Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra informačních technologií

Studijní program: Aplikovaná informatika

Obor: Informační systémy a technologie

Behaviour Driven Development a Scrum v korporátním prostředí

DIPLOMOVÁ PRÁCE

Student: Bc. Barbora Kulhánková

Vedoucí: doc. Ing. Alena Buchalcevová, Ph.D.

Oponent: Ing. Jan Kučera, Ph.D.

2016

Prohlášení

Prohlašuji, že jsem diplomovou práci zpracovala samostatně a že jsem uvedla všechny použité prameny a literaturu, ze které jsem čerpala.

V Praze dne 20. 4. 2016 .....

Barbora Kulhánková

Poděkování

Na tomto místě děkuji v první řadě paní doc. Ing. Aleně Buchalcevové, Ph.D., za cenné rady a ochotu při vedení diplomové práce. Dále chci poděkovat všem, kdo mě podporovali po dobu mého vysokoškolského studia, a to jak v profesním, tak v osobním životě.

Abstrakt

Agilní metodiky vývoje softwaru už nejsou doménou jen malých firem nebo startupů, ale dostávají se i do korporátního prostředí. Důkazem tomu může být i to, že Scrum je nejrozšířenějším přístupem k vývoji softwaru. Diplomová práce tak cílí na využití agilních metodik, konkrétně metodiky Scrum v korporátní sféře. Autorka představuje přístup Behaviour Driven Development a navrhuje, jak by mohl tento přístup pomoci řešit nedostatky nebo bariéry metodiky Scrum ve velkých podnicích. Hlavním výstupem práce je autorská metodika ScrumFlow, založená na metodice Scrum a rozšířená o postupy Behaviour Driven Development. Metodika ScrumFlow je autorkou publikována na web a je volně dostupná.

Klíčová slova

Agilní vývoj softwaru, agilní metodiky, Behaviour Driven Development, Scrum, SpecFlow, Gherkin

Abstract

Agile software development methodologies these days are not used only in small enterprises or startups, they are getting spread around large enterprises as well. An evidence of this fact could be, that Scrum is nowadays the most commonly used approach to software development. So this thesis focuses on usage of agile approaches, Scrum methodology in particular, in large enterprises. Author presents an approach called Behaviour Driven development and proposes how this approach could help dealing with imperfections or barriers of Scrum when used in large enterprises.The major outcome of this thesis is author's own methodology called ScrumFlow, based on Scrum methodologyy and extended by Behaviour Driven Development approaches. ScrumFlow methodology is published on a website and available for free.

Keywords

Agile software development, agile methodologies, Behaviour Driven Development, Scrum, SpecFlow, Gherkin

Obsah

1 Úvod

Agilní metodiky vývoje softwaru již dávno nejsou doménou tzv. startupů nebo malých inovativních projektů, například metodika Scrum je v současnosti nejpoužívanějším přístupem k vývoji softwaru. Od vydání agilního manifestu už uplynulo patnáct let, a agilní metodiky se staly velmi rozšířenými napříč odvětvími vývoje softwaru. Některé jsou rozšířené více, jiné méně, další se postupem času vyvíjejí, rozšiřují nebo jsou překonané jinými. Rozhodně však agilní vlna změnila pohled na přístup k vývoji softwaru napříč celou oblastí.

Zejména metodika Scrum je hojně využívána i v korporátní sféře. Scrum je využívána například v bankovním prostředí (nejednou velkou bankou) pro interní vývoj softwaru. Ve sféře velkých podniků a nadnárodních korporací se často jedná o velké projekty – i tam je už Scrum běžně nasazován. Důkazem výše uvedené evoluce agilních metodik je i existence přístupů jako Scrum in Scrum nebo LeSS, které si kladou za cíl adaptovat metodiku Scrum právě pro velké projekty, přestože původní agilní metodiky nejspíš nevznikaly přímo pro projekty o několika softwarových týmech. Podle průzkumů už je Scrum využíván i v distribuovaných týmech, což je doména právě nadnárodních korporací.

Většina softwarových vývojářů se díky masové rozšířenosti agilních metodik během posledních let s některým z agilních přístupů za svou praxi setkala. Vývojový tým jako celek tak může mít velmi dobré znalosti agilních přístupů. Jenže projektový tým se neskládá pouze z vývojářů a projekt není pouze projektový tým. A právě ostatní členové projektového týmu nebo spíše okolí projektového týmu v korporaci znalost agilních metodik zpravidla nemají. Důvod k tomu je celkem logický: vývojový tým může vyvíjet agilně uvnitř korporace, ale adaptovat celou korporaci na agilní přístupy je v praxi nejspíš nemožné, takže nutně dochází k určitým omezením. I kdyby celý projektový tým postupoval podle agilních metodik, vždy bude v rámci organizace narážet na svoje okolí nefungující agilně, se kterým nevyhnutelně musí komunikovat. Nutně tak dochází k mnohým rozporům. Pro příklad jeden z nich: agilní vývojář podle agilního manifestu upřednostňuje dodanou hodnotu zákazníkovi před rozsáhlou dokumentací. V tradičně řízené organizaci ale většinou existují nějaké procesní rámce nebo nařízení, která vyžadují určitý stupeň dokumentace. Zavedení agilní metodiky v korporátním prostředí tak neznamená nutně a automaticky zlepšení procesu vývoje softwaru právě kvůli

překážkám, které obě strany musí překonávat, a je nutné najít nějaký kompromis, který bude únosný pro obě všechny zúčastněné.

Jednou z dalších významných bariér mezi agilním projektem a jeho okolím v rámci tradičně řízené organizace může být i komunikace. Komunikace vývojového týmu a zákazníka bývá obecně problém, a pokud je k této komunikační bariéře přidán ještě výše popsaný nesoulad rozdílných přístupů k řízení vývoje nebo projektu, problém může být o to významnější. A zde to může vypadat, že agilní metodiky mírně selhávají. Většina agilních metodik totiž sice akcentuje komunikaci se stranou zákazníka (byznysu), avšak přímo neřeší komunikační bariéru mezi byznysem a vývojovým týmem. Jenom doporučením, že mají obě strany komunikovat nebo že má být zákazník více vtažen do vývoje, není vyřešen problém, který je na komunikaci dvou stran nejdůležitější: rozdílné smýšlení, rozdílný jazyk (a není zde myšlena čeština nebo angličtina, spíše myšlenkový proud), kterým obě strany mluví. Vývojář bude vždy vidět spíše technickou stránku věci, zákazník naopak jen svůj požadavek, který ale často není schopen popsat tak, aby ho vývojář pochopil. Vzniká tak frustrace na obou stranách, které se nemohou domluvit, nemohou najít společnou řeč. Agilním přístupem, který tuto bariéru přímo řeší, je Behaviour Driven Development. Pokud se tedy vezme část přístupu Behaviour Driven Development a vhodně vsadí do metodiky Scrum, vzniká metodika, která může pomoci týmům lépe komunikovat s byznysem a zvýšit transparentnost práce týmu pro byznys. Autorka tak rozebírá Behaviour Driven Development zejména z pohledu využití jako techniky pro zlepšení testování a specifikaci požadavků, a možnosti jejich integrace s metodikou Scrum, jakožto využívaným přístupem pro korporátní softwarové projekty.

1.1 Vymezení tématu a zdůvodnění jeho výběru

Z výše uvedeného vyplývá, že zavedení agilní metodiky v korporátním prostředí může být problematické, a je třeba řešit například komunikační bariéru mezi zúčastněnými stranami. V prostředí velkých podniků bývá nasazována metodika Scrum, což je procesní rámec řízení vývoje softwaru. Diplomová práce je zaměřena na Behaviour Driven Development a klade si za cíl rozšířit metodiku Scrum právě o techniky Behaviour Driven Development.

Autorka zde využívá několikaleté praxe v oblasti testování softwaru a práci zakládá na aktuální zkušenosti ze Scrum týmu vyvíjejícího interní bankovní aplikaci. Specifikace zde často bývají doplněny o příklady a akceptační kritéria ve strukturovaném formátu „GIVEN –

WHEN – THEN“. Nabízí se tak možnost automatizace scénářů těchto kritérií ve stylu Behaviour Driven Development. Tohoto potenciálu však vývojový tým doposud nevyužil, testování probíhá pomocí testovacích scénářů v nástroji HP Quality Center a automatizovaných testů v rámci aplikace. Mezi nimi však není jasný vztah, automatizované testy jsou nesrozumitelné pro byznys a naopak samotné testovací scénáře nejsou automatizované, což dělá testování časově náročné. Zde navrhované řešení podobných situací spočívá ve využití přístupu Behaviour Driven Development – využití automatizace scénářů jednotlivých požadavků psaných pomocí přirozeného jazyka. Tzn. uživatel je schopen definovat požadavky v pro něj přijatelné formě, která je zároveň jednoznačná a srozumitelná pro vývojový tým. Tím dochází ke dvěma benefitům: zpřehlednění výstupu testování pro stranu byznysu a možnost případného reportingu, a jasně definované požadavky, které lze přímo převést na automatizované testy.

1.2 Cíle práce

Hlavním cílem diplomové práce je rozšíření metodiky Scrum tak, aby byla lépe použitelná v korporátním prosředí. Konkrétní rozšíření metodiky Scrum využívá přístupu Behaviour Driven Development a přidává kromě jiného způsob definice požadavků a s nimi souvisejících scénářů ve formě, která je jednoduše čitelná i automatizovatelná. Tedy vhodná jak pro byznys, tak pro vývojový tým. Výsledkem je vytvoření společné komunikační platformy pro byznys a vývojový tým, a tím snížení komunikační bariéry mezi oběma stranami. Kromě zlepšení samotné komunikace navržené rozšíření přinese také zlepšení transparentnosti práce týmu, zjednodušení vytváření dokumentace a reportingu, nebo jednoznačné provázání automatizovaných testů s uživatelskými požadavky. Metodika Scrum je proto rozšířena o přístup Behaviour Driven Development - autorka navrhuje vlastní metodiku ScrumFlow, která je právě rozšířením metodiky Scrum o Behaviour Driven Development.

Mezi dílčí cíle dále patří:

- Provedení rešerše a představení přístupu Behaviour Driven Development,
- porovnání dostupných softwarových nástrojů pro Behaviour Driven Development
- představení konkrétního nástroje vhodného pro korporátní sféru, včetně práce s ním, a ukázek implementace na několika scénářích,

- vytvoření základní příručky pro práci s nástrojem SpecFlow,

- implementace navržené metodické podpory ScrumFlow v nástroji Eclipse Process Framework Composer. (Protože je metodika ScrumFlow kompletně v češtině, zahrnuje i překlad metodiky Scrum do češtiny.)

1.3 Způsoby dosažení cílů

K dosažení cílů jsou nejprve zmapovány existující publikace o hlavním tématu, Behaviour Driven Development, které jsou poté utříděny do komentované rešerše. Rešerše je provedena i na téma metodiky Scrum. Poté jsou vymezena specifika korporátního prostředí. Na základě těchto zdrojů je charakterizována metodika Scrum s důrazem na korporátní prostředí. Dále je představen přístup Behaviour Driven Development – opět s ohledem na jeho možnosti v korporátním prostředí. Dále jsou na základě dostupné dokumentace popsány softwarové nástroje pro Behaviour Driven Development, které jsou poté porovnány. Za pomoci dostupné dokumentace je ve vybraném nástroji provedena praktická ukázka, která na jednoduchých příkladech demonstruje jeho použití. Celý postup vytváření praktických příkladů je dokumentován, což je základem pro českou uživatelskou příručku ke SpecFlow. Syntézou získaných informací o metodice Scrum, jejích omezeních, a přístupu Behaviour Driven Development je poté navrženo rozšíření metodiky Scrum, čímž vzniká nová metodika, ScrumFlow. Metodika ScrumFlow je dále implementována pomocí nástroje Eclipse Process Framework Composer a publikována na webu, čímž je zajištěna nejen její samotná publikace, ale i možnost použití veřejností a dalšího rozvoje.

1.4 Předpoklady a omezení práce

Jak vyplývá z úvodu, motivem autorky je vlastní zkušenost z práce v konkrétním Scrum týmu v korporátním prostředí. Práce je psaná v obecné rovině, není *šita na míru* konkrétnímu týmu, ale snaží se vystihnout specifika velké organizace. Vybrané nástroje sice tým buď už používá (Microsoft Visual Studio) nebo by byl přechod na ně vzhledem k ostatním jednoduchý (SpecFlow), důležitější aspekt jejich výběru je ale fakt, že se jedná o pokročilé nástroje, které jsou v korporátní sféře využívány. A právě na korporátní prostředí práce cílí.

Praktická ukázka automatizace pomocí nástroje SpecFlow je pro zjednodušení provedena jako test front-endu webové aplikace pomocí nástroje Selenium Webdriver. Jak je ale v dané kapitole zdůrazněno, ukázky jsou pouze ilustrační a nástroj SpecFlow tak jde použít pro testování libovolné vrstvy aplikace, nejen webového rozhraní. Analogicky, práce s ostatními nástroji je velmi podobná, Behaviour Driven Development se tedy neomezuje pouze na práci v jazyce C#, ostatní dále popsané nástroje se používají i v jiných programovacích jazycích.

1.5 Struktura práce

V úvodu práce autorka definuje specifika IT ve velkém podniku, na kterých dále staví. Poté představuje metodiku Scrum obecně, ale i z hlediska testování softwawru. Poukazuje na nedostatky, respektive oblasti, které Scrum jako procesní rámec neřeší zejména v kontextu využití metodiky Scrum v korporátním prostředí, a zkoumá možnosti rozšíření metodiky Scrum.

V další části je představen přístup Behaviour Driven Development, kde je popsáno, jak by mohl pomoci řešit problémy vzniklé použitím metodiky Scrum, eliminovat její nedostatky, vykrýt bílá místa. Poté se autorka zaměřuje na techniky BDD v oblasti specifikace, testování a dokumentace, a také softwarové nástroje pro Behaviour Driven Development. Ty jsou představeny, setříděny, porovnány, a jeden z nástrojů je následně vybrán k detailnější analýze.

Na vybraném nástroji, SpecFlow autorka demonstruje použití jazyka Gherkin pro definici požadavků, ale i samotnou implementaci automatizovaných scénářů požadavků.

Poslední část práce je věnovaná návrhu metodiky ScrumFlow, která je rozšířením metodiky Scrum o Behaviour Driven Development. Metodika je popsána, daná rozšíření jsou zdokumentována, a celá metodika je publikována online jako balíček nástroje Eclipse Process Framework Composer.

1.6 Výstupy a očekávané přínosy

Hlavním přínosem diplomové práce je návrh metodiky ScrumFlow, respektive rozšíření metodiky Scrum o techniku Behaviour Driven Development, což může v praxi pomoci překonávat komunikační bariéru mezi byznysem a členy vývojového týmu. Souvisejícím přínosem je implementace a následná online publikace navržené metodiky ScrumFlow

v nástroji Eclipse Process Framework Composer. Publikací metodiky je zajištěno její možné nasazení v praxi, případně její další rozvoj.

V oblasti samotné automatizace je přínosem představení praktických příkladů spolu s českou uživatelskou příručkou k nástroji SpecFlow, která doposud neexistuje, a může tak ještě pomoci přenést uvedené testovací techniky do praxe, stejně jako přehledné srovnání dostupných nástrojů pro jednotlivé programovací jazyky.

1.7 Základní pojmy

Vodopádový model vývoje softwaru

Vodopádový model je nejstarším modelem životního cyklu vývoje softwaru, který vznikl již v 70. letech. Přinesl rozdělení procesu vývoje softwaru na jednotlivé fáze od analýzy po zavedení a údržbu, a pomohl vývoj systematizovat. Podle vodopádového modelu je možné k další fázi přejít vždy pouze po dokončení fáze předchozí, vrátit se je možné pouze o jednu fázi zpět. V praxi vodopádový model naráží na to, že často není možné specifikovat požadavky na začátku projektu, nebo pozdní integraci celého systému a s ní související nutné opravy v pozdní fázi. (Buchalcevová, 2009)

Testovací případ

Soubor vstupních hodnot, exekučních předpokladů, předpokládaných výsledů a důsledků exekuce vytvořených ke konkrétnímu cíli nebo podmínce, jako je zkouška patřičné cesty programem nebo k ověření souladu se specifickým požadavkem. (International Software Testing Qualifications Board, 2015 , s. 62) Testovací případ je podkladem pro testovací scénář.

Testovací scénář

Dokument specifikující sled akcí pro exekuci testu. Synonymy jsou *testovací procedura* nebo *testovací skript*. (International Software Testing Qualifications Board, 2015 , s. 69) Testovací scénář se zpravidla skládá z jednotlivých testovacích případů.

Akceptační kritéria

Výstupní kritéria, která komponenta nebo systém musí splňovat k tomu, aby prošel akceptací uživatelem, zákazníkem nebo jinou pověřenou entitou (International Software Testing Qualifications Board, 2015 , s. 1).

Akceptační testování

Formální testování s ohledem na uživatelovy potřeby, požadavky a byznys procesy, jehož cílem je rozhodnout, jestli systém splňuje akceptační kritéria. Umožňuje uživateli, zákazníkovi, případně jiné autoritě rozhodnout, jestli bude dodaný systém akceptován nebo ne. (International Software Testing Qualifications Board, 2015 , s. 1)

Behaviour Driven Development (BDD)

Vývoj řízený chováním, metodika vyvinutá ze zaběhnutých agilních principů. Zahrnuje spolu s vývojem i oblast agilní analýzy a automatizovaných akceptačních testů, které zároveň slouží jako dokumentace produktu. (North, 2006).

Scrum

Agilní framework pro komplexní projekty. Původně byl vytvořen pro softwarové projekty, lze ho ale využít pro jakýkoliv komplexní inovativní projekt i mimo softwarové odvětví. (Scrum Alliance, 2016)

Gherkin

Přirozený jazyk (existuje implementace pro více než 60 světových jazyků včetně Češtiny) s přidanou strukturou. Stále je snadno srozumitelný ne-programátory, ale natolik strukturovaný, aby umožnil výstižný popis příkladů vedoucích k vyjasnění byznys pravidel většiny oblastí lidské činnosti. (Cucumber Ltd., nedatováno)

Product Backlog Item (PBI)

Česky položka produktového backlogu, je podle metodiky Scrum jednotka práce, která je dostatečně malá, aby mohla být dokončena v rámci jednoho sprintu (iterace vývoje). Položky se dále dělí na jednotlivé úkoly. (Scrum Alliance, 2007)

User Story

High level, nebo také *všeobecný*, byznys požadavek, typicky popsaný jednou nebo více větami běžného byznys jazyka. Zachycuje, jakou funkcionalitu uživatel potřebuje a její akceptační kritéria. Typicky je *user story* využívaná v prostředí agilního vývoje. (International Software Testing Qualifications Board, 2015 , s. 74)

Velký podnik

Velký podnik je podnik o více než 250 zaměstnancích, jehož aktiva/majetek nepřesahují korunový ekvivalent částky 43 mil. EUR nebo má obrat/příjmy nepřesahující korunový ekvivalent 50 mil. EUR (Cardová, 2006).

Korporace

V kontextu této práce je pojmem *korporace* myšlena nadnárodní korporace. Nadnárodní korporace je podnik, firma se sídlem v jedné zemi, která svoji činnost provozuje ve více zemích nebo státech prostřednictvím zahraničních, vlastněných nebo kontrolovaných dceřiných společností (ManagementMania.com, 2015). Nadnárodní korporace zpravidla splňuje výše uvedené charakteristiky velkého podniku, pro potřeby této práce je nadnárodní korporací myšlena korporace spadající do kategorie velkých podniků.

2 Rešerše prací

Rešerše prací je rozdělena na dvě podkapitoly podle hlavních okruhů, kterým se práce věnuje. Obsáhlejší z nich je oblast Behaviour Driven Development, která je hlavním tématem práce. Z toho důvodu je do rešerše zahrnuta veškerá nalezená literatura a další doplňující zdroje (např. multimediální). Ve druhé podkapitole jsou zmapovány zdroje k metodice Scrum. Avšak vzhledem k četnosti existujících zdrojů a faktu, že Scrum není hlavním předmětem rešerše, slouží podkapitola spíše k orientaci v použitých zdrojích, vybírá zdroje přímo relevantní v kontextu práce. Pro získání dostatku zdrojů, na základě kterých lze získat povědomí o metodice, však bohatě postačuje.

2.1 Behaviour Driven Development

První cesta v literatuře, pokud jde o Behaviour Driven Development, míří k samotmému autorovi metodiky, Danu Northovi a jeho článkům o BDD – například prvnímu, *Introducing BDD* (North, 2006), nebo jeho prezentaci spolu s Liz Keogh, (North a Keogh, 2009) která je také významnou spoluautorkou v oblasti BDD. Na samotný Behaviour Driven Development svou prací v akademických článcích navazují další autoři, například Carrera et al. (2014), kteří staví na BDD další metodiku, BEAST, dále třeba Diepenbeck et al. (2012) přenáší BDD i do oblasti hardwaru. Co se týká využití přirozeného jazyka, jde ve své práci výrazně dál Soeken et al. (2012), který nevyužívá strukturovanou syntaxi jazyka Gherkin jako ostatní, ale snaží se určitým způsobem zpracovávat přímo přirozený jazyk v jeho běžné syntaxi. Tyto navazující publikace jsou už ale mimo rámec práce.

Pro oblast Behaviour Driven Development je větší množství informací publikováno v knihách. Velmi významnou roli zde hraje kniha BDD In Action: Behavior Driven Development for the Whole Software Lifecycle (Smart, 2014), která popisuje velmi komplexně přístup BDD. Není tedy zaměřena pouze na konkrétní nástroje, více než ostatní knihy v dané oblasti se věnuje myšlence a filosofii BDD a nabízí i jiné úhly pohledu než ty technické, ze strany vývoje. Významnou roli dále hraje edice The Pragmatic Bookshelf, ve které se v oblasti BDD autorka opírá o tři důležité zdroje: Cucumber Recipes (Ian Dees, 2013), The Cucumber Book (Wynne M., 2012) and The RSpec Book (Chelimnsky, 2010). Poslední jmenovaná publikace je sice zaměřená na jazyk Ruby, základní principy jsou ale

obecné, a čerpat se z ní tedy dá nezávisle na programovacím jazyce. Totéž se dá říci o knize The Cucumber Book, avšak s tím, že je kladen větší důraz na principy BDD a jejich obecné vysvětlení, než na samotnou syntaxi programovacího jazyka. Ještě komplexnější pohled nabízí kniha Cucumber Recipes, která kromě představení obecných principů uvádí praktické příklady pro různé nástroje a programovací jazyky. Další významnou knihou, ze které autorka čerpá, je Specification by Example (Adzic, 2011), jejíž silnou stránkou je představení určitých *best practices* v oblasti automatizovatelných specifikací.

V českých publikacích lze najít pouze dva prameny: bakalářská práce kolegy Vodičky (2012) z VŠE, která je zaměřena obecně na BDD a jeho kontext ve vývoji softwaru, a diplomová práce kolegy Nečase (2011) z Masarykovy univerzity, která sice zasahuje do oblasti testování webových aplikací; na rozdíl od této práce je ale zaměřena jako konkrétní projekt na jedné aplikaci.

Kromě běžných zdrojů jako knihy a online články jsou doplňujícím zdrojem k získání povědomí o problematice BDD online dokumentace k jednotlivým nástrojům BDD, například SpecFlow (TechTalk, nedatováno). Dále se v oblasti BDD velmi dobře čerpá z online kurzů na portálu PluralSight (PluralSight, nedatováno).

2.2 Scrum

Metodika Scrum je populárním a rozšířeným přístupem, a tak není překvapující množství publikací zahrnující metodiku Scrum. Výchozím bodem k získání základního povědomí a orientaci v metodice Scrum může být například článek *What is Scrum?* Spolu s krátkým videem demonstrujícím metodiku Scrum od Scrum Alliance (nedatováno). Dalším zdrojem už může být přímo Průvodce Scrumem (Schwaber a Sutherland, 2013) od samotných autorů metodiky. Jeho velkým přínosem je český překlad, tato publikace je tedy místem, kde jsou zavedeny české ekvivalenty terminologie metodiky Scrum, alespoň pro část pojmů. Vhodným doplňkem při hledání základních principů metodiky Scrum je například *Scrum: a Breathtakingly Brief and Agile Introduction This overview of roles , artifacts and the sprint The Elements of Scrum* (Sims a Johnson, 2012). Obě publikace nabízejí čistě přehled metodiky a jejích součástí. Vhodným doplňkem jsou tak další publikace zachycující Scrum v praxi, například *A Scrum Adoption Survey* (Paulk, 2013), dále kniha *Agile Project Management with Scrum* od Kena Schwabera (2004) nebo kniha *Suceeding with Agile:*

Software Development Using Scrum (Cohn, 2010). Ještě praktičtějším směrem pak jde kniha *Professional Scrum with Team Foundation Server 2010* (Resnick et al., 2011), která je zaměřena přímo na aplikaci metodiky Scrum za pomoci nástroje Microsoft Team Foundation Server. Z perspektivy testování na Scrum nahlíží Tilo Linz ve své knize Testing in Scrum (2014).

Akademických prací zahrnující metodiku Scrum je napsáno snad nespočet. Pokud zůstaneme na půdě VŠE, například kolega Košata (2010) ve své bakalářské práci popisuje metodiku Scrum. Práce tak může sloužit jako další pomocný materiál pro nasazení metodiky Scrum,

Významná je dále například disertační práce kolegy Tománka (2010), který navrhuje metodiku kombinující řízejí projektů pomocí PRICE2 a Scrum, nebo disertační práce kolegy Balady (2015), který posuzuje metodiku Scrum v komplexním prostředí a rozšiřuje ji tak, aby odpovídala právě komplexnímu prostředí a obsahovala například fázi globální analýzy a návrhu, kterou jinak agilní metodiky *neznají*.

3 Specifika velkého podniku

Jak bylo zmíněno v úvodu, za *velký podnik* je považován podnik s více než 250 zaměstnanci. Zajímavé je Wendtovo (2011) rozdělení z pohledu IT. Podle něj je organizace považována za *velkou* ve chvíli, kdy dorostla do stádia, ve kterém potřebuje samostatné, plně vytížené IT oddělení odborníků se specifickými znalostmi. Ti spravují specifické aplikace nebo infrastrukturu. To znamená, že velký podnik má zaměstnance s tituly jako je databázový administrátor nebo administrátor pro správu dat, a zpravidla uchovává velké objemy dat – stovky terabajtů a více. Nutně ale nemusí mít tisíce zaměstnanců.

Wendtova definice je tím pádem v souladu s definicí o počtu zaměstnanců. Ve velkém podniku jsou zpravidla týmy dedikované na jednotlivé projekty na různých úrovních. Pokud podnik vyvíjí vlastní software, bývají tyto týmy specializované na vývoj, infrastrukturu, administrativu, atd. Například podle průzkumu (Al-allaf, 2010) má velký podnik zpravidla více než 60 softwarových vývojářů. Což v praxi znamená více než jeden vývojový tým.

Velký podnik má, kromě velkého počtu vývojářů a rozsáhlého IT oddělení obecně ještě několik dalších specifik. Jsou jimi například právní regulace týkající se procesu vývoje softwaru. V praxi to jednoduše znamená, že musí mít formalizované procesy v IT. (Crookshanks, 2015) Jestliže v malém podniku není nutné mít na vše formální dokumentaci, ve velkém podniku to nutné je.

Velké podniky také zpravidla mají oddělená prostředí a vrstvy, kam mají přístup pouze specifické osoby nebo týmy, aby firma co nejvíce eliminovala různá pracovní rizika. Například změna v databázi nelze provést ručně; je nutné zaslat SQL skript databázovému administrátorovi, aby změnu provedl. (Crookshanks, 2015)

Dalším specifikem je podle Crookshankse (2015) to, že ve velkém podniku bývá většina vývoje směřována *dovnitř* firmy. To znamená, že firma interně vyvíjí software pro vlastní potřebu, a zákazníkem, nebo také uživatelem výsledného produktu, je tak zpravidla jiný zaměstnanec firmy, ne jako u jiných organizací, kdy je firmou vyvíjen software pro jinou firmu nebo zákazníka. Proto, mluvíme-li o zákazníkovi a byznysu, jedná se ve velké organizaci zpravidla o stejnou stranu. Interní aplikace slouží k podpoře byznysu, takže byznys uživatel je zákazníkem vývojového týmu a koncový uživatel aplikace. Uživatelů většinou

bývají menší objemy než u jiných typů aplikací, není to ale pravidlem. Zajímavější je jiný jev: často si koncový uživatel přímo může definovat své požadavky.

Kromě interního zákazníka má takováto firma také většinou veškerou infrastrukturu uvnitř firmy. Odpadá tak možnost cloudových škálovatelných řešení vně firmy – často jsou spojeny i obavy o legislativní aspekty nebo konkurenceschopnost firmy – jak bylo výše uvedeno, velká firma má mnoho stovek terabajtů dat, z kterých část z nich jsou citlivé údaje o klientech nebo data zajišťující konkurenční výhodu, jejchž případný únik by znamenal třeba i likvidaci firmy.

Celkem by se tedy specifika velké organizace z pohledu IT dala shrnout následovně:

- Specializované IT oddělení
- Více vývojových týmů
- Právní nároky a snaha o eliminaci rizik -> formalizovaný a dokumentovaný proces vývoje
- Zákazník je zpravidla zaměstnanec firmy, ne externí osoba
- Veškerá infrastruktura uvnitř firmy

Z uvedených charakteristik je tak zřejmé, že IT ve velkém podniku – ať už se jedná o správu nebo vývoj nových aplikací, je vždy *těžší*, rigoróznější, svázáno více pravidly vyplývajícími ať už z interních nebo externích nařízení.

Není proto automaticky možné nasazovat agilní, lehké metodiky, které byly původně vytvořeny pro malé projekty agilních organizací, a očekávat, že budou přirozeně fungovat. Vždy je nutná určitá adaptace – ze strany organizace a ze strany metodiky, nejčastěji nespíš, musí ustoupit obě strany, ale *ohnout* metodiku je snadnější, než *ohnout* například právní rámce.

4 Metodika Scrum

Přestože patří do rodiny agilních metodik, které jsou záležitostí hlavně 21. století, počátky metodiky Scrum se datují už do 90. let 20. století (Suscheck, 2014). Metodika Scrum je v posledních letech rozšířeným přístupem k vývoji softwaru - v současnosti je Scrum nejpoužívanější agilní metodikou - podle průzkumu Forrester Inc. (Giudice, 2014) zabírá Scrum první místo mezi používanými přístupy k vývoji softwaru napříč organizacemi. Podle téhož průzkumu je metodika Scrum výrazně rozšířenější než tradiční, dříve používaný, vodopádový vývoj. Z toho vyplývá důležitý fakt: Scrum, a tím pádem agilní metodiky již nejsou doménou pouze malých podniků nebo startupů, ale zaujímají významnou část na poli vývoje softwaru.

4.1 Základní principy a charakteristika metodiky Scrum

Podle Průvodce Scrumem (Schwaber a Sutherland, 2013) je metodika charakterizována následovně:

„Nástroj, s jehož pomocí mohou lidí zvládnout složité adaptivní problémy a při tom dodávat produkty s vysokou přidanou hodnotou. Scrum je:

- jednoduchý
- srozumitelný
- extrémně obtížný pro dokonalé zvládnutí.“

Scrum je nástroj, procesní framework, který může být použit k tvorbě komplexních produktů. Nespecifikuje konkrétní praktiky. Scrum zajišťuje zpětnou vazbu, díky Scrumu je jeho uživateli umožněno lépe identifikovat, co není řešeno optimálně, a může tak v procesu dělat změny vedoucí právě k optimalizaci. (Mark, 2013)

Scrum staví na třech základních principech. Těmi jsou:

- Transparentnost,
- Kontrola,
- Adaptace.

Transparentnost je stav, kdy jsou důležité aspekty procesu viditelné všem, kteří mají zájem na jeho výsledku. Zpravidla je k tomu třeba definovat nějaký standard, například to, že si

všichni pracovníci jasně definují stav, kdy je daný úkol hotov (Schwaber a Sutherland 2013, s. 3). **Kontrola** je nástroj k co nejčasnějšímu odhalení případných odchylek v procesu. Kontrolují se přímo samotné artefakty vzniklé procesem (Schwaber a Sutherland 2013, s. 3). **Adaptace** je nutný důsledek, jestliže byly při kontrole zjištěny výrazné odchylky. Kontrola a adaptace jsou zajištěny sérií činností, které metodika definuje, a jsou popsány dále (Schwaber a Sutherland 2013, s. 4).

4.2 Role

Většina metodik definuje nějaké role, výrazně se však liší v tom, do jaké úrovně detailu jsou tyto role popsány. Například podle Becka (2005) metodika Extreme Programming definuje celkem sedm rolí, kde každá z nich má přesně dané svoje činnosti a odpovědnosti.

Metodika Scrum definuje jen několik základních rolí, kterým ale nedefinuje přesné činnosti. Definice rolí se v metodice Scrum zaměřuje spíše na výsledek než způsob, jak ho bylo dosaženo. Metodika Scrum definuje tyto role:

- Scrum Master
- Vývojový tým (Development Team)
- Vlastník produktu (Product Owner)
- Scrum tým (Scrum Team)

V následujících podkapitolách jsou jednotlivé role stručně charakterizovány.

4.2.1 Scrum Master

Scrum master je jakýsi kouč Scrum týmu. Neřídí vývoj, neříká členům týmu, jak mají implementovat jednotlivé funkcionality. Jestliže výstupem práce vývojového týmu je výsledný produkt, výstupem Scrum mastera je skvěle funkční, sebe-organizující tým. Scrum master je hlídač a opatrovník týmu, který pomáhá týmu naučit se a aplikovat agilní praktiky tak, aby tým mohl co nejlépe fungovat. Je týmu k dispozici, aby pomohl odstranit jakékoliv překážky bránící mu v jeho práci. Je ale stále důležité si uvědomovat, že Scrum master však není šéf týmu, je na stejné úrovni, jako ostatní členové týmu, pouze má jiné znalosti a zodpovědnost. (Sims a Johnson, 2012)

4.2.2 Vývojový tým (Development Team)

Vývojový tým je sebe-organizující tým, který má jako celek mít všechny schopnosti potřebné k dodání produktu. Úkolem každého ze členů týmu je pomoci týmu dodat použitelný produkt v každém ze sprintů. Což někdy znamená podílet se na práci v oblasti své specializace, jindy ale členové týmu potřebují pracovat i mimo oblast své specializace, protože nejdůležitější je posouvat jednotlivé položky ze skupiny „probíhající“ do skupiny „hotovo“. Každý jednotlivý člen se musí naučit ne „dělat svou práci“, ale „dělat práci“, přestat myslet ve smyslu, co je jeho kompetence, a co už není. Je důležité se příliš nezaměřovat na to, co tým právě vyvíjí, ale co je hotovo, co lze dodat zákazníkovi. Protože to jediné je ve výsledku důležité. Tým jako celek je zodpovědný za implementaci požadavků (včetně toho, *jak* jsou implementovány, a odhadů doby implementace, tým je *vlastníkem* svých odhadů). (Sims a Johnson, 2012)

4.2.3 Vlastník produktu (Product Owner)

Vlastník produktu reprezentuje stranu zákazníka a hájí jeho zájmy. Má hlavní zodpovědnost za návratnost investic vložených do vývoje. Čehož dosahuje tím, že má kontrolu nad produktovým backlogem, a také tím, že se pravidelně ujišťuje, že tým plně rozumí požadavkům zákazníka. Snaží se tak zabránit dodání nepoužitelného produktu. Vlastník produktu je osoba, která rozhoduje o prioritách, která položka bude vyvíjena, a která ne. Vlastník produktu je také jediná osoba, která může vývojovému týmu říci, na čem má v danou chvíli pracovat, co je priorita. On, a nikdo jiný. (Sims a Johnson, 2012)

4.2.4 Scrum tým

Scrum tým je souhrnná role, zahrnující Scrum mastera, Vlastníka produktu a Vývojový tým. Cílem Scrum týmu je být samostatnou, sebe-organizující, multifunkční jednotkou, která není blokována čekáním na vstup někoho z vně týmu. (Schwaber a Sutherland, 2013)

4.3 Činnosti

Scrum definuje několik činností, žádná z nich však neříká přímo, co nebo jak se má dělat – ve vztahu k samotnému vývoji softwaru. Činnosti definované Scrumem zasahují spíše oblast procesní. Hlavní činností metodiky Scrum je **Sprint.**

Sprint je hlavní činností metodiky Scrum protože právě ve sprintu tým vytváří novou přidanou hodnotu produktu. Během sprintu tým nesmí být rušen dodatečnými požadavky, je

třeba zajistit, aby se tým mohl koncentrovat na stanovený cíl sprintu. Scrum připouští iteraci dlouhou až třicet dní, což je déle než u jiných iterativních metodik. Důvodem je, že přírůstek vytvořený během sprintu musí být, dle metodiky Scrum, funkční, a dodatelný zákazníkovi. (Scrum Alliance, 2007)

Další skupinou činností metodiky Scrum jsou přesně definované schůzky:

- Denní schůzka (Daily Scrum)
- Vyhodnocení sprintu (Sprint Review)
- Retrospektiva sprintu (Sprint Retrospective)
- Plánovací schůzky (Sprint Planning)

4.3.1 Denní schůzka (Daily Scrum)

Denní schůzka se koná každý pracovní den a je přesně časově ohraničená na 15 minut. Účastní se jí Vývojový tým, a cílem je především synchronizace aktivit jednotlivých členů týmu. Ve zkratce každý odpovídá na tři otázky:

→ Co jsem včera udělal pro to, abych pomohl týmu splnit cíl sprintu?

→ Co budu dělat dnes pro to, abych týmu pomohl splnit cíl sprintu?

→ Vidím nějaké překážky, které brání mně nebo vývojovému týmu ve splnění cíle sprintu?

Díky odpovědím na tyto otázky se vzájemně zlepšuje znalost každého člena týmu o projektu, a eliminuje nutnost dalších schůzek (Schwaber a Sutherland, 2013). Z denní schůzky může vyplynout například přerozdělení úkolů, pomoc juniorním kolegům, apod.

4.3.2 Vyhodnocení sprintu (Sprint Review)

Vyhodnocení sprintu se koná na konci každého sprintu. Zkoumá přírůstek, který během schůzky vyhodnocují samotní členové týmu a zainteresované strany (stakeholders). Dochází k vysvětlení ze strany týmu, které položky sprint backlogu byly dokončeny, které ne, a proč. Na základě těchto informací se poté upravuje produktový backlog. (Schwaber a Sutherland, 2013)

4.3.3 Retrospektiva sprintu (Sprint Retrospective)

Retrospektiva sprintu probíhá také na konci každého sprintu, po vyhodnocení. Cílem retrospektivy je analyzovat právě proběhlý sprint a identifikovat aspekty, které v daném sprintu fungovaly dobře, a které je případně možné zlepšit. Pokud jsou identifikována možná zlepšení, je vytvořen jejich plán, který se zavádí hned v příštím sprintu.(Schwaber a Sutherland, 2013)

4.3.4 Plánovací schůzka (Sprint Planning)

Plánovací schůzka probíhá také jednou za sprint, na jeho úplném konci, tj. po retrospektivě, aby bylo možné plánovat již s ohledem na změny, které z retrospektivy vyplynuly. Celý tým se zde účastní vytváření plánu příštího sprintu. Tým odhaduje, které položky produktového backlogu je schopen během příštího sprintu dodat, a následně stanovuje cíl sprintu. (Schwaber a Sutherland, 2013)

4.4 Artefakty

Metodika Scrum definuje několik artefaktů. Jsou jimi produktový backlog (*Product backlog*), backlog sprintu (*Sprint backlog*), burndown chart a úkolová tabule (*Task board*). Ty jsou níže charakterizovány.

4.4.1 Produktový backlog

Produktový backlog je seznam všech požadavků na výsledný produkt. Backlog zahrnuje veškeré položky, ať už funkcionality, opravy defektů, změny dokumentace, zkrátka všechno, co má smysl a hodnotu pro výsledný produkt. Všechny tyto položky se souhrnně nazývají *položky produktového backlogu*, v originálním znění *product backlog items*, zažitá je zkratka PBI. Místo pojmu položka produktového backlogu je však některými týmy preferován pojem „User Story“, který by se dal do češtiny přeložit jako „*příběh uživatele*“, lépe *„příběh užití“*, zkrátka funkcionalita popsaná z pohledu uživatele, reflektující přidanou hodnotu produktu pro něj. Použití pojmu User Story má nespornou výhodu v tom, že neustále připomíná, že tým vyvíjí produkt uspokojující potřeby uživatele (Sims a Johnson, 2012). Jednotlivé položky produktového backlogu jsou řazeny podle priority, za tuto prioritizaci a celkově obsah a dostupnost backlogu je zodpovědný vlastník produktu. Což je neustálá práce, protože obsah produktového backlogu se neustále mění, jde o živý dokument, který zachycuje všechny

zamýšlené změny produktu. Dokud existuje produkt, existuje i jeho produktový backlog. (Schwaber a Sutherland, 2013)

4.4.2 Backlog sprintu

Backlog sprintu je část produktového backlogu. Backlog sprintu je reprezentován dvěma soubory úkolů. Těmi, které představují cíle sprintu, a dalšími vybranými položkami produktového backlogu (Scrum Alliance, 2007). V rámci sprintu se v backlogu postupuje podle priorit, nejdříve jsou plněny úkoly reprezentující cíl sprintu.

4.4.3 Burndown chart

Burndown chart by se dalo do češtiny přeložit jako *schéma (graf) hoření.* Ono hoření je v terminologii Scrum trend práce v průběhu sprintu, releasu nebo produktu jako celku. Zdrojem dat je tak, v případě sprintu, backlog sprintu. Zbývající, nedokončená práce je zanesena na vertikální ose, časové úseky na horizontální ose. (Schwaber, 2004 , s. 130) Cílem je dostat se na konci časového úseku na nulovou hodnotu zbývající práce. Burndown chart může týmu poskytovat přehled o tom, jak si stojí, ale vždy je třeba na něj nahlížet pouze jako orientační, doplňkový nástroj, jinak hrozí, že prioritou týmu se stane „kreslit“ perfektní burndown chart, a nesoustředí se tolik na cíl sprintu.

4.4.4 Úkolová tabule

Takzvaný „Task Board“ neboli tabule s úkoly (někdy také nazývána *Scrum Board*) může mít podobu fyzické tabule s úkoly v podobě papírových lístků nebo ji může reprezentovat softwarový nástroj. Cílem tabule je mít přehled o tom, které položky (úkoly) jsou v které fázi práce. Nejjednodušší tabule má tři sloupce – první sloupec reprezentuje úkoly, na kterých ještě nikdo z týmu nezačal pracovat. Druhý sloupec znázorňuje úkoly, na kterých aktuálně někdo z týmu pracuje. A třetí sloupec znázorňuje úkoly, které jsou dokončeny, a splňují předem stanovenou definici, co je *hotovo.* Jednotlivé úkoly se tak přesouvají po tabuli, a zvyšují celkové povědomí o práci a transparentnost nejen v rámci týmu, ale i z pohledu jiných zainteresovaných osob (Sims a Johnson, 2012).

4.5 Scrum v korporátním prostředí

Podle Šochové a Kunce (2014, s 119) je Scrum dnes již obvyklou metodikou i ve velkých korporacích, jako jsou banky, mobilní operátoři nebo výrobní podniky. Scrum tedy lze

používat i ve velkých podnicích, rozdíl mezi zavedením metodiky Scrum v malém a velkém podniku je hlavně náročnost nasazení: ve velké firmě může být zapotřebí celý tým lidí, které nasazení připraví, a může jít o dlouhodobý proces.

Jak uvádí například Schwaber (2007), znalost agilního vývoje byla pro vývojáře výhodou ve 20. století. Dnes, v 21. století se agilní znalost agilního přístupu nepovažuje za výhodu, ale za samozřejmost. Alespoň u vývojářů. Problémem může být fakt, že celé korporace se nemohou adaptovat na agilní postupy, přestože vyvíjejí agilně. Projektoví manažeři se minulých dvacet pět let učili o vodopádovém modelu, který je dokonale popsaný většinou procesních příruček, a je označován za správný postup. Návrh použít agilní potup tak většinou vzbuzuje spíše nedůvěru a řadu „Ano, ale…“ (Schwaber, 2007). Ale průzkum (Giudice, 2014) ukazuje, že je i přes to všechno Scrum velmi populárním přístupem.

Problémem nasazení metodiky Scrum ve velké korporaci tak může být především v okolí projektu: jsou zavedeny určité standardy, vyšší management má určité požadavky, které projektový tým musí dodržovat nehledě na to, že pracuje podle agilní metodiky. Čistě agilní přístup může být v rozporu s firemní kulturou, a tak je nutné vždy hledat určitý kompromis. V případě metodiky Scrum nejspíš nedojde přímo k rozporu – metodika nedefinuje nic, co by se mohlo vylučovat s tradičním přístupem k řízení projektu, potažmo podniku. S největší pravděpodobností je ale třeba Scrum rozšířit, aby byl lépe použitelný v korporátním prostředí.

Ve velkých podnicích bývají obecně vyšší nároky na formální proces vývoje softwaru, jeho dokumentaci a kontrolovatelnost. I to je nejspíš jeden z důvodů, proč je v korporátním prostředí kladen důraz na dokumentaci nejen samotného produktu, ale i průběhu jeho testování.

4.6 Scrum a řízení kvality

Ve velkém podniku bývají managementem vyžadovány přesné plány testování, jasně dokumentované jednotlivé fáze testování a jejich výstupy. Tato dokumentace spočívá jak v definici testovacích scénářů, tak v reportování samotných běhů testů. V rámci agilních metodik nic z toho není definování. V rámci Scrum například neexistuje role testera, tudíž nejdou definovány ani jeho schopnosti nebo odpovědnosti. Přesto z metodiky Scrum vyplývají některá specifika. Nejdříve je ale nutné odlišit agilní způsob testování od tradičního přístupu, poté jsou tato specifika více zřejmá.

4.6.1 Agilní versus tradiční testování

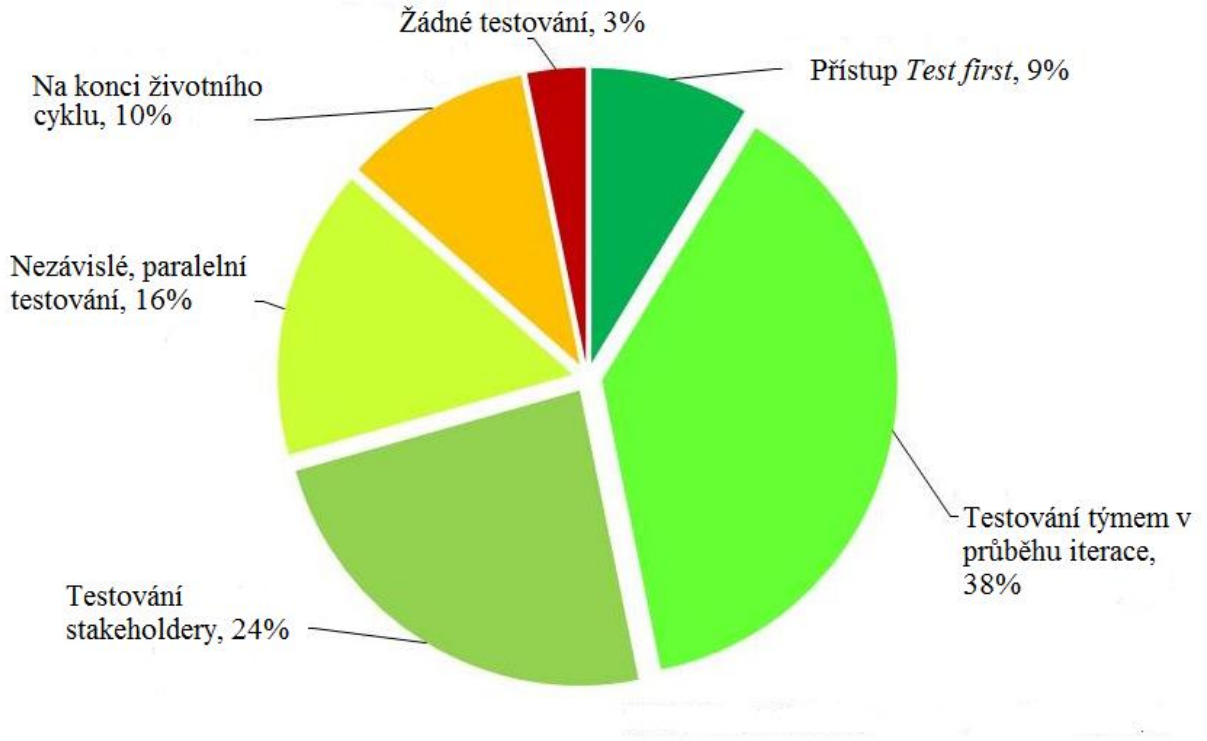
Tradiční přístup k vývoji softwaru počítá s rolí test manažera, který řídí testování produktu – jeho dvě hlavní zodpovědnosti a role, podle Linze (2014, s. 56) jsou:

- Organizace a vedení testovacího týmu (týmů) – role projektového vedoucího pro *podprojekt testování*
- Plánování rozsahu a způsobu testování – role experta na oblast testování

Ve velkém podniku je většinou kladen důraz na to, aby všechny fáze testování byly přesně zdokumentovány, probíhá formální akceptační testování, apod. Agilní metodiky žádnou takovouto formalizaci neznají.

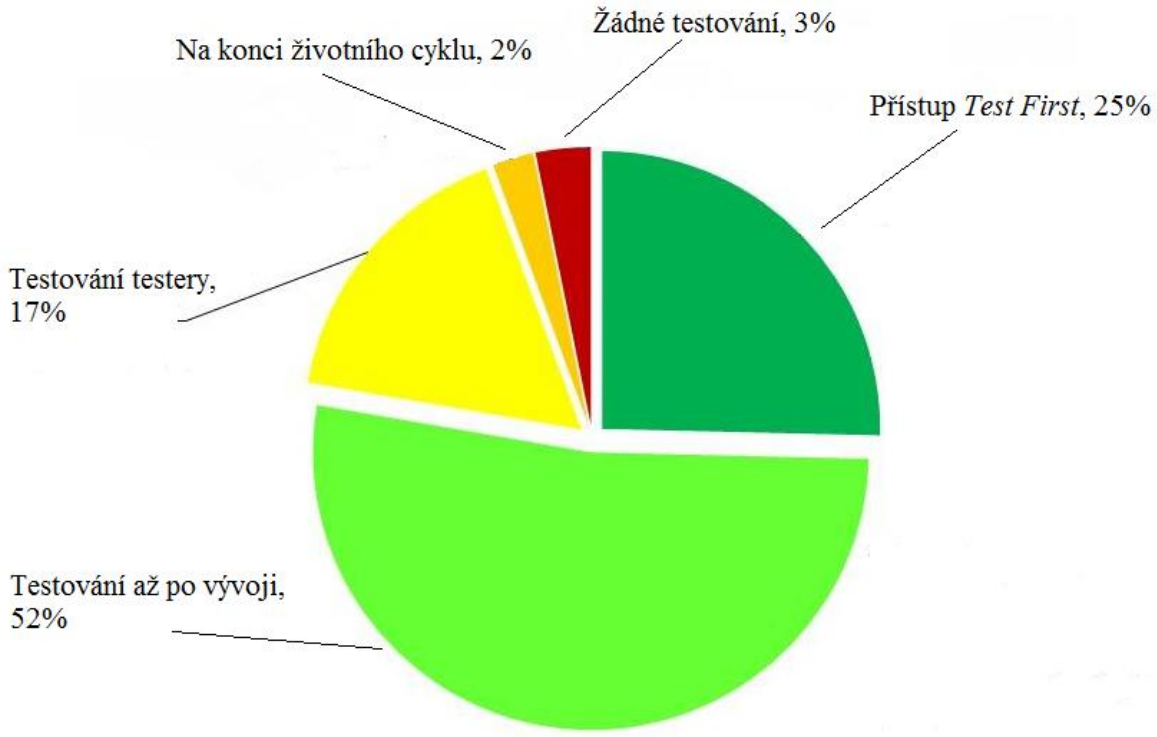
4.6.2 Testování v agilním prostředí

Scott Ambler (2012) ve svém průzkumu testování v agilním prostředí došel k zajímavým závěrům, viz Obrázek 1. Z Amblerova průzkumu vyplývá například, že většina akceptačního testování je prováděna samotným vývojovým týmem.



Obrázek 1: Přístup agilních vývojových týmů k akceptačnímu testování. Zdroj:(Ambler, 2012)

Z uvedeného průzkumu dále vyplývá, že u interního testování v rámci vývoje je výrazně častěji využíván přístup *test first* než u akceptačního testování. Využití akceptačního testování až po vývoji může často vést k tomu, že je sice funkcionalita dobře otestována, ale je až pozdě zjištěno, že zákazník chtěl funkcionalitu zcela jinou.



Obrázek 2: Přístup agilních vývojových týmů k testování v rámci vývoje. Zdroj:(Ambler, 2012)

4.6.3 Testování v rámci metodiky Scrum

Jak bylo výše uvedeno, tradiční přístup počítá s rolí test manažera, který řídí testování produktu. Přístup metodiky Scrum je odlišný. Scrum tým je sebe-organizující, a multifunkční, a hlavně má vždy zodpovědnost za práci jako celek. Programování a testování jsou prováděny v rámci jednoho týmu. V metodice Scrum, na rozdíl od tradičního přístupu, neexistuje test manažer. Jediný, kdo může mít zodpovědnost za organizaci testování, je Scrum Master, ale zodpovědnost za testování (stejně jako za vývoj, jednoduše za produkt jako celek) je na celém týmu (Linz, 2014).

Přestože je za testování (včetně určení jeho strategie a rozsahu) zodpovědný celý tým, který sdílí v případě potřeby testovací úkoly, testování často vyžaduje specifické schopnosti a ke správným rozhodnutím ohledně testování jsou třeba praktické zkušenosti z této oblasti. Proto Linz (2014) doporučuje mít ve Scrum týmu alespoň jednoho člena, který má potřebnou kvalifikaci a praxi v oblasti testování. Takový člen tak v podstatě zastává roli tradičního test manažera, přestože ji nemá definovanou.

Přestože Scrum nedefinuje, které úrovně testování mají být provedeny, neznamená to vlastně žádnou změnu. Scrum neznamená, že by se nějaká část systému neměla testovat. Podle Linze (2014) Scrum ale doporučuje, aby jednotlivé úrovně testování byly vykonávány paralelně, ne sekvenčně, je tomu opět u tradičních projektů. Scrum, a obecně agilní přístup, tak nemění způsob testování z hlediska rozsahu, ale z hlediska způsobu vykonávání stejných aktivit.

Protože výstupem každého sprintu je přírůstek produktu, který je možné okamžitě dodat zákazníkovi, musí tento přírůstek být v daném sprintu nejen implementován, ale i otestován. Z toho vyplývá, že je testování prováděno nejen v rámci jednoho týmu, ale i v rámci jedné iterace, snaha je tedy o co největší paralelizaci testování a vývoje. Snahou vývojáře tedy je dodat co nejdříve testovatelnou jednotku práce.

Vzhledem k potřebě reportingu v oblasti řízení kvality je zde nutné uvažovat o rozšíření metodiky Scrum. Vhodným rozšířením by také bylo možné přidat ověření, že je vyvíjena správná funkcionalita – řešení problému vyplývajícího z podkapitoly 4.6.2.

4.7 Možnosti rozšíření metodiky Scrum

Jak autoři metodiky Scrum uvádějí v Průvodci Scrumem (Schwaber a Sutherland, 2013), „*Scrum není proces pro samotný vývoj produktů, je to spíše procesní rámec, uvnitř kterého lze používat jiné procesy a techniky*“. Toto tvrzení přímo vybízí k rozšíření metodiky o další techniky, zpravidla je vhodné toto rozšíření na nižší úrovni, tedy o techniky zabývající se přímo samotným vývojem softwaru a souvisejícími praktikami. Nabízejí se tak přímo techniky podporující samotný proces vývoje softwaru nebo konkrétní testovací techniky – nic z toho nekoliduje s procesním rámcem Scrum, který definuje spíše role a činnosti na úrovni řízení vývoje, ne samotného procesu vývoje.

Vzhledem k oblasti testování by bylo vhodné metodiku Scrum rozšířit o další roli, roli testera. Dále, vzhledem k výsledkům průzkuma Scotta Amblera (2012) uvedených výše, je vhodné

zavést přístup, kdy je lépe spolupracováno na specifikaci požadavků. Vývojové týmy příliš nevyužívají přístup *test first* u akceptačních kritérií, přestože pro jednotkové testování tuto techniku využívá čtvrtina týmů. Z toho vyplývá, že přestože agilní metodiky akcentují zapojení zákazníka do vývoje, stále není dostatečně ověřováno, že je vyvíjena správná funkcionalita, kterou zákazník opravdu chce.

Z výše uvedeného průzkumu Forrest Inc. (Giudice, 2014) vyplývá, že většina firem v praxi nepoužívá Scrum samostatně, bez dodatečných úprav. Většina firem, které byly součástí průzkumu, uvádí, že Scrum nepoužívají samostatně jako jediný přístup k vývoji softwaru. Většinou týmů je kombinován s technikami Extrémního programování, Test Driven development, apod.

4.8 Shrnutí

Podle Buchalcevové (2009, s. 145) je metodika Scrum zaměřena na řízení projektu, nepoopisuje procesy, ale definuje praktiky. Scrum tak nezasahuje do samotného vývoj softwaru, z čehož vyplývá možnost, až možná potřeba, rozšíření metodiky o další techniky zasahující právě samotný vývoj softwaru. Z průzkumu (Giudice, 2014) vyplývá, že v praxi tomu tak opravdu je, a firmy v praxi nepoužívají Scrum samotný, dochází k jeho rozšiřování. Metodika Scrum se tak dá považovat za poměrně otevřený procesní rámec, a formálně nic nebrání jejímu rozšíření o další postupy.

Vzhledem k problémům vznikajícím nasazením metodiky Scrum do korporátního prostředí je vhodné využít přístup Behaviour Driven Development (viz kapitola 5), který si klade za cíl přemostění bariéry mezi technickými a netechnickými členy projektového týmu, případně jinými zainteresovanými stranami.

5 Behaviour Driven Development

Behaviour Driven Development není metodikou, nedefinuje role ani činnosti, nepokrývá celý životní cyklus vývoje softwaru. V této kapitole je nejprve popsán jeho vývoj z přístupu Test Driven Development, který je stručně charakterizován, aby bylo zřejmé, proč došlo k evoluci do Behaviour Driven Development. Dále je již popsán samotný Behaviour Driven Development, jeho principy a techniky. V závěru kapitoly je popsán BDD z perspektivy korporátního prostředí a možnosti využití Behaviour Driven Development v rámci velkých podniků.

5.1 Vývoj přístupu Behaviour Driven Development

Behaviour Driven Development je přístup využívaný v agilním prostředí, reflektuje tedy principy popsané v Agilním Manifestu (Beck et al., 2001). Jak uvádí například Astels (2006), Behaviour Driven Development navazuje na Test Driven Development, přístup, který je využíván například v Extrémním programování.

5.1.1 Test Driven Development

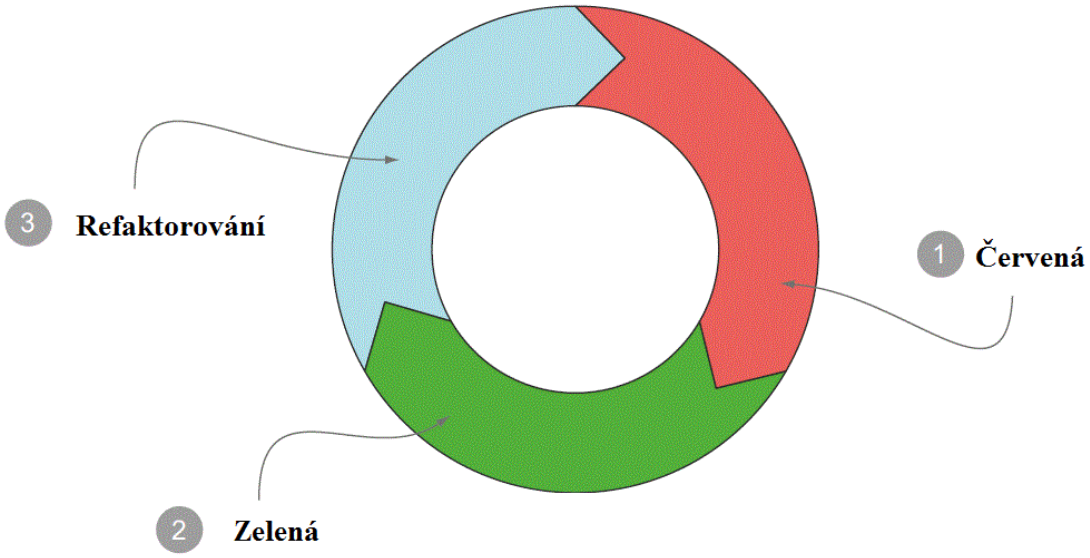
Vývoj řízený testy neboli Test Driven Development (TDD) je způsob vývoje softwaru využívající přístup *test first*, kdy vývojář nejdříve napíše test a poté napíše přesně tolik kódu, aby test prošel. Cílem Test Driven Development je psát *čistý kód, který funguje* (ANON. nedatováno). Podle Becka (2003, s. 8) následuje vývojář v TDD dvě hlavní pravidla:

- Piš nový kód pouze v případě, že pro něj máš nejprve napsaný neprocházející automatizovaný test,
- Eliminuj duplicity.

A tato dvě pravidla aplikuje pomocí tří základních kroků:

- **Červená** – napiš malý test, který neprojde, dost možná se nejprve ani nezkompiluje
- **Zelená** – co nejrychleji *zprovozni* test: napiš minimální nutné množství kódu v libovolné kvalitě
- **Refaktorování –** eliminuj duplicity způsobené rychlým napsáním funkcionality, napiš kód lépe.

Červená – zelená – refaktorování je tak jakousi mantrou provázející celý proces vývoje pomocí Test Driven Development. (Beck, 2003, s. 9) Tento cyklus znázorňuje Obrázek 3.



Obrázek 3: Cyklus Test Driven Development. Zdroj:(Smart, 2014)

Důsledkem Test Driven Development je pokrytí funkcionalit testy. Není možné očekávat, že tyto testy nahradí například testy výkonu nebo testy použitelnosti systému, ale i přesto lze očekávat nízkou hustotu defektů v kódu. Takže role testera se vlastně mění z čisté supervize spíše na prostředníka komunikace mezi těmi, kdo vědí, co systém má dělat a těmi, kdo systém implementují. (Beck, 2003, s. 105)

5.1.2 Od Test Driven Development k Behaviour Driven Development

Jak uvádí Wynne (2012), jednotkové testy slouží k ujištění, že je produkt vyvíjen správně, ale akceptační kritéria slouží k ujištění, že je vyvíjen správný produkt. Díky Test Driven Development je tak sice důkladně již v rané fázi vývoje ověřeno, že je daná funkcionalita implementována správně. Na druhou stranu, není nikde ověřeno, že danou funkcionalitu takto zákazník chtěl. Dochází tak sice k výraznému zvýšení kvality kódu díky jeho pokrytí testy, ale zákazníkovi přesto může být dodán pro něj naprosto nepoužitelný produkt.

Behaviour Driven Development je tak evolucí Test Driven Development přidáním aspektu zákazníka. Podle Astelse (2006) je Behaviour Driven Development posun od *psaní testů pro ověření funkčnosti jednotlivých metod* k *definici požadavku zákazníka*. Což je významný

posun: dochází ke dvojímu ověření – jak že je produkt vyvíjen správně, tak že je vyvíjen správný produkt. BDD může pomoci byznysu definovat očekávané výstupy, zlepšit porozumění ze strany vývoje o zákazníkových požadavcích, ale i zlepšit porozumění ze strany byznysu, co se týká potenciálních technických omezení souvisejících s požadavkem. Ale hlavně, pomáhá vývojovému týmu ve spolupráci s byznysem dodat software, který naplňuje stanovené byznys cíle (Inviqa, 2015).

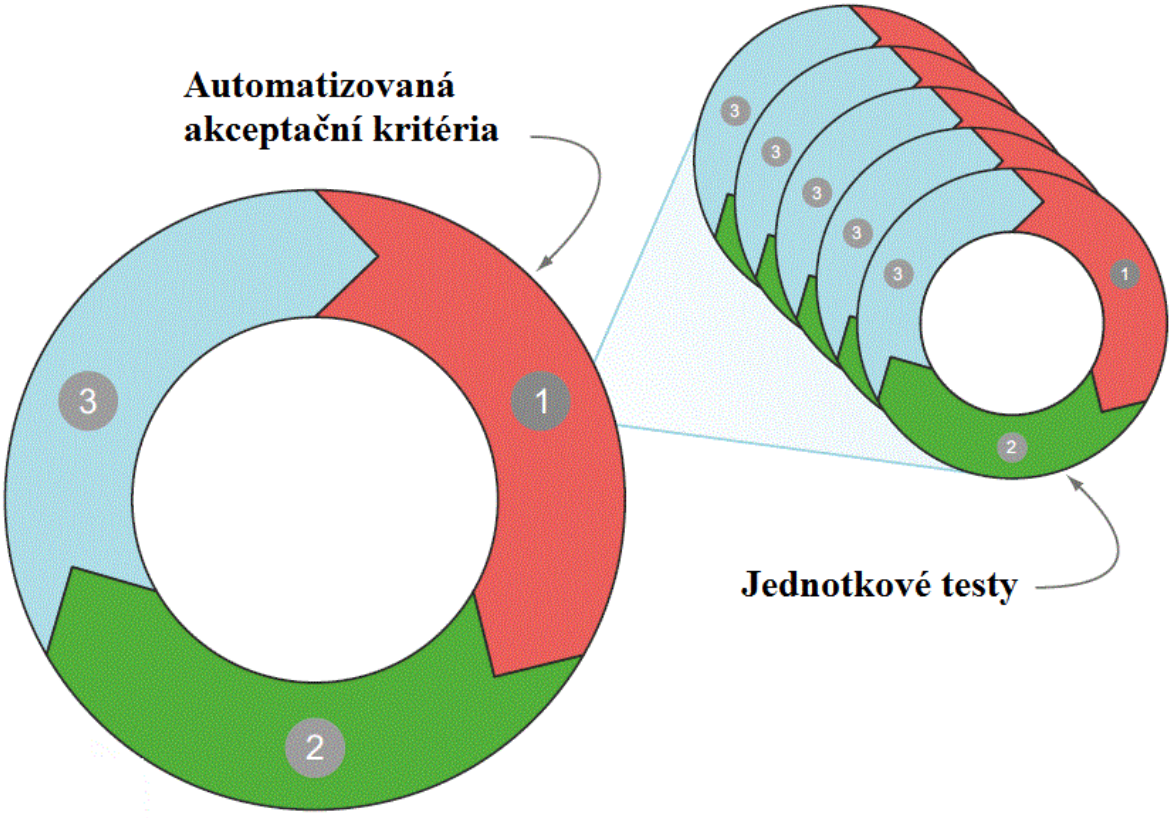
Obrat nastává i v samotném pohledu na testy, chceme-li, v terminologii. Tým praktikující Behaviour Driven Development neříká *testy*, ale *scénáře*. Stejně jako nemá *jednotkové testy třídy*, ale *testy reflektující funkcionalitu třídy*. Nepoužívá pojem *funkční testy*, těm raději říká *specifikace chování produktu* (Agile Alliance and Institut Agile, 2013). Je zde vidět určitý posun ve vnímání testů od technického pohledu, jestli kód vrací správné hodnoty, k více funkčnímu, uživatelskému pohledu. Test začíná být něčím, co ověřuje, že celá funkcionalita dělá, co má, a hlavně, co zákazník očekává. Velmi dobře tento myšlenkový posun znázorňuje slovník Liz Keogh, viz Tabulka 1: Slovník TDD – BDD.Zdroj:(Keogh, 2009). Takovýto *překlad* samozřejmě není možné brát doslovně, ale k určité ilustraci myšlení v rámci TDD a BDD slouží dobře.

Test Driven Development	**Behaviour Driven Development**
Třída pokrytá testem	Popsaná třída
Metoda pokrytá testem	Chování, které má přidanou hodnotu
Co testovat?	Jakou přidanou hodnotu má tato třída?
Jak testovat?	Jak chci tuto třídu použít?
Rozhraní	Role
Mock	Spolupracovník hrající <tuto> roli
Design	Zodpovědnost
Procházející (test)	Fungující, přinášející hodnotu
Neprocházející (test)	Má to dělat přesně, co jsem popsal?
Ověřit, že…	Ujistit se, že…
Vrací *true / false*	Říká mi, že…
Vrací <*objekt*>	Dává mi…
Implementuje <*rozhraní*>	Poskytuje <*výhodu role*>
Psát kód čistě, aby se *nerozbil*	Psát kód, který je jednoduše použitelný, srozumitelný, upravitelný
100% pokrytí	Můžeš provést změnu v kódu, máš dostatek informací, abys to mohl udělat bezpečně.

Tabulka 1: Slovník TDD – BDD.Zdroj:(Keogh, 2009)

Behaviour Driven Development pomáhá týmu zaměřit své síly na identifikaci, porozumění, a dodávání funkcionalit, které jsou z pohledu byznysu hodnotné. Tyto funkcionality mají být správně jak navrhovány, tak implementovány. (Smart, 2014, s. 35)

Jestliže je přístup Test Driven Development složen z cyklu červená – zelená – refaktorování, Behaviour Driven Development těchto cyklů obsahuje několik, a to na různých úrovních detailu. (Smart, 2014) Behaviour Driven Development si tak lze představit jako na Obrázek 4, kde jsou znázorněny dvě úrovně testů: automatizovaná akceptační kritéria, nejkomplexnější testy a jednotkové testy, tedy testy na nejnižší úrovni. Zahrnuty mohou být všechny ostatní úrovně – jednotlivé úrovně testů jsou spolu vždy logicky provázány. V principu - pro každou vyšší úroveň je třeba mít několik testů nižší úrovně. Důležité je si uvědomit, že Behaviour Driven Development nejsou jenom automatizovaná akceptační kritéria



Obrázek 4: Cyklus Behaviour Driven Development. Zdroj:(Smart, 2014), překl. Autorka

5.2 Principy Behaviour Driven Development

Chelimnsky (2010, s. 139) definuje tři základní principy Behaviour Driven Development:

- Dostatečné je dostatečné
- Dodávej stakeholderům přidanou hodnotu
- Všechno je o *chování*

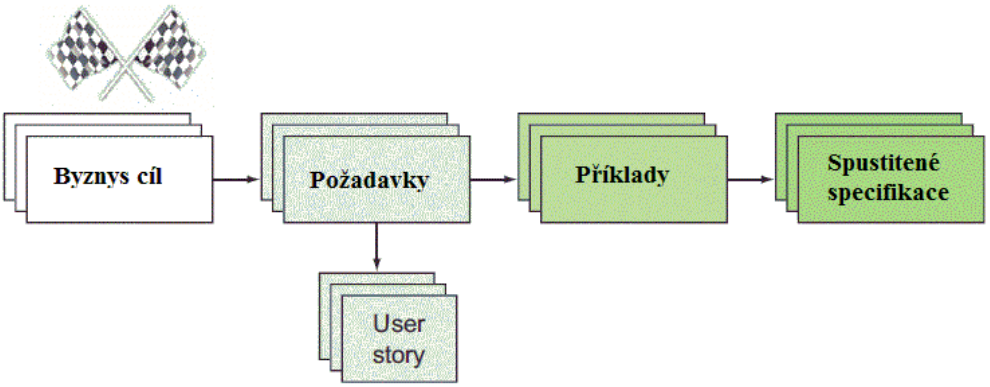
Dostatečné je dostatečné v podstatě znamená, že stačí plánovat, analyzovat a navrhovat systém jen do té míry, aby mohl být započat vývoj. Všechno navíc je jen zbytečná práce. Tento princip lze aplikovat nejen na vývoj, ale i na ostatní procesy: automatizovat sice ano, ale ne do zbytečné míry. Automatizovat sestavení a nasazení produktu má smysl, ale snažit se automatizovat úplně všechno je zbytečná práce.

Dodávej zainteresovaným přidanou hodnotu. Jednoduše nic, co nevytváří přidanou hodnotu pro zainteresované osoby (stakeholdery) anebo není nutným předpokladem pro vytvoření další přidané hodnoty, nemá smysl dělat. Nedělej to.

Všechno je o chování. Je jedno, jestli se jedná o kód nebo třeba konfiguraci. Vždy lze použít stejné myšlení a stejné jazykové konstrukce, abychom popsali chování, nezávisle na úrovni granularity.

Tyto základní principy opět vycházejí z Test Driven Development, který například říká, že je nutné napsat vždy jen tu nezbytnou část kódu, aby prošel předem definovaný test.

5.3 Praktiky a techniky Behaviour Driven Development



Obrázek 5: Vztah byznys cílů, požadavků, user stories, příkladů a spustitelných specifikací. Zdroj:(Smart, 2014), překl.autorka

Základní praktikou Behaviour Driven Development je popis jednotlivých požadavků pomocí *user stories* a konkrétních příkladů. Právě příklady hrají v Behaviour Driven Development jednu z hlavních rolí. Vztah jednotlivých definic je znázorněn na Obrázku Obrázek 5.

Každý byznys cíl je reflektován nějakým požadavkem na dodávaný software. Těchto požadavků může být k danému byznys cíli více. Naopak, požadavků je třeba vyvíjet přesně tolik, aby pomohly naplnit byznys cíle, ne víc. Jednotlivé požadavky mohou být *rozpadnuty* na několik *user stories*. Někdy lze požadavek definovat pouze jednou *user story*, jindy je jich potřeba více. Jednoduchým vodítkem je, že jedna *user story* by vždy měla být jen tak komplexní, aby byla jednoduše a samostatně implementovatelná v rámci jedné iterace. Každému požadavku dále náleží konkrétní příklady – tedy jaký vstup a výstup uživatel očekává. Na samotném konci řetězce jsou spustitelné specifikace – tedy jakési automatizované testy ověřující definovanou funkcionalitu. Tyto testy jsou založeny právě na příkladech popsaných uživatelem. (Smart, 2014)

Spustitelné specifikace vyplývající z požadavků a scénářů mají podle Smarta (2014) kromě samotného vyjasnění požadavků další dvě důležité role: dokumentace a validace.

5.3.1 Živá dokumentace

Živá dokumentace, která je výstupem spustitelných specifikací, funguje jak jako dokumentace požadavků, tak jako technická dokumentace. Jako živá dokumentace totiž slouží samotné požadavky, které, protože jsou spustitelné, je možné rychle ověřit. Existuje tak v jakýkoliv okamžik přehled toho, které požadavky jsou aktuálně naimplementovány.

Je třeba zdůraznit, že dokumentace nebo reporty vzniklé jako výstup Behaviour Driven Development nejsou například výstupem v příkazovém řádku nebo logem. Mají formu srozumitelnou pro byznys, a proto se dají využít přímo k reportingu

5.3.2 Automatická validace

Automatizované testy vzniklé spustitelnými specifikacemi nejsou jenom jednorázově funkční artefakt. Slouží po celý zbytek životního cyklu produktu – pokud nedošlo ke změně požadavku nebo implementaci jiného požadavku, který vylučuje ten původní – jako automatizované regresní testy. Takové testy vždy hlídají, že se při implementaci nového požadavku nerozbila funkčnost některého z dříve implementovaných. Velkou výhodou

takovýchto testů je, že jsou vytvořeny hned v úvodu implementace, vývojový tým tedy *nestojí* nic navíc to, že má celý systém velmi dobře pokryt regresními testy, které mohou pomoci odhalit případný vzniklý defekt jeho v raném stádiu. A opět, díky generování živé dokumentace lze díky Behaviour Driven Development dokumentovat průběh testování.

5.4 Behaviour Driven Development v korporátním prostředí

Z charakteristiky metodiky Scrum a úvahy nad specifikami korporátního prostředí vyplynuly dodatečné požadavky na metodiku, jestliže má být nasazena v korporátním prostředí:

- Formální dokumentace testování
- Detailní dokumentace produktu
- Komunikace mezi vývojovým týmem a okolím

5.4.1 Formální dokumentace testování

Přestože je například Smartem (2014, s. 18) akcentováno, že BDD není pouze přístup k testování nebo způsob, jak automatizovat testovací scénáře, přináší Behaviour Driven Development zajímavý pohled z testovací perspektivy.

Z výše uvedeného vyplývá, že hlavním přínosem BDD je komunikace mezi zákazníkem a vývojovým týmem. Hlavním přínosem je definice požadavků a scénářů, která je oběma zainteresovaným stranám jasná a jednoznačná. Toho se však dá využít nejen definicí spustitelných akceptačních kritérií. Tester může snadno k jednotlivým požadavkům přidávat další scénáře, které plánuje testovat. Sám tak získává transparentní přehled o postupu vývoje a testování. Celý vývojový tým, především testeři můžou těžit z toho, že automatizované testy nejsou jen jednoslovně pojmenovaná metoda, ale jsou k nim navázány konkrétní scénáře. Ty fungují i jako *živá* dokumentace Zvlášť ve velkém týmu může být výhodou mít automatizované testy popsané jednoduchou větou v přirozeném jazyce. V případě začlenění těchto výhod i do integračního serveru je jednoduše sledovatelné, které scénáře a funkcionality právě fungují nebo naopak obsahují defekt.

Gherkin tak tvoří společnou platformu pro zákazníka, vývojáře, ale i testera nebo případně analytika. Všichni zúčastnění jsou schopni rozumět dané syntaxi. Zákazník může definovat své požadavky a zapsat je v jazyce Gherkin. Tester spolu s analytikem poté může vymyslet další dodatečné scénáře, které bude testovat.

Popsané scénáře poté tester (případně ve spolupráci s vývojářem) automatizuje pomocí vhodného nástroje – jak je ukázáno později. Ze scénářů se tak stávají automatizovaná akceptační kritéria a automatizované regresní testy, které dokáží dobře sledovat aktuální stav systému.

5.4.2 Dokumentace produktu

Přístup Behaviour Driven Development staví na principu spustitelných akceptačních kritérií. Ta však slouží nejen k jednoznačné definici uživatelských požadavků a akceptačních kritérií. Dají se velmi dobře využít jako živá dokumentace produktu. Pokud jsou dostatečně kvalitně popsána akceptační kritéria a jejich scénáře, vzniká velmi kvalitní, živá produktová dokumentace, která, na rozdíl od běžné dokumentace, reflektuje reálný stav produktu.

Živá dokumentace reflektuje aktuální stav ze dvou hlavních důvodů: popisuje systém aktuálními požadavky – nestane se, že by byl zadokumentován požadavek, který již není například kvůli změně produktu aktuální. A pokud náhodou ano, bude mít takovýto požadavek nebo konkrétní scénář červené testy. Výstup z běhu automatizovaných testů je totiž druhou částí živé dokumentace, která dává dokonalý přehled o tom, v jakém stavu se produkt aktuálně nachází.

Výstupy z pohledu dokumentace jsou tak z Behaviour Driven Development hned tři:

- Dokumentace požadavků
- Popis funkcionalit systému a jeho komplexní dokumentace pomocí scénářů
- Aktuální stav požadavků na základě výstupu z běhu automatizovaných testů

Zásadní výhodou je fakt, že k vytvoření výše uvedených dokumentů není potřeba téměř žádné zvýšené úsilí, a dokumenty není třeba aktualizovat; jak bylo výše uvedené, jsou generovány automaticky, na základě aktuálního stavu systému.

5.4.3 Komunikace s okolím projektu

Komunikace s okolím projektu vlastně úzce souvisí s uživatelsky přívětivou dokumentací: díky reportingu v reálném čase roste transparentnost vývoje. Management si kdykoliv může zjistit, v jaké fázi vývoje se tým a jeho produkt právě nachází.

5.5 Styčné body s metodikou Scrum

Z výše uvedené charakteristiky metodiky Scrum a přístupu Behaviour Driven Development vyplývá, že každý z přístupů má úplně jinou granularitu, a zaměřuje se na zcela odlišné záležitosti: Scrum je procesní rámec, spíše projektová metodika, na druhé straně Behaviour Driven Development je přístup, který zasahuje do samotného procesu vývoje. I tak lze ale po analýze obou přístupů tušit, že existují určité styčné body metodiky Scrum a Behaviour Driven Development. Oba přístupy staví na uživatelské definici požadavků – ať už v podobě User Stories nebo v podobě akceptačních kritérií. Dále, metodika Scrum si jako jeden ze svých hlavních cílů klade zvýšení transparentnosti vývoje. Což BDD svým *zlidštěním* automatizovaných testů poskytuje.

5.5.1 User Stories

Metodika Scrum využívá k definici požadavků *user stories*. Pokud se podíváme na požadavek definovaný podle přístupu Behaviour Driven Development, kopíruje strukturu User Story, která vypadá zhruba následovně:

Jako uživatel
Chci mít možnost stisknout tlačítko
A zobrazit seznam zákazníků.

Požadavky známe z Behaviour Driven Development mají přesně tuto strukturu. Resnick (2011 , s. 86) při zmínce o akceptačních kritériích, které by každá User Story měla mít, uvádí, že akceptační kritéria se dají popsat v jazyce Gherkin a automatizovat. A odkazuje na nástroj SpecFlow. Pro tým vyvíjejí podle metodiky Scrum tak není problém začít využívat přístup Behaviour Driven Development, protože už zná strukturu user stories.

Přestože Smart (2014, s. 129) uvádí, že Požadavek nerovná se User Story (a jeho odůvodnění o různé úrovni granularity jsou na místě), připouští, že jsou týmy, které mezi tyto pojmy dávají rovnítko. *User story* může pokrývat celý požadavek, někdy může být k pokrytí požadavku potřeba více *user stories.* Pro zjednodušení, a částečně, i protože nástroj SpecFlow používá pojem Požadavek, je v metodice používán tento pojem. Kvůli tomu by ale měla vždy být snaha o co nejméně komplexní požadavky.

Požadavek není možné přímo zaměnit za *user story*, user story může zasahovat pouze část požadavku. Na druhou stranu existují požadavky, k jejichž popisu stačí pouze jedna *user story*, v tom případě požadavek = *user story*. Důležitá je ale shoda ve struktuře a tak vzájemná srozumitelnost.

5.5.2 Transparentnost

V kapitole o Scrum byla jako jeden z hlavních principů metodiky uvedena transparentnost. Právě transparentnost použití přístupu Behaviour Driven Development posiluje, a to následujícími způsoby:

- Definice požadavků v jazyce Gherkin
- Přehledný reporting, co bylo testováno, a s jakým výsledkem

Využití jazyka Gherkin pro definici požadavků má jasný přímý důsledek: obě strany vědí (a mají v to důvěru), co má být implementováno. Definice požadavků včetně scénářů v jazyce Gherkin je pro obě strany srozumitelná a jednoznačná. A dále, zákazník může mít srozumitelný přehled o tom, co je právě vyvíjeno nebo které scénáře jsou funkční.

5.6 Shrnutí

Metodika Behaviour Driven Development si klade za cíl přemostění bariéry mezi technickými členy týmu a byznysem nebo obecně uživatelem produktu. A to nejen mnoha dostupnými nástroji, ale hlavně se snaží „najít společnou řeč“ obou stran. Povzbuzuje vzájemnou komunikaci a dává oběma stranám společný komunikační prostředek – definice požadavků. Autoři metodiky sice vyzývají, že se nejedná pouze o testování, ale o komplexní přístup, avšak právě do oblasti testování se mnoho z BDD dá přenést a aplikovat i v rámci jiných metodických postupů. A hlavně – zlepšení transparentnosti a komunikace v rámci projektu se dá očekávat, i když se postupů BDD využije *pouze* v oblasti testování, protože vzniklé spustitelné scénáře jsou tím důležitým výstupem, živou projektovou dokumentací s velkou vypovídací hodnotou.

6 Nástroje pro Behaviour Driven Development

Behaviour Driven Development využívá mnohé softwarové nástroje, které se ve své podstatě specializují na testování. Jejich princip spočívá v přímém propojení testovacího scénáře s automatizovaným testem, tedy automatizaci testovacích scénářů. Téměř všechny BDD nástroje využívají jazyk Gherkin, jeho základní syntaxe je tedy využita napříč nástroji, liší se pouze konkrétní implementací pro daný programovací jazyk.

6.1 Jazyk Gherkin

Gherkin je jazyk ke specifikaci uživatelských požadavků. Gherkin byl navržen tak, aby byl jednoduše srozumitelný byznysu a jiným zainteresovaným osobám, a zároveň snadno automatizovatelný pomocí nástrojů BDD (Smart, 2014, s. 50). Jde o přirozený jazyk s přidanou strukturou, právě aby odpovídal oběma výše uvedeným požadavkům. V současnosti existuje implementace Gherkin pro více než 60 světových jazyků včetně češtiny (Cucumber Ltd., nedatováno), čímž je zajištěna maximální srozumitelnost přirozeného, ideálně mateřského jazyka obou zúčastněných stran. Cílem jazyka Gherkin je maximální jednoduchost – aby byl jednoduše pochopitelný lidmi, kteří neprogramují – ale zároveň dostatečná struktura, aby byly definice v něm uvedené jasné, a jednoznačně automatizovatelné. Jazyk Gherkin využívá formátu požadavků, které mají strukturu *user stories* známých například právě z metodiky Scrum.

6.1.1 Syntaxe jazyka Gherkin

Základní syntaxe Gherkin je založena na několika klíčových slovech, jejichž výčet a překlad lze nalézt přímo ve zdrojovém kódu projektu Gherkin (Cucumber Ltd., 2016). Níže je vždy uvedeno české klíčové slovo, plus v závorce jeho originální znění v angličtině. Pokud je českých variant více, jsou rovnocenné a byly vytvořeny kvůli rozdílné logice českého a původního anglického jazyka.

- Požadavek (feature),
- Pozadí, kontext (background),
- Scénář (scenario),
- Pokud (given),
- Když (when),

- Pak (then),
- A, a také (and),
- Ale (but),
- Příklady (examples)

Požadavek je klíčové slovo, kterým začíná každý *feature* soubor. Nemá význam pro samotné chování testů nebo systému, ale identifikuje místo pro popis dále specifikované funkcionality. Ihned za klíčovým slovem následuje název funkcionality (příhodné je mít jednotný název souboru a funkcionality), na dalších několika řádcích poté její popis. Popis může být na libovolný počet řádků, jeho konec je rozpoznán jednoduše příchodem dalšího klíčového slova, za validní Gherkin požadavek je požadavek považován pouze v případě, že následuje klíčové slovo Scénář, Pozadí nebo Náčrt scénáře (Wynne M., 2012, s. 47). Struktura požadavku vypadá následovně:

Požadavek: Odeslání zprávy
Popis funkcionality Odeslání zprávy
Který může být dlouhý libovolný počet řádků, přičemž není rozdíl, jestli jsou informace odděleny novým řádkem nebo ne.

Mohou být odděleny i více řádky, pro Gherkin to není rozdíl.

Scénář slouží k vyjádření požadovaného chování aplikace. Každý soubor *feature* obsahuje alespoň jeden, zpravidla však více scénářů. Každý scénář je konkrétní případ, jak se má systém chovat v dané situaci, malý příběh, popisující něco, co systém má umět. Scénáře popisují i okrajové případy a požadované chování za nestandardních situací. Všechny scénáře mají stejný vzorec:

1. Systém je v určitém stavu,
2. směrem k systému je vyslána určitá akce,
3. očekávaná reakce systému.

Tento vzorec umožňuje jasně rozpoznat, jestli se systém chová, jak bylo požadováno nebo ne (Wynne M., 2012, s. 48). Správně napsaný scénář je deklarativní, ne imperativní. Popisuje, *co* má systém vykonat, ne *jak* má akci vykonat. (Smart, 2014, s. 162)

Pokud je klíčové slovo použité k definici kontextu, pozadí, výchozího stavu systému (viz bod 1 odstavce Scénář) (Wynne M., 2012, s. 48). Může obsahovat také soubor prerekvizit, které musely být vykonány, aby se systém dostal do požadovaného stavu (Smart, 2014, s. 160).

Když znázorňuje akci vyslanou směrem k systému, testované chování (viz bod 2 odstavce Scénář), typicky například interakce uživatele (Wynne M., 2012, s. 49). Smart (2014, s. 161) poukazuje na fakt, že je důležité vystihnout správnou úroveň detailu, aby byl krok jednoznačně definován, ale na druhou stranu neobsahoval nedůležité informace.

Pak popisuje očekávaný výstup, reakci systému (viz bod 3 odstavce Scénář). Slouží ke kontrole, že výstup interakce je korektní a očekávaný (Wynne M., 2012, s. 50). Smart (2014, s. 162) upozorňuje, že je třeba striktně rozlišovat mezi kroky uvedenými jako Když a Pak. Je třeba mít na paměti interakci kroku Když a výsledek v kroku Pak.

A, a také umožňuje přidat víc každého z předchozích kroků. K systému mohou být vyslány například dvě akce nebo uživatel očekává, že odezva systému spočívá ve více krocích. Klíčové slovo A není povinné, je možné mít více řádků kroků Pokud, Když nebo Pak, ale kvůli lepší čitelnosti je vhodné využít i klíčové slovo A, případně A také.

Ale je podobný případ jako A, s rozdílem, že nejde o logickou spojku slučovací, ale vylučovací. Také se nejedná o povinné klíčové slovo.

Celý požadavek a scénář v souboru .feature vypdá například jako ve Výpis 1.

```
Požadavek: Hledání Googlem
      Abych našel nějaké téma na internetu
      Jako uživatel
      Chci vyhledávat Googlem a procházet výsledky

Scénář: Vyhledej 'SpecFlow' a ověř, že se zobrazily nějaké výsledky
      Pokud mám v prohlížeči otevřenou stránku Google
      A zadal jsem do vyhledávacího řádku 'SpecFlow'
      Když stisknu tlačítko Hledat Googlem
      Pak vidím výsledky vyhledávání
```

Výpis 1, Požadavek a scénář v souboru .feature

```
Požadavek: Hledání Googlem
      Abych našel nějaké téma na internetu
      Jako uživatel
      Chci vyhledávat Googlem a procházet výsledky

Scénář: Vyhledej 'SpecFlow' a ověř, že se zobrazily nějaké výsledky
      * mám v prohlížeči otevřenou stránku Google
      * zadal jsem do vyhledávacího řádku 'SpecFlow'
      * stisknu tlačítko Hledat Googlem
      * vidím výsledky vyhledávání
```

Výpis 2, Substituce klíčových slov hvězdičkami

Základní klíčová slova mohou být dokonce zastoupena pouze hvězdičkou, scénář potom ale ztrácí na čitelnosti a přehlednosti, viz Výpis 22, kde je stejný scénář jako v předchozím příkladu, pouze jsou nahrazena klíčová slova hvězdičkou. Tato substituce možná mírně zjednoduší psaní scénáře, jeho pozdější čitelnost a srozumitelnost je ale horší. Použití klíčových slov se tak jeví jako lepší pro zachování všech výhod jazyka Gherkin.

Příklady mohou být například testovací data pro daný scénář. Pokud má scénář něco vypočítat, za klíčovým slovem Příklady je uvedeno v tabulce několik ukázkových výpočtů.

6.1.2 Struktura jazyka Gherkin

Požadavky náležící stejné funkcionalitě jsou seskupeny do jednoho textu, souboru s příponou *.feature*. Soubor *feature* obsahuje krátký popis předmětné funkcionality, následují konkrétní scénáře nebo příklady, jak má systém fungovat. (Smart, 2014) Soubor *feature* je uložen jako prostý text, a lze ho editovat v běžném textovém editoru, není potřeba vývojové prostředí.

6.2 Přehled nástrojů pro BDD

Na myšlence Behaviour Driven Development je založeno velké množství nástrojů. Níže v tabulce 2 je jejich přehed z hlediska programovacích jazyků a funkcionality. Zdrojem pro tuto kapitolu jsou dokumentace jednotlivých produktů.

Jazyk	**Nástroje**
Java	JBehave, JDave, Easyb, Concordion, FitNesse
C#	SpecFlow, NBehave, NSpec, FitNesse
Ruby	Cucumber, MinitTest
PHP	Behat
Python	Behave, FitNesse
JavaScript	Jasmine, CucumberJS
iOS	Frank
Android	Calabash

Tabulka 2: Přehled BDD nástrojů podle programovacího jazyka. Zdroj: Dokumentace jednotlivých nástrojů

Snad pro všechny běžně používané platformy a programovací jazyky existuje nějaký softwarový nástroj pro Behaviou Driven Development. BDD nástroj najde programátor v jazyce Java, C#, PHP a další. Dokonce existuje i nástroj pro testování iOS a Android aplikací. V Tabulka 2: Přehled BDD nástrojů podle programovacího jazyka. Zdroj: Dokumentace jednotlivých nástrojů je vidět přehled nástrojů pro jednotlivé programovací jazyky.

6.2.1 Nástroje pro jednotlivé programovací jazyky

Nejvíce zastoupeny jsou nástroje pro jazyky Java a C#, což je důsledek toho, že tyto jazyky jsou nejpoužívanější – a zejména v korporátní sféře jsou velmi využívané. Je ale potřeba zdůraznit, že nástroje mezi sebou není možné zcela jednoznačně porovnávat kvůli implementacím pro různé programovací jazyky; těžko zvítězí nástroj pro PHP nad nástrojem pro Javu, když je potřeba nástroj pro vývoj v Javě. Není i proto důležité se snažit popsat detailně všechny nástroje, z přehledu nástrojů v Tabulka 3: Přehled BDD nástrojů. Zdroj: dokumentace jednotlivých nástrojů Tabulka 3: Přehled BDD nástrojů. Zdroj: dokumentace jednotlivých nástrojů

však vyplývá několik závěrů. Zásadní informace o nástrojích jsou vybrány níže.

6.2.2 Jazyk specifikace

Značná část BDD nástrojů je založena na jazyce Gherkin. Ten je jakýmsi standardem v definici požadavků pomocí Behaviour Driven Development. Ostatní nástroje většinou používají vlastní způsob definice požadavků, který ale často nebývá tak dobře odstíněn od zdrojového kódu, což definice dělá méně čitelné, a také už jsou definice závislé na konkrétním nástroji. Výjimkou je Jasmine, který sice nativně nevyužívá Gherkin, ale existuje rozšíření, které právě Gherkin doplňuje.

Výhodou použití nástroje založeného na jazyce Gherkin je možnost soubor se specifikacemi přenést a použít i s jiným nástrojem. Vývojový tým tak má svobodu volby nástroje, může ho v průběhu času změnit. Není ani problém se změnou programovacího jazyka, na kterém je Gherkin naprosto nezávislý.

6.2.3 Integrace nástroje do vývojového prostředí

Další, čím se nástroje liší, je jejich integrace do vývojového prostředí (v tabulce jako IDE). Nástroje jako JDave, SpecFlow, Cucumber nebo NSpec mají dokonalou integraci do vývojového prostředí. Znamená to, že nefungují pouze jako například příkaz v konzoli, ale jejich příkazy jsou po instalaci přítomné v kontextovém menu vývojového prostředí. To značně zjednodušuje práci s nástrojem, a usnadňuje přechod na něj.

6.2.4 Licence nástrojů

Všechny nástroje jsou distribuovány pod nějakým typem otevřené licence a jsou k dispozici zdarma. Jedinou výjimku tvoří SpecFlow, které je v základu také zdarma, ale některé jeho doplňky jsou placené. Ty však není nutné používat, lze je substituovat jinými.

Nástroj	Licence	Gherkin	Integrace do IDE	Reporting	Další funkce
JBehave	Otevřená	Ano	Ano	Text	X
JDave	Otevřená	Ne	Ano	XML	X
Easyb	Otevřená	Ano	Ne	Ne	X
Concordion	Otevřená	Ne	Ne	Ano	Backlog
SpecFlow	Otevřená + doplňky	Ano	Ano	Ano	Backlog, Pickles
NBehave	Otevřená	Ano	Ano	Ano	X
NSpec	Otevřená	Ne	Ano	Ne	X
Cucumber	Otevřená	Ano	Ano	Ano	X
Behat	Otevřená	Ano	Ano	Ne	X
Behave	Otevřená	Ano	Ano	Ne	X
Jasmine	Otevřená	Ne (doplněk)	Ne	Ne	X
FitNesse	Otevřená	Ne	Ano	Ano	X
Robot Framework	Otevřená	Ne	Ano	Ano	X
Codeception	Otevřená	Ne	Ne	Ne	X
Frank	Otevřená	Ano	Ne	Ne	X
Calabash	Otevřená	Ano	Ano	Ano	X
MiniTest	Otevřená	Ne	Ne	Ano	X

Tabulka 3: Přehled BDD nástrojů. Zdroj: dokumentace jednotlivých nástrojů

6.2.5 Použití nástrojů na integračním serveru

Snad všechny nástroje je možné nakonfigurovat pro použití na integračním serveru. Buď mají podporu v podobě pluginu nebo se dokonfigurují na integračním serveru, všechny je ale na integračním serveru možné použít.

6.2.6 Reportovací funkcionality nástrojů

Některé BDD nástroje zvládají pokročilý reporting. Ze všech je možné reportovat alespoň v textovém formátu, v tabulce 6 jsou však jako *Ano* uvedeny pouze ty nástroje, které umí pokročilý, například grafický reporting Jednoduše ty, které zvládají bez přidaného úsilí generovat dokumenty přímo použitelné pro management. Těmi jsou například SpecFlow, Cucumber nebo JBehave. Nástroj Concordion umí také vytvářet přehledné reporty, jeho nevýhodou je však to, že není založen na jazyce Gherkin. Stejně jsou na tom nástroje FitNesse a Robot Framework, které poskytují kvalitní reporty, ale nevyužívají jazyk Gherkin.

6.2.7 Další funkce BDD nástrojů

Některé z nástrojů umí ještě další nadstandardní funkce, většinou pomocí dalších doplňků. Nástroj Concordion, stejně jako SpecFlow, umí pomoci se správou produktového backlogu. Pro nástroj SpecFlow ještě existuje doplňek Pickles, který umí generovat podrobnější dokumentaci, a integrovat do ní například výsledky testů.

6.3 Výběr vhodného BDD nástroje

Většina nástrojů poskytuje zhruba stejnou základní funkcionalitu, většina z nich využívá syntaxi Gherkin pro psaní spustitelných scénářů v přirozeném jazyce. V první řadě je při výběru vhodného softwarového nástroje pro Behaviour Driven Development samozřejmě třeba zohlednit programovací jazyk. Pokud je pro daný jazyk nástrojů k dispozici více, může být dobrým důvodem, proč vybrat určitý nástroj, třeba uživatelská komunita. Dále je ale nutné vzít v úvahu i specifické požadavky na nástroj, protože právě v nich se jednotlivé nástroje zásadně liší.

Pokud má Behaviour Driven Development řešit určité problémy musí i nástroj pro implementaci podle Behaviour Driven Development reflektovat tyto cíle. Cíle nasazení BDD jsou, popsané v kapitole 5.4:

- Formální dokumentace testování
- Detailní dokumentace produktu
- Komunikace mezi vývojovým týmem a okolím

Nástroje jsou níže popsány z hlediska reflektování uvedených cílů.

6.3.1 Formální dokumentace testování

Vzhledem k potřebě dokumentace i v oblasti řízení kvality většina velkých firem používá nějaký softwarový nástroj pro správu testovacích scénářů a dokumentaci běhu jednotlivých testů. Typickým příkladem takového nástroje je HPQC. Pokud tým nasadí Behaviour Driven Development, existují nástroje, které automaticky vytvářejí reporty běhu testů – alespoň těch funkcionalit, jejichž scénáře jsou popsány v jazyce Gherkin a automatizovány. Takovýto výstup je ale pro management mnohem hodnotnější než výstup například z HPQC, protože je ve formě „scénář – výsledek“. Tzn. čitelný a srozumitelný popis části funkcionality, a

výsledek – jestli daný scénář funguje dle očekávání nebo ne. Jedná se o nejpřehlednější způsob, jak formalizovat automatizované testování a využívat ho k reportingu.

6.3.2 Detailní dokumentace projektu

Manifest agilního vývoje softwaru (Beck et al., 2001) mimo jiné říká, že upřednostňuje „fungující software před vyčerpávající dokumentací“. Metodika Scrum také nedefinuje žádné dokumenty. V korporátním prostředí jsou však většinou přesně dána pravidla na dokumentaci a povinný reporting projektu.

Pro korporátní sféru jsou tak vhodné nástroje s rozsáhlými reportingovými možnostmi – a to buď s možností vytvořit případně upravit si report do požadované podoby, nebo nabídnout rovnou v základu detailní reportingové možnosti. Zde výrazně vynikají nástroje SpecFlow a Cucumber, které nabízejí detailní reporty v atraktivní grafické podobě, včetně grafů a statistik pro jednotlivé požadavky. Což je pro management jistě čitelnější forma než textový výstup z konzole, který by naopak mohl stačit, pokud by byl nástroj použit pouze po potřeby vývojového týmu, a nebyl by potřeba dodávat žádný výstup směrem ven.

6.3.3 Komunikace mezi vývojovým týmem a okolím

Protože většina nástrojů používá jazyk Gherkin, je vhodné využít právě nástroj založený na jazyce Gherkin. Výhoda využití nástrojů využívajících Gherkin je zřejmá - syntaxe požadavků je kompatibilní napříč nástroji, jazyk Gherkin je implementován stále stejně, což je vlastně další zjednodušení pro zákazníka – i kdyby tým změnil nástroj, se kterým pracuje, nebo i samotnou technologii vývoje, zákazník požadavky definuje stále ve stejném, srozumitelném formátu.

6.4 Shrnutí

Existuje opravdu široké spektrum nástrojů pro implementaci podle Behaviour Driven Development, které vyhoví snad každému programovacímu jazyku. Téměř všechny mají společného jmenovatele – jazyk Gherkin pro definici požadavků, na kterém jsou založeny. V základních funkcích se nástroje mezi sebou příliš neliší, naopak se liší významně v přítomnosti nějakých nadstavbových funkcí nebo rozšíření. A právě ta tvoří důležitou součást benefitů, které může nasazení Behaviour Driven Development přinést. Pro korporátní sféru tak nejspíš budou důležité především nástroje, které zvládají pokročilý reporting nebo týmu pomohou s vytvořením živé, samoudržující se produktové dokumentace. Z uvedeného

srovnání tak vyplývá, že tyto pokročilé funkcionality mají pouze dva nástroje: Cucumber a SpecFlow. Z nich každý je vytvořen pro jiný programovací jazyk.

K detailnější analýze je vybrán nástroj určený pro .NET, SpecFlow. Jedná se nejen o pokročilý, ale i rozšířený nástroj. Důvodem je jeho rozšíření mezi uživateli (z čehož vyplývá dobrá podpora uživatelské komunity), dostupnost oficiálních školení a volitelných doplňků, a integrace do vývojového prostředí Microsoft Visual Studio. Což je kombinace vlastností, kterými získává výraznou výhodu a může být silným nástrojem pro korporátní sféru.

7 Nástroj SpecFlow

Kapitola věnovaná nástroji SpecFlow zahrnuje vše, co je potřeba do začátku testování se SpecFlow. Od popisu jeho součástí, instalace, a pak samotné práce s ním. V kapitole je vytvořen jednoduchý testovací projekt pro testování webových aplikací, na kterém jsou vytvořeny praktické ukázky implementace testovacích scénářů pomocí SpecFlow. Kapitola tak může sloužit jako vodítko k orientaci v nástroji a pomůže v prvních krocích při automatizaci testovacích scénářů ve stylu Behaviour Driven Development.

Praktické příklady jsou provedeny ve vývojovém prostředí Microsoft Visual Studio pomocí programovacího jazyka C# z důvodu možnosti případného pozdějšího využití na projektu, kde autorka působí jako tester. Konkrétní implementace se však v jednotlivých jazycích a nástrojích příliš neliší, důkazem může být právě to, že autorka hojně čerpá ze zdrojů využívajících jazyk Ruby a jeho nástroje pro BDD, Cucumber. Jako výchozí bod se hodí například kniha Cucumber Recipes, kde je popsána instalace, vytvoření prvního projektu, a základní kroky se SpecFlow, vše ale v angličtině.

Spolu s nástrojem SpecFlow je ve vytvořeném testovacím projektu použit nástroj Selenium Webdriver sloužící k interakci s libovolnou webovou aplikací. Selenium ale není vůbec nutné použít, existují i jiné nástroje pro testování grafického webového rozhraní. A hlavně, scénáře automatizované pomocí SpecFlow se nemusí vůbec týkat interakce s webovým rozhraním.

7.1 Základní informace o nástroji SpecFlow

Vývoj SpecFlow začal v roce 2009 zaměstnanci firmy TechTalk. Nyní je volně k dispozici jako otevřený (tzv. open source) projekt. SpecFlow je implementace Cucumber, využívá oficiální parser pro jazyk Gherkin a nabízí plnou integraci pro framework Microsoft .NET, Silverlight, Windows Phone a open source prlatformu pro .NET, Mono. Distribuováno je SpecFlow pomocí NuGet balíčků pro Microsoft Visual Studio nebo pomocí repositářů pro Mono. (TechTalk, nedatováno) V kontextu korporátní sféry je tak zajímavá zejména integrace do Microsoft Visual Studia.

Jak bylo výše uvedeno, samotné SpecFlow je dostupné jako open source, je tedy k použití zdarma. K dispozici je však několik doplňků: SpecFlow+Runner, SpecLog a Pickles. První dva jmenované už nejsou zdarma, ale nabízejí zajímavé funkce: report běhu testů, použití

s integračním serverem, a správa produktového backlogu. Poslední jmenovaný je doplněk vytvořený třetí stranou, avšak přímo pro SpecFlow, a je opět zdarma. Pickles vytváří živou dokumentaci produktu – například ve formátu HTML (TechTalk, nedatováno). Právě tyto doplňky jsou silnou stránkou SpecFlow. Samotné SpecFlow je srovnatelné s ostatními nástroji pro BDD (stejně tak je zdarma), celý ekosystém ale tvoří výrazně výkonnější celek vhodný například díky reportingu právě pro korporátní prostředí.

7.2 Instalace a konfigurace SpecFlow

Instalace SpecFlow je jednoduchá zejména díky jednoduché instalaci Windows NuGet balíčků. Samotné SpecFlow je reprezentováno pouze jedním balíčkem – TechTalk.SpecFlow. Kromě této knihovny je potřeba libovolný spouštěč unit testů. SpecFlow je v tomto směru konfigurovatelné, nabízí možnost využití MsTest nebo NUnit. Má ale i vlastní spouštěč testů, SpecRun, který nabízí přidané funkcionality – například v podobě reportingu. Pokud chce uživatel tedy i reporting, což je jednou z hlavních silných stránek SpecFlow a odlišuje ho od konkurence, je třeba referencovat ještě druhý balíček, TechTalk.Specflow+Runner. SpecFlow+Runner je nakonfigurován jako spouštěč testů ve výchozím nastavení, v případě použití jiného spouštěče testů se toto nastavuje parametrem „unitTestProvider“ v konfiguračním souboru aplikace (App.config), který je vidět ve Výpis 3.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="specFlow"
type="TechTalk.SpecFlow.Configuration.ConfigurationSectionHandler,
TechTalk.SpecFlow" />
    </configSections>
  <specFlow>
    <language feature="cs-CZ"></language>
    <runtime stopAtFirstError="false"></runtime>
   <unitTestProvider name="SpecRun" />
     <plugins>
   <add name="SpecRun" />
    </plugins>
</specFlow>
</configuration>
```

Výpis 3, Konfigurační soubor nástroje SpecFlow

Stejně tak se v konfiguračním souboru parametrem „language“ nastavuje jazyková mutace Gherkin. V případě Výpisu 3 je SpecFlow nastaveno, aby používalo český překlad jazyka Gherkin, tedy česká klíčová slova. V takovém nastavení SpecFlow nerozpozná originální

klíčová slova (Given, When, Then), bohužel s nimi vytváří šablonu souboru .feature, nezávisle na nastaveném jazyce. Vznikne tak šablona, kterou SpecFlow nerozpozná. Přepsáním klíčových slov do nastaveného jazyka je ale problém odstraněn.

Jak bylo výše zmíněno, SpecFlow vygeneruje šablonu pro soubor .feature, do kterého uživatel poté dopíše svůj požadavek, konkrétní scénáře a příklady. Součástí šablony je ukázkový požadavek s ukázkovým scénářem, uživatel je tak veden vytvářením svého požadavku, struktura je naprosto zřejmá i bez předchozí znalosti jazyka Gherkin.

Další součást, která je potřeba, je soubor s kroky samotné automatizace scénářů, třída Steps. Vytvořením třídy Steps je uživatel také veden, SpecFlow nabzízí vygenerování jakési kostry, kterou uživatel poté vyplní jednotlivými kroky. Nemusí ale řešit například, jak propojit soubor definice požadavků se souborem exekuce testu, o to se stará samotné SpecFlow.

7.3 Definice požadavků

Požadavky jsou v rámci nástroje SpecFlow popsány souborem .feature. Jedná se o textový soubor, který je pojmenován podle požadavku, s příponou .feature. Pro každý požadavek je vytvořen nový soubor. V souboru jsou přirozeným jazykem popsány funkcionality aplikace a jejich scénáře, tedy konkrétní případy užití. Každá funkcionalita má samostatný soubor, k jedné funkcionalitě se ale může vázat více scénářů. Jednotlivé scénáře obsahují prerekvizity (GIVEN), kroky, které mají být vykonány (WHEN), a jejich očekávaný výsledek (THEN). Pro jednoduchou funkcionalitu „Google vyhledávání“ soubor .feature se čtyřmi scénáři může vypadat jako je vidět ve Výpis 44.

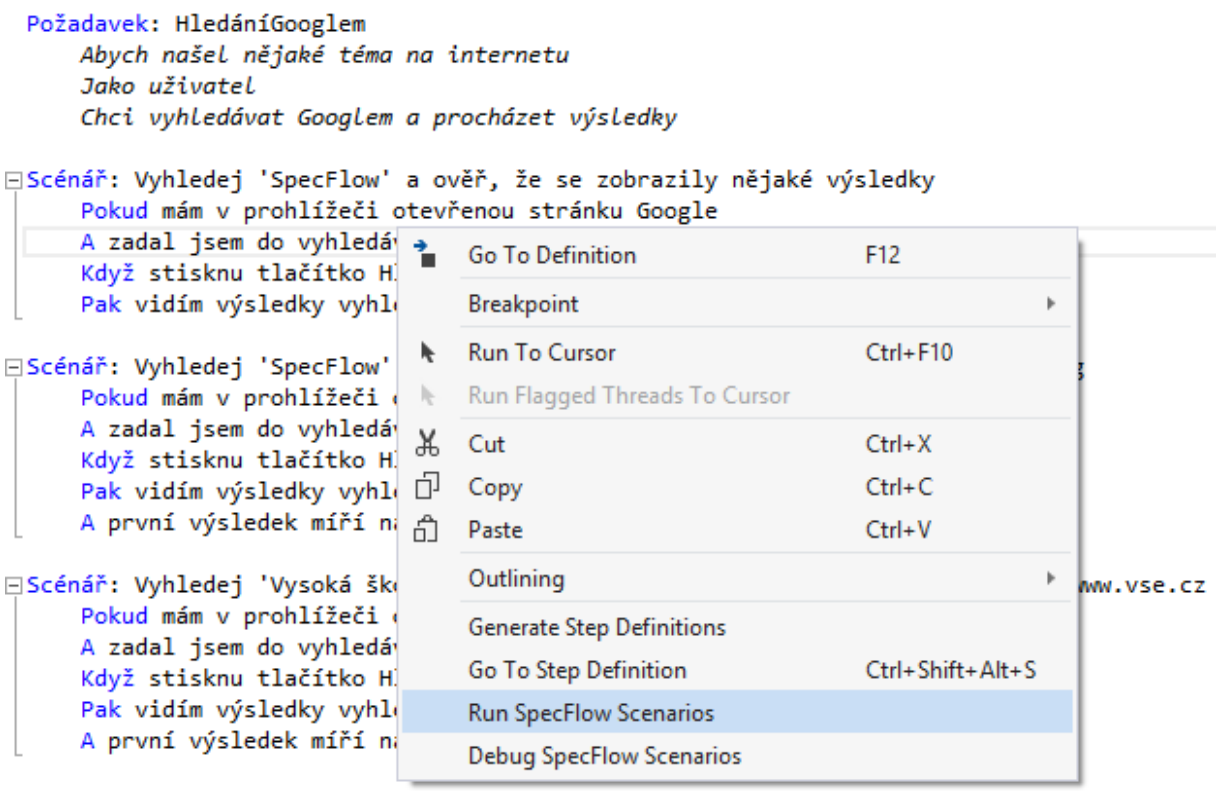
Jak je patrné z Výpis 4, přestože je zkopírovaný z testovacího projektu otevřeného ve vývojovém prostředí (Microsoft Visual Studio), jeho struktura spíše než třídu zdrojového kódu připomíná přirozený text. Pokud by nebyla klíčová slova v souboru automaticky zvýrazňována, ztratila by se přirozeně ve větě. Což je důkaz pro stále velkou přirozenost jazyka Gherkin. Takováto definice požadavků tedy zůstává pro uživatele stále jednoduchá – možná díky nutnosti dodržení stručnosti a přesnosti dané potřebou strukturovat jednotlivé scénáře, ještě jednodušší, než popis funkcionalit stránkami textu. Ale pro vývojový tým takto strukturovaná definice přestává být prázdným a vágním požadavkem, který se ztrácí v textu. Stává se požadavkem, který, jak je vidět na Obrázek 64, lze jednoduše navázat na testovací třídu a spustit jako test. Scénáře jsou pro přehlednost očíslovány, není to ale podmínkou.

Ve scénáři 1 je vidět základní struktura Pokud, Když, Prak, rozšířená o jeden krok A ve výchozích předpokladech. Scénáře 2 a 3 jsou dále rozšířeny o další krok A v očekávaných výsledcích. Jak je dále vidět, scénáře 2 a 3 mají totožnou nejen strukturu, ale mají i velice podobný obsah. Liší se jen použitým vyhledávacím výrazem a výsledkem vyhledávání. Z pohledu byznysu může jít o zcela odlišné funkcionality, z pohledu vývojáře ale jde o použití totožné metody: kroky, které vedou k vyhledání výrazů SpecFlow a Vysoká škola Ekonomická jsou otevřená stránka Google, napsaná vyhledávací fráze a stisknutí tlačítka „Hledat Googlem“. Vyhledávací fráze je tedy parametrem. Analogicky je parametrem i výsledek vyhledávání. Parametr je zapisován jednoduše a přirozeně – obklopením jednoduchými uvozovkami, jak je vidět ve Výpisu 4.

```
Požadavek: Hledání Googlem
       Abych našel nějaké téma na internetu
       Jako uživatel
       Chci vyhledávat Googlem a procházet výsledky

Scénář: 1: Vyhledej 'SpecFlow' a ověř, že se zobrazily nějaké výsledky
       Pokud mám v prohlížeči otevřenou stránku Google
       A zadal jsem do vyhledávacího řádku 'SpecFlow'
       Když stisknu tlačítko Hledat Googlem
       Pak vidím výsledky vyhledávání

Scénář: 2: Vyhledej 'SpecFlow' a ověř, že první výsledek míří na www.specflow.org
       Pokud mám v prohlížeči otevřenou stránku Google
       A zadal jsem do vyhledávacího řádku 'SpecFlow'
       Když stisknu tlačítko Hledat Googlem
       Pak vidím výsledky vyhledávání
       A první výsledek míří na stránku 'www.specflow.org'

Scénář: 3: Vyhledej 'Vysoká škola Ekonomická' a ověř, že první výsledek míří na
www.vse.cz
       Pokud mám v prohlížeči otevřenou stránku Google
       A zadal jsem do vyhledávacího řádku 'Vysoká škola Ekonomická'
       Když stisknu tlačítko Hledat Googlem
       Pak vidím výsledky vyhledávání
       A první výsledek míří na stránku 'www.vse.cz'
```

Výpis 4, soubor HledáníGooglem.feature

Důležité na scénářích je to, že jdou přímo klepnutím spustit, není potřeba složitě dohledávat sadu automatizovaných testů pro danou funkcionalitu (Obrázek 6). Pro zainteresovanou osobu se tak mění „proběhlá sada automatizovaných testů, které snad něco dělají“ na „jsou otestovány funkcionality aplikace“. Anebo ještě lépe, na „moje požadavky jsou otestovány“. Zainteresovaným stranám použití srozumitelných scénářů i pro automatizované testování funkcionalit dává možnost lepšího vhledu do aktuální fáze vývoje produktu.



Obrázek 6: Spustitelnost jednotlivých scénářů. Zdroj: Autorka

7.4 Třída Steps

Třída Steps je druhou důležitou součástí SpecFlow. Obsahuje definice kroků, tedy metody, kterými se jednotlivé kroky vykonávají. Každý krok popsaný v souboru .feature k sobě má jednoznačně navázanou jednu metodu z třídy steps. Jeden krok má právě jednu metodu třídy Steps, ale jedna metoda třídy Steps může být přiřazena k více krokům.

Typickým příkladem situace, kdy jedna metoda může mít přiřazeno více kroků, je parametrizace. Scénáře „Vyhledej Vysoká škola Ekonomická a ověř, že první výsledek míří na www.vse.cz“ a „Vyhledej SpecFlow a ověř, že první výsledek míří na www.specflow.org“

Třída GoogleSearchSteps, tedy definice metod pro test požadavku Vyhledávání v Google může vypadat jako ve Výpis 5.

```
[Binding]
public class GoogleSearchSteps : TestBase
{
 public static string googleUrl = "http://www.google.com";

 [Given(@"mám v prohlížeči otevřenou stránku Google")]
 public void PokudMamVProhlizeciOtevrenouStrankuGoogle()
   {
    googlePage.GoToUrl(googleUrl);
   }

 [Given(@"zadal jsem do vyhledávacího řádku '(.*)'")]
 public void PokudZadalJsemDoVyhledavacihoRadku(string dotazHledani)
   {
    googlePage.EnterSearchExpression(dotazHledani);
   }

 [When(@"stisknu tlačítko Hledat Googlem")]
 public void KdyzStisknuTlacitkoHledatGooglem()
   {
    googlePage.Search();
   }

 [Then(@"vidím výsledky vyhledávání")]
 public void PakVidimVysledkyVyhledavani()
   {
    Assert.IsNotNull(googlePage.GetResults());
   }

 [Then(@"první výsledek míří na stránku '(.*)'")]
 public void PakPrvniVysledekMiriNaStranku(string vysledekHledani)
  {
   googlePage.NavigateToNthResult(0);
        Assert.IsTrue(googlePage.GetActualUrl().Contains(vysledekHledani));
        }
```

Výpis 5, Třída Steps

Atribut [Binding] v hlavičce říká, že se jedná o třídu, která je *navázána* na nějaký scénář. Atributy [Given], [When] a [Then] říkají, že se jedná o metodu, která je metodou definice kroku, a označují spolu s textovým řetězcem krok, na který má být daná metoda navázaná. Ve Výpis 5 je vidět, že přestože je SpecFlow nakonfigurováno do češtiny, atributy třídy Steps reflektují originální, tj. anglická klíčová slova. Proto je vždy důležité znát nejen klíčová slova v nakonfigurovaném jazyce, ale i jejich originální znění.

K provázání se scénářem v souboru .feature dochází automaticky, pomocí příkazu Generate Step Definitions (*generovat definici kroku,* viz Obrázek 6) nad konkrétním scénářem je vytvořena šablona pro definici kroku, včetně reference mezi krokem a scénářem. Na vývojáři je tedy doplnění metody tak, aby odpovídala kroku. Například metoda definující krok k*dyž*

stisknu tlačítlo Hledat v uvedeném příkladu volá metodu *googlePage.Search();*, což je metoda pro stisknutí tlačítka hledat na předem vytvořeném objektu stránky Google.

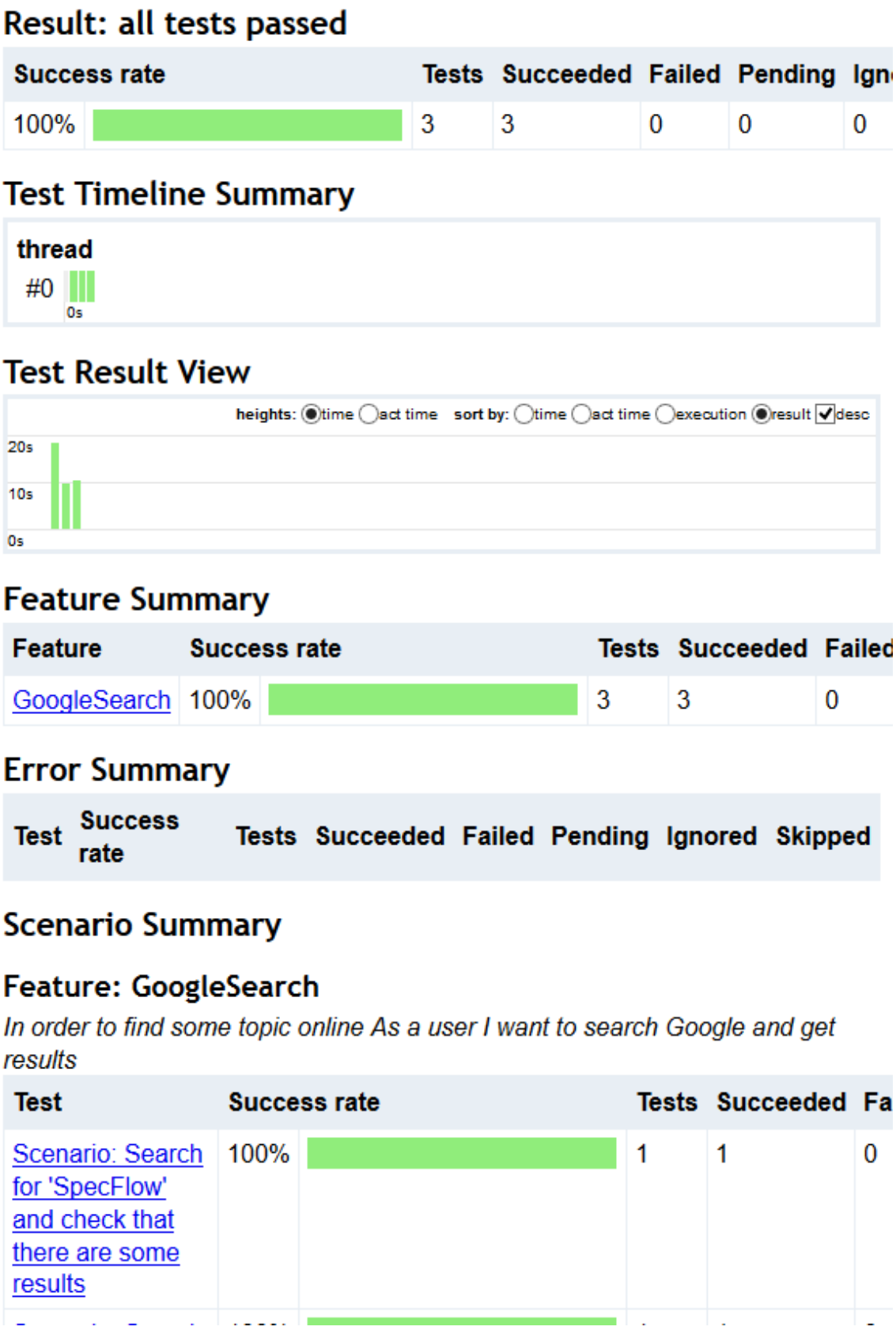
Každá definice kroku z třídy Steps je provázána s nějakým krokem definovaným v souboru .feature a naopak. Jedna definice kroku z třídy Steps může být navázána i k více krokům ve více scénářích, nejběžněji se tohoto využívá při parametrizaci kroků. Znovupoužitelnost definic kroků se ale velmi hodí i v případě, kdy z uživatelského pohledu znějí dva kroky odlišně, technicky ale dělají totéž. Parametrizace kroků je vidět na scénářích „Vyhledej ‚SpecFlow' a ověř, že první výsledek míří na www.specflow.org." a „Vyhledej ‚Vysoká škola Ekonomická' a ověř, že první výsledek míří na www.specflow.org". Z funkčního pohledu se jedná o totožný krok: vložení textového řetězce do určitého pole, a stisknutí určitého tlačítka. Proto je i metoda obou těchto kroků stejná, liší se pouze parametrem, který je reprezentován regulárním výrazem. Parametr je ze souboru .feature rozpoznán automaticky, stačí parametr dát do jednoduchých uvozovek. Metoda pro oba uvedené scénáře je znázorněna ve Výpis 6.

```
[Given(@"zadal jsem do vyhledávacího řádku '(.*)'")]
 public void PokudZadalJsemDoVyhledavacihoRadku(string dotazHledani)
 {
  googlePage.EnterSearchExpression(dotazHledani);
 }
```

Výpis 6, Krok s parametrem

7.5 Reporting a živá dokumentace

Jak bylo výše zmíněno, spouštěč testů speciálně vyvinutý pro SpecFlow, SpecFlow+Runner zahrnuje i reportovací funkci. Což v praxi vypadá tak, že po každém běhu určité sady testů jsou v příslušné složce (<adresář projektu>\ TestResults) vytvořeny dva soubory. Jeden ve formátu .txt, druhý ve formátu .html. V praxi bude nejspíš využitelnější druhý z uvedených, protože má pro uživatele větší vypovídací hodnotu. Jsou v něm uvedeny konkrétní proběhlé scénáře, statistiky jednotlivých funkcionalit a mnoho dalších informací v přehledné grafické formě. Je zřejmé, že tento typ reportu cílí na management a stakeholdery. Ukázka reportu je vidět na Obrázek 7.



Obrázek 7: Grafický report SpecFlow. Zdroj: Autorka

Pro generování živé dokumentace existuje doplněk SpecFlow, nástroj Pickles. Pickles je distribuován pomocí NuGet balíčku, opět je tedy jednoduchá integrace do Microsoft Visual Studia. Pickles je dostupný pod OpenSource licencí, je tedy zdarma. Pickles umožňuje generovat dokumentaci z požadavků a scénářů psaných v jazyce Gherkin, ze souboru .feature.

Pickles umí, kromě samotného generování dokumentace, přidat i výsledky posledního běhu testů. Tím vlastně nahrazuje a rozšiřuje reporting SpecFlow+Runner. Takže je možné používat alternativní spouštěč testů místo SpecFlow+Runner, například NUnit (jednoduchá změna v konfiguračním souboru – viz výše), a je tak možné SpecFlow využívat kompletně zdarma, v rámci OpenSource licence.

Nástroj Pickles je také možné přímo používat na integračním serveru, dokumentace je tak automaticky generována při každém sestavení. Tím je zaručeno, že dokumentace vždy reflektuje aktuální stav, včetně výstupu automatizovaných testů.

7.6 Shrnutí

Práce s nástrojem SpecFlow je jednoduchá a poměrně intuitivní od instalace a úvodního nastavení (tam raději pomůže dokumentace) po implementaci scénářů. Nástroj uživatele vede od prvních kroků a pomáhá vytvořenými šablonami a kostrami. Jediný nedostatek je generování šablon souboru .feature v angličtině, přestože je nakonfigurováno použití jiného jazyka. To je jediný matoucí moment, kdy si SpecFlow vytvoří soubor, jehož klíčová slova nerozezná. Jinak je integrace jazyka Gherkin bezchybná a práce se scénáři a přiřazenými kroky k nim jednoduchá.

Soubor .feature je editovatelný i mimo vývojové prostředí, například v Poznámkovém bloku, což zase posouvá jeho využitelnost. Akceptační kritéria k jednotlivým user stories tak mohou být přímo definována souborem .feaure, který uživatel může jednoduše vytvořit.

Hlavní síla SpecFlow spočívá v rozsáhlé uživatelské komunitě, a hlavně v množství volitelného rozšíření. SpecFlow je celý *ekosystém* produktů, které mají dohromady větší sílu než ostatní nástroje, hlavně v kontextu korporátní sféry: propojení s integračním serverem přináší neustálý přehled o stavu produktu, stejně jako přehledný reporting, pokročilá správa produktového backlogu nebo automatické vytváření živé dokumentace produktu.

8 Návrh metodiky ScrumFlow

ScrumFlow je autorskou metodikou vzniklou jako výstup hlavního cíle diplomové práce. ScrumFlow je integrací Behaviour Driven Development do metodiky Scrum. Přívlastek *Flow* může asociovat známý nástroj pro Behaviour Driven Development, SpecFlow. Má ale také znázorňovat *flow,* tedy jakýsi proud, tok, v tomto případě informací. Metodika ScrumFlow si klade za cíl zlepšení a zjednodušení komunikace mezi vývojovým týmem a byznysem.

Z charakteristiky metodiky Scrum je zřejmé, že je v rámci ní možné, a autory doporučené, používat další procesy, postupy nebo techniky. Metodika ScrumFlow tedy vychází z metodiky Scrum a je jejím rozšířením o Behaviour Driven Development. Z charakteristik obou přístupů v předchozích kapitolách je zřejmé, že se jejich kombinování nijak zvlášť nevylučuje. Dokonce mezi nimi lze najít styčný bod – použití požadavků ve struktuře *user stories*, které mohou vzájemné integraci výrazně pomoci. Metodika ScrumFlow zahrnuje všechny prvky metodiky Scrum, kterou rozšiřuje, níže jsou popsána všechna navržená rozšíření.

8.1 Určení ScrumFlow

Metodika ScrumFlow najde uplatnění na jakémkoliv projektu, kde lze nasadit Scrum. Primárně ale cílí na využití ve velkých podnicích, kde je nutné dodržovat určité standardy z hlediska reportingu, a je zde složitější komunikace.

Metodika ScrumFlow se snaží zjednodušit komunikaci mezi byznysem a vývojovým týmem, a poskytovat jednodušší dokumentaci pro management.

8.2 Činnosti ScrumFlow

Metodika ScrumFlow nedefinuje nové činnosti. Činnosti metodiky ScrumFlow jsou tak identické s činnostmi metodiky Scrum, které jsou podrobně popsány v kapitole 164.3.

8.3 Artefakty ScrumFlow

Metodika ScrumFlow definuje oproti metodice Scrum čtyři nové artefakty. Tyto přidané artefakty metodikou ScrumFlow jsou

- Spustitelný scénář požadavku,
- Report testů požadavku,

- Požadavek popsaný v jazyce Gherkin,
- Živá dokumentace.

Ostatní artefakty jsou stejné jako u metodiky Scrum, jejich detailní popis je v kapitole 4.4.

Požadavek popsaný v jazyce Gherkin je základní artefakt ScrumFlow, od kterého se odvíjí ostatní. Každý požadavek má být ve ScrumFlow důsledně zaznamenán v jazyce Gherkin ve formě, která byla výše popsána. Pomůže k tomu šablona popisu požadavků, která je jedním ze vzorů dokumentů metodiky ScrumFlow.

Spustitelný scénář požadavku je takový scénář, který je zapsán v jazyce Gherkin, ale už jsou k němu navázány potřebné metody a funguje jako spustitelná specifikace nebo akceptační test, chceme-li. Spustitelný scénář požadavku vychází z požadavku v jazyce Gherkin a je základem pro vytváření dokumentace, sady regresní testů a reporting.

Report běhu testů požadavku staví na spustitelném scénáři. Nástroj SpecFlow (případně Cucumber) umí automaticky generovat přehledné reporty z každého spuštění sady testů. Stačí tedy použít nějaký z nástrojů, který umí report generovat, a není potřeba další významné aktivity týmu.

Živá dokumentace je v podstatě soubor všech požadavků popsaných v jazyce Gherkin. Už to se dá považovat za živou dokumentaci. Lepší forma ale je generování dokumentace z těchto požadavků – například pomocí výše popsaného nástroje Pickles. Ten umí do jednoho dokumentu integrovat výsledky testů a popisy požadavků. Opět je ale výsledný dokument ve formě, která je čitelná pro management, a protože je automaticky generován, tým nemá s dokumentací přidanou práci.

8.4 Role ScrumFlow

Jedním z rozšíření ScrumFlow jsou dvě dodatečné role: Vývojář softwaru a Tester softwaru. Obě role náleží do Scrum týmu. Metodika ScrumFlow tedy obsahuje následující role:

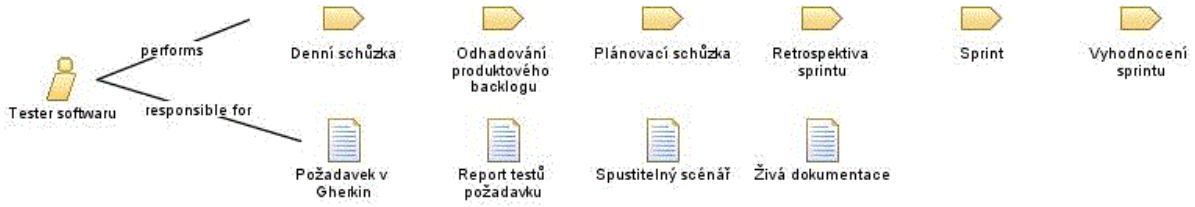
- Vlastník produktu
- Scrum Master
- Scrum tým
- Vývojář softwaru
- Tester softwaru.

Role metodiky Scrum jsou popsány v kapitole 4.2. Přestože originální metodika Scrum nedefinuje specifické role ve vývojovém týmu, Linz (2014) doporučuje mít v týmu členy s dostatečnými schopnostmi a praxí v oblasti testování. Ostatní specializace se vždy budou odvíjet od konkrétního produktu a technologií použitých pro jeho vývoj, testování je ale potřeba vždy, nezávisle na technologii nebo způsobu vývoje. Specializovaný tester má o to větší odůvodnění v případě použití konkrétních, případně složitějších, testovacích praktik.

V praxi ke specializaci těchto dvou rolí většinou dochází. Přestože tyto role nyní formalizujeme, tato specializace ve ScrumFlow ale neznamená, že Vývojář nemůže testovat. V pojetí metodiky ScrumFlow mají Tester i Vývojář velmi podobný objem znalostí – od znalosti programovacího jazyka přes například UML nebo SQL. U Testera jsou však dominující testovací techniky, u Vývojáře naopak znalost programovacího jazyka. Ale v případě nedostatku kapacit na jedné straně druhá strana pomáhá. V praxi je v týmu většina Vývojářů a jen jeden, nebo několik málo Testerů. Vývojář tak v případě potřeby vypomáhá s testováním a Tester funguje v podstatě v roli tradičního test manažera – zastává funkci autority, která na základě svých zkušeností s testováním rozhoduje o rozsahu a způsobu testování.

Všichni členové týmu, včetně Vývojáře softwaru a Testera softwaru se účastní činností definovaných metodikou Scrum, tedy schůzek, odhadů produktového backlogu a podobně. Níže jsou popsané pouze činnosti specifické pro ScrumFlow, tedy rozšiřující metodiku Scrum.

8.4.1 Tester softwaru



Obrázek 8: Přehled role Tester softwaru. Zdroj: Autorka

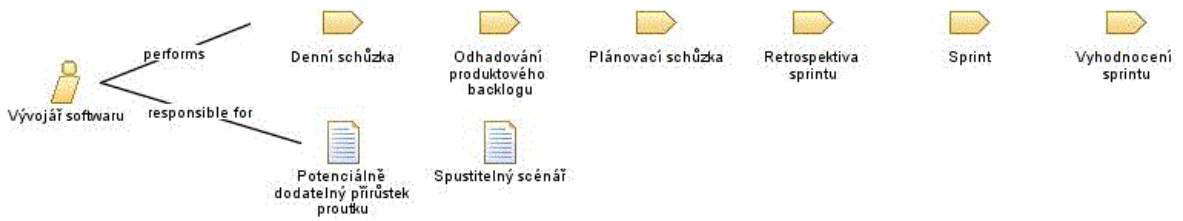
Jak je vidět na obrázku 8, Tester sofwaru se účastní téměř všech činností. Tester je součástí Scrum týmu, takže je přítomen na denních i plánovacích schůzkách, retrospektivě, plánování, i vyhodnocení. Je zodpovědný za artefakty Požadavek v Gherkin, Spustitelný scénář, Report

testů požadavku a Živá dokumentace. Neznamená to, že by všechny tyto artefakty nutně musel vytvářet Tester: Požadavek v Gherkin vytváří ve spolupráci se zákazníkem, případně ho může celý vytvořit zákazník. Tester je ale výstupní kontrolou, že je požadavek zapsán syntakticky správně, srozumitelně, a obsahuje alespoň jeden scénář. Spustitelný scénář, tedy automatizaci scénáře, může vytvářet buď Tester, nebo Vývojář – nejčastěji na Spustitelném scénáři nejspíš budou spolupracovat oba. Report testů požadavku je generován automaticky, Tester je ale osoba zodpovědná například za to, že Report je v pořádku. Případně, že probíhá manuální testování, doplňuje Tester Report o výsledky manuálního testování. Živá dokumentace je také generována automaticky – z definice požadavků. Vzhledem k tomu, že je Tester zodpovědný za podobu definice požadavků, je tak v podstatě zodpovědný i za Živou dokumentaci.

To, že je v týmu Tester, tedy specialista na testování softwaru, neznamená, že Vývojář nemůže testovat. Testerova hlavní odpovědnost je definice rozsahu a způsobu testování, tedy v podstatě role test manažera v tradičním pojetí. Pokud je ale nedostatečná kapacita na straně testování, Vývojář se snaží pomoci, protože neotestovaná funkcionalita znamená nedokončený požadavek, za který je vždy zodpovědný celý tým, ne pouze tester nebo naopak vývojář.

8.4.2 Vývojář softwaru

Vývojář softwaru je člen Scrum týmu, jehož primární náplní práce je samotný vývoj. Může se jednat o programátora nebo například databázového specialistu – vývojář je jednoduše člověk, jehož zodpovědností je dodávat časté přírůstky k testování, a stejně jako u ostatních členů týmu, mít na konci sprintu potenciálně dodatelný přírůstek produktu. Kromě toho je Vývojář softwaru zodpovědný za artefakt Spustitelný scénář.



Obrázek 9: Přehled role Vývojář softwaru. Zdroj: Autorka

Vývojář dostane od Testera požadavky a scénáře zapsané v jazyce Gherkin, a jeho zodpovědností je scénáře automatizovat – vytvořit z nich automatizovaná akceptační kritéria. Nemusí nutně celou automatizaci dělat sám, někdy může stačit pomoc Testerovi s automatizací. Jak bylo uvedeno výše, nejčastěji na spustitelných scénářích budou Tester s Vývojářem spolupracovat. Vývojáři automatizace scénáře pomůže pochopit funkcionalitu, kterou má vyvíjet. Jak je vidět na Obrázek 9, i Vývojář softwaru se účastní stejných schůzek a činností jako zbytek ScrumFlow týmu.

8.5

8.5 Doporučení a vzory dokumentů

Metodika Scrum definuje jeden vzor dokumentu a jedno doporučení.

Ukázka popisu požadavků je konkrétní příklad definice požadavku a jeho scénářů, který může tým použít pro orientaci v jazyce Gherkin a způsobu, jak vytvářet požadavky. Ukázka popisu požadavků je soubor ve formátu .feature, který je možné použít a spustit.

Šablona popisu požadavku již obsahuje pouze klíčová slova ve správné struktuře, a základní jednoduchá vodítka, jak šablonu vyplnit, viz Výpis 7. Šablona popisu požadavků je soubor ve formátu .feature, který je po doplnění možné použít a spustit.

```
Požadavek: <Výstižný název požadavku>
       <Popis požadavku ve formátu jako User story – Jako uživatel…>

Scénář: <Název scénáře>
      Pokud <výchozí stav>
      (A <výchozí stav>)
      Když <podmínka, akce vyslaná směrem k systému>
      Pak <očekávaný výsledek>
      (A <očekávaný výsledek>)

Scénář: <Název scénáře>
      (...)
```

Výpis 7, Šablona popisu požadavku

8.6 Implementace metodiky ScrumFlow

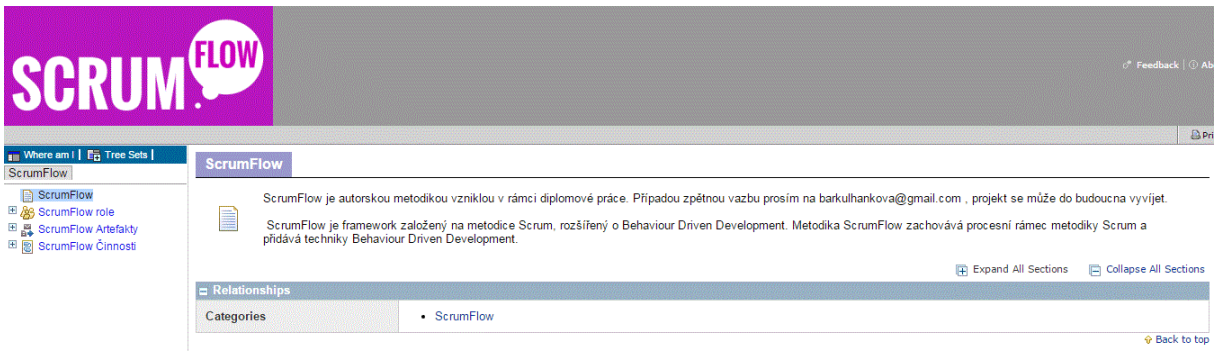
Pro možnost publikace, a tedy i šíření metodiky ScrumFlow a jejího případného dalšího rozvoje je metodika implementována v nástroji Eclipse Process Framework Composer, a publikovaná online.

8.6.1 Eclipse Process Framework Composer

Peter Haumer (2007), charakterizuje Eclipse Process Framework Composer (dále EPFC) jako nástroj, platformu, pro procesní inženýry, projektové vedoucí a všechny manažery, kteří jsou zodpovědní za implementaci procesů pro vývoj softwaru. EPFC je tak nástrojem, ve kterém lze spravovat (a poté publikovat) metodiky a jiné procesní rámce, nezávisle na konkrétním zvoleném přístupu k vývoji. Nástroj umožňuje buď vytvářet zcela vlastní procesní rámce a metodiky nebo otevřít, prohlížet a upravovat již existující knihovny. Eclipse Process Framework Composer je dostupný zdarma, stejně jako několik knihoven metodik.

8.6.2 ScrumFlow v Eclipse Process Framework Composer

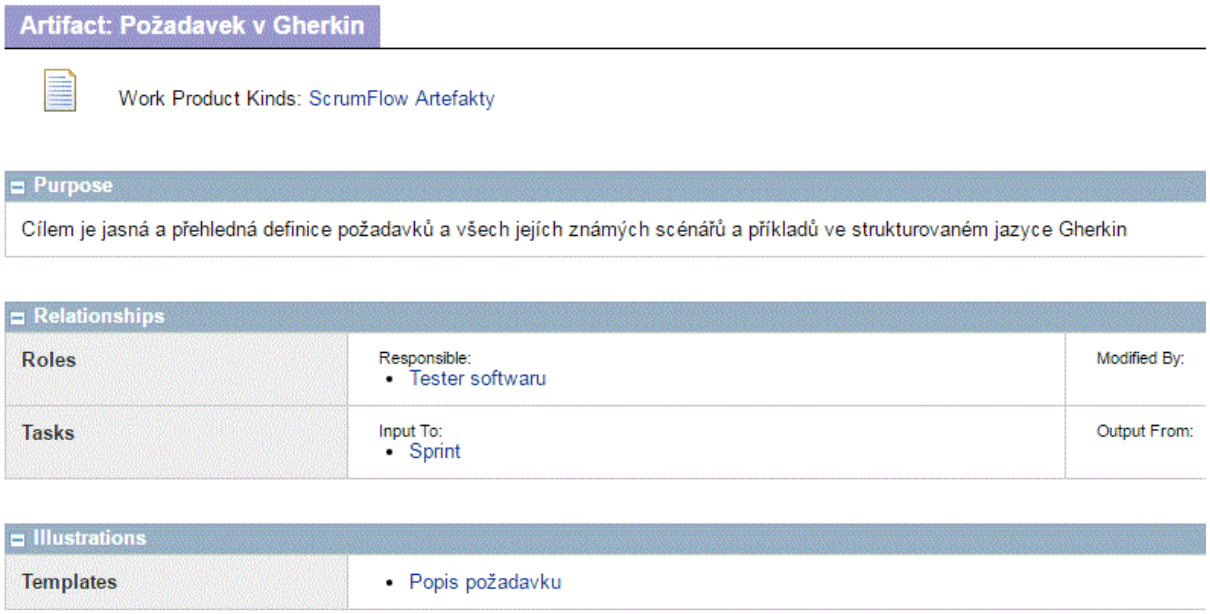
Základem implementace metodiky ScrumFlow je opět metodika Scrum, tedy knihovna metodiky Scrum stažená ze stránek Eclipse. Její implementace ale nebyla doposud přeložena do češtiny, autorkou byl tedy v souladu s Průvodcem Scrumem (Schwaber a Sutherland, 2013) proveden překlad, plus rozšíření popsané výše, čímž vznikla implementace metodiky ScrumFlow. Balíček metodiky ScrumFlow je autorkou publikován a metodika je tak dostupná zdarma na adrese http://scrumflow.czweb.org/. Na webu lze metodiku jednoduše prohlížet, viz Obrázek 10.



Obrázek 10: ScrumFlow online. Zdroj: Autorka

V konfiguraci, která je publikována na webovou stránku, jsou vytvořeny tři přehledy: role, artefakty a činnosti metodiky ScrumFlow. Mezi jednotlivými prvky metodiky jsou vytvořené reference, což umožňuje jednoduché procházení mezi těmi, které s sebou navzájem souvisí, viz Obrázek 11. Na něm je znázorněn jeden z artefaktů ScrumFlow, Požadavek v jazyce Gherkin. Za něj je zodpovědná role Tester softwaru, na kterou vede odkaz, a je vstupem činnosti Sprint, na kterou také vede odkaz. Dále k danému artefaktu náleží šablona – Popis

požadavku, na kterou je ze stránky také odkazováno. Související prvky jsou tak lehce dohledatelné.



Obrázek 11: Reference mezi prvky metodiky. Zdroj: Autorka

8.7 Očekávané přínosy ScrumFlow

Metodika Scrum je v poslední dobou velmi rozšířeným přístupem k vývoji softwaru i napříč odvětvími, která mají k agilním postupům daleko, příkladem budiž bankovní sektor. Snaha o skloubení agilního vývoje na jedné straně a potřeby pevné struktury a tradiční projektové dokumentace na straně druhé vede logicky k určité míře nepochopení a diskomfortu obou zúčastněných stran. Agilní přístup se snaží zapojit zákazníka v co nejranější í možné fázi do vývoje – jako zpětnou vazbu. Agilní přístup obecně o maximální možnou automatizaci testů. Na druhou stranu, bankovní prostředí vyžaduje mnoho reportů například z HPQC o výsledcích spuštěných testovacích scénářů, statistiky prošlých a neprošlých scénářů v jednotlivých iteracích, apod. Jednoduše, zákazník, zejména pokud má na produktu finanční zájem, chce vidět, že vývoj postupuje, a slibované funkcionality byly dostatečně otestovány.

Rozšířením metodiky o techniky metodiky Behaviour Driven Development je očekáván přínos v oblasti transparentnosti – akceptační kritéria lze svázat s automatizovanými testy a vytvořit tak automatizované akceptační testy, které jsou ale stále čitelné a srozumitelné pro zainteresované strany, například uživatele, nebo vlastníka produktu. Automatizované testovací scénáře má smysl použít ale nejen pro akceptační testy. Mohou být přínosem i pro

samotný vývojový tým, který také rychleji vidí, co přesně je pomocí Continuous Integration automaticky testováno. A na druhou stranu – i uživateli dává smysl vidět, že například proběhl integrační test, a nově vyvíjená komponenta už spolupracuje se zbytkem systému. Jednoduše, není potřeba čekat na konec sprintu, až projde celý akceptační test, ale byznys může vidět už v jeho průběhu, že *se něco děje*. Dá se tak očekávat, že zavedení automatizovaných testovacích scénářů povede ke zvýšení transparentnosti, a díky tomu i důvěry mezi vývojovým týmem a ostatními zainteresovanými stranami.

Dále klesá časová náročnost pro testery v týmu. Většinou je nutné mít kromě sady automatizovaných testů i manuální testovací scénáře, jejichž vykonání je někde potřeba mít zdokumentované. Tento mezikrok odpadá přímou vazbou automatizovaných testů na testovací scénáře. Stačí vygenerovaný report běhu automatizovaných testů, který supluje například proběhlou instanci testu v nástroji HPQC.

8.8 Rizika ScrumFlow

Zavedení každého nového přístupu s sebou může nést určitá rizika. Níže jsou nastíněna možná rizika z oblasti Lidské zdroje, Náklady a Kvalita.

8.8.1 Lidské zdroje

Rizikem metodiky ScrumFlow jsou lidské zdroje. Testeři musejí být schopni psát si automatizované testy, anebo alespoň zapsat spolu se stranou byznysu testovací scénáře ve formátu, aby k nim developer mohl dopsat potřebnou automatizaci. Na druhou stranu, v roli testera by měl být člověk se zkušeností s testování, čímž se toto riziko eliminuje.

Obecná vyšší časová náročnost na psaní a údržbu scénářů a automatizovaných testů ale není očekávána, protože automatizované testovací scénáře z dlouhodobého hlediska čas šetří. Není k nim potřeba dodatečná dokumentace v dalším nástroji (např. HPQC), a odpadá velké množství manuálního testování, přístup BDD podporuje co největší automatizaci testů.

Dalším rizikem v oblasti lidských zdrojů je strana uživatele – byznysu. Ta musí být ochotná změnit návyky a začít specifikovat své požadavky strukturovaně. Důležité je tedy motivovat byznys vysvětlením následné transparentnosti, případně pomoci alespoň zpočátku se strukturou Gherkin. Toto riziko se dá eliminovat pouze do té míry, do jaké je daná strana ochotna spolupracovat a změnit svůj přístup.

8.8.2 Náklady

Rizika v oblasti nákladů nejsou zavedením metodiky ScrumFlow očekávána, naopak by lepší komunikací a zpětnou vazbou měly odpadnout vícenáklady způsobené implementací funkcionality, kterou uživatel nechtěl nebo jak ji uživatel nechtěl. Zpětná vazba je v podstatě nastavena dříve než je tomu běžné u jiných přístupů, a k odhalení případných nesrovnalostí dojde tím pádem také dříve. Což vede ke snížení nákladů, protože ve vývoji softwaru platí, že čím dříve je chyba odhalena, tím menší jsou náklady na její odstranění.

8.8.3 Kvalita

Nasazení každého nového přístupu s sebou nese riziko dočasné snížení kvality. Ani u metodiky ScrumFlow se tomuto riziku nedá zcela předejít, přestože je z dlouhodobého hlediska očekáváno naopak zvýšení kvality. Alespoň částečné eliminaci rizika snížení kvality ale pomůže fakt, že ScrumFlow vychází ze známé a často využívané metodiky Scrum, tedy procesní rámec zůstává stejný, a změny nejsou natolik dramatické, aby musely nutně ohrozit kvalitu dodaného produktu.

Naopak je v dlouhodobém hledisku očekáván nárůst kvality vzhledem ke zlepšení pokrytí automatizovanými testy a tím i dřívějšímu odhalení případných defektů.

8.9 Kdy použít ScrumFlow

ScrumFlow je navržena tak, aby řešila specifické problémy vznikající nasazením metodiky Scrum v korporátním prostředí. Tyto problémy ale mohou nastat (třeba v menším měřítku) i v menším podniku, metodika ScrumFlow se tedy neomezuje pouze na použití v korporaci.

Obecně lze říci, že je vhodné nasadit ScrumFlow v situaci, kdy Scrum Master není schopen přemostit bariéru mezi vlastníkem produktu, přpadně stakeholdery nebo uživateli, a vývojovým týmem. Techniky Behaviour Driven Development mohou v této situaci pomoci najít společnou řeč.

Nutnou podmínkou nasazení ScrumFlow je přítomnost zkušeného testera ve Scrum týmu. Scrum sice nedefinuje roli testera, ale v praxi je běžné, že je tester v týmu přítomen. ScrumFlow tak formalizuje roli testera a definuje požadavky na něj.

8.10 Shrnutí

Metodika ScrumFlow může sloužit jako vodítko pro týmy, které vyvíjejí podle metodiky Scrum, ale nedochází k dostatečnému porozumění mezi vývojovým týmem a byznysem. Hlavní, na koho metodika ScrumFlow ale cílí, jsou Scrum týmy ve velkých firmách, které mají problémy s reportováním a vytvářením dokumentace. Přístup BDD použitý v rámci ScrumFlow přináší generování dokumentů a zjednodušení komunikace mezi vývojovým týmem a byznysem. Metodika Scrum je doplněna o automatizaci testovacích scénářů (respektive akceptačních kritérií, ale neomezuje se pouze na ně) pomocí přístupu metodiky Behaviour Driven Development a pro ni dostupných nástrojů. Zavedením jazyka Gherkin pro komunikaci požadavků na produkt je výrazně zvýšena transparentnost a jednoznačnost zadání. Gherkin nabízí jakousi platformu, „společnou řeč“ vývojového týmu a byznysu.

9 Závěr

Na základě charakteristiky metodiky Scrum a specifik jejího nasazení v korporátním prostředí bylo identifikováno několik problémů, na které týmy narážejí při použití agilního přístupu v tradičně řízené organizaci. Autorka tak navrhuje přístup Behaviour Driven Development, který může pomoci tyto problémy řešit. Nejprve ale ověřuje, že neexistuje formální překážka bránící jejich vzájemné integraci. Naopak, tvůrci metodiky Scrum přímo vybízejí k využívání dalších postupů a technik. Behaviour Driven definuje přesněji určité techniky k definici uživatelských požadavků a jejich automatizace.

Využití Behaviour Driven Development klade na danou oblast nový úhel pohledu: posouvá testování (někdy nesmyslných a rozsáhlých) specifikací k testování toho, co zákazník doopravdy chce. Tedy ověření, jestli se vývoj ubírá správným směrem. Automatizované testy přestávají být pro stranu zákazníka černou skříňkou, *která snad něco testuje*. Díky využití automatizace požadavků a scénářů zainteresovaná osoba v reálném čase vidí (a hlavně tomu, co vidí i rozumí), které požadavky jsou aktuálně ve vývoji, které jsou funkční nebo které jsou zrovna například kvůli integraci s novou komponentou nefunkční. Výrazně se tak zlepšuje integrace strany zákazníka do samotného vývoje – a to je přesně jeden z klíčových aspektů, o které agilní vývoj usiluje. Ale zároveň jsou pomocí Behaviour Driven Development lépe naplněny požadavky například na reporting, aniž by tím tým byl zbytečně zatížen.

Integraci Behaviour Driven Development autorka formalizuje vytvořením metodiky ScrumFlow, která je vhodná zejména pro týmy, které již používají metodiku Scrum.

9.1 Shrnutí dosažených výsledů

Hlavním přínosem této práce je metodika ScrumFlow. ScrumFlow je autorkou navržená metodika, a je rozšířením populární metodiky Scrum o přístup Behaviour Driven Development. ScrumFlow přidává dvě nové role, několik nových artefaktů a vzory dokumentů. Zavádí specifikaci uživatelských požadavků pomocí formalizovaných scénářů zapsaných v jazyce Gherkin. Ten se stává společnou komunikační platformou pro zákazníka a vývojový tým. Dále slouží jako živá dokumentace a reporting v reálném čase, což jsou pro management velmi hodnotné výstupy. Vývojový tým zase může profitovat z pokrytí aplikace regresními testy. Produkt je tak snadněji udržovatelný, případné chyby jsou odhaleny ihned.

Metodika ScrumFlow je autorkou publikována na adrese www.scrumflow.czweb.org, kde je volně k dispozici.

9.2 Zhodnocení naplnění cílů

Hlavním cílem diplomové práce bylo stanoveno rozšíření metodiky Scrum o přístup Behaviour Driven Development a pokusit se jím vyřešit některé problémy vznikající nasazením agilní metodiky v korporátním prostředí. Autorka navrhuje metodiku ScrumFlow a zdůvodňuje, čím metodika pomůže. Především se jedná o zlepšení komunikace a definici uživatelských požadavků, lepší transparentnost a automatizované generování dokumentace, což jsou aspekty, které rozhodně metodice Scrum v korporaci pomohou. Stakeholdeři mají k dispozici častěji reporty, které jsou aktuální (téměř v reálném čase), a uživatelé spolu s vývojáři mají mezi sebou platformu pro definici uživatelských požadavků. Systém je navíc lépe pokryt automatizovanými testy, u kterých je zřejmé, kterou funkcionalitu pokrývají.

Dílčí cíl, provedení rešerše na téma Behaviour Driven Development reflektuje kapitola 2, která staví teoretický základ pro charakteristiku Behaviour Driven Development. V kapitole 6 jsou zase představeny a porovnány dostupné softwarové nástroje pro Behaviour Driven Development, čímž je splněn další dílčí cíl.

Kapitola 7 je věnována nástroji SpecFlow, který byl autorkou vybrán jako jeden z vhodných pro použití v korporátním prostředí – zejména kvůli jeho pokročilým možnostem a doplňkům, díky kterým dává více možností než jeho konkurenti. V kapitole 7 je popsán jak nástroj samotný, tak instalace a úvodní konfigurace. Kapitola 7 tak plní další z dílčích cílů práce, kromě samotného představení nástroje SpecFlow slouží jako jeho jednoduchá uživatelská příručka.

Metodika ScrumFlow byla autorkou implementována v nástroji Eclipse Process Framework Composer Implementace je kompletně v českém jazyce (s výjimkou pojmů metodiky Scrum, ke kterým se běžně nepoužívá český ekvivalent, a jsou již v českém prostředí zažity anglicky). Metodika ScrumFlow je publikována a volně k dispozici na webu www.scrumflow.czweb.org. Poslední z definovaných dílčích cílů práce je tímto také naplněn.

9.3 Náměty pro další práci

Vzhledem k faktu, že námět práce pochází z praxe, se k další práci přímo nabízí uplatnění získaných poznatků v praxi, tedy aplikace navržené metodiky ScrumFlow v praxi Scrum

týmu. Zajímavá by mohla poté být studie vztahu vývojového týmu se zákazníkem ve fázi použití čisté metodiky Scrum, a porovnání s nasazenou metodikou ScrumFlow. Je očekáváno zlepšení vzájemné důvěry a komunikace, ale až ověření praxí ukáže, do jaké míry k tomuto zlepšení dojde. Zajímavé by bylo také analyzovat postoj tradičního managementu k využití spustitelných specifikací jako projektové dokumentace. Což by mohlo být dalším tématem – je takováto dokumentace pro produkt dostatečná? Případně, do jaké míry je možné automatizovat vytváření projektové dokumentace, a případně jakými jinými způsoby?

Pokud jde o oblast automatizace scénářů, dalším zajímavým tématem nad rámec této práce by byla například studie na téma adaptace běžných scénářů na ty automatizované. Protože v praxi nejspíš tým na tuto praktiku nebude přecházet v *den jedna*, ale ve fázi, kdy už některé existující scénáře má.

Dalším přínosem v oblasti automatizace založené na přirozeném jazyce, a srozumitelnosti, by byla lokalizace větší části nástrojů BDD. Nejdůležitější část, Gherkin, v českém jazyce sice existuje, takže psaní uživatelských požadavků v přirozeném mateřském jazyce je možné. Trochu horší je ale práce s generovanými reporty, kde dochází k jazykovým kolizím. Kostra reportu je stále anglicky, a v něm jsou česky definované scénáře a kroky. Pokud bychom samozřejmě šli do důsledku, můžeme se snažit o překlad programovacího jazyka a vývojového prostředí. To jsou ale nástroje primárně určené pro samotné vývojáře. Jazyková kolize tak možná působí nekonzistentně, ale pracovat se v anglicky psaném zdrojovém kódu s česky psanými požadavky dá, aniž by tato práce byla výrazně rušivá. Uživatel ale přichází do styku se dvěma částmi, kterými jsou definice požadavků, a právě uvedený report, který je pro byznys důležitý, ale působí nekonzistentně. Lokalizace alespoň těchto reportů do stejných jazyků, pro které existuje implementace jazyka Gherkin, by tak byla velice přínosná.

10 Terminologický slovník

Agilní metodika	Alternativa k vodopádovému modelu, přístup k vývoji softwaru založený na Agilním manifestu. (Beck a kol., 2001)
Akceptační kritéria	Výstupní kritéria, která komponenta nebo systém musí splňovat k tomu, aby prošel akceptací uživatelem, zákazníkem nebo jinou pověřenou entitou (International Software Testing Qualifications Board, 2015 , s. 1).
Behaviour Driven Development	Přístup k vývoji softwaru využívaný v agilním prostředí, reflektuje tedy principy popsané v Agilním Manifestu (Beck et al., 2001). Podle Astelse (2006), Behaviour Driven Development navazuje na Test Driven Development, přístup, který je využíván například v Extrémním programování.
Feature File	Soubor pro definici požadavků v jazyce Gherkin
Gherkin	Přirozený jazyk (existuje implementace pro více než 60 světových jazyků včetně Češtiny) s přidanou strukturou. Stále je snadno srozumitelný ne-programátory, ale natolik strukturovaný, aby umožnil výstižný popis příkladů vedoucích k vyjasnění byznys pravidel většiny oblastí lidské činnosti. (Cucumber Ltd., nedatováno)
Požadavek	Definice požadavku v jazyce Gherkin pomocí jedené nebo více user stories.
Scénář	Spustitelná součást popisu požadavku v jazyce Gherkin
Scrum	Agilní framework pro komplexní projekty. Původně byl vytvořen pro softwarové projekty, lze ho ale využít pro jakýkoliv komplexní inovativní projekt i mimo

	softwarové odvětví. (Scrum Alliance, 2016)
SpecFlow	Softwarový nástroj pro Behaviour Driven Development v jazyce C#.
User Story	*High level*, nebo také *všeobecný*, byznys požadavek, typicky popsaný jednou nebo více větami běžného byznys jazyka. Zachycuje, jakou funkcionalitu uživatel potřebuje a její akceptační kritéria. Typicky je *user story* využívaná v prostředí agilního vývoje. (International Software Testing Qualifications Board, 2015 , s. 74)

Seznam literatury

ADZIC, Gojko, 2011. *Specification by Example* [online]. ISBN 9781617290084. Dostupné z: http://it-ebooks.info/book/594/\nfile:///Users/a_/Backup/Papers2/Articles/2011/Adzic/2011 Adzic.pdf\npapers2://publication/uuid/E91CB152-D713-4F6D-B4C1-299268797848

AGILE ALLIANCE AND INSTITUT AGILE, 2013. *Guide to Agile Practices - BDD* [online] [vid. 12. březen 2016]. Dostupné z: http://guide.agilealliance.org/guide/bdd.html

AL-ALLAF, Omaima N A, 2010. The Adoption of Agile Processes in Large Web Development Enterprises : A Survey in Jordan. roč. 2, č. 3, s. 206–216.

AMBLER, Scott W., 2012. *Results from the November 2012 Agile Testing Survey* [online] [vid. 10. duben 2016]. Dostupné z: http://www.ambysoft.com/surveys/agileTesting201211.html

ANON., nedatováno. *Introduction to Test Driven Development (TDD)* [online] [vid. 10. duben 2016]. Dostupné z: http://www.agiledata.org/essays/tdd.html

ASTELS, Dave, 2006. A New Look at Test-Driven Development. *Divulgado em http://blog. daveastels. com/files/ ...* [online]. s. 0–7. Dostupné z: http://www.elgustodecrecer.es/media/pdf/uploaded/menu/12375.pdf

BALADA, Jakub, 2015. *Agilní metodiky v komplexním prostředí* [online]. B.m. Vysoká škola ekonomická v Praze. Dostupné z: https://www.vse.cz/vskp/id/1276363

BECK, Kent, 2003. Test-Driven Development By Example. *Rivers* [online]. roč. 2, č. c, s. 176. ISSN 1660-1769. Dostupné z: doi:10.5381/jot.2003.2.2.r1

BECK, Kent, Mike BEEDLE, Arie BENNEKUM VAN, Alistair COCKBURN, Ward

CUNNINGHAM, Martin FOWLER, James GRENNING, Jim HIGHSMITH, Andrew HUNT,

Ron JEFFRIES, Jon KERN, Brian MARICK, Robert C. MARTIN, Steve MELLOR, Ken

SCHWABER, Jeff SUTHERLAND a Dave THOMAS, 2001. *Manifest Agilního vývoje*

software [online] [vid. 6. březen 2016]. Dostupné z:

http://www.agilemanifesto.org/iso/cs/manifesto.html

BUCHALCEVOVÁ, Alena, 2009. *Metodiky budování informačních systémů*. 1. vyd. Praha: Oeconomica. ISBN 978-80-245-1540-3.

CARDOVÁ, Zdenka, 2006. Malé a střední podniky – vymezení a specifika účetnictví a výkaznictví. *Český finanční a účetní časopis*. č. 1, s. 74–79.

CARRERA, Álvaro, Carlos a. IGLESIAS, Mercedes GARIJO, Alvaro CARRERA, Carlos a. IGLESIAS a Mercedes GARIJO, 2014. Beast methodology: An agile testing methodology for multi-agent systems based on behaviour driven development. *INFORMATION SYSTEMS FRONTIERS* [online]. roč. 16, č. 2, s. 169–182. ISSN 1387-3326. Dostupné z: doi:10.1007/s10796-013-9438-5

COHN, Mike, 2010. Suceeding with Agile: Software Development Using Scrum. s. 503.

CROOKSHANKS, Edward, 2015. *Practical Enterprise Software Development Techniques - Tools and Techniques for Large Scale Solutions*. ISBN 978-1-4842-0620-1.

CUCUMBER LTD., 2016. *Gherkin/gherkin-languages.json* [online]. 2016. B.m.: GitHub repository. Dostupné z: https://github.com/cucumber/gherkin/blob/master/gherkin-languages.json

CUCUMBER LTD., nedatováno. *Gherkin* [online] [vid. 6. březen 2016]. Dostupné z: https://cucumber.io/docs/reference

DIEPENBECK, Melanie, Mathias SOEKEN, Daniel GROSSE a Rolf DRECHSLER, 2012. Behavior Driven Development for Circuit Design and Verification. In: *2012 IEEE INTERNATIONAL HIGH LEVEL DESIGN VALIDATION AND TEST WORKSHOP (HLDVT)*. IEEE International High Level Design Validation and Test Workshop. ISBN 978-1-4673-2897-5.

GIUDICE, Lo, 2014. How Can You Scale Your Agile Adoption ?

HAUMER, Peter, 2007. Eclipse Process Framework Composer Part 1. *IBM Rational Software* [online]. č. 2nd Revision, s. 1–25. Dostupné z: http://www.eclipse.org/epf/general/EPFComposerOverviewPart1.pdf

CHELIMNSKY, David, 2010. *The RSpec Book*. ISBN 9780978739225.

IAN DEES, Matt Wynne Aslak Hellesøy, 2013. Cucumber Recipes [online]. s. 1–266. Dostupné z: http://uploaded.net/file/tpl2o8m6/b20130427mge_CucumberRecipe.pdf\nfile:///Users/a_/Backup/Papers2/Articles/2013/Ian Dees/2013 Ian Dees.pdf\npapers2://publication/uuid/F1EBFC18-D8EE-42E7-BFF7-0111E7DF3A46

INTERNATIONAL SOFTWARE TESTING QUALIFICATIONS BOARD, 2015. *Standard glossary of terms used in Software Testing*. ISBN 9781844803552.

INVIQA, 2015. *The Beginner's Guide to BDD* [online] [vid. 1. březen 2016]. Dostupné z: http://inviqa.com/insights/bdd-guide

K.BECK, C.Andres, 2005. Extreme Programming Explained. *Writing* [online]. s. 85–110. ISSN 20161641. Dostupné z: http://www.lalilуna.de/assets/tutorials/junit-testing-en.pdf

KEOGH, Liz, 2009. *Translating TDD to BDD* [online] [vid. 1. leden 2016]. Dostupné z: http://lizkeogh.com/2009/11/06/translating-tdd-to-bdd/

KOŠATA, Václav, 2010. *Metodika Scrum* [online]. B.m. Vysoká škola ekonomická v Praze. Dostupné z: https://www.vse.cz/vskp/id/1171836

LINZ, Tilo, 2014. *Testing in Scrum A Guide for Software Quality Assurance in the Agile World*. ISBN 978-1-4920-0152-2.

MANAGEMENTMANIA.COM, 2015. *Nadnárodní korporace (Multinational corporation)* [online] [vid. 10. duben 2016]. Dostupné z: https://managementmania.com/cs/nadnarodni-korporace

MARK, C, 2013. A Scrum Adoption Survey.

NEČAS, Ivan, 2011. *BDD as a Specification and QA Instrument* [online]. B.m. b.n. Dostupné z: http://is.muni.cz/th/208305/fi_m/master-thesis.pdf

NORTH, Dan, 2006. *Introducing BDD* [online]. [vid. 4. leden 2016]. Dostupné z: http://dannorth.net/introducing-bdd/

NORTH, Dan a Liz KEOGH, 2009. *Introduction to BDD* [online]. [vid. 14. prosinec 2015].

PAULK, Mark C., 2013. A Scrum Adoption Survey.

PLURALSIGHT, nedatováno. *Pluralsight | Unlimited, online training for IT, Developer and Creative pros* [online] [vid. 1. prosinec 2015]. Dostupné z: https://www.pluralsight.com/

RESNICK, Steve, Aaron BJORK, Michael De La MAZA a & 0 MORE, 2011. *Professional Scrum with Team Foundation Server 2010* [online]. ISBN 9780470943335. Dostupné z: http://www.amazon.com/Professional-Scrum-Team-Foundation-Server/dp/0470943335/ref=sr_1_fkmr0_2?ie=UTF8&qid=1398183214&sr=8-2-fkmr0&keywords=Professional+Scrum+with+Team.Foundation.Server.2010.Apr.2011

SCRUM ALLIANCE, 2007. *Glossary of Scrum Terms* [online] [vid. 15. únor 2016]. Dostupné z: https://www.scrumalliance.org/community/articles/2007/march/glossary-of-scrum-terms#1128

SCRUM ALLIANCE, nedatováno. *What is Scrum? An Agile Framework for Completing Complex Projects* [online] [vid. 1. březen 2016]. Dostupné z: https://www.scrumalliance.org/why-scrum

SCHWABER, K, 2004. *Agile Project Management with Scrum* [online]. ISBN 073561993X. Dostupné z: doi:10.1201/9781420084191-c2

SCHWABER, Ken, 2007. *The Enterprise and Scrum* [online]. ISBN 9780735623378. Dostupné z: http://books.google.com/books?id=KKyUAAAACAAJ&pgis=1

SCHWABER, Ken a Jeff SUTHERLAND, 2013. ™ Průvodce Scrumem.

SIMS, Chris a Hillary Louise JOHNSON, 2012. *Scrum : a Breathtakingly Brief and Agile Introduction This overview of roles , artifacts and the sprint The Elements of Scrum*. B.m.: Dymaxicon. ISBN 9781937965044.

SMART, John Ferguson, 2014. *BDD In Action: Behavior Driven Development for the Whole Software Lifecycle* [online]. ISBN 9781617291654. Dostupné z: https://www.ebooks-it.net/ebook/bdd-in-action

SOEKEN, Mathias, Robert WILLE a Rolf DRECHSLER, 2012. Assisted Behavior Driven Development Using Natural Language Processing. In: FURIA, CA AND NANZ, S, ed. *OBJECTS, MODELS, COMPONENTS, PATTERNS, TOOLS 2012*. Lecture Notes in Computer Science. ISBN 978-3-642-30561-0.

SUSCHECK, Charles, 2014. *The Origins and Future of Scrum: An Interview with Ken Schwaber* [online] [vid. 4. duben 2016]. Dostupné z: https://www.agileconnection.com/interview/origins-and-future-scrum-interview-ken-schwaber?page=0%2C0

ŠOCHOVÁ, Zuzana a Eduard KUNCE, 2014. *Agilní metody řízení projektů*. Brno: Computer Press. ISBN 978-80-251-4194-6.

TECHTALK, nedatováno. Documentation | SpecFlow [online]. [vid. 14. leden 2016 a]. Dostupné z: http://www.specflow.org/documentation/

TECHTALK, nedatováno. *SpecFlow - Cucumber for .NET* [online] [vid. 4. březen 2016 b]. Dostupné z: http://www.specflow.org/

TECHTALK, nedatováno. *SpecFlow eco-system | SpecFlow* [online] [vid. 4. březen 2016 c]. Dostupné z: http://www.specflow.org/specflow-eco-system/

TOMÁNEK, Martin, 2010. *Řízení projektů agilního vývoje softwaru na základě PRINCE2 a*

Scrum [online]. B.m. Vysoká škola Ekonomická v Praze. Dostupné z: https://www.vse.cz/vskp/id/1281273

VODIČKA, Petr, 2012. *Behaviour Driven Development*. B.m. b.n.

WENDT, Jerome M., 2011. *Redefining "SMB", "SME" and "Large Enterprise"* [online] [vid. 12. duben 2016]. Dostupné z: http://www.dcig.com/2011/03/redefining-smb-sme-and-large-enterprise.html

WYNNE M., Hellesøy A., 2012. *The Cucumber Book* [online]. ISBN 9781934356807. Dostupné z: http://ompldr.org/vZXN5aA/the-cucumber-book_p1_0.pdf

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu