TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

Řízení vývoje softwaru

Diplomová práce

Studijní program:	N6209 Systémové inženýrství a informatika
Studijní obor:	Manažerská informatika
Autor práce:	**Bc. Vojtěch Vazda**
Vedoucí práce:	doc. Ing. Klára Antlová, Ph.D. Katedra informatiky

Liberec 2020

TECHNICKÁ UNIVERZITA V LIBERCI
Ekonomická fakulta

Zadání diplomové práce

Řízení vývoje softwaru

Jméno a příjmení: **Bc. Vojtěch Vazda**
Osobní číslo: E17000269
Studijní program: N6209 Systémové inženýrství a informatika
Studijní obor: Manažerská informatika
Zadávající katedra: Katedra informatiky
Akademický rok: **2019/2020**

Zásady pro vypracování:

1. Typy nástrojů a význam nástrojů pro řízení vývoje SW
2. Jednotlivé etapy vývoje
3. Návrh zefektivnění vývojového cyklu
4. Zhodnocení, zpětná vazba, případný návrh dalšího řešení

Rozsah grafických prací:
Rozsah pracovní zprávy: 65 normostran
Forma zpracování práce: tištěná/elektronická
Jazyk práce: Čeština

Seznam odborné literatury:

ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ, 2013. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press. ISBN 978-80-251-3816-8.
ROUDENSKÝ, Petr, 2018. *Kvalita softwaru: teorie a praxe.* Upravené a rozšířené 2. vydání. Prostějov: Computer Media. ISBN 978-80-7402-322-4.
ŠOCHOVÁ, Zuzana a Eduard KUNCE, 2019. *Agilní metody řízení projektů: průvodce testováním.* 2. vydání. Brno: Computer Press. ISBN 978-80-251-4961-4.
SOMMERVILLE, Ian a Eduard KUNCE, 2013. *Softwarové inženýrství: průvodce testováním*. 2. vydání. Brno: Computer Press. ISBN 978-80-251-3826-7.
HARNED, David, 2018. H*ands-On Agile Software Development with JIRA: Design and manage software projects using the Agile methodology.* Birmingham: Packt. ISBN 978-1-78953-213-5.
PROQUEST. 2019. *Databáze článků ProQuest* [online]. Ann Arbor, MI, USA: ProQuest. [cit. 2019-10-02]. Dostupné z: http://knihovna.tul.cz/
E-books: https://knihovna-opac.tul.cz/
Konzultant Ing. Jan Jonášek

Vedoucí práce: doc. Ing. Klára Antlová, Ph.D.
Katedra informatiky

Datum zadání práce: 31. října 2019
Předpokládaný termín odevzdání: 31. srpna 2021

L.S.

prof. Ing. Miroslav Žižka, Ph.D.
děkan

doc. Ing. Klára Antlová, Ph.D.
vedoucí katedry

V Liberci dne 31. října 2019

Prohlášení

Prohlašuji, že svou diplomovou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé diplomové práce a konzultantem.

Jsem si vědom toho, že na mou diplomovou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé diplomové práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li diplomovou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má diplomová práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

20. dubna 2020

Bc. Vojtěch Vazda

Anotace

Diplomová práce je zaměřena na problematiku projektového řízení při vývoji softwaru pomocí agilní metodiky. Cílem diplomové práce je porozumění pojmu projektové řízení, stanovit rozdíl mezi tradiční metodikou projektového řízení a agilní metodikou projektového řízení a tuto agilní metodiku dále charakterizovat. Charakteristika probíhá ve vymezení jednotlivých pojmů, nástrojů a způsobů, které se v agilní metodice objevují. To je následně použito k analýze současného stavu projektového řízení v konkrétní společnosti, která se zabývá vývojem softwaru pro odpadové hospodářství v České republice a Slovensku. Na základě této analýzy je provedena optimalizace pro zlepšení projektového řízení dle navržených bodů. Na konci diplomové práce je provedeno zhodnocení optimalizace projektového řízení a další námět pro budoucí optimalizační procesy.

Klíčová slova

projektové řízení, agilní metodika, Scrum, Product Owner, Scrum Master, Product Backlog, Sprint, Sprint Backlog

Annotation

The diploma thesis is focused on project management in software development using agile methodology. The aim of this thesis is to understand the concept of project management, to determine the difference between traditional project management methodology and agile project management methodology and further characterize this agile methodology. Characteristics proceed in the definition of individual terms, tools and ways that appear in agile methodology. This is then used to analyze the current state of project management in a particular company that develops software for waste management in the Czech Republic and Slovakia. Based on this analysis, optimization is performed to improve project management according to the proposed points. At the end of the thesis there is an evaluation of project management optimization and another theme for future optimization processes.

Key Words

project management, agile methodology, Scrum, Product Owner, Scrum Master, Product Backlog, Sprint, Sprint Backlog

Poděkování

Velmi bych chtěl poděkovat doc. Ing. Kláře Antlové Ph.D. za odbornou pomoc a veškeré rady, které mi poskytnula při tvorbě této diplomové práce. Dále bych chtěl poděkovat Ing. Janovi Jonáškovi za veškerých odborné konzultace a interní informace, které mi poskytl pro vypracování této diplomové práce. V neposlední řadě chci poděkovat celé své rodině za poskytovanou podporu při celém průběhu mého studia na Ekonomické fakultě Technické Univerzity v Liberci.

Děkuji!

Seznam obrázků

Seznam zkratek

BA – Business Analýza

CASE – Computer Aided Software Engineering

CI – Centrum Informací

ESVP – Druh retrospektivy (Explorer, Shopper, Vacationer a Prisoner)

GUI – Graphic User Interface

ID – Identifikátor

ISB – Information System Branch

IT – Informační Technologie

PC – Personal Computer

ROI – Return On Investment

SDLC – Software Development Life Cycle

SK – Slovensko

TDD – Test Driven Development

VPN – Virtual Private Network

WIN – Windows (operační systém od společnosti Microsoft Corporation)

Seznam tabulek

Úvod

Současný svět se stává stále složitější a rychlejší. Zvyšuje se počet konkurentů a tím se i zvyšuje úsilí o splnění veškerých zákaznických požadavků. Na zvyšující se náročnost musí společnosti zareagovat vhodnými nástroji. Veškeré požadavky musí být vypracovány s co nejvyšší kvalitou a předány v požadovaném termínu. Zároveň musí být dodány s patřičnou technickou podporou a uživatelskou dokumentací, která zákazníkovi dokáže usnadnit práci a vyřešit případné problémy, které se mohou objevit. Toto uspokojení dokáže prohloubit vztahy mezi zákazníkem a společností, která vytváří software.

Pokud však chce společnost být takto úspěšná, musí stavět na pevných základech, které musí neustále budovat a upevňovat ve svém interním prostředí. Jedním ze stavebních kamenů je projektové řízení při vývoji softwaru, čeho se týká taky tato práce.

Cílem této diplomové práce je zanalyzovat současnou situaci projektového řízení při vývoji softwaru v konkrétní společnosti. Konkrétní společnost se zabývá vývojem softwaru pro podporu a evidenci odpadového hospodářství. Na základě této analýzy bude provedeno kritické zhodnocení a navržení optimalizace pomocí několika bodů, které budou mít za výsledek zlepšení stávajícího stavu projektového řízení ve firmě. Společnost se zaměřuje na projektové řízení pomocí agilních metod, proto budou determinovány rozdíly mezi tradiční metodikou projektového řízení a agilní metodikou projektového řízení. Následně se práce zaměřuje na agilní metodiku projektového řízení. V práci budou popsány jednotlivé role projektového řízení, nástroje a způsob, jak vést projektové řízení vývoje softwaru pomocí agilní metodiky.

Následně bude provedena analýza využívání projektového řízení v organizaci, kde bude zobrazeno, jaké využívá cykly projektového řízení a jakým způsobem celou tuto činnost provádí. Na základě této analýzy bude provedeno vyhodnocení a zároveň s tímto vyhodnocením budou navrženy body pro optimalizaci celého procesu projektového řízení. Tyto návrhy budou prezentovány společnosti, ve které se optimalizace provádí. Na základě zpětné vazby a celkového výstupu od společnosti bude celá optimalizace vyhodnocena. V závěru diplomové práce se provedou další návrhy a náměty projektového řízení, kterých

by se společnost mohla v budoucnosti věnovat, aby dále posílila svoji efektivitu a produktivitu.

1 Typy nástrojů a význam nástrojů pro řízení vývoje softwaru

V současnosti se v obor IT dostal do stavu, kdy je třeba mít vše jasně, věcně a přesně definované. Nepřesně nebo nedostatečně definované pojmy nebo termíny, chybějící shoda názorů všech zúčastněných, mohou vést k nepřesnostem nebo špatnému pochopení celkové myšlenky a koncepce produktu, a následně ke špatnému zpracování produktu, dodatečným nákladům, časové prodlevě, stresu z okolního tlaku a podobně. I tomto tématu se zabývá článek (2016: Digitální revoluce přinese tlak na zrychlení procesů a kyberbezpečnost, 2016). První kapitola diplomové práce slouží k definování důležitých pojmů projektového řízení.

Důležité informace o tom, jak by společnosti měly používat podpůrný software pro zvýšení efektivnosti práce, poskytuje kniha „*Podniková informatika: počítačové aplikace v podnikové a mezipodnikové praxi*. 3., aktualizované vydání“ od autorů Libor Gála, Jan Pour a Zuzana Šedivá.

Dobrý úvod do procesního řízení vývoje softwaru, tradiční a agilní metodiky lze získat z knihy „*Scrum: průvodce agilním vývojem softwaru*.“ od autora Josef Myslín. Kniha popisuje charakteristiku tradiční metodiky, konkrétní praktické druhy tradičních metodik, charakteristiku agilní metodiky a konkrétní praktické druhy agilních metodik.

Dalším cenným zdrojem informací ze světa agilního procesního řízení softwaru poskytuje kniha „*Projektový management: komplexně, prakticky a podle světových standardů“* od autora Jana Doležala. Kniha poskytuje informace o úvodu projektového managementu, důležitých činnostech před začátkem projektu, řízení projektu a agilní metodě řízení projektu.

Velmi detailně je vysvětleno agilní řízení projektu v knize „*Agilní metody řízení projektů*.“ od autorů – Zuzana Šochová a Eduard Kunce. Kniha z praktického pohledu vymezuje spousta pojmů a procesů z agilního prostředí řízení projektů. Jsou popisuje role projektového týmu, používané nástroje a jejich důležitost. Informace jsou doplněné poznatky z reálného světa a zkušeností autorů.

Zajímavou zahraniční publikaci uvádí autor Gunther Verheyen v díle „*Scrum – A Pocket Guide - 2nd edition:: A Smart Travel Companion*.“. V knize jsou zajímavě popsány informace ohledně agilního paradigmatu a o Scrum metodě.

Kniha „*Hands-On Agile Software Development with JIRA: Design and manage software projects using the Agile methodology*.“, kde je autorem David Harned, pojednává o agilním řízení projektu z pohledu podpůrného systému JIRA. V knize jsou informace o jednotlivých principech a důležitých prvcích systému JIRA. Čtenář se zde může dozvědět, jak řídit práci pomocí Sprintů, Epiců a jednotlivých úkolů, jak tvořit reporty a podobné uživatelské výstupy, verzování aplikace a využití pokročilého JQL vyhledávače.

1.1 Řízení vývoje softwaru

Aby bylo možné přesně vymezit, co to vlastně řízení softwaru znamená, tak je vhodné rozdělit si celý tento pojem do jednotlivých slov a přesně určit a vymezit jejich význam.

1.1.1 Řízení

Řízení je spojeno s několika pojmy zabývající se reakcemi, celkovou interakcí a odezvou, jak uvádí (Gála, 2015, s. 15).

Přesto „řízení“ má několik významů a definicí. Jak uvádí (Gála, 2015, s. 16) *„ ...*

- *Řízením rozumíme především vztah mezi řídicím subjektem a řízeným objektem, přičemž jak subjekt řízení, tak objekt řízení jsou zobrazovány jako systém a mají vlastnosti systému.*
- *Řízení, jako svůj hlavní smysl a cíl, vytváří obraz příštího stavu řízeného objektu.*
- *Řízení rovněž zajišťuje, aby vytvořený obraz byl ve vývoji řízeného objektu uplatněn.*
- *Řízení obsahuje kontrolu o tom, zda a jak byl záměr řízení skutečně realizován, přičemž tato kontrola je současně vstupem do dalšího cyklu řízení.*
- *Řízení má systémové vlastnosti (soudržnost částí v celku, schopnost jejich spolupráce, schopnost interakce s okolím, schopnost dynamické adaptability a*

směřování vývoje celku k určitému cíli), a proto má tendenci předcházející znaky realizovat s maximální efektivností…“

Z citovaného textu lze usoudit, že pojem „řízení“ je opravdu komplexní a obsahuje významově několik prvků. Zaobírá vše od subjektu, který provádí řízení a činnost, která je třeba řídit. V případě této diplomové práce bude pracováno s pojmem „řízení“ v oblasti softwaru, řízení jeho vývoje, koordinace jednotlivých členů, kteří se řízením přímo starají o jednotlivé činnosti (fáze), nebo členy, kteří jsou důležitým článkem ve vývojovém procesu softwaru.

1.1.2 Vývoj

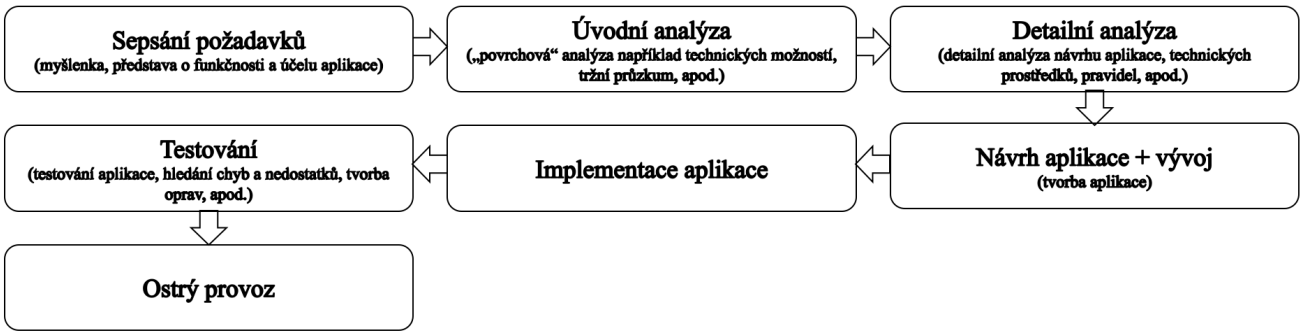
V celkovém pojetí „vývoj softwaru“ znamená velice komplexní pojem. Pro vývoj softwaru se využívají nástroje. Mezi ně se řadí překladače a editory. Spolu se skládají do takzvaných sad a společně tvoří integrovaný komplex na tvorbu aplikace. Tento integrovaný komplex se dále řadí mezi aplikační software. Díky tomuto vývojář získá možnost přistoupit k editoru, aby mohl tvořit kód.

Vývojáři získávají díky tomuto komplexu mnoho výhod. Mezi hlavní výhodu patří to, že vývojáři je umožněn hladký přesun mezi editorem a nástrojem na ladění. Dalšími výhodami jsou i zjednodušené testování a jednodušší přístup k jednotlivým prvkům (Brookshear, 2013, s. 281).

Aby výše uvedeným řádkům dokázal porozumět i člověk, který není obeznámen s problematikou tvorby softwaru, tak bude snaha o lehce lidštější přístup. K tomuto bych použil přirovnání například ke stavbě domu. Pokud je snaha postavit dům, tak musí existovat přesné zadání, projekt, schvalovací proces, stavba domu, kontrola a předání výsledného produktu. Velmi podobné to je i u vývoje softwaru. K tomu, aby mohl být vyvinut nějaký software, tak je prvním bodem rozhodně myšlenka. Pokud zadavatel, manažer nebo analytik nemá myšlenku nebo nápad na funkci aplikace a jejímu přesnému účelu, kterému bude sloužit, tak dlouhodobý vývoj prakticky není možné uskutečnit.

Pokud existuje myšlenka, představa o funkčnosti nebo účelu aplikace, tak je možné sepsat základní body. Jedná se například o prostředí aplikace, základní funkce, úchově dat. Jedná se o úvodní, a ne do hloubky vytvořenou ranou analýzu. Po tomto jednoduchém sepsání úvodních bodů začne probíhat detailní analýza. V této analýze se řeší již hlubší funkčnost aplikace, a proto by se v této části mělo dbát na pečlivost a promyšlenost. Tato část bude podrobněji popsána v další kapitole této práce. Po této fázi probíhá návrh programu, jeho vývoj, kde dostávají prostor převážně vývojáři se svojí činností, kdy tvoří kód aplikace. Po této fázi dochází k implementaci neboli zavedení změn, úprav, či nových věcí do vnitřní části aplikace. Po této fázi se musí vytvořená nebo upravená aplikace řádně otestovat. Stále zde platí, že nikdy nic není dovedeno do úplné dokonalosti a tam kde figuruje lidský faktor, tak to platí dvojnásob. Ladí se tedy aplikace, kdy probíhají různé testovací scénáře k nalezení chyb a nedostatků. Po úspěšném odladění aplikace nastává ostrý provoz. Ostrý provoz prakticky znamená stav, kdy je aplikace propuštěna k používání mezi uživatele.

Celý tento proces, který tvoří jednotlivé kroky, se nazývá životní cyklus softwaru.



Obrázek 1 - životní cyklus aplikace
(zdroj: vlastní tvorba)

1.1.3 Software

Software je programové vybavení počítače. Jedná se přesněji tedy o kterýkoliv program, který je možné mít v počítači a spustit jej. K jednodušší definice není třeba rozsáhle popisovat rozdíly mezi systémovým a aplikačním softwarem. Software je vše, co je možné spustit a s čím je možné pracovat na obrazovce počítače, notebooku, tabletu, mobilu a jiných

elektronických zařízeních. Pan Myslín to dokonce popisuje jako duši, která dává elektronickému zařízení nějaký smysl. Bez tohoto by to byl jen kus hardwaru, který by ale nijak nepracoval. Se softwarem to je však jinak. Software udává směr tomuto hardwaru na základě uživatelského pokynu. Software má sloužit uživatelům, jeho potřebám a má dopomoci k dosažení požadovaných uživatelských cílů (Myslín, 2016, s.10).

I když se jedná o základní termíny, které většina lidí zná a dokáže si představit jejich význam, tak i přesto si myslím, že tato část zde má své opodstatnění. Slouží k přesnému definování významu termínů.

1.2 Typy nástrojů pro řízení vývoje softwaru

Tato část práce popisuje jednotlivé nástroje neboli metodiky pro řízení vývoje softwaru. Metodiky se dají rozdělit do dvou hlavních skupin, a to skupina tradičních metodik a skupina agilních metodik.

1.2.1 Tradiční metodika

O tradiční metodikách se hovoří převážně z důvodu, aby se odlišily od současných modernějších metodik, které spadají do agilní skupiny nástrojů a postupů. To však ale neznamená, že se tradiční metodiky v současnosti už neuplatňují nebo nepoužívají.

V tradičních metodikách je prosazován postup, kdy se veškeré procesy seskupí. Je to z důvodu, aby vznikalo co nejméně nepřesností a neurčitostí. Díky tomu je pak možné lépe určovat jednotlivé termíny a požadavky od zákazníků. Metoda je charakterizována přesným stanovením rolí. Pracovníci mají určené přesně zaměření, a to jim dává prostor k realizaci. V tomto probíhá jejich pracovní činnost, která má pevnou hranici. Jakmile se dojde na hranici svého prostoru, tak zde pracovníci končí. Jeden konkrétní pracovník může působit ve více rolích, ale musí se dát pozor na přetížení zdroje (pracovníka). Mimo svůj realizační prostor se dále ve většině případů příliš nezasazují. To se může jevit jako zásadní nevýhoda, jelikož je jednotlivých procesních rolí větší množství a tím může být požadováno větší množství zapojených pracovníků zapojených do procesu vývoje softwaru. Příklad obecných

rolí může být například databázový architekt, analytik, vývojář, tester, grafik, projektový manažer, specialista na databázi. Další nevýhodou se jeví fakt, že aby mohl být projekt úspěšně splněn, tak je nutné, aby spolu všichni tito pracovníci spolupracovali a komunikovali. A je známo, že čím více lidí spolu komunikuje a řeší společný problém, tak tím více se jednotlivé procesy řešení prodlužují a i komplikují. Výhodou je však to, že jakmile má každý pracovník svůj prostor a angažuje se pouze jen v přiděleném prostoru, tak se pracovník může zaměřit na jednotlivých pár činností a v nich se zdokonalovat jak praxí, tak zkušenostmi anebo školením (Myslín, 2016, s. 23).

Obecným předpokladem k použití této metodiky je, že je kladen velký důraz na důkladné zpracování veškeré dokumentace. Do dokumentace se zpracovává vše od komunikace se zákazníkem, jednotlivé požadavky, průběh vývoje, požadavky ze změnového řízení, testování a reportování chyb až po přesunu do ostrého provozu a udržování. Vývojáři začínají s faktickým vývojem až v momentě, kdy je dostupná kompletní analýza s popisem na požadovaný konečný stav. To přináší na zákazníka mírný pocit zoufalství a beznaděje. Převážně je to způsobeno časovou náročností tradičního modelu a díky časové náročnosti je zákazník dlouhou dobu bez nějakého hmotného výsledku práce (Myslín, 2016, s. 23).

Časová náročnost a větší počet pracovníků na projektech se projevuje i v provádění jakýchkoliv změn. Některé implementování změn se může provádět v řádu týdnů i měsíců, než se projde celý proces této metodiky. Přes tyto uvedené nevýhody se může jevit tradiční způsob procesu vývoje softwaru jako velice nepraktický. Toto však vyvrací bezesporu velké výhody, kterými jsou pravidla, zásady, předpokládatelnost a očekávatelnost. Jsou zde jasně dané role, rozdělení jednotlivých úkolů a prací a tím je do určité míry dán i časový plán a náklady (finanční a zdroje).

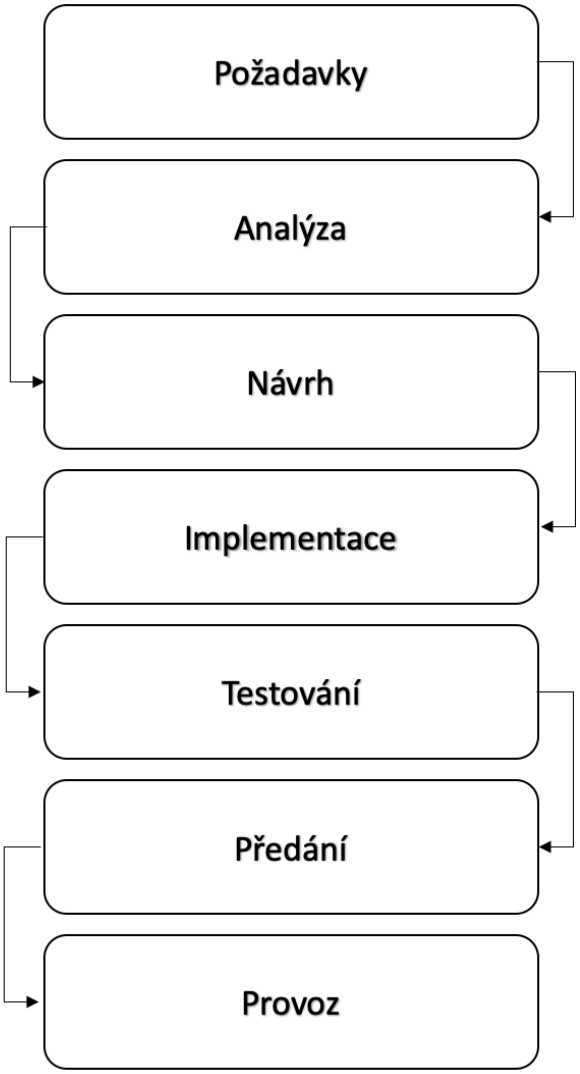
Tyto obvyklé metodiky se mohou uplatňovat na úkolech, kde je třeba řešit obsáhlé dílčí úkoly s vyšší časovou náročností. Dále to mohou být úkoly, kde je tým rozdělen zeměpisnou lokací, nebo úkoly, kde se implementují různé technologie či druhy softwarů. I přes uvedené nevýhody výše se tyto tradiční metodiky uplatňují u projektů, kdy je dán jednoznačný rozpočet anebo je nutné dodržet danou funkcionalitu či časový harmonogramu celého projektu (Myslín, 2016, s. 24).

1.2.2 Typy tradičních metodik

Tato dílčí kapitola popisuje a charakterizuje některé z tradičních metodik při projektovém řízení vývoje softwaru.

1.2.2.1 Waterfall model (vodopádová metodika)

Jedná se o jednu z nejstarších metodik pro projektové řízení vývoje softwaru. Jde o přímý a jednoduchý model. Tento druh modelu vznikl v sedmdesátých letech. V současné době je pro velkou většinu řízení projektu nedostatečný. Jednotlivé kroky, které charakterizují tento model, jsou zobrazeny na obrázku č. 2 – vodopádový model.



Obrázek 2 - vodopádový model
(zdroj: upraveno dle Myslín, 2016, s. 25)

Z obrázku je patrné, že tato metodika je velmi přímočará. To znamená, že každá jedna fáze přechází na další jednu konkrétní fázi. V tom vzniká velká přehlednost, kdy vedoucí projektu a všichni účastníci projektu vědí, v jaké fázi se aktuálně projekt nachází. Tím je „ošetřena“ situace, kdy by se prováděla jiná činnost na projektu než ta, která souvisí s danou fází. Jak už bylo ale zmíněno výše, tak i přes jednoduchost, která je sice v pořádku, má tento model příliš nevýhod hlavně u komplexnějších a obtížnějších projektů (Myslín, 2016, s. 25.)

V tom, jak je každý softwarový projekt jiný, tak jsou i jinak určeny jednotlivé SDLC kroky (Software Development Life Cycle). Závislost na určení těchto kroků mají jak interní, tak i externí vlivy. Vždy ale jde celý proces chronologicky za sebou (Singh, 2019, s. 4).

Velmi důležitý nedostatek, který tato metodika nemá vyřešený, je zapojení zákazníka do procesu do vývoje. V celém procesu se zákazník dostane ke slovu pouze při předání požadavků na software a následně až už předání hotového produktu. Zde vzniká problém, že zákazník bohužel nemá znalosti a nemusí znát veškeré možnosti anebo nezná omezení, které mohou vzniknout na základě technologie nebo legislativy. U předání uživatel vidí zas až hotový výsledek v podobě celkového produktu. Tím pádem od něj mohou vzniknout připomínky na nedostatky, chyby, nepřesnosti a podobně. Jakmile se některá z těchto věcí objeví, tak se celá hotová aplikace přesouvá opět na začátek. A i díky této vzniklé nepřesnosti může vzniknout stav, že se musí přepracovat celý projekt. A s tím jsou spojeny nepříjemnosti v podobě dalšího využívání pracovních kapacity, které mohly řešit již jiné projekty, náklady na projekt, prodloužení časového harmonogramu a podobné. Zadavatel projektu tyto zádrhely v procesu taky nebere pozitivně (Myslín, 2016, s. 25.).

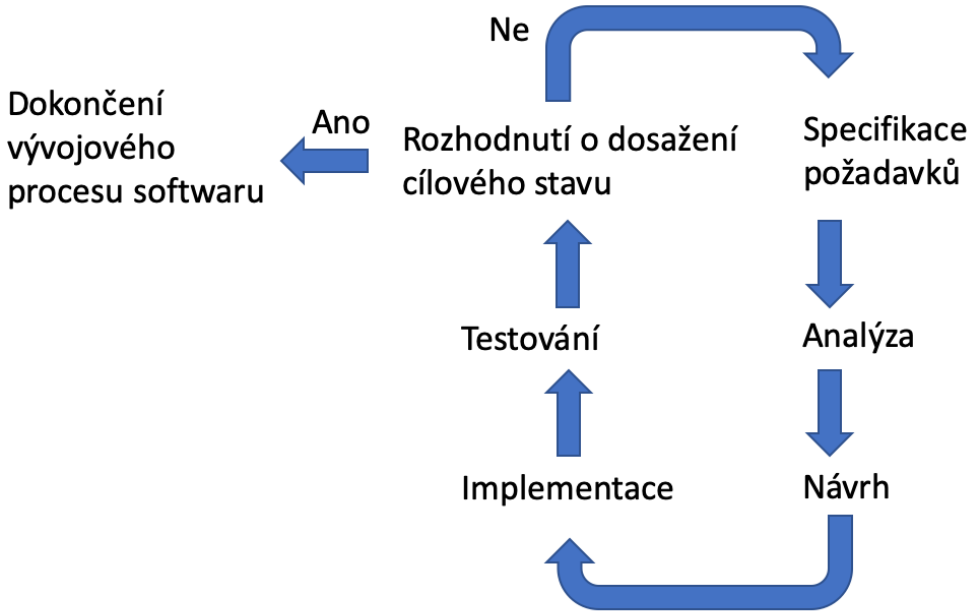
1.2.2.2 Iterativní metodiky

V této kapitole není popisována jedna konkrétní metoda, ale spíše celý druh metodiky obecně. Tato metodika bere v potaz největší nevýhody vodopádové metodiky a je zde snaha tyto neduhy vyřešit. To znamená, že je zde zčásti řešeno to, že se zde již nevyskytuje taková velká časová náročnost na ukázání prvních výsledků. Dále zde je systém pro rychlejší nalezení nedostatků, zapojení zákazníka do průběhu vývoje nebo probíhá dřívější otestování vyvíjené aplikace. Nedostatky vodopádové metodiky jsou řešeny tím, že jsou zde prováděny takzvané „iterace“. Iterace se provádí v celém průběhu vývoje aplikace, tudíž od začátku vývoje až do konečného stavu. Tvoření iterací spočívá v tom, že se vývoj aplikace provádí

od hrubého vývoje a každou iterací se provádí další vývoj aplikace. Postupně se od tohoto hrubého vývoje ladí celý vývoj až po největší detaily. Tyto iterace probíhají v šesti krocích. Jednotlivé kroky jsou charakterizovány v následující tabulce (Myslín, 2016, s. 26.).

1. Specifikace požadavků	Požadavek na uživatele, aby sdělil, co je vůbec třeba provést za úpravy.
2. Analýza	Vykonají se nutné přeměření požadavků.
3. Návrh	Navrhuje se pořadí pro práci při vykonávání vývoje.
4. Implementace	Provedení činnosti a změn a zahrnutí do celkového softwaru.
5. Testování softwaru	Je zkontrolováno, zda bylo dosaženo požadovaného stavu při postupu vývoje během iterace.
6. Posouzení cílového stavu	Pokud není dosaženo cílového stavu softwaru, tak se v tomto cyklu vrací proces vývoje k bodu číslo jedna.

Tabulka 1 – charakteristika jednotlivých kroků konkrétní iterace
(Zdroj: Myslín, 2016, s. 26)



Obrázek 3 - iterativní metoda vývoje softwaru
(Zdroj: upraveno dle Myslín, 2016, s. 27)

Na základě cyklu, který je zobrazen na obrázku číslo tři, je vidět právě základ všech iterativních metodik. Těchto iterativních metodik existuje několik. Jako konkrétní metodiky jsou uvedeny například Rational unified proces nebo spirálová metodika.

Společnou vlastností všech těchto metodik je hned několik bodů. Po každé iteraci vývojového procesu je spustitelný kód. Uživatel po každé iteraci získá funkční software, kde je vidět postup z vývoje konkrétní iterace. S tímto postupem se může seznámit a na základě toho vrátit vývojové společnosti zpětnou vazbou některé poznatky. Může se jednat buď o návrhy k vylepšení softwaru, nalezené nedostatky či chyby, změny v prostředí, funkčnosti a podobně. Další vlastností je, že každá iterace je důkladně otestována. Při otestování jsou nalezeny chyby a tyto chyby jsou opravovány v rámci dalšího začátku následující iterace. Poslední vlastností je, že každou iterací se ke slovu dostává i zákazník a jak bylo zmíněno výše, tak zákazník aplikaci otestuje ze svého pohledu a svých nároků a potřeb. Na základě toho provede posouzení, zda je vývoj aplikace provádí správným směrem a oponuje

vývojové firmě různými náměty na změnu, rozpory od jeho původních požadavků a jiných zlepšení.

V iterativních metodikách se využívají i některé nástroje, o kterých je také dobré se zmínit. Mezi toto patří nástroje typu CASE (Computer Aided Software Engineering – softwarové inženýrství podporované počítačem). Nástroje typu CASE mohou být využívány díky tomu, že dovolují uplatňovat grafické modely. Tyto grafické modely mohou neuvěřitelně podporovat práci na větších projektech. Tato podpora na větších projektech je uplatňována protože, grafické modely ulehčují orientaci v projektu. Zlepšená orientace je poté znát díky lehčímu a lepšímu pochopení jednotlivých spojitostí v rámci průběhu celého procesu vývoje projektu. Je samozřejmé, že pomůcky poskytující CASE nástroje, je možné využívat i v jiných rozdílných metodikách. Zde je to však zmíněno z důvodu, že právě v iterativních metodikách tyto nástroje ukazují svojí opravdovou sílu využitelnosti. Kdyby se CASE nástroje využívaly například u vodopádového modelu, tak zde vzniká velká překážka v tom, že by celý projekt musel být hned od začátku celkově vymodelován k dokonalosti. To je velmi velký a náročný nárok na celý projekt. Proto by bylo možné to brát i jako negativní požadavek nebo činnost při této metodice a tím pádem i zcela nadbytečný úkon (Myslín, 2016, s. 28.).

1.2.3 Agilní vývoj softwaru

Tato kapitola celkově popisuje význam agilního vývoje softwaru, jeho vlastnosti a charakteristiky.

Agilní vývoj je spojen s některými charakteristickými pojmy. Patří k nim výrazy typu svižný, okamžitý, rychlý nebo interaktivní. Pomocí tohoto termínového úvodu se dostaneme k základním bodům charakteristiky popisu agilního vývoje softwaru (Šochová, 2019, s.15).

1.2.3.1 Individuální osoba před procesním vývojem a softwarovým náčiním

Metoda se zakládá na hypotéze, že spolupracující skupiny dosahují daleko lepších výsledků než osamocení pracovníci, kteří společně pracují v rámci vývojového procesu. Softwarové nástroje pouze pomáhají s vývojem, ale k úspěchu nejsou až takovým dílem třeba, jak by se zdálo. Proto v dnešní době zažívají úspěchy i takzvané startupy. Je to z důvodu toho, že tyto

startupy pracují vzájemně a těsně ve skupinách, aniž by používaly některé pokročilé softwarové nástroje. Jde o to, že právě tato těsná spolupráce skupiny přináší větší šanci na celkový úspěch. Aneb i když má uživatel sebelepší softwarový nástroj, ale nepřekypuje příliš inteligencí, tak i tento sebelepší nástroj takovému uživateli příliš užitku nepřinese (Šochová, 2019, s.17).

1.2.3.2 Software, který pracuje správným a požadovaným způsobem, má přednost před důkladně sepsanou dokumentací

U tohoto bodu je důležité mít na zřeteli jednu důležitou věc, a to potřeby jednotlivých uživatelů, co se týče nákupu softwaru. Z logiky věci vyjde důležitý předpoklad, že uživatel si radši pořídí software, který pracuje a funguje požadovaným způsobem než software, který má absolutně vypracovanou dokumentaci do sebemenšího detailu, ale ve výsledku může fungovat jiným způsobem. V praktickém příkladu je možné použít například nákup konkrétního produktu, a to například telefonu. Kdo kdy otevřel krabičku s nově nakoupeným telefonem a místo, aby telefon spustil, tak začal nejdříve číst manuál a seznamovat se s produktem touto cestou? Jedná se o velmi malé procento uživatelů. Z toho plyne, že většina uživatelů se jednoduše seznamuje s produktem tím, že jde praktickou cestou. Je ale důležité upozornit, že dokumentace má svoji úlohu a vyplatí se ji vypracovávat. Měla by být ale vypracovávaná až na druhém místě, po práci na konkrétním produktu a měla by uživateli poskytovat pouze rámcové informace o produktu.

Dalším případem vytváření dokumentace je dokumentace funkcí v procesu vývoje neboli dokumentace interní. Interní dokumentace by také neměla mít vyšší prioritu než samotný vývoj softwaru. V tomto ohledu je důležité, aby panovala komunikace na vysoké úrovní mezi vývojáři, analytiky a testery. Pokud existuje takováto komunikace, tak je pak možné se pouze dohodnout, co je třeba zadokumentovat pro budoucí pracovníky. Tím je opět nutné se vyhnout nesprávnému dojmu z předchozího tvrzení a to, že by se neměla zpracovávat dokumentace vůbec. Dokumentace má svoje místo ke zpracování, pouze je třeba brát v úvahu její prioritu a množství (Šochová, 2019, s.19).

1.2.3.3 Blízká součinnost se zákazníkem je důležitější než dohady nad smlouvou

V každém případě je třeba brát ohled na zákazníka. Na zákazníka je třeba brát velký ohled z důvodu, že software se vyvíjí s cílem ho prodat právě konkrétnímu zákazníkovi. Je vhodné

s ním tedy komunikovat, tázat se ho na požadavky, názory a jeho celkový pohled na věc. Je pravda, že zákazník je také pouze lidská bytost, která čas od času změní svoje tvrzení. S tím je nutné počítat. Ovšem z hlediska dlouhodobé komunikace je možné ze zákazníka vytvořit takzvaného externího pracovníka týmu, který vyvíjí software.

Uzavření dohod nebo smluv je rozhodně důležitým krokem k navázání spolupráce, ale tato dohoda by se neměla uzavírat „za každou cenu“. Je podstatné, aby došlo k sebekritickému zamyšlení, co je vůbec možné spoluprací získat, co není a pokusit se uzavřenou smlouvou dostat co nejvíce k možné skutečnosti. Zákazník, který dostane, co chtěl a odejde od obchodu spokojený, tak bude vytvářet kladnou referenci na vývojovou společnost. Tato reference se v budoucnosti může zhodnotit získáním dalších zakázek. Během valné většiny vývojových procesů dojdou obě strany do bodu, kdy bude muset být tvořen kompromis z obou stran. Mohou vznikat takzvané change requesty, kdy polovinu například zpracuje vývojová společnost na svoje náklady a polovina zaplatí za extra peníze zákazník jako zisk nové funkčnosti (Šochová, 2019, s. 21).

1.2.3.4 Dostatečně odpovídat na změny je důležitější než striktní dodržování časového harmonogramu

Současná doba se vyznačuje několika charakteristickými vlastnostmi. Vlastnosti doby jsou, že je rychlá, neustálená a stále se mění. Je třeba na takový proměnlivý stav včas reagovat a případně pozměnit původní plán. Je možné si představit konkrétní případ, který může nastat. V poslední závěrečném kroku vývojového procesu softwaru přijde zákazník, který si software objednal. Sdělí, že je třeba provést důležitou úpravu a že na tom závisí celkové přežití zákaznické firmy. Na toto je možné zareagovat dvěma způsoby. Buď se bude stále dodržovat dříve navrhnutý plán a ten se dokončí anebo na to dodavatelská firma zareaguje, zohlední požadavky ve vývojovém procesu a zpracuje je. Samozřejmě, že ta správná cesta je druhá možnost. Díky tomuto vyhovění na požadavky získáme dalšího dlouhodobějšího zákazníka, získáme další reference, která nám přinese konkurenční výhodu. Z toho vyplývá, že časové harmonogramy a celkově plány projektového vývoje jsou důležité, ale měly by se brát pouze jako taková instruktáž. Nemělo by se stát, že by se tohoto plánu mělo držet „zuby nehty“. (Šochová, 2019, s. 23).

1.2.4 „Lean“ metodika

Metodika, která existuje vedle tradiční a agilní. Tato metodika se vyznačuje co nejužším vývojovým procesem. Blíže má spíše k agilní metodice, jelikož zde nejsou používané striktní procesy. Tuto metodiku používá například velice známá automobilová společnost Toyota. „Lean“ metodika neboli štíhlý proces výroby spočívá v tom, že se požadovaná práce vykonává, až když je potřeba. Například zmíněná společnost Toyota, tak zde by se vyráběl díl na sklad až v momentě, kdy je opravdu nutně potřeba. Takto řídí proces výroby systém tahu. Tuto metodiku je možné použít i při softwarovém vývoji. Aplikuje se tak, že se omezí celková práce „work in progress“ a zaměří se jednotlivé úkoly, aby se dokončily v co nejkratším časovém horizontu. Pokud se tyto úkoly dokončí, tak se dále v tom konkrétním procesu dál nepokračuje, dokud nepřijde další potřeba nebo požadavek s prioritou. Úvaha této metodiky se uplatňuje u metody s názvem Kanban. Lean Software Development charakterizují body popsané níže (Šochová, 2019, s. 25).

1.2.4.1 Odstranění přebytečných činností

Nemá cenu trávit čas na některém úkolu nebo činnosti, která se ve výsledku ani nevyužije. Jednalo by se o téměř zbytečný náklad, který přinese opravdu málo užitku, a ještě méně nějakého příjmu. Oproti tomu se vyplatí investovat tento čas, který ušetříme, tam, kde to bude mít větší smysl (Šochová, 2019, s. 25).

1.2.4.2 Zlepšování a učení se během procesu

Vývojářskému týmu se může lehce stát jedna nepříjemná věc. Pokud bude slepě bez nějakého hlubšího zamyšlení vypracovávat svoji činnosti dle nějakých směrnic, norem nebo pravidel, tak se vystavuje riziku. Riziko spočívá v tom, že může vzniknout během procesu chyba a na základě těchto pravidel se může stejná chyba opakovat. Ke konci vývojového procesu se tak může stát, že se nahromadí několik těchto chyb nebo nedostatků. Řešením problému je minimálně získání feedbacku od zákazníka. Tento feedback dokáže vývojářský tým lehce narovnat, aby se zaměřil na důležitější činnosti místo tvoření dalších zbytečností (Šochová, 2019, s. 25).

1.2.4.3 Tvoření rozhodnutí v nejzazším termínu

Existuje hypotéza, že čím déle se čeká na vytvoření rozhodnutí, tím lépe. Lépe je to z toho důvodu, že rozhodující osoba pak bude mít co největší množství informací a bude v lepší rozhodovací pozici, než kdyby tvořil nějaké unáhlené rozhodnutí (Šochová, 2019, s. 26).

1.2.4.4 Rychlý vývojový proces

Základní myšlenka „lean" metodiky je rychlý vývojový proces. Čím dříve odešlu zákazníkovi vykonanou práci a vytvořený software, funkčnosti nebo cokoli jiného, tím dříve dostanu od zákazníka feedback. Feedback pak slouží k dotažení nedostatků nebo chyb, které budou řešeny v dalším vývojovém cyklu neboli iteraci (Šochová, 2019, s. 26).

1.2.4.5 Odpovědnost a víra v tým

Aby tato metodika byla co nejvíce efektivní, tak musí fungovat vývojový tým, ve který je vložena patřičná důvěra. Tým s důvěrou bude motivovanější a tím pádem i efektivnější (Šochová, 2019, s. 26).

1.2.4.6 Orientace na konečný stav

Zde se hodí velice trefná citace dle (Šochová, 2019, s. 26) „*…jednotlivé chyby a selhání nejsou podstatné, jestliže se z nich poučíte. „Think big, act small, fail fast; learn rapidly…"*.

Z citovaného textu je vidět krásně myšlenka celé „lean" metodiky. Je třeba se podívat dopředu a pokud možno vidět končenou představu. Pouze v takovém případě je možné zjistit, zda výsledný stav bude úspěch nebo naopak. S tím je spojená další věc. Konečný stav není pouze o vyhotoveném softwaru, který se prodá zákazníkovi. Jde i o dojem, který výsledný software poskytuje. Je třeba tedy dávat velkou ostražitost na celkovou kvalitu. (Šochová, 2019, s. 26).

1.2.5 Výhody agilních metod

Agilní metody procesního vývoje softwaru mají hned několik výhod, které jsou popsány v následujících odstavcích.

1.2.5.1 Flexibilita

Striktní metody se vyznačují zdlouhavými procesy. Tyto zdlouhavé procesy jsou důsledkem samotné striktní metody. Analytický tým musí nejdříve celou potřebu od zákazníka projít a zjistit možnosti. Dále je nutné tuto potřebu od vývojářského týmu napsat. Posledním krokem je testovací fáze. Přesně tato celková časová náročnost u agilních metod odpadá, protože se vyznačují flexibilitou. V současné době, kdy zákazník chce požadovanou věc v co nejkratším časovém úseku, je vhodnější dodávat jednotlivé dílčí požadavky po menších částech, ale zato nejlépe v okamžiku, kdy jsou vytvořené (Šochová, 2019, s. 31).

1.2.5.2 Efektivnost

Jak bylo popsáno již výše, tak na základě vyhodnocení několika studií se došlo k závěru, že práce jednotlivce je méně efektivní než práce spolupracujících jednotlivců neboli týmu. Co se týče konkrétně vývoje softwaru, tak několik studií ukázalo, že nejefektivnější způsob práce v týmu je takzvané párové programování. Tato možnost se vyznačuje tím, že na jednotlivé činnosti jsou připojeny dva pracovníci. Další efektivní možností, kterou doporučuje Scrum, je utvořit pevně spolupracující tým, který má společný požadovaný výsledek. Zpočátku může trvat, než si jednotlivci na sebe dokáží navyknout, ale pozdější efektivita tyto úvodní problémy dokáže velice vynahradit (Šochová, 2019, s. 31).

1.2.5.3 Očekávatelnost

Agilní metodika se dále vyznačují tím, že pokud vývojový tým nedokáže plně tvořit předpoklady časového harmonogramu, a nikdy se tento časový harmonogram nepovede splnit, tak právě agilní metodiky odhalují nový způsob určování časového harmonogramu. Časový harmonogram se určuje pouze v relativních číslech, a i v tomto procesu se zapojuje celý pracovní tým. Další možností, jak zlepšit očekávatelnost při procesním vývoji v agilní metodice je, že se celý projekt rozdělí na několik menších částí. Tyto menší části se pak dají lépe časově odhadnout, než jednu velkou část projektu (Šochová, 2019, s. 32).

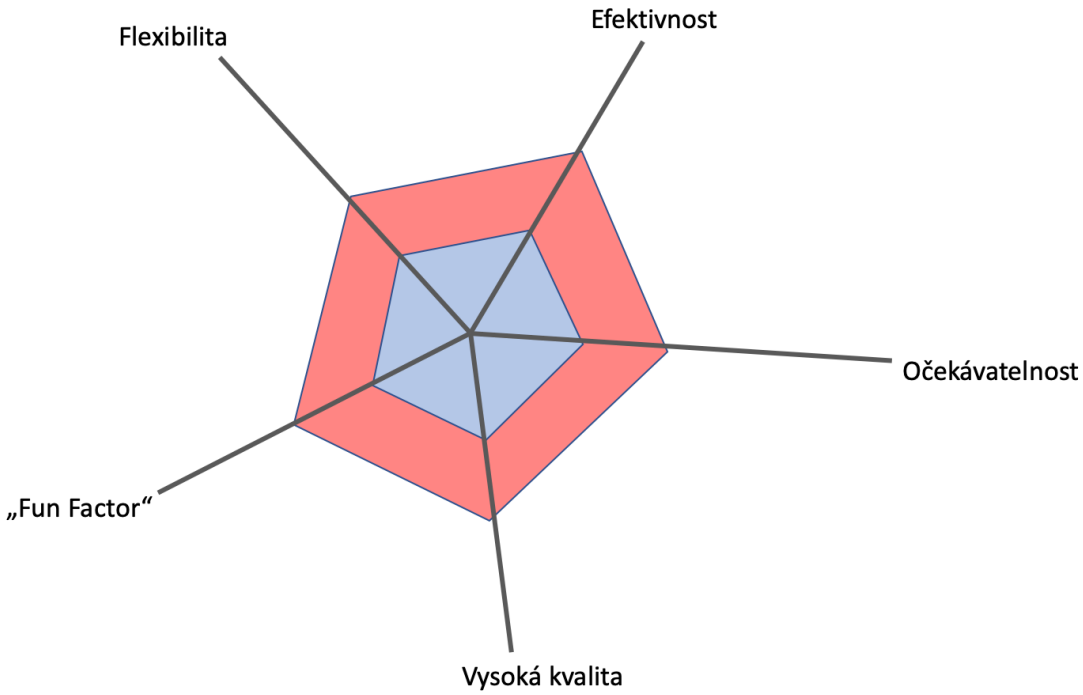
1.2.5.4 Vysoká kvalita

Tento bod je hodně zaměřen na zákazníka, který požaduje software. Tvoří se s ním celá analýza, aby dodavatelská firma zjistila přesné důvody, co požaduje, proč to požaduje, jak to požaduje a jak bude s novou věcí pracovat. Poté mu je celý projekt po malých kouscích dodáván. Tím je mu zobrazována skutečnost softwaru a zákazník může určovat směr vývoje

projektu skrze zpětnou vazbu. Zvyšuje se tak celková kvalita celkového výsledného softwaru. Nemůže se tedy pak na konci projektu stát, že zákazník bude brát výsledek projektu jako nepoužitelnou věc, což se při striktních metodách může opravdu stát. Další atribut vysoké kvality je ten, že jak se o celý proces stará jeden tým, tak tento tým si udržuje nějaký standard, který snižuje počet vytvořených nedostatků nebo chybovost a tvoří dlouhodobě udržitelný programový kód (Šochová, 2019, s. 32).

1.2.5.5 „Fun Factor“

Ač tento poslední bod může vypadat jako hloupost, tak opak je pravdou. Tím, že zde probíhá dlouhodobá práce jednoho týmu na jednotlivých procesech, tak vzniknou mezi jednotlivými členy i přívětivé osobní vztahy. Budou si rozumět jak mezi sebou, tak i zákazníkům. Navzájem zde bude probíhat proces motivování. A je opravdu velký rozdíl, jestli vývoj softwaru tvoří jeden otrávený vývojář nebo takovýto tým (Šochová, 2019, s. 32).



Obrázek 4 - imperativ vlastností agilní metody řízení procesu
(Zdroj: upraveno dle Šochová, 2019, s. 34)

1.3 Scrum

Scrum je agilní metodika, která využívá principy otevřené a pochopitelné debaty jednotlivých členů v procesu, samostatně organizovaného kolektivu a flexibilního způsobu vykonávání pracovních činností. Aby Scrum metodika mohla fungovat a produkovat požadované výsledky, tak zde musí existovat speciální role týmu. Jedná se o Scrum Mastera a Product Ownera (Šochová, 2019, s. 43).

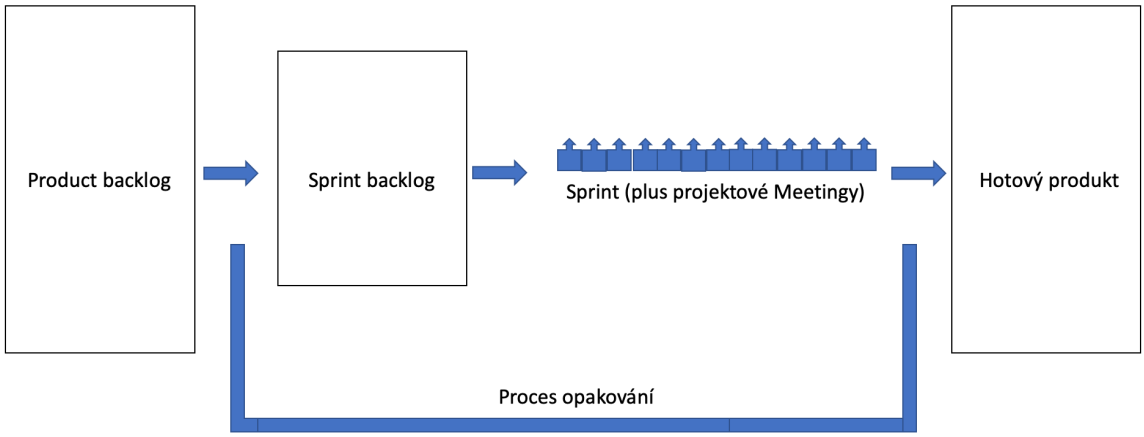
Projekt zde probíhá v krátkých intervalech. Zpravidla se jedná o několikatýdenní úseky, které mají speciální terminologii a to „**Sprint**“. Na konci každého Sprintu dochází k vyhodnocení, které dopomáhá k rychlému získání výsledku. Zároveň to pomáhá i k odhalení problémů a nedostatků a vytvoření reakce na ně. Tato metoda je zhruba čtyřikrát více efektivní než klasický přístup k řízení procesů. Problém při přechodu na tuto metodu může spočívat ve stereotypním myšlení firmy a celková představa o uspořádání firmy (Porada: Co je to agilní metoda řízení?, 2018).

Na konci intervalu, a tedy na konci konkrétního Sprintu by měla být hotova nějaká funkcionalita, respektive celý projekt. Jestliže je požadována část a tato část se odhaduje s vyšší časovou náročností, než pojme jeden sprint, tak by se měla rozdělit na několik částí. To celému projekt snižuje šanci na výskyt chyb a zvyšuje se přesnost při tvorbě časového harmonogramu.

Ze začátku celého projektu dojde k nahromadění požadavků od zadavatele projektu. Tím projektovému týmu vzniká takzvaný „Product Backlog“. Toto by se dalo představit jako takový spis, ve kterém jsou sepsány veškeré požadované funkce na výsledný produkt. Poté se na základě určení priorit jednotlivých funkcionalit provede zařazení do takzvaného „Sprint Backlogu“, který si lze představit jako takový časový harmonogram jednotlivých fází projektu. Během projektu celý projektový tým provádí takzvané „Meetingy“, kde diskutuje a rozhoduje o věcech souvisejících s projektem, a o jeho jednotlivých částech (Doležal, 2016, s. 315).

Na obrázku níže je zobrazeno jednoduché schéma toho, jak si lze představit takový Scrum model projektu s uvedenými termíny jako je „Product Backlog“, „Sprint Backlog“, „Sprint“ a jeho postupný proces. Nejdříve se tedy začíná Product Backlogem s jednotlivými

požadavky na projekt, poté Sprint Backlogem, kde jsou uvedené požadavky v jednom Sprintu. Sprint Backlog a Sprint fungují v cyklu. Opakují, dokud nejsou hotovy veškeré požadavky z Product Backlogu a na základě toho vznikne i hotový produkt.



Obrázek 5 - Sprint a jeho schéma
(Zdroj: upraveno dle Doležal, 2016, s. 314)

V následujících kapitolách jsou popisovány jednotlivé role v týmu a pojmy, které se používají při agilní metodě Scrum.

1.3.1 Hlavní pozice metody

V této kapitole jsou popsány hlavní role projektového týmu z konkrétní agilní metodiky Scrum.

1.3.1.1 Scrum Master

Tato role týmu by se dala přirovnat k obyčejnému týmovému vůdci. Ovšem od této konkrétní role se liší díky několika vlastnostem. Osoba, která zastává tuto roli v týmu, má za cíl sestavit tým. Na tým jsou kladeny požadavky, jaké vlastnosti má splňovat. Především samostatnost, efektivnost a schopnost organizovat jednotlivé činnosti. Scrum Master je nápomocen svému

týmu, kterému pomocí podpory, konzultací a jiných nástrojů zajišťuje, aby tým byl vysoce výkonný. Dále se snaží odstraňovat veškeré nedostatky a chyby, které v týmu panují anebo se snaží zabránit vzniku možných chyb a nedostatků. V neposlední řadě má na starost motivaci a team leading, kde musí být snaha i o případné zaučení týmu. Z toho plyne, že Scrum Master by měl být člověk, který je na vysoké komunikační úrovni, dokáže se vcítit do jakéhokoliv problému a dokáže tlumit a elimininovat veškeré problémy v rámci svého týmu. Jako takový by měl být efektivní a musí umět odhadnou, kdy je potřeba provést změnu a případnou změnu iniciovat. Je to velmi důležitá část týmu. Když plní všechny svoje závazky, tak tým pracuje tak jak má. Plní se požadované cíle. Pokud by se role Scrum Mastera měla přirovnat k jinému oboru, tak Scrum Master je seřizovač stroje a stroj je projektový tým. Snaží se, aby vše pracovalo tak jak by mělo, aby se žádná ozubená kolečka nepřestala točit a stroj pracoval plynule na sto procent. Aby byl Scrum Master nejefektivnější, tak by měl sedět se svým týmem v jedné místnosti. Pracuje – li Scrum Master s požadovanými výsledky, tak je celý tým efektivnější, tvoří s vyšší kvalitou a motivací. Tím i společnosti, ve které pracuje, zajištuje vyšší zisky (Šochová, 2019, s. 43).

Scrum Master však není odpovědný za to, zda se zákazníkovi dodá požadovaný produkt nebo ne. Scrum Master se opravdu stará pouze o svůj tým a o to, aby byl tým funkční, produktivní a efektivní. Na roli Scrum Mastera by se měla dosazovat jiná osobnost než na roli vůdčího celého týmu, protože ten většinou prosazuje a tvoří rozhodnutí za tým, nebo na roli experta, protože ten zas ve většině případů ani nepokládá otázky. Ani projektový manažer se nehodí na Scrum Mastera, protože vše řídí sám a provádí mikromanagement své organizace týmu, priorit a dodání požadovaného stavu zakázky. Některé společnosti přistupují i k tomu, že jeden Scrum Master řídí několik podřízených týmů. Toto ale taky není správná role Scrum Mastera. Vlastnosti Scrum Mastera by měly být dobrá komunikace, schopnost vést a učit ostatní lidi v týmu a facilitovat. Zároveň by Scrum Mastera neměla dělat tatáž osoba, která již má roli Product Ownera, jelikož tyto dvě role mají protichůdné cíle. (Jak vybrat Scrum Mastera aneb proč i zkušení Scrum Masteři selhávají v Agilní transformaci, 2014).

Pokud je shrnuta v konkrétních bodech role Scrum Mastera, tak:

- Provádí tvorbu prostředí projektového týmu
- Přebírá odpovědnost za řízení Scrum fází
    - Facilituje jednání, dbá na dodržování časového harmonogramu
- Odklízí vyrušující prvky, aby se tým mohl soustředit na produkt
- Podporuje Product Ownera
- Scrum Master nemá a nezískává rozhodovací moc

(Doležal, 2016, s. 316)

1.3.1.2 Product Owner

Jak už z přeloženého názvu této role lze poznat, tak se jedná o roli, která má na starosti produkt. Jedná se o další klíčovou osobu při použití agilní metodiky Scrum. Je to spíše marketingový expert, který udává celému projektu vizi a cíl. Mezi to samozřejmě patří uspokojení potřeb zákazníka a zadavatele projektu. Jelikož se projekt během času může měnit, rozrůstat nebo jinak „bobtnat“, tak Product Owner má spíše takovou „uzemňovací“ funkci (15 Project Management Methodologies You Need to Know About, c2016).

Jinými slovy Product Owner je jakýsi produktový manažer v agilním světě. Úspěšný Product Owner musí mít výborné podnikatelské vlastnosti. Přebírá veškerou zodpovědnost za financování celého projektu při rozhodování jak za obchodní stránku, tak i za IT strategii. Jestliže je nastavení mysli Product Ownera správné, tak by se měl brát v úvahu návratnost investic z projektu (ROI – return on investment), protože správný postup Product Ownera je investování financí společnosti, jako by investoval finance svoje. Product Owner zodpovídá za tvorbu Product Backlogu spolu s celým týmem, stakeholdery, zákazníky a ostatními členy týmu projektu. Product Owner by neměla být osoba, která se kombinuje se Scrum Masterem. (McGreal, ©2018, s. 45).

Pokud je shrnuta v konkrétních bodech role Product Ownera, tak:

- Každý projekt má pouze jednoho Product Ownera
- Zodpovídá u projektu za maximalizaci investic, které se navrátí (ROI)
    - To znamená, že dokáže s projektovým týmem pracovat efektivně a na částech, které přináší největší hodnotu
- Zodpovídá za Backlog, čímž určuje, jakou mají prioritu jednotlivé úkony
- Poskytuje celkový popis, požadavky a specifikaci požadovaného stavu projektu a produktu díky komunikaci se zákazníky a zadavateli projektu
- Product Owner je ta osoba, která určuje, zda je produkt hotový a tím pádem i výsledný

(Doležal, 2016, s. 315)

1.3.1.3 Self-organized tým

Jedná se o další stavební kámen pro správné fungování agilní metodiky pří procesním řízení projektu. V předchozích dvou kapitolkách byla řeč především o jednotlivcích. Nyní se jedná o celý tým.

Vývoj softwaru je velmi komplexní problematika, kdy je třeba, aby fungovala správně veškerá komunikace a týmová spolupráce. Tyto úkony není možné dopředu naplánovat a tím, že každá firma je odlišná, má odlišné požadavky, tak bude mít i odlišný projektový tým. Základní a snad tou nejdůležitější predispozicí self-organized týmu je to, že všichni členové konkrétního týmu musí mít společný cíl (!). Za tímto společným cílem musí sdílet společnou vizi a táhnout za jeden provaz. Další neméně důležitou vlastností self-organized týmu je důvěra. Jestliže si jednotlivé členové týmu nebudou důvěřovat anebo i zákazník nebude důvěřovat projektovému týmu, tak se požadovaného cílového stavu dosahuje o poznání složitěji. V neposlední řadě je umožnění self-organized týmu a všem jeho členů, aby se podíleli na stavbě Product Backlogu. Jestliže konání týmu skončí neúspěchem, tak se nehledá viník. Za neúspěch může celý projektový tým. Pokud se například v posledním Sprint Backlogu nestihlo otestovat vše, protože v týmu je pouze jeden tester, tak to není chyba toho testera, ale celého projektového týmu. Časová organizace měla být tomuto faktu lépe uzpůsobena, aby se toto nestalo a vše se dokončilo tak v pořádku. Projektový tým také nemůže rozhodovat o svých členech. O členech se rozhoduje na úrovni organizační struktury

podniku. Tým také nerozhoduje, dle jaké metodiky se budou jednotlivé projekty zpracovávat. V projektovém týmu panuje organizační duch, který se řídí dle pravidel dané metodiky (Šochová, 2019, s. 49).

- Self-organized tým by měl být tvořen cca sedmi členy. Pokud by se stalo, že by se tým rozrostl do většího počtu, tak dochází k přerozdělení celého projektového týmu do několika menších projektových týmu. Je to z důvodu toho, že v týmu musí probíhat personální vazba a menší tým je lépe udržitelný a řiditelný.
- V Self-organized týmu by měly být zastoupeny všechny potřebné odbornosti, které jsou při řešení projektu třeba. Je to proto, aby se co nejvíce dílčích úkolů tvořilo paralelně a tím se zvyšovala efektivnost a snižovala časová náročnost celého projektu.
- Podmínkou týmu je, aby byl v kontaktu se zákazníkem nebo zadavatelem projektu, aby se mohlo v procesu projektu upřesňovat zadání a jednotlivé kroky a aby se komunikovali dílčí úkony a požadované přínosy.
- Tým, který je takto samoregulující, by měl mít faktický fyzický kontakt kvůli urychlení komunikace a díky tomu snížení časové náročnosti a zvýšení efektivnosti.
- Tým v závislosti na celkové osobní komunikaci a časového odhadu dohaduje cíle jednotlivých sprintů, jejich plánování a následně za jejich splnění.

(Doležal, 2016, s. 315)

V každém týmu jsou jednotliví členové týmu, kteří jsou experti na různé typy činnosti při vývoji softwaru. V tomto případě musí existovat stav, že právě tito jednotliví členi projektového týmu jsou vzájemně zastupitelní. A je jedno, jestli se jedná o různé komponenty, Framework, GUI, databáze, či některý určitý systém. V týmu se provede dohoda toho, kdo bude na čem pracovat a kdo bude na čem pomáhat. Tím se začne pomalu budovat tým, který bude ve výsledku pracovat jako efektivní celek. Toto vybudování však není krátkodobá činnost. Jedná se o investici do flexibilnosti a efektivnosti úspěšného projektového týmu. Vybudování takového týmu není ani tolik náročné, jak by se mohlo zdát na první pohled. Musí se dbát velké pozornosti na styl práce a vybudování nějaké sdílené databáze znalostí. Role analytika, programátora nebo testera bude v některých činnostech taky sdílená v určitých případech. Tyto role mohou vymýšlet různé test-casy, kde zkouší

vymyšlený postup aplikace, kde provádí test právě té aplikace. Právě tímto testem probíhá vzájemná pomoc rolí. A nemusí to končit pouze u testu. Dále pomoc může probíhat i tím, že se provede následná analýza. Scrum tým, který pracuje optimálně, tak se na každé části backlogu domlouvá dohromady. Výhoda spočívá v tom, že jak všechny tři role spolupracují současně spolu na jedné konkrétní věci, tak vzniká proces. V tomto procesu se rovnou při vývoji mohou eliminovat některé chyby, či upravit původně nepříznivé funkcionality. Poslední důležitý faktor efektivního týmu při řízení projektu je ten, že všichni členové by měli fungovat na full-time. To je z důvodu vhodné alokace lidských zdrojů na jednotlivé dílčí úkoly projektu. Všichni členové Self-organized týmu jsou zároveň i členové vývojového týmu. To znamená, že všichni členové projektového týmu se snaží docílit absolutní maximum úsilí, aby po každém sprintu mohla být dodána některá další část projektu zákazníkovi (Verheyen, 2019, s. 54-58)

1.3.1.4 Zákazník

Agilní procesní metodiky jsou také o tom, že zákazník nebo zadavatel projektu je pevnou součástí projektového týmu. To znamená, že je snaha aktivně zapojovat zákazníka do projektu. Je to z důvodu efektivnosti, určování směrování celého projektu a debatách o funkcionalitách či jiných změnách. Aby mohl být zákazník součástí projektového týmu, tak je potřeba, aby projektový tým splňoval určité vlastnosti. Musí znát sebe, svoje schopnosti, které sebekriticky dokáže obhájit, aby se nestalo, že nadnáší svoje celkové schopnosti. Dále musí znát dobře svého zákazníka, jeho vizi a potřeby a musí rozumět druhu podniká v kterém se pohybuje. Mezi projektovým týmem a zákazníkem musí panovat vzájemná důvěra, úcta a autorita. Vybudování takových vztahů je dlouhodobý proces. Těžko je možné projevovat vzájemnou důvěru na základě dvou schůzek. Postupným budováním důvěry se zákazník a projektový tým může dostat do bodu, kdy bude zákazníkovi ukazován i celkový Backlog. Tím je zobrazeno zákazníkovi, jak projektový tým pracuje, jaké má priority a případně nad těmito zjištěnými zkušenostmi s projektovým týmem komunikovat. Toto je pro zákazníka velmi pozitivní prvek, jelikož zákazník je brán jako součást týmu a má důležitou roli a funkci. Spolupráce se zákazníkem se nebere jako fixed time, fixed price model. Se zákazníkem se musí jednat na základě rámcové smlouvy, která je domluvena a schválena oběma stranami. Zadání není přesně fixováno a různé detailní funkcionality a jiné funkce se dohadují během průběhu projektového řízení (LeMay, 2019, s. 25).

1.3.1.5 Manažer a projektový manažer

Téměř poslední část projektového týmu, která je také velmi důležitá. Hlavní náplní manažera v agilním světě řízení projektu je starost o vytvoření vhodného prostředí. V tomto prostředí pak mohou projektové týmy založené na agilním řízení vhodně pracovat. Jak lze vidět, tak tuto činnost provádí i samotný Scrum Master. Manažer mu s touto činnosti právě pomáhá a úzce s ním na tom spolupracuje. Manažeři zastávají například konkrétní oblasti, ale nestarají se o jednotlivé členy projektového týmu. Manažeři a projektový manažeři tedy provádí s ohledem a komunikaci na ostatní členy týmu strategická rozhodnutí a denní činnost a práci delegují pak na jednotlivé týmy. Tím ale pravomoc klasických manažerů v agilním světě končí. Spíše by dalo říci, že v agilním prostředí již klasičtí manažeři nemají své místo. To ale neznamená, že by přechodem z klasické metodiky do agilní metodiky přišli manažeři a produktový manažeři o práci. Spíše se transformují do Scrum Masteru, Product Ownerů anebo stakeholderů, což je podpora pro projekt ve smyslu snahy pochopení potřeb zákazníků (Šochová, 2019, s. 57-58).

Tímto jsou představeny jednotlivé role projektového týmu v agilním světě. V dalších částech diplomové práce budou demonstrovány a popsány jednotlivé procesy a pojmy z agilního řízení. Některé tyto termíny již byly zmíněny u jednotlivých rolí a jejich činností. I tyto budou termíny budou nyní vysvětleny.

1.3.2 Nástroje metody Scrum

V této kapitole diplomové práce jsou popsány hlavní nástroje, které se využívají v agilní metodě Scrum.

1.3.2.1 Sprint

Jedná se o nejznámější iteraci v agilní metodice Scrum. Tato iterace je vyvozena z toho, že k ní dochází opakovaně a pravidelně. Jednodušeji řečeno, je to cyklus, ve kterém vývojový tým vytvoří a předá hotovou funkcionalitu. Tím, že se jedná o cyklus, tak se dodávají dílčí části projektu. Výhodou je, že se dodávají dle časového harmonogramu. Tento cyklus má určitou časovou náročnost, která se nemění. Jak již bylo řečeno, tak každý Sprint má dodat některou funkcionalitu. Tato funkcionalita je uvedena ve Sprint Goalu, který je vysvětlený

dále. Cyklus by měl mít určenou časovou náročnost, aby se funkcionalita ve Sprint Goalu dokázala dotáhnout do konce, jelikož po Sprintu přichází Sprint Review, což je událost, kdy se funkcionalita ukazuje zákazníkovi nebo zadavateli projektu. Tím projektový tým dostává zpětnou vazbu a názor od zákazníka, ale i od jednotlivých členů projektového týmu. Každý Sprint projektovému týmu dále přináší informace o dynamice obchodního odvětví. V obecném měřítku lze tedy říci, že čím kratší je tento cyklus, tak tím lépe. Projektový tým dostává rychlejší zpětnou vazbu a zákazník dostává častěji požadované funkcionality. Tím projektový tým získává možnost rychlejšího uzpůsobení (Šochová, 2019, s. 69).

1.3.2.2 Sprint Goal

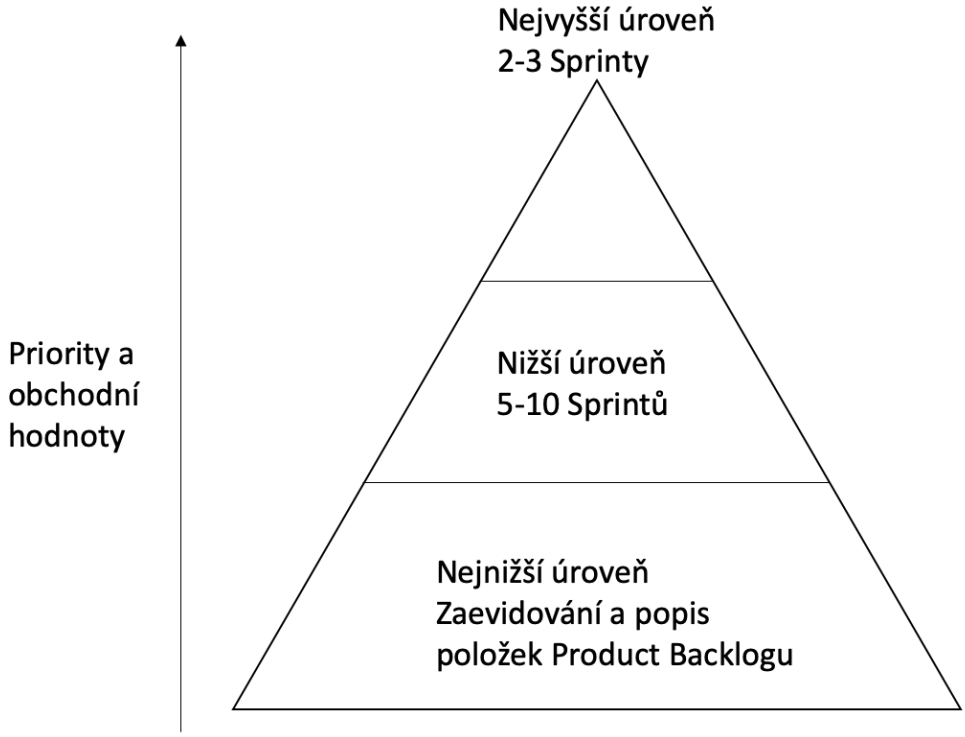
Sprint Goal je důležitá část Scrum metodiky. Sprint Goal je cíl, který projektový tým chce dosáhnout během jednoho Sprintu. Sprint Goal, který je správně nastavený, splňuje převážně zákaznické potřeby. Sprint Goal plánuje z většiny případů hlavně role Product Ownera a Development tým neboli část projektového týmu se zaměřením na vývoj. Nastavený cíl Sprint Goalu se nesmí během Sprintu již změnit. Cíle sprintu se mohou měnit na základě požadavku. Cílem může být například zvýšení finančních přehledů, zrychlení odezvy systému anebo třeba zvýšit celkový počet zákazníků, který se vrátí. Sprint Goal, který je ideálně podporován, tak je silný nástroj procesního řízení v agilní metodice (Ockerman, 2019, s. 92).

Product Owner, který chce utvořit co nejvhodnější Sprint Goal, musí umět techniku Prioritizování. Technika spočívá v tom, že se určují jednotlivé úkoly Product Backlogu, které bude projektový tým zpracovávat v konkrétním Sprintu. Správný způsob, jak určit priority jednotlivých činností projektu je, že se dotazuje zákazník nebo zadavatel projektu a zpětnou vazbou se získají informace k určení priorit. Tyto priority se následně aplikují při tvorbě Product Backlogu projektu. Činnosti, které mají vyšší prioritu, se Product Backlogu posouvají na začátek fronty a vypracovávají se dříve než věci s nižší prioritou. Hlavní důvod používaní Prioritizování činností projektu je ten, že business hodnota je vytvořena poměrem. Tento poměr je rozdělen na 80 % business hodnoty a 20 % funkcionalit. To si musí Product Owner uvědomovat a tyto věci musí dodávat a upřednostňovat v časovém harmonogramu. Další uvědomění spočívá v tom, že jestliže zákazník obdrží software, tak zhruba 65 % funkcionalit nebude používat. Tyto funkcionality by měl Product Owner odhadnout a nikdy nebo s nejnižší prioritou zapsat do Product Backlogu (Šochová, 2019, s. 63).

Činnosti zapsané v Product Backlogu se nazývají jako Product Backlog Item. Jsou to funkcionality, které se do Product Backlogu zapisují na základě User Story neboli na požadavku a komunikaci od zákazníka. Je to vždy popsáno z pohledu, aby požadované funkcionalitě rozuměl převážně zákazník, aby došlo ke správnému pochopení funkcionality a aby si zákazník mohl představit funkcionalitu v reálném světě (Šochová, 2019, s. 67).

1.3.2.3 Product Backlog

Product Backlog lze přirovnat k seznamu věcí, co je třeba v projektu udělat. Tento seznam vytváří projektový tým, manažeři a zákazník nebo zadavatel projektu. O celkovou podobu Product Backlogu se stará Product Owner. Položky v Product Backlogu jsou funkcionality, které je třeba vytvořit a které se předávají zákazníkovi nebo zadavateli projektu. Tento seznam obsahuje u položek i jejich odhad na časovou náročnost a náročnost na vytvoření. Jednotlivé odhady sestavuje Product Owner se členy projektového týmu. Product Backlog slouží převážně k tomu a dalo by se říci, že je to i jeho hlavní záměr pro jeho tvorbu. Slouží k tomu, aby se právě jednotlivé funkcionality dostávali k zákazníkovi. Větší softwarové projekty v dnešní době není možné tvořit tak, že se nejprve vytvoří kompletní celý software a až když je tento software hotový, tak se nabídne zákazníkovi. Projektový tým by získal zpětnou vazbu až v momentě prodeje, a to není ideální. Product Backlog si lze představit graficky jako takovou pyramidu. Nejvyšší úroveň tvoří uživatelské požadavky (User Stories) s nejvyšší prioritou. Tyto požadavky by měly být připraveny, správně analyticky popsány a podrobně sepsáno zadání vývojovému týmu zhruba dva až tři celé Sprinty dopředu. Na další nižší úrovni jsou méně žádané funkcionality. Tyto funkcionality ještě nemusí být úplně detailně vypracované, ale projektový tým by měl takové funkcionality rozdělit do menších částí a ty postupně zpracovávat. Takových položek se v Product Backlogu vyskytuje zhruba na pět až deset Sprintů. Další úrovně se jeví v Product Backlogu jako nejasné a neurčité. Funkcionality zde nemají vysokou prioritu a původní zadání je velmi pravděpodobné, že se ještě několikrát změní (Milošević, [2015], s. 304).



Obrázek 6 - Product Backlog pyramida priorit a položek
(Zdroj: upraveno dle Šochová, 2019, s. 72)

Každá položka v Product Backlogu by měla obsahovat údaje z následující tabulky:

ID	Unikátní identifikační číslo položky
Epic	Název či jiný popis, který projektovému týmu zajistí snadnější orientaci v Product Backlogu
Název řešitele a název zákazníka nebo zadavatele projektu	Slouží pro lepší orientaci a pozdější výstupy a statistiky, kde lze získat několik informací o jednotlivých členech projektového týmu (efektivita, …) nebo o zákazníkovi (jaké funkcionality požaduje, z jakého oboru a podobně…).
Zpětnovazební komentáře (Review Comments)	Jedná o zpětnou vazbu od projektového týmu nebo zákazníka a zadavatele projektu.
Informace o testování	Projektová část týmu, která má na starosti testování zde zapisuje své poznatky během tohoto procesu. Případně popisuje postupy k navození nedostatků či chyb, aby mohl předat nazpět (reklamovat) vývojářskému týmu, který na základě toho nedostatky a chyby opraví (fix).
Attachment	Konkrétní přílohy k dané položce Product Backlogu. Zde je možné si představit opravdu vše, co je spojené s konkrétní funkcionalitou. Je jedno, jestli se jedná o nějaký obrázek, dokument a podobné.
Komentáře	Pro kohokoli z projektového týmu je dobré mít možnost část s komentáři, kde je prostor k vyjádření názoru.
Priorita	Jak bylo popisováno výše, tak každá položka Product Backlogu má určenou svoji prioritu a na základě této priority jsou sestavovány Sprinty.

Tabulka 2 - vlastnosti položky z Product Backlogu
(Zdroj: Milošević, [2015], s. 306)

S Product Backlogem je spojen ještě Sprint Backlog. Jedná se ve stejným smyslu také o seznam. Rozdíl je v tom, že jsou v něm vypsány položky funkcionalit, které jsou třeba dodat již v konkrétním Sprintu (Milošević, [2015], s. 306).

1.3.2.4 User Story

User Story je pojem s jednodušší definicí. Tento pojem obsahuje vlastně to, co požaduje zákazník nebo zadavatel projektu od projektového týmu, aby dodal. Požaduje věci, které mu přinesou nějakou obchodní hodnotu (Business Value). Jedná se o popis od zákazníka, kdy dodává projektovému týmu svůj pohled na svět. Činitel, který požaduje vytvořit novou funkcionalitu, by měl dodat jednoduchý a lehce pochopitelný popis funkcionalit, aby projektový tým mohl provést analýzu požadavku, přiřadit požadavku prioritu a zakomponovat ho do některé úrovně Product Backlogu. User Story se dodá jako hotová, kdy splňuje požadovanou funkcionalitu, která je zakomponovaná do systému a je plně funkční a připravená k použití (Šochová, 2019, s. 77).

User Story by měla mít tyto vlastnosti („INVEST“ metoda):

Independent	Jde o to, že jednotlivé User Stories by na sobě neměly být závislé. Každé funkcionalitě se vytvoří konkrétní požadavky na funkčnost v rámci již vytvořeného projektového stavu a ten se implementuje.
Negotiable	Každá User Story má základ v diskuzi. Tato diskuze získá projektovému týmu lepší porozumění a pohled na věc a získají také detailnější specifikaci o funkcionalitě.
Valuable	Každá User Story musí mít pro zákazníka nějakou obchodní hodnotu. Pokud User Story takovou hodnotu nemá, tak zákazník ani projektový tým prakticky neví, proč takový požadavek vznikl.
Estimable	Jednotlivé User Story by měly být vlastnost ohodnotitelnosti. Tuto hodnotu určuje projektový tým. Základem toho je, že každý dokáže porozumět dané funkcionalitě.
Small	Každá User Story by měla být malá a krátká. Nejlépe, aby byla vytvořena za polovinu Sprintu. Pokud to není možné, tak se taková User Story dělí na menší části.
Testable	Jak název napovídá, tak každá User Story musí být otestovatelná. Získáváme jak funkční pohled, tak i obchodní pohled na danou funkcionalitu a určuje se, čeho se danou věcí dosáhlo.

Tabulka 3 - popis vlastností "INVEST" metody
(Zdroj: Šochová, 2019, s. 79-80).

1.3.3 Další praktiky metody Scrum

- **Epic**
  - Položky z Product Backlogu mají obsáhlejší náročnost, se formují do takzvaných Epiků. Epiky není možné je plánovat do Sprintu, jelikož se nedá

zhodnotit jejich náročnost projektovým týmem. Jednoduše jsou příliš rozsáhlé a jejich vypracování tvoří nejistotu. Jednotlivé Epiky jsou nezávislé položky Product Backlogu a Epiky se rozdrobují na další menší části (Cobb, 2015, s. 67).

- **Definition of Done**
  - Definition of Done jsou globální podmínky, které by měly splňovat jednotlivé položky z Product Backlogu. Jedná se například o to, že funkcionalita je implementována v softwaru a je schválena Product Ownerem, veškeré funkcionality prošly testem, je vytvořena dokumentace a další definované požadavky. Vzniká na základě Product Ownera a projektovým týmem dle vzájemné komunikace (Roudias, ©2015, s. 149).
- **Scrum Table**
  - Scrum Table je přehled se třemi sloupci: Sprint Backlog, In Progress, Done. Tento přehled se používá v rámci Sprintu, kdy se zobrazuje stav jednotlivých User Stories v daném sprintu. (Šochová, 2019, s. 87).
- **Kanban Table**
  - Kanban Table je fyzická tabule s papírovými kartami, které jsou napsané ručně. Je to obdoba Scrum Table, ale ve fyzické podobně. Podporuje to větší zapálení a motivaci do práce, jelikož se přesouvají napsané karty na tabuli ručně. Je zde zobrazeno, co který jednotlivec zrovna má rozpracováno a co má již hotovo (Šochová, 2019, s. 90).
- **Planning Poker**
  - Planning Poker je metoda projektového týmu, kdy se hodnotí náročnost funkcionality. Jednotliví členové týmu položí před sebe kartu s číslem a následně je ve stejný okamžik otočí. Scrum Master se následně ptá člena s nejvyšším číslem a člena s nejnižším číslem na názor, proč zvolil takovéto číslo. Po získání této zpětné vazby pro celý tým se provádí další kola. Cílem je stav, kdy se celý projektový tým shodne na číslu náročnosti. Planning Poker obsahuje karty s čísly 0,½,1,3,5,8,13,20,40,100, nekonečno, ? a karta s obrázkem kávou. Nekonečno je pro příliš složité funkcionality, otazník slouží k funkcionalitám, kdy člen týmu ještě nedokáže utvořit rozhodnutí a musí položit doplňující dotaz a karta s obrázkem kávy se používá pro pauzu

v případě časově náročného hodnocení Product Backlogu (Maximini, [2018], s. 318).

- **Velocity**
  - Velocity je metoda určení rychlosti plnění zpracování User Stories. Hodnota rychlosti je součet hodnocení jednotlivých dokončených úkolů. Jestliže se stane, že některý prvek není dokončen, tak jeho hodnota se do celkové rychlosti nepočítají. Projektové týmy, které pracují korektně, mají stabilní rychlost nebo rychlost, která je předvídatelná (Šochová, 2019, s. 103).
- **Standup a Daily Scrum meetingy**
  - Standup a Daily Scrum meetingy jsou hlavní pilíře agilních metodik. Jednoduše řečeno, je to schůzka projektového týmu na denní bázi. Prvek, který je jednoduchý na implementaci v celém řízení. Výsledkem je, aby se celý tým na chvíli zvednul od svého pracoviště a s ostatními členy se setkal například u Scrum Table. Diskutuje se o provedených činnostech a o činnostech, které následují. Případně se hovoří o vzniklých problémech. Tyto meetingy řídí Scrum Master. Je to platforma pro šíření informací (Šochová, 2019, s. 107).
- **Retrospektiva**
  - Retrospektiva je nástroj jak agilních metodik, tak i tradičních metodik. Nástroj slouží pro zisk feedbacku. Jde o to, že projektový tým se sejde u „kulatého“ stolu a diskutuje nad současným stavem a možnými změnami, které by vedly ke zlepšení stavu (Šochová, 2019, s. 111).
  - **Lepítka**
    - Používají se barevná lepítka, na které jednotliví členové projektového týmu píší, které věci by chtěli, aby zůstali bez změny a kde by se naopak měla projevit nějaká změna. Lepítka se lepí do hvězdice rozdělené pěti paprsky. Paprsky dělí prostor na pět částí. Jednotlivé části jsou nazvány Začít, Více provádět, Pokračovat, Méně provádět a Přestat provádět. Po provedení této analýzy se provede vyhodnocení (Šochová, 2019, s. 113).
  - **Časová osa**

        - V místnosti projektového týmu se vytvoří na tabuli časová osa. Jednotliví členové v průběhu Sprintu tvoří lístečky s konkrétními událostmi, které měly úspěch a které by naopak vyžadovaly zlepšení. (Šochová, 2019, s. 114).
    - **Fishbone**
        - Tento nástroj slouží k identifikaci problému a jeho vzniku. K tomu slouží otázky Co, Kdo, Kde, Proč, Kdy. Jednotliví členové týmu zde zapisují své názory na jednotlivé otázky. Poté se provede vyhodnocení. Cílem vyhodnocení je odstranit vzniklý problém (Šochová, 2019, s. 115).
    - **Loď**
        - Hravější alternativa k technice Časová osa. Nakreslí se loď s několika kotvami. Lístečky se následně lepí na kotvy, které brzdí projektový tým a zhoršují jeho efektivitu. Lístečky, které se lepí k plachtě naopak efektivitu projektového týmu zvyšují (Šochová, 2019, s. 115).
    - **ESVP**
        - Retrospektiva, která demonstruje čtyři role. Jsou jimi role Explorer, Shopper, Vacationer a Prisoner. Obrázek s těmito rolemi se nalepí ve společné místnosti a členové týmu se musí označit u jedné demonstrované role, která na ně nejvíce sedí. Explorer je proaktivní na meetinzích, hledá způsoby, učí se a chce se co nejvíce zlepšovat. Shopper je spíše posluchač, který si poslechne na meetingu veškeré nápady. Příliš proaktivní zde ale není. Vacationer je člen týmu, který meeting bere jako odpočinek od každodenní práce, ale bere ho i jako zpestření. Jeho aktivita závisí na situaci. Prisoner je člen týmu, který se cítí, že k tomu je doslova donucen a raději by se věnoval něčemu jinému (Šochová, 2019, s. 116).
- **Backlog Refinement**
    - Jedná se o činnost projektového týmu, kde se vysvětluje produkt, jeho vize. Cíl je, aby všichni členové týmu byli obeznámeni a rozuměli dané problematice. Neprovádí se meetingem, ale stylem „kdo má zrovna čas“ (Šochová, 2019, s. 119).

- **Backlog Grooming**
    - Jedná se o činnost pro ne příliš self-organized týmy. Vymezuje se nějaký časový interval pro porozumění jednotlivým prvkům v Product Backlogu (Šochová, 2019, s. 119).
- **Sprint Planning**
    - Provádí se plánování následujícího sprintu. Product Owner provede nástin Sprint Goalu. Scrum Master moderuje meeting, aby jednotlivý členové byli při plánování uvědomělí. Vytvořil se sprint, který je opřen o reálný základ a dokáže se také s nejvyšší pravděpodobností dokončit. (Šochová, 2019, s. 121).
- **Sprint Review**
    - Jedná se o zpětnou vazbu od zákazníka nebo zadavatele projektu na dodaný sprint od projektového týmu. Zpětná vazba se týká celé dodávky sprintu ne jednotlivých User Stories (Šochová, 2019, s. 125).
- **Pair Programming**
    - Jde o metodu vývojového týmu v projektu. Predispozice tohoto nástroje je, že „Více hlav, více ví“. Dva programátoři sedí u jednoho PC. K dispozici mají pouze jednu klávesnici a jednu myš. Dochází zde ke spolupráci, kdy jeden programátor myslí, kontroluje a přichází s nápady a druhý píše kód. Studie potvrzuje, že tato metoda je rychlejší, než kdyby oba programátoři měli svůj PC a každý by pracoval na svém (Šochová, 2019, s. 127).
- **Mob Programming**
    - Obdoba nástroje Pair Programming, jen s tím rozdílem, že u jednoho PC nepracují pouze dva programátoři, ale rovnou celý vývojový tým (Šochová, 2019, s. 128).
- **Code Review**
    - Metoda, kdy se zapojuje více členů do procesu. Jeden programátor napíše část kódu. Jiný programátor rovnou kontroluje zrovna napsaný kód a poskytuje zpětnou vazbu. Tato metoda má za cíl dříve odhalit vytvořené chyby a tím snižovat náklady na jejich opravy. Výhodou je, že se může provádět jak se seniorskými členy týmy, tak i s těmi juniorskými. Zároveň

juniorští členi týmu získávají více informací o projektu (Šochová, 2019, s. 128).

- **Sdílený kód**
    - Hlavní stavební kámen agilního procesu vývoje softwaru. Nikdo není vlastník napsaného programu. Každý člen vývojového týmu zapisují svoji část kódu na požadovaném místě v rámci jednoho společného kódu softwaru. Je třeba určit garanta kódu, který zodpovídá za výsledný stav kódu (Šochová, 2019, s. 128).
- **Coding Standard**
    - Jedná se o souhrn pravidel při psaní kódu, kdy se definuje, jakým způsobem se budou definovat proměnné, jaká se používá konvence. Výsledkem je přehledný kód aplikace, ve které se dokáže vyznat i programátor, který konkrétní kód nepsal (Šochová, 2019, s. 129).
- **Test Driven Development (TDD)**
    - Agilní metodika testování spočívá v tom, že tester provádí testování aplikace již v procesu vytváření nových funkcionalit. Tento nástroj má za cíl snížit časovou náročnost testů, urychlit protestování nových funkcionalit a seznámit testery s funkcionalitou a specifikami (Šochová, 2019, s. 132).
- **Refactoring**
    - Refactoring je jinými slovy optimalizace kódu aplikace. Aplikace je napsaná nějakým způsobem a splňuje požadavky na funkcionalitu. Pokud se však daná část kódu refactoringuje, tak to znamená, že se hledá vhodnější a lepší způsob, jak danou funkcionalitu naprogramovat. Jedná se o ne příliš žádanou aktivitu, ale za výsledek se bere snížení technického dluhu (Šochová, 2019, s. 133).
- **Technický dluh**
    - Stav, kdy projektový tým přestane hledat nové technologie, které by se daly použít v rámci tvořeného projektu. Veškeré složitější funkcionality se snaží napsat a ohnout tak, aby se splnilo zadání. Takovýto průběh tvoření aplikace v jednom okamžiku projektový tým dostihne a každá nová funkcionalita stojí vývojový tým více a více časové náročnosti. Ve výsledku je to díky tomu, že používaná technologie je už stará, architektura je na současné potřeby

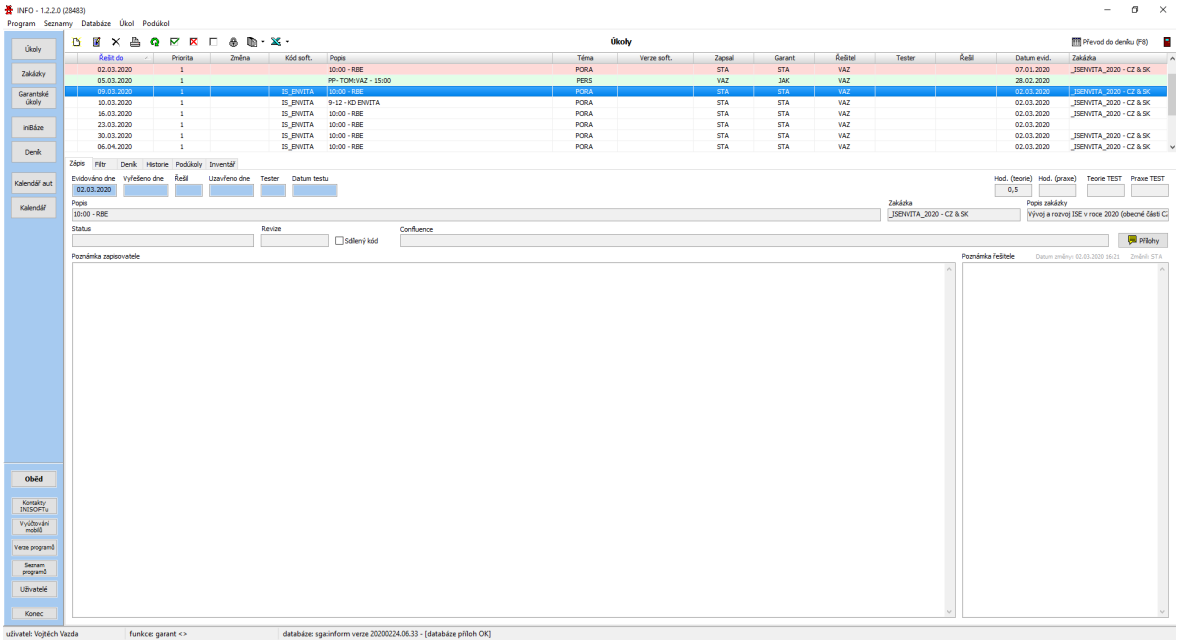
projektu špatně zanalyzovaná a projekt se dostal do stavu, který původně nebyl zamýšlen. Tomuto stavu zabraňuje právě zmíněný Refactoring (Šochová, 2019, s. 135).

2 Jednotlivé etapy vývoje

Tato kapitola se zabývá aktuálním stavem společnosti, ve které se v následujících fázích diplomové práce navrhuje optimalizace procesu řízení vývoje softwaru. V kapitole jsou charakterizovány nástroje, které firma využívala a nástroje, které firma používá v současné době. Kapitola pokračuje v detailním popisu aktuální podoby projektového řízení vývoje softwaru. Nedostatky projektového řízení jsou demonstrovány na základě teorie popsané v první kapitole diplomové práce.

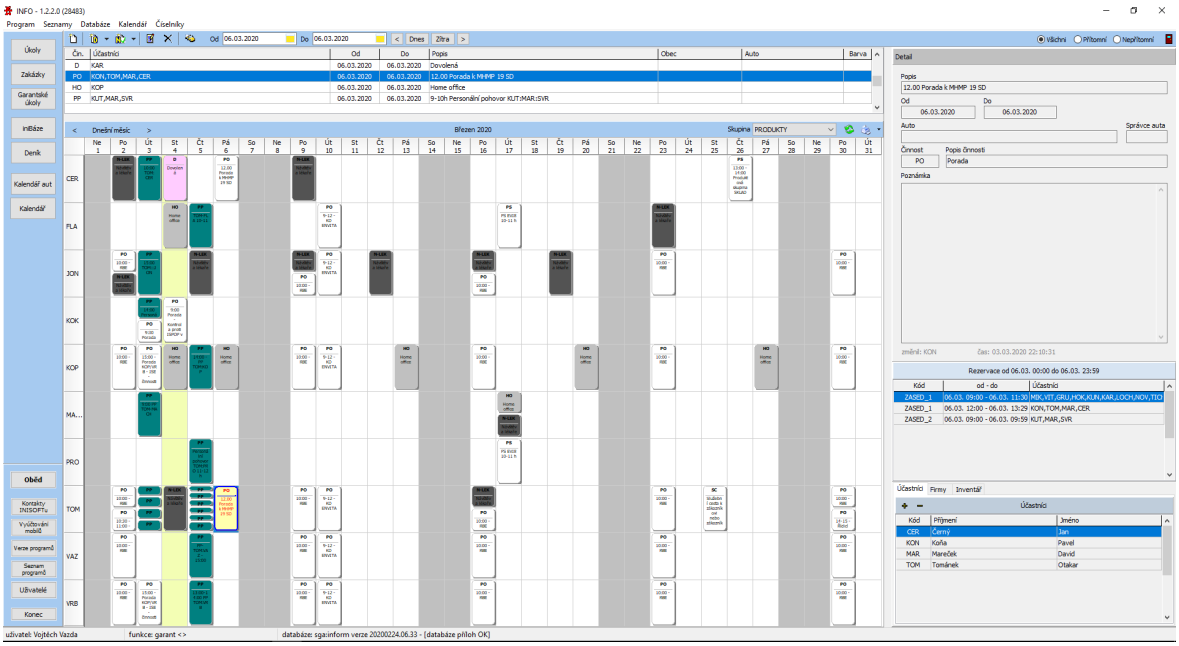
2.1 Informační systém INFO

Jedná se nástroj vytvořený přímo konkrétní společností, kterou se zabývá tato diplomová práce. INFO je informační software, ve kterém se organizovaly procesní prvky společnosti. Firma ho využívala napříč všemi odděleními pro evidenci úkolů obchodního oddělení, oddělení podpory zákazníka, oddělení poradenství a oddělení vývoje. V době, kdy se jednalo stále o malou společnost, byl informační systém INFO dostačující nástroj. Tím, jak se organizace rozvíjela a rostla, tak se začaly ukazovat nedostatky při projektovém řízení vývoje. Pro zmíněná oddělení firmy (mimo oddělení vývoje) je nadále dostačující. Členi projektového týmu nadále používají informační systém INFO pro kalendář, předávání úkolů (ne vývojových) mezi pracovníky, plánování schůzek a podobně. Firma se pokoušela informační systém INFO upravit, aby v něm bylo možné řídit vývoj softwaru. Upravily se i možnosti workflow. I po těchto optimalizačních snahách se ukázalo, že pro společnost bylo ekonomicky vhodnější nakoupit hotový software s požadovanou funkčností, než si vytvořit a upravovat vlastní od začátku. Členům projektového týmu se po přihlášení zobrazil seznam úkolů a termín pro dokončení požadované úkoly. Dále bylo v systému možno zobrazit kalendář všech uživatelů, zjistit jejich časový harmonogram a případně si jejich čas zaregistrovat pro schůzi či jiný druh meetingu. Konkrétně plánování schůzí a meetingů stále provádí v softwaru INFO.



Obrázek 7 - INFO – zásobník úkolů pracovníka
(Zdroj: převzato ze systému INFO)

Díky tomu, že SW INFO je univerzální nástroj pro všechna oddělení a nebyl určen primárně k řízení vývoje, dal se v něm v omezené míře vývoj řídit tak, že to odpovídalo spíše tradiční metodice. Hlavní nevyhovující vlastností je to, že si neumí poradit s agilními praktikami jako jsou Sprinty, třídění položek Product Backlogu na jednotlivé Epiky a podobně. INFO si dokáže poradit pouze s procesem, kdy se jednotlivé kroky projektu dotváří postupně „za sebou“ jak je tomu například ve Waterfall modelu. V současné době se INFO používá k plánování událostí v kalendáři a jako souhrnný pracovní deník. Tento software je napojen na pokročilejší podpůrné systémy JIRA a Confluence od firmy Atlassian. INFO od JIRA a Confluence získává informace o plnění úkolů, časových zápisů pracovníků a vytváření úkolů. Jedná se ale pouze o přenos informací, kde zdrojem těchto informací jsou právě systémy JIRA a Confluence.



Obrázek 8 - INFO – kalendář členů produktového týmu
(Zdroj: převzato ze systému INFO)

2.1.1 Výhody

V této podkapitolce shrnu výhody systému INFO. Velkou výhodou je přehlednost úkonů a úkolů jednotlivých pracovníků za předpokladu klasické metodiky (například Waterfall modelu). Další výhodou je to, že to autentický systém vyrobený přímo firmou, tudíž i implementace případných oprav a úprav není tak časově náročná. Poslední hlavní výhodou tohoto systému je variabilita systému v podobě nastavení jednotlivých událostí, úkolů a podobně.

2.1.2 Nevýhody

Hlavní nevýhodou tohoto systému, kvůli kterému se od něj i opustilo, je to, že nedostačuje již potřebám pro agilní metodiku při projektovém řízení vývoje softwaru. Jelikož se jedná o starý software (20 let), který se stála záplatuje a nabalují další věci, které ve výsledku mohou působit protichůdně. V takto zastaralém SW může jedna změna způsobit i nežádoucí dopady tam, kde to nikdo nečeká (a nezřídka se tak děje). Přepracování celé architektury informačního systému INFO k tomu, aby dostačovala agilním metodě by, stála firmu velké

časové úsilí a finanční náklady. Finanční náklady a časové úsilí by byly tak vysoké, že se společnost rozhodla přejít pod systém JIRA a Confluence od jiné společnosti Atlassian, která se o tuto problematiku zabývá.

Další nevýhodou shledávám to, že se nejedná o webovou službu, ale pouze WIN aplikaci. Tudíž je možné v systému pracovat pouze za předpokladu připojení k firemní síti. To bývá občas frustrující pro členy projektového týmu, kteří jezdí například po zákaznících a aby se mohli přihlásit do INFO systému, tak musí používat VPN síť.

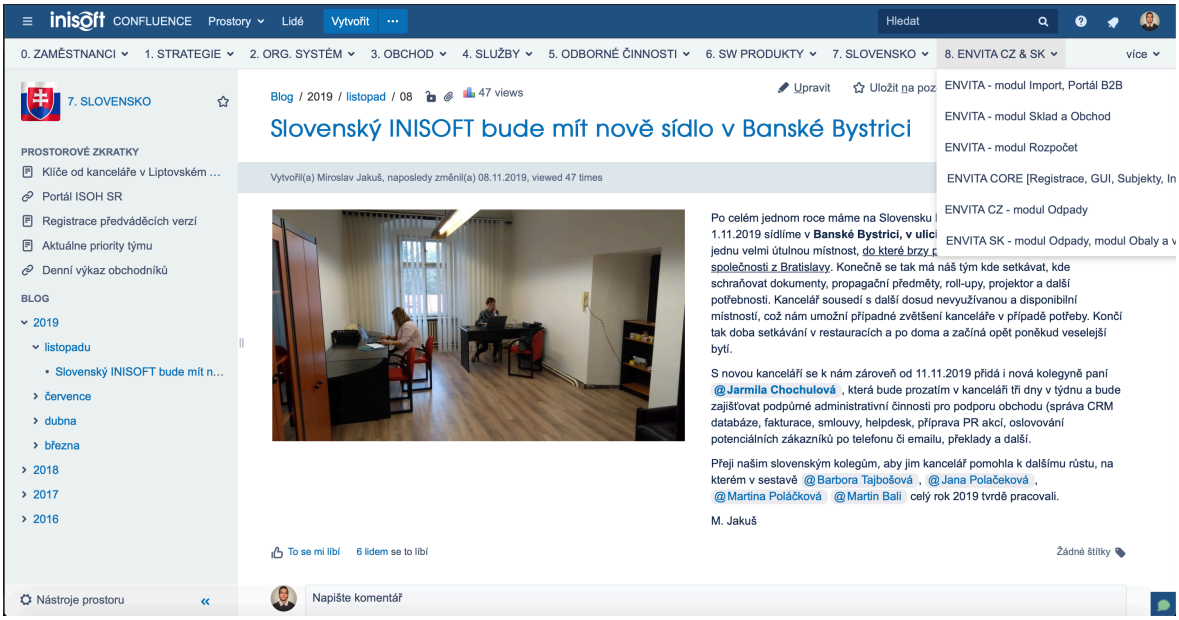
2.2 Proprietární software Confluence pro spolupráci na dokumentech

Jedná se o systém vytvořený firmou Atlassian, který slouží jako podpora projektového řízení společnosti v oblasti komunikace. Podporuje se vnitrofiremní komunikace, business komunikaci a komunikaci společnosti a jejími partnery.

Tento systém funguje jako webová služba, tudíž je možné se k ní dostat odkudkoliv, kde je pracovní stanice s připojením k internetu. Každý uživatel se přihlašuje pomocí svého přihlašovacího jména a hesla. Po přihlášení se uživatel objeví na úvodní obrazovce, kde jsou zobrazeny hlavní zprávy společnosti, o kterých by měly vědět všichni pracovníci. Dále se zde nachází menu. Pomocí menu se uživatel dostane do různých prostorů zabývající se různými oblastmi společnosti. Vezme-li se příklad, že uživatel potřebuje řešit problém ohledně vyvíjeného produktu se strategií na Slovensku, tak musí najít konkrétní produkt, kterého se problém týká a v tomto prostoru bude podprostor zaměřující se přímo na Slovensko.

Jednotlivé prostory jsou důkladně členěny a tříděny. Pomáhá to k přehlednosti jednotlivých témat, že se vyskytují na místech, na kterých by uživatelé čekali. Zároveň se toto členění osvědčuje, protože uživatelé, kteří danou problematiku mají na starosti, vidí pouze potřebné informace. To pomáhá k udržení pozornosti na oblast, kterou člen projektového týmu opravdu řeší specifickým požadavkem a neodvádí ho to do jiných prostorů. Navíc to slouží i uživatelům, kteří se na problematice třeba přímo neúčastní, ale mohou se jako vzdálení konzultanti k problému vyjádřit. K tomu slouží sekce s komentáři pod každým prostorem, článkem, blogem a vláknem. V tomto systému je tato sekce velmi propracovaná, protože

podporuje odkazy jak na uživatele, tak na jiné prostory, kam se uživatel může jedním klikem přes odkaz dostat. Navíc označeným uživatelům chodí upozornění přímo na email, tudíž nemůže nastat situace, že se řeší důležitá problematika a osoba, která je řešitelem se o tom nedozví.



Obrázek 9 - ukázka prostředí Confluence
(zdroj: převzato ze systému Confluence)

Na obrázku číslo devět je vidět konkrétní prostředí z Confluence týkající se tématu „Slovensko“. Stromovou strukturou přes sekci „2019“ a „listopad“ se uživatel dostal ke zprávě, že v tomto období se společnosti podařilo získat nové kancelářské prostory v Banské Bystrici. Zároveň je zde hezky vidět označování uživatelů v textu, kdy označení uživatelé jsou se znakem zavináče („@“) modře zvýraznění. Všem těmto uživatelům přišlo upozornění na email s odkazem na tento článek. Dále je zde vidět sekce komentářů i s rychlou zpětnou vazbu „Uživateli se to líbí“ neboli „lajkem“. Jedná se o obdobnou funkcionalitu, jakou používají sociální sítě (například Facebook). V horní listě je vidět, které další hlavní prostory se v systému nachází a kam může uživatel následně přejít. Jedná se o hlavní prostor, který se poté stromově člení až po nejnižší úroveň a cílový článek.

2.2.1 Výhody

Velkou výhodou tohoto řešení je webový přístup. Tím pádem můžou členi projektového týmu využívat systém odkudkoliv jen díky přihlašovacímu jménu, heslu a internetovému připojení. V tom je shledána výhoda oproti předešlému systému INFO.

Další výhody jsou propracovanost prostorů, oblastí, článků a okamžitou možností zpětné vazby a odkazováním se na jednotlivé prvky systému (uživatele, články, a podobně). Co se týče komunikativní úrovně, tak se jedná o velmi propracovaný systém.

2.2.2 Nevýhody

Ač je „označování členů projektového týmu“ zmíněno jako výhoda, tak se to může stát i nevýhodou. Jedná se o to, že pokud je člen projektového týmu veden v několika prostorech a má na starost několik okruhů komunikace, ostatní uživatelé ho mohou několikrát označit v článku, v komentáři. Na každé jedno označení přijde uživateli upozornění do emailu. Může se stát to, že takto vytíženému členu projektového týmu, který je například týden na dovolené, že se přihlásí ke svému pracovnímu emailu a objeví se mu zde opravdu velké množství nových emailů vzniklých na základě označení.

Další nevýhodou je to, že pokud se uživatelé nestarají o správu jednotlivých prostorů, tak zde může vzniknout nelogičnost a neuspořádanost. Toto pak neguje hlavní výhodu systému. Nesprávou systému může vzniknout znět neuspořádaných prostorů a článků, které spolu nijak nesouvisí a uživatelé se v něm nevyznají. Při neuváženém zapisování všech informací, a často velmi nepodstatných, systém bobtná, informací je mnoho, špatné se dohledávají ty důležité, a může dojít k zahlcení systému a vzniku nepřehlednosti. Jsou lidé, kteří si pletou systém s literární platformou, sepisují zbytečně obsáhlé články, které mají nízkou nebo žádnou informační hodnotu. Dělají to především proto, že při zběžném pohledu a bez detailního studia to vypadá, že vyvíjejí prospěšnou aktivitu. Pro ostatní členy týmu to znamená přečíst nepodstatný článek, protože byli označení v textu nebo je článek v prostoru, který sledují a tím pádem jim chodím informační email i bez přímého označení.

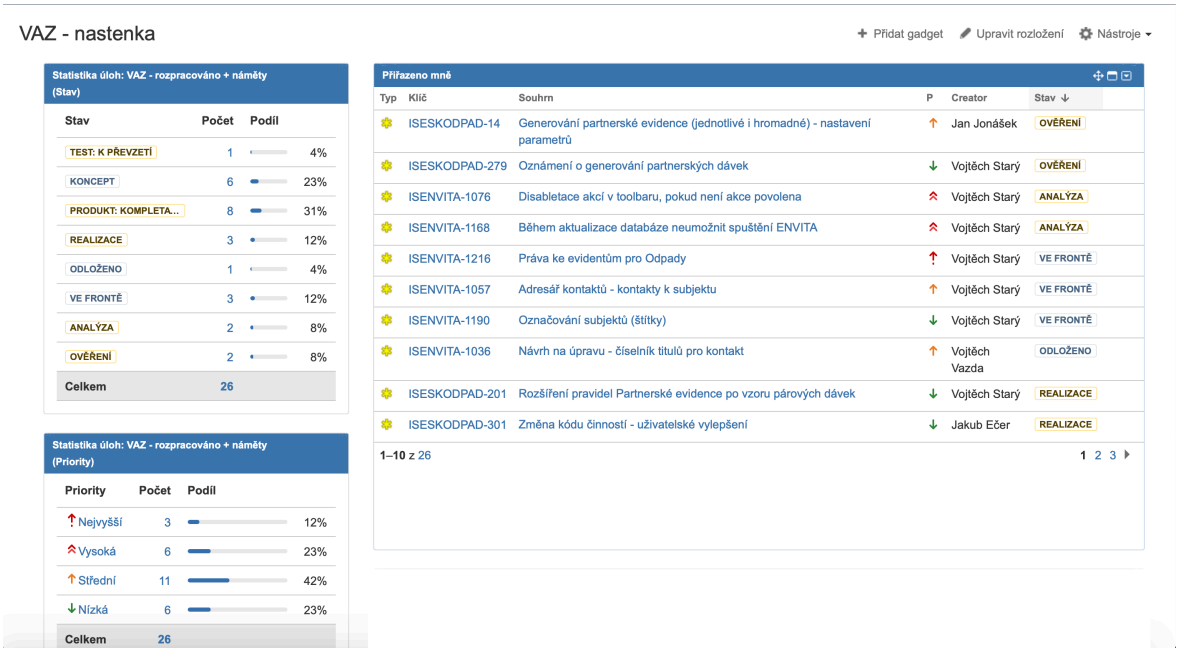
2.3 Vývojové prostředí JIRA pro sledování projektů

Další podpůrný systém od firmy Atlassian a druhý systém používaný společností, kterou se zabývá tato diplomová práce. Funkce tohoto systému spočívá v podpoře vývojového cyklu při tvorbě softwaru. Je zde podpora různých agilních nástrojů a praktik. Jedná se například o Product Backlog, Sprint Backlog, Sprint, Epic a podobně. Dále jsou zde zobrazeny jednotlivé kroky při vývoji různých funkcionalit. V tomto systému spolu komunikují převážně analytici, produktový manažeři, vývojáři a testeři, obchodní oddělení, oddělení podpory, a všichni další, co se na projektu, jakkoliv podílejí. Do JIRA může mít přístup i zákazník, a vidět aktuální stav. Přístup lze omezit právy a je tak možné některé interní informace zachovat před zákazníkem skryty.

V tomto systému se spravuje celý vývojový cyklus softwaru a funkcionalit softwaru. Od sepsání námětu od zákazníka přes analýzu námětu s diskuzí vývojového týmu, sepsání požadavku na vývoj a zpětné vazby od oddělení vývoje, test funkcionality v Epicu, Merge funkcionality do hlavní vývojové větve a otestování funkcionality v hlavní vývojové větvi. Obsáhlejší analýzy funkcionalit či diskuzí vývojového týmu probíhají v systému Confluence. Možná se nyní zdá, že vedení těchto prvků v druhém systému je zbytečné až nevhodné. Opak je ale pravdou, protože v obou systémech existuje možnost propojení a tvoření odkazů jak na články, tak i na jednotlivé členy. Systémy JIRA a Confluence spolu vzájemně spolupracují a vzájemně se dají propojit makry. Dokumentace, popis funkčností a jiný dokument vznikne v Confluence a z JIRA je možné se na původní dokument v Confluence odkázat, anebo naopak. Praktický příklad je, že produktový tým obdrží požadavek na funkcionalitu do vyvíjeného softwaru. Celý požadavek od zákazníka se zaeviduje jako námět. Oddělení produktu zpracuje analýzu námětu v Confluence, kde se analýza zpracuje s patřičnou přehledností. Tato přehlednost je zachována i po připomínkování a vyjádření celého produktového oddělení. Jakmile je analýza kompletní a odsouhlasena jak produktovým oddělením, tak i oddělením vývoje a i zákazníkem, tak se uvede odkaz v námětu a námět se označí ve stavu „Analýza“. K námětu se vytvoří nový úkol, kde se píše již konkrétní zadání na vývojářský tým. Tento úkol se nazývá jako realizační a propojí se s úkolem námětu. Námět se označí do stavu „Realizace“. Nyní si celý realizační úkol projde cyklem. Konkrétním cyklům se věnuje diplomová práce v kapitole 2.4. Jakmile dojde k jeho Mergi do hlavní větve programu, tak se námět označí jako

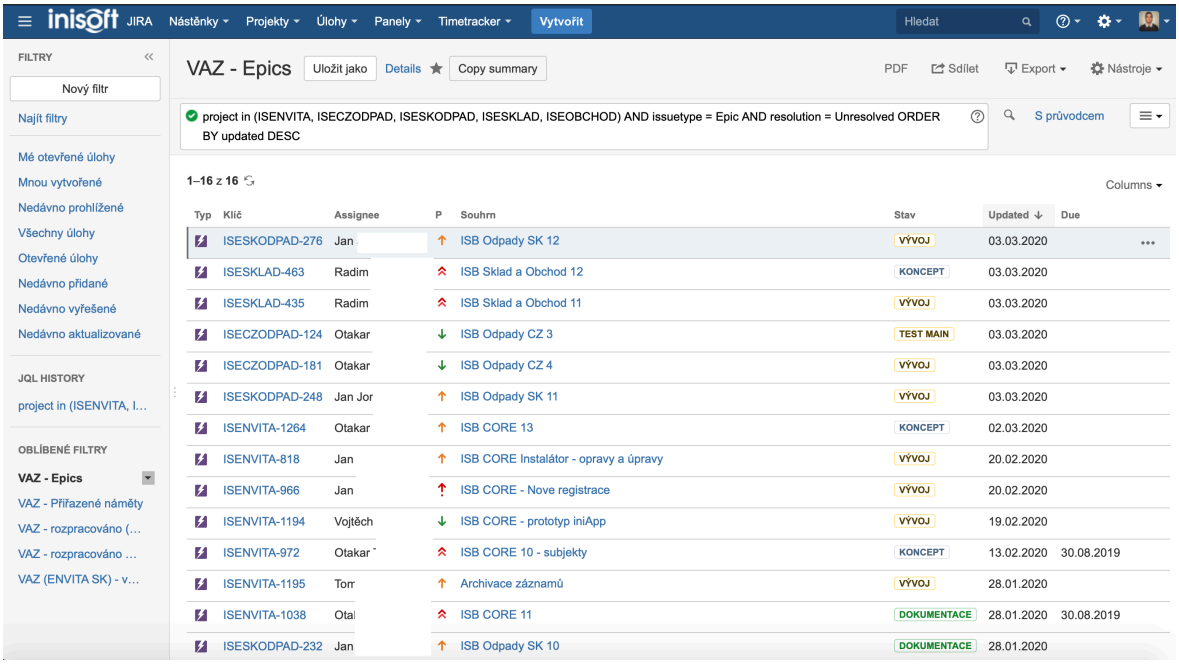
„Pilotáž“, kdy zákazník obdrží první iteraci funkcionality. Pokud by zákazník poskytl zpětnou vazbou požadavky na nějaké dodělávky nebo nedostatky, tak se provedou. Pokud ne, tak se celý námět přesune do stavu „Hotovo“.

Jedná se o jednodušší příklad vývojového cyklu funkcionality, který je však velmi četný. Na následujících obrázcích bude zobrazeno reálné prostředí systému s konkrétními pohledy na různé části projektů.



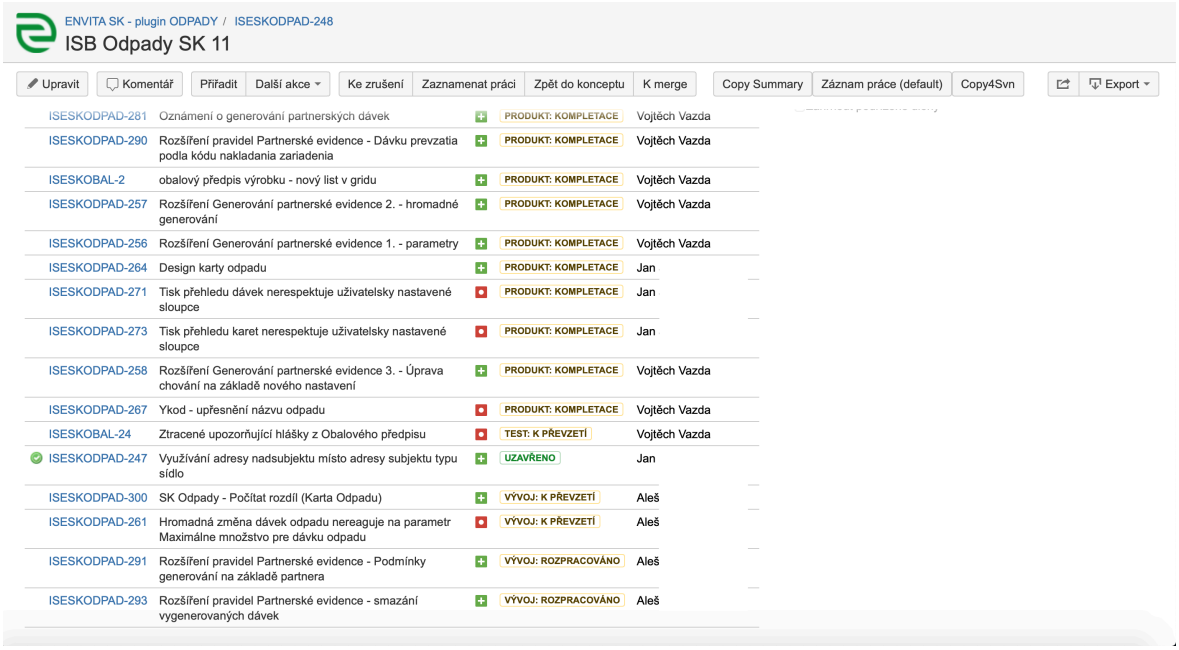
Obrázek 10 - úvodní pohled jednotlivého člena produktového týmu v nástroji JIRA (zdroj: převzato ze systému JIRA)

Na obrázku číslo deset je vidět jeden z možných pohledů (nástěnku lze libovolně přizpůsobit a dát na ni požadované informace daného uživatele), kdy se uživatel a člen produktového týmu přihlásí do nástroje JIRA. Přihlášení probíhá na základě přiděleného uživatelského jména a hesla. Na obrazovce jsou zobrazeny úkoly, které má řešit právě konkrétní člen týmu. U jednotlivých úkolu jsou určeny jejich priority, jejich stav, kdo je vytvořil a zda se jedná o námět, Epic nebo realizační úkol. Samozřejmě jsou tyto údaje volitelné a uživatelé si zde mohou navolit přesně ty informace, které potřebuje. V levé části obrazovky jsou pak zobrazeny počty úkolů rozdělené na základě jejich stavů a na základě jejich priorit. Opět jsou tyto údaje volitelné a člen projektového týmu si sám určuje, jaké údaje zde chce mít zobrazeny a jaké jsou pro něj irelevantní.



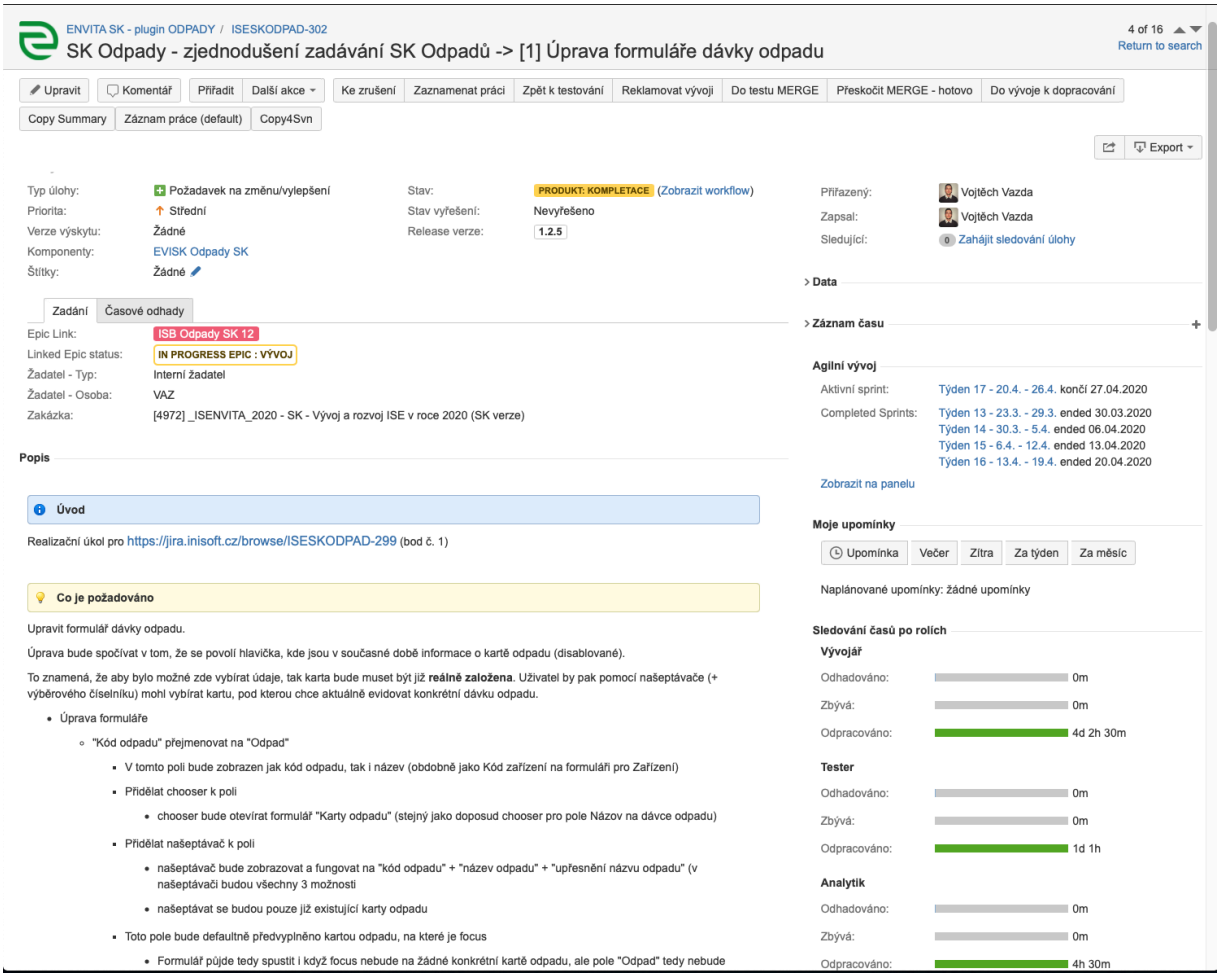
Obrázek 11 - seznam rozpracovaných Epics
(zdroj: převzato ze systému JIRA)

Na obrázku číslo jedenáct je zobrazen seznam aktivních Epiců. Dále jsou zde údaje o tvůrci Epicu, jeho priorita, název, stav a datum, kdy se naposled Epic aktualizoval. Na informaci ve sloupci „Stav“ je zobrazeno, který Epic je zrovna ve vývoji (Stav = „Vývoj“), který je téměř na konci vývojového cyklu a funkcionality se testují již v hlavní větvi softwaru (Stav = „Test Main), které jsou již hotovy a tvoří se již uživatelská dokumentace (Stav = „Dokumentace“). Dále jsou zde Epiky, které stále čekají na startu vývojového cyklu (Stav = „Koncept“).



Obrázek 12 - Product Backlog konkrétního Epicu
(zdroj: převzato ze systému JIRA)

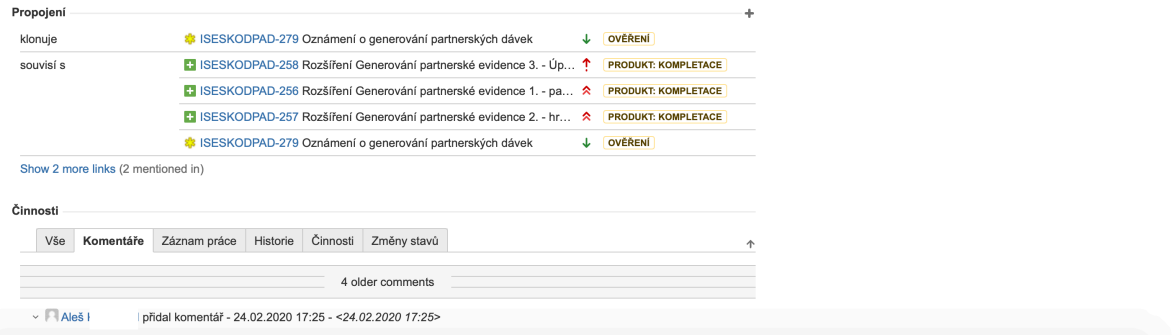
Na obrázku číslo dvanáct je zobrazen konkrétní příklad Epicu. Jedná se o Epic s názvem „ISB Odpady SK 11“. Na hlavní liště jsou tlačítka, které mohou změnit stav Epicu. Pod tlačítky je pak vyobrazen celý Product Backlog tohoto konkrétního Epicu. Úkoly jsou označeny jednoznačným a unikátním identifikátorem, který se používá pro propojení s jinými úkoly. Dále je zde zobrazen název úkolu, jeho priorita, stav a uživatel, který úkol aktuálně řeší.



Obrázek 13 - ukázka konkrétního realizačního úkolu
(zdroj: převzato ze systému JIRA)

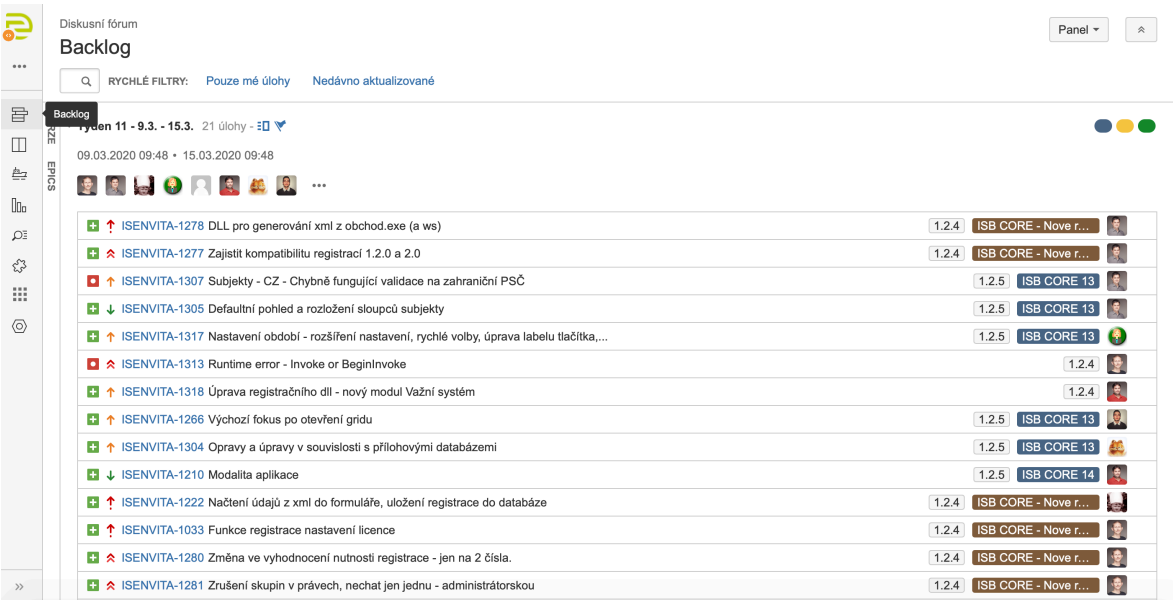
Na obrázku číslo třináct je zobrazen konkrétní realizační úkol. V horní části se nachází název úkolu a pod názvem jsou na liště tlačítka, které mění tomuto úkolu jeho stav. Pod tlačítky se zobrazují důležité informace o úkolu jako jeho priorita, stav, komponenty, požadovaná verze hlavní větve aplikace, aktuální Epic a jeho stav. Dále jsou zde informace o uživateli, který úkol vytvořil, uživateli, který úkol aktuálně řeší a záznamech času podle rolí, který byl v jednotlivých fázích řešení úkolu vynaložen. Dále je zde zobrazeno, v jakém je úkol týdenním Sprintu. Pokud úkol není vyřešen v rámci plánovaného Sprintu, může být posunut do dalšího, a zůstane zde historie Sprintů, ve kterých byl zařazen. Po těchto informacích následuje členěný text, kde je popsán požadavek na vývojový tým.

Úkol s námětem je řešen obdobně jen s tím rozdílem, že se u něj vyskytuje méně informací. Informace navíc, která se u něj vyskytuje, je pouze zákazník, který funkcionalitu požaduje.

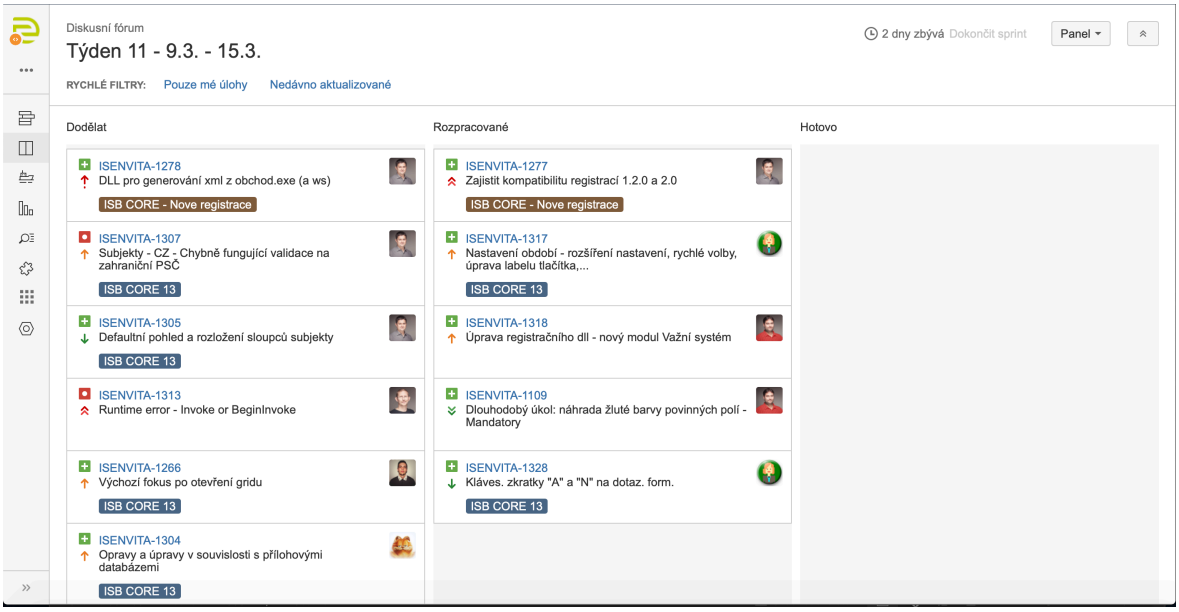


Obrázek 14 - propojení úkolů a komunikace produktového týmu v sekci komentáře (zdroj: převzato ze systému JIRA)

Na obrázku číslo čtrnáct je zobrazeno konkrétní propojení jednotlivých úkolů. Propojení úkolů pomáhá k přehlednosti v rámci systému a uživateli pomáhá vyhledat požadované úkoly a náměty, které s konkrétním úkolem souvisí. Zároveň se v tomto propojení mohou vyskytovat odkazy do systému Confluence, kde může být popsána důkladná analýza problematiky. V dolní části jsou pak zobrazeny reakce jednotlivých členů produktového týmu.



Obrázek 15 - Product Backlog v JIRA (Zdroj: převzato ze systému JIRA)



Obrázek 16 - týdenní Sprint v JIRA
(Zdroj: převzato ze systému JIRA)

Na obrázcích číslo patnáct a šestnáct je poté vidět vizuální zobrazení Product Backlogu a plánovaného týdenního Sprintu v systému JIRA.

2.3.1 Výhody

Systém je opravdovým pomocníkem při podpoře agilní metodiky v produktovém vývoji softwaru. Jeho velkou výhodou je převážně možnost customizace podle potřeb jednotlivých členů projektového týmu. Dále je nedílnou výhodou systému právě podpora agilních metodik, kde je možné zaznamenávat Epiky, Sprinty, Product Backlogy, Sprint Backlogy a jiné. Dalšími menšími výhodami jsou například možnosti propojení jednotlivých úkolů a Epiců a jednotlivých členů projektového týmu.

Z vlastní zkušenosti při práci v původním systému INFO a nyní v JIRA je možné tvrdit, že pro agilní metodiku je INFO zcela nepoužitelné. Proto vnímám přechod společnosti na systém JIRA jako racionální. Stejně to cítila většina členů projektového týmu a dlouho volali po takovém profesionálním nástroji.

2.3.2 Nevýhody

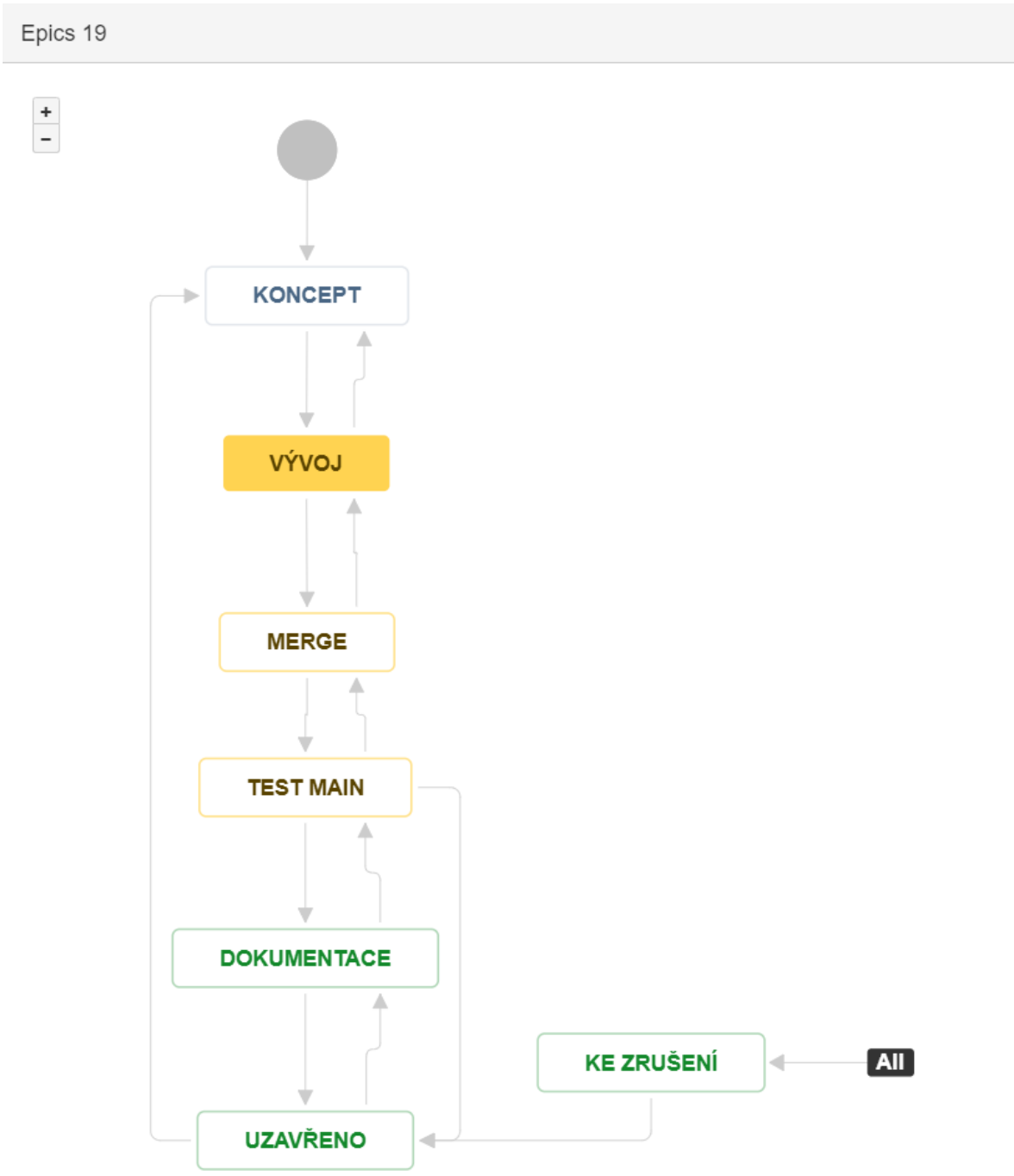
Velkou nevýhodu systému je možné určit hlavně pro nové členy projektového týmu. Systém JIRA nabízí opravdu mnoho funkcí a funkcionalit, možností pohledů a nastavení pohledů, že novému členu to může zpočátku přijít zmatečné. I přes školení na práci se systémem JIRA nejdříve uživatel věnuje mnoho času převážně tomu, aby se v systému orientoval a aby si ho přizpůsobil svým potřebám a své činnosti, ve které bude v rámci projektového týmu působit.

2.4 Aktuální vývojový cyklus vývoje softwaru

V této kapitole je představen aktuální stav vývojového cyklu softwaru ve společnosti, na kterou se zaměřuje tato diplomová práce. V současné době společnost využívá tři druhy vývojových cyklů. Cykly jsou vytvořené pro „Epic“, „Námět (od zákazníků nebo interní)“ anebo „Požadavek na úpravu programu anebo pro případné chyby“. Rozdíl mezi námětem a návrhem na úpravu programu je, že námět je pro funkcionalitu, která ještě neabsolvovala úvodní analýzu. Návrh na úpravu proces úvodní analýzy již absolvoval a jedná se už o přesné zadání oddělení vývoji.

2.4.1 Vývojový cyklus pro Epic

Na obrázku číslo sedmnáct je znázorněn vývojový cyklus pro Epic. Počátek celého cyklu zobrazuje šedý puntík, který pokračuje krokem procesu „Koncept“. Dalšími stavy jsou „Vývoj“, „Merge“, „Test Main“, „Dokumentace“, „Uzavřeno“ a „Ke zrušení“. Průchod mezi jednotlivými kroky znázorňují šipky. Vždy je možné posunout Epic pouze na následující krok vývojového cyklu anebo reklamovat zpět na právě předcházející. Je to z důvodu udržitelnosti systému a přehlednosti při zpětné kontrole Product Ownera. V následujících podkapitolách je popsáno, jaké činnosti se v jednotlivých krocích vývojového cyklu provádí.



Obrázek 17 - vývojový cyklus pro Epic
(Zdroj: převzato a upraveno z interní dokumentace společnosti)

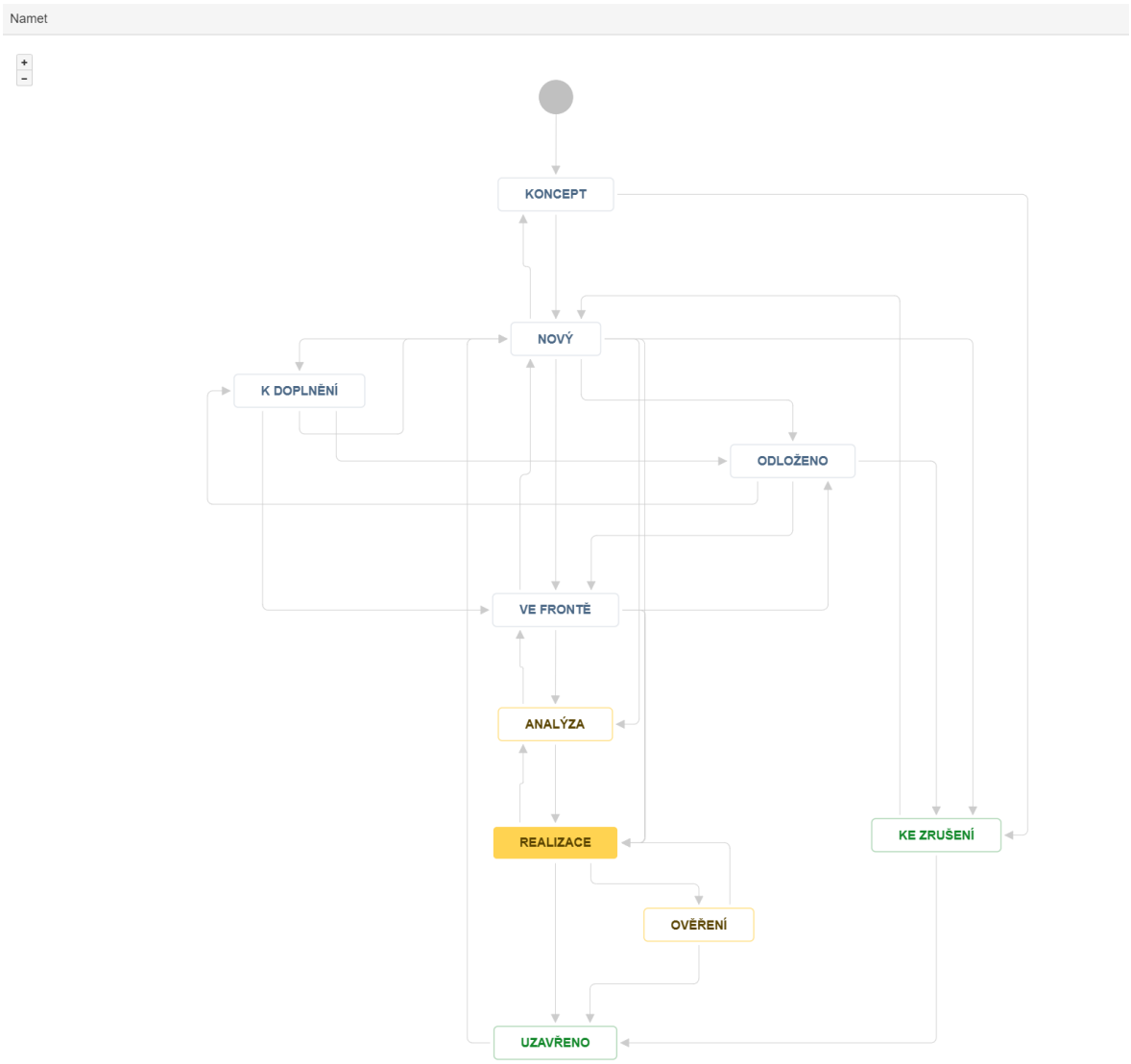
- Koncept
    - V kroku „Koncept“ se provádí úvodní zaznamenání a založení Epicu. V tomto stavu se nachází Epic v momentě, kdy ho člen produktového týmu založí. Jakmile je Epic založen, tak členové projektového týmu po konzultaci plní Epic úkoly z Product Backlogu. Plnění Epicu úkoly z Product Backlogu je ukončeno, jakmile obsahuje maximální časovou náročnost nebo požadované zadání vývoji pro první iteraci dané funkcionality. V tomto momentě se Epic posouvá do stavu „Vývoj“.
- Vývoj
    - Jak už název napovídá, tak se jedná o stav Epicu, kdy je jako celek předán oddělení vývoje. Vývojový tým na základě priorit a konzultace sestaví týdenní Sprint Backlog. V týdenním Sprintu se provádí vypracování úkolů ze Sprint Backlogu. Po zpracování a otestování jednotlivých úkolů z Epicu je vývoj pro Epic uzavřen a následuje další krok „Merge“.
- Merge
    - Merge je stav, kdy projektový tým rozhodne, že Epic je připraven, aby se implementoval do hlavní vývojové větve softwaru. Tento pokyn obdrží oddělení vývoje od Product Ownera, aby nově vytvořený kód aplikace z Epicu naimplementoval do stabilní verze aplikace. Zde může nastat několik problémů, například když se provádí merge více Epiců po sobě (současně to není možné), může vznikat konflikt na místě kódu, který je upravován zároveň ve více Epicích, nebo může být předchozím mergem upravena struktura databáze, a tak neodpovídá tomu, s čím pracuje mergovaný Epic. Po vyřešení všech konfliktů a naimplementování nových funkcionalit do hlavní větve programu je Merge úspěšně dokončen.
- Test Main
    - Epic je zmergován do hlavní větve softwaru. Nyní je na základě dohody produktového týmu a Product Ownera proveden přesun celého Epicu do kroku “Test Main“. Jedná se o pokyn k protestování celé hlavní větve aplikace, zda se díky Mergi neobjevily nějaké chyby. Tyto nově vzniklé chyby je třeba objevit a reportovat pro oddělení vývoje. Oddělení vývoje provádí opravy již v hlavní vývojové větvi aplikace. Jakmile je celá aplikace

otestována a další chyby se neobjeví, tak se pokračuje ke kroku „Dokumentace“.

- Dokumentace
    - Dokumentace je stav, kdy je hlavní stabilní větev aplikace obohacena o nové funkcionality vzniklé na základě Epicu. Aplikace je stabilní, otestovaná a bez objevených chyb. Oddělení podpory dostává pokyn pro tvorbu uživatelské dokumentace, která se konzultuje s oddělením produktového týmu. Dokumentace se po vytvoření publikuje na webu společnosti v prostoru „Centrum Informací“, což je veřejná část Confluence přístupná zákazníkům.
- Uzavření
    - Proces, kdy produktový tým zkontroluje a zajistí, že daný Epic je ve všech krocích úspěšně splněn. Jakmile se toto úspěšné ukončení potvrdí, tak se Epic uzavírá v kroku „Uzavření“.
- Ke zrušení
    - Jedná se o speciální případ, který není úplně zakomponovaný v celém vývojovém procesu pro Epic. „Ke zrušení“ znamená, že produktový tým rozhodne, že se daná oblast funkcionalit nebude realizovat. Produktový tým dá pokyn, že se celý Epic přesune do stavu „Ke zrušení“. Do tohoto stavu se může dostat z jakéhokoliv jiného, pokud se pro to projektový tým rozhodne. Následným krokem je „Uzavření“, ale s tím, že daný Epic nebyl realizován.

2.4.2 Vývojový cyklus pro Námět

Na obrázku číslo osmnáct je znázorněn vývojový proces pro náměty funkcionalit od zákazníků, členů týmu produktu nebo jiných uživatelů. Jedná se tedy spíše o konzultační část mezi oddělením obchodu a zákazníkem a následně analytickou část mezi odděleními obchodu, produktu a vývoje. V části projektu „Námět“ nedochází k žádné faktické změně softwaru. Dochází zde k analýze funkcionalit a rozhodnutí, zda se budou realizovat nebo nikoliv. V typu procesu „Námět“ se objevují jako procesní kroky „Koncept“, „Nový“, „Ve frontě“, „K doplnění“, „Odloženo“, „Analýza“, „Realizace“, „Ověření“, „Uzavřeno“ a „Ke zrušení“. Jednotlivé kroky jsou popsány v následujících podkapitolách.



Obrázek 18 - vývojový cyklus pro Námět
(Zdroj: převzato a upraveno z interní dokumentace společnosti)

- Koncept
    - Koncept je při druhu úlohy „Námět“ prvním procesní krok. Pokud projektový tým založí úlohu typu „Námět“, tak při uložení úlohy začíná vývojový cyklus na stavu „Koncept“. V konceptu se zaznamenávají úvodní myšlenky o nové funkcionalitě, případně citace od zákazníka a odkazy na legislativu. Jakmile projektový tým rozhodne, že se funkcionalita bude řešit dále, tak se námět posouvá k dalšímu kroku.
- Nový

- V procesním kroku „Nový“ je daný námět popsán požadavkem od zákazníka, který je podložen například na základě legislativy. Tím, že se software vyrábí jak pro českou legislativu odpadového hospodářství, tak i pro slovenskou legislativu odpadového hospodářství, tak je vhodné tyto informace získávat. V tomto kroku projektový tým konzultuje funkcionalitu, přiřazuje se konkrétní řešitel neboli produktový manažer a stanovuje se časový harmonogram a verze, ve které se námět objeví v softwaru.

- Ve frontě
  - Jestliže se rozhodne projektový tým, že požadovaný návrh je možné splnit až k nějakému vzdálenějšímu termínu, tak se námět posouvá ke stavu „Ve frontě“. V tomto stavu námět sečkává, dokud projektový tým na základě diskuze neurčí, že už je čas danou funkcionalitu z námětu začít tvořit. Na základě rozhodnutí projektového týmu o zařazení námětu k vývoji se posouvá ke stavu „Analýza“.
- K doplnění
  - Jedná o podpůrný krok projektového řízení úlohy „Námět“. Používá se v případě, kdy zákazník předá požadavek na výrobu funkcionality. Může se však stát, že požadavek není podložen legislativou země, nebo je zadán s nedostatečným množstvím potřebných informací. V tomto případě se námět dostává na stav „K doplnění“, kde řešitel musí dohledat informace k funkcionalitě, spojit se se zadavatelem, případně se spojit s poradcem legislativy. Na základě všech získaných informací, které se k námětu získají a doplní se do úlohy, tak vzniká mezi produktovým týmem nová diskuze ohledně zahrnutí do vývoje a časový termín.
- Odloženo
  - Stav „Odloženo“ dostávají úlohy, které na základě diskuze produktového týmu není třeba v aktuálním časovém úseku řešit a investovat pro ně výrobní kapacity. Námět je kompletní, ale je odložen na pozdější vypracování. Předběžně se takovému námětu vyplní některá z budoucích verzí softwaru. Jakmile se po nějaké době vývoj posune blíže k této verzi, tak produktový tým znovu jedná nad tímto námětem a rozhoduje se buď o dalším odložení úlohy nebo o naplánování úlohy do vývoje.

- Analýza
    - Do stavu „Analýza“ se úloha typu „Námět“ dostane, pokud se projektový tým rozhodl, že už se bude implementovat. Požadavek je podložen legislativou a je zkonzultován se zadavatelem, že námět takto odpovídá jeho představám. Řešitel, který obdrží tento námět k řešení, musí provést hloubkovou analýzu, aby zjistil technické možnosti. Novou funkcionalitu musí zkonzultovat s oddělením vývoje, s ostatními produktovými manažery a poté tuto zpětnou vazbu zpracovat. Na základě všech těchto informací provádí analýzu a navrhuje se několik variant, jak požadovanou funkcionalitu do softwaru implementovat. Návrh se provádí tak, aby byl námět co nejpřehlednější a co nejprůhlednější pro ostatní členy projektového týmu (UI, vývojové diagramy, detailní popis jednotlivých funkcí atd.). Tato hloubková analýza se provádí do systému Confluence a pomocí odkazů se propojuje s námětem.
    - Jakmile je vytvořen jeden až několik možných zpracování námětů dané funkcionality do aplikace, tak se svolá meeting, kde produktový manažeři a Product Owner jednotlivé náměty konzultují. Tyto meetingy jsou časově náročně podle náročnosti funkcionality. Prakticky to znamená, že některé náměty se pouze odsouhlasí a některé se konzultují a „napadají“, aby vzniklo nejvíce vhodné řešení, se kterým bude souhlasit co možná nejvíce členů projektového týmu. Pokud je námět odsouhlasen, tak se posouvá na krok „Realizace“. Jestliže je námět „napaden“ ostatními členy produktového týmu nebo Product Ownerem, tak se na základě této zpětné vazby námět upraví a poté se posouvá na krok „Realizace“.
- Realizace
    - Produktový manažer, který získal námět na starosti, má danou úlohu odsouhlasenou celým projektovým týmem i Product Ownerem. V dalším kroku je nutné sepsat podrobné zadání pro oddělení vývoje. Zadání musí být sepsané co nejpodrobněji, ale zároveň výstižně a se všemi potřebnými informacemi, aby se ušetřily časové náklady programátora. Zadání by mělo být předáno včetně obrázků pro ilustraci, vstupních souborů (například soubor pro import dat), či jiných souborů, které budou s novou funkcionalitou spolupracovat. Lpí se na tom, aby zadání byla krátká a celý námět

„rozdroben“ do co nejvíce drobných, ale ucelených vývojových úkolů. Každý tento menší úkol obsahuje část vývojového zadání z nově tvořené funkcionality. Je to z důvodu nečekaných změn ve Sprint Backlogu, či celém Product Backlogu. Zároveň to ulehčí práci i při následném testu, protože menší úkol se lépe testuje na soulad se zadáním než rozsáhlý celek funkcí.

  - Veškeré úlohy se díky podpůrnému informačnímu systému pro podporu projektového řízení (JIRA) propojí a vzniká tím uspořádání jednotlivých vývojových úkolů. „Realizace“ trvá do doby, dokud nejsou oddělením vývoje vypracovány veškeré úlohy a produktovým manažerem posouzené jako hotové.

- Ověření
  - Stav „Ověření“ získá původní úkol s námětem, jakmile jsou veškeré vývojové úkoly hotovy. Zde produktový manažer, Product Owner a zadavatel funckionality prověřuje vytvořenou funkcionalitu na základě zadání a výsledný vytvořený produkt. Jakmile je vše v pořádku u všech zainteresovaných členů projektového týmu a zákaznické strany, tak se celý úkol s námětem posouvá do stavu „Uzavřeno“. Jestliže se nalezne nějaká neshoda, tak se na základě komunikace zjišťuje důvod neshody a celý základní úkol s námětem se posouvá o úroveň výše na úroveň „Realizace“. Zde probíhají veškeré úkony znovu, dokud se úkol s námětem opět nedostane do stavu „Ověření“ a do nového kola zjištění správnosti zpracování funkcionality a celkového námětu.
- Uzavřeno
  - Stav „Uzavřeno“ je krokem po „Ověření“. Jedná se o stav, kdy se při kroku „Ověření“ rozhodlo, že je daná funkcionalita správně vypracovaná na základě námětu zadavatele. Product Owner tedy celý námět posouvá na krok „Uzavřeno“.
- Ke zrušení
  - Jedná se o krok projektového řízení, který je mimo celý cyklus. Tento krok znamená, že se celý produktový tým s Product Ownerem na základě diskuze rozhodnul, že daný námět není vhodný na zpracování (např. protože je v rozporu s legislativou, neodpovídá celkové koncepci produktu, je to

duplicitní nebo podobná funkcionalita k již existující atd.). V tom případě je rozhodnuto, že se námět nebude realizovat a celý námět se dostává do stavu „Ke zrušení“. I tyto náměty, které jsou takto uzavřeny, se evidují, aby bylo možné je zpětně dohledat a zjistit rychle důvod, proč se námět uzavřel a nerealizoval. To je velice vhodné, kdyby se požadavek na podobný námět opakoval například od jiného zadavatele.

2.4.3 Vývojový cyklus pro Požadavek na úpravu / Chybu

Vývojový cyklus pro „Požadavek na úpravu“ a „Chybu“ je nejrozsáhlejším a nejsložitějším vývojovým cyklem. Tento cyklus se používá na všechny dílčí úkoly a je jedno jestli to jsou úkoly, které řeší novou funkcionalitu, úpravu stávající, dodělávky, anebo opravy chyb. Na oba druhy úloh jsou kladeny stejné požadavky a je tedy nutné, aby oba druhy požadavků prošly co nejvíce strukturovaným projektovým cyklem. Takto strukturovaný vývojový cyklus dokáže projektovému týmu pomoci při odhalení velkého počtu nedostatků, které mohou vzniknout během vývoje, nebo které se během vývoje opomenuly dořešit.

V tomto vývojovém cyklu softwaru je zahrnuta jak produktová část, kde produktové oddělení řeší analýzu jednotlivých úkolů, sepsání vývojového zadání a konzultaci s ostatními členy produktového týmu, Product Ownerem a oddělením vývojového týmu. Tyto kroky jsou označeny přídomkem stavu úlohy „Produkt:“.

Dále jsou zahrnuty vývojové části úloh. Zde oddělení vývoje řeší celkové vypracování zadání funkcionality, které se řeší ve velké části nejprve ve vlastní vývojové větvi dané Epicem a následně se provádí Merge do hlavní vývojové větve aplikace. Během toho oddělení vývoje konzultuje jednotlivé kroky s oddělením produktu s konkrétním produktovým manažerem, který má úkol na starosti a je zde vedený jako řešitel. Tato část je označena přídomkem „Vývoj:“

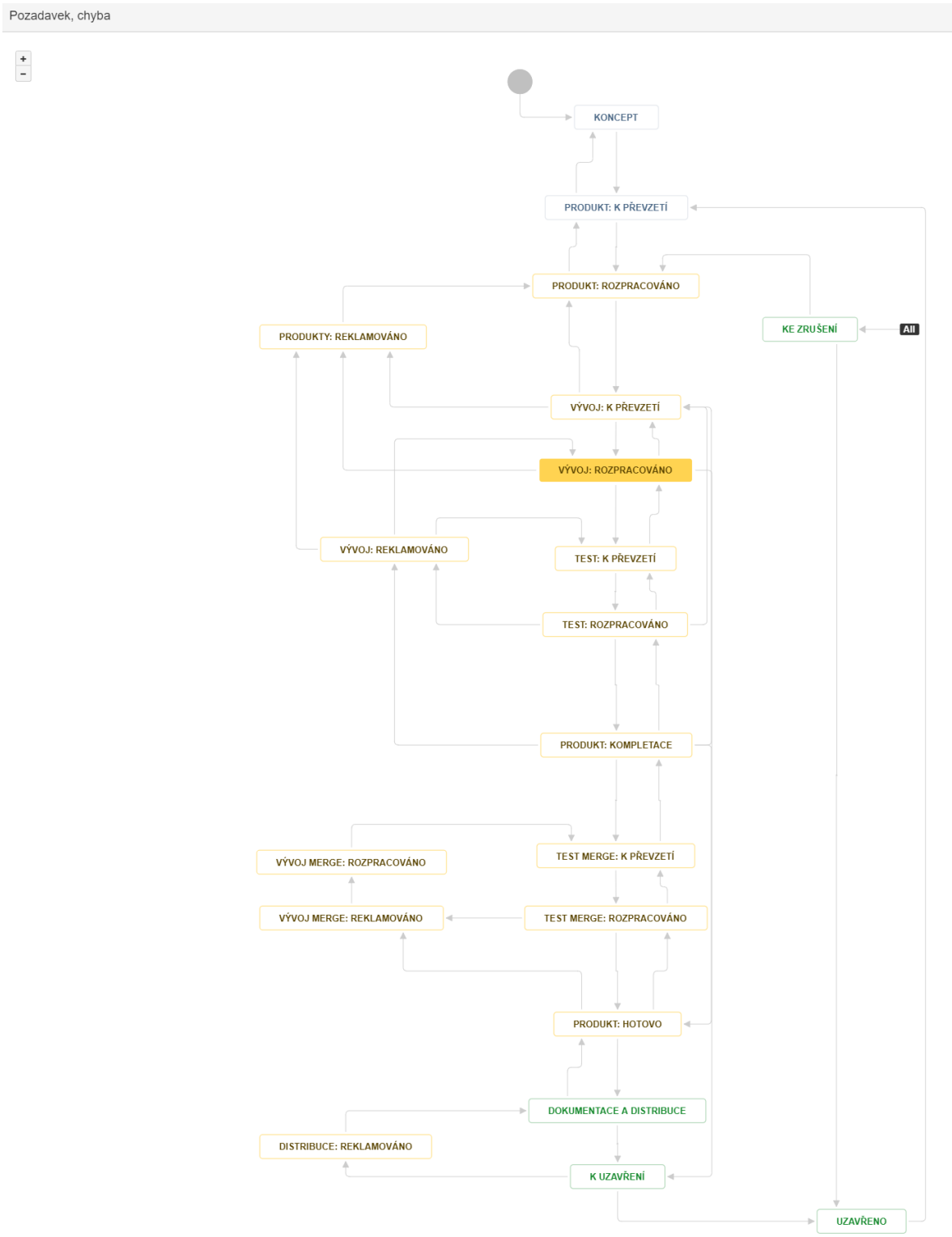
V další části je řešeno testování aplikace a nových funkcionalit s případnou opravou nalezených chyb v rámci Epicu. Tato část vývojového cyklu je označena přídomkem „Test:“. Pokud úkol projde testem a je v souladu se zadáním, přechází do stavu „Produkt: Kompletace“. V tomto stavu čeká na dokončení všech ostatních úkolů z Epicu. Jakmile jsou

dokončeny i ostatní úkoly z Epicu, dochází k merge vývojové větve (Epicu) do hlavní větve (viz kapitola 2.4.1.3 Merge), a probíhá druhá vlna testování, označena přídomkem „Test Merge:“. Tato fáze slouží k ověření, že funkcionalita není ovlivněna mergem do hlavní větve a je stejná jako ve vývojové větvi.

Po ověření se úkol posouvá do stavu „Produkt: Hotovo“, kdy je považován z hlediska vývoje za dokončený a plně funkční v hlavní stabilní větvi softwaru.

V závěrečné části probíhá tvorba dokumentace jednotlivých změn v aplikaci. Toto se děje v kroku „Dokumentace a distribuce“.

V následujících podkapitolách je podrobně sepsána činnost jednotlivých kroků projektového cyklu pro typ úloh „Požadavek na změnu“ a „Chyba“.



Obrázek 19 - vývojový cyklus pro Požadavek na úpravu / Chybu
(Zdroj: převzato a upraveno z interní dokumentace Inisoft, s.r.o.)

- Koncept
    - Koncept při druhu úlohy „Požadavek na úpravu“ a „Chyba“ je prvním krokem. K tomuto kroku vývojového procesu jsou tyto druhy úlohy přiřazeny v momentě, jakmile jsou projektovým týmem vytvořeny a uloženy.
    - V „Konceptu“ člen projektového týmu sepisuje základní informace ohledně požadavku na úpravu softwaru nebo chybě. Jestliže má člen produktového týmu dostatečné informace, aby vytvořil plnohodnotný úkol a tyto informace přehledně a správně zpracuje, tak je možné úkol posunout do dalšího kroku „Produkt: K převzetí“.
- Produkt: K převzetí
    - Úlohu dostává konkrétní řešitel v oddělení produktového týmu. V tomto úkole zanalyzuje obdržené požadavky a související informace. Jeho úkolem je posoudit, zda jsou informace správné a dostatečné a na základě diskuze s Product Ownerem zařadit úlohu do správného Epicu se správnou verzí vydání nové verze softwaru. Po provedení těchto úkonu se úloha posouvá k dalšímu kroku „Produkt: Rozpracováno“.
- Produkt: Rozpracováno
    - V tomto kroku se úloha stále drží u produktového manažera, který je uveden jako řešitel. Tento krok je pro doplnění důležitých informací, které se získají například analýzou legislativy či konzultace s ostatními členy jak produktového oddělení, tak i oddělení vývoje. V závěrečném kroku projektový manažer zkontroluje správné umístění úlohy v rámci Epics a verze. Po kontrole se posouvá úloha dále na krok „Vývoj: K převzetí“.
- Produkt: Reklamováno
    - Tento krok vývojového cyklu softwaru je pro stav, kdy oddělení vývoje obdrží úkol, ale na základě stávajících informací není schopen úlohu úspěšně zpracovat. Po konzultaci s Product Ownerem a předcházejícím projektovým manažerem, který byl řešitel dané úlohy, se úloha vrací o krok zpět na projektového manažera, který úlohu zpracovával. Úkolem projektového manažera je na základě konzultace doplnit chybějící informace, nebo na základě nově zjištěných informací úlohu upravit. Jestliže je tento krok

proveden úspěšně, tak se úloha posouvá na další krok a to „Vývoj: K převzetí“.

- Vývoj: K převzetí
    - V tomto kroku vývojového cyklu dostává úlohu oddělení vývoje. Úloha je přiřazena vedoucímu oddělení vývoje, který na základě komunikace s Product Ownerem urči Sprint Backlog. Po určení Sprint Backlogu a zařazení konkrétní úlohy do Sprintu daného týdne se určí podle Kanban Table vhodný programátor, který bude konkrétní úlohu zpracovávat, a úloha se mu přiřadí. Po provedení těchto kroků se ještě nemění stav kroku vývojového cyklu softwaru. Ten se mění, jakmile přiřazený programátor zanalyzuje obsah úlohy a označí ji, že na ní začíná pracovat. V tomto momentě se daná úloha posouvá na „Vývoj: Rozpracováno“.
- Vývoj: Rozpracováno
    - Přiřazený programátor úlohu označil, že na ní začal pracovat. Programátor nyní tvoří algoritmus a sepisuje kód, aby splnil zadání úlohy. Při řešení komunikuje s ostatními členy oddělení vývoje a případně i s projektovým manažerem, který úlohu zpracoval. Při zpracování celé úlohy probíhá komunikace mezi přiřazeným programátorem a členy projektového týmu k případnému vysvětlení požadavku. Je to z důvodu toho, že každý člověk si může psaný text vyložit po svém a je vhodnější ústní upřesnění a nasměrování programátora k vytvoření požadovaného algoritmu, než aby programátor vytvořil „něco“ podle sebe a ve výsledku odevzdal algoritmus s nepožadovaným výsledkem. Zároveň je zde i prostor přímo pro invenci programátora, pokud ho při vypracování úlohy napadne vhodnější řešení. Na základě tohoto zjištění navrhne úpravu zadání úlohy pro vypracování. Tento návrh konzultuje s projektovým manažerem a na základě toho se rozhodují, jestli se bude pokračovat původní zadání úlohy anebo navrženým a upraveným zadáním. Jakmile programátor vytvoří algoritmus, který splňuje zadání, tak se úloha posouvá na krok vývojového cyklu „Test: K převzetí“.
- Vývoj: Reklamováno

- Tento krok vývojového cyklu softwaru označuje stav, kdy úlohu obdrží zpět programátor od testera poté, co tester nalezne chybu či nesrovnalost oproti zadání.
- Programátor, který obdrží úlohu nazpátek, tak provádí úpravy na základě zpětné vazby od testera. Tento krok se při procesu vývoje úlohy nemusí vůbec stát anebo se může stát i vícekrát. Záleží na obtížnosti zadání, zkušenosti a šikovnosti programátora, jak danou úlohu dokáže zpracovat. Vždy když vrací úlohu zpět testerovi, tak posouvá na krok vývojového cyklu „Test: K převzetí“.

- Test: K převzetí
  - První krok vývojového cyklu softwaru, který má na starosti tester. Tester si v tomto kroku musí přesně projít zadání úlohy, nastudovat si, co přesně bylo cílem za vytvoření funkcionality anebo úpravy funkcionality. Dále si musí zjistit v jakém Epicu je daná funkcionalita vytvořena a v jaké verzi je publikován build aplikace. Jakmile tester takto úlohu nastuduje a zanalyzuje, tak přesouvá celou úlohu na krok „Test: Rozpracováno“.
- Test: Rozpracováno
  - Tester posunul úlohu na krok „Test: Rozpracováno“. Tím si otevřel úlohu, kterou obdržel od oddělení vývoje a nyní má za úkol otestovat a projít publikovanou verzi softwaru s funkcionalitou a provést několik schémat testů. Jelikož se jedná o první vlnu testů, tak je velice pravděpodobné, že se některé nedostatky testerovi povedou objevit. Všechny nedostatky sepíše detailně, co je nekorektně zpracováno, případně v jakém stavu a při jakých krocích se objevuje chyba a úlohu vrací o krok zpět na krok „Vývoj: Reklamování“. Zároveň s tímto přiřazuje nazpátek programátora, který úlohu vypracovával.
  - Jakmile s testováním skončil a nalezl nedostatky, tak posouvá úkol zpět v rámci vývojového cyklu na krok „Vývoj: Reklamováno“. Pokud se testerovi při provádění testovacích schémat nepodaří objevit nedostatek nebo chybu ve fungovaní funkcionality a celého softwaru a software je stabilní (neobjevují se jiné chyby), tak úlohu posouvá dále v rámci vývojového cyklu na krok „Produkt: Kompletace“.

- Produkt: Kompletace
    - Krok „Produkt: Kompletace“ znamená, že úloha prošla první iterací vývoje, je otestovaná a funkční v rámci Epicu. Tento krok dává úlohám vlastnost, že jsou připraveny k Mergi do hlavní vývojové větve. Pokud všechny úlohy v rámci jednoho Epicu získají tento krok v rámci vývojového cyklu softwaru, tak se celý Epic posouvá do stavu „Merge“.
- Test Merge: K převzetí
    - Tímto krokem začíná druhé kolo testování. Úloha se z Epicu dostala do hlavní vývojové větvě softwaru. Nyní je požadavek na testera zajistit opětovné otestování úlohy, aby ověřil, že během procesu Merge nedošlo k tomu, že nová funkcionalita přestala fungovat, začaly se objevovat některé chyby anebo jiné nechtěné konflikty, které mohly vzniknout právě Mergem do hlavní vývojové větve softwaru.
- Test Merge: Rozpracováno
    - Tester se zaměří na jednu konkrétní úlohu a posune ji do stavu „Test Merge: Rozpracováno“. V tomto kroku inicializoval testování úlohy ve druhé vlně a testuje úlohu v hlavní vývojové větvi aplikace. Opět veškeré nedostatky, či případné chyby podrobně zapisuje i například s přiloženými obrázky nebo logy aplikace. Po dotestování posouvá úlohu buď na krok vývojového cyklu softwaru „Vývoj Merge: Reklamováno“, když se během testu objevily chyby anebo na „Produkt: Hotovo“, pokud se během testu už chyby neobjevily.
- Vývoj Merge: Reklamováno
    - „Vývoj Merge: Reklamováno“ je krok vývojového cyklu softwaru, do kterého se úloha dostává na základě druhé vlny testování. Jakmile tester objeví ve druhé vlně testu chybu, znamená to, že funkcionalita z úlohy je již implementovaná v hlavní větvi softwaru. Oddělení vývoje a původní řešitel programátor obdrží úlohu nazpátek, aby na základě zpětné vazby od testera provedl následné nápravy.
    - Jelikož se jedná o druhou vlnu testování, tak nebývá pravidlem, že by se v tomto kroku už objevovaly některé chyby, zvlášť když se první vlna testování prováděla poctivě. Avšak se může stát, že se chyby naleznout i při tomto kroku vývojového cyklu softwaru.

- Vývoj Merge: Rozpracováno
    - V tomto kroku vývojového cyklu softwaru se provádí opravná činnost od programátora na základě zjištěných chyb z druhé vlny testování po implementování úlohy do hlavní větve softwaru. Programátor musí provést analýzu zpětně vazby od testera a na základě zpětné vazby provést nápravu funkčnosti aplikace. Po provedení nápravy posouvá úlohu v rámci vývojového cyklu na krok „Test Merge: K převzetí“ k opětovnému otestování právě provedené nápravy.
- Produkt: Hotovo
    - Do stavu „Produkt: Hotovo“ se úloha dostává po úspěšném Mergi a úspěšném otestování ve druhé vlně. Aby se úloha do tohoto stavu dostala, tak musí projít vývojovým cyklem bez nalezených chyb nebo nepřesností vzhledem k zadání. V tomto stavu se úloha „odškrtává“ vzhledem k vývojovým činnostem projektového týmu. Zbývá ještě provést a zpracovat uživatelskou dokumentaci na nově vzniklou funkcionalitu z úlohy. Tím se úloha posouvá do stavu „Dokumentace a distribuce“.
- Dokumentace a distribuce
    - Úloha je z pohledu vývoje a analýza dokončena. Úlohu obdrží k dořešení oddělení podpory zákazníka. Zde je jejich činnost a cíl ten, že si musí funkcionalitu z úlohy podrobně nastudovat a zpracovat uživatelskou dokumentaci, která se publikuje do uživatelského prostoru na webu „Centrum Informací“. Po zpracování dokumentace a nastudování nové funkcionality se úloha dostává do stavu „K uzavření“.
- Distribuce: Reklamováno
    - Jedná se o poslední krok vývojového cyklu softwaru, který vrací úlohu nazpátek. Děje se to v momentě, kdy oddělení podpory zákazníka zjistí při práci na úloze nějakou nesrovnalost. Tuto nesrovnalost musí detailně popsat a předat veškeré informace, podle kterých se jedná opravdu o nesrovnalost. Jakmile tyto informace a nedostatky sepíší, tak s tímto krokem vrací úlohu zpět na původního produktového manažera, aby si zpětnou vazbu od oddělení produktu zákazníka nastudoval. Na základě toho se a případné diskuze se rozhodne jakým, způsobem se bude nadále pokračovat. Může se buď vytvořit

nová úloha, která budou souviset s touto původní a bude zpracovávat zpětnou vazbu od člena oddělení podpory zákazníka. Zpětná vazba se může se také vyřešit v rámci původní úlohy. Úloha se dostane znovu do oddělení vývoje a zde se nedostatky vyřeší. Poslední možným způsobem řešení je ten, že se na základě diskuze s projektovým týmem a Product Ownerem nebude zpětná vazba řešit a úkol se posouvá na krok „K uzavření“.

- K uzavření
    - V kroku vývojového cyklu softwaru „K uzavření“ se řeší poslední evidenční informace úlohy, kdy se eviduje, jestli byla úloha opravdu realizována, nebo případně nebyla realizována a důvod této nerealizace. Dále se zde evidují verze aplikace, ve kterém se úloha nakonec reálně vyřešila. Posledním prvkem tohoto kroku je, že se musí předat informace Product Ownerovi, aby získal podvědomí o dokončení úlohy. Jakmile je toto provedeno, tak se úloha uzavírá krokem „Uzavřeno“.
- Ke zrušení
    - Speciální krok vývojového cyklu softwaru je „Ke zrušení“. Do tohoto kroku se úloha může dostat prakticky z každého jiného kroku. Tento krok „Ke zrušení“ uvádí, že se daná úloha nebude realizovat a zjistilo se to již během jiného kroku, a to ať už při úvodní analýze, detailnější analýze nebo později. V tomto kroku se u úlohy uvádí, proč se daná úloha nebude realizovat a přesunem do kroku „Ke zrušení“ se úloha zruší. Jakmile se tyto informace uvedou, tak se úloha posouvá na poslední krok vývojového cyklu softwaru, a to „Uzavřeno“.
- Uzavřeno
    - Posledním krokem vývojového cyklu softwaru a konkrétní úlohy. Úloha prošla celým tímto vývojovým cyklem, který je popsán a znázorněn výše, nebo byla zrušena. Evidují se i zrušené úlohy. Je to z důvodu zpětné dohledatelnosti.

3 Návrh zefektivnění vývojového cyklu

V následující kapitole je sepsáno několik návrhů na zefektivnění cyklu pro vývoj softwaru, na základě definované teorie v diplomové práci a praktických zkušeností získaných během práce ve společnosti, kterou se zabývá tato diplomová práce.

3.1 Určení jednoznačné metodiky

Během tvorby analýzy aktuálního stavu ohledně způsobu projektového řízení vývoje softwaru bylo určeno, že jeden z hlavních bodů nedostatků je, že nikde není jasně určeno, jakým způsobem společnost provádí projektové řízení. Při analýze bylo zjištěno, že se jedná o stav „někde na půli cesty“ mezi tradiční metodikou a metodikou agilní. To znamená, že některé činnosti se provádí stále způsobem tradičních metodik, i když se společnost snaží přesunout do agilního světa projektového řízení vývoje softwaru. Jak je sepsáno v druhé kapitole této diplomové práce, tak je ve společnosti použit velmi strukturovaný procesní graf, kterým by měl projít každý prvek Product Backlogu podle druhu úlohy. Problém však nastává v momentě, kdy se u některých úloh takto nejedná, a někteří členové produktového týmu mají silnější vliv a dokáží si své úlohy „protlačit“ svým způsobem bez ohledu na strukturovaný procesní graf pro projektové řízení vývoje softwaru. Tyto upřednostněné úlohy následně neprobíhají vůbec žádným Epicem, ale jsou vyvíjeny přímo do hlavní vývojové větve programu. Zpravidla jsou zde tvořeny iterační metodou z tradičních metodik. Toto obcházení vývojového procesu může způsobit problémy v plánování týdenních Sprintů. Navíc mohou vznikat nepřesnosti či jiné nepřehlednosti v daných plánech či dokonce využití pracovní kapacity jednotlivých oddělení projektového týmu. Z technického hlediska je pak problém merge jednotlivých Epiců (vývojových větví), který je pak náročnější do změněného kódu proti stavu na začátku, kdy se Epic (větev) oddělil.

Zároveň je to doprovázeno dalším závažným problémem, který byl identifikován. Vedoucí pozice projektového týmu jsou zastoupeny pracovníky, kteří nemají úplně velké zkušenosti s projektovým řízením pomocí agilní metodiky. Tento stav vede k tomu, že se agilní metodiky prosazují od členů projektového týmu, kteří jsou v organizační struktuře společnosti níže. Tento přístup je shledán za ne velice vhodný. Může docházet ke střetům

mezi členy projektového týmu. Projekt je veden dle procesního grafu vývoje softwaru a do této části projektu vstoupí člen projektového týmu z vyšší pozice organizační struktury, který nemá tolik zkušeností z agilní metodiky řízení a razí svoji cestu. Tato vlastní cesta je zpravidla postavena na tradiční metodice. V momentě, kdy dojde ke střetu názorů, tak se projet začne řídit podle člena projektového týmu, který je v organizační struktuře firmy výše.

Návrhem pro zlepšení stavu je vydání závazné interní směrnice, která stanoví proces řízení vývoje softwaru podle agilních metodik a začne se prosazovat od nejvyšších pozic členů projektového týmu. Jakmile se tohoto docílí, tak se začne prosazovat jeden shodně obecný postup při řízení projektu, který se bude razit od nejvyšších pozic až po ty nejspodnější pozice projektového týmu. Tím se omezí velký počet názorových rozporů při procesním řízení a sníží se časová náročnost průběhu vývoje softwaru. Zároveň vznikne jakási předvídatelnost jednotlivých procesních kroků.

V souvislosti s tímto je návrh pro zvýšení úsilí při zaškolování jednotlivých členů, nebo alespoň vedoucích pozic, ve spojitosti s projektovým řízením vývoje softwaru pomocí agilních metodik. Tato druhá část návrhu bude zpočátku pro společnost náklad. Přinese to však společnosti tolik cenné zkušenosti ohledně této problematiky. Tyto zkušenosti pomůžou postavit pevné základy pro řízení projektů pomocí agilních metodik. Na těchto pevných základech je pak možné dále stavět, a profitovat z toho bude jak celý projektový tým společnosti, tak i společnosti jako taková. Výsledek návrhu bude zefektivnění a zoptimalizovaní projektového řízení. Tato optimalizace a zvýšení efektivnosti projektového řízení ve výsledku sníží časové a finanční náklady při vypracování jednotlivých projektů a zvýší i celkové příjmy firmy.

3.2 Jasné určení role Product Ownera v projektovém týmu

Tento návrh pro optimalizaci projektového řízení při vývoji softwaru v agilní metodice souvisí s předcházejícím bodem. Při analýze bylo zjištěno, že není definovaná role Product Ownera, která je brána v úvahu při projektovém řízení vývoje softwaru napříč celou firmou. Product Owner zde existuje pouze jako jakási „neoficiální“ role, která se bere v potaz pouze pomocí interní komunikace mezi oddělením vývoje, oddělením analýzy pro produkt a oddělením technické podpory. Propojení oddělení obchodu a Product Ownera je velice slabé.

Místo toho se děje, že obchodník po získání zpětné vazby vynechá komunikaci s Product Ownerem a s požadavkem na analýzu přichází rovnou do oddělení analýzy. Vznikají tak nepřesnosti a nepředvídatelnosti při organizaci a plánování časového harmonogramu jednotlivých oddělení. K tomu Product Owner ztrácí kontrolu při řízení a přehled o jednotlivých požadavcích na vývoj. Zároveň Product Owner ztrácí možnost plánování Product Backlogu a následně i jednotlivých Sprintů a Sprint Backlogů. Omezují se jeho možnosti při plánování vydávání jak iterací funkcionalit pro pilotní zákazníky, tak i plánování vydávání oficiálních verzí softwaru pro všechny zákazníky. Díky těmto odkladům při vydávání verzí mohou vznikat záporné zpětné vazby od zákazníků, nesrovnalosti v komunikaci a podobně. Všechny tyto záporné vlastnosti mohou vrhat špatné světlo na firmu mezi zákazníky i potencionálními zákazníky. Pro firmu se tento problém projeví snížením celkových příjmů, ale může se i projevit zvýšením celkových nákladů při tvorbě jednotlivých funkcionalit a oficiálních verzí softwaru.

Návrhem v rámci tohoto bodu je to, aby se v organizační struktuře firmy přímo vytvořila pozice Product Ownera. Tato role bude existovat oficiálně „úředně“ ve firemní politice. Člen projektového týmu, který zastoupí tuto roli bude mít jasně definované činnosti a cíle, které by měl plnit. V první kapitole této práce je pozice Product Ownera teoreticky definována a definice také obsahuje to, že by role Product Ownera neměla být spojovaná s nějakou další rolí v rámci organizační struktury nebo projektového týmu. Je to z toho důvodu, aby si Product Owner dokázal nastavit potřebné cíle při vývoje softwaru s potřebným nadhledem. Kdyby se stalo, že role Product Ownera bude zastoupena členem projektového týmu, který již zastává ještě nějakou další roli, tak okamžitě Product Owner ztrácí potřebný nadhled. Tím může nastat problém, že při plánování při vývoji softwaru se upřednostňuje pouze nějaká primární část projektu a další jsou „odstaveny“ na druhou kolej. Toto se nesmí stát, a proto by role Product Ownera měla být zastoupena členem projektového týmu, který bude nezávislý vůči dalším rolím projektového týmu nebo pozic organizační struktury firmy.

Jakmile vznikne oficiálně role Product Ownera, která bude zastoupena nezávislým a schopným členem projektového týmu, tak vznikne správné plánování Product Backlogu, plánování Sprint Backlogů a konkrétních Sprintů. S tím je spojeno, že vznikne komplexnější přehled zdrojů a kapacit při plánování vývoje softwaru. Zároveň vznikne důvěryhodnější odhad při plánování vydávání oficiálních verzí pro zákazníky.

Zavedením činností, které s sebou role Product Ownera se zlepší komunikace se zákazníkem, což by mělo přinést převažující kladnou vazbou. Tímto vytvořeným kladným podvědomím o firmě a její schopnosti tvořit a úspěšně dokončovat jednotlivé projekty se otevírá možnost oslovit pro spolupráci více potencionálních zákazníků. Firma získá větší celkové příjmy jak ze stávajících, tak i z nově získaných potencionálních zákazníků. Zároveň omezí celkové náklady při nepřesnosti plánování nebo jiných problémech s tím spojených.

3.3 Jasné určení role Scrum Master v projektovém týmu

Jestliže je v předchozí kapitole zanalyzován návrh zavedení role Product Ownera do oficiální role projektového týmu, jelikož existuje ve společnosti pouze v neoficiální formě, tak u role Scrum Master je situace jiná. Při analýze současné situace při projektovém řízení vývoje softwaru se zjistilo, že role Scrum Master v projektovém týmu neexistuje vůbec. Jak vyplývá z definice role Scrum Mastera, tak se jedná o stejně důležitou roli jako Product Owner. Role Scrum Mastera má za cíl propojit celý projektový tým do kompaktní a efektivní entity společnosti. Měl by se starat o celý tým tak, aby spolu co nejvíce úzce spolupracoval, aby zde panovala pravidelná komunikace a pokud by v projektovém týmu vznikl problém, tak je to právě Scrum Master, který tento problém řeší do zdárného a uspokojivého závěru. Zároveň je to takový dirigent při schůzích projektového týmu, aby diskuze probíhala správnou nekonfliktní formou a postupovala efektivně v celém průběhu.

Ve společnosti je v současnosti projektový tým, který je rozčleněn podle frameworků aplikace. To znamená, že celá aplikace je rozdělená do několika částí. Rozdělení je zobrazeno v tabulce níže.

Framework	Počet členů z projektového týmu	Popis frameworku
CORE	6 členů	V tomto frameworku se vyvíjejí funkcionality, které se používají napříč celou aplikací. Jedná se například o předky formulářů, obecná chování (tlačítka, tisky, výběrové seznamy, číselníky, funkce zvyšující komfort ovládání, práva a podobně). Nedílnou součástí je i prostředí celé aplikace a její databázový model. Tento framework je základním a je stavebním kamenem pro ostatní.
Odpady SK	2 členové	V tomto frameworku se vytváří funkcionality zaměřené na práci pro modul Odpady SK pro odpadové hospodářství. V tomto modulu se respektují zákony Slovenské republiky. Jedná se hlavně o zákon 79/2015 Z. z. a jeho prováděcí vyhlášky (365/2015 Z. z., 366/2015 Z. z, 371/2015 Z. z., atd.) Ministerstva životního prostředí Slovenské republiky.
Odpady CZ	2 členové	V tomto frameworku se vytváří funkcionality pro modul Odpady CZ pro odpadové hospodářství. V tomto modulu se respektují zákony České republiky. Jedná se o hlavní vyhlášku 383/2001 Sb. Vyhláška o podrobnostech nakládání s odpady a 185/2001 Sb. Zákon o odpadech Ministerstva životního prostředí České republiky.
Sklad a Obchod	3 členové	V tomto frameworku se vytváří funkcionality, jak již název napovídá, pro práci se skladovým hospodářstvím a obchodní činností firmy. Modul je v současné době napojen na modul Odpady SK, proto respektuje i stejnou legislativu.

Tabulka 4 - rozdělení produktové části projektového týmu na základě frameworků softwaru (zdroj: vlastní)

V tabulce výše je znázorněno, jakým způsoben je utvořena projektová část produktového týmu. Členové projektového týmu, kteří pracují na konkrétním frameworku (Odpady SK, Odpady CZ, Sklad a Obchod) tvoří úzce spolupracující skupinu. Jakmile se tyto jednotlivé skupinky sejdou na jedné schůzi při diskuzi funkcionalit do frameworku CORE, tak mohou vznikat několika hodinové diskuze, a je to logické, jelikož každý člen projektového týmu má svoji představu, jak by daná funkcionalita měla fungovat. Jenže tyto několikahodinové diskuze nebývají příliš efektivní a jsou velmi náročné na časovou kapacitu všech zúčastněných členů projektového týmu při konkrétním meetingu. Opět vzniká posloupnost událostí, které vzniknou touto neefektivností. Tím, že se ztrácí časová kapacita při těchto neefektivních debatách, tak vznikají časové deficity u jiných dalších činnostech jako například práce na analýzách nových funkcionalit, tvorba zadání pro oddělení vývoje a podobné. Tímto vzniká neefektivnost při odbavování jednotlivých úkolů a tím vznikají dodatečné celkové náklady a klesají celkové příjmy firmě.

Přesně tomuto má zamezit role Scrum Mastera. Ten by těmto nesrovnalostem měl zamezit, dirigovat celý meeting a tlačit ho správným směrem, aby nespadl do nějakého neefektivního začarovaného kruhu, který trvá několik hodin. Scrum Master by měl celý meeting vést, klade správné otázky na jednotlivé členy projektového týmu a tím tvoří produktivní prostředí pro ostatní členy projektového týmu. Zároveň může používat různé nástroje pro podporu efektivity projektového týmu.

Návrhem tohoto bodu je tedy oficiálně definovat roli Scrum Mastera. Vymezit jasné cíle Scrum Mastera a vymezit jasně projektový tým, který bude mít na starosti. Rolí Scrum Mastera musí být zastoupen členem projektového týmu, který bude komunikativní, nezávislý vzhledem k ostatním rolím projektového týmu a bude produktivní s vhodnými vlastnostmi. Vhodné vlastnosti jsou definované v této diplomové práci v první kapitole. Zároveň bude podporovat člena projektového týmu v roli Product Ownera. Scrum Master odstraní neefektivnosti a nedostatky při fungování celého projektového týmu.

3.4 Optimalizace tvorby dokumentace

Bod „Optimalizace tvorby dokumentace“ popisuje návrhy pro tvorbu dokumentace. Jedná se o tvorbu dokumentace interní (návrhy požadavků, analýzy, vývojové zadání vývojářům a podobné…) i externí (uživatelská dokumentace k funkcionalitám, programu, a podobné).

3.4.1 Optimalizace tvorby dokumentace pro interní činnosti

Z pohledu tvorby interní činnosti je třeba optimalizovat tvorbu dokumentace kvůli několika nedostatkům. Při analýze současného stavu projektového řízení při vývoje softwaru ve společnosti bylo zjištěno, že hlavní problém tkví převážně v tom, že neexistují žádné obecné předpoklady anebo pravidla pro tvorbu dokumentace. Společnost používá systém Confluence a JIRA, kde se tvoří i interní dokumentace. Je to rozděleno tak, že v Confluence se tvoří spíše BA, analýzy projektů a funkcionalit, diskuze k legislativě, diskuze k vývoji a směrování softwaru a podobně, a v JIRA se tvoří přesné náměty od zákazníků co požadují, zadání pro oddělení vývoje a případně výzkumné úkoly, když je třeba provést nějakou analýzu k funkcionalitě ohledně možností, použití technologií a podobného.

Tato část bude tudíž rozdělena na další dvě podkapitoly.

3.4.1.1 Optimalizace tvorby dokumentace v Confluence

Při analýze současné situace bylo zjištěno, že v podpůrném systému „Confluence“ existují takzvané prostory. Tyto prostory vymezují témata, které by měly všechny příspěvky plnit svým obsahem. Existuje tedy několik hlavních prostorů, které se v nižších úrovních ještě rozdělují, přesně definují. Jedná se tedy o stromovou strukturu. Zde je problém v tom, že se jedná o ne přesně definovanou strukturu. V reálném případě je velmi obtížné naleznout požadovanou dokumentaci, protože jeden konkrétní článek může odpovídat několika tématům a tím pádem může být zařazen pod několik prostorů. To vše ztěžuje přehlednost v tomto systému Confluence. Druhý problém je v tom, když uživatel chce vytvořit článek a není možné přesně určit prostor, kam článek patří. Pokud se tento nedostatek opakuje a neprovede se změna k nápravě, jedná o stále se prohlubující problém. To je nepřijatelné v tom, že se stále více znepřehledňují jednotlivé prostory v Confluence.

Zde je návrh pro zlepšení, že se jasně definuje stromová struktura prostorů pro psaní dokumentace. Tato stromová struktura bude jasně a přesně definovaná, aby se tvůrce dokumentace nemusel rozhodovat mezi různými prostory, kam svoji práci vloží. Zároveň se provede zaškolení všech členů projektového týmu. Školení se bude provádět na téma ohledně prostorů v Confluence, jak psát celkovou dokumentaci a do jakých prostorů ji vkládat.

Druhý návrh na zlepšení je, že se z projektového týmu vyberou členové týmu, kteří obdrží jednotlivé prostory pod svoji správu a tyto prostory budou udržovat přehledném stavu s články, které obsahově splňují daný prostor. Tím by se zamezilo, že by se v daných prostorách začaly tvořit nepřehlednosti.

3.4.1.2 Optimalizace tvorby dokumentace v JIRA

V systému JIRA se tvoří dokumentace převážně pro úvodní analýzu funkcionalit, a hlavně zadání pro oddělení vývoje. Při analýze současného stavu tvorby dokumentace do systému JIRA nebyly nalezeny tak závažné nedostatky. Avšak některá vylepšení by zde bylo také možné aplikovat.

Při tvorbě úvodní analýzy k novým funkcionalitám pro software se používá systém JIRA. Slouží k tomu, aby si konkrétní člen projektového týmu zjistil požadované úvodní informace, možné využití technických prostředků a všechny tyto informace si utřídil a urovnal. Jakmile se stav úvodní analýzy přesune do již detailní přímé analýzy k nové funkcionalitě, musí se tento proces začít evidovat v systému Confluence, kde se tato dokumentace schraňuje a ukládá kvůli přehlednosti. Občas se v současném stavu ale stává, že někteří členové projektového týmu provádí úvodní analýzu k funkcionalitě v JIRA. Následující detailní analýzu zde ponechají a tato detailní analýza zůstane pouze v systému JIRA, aniž by se dostala do systému Confluence. Návrhem pro změnu je tedy to, aby se tento proces optimalizoval a apelovalo se na všechny členy projektového týmu, aby správně plnili svoje povinnosti při tvorbě dokumentace.

Druhým nedostatkem, který se objevil při analýze nedostatků tvorby dokumentace v JIRA, je, že se nedodržuje obecná struktura při psaní dokumentace JIRA. Tím je myšleno, že se nedodržuje žádný formát styl, co se týče nadpisů, stylu textu nebo členění textu

v dokumentaci. To je v následujícím čase pro ostatní členy projektového týmu nepřesné a nepřehledné a při práci se s takovouto dokumentací stráví více časových prostředků jen kvůli tomu, že dokumentace nemusí být úplně přehledná. Návrhem pro změnu je tedy to, aby se zavedla obecná stylistická forma při tvorbě dokumentace. Tato jednotná stylistická forma dokáže ušetřit čas ostatním členům projektového týmu při práci s dokumentací.

3.4.2 Optimalizace tvorby dokumentace pro externí činnosti

Z pohledu tvorby dokumentace pro externí činnosti je myšleno to, jakým způsobem se tvoří uživatelská dokumentace pro software a jaké nedostatky, které při této činnosti vznikají.

Uživatelskou dokumentaci tvoří člen projektového týmu z oddělení podpory. Ten je povinen si projít přidělené úlohy. Úlohy, které jsou označeny stavem „K dokumentaci“ musí zpracovat do prostoru „Centrum Informací“ na oficiálních webových stránkách společnosti. Tento prostor se tvoří pro zákazníka jako první úroveň podpory a představení nových funkcionalit softwaru. Při tvorbě této dokumentace si člen projektového týmu z oddělení podpory musí nastudovat nové funkcionality z úloh a případně tyto nové funkcionality konzultovat s oddělením produktu, aby se dospělo ke správnému pochopení funkcionality od člena oddělení podpory a tím pádem i správnému zaznamenání nové funkcionality do dokumentace do prostoru Centra Informací.

Zde je zjištěno z analýzy současného stavu, že k tvorbě dochází v nedostatečném časovém předstihu a toto nekompletní vytvoření uživatelské dokumentace v Centru Informací brzdí vydávání nových verzí aplikace. Z teoretického hlediska, které je popsáno v první kapitole diplomové práce bylo determinováno, že uživatelská dokumentace nemá být tvořena sáhodlouhými texty, ale stručně zaznamenáno, v čem nová funkcionalita spočívá a jak ji použít. To vše jasně a stručně. Tento předpoklad je splněn. Druhý předpoklad je, že by tato uživatelská dokumentace měla být dostupná s vydáním nové verze. U tohoto předpokladu je snaha také o plnění, ale vydání verze je právě zbrzděno tvorbou dokumentace. Problém je způsoben tím, že se pro tvorbu dokumentace vymezuje méně pracovních kapacit a zpravidla uživatelskou dokumentaci tvoří pouze jeden konkrétní člen projektového týmu. Návrhem pro řešení tohoto problému je zvýšit kapacitu alokovanou pro tuto činnost, aby se urychlil

proces tvorby uživatelské dokumentace do prostoru Centra Informací a tím se zamezilo problémům a odložením s vydáním nové verze softwaru.

3.5 Optimalizace procesu tvorby Product Backlogu

Z analýzy současného stavu projektového řízení vývoje softwaru a konkrétně činnosti tvorby Product Backlogu se zjistilo několik nedostatků. Hlavním nedostatkem při této činnosti je, že tvorba Product Backlogu v jednotlivých Epicích není příliš kontrolovaná. Ve velkém procentu případu se stává, že když se provádí tvorba Product Backlogu pro Epic z frameworku CORE, tak se tento Epic rozroste do nesmírné velikosti. Je to dáno tím, že do tohoto frameworku zasahují všichni členové produktového týmu z oddělení Produktu. Tím, že je tento Epic obstaráván větším počtem členů, tak se velice rychle plní úlohami. Jakmile je tento Epic rychle naplněn úlohami, u kterých se neřídí jednotně priority (každý člen projektového týmu stanovuje prioritu sám, a obyčejně jako vysokou prioritu), stává se pro oddělení vývoje problém s dodržením termínů pro vypracování úloh z Epicu. Toto nedodržování má následek, že se zbrzdí jednotlivé procesy z projektového cyklu u některých úloh. To má za výsledek zpoždění testů nových funkcionalit, tvorby dokumentace na nové funkcionality a vydání nových oficiálních verzí softwaru.

Návrhem pro zlepšení je zvýšená kontrola tvorby Product Backlogů pro jednotlivé Epiky a zvláštně pro Epiky z frameworku CORE. Je třeba více úlohy do jednotlivých Epiců třídit dle priorit a do konkrétních Epiců zařadit pouze omezené množství úloh s ohledem na časovou náročnost a pracnost. Ziskem by měla být zvýšená přehlednost pro celý projektový tým u jednotlivých Epiců a snížení časové náročnosti při vypracování jednotlivých Epiců. Výsledkem by mělo být také to, že je možné vydávat mnohem častěji, jak verze pro pilotní zákazníky, tak i oficiální verze pro všechny zákazníky. Tyto verze nejsou sice tak obsáhlé novými funkcionalitami, ale společnost získává mnohem častěji zpětnou vazbu od zákazníků. Hlavním benefitem tohoto návrhu je zamezení tomu, aby se nedokázaly dodržet termíny pro vydávání nových verzí aplikace.

4 Zhodnocení, zpětná vazba, případný návrh dalšího řešení

Výsledkem zkoumání možností optimalizačního procesu v hodnocené společnosti, ve které byla prováděna analýza současného stavu projektového řízení při vývoji softwaru, je stav, kdy firmě byly poskytnuty veškeré optimalizační body. Firma na základě všech získaných podkladů provedla vlastní analýzu současného stavu projektového řízení. Pomocí vlastní analýzy došla k závěru, že se shoduje ve všech navržených optimalizačních bodech, které vyšly z analýzy zpracované v této diplomové práci. Tyto body jsou v současné době postupně zpracovávány a implementovány do projektového řízení společnosti. Implementace se provádí s cílem zvýšení celkové efektivity práce a s tím spojené zvýšení celkových příjmů společnosti a snížení celkových nákladů společnosti.

Veškeré optimalizační body navržené v této diplomové práci pomohou posuzovanou společnost posunout dále do agilních metodik projektového řízení. Vytvoření oficiálních rolí Product Ownera a Scrum Mastera považuji za velmi důležité a závažné. Společnost, která se snaží vyvíjet moderní aplikace a celé projektové řízení provádí pomocí agilní metody, tyto dvě role ve svém projektovém týmu nutně potřebuje. Obě role jsou stavebním kamenem řízení projektů pomocí agilních metod.

Určení jednoznačné metodiky (tradiční nebo agilní) považuji za další možnost ke zvýšení efektivity. Z prováděné analýzy vyplynulo, že společnost se snaží řídit projekty pomocí agilních metod. Nedostatkem však je, že jsou agilní metody uplatňovány pouze částečně, a navíc jen v některých činnostech. Tím dochází k pravidelným střetům při předávání úloh mezi konkrétními odděleními a řešiteli. Určení a stanovení agilní metody od vedení společnosti a zaštítění ji například do interní dokumentace považuji za další velký přínos v posuzované společnosti, které pomohou více provázat firmu agilními metodami.

Dále věřím že také změna při tvorbě dokumentace (interní i externí) přinese posuzované společnosti velký přínos, ač se to na první pohled nemusí zdát. Přínos bude spočívat v celkové přehlednosti dokumentace, množství poskytovaných informací a celkovém zlepšení vztahů mezi členy projektového týmu, a i mezi společností a zákazníky.

Společnost by nadále měla postupovat tak, že zapracuje veškeré provedené změny a celý nový způsob projektového řízení nechá v neměnném stavu po určité časové období. Po tomto období provede vyhodnocení a novou analýzu stavu projektového řízení. Na základě nové analýzy se určí další optimalizační body pro projektové řízení, kterými se případně vyřeší nově objevené nedostatky. Při provádění těchto optimalizačních cyklů projektového řízení by se mělo dbát na počet řešených optimalizačních bodů. Každý optimalizační cyklus by měl obsahovat pouze jen několik optimalizačních bodů týkajících se hlavních nedostatků. Takovýto styl provádění optimalizačních cyklů by měl zamezit vzniku nedostatků projektového řízení z důvodu velkých a obsáhlých změn. Opakovaným prováděním optimalizačního cyklu by posuzovaná společnost měla dosáhnout vyřešení několika hlavních nedostatků v celém svém projektovém řízení a zvýšit tím tak svoji celkovou efektivitu při tvorbě softwaru. Celková efektivita spočívá v tom, že se provede více jednotlivých druhů prací (obchodní a vývojové činnosti, analýz, poradenství, technické podpoře a další) za méně potřebného času.

Závěr

Výsledkem této diplomové práce je analýza projektového řízení ve společnosti, která se zabývá vývojem softwaru pro evidenci odpadového hospodářství v České republice a ve Slovenské republice. Na začátku diplomové práce je vymezena teorie projektového řízení tak, aby byl celý okruh projektového řízení správně pochopen. Jsou zde vysvětleny rozdíly mezi jednotlivými metodami způsobu vedení projektu, a to jak tradičními metodikami, tak i agilními metodikami. Následně jsou v práci popsány veškeré důležité pojmy, se kterými se člověk může potkat při projektovém řízení. Postupně se celá práce začíná více zaměřovat spíše na agilní metodiky, protože s touto metodou pracuje i posuzovaná společnost.

V další fázi práce je prováděna celková analýza současného stavu projektového řízení společnosti při vývoji softwaru. V této části se klade důraz na používané systémy, které se ve společnosti využívají pro podporu projektového řízení. Jedná se o informační systémy INFO, od kterého společnost postupně opouští, JIRA a Confluence, což jsou systémy přesně uzpůsobeny na agilní projektové řízení. Společnost si toho je vědoma a postupně na systémy JIRA a Confluence migruje velkou část svých činností spojených s projektovým řízením. Dále se analýza zabývá procesem práce na jednotlivých úlohách (Námět, Požadavek na úpravu / Chybu, Epic), kde je přesně vymezen procesní graf s popisem jednotlivých kroků. Analýza se také zabývá nástroji agilních metod a způsobem používání těchto nástrojů ve firmě (Epic, Sprint, Product Backlog, Sprint Backlog a další).

Na základě provedené komplexní analýzy současného stavu projektového řízení ve společnosti jsou vymezeny hlavní nedostatky, které se podařilo objevit. Tyto nedostatky jsou pečlivě charakterizovány a jsou navrženy optimalizační kroky, které tyto nedostatky dokáží odstranit. Celá optimalizace je zpracována v souladu s možnostmi společnosti, kdy jsou navržené body střízlivými požadavky nevyžadující vynaložení ohromných finančních prostředků.

Veškeré optimalizační body byly v zabývané společnosti prezentovány. Společnost tato diplomová práce natolik zaujala, že provedla vlastní interní analýzu současného stavu projektového řízení a provedla porovnání s analýzou, která je vypracovaná v této diplomové

práci. Díky tomuto celkovému procesu došla k závěru, že navrhované optimalizační body bude postupně implementovat do svého projektového řízení.

Na základě tohoto celkového procesu a komunikaci se v poslední kapitole práce provádí zhodnocení navrhovaných optimalizačních bodů. Veškeré body vedou ke zvýšení efektivnosti práce a tím zvýšení příjmů společnosti a omezení nákladů vynakládaných na vývoj vlastního softwaru.

Seznam použité literatury

GÁLA, Libor, Jan POUR a Zuzana ŠEDIVÁ, 2015. *Podniková informatika: počítačové aplikace v podnikové a mezipodnikové praxi*. 3., aktualizované vydání. Praha: Grada Publishing. Management v informační společnosti. ISBN 978-80-247-5457-4.

BROOKSHEAR, J. Glenn, David T. SMITH a Dennis BRYLOW, 2013. *Informatika*. Brno: Computer Press. ISBN 978-80-251-3805-2.

MYSLÍN, Josef, 2016. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press. ISBN 978-80-251-4650-7.

ŠOCHOVÁ, Zuzana a Eduard KUNCE, 2019. *Agilní metody řízení projektů*. 2. vydání. Brno: Computer Press. ISBN 978-80-251-4961-4.

SINGH, Ajit, 2019. *Agile Methodology With Scrum*. Germany: GRIN Verlag. ISBN 9783668979697.

Poradna: Co je to agilní metoda řízení? [online], 2018. Praha: BusinessInfo.cz [cit. 2020-02-09]. Dostupné z: https://www.businessinfo.cz/clanky/poradna-co-je-to-agilni-metoda-rizeni/

Jak vybrat Scrum Mastera aneb proč i zkušení Scrum Masteři selhávají v Agilní transformaci[online], 2014. Praha: Zuzana Šochová [cit. 2020-02-09]. Dostupné z: https://soch.cz/blog/management/agile/scrum-management/jak-vybrat-scrum-mastera-aneb-proc-i-zkuseni-scrum-masteri-selhavaji-v-agilni-transformaci/

DOLEŽAL, Jan, 2016. *Projektový management: komplexně, prakticky a podle světových standardů*. Praha: Grada Publishing. Expert (Grada). ISBN 978-80-247-5620-2.

15 Project Management Methodologies You Need to Know About: Learn the highlights of each methodology [online], c2016. Tallinn: Toggl [cit. 2020-02-12]. Dostupné z: https://toggl.com/project-management-methodologies/

MCGREAL, Don a Ralph JOCHAM, ©2018. *The professional product owner: leveraging Scrum as a competitive advantage*. Boston: Addison-Wesley. Professional scrum series. ISBN 978-0134686479.

VERHEYEN, Gunther, 2019. *Scrum – A Pocket Guide - 2nd edition:: A Smart Travel Companion*. 2nd. The Netherlands: Van Haren. ISBN 978-9401803755.

LEMAY, Matt, 2019. *Agile for Everybody: Creating Fast, Flexible, and Customer-First Organizations*. 2nd. Sebastopol: O`Reilly Media. ISBN 978-1-492-03351-6.

OCKERMAN, Stephanie a Simon REINDL, 2019. *Mastering professional scrum: coaches' notes for busting myths, solving challenges, and growing agility*. Boston: Addison-Wesley. ISBN 978-0134841526.

MILOŠEVIĆ, Dragan a Russ J. MARTINELLI, [2015]. *Project management toolbox*. Second edition. Hoboken, New Jersey: Wiley. ISBN 978-111-8973-202.

COBB, Charles G., 2015. *The Project Manager's Guide to Mastering Agile: Principles and Practices for an Adaptive Approach*. New Jersey: John Wiley. ISBN 978-1-118-99104-6.

ROUDIAS, Jihane, ©2015. *Mastering Principles and Practices in PMBOK, Prince 2, and Scrum: Using Essential Project Management Methods to Deliver Effective and Efficient Project*. New Jersey: Pearson Education. ISBN 978-0-13-406081-1.

MAXIMINI, Dominik, [2018]. *The scrum culture: introducing agile methods in organizations*. Second edition. Cham, Switzerland: Springer. Management for professionals (Springer). ISBN 978-331-9738-420.

HARNED, David, 2018. *Hands-On Agile Software Development with JIRA: Design and manage software projects using the Agile methodology*. Birmingham: Packt. ISBN 978-1-78953-213-5.

2016: Digitální revoluce přinese tlak na zrychlení procesů a kyberbezpečnost[online], 2016. Praha: CIO Business World on-line [cit. 2020-04-20]. Dostupné z: https://businessworld.cz/bezpecnost/2016-digitalni-revoluce-prinese-tlak-na-zrychleni-procesu-a-kyberbezpecnost-12748