Testovanie objektovo orientovaného softvéru

RUDOLF FÁBER

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

Abstrakt. Testovanie je dôležitou etapou vývoja každého softvérového produktu, objektovo orientovaný vývoj softvéru nevynímajúc. Napriek tomu, má testovanie objektovo orientovaných programov určité zvláštnosti voči testovaniu iného typu softvéru. Táto esej sa zaoberá významom testovania, rôznymi technikami testovania objektovo orientovaného softvéru a ich možnými výhodami a nevýhodami. Porovnávajú sa rôzne stratégie testovania a prístupy, ktoré tieto stratégie využívajú. Detailnejšie sa esej zaoberá aj testovaním samotných tried na základe ich formálnej špecifikácie. V ďalšej časti sa zaoberá regresným testovaním, jeho významom pri údržbe softvéru a predpokladov potrebných pre efektívne vykonávanie regresných testov.

Úvod

Testovanie je jednou z etáp životného cyklu každého programu. V súčasnosti existuje veľké množstvo metód pre testovanie rôznych softvérových produktov, vrátane produktov vytvorených objektovo orientovaným vývojom. Veľké množstvo existujúcich metód či postupov však nezaručuje požadované výsledky, pokiaľ nie sú používane vhodne.

Testovanie ako prostriedok k dosiahnutiu kvality

Pri posudzovaní každého softvérového produktu je jeho kvalita jedným zo základných predpokladov pre to, aby mohol byť projekt považovaný za úspešný. Preto je povinnosťou každého manažéra dohliadať na to, aby bol vytváraný produkt čo najkvalitnejší, teda aby do maximálnej miery spĺňal vstupnú špecifikáciu.

Testovanie softvérového produktu je základným spôsobom pre zabezpečenie jeho požadovanej kvality. Pre dosiahnutie tejto kvality je potrebné testy dôsledne naplánovať, navrhnúť ich štruktúru, pravidelne ich vykonávať a samozrejme dohliadať na výsledky testov a správne ich vyhodnocovať. Testovanie je všadeprítomné

Manažment v softvérovom inžinierstve, október 2006, s. 1-7.

v procese vývoja softvéru, pričom jeho úlohou je zaistiť, že návrh a implementácia programov spĺňa špecifikované požiadavky [1].

Veľké množstvo súčasných softvérových projektov je vytváraných podľa princípov objektovo orientovaného programovania. Aj tieto produkty je samozrejme potrebné testovať. Pri testovaní je však potrebné brať do úvahy špecifiká tohto spôsobu tvorby produktu a prispôsobiť im aj samotné testy a testovanie celkovo.

Možnosti testovania

Vo väčšine zdrojov môžeme nájsť nasledovné rozdelenie testovania [1,2] :

- Testovanie založené na špecifikácii – metóda čiernej skrinky (black-box testing)
- Testovanie založené na štruktúre alebo na programe – metóda bielej skrinky (white-box testing)

Testovanie metódou čiernej skrinky sa zameriava na overovanie funkčnosti a správania sa produktu alebo časti produktu z externého pohľadu, čiže z hľadiska používateľa, prípadne iného softvérového systému. Testovanie metódou bielej skrinky sa zameriava na kontrolu vnútornej štruktúry produktu. Pri tomto type testovania je potrebné brať do úvahy štruktúru zdrojového kódu a spôsob riadenia programu.

Nie je možné jednoznačne určiť, že pri určitých typoch programov je potrebné využívať iba jeden z uvedených druhov testovania. Keď beriem do úvahy oblasť objektovo orientovaného programovania tak takmer každý program je potrebné testovať metódou čiernej skrinky na overenie jeho správania a reagovania na dané vstupy. V väčšine prípadov však je potrebné používať aj metódu bielej skrinky – najmä pri komplikovanejšej vnútornej logike programu, či pri odhaľovaní príčiny závažnej chyby.

Špecifiká testovania objektovo orientovaných programov

Obidve metódy testovania spomenuté v predchádzajúcej kapitole sa dajú uplatniť na program vytváraný v ľubovolnej programovacej paradigme. Testovanie objektovo orientovaných programov však prináša svoje špecifiká, ktoré vychádzajú zo samotnej základnej myšlienky tejto paradigmy. Podľa [2] je potrebné odpovedať na nasledujúce otázky :

- Môžu byť rôzne testovacie techniky uplatnené na triedy a na skupinu tried s určitým počtom príbuzných tried?
- Ktoré testovacie modely, metódy generovania testov a testovacie kritériá môžu byť použité pri čiastkových testoch (unit tests) pre objektovo orientovaný softvér?
- Ako systematicky vykonať čiastkové testy pre objektovo orientovaný program?

V [2] sú bližšie rozoberané niektoré formálne spôsoby testovania objektovo orientovaných programov – napríklad testovanie prostredníctvom toku údajov vyjadreného grafom, či testovanie založené na naviazaní dát. Ja by som však rád ponúkol vlastný pohľad a pokúsil sa zodpovedať na spomenuté otázky.

Pri testovaní skupiny tried či celého produktu je potrebné brať do úvahy, čo je úlohou daných tried, aké dáta triedy združujú a ako sú triedy hierarchizované. Pri navrhovaní testov, najmä testov pre metódu čiernej skrinky však netreba zabúdať, že podstatou je dosiahnuť kvalitu softvéru ako celku. V niektorých prípadoch je samozrejme potrebné poznať vnútornú štruktúru programu. Správny postup pri testovaní celého produktu by podľa mňa mal vyzerať nasledovne :

1. Testovanie celého produktu na základe vstupnej špecifikácie. Posúdenie, či produkt spĺňa požadované vlastnosti podľa špecifikácie, či je vlastne produkt správny. Toto testovanie je testovanie metódou čiernej skrinky. Pokiaľ sa v tomto kroku produkt správa korektne, neznamená to ešte, že každá jeho súčasť je korektná!
2. Testovanie jednotlivých identifikovaných častí produktu – čiastkové testovanie. Takýmito časťami môžu byť napríklad úplne samostatné moduly, pokiaľ je produkt komplexný a zložený z viacerých menších, skoro samostatných produktov. Pri väčšom produkte je možné (a vhodné) túto fázu vykonávať iteratívne, na rôznych úrovniach zjemnenia. Pri tomto kroku stále ide o testovanie metódou čiernej skrinky. Tejto fáze treba venovať veľkú pozornosť, pretože práve tu by mali byť odhalené mnohé chyby, ktoré sa pri pohľade zhora (krok 1) nemusia prejaviť. Množiny testovacích dát by mali byť navrhnuté tak, aby pokryli všetky možné situácie, ktoré môžu nastať vzhľadom k povahe testovanej časti.
3. Testovanie jednotlivých tried, alebo malých skupín úzko previazaných tried metódou bielej skrinky. Týmto spôsobom vo väčšine prípadov nie je potrebné otestovať všetky časti systému. Ako som už spomínal, je potrebné sa zamerať najmä na triedy , ktoré vykonávajú zložitejšie operácie s dátami, prípadne komplikovanejšie výpočty. Testovaním tried sa ešte budem detailne zaoberať neskôr.

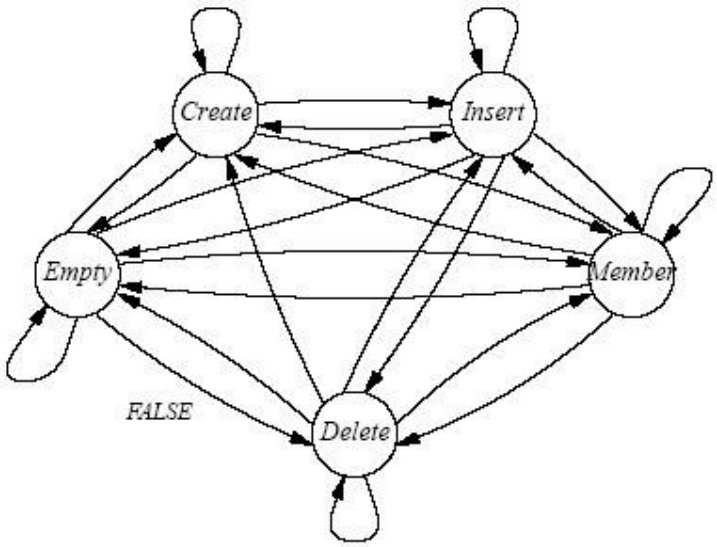
Celý tento postup nestačí aplikovať jedenkrát. Postup by mal začať od kroku 1 a postupne sa zjemňovať ku krokom 2 a 3. Tieto kroky by mali presnejšie objasniť príčinu chýb identifikovaných v kroku jedna, ale aj pomôcť objaviť ďalšie chyby. Po vykonaní potrebných úprav by testovanie malo prebiehať od krokov 2 alebo 3 nahor. Celý proces môže byť v závislosti od veľkosti produktu iteratívny. Samozrejme, ide iba o navrhnutý základný rámec testovania a vzhľadom k povahe produktu je možné ho rôznym spôsobom meniť.

Testovanie jednotlivých tried

Podľa [1] pri testovaní jednotlivých tried sa dá vychádzať zo skutočnosti, že trieda je implementáciou abstraktného dátového typu. Pred samotnými metódami testovania je

potrebné určiť, aké typy chýb môžu vzniknúť v triede. Trieda formálne pozostáva z množiny operácií a stavu, ktorý je určený hodnotami jej atribútov. Môže sa chybne správať buď z dôvodu dosiahnutia nesprávneho stavu, alebo z dôvodu chyby operácie. Triedu je možné formálne testovať nasledujúcimi 3 spôsobmi [1] :

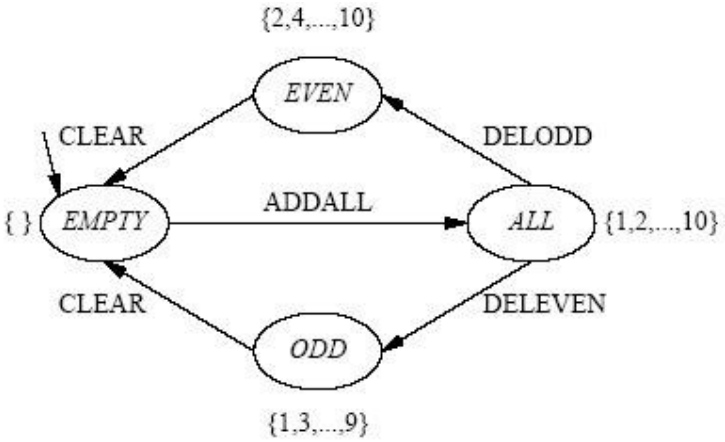
1. *Na základe orientovaného grafu operácií*. Pri tejto metóde je správanie triedy modelované orientovaným grafom, v ktorom uzlami sú operácie danej triedy a orientovaná hrana (o1,o2) symbolizuje, že operácia o2 môže byť zavolaná na základe volania operácie o1 (Obr. 1). Ako testovacie scenáre sa v tomto prípade používajú postupnosti operácií, ktorými sa snažíme pokryť všetky uzly, hrany alebo vetvy. Pre pokrytie všetkých uzlov by v tomto prípade mohla poslúžiť napríklad postupnosť : Create, Insert, Insert, Delete, Member, Empty. Analogicky sa dá vytvoriť postupnosť pre pokrytie všetkých hrán. Pre mnoho takto zostrojených grafov však platí, že sú cyklické, a preto nie je možné navrhnúť testovací scenár pokrývajúci všetky cesty. Dá sa však navrhnúť scenár, ktorý pokrýva cesty do určitej stanovenej dĺžky.



Obr. 1. Graf operácií pre množinu celých čísel [1]

2. *Prístup na základe grafu*. Napriek podobnému názvu je prístup pri tejto metóde úplne iný. Správanie triedy je v tomto prípade modelované pomocou testovacieho grafu (testgraph). Je to orientovaný graf, v ktorom uzly zodpovedajú stavom triedy a je určený začiatočný uzol zodpovedajúci začiatočnému stavu triedy (Obr. 2). Hrana (s1,s2) zodpovedá postupnosti operácií potrebných k prechodu zo stavu s1 do stavu s2. Testovacie scenáre

sa, podobne ako v predchádzajúcom prípade, volia tak, aby pokryli všetky uzly, vetvy a podobne. V tomto prípade napríklad scenár na pokrytie všetkých uzlov vyzerá nasledovne : <EMPTY, ALL, EVEN, EMPTY, ODD>



Obr. 2. Testovací graf pre množinu celých čísel [1]

3. *ASTOOT prístup*. ASTOOT znamená A Set of Tools for Object-Oriented Testing, čiže množina nástrojov pre objektovo orientované testovanie. Tento prístup je založený na formálnej algebraickej špecifikácii triedy. Prístup je založený na nasledujúcich pravidlách
   - Dve postupnosti operácií S1 a S2 sú ekvivalentné, ak S1 môže byť redukované na S2 uplatnením axióm špecifikácie.
   - Dva objekty O1 a O2 sú ekvivalentné práve vtedy a len vtedy, keď sú v rovnakom abstraktnom stave
   - Implementácia triedy je správna práve vtedy a len vtedy, keď každé dve ekvivalentné postupnosti operácií S1 a S2 spôsobujú ekvivalentnú zmenu stavu. To znamená, že ak existujú objekty O1 a O2, ktoré sú v počiatočnom stave ekvivalentné, tak tieto objekty zostávajú ekvivalentné aj po uplatnení postupnosti S1 na jeden a S2 na druhý z nich.

Testovací scenár v tomto prístupe má tvar (S1, S2, príznak), kde S1 a S2 sú postupnosti operácií a príznak je buď ekvivalentné, alebo neekvivalentné, v závislosti od toho, či S1 a S2 majú byť podľa špecifikácie ekvivalentné. Postupnosť S1 sa nazýva originálna postupnosť, postupnosť S2 zjednodušená postupnosť.

Vplyv zmien v programe na testovanie

Každý program prechádza procesom určitého vývoja. U niektorých je tento proces triviálny a po dokončení programu a jeho otestovaní už nedochádza k zmenám v programe. V mnohých prípadoch si však okolnosti vyžiadajú doplnenie špecifikácie a následnú požiadavku na zmenu hotového programu. Samozrejme, zmena programu si vyžaduje jeho opätovné testovanie. Takéto testovanie sa zvykne nazývať regresné testovanie. Nie je však múdre, vymýšľať všetky testy úplne odznova. Vo väčšine prípadov je možné aspoň časť z testov pôvodného programu použiť znova, či už v pôvodnom stave, alebo po určitej zmene. Vhodným testovaním len zmenených, alebo pridaných častí môže byť ušetrené veľké množstvo času. Je vhodné si identifikovať, akým spôsobom rôzne zmeny programu vyžadujú zmeny testovacích scenárov.

V kontexte objektovo orientovaného programovania sú z tohto pohľadu zaujímavé najmä dva základné princípy tejto paradigmy : dedičnosť a polymorfizmus. Podľa [2] je potrebné brať do úvahy nasledujúce fakty :

1. Keď je triede pridaný potomok alebo predok triedy, tak metódy ktoré sú dedené musia byť znovu otestované.
2. Aj ak nová a stará metóda sú sémanticky podobné, je potrebné tieto zmenené metódy znovu otestovať.
3. Ak je poradie predkov triedy zmenené je potrebné túto triedu znovu otestovať.

Je potrebné vhodne identifikovať zmeny v programe na to, aby mohli byť identifikované testovacie scenáre, ktoré je treba zmeniť, prípadne či je potrebné doplniť nové testovacie scenáre. Na to je potrebné identifikovať vzťahy medzi triedami ako v starom, tak aj v novom programe. Porovnávanie samotných zdrojových kódov by bolo veľmi náročné a neefektívne, úspešne sa na popísanie týchto vlastností a aj identifikovanie zmien dá použiť diagram tried. Prostredníctvom zmien vo vzťahoch medzi triedami je identifikovaný dopad zmien v programe.

Už teda vieme identifikovať zmeny vo vzťahoch medzi triedami, otázkou však je, aké zmeny testov si tieto zmeny vzťahov vyžadujú. Je samozrejmé, že pokiaľ v programe pribudnú úplne nové triedy, či celé skupiny tried, je potrebné pre ne vyvinúť samostatné testovacie scenáre. Niektoré triedy však nie sú nové, môžu ale obsahovať nové metódy. Je potrebné otestovať nielen nové metódy, ale aj metódy, ktoré tieto nové metódy využívajú.

Údržba softvérového produktu je väčšinou najdlhšie trvajúcou fázou jeho životného cyklu. Z už spomenutých skutočností vyplýva, že zmeny počas tejto fázy často vyžadujú uplatnenie princípov regresného testovania. K tomu, aby toto testovanie bolo efektívne a zaručovalo dosiahnutie požadovanej kvality, je potrebné mať nielen dôsledne zdokumentovaný samotný produkt a vhodne navrhnuté testy, ale aj zdokumentované na čo ktorý test slúžil, čiže kvalitu ktorej časti produktu overoval.

Záver

Pri objektovo orientovaných softvérových produktoch, tak ako pri každých iných produktoch, je testovanie nutnou podmienkou k zabezpečeniu požadovanej kvality produktu. V práci som spomenul viaceré špecifiká testovania takéhoto produktu. Mnohé zo spomenutých metód je možné aplikovať, vždy je však nutné brať do úvahy špecifiká konkrétneho vyvíjaného produktu. Aj v oblasti testovania softvéru však dochádza k neustálym zmenám, vyplývajúcim aj zo zmien v celej oblasti. Je preto potrebné udržiavať si v týchto metódach prehľad a vyberať si vhodné pre daný produkt.

Použitá literatúra

1. Gu D., Sarwar A., Zhong Y.: On Testing of Classes on Object-Oriented Programs. In: Proceedings of the 1994 conference of the Centre for Advanced Studies on Collaborative research, 1994
2. Chen C., Gao J., Hsia P., Kung David C., Toyoshima Y.: Object-Oriented Software Testing – Some Research And Development. In: Third IEEE International High-Assurance Systems Engineering Symposium, 1998

Annotation

Testing of Object-Oriented Software

Testing is an important phase of development of every software system, including object-oriented software development. However, testing of object-oriented software has its own special features, that testing of other software has not. This essay is concerned with importance of testing, different testing techniques and their possible advantages and disadvantages. Essay contains comparison of different testing strategies and approaches that are used by these strategies. Essay is concerned with testing of single classes based on their formal specification more in detail. In next part it is concerned with regression testing, its importance in the software maintenance phase and necessary assumptions for effective execution of regression tests.