Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky

Katedra systémové analýzy

Studijní program: Aplikovaná informatika

Obor: Informatika

Systémový přístup k agilním metodikám vývoje software

BAKALÁŘSKÁ PRÁCE

Student : Daniel Urban

Vedoucí : doc. Ing. Stanislava Mildeová, CSc.

Oponent : Mgr. Lukáš Konečný

2016

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a že jsem uvedl všechny použité prameny a literaturu, ze které jsem čerpal.

V Praze dne 4. 5 2016 . . . . . .

Daniel Urban

Poděkování

Rád bych tímto poděkoval svojí vedoucí práce, paní doc. Ing. Stanislavě Mildeové, CSc. za její rady, nápady a objektivní kritiku, které mi významně pomohly k úspěšnému dokončení práce.

Dále bych rád poděkoval své rodině, přátelům a kolegům za jejich trpělivost a podporu.

Abstrakt

Tato bakalářská práce je zaměřena na zkoumání agilních metodiky vývoje software pomocí systémového přístupu.

Práce má dva cíle. Prvním z cílů je popis a rozbor agilních metodik pomocí systémového přístupu vzhledem k projektu řešeného v této práci. Druhým a hlavním cílem práce je po analýze metodik výběr vhodné agilní metodiky pro studentský projekt v bankovní instituci, v rámci kterého je vyvíjena aplikace pro potřeby oddělení lidských zdrojů. Podcílem je porovnání současného přístupu aplikovaného na vývoj s výsledkem výběru metodik.

Klíčová slova

agilní metodiky, systémový přístup, SCRUM, extrémní programování, Lean, Kanban

Abstract

This bachelor thesis is focused to researching agile methodologies of software development using a systems approach.

This thesis has two main objectives. The first objective is the description and analysis of agile methodologies using a systems approach with respect to the project conducted in this thesis. The second and primary objective is selection of the appropriate agile methodology for student‘s project in a bank institution following analysis methodologies within which the application is developed for the needs of the Human Resources Department. Partial objective of this to compare the current approach applied to the development of the result of the selection of methodologies.

Keywords

agile methodologies, systems approach, SCRUM, extreme programming, Lean, Kanban

Obsah

1 Úvod 1
1.1 Cíl práce 1
1.2 Předpoklady a omezení práce 1
1.3 Struktura práce 1
1.4 Očekávaný přínos práce 2
1.5 Rešerše prací zabývajících se agilními metodikami 2
2 Tradiční metodiky 5
2.1 Vodopádový model 5
2.1.1 Charakteristika vodopádového modelu 5
2.1.2 Výhody a nevýhody vodopádového modelu 6
2.2 Spirálový model 6
2.2.1 Charakteristika spirálového modelu 6
2.2.2 Výhody a nevýhody spirálového modelu 8
2.3 Ostatní rigorózní metodiky 8
3 Charakteristika vybraného projektu 9
4 Systémový přístup k agilním metodikám 12
4.1 Faktory pro výběr zkoumaných metodik 12
4.2 Úhly pohledu na agilní metodiky 13
4.3 Současný stav využívání agilních metodik 13
4.4 Charakteristika agilních metodik 16
4.5 Prvky agilních metodik 18
4.6 Všeobecné výhody a nevýhody agilních metodik 19
4.7 SCRUM 20
4.7.1 Historie 20
4.7.2 Pojmy a role 20
4.7.3 Charakteristika 28
4.7.4 Další praktiky metodiky SCRUM 31
4.7.5 Výhody a nevýhody SCRUM 34
4.8 Extrémní programování (XP) 35
4.8.1 Charakteristika XP 35
4.8.2 Praktiky XP 40

4.8.3 Výhody a nevýhody XP 44
4.9 Lean software development 44
4.9.1 Principy Lean software development 46
4.9.2 Shrnutí Lean software development 48
4.10 Kanban 49
4.10.1 Charakteristika Kanbanu 49
4.10.2 Výhody a nevýhody Kanbanu 52
4.11 Ostatní agilní metodiky 53
5 Výběr vhodné metodiky a její aplikace 54
5.1 Výběr vhodné metodiky pomocí vícekriteriálního rozhodování TOPSIS 54
5.2 Výběr vhodné metodiky pomocí METES 59
5.2.1 Kritéria 59
5.2.2 Výběr metodiky 60
5.3 Aplikace vybrané metodiky 63
6 Závěr 64
6.1 Přínos práce 65
6.2 Možná budoucí rozšíření práce 65
Terminologický slovník 66
Seznam literatury 68
Seznam obrázků a tabulek 68
Seznam obrázků 73
Seznam grafů 73
Seznam tabulek 73
Příloha A: Výpočty vícekriteriálního rozhodování TOPSIS 75
Příloha B: Výpočty systému METES 77

1 Úvod

V současnosti již agilní metodiky nejsou žádnou novinkou v přístupu k vývoji software, ale používají se ve velké míře. O to důležitější ovšem je, pečlivě vybírat vhodnou metodiku pro daný tým a projekt. V některých případech je možné, že danému projektu budou vyhovovat i metodiky, které jsou v různých publikacích zavrhovány jako již zastaralé.

Tato práce se zabývá metodikami vývoje software, které dle (Chlapek, 2008) patří do kategorie Metodiky věcného řešení projektů.

Motivací pro výběr tohoto tématu je moje účast na projektu, který je v této práci zmiňován a zkoumán. Jelikož považuji za důležité používat kritické myšlení, chtěl jsem touto prací ověřit vhodnost na projektu využívané metodiky. Součástí této motivace je i prohloubení znalostí ostatních metodik a historie vývoje přístupu k vývoji software.

1.1 Cíl práce

Cíl této bakalářské práce by se dal rozdělit na dva na sebe navazující dílčí cíle.

- Za prvé je to především, detailně popsat vybrané agilní metodiky a praktiky s nimi spjaté a především analyzovat agilní metodiky vývoje software pomocí systémového přístupu.
- Navazujícím druhým hlavním cílem, po analýze metodik, je vybrat vhodnou agilní metodiku – v tomto případě pro studentský projekt, který se zbývá vývojem aplikace v bankovní instituci pro potřeby oddělení lidských zdrojů.
    - Dílčím cílem je srovnat výsledky vybraných metodik s aktuálně využívaným přístupem k vývoji v organizaci.

1.2 Předpoklady a omezení práce

Práci mohou číst i lidé neznalí oboru, avšak aby čtenář plně pronikl do tématu je předpokládána alespoň elementární znalost IT a vývoje softwaru.

Předpokládaným čtenářem je člověk, který chce prohloubit své znalosti týkající se přístupu k vývoji software, nebo člověk, kterého zajímá případová studie obsažená v této práci.

1.3 Struktura práce

Práce je členěna do šesti kapitol.

První kapitolou je Úvod, ve kterém jsou uvedeny cíle práce, předpoklady a omezení práce, struktura práce a očekávaný přínos práce.

V druhé kapitole je stručný úvod do tradičních přístupů k vývoji.

Ve třetí kapitole je popsán projekt a jsou z něj vybrány nejdůležitější charakteristiky.

Ve čtvrté a nejobsáhlejší kapitole je úvod do agilních principů, historie a trendů a poté jsou pomocí systémového přístupu analyzovány a zkoumány jednotlivé vybrané metodiky.

V páté kapitole je posouzena vhodnost vybraných agilních praktik a přístupů vůči charakteristikám projektu pomocí vícekriteriálního přístupu. Výsledek této kapitoly je doplněn o pohled skrze systém METES.

V poslední šesté kapitole je práce shrnuta a uveden závěr.

Po závěru následuje seznam literatury, terminologický slovník, seznam obrázků a tabulek a přílohy.

1.4 Očekávaný přínos práce

Očekávaný přínos této práce lze rozdělit do dvou kategorií a to části teoretické a praktické.

Očekávaným přínosem práce je v rovině teoretické seznámení čtenáře nejprve stručně s tradičními koncepty vývoje. V další části se jedná o rozbor projektu na jednotlivé charakteristiky, kdy následuje úvod do principů agilních metodik, jejich historie a aktuální trendy. V další kapitole jsou nejprve vybrány zkoumané metodiky a poté dojde k detailní analýze vybraných agilních metodik pomocí systémového přístupu.

V praktické části je rozbor samotného projektu. Přínosem by měla být možnost využití jednotlivých agilních metodik. Poté je pomocí vícekriteriálního rozhodování TOPSIS vybrána vhodná metodika a pomocí systému METES je potvrzen tento výběr.

1.5 Rešerše prací zabývajících se agilními metodikami

Jelikož je téma agilních metodik obsáhlé a mnohokrát zpracované, je důležité provést rešeršní práce.

Základním stavebním kamenem této práce je kniha od Václava Kadlece - *Agilní programování: metodiky efektivního vývoje software* (Kadlec, 2004). Jedná se sice o knihu přes 10 let starou, což zejména v oblasti vývoje software je značná doba, avšak základní prvky zde uvedené stále platí a jedná se o nejrozsáhlejší literaturu dostupnou v českém jazyce.

Druhou knihou, ze které vychází především metodika SCRUM ale i ostatní agilní metodiky je kniha *Agilní metody řízení projektů* od Zuzany Šochové a Eduarda Kunce (Zuzana Štochová a Eduard Kunce, 2014). Agilní metodiky v této knize jsou pojaty spíše komplexně a propojeně.

Třetí knihou, ze které tato práce vychází, je kniha *Learning Agile: Understanding Scrum, XP, Lean, and Kanban* (Stellman a Greene, 2015). Tato kniha byla pro tuto práci vybrána z důvodu, že se jedná o nejaktuálnější literaturu týkající se čtyř agilních metodik, které jsou zmíněny v názvu knihy, a jsou zde tím pádem popsány aktuální trendy daných metodik.

Poslední knihou, ze které vycházejí všechny agilní metodiky je *Informační management v informační společnosti* (Doucek et al., 2013). V této jsou stručně popsány jednotlivé metodiky, avšak jedná se o další kompetentní náhled a názor na tuto problematiku.

Důležitým zdrojem, ze kterého vychází kapitola SCRUM, je *™Průvodce Scrumem* od hlavních představitelů této metodiky Kena Schwabnera a Jeffa Sutherlanda. Jsou zde věcně popsány všechny principy a artefakty metodiky.

Popis a analýza metodiky XP vychází z knihy, která XP přinesla. Konkrétně se jedná o *Extreme programming explained: Embrace change* od zakladatele metodiky Kenta Becka. V knize jsou popsány všechny přístupy, role a artefakty, které metodika volí.

Lean principy aplikované v IT, tedy jako Lean software Development popisuje nejlépe, a proto je vybrána jako hlavní zdroj, kniha Lean software development: an agile tookit od Mary a Toma Poppendieck (Poppendieck a Poppendieck, 2003).

Kapitola popisující Kanban vychází z především z titulu *Kanban: Successful Evolutionary Change for Your Technology Business* od Davida J. Andersona (Anderson, 2010), který Kanban přenesl do IT.

Praktická část práce, tedy výběr vhodné metodiky, vychází z knihy od paní docentky *Buchalcevové - Metodiky budování informačních systémů* (Buchalcevová, 2009). Kniha je již staršího data, avšak jako teoretický podklad je naprosto dostačující.

Bakalářské, diplomové či disertační práce, které nejvíce souvisí s tématikou této práce, jsou následující:

- Metodika SCRUM a její programová podpora

Bakalářská práce od Jana Marka: *Metodika SCRUM a její programová podpora* (Marek, 2013) je zaměřena jak název napovídá, na programovou podporu pro aplikaci metodiky SCRUM. Cílem této práce je výběr vyhovující programové podpory a následné porovnání a vyhodnocení programů podle daných kritérií. Oproti zmiňované práci se tato práce zabývá více analýzou agilních metodik a aplikací dané metodiky je věnováno méně prostoru. Práce je z roku 2013, čímž se řadí k aktuálnějším pracím v této problematice.

- Srovnávací analýza metodik vývoje software

Práce *Srovnávací analýza metodik vývoje software* od Vladimíra Popelky z roku 2009 (Popelka, 2009) se zabývá porovnáváním rigorózní metodiky RUP s agilními metodikami XP, TDD a FDD. K porovnání používá vícekriteriální optimalizační metodu TOPSIS. Oproti práci Vladimíra Popelky jsou v této práci analyzovány další agilní metodiky a je věnován menší důraz na hledisko testování.

- Problematika výběru agilní metodiky vývoje software

Diplomová práce z roku 2014 od Bc. Roberta Fujdiara: *Problematika výběru agilní metodiky vývoje software* (Fujdiar, 2014) je tématem mé práci nejblíže. Autor se zabývá především čtyřmi metodikami a to XP, SCRUM, Kanban a ScrumBan. Cílem této práce je pomoci při výběru agilní metodiky pro vývoj software. Autor této práce vybral metodiku pro celou firmu, kdežto cílem mojí práce je vybrat metodiku pro konkrétní projekt.

- Kanban a možnosti jeho využití v bankovním prostředí

Jedná se o diplomovou práci (Hefnerová, 2015) od Bc. Lucie Hefnerové, která se, jak je z názvu práce patrné zabývá pouze Kanbanem. Kanban je v této práci velice detailně popsán jak z hlediska teorie, tak z hlediska následné případové studie. Tato práce je ovšem zaměřena úzce na Kanban a ostatní metodiky jsou pouze brány v potaz při kombinaci s Kanbanem.

- Lean software development

Od stejné studentky, tedy Lucie Hefnerové je bakalářská práce *Lean software development* (Hefnerová, 2012), který se zabývá tímto přístupem. Stejně jako její práce zaměřená na Kanban, je i tato úzce zaměřená na princip Lean software developmentu

2 Tradiční metodiky

Nejprve je nutné definovat, co tradiční metodiky jsou. Jelikož agilní metodiky, tak jak je dnes známe, se prosazují přes patnáct let, tj. od doby vydání agilního manifestu v roce 2001 (K. Beck, M. Beedle, A. van Bennekum, et al. 2001), mohlo by se tedy zdát, že se již jedná o tradiční metodiky. Avšak podle definice slova tradice od Hartla a Hartlové (Hartl a Hartlová, 2000), tradice procházejí generacemi. Z tohoto úhlu pohledu jsou tradičními metodikami metodiky rigorózní.

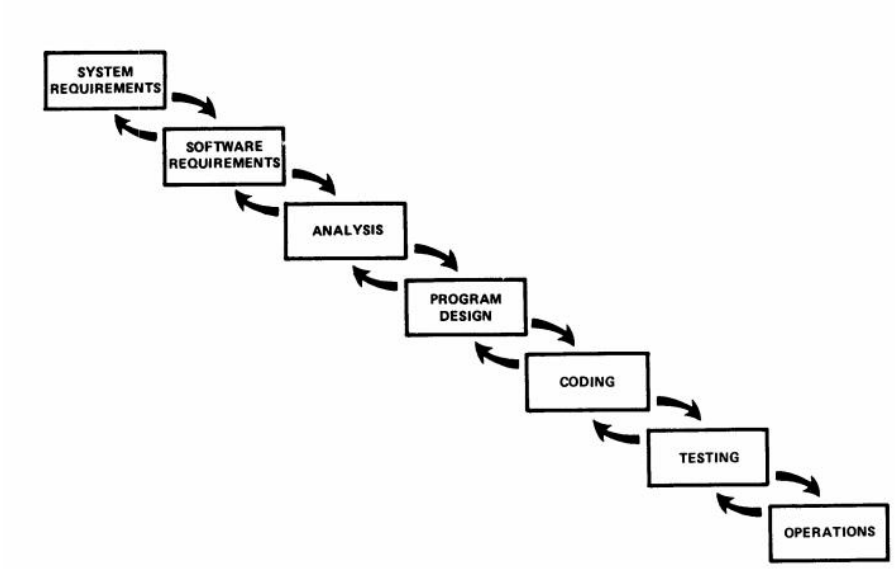
2.1 Vodopádový model

Nejedná se o metodiku v pravém slova smyslu, ale o životní cyklus vývoje software. Přesto je důležité tento přístup k vývoji software zohlednit, jelikož se jednalo ve své době o průlomový přístup. Jako první přinesl členění procesu vývoje software na jednotlivé aktivity.

2.1.1 Charakteristika vodopádového modelu

Pro vodopádový model je charakteristické především rozdělení procesu do fází a posloupnost těchto fází.

Jednotlivé fáze modelu tak jak je v roce 1970 definoval Dr. Winston Royce ve své práci *Managing the Development of Large Software Systems* (Winston W. Royce, 1970), tyto fáze vyjadřuje schéma v této publikace (viz obr 1).



Obrázek 1: Vodopádový model (Zdroj: Winston W. Royce, 1970, str. 329)

Jak je na schématu naznačeno je možné postupovat vždy po jedné fázi i s tím, že je možné se vrátit o jeden krok zpět. Tato struktura by mohla vzdáleně připomínat agilní přístup, avšak zde je možné se vrátit vždy pouze o jeden krok zpět a vždy je třeba projít celým schvalovacím procesem dané fáze.

Po tom, co je dokončena fáze integrace a testování a je předán produkt zákazníkovi, je možné se vrátit do fáze návrhu a provést potřebné úpravy.

2.1.2 Výhody a nevýhody vodopádového modelu

Z charakteristiky vodopádového modelu vyplívá, že je třeba na začátku přesně definovat požadavky na produkt, protože každou chybnou specifikaci je možné napravit až po předání výsledného produktu. Zákazník je tedy pro vývojový tým potřebný pouze v první fázi (při analýze a specifikaci požadavků) a v poslední fázi (údržba), což je z dnešního pohledu na vývoj software zcela nedostačující a neflexibilní.

Druhou nevýhodou tohoto přístupu k vývoji je, že zde nastávají velké nárazy jak na tým, tak na zákazníka. V prvních třech fázích je práce na vývoje v podstatě jen analytická a tím pádem jsou nevytíženi programátoři a testeři. Ve fázi programování jsou naopak vytíženi pouze vývojáři, poté následují velký náraz na testery, a finálním krokem je dodání hotového produktu zákazníkovi. Zákazník musí poté investovat velké úsilí a zdroje na zaučení lidí, kteří budou s programem pracovat.

Nevýhod vodopádového modelu je z dnešního pohledu na vývoj velké množství. Za zmínku dle (Doucek et al., 2013) stojí například tyto: při změně požadavků na software během vývoje, musí zákazník čekat až do finálního předání a teprve poté může nastat úprava programu. A s tím souvisí i možné nepochopení potřeb zákazníka, na které se bohužel přichází až při finálním předání produktu.

Většina výhod vodopádového modelu postupem času vymizela. Ve své době výhody převládaly nad nevýhodami, avšak z dnešního pohledu se zde jeví pouze jedna jediná větší výhoda a to jednoduchost. Aplikovat na vývoj vodopádový model je velmi snadné, snad i přirozené. Z této výhody by ovšem mohli těžit jen malé krátkodobé projekty, na kterých by pracovalo několik málo lidí. Druhou a poslední výhodou, která stojí za zmínku je jednoduchost na řízení projektu. V každém okamžiku vývoje je jasné na čem se má pracovat a v jakém stavu aplikace je.

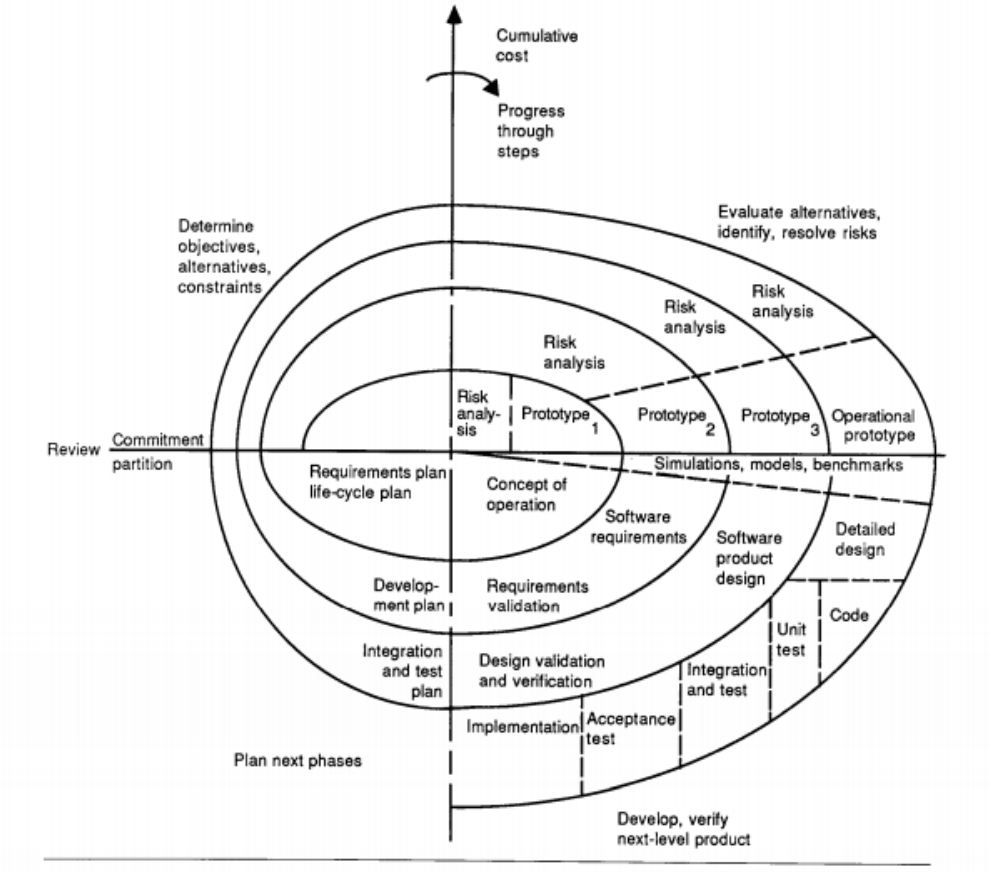
2.2 Spirálový model

Spirálový model je stejně jako vodopádový model spíše životním cyklem, nežli metodikou. Se spirálovým modelem přišel už v roce 1985 Barry Boehm. Ovšem až v roce 1988 mu vyšel známější článek *A Spiral Model of Software Development and Enhancement* (Barry Boehm, 1988), ve kterém definoval tento přístup k životnímu cyklu software.

2.2.1 Charakteristika spirálového modelu

Oproti vodopádovému modelu zde Boehm přidal iterativní přístup. Tento přístup přináší do vývoje software opakující se cykly, což umožňuje upřesňovat specifikace, či požadavky na produkt. Odpadá tím chybovost vodopádového modelu, kde je nutné vše přesně specifikovat na počátku vývoje a úpravy požadavků jsou možné až po předání kompletního produktu.

Stejně jako ve vodopádovém modelu je zde výchozí schéma (viz Obr. 2) a to z roku 1988.



Obrázek 2: Schéma vodopádového modelu (Zdroj: Barry Boehm, 1988, str. 64)

Schéma je na první pohled trochu chaotické a nedá se v něm orientovat tak rychle jako ve schématu vodopádového modelu.

Zjednodušeně lze říci, že cykly jsou rozděleny do čtyř kvadrantů a to následovně:

- Levý horní kvadrant: zde se provádí analýza, stanovují se cíle, alternativy a omezující podmínky pro následující iterace.
- Pravý horní kvadrant: v tomto kvadrantu se provádí analýza rizik a poté se vytváří prototypy, aby se ujasnili specifikace.
- Pravý dolní kvadrant označuje samotný vývoje, testování a implementaci aplikace.
- V levém dolním kvadrantu se provádí plánování.

Vývoj podle schématu probíhá od středu směrem ven. Každý cyklus spirály má svůj specifický význam.

V prvním cyklu je cílem především najít globální rizika a určit koncept vývoje. Druhý cyklus se zabývá konstrukcí a specifikací systému. Ve třetím cyklu se vytváří a ověřuje detailní design produktu. A v závěrečném čtvrtém cyklu se implementuje, testuje a integruje produkt. Každý z těchto cyklů obsahuje výše zmíněné čtyři fáze (představované kvadranty).

Spirálový model vývoje software patří do kategorie tzv. riziky řízených přístupů. To spočívá v tom, že je kladen důraz na analýzu rizik a problémů a až poté co jsou daná rizika vyhodnocena, se pokračuje ve vývoji s případnými úpravami.

Ze spirálového modelu vzniklo několik dalších modelů a to např.: model Win-Win či metodika MBASE.

2.2.2 Výhody a nevýhody spirálového modelu

Spirálový model přináší oproti vodopádovému několik výhod.

Analýza rizik v každém cyklu přináší menší pravděpodobnost, že se zvolí nesprávné řešení. Při analýze alternativ v levém horním kvadrantu, je možné přijít s využitím již existující komponenty a tím ušetřit zdroje. Díky komplexnosti modelu, což zahrnuje plánování, ověřování i analýzu rizik, je vhodný na rozsáhlé a složité projekty. Zároveň je produkt testován v každé iteraci, což přináší nejenom včasné odhalení chyb, ale ve srovnání s vodopádovým modelem odpadá i nárazovost na testery.

Hlavní nevýhodou spirálového modelu je jeho komplikovanost. Tato vlastnost je zbytečná především pro malé projekty. Pro velké projekty je nevýhodou, která přetrvává od vodopádového modelu, že se komplexní software dodává až před skončením posledního. Stejně tak lze mezi nevýhody zařadit to, že je možné měnit požadavky až po dokončení každého cyklu a ne v jeho průběhu. V tomto modelu jsou dle (Doucek et al., 2013) již kladeny větší nároky na tým. Je nutné mít kvalitní a zkušené vývojáře a především analytiky, na kterých rizikově řízený přístup stojí a padá.

2.3 Ostatní rigorózní metodiky

Pro detailní seznámení s historií a současným stavem vývoje software je dobré zmínit metodiky UP (Unified Process) a její modifikace jako RUP (Rational Unified Process) či OpenUP. Kvůli zaměření a rozsahu této práce však nejsou tyto a další metodiky, či přístupy k vývoji software nezbytné.

3 Charakteristika vybraného projektu

Pro správné uchopení slova projekt je nejprve nutné si ho definovat. Volně přeložená definice od (Project Management Institute, 2013) označuje projekt jako „*dočasné úsilí vynaložené na vytvoření unikátního produktu, služby nebo výsledku*“.

Studentský projekt v této práci představuje vývoj aplikace pro podporu oddělení lidských zdrojů v nejmenované bankovní instituci, což je pro tento projekt zákazník. Vývojový tým a zákazník jsou ve stejné nejmenované instituci. Jelikož se jedná o studentský interní projekt, není třeba zde řešit problematiku budgetu. Na tomto projektu pracuje tým o třech lidech. Dva z nich jsou programátoři, třetím členem týmu je tester-analytik, představovaný mnou. Nad projektem dohlíží developer na seniorské pozici, který se projektu věnuje jeden den v týdnu. Tým má k dispozici tři pracovní místa, většinu času je tedy spolu v jedné místnosti.

CH1: Tým je složen ze tří studentů.

CH2: Senior developer (teamlead) je na projektu jeden man-day/týden.

CH3: Tým má k dispozici tři pracovní místa v jedné místnosti.

Všichni tři členové týmu jsou studenti, je tedy zapotřebí brát v potaz nepravidelný čas docházky. A to jak v rámci pracovního týdne tak i jednotlivých dnů, kdy některý student může přijít v osm ráno a jiný až ve tři hodiny odpoledne. Tato charakteristika projektu ztěžuje dodržování některými metodikami předepsaných meetingů.

CH4: Nepravidelná docházka – různé dny, různé časy příchodu, nepravidelnost.

Projekt nemá vizi velkého růstu, takže není nutné řešit škálovatelnost vybírané metodiky.

Tím, že se jedná o aplikaci agilní metodiky, není třeba brát v potaz jednotlivé role z pohledu vývojářských znalostí. Je však třeba dbát na znalosti jednotlivých členů z pohledu schopnosti vývoje jako týmu. Výše zmínění programátoři/studenti mají obdobné znalosti všech na projektu využívaných technologií.

Pokud stojí před týmem vývoj těžší funkcionality, využívá se párového programování a to jak student-student, tak student-senior, kdy senior přechází do role mentora či učitele.

CH5: Znalosti členů týmu (mimo senior developera) jsou na tzv. junior úrovni.

Složitější to je v případě metodiky SCRUM s určením Scrum mastera, jelikož teamleader, jak již bylo zmíněno, pracuje na projektu pouze z 20 %. Zároveň je dle (Zuzana Štochová a Eduard Kunce, 2014) třeba dávat pozor na slučování jakékoliv jiné role s rolí Scrum mastera.

Štochová zároveň pozměňuje roli manažera, který neřeší každodenní operativu, ale spíše se stará o strategická rozhodnutí. Takto by role vyhovovala možnostem teamleadera.

Jelikož tým společně pracuje již přes čtyři měsíce a tím pádem se stává tzv. self-organized, dostává se role Scrum mastera do pozadí, což je velmi vhodné, jelikož tým nemůže mít dedikovaného Scrum mastera.

Problém nastává též u role zákazníka. Je sice jasně definované kdo má absolutní právo na rozhodování, ale tento člověk nemá časové kapacity na dostatečné věnování se projektu. Má tedy zastupujícího, který projektu věnuje větší penzum času a předává požadavky, avšak stále to není ideální Product owner podle SCRUM, a už vůbec neodpovídá požadavku metodiky XP, kdy má být zákazník na pracovišti neustále. Tato situace je řešena komunikací mezi pravým Product ownerem, jeho zastupujícím a mnou. Já jsem tedy v roli definované (Zuzana Štochová a Eduard Kunce, 2014) jako tzv. Product owner proxy, kdy mám určité právo na rozhodování v rámci toho, co již bylo definováno, ale ovšem ne úplné. Zároveň je mým úkolem již zmíněná komunikace se zákazníkem, pokud je třeba nějaký požadavek lépe specifikovat.

CH6: Interní zákazník, který má všechnu pravomoc k rozhodování o vývoji a má velmi omezené časové možnosti. Zákazník má dedikovaného zástupce, který má více času, přesto nedostatečně. Zbylou pravomoc má jeden člen týmu.

Jelikož se jedná o korporátní prostředí, je omezený výběr softwarových nástrojů pro podporu agilního vývoje. Omezení je o to větší, že se jedná o studentský projekt, který jak již bylo zmíněno, nemá budget na libovolný výběr nástrojů. K dispozici je tedy buďto nástroj Team Foundation Server (TFS) od Microsoftu, nebo Excel (který je dle (VersionOne, Inc. 2016) nejvyužívanější). Jako nástroj pro vývojáře je použito MS Visual Studio 2013. Aplikace je psána v jazycích C#, ASP.Net, Javascript (a Javascriptové frameworky jako jQuery, AngularJS, KnockoutJS apod.), HTML a CSS. Pro verzování se nyní používá TFS, kde jsou vytvořené automatické buildy a využívá se u nich CI.

Z hlediska současného přístupu k vývoji se nepoužívá žádná metodika striktně, ale vychází se z metodiky SCRUM. Vývoje probíhá v iteracích po dvou týdnech. Iterace začíná vždy plánování, kde se využívá planning pokeru pro ohodnocení náročnosti jednotlivých položek. Nárazově se využívá denních meetingů. Meetingy shrnující proběhlé iterace se nevyužívají. Přístup ke kódu aplikace mají všichni členové týmu. Dokumentace je minimální, je vedena uživatelská příručka, několikrát za rok je třeba vyplnit firemní dokumenty. Vedeny jsou také dokumenty o právech na úkony v aplikaci pro jednotlivé role.

CH7: Dokumentace na úrovni firemních předpisů a uživatelské příručky.

V projektu se momentálně nevyvíjí nové epic (viz kapitola 4.7.2.11) funkcionality. Aktuálně se vyvíjí vylepšení stávajících funckionalit a především se opravují odhalené chyby a upravuje se aplikace podle nových požadavků. Ideální se tedy jeví a aktuálně praktikované jsou dvoutýdenní iterace. Na konci každé iterace následuje release.

CH8: Dvoutýdenní iterace, kde má přednost údržba aplikace a poté co jsou dokončeny udržovací práce, se vyvíjí nová funkcionalita.

Jelikož se jedná o aplikaci pro oddělení lidských zdrojů, je nezbytné mít funkční bezchybovou aplikaci. V aplikaci se nacházejí citlivé údaje. Tím pádem jsou v aplikaci využívané jednotlivé role pro omezování přístupu do určitých sekcí aplikace.

Pro testování se využívají jednotkové testy a automatizované testy uživatelského rozhraní. Dále se využívá průběžného ručního testování aplikace na testovacím serveru. Vždy po ukončení iterace je aplikace na den dostupná pouze na server UAT, kde je dedikovaný člověk, který otestuje novou funkcionalitu, opravené chyby a zpracované požadavky. Server UAT daný den slouží pro testování členem týmu na prostředí, které je identické s ostrým provozem, akorát na něm nejsou uložena citlivá data.

CH9: Nezbytná bezchybnost aplikace. Provádí se testování na UAT a automatizované testy.

4 Systémový přístup k agilním metodikám

Systémový přístup definuje (Habr a Vepřek 1986, s. 15) takto: *„Za systémový přístup považujeme způsob myšlení, způsob řešení problémů či způsob jednání, při němž jsou jevy chápány komplexně ve svých vnitřních a vnějších souvislostech.“*

Další definice systémového přístupu pochází z (Vodáček a Vodáčková 2009, s. 33) a je definován takto: *„Pod vyjádřením systémový přístup se rozumí účelový způsob celistvého (komplexního, holistického) myšlení, formulování i řešení problémů a interpretace dosažených výsledků rozboru. Znamená to, že zkoumané jevy jsou chápány ve své jednotě vnitřních a vnějších souvislostí. Dochází k účelovému zjednodušení uvažovaných reálných jevů nebo procesů, ale s dodržením požadavku na požadovanou přesnost, spolehlivost a vypovídací schopnost závěrů rozboru.“*

Systémový přístup tedy zdůrazňuje účelovost pohledu.

Jelikož existuje velké množství agilních metodik a rozsah této práce neumožňuje zkoumat podrobně všechny metodiky, je z pohledu účelu této bakalářské práce vybráno několik hlavních metodik a přístupů a ty jsou zkoumány.

Na základě výše uvedené definice systémového přístupu jako způsobu myšlení, způsobu řešení problémů a způsobu jednání, jsou u zkoumaných metodik zvažovány různé faktory a aspekty pohledu.

Pro některé ukázky aplikačního prostředí podporující agilní metodiky byl vybrán software Team Foundation Server od Microsoftu, jelikož je v rámci licence dostupný studentům VŠE zdarma a jedná se o software, který je dle průzkumu (VersionOne, Inc. 2016) využívaný 24 % respondentů.

4.1 Faktory pro výběr zkoumaných metodik

Pro výběr zkoumaných metodik jsou pro tuto práci důležité dva faktory.

Prvním faktorem je možnost aplikace metodiky ve firmě. Jako ve většině velkých firem jsou zde omezení ze strany licencí (souvisí s rozpočtem projektu), proto je třeba dbát na to, jaké firma umožňuje používat vývojové prostředí a zdali jsou dané metodiky pro toto prostředí použitelné i bez dalších finančních nákladů. Tento problém odpadá se zvětšující se velikostí projektu, kde je více finančních prostředků. Avšak, jak již bylo zmíněno, zde se jedná o studentský projekt, tudíž je třeba brát na tuto stránku věci větší zřetel.

Druhým faktorem je popularita metodiky. Čím více je daná metodika pro vývoj softwaru využívaná, tím je snazší dohledat si o dané metodice více informací a v případě problémů s aplikací najít vhodné řešení. Tato práce vychází z nejaktuálnějšího průzkumu The 10th Annual State of Agile™ Report (VersionOne, Inc. 2016), který byl zveřejněný začátkem dubna 2016. Dle toho průzkumu 58 % firem využívá metodiku SCRUM, 10 % využívá SCRUM/XP (samostatný XP využívá jen 1 %), na třetím místě jsou hybridně využívané metodiky, na čtvrtém a pátém místě jsou Scrumban (7 %) a Kanban (5 %) a na dalších místech jsou rozdíly již nepatrné.

S ohledem na předchozí dva faktory se tato práce bude zaobírat především metodikami SCRUM, XP (SCRUM/XP), Kanban (Scrumban) a Lean Software Development, jelikož odpovídají v praxi nejvyužívanějším metodikám a jejich aplikace není zpoplatněna.

V první části této kapitoly jsou vysvětleny principy agilních metodik. Jedná se o principy, které jsou všeobecně platné pro všechny agilní metodiky.

V dalších částech této kapitoly jsou jednotlivé metodiky popisovány a analyzovány. První popisovanou a analyzovanou metodikou je metodika SCRUM. U dalších metodik jsou poté popsány jejich charakteristické rysy, artefakty a principy a jsou spíše porovnávány s obecnými principy agilních metodik, nebo s jinými agilními metodikami, především metodikou SCRUM. S touto metodikou z toho důvodu, že je popsána jako první, ale především je na ní kladen vzhledem k procentuálnímu využívání.

Nejdůležitější ovšem je, si všimnout rozdílu mezi jednotlivými metodikami.

4.2 Úhly pohledu na agilní metodiky

Na problematiku agilních metodik se lze dívat z mnoha úhlů pohledu.

Jeden z úhlů pohledu vychází z rolí v týmu. Ve zkratce se jedná o pohled manažera, programátora, testera a zákazníka. K tomu některé metodiky mají přímo své definované role, např. ve SCRUM metodice se nerozdělují role na programátora a testera, ale jedná se o tým.

Dalším pohledem může být pohled přes proces vývoje SW. Některé metodiky tento pohled řeší více a některé jsou naopak pouze jakýmisi rámci, které lze určitým způsobem ohýbat.

Důležitým pohledem pro tuto práci jsou omezení při aplikaci dané agilní metodiky z hlediska korporátních pravidel (všeobecně pravidel ve firmách). V korporátních organizacích je velmi mnoho nařízení, pravidel, atd., která je třeba dodržovat, a která mohou danou metodika zcela vyřadit z výběru. Důležitým aspektem pro velké firmy je i možnost dohledu. Jelikož ve velkých firmách je dost často hlavním měřítkem úspěšnosti projektu možnost kontroly projektu ze strany manažera.

Při aplikaci či přestupu na novou metodiku je třeba zvážit možnosti a omezení vycházející z dosud používané metodiky. Dost často je snazší aplikovat metodiku na nový projekt, než přecházet v projektu z jedné metodiky na jinou. I když podle (Kadlec 2004, s. 153) je třeba metodiku SCRUM jednodušší aplikovat na organizace, kde se již podle určité metodiky vyvíjí a SCRUM použít jako způsob vedení projektů.

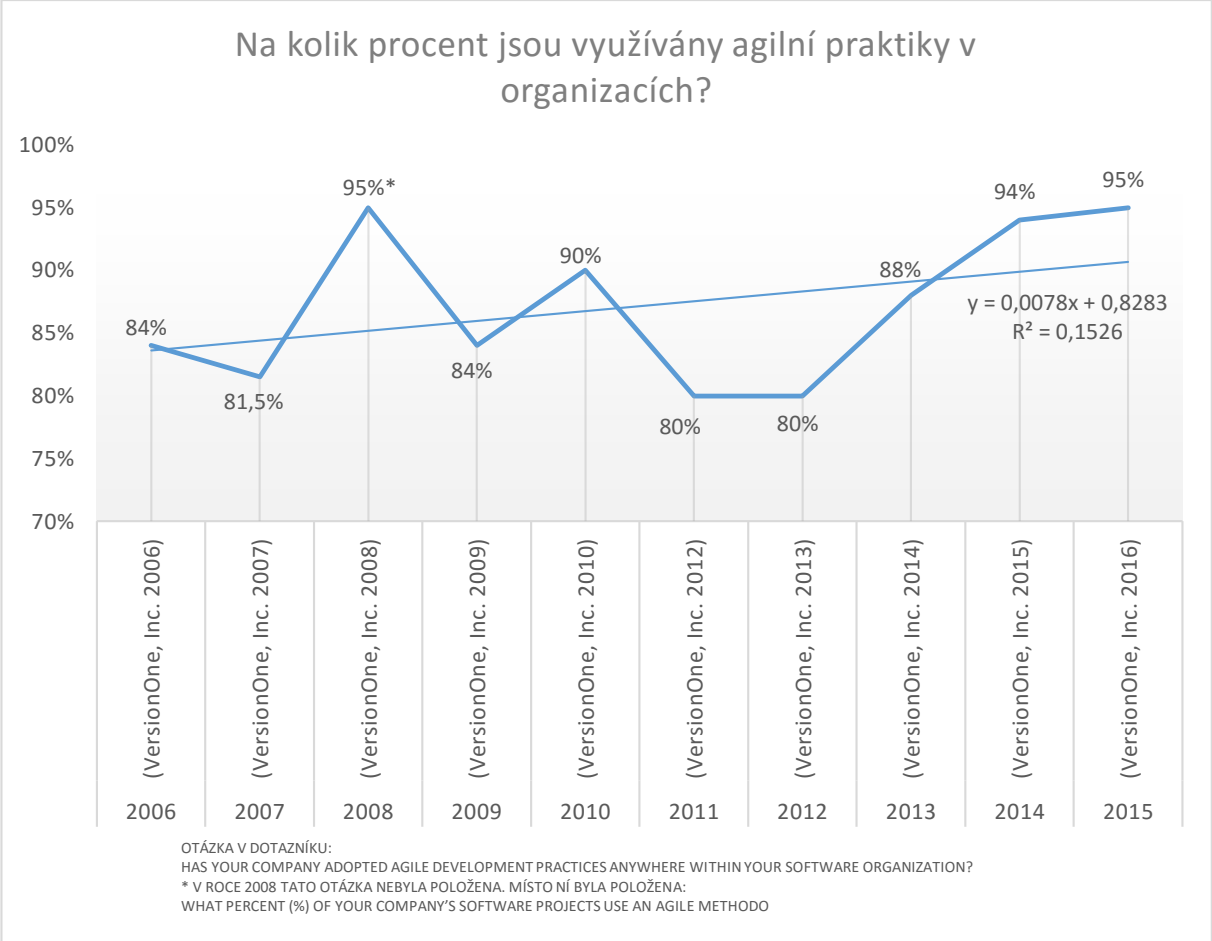
Všechny tyto úhly pohledu vycházejí z výše zmíněných charakteristik týmu, které je třeba brát v potaz.

4.3 Současný stav využívání agilních metodik

Celosvětové statistiky o využívání agilních metodik vydává společnost VersionOne, které vychází z jejich dotazníkových průzkumů The State of Agile. Na následujícím grafu Graf 1, můžeme vidět procentuální vývoj využívání agilních metodik. V dotazníku z roku 2008 (VersionOne, Inc. 2008) nebyla položena stejná otázka jako v ostatních letech, proto bylo nutné v práci využít jiné otázky a z těch vycházet. V těchto statistikách je vidět, mimo

odchýlené roky 2008 a 2010, postupné stoupání procentuálního využívání agilních metodik. Tyto odchylky mohou být způsobené měnícími se dotazovanými firmami a jejich složením, jelikož například v dotazníku z roku 2009 (VersionOne, Inc. 2009) bylo oproti dotazníku z roku 2008 70 % dotazovaných firem nových.

Stoupající trend je vidět i na vložené spojnici trendu (viz Graf 1).



Graf 1: Vývoj využívání agilních metodik (Zdroj: autor, dle VersionOne)

V České Republice byly provedeny dva rozsáhlejší průzkumy o využívání agilních metodik. První průzkum byl proveden v roce 2006 (Alena Buchalcevová a Marek Leitl, 2006). Data z tohoto průzkumu jsou již zastaralé, ale mohou sloužit jako úvodní pohled na využívání agilních metodik v ČR.

V tomto dotazníku uvedlo 57 % respondentů, že využívají firemní standardy. Druhou nejvyužívanější metodikou je RUP s 19 %. Dále 14 % firem nevyužívá žádné metodiky. Nejvyužívanější agilní metodikou bylo XP s 5 %. Pod pojmem firemní standardy se může skrývat jak tradiční přístup tak agilní. Toto pomáhá osvětlit jiná část dotazníku, která se zabývala mírou agility vývoje ve firmě podle přikládaného významu podstatným prvkům agilního vývoje, kde se většina firem pohybovala mezi 48-62 % v takzvaném indexu agility. Z toho vyplívá, že nějaké agilní prvky využívané jsou, ale o značnou převahu se nejedná.

První je vhodné porovnat užívání agilních metodik ve stejném roce, tedy 2006, ve světě a v ČR. Již zmíněná společnost VersionOne vydala svůj první průzkum „State of agile“ též

v roce 2006 (VersionOne, Inc. 2006). V tomto dotazníku se vyjádřilo 84 % respondentů, že využívá agilní metodiky. Z toho 40 % metodiku SCRUM, která ve stejném roce v ČR ani nebyla zmíněna. Metodiku XP označilo jako primárně využívanou 23 % dotázaných firem, což je více než čtyřnásobek oproti ČR. Samozřejmě je nutné brát v potaz, že průzkumu v ČR se účastnilo pouze 21 firem a celosvětového průzkumu 722 firem. Je zde tedy možná statistická odchylka, ale i přesto má toto porovnání svou hodnotu.

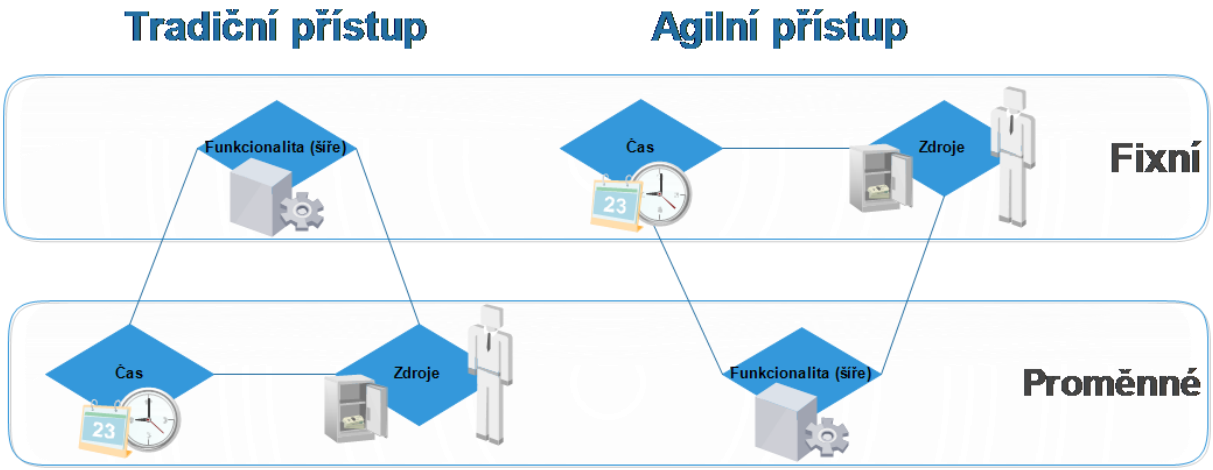
Druhý významnější průzkum, a zároveň nejaktuálnější byl v ČR proveden v roce 2013 společností Etnetera (Dalibor Pulkert, 2014). Tohoto průzkumu se účastnilo 171 firem. Z tohoto počtu se 81 % vyjádřilo, že používají agilní přístupy ve své firmě a 17 %, že agilní přístupy neuplatňují. Zbytek dotazovaných uvedl, že neví, nebo že nejsou kompetentní toto hodnotit. Další zajímavou statistikou je, že 91 % dotázaných se vyjádřilo, že používají agilní přístupy 5 let a méně, což koresponduje s dotazníkem z roku 2006. V tomto průzkumu se již projevila nadvláda metodiky SCRUM, kterou praktikuje 87 % dotázaných, XP pouze 13 %. Na druhém místě je Kanban s 33 %. Je třeba zmínit, že se nejedná o statistiku jediné využívané metodiky, ale o to jaké frameworky firma používá, tedy že firma může používat i více metodik. Porovnání průzkumů z roku 2006 a 2013 v ČR dokazuje, že zavedenost agilních metodik a přístupů do firem za toto období dramaticky vzrostlo.

Ve stejném roce tedy 2014 vyšel průzkum od společnosti VerisonOne a to konkrétně 8th Annual State of Agile™ Survey (VersionOne, Inc. 2014). Tento průzkum měl 3 501 respondentů, kde se vyjádřilo 88 %, že používají agilní metodiky. Porovnání těchto celosvětových 88 % a výše zmíněných 81 % v ČR nám tvoří rozdíl zavedenosti pouhých 7 %. Kdežto rozdíl zavedenosti v ČR (Alena Buchalcevová a Marek Leitl, 2006) oproti celosvětovému průměru z roku 2006 (VersionOne, Inc. 2006) byl vyšší (nedá se přesně určit, kvůli nejednotnosti položené otázky, ale dalo by se říci, že se jedná cca o rozdíl 30 %.) V nejaktuálněji vydaném průzkumu od VersionOne z roku 2016 (VersionOne, Inc. 2016), kde bylo dotazování 3 880 firem v období od července 2015 do listopadu 2015, odpovědělo 95 % respondentů, že používá agilní přístup. To, že zmíněných 95 % firem používá agilní přístupy, je důležité číslo, ale podstatné je osvětlit jaký podíl ve firmách mají agilní metodiky. Aktuálně je ve firmách 43 % týmů, kde převládá agilní přístup a 4 % týmů, které pracují zcela bez agilního přístupu. Tato čísla vypovídají o míře nasazení agilních principů. Ve většině firem již jsou agilní metodiky zavedeny, takže otázka zavádění již není tak aktuální. Oproti tomu jsou aktuálnější otázky, které se zabývají tím, které nástroje pro agilní vývoj použít, jaké techniky využívat a všeobecně jak zlepšit agilní vývoj. Dalším důležitým faktorem je čas, po který firmy agilní přístup již aplikují. Podle průzkumu 25% firem aplikuje agilní metodiky více než pět let, 32 % tři až pět let a méně jak dva roky 43 %. Tato poslední skupina může být značně problematická, protože pokud si agilní metody aplikují ve firmě samy, bez pomoci zkušených vývojářů či Scrum masterů a podobných specializovaných rolí, mohou nastat problémy způsobené nedostatečnými zkušenostmi.

4.4 Charakteristika agilních metodik

Charakteristické rysy agilních metodik vycházejí z agilního manifestu (K. Beck, M. Beedle, A. van Bennekum, et al. 2001). Jedná se o dvanáct principů, které byly sepsány v roce 2001.

Podle (Kadlec 2004, s. 119) je zásadní rozdíl mezi rigorózními metodikami vývoje a agilními metodikami vývoje určen třemi body a těmi jsou: čas, zdroje a funkcionalita (někdy popisována jako šíře, či rozsah zadání). Při vývoji podle rigorózních metodik jsou proměnnými čas a zdroje. Fixní je naopak funkcionalita (viz Obr. 3). Tato teorie vychází z předpokladu, že je jasně zadáno, co daný produkt má umět, ale není známo, jak dlouho bude jeho vývoj trvat a jaká bude jeho finální cena.



Obrázek 3: Tradiční versus agilní přístup (Zdroj: autor, dle [Kadlec, 2004, str. 119])

Oproti tomu u vývoje podle agilních metodik jsou fixní čas a zdroje a naopak funkcionalita je proměnná. Na začátku je tedy stanoveno zadavatelem projektu kolik peněz je ochoten investovat a jak dlouho může na finální produkt čekat. Předem ovšem není známo, jaké funkcionality se za těchto okolností dosáhne. Ale díky tomu, že se jedná o agilní vývoj, je možné pružně reagovat na priority, či změny a to i tím způsobem, že zadavatel přehodnotí zdroje, které chce do projektu investovat a tím pádem může projekt narůstat. Zásadní rozdíl je zásadní v tom, že si výši vložených zdrojů zadavatel určuje sám a není mu jejich výše na konci pouze předložena.

Druhou podstatnou vlastností všech agilních metodik je iterativní a inkrementální vývoj. Tyto vlastnosti znamenají, že je zákazníkovi dodáván produkt průběžně v podobě různých funkčních přírůstků. Toto je v současné době velmi důležité, jelikož je vyvíjen stále větší tlak na rychlost vývoje a pokud jedna firma technologicky zaspí, je dost možné, že bude rychle překonána konkurencí. Příkladem může být aplikace Instagram. Jedná se aktuálně o jednu z nejpoužívanějších sociálních sítí, kde je vývoj nových funkcí a vylepšování produktu zásadní. V článku (Malan Joubert, 2016) se vyjádřil Michael Groven, produktový inženýr Instagramu, že jeho tým provádí release čtyřicetkrát denně. Jedná se sice o jednu z největších internetových aplikací, takže nelze toto číslo aplikovat automaticky na všechny produkty, i přesto je zde vidět ona potřeba neustále vyvíjeného produktu. Dochází tak k předcházení dodávky produktu ve formě velkého třesku a zákazník není nucen věnovat novým verzím tolik úsilí.

Třetí velmi důležitou charakteristikou je důraz na komunikaci. A to ve dvou směrech – směrem k zákazníkovi a uvnitř týmu. Týmová komunikace je dost často přímo implementována v agilních metodikách. Příkladem můžou být Daily scrums v metodice SCRUM a párové programování v metodice XP. Zlepšením komunikace v týmu se předchází jak problémům uvnitř týmu, tak problémům v celém projektu. Komunikace směrem k zákazníkovi zase zlepšuje rychlost reakce na změny a případně vzniklé problémy.

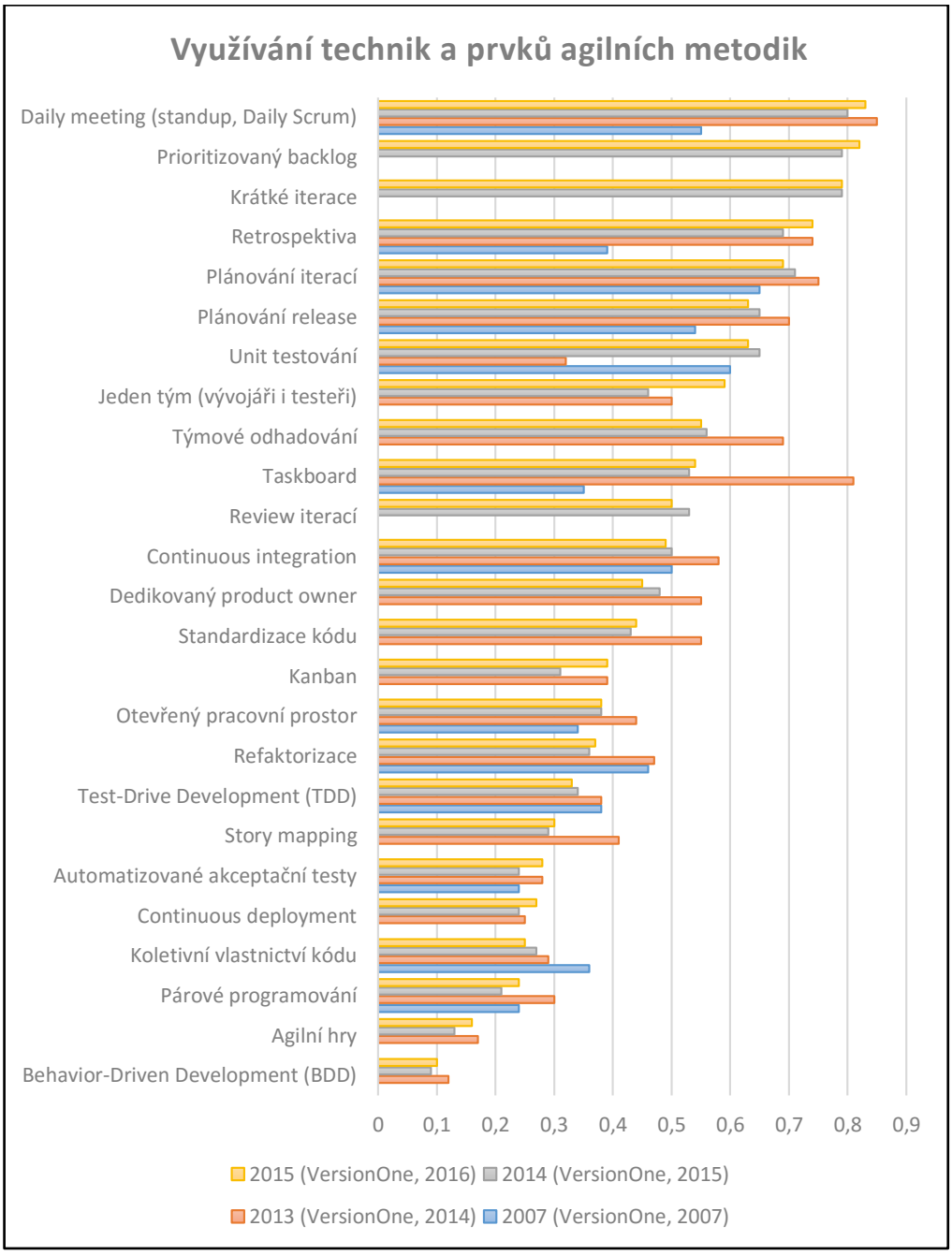
V pořadí čtvrtou charakteristikou agilních metodik je důraz na průběžné testování. Využívá se v hojné míře automatizované testování a to jak tzv. Unit testů, tak automatizované testování uživatelského rozhraní pomocí např. Seleniových testů. Pokud je v týmu člověk specializovaný na testování, dochází k jeho využití po celou dobu vývoje a ne jen ve formě nárazových testování, jak tomu bylo při využití rigorózních metodik.

Pátým znakem agilních metodik je kladení důrazu na kód oproti dokumentaci. V principu to znamená, že pokud je vytvářen kvalitní a čitelný zdrojový kód, tak může zároveň sloužit jako nosič informací a není třeba tak velké množství dokumentace. V mnoha metodikách je přímo uvedeno, kdo za kód nese zodpovědnost. V metodice SCRUM je to ten, kdo kód psal, oproti tomu v metodice XP jsou všichni členové týmu zodpovědní za každou část kódu. V metodice XP je také využíváno již zmíněného párového programování.

4.5 Prvky agilních metodik

Popis prvků a technik agilních metodik je rozčleněn mezi jednotlivé metodiky, ze kterých vznikaly, nebo ke kterým se nejčastěji váží. Pro orientaci v těchto prvcích, které je možné aplikovat napříč různými metodikami, je vhodné si udělat jejich soupis, případně si některé prvky dovysvětlit.

Mezi nejpoužívanější prvky a techniky agilních metodik podle (VersionOne, Inc. 2016) patří:



Graf 2: Vývoj využívaní agilních praktik (Zdroj: autor, dle VersionOne)

Pozn. Za každou technikou či prvkem agilní metodiky je v závorce uvedena kapitola, ve které je daný prvek popsán.

V grafu jsou zobrazeny i statistiky z dotazníků z roků 2015 (VersionOne, Inc. 2015), 2014 (VersionOne, Inc. 2014) a pro ilustraci i z roku 2007 (VersionOne, Inc. 2008) kdy poprvé společnost VersionOne začala dané techniky zohledňovat.

Mezi dotazníky z roků 2016 a 2015 nejsou skokové rozdíly, paradoxně ve většině případů nasazení dopadá dotazník z roku 2014 lépe než z roku 2016. Toto však může být způsobeno typem dotazovaných firem (v dotazníku z roku 2016 bylo jen 26 % firem ze softwarového odvětví, kdežto v dotazníku z roku 2014 to bylo už 81 %), počtem dotazovaných firem a lokací dotazovaných firem (v roce 2014 bylo mezi dotazovanými firmami 86 % ze Severní Ameriky a z Evropy, kdežto v dotazníku z roku 2016 jen 82 %.)

4.6 Všeobecné výhody a nevýhody agilních metodik

Mezi obecné nevýhody agilních metodik patří jednoznačně nároky na vývojářský tým. Je nutné buďto najít vhodné lidi pro tento způsob vývoje, nebo investovat dostatek energie na přesvědčení členů týmu, že tyto metodiky mají své opodstatnění a jsou přínosné. Pokud se v týmu najdou lidé, kteří nebudou s filosofií agilních metodik ztotožněni, mohou celý tým a agilní přístup rozkládat. Vyšší nároky na vývojářský tým jsou především v oblasti sebeřízení, přebírání zodpovědnosti a schopnosti se správně rozhodovat.

Velký důraz je též kladen na bezproblémovou komunikaci a to jak v rámci týmu, tak se zákazníkem. Co se zákazníka týče, je nutné s ním být, oproti klasickým – rigorózním metodikám, v úzkém kontaktu. Některé metodiky v čele s XP dokonce vyžadují jeho neustálou přítomnost na pracovišti. Velmi často se stává, že zákazník na vývojáře nebo IT všeobecně, nahlíží jako na někoho, kdo se ho snaží připravit o peníze a je těžko kontrolovatelný. V agilním přístupu je nutné zákazníka přesvědčit, že jsou vývojáři na jeho straně a snaží se mu pomoci. Díky agilnímu přístupu je zákazník také daleko více angažovaný a má větší kontrolu nad přírůstkem produktu. Agilní metodiky se nesnaží přijít s rozpočtem na celý projekt. To se sice snažily rigorózní metodiky, avšak podle (Zuzana Štochová a Eduard Kunce, 2014) se to 70% projektům nepodařilo. Zde se tedy týmy soustředí na odhadnutí počátečních nákladů, mezi které patří například licence, nástroje, apod. a poté má zákazník možnost ovlivňovat, kdy, jak a co se bude vyvíjet. Jelikož jsou agilní metodiky vedené v iteracích, může zákazník vývoj ukončit v polovině odhadované doby celého projektu, a přesto dostane funkční verzi produktu, která může splňovat jeho nejzákladnější požadavky.

Zde se vlastně překlápí výčet nevýhod, na výčet výhod. Z pohledu zákazníka již byla značná část výhod zmíněna. Mezi logickou, ale ještě nezmíněnou výhodu, také patří možnost okamžité reakce zákazníka. Zákazník může během vývoje reagovat na tržní změny, různé nepředvídatelné okolnosti nebo změny požadavků na produkt. Pro zákazníka je též výhodné dostávat jednotlivé přírůstky produktu postupně a ne až jako komplexní balík na konci, z pohledu implementace. Je snazší zaučovat uživatele produktu každý měsíc o nové přidané funkce, než jim předat jednou za několik let zcela nový software a zaučovat je úplně od nuly.

4.7 SCRUM

V této kapitole je popsána metodika SCRUM, její charakteristika, principy, techniky a artefakty. Jako základní zdroj této kapitoly slouží dokument *Průvodce Scrumem* od autorů samotné metodiky Kena Schwabera a Jeffa Sutherlanda (Schwaber a Sutherland, 2013).

4.7.1 Historie

První zmínka o slově SCRUM sahá do roku 1986 kdy Hirotaka Takeuchi a Ikujiro Nonaka publikovali článek New New Product Development Game (Hirotaka Takeuchi a Ikujiro Nonaka, 1986). Zde použili analogii slova scrum, které se používá pro skrumáž v rugby. Český termín pro scrum je mlýn.

Druhým důležitým milníkem pro SCRUM je konference OOPSLA v roce 1995, kde Ken Schwaber a Jeff Sutherland prezentovali a následně publikovali práci SCRUM Development Process (Ken Schwaber a Jeff Sutherland, 1995).

Třetím a nejdůležitějším historickým milníkem pro SCRUM je rok 2001. V tomto roce byla prezentována kniha Agile Software Development with Scrum (Schwaber a Beedle, 2001). Tento rok byl přelomový pro agilní metodiky všeobecně, neboť byl zveřejněn Agilní Manifest (K. Beck, M. Beedle, A. van Bennekum, et al., 2001) jak již bylo zmíněno výše. Dalšími důležitými milníky bylo založení Scrum Alliance v roce 2002, s čímž přišla oficiální certifikace ScrumMaster a založení scrum.org v roce 2009.

Toto je pouze souhrn nejdůležitějších prací a institucí. Metodika SCRUM se však vyvíjela i díky dalším důležitým publikacím a dle (scrumguides.org, 2016) bylo již vydáno více než 1000 knih na toto téma. Toto číslo zcela jistě není konečné, jelikož se jedná o nejvíce využívanou metodiku a publikací stále přibývá.

4.7.2 Pojmy a role

Aby bylo možné chápat následující text, je nutné si definovat základní pojmy a role v metodice SCRUM.

4.7.2.1 Product Owner

Product Owner (vlastník produktu) určuje směr vývoje. Má na starost celý Product backlog, což znamená, že určuje priority a musí jasně formulovat jednotlivé položky tak, aby jim vývojový tým rozuměl. Vždy se jedná o jednu konkrétní osobu.

4.7.2.2 Scrum master

Scrum master je v podstatě moderátor celé metodiky SCRUM. Jeho úlohou je vést tým tak, aby dodržoval a postupoval podle metodiky, jejích technik a pravidel. Dá se říci, že se jedná o vedoucího týmu, avšak ne o klasického managera. Klasický manager je postavený nad týmem a dost často se dostává do situace, že jeho podřízení jsou mu v opozici. Pro Scrum mastera je zásadní, aby ho členové týmu chápali jako spoluhráče a fungovala zde absolitní důvěra. Scrum master se vlastně snaží pomáhat na všechny strany. Vlastníkovi produktu pomáhá s Product backlogem, jeho správou a správným plánováním. Vývojový tým vede k sebeorganizaci a multifunkčnosti. Řeší problémy, které mohou bránit týmu ve vývoji. Moderuje meetingy.

4.7.2.3 Tým

Třetí rolí jsou vývojáři ve *Vývojovém týmu.* Tým by měl být ve velikosti 5-9 lidí. Tým je sebeorganizujícím článkem, kde je kladen vysoký důraz na profesionalitu jednotlivých členů, kteří mohou mít různá zaměření, neexistuje zde však rozdělení na jiné role než vývojář.

4.7.2.4 Stakeholders

Další rolí jsou takzvaní *stakeholders.* Dle (Schwaber, 2007) jedná se o všechny lidi, kteří mají zájem na projektu. Všichni stakeholdeři i zákazníci jsou zastoupeni Product ownerem.

Mezi pojmy či artefakty metodiky SCRUM lze zařadit například.

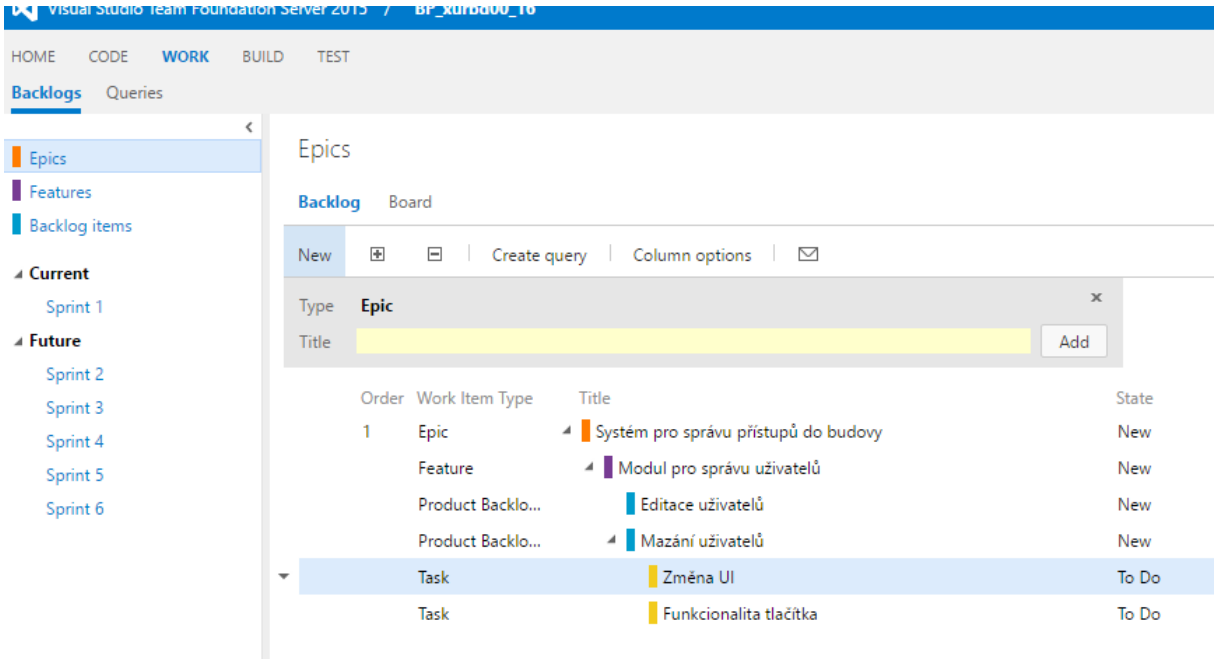
4.7.2.5 Velocity

Velocity neboli rychlost sleduje výkonost Sprintů. Po dokončení daného sprintu se sečte ohodnocení dokončených položek (více viz kapitola 4.7.4.1) a tento údaj udává velocity za daný sprint.

Žádoucí je, aby tým měl v průběhu Sprintů stabilní velocity a byl tedy předvídatelný. Důležité je, že se měří pouze týmové velocity a ne volocity jednotlivce. Nemělo by se také jednat o metodu, kterou bude využívat manažer na hodnocení týmu. Tým by poté mohl nereálně odhadovat položky a manažer by tak ztratil důležitý ukazatel výkonnosti týmu. Pokud tým nepodává z tohoto pohledu stabilní výkon, je třeba zjistit důvod. Mezi nejčastější důvody patří nesehranost týmu – tým je nestabilní, neumí příliš odhadovat apod. Dalším důvodem může být velký počet chyb, které je třeba opravit, technický dluh nebo učení se nové technologie a praktik. Taktéž je těžké určit dosažitelnou hodnotu velocity při prvních Sprintech, kdy ještě tým nemůže vycházet z minulosti.

4.7.2.6 Product backlog

Product backlog je seznam všech položek, které je třeba vyvinout. V češtině je někdy označován jako produktový backlog. Každá položka na seznamu má svou prioritu, kterou určuje vlastník produktu. Produkt backlog nikdy nemůže být kompletní, jednotlivé položky se z něj odebírají (tím, že se dokončují) a nové požadavky přibývají. Pro každou položku je důležité určit její prioritu, jednoznačný popis a odhad náročnosti. Nejjednodušší názorná ukázka backlogu je na obrázku Obr. 4. Podobně vypadá i Sprint backlog.



Obrázek 4: Ukázka Product backlog (Zdroj: autor)

4.7.2.7 Sprint backlog

Posledním důležitým pojmem je *Sprint backlog*. Do tohoto seznamu se přendávají položky z Product backlogu, které jsou potřebné pro splnění cíle sprintu. O tom, které položky se do Sprint backlogu zařadí, rozhoduje Product owner, který ale musí brát v potaz odhad vývojového týmu v tom směru, co lze za daný sprint dodat. Položky ve Sprint backlogu jsou již detailně popsány tak, že je jasné co je třeba udělat, aby daná položka mohla být dokončena. Stejně jako Product backlog se Sprint backlog v čase mění, tím jak se doplňují a upravují jednotlivé položky směrem k cíli sprintu. Dalším atributem, který se mění, jsou časové odhady, které je třeba vždy aktualizovat podle reálných možností. Pokud nějaká položka byla odhadnuta na 8 hodin práce a vývojář na ni pracoval 6 hodin, neznamená to, že nyní automaticky zbývají 2 hodiny práce. Je třeba vždy odhadnout, kolik reálně zbývá. Na tomto příkladu, se mohla objevit překážka, která vývojáři zabrala 4 hodiny práce navíc, tím pádem se odhad zmenší pouze na 6 hodin zbývající práce. Paradoxně se tedy může stát, že po celodenní práci vývojáře dokonce může nastat zvýšení odhadované délky práce na položce. Odebírat a přidávat položky do Sprint backlogu může během sprintu pouze vývojový tým. Tato potřeba může vzniknout, pokud se během sprintu objeví, že některá z položek ve Sprint backlogu je již zastaralá či nepotřebná, nebo v opačném případě pokud je některá položka nezbytná k splnění cíle sprintu a je třeba ji přidat.

4.7.2.8 User story

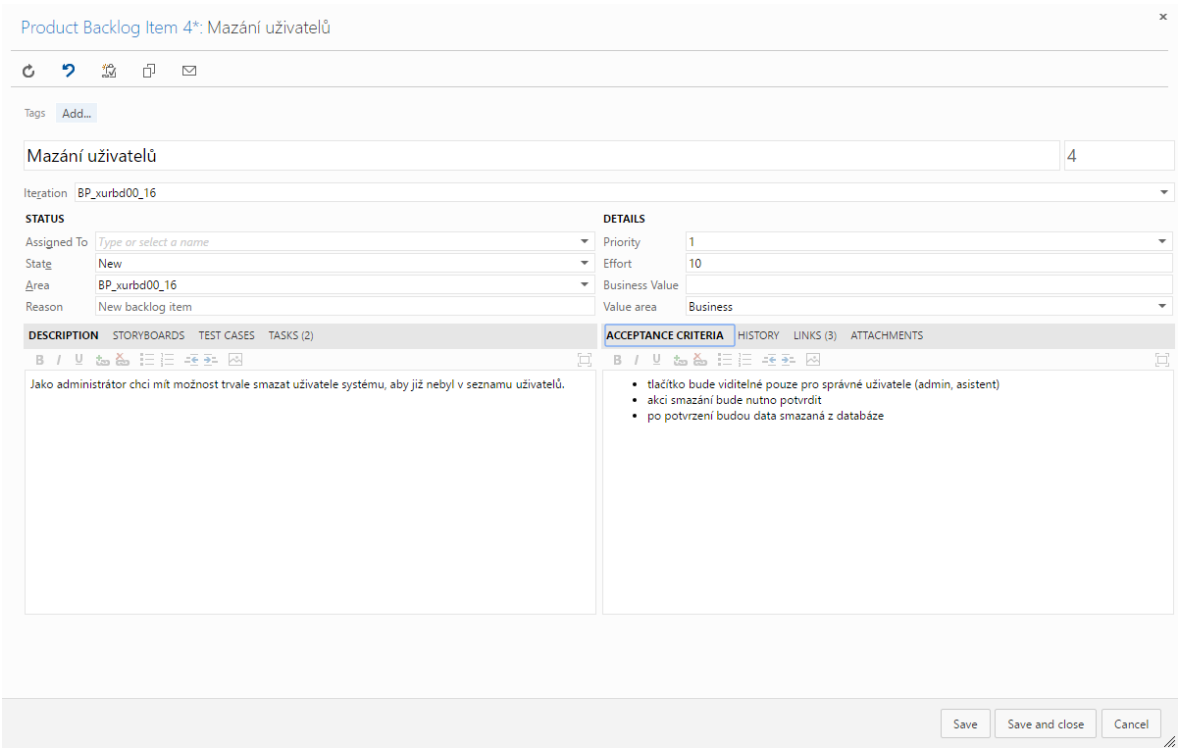
Položky v Product backlogu se nazývají *User story* (v TFS je použit název Backlog Item). Pokud se zkombinuje obsah User story dle (Zuzana Štochová a Eduard Kunce, 2014) a informací z TFS, měly by být k User story přiřazeny zhruba následující údaje:

- ID
- odkaz na Epic (vysvětleno níže)
- odkaz na Super user story (viz níže)
- jméno US
- user story (popis funkcionality)
- akceptační kritéria (viz níže)
- informace o testech
- attachment (přílohy, například obrázek či diagram jak má funkcionalita vypadat)
- komentáře
- priorita
- estimates (zbývající čas práce, effort)

Nejběžnější zápis User story je:

*„**Jako** role **chci** funkcionalitu**, abych** dostal business value.“*

Příkladem může být (viz Obr. 5): *„Jako administrátor chci mít možnost trvale smazat uživatele systému, aby již nebyl v seznamu uživatelů.“*



Obrázek 5: Ukázka User story a její vlastnosti (Zdroj: autor)

Jednoduše lze tedy rozdělit User story do tří otázek: Kdo, Co, Proč.

Vlastnosti User story by měli být INVEST, tedy:

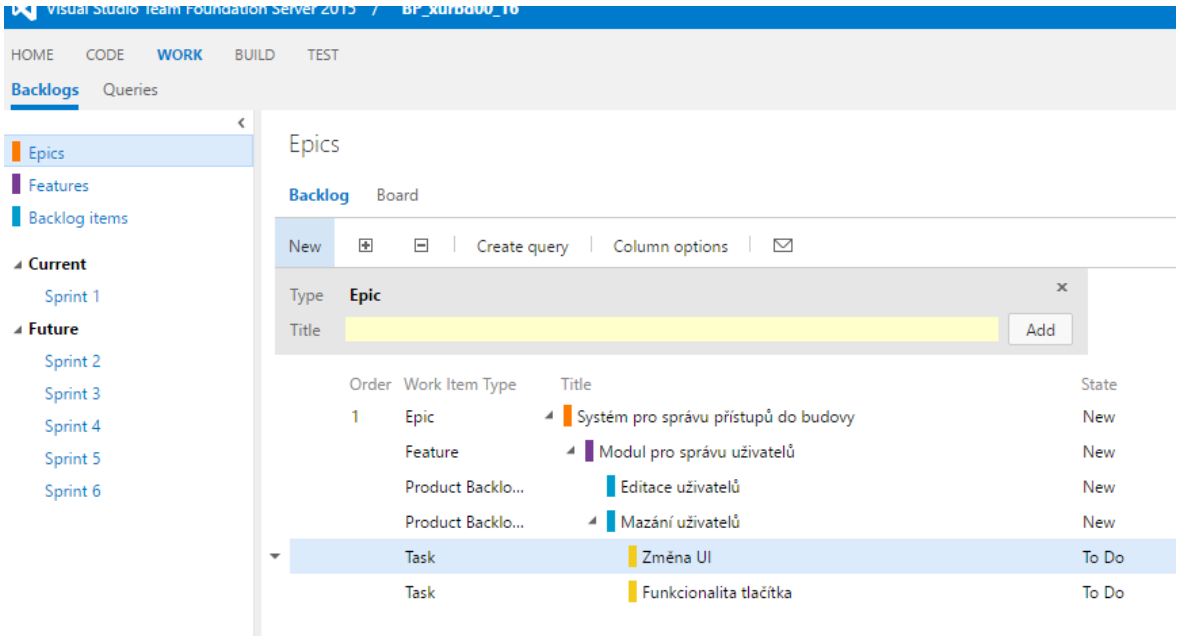
- Independent (nezávislá)
    - V této hodnotě jde především o nezávislost jednotlivých User story mezi sebou.
- Negotiable (popsatelná)
    - Zde jde o to aby Product owner dokázal správně vysvětlit User story. Vysvětlení by mělo být stručné a jednoznačné.
- Valuable (hodnotná)
    - Jde o hodnotu pro zákazníka. Každému členu týmu musí být jasné, proč se daná funkcionalita vyvíjí.
- Estimable (ohodnotitelná)
    - Důležité je též, aby User story byla náročnostně, či časově ohodnotitelná.
- Small (malá)
    - User story by neměla zabrat více než polovinu sprintu. Pokud je větší, měla by se rozdělit do více částí.
- Testable (testovatelná)
    - User story by měla být testovatelná, tedy by mělo být zřejmé, kdy je hotová. Tato vlastnost souvisí s dobře nadefinovanými akceptačními kritérii.

Podstatou User story je zachytit hodnotu pro zákazníka. V User story se neřeší technologické aspekty. Ty přicházejí na řadu až po rozpadu User story na jednotlivé úlohy (Tasky).

4.7.2.9 Task

Tasky jsou již jednotlivé kroky pro vývojáře, které je třeba udělat, a zabývají se technologickou částí. Pro výše zmíněný příklad User story, kdy je požadována možnost mazat uživatele by tasky mohli být následující (viz Obr. 6):

- Změna UI / přidání tlačítka (v popisu Tasku může být například návrh, jak by tlačítko mělo vypadat a kde by mělo být umístěno).
- Funkcionalita tlačítka (v popisu bude, že kliknutím na tlačítko dojde k vymazání uživatele z databáze)

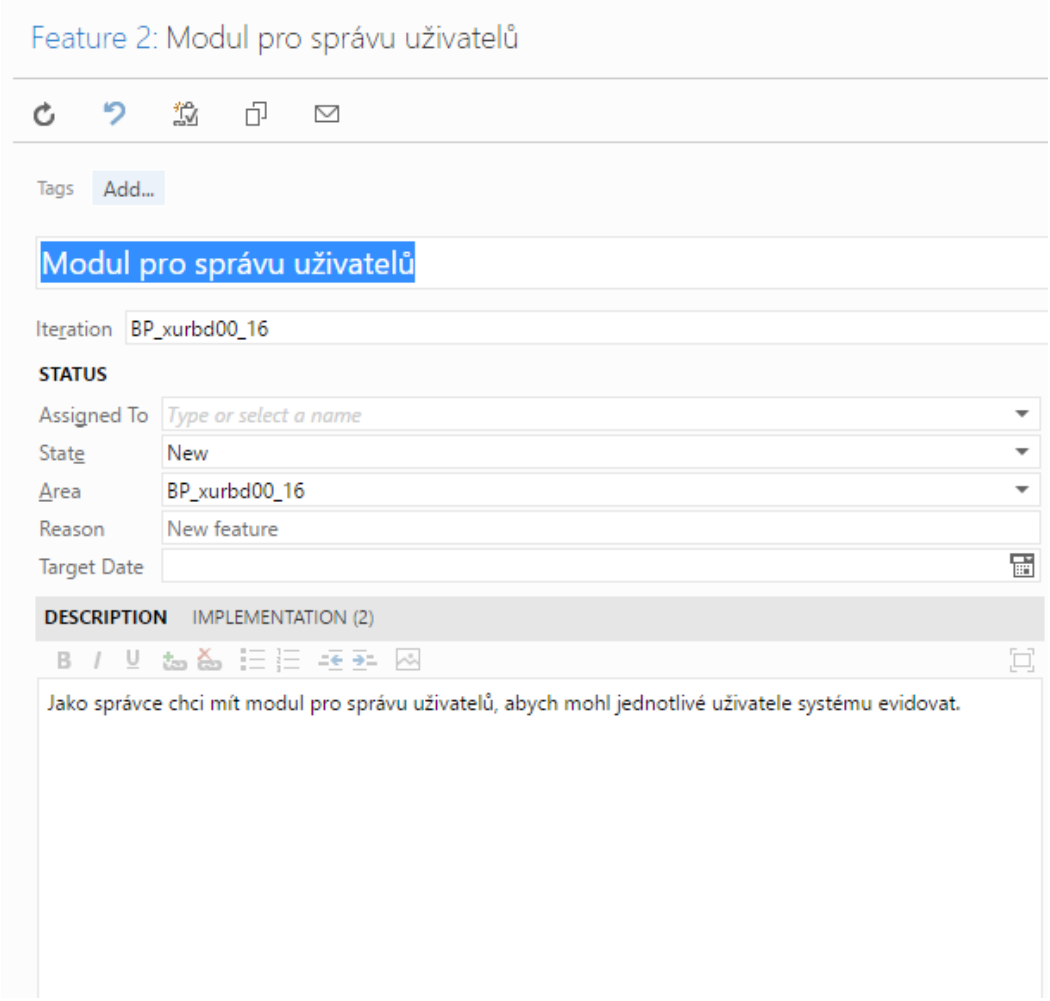


Obrázek 6: Ukázka Tasku (Zdroj: autor)

Tasky je možné rozkládat do dalších Tasků pro detailnější popis, avšak rozdělení User story do jednotlivých Tasků je ve většině případů dostačující. Tímto je popsáno rozkládání do nižších úrovní, než je User story. Opakem je sdružování jednotlivých User story do logických shluků. Ty se obecně nazývají Super user story (v TFS jsou nazývány jako Feature).

4.7.2.10 Super user story

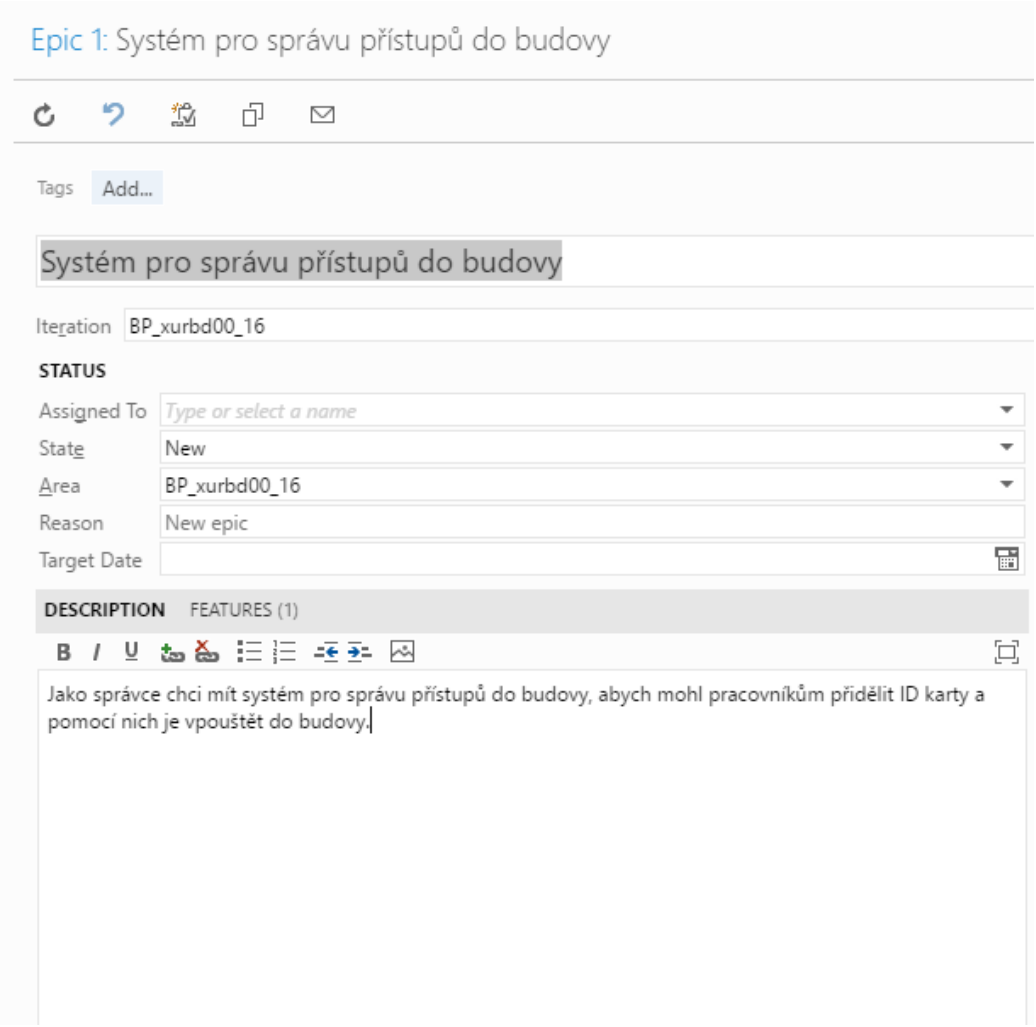
Super user story již nemusí striktně dodržovat zásady INVEST. Jedná se spíše o přehlednost v Product backlogu díky seskupování jednotlivých User story. Pokud by se pokračovalo ve výše zmíněném příkladu, mohla by Super user story vypadat následovně (viz Obr. 7): *„Jako správce chci mít modul pro správu uživatelů, abych mohl jednotlivé uživatele systému evidovat.“* Pod touto Super user story by byla již zmíněná User story, ale i další jako například: *„Jako asistent administrátora chci mít možnost upravovat údaje uživatelů.“* Super user story by nám tedy zastřešovala modul pro správu uživatelů. Pod ní by byly jednotlivé User story, které jsou rozloženy do Tasků. Nad tím vším je možné mít přiřazeny Epic.



Obrázek 7: Ukázka Super User story (Zdroj: autor)

4.7.2.11 Epic

Epic je nejvyšší možný blok funkcionalit, který je rozložení do Super user story a dále. Epic by v našem případě mohl vypadat takto (viz Obr. 8): *„Jako správce chci mít systém pro správu přístupů do budovy, abych mohl pracovníkům přidělit ID karty a pomocí nich je vpouštět do budovy.“*



Obrázek 8: Ukázka Epic (Zdroj: autor)

4.7.2.12 Akceptační kritéria

Pro to, aby bylo možné danou User story považovat za dokončenou, je třeba splnit tzv. akceptační kritéria. Jedná se o seznam podmínek, které je nutné splnit, aby mohla být daná User story akceptovaná jako dokončená. Příklad akceptačních kritérií je uvedený na obrázku (viz Obr 5.).

Dle (Zuzana Štochová a Eduard Kunce, 2014) není nutné akceptační kritéria zaznamenávat za každou cenu. Existují projekty, kde je vztah týmu a Product owner na dobré úrovni a je dosaženo dobré úrovně komunikace. Pokud jsou však často User story zamítnuty je dobré si akceptační kritéria sepisovat.

4.7.2.13 Škálovatelnost

V rámci větších projektů je důležité brát ohledy na škálovatelnost metodiky. Pro metodiku SCRUM existují škálovací frameworky jako například LeSS (Large Scale Scrum) nebo SAFe (Scaled Agile Framework) či jednoduše SCRUM of SCRUMS. Pro projekt zmíněný v této práci však škálovatelnost není důležitým kritériem, jelikož vize růstu projektu není. Jelikož se jedná o agilní metodiky, mělo by se reagovat až na případnou změnu vize.

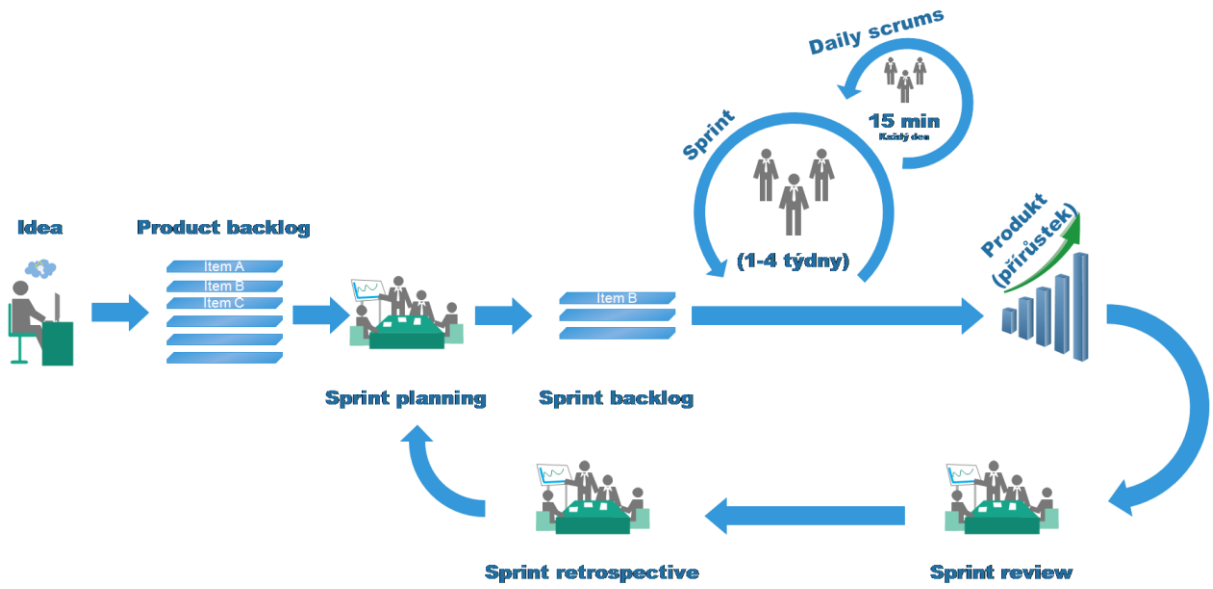
4.7.3 Charakteristika

SCRUM je jako všechny agilní metodiky rozdělený do iterací. Tyto iterace se nazývají *Sprint.* Sprint by zpravidla měl trvat 1-4 týdny. Za kratší dobu by toho tým nezvládl moc vyvinout a delší časové období by se stávalo méně flexibilní. Výsledkem každého sprintu by měl být vlastně takový mini projekt, na jehož konci by měl být dodán funkční celek, který by šlo případně nasadit do provozu.

Sprint musí mít předem určenou délku, nelze tedy prodlužovat, či zkracovat. Jedinou možnou časovou změnou je úplné zrušení sprintu. Zrušení sprintu může provést pouze Product owner. Důvodem pro jeho zrušení může být zastaralost cíle sprintu. Tento stav může nastat při změnách strategie, situace na trhu nebo změně technologických podmínek. Dále nelze provést změny, které by ohrožovaly cíl sprintu.

Sprint je složen ze *Sprint Planning Meeting, Daily Scrums*, vlastního vývoje, *Sprint Review* a *Sprint Retrospective*.

Posloupnost kroků ve sprintu ukazuje následující schéma (viz Obr. 9).



Obrázek 9: Schéma metodiky SCRUM (Zdroj: autor)

Je třeba zmínit, že mimo toto schéma se nacházejí podle (Kadlec 2004, s. 151) dva důležité kroky.

Prvním etapou vývoje software, než se začne se samotnými sprinty, je předehra. Ta má tři podúrovně – plánování, systémová architektura a vysokoúrovňový návrh. Zde se vlastně vytváří produktový backlog a přidávají se do něj jednotlivé položky.

Druhou etapou mimo toto schéma je dohra. Zde se jedná převážně o předávací procesy, uzavření software nebo samotné předání v požadovaném stavu a dodání potřebné dokumentace.

4.7.3.1 Sprint Planning Meeting

Jedná se o úvodní plánovací schůzku pro určení činností pro příští sprint. Plánování by se měl účastnit celý scrum tým a schůze by neměla trvat více než osm hodin. Průběh schůze určuje Scrum mater, jehož odpovědností je i věcnost a dodržení časového úseku schůze.

Plánovací schůze se věnuje dvěma základním otázkám:

1. Jaký přírůstek bude dodán na konci sprintu?

Cíl sprintu určuje Product Owner. Podle cíle vybírá jednotlivé položky z Product Backlog. Pro plánovací fázi jsou nezbytné informace o plánové kapacitě příštího sprintu a výkonu týmu ve sprintu minulém (i s ohledem na sprinty předchozí). Vývojový tým odhaduje náročnost jednotlivých položek. Často se pro odhadování náročnosti položek využívá tzv. Planning poker, který je popsán níže. Položky, které tým ohodnotil tak, že je zvládne dodat, musí korespondovat s cílem sprintu.

2. Jakou práci bude nutno pro dosažení cíle vykonat.

Po dokončení předchozí fáze tým rozebírá jednotlivé položky a rozděluje je na dílčí kroky, které je nutné provést pro vytvoření dané položky. Je velmi obtížné předem přesně určit, jaká položka zabere kolik času. Tým se přesto snaží ohodnotit časově jednotlivé položky a především u položek (*items*), které bude zpracovávat nejdříve, se snaží rozepsat jednotlivé potřebné kroky (*tasks*) pro položku.

4.7.3.2 Daily Scrums

Daily scrums, někdy označované jako standup, jsou jedním z hlavním principů metodiky SCRUM. Jedná se o každodenní meetingy, které by měli být realizovány v krátkém čase např. 15 minut. Měli by být konzistentní co se místa a času zahájení týče. Tyto meetingy moderuje Scrum master, jehož povinností je, aby členové mluvili jen ke kladeným otázkám a neprotahoval se tím čas meetingu. Jednou z best practices je požadavek na provádění Daily scrums ve stoje. To by mělo způsobit tlak na čas, jelikož většině lidí není pohodlné stát dělší časový úsek.

Při Daily scrums odpovídá každý účastník na tři otázky:

1. Co udělal, nebo dokončil od minulé porady, tedy za včerejší den.
2. Co bude dělat dnes, čili které úkoly ho čekají.
3. Zhodnotit jestli před ním stojí nějaké překážky nebo omezení.

Po Daily scrumu se mohou jednotliví členové týmu sejít a řešit další věci mezi sebou. Díky tomu, že jim toto neumožňuje přímo Daily scrum, nedochází k plýtvání času ostatních členů týmu.

Daily scrums jsou přínosné pro tým i tím, že každý člen má přehled o nejaktuálnějším stavu položek ve sprintu a díky třetí otázce se nemůže stát, že se jednotlivec zadrhne na nějakém problému. Toto přináší daleko rychlejší reakce a zamezí prodlevám ve vývoji. Pokud má například nějaký vývojář problém s určitým postupem a sdělí tento problém na meetingu, může mu jiný vývojář pomoci tento problém vyřešit a odstraní se tím riziko zpoždění dodání.

4.7.3.3 Sprint Review

Jak je z názvu patrné jedná se o určité vyhodnocení právě dokončeného sprintu. Této schůzky se mohou zúčastnit všichni zainteresovaní členové, tedy vývojářský tým, Product owner a stakeholders. Při délce trvání sprintu jeden měsíc by Sprint review neměl zabrat více než čtyři hodiny. Pokud je sprint kratší, Sprint review se úměrně tomu krátí

Při tomto meetingu se rozebírá, které položky se povedlo dokončit a které naopak ne. U těch, které se podařilo dokončit, následuje i praktická ukázka výsledku. Dále se hodnotí, co se během sprintu dařilo a co naopak stálo v cestě úspěšnému dokončení a jak se problémy řešili.

Product owner se zaobírá také product backlogem. Dle výsledků sprintu upravuje priority položek, reaguje na změny, které ovlivňují položky v product backlogu. Všichni účastníci meetingu společně navrhují, o kterých položkách z product backlogu by se mělo uvažovat při následujícím plánování.

4.7.3.4 Sprint Retrospective

Sprint retrospective (retrospektiva) je meeting, kterého se účastní scrum tým. Tato schůzka se koná po Sprint review a zároveň před Sprint planning meeting. Při délce sprintu jeden měsíc by měla tato schůze trvat zhruba tři hodiny. Při retrospektivě se jednotliví členové týmu vyjadřují k tomu, s čím byli spokojení, co by chtěli zlepšit, co bylo problematické a je tedy třeba to zrušit, a závěrem také co by chtěli zavést. Jedná se především o procesy, techniky, praktiky a zvyklosti.

Dle (Zuzana Štochová a Eduard Kunce, 2014) je obvyklé, aby retrospektiva měla tyto fáze:

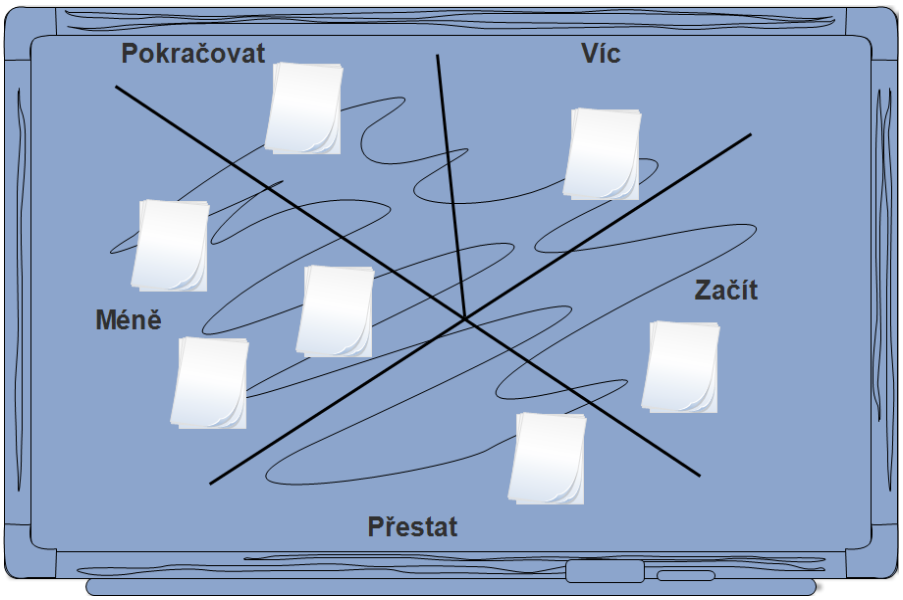
- Úvod
  - Jde především o seznámení se s aktuálním programem a připomenutí si minulé retrospektivy.
- Sběr dat
  - V této fázi se sbírají data o aktuálním stavu: co funguje, co ne, případně co by šlo zlepšit.
- Hlubší porozumění informacím
  - Zde se hledají možnosti jak zlepšit danou věc a identifikují se problémy, které danou věc brzdí.
- Brainstorming možností
  - Při brainstormingu se všichni snaží přicházet s možnými vylepšeními a návrhy.

- Shrnutí konkrétních akcí
    - Zde se provádí shrnutí a konkretizace akcí a změn, které se budou aplikovat na příští sprint.

Mezi best practices retrospektivy patří využívání různých druhů technik jako je dle (Zuzana Štochová a Eduard Kunce, 2014) například Retrospektiva s lepítky.

Kdy se vytvoří hvězdice, jako je na obrázku (viz Obr. 10). Názvy jednotlivých kolonek Začít, Přestat, Méně, Pokračovat a Víc vypovídají samy za sebe. Každý člen týmu nalepí lísteček s názvem praktiky do kolonky podle svého uvážení. Poté moderátor – Scrum master prochází jednotlivé části hvězdice a jednotlivé lístečky uvnitř.

Vylepšením této techniky může být hlasování, kdy každý člen týmu má definovaný počet hlasů a těmi podle svých preferencí hodnotí lístečky. Lístečky s nejvíce body mají největší prioritu k diskuzi.



Obrázek 10: Ukázka best practices při Retrospektivě (Zdroj: autor, dle [Zuzana Štochové a Eduard Kunce, 2014, str. 73])

Existuje velké množství technik, kterými lze vylepšit retrospektivu. Mezi ně patří například časová osa sprintu, Fishbone, loď či ESVP (více viz (Zuzana Štochová a Eduard Kunce, 2014)).

4.7.4 Další praktiky metodiky SCRUM

Mezi jedny z dalších ve SCRUMU využívaných praktik patří například Planning poker pro ohodnocení položek story pointy, či Scrum board.

4.7.4.1 Planning poker, story point

Jedná se o techniku využívanou pro odhadování náročnosti práce a účastní se jí celý tým. Využívá se zde několika typů ohodnocení, obecně tomu říkejme story pointy. Nejprve Product owner představí danou funkcionalitu, tak aby jí všichni členové týmu rozuměli. Poté si každý člen vybere kartu s bodovým ohodnocením, které vystihuje náročnost vývoje dané funkcionality. Scrum master poté nechá všechny karty najednou otočit. Následně se člen či členové týmu s nejvyšším bodovým ohodnocením snažím vysvětlit zbytku týmu,

proč si myslí, že bude danou funkcionalitu tak složité vyvinout. Totéž udělá i člen či členové s nejnižší bodovým ohodnocením. Je ovšem třeba dbát na to, aby ohodnocení bylo věcné a zakládalo se na faktech, jako např.: „Vyvíjel jsem podobný modul, tudíž mám již určitou znalost této technologie a nebude tolik složité danou funkcionalitu vyvinout.“ nebo „Daná funkcionalita sice nebude složitá na vývoj, ale je potřeba pozměnit architekturu a sepsat mnoho nových testů, odhaduji tedy vyšší náročnost“. Nemělo by dojít ke komentářům typu: „Přijde mi to prostě složitější.“ Tento princip pomůže v případech, kdy má někdo větší znalost, nebo naopak menší znalost ohledně náročnosti vývoje dané funkcionality. Po této fázi členové týmu hodnotí náročnost funkcionality znovu. Pokud se všichni shodnou na náročnosti pokračuje se na další položku. Nejde tedy o kompromis nebo rozhodnutí jednoho člena týmu, ale o konzistentní postoj celého týmu.

Pro ohodnocení náročnosti se používá několik typů ohodnocení (seřazeno podle oblíbenosti)

1. Story points: 0,1/2,1,2,3,5,8,13,20,40,100, ∞,?,kafe.
2. Velikost oblečení: xs,s,m,l,xl,xxl, ∞,?,kafe.
3. Fibonacciho posloupnost: 0,1,2,3,5,8,13,21,34,55,89,144, ∞,?,kafe.

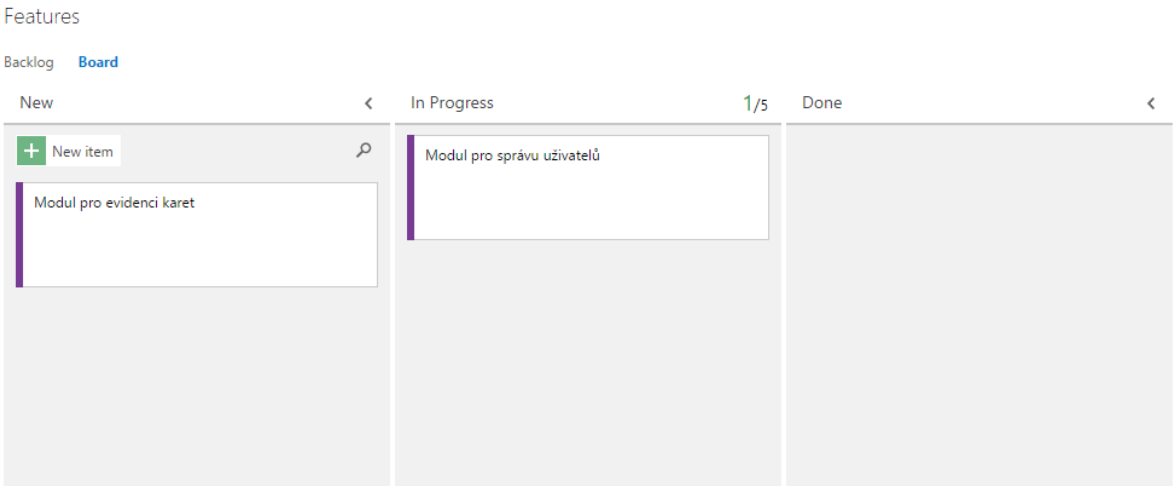
Je důležité brát story pointy jako jednotku složitosti a ne jednotku času. Aby si pod tímto ohodnocením týmy nepředstavovaly časovou náročnost, některým pomáhá spíše druhá zmíněná varianta. Nikde není specifikováno, kolik práce by dané počet story pointů měl vyjadřovat. Vždy záleží na týmu, jakou si určí škálu. Některý tým může ohodnotit funkcionalitu velikostí S a druhý tým může tu stejnou funkcionalitu ohodnotit velikostí L, přesto oba týmy mohou tuto metodu aplikovat správně.

Vysoká čísla jako 40 či 100 představují Epic funkcionality a bylo by moudré je rozdělit na více funkcionalit, které nebudou tak náročné na vývoj. Symbol nekonečna představuje funkcionalitu, kterou z nějakého důvodu nelze vyvinout v daný moment. Může to být neznalostí technologie, neexistencí technologie nebo to může být způsobeno nedostatkem zdrojů, ať už lidských či technických. Otazník znamená, že nějaký člen týmu nemá zdání, o co se jedná a je třeba mu funkcionalitu a úkony, které je třeba pro její vývoj udělat, lépe vysvětlit. Symbol kafe znamená, že člen týmu je z plánování unavený, a bylo by vhodné dát si pauza. Tato situace většinou nenastane u zkušených týmů, které již několik společných sprintů absolvovaly. Zpravidla k tomuto dochází, pokud se s vývojem produktu začíná a hodnotí se velká část nebo celý product backlog.

Mnoho týmů používá story pointy na ohodnocení jak product backlogu, tedy středně dobých a dlouhodobých plánů, tak i sprintů. Sám zakladatel této metody Mike Cohn (Cohn, 2014) však nedoporučuje plánování sprintů ve storypointech. Udává, že pokud průměrné velocity je 20 story pointů za sprint a zbývá nám šest sprintů, je možné říci, že za těchto šest sprintů jsme schopni udělat zhruba 120 story pointů. Problémové se podle něj stává tvrzení, že pokud tedy máme průměrnou velocitu na 20 storypointů, tak bychom následující sprint měli splnit též 20 storypointů. Pro takto krátkodobé plánování (tedy například pro příští sprint), doporučuje využívat pouze časové odhady.

4.7.4.2 Scrum board

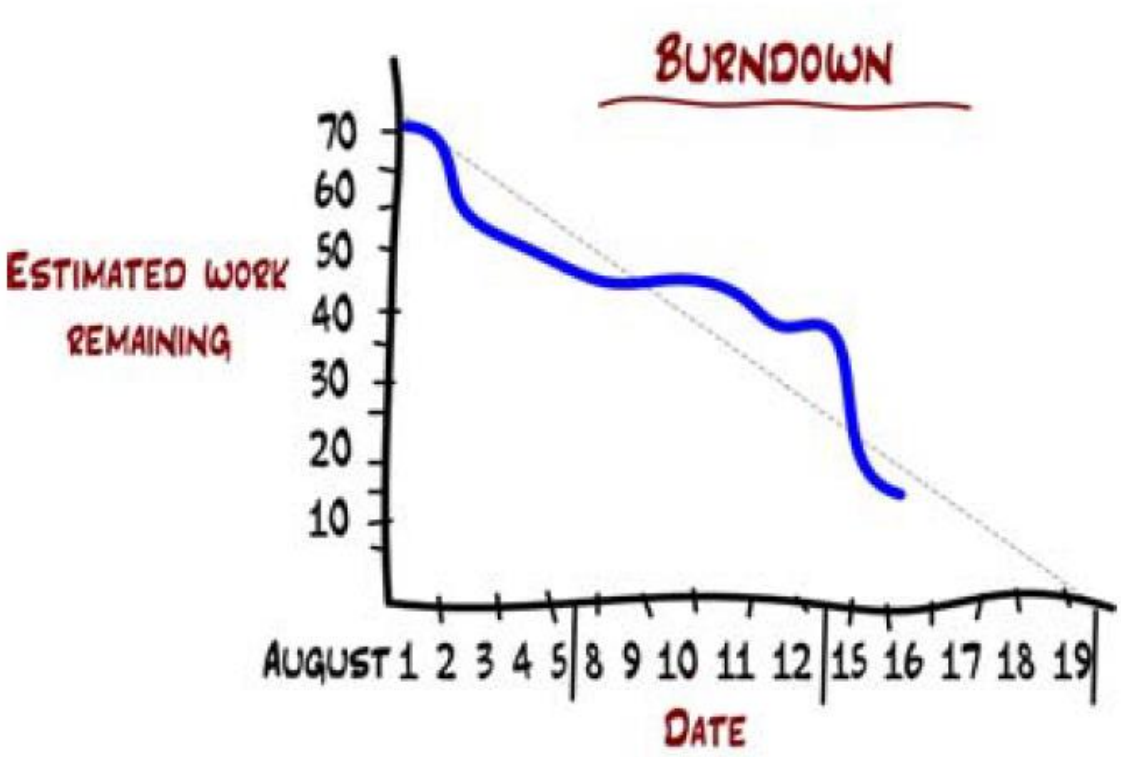
Scrum board je rozdílně zobrazení jednotlivých položek v Product backlogu, či ve Sprintu. Jedná se o tabuli, kde jsou jednotlivé položky rozděleny do sloupců podle aktuálního stavu, ve kterém se nacházejí. Základní rozložení Scrum boardu je vidět na obrázku (viz Obr, 11), kde jsou tři kategorie: Nové položky (New, To-do, Backlog), Rozpracované položky (In Progress) a Hotové položky (Done). Tyto kategorie se dají různě upravovat pro potřeby týmu, například přidáním sloupce pro testování apod.



Obrázek 11: Ukázka Scrum board (Zdroj: autor)

4.7.4.3 Burndown chart

Burndown chart je graf, který slouží pro sledování aktuální výkonosti týmu v daném sprintu. Jako ukázku zde je burndown chart (viz. Obr 12) z knihy Kanban and Scrum: making the most of both (Kniberg a Skarin, 2010). Na osy X je uveden čas a na ose Y jsou uvedeny nejčastěji odhadované hodiny zbývající práce. Jedná se o hodiny, které se odhadují vždy při Sprint planning meeting. Tento graf pomáhá jako další technika odhalit možné problémy. Ideální by bylo, pokud by aktuální výkonnost týmu (modrá čára) kopírovala předpokládanou výkonnost týmu (čerchovaná čára). Čím více se hodnotám tým blíží, tím více je zkušený a dokáže lépe odhadovat náročnost práce. Burndown chart slouží především pro Product ownera, který má díky němu přehled kolik funkcionality se stihne ještě udělat. Burndown chart by neměl sloužit jako metrika pro manažery, kteří by poté tlačili na tým.



Obrázek 12: Burndown chart (Zdroj: Kniberg a Skarin, 2010, str. 38)

4.7.5 Výhody a nevýhody SCRUM

Mezi hlavní výhody při používání metodiky SCRUM patří možnost rychle reagovat na změny a rizika. Tuto vlastnost má hlavně díky Daily scrums. Předchází se tím narůstajícím problémům a dochází k zefektivňování procesu vývoje.

Ve SCRUMu mají též vývojáři velkou svobodu ve volbě nástrojů a postupů při vyvíjení požadovaných položek. Tato svoboda může být někdy omezena u vnitrofiremních týmů, kde se mohou používat pouze určité nástroje a postupy.

Výhoda a zároveň nevýhoda této metodiky může spočívat v tom, že se jedná v podstatě o rámec, kde nejsou přesně definované kroky. SCRUM lze tedy ohýbat podle potřeb. To

může některým firmám vyhovovat, ale pro některé (především nezkušené) může být absence jasně definovaných kroků problematická.

Hlavní nevýhody vyplívají z obecných nevýhod agilních metodik. To jsou zejména vyšší nároky na tým týkající se jejich přístupu k práci, motivaci a sebeřízení. Část lidí pro tento přístup může být nevhodná, jelikož potřebuje být pod klasickým vedením. Nevýhodou metodiky SCRUM může být též velikost týmu. Při počtu do deseti členů je složitější aplikovat tuto metodiku na velké projekty. Zde musí docházet ke koordinaci více týmu, což zvyšuje nároky jak na týmů tak na Scrum mastera.

4.8 Extrémní programování (XP)

Metodiku Extrémního programování (XP) představil expert na vývoj softwaru Kent Beck v roce 1999 ve své práci (Beck, 1999). Z této práce, pokud tomu není v textu uvedeno jinak, věcně vychází následující podkapitola.

4.8.1 Charakteristika XP

Dle (Kadlec 2004, s. 125) je základním filozofickým východiskem metodiky XP přesvědčení: *„že jediným exaktním, jednoznačným, změřitelným, ověřitelným a nezpochybnitelným zdrojem informací je zdrojový kód.“* Tato charakteristiky osvětluje základy metodiky XP a její jednotlivé části jsou rozvedeny v následujícím textu.

Dle zakladatele této metodiky (Beck, 1999) stojí XP na těchto základních principech:

- Rozlišovat mezi rozhodnutími byznysu a účastníky projektu (stakeholders).
- Vytváření testů (unit testy) ještě před psaním kódu a následné průběžné testování.
- Integrování a testování celého systému i několikrát denně.
- Využívání párového programování.
- Zahájení projektu s jednoduchým návrhem, který se bude průběžně navrhovat a vylepšovat.
- Dávat do produkce i systém s minimální funkčností, který se následně rozšiřuje a upravuje.

Pod prvním bodem si můžeme představit angažovanost týmu a zlepšení komunikace se zákazníkem, která je v XP (jak název metodiky napovídá) dovedena do extrému. To znamená, že navrhuje tzv. princip zákazníka na pracovišti.

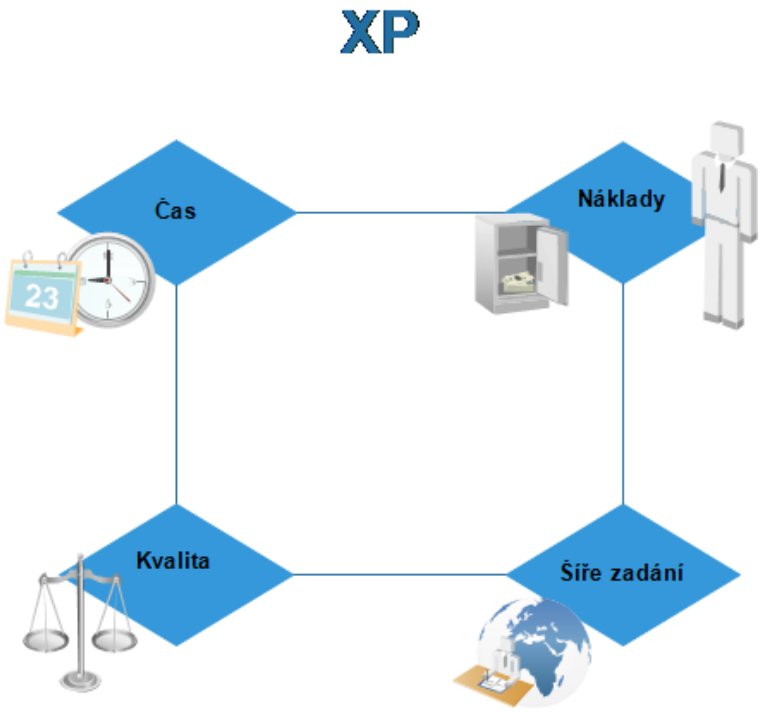
Druhý bod zavádí již zmíněné unit testy. Navíc je zde takzvané funkcionální testování, které provádí sám zákazník.

Třetí bod se věnuje nasazování přírůstků produktu a testování, které je v XP průběžné.

Pod čtvrtým bodem je myšlenka již zmíněného párového programování společně s dalšími principy, jako je společné vlastnictví kódu.

Pod pátým a šestým bodem si lze představit myšlenku co největší jednoduchosti, tedy dodávání co nejjednodušší verze produktu, který může fungovat a iterační přístup, kterým jsou agilní metodiky charakteristické.

Kent Beck ve své práci zmiňuje čtyři proměnné (viz Obr. 13): náklady, čas, kvalitu a šíři. Dle (Kadlec, 2004) je to modifikace již zmíněných tří proměnných (viz Obr. 3), kde kvalita odpovídá funkcionalitě, čas zůstává neměnný, náklady odpovídají zdrojům a šíře zadání je nová proměnná.



Obrázek 13: Čtyři proměnné v metodice XP (Zdroj: autor)

Důležitým faktorem je v tom pohledu na proměnné, možnost si vybrat tři proměnné a ty ovlivňovat. Výsledkem pak bude čtvrtá proměnná, kdy si zákazník či manažer řídí vstupy na těchto třech vybraných proměnných a dostává výstup čtvrté.

Podle Becka však nelze tyto proměnné jednoduše a neomezeně ovlivňovat. To uvádí na příkladu, že nejde vždy získat software rychle při zvýšení nákladů (tj. přidáním peněz), kdy to přirovnává ke rčení „Devět žen společně nemůže odnosit dítě za jeden měsíc.“ Je tedy třeba do vybraných proměnných vkládat relevantní vstupy a požadovat od nich přiměřený výstup.

Pro XP jsou nejdůležitější tyto čtyři hodnoty:

- komunikace
- jednoduchost
- zpětná vazba
- odvaha

Prvním bodem je komunikace, zde bývá často jádro problémů či zpoždění. Problémy v projektech vznikají například tím, že někdo udělá chybu a stydí, se nebojí ji přiznat. Pokud by však danou chybu oznámil z počátku, bylo by její odstranění značně jednodušší, než když chyba tzv. probublává. Příčinou tohoto velmi často bývá špatný přístup manažerů, kteří místo ocenění, že jim někdo chybu dostatečně brzy nahlásil či přiznal, daného člověka

potrestají. Této problematice se ve své práci (Kadlec, 2004) věnuje Václav Kadlec, který zde rozebírá využitelnost této metodiky v našich zeměpisných šířkách, jelikož je česká mentalita známá tím, že se každý bojí udělat chybu místo toho, aby se pouštěl do odvážných věcí, které jsou pro tuto metodiku třeba. Druhým potenciálním kritickým místem může být komunikace se zákazníkem, kde může být chyba obousměrná. Zákazník může například nepřesně definovat svůj požadavek, programátor či manažer si tento požadavek neověří a aplikují ho tak, jak byl pochopen, tedy špatně a chyba je na obou stranách. Důležité je tedy pokládat správné otázky ohledně projektu a dostatečně potvrzovat pochopení požadavků.

Druhým bodem je jednoduchost. V XP se manažer či kouč ptá týmu: „Jaká je nejjednodušší věc, která může fungovat?“ Základní myšlenkou je tedy rozkládat funkcionalitu do co nejmenších funkčních částí a to pouze těch částí, které jsou momentálně požadovány. Příkladem může být projekt, který je součástí této práce. V klasickém pojetí by se aplikaci pro personální oddělení navrhovala a už v návrhu a analyzování by byla snaha obsáhnout co největší možnou funkcionalitu. V porovnání XP a klasického přístupu by se pak k této aplikaci přistupovalo například takto. Požadavek zní: „Je třeba vyvinout aplikaci pro správu uchazečů o zaměstnání.“ V klasickém pojetí vývoje se uvažuje o tom, co vše je třeba mít ve správě uchazečů, na jaké systémy by v budoucnu aplikace mohla být navázána a co vše má pokrýt. V XP se nejprve naprogramuje část aplikace, kde bude pouze seznam kandidátů, kam půjde kandidáty přidávat. Dále se naprogramuje část, kde budou detailnější informace o kandidátech. Jedna z přidávaných částí by se mohla věnovat přidávání a evidenci pracovních pohovorů. Až po roce vývoje by mohl například přijít požadavek na propojení se systémem správy zaměstnanců. Toto vše se v metodice řeší, až když existuje požadavek na funkcionalitu, ne dříve.

Třetím bodem je zpětná vazba. Tento bod je úzce spjat s prvním bodem, tedy komunikací. Což znamená, že čím více a lépe komunikujeme, tím více můžeme získávat zpětné vazby. Dále je úzce spjat s testováním. Tím, že programátoři píší unit testy, se jim dostává okamžité odezvy, zda metoda pracuje správně. Dlouhodobější zpětné vazby se dosahuje pomocí testů funkcionality, které píší zákazníci, či testeři. Tyto testy ověřují nejen implementovanou funkcionalitu aplikace ale i to, zda aplikace dělá to, co po ní zákazníci požadují.

Posledním bodem je odvaha. Zde jak (Kadlec, 2004) tak (Beck, 1999) představují odvahu na příkladu práce s kódem. Uvádějí příklady, kdy tým zjistí zásadní chybu v architektuře, kdy je už naprogramováno třeba 70 % projektu, přesto tým musí sebrat odvahu a stávající kód přepsat či dokonce zahodit.

Průběh projektu je řízen pomocí tzv. Plánovacích her.

Plánovací hry slouží k rozhodování o průběhu projektu a náplni jednotlivých iterací. Plánovacích her se musí podle (Beck, 1999) účastnit všichni účastníci projektu, tedy jak obchodníci, tak vývojáři a manažeři. Na počátku projektu je vhodné vytvořit hrubý plán projektu, který se v průběhu aktualizuje a mění. Toto plánování by nemělo zabrat více než několik hodin.

Po tom co je hrubý plán vytvořen se již vytvářejí tzv. plány verze**. Plán verze** se sestavuje na základě analýzy požadavků zákazníka. Poté co jsou potřeby a požadavky zákazníka zanalyzovány, se využívá **plánovací hry**. Plánovací hra zde slouží k rozdělení požadavků zákazníka do tzv. **karet zadání**, které zobrazují jednotlivé funkcionality a jejich podrobný rozbor.

Plánovací hra se podle (Beck, 1999) skládá z těchto fází:

1. Průzkum (Exploration phase)

   Nejprve vedení sepíše jednotlivá zadání ve formě User story. Poté vývojáři odhadují náročnost a dobu implementace User story. Pokud se týmu nedaří odhadnout celou User story, je vhodné, aby je vedení rozdělilo na více User story.

2. Závazek (Commitment phase)

   Výsledkem této fáze je, že vedení vybere šíři zadání a data pro následující release. Tým musí odsouhlasit, že je schopný požadavek v daném čase splnit. V této fázi se nejprve User story rozdělí do tří sloupců podle přidané hodnoty. Poté se User story dělí podle proveditelnosti. Poté vývojáři sdělí vedení, jaká bude ideální rychlost vývoje (velocity). Po těchto krocích vedoucích k získání potřebných odhadů vedení rozhoduje o tom, které User story se budou realizovat.

3. Řízení (Steering phase)

   Tato slouží k řízení vývoje a případným úpravám plánu. Jelikož se jedná o agilní metodiku, je možné i během této fáze přidávat, pokud se objeví, důležité User story, či případně měnit odhady podle aktuálního stavu. Tato fáze je složena z jednotlivých iteračních plánovacích her.

Iterační plánovací hra se skládá z podobných kroků jako plánovací hra, oproti ní se však iterační plánovací hra zaměřuje na konkrétní funkcionality a její výsledek je zaznamenán na úkolové karty. **Úkolové karty** detailně popisují jednotlivé funkcionality a na jejich plánování se podílí všichni členové týmu.

Metodika XP určuje několik fází projektu:

1. průzkum
2. plánování
3. iterace až do uvolnění prvního release
4. zprovozňování
5. údržba
6. smrt

Více o těchto fázích v (Kadlec 2004) nebo (Beck 1999).

Podle (Beck 1999) jsou v XP definovány tyto role:

- Programátor

  Dalo by se říci, že se jedná o základní stavební kámen metodiky XP. Úlohou programátora není pouze psát kód, ale důležitá je i komunikace, psaní testů (unit testy). Všeobecně jsou na programátora kladeny vyšší nároky, než u klasicky specializovaných programátorů.

- Zákazník

  Pro metodiky XP další nepostradatelná role. Je důležité, aby byl zákazník na pracovišti neustále, sžil se s metodikou, dokázal tvořit testy funckionality a upřesňoval požadavky. Důležité je, aby zákazník přijmul spoluzodpovědnost za projekt.

- Tester

  Jelikož samotné testování provádí již programátor, tkví role testera spíše v asistenci zákazníkovi při psaní testů funkcionalit.

- Stopař

  Jeho hlavní činností je sledování průběhu vývoje, kdy provádí časové odhady, sbírá informace a hledá možné příčiny zpoždění projektu.

- Kouč

  Jedná se o obdobnou roli jako je v metodice SCRUM Scrum master. Jeho úkolem je kontrolovat správnost postupů dle metodiky, hledat možné problémy a alternativní řešení a především dbát na fungující komunikaci mezi jednotlivými členy týmu.

- Konzultant

  Jelikož v metodice XP by měli být programátoři co nejvšestrannější, je tato role potřebná v určitých oblastech pro technickou konzultaci.

- Velký šéf

  V organizaci se jedná o klasického manažera. Je důležité, aby přijmul metodiku XP v celém rozsahu, což může být někdy, vzhledem k principům a praktikám této metodiky komplikace. Charakteristické rysy této role jsou odvaha, trpělivost a důvěra.

Pro webové projekty jsou v (Wallace et al. 2003) definovány tyto role:

- Stratég – obdoba kouče
- Programátor rozhraní – má na starosti technologie jako CSS, Javascript, HTML
- Grafický návrhář
- Programátor serverové strany – role obdobná roli výše zmíněného programátora
- Rádce - obdoba stopaře a konzultanta dohromady
- Tester – rozšířená o testování uživatelského rozhraní (Selenium), grafického návrhu, apod.

4.8.2 Praktiky XP

Další částí XP jako celku jsou využívané praktiky. Celkem je těchto praktik 12 a to:

- Plánovací hra
- Testování
- Malé verze
- Metafora
  - Jedná se o jednoduchý příběh, který popisuje, jak má systém fungovat.
- Jednoduchý návrh
- Kolektivní vlastnictví kódu
- Refaktorizace
- Průběžná integrace (CI)
- Párové programování
- Čtyřicetihodinový pracovní týden
- Zákazník na pracovišti
- Standardy psaní zdrojového kódu

4.8.2.1 Testování

Pod tímto prvkem se skrývají psaní testů a testovaní samotné a to především průběžné. Pro webovou aplikaci je nutné zmínit dvě formy testů a to unit testy a automatizované testování UI. Samozřejmě lze využít více typů testování, ale ty jsou mimo rozsah a zaměření této práce.

Unit testy (jednotkové testy) se využívají pro testování co nejmenších částí kódu. Jednotlivé testy by měli být co nejméně závislé na sobě. Z počátku může být psaní unit testů zpomalující a to obzvlášť zpětně, pro již existující a běžící projekt. Avšak i když je snazší vytvářet testy pro nové projekty, je vhodné se psaní těchto testů věnovat. V celkovém důsledku se unit testy vyplatí, protože pokud jsou psány dobře, dokáží odhalit spoustu chyb. Výhodné je to obzvlášť v agilním přístupu, kdy je třeba testovat průběžně. Další výhodou je situace kdy mohou všichni členové týmů upravovat jakoukoliv část kódu. V tomto případě unit testy odhalí změny funkčnosti jiné části kódu, je tím tedy chráněna funkčnost kódu jako celku. Nejčastěji se testuje ve smyslu při tomto vstupu, očekávám tento výstup. Dále pak opačně, kdy se schválně zadá špatný vstup a kontroluje se, jestli daná část kódu opravdu ukáže chybu a nebude fungovat. Existuje, však obrovské množství možností na co lze jednotkové testy vytvářet, jako například testy na extrémní hodnoty

apod. Pro psaní unit testů se využívají frameworky jako například.: NUnit či MSUnit (.net), JUnit (Java), apod.

Hojně se využívá i **automatického testování UI**. Jedná se o testy, které simulují aktivitu uživatele. Těmito testy se prochází nejběžnější scénáře testů, jelikož tyto testy dokáží testovat danou funkcionalitu mnohonásobně rychleji než člověk a lze jde spouštět kdykoliv. Z velké části tedy odstraňují nutnost využívání testerů-uživatelů. Neodstraňují tuto nutnost však zcela, jelikož jsou různé faktory, které otestovat nedokáží. Více o tomto testování se lze dozvědět v odborných knihách či práci Ing. Lucie Třískové (Třísková 2015). Stejně jako u jednotkových je zpočátku investice (časová i finanční) vysoká, ale postupem času se vyplácí více a více.

4.8.2.2 Kolektivní vlastnictví kódu

Zatímco v metodice SCRUM dle (Kadlec 2004) je využíváno individuálního vlastnictví kódu, v metodice XP je využíváno kolektivního vlastnictví kódu.

Filozofie metodiky XP tkví v tom, že by všichni členové týmu měli mít zodpovědnost za celý systém dohromady. Oproti tomu stojí filozofie metodiky SCRUM, kde je individuální vlastnictví využíváno k motivaci jednotlivce mít co nejvyšší kvalitu.

4.8.2.3 Refaktorizace

Pro pochopení refaktorizace je vhodná definice toho pojmu od Marka Pilgrima (Pilgrim 2010): *„Refaktorizace je činností, kdy vezmete fungující kód a uděláte z něj ještě lepší. „Lepší“ obvykle znamená „rychlejší“, ale může to taky znamenat „používající méně paměti“ nebo „používající menší diskový prostor“ nebo je prostě „elegantnější“. Refaktorizace je z hlediska dlouhodobého zdraví každého pro-gramu důležitá, ať už to znamená cokoliv pro vás, pro váš projekt nebo pro vaše okolí.“*

Podle Kadlece (Kadlec 2004) lze tuto definici doplnit o ne zcela zřejmou věc a to je, že se při refaktorizaci nemění zákaznická funkcionalita kódu, ale pouze struktura. Tento prvek tedy souvisí s prvkem testování (viz kapitola 4.8.2.1), jelikož pokud se provádí refaktorizace, nemělo by dojít k nutnosti změny unit testů. Test by měl vykazovat stejný výsledek. Existují však případy kdy je změna testů odůvodněná, podle (Kadlec 2004) například u změny rozhraní objektu.

Dále není doporučováno refaktorizovat nárazově, ale refaktorizovat kontinuálně. Tedy jde, o to aby tým aktivně vyhledával a měl zájem vidět prostor pro refaktorizaci.

Podrobněji se refaktorizaci věnuje ve svých knihách a to především v (Fowler a Beck 1999), Martin Fowler.

Existují však zdroje, které refaktorizace za přínosnou příliš nepovažují.

Jedním z těchto zdrojů je poměrně nedávno provedená studie (Kannangara a Wijayanake 2015), kde ve většině udávaných změn dopadl refaktorizovaný kód dokonce hůř než původní kód. Jediné zlepšení a to o nepříliš významná 4 %, bylo uvedeno u údržby kódu.

Ve studii (Alshayeb 2011) dopadla refaktorizace taktéž negativně. V této studii dokonce nebylo zaznamenáno žádné celkové zlepšení (Pouze zlepšení části kódu, avšak u větší části stejného kódu bylo zaznamenáno zhoršení.)

4.8.2.4 Continuous integration (CI – Průběžná integrace)

Tento pojem ovšem více než samotný Kent Beck (Beck 1999) proslavil článkem Martin Fowler v roce 2000. V nejaktuálnější verzi tohoto článku z roku 2006 ji popisuje (Martin Fowler 2006) jako metodu vývoje, kde členové týmu integrují jejich práci často, nejlépe denně či několikrát denně. Každá z těchto integrací nového kódu je vytvořena automatických buildem - CI, obsahujícím Unit testy (viz výše), který se snaží detekovat chyby v co nejrychlejším čase.

Automatické buildy jsou součástí tohoto přístupu a výrazně urychlují práci oproti ručně sestavovaným buildům.

Pro softwarové projekty je doporučené a v dnešní době se hojně využívá tzv. Repository čili úložiště zdrojových kódů. Využívání repository zajišťuje konzistenci souborů, zálohování práce, verzování, historii souborů, spojování kódu a informaci o tom kdy a kdo provedl jaké změny. Dalších výhodou je nenutnost mít u sebe celý zdrojový kód, což by způsobovalo vyšší nároky na stanici, na které vývojář pracuje, ale stačí si z repository stáhnou pouze část kódu, s touto částí pracovat a poté jí znovu nahrát.

Mezi možné komplikace při aplikování CI může patřit větší náročnost na přenos dat a s tím související závislost na centralizovaném sytému. Většina vývoje však stejně probíhá za pomocí online nástrojů, komunikace a obecně připojení k síti, takže pokud by neměla fungovat síť, nebyla by závislost na centralizovaném systému jediným problémem.

Mezi hlavní důvody proč využívat CI patří vysoká chybovost v kódu, složitost vytváření nových buildů, neorganizovanost ve verzování a s tím související nepořádek v úložišti zdrojových kódů. Hlavním důvodem na pozadí ovšem podle (Krill 2011) zůstává ekonomická výhodnost, jelikož je levnější platit provoz počítačů, než lidí.

Mezi hlavní zástupce CI patří například Jenkins, Team Foundation Server (Visual Studio) či Cruise Control.

Přístup nabourávající klasický CI je tzv. branch a merge. Tato technika je velmi využívána v populárních Repository jako je Git či Mercurial. Jde o to, že z hlavního trunku se vytvoří branch, na něm vývojář pracuje a až je s výsledkem spokojený, provede merge a rozšíří hlavní trunk. Kombinace těchto dvou přístupů je v současné době asi nejvíce využívána, jelikož jeden ze zkladních principů agilního manifestu (K. Beck, M. Beedle, A. van Bennekum, et al. 2001) je „Working software is the primary measure of progress.” Což v některých případech, při využívání CI, může vést ke komplikacím. Například pokud je část kódu, na které vývojář pracuje, samostatně nespustitelná, je lepší udělat branch a až bude kód ve spustitelném a ověřitelném stavu, může se provést branch a CI.

4.8.2.5 Pair programming (párové programování)

Jedná se o techniku, kterou uvedl ve své práci (Beck 1999) Kent Beck, kdy u jednoho počítače sedí dva programátoři a pouze jeden z nich píše v daný moment kód. Druhý

programátor má za úkol dohlížet na správnost architektury a kódu a pomáhat ho vylepšit. Existuje více druhů přístupu k párovému programování.

Tato dvojice se pravidelně střídá. Střídání může probíhat buď po dokončení určitého úkolu, nebo po časovém úseku. Důležité, je podle něj, aby programátoři byli na obdobné úrovni. Pokud by spolu programovali senior a junior programátor, nejednalo by se o párové programování, ale spíše o vztah učitel-žák. Podle (McConnell 2005) je zase dbát na standardizaci kódování, aby se programátoři nad stylem nehádali.

Párové programování slouží především pro větší bezchybnost kódu (například pro aplikaci, které vyžadují naprostou bezchybnost – přístroje v medicíně, apod.) nebo podle (Zuzana Štochová a Eduard Kunce 2014) pro zlepšení komunikace a vztahů v týmu a v neposlední řadě může sloužit pro rychlejší seznámení s projektem pro nové členy týmu. Existují i studie jako například (Cockburn a Williams 2001), které prokazují, že párové programování ve výsledku nemusí dražší o 100 %, jak by se mnoho lidí mohlo domnívat, ale pouze o 15%. Tato studie přichází i s dalšími zajímavými zjištěními a to, že při párovém programování vzniká menší počet řádků kódu. V této studii je i zachyceno zlepšení vztahů na pracovišti a větší spokojenost s prací. Tyto výhody párového programování dokazuje i jiná studie (Williams et al. 2000)

V neposlední řade může podle (Wray 2010) párové programování sloužit pro ověření znalostí a produktivity určitého programátora. Efektivnost programátorské práce je velmi těžce měřitelná, nedá se určovat kvalita programátora podle počtu napsaných řádků kódu. Pokud však máme ve firmě programátora, jehož kvalitami jsme si jistí, nebo dokonce tímto programátorem jsme my samy, můžeme ve zkušební době daného kandidáta na pozici nechat pracovat společně s naším ověřeným. Toto je možné aplikovat i na jedince, kteří již ve firmě působí, a máme o nich pochybnosti.

4.8.2.6 Žádné přesčasy

Tento prvek metodiky XP je především ochranou pro programátory a ostatní členy týmu před přetěžováním. Pokud má projekt časový skluz, tak velmi často dochází právě k přetěžování lidí. Nedodržování termínů se snažím předcházet všeobecně agilní metodiky iteračním inkrementálním přístupem. Pokud je produkt dodáván průběžně a dochází k dobré komunikaci mezi zákazníkem a vývojářským týmem mělo by se nedodržování termínů předejít. I přesto se v XP zavedl tento prvek.

Jde tedy především o zakázání pracování přesčasově z přikázání, jelikož při neúměrné pracovní zátěži klesá produktivita a zvyšuje se chybovost. Nejedná se tedy o to, že by měla být fixní pracovní doba, kterou programátor nesmí překročit. Pracovní doba vývojářů by naopak neměla být fixní, nejedná se o profesi, kde je nutná přítomnost pracovníka v přesně daný čas. Při programování je nutné se na práci velmi soustředit, a pokud se pracovník dostane do stavu flow (více viz (Csikszentmihalyi 1991)), byla by chyba mu bránit v prodloužení pracovní doby a tím ho z toho stavu vytrhnout.

4.8.2.7 Zákazník na pracovišti

V metodice SCRUM je role zákazníka definovaná rolí Product owner (viz kapitola 4.7.2.1), kde je důležité tuto roli obsadit, ten kdo jí obsadí, musí mít naprostou znalost produktu a

musí mít příslušnou pravomoc. Nezáleží tolik na tom, zda to bude člověk přímo od zákazníka, jeho zástupce, či někdo z firmy. Role Product Ownera je důležitá v zmíněných situacích v kapitole 4.7.3 a není vyžadována jeho neustálá přítomnost na pracovišti.

Oproti tomu XP přichází se zákazníkem na pracovišti, který zde funguje na plný úvazek. Je důležité, aby byl fyzický přítomný u vývojářského týmu a odpovídal na jejich dotazy, či upřesňoval požadavky.

4.8.2.8 Standardy psaní kódu

Standardy pro psaní kódu (Coding Standard) je označení pro stejný styl psaní zdrojového kódu. Tato praktika je dle (Zuzana Štochová a Eduard Kunce 2014) užitečná především při růstu projektu, kdy je kód přehledný i pro lidi, kteří ho nepsali. Jelikož jedním z dalších prvků agilních metodik je sdílený kód (viz kapitola 4.8.2.2) je tato standardizace kódu velmi užitečná.

Dalším principem agilních metodik dle (Kadlec 2004), se kterým standardizace kódu souvisí, je minimum dokumentace. Jelikož v agilních metodikách se dokumentace minimalizuje a hlavním nosným prvkem informací je zdrojových kód, je nutné tento kód mít přehledný. Podle obou zmíněných zdrojů, je aplikaci standardů z počátku nepříliš dobře vnímána vývojáři, avšak je nutné už od počátku projektu na toto dbát. Podle (Zuzana Štochová a Eduard Kunce 2014) je v některých projektech nedodržení standardizace důvodem k neakceptaci user story v dané iteraci.

Třetím prvkem, se kterým souvisí tento prvek, je CI (viz kapitola 4.8.2.4). Do automatických buildů je možné přidat kontrolu standardizace kódu a tím usnadnit kontrolu dodržení tohoto přístupu.

4.8.3 Výhody a nevýhody XP

Podle (Kadlec 2004) většina výhody metodiky XP plynou z její jednoduchosti a přirozenosti ji aplikovat. Udává jako výhodu, že lidé rádi komunikují, mohou být samostatní, mají rádi funkční software, apod.

Paradoxem těchto výhod, které plynou přirozenosti, je složitost zavádění XP. V současné společnosti je složité se vrátit k jednoduchosti, nebát se pokládat více otázky, i když mohou znít hloupě. Složité je i zjednodušovat pojmy a požadavky, přiznat svou neznalost v některých disciplínách apod. Kadlec se také zmiňuje o vzdělávacím systému, který stále učí individualismu a ne týmové práci. Bývalý více prezident Microsoftu Jan Mühlfeit (Mühlfeit 2016) k systému školství dodává, že když ve škole žák začne spolupracovat s ostatními, tak je potrestán a je špatně ohodnocen, kdežto jeho zaměstnanci dostávají za spolupráci bonusy.

Výhody při zavedení metodiky XP také shrnuje studie (Layman et al. 2004), která dospěla k výsledkům, že byla zvýšena produktivita o 50 %, pre-release kvalita o 65 % a post-release kvalita o 35 %.

4.9 Lean software development

Metodika Lean software development (dále jako Lean) má své kořeny v 50. letech 20. století. S filozofií lean tehdy přišel japonský automobilový výrobce Toyota. Tehdy jim šlo o

odstranění zbytečností z vývoje a produkce a tím zvyšovat efektivitu a snižovat náklady. Tehdejší systém se jmenoval Toyota Production Systém neboli Lean Manufacturing.

Lean manufactring vychází z následujících pěti principů:

- Definice hodnoty
- Identifikace částí hodnotového toku
- Tvorba hodnotového toku
- Princip tahu
- Dosahování dokonalosti

Lean manufacturing definuje sedm typů plýtvání, které zformuloval Taiichi Ohno (Ohno 1978) následovně:

Tabulka 1: Sedm typů plýtvání dle Lean manufacturing (Zdroj: autor, dle [Ohno, 1978])

Česky z (Ohno 1978)	Anglicky z (Ohno 1978)	Anglicky z (Poppendieck a Poppendieck 2003)
Nadprodukce	Waste of overproduction	Overproduction
Čekání	Waste of time on hand (waiting)	Waiting
Přeprava	Waste of transportation	Transportation
Zpracování	Waste of processing itself	Extra processing
Pohyb	Waste of movement	Motion
Zásoby	Waste of stock on hand (inventory)	Inventory
Vady	Waste of making defective products	Defects

Tyto typy plýtvání v roce 2003 upravila kniha Lean software development: An Agile Toolkit (Poppendieck a Poppendieck 2003), která přizpůsobuje Lean manufacturing na vývoj software tedy na Lean software development. Sedm typů plýtvání platících pro software jsou následující:

Tabulka 2: Sedm typů plýtvání v Lean software development (Zdroj: autor, dle (Poppendieck a Poppendieck, 2003))

Sedm typů plýtvání česky	Sedm typů plýtvání anglicky
Nedokončená práce	Partially Done Work
Zbytečné činnosti	Extra Processes
Zbytečná funkcionalita	Extra Features
Střídání úkolů	Task Switching
Čekání	Waiting
Pohyb	Motion
Defekty	Defects

Poppendieckovi k těmto sedmi typům plýtvání přidali ještě osmý typ a to:

- Management Activities

4.9.1 Principy Lean software development

Lean software development vychází podle (Poppendieck a Poppendieck 2003) ze sedmi principů. K naplnění těchto principů slouží různé nástroje, ty vycházejí, pokud není uvedeno v textu jinak ze stejné knihy jako tyto principy a jsou označeny tučným písmem.

4.9.1.1 Eliminace plýtvání

Typy plýtvání byly již zmíněny v kapitole 4.9 Lean.

První zmíněný typ je nedokončená práce. Jedná se o práci, které už byla věnována určitá suma zdrojů a přitom tato práce nepřinesla zatím žádný užitek. Je dost možné, že než se nedokončená práce dokončí, stane se zastaralou a tím pádem budou vynaložené zdroje ztraceny zbytečně.

Druhým typem jsou zbytečné činnosti. V Lean přístupu k software se jedná o činnosti, které nejsou potřebné a nepřinášejí tím pádem žádnou přidanou hodnotu. Podle (Kadlec 2004) může být příkladem provádění určitých činností ručně, přesto že existují automatizované nástroje. Podle (Norrmalm 2011) to může být byrokracie, kdy je třeba hodně dokumentace, podepisování a schvalování dokumentů apod.

Třetí je zbytečná funckionalita. Vyvíjet by se měli pouze funkcionality, které uživatel software bude využívat. Zní to jednoduše, ale často se vyvíjí zbytečnosti, které mají pouze potenciál na využití a ne jistotu. Důležité je mít někoho (např. v SCRUM to řeší Product owner), kdo bude určovat priority požadavků a bude odfiltrovávat zbytečnosti. Podle (Poppendieck a Poppendieck 2003) můžou zbytečné funkcionality zastarávat, než se vůbec začnou využívat.

Střídáním úkolů jednotlivých programátorů dochází k plýtvání jejich výkonností. Pokud programátor pracuje na více projektech, či úkolech naráz zabere mu každá změna spoustu času a energie.

Čekání může nastat při vývoji software například při tom, když jsou jednotlivý členové úzce specializovaní. Příkladem může být výše zmíněný vodopádový model, kde testeři čekají, než vývojáři naprogramují aplikaci.

Pohyb lze především chápat jako zbytečný či problémový přenos informací. Jedním typem dle (Poppendieck a Poppendieck 2003) je přenos informací mezi lidmi. Programátorovi například není jasné, co je míněno danou funkcionalitou, tím pádem vzniká neefektivita, jelikož musí dohledávat zákazníka, či manažera pro upřesnění požadavků. Může se také jedna o tým, který je rozdělen do více místností či budov. Řešením tohoto problému v metodice XP je zákazník na pracovišti a tým v jedné místnosti. Druhým typem je přesun artefaktů, kde dochází k entropii. Příkladem je přesun kódu od vývojářů po testery, kdy testerům trvá delší čas, než kód pochopí. Třetím typem dle (Kadlec 2004) může být čekání, než se stáhne nějaká část aplikaci, či hledání informací o knihovnách, verzích produktů apod.

Posledním typem plýtvání jsou defekty. Zde je zásadní pozdní odhalení chyb. Řešením můžou být například automatizované průběžné testy, unit testy či jako využívání CI jako celku.

Osmým přidaným typem jsou aktivity managementu. Zde se jedná především o neefektivnost ve smyslu provádění aktivit, které nepřinášejí přidanou hodnotu.

Jako nástroje pro naplnění tohoto principu uvádí (Poppendieck a Poppendieck 2003) **vidění plýtvání**, kdy je nejprve nutné hledat, kde se plýtvá, k tomu pomáhá výše zmíněných sedm (plus jedna) typů plýtvání. Druhým nástrojem je **mapa hodnotového toku**, kde se hledají úzká místa procesu vývoje a prostoje.

4.9.1.2 Rozvinout učení

Pod tímto principem si lze představit jak učení členů týmu tak i zlepšování kvality. Jedná se jak o učení ve smyslu technologií a technik, tak především ve smyslu zlepšování komunikace v týmu a se zákazníkem. Například se zde podle (Kadlec 2004) využívá **zpětná vazba**, **iterativní vývoj**, kde se na jednotlivé iterace tým intenzivně soustředí a nakonec **synchronizace** jednotlivých částí vývoje.

4.9.1.3 Pozdní rozhodování

Nejedná se o rozhodování na poslední chvíli ve smyslu odkládání povinností a utíkání před řešením problému. Dle (Stellman a Greene 2015) se jedná o přístup, kde je dobré se rozhodovat až ve chvíli, kdy je nashromážděno dostatečné množství pro správné rozhodnutí. Důležitá je i časová složka, kde je řečeno se, že se jedná o poslední odpovědný moment.

Podle (Kadlec 2004) jsou vhodnými nástroji pro tento princip například: **přidělování zodpovědnosti**, uvažování o všech možných **variantách**, **sdílení informací**, používání **automatizování testovacích případů** a používání **krizových scénářů** při dramatických změnách.

4.9.1.4 Časté dodávky

Někdy též zmiňované jako dodávání tak často, jak je to jen možné. Zde je základem opět **iterativní vývoj**, kde dochází k častým releasům produktu. Jak již bylo zmíněno v principech agilního vývoje jako takové, lze tím zajistit rychlou reakci na změny požadavků. Jako nástroj pro splnění tohoto principu sem lze zařadit i **Kanban**.

4.9.1.5 Pravomocní pracovníci

V tomto principu jde o to vytvořit prostředí, ve kterém se budou pracovníci cítit dobře. Měli by mít dostatek podpory, mělo by jim být naslouchánо a důvěřováno. Stejně tak by měly být rozdělené pravomoci, aby se vytvořili podmínky pro **zodpovědnost** a tím vzniklou **motivaci.** Důležité je mít vhodné osobnostní typy a expertizu lidí, tedy podporovat zaměstnance ve školení **a spolupráci (komunikaci)**. Důležité je také dbát na to aby vedení týmu bylo spíš ve smyslu **leadership**u, než managementu.

4.9.1.6 Integrita

Integritu lze podle (Poppendieck a Poppendieck 2003) rozdělit na integritu vnímání a konceptuální integritu.

Integrita vnímání označuje celkový zákaznický dojem z produktu. Zahrnuje tedy například, jak je produkt prezentován, doručován, instalován, jak je intuitivní na používání, kolik stojí, jak často je updatován, jak se vypořádává s problémy. Jako příklad jak měřit, či zjistit

integritu vnímání uvádí (Poppendieck a Poppendieck 2003) situaci, kdy člověk přijde o počítač. Další den si pořídí nový a bude si chtít nainstalovat programy, ty které se mu vybaví ihned, mají dobrou integritu vnímání.

Konceptuální integrita označuje, jak funguje systém jako celek. Jestli je dobře udržovatelný, flexibilní apod. Příkladem může být organizace, která má dva různé weby. Otázkou konceptuální integrita poté je, jestli je stejný design obou webů. Pokud je špatná konceptuální integrita, je špatná i integrita vnímání.

Pro integritu systému se využívají jako nástroje i **refaktorizace** a **testování** (již zmíněné v metodikách XP a SCRUM)

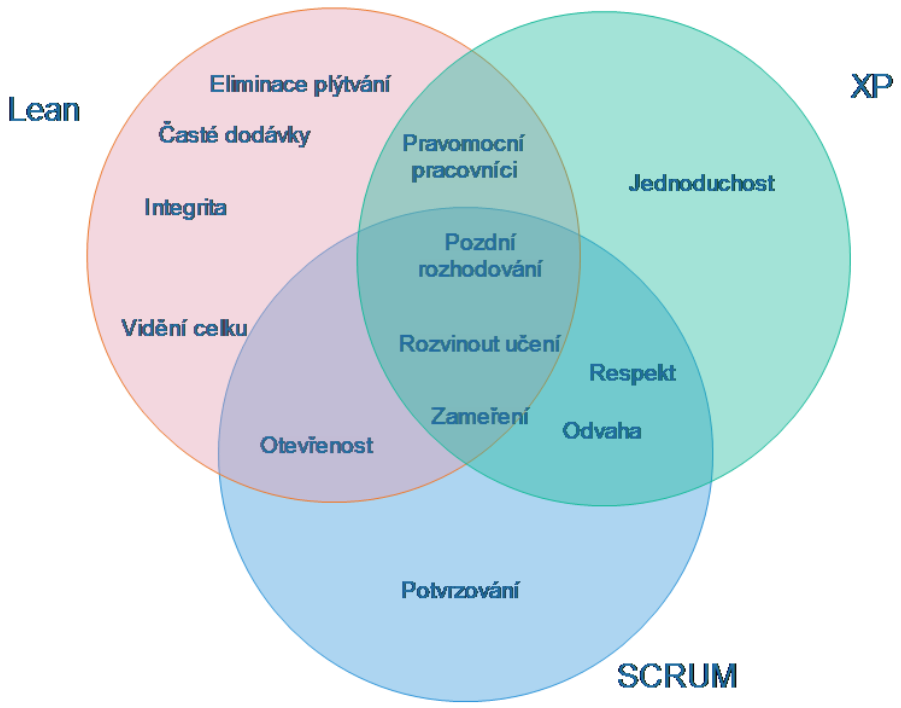
4.9.1.7 Vidění celku

Tento princip se zabývá především optimalizací. Hlavní myšlenkou je zabývat se optimalizací systému jako celku a ne optimalizací jednotlivých částí. Vychází se zde z toho, že zákazník kouká na funkčnost systému jako celku a nezabývá se příliš tím, zda funguje ta nebo ta část systému. Pro dosažení principu vidění celku se využívají různé **metriky** (např. metoda pěti „Proč?“) a správně nastavené **smlouvy**.

4.9.2 Shrnutí Lean software development

Jak je zřejmé z výše zmíněných principů metodiky Lean, nejedná se o striktně dané postupy a procesy, které je třeba dodržovat. Spíše jde o jakousi filozofii jak přistupovat k vývoji software a sadu doporučených praktik jak jednotlivé principy dodržovat.

Pro shrnutí a srovnání hodnot doposud zmíněných metodik poslouží obrázek (viz Obr. 14) převzatý z (Stellman a Greene 2015).



Obrázek 14: Prolínání principů Lean, XP a SCRUM (Zdroj: Autor, dle [Stellman a Greene, 2015, str. 272])

Na tomto obrázku je znázorněné prolínání hodnot jednotlivých metodik. Jako důležitou poznámku k *Potvrzování* uvádí (Stellman a Greene 2015), že potvrzování je důležitou součástí všech tří metodik, avšak SCRUM potvrzování explicitně označuje za hodnotu. Podobné okomentování by šlo analogicky aplikovat na téměř všechny hodnoty. Příkladem jsou nástroje uvedené u metodiky Lean pro princip integrity, tedy refaktorizace a testování, které se využívají i u metodik SCRUM a XP. Tím pádem přinášejí podobné nebo stejné hodnoty. Toto schéma se tedy snaží ukázat ty nepatrné rozdíly mezi jednotlivými metodikami.

Jednou z případových studií, která se zabývá účinností Lean principů je případ cloudového úložiště Dropbox (Houston 2010). Dropboxu se díky Lean principům povedlo vyrůst z 100 000 uživatelů po čtyři miliony za patnáct měsíců. To celé bez velkých reklam.

4.10 Kanban

Kanban dle (Zuzana Štochová a Eduard Kunce 2014) historicky vychází z Japonska, kde prapůvodně sloužil jako systém pro řízení počtu lidí v chrámu. Ten kdo vcházel dovnitř, dostal lístek a při odchodu ho zase odevzdal. Tím, že do chrámu mohl vstoupit pouze člověk s lístkem a počet lístků byl omezen, bylo zajištěno, že v chrámu byl takový počet lidí, jaký kapacita chrámu umožňovala.

Kanban se poté přenesl spolu s metodou Lean manufacturing do japonských automobilek a to především Toyoty.

Poprvé byl dle (Anderson 2010) Kanban využit v softwarovém inženýrství v roce 2004 v Microsoftu.

V IT se Kanban využívá především v týmech, které se starají o údržbu (maintainance) a to díky tomu, že se jedná o velice flexibilní proces, který téměř nic nenařizuje. S tím se pojí rovnou výhoda a nevýhoda Kanbanu – nejsou zde jasně dané procesy a záleží na týmu, jak si zvolí, že bude vyvíjet. Sám autor Kanbanu aplikovaného v IT Anderson upozorňuje, že se nejedná o metodiku. Kanban by dle něho měl sloužit spíše k zlepšení už používané metodiky. I přesto, že Kanban není metodikou, která předepisuje jasné procesy, je poměrně často jako metodika, či přístup k práci využíván.

Tato kapitola vychází, pokud není řečeno, jinak z knihy Kanban: Successful Evolutionary Change for Your Technology Business od Davida J. Andersona (Anderson 2010).

4.10.1 Charakteristika Kanbanu

Kanban pro vývoj software stojí na pěti základních principech:

- Vizualizace toku práce (Visualize Workflow).

  Vizualizace toku práce slouží především k odhalování úzkých míst a k lepšímu přehledu o toku práce. Jako nástroj se zde využívá tzv. Kanban board.

- Omezit rozpracovanou práci (Limit Work-in-Progress, neboli WIP).

  Cílem je omezit počet rozpracovaných úkolů, aby došlo k větší koncentraci na jednotlivé úkoly. Využívá se zde limitů pro jednotlivé fáze workflow.

- Měření a řízení toku

  Princip vychází z vyrovnaného toku, kdy je například nový úkol započat až poté co je dokončen jiný.

- Jasně definovaná pravidla procesu

  Je vhodné mít stanovená pravidla pro určení, jak má workflow postupovat a za jakých podmínek. Pravidlem může být například stanovený limit pro počet aktuálně rozpracovaných úkolů zároveň. Dále lze pravidly nastavit jak se má prioritizovat, apod.

- Používání modelů pro odhalení příležitostí k zlepšení.

  Jedná se o využití Kanban pro sledování produktivity, hledání úzkých míst a jejich následné odstraňování.

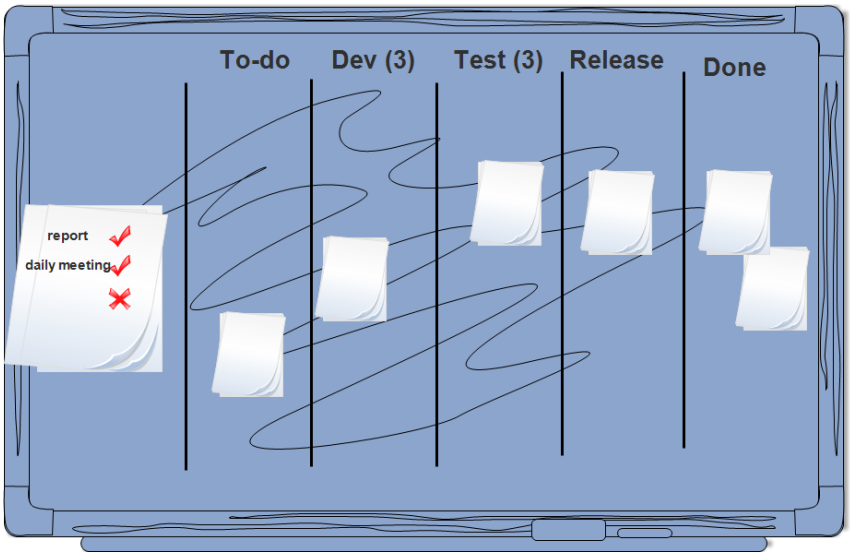
Kanban, i přesto že je škálovatelný, je vhodný spíše pro menší projekty. Pokud bychom chtěli využít Kanban pro velké projekty, byl by nepřehledný hlavní nástroj Kanbanu – Kanban board.

Kanban vychází z karet, kde každé činnosti je přidělena jedna karta. Tímto přístupem lze omezit počet jednotlivých činností v systému. Pokud bychom chtěli vytvořit novou činnost, ne třeba nejprve nějakou činnost dokončit a tím pádem se uvolní karta pro použití na novou činnost.

Kanban nemá předepsané role či artefakty, ale pracuje s několika pojmy, které je dobré vysvětlit:

- Backlog (To-do) – již vysvětleno v kapitolách 4.7.2.6. Product backlog a 4.7.2.7 Sprint backlog věnovaných metodice SCRUM.
- Úkol (Činnost, Work item, task) – jedná se o danou činnost, která je většinou symbolizována již zmíněnými kartami
- Dráha (Swimline, fáze) – jde o horizontální rozdělení Kanban tabule na určité logické celky. Těmito celky mohou být například: různé priority, členové týmu či týmy.
- WIP – jedná se o aktuálně rozpracované úkoly.
- WIP limit – určuje maximální počet WIP.
- Kanban board (tabule) – označení pro fyzickou, či softwarovou tabuli, kde jsou rozmístěny jednotlivé Kanban karty.
- Lead time – čas potřebný k dokončení úkolu.

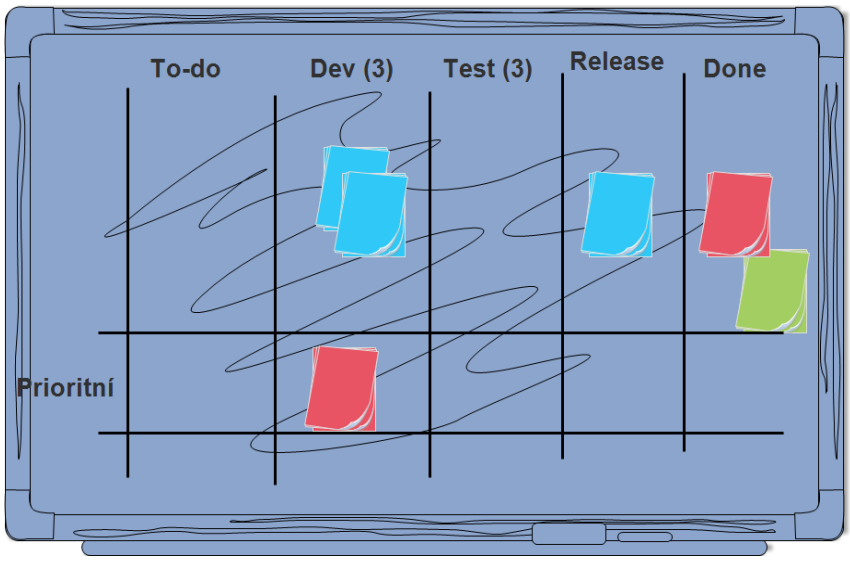
Pro ilustraci kanban boardu poslouží níže uvedený obrázek (viz Obr. 15), který zároveň ukazuje WIP limit. Toho si můžeme všimnout u jednotlivých názvů sloupců. Pro opakující se úkoly lze využít odškrtávací seznam (viz Obr. 15), jelikož by se při klasickém využití karet tabula stávala nepřehlednou.



Obrázek 15: Kanban board (Zdroj: autor)

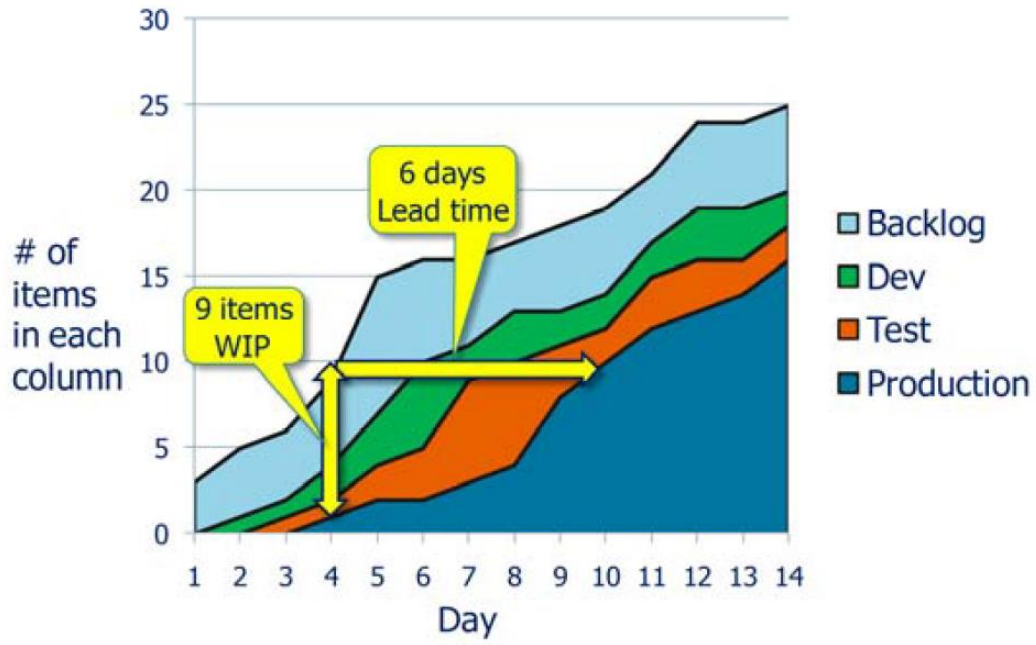
Jednou z best pracites je zavedení plavecké dráhy, ve které budou položky, které například čekají na schválení. Ty se nepočítají do limitu WIP, jelikož se s nimi nepracuje. V korporátní prostředí je dost často nutné čekat, než se vyřídí složité schvalovací procesy a to by týmu mohlo ubírat kapacitu.

Pro prioritizování Kanban karet se používá barevné odlišení karet. Toto lze analogicky aplikovat i na rozdělení z pohledu vývoje, kdy může být barevně rozlišeno, zda se jedná o vývoj nové funkcionality, bug či něco jiného. Pro určení těchto hodnot karet lze využít již zmíněné swimliny, kam se mohou jednotlivé karty umisťovat. Kombinací těchto dvou pomůcek lze docílit, že na jedné tabuli budou zobrazeny jak priority, tak typy jednotlivých činností. Na obrázku (viz Obr. 16) můžeme vidět červeně označené bugy, modře nové funkcionality a zeleně údržbové práce. Jeden bug je zařazen v prioritní swimline, bude mít tedy přednost před novými funkcionalitami, které jsou ve stejném sloupci. Na stejném obrázku je znázorněná další praktika, kdy se jedna činnost přelepí přes další a tím se označí závislost těchto činností.



Obrázek 16: Kanban board se swimline (Zdroj: autor)

Další pomůckou pro aplikaci Kanbanu je Cumulative Flow Diagram, který se využívá pro sledování počtu rozpracovaných úkolů v čase a tím umožňuje identifikovat úzká místa ve workflow. Na obrázku (viz Obr. 17) je možné vidět Cumulative Flow Diagram. Pro každý den je poté možné určit počet WIP na ose Y a jednotlivé dny na ose X. Na obrázku je vidět, že byla jedna činnost v testovací fázi, dvě činnosti ve fázi vývoje a 5 činností v backlogu, tedy celkem 9 činností. Z bodu kde končí Backlog lze vést horizontální přímku (kolmici k ose X) a tam kde se kolmice dotkne stavu Hotovo, přímka končí. To nám umožňuje určit průměrnou čas potřebný k dokončení úkolu – tedy lead time, který v tomto případě byl 6 dnů.



Obrázek 17: Cumulative flow diagram (Zdroj: Kniberg a Skarin, 2010, str. 39)

4.10.2 Výhody a nevýhody Kanbanu

Pokud bychom chtěli srovnávat přístup metodiky SCRUM a Kanbanu, je dobré si představit Scrumboard a Kanbanboard. Scrumboard bude mít na začátku sprintu plný první sloupec, v průběhu spritnu se budou jednotlivé činnosti přesouvat a na konci sprintu dojde k naplnění posledního sloupce. Oproti tomu Kanban bude mít téměř vždy nějaké karty ve všech sloupcích. Tato ilustrace podkresluje využívání Kanbanu v maintainance týmech.

Dalším problémem je, že Kanban téměř nic nepředepisuje, což může být a korporátní prostředí až příliš velká benevolence, kterou je díky firemním procesům těžké připustit.

Zuzana Štochová ve své knize (Zuzana Štochová a Eduard Kunce 2014) zdůrazňuje, že úspěšné implementace se neskládají pouze z Kanbanu, ale z kombinace s ostatními metodikami, které nějaké procesy předepisují, např.: XP či SCRUM.

Pro podporu aplikace Kanbanu hovoří různé studie. Jednou z nich je (M.C. Partners & Associates 2015), ve které byla pomocí Kanbanu zkrácena doba dodání na trh z šedesáti a více dnů na pouhých třináct dnů. Zvýšena byla také kvalita dodávaného produktu a bylo také dosaženo finančních úspor.

4.11 Ostatní agilní metodiky

Existuje obrovské množství agilních metodik. Mezi významné agilní metodiky, které nebyly v práci zmíněny, patří například FDD (Feature-Driven Development) a TDD (Test-Drive Development)

FDD je iterativní metodikou, kdy je vývoj produktu řízen vlastnostmi produktu. Dle (Buchalcevová 2009) se jedná o metodiku, která již definuje nějaké procesy a je formálnější než ostatní agilní metodiky. Lze ji tedy použít na rozsáhlejší projekty.

Základním prvkem TDD je dle (Doucek et al. 2013) povýšení testování na základní aktivitu v rámci vývojového cyklu. Dost často je totiž možné se setkat s přístupem, kdy je testování bráno jako nepříliš podstatná část vývoje. Principem TDD je nejprve napsat test, který testuje chtěnou funkcionalitu, až poté se píše kód. Jakmile test projde, přestává se s kódem, protože splňuje požadavek.

Mezi další agilní metodiky patří například Crystal, MSF4ASD (Microsoft Solution Framework for Agile Software Development, Agile Unified Process a další.

5 Výběr vhodné metodiky a její aplikace

Pro výběr vhodné metodiky pro projekt jsou využity dva hlavní přístupy. Jedním je vícekriteriální rozhodování TOPSIS na základě charakteristik projektu a druhým přístupem je metodika METES.

Dle (Buchalcevová, 2009) jsou těžké neboli rigorózní metodiky vhodné pro rozsáhlé projekty, které řeší velké týmy, a tady tomu tak není. Což je důvodem, proč tyto metodiky nebrat v potaz při výběru.

Podle (Kadlec, 2004)je SCRUM vhodný pro malé týmy, které nevyvíjí příliš rozsáhlé projekty. A především, třeba v porovnání s XP, je kladen větší důraz na řízení týmu a možností do organizace zasáhnout. To je z hlediska korporátního prostředí velmi důležitá vlastnost.

Při porovnávání metodiky SCRUM a XP pro internetové projekty, což výše zmíněný projekt v podstatě je, udává Kadlec ve své knize, že ve prospěch XP hovoří větší množství informací o metodice XP. Toto je nyní zcela jistě neplatná informace, jelikož od doby vydání jeho knihy, se poměry využívání metodik změnili ve prospěch SCRUM.

Dále Kadlec (Kadlec 2004, s. 135) ve své knize, s odkazem na zakladatele metodiky Kenta Becka, upozorňuje na využívání metodiky XP jako celku: *„Kent Beck věnuje hodně energie upozornění, že o Extrémní programování se jedná pouze v případě, že dodržujeme všechny uvedené činnosti zároveň. Pokud zavedeme do týmu pouze čtyřicetihodinovou pracovní dobu, refaktorizaci a průběžné testování, nemůžeme tvrdit, že využíváme XP.“* Pokud bychom však přistoupili na toto upozornění od zakladatele metodiky XP, bylo by velmi obtížné aplikovat metodiku XP.

Kanban se dle (Zuzana Štochová a Eduard Kunce, 2014) hodí převážně pro údržbové týmy, kde nejde práce rozdělit na iterace, a pro vývoj je tedy vhodné volit metodiky XP či SCRUM.

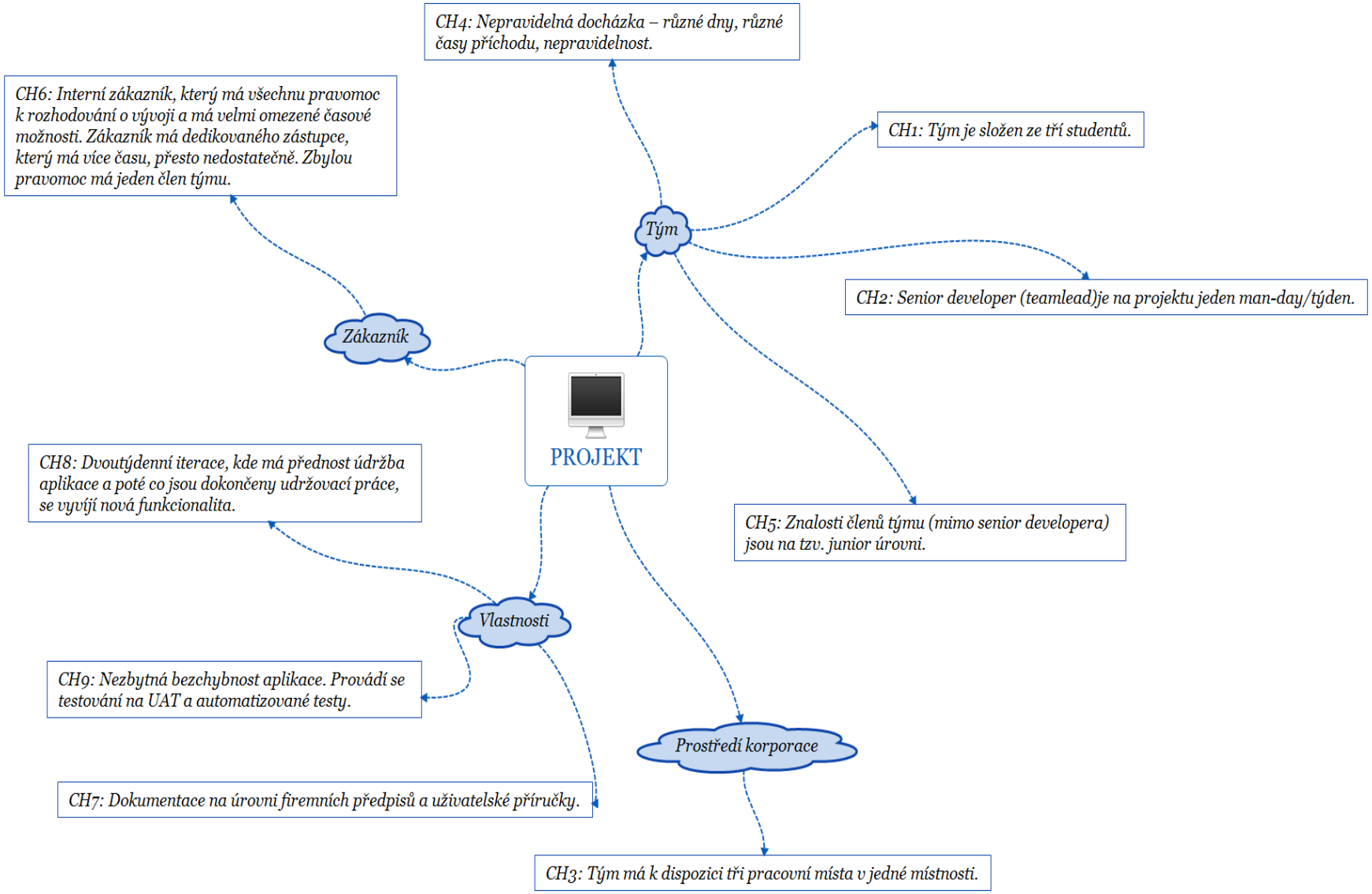
Aktuálním trendem je myšlenka kombinace metodik. K této myšlence se připojuje i (Stellman a Greene, 2015), kde jsou jednotlivé metodiky odděleny. SCRUM je popsán jako kompletní systém, který slouží pro řízení a dodání projektu. XP se zaměřuje především na samotný vývoj. Kanban pomáhá týmům zjednodušit a zlepšit cestu při vývoji software. Lean je brán jako myšlenkový koncept, kterým lze propojit všechny tyto přístupy dohromady.

Pro výběr vhodné metodiky či přístupu pro projekt jsou použity dva přístupy, jedním je vícekriteriální rozhodování a druhým je metodika METES.

5.1 Výběr vhodné metodiky pomocí vícekriteriálního rozhodování TOPSIS

Pro výběr vhodné metodiky je použita vícekriteriální rozhodování TOPSIS hodnotící jednotlivé metodiky z pohledu naplnění charakteristik projektu.

Pro lepší přehlednost jednotlivých charakteristik uvedených v kapitole 3 Charakteristika vybraného projektu je vytvořena myšlenková mapa (viz Obr. 18). Hlavním námětem této myšlenkové mapy je projekt zpracovávaný v této práci. Z tohoto námětu se větví jednotlivé charakteristiky projektu označené v kapitole 3 Charakteristika vybraného projektu.



Obrázek 18: Myšlenková mapa charakteristik projektu (Zdroj: autor)

Vícekriteriální rozhodování neboli vícekriteriální hodnocení variant se zabývá problematikou výběru nejlepší varianty. Nejlepší varianta je vždy vybrána z konečného počtu variant, které jsou hodnoceny. Většinou je kritériím přiřazena různá důležitost, tedy váha. Kritéria, která mají větší váhu, jsou pro rozhodování důležitější než kritéria s menší váhou. Vícekriteriální rozhodování se používá v situacích, kde nelze na první pohled vidět jednoznačnou převahu jedné varianty, kterou by bylo vhodné si vybrat.

Metoda TOPSIS (Technique for Order Preference by Similarity to Ideal Solution) je založena na minimalizaci vzdálenosti od ideální varianty. Čím více se jednotlivé varianty blíží k ideální variantě, tím lepší jsou. Ideální varianta je reprezentovaná nejlepší možnou volbou. Opakem ideální varianty je bazální varianta, která říká, že čím je větší vzdálenost od této varianty, tím se jedná o lepší variantu. Bazální varianta je reprezentovaná nejhorší možnou volbou.

Pro každou charakteristiku projektu jsou uvedeny techniky, principy či vlastnosti metodiky, které danou charakteristiku pomáhají vyřešit.

Následně jsou metodiky ohodnoceny na škále, která vychází z předchozího systémového přístupu k jednotlivým agilním metodikám.

Hodnocení je rozděleno po následujících krocích (pro metodu výpočtu TOPSIS bylo využito minimalizace kritérií):

1 – zcela řeší charakteristiku

3 – převážně řeší charakteristiku

5 – částečně řeší charakteristiku

7 – téměř neřeší charakteristiku

9 – neřeší charakteristiku

Ohodnocení a přiřazení praktik a technik je možné vidět v následující tabulce (viz Tab. 3).

Tabulka 3: Ohodnocení řešení charakteristik projektu pomocí technik a praktik jednotlivých metodik (Zdroj: autor)

Charakteristika	Metodika	Praktika či vlastnost	Ohodnocení
CH1	SCRUM	Doporučená velikost týmu je 5-9 lidí.	5
	XP	XP je vhodný pro malé týmy, ale jak klasické XP, tak XP pro webové projekty předpokládá více rolí.	7
	Lean	Lean se o velikosti týmu nezmiňuje.	7
	Kanban	Neřeší velikost týmu, jde o přehlednost Kanban-boardu, který bude ve 4 lidech přehledný.	5
CH2	SCRUM	Nedoporučuje se, aby kdokoliv pracoval na více projektech. Teamlead v roli Scrum mastera nebude mít dostatek času se intenzivně věnovat týmu.	7
	XP	Jak již bylo zmíněno, XP předpokládá více rolí a role stratéga či kouče na 1 man-day je nedostačující.	7
	Lean	Lean definuje tzv. task-switching.	9

	Kanban	Kanban board umožňuje se rychle zorientovat a přiřazovat priority.	5
CH3	SCRUM	Tým má být na jedno místě.	1
	XP	Tým má být na jedno místě.	1
	Lean	Plýtvání typu Pohyb hovoří o výhodách společného usazení.	3
	Kanban	Při využití fyzického Kanban boardu nedostačující, při využití software boardu je problém eliminován.	3
CH4	SCRUM	K synchronizaci pomáhají meetingy (Daily scrum, retrospektiva, apod.) a Scrum board.	1
	XP	Kolektivní vlastnictví kód – přístup ke kódu.	5
	Lean	Lean by mohl chápat jako přenos informací mezi studenty jako plýtvání. Hovoří o tom plýtvání typu Pohyb.	5
	Kanban	Kanban board je přístupný stále.	1
CH5	SCRUM	Lze aplikovat menší počet story pointů udělaných za sprint.	5
	XP	Párové programování	1
	Lean	Jeden z principů Lean je Pravomocní pracovníci, kde se přímo hovoří o rozvoji pracovníků.	1
	Kanban	Kanban umožňuje pouze sledovat a měřit výkonost, avšak nepodporuje učení přímo.	7
CH6	SCRUM	Product Owner a Proxy Product Owner / Prioritizování	1
	XP	Zákazník má být s týmem na pracovišti neustále.	9
	Lean	Jedním typem plýtvání je pohyb. Díky složité komunikaci vidí lean toto jako problematické.	7
	Kanban	Prioritizování činností na Kanban board může provádět zákazník či ten kdo má jeho pravomoc	5
CH7	SCRUM	Jako dokumentace slouží product backlog a zápisy v něm.	1
	XP	XP označuje jako dokumentaci samostatný kód.	5
	Lean	Lean pod firmou vyžadovanými dokumenty vidí plýtvání.	9
	Kanban	Kanban nepracuje s žádnou dokumentací. Pouze lze využít Kanban board, kde budou evidovány činnosti týkající se dokumentace.	7
CH8	SCRUM	Doporučuje se iterace 2-4 týdny.	1
	XP	Dvoutýdenní iteraci odpovídají malým verzím, o kterých XP pojednává, refaktorizace pomáhá při údržbě.	1
	Lean	Časté dodávky jsou jedním z principů Lean.	1
	Kanban	Neexistují iterace. Kanban se nehodí příliš pro vývoj, ale pro údržbu naprosto.	5
CH9	SCRUM	Pomocí meetingů se předchází nejasnostem a provádí se testování.	3
	XP	Využívá se CI (s automatizovanými testy), standardizace, kolektivní vlastnictví kódu.	1
	Lean	Princip pozdního rozhodování.	3
	Kanban	Lze pouze vytvořit sloupec Test a v něm řešit testování jednotlivých činností.	5

Ke stanovení vah kritérií je použita bodovací metoda (Pelikán et al., 2011), kdy každý člen týmu má přiděleno 100 bodů, které rozdělí mezi devět charakteristik. Bodování všech členů týmu včetně teamleada mají stejnou váhu a je možné ho vidět v následující tabulce (viz Tab. 4).

Tabulka 4: Přidělené váhy charakteristik od jednotlivých členů týmu (Zdroj: autor)

	Člen 1	Člen 2	Člen 3	Člen 4	Celkem	Váha
CH1	8	6	25	17	56	0,1400
CH2	8	16	5	10	39	0,0975
CH3	8	10	0	7	25	0,0625
CH4	6	12	10	17	45	0,1125
CH5	20	17	10	7	54	0,1350
CH6	20	7	15	12	54	0,1350
CH7	2	2	5	3	12	0,0300
CH8	18	20	20	17	75	0,1875
CH9	10	10	10	10	40	0,1000
Celkem	100	100	100	100	400	1

Nejprve je dobré zkontrolovat, zda neexistují dominantní varianty (Pelikán et al. 2011), tedy pokud některá varianta je všech ohledech stejná, nebo lepší. Z pohledu na tabulku je vidět, že XP je dominantní Lean. Je tedy jasné, že Lean nemůže být vyhodnocen jako nejvhodnější přístup k danému projektu. Přesto je pro názornost vypočtena též.

Pro výpočet je využit doplněk pro aplikaci Excel, který se jmenuje SANNA 2014 (Jablonský 2014). Jelikož není cílem BP se věnovat vícekriteriálnímu rozhodování, jsou pouze v příloze A: Výpočty vícekriteriálního rozhodování TOPSIS, uvedeny tabulky výpočtu pomocí SANNA 2014.

Tabulka 5: Výpočet vícekriteriálního rozhodování TOPSIS (Zdroj: autor)

	SCRUM	XP	Lean	Kanban	Váha
CH1	5	7	7	5	0,1400
CH2	7	7	9	5	0,0975
CH3	1	1	3	3	0,0625
CH4	1	5	5	1	0,1125
CH5	5	1	1	7	0,1350
CH6	1	9	7	5	0,1350
CH7	1	5	9	7	0,0300
CH8	1	1	1	5	0,1875
CH9	3	1	3	5	0,1000
CI	**0,72361**	**0,49583**	**0,44858**	**0,47079**	**1**

Největší hodnotu relativního ukazatele vzdálenosti, v tabulce (viz Tab. 5) označené jako CI, má metodika SCRUM. Metodika SCRUM je tedy v závislosti na daném hodnocení přístupu k charakteristikám projektu a v závislosti na týmem určených váhách, nejvhodnější metodikou pro daný projekt.

5.2 Výběr vhodné metodiky pomocí METES

Pro jiný pohled na výběr agilní metodiky byl vybrán systém METES od (Buchalcevová 2009). Systém hodnocení a výběru metodik METES je určen pro výběr informační systém. Tento systém tedy není určen přímo pro výběr metodiky pro projekty obdobné velikosti a obdobného typu. Určitá analogie ovšem existuje a proto byl tento systém pro rozhodování o vhodnosti metodik vybrán.

Tuto kapitolu lze chápat jako jiný typ pohledu na výběr metodik, ne jako výběr samotný a to z důvodů, které jsou uvedeny níže.

5.2.1 Kritéria

Hodnocení kritérií bylo vybráno pouze mnou a mým pohledem na projekt. Pro více informací o těchto kritériích slouží výše zmíněná kniha (Buchalcevová 2009).

5.2.1.1 Kritéria skupiny Proces

- Kritérium Rozsah: 3 – procesy implementace SW, podpory SW a projektové procesy
- Kritérium Model životního cyklu: 5 – iterativní model, velmi krátké iterace (do měsíce)
- Kritérium Role: 1 – 2-5 SW inženýrských rolí v projektu
- Kritérium Podrobnost popisu procesu: 0 – nejsou definovány žádné procesy
- Kritérium Dokumenty: 2 – jen dokument Specifikace softwarových požadavků
- Kritérium Metriky: 1 – definuje některé metriky a způsob měření, ale jejich význam nepřeceňuje
- Kritérium Řízení kvality: 2 – provádí se testování softwaru

5.2.1.2 Kritéria skupiny Podpora

- Kritérium Celistvost zdrojů: 4 – metodika je v jednom základním zdroji a navíc jsou k dispozici další zdroje
- Kritérium Dostupnost: 4 – open-source licence
- Kritérium Podpora metodiky softwarovými nástroji: 2: metodika je přímo zabudována do jednoho vývojového nástroje
- Kritérium Podpora zavedení metodiky: 2 – poskytnutí konzultantů do firmy, která metodiky zavádí
- Kritérium Přizpůsobení metodiky: 4 – doporučuje přizpůsobování metodiky i v průběhu projektu
- Kritérium Výuka na vysokých školách: 4 – učí se jen v rámci přednášek v ČR
- Kritérium Školení a certifikace: 5 – školení a certifikace i v ČR
- Kritérium Lokalizace: 0 – není v českém jazyce

5.2.1.3 Kritéria skupiny Produkt

- Kritérium Důležitost produktu: 2 – systém podporující fungování organizace
- Kritérium Délka projektu: 5 – nad 24 měsíců
- Kritérium Stálost požadavků: 2 – procento změn požadavků je cca 30%
- Kritérium Znuvupoužitelnost: 3 – snaha vytvářet znovupoužitelné spustitelné komponenty v rámci projektu
- Kritérium Velikost řešení: 2 – 41-100 případů užití

5.2.1.4 Kritéria skupiny Lidé

- Kritérium Zkušenost manažera projektu: 0 – 5+ let
- Kritérium Kvalifikace členů týmu: 0 – více než 70% členů týmu je kvalifikovaných s širokým zaměřením
- Kritérium Motivace členů týmu: 0 – motivovaní jedinci vysokých morálních kvalit, sdílí znalosti, sami se organizují
- Kritérium Dostupnost uživatelů: 1 – uživatel je k dispozici denně
- Kritérium Velikost týmu: 0 – 1-4
- Kritérium Rozmístění: 0 – v jedné místnosti

5.2.2 Výběr metodiky

Jelikož jsou v knize hodnoceny pouze dvě z metodik, které jsou v této práci zmiňovány, je zde posouzena vhodnost pouze těchto dvou metodik a to XP a SCRUM. Jednotlivé hodnoty vychází z výše zmiňovaných ohodnocených kritérií a z hodnot jednotlivých metodik, které jsou určeny v (Buchalcevová 2009). Některé z těchto hodnot již mohou být zastaralé, příkladem je kritérium Lokalizace, kde je jak pro SCRUM, tak pro XP uvedeno, že metodika není v českém jazyce. Jelikož i oficiální průvodce metodikou SCRUM (Schwaber a Sutherland 2013) je lokalizovaný do českého jazyka a existuje spousta dalších zdrojů, je nutné brát ohled na aktuálnost podobných hodnot.

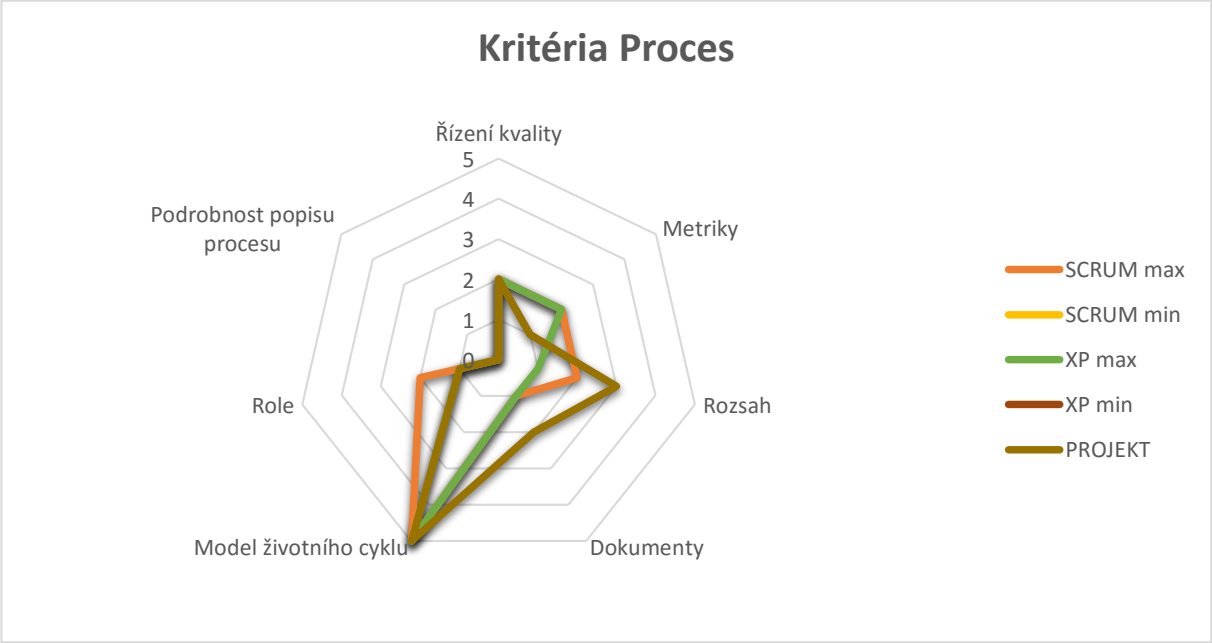
Ze systému METES jsou vybrána kritéria, ty jsou ohodnocena a srovnána s kritérii výše zmíněných dvou metodik. Systém METES není využit pro přímý výběr, jelikož by nebylo dostatečné mít pouze dvě metodiky a v systému METES jsou jako další zvažovány metodiky RUP, OpenUP, FDD a MSF, což je pro tuto práci nedostatečné.

Všechny grafy v následujících podkapitolách vycházejí z přílohy B: Výpočty systému METES, kde jsou tabulky, ze kterých jednotlivé grafy vycházejí. Minimální a maximální hodnoty uvedené pro metodiky SCRUM a XP vycházejí z (Buchalcevová 2009).

5.2.2.1 Kritéria Proces

Z porovnání kritérií ze skupiny Procesy je v grafu (viz Graf 4) vidět obdobný tvar grafů jak mezi metodikami XP a SCRUM tak i v porovnání s projektem. Nedostatečnost metodik je vidět u kritéria Dokumenty, to si lze snadno domyslet, jelikož se jedná o korporátní prostředí, kde je určitý stupeň dokumentace nezbytný.

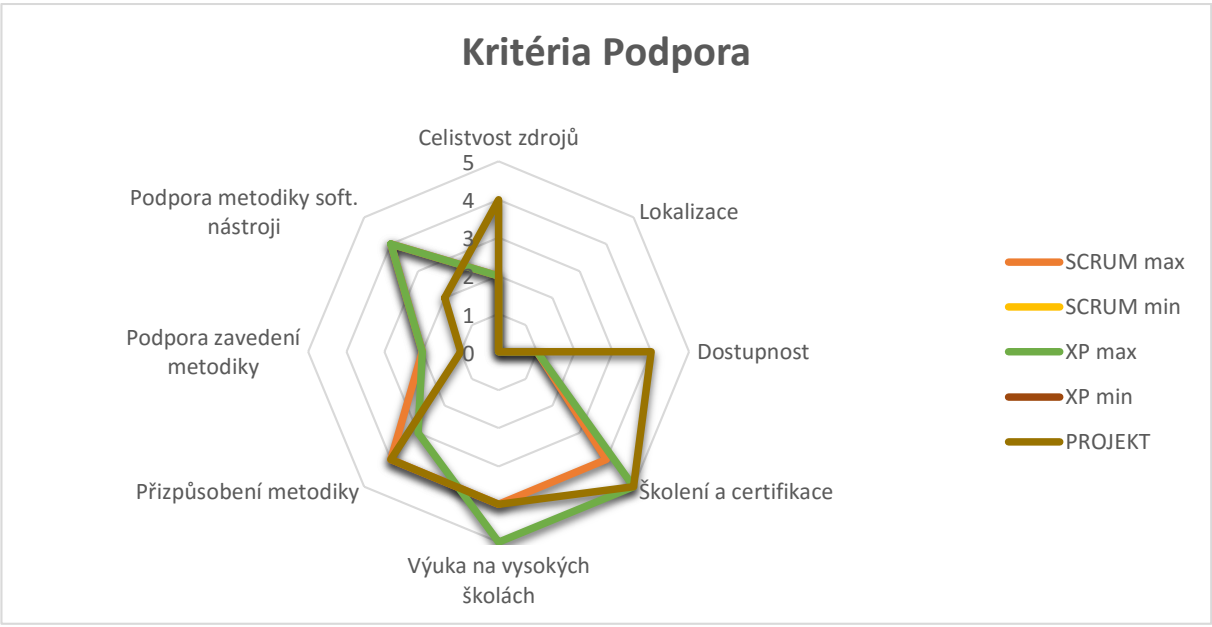
O něco blíže má k projektu díky hodnocení Rozsahu metodika SCRUM.



Graf 3: Paprskový graf ukazující Kritéria Proces (Zdroj: autor)

5.2.2.2 Kritéria Podpora

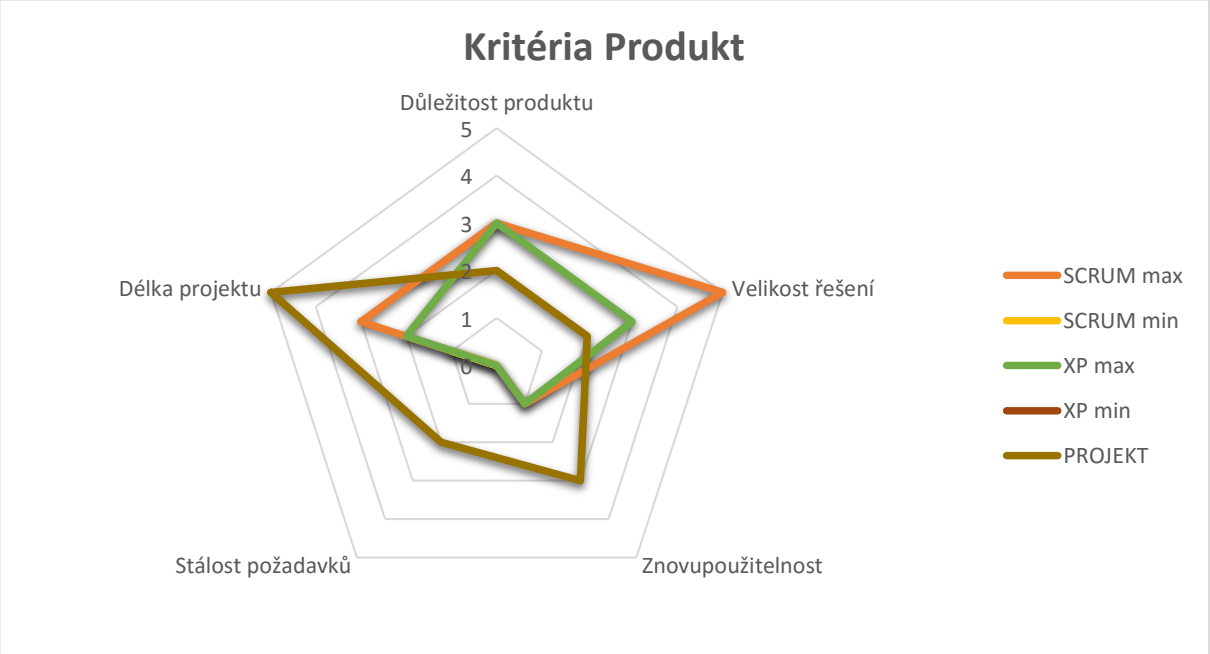
Při pohledu na graf (viz Graf 5) kritérií Podpora můžeme rovněž podobnost tvaru všech tří porovnávaných položek. V kritériu Školení a certifikace je pro projekt vhodnější metodika XP, avšak zrovna o tomto kritériu by se dalo pochybovat, jelikož od doby vydání (Buchalcevová 2009) uplynulo již několik let a nyní je dle mého názoru podpora metodiky SCRUM na lepší úrovni. Naopak z pohledu kritéria Přizpůsobení metodiky je vhodnější SCRUM. Obě metodiky selhávají v kritériích Celistvost zdrojů a Dostupnost. Tyto dvě kritéria jsou nejspíše s uplynulým časem taktéž neaktuální.



Graf 4: Paprskový graf ukazující Kritéria Podpora (Zdroj: autor)

5.2.2.3 Kritéria Produkt

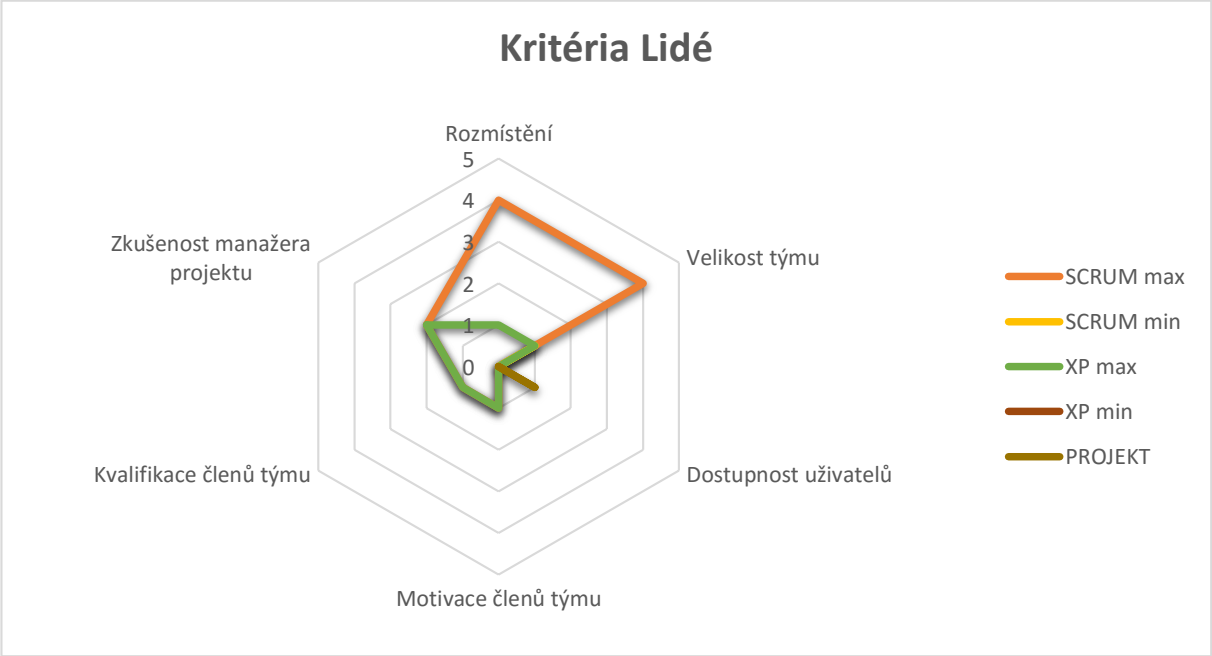
Ve skupině kritérií produkt je již na grafu (viz Graf 6) vidět rozdílný tvar položek. O něco blíže má projekt k metodice SCRUM a to díky délce projektu. Obě metodiky ovšem selhávají v kritériích Stálost požadavků a Znovupoužitelnost.



Graf 5: Paprskový graf ukazující Kritéria Produkt (Zdroj: autor)

5.2.2.4 Kritéria Lidé

Skupina kritérií Lidé je stejně jako předchozí skupina nejednotná pro metodiky a projekt. Z grafu (viz Graf 7) je možné vidět, že ani jedna metodika nesplňuje kritérium Dostupnost uživatelů. Co však z grafu patrné není, ale je to patrné z tabulky (viz Tab. 12: Kritéria skupiny Lidé ze systému METES v příloze B), že metodika SCRUM nesplňuje kritérium Velikost týmu. Ostatní kritéria jsou již splněna oběma metodikami.



Graf 6: Paprskový graf ukazující Kritéria Lidé (Zdroj: autor)

5.3 Aplikace vybrané metodiky

Jestliže je vybrána na základě dosavadního zhodnocení metodika SCRUM, lze se podívat, kde tkví její slabiny.

Z pohledu charakteristik projektu řešených ve vícekriteriálním rozhodování se jedná o charakteristiky CH1, CH2 a CH5. Mimo tyto charakteristiky je možné celou metodiku SCRUM aplikovat na projekt, nebo jen vybrat principy a techniky, které dané charakteristiky řeší.

První charakteristiku, tedy velikost týmu neřeší žádná ze zkoumaných metodik. Pro SCRUM i XP se jedná o moc malý počet členů týmu. Stejné hodnocení pro tuto charakteristiku má i Kanban, který se ovšem velikostí týmu nezabývá vůbec, takže řešení nepřináší.

U druhé charakteristiky je problém s tím, že se senior developer věnuje projektu pouze jeden man-day za týden. S problematikou této charakteristiky by mohlo pomoci využívání Kanban boardu v TFS, jelikož by tím odpadala složitost přenosu informací. Nejedná se o všem o dostatečně kvalitní řešení a všeobecně by se projektu měli věnovat lidé na plný úvazek.

S pátou charakteristikou se váže problematika znalostí týmu jako celku, jelikož se jedná o studenty, mimo zmiňovaného senior developera. Pokud je uvažován SCRUM jako výchozí metodika, došlo by k vyřešení této problematiky využitím párového programování z metodiky XP.

6 Závěr

Současný trend využívání agilních metodik není použití jedné metodiky a striktním aplikování jejích pravidel, ale o určení vhodného rámce pro daný projekt a o výběru vhodných agilní technik a jejich správnému mixu. Což dokazuje kapitola 5. Výběr vhodné metodiky a její aplikace, kdy zde byla vybrána metodika SCRUM a to i přesto, že nenaplňovala všechny požadavky úplně dokonale. Pokud by ovšem práce připouštěla možnost kombinování praktik, bylo by zcela jistě dosaženo lepších výsledků.

Jako ideální se jeví kombinace všech v této práci rozebíraných metodik, kdy SCRUM slouží jako systém pro řízení a dodání projektu, XP rozšiřuje samostatný vývoj, Kanban pomáhá zjednodušovat a zlepšovat vývoj a údržbu a Lean je možné brát jako myšlenkový koncept, který celý agilní přístup k vývoji zastřešuje.

V kapitole o tradičních přístupech k vývoji softwaru byli vybraní zástupci těchto přístupů popsáni. Důvod, proč tyto přístupy nezvažovat pro vývoj software u projektu řešeného v této práci je, že těžké metodiky jsou vhodnější pro rozsáhlé projekty, které řeší velké týmy, a tady tomu tak není.

V této práci byly stanoveny dva hlavní cíle. Prvním cílem bylo pomocí systémového přístupu a analýzy prozkoumat a detailně popsat vybrané agilní metodiky. Druhým a hlavním cílem bylo vybrat vhodnou metodiku pro studentský projekt a srovnat výsledek s aktuálně využívaným přístupem na projektu.

Těchto dvou cílů je dosaženo následovně:

- Nejprve byl zanalyzován studentský projekt, jeho charakteristiky a současné přístupy. Poté byly v kapitole 4 Systémový přístup popsány aktuální trendy a všeobecné charakteristiky, výhody a nevýhody agilních metodik. Dalším krokem byl detailní popis jednotlivých metodik. Jako první byla v kapitole 4.7 Scrum rozebrána metodika SCRUM. Hned po ní byla v kapitole 4.8 Extrémní programování (XP) rozebrána metodika XP. Za ní následoval v kapitole 4.9 Lean a následně v kapitole 4.10 Kanban. Jako poslední byl úvod do ostatních agilních metodik v kapitole 4.11 Ostatní agilní metodiky.
- Druhého a hlavního cíle bylo dosaženo pomocí vícekriteriálního rozhodování TOPSIS. Nejprve byly k jednotlivým charakteristikám projektu přiřazeny praktiky a přístupy jednotlivých metodik. Poté byly jednotlivé metodiky ohodnoceny, dle toho, na kolik naplňují a řeší charakteristické vlastnosti a problematiku projektu. Jednotlivým charakteristikám byla poté od každého člena týmu přiřazena váha. Po ohodnocení charakteristik a přiřazení vah, byl pomocí aplikace Excel a doplňku SANNA 2014 proveden výpočet, ze kterého vyplynulo, že nejlépe naplňuje potřeby projektu metodika SCRUM. Dalším krokem bylo zhodnotit vhodnost metodik vůči projektu a to pomocí systému METES. Tím se zároveň naplnil podcíl, kterým bylo porovnat aktuální přístup využívaný na projektu s výsledkem zkoumání. Dochází se tedy k závěru, že projekt využívá téměř ideální přístup. Zlepšení by nastalo, pokud by se aplikovaly na některé další principy metodik SCRUM, XP a přístupů z Lean a Kanban.

6.1 Přínos práce

Přínos této práce je lze rozdělit do dvou kategorií a to teoretické části a praktické.

V teoretické části se čtenář této práce dozví dostatečné množství informací k tomu, aby pochopil agilní principy a porozuměl zkoumaným metodikám. V textu jsou faktické informace o jednotlivých metodikách vycházejících především z textů, které metodiky formovaly. Dále je text doplněn o vybrané best practices a různé případové studie. Přínosem této části práce je též aktuální pohled na využívání agilních metodik a jejich praktik.

V praktické části práce je zkoumána možnost aplikace agilních metodik na projekt. Nejprve je ohodnoceno, jak metodiky splňují požadavky projektu a poté je přidělena jednotlivých charakteristikám projektu důležitost neboli váha. Pro výběr metodik je použito vícekriteriálního hodnocení TOPSIS. Jako doplnění pohledu na věc je využito systému METES, který je využit pro zhodnocení naplnění charakteristik projektu metodikami SCRUM a XP.

6.2 Možná budoucí rozšíření práce

Navrhované možné rozšíření práce by mělo za cíl sestavit vlastní metodiku pro projekt. Vycházet by bylo možné a doporučené z již existujících metodik. Důvodem pro toto rozšíření by bylo, že i přesto, že je v této práci metodika vybrána, neřeší úplně všechny charakteristiky projektu dokonale.

Druhým možným rozšíření práce by byla případná analýza agilních metodik z jednoho úhlu pohledu detailněji. Příkladem může být pohled přes psychologii, kde by se řešilo vhodné složení týmů, hledaní vhodných členů a řešení konfliktů.

Posledním možným rozšířením by bylo vytvoření hodnotícího systému pro výběr metodik, který by bral v potaz aktuálnost některých kritérií. Mohlo by se tedy jednat i o rozšíření či aktualizaci systému METES.

Terminologický slovník

Termín	Zkratka	Význam (zdroj)
Artefakt		V této práce je artefaktem míněný někým vytvořený objekt. (autor)
Best practices		Volně přeloženo se jedná o určitou sadu nejlepších praktik, která je ověřena mnohými uživateli. (autor)
Branch		Branch označuje zduplikování a oddělení části kódu pro oddělený vývoj či zálohování. (autor)
Bug		Označuje chybu v softwaru. (autor)
Build		Jedná se o kompilovanou verzi vyvíjeného software (Wikipedia 2016)
CI	CI	Průběžná integrace. Více popsaná v kapitole
Extrémní programování	XP	Agilní metodika. Více popsaná v kapitole (autor)
Feature Driven Development	FDD	Agilní metodika. Více popsaná v kapitole (autor)
Framework		Softwarová struktura sloužící jako podpora při programování. Může obsahovat knihovny API, podporu pro návrhové vzory a doporučené postupy pro vývoj.
Man-day		Jedná se o pojem, který označuje jeden pracovní den jednoho pracovníka. Termín se využívá především u lidí najímaných externě a převážně v oboru IT. (autor)
Merge		Pojem, který slouží k označení pro spojení dvou částí kódu, kdy z počátku byl jeden společný kód, ten se rozdělil a vývoj těchto částí probíhal po nějakou dobu odděleně. (autor)
Metodika		Jedná se o označení pro komplexních postupů a návodů pro vývoj softwarové aplikace. (Kadlec 2004)
Microsoft Team Foundation Server	TFS	Jedná se o softwarový nástroj, který poskytuje několik služeb. Mezi ně patří repository, verzování, podpora procesů a metodik. (autor)
Rational Unified Process	RUP	Rigorózní metodika. Více popsaná v kapitole (autor)
Release		Jedná se o verzi softwaru, která je uvolněna. Označena může být tímto pojmem i činnost uvolnění verze. (autor)
Repository		Označuje centrální úložiště dat. (autor)
Selenium		Jedná se o software Selenium Webdriver, který je využíván pro automatizované testování uživatelského rozhraní. (autor)

Teamleader/teamlead		Jedná se o člověka, který vede tým o několika lidech. Nejedná se o klasického restriktivního manažera, ale spíše o člověka, který se snaží táhnout tým za úspěchem. (autor)
Test Driven Development	TDD	Agilní metodika. Více popsaná v kapitole (autor)
Trunk		Jedná se o označení pro hlavní kód, na kterém probíhá vývoj software. (autor)
User Acceptance Testing	UAT	Jedná se o testovací fázi projektu, ve které uživatelé systému hodnotí, zda systém odpovídá kladeným požadavkům. (Definity 2016)
Workflow		Workflow označuje tok informací v podnikovém procesu. (Management Mania 2016)
Work-in-progress	WIP	Počet aktuálně rozpracovaných položek (autor)

Seznam literatury

ALENA BUCHALCEVOVÁ a MAREK LEITL, 2006. Průzkum používání agilních metodik v ČR. In: *Objekty 2006*. Praha: PEF ČZU, s. s. 125–136. ISBN 80-213-1568-7.

ALSHAYEB, Mohammad, 2011. The Impact of Refactoring to Patterns on Software Quality Attributes. *Arabian Journal for Science and Engineering*. 11., roč. 36, č. 7, s. 1241–1251. ISSN 1319-8025, 2191-4281. DOI:10.1007/s13369-011-0111-3

ANDERSON, David J., 2010. *Kanban: successful evolutionary change for your technology business*. Sequim, WA: Blue Hole Press. ISBN 978-0-9845214-0-1.

BARRY BOEHM, 1988. A Spiral Model of Software Development and Enhancement. *Computer* [online] [cit. 2016-03-16.]. 5., roč. 21. ISSN 0018-9162. Dostupné z: <https://www.dimap.ufrn.br/~jair/ES/artigos/SpiralModelBoehm.pdf>.

BECK, Kent, 1999. *Extreme programming explained: embrace change* [online] [cit. 2016-03-20.]. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc. ISBN 0-201-61641-6. Dostupné z: <http://software2012team23.googlecode.com/git-history/5127389d21813c2bd955c53999f66cede994578b/docs/literature/Extreme_Programming_Explained_Kent_Beck_1999.pdf >.

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

COCKBURN, Alistair a Laurie WILLIAMS, 2001. The costs and benefits of pair programming. In: *Extreme programming examined*. Boston: Addison-Wesley Longman Publishing Co., Inc. ISBN 0-201-71040-4.

COHN, Mike, 2014. *Why Sprints Should Be Planned with Hours, Not Points* [online] [cit. 2016-04-25.]. Dostupné z: <https://www.scrumalliance.org/community/spotlight/mike-cohn/september-2014/why-sprints-should-be-planned-with-hours-not-point>.

CSIKSZENTMIHALYI, Mihaly, 1991. *Flow: the psychology of optimal experience*. New York: HarperPerennial. ISBN 978-0-06-092043-2.

DALIBOR PULKERT, 2014. Průzkum agilního řízení v ČR. *Etnetera* [online] [cit. 2016-04-01.]. Dostupné z: <http://www.etnetera.cz/public/1b/43/e5/52571_103079_agilni_dotaznik_report_2013_5.pdf>.

DEFINITY, 2016. *UAT* [online] [cit. 2016-05-01.]. Dostupné z: <http://www.definity.cz/cz/sekce/rozcestnik-sluzeb-vyvoj-software-na-miru-uat-13/>.

DOUCEK, Petr, Miloš MARYŠKA, Lea NEDOMOVÁ a Ondřej MATUŠÍK, 2013. *Informační management v informační společnosti*. Praha: Professional Publishing. ISBN 978-80-7431-097-3.

FOWLER, Martin a Kent BECK, 1999. *Refactoring: improving the design of existing code*. Reading, MA: Addison-Wesley. The Addison-Wesley object technology series. ISBN 978-0-201-48567-7.

FUJDIAR, Robert, 2014. *Problematika výběru agilní metodiky vývoje software* [online] [cit. 2016-03-20.]. 2014. B.m.: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Drahomír Chocholatý. Dostupné z: <https://isis.vse.cz/zp/portal_zp.pl?podrobnosti_zp=45106>.

HABR, Jaroslav a Jaromír VEPŘEK, 1986. *Systémová analýza a syntéza (zdokonalování a projektování systémů)*. 2., přeprac. vyd. Praha: Nakladatelství technické literatury.

HARTL, Pavel a Helena HARTLOVÁ, 2000. *Psychologický slovník*. Praha: Portál. ISBN 978-80-7178-303-9.

HEFNEROVÁ, Lucie, 2012. *Lean sofware development* [online] [cit. 2016-03-25.]. 2012. B.m.: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Alena Buchalcevová. Dostupné z: <https://www.vse.cz/vskp/33218_lean_software_development>.

HEFNEROVÁ, Lucie, 2015. *Kanban a možnosti jeho využití v bankovním prostředí* [online] [cit. 2016-03-28.]. 2015. B.m.: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Alena Buchalcevová. Dostupné z: <https://www.vse.cz/vskp/45047_kanban_a%C2%A0moznosti_jeho_vyuziti_v%C2%A0bankovnim_prostredi>.

HIROTAKA TAKEUCHI a IKUJIRO NONAKA, 1986. The New New Product Development Game. *Harvard Business Review* [online] [cit. 2016-04-20.]. 1., roč. Leden 1986. Dostupné z: <https://hbr.org/1986/01/the-new-new-product-development-game>.

HOUSTON, Drew, 2010. Dropbox: Startup Lessons Learned. In: [online] [cit. 2016-04-21.]. B.m. Dostupné z: <http://www.slideshare.net/gueste94e4c/dropbox-startup-lessons-learned-3836587>

CHLAPEK, Dušan, 2008. *Návrh metodického rámce řízení a koordinace projektů IS/ICT* [online] [cit. 2016-03-15.]. 2008. B.m.: Disertační práce, VŠE. Dostupné z: <https://www.vse.cz/vskp/eid/27160>.

JABLONSKÝ, Josef, 2014. *SANNA 2014 - MS Excel based system for multicriteria evaluation of alternatives* [online] [cit. 2016-05-01.]. Dostupné z: <http://nb.vse.cz/~jablon/sanna.htm>.

KADLEC, Václav, 2004. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press. ISBN 978-80-251-0342-5.

KANNANGARA, S.H. a W.M.J.I. WIJAYANAKE, 2015. An Empirical Evaluation of Impact of Refactoring on Internal and External Measures of Code Quality. *International Journal of Software Engineering & Applications*. 31. 1., roč. 6, č. 1, s. 51–67. ISSN 09762221, 09759018. DOI:10.5121/ijsea.2015.6105

K. BECK, M. BEEDLE, A. VAN BENNEKUM, ET AL., 2001. *The Agile Manifesto* [online] [cit. 2016-03-10.]. Dostupné z: <http://www.agilemanifesto.org/>

KEN SCHWABER a JEFF SUTHERLAND, 1995. SCRUM Development Process. In: *OOPSLA* [online] [cit. 2016-04-02.]. Austin, Texas (US): Ken Schwaber. Dostupné z: <http://www.jeffsutherland.org/oopsla/schwapub.pdf>

KNIBERG, Henrik a Mattias SKARIN, 2010. *Kanban and Scrum: making the most of both*. s. l.: C4Media. InfoQ enterprise software development series. ISBN 978-0-557-13832-6.

KRILL, Paul, 2011. Why the Time is Now for Continuous Integration in App Development. *InfoWorld.Com* [online] [cit. 2016-04-01.]. 7. 7., roč. 2011. Dostupné z: <http://search.proquest.com/docview/875325501?accountid=17203>

LAYMAN, Lucas, Laurie WILLIAMS a Lynn CUNNINGHAM, 2004. *Exploring Extreme Programming in Context: An Industrial Case Study* [online] [cit. 2016-04-15.]. Los Alamitos, Calif.: IEEE Computer Society. ISBN 978-0-7695-2248-7. Dostupné z: <ftp://atenas.cpd.ufv.br/dpi/mestrado/XP/Exploring%20extreme%20programming%20in%20context-%20an%20industrial%20case%20study.pdf>

MALAN JOUBERT, 2016. *How Instagram does 40+ daily deployments* [online] [cit. 2016-04-30.]. Dostupné z: <https://www.offerzen.com/blog/how-instagram-does-40-plus-daily-deployments>

MANAGEMENT MANIA, 2016. *Workflow* [online] [cit. 2016-04-28.]. Dostupné z: <https://managementmania.com/cs/workflow>

MAREK, Jan, 2013. *Metodika SCRUM a její programová podpora* [online] [cit. 2016-03-20.]. 2013. B.m.: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Jarmila Pavlíčková. Dostupné z: <https://www.vse.cz/vskp/39705_metodika_scrum_a%C2%A0jeji_programova_podpora>

MARTIN FOWLER, 2006. *Continuous Integration* [online] [cit. 2016-04-18.]. Dostupné z: <http://www.martinfowler.com/articles/continuousIntegration.html>

MCCONNELL, Steve, 2005. *Dokonalý kód: umění programování a techniky tvorby software*. Brno: Computer Press. ISBN 978-80-251-0849-9.

M.C. PARTNERS & ASSOCIATES, 2015. *Lean/Kanban Case-study* [online] [cit. 2016-05-02.]. 2015. Dostupné z: <http://www.mcpa.biz/wp-content/uploads/2012/12/Kanban-Case-Study-FINAL.pdf>

MÜHLFEIT, Jan, 2016. *Jan Mühlfeit - Pro Billa Gatese bych pracoval i za stravenky | LIDÉ Z PRAXE* [online] [cit. 2016-05-03.]. 2016. MÜHLFEIT, Jan. Youtube. Dostupné z: <https://www.youtube.com/watch?v=ttSX0qkWBd0>

NORRMALM, Thomas, 2011. *Achieving Lean Software Development* [online] [cit. 2016-04-03.]. 2011. B.m.: Uppsala Universitet. Dostupné z: <http://uu.diva-portal.org/smash/get/diva2:400627/FULLTEXT01>

OHNO, Taiichi, 1978. *Tovota Production System: Beyond Large-Scale Production* [online] [cit. 2016-04-28.]. 1978. Dostupné z: <http://www.kellogg.northwestern.edu/course/opns430/modules/lean_operations/ohno-tps.pdf>

PELIKÁN, Jan, Vladislav CHÝNA, VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE a FAKULTA INFORMATIKY A STATISTIKY, 2011. *Kvantitativní management*. Praha: Oeconomica. ISBN 978-80-245-1830-5.

PILGRIM, Mark, 2010. *Ponořme se do Python(u) 3*. Praha: Cz.Nic. ISBN 978-80-904248-2-1.

POPELKA, Vladimír, 2009. *Srovnávací analýza metodik vývoje software* [online] [cit. 2016-03-28.]. 2009. B.m.: Vysoká škola ekonomická v Praze, Fakulta informatiky a statistiky. Vedoucí práce Helena Kučerová. Dostupné z: <http://info.sks.cz/www/zavprace/soubory/54300.pdf>

POPPENDIECK, Mary a Tom POPPENDIECK, 2003. *Lean software development: an agile toolkit*. Nachdr. Boston: Addison-Wesley. The agile software development series. ISBN 978-0-321-15078-3.

PROJECT MANAGEMENT INSTITUTE, ed., 2013. *A guide to the project management body of knowledge (PMBOK guide)*. Fifth edition. Newtown Square, Pennsylvania: Project Management Institute, Inc. ISBN 978-1-935589-67-9.

SCRUMGUIDES.ORG, 2016. The History of Scrum. *SCRUM GUIDES* [online] [cit. 2016-03-15.]. Dostupné z: <http://www.scrumguides.org/history.html>

SCHWABER, Ken, 2007. *The enterprise and Scrum*. Redmond, Wash: Microsoft Press. Best practices. ISBN 978-0-7356-2337-8.

SCHWABER, Ken a Mike BEEDLE, 2001. *Agile software development with Scrum*. Upper Saddle River, NJ: Prentice Hall. Series in agile software development. ISBN 978-0-13-067634-4.

SCHWABER, Ken a Jeff SUTHERLAND, 2013. *™Průvodce Scrumem* [online] [cit. 2016-03-03.]. srpen 2013. B.m.: Scrum.Org and ScrumInc. Dostupné z: <http://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-CS.pdf#zoom=100>

STELLMAN, Andrew a Jennifer GREENE, 2015. *Learning Agile: [understanding Scrum, XP, Lean, and Kanban]*. 1st ed. Beijing: O'Reilly. ISBN 978-1-4493-3192-4.

TŘÍSKOVÁ, Lucie, 2015. *Testování webových aplikací s využitím nástroje Selenium Webdriver* [online] [cit. 2016-04-05.]. 2015. B.m.: Vysoká škola ekonomická v Praze. Dostupné

z: <https://www.vse.cz/vskp/45087_testovani_webovych_aplikaci_s%C2%A0vyuzitim_nastroje_selenium_webdriver>

VERSIONONE, INC., 2006. *Survey: The State of Agile Development* [online] [cit. 2016-04-15.]. Dostupné z: <https://www.versionone.com/pdf/2006-state-of-agile-survey.pdf>

VERSIONONE, INC., 2008. *3rd Annual Survey: 2008, The State of Agile Development* [online] [cit. 2016-04-15.]. Dostupné z: <https://www.versionone.com/pdf/StateOfAgileDevResults2008.pdf>

VERSIONONE, INC., 2009. *State of Agile Survey: The State of Agile Development, 4th Annual* [online] [cit. 2016-04-15.]. Dostupné z: <https://www.versionone.com/pdf/2009_State_of_Agile_Development_Survey_Results.pdf>

VERSIONONE, INC., 2014. *8th Annual State of Agile™ Survey* [online] [cit. 2016-04-15.]. Dostupné z: <https://www.versionone.com/pdf/2013-state-of-agile-survey.pdf>

VERSIONONE, INC., 2015. *The 9th annual State of Agile™Survey* [online] [cit. 2016-04-15.]. Dostupné z: <https://www.versionone.com/pdf/state-of-agile-development-survey-ninth.pdf>

VERSIONONE, INC., 2016. *The 10th annual State of Agile™Survey* [online] [cit. 2016-04-15.]. Dostupné z: <https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf>

VODÁČEK, Leo a Oľga VODÁČKOVÁ, 2009. *Synergie v moderním managementu*. Praha: Management Press, s.r.o. ISBN 978-80-7261-190-4.

WALLACE, Doug, Isobel RAGGETT a Joel AUFGANG, 2003. *Extreme programming for Web projects*. Boston, MA: Addison-Wesley. ISBN 978-0-201-79427-4.

WIKIPEDIA, 2016. *Průběžná integrace* [online] [cit. 2016-05-04.]. Dostupné z: <https://cs.wikipedia.org/wiki/Pr%C5%AFb%C4%9B%C5%BEn%C3%A1_integrace>

WILLIAMS, L., R.R. KESSLER, W. CUNNINGHAM a R. JEFFRIES, 2000. Strengthening the case for pair programming. *IEEE Software* [online] [cit. 2016-04-10.]. 8., roč. 17, č. 4, s. 19–25. ISSN 07407459. DOI:10.1109/52.854064

WINSTON W. ROYCE, 1970. Managing the Development of Large Software Systems. In: *Technical Papers of Western Electronic Show and Convention* [online] [cit. 2016-04-25.]. Dostupné z: <http://leadinganswers.typepad.com/leading_answers/files/original_waterfall_paper_winston_royce.pdf>

WRAY, S., 2010. How Pair Programming Really Works. *IEEE Software* [online] [cit. 2016-04-18.]. 1., roč. 27, č. 1, s. 50–55. ISSN 0740-7459. DOI:10.1109/MS.2009.199

ZUZANA ŠTOCHOVÁ a EDUARD KUNCE, 2014. *Agilní metody řízení projektů*. Brno: Computer Press. ISBN 978-80-251-4194-6.

Seznam obrázků a tabulek

Seznam obrázků

Obrázek 1: Vodopádový model (Zdroj: Winston W. Royce, 1970, str. 329) 5
Obrázek 2: Schéma vodopádového modelu (Zdroj: Barry Boehm, 1988, str. 64) 7
Obrázek 3: Tradiční versus agilní přístup (Zdroj: autor, dle [Kadlec, 2004, str. 119]) 16
Obrázek 4: Ukázka Product backlog (Zdroj: autor) 22
Obrázek 5: Ukázka User story a její vlastnosti (Zdroj: autor) 23
Obrázek 6: Ukázka Tasku (Zdroj: autor) 25
Obrázek 7: Ukázka Super User story (Zdroj: autor) 26
Obrázek 8: Ukázka Epic (Zdroj: autor) 27
Obrázek 9: Schéma metodiky SCRUM (Zdroj: autor) 28
Obrázek 10: Ukázka best practices při Retrospektivě (Zdroj: autor, dle [Zuzana Štochové a Eduard Kunce, 2014, str. 73]) 31
Obrázek 11: Ukázka Scrum board (Zdroj: autor) 33
Obrázek 12: Burndown chart (Zdroj: Kniberg a Skarin, 2010, str. 38) 34
Obrázek 13: Čtyři proměnné v metodice XP (Zdroj: autor) 36
Obrázek 14: Prolínání principů Lean, XP a SCRUM (Zdroj: Autor, dle [Stellman a Greene, 2015, str. 272]) 48
Obrázek 15: Kanban board (Zdroj: autor) 51
Obrázek 16: Kanban board se swimline (Zdroj: autor) 51
Obrázek 17: Cumulative flow diagram (Zdroj: Kniberg a Skarin, 2010, str. 39) 52
Obrázek 18: Myšlenková mapa charakteristik projektu (Zdroj: autor) 55

Seznam grafů

Graf 1: Vývoj využívání agilních metodik (Zdroj: autor, dle VersionOne) 14
Graf 2: Vývoj využívaní agilních praktik (Zdroj: autor, dle VersionOne) 18
Graf 4: Paprskový graf ukazující Kritéria Proces (Zdroj: autor) 61
Graf 5: Paprskový graf ukazující Kritéria Podpora (Zdroj: autor) 61
Graf 6: Paprskový graf ukazující Kritéria Produkt (Zdroj: autor) 62
Graf 7: Paprskový graf ukazující Kritéria Lidé (Zdroj: autor) 63

Seznam tabulek

Tabulka 1: Sedm typů plýtvání dle Lean manufacturing (Zdroj: autor, dle [Ohno, 1978]) . 45
Tabulka 2: Sedm typů plýtvání v Lean software development (Zdroj: autor, dle (Poppendieck a Poppendieck, 2003)) 45
Tabulka 3: Ohodnocení řešení charakteristik projektu pomocí technik a praktik jednotlivých metodik (Zdroj: autor) 56
Tabulka 4: Přidělené váhy charakteristik od jednotlivých členů týmu (Zdroj: autor) 58
Tabulka 5: Výpočet vícekriteriálního rozhodování TOPSIS (Zdroj: autor) 58
Tabulka 6: Input data (Zdroj: autor, dle SANNA 2014) 75
Tabulka 7: Modified input data (Zdroj: autor, dle SANNA 2014) 75
Tabulka 8: Normalised criterion matrix R (Zdroj: autor, dle SANNA 2014) 75
Tabulka 9: Weighted criterion matrix W (Zdroj: autor, dle SANNA 2014) 76

Tabulka 10: Kritéria skupiny Proces ze systému METES (Zdroj: autor)..... 77
Tabulka 11: Kritéria skupiny Podpora ze systému METES (Zdroj: autor)..... 77
Tabulka 12: Kritéria skupiny Lidé ze systému METES (Zdroj: autor)..... 77
Tabulka 13: Kritéria skupiny Produkt ze systému METES (Zdroj: autor) 77

Příloha A: Výpočty vícekriteriálního rozhodování TOPSIS

Všechna data v této příloze jsou čerpána z výpočtů v aplikace Excel za pomocí doplňku SANNA 2014. Cílem této práce není rozbor vícekriteriálního rozhodování, takže postup není detailněji rozepisován.

Pozn.: v tabulka 9 bylo zmenšeno písmo, aby se vešla do práce.

Tabulka 6: Input data (Zdroj: autor, dle SANNA 2014)

	MIN	MIN	MIN	MIN	MIN	MIN	MIN	MIN	MIN
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8	CH9
SCRUM	5	7	1	1	5	1	1	1	3
XP	7	7	1	5	1	9	5	1	1
Lean	7	9	3	5	1	7	9	1	3
Kanban	5	5	3	1	7	5	7	5	5
Váhy	0,14000	0,09750	0,06250	0,11250	0,13500	0,13500	0,03000	0,18750	0,10000

Tabulka 7: Modified input data (Zdroj: autor, dle SANNA 2014)

	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8	CH9
SCRUM	2	2	2	4	2	8	8	4	2
XP	0	2	2	0	6	0	4	4	4
Lean	0	0	0	0	6	2	0	4	2
Kanban	2	4	0	4	0	4	2	0	0
Váhy	0,14000	0,09750	0,06250	0,11250	0,13500	0,13500	0,03000	0,18750	0,10000

Tabulka 8: Normalised criterion matrix R (Zdroj: autor, dle SANNA 2014)

	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8	CH9
SCRUM	0,70711	0,40825	0,70711	0,70711	0,22942	0,87287	0,87287	0,57735	0,40825
XP	0,00000	0,40825	0,70711	0,00000	0,68825	0,00000	0,43644	0,57735	0,81650
Lean	0,00000	0,00000	0,00000	0,00000	0,68825	0,21822	0,00000	0,57735	0,40825
Kanban	0,70711	0,81650	0,00000	0,70711	0,00000	0,43644	0,21822	0,00000	0,00000
Váhy	0,14000	0,09750	0,06250	0,11250	0,13500	0,13500	0,03000	0,18750	0,10000

Tabulka 9: Weighted criterion matrix W (Zdroj: autor, dle SANNA 2014)

	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX	MAX			
	CH1	CH2	CH3	CH4	CH5	CH6	CH7	CH8	CH9	di+	di-	ci
SCRUM	0,09899	0,03980	0,04419	0,07955	0,03097	0,11784	0,02619	0,10825	0,04082	0,08419	0,22041	**0,72361**
XP	0,00000	0,03980	0,04419	0,00000	0,09291	0,00000	0,01309	0,10825	0,08165	0,17824	0,17529	**0,49583**
Lean	0,00000	0,00000	0,00000	0,00000	0,09291	0,02946	0,00000	0,10825	0,04082	0,18596	0,15128	**0,44858**
Kanban	0,09899	0,07961	0,00000	0,07955	0,00000	0,05892	0,00655	0,00000	0,00000	0,18119	0,16118	**0,47079**
Váhy	0,14000	0,09750	0,06250	0,11250	0,13500	0,13500	0,03000	0,18750	0,10000			
Ideal	**0,09899**	**0,07961**	**0,04419**	**0,07955**	**0,09291**	**0,11784**	**0,02619**	**0,10825**	**0,08165**			
Basal	**0,00000**	**0,00000**	**0,00000**	**0,00000**	**0,00000**	**0,00000**	**0,00000**	**0,00000**	**0,00000**			

Příloha B: Výpočty systému METES

Tabulka 10: Kritéria skupiny Proces ze systému METES (Zdroj: autor)

	SCRUM max	SCRUM min	XP max	XP min	PROJEKT
Řízení kvality	2	0	2	0	2
Metriky	2	0	2	0	1
Rozsah	2	0	1	0	3
Dokumenty	1	0	1	0	2
Model životního cyklu	5	0	5	0	5
Role	2	0	1	0	1
Podrobnost popisu procesu	0	0	0	0	0

Tabulka 11: Kritéria skupiny Podpora ze systému METES (Zdroj: autor)

	SCRUM max	SCRUM min	XP max	XP min	PROJEKT
Celistvost zdrojů	2	0	2	0	4
Lokalizace	0	0	0	0	0
Dostupnost	1	0	1	0	4
Školení a certifikace	4	0	5	0	5
Výuka na vysokých školách	4	0	5	0	4
Přizpůsobení metodiky	4	0	3	0	4
Podpora zavedení metodiky	2	0	2	0	1
Podpora metodiky soft. nástroji	4	0	4	0	2

Tabulka 12: Kritéria skupiny Lidé ze systému METES (Zdroj: autor)

	SCRUM max	SCRUM min	XP max	XP min	PROJEKT
Rozmístění	4	0	1	0	0
Velikost týmu	4	1	1	0	0
Dostupnost uživatelů	0	0	0	0	1
Motivace členů týmu	1	0	1	0	0
Kvalifikace členů týmu	1	0	1	0	0
Zkušenost manažera projektu	2	0	2	0	0

Tabulka 13: Kritéria skupiny Produkt ze systému METES (Zdroj: autor)

	SCRUM max	SCRUM min	XP max	XP min	PROJEKT
Důležitost produktu	3	0	3	0	2
Velikost řešení	5	0	3	0	2
Znovupoužitelnost	1	0	1	0	3
Stálost požadavků	0	0	0	0	2
Délka projektu	3	1	2	0	5