Testovanie – metóda zabezpečenia kvality softvérového produktu

MIROSLAV JAKUŠ

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

Abstrakt. Vytvoriť softvér načas a dodržať pritom rozpočtové obmedzenia nie je v súčasnej dobe postačujúce, ak je výsledný produkt plný chýb a nedostatkov. Zákazník stále viac požaduje nejaké zabezpečenie, či priamo predvedenie kvality kupovaného softvéru. Preto sa čoraz väčší dôraz kladie na výskum rôznych metód a techník zlepšujúcich kvalitu vyvíjaného softvéru, jej meranie alebo zabezpečenie. Primárnou validačnou a verifikačnou technikou zabezpečovania kvality je testovanie, a teda zastáva dôležitú úlohu počas celého životného cyklu vyvíjaného softvéru – dať vývojárovi istotu, že vyvíjaný produkt je kvalitný a vhodný na ďalšie šírenie. Táto práca prezentuje pohľad na kvalitu softvérového systému, jej zabezpečenie a rozoberá najpoužívanejšie metódy a techniky testovania kvality softvérového systému.

Úvod

Každý z nás si je veľmi dobre vedomý kritických problémov spojených s vývojom softvérových systémov – odhadované ceny za vývoj a samotnú prevádzku sú prekračované, dodanie je častokrát odložené, a nakoniec, systém po dodaní nepracuje adekvátne požiadavkám. Takýto softvér sa potom stáva kritickým prvkom väčšiny systémov, do ktorých bol začlenený, z dôvodu jeho ceny a kritickej funkcionality. Už pri tejto úvahe jasne vidíme, že je potrebné overiť a zabezpečiť kvalitu vyvíjaného softvéru ešte skôr, ako náklady na zabezpečenie kvality a korektnej funkcionality softvéru prekročia únosné medze a odradia zákazníka.

Vymedzenie kvality je kľúčovým faktorom v našom každodennom živote. Či už zoberieme do úvahy športové súťaže, súťaže krásy alebo súťaže v ochutnávaní vína, kvalita je v nich meraná v podstate priamo – jednoduchým porovnávaním podľa dopredu určeného spôsobu. Napríklad víno môžeme porovnávať na základe chuti, farby, vône,... Avšak takýto spôsob porovnávania je príliš subjektívny – čiže, ak chcem aby takéto meranie malo nejakú váhu, musí byť vykonávané expertom.

Manažment v softvérovom inžinierstve, október 2006, s. 1-9.

Subjektivita a špecifickosť zohrávajú svoju rolu aj pri určovaní kvality softvérových systémov. Na vyriešenie tohto problému teda potrebujeme presnejšie definovať kvalitu softvéru ako takú, a taktiež aj spôsob merania tejto kvality na zabezpečenie objektívnej analýzy. Tu sa teda musím spýtať – dá sa vôbec kvalita softvéru odmerať? Odpoveď na túto otázku je áno, dokonca bol publikovaný aj pokus o zautomatizovanie procesu merania softvéru [1].

Kvalita softvéru

Na odhad kvality softvérových systémov môžeme použiť veľké množstvo rôznych atribútov. Veľa z nich je považovaných za systémové – môžeme ich aplikovať na celý systém, prípadne sú preklenuté cez viaceré časti softvérového systému. Čo však definuje dôležitú množinu kvalitatívnych atribútov je závislosť na perspektíve nezaujatého pohľadu. Môžeme tu teda sledovať akýsi stret viacerých záujmov – zatiaľ čo zákazník kladie väčší dôraz na výkon a používateľnosť softvéru, riadenie vývoja môže považovať za dôležitejšie bazírovať na zabezpečení vyššieho stupňa udržovateľnosti a znovupoužiteľnosti systému.

Preiss a kol., uviedli v [6] klasifikáciu kvalitatívnych atribútov na dve základné triedy:

- Atribúty postrehnuteľné v systéme pri spustení. Môžeme ich pozorovať počas behu systému. Sú úzko späté so správaním systému, preto ich musíme zahrnúť do špecifikácie správania systému, a teda musíme ich uvažovať už pri návrhu správania systému
- Atribúty nepostrehnuteľné za behu systému. Zvyčajne ich postrehneme až počas samotného životného cyklu systému (napr. počas udržovania systému). Avšak aj napriek tomu je potrebné uvažovať ich pri navrhovaní systému.

Najčastejšie používané kvalitatívne atribúty sú zobrazené v tabuľke Tab. 1 tak, ako ich uviedol Preiss a kol. v [6]. Treba však poznamenať, že odhad úrovne dosiahnutej kvality je vždy kontextovo závislý a častokrát veľmi subjektívny.

Not observable at runtime, but over product life-cycle		*Observable at runtime*	
Main Category	**Sub-category**	**Main Cate-gory**	**Subcategory**
Testability	Accoun-ability	*Use-ability*	Accessibility
Portability	Mobility		Administrability
	Nomadicity		Understand-ability
Integra-bility	Compose-ability	*Depend-ability*	Availability
	Interoper-ability		Degradability
	Adaptability		Durability
	Openness		Reliability
Maintain-ability	Evolvability		Stability
	Extensibility		Survivability
	Modifiability		Fault tolerance
	Changability	*Security*	
	Upgradability	*Safety*	
	Tailorability	*Perfor-mance*	Responsiveness
Reuse-ability			Accuracy
Deploy-ability	Distribute-ability		Footprint
	Configure-ability		Schedulability
			Scalability
			Coherency
			Timeliness
			Integrity

Tab. 1. Rozdelenie kvalitatívnych atribútov na triedy.

Verifikácia a Validácia

Tieto pojmy úzko súvisia s pojmom kvalita softvéru. Sú to vlastne metódy, príp. postupy, pomocou ktorých sa snažíme odhadnúť kvalitu vyvíjaného softvéru. Ako som už spomínal v úvode, kvalitu vyvíjaného softvéru je potrebné zabezpečiť už počas jeho vývoja a nie až kompletným testovaním pred odovzdaním. Na to nám slúžia práve verifikácia a validácia (V&V).

Softvérová V&V je vlastne nejaká množina aktivít, ktorých cieľom je podporiť kvalitu vyvíjaného softvéru priamo počas vývoja. Je definovaná štandardom IEEE Std. 1012 pre validáciu a verifikáciu softvéru [3]. I napriek tomu, že poznáme viacero

spôsobov definície životného cyklu programu, môžeme s určitosťou vymedziť niekoľko fáz, ktoré sa nachádzajú v každej takejto definícii:

- *Definícia požiadaviek,*
- *Analýza,*
- *Návrh,*
- *Implementácia,*
- *Testovanie*

V každej fáze vývoja môžu nastať chyby, ktoré v konečnom dôsledku ovplyvnia kvalitu výsledného produktu, a taktiež aj cenu, či časový plán vývoja. ***Verifikácia*** slúži na odhalenie a prípadnú korekciu chýb počas každej fázy vývoja, čiže vlastne verifikáciou môžem zistiť, či ten čiastkový produkt každej fázy spĺňa požiadavky preň vymedzené. Avšak ani takáto verifikácia mi nezabezpečí, aby výsledný produkt splnil určené požiadavky. ***Validácia*** sa väčšinou používa až po ukončení fázy vývoja softvéru, na zisťovanie, do akej miery vytvorený produkt spĺňa potreby pre zamýšľané použitie (teda, či vlastne výsledný produkt robí v skutočnosti to, čo má).

Testovanie

Ako už Pressman v [8] prehlásil:

"Testovanie softvéru je kritickým prvkom zabezpečenia kvality softvéru a reprezentuje konečné zhodnotenie špecifikácie, návrhu a implementácie.."

S týmto výrokom sa myslím že môžeme stotožniť úplne všetci. Ak si chcem byť istý, že môj softvér je vyvinutý bez chýb, musím jednoducho vykonať potrebné testovania. Ak by som chcel predať produkt, ktorý by nebol dostatočne odladený a zbavený všetkých chýb a nedostatkov, musím rátať s tým, že mi takýto predaj viacej uškodí ako pomôže – minimálne v očiach zákazníka, ktorému sa takýto výrobok „pritrafí". Testovanie mi však neponúka úplnú istotu, že vyvinutý softvér je korektný. Dáva mi možnosť vykonať sériu experimentálnych vyhodnotení produktu.

Priame ciele testovania môžem zhrnúť do nasledovných bodov:

1. Testovanie je proces vykonávania (spúšťania) vytvoreného softvéru za účelom odhalenia chyby
2. Dobrý skúšobný prípad je taký, ktorý má vysokú pravdepodobnosť nájdenia doteraz ešte neodhalenej chyby
3. Úspešný test je taký, ktorý nájde doteraz ešte neodhalenú chybu

Ak použijem systematický prístup, testovaním vygenerujem aktivity cez celý životný cyklus softvéru. Celé to začína už pri vytvorení špecifikácie, kedy môžem začať odvodzovať akceptačné testy. Vytvorenie návrhu mi zasa dáva podklad pre zavedenie integračných a systémových testov. Dokončenie vývoja každého modulu

zasa spustí proces vytvárania a spustenia testov modulov. Navyše, hocijaká aktivita spojená s udržiavaním produktu by mala vyvolať opätovné vykonanie testov a prípadné aktualizovanie výsledkov tak, aby sa dala použiť metóda regresného testovania.

Testovacie metódy

Testovanie môžem vo všeobecnosti rozdeliť podľa spôsobu vykonávania testu na dve hlavné skupiny:

- Manuálne,
- Automatizované

V súčasnosti je oveľa viacej využívaná manuálna metóda testovania, kedy testovanie vykonáva skupina osôb – *testerov* priamou interakciou s testovaným produktom najčastejšie podľa dopredu stanoveného testovacieho postupu. Automatizovaný prístupe k testovaniu sa zavádza väčšinou do rozsiahlych projektov, kedy by nebolo možné manuálne otestovať všetky časti produktu – trvalo by to neúmerne dlho. Využije sa teda nejaký automatizovaný nástroj. Samotná automatizácia je rozdelená do viacerých kategórií, od nástrojov na správu testov až po nástroje na vykonávanie testov.

Ďalším dôležitým delením testovacích metód je delenie podľa spôsobu prístupu k vytváraniu skúšobných prípadov na:

- testovanie formou bielej skrinky,
- testovanie formou čiernej skrinky

Testovanie formou bielej skrinky

Táto metóda využíva vnútorný pohľad na systém pri návrhu skúšobných prípadov založených na vnútornej štruktúre. Vyžaduje si veľa skúseností s programovaním pri identifikácii všetkých možných ciest v systéme. Tester vyberá vstupy pre skúšobný prípad, vyskúša tieto vstupy a určí výstupy. Pri tejto metóde musím poznať zdrojový kód a musím mať k nemu prístup. Väčšinou sa táto metóda používa počas vývoja softvéru – pri krokovaní cez zdrojový kód.

Toto testovanie je silne závislé na implementácii konkrétneho problému. To v praxi znamená, že keď zmením implementáciu nejakej funkcionality v mojom programe, s veľkou pravdepodobnosťou budem musieť upraviť aj príslušný skúšobný prípad k nej prislúchajúci.

I napriek tomu, že túto formu testovania môžeme použiť na testovanie na úrovni modulov, integračnej úrovni alebo na úrovni systému, zvyčajne sa táto metóda používa len na úrovni modulov. Čiže, ak zvyčajne sleduje prepojenia v rámci modulov, môže byť rovnako dobre použitá na testovanie prepojení medzi modulmi počas integrácie, či medzi podsystémami počas systémového testu. Hoci tento spôsob návrhu testovania

odhalí obrovské množstvo skúšobných prípadov, nemusí odhaliť neimplementované časti systému, či chýbajúce požiadavky.

Testovanie formou čiernej skrinky

Táto metóda nevyžaduje znalosť zdrojového kódu systému počas vytvárania skúšobných prípadov. To znamená, že pri návrhu skúšobných testov sa použije externý pohľad na testovaný objekt. Tester nemá možnosť zistiť, akú má testovaný objekt vnútornú štruktúru, namiesto toho určí správne vstupy a k nim prislúchajúce nesprávne vstupy a počas testovania sleduje, či sa testovaný objekt správa podľa predpokladov – teda, či podáva korektný výstup

Tento prístup k návrhu testovania je aplikovateľný na všetky úrovne vývoja – moduly, integráciu, samotný systém a akceptáciu. Čím vyššia je úroveň, a teda aj skrinka je väčšia a komplexnejšia, tým viac sme nútení použiť testovanie formou čiernej skrinky, aby sme celý proces zjednodušili. Táto metóda zaručene odhalí neimplementované časti systému, avšak nie je isté, že sa preskúmajú všetky prepojenia v rámci systému.

Kwang a Eun vo svojej práci [5] empiricky porovnávajú päť rôznych spôsobov implementácie čiernej skrinky. Vybrali najpoužívanejšie formy čiernej skrinky a na základe testovania identického softvéru porovnávajú dosiahnuté výsledky. Zistili, že nimi porovnávané metódy testujú softvér na rôznych úrovniach kódových konštrukcií. Navrhujú preto pri plánovaní testovania využiť kombináciu metód OCL a metódu rozšírených prípadov použitia. Pre obšírnejší popis a detailnejšie zobrazenie ich výsledkov prosím pozri [5].

Najnovšie metódy testovania

V poslednej dobe sa veľmi rozšírilo programovanie pomocou objektov, a taktiež programovanie založené na komponentoch.

Gao a kol. sa vo svojej práci [2] venujú výskumu systematického regresného testovania pre softvér založený na komponentoch. V takýchto systémoch kvalita produktu závisí od kvality jednotlivých komponent. Pri ľubovoľnej zmene niektorej zo zahrnutých komponent, táto musí byť opäť otestovaná a zahrnutá medzi ostatné, aby vytvorili celkový systém. Venovali sa hlavne spôsobu zisťovania zmien v upravených komponentoch a vplyvu týchto zmien na úrovni modulov, a teda na základe toho hľadali opätovne použiteľné skúšobné prípady v sade skúšobných prípadov prislúchajúcich príslušnej komponente. Nimi popisovaná metóda poskytuje podporu pre zmenu testov a analýzu vplyvu pre sadu testov vo forme čiernej skrinky pre danú komponentu. Tento prístup implementovali do nástroja COMPTest.

OMEN prístup

Tento prístup predstavili Souter a Pollock v [7]. Predstavujú štrukturálny prístup k testovaniu objektovo orientovaných softvérov založených na pôsobení elementárnych „read/write“ operácií medzi objektami. Názov tomuto prístupu vytvorili ako skratku k anglickému názvu:

„Object Manipulations in addition to using Escape iNformation“ [7], čiže

Manipulácie s objektami v súvislosti s používaním informácie na poskytnutie relevantnej informácie testerovi v prostredí interaktívneho testovacieho nástroja. Vo svojej práci demonštrujú ukážku tohto princípu na konkrétnej špecifickej testovacej technike na základe pozorovania dátového toku a elementárnych „read/write“ operácií.

Deštruktívne testovanie

Tento prístup predstavuje novú paradigmu pre testovanie. Predstavil ho Kiumi Akingbehin v [4]. Pri tradičnom testovaní sa testuje, či softvér spĺňa požadovanú špecifikáciu. Pri deštruktívnom testovaní však toto nezachytíme. Princípom pri tejto paradigme je testovanie, či softvér podáva správne výsledky, keď mu na vstupe zadáme nesprávne hodnoty, príp. ak ho nesprávne používame. Deštruktívne testovanie však neslúži ako náhrada tradičných testovacích prístupov, naopak, dopĺňa konvenčné testovacie techniky.

Kiumi porovnáva deštruktívne testovanie s tradičnými testovacími technikami, a tieto rozdeľuje do troch tried podľa kompatibility s deštruktívnym testovaním:

- Trieda A – nekompatibilné s deštruktívnym testovaním
- Trieda B – po úprave kompatibilné s deštruktívnym testovaním
- Trieda C - kompatibilné s deštruktívnym testovaním

Na základe týchto tried rozdelil najpopulárnejšie testovacie techniky do prehľadnej tabuľky. Pre viac informácií ohľadne deštruktívneho testovania pozri [4].

Pretože deštruktívne testovanie nenahrádza tradičné testovanie, ale pôsobí len ako doplňujúce testovanie, nemôže byť škodlivé pre vyvíjaný softvér. Teda deštruktívne testovanie je pre vývoj softvéru prínosom.

Záver

Podľa môjho názoru, testovanie zastáva veľmi dôležitú úlohu v celom procese vývoja softvérového systému. Odovzdať neodladený produkt, plný chýb, do rúk klienta by znamenalo prakticky deštrukciu firmy na finančnom trhu. Už dlhšiu dobu pracujem v malej softvérovej firme. Projekty väčšinou riešime v 3-4 členných tímoch. Tak som mal možnosť sám na sebe odskúšať, aká je kvalita produkovaných častí celého systému dôležitá. Nestačí len dodržiavať štandardnú mennú konvenciu kvôli jednoduchšej integrácii jednotlivých častí do výsledného celku, ale treba udržiavať aj kvalitu týchto

častí na najvyššej úrovni. My sme na zabezpečenie kvality používali jednu z tu popísaných metód testovania, a to metódu testovania formou čiernej skrinky. Avšak nespoľahli sme sa len na čiernu skrinku, ale použili sme aj metódu deštruktívneho testovania, aby sa zabezpečilo korektné správanie systému aj pri nesprávnych vstupoch.

Použitá literatúra

1. Bernhard Daubner, "Empowering Software Development Environments by Automatic Software Measurement," metrics, p. 45, 11th IEEE International Software Metrics Symposium (METRICS'05), 2005.
2. Jerry Gao, Deepa Gopinathan, Quan Mai, Jingsha He, "A Systematic Regression Testing Method and Tool For Software Components," compsac, pp. 455-466, 30th Annual International Computer Software and Applications Conference (COMPSAC'06), 2006
3. IEEE, „IEEE standard for Software Verifikation and Validation“ in IEEE Std. 1012-1998, 1998.
4. Kiumi Akingbehin, "Towards Destructive Software Testing," icis-comsar, pp. 374-377, 5th IEEE/ACIS International Conference on Computer and Information Science and 1st IEEE/ACIS International Workshop on Component-Based Software Engineering,Software Architecture and Reuse (ICIS-COMSAR'06), 2006.
5. Kwang Ik Seo, Eun Man Choi, "Comparison of Five Black-box Testing Methods for Object-Oriented Software," sera, pp. 213-220, Fourth International Conference on Software Engineering Research, Management and Applications (SERA'06), 2006.
6. Otto Preiss, Jason Wong, Alain Wegmann, "On Quality Attribute Based Software Engineering," euromicro, p. 0114, 27th Euromicro Conference 2001: A Net Odyssey (euromicro'01), 2001.
7. Souter, A. L. and Pollock, L. L. 2000. OMEN: A strategy for testing object-oriented software. In Proceedings of the 2000 ACM SIGSOFT international Symposium on Software Testing and Analysis (Portland, Oregon, United States, August 21 - 24, 2000). M. J. Harold, Ed. ISSTA '00. ACM Press, New York, NY, 49-59.
8. Pressman, R. S., Software Engineering (3rd Ed.): A Practitioner's Approach, McGraw-Hill, Inc., 1992.

Annotation

Testing – method for software product quality assurance

To develop software system on time and within a budget is not good enough if software produced is full of defects. Customer wants to be assured of quality or even expect concrete

demonstration of software quality. Hence more software engineering research has focused on methodologies and techniques for improving software quality, measuring software quality and software quality assurance.

Primary task in verification and validation process for software quality assurance is testing, so it stands for very important role through whole lifecycle of software product – to make sure, that developed software is good enough to place it on market.

This essay provides an overview of software quality. The software quality assurance and most common methodologies and techniques for testing software quality are discussed.