Kvalitné testovanie – kľúč k dobrému projektu, ako ho však zabezpečiť?

DUŠAN ZAHORANSKÝ

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

Abstrakt. Tvorba projektov malých alebo veľkých, softvérových alebo iných zo sebou prináša aj vznik chýb. Programátori tvoria kód, súčasne aj chyby, jednoducho sa tomu nedá zabrániť. K odhaleniu a minimalizácií nežiaducich a nepredvídaných chýb aplikácie alebo k posúdeniu stavu projektu slúži testovanie. Existuje množstvo spôsobov a plánov ako efektívne a kvalitne testovať. Nie vždy sa však plány podarí zrealizovať. Často je náročné zabezpečiť kvalifikovaný personál alebo sa na splnenie testovacích plánov nezvýši dostatok času. Testovacie plány môžu tiež ľahko stroskotať, ak sa nevyužije vhodná metóda, prípadne nástroj. Ako vybrať vhodný spôsob testovania? Akých testerov vybrať do tímu? Kedy testovanie zapojiť do projektu? Na tieto otázky sa teraz skúsim pozrieť bližšie. Pokúsim sa odhaliť niektoré chybné rozhodnutia a postupy, ktoré sa pri testovaní stávajú a situácie, kedy sa testovanie môže stať neefektívnym a zbytočne zdržiavať priebeh projektu.

Úvod, alebo ako to zvyčajne funguje

A. Bertolino vo svojej publikácií [1] formulovala testovanie softvéru ako široký pojem, ktorý zahŕňa rôzne aktivity od testov malých kusov kódu programátorom, až po validáciu veľkých informačných systémov zákazníkom. Pravdivosť tohto tvrdenia sa dá len veľmi ťažko spochybniť. S postupujúcim časom sa projekty stávajú stále komplexnejšie, väčšie a náročnejšie na zdroje. Tento trend ovplyvňuje aj testovanie, ktoré predstavuje neustále sofistikovanejší a časovo namáhavejší proces.

Moderné vysoko-úrovňové programovacie jazyky pomáhajú predikcii na úrovní kompilácie, na druhej strane programátori nadobúdajú dojem, že chyby v kóde nemajú. To je dôvodom prečo sa menej venujú testovaniu metódy, alebo fragmentu kódu, ktorý práve vytvorili. Táto ľahostajnosť však podnecuje častejší vznik logických chýb aplikácie, ktoré nedokáže odhaliť ani kompilátor vysoko-úrovňového jazyka.

V rozsiahlych projektoch sa niektoré časti vytváraného systému stávajú ťažko testovateľnými. Neraz dochádza k integrácií systémov, ktoré bežia v inej časti sveta

ako vyvíjaný produkt, vtedy neexistuje možnosť, ako s nimi otestovať komunikáciu. Prvý reálny integračný test sa v mnohých prípadoch udeje až počas ostrej prevádzky systému.

Architektúra systémov je dnes bežne tvorená z niekoľkých vrstiev. Vo väčších, no dnes už aj v menších projektoch je obvykle architektúra trojvrstvová. Programátor bežne pracuje iba na jednej z nich a o zvyšných nemá veľký prehľad. Spraví prácu, ktorú mal „u seba“ na svojej vrstve projektu. Funkcionalita ktorú práve implementoval volá metódu z nižšej vrstvy, napríklad z vrstvy databázovej. Pri jednoduchom testovaní, ktoré sám narýchlo vymyslel mu program nespadne na žiadnej chybe. Spokojne sa tvári, že práca je hotová a jeho kód funguje. Metódu databázovej vrstvy však volá s inými parametrami, metóda zbehne, ale dáta ktoré práve zapíše do databázy sú iné ako očakával. Takto jednoducho vznikla chyba v projekte, o ktorej nikto nevie. Ako to celé skončí a akú úlohu zohrá testovanie?

Pokiaľ má programátor šťastie, tak manažér projektu vytvoril testovací plán, v ktorom nezabudol, že takéto chyby pri vývoji môžu nastať. Zostavil tým testerov, ktorí naplánované testy vykonajú, chyba sa odhalí a projekt je zachránený. V odlišnom scenári takéto šťastie nemá. Chybu si ešte hodnú chvíľu nikto nevšimne. Vytvorený systém sa dostane do prevádzky, kde sa nahromadí množstvo chybných dát. Keď funkcionalita systému pracujúca s týmito dátami prestane fungovať správne, niekto chybu odhalí a nahlási vývojovému tímu. V projekte nastane panika a v niekoľkých zdĺhavých iteráciách sa budú získavať dáta z prevádzkového prostredia, kým sa nezistí, ako sa chybné dáta vytvorili. Následkom je minimálne niekoľko dňová nefunkčnosť časti systému a nespokojnosť zákazníka.

Obrazom moderných trendov vo vývoji softvéru a následnou absenciou testovacích plánov sa snažím poukázať na základný zmysel testovania, ktorý podľa [3] spočíva v znižovaní rizika vzniku neznámych problémov v produkte. Vzniku takýchto neočakávaných chýb sa dá vyhnúť výberom vhodných spôsobov testovania v závislosti od čŕt projektu a snažiť sa nepodceniť žiaden z jeho aspektov.

Testovanie som nemal nikdy možnosť vnímať z pohľadu testera alebo manažéra, ktorý pripravuje testovacie plány. Napriek tomu si myslím, že často krát práve programátor vidí najlepšie, kedy je ktorý spôsob testovania účinný a koľko chýb odhalil. Mnohokrát keď sa chyba objaví až počas prevádzky systému je mi jasné, že nejaký spôsob testovania nebol vykonaný dôsledne, alebo dokonca vôbec.

Spôsoby testovania

Testovanie by malo byť rozvrhnuté spôsobom, aby sa v rôznych štádiách zameralo na rozličné ciele [3]. Existuje množstvo spôsobov ako testovať. Nie je problém vyhľadať informácie o tom ako fungujú a využiť ich v prospech projektu. Niekedy je problémom práve fakt, aké spôsoby testovania skombinovať. Najlepšou voľbou je zostaviť niekoľko spôsobov testovania, ktorými sa pokryje viacero aspektov projektu, otestuje sa každé zabudnuté miesto, ktoré by mohlo spôsobovať chyby .

Pri výbere rôznych testov je potrebné zvážiť viacero špecifík projektu, ako sú:

1. Požiadavky na projekt
2. Veľkosť projektu
3. Zložitosť projektu
4. Spôsob vývoja projektu
5. Toleranciu zákazníka k chybám
6. Závažnosť chýb
7. Životný cyklus projektu
8. Schopnosti programátorov
9. Schopnosti testerov

Nech už je projekt akýkoľvek niektoré testy by v projekte určite absentovať nemali. Podľa môjho názoru medzi ne určite patrí jednotkové testovanie a záťažové testovanie. Niektorým sa na druhej strane vyhnúť nedá, ako napríklad akceptačné testovanie.

Jednotkové testovanie (angl. Unit testing)

Ak by som mal použiť iba jediný spôsob testovania, kvôli nedostatku času alebo iných prostriedkov, bolo by to jednotkové testovanie. Jednotkové testovanie je najlacnejšia spätná väzba akú môže vôbec programátor dostať [3]. Používa sa po implementácii novej funkcionality, čo môže byť metóda, ale aj blok, či dokonca pár riadkov kódu. Vytvorenie kvalitných jednotkových testov je úlohou programátora samotného. Kľúčom je pokryť testom celý kód, nie iba niekoľko možných variantov správania aplikácie.

Záťažové testovanie

Záťažové testy síce primárne slúžia na otestovanie stanovenej rýchlosti odozvy systému, alebo rýchlosti spracovania dát, napriek tomu práve ony môžu odhaliť napohľad neviditeľné a najzákernejšie chyby implementácie. Preto si myslím, že nie len projekty veľkého rozsahu, ale aj menšie by mali aspoň čiastočne takéto testovanie vyskúšať. Stačí k tomu málo, nejaký nástroj na hromadné spúšťanie vstupov a dobre spracovaná odozva z týchto testov, napríklad štatistika vytvorená profilovacím nástrojom a tieto chyby môžeme ľahko odhaliť. Najčastejšie medzi ne patria úniky pamäte, nekonečné cykly, alebo uviaznutia na mŕtvom bode (angl. deadlocks).

Akceptačné testovanie

Akceptačné testy sú prirodzenou súčasťou minimálne pri projektoch väčšieho rozsahu. Tieto testy majú zaručiť, že projekt spĺňa špecifikáciu požiadaviek a v mnohých prípadoch rozhodnú, či sa daná verzia produktu môže nasadiť do prostredia prevádzky. Akceptačné testovanie prebieha po ukončení projektu, jeho etapy, alebo iterácie kde bola zmenená funkcionalita. Akceptačné testy sú často inicializované a vykonávané

u zákazníka. Zistenie chyby v tomto testovaní predstavuje zdržanie nasadenia projektu, keďže ich obvykle nie je možné vykonávať súčasne s vývojom. Zákazník si často krát neželá aby bola vykonávaná akákoľvek zmena na projekte počas priebehu testov. Akceptačné testy zjednodušene predstavujú obdobie, kedy je vývoj projektu zastavený, nepridáva sa do projektu žiadna nová funkcionalita. Výsledok testu môže znamenať úspech projektu, resp. etapy projektu, alebo nájdenie chýb. V tom prípade sa väčšinou pokračuje opravou chýb a opätovným vykonaním akceptačných testov. Výsledok testov, našťastie som sa sním zatiaľ nestretol, môže byť aj neakceptovanie projektu.

Absentovanie tohto druhu testov môže mať veľmi zlé následky v období údržby projektu. Počas vývoja projektu som sa stretol zo systémom, v ktorom tento druh testov viditeľne chýbal, no zrejme aj niekoľko ďalších. Nazvem ho systém X. So službami, ktoré systém X ponúkal na rozhraní sa bolo potrebné integrovať. Tento systém bol už ukončený, na prvý pohľad fungoval a zdal sa byť stabilný. Problém nastal pri pohľade z bližšia. Vtedy sa zistilo, že systém X nespĺňa špecifikáciu požiadaviek podľa ktorej sa mal správať. Výstupy ktoré sa od neho očakávali boli často v iných dátových štruktúrach, nebolo dodržané ohodnotenie dátových väzieb a podobne. Dôsledok toho všetkého bolo veľké zdržanie vývoja projektu, ktoré spôsobilo čakanie na opravu chýb v systéme X. Odstránenie chýb postihlo jeho dátovú reprezentáciu a logiku ich spracovania. Vývojári pravdepodobne strávili viac času opravou chýb po ukončení projektu systému X ako pri jeho samotnom vývoji.

Vývoj riadený testovaním (angl. Test driven development)

Výborným postupom ako donútiť vývojový tím zamýšľať sa nad testovaním už v počiatočných štádiách projektu je využitie agilnej metódy vývoja nazývanej vývoj riadený testovaním (angl. skratka TDD).
Vývoj touto technikou pozostáva z niekoľkých krokov [3]:

1. Programátor vytvorí test pre novú funkcionalitu, ktorá ešte nie je implementovaná. Test končí neúspešne, pretože implementácia zatiaľ chýba.
2. Vytvorí sa jednoduchý kód, ktorý má šancu prejsť testom.
3. V prípade, že kód prejde úspešne testom, programátor môže pokračovať úpravou kódu, spraviť ho jednoduchší a rýchlejší. Ak je test neúspešný, pokračuje sa opravou kódu.
4. Programátor spúšťa testy znovu aby sa uistil, že úprava kódu nezničila už vytvorenú funkcionalitu. Tento cyklus pokračuje, kým nie je všetko úspešne implementované

Podľa môjho názoru má napísanie testu ešte pre samotnou implementáciou ďalšiu nespochybniteľnú výhodu. Práve príprava testov pomáha programátorovi predstaviť si štruktúry dát alebo algoritmus ešte predtým, než to začne implementovať. Aj v prípade, ak by tieto testy vytváral manažér projektu, pokiaľ pokryje celú škálu možných vstupov a výstupov, je programátorovi už z testov jasné ako sa má nová funkcionalita správať a reagovať na rôzne prípady vstupu. Využitie už vytvorených testov má veľký význam aj v neskorších štádiách projektu, kedy sa programátor rozhodne prerobiť kódu

vo väčšej škále. Vie, že keď má ostať funkcionalita zachovaná, tak musí zachovať rovnaké výsledky ako z predchádzajúcich testov. Aj v prípadoch, kedy na testovanie nezostane dosť času, je vhodné TDD využiť primárne aspoň na kritické a najdrahšie časti systému.

Testovacie prostredia a dáta

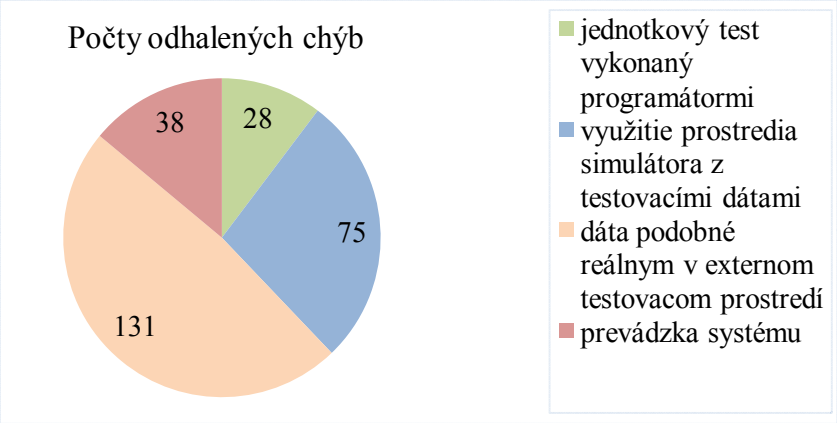
V mnohých prípadoch sa za najpodstatnejší pohľad na testovanie považuje spôsob testovania, často sa zabúda, že tak isto dôležitým aspektom je aj fakt v akom prostredí a s využitím akých prostriedkov sa testovanie vykoná. Jedným z dôležitých predpokladov kvalitného testovania sú dáta, na ktorých sa funkcionalita testuje.

Asi prvým testom, ktorý by mal každý programátor vykonať je jednotkový test, kde sa funkčnosť programu overí obvykle na statických dátach. Vytvorenej metóde sa explicitne podhodí niekoľko vstupných dát a pozrie sa, či vyprodukovala požadovaný výstup. Takýto spôsob je najrýchlejší, určite netreba jeho význam podceňovať. Dokáže odhaliť množstvo menších chýb, ako je napríklad nenaplnenie premennej hodnotou, nekonečný cyklus, alebo neošetrenie špeciálnych stavov programu.

Druhým krokom v testovaní novej funkcionality by mal byť simulátor, ktorý dokáže dáta dynamicky meniť v závislosti od vstupu. V prípade, že vytvorená metóda komunikuje zo systémom tretej strany, ktorý nemáme k dispozícii, simulátor by mal poskytovať volania podobne ako tento reálny systém. Pokiaľ vytvorenou metódou čítame dáta z databázy, simulátor by mal obsahovať testovaciu databázu z dátami rovnakej štruktúry ako reálne a aspoň minimálnu množinu dát, ktoré sú potrebné na otestovanie danej funkcionality. Týmto spôsobom môžeme zistiť niektoré ťažšie odhaliteľné chyby. Môžu to byť chyby typu nezatvárania pripojení na databázu, alebo zlá implementácia klienta, ktorý sa nedokáže pripojiť na server. Pokiaľ by sa tieto chyby neodhalili dosť zavčasu, prostriedky na ich odstránenie budú ďaleko väčšie.

Pokiaľ je projekt natoľko dôležitý, že chyba počas prevádzky by bola veľmi závažná, ďalším krokom testovania sa stáva testovanie na dátach, ktoré napodobujú reálne (angl. real-like data). Takýto krok sa vykonáva obvykle v testovacom stredisku zákazníka. Zákazník si nemôže dovoliť nasadiť do prevádzky systém s chybami, preto vykonáva tento krok testovania, aby znížil riziko vzniku chýb po nasadení systému. Práve týmto testovaním sa odhalia chyby, ktoré sa predchádzajúcim testovaním nepodarilo nájsť. Predpokladá sa, že systém, ktorý prejde týmto krokom už chyby neobsahuje. Samozrejme skutočnosť je iná, nikdy sa testovaním neodhalia všetky chyby, ešte som sa s tým nestretol. Napriek tomu sa týmto testovacím krokom pravdepodobnosť vzniku chýb v dôležitých častiach systému rapídne zníži.

Na ukážku som zistil koľko chýb sa podarilo odhaliť jednotlivým testovacím prostrediam. Graf na obrázku 1 zachytáva počet odhalených chýb v závislosti od testovacieho prostredia vo vývoji jedného reálneho projektu.



Obr. 1. Porovnanie množstva odhalených chýb v závislosti od použitých testovacích dát a prostriedkov.

Tvorba testovacích tímov a plánov

Príprava testovacích plánov v sebe zahŕňa skupinu činností ako príprava testovacích scenárov a procedúr a výber vhodných testovacích nástrojov. Podľa pána R. Hefnera v [2] plán testovania softvéru popisuje testovacie prostredie, ktoré bude použité na testovanie, identifikuje test, ktorý sa uskutoční a poskytuje časový plán testovacích aktivít. Všetky tieto črty by mal každý testovací plán samozrejme zahŕňať. Myslím si však, že rovnako dôležitou úlohou každého manažéra projektu je aj tvorba testovacích tímov, ktoré tento plán vykonajú. Niekedy potrebuje testera, ktorý rozumie návrhu systému, hoci nemá žiadne programovacie znalosti. Občas je to tester, ktorý je zvedavý a hľadá spôsoby ako sa nabúrať do systému a nachádza jeho slabé miesta. Inokedy je to ten, ktorý dokáže vytvoriť automatizované a dobre navrhnuté testy.

Vytvorením testovacieho tímu práca manažéra nekončí, dôležitou úlohou je začlenenie tímu do projektu. Testeri, ktorí vidia do procesu vývoja projektu, dokážu lepšie odhaliť chyby, sústrediť sa na kritické časti systému a dôsledne ich otestovať. Preto testovací tím netreba vynechávať z projektových stretnutí a oboznamovať ich o postupoch a rozhodnutiach pri vývoji.

Akého testera nepotrebujem!

Testovanie systému je práve také zložité ako jeho vývoj [3]. Práve preto tak ako pri výbere zručných programátorov, je dôležité vybrať si aj dobrých testerov.

Pokiaľ zanedbám špecializáciu testerov, dostaneme sa k podstatnej črte, ktorá by mala spájať všetkých z nich a to je dôsledný prístup k testovaniu. Tester, ktorý sa o systém vôbec nezaujíma, nepozná jeho architektúru a návrh, dokáže projekt o mnoho

viac zdržať ako mu pomôcť. V lepšom prípade len nekvalitne testuje systém, lebo nepozná jeho stavbu a očakávané správanie. V tom horšom prípade, zdržiava a znepríjemňuje prácu programátorom. Reportuje chyby systému, ktoré nie sú chybami, ale korektným správaním aplikácie v špeciálnych situáciách. Môže to byť napríklad chybové okno reagujúce na nesprávny vstup používateľa, nesprávne nastavenie klienta, alebo chyba externého systému. Situácie kedy si tester pomýli správanie aplikácie s chybou sú bežné, pokiaľ si ani len neprečíta používateľskú príručku. Percento zamietnutých reportovaných chýb sa bežne šplhá až do výšky 20 percent celkového počtu.

Testeri pri odhalení chyby sprostredkujú spätnú väzbu programátorom. Zvyčajne opis chyby reportujú do systému určeného na správu projektu. Práve spôsob, akým dokážu chybu opísať a aké informácie poskytnú o postupe odhalenia chyby, dokáže programátorovi ušetriť množstvo času. Ak je tester iniciatívny, k chybe vyhľadá v ladiacich výpisoch aplikácie ďalšie informácie, ktoré uľahčia jej identifikáciu. Na druhej strane tester, ktorému sa nechce hlbšie zaoberať popisom chyby dokáže jej odstránenie značne predĺžiť. Mnohokrát som sa stretol s prípadom, kedy popis chyby obsahoval obrazovku aplikácie s chybovým dialógom, ktorý neumožňoval ani len identifikovať o akú chybu ide, a akým spôsobom vznikla. Takáto situácia si zbytočne vyžaduje dodatočnú e-mailovú alebo inú komunikáciu s testerom a vývojový tím stráca drahocenný čas.

Kedy zapojiť testovanie do projektu?

Životný cyklus vývoja projektov môže byť rôzny. V žiadnom prípade by sa však testovanie nemalo odkladať na poslednú chvíľu. Uskutočňovať testovanie súbežne s vývojom projektu sa oplatí z jednoduchého dôvodu. Ak sa testovanie nenaplánuje paralelne z vývojom projektu, ale skončí ako oddelená aktivita, iba tým predlžuje vývoj projektu. Neskoro sa odhalia chyby a je potrebné vynaložiť viac úsilia na ich odstránenie. Najhoršou praktikou je využívanie skratiek v projekte, požadovanú funkcionalitu rýchlo implementovať takmer bez akéhokoľvek testovania. Neskôr to väčšinou projekt zdrží oveľa viac.

Pokiaľ sa chyba odhalí dostatočne skoro, programátor dokáže hneď zareagovať a zistiť čo ju mohlo spôsobiť. V opačnom prípade, hoci opravuje svoj vlastný kód, po niekoľkých týždňoch, alebo mesiacoch sa v ňom nemusí na prvý pohľad vôbec vyznať. Testovanie už počas vývoja poskytuje výhodu dostupností potrebných informácií a dát na odhalenie toho, čo chybu spôsobuje. Pokiaľ sa chyba objaví až v prostredí prevádzky systému je často náročnejšie dostať sa k ladiacim výpisom aplikácia, alebo k nazretiu do dát v databáze.

Záver

Testovanie softvéru zohráva vo vývoji projektov veľmi dôležitú úlohu. Ak je prevedené dômyselne, znižuje tým pravdepodobnosť, že projekt, na ktorý sa vynaložilo obrovské úsilie, zlyhá na nepredvídaných okolnostiach v dôležitých okamihoch.

Cieľom eseje bolo poukázať na postupy, ktoré pomáhajú zvyšovať kvalitu testovania pri vývoji projektov a tiež poukázať na vplyvy, ktorým je lepšie sa vyhnúť. Pokiaľ má niekto pocit, že sú opísané postrehy prehnané alebo sa s nimi ešte nestretol, má zrejme to šťastie, že pracuje s týmom testerov a manažérov, ktorý vedia vypracovať kvalitné testovacie plány a zabezpečiť ich úspešné uskutočnenie. Možno však niekto spoznal situácie, ktoré sa aj v jeho okolí stávajú alebo sa nimi inšpiruje pri budúcej tvorbe testovacích plánov. Možno práve táto esej niekomu pomôže vyhnúť sa zlým praktikám a ukáže mu ako zlepšiť kvalitu svojich projektov.

Použitá literatúra

1. Bertolino, A.: *Software testing research: Achievements, challenges, dreams*, Future of Software Engineering 2007. IEEE-CS Press, 2007.
2. Hefner, R.: *Collaborative Test Management*, *Software*, Engineering Workshop, Annual IEEE/NASA Goddard. IEEE Computer Society, 2002.
3. Rothman, J.: *Manage It!,* Pragmatic Bookshelf, Dallas (2007), 255-277.

Annotation

Qualitative testing – a key to good project, how to provide it?

The development of projects, small or big, software or of other types may also mean a creation of errors. The programmers create a code, but also errors which simply cannot be avoided. Testing serves to reveal and minimize the unwanted and unpredicted application failures, or to evaluate the project state. There are a lot of ways and plans how to perform testing effectively and qualitatively. However, the plans are not always performed successfully. It is often demanding to provide a skilled staff, or there is not enough time to carry out the testing plans. Testing plans may also fail easily if the used methods or tools are inappropriate. How to choose the appropriate way of testing? What techniques to combine? When to involve testing in the project? I will try to look at these questions more closely and reveal some erroneous decisions and practices that may occur in testing, as well as situations when testing can be ineffective and cause delays of the project development.