VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

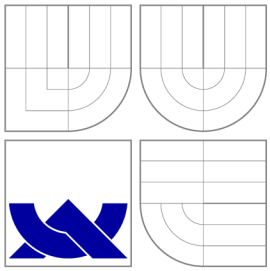
PŘÍPAD UŽITÍ PROCESU BDD V TESTOVÁNÍ APLIKACÍ V PHP

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

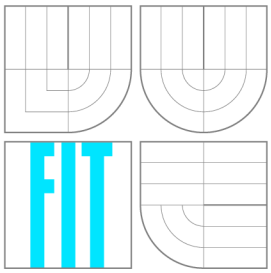
AUTOR PRÁCE
AUTHOR

FARIDA KUDAIBERDIYEVA

BRNO 2015



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

PŘÍPAD UŽITÍ PROCESU BDD V TESTOVÁNÍ APLIKACÍ V PHP

USE-CASE OF BDD PROCESS IN TESTING OF PHP APPLICATIONS

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — FARIDA KUDAIBERDIYEVA
AUTHOR

VEDOUCÍ PRÁCE — Ing. ALEŠ SMRČKA, Ph.D.
SUPERVISOR

BRNO 2015

Abstrakt

Bakalářská práce je zaměřena na odhalení hlavních výhod a nevýhod použití chováním řízeného vývoje softwarovými firmami jak v České Republice, tak i v zahraničí. Věnuje se odlišnostem a mezerám v tomto přístupu testování webových aplikací oproti klasickému přístupu. Za klasický přístup se považuje použití nástroje Selenium při tvorbě testovacích sad. Cílem práce je zjistit, zda použití BDD procesu je vhodnější a efektivnější oproti klasickému přístupu testování software. Pro dosažení cíle byly porovnány dva nástroje: PHP Selenium Client od Nearsoft a Behat. Kritéria zvolená pro porovnání jsou: pokrytí testovacích případů, způsob zápisu a vykonání scénáře, srozumitelnost pro počítačově nezaložené lidi, čas potřebný pro napsání testů, pravděpodobnost nezanesení chyby při vytváření testu a úroveň abstrakce testu. Z dosažených výsledků lze stanovit hlavní výhody a nevýhody použití každého z těchto nástrojů. Nejefektivnějším přístupem je kombinace obou tedy použití nástroje Behat s přidáváním vlastních vět a zdrojového kódu. Nejméně efektivní je způsob testování využitím pouze implicitních vět nástroje Behat. Přínosem této práce je zjištění, zda je BDD vhodný a efektivní pro společnost Dixons Carphone (dříve Dixons Retail). Zjištění je založeno na reálných příkladech.

Abstract

This bachalor's thesis is focused on revealing of main advantages and disadvatages of Behavior Driven Development using by software companies in Czech Republic and abroad. It is dedicated to diferences and grey areas of BDD testing technique in comparison with classic approach of testing of web application using Selenium framework. The goal of this thesis is to find out, whether using of BDD is more appropriate and effective than classic approach of testing. Two frameworks have been compared: PHP Selenium Client by Nearsoft and Behat. Criteria for comparing are: coverage of test cases, way of writing and execution of the scenario, understandability for non-programmers, time for tests creating, probability of not introducing new faults while creating the test, and the level of test's abstraction. From the achieved results is possible to define main pros and cons of each framework. The most effective way of testing is the combination of two of them, i.e. using of Behat framework with adding user-defined sentences and programming code. The least effective way of testing is using just sentences provided by Behat. Additional goal of this bachalor's thesis is to find out, if BDD is appropriate and effective for Dixons Carphone (earlier Dixons Retail) company by providing of independended view on the problem based on real examples.

Klíčová slova

Testování, automatické testování, akceptační testování, chováním řízený vývoj, Dixons Carphone, nástroj Behat, Selenium.

Keywords

Testing, automated testing, acceptance testing, Behavior Driven Development, Dixons Carphone, Behat framework, Selenium.

Citace

Farida Kudaiberdiyeva: Případ užití procesu BDD v testování aplikací v PHP, bakalářská práce, Brno, FIT VUT v Brně, 2015

Případ užití procesu BDD v testování aplikací v PHP

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Ing. Aleše Smrčky Ph.D.
Další informace mi poskytla firma Dixons Carphone.
Uvedla jsem všechny literární prameny a publikace, ze kterých jsem čerpala.

.....
Farida Kudaiberdiyeva
14. května 2015

Poděkování

Ráda bych poděkovala Ing. Aleši Smrčkovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce. Mé poděkování také patří zaměstnancům firmy Dixons Carphone za spolupráci a poskytnutí projektu pro testování.

Obsah

Kapitola 1

Úvod

Tato bakalářská práce vznikla jako reakce na použití chováním řízeného vývoje (angl. *Behavior Driven Development*) softwarovými firmami jak v České Republice, tak i v zahraničí. Tato ještě poměrně nová technika se v ČR teprve začíná prosazovat a zatím stále zůstává něčím nepříliš prozkoumaným. Jedním z představitelů použití chováním řízeného vývoje (dále jen BDD) v praxi v České Republice je firma **Dixons Carphone**, která mi poskytla možnost se blíže seznámit s projekty založenými na BDD. Zajímalo mě, jaké jsou hlavní výhody a nevýhody použití této techniky, jaké jsou odlišnosti a mezery (pokud nějaké jsou) v tomto přístupu testování oproti klasickému přístupu. Za klasický přístup je považováno použití nástroje **Selenium** při tvorbě testovacích sad. Z těchto předpokladů vyplynul **cíl** této bakalářské práce, kterým je zjistit, zda použití BDD procesu je vhodnější a efektivnější oproti klasickému přístupu testování software.

Při pohledu do historie se dozvíme, že agilní technika BDD vznikla jako odpověď na již existující - testy řízený vývoj (angl. *Test Driven Development*) - dále jen TDD. TDD byl docela skepticky vnímán programátory. Z jejich pohledu nedávalo smysl psát testy dřív než byl napsán alespoň nějaký zdrojový kód [18]. BDD řeší tento problém a navíc spojuje zákazníka, QA týmy, vývojáře a business analytiky dohromady. Protože procesu vytvoření specifikace účastní každý z nich, všichni vědí o co se jedná. Následující kroky vývoje jsou přesně definovány již od začátku. Stejné platí o testovacích scénářích. Přirozené názvy testů a Gherkin syntaxe (podkapitola 3.1) se používají proto, aby jednotlivým testům a scénářům mohl porozumět jak zákazník, tak i business analytik bez nutnosti chápání zdrojového kódu testů.

Existuje poměrně velké množství nástrojů, založených na BDD. Např.: *Behat* - napsaný pro PHP [13], *Concordion* - původně Java [14], *Cucumber* - Ruby, při použití zásuvního modulu *Cuke4Duke Maven* od Maven je podporována Java [18], *JBehave* - Java [15] a další. Podkapitola 3.2 je věnována podrobnějšímu popisu a porovnání těchto nástrojů.

Většina BDD nástrojů je založena na Gherkin syntaxi - podkapitola 3.1. Testovací scénáře obvykle využívají *given-when-then* šablonu (více o tom v kapitole 3) pro napsání testů. Proces jejich vytváření lze rozdělit do několika kroků:

- Vytvoření příběhu (angl. *story*).
- Přemapování příběhu do jazyka, který podporuje zvolený nástroj (angl. *glue code*).
- Spuštění testů a vytvoření reportu.

V rámci této bakalářské práce budeme pracovat s nástrojem **Behat**. Důvody zvolení tohoto nástroje a jeho vlastnosti jsou popsány v podkapitolách 3.2.1 a 3.2.5. Pro dosažení

stanoveného cíle je provedeno porovnání na testovací sadě napsané pomocí dvou nástrojů: **Behat** (BDD nástroj, jazyk – PHP) a **PHP Selenium Client od Nearsoft** (Selenium, jazyk – PHP). Kritéria hodnocení jsou:

a) pokrytí testovacích případů,

b) způsob zápisu a vykonání scénáře,

c) srozumitelnost pro počítačově nezaložené lidi,

d) čas potřebný pro napsání testů,

e) pravděpodobnost *nezanesení* chyby při vytváření testu a

f) úroveň abstrakce testu.

Byly také prostudovány dva způsoby práce s nástrojem Behat. To jsou vytvoření testů bez přidávání vlastního kódu, tedy využitím pouze implicitních vět, které Behat poskytuje, a vytvoření testů s přidáním vlastního kódu a vět.

Předmětem testování je projekt *Guest Checkout* (více o tom v podkapitolách 4.1 a 4.1.1) firmy **Dixons Carphone** zaměřený na anonymní nákup, který je implementován na stránce Currys[1]. Testovací scénáře byly také poskytnuty firmou **Dixons Carphone**.

[1] http://www.currys.co.uk

Kapitola 2

Automatické akceptační testování

Podle ANSI/IEEE 1059 standardu testování se definuje jako *proces analýzy software za účelem detekce rozdílu mezi existujícími a očekávanými stavy (sem patří vady/chyby/selhání) a také za účelem vyhodnocení vlastností software (dále jen SW).* Jinými slovy testování je proces validace a verifikace SW. Účelem testování je tedy poskytnout zákazníkovi co nejkvalitnější produkt (software).

Testování SW se provádí na několika úrovních. Některé z nich jsou uvedeny níže:

- **Jednotkové** (angl. *Unit*) - testování na úrovni zdrojového kódu. Účelem je zmenšit rozsah testování na vyšších úrovních a poskytnout uživateli základní metriku o spolehlivosti jednotek. Obvykle provádí vývojář.
- **Integrační** (angl. *Integration*) - ověření rozhraní modulů v podsystému, jejich předpoklady a vzájemnou komunikaci. Typy odhalených chyb: chyby v rozhraní a předpokládaných stavech systému.
- **Systémové** (angl. *System*) - účelem je zjistit, zda splňuje SW jako celek specifikaci požadavků. Dají se odhalit chyby v návrhu a specifikaci.
- **Akceptační** (angl. *Acceptance*) - zjištění, zda SW splňuje požadavky zákazníka a zda plní svůj primární účel.

V rámci této bakalářské práce se zaměříme na integrační (podkapitola 2.1) a akceptační (podkapitola 2.2) testování.

2.1 Integrační testování

Integrační testování je logickou nadstavbou jednotkového testování. Nejjednodušší formou je testování komunikace mezi dvěma již otestovanými dříve jednotkami, které byly sloučeny do jedné komponenty. V reálném světě se slučuje více jednotek do více komponent, které se následovně slučují do větších částí programu [6].

Cílem integračního testování je otestovat rozhraní a integrace uvnitř subsystému.

Existují 3 hlavní strategie používané v integračním testování [1]:

- **Velký třesk** (angl. *Big Bang*) - zahrnuje integraci modulů k sestavení kompletního systému. Tento přístup se považuje za vysoce rizikový z důvodu požadování

korektní dokumentace pro zabránění selhání. Výhodou tohoto přístupu je to, že systém je během testování již kompletní. Nevýhoda, která z toho vyplývá, je obtížnost nalezení důvodů selhání kvůli pozdější integraci.

- **Zdola nahoru** (angl. *Bottom-Up*) - zahrnuje testování nízkoúrovňových komponent, následované testováním vysokoúrovňových komponent. Testování pokračuje, dokud není otestována celá hierarchie komponent. Tato strategie poskytuje efektivní odhalení chyb a také umožňuje propojit vývoj a testování. To zvýší efektivitu výsledného systému.

- **Shora dolů** (angl. *Top-Down*) - v první řádě zahrnuje testování vyšších modulů (např. GUI nebo hlavní menu). Při testování systému nebo komponent se používají tzv. *stubs* simulující nízkoúrovňové jednotky, které nemusí být v době provedení testu k dispozici. Podsystémy se testují samostatně. Velkou výhodou této strategie je vysoká konzistence testovaného produktu, protože se testování provádí v prostředí blízkém realitě. Nevýhodou tohoto přístupu je to, že se základní funkcionalita systému testuje až na konci [8].

Lze se také setkat se strategií *Sandwich testing* (kombinace Zdola nahoru a Shora dolů) nebo s jinými kombinacemi zmíněných strategií.

Důležitým předpokladem pro integrační testování je úspěšný průchod komponent jednotkovým testováním.

2.2 Akceptační testování

Myšlenka automatického akceptačního testování přišla s extrémním programováním jako součást TDD. Místo toho, aby zákazník předával požadavky týmu vývojářů bez možnosti větší zpětné vazby, zákazník, vývojář a tester spolupracují na vývoji automatických testů, kde výstupem je hodnota, požadována zákazníkem. Akceptační se tomu říká proto, že tyto testy vyjadřují, co SW má dělat a jaké chování, by zákazník považoval za přijatelné. Test selže již v momentu vytvoření, protože zatím neexistuje žádný zdrojový kód (TDD). Bez ohledu na selhání, ten test je jasným signálem pro celý tým, co je pro zákazníka důležité a co je třeba udělat pro to, aby to odpovídalo zákazníkovým očekáváním [18].

Existuje několik typů akceptačního testování:

- **User Acceptance Testing** - testování provádějí uživatelé, kteří hodnotí, zda výsledný software odpovídá specifikovaným požadavkům. Jedním z možných způsobů provedení tohoto testování je poskytnutí uživatelům *beta trial* verzi systému na internetu [9]. Jinak, pokud se rozhodne o pozorování uživatelů, může to nést jak své výhody, tak i nevýhody. Zřejmou nevýhodou tohoto přístupu je stresující prostředí pro uživatele: všichni se nedokážou uvolnit, když jsou pozorováni. Druhým stresujícím faktorem je neznalost toho, co mají dělat. Často uživatelé nejsou schopni přemýšlet jako testeři, nevědí, co mají testovat a hlavně jak. Obvykle kvůli tomu uživatel není schopen otestovat složitější práci s aplikací a skončí u jednoduchých kroků [9].

- **Operational (Production) Acceptance Testing** - zaměřené na spolehlivost systému, na jeho schopnost se zotavovat apod.[2].

- **Contract Acceptance Testing** - testování, zda SW odpovídá všem smluvním kritériím.

- **Compliance (Regulation) Acceptance Testing** - testování, zda SW odpovídá bezpečnostním, právním nebo vládním předpisům [8].

Každý proces testování má své výhody i nevýhody, stejně tak akceptační testování. Níže naleznete některé klady a zápory tohoto typu testování.

(+) Výhody akceptačního testování:

- Akceptační testování může být jak manuální, tak i automatické. Automatické testování může být použito pro testování jakékoli webové stránky.
- Lze otestovat JavaScript a AJAX požadavky (manuálně i automaticky).
- Proces testování může být sledován zákazníkem nebo manažery.
- Je nejméně ovlivněno drobnými změnami v kódu [4].

(-) Nevýhody akceptačního testování:

- Nejpomalejší ze všech typů testování.
- Malý počet testů může vést k falešně správným výsledkům [4].
- Testování JavaScriptu může být obtížné v případě automatického testování.

Existuje řada nástrojů pro akceptační testování: *Cucumber, Behat - BDD (podkapitoly 3.2.3 a 3.2.1), Framework for Integrated Test (Fit) - open-source, FitNesse, Selenium (podkapitola 2.2.1), Mocha - založený na JavaScriptu* a jiné.

2.2.1 Testovací framework Selenium

Selenium [16] je automatizovaný nástroj pro testování webových aplikací. Autorem je Jason Huggins. Selenium plní úkoly prohlížečů řízením procesu prohlížení přes operační systém. Testy se spouštějí rovnou v prohlížeči. Tím simulují kroky reálného uživatele. Zmíněné testy lze použít jak pro akceptační testování, tak i pro testování kompatibility – testování webové aplikace v různých prohlížečích a pod různými operačními systémy.

Natroj Selenium se skládá z několika komponent:

- **Selenium IDE** (Integrované vývojové prostředí - angl. *Integrated Development Environment*) - nástroj pro vývoj testovacích případů. Existuje jako zásuvní modul pro prohlížeč Firefox. Má k dispozici kontextové menu, které nám dovoluje prvně zvolit UI element na aktuální stránce a potom zvolit ze seznamu jeden z příkazů nástroje Selenium s předdefinovanými parametry v závislosti na kontextu [16].
- **Selenium Remote Control (RC)** - nástroj pro vytvoření automatických UI zaměřených testů pro webové aplikace. Testy mohou být napsány v libovolném programovacím jazyce. Sestává ze dvou části:
  - Server, který automaticky spustí a ukončí prohlížeče. Chová se podobně jako HTTP proxy pro zpracování webových požadavků od těchto prohlížečů.
  - Knihovny na straně klienta pro zvolený programovací jazyk [16].

- **Selenium – WebDriver** - nástroj slučující v sobě obě dvě části komponenty Selenium RC (server + API). Byl vyvinut pro lepší podporu dynamických webových stránek, kde se elementy na stránce mohou změnit bez znovunačtení stránky. Cílem WebDriveru je dodat objektově-orientované API, které poskytuje vylepšená řešení moderních pokročilých problémů spojených s testováním webových aplikací [16]. Budeme ho využívat při testování stránky Currys[1]. Podporované programovací jazyky jsou: Java, C#, Python, Ruby, Perl - neoficiální, PHP - neoficiální, Javascript - neoficiální atd. Více o tom v podkapitole 2.2.2.

- **Selenium – Grid** - nástroj dovolující spuštění testů na několika strojích současně (paralelní vykonávání), řízení mnohonásobných prostředí z jednoho centrálního místa. V tomto případě jeden ze serveru se chová jako rozbočovač (angl. *hub*). Testy kontaktují tento server pro obdržení povolení k přístupu k instancím prohlížečů. Rozbočovač má k dispozici seznam serverů, které poskytují přístup k těmto instancím (*WebDriver nodes*) a povoluje testům tyto instance využívat [16].

2.2.2 Komponenta Selenium – WebDriver

Proto, abychom mohli vytvářet skripty pro komunikaci se Selenium Serverem (Selenium RC, Selenium Remote Webdriver) nebo pro vytvoření lokálního Selenium WebDriver skriptu, potřebujeme využít knihovnu pro konkrétní programovací jazyk. Jako první jsem hledala BDD nástroj pro testování (více o tom v podkapitole 3.2). Zvolila jsem nástroj Behat (podkapitola 3.2.5). Implementačním jazykem tohoto frameworku je PHP. Proto jsem se rozhodla provádět testování využitím Selenia také v tomto programovacím jazyce.

Pro jazyk PHP neexistuje oficiální verze knihovny, jsou však dostupné neoficiální od níže uvedených autorů:

- **Chibimagic**[2].
- **Lukasz Kolczynski**[3].
- Společnost **Facebook**[4].
- **Adam Goucher**[5] – doporučeno SeHQ.
- **PHPUnit Selenium**[6] – komunikace přes WebDriver API je implementována pouze částečně.
- Společnost **Nearsoft**[7].

Pro testování v rámci bakalářské práce jsem zvolila posledně uvedenou verzi PHP Selenium klienta, protože všechny dříve zmíněné nástroje, s výjimkou PHPUnit Selenia, se snaží co nejvíce napodobit oficiální knihovny pro jazyk Java, Python atd. Liší se pouze mírou té podoby. Nezvolila jsem doporučenou verzi knihovny z důvodu, že jsem chtěla zjistit rozdíl mezi oficiální a neoficiální verzemi knihovny.

[1] http://www.currys.co.uk
[2] `https://github.com/chibimagic/WebDriver-PHP/`
[3] `https://code.google.com/p/php-webdriver-bindings/`
[4] `https://github.com/facebook/php-webdriver`
[5] `https://github.com/Element-34/php-webdriver`
[6] `https://phpunit.de/manual/3.7/en/selenium.html`
[7] `https://github.com/Nearsoft/PHP-SeleniumClient`

PHP – SeleniumClient od Nearsoft

Knihovna PHP – SeleniumClient od Nearsoft dovoluje interakci s Selenium Serverem V2 v PHP. Komunikuje s WebDriver API skrz protokol – *JsonWireProtocol*.

Jedním z cílů této knihovny je poskytnout klientovi co největší podobu oficiálním knihovnám nástroje Selenium např. pro jazyk Java nebo C#. Větší část metod má stejný název jako v oficiálních knihovnách. To dovoluje vývojáři a/nebo testerovi aplikovat stejné techniky testování jako např. v Javě [10].

Požadované prostředí:

- PHP 5.3.8 a vyšší,
- PHPUnit,
- Curl,
- Nearsoft/PHPSelenium Client a
- SeleniumHQ Jar File.

PHPUnit

PHPUnit Selenium nabízí dvě testovací třídy: `PHPUnit_Extensions_Selenium2TestCase` a `PHPUnit_Extensions_SeleniumTestCase`. První třída umožňuje použití Selenium WebDriver API, které je však implementováno pouze částečně. Druhé rozšíření implementuje klient - server protokol pro komunikaci s Selenium serverem.

Je tu také k dispozici třída, která přidává možnost vytvoření *příběhů* (viz kapitola 3) – `PHPUnit_Extensions_Story_TestCase`. Metody `given()`, `when()` a `then()` reprezentují krok. Metoda `and()` představuje volitelnou spojku mezi kroky. Dále máme k dispozici abstraktní metody: `runGiven(&$world, $action, $arguments)`, `runWhen(&$world, $action, $arguments)` a `runThen(&$world, $action, $arguments)`, které musí být implementovány. Potom příběh, napsaný pomocí PHPUnit frameworku, může vypadat následovně:

Příklad 1:

```
class Jablka extends PHPUnit_Extensions_Story_TestCase
{
  public function rozdavaniJablek(){
    $this->given('Plny kos s jablky')
         ->when('Pepa si vzal nekolik jablek', 3)
         ->and('Kuba si vzal nekolik jablek', 5)
         ->and('Katka si vzala nekolik jablek', 2)
         ->then('Celkem si vzali jablek', 10);
  }
  public function runGiven(&$world, $action, $arguments){
    /* musí být implementováno*/
  }
  public function runWhen(&$world, $action, $arguments){
    /* musí být implementováno*/
  }
  public function runThen(&$world, $action, $arguments){
    /* musí být implementováno*/
  }
}
```

Kapitola 3

Chováním řízený vývoj

V průběhu testování se často potýkáme s otázkami typu co testovat/netestovat, jak hluboko se zaměřit při testování. Na tyto otázky našla odpověď nová agilní technika – *chováním řízený vývoj* (angl. *Behavior Driven Development*) - dále jen BDD [12].

BDD vznikl jako odpověď na již existující techniku – *testy řízený vývoj* (angl. *Test Driven Development*) - dále jen TDD. TDD je založen na myšlence napsat testy dříve než jsou napsány zdrojové kódy. To způsobilo negativní reakci ze strany vývojářů, kteří to považovali za ztrátu času. Další nevýhodou této techniky bylo to, že testy byly především zaměřeny na verifikaci tříd a metod a ne na to, co by zdrojový kód měl provádět.

Vznik BDD způsobil přechod od přemýšlení nad testy do přemýšlení nad chováním systému. To spojilo zákazníky, business analytiky, QA týmy a vývojáře dohromady. Proces BDD lze popsat následujícími kroky:

1. Zákazník popisuje business analytikovi své požadavky na výsledný systém.
2. Business analytik, vývojář a tester (někdy také zákazník) dávají požadavky na software dohromady a strukturují je do podoby scénářů.
3. Vývojář využívá tyto scénáře při vytváření SW.
4. Tester využívá tyto scénáře pro testování SW.

Tenhle postup je velice efektivní v případě, že nedošlo k žádné chybě při specifikaci produktu nebo při vytváření scénářů. Jinak následující kroky na konci mohou být zákazníkem úplně znehodnoceny. To lze považovat za velké riziko při volbě BDD.

Pokud se podíváme na proces vytváření scénářů a testů, setkáme se s první odlišností při práci s BDD a to jsou přirozené názvy testů, které jsou obvykle reprezentovány větami. Tedy název každého testu vyjadřuje jeho účel, proto testu může porozumět i člověk, který nemá moc zkušeností s programováním nebo testováním. Například test, který ověřuje, zda uživatel zadal povinný údaj (jméno), by mohl mít název `testSelzePriChybejicimJmene` [12].

S časem autoři BDD přišli na to, že pouze přirozený název testu nestačí pro vyjádření jeho chování a důvodu, proč test byl napsán. Proto vzniká další odlišnost BDD procesů od jiných a to jsou *given-when-then scénáře*, které podle [12] mají podobu:

> **Given** *some initial context (the givens),*
> **When** *an event occurs,*
> **Then** *ensure some outcomes.*

Klíčové slovo **Given** vyjadřuje počáteční podmínky pro spuštění testu. Může mít následující podobu:

> **Given** `I am on "example.com" page`.

Tedy testování začíná na stránce *example.com*.

Za **When** a **Then** obvykle následuje popis události a jejího důsledku. Pokud je potřeba popsat více událostí a/nebo důsledků lze je spojit klíčovým slovem **And**. Např.:

> **When** `I fill in "name" with "MyName"`
> **And** `I press "formSubmit"`
> **Then** `I should be on "submitPage.com"`
> **And** `I should see "Your name is MyName"`

Společně s *given-when-then* scénáři vzniká jazyk založený na BDD principech – *Gherkin*. Spolu s tím vznikají různé nástroje (*Behat*, *Concordion*, *Cucumber*, *JBehave* atd.), které využívají pevně definovanou Gherkin syntaxi.

3.1 Jazyk Gherkin

V této kapitole si přiblížíme syntaxi jazyka Gherkin. Klíčová slova uvedená níže se využívají nástroji *Behat* a *Cucumber*. Jiné nástroje mohou mít své ekvivalenty zmíněných zde klíčových slov a názvů souborů. Níže uvedená informace je založena na zdroji [17].

Gherkin je jazyk, který využívá odsazení textu pro definici struktur. Každá věta ukončená symbolem konec řádku je považována za *krok*. Většina řádků napsaných v Gherkinu začíná jedním z klíčových slov: *Feature:, Scenario:, Given, When, Then, And, But, Background* atd. Testovací sady jsou rozděleny do souborů s příponou `.feature` (dále v textu bude použito jako soubory s příběhy). Obsahem těchto souborů jsou *příběhy* (angl. *stories*).

Každý takový soubor musí začínat řádkem s klíčovým slovem *Feature:*. První věta následující za *Feature:* a ukončena symbolem konce řádku se překladačem interpretuje jako název testovací sady. Další maximálně tři odsazené řádky (pokud nějaké jsou) nejsou analyzovány překladačem a mají význam detailnějšího popisu testovací sady. Obvykle obsahují informaci o přínosu testovacího případu pro business analytiky.

Za *Feature:* následují scénáře začínající klíčovým slovem *Scenario:*, umístěné na novém řádku. Za slovem *Scenario:* může být na stejném řádku umístěn název scénáře. Jeden soubor s příběhy může obsahovat jeden a více scénářů. Jeden scénář sestává z jednoho nebo více kroků. Krok začíná jedním z klíčových slov: *Given, When, Then, And, But.*

Navíc Gherkin umožňuje vytvářet tzv. *scenario outlines*. Z důvodu neustálenosti českého ekvivalentu budeme používat anglický název. Jejich účelem je vytvořit šablonu scénáře, kterou lze vícenásobně spouštět s různými hodnotami. K tomuto se využívají: tabulka s hodnotami, kde počet řádků odpovídá počtu spuštění scénáře, a klíčová slova: *Scenario Outline:, Examples:, <název sloupce v tabulce s hodnotami>*. Viz příklad 3:

> **Příklad 2:** **Scenario Outline:** *Utrata penez,*
>
> **Given** `mame <celkem> penez`,
>
> **When** `utratime <cast> na darky k Vanocum`,
>
> **Then** `nam zustane <zustatek>`.
>
> **Examples:**

<celkem>	<cast>	<zustatek>
2000	1850	150
5000	3000	2000

Gherkin také umožňuje pomocí značek (angl. *tags*) uskupit několik *Feature* a scénářů, bez ohledu na to, v jakém souboru se nacházejí. Dalším prvkem Gherkin standardu jsou tzv. *Background* scénáře, které se spouštějí pokaždé před provedením každého následujícího scénáře. Označují se klíčovým slovem *Background:*.

3.2 Nástroje pro chováním řízený vývoj

BDD je mostem spojující přirozený jazyk (příběhy) s automatickými testy (JUnit apod.). Specifikace testů by měla být napsána ve srozumitelné pro business analytiky formě. To stejné platí pro případy, kdy test z nějakého důvodu selže. Člověk, který nemá zkušenosti s programováním, by tomu měl rozumět. To je hlavním cílem BDD nástrojů. Poskytnout srozumitelnost a přívětivost pro počítačově nezaložené lidi. Jak již bylo zmíněno v kapitole 3, při tvorbě testů se používají *given-when-then* scénáře a Gherkin syntaxe.

3.2.1 Nástroj Behat

Behat je jedním z BDD zaměřených nástrojů, který pro vytvoření scénářů a příběhů využívá syntaxi jazyka *Gherkin* (viz kapitola 3.1), založený na *given-when-then scénářích*. Implementačním jazykem je PHP. Příběhy se ukládají do souborů s příponou `.feature`, zdrojové kódy jsou potom ve `FeatureContext.php`.

Při vytváření nových vět ve scénářích Behat nám automaticky nabízí šablonu pro nový test, kde název testu odpovídá zadané větě. Základní vztah je: 1 řádek = 1 krok. Při dodržení tohoto vztahu lze jednoduše vystopovat, kde a proč k chybě došlo bez náhledu do zdrojového kódu. Proto tomu může porozumět i člověk, který nemá moc společného s programováním.

Behat nám také nabízí možnost vytvoření příběhů v 10 různých jazycích včetně češtiny. V českém jazyce to může mít následující podobu[1]:

Příklad 3:

Scénář: *Vymazání paměti agentovi*

Pokud `existuje agent "A"`

A také `existuje agent "B"`

Když `vymažu paměť agentovi "B"`

Pak `by měl existovat agent "A"`

Ale `agent "B" by existovat neměl`

(+) Výhody použití nástroje Behat:

- Jednoduchá orientace pro PHP vývojáře.
- Struktura souborů s příběhy je jednoduchá a srozumitelná.
- Poskytuje implicitní věty pro vytvoření příběhů. Není třeba vytvářet vlastní kódy.

[1] Příklad převzat z: bin/behat –lang=cs –story-syntax

- Testování webových aplikací je podporováno komponentou *Mink* založenou na nástroji Selenium.
- Dobrá dokumentace.
- Podpora pozadí (angl. *background*).
- Podpora značek (angl. *tags*).
- Při vytvoření nové věty, nabídne hotovou šablonu pro doplnění do `FeatureContext.php` souboru.
- Je podporován několikařádkový vstup (tabulky/PyStrings).

(-) Nevýhody použití nástroje Behat:

- Některé Selenium funkce nejsou podporovány (např. přepínání mezi okny). Pro jejich použití je nutné upravovat zdrojové soubory nástroje Behat (platí pro verzi 2.4 stable).
- V některých případech si nevystačíme s implicitními větami, je nutné vytvářet vlastní.

3.2.2 Nástroj Concordion

Hlavní odlišností nástroje *Concordion* je způsob zápisu specifikací. V tomto případě specifikace se zapisuje pomocí HTML kódu a implementace samotných testů v Javě. Podobně jako *Cucumber* - podkapitola 3.2.3, podporuje hodně jiných jazyků a platforem. Aktuálně existují verze pro .NET, Python, Fantom, Scala a Ruby.

Není vázán na BDD syntaxi (Gherkin). Tester a business analytik v tomto případě mají volnost při zápisu specifikace, proto specifikace může být napsána přirozeným jazykem. Lze také do specifikace přidávat obrázky či tabulky [14].

(+) Výhody použití nástroje Concordion:

- Velký výběr implementačních jazyků.
- Pěkný vzhled specifikací.
- Podpora JUnit.
- Jednoduché zprovoznění.
- Přirozený jazyk při zápisu specifikace.
- Lze použit jako dokumentaci.

(-) Nevýhody použití nástroje Concordion:

- Tester/business analytik musí mít alespoň základní znalost HTML kódů.
- Neodpovídá Gherkin standardům.

3.2.3 Nástroj Cucumber

Původně implementačním jazykem je Ruby. Při použití zásuvního modulu *Cuke4Duke Maven* od Maven je podporována Java [18] (Cucumber-JVM). Proces vytváření testů se řídí Gherkin standardy. Příběhy se ukládají do souborů s příponou `.feature`. Vyvinut především pro akceptační testování.

(+) Výhody použití nástroje Cucumber:

- Velký výběr implementačních jazyků.
- Struktura souborů s příběhy je jednoduchá a srozumitelná. Plně odpovídá Gherkin standardům.
- Podporuje více jak 40 jazyků pro popis příběhů [18].
- Dobrá dokumentace v tištěné podobě - zdroj [18].
- Podpora pozadí (angl. *background*).
- Podpora značek (angl. *tags*).
- Pěkné implicitní formátování test reportů.
- Je podporován několikařádkový vstup.

(-) Nevýhody použití nástroje Cucumber:

- Nedostačující dokumentace pro Cucumber-JVM.
- Cucumber-JVM nepodporuje HTML výstup test-reportů.
- Je obtížné nastavit vlastní test-report formátování.

3.2.4 Nástroj JBehave

JBehave je nástroj podporující BDD, pokročilý TDD a také akceptační - TDD (angl. *Acceptance - Test Driven Developement*) - dále jen ATDD [15]. Autorem JBehave je Dan North - jeden z autorů BDD [12]. Implementačním jazykem je Java. Proces vytvoření testů sestává z několika kroků [15]:

- **Vytvoření příběhů.** Příběhy se zapisují do souboru s příponou `.story`. Každý krok musí začínat klíčovým slovem (*Given-When-Then* nebo *And*)
- **Mapování kroků do odpovídajících metod v jazyce Java.** Příklad 4 je převzat ze zdroje [15]:

  Příklad 4:

  ```
  @When("I toggle the cell at ($column, $row)")
  public void iToggleTheCellAt(int column, int row) {
       game.toggleCellAt(column, row);
  }
  ```

- **Konfigurace třídy `Java Embeddable`, která spojuje JBehave s příběhy ve souborech s příponou `.story`.** Provádí se pouze jednou. Nejjednodušší konfigurace je 1:1, tedy jedná třída odpovídá jednomu souboru s příběhem.

- **Spuštění a prohlížení reportu.** Spouštět testy lze jako JUnit testy z jakéhokoli IDE, které podporuje jazyk Java. Reporty jsou přizpůsobitelné volbě uživatele.

(+) Výhody použití nástroje JBehave:

- Jednoduchá orientace pro Java vývojáře.
- Struktura souborů s příběhy je jednoduchá a srozumitelná.
- Dobrá integrace s IDE Eclipse a Maven (další velká výhoda pro Java vývojáře).
- Testování webových aplikací je podporováno vlastní JBehave komponentou založenou na nástroji Selenium.
- Vysoce konfigurovatelný.
- Pěkné HTML formátování test reportů.
- Dobrá dokumentace.

(-) Nevýhody použití nástroje JBehave:

- Konfigurace je někdy složitá a matoucí.
- Podporuje pouze příběhy (angl. *stories*), ne *feature*.
- Není podporován několikařádkový vstup.
- Není podporováno pozadí (angl. *background*) – neodpovídá *Gherkin* standardům [7].

3.2.5 Volba nástroje

Při rozhodování o volbě nástroje pro mě bylo klíčové, aby nástroj plně odpovídal Gherkin standardům. Po této úvaze se výběr zúžil na dva nástroje: *Cucumber* a *Behat*. Vzhledem k tomu, že společnost *Dixons Carphone* implementuje všechny své projekty v jazyku PHP, rozhodla jsem se použít stejný skriptovací jazyk pro testování. Proto jako testovací nástroj byl zvolen nástroj Behat.

Kapitola 4

Případ užití pro srovnání různých přístupů testování

Vytvořená testovací sada je založena na scénářích z dokumentu poskytnutého firmou *Dixons Carphone — Request For Change 2433 Guest Checkout* [3]. Tento dokument obsahuje kompletní specifikaci projektu anonymní nákup (angl. *Guest Checkout*) – podkapitola 4.1 (anonymní nákup na e-shopu implementovaný na stránkách Currys[1] a PC World[2]) a testovací scénáře. Jako jazyk pro vytvoření příběhů byla zvolena angličtina, protože je *Dixons Carphone* anglickou firmou a celá specifikace je napsána také v tomto jazyce.

Testovací sada se skládá z 21 testovacích případů, které pro dosažení cíle byly napsány několika způsoby a byla porovnána efektivita každého z přístupů. Zvolené způsoby jsou:

- Napsat testovací sadu využitím pouze implicitních vět nástroje Behat - podkapitola 4.2.1.
- Napsat testovací sadu využitím implicitních vět nástroje Behat s přidáváním vlastních vět a zdrojového kódu - podkapitola 4.2.2.
- Napsat testovací sadu využitím nástroje Selenium - podkapitola 4.3.

4.1 Rozšíření systému e-komerce – anonymní nákup

Rozšíření systému e-komerce – anonymní nákup (angl. *Guest Checkout*), dále jen Guest Checkout, je jednou ze známých praktik v e-komerci. Smyslem této praktiky je dovolit uživateli nakupovat na internetu bez nutnosti registrace. Jinými slovy se jedná o nákup v e-obchodech jako hosté. Výhodou použití *Guest Checkout*u je usnadnění a urychlení nákupu pro uživatele.

4.1.1 Guest Checkout na currys.co.uk - Dixons Carphone

Guest Checkout na webové stránce Currys je implementován podle specifikace v dokumentu *Request For Change 2433 Guest Checkout* - [3]. Zmíněný dokument specifikuje požadavky na umožnění uživatelům nakupovat zboží bez nutnosti se registrovat, tedy nakupovat jako hosté.

[1] `http://www.currys.co.uk`
[2] `http://www.pcworld.co.uk`

Uživatelé mohou nakoupit jakékoli zboží nebo službu jako hosté s výjimkou stahovatelných zboží (angl. *downloads*). Proces registrace je pro zákazníky stále dostupný.

Registrovaní uživatelé mohou nakupovat buď jako hosté nebo jako registrovaní zákazníci. Pokud se jedná o nového uživatele (jeho emailová adresa není uložena v databázi), bude mu nabídnuta možnost registrace. Pokud uživatel bude chtít pokračovat jako host, nebude následně žádán o heslo. Potřebuje však v tomto případě pokaždé zadávat adresu pro doručení a placení zboží. Žádný z *Guest Checkout* nákupů není bez souhlasu registrovaných zákazníků zobrazován na jejich profilové stránce *My Account*.

4.2 Testovací sada s využitím nástroje Behat

Testovací sada založená na nástroji Behat sestává z 21 testů. Scénáře jsou převzaty z dokumentu *Request For Change 2433 Guest Checkout*.

Prvním krokem je instalace. Instalace Behatu se provádí využitím nástroje *Composer*. Předem je nutné vytvořit soubor `composer.json`, který bude obsahovat všechny potřebné informace pro instalaci (pro více informací viz zdroj [13]). Je nutné, aby verze nástroje Behat, Selenium–standalone serveru a prohlížeče Firefox byly kompatibilní. Verze, se kterými pracuji, jsou:

- **Operační systém** — MacOS X 10.10.2:
  - **Behat** — 2.4.6, **Mink**[3] — 1.5.0, **Gherkin** — 2.2.9,
  - **Selenium Server** — 2.43.1,
  - **Firefox** — 33.1.1.

Jako první se musí spustit Selenium Server: `java -jar selenium-server-standalone--2.43.1.jar`. Potom následuje spuštění jednotlivých/všech testů. Spuštění se provádí z příkazového řádku pomocí příkazu: `bin/behat [[--tags=@...] ...]`.

Verze nástroje Behat, kterou jsem použila pro testování, nepodporuje funkci přepínání mezi okny a načtení názvů aktuálně otevřených oken prohlížeče do pole. Zmíněné funkce jsou dostupné na webu nové verze nástroje Behat/Mink[4]. Z důvodu nestability zmíněné nové verze, nelze ji stáhnout přes *Composer*, proto bylo nutné upravit zdrojový kód souboru `Selenium2Driver.php` ručně. Výsledek operace `diff -u` nad starou a novou verzemi tohoto souboru vypadá následovně:

Příklad 5:

```
--- Selenium2Driver.php.old.php 2015-03-03 19:58:08.000000000 +0200
+++ Selenium2Driver.php 2015-04-18 13:28:25.000000000 +0200
@@ -427,7 +427,29 @@
-
+ /*
+ ** Přidáno~manuálně z url
+ ** https://github.com/pthurmond/MinkSelenium2Driver/blob/master/
     src/Behat/Mink/Driver/Selenium2Driver.php
+ */
+ /**
+  * Return the names~of all open windows
+  *
```

[3] viz podkapitola 3.2.1

[4] Dostupné na `https://github.com/Behat/MinkSelenium2Driver`

```
+   * @return array+ Array of all open window's~names.
+   */
+ public function getWindowNames()
+ {
+     return $this->wdSession->window_handles();
+ }
+ /**
+   * Return the name of the currently active window
+   *
+   * @return string+ The name of the current window.
+   */
+ public function getWindowName()
+ {
+     return $this->wdSession->window_handle();
+ }
```

4.2.1 Testovací sada bez přidávání vlastního zdrojového kódu

Hlavní výhodou využití implicitních vět pro tvorbu testovací sady je zjednodušení práce pro testera. Například je nám k dispozici věta, která zvolí jednu z možností (option) z výběrového pole (select)[5]:

When `/^(?:|I ) select "(<option>)" from "(<select>)"`

V tomto případě tester nemusí mít příliš hluboké znalosti programování. Důležité je, aby se dokázal orientovat v HTML kódu stránky a správně identifikovat potřebný element. V závislosti na větě, kterou použije, má možnost se odkázat na element pomocí *id, name, label, value, CSS selectoru* apod.

Potom *příběh* může vypadat následovně (podle syntaxe Gherkin):

Příklad 6:

@25.21

Scenario: *Customer Eric Evans with download product and small box product in basket enters recognised email address. System displays "Welcome back Eric"and prompts for password. Customer enters wrong password. System shows invalid password message. Guest checkout option is greyed out and customer enters valid password and system progresses to delivery options screen[3].*

...

When `fill in "search" with "iphone 5c tough naked case - clear"`

Then `I press "search_bttn"`

And `I follow "CASE-MATE iPhone 5c Tough Naked Case - Clear"`

...

And `I should see an`[6] `"div.btn.btnDisabled" element`

...

Nevýhoda tohoto přístupu vyplývá z omezeného počtu implicitních vět, které neumí pokryt všechny možné stavy systému. Scénář napsaný tímto způsobem se odlišuje pevnými kroky a pevnou definici elementů, které při testování používá. Problém nastává v případech,

[5] Seznam všech dostupných vět: `bin/behat -di`

[6] Neurčitý člen *an* je součástí implicitní věty.

kdy například potřebujeme vyhledat nějaké zboží ze seznamu. Pomocí vět lze odkázat pouze na konkretní zboží podle názvu, id apod. Může ale nastat případ, kdy se toto zboží přestane prodávat. To potom povede k selhání testu. Tedy pomocí implicitních vět neumíme říci, že potřebujeme *nějaké* zboží, které má *nějaké* vlastnosti (je skladem, lze dovést domu apod.). Umíme odkázat pouze na konkretní zboží. Z toho vyplývá, že úroveň abstrakce implicitních vět je hodně nízká.

Další problém spojený s volbou tohoto přístupu se vztahuje k nepřirozenosti některých vět po přidání identifikátorů. Podle pravidel BDD [5] CSS selektory a technické názvy elementů by se neměly ve specifikaci a scénářích používat. V tomto případě nastává otázka, jestli je přirozenost vět doopravdy tak důležitá a jaká míra nepřirozenosti je pro nás přijatelná. Protože, pokud nechceme použít technické názvy některých elementů pro definici kroku, přicházíme o možnost použít implicitní věty. To vede k tomu, že i ty nejjednodušší věty musíme programovat sami. V tomto případě jediným rozdílem mezi nástroji Behat a PHP Selenium Client od Nearsoft se stávají soubory s příběhy.

4.2.2 Testovací sada s přidáváním vlastního zdrojového kódu

Důležitou výhodou přidávání vlastního kódu a vlastních vět do scénářů je urychlení napsání testovacích kroků. Tak například proces vyplnění formuláře s osobními údaji pomocí implicitních Behat vět lze rozepsat na 7 až 10 řádků, kde každý z nich se liší od předchozího pouze ID elementu, do kterého se zapisuje, a textem, který se do tohoto elementu zadává. Příklad 7 představuje část jednoho z takovýchto testů:

Příklad 7:

@25.29

Scenario: *Customer Dave Dawson with small box product in basket (test 25.18) selected guest checkout option. Dave enters guest details where name, delivery address and billing address are all the same as his account already holds* [3].

...

Then `I press "guest_checkout"`

And `I fill in "sFirstname" with "Dave"`

And `I fill in "sLastname" with "Dawson"`

And `I fill in "sDelPostalCode" with "AB10 1AG"`

And `I fill in "sDelAddressLine" with "address 123"`

And `I fill in "sDelCity" with "London"`

And `I fill in "sDelPhone" with "0800 0929090"`

Then `I press "formSubmit"`

...

Tento způsob zápisu je funkční a správný, není ale efektivní. Pomocí přidání vlastního kódu a vytvoření nové věty, výše uvedený příklad 7 lze zapsat následujícím způsobem (viz příklad 8):

Příklad 8:

@25.27

Scenario: *New customer Gordon Green (test 25.24) enters name, delivery address and different billing address and continues to delivery option screen* [3].

...

Then `I press "guest_checkout"`

And `I fill form with:`

sFirstname	sLastname	sPostalCode	sAddressLine1	sCity	sPhone
Gordon	Green	M40 8NJ	address 123	Manchester	0800 1929095

Then `I press "formSubmit"`

...

Sémantika obou dvou testovacích případů zůstala stejná, změnila se ale efektivita zápisu. Cenou za efektivitu je ale menší přehlednost při selhání testu. V případě implicitních vět (příklad 7) je moc dobře vidět, co se nepodařilo a na jakém kroku test neprošel. V druhém případě (příklad 8) to tak jasné není. Pomoci v tomto případě může snímek obrazovky (který je třeba také ručně nastavit pomocí zdrojového kódu) nebo studování zdlouhavé výjimky. Jinak se v případě selhání zčervená celý krok **And** `I fill form with:` a nebude z toho patrné, co přesně chybu způsobilo.

Dalším možností použití vlastních vět je příklad 9, kde věty `When I buy a product and "continue"` a `And I buy a download` zaručí, že virtuální zákazník Harry Hill si vloží do košíku nějaké pouzdro pro *iphone 5c*, které je dostupné pro vyzvednutí v obchodě a pro dovoz domu, a bude pokračovat v nákupu stahovatelného souboru *membership*. To, že za jeho emailovou adresou následuje upřesnění *in London*, znamená, že bude chtít první vložené do košíku zboží, tedy pouzdro pro iphone 5c, vyzvednout v Londýně.

Příklad 9:

@25.25

Scenario: *New customer Harry Hill places reserve and collect order. Harry Hill with download and small box product in basket enters same email address in checkout process. System does not display "Welcome back"message and continues to Guest Details screen. Check email address previously entered for reserve and collect can be used* [3].

Given `I am on homepage`

When `I buy an "iphone 5c case" product and "continue"`

And `I buy a "membership" download as "harryHill@email.com : in London"`

...

Stejnou větu `When I buy a product` lze také použít jiným způsobem. Viz příklad 10. Použití odlišných klíčových slov **When, Then, And, But** ve stejné větě překladač nerozlišuje. To neplatí pro slova typu **Given, Background** apod.

Příklad 10:

@25.48

Scenario: *New customer George with big box product in basket is offered option to create an account, but they do not want to do that, so they ignore the create account and choose a card type and continue to payment details screen for their guest checkout order* [3].

Given `I am on homepage`

Then `I buy a "tv 55" product as "newGeorge@email.com"`

...

Z výše uvedeného příkladu 10 je vidět, že se formát věty změnil i přesto, že se jedná o stejnou větu jako v příkladě 9. Z toho vyplývá další výhoda tohoto přístupu – máme volnost při vytváření vlastních vět. Záleží pouze na rozhodnutí testera, jak tu větu zapíše a jak ji bude chtít používat. Lze tím potom zaručit srozumitelnost pro počítačově nezaložené lidi. S volností však přichází větší pravděpodobnost zanesení chyby při vytváření testu, se kterou je třeba také počítat při volbě této metody testování.

Identifikace zmíněné věty v souboru `FeatureContext.php` vypadá následovně:

Příklad 11:

```
/**
 * @Then /^I buy (?:a|an) "([^"]*)" product (?:as|and) "([^"]*)"$/
 * @param $searchProduct - název vyhledávaného zboží
 * @param $email - email zákazníka nebo 'continue' (pokračování v
     nákupu)
 */
public function iBuyAProductAs($searchProduct, $email){
 /* zdrojový kód */
}
```

4.3 Testovací sada s využitím nástroje Selenium

První problém se kterým jsem se setkala byla volba nástroje Selenium pro jazyk PHP. Oficiální verze nástroje Selenium pro tento skriptovací jazyk neexistuje. Proto jsem musela volit jeden z několika neoficiálních (viz podkapitola 2.2.2). První moje volba nebyla úspěšná. Nástroj **PHPUnit Selenium** nepodporoval některé funkce, které jsem potřebovala. Z tohoto důvodu jsem zvolila **PHP Selenium Client** od Nearsoft (podrobněji o tomto nástroji přečtete v podkapitole 2.2.2).

Odlišnost práce s PHP Selenium Clientem spočívá hlavně v tom, že testerovi nejsou k dispozici žádné pomocné funkce, pokud nebudeme brát v úvahu např. vyhledávaní elementů. To znamená, že než se tester pustí do pohodlnějšího a rychlejšího testování, musí vydělit nějaký čas na vytvoření takových funkcí (metod). Výsledkem je větší množství času pro napsání testovacího případu a stejně jako v případě nástroje Behat s přidáváním vlastního kódu zvyšujeme pravděpodobnost zanesení chyby.

Navíc, pokud budeme chtít kombinovat BDD s klasickým přístupem testování a sledovat, jaké kroky se přesně vykonávají, je nutné přidávat za každým krokem výpis popisující daný krok. Protože bez takovýchto výpisů srozumitelnost testů pro počítačově nezaložené lidi o hodně klesá.

Příklad 12 představuje jednu z pomocných funkcí, kterou jsem vytvořila pro vyplnění formuláře na základě hodnot asociativního pole `$form`, ve kterém klíč je ID elementu, hodnota - hodnota, která bude vložena do políčka. V tomto případě lze také provést paralelu s vyplněním formuláře pomocí tabulky v nástroji Behat (příklad 8). Datový typ `Table`, který je použit v testu 8, představuje vícerozměrné asociativní pole, kde jeden řádek tabulky odpovídá jednomu z těchto polí. V případě mé vlastní funkce 12 lze navíc do asociativního pole `$form` přidat hodnotu potvrzovacího tlačítka s klíčem `submit`.

Příklad 12:

```
public function fillForm($form){
    if(is_array($form)){
      foreach ($form as $key => $value){
        if($key === "submit")
```

```
            $this->clickOnElement(By::id($value));
          else
            $this->fillField(By::id($key),$value);
        }
      }
      else{
        throw new Exception("Array required",1);
      }
    }
```

Funkce využívá také další mnou vytvořené pomocné metody `fillField()` a `clickOnElement()`. Použití funkce `fillForm()` (příklad 12) v testu vypadá následovně:

Příklad 13:

```
/*
** Customer with an account has confirmation of their guest checkout
     order
*/
public function test25_55(){
...
 $this->form["sFirstname"] = "Dave";
 $this->form["sLastname"] = "Dawson";
 $this->handler->fillForm($this->form);
...
}
```

Vyplnění formuláře jsem prováděla třemi způsoby. Jeden z nich je pomocí asociativního pole, tedy uvedeny dříve (příklady 12 a 13). Druhý způsob využívá mou vlastní funkci `fillFormAs()`, která má jméno a příjmení virtuálního zákazníka jako parametry. Tato funkce byla vytvořena z důvodu, že virtuální zákazníci, kteří byli použiti pro testování, mají stejné osobní údaje s výjimkou jména a příjmení a také bylo třeba pokaždé vyplnit stejný formulář. Tedy v této funkci jsou ID políček a hodnoty zadány napevno. Třetí způsob je použití funkce `fillField()`, kde definujeme každé políčko zvlášť (podobně jako test 7, kde se využívá pouze implicitní věta Behatu `I fill in ""with ""`). Poslední způsob ztrácí efektivitu v případech, kdy jedna testovací třída obsahuje několik testů, kde je třeba vyplnit stejný formulář a kde se mění jen málo hodnot. V tomto případě místo opisování stejného kódu lze použít asociativní pole, kde změníme pouze ty hodnoty, které se liší. Navíc metoda `fillForm()` umožňuje vyplnit jakýkoliv formulář.

Stejné funkce lze bez problémů implementovat i v Behatu. To vede k závěru, že co se týká doplnění vlastního kódu do `FeatureContext.php`, moc velký rozdíl ve složitosti není patrný. Pokud ale je pro nás důležité dodržení BDD praktik (aby testům porozuměli také programátorsky nezaložení lidé), musíme navíc za každým krokem přidávat výpis toho, co se přesně provedlo. To potom lze považovat za komplikaci, která navíc znepřehledňuje zdrojový kód testů.

4.4 Výsledky srovnání různých přístupů testování

Podle dosavadních výsledků a závěrů lze usoudit, že záleží především na tom, na co je kladen důraz při vývoji a testování SW. Každý z přístupů má své klady i zápory. Hodnocení třech přístupů testování (Behat bez přidávání vlastního zdrojového kódu – *Behat bez*

vlastního kódu, Behat s přidáváním vlastního zdrojového kódu – *Behat + vlastní kód*, testování pomocí Selenia – *PHP Selenium Client*) podle níže uvedených kriterií je znázorněno v tabulce 4.1.

Kritéria hodnocení:

a) pokrytí testovacích případů,

b) způsob zápisu a vykonání scénáře,

c) srozumitelnost pro počítačově nezaložené lidi,

d) čas potřebný pro napsání testů,

e) pravděpodobnost *nezanesení* chyby při vytváření testu a

f) úroveň abstrakce testu.

Žádné chyby během testování nebyly odhaleny z důvodu vyspělé implementace rozšíření *Guest Checkout*, proto procento odhalených chyb není uvedeno v tabulce.

	Behat bez vlastního kódu	**Behat + vlastní kód**	**PHP Selenium Client**
a)	Nešlo pokrýt všechny případy	Byly pokryty všechny testovací případy	
b)	(+)přehledné kroky (-)neefektivní zápis	(-)nepřehledné kroky (+) efektivní zápis	(-) manuální výpis (+) efektivní zápis
c)	**Nízká:** při použití css selektorů	**Vysoká:** vlastní věty	**Hodně nízká:** bez výpisů kroků
d)	**Odhadem 15min/test:** Rychlé vytvoření testů	**Odhadem 15min/test +30min:** na vlastní kód	**Odhadem 18min/test +37min:** pomocné funkce
e)	**Velká:** 9.5×10^{-5}%, bez vlastního kódu	**Střední:** 1.32×10^{-21}%, chyby ve vlastním kódu	**Malá:** 2.068×10^{-23}%, chyby ve vlastním kódu + pomocné funkce
f)	**Nízká:** bez vlastního kódu	**Vysoká:** použití vlastního kódu	

Tabulka 4.1: Hodnocení přístupů testování podle kritérií a) až f).[7]

Poznámka k tabulce: Barva políčka vyjadřuje úspěšnost testovací metody. Zelená [■] znamená vysokou úspěšnost, žlutá [■] - průměrnou a červená [■] - nízkou. Označení "(+)/(-)"nese význam kladu/záporu. "+"znamená "navíc".

Podle výsledků z tabulky je vidět, že metoda testování *Behat + vlastní kód* je nejefektivnější. Přestože *PHP Selenium Client* měl více nízkých hodnot, za nejméně efektivní přístup považuji metodu *Behat bez vlastního kódu*. Protože označuji nemožnost pokrýt všechny testovací případy a nízkou úroveň abstrakce testů za závažnější problém, který potom povede k větším potížím s udržovatelností testů, než u dvou zbylých metod. Hodnoty kritérií c) až e) v případě PHP SC záleží především na schopnostech testera. Tedy jsou ovlivnitelné. To neplatí pro kritéria a) a f).

Čas potřený na napsání testů byl vypočítán přibližně. Bral se v úvahu průměrný čas pro vytvoření jednoho testu v minutách. K tomu se navíc přičetl čas pro vytvoření vlastního kódu a pomocných funkcí. Protože pomocí implicitních vět nešlo pokrýt všechny testovací případy, nebyl vypočten celkový čas pro vytvoření kompletní testovací sady.

Výpočet pravděpodobnosti nezanesení chyby při tvorbě testu jsem počítala pomocí *binomického rozdělení*, kde jako počet pokusů jsem použila počet řádků zdrojového kódu/vět v testu 8. Použila jsem vzorec 4.1.

$$P(K) = \binom{N}{K} \times \ (p)^K \times \ (1-p)^{N-K} \tag{4.1}$$

Písmeno K označuje počet chyb, N – počet řádků zdrojového kódu/vět, p – pravděpodobnost chyby na jednom řádku, tady budeme pracovat s hodnotou 1/2, protože řádek buď chybu obsahuje nebo ne. *P(K)* je potom pravděpodobnost, že ke K chybám dojde, v našem případě $K = 0$, tedy bez chyb.

Kapitola 5

Závěr

Tato práce se zaměřuje především na rozdíly mezi BDD a klasickým přístupem testování. Důležité bylo zjistit jaké, každý z přístupů, má slabé stránky, jaký přístup je vhodnější a pro jaké účely.

Pokud je pro společnost důležité, aby zákazník, business analytik, tester a vývojář spolupracovali a aby každý měl přehled o tom, co se vyvíjí a v jakém je vývoj aktuálně stadiu, BDD je dobrou volbou. Další otázka, která nastává, je do jaké míry chceme BDD v praxi využít. Podle pravidel BDD – [5, 11], se nesmí ve specifikaci a scénářích použít žádný technický název elementu. To znamená, že v případě Behatu přicházíme o možnost použít některé implicitní věty. Takové rozhodnutí potom může vést k tomu, že tester bude nucen přepisovat implicitní větu do vlastní s tím rozdílem, že vlastní věta místo technického názvu elementu bude obsahovat přirozený. Z hlediska efektivity a času tento přístup nelze považovat za nejlepší řešení. Jako druhé možné řešení je použití přirozených názvů jako identifikátorů při vytváření HTML kódu stránky. V tomto případě se vyžaduje větší spolupráce s vývojáři.

Pokud si zvolíme nekomplikovat práci testerům a vývojářům, můžeme zkombinovat BDD s klasickým přístupem tak, že necháme prostor pro technické názvy elementů. Je z tabulky 4.1 patrné, že kombinace implicitních vět i vlastního kódu, tedy testovací metoda *Behat + vlastní kód*, je nejefektivnější. Možnost vytvářet své vlastní věty zvyšuje úroveň abstrakce testů a dává větší volnost při vytváření testovací sady než v případě použití pouze implicitních vět. Další výhodou tohoto přístupu, která vyplývá z volnosti při tvorbě testovacích kroků, je schopnost vytvářet srozumitelné pro business analytiky věty, které mohou popisovat business hodnotu daného testu.

Použití pouze implicitních vět pro vytvoření testovací sady může být v některých případech hodně omezující. Týká se to hlavně úrovně abstrakce testů a formulování podmíněných kroků, například v případě odkazování se na elementy, které se objevují pouze za některých podmínek.

Volba PHP SC pro testování znamená především přenechávání vytvoření všech pomocných funkcí na testerech. To potom zvyšuje pravděpodobnost zanesení chyb při tvorbě testů. Na druhou stranu vytvoření vlastních pomocných funkcí může být mnohém efektivnější, než použití implicitních v případě Behatu, protože máme možnost tomu přidat větší režii, která může urychlit proces testování.

Když se na nástroje Behat a PHP SC podíváme ze stránky podporování funkcí Selenia z oficiálních verzí, např. pro jazyk *Python*, oba dva nástroje v některých případech selhávají. Např. u nástroje Behat (verze 2.4 stable) není možné přepínat mezi okny. PHP SC s tímto problém nemá. Komplikace ale nastávají při využití funkce návratu v historii

– `back()`. Nelze se odkázat na dříve načtené elementy ze stránky 1 po zpětném návratu. Bez ohledu na nezměněnou cestu k elementu PHP SC považuje tuto definici za neplatnou a vyžaduje znovu načtení elementů. Pomocí nástroje Behat to jde udělat bez problémů.

Z důvodu stálého vývoje obou frameworků lze narazit i na další situace, které se řeší obtížněji v Behatu na rozdíl od PHP SC i opačně. Proto záleží na cíli, který si klademe ještě před začátkem testování. Pokud pro nás není BDD nějakým způsobem atraktivní a business hodnota není pro nás až tak důležitá, je jednodušší pro testování využít PHP SC. V opačném případě se pustit do studování Behatu.

V případě firmy *Dixons Carphone* považuji jejich volbu BDD a nástroje Behat za vhodnou, hlavně z důvodu, že primárním zaměřením této společnosti je obchod a ne vývoj SW. Tedy spolupráce business analytiků, vývojářů a testerů je hodně důležitá.

Do budoucna plánuji zkoumat i další možnosti využití nástroje Behat. Chci se zaměřit na testování aplikací bez uživatelského grafického rozhraní pomocí tohoto nástroje.

Literatura

[1] Amman, P.; Offutt, J.: *Introduction to software testing*. Cambridge University Press, 2008, iSBN 978-0-521-88038-1.

[2] Baukes, M.: Operational Acceptance Testing. Dostupné z: `http://www.scriptrock.com/blog/operational-acceptance-testing`, [cit. 2014-09-23].

[3] Bleasdale, M.: *Request For Change 2433 Guest Checkout*. Dixons Carphone, Březen 2014.

[4] Bodnarchuk, M.: Introduction. Dostupné z: `http://codeception.com/docs/01-Introduction`, [cit. 2014-09-23].

[5] Dec, D.: Behaviour-Driven Development. Dostupné z: `http://www.future-processing.pl/blog/behaviour-driven-development/`, Říjen 2014, [cit. 2015-03-30].

[6] Keyes, J.: *Software engineering handbook*. Taylor and Francis e-Library, 2005, iSBN 0-8493-1479-8.

[7] Kolesnik, N.: JBehave vs Cucumber JVM comparison. Dostupné z: `http://mkolisnyk.blogspot.in/2013/03/jbehave-vs-cucumber-jvm-comparison.html`, Březen 2013, [cit. 2014-12-17].

[8] McKay, J.; Bath, G.: *Software test engineer's handbook: a study guide for the ISTQB test analyst advanced level certificates*. Rocky Nook Inc., 2008, iSBN 978-1-933952-24-6.

[9] Myers, G. J.; Sandler, C.; Badgett, T.: *The art of software testing*. John Wiley and Sons, 2012, iSBN 978-1-118-03196-4.

[10] Nearsoft Inc.: PHP-SeleniumClient V2 Documentation. Dostupné z: `http://nearsoft-php-seleniumclient.herokuapp.com/docs/v2/`, [cit. 2015-04-23].

[11] Nicklas, J.: You're Cuking It Wrong. *Elabs*, Srpen 2012.

[12] North, D.: INTRODUCING BDD. Dostupné z: `http://dannorth.net/introducing-bdd/`, 2006, [cit. 2014-09-18].

[13] WWW stránky: Behat documentation. Dostupné z: `http://docs.behat.org/en/v2.5/`, [cit. 2014-12-12].

[14] WWW stránky: Concordion documentation. Dostupné z: `http://concordion.org/`, [cit. 2014-12-17].

[15] WWW stránky: JBehave documentation. Dostupné z: `http://jbehave.org/introduction.html`, [cit. 2014-12-17].

[16] WWW stránky: SeleniumHQ Browser Automation. Dostupné z: http://www.seleniumhq.org/, [cit. 2015-04-20].

[17] WWW stránky: Writing Features. Gherkin syntax. Dostupné z: `http://docs.behat.org/en/latest/guides/1.gherkin.html`, [cit. 2014-12-21].

[18] Wynne, M.; Hellesoy, A.: *The Cucumber Book: Behavior-Driven Development for Testers and Developers*. Pragmatic Programmers, LLC, 2012, iSBN 978-1-934356-80-7.

Příloha A

Obsah CD

Adresářová struktura CD:

- **Behat** – složka s testy, napsanými pomocí nástroje Behat
  - `behat.yml` – soubor s informacemi o domovské stránce, prohlížeči a sezení
  - `bin` – složka s binárním souborem `behat`
  - `composer.json` – soubor obsahující informace o požadavcích na instalaci
  - `composer.lock` – soubor obsahující informace o nainstalovaných programech
  - `composer.phar` – soubor instalující programy, uvedené v composer.json
  - `features` – složka s příběhy a souborem se zdrojovým kódem `FeatureContext.php`
  - `vendor` – složka se zdrojovými kódy nástroje Behat
- **Selenium** – složka s testy, napsanými pomocí nástroje Selenium
  - `composer.json` – soubor obsahující informace o požadavcích na instalaci
  - `composer.lock` – soubor obsahující informace o nainstalovaných programech
  - `composer.phar` – soubor instalující programy, uvedené v composer.json
  - `screenshots` – složka pro snímky obrazovky
  - `tests` – složka s testy
  - `vendor` – složka se zdrojovými kódy nástroje Selenium + soubor s vlastními funkcemi `ChooseElements.php`
- **selenium-server-standalone-2.43.1.jar** – Selenium–standalone server

Příloha B

Návod na instalaci testovacích nástrojů a spuštění testovacích případů

B.1 Spuštění testů pomocí nástroje Behat

Prvním krokem je **instalace**. Instalace Behatu se provádí využitím nástroje *Composer*. Předem je nutné vytvořit soubor `composer.json`, který bude obsahovat všechny potřebné informace pro instalaci (pro více informací viz zdroj [13]). Je nutné, aby verze Behatu, Selenium–standalone serveru a prohlížeče Firefox byly kompatibilní. Verze, se kterými pracuji, jsou:

- **Behat** — 2.4.6, **Mink**[1] — 1.5.0, **Gherkin** — 2.2.9,
- **Selenium Server** — 2.43.1,
- **Firefox** — 33.1.1.

Spuštění:

1. Spustění Selenium serveru: `java -jar selenium-server-standalone-2.43.1.jar`.
2. Spuštění jednotlivých/všech testů z příkazového řádku: `bin/behat [[--tags=@...]<`*další volitelné funkce*`>]`.

Testy na stránkách obsahujících `common.fo-currys` v názvu url nelze spustit bez připojení k síti společnosti Dixons Carphone.

B.2 Spuštění testů pomocí nástroje Selenium

Instalace nástroje PHP Selenium Client se provádí pobodně jako instalace nástroje Behat. Prvním krokem je vytvoření souboru `composer.json`, kde musí být uvedeny informace o verzích jazyka PHP a nástroje PHP Selenium Client. Případně také informace o verzi PHPUnit, pokud žádnou nemáte nainstalovanou. Verze, se kterými pracuji, jsou:

[1] viz podkapitola 3.2.1

- **PHP** — 5.3.8,
- **PHPUnit** — 4.0.6,
- **Curl** — 7.37.1,
- **Nearsoft/PHPSelenium Client** — 2.0,
- **Selenium Server** — 2.43.1,
- **Firefox** — 33.1.1.

Spuštění:

1. Spustění Selenium serveru: `java -jar selenium-server-standalone-2.43.1.jar`.
2. Spuštění jednotlivých testů: `phpunit tests/`<*název souboru*>.

Testy na stránkách obsahujících `common.fo-currys` v názvu url nelze spustit bez připojení k síti společnosti Dixons Carphone.