Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Návrh implementace agilní metodiky pro vývoj softwaru

Denisa Tomková

!!!

Místo tohoto textu vložte PŘEDNÍ stranu zadání práce, které si můžete vyexportovat do PDF v IS.CZU.cz, pokud již máte schválené zadání i děkanem PEF.

!!!

!!!

Místo tohoto textu vložte ZADNÍ stranu zadání práce, které si můžete vyexportovat do PDF v IS.CZU.cz, pokud již máte schválené zadání i děkanem PEF.

V případě, že Vaše zadání je na více než 2 strany, vložte i další strany.

!!!

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Návrh implementace agilní metodiky pro vývoj softwaru" jsem vypracovala samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autorka uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušila autorská práva třetích osob.

V Praze dne 31.3.2023 _____

Poděkování

Ráda bych touto cestou poděkovala vedoucímu své diplomové práce Ing. Markovi Píckovi, Ph.D. za odbornou a pedagogickou pomoc. Děkuji také svému šéfovi Ing. Petrovi Loudovi za pomoc při zpracování vlastní části práce a také své rodině za podporu.

Návrh implementace agilní metodiky pro vývoj softwaru

Abstrakt

Tato diplomová práce se zabývá návrhem implementace agilní metodiky pro reálný tým, který se zabývá vývojem softwaru. V teoretické části diplomové práce jsou stručně popsány modely životního cyklu vývoje softwaru, agilní metodiky, a to Scrum, Extrémní programování a Feature Driven Development, ze kterých je v praktické části vybraná vhodná metodika. Dále přináší přehled dalších modelů a metodik, pro které jsou krátce uvedeny nejdůležitější principy. Poslední část teoretické části tvoří popis metod vícekriteriální analýzy variant, a to metod stanovení vah a metod stanovení pořadí variant. Praktická část diplomové práce se zabývá analýzou současné situace daného týmu. Na základě vícekriteriální analýzy variant je vybrána optimální metodika pro daný tým. V poslední části praktické části je sepsán návrh implementace vybrané metodiky a jeho shrnutí.

Klíčová slova: vývoj softwaru, agilní metodiky, vícekriteriální analýza variant, implementace metodiky, Scrum, Extrémní programování, Feature Driven Development

Design implementation of agile methodologies in software development

Abstract

This diploma thesis is focused on design implementation of agile methodology for a real team which deals with software development. In the theoretical part of the thesis there are briefly described models of software development life cycle, agile methodologies, namely Scrum, Extreme Programming, and Feature Driven Development, one of them is chosen as the optimal methodology in the practical part of the thesis. Then it brings the overview of another models and methodologies, each with a brief summary of their important principals. The last part of the theoretical part consists of a description of methods of multicriteria analysis of variants, specifically methods for determining the weights and methods for determining the order of variants. The practical part of thesis deals with analysis of the actual situation of the given team. Based on multicriteria analysis of variants there is chosen the optimal methodology for the given team. In the last part of the practical part there is described design implementation of agile software development and its summary.

Keywords: software development, agile methodologies, multicriteria analysis of variants, implementation of methodology, Scrum, Extreme Programming, Feature Driven Development

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

ASD – Adaptivní vývoj softwaru
FDD – Feature Driven Development
HR – Human resources
RC – Release candidate
RUP – Rational Unified Process
TDD – Test Driven Development
UI – Uživatelské rozhraní
UML – Unified Modeling Language
VAV – Vícekriteriální analýza variant
WSA – Metoda váženého součtu
XP – Extrémní programování

1 Úvod

V dnešní době, kdy se mění požadavky zákazníka ze dne na den, je tradiční způsob vývoje a myšlení v IT již nedostačující. Tradiční „těžké“ metodiky poskytují, i přes svou velikost a složitost, stále velké množství výhod a tím je především lepší kontrolovatelnost a přímočařejší řízení. Zákazníci jsou náročnější, konkurence je obrovská. Je potřeba dokázat vyrábět software rychleji, ale ne za cenu kvality, přizpůsobit se novým technologiím a také je nezbytné dokázat reagovat na rychle měnící se požadavky zákazníka. Proto jsou agilní metodiky díky svému efektivnímu přístupu tak populární.

Reálný tým, pro který je návrh implementace agilní metodiky sepsán, je součástí společnosti, která dodává zákazníkovi hardware, software na provoz daného hardwaru, poskytuje následné služby, ale také dodává software, který vyvíjí daný tým. Tým by chtěl zefektivnit své procesy, aby byl pružněji schopen doručovat požadovanou funkcionalitu, plnil naplánovanou práci ve stanoveném termínu a v požadované kvalitě. Celkově by chtěl pracovat efektivněji a v příjemnějším prostředí. Pro snahu o dosažení těchto cílů si daný tým selektivně vybral pár postupů a procesů z agilní metodiky Scrum, které postupně zavedl, ale zavedl je v modifikované formě, která nepřináší očekávané výsledky. V rámci této práce autorka po dohodě s týmem vytvoří návrh pro implementaci agilní metodiky, která bude nejvíce vyhovovat potřebám a požadavkům týmu a přinese jim vytoužené výsledky a především efektivnější způsob práce. Pro výběr vhodné agilní metodiky byly po dohodě s týmem zvoleny metodiky Scrum, Extrémní programování a Feature Driven Development.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem diplomové práce je návrh implementace agilní metodiky pro reálný tým, který se zabývá vývojem softwaru.

Dílčí cíle vedoucí k naplnění tohoto hlavního cíle jsou:

- popis agilních metodik a vícekriteriální analýzy variant;
- charakteristika vybraného reálného týmu a analýza jeho současné situace a stávajících procesů;
- porovnání vybraných agilních metodik vývoje softwaru;
- výběr optimální metodiky pro daný tým.

2.2 Metodika

V teoretické části práce bude uveden krátký úvod do softwarového inženýrství a základní popis dvou z nejznámějších modelů životního cyklu vývoje software, a to konkrétně vodopádový a spirálový model. Dále budou podrobněji popsány tři agilní metodiky, jež slouží jako vstupní základ pro vlastní práci, kde bude na základě vícekriteriální analýzy variant vybrána optimální metodika pro sestavení návrhu její implementace pro reálný tým. Následně bude stručně uveden přehled dalších modelů a metodik. Poslední částí teoretické části práce bude představení vícekriteriální analýzy variant, metod stanovení vah kritérií a metod stanovení pořadí variant, což bude následně použito pro výběr optimální metodiky pro stanovený tým.

V rámci vlastní práce bude provedena analýza konkrétního IT týmu, složení daného týmu, jeho aktuální situace, stávajících procesů v týmu a používaných nástrojů k vývoji, testování a uchovávání dokumentace. Pro vícekriteriální analýzu agilních metodik, konkrétně Scrum, Extrémní programování a Feature Driven Development, budou stanovena kritéria, na základě kterých bude vybrána jediná agilní metodika, jenž bude považována za optimální řešení. Váhy jednotlivým kritériím budou stanoveny pomocí Saatyho metody stanovení vah. Hodnocení jednotlivých kritérií provede samostatně tým, pro který bude agilní metodika vybírána. Následně bude autorkou práce na základě spolupráce s týmem

zpracováno odůvodnění daného ohodnocení kritérií. Po stanovení vah jednotlivých kritérií bude použita metoda stanovení pořadí variant, metoda TOPSIS.

Výsledkem práce bude návrh implementace metodiky, jež byla vybrána jako optimální řešení podle vícekriteriální analýzy požadavků na danou metodiku. Návrh bude rozdělen do tří fází a každá fáze bude mít časové určení předpokládané doby implementace. Návrh implementace metodiky bude shrnut na konci vlastní práce.

3 Teoretická východiska

3.1 Softwarové inženýrství

Dle definice IEEE (1990) je softwarové inženýrství „*systematický, disciplinovaný a kvalifikovaný přístup k vývoji, tvorbě a údržbě softwaru*“. Softwarové inženýrství se zabývá všemi aspekty produkce softwaru od prvotní fáze specifikace systému až po jeho následnou údržbu v provozu. Usiluje o dosahování výsledků v požadované kvalitě doručovaných ve stanoveném čase a v rámci schváleného rozpočtu (Sommerville, 2013).

Podle Křena, Kočí (2010) mezi hlavní cíle softwarového inženýrství patří management projektu, konkrétně řízení životního cyklu projektu a dosažení požadovaného výsledku v požadovaném čase a techniky, a to analýza, návrh, programování, testování.

3.1.1 Softwarový produkt

Softwarový produkt je výrobkem určeným k předání uživateli. Softwarový produkt je tvořen souhrnem počítačových programů, jednotlivých procedur, pravidel a s nimi související dokumentace a data k provozu daného softwaru (Ráček, 2013).

Sommerville (2013) definuje dva typy softwarového produktu:

1. **Obecný produkt** – samostatné systémy, jejichž autorem je vývojářská firma, a které se prodávají zákazníkům jako hotové produkty. Příkladem jsou různé databáze, textové editory, grafické balíčky, nástroje pro řízení projektů, účetní systémy apod.
2. **Přizpůsobený (zakázkový) produkt** – produkty vyrobené na základě konkrétních požadavků zákazníka. Čas doručení je zpravidla delší a cena zde bývá výrazně vyšší.

Základní rozdíl mezi těmito dvěma typy softwarových produktů spočívá především ve specifikaci softwaru. U obecných produktů si specifikaci obvykle řídí samotná vývojová organizace, kdežto u těch zakázkových řídí specifikaci výsledného produktu zákazník, který si daný software objednal. S postupem doby ale vznikají většinou obecné produkty, které se následně přizpůsobují tak, aby vyhovovaly požadavkům zákazníka.

Podle Sommerville (2013) klíčovými vlastnostmi kvalitního softwaru jsou:

- **Schopnost údržby** – software musí být vyvíjen tak, aby dokázal reagovat na proměnlivé požadavky zákazníků. Jedná se o klíčovou vlastnost, protože změny jsou nevyhnutelnou součástí vývoje.
- **Spolehlivost a bezpečnost** – spolehlivost softwaru přináší stabilitu a bezpečnost. Spolehlivý software by při selhání neměl způsobit ekonomické či fyzické škody. Neoprávnění uživatelé by k softwaru neměli mít přístup.
- **Efektivita** – zahrnuje rychlost reakce, dobu zpracování, využití paměti apod.
- **Přijatelnost** – software musí být přijatelný pro okruh lidí, pro který je zamýšlen. Musí být srozumitelný, použitelný a kompatibilní s jinými systémy, které využívají.

3.1.2 Strategie vývoje software

Strategií vývoje software může být obecně výběr metodiky softwaru, čímž se rozumí souhrn různých postupů, pravidel a nástrojů, jenž se používají pro návrh, plánování a řízení vývoje softwaru. Existuje škála metodik, které vznikly za posledních několik let, každá má nějaké klady a každá nějaké zápory. Mezi příčiny vzniku takového množství metodik může být rozmanitost firemních kultur v dané organizaci, nebo také použité technologie, které vyžadují různé metody a techniky. Jedna metodika nemusí být vždy vhodná k vývoji všech typů softwaru.

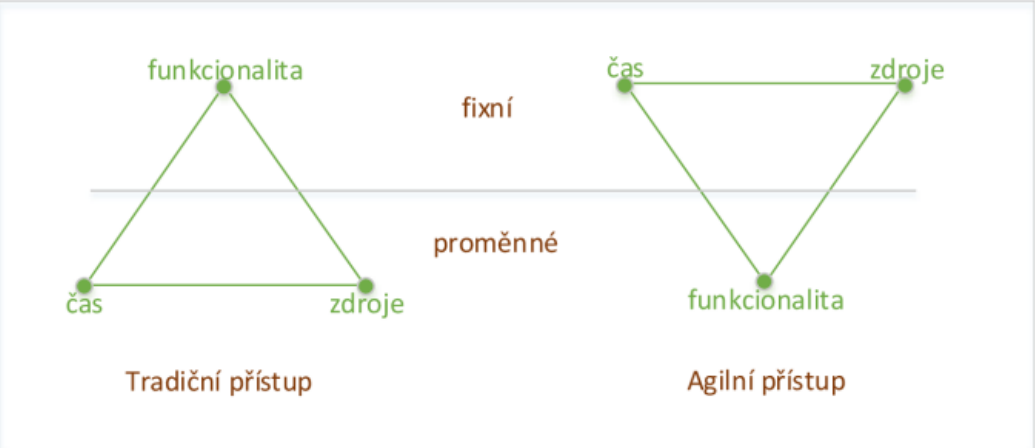
Široký výběr metodik pak představuje značné potíže při volbě té správné pro jednotlivé firmy, či samostatné týmy. Důvodem je především to, že metodiky nejsou popsány jednotným způsobem a nejsou jednoznačně kategorizovány (Buchalcevová, 2009).

Podle Buchalcevové (2009) jsou kritéria, která vedou k usnadnění volby konkrétní metodiky následující:

- **Zaměření metodiky**
    - globální metodiky – vývoj, řízení a provoz informačního systému v rámci celého podniku
    - projektové metodiky – vývoj informačního systému v konkrétní oblasti

- **Rozsah metodiky**
  - metodiky, které pokrývají celý životní cyklus vývoje softwaru
  - dílčí metodiky, které se zabývají jen částí cyklu životního vývoje software (př. návrh, implementace, testování)
- **Váha metodiky**
  - těžké metodiky – přesně definované procesy, postupy a artefakty
  - lehké metodiky – principy a praktiky místo přesných procesů
- **Typ řešení**
  - vývoj nového řešení
  - rozšíření stávajícího řešení
  - integrace řešení
  - implementace typového řešení
- **Doména – předmětná oblast**
  - „*customer relationship management*“ – řízení vztahu se zákazníky
  - „*e-learning*“ – elektronické vzdělávání
  - obecný software
- **Přístup k řešení**
  - strukturovaný vývoj
  - objektově-orientovaný vývoj
  - komponentový vývoj

Obzvláště na základě kritéria Váha metodiky lze jednotlivé metodiky rozdělit na dva základní typy, těžké neboli tradiční (rigorózní) metodiky a lehké neboli agilní metodiky. Tradiční metodiky kladou důraz zejména na plánování, řízení a měření definovaných procesů při vývoji software. Naopak agilní metodiky kladou důraz především na projekt, který je zapotřebí vyvíjet rychlým tempem a postup, kterým se dosáhne cíle, je druhotný. Hlavní myšlenkou agilních metodik je co nejdříve začít vyvíjet a postupně dělat úpravy na základě zpětné vazby od zákazníka. Dalším charakteristickým rozdílem těchto dvou typů metodik je jejich pohled na zdroje.



Obr. 1 Tradiční vs. agilní metodiky – pohled na zdroje (Kadlec, 2004)

Hlavním cílem pro tradiční metodiky je splnit požadavky na systém, které jsou již na začátku samotného vývoje stanoveny jako fixní. Proměnné čas a zdroje se stanovují na základě těchto neměnných požadavků. Při agilním vývoji jsou naopak požadavky na systém považovány za proměnnou a mohou se v průběhu vývoje měnit v závislosti na zdrojích a času, které jsou pro agilní vývoj fixní (Buchalcevová, 2009).

3.2 Modely životního cyklu vývoje software

Životní cyklus vývoje softwaru nebo také proces vývoje softwaru je v podstatě posloupnost určitých aktivit, které vedou k vytvoření softwarového produktu.

Životní cyklus vývoje softwaru obsahuje čtyři základní aktivity (Sommerville, 2013):

1. **Specifikace** – definice funkcionality vyvíjeného softwaru a operační omezení.
2. **Vývoj** – návrh a programování softwaru.
3. **Validace** – kontrola stavu softwaru, zda odpovídá požadavkům zákazníka.
4. **Evoluce** – úprava softwaru tak, aby odpovídal proměnlivosti požadavků zákazníka.

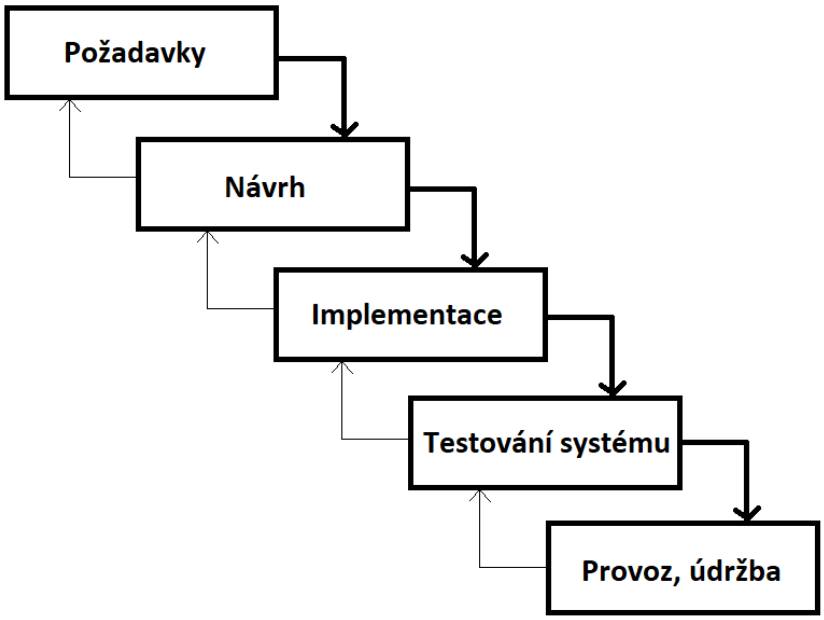
Model životního cyklu vývoje softwaru představuje určitý rámec realizace procesů životního cyklu v časovém sledu. Mezi nejznámější modely patří vodopádový model a spirálový model (Buchalcevová, 2009).

3.2.1 Vodopádový model

Vodopádový model patří k nejstarším modelům vývoje softwaru. Vodopádový model vznikl v sedmdesátých letech a je charakterizován tím, že jednotlivé fáze životního cyklu softwaru se provádí postupně, navazují jedna na druhou a nijak se neprolínají. K další fázi vývoje se přistupuje až ve chvíli, kdy je předchozí fáze zcela dokončená, nelze tedy vyvíjet více fází najednou (Kadlec, 2004). Tento přístup měl smysl především v dřívějších dobách, kdy většina systémů byla vymýšlena hlavně samotnými vývojáři s pouze malými nebo žádnými náměty od zainteresovaných stran (Laplante, Neill, 2004).

Hlavní fáze vodopádového modelu jsou (Sommerville, 2013):

- **Analýza a definice požadavků** – na základě komunikace s uživateli systému se určují systémové služby, omezení a cíle. Podrobněji definované požadavky slouží jako specifikace systému.
- **Návrh systému** – navrhuje se celková systémová architektura. Návrh systému identifikuje a popisuje základní abstrakce softwarového systému a jejich vztahy.
- **Implementace a testování jednotek** – v této fázi se realizuje návrh systému jako sada programů či programových jednotek. Samotné testování jednotek ověřuje, zda každá splňuje danou specifikaci.
- **Integrace a testování systému** – programové jednotky se integrují a testují jako celek, aby se zjistilo, zda software odpovídá požadavkům. Po otestování se předává kompletní systém zákazníkovi.
- **Provoz a údržba** – systém se předává k provozu a opravují se chyby, které nebyly nalezeny v dřívějších fázích životního cyklu, vylepšuje se implementace systémových jednotek a zdokonalují se systémové služby na základě nových požadavků.



Obr. 2 Vodopádový model (vlastní zpracování)

Na obrázku životního cyklu vodopádového modelu lze vidět, že model je striktně sekvenční, z jedné fáze vývoj postupuje do další a pokud se zjistí, že v předchozí fázi nastala chyba, vrací se vývoj o jeden krok zpět. Teprve ve fázi provoz a údržba je možnost vrátit se ke kterékoliv fázi a umožnit tak provedení jakýchkoliv změn (Wideman, 2004).

Ve vodopádovém modelu se nepracuje s iteracemi, častými novými verzemi, nefunguje zde ani průběžná komunikace se zákazníkem. Především kvůli absenci komunikace se zákazníkem a absenci flexibility se tento model stal pro velké projekty nedostatečný (Kadlec, 2004). I přes svou nedostatečnost je však nadále využíván v projektech, kde by využíván být neměl (Myslín, 2016).

3.2.2 Spirálový model

Spirálový model jako první popsal americký softwarový inženýr Barry W. Boehm v roce 1986 jako model zaměřený na analýzu rizik. Tento model navazuje na vodopádový model, řeší některé jeho nedostatky. Spirálový model patří do skupiny takzvaných přístupů řízených riziky, což znamená, že vstup do další fáze závisí na pečlivě provedené analýze rizik a všech možných problémů. Rizika v kontextu spirálového modelu lze chápat v nejobecnějším slova smyslu.

Model je založen na iterativním přístupu, a především zavádí opakovanou analýzu všech rizik. Díky tomu se lépe vyrovnává s pozdější úpravou požadavků. Jednotlivé kroky se neustále opakují, dokud není produkt hotov. Hlavní myšlenkou je navazování nových částí na důkladně prověřený základ. Vývoj se provádí na základě hrubé specifikace požadavků, která je následně v pozdějších fázích upřesňována. Model je vhodný i pro větší projekty.

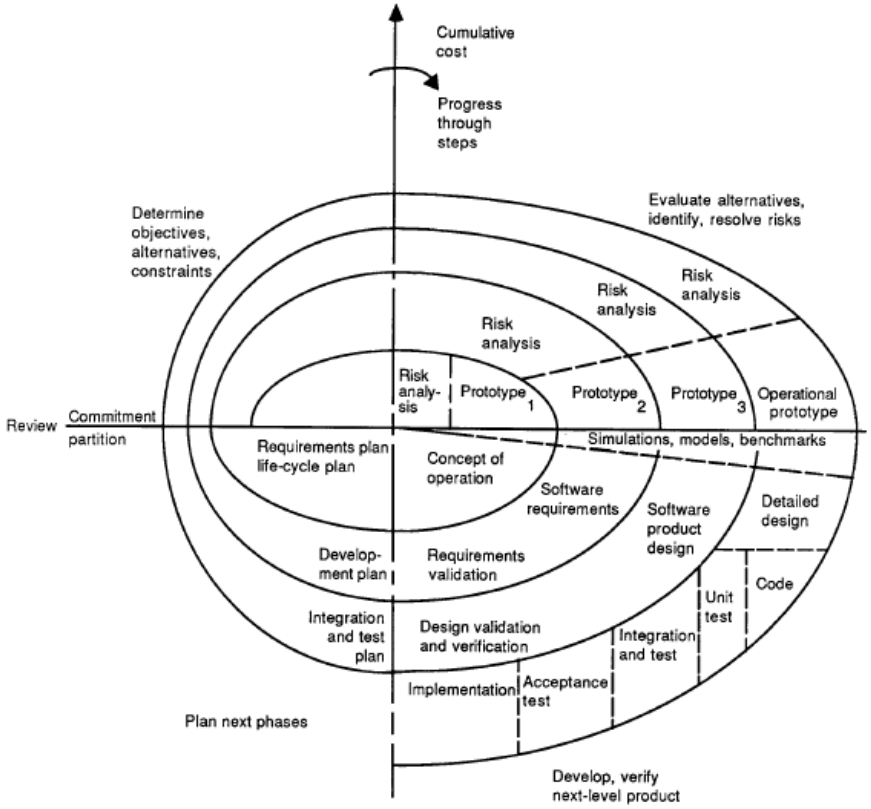
Celý proces je reprezentován pomocí spirály, nikoli jako posloupnost aktivit. Každá smyčka spirály představuje jednu fázi softwarového procesu. Spirálový model kombinuje prevenci změn s tolerancí změn, změny jsou výsledkem rizik projektu. Každá smyčka ve spirále se dělí na čtyři sektory (Boehm, 2000).

Fáze cyklu spirálového modelu jsou následující (Sommerville, 2013):

- **Určení cílů, alternativ a omezení** – definuje konkrétní cíle pro danou fázi projektu. Identifikují se různá omezení a rizika, vytváří se podrobný plán řízení. Případně lze naplánovat možné alternativy.
- **Hodnocení alternativ, identifikace a řešení rizik** – pro každé identifikované riziko se provádí podrobná analýza. Na základě analýzy se přijímají kroky k omezení rizika.
- **Vývoj a validace produktu** – po hodnocení rizik dochází k volbě modelu vývoje systému, zda půjde vývoj cestou tvorby prototypu, či zda bude nejvýhodnější použít vodopádový model.
- **Plánování další fáze** – přichází revize a zhodnocení, zda pokračovat další smyčkou spirály. Pokud padne rozhodnutí pokračovat, sestaví se plány na další fázi projektu.

Po každé fázi následuje testování, hodnocení a předání dílčích výsledků. Software se tedy testuje pravidelně už od raných fází. Díky pravidelnému a včasnému testování dochází k brzkému odhalování chyb. Najde-li se v počátcích vývoje chyb víc, upraví se následně analýza.

Spirálový model, jak ho původně publikoval Boehm (1988) lze vidět na následujících obrázku.



Obr. 3 Spirálový model (Boehm, 2000)

Model je vhodný na komplexní projekty, je však náročný na plánování jednotlivých iterací, pro malé projekty se tudíž může stát model zbytečně složitým (Boehm, 2000).

3.3 Agilní metodiky vývoje software

Agilní metodiky jsou poměrně novým přístupem, který přináší systémovou integraci do firemního prostředí a současně zpochybňuje aktuální stav. Díky častější a hodnotnější zpětné vazbě od zainteresovaných stran, agilní techniky zlepšují výkonnost technologických projektů a díky lepšímu řízení organizačních změn, týmové práci a komunikaci, také zvyšují konkurenční výhody jednotlivých firem (Paquette, Frankl, 2015). Změny vždy vylepšují projekt a poskytují přidanou hodnotu (Eby, 2016).

3.3.1 Agilní manifest

Agilní manifest byl důsledkem průmyslové frustrace v 90. letech. Enormní časová prodleva mezi zadáním požadavků zákazníka a doručení požadovaného softwaru vedla ke zrušení mnoha projektů, které neustály často měnící se požadavky na výsledný software. V roce 2001 se sešlo 17 vývojářů a dali spolu dohromady prohlášení publikované právě pod názvem Agilní manifest, jež se stal základní kamenem agilního přístupu (Eby, 2016).

Agilní manifest definuje čtyři základní hodnoty (Agile, 2001):

- jednotlivci a interakce před procesy a nástroji,
- fungující software před vyčerpávající dokumentací,
- spolupráce se zákazníkem před vyjednáváním o smlouvě,
- reagování na změny před dodržováním plánu.

Výše uvedené základní hodnoty agilního vývoje zdůrazňují, na co je třeba se zaměřit před neúprosným dodržováním procesů a plánů, či vyčerpávající dokumentací (Eby, 2016).

Principy stojící za Agilním manifestem (Agile, 2001):

- Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
- Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
- Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
- Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
- Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
- Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.
- Hlavním měřítkem pokroku je fungující software.
- Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.

- Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.
- Jednoduchost – umění maximalizovat množství nevykonané práce – je klíčová.
- Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.
- Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

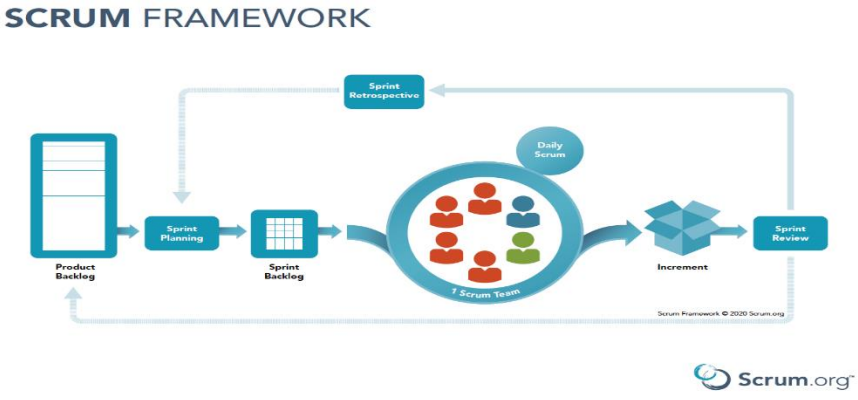
3.3.2 Scrum

Scrum vznikl na přelomu 20. a 21. století, jeho autory jsou Ken Schwaber, Jeff Sutherland a Mike Beedle. Všichni byli zkušení vývojáři, proto se rozhodli vyvinout efektivnější metodiku, která bude lépe reagovat na požadavky zákazníka. Pojem scrum vznikl podle formace v rugby, ve které tým funguje jako celek, stejně tak i vývojový tým se snaží jako celek vyvinout kvalitní a funkční software (Kadlec, 2004; Cohn, 2010).

3.3.2.1 Charakteristika

Scrum patří mezi nejpoužívanější a nejoblíbenější agilní metodiky. Je založený na iterativním a inkrementálním vývoji. Iterativní a inkrementální přístup této metodiky zvyšuje schopnost reagovat na případné problémy v rámci vývoje. Tým získává pravidelnou zpětnou vazbu zákazníka (Bell et al., 2017).

Podstatným základem Scrumu je komunikace, nejen v rámci týmu, ale i mezi týmem a zadavatelem. Scrum přináší zadavateli více svobody při udávání směru, kterým se bude vývoj ubírat a vývojáři jsou schopni pružně reagovat na technické překážky, aniž by překračovali stanovené termíny (Koďousková, 2021).



Obr. 4 Implementace Scrum metodiky na týmové úrovni (scrum.org)

3.3.2.2 Role

Scrum tým tvoří obvykle 10 a méně lidí, skládá se z jednoho scrum mastera, vlastníka produktu a samotného týmu vyvíjející produkt. Fungují jako jeden tým bez hierarchie, jedná se o soudržnou jednotku profesionálů (Cohn, 2010; Schwaber a Sutherland, 2020), kteří jsou v danou dobu zaměřeni na jeden konkrétní cíl, tzv. produktový cíl.

Scrum tým nese odpovědnost za všechny činnosti související s dodávaným produktem, ať se jedná o spolupráci se zainteresovanými osobami, ověřování, údržbu, provoz, výzkum, vývoj nebo další, co by mohlo být požadováno. Scrum tým je odpovědný za vytvoření hodnotného a užitečného přírůstku, tzv. inkrementu, každého sprintu (Schwaber a Sutherland, 2020).

Mezi základní role patří (Schwaber a Sutherland, 2020):

- **Vlastník produktu** (Product owner) reprezentuje stranu zainteresovaných subjektů a je hlasem zákazníka. Kromě toho má na starosti prioritizaci produktu a s tím i řízení backlogu (Koďousková, 2021). Odpovídá za maximalizaci hodnoty produktu, která vyplývá z práce celého týmu.
- **Scrum master** je odpovědný za správné založení a následné fungování Scrumu v týmu. Pomáhá týmu porozumět Scrumu a jeho fungování v praxi. Odpovídá za odstranění překážek týmu na dodání produktových cílů (Schwaber a Sutherland, 2020). Scrum master se stará o to, aby se developeři mohli plně soustředit na svou práci, obstarává komunikaci s okolím, vyřizuje jejich požadavky a snaží se udržovat motivaci v týmu (Koďousková, 2021).
- **Development tým** je zodpovědný za vytvoření samotného produktu. Členové týmu úzce spolupracují a každý je schopen zastat více než jedinou funkci. Tým si sám určuje cíl pro každý sprint, organizuje si práci a dělá cokoliv v rámci daných pravidel, aby cíle sprintu dosáhl. Na konci každého sprintu tým prezentuje výslednou práci vlastníkovi produktu.

3.3.2.3 Artefakty

Scrum artefakt představuje určitou práci nebo hodnotu a obsahuje konkrétní závazek, aby bylo zajištěno, že poskytne informace na základě, kterých lze měřit pokrok.

Artefakty Scrumu jsou (Schwaber a Sutherland, 2020):

- **Product backlog –** je zdrojem práce pro scrum tým. Jedná se o seznam funkcionality, kterou je potřeba během vývoje vykonat. Na základě daného seznamu se vytvoří soupis obecnějších úkolů, které se vloží do backlogu. Dané úkoly je následně nutné seřadit podle jejich důležitosti při budování přidané hodnoty. Na základě prioritizace se úkoly rozdělí do větších skupin a dále se řeší pouze ta s nejvyšší prioritou. Dále u jednotlivých úkolů scrum tým odhadne časovou náročnost, s těmito úkoly se poté pracuje v rámci jednoho sprintu (Koďousková, 2021). Závazkem product backlogu je produktový cíl, který popisuje budoucí stav produktu, podle kterého se scrum tým řídí a plánuje svou práci.
- **Sprint backlog** – je viditelným obrazem práce týmu v reálném čase. Vytváří se během schůzky plánování sprintu. Vzniká jako množina požadovaných vlastností s nejvyšší prioritou z product backlogu. Tyto vlastnosti jsou dále rozděleny na jednotlivé menší úkoly, které jsou týmem ohodnocovány podle náročnosti. Sprint backlog je tedy seznamem úloh, které by měly být splněny v rámci daného sprintu.
- **Přírůstek** (inkrement) **–** tvoří všechny dokončené úkoly v průběhu posledního sprintu a všech předchozích. Na konci každého sprintu musí přírůstek splňovat stanovené podmínky pro akceptaci dodávky.

3.3.2.4 Události

Každá událost ve Scrumu je brána jako příležitost ke kontrole a přizpůsobení artefaktů Scrumu. Scrum události se používají k vytvoření určité pravidelnosti a k minimalizaci schůzek, které při využívání Scrumu nemusí být potřebné.

Události Scrumu jsou (Šochová, Kunce, 2019):

- **Sprint –** nebo také iterace je základní jednotkou Scrumu. Délka sprintu je časově ohraničena, obvykle se jedná o 14denní cyklus. Během sprintu pracují členové týmu na úkolech ze sprint backlogu. Výstupem každého sprintu by vždy měla být nějaká dokončená funkcionalita či fáze vývoje produktu. Zákazník po každém sprintu dostane otestovanou verzi s novou funkcí. Může tedy sledovat, zda se vývoj ubírá správným směrem, vznášet různé připomínky, případně ovlivňovat další postup (Koďousková, 2021).

- **Backlog grooming –** je společnou schůzí pro vlastníka produktu a vývojový tým, obvykle se plánuje do poloviny sprintu. Na této schůzi vysvětluje vlastník produktu týmu jednotlivé úkoly, které jsou v product backlogu. Případné nové úkoly tým ohodnotí. Pro ohodnocování se obvykle používají relativní jednotky. Tým při ohodnocení jednotlivých úkolů bere v potaz jejich náročnost, komplexitu a možná rizika.
- **Plánovací porada** (sprint planning) – je zahajovací událostí celého sprintu. V rámci plánovací porady se stanoví práce, která má být splněna během následujícího sprintu. Vlastník produktu představuje cíl daného sprintu a zodpovídá otázky vývojového týmu (Schwaber a Sutherland, 2020).
- **Standup** (daily scrum) – se koná každý den ve stejný čas na stejném místě obvykle po dobu 15 minut. Účastní se ho celý tým včetně vlastníka produktu a scrum mastera, pracují-li aktivně na položkách sprint backlogu a sdílí s ostatními na čem pracoval včera, na čem bude pracovat dnes a zda nejsou nějaké překážky k výkonu jeho práce. Standup zlepšuje komunikaci v týmu, identifikuje případné překážky, které je možno ihned začít řešit. Podporuje také rychlé rozhodovaní a eliminuje potřebu dalších schůzek. Standup je výhradně určen pro potřeby vývojového týmu, kdokoliv jiný je vítán, ale nesmí žádným způsobem ovlivňovat jeho průběh.
- **Sprint Review** – je předposlední aktivitou sprintu. Jejím účelem je zkontrolovat výsledek sprintu, který je následně scrum týmem prezentován klíčovým zainteresovaným stranám. Jedná se o kontrolu úspěšně provedené či dosud neprovedené práce v uplynulém sprintu (Schwaber a Sutherland, 2020).
- **Retrospektiva** (sprint retrospective) **–** základ retrospektivy je od členů týmu nasbírat informace o současném stavu, co funguje, nefunguje a co je potřeba zlepšit. Na základě získaných informací se identifikují příčiny případných problémů či prostor pro zlepšení a určí se konkrétní kroky k vyřešení. Cílem retrospektivy je naplánovat způsoby, jakými zvýšit kvalitu a efektivitu práce.

3.3.3 Extrémní programování

Počátky extrémního programování neboli XP se datují do 90. let, kdy se Kent Beck snažil najít lepší způsob procesu vývoje software, když pracoval na projektu pro

automobilovou společnost Daimler Chrysler. Jeho nový, extrémní, způsob se ukázal jako velmi úspěšný (Sauter, 2006).

XP využívá obecně známé principy a postupy, které dotahuje do extrémů. Neustále reviduje zdrojový kód, všichni neustále testují, pravidelně se refaktoruje, nechává se systém co nejjednodušší, ale tak aby stále fungoval, pracuje se v krátkých iteracích (Buchalcevová, 2009).

3.3.3.1 Charakteristika

Extrémní programování neboli XP je snadná metodika pro týmy, které vyvíjejí software a musí se popasovat s rychle a často měnícím se zadáním (Buchalcevová, 2009).

XP je metodikou vývoje softwaru, která se zabývá možným rizikem na všech úrovních vývojového procesu, toto riziko se snaží snižovat na minimum, zlepšuje schopnosti reagovat na změny a zlepšuje produktivitu během celého životního cyklu systému. Možná rizika jsou např. zpoždění harmonogramu, zrušený projekt, nepochopení zadání, změny zadání nebo přemíra funkcí (Beck, 2002).

3.3.3.2 Role

XP z agilních metodik nejvíce pracuje s lidským faktorem a různým složení týmů. Je určena pro malé až středně velké týmy (2–10 vývojářů).

Role jsou následující (Kadlec, 2004):

- **Vývojář** – píše jednotkové testy, programuje a refaktoruje. Vývojář je v XP brán jako srdce týmu.
- **Zákazník** – úzce spolupracuje s vývojáři, aby věděli, na čem mají prioritně pracovat. Spoluzodpovídá za vývoj.
- **Tester** – role testera v XP se liší od ostatních agilních metodik. Testy píše vývojář, tester pomáhá zákazníkovi s funkčními testy.
- **Stopař** – má největší přehled o projektu, analyzuje zpětnou vazbu a sbírá z ní informace.
- **Kouč –** stará se o dodržování zásad XP. Udržuje funkční komunikaci uvnitř týmu.
- **Konzultant** – je odborníkem na konkrétní oblast, pro kterou je software vyvíjen.

- **Velký šéf** – vedoucí týmu. Pokud nejsou problémy nezasahuje. Udržuje motivaci v týmu.

3.3.3.3 Principy

Abychom byli úspěšní, je třeba mít styl zachovávající důslednou sadu hodnot, které slouží potřebám člověka i společnosti. Pro zamezení sporu mezi krátkodobými cíli jednotlivců a dlouhodobými sociálními cíli, vyvinuly společnosti sdílenou sadu hodnot, které byly podpořeny mýty, rituály, tresty a odměnami. Bez těchto hodnot mají lidé tendenci vrátit se ke svým krátkodobým osobním zájmům.

XP stojí na čtyřech základních hodnotách (Beck, 2002):

1. **Komunikace** – na základě špatných otázek a odpovědí dojde ke špatnému nebo žádnému rozhodnutí v kritické oblasti. XP využívá k udržení komunikace např. testování jednotek, párové programování, odhadování úkolů, a také kouče, který se snaží eliminovat výpadky mezi lidmi.
2. **Jednoduchost** – je lepší udělat jednoduchou věc dnes a udělat případnou změnu zítra, než udělat věc složitější a komplexnější, která se nakonec nemusí vůbec využít. Jednoduchost a komunikace mají vzájemně doplňující vztah. Čím jednodušší systém, tím méně je třeba o něm komunikovat.
3. **Zpětná vazba** – brzké uvedení nejhodnotnější části systému do provozu je jednou ze strategií plánovacího procesu. Tím dostávají vývojáři konkrétní zpětnou vazbu o kvalitě svých rozhodnutí a vývojovém procesu. Konkrétní zpětná vazba jde ruku v ruce s komunikací a jednoduchostí. Čím více konkrétní zpětné vazby máme, tím snazší je komunikace. Pokud je komunikace jasná, budeme vědět, co měřit a testovat. Jednodušší systémy se snáze testují.
4. **Odvaha** – typickým příkladem odvahy při vývoji je zahodit zdrojový text po celém dni psaní, které zdánlivě nikam nevede a začít druhý den od znova, to chce odvahu. Odvaha v kombinaci s komunikací, jednoduchostí a zpětnou vazbou se stává velmi cennou. Komunikace podporuje odvahu, protože nabízí možnost experimentu s vyšším rizikem, ale také ziskem. Jednoduchost podporuje odvahu, protože s jednoduchým systémem si můžeme dovolit být více odvážní. Zpětná vazba

podporuje odvahu, protože máme jednotkové testy, díky kterým si můžeme ověřit, že jakákoliv změna zachovala správnou funkčnost systému.

3.3.3.4 Praktiky

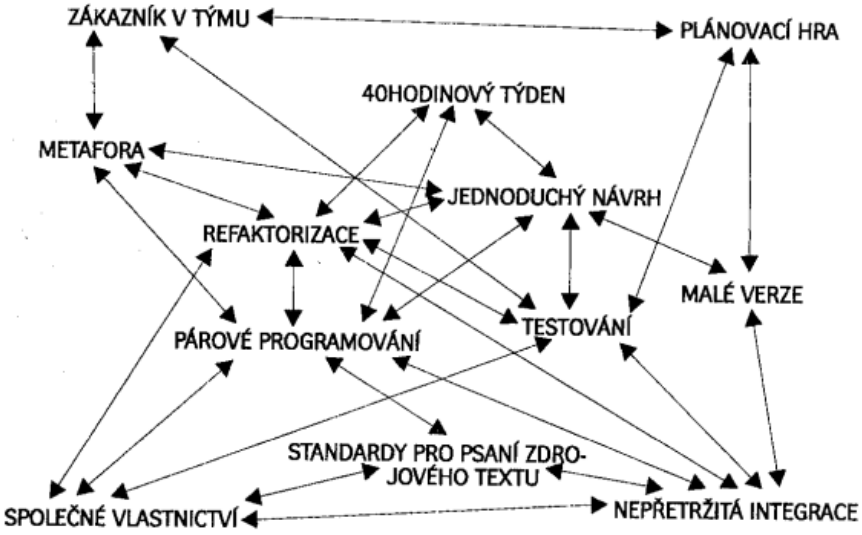
V rámci XP se dodržuje 12 základních praktik vývoje. V místě, kde existuje slabá stránka postupu, zakryjí tuto slabinu silné stránky dalších postupů. Nutno říci, že se nejedná o originální postupy XP, ale ty, které XP využívá, o postupy, které se dříve celkově využívaly a většina z nich byly nahrazeny složitějšími procesy, jakmile byla objevena jejich slabina. Žádný postup neobstojí sám o sobě a potřebuje také ostatní k tomu, aby jej udržel v rovnováze.

Základní praktiky při vývoji v XP jsou (Beck, 2002):

1. **Plánovací hra** – slouží k řízení projektu. Zákazník prostřednictvím plánovací hry rozhoduje o šíři zadání a stanovuje priority požadavků. Vývojáři během plánovací hry stanovují časový odhad pro vývoj jednotlivých požadavků, radí zákazníkovi v technických aspektech a vytvářejí podrobný harmonogram nadcházející iterace.
2. **Malé verze** – vydávané verze by měly být co nejmenší a měly by obsahovat nejhodnotnější funkcionalitu tak, aby celek stále dával smysl. XP dodává malé verze v kratších intervalech místo objemnějších verzí po půl roce.
3. **Metafora** – zastupuje význam architektury. Je třeba zdůraznit cíl architektury, čehož se XP snaží dosáhnout tím, že vytvoří příběh, ve kterém tým bude pracovat, který se snadno sděluje a zdokonaluje. Metafora pomáhá v projektu všem pochopit základní prvky systému a vztahu mezi nimi.
4. **Jednoduchý návrh** – vyvíjený software má neustále co nejjednodušší podobu, která pokrývá nejaktuálnější požadavky zákazníka. Nic se nevyvíjí dříve, než je to potřeba, protože to nemusí být nikdy potřeba.
5. **Testování** – je nedílnou součástí vývoje, bez které nemůže program existovat. Vývojáři píšou testy jednotek, aby měli jistotu, že to, co píšou, funguje tak, jak zamýšleli. Zákazníci píšou testy funkcionality, aby měli důvěru v to, že program bude pracovat tak, jak požadují.

6. **Refaktorizace** – znamená úprava stávajícího programu, která může program zjednodušit. Pokud je program správně napsán, je možno refaktorovat program po relativně malých částech a s malým rizikem.
7. **Párové programování** – zdrojový kód píšou dva vývojáři u jednoho počítače s jednou klávesnicí a myší. Ten, který kód píše, přemýšlí o nejlepším způsobu implementace dané části, ten druhý přemýšlí o tom, zda je jejich přístup správný nebo zda je možné návrh zjednodušit tak, aby případný problém prostě zmizel. Seskupení do párů je dynamické, ráno může tvořit vývojář pár s jedním, odpoledne s jiným.
8. **Společné vlastnictví** – všichni mají zodpovědnost za celý vyvíjený software. Pokud vývojář vidí nějakou příležitost ke zlepšení části kódu nebo opravy, udělá to bez čekání na odpovědnou osobu.
9. **Nepřetržitá integrace** – neustále se integruje a testuje, nejméně jednou za den. Pro integraci je vyhrazen samostatný počítač. Vývojáři si na něj nasadí aktuální verzi, stáhnout do ní své změny a spouští testy, dokud všechny nejsou zelené. Pokud nějaký test nefunguje, opraví ho hned. Pokud nejsou schopni padlé testy opravit, pravděpodobně je jednodušší zahodit, co vytvořili a začít znovu.
10. **40hodinový týden** – XP nedovoluje pracovat přesčasy dva týdny po sobě. Jeden týden je v pořádku, pokud chceme něco dohánět. Pokud ale je třeba pracovat přesčasy druhý týden za sebou, tak zřejmě je chyba jinde a odpracované hodiny navíc to nespraví.
11. **Zákazník na pracovišti** – součástí týmu musí být skutečný zákazník, který bude systém opravdu používat. Zákazník je však pouze fyzicky oddělen o svého týmu, je po ruce vývojářům, ale to neznamená, že nebude schopen stíhat svou vlastní práci.
12. **Standardy pro psaní zdrojového textu** – vývojáři přeskakují z jedné části systému do druhé, vzájemně refaktorují části kódu. Dodržování standardů pro psaní zdrojového textu usnadňuje komunikaci mezi vývojáři.

Na následujících obrázku lze vidět, jak jsou všechny využívané praktiky XP vzájemně propojeny.



Obr. 5 Vzájemné propojení XP praktik (Beck, 2002)

3.3.4 Feature Driven Development

Feature Driven Development (dále FDD) v češtině znamená vývoj řízený funkcemi nebo také užitnými vlastnostmi softwaru. Základy FDD položil Petr Coad, jakožto spojení iterativního přístupu a praktik, které se osvědčili v průmyslu. Později pak společně s pány Jeff de Luca a Stephen Palmer metodiku rozšířil a zdokonalil. Jejich snahou bylo, aby byl vývoj softwaru díky FDD jednodušší, efektivnější a čitelnější (Coad et al., 1999).

3.3.4.1 Charakteristika

Metodika FDD se zaměřuje na návrh a implementaci softwaru. Její výhody spočívají v neustálém monitoringu stavu projektu, v dohlížení na kvalitu vývoje a v častých dodávkách fungujícího softwaru zákazníkovi. Metodika se skládá z pěti po sobě jdoucích procesů, z nichž poslední dva se iterativně opakují (Palmer, Felsing, 2002; Cockburn, 2005).

Hlavní myšlenkou FDD je rozdělit celý systém na množinu užitných vlastností, které se následně postupně implementují (Palmer, Felsing, 2002). Užitnou vlastností (feature) si lze představit výsledek, který je přínosný pro zákazníka, je měřitelný, srozumitelný a realizovatelný v krátkých iteracích (Buchalcevová, 2005b).

3.3.4.2 Role

FDD definuje jednotlivé role v rámci týmu. Každá role má jasně dáno, co je jejím úkolem v jednotlivých fázích vývoje. V jedné roli může působit více členů týmu, stejně tak každý člen týmu může zastávat více rolí. Z hlediska důležitosti se role dělí na klíčové, podpůrné a ostatní.

Klíčové role jsou (Palmer, Felsing, 2002):

- **Projektový manažer** (Project Manager) zastává roli vedoucího celého projektu. Zodpovídá především za administrativní a finanční stránku projektu. Mezi jeho kompetence patří také rozsah projektu, časový harmonogram a personální obsazení v týmu.
- **Hlavní architekt** (Chief Architect) je zodpovědný za celkový návrh systému. Zastává roli koordinátora ostatních členů při navrhování a má poslední a rozhodující slovo. V případě složitého projektu se jeho role může rozdělit na dvě – technický architekt a doménový architekt.
- **Vývojový manažer** (Development manager) řeší každodenní problémy týkající se vývoje. Zastává roli koordinátora mezi jednotlivými vývojovými týmy, řeší případné spory a rozdělení prostředků mezi týmy.
- **Hlavní vývojář** (Chief Programmer) je plnohodnotný člen týmu, současně řídí jednotlivé menší týmy vývojářů, obvykle 3-5 členů. Hlavní vývojář se podílí na prvotní analýze, stanovení požadavků na systém a jeho návrhu.
- **Vlastník třídy** (Class Owner) je členem týmu vedeného hlavním vývojářem. Vlastník třídy je zodpovědný za konkrétní třídu, kterou vlastní a spravuje po celou dobu vývoje. Z vlastníků třídy se tvoří takzvané týmy pro vlastnosti, jejichž složení se může s každou iterací měnit.
- **Doménový expert** (Domain Expert) dokonale rozumí oblasti, pro kterou je software vyvíjen. Poskytuje týmu zpětnou vazbu a kontroluje, že projekt skutečně odpovídá požadavkům zákazníka.

Podpůrné role jsou (Palmer, Felsing, 2002):

- **Manažer pro dodávky** (Release Manager) má na starosti kontrolu postupu vývoje, shromažďuje informace o tom, co je již hotové a na čem se dále pracuje.

- **Jazykový specialista** (Language Guru) dokonale ovládá používaný programovací jazyk a své znalosti předává ostatním členům týmu. Jazykový specialista může být i externím konzultantem.
- **Konstruktér** (Build Engineer) je zodpovědný za tzv. build proces (sestavení celého systému) a všechny s tím související automatizované procesy (např. buildovací skripty, generace dokumentace apod.). Stará se také o správu verzovacího systému.
- **Nástrojář** (Toolsmith) má především na starosti tvorbu podpůrných programů, které usnadňují práci ostatním vývojářům.
- **Administrátor** (System Administrator) má na starost údržbu serverů a vnitřní infrastruktury. Podílí se s konstruktérem na buildovacím procesu a na nasazovaní nových verzí na ostatní prostředí.

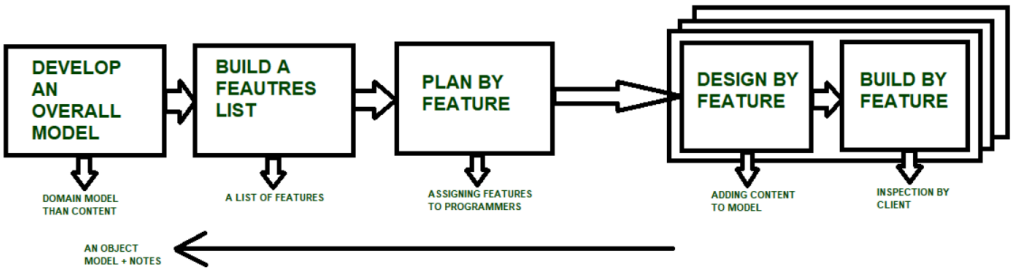
Ostatní role jsou (Palmer, Felsing, 2002):

- **Tester** testuje vyvíjený software a ověřuje, že odpovídá požadavkům zákazníka. Tester může být součástí týmu nebo pracovat v rámci externího oddělení.
- **Správce nasazení nových verzí** (Deployer) má na starosti nasazování nových verzí přímo u zákazníka.
- **Písař** (Writer) píše uživatelskou dokumentaci.

3.3.4.3 Procesy

Metodika FDD definuje pět na sebe navazujících procesů, které mají jednotný formát, jsou to *popis procesu, vstupní kritéria, úkoly, verifikace a výstupní kritéria*. Jedná se o pouze obecný rámec, kterého by se tým měl držet. FDD striktně neurčuje jak a pomocí jakých nástrojů by měl být daný krok proveden, pouze říká, co se má udělat. To, jakým postupem bude krok proveden, je v rukou vedoucích týmů, u kterých se klade větší důraz na zkušenosti. Tento přístup zamezuje pasivní následování jednotlivých předepsaných kroků.

Na následujících obrázku je uveden přehled procesů definovaných v této metodice, které jsou dále charakterizovány (Palmer, Felsing, 2002).



Obr. 6 Metodika FDD (Palmer, Felsing, 2002)

- **Vypracování celkového modelu** – pokrývá hrubý model celé domény (Buchalcevová, 2005a). Úkoly této fáze jsou především studium domény, prostředí, ve kterém bude plánovaný systém nasazen a vytvoření celkového modelu. Výstupem této fáze je diagram tříd a alternativy řešení.
- **Sestavení seznamu užitných vlastností** – vytvoří se seznam vlastností podle požadavků zákazníka, který pokrývá co největší možnou oblast navrhovaného systému. Během vývoje lze další vlastnosti přidávat nebo odebírat. Výsledkem je setříděný seznam užitných vlastnosti.
- **Plánování užitné vlastnosti** – se řídí seznamem užitných vlastností vytvořeným v předchozím procesu. Sestaví se plánovací tým, který naplánuje pořadí implementace jednotlivých užitných vlastnosti. Ke každé vlastnosti se přiřadí hlavní vývojář a určí se termín dokončení. Výstupem je hotový plán vývoje (Buchalcevová, 2005a).
- **Návrh užitné vlastnosti** – tento a následující proces se iterativní opakují. Tuto fázi řídí hlavní vývojář, kterému byla konkrétní užitná vlastnost přidělena. Pro danou vlastnost vybere potřebné třídy a kontaktuje odpovídající vlastníky tříd. Ti vytvoří dočasný tým. Vypracují návrhový balíček, který obsahuje detailně zpracovaný sekvenční diagram a rozšířené jednotlivé třídy a metody v diagramu tříd.
- **Realizace užitné vlastnosti** – v poslední fázi se implementují jednotlivé vlastnosti. Vlastníci tříd realizují metody, vytváří testovací případy. Po úspěšném testování se třídy vloží do verzovacího systému.

3.3.4.4 Praktiky

Metodika FDD je postavena na klíčovém souboru praktik. Nejedná se o nové praktiky, ale přínos FDD spočívá v jejich správné kombinaci, kdy se dané praktiky a techniky navzájem doplňují a ovlivňují. Jejich využívání vede tým k efektivnosti. Podle Palmera, Felsinga (2002) však není nutné praktiky zavádět všechny najednou. Je třeba se přizpůsobit aktuální situaci, znalostem, zkušenostem týmu a aplikovat praktiky postupně.

Praktiky v rámci FDD jsou (Palmer, Felsing, 2002):

- **Doménové objektové modelování** – tvoří diagramy tříd UML. Pro zachycení chování se objektů se využívají sekvenční diagramy, které zachycují vztahy mezi nimi. Vytvoření takového modelu má zabránit nekonzistenci a nedorozumění mezi jednotlivými týmy. Doménový objektový model reprezentuje celkový rámec, do kterého se vkládá funkcionalita jednotlivých užitných vlastností. To pomáhá udržet konceptuální strukturu systému (Buchalcevová, 2005a).
- **Vývoj podle užitných vlastností –** stěžejním pojmem FDD metodiky je *vlastnost.* Zaměřením prioritně na dané vlastnosti se vývoj věnuje těm požadavkům, které od zákazníka opravdu má. Tím se eliminuje možnost odchýlení se od těchto požadavků.
- **Vlastnictví tříd –** existují dva typy vlastnictví tříd, individuální a kolektivní. FDD prosazuje individuální vlastnictví tříd, kdy spoléhá na zodpovědnost jednotlivých vlastníků. Při přidávání nové funkcionality vlastník třídy ověřuje, že se nezmění účel dané třídy a změny jsou provedeny správně. Problém může nastat z hlediska kapacity vývojářů, nebo při odchodu vlastníka dané třídy. To se snaží FDD eliminovat pomocí dalších praktik (Buchalcevová, 2005a).
- **Týmy pro užitné vlastnosti –** jsou sestavovány vždy pro implementaci určité vlastnosti, protože každá vlastnost většinou zasahuje do více tříd. Jelikož jsou vlastnosti implementovány paralelně, může být každý člen více týmů najednou. Týmy sestavují jednotliví vedoucí vývojových týmů, následně jsou po dokončení implementace rozpuštěny.
- **Inspekce –** provádí se inspekce nejen zdrojového kódu, ale také modelů, které vznikají během návrhu. Inspekce slouží k odhalovaní chyb, kterým se během vývoje prakticky nedá vyhnout.

- **Pravidelné buildy –** kód všech hotových užitných vlastností se v pravidelných intervalech, týdny i dny, integruje do jednoho celku. Pravidelná integrace může včas odhalit případné chyby. Také po každé integraci může tým předvést současný stav systému zákazníkovi a získat tak zpětnou vazbu.
- **Řízení konfigurací –** uchovávání verze dokumentu se specifikací požadavků je důležité, protože se jedná o předmět smlouvy mezi dodavatelem a zákazníkem. Také je třeba udržovat verze jednotlivých analytických a návrhových meziproduktů, testovacích případů, různé dokumentace apod.
- **Reporting/viditelnost výsledků –** metodika FDD se snaží o minimalizaci činností, jako jsou sběr informací o stavu projektu, různé porady o stavu projektu, nepotřebné formuláře, a to nejlépe automatizovaným způsobem sběru těchto informací. K tomu slouží jednoduchá metoda FDD, která sbírá přesné a spolehlivé informace o stavu projektu, jejíž výstupem jsou zprávy v různých formátech (Buchalcevová, 2005a).

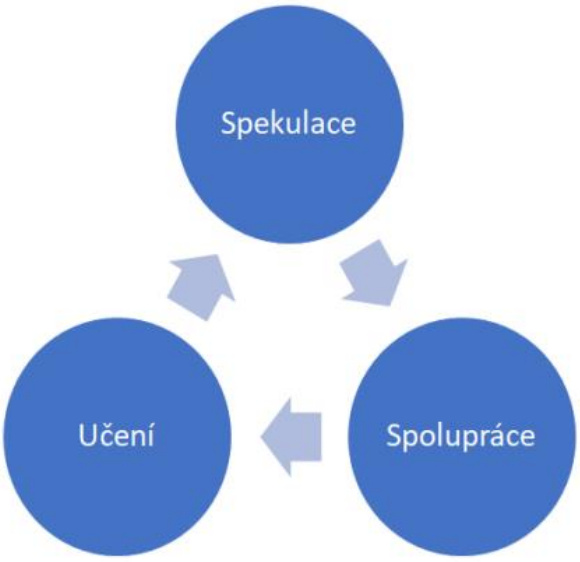
3.4 Přehled dalších metodik a modelů

Adaptivní vývoj softwaru

Adaptive Software Development neboli Adaptivní vývoj softwaru (dále ASD) navrhl v roce 1973 Jim Highsmith jako techniku k výrobě komplexního systému. V tomto procesu nejsou žádné předem naplánované kroky. Cílem je rychlý vývoj komplexního systému, který většinou nelze rozdělit na menší samostatné části. ASD počítá s častými změnami zadání. Komunikace mezi týmem a zákazníkem je pro tuto metodiku nezbytná.

Cyklus ASD obsahuje tři fáze (Highsmith, 2000):

1. **Spekulace** – nahrazuje slovo plán. Tým při spekulaci bere v potaz možnost řešení složitých problému, které mohou při vývoji nastat. Spekulace podporuje experimentování a výzkum.
2. **Spolupráce** – komplexní systém přináší velký objem informací. Efektivní spolupráce se zákazníkem a komunikace hraje v týmu klíčovou roli.
3. **Učení** – tato fáze je nezbytnou součástí pro úspěšný projekt.



Obr. 7 Proces ASD (Highsmith, 2000)

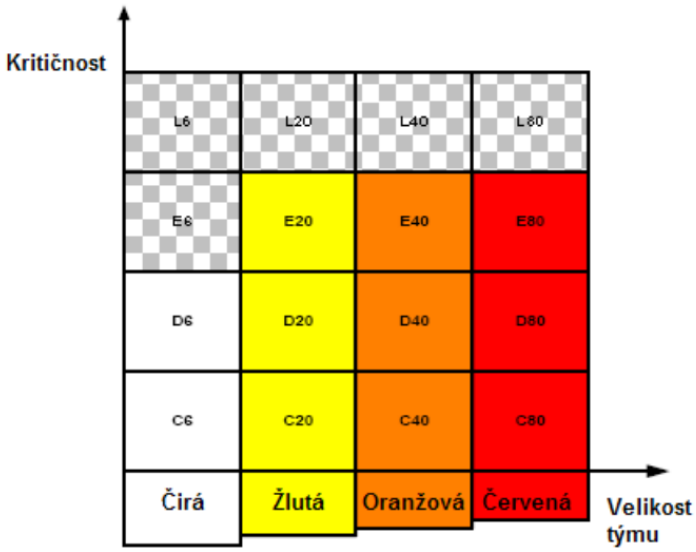
Po každé iteraci následuje hodnocení a sumarizace výsledků. Tým se seznámí s výsledky posledního cyklu, novými změnami v projektu a s dalšími požadavky od zákazníka. Základním kamenem je proces učení, kdykoliv se může nějaké rozhodnutí označit za nesprávné a projekt přepracovat. Tým se takto poučí do příště.

Crystal metodiky

Crystal není název pouze jedné metodiky, jedná se o označení celé skupiny metodik, které vymyslel Alistair Cockburn. Jednotlivé metodiky nesou jméno podle barvy, odlišují se svou vhodností pro týmy podle velikosti a podle složitost jejich projektu. Čím je barva tmavší, tím je metodika robustnější a vhodnější pro složitější projekty. Jednotlivé barvy jsou čirá, žlutá, oranžová, červená, hnědá, modrá a fialová. Metody neudávají přesné postupy, ale obsahují řadu doporučení, jak je přizpůsobit vlastním potřebám.

Vhodná metodika se zvolí na základě hodnot tří vlastností projektu. Těmi jsou velikost vývojového týmu, kritičnost projektu a priorita projektu. Priorita projektu je vlastnost, která vyzdvihuje nejvýznamnější stránku projektu. Je třeba se rozhodnout, zda se bude průběh projektu optimalizovat pro maximální produktivitu vývoje nebo zda se dá přednost důkladnému měření probíhajícího projektu.

Následující obrázek ilustruje volbu vhodné metodiky.

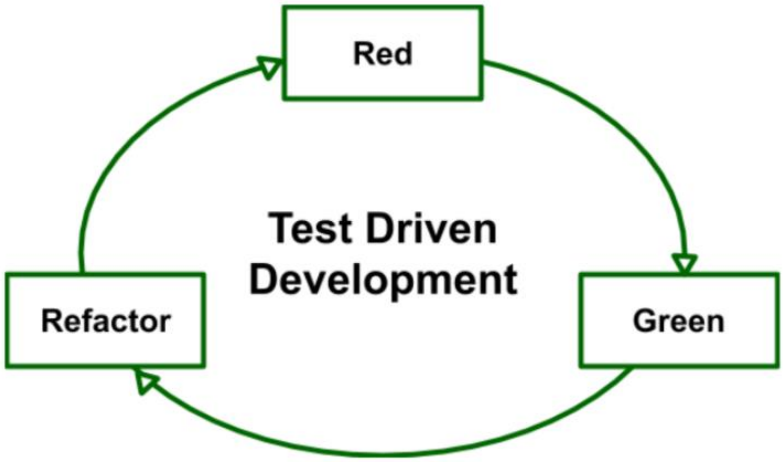


Obr. 8 Volba vhodné Crystal metodiky (Abrahamsson et al., 2002)

Čtverečkované oblasti jsou takové, které jsou nedosažitelné nebo pro ně neexistuje příslušná Crystal metodika. Robustnější metodiky jsou postupně navrhovány a ověřovány v praxi (Abrahamsson et al., 2002).

Test Driven Development

Podstata Test Driven Development neboli TDD spočívá v tom, že testování není považováno jen za jednu z fází vývoje, ale je především nejdůležitější fází vývoje. Základní myšlenkou TDD je tedy před započetím implementace dané funkcionality napsat test, který ji spolehlivě otestuje. Poté se začne s implementací a implementuje se tak, aby byl splněn test, nic navíc. Následně se optimalizuje kód testu i funkcionality, tím je hotová jedna iterace (Beck, 2001; GeeksforGeeks, 2020). Cílem toho způsobu je snížit časovou náročnost testů a urychlit testování nové funkcionality (Šochová, Kunce, 2019).



Obr. 9 Proces TDD (GeeksforGeeks, 2020)

Aby mohly být jednotlivé části funkcionality kvalitně otestovány, musí test splňovat několik podmínek (Beck, 2001):

- **Rychlost** – nepouští se pouze jeden test, ale celá sada, testy musí být jednoduché a probíhat rychle.
- **Jednoduchost** – test se musí zaměřovat na konkrétní testovanou funkcionalitu. Za předpokladu, že by byl test moc složitý, rozdělí se do menších částí.
- **Kvalita** – pro kvalitní testování jsou zapotřebí kvalitní testy. Proto se na konci každé iterace optimalizuje kód testu.
- **Automatizace** – jelikož se pouští testy jako jedna sada, musí test probíhat bez jakékoliv interakce od uživatele.
- **Nezávislost** – jednotlivé testy musí být vzájemně nezávislé a také nezávislé na prostředí a pořadí spouštění testů.

Lean Development

V názvu metodiky je slovo štíhlý, a to je také způsob, jakým tato metodika přistupuje k vývoji softwaru. To znamená, že je třeba dělat věci, jen když jsou potřeba, omezit rozdělanou práci a soustředit se pouze na to, aby jednotlivé požadavky byly doručeny co nejdříve. Lean Development nepopisuje, jak přesně vyvíjet, ale poskytuje řadu pravidel a principů, s nimiž bude vývoj efektivnější a zároveň levnější.

Obecná pravidla Lean Developmentu jsou následující (Kadlec, 2004):

- odebrat vše, co je navíc
- mít minimální zásoby
- maximalizovat tok
- vyvíjet jen tehdy, když je poptávka
- rozhoduje se tým
- uspokojit zákazníka je hlavní cíl
- zavést zpětnou vazbu
- odstranit lokální optimalizaci
- mít partnerství s dodavateli
- mít možnost se neustále zlepšovat

Tato metodika je některými autory chápána spíše jako samostatná metoda, která má svou vlastní kategorii a je rovnocenná agilním metodikám, jiní ji pod agilní metodiky zařazují (Ballard, 2003).

Rational Unified Process

Jedná se o hybridní proces, který kombinuje znaky obecných modelů procesů. Znázorňuje optimální postupy při specifikaci a návrhu projektu a podporuje prototypování a inkrementální dodávání. Model je založen na iterativním a přírůstkovém procesu, kdy každá iterace splňuje veškeré prvky vývoje software, těmi jsou plánování, analýza a návrh, vývoj, integrace a testování, provoz a údržba. Iterace vytvářejí tzv. základní linie, které jsou složeny z částečně kompletní verze finálního softwaru a příslušné dokumentace. Základní linie jsou na sebe vrstveny, dokud nevznikne finální verze systému. Rozdíl mezi dvěma liniemi se označuje jako přírůstek neboli inkrement.

Každá iterace obsahuje pět základních postupů (Arlow, Neustadt, 2007):

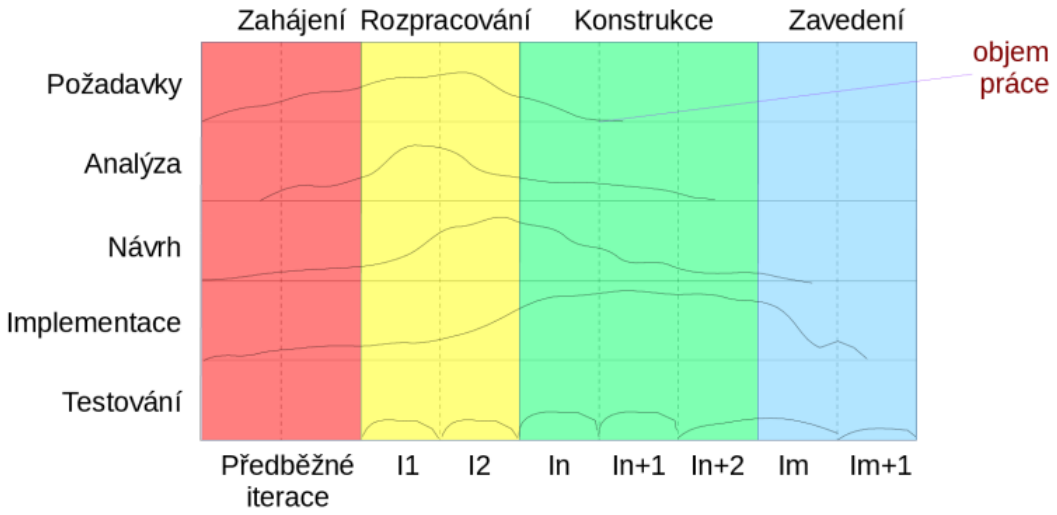
1. **Požadavky** – specifikace funkčnosti systému.
2. **Analýza** – upřesnění požadavků na systém a jejich strukturování.
3. **Návrh** – realizace konkrétních požadavků a vytvoření architektury systému.
4. **Implementace** – samotný vývoj systému.

5. **Testování** – ověření funkčnosti stávající implementace systému a validace splnění požadavků.

RUP je model, který se rozděluje na čtyři fáze. Jednotlivé fáze modelu souvisejí spíše s podnikový hledisky než s technickými, jako tomu je například u vodopádového modelu.

Fáze modelu RUP (Sommerville, 2003):

1. **Zahájení** – období plánování projektu.
2. **Rozpracování** – zpracování architektury systému.
3. **Konstrukce** – počátky provozu schopnosti.
4. **Zavedení** – nasazení kompletního produktu u zákazníka.



Obr. 10 RUP proces (Arlow, Neustadt, 2007)

3.5 Vícekriteriální analýza variant

Vícekriteriální analýza variant (dále VAV) nebo také vícekriteriální hodnocení variant je vedle vícekriteriálního programování dalším způsobem řešení úloh vícekriteriálního rozhodování. Je to proces rozhodování mezi možnými variantami na základě zvolených hodnotících kritérií.

Hlavním cílem VAV je nalezení optimální varianty a také uspořádání variant od nejlepší po nejhorší. Nejlepší varianta je nejblíže k ideální variantě a nejdále od bazální varianty. Ideální variantou se rozumí taková varianta, která má nejlepší hodnoty ve všech

stanovených kritériích. Naopak bazální varianta je taková, která má nejhorší hodnoty pro zvolená kritéria. Mezi variantami nelze najít takovou, která by jednoznačně dominovala, tudíž ideální varianta neexistuje a nejlepší řešení se nazývá kompromisní (Fotr, Švecová, 2016).

3.5.1 Metody stanovení vah kritérií

Nutnou součástí vícekriteriálního rozhodování je stanovení vah jednotlivých kritérií. Obecně platí, že čím vyšší váhu má dané kritérium, tím větší je jeho význam při rozhodování (Schneiderová Heralová et al., 2011). Je požadováno, aby váhy byly nezáporné hodnoty, a také aby byly normované, tzn. jejich součet se rovná 1 (Klicnarová, 2010).

Metody stanovení vah kritérií lze dělit následovně (Fotr, Švecová, 2016):

- **Metody přímého stanovení vah –** tyto metody patří mezi nejjednodušší. Konkrétní váhy jsou přiřazovány jednotlivým kritériím na základě vlastních zkušeností a dostupných informací. Patří sem např. bodovací metoda, porovnání pomocí preferenčního pořadí.
- **Metody párového porovnávání –** podstatou této metody je zjišťování vztahů mezi dvojicemi kritérií. Patří sem např. Fullerova metoda, Saatyho metoda.
- **Metody postupného rozvrhu vah –** využívá se za předpokladu velkého množství kritérií. Nejprve jsou kritéria hodnocena v rámci skupin na základě příbuznosti. Poté se ohodnotí dané skupiny a výsledná váha pro jednotlivá kritéria se získá jako součin normovaných vah daného kritéria a příslušné skupiny. Patří sem např. kompenzační metoda.

3.5.1.1 Bodovací metoda

Bodovací metoda je metodou přímého stanovení vah. Podstatou této metody je stanovení důležitosti jednotlivých kritérií určitým počtem bodů v rámci určené bodovací stupnice. Pro body může být využita i desetinná čísla a více kritérií lze přiradit stejnou hodnotu (Jablonský, 1994).

Každé kritérium je ohodnoceno určitým počtem bodů, čím důležitější kritérium, tím více bodů získá. Při využití stupnice 0–10 je hodnocení 0 pro zcela bezvýznamné kritérium pro danou variantu a hodnocení 10 se považuje za absolutně důležité (Šubrt, 2011).

Výpočet vah je třeba normalizovat podle následujícího vztahu:

$$v_j = \frac{b_j}{\sum_{j=1}^{n} bj}$$

3.5.1.2 Saatyho metoda

Saatyho metoda je metodou kvantitativního párového porovnávání kritérií. Pro hodnocení kritérií se využívá devítibodová stupnice (lze využít i sudé mezistupně) (Schneiderová Heralová el al., 2011):

- 1 – rovnocenná kritéria *i* a *j*
- 3 – slabá preference *i* před *j*
- 5 – silná preference *i* před *j*
- 7 – velmi silná preference *i* před *j*
- 9 – absolutní preference *i* před *j*

Kritérium	Krit. 1	Krit. 2	Krit. 3	Krit. 4	Krit. 5	Krit. 6	Geometrický průměr	Normovaná váha
Kritérium 1	1	7	3	9	8	2	3,80	0,42
Kritérium 2	1/7	1	1/4	3	2	1/2	0,69	0,08
Kritérium 3	1/3	4	1	7	5	3	2,28	0,25
Kritérium 4	1/9	1/3	1/7	1	1/3	1/7	0,25	0,03
Kritérium 5	1/8	1/2	1/5	3	1	1/5	0,44	0,05
Kritérium 6	1/2	2	1/3	7	5	1	1,51	0,17
Součet vah								*1,00*

Obr. 11 Stanovení vah pomocí Saatyho matice (Schneiderová Heralová et al., 2011)

Pro hodnocení se sestaví tabulka s jednotlivými kritérii, které jsou v řádku a stejně tak ve sloupci. Jednotlivá kritéria se mezi sebou vzájemně porovnají na základě preferencí. Významnější kritérium v řádku oproti kritérium ve sloupci získá celé číslo, naproti tomu se pro sloupcové kritérium v řádku zapíše převrácená hodnota. Na diagonále jsou porovnávána identická kritéria, tudíž mají hodnotu 1.

Pro každé kritérium se vypočte geometrický průměr, který představuje nenormované váhy kritérií. Stanovení nenormovaných vah je také možno provést pomocí aritmetického

průměru. Normovaná kritéria jsou vypočítána dělením nenormované váhy kritéria sumou nenormovaných vah všech kritérií (Schneiderová Heralová el al., 2011).

3.5.2 Metody stanovení pořadí variant

Cílem metod vícekriteriálního rozhodování je stanovit pořadí jednotlivých variant na základě zvolených kritérií. Nejlépe hodnocená varianta se považuje za nejlepší kompromisní variantu. Metody vícekriteriálního rozhodování lze členit podle počtu rozhodovatelů zapojených do rozhodovacího procesu. Rozhodovatel může být jeden nebo jich může být více (Triantaphyllou, 2000).

Dále se metody stanovení pořadí variant člení podle požadovaného druhu informací (Friebelová, 2007):

- **Metody vyžadující znalost aspirační úrovně** – do této skupiny se řadí metody konjunktivní, disjunktivní a metoda PRIAM. Aspirační úrovně jednotlivých kritérií jsou porovnávány s kriteriálními hodnotami všech variant. Varianty se dělí na dvě skupiny, a to efektivní, ty, které vykazují lepší či stejné kriteriální hodnoty v porovnání s aspirační úrovní a neefektivní, ty, které vykazují horší kriteriální hodnoty, než má aspirační úroveň. Při postupném zpřísňování aspirační úrovně může zůstat pouze jedna, kompromisní varianta.
- **Metody vyžadující ordinální informaci** – do této skupiny se řadí metoda pořadí, metoda lexikografická, permutační nebo také metoda ORESTE.
- **Metody požadující kardinální informaci** – se člení dále na podskupiny podle principu, na kterém je hodnocení založeno.
    - metody maximalizace užitku – metoda bodovací, metoda váženého součtu, metoda AHP (Analytický hierarchický proces)
    - metody minimalizace vzdálenosti od ideální varianty – metoda TOPSIS
    - metody preference – PROMETHEE, ELECTRE
    - metody mezní míry substituce.

3.5.2.1 Metoda váženého součtu

Weighted Sum Approach neboli metoda váženého součtu (zkráceně WSA) je nejstarší a velmi používaná metoda pro určení pořadí variant. Je založena na konstrukci

lineární funkce užitku na stupnici od 0 do 1. Nejlepší variantě je přirazena hodnota užitku 1 naopak nejhorší variantě je přirazena hodnota užitku 0. Zbylé varianty mají užitek mezi těmito krajními hodnotami.

Prvky y_{ij} vstupní kriteriální matice se nahradí hodnotami $y`_{ij}$, které představují užitek variant X_i při hodnocení podle kritéria Y_j. D_j představuje nejnižší hodnotu j-tého kritéria a H_j představuje nejvyšší hodnotu j-tého kritéria.

- získání hodnot pro maximalizační kritéria

$$y`_{ij} = \frac{y_{ij} - D_j}{H_j - D_j}$$

- získání hodnot pro minimalizační kritéria

$$y`_{ij} = \frac{H_j - y_{ij}}{H_j - D_j}$$

- celkový užitek varianty X_i se vypočítá jako vážený součet dílčích užitků podle jednotlivých kritérií

$$u(X_i) = \sum_{j=1}^{k} v_j y`_{ij}$$

Varianty jsou následně uspořádány podle klesajícího užitku $u(X_i)$ (Jablonský, 2004).

3.5.2.2 Metoda TOPSIS

Princip metody TOPSIS spočívá ve výběru nejlepší varianty a to takové, která má minimální vzdálenost od ideální varianty a zároveň maximální vzdálenost k bazální variantě. Pro výpočet je nutné, aby všechna kritéria byla maximalizační. Není-li tomu tak, je nutné je na maximalizační převést (Friebelová, Klicnarová 2007).

Výpočet metody TOPSIS je rozdělen do šesti kroků (Fiala, 2008):

1. **Stanovení kriteriální matice $\boldsymbol{Y} = (\boldsymbol{y_{ij}})$**

   Sestaví se kriteriální matice, která se upraví tak, aby byla všechna kritéria maximalizační. Pro kritéria minimalizačního charakteru se určí nejhorší maximální

hodnota. Od nejhorší maximální hodnoty odečteme hodnoty kritéria pro danou alternativu.

2. Získaná upravená matice

Jedná se o mezi výpočet pro normalizovanou kriteriální matici **R**. Jednotlivé hodnoty kriteriální matice se umocní na druhou $y^2{}_{ij}$, vypočte se jejich suma $\sum y^2{}_{ij}$ a následně odmocnina $\sqrt{\sum y^2{}_{ij}}$.

3. Vytvoření normalizované kriteriální matice $\boldsymbol{R} = (\boldsymbol{r_{ij}})$

Normalizovaná matice **R** se vypočítá tak, že každá hodnota kritéria (matice krok č.1) se vydělí odmocninou sumy z předchozího kroku.

$$r_{ij} = \frac{y_{ij}}{\sqrt{\sum y^2{}_{ij}}}$$

4. Vytvoření vážené kriteriální matice Z

Vážená kriteriální matice **Z** se získá tak, že každá hodnota ve sloupci matice **R** se vynásobí stanovenou vahou daného kritéria.

$$z_{ij} = w_j r_{ij}$$

5. Výpočet vzdáleností od ideální varianty

Z matice **Z** se získá ideální $h_1 = max\ z_{ij}$ a bazální $d_1 = min\ z_{ij}$ varianta pro každé kritérium. Dále se pro všechna kritéria vypočítá vzdálenost od ideální varianty.

$$d_i{}^+ = \sqrt{\sum_j (w_{ij} - H_j)^2}$$

A dále se vypočítá vzdálenost od bazální varianty.

$$d_i{}^- = \sqrt{\sum_j (w_{ij} - D_j)^2}$$

6. Výpočet relativního ukazatele vzdálenosti

Jako poslední se vypočítá relativní ukazatel vzdálenosti od bazální varianty. Varianta s nejvyšším ukazatelem se jeví jako optimální.

$$c_i = \frac{{d_i}^-}{\left({d_i}^+ + {d_i}^-\right)}$$

4 Vlastní práce

Vlastní práce se bude zabývat analýzou současné situace konkrétního vývojového týmu. Na základě výsledku metody TOPSIS bude pro daný tým vybrána vhodná agilní metodika. Zvolené metodiky pro výběr jsou Scrum, Extrémní programování a Feature Driven Development. Stěžejní částí této práce je následný návrh na implementaci vybrané metodiky pro daný tým a shrnutí tohoto návrhu.

4.1 Analýza současné situace

Tato kapitola představuje konkrétní vývojový tým, pro který bude vybrána vhodná agilní metodika a následně bude sepsán návrh pro zavedení té metodiky. Dále stručně popisuje složení týmu, jakým způsobem probíhá samotný vývoj, procesy týmu, konkrétně důležité týmové schůze nutné pro koordinaci jednotlivých členů a používané nástroje pro vývoj, testování, dokumentaci.

4.1.1 Charakteristika týmu

Firma, ve které tým pracuje, je nadnárodní společností zabývající se průmyslovou automatizací a digitální transformací procesů průmyslových firem. Zákazníkům pomáhá přinášet hodnotu, vylepšovat efektivitu a někdy i vytvářet nové trhy skrze výrobu hardware společně s vestavěným softwarem. Dále vytváří software na programování a provozování daného hardware. Také poskytuje služby v oblasti životního cyklu vybavení továren.

V rámci tvorby softwaru, tým, pro který bude v rámci Vlastní práce vypracován návrh na implementaci vybrané agilní metodiky, pracuje na informačním systému, který podporuje dashboardovou UI aplikaci, ukládá systémová a tzv. time series data a provádí další činnosti na pozadí. Tento systém je tzv. cloud native neboli je vyvíjený přímo pro cloud prostředí ve formě webové aplikace tvořené mini a micro servisami. Jedná se však o tým, který pracuje čistě na vrstvě operující s daty, prezentační vrstva je vyvíjena samostatným týmem. Tyto dva týmy úzce spolupracují na dodání kompletního řešení.

4.1.2 Složení týmu

Tým se skládá z deseti členů, a to vedoucího týmu, projektového manažera, vedoucího vývojářů, čtyř vývojářů, vedoucího testerů a třech testerů. Část týmu pracuje ve společné

kanceláři, část týmu pracuje z domova a do kanceláře dojíždí podle potřeby, případně na celotýmové schůze, ale i přesto se málokdy stane, že by se celý tým sešel pohromadě.

- *Vedoucí týmu* má na starosti celý tým, s tím spojené povinnosti řízení týmu a komunikaci s potenciálními nebo stávajícími zákazníky. Dále řeší společně s projektovým manažerem a za podpory HR praktické administrativní úkony pro tým (provoz týmu, výběr kandidátů, pohovory).
- *Projektový manažer* má na starosti více projektů. Připravuje dokumenty pro alokaci rozpočtu pro tým, řeší softwarové licence (nákup, obnova) a také má na starosti, aby daný tým následoval firemní procesy.
- *Vedoucí vývojářů* je pravou rukou vedoucího týmu, sám je velmi zkušeným vývojářem a rozděluje práci mezi ostatní vývojáře, kterým je zároveň k dispozici, když si neví rady. Dále vybírá technologie a určuje směřování architektury daného softwaru.
- *Vývojáři* vyvíjejí daný softwarový produkt na základě rozdělených úkolů vedoucím vývojářů. Jsou zodpovědní za vývoj a revizi kódu (code review) a jednotkové testování.
- *Vedoucí testerů* komunikuje převážně s vedoucím vývojářů a nadále rozděluje práci jednotlivým testerům. Sám je také testerem, udává směr a úroveň testování v týmu.
- *Testeři* konzultují funkčnost softwaru s vývojáři, testují implementovaný software, mají na starosti testovací scénáře a vývoj automatických testů.

4.1.3 Aktuální situace

Tým funguje v rámci agilních metodik, kdy si selektivně vybral některé prvky Scrumu, které byly upraveny prvotním potřebám týmu, proto bude v následujících kapitolách použitý slovník Scrumu. Tým pracuje v třítýdenních sprintech.

Tým rozvíjí a vlastní softwarový produkt, který je používán jako součást dodávky pro zákazníka. Nabízený software slouží jako základ a na něm interní adoptéři (používají dodávaný software k vytvoření řešení pro koncového zákazníka) staví řešení pro zákazníka na míru jeho potřebám. Většina požadavků na funkcionalitu pochází od interních stakeholderů v rámci společnosti. Jednotlivé požadavky jsou uspořádány podle priorit v nástroji pro řízení úkolů (ve firmě se používá nástroj Jira) a následně rozděleny na menší

úkoly, které jsou během pravidelné plánovací porady přidělené jednotlivým vývojářům a případně i testerům.

Samotná implementace probíhá většinou tak, že vývojář napíše nebo upraví design dokument dle definice zadaného úkolu a začne s implementací. Po dokončení implementační části předá kód na revizi jinému vývojáři. Po dokončení revize a následné případné úpravy kódu předá vývojář svou práci k testování.

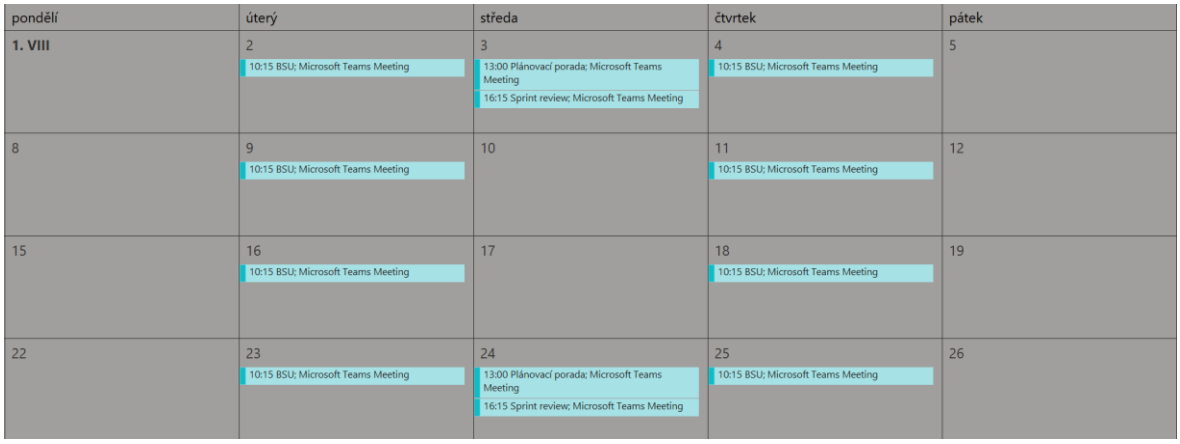
Testování jednotlivých úkolů probíhá způsobem, kdy tester si sedne s vývojářem a probírají všechny možné validní i nevalidní scénáře, které během používání nové funkcionality mohou nastat. K ruce mají upravený design dokument, podle kterého kontrolují, zda design dokument souhlasí s danou implementací. Tester pak samostatně sepíše nové a upraví stávající testovací scénáře, případně jednotlivé testy a podle toho funkčnost implementace verifikuje. Samotné testování kompletní funkčnosti probíhá v interním prostředí, které věrně simuluje produkční prostředí. Hotové a validní testovací scénáře slouží jako podklad pro testování adoptéry v produkci.

Na základě toho, že tým spolupracuje denně s adoptéry, kteří dostávají novou funkcionalitu i v průběhu sprintu, stává se pravidlem, že co je ve sprintu naplánováno ztrácí v danou chvíli prioritu a mění se obsah práce ve sprintu ad hoc. Jednotlivé úkoly na sebe často hodně navazují, prolínají se, tým není často schopen doručit funkční celek na konci sprintu. Tým pracuje často pod tlakem, protože adoptéři chtějí, aby požadovaná funkcionalita byla dodána co nejdříve, což ve výsledku občas způsobí, že kvalita implementace a otestování není na takové úrovni, jaké by se očekávalo.

V týmu vede přímá komunikace, často se důležité věci nezaznamenávají. Pravděpodobně proto, že většina dokumentů na nachází ve sdíleném úložišti Microsoft SharePoint, ale občas se stane, že po úpravě nějakého dokumentu, ať už je to design dokument nebo testovací scénář, ho zapomene dotyčný nahrát na úložiště. SharePoint sice nabízí možnost editace dokumentu přímo na sdíleném úložišti, ale to nefunguje úplně spolehlivě. Proto je lepší stáhnout si soubor a udělat úpravy lokálně, avšak pracují-li na daném dokumentu dva, můžou si vzájemně přepsat své změny. Dále tým nemá na SharePointu danou jasnou strukturu, kde přesně mají být jaké dokumenty, ty jsou pak těžce dohledatelné a vznikají duplicity. V týmu chybí formální revize designu, a tudíž designová dokumentace má rozdílnou úroveň kvality dle osoby, která ji zrovna zpracovávala.

4.1.4 Procesy v týmu

Na následujícím obrázku lze vidět, jak má tým naplánovaný běžný sprint.



Obr. 12 Standardně naplánovaný sprint (vlastní zpracování)

Plánovací porada

Tým pracuje v třítýdenních sprintech, nový sprint vždy začíná ve středu. Start sprintu začíná plánovací poradou, která trvá přibližně 2 hodiny.

Vedoucí týmu společně s vedoucím vývojářů postupně představují jednotlivé nové úkoly, popřípadě reportované chyby, nejčastěji od adoptérů, zapsané v Jiře, které jsou třeba udělat nebo opravit v následujícím sprintu. Poté celý tým včetně testerů ohodnotí pracnost daných úkolu jako počet dní, kolik si myslí, že na tom celkově stráví. Jednotlivé úkoly se následně přiřadí konkrétnímu vývojáři a testerovi, kteří buď o danou práci mají zájem, anebo jim ještě stále zbývá na další sprint volná kapacita. Plánovací porada končí, když mají všichni členové týmu naplánovanou práci na celý nadcházející sprint.

Plánovací porady se účastní povinně všichni členové týmu (s výjimkou dovolených). Protože část týmu pracuje z domova a část týmu pracuje v kanceláři, koná se porada v zasedací místnosti s připojením ostatních členů pracujících z domova prostřednictvím aplikace Teams (komunikační nástroj pro firmy).

Standup

Standup se koná dvakrát týdně, pravidelně vždy v úterý a čtvrtek od 10:15. Desátá hodina je čas, kdy už musí být všichni buď přítomni v kanceláři nebo u svého počítače doma.

Standup je v kalendáři naplánován na 15 minut. Účastní se ho celý tým včetně vedoucího týmu. Každý po jednom mluví o tom, na čem dělal během předchozích dní a na čem plánuje pracovat do dalšího standupu, zda ho něco nebrzdí v jeho práci, nebo si s něčím

neví rady, popřípadě zda je jeho práce již připravena na revizi kódu či testování. Vedoucí týmu či pod týmů (vývojářů, testerů) mají možnost rozdělit případné nově zaevidované RC anomálie. Celý standup se dokumentuje, vždy jeden vybraný člen týmu zapisuje do textového editoru (v rámci týmu se využívá Microsoft OneNote), kdo co dělal nebo na čem aktuálně pracuje. Dokument se poté sdílí na SharePoint k ostatním zápisům.

Standup se koná pravidelně prostřednictvím aplikace Teams, takže tyto porady nejsou ve stoje dle definice Scrumu, ale tým sedí každý u svého počítače.

Pravidelně není tým schopen 15 minut dodržet a schůze se prodlužuje. Často se rozvíjí diskuse nad aktuálním stavem prací, či jiným příbuzným tématem. Vedoucím týmu často vstupuje do diskusí, také využívá standup pro řešení případných budoucích plánů nebo ke sdělení současných problémů, se kterými se tým musí nebo bude muset vypořádat.

Sprint review

Po konci každého sprintu se pořádá velké sprint review všech spolupracujících týmů s adoptéry, které je sice povinné pro všechny členy, ale hovoří tam vždy pouze jeden zástupce týmu. V případě daného týmu je to vedoucí vývojářů, který reportuje všechny úkoly ze sprintu. Říká, co tým během posledního sprintu zvládl dokončit, na čem nadále pracuje a zda se očekává, že daný úkol bude doručen v dalším sprintu. Protože je týmů hodně a sprint review by trvalo dlouho, kdyby každý prezentoval demo své práce, prezentuje se pouze vybraná dokončená funkcionalita, většinou na základě dobrovolnosti nebo dokončí-li se větší část funkcionality.

Retrospektiva

Retrospektivy se opět účastní celý tým. Tato schůze probíhá spíše výjimečně, zhruba jednou za 2–3 měsíce, naplánovaná je na 2 hodiny, ale ve většině případů není celý čas využit. Schůzi moderuje pokaždé jiný člen týmu. Retrospektiva probíhá prostřednictvím aplikace Teams. Každý má příležitost říct, co se mu od poslední retrospektivy líbilo, co se mu povedlo, co se mu osvědčilo, popřípadě co mu vadilo nebo se nepovedlo. Pokud se jedná o negativní zkušenosti nebo věci, které je potřeba vylepšit, sepíše se plán úkolů, které by měly být do další retrospektivy splněny, zbude-li čas.

Příští retrospektiva tedy začíná rozborem plánu s úkoly z poslední retrospektivy, ale protože za něj nemá nikdo přímou zodpovědnost a retrospektiva není v týmu pravidelná

schůze, většinou dojde k tomu, že úkoly z plánu nebyly splněny nebo se na ně zapomnělo, takže není, jak hodnotit posun týmu, roste nedůvěra k této schůzi a ztrácí tak svůj smysl. Ne každý člen má motivaci do diskuse přispět svými starostmi nebo nápady, protože již dopředu počítá s tím, že je to zbytečné, daný problém se řešit nebude a na nápady není čas.

4.1.5 Používané nástroje

Administrativní oblast

- **Microsoft**
  - OneNote – zápis ze standup schůze
  - Outlook – emailová komunikace, kalendář pro organizaci schůzek, simulované phishing kampaně k tréninku zaměstnanců
  - SharePoint – uchování definice procesů, dokumentace vydaných verzí, úložiště pro design dokumenty a testovací scénáře
  - Teams – celofiremní komunikační nástroj (chaty, hovory, live eventy), zpřístupnění některých dokumentů ze SharePointu
- **JIRA**
  - zachycení jednotek práce
  - plánování práce v jednotlivých sprintech
  - reporty (sprint report)

Technická oblast

- **Git, GitLab**
  - zdrojový kód, design dokumenty, automatické testy, kontrola kódu, binární soubory na podporu testovacích aktivit
- **BlackDuck**
  - skenování zranitelností,
  - materiály pro firemní proces zvaný *Design for security* potřebný pro vydání nových verzí
- **Visual Studio**
  - pro vývoj samotného softwaru
- **Microsoft SQL Server Management Studio, Azure Storage Explorer**
  - testovací nástroje

- **SoapUI**
  - pro vývoj automatizovaných testů
- **ApTest**
  - interní nástroj, který slouží jako databáze testovacích případů

4.1.6 Shrnutí analýzy

Tým vykazuje snahu ubírat se cestou agilního vývoje, pracuje v třítýdenních sprintech, využívá selektivně některé prvky Scrumu, které upravil svým aktuálním potřebám. Jsou zde zavedeny pravidelné plánovací porady na začátku každého sprintu, standup dvakrát týdně, na konci každého sprintu se koná celofiremní sprint review a jednou za pár sprintů tým naplánuje retrospektivu. V týmu vyvstává problém s uchováváním a dohledatelností potřebných dokumentů.

Plánovací porady probíhají na začátku každého sprintu. Během této schůze se rozdělí práce mezi jednotlivé členy týmu. Tým komunikuje se zákazníkem na denní bázi, který paralelně testuje dodanou funkcionalitu na produkčním prostředí. Nové funkce mu ale nejsou dodávány na konci samotného sprintu jako funkční celky, ale dodávají se s každou novou funkcionalitou po dotestování i během sprintu. Často se tak stává, že většina původních naplánovaných úkolů ve sprintu nejsou splněny, protože se ve sprintu prioritně řeší nově zaevidované chyby. Tým tak není schopen správně odhadnout množství práce, kterou je schopen za daný sprint doručit.

Standup se koná dvakrát týdně. Velkým problémem této schůzky je, že se často rozvíjí delší diskuse nad aktuálními nebo jinými tématy, celý standup se zbytečně protahuje a zdržuje ty, kteří případně k danému tématu nemají co říct. Výstup ze standupu se zapisuje do OneNote a dále se s ním nepracuje.

Sprint review se koná na konci každého sprintu. Jedná se o celofiremní schůzi, na které se reportuje odvedená práce za poslední sprint a prezentuje vybraná nová funkcionalita.

Retrospektiva probíhá v týmu velmi málo, a to jednou za pár sprintů. Průběh retrospektivy se nezapisuje, vytváří se pouze plán úkolů pro vyřešení zmiňovaných problémů a na začátku další retrospektivy se zjišťuje, kam tým pokročil a jestli došlo ke zmírnění nebo vyřešení stávajících problémů. Většinou to ale dopadá tak, že se na plán s úkoly tak trochu pozapomnělo, tým se dopředu neposouvá a retrospektiva ztrácí na významu.

Dalším zajímavým výstupem z analýzy je, že tým používá mnoho nástrojů na stejné či podobné věci. Dochází pak k tomu, že jsou dokumenty (design, dokumenty, tutoriály, návody) občas těžce dohledatelné, tvoří se duplicity anebo se nic nezapíše, informace si předávají v týmu mezi sebou osobně, každý má ve svém PC své vlastní poznámky nebo také dokumenty, které zapomněl nahrát na SharePoint.

4.2 Výběr vhodné metodiky pro daný tým

Agilní metodiky popsané v teoretické části práce obsahují velmi podobné principy a postupy, sdílejí podobné hodnoty, přesto je každá metodika jiná. Každá přináší trochu odlišný pohled na vývoj softwaru a odlišně posuzuje konkrétní případy.

V rámci této práce bude pro výběr vhodné metodiky použita optimalizační metoda minimalizace vzdálenosti od ideální varianty, metoda TOPSIS. Vhodná metodika bude následně vybrána ze tří agilních metodik na základě šesti kritérií, které jsou pro daný tým klíčové. Kritéria, kromě Složitost metodiky, kterou tým požaduje co nejjednodušší, jsou vymezena jako maximalizační.

4.2.1 Stanovení hodnot kritérií

Celkem podstatný problém při výběru vhodné metodiky pro konkrétní projekt je neexistence jednotné klasifikace metodik, která by usnadnila způsob porovnání a následný výběr. Protože každý projekt je jedinečný, jsou vyžadovány různé metodiky.

Pro porovnání metodik pro vývoj softwaru budou využita některá kritéria, která nastavila Buchalcevová (2005b), těmi kritérii jsou: *dostupnost zákazníka, řízení změn, velikost týmu, podpora životního cyklu a složitost metodiky*. Po konzultaci s celým týmem bylo stanoveno další podstatné kritérium a tím je *komunikace.*

Pro každé kritérium následuje krátký popis, který charakterizuje význam daného kritéria. Seznam kritérií a jejich hodnoty jsou uvedeny v tabulce dále (Tabulka 1).

Kritérium č. 1: Dostupnost zákazníka

Tým je v každodenní spolupráci s adoptéry, kteří často vstupují do procesu při vývoji. Buď mají nové nebo pozměněné požadavky nebo reportují chyby z poslední nasazené verze. Z pohledu týmu je třeba taková metodika, která zákazníka pravidelně zapojuje do procesu vývoje.

Kritérium č. 2: Řízení změn

Kritérium řízení změn úzce souvisí s předchozím kritériem. Jak již bylo řečeno, adoptéři často vstupují do procesu při vývoji, zasahují do naplánovaného sprintu novými, pozměněnými požadavky nebo evidují chyby, které chtějí co nejdříve opravit, tudíž se tým musí být schopen velmi dobře těmto změnám přizpůsobovat. Účelem tohoto kritéria je zhodnotit jednotlivé metodiky, jak se vypořádávají s případnými změnami požadavků v průběhu vývoje.

Kritérium č. 3: Velikost týmu

Toto kritérium hodnotí velikost týmu na základě počtu jeho členů. Agilní metodiky pracují s menším počtem členů v týmu, ale jsou i takové, které se dokáží přizpůsobit projektům řešené většími týmy. Tým, pro který bude metodika zvolena čítá do 10 členů.

Kritérium č. 4: Podpora životního stylu

Agilní metodiky používají principy definované Agilním manifestem. Nicméně každá metodika má něco, čím se odlišuje od ostatních. Příkladem může být přímé zaměření na konkrétní vývojovou fázi. Naopak jiné vývojové fáze nemusí být vůbec zahrnuty v procesech nebo praktikách dané metodiky. Toto kritérium hodnotí, zda daná metodika pokrývá celý životní cyklus nebo jen některé oblasti. Tým se zabývá procesem vývoje od stanovování požadavků až po následnou údržbu softwaru.

Kritérium č. 5: Složitost metodiky

Každá metodika je jinak podrobná, propracována, obsáhlá. Některé definují konkrétní procesy a praktiky, některé definují pouze doporučené postupy. Tým cílí na metodiku, která je nejméně náročná na osvojení, aby svou obsáhlostí, principy a praktikami nesnižovala produktivitu a efektivitu týmu.

Kritérium č. 6: Komunikace

Komunikace je základ. Je nezbytnou součástí efektivity týmu. Pokud si členové týmu vzájemně nesdílejí potřebné informace, vede to k prodlužování doručování jednotlivé práce, k neshodám v týmu a nedorozumění. Tým potřebuje metodiku, která podporuje komunikaci ve větším měřítku.

Kritérium	Hodnota kritéria
1. Dostupnost zákazníka	a) zákazník je součástí týmu b) zákazník je k dispozici v průběhu vývoje c) zákazník je přítomen jen na začátku a na konci vývoje
2. Řízení změn	a) nereaguje na změny b) reaguje na změny c) metodika klade velký důraz na změny
3. Velikost týmu	a) 1–10 c) 11–50 d) 50 a více
4. Podpora životního cyklu	a) pokrývá jen některé oblasti životního cyklu b) pokrývá celý životní cyklus
5. Složitost metodiky	a) velmi náročná b) náročná c) mírně náročná d) jednoduchá metodika
6. Komunikace	a) pravidelná b) dostačující c) jen v nutnosti

Tabulka 1: Zvolená kritéria a jejich hodnoty

4.2.2 Stanovení vah pro kritéria

Vývojový tým stanovil váhy pro jednotlivá kritéria uvedené v následující tabulce (Tabulka 2) za použití Saatyho metody kvantitativního párového srovnání. Stanovení vah jednotlivých kritérií bude použito dále při výběru optimální metodiky pomocí metody TOPSIS.

	Dostupnost zákazníka	Řízení změn	Velikost týmu	Podpora živ. cyklu	Složitost metodiky	Komunikace	Váhy
Dostupnost zákazníka	1	1/3	5	1/3	3	1/5	0,1
Řízení změn	3	1	7	3	5	1/3	0,25
Velikost týmu	1/5	1/7	1	1/5	1/3	1/7	0,03
Podpora živ. cyklu	3	1/3	5	1	3	1/3	0,16
Složitost metodiky	1/3	1/5	3	1/3	1	1/5	0,06
Komunikace	5	3	7	3	5	1	0,36
						∑	1

Tabulka 2 Stanovení vah Saatyho metodou

4.2.3 Ohodnocení metodik podle hodnot kritérií

Metodiky byly vzájemně porovnány na základě kritérií uvedených v Tabulka 1. Dále je zapotřebí ohodnotit jednotlivé metodiky podle stanovených kritérií. Každé metodice je přiřazena hodnota kritéria a jeho číselné ohodnocení. Jednotlivá kritéria jsou hodnocena na základě stupnice od 1 do 10. Maximální hodnota 10 znamená, že metodika nejvíce odpovídá požadavkům daného týmu a minimální hodnota 1 znamená, že metodika nejméně odpovídá požadavkům daného týmu. Přiřazení hodnot jednotlivých kritérií a jejich číselné ohodnocení je zobrazeno v následujících tabulkách pro každou metodiku zvlášť (Tabulka 3–5).

Scrum		
Kritérium	**Hodnota kritéria**	**Hodnocení**
Dostupnost zákazníka	uživatel je součástí týmu	10
Řízení změn	metodika klade velký důraz na změny	8
Velikost týmu	1–10 členů	8
Podpora životního cyklu	pokrývá celý životní cyklus	10
Složitost metodiky	jednoduchá metodika	2
Komunikace	pravidelná	10

Tabulka 3 Hodnocení Scrum

Extrémní programování		
Kritérium	**Hodnota kritéria**	**Hodnocení**
Dostupnost zákazníka	uživatel je součástí týmu	7
Řízení změn	reaguje na změny	9
Velikost týmu	1–10 členů	8
Podpora životního cyklu	pokrývá jen některé oblasti životního cyklu	8
Složitost metodiky	jednoduchá metodika	3
Komunikace	dostačující	8

Tabulka 4 Hodnocení XP

Feature Driven Development		
Kritérium	**Hodnota kritéria**	**Hodnocení**
Dostupnost zákazníka	uživatel je součástí týmu	8
Řízení změn	metodika klade velký důraz na změny	6
Velikost týmu	1–10 členů	10
Podpora životního cyklu	pokrývá jen některé oblasti životního cyklu	7
Složitost metodiky	mírně náročná	6
Komunikace	dostačující	6

Tabulka 5 Hodnocení FDD

4.2.4 Odůvodnění číselného hodnocení

Tato kapitola je interpretací předchozí kapitoly Ohodnocení metodik podle hodnot kritéria. Hlavním cílem této kapitoly je odůvodnění číselného vyjádření hodnot kritérií pro každou metodiku.

Dostupnost zákazníka

Na základě tohoto kritéria se jeví jako nejlepší metodika taková, kdy je zákazník součástí týmu nebo maximálně k dispozici. Jedině vývoj takového softwaru, kde zákazník upřesňuje své požadavky, poskytuje pozitivní či konstruktivní zpětnou vazbu, může mít šanci úspěchu a splnit tak očekávání zákazníka. Všechny porovnávané metodiky toto kritérium splňují, avšak každá trochu jinak.

XP svými praktikami podporuje přítomnost zákazníka na pracovišti, ale velmi záleží na ochotě objednavatele softwaru čas svého zaměstnance poskytnout. Kdežto metodika Scrum má zákazníka na pracovišti k dispozici vždy, zprostředkovaně, a to v roli vlastníka produktu. Vlastník produktu je hlasem zákazníka, úzce s ním spolupracuje a společně vytváří seznam požadavků. Metodika FDD definuje roli doménového experta, kterým může být samotný zákazník nebo kdokoliv, kdo dané oblasti, pro kterou je software vyvíjen, perfektně rozumí a který také dohlíží na to, že vyvíjený produkt odpovídá požadavkům zákazníka.

Řízení změn

Daný vývojový tým denně napřímo spolupracuje s adoptéry, musí být schopen velmi pružně reagovat na změnu požadavků, potřebuje metodiku, která mu to svými postupy a praktikami umožní co nejlépe a nejjednodušeji.

Všechny porovnávané metodiky jsou iterativní. Nejlépe v hodnocení vyšla metodika XP, je to metodika, která pracuje v extrémech, čímž může být i velmi krátká iterace v řádu pár dní, záleží však na konkrétních týmech, jak dlouhou iteraci si stanoví. Scrum a FDD doporučuje většinou dvoutýdenní iterace. U FDD však může být nový požadavek větším problémem na základě toho, že pro každou vlastnost sestavuje určitý vývojový tým, což může být nepatrně časově náročnější z hlediska dostupnosti a vytíženosti jednotlivých vlastníků tříd.

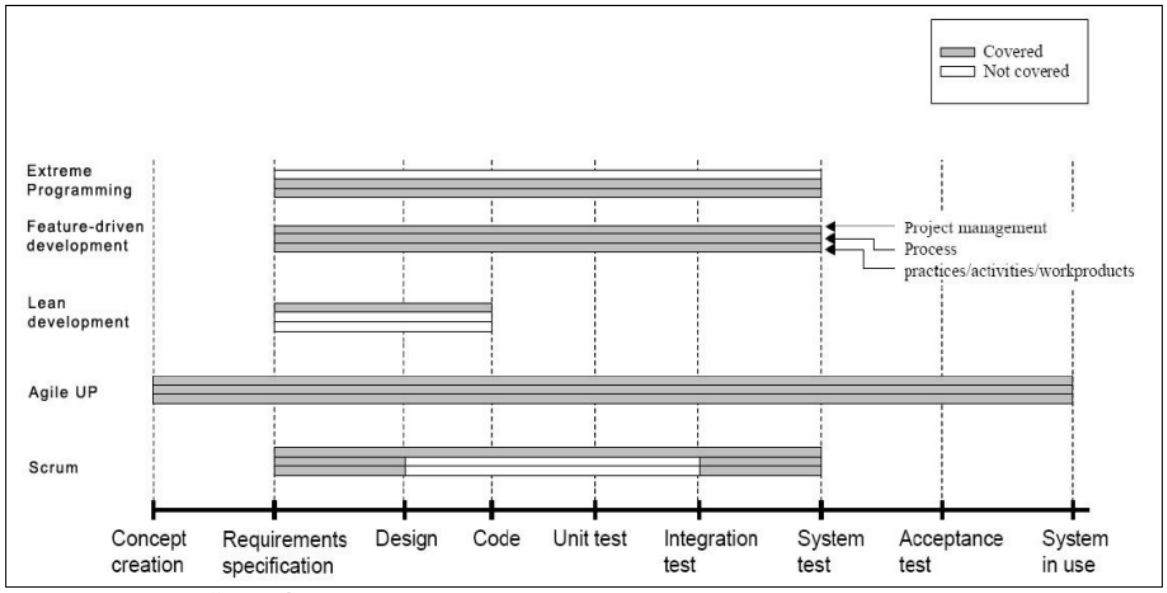
Velikost týmu

Podle toho kritéria jsou jednotlivé metodiky posuzovány na základě toho, jak jsou schopné přizpůsobit se velikosti vývojového týmu. Daný tým čítá pouze 10 členů, včetně vedoucího týmu, tudíž nejvhodnější metodikou bude taková, která se přizpůsobuje především menšímu počtu členů v týmu.

XP je vhodná pro 2-10 vývojářů a Scrum pracuje s týmy o velikosti zhruba do 10 členů. Obě metodiky proto splňují toto kritérium. FDD k velikosti týmu přistupuje dynamicky. Je vhodná i pro větší týmy, takže je flexibilní v případě rozšíření velikosti týmu, a tvoří jednotlivé menší vývojové týmy vždy podle potřeby implementace konkrétní užitné vlastnosti. Z důvodu většího rozsahu má FDD nejlepší hodnocení.

Podpora životního cyklu

Životní cyklus začíná návrhem softwaru a končí provozem a případnou údržbou softwaru. Daný tým pracuje na již zavedeném a rozběhlém softwaru, který testují adoptéři v produkčním prostředí, takže se dá říci, že vyhovující metodika je taková, která podporuje životní cyklus od stanovování požadavků na systém až po údržbu systému v provozu.



Obr. 13: Podpora životního cyklu (Abrahamsson el al., 2011)

Na základě obrázku výše lze říci, že všechny tři porovnávané metodiky splňují požadavek týmu na podporu celého životní cyklu. Fáze akceptačního testování a používání softwaru je pokryta zprostředkovaně adoptéry. Pro XP je však dominantní částí testování, kterému se vývojáři věnují přednostně (psaní jednotkových testů), FDD se zaměřuje

především na návrh a implementaci. Nejlépe vychází metodika Scrum, která neupřednostňuje konkrétního fáze životního cyklu software a všem fázím se věnuje rovnoměrně.

Složitost metodiky

Toto kritérium hodnotí náročnost vybraných metodik. Čím jednodušší metodika, tím snazší je ji v týmu zavést. Protože měnit návyky zaběhnutým vývojářům není úplně jednoduché, žádoucí je tudíž metodika, která není příliš náročná na její používání.

Nejlépe hodnocena vyšla metodika Scrum. Jednak proto, že nedefinuje příliš mnoho procesů a postupů, předpisuje pár týmových pravidelných schůzek a jednak proto, že tým selektivně pracuje s některými praktikami této metodiky, byť upravenými. XP patří také mezi jednodušší metodiky. Nepopisuje žádné složité procesy, poskytuje 12 praktik, při jejichž zavedení lze s metodikou pracovat nejefektivněji. Metodika FDD byla ohodnocena jako mírně náročná. To z toho důvodu, že metodika předepisuje v každé iteraci vypracovat návrh, naimplementovat užitnou vlastnost a za předpokladu, že daná vlastnosti nesplňuje požadavky, je třeba se vrátit na začátek a přepracovat návrh. To zvyšuje časovou náročnost vývoje.

Komunikace

Toto kritérium hodnotí, jakým způsobem daná metodika prosazuje komunikaci v týmu a zda vůbec. Bez efektivního způsobu komunikace se mohou prodlužovat lhůty, a také může docházet k dezinformacím nebo se správné informace nedostanou ke správným lidem.

Na základě tohoto kritéria nejlépe vychází metodika Scrum, a to především proto, že předepisuje schůzi na denní bázi, která je přímo určená pro sdílení informací, aktuálního stavu či vyskytnutých problémů jednotlivých členů týmů. Pro XP patří komunikace patří mezi základní hodnoty, využívá různé postupy, jak udržet komunikaci v týmu, např. párové programování. Pokud komunikace vázne, má XP svého kouče, který se snaží komunikaci dostat zpět do fungujícího stavu. U metodiky FDD je udržení komunikace podporováno dynamickým vytváření týmu složených z vlastníků tříd.

4.2.5 Výběr vhodné metodiky

Na základě stanovených vah kritérií a následného číselného ohodnocení jednotlivých metodik podle daných kritérií, bude vybrána optimální metodika pomocí metody stanovení pořadí variant, metoda TOPSIS. Veškeré výpočty jsou uvedeny v Příloha 2 – Metoda TOPSIS.

	Ci	Pořadí
Scrum	**0,872**	**1**
XP	0,579	2
FDD	0,054	3

Tabulka 6: Pořadí variant

Podle výsledku metody TOPSIS (Tabulka 6) se jeví jako optimální varianta metodika Scrum, protože relativní ukazatel vzdálenosti od bazální hodnoty má nejvyšší hodnotu. Pro tuto metodiku bude v následující kapitole sepsán návrh její implementace pro daný tým.

4.3 Návrh na implementaci vybrané metodiky

Jelikož provádět velké změny v zaběhnutém týmu může způsobit spíše chaos než přinést užitek, a především vývojáři bývají většinou k velkým změnám skeptičtí a těžce spolupracují, bude návrh implementace rozdělen do tří fází. Z jedné fáze do druhé se bude přecházet až ve chvíli, kdy bude zavedeno vše z předchozí fáze a všichni budou se změnou sladěni. Hrubý odhad implementace celého návrhu je 6–7 měsíců.

V týmu chybí dvě podstatné role, a to scrum master a vlastník produktu. V roli scrum mastera budu po celou dobu vystupovat já, jakožto člen daného týmu. Náplň práce scrum mastera mimo jiné spočívá v dohlížení a vedení schůzek, mezi které patří plánovací porada, standup a retrospektiva. Dohlíží na to, aby produktivita týmu nebyla ovlivňována externími a negativní vlivy a vytrvale vysvětluje principy Scrumu a jeho procesy.

Doposud roli vlastníka produktu neoficiálně vykonávali společně vedoucí týmu a vedoucí vývojářů. Náplň práce vlastníka produktu je především správa product backlogu. Stanovuje priority jednotlivým úkolům v backlogu, vysvětluje jejich náplň týmu, proč jsou třeba a proč je zákazník chce. Pak podle priorit vybírá úkoly do následujících sprintu. Je také zodpovědný za kvalitu doručovaného produktu.

4.3.1 První fáze

První fáze zahrnuje seznámení týmu s metodikou Scrum jako celek. Jelikož daný tým pracuje s některými částmi této metodiky, které přizpůsobil svým potřebám, začneme se stávajícími, již existujícími procesy, které upravíme tak, aby vyhovovaly definici Scrumu. Konkrétně se jedná o důležité schůze standup, který je teď využíván víceméně jako komunikační kroužek, a retrospektivu, ke které je přistupováno jako ke zbytečné schůzi bez jakéhokoliv efektu. Součástí první fáze je také návrh lepšího seskupení dokumentace na jedno místo za účelem lepší dohledatelnosti, udržitelnosti a ke snížení duplicit dokumentů.

Kvůli zavádění pravidelné retrospektivy, která se koná na konci každého sprintu, je časový odhad pro zavedení této fáze na 3 sprinty (9 týdnů).

4.3.1.1 Seznámit tým s metodikou Scrum jako celek

Na schůzi pro celý tým scrum master představí Scrum, jak je definován, jaké role jsou ve Scrumu definovány, jaké jsou jejich povinnosti a náplň práce. Scrum master seznámí všechny členy týmu s konkrétními procesy a postupy Scrumu, jakým způsobem s nimi pracovat, aby dávaly smysl a jejich využívání vedlo k efektivitě. Na základě množství dotazů jednotlivých členů týmu bude stačit jedna schůze, nebo se rozdělí na více schůzek, aby tým byl schopen z většího množství informací pochytit a zpracovat co nejvíce.

4.3.1.2 Standup každý den

Na základě provedené analýzy víme, že standup často přesahuje plánovaný časový rozsah, kvůli častému řešení věcí nad rámec a tím i postrádá svůj hlavní význam, protože Scrum popisuje standup jako krátkou synchronizační schůzi členů vývojového týmu.

Doteď standup probíhal každé úterý a čtvrtek v 10:15. V první řadě tým zavede tuto schůzku každý den. Čas zůstane stejný, protože to je doba, kdy musí být všichni členové týmu v kanceláři nebo doma u počítače. Vynechá se den, kdy je plánovací porada a sprint review. Jelikož členové týmu pravidelně reportovali svou za práci za poslední 2–3 dny, mít Standup každý den vyřeší problém přetahování určeného času pro tuto schůzku. Také se zkrátí prodleva při vykonávání jednotlivých úkolů, protože pokud si někdo nebude vědět rady, jak při plnění úkolu dále postupovat, na standupu se s někým, kdo mu pomůže, domluví.

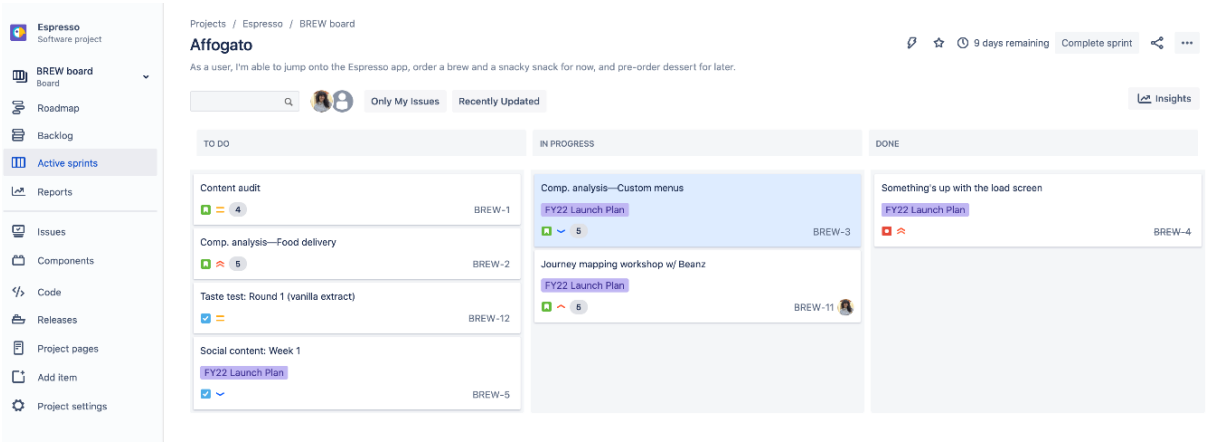
Dále se pro každého člena týmu omezí standup pouze na tři podstatné otázky, které musí každý zodpovědět:

- Co dělal včera? – podstatné je, co každý opravdu dokončil, vyprávění o cestě k cíli zbytečně schůzku prodlužuje.
- Co bude dělat dnes? – zda má vše potřebné k pokračování na svém úkolu.
- Má nebo nemá s něčím problém. – tato otázka je klíčová, případné problémy s pokračováním či dokončením úkolu se začnou řešit okamžitě.

Standup povede scrum master, který má za úkol ohlídat, že každý zodpověděl výše uvedené otázky a v případě, že se rozvine diskuse nad nějakým tématem, navrhne, aby členové, jichž se to týká, zorganizovali pro vzniklý problém samostatnou schůzku, kde to společně vyřeší. Jelikož se standup koná prostřednictvím aplikace Teams, ušetří to čas i všem ostatním, kterých se dané téma aktuálně netýká.

Kromě vývojářů a testerů, kteří sdílejí informace s ostatními, jakým způsobem pokračují ve své práci, a pro které je tato schůze především určená a zároveň povinná, může se ke standupu připojit kdokoliv. Ovšem pouze jako pozorovatel, který do schůze nijak nezasahuje a neovlivňuje její průběh. Tudíž do standupu nesmí zasahovat ani diskuse o budoucích plánech či problémech, které se nějakým způsobem týmu dotýkají. Pro tato sdělení bude naplánována samostatná schůze.

Další vylepšení a urychlení pro standup je vynechání zápisu z této schůzky, s kterým se dále nijak nepracuje. Tým využije online tabuli Active Sprints board. Jedná se o nástroj, který poskytuje Jira. Active Sprints board je zobrazen na následujícím obrázku.



Obr. 14 Active Sprints board (support.atlassian.com)

Tým postupně projde jednotlivé úkoly ve sprintu a ujistí se, v jaké fázi se daný úkol nachází, případně aktualizuje jeho stav. Úpravu online tabule bude mít nadále každý standup na starosti někdo jiný z týmu.

4.3.1.3 Retrospektiva po každém sprintu

Retrospektiva je velmi cenným zdrojem zpětné vazby uvnitř týmu. Bude pravidelně organizována na konci každého sprintu. Tým má živě v paměti, co se za poslední sprint odehrálo, co se jim vydařilo, nebo naopak nepodařilo, s čím byli spokojeni a s čím ne anebo co by na základě poslední sprintu chtěli vylepšit.

Než se tato schůze ustálí, povede ji scrum master, až scrum master uzná, že se tato schůze stala běžnou součástí pracovního sprintu a funguje, povede retrospektivu pokaždé někdo jiný z týmu, posílí se tím pocit zodpovědnosti každého člena. Protože ne každý má chuť a zájem do diskuse přispět něčím svým, začne se jednodušší formou schůze. Jako nejlepší způsob, jak opravdu získat alespoň nějaké informace od každého, je laicky řečeno takzvané kolečko. To znamená, že mluví jeden po druhém, a hlavně vždy mluví pouze jeden.

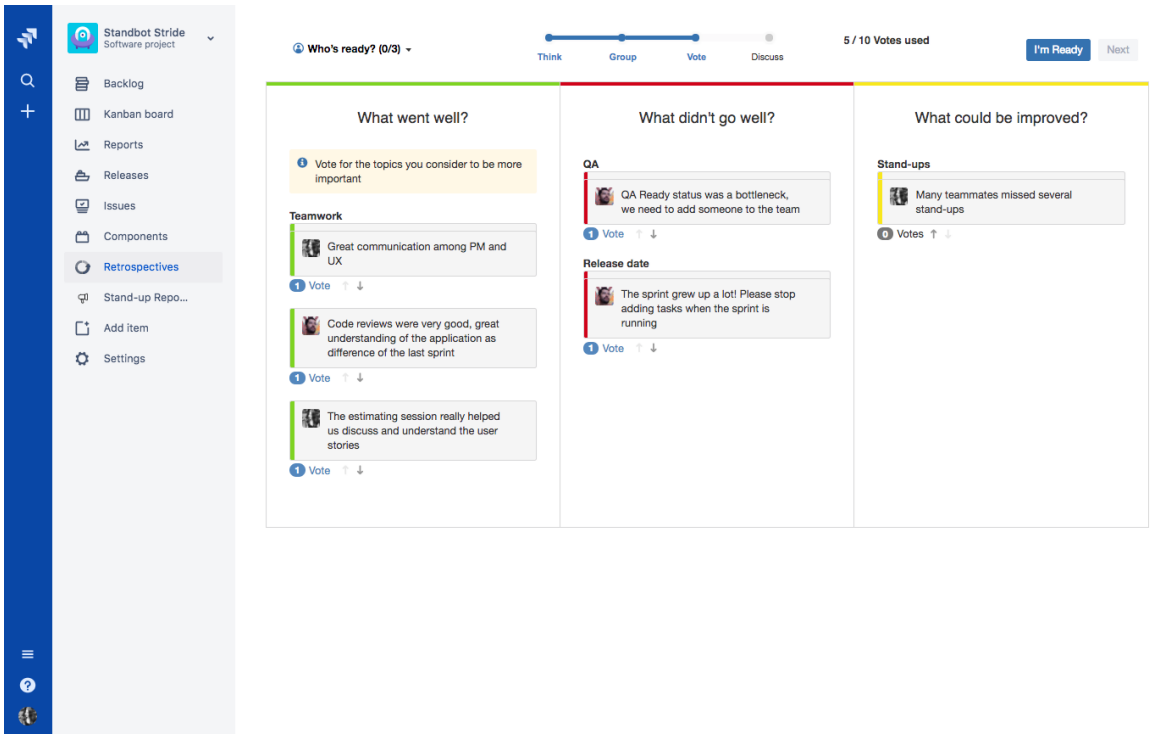
Každý zodpoví moderátorovi schůze podobně jako u standupu, tři otázky:

- Co se mu v poslední sprintu líbilo a chce v tom pokračovat?
- Co se mu nelíbilo a chtěl by to změnit?
- Co by zavedl nového?

Ve chvíli, kdy kolečkem projdou všichni, moderátor shrne, co vše zaznělo a následuje diskuse na jednotlivá témata. V případě negativních připomínek se tým pokusí najít možná řešení daných problémů. To znamená, jakým způsobem se pokusí odstranit nebo aspoň zmírnit zmíněné nedostatky v následujícím sprintu. Poté, co se proberou daná témata, sepíše tým plán úkolů, jak je zvyklý. Tentokrát s podstatnou změnou, a to takovou, že ke každému úkolu se rovnou přiřadí jeden konkrétní člověk, který bude mít za daný úkol zodpovědnost. Aby se dosáhlo požadovaného výsledku, bude to ten, který návrh na změnu pronesl, protože sám bude mít nejlépe pod kontrolou, zda změna jde správným a očekávaným směrem. Scrum master bude mít zodpovědnost za kontrolu plnění stanovených úkolů během sprintu.

Ve chvíli, kdy jsou v rámci schůze probrána potřebná témata, je sepsán plán úkolů se zodpovědnými osobami za jednotlivé úkoly, retrospektiva končí. Tato schůzka by neměla trvat déle jak 2 hodiny.

Tým bude používat pro vedení této schůze Jira Retrospectives. Jedná se o placenou aplikaci Jira software, který je velmi jednoduchý, účinný a efektivní. Obsahuje jednotlivé sloupce pro témata podle typu, co šlo dobře, co nešlo dobře a co by mělo být vylepšeno. Poté, co každý vepíše své názory či připomínky do příslušných sloupců, každý člen přidá k jednotlivým příspěvkům svůj hlas, čímž vyjádří buď svůj souhlas anebo nesouhlas s tím, co je napsáno. Jednotlivé příspěvky se následně seskupí do tematických celků. Pokud je témat k řešení mnoho, tak ta, která dostanou nejvíce hlasů se dále rozebírají v rámci aktuální schůzky, vedou se diskuse a následně se sepíše plán úkolů, ke kterým je přiřazen konkrétní člen týmu, který za splnění daného bodu ponese zodpovědnost. Témata, na která se během dané schůze nedostalo, se proberou na příští retrospektivě.



Obr. 15 Retrospektiva v Jira (marketplace.atlassian.com)

Jako záložní variantu, nelze-li používat placený doplněk Jiry, využije tým kombinaci Microsoft Powerpoint a Excel. Tabulky na stejné bázi jako Jira Retrospectives vytvoří ve vlastním souboru, který nahraje a uloží do jiné aplikace Jiry, a to Confluence, aby byl přístupný pro všechny členy týmu.

4.3.1.4 Dokumentace v JIRA

Z analýzy vyplynulo, že tým používá mnoho nástrojů pro uchovávání dokumentace. Ať už se jedná o návody pro nový nástup, design dokumenty, testovací scénáře nebo různé tutoriály. Některá dokumentace se uchovává na SharePointu, jiná na GitLabu, v horším případě některé potřebné informace se uchovávají v pracovních počítačích jednotlivých členů. Pro uchovávání veškeré dokumentace bude tým využívat Jira nástroj, dokumentační portál Confluence, kterým nahradí uchovávání dokumentace na SharePointu a na GitLabu a nastaví si jasnou strukturu dokumentace. Tím se nejen sníží počet používaných nástrojů, v tomto případě pro dokumentaci, ale zlepší se i dohledatelnost a udržitelnost jednotlivých dokumentů.

Bonusem je, že do Jiry mají přístup všichni, včetně adoptérů, což na SharePoint a GitLab nutně mít nemusí. Pokud není důležitá dokumentace na jednom místě, nebo není místo, které odkazuje na všechno potřebné, může být problém pro případný nový nástup, který nebude vědět na koho se obrátit, od koho získat správné informace, kde ty informace dohledat apod.

4.3.2 Druhá fáze

Druhá fáze zahrnuje oficiální zavedení nové role do týmu, a to konkrétně vlastníka produktu. Vytvoření nové role je v samostatné fázi, protože je potřebná pro zavádění nových procesů ve fázi následující. Tým momentálně není schopen určit, zda se role ujme někdo z řad již stávajících členů, nebo se bude jednat o nově příchozího člověka, který se bude muset nejdříve zaučit, proto není jednoduché určit časový odhad, ale nemělo by to trvat déle než 1–2 měsíce v závislosti na tom, kdo bude novou roli vykonávat.

4.3.2.1 Rozdělení rolí

Roli vlastníka produktu aktuálně neoficiálně vykonávají částečně vedoucí týmu, který převážně komunikuje s adoptéry a případně s potenciálními zákazníky a částečně vedoucí vývojářů, který plánuje práci pro jednotlivé sprinty a zprostředkovaně vytváří spojku mezi celým týmem a adoptéry. Jelikož ale s adoptéry komunikuje primárně vedoucí týmu a vedoucí vývojářů dostává většinu informací právě od vedoucího týmu, může dojít například ke špatnému pochopení zadání či různým nepřesnostem v zadání, ke špatné organizaci a naplánování sprintu.

Vlastník produktu je jeden člověk, který má jasně definované činnosti a cíle, které by měl plnit. Člověk, který osobně a přímo komunikuje s adoptéry, přebírá požadavky ale také zpětnou vazbu, a zároveň plánuje náplň následujících sprintů, tak aby byla včas a řádně doručena požadovaná funkcionalita. Vlastník týmu nezastává v týmu žádnou jinou roli, aby měl dostatek prostoru pro kvalitní výkon své funkce a neztrácel nadhled. Mohlo by se stát, že by jinak byla upřednostňována určitá část softwaru a ostatní, stejně či více důležité, by byly odsunuty na druhou kolej.

Nejjednodušší cestou, jak obsadit roli Vlastníka produktu, je vybrat si z řad vlastního týmu. Mělo by se jednat o zkušeného člena týmu, který danému softwaru dobře rozumí, je komunikativní a umí předávat informace. Pokud se nenajde v týmu vhodná osoba, a pokud to rozpočet dovolí, musí vedoucí týmu najmout nového člověka, který tým posílí. V takovém případě se ale musí počítat s tím, že doba na zaučení a porozumění softwaru jako celku může být výrazně delší, zároveň to zabere čas i tomu, kdo bude novou posilu zaučovat.

4.3.3 Třetí fáze

Třetí fáze je fází, kdy jsou do týmu již zavedené a obsazené podstatné role a jsou upraveny stávající procesy a postupy tak, aby odpovídaly tomu, jak jsou definovány ve Scrumu. Na řadě je nyní zavedení nových procesů do týmu. Tato fáze nemusí být nutně poslední, ale pro tento návrh ano. Prostor pro zlepšení je vždy a v první řadě záleží především na kvalitách a schopnostech scrum mastera, který správně vyhodnotí, které procesy jsou v týmu potřeba a které ne. V rámci třetí fáze bude zavedeno vlastní sprint review pro posílení zodpovědnosti jednotlivých členů v týmu, backlog grooming, ve kterém se tým předem seznámí s product backlogem a naučí se lépe plánovat budoucí sprinty. Díky backlog groomingu dojde k urychlení samotného plánovaní. Tato fáze zabere, opět vzhledem k aktivitám konajících se jednou za sprint, odhadem 4 sprinty (12 týdnů).

4.3.3.1 Vlastní sprint review

Vlastní sprint review se bude konat po konci každého sprintu před plánovací poradou. Cílem tohoto vlastního sprintu review je nejen zapojení všech členů týmů, aby se naučili být zodpovědnější za svou odvedenou nebo neodvedenou práci, ale také a především, je to cesta k získání přímé zpětné vazby, ať už pozitivní nebo konstruktivní, kterou každý jednotlivec ke své práci nutně potřebuje.

Tuto schůzi povede po celou dobu scrum master. Během vlastního sprint review se všechny úkoly, které byly na uplynulý sprint naplánovány, jeden po druhém projdou, zjistí se, co vše je hotové, a které úkoly naopak splněny nebyly. U každého dokončeného úkolu se zhodnotí jeho pracnost, jestli tým při odhadu nepodcenil nebo naopak nepřecenil některé aspekty daného úkolu, zda se nevyskytly neočekávané nebo opomenuté problémy, nebo zda byl úkol jednodušší, než tým původně předpokládal. U nedokončených úkolů se zjistí, co zabránilo tomu, aby byly včas doručeny a zda tomu šlo předejít či ne. Tato analýza jednotlivých úkolů přispěje k lepším a přesnějším odhadům pracnosti během plánování.

Součástí vlastního sprint review je také prezentace nebo demo veškeré odvedené práce za uplynulý sprint. Všichni členové poté mají přehled, kam se jejich software posouvá, protože ne všichni pracuje na všech úkolech, a prezentující dostane zpětnou vazbu nejen od vedoucího týmu ale i svých kolegů. Na projití jednotlivých úkolů a případné prezentace či dema dokončené práce by měla stačit jedna hodina.

4.3.3.2 Zavedení backlog groomingu

Pro tuto schůzi je klíčovou osobou vlastník produktu, který sbírá požadavky od adoptérů a případně interních stakeholderů. Backlog groomingu se tedy účastní vlastník produktu, vývojový tým a scrum master jako moderátor. Tato schůze bude organizována v polovině každého sprintu. Hlavním cílem backlog groomingu je příprava product backlogu pro plánování budoucích sprintů. Vlastník produktu na základě domluvy s adoptéry rozhodne, které úkoly mají největší prioritu a podle toho je v product backlogu seřadí. Na schůzi vlastník produktu seznámí všechny přítomné členy týmu s novými úkoly a poté tým společně jednotlivé úkoly ohodnotí. Pracnost úkolu ale nebude určovat počet dní, kolik pravděpodobně vývojář s testerem na daném úkolu stráví, jak je tým dosud zvyklý, ale bude dané úkoly ohodnocovat v relativních jednotkách.

Určení relativních jednotek:

- 1 – menší úkol, drobná úprava v kódu, testování jen verifikace bez nebo s minimální úpravou stávajících testů.
- 3 – standardní úkol, implementace menší části funkcionality, testování není obsáhlé, zahrnuje vytvoření 1-2 testů.

- 5 – větší úprava funkcionality, může být ovlivněna větší část softwaru, testování zahrnuje úpravu stávajících testů, popřípadě vytvoření nových.
- 8 – nová funkcionalita, očekává se delší čas strávený při implementaci, jakým způsobem a jak novou funkcionalitu zakomponovat, testování bude náročnější při seznámení s novou částí softwaru a vymyšlení nových testovacích případů.

Protože se většina schůzí odehrává prostřednictvím aplikace Teams, pro hodnocení využije tým chat této aplikace. Do chatu se do každé zprávy zvlášť napíše hodnocení (1, 3, 5, 8, víc). Poté dá každý člen týmu palec nahoru pro své hodnocení daného úkolu. Scrum master se zeptá člena týmu s nejnižším hodnocením, proč si myslí, že je úkol jednodušší, než si myslí ostatní a člena s nejvyšším hodnocením, proč si myslí, že je daný úkol tak náročný. Oba vysvětlí své názory a všichni mají možnost své hodnocení upravit, pokud si myslí, že něco při původním odhadu nedomysleli nebo přecenili. Takto se pokračuje s každým dalším úkolem, u kterého je třeba ohodnotit pracnost. Pokud nějaký úkol je ohodnocen jako *víc*, vlastník produktu tento úkol rozdělí na menší úkoly.

Začít myslet v relativních jednotkách není z minuty na minutu. Je to o zkušenostech týmu, každý další backlog grooming bude podstatně jednodušší, protože tým už bude vědět, jak ohodnotil úkoly v minulých sprintech, zda hodnocení odpovídalo realitě, a tudíž velikost nových úkolů budou ohodnocovat v porovnání s těmi dokončenými.

4.3.3.3 Zlepšení plánování sprintu

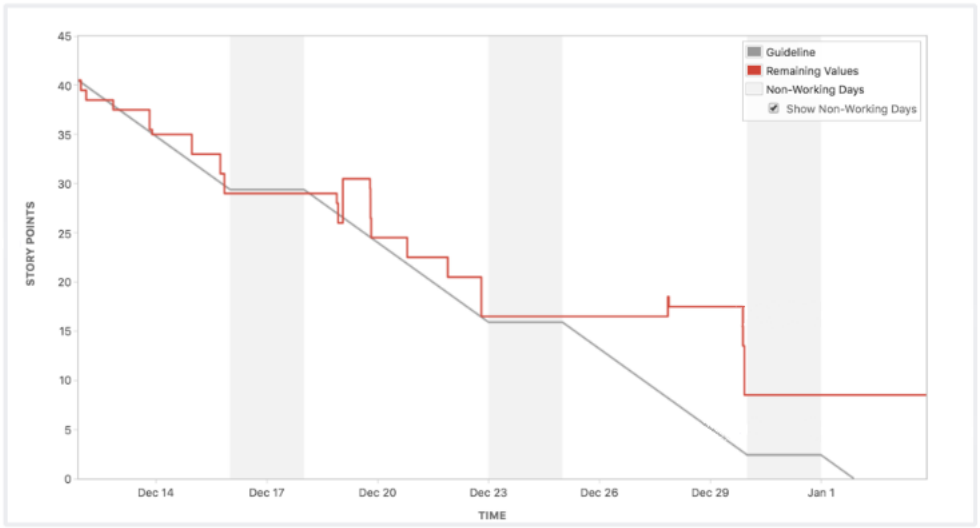
Zlepšení stávajících procesů je zařazeno do první fáze, ale pro zlepšení plánování sprintu bylo třeba nejprve zavést backlog grooming ve fázi tři, díky kterému se plánovací porada výrazně zkrátí a stane se efektivnější.

Plánování se účastní vlastník produktu, vývojový tým a také scrum master, opět jako moderátor. Díky backlog groomingu je tým seznámen s product backlogem dopředu a jednotlivé úkoly pro nadcházející sprint jsou již ohodnoceny, a to v relativních jednotkách.

Tým s vlastníkem produktu prochází product backlog, který je seřazen podle priorit, od shora a přidává jednotlivé úkoly do sprint backlogu, které jsou rovnou přirazeny na jednotlivé vývojáře a testery. Takto se pokračuje do doby, dokud nemají všichni dostatek práce na celý nadcházející sprint, kterou jsou schopni dokončit, ovšem s ohledem na volnou

kapacitu pro případné zaevidované chyby během sprintu. Plánovací porada by neměla trvat déle než 30 minut.

Pro kontrolu množství naplánovaných, již dokončených, a také stále ještě nehotových úkolů použije tým Burndown graf, který poskytuje Jira. Burndown graf zobrazuje pokrok týmu v rámci sprintu, jak lze vidět na následujícím obrázku.



Obr. 16: Burndown graf (upraveno dle atlassian.com)

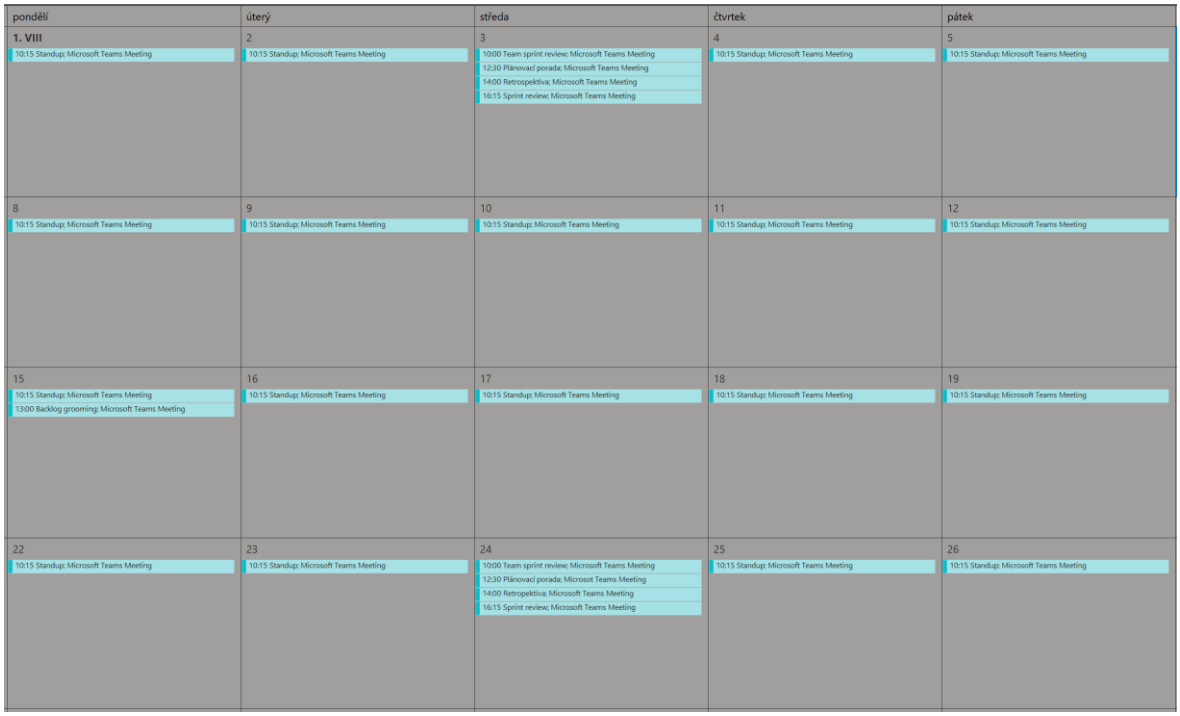
4.3.4 Shrnutí návrhu implementace metodiky

Cílem návrhu není implementovat kompletně celý Scrum. Scrum nabízí mnoho dalších praktik a postupů, které stojí za zvážení zavést, ale v rámci toho návrhu jsou zavedeny ty nejpodstatnější, díky kterým si tým může postavit pevnou základnu, dále se posouvat a až si osvojí veškeré nové, i stávající upravené procesy, může zkoušet zavádět další.

První fáze zahrnuje seznámení týmu s metodikou Scrum jako celek, zavedení standupu každý den, retrospektivy na konci každého sprintu a návrh na seskupení dokumentace do Confluence. Odhad první fáze jsou 3 sprinty (9 týdnů). Druhá fáze zahrnuje oficiální vytvoření a začlenění nové role do týmu, a to vlastníka produktu, odhad druhé fáze je 1–2 měsíců. Třetí fáze je věnována zavedení nových procesů a postupů v týmu, a to konkrétně vlastní sprint review, backlog grooming, pro který je navrhnut efektivnější způsob ohodnocování úkolů, a právě díky backlog groomingu i zlepšení plánovací porady. Třetí fáze

je odhadnuta na 4 sprinty (12 týdnů). Odhad implementace metodiky Scrum dle návrhu této práce činí 6–7 měsíců.

Ve sprintu přibyly 3 důležité schůze, backlog grooming, vlastní sprint review a retrospektiva. Standup je naplánován místo dvou dní na každý den v týdnu, s výjimkou dne věnovaného plánování a sprint review.



Obr. 17 Ukázka sprintu po dokončení fázi (vlastní zpracování)

Po dokončení první fáze bude tým znát Scrum, jeho praktiky, procesy a postupy. Díky standup schůzi každý den nebude docházet k natahování dokončování úkolů, protože každý bude denně sdílet svůj pokrok a případně se mu dostane rychlejší pomoci než po třech dnech samostatného a neúspěšného bádání. Také díky pravidelné retrospektivě se budou členové více cítit jako součást týmu, protože budou moci sdílet své pocity, návrhy, věci, které se jím nelíbily a bude na tyto věci brán ohled a budou se nějakým způsobem řešit.

Po dokončení druhé fáze přibude do týmu nová posila, nebo jen nová role v podobě jedné osoby, a to vlastníka produktu. Ten bude mít dokonalý přehled o požadované a naplánované práci, bude přímo komunikovat a adoptéry a potenciálními zákazníky a tyto nabyté znalosti bude předávat týmu dál, takže budou všichni řádně a včas informováni.

Po dokončení třetí fáze se budou všichni členové cítit zodpovědnější za práci, kterou vykonávají, protože budou dostávat pravidelnou přímou zpětnou vazbou po každém sprintu.

Přibude důležitá schůze, backlog grooming, díky které se zrychlí a zefektivní plánovací porada a kde se tým seznámí předem s product backlogem, V rámci této schůze bude tým ohodnocovat pracnost jednotlivých úkolů. Naučí se ohodnocovat práci v relativních jednotkách, díky čemuž bude schopen lépe plánovat nadcházející sprint a včas doručovat slibovanou funkcionalitu.

5 Závěr

První část diplomové práce je věnována teorii, která je důležitá pro zpracování vlastní části diplomové práce. Teoretická část je zaměřena primárně na popis agilních metodik, ze kterých je na základě vícekriteriální analýzy variant vybrána optimální metodika pro konkrétní vývojový tým. Dané agilní metodiky jsou stručně charakterizovány, jsou představeny stanovené role v rámci metodiky a také jsou vysvětleny základní principy a postupy daných metodik, které jsou základem pro hodnocení stanovených kritérií pro vícekriteriální analýzu variant. V teoretické částí je dále popsán úvod do softwarového inženýrství, modely životního cyklu vývoje softwaru, přehled dalších metodik a modelů a také je stručně představena vícekriteriální analýza, metody pro stanovení vah kritérií a metody stanovení pořadí variant.

V druhé části diplomové práce byla provedena analýza současné situace daného vývojového týmu, jeho charakteristika, aktuální situace a používané procesy a nástroje uvnitř týmu. Byla stanovena hodnotící kritéria jako základ pro vícekriteriální analýzu. Tato kritéria byla stručně popsána a byly jim stanoveny hodnoty. Ve spolupráci autorky s vývojovým týmem byly následně vybrané agilní metodiky ohodnoceny a toho ohodnocení bylo v následující kapitole odůvodněno. Po domluvě s týmem byly stanoveny váhy jednotlivých kritérií. Dále pomocí metody TOPSIS byla vybrána optimální agilní metodika.

Hlavním cílem této práce bylo vytvořit návrh implementace agilní metodiky pro reálný tým. Výstupem vlastní práce je hotový návrh implementace, který je rozvržen do třech fází se stanoveným předpokládaným časovým plánem. Tímto byl splněn hlavní cíl práce.

V rámci návrhu nebyla navrhnuta kompletní implementace celé metodiky, ale pouze vybrané podstatné procesy a postupy, díky kterým lze snadněji pokračovat v implementaci případných dalších postupů a procesů.

6 Seznam použitých zdrojů

Knižní zdroje:

1. ABRAHAMSSON, P.; SALO, O.; RONKAINEN, J.; WARSTA, J.: *Agile software development methods: Review and analysis*, VTT Technical Research Centre of Finland, 2002, 951–38-6010-8.
2. ARLOW, Jim; NEUSTADT, Ila. *UML 2 a unifikovaný proces vývoje aplikací, objektově orientovaná analýza a návrh prakticky.* 2. vyd. Brno: Computer Press, 2007. ISBN: 978-80-251-1503-9.
3. BECK, Kent. *Extrémní programování.* 1. vydání. Praha: Grada Publishing, spol. s.r.o, 2002. ISBN 80-247-0300-9.
4. BECK, Kent; ANDRES, Cynthia. *Extreme programming explained: embrace change*. 2nd edition. Boston, MA: Addison-Wesley, 2005. ISBN 978–0321278654.
5. BELL, Laura; BRUNTON-SPALL, Michael; SMITH. Rich a BIRD, Jim. *Agile Application Security: Enabling security in a continuos delivery pipeline.* Sebastopol: O'Reilly Media, 2017. ISBN 978-1-491-93884-3.
6. BOEHM, Barry. *Spiral development: Experience, principles and refinements*, Carnegie Mellon University, Software Engineering Institute, 2000.
7. BUCHALCEVOVÁ, Alena. *Metodiky budování informačních systémů.* Vysoká škola ekonomická v Praze. Praha: Oeconomica, 2009. ISBN 978-80-245-1540-3.
8. BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů: kategorizace, agilní metodiky, vzory pro návrh metodiky.* Praha: Grada, 2005b. Management v informační společnosti. ISBN 80-247-1075-7.
9. COAD, P.; LEFEBVRE, E.; DE LUCA, J. *Java Modeling In Color With UML: Enterprise Components and Process*. Upper Saddle River: Prentice Hall, 1999.
10. COCKBURN, Alistair. *Use Cases: Jak efektivně modelovat aplikace*. Praha: Computer Press, 2005. 264 s. ISBN 80-251-0721-3.
11. COHN, M.: *Succeeding with agile: software development using scrum*. Upper Saddle River, NJ: Addison-Wesley, 2010, ISBN 03-215-7936-4.
12. FIALA, Petr. *Modely a metody rozhodování.* 2. přepracované vyd. Praha: Oeconomica, 2008. 292 s. ISBN 978-80-245-1345-4.
13. FOTR, Jiří; ŠVECOVÁ, Lenka. *Manažerské rozhodování – Postupy, metody a nástroje*. 3. vyd. Praha: Ekopress, 2016. ISBN 978-80-87865-33-0.

14. FRIEBELOVÁ, J.; KLICNAROVÁ, J. *Rozhodovací modely pro ekonomy*. České Budějovice: Jihočeská univerzita v Českých Budějovicích. 2007.
15. HIGHSMITH, James A. *Adaptive software development: a collaborative approach to managing complex systems*. New York: Dorset House Pub., 2000. ISBN 9780932633408.
16. JABLONSKÝ, Josef. *Operační výzkum: kvantitativní modely pro ekonomické rozhodování.* Praha: Professional Publishing, 2002. ISBN 80-86419-23-1.
17. KADLEC, Václav. *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press, 2004. ISBN 80-251-0342-0.
18. KŘENA, Bohuslav; KOČÍ, Radek. *Úvod do softwarového inženýrství.* Studijní opora. Brno, 2010.
19. LAPLANTE, Phillip A.; NEILL, Colin J. *The Demise of the Waterfall Model Is Imminent. Queue – Game Development.* Volume 1. Issue 10, 2004. 10-15 s. ISSN:1542-7730
20. MYSLÍN, Josef. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016. ISBN 978-80-251-4650-7.
21. PALMER, S.R.; FELSING, J.M. *A Practical Guide to Feature-Driven Development.* Upper Saddle River: Prentice Hall, 2002. 304 s. ISBN 978-0130676153.
22. PAQUETTE, P.; FRANKL, M. *Agile Project Management for Business Transformation Success*. New York: Business Expert Press, 2015. ISBN 978-1-63157-323-1.
23. RÁČEK, Jaroslav. *Softwarové inženýrství II.* Učební materiály. Brno, 2013.
24. SCHNEIDEROVÁ HERALOVÁ, R., BERAN, V., DLASK, P. *Rozhodování: (vstupní data, významnost kritérií, hodnocení variant).* Praha: České vysoké učení technické v Praze, 2011. ISBN 978-80-01-04982-2.
25. SOMMERVILLE, Ian. *Softwarové inženýrství*. Brno: Computer Press, 2013. 680 s. ISBN 978-80-251-3826-7
26. ŠOCHOVÁ, Z.; KUNCE, E. *Agilní metody řízení projektů*. 2. vydání. Brno: Computer Press, 2019. ISBN 978-80-251-4961-4.
27. ŠUBRT, Tomáš a kol. *Ekonomicko – matematické metody*. Plzeň: Aleš Čeněk, 2011. str. 351. ISBN 978-80-7380-345-2.

28. TRIANTAPHYLLOU, E. *Multi-criteria Decision Making Methods: A Comparative Study*. NY: Springer New York, 2000. ISBN 978-0-7923-6607-2.

Online zdroje:

1. BALLARD, Glenn; HOWELL Gregory A. *Lean project management* [online]. 2003. [cit. 27.2.2023]. Dostupné z: https://www.researchgate.net/publication/230899754_Lean_project_management
2. BUCHALCEVOVÁ, Alena. *Metodika Feature-Driven Development neopouští modelování a procesy, a přesto přináší výhody agilního vývoje.* Katedra informačních technologií, VŠE Praha [online]. 2005a. [cit. 28.2.2023]. Dostupné z: https://nb.vse.cz/~buchalc/clanky/tsw2005.pdf
3. EBY, K. *Comprehensive Guide to the Agile Manifesto*. [online]. 2016 [cit. 30. 1. 2023]. Dostupné z WWW: https://www.smartsheet.com/comprehensive-guide-values-principlesagile-manifesto.
4. *IEEE Standard Glossary of Software Engineering Terminology*, in IEEE Std 610.12-1990. [online]. 1990, [cit. 23.3.2023]. Dostupné z: https://ieeexplore.ieee.org/document/159342
5. KLICNAROVÁ J. *Vícekriteriální hodnocení variant – metody* [online]. České Budějovice: Jihočeská univerzita v Českých Budějovicích, [cit. 27.3.20223]. Dostupné z: http://www2.ef.jcu.cz/~janaklic/oa/VHV_II.pdf
6. KOĎOUSKOVÁ, Barbora. *Metoda scrum pro začátečníky: Co to je a jak to funguje.* [online] 2021, Rascasone [cit. 4. 6. 2022]. Dostupné z: https://www.rascasone.com/cs/blog/co-je-scrum-jak-funguje.
7. AGILE. *Manifesto for Agile Software Development*. [online] 2001. [cit. 30. 1. 2023]. Dostupné z http://www.agilemanifesto.org/.
8. SAUTER, Vicki. *Extreme programing* [online]. 2006. [cit. 24.3.2023]. Dostupné z: http://www.umsl.edu/~sauterv/analysis/f06Papers/Hutagalung/
9. *Test Driven Development (TDD).* GeeksforGeeks [online] 2022. [cit. 27.2.2023]. Dostupné z: https://www.geeksforgeeks.org/test-driven-development-tdd/
10. WIDEMAN, Max. *The Role of the Project Life Cycle (Life Span) in Project Management.* [online], 2004. [cit. 25.1.2023]. Dostupné z: http://www.maxwideman.com/papers/plc-models/intro.htm.

7 Přílohy

Příloha 1 – Stanovení vah kritérií pomocí Saatyho metody
Příloha 2 – Metoda TOPSIS

Příloha 1 – Stanovení vah kritérií pomocí Saatyho metody

i \ j	1	2	3	4	5	6	geometrický průměr	váhy
1	1	1/3	5	1/3	3	1/5	0,833	0,0984
2	3	1	7	3	5	1/3	2,172	0,2566
3	1/5	1/7	1	1/5	1/3	1/7	0,255	0,0301
4	3	1/3	5	1	3	1/3	1,308	0,1545
5	1/3	1/5	3	1/3	1	1/5	0,487	0,0575
6	5	3	7	3	5	1	3,411	0,4029
						Σ	8,466	1

Tabulka 7: Výpočet vah

Příloha 2 – Metoda TOPSIS

Kritérium	1	2	3	4	5	6
Váha	0,1	0,26	0,03	0,15	0,06	0,4

Tabulka 8: Váhy

Kritérium	1 max	2 max	3 max	4 max	5 min	6 max
Scrum	10	8	8	10	2	10
XP	7	9	8	8	3	8
FDD	8	6	10	7	6	6

Tabulka 9: Kriteriální matice metody TOPSIS

Kritérium	1 max	2 max	3 max	4 max	5 max	6 max
Scrum	10	8	8	10	4	10
XP	7	9	8	8	3	8
FDD	8	6	10	7	0	6

Tabulka 10: Převod min na max

Kritérium	1 max	2 max	3 max	4 max	5 max	6 max
Suma	213	181	228	213	25	200
Odmocnina	14,5945	13,45362	15,09967	14,59452	5	14,14214

Tabulka 11: Pomocný výpočet pro normalizovanou matici

Kritérium	1 max	2 max	3 max	4 max	5 max	6 max
Scrum	0,68518869	0,5946355	0,5298129	0,68518869	0,8	0,70710656
XP	0,47963208	0,66896493	0,5298129	0,54815094	0,6	0,56568524
FDD	0,54815094	0,44597662	0,66226612	0,47963208	0	0,42426394

Tabulka 12: Normalizovaná kriteriální matice

Kritérium	1 max	2 max	3 max	4 max	5 max	6 max
Scrum	0,06851887	0,15460523	0,01589439	0,1027783	0,048	0,28284262
XP	0,04796321	0,17393088	0,01589439	0,08222264	0,036	0,2262741
FDD	0,05481501	0,11595392	0,019886798	0,07194481	0	0,16970558

Tabulka 13: Vážená kriteriální matice

H	0,069	0,174	0,02	0,103	0,048	0,283
D	0,047	0,115	0,016	0,072	0	0,17

Tabulka 14: Ideální a bazální varianta

Kritérium	1 max	2 max	3 max	4 max	5 max	6 max
Scrum	0	0,000373	0,000016	0	0	0
XP	0,000423	0	0,000016	0,000423	0,000144	0,0032
FDD	0,000188	0,003361	0	0,000951	0,002304	0,0128

Tabulka 15: Vzdálenost variant od ideální varianty Di +

Kritérium	1 max	2 max	3 max	4 max	5 max	6 max
Scrum	0,000423	0,001494	0	0,000951	0,002304	0,0128
XP	0	0,003361	0	0,000106	0,001296	0,0032
FDD	0,000047	0	0,000016	0	0	0

Tabulka 16: Vzdálenost variant od ideální varianty Di -

	Suma di +	di +	Suma di -	di -
Scrum	0,000389	0,01972308	0,017972	0,13405969
XP	0,004206	0,06485368	0,007963	0,08923564
FDD	0,019604	0,14001428	0,000063	0,00793725

Tabulka 17: Vzdálenosti variant od ideální a bazální varianty

	ci	Pořadí
Scrum	0,872	1
XP	0,579	2
FDD	0,054	3

Tabulka 18: Pořadí variant