušetrí obrovské množstvo práce. Vzniká teda otázka, ako je možné, že dvaja testeri pracujúci na tom istom projekte, s rovnakými školeniami a znalosťami o testovanom produkte, používajúci rovnaké podporné nástroje, dokážu podávať diametrálne odlišné výkony. A to nielen čo do kvantity, ale hlavne kvality práce.

Pre pozorovanie a vyhodnocovanie práce testerov je potrebné sledovať čo najväčšie množstvo z nich. Pozorovania opísané v tejto eseji boli preto získane sledovaním práce testerov vo väčšej spoločnosti zaoberajúcej sa vývojom komplexných aplikácií. Vzhľadom na množstvo vyvíjaného softvéru disponuje vlastným testovacím oddelením, v ktorom majú zamestnanci za úlohu tento softvér iba testovať. Nie sú preto zaťažovaní a rozptyľovaní inou prácou, čo pozorovanie značne zjednodušilo.

Manažment projektov softvérových a informačných systémov, október 2008, s. 1-7.

Iteratívny vývoj aplikácie

Iteratívny vývoj softvérových aplikácií a systémov patrí medzi najpoužívanejšie spôsoby prístupu k ich tvorbe. Metodológia je založená na cyklickom vývoji jednotlivých modulov ale aj aplikácie ako celku. Na obr. 1 je schematicky zobrazený životný cyklus vývoja aplikácie. Hlavnou výhodou tohto prístupu je, že v každom okamihu existuje fungujúca verzia aplikácie, ktorá môže byť nasadená do prevádzky. Spočiatku má len základnú funkcionalitu, ktorá sa ale rozširuje každou ďalšou iteráciou. Požiadavky na zmenu a teda ďalšiu iteráciu vznikajú jednak počas jednotlivých fáz cyklu alebo z vonku, kladením nových požiadaviek na funkcionalitu [4].



Obr. 1. Iteratívny vývoj softvéru [2].

Úloha testerov sa pri takomto spôsobe vývoja značne zjednodušuje. Namiesto testovania celej aplikácie stačí otestovať tie jej moduly, ktoré boli v rámci poslednej iterácie modifikované alebo pri nich existuje riziko, že by sa s takýmito modulmi mohli dostať do konfliktu. Identifikácia týchto modulov je síce za istých okolností náročná úloha, na druhej strane si vyžaduje oveľa menej zdrojov ako prípadné komplexné pretestovanie aplikácie ako celku.

V popisovanej spoločnosti funguje iterácia na pomerne jednoduchom princípe. Na začiatku cyklu analytik vypracuje analýzu pre novú verziu, a pripraví zadania pre jednotlivé zmeny, v rámci ktorých identifikuje aj zasiahnuté moduly. Programátori následne tieto zmeny implementujú, poprípade opravia chyby odhalené pri predošlej iterácii. Dobrým zvykom je upozorniť testerov na obzvlášť kritické časti aplikácie, ktoré treba otestovať dôkladnejšie, resp. označiť tie zmeny, ktoré testujúci nemá ako otestovať. Jedná sa napríklad o zmeny hlboko v infraštruktúre, ktoré sa navonok nijako neprejavujú, a preto ich testujú samotní programátori. Následne testeri za pomoci

rôznych nástrojov, jednotkových a automatizovaných testov, ale v nemalej miere aj osobným používaním aplikácie, tieto moduly otestujú, konzultujú nejasné skutočnosti a prípadne zaznamenajú odhalené chyby aplikácie.

Typy schopností testerov

Medzi základné poznatky získané pozorovaním členov testovacie oddelenia patrí fakt, že tester s väčším množstvom znalostí nie je automaticky aj lepším testerom. Práve naopak, v niektorých prípadoch sa ukázal pravý opak. V nasledujúcej časti opíšem niektoré zo základných schopností a znalostí, ktorými by mali testeri disponovať. Taktiež poukážem na prípady, keď príliš veľké znalosti testerov boli skôr na škodu ako na osoh.

Analýza a znalosť aplikácie

Jednoznačný a jediný pravdivý opis správania sa testovanej aplikácie by mal byť obsiahnutý v jej analýze, poprípade iných častiach technickej dokumentácie. Za predpokladu, že toto platí, je úloha testera ohľadom nej elementárna – musí ju byť schopný prečítať. Časy keď bola písaná v rôznych dokumentoch formou slohu, prípadne tabuliek, sú však dávno minulosťou, a tak vzniká požiadavka na znalosť štandardnej UML notácie. Bohužiaľ táto skutočnosť je často zanedbávaná. Pri nástupe nový tester buď zaklame a povie, že ju ovláda alebo si jeho informaticky vzdelaný zamestnávateľ povie, že nie je nič ľahšie ako ho túto notáciu pomerne rýchlo naučiť. Ak sa novému testerovi nemal kto po nástupe dôsledne venovať, výsledok je v praxi taký, že tester pri pohľade na analýzu aj po dlhšom čase iba tuší, čo má vlastne pred sebou a vzniká množstvo problémov. Situácia sa ešte komplikuje ak sa rôzne zmeny v systéme dohodnú rýchlo priamou komunikáciou so zákazníkom a zmeny sú do analýzy dopracované až dodatočne, keď je už testovanie dokončené. Vtedy treba testerovi vysvetliť, aký bude rozdiel medzi analýzou a skutočnou požiadavkou, avšak na to by musel poznať aspoň tú analýzu.

Pre vysokú fluktuáciu pracovnej sily a pre zaraďovanie pracovníkov na viacero projektov sa v pozorovanej firme vyskytoval pomerne často nasledovný problém. V systéme bolo treba doplniť nový, poprípade upraviť už existujúci modul, avšak vývojár, ktorý by túto úpravu za normálnych okolností vykonal, sa jej nemohol z rôznych dôvodov venovať. Prácu teda dostal iný vývojár, ktorý systém prakticky nepoznal. Pre jednoduchosť zmeny a dobre navrhnutú implementáciu systému nemusel stráviť čas komplexným študovaním aplikácie, v krátkom čase ju implementoval a následne posunul na testovanie. Napokon ale strávil niekoľkonásobne viac času komunikáciou s testerom, ktorému bolo treba pomôcť zistiť ako danú zmenu otestovať, pretože ten mal iba základnú predstavu, o čo ide, hoci systém testoval už dlhšie obdobie. Problém bol evidentný – nebol schopný čítať analýzu a riadiť sa ňou. A to aj napriek faktu, že práve to je jeho primárna úloha.

Malo by byť teda v záujme celého vývojárskeho tímu, aby aj testeri dokonale ovládali analýzu systému a poznali ho.

Vynikajúcim spôsobom ako ozrejmiť vývojárskemu tímu a teda aj testerom, čo aplikácia robí a načo slúži, sú školenia prebiehajúce v úvodných fázach rozbiehajúceho sa projektu. Získajú tu konkrétne informácie o tom, čo sa deje v reálnom svete, ak sa v systéme spustí nejaká činnosť alebo akcia. Ich pohľad na systém potom nie je zbytočne abstraktný a ako celok dáva oveľa väčší zmysel.

Na druhej strane až príliš rozsiahle skúsenosti aplikácie na strane testerov môžu spôsobiť aj problémy. Časom si napríklad zvyknú spoliehať sa na fakt, že niektoré moduly jednoducho fungujú, hľadajú skratky ako si zjednodušiť prácu a podobne. Stačí však aby programátor spravil aj malý zásah do takéhoto modulu, zabudol na ňu testera upozorniť a prípadná chyba nemusí byť odhalená.

Organizácia práce

Jeden z hlavných faktorov rozhodujúcich o efektivite práce testera je jeho schopnosť dobre si prácu zorganizovať.

Najmä v stresových situáciách, pri veľkom množstve modulov čakajúcich na pretestovanie majú viacerí testeri (ale nie len oni) tendenciu podliehať panike. Namiesto toho aby zvýšili svoju efektivitu, dochádza k presnému opaku a to aj napriek prípadnému predĺženiu pracovnej doby v rámci krízového režimu. Dá sa doslova povedať, že nevedia čím skôr začať, a preto začnú naraz pracovať na všetkých testoch bez toho, aby ich postupne ukončovali.

Pri sledovaní práce testerov som zaznamenal aj skutočnosť, že niektorí sa opakovane pýtajú ostatných členov vývojového týmu tie isté otázky. Nedalo sa s istotou povedať, že by sa opakovali len zložitejšie otázky týkajúce sa komplexných problémov, elementárne problémy mali totiž takisto významné zastúpenie. Rovnako dĺžka praxe testujúceho sa v tejto problematike ukázala irelevantná. Dôvod kladenia otázok bol pritom úplne jednoduchý. Rovnaké otázky totiž opakovali len tí členovia testovacieho oddelenia, ktorí neboli zvyknutí používať nástenku, elektronické poznámky alebo inú formu osobnej dokumentácie. Aj takáto maličkosť pritom dokázala v značnej miere spomaľovať celý vývojový tím ktorý musel neustále odpovedať na opakujúce sa otázky, pretože testy bolo treba vykonať bez ohľadu na to, čo si o prístupe testerov osobne mysleli.

Problematickou sa ukázala aj otázka vzájomnej koordinácie testerov. Jednotlivé testy si totiž prideľovali náhodne. Pri prideľovaní vôbec nebola zohľadňovaná komplikovanosť jednotlivých testov, ani fakt, že každý tester má rôzne znalosti jednotlivých systémov. Výsledkom bolo, že tester testoval systém, ktorý nikdy netestoval a nepozná ako funguje a to aj napriek tomu, že iný tester, ktorý s ním skúsenosti má, v tej dobe nemal čo robiť, poprípade mal len minimum práce.

Najväčším prekvapením bol rozhodne fakt, že väčšina z testerov sa snažila vykonať čo najväčší počet testov, mali totiž zavedený vlastný bodovací systém na sledovanie práce jednotlivcov. Takýto spôsob riadenia a organizácie sa ukázal ako maximálne neefektívny, pretože značná časť testerov sa snažila rýchlo vykonať väčší počet jednoduchších testov. Kritické a komplikované testy tak zostávali väčšinou neriešené až dovtedy, kým neboli upozornení, že meškajú. Navyše niektoré testy neboli vykonané dôkladne, aby tak mohli rýchlejšie získať bod do hodnotenia.

Podporné nástroje

Testovanie a hlásenie chýb prebieha v praxi za podpory rôznych pomocných nástrojov, či už štandardných alebo takých, ktoré si vývojársky tím sám vytvoril pre vlastné potreby. Výsledkom testovania sú potom z pohľadu tímu štandardizované hlásenia, do ktorých sa vypĺňajú rôzne skutočnosti a fakty zistené pri procese testovania. Jednotlivé hlásenia rôznych chýb by teda mali na prvý pohľad vyzerať rovnako.

Realita je však iná. Traja rôzni testeri postupne zahlásili novoobjavené chyby v systéme. Prvý tester zaznamená objavenú chybu pomocou vnútrofiremného systému na to určeného. Správne vyplnené hlásenie obsahuje všetky potrebné informácie na došetrenie a opravu chyby programátorom. Pri dobre navrhnutej vyvíjanej aplikácii, ktorá generuje kvalitné kontrolné záznamy kopírované testerom priamo do hlásenia, stačí často jediný pohľad a vie, kde nastala chyba. Navyše tester si dal námahu a pomocou malého podporného nástroja našiel v logoch vstupné parametre volania testovanej činnosti, ktoré zákazník a teda ani testujúci nevidí, programátorovi ale pomôžu pri hľadaní chyby.

Report druhého testera bol vyplnený v tom istom formulári, bohužiaľ bez akýchkoľvek informácií. K hláseniu bol iba priložený dokument so záberom obrazovky aplikácie a textom „toto mi nejde“. Namiesto efektívneho riešenia chyby tak programátor stráca čas zisťovaním, čo vlastne tester robil a čo mu nejde. Spôsobené to bolo neschopnosťou testera používať spomínaný nástroj na hlásenie chýb. Doslova nevedel čo kam a ako napísať, tak pre istotu nenapísal nič.

Naopak tretí tester identifikoval problém až do tej miery, že programátorovi presne napísal, kde a akú má chybu. Využil tak svoje značné znalosti aplikácie a rôznych nástrojov, ktoré si programátori vyvinuli na rýchlejšiu identifikáciu a riešenie chýb. Tieto znalosti získal tak, že sám študoval z dokumentácie, ako je aplikácia implementovaná a ako tieto nástroje fungujú, poprípade mu to ukázal niekto z programátorov. Problémom je, že na takto presnú identifikáciu chyby spotreboval oveľa viac času, ako by potreboval programátor, ktorý dané nástroje ovláda a vie ich efektívne používať. Zbytočne tak spomalil resp. odložil testovanie iných modulov, čo je vzhľadom na dodržiavanie časového plánu vývoja celej aplikácie nevýhoda.

Treba však veľmi dobre rozhodnúť, ktoré z existujúcich nástrojov by testeri mali byť schopní používať. Následne ich treba v používaní týchto nástrojov dostatočne vyškoliť a neustále ich nabádať k dodržiavaniu pravidiel ich používania. Členovia vývojového tímu by si mali uvedomiť, že aj dobre mienená rada alebo ukážka typu „takto to robíme my“ nemusí priniesť želaný efekt. Je možné, že pomôže vyriešiť aktuálny problém, ale z dlhodobého hľadiska môže byť skôr na škodu ako na osoh.

Veľkosť tímu

Táto esej je založená na pozorovaní práce testerov vo veľkej firme, iba zdôvodu možnosti pozorovať viacerých testerov naraz. Takmer všetky opísané skutočnosti sú rovnako pravdivé aj pre testerov pracujúcich na malých projektoch len s niekoľkými ďalšími členmi.

Pre takéto tímy často platí, že tester je len jeden a ak treba, tak s testovaním pomáhajú ostatní členovia tímu. V tom prípade je práve na ňom aby testovanie koordinoval a riadil, keďže je zaň zodpovedný a v tíme je jediný, kto má s testovaním danej aplikácie väčšie skúsenosti.

Komunikácia medzi členmi takéhoto tímu býva často menej formálna, takže je nevyhnutná vyššia miera komunikatívnosti testera. Aj keď snaha zachovať formálnu komunikáciu za účelom dokumentovania činnosti zostáva, mnohé veci sa riešia iba ústne.

Osobnosť

Na tvorbe softvérových systémov sa podieľa veľké množstvo ľudí zameriavajúcich sa na jednotlivé fázy vývoja. Testovanie softvéru je dôležitým prostriedkom na zaručenie kvality konečného produktu. Napriek existencii mnohých pomocných nástrojov a automatizovaných testov, výsledok závisí od spolupráce všetkých členov vývojového týmu. Napriek tomu, ciele každého člena týmu sa môžu líšiť. Skutočnosť že každý člen tímu má iný typ osobnosti, ciele a spôsob práce, môže rýchlosť práce výrazne ovplyvniť ako k lepšiemu, tak aj k horšiemu [5].

Najpodstatnejšou časťou osobnosti testera, vzhľadom na jeho prácu, sa ukázala byť správna miera schopnosti komunikácie. Tu sa v plnej miere ukazuje rozdielnosť jednotlivých osobností a pováh. Niektorí testeri sa miestami až báli napísať analytikovi alebo programátorovi, pretože si neboli istí, či sa nechcú opýtať nezmyselnú otázku, alebo svoju otázku nevedeli sformulovať. Opačným extrémom boli testeri, komunikujúci až priveľmi. Iba zlomok ich komunikácie sa totiž týkal práce a zbytočne spomaľovali ostatných otázkami typu „ako sa máš?“.

Nikdy, poprípade len v ojedinelých prípadoch platí, že systém testuje len jedna osoba. Tak ako každý iný tím, aj tím testerov potrebuje jasné vedenie a organizáciu. V mnohých prípadoch je však testovacie oddelenie vedené príliž neformálnym spôsobom. Minimálne niektorí členovia tímu by teda mali mať schopnosť viesť a usmerňovať ostatných.

Testeri sa musia často vyrovnať ešte s ďalším problémom. V rámci vývojového cyklu sú to totiž práve oni, kto ukončuje prácu a teda oni, kto musí dodržať finálny dátum ukončenia práce. Nedostatok času na testovanie spôsobuje neustály stres, s ktorým sa musia byť schopní vysporiadať [1].

Záver

Pri pozorovaní práce členov testovacieho oddelenia firmy zaoberajúcej sa vývojom aplikácii som sa zameral na sledovanie súvislostí medzi ich znalosťami, ktorými disponujú a kvalitou ich práce. Vďaka sledovaniu väčšieho počtu testerov bolo toto pozorovanie o čosi jednoduchšie, avšak takmer všetky zistené skutočnosti sa vyskytujú aj u testerov pracujúcich v menších tímoch zložených z iba niekoľkých jednotlivcov.

Pre všetkých pozorovaných testerov platil základný a elementárny fakt. Ich celková efektivita a kvalita vykonanej práce je priamo úmerná dĺžke ich praxe, ale bola len veľmi málo závislá od množstva ich znalostí. Postupne sa totiž učia spolu s ostatnými členmi tímu zvyšovať svoju efektivitu a tak zvyšujú svoj výkon. Viacej znalostí ako ich minimálne potrebné množstvo ale nezaručuje aj lepšie testovanie [3].

Očakávaným záverom je, že neskúsený tester spôsobuje celému vývojárskemu tímu problémy. Neskúsenosť je samozrejme tolerovaná určité obdobie, počas ktorého sa ostatní členovia tímu snažia byť trpezliví a vyškoliť ho. Ak je však aj po dlhšom období tester neefektívny a robí chyby, znamená to, že sa niekde stala chyba.

Na druhej strane je faktom že testeri majú jedinú úlohu – testovať. Ak sa tester pýta na vnútornú implementáciu systému, vie ako funguje a snaží sa odhaľovať chyby študovaním kódu, je vhodné zamyslieť sa nad náplňou jeho práce. Zbytočne totiž spomaľuje testovanie a možno by bolo dobré skúsiť ho začleniť do implementačnej fázy vývoja. Samozrejme za okolnosti že má potrebné vzdelanie resp. schopnosti.

Použitá literatúra

1. Cohen C. F., Birkin S. J., Garfield M. J., Webb H. W.: Managing conflict in software testing, *Communications of the ACM*, Vol. 47, No. 1 (2004) 76-81
2. http://en.wikipedia.org/wiki/Image:Iterative_development_model_V2.jpg
3. Huang L., Holcombe M.: How good are your testers? An assessment of testing ability, *Testing: Academic and Industrial Conference Practice and Research Techniques*. 2007
4. Lambers L., Ehrig H., Mariani L., Pezzè M.: Iterative model-driven development of adaptable service-based applications. *Proceedings of the twenty-second IEEE/ACM international conference on Automated software engineering.* 2007
5. Tjosvold D.: The Conflict Positive Organization. Addison-Wesley Publishing Co., Reading, MA, 1992.

Annotation

How good should testers be?

Testing is one of the most important stages of software development. Its importance is even bigger in iterative development as testing phase of one cycle takes place simultaneously with the implementation phase of the following cycles. Thus, the abilities of testers affect not only speed and quality of testing, but also development of whole speed. Good tester can be of great help for programmer, but on the other hand, bad tester is able to slow down not only testing but also whole development of the system by adding work to other people working with him. Too much knowledge of the system may also lead to lower quality of testing, so it is general rule that the testers shouldn't know how the core of the system works. In this essay, I will look into the issue of right degree of testers knowledge about tested systems, as well as their ability to effectively various support tools.