Testovanie a prehliadka kódu alebo ako testovať softvér efektívne

JÁN KOHUT

Slovenská technická univerzita
Fakulta informatiky a informačných technológií
Ilkovičova 3, 842 16 Bratislava
jkohut[zavináč]atlas[.]sk

Abstrakt. Veľký vplyv na kvalitu softvéru má testovanie. V tejto práci predstavujem niektoré charakteristiky testovania. Testovanie je pomerne nákladná činnosť, preto sú snahy ho urobiť čo najefektívnejším. Jednou z techník, ktorá je pomerne efektívna, je prehliadka kódu. Objektovo orientované programovanie prináša so sebou viaceré úskalia, ktoré sťažujú prehliadku kódu. V tejto práci opisujem niektoré techniky zamerané práve na inšpekciu kódu napísaného v objektovo orientovanom jazyku. Čitateľ sa taktiež dozvie o ďalších technikách slúžiacich na testovanie, ich možnostiach a vhodnosti. Záver práce patrí mojim postrehom k testovaniu, jeho vhodnosti, rozsahu a skúsenostiam získaných pri tvorbe a testovaní viacerých projektov.

Úvod

Je známy citát Dijkstru [4] o softvérovom testovaní: „testovanie môže iba ukázať prítomnosť chýb, ale nikdy ich neprítomnosť.“ Testovanie sprevádza vývoj softvéru odkedy sa vyvíja softvér. Hľadanie chýb a ladenie programov sa stáva nevyhnutnou súčasťou procesu vývoja softvéru, takou neodmysliteľnou ako je napríklad samotná implementácia. Zameranie projektov kladie rôzne požiadavky na ich funkcionalitu a robustnosť, ale v konečnom dôsledku otestovať aspoň niektoré časti softvéru musí každý vývojár. To, čo slúži na preukázanie funkcionality v jednoduchom školskom projekte, nebude svojou kvalitou postačovať firemnému zákazníkovi, kde výpadky výroby v dôsledku zlyhania softvéru stoja peniaze. Bezpečnosť ľudských životov kladie ešte vyššiu požiadavku na čo možno najbezpečnejší softvér a zariadenia, ktoré obsluhuje. Tieto ale i iné dôvody vedú k snahám o čo najviac otestovaný softvér.

Testovanie softvéru pokrýva široké spektrum rozličných aktivít, od testovania malých kúskov kódu vývojárom po akceptačné testovanie zákazníkom veľkých informačných systémov. V rozličných stupňoch, testovanie by sa malo zacieliť na rôzne predmety záujmu ako sú napríklad testovanie so zameraním sa na:

- Používateľské požiadavky.

Manažment v softvérovom inžinierstve, október 2007, s. 1-8.

- Robustnosť v kritických situáciách alebo pri nesprávnych vstupoch.
- Výkonnosť.
- Použiteľnosť.
- Operačnú spoľahlivosť.
- A mnoho iných.

Testovanie prináša so sebou tieto otázky [2]:

- Prečo hľadáme chyby a testujeme?
- Ako robíme jednotlivé testy? Máme v tom nejaký systém alebo to robíme ad hoc?
- Ako veľa testujeme?
- Čo testujeme? Celý systém alebo iba jednotlivé časti?
- Kde vykonávame testovanie ? Pri vývoji alebo v prostredí kde bude systém nasadený?
- Kedy počas životného cyklu projektu vykonávame testovanie?

Postupom času vznikli tieto 4 sny, predstavy výskumníkov, ktorí sa venujú testovaniu.

- Univerzálna teória o testovaní.
- Testovo založené modelovanie.
- 100 % automatické testy.
- Maximálne efektívne testovacie inžinierstvo.

Testovanie z praktického pohľadu

Úrovne testovania

Testovanie softvéru je zvyčajne vykonávané na rozličných úrovniach vývojového cyklu a údržby softvéru. Cieľom testovania môže byť jednoduchý modul, skupina modulov alebo celý systém. Podľa týchto cieľov je možné testovanie rozdeliť na konceptuálne odlišné skupiny: Unit testy, integračné testy a systémové testy [3].

Unit testy
Unit testy verifikujú funkcionalitu izolovaných softvérových častí, ktoré sú samostatne testovateľné. Záleží na kontexte v ktorom je daný subprogram alebo veľký komponent

testovaný. Zvyčajne sa vykonávajú programátorom, kód je testovaný s podporou ladiacich nástrojov. Mal by ich vytvárať programátor, ktorý vytvoril daný kus kódu.

Integračné testy

Integračné testovanie je proces verifikácie interakcie medzi softvérovými komponentmi. Klasické integračné testovacie stratégie, ako zhora dole alebo zdola hore sa používajú s tradične, hierarchicky štruktúrovaným softvérom.

Systémové testy

Systémové testovanie sa zameriava na správanie celého systému. Väčšina chýb by mala byť už identifikovaná počas testovania komponentov a integračného testovania. Systémové testovanie sa obyčajne považuje za vhodné pre porovnanie vytvoreného systému s nefunkcionalnými požiadavkami ako napríklad bezpečnosť, rýchlosť, presnosť a spoľahlivosť.

Testovanie skrinky

Základná koncepcia testovania sa delí na testovanie bielej skrinky (white-box) a čiernej skrinky (black-box). V prvom prípade je stredobodom záujmu štruktúra, v druhom prípade funkcionalita softvéru. Okrem tohto pohľadu, t.j. štruktúra verzus funkcionalita sa používa aj tretia možnosť: šedá skrinka (grey-box). Je to akýsi prienik bielej a čiernej skrinky. Používa sa zvyčajne v aplikáciách typu klient - server, kde tester má kontrolu nad vstupom a výstupom, môže vkladať a prezerať hodnoty v SQL databáze. Podľa týchto dát vie posúdiť, či program pracuje správne.

Vyššie spomenuté rozdelenie testovania je iba jedným z viacerých spôsobov ako sa je možné pozerať na testovanie softvéru. Testovanie sa môže ďalej zameriavať na akceptačné testovanie, inštalačné testovanie, alfa a beta testovanie, výkonnosť, stresové testovanie, testy pri páde systému, regresné testovanie a mnohé iné.

Prehliadka kódu

Je asi prvou činnosťou, ktorá sa vykonáva na zabezpečenie určitej kvality kódu. Programátor si pozrie ako napísal zdrojový kód, pozrie sa, či je správne naformátovaný a je logicky správny. Podľa môjho názoru, ak sa zanedbá aspoň jednoduchá a rýchla prehliadka kódu, vzniknú oveľa vyššie náklady neskôr v ďalších fázach vývoja a testovania. Niekedy je ťažké vymyslieť a vykonať testy, resp. nimi pokryť všetky možnosti otestovania daného kusu kódu. Prehliadka ma oproti testom v tomto smere niektoré výhody.

Veľmi nápomocnou pri prehliadke kódu je podľa mňa podpora na strane vývojového prostredia a rôzne vizualizačné nástroje. Prehliadka klasického textu (zdrojového kódu) veľmi skoro znechutí každého, ale ak je zvýraznená syntax jazyka je to veľká pomôcka pre vývojára. Grafická vizualizácia niektorých častí kódu pracuje na vyššej úrovni ako sme schopní vidieť priamo v kóde a pomáha vidieť funkcionalitu kódu v širších súvislostiach aj s jeho chybami.

Podľa štúdie [1] zavedenie vhodných nástrojov pre analýzu a vizualizáciu zdrojových kódov môže byť efektívna pomoc na zlepšenie testovania softvéru a zvýšenie robustnosti systému.

Zvyčajne si pod pojmom prehliadka kódu predstavíme „prebehnutie“ zrakom po procedúrach a funkciách v zdrojovom kóde. Avšak v súčasnosti sa väčšina programov vytvára v objektovo orientovanom (OO) jazyku. To kladie na osobu, ktorá vykonáva prehliadku kódu vyššie nároky, lebo OO programovanie prináša do kódu viaceré aspekty, ktoré si treba pri tejto prehliadke uvedomiť. To, čím sa OO jazyky odlišujú od procedurálnych jazykov: skrývanie dát, dedičnosť, polymorfizmus, dynamické viazanie objektov, má vplyv na spôsob ako je OO systém štruktúrovaný a vykonávaný. Zle pochopenie OO paradigmy môže prekážať v efektívnej prehliadne OO systémov [5]. Na pochopenie jedného riadku kódu je niekedy potrebné prejsť a porozumieť viacerým metódam, triedam alebo knižniciam. Hoci podobný problém môže existovať aj v procedurálnom programovaní, tu je táto črta oveľa viac výrazná.

Na prehliadku OO kódov boli vyvinuté viaceré techniky, napr. abstraktne riadená inšpekcia (abstraction-driven inspection) ale už zo samotnej podstaty OO paradigmy nie je podľa mňa možné systematicky pokryť inšpekciou celý zdrojový kód, resp. 100% predpokladať ako sa bude daná časť kódu alebo celý program správať. Určité výhrady voči systematickým technikám vyjadril E.Gamma [6]: „dve štruktúry(run-time a compile-time) sú do značnej miery nezávislé. Snažiť sa porozumieť jednej z druhej je ako snažiť sa porozumieť dynamike živého ekosystému zo statickej taxonómie rastlín a zvierat alebo opačne.“

Tri čítacie techniky

Čítacie techniky definujú stratégiu na individuálnu prehliadku dokumentu. Pomáhajú nájsť chyby a môžu tiež zvýšiť efektívnosť tímu, ktorý vykonáva prehliadku zdrojových kódov. V práci [5] sú vybrané tri techniky:

- Kontrolný zoznam.
- Stratégia založená na prípadoch použitia.
- Systematická abstrakciou riadená technika.

Výber systematickej abstrakciou riadenej techniky odôvodnili autori faktom, že to aspoň nemôže byť horšie ako hľadanie chýb ad hoc a prináša benefit v lepšom porozumení kódu. Avšak systematický postup niekedy umožňuje nevšimnúť si jednoduché chyby. Na vybalansovanie systematickej techniky čítania je predstavená technika kontrolného zoznamu. Treťou stratégiou je stratégia založená na prípadoch použitia.

Kontrolný zoznam

Kontrolný zoznam je založený na sérii špecifických otázok, ktoré majú za cieľ upriamiť inšpektorovu pozornosť na bežné zdroje chýb. Podľa niektorých autorov [5] by kontrolný zoznam mal vychádzať z relevantných historických informácií a nemal by to byť kontrolný zoznam získaný odkiaľkoľvek, pretože môže strácať relevantnosť.

Základom kontrolného zoznamu sú dva komponenty. Prvým je list potenciálnych problémov, druhým rady ako identifikovať chyby v prípade každého problematického bodu. Otázky sú zvyčajne rozdelené do skupín podľa zamerania (dedičnosť, referencovanie dát, prekonávanie metód). Snahou stratégie kontrolného zoznamu je podporiť inšpektora v lepšom porozumení kódu. Ako inšpektor postupne prechádza cez rôzne skupiny otázok, postupne sa pohybuje z vyššej úrovne abstrakcie na detailnejšiu a jemnejšie zrnitú perspektívu. Každá zo skupiny otázok vyžaduje porozumieť viac a viac každej metóde. Nakoniec poslednej otázke v skupine otázok k metódam: „Vyhovuje metóda špecifikácii?“ by malo byť po zodpovedaní predchádzajúcich otázok ľahšie zodpovedať. V Tab.1 je kontrolný zoznam urobený podľa [5].

Pre každú triedu:		
	Črta	Otázka
1	Dedičnosť	Je dedičnosť úplne implementovaná podľa návrhu v triede?
2		Je primerane implementovaná?
3	Konštruktor	Sú všetky premenné inicializované na správne hodnoty?
4		Ak voláme *super*, je použite správne?
Pre každú metódu:		
6	Referencia dát	Sú všetky parametre použité v metóde?
...		
14	Správanie metód	Sú všetky kroky a stavy urobené správne?
15		Je návratový typ a jeho hodnota správna?
16		Vyhovuje metóda špecifikácii?
Pre každú triedu:		
17	Prekonávanie metód	Ak zdedené metódy sa potrebujú správať odlišne, sú prekonané?
		Sú všetky použité metódy prekonávané správne?
...		

Tab.1. Kontrolný zoznam [5]**.**

Stratégia založená na prípadoch použitia

Snahou stratégie založenej na prípadoch použitia je prispôsobiť sa dynamickej povahe OO systémov skúmajúc odpovede kódu pri inšpekcii typických vstupných scenároch. Cieľom techniky je overiť, či každý objekt je schopný reagovať správne na všetky možné spôsoby, ako môže byť použitý. Prípady použitia sa použijú už vytvorené prípady z fázy návrhu systému. Technika sa snaží prinútiť inšpektora uvažovať v kontexte v ktorom sa objekty používajú. To je v kontraste s technikami založenými na systematickom a kontrolnom zozname, ktoré uvažujú triedy vo viac všeobecnom

kontexte. Technika je vhodná ako doplnok k ďalším metódam prehliadky kódu, pretože ťažko je možné vytvoriť prípady použitia, ktoré by pokryli celý zdrojový kód. Problém môžu byť aj prípady použitia, ktoré nie vždy sú vytvorené vo fáze návrhu systému.

Systematická technika

Filozofiou systematickej techniky je prinútiť inšpektorov systematickým spôsobom ako budú prechádzať cez kód, vytvoriť abstraktnú špecifikáciu pre každú metódu. Základnou myšlienkou pre vytvorenie určitej abstrakcie je hlbšie porozumenie kódu a poskytnúť sumár metód pre referenciu v budúcej inšpekcii. Všetky aspekty metódy by mali byť systematicky prečítané a malo by im byť porozumené.

Porovnanie techník

Ako som už spomenul v texte predtým, tradičné techniky čítania nemusia byť vhodné a efektívne pre inšpekciu veľkých OO systémov. Na ich inšpekciu boli predstavené tri techniky. Každá z týchto techník je pomerne silná a pomáha nájsť určité druhy chýb, umožňujú lepšie pochopiť funkcionalitu a architektúru zdrojového programu. Podľa môjho názoru je najefektívnejšia metóda, v prípade ak je pre náš najdôležitejším kritériom cena testovania, systematická technika. Podobá sa inšpekcii procedurálnemu kódu, pričom ale uvažujeme objektovo. Kontrolný zoznam je pomerne náročný na čas, ale ak by človek s ním dlhšie pracoval, môže pochytiť niektoré aspekty testovania touto metódou a uplatniť ich pri iných technikách. Stratégia založená na prípadoch použitia je podľa mňa efektívna v aplikáciách kritických na chyby.

Testovacie techniky

V predchádzajúcej kapitole som sa venoval inšpekcii zdrojových kódov. Je to pravdepodobne najzákladnejšia a najefektívnejšia technika, ktorú musí vykonať v určitej miere každý programátor, ak chce zabezpečiť aspoň minimálnu kvalitu a funkčnosť kódu. Okrem inšpekcie kódu existuje mnoho techník slúžiacich na testovanie. Keďže nie všetky je možné použiť, v tejto kapitole predstavím aspoň niektoré kritéria a techniky pre výber toho „správneho“ prístupu, respektíve ich kombinácii.

Ad hoc testovanie

Najrozšírenejší prístup k testovaniu. Je založený na intuícii, skúsenostiach a zručnostiach testera. Môže byť nápomocný pre identifikovanie niektorých špecifických testov, ale keďže to nie je systematická technika, jej efektívnosť v podstatnej miere závisí od kvalít testera. Podľa mňa je najvhodnejšia na malé školské projekty. Hoci je to najprirodzenejší spôsob testovania, nemusí sa uplatniť vo firemnom prostredí pre svoju nesystémovosť a nedostatky.

Prieskumné testovanie

Prieskumné testovanie je definované ako simultánne učenie, návrh testov a vykonávanie testov. Plán testov sa dynamický vyvíja podľa získaných znalostí počas testovania. Medzi výhody tohto testovania patri potreba menších zdrojov na prípravu testov, ako by to bolo s pripraveným plánom testovania, nevýhodu zase, že je ťažké exaktne preukázať, ktoré testy sa skutočne vykonali.

Rozhodovacia tabuľka

Rozhodovacia tabuľka predstavuje logické vzťahy medzi podmienkami a akciami. Testovacie prípady sú systematický vytvorené na každú možnú kombináciu podmienok a akcií.

Analýza hraničných hodnôt

Ako vstupné údaje na testovanie sa vyberajú hraničné hodnoty z potenciálnej množiny vstupných dát, o ktorých sa predpokladá, že by mohli viesť k chybám. Rozšírením tejto techniky je testovanie robustnosti systému, kde sa zadávajú vstupy mimo domény vstupných hodnôt. Snahou tohto testovania je urobiť program robustnejším voči neočakávaným alebo chybným vstupom.

Kritéria založené na kontrole toku

Je to prístup, ktorý sa snaží k pokrytiu všetkých príkazov alebo blokov príkazov v programe. Sleduje v kóde ako sa vykonáva program, jeho vetvenia i podmienky. Ak takto sa skontroluje kód a všetky vetvy programu sa aspoň raz otestujú, hovoríme že je kód pokrytý na 100%. Štúdia [1] uvádza, že pri použití vizualizačných nástrojov zdrojových kódov, tie boli pokryté na 70 až 80%. Dosiahnutie vyššej účinnosti je často veľmi náročné, pretože architektúra systému robí takmer nemožné pokryť určité časti kódu.

Záver

V práci som predstavil základné myšlienky a ciele testovania i niektoré techniky, ktoré sa venujú testovaniu. Z techník som najviac pozornosti venoval inšpekcii kódu, najmä napísanému v OO jazyku, keďže sa v ňom v súčasnosti tvorí veľká časť programov. Okrem inšpekcie kódu sú v práci predstavené techniky založené na intuícií i na presných postupoch a protokoloch. Hoci je testovanie nevyhnutnosťou pre zabezpečenie kvality softvéru, prináša so sebou aj určité náklady. Snahou tejto práce je pomôcť vývojárom a testerom čo najefektívnejšie testovať.

Samotné testovanie neobľubuje väčšina vývojárov, mňa nevynímajúc. Ak je to projekt do školy, kde je cieľom preukázať iba nejakú funkcionalitu, testujem iba tú, aj to prevažne iba ad hoc metódou. Pri práci vo firme sa ale musím riadiť iným prístupom k testovaniu, a teda i samotnej kvalite kódu. Kód, ktorý vytvorím, priebežne otestujem len so základnou množinou testovacích údajov, t.j. vyskúšam či daná časť softvéru vykonáva to čo má, poprípade to vyskúšam v niektorých krajných situáciách. Ak sa

správa požadovane, idem vo vývoji aplikácie ďalej. Ako sa postupne dopĺňa funkcionalita vytváranej aplikácie, používa sa viac a viac funkcií a so zložitejšou množinou dát a v zložitejších situáciách, občas vyskočí niektorá chyba v projekte. Zväčša sa dodržiava zásada, že každý programátor je zodpovedný za kvalitu kódu a funkcionalitu tej časti programu, ktorú vytvoril. Ak sa k danej chybe nik nehlási, nástroje pre logovanie a/alebo verzionovanie pomôžu vypátrať „vinníka“. Takéto priebežné testovanie počas vyvíjania produktu priveľmi nenudí a neberie programátorovi chuť a radosť z programovania. Zároveň poskytuje pomerne funkčný a otestovaný produkt.

Použitá literatúra

1. Berner, S., Weber, R., Keller, R.K.: Enhancing Software Testing by Judicious Use of Code Coverage Information. In: *Proceedings of the 29th International Conference on Software Engineering*, IEEE Computer Society, Washington, DC (2007), 612-620.
2. Bertolino, A.: Software Testing Research: Achievements, Challenges, Dreams. In: *International Conference on Software Engineering, 2007 Future of Software Engineering*, IEEE Computer Society, Washington, DC (2007), 85-103.
3. Bourque, P., Dupuis R., eds, IEEE Computer Society: Guide to the Software Engineering Body of Knowledge: 2004 Version, www.swebok.org, October 2007.
4. Dijkstra, E.: Notes on structured programming. Technological Univ. Eindhoven, Einhoved (1970).
5. Dunsmore, A., Roper, M., Wood, M.: Further investigations into the development and evaluation of reading techniques for object-oriented code inspection. In: *Proceedings of the 24th International Conference on Software Engineering* , ACM Press, New York (2002), 47-57.
6. Gamma, E., et al.: Design Patterns, Addison-Wesley, London (2004).

Annotation

Testing and code inspection or how testing software effectively

Large influence on quality of software has code testing. In this document are presented some characteristics of code testing. Testing is cost activity, so everybody wants to make testing effectively as is possible. Very effective technique is code inspection. However, testing of object-oriented software has its own special features, that testing of other software has not. Essay is concerned on some techniques to inspection object-oriented code. In next part of essay are presented different testing techniques, their advantages and disadvantages. Aim of essay is show some testing techniques and helps readers to make testing more effectively.