Rozbor práce

Behaviour Driven Develompent: Od teórie k praxi

> *„Test Driven Development (TDD) je jednou z praktik aplikovanych pri agilnom vyvoji softwaru. Vychadza z pravidla XP metodiky: testovať všetko, čo by sa mohlo mohlo pokaziť. Ak tuto myšlienku privedieme do extremu, dospejeme k zaveru, že pokaziť sa može všetko – zaklad TDD.“*

Základem Test Driven Development (TDD) není myšlenka testovat vše, co lze pokazit. Jak kolega uvedl správně, tato myšlenka vychází z Extreme Programming (XP) metodiky, ale základem TDD[1] je poněkud něco odlišného. Základní myšlenka této metodiky spočívá v tzv. test-first programming (coding), který právě také vychází z myšlenky XP metodiky, ale jeho záměr je poněkud odlišný. Ward Cunningham[2], známý tvůrce wiki a pionýr v oblasti návrhových vzorů a XP, o tom prohlásil: „Test-first coding is not a testing technique.“. Na první pohled poněkud zvláštní myšlenka, když se ale do důsledku zamyslíte, tak první co musíte udělat před samotným kódováním, je si ujasnit, co to vlastně budeme dělat. Myšlenka se tak přesune do oblasti analytické a návrhové, v zásadě si ujasnit co, proč a jak. Tahle pointa je velmi dobře rozvedena v článku *Aim, Fire*[3] od Kenta Becka[4] v časopise IEEE Computer Society.

> *„Kód vytvorený pomocou TDD má niekoľko výhod. Fakt, že všetky testy po najnovších úpravách kódu prebehli úspešne, nás môže ubezpečiť, že do systému nebola zavedená nová chyba. To urýchľuje integráciu nových zmien do systému, čo je nevyhnutné pri inkrementálnom spôsobe vývoja softwaru. Výsledky testovania je možné pomocou nástrojov continous integration sledovať v čase a robiť tak rôzne závery, predikcie a hodnotenia.“*

Základní idea testování pomocí TDD netkví v předpokladu, že pomocí testů zajistíme nezavlečení nějaké chyby do systému po upravě, ale v zajištění quality assurance (QA) a produktivity práce. Napadá mě několik situací, které jsem zažil, a v tomto bodě došlo k zásadnímu selhání testování. Tohle bychom měli raději přenechat formální verifikaci, která se hodí na tyhle účely :o). Pohled na QA a produktivitu z pohledu firem na TDD přináší v jedné kapitole zajímavá studie *Test driven development: empirical body of evidence*[5] nebo ke kvalitě všeobecně v TDD *Improving Application Quality Using Test-Driven Development (TDD)*[6]. Věta o continuous integration[7] je dle mého ne vhodně formulována. Je nutno si ujasnit, že jde o praktiku se svými specifikami a různými workflow,

1 http://en.wikipedia.org/wiki/Test-driven_development
2 http://en.wikipedia.org/wiki/Ward_Cunningham
3 http://athenea.ort.edu.uy/publicaciones/ingsoft/recursos/articulos/AimFire.pdf
4 http://en.wikipedia.org/wiki/Kent_Beck
5 http://www.agile-itea.org/public/deliverables/ITEA-AGILE-D2.7_v1.0.pdf
6 http://www.developerfusion.com/article/6875/improving-application-quality-using-testdriven-development-tdd/
7 http://en.wikipedia.org/wiki/Continuous_integration

a ne o využívání nástroje XYZ od firmy KLM. Ono ty nástroje fungují mnohdy různě a poskytují rozličnou funkcionalitu a záleží také i docela na verzovacím systému jaký používáte. Touhle praktikou se podrobněji zabývá článek Martin Fowlera *Continuous integration*[8].

> *„Navzdory výhodám, prax ukazuje, že TDD so sebou prináša aj nežiadúce vedľajšie efekty, na ktoré upozorňuje napr. Dave Astels v článku A New Look at Test-Driven Development (Astels, 2004). jako jednu z hlavných nedostatkov TDD vidí jazyk (v zmysle slovnej zásoby), ktorý používa. Ako základná jednotka je používaná tzv. unit. Jej rozsah nie je presne definovaný, ale často odráža štruktúru samotného kódu (triedy, metódy). Programátora to vedie k písaniu testu sústredného na určitú časť kódu (testovanie metódy na daných vstupoch...) vo veľmi špecifickom kontexte. Zároveň má tendenciu testovať čiste implementačné detaily, ako napr. určitý typ premennej a pod.“*

> *„Písanie testov ako záruka toho, že kód funguje správne (test = verifikácia), zdá sa, neprináša také úspechy, ako sa od neho očakáva. Jedným z dôvodov tohto problému môže byť fakt, že testovanie sa sústreďuje na to, čo daný objekt je, než na to, čo daný objekt robí (Chelimsky, 2010).“*

Problém TDD nespočívá v nějakých nedostatcích, ale spíše v nepochopení smyslu TDD, jak to ostatně naznačuje autor článku *A New Look at Test-Driven Development*[9] nebo Dan North[10] (otec BDD) ve svém článku *Introducing BDD*[11]. V TDD jde především o specifikování a ne o verifikaci, viz článek Davea Astela. Dave Astels nekritizuje jazyk TDD (ono je to docela nesmysl táto teze, protože si žádný jazyk nespecifikuje), ale fungování xUnit frameworků a jejich jazykové vyjadřovací prostředky, které jsou příliš test-centric, a proto se nehodí na specifikaci chování.

> *„Dave Astels vo svojom článku (Astels, 2004) naznačil možné riešenie nedostatkov, ktoré so sebou prináša TDD a opiera sa Sapir-Whorfovu hypotézu, že existuje prepojenie medzi jazykovými kategóriami jazyka, ktorým človek hovorí, a tým, ako daný človek vníma svet a správa sa v ňom. V našom prípade to teda znamená, že ak chceme zmeniť pohľad, ako sa pozeráme na objekty z pohľadu testovania, musíme najprv zmeniť jazyk, ktorým ich popisujeme.“*

Osobně bych první větu spíše formuloval ve smyslu, že se zabývá možnými řešeními, protože

8 http://www.martinfowler.com/articles/continuousIntegration.html

9 http://techblog.daveastels.com/files/BDD_Intro.pdf

10 http://dannorth.net/

11 http://dannorth.net/introducing-bdd

skutečným otcem BDD je Dan North. Tahkle mi to připadá, že s myšlenkou BDD přišel právě on. Oni hledali spíše takové řešení, které by lépe popsali specifikaci chování.

> *„Na rozdiel od xUnit-orientovaných frameworkov, BDD používa pri vyjadrovaní inú slovnú zásobu, ktorá sa snaží viesť k popisu toho, ako sa má daný objekt správať. Namiesto testovacieho prípadu (test case) používa ako základnú štruktúru kontext. Slovo test je nahradené slovom should. Namiesto overovania, že objekt sa správa podľa predpokladu (assertEquals(expected, actual)), píšeme špecifikáciu toho, ako sa má objekt správať (shouldBeEqual(actual, expected)). Cieľom týchto zmien v slovníku je teda zmeniť náš pohľad na to, akým spôsobom popisovať kód tak, aby bolo hlavné ťažisko na sústredené na špecifiká jeho správania a nie na jeho vnútornú štruktúru.“*

Jak už bylo výše naznačeno, problém tkvěl ve vyjadřovací prostředcích xUnit frameworků. To slovo kontext je jen jiné a jasnější pojmenování test case. Pro lepší vyjádření vznikly tvz. Behaviour Specification Frameworky, které smyslem jsou podobné xUnit frameworkům, ale používá jiný prostředek vyjádření. Co tu kolega popisuje je možnost udělat i udělat v xUnit-like frameworků, například v Shoulda[12], která zachová stejný prostředek vyjádření, ale rozšiřuje o výše uvedená slova. Trošku nevidím souvislost mezi xUnit frameworky a vnitřní struktury kódu (? kód implementace ?) nebo nerozumím, co tím chtěl autor říct. Jestli chce strukturu kódu popsat, tak na to jsou mock. Takhle to vnímám já.

> *„Cieľom BDD by mala byť skôr špecifikácia správania, než výsledná kontrola, že funguje. Tým sa testovanie lepšie sústreďuje na požiadavky, ktoré sú na neho kladené, s vynechaním implementačných, na výsledné fungovanie nepodstatných, detailov. Výsledkom tohto procesu by mala byť spustiteľná dokumentácia, teda textový dokument zrozumiteľný odborníkom doménovej oblasti, ktorý je zároveň možné spustiť pomocou vhodných nástrojov so všetkými výhodami, ktoré boli objavené už při používaní TDD. Je nutné podotknúť, že BDD a TDD sú skôr podobné, ako rozdielne, a veľa osvedčených techník zostáva zachovaných. Rozdiel je hlavne v slovníku a v uhle pohľadu, ktorým sa na problematiku pozeráme.“*

Některé věci jsem už výše poznamenal a často je znova poznamenává. Tady kolega míchá více věcí dohromady. Ty tzv. textové dokumenty ve formě spustitelné dokumentace jsou tzv. user story, kterými se zabývá až později, takže mi zde v tomhle kontextu nedávají smysl. Je si nutné také uvědomit existuje něco jako analyst level (user story – požadavky) a code level (unity). Doteď se zabýval totiž code level. Jak správně kolega poznamenává, rozdíl je ve slovníku, ale už nesouhlasím

12 http://github.com/thoughtbot/shoulda

s tvrzením, že jde o úhel pohledu. Jde o ty vyjadřovací prostředky, jak jsem se několikrát zmiňoval. Pokud se podíváte na ty příklady, tak je vám je jasné, že dělají to samé, ale liší se opravdu v těch prostředcích.

> *„Postupom času sa BDD prepracúva z určitej formy TDD k plnohodnotnej metodike vývoja softwaru svlastnou filozofiou, procesmi a nástrojmi. The Rspec book (Chelimsky, 2010) uvádza tri hlavné princípy, ktorými by sa mal BDD riadiť:"*

Co je TDD a co je BDD, se budu věnovat v závěru. Ona ta pointa je docela překvapivá :o). Tady bych poznamenal, že by bylo vhodně, že tyto tři principy definoval Dan North. David Chelimsky je sice významná osobnost v této oblasti, i tu knihu mám já, ale stálo by za to jasné říct, kdo to dávno předtím definoval. Ostatně Dan North je také významným spoluautorem[13] té knihy.

> *„Životný cyklus softwaru podľa BDD začína špecifikáciou funkcií, ktoré má software ponúkať, aby poskytol zákazníkovi prínos. Tento prístup sa v základe nelíši od tradičných top-down techník. Kľúčová je aplikácia bodu 1. na túto aktivitu. Fázu analýzy ukončujeme v okamihu, keď máme dostatok podkladov na to začať implementovať. Je dobré myslieť na to, že zákazník za nami prišiel, pretože primárne potreboval riešiť určitý problém. Prílišne detailná špecifikácia požiadaviek môže viesť tomu, že úspech projektu začne byť posudzovaný podľa rozpísaných požiadaviek a nie podľa toho, čo zákazník potrebuje, čo môže viesť k sklamaniu, keď zákazník po dodaní systému zistí, že jeho pôvodný problém zostáva nevyriešený."*

Rozhodně bych zde nehovořil o životním cyklu, to přenechme vývojovým metodikám jako Scrum, Lean a dalším. BDD je zejména technika[14], která se pak uplatňuje v některých metodikách. Raději bych mluvil o stylu a postupu než .

> *„Aby mohla byt teória uvedená do praxe, sú potrebné nástroje, ktoré vývojárom umožnia podľa metodiky BDD vyvíjať svoje projekty. Prvé použiteľne nástroje, ktoré tento smer postupujú, sa objavili pre programovací jazyk Ruby, pravdepodobne vďaka pestrej možnosti syntaktických konštrukcií, pomocou ktorej je možné jednoducho vytvárať domain-specific jazyky (DSL, viz Rails). Medzi najznámejšie dostupné BDD nástroje patria frameworky Rspec a Cucumber."*

Často míchá pojmy jako framework a nástroj. Dle mě to není totéž. Framework pro vás vytváří něco jako kostru, pomocí které něco stavíte (analogie s obyčejným lešením, mám praxi doma to stavět :o)). Nástroj je něco, co používáte k nějakému účelu (analogie s kladivem, pomocí něho můžete něco

13 http://www.pragprog.com/titles/achbd/the-rspec-book

14 http://en.wikipedia.org/wiki/Behavior_Driven_Development

ukovat i postavit – použijete dřevo a hřebíky). Framework (Rspec) v tomhle směru se zabývá jednotlivými částmi v podobě syntaxe (lešenářské trubky) vystavíte celek v podobě kódu (lešení) nebo hierarchii (moc složité lešení). Nástroj (Cucumber) něco zpracovává (kladivo zpracovává roztavené železo), v tomto případě user stories, resp. parsuje text v podobě user stories.

„Ako môžeme vidieť, tento výpis je podobný špecifikácií, ktorou by sme daný objekt popísali. Namiesto toho, aby sme ako jednotku testovania vybrali napr. metódu save a pomocou assertions očakávali výsledky pri jednotlivých typoch vstupov, sme vždy nadefinovali určitý kontext správania a v ňom popísali, ako sa má objekt a jeho metódy správať. Podobnosť so špecifikáciou nie je náhodná. Práve táto skutočnosť nám uľahčuje písanie testov pred implementáciou. Pri písaní štandardných unit testov je často očakávané, že poznáme štruktúru kódu, ktorý však ešte nie je naimplementovaný. Tým sa dostávame k problému sliepky a vajca: už počas vytvárania testu robíme rozhodnutia o tom, aké bude mať objekt metódy, a teda de facto implementujeme.“

„Okrem samotnej štruktúry prináša RSpec aj nástroje na vytváranie tzv. stubs (mocks), teda určitých náhrad za skutočné implementácie. Tomuto riešeniu pomáha zase z velkej časti dynamická povaha jazyku Ruby, ktorý jednoducho umožňuje definovať metódy za behu.“

„Objektu customer sme týmto príkazom vytvorili metódu name, ktorá pri jej volaní vráti hodnotu „Peter“. Takto vytvorený objekt môžeme použiť pri komunikácií s objektom, na ktorého chovanie sa sústreďujeme. Je možné taktiež definovať, že chceme, aby testovaný objekt použil určitú metódu stubu takto:“

„Táto konštrukcia je však odporúčaná len pre prípady, kde explicitne požadujeme použitie danej metódy (napr. zaznamenanie určitej činnosti do logu aplikácie). V ostatných prípadoch by to znamenalo zbytočné prepojenie so samotnou implementáciou.“

V této části práce vidím několik rozkolů. Některé problémy tkví v tom, jak autor pochopil uvedenou látku a co jsem napsal. Hlavní problém vidím, možná je to jen mé nepochopení, co se skrývá za tvrzením „Pri písaní štandardných unit testov je často očakávané, že poznáme štruktúru kódu, ktorý však ešte nie je naimplementovaný“. Jakýkoliv xUnit framework je pouze prostředek, nic neříká o tom, co se má dělat jak a proč. To je záležitost něčeho dalšího. Stejné psí kusy můžete dělat i v Rspec pomocí mocků, teda pokud rozumím té myšlence. Jen to nebude segedínském guláši ale v maďarském. Pohled test unit = TDD je chybný :o(. Rspec je ve své podstatě taky test unit.

V prvom rade je nutné uviesť, že iniciatíva BDD v žiadnom prípade nezavrhuje techniku TDD. Iba poukazuje na nedostatky, ktoré môže táto technika prinášať, a ponúka možné riešenie.

Z veľkej časti z TDD vychádza a stavia na jej základoch. Prínos BDD by mohol byť v pridaní ďalšej motivácie, prečo je písanie automatizovaných testov také dôležité – špecifikácia funkcií. Pri testovaní sa vývojár často stretáva so situáciou, že testovanie považuje za stratu času a má tendenciu sa mu vyhýbať. Vedenie projektu môže taktiež opomínať dôležitosť testovania, hlavne v spojení s blížiacimi sa termínmi odovzdania projektu, kedy je testovanie jedna z prvých činností, ktoré sú vynechávané z každodenných aktivít. Verím, že tieto nové postupy nájdu svoje uplatnenie a budú viesť k lepším produktom, príjemnejšej atmosfére pri vývoji a k potešeniu všetkých, ktorí sa tieto postupy rozhodnú používať.

Mám rád poznámku[15] Davida Chelimského na téma TDD vs. BDD. V konečné důsledku lze chápat BDD jako podmnožinu TDD, která se lépe ujímá role myšlenky TDD a definuje ji zřetelněji a čistěji v daném kontextu. Právě v tomhle s Davidem Chelimským souhlasím, že nelze nějak vymezovat TDD a BDD. Myšlenka je velmi podobná, jen se nějak zásadněji liší prostředky a cesty jak k nim dospět. Ostatně TDD technika zná user story, acceptance test-driven planning a podobně. Hlavní rozdíl vidím v těch vyjadřovacích prostředcích jako je například ten jazyk. Proto chci klást na toto důraz v té prezentaci na jaro. Oni se právě tohle snažili zdůraznit a rozlišit a více jít do té analytické části, protože požadavky je také nutno testovat :o).

Možná jsem úplně mimo já, ale vycházím z vlastních zkušeností z Londýna, kde jsem se účastnil přednášek Dana Northa a Aslaka Hellesøya, kolegů z práce tam a z toho, co jsem si doposud přečetl. Popravdě často je to jen souboj slovíček a toho, že si lidé nerozumí navzájem z důvodu, že tam to, je pro někoho něco naprosto jiného jako pro toho druhého. přes mou chabou znalost angličtiny jsem si všiml, že si občas přední představitelé BDD sem tam protiřečí, či spíše věci dávají do jiného kontextu než řekli jinde. Což právě Dan North přiznal po skončení své jedné přednášky. Tady už nastupuje filozofie, lingvistika a psychologie.

Jinak uvedenou práci nechci nijak shazovat. Dle mého názoru je to dobře zpracováno, hlavní pointa je tam uvedena. Jen myslím, že některé části nebyly šikovně formulovány nebo si přečetl špatně prameny, které uváděl. Jde zejména o práci Davea Astela, která se místy někdy docela rozcházel s tvrzeními kolegy.

Osobně bych tu práci nenapsal lépe, spíše naopak, mnohem hůře. Ono je snaží kritizovat než něco zpracovat a sestavit sám :o).

15 http://gilesbowkett.blogspot.com/2008/02/david-chelimsky-on-tdd-vs-bdd.html