Testovať! Ale ako?

JÁN KOLESÁR

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
jkolesar@gmail.com

Abstrakt. Testovanie je nevyhnutnou súčasťou vývoja softvérového projektu. Je to jedna z metrík kvality projektu. Kvalitný softvér predsa musí byť aj kvalitne otestovaný. No v súčasnosti existuje veľké množstvo metód testovania. Ktorú z nich by teda mal zodpovedný programátor použiť? V tejto práci sa zaoberám vybranými metódami a technikami testovania. Zhodnotím výhody a nevýhody unit testov, test-driven development (techniky metódy bielej skrinky) a niektorých techník metódy čiernej skrinky.

Úvod

Softvérový priemysel je obrovský a rýchlo sa rozvíjajúci obchod. S jeho rozvojom sa rozvíja aj konkurenčný boj softvérových spoločností o zákazníka. Softvérové systémy vyvíjané týmito spoločnosťami sú produkty, ktoré sa musia predať, aby priniesli zisk. Vzniká tu teda tlak na zvýšenie schopnosti konkurovať ostatným produktom na trhu a tým aj na zvýšenie kvality softvérových systému.

Jednou z možností, ako zvýšiť kvalitu softvéru, je kvalitné testovanie. V zásade sa dajú spôsoby testovania softvérových systémov rozdeliť na dve skupiny. Sú to metódy „čiernej skrinky“ (black box testing) a metódy „bielej skrinky“ (white box testing).

V tejto práci sa budem venovať niektorým metódam z obidvoch spomínaných skupín, no prevažne metódam zo skupiny metód bielej skrinky.

Metódy „čiernej skrinky“

Ako už vyplýva z názvu tejto skupiny metód, testovaný objekt vnímame ako čiernu skrinku. To znamená, že nevidíme čo je vo vnútri a ako v skutočnosti pracuje. Jediné čo je nám o testovanom objekte známe je jeho špecifikácia. Vieme teda čo má byť na výstupe pri daných vstupoch. Z tohto dôvodu tieto metódy bežne nepoužívajú programátori [1]. Na druhú stranu laického užívateľa nezaujíma ako softvér vo vnútri pracuje, ale či spĺňa špecifikáciu. Teda či spĺňa úlohu, pre ktorú bol vyvinutý.

Manažment v softvérovom inžinierstve, október 2007, s. 1-1.

Medzi tieto metódy patria napríklad metóda „*akceptačných testov*“ (user acceptance testing) a testovanie na základe prípadov použitia.

Akceptačné testy

Akceptačné testy sú jednou z posledných fáz vývoja softvérového systému. Obvykle sú vykonávané pred tým, ako zákazník príjme nový systém [2]. Na vytváraní akceptačných testov sa aktívne zúčastňuje budúci používateľ. Samotné testy by mali byť vytvorené podľa špecifikácie ešte pred vytvorením softvérového systému, ktorý majú testovať. Plán testov býva rozdelený do niekoľkých náročnostných vrstiev. Obvykle od najjednoduchších problémov ako sú napríklad gramatické chyby alebo kozmetické nedostatky, až po najhoršie problémy typu spadnutie systému či úplná nefunkčnosť systému. Myslím si že táto metóda testovania môže pomôcť programátorovi lepšie pochopiť požiadavky užívateľa, no počas fázy vývoja samotného softvéru (písania zdrojových kódov), mu už nepomôže. Ďalším výsledkom úspešného vykonania akceptačných testov je že sa na konci vývoja môže užívateľ aj programátor uistiť že softvér funguje. Mali by teda byť súčasťou vývoja softvérového systému. No pre celkový vývoj nestačia.

Testovanie na základe prípadov použitia

Otestovať všetky možné konfigurácie systému je takmer vždy nemožné. To znamená, že testy programov sú nevyhnutne nekompletné. Preto je potrebné určiť si podmnožinu testov, ktoré by mohli odhaliť čo najväčšie množstvo chýb. Pri tejto metóde je množina testov vytvorená na základe diagramov prípadov použitia. Tieto diagramy obvykle bývajú súčasťou špecifikácie a popisujú funkcie systému a postupy, ktoré vedú k ich využitiu. Pre vytvorenie testov je teda potrebný popis funkcie systému, postup pre jej využitie, množina vstupných dát, množina očakávaných výstupných dát a množina možných výnimočných stavov systému.

Táto metóda testuje systém priamo jeho používaním. To znamená ž otestuje len vonkajšie rozhranie systému. No mohla by sa použiť ako doplnková metóda ku iným testom.

Metódy „bielej skrinky“

Na rozdiel od metód čiernej skrinky, tu poznáme vnútornú štruktúru testovaného objektu. Teda poznáme zdrojové kódy softvérového systému a na základe nich ho testujeme. Metódy tejto skupiny využívajú samotný programátori počas vývoja softvéru.

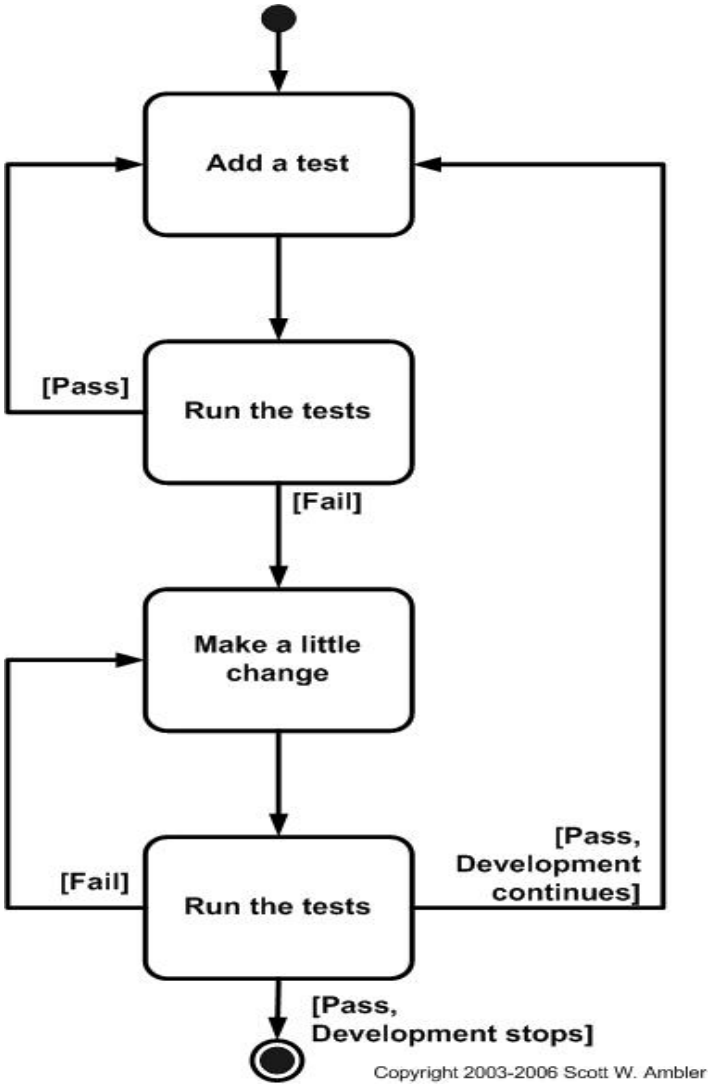
Do tejto skupiny metód patria napríklad metóda „*testovania modulov*“ (unit testing) a „*vývoj riadený testami*“.

Testovanie modulov

Testovanie modulov je proces testovania jednotlivých podprogramov, procedúr či funkcií programu. Radšej ako testovanie programu ako celku, je testovanie zamerané na testovanie malých blokov kódu. Podľa knihy [4] na to existujú tri dôvody. Po prvé, keďže pozornosť je od začiatku zameraná na malé bloky programu, je testovanie modulov spôsob manažovania kombinovaných prvkov testovania. Po druhé testovanie modulov uľahčuje úlohu debugovania (proces určovania a opráv objavených chýb). Keď sa totiž vyskytne chyba, je jasné v ktorom module sa nachádza. Po tretie testovanie modulov zavádza paralelizmus do testovania programu. Umožňuje totiž testovanie viacerých modulov súčasne. Úlohou testovania modulov je porovnať funkcionalitu modulu so špecifikáciou daného modulu. Myslím že tieto dôvody sú veľkým prínosom a ukazujú výhody tejto metódy. No aby sa testy modulov dali použiť, musí byť zdrojový kód programu dobre separovateľný. Jednotlivé moduly by mali byť navzájom nezávislé, čo sa niekedy dá len ťažko dodržať. V praxi sa preto niekedy testujú naraz skupiny modulov.

Vývoj riadený testami

Vývoj riadený testami (test-driven development) vlastne nie je len metóda testovania, ale technika programovania založená na testovaní. Vývoj riadený testovaním spája test-first design a refactoring [3]. Je to teda technika, kde programátor napíše test pred tým, ako napíše program, ktorý nim bude testovať. Kroky test-first design sú znázornené na obrázku nižšie (**Error! Reference source not found.**).



Obrázok 1 UML diagram krokov v test-first design

Programátor teda najprv pridá test, ktorý by mal skončiť neúspešne kvôli absencii implementácie testovanej funkcionality. Následne ho spustí aby sa presvedčil, že test zlyhá. Ďalším krokom je doplnenie programu tak aby test prešiel. Novo doplnený kód môže narušiť funkčnosť už existujúceho kódu, preto je potrebné spustiť aj ostatné testy. Keď všetky testy prejdú programátor prejde k refaktorizácii, aby ním napísaní kód zodpovedal vysokej kvalite (aby bol „čistý“). Ak aj po refaktorizácii testy prejdú programátor začne odznovu (pridá ďalšie testy). Vývoj riadený testami má dve jednoduché pravidlá. Programátor musí napísať kód programu len ak nejaký z testov

zlyhá. Vlastne by nemal pridať ani riadok kódu, kým preň nebude mať test. Druhé pravidlo je, že by mal postupovať po malých krokoch.

Vývoj riadený testovaním núti programátora myslieť na dizajn pred tým ako začne písať funkčný kód. Jeho cieľom je napísať čistý kód, ktorý funguje. Keďže programátor postupuje po malých krokoch a vytvorí testy pre kód pred samotným kódom, dokáže odhaliť chyby omnoho skôr ako tradičné spôsoby vývoja softvéru. Toto je obrovská výhoda. Cena za úsilie vynaložené na odstránenie chyby nesmierne rýchlo stúpa. Čím neskôr programátor chybu odhalí tým horšie sa opravuje. Zmena totiž zasiahne vyššiu časť kódu. Ďalšou výhodou tejto metódy je samotný zdrojový kód testov. Je to totiž priama ukážka spôsobu použitia jednotlivých častí programu. Poskytuje tak dobrú pomôcku k dokumentácii.

Samozrejme aj táto metóda má nedostatky. Nedá sa totiž použiť tam kde nie je možné použiť automatické testovanie. To sa prejaví najmä pri vývoji grafického používateľského prostredia, distribuovaných objektov, ... Rovnako je nutné uvedomiť si, že vývoj riadený testovaním nezaručuje správnosť programu. Pokiaľ testy nezodpovedajú špecifikácii kód, ktorý pre ne vytvoríte nebude spĺňať svoj účel. Pre dosiahnutie správneho programu musíte teda najprv vytvoriť správne testy a to si už vyžaduje aj nejaké skúsenosti.

Napriek nedostatkom si myslím že je to veľmi užitočná metóda a jej využitie prinesie viac úžitku za rozumnú cenu.

Záver

Obidve skupiny metód majú svoje výhody aj nevýhody. Metódy čiernej skrinky obvykle nie sú používané priamo programátormi, no dajú sa použiť na testovanie modulov tretích strán. Ak programátor použije externú knižnicu mal by si byť istý jej správnosťou alebo ju otestovať. Hovorí sa že nikto by nemal kontrolovať svoj vlastný výtvor (teda nemal by byť posledný čo ju kontroluje). Iný tester sa na vec môže pozrieť z inej strany. Môže tak odhaliť chybu aj tam, kde by ju autor programu nečakal a nehľadal.

No pri vývoji je nutné použiť aj metódy bielej skrinky. Človek je omylný a ani programátor nie je výnimkou. Potrebuje teda kontrolovať svoje dielo, aby zabránil vneseniu chýb. Či odhalil a odstránil už zanesené chyby.

Ani jedna metóda nie je dokonalá no poskytuje určitú istotu a pomoc pri vývoji softvéru. Ale najlepšia voľba je kombinácia viacerých metód. Aspoň po jednej z obidvoch skupín. To čo jedna nedokáže to môže zabezpečiť druhá.

Použitá literatúra

1. Dictionary LaborLaw Talk: Black box testing, [cit 2007-11-23]. URL:<http://dictionary.laborlawtalk.com/Black_box_testing>.
2. Dictionary LaborLaw Talk: User acceptance testing, [cit 2007-11-23]. URL:<http://dictionary.laborlawtalk.com/User_acceptance_testing>.

3. Introduction to Test Driven Design (TDD), [cit 2007-11-23]. URL:<http://www.agiledata.org/essays/tdd.html>.
4. The Art of Software Testing, [cit 2007-11-23]. URL:<http://books.google.com/books?hl=en&lr=&id=tzI4S5x5smkC&oi=fnd&pg=PR11&dq=preparing+testing+of+software+projects&ots=S4tM7LsDlG&sig=BovZnM2MkcWsluVwexUBLbPHLS4#PPA91,M1>.

Annotation

Test! But how?

Testing is a necessary part of software project development. It is one of metrics of project quality. Software with high quality must by test with high quality, too. But in present there are many testing methods. So which should responsible programmer use? I entertain some of them in this work. I will valorize benefits and limitations of unit testing, test-driven development (white box methods) and some black box methods.