TESTOVAŤ? PREČO? KEDY? AKO?

Každý vie, že testovať treba, ale nikto netuší ako.

Marek Mego

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

Abstrakt. *Každá činnosť vrátane zastávanie určitej pozície na práci v softvérovom projekte so sebou prináša riziko vytvárania chýb, ktoré znehodnocujú výsledný softvérový produkt, znižujú jeho akosť. Sú dve možnosti ako tento problém riešiť. Tou prvou je snaha eliminovať generovanie chýb, čo je však nadľudská a pravdepodobne neriešiteľná úloha. Naopak druhou, omnoho jednoduchšou možnosťou je snaha vyhľadať tieto chyby a zabrániť tak ich prípadnému šíreniu, čo vo veľkej miere prispeje ku kvalite výsledného softvérového produktu. Vyhľadanie chýb možno riešiť testovaním. Práve aspekty testovania, kedy a ako testovať, sa pokúsim objasniť v nasledujúcich riadkoch. Načrtnem možné spôsoby testovania a porovnám ich medzi sebou. Uvediem zaujímavú vývojovú metodiku Test Driven Development a uvediem svoj postoj k problematike.*

Kľúčové slová: *testovanie, kvalita, Walktrough, Inspection, Biela skrinka, Čierna skrinka, Test Driven Development*

Záhadná nutnosť testovania?

Výskum v Spojených štátoch amerických [3] ukázal že testovanie na softvérovom projekte môže zabrať 50% a viac celkového času na projekte. Schválne nenaznačujem či je to veľa alebo málo každopádne je celkom zrejmé, že väčšina z nás toľko času testovaním nad softvérovým projektom nikdy nestrávila. Prečo ? Kladiem si rovnakú otázku. Po krátkej úvahe však nachádzam niekoľko odpovedí. Pravdepodobne sa ešte väčšina z nás nestretla s projektom kde ide "o život", ktorého výsledok a prezentácia to celé rozhodne. Druhým dôvodom je iste časový nedostatok. Výsledný produkt sa podarí dokončiť pár minút pred

koncom termínu, všetci sme spokojní, v kútiku dúfame, že sa nevyskytnú žiadne mimoriadne poruchy. A oni sa niekedy ozaj aj nevyskytnú. Ktorý prístup je ale správny? Venovať 50% času testovaniu, ktoré vyzerá to tak, že odhalí nedostatky, alebo "to risknúť" ? Všeobecná odpoveď existuje, správne je venovať čas testovaniu a zabezpečiť úplnú funkčnosť výsledného produktu, ale naozaj sa treba touto "všeobecnou radou" riadiť ? Ukážem na inom príklade.

Antonia Bertolino napísala: "Testovanie sa všeobecne používa na zaistenie kvality" [1] . Jediná neznáma v tejto citácii je kvalita, preto som sa snažil nájsť definíciu kvality v rovnakom dokumente prípadne v dokumente na ktoré sa Antonia Bertolino odvoláva. A našiel som. Podľa [**Chyba! Nenalezen zdroj odkazů.**] existujú akési základné atribúty kvality, ktoré boli neskôr rozšírené a bol z nich vytvorený ISO štandard.

Spomínané atribúty kvality sú:

- Správnosť - ako program spĺňa požadované vlastnosti
- Spoľahlivosť - schopnosť programu splniť presne to, čo sa od neho očakáva
- Integrita - schopnosť odolať neúmyselným aj úmyselným útokom na softvér
- Intuitívnosť – miera jednoduchého a ľahkého používania systému
- Efektívnosť – množstvo prostriedkov potrebné na vykonanie funkcií programu
- Udržiavateľnosť – opisuje ako ľahko je možné vyhľadať problém
- Flexibilita - opisuje ako ľahko je možné vykonať zmenu v softvéri
- Testovateľnosť – ako ľahko je možné otestovať deklarované vlastnosti nástroja
- Interoperabilita – opisuje mieru integrovateľnosti jedného systému do druhého
- Znovupoužiteľnosť – schopnosť znovu použiť celý nástroj alebo jeho časti v inom nástroji
- Prenositeľnosť – schopnosť presunúť nástroj z jednej platformy na inú

Spomenuté atribúty kvality sú nespochybniteľne dôležité, avšak natíska sa mi otázka. Sú všetky vždy dôležité ? Sám autor [3] ponúka odpoveď. Každý prikladáme jednotlivým atribútom rôznu dôležitosť. Podľa tejto vety usudzujem, že každý si predstavujeme kvalitný softvér inak. Keď si spomenieme na predchádzajúci výrok, testovanie sa všeobecne používa na zabezpečenie kvality, a je zrejmé, že každý túto kvalitu chápeme trochu inak môžem zodpovedne prehlásiť, že sú situácie kde testovanie jednoducho nie je potrebné, čiže sú situácie keď naozaj nie je potrebné venovať 50% celkového času na softvérovom projekte testovaním.

Nechcel by som však spochybňovať testovanie ako také, ktoré potvrdzujem je nesmierne dôležité pri vývoji softvérového produktu. Naopak by som chcel poukázať akým spôsobom sa miera času spotrebovaná testovaním vyvíjala.

	Requirements Analysis	Preliminary Design	Detailed Design	Coding and Unit Testing	Integration and Test	System Test
1960s – 1970s	10%			80%	10%	
1980s	20%		60%		20%	
1990s	40%	30%		30%		

Obr. 1. Vývoj práce na softvérovom projekte.

Z predchádzajúcej tabuľky je zrejmé, že v sedemdesiatych rokoch sa testovaniu venovalo až 90% času! V rokoch 1980-1990 to bolo už "len" 80 % času a ku koncu tisícročia už len 30% času. Týmto by som chcel poukázať na znižujúce sa nároky na testovanie a zároveň by som si rád dovolil malú špekuláciu. Takýmto tempom sa rýchlo presunie dôležitosť úkonov na softvérovom projekte práve na prvotné analýzy a detailné návrhy, simulácie tak aby samotné testovanie a programovanie produktu predstavovalo len zanedbateľnú časť. Nakoniec táto odvážna úvaha nemusí byť nutne zlá, keďže je všeobecne známe, že oprava chyby v softvérovom projekte stojí tým viac, čím viac je projekt v jeho pokročilej fáze. Čiže samotné opravenie chyby v analýzach, simuláciach a návrhoch bude i ekonomické.

Ako?

Hoci som sa v predchádzajúcej časti snažil naznačiť, že možno nie vždy je testovanie potrebné, predsa len treba povedať že pri väčšine prípadov testovanie potrebné a užitočné je. Tu sa naskytá otázka, aký druh testovania je vhodné použiť? Ako testovať?

Existuje niekoľko rozdelení. Podľa skutočnosti či softvér treba reálne spustiť a získať výsledok na základe vstupných hodnôt, testovanie rozdeľujeme na statické a dynamické.

Pri dynamickom je nutnosť spustiť softvér , pri statickom nie.

Dynamické testovanie reprezentuje predovšetkým dvojica testov. Biela a čierna skrinka.Pri testovaní pomocou čiernej skrinky používateľ nemá znalosť o vnútornej štruktúre systému, použitých algoritmoch ale môže iba meniť vstupy na základe ktorých mu program vytvorí výstup, ktorý je následne skontrolovaný s očakávaným výstupom. Naopak pri testovaní pomocou bielej skrinky má používateľ dokonalý obraz o štruktúre systému, použitej logike a teda vie veľmi presne odhadnúť množinu vstupov. napríklad tak aby pokryl všetky rozhodovacie body. Z uvedených vlastností hodnotím obe skrinky ako použiteľné avšak je jasné, že ak je možnosť použiť bielu, nie je dôvod robiť to inak, keďže pri testovaní pomocou čiernej skrinky nemusíme nikdy odhadnú tú správnu množinu vstupov, ktorá poukáže na chyby. Myslím, že obe testovania sú vo všeobecnosti používané a ľudia vykonávajúci testovanie ani len netušia že testujú softvér pomocou bielej resp. čiernej skrinky. Tento fakt len umocňuje správnosť a prirodzenosť testov.

Statické testovanie tiež predstavím pomocou dvoch zaujímavých testov, Walkthrough a Inspection.

Walktrough je neformálna prehliadka výsledku tvorivej činnosti pri ktorej autor oboznamuje členov týmu prípadne iných ľudí o výsledku svojej práce. Ľudia prítomný na prehliadke majú úlohu, respektíve ich cieľom je pýtať sa detaily o projekte a následne identifikovať možný problém, prípadne poukázať nato, že existuje lepšie riešenie. Myslím si, že práve tu tkvie obrovská výhoda tohto spôsobu testovania. Vysvetlím na prípade, ktorý sa mne a myslím, že aj ostatným vo všeobecnosti stáva často či už v roli autora alebo človeka prehliadajúci si projekt. Človek je určite druh omylný, z toho vyplýva, že je potrebná kontrola. Avšak kto túto kontrolu vykonáva ? Nie vždy sa nájde iný dostatočne motivovaný tester, a preto dielo musí skontrolovať sám autor. Tu však nastáva problém. Ako môže autor objektívne spraviť kontrolu diela ktoré sám vytvoril ? Myslím, že by jeho myseľ musela prejsť dodatočným “zresetovaním” pretože inak je nemožné spraviť objektívnu kontrolu vzhľadom nato, že autor sám dielo vytváral a môže byť vnútorne

presvedčený o svojej pravde a z toho dôvodu je veľká pravdepodobnosť, že chyby prehliadne presne tak, ako to urobil pri tvorbe diela. Spomínaná paralela s bežným životom spočíva v tom, že veľa krát si povieme niečo na spôsob : "Tri krát si to skontrolujem a je to hotové" . V skutočnosti iba tri krát prehliadnem tú istú chybu. Niekedy síce postačí malá finta v podobe zväčšenia časových rozostupov medzi tvorbou diela a jeho skontrolovaním, avšak ani toto nemusí byť úplne korektné. Myslím si, že najlepším možným riešením je práve testovanie Walktrough respektíve jeho obdoby. Až tu, keď má možnosť sa pozrieť na produkt viacero nezávislých pozorovateľov je možnosť získať korektnú spätnú väzbu od každého z nich. Podľa môjho názoru, výhoda spočíva práve v tom, že každý už máme určité skúsenosti na práci na softvérovom projekte, pričom každý môže mať úplne iné skúsenosti pri riešení problémových situácii. A práve tieto skúsenosti sa dajú použiť pri prehliadke práce cudzieho autora. Možno urobil rovnaké chyby, ktoré my už teraz po určitých skúsenostiach vieme detekovať. Myslím, že tento druh testovania je jedným z najlepších, ak nie najlepším. Aj z toho dôvodu, že je prirodzené rozprávať o projekte, diskutovať a tým nenútene hľadať chyby ale aj pre mimoriadnu efektivitu tohto testovania. Chyby v tomto testovaní sa dajú odhaliť hneď nazačiatku vývoja softvérového projektu. Walkthrough totiž nemožno chápať len ako prehliadku implementovanej časti, ale prehliadku ľubovoľnej časti v živote vývoja softvérového projektu ako napríklad tvorba špecifikácie, návrhu či dokumentácie.

Príbuzným z kategórie prehliadok je tiež metóda Inspection. V zásade ide o to isté ako u Walkthtrough, avšak prehliadka diela je formálna. To znamená, že existuje protokol podľa ktorého je dané, kto akú úlohu má pri testovaní vykonávať, prípadne aké vlastnosti projektu sa majú sledovať. Myslím si, že tento spôsob je dobrý ale ten predchádzajúci bol lepší. K tomuto tvrdeniu ma núti poznanie, ktoré hovorí, že ak je niečo presne stanovené podľa šablóny a postup je exaktný, chýba tu moment prekvapenia. A sú to práve vzniknuté chyby, ktoré tento moment prekvapenia skôr či neskôr prinesú.

Ďalej existuje rozdelenie možností testovania podľa fázy v akej sa softvérový projekt nachádza. Sú to:

- Unit Testing – predstavuje testovanie najmenšej nedeliteľnej jednotky programu.
- Integration Testing – testuje integráciu dvoch alebo viacerých jednotiek spájajúcich sa do veľkého celku.
- System Testing- Systém sa testuje ako celok. Testuje sa funkčnosť , bezpečnosť, spoľahlivosť a podobne.
- Acceptance Testing – Testuje sa systém tak, ako by ho zákazník používal v praxi. Cieľom je skôr získať dôveru v systém než nájdenie chyby.

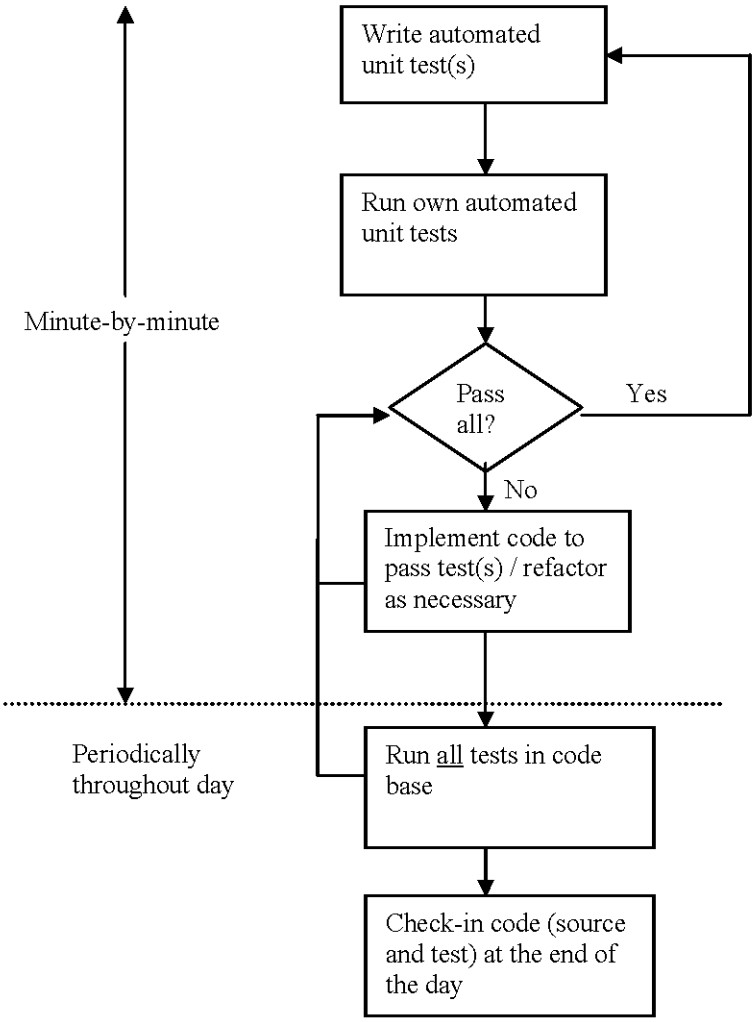
Myslím, že všetky tieto testovania majú svoje opodstatnenie v životnom cykle softvérového projektu, ale predsa len by som chcel upozorniť na posledné testovanie a síce akceptačné testovanie. Nie som si istý ako často sa to stáva, ale myslím si, že v prípade že sa pri tomto testovaní nájde dôležitá chyba, ktorá by teoreticky spôsobila zmenu systému v jeho základoch, niečo v celom procese je zle, niečo bolo zanedbané. Potom už nezostáva nič iné, len sa vyhovárať na Murphyho zákon, ktoré hovorí, že chyba sa objaví v tej najnevhodnejšej chvíli.

Test Driven Development

„Najprv testy, potom kód!" To je charakteristika tejto prevratnej metódy testovania. Jej cieľom nie je ani tak upozorniť ako testovať, ale kedy testovať. Ideou tohto prístupu je vytvoriť test ešte skôr ako samotný kus kódu, ktorý chceme testovať . Bláznivé, nie ? Aspoň ja som si to myslel pri mojom prvom stretnutí s prístupom TDD. Nedôverčivosť však opadla po bližšom skúmaní, ako táto technika funguje. Vývoj v softvérovom projekte pomocou techniky TDD sa skladá z iterácii, ktoré sa neustále opakujú po každom pridanom a otestovanom kuse kódu.

Spomínané iterácie sú:

- Napísať niekoľko automatických testov pre triedu, ktorú ideme ideme programovať
- Spusitť novo vytvorený test a presvedčiť sa, že naozaj zlyhá, keďže ešte nič nebolo implementované
- Implementovať kód vďaka, ktorému by mal byť novovytvorený test triedy úspešný.
- Refaktorovať, pozmeniť kód triedy podľa potreby
- Spustiť všetky doterajšie testy a uistiť sa tým, že nedošlo k porušeniu ostatných častí implementácie.



Obr. 2. Znázornenie iterácii potrebných pri TDD.

Na základe uvedeného si myslím, že táto technika je zaujímavá predovšetkým v dvoch rovinách.

V prvej rovine je dôležité predovšetkým, že TDD núti vývojára testovať, nedovolí mu nechať si testovanie na neskôr a tým pádom neodhaliť chyby už v ich začiatkoch. Toto je nesmierne dôležité, pretože inak by vývojár musel hľadať svoje chyby pri časovo náročnom testovaní. Na druhej strane tiež takáto forma testovania pridáva dôveru v samotného programátora, pretože po každom zbehnutom teste vývojár vie, že všetko funguje a nemusí dúfať po náročnom vývoji softvéru, že všetko bude fungovať.

Čo sa týka druhej roviny, zaujímavou vlastnosťou je proces integrácie novej jednotky medzi už existujúce. Vždy pri pridaní novej jednotky sa skontrolujú už existujúce čiže nemôže dôjsť k situácii, keď v strede vývoja softvéru zistíme, že tieto dve časti sú navzájom nekompatibilné.

A sú to práve takéto na prvý pohľad maličkosti, ktoré môžu skrátiť dobu vývoja softvéru o nezanedbateľnú hodnotu, a spraviť samotný produkt ešte viac robustnejším.

Záver

V práci som sa snažil poskytnúť odpovede na základné otázky, ktoré by si mohol položiť človek, ktorý prvý krát chce mať skúsenosť s testovaním a síce, či vôbec testovať, ako testovať a kedy. Myslím, že sa mi podarilo fundovane zodpovedať na všetky otázky. Chcel by som však vyzdvihnúť metódu testovania Walktrough a techniku vývoja softvéru Test Driven Development. Keby som sa mal zamerať pri testovaní len na dve veci, sú to práve tieto dve a to hlavne kvôli ich efektívnosti, jednoduchosti a pri Walktrough aj prirodzenosti.

Dovolím si tvrdiť, že pri testovaní pomocou týchto dvoch metód, pri dodržaní zásad TDD techniky a zabezpečení adekvátne skúsených pozorovateľov pri metóde Walkthrough, softvér bude kvalitný a to aj podľa štandardov ISO spomenutých v eseji.

Použitá literatúra

1. Bertolino, A.: *Software Testing Research: Achievements, Challenges, Dreamse*. Future of Software Engineering, 2007. ISBN 0-7695-2829-5.
2. Bieliková, M.: *Softvérové inžinierstvo, Princípy a manažment*. Vydavateľstvo STU, Bratislava, 2000.
3. NIST: *The economic impacts of inadequate infrastructure for software testing*. 2002. [cit 2009.16.10] Dostupné na internete: <http://www.nist.gov/director/prog-ofc/report02-3.pdf>
4. Thirumalesh B., Nachiappan, N.: *Evaluating the efficacy of test-driven development: industrial case studies*. Future of Software Engineering, 2006. ISBN 1-59593-218-6.

Annotation

Need to test? Why? How? When?

Any activity, including the holding of a position in software project brings the risk of creating unpredictable errors that degrade the resulting software product and reduce its quality. There are two ways to solve this problem. The first is to eliminate the generation of errors, which is superhuman, and probably unsolvable task. On the contrary, the second way and much simpler option is to find these errors and avoid possible spread of these errors, which largely contribute to the quality of the resulting software product. Finding errors can be managed by testing. In this document, I will try to clarify aspects of testing, when and how to test. I will mention possible ways of testing and compare them between each other. I will introduce interesting development methodology, Test Driven Development, and also i will mention my opinion about this issue.