Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra systémového inženýrství

Bakalářská práce

Analýza agilních metodik vývoje aplikací ve vybraném podniku

Vojtěch Kocura

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Vojtěch Kocura

Systémové inženýrství a informatika
Informatika

Název práce

Analýza agilních metodik vývoje aplikací ve vybraném podniku

Název anglicky

Analysis of agile development methodologies in selected company

---

Cíle práce

Cílem práce je vymezit odlišnosti mezi klasickými a agilními metodikami vývoje aplikací, přiblížit principy fungování jednotlivých metodik a navrhnout zlepšení řízení IT projektů ve vybraném podniku.

1) Charakteristika klasických metodik.
2) Popis a charakteristika agilních metodik.
3) Analýza odlišností mezi klasickými a agilními metodikami.
4) Analýza řízení IT projektů ve vybraném podniku.
5) Návrh na zefektivnění řízení projektů ve vybraném podniku.
6) Zhodnocení návrhů a doporučení dalšího postupu.

Metodika

K dosažení cílů teoretické části vede studování odborných literatur a článků. Poznatky ze zdrojů slouží ke zpracování praktické části.

Praktická část spočívá v pozorování řízení vývoje systémů v daném podniku zabývajícím se podnikovými informačními systémy. Z poznatků je navrhnuto řešení na optimalizaci řízení IT projektů.

Doporučený rozsah práce

30-60 stránek

Klíčová slova

Projektové řízení, agilní metodiky, SCRUM, vývoj IS

Doporučené zdroje informací

AXELOS GLOBAL BEST PRACTICE. *PRINCE2 Agile®.* Norwich: Axelos, 2015. ISBN 978-0-11-331467-6.

AXELOS LIMITED. *Managing successful projects with PRINCE2®.* Norwich: TSO, 2017. ISBN 978-0-11-331533-8.

DOLEŽAL, J. – MÁCHAL, P. – LACKO, B. *Projektový management podle IPMA.* Praha: Grada, 2009. ISBN 978-80-247-2848-3.

PRINCE2 Agile /. ISBN 978-0113314676.

ŠOCHOVÁ, Z. – KUNCE, E. *Agilní metody řízení projektů.* Brno: Computer Press, 2019. ISBN 978-80-251-4961-4.

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Petra Pavlíčková, Ph.D.

Garantující pracoviště

Katedra systémového inženýrství

Elektronicky schváleno dne 29. 10. 2020

doc. Ing. Tomáš Šubrt, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 5. 11. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 05. 03. 2021

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Analýza agilních metodik vývoje aplikací ve vybraném podniku" jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 13.03.2021 _____

Poděkování

Rád bych touto cestou poděkoval paní Ing. Petře Pavlíčkové, Ph.D. za rady, konzultace a vedení bakalářské práce.

Analýza agilních metodik vývoje aplikací ve vybraném podniku

Abstrakt

Agilní řízení vývoje stále více nabývá na významnosti a je stále více využíváno v praxi. Společně s tradičním řízením představuje různé metodiky vývoje aplikací s rozdílnými praktikami, výhodami a nevýhodami.

Teoretická část práce popisuje konkrétní tradiční a agilní metodiky vývoje. Z tradičního řízení přibližuje metodiky Vodopádového modelu, Spirálového modelu a modelu „napiš a oprav.“ Z agilního řízení pak například SCRUM, Kanban, Crystal či Extrémní programování. Práce popisuje praktické využití zmíněných metodik a definuje jejich klady a zápory. Pozornost je dále věnována obecným rozdílům mezi agilním a tradičním způsobem řízení vývoje. Teoretická část také přibližuje standardy projektového řízení PMBOK a PRINCE2.

Praktická část práce je věnována analýze využívaných metodik vývoje v podniku. Na základě studování teorie a pozorování dvou odlišných vývojových týmů jsou definovány využívané metodiky. Dále jsou popsány odchylky od pravidel konkrétních metodik včetně důvodu, proč k odchylkám mohlo dojít. Poté je oběma týmům podán návrh řešení, který by mohl přinést zefektivnění práce a komunikace. V závěru práce je uskutečněn rozhovor se šéfem vývoje jednoho z týmů, kde reaguje na navržené změny.

Klíčová slova: SCRUM, Vodopádový model, agilní metodiky, tradiční metodiky, metodiky vývoje, Extrémní Programování, Kanban, projektové řízení

Analysis of agile development methodologies in selected company

Abstract

Agile development constantly increases its importance, and it is used more and more in practice. Together with traditional project management it offers various methodologies of software development with different best practices, pros and cons.

The theoretical part of the work describes specific traditional and agile development methodologies. From traditional management it mentions Waterfall model, Spiral model and “build and fix” model. From agile management it defines for example SCRUM, Kanban, Crystal methodology, Extreme Programming and so on. The work describes practical usage of mentioned methodologies and defines its advantages and disadvantages. Attention is also devoted to general differences between agile and traditional management. The theoretical part also describes standards of project management PMBOK and PRINCE2.

The practical part of the work is devoted to analysis of used development methodology in specific company. Used methodologies are defined based on theoretical studying and observation in two different development teams. The work describes changes compared to the best practices of each methodology with explanation, why those changes may be made. Also, the suggestion of solution is given to both teams, which could lead to increased efficiency both at work and communication. At the end of the work an interview with head of development of one team is held, where he reacts to the suggestion of solution.

Keywords: SCRUM, Waterfall methodology, agile methodologies, traditional methodologies, development methodologies, Extreme Programming, Kanban, project management

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

S rozvojem počítačů, kdy se projekty staly komplexnější a abstraktnější, se musel rozvíjet i způsob, jak tyto projekty co nejefektivněji řídit. Postupně proto vznikaly tradiční metodiky řízení vývoje, jejímž nejznámějším zástupcem je Vodopádový model. Ten byl paradoxně zprvu popsán jako nefungující model, kvůli své jednoduchosti použití se však v určitém typu vývoje software využívá dodnes. Největším úskalím Vodopádového modelu je fakt, že celý projekt musí být naplánován dopředu a během vývoje je téměř nemožné reagovat na změny požadavků. Postupem času, jak se rozvíjely technologie a na kvalitní software byl kladen stále větší důraz, se nemožnost reakce na změnu stávala stále větším problémem. Vznikaly proto nové tradiční metodiky vývoje, jako například Spirálový model, nebo docházelo k různým modifikacím Vodopádového modelu. Radikálnější změna v řízení projektů přišla se vznikem agilních metodik.

Agilní metodiky zakládají svůj úspěch na iteračním, stále se opakujícím cyklu vývoje, spojeném s častou komunikací se zákazníkem. Kvůli řízení vývoje v cyklech a časté komunikaci tak mohou snadno reagovat na změny požadavků v libovolné fázi projektu. Zástupců agilních metodik je mnoho, mezi nejznámější patří například SCRUM, Extrémní programování a Kanban.

Oba způsoby řízení vývoje, jak tradiční, tak agilní, mají své výhody a nevýhody, které jsou v práci popsány. V teoretické části jsou dále přiblíženy konkrétní metodiky společně s jejich pravidly, praktikami a využitím v praxi. Pozornost je věnována i standardům projektového řízení PMBOK a PRINCE2 nebo často využívaným praktikám v programování. Jmenovitě práce přibližuje například kolektivní vlastnictví kódu, refaktorování či párové programování.

Praktická část se zabývá analýzou využívaných metodik řízení vývoje ve dvou konkrétních developerských týmech. Na základě pozorování praktik týmů je definována metodika, které se vývoj nejvíce podobá. Dále je pozornost věnována odchylkám od standardů a přejímání praktik z jiných metodik. V praxi vývojáři velmi často nedodržují všechna pravidla dané metodiky, neboť to pro daný tým může být nevýhodné. Na druhou stranu mnohdy dochází k přejímání některých výhodných praktik z jiné metodiky.

Na základě těchto odchylek a přejímání je oběma týmům podán návrh řešení, který by mohl zefektivnit rychlost práce a zlepšit komunikaci. V závěru práce je poskytnut rozhovor se šéfem vývoje jednoho z týmů, kde se autor dotazuje mimo jiné na názor ohledně navržených změn ve vývoji.

2 Cíl práce a metodika

2.1 Cíl práce

Bakalářská práce je zaměřena na problematiku tradičních a agilních metodik řízení vývoje softwaru. Hlavním cílem teoretické části je seznámení s konkrétními zástupci obou přístupů k projektovému řízení, jejich pravidla a praktiky. Dále je kladen cíl přiblížit hlavní výhody a nevýhody agilních a tradičních metodik spolu s jejich nejvýznamnějšími rozdíly.

Hlavním cílem praktické části je analyzovat využívanou metodiku vývoje ve vybraném podniku a následně navrhnout řešení, které by mohlo mít kladný vliv na projektové řízení.

2.2 Metodika

Podklady pro zhotovení teoretické části jsou čerpány z odborné literatury a informačních zdrojů. Práce se zabývá obecnými faktory tradičních a agilních metodik, aby bylo možné oba způsoby porovnat. Dále je studium věnováno konkrétním zástupcům obou metodik za účelem přiblížení jejich pravidel, praktik a vhodnosti využití.

V praktické části je na základě těchto znalostí utvořena analýza využívané metodiky v konkrétním podniku. Porovnáním teoretických znalostí s pozorováním v praxi je utvořen návrh řešení pro oba sledované vývojové týmy za účelem zvýšení efektivnosti práce.

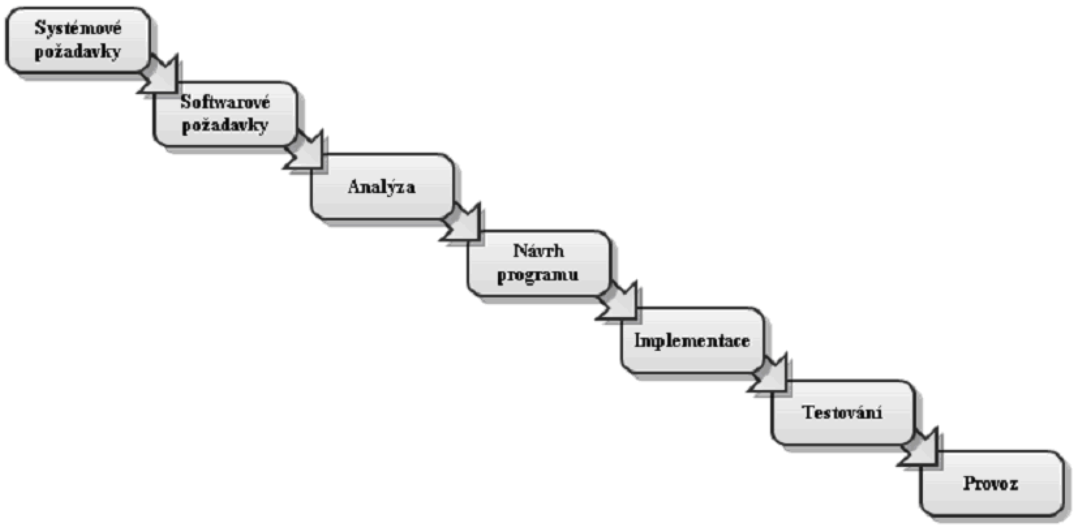
3 Teoretická východiska

3.1 Tradiční metodiky vývoje

3.1.1 Vodopádový model

Vodopádový model, nebo také vodopádový přístup, je jedním z pojetí vývoje softwaru a řízení projektů. Jedná se o nejstarší model popisující cyklus tvorby softwaru, a tudíž se zařazuje mezi tradiční metodiky. Jeho použití je ideální u malých projektů, které mají předem definovaný jasný cíl. Předpokládá se, že se cíl nebude v čase měnit. Z tohoto důvodu je kladen velký důraz na prvotní plánování, aby se vize zákazníka a vývojářů co nejvíce shodovala. [1][2]

První zmínka o vodopádovém modelu pochází z publikace z roku 1970 od W.W. Royce, který vymezil sedm fází vývoje. Následující schéma popisuje konkrétní fáze vývoje přeložené do češtiny.



Obrázek 1: Schéma vodopádového modelu [2]

Různé zdroje uvádí různý počet fází vývoje a různé názvy. Někdy se spojují například fáze systémových a softwarových požadavků do obecnější fáze s názvem Požadavky, nebo se fáze testování zahrne do fáze implementace. Dále jsou více přiblíženy fáze vývoje vodopádového modelu a je vysvětleno, co každá fáze obnáší: [3]

1. **Požadavky:** Ve vodopádovém modelu se klade velmi velký důraz na prvotní požadavky zákazníka na software, což je dáno nemožností rychlé reakce na změnu požadavků během vývoje. V této klíčové fázi celého projektu se tedy shromáždí a řádně zdokumentují všechny zákazníkovy požadavky na software, podle kterých je potom naplánován celý vývoj. Do vývoje již zákazník příliš nezasahuje a na konci projektu je mu předán software, který si objednal.

2. **Design:** Jiný název pro fázi analýzy, kde vývojový tým zpracovává zákazníkovy požadavky na software. Tuto fázi lze rozdělit do dvou menších, a to na logický a fyzický design. V logickém designu tým přemýšlí nad možnostmi řešení a vytváří hardwarové požadavky na projekt. Ve fyzickém designu pak své teoretické myšlenky převádí do konkrétních specifikací.

3. **Implementace:** Jedná se o fázi, kdy se píše samotný kód softwaru na základě předchozích specifikací. Některé zdroje rozlišují fázi implementace a testování, některé zdroje uvádí testování jako součást implementace. V rámci testování se provádí unit testy, a jakmile je vše řádně otestováno, přechází se k další fázi vývoje.

4. **Ověření:** V této fázi je produkt prezentován zákazníkovi a řeší se, zdali byla dodržena smluvní dohoda a zdali se výsledek s touto dohodou shoduje.

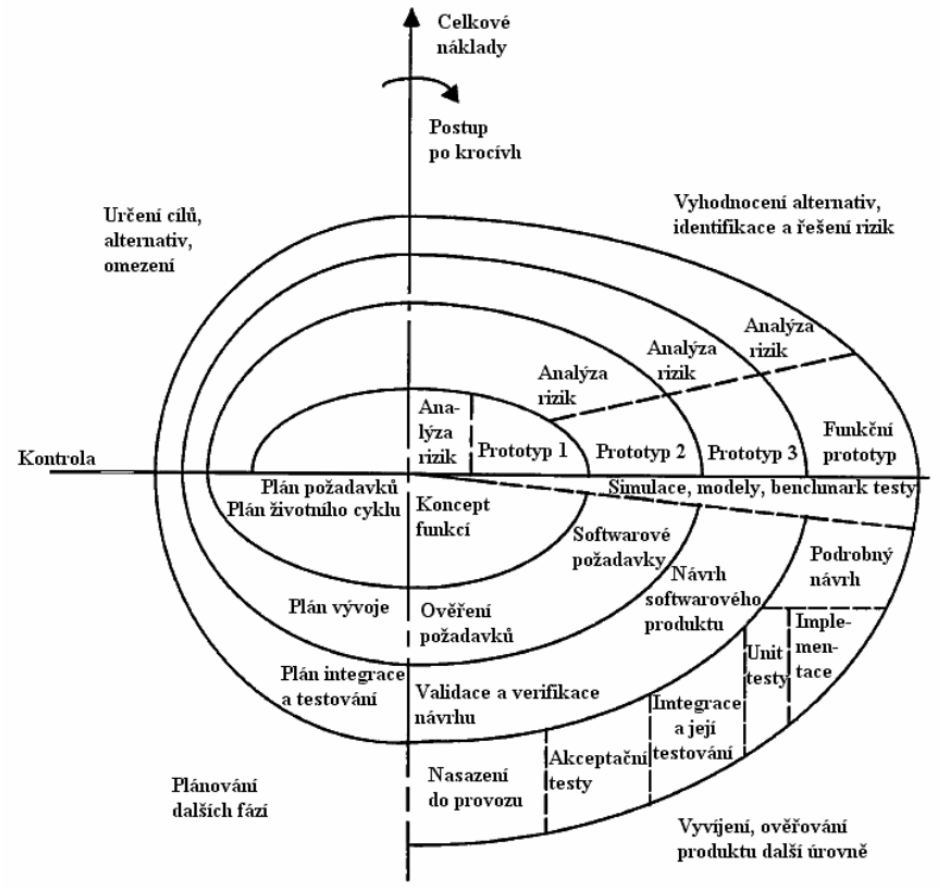
5. **Provoz:** Poslední fází vodopádového vývoje je nasazení softwaru do provozu a jeho následná údržba. Zákazník používáním systému objevuje chyby, které musí vývojový tým průběžně odstraňovat.

Schéma také naznačuje, proč se tomuto modelu říká vodopádový. Celý tento přístup je brán sekvenčně, tudíž další fáze vývoje nemůže nastat dříve, než skončí fáze předchozí. Jak již bylo zmíněno, tento přístup je vhodný pouze pro projekty s jasným cílem, neboť cíl za chodu projektu lze měnit jen velmi těžko. Další nevýhodou je velmi pracná a nákladná oprava chyb, pokud nejsou zachyceny v prvotních fázích vývoje. Naopak výhody vodopádového modelu jsou v jeho jednoduchosti oddělených fází a také v tom, že lze celý projekt dopředu naplánovat, tudíž lze odhadnout časovou i finanční náročnost. [4]

V dnešní době jsou velkým konkurentem vodopádového modelu agilní metodiky. Kvůli nemožnosti reagovat na změnu se vodopádový model využívá v jednodušších projektech, v těch komplexnějších stále častěji vidíme využití metodik agilních. Existují různé modifikace vodopádového modelu, které se snaží být více „agilnějšími." Příkladem může být Sashimi model, ve kterém se jednotlivé fáze vývoje překrývají a dochází tak k časnějšímu objevení chyb. [5] Dále se využívají jiné modely tradičních metodik, nebo se přechází k agilnímu pojetí vývoje, kde je reakce na změnu ještě rychlejší.

3.1.2 Spirálový model

Spirálový model poprvé popsal ve své publikaci z roku 1986 Barry Boehm. Na rozdíl od vodopádového přístupu, který probíhá sekvenčně, je spirálový model založen na iteracích. Je rozdělen do čtyř fází, nebo lépe řečeno do čtyř kvadrantů, jak je znázorněno na následujícím schématu. [6] [7]



Obrázek 2: Schéma spirálového modelu [6]

Životní cyklus na tomto schématu začíná na vnitřní spirále ve druhém kvadrantu. Vývoj prochází všemi fázemi, které se na sebe iterativně nabalují, až dojde k nasazení softwaru do provozu. Schéma je pouze ilustrativní a v praxi tak není počet dokončených fází definován. První fáze je určení cílů, alternativ a omezení, druhá fáze je vyhodnocení alternativ a řešení rizik, třetí fáze je vývoj a ověřování dalších úrovní produktu a poslední čtvrtá fáze je plánování dalších fází. Mezi každými fázemi dochází k analýze rizik. Výhodou spirálového modelu je tudíž zhodnocení rizik před každou fází. Spirálový model také svou iterační povahou reaguje na změny ve vývoji značně snadněji než klasický vodopádový model. Nevýhodami tohoto modelu jsou vysoké finanční náklady a nemožnost odhadnutí časové náročnosti. Pro své vysoké náklady je doporučován pro velké a složité projekty. [6] [7]

3.1.3 Model „Napiš a oprav“

Dalším, ne příliš používaným a nedoporučovaným modelem vývoje je model „napiš a oprav,“ anglicky „build and fix“. Tento model se skládá pouze ze dvou fází. První fáze je vývoj softwaru pro zákazníka bez jakéhokoliv testování a bez jakýchkoliv specifických a designových požadavků. Poté se software předá zákazníkovi, a pokud ten není s výsledkem spokojen, přechází se ke druhé fázi, kterou je opravování nedostatků. Takto se proces vývoje opakuje až do chvíle, kdy je zákazník s produktem zcela spokojen. Výhodou je jednoduchost modelu, protože má pouze dvě fáze.

Nevýhodami je velká časová náročnost, velké finanční náklady a prakticky nemožnost údržby softwaru. Kvůli převládání nevýhod modelu se v praxi příliš nevyužívá. [8] [9]

3.2 Agilní metodiky vývoje

3.2.1 Principy agilních metodik

Agilní metodiky vznikly později než ty tradiční, a to z důvodu umožnění rychlejší reakce na změnu přání zákazníka. Určitý řád agilním metodikám nastolil Manifest agilního vývoje, který byl vytvořen roku 2001 sedmnácti autory. [10] Manifest hovoří o čtyřech hlavních hodnotách: [11]

1. Jednotlivci a interakce před procesy a nástroji
2. Fungující software před vyčerpávající dokumentací
3. Spolupráce se zákazníky před vyjednáváním o smlouvě
4. Reagování na změny před dodržením plánu

V závěru manifestu je navíc poznámka: „Jakkoliv jsou body napravo hodnotné, bodů nalevo si ceníme více.“ [11] První hodnota manifestu dává přednost dobré spolupráci v týmu před nejlepšími nástroji. Ve druhé hodnotě lze pozorovat rozdíl v psaní dokumentace u tradičních a agilních metodik. Agilní přístup dává větší hodnotu praktickému vyzkoušení softwaru zákazníkem a následné zpětné vazbě před studováním obsáhlých manuálů. Třetí bod manifestu klade větší důraz na pravidelnou komunikaci se zákazníkem a tolik nelpí na dodržení původní dohody, protože vize zákazníka se může v čase měnit. Poslední bod dává zákazníkovi možnost již zmíněných změn a celkově plán vývoje není tak striktně daný, jako u tradičních metodik. [12] Častá a pravidelná komunikace se zákazníkem spolu se schopností reagovat na změnu jsou ty nejzásadnější rozdíly, které oddělují tradiční a agilní přístup vývoje aplikací.

3.2.2 SCRUM

Scrum je nejrozšířenější a nejvíce oblíbenou agilní metodikou. Vývoj probíhá v pravidelných iteracích a své principy staví na beztřídním samostatně organizovaném vývojovém týmu, kde si jsou všichni rovni. Velký důraz je kladen na dodávání softwaru zákazníkovi v krátkých a pravidelných intervalech, takzvaných sprintech. S častými release aplikace je spojena i častá zpětná vazba od zákazníka, aby se vize obou stran co nejvíce sjednotila. Pravidelná komunikace také umožňuje reagovat na změny v přání zákazníka, což je hlavním principem agilních metodik.

3.2.2.1 Role v týmu

Ačkoliv je Scrum tým beztřídní a samostatně organizován, přece jen rozlišujeme určité role v týmu. Mezi nejběžnější role patří tyto:

Developer

Jednou z rolí v týmu jsou developeři, jejichž primárním úkolem je mít funkční produkt na konci sprintu. Funkčního produktu tým dosáhne splněním takzvaného sprint goal neboli cíle celého sprintu. K tomu slouží sprint backlog, což je seznam funkcionalit, které se mají během daného sprintu udělat. Developeři si sami vybírají položky z product backlogu do sprint backlogu tak, aby byli schopni splnit sprint goal včas. Zmínkou o product backlogu se dostáváme ke druhé roli scrum týmu, kterou je Product owner. [12] [13]

Product owner

Product owner je pomyslný vlastník produktu. Je součástí scrum týmu a funguje jako prostředník mezi zákazníkem a zbytkem týmu. Jeho hlavním úkolem je definování produktové vize zákazníka a sjednocení cíle celého projektu pro obě strany. Na základě komunikace pak sestavuje product backlog, což je seznam funkcionalit, které přináší zákazníkovi nějakou hodnotu. Jednotlivým funkcionalitám přiděluje prioritu a rozděluje je na tak malé části, aby jim zbytek vývojového týmu rozuměl. [13] Praktiky Scrumu uvádí, že by měl product owner trávit okolo 80 % času u zákazníka a 20 % času v týmu. [12]

Scrum master

Další důležitou rolí je Scrum master. Jeho hlavním úkolem je zajistit správný chod scrum vývoje. To znamená vzdělávat zbytek týmu v teoretických praktikách scrumu, dohlížet na jejich dodržování, řídit schůzky týmu a obecně pomáhat k dosažení cílů. [14] Zuzana Šochová ve své knize Agilní metody řízení projektů uvádí, že Scrum master by neměl být technicky nejzkušenějším členem týmu, neboť by dával svým metodám a nápadům větší váhu než nápadům zbytku týmu. Také by se v praxi neměli žádné role kombinovat, tudíž jeden člen by měl mít přiřazenou pouze jednu roli. [12]

Manažer

Někdy se také setkáme s rolí manažera. Jeho funkce však nespočívá v delegování úkolů a dohlížení na pracovní tempo developerů, jak je tomu v běžné situaci, protože je celý tým samostatně organizovaný. Jeho hlavním úkolem je zajistit celému týmu vhodné pracovní prostředí a případně pomáhat Scrum masterovi s obtížnějšími úkoly. [12]

3.2.2.2 Cyklus vývoje pomocí Scrumu

Důležitou praktikou v rámci vývoje aplikací metodikou Scrum jsou pravidelné a časté schůzky celého vývojového týmu. V následujících kapitolách je chronologicky znázorněn průchod jedním sprintem.

Backlog grooming

Tento meeting se pořádá většinou na začátku a uprostřed sprintu. Product owner představuje položky z product backlogu developerům tak, aby všichni pochopili funkcionality stejně. Cílem je tedy pochopení celého product backlogu, což je seznam funkcionalit vytvořených product ownerem. Funkcionality se rozdělují do tří úrovní podle toho, jak jsou velké a složité. Nejprve product owner představí ty největší a zatím velmi nedetailní funkcionality, takzvané Epicy. Epic funkcionality se dají rozdělit na více menších a konkrétnějších funkcionalit, Super User Stories, které product owner představí jako druhé. Poslední pak na řadu přichází User Stories, což jsou dostatečně malé položky product backlogu, které mohou být naplánované do sprintu. Jakmile je celý tým seznámen s položkami product backlogu, přechází se na plánování sprintu. [12] [15]

Plánování sprintu

Vždy před začátkem nového sprintu by měla být svolána schůzka, která poslouží k naplánování sprintu následujícího. Product owner na tomto meetingu představuje sprint goal společně s položkami z product backlogu, které mají velkou prioritu. Na základě cíle sprintu se společně s developery přidávají položky z product backlogu do sprint backlogu tak, aby byl cíl splněn a developeři měli dostatek času na vývoj aplikace. Celá schůzka je řízena Scrum masterem, který dohlíží na hladký chod meetingu a ujišťuje se, že se developeři s product ownerem domluví na společném cíli produktu a sprint backlogu. [16]

Daily Scrum

Daily scrum jsou každodenní schůzky týmu vedené Scrum masterem. Podle Zuzany Šochové v knize Agilní metody řízení projektů by měly schůzky trval přibližně 5 minut, [12] příručka The Scrum Guide uvádí přibližně 15 minut. [13] Schůzka se většinou koná každé ráno a účastníci by měli stát, aby se zajistila dynamičnost rozhovoru a nedocházelo tak ke ztrátě pozornosti. Proto se tento meeting také často označuje jako daily standup. Na daily scrumu vývojáři hovoří, na čem pracovali včera, na čem budou pracovat dnes a zdali narazili na nějaká problémy. Schůzka slouží k pozorování, jak se tým blíží ke splnění cíle zadaném pro tento sprint, tedy ke sprint goal. [12] [13] [17]

Sprint

Sprint je samotný proces vývoje softwaru. Je fixně dlouhý, většinou trvá dva týdny až jeden měsíc. Developeři vyvíjí funkcionality ve sprint backlogu. Cílem však není odškrtat všechny položky ve sprint backlogu, ale dosáhnout předem stanoveného sprint goal. Během sprintu se realizují již zmiňované denní schůzky. [12]

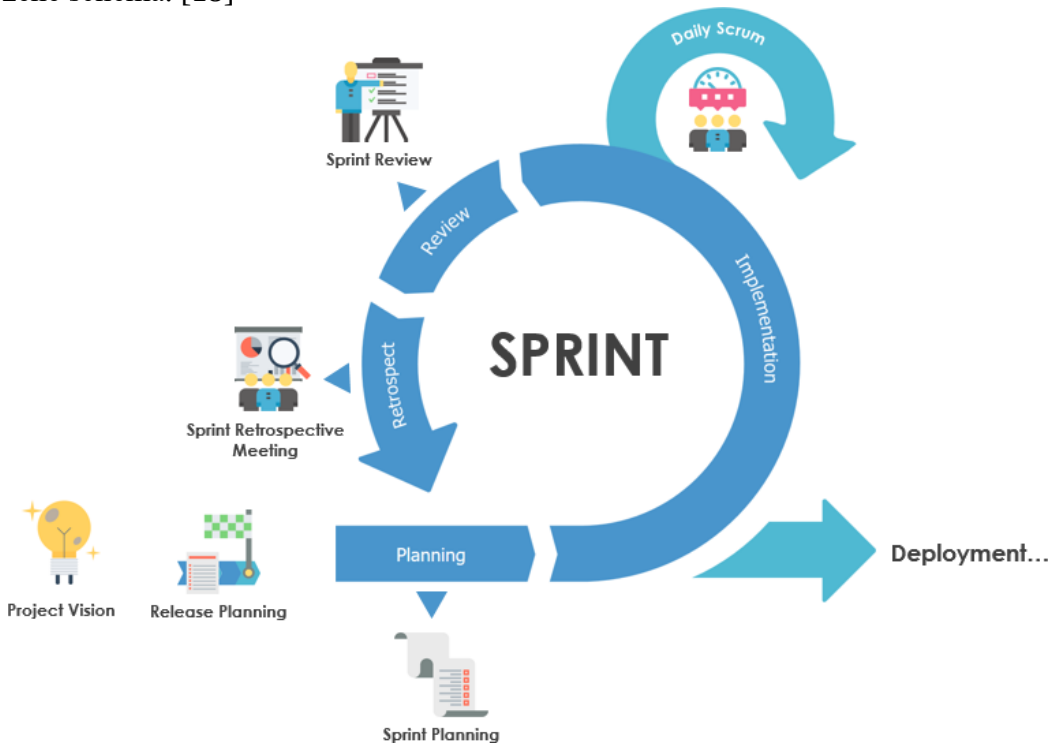
Sprint review

Sprint review je pak schůzka konaná po skončení sprintu. Slouží ke zkontrolování výsledku sprintu a k jeho následné prezentaci zákazníkovi.

Na základě zpětné vazby se diskutuje o dalším postupu ke splnění celého produktového cíle, anglicky product goal. [13]

Retrospektiva sprintu

Retrospektiva se stejně jako review koná po skončení sprintu. Rozdíl je však v tom, že retrospektiva do hloubky zkoumá individuální problémy členů týmu s cílem co nejvíce zvýšit efektivitu ve sprintu následujícím. Schůzky by se měli zúčastnit developeři, scrum master i product owner. Diskutuje se o praktikách, které fungují a nadále by se měly používat, ale také o těch, kde byl nějaký problém a tým hledá způsob, jak se problému zbavit. Maximální doporučovaná délka trvání schůzky jsou tři hodiny při sprintu dlouhém jeden měsíc. Celý iterační cyklus sprintu od plánování po retrospektivu znázorňuje přiložené schéma. [18]



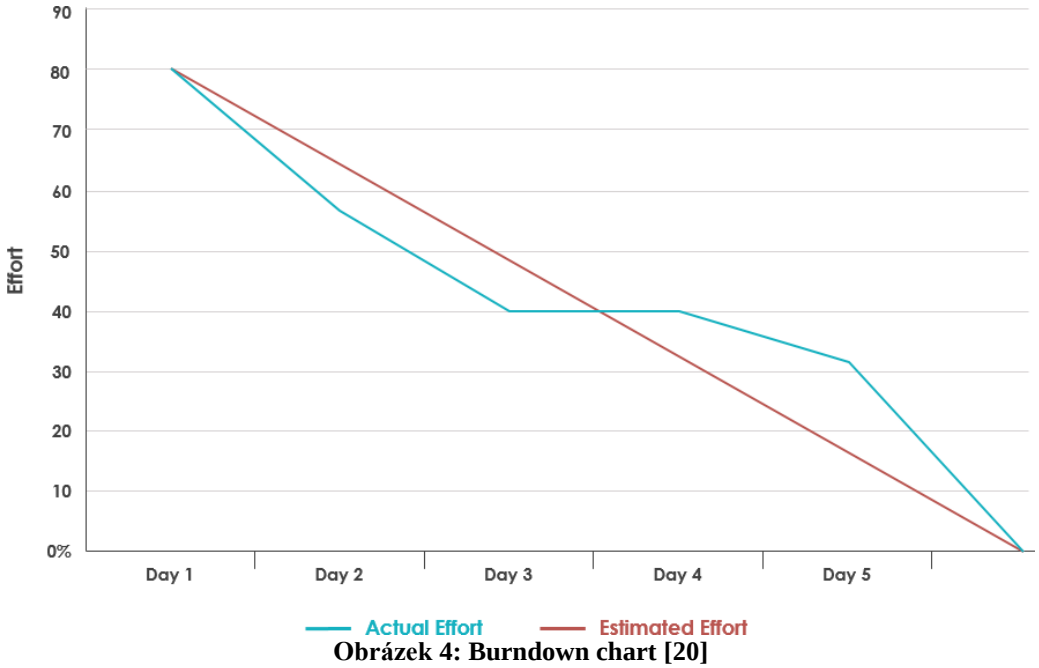
Obrázek 3: Iterační cyklus sprintu [18]

3.2.2.3 Definition of done

Definition of done je seznam podmínek, které musí funkcionalita splňovat, aby se mohla prohlásit za hotovou. Tento seznam podmínek je stejný pro všechny user stories a neměl by se během vývoje nijak zásadně měnit, pouze může docházet k drobnému zpřísňování podmínek. Na konkrétním znění definition of done se domlouvá product owner spolu s developery. Jestliže user story nesplňuje všechny podmínky, ještě není funkcionalita hotová a musí se vrátit zpět do product backlogu. V praxi by seznam podmínek mohl vypadat třeba takto: napsaný kód, otestováno, napsána dokumentace, funguje na test serveru, schváleno product ownerem. [19]

3.2.2.4 Burndown chart

Burndown chart je grafické znázornění zbývající práce s ohledem na zbývající čas. Je velmi důležitý v agilním vývoji, zejména pro Scrum, ale může se objevit skoro v každých metodikách vývoje. Na vertikální ose grafu se většinou zobrazuje počet zbývající práce a na horizontální ose zbývající čas do nějakého termínu odevzdání, například do konce sprintu. Součástí grafu je také přímka, která znázorňuje předem stanovený odhad, jak rychle by měl vývojový tým pracovat, aby stihl všechnu práci v termínu. Porovnáním odvedené práce se zbývajícím časem pak lze posoudit, jestli tým práci nestíhá, nebo naopak odvedl větší množství práce, než se původně odhadovalo. [20] Příklad burndown chartu lze pozorovat na následujícím schématu. Červená přímka znázorňuje odhad rychlosti práce, modrá křivka skutečnou rychlost práce. Nachází-li se křivka pod přímkou, vývojový tým pracuje rychleji, než se předpokládalo.



Obrázek 4: Burndown chart [20]

3.2.3 Extrémní programování

Další agilní metodikou je Extrémní programování. Popsal ji Kent Beck v knize Extreme Programming Explained. [21] Tato metodika však nepřichází s žádnými novými praktikami, jako tomu bylo například u Scrumu, ale pouze dohání do extrémů již zažité praktiky tradičního vývoje tak, aby se zajistilo pravidelné a časté odevzdávání verzí zákazníkovi a aby bylo možné reagovat na zpětnou vazbu. Například, pokud se v týmu osvědčilo testování kódu, tak se bude testovat vše a pořád pomocí unit testů. Osvědčila se jednoduchost kódu? Odteď se bude psát co nejjednodušší kód, který je právě potřeba. Do extrému se také dotáhly iterační cykly vývoje. U Scrumu mluvíme o sprintech dlouhých dva týdny až měsíc, v Extrémním programování jsou iterace dlouhé v řádu dní či hodin. Sebemenší funkcionalita, která byla řádně otestována, se hned vkládá do produkční verze programu. [22] [23]

Ačkoliv může tato metodika působit lehce chaotickým dojmem, tak si velmi zakládá na dobré komunikaci se zákazníkem i mezi členy týmu.

V praxi tak dochází k párovému programování a ke kolektivnímu vlastnictví kódu. Velký důraz je také kladen na zpětnou vazbu, která je rychlá a častá, neboť programátoři píšou jednotkové testy téměř na všechny části kódu. V extrémním programování existuje dvanáct základních praktik, které by měl vývojový tým dodržovat. Mezi některé patří vydávání malých verzí, párové programování, práce čtyřicet hodin týdně, jednoduchost celého kódu či metafora. Metafora slouží k pojmenování částí systému tak, aby si vývojáři pojmy lépe zapamatovali a mohli o softwaru snadněji komunikovat. [22] [24]

3.2.3.1 Cyklus vývoje

Vývoj začíná zadáním, kde se sepíšou user stories a zákazníkovy požadavky na software. Zákazník může během procesu své požadavky měnit. Po sepsání user stories přichází fáze plánování, kde se vytvoří časový harmonogram a jednotlivé iterace. Následuje fáze designu, kde se u Extrémního programování cení jednoduchost a nepřidávají se žádné prvky, dokud neproběhne komunikace se zákazníkem. Poté přichází na řadu programovací část, kde se nejdříve píšou jednotkové testy a až potom program. Zajímavostí je, že veškeré programování probíhá ve dvojicích, podle praktik pair programmingu. Poslední fází je pak testování, kdy musí kód projít všemi jednotkovými testy i celkovým testováním softwaru. [22] [23] Na následujícím schématu lze pozorovat cyklus vývoje Extrémního programování a časový odhad toho, jaká je časová náročnost jednotlivých praktik.

Obrázek 5: Schéma cyklu vývoje Extrémního programování [25]

3.2.4 Kanban

S metodikou Kanban, která upravovala řízení produkce, přišel poprvé Taiichi Ono v polovině dvacátého století ve firmě Toyota. Myšlenka celé metodiky spočívá ve vyrábění součástek až ve chvíli, kdy je to skutečně potřeba.

Poptávka zákazníků se totiž může měnit a skladování nepotřebných součástek může být nákladné. Celý proces byl realizován pomocí Kanban kartiček, kdy každá součástka měla přiřazenu jednu kartičku až do doby, dokud nebyla prodána. Když se tak stalo, kartička putovala do výrobny jako signál pro zhotovení nové součástky. [26]

Myšlenku Kanbanu převzaly také metodiky vývoje softwaru. Týmy řízené Kanbanem začaly dodávat zákazníkovi software přesně na čas a nevyvíjejí navíc nic, než je v danou chvíli třeba. Klíčem k tomu je minimalizovat rozpracované položky a pevně si stanovit počet funkcionalit, na kterých se bude v danou chvíli pracovat. To týmu umožňuje vysokou časovou flexibilitu. Převzala se i myšlenka vizualizace pokroku pomocí Kanban kartiček, k čemuž se využívá Kanban tabule. [27]

3.2.4.1 Kanban tabule

Kanban tabule, ať už fyzická či virtuální, se využívá k vizualizaci pokroku ve vývoji. Má tři sloupce, které rozlišují, v jakém stádiu se nachází jednotlivé user stories. Konkrétně se jedná o sloupce „to do," „in progress" a „done" neboli stádia user stories které se teprve musí udělat, které jsou rozpracované a které jsou již hotové. Někdy si podniky Kanban tabuli lehce upravují a občas se setkáme s tabulí, která má sloupců více, jako je tomu například na následujícím obrázku.

Obrázek 6: Kanban tabule [27]

K vizualizaci na fyzické Kanban tabuli dochází pomocí lístečků, kde si vývojáři vybírají user stories ze sloupce ještě nerozpracovaných user stories a následně je lepí do příslušného sloupce rozpracování. [26] V praxi se však více používají virtuální Kanban tabule, kde jednotlivé user stories mají přiřazeného developera a stupeň priority.

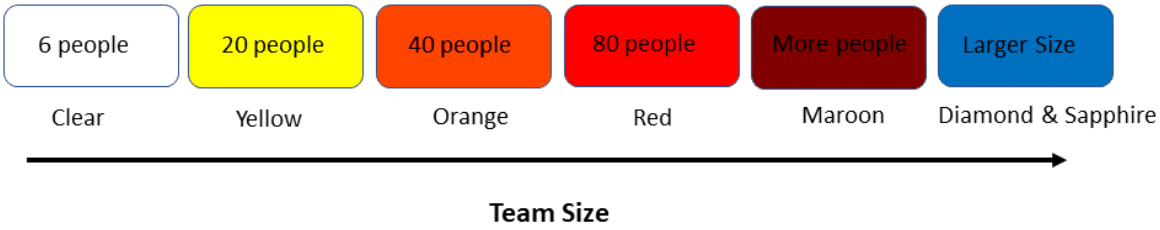
Metodika Kanban se často doplňuje o praktiky jiných metodik, jako například o pravidelnou retrospektivu, denní meetingy či backlog s user stories. [12]

Naopak v praxi agilní týmy využívající jinou metodiku, jako například Scrum, velmi často využívají vizualizace pokroku vývoje a využívají k tomu právě zmiňovanou Kanban tabuli.

3.2.5 Metodika Crystal

Tuto agilní metodiku vytvořil Alistair Cockburn, který stál také u zrodu agilního manifestu (viz kapitola Principy agilních metodik). Metodika Crystal byla vytvořena pro IBM roku 1991 pro objektově orientované projekty.

Na rozdíl od ostatních metodik, které mají pevně stanovené praktiky bez ohledu na velikost týmu, nabízí metodika Crystal větší prostor pro modifikaci. Alistair Cockburn nahlíží na každý projekt jako na unikátní, tudíž pro něj neměla pevná pravidla velký význam. Jeho cílem bylo spíše dát týmům možnost zvolit si takové praktiky ve vývoji projektu, které přesně sedí na velikost týmu i obsah projektu. Cockburn proto definoval doporučené praktiky a postupy, které se mění podle velikosti týmu. Některé praktiky je však doporučené dodržovat, ať je tým jakkoliv velký. Jedná se například o pravidelné a rychlé dodávání softwaru, realizování retrospektivy spojené s poučením se z vlastních chyb, a v neposlední řadě například podprahová komunikace. Poslední zmíněný princip hovoří o tom, že všichni vývojáři by měli pracovat na stejném místě pro co největší zefektivnění komunikace. Metodiky, které jsou doporučovány s ohledem na velikost týmu, Cockburn popsal v jednotlivých kategoriích, které pojmenoval podle barev následovně: [28]



Obrázek 7: Velikosti týmů v metodice Crystal [28]

1. **Crystal Clear:** Bílá barva, určeno pro týmy o velikosti 1-6 lidí. Metodika zde doporučuje například stanovit si pevnou cenu za produkt či provádět dokumentaci.

2. **Crystal Yellow:** Žlutá barva, určeno pro týmy o velikosti 7-20 lidí. Doporučuje se vlastnictví kódu každým vývojářem, jednotliví developeři tudíž pracují každý na svém kódu. Jelikož je tým ještě poměrně malý, tak by měl získávat zpětnou vazbu přímo od zákazníka, aby se zamezilo neporozumění nepřímou komunikací. Dále by měl tým praktikovat například automatizované testování k odhalení chyb.

3. **Crystal Orange:** Oranžová barva, určeno pro týmy o velikosti 21-40 lidí. Alistair Cockburn uvádí, že projekty týmu o této velikosti by měli trvat jeden až dva roky. Dále by se jednotliví členové měli rozdělit do skupin podle svého profesního zaměření. Doporučená doba mezi jednotlivými release je pak tři až čtyři měsíce.

4. **Crystal Red:** Červená barva, určeno pro týmy o velikosti 41-80 lidí. Zde se doporučuje nasazení tradičních metodik vývoje.

5. **Crystal Maroon:** Hnědá barva, určeno pro týmy o velikosti 81-200 lidí. Praktiky se již příliš neliší, vyvíjeny by měly být opravdu velké projekty. [28]

Poslední, lehce oddělené jsou metodiky Crystal Diamond a Crystal Sapphire. Na rozdíl od předchozích metodik, které řešily velikost týmu, tyto řeší kritičnost projektu. Alistair Cockburn definoval čtyři situace, které mohou nastat při selhání systému. První, s nejmenším dopadem, je ztráta komfortu. Systém lze stále využívat, ale stojí to větší úsilí. Druhou situací je malá ztráta peněz, která však nebude mít větší dopad na fungování firmy. Třetí situací je pak větší ztráta peněz, kterou již firma znatelně pocítí. Poslední, nejhorší situací, je pak ztráta životů. [29]

3.2.5.1 Role

Metodika Crystal Clear je výjimečná v tom, že se skládá pouze z jednoho týmu. Ostatní metodiky mají sice týmů více, ale Alistair Cockburn popsal jednotlivé role, které by měl každý tým obsahovat. Jednou z hlavních rolí je Sponzor, který má na starosti financování projektu. Další je Uživatel, který je obdobný například roli Product Owner ve Scrumu. Jedná se o zákazníka či jeho zástupce, který daný produkt testuje a poskytuje zpětnou vazbu. Dále tým musí obsahovat hlavního vývojáře, ostatní vývojáře a testery. Role Písař má pak na starosti tvorbu dokumentace. Jelikož je rolí poměrně hodně a členů týmu málo, tak v praxi běžně jeden člověk zastává vícero rolí. [28] [29]

3.2.6 Feature Driven Development

Další agilní metodikou je Feature Driven Development, zkráceně FDD, která je vhodná pro velké a složité projekty. Metodika byla poprvé popsána v knize Java Modeling in Color with UML, jejímž autorem je Peter Coad a spol. [30] Poprvé byla aplikována v praxi u projektu v bance roku 1997 v padesátičlenném týmu pod vedením Jeffa De Luca a spolupracovníků. [31]

Jedná se o metodiku založenou na pravidelných iteracích. Na rozdíl od Scrumu, který je založen na pravidelném odevzdávání balíčků funkcionalit zákazníkovi, se Feature Driven Development soustředí na malé užitné vlastnosti pro zákazníka, tzv. features. To umožňuje vydávat funkcionality ještě rychleji. Iterace trvá dva týdny, a pravidlem této metodiky je, že pokud je užitná vlastnost natolik velká, že se nestihne udělat za dobu jedné iterace, tak se musí rozdělit na několik menších. FDD si více než ostatní agilní metodiky zakládá na tvorbě dokumentace, která slouží ke komunikaci vývojářů, a tudíž se nesvolávají pravidelné meetingy tak často. Celý vývoj je rozdělen do následujících pěti procesů: [31] [32]

1. **Vypracování celkového modelu.** Nejdříve tým sbírá data, aby co nejlépe porozuměl zákazníkovi. Učí se zákazníkovu vizi projektu a cíle, kterých chce dosáhnout. Na základě těchto znalostí pak tým sestaví hrubý návrh celkového modelu.

2. **Sestavení seznamu užitných vlastností.** V tomto procesu se hrubý model rozdělí na užitné vlastnosti tak velké, aby práce na nich nepřesáhla jednu iteraci, tudíž dva týdny. Všechny užitné vlastnosti, již podle názvu, přináší užitek zákazníkovi a lze je přirovnat k jednotlivým user stories ve Scrumu (viz. kapitola Scrum).

3. **Plánování užitné vlastnosti.** V tomto procesu se určí pořadí, ve kterém se budou jednotlivé features realizovat, a přiřadí se jim termín dokončení a konkrétní vývojáři.

4. **Návrh užitné vlastnosti.** Zde se určí, kdo bude pracovat na jaké části užitné vlastnosti.

5. **Realizace užitné vlastnosti.** V této fázi je vlastnost otestována a vytvoří se její prototyp. Vlastnost se poté vloží do nové verze softwaru.

3.2.7 Rational Unified Process

Rational Unified Process, zkráceně RUP, je metodika vývoje od firmy Rational. Tato firma také vyvinula Unified Modeling Language (UML), což je grafický jazyk pro vizualizaci, na kterou klade tato metodika velký důraz. Celkově vymezuje Rational Unified Process následujících šest best practices: [33]

1. **Iterativní vývoj:** Stejně, jako většina agilních metodik, i RUP využívá iterativní vývoj. Umožňuje tak reagovat na změny v požadavcích. Množství práce přidané jednou iterací se označuje jako přírůstek. Jednotlivými fázemi je plánování, správa požadavků, analýza a návrh, implementace, testování, zhodnocení iterace a plánování iterace následující. Jedna iterace by neměla být delší než tři měsíce.

2. **Průběžné ověření kvality:** I u této metodiky platí, že čím později se chyby odhalí a opraví, tím je oprava nákladnější. Proto se přibývající funkcionality neustále testují v rámci iterativního vývoje a dochází k průběžnému hodnocení kvality.

3. **Vizuální modelování:** RUP vyzdvihuje význam vizualizace komponent a funkcionalit tím, že využívá již zmiňovaný Unified Modeling Language, který firma za tímto účelem vyvinula. Vizualizace funkcionality napomáhá k lepší reprezentaci a zlepšuje tak komunikaci.

4. **Řízení změn:** V agilním světě musí být vývoj schopný reagovat na změnu, tudíž se v rámci řízení změn dokumentují veškeré změny provedené v kódu.

5. **Komponentová architektura:** Řešený problém lze rozdělit na dílčí komponenty, které se poté mohou rozdělit konkrétním týmům na zpracování. Rozdělení na komponenty tak umožňuje snadnější rozvržení práce mezi týmy a vývojáři. Již hotové komponenty se mohou znovu použít, což zvyšuje efektivitu práce.

6. **Správa požadavků:** Rational Unified Process, na rozdíl od tradičních metodik, počítá s možností zákazníka měnit své požadavky během vývoje. Všechny změny na požadavky jsou proto řádně dokumentovány.

Celý projekt se rozděluje do čtyř fází a v každé z nich probíhá několik iteračních cyklů. Jednotlivými fázemi je zahájení projektu, rozpracování, konstrukce a zavedení. Cílem zahájení projektu je pochopit požadavky zákazníka a sestavit požadavky na systém. Na konci fáze rozpracování by měl být hotový spustitelný software a během konstrukce by se měl produkt implementovat a integrovat do systému. V poslední fázi zavedení se software uvede do provozu a probíhají malé změny na základě zpětné vazby od zákazníka. [33]

3.3 Standardy projektového řízení

V rámci zlepšení projektového řízení vzniklo několik standardů a příruček, jak projekty co nejúspěšněji řídit. Tyto standardy bychom mohli přirovnat k metodikám vývoje, zpravidla jsou však starší a obecnější. Mezi nejznámější a nejvíce používané standardy projektového řízení patří PMBOK Guide a metodika PRINCE2.

3.3.1 PMBOK

PMBOK, zkratka z Project Management Body of Knowledge, je standard projektového řízení od organizace PMI (Project Management Institute). Metodika je rozšířena převážně v USA a standardy projektového řízení popisuje v příručce PMBOK Guide. [34] První verze příručky byla vydána roku 1996 a v současné době existuje již šesté vydání. Kniha obsahuje podrobné definice procesů ve všech fázích projektu a návody, jak procesy provádět a řídit. Zdroje uvádí, že PMBOK Guide obsahuje celkově 132 nástrojů a technik, které lze odděleně aplikovat v rámci řízení projektů. [35] Organizace PMI, jenž je autorem této příručky, nabízí velkou řadu certifikací z oblasti projektového řízení.

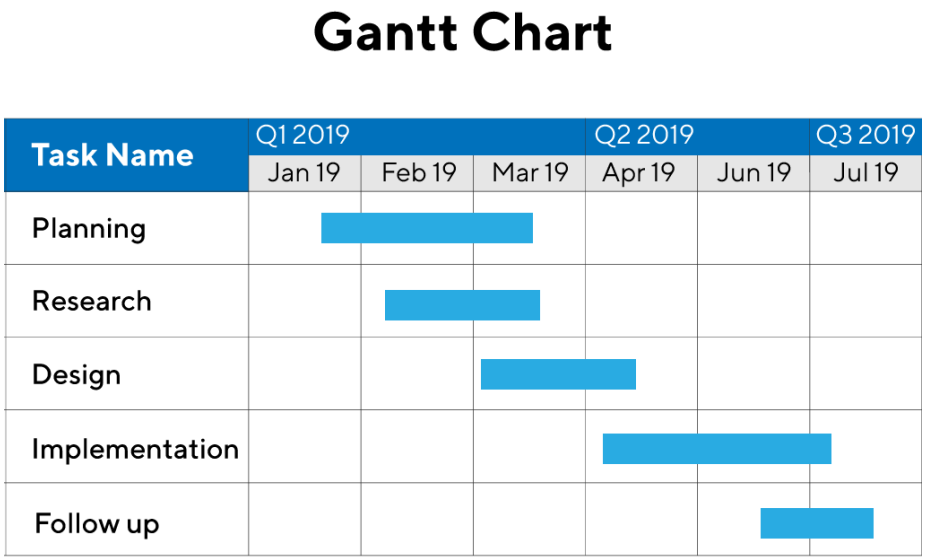
3.3.2 PRINCE2

Na rozdíl od standardu PMBOK, který je nejrozšířenější v USA, PRINCE2 je nejrozšířenější metodikou projektového řízení v Evropě. Rozvoj metodiky financovala vláda Velké Británie a veškeré postupy jsou popsány v příručce Managing Successful Projects with PRINCE2 s posledním vydáním roku 2017. [36] Základem metodiky byla analýza problémů, na kterých projekt selhává.

Z analýzy byly vytvořeny konkrétní postupy, jak projekty řídit. PRINCE2 klade větší důraz na kvalitní business case, než například PMBOK. Business case, česky projektový záměr, je dokument, který slouží k prezentaci projektu například investorům. Obsahuje náklady, výnosy, rizika a další aspekty projektu. Je to první krok, který se musí provést při zahajování nového projektu. Dále se metodika věnuje důkladnému popisu rolí a procesům v řízení projektu. [35] [37] Metodika PRINCE2 také nabízí velkou řadu certifikací pro projektové manažery.

3.3.3 Ganttův diagram

Ganttův diagram není standardem řízení projektu, ale využívá se jako nástroj pro zlepšení projektového řízení. Poprvé ho vytvořil Henry Gantt na počátku 20. století a v dnešní době je nezbytnou součástí každého plánování projektu. Přestože není pevně spjatý s vývojem softwaru, tak se s ním v tomto odvětví také často setkáme. Diagram graficky znázorňuje průběh projektu v čase. Osa Y grafu slouží k popisu dané činnosti a osa X představuje čas. Porovnáním odhadované doby trvání konkrétní činnosti s reálnou dobou trvání lze určit, jestli se na činnosti začalo pracovat v předstihu, nebo naopak později, než se plánovalo. [38] Nevýhodou je to, že diagram neznázorňuje, v jaké fázi rozpracování se jednotlivé činnosti projektu nachází, tudíž je vhodná kombinace s Burndown grafem (viz. kapitola Scrum). Následující schéma znázorňuje velmi zjednodušený příklad Ganttova diagramu. V praxi se každý úkol na ose Y dělí na mnoho menších úkolů a barevně se vyznačují kritické činnosti projektu.



Obrázek 8: Ganttův diagram [39]

3.4 Praktiky programování v agilních metodikách

V této kapitole jsou představeny různé praktiky v programování, které nejsou typické výhradně pro agilní metodiky, ale velmi často se zde vyskytují.

3.4.1 Kolektivní vlastnictví kódu

Jednou z praktik agilního programování je kolektivní vlastnictví kódu. Často se využívá například v Extrémním Programování a vychází z myšlenky, že žádný kus kódu nikomu nepatří, každý člen týmu má ke všemu přístup a může všechny části kódu přepisovat. Každý tak může kód vylepšit, zjednodušit, nebo pokračovat v práci vývojáře, který je například nemocný. Tým jako celek pracuje na stejném úkolu se stejným cílem, což má různé výhody. Tým se tak více stmelí, minimalizuje se počet chyb a členové si mohou předávat své znalosti a dovednosti. Ke správnému a plynulému chodu kolektivního vlastnictví kódu je zapotřebí přísný coding standard, aby všichni členové týmu kódu rozuměli. [40]

3.4.2 Coding standards

Coding standards jsou určitá pravidla psaní kódu tak, aby jednotliví vývojáři kódu rozuměli. Jedním z pravidel je například pojmenovávání tříd, metod a proměnných. Tým by se měl sjednotit v tom, zdali využívat tzv. pojmenování camel case, pascal case či snake case. Camel case vypadá takto: prvniPromenna, pascal case takto: PrvniPromenna a snake case takto: prvni_promenna. Další pravidla coding standards mohou definovat například způsob a frekvenci komentářů v kódu, dokumentaci či unit testování. [41]

3.4.3 Pair programming

Párové programování spočívá v psaní kódu dvou vývojářů u jednoho počítače. Jeden z vývojářů je tzv. driver, česky řidič, který kód píše. Druhý je navigátor, který na kód dohlíží a hledá chyby. Každých několik minut by se pak role měly vystřídat. Párové programování s sebou nese několik výhod. Zredukuje se tak počet chyb, vývojáři si předávají nové vědomosti, není nutné dělat code review a rychle se tímto způsobem zaškolí noví vývojáři. Nevýhoda párového programování může nastat, pokud vývojáři píší jednodušší kód, kde nevznikne takové množství chyb a dva lidé u jednoho počítače nejsou tak užiteční. V tomto případě by bylo nasazení dvou vývojářů k jednomu počítači nevýhodné jak z hlediska času, tak z hlediska peněz. [42]

3.4.4 Mob programming

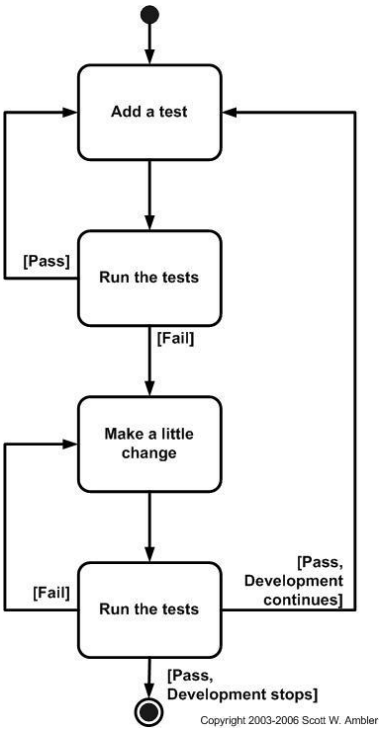
Mob programming je extremistická verze párového programování. Zakládá si na stejných principech s tím rozdílem, že u jednoho počítače nesedí dva lidé, ale celý vývojový tým včetně zákazníka nebo jeho zástupce. Mezi velké výhody patří velmi rychlá komunikace, protože se celý tým nachází v jedné místnosti. Dalšími výhodami je lepší schopnost rozhodování či minimalizace množství chyb. [43] Jak již bylo zmíněno v kapitole o párovém programování, mob programming může mít stejnou nevýhodu. Pokud tým správně nespolupracuje či úkol není příliš komplexní, může být nasazení celého týmu k jednomu počítači časově i finančně nevýhodné.

3.4.5 Refactoring

Podle knihy Refactoring od Martina Fowlera je refaktorování proces změny softwaru takovým způsobem, kdy se změní vnitřní struktura kódu bez změny jeho chování. Vylepšuje se tak design kódu s cílem minimalizovat počet chyb a bugů. [44] Refactoring je důležitou praktikou ve vývoji softwaru i u tradičních metodik, ty agilní však refaktorování kladou ještě větší důraz.
Zuzana Šochová v knize Agilní metody řízení projektů dokonce uvádí, že po každé změně v kódu by se ihned měly hledat prostory, kde kód refaktorovat. [12]

3.4.6 Test Driven Development

Test Driven Development, zkráceně TDD, česky programování řízené testy, je způsob vývoje pomocí pravidelných unit testů. V praxi programátor nejdříve napíše jednotkový test, který musí skončit neúspěchem, jelikož daná funkcionalita ještě není naprogramována. Poté se napíše kód funkcionality co nejjednodušeji tak, aby unit testem prošel. Pokud projde, provede se refactoring kódu a přejde se k další funkcionalitě znovu od začátku. Proces je znázorněn na následujícím vývojovém diagramu. [45]



Obrázek 9: Vývojový diagram Test Driven Development [45]

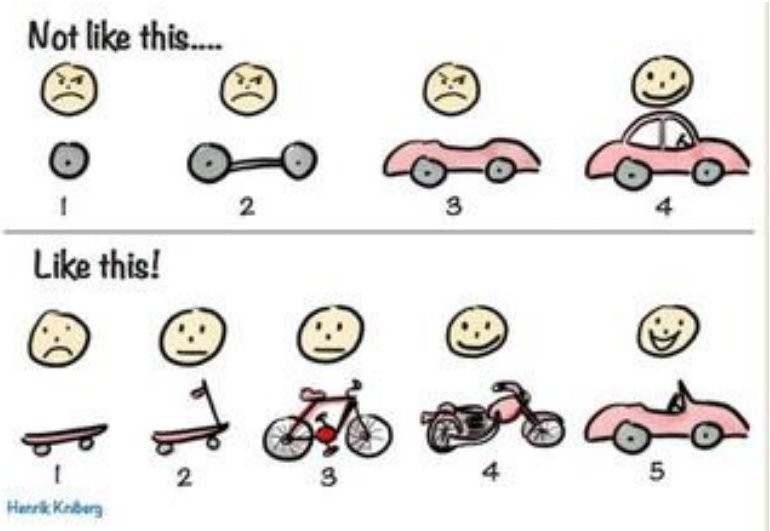
3.5 Rozdíly mezi metodikami

Z pohledu autora práce je hlavním rozdílem mezi tradičními a agilními metodikami vývoje softwaru iterativní a sekvenční vývoj. Najdeme výjimky, kdy tradiční metodiky využívají iterační cyklus, jako například Spirálový model, ale agilní metodiky nevyužívají sekvenční postup. Obecně tak platí, že tradiční metodiky využívají sekvence a agilní iterace.

S tímto rozdílem nastává další změna, kterou je schopnost či neschopnost reagovat na změny požadavků v průběhu vývoje. Při sekvenčním přístupu je za chodu téměř nemožné reagovat na změny přání zákazníka, tudíž je kladen velký důraz na prvotní plánování a sjednocení vize projektu obou zainteresovaných stran. S tím je spojena i obsáhlejší dokumentace požadavků. Jelikož je v současné době velmi obtížné neprovádět změny během vývoje, tak se tradiční metodiky se sekvenčním přístupem hodí buď na menší a méně komplexní projekty, nebo na projekty takového typu, kde je předem jasný cíl projektu. Příkladem vhodným k použití sekvenčního přístupu může být tvorba podnikového informačního systému pro firmu. Cíl vývoje takového softwaru může být jednoznačnější než vývoj jiných aplikací, a to pokrýt správu všech agend podniku, jako například mzdy, finance či výrobu. Samozřejmě i zde se přání jednotlivých firem budou lehce měnit a je potřeba se na konkrétním cíli shodnout před začátkem vývoje. Neznamená to, že by komunikace se zákazníkem během vývoje již neprobíhala, pouze není tak častá, jak je tomu u agilních metodik.

Naopak agilní metodiky s iterativním cyklem se vyznačují schopností reakce na změny v požadavcích zákazníka. Tato schopnost je dána pravidelnou a častou komunikací se zákazníkem na konci každého cyklu, například na konci sprintu ve Scrumu. Software se dodává v menších částech a vždy zohledňuje zákazníkovu zpětnou vazbu. Díky časté komunikaci jak uvnitř týmu, tak obou stran vzájemně, jsou agilní metodiky vhodnější pro tvorbu komplexnějších projektů s méně jasným cílem na začátku. Odpadá tak nutnost rozsáhlé dokumentace i sjednocení vize zákazníka s týmem za každou cenu hned na začátku projektu. Příkladem vhodným pro využití agilních metodik může být vývoj webových, desktopových či mobilních aplikací, kde nemusí být dopředu známy všechny funkcionality, které bude potřeba v projektu obsáhnout.

Rozdíly mezi tradičními a agilními metodikami, konkrétně rozdíl iterativního a sekvenčního přístupu, lze zobrazit například na následujícím schématu.



Obrázek 10: Rozdíl mezi tradičními a agilními metodikami [46]

Horní řada ukazuje postup vývoje pomocí sekvenčního přístupu, dolní řada pak pomocí iterativního cyklu. Schéma není vhodné z důvodu, že upozaďuje tradiční metodiky a pozdvihuje ty agilní, ale lze na něm dobře pozorovat rozdíl ve vývoji.

Sekvenčním přístupem se vyvíjí software, který až do samotného konce nelze využít. Iterativní přístup vytváří fungující software, na který se postupně nabalují další funkcionality. Na otázku, zdali jsou lepší agilní či tradiční metodiky, však nelze odpovědět, neboť se obě metodiky hodí v jiné situaci. Tradiční se vyplatí využít v malých projektech pro svou jednoduchost, kde by naopak agilní byly zbytečnou přítěží. Agilní využijeme v komplexních projektech, kde by tradiční svou jednoduchostí neobstály. Tradiční metodiky mají tu výhodu, že je dopředu pevně stanoven rozsah práce, doba trvání i náklady. Pokud vzniknou problémy na straně vývojáře, zákazníkovy náklady se tak podle smlouvy nezvýší. Na druhou stranu agilní metodiky žádný pevný řád stanovený nemají, tudíž se práce může protáhnout či prodražit.

4 Vlastní práce

4.1 Seznámení s podnikem

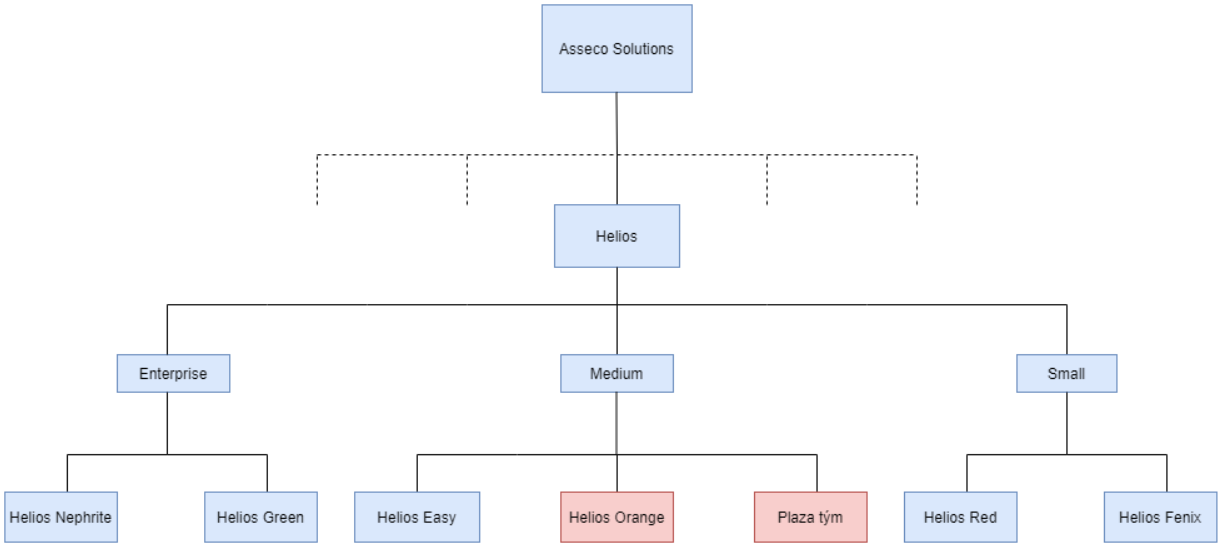
Autor vytvářel praktickou část práce ve spolupráci s firmou Helios, zabývající se vývojem podnikových informačních systémů pro různá odvětví. Podnik Helios spadá pod nadnárodní společnost Asseco Solutions působící převážně na českém a slovenském trhu, ale distribuce probíhá i do dalších států v Evropě.

Po diskuzi se šéfem vývoje jednoho z týmů lze vymezit interní rozdělení jednotlivých vývojových týmů a jejich produktů podle velikosti cílových firem, na které je software aplikován. Vznikají tak 3 segmenty, ve kterých Helios realizuje svou činnost. První, zároveň největší segment, nese název Enterprise. Do tohoto segmentu lze zařadit dva ze šesti vývojových týmů firmy Helios, a to konkrétně Helios Nephrite a Helios Green. Cílovou skupinou jsou velké firmy, pro které nabízí moduly například z oblasti řízení firmy, řízení výroby či evidence skladových zásob.

Druhý segment, nazývaný Medium, slouží středně velkým firmám. Do této kategorie spadají vývojové týmy Helios Easy, Helios Orange a Plaza tým. První dva zmíněné také vyvíjí podnikové informační systémy. Plaza tým se od ostatních liší tím, že se specializuje čistě na vývoj platforem pro webové aplikace. Helios Orange i Plaza tým jsou předmětem hlubšího zkoumání autora práce a dále jim bude věnovaná větší pozornost.

Do třetího a zároveň posledního segmentu Small lze zařadit vývojové týmy Helios Red a Helios Fenix. Helios Red nabízí zpracování všech agend podnikatelům a malým firmám. Helios Fenix se pak zaměřuje na spolupráci s organizacemi veřejné správy.

Následující schéma graficky znázorňuje hierarchii vývojových týmů ve firmě Helios. Schéma obsahuje rozdělení na tři segmenty, pod které spadají konkrétní vývojové týmy. Zvýrazněny jsou týmy, kterým je v dalších kapitolách věnována větší pozornost.



Obrázek 11: Hierarchie vývojových týmů ve firmě Helios [Vlastní zpracování]

4.2 Spolupráce s Helios Orange

Autor práce docházel osobně do místa působení firmy a zjišťoval, jakou metodiku vývoje tým využívá. Nejprve je nastíněn obecný popis týmu a praktiky, které realizuje. Na základě tohoto pozorování je provedena analýza využívané metodiky a je podán návrh řešení na zlepšení efektivity řízení vývoje. Sběr informací probíhal skrze pozorování a kladení dotazů.

4.2.1 Obecný popis

Vývojový tým Helios Orange vytváří podnikové informační systémy. Tým se skládá z analytiků, programátorů a produktového manažera. Programátoři pracují ve vývojovém prostředí Delphi, programují tedy v jazyce Object Pascal. Tým využívá praktiky stand-up meetingů, ale upravují si je podle svých potřeb. Schůzka členů týmu probíhá pouze jednou týdně ve čtyřech různých kategoriích se čtyřmi skupinami vývojářů. Systém je rozdělen do čtyř agend, a to následovně: mzdy, ekonomika, logistika a jádro. Každého stand-upu se účastní šéf vývoje vždy s vývojovým týmem zabývajícím se konkrétní agendou. Jednou týdně také probíhá meeting mobilního vývoje systému, kde řeší průběžné aktualizace.

Během autorova pozorování bylo plánování ztíženo šířením koronaviru, tudíž také vždy probíhal hovor přes aplikaci Teams pro členy týmu, kteří měli nařízenou domácí karanténu. Současně se na monitor promítá seznam požadavků v programu VIS – Vnitřní Informační Systém. Seznam svou strukturou připomíná Kanban tabuli. Každá položka má přiřazeného programátora a analytika, kteří na ní pracují, zobrazuje, ve kterém stádiu zpracování se položka nachází, o co se v ní jedná a jakou má přiřazenou prioritu.

Z hlediska priorit řeší vývojový tým čtyři kategorie. Jedná se o chyby v systému, přání zákazníka, strategické požadavky a legislativu. Ačkoliv velmi důležité jsou všechny čtyři kategorie, tak z hlediska prosperujícího chodu firmy má velkou prioritu legislativa a strategické požadavky ve smyslu posunutí firmy dále. Interní statistika uvádí, že ročně tým řeší okolo čtyř tisíc přání zákazníků a tři tisíce legislativních požadavků dohromady jak českých, tak slovenských. Strategický vývoj zabere 50 % času, legislativa, chyby a přání zákazníků zbylých 50 %. Ze všech zmíněných kategorií vzniká roční plán vývoje a později dochází ke kvartálnímu upřesnění. Dochází tak ke kontrole stavu vývoje, kde jednou z pomůcek je jak již zmiňovaná elektronická, tak i fyzická Kanban tabule. Fyzická tabule se používá při vývoji nové verze systému.

Kvartální upřesnění plánu vývoje se provádí z důvodu, že tým aplikuje kvartální release, ve kterém prezentuje novinky zákazníkovi. Následuje zpětná vazba zákazníka a diskuze o další vizi spolupráce.

4.2.2 Stand-upy

Autor bakalářské práce se zúčastnil postupně stand-upů tří ze čtyř agend, kterými se tým zaobírá.

Na stand-upu části týmu zabývajícím se logistikou se sešlo sedm lidí fyzicky – šéf vývoje, jeden produktový manažer, jeden analytik a tři programátoři. Virtuálně se připojil jeden analytik a jeden programátor kvůli nařízené karanténě. Šéf vývoje řídí diskuzi a postupně programátoři a analytici popisují, kam postoupili s prací, a odhadují, kdy budou mít úkol hotový. Rozdělují si úkoly v Kanban tabuli v programu VIS. Meeting trval okolo třiceti minut, což je i standardní doba trvání stand-upu tohoto týmu.

Dále probíhá stand-up ekonomického a potom mzdového týmu. Složení členů týmu a jejich funkce se lehce mění, jinak postup stand-upu probíhá podobně, jako u logistiky. Probíhá debata o aktuálních úkolech a rozdělují se úkoly nové, jen se samozřejmě mění náplň úkolů s ohledem na agendu, kterou se zabývají. Pouze schůzka mzdového týmu byla kratší, trvala okolo patnácti minut.

Následující tabulka shrnuje nejvýznamnější praktiky a informace získané o týmu. Zjištěné poznatky poslouží jak k analýze využívané metodiky týmu, tak k následnému návrhu řešení.

Helios Orange	
Zaměření vývoje	Podnikové informační systémy
Cyklus vývoje	Sekvenční
Četnost release	Kvartální
Nástroj pro organizaci práce	Vnitřní Informační Systém (VIS)
Četnost schůzek	1x týdně
Délka schůzek	Přibližně 30 minut

Tabulka 1: Shrnutí informací o Helios Orange [Vlastní zpracování]

4.3 Přiblížení metodiky vývoje Helios Orange

Jak je v praxi zvykem, tak ani k Helios Orange nelze přiřadit jednu konkrétní metodiku vývoje a prohlásit, že využívá pouze tuto metodiku bez žádných odchylek od standardů. Často dochází ke směsi více metodik vývoje a k poupravování jednotlivých pravidel.

Helios Orange konkrétně využívá vodopádový model s minoritními prvky agilního vývoje. Vhledem k zaměření podniku na vývoj podnikových informačních systémů, kde se cíle tolik neliší a zpětná vazba od zákazníka neprobíhá tak často, se jeví využívaný neagilní vodopádový model jako nejlepší řešení. Tým však do určité míry přebírá i myšlenky agilního vývoje, konkrétně Scrumu a Kanbanu.

4.3.1 Převzetí praktik Scrumu

Příkladem využití praktik Scrumu jsou stand-upy, které se svým průběhem téměř neliší. Diskutuje se o rozpracovaných úkolech, přidělují se úkoly nové a odhaduje se časová náročnost. Rozdíl je pouze v četnosti schůzek, kdy Scrum meetingy jsou každý den, ale schůzky pozorovaného týmu pouze jedenkrát týdně s každou agendou týmu. Za to jsou však schůzky delší. Netrvají patnáct minut, jak uvádí Scrum příručka s názvem The Scrum Guide, [13] ale každá trvá okolo třiceti minut.

Další částí vývoje, kde se tým lehce odchýlil od klasických metodik, je vývoj mobilní verze systémů. U mobilních zařízení jsou typické časté aktualizace a s tím spojená potřeba častější zpětné vazby od zákazníka. Zde by byla klasická metodika vývoje ve standardní formě jen těžko udržitelná, tudíž tým přistoupil k častějšímu odevzdávání práce zákazníkovi a k častější zpětné vazbě.

4.3.2 Převzetí praktik Kanbanu

Jedinou, za to velmi důležitou praktikou, kterou tým využívá, je Kanban tabule. Využívána je jak elektronická, tak fyzická verze. Elektronická tabule slouží k vizualizaci jednotlivých úkolů. Zobrazuje stavy úkolů a přiřazuje jim priority a programátory. Elektronická Kanban tabule je velmi důležitou praktikou ve všech stand-upech, které firma realizuje.

Fyzická tabule se v dnešní době již moc nevyužívá, výjimkou je pouze plánování vývoje nové verze systému.

4.3.3 Shrnutí

Z poznatků, které autor práce o vývojovém týmu Helios Orange zjistil, lze definovat využívané metodiky vývoje.

Základní a nejdůležitější praktikou je sekvenční přístup, kvartální release, a ne tak častá komunikace se zákazníkem, jak je tomu u agilního vývoje. Z povahy informačních systémů, kde se cíl vývoje téměř nemění, tudíž jsou výhodné větší prodlevy mezi jednotlivými release a zároveň není potřeba tak častá zpětná vazba, je nasazení vodopádového modelu ideální. Tým se však nedrží čistě klasického vývoje. Z agilního světa přebírá pár praktik, které považuje za výhodné, jako například použití Kanban tabule či pravidelné stand-upy. Vzniká tak dobře organizovaný a úspěšný tým na poli ERP systémů využívající klasické metodiky vývoje obohacené o ty nejvýhodnější praktiky agilního světa.

4.4 Spolupráce s Plaza týmem

Druhým týmem bližšího zkoumání je vývojový tým zabývající se výhradně vývojem platformy pro webové aplikace. V následujících kapitolách je nastíněn obecný popis týmu, analýza využívané metodiky a návrh řešení pro zefektivnění práce.

4.4.1 Obecný popis

Na rozdíl od prvního týmu, který byl nucen přistoupit ke komunikaci přes platformu Teams kvůli šíření koronaviru, tento tým takto funguje běžně. Je totiž trochu netypicky rozdělen na jeden tým s vývojáři v České republice a na Slovensku, a na druhý tým se sídlem v Indii. Tuzemský tým se zabývá vývojem core platformy, na které aplikace běží, tým v Indii pak vývojem samotné aplikace. To znamená, že vývojáři z Indie jsou závislí na core balíčcích zprostředkovávaných vývojáři platformy. Dochází tak ke vzájemné komunikaci a spolupráci. Předmětem pozorování autora práce byl vývoj platformy českými a slovenskými vývojáři.

Tým realizuje denní stand-upy, technické hovory a review s plánováním. Daily stand-upy probíhají každý den ráno, technické hovory pak obden a review s plánováním jedenkrát týdně. Autor se postupně zúčastnil denního meetingu a následně review s plánováním.

4.4.2 Daily stand-upy

Probíhají přes platformu Teams. Hovoru se účastnilo celkem 6 lidí, kteří postupně sdíleli obrazovky a probíhala diskuze, na čem pracovali minulý den, na čem budou pracovat dnes, jak dlouho úkol potrvá a jestli nenarazili na nějaké potíže. Meetingy jsou korigovány šéfem vývoje týmu. Celkově stand-up trval dvacet minut.

4.4.3 Review a plánování

Review se opět účastnilo 6 lidí skrze platformu Teams, kde šéf vývoje schůzku řídí. Z důvodu interních informací není sdělen konkrétní obsah review, obecně však šlo o postupný popis a prezentaci práce jednotlivých vývojářů na probíhajícím sprintu. Review probíhalo sedmdesát minut a neprodleně po skončení začalo plánování.

Plánování probíhalo formou diskuze, kde tým určuje, na čem bude kdo pracovat. Současně s diskuzí přesouvá moderátor schůzky položky z product backlogu do sprint backlogu pomocí softwaru Jira. Dále se konkrétní user stories přiřazují jednotlivým vývojářům, nastavuje se jejich priorita a stav rozpracování, ve kterém se user story nachází. Některé user stories se nepřejímají z product backlogu, ale vytváří se zcela nové. Celkově plánování trvalo 35 minut.

Přiložená tabulka zobrazuje souhrn informací o Plaza týmu, které slouží k identifikaci využívané metodiky vývoje a k následnému návrhu řešení.

Plaza tým	
Zaměření vývoje	Platformy pro webové aplikace
Cyklus vývoje	Iterační
Četnost release	1x za 2 týdny
Nástroj pro organizaci práce	Jira
Četnost schůzek stand-up	Každý den
Délka schůzek stand-up	Přibližně 20 minut
Četnost review a plánování	1x týdně
Délka review	Přibližně 70 minut
Délka plánování	Přibližně 35 minut
Forma komunikace	Online

Tabulka 2: Shrnutí informací o Plaza týmu [Vlastní zpracování]

4.5 Přiblížení metodiky vývoje Plaza týmu

S vývojem platforem pro webové aplikace, kterým se tento tým věnuje, přichází nutnost častějších aktualizací a častější zpětné vazby, než u vývoje podnikových informačních systémů. Také cíl celého projektu nemusí být ve fázi plánování tak jasně vymezen. Plaza tým tudíž musí pracovat agilněji, konkrétně má nejblíže k metodice Scrum. Jelikož je téměř nemožné, ba dokonce nevýhodné, dodržovat všechny standardy metodik vývoje, tak i u tohoto týmu lze pozorovat řadu odchylek od praktik Scrumu.

4.5.1 Odchýlení od Scrumu

Na první pohled nejvíce viditelnou změnou je fakt, že vývojový tým se schází pouze virtuálně. To je velká odchylka od praktik Scrumu, který klade velký důraz na časté osobní kontakty, příjemné pracovní prostředí a co nejlepší spolupráci v týmu. Tato změna je způsobena převážně tím, že část vývojářů žije na Slovensku, a také tím, že dochází k úzké spolupráci s vývojovým týmem v Indii.

Druhou změnou je nedělání retrospektivy, což je podle praktik Scrumu schůzka sloužící k poohlédnutí se za minulými sprinty a poučení se z vlastních chyb. Další meeting, o kterém se Scrum zmiňuje a tým si ho lehce poupravil, je backlog grooming. Tato schůzka probíhá většinou jednou na začátku a podruhé uprostřed sprintu. Slouží vývojářům k lepšímu pochopení jednotlivých user stories a celkové vize produktu.

Plaza tým tento meeting nepořádá pravidelně, jak doporučují nejlepší standardy metodiky Scrum, ale pouze tehdy, pracují-li na nějakém novém tématu.

Další změny již nejsou tak zásadní, ale lze se zmínit například o daily stand-upech, které trvají přibližně dvacet minut. V čistém Scrumu se mluví o délce okolo patnácti minut, tým má tudíž stand-upy o něco delší. Poslední změnou, která stojí za zmínku, je aplikování definition of done na user stories. Definition of done je soubor podmínek, které musí úlohy splňovat, aby se daly na konci sprintu prohlásit za hotové. Praktiky hovoří o nasazení souboru podmínek na všechny úlohy v neměnné podobě, časem lze podmínky zpřísňovat. Obecně se vyplatí definition of done zavádět tehdy, pracuje-li ve firmě analytik i vývojář. Plaza tým v praxi zavádí definition of done pouze na některé úlohy. Je to způsobeno převážně tím, že vývojáři rovnou sami analyzují, tudíž pro ně nemá tato praktika takový význam.

4.5.2 Převzetí praktik Kanbanu

Stejně, jako Helios Orange, i Plaza tým přebírá Kanban tabuli. Využívají ji k plánování sprint backlogu prostřednictvím softwaru Jira. Software umožňuje například přesouvat user stories z product backlogu do sprint backlogu, přiřadit user story vývojáři či zobrazit detailní popis epiců a super user stories.

4.5.3 Shrnutí

Z pozorování autora práce lze definovat používané metodiky Plaza týmu a jejich odchýlení se od standardů.

Povahou vývoje webových aplikací, kde jsou velmi důležité časté aktualizace a následná zpětná vazba, je ideální agilní pojetí vývoje. Pravděpodobně z důvodu atypického působení týmu výhradně ve virtuálním prostředí lze prohlásit, že tým využívá zjednodušený Scrum. Scrum proto, že využívá nejvíce praktik z této metodiky, jako například daily stand-upy či rozdělení vývoje na sprinty. Zjednodušený pak proto, že některé praktiky nerealizuje, nebo je realizuje pouze částečně. Příkladem nerealizované praktiky je retrospektiva, částečně realizované pak definition of done. Autor práce se domnívá, že ke zjednodušení dochází zejména kvůli relativně malému počtu osob ve vývojovém týmu. Čistý Scrum obecně počítá s větším počtem zaměstnanců a nutnost nasazení všech praktik a jejich následné kladné výsledky vzrůstají s počtem vykonávaných user stories a počtem zaměstnanců. Ve větším týmu tak bez striktních kontrol a retrospektiv může docházet ke zmatkům, v menších naopak nejsou nutností a mohou zhoršovat efektivitu práce.

Tým na druhou stranu přebírá i praktiky jiných metodik, konkrétně například Kanban tabuli. Vzniká tak navzdory pomyslné online bariéře dobře spolupracující a úspěšný tým, který zvyšuje svou kvalitu osekáváním pro ně méně potřebných praktik Scrumu a přebíráním praktik potřebných, například z Kanbanu.

4.6 Návrh řešení

V této kapitole jsou autorem práce navrženy možné změny ve vývoji softwaru, které by mohly týmům přinést zvýšení efektivnosti vývoje. Konkrétně se jedná o návrhy na zlepšení komunikace v týmu a zefektivnění práce. Návrh změn je utvořen na základě studování teorie o metodikách přístupu k řízení vývoje a praktickém pozorování a sběru informací v konkrétních týmech.

4.6.1 Helios Orange

Helios Orange využívá vodopádový přístup vývoje s prvky Scrumu. První změnou, která by podle autora práce stála za vyzkoušení v praxi, je upravení četnosti a délky pravidelných meetingů týmu. Tým je rozdělen do čtyř agend. V současné době probíhá schůze s každou agendou jedenkrát týdně s délkou trvání okolo třiceti minut. Obsahem je rozvržení práce jednotlivých členů na daný týden. Efektivitu těchto meetingů by mohl zvýšit nárůst počtu týdenních schůzek a zároveň zkrácení jejich doby trvání. Konkrétně autor práce navrhuje místo jedné třiceti minutové schůze v pondělí dvě patnácti minutové schůze rovnoměrně rozložené během týdne. Scrum doporučuje takto krátké schůze, protože po delší době mohou členové týmu ztrácet pozornost. O problému ztrácení pozornosti svědčí i doporučení Scrumu, že by účastníci meetingu měli stát, aby se schůze zbytečně neprodlužovala a všichni se soustředili na plynulou konverzaci. Z doporučení konání meetingu ve stoje je také odvozen název daily standup. Helios Orange sice necílí na dodržování praktik Scrumu, ale rozložení a zkrácení schůzek by mohlo efektivitu komunikace zvýšit.

Nástroj na organizaci práce

Druhé doporučení se týká programu určeného pro vizualizaci a organizování práce. K tomuto účelu tým využívá Vnitřní Informační Systém, zkráceně VIS. Na první pohled lze poznamenat, že VIS nedodržuje současné trendy designu a práce s ním může trvat delší dobu, než u moderních konkurentů. S programem na organizaci práce tým pracuje na zmiňovaných schůzkách, tudíž rychlejší operace v programu by rovněž mohla efektivitu meetingů zvýšit. Společně s přechozím doporučením týkajícím se zkrácení schůzek, by kombinace s dynamickým a rychlým programem mohla přinést značné rozdíly v efektivnosti. Mezi moderní konkurenty Vnitřního Informačního Systému patří například software Jira, se kterým spolupracuje druhý pozorovaný tým Plaza. Autor doporučuje zhodnocení, zdali by přechod na jiný program nenavýšil produktivitu vývoje. Otázkou však zůstává, jestli užitek plynoucí ze změny softwaru bude větší, než spojené náklady a učení se s novým programem.

Následující tabulka shrnuje aktuálně využívané praktiky v týmu Helios Orange a navrhované změny, které by mohly zlepšit efektivitu vývoje.

Helios Orange	Aktuální praktika	Navrhovaná změna
Četnost schůzek	1x týdně	2x týdně
Délka schůzek	Přibližně 30 minut	Přibližně 15 minut
Nástroj pro organizaci práce	Vnitřní Informační Systém (VIS)	Například software Jira

Tabulka 3: Návrh řešení pro Helios Orange [Vlastní zpracování]

4.6.2 Plaza tým

Plaza tým vyvíjí platformy pro webové aplikace v lehce nestandardním režimu, a to v režimu zcela online prostřednictvím home office. První otázka, která člověka napadne pro tým pracující v takovém režimu je, zdali by vývoj nebyl efektivnější, pokud by tým pracoval osobně na jednom místě. Scrum klade velký důraz na samostatně organizovaný tým a co nejlepší formu komunikace. Komunikace i efektivita práce by se při osobním kontaktu skoro jistě o něco zlepšila, nevýhodou jsou ale opět výdaje za pronájem kancelářských prostor a změna zaběhlých zvyklostí členů týmu. Autor práce navrhuje zhodnotit klady a zápory pronájmu prostor za účelem osobního setkávání.

Retrospektiva a Backlog grooming

Další návrh, který stojí za zhodnocení týmem, je realizace určitých typů schůzek, které tým momentálně nerealizuje. Konkrétně se jedná o retrospektivu a backlog grooming. Retrospektiva je schůzka, která slouží k poohlédnutí se za předchozími sprinty a poučení se z vlastních chyb. Backlog grooming je naopak meeting na začátku sprintu, který slouží ke správnému pochopení všech funkcionalit developery. Plaza tým backlog grooming provádí, ale pouze nepravidelně. Scrum obě schůzky doporučuje provádět pravidelně, ale již nezohledňuje velikost týmu. Plaza tým nemá příliš mnoho pracovníků, tudíž funkcionalit není tolik a tím pádem se snadněji pochopí. Proto možná realizace zmíněných schůzek nemusí být potřeba, ale zhodnocení jejich zavedení ve vývoji určitě stojí za úvahu, zdali by neměly kladný vliv na efektivitu práce.

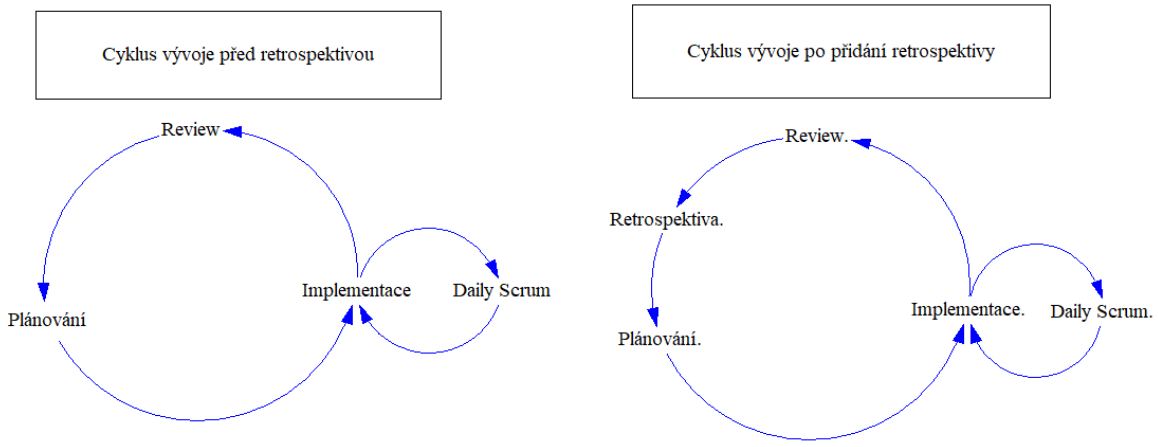
O prokazatelném zlepšení řízení vývoje využíváním pravidelné retrospektivy hovoří například kniha Getting value out of agile retrospective. [47] Autoři uvádí, že pokud vývojový tým dodržuje pravidelné retrospektivy, na kterých řeší prostory pro zlepšení v předchozí iteraci, tak bude docházet k nepřetržitému zlepšování vývoje a k nárůstu business value. O kladném vlivu retrospektivy mluví také kniha Agile Procecess in Software Engineering and Extreme Programming. [48] V knize je zveřejněn výsledek aplikování agilních praktik na pět vývojových týmů v praxi.

Agile Practice	understandability	knowledge transfer	communication	transparency	feedback	satisfaction	team empowerment	focusing	structuredness	planning	time-to-market	risk management	# adressed benefits	# using projects
Daily Stand-Up			3	5						2			3	4
Sprint				2				1	2	1		1	5	5
Sprint Review	1		2	1						1			4	4
Retrospective		1			1	1							3	4
Sprint Planning						1				4		1	3	5
Backlog	1			2					1	1			4	5
Burndown-Charts				3									1	4
Scrum Master						1	1						2	4
Product Owner				1			1			1			3	4
80%-Rule									1	2		2	3	2
Planning Poker							1			2			2	1
User Stories								1					1	4
Epics										3			1	3
Cont. Integration				2							1		2	3
Scrum-of-Scrums			4	1					1	1			4	3
# practices	2	1	3	8	1	3	3	2	4	10	1	3		

Obrázek 12: Výsledek aplikování agilních metodik v praxi [48]

Sloupec vpravo znázorňuje, kolik z testovaných pěti týmů zaznamenalo užitek realizováním retrospektivy. Čtyři z pěti týmů hlásily kladné změny v projektovém řízení, proto je z tohoto důvodu Plaza týmu realizace retrospektivy doporučena.

Následující schéma vizualizuje změny v cyklu vývoje, kdyby se Plaza tým rozhodl retrospektivu realizovat. První by byla fáze plánování a poté by následovala implementace s tolikrát se opakujícím Daily Scrumem, kolik dní trvá jeden sprint. Poté by přišlo na řadu review, zmiňovaná retrospektiva a plánování dalšího sprintu.



Obrázek 13: Změna cyklu vývoje přidáním retrospektivy [Vlastní zpracování]

Definition of done

Poslední navržená změna se týká definition of done. Jedná se o seznam podmínek, které funkcionalita musí splňovat, aby se mohla prohlásit za hotovou. V best practices Scrumu by se definition of done mělo aplikovat na všechny funkcionality bez výjimky. V praxi Plaza tým využívá tento seznam podmínek pouze na některé funkcionality, což by mohlo vyvolat zmatek, jaká funkcionalita je již hotová a na které se ještě pracuje. I zde platí, že každá mince má dvě strany, tudíž je otázkou, jestli tým vytváří tolik funkcionalit, aby se vyplatilo definition of done zavádět. Při malém počtu funkcionalit to může být zbytečná praktika, ale autor doporučuje zavedení praktiky zvážit.

Přiložená tabulka znázorňuje momentálně realizované praktiky Plaza týmu společně s navrhovanou změnou, která by mohla přinést kladný dopad na řízení vývoje.

Plaza tým	Aktuální praktika	Navrhovaná změna
Forma komunikace	Online	Osobní setkávání
Retrospektiva	Nerealizováno	Realizovat
Backlog grooming	Prováděn nepravidelně	Provádět pravidelně
Definition of done	Zaváděno částečně	Zavádět v plném rozsahu

Tabulka 4: Návrh řešení pro Plaza tým [Vlastní zpracování]

4.7 Rozhovor s Helios Orange

V rámci praktické části dále proběhl rozhovor se šéfem vývoje týmu Helios Orange. Dotazy autora práce směřovaly převážně k navrhovaným změnám pro tým, k vizi do budoucna a k dopadům koronavirové krize na vývoj softwaru. Konkrétní dotazy a parafrázované odpovědi zněly následovně:

Plánujete do budoucna nějaké změny ve vývoji?

Tým plánuje změny v architektuře, konkrétně by se chtěl více zaměřit na aplikační servery, kde by tým poskytoval určité mikro služby pro zákazníka.

Plánujete změny ve využívání home office na základě nynějších zkušeností?

Helios Orange si zakládá na osobních vztazích, tudíž neplánuje extremistickou variantu přechodu zcela na home office po vzoru Plaza týmu. Na druhou stranu se práce z domova poměrně osvědčila, tudíž by v budoucnu k vyšší míře využívání home office mohlo docházet. Konkrétně není vyloučena možnost pravidelných prací z domova jedenkrát až třikrát v týdnu.

Ovlivnila koronavirová krize efektivitu vývoje či zájem o produkty?

Efektivitu vývoje krize skoro neovlivnila. Co se týče zájmu o produkty, tak se celkově krize počítačových a informačních technologií nedotkla, nebo pouze minimálně. Konkrétně tým paradoxně zaznamenal jeden z nejúspěšnějších roků.

Ovlivnila vývoj významná změna legislativy v lednu 2021?

V lednu 2021 došlo ke změně legislativy, kdy se změnil vzorec výpočtu čisté mzdy zaměstnanců. Dále se přestaly evidovat dovolené zaměstnanců ve dnech, ale přistoupilo se k evidování v řádu hodin. Dotazovaný podnik zpracovává mimo jiné mzdové agendy firmám, tudíž změna legislativy ovlivnila režim vývoje týmu. Konkrétně byl tým nucen k dočasnému agilnějšímu pojetí vývoje, v praxi docházelo k častějším schůzkám developerů a k častější komunikaci se zákazníkem.

Jaký je Váš názor na návrh řešení ohledně zvýšení počtu schůzek a zkrácení doby trvání?

Je to jedno z možných řešení, ke kterému by tým v budoucnu mohl přistoupit. Kdyby se tým rozhodl jít více po vzoru Scrumu, tak nejsou vyloučeny ani schůzky každý den. V současné době si však šéf vývoje nemyslí, že by zkrácení schůzek a větší četnost vedlo k lepšímu vývoji.

Jaký je Váš názor na návrh změny ohledně nástroje pro organizaci práce?

Šéf vývoje hodnotí stávající verzi využívaného Vnitřního Informačního Systému jako nedostačující a tým plánuje určité změny. Společně s autorem práce se shodl, že vizualizace v nástroji není optimální. Konkrétně by chtěl změnit organizaci, kdy by každý vývojář viděl svou hotovou a nastávající práci v rámci konkrétního cyklu vývoje. V současné době jsou práce jednotlivých vývojářů uspořádány nepřehledně na jednom místě. Do budoucna tým tedy plánuje určitou nástavbu nad nynější Vnitřní Informační Systém, která by současné nedostatky eliminovala.

5 Výsledky a diskuse

Pozorované výsledky v realizovaných praktikách týmů se lehce lišily od autorova očekávání. Záměrně byla vybrána spolupráce s jedním týmem vyvíjejícím tradičním vodopádovým modelem a jedním týmem vyvíjejícím pomocí agilní metodiky Scrum. Tento výběr proběhl proto, aby bylo možné nastínit rozdíly mezi oběma způsoby projektového řízení. Výsledky se však lišily od očekávání, protože Plaza tým vyvíjí software trochu atypicky pouze z prostředí home office bez osobního kontaktu. Na základě tohoto faktu se jedna z navrhovaných změn týkala zhodnocení, zdali by osobní setkávání nezvýšilo efektivitu práce. Komunikace s Plaza týmem výhradně online s sebou přinesla také neočekávanou formu sběru informací, a to pomocí videohovorů.

Návrh řešení pro tým Helios Orange do značné míry ovlivnilo šíření koronaviru a s ním spojená vyšší míra online komunikace. Tento tým se standardně setkává na osobních schůzkách v kancelářích, kvůli nařízeným karanténám a home office však byl nucen využívat videohovory mnohem častěji. Z tohoto důvodu byl v návrhu řešení kladen výrazně větší důraz na zlepšení komunikace. Proto byl podán návrh na zvážení změny softwaru pro organizaci a vizualizaci práce. Zastaralejší nástroj pro organizaci práce se ve standardním režimu vykompenzuje lepší komunikací na osobních schůzkách, v době online meetingů se však významnost dobrého softwaru zvyšuje. Na základě toho byl týmu doporučen program Jira, který využívá již zmiňovaný Plaza tým.

Celkově je návrh řešení složen ze subjektivního pozorování autora práce a nemůže zaručit obecné stoprocentní úspěchy. S téměř každou navrhovanou změnou přichází zvýšení buď jednorázových, nebo dlouhodobých nákladů. Lehce extremistickým návrhem je zařízení kancelářských prostor pro Plaza tým, neboť zde by náklady vzrostly razantně. Naopak doporučení, které by náklady vůbec nezvýšilo, je modifikace četnosti schůzek pro Helios Orange. Navrhovaná změna spočívá ve dvou kratších schůzkách během týdne místo jedné dlouhé. Takové opatření by pouze narušilo zaběhlé zvyklosti členů týmu.

Otázkou u všech navržených změn zůstává, zdali je užitek plynoucí ze změny větší, než vynaložené náklady a narušení zvyklostí. Zhodnocení přínosu a vyčíslení nákladů by mohlo být dalším předmětem zkoumání, pokud by se vývojové týmy rozhodly řídit se návrhem řešení. K dosažení analýzy dílčích návrhů by mohlo vést testování návrhů v praxi, odhad kalkulace nákladů a porovnání s podniky, které podobnou změnu ve vývoji také učinily. Při rozhodnutí realizovat danou změnu v projektovém řízení by následovalo zkoumání změny efektivnosti vývoje a porovnání s dobou, kdy opatření ještě nebylo zavedeno. Na základě tohoto zkoumání by pak došlo k rozhodnutí, jestli má daná modifikace kladný dopad, nebo jestli se tým vrátí k předchozím praktikám.

6 Závěr

Hlavním cílem bakalářské práce byla analýza využívaných metodik vývoje v konkrétních developerských týmech a následný návrh řešení pro zefektivnění práce. K analýzám a návrhu řešení došlo na základě studování teorie agilních i tradičních způsobů řízení vývoje.

V teoretické části práce byly přiblíženy různé způsoby vývoje softwaru. Byly zmíněny nejznámější a nejvíce využívané tradiční metodiky, jmenovitě Vodopádový model, Spirálový model a model „Napiš a oprav.“ U každého způsobu vývoje byly popsány nejzákladnější pravidla, praktiky a vhodnost využití v praxi. Poté byly přiblíženy metodiky agilní, jejich principy vývoje a konkrétní zástupci. Zmíněn byl například Scrum, Extrémní programování, Kanban, metodika Crystal, Feature Driven Development a další. Pozornost byla dále věnována nejzásadnějším rozdílům mezi tradičními a agilními metodikami, jejich výhody a nevýhody využívání v praxi. Poté byly nastíněny standardy projektového řízení PMBOK a PRINCE2, čím se každý standard vyznačuje a kde se nejvíce využívá. Práce se dále zabývala některými praktikami v programování, které se často využívají v agilním vývoji softwaru. Konkrétně bylo popsáno například kolektivní vlastnictví kódu, párové programování, mob programming či refaktorování. V poslední řadě byl zmíněn Ganttův diagram jakožto nástroj pro řízení projektů a Burndown chart pro zhodnocení efektivnosti Scrum metodiky.

Praktická část bakalářské práce byla věnována analýze metodik vývoje ve dvou developerských týmech. Nejdříve byly popsány obecné informace o týmech a konkrétní praktiky, které týmy k vývoji softwaru využívají. Na základě využívaných praktik následovala analýza, které metodice vývoje se praktiky nejvíce přibližují. Jak v praxi bývá zvykem, tak se ani jeden z pozorovaných týmů nedržel pouze jedné metodiky, jejíž praktiky by splňoval do posledního doporučení. Jeden z týmů využívá z velké části tradiční metodiku Vodopádového modelu s prvky Scrumu a Kanbanu. Druhý tým řídí vývoj s využitím převážně Scrumu a rovněž s některými prvky Kanbanu. Zkoumány byly rozdíly oproti best practices Scrumu, podle kterých druhý tým některé praktiky nerealizuje nebo si je nějakým způsobem modifikuje. Změny jsou v praktické části zaznamenány a popsány včetně možného důvodu, proč k nerealizaci nebo modifikaci může docházet. Na základě těchto změn oproti standardům je následně doporučen návrh řešení pro oba týmy, který by mohl přinést kladné změny ve vývoji, konkrétně by mohl zvýšit efektivitu práce, komunikace či dynamiku pravidelných meetingů. V rámci praktické části je také uskutečněn rozhovor se šéfem vývoje jednoho z týmů, kde je mimo jiné nastíněn názor a reakce na navržené změny ve vývoji.

Jedna z navržených změn prvnímu týmu byla v četnosti a délce pravidelných schůzek, druhý návrh se týkal zhodnocení změny softwaru využívaného pro organizaci a vizualizaci práce. Druhému týmu byla navržena realizace některých praktik Scrumu, které tým buď nerealizuje, nebo realizuje pouze částečně. Konkrétně se jednalo o navržení schůzek retrospektivy a backlog grooming. Dále byla navržena změna ve využívání definition of done, a to na všechny funkcionality bez výjimky. Poslední, trochu extremistickou změnou, bylo doporučení zvážit, zdali by přechod z online komunikace na osobní setkávání nezlepšil efektivitu práce a komunikace. Tato změna by vyžadovala velké náklady spojené s pronájmem kancelářských prostor.

Vhodnost a možný užitek z této změny, společně se všemi ostatními navrženými opatřeními, je více do hloubky probrán v kapitole Diskuse. Téměř veškerá doporučení s sebou přináší nutnost zvýšení nákladů. Pokud by se týmy rozhodly doporučeními řídit, musela by dále následovat detailní analýza kalkulace nákladů a odhad přínosu pro vývoj softwaru. V diskuzi je také více vysvětleno, proč se autor práce ke konkrétním návrhům uchýlil a jaký užitek by z toho mohl pro vývojové týmy vzniknout.

V teoretické části byly detailně nastíněny témata týkající se vývoje aplikací. V praktické části došlo k úspěšné analýze využívaných metodik, byl podán návrh řešení a bylo vysvětleno, proč k navrženým změnám došlo. Splněním těchto bodů byl rovněž splněn hlavní cíl celé práce.

7 Seznam použitých zdrojů

[1] Vodopádový model (Waterfall model). *Management mania* [online]. [cit. 2021-01-19]. Dostupné z: https://managementmania.com/cs/vodopadovy-model-waterfall-model

[2] Vodopádový model. *Testování softwaru* [online]. [cit. 2021-01-19]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/vodopadovy-model/

[3] Waterfall Model. *Project Manager* [online]. [cit. 2021-02-19]. Dostupné z: https://www.projectmanager.com/waterfall-methodology

[4] MALKUSOVÁ, Tereza. Je lepší vyvíjet vodopádem nebo scrumem? *Aitom* [online]. 23.05.2016 [cit. 2021-01-20]. Dostupné z: https://www.aitom.cz/co-je-noveho/je-lepsi-vyvijet-vodopadem-nebo-scrumem

[5] Software Process Models: Sashimi. *Thomas A. Alspaug* [online]. [cit. 2021-01-19]. Dostupné z: https://www.thomasalspaugh.org/pub/fnd/softwareProcess.html#Sashimi

[6] Spirálový model. *Testování softwaru* [online]. [cit. 2021-01-20]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model/

[7] PAL, SAYAN KUMAR. Software Engineering | Spiral Model. *GeeksForGeeks* [online]. 28 May, 2018 [cit. 2021-01-20]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-spiral-model/

[8] THAKUR, Dinesh. What is build and fix model or ad hoc model? and Explaini its Advantages and Disadvantages. *Computer notes* [online]. [cit. 2021-01-20]. Dostupné z: https://ecomputernotes.com/software-engineering/build-and-fix-model

[9] SDLC Models(Software Development Life Cycle Models) : Introduction. *Minigranth* [online]. [cit. 2021-01-20]. Dostupné z: https://minigranth.in/software-engineering-tutorial/sdlc-models

[10] History: The Agile Manifesto. *Agile manifesto* [online]. 2001 [cit. 2021-01-20]. Dostupné z: https://agilemanifesto.org/history.html

[11] Manifest Agilního vývoje software. *Agile manifesto* [online]. 2001 [cit. 2021-01-20]. Dostupné z: https://agilemanifesto.org/iso/cs/manifesto.html

[12] ŠOCHOVÁ, Zuzana a Eduard KUNCE. *Agilní metody řízení projektů*. 2. vydání. Brno: Computer press, 2019, 200 s. ISBN 978-80-251-4961-4. EAN: 9788025149614.

[13] SCHWABER, Ken a Jeff SUTHERLAND. *The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game* [online]. In: . November 2020, s. 14 [cit. 2021-01-21]. Dostupné z: https://www.scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100

[14] ŠOCHOVÁ, Zuzana. Kdo je to Scrum Master? A kdo Product Owner?: SCRUM MASTER ROLE. *Zuzi's blog* [online]. 11. 02. 2011 [cit. 2021-01-21]. Dostupné z: https://soch.cz/blog/management/agile/scrum-management/kdo-je-to-scrum-master-a-kdo-product-owner/

[15] ŠOCHOVÁ, Zuzana. Backlog Grooming. *Zuzi's blog* [online]. 18. 12. 2014 [cit. 2021-01-21]. Dostupné z: https://soch.cz/blog/management/agile/scrum-management/backlog-grooming/

[16] Sprint Planning. *Agile Alliance* [online]. [cit. 2021-01-21]. Dostupné z: https://www.agilealliance.org/glossary/sprint-planning/

[17] 5 Best Practices for Running a Daily Standup Meeting. *Storm board* [online]. February 11, 2019 [cit. 2021-01-21]. Dostupné z: https://stormboard.com/blog-archive/best-practices-running-daily-standup-meeting

[18] What is Sprint Retrospective Meeting in Scrum? *Visual Paradigm* [online]. [cit. 2021-01-21]. Dostupné z: https://www.visual-paradigm.com/scrum/what-is-sprint-retrospective-meeting/

[19] ŠOCHOVÁ, Zuzana. Definition of Done. *Zuzi's blog* [online]. 22. 03. 2018 [cit. 2021-01-22]. Dostupné z: https://soch.cz/blog/management/agile/definition-of-done/#more-836

[20] What is Burndown Chart in Scrum? *Visual Paradigm* [online]. [cit. 2021-02-15]. Dostupné z: https://www.visual-paradigm.com/scrum/scrum-burndown-chart/

[21] BECK, Kent a Cynthia ANDRES. *Extreme Programming Explained*. 2. vydání. USA: Addison-Wesley Professional, 2004. ISBN 978-0321278654.

[22] Extrémní programování. *Cs wiki* [online]. 11.11.2020 [cit. 2021-01-22]. Dostupné z: https://www.cswiki.cz/wiki/Extr%C3%A9mn%C3%AD_programov%C3%A1n%C3%AD

[23] GROSSMANN, Lukáš. Extrémní programování. *IT network* [online]. [cit. 2021-01-22]. Dostupné z: https://www.itnetwork.cz/navrh/metodiky/extremni-programovani

[24] BREWER, John. Introduction to Extreme Programming. *Jera* [online]. [cit. 2021-01-22]. Dostupné z: http://www.jera.com/techinfo/xphandout.pdf

[25] Extreme Programming. *Science Jrank* [online]. [cit. 2021-01-22]. Dostupné z: https://science.jrank.org/programming/Extreme_Programming.html

[26] A History of Kanban. *Kanban tool* [online]. [cit. 2021-01-22]. Dostupné z: https://kanbantool.com/kanban-guide/kanban-history

[27] The Kanban method in IT development projects. *Bocasay* [online]. July 29, 2020 [cit. 2021-01-22]. Dostupné z: https://www.bocasay.com/kanban-method-it-development-projects/

[28] SINGH, Virender. Crystal Method in Agile. *Tools QA* [online]. October 1, 2019 [cit. 2021-02-20]. Dostupné z: https://www.toolsqa.com/agile/crystal-method/

[29] KOTRLA, Tomáš. *Agilní metodiky vývoje software* [online]. Brno, 2005 [cit. 2021-02-20]. Dostupné z: https://is.muni.cz/th/y0eky/. Diplomová práce. Masarykova univerzita v Brně, Fakulta informatiky. Vedoucí práce Barbora Bühnová.

[30] COAD, Peter, Jeff DE LUCA a Eric LEFEBVRE. *Java Modeling In Color With UML: Enterprise Components and Process*. Prentice Hall, 1999, 221 s. ISBN 978-0130115102.

[31] LYNN, Rachaelle. What is FDD in Agile? *Planview* [online]. [cit. 2021-02-03]. Dostupné z: https://www.planview.com/resources/articles/fdd-agile/

[32] BUCHALCEVOVÁ, Alena. *METODIKA FEATURE-DRIVEN DEVELOPMENT NEOPOUŠTÍ MODELOVÁNÍ A PROCESY, A PŘESTO PŘINÁŠÍ VÝHODY AGILNÍHO VÝVOJE* [online]. Praha [cit. 2021-02-03]. Dostupné z: https://nb.vse.cz/~buchalc/clanky/tsw2005.pdf. VŠE Praha, Katedra informačních technologií.

[33] RYTÍŘ, Vladimír. *Rational Unified Process jako metodika vývoje softwaru* [online]. Brno, 2008 [cit. 2021-02-15]. Dostupné z: http://hdl.handle.net/11012/53201. Diplomová práce. Vysoké učení technické v Brně. Fakulta informačních technologií. Ústav informačních systémů. Vedoucí práce Jitka Kreslíková.

[34] PROJECT MANAGEMENT INSTITUTE. *A guide to the project management body of knowledge (PMBOK Guide)*. 6. vydání. Newtown Square: Project Management Institute, [2017]. ISBN 978-162-8251-845.

[35] GAŠPAROVIČ, Ivan. PRINCE2 vs. PMBOK. Jaké postupy jsou lepší pro řízení projektů? *Tayllorcox* [online]. 10.12.2020 [cit. 2021-02-16]. Dostupné z: https://www.tx.cz/blog/prince2-metodika-axelos-vs-pmbok-od-pmi

[36] BENNETT, Nigel. *Managing Successful Projects with Prince*. 6. vydání. Anglie: TSO, 2017. ISBN 9780113315338.

[37] Projektový záměr (Business Case). *Management mania* [online]. 05.06.2018 [cit. 2021-02-17]. Dostupné z: https://managementmania.com/cs/projektovy-zamer-business-case

[38] What is a Gantt Chart? *Project Manager* [online]. [cit. 2021-02-24]. Dostupné z: https://www.projectmanager.com/gantt-chart

[39] Gantt Chart. *Product Plan* [online]. [cit. 2021-02-15]. Dostupné z: https://www.productplan.com/glossary/gantt-chart/

[40] W., Bjorn. EXTREME PROGRAMMING PRACTICE: Collective Code Ownership. *Explain Agile* [online]. July 28, 2018 [cit. 2021-02-04]. Dostupné z: https://explainagile.com/agile/xp-extreme-programming/practices/collective-code-ownership/

[41] SENGAYIRE, Prince. CODING STANDARDS AND CONVENTIONS IN SOFTWARE DEVELOPMENT TEAM. *Medium* [online]. Nov 25, 2019 [cit. 2021-02-10]. Dostupné z: https://medium.com/@psengayire/the-importance-of-coding-standards-and-conventions-in-the-software-development-team-how-they-can-5d252556a05

[42] ALTVATER, ALEXANDRA. What is Pair Programming? Advantages, Challenges, Tutorials & More. *Stackify* [online]. JUNE 22, 2017 [cit. 2021-02-10]. Dostupné z: https://stackify.com/pair-programming-advantages/

[43] Mob Programming. *Agile Alliance* [online]. [cit. 2021-02-10]. Dostupné z: https://www.agilealliance.org/glossary/mob-programming

[44] FOWLER, Martin. *Refactoring*. Pearson Education, 2019, 448 s. ISBN 0134757599.

[45] Introduction to Test Driven Development (TDD). *Agile data* [online]. [cit. 2021-02-14]. Dostupné z: http://agiledata.org/essays/tdd.html

[46] KNIBERG, Henrik. The Myth of Incremental Development. In: *Herding Cats* [online]. JULY 16, 2014 [cit. 2021-02-17]. Dostupné z: https://herdingcats.typepad.com/my_weblog/2014/07/the-myth-of-incremental-development.html

[47] GONÇALVES, Luis a Ben LINDERS. *Getting Value out of Agile Retrospectives: A Toolbox of Retrospective Exercises*. lulu.com, 2014. ISBN 1304789624.

[48] KRUCHTEN, Philippe a Steven FRASER, COALLIER, François, ed. *Agile Processes in Software Engineering and Extreme Programming*. Cham: Springer, 2019. ISBN 978-3-319-57632-9.