Slovenská technická univerzita v Bratislave
Fakulta informatiky a informačných technológií
Ilkovičova 2, 842 16 Bratislava 4

Metodika testovania

Akademický rok: 2016/2017
Vedúci práce: Ing. Karol Rástočný, Phd.
Členovia tímu: bc. Ondrej Čičkán, bc. Šimon Dekrét, bc. Dušan Javorník
bc. Dušan Jom, bc. Miroslav Laco, bc. Anton Ján Vrban
Dátum poslednej zmeny: 14.12.2016
Vypracoval: Miroslav Laco

Účel metodiky

V rámci predmetu Tímový projekt implementujeme veľké množstvo funkcionality, ktorá je navzájom prepojená. Túto funkcionalitu implementujeme v tíme, pričom na implementácii jednej funkcionality môžu pracovať viacerí členovia tímu. Každý člen tímu potrebuje časť funkcionality vyvíjať nezávisle od ostatných, aby si mohol korektnosť svojej implementácie overiť. Zároveň potrebuje každý člen tímu zabezpečiť, aby svojou implementáciou neporušil korektnú funkcionalitu, ktorá bola už v minulosti implementovaná, a to s ohľadom nato, že člen tímu nemusí poznať pozadie predošlej implementácie (postačujú jej princípy). Preto sme zaviedli v tíme povinnosť unit testingu. **Unit testing** je spôsob testovania korektnosti implementácie určitej atomickej časti funkcionality. V tejto metodike je opísaná konvencia, postup a princípy unit testingu pri vyvíjaní tímového projektu tímom RavensTeam.

Roly

Programátor

Osoba zodpovedná za implementáciu a vykonávanie testov pre vytvorený kód.

Zodpovedná osoba za user story

Osoba zodpovedná za pokrytie kódu, ktorý vznikol v rámci user story, testami. V prípade zistených možností vylepšenia metodiky testovania kontaktuje s návrhmi manažéra pre testovanie. Rovnako sa naňho obracia ak chce udeliť výnimku v oblasti testovania pre user story.

Code reviewer

Osoba zodpovedná za kontrolu pokrytia kódu, ktorý prehliada, testami. V prípade problémov kontaktuje manažéra pre testovanie.

Manažér testovania

Osoba zodpovedná za definovanie štandardov pre testovanie. Rieši problémy identifikované pri testovaní. Je zodpovedný za continuous improvement v oblasti testovania softvéru. Kontroluje priebeh procesov súvisiacich s testovaním.

Unit testing

Doposiaľ používaným IDE pre vývoj tímového projektu tímom RavensTeam je IDE Visual Studio s napojením na Team Foundation Server. Preto sme zvolili framework pre testovanie, ktorý je priamo integrovaný do IDE Visual Studio- **Microsoft unit test framework for managed code**[1]. Pre izoláciu v minulosti implementovaných funkcionalít v kóde sme zvolili pôvodne framework priamo integrovaný do IDE Visual Studio- Microsoft Fakes[2]. Pre používanie Microsoft Fakes bolo potrebné mať nainštalované IDE Visual Studio 2015 v konfigurácii Enterprise s nainštalovaným Update 3. Kvôli komplikovanosti spomínaného framework sme tento po 4. šprinte prehodnotili a zvolili pre ďalšie šprinty framework **Moq**[3].

---

[1] základný prehľad - https://msdn.microsoft.com/en-us/library/ms182532.aspx
[2] https://msdn.microsoft.com/en-us/library/hh549175.aspx
[3] https://github.com/Moq/moq4

Po zmene nie je potrebné meniť predošlé unit testy. Nie je taktiež potrebné mať konkrétnu konfiguráciu IDE Visual Studio. Je potrebné mať nainštalovaný NuGet pre Moq. Pre prístup k panelu testov vo Visual Studio klikneme v hornom menu IDE na položku Test > Windows > Test Explorer.

Základné princípy

Atomické časti funkcionality implementujeme do samostatných metód. Pokiaľ je to možné a ako z doposiaľ riešených úloh (taskov) z Team Foundation Server vyplýva, tak jeden task zodpovedá jednej alebo viacerým atomickým častiam funkcionality. Každý task má implementovať minimálne 1 samostatnú metódu v ktorej je implementovaný (prípadne združený súbor metód ak sa jedná o implementáciu Controllera, Service, Modelu..). Každá z takto implementovaných metód vrátane konštruktora triedy a komplexných get/set metód (zložitejších ako jednoduchý prístup k premennej triedy) musí mať napísaný zodpovedajúci unit test. Pre splnenie task-u je potrebné, aby napísané testy pre task a aj všetky doposiaľ napísané testy boli úspešne ukončené.

Každá ucelená funkcionalita vystavuje rozhranie (Interface) a sprehľadňuje tak nielen kód samotný, ale aj zjednodušuje unit testing.

V implementácii riešenia je vytvorený práve 1 unit test projekt. Jeho štruktúra zodpovedá štruktúram ostatných projektov riešenia vynímajúc netestovateľné časti implementácie. Medzi časti implementácie, ktoré sa z testovania vynímajú patria:

- Views
- User interface
- Konfiguračné súbory
- Projektové súbory
- Závislosti

Platí, že okrem výnimiek spísaných vyššie prislúcha každej metóde implementujúcej atomickú časť funkcionality unit testovacia metóda. Pokiaľ je potrebné atomickú časť funkcionality izolovať, používa sa nato mock. Mock je spôsob, ktorým sa pri preklade kódu, alebo za behu program nahrádza časť kódu tak, aby mala falošná metóda mock-u presne definovaný výstup.

Platí, že všetky závislosti testovanej metódy musia byť mockované, čiže nahradené falošnými údajmi generovanými pomocou Moq framework-u. Nie je dovolené pri testovaní metódy volať inú nemockovanú metódu pre zabezpečenie základného princípu unit testingu.

Nie je povolené poslať kód na posudzovanie pokiaľ sme implementáciu nepokryli testami. Výnimkou sú závažné bug fixy s vysokou prioritou, ďalej implementácie, ktorých testovanie vyžaduje rozsiahlejšiu konzultáciu, ktorá by brzdila ďalší vývoj a implementácie, ktoré musia byť súrne zapracované do hlavnej vývojovej branche. Z unit testingu sa vynímajú tiež prípady, ktoré dostali výnimku povinnosti posudzovania kódu. Platí však, že pre všetky vyňaté prípady spomínané vyššie je potrebné dopracovať unit testy čo najskôr, ako to bude dodatočne možné.

Za pokrytie kódu user story unit testami zodpovedá osoba zodpovedná za user story. Za kontrolu pokrytia kódu user story testami zodpovedá osoba, ktorá pre user story vykonáva code review. Pokiaľ táto osoba zistí nedostatočné, alebo chýbajúce unit testy, musí pri code review napísať upozorňujúci komentár a vyčkať na opravu.

Postup pri testovaní

1. Implementujeme metódu zodpovedajúcu atomickej časti funkcionality.
2. Vytvoríme testovaciu metódu pre implementovanú metódu. Pokiaľ je metóda v triede, ktorá v testovacom projekte neexistuje, vytvoríme najprv túto triedu.
3. Skontrolovanie závislostí testovanej metódy. Pre závislosti vytvoríme Mock pomocou Moq framework-u.
4. Zvolíme si vhodné vstupy metódy, pričom pokrývame tieto prípady:
   - Bežná očakávaná hodnota na vstupe
   - Hraničné hodnoty
   - Hodnoty mimo rozsahu
5. Determinujeme výstupy testovanej metódy na základe zvolených vstupov.
6. Pre každú sadu vstupov voláme testovanú metódu a používame asertovacie porovnanie[4] očakávaného výsledku metódy so skutočným výsledkom. Poznámka- každá sada vstupov musí byť otestovaná asertívnym porovnaním v dosiahnuteľnom kóde.
7. Spustíme všetky unit testy v Test Explorer paneli.
8. Pokiaľ všetky testy úspešne prebehli, testovanie končí. Pokiaľ aspoň jeden z testov neprebehol správne, implementáciou odstránime chybu v kóde a pokračujeme bodom 7.

Odporúčania k testovaniu

Odporúča sa navrhovať implementáciu funkcionalít systému tak, aby mohla jedna ucelená funkcionalita vystaviť rozhranie pre systém. Potom všetky ostatné interné a externé sučasti systému využívajú miesto konkrétnych tried rozhranie pre prístup k vybraným metódam. Toto pravidlo by malo ešte silnejšie platiť pri komunikácii medzi projektami vrámci riešenia. Zabezpečí sa tak možnosť jednoducho odizolovať funkcionality v systéme.

Odporúča sa vyvarovať duplicitným výnimkám (Exceptions) vrámci 1 metódy implementujúcej atomickú časť funkcionality. Duplicitné výnimky nie je možné bez doplňujúcej úpravy korektne otestovať a takýto test je považovaný za nedostatočný. Pokiaľ nastane situácia, kedy je nutné použiť rovnaké výnimky vrámci 1 metódy, tak tieto výnimky rozšírime pri implementácii dodatočným opisom druhu výnimky vo forme string. Dodatočný opis si vieme pri implementácii unit test metódy po zachytení výnimky asertovacím porovnaním pre string porovnať so želaným druhom duplicitnej výnimky.

[4] https://msdn.microsoft.com/en-us/library/ms182530.aspx