Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Študijný program: Informatika

Peter Beňuš

Automatické testovanie softvéru

Bakalársky projekt 1

Vedúci bakalárskeho projektu: Ing. Karol Rástočný
December 2015

Obsah

1 Testovanie softvéru **2**
1.1 Podľa postupu testovania 2
1.2 Podľa spôsobu testovania 3
1.3 Podľa úrovne testu 3

2 xUnit **4**
2.1 Nástroje patriace do skupiny xUnit pre .NET 5

Literatúra **6**

1 Testovanie softvéru

Testovanie softvéru je empirická činnosť, ktorá skúma kvalitu testovaného produktu alebo služby vykonávaná na podanie informácií o kvalite všetkým zainteresovaným osobám [4]. V súčastnosti existuje veľa spôsobov testovania a veľa častí životného cyklu softvéru v ktorých sa aplikujú iné typy testov. Testovanie softvéru môžem rozdeliť na kategórie podľa postupu, ktorý sa používa pri testovaní, podľa spôsobu testovanie a podľa úrovne testu.

1.1 Podľa postupu testovania

- **Testovanie formou čiernej skrinky**
  Testuje funkcionalitu bez informácií o tom ako je softvér implementovaný. Tester dostane iba informácie o tom, aký by mal byť výsledok testu po zadaní vstupných dát a kontroluje výstup softvéru či sú výstupné dáta totožné s očakávanými [3]. Test je konštruovaný z funkcionálnych vlastností, ktoré sú špecifikované požiadavkách na program [5]. Výhodou tohto typu testovania je, že tester nie je ovplyvnený štruktúrou zdrojového kódu a tým môže odhaliť chyby aj tam, kde to programátor nehľadal lebo to považoval za správne pri pohľade na zdrojový kód, ale bola tam chyba, ktorá zo zdrojového kódu nemusí byť viditeľná (napríklad nesprávne, resp. nedostatočné ošetrenie nekorektných vstupov). Hlavnou výhodou je, že tester nemusí poznať zdrojový kód a preto môže oveľa rýchlejšie vytvoriť testy. Nevýhodou je úroveň otestovania systému, pretože tvorca testov nevie ako program funguje, a preto veľmi pravdepodobne nebude schopný vytvoriť test, ktorý by testoval všetky vetvy programu. Používa sa pri jednotkovom, integračnom, systémovom a akceptačnom testovaní. Okrem použitia pri rôznych úrovní testov sa využíva aj na validáciu softvéru [3].

- **Testovanie formou bielej skrinky**
  Pri tomto spôsobe testovania je testerovi známa vnútorná štruktúra softvéru a aj konkrétna implementácia. Test sa tvorí tak, aby bola otestovaná každá vetva zdrojového kódu [3]. Používa sa pri jednotkovom testovaní na skoré odhalenie všetkých chýb, pri integračnom testovaní na testovanie správnej spolupráce rôznych jednotiek programu a aj pri regresnom testovaní, kde sa používajú recyklované testovacie prípady z integračného a jednotkového testovania a taktiež slúži na verifikáciu. Výhodou je

schopnosť otestovať komplexne všetky vetvy, ktoré program na danej úrovni testu vykonáva, ale nevýhodou je, že tester musí mať dobré vedomosti o zdrojovom kóde a v niektorých prípadoch tvorenia testov môže byť znalosť zdrojového kódu nevýhoda.

- **Testovanie formou sivej skrinky**
  Spôsob testovania, pri ktorom je známy zdrojový kód (nemusí byť sprístupnený úplne celý), ale testy sa vykonávajú rovnako ako pri testovaní formou čiernej skrinky. Používa sa napríklad pri integračnom testovaní ak máme dva moduly od rôznych vývojárov a odkryté sú len rozhrania [3]. Poskytuje výhody obidvoch predchádzajúcich prístupov, ale má oproti nim aj nejaké nevýhody. Oproti testovaniu čiernou skrinkou má výhodu v lepšom pokrytí rôznych vetiev zdrojového kódu, ale je časovo náročnejšie na tvorbu testov. Oproti testovaniu bielou skrinkou je menej náročné na znalosť zdrojového kódu, pretože ho nemá prístupný celý, ale nepokrýva všetky vetvy programu a preto je menej komplexné.

1.2 Podľa spôsobu testovania

- **Statické testovanie**
  Statické testovanie je často implicitné. Zahŕňa napríklad kontrolu zdrojového kódu programátorom jeho čítaním hneď po napísaní, kontrolu štruktúry a syntaxe kódu nástrojom alebo editorom, v ktorom sa zdrojový kód píše. Program nie je potrebné spúšťať, ale analýza zdrojového kódu založená na upravovacích pravidlách zistí v zdrojovom kóde rôzne možné chyby, ktoré sa zvyčajne objavujú v spravovaní pamäte, neinicializovaných premenných, výnimke nulového smerníku, porušení prístupu k poľu a taktiež pretečení vyrovnávacej pamäti [7].

- **Dynamické testovanie**
  Dynamické testovanie prebieha už na spustenom softvéri. Začína skôr ako je úplne dokončený softvér, pretože sa počas vývoja testujú aj menšie spustiteľné časti. K dynamickému testovaniu sa viaže validácia.

1.3 Podľa úrovne testu

- **Jednotkové testovanie**
  Jednotkové testovanie je metóda testovania softvéru, pri ktorej sa testujú individuálne komponenty (jednotky) zdrojového kódu. Zvyčajne nie je testovacou fázou v zmysle nejakého obdobia na tvorbe projektu, ale skôr je to posledný krok písania časti zdrojového kódu [1]. Programátori takmer vykonávajú jednotkové testovanie takmer stále, či už pri testovaní vlastného zdrojového kódu alebo kódu iného programátora [1]. Kvalitné testovanie na tejto úrovni môže výrazne znížiť cenu a čas potrebný na vývoj celého softvéru [3].

- **Testovanie komponentov**
  Počas testovania komponentov sa testeri zameriavajú na chyby v ucelených častiach systému. Vykonávanie testu zvyčajne začína, keď je už prvý komponent funkčný

spolu so všetkým potrebným (napr. ovládače) na fungovanie tohto komponentu bez zbytku systému [1].
Testovanie komponentov má sklon viezť k štrukturálnemu testovaniu alebo testovaniu formou bielej skrinky. Ak je komponent nezávislý môže sa použiť aj testovanie formou čiernej skrinky [1].

- **Integračné testovanie**
  V integračnom testovaní sa testeri zameriavajú na hľadanie chýb vo vzťahoch a rozhraniach medzi pármi a skupinami komponentov. Integračné testovanie musí byť koordinované, aby sa správna množina komponentov spojila správnym spôsobom a v správnom čase pre najskoršie možné odhalenie integračných chýb [1]. Niektoré projekty nepotrebujú formálnu fázu integračného testovania. Ak je projekt množinou nezávislých aplikácií, ktoré nezdieľajú dáta alebo sa nespúšťajú navzájom, môže byť táto fáza preskočená [1].

- **Systémové testovanie**
  Systémove testovanie je vykonávané na úplnom a integrovanom systéme za účelom vyhodnotenia súladu systému z jeho špecifikovanými požiadavkami [2]. Niekedy, napríklad pri testovaní inštalácie a použiteľnosti, sa tieto testy pozerajú na systém z pohľadu zákazníka alebo koncového používateľa. Inokedy sú testy zdôrazňujú konkrétne aspekty, ktoré môžu byť nepovšimnuté používateľom, ale kritické pre správne fungovanie systému [1].

- **Akceptačné testovanie**
  Akceptačné testovanie je formálne testovanie zamerané na potreby používateľa, požiadavky a biznis procesy vedúce k rozhodnutiu či systému vyhovuje alebo nevyhovuje akceptačným kritériám a umožniť používateľovi, zákazníkovi alebo inému splnomocnenému subjektu či má alebo nemá byť systém akceptovaný [6].
  Narozdiel od predchádzajúcich foriem testovania, akceptačné testovanie demonštruje, že systém spĺňa požiadavky [1].
  V komerčnej sfére sú niekedy tieto testy nazývané aj podľa toho kým sú vykonávané älfa testy"(používateľmi vo firme) alebo "beta testy"(súčasnými alebo potenciálnymi zákazníkmi) [1].

2 xUnit

xUnit je označenie pre skupinu frameworkov, ktoré slúžia na jednotkové testovanie. Vznikol pôvodne pre programovací jazyk Smalltalk a veľmi rýchlo sa stal známym a úspešným. Dnes už majú všetky bežne používané programovacie jazyky minimálne jeden vlastný framework na jednotkové testovanie a mnoho z nich je odvodených práve od xUnit. http://www.martinfowler.com/blik

Spoločné znaky frameworkov patriacich do skupinu xUnit:

- **Test runner** Je to spustitešný program, ktorý vykoná test a zároveň vytvorí správu o výsledku testu.

- **Test case** Je to základná trieda, od ktorej sú odvodené všetky testy.
- **Test fixtures** Množina podmienok definovaných programátorom, ktoré musia byť splnené pred vykonaním testu. Po teste by mali byť vrátené do pôvodného stavu.
- **Test suites** Množina testov, ktoré zdieľajú podmienky potrebné pre spustenie testu.
- **Vykonanie testu** Vykonanie individuálneho jednotkového testu.
- **Test result formatter** Výsledok testu môže byť v jednom alebo viacerých formátoch. Okrem textu čitateľného pre človeka sa často používa aj výstup vo formáte XML.
- **Assertion** Je to funkcia alebo makro, ktorá definuje stav testovanej jednotky. Zvyčajne je to logická podmienka, ktorá pravdivá ak je výsledok testu správny. Zlyhanie väčšinou končí volaním výnimky, ktorá ukončí vykonávanie testu.

2.1 Nástroje patriace do skupiny xUnit pre .NET

Fixie Patrí k novším frameworkom a umožňujem programátorovi vytvárať a vykonávať jednotkové testovanie. Výhoda, ktorú Fixie prináša, že je založený na konvencií. Preto programátor pri písaní testu nemusí používať atribúty na označovanie tried a metód. Keď je dodržaná konvencia tak Fixie vie podľa názvu zistiť či ide o metódu alebo triedu. Ak by predvolená konvencia nebola vyhovujúca, je možné vytvoriť si vlastnú a následne sa riadiť ňou. Na internete je možné aj nájsť plugin pre ReSharper, avšak zatiaľ len beta verziu. https://visualstudiomagazine.com/articles/2015/04/22/fixie-c-sharp-testing.aspx

MbUnit Je to rozšíriteľný framework, ktorý okrem toho, že prijíma vzory xUnit ide ešte ďalej a poskytuje programátorovi viac, ako napríklad:

- **Porovnávanie XML (XML assertions)** MbUnit obsahuje metódy pomocou, ktorých sa dajú porovnávať aj hodnoty v XML súboroch. https://vkreynin.wordpress.com/2010/07/18/test/
- **Paralelizovateľné testy** Každý test, ktorý je označený ako paralelný bude pri vykonávaní spustený spolu z ostatnými paralelizovateľnými testami. http://blog.bits-in-motion.com/2009/03/announcing-gallio-and-mbunit-v306.html
- **Externé zdroje dát** Dáta používaného v testoch môžu byť uložené v rôznych typoch súborov (XML, CSV, a pod.) a počas testu používané priamo z nich.

Okrem tohto je MbUnit aj generatívny framework. čo znamená že dokáže z jednoduchého jednotkového testu urobiť niekoľko ďalších. Od roku 2013 uz ale nie sú žiadne commity na GitHube a preto ho môžeme považovať za už nevyvíjaný softvér. http://stackoverflow.com/questions/3678783/mbu vs-nunit

Mock Je to simulovaný objekt, ktorý imituje správanie reálneho objektu. Zvyčajne sa používajú pri jednotkovom testovaní. Mockovanie je celé o imitovaní (faking) reálnych objektov a robení operácií kontrolovaným spôsobom.http://www.agile-code.com/blog/mocking-with-moq/

Moq Je podľa tvorcov jediná mockovacia knižnica, ktorá je vytváraná od začiatku, tak aby využila naplno výhody .NET Ling (Language-Integrated Query) strom výrazov a lambda výrazy.

- Strong-typed: no strings for expectations, no object-typed return values or constraints
- Neprekonaná integrácia s intellisense vo Visual Studio: všetko je plne podporované intellisense vo Visual Studio
- Žiadne nahraj/prehraj idiómy(jazyk) (No Record/Replay idioms to learn.) na učenie. Stačí vytvoriť mock, nastaviť ho, použíť ho a voliteľne potvrdiť ich volania (netreba potvrdzovať mocky, keď vystupujú len ako stuby (stubs) alebo keď sa robí klasickejšie stavovo-založené (state-based) testovnie kontrolovaním navratových hodnôt testovaného objektu).
- Veľmi nízka učiaca krivka (learning curve). Pre väčšinu častí ani netreba čítať dokumentáciu.
- Granulovaná kontrola (granular control) nad správaním mocku s jednoduchou MockBehavior enumeráciou (netreba vedieť teoretické rozdiely medzi mockom, stubom, imitáciou (fake), dynamickým mockom a pod.)
- Je možne mockovať rozhrania aj triedy
- Override expectations: can set default expectations in a fixture setup, and override as needed on tests
- Posielať argumenty pre mockované triedy
- Ohraničiť (Intercept) a vyvolať (raise) akcie (events) na mockoch
- Intuitívna podpora pre out/ref argumenty

https://github.com/Moq/moq4

Literatúra

[1] Enrique Alba and Francisco Chicano. Observations in using parallel and sequential evolutionary algorithms for automatic software testing. *Computers & Operations Research*, 35(10):3161–3183, 2008.

[2] Standard Computer Dictionary. *Standard Computer Dictionary*.

[3] Elfriede Dustin. *Effective Software Testing: 50 specific ways to improve your testing*. 2002.

[4] Cem Kaner. Exploratory Testing. *Quality Assurance International*, (c), 2006.

[5] A.A. Omar;F.A. Moha. A survey of software functional testing methods. *Software Engineering Notes*, page 75, 1991.

[6] E. V Veenendaal. Standard glossary of terms used in Software Testing, Version 1.2. *International Software Testing Qualification Board*, 1:1–51, 2010.

[7] Wang Wei. From source code analysis to static software testing. *2014 IEEE Workshop on Advanced Research and Technology in Industry Applications (WARTIA)*, pages 1280–1283, 2014.