„Čiernobiele“ testovanie

MICHAL BEBJAK

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

Abstrakt. Testovanie je dôležitou súčasťou životného cyklu softvérového projektu. Testovacie metódy v zásade rozdeľujeme na testovanie metódou čiernej resp. bielej skrinky. Táto práca sa zaoberá porovnaním oboch metód a techník používaných pri týchto metódach. V prípade metód čiernej skrinky sa zameriava na testovanie na základe prípadov použitia, rozšírených prípadov použitia, diagramu spolupráce a formálnej špecifikácie. Z metód bielej skrinky sa zameriava hlavne na testovanie modulov. Obe skupiny metód testovania majú rôznu oblasť pokrytia a používajú sa pri rôznych metódach tvorby softvéru. Môžu sa však vhodne dopĺňať. V práci sú uvedené aj skúsenosti s jednotlivými metódami testovania.

Úvod

Testovanie je dôležitou súčasťou životného cyklu softvérových systémov. Je potrebné pre vyhodnotenie správnosti, kompletnosti a kvality vytváraného softvérového systému. V zásade existujú dve možnosti pre testovanie softvéru. Prvou možnosťou je testovanie metódou bielej skrinky. Testovanie metódou bielej skrinky je založené na znalosti vnútornej štruktúry produktu a jeho zdrojového kódu. Druhou možnosťou je testovanie metódou čiernej skrinky. Táto metóda nepožaduje znalosť vnútornej štruktúry testovaného systému.

Táto esej prináša prehľad testovacích metód z jednej aj druhej skupiny. Jednotlivé testovacie metódy porovnáva na základe pokrytia a uvádza tiež skúsenosti s týmito metódami.

Testovanie metódou čiernej skrinky

Existuje viacero testovacích metód spadajúcich do kategórie metód čiernej skrinky. Na základe článku [1] som sa rozhodol pre porovnanie nasledujúcich metód:

- Testovanie založené na prípadoch použitia
- Testovanie založené na diagramoch spolupráce

Manažment v softvérovom inžinierstve, október 2006, s. 1-8.

- Testovanie založené na rozšírených prípadoch použitia
- Testovane založené na formálnej špecifikácii (OCL alebo ObjectZ)

Všetky tieto metódy sú zamerané na Objektovo-orientovaný softvér a z toho aj vychádzajú ich vstupné údaje. V článku [1] boli tieto metódy porovnávane na dvoch systémoch. Prvým systémom bol jednoduchý systém pre bankomat. Druhým bol systém pre plánovanie stretnutí. V tejto eseji budú uvedené výsledky dosiahnuté pri použití testov na druhom systéme.

Testovanie založené na prípadoch použitia

Pri tejto metóde sa jednotlivé testy tvoria z jednoduchého diagramu prípadov použitia. V prvom kroku je potrebné popísať základné funkcie systému a postupy, ktoré vedú k využitiu funkcii systému. V ďalšom kroku sa popíšu vstupné, očakávané výstupné údaje a možné výnimočné stavy. Testy potom kontrolujú funkcionalitu systému popísanú v diagrame prípadov použitia.

Testovanie založené na diagramoch spolupráce

Diagramy prípadov použitia zobrazujú interakcie medzi objektami. Testy pozostávajú zo vstupných údajov a sekvencie operácii vybranej z diagramu prípadov použitia.

Testovanie založené na rozšírených prípadoch použitia

Pri tvorbe testov sa v prvom kroku vyberie prípad použitia. V druhom kroku sa vytvorí scénar testu. Scénar sa tvorí za pomoci diagramu spolupráce a pozostáva zo sekvencie volaní metód jednotlivých tried. Nakoniec sa určia vstupné a výstupné údaje pre test.

Testovanie založené na formálnej špecifikácii

Táto metóda vyžaduje aby boli všetky objekty formálne špecifikované. Formálna špecifikácia sa tranformuje na stavový diagram zobrazujúci možné prechody medzi jednotlivými stavmi objektu. Test sa vytvorí na základe zvolenia jednej cesty zo stavového diagramu a určenia vstupných a výstupných údajov. Pre formálnu špecifikáciu je možné použiť jazky ObjectZ a OCL.

Výsledky testovania metódou čiernej skrinky

Každá metóda testovania generuje rozdielne testy či už sa jedná o formu testu, objem testovacích údajov alebo rozsahu pokrytia. V tejto kapitole budú porovnané jednotlivé metódy najmä na základe pokrytia.

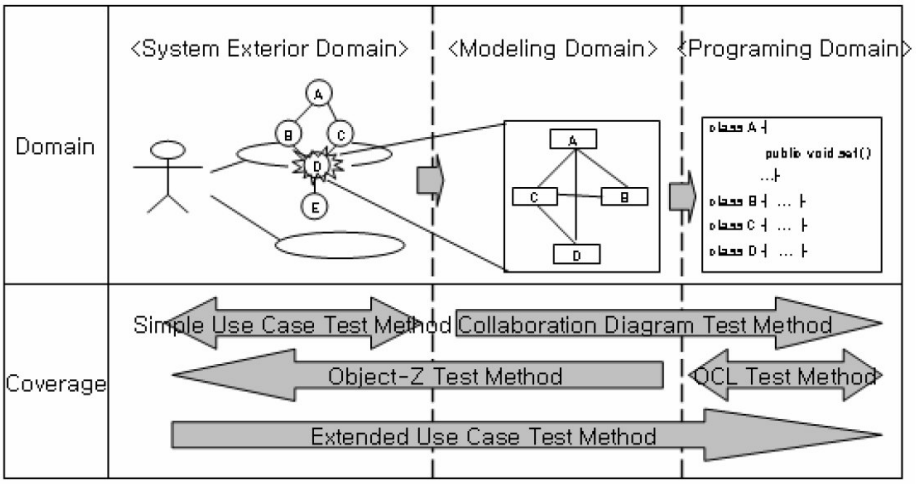
Obrázok 1 zobrazuje pokrytie oblastí systému jednotlivými testovacími metódami. Obrázok bol prebratý z článku [1] a preto sú popisy v anglickom jazyku. Rozdeľuje systém na tri oblasti:

- oblasť vonkajšieho rozhrania systému (system exterior domain)
- oblasť modelu systému (modeling domain)

- implementačná oblasť (programming domain)

Zobrazuje pokrytie nasledovných metód testovania:

- testovanie založené na prípadoch použitia (simple use case test method)
- testovanie založené na rozšírených prípadoch použitia (extended use case test method)
- testovanie založené na diagramoch spolupráce (collaboration diagram test method)
- testovanie založené na formálnej špecifikácii v jazyku ObjectZ / OCL (ObjectZ / OCL test method)



Obr. 1. Oblasti pokrytia jednotlivých metód [1]

Metóda testovania založená na prípadoch použitia pokrýva iba oblasť vonkajšieho rozhrania systému a vôbec neberie so úvahy ďalšie oblasti. Metóda testovania založená na rozšírených metódach použitia, vychádzajúca z metódy testovania založenej na prípadoch použitia, berie do úvahy aj ďalšie oblasti systému. Táto metóda pokrýva všetky oblasti a to oblasť vonkajšieho rozhrania, oblasť modelu a aj implementačnú oblasť. V prípade, že sa počas testovania nájde chyba, vytvoria sa ďalšie testy, ktorá sa zamerajú konkrétnejšie na túto chybu. Napríklad: ak sa nájde chyba v module D vonkajšieho rozhrania, nový test sa vytvorí špecifickejšie a bude testovať kód prislúchajúci k modulu D.

Testovanie založené na formálnej špecifikácii jazykom ObjectZ sa zameriava najmä na oblasť vonkajšieho rozhrania a oblasť modelu systému. Špecifikácia OCL generuje testy zamerané na implementačnú oblasť.

Metóda testovania na základe špecifikácie v jazyku ObjectZ aj metóda testovania založená na sekvenčných diagramoch vychádzajú z oblasti modelu systému. Rozdiel je v tom, že metóda testovania na základe diagramu spolupráce sa ďalej zameriava na implementačnú oblasť systému. Táto metóda je vhodná pokiaľ sa testovanie zameriava

na vnútorný beh programu – tok riadenia a údajov v jednotlivých objektoch. Pokiaľ chceme zamerať testovanie na funkcionalitu systému, je vhodné použiť metódu testovania na základe formálnej špecifikácie. Táto metóda sa totiž ďalej zameriava na oblasť vonkajšieho rozhrania systému.

Pokrytie jednotlivých metód testovania

Článok [1] uvádza aj percentuálne pokrytie jednotlivých metód testovania. Tieto výsledky pochádzajú z aplikácie jednotlivých metód na systéme pre plánovanie stretnutí. Výsledky je možné vidieť v tabuľke Tab. 1. V prvom riadku sú uvedené jednotlivé metódy testovania. V prvom stĺpci sú uvedené skúmané oblasti (číslo v zátvorke uvádza celkový počet entít v danej oblasti).

	Testovanie založené na prípadoch použitia	Testovanie založené na diagramoch spolupráce	Testovanie založené na špecifikácii v jazyku ObjectZ	Testovanie založené na špecifikácii v jazyku OCL	Testovanie založené na rozšírených prípadoch použitia
Zasiahnuté premenné (59)	33	25	23	47	53
Zasiahnuté motódy (51)	12	26	30	26	36
Počet zasiahnutých bodov (110)	45	51	53	73	89
Pokrytie	41%	46%	48%	66%	81%

Tab. 1. Pokrytie jednotlivých metód testovania [1]

Z tabuľky vyplýva, že najväčšie pokrytie dosahuje metóda testovania založená na rozšírených prípadoch použitia. Pokrytie tejto metódy je 80%, čo je skoro dvojnásobok pokrytia ostatných metód. Dobré výsledky dosahuje ešte metóda testovania založená na formálnej špecifikácii v jazyku OCL. Táto metóda dosahuje pokrytie 66%.

Dôvodom pre vysoké percento pokrytia pri testovaní metódou založenou na rozšírených prípadoch použitia je to, že táto metóda zahŕňa testovanie v oblasti modelu a vonkajšieho rozhrania, ako aj v implementačnej oblasti. Tým, že táto metóda zahŕňa aj implementačnú oblasť preniká do skupiny testovaní metódou bielej skrinky.

Testovanie metódou bielej skrinky

Testovanie metódou bielej skrinky vychádza zo znalosti vnútornej štruktúry a zdrojového kódu testovaného systému. Testy z tejto skupiny sa snažia pokryť čo najväčšiu časť zdrojového kódu. Do tejto skupiny zaraďujeme nasledovné metódy testovania:

- testovanie modulov
- statická a dynamická analýza zdrojového kódu
- pokrytie príkazov
- pokrytie vetiev

Statická a dynamická analýza zdrojového kódu

Pri statickej analýze sa postupne prechádza kód a hľadajú sa možné chyby. Pri dynamickej analýze sa tento kód vykonáva a tiež sa hľadajú možné chyby.

Pokrytie príkazov

Pri tejto metóde sa testy vytvárajú tak, aby sa každý príkaz zdrojového kódu vykonal aspoň raz.

Pokrytie vetiev

Pri tejto metóde sa testy vytvárajú tak, aby sa pokryli všetky vetvy programu.

Testovanie modulov

Testujú sa jednotlivé moduly systému. Tieto testy sa vytvárajú súčasne s vývojom jednotlivých modulov. Táto metóda bude bližšie popísaná v nasledujúcej kapitole.

Testovanie modulov

Testovanie modulov (unit testing) patrí do skupiny testovania metódou bielej skrinky. V článku [2] sú uvedené skúsenosti s testovaním modulov. Na základe tohoto článku sa pokúsim definovať, čo testovanie modulov je, kedy sa používa a aké sú jeho výhody a nevýhody.

Podľa [3] je testovanie modulov testovanie vykonávane vývojárom za účelom dokázania funkčnosti a správnosti modulu vzhľadom k špecifikácii modulu. Z prieskumu realizovaného v [2] vyplynulo, že testovanie modulov je najčastejšie chápané ako testovanie najmenších samostatných modulov systému. Testovanie modulov by sa malo zamerať na jeden modul a ignorovať zvyšok systému. Jednotlivé testy môžu byť vykonávané samostatne alebo pomocou automatizovaného prostredia. Automatizácia testov je želanou vlastnosťou.

Vytváranie testov, je na rozdieľ od vytvárania testov skupiny testovania metódou čiernej skrinky, v zodpovednosti vývojóvého oddelenia a nie testovacieho oddelenia. Tieto testy zvyčajne píšu a vykonávajú sami vývojári. Testy sa spúšťajú súčasne s vývojom systému a tým dávajú vývojárom rýchlu odozvu. Testy sa zvyčajne realizujú v čase od pár sekúnd po niekoľko minút. Rozsiahlejšie skupiny testov však môžu bežať aj niekoľko hodín.

Hlavným dôvodom pre vytváranie testov je zabezpečenie a dokázanie funkcionality jednotlivých modulov. Testovanie modulov testuje, či modul spĺňa čo by podľa vývojára spĺňať mal. V testami riadených metodológiach tvorby softvéru sa tieto testy používajú ako špecifikácia pre vytváranie modulov. Modul je hotový, ak splní všetky podmienky testu.

Výhody testovania modulov

Testovanie modulov je možné využiť aj na testovane externých modulov, prípadne modulov tretích strán. Takýmito testami je možné odhaliť chyby resp. zmeny v externých moduloch využívaných našim systémom. Toto umožňuje skontrolovať, či je možné použiť novú verziu modulu. Podľa [2] sa testovanie externých modulov, aj napriek svojim výhodám, používa zriedka.

Testovanie modulov umožňuje zaviesť automatizáciu testovania. Testovanie modulov je možné integrovať s build systémom, prípadne vykonávať testy v pravidelných intervaloch. Automatizácia je podľa [2] želaná vlastnosť. Jej nasadenie vyžaduje ďalšie zdroje, ale prínosy sú vo väčšine prípadov väčšie ako náklady.

Testy vytvárajú vývojári, ale k vytváraniu testov je možné prizvať aj testovacie oddelenie, prípadne použiť testy tretích strán. Toto zvyšuje kvalitu a rozsah testovania.

Automatizované a komplexné testy umožňujú kontrolovať funkčnosť pri veľkých zmenách kódu, akým je napríklad refaktorizácia. Návrh systémov je málokedy dokonalý. Moduly nie sú vždy úplne oddelené a testovanie modulov dokáže odhaliť, či zmeny v jednom module neovplyvnili funkčnosť ostatných modulov.

Nevýhody testovania modulov

Podľa [2] je najväčšou slabosťou testovania modulov nemožnosť resp. zložitosť testovania grafického používateľského prostredia (hlavne čo sa týka automatizácie týchto testov).

Problémom pri testovaní modulov je tiež určenie a vymedzenie modulov. V praxi sa často namiesto jednotlivých modulov testujú skupiny modulov. Pri moduloch, ktoré sú závislé na veľkých dátových štruktúrach je problémom aj vytvorenie vhodných vstupných údajov.

Automatizácia testovania modulov je želanou vlastnosťou, ale jej zavedenie a udržiavanie vyžaduje veľké zdroje. Po čase vzniká veľké množstvo testov, ktoré je potrebné vyvíjať spolu so systémom.

Testy vytvárajú vývojári. Toto má výhodu, že poznajú zdrojový kód, ktorý testujú. Dôkladná znalosť kódu môže viesť k tomu, že nemusia byť schopní vytvoriť kritické testy pre daný modul. Pokiaľ neexistuje metodika pre vytváranie testov, nie je

možné zaručiť, že budú vytvorené testy dostatočné. Rovnako môže nastať situácia, že sa vytvára veľké množstvo testov, ktoré sú buď duplicitné, alebo zbytočné.

Dôležitou časťou testovania, je určenie rozsahu testovania. Na určenie rozsahu sa používajú rôzne metriky (napr. pokrytie, prípady použitia, ...). Podľa [2] je určenie rozsahu problém a väčšina spoločností nemá pre túto oblasť jednoznačné kritériá.

Pre testovanie modulov neexistujú modely určujúce náklady a prínosy z takéhoto testovania. Toto sťažuje rozhodnutia o zavedení testovnia. Problémom je tiež motivácia vývojárov vytvárať testy.

Použitie jednotlivých metód testovania

Pre každý softvérový projekt je potrebné vybrať vhodnú metódu testovania. Uvedené metódy testovania „čiernou skrinkou“ sú vhodné najmä pre projekty vyvíjané štandardným spôsobom (analýza, návrh, implementácia, testovanie) s dlhými vývojovými cyklami. Vyžadujú kvalitnú, kompletnú a aktuálnu dokumentáciu a modely.

V súčasnosti sa však čoraz častejšie začínajú presadzovať agilné metódy vývoja softvéru. Tieto metódy sa podľa môjho názoru používajú hlavne pri projektoch menšieho rozsahu. Pri týchto projektoch by bolo vytváranie a udržiavanie kompletnej dokumentácie a modelov zbytočné. Pri agilných metódach sú jednotlivé cykly skrátené a zmeny sú veľmi časté. Tu je podľa mňa vhodné použiť metódu testovania modulov, čo je testovanie „bielou skrinkou“. Jednotlivé testy môžu slúžiť ako špecifikácia pre vytvárané moduly a zároveň aj ako nástroj pre ich overovanie a testovanie. Dôležitým faktorom je aj motivácia vývojárov. Vytváranie testov predstavuje prácu „navyše“ a preto je potrebné vývojarov o ich užitočnosti presvedčiť. Pokiaľ nebudú sami schopní rozpoznať výhody tohoto spôsobu testovania, nebudú mať dôvod poctivo vytvárať tieto testy. V prípade, že sa projekt začína na zelenej lúke, je vhodné aplikovať testovanie modulov už od začiatku. Väčšina projektov však vychádza z už existujúceho, zdedeného kódu. V tomto prípade je nasadenie testovania modulov zložitejšie, ale možné. V prípade agilných metód však má význam aj testovanie „čiernou skrinkou“. Najužitočnejšiou z metódou tejto skupiny je podľa mňa metóda testovania založená na prípadoch použitia a jej rozšírenie, metóda testovania založená na rozšírených prípadoch použitia. Tieto metódy nájdu uplatnenie najmä pri integrácii systému a vo fázach pred jeho dokončením. Metóda testovania založená na rozšírených prípadoch použitia umožňuje vyvinúť tlak na vytvorenie ďalších testov modulov v častiach systému, ktorý neprešiel testami vyššej úrovne.

Záver

V tejto eseji boli popísané a zhodnotené jednotlivé metódy testovania, ich pokrytie, výhody a nevýhody. Uvádza tiež skúsenosti s použitím týchto metód. V poslednej kapitole uvádza návrh použitia jednotlivých metód pri agilnej metóde vývoja softvéru.

Použitá literatúra

1. Kwang Ik Seo, Eun Man Choi: Comparison of Five Black-box Testing Methods for Object-Oriented Software. *Proceedings of the Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06)*
2. Per Runeson: *A Survey of Unit Testing Practices*. IEEE Software, July/August 2006.
3. K T. Koomen and M. Pol, *Test Process Improvement—A Practical Step-by-Step Guide to Structured Testing*, Addison-Wesley, 1999.

Annotation

"Black and white box" testing?

Testing is an important part software developement cycle. There are two main groups of testing methods. First one is a black-box testing and the second one is a white-box testing. In this essay both groups are compared and techniques for these methods are explained. In case of black-box testing, stress is given on use case, extended use case, collaboration diagram and formal specification testing. Unit testing is explained and reviewed as one of the white-box testing methods. Both groups of testing methods have different coverage and are used in different software developement methodologies. But if right combined, they can help in assuring software quality. In this paper, practical experience, strengths and weaknesses of this methods are explained.