Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Metodiky vývoje softwaru

Bc. Marek Javůrek

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Marek Javůrek

Informatika

Název práce

Metodiky vývoje softwaru

Název anglicky

Methodologies for Software Development

Cíle práce

Cílem diplomové práce je zavedení efektivního způsobu vývoje softwarových produktů v konkrétní firmě. Dílčím cílem je definovat metodu výběru podpůrného nástroje pro vývoj agilních softwarových projektů. Pomocí této metody bude vybrán vhodný nástroj pro konkrétní projekt ve firmě.

Metodika

Teoretická část diplomové práce bude založena na studiu odborných informačních zdrojů. V praktické části bude popsáno, jak probíhá zavedení konkrétní agilní metodiky pro vývoj softwarového produktu v praxi a jaké problémy či výhody jsou s konkrétní metodikou spojené. Na konkrétním projektu bude simulován průběh několika plánovacích a vývojových iterací s případným navrhnutím změn, které povedou k zlepšení tak, aby maximálně splňovaly cíle projektu, požadavky zadavatelské firmy a povedou k úspěšnému ukončení projektu. Následně bude provedeno porovnání technických a ekonomických aspektů dostupných nástrojů pro podporu agilního vývoje softwaru a nalezeno vhodné řešení pro konkrétní projekt.

Doporučený rozsah práce

50 – 60 stran

Klíčová slova

agilní vývoj, software, projekt, řízení projektů IT

Doporučené zdroje informací

BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů : kategorizace, agilní metodiky, vzory pro návrh metodiky.* Praha: Grada, 2005. ISBN 80-247-1075-7.

DOHNAL, J. *Řízení vztahů se zákazníky : procesy, pracovníci, technologie.* Praha: Grada, 2002. ISBN 80-247-0401-3.

KOMZÁK, Tomáš. Řízení IT projektů pro úplné začátečníky. 1. vyd. Brno: Computer Press, 2013, 213 s. ISBN 978-80-251-3791-8.

LEHTINEN, J. *Aktivní CRM : řízení vztahů se zákazníky.* Praha: Grada, 2007. ISBN 978-80-247-1814-9.

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

POCOVÁ, Ludmila. T-Kit – Řízení projektů. Praha: Česká národní agentura Mládež, Národní institut dětí a mládeže, 2007. ISBN 9788086784533.

SVOZILOVÁ, A. *Projektový management.* Praha: Grada, 2011. ISBN 978-80-247-3611-2.

ŠOCHOVÁ, Zuzana a Eduard KUNCE. Agilní metody řízení projektů. 1. vyd. Brno: Computer Press, 2014, 175 s. ISBN 978-80-251-4194-6.

Předběžný termín obhajoby

2016/17 LS – PEF

Vedoucí práce

Ing. Marek Pícka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 1. 11. 2016

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 25. 03. 2017

Čestné prohlášení

Prohlašuji, že svou diplomovou práci "Metodiky vývoje softwaru" jsem vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 20. 3. 2017 _____

Poděkování

Rád bych touto cestou poděkoval panu Ing. Markovi Píckovi, Ph.D. za cenné rady, věcné připomínky a vstřícnost při konzultacích v průběhu zpracování celé diplomové práce.

Metodiky vývoje softwaru

Souhrn

V současné době jsou agilní metodiky trendem v oblasti vývoje software. Nelze ovšem obecně říci, že jsou agilní metodiky lepší než tradiční (rigorózní) metodiky. Vždy je třeba analyzovat prostředí konkrétního podniku, projektu a okolí, aby bylo možno vybrat vhodnou metodiku. Tato práce se zabývá volbou vhodné metodiky a následně i implementací vybrané metodiky v prostředí konkrétní firmy. Dále se věnuje dostupným softwarovým nástrojům určeným pro podporu vývoje softwarových projektů a stanovením metody pro výběr vhodného nástroje s ohledem na požadavky podniku a projektu.

Klíčová slova: agilní vývoj, agilní metodiky, vývoj software, Extrémní programování, Kanban, Scrum, Feature Driven Development

Methodologies for Software Development

Summary

Agile Methodologies represent a new trend on a software development field.

Compared to rigid methodologies, we cannot determine whether agile methodologies is the best way for a software development. It is necessary to consider the specific company settings and project targets to choose the most suitable methodology using the tools

This thesis describes the main features of both the rigid and agile methodologies as well as the special software tools designated for a software project development.

In the next section the author describes his own experiences from both the optimal methodology selection process and the implementation process applying the specific methodology in the software development project in an active company with whom he worked. He compares several project administration tools as well as his contribution to the selection process of the best suitable SW for that company where he set up and applied his own multi-criterion analysis.

Keywords: Agile software development, Agile Methodologies, Software development, Extreme programming, Kanban, Scrum, Feature Driven Development

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Oblast softwarového inženýrství prošla v posledních letech značným vývojem. Kromě tradičních metodik vývoje softwaru se začaly často diskutovat a v praxi využívat metodiky agilní, které umožňují pružně reagovat na změny a výsledný produkt dodávají po částech a průběžně. Obecně ovšem není možné konstatovat, že jsou agilní metodiky lepší než tradiční metodiky nebo že se podle obecně zažitého názoru nehodí pro korporace a velké projekty. Vždy záleží na konkrétním projektu, prostředí a u agilních metodik zejména na komunikaci mezi zákazníkem a vývojovým týmem.

Diplomová práce navazuje na Bakalářskou práci autora na téma "Systém pro řízení projektů a podpory ve firmě", ve které byl proces vývoje software popsán pouze okrajově. Tato práce se zabývá metodikami vývoje softwaru, volbou optimální metodiky pro konkrétní projekt a také volbou softwarového nástroje pro podporu metodiky. Volba správné metodiky je klíčová a má přímý dopad na kvalitu výsledného softwarového produktu. V teoretické části jsou detailně popsány vybrané agilní metodiky a tradiční metodiky a postupy. Každá z metodik je zhodnocena a jsou popsány její silné a slabé stránky. Největší prostor je věnován agilním metodikám, zejména metodice Scrum, jejíž zavedení v konkrétním podniku je popsáno v praktické části této práce. V ní jsou rovněž popsány konkrétní problémové situace, se kterými se vývojový tým musel potýkat při implementaci metodiky, a také jejich řešení.

Pro podporu administrace procesů při řízení vývoje softwarových projektů existují různé podpůrné nástroje. Zejména softwarových podpůrných nástrojů jsou na trhu stovky, a proto může být jeho volba značně komplikovaná a vyžaduje určitý stupeň kvalifikace v oboru. Každý nástroj nabízí jiné funkce, má jiné vlastnosti a je třeba ho vybírat s ohledem na konkrétní záměry a cíle konkrétního podniku. Nevhodný výběr podpůrného nástroje a jeho následná změna po zavedení sebou nese značné náklady. Jedním z hlavních cílů této práce je stanovení metody, pomocí které je možné vybrat co nejvhodnější nástroj pro konkrétní projekt.

2 Cíl práce a metodika

Hlavním cílem práce je určit vhodnou metodiku pro vývoj softwarového produktu a její zavedení v konkrétním podniku. Spolu s výběrem metodiky bude dále provedena analýza priorit daného podniku, na základě které bude vybrán softwarový nástroj, který bude danou metodiku podporovat.

Dílčím cílem je tedy i analýza dostupných softwarových nástrojů pro podporu metodik vývoje softwaru, jejich filtrace dle zvolených kritérií a základě stanovení konkrétních priorit podniku bude vybrán podpůrný software pro administraci činností a úkolů po zavedení nové metodiky.

Všechny cíle diplomové práce souhrnně popisuje Tabulka 1.

Tabulka 1: Cíle diplomové práce (Zdroj: Autor)

ID cíle	Popis cíle	Metoda dosažení cíle
1	Analýza a popis zavedení vhodné metodiky pro vývoje software.	Pozorování současného stavu vývoje SW v podniku. Analýza problémů a využití dostupných metod pro volbu vhodné metodiky vývoje softwaru.
2	Provedení analýzy trhu a vzájemné porovnání dostupných nástrojů pro podporu metodik vývoje softwaru.	Analýza trhu a identifikace dostupných nástrojů. Stanovení filtračních kritérií pro užší výběr nástrojů. Stanovení filtračních kritérií pro vzájemné porovnání nástrojů. Porovnání nástrojů s interpretací výsledků.
3	Identifikace a popis výhod a nevýhod spojených s jednotlivými metodikami vývoje software a popis procesu výběru vhodné metodiky.	Analýza dostupných zdrojů o uvedené problematice a jejich diskuze.

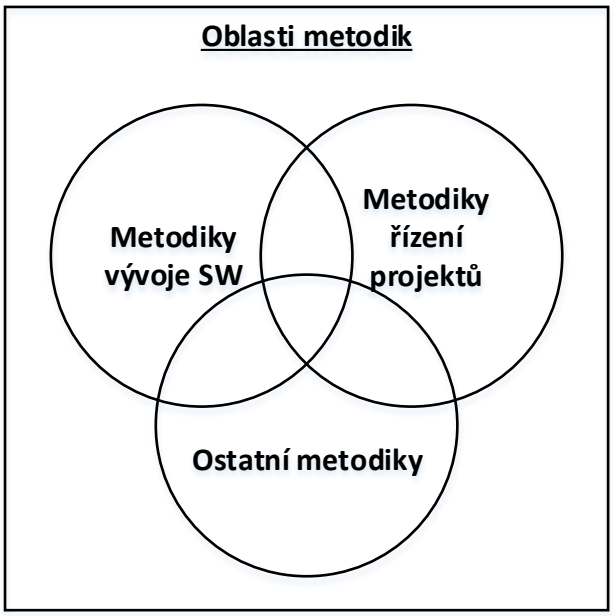
Tato diplomová práce je kombinací teoretického a empirického typu práce.

3 Teoretická východiska

3.1 Principy vývoje SW projektů

Abychom plně pochopili metodiky vývoje SW projektů, je nutné představit principy projektového řízení. Jelikož se tyto oblasti v určitých částech protínají a doplňují, tak jejich správnou kombinací zvyšujeme šanci na úspěšné dokončení SW, a tím pádem i úspěšný konec projektu. To vše za jedním společným cílem, dodat klientovi SW fungující podle jeho představ, v určité kvalitě, ve stanovených termínech a za daného rozpočtu (s určitými omezenými zdroji). [1]

Není možné zavést jednotné dělení metodik dle jejich účelu, protože každá metodika pokrývá jinou část ze životního cyklu projektu. Některé metodiky se snaží pokrýt celý proces od předprojektové fáze až po následný servis a podporu projektu po jeho ukončení. Jiné metodiky se naopak zaměřují pouze na část samotného vývoje SW. Na níže uvedeném obrázku č. 1 je uvedeno rozdělení metodik do tří základních oblastí. Jedná se pouze o zjednodušený pohled na celou problematiku a rozhodně ho nelze brát jako jediné možné rozdělení metodik. [16][2]



Obrázek 1: Průnik oblastí metodik (Autor dle [2]**)**

Níže jsou uvedeny konkrétní zástupci metodik z každé výše definované oblasti:

- Metodiky vývoje SW
    - Rigorózní metodiky

        - Vodopádový model,
        - Spirálový model,
        - Unified Software Development Process (zkráceně Unified Process),
        - Rational Unified Process,
        - Multidimensional Management and Development of Information Systems (MMDIS)
    - Agilní Metodiky
        - Scrum,
        - Kanban,
        - Extrémní programování (Extreme Programming, XP),
        - Feature Driven Development (FDD),
        - Agilní modelování (Agile Modeling)
- Metodiky řízení projektů
    - PRINCE2,
    - PMBOK,
    - PMI
- Ostatní metodiky
    - ITIL,
    - COBIT

Tato práce je zaměřena pouze na oblast metodik vývoje SW. V rámci této oblasti jsou popsáni vybraní zástupci zejména agilních metodik a vybraných tradičních metodik. Samostatná kapitola je věnována výběru vhodné metodiky pro konkrétní projekt.

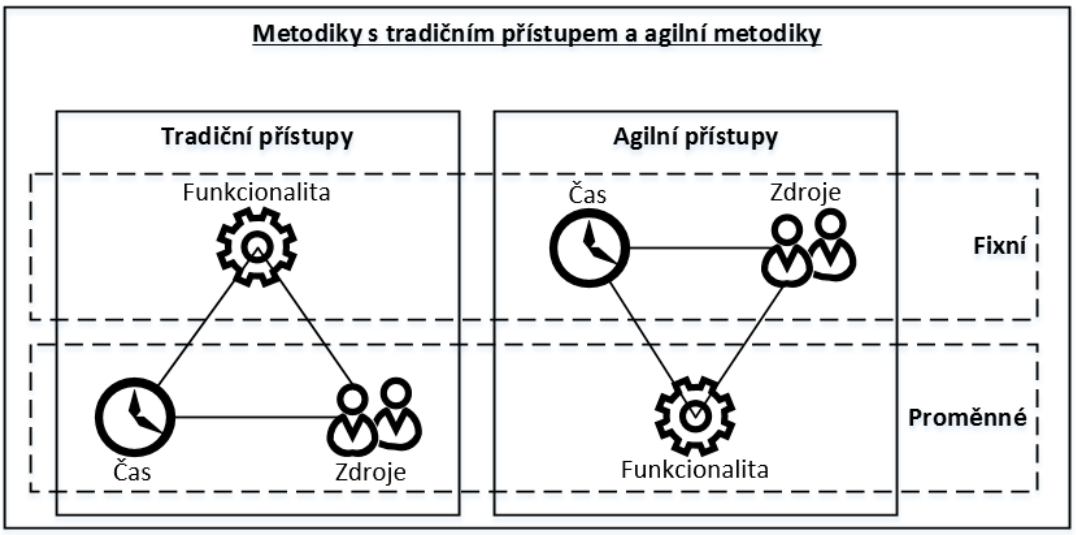
3.2 Projekt a projektový trojimperativ

Projektový trojimperativ neboli trojúhelník projektového řízení definuje tři roviny, ve kterých se při realizaci projektu pohybujeme. Jedná se o:

- Čas (Termíny)
- Zdroje (Náklady)
- Funkcionalita (Cíl)

Pro úspěšnou realizaci projektu se tedy musíme pohybovat v těchto třech rovinách, které nám vymezují prostor. Rozdíl mezi tradičním přístupem a agilním přístupem je

znázorněn na následujícím obrázku, který srovnává význam základních proměnných při vývoji SW [5]:

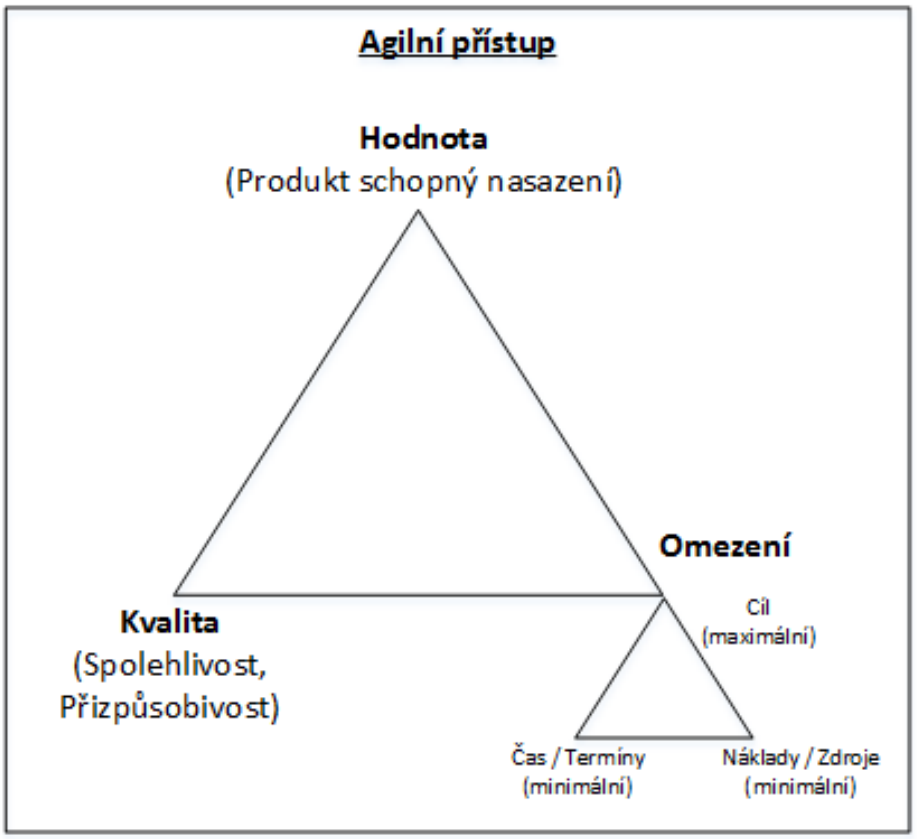


Obrázek 2: Metodiky s tradičním přístupem a agilní metodiky (Zdroj: Autor dle [3]**)**

3.2.1 Agilní trojimperativ

Jiný pohled na agilní projektový trojimperativ ukazuje Jim Highsmith [3]. Uvádí trojúhelník, který ukazuje skutečné cíle projektu.

1. Vytváření hodnoty, kterou zákazník očekává
2. Budování kvality, která zrychlí proces vývoje a vytvoří života schopnou platformu pro budoucí rozvoj
3. Dodržení určitých omezení (cíl, zdroje a termíny)



Obrázek 3: Agilní trojimperativ (Zdroj: Autor dle [3]**)**

Pro správné a úplné pochopení uvedeného agilního projektového trojimperativu je nutné si jeho jednotlivé části rozebrat blíže.

Hodnota

Studie ukazují, že více než 50% funkčnosti SW je využito zcela výjimečně nebo vůbec. Přestože jsou určité části této málo využívané funkcionality nutné (například finanční uzávěrka na konci roku), je zde stále velké procento funkcionality, která je zcela nevyužívána. Z toho vyplývá, že dokončený rozsah (funkcionalita) není zdaleka tak relevantní kritérium pro kontrolu stavu projektu. Namísto funkcionality bychom měli používat hodnotu, mimo to bychom se měli spíše ptát “Můžeme náš produkt již nasadit?” namísto “Implementovali jsme všechny požadavky?” Pro příklad uvádí projekty, které byly nasazeny s 20% až 30% funkcionalitou oproti očekávanému rozsahu a zákazník byl spokojen, protože jeho základní požadavky byly splněny.

Kvalita

Agilní trojúhelník vyzdvihuje roli kvality, na níž se můžeme dívat ve dvou horizontech.

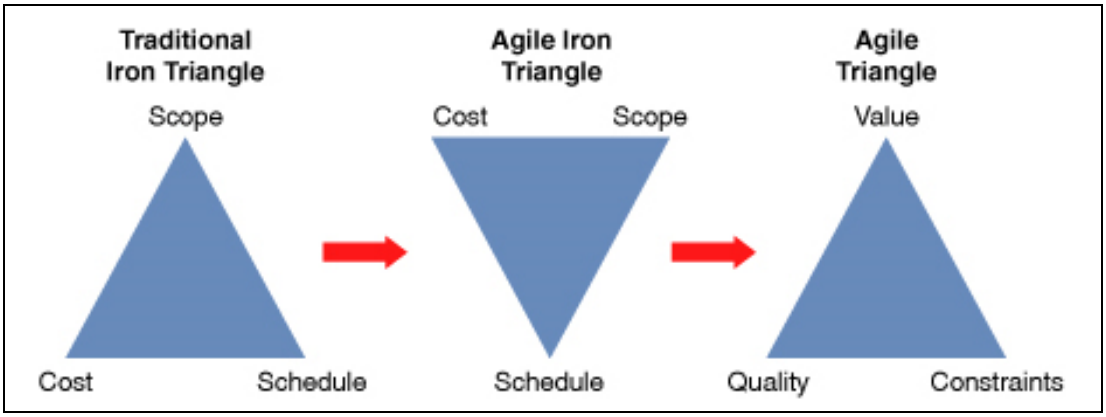
1. **Současná kvalita** ukazuje na kvalitu současné iterace vydání produktu. Měří spolehlivost produktu. Pokud produkt funguje spolehlivě, pak přináší zákazníkovi požadovanou hodnotu ve formě implementovaných požadavků. V opačném případě, pokud je produkt nespolehlivý nebo opakovaně selhává, tak není schopný přinášet požadovanou hodnotu.
2. **Budoucí kvalita** odpovídá na otázku "Bude produkt přinášet požadovanou hodnotu v budoucnosti?". Schopnost dodávat produkt v budoucnosti testuje aplikaci z hlediska schopnosti reagovat na plánované i neplánované změny. Často můžeme použít flexibilní návrh aplikace pro očekávané změny, ale hlavní strategie pro vyrovnání se s neočekávanými změnami je udržování technického dluhu na nízké úrovni.

Omezení

Poslední částí agilního projektového trojúhelníku jsou omezení, tedy cíl, zdroje a termíny. Není tomu tak, že by tyto části byly nedůležité, ale jednoduše to nejsou cíle projektu. Omezení jsou kritická pro dodání, udávají jasné hranice, ve kterých musí tým operovat. Nicméně pouze jedna z uvedených částí (cíl, zdroje a termíny) může být tou klíčovou a na agilních projektech to bývají termíny.

Agilní trojimperativ nám tedy nabízí zcela odlišný způsob pohledu na úspěch. Je možno jej interpretovat jako klíč k pochopení paradoxu přizpůsobeni naproti dodržení plánu. [3][5]

Níže uvedený obrázek tedy souhrnně zobrazuje evoluci projektového trojimperativu.



Obrázek 4: Evoluce agilního trojimperativu (Zdroj: [3]**)**

3.3 Míra úspěšnosti IT projektů

Míru úspěšnosti IT projektů dobře charakterizuje Chaos Report [4], který je publikován každoročně od roku 1994 a přináší globální pohled na stav úspěšnosti IT projektů. Studie z roku 2015 publikuje souhrnný pohled vycházející z porovnání výsledků více než 50 tisíc projektů z celého světa, které byly realizovány v letech 2011 až 2015.

Pro účely studie byly definovány dvě základní měřítka. Tato základní měřítka potřebujeme pro pochopení níže uvedené tabulky číslo 1, která je součástí studie.

Stav projektu:

1. Úspěšný projekt
    a. Projekt byl dokončen za předem stanovený čas, ve stanoveném rozpočtu a se všemi vlastnostmi a funkcemi s jakými byl naplánován.
2. Projekt dokončen s výhradami
    a. Projekt byl dokončen, ale přesáhl rozpočet nebo překročil stanovený časový rámec nebo byl dodán s méně funkcemi, než bylo naplánováno
3. Neúspěšný projekt
    a. Projekt byl zastaven v určité fázi vývojového cyklu

Velikosti projektu:

1. Malý projekt
    a. Rozpočet do 1,5 mil. USD (38 mil. Kč)
2. Středně velký projekt
    a. Rozpočet od 1,5 mil. USD (38 mil. Kč) do 6 mil. USD (152 mil. Kč)
3. Velký projekt
    a. Rozpočet od 6 mil. USD (152 mil. Kč) a výše

Nyní jsme definovali základní měřítka, která potřebujeme pro pochopení níže uvedené tabulky, která je součástí studie.

Tabulka 2: Úspěšnost IT projektů s vazbou na zvolenou oblast metodik (Zdroj: Autor dle [4]**)**

Velikost projektu	Metodika	Úspěšný projekt	Projekt dokončen s výhradami	Neúspěšný projekt
Bez rozlišení velikosti projektu	**Agilní**	39%	52%	9%
	Rigorózní	11%	60%	29%
Velké projekty	**Agilní**	18%	59%	23%
	Rigorózní	3%	55%	42%
Středně velké projekty	**Agilní**	27%	62%	11%
	Rigorózní	7%	68%	25%
Malé projekty	**Agilní**	58%	38%	4%
	Rigorózní	44%	45%	11%

Z výsledků studie je zřejmé, že agilní metodiky jsou obecně úspěšnější než rigorózní metodiky. Pokud se na výsledek podíváme z jiného úhlu pohledu, můžeme také konstatovat, že je lepší realizovat malé projekty. Pokud není možné rozsah projektu zmenšit a práce například rozdělit na více menších projektů, je opět dle studie vhodné zvolit agilní metodiky i pro velké projekty. Tyto výsledky jsou konzistentní i s historickými daty, nejedná se tedy pouze o neočekávaný výkyv v roce 2015. [6][4]

Dále jsou dostupné průzkumy State of Agile Report 2015 [7] a 2015 State of Scrum Report [8], které pojednávají zejména o použitých podpůrných nástrojích, o zkušenostech s agilními metodikami a o používaných praktikách agilních metodik. [7][8]

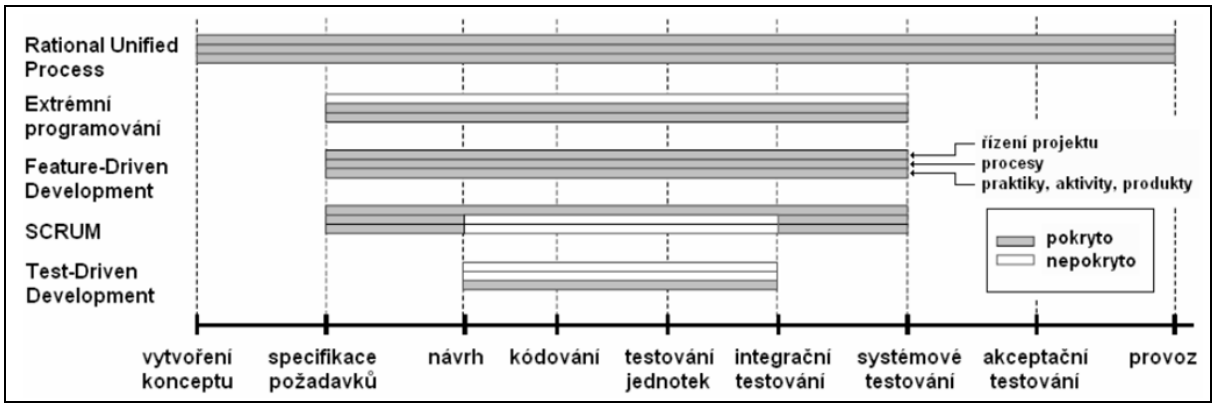
3.4 Úvod do metodik

Každá metodika se zabývá určitou množinou aspektů procesu vývoje software a představuje souhrn doporučených praktik a postupů. Některé metodiky se věnují problematice vývoje SW jako celku, ale většina zástupců metodik pokrývá vždy jen určitou část procesu vývoje SW. Jednotlivé metodiky se tedy liší svým rozsahem, použitými technikami a také formální náročností.

V současné době existuje nepřeberné množství metodik zabývajících se problematikou vývoje SW. Každá z metodik je více či méně odlišná od druhé a je tedy vhodná pro rozdílné typy projektů. Některé metodiky se naopak do značné míry překrývají. Důvod je jednoduchý, každý projekt je jedinečný a specifický, má určitá omezení a používá jinou technologii. Metodika je ve své podstatě zachycení určitého stavu, jak v

nějakém okamžiku na konkrétním projektu dané procesy a postupy fungovaly úspěšně. To ovšem neznamená, že po zavedení dané metodiky na vybraný projekt se stejnými rysy bude vše fungovat. Metodiku musíme obvykle přizpůsobit našim potřebám, tedy upravit pro potřeby našeho konkrétního projektu. Často je potřeba z metodiky určité její části vynechat a naopak si z jiné metodiky jiné její části vypůjčit. To je ovšem proces, který není možné implementovat ihned, proto se snažíme vybrat vhodnou metodiku, implementovat ji a postupem času ji na základě monitorování průběhu vývojového cyklu upravovat pro dané prostředí. Jednoduše řečeno, žádná metodika nepředstavuje jednoduchý a univerzální návod, jak postupovat při vývoji SW. Vždy záleží na tom, jak dobře dokážeme danou metodiku v daném prostředí implementovat.

Na níže uvedeném obrázku můžeme vidět pokrytí životního cyklu vývoje SW u vybraných metodik.



Obrázek 5: Srovnání metodik z pohledu pokrytí životního cyklu SW (Zdroj: [9]**)**

Také historie metodik vývoje SW je relativně nové odvětví, na rozdíl například od stavebnictví, kdy první architektonické myšlenky se objevily ve 3. tisíciletí př. n. l. v sumerské architektuře. Již v této době lidé využívali základních početních úkonů a systematicky si předávali zkušenosti. Naproti tomu první metodika pro vývoj informačních systémů byla sestavena až v roce 1960 pod jménem SDLC (Systems Development Life Cycle). [9]

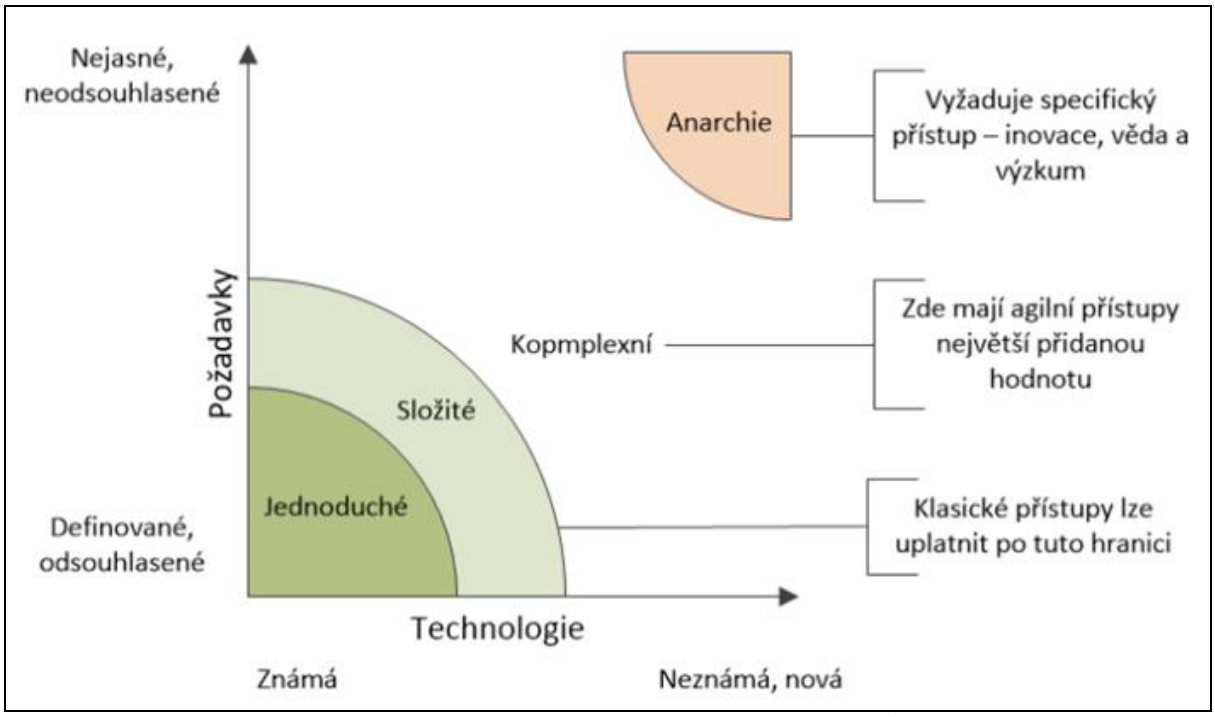
3.4.1 Volba vhodné metodiky

Volba vhodné metodiky má důležitou roli při vývoji SW. Při jejím výběru musíme myslet na to, že každý projekt, zákazník, vývojový tým a technologie jsou specifické, ale také že metodika nám sama o sobě úspěch nikdy nezaručí. Proto je důležité pamatovat na tyto základní premisy:

1. pro každý projekt se hodí jiný přístup (tedy jiná metodika);
2. každá metodika vyžaduje určité úpravy pro konkrétní implementaci

Metodika zavádí určitá pravidla a řád do procesu vývoje SW a předchází tím případným nejasnostem a nedorozuměním. Určuje jasnou roli každého člena týmu, každý ví, co a kdy má dělat. S tím souvisejí i předem dané kompetence a odpovědnosti. Použití vhodné metodiky obecně snižuji riziko neúspěchu. [10]

Na níže uvedeném obrázku je dle E. Macháčkové znázorněn graf volby typu metodiky dle zkušenosti s technologií (vodorovná osa) a stavu požadavků (svislá osa).



Obrázek 6: Volba typu metodiky dle technologie a stavu požadavků (Zdroj: [10]**)**

Jak je tedy zřejmé, tradiční metodiky není vhodné použít pro vývoj SW projektů, u kterých se předpokládá během jejich vývoje velké množství změn, případně v situacích, kdy zákazník neví nebo ani s pomocí dodavatele nedokáže jasně definovat svoje potřeby. Rovněž není vhodné použití tradiční metodiky u projektů, kde bude použita nová technologie, se kterou nemá dodavatelský vývojový tým zatím žádné zkušenosti. To by mohlo vést k velice zdlouhavému a složitému procesu s nejasným výsledkem.

Přehled klíčových hledisek a jejich předností a slabin je dle A. Buchalcevové [11] následující:

Tabulka 3: Přehled klíčových hledisek a jejich předností a slabin (Zdroj: [11]**)**

	M e t o d i k a	
	Rigorózní metodiky	**Agilní metodiky**
Náplň metodiky	Zaměřují se na explicitní znalost a pohlížející na lidi jako na sekundární faktor.	Zaměřují se na „tacit" znalosti, chápou lidi jako klíčové faktory úspěchu.
Podrobnost metodiky	Procesy a činnosti jsou popsány velmi podrobně.	Definována tzv. sotva dostatečná metodika, která se zaměřuje na činnosti, které vytvářejí hodnotu, a eliminuje činnosti, které hodnotu nepřinášejí.
Kvalita	Zaměření na kvalitu procesů a předpoklad, že kvalitní procesy povedou ke kvalitnímu výsledku.	Zaměření na hodnotu pro zákazníka a vysokou kvalitu produktu.
Předvídatelnost	Předpokládá předvídatelnost budoucnosti, důraz na anticipaci (sběr požadavků předem, plánování předem).	Předpokládá nepředvídatelnost budoucnosti, důraz na adaptaci na změny (přírůstkové shromažďování požadavků, plánování pro iteraci).
Změny	Změny podléhají řízení změn a je snaha změny minimalizovat.	Snaha změny umožnit a využít je, umožňují zákazníkům přehodnotit své požadavky s ohledem na nové znalosti.
Definovatelnost procesu vývoje software	Vývoj software je definovaný proces, je možné jej bez problémů opakovat.	Vývoj software je empirický proces, nemůže být konzistentně opakován, ale vyžaduje konstantní monitorování a adaptaci.
Hodnota pro zákazníka	Předpoklad, že dobré procesy vedou k dobrým výsledkům, je příliš zaměřen na vlastní procesy, ne na výsledky pro zákazníka.	Nejvyšší prioritou je uspokojovat zákazníka.
Participace zákazníka na projektu	Jen v počátečních a koncových fázích, po podpisu dokumentu specifikace požadavků řízení přebírá tým technologických pracovníků.	Přesun nositele řízení z týmu na zákazníka, zákazník je řídícím subjektem během celého projektu, při každé iteraci zákazník může měnit priority funkcí.
Rozsah řešení	Vývojáři se snaží do systému zabudovat všechny funkce, které by mohl zákazník v budoucnu potřebovat.	Pouze požadované funkce, požadavek minimalizace.
Vztah zákazník–vývojář	Zajištěn smluvně, nedůvěra.	Důvěra a spolupráce.
Lidský faktor	Sekundární, dokumentačně zaměřené procesy se snaží vykázat lidi do role zaměnitelné součástky.	Primární, využívá individualit a silných stránek lidí.
Kvalifikace lidí	Stačí standardní jedinci.	Důraz na schopnosti, znalosti a

	Metodika	
	Rigorózní metodiky	**Agilní metodiky**
		dovednosti lidí.
Specialisté vs. generalisté	Požadavek úzké specializace lidí.	Požadavek integrace znalostí a stálé kooperace, sdílení znalostí v týmu, týmové řešení problému, spíše generalisté než specialisté.
Způsob řízení	Tradiční způsob řízení je formován na základě nedůvěry, direktivní řízení, kontroly.	Vůdcovství a spolupráce, je formováno na důvěře a respektu.
Význam programování při vývoji SW	Důraz a hodnota jsou kladeny na architekturu, požadavky a návrh, kódování a testování jsou chápány jako činnosti s nízkou „konstrukční“ hodnotou.	Důraz na programování jako činnost přinášející hodnotu.
Jednoduchost	Spíše složité řešení, které se snaží obsáhnout i budoucí požadavky.	Důraz na jednoduché řešení žádné zabudovávání budoucích požadavků.
Jednoduchá vs. složitá pravidla	Metodiky se snaží popsat vše, s čím se může vývojový tým setkat.	Obsahují generativní pravidla – minimální množinu věcí, které musíte dělat ve všech situacích.
Modelování	Velký důraz na modelování, zejména modelování předem – big design in front of, potom se zmrazí požadavky.	Agilní modelování, při modelování nejde o model jako takový, ale o akt modelování, smyslem modelování je komunikace.
Forma komunikace	Převážně písemná.	Důraz na komunikaci tváří v tvář.
Dokumentace	Rozsáhlá dokumentace.	Podstatná není dokumentace, ale pochopení.
Způsob vývoje	Spíše vodopádový, případně iterativní a přírůstkový s dlouhými iteracemi.	Přírůstkový vývoj s velmi krátkými iteracemi.
Ekonomika	Zdroje bývají proměnnou veličinou, která zpravidla roste.	Snaha vždy realizovat nejvyšší hodnotu z daných peněz, cílem je hodnota pro zákazníka, ne perfektní systém, hodnota je kombinací funkcí produktu, které odpovídají potřebám zákazníka v určitý čas a za určitou cenu.

3.4.2 Systémy pro výběr metodiky

Systémem, ze kterého vychází například i česká metodika METES, popsaná v kapitole 3.4.2.1, je metoda navržená a patentovaná Davidem Heckselem s názvem “System and method for software methodology evaluation and selection” [13]. Patent byl publikován 2. prosince 2004 a téhož roku byl zveřejněn na osobních stránkách Hecksela.

Bohužel v současné době jsou stránky již nedostupné a obsah je přístupný pouze přes internetový archiv na stránce Archive.org. [12]

Systém definuje 62 atributů označených jako klíčové pro výběr metodiky. Atributy jsou rozděleny do 4 skupin:

1. Lidé
2. Proces
3. Technologie
4. Kořenové atributy

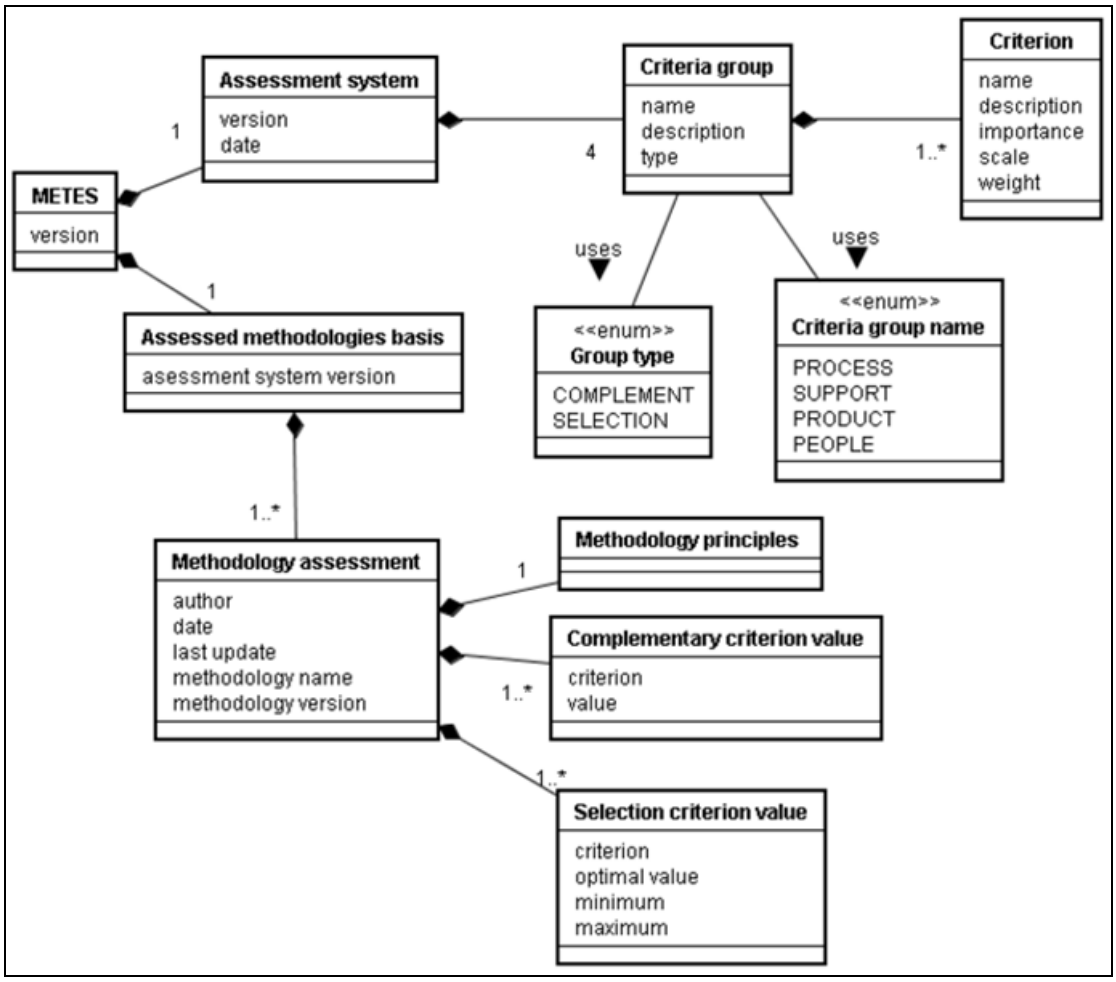
Vybrané atributy, včetně jejich částečného ohodnocení v souvislosti s konkrétní metodikou, tak jak je uvedeno přímo ve výše uvedeném patentu, jsou uvedeny v příloze G.

Tento systém výběru metodiky neobsahuje výchozí ohodnocení ani malé části metodik, ani přesný postup výpočtu. Jedná se o konceptuální model, a proto by jeho použití pro konkrétní projekt v praxi znamenalo značnou výzkumnou a analytickou kapacitu pro přípravu konkrétní implementace.

3.4.2.1 Výběr vhodné metodiky pomocí systému METES

Systém METES, celým názvem Methodology Evaluation System, je systém umožňující výběr vhodné metodiky pro konkrétní projekt a jeho výchozí komponenty jsou připraveny k přímému použití na konkrétním projektu. Autorkou systému je doc. Ing. Alena Buchalcevová, Ph.D., která působí na katedře informačních technologií Vysoké školy ekonomické v Praze. [13]

Struktura systému METES je zachycena formou UML 2.0 diagram na obrázku níže.



Obrázek 7: Konceptuální model systému METES (Zdroj: [12])

Systém METES se tedy strukturou dělí na dvě hlavní části, popis jednotlivých metodik a jejich kritérií ("Methodology assesment") a samotný posuzovací a výběrový systém obsahující vybraná kritéria ("Assesment system"). Jednotlivá hodnotící kritéria vybraných metodik jsou dále rozdělena do skupin:

- Klíčová kritéria
  - Produkt
  - Lidé
- Doplňková kritéria
  - Proces
  - Podpora

Jednotlivá kritéria mají stanoveny váhy pro deterministické vyjádření jejich důležitosti relativně k ostatním kritériím. Systém METES obsahuje i výchozí nastavení vah, je tedy možné výchozí váhy použít, nebo je lze stanovit dle priorit daného projektu.

Níže uvedená tabulka č. 4 prezentuje optimální hodnoty výběrových kritérií pro každou z metodik stanovené přímo systémem METES. Pokud je třeba rozšířit základní bázi metodik, je to možné, předpokladem je ovšem detailní analýza nové metodiky.

Tabulka 4: Optimální hodnoty výběrových kritérií posuzovaných metodik (Zdroj: [12]**)**

Klíčové kritérium	RUP	Open UP	FDD	Scrum	XP	CMMI
Kritičnost projektu	5	2	3	3	3	5
Délka projektu	4	2	2	3	2	4
Stabilita požadavků	2	1	1	0	0	3
Znovupoužitelnost	3	2	2	1	1	3
Velikost projektu	5	2	5	5	3	5
Nedostatek manažerských zkušeností	4	4	3	2	2	4
Nedostatek kvalifikace týmu	5	5	3	1	1	4
Nedostatek motivace týmu	4	4	2	1	1	4
Dostupnost zákazníka	4	3	1	0	0	4
Velikost týmu	5	2	3	3	1	5
Geografické rozložení týmu	5	1	1	3	1	5

Výše uvedená tabulka je v příloze H znázorněna ve formě pavučinového grafu.

Tabulka č. 5 uvádí výchozí váhy výše uvedených výběrových kritérií. Tyto váhy jsou opět výchozí součástí systému METES. Váhy je možno opět upravit dle specifik projektu.

Tabulka 5: Výchozí váhy výběrových kritérií (Zdroj: [13]**)**

Klíčové kritérium	Hodnota
Kritičnost projektu	0,219
Délka projektu	0,133
Dostupnost zákazníka	0,200
Velikost týmu	0,169
Geografické rozložení týmu	0,113
Stabilita požadavků	0,041
Znuvupoužitelnost	0,033
Velikost projektu	0,039
Nedostatek manažerských zkušeností	0,015
Nedostatek kvalifikace týmu	0,020
Nedostatek motivace týmu	0,020

Další součástí potřebnou k vyhodnocení optimální metodiky pro konkrétní projekt je stanovení projektových výběrových kritérií, tedy bodové ohodnocení současného stavu projektu. Toto ohodnocení není a ani nemůže být součástí systému METES, protože je každý projekt je jedinečný a je třeba ho stanovit individuálně. Stupnice, dle které se jednotlivá klíčová kritéria hodnotí, je uvedena v příloze I.

Na základě vah, optimálních hodnot výběrových kritérií a projektových výběrových kritérií je možno vypočítat pomocí váženého součtu nejmenší vzdálenost aktuálního stavu projektu od optimálního řešení, tedy optimální metodiku. Vzorec je uveden na obrázku níže.

$$\sum_{i=1}^{11} (\left| pv_i - mopt_i \right| * vv_i)$$

where
pv_i project selection criteria values
vv_i selection criteria weights
$mopt_i$ optimal values of selection criteria for the methodology

- Proměnná pv_i udává projektová výběrová kritéria (nutno stanovit individuálně pro každý projekt)
- Proměnná vv_i udává váhy kritérií (tabulka č. 5)
- Proměnná $mopt_i$ udává optimální hodnoty výběrových kritérií pro konkrétní metodiku (tabulka č. 4)

Výpočet je tedy nutno provést pro každou z metodik a výsledky porovnat. Nejmenší z hodnot udává optimální metodiku. Tento postup je nutno aplikovat na klíčová výběrová kritéria. Pro výběrová kritéria doplňková je výpočet volitelný a vzorec následující:

$$\sum_{i=1}^{15} (md_i * vd_i)$$

where
md_i values of the complementary criteria for methodology
vd_i complementary criteria weights

- Proměnná md_i udává hodnoty doplňkových výběrových kritérií pro konkrétní metodiku
- Proměnná vd_i udává váhy doplňkových výběrových kritérií

3.5 Přístupy k vývoji SW

Při návrhu SW nebo obecně jakkoli více či méně složitého programu se zpravidla postupuje způsobem “shora dolů”, tedy od obecného pohledu na věc až po jednotlivé dílčí problémy, které jsou řešeny do potřebného detailu.

Postup návrhu tedy přirozeně probíhá v určitých fázích. První je specifikace úlohy/problému, tedy určení toho, co zadavatel od programu očekává, jaké údaje chce do programu vkládat a jaké výstupy z programu očekává. Dále přichází na řadu detailní analýza dané problematiky, tedy tvorba systémové specifikace, tj. určení způsobu řešení konkrétních úkolů a obecně tvorba zadání pro vývojáře, která by měla být akceptována zákazníkem. [17]

Po návrhu standardně nastává fáze realizace, testování, předání výsledného produktu zákazníkovi a případný servis a rozvoj výsledného SW.

Uvedený postup může být pro každý projekt ale značně odlišný a každá fáze s sebou může přinést jiné problémy. Od zrodu softwarového inženýrství bylo definováno mnoho metodik, jak k tomuto procesu přistupovat. Tyto metodiky jsou rozděleny do dvou hlavních kategorií:

1. Tradiční metodiky
2. Agilní metodiky

Pod každou z těchto kategorií metodik spadá nespočet konkrétních metodik a každá z nich má své výhody a nevýhody.

Je běžné, že podniky využívají určitou metodiku nebo kombinaci metodik k sjednocení a ulehčení celého procesu. Dodržování vybraných metodik pak následně požadují i po subdodavatelích a dalších smluvních stranách. [17]

3.5.1 Klasické metodiky

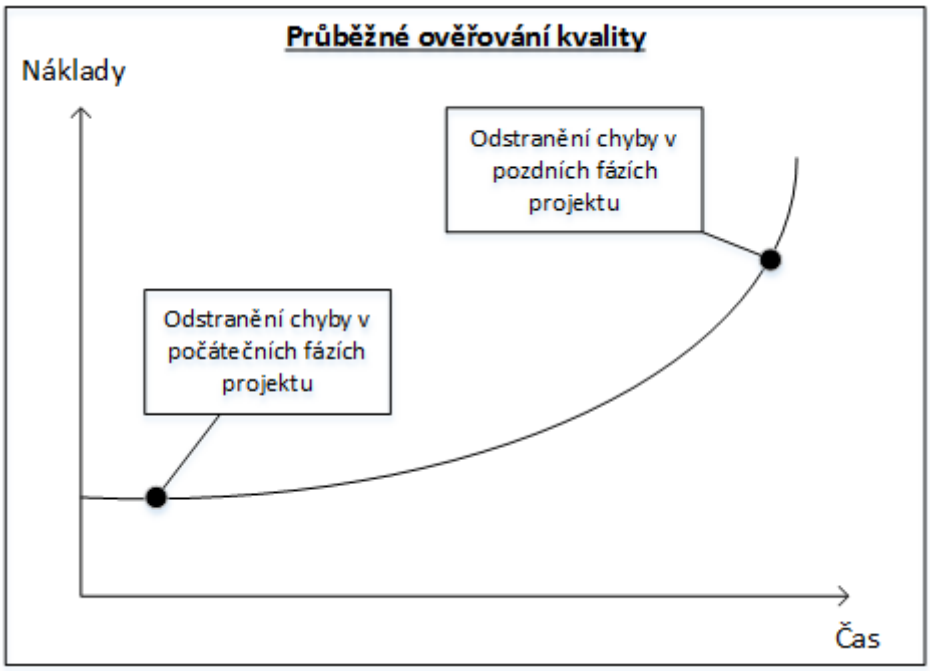
Klasické metodiky, nazývané také jako rigorózní metodiky nebo těžké metodiky, jsou metodiky, které se vyznačují zejména striktně propracovaným procesem řízení a plánování. Tyto metodiky se opírají o základní předpoklad, a to že zákazník zná svoje potřeba a ví, jaký výsledný produkt očekává. Požadavky na SW, dále již neměnné, lze definovat ve fázi analýzy, která předchází fázi vývoje. Většina metodik je odvozena od vodopádového modelu, který je popsán v kapitole 3.5.1.2.

Jeden ze základních problémů tradičních metodik zmiňuje V. Kadlec ve svém článku "Rational Unified Process: základní pojmy":

"Fundamentální problém tradičního, sekvenčního přístupu spočívá v tom, že před sebou tlačí (často dosud netušená) rizika, jejichž odstranění je postupně čím dále dražší. Naproti tomu v iterativním a inkrementálním přístupu (Agilní přístup), který vychází z

Boehmova spirálového modelu, dochází k detekci rizik průběžně. Mezi další výhody iterativního vývoje patří snazší správa změn, lepší znovupoužitelnost, dokonalejší přiblížení k zákazníkovi, atd." [14]

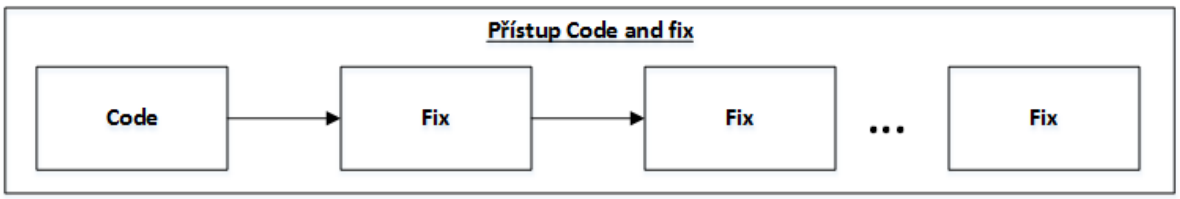
Výše uvedené potvrzuje i následující graf vycházející z obrázku Scotta Amblera, který udává výši nákladů na odstranění chyby v závislosti na čase odhalení chyby.



Obrázek 8: Průběžné ověřování kvality (Zdroj: [15]**)**

3.5.1.1 Přístup "Code and fix"

Přístup "Code and fix" se nedá nazývat metodikou, ale je takzvaným návrhovým antivzorem (angl. Anti-pattern), což v oblasti softwarového inženýrství představuje obecný postup při řešení opakujících se problémů v návrhu SW. Tento přístup je všeobecně vnímán jako nesprávný a neefektivní. [21]



Obrázek 9: Vývojový cyklus přístup Code and fix (Zdroj: Autor)

Vývoj není řízen žádnou vědomou strategií nebo metodikou a tento postup bývá často výsledkem časového tlaku na vývojový tým. Bez jakékoliv přípravy začnou vývojáři produkovat zdrojový kód, což má za následek špatnou čitelnost kódu a obecně žádné pevně

definované architekturní stavební kameny projektu. Testování produktu začíná velice často až v pozdní fázi vývojového cyklu, kdy je oprava chyb již nákladnější.

3.5.1.1.1 Silné a slabé stránky metodiky

3.5.1.1.1.1 Silné stránky

- Přímočarost postupu
- Možnost rychlého dodání alespoň částečně funkčního software zákazníkovi

3.5.1.1.1.2 Slabé stránky

- Zdrojový kód je špatně čitelný pro jiného vývojáře
- Zdrojový kód je náročný na údržbu a opravy
- Špatná struktura (design) systému díky nekontrolovaným změnám
- Nekontrolovatelný a nepředvídatelný vývoj
- Obtížné řízení projektu
- Žádná nebo minimální dokumentace
- Použitelné jen pro drobné projekty se seniorními vývojáři

3.5.1.1.2 Shrnutí

Nezodpovězenou otázkou zůstává, v jaké je výsledný produkt kvalitě. Nevýhody u tohoto přístupu jednoznačně převažují nad jeho výhodami a použití této “metodiky” nelze jinak než výrazně nedoporučit.

3.5.1.2 Vodopádový model

Tento přístup je důležitý pro pochopení dalších přístupů a metodik. Jeho základní principy jsou použity v dnešních moderních metodikách. Vodopádový model je použitelný omezeně, a to pouze u projektů, kde jsou předem známy všechny požadavky, které se v průběhu vývoje již nebudou měnit, což je výjimečný případ. [23]

Jednotlivé fáze vodopádového modelu:

Analýza požadavků

Tato fáze zahrnuje analýzu a specifikaci požadavků zákazníka. Výsledkem musí být přesná specifikace požadavků na SW, obvykle ve formě funkční specifikace. Pokud tomu tak není, pak vzniklé problémy a časový skluz si celý tým již ponese dále celým projektem až do jeho konce. V následujících fázích je již pouze minimální prostor pro změny, je tedy nutné věnovat této fázi velkou pozornost.

Návrh

Konceptuální návrh SW na základě požadavků, které jsou výsledkem fáze předchozí. Výsledkem je přesné zadání pro vývojový tým (SW architekty a programátory), obvykle tedy sepsaná systémová specifikace. Znamená to ale nejen dbát na dokonalou systémovou specifikaci, ale i na to, aby zákazník opravdu pochopil, co jsi objednal a co dostane. Je tedy vhodné, aby za stranu zákazníka revidovalo vzniklou systémovou specifikaci více osob, které mají nejen přehled o business záměru projektu, ale i znalosti z oblasti IT. Pro eliminaci rizik je taktéž vhodné vytvořit "drátěný model" aplikace, kde bude znázorněno GUI a klient si lépe představí výsledek.

Implementace

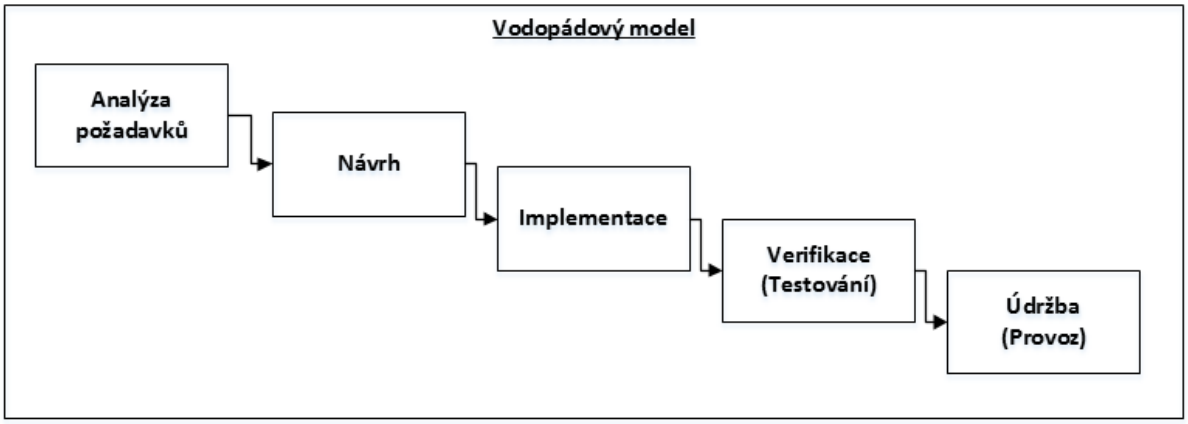
Tvorba software na základě předchozí fáze návrhu, která zahrnuje jak vývoj (psaní zdrojového kódu, ladění kódu), tak i průběžné testování dílčích částí produktu.

Verifikace (Testování)

Testování výsledného systému jako celku. Testuje se, zda jsou splněny všechny definované požadavky a zda je SW stabilní a v souladu se schválenou systémovou specifikací vytvořenou ve fázi návrhu.

Údržba (Provoz)

Provoz představuje nejdelší fázi, která je často pokryta samostatnou smlouvou (servisní smlouvou). Jeden z hlavních důvodů je, že tato fáze obvykle trvá po celou dobu životnosti dodaného produktu a je třeba, aby na straně dodavatele byly neustále k dispozici zdroje pro případnou opravu chyb či rozvoj produktu.



Obrázek 10: Vývojový cyklus Vodopádového modelu (Zdroj: Autor)

3.5.1.2.1 Silné a slabé stránky metodiky

3.5.1.2.1.1 Silné stránky

- Proces vývoje je rozdělen do fází
- Díky rozdělení do fází je možnost rozdělit práci podle specializací
- Jednoduchý, přehledný a jasný postup

3.5.1.2.1.2 Slabé stránky

- Nemožnost vracet se v jednotlivých fázích procesu zpět
- Nelze reagovat na změny potřeb zákazníka po ukončené analýzy
- Nutnost kvalitních a neměnných prvotních vstupů od zadavatele

Nevýhody tohoto přístupu jsou dle V. Kováře zejména tyto:

"Největší nedostatek tohoto modelu, díky kterému je dnes prakticky nepoužitelný, je fakt, že všechny fáze proběhnou pouze jednou v jednom průchodu a nelze se vracet zpět. To je v praxi těžko splnitelný předpoklad, k jednotlivým fázím je mnohdy zapotřebí se vracet opakovaně. Jednoduše není možné nejprve provést kompletní analýzu, pak kompletní design, pak kompletní implementaci, následně vše otestovat a pak hotový produkt předložit zákazníkovi – dodržet přesně tuto posloupnost není jednoduché. Navíc zákazník obvykle nemá předem jasné všechny požadavky. Negativní je také fakt, že zákazník má software k dispozici až v poslední fázi po jeho dokončení. To také znemožňuje zapracování dodatečných požadavků v průběhu vývoje. Další velkou nevýhodou tohoto modelu je značné riziko, že zákazník nebude dodaný produkt (software) akceptovat. To zvyšuje riziko neúspěchu celého projektu a tím i jeho cenu. Cena projektu se zvyšuje také díky obtížnému odstraňováni chyb." [16]

3.5.1.2.2 Shrnutí

Nevýhody vodopádového modelu se snaží pokrýt Spirálový model. Spirálový model povoluje postup do další fáze až na základě důsledně provedené analýzy všech rizik a možných problémů vzniklých z fáze předchozí. Model je založen na iterativním přístupu a především zavádí opakovanou analýzu všech rizik. Pozdější úprava požadavků zákazníka je tak věc, se kterou se tento přístup vyrovná lépe než vodopádový model. [18]

3.5.1.3 Rational Unified Process (RUP)

Rational Unified Process je metodika vývoje softwaru použitelná pro různé rozsahy projektu, kterou lze díky dynamické škálovatelnosti přizpůsobit specifickým potřebám. Obecně je tato metodika vhodnější pro větší projekty a rozsáhlejší vývojové týmy. Metodika klade důraz na analýzu, plánování, řízení zdrojů a dokumentaci. [26]

Hlavní filosofií je objektově orientovaný a iterativní přístup k vývoji SW. Metodika RUP vychází z kolekce těchto osvědčených praktik a postupů při vývoji softwaru:

1. Iterativní vývoj
2. Aktivní správa požadavků
3. Komponentová architektura
4. Vizuální modelování
5. Ověřování kvality software
6. Řízení změn

Fáze projektu

Vývoj projektu má podle RUP tyto fáze:

1. Zahájení
    a. Definice účelu, rozsah projektu a jeho kontextu.
2. Příprava
    a. Analýza požadavků zákazníka a definice základů architektury.
3. Konstrukce
    a. Samotné programování, tj. psaní zdrojových kódů.
4. Předávání
    a. Předání hotového produktu zákazníkovi nebo zahájení dalšího cyklu (iterace) vývoje.

Každá z těchto fází (kromě první) může být rozdělena do více částí, zvaných iterace.

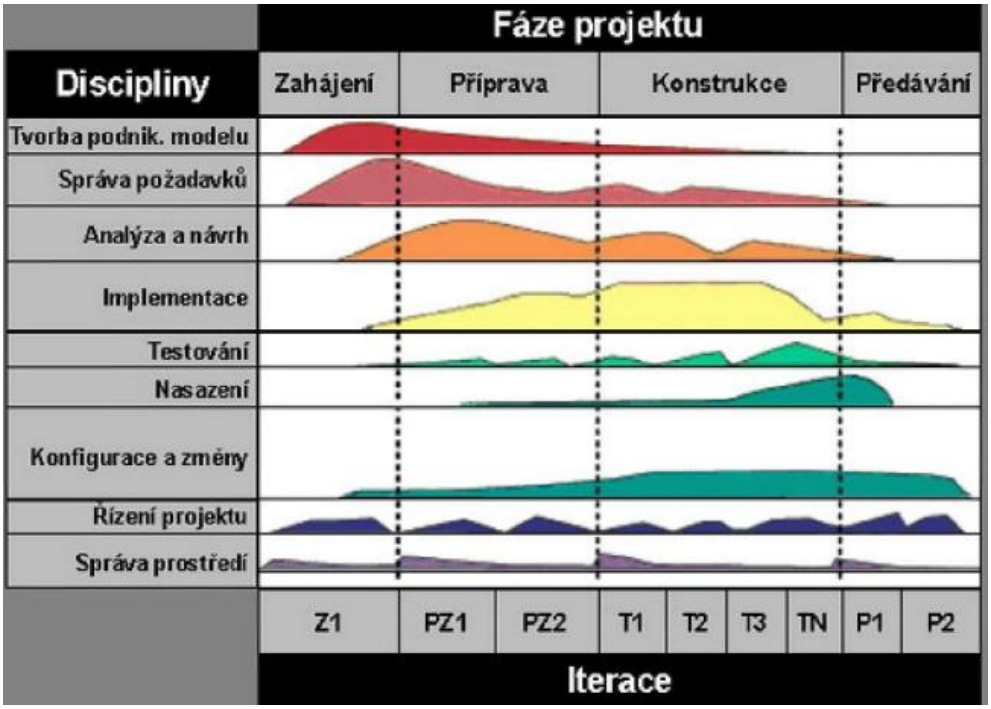
Disciplíny

Jako disciplíny se nazývají činnosti, které provázejí vývoj projektu:

1. Tvorba modelu
2. Správa požadavků
3. Analýza a návrh
4. Implementace
5. Testování

6. Nasazení
7. Konfigurace a změny
8. Řízení projektu
9. Správa prostředí

Na níže uvedeném obrázku jsou na vodorovné ose znázorněny uvedené fáze projektu a na horizontální ose výše uvedené disciplíny.



Obrázek 11: Fáze projektu v metodice RUP (Zdroj: [19]**)**

3.5.1.3.1 Silné a slabé stránky metodiky

3.5.1.3.1.1 Silné stránky

- Metodika pokrývá velkou část životního cyklu software
- Důraz na dokumentaci
- Dobře dostupné výukové a tréninkové materiály

3.5.1.3.1.2 Slabé stránky

- Nutnost velmi zkušených členů týmu
- Příliš komplexní vývojový proces
- Integrace na okolní systémy po celou dobu vývoje přináší vícepráce

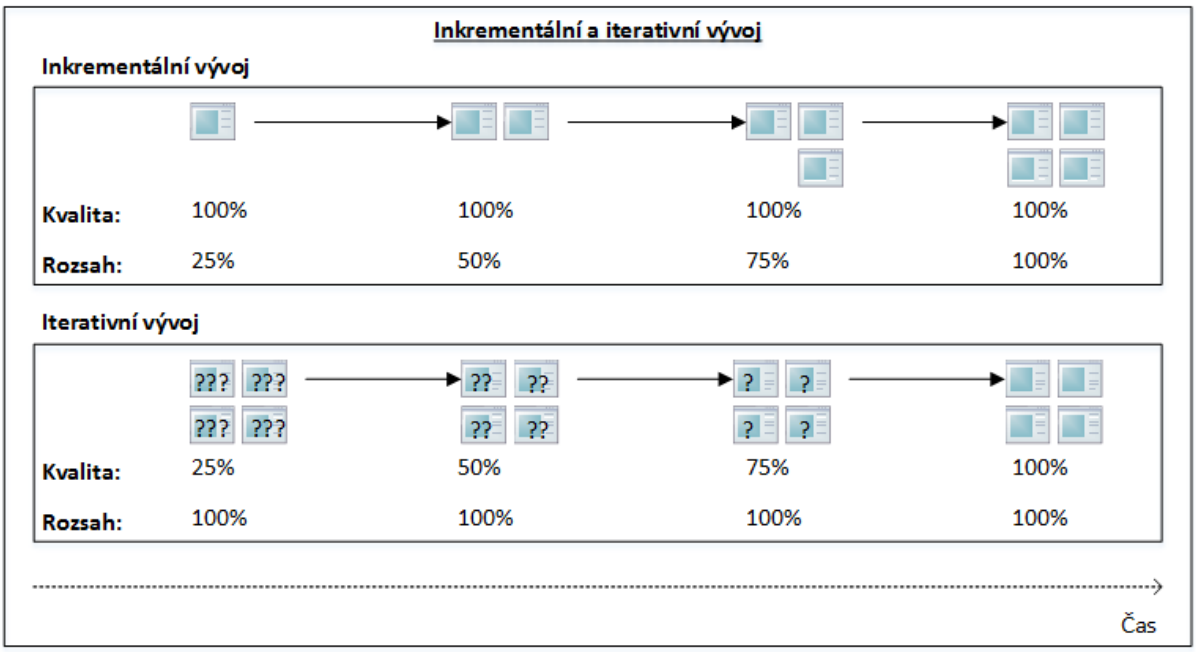
3.5.2 Agilní metodiky

Hlavním důvodem vzniku agilních metodik je potřeba rychlé reakce na změny s přijatelnými finančními náklady. V případě užití tradičních metodik tyto náklady exponenciálně rostou, což je v současné době, kdy je nutno reagovat na změny na trhu čím dál rychleji, jeden z hlavních problémů. Také testování probíhá v průběhu projektu (iterativně), zatímco u tradičních metodik se testování provádí až ke konci projektu. Agilní přístupy obecně nevedou k menšímu počtu chyb v průběhu vývoje, ale vedou k jejich včasnému odhalení a k rychlejší a efektivnější nápravě. Iterativní vývoj sebou přináší časté vydávání nových verzí výsledného SW, a tím i včasnou zpětnou vazbu od zákazníka.

Agilní metodiky ale také v určitých oblastech kladou jak na vývojový tým, tak na zákazníka vyšší nároky než metodiky tradiční. Například z důvodu, že analýza neprobíhá na začátku projektu jako jedna etapa, ale probíhá v podstatě neustále, je potřeba, aby zákazník nebo zástupce zákazníka byl součástí týmu. Neustálá komunikace se zákazníkem je zde naprosto klíčová. Tento požadavek může být v určitém prostředí obtížně splnitelný, například pokud se jedná o korporaci, kde je organizační struktura rozsáhlá a kompetence roztříštěné mezi mnoho subjektů. Pak může reakce na dotazy trvat déle (týdny, měsíce) a v tomto momentu agilní metodiky často selhávají. Jejich implementace v tomto prostředí však není zcela nemožná. Pojednává o tom samostatná kapitola 3.5.2.8.

3.5.2.1 Inkrementální a iterativní vývoj

Inkrementální a iterativní vývoj jsou pojmy, se kterými se v oblasti Agilních metodik vývoje SW setkáme velice často. Pro jejich pochopení je nutné nejprve oba pojmy definovat, s čímž nám může pomoci níže uvedený obrázek.



Obrázek 12: Inkrementální a iterativní vývoj (Zdroj: [16]**)**

3.5.2.1.1 Inkrementální vývoj

Inkrementální vývoj spočívá v tom, že postupně dodáváme zákazníkovi jednotlivé části systému (takzvané inkrementy), které mají vždy implementovanou veškerou požadovanou funkcionalitu. Postupně dodáváme další inkrementy, dokud není systém kompletní. Jedná se o přírůstky rozsahu (kvantity), kde kvalita jednotlivých inkrementů je úplná. [2]

3.5.2.1.2 Iterativní vývoj

Iterativního vývoj je založen na přístupu, kdy systém dodáváme kompletní ale s omezenou (částečnou) funkcionalitou. Postupně provádíme jednotlivé iterace, dokud nedosáhneme plné kvality (funkčnosti) systému. V každé iteraci vylepšujeme funkcionalitu celého systému. Systém je tedy vždy úplný a při každé iteraci se vylepšuje jeho kvalita. V iterativním vývoji se jedná o přírůstky kvality. [16]

3.5.2.1.3 Kombinace obou přístupů

Iterativní vývoj je dnes základem všech moderních metodik vývoje software, jak klasických, tak i agilních. Nicméně iterativní vývoj je ve většině případů kombinován s vývojem inkrementálním. Důvod je jednoduchý, u větších systémů nejsme schopni vytvořit a dodat najednou kompletní systém. Velké projekty je vždy nutné rozdělovat na menší části (subsystémy nebo moduly).

3.5.2.2 Základní principy agilních metodik

Agilní metodiky vycházejí z určitých základních principů, kterými jsou:

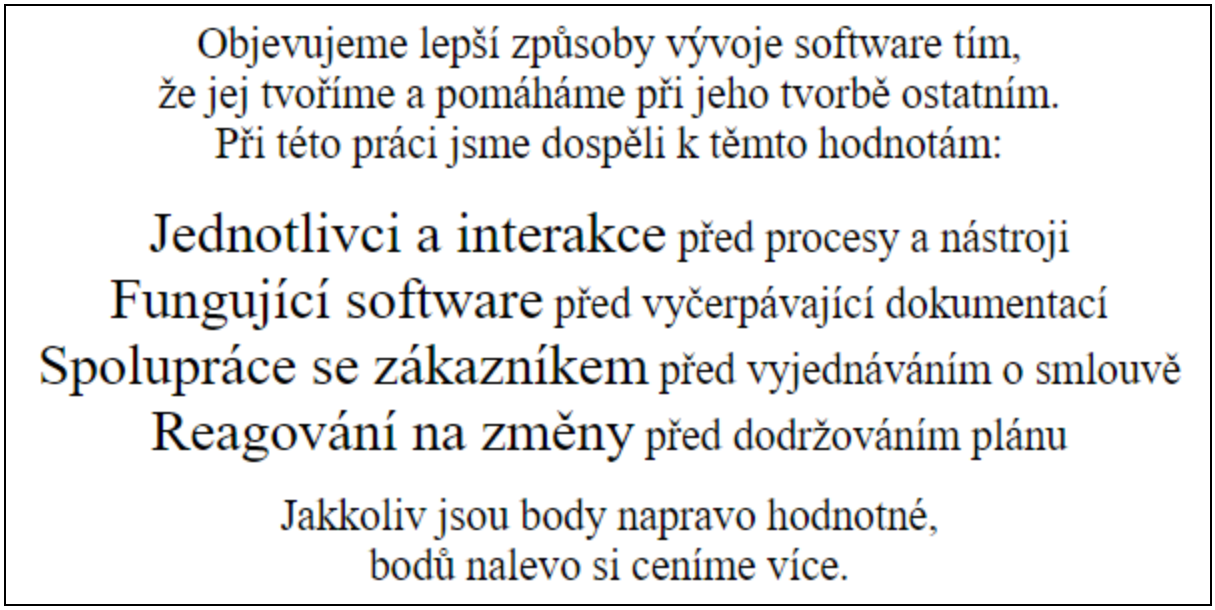
1. **Iterativní a inkrementální vývoj s velmi krátkými iteracemi:**
   - plán vývoje (plán projektu) je sestaven tak, že nové funkce se dodávají často (třeba i denně). Zákazník je průběžně konfrontován s novými konfiguracemi; má pocit, že se „něco děje“ a zároveň jasně vidí, jakou rychlostí vývoj postupuje. Není-li spokojen, má možnost např. upravit požadovanou funkcionalitu.
2. **Důraz na přímou, osobní komunikaci v rámci týmu:**
   - ve všech sociálních skupinách obecně platí, že za nejzávažnějšími problémy stojí špatně fungující komunikace. Agilní metodologie se pokouší předejít těmto problémů tím, že komunikaci prohlašují za jednu z forem vývoje a integrují ji přímo do vývojového procesu. Praktickými důsledky jsou nejrůznější techniky, např. časté schůzky týmu se zákazníkem, párové programování, refaktorizace kódu apod.
3. **Permanentní sepětí a komunikace se zadavatelem, resp. uživatelem:**
   - zákazník by měl být členem vývojového týmu, měl by komunikovat s vývojovým týmem, spolupodílet se na návrhu a spolurozhodovat o testech.
4. **Rigorózní, opakované, průběžné automatizované testování:**
   - díky častým změnám systému je nutné průběžně ověřovat jeho správnost. Testy by měly být automatizované (ručně prováděné testy jsou obvykle závislé na subjektivních vlastnostech rukou, které je provádí) a měly by být napsány dokonce před implementací testované části.

“Agilní metodiky naopak srážejí význam formálních artefaktů a dokumentů. Vycházejí z neochvějného přesvědčení, že základním, jednoznačným, exaktním, nezpochybnitelným a v podstatě jediným spolehlivým nositelem informace je zdrojový kód. Na jeho tvorbu proto kladou největší důraz; s jeho vytvářením začínají skoro bezprostředně po stanovení cílů projektu (lépe řečeno po napsání testů příslušného modulu); při jeho psaní doporučují dodržovat striktní pravidla (štábní kultura) apod.” [20]

3.5.2.3 Agilní manifest

V únoru roku 2001 se Utahu sešli odborníci z oblasti softwarového inženýrství a vývoje softwaru, aby diskutovali o odlehčených metodách vývoje. Sepsali Manifest agilního programování, kde definovali přístup k vývoji nyní známý jako agilní programování.

Základem agilního přístupu je tedy Agilní manifest, který není prezentován ve formě předpisu či striktního dokumentu, ale ve formě prohlášení, které se může na první pohled zdát jako podivný a nejasný popis toho, jak nelze systematicky odvádět práci. Nicméně opak je pravdou.



Obrázek 13: Manifest Agilního vývoje software (Zdroj: [22]**)**

3.5.2.3.1 Priority agilního programování

Z Agilního manifestu tedy vyplývají čtyři základní priority agilního programování, které jsou níže detailněji rozebrány a vysvětleny. [22]

1. **Jednotlivci a interakce před procesy a nástroji**
    a. Pro agilní metodiky jsou důležití jednotlivci, jejich přínos projektu a jejich potřeby. Procesy a nástroje agilního přístupu jsou koncipovány jednoduše, ale tak, aby splnili svůj účel, nikoli tak, jak tomu je v tradičních metodikách, kdy se lpí na přesných procesech.
2. **Fungující software před vyčerpávající dokumentací**
    a. Fungující produkt je primárním cílem všech metodik a dokumentace je nástroj, který má pomoci k dosažení tohoto cíle. Agilní metodiky dokumentaci v

žádném případě nezavrhují, pouze se snaží o to, aby dokumentace vznikala přirozeným způsobem, kdy v ní bude obsaženo to důležité a nebudeme například lpět na přesném formálním popisu dodržujícím předepsané šablony.

3. **Spolupráce se zákazníkem před vyjednáváním o smlouvě**
    a. Tento bod rozhodně nepopírá nutnost smluv. Pouze upozorňuje na to, že všichni (jak zákazník, tak dodavatel) mají stejný cíl a chtějí fungující produkt. Proto by měli co nejvíce komunikovat a spolupracovat, což je cesta k úspěchu, nikoli lpět na smluvních závazcích.
4. **Reagování na změnu před dodržováním plánu**
    a. Je nutné vycházet z cíle, kdy nejvyšší prioritou je zákazníkovi dodat produkt fungující podle jeho představ. V naprosté většině případů zákazník nedokáže předem přesně specifikovat své požadavky v takovém detailu, aby nedošlo po stanovení ceny k určité formě zpřesnění jeho požadavků. Tomuto se agilní metodiky nebrání a nekladou zbytečné byrokratické překážky, ale naopak se snaží vyjít vstříc.

Výše uvedené body rozhodně nemají znamenat, že se přestaneme řídit jakýmkoli řádem. Interpretovat takto rozdíl mezi agilními a tradičními metodikami není možné. Jedná se pouze o jiný pohled na věc, který staví z velké části na důvěře a na lidských vlastnostech, což jsou věci, které nelze smluvně podchytit. Takovýto vztah nemusí být jednoduché mezi zákazníkem a dodavatelem navodit a vyžaduje to určitý čas pro vybudování potřebné důvěry. Zkušenosti ale ukazují, že je to cesta, která vede k cíli.

V manifestu je dále definováno 12 principů agilního programování, které jsou uvedeny níže spolu s vysvětlením každého jednotlivého bodu.

3.5.2.3.2 Principy agilního programování

1. **Nejvyšší prioritou je uspokojit zákazníka díky rychlému a průběžnému dodávání kvalitního softwaru.**
    - Jednoduše řečeno, celý projekt má jeden cíl, a to uspokojit zákazníka. Vše ostatní je pouhým nástrojem k dosáhnutí tohoto cíle a je tomuto cíli podružné.
    - Rychlé a průběžné dodávání produktu dává zákazníkovi možnost vidět průběh prací, a tedy včas vyjádřit připomínky k dodávanému dílu.
2. **Změnové požadavky jsou vítány, dokonce i v průběhu vývoje. Agilní procesy je zpracují tak, aby zákazníkovi přinášely konkurenční výhody.**

- Jak již bylo uvedeno výše, vyhnout se změnám není prakticky možné. V agilních přístupech nejsou nikterak potlačovány, a pokud je požadovaná změna relevantní pro přínos hodnoty do projektu, je vítaná.

3. **Dodávejte fungující software často, v intervalech týdnů až měsíců. Upřednostňujte kratší intervaly dodání.**
   - Zákazník je vždy potěšen, pokud vidí, jak se celé dílo formuje do výsledného stavu. Zákazník sám může dokonce před implementací všech požadovaných vlastností rozhodnout o vydání produktu, který nebude zcela hotov a zcela dokonalý, ale díky včasnému uvedení na trh přinese požadovanou hodnotu a náskok před konkurencí.
   - Časté vydávání nových verzí nese již zmíněnou výhodu včasného odhalení chyb v produktu. Čím dříve je chyba odhalena, tím nižší jsou náklady na její odstranění.
4. **Lidé z businessu a vývojáři musí spolupracovat každý den během celého projektu.**
   - Tento bod klade relativně velké nároky na stranu zákazníka a je často opomíjen. Agilní metodiky počítají s tím, že zákazník bude členem dodavatelského týmu a komunikace mezi zákazníkem a analytiky nebo vývojáři nebude zatížena žádnou bariérou. Pokud zákazník nepochopí, že je jeho pravidelná účast a zpětná vazba k projektu bezpodmínečně nutná, není možné agilní metodiky aplikovat. Jak uvádí Jiří Knesl v přednášce s názvem "SprintMethod" na konferenci DevFest z roku 2012 pořádanou firmou GUG.cz ve spolupráci s českou pobočkou Google, tak není možné pracovat agilně se zákazníkem, kterému trvá 2 měsíce odpovědět na jeden email. Jedná se o skutečný příklad z praxe, kdy byla zákazníkem jedna z velkých českých zdravotních pojišťoven. Organizační struktura a procesy ve firmě nebyly nastaveny tak, aby mohly pružně reagovat na dotazy dodavatele, a použití agilních metodik tedy nebylo možné. [31]
5. **Pro práci na projektu vybírejte motivované jedince. Dejte jim prostředí a podporu, kterou potřebují, a důvěřujte jim, že práci dokončí.**
   - Agilní metodiky staví na samostatnosti a profesionalitě lidí. Nelze tedy složit agilní dodavatelský tým z juniorů v dané oblasti zaměření. V nepříznivých podmínkách lze s kvalitními lidskými zdroji projekt udržet při životě, bez

kvalifikovaných členů bude předurčen k zániku. Agilní metodiky často nevyžadují přesně stanovenou hierarchii, každý by měl totiž vědět, kde je jeho pozice, jaké má pravomoci a odpovědnosti.

6. **Nejúčinnější metoda sdílení informací vývojářskému týmu (i uvnitř tohoto týmu) je osobní setkání.**
    - V písemné formě se může často informace ztratit nebo ji může protistrana pochopit jinak, než bylo zamýšleno. Pokud bude dodrženo pravidlo číslo 4, není důvod nekomunikovat osobně a tato možnost by měla být maximálně využívána. Při osobním setkání je možno vnímat i mimoslovní komunikaci - mimiku, gesta a pocity zákazníka, které můžou být také vypovídajícím zdrojem informací.
7. **Fungující software je hlavním měřítkem postupu vývoje.**
    - Ze všech možných měřitelných ukazatelů, jako je počet chyb a otestovanost systému, má největší význam fungující produkt, se kterým je zákazník spokojený.
8. **Agilní procesy podporují udržitelný vývoj. Sponzoři, vývojáři i uživatelé by měli být schopni dodržovat stálý výkon, dokud je třeba.**
    - Udržení konstantního pracovního tempa během dodávky produktu je klíčovým požadavkem, který klade nároky na všechny zainteresované strany. Tento požadavek podporuje i bod 5, který spoléhá na profesionalitu jedinců, kteří vědí, jaké tempo je nutné zvolit a po jakou dobu je nutné ho udržet.
9. **Průběžná pozornost věnovaná technické dokonalosti a dobrému návrhu posiluje agilní přístup.**
    - Iterativní přístup k vývoji nám dává "průběžnou pozornost", kdy odevzdáváme produkt po částech a zpětnou vazbu máme k dispozici průběžně, nikoli až po naprogramování kompletního SW jako u tradičních metodik, kdy je už pozdě provádět konceptuální změny v návrhu SW. Technickou dokonalost a dobrý návrh tedy neustále ověřujeme a zlepšujeme.
10. **Základem je jednoduchost – umění co nejvíce práce vůbec nedělat.**
    - O tento bod se opírá celý agilní vývoj. Předem předpokládáme, že od zákazníka přijdou změnové požadavky, že se bude zadání (více či méně) měnit, že budeme již naprogramované části kódu refaktorovat, nebo zcela

přeprogramovávat. Programujeme tedy pouze to, co je nezbytně nutné pro dodání kvalitního produktu. Nevymýšlíme přehnaně robustní kód, který bude sice univerzální a připravený na budoucí požadavky, o kterých ale v tuto chvíli nic nevíme, a je velké nebezpečí, že budeme nuceni již vytvořený kód zcela přeprogramovat. Je tedy důležité rozpoznat, co je a co není důležité, a to nedůležité jednoduše nedělat.

11. **Nejlepší architektury, požadavky a návrhy vznikají v týmech, které se samy organizují.**
    - Agilní metodiky podporují iniciativu pracovníka ke zlepšení produktu, a to bez ohledu na jeho pozici a odbornost. Rozdělení rolí zde není natolik striktní a důraz je kladen na týmovou spolupráci.
12. **Tým v pravidelných intervalech vyhodnocuje svou práci a upravuje své postupy tak, aby byl co nejefektivnější.**
    - Tým by měl své fungování optimalizovat, zlepšovat a odstraňovat chyby ve svém fungování, na které v průběhu prací narazil. Lze sem zařadit například proces automatizace nových a opakujících se procesů.

Projdeme-li výše zmíněné principy, zjistíme, že se nejedná o nic nelogického nebo složitého, ale že se jedná o jasně a jednoduše definované principy, kterými bychom se měli při vývoji SW řídit, a většinou se jimi již alespoň z části řídíme.

3.5.2.4 Scrum

Scrum je agilní metodikou pro vývoj a údržbu informačních systémů. Popis metodiky je výstižně popsán v dokumentu “The Scrum Guide - The Definitive Guide to Scrum: The Rules of the Game”, který má pouze 16 stran, a právě jeho délka přispěla k rozšíření této metodiky po celém světě. Metodika prochází neustálými revizemi a její poslední verze je z července roku 2016. Český překlad je dostupný ze stránek scrumguides.org, bohužel ale pouze ve verzi z července roku 2013. [33]

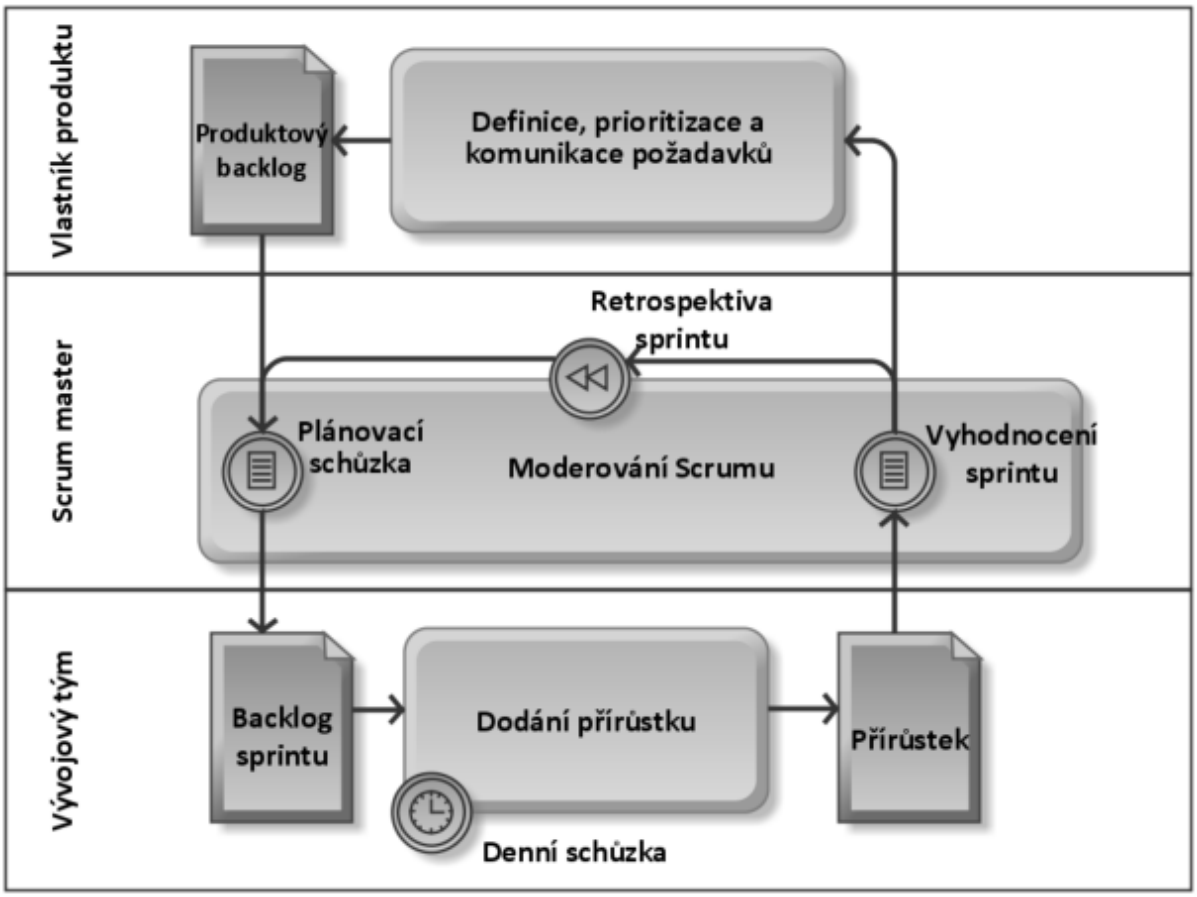
Sami autoři charakterizují Scrum třemi základními body:

1. Jednoduchý
2. Srozumitelný
3. Extrémně obtížný pro dokonalé zvládnutí

V následujících kapitolách budou popsány role definované metodikou Scrum, pojmy a artefakty, které metodiku utvářejí. [34, 36]

3.5.2.4.1 Role

Jak již bylo zmíněno, agilní metodiky nejsou tak byrokraticky náročné, a to platí i zde, kdy rozlišujeme pouze tři základní formální role - Vlastník produktu, Scrum master a Člen vývojového týmu. Omezení tří rolí však neznamená omezený rozsah vykonávaných prací. Vizualizaci rolí a vazby mezi nimi popisuje obrázek č. 14 uvedený níže.



Obrázek 14: Procesní model Scrum (Zdroj: [24]**)**

3.5.2.4.1.1 Vlastník produktu

Vlastníkem produktu (angl. Product Owner) je jednoduše řečeno zákazník, často reprezentovaný zástupcem zákazníka, což může být jak externí subjekt, tak interní zaměstnanec firmy zákazníka. Mělo by se ovšem ideálně jednat vždy o jednu osobu. Jak uvádí R. Pichler, je nutné se vždy snažit o určení právě jedné zodpovědné osoby, jinak může ve výsledku docházet k nekonzistenci výsledného produktu a obecně delší reakční a rozhodovací době, což není v souladu s agilními metodikami, které se snaží jednat přímo, rychle a jasně. R. Pichler uvádí tři možná řešení tohoto problému [39]:

1. **Stanovení hlavního vlastníka produktu**, který určuje plán a strategii a řídí podřízené vlastníky produktů, kteří nejsou globálními vlastníky produktu, ale zodpovídají pouze za určitou oblast nebo komponentu.
2. **Rozdělení produktu na více samostatných produktů**, kde bude mít každý nově vzniklý produkt svého vlastníka. Tuto možnost lze kombinovat s první možností, kdy budou produktoví vlastníci jednotlivých produktů zastřešeni jedním hlavním vlastníkem.
3. **Rozdělení na strategické** (strategie, plán, finanční plán) **a taktické** (prioritizace úkolů, spolupráce s vývojovým týmem) **produktové role**. Toto rozdělení rolí používá například metodika SAF jako základ pro výchozí rozdělení pravomocí. Nicméně tento způsob není doporučen, pokud není vyžadována úzká integrace taktických a strategických rozhodnutí, což může vést k nesouladu výsledných rozhodnutí. Možné použití se nabízí v případech, kdy je již produkt hotov a je pouze rozšiřován a vylepšován, a kdy se žádné velké a zásadní změny v jeho koncepci již neočekávají.

Role vlastníka produktu sebou nese zodpovědnost za následující činnosti:

1. Definování finální podoby produktu (vize)
2. Definování funkcionality (backlog)
3. Prioritizace definované funkcionality

ad 1) Definicí podoby produktu (definicí vize) se rozumí určování směru a cíle projektu. Každý projekt je jedinečný a i během vývoje produktu jej mohou ovlivňovat jak vnější (mimo firmu zákazníka), tak vnitřní (uvnitř firmy zákazníka) podněty a změny, a je tedy potřeba neustálé interakce a korekce směru projektu. [42]

ad 2) Jednotlivé definované funkcionality se řadí zpravidla do dvou seznamů: projektový backlog a sprint backlog (podmnožina projektového backlogu). Definice funkcionalit a jejich podrobný popis je sice zodpovědností vlastníka produktu, ale jako i ostatní úkoly v metodice Scrum je nutná jejich komunikace se všemi zúčastněnými stranami. Často se stává, že zákazník ani vlastník produktu neznají technické možnosti a limity dané platformy a programovacího jazyka, ale právně od nich se může výsledná funkcionalita odvíjet. Tento moment je, podobně jako ostatní činnosti v metodice Scrum, zejména o spolupráci a komunikaci. [34, 36]

ad 3) Stanovení priorit je čistě na vlastníkovi produktu a dodavatelský vývojový tým by neměl rozhodnutí ovlivňovat. Dodavatel může do stanovení priorit zasáhnout pouze z hlediska technické realizovatelnosti (např. pokud byla dána priorita požadavku, který je technicky závislý na jiném požadavku, který má prioritu výrazně nižší a nebude tedy v blízké době realizován). Priority se mohou v čase měnit a můžou být závislé na dlouhodobých nebo krátkodobých cílech firmy nebo na smluvních nebo dokonce etických základech. Proto je pouze vlastník produktu tím subjektem, který priority určuje. [25]

3.5.2.4.1.2 Scrum master

Tuto roli lze charakterizovat jako manažerskou, nicméně je nutno zdůraznit, že Scrum počítá s kvalifikovanými členy týmu, kteří vědí, co mají dělat a kde je jejich místo. Tato role tedy nemá za cíl jednotlivé členy přímo řídit, ale spíše je ochraňovat a zajistit vše potřebné pro výkon jejich práce.

Základní úkoly Scrum mastera by se daly zjednodušeně vyjmenovat jako:

1. Pomoc týmu
2. Řešení problémů
3. Motivace týmu
4. Ochrana týmu (dozor nad pravidly metodiky Scrum)

Zde si zaslouží rozvést zejména bod 4, který mluví o ochraně týmu. Jedná se zejména o ochranu před vlivy, které mohou tým vyrušovat při práci nebo mu práci komplikovat. Nejčastějším jevem z praxe je v tomto případě situace, kdy je iterace (sprint) v průběhu a zákazník nebo zástupce zákazníka (Vlastník produktu) potřebuje nějaký požadavek rychle dokončit nebo pozměnit a komunikuje přímo s vývojářem (za stranu dodavatele). V průběhu sprintu jsou již ale práce rozděleny, odhadnuty a naplánovány a takovýto zásah zvenčí může celý sprint ohrozit, a tím ohrozit i celý projekt. Tyto tendence musí Scrum master odhalit, odstínit od jejich řešení vývojový tým a jejich řešení naplánovat například do dalšího sprintu. Tuto problematiku s určitou dávkou humoru, ale zcela výstižně popisuje například Jeff Sutherland v krátkém videoklipu s názvem “Scrum Master movie”. [41]

3.5.2.4.1.3 Vývojový tým

Vývojový tým se skládá z členů týmu, a to jsou právě ty subjekty, které odvádějí potřebnou práci k dokončení projektu. Administrativu za ně z velké části řeší Scrum master. Úkolem vývojového týmu je **implementovat jednotlivé funkcionality** zařazené v

daném sprintu. Nedílnou součástí práce všech členů týmu je **připomínkování projektu**, kde každý má právo a vlastně i povinnost se věcně a konstruktivně vyjádřit jak ke konkrétním problémům, tak obecně k tomu, jak projekt zlepšit.

Jak již bylo řečeno, metodika Scrum je ve své podstatě jednoduchá a v oblasti hierarchie týmu nedefinuje další specifické pozice členů týmu (např. roli, vývojáře, analytika a testera).

3.5.2.4.2 Pojmy a artefakty

V této kapitole se seznámíme s terminologií metodiky Scrum, s definicí pojmů, které byly zmíněny v předchozí kapitole.

Sprint je iterací, tedy základním principem agilních metodik. Typicky se jedná o interval dlouhý 7 až 30 dní, jeho délka by ale měla být stanovena s ohledem na celkovou délku projektu a na velikost a dynamiku projektového týmu. Délka sprintu je zpravidla po dobu trvání projektu fixní a sprinty se opakují, dokud není produkt v požadovaném rozsahu a kvalitě.

User story je obecně v agilních metodikách přijímanou podobou toho, jak by měl být definován požadavek na funkcionalitu. Základní podstatou je, aby bylo jasně zodpovězeno na otázky:

1. Jako kdo?
2. Chci co?
3. Aby proč?

Příklad dvou variant z praxe, který uvádí Zuzana Šochová a Eduard Kunce:

1. User story: “Kontaktní formulář”
2. User story: “Jako administrátor chci kontaktní formulář, abych se včas dozvěděl o chybách v systému.”

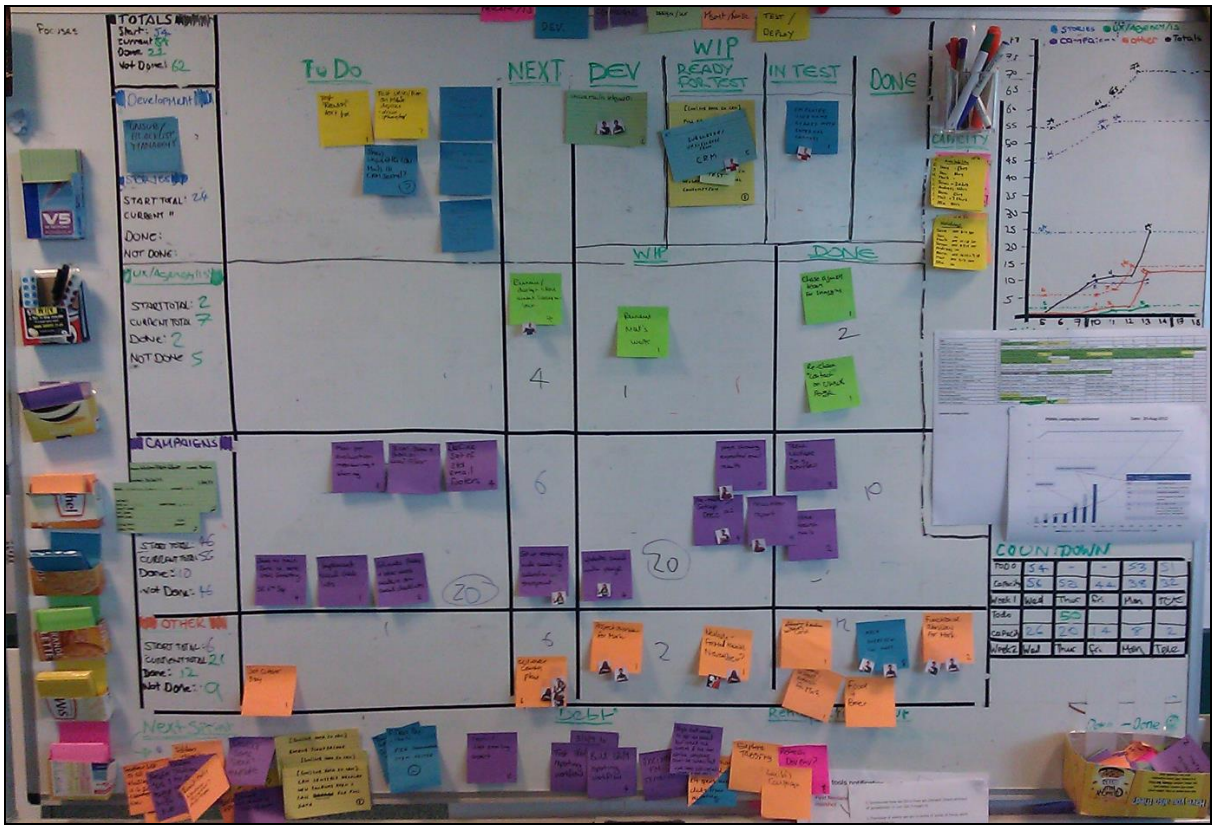
“Je velice pravděpodobné, že když jste výše uvedené četli, vybavily se vám dvě různé věci. Myslíte si, že je to z kontextu firmy a produktu jasné? Ne vždy. Většinou je to jasné pouze Vlastníkovi produktu, ten má v hlavě obrázky, jak přesně se má systém chovat a jak má vypadat. On je součástí příběhu zákazníka, ale v User story jde o to, aby ten obrázek, který má v hlavě, sdělil ostatním, tak aby ho ani oni nikdy nezapomněli.” [43]

Product backlog je prioritizovaný katalog všech požadavků na funkcionality, které mají být v rámci projektu implementovány. Jeho tvorbu má na starosti vlastník produktu (popsán v kapitole 3.5.2.4.1.1) a ve spolupráci s dodavatelem by jednotlivé položky

(typicky User story) měly být již klasifikovány společně definovanou (relativní) stupnicí náročnosti.

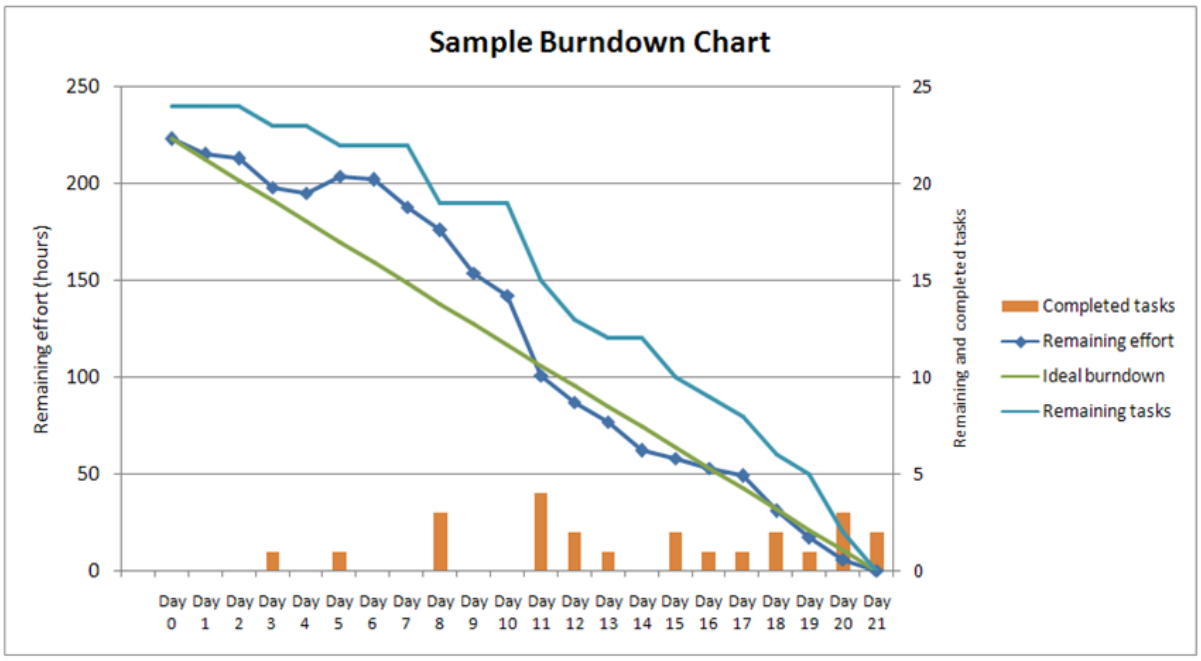
Sprint backlog je podmnožinou Product backlogu a definuje funkčnosti zahrnuté do konkrétního sprintu. Volba obsahu sprint backlogu probíhá zpravidla dle priorit stanovených v product backlogu.

Scrum tabule je vhodným a často používaným nástrojem nejen v metodice Scrum, ale zejména v metodice Kanban popsané v kapitole 3.5.2.7. Z této tabule by mělo být zřejmé, jaké funkcionality má tým v daném sprintu dokončit a v jakém jsou stavu. Tabule může být reprezentována fyzickou podobou, jak ilustruje níže uvedená fotka, nebo například pomocí podpůrných SW nástrojů, kterým se věnuje kapitola 4.2. Ukázku SW implementace Scrum tabule v nástroji YouTrack zachycuje příloha B. Struktura tabule je rozepsána v kapitole 3.5.2.7.1, která se věnuje metodice Kanban.



Obrázek 15: Fyzická implementace Scrum tabule (Zdroj: [25]**)**

Burndown graf se využívá k zobrazení průběhu prací v daném sprintu, případně k predikci termínu dokončení sprintu. Níže uvedený burndown graf má na ose x uvedeny jednotlivé dny sprintu (0 až 21) a na ose y je uveden součet odhadnutých hodin všech funkcionalit zařazených do daného sprintu (225 hodin).



Obrázek 16: Ukázkový burndown graf (Zdroj: [27]**)**

3.5.2.4.3 Silné a slabé stránky metodiky

3.5.2.4.3.1 Silné stránky

- Metodika umožňuje měřit počet generovaných chyb dle jednotlivých sprintů a tím sledovat i tendenci počtu chyb
- Efektivní oprava chyb, díky menšímu časovému odstupu mezi nahlášením chyby a přeprogramováním postižené funkcionality
- Software se opakovaně testuje v každém sprintu
- Umožňuje přesněji odhadovat náklady na celý vývoj softwaru. Jelikož je vývoj rozdělen do jednotlivých sprintů, je možné brzy vyčíslit náklady na jeden sprint a tento odhad použít pro odhad nákladů celého životního cyklu vývoje softwaru.

3.5.2.4.3.2 Slabé stránky

Zejména se jedná o jednoduchost Scrumu a o to, že Scrum je ve své podstatě pouze rámcem pro vývoj SW a občas nebývá vůbec označován jako metodika. Například Martin Hinshelwood ve svém komentáři výše uvedené potvrzuje a uvádí, že pro úspěšné fungování Scrumu ve většině větších firem je ještě potřeba implementovat “obalovou” metodiku, která ve svých základech samozřejmě nesmí popírat principy Scrumu. Dále uvádí, že slabou stránkou je také rozsah metodiky, kde nejsou explicitně pokryty části podniku, jako infrastruktura, management, helpdesk, a je třeba jejich správu implementovat do Scrumu svépomocí.

Dále je možné uvést:

- Tým musí být složen ze zkušených a angažovaných jedinců
- Scrum master musí věřit vývojovému týmu a naopak. Nedůvěra anebo přílišná kontrola může vést k frustraci
- Ztráta člena vývojového týmu během projektu je velkým zásahem do fungování a může projekt ohrozit
- Náročná kontrola kvality projektu - nutnost regresních testů po každém sprintu

3.5.2.5 Feature Driven Development (FDD)

Metodika byla formulována v roce 1997 pro bankovní projekt o plánované délce 15 měsíců a lidskými zdroji o velikosti 50ti osob. Metodika zachovává proces řízení v mírně odlehčené formě, ale hlavně vyzdvihuje proces modelování při vývoji SW. Iterační cyklus může být kratší (od 2 dní) a může se prodlužovat (až do 2 týdnů) v závislosti na požadavcích projektu. [27]

3.5.2.5.1 Procesy

Procesy v metodice FDD dle Aleny Buchalcevové: *"FDD na rozdíl od ostatních agilních metodik popisuje postup při vývoji softwaru ve formě procesů. Význam procesů však nepřeceňuje, neboť cílem není splnění předepsaného procesu, ale vytvoření fungujícího softwaru splňujícího požadavky zákazníka. FDD definuje "lehké" procesy, každý z nich je popsán jen na 2 stránkách. Takto definovaný proces umožňuje škálovat metodiku i na větší projekty a rychleji zapojit nové zaměstnance. Popis procesu má být co nejkratší, doporučuje se použít vzor ETVX: Entry, Task, Verification, eXit, to znamená u každého procesu:*

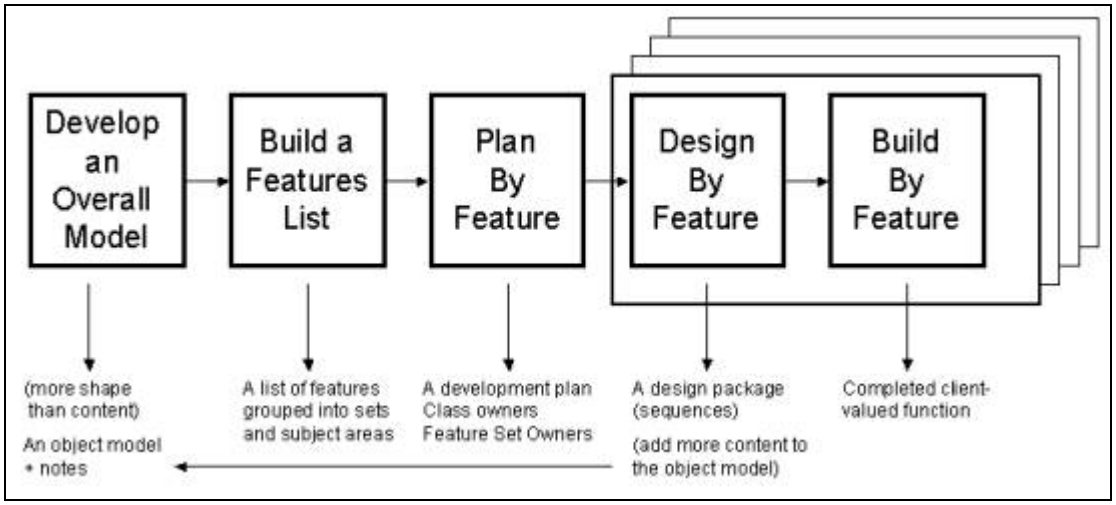
- *specifikovat jasně vstupní kritéria,*
- *vyjmenovat úkoly, každý úkol má - název, kdo se jej účastní, zda je povinný, popis*
- *úkolu,*
- *nástroje verifikace,*
- *výstupní podmínky procesu."* [29]

Konkrétními procesy jsou:

1. Vypracování celkového modelu
2. Sestavení seznamu užitných vlastností

3. Plánování užitné vlastnosti
4. Návrh užitné vlastnosti
5. Realizace užitné vlastnosti

Na níže uvedeném obrázku č. 17 jsou jednotlivé procesy vizualizovány.



Obrázek 17: Procesy metodiky FDD (Zdroj: [27]**)**

3.5.2.5.2 Praktiky používané FDD

Níže uvedený seznam osmi používaných praktik při vývoji SW s metodikou FDD. Metodika nedefinuje zcela nové praktiky a postupy při vývoji SW, ale vhodnou kombinací již známých postupů dává vývoji SW komplexní a fungující rámec. [29]

Tyto praktiky jsou:

1. **Doménový objektový model** se skládá zejména z diagramu tříd sekvenčních diagramů. V průběhu prací na projektu je rozšiřován a doplňován o nové objekty. Jednotný model má za cíl udržet přehled týmu o koncepci systému a má předcházet případným nedorozuměním.
2. **Vývoj podle užitných vlastností** znamená zaměření se na ty funkcionality systému, které mají pro zákazníka hodnotu. Zmíněná "užitá vlastnost", neboli feature, musí být realizovatelná v jedné délce definované iterace, jinak musí být funkčnost rozdělena na více částí.
3. **Vlastnictví kódu** (respektive části kódu - obvykle třídy) je v metodice FDD přisuzováno právě jednomu odpovědnému vývojáři, což je zásadní rozdíl oproti metodice XP (viz. Kapitola 3.5.2.6), kdy je naopak kód ve vlastnictví všech a je požadována plná zastupitelnost vývojářů.

4. **Týmy pro užitné vlastnosti** jsou menší jednotky o třech až šesti vývojářích, které implementují požadované feature. Každá užitá vlastnost ve většině případů vyžaduje interakci více programových tříd a každá z tříd má právě jednoho vlastníka kódu (zdrojového kódu). Právě tato vazba definuje tyto týmy.
5. **Inspekce** kódu neboli vzájemná kontrola mezi vývojáři snižuje počet chyb a obecně zlepšuje kvalitu kódu. Jak uvádí například Steve McConnell, tak většina vývojových studií zjistila, že inspekce kódu jsou levnější než testování hotového produktu. Dále uvádí, že studie NASA prokázala, že při čtení zdrojového kódu bylo odhaleno o 80% více chyb za hodinu než během testování. Níže uvedený obrázek č. 18 prezentuje procento chyb odhalených vybranými technikami. [28]

Způsob odstranění	Nejnižší poměr	Průměr	Nejvyšší poměr
Neformální revize návrhu	25 %	35 %	40 %
Formální inspekce návrhu	45 %	55 %	65 %
Neformální revize kódu	20 %	25 %	35 %
Formální inspekce kódu	45 %	60 %	70 %
Modelování nebo prototypování	35 %	65 %	80 %
Osobní ruční kontrola kódu	20 %	40 %	60 %
Testování jednotek	15 %	30 %	50 %
Testování nové funkce (komponenty)	20 %	30 %	35 %
Test integrace	25 %	35 %	40 %
Regresní testování	15 %	25 %	30 %
Test systému	25 %	40 %	55 %
Menší beta-testování (<10 pracovišť)	25 %	35 %	40 %
Rozsáhlé beta-testování (>1000 pracovišť)	60 %	75 %	85 %

Obrázek 18: Procento nalezených chyb dle techniky (Zdroj: [28]**)**

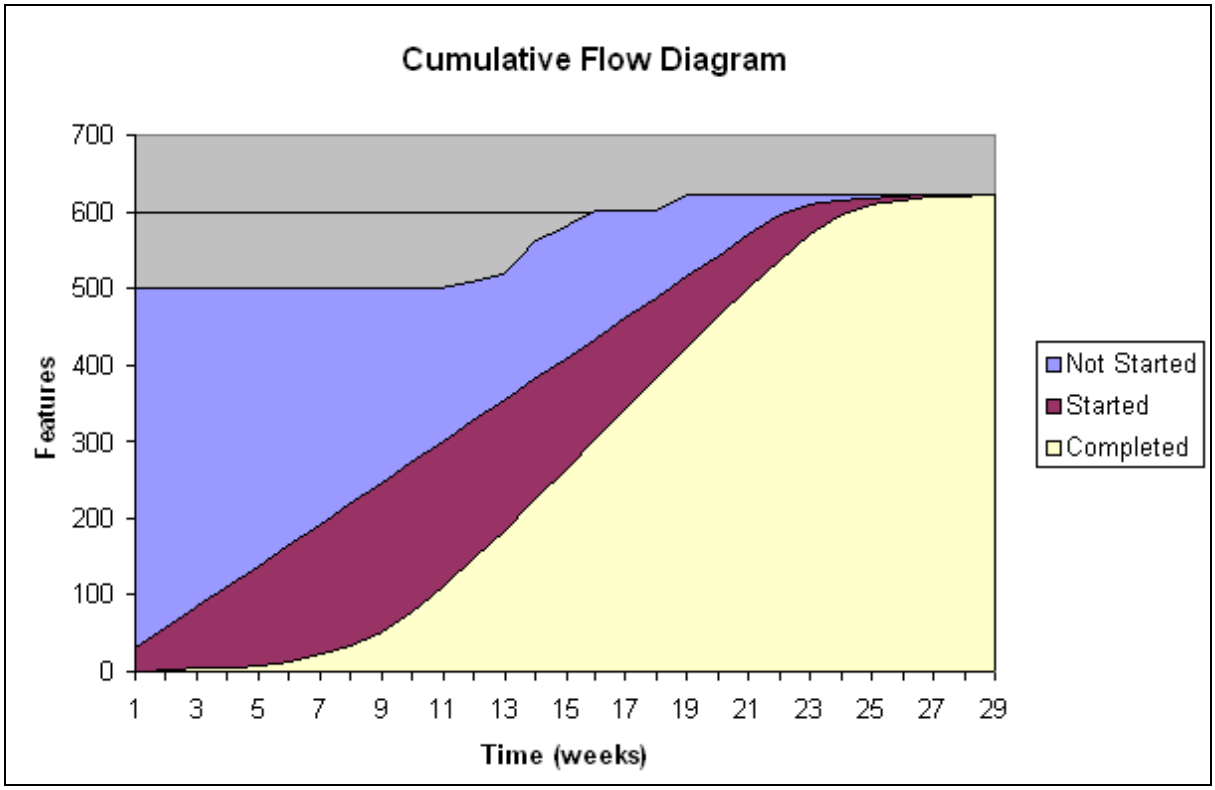
6. **Pravidelné buildy** (vydávání verzí aplikace) umožňují nejen častěji prezentovat výsledek práce zákazníkovi, ale také průběžně ověřují integraci a kompletní sestavení celého systému. Tendencí současné doby, nejen v rámci agilních metodik, je buildy dělat tak často, jak je jen možné, a tato činnost dostala i své označení: průběžná integrace (angl. Continuous Integration), kdy zdrojové kódy umístěné na serveru se ve stanovených intervalech automatickými scripty kompilují a spouští testy. Tím se pravidelné buildy generují automatizovaně a kontrola nad stavem nasaditelnosti produktu je v podstatě absolutní.

7. **Řízení konfigurací** bývá často vztahováno pouze na zdrojový kód, ale je velmi důležité uchovávat také verze systémové specifikace, protože je obvykle předmětem smlouvy mezi zákazníkem a dodavatelem a také podkladem pro testovací scénáře.
8. **Reporting výsledků**, tedy monitorování stavu prací na projektu, je nedílnou součástí každé metodiky. FDD se snaží činnosti spojené s monitorováním stavu projektu efektivně korigovat a nadbytečné množství schůzek a formálních dokumentů, což je častý jev v tradičních metodikách, redukuje na několik málo formátů zpráv informujících o stavu projektu.

3.5.2.5.3 Kumulativní vývojový diagram

Kumulativní vývojový diagram (angl. Cumulative Flow Diagram) je v metodice FDD velmi častým vizuálním nástrojem pro vyjádření stavu projektu. Diagram odpovídá na otázky: Kolik práce bylo odvedeno? Kolik prací právě probíhá? Kolik práce musí být ještě uděláno?

Diagram zobrazuje na ose x čas a na ose y počet požadovaných funkcionalit (v metodice FDD se jedná o features).



Obrázek 19: Kumulativní diagram (Zdroj: [29]**)**

3.5.2.5.4 Silné a slabé stránky metodiky

Pokud budeme metodiku FDD porovnávat s metodikou Scrum, narazíme na více společných hodnot, jako například spolupráce, důraz na kvalitu komponent a další, v podstatě obecné hodnoty agilních metodik. Rozdíl nalezneme například v tom, že FDD dává konkrétní inženýrské praktiky při návrhu a psaní SW oproti Scrumu, který toto nechává na vývojovém týmu. FDD bylo formulováno tak, že bere v úvahu přirozeně silné a slabé stránky lidí a snaží se z nich vytěžit nebo naopak je eliminovat. Dále bylo dokázáno, že FDD je vysoce efektivní při záchraně komplexních projektů, a to zejména díky jednotnému seznamu všech požadovaných a prioritizovaných funkcionalit a rychlému vývoji.

3.5.2.5.4.1 Silné stránky

- Jednoznačné vlastnictví programových tříd a implementovaných vlastností (features)
- Stručné procesy

3.5.2.5.4.2 Slabé stránky

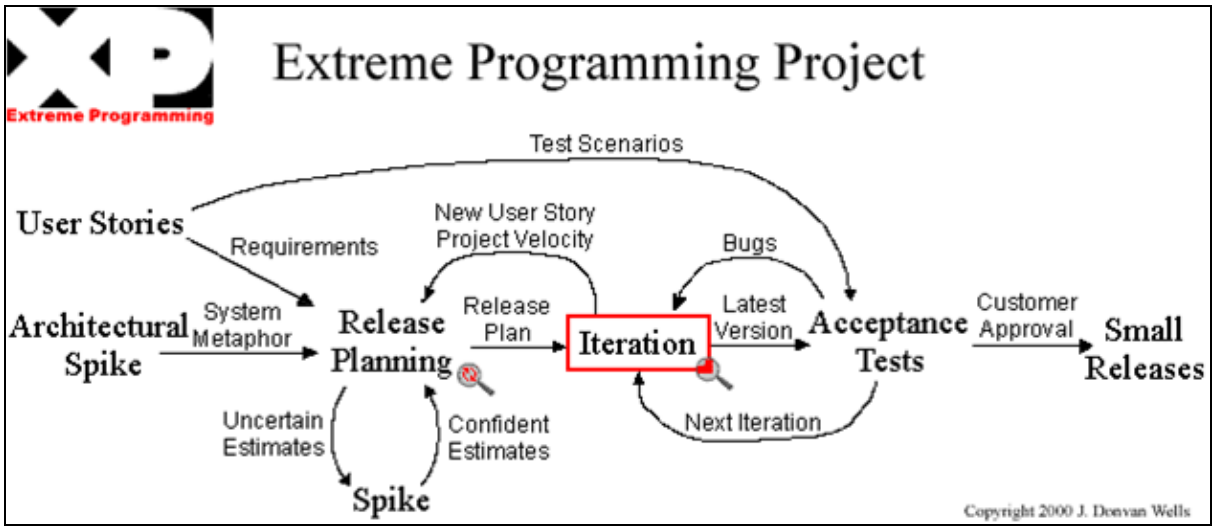
- Méně psané dokumentace (nahrazena komentovaným kódem)
- Velká zodpovědnost padá na SW architekty

3.5.2.6 Extrémní programování (XP)

Metodiku Extrémního programování navrhnul a uceleně popsal Kent Beck ve své knize "Extreme programming explained: embrace change". Sám autor metodiku charakterizuje následovně: *"XP je lehký, účinný, nepříliš rizikový, pružný, předvídatelný, vědecký a zábavný způsob, jak vyvíjet software."* [30]

Extrémní programování (angl. Extreme Programming) je agilní metodika, která je v současnosti vytlačována metodikou Scrum. Metodika se zaměřuje na praxí ověřené praktiky, které při vývoji SW fungují, a jejich aplikaci dotahuje do extrému, proto tedy extrémní programování. Metodika umožňuje měnit výsledný SW produkt i v pozdějších fázích vývoje. Základní hodnoty metodiky samozřejmě ctí agilní manifest a výrazně se podobají metodice Scrum. Sama metodika se tedy zaměřuje hlavně na jednoduchost, komunikaci a zpětnou vazbu od zákazníka. Tím, že metodika využívá vybrané praktiky na maximum, tak je i náročnější na komunikaci a spolupráci mezi jednotlivými členy týmu i na komunikaci a spolupráci se zákazníkem.

Níže uvedený obrázek vystihuje vývojový cyklus projektu za použití metodiky XP. [44]



Obrázek 20: Vývojový cyklus metodiky XP (Zdroj: [32]**)**

V uvedeném procesu je zajímavým pojmem, který si zaslouží bližší vysvětlení, termín "Spike". Jedná se o metodu řešící problém, kdy je po vývojáři požadován odhad pracnosti určité komponenty nebo obecně jakékoli funkcionality, avšak odhad je natolik nepřesný, že se na něj nedá spolehnout. V tomto případě se použije "Spike solution", kdy se dle zadání naprogramuje nejrychlejší možné řešení. Zdrojový kód nemusí implementovat detaily, u kterých není pochyb o jejich zvládnutí, jde o to, aby programátor prošel všemi rizikovými místy, a tím plně pochopil problematiku dané domény.

Po vytvoření "Spike" se naprogramovaný zdrojový kód zahodí a začne se programovat znovu, již dle všech standardů a detailně. Vývojář již ale zná problémy a dokonce je začal už řešit v rámci Spike a k problému tedy už přistupuje tak, že problematické místo pro něj není nové a může se vyvarovat případných chyb.

Metodika XP se skládá z 12 základních praktik:

1. **Business praktiky**
    a. Plánovací hra
    b. Zákazník na pracovišti
    c. Vydávání malých verzí
    d. Metafora
2. **Týmové praktiky**
    a. Párové programování
    b. Společné vlastnictví kódu

c. Standardy kódu
d. Udržitelné tempo

3. Programovací praktiky

a. Průběžná integrace
b. Jednoduchý návrh
c. Refaktorizace kódu
d. Testování

Jednou z hlavních praktik, která tuto metodiku odlišuje od ostatních agilních metodik, je tzv. "Párové programování". Základem této praktiky je, že dva programátoři společně pracují na určité funkcionalitě. Vlastní práce probíhá obvykle u jednoho PC, oba pracují na stejném zdrojovém kódu, po určité době se střídají, každý sleduje, ovlivňuje a doplňuje práci toho druhého. Tento postup výrazně zvyšuje kvalitu výsledného produktu. Vyšší kvalita kódu šetří čas v budoucnu, kdy by se nízká kvalita projevila při nákladné změně již hotových komponent nebo nutné refaktorizaci. Další výhodou je, že tato praktika podporuje "Společné vlastnictví kódu", žádný z vývojářů nebude tedy jediným, kdo zná určitou část systému, vždy existuje minimálně jeden další, který se na tomto kódu podílel také. A v neposlední řadě vývojáři mezi sebou sdílí znalosti, vzájemně se učí a zlepšují tedy své dovednosti. [24]

Nicméně pokud by zkušenosti a dovednosti dvou kooperujících programátorů byly výrazně na jiné úrovni, mohl by se méně zkušený programátor stát zcela nečinným a praktika by pozbyla smysl. Nebo pokud by k sobě konkrétní dva vývojáři chovali vyslovené antipatie, pak by tato technika byla určitě zcela nefunkční. Je tedy důležité, aby tato praktika byla využita ve správném prostředí.

3.5.2.6.1 Silné a slabé stránky metodiky

3.5.2.6.1.1 Silné stránky

- Ušetření nákladů díky eliminaci neproduktivních aktivit při vývoji SW
- Vysoká kvalita kódu díky párovému programování

3.5.2.6.1.2 Slabé stránky

- Velká náročnost na disciplínu vývojářů
- Zákazník nemusí akceptovat požadavek na jeho zapojení do projektu v požadované extrémní míře

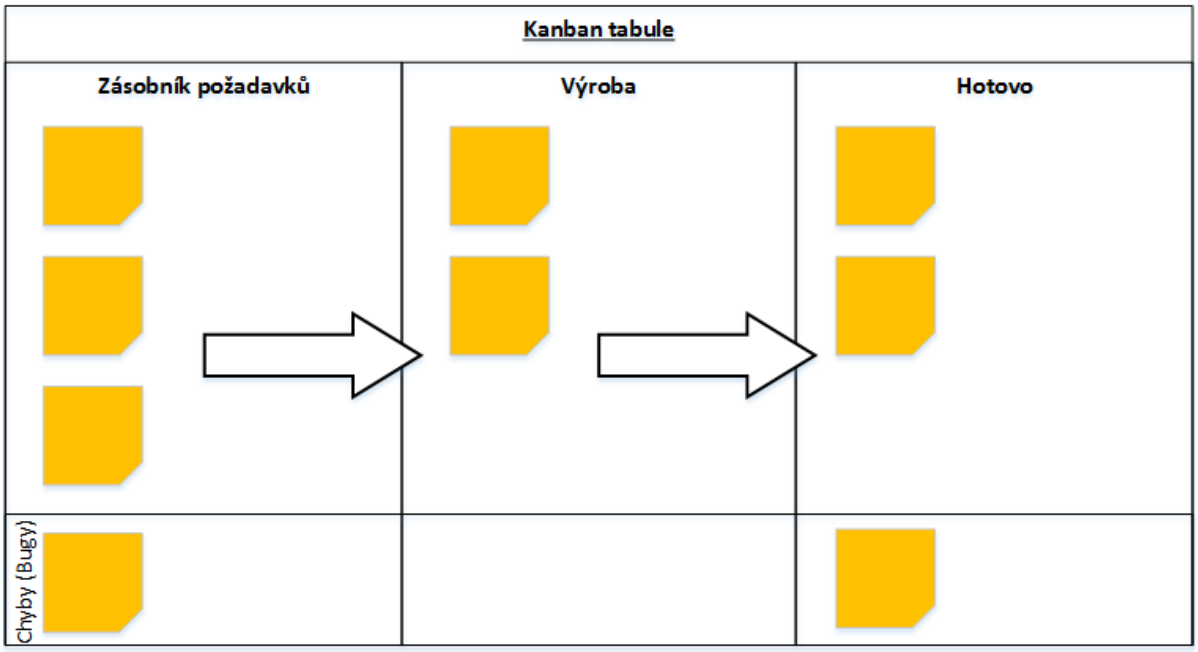
3.5.2.7 Kanban

Metodika Kanban je velice flexibilní, její pravidla jsou jedny z nejobecnějších a metodika v podstatě nic nenařizuje. Své uplatnění najde často v údržbových týmech, kde oprava chyb může trvat těžko predikovatelnou dobu a definice pevných iterací není účelná. Metodika staví pouze několika základních principech [45]:

1. **Omezení rozpracovaných úkolů**
    a. Omezení počtu rozpracovaných úkolů eliminuje "multitasking", který musí pracovníci zvládat pokud, mají rozpracováno více úkolů najednou. Souběžná práce na více úkolech nepřináší žádný pozitivní efekt, naopak je možné se dostat do stavu, kdy vše trvá nepřiměřeně dlouho a nic není hotovo. Při použití fyzické tabule pro reprezentaci Kanban tabule je nutné uvést toto omezení nad každý sloupec (stav našeho procesu), v případě podpůrného SW nástroje pro vizualizaci Kanban tabule je tento limit hlídán programově a více úkolů než je stanovený limit do konkrétního sloupce nezařadíme.
2. **Vizualizace postupu**
    a. Nejběžnějším přístupem k vizualizaci postupu je fyzická tabule rozdělená do sloupců, které reprezentují kroky našeho pracovního postupu (angl. workflow). Do jednotlivých sloupců jsou pak umisťovány post-it lístky, které reprezentují konkrétní úkoly. Přesouváním post-it lístků mezi sloupci posouváme úkol v námi zvoleném pracovním postupu vpřed.
3. **Minimalizace času průchodu**
    a. Jednou ze stěžejních veličin, kterou Kanban používá a měří, je "lead time", tedy čas, za jak dlouho se konkrétní úkol dostane z jednoho stavu do jiného, typicky z výchozího stavu (tedy ze "Zásobníku požadavků") do stavu "Hotovo". Kanban častěji pracuje s kumulovaným "lead time", tedy za jak dlouho projde pracovním postupem průměrný úkol.
4. **Explicitní procesní politiky**
    a. Explicitní definování procesních politik přinese ten benefit, že budeme schopni racionálně a objektivně diskutovat nad jednotlivými problémy a následně pracovat na jejich zlepšení. Příklady takových politik mohou být: definice "Hotového" úkolu; eskalační mechanizmus nebo zodpovědnost v jednotlivých procesních stavech.

3.5.2.7.1 Tabule

Kanban tabule odpovídá konkrétnímu procesu definovanému na projektu. Tabule tedy může mít dynamický počet sloupců. Dále bývá tabule často dělena nejen na vertikální dráhy (sloupce) ale i na horizontální dráhy, které mohou reprezentovat rozdělení úkolů dle jednotlivých vývojových týmů, případně dle typu úkolu, jak je znázorněno na obrázku níže. [33]



Obrázek 21: Jednoduchá Kanban tabule (Zdroj: Autor)

3.5.2.7.2 Silné a slabé stránky metodiky

Definovat silné a slabé stránky u této metodiky není zcela na místě. Metodika je velice jednoduchá (občas není ani nazývána metodikou), a to může být v určitém případě právě silná stránka a jindy zase slabá stránka, protože budeme muset metodiku sami upravit a rozhodnout se, jaké praktiky vývoje v ní budeme používat.

3.5.2.8 Škálování agilních metodik

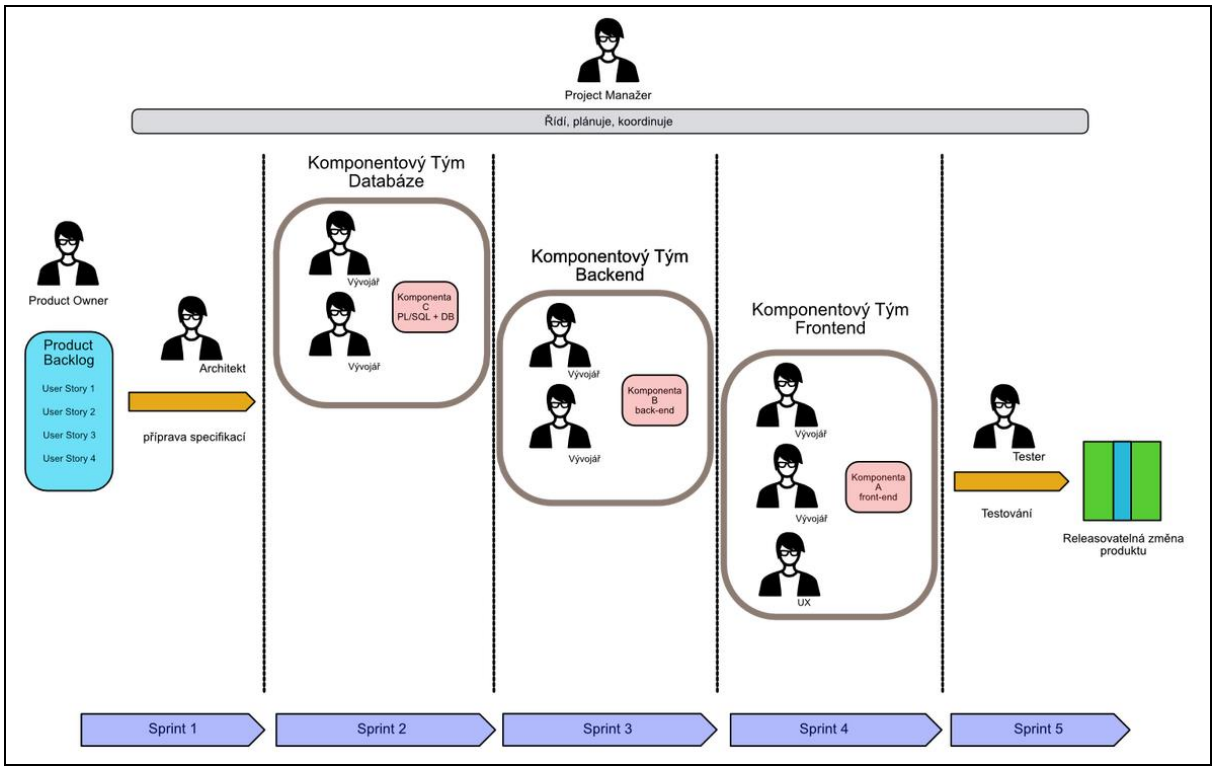
Častým a zažitým mýtem je tvrzení, že se agilní metodiky nehodí na velké projekty, ale jsou vhodné jen pro malé firmy a malé projekty. Tento názor pochází z dob, kdy rané agilní projekty byly malé nebo postavené nad metodikou XP. Dnes již víme, že byly již úspěšně realizovány agilní projekty s týmy přesahujícími 500 lidí. Je obecnou pravdou, že velké SW projekty mají vyšší míru selhání než projekty menšího rozsahu, a to bez ohledu na to, zda jsou vyvíjeny agilními nebo tradičními metodikami, jak je uvedeno v kapitole 3.3. Jednoduchým řešením může být rozdělení projektu na několik menších projektů

(subdodávek), které budou mít na starosti menší specializované týmy alokované na jednom místě a používající agilní metodiky. To je bohužel často v rozporu s dnešními organizačními a obchodními požadavky.

Agilní metodiky se snaží maximalizovat hodnotu, kterou produkt přinese zákazníkovi. Tvrzení, že se agilní metodiky nehodí na velké projekty nebo do velkých firem, je tedy nesprávné, agilní metodiky jsou vhodné pro každé prostředí a správnou otázkou tedy je, jak zavést agilní metodiky do prostředí těchto velkých firem?

Větší projekty s vyšší složitostí, velikostí a geografickou roztříštěností požadují dodatečnou personální strukturu pro zajištění jejich řízeného fungování, vyšší míru dokumentace a určité předem stanovené pevné procesy. To jsou charakteristiky, kterými mohou velké agilní projekty připomínat projekty tradiční, ale v základu projektu musí být agilní základy - tedy respektující Agilní manifest.

Jak již bylo zmíněno, zavedení agilních metodik vyžaduje určité změny, a ve velké organizaci jsou tyto změny vetší, a tedy i citelnější, mnohdy se musí měnit postupy a principy zažité i několik dekád. Jak uvádí Martin Pavlas ve svém článku "AGILNĚ VE VELKÝCH ORGANIZACÍCH", tak největší překážkou pro zavedení agilních metodik je právě organizační uspořádání vývojových týmů a neochota ono uspořádání změnit. Obvykle se jedná o týmy specializující se na vybrané technologie nebo komponenty systému, jak ukazuje níže uvedený obrázek. [35]



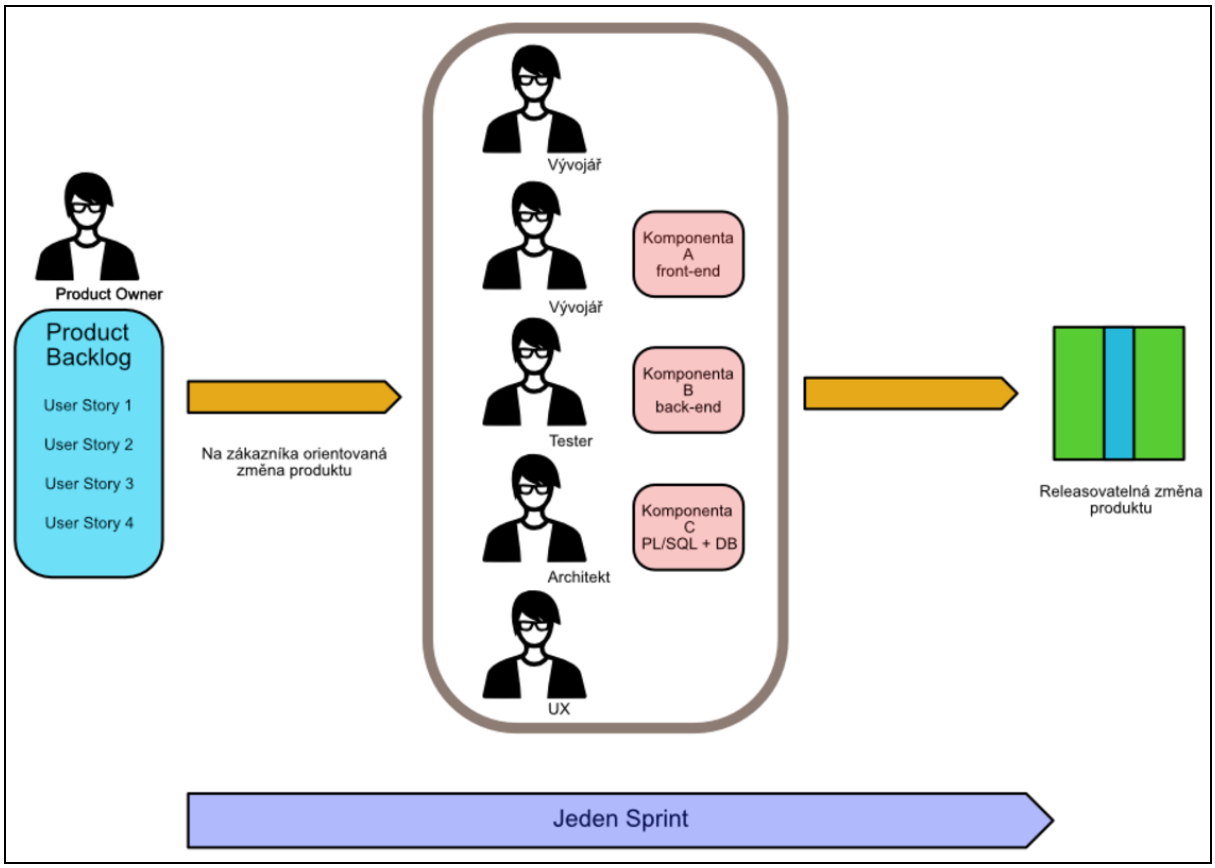
Obrázek 22: Komponentové týmy v agilním vývoji (Zdroj: [35]**)**

Toto uspořádání představuje zásadní problém, pokud firma požaduje implementaci funkcionalit iterativně a rychle - tedy agilně. Problémy představují zejména:

1. *"User Stories, které jsou orientované na doručení hodnoty zákazníkovi, se musí vždy rozpadnout do menších celků, a ty jsou zpracovány jednotlivými týmy. Tyto menší celky už samy o sobě ztrácí informaci o přínosu zákazníkovi a stávají se obyčejným projektem s technickým zadáním. Hodnota vývojářů jako kompetentních odborníků schopných hledat dobrá řešení pro danou potřebu zákazníka je výrazně nebo úplně eliminována."* [35]
2. *"Týmy nemohou user story jako takovou otestovat z hlediska její funkce a logiky. Takové testovaní se provádí až po integraci všech dílčích menších částí a to speciálním testovacím týmem podle testovacího plánu vytvořeného project managerem nebo testovacím specialistou."* [35]
3. *"Nečitelnost a složitost doručovaní rozdrobených User Stories může vést k ještě větší potřebě vše dlouhodobě plánovat. Tak se společnost místo posunu k větší akce-schopnosti začne obloukem vracet ke klasickému vodopádu."* [35]

Aby se výše popsaný scénář podařilo eliminovat, je třeba uvedené komponentové týmy rozbít a přetvořit je v takzvané “Feature týmy”, tedy týmy, které budou složeny ze specialistů napříč podnikovou infrastrukturou, s potřebnými dovednostmi a tak, aby byl nový tým schopný samostatně vyvinout požadovanou funkcionalitu od začátku až do konce (angl. end-to-end).

Níže uvedený obrázek ukazuje zpracování požadavku během jednoho release (vydání produktu), oproti obrázku č. 22, kde se stejný požadavek realizoval pět sprintů, pomocí komponentových týmů.



Obrázek 23: Feature tým (Zdroj: [35]**)**

Pro správné fungování je nutné dát Feature týmům volnost a ideálně je zbavit externích závislostí tak, aby byla dodávka funkcionality pouze v rukou týmu. Členové týmu by měli být stabilní a sladění v jeden dobře fungující tým. Dále je třeba se vyvarovat sdíleným specialistům napříč týmy; tato osoba bývá často přetížená a stává se “brzdou” týmů. V případě sdíleného programátora to většinou poukazuje na fakt, že určitý zdrojový kód není dobře napsaný nebo že vůbec není zdokumentovaný. Existence sdílených rolí by měla být hned od počátku eliminována.

Jednou z metodik, která se snaží zavést agilní metodiky do firemního prostředí, je **Large-Scale Scrum** (LeSS). LeSS staví na principech Scrumu a přináší metodický rámec pro aplikaci Scrumu pro větší organizace, velké produkty a více týmů.

Jeho stručné porovnání s metodikou Scrum uvádí Zuzana Šochová:

"Jediné, co se změnilo je, že na úrovni development týmů se zajímáme, co dělají ostatní týmy, jednotliví členové spolu řeší závislosti. Ne co se týče businessu, protože položky Backlogu jsou nezávislé, ale závislosti v kódu, dané konkrétním řešením dané Story. Dále obvykle posíláme zástupce týmů, aby chodili jako pozorovatelé na Standup ostatních týmů a identifikovali tak včas případné návaznosti a rizika. Na úrovni produktu už není jen na Product Ownerovi, aby před Sprintem vybral priority pro příští Sprint, ale obvykle se takového výběru zúčastní i zástupci týmů, aby zohlednili jejich různé zkušenosti. Nakonec je tu ještě jeden nový meeting – Overall Retrospective – jejímž cílem je udělat retrospektivu na téma jak nám jde spolupráce. A jak se asi dá očekávat, jsou tam ScrumMasteři, Product Owner, a zástupci týmů a koná se vždy po skončení Sprintu." [39, 46]

Další pohled na to, jak přizpůsobit agilní metodiky pro prostředí velkých firem, přináší Scott W. Ambler ve studii firmy IBM "The Agile Scaling Model (ASM): Adapting Agile Methods for Complex Environments". [46]

4 Vlastní práce

Vlastní práce se zabývá dvěma hlavními oblastmi. Nejprve popisuje zavedení konkrétní metodiky pro vývoj softwarových projektů v praxi. Dále porovnává podpůrné nástroje pro vývoj softwarových projektů, a to jak z hlediska funkčního, tak z hlediska ekonomického na základě předem stanovených kritérií.

4.1 Vzorový příklad užití konkrétní metodiky

V níže uvedené kapitole je uveden vzorový (v praxi úspěšně zavedený) příklad užití metodiky s agilním přístupem. Na projektu se autor práce osobně podílel. V textu práce nejsou z důvodu ochrany firemního know-how uvedeny konkrétní jména firem zadavatelů ani dodavatelů.

4.1.1 Užití agilní metodiky vývoje SW

Jedná se o příklad úspěšného zavedení agilní metodiky Scrum ve firmě působící na finančním trhu v České republice. V níže uvedených kapitolách je popsáno prostředí zadavatele a dodavatele. Autor práce působil ve firmě po dobu jednoho roku jako produktový manažer za stranu zadavatele. Ve firmě ovšem působil celkem déle než pět let na různých pozicích, což mu přineslo komplexní pohled na fungování podniku a firemních procesů. Tyto znalosti se později úspěšně uplatnily právě při zavádění nové metodiky pro vývoj SW.

4.1.1.1 Profil zadavatele

Jedná se o malou firmu (do 30 zaměstnanců včetně externích spolupracovníků) působící na českém finančním trhu od roku 2008, která se snaží být inovativní a nabízet finanční produkty jednoduše a atraktivně. K tomu využívá moderní informační a komunikační technologie. V roce 2013 proběhla akvizice společnosti působící na stejném trhu, kterou se firma výrazně rozrostla.

4.1.1.2 Profil dodavatele

Dodavatelem není v tomto konkrétním případě pouze jeden subjekt, ale ve stabilním složení jsou to subjekty tři. Jejich hlavním úkolem je kontinuální rozvoj a údržba firemních informačních a komunikačních systémů.

Jsou to zejména:

1. Externí dodavatel provizního systému
2. Externí dodavatel webového portálu
3. Interní vývojový tým

Dále jsou nárazově řešeny menší SW projekty, které realizují jednorázově najatí externí spolupracovníci. Případné větší projekty se zásahem do více firemních systému je nutno řešit se všemi dodavateli a práce je nutno sladit tak, aby probíhaly dle očekávání a dle naplánovaných harmonogramů všech zúčastněných stran.

Dodavatelem je tedy jak interní oddělení zadavatelské firmy, tak externí subjekty.

4.1.1.3 Popis problému

Před spojením výše uvedených společností byla jejich organizační struktura jednoduchá a přímočará. Interní vývojové týmy se skládaly z dvou až tří jedinců a složitost systému byla na úrovni, kdy o ni mělo přehled i vedení firmy, které se do rozvoje IS aktivně zapojovalo a rozhodovalo i o technických detailech, které by prakticky ani nemuselo řešit.

Potřebu systematického řízení vývoje SW projektů pocítily obě firmy ještě před jejich sloučení, během posledních 4 let byl jejich meziroční růst tržeb kolem 50%. Po akvizici firem bylo nutno v oblasti IS provést jejich konsolidaci a obecně složitost systémů narostla. Nové vedení firmy již neznalo přebírané systémy tak dobře jako původní vedení a rovněž se sloučily dva vývojové, dva marketingové a dva obchodní týmy, které se doposud neznaly. Každý vedoucí člen konkrétního týmu měl nové požadavky, které se snažil prosadit, aby jeho oddělení pracovalo lépe a aby měl s prací ohledně akvizice firmy co nejméně nové práce.

Z toho během krátké doby vyplynulo hned několik problémů, které bylo nutno okamžitě řešit, aby byly obchodní požadavky na rozvoj IS realizovány dle očekávání včas a v požadované kvalitě.

Hlavní identifikované problémy odhalené v prvním měsíci fungování obou firem:

1. Požadavky vedení na zásadní změny IS bez znalosti dopadů a rozsahu prací
2. Očekávání termínů IT dodávek jsou v nesouladu s realitou
3. Větší množství požadavků, kde všechny mají nejvyšší prioritu. Zadavatelé požadavků se předhánějí, čí požadavek je důležitější.

Ani v jedné firmě dříve, před sloučením, neexistoval systém pro řízení projektů nebo metodika pro vývoj SW. Vše se díky menšímu rozsahu firem dařilo relativně efektivně řídit za pomoci zkušeností a dovedností řídících pracovníků a s podpůrnými SW nástroji jako je Microsoft Excel nebo LibreOffice. V novém firemním prostředí byl tento přístup k věci nedostatečný, a proto byl v této oblasti identifikován závažný problém.

4.1.1.4 Výběr vhodné metodiky

Vzhledem k dynamičnosti firmy, která musí rychle reagovat na změny a trendy na finančním trhu a musí být konkurenceschopná, bylo od samého počátku identifikování problému diskutováno použití agilních metodik pro vývoj SW a řízení chodu a distribuce práce pro IT oddělení.

I přes osobní zkušenost některých pracovníků s metodikou Scrum bylo vedením požadováno, nenechat výběr metodiky pouze na osobních preferencích pracovníků, ale přistoupit k výběru metodiky systematicky. Výběr vhodné metodiky byl ověřen pomocí metody METES popsané v kapitole 3.4.2.1.

Váhy jednotlivých výběrových kritérií byly ponechány tak, jak je definuje metoda METES. Optimální hodnoty výběrových kritérií pro jednotlivé metodiky (RUP, Open UP, FDD, Scrum, XP a MSF CMMI) byly rovněž ponechány podle metody METES. Ohodnocení jednotlivých kritérií je pro každý projekt zcela individuální a není možno určit univerzální hodnoty, které lze použít. Stupnice hodnocení pro každé uvedené kritérium je uvedena v příloze I.

V níže uvedené tabulce č. 6, je uvedeno hodnocení projektových výběrových kritérií, které bylo stanoveno na workshopu, kterého se zúčastnilo nové vedení firmy a IT oddělení, které workshop iniciovalo. Každé kritérium bylo prodiskutováno a zástupcům vedení firmy vysvětleno tak, aby jeho význam plně pochopili. Bylo jednáno s vědomím, že se jedná se o klíčové rozhodnutí, na jehož základě bude vybrána vhodná metodika, a tedy každá budoucí změna metodiky bude pro zaměstnance a spolupracovníky firmy náročná a pro firmu samotnou také nákladná.

Tabulka 6: Projektová výběrová kritéria (Zdroj: Autor)

Klíčové kritérium	Hodnota
Kritičnost projektu	3
Délka projektu	4
Dostupnost zákazníka	0
Velikost týmu	2
Geografické rozložení týmu	2
Stabilita požadavků	0
Znovupoužitelnost	0
Velikost projektu	3
Nedostatek manažerských zkušeností	0
Nedostatek kvalifikace týmu	0
Nedostatek motivace týmu	0

Na základě výše stanovených hodnot byl vypočítán vážený součet vzdáleností od optimálního řešení pro každou z vybraných metodik.

Výsledky výpočtu jsou uvedeny v tabulce č. 7. Jako optimální metodika byla ve výsledku vypočtena metodika Scrum, která byla vedením i ostatními pracovníky na základě tohoto ověření metodou METES akceptována.

Tabulka 7: Vážený součet vzdáleností od optimálního řešení (Zdroj: Autor)

Metodika	RUP	Open UP	FDD	Scrum	XP	CMMI
Vážený součet vzdáleností od optimálních hodnot	2,583	1,584	1,078	**0,596**	0,651	2,604

4.1.1.5 Zavedení metodiky

Před samotným zavedením metodiky proběhla analýza a kompletace všech dosavadních neuzavřených požadavků na vývoj (viz následující tabulka):

Tabulka 8: Stav požadavků před zavedením metodiky (Zdroj: Autor)

Status požadavku	Urgentní	Celkem
Ve frontě	21	35
Rozpracované	10	15
- Ve zpoždění oproti plánu	6	7

Z tabulky vyplývá, že velká většina požadavků, na kterých se pracovalo (stav "Rozpracované") byly označeny jako urgentní. Takovýto stav nebyl a není v žádném případě žádoucí. Byl zapříčiněn častými změnami priorit požadavků, v některých případech i rivalitou jednotlivých oddělení, která se "předbíhala" s tím, co je prioritou, a v mnoha případech zadávající vůbec neznali požadavky ostatních oddělení firmy na rozvoj IS. IT oddělení bylo ovšem pouze jedno, požadavky se zde hromadily, všechny s označením "urgentní". Vedení firmy potřebovalo provádět rychlá a jasná rozhodnutí s předem definovanými kompetencemi, zejména kdo může určit prioritu požadavku a kdo může stanovit datum realizace požadavku.

Tato situace vede k prvnímu kroku, který je nutný pro úspěšné zavedení metodiky, a tím je výběr vhodného podpůrného nástroje pro agilní vývoj SW. Nástroj by měl sloužit jako databáze všech požadavků s přehlednými tabulemi (angl. dashboards; agile boards), kde bude jasně graficky prezentováno, jakou prioritu má konkrétní požadavek vzhledem k ostatním požadavkům a kdy je očekávané datum jeho realizace. Do tohoto podpůrného nástroje by měli mít přístup všichni zaměstnanci, včetně externích (s možnými omezenými právy).

Volba požadovaných kritérií, která musí podpůrný nástroj splňovat, a samotná metoda výběru nástroje je detailně rozepsána v kapitole 4.1.1.4.

4.1.1.5.1 Příprava před zavedením metodiky

4.1.1.5.1.1 Role

Před celofiremním přechodem na novou metodiku bylo nutno všechny pracovníky seznámit s jejich pravomocemi a rolemi. Rozdělení rolí bylo v prostředí firmy vcelku jednoznačné. Roli **Product Owner** má dle doporučení metodiky Scrum zastávat jedna osoba, proto tato role byla bez pochyb přiřazena majiteli firmy, který do té doby firmu aktivně vedl.

Role **Scrum Master** byla přiřazena dosavadnímu manažerovi, tedy celého IT oddělení. Ten měl s metodikou Scrum praktickou zkušenost a také velký respekt jak mezi svými podřízenými, tak mezi vedoucími pracovníky ostatních firemních oddělení.

Development Team tvořila skupina 2 vývojářů, 3 analytiků a dvou externích firem začleněných do nového agilního vývoje SW. Tento tým se navzájem již dobře znal a byl složen o pracovníků, kteří ve firmě působili nejdéle.

Roli zákazníka, který definuje své potřeby, zastupují jak řadoví zaměstnanci firmy, tak stakeholdeři, tedy vedoucí pracovníci různých oddělení, kteří zodpovídají za určitý finanční rozpočet, včetně samotného IT oddělení, tedy Development teamu.

Délka sprintu (časové jednotky splnění sady úkolů) byla stanovena na 2 týdny. Kratší sprinty by znamenaly velice malé přírůstky hodnoty produktu, a delší sprinty zase netransparentní proces a pro obchodní oddělení neakceptovatelné prodlení při vydávání nových funkcionalit.

4.1.1.5.2 Vývojový cyklus

4.1.1.5.2.1 První plánovací a vývojová iterace (Sprint #1)

První sprint je vždy tím nejnáročnějším pro všechny zainteresované subjekty. Průběh prvního sprintu je třeba dobře naplánovat, sledovat jeho průběh a v následné retrospektivě se poučit, aby další sprint proběhl již bez zjištěných rizik a problémů.

Scrum master a vývojový tým v našem případě identifikoval dvě hlavní činnosti jako klíčové před zahájením sprintu:

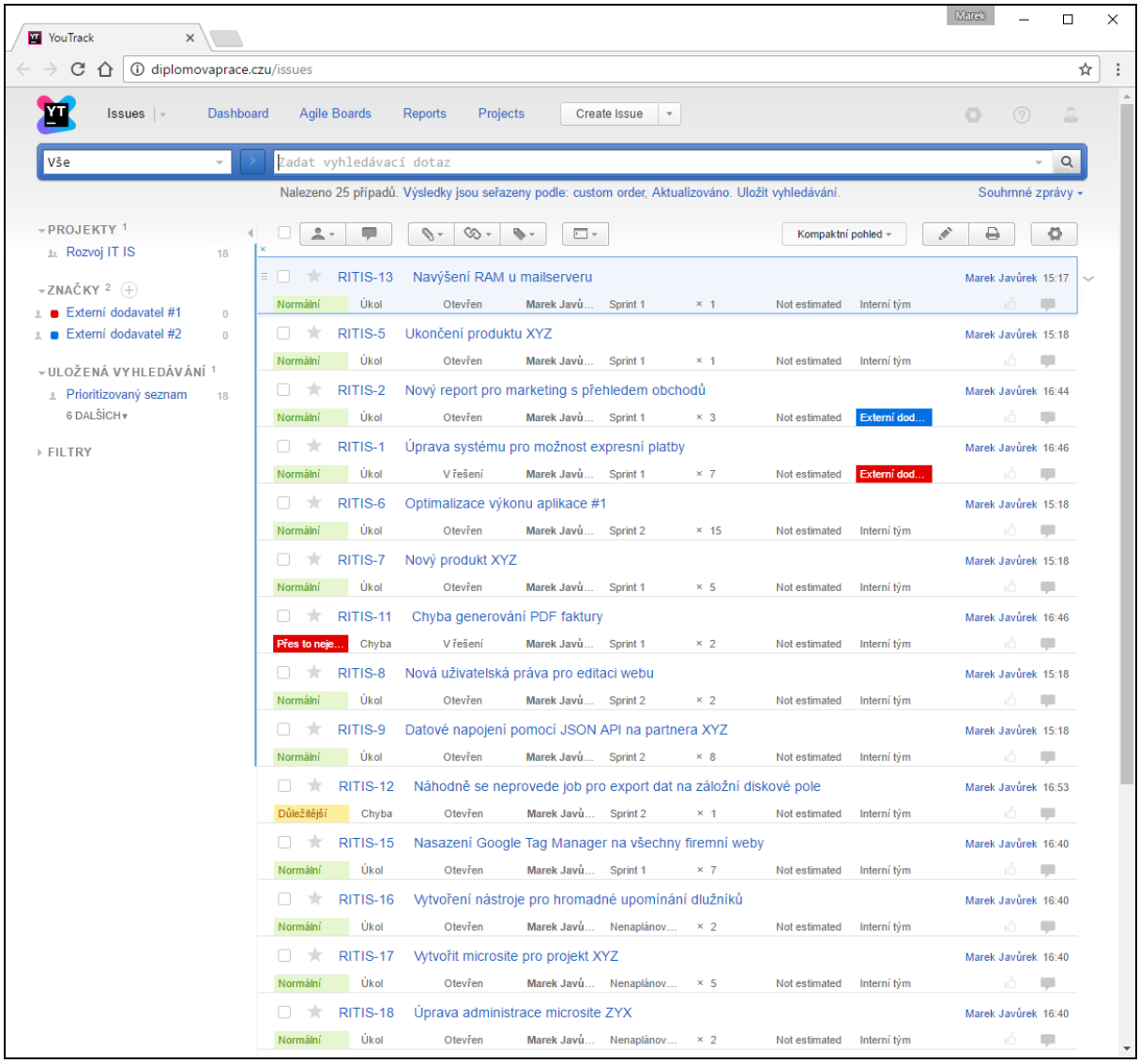
1. Revize současných požadavků
2. Realistické odhadování časové náročnosti požadavků

Revize současných požadavků proběhla v součinnosti s vedením firmy. Jak uvádí tabulka č. 8, vývojový tým eviduje celkem 50 požadavků, některé z nich jsou pouze ve stavu “Ve frontě”, tedy dosud neřešené, jiné jsou již rozpracované. Po revizi všech požadavků jich zůstalo k řešení pouze 25, realizaci ostatních vedení firmy zrušilo. Realizace většiny vyřazených požadavků byla shledána jako neefektivní nebo nepřínosná a priority byly kladeny na jiné funkce, které přinesou větší hodnotu.

Realistické odhadování časové náročnosti požadavků je obtížnou disciplínou. Žádný z vývojářů nemůže definitivně určit náročnost implementace daného konkrétního požadavku předem. Nezřídka se stává, že časovou náročnost musí pod tlakem odhadovat analytik nebo určitý vedoucí pracovník, a samotný vývojáře je pak následně nucen, aby se do navrženého časového rámce vešel, i když to může být často nereálné. Toto může mít negativní dopad na kvalitu kódu a v konečném důsledku zvýšit celkové náklady nad původní plán. Pro účel efektivního odhadování časové náročnosti splnění jednotlivých požadavků byla zvolena metoda “Planning Poker”, ke které jsou třeba plánovací karty uvedené na obrázku č. 24.

Obrázek 24: Plánovací karty (Zdroj: [37]**)**

Metoda spočívá v tom, že se členům vývojového týmu (v tomto případě vývojářům, analytikům i Scrum masterovi) rozdají karty s hodnotami 0, 1/2, 1, 2, 3, 5, 8, 13, 20, 40, "?" (neznámá náročnost), "kafe" (náročnost menší než 1) a "∞" (náročnost úkolu tak velká, že je nutné úkol rozdělit na více dílčích úkolů). Tyto hodnoty reprezentují body určující relativní složitost k referenčnímu úkolu se známou hodinovou náročností (angl. Story points), počet dní nebo například počet hodin, nutných k realizaci odhadovaného požadavku. Metodika Scrum doporučuje neodhadovat náročnost exaktně v hodinách nebo dnech, ale právě ve zmíněných bodech. Nicméně všichni vývojáři v našem sledovaném vývojovém týmu měli odbornou praxi v délce pět až třináct let a bylo rozhodnuto pro používání hodnot v řádu dnů. Postup je tedy takový, že každý z členů týmu si vybere z rozdaných karet jednu podle svého vlastního uvážení a následně ji všichni společně zveřejní. Výběr tedy není ovlivněn rozhodnutím ostatních. Pokud jsou hodnoty všech vyložených karet stejné, použije se vybraný odhad, v opačném případně započne diskuze, která má za cíl vyjasnit, proč se odhady liší a jaký má kdo na danou problematiku názor. Tato metoda postupně naučí přesněji odhadovat časovou náročnost jednotlivých činností i pracovníky, kteří sami neprogramují. Pokud je úkol špatně časově odhadnut a jeho skutečná realizace převyšuje plánovaný odhad, je to zároveň chybou celého týmu, nikoli pouze dotyčného vývojáře implementujícího danou funkčnost.



Obrázek 25: Product backlog (Zdroj: Autor)

Před zahájením samotného sprintu je nezbytné ještě požadavky **prioritizovat** a **určit kapacitu vývojového týmu**. Prioritizaci požadavků má dle metodiky Scrum na starosti **Product owner**, v tomto případě majitel firmy. Ten na základě aktuálních cílů firmy prioritizoval seznam zadaných úkolů. Ze zadaného seznamu bude naplánován první sprint tak, že se budou do sprintu zařazovat požadavky do té doby, než bude naplněna kapacita tohoto sprintu. V interním týmu jsou 2 vývojáři, teoreticky je tedy možno na jeden sprint naplánovat práci v objemu 20 dní (pozn.: 1 den = 8 hodin). Prakticky je ovšem nutno mít vždy časovou rezervu, do které typicky spadají níže identifikované činnosti:

- **Oprava chyb**
  - Počet chyb není možné přesně odhadnout, a tudíž velikost rezervy exaktně naplánovat, nicméně můžeme vycházet z průměrného počtu chyb za období 2

týdnů a průměrné doby opravy chyby. Objem času potřebný pro opravu chyb bude nutné vyhodnocovat po každém sprintu, a tím odhad do dalších sprintů vždy upřesňovat.

- **Konzultační činnost**
  - Zejména analytici často potřebují při přípravě User stories konzultovat technické možnosti a omezení daného programovacího jazyka s programátory.
  - Programátory bývá tato aktivita vnímána jako vyrušování či obtěžování, nicméně je nedílnou součástí vývojového procesu. Analytik, který by navrhoval funkčnosti, které nejsou v souladu s architekturními základy projektu, nebo grafické komponenty, které nejsou součástí používaného frameworku, a které je nutno je od základu programovat, by v důsledku způsobil, že projekt by neúměrně prodražil a zadavatel by byl nespokojen a mohl celou realizaci odmítnout.

Těmto dvěma činnostem byl přiřazen rámec 20% z celého sprintu, tím tedy ve sprintu zbývá 16 dní čistého času práce vývojářů. Celkové kapacity interního týmu a externích dodavatelů jsou uvedeny v tabulce níže.

Tabulka 9: Celkové kapacity interního týmu a externích dodavatelů (Zdroj: Autor)

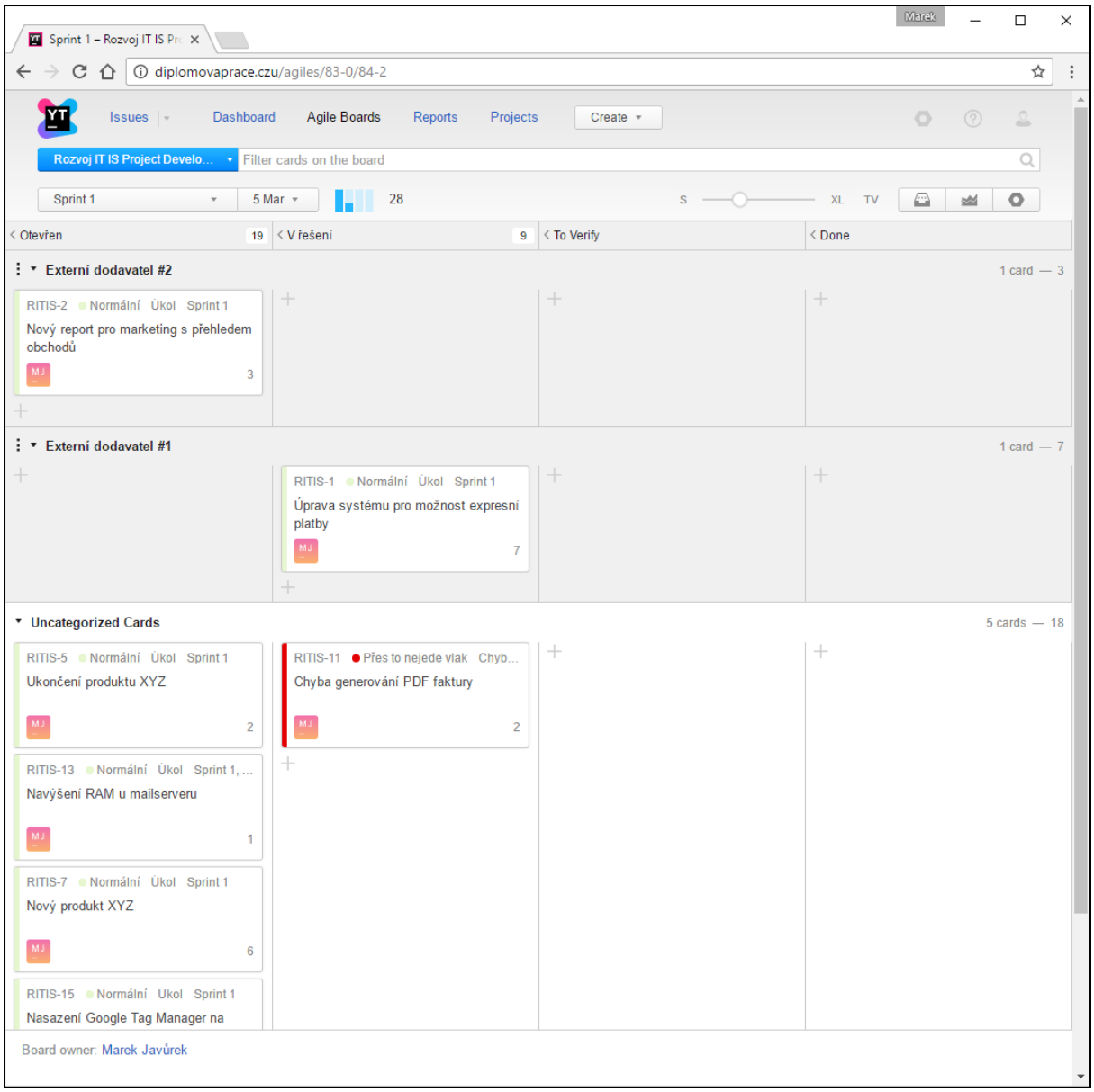
Subjekt	Celková kapacita sprintu (MD)	Rezerva pro případnou opravu chyb (%)	Výsledná možnost alokace (MD)
Interní vývojový tým	20	0,2	16
Externí dodavatel #1	40	0,1	36
Externí dodavatel #2	100	0	100

Dle priorit byl sestaven plán prvního sprintu uvedený na obrázku č. 26 níže. Z rozložení hodin vyplývá, že externí dodavatelé jsou vytíženi minimálně. Interní vývojový tým má nové úkoly v objemu 16 MD (Man Day = člověkoden) a dále nově rozpracovanou opravu jedné chyby, které je odhadnuta na 2 MD. Celkově tedy plán odpovídá přesně vypočítané kapacitě týmu. Plán byl finálně potvrzen vedením a následující den začal vývoj dle nové metodiky.

Tabulka 10: Vytížení vývojových týmu v 1. sprintu (Zdroj: Autor)

Subjekt	Výsledná možnost alokace (MD)	Vytížení (%)
Interní vývojový tým	16	1
Externí dodavatel #1	36	0,19
Externí dodavatel #2	100	0,03

Grafická reprezentace Scrum tabule v nástroji YouTrack vypadala následovně:



Obrázek 26: Plán prvního sprintu v podpůrném nástroji YouTrack (Zdroj: Autor)

Průběh sprintu

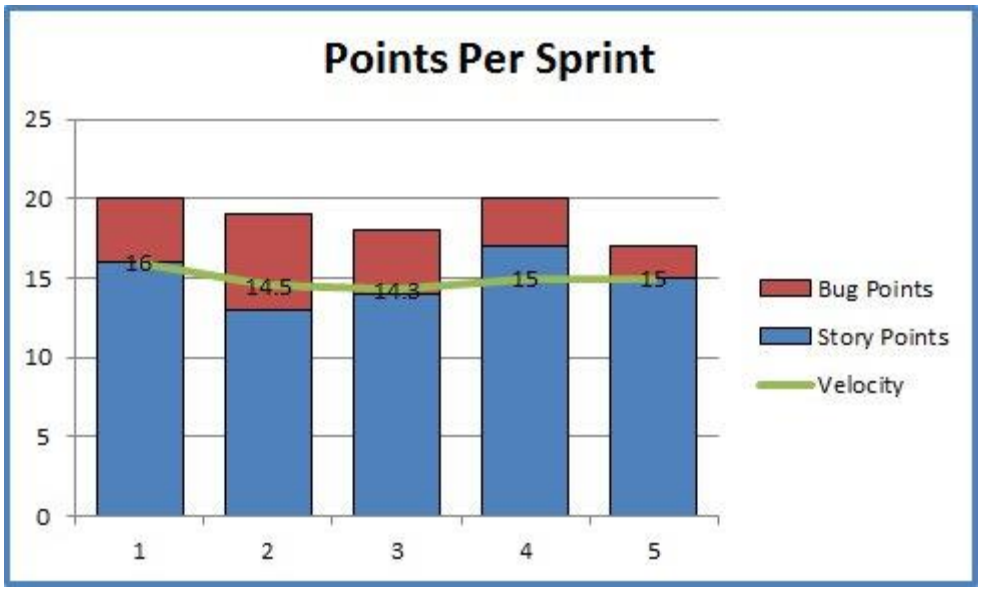
Během sprintu probíhaly každý den ráno v 9:00 pravidelné schůzky nazývané **Stand-up meeting**, na kterých byla povinná účast vývojového týmu. V průběhu prvního sprintu nebylo nutné řešit žádné kritické zásahy do metodiky nebo do jejího fungování, jako například zrušení sprintu. Tým programátorů a analytiků se však potýkal s takovými požadavky, které měly za cíl novou metodiku obejít a docílit tím okamžité implementace. I když se jednalo o zdánlivě jednoduché úkoly, mohly by se v průběhu řešení obrátit ve složitější, a tím zcela narušit hladký průběh sprintu. Díky správné činnosti Scrum mastera byly všechny tyto požadavky zachyceny, jejich okamžitá realizace byla odmítnuta, byly v souladu s metodikou zařazeny do fronty (Backlogu), analyzovány a předány k prioritizaci. Některá oddělení, zejména jejich vedoucí pracovníci, s tímto odložením nebyli zcela spokojeni, nicméně vrcholné vedení společnosti a Product Owner podpořilo vývojový tým a Scrum mastera v dodržování pravidel nové metodiky.

Zhodnocení sprintu

Zhodnocení sprintu a spolupráce, nazýváno také jako **Sprint review**, Review nebo Retrospektiva, je povinnou součástí metodiky Scrum. Jedná se o schůzku, které se účastní zejména vývojový tým se Scrum masterem a volitelně i zástupce zákazníka. Schůzka se obvykle koná následující den ráno po ukončení sprintu.

První zhodnocení přineslo zajímavé poznatky a důležité informace - z celkových 7 položek (z toho 5 položek na interní vývojový tým) sprint backlogu v celkovém objemu 26 plánovaných dní (16 dní pro interní vývojový tým) bylo realizováno pouze 6 položek. Nerealizovaná funkcionalita, určená pro interní vývojový tým, nebyla dodána kvůli zpoždění dodávky externího dodavatele a navýšenému počtu nahlášených chyb ke konci sprintu.

Pro optimální naplánování dalšího sprintu definuje metodika Scrum proměnnou rychlost (angl. Velocity), která může být počítána souhrnně, tedy za celý vývojový tým, nebo individuálně, pro každého vývojáře zvlášť. Vzhledem ke srovnatelné výkonnosti obou vývojářů v našem sledovaném týmu bude tento ukazatel stanoven vždy za celý tým, tedy souhrnně. Níže je uveden exemplární průběh této veličiny v průběhu 5 sprintů na virtuálním projektu.



Obrázek 27: Rychlost (Velocity) Scrum týmu v průběhu pěti sprintů (Zdroj: [38]**)**

Jak vyplývá z uvedeného obrázku, rychlost se v každém sprintu mění a dlouhodobým cílem je tuto veličinu stabilizovat tak, aby bylo možné objem práce budoucích sprintů plánovat co nejpřesněji, aby byla maximálně využita kapacita týmu a zároveň aby nebyl tým nepřetěžován.

Na začátku prvního sprintu byla stanovena kapacita sprintu na 16 dní, tedy rychlost = 16. Nerealizované požadavky byly v objemu 1 den. Nová hodnota veličiny rychlost bude pro další sprint 15 (tedy 15 dní).

Uzavření sprintu

Další uskutečněnou schůzkou bylo předvedení splněných úkolů zákazníkovi (angl. Review). Schůzky se účastnil jak vývojový tým, tak i zákazník, tedy jednotlivá oddělení firmy, která měla v backlogu daného sprintu zadané požadavky k realizaci.

Schůzka (Review) byla naplánovaná vždy v ten samý den jako schůzka Zhodnocení sprintu (Retrospektiva) a vytvořená funkcionalita byla v reálném čase prezentována zákazníkovi v prostředí testovací aplikace. Hlavním účelem schůzky je ukázka finální funkčnosti před nasazením do produkčního prostředí. Neméně důležitá je i role motivační, kdy zákazník vidí skutečný výsledek a ví, že vývojový tým, který realizuje jeho požadavky, je skutečně funkčním oddělením. Tato forma se v praxi velmi osvědčila a obecně zlepšila celkový pohled firmy na vývojový tým.

Obecně bylo uzavření prvního sprintu vnímáno jako úspěšné. Zákazník byl spokojen, což mělo pozitivní dopad i na motivaci vývojového týmu k dalšímu postupu a zdokonalování.

Obě schůzky byly uskutečněny v jediný den, a to v den po konci sprintu a zároveň v den před začátkem dalšího sprintu. Toto jednodenní okno umožnilo všem zúčastněným absolvovat obě schůzky bez nátlaku a strachu ze zpoždění prací a zároveň to umožnilo týmu pečlivě naplánovat další sprint. Hlavním cílem bylo poučit se z chyb z předešlého sprintu, chyby eliminovat a naopak praktiky, které se osvědčily, používat nadále.

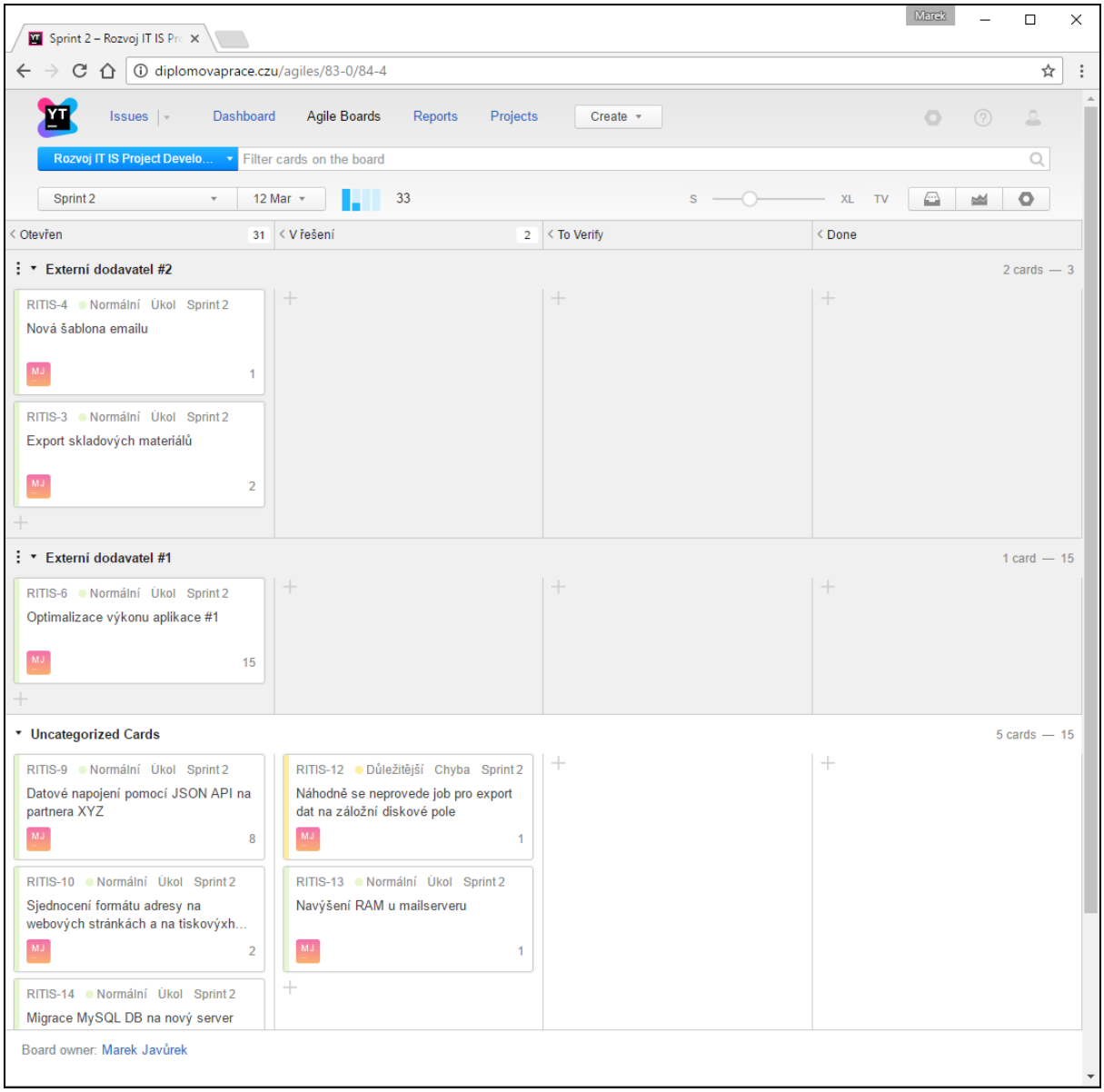
4.1.1.5.2.2 Druhá plánovací a vývojová iterace (Sprint #2)

Druhý sprint začíná hned následující den po plánovacím dni. Plán pro druhý sprint byl sestaven ve spolupráci s Vlastníkem produktu (Product owner). Kapacita sprintu pro interní vývojový tým byla snížena dle vypočítané rychlosti týmu, která se snížila z hodnoty 16 na hodnotu 15. Zároveň tento sprint přinesl výrazný nárůst objemu požadavků na jednoho z externích dodavatelů. Výsledné vytížení všech vývojových týmů je souhrnně znázorněno v níže uvedené tabulce:

Tabulka 11: Vytížení vývojových týmu v 2. sprintu (Zdroj: Autor)

Subjekt	Možnost alokace (MD)	Alokace pro sprint (MD)	Vytížení (%)
Interní vývojový tým	15	15	100%
Externí dodavatel #1	36	15	69%
Externí dodavatel #2	100	3	3%

Grafická reprezentace Scrum tabule v nástroji YouTrack vypadala následovně:



Obrázek 28: Plán druhého sprintu v podpůrném nástroji YouTrack (Zdroj: Autor)

Průběh sprintu

Vzhledem vyššímu objemu požadavků na externí dodavatelskou firmu, která se také účastní agilního vývoje se stejnými pravidly jako interní vývojový tým, ale nenachází se na stejné lokalitě (pouze ve stejném městě), bylo nutné vyřešit formu každodenních Standup meetingů, které byly v prvním sprintu ve vztahu k externím dodavatelům zcela vynechány.

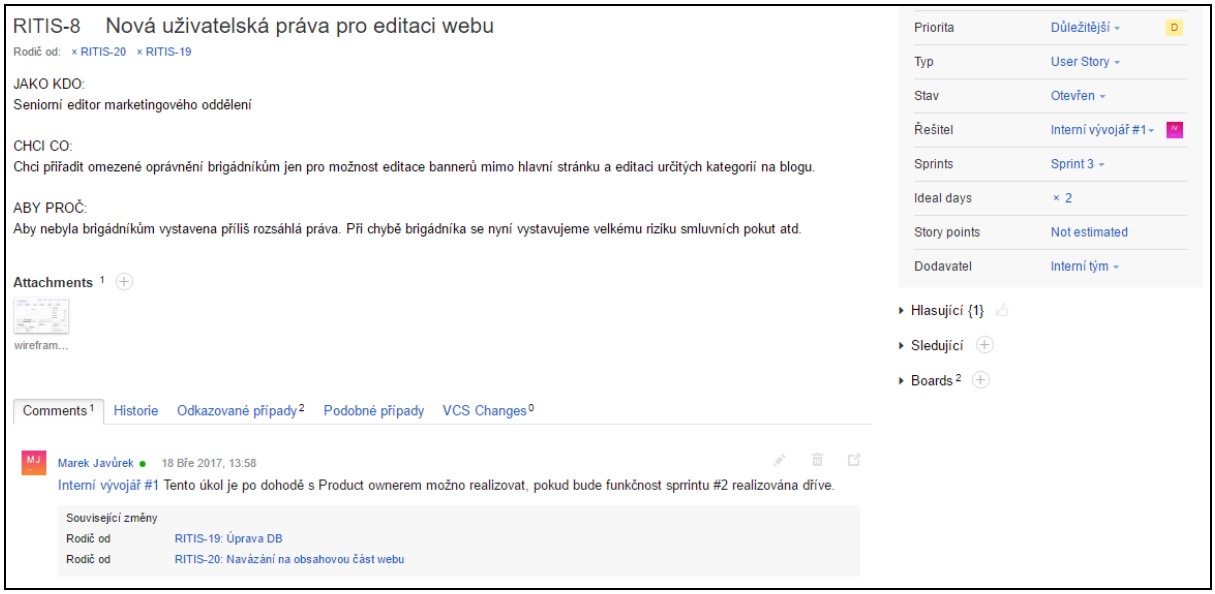
Dodavatel, respektive konkrétně jeho vývojáři mají stejné dotazy a problémy jako interní vývojáři a je nutno je konzultovat a řešit rychle, nejlépe ihned, nebo nejdéle na denní bázi. Jako přijatelná forma byla zvolena video konference, která se bude odehrávat

společně v čas interního Standup meetingu. Technicky toto řešení neklade žádné překážky, obě firmy disponují zasedací místností s televizorem a kamerou.

I přes výše zmíněný komunikační kanál je s dodavateli na začátku každého sprintu naplánována osobní schůzka, na které budou představeny a vysvětleny jednotlivé položky sprint backlogu, aby bylo minimalizováno riziko špatného pochopení zadávaných funkcionalit.

Zhodnocení sprintu

Z celkových 7 položek (4 položky na interní vývojový tým) sprint backlogu v celkovém objemu 33 plánovaných dní (15 dní pro interní vývojový tým) byly všechny položky sprint backlogu realizovány. Během druhého sprintu bylo nahlášeno 6 chyb, které však tým opravil vše čase stanoveném na opravy. Jeden z vývojářů interního vývojového týmu dokonce dokončil přidělenou práci o 1,5 dne dříve, než bylo plánováno, a započal práce na úkolu, který byl plánován až do třetího sprintu.



Obrázek 29: Položka (User story) produktového backlogu (Zdroj: Autor)

Na začátku druhého sprintu byla stanovena kapacita sprintu na 15 dní, tedy rychlost = 15. Nicméně splněním dalšího zařazeného úkolu z product backlog sezamu se rychlost zvýšila na hodnotu 17. Nová hodnota této veličiny pro další sprint bude stanovena jako aritmetický průměr dosavadních (skutečných) rychlostí za všechny proběhlé sprinty. Tedy obecný vzorec je:

$$\bar{V} = \frac{\sum_{i=1}^{n} V_{sprint\ (i)}}{n}$$

Níže je uvedena souhrnná tabulka odhadovaných a skutečných rychlostí:

Tabulka 12: Souhrnná tabulka odhadovaných a skutečných rychlostí (Zdroj: Autor)

Interní vývojový tým	Odhadovaná rychlost týmu	Skutečná rychlost týmu
Sprint #1	16	15
Sprint #2	15	17

Výsledná hodnota pro třetí sprint je:

V = (15 + 17) / 2 = **16**

Tento údaj je používán pro hrubý odhad a snaha o exaktní predikování této veličiny nám nemusí nutně přinést lepší výsledky. Opět je třeba naslouchat týmu a do odhadů budoucí rychlosti zahrnout plánované dovolené, nepřítomnost kvůli školením, či přijetí nového člena týmu, což může mít zprvu negativní dopad na rychlost týmu.

Uzavření sprintu

Schůzka je pevně stanoveným bodem v čase, se kterým zadavatelé požadavků počítají a vědí, že uvidí své požadavky implementované v testovacím prostředí, které odpovídá reálnému stavu produkční verze. Prezentační schůzka hotových funkčností se tak stala překvapivě pozitivním milníkem v procesu vývoje firemního SW.

Co se týká komunikace s externími dodavateli, každodenní Stand-up meeting se nepodařilo z technických důvodů navázat na denní bázi, nicméně i tak probíhala v dostatečné míře pro vyjasnění všech otázek. Prozatím je tato forma komunikace dostačující nicméně tím, že se jedná o externího dodavatele, který je geograficky na jiném místě, vyžaduje si tato forma spolupráce při agilní metodice vývoje produktu obecně vyšší pozornost. Při větších a náročnějších požadavcích bude nutná **osobní konzultace** interních analytiků s dodavatelskými vývojáři s vyšší frekvencí než doposud.

4.1.1.6 Možnosti zlepšení procesu vývoje SW

Jak vyplývá z předchozí kapitoly, zavedení metodiky Scrum ve sledované firmě proběhlo bez závažnějších problémů a během několika plánovacích iterací se podařilo průběh sprintů stabilizovat (tedy stanovit optimální rychlost týmu) a sladit očekávání obchodního (i dalších oddělení) s realitou, což byl rovněž jeden z hlavních cílů nově

nasazené metodiky. Nicméně zavedené řešení bylo v době sledování v rané fázi svého životního cyklu a bylo jasné, že bude třeba ho neustále monitorovat a upravovat a nepolevit a kontinuálně kontrolovat a dodržovat činnosti definované metodikou Scrum, což klade velké nároky zejména na osobu Scrum mastera. V té době nebyly vůbec řešeny krizové scénáře jako například onemocnění člena Scrum týmu, odchod Scrum mastera z firmy a další.

Hlavním identifikovaným problémem, který nezpůsobili interní vývojáři a který měl za následek velké množství hlášených chyb, byla **absence automatizovaného testování** stávajícího a nového programového kódu (Unit testing). Jako řešení tohoto problému byl navržen následující postup, který byl zákazníkem i týmem akceptován. Řešení se skládalo ze dvou částí:

1. **Nově vytvářený zdrojový kód** bude bez výjimky pokryt testy. Znamená to tedy mírné zvýšení odhadů náročnosti při ohodnocení náročnosti nových požadavků. Tato investice se však v budoucnu vyplatí právě sníženým rizikem možného zanesení chyb do nově vzniklých součástí systému.
2. **Stávající zdrojový kód** se bude testy pokrývat postupně, pokud se práce v probíhajícím sprintu stihne dříve, než bylo plánováno, tak se nebude do sprintu zařazovat úkol z produktového backlogu, ale čas se věnuje na postupné pokrývání kódu testy. Tento stav bude trvat do té doby, než bude současný kód pokryt plně Unit testy. Odvedená práce bude započítána do proměnné rychlost (velocity), bude s ní tedy nakládáno jako s prací odvedenou na standardních položkách sprint backlogu.

Navržená praktika měla být aplikována v následující sprintech, jejichž popis již není součástí této práce.

4.2 Nástroje pro podporu řízení a vývoje agilních SW projektů

V této kapitole jsou popsány jednotlivé typy nástrojů pro podporu řízení a vývoje, které mají svoje nedílné místo v oboru agilních metodik. Obecně lze nástroje rozdělit do tří skupin: fyzické, podpůrné a SW nástroje.

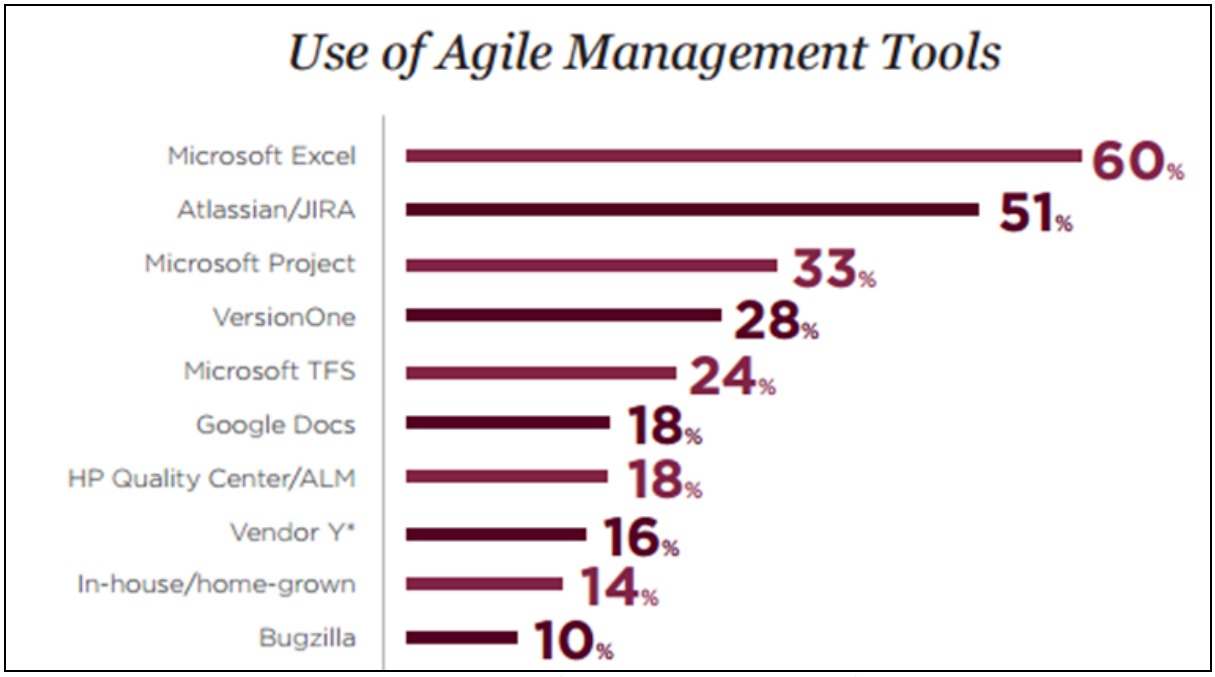
Fyzické nástroje byly zmíněny v teoretické části této práce, a to v kapitolách věnujících se metodikám Scrum a Kanban. Jedná se například o tabuli umístěnou

v kanceláři, na které jsou nalepené post-it lístečky, které reprezentují jednotlivé úkoly a sloupce na tabuli vyjadřují stav úkolu.

Podpůrné nástroje jsou nástroje, které samotnou metodiku nijak neřídí a nejsou ani potřebné pro její implementaci, nicméně poskytují určité nové přehledné pohledy na průběh a stav projektu. Jedná se například o Burndown Chart, který je podrobněji popsaný v kapitole zabývající se metodikou Scrum, nebo Cumulative Flow Diagram, vysvětlený v kapitole zabývající se metodikou Feature Driven Development.

SW nástroje jsou komplexní nástroje, ve kterých jsou programově implementovány jak fyzické nástroje, tak i podpůrné nástroje. Standardně také přinášejí nástroje pro základní správu projektových zdrojů a další typy diagramů pro zobrazení harmonogramu, jako je například Ganttův diagram.

Průzkum State of Agile Report 2015 ukazuje, že nejvíce používaným nástrojem pro agilní řízení, je tabulkový procesor Microsoft Excel[1].



Obrázek 30: Podíl použití nástrojů pro agilní řízení projektů. (Zdroj: [7]**)**

Zástupci SW nástrojů jsou představeni v následující kapitole. Jejich výběr proběhl na základě požadavků zadavatelské firmy.

[1] Respondenti výzkumu mohli označit více možností

Jako báze dostupných nástrojů byla použita online databáze serveru Project Management Zone, která k 3. 2. 2017 obsahovala 207 SW nástrojů pro plánování projektů. Pro zúžení výběrového souboru dat bylo nutné provést prvotní selekcí na základě kritérií, která byla následující:

- **Podpora agilní metodiky Scrum**
  - Ve firmě byla zvolena agilní metodika Scrum, nicméně nástroj by měl být dostatečně univerzální, aby bylo možné si z metodiky použít pouze to, co bude potřeba, a určité komponenty metodiky naopak nevyužívat.
- **Ověřené a používané řešení**
  - Další požadavek byl zvolit řešení, které bude globálně používané a časem ověřené. Ideálně k tomu pak zvolit nástroj, se kterým má již některý zaměstnanec zkušenost, což pomůže rychlejší integraci nástroje do prostředí firmy.
- **Podpora a aktivní vývoj produktu**
  - S přechodem na agilní metodiky vynaložila firma určité náklady a jejich součástí bylo i zaškolení pracovníků v novém nástroji. Je tedy potřeba, aby byla zajištěna podpora a rozvoj nástroje, v opačném případě by změna nástroje znamenala další nežádoucí náklady. Z toho vyplývá, že vývoj nástroje nesmí být ukončen.
- **On-line spolupráce na projektu**
  - Ve firmě pracují jak interní zaměstnanci, tak z části externisté, kteří nemusí být vždy fyzicky přítomni na pracovišti, a je tedy třeba, aby měli všichni pracovníci vždy náhled na seznam požadavků, viděli jejich rozřazení do sprintů, jejich priority atd.

Další zúžení souboru nástrojů bylo provedeno na základě výše uvedeného bodu "Ověřené a používané řešení", tedy dle již zmiňované studie The State of Agile Report, kdy se vybraly pouze ty nástroje, které jsou ověřené na reálných agilních projektech.

Na základě výše uvedených kritérií byly vybrány čtyři SW nástroje, které již splňují všechny uvedená kritéria: systém **JIRA** (od firmy Atlassian), **YouTrack** (od firmy JetBrains), **LeanKit** (od firmy LeanKit) a **Team Foundation Server** (od firmy Microsoft). Každý z těchto nástrojů je v následující kapitole ohodnocen dle stanovených kritérií.

4.2.1 Stanovení kritérií a metodiky porovnání SW nástrojů

Metoda pro porovnání byla inspirována diplomovou prací Lucie Hefnerové [40]. Jednotlivá kritéria jsou stanovena dle požadavků firmy, tedy po konzultacích s vedením firmy a IT ředitelem firmy. Pro každý SW nástroj bude následně hodnocena míra splnění níže definovaných kritérií. Kritéria jsou hodnocena body na stupnici od jedné do pěti, kde 1 představuje ohodnocení nejlepší a 5 nejhorší. Pro každý nástroj je následně vytvořen vážený průměr, což reprezentuje celkové hodnocení nástroje. Jednotlivé váhy samotných kritérií (stupnice 1 až 10, kde 10 znamená nejdůležitější kritérium a 1 nejméně důležité kritérium) byly stanoveny v součinnosti s vedením firmy a jsou uvedeny v následující kapitole.

Tabulka 13: Váhy kritérií pro konkrétní projekt (Zdroj: Autor)

Kritérium	Váha (0-10)
Kritérium: Podpora notifikačních kanálů	6
Kritérium: Podpora více projektů	3
Kritérium: Přehledy	8
Kritérium: Kvalita uživatelského rozhraní	6
Kritérium: Použitelnost na mobilních platformách	2
Kritérium: Modifikace struktury dashboardů	7
Kritérium: Application Programming Interface (API)	3
Kritérium: Nastavení uživatelských oprávnění	7
Kritérium: Serverová Platforma	5
Kritérium: Cena v horizontu 5 let	10
Kritérium: Forma podpory	8

4.2.2 Kritéria pro výběr nástroje

Kritérium: Podpora notifikačních kanálů

Kritérium definuje, jakým způsobem jsou uživatelé nástroje informováni o aktivitách v systému a jaké jsou možnosti nastavení notifikací.

Tabulka 14: Kritérium: Podpora notifikačních kanálů (Zdroj: Autor)

Hodnocení	Popis
1	Komplexní podpora notifikací (pro projekt, pro typ požadavku, vazba na konkrétní akce v systému) uživatelsky nastavitelná - více kanálů.
2	Rozšířená podpora notifikací uživatelsky nastavitelná - více kanálů.
3	Základní podpora notifikací uživatelsky nastavitelná - jeden kanál.

4	Základní podpora notifikací globálně nastavitelná.
5	Žádná podpora notifikací.

Kritérium: Podpora více projektů

Kritérium definuje, kolik projektů a týmů je schopen nástroj obsloužit v jeden moment.

Tabulka 15: Kritérium: Podpora více projektů (Zdroj: autor)

Hodnocení	Popis
1	Podpora neomezeného množství projektů.
2	Podpora omezeného počtu projektů s možností dokoupení dalších.
3	Podpora omezeného počtu projektů.
4	Podpora do 5 projektů.
5	Podpora pouze 1 projektu.

Kritérium: Podpora více projektů

Kritérium definuje schopnosti nástroje vytvářet a uživatelsky definovat přehledy.

Tabulka 16: Kritérium: Podpora více projektů (Zdroj: Autor)

Hodnocení	Popis
1	Komplexní podpora přehledů - přednastavené přehledy a grafy s možností definovat vlastní a určovat u nich přístupová oprávnění.
2	Rozšířená podpora přehledů uživatelsky nastavitelné.
3	Základní podpora přehledů uživatelsky nastavitelná - bez grafů.
4	Základní podpora přehledů globálně definováno.
5	Žádná podpora přehledů.

Kritérium: Kvalita uživatelského rozhraní

Kritérium definuje kvalitu GUI a UX obecně. Je důležité, aby se nástroj dobře používal, protože s ním budou zaměstnanci v kontaktu prakticky neustále.

Tabulka 17: Kritérium: Kvalita uživatelského rozhraní (Zdroj: Autor)

Hodnocení	Popis
1	Propracované GUI a UX s možností personalizace a nastavení klávesových zkratek pro každého uživatele zvlášť.
2	Personalizované GUI s možností globálně měnit vzhled a klávesové zkratky.
3	Základní GUI s globálně nastavenými klávesovými zkratkami.
4	Základní GUI bez UX.
5	Pouze textové rozhraní.

Kritérium: Použitelnost na mobilních platformách

Kritérium definuje přístupnost nástroje nejen prostřednictvím webového rozhraní, ale i přes mobilní zařízení.

Tabulka 18: Kritérium: Použitelnost na mobilních platformách (Zdroj: Autor)

Hodnocení	Popis
1	Více než dvě nativní aplikace (Android, iOS, Windows 10 Mobile) a responzivní webové rozhraní.
2	Dvě nativní aplikace (Android, iOS) a responzivní webové rozhraní.
3	Jedna nativní aplikace (Android) a responzivní webové rozhraní.
4	Bez nativní aplikace ale responzivním webovým rozhraním.
5	Bez nativní aplikace pro mobilní zařízení a bez responzivního webového rozhraní.

Kritérium: Modifikace struktury dashboardů

Modifikace struktury dashboardů je důležitá, protože každý projekt může mít jinou granularitu dělení úkolů a jejich fází před nasazením.

Tabulka 19: Kritérium: Modifikace struktury dashboardů (Zdroj: Autor)

Hodnocení	Popis
1	Dashboardy lze s příslušným oprávněním upravovat přímo (tažením myši).
2	Dashboardy lze s příslušným oprávněním upravovat v nastavení - přímo při pohledu na dashboard.
3	Dashboardy lze s příslušným oprávněním upravovat v nastavení nástroje.
4	Dashboardy lze editovat pouze všechny najednou, tedy globálně.
5	Dashboardy nelze modifikovat - mají pevně danou strukturu.

Kritérium: Application Programming Interface (API)

API umožňuje integrovat funkčnosti nového systému do stávajícího ERP systému. Dále poskytuje možnost automatizace vybraných procesů.

Tabulka 20: Kritérium: Application Programming Interface (API) (Zdroj: Autor)

Hodnocení	Popis
1	API pokrývající celou funkčnost nástroje s popsanou dokumentací a s více možnými komunikačními datovými formáty.
2	API pokrývající celou funkčnost nástroje s popsanou dokumentací a s jedním možným komunikačním datovým formátem.
3	API s dokumentací a pokrývající pouze část funkcionalit nástroje.
4	API s neúplnou dokumentací a pokrývající pouze část funkcionalit nástroje.
5	Žádná podpora API.

Kritérium: Nastavení uživatelských oprávnění

Každý pracovník, od brigádníka, přes střední management až po vrcholné vedení, má jiná práva (Create, Read, Update, Delete) na jednotlivé funkčnosti a je třeba je systematicky řídit.

Tabulka 21: Kritérium: Nastavení uživatelských oprávnění (Zdroj: Autor)

Hodnocení	Popis
1	Komplexní podpora řízení uživatelských oprávnění s možností vytváření skupin a nastavení práv ke všem dostupným entitám v nástroji.
2	Rozšířená podpora řízení uživatelských oprávnění s možností vytváření skupin - omezené nastavení práv k entitám.
3	Globální řízení uživatelských oprávnění.
4	Předdefinovaná uživatelská oprávnění - bez možnosti změny.
5	Žádná podpora řízení uživatelských oprávnění.

Kritérium: Serverová Platforma

Infrastruktura firmy se může v průběhu času mírně měnit a podpora více platforem snižuje riziko dalších investic v budoucnu.

Tabulka 22: Kritérium: Serverová Platforma (Zdroj: Autor)

Hodnocení	Popis
1	Nástroj podporuje více než dvě platformy - alespoň jedna platforma nevyžaduje další náklady.
2	Nástroj podporuje dvě platformy - alespoň jedna platforma nevyžaduje další náklady.
3	Nástroj podporuje dvě platformy - platformy vyžadují další náklady.
4	Nástroj je určen pouze pro jednu platformu - platforma nevyžaduje další náklady.
5	Nástroj je určen pouze pro jednu platformu - platforma vyžaduje další náklady.

Kritérium: Cena v horizontu 5 let

Kritérium pásmově definuje cenu nástroje v horizontu 5 let (do 50 uživatelů).

Tabulka 23: Kritérium: Cena v horizontu 5 let (Zdroj: Autor)

Hodnocení	Popis
1	0 Kč až 149 999Kč
2	150 000 Kč až 299 999 Kč
3	300 000 Kč až 599 999 Kč
4	600 000 Kč až 999 999 Kč
5	1 000 000 Kč a více

Kritérium: Forma podpory

Kritérium definuje dostupnost a formu podpory poskytovanou dodavatelem nástroje. V určitých momentech může být rychlá podpora pro firmu kritická.

Tabulka 24: Kritérium: Forma podpory (Zdroj: Autor)

Hodnocení	Popis
1	Podpora dostupná 24/7 - více komunikačních kanálů.
2	Podpora dostupná 24/7 - jeden komunikační kanál.
3	Podpora dostupná pouze v pracovní dobu - více komunikačních kanálů.
4	Podpora dostupná pouze v pracovní dobu - jeden komunikační kanál.
5	Bez garantované podpory.

4.2.3 Konkrétní SW nástroje

Níže uvedená tabulka č. 25 je přehledem vybraných nástrojů pro podporu agilních metodik. V tabulce jsou uvedeny i finanční náklady potřebné k provozu jednotlivých systémů. Náklady jsou rozděleny do skupin dle stanovených hledisek:

1. **Počet uživatelů**
    a. 0 až 10
    b. 11 až 50 uživatelů
2. **Typ licence**
    a. Web-based service (SaaS)
    b. Hostované řešení

Tabulka 25: Přehled vybraných nástrojů pro podporu agilních metodik (Zdroj: Autor)

Systém	JIRA	YouTrack	LeanKit	Team Foundation Server
Vývojář	Atlassian	JetBrains	LeanKit	Microsoft
Webová stránka	atlassian.com/software/jira	jetbrains.com/youtrack	leankit.com	visualstudio.com/cs/tfs
Rok prvního vydání	2002	2009	2009	2005
Licence	Web-based service (SaaS), Hostované řešení	Web-based service (SaaS), Hostované řešení	Web-based service (SaaS)	Web-based service (SaaS), Hostované řešení
Webové rozhraní	ano	ano	ano	ano
Operační systém (v případě hostovaného řešení)	Linux, Solaris, Windows	Linux, OS X, Solaris, Windows	ne	Windows

Systém	JIRA	YouTrack	LeanKit	Team Foundation Server
Nativní mobilní aplikace	ne (responzivní webové rozhraní)	Android, iOS	Android, iOS	Android, iOS, Windows Phone
API	ano	ano	ano	ano
Více uživatelů	ano	ano	ano	ano
Role uživatelů	ano	ano	ano	ano
Autentifikace uživatelů	Password, LDAP	LDAP, Google sign-on, Yahoo sign-on	Password	Password, HTTP authentication, Active Directory
Notifikační kanál	Email	Email	Email, RSS feeds	Email
Další vlastnosti				
Hierarchické úkoly	ano	ano	ano	ano
Opakující se úkoly	ne	ne	ne	ne
Závislosti mezi úkoly	ano	ne	ano	ano
Ganttův diagram	ano	ne	ne	ne
Vykazování času	ano	ano	ano	ne
Sledování nákladů	ano	ne	ne	ne
Podpora metodiky Scrum	ano	ano	ano	ano
Podpora metodiky Kanban	ano	ano	ano	ano
Cena[2]				
Web-based service (SaaS)				
do 10 uživatelů/rok	252 Kč	zdarma	96 595 Kč	9 056 Kč
11-50 uživatelů/rok	90 558 Kč	41 100 Kč	482 976 Kč	105 651 Kč
Hostované řešení				
do 10 uživatelů (jednorázová	252 Kč	zdarma	není	125 523 Kč

[2] Všechny ceny jsou přepočítané kurzem vyhlašovaným ČNB, ke dni 3. 2. 2017 kdy 1 USD = 25,155 CZK

Systém	JIRA	YouTrack	LeanKit	Team Foundation Server
platba)				
11-50 uživatelů (jednorázová platba)	83 012 Kč	20 550 Kč	není	627 617 Kč

4.2.4 Bodové ohodnocení vybraných nástrojů

Níže uvedená tabulka č. 26 souhrnně ukazuje:

1. Váhy jednotlivých kritérií
2. Bodové ohodnocení jednotlivých podpůrných nástrojů
3. Výpočet váženého průměru

Tabulka 26: Výběr vhodného podpůrného nástroje (Zdroj: Autor)

Kritérium	Váha (0-10)	JIRA	YouTrack	LeanKit	Team Foundation Server
Kritérium: Podpora notifikačních kanálů	6	1	1	3	1
Kritérium: Podpora více projektů	3	1	1	1	1
Kritérium: Přehledy	8	1	1	2	1
Kritérium: Kvalita uživatelského rozhraní	6	1	1	2	1
Kritérium: Použitelnost na mobilních platformách	2	4	2	2	1
Kritérium: Modifikace struktury dashboardů	7	1	1	2	2
Kritérium: Application Programming Interface (API)	3	1	2	3	1
Kritérium: Nastavení uživatelských oprávnění	7	1	1	1	1
Kritérium: Serverová Platforma	5	1	1	5	5
Kritérium: Cena v horizontu 5 let	10	3	1	5	5
Kritérium: Forma podpory	8	1	1	4	1
Vážený průměr:		1,400	**1,077**	2,923	2,031

5 Zhodnocení výsledků a doporučení

Zavedení metodiky Scrum ve sledované firmě proběhlo bez závažnějších problémů a během několika plánovacích iterací se podařilo průběh sprintů stabilizovat (tedy stanovit optimální rychlost týmu) a sladit očekávání obchodního (i dalších oddělení) s realitou, což byl rovněž jeden z hlavních cílů nově nasazené metodiky.

Stav po zavedení metodiky jasně prezentuje kladné výsledky zavedení metodiky Scrum. V podniku se zlepšil systém vývoje softwaru. Před zavedením metodiky bylo evidováno 47% požadavků ve zpoždění z celkových 15 požadavků na novou funkcionalitu softwaru. Po zavedení metodiky se nenacházel v backlogu (seznamu požadavků) jediný požadavek ve zpoždění oproti plánu.

Nejobtížnějším úkolem při zavádění nové metodiky vývoje softwaru je správně naformulovat požadavky očekávání zadavatele. Také je důležité dodržovat základní principy dané metodiky a nesnažit se je hned po zavedení metodiky porušovat a obcházet, což klade velké nároky na osobní disciplínu každého účastníka realizačního týmu. Konkrétně v metodice Scrum je důležité v žádném případě nepřijímat požadavky na realizaci dodatečné funkčnosti během běžícího sprintu, protože takovéto zásahy mohou celý vývojový cyklus významně narušit na dlouho dobou dopředu.

Stanovení schůzek typu Review a Retrospektiva v jeden mezní den, který nespadá do žádného sprintu, přináší všem zúčastněným možnost absolvovat obě schůzky bez nátlaku a strachu ze zpoždění prací a zároveň to umožňuje týmu pečlivě naplánovat další sprint.

Poučení se z chyb z předešlého sprintu umožňuje chyby eliminovat a naopak praktiky, které se osvědčily, používat nadále. Neustálá revize vývojového procesu je klíčem úspěšnému fungování metodiky.

Po zavedení metodiky, kdy byl zákazník s vývojem i požadovaným výsledným produktem spokojen, výsledný stav měl pozitivní dopad na činnost vývojového týmu i na jeho motivaci k dalšímu postupu a zdokonalování. Zlepšilo se plánování, všechna oddělení ve firmě měla lepší přehled o tom, kdy budou jejich požadavky realizovány, jakou mají prioritu a kde vznikají problémy.

Zobecnění výsledků této práce je možné pouze z části. Úspěšný výsledek zavedení zvolené metodiky Scrum ve sledované konkrétní firmě lze považovat za vzorový, nikoli však za obecně platný návod pro zavedení agilní metodicky do všech podobných firem.

Obecná doporučení:

- Pokud v podniku dosud žádná metodika není, může být agilní metodika vhodným začátkem pro optimalizaci procesů. Principy agilní metodiky jsou jednoduché - krátké seznamy úkolů, postupné dodávky. Vhodné pro malé týmy i pro jednotlivce.
- Základem agilních metodik je živá, rychlá, flexibilní komunikace zadavatel-dodavatel, která posouvá vývoj produktu správným směrem vpřed. Tato metoda není vhodná pro firmy s rigidním hierarchickým systémem, který vyžaduje komunikaci osob na stejné úrovni hierarchie (ředitel-ředitel, vedoucí-vedoucí, apod.).
- Z pohledu zadavatele i dodavatele může být agilní metoda doporučena jako optimální v případě nejasného zadání výsledného požadovaného produktu, kdy není možno stanovit rozpočet, nebo z různých důvodů ještě nejsou známy všechny požadované funkcionality produktu. V tomto případě je nezbytně nutný flexibilní přístup zadavatele, který není vázán žádnými kriticky omezujícími faktory (např. trváním na léty používaných, avšak zastaralých postupech a neochotou ke změnám) a je připraven i na variantu, kdy zcela kompletního výsledného produktu nemusí být dosaženo.
- Z pohledu dodavatele může být agilní metoda doporučena pro realizaci velkých projektů ve velkých firmách, kdy cílový úkol je velkého rozsahu a jeho řešení zasahuje mnoho interních sekcí. Zde je nezbytně nutná velká zkušenost na straně dodavatele při koordinaci svého vývojového týmu, který bude řešit vývoj produktu tak, aby byl i přes svůj velký rozsah stále konzistentní. V tomto případě je nezbytně nutná otevřená a bezproblémová komunikace zejména na straně velké firmy, mezi vedoucími jednotlivých sekcí, kde musí být v maximální míře potlačena rivalita a maximálně podporována vzájemná součinnost pro dosažení nejlepšího požadovaného výsledného produktu.

6 Závěr

Hlavním cílem práce bylo vybrat vhodnou metodiku a zavést ji v konkrétním podniku. Tento cíl byl splněn. Pomocí systému METES byla vybrána metodika Scrum, která byla po nutných připravách zavedena v konkrétním podniku. Při zavádění metodiky musely být řešeny problémy jako například relevantní odhadování náročnosti jednotlivých požadavků pomocí agilních karet (kdy se na odhadu podílí nejenom vývojář ale celý vývojový tým) nebo například prvotní nevole ze strany zákazníka k přizpůsobení se novému způsobu vývoje.

Dílčím cílem byla analýza dostupných softwarových nástrojů pro podporu metodik vývoje softwaru, jejich filtrace dle zvolených základních kritérií a další podrobná selekce na základě rozšiřujících kritérií. A autor stanovil celkem jedenáct rozšiřujících kritérií, která jsou hodnocena body na stupnici od jedné do pěti. Pro každý nástroj je následně vytvořen vážený průměr, na základě vah jednotlivých kritérií, což reprezentuje celkové hodnocení nástroje. Na základě stanovení konkrétních priorit (vah a bodového hodnocení) podniku byl vybrán podpůrný software YouTrack od firmy JetBrains s.r.o.

Výsledky zavedení metodiky v konkrétní firmě potvrzují, že pouhé zavedení metodiky není jednorázová činnost, která týmu zaručeně pomůže dodávat kvalitnější produkt, ale je to spíše směr, který nám pomáhá posouvat se k cíli. Stav kdy vývojový tým maximálně využívá svůj potenciál a přitom není přetěžován, je ideálním stavem pro každý podnik a pro každý tým. K tomuto stavu se lze přiblížit zodpovědným a kvalifikovaným výběrem metodiky.

Na vývojový tým i zadavatele klade každá metodika jiné požadavky. Je tedy rozhodně vhodné, aby všechny subjekty, které budou v dané metodice aktivně zapojeny, absolvovaly školení pro vybranou metodiku. Jinak může docházet k nejasnostem v používané terminologii a hlavně nastavení správného očekávání ohledně výsledků a kompetencí jednotlivých členů týmu.

Přístup agilních metodik umožňuje dodávat průběžné výsledky rychle a tím získá vývojový tým cennou zpětnou vazbu od zadavatele. Tím jsou ale agilní metodiky také ale náročnější na komunikaci. Agilní metodiky obecně vyžadují, aby zúčastněné strany o projektu neustále přemýšlely a snažili se nastavené procesy revidovat a zlepšit dle aktuálních nedostatků a problémů. Obecně agilní metodiky nevnímají člověka jako

pouhopouhý lidský zdroj a jako nahraditelný prvek ale naopak jako na důležitou součást týmu a snaží se maximálně využít lidský potencionál.

7 Seznam použitých zkratek

Termín	Zkratka	Význam
Software	SW	
Informační technologie	IT	
Scaled Agile Framework	SAF	
Man day	MD	Člověkoden, Jeden den práce člověka.
Extrémní programování	XP	
Enterprise Resource Planning	ERP	
Multidimensional Management and Development of Information Systems	MMDIS	
Feature Driven Development	FDD	
Project Management Body of Knowledge	PMBOK	
Project Management Institute	PMI	
PRojects IN Controlled Environments 2nd Version	PRINCE2	
Information Technology Infrastructure Library	ITIL	
Control Objectives for Information and Related Technologies	COBIT	
Methodology Evaluation System	METES	

8 Seznam použitých zdrojů

[1] KOVÁŘ, Vladimír. *Interní materiál společnosti UNICORN*. 2007.

[2] KOVÁŘ, Vladimír a KOLEKTIV. *02 Přehled současných metodik I*. Praha: UNICORN, 2007. Interní materiál společnosti UNICORN.

[3] HIGHSMITH, James A. *Agile project management*. 2nd ed. Upper Saddle River, NJ: Addison-Wesley, nedatováno. creating innovative products. ISBN 03-216-5839-6.

[4] HASTIE, Shane, Stéphane WOJEWODA a Jennifer LYNCH. Standish Group 2015 Chaos Report - Q&A with Jennifer Lynch. In: . nedatováno.

[5] HIGHSMITH, Jim. Beyond Scope, Schedule, and Cost: The Agile Triangle. In: . nedatováno.

[6] KRISTIANSEN, Stephan. Success rate of agile projects. In: . nedatováno.

[7] *The State of Agile Report 2015* [online]. [vid. 2017-01-28]. Dostupné z: https://versionone.com/pdf/VersionOne-10th-Annual-State-of-Agile-Report.pdf

[8] *2015 State of Scrum Report* [online]. [vid. 2017-01-28]. Dostupné z: http://www.pm-progetti.it/download/scrum-alliance-state-of-scrum-2015.pdf

[9] JAROSLAV, Ráček. *Agilní metodiky vývoje SW* [online]. [vid. 2017-01-28]. Dostupné z: https://is.muni.cz/el/1433/podzim2013/PA017/um/SWE2_07_agilni.pdf

[10] MACHACKOVA, Eva. *Budoucnost bude agilní. I pro projektové manažery* [online]. [vid. 2017-01-28]. Dostupné z: https://zework.wordpress.com/2014/06/20/budoucnost-bude-agilni-i-pro-projektove-manazery/

[11] BUCHALCEVOVÁ, Alena. *Metodiky vývoje a údržby informačních systémů*. Praha: Grada Publishing, 2005. kategorizace, agilní metodiky, vzory pro návrh metodiky. ISBN 8024710757.

[12] BUCHALCEVOVA, Alena A O - 10.1007/978-1-4419-7355-9_47. *The Methodology Evaluation System Can Support Software Process Innovation* [online]. New York, NY: Springer New York. 2011. ISBN 9781441972057. Dostupné z: http://link.springer.com/10.1007/978-1-4419-7355-9_47

[13] System and method for software methodology evaluation and selection.

[14] *Rational Unified Process: základní pojmy*

[15] AMBLER, Scott. The Full Life-Cycle Object-Oriented Testing (FLOOT) Method. In: . nedatováno.

[16] KOVÁŘ, Vladimír. *01 Metodika vývoje software - jak se řídí projekt*. 2007. Interní materiál společnosti UNICORN.

[17] SONG, William Wei. *Information systems development*. nedatováno. Asian experiences. ISBN 978-144-1972-057.

[18] *Spirálový model* [online]. [vid. 2017-01-28]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model/

[19] EELES, Peter. Layering Strategies. *IBM developer Works* [online]. 2001. Dostupné z: http://www.acis.org.co/fileadmin/Curso_Memorias/Curso_CMMI_Sep06/Modulo 2 - Product Engineering /RUP Overview/application developer/papers/pdf/tp199_layering_strategies.pdf

[20] KADLEC, Václav. Programujte agilně, nic jiného vám nezbývá! A nebo ano? In: . nedatováno.

[21] *BRUCKNER - Tvorba informačních systémů - Principy, metodiky, architektury.pdf*

[22] *Manifest Agilního vývoje software* [online]. [vid. 2017-01-28]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html
[23] BOEHM, Barry a Richard TURNER. *Balancing Agility and Discipline: A Guide for the Perplexed* [online]. 2003. ISBN 0321186125. Dostupné z: doi:10.1007/978-3-540-24675-6_1
[24] TOMÁNEK, Martin. Současný stav používání agilních metodik ve světě a v ČR. *Acta Informatica Pragensia* [online]. 2015, **4**(1), 4–17. ISSN 1805-4951. Dostupné z: doi:10.18267/j.aip.48
[25] CROMARTY, Simon. 5S Your Scrum Board – (Part 2) – In depth. In: [online]. nedatováno. Dostupné z: http://theagilepirate.net/archives/tag/information-radiator
[26] POLLICE, G. Using the RUP for Small Projects: Expanding Upon eXtreme Programming. *A Rational Software White Paper* [online]. 2003. Dostupné z: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:Using+the+RUP+for+Small+Projects:+Expanding+Upon+eXtreme+Programming#1
[27] *FDD: Processes* [online]. [vid. 2017-01-28]. Dostupné z: http://www.step-10.com/SoftwareProcess/FeatureDrivenDevelopment/FDDProcesses.html
[28] MCCONNELL, Steve. *Dokonalý kód*. 1. Brno: Computer Press a.s., 2005. ISBN 9788025140512.
[29] BUCHALCEVOVÁ, Alena. METODIKA FEATURE-DRIVEN DEVELOPMENT NEOPOUŠTÍ MODELOVÁNÍ A PROCESY, A PŘESTO PŘINÁŠÍ VÝHODY AGILNÍHO VÝVOJE. nedatováno.
[30] BECK, Kent. a Cynthia. ANDRES. *Extreme programming explained*. 2nd ed. Boston, MA: Addison-Wesley, 2005. embrace change. ISBN 0321278658.
[31] KNESL, Jiří. SprintMethod. In: *YouTube* [online]. Google: 2012, 2012. DevFest. Dostupné z: https://www.youtube.com/watch?v=WcwByZA4OvM
[32] *ExtremeProgramming.org home* [online]. [vid. 2017-01-28]. Dostupné z: http://www.extremeprogramming.org/map/project.html
[33] ŠOCHOVÁ, Zuzana. *Zuzi's blog* [online]. 2015. Agile and Lean, Scrum, Kanban, XP @ Business. Dostupné z: http://soch.cz/blog/
[34] MYSLÍN, Josef. *Scrum*. 1. vydání. Brno: Computer Press, 2016. průvodce agilním vývojem softwaru. ISBN 978-80-251-4650-7.
[35] *AGILNĚ VE VELKÝCH ORGANIZACÍCH* [online]. [vid. 2017-01-02]. Dostupné z: https://blog.think-forth.com/2016/07/14/agilne-ve-velkych-organizacich/#comments
[36] POPEL, Martin. Vyu ž ití metodiky SCRUM ve v ý vojov ý ch t ý mech. 2015.
[37] *How to Make Agile Estimation Process Easy with Planning Poker* [online]. [vid. 2016-12-23]. Dostupné z: http://www.softwaretestinghelp.com/planning-poker-scrum-poker-cards-agile-estimation/
[38] *DEALING WITH BUGS IN SCRUM* [online]. [vid. 2017-02-03]. Dostupné z: http://blog.softartisans.com/2011/11/14/dealing-with-bugs-in-scrum/
[39] ROMAN PICHLER. *How to Scale the Scrum Product Owner* [online]. [vid. 2017-02-01]. Dostupné z: http://www.romanpichler.com/blog/scaling-the-product-owner/
[40] HEFNEROVÁ, Lucie. *Kanban a možnosti jeho využití v bankovním prostředí*. Praha, 2015. b.n.
[41] SUTHERLAND, Jeff. A funny Scrum Master movie with Jeff Sutherland. In: [online]. nedatováno. Dostupné z: https://www.youtube.com/watch?v=oheekef7oJk
[42] SUTHERLAND, Jeff a Ken SCHWABER. *The Scrum Guide*. The Definitive Guide to Scrum: The Rules of the Game

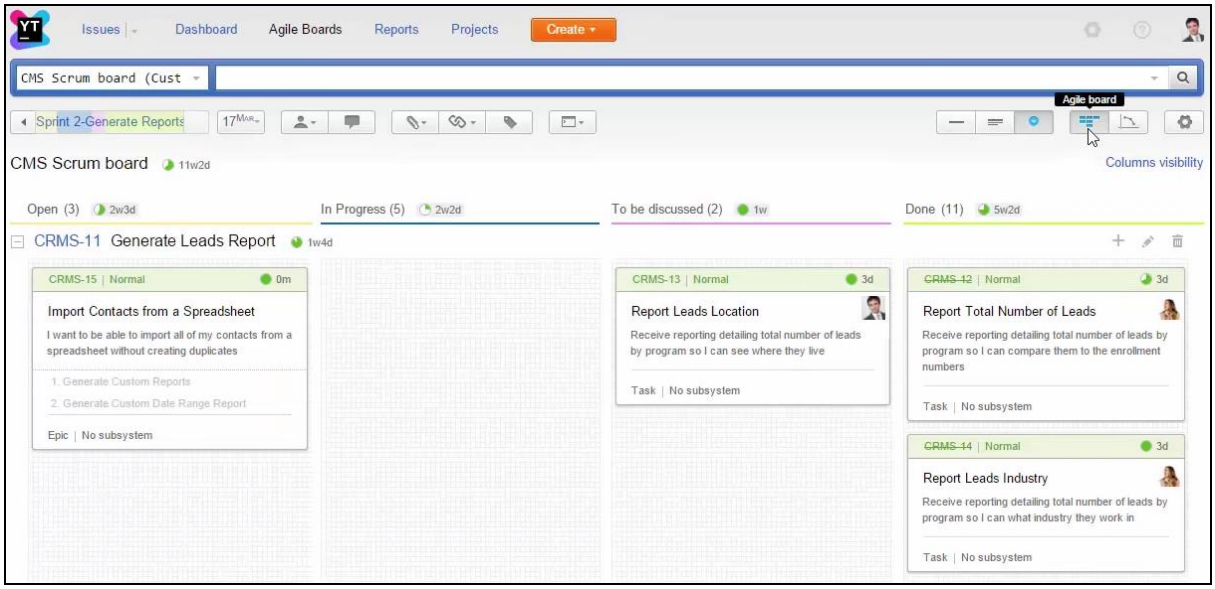
[43] ŠOCHOVÁ, Zuzana a Eduard KUNCE. *Agilní metody řízení projektů*. 1. vyd. Brno: Computer Press, 2014. ISBN 978-80-251-4194-6.
[44] DALALAH, Ahmad. Extreme Programming: Strengths and Weaknesses [online]. 2013. Dostupné z: http://www.acit2k.org/ACIT/2013Proceedings/132.pdf
[45] ANDERSON., David J. *Kanban*. 1. vydání. Sequim, Wash: Blue Hole Press, 2010. successful evolutionary change in your software business. ISBN 0984521402.
[46] AMBLER, Scott W. *The Agile Scaling Model (ASM)* [online]. 2009. Adapting Agile Methods for Complex Environments. Dostupné z: https://www.researchgate.net/profile/Scott_Ambler/publication/268424579_Adapting_Agile_Methods_for_Complex_The_Agile_Scaling_Model_ASM_Adapting_Agile_Methods_for_Complex_Environments/links/55003e780cf28e4ac347ee34.pdf?origin=publication_detail

9 Přílohy

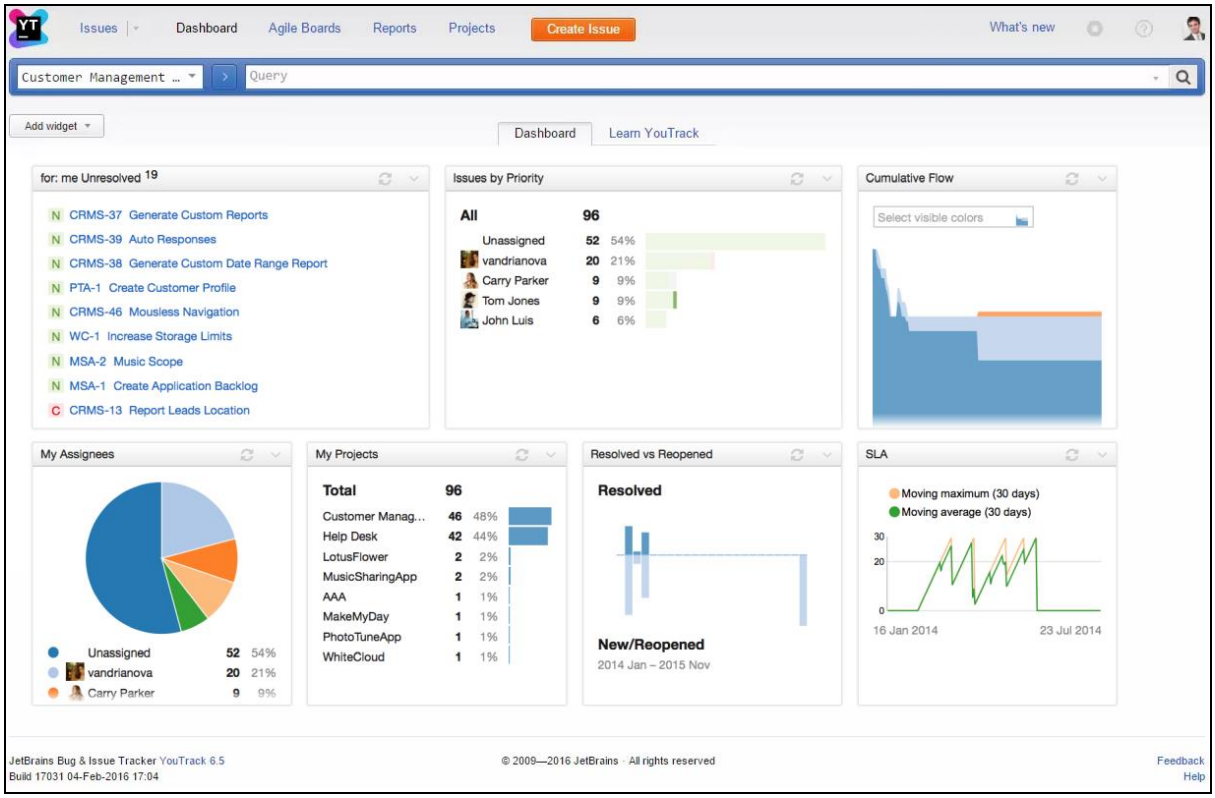
Příloha A: Hlavní benefit, klady, zápory a vhodnost metodiky

	Ústřední benefit	Klady	Zápory	Vhodné pro
Scrum	Maximální spolehlivost časových odhadů	Nejrozpracovanější teorie; Zvládá i zákazníka na pracovišti; Nejpoužívanější - nejjednodušší implementace; fixní délka sprintů usnadňuje analýzy, vyvíjení abstrakce a dává klidnější pracovní řád	Fixní sprinty nejsou vhodné pro týmy, které se o funkcích, které musí ven, dozví těsně před nutným začátkem práce. Nevhodné pro 1-2 členné startupy. Nevhodné pro vývoj hardware	Týmy > 2 lidí vyvíjející dlouhodobě pro zákazníka, nebo vyvíjející svůj produkt.
SprintMethod	Jednoduchost, napojení na marketing	Vhodné pro vývojáře hardware, tam, kde není možné dostat zákazníka na pracoviště, pro jednotlivce, malé týmy, startupy, školní projekty, pro výzkumné projekty, tam, kde není možné odhadnout čas, protože je stráven výrazný čas analýzami	Více modulů znamená, že ikdyž někdo implementuje SprintMethod, neovládá ho ještě celý.	Pro všechny, komu se z nějakého důvodu nehodí Scrum. SprintMethod je záměrně vytvářen jako doplňková metodika.
Extrémní programování	Nejvyšší možná kvalita	Využití agilních technik do maxima. Maximální rychlost učení se. Spolehlivá metodika.	Nespolehlivé odhady toho, kdy bude práce hotová.	Tam, kde každá chyba hodně bolí (jaderné elektrárny, kardiostimulátory apod.)
Lean Software Development (nesprávně nazývaný Kanban)	Nejflexibilnější	Možnost rychle zareagovat na změnu. Přehledný kanban board. Velmi minimalistická forma řízení.	Naprosto neprodejné pro zákaznické zakázky. Téměř nemožné dělat časové odhady.	Pro vlastní produktový vývoj.
Getting Real	Napojení na marketing, jednoduchost	Rozpracovaná strategie obsahující i tipy pro hledání lidí, marketingu, výběr vlastností k implementaci atd.	Nejméně literatury k tématu (pouze knihy od 37signals).	Pro malé týmy vyvíjející svůj produkt v malém týmu 1 vývojáře, 1 UX/grafika a 1 kodéro-manažera
Scrumban	Metodika jen pro přechod od Scrumu k Lean Software Developmentu, není samostatnou metodikou. Jedná se o ne výrobní strategii, ale change management strategii			Dočasně pro ty, kdo přechází od Scrumu k Lean Software Developmentu

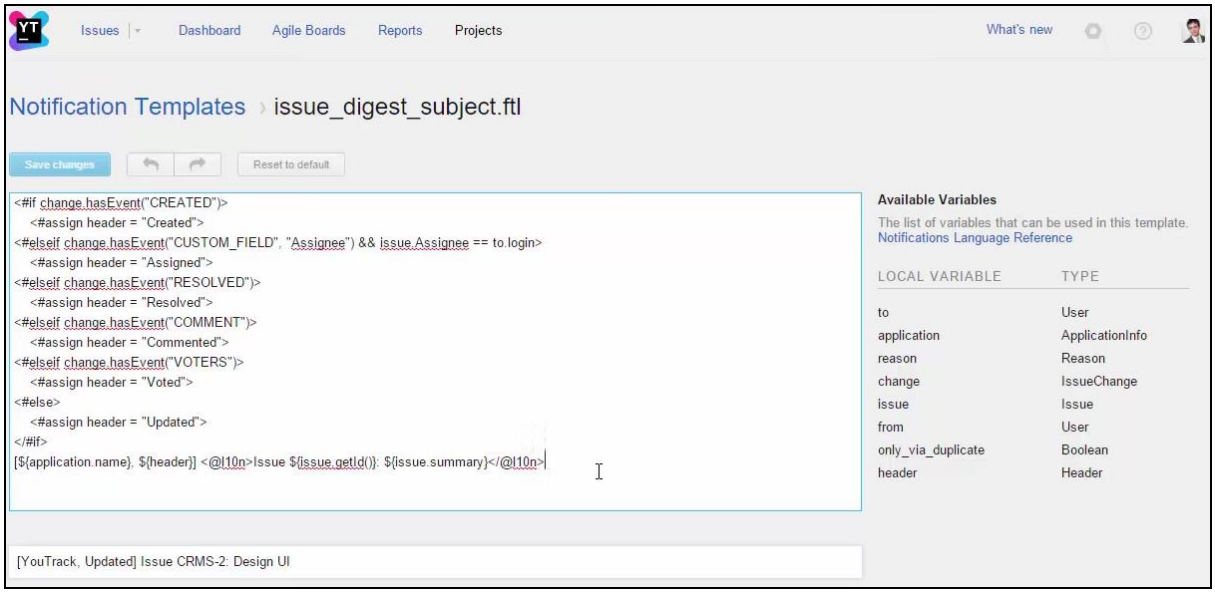
Příloha B: Ukázková tabule v systému YouTrack



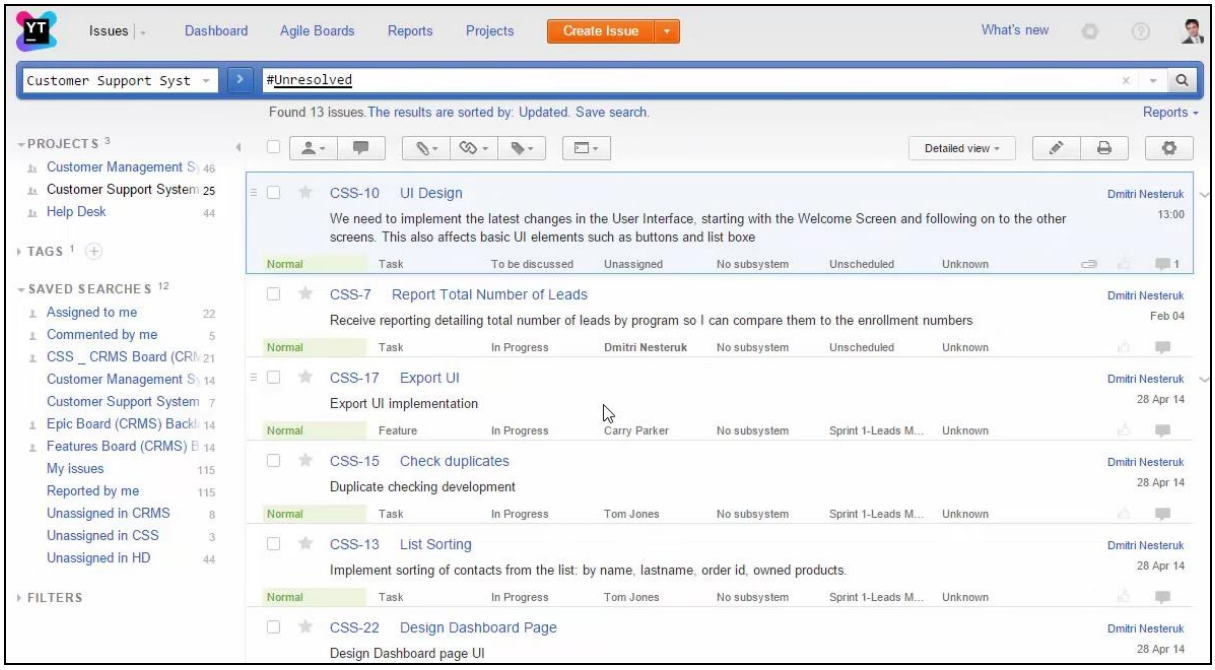
Příloha C: Ukázkový dashboard s vybranými grafy v systému YouTrack



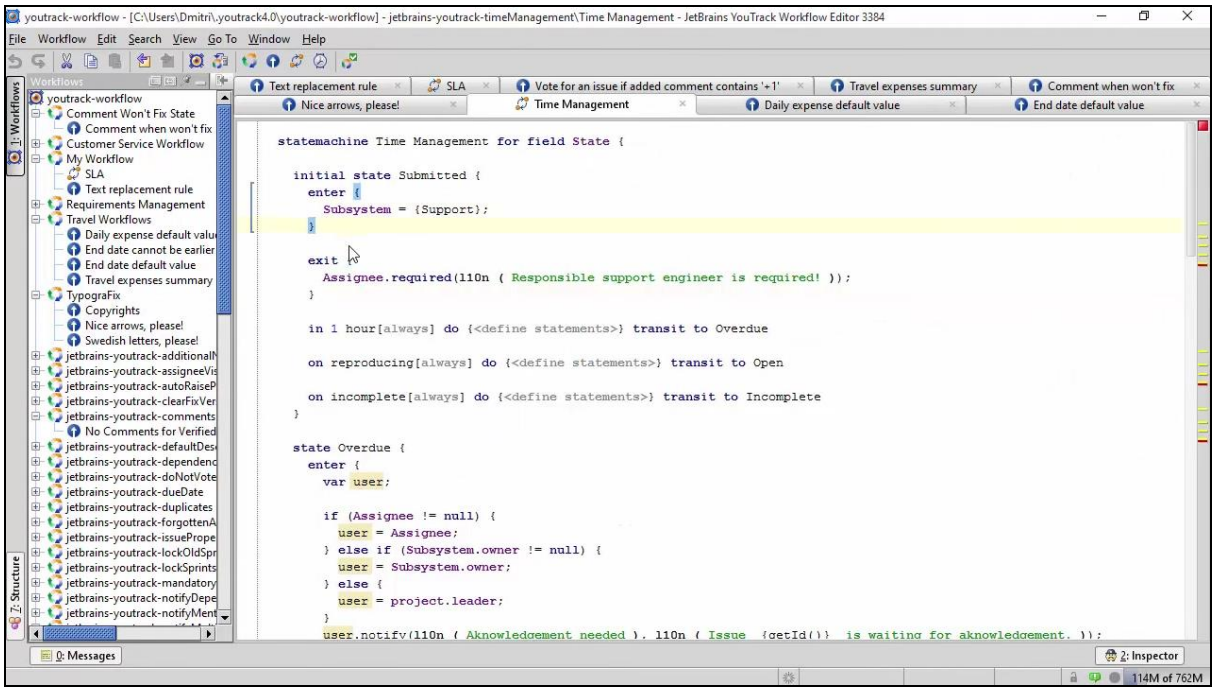
Příloha D: Nastavení obsahu zasílané notifikace v systému YouTrack



Příloha E: Přehled všech nevyřešených požadavků v systému YouTrack



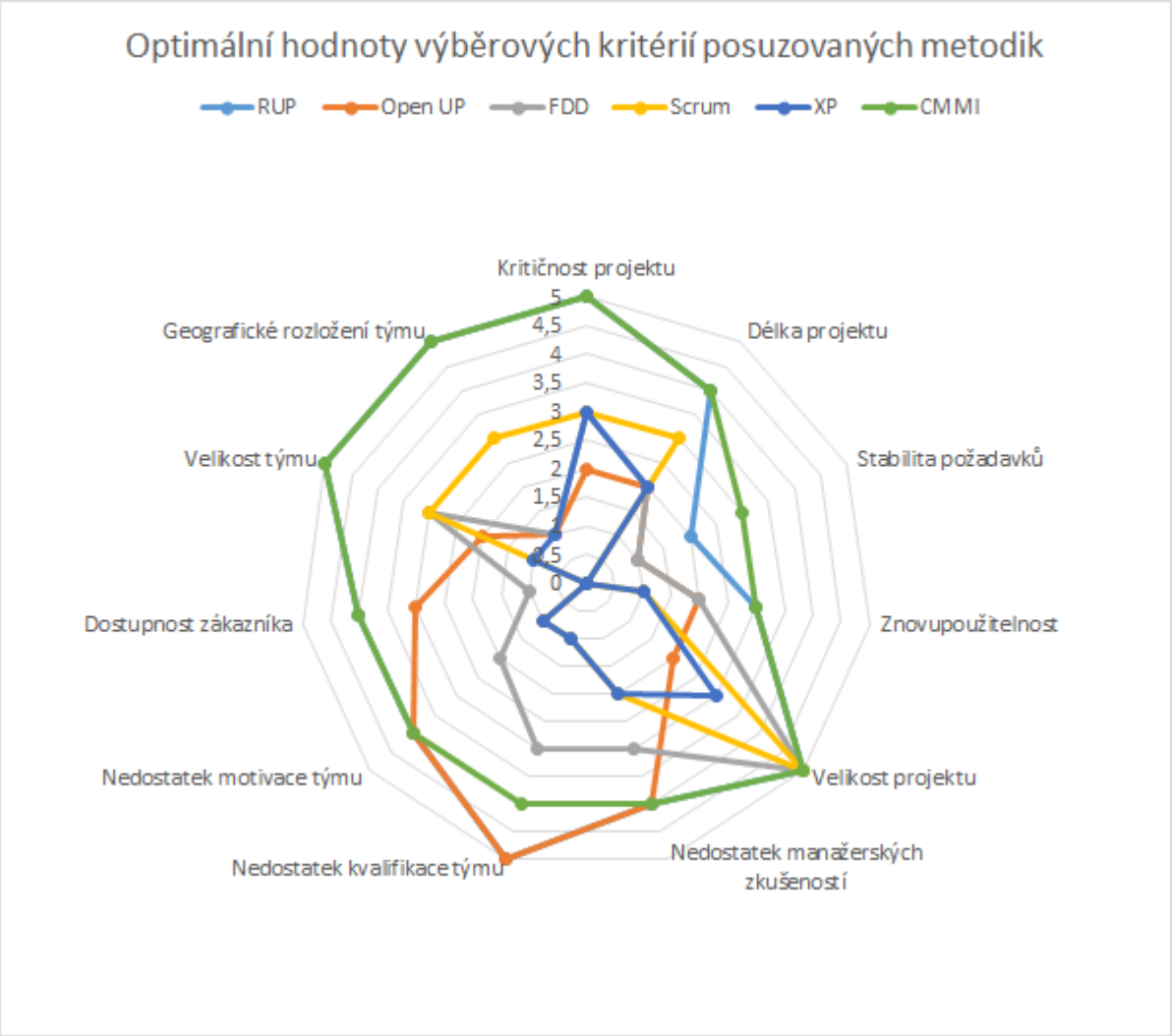
Příloha F: “YouTrack Workflow Editor” pro definici vlastních akcí a procesů v nástroji YouTrack



Příloha G: Vybrané atributy, včetně jejich částečného ohodnocení dle Hacksela

People	SunTone	RUP	SCRUM	XP	Waterfall
Communication					
- Number of Geographic Locations					
* One (1)				+	+
* Few (2 - 3)				-	N
* Many (> 3)				- -	N
- Accessibility of requirements providers					
* low	-	-	-	- -	N
* medium	+	N	N	-	
* high	+	N	+	+	
- offshore component					
* low			+	+	N
* medium			-	- -	N
* high			N	N	N
- Project size					
* low				++	-
* medium				-	N
* high				- -	+
- Communication Index					
* low (1 - 2)				- -	N
* medium (3 - 5)				-	N
* high (6 - 7)				+	-
Leadership					
- Release Manager Leadership					
N/A			-	N	-
low			-	N	-
medium			N	N	N
high			+ +	N	+
- Lead Architect Leadership					
N/A	- -				
low	- -				
medium	N				
high	+ +				
Experience					
- Project Manager Experience					
N/A	-		- -		
low (0 - 1)	N		- -		
medium (2 - 4)	+		N		
high (5 - 6+)	+ +		+		
- Lead architect experience					
N/A	- -				
low (0 - 1)	- -				
medium (2 - 4)	N				
high (5 - 6+)	+ +				
Skill level					
* low (1 - 2)				- -	-
* medium (3 - 5)				N	N
* high (6 - 7)				+	N

Příloha H: Optimální hodnoty výběrových kritérií posuzovaných metodik



Příloha I: Stupnice hodnocení výběrových kritérií dle systému METES

Důležitost produktu

Stupnice hodnocení:

- 0 – jen pilotní projekt
- 1 – doplňkový systém
- 2 – systém podporující fungování organizace
- 3 – systém kritický pro poslání (národní organizace)
- 4 – systém kritický pro poslání (nadnárodní organizace)
- 5 – systém, na kterém závisí životy lidí

Délka projektu

Stupnice hodnocení:

- 0 – do 1 měsíce
- 1 – do 3 měsíců
- 2 – do 6 měsíců
- 3 – do 12 měsíců
- 4 – do 24 měsíců
- 5 – nad 24 měsíců

Stálost požadavků

Stupnice hodnocení:

- 0 – požadavky není předem možné detailně stanovit
- 1 – požadavky se více než z 50% mění
- 2 – procento změn požadavků je cca 30%
- 3 – požadavky lze definovat předem, mění se jen priority požadavků
- 4 – požadavky lze definovat předem, mění se, ale snahou je změny potlačovat
- 5 – požadavky lze definovat předem a nemění se

Znuvupoužitelnost

Stupnice hodnocení:

- 0 – cílem není znuvupoužitelnost
- 1 – snaha používat již hotové komponenty
- 2 – snaha vytvářet znovupoužitelné třídy v rámci projektu
- 3 – snaha vytvářet znovupoužitelné spustitelné komponenty v rámci projektu
- 4 – snaha vytvářet znovupoužitelné spustitelné komponenty v rámci organizace
- 5 – cílem je maximální znuvupoužitelnost v rámci organizace

Velikost řešení

Stupnice hodnocení:

- 0 – 1 až 10
- 1 – 11 až 40
- 2 – 41 až 100
- 3 – 101 až 200
- 4 – 201 až 300
- 5 – 300 a více

Zkušenost manažera projektu

Stupnice hodnocení:

- 0 – 5 a více
- 1 – 4 až 5 let
- 2 – 3 až 4 roky
- 3 – 2 až 3 roky
- 4 – 1 až 2 roky
- 5 – 0 až 1 rok

Kvalifikace členů týmů

Stupnice hodnocení:

- 0 – více než 70% členů týmu je kvalifikovaných, s širokým zaměřením
- 1 – více než 70% členů týmu je kvalifikovaných, ale specializovaných
- 2 – cca 50% členů týmu je málo kvalifikovaných
- 3 – více než 60% členů týmu je málo kvalifikovaných
- 4 – více než 70% členů týmu je málo kvalifikovaných
- 5 – více než 80% členů týmu je málo kvalifikovaných

Motivace členů týmu

Stupnice hodnocení:

- 0 – motivovaní jedinci vysokých morálních kvalit, sdílí znalosti, sami se organizují
- 1 – aktivní, motivovaní jedinci, sdílí znalosti
- 2 – plní zadané úkoly, sdílí znalosti
- 3 – plní zadané úkoly, nesdílí znalosti
- 4 – jedinci jsou špatně motivováni a snaží se vyhýbat úkolům, nesdílí znalosti
- 5 – žádná motivace

Dostupnost uživatelů

Stupnice hodnocení:

- 0 – uživatel je součástí týmu, má odpovědnost za požadavky
- 1 – uživatel je k dispozici denně
- 2 – uživatel je k dispozici kdykoliv na vyžádání
- 3 – uživatel je k dispozici na začátku, konci a v průběhu projektu, v předem určených milnících
- 4 – uživatel je k dispozici jen na začátku a na konci projektu
- 5 – uživatel není dostupný

Velikost týmu

Stupnice hodnocení:

- 0 – 1 až 4
- 1 – 4 až 10
- 2 – 11 až 20
- 3 – 21 až 50
- 4 – 51 až 100
- 5 – více než 100

Rozmístění

Stupnice hodnocení:

- 0 – v jedné místnosti
- 1 – v jedné budově
- 2 – více míst v jednom městě
- 3 – dvě místa v jedné zemi
- 4 – více míst v jedné zemi
- 5 – mimo jednu zem