Semestrální práce ke kurzu 4IT421 Zlepšování procesů budování IS	
Semestr	ZS 2017/2018
Autoři	Daniel Sládek | xslad23 Renat Kulalov | xkulr05 Zbyšek Vacek | xvacz00
Téma	Are Unit Tests Part of Your Team's Performance Reviews?
Datum odevzdání	17.12.2017

Abstrakt

Práce se zabývá tématem jednotkových testů a jejich rolí při hodnocení výkonnosti zaměstnance.

Klíčová slova

Unit tests, pefrormance review, jednotkové testy, výkonnost, hodnocení zaměstnance

Obsah

Úvod

Mluvíme-li o hodnocení vývojářů, mohou být jednotkové testy občas trochu opomíjenou doménou. Zeptáme-li se některých vedoucích pracovníku na to, co si představí pod pojmem „dobrý programátor“, obvykle si na oblast jednotkových testů ani nevzpomenou. Vzhledem k tomu, jaký je přínos unit testů na celkovou kvalitu kódu, potažmo projektu, je vhodné se na toto téma podívat z blízka a ujasnit si, jak smysluplně psát jednotkové testy, jestli by se měly promítnout do hodnocení zaměstnance a případně jakým způsobem.

Cílem práce je tedy proto ukázat výhody unit testů při práci na softwarovém projektu a zejména představit doporučení a zásady, jak jednotkové testy psát.

Dále je v práci naznačeno místo a roli jednotkových testů při hodnocení zaměstnanců. Jaké výhody přináší hodnocení zaměstnancem napsaných testů a naopak, jaké problémy či nevýhody z něj při špatném přístupu mohou plynout. K tomuto tématu dodáme i naše osobní zkušenosti a názory.

Přínos unitových testů pro softwarový projekt

Jednoduše unitový test se dá definovat jako automatizovaný test, jejímž cílem je ověřování správné funkčnosti jednotek zdrojového kódu. Pod jednotkou se může rozumět funkce nebo procedura. V objektově orientovaném programování jednotkou se často rozumí třída nebo metoda.

Přidáním unitových testů do procesu budovaní softwaru, nebo jako část CI procesu, přinášíme do projektu možnost odhalovat chyby brzy ve vývojářském cyklu. Dřív, než aplikace dostane k testerům nebo ke klientům, což znamená že oprava dané chyby bude stát méně, než kdyby byla nalezena při testováni nebo dokonce ostrém provozu.

Mezi dalšími benefity použití unitových testů patří:

- Zvyšují kvalitu kódu
- Zjednodušují integraci
- Prokazují konkrétní progress

V dnešní době psaní unitových testů není zcela běžné, nicméně benefity, které jednotkové testy přinášejí, jsou velmi významné a přínosné, aby psaní unitových testů bylo úplně přehlíženo nebo ignorováno.

Zásady a způsoby psaní unit testů

Pro psaní dobrých unit testů, by se mělo dodržovat několik hlavních zásad. Jeremy Miller (Miller, 2005) definoval hlavní zásady jako:

- Unitové testy by měly být atomické
- Nezávislé a izolované
- Měly by mít jasný záměr
- Jednoduše nastavitelné
- Rychlé

Unitové testy by měly být atomické

Atomický kód znamená, že kód by se při dokončení měl vrátit do původního stavu a neměl by mít vedlejší efekty. U unitových testů to prakticky znamená, že když spustíme jeden kód několikrát za sebou, vždy by měl vrátit ten stejný výsledek.

Docílit se toho dá, pokud se vyvarujeme používání globálního stavu kódu, jako jsou globální či statické proměnné, využívání externích dat (zapisování do souborů, dotazy do databáze) nebo přepisovat nějaké nastavení aplikace. Jestliže, něco z toho použijeme, je to vždy potřeba uvést do původního stavu a vyvarovat se toho, že by to mělo mít efekt na jiné testy. Nejlepší, ale je vyvarovat se těchto věcí úplně.

Nezávislé a izolované testy

Jak udělat to, aby testy nebyly závislé bylo v zmíněno v předešlém způsobu, zde jde ale především o to, že by testy neměly být závislé i navzájem na sobě. Tím pádem by mělo být jedno, v jaké pořadí se testy spouští.

Izolované testy jsou takové, které jsou napsány bez potřeby volat cizí objekty, které nepřímo souvisejí s testy. U těchto testů by měly jít data “zfalšovat”, tomuto způsobu izolace testu se říká mocking. Ten řeší to, že část aplikace, která potřebuje ke své funkčnosti vytvořit jiné objekty, tak změna objektu nemá vliv na test. Mock objekty jsou náročné na tvorbu a funguje

proto několik služeb, které poskytují mock data nebo izolaci testů, jedna z těchto služeb pro .NET, C & C++ je např. TypeMock (www.typemock.com).

Jasný záměr unit testu

Při vytváření testovací metody by měl umět vývojář nazvat metodu, podle toho co by měla testovat. Často se totiž stává, že název metody je nejasný a z toho vyplývá, že ani samotný vývojář tedy nemůže vědět co přesně chce testovat.

Jestliže se toto pravidlo nedodržuje, tak jsou testy náchylné na to, stát se velmi komplexní a rozsáhlé a např. i jedna metoda toho testuje mnohem víc než pouze jednu jednotku. Tyto testy jsou náchylné na změny v kódu, a i menší část má velký vliv na výsledek celého testu.

Dle Eli Lopiana (Lopian, 2017) je mnohem lepší psát menší části kódu, které testují jen malou část aplikace, protože je mnohem snazší při selhávajícím testu najít, kde je chyba. Dále má výhody v tom, že kód je čitelnější a mnohem snazší test upravit a udržovat.

Tohle je také jedna z věcí, která by se měla při hodnocení zaměstnance zohledňovat. Je lepší sledovat jak je test udržitelný a snadno měnitelný, než jak dlouho jej zaměstnanec psal.

Jednoduše nastavitelné

Když nový zaměstnanec nastoupí do firmy znamená to většinou pár dnů v horším případě pár týdnu na zprovoznění projektu. Tento způsob vytvoření projektu je samozřejmě neefektivní. Databáze a ostatní zdrojové soubory by měly být nahrazeny mock zdroji a ty by měly být vytvořeny automatický buildem aplikace.

Rychlé

Podle všeho asi nejméně důležitý aspekt unit testu. Pokud ale vyhodnocení a spuštění všech unit testů trvá půl hodiny nebo déle, je jasné, že zaměstnanec toho moc naprogramovat nestihne. Je proto lepší používat místo databáze a složitého získávání dat mock zdroje a tím zrychlit běh testů.

Existuje mnoho způsobů a best-practise psaní unit testů, ale většina je odvozena od těchto 5 základních pravidel. Pro kompletnost zde zmíníme ještě další, které sepsal vývojář a konzultant Dror Helper a to:

- Jmenné konvekce

- Opakuj sám sebe
- Testování výsledků, ne implementace

Jmenné konvekce

Jedna z důležitých částí testů je také jeho čitelnost. Jako první je při spuštění testů vždy vidět název metody. Je proto velmi důležité, aby při případném selhání název jasně vysvětloval, co přesně se testuje. Další věc je mít také nastavené jmenné konvekce kódu, ale to už spíše patří obecně do programování.

Opakuj sám sebe

V programování je široce známý pojem DRY (Don't repeat yourself), neboli neopakuj sám sebe. V programování je velmi důležitá jednoduchost kódu a udržovatelnost. Proto je vhodné vyhýbat se duplicitám. V unit testech, je ale nejdůležitější čitelnost, a proto je povolený mít duplicitní kód. Tento způsob je zdůvodněný tím, že je lepší mít několik podobných testů než jeden neduplikovaný test, který selhává.

Testování výsledků, ne implementace

Většina vývojářů, kteří znají kód, pro který píší test se dopouštějí časté chyby a to, že testují vnitřní funkcionalitu kódu, místo toho, aby testovaly přímo výsledek. Běžně se tento problém pozná tím, že se změnil požadavek na implementaci a poté test napsaný na tento kód už neprochází.

Tyto zmíněné zásady unitového testování by měl každý vývojář dodržovat a zároveň by mohly být zahrnuty do celkového hodnocení kódu, který odevzdává.

Ideální role jednotkových testů v rámci hodnocení zaměstnance

Pokud nebudeme měřit jednotkové testy, nebude se jejich kvalita zlepšovat. Mnoho týmů unit testy nebude psát, pokud podle nich nebudou hodnoceni a nebudou hrát roli při odměňování a kariérním růstu. Pokud hodnocení výkonnosti zaměstnance nezahrnuje i unit testy, zpravidla se zhoršuje kvalita a udržitelnost zdrojového kódu.

Správné perfromance review by tedy mělo zahrnovat hodnocení jednotkových testů a obsahovat metriky jak kvantitativní (počet testů, pokrytí testů), tak i kvalitativní (složitost testů, kvalita kódu). Pokud bychom využívali pouze kvantitativních metrik, mohli bychom narazit na problém kvality. Je velmi pravděpodobné, že v některých případech by si vývojáři usnadnili práci a napsali příliš zjednodušující, nesmyslný či vyloženě špatný test. Zavedením obou typů metrik v podobách přijatelných pro všechny členy týmu si zavedeme dobrou praktiku vývoje softwaru a zajistíme si rychlejší tvorbu kvalitního kódu.

Ukázkou některých kvantitativních metrik, které by bylo vhodné zavést do výkonnostního hodnocení může být:

- Počet nových / změněných testů
- Pokrytí kódu
- Jsou testy zaměřené na nový kód vs. legacy kód
- Počet bugů nahlášených od Quality assurance
- Stížnosti zákazníka
- Míra chyb

Naopak některými zajímavými kvalitativními metrikami může být:

- Otevřenost jedince k jednotkovým testům
- Komplexita kódu
- Vzdělávání se v oboru testování
- Kvalita kódu

Při zavádění těchto metrik do hodnocení je dobré hledět zejména na cíl, ne pouze na metriky. Perfromance review, které se soustředí pouze na metriky je bohužel běžnou praxí, ale nejedná se o příliš účinný nástroj. Příkladem neefektivnosti takového hodnocení může být testování irelevantního kódu. To nám sice zvýší celkové pokrytí testy, ale zároveň nám ve skutečnosti nepřidává žádnou reálnou hodnotu. Proto by se správné hodnocení mělo zaměřovat zejména na stanovené byznysové cíle a umožnit otevřenou diskuzi a komunikaci.

Z pohledu agilního vývoje může působit klasické výkonnostní hodnocení těžkopádně a spíše jako nástroj rigorózních metodik. Lze jej však upravit na míru agilních praktik. Místo nějakého dlouhého checklistu by mělo být hodnocení spíše pravidelná, volnou formou pojatá zpětná vazba. Je lepší přejít z hodnocení jednou ročně na vyšší frekvenci, ať už se jedná o čtvrtletí, měsíce či týdny. Častou zpětnou vazbu s mixem vhodných kvalitativních a kvantitativních cílů lze jistě označit za dobrou praktiku. I agilní manifest totiž preferuje

změnu před následováním plánu. Volná forma hodnocení nám zajišťuje zaměření se na byznysové cíle místo čistých metrik.

Takovýto přístup by se dal označit za ideální roli jednotkových testů v hodnocení zaměstnance. Nehledě na frekvenci a způsob hodnocení, unit testy by měly být vždy jeho součástí. Zařazením unit testů do performance review, jako jedné z klíčových metrik, vysíláme zaměstnancům vzkaz, že si vážíme kvalitního, udržitelného kódu a moderních programovacích technik. Výsledkem by měl být lepší kód s méně chybami a více motivovaní zaměstnanci.

Názor autorů práce na jednotkové testy a hodnocení zaměstnanců

V této části práce rozebíráme jednotkové testy a jejich hodnocení prakticky. Nejdříve popisujeme situaci jednotkových testů v konkrétní organizaci a následně vyjadřujeme vlastní názor na jednotkové testy a jejich roli při hodnocení zaměstnanců.

Popis situace v konkrétní organizaci

Budeme popisovat aktuální situaci ve firmě Bonami. Bonami je český startup se zaměřením na bytové doplňky, home decor a nábytek. Funguje na principu flash sale prodejce. Nabízí určité množství produktů po omezenou dobu. V případě Bonami se produkty prodávají ve více souběžně běžících kampaních, které trvají obvykle jeden týden. Mimoto některé oblíbené produkty, takzvané skladovky, jsou dostupné neustále, a lze je kupovat nehledě na kampaně. Aktuálně Bonami působí ve čtyřech zemích. Jednotlivé jazykové mutace se spouštěli v tomto pořadí – Česko, Polsko, Slovensko a Rumunsko. Bonami.cz bylo založeno v roce 2013. Stojí za ním Václav Štrupl a investiční skupina Miton.

Bonami hodně využívá takzvaný *emotional shopping*, česky někdy označovaný jako nakupování očima. V portfoliu firmy proto nalezneme krásné designové výrobky, s pěkně vyvedenými fotografiemi a vlastními texty. Na škodu produktu není ani vyšší cena, protože celkový dojem v nás vyvolává pozitivní emoce oproti „fádním“ výrobkům u konkurence. Mnohdy se jedná o impulzivní nakupování díky aktuálnímu nadšení z produktu.

Bonami je webová aplikace, využívající technologií jako PHP, JavaScript, NodeJs, Redis, Elasticsearch a mnoho dalších. Ve firmě aktuálně působí dvanáct vývojářů rozdělených na tři samostatné týmy:

- Frontendový tým
- Backendový tým
- Mobilní vývoj

Každý tým má svoji doménu působnosti, a i mírně odlišné zvyky a coding standards.
I když je název trochu zavádějící, frontend tým se nevěnuje pouze frontendu aplikace. Pracuje primárně na funkcích, které vidí koncový uživatel. Příkladem může být práce na samotném obsahu a vzhledu webu, objednávkový proces a podobně. Zasahuje tedy do frontendu i backendu aplikace. Srozumitelnější, i když pochopitelně zavádějící, termín pro název týmu by mohl být frontoffice vývoj. V tomto týmu se kromě integračních testů píši i testy jednotkové. U částech kódu v PHP se využívá známého frameworku PHPUnit od Sebastiana Bergmana. V jazyce JavaScript existuje mnoho frameworků na psaní jednotkových testů. Mezi nejznámější z nich patří Jasmine, Ava, Mocha nebo Jest. I přes dostatek možností, jak testy pro javascript vytvářet, se v tomto týmu žádné testy na javascriptový kód nepíšou.

Backendový tým se zaměřuje na funkcionality, které nejsou na první pohled pro zákazníka viditelné. Například skladový systém, propojení s finančními systémy atd.

V obou týmech panuje ohledně unit testů podobná situace. Jejich psaní je vítáno a podporováno. Zároveň však není až na výjimky vynucováno. Pokrytí codebase testy tedy není příliš velké. Píšou se zpravidla testy na důležité komponenty, jako jsou různé služby na rozhodování, komponenta na detekce dopravy zdarma či služba na rozřazování produktů do kategorií. Často se spoléhá, že případná nefunkčnost jednotlivých komponent se projeví při integračních, respektivě systémových testech. Několika scénáři se například testuje průchod celým objednávkovým procesem od přidání do košíku po dokončení objednávky. Tento přístup povětšinu času firmě stačí, protože pokryje funkčnost kritické části aplikace. Bohužel se neotestují zdaleka všechny scénáře, jako kdyby byly všude použity i jednotkové testy. To se občas projeví i na nalezených chybách v aplikaci.

Mobilní tým je aktuálně malý a relativně mladý. Testy se v něm až na výjimky nepíší.

Hodnocení zaměstnanců, respektive vývojářů probíhá zpravidla alespoň čtvrtletně. Někdy i v kratších intervalech. Probíhá volnou formou s několika předpřipravenými tématy k diskuzi. Je zaměřeno spíše obecně než přímo na jasně dané body. Se zaměstnancem se řeší, co se mu

dle managementu povedlo, co nepovedlo, jaký má přínos pro tým, co by měl zlepšit, v čem vyniká a podobně. Komunikace při této zpětné vazbě probíhá oběma směry. Kromě samotného hodnocení zaměstnance se naopak i zaměstnanec vyjadřuje k managementu, k situaci v jeho týmu, potažmo celé firmě.

Testování obecně nebývá při hodnocení programátora příliš velkým tématem. Do diskuze se dostane většinou v souvislosti s nějakým jiným tématem. Obvykle při rozebírání nějaké větší chyby či problému, který způsoboval kód nepokrytý testy. Rozhodně nejsou nastaveny žádné kvantitativní metriky. Dalo by se říci, že je nastavena kvalitativní metrika komplexity a metrika smysluplnosti jednotkových testů. Je zde však nesamostatně zahrnuta pouze obecně pod komplexitou a kvalitou kódu. Příliš komplexní testy jsou objeveny už při code review a vráceny na přepracování. Obdobně nesmyslné testy jsou také odhaleny při codereview a přepsány či odstraněny.

Svým způsobem tedy jsou jednotkové testy součástí hodnocení zaměstnance. Většinou v něm ale mají mizivé zastoupení.

Osobní názor autorů

Jednotkové testy jsou nepochybně důležitou a dobrou moderní programovací technikou. Jejich psaní by mělo být podporováno a pokud to dává smysl i vyžadováno. Podle našeho názoru by se jejich role při hodnocení měla odvíjet od samotného způsobu a formy performance review v dané firmě. Pokud probíhá formou vyplňování nějakého obřího formuláře, není problém přidat si pár políček s metrikami týkajících se unit testů. Pokud má volnou a nepříliš strukturovanou formu, je dobré si alespoň vzpomenout na množství a kvalitu testů daného zaměstnance.

Jsou ale jednoznačně důležitější témata, o kterých se bavit při hodnocení zaměstnance. To je vidět i v samotném článku, který sloužil jako výchozí bod této práce. Schizofrenně nejdříve vyjmenovává spousty možných metrik jen proto, aby je později tak trochu zahodil s tím, že je lepší se o testech bavit v rámci celkových byznysových cílů.
Hodnocení testů dle nás nemusí vždy přinést očekávané přínosy. Pokud budeme mít výborného zaměstnance, experta na nějakou doménu a bude-li ze své zásady odmítat psát testy, těžko s tím něco uděláme. Jen díky výsledkům svého hodnocení svůj přístup pravděpodobně nezmění.

Vezmeme-li pouze dříve uvedené metriky, přijdou nám z nich nejsmysluplnější tyto:

- Počet nových / změněných testů
- Komplexita kódu
- Kvalita kódu

Pokud bychom měli dodat nějakou vlastní metriku, byla by to **smysluplnost** testů. Ta nám přijde jako jednoznačně nejdůležitější metrika. Je lepší mít příliš komplexní test, který ale funkčně pokrývá důležitou logiku, než mít vysoce kvalitní a přiměřeně komplexní test na triviální kód typu změny barvy tlačítka.

V souhrnu si tedy myslíme, že je obecně dobré přihlížet při hodnocení zaměstnance k testům. Jakým způsobem, s jakou intenzitou a případně jestli vůbec však záleží na typu firmy a situace v ní. U malého startupu nemá cenu sledovat u každého zaměstnance velké množství metrik. V případě obří korporace to však může být v pořádku.

Závěr

V práci jsme probrali a ukázali výhody unit testů při práci na softwarovém projektu. Představili jsme některé obecné přístupy a zásady jejich psaní. Ukázali jsme si jednak ideální místo jednotkových testů při hodnocení zaměstnance a jednak realitu v jedné z firem, se kterou máme osobní zkušenost. V úvodu vytyčené cíle se nám podařilo splnit.

Použitá literatura

LOPIAN, Eli. Are Unit Tests Part of Your Team's Performance Reviews? [online]. 2017 [cit. 2017-12-5]. Dostupné z: https://www.infoq.com/articles/unit-tests-performance-review?utm_source=infoqWeeklyNewsletter&utm_medium=WeeklyNL_EditorialContent_culture-methods&utm_campaign=05302017news&utm_content=other

MILLER, Jeremy. Qualities of a Good Unit Test [online]. 2005 [cit. 2017-12-7]. Dostupné z: http://codebetter.com/jeremymiller/2005/07/20/qualities-of-a-good-unit-test/

BERTEIG, Mishkin. The qualities of an ideal test [online]. 2005 [cit. 2017-12-7]. Dostupné z: http://www.agileadvice.com/2005/05/17/agileengineering/the-qualities-of-an-ideal-test/

HELPER, Dror. 8 Principles of Better Unit Testing [online]. 2012 [cit. 2017-12-8]. Dostupné z: http://www.agileadvice.com/2005/05/17/agileengineering/the-qualities-of-an-ideal-test/