Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra managementu

Zavedení agilní metodiky vývoje softwaru

Diplomová práce

Autor: Bc. František Linder
Studijní obor: Informační management

Vedoucí práce: doc. Ing. Hana Mohelská, PhD.
Odborný konzultant: Ing. Filip Ježek
GIST, s.r.o.

Hradec Králové duben 2019

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury a uvedených zdrojů.

vlastnoruční podpis

V Hradci Králové 26.4.2019 Bc. František Linder

Poděkování:

Děkuji mé vedoucí práce doc. Ing. Haně Mohelské, PhD. za metodické vedení, cenné připomínky a trpělivost při zpracování této diplomové práce. Rovněž bych chtěl poděkovat Ing. Filipovi Ježkovi ze společnosti GIST, s.r.o. za konzultační rady.

Anotace

Název: Zavedení agilní metodiky vývoje softwaru

Diplomová práce je zaměřena na metodiky vývoje softwaru. Záměrem je vybrat a navrhnout vhodnou agilní metodiku pro zkoumanou společnost. Teoretická část na úvod představuje obor softwarové inženýrství a s ním spojené hlavní milníky ve vývoji softwaru. Dále jsou v teoretické části představeny dva způsoby vývoje softwaru, jedním je tradiční přístup a tím druhým je agilní přístup. V obou případech jsou vybrány hlavní metodiky, které reprezentují daný způsob vývoje softwaru. Aplikační část je rozdělena na čtyři kapitoly. V úvodu je analyzován současný stav procesu vývoje softwaru ve společnosti GIST, s.r.o. Další kapitola je věnována případové studii, která rozebírá používání agilních metodik a nástrojů po celém světě. Závěrečné dvě kapitoly diplomové práce se soustředí na výběr vhodné agilní metodiky pro analyzovaný podnik a následně se věnují návrhu na zavedení vybrané metodiky a jejich praktik.

Annotation

Title: Implementing Agile Software Development Methodology

This diploma thesis focuses on software development methodologies. The purpose of this thesis is to select and to propose an appropriate agile methodology for the company under review. The theoretical part introduces the field of software engineering and its main milestones in software development. Furthermore, two ways of software development are introduced in the theoretical part, one is the traditional approach and the other one is the agile approach. In both cases, the main methodologies that represent the software development method are selected. The application part is divided into four chapters. The introduction analyses the current state of software development in the company GIST, s.r.o. The next chapter presents a case study that analyses the use of agile methodologies and agile tools across the world. The last two chapters of this diploma thesis focus on the selection of the appropriate agile methodology for the analysed company and then they deal with the proposal to introduce the selected methodology and their practices.

Obsah

1 Úvod

Často slýcháváme, jak se doba rychle mění, co bude dál za pár let a tak dále. Mnohdy jsou tyto fráze spojovány s informačními a komunikačními technologiemi, které zastávají v dnešní společnosti dominantní pozici. Jsou všude kolem nás a není takřka dne, aby se nám podařilo těmto technologiím vyhnout. Stejně jako se vyvíjely například různé operační systémy, tak se vyvíjely způsoby řízení vývoje softwaru, a právě na vývoj softwaru je zaměřena tato práce.

Teoretická část práce je rozdělena do tří částí. V první části jsou uvedeny základní informace o softwarovém inženýrství, pod které vývoj aplikace spadá. Jedná se o jeden z nejmladších inženýrských oborů, který je pravděpodobně i jeden z nejrychleji rostoucích. K historii softwarového inženýrství patří softwarová krize, která odstartovala diskuzi o založení nového oboru. V úvodní kapitole se rovněž dozvíme, jak se dá měřit kvalita vývoje procesu softwaru pomocí modelu CMM. Poté jsou představeny dva přístupy k vývoji softwaru.

Druhá část popisuje tradiční vývoj softwaru a byl určitou reakcí na softwarovou krizi. Tradiční metodiky kladou důraz na fáze analýz, specifikací, testů apod. a právě jejich absence byla příčinou zmíněné krize. V práci jsou představeny tři tradiční metodiky, první je pro všechny známý Vodopádový model životního cyklu, dále Boehmův spirálový model a metodika Rational Unified Process.

Třetí část obsahuje agilní přístup k vývoji softwaru, který je hlavní naplní této práce. Podstatou všech agilních metodik je Agilní manifest, který obsahuje čtyři hodnotné výroky a tvoří tak základ pro agilní vývoj softwaru. Dále je uvedeno několik vybraných artefaktů agilního programování, bez kterých by se samotné metodiky neobešly. Na závěr této části jsou představeny tři nejpoužívanější agilní metodiky, a to Extrémní programování (XP), Kanban a Scrum.

Aplikační část je rozdělena do čtyř kapitol. První představuje společnost GIST, s.r.o. a popisuje proces vývoje softwaru uvnitř podniku. Druhá kapitola analyzuje agilní chování firem po celém světě a sleduje nastupující trendy. Třetí kapitola vybírá

vhodnou agilní metodiku pro zkoumanou společnost za pomoci vícekriteriální analýzy. Poslední část aplikační části se věnuje návrhu implementace vybrané agilní metodiky.

1.1 Cíl práce a metodika

Cílem diplomové práce je přehledně a srozumitelně popsat agilní metodiky řízení vývoje softwaru, jejich hlavní rozdíly od tradičních metodik a navrhnout postup, jak by měla softwarová společnost postupovat při přechodu od vodopádové k agilní metodice.

V teoretické části, která se zabývá softwarovým inženýrstvím a metodikami vývoje softwaru, je nejprve provedena rešerše dle uvedené odborné literatury, a to zejména z knih Softwarové inženýrství od Ian Sommerville, Agilní programování od Václava Kadlece, Scrum: Průvodce agilním vývojem softwaru od Josefa Myslína, Agilní metody řízení projektů od Zuzany Šochové a Eduarda Kunce a Skvělý ScrumMaster od Zuzany Šochové. Právě Zuzana Šochová je uznávanou agilní koučkou a držitelkou mnoha certifikátů od americké Scrum Alliance, která je nejuznávanější certifikační autoritou v agilním světě.

V aplikační části je představena softwarová společnost GIST, s.r.o. a analyzován její proces vývoje aplikace GIST Intelligence. Pro zpracování této části byly získány informace prostřednictvím polostrukturovaných rozhovorů s produktovým ředitelem firmy. Dále jsou v aplikační části porovnány průzkumy společnosti CollabNet VersionOne, která již od roku 2006 provádí dotazníkové šetření zaměřené na agilní vývoj po celém světě. V závěru práce je provedena vícekriteriální analýza pro výběr vhodné agilní metodiky, která bude navržena k implementaci v podniku GIST, s.r.o. Pro objektivní stanovení vah bylo provedeno dotazníkové šetření, ve kterém pět zaměstnanců (expertů) provedlo párové hodnocení vybraných kritérií. Z výsledků byly za pomoci Saatyho matice stanoveny váhy jednotlivých kritérií a tato kritéria byla ohodnocena zvolenými agilními metodikami. Ve výsledku nástroj pro podporu rozhodování Expert Choice 2000 vybral nejvhodnější agilní metodiku.

2 Softwarové inženýrství

2.1 Úvod do softwarového inženýrství

2.1.1 Definice

Najít jednu správnou a jedinou definici pro softwarové inženýrství je jako najít jednu správnou a jedinou ženu na celý život. Nicméně softwarové inženýrství můžeme chápat jako technickou disciplínu, která zavádí a ctí inženýrské principy od počátku specifikace softwaru až po jeho údržbu. Správný softwarový produkt poznáme až ve chvíli, kdy je připraven k předání zákazníkovi, který za něj zaplatí (Sommerville, 2013).

2.1.2 Typy softwarového produktu

Na softwarovém trhu se setkáváme se dvěma typy softwarových produktů:

1) Obecné produkty – klasické „krabicové“ programy, které jsou volně dostupné pro všechny zákazníky. Příkladem mohou být různé textové procesory, grafické editory, účetní systémy apod.
2) Přizpůsobené produkty – mnohdy nazývané jako zakázkové, protože zákazník si diktuje konkrétní funkcionality a další požadavky, které softwarová firma musí vzít v potaz a dle toho vytváří software „ušitý“ na míru. Vzhledem k časové náročnosti a individuálnímu přístupu se tento fakt odráží i na konečné ceně, která bývá násobně vyšší (Sommerville, 2013).

2.1.3 Charakteristické rysy softwarového produktu

- **Schopnost údržby** – software by měl být napsán takovým způsobem, aby byl schopen zareagovat na změny potřeb zákazníka a byl tak lehce upravitelný.
- **Spolehlivost a bezpečnost** – v případě výpadku by systém neměl způsobit žádnou fyzickou ani ekonomickou škodu. Díky správně nastavené bezpečnosti systém neumožní neoprávněným uživatelům přístup nebo poškodit systém.
- **Efektivita** – software by měl hledět na rozumné a pouze nutné hardware požadavky, neměl by plýtvat operační pamětí a cykly procesoru. Je tedy v zájmu vyvinout co nejefektivnější systém, který bude rychle reagovat na uživatelovy požadavky.

- **Přijatelnost** – nutno pamatovat, že software nevyvíjíme pro sebe, nýbrž pro koncového uživatele, který s ním bude pracovat, musí tedy být pro něj použitelný a dostatečně srozumitelný. Zároveň by měl být kompatibilní s jinými systémy, které uživatel používá (Sommerville, 2013).

2.1.4 Software vs. počítačové programy

Když se řeknou pojmy software a počítačový program, leckdo by mohl tvrdit, že se jedná o synonyma, ovšem není tomu úplně tak, i když jsou si velice blízké. Počítačový program si můžeme vytvořit sami se základními znalostmi nějakého programovacího jazyka, tento program by pravděpodobně sloužil pouze pro soukromé účely, tudíž nějakou dokumentaci a uživatelskou příručku bychom k programu nevytvářeli. A v tom se právě liší software od počítačového programu. U softwaru již musíme počítat, že počítačové programy vyvíjíme pro někoho jiného, většinou pro zákazníky, takže správný software musí obsahovat pochopitelně napsané programy, které jsou doprovázeny související dokumentací a konfiguračními daty, které jsou nezbytné pro bezproblémový chod těchto programů. Dodána může být i systémová dokumentace, která popisuje strukturu systému, uživatelská dokumentace, která poskytuje rady při používání systému, a webové stránky, kde jsou uživateli poskytnuty aktuální informace o produktu (Sommerville, 2013).

2.2 Historie softwarového inženýrství

Obor softwarové inženýrství patří k těm nejmladším inženýrským oborům, který zažívá neustálý rozvoj a prochází silnými změnami, což je také důvodem, proč mnohé aspekty softwarového inženýrství nejsou exaktně stanoveny.

Průkopnická léta se odehrávala do poloviny 60. let minulého století, do této doby měli přístup k velice drahým přístrojům pouze šťastlivci a nadšenci, ti vytvářeli jednoúčelové programy danému počítači. V žádném případě nešlo o řízený přístup k vývoji.

Jakožto poprvé jsme se mohli doslechnout pojmu softwarové inženýrství na konferencích NATO v letech 1968 a 1969 k diskuzi o softwarové krizi (Naur a Randell, 1969). Ukázalo se, že individuální přístup k vývoji softwaru nelže škálovat na velké a složité softwarové systémy. Vývoj těchto systémů nabíral zpoždění, byl nespolehlivý a

sahal hluboko do kapsy. Bylo tedy navrženo, že inženýrský přístup k vývoji softwaru by měl snížit náklady a vést ke spolehlivějšímu softwaru.

V 70. letech se poprvé na trhu objevily „krabicové“ běžně dostupné programy, které doplnily dosavadní jedinečné řešení šité na míru. Na přelomu 70. a 80. let dochází na základě objednávky amerického ministerstva obrany k vývoji programovacího jazyka Ada, který zahrnoval pojmy strukturovaného programování. Jazyk byl pojmenován na počest hraběnky Ady Lovelace (1815–1852), která je považována za první programátorku. V 80. letech se software rozvíjí a společně s ním celý obor softwarové inženýrství. V tomto období nastupují softwarově-inženýrské metodiky, registrujeme rozvoj objektově orientovaných přístupů a v neposlední řadě vznikají tzv. CASE nástroje, které mají za úkol prostřednictvím softwaru podporovat vývoj nového softwaru.

Až v roce 1997 bylo softwarové inženýrství uznáno jako obor s certifikátem v USA (Kadlec, 2004; ‘Software engineering history’, 2008; Sommerville, 2013).

2.3 Softwarová krize

Psala se šedesátá léta minulého století a obor softwarové inženýrství zažil vzestup, na který, jak se později ukázalo, nebyl připraven. Vzhledem k rychlému vývoji v tomto oboru úměrně rostly zakázky pro vývoj softwaru. Tento vývoj se však setkával s problémy, které celé projekty prodražovaly, prodlužovaly, či vůbec nebyly dokončeny. Pro představu bývá uvedena statistika z tehdejších zakázek pro americkou vládu. Přes 40 % zakázek bylo dodáno, ale nikdy nebyly úspěšně použity, cca 25 % bylo zaplaceno, ale nikdy nedodáno, zhruba 15 % bylo po drastických úpravách zahozeno, necelá 3 % byla po úpravě použita a konečně pouze 2 % softwarů byly použitelné beze změny.

Příčin byla celá řada, vzájemně se kombinovaly, což mělo za výsledek extrémní neefektivitu ve vývoji softwarů. Jednou z příčin byla špatná komunikace mezi zákazníkem a analytikem, v té době role analytika ani neexistovala, takže zákazník většinou komunikoval přímo s vývojářem. Další příčinou byl nesprávný přístup programátorů, kteří spíše chtěli předvést svůj programátorský um a požadavky zákazníka nechávali na druhé koleji. Nesprávné odhady rovněž přispěly softwarové

krizi, ať už to byl chybný odhad času, ceny, rozsahu či efektivity. S nesprávnými odhady jde ruka v ruce špatné plánování, na začátku chyběl konkrétně zpracovaný plán a ani nebyla možnost vycházet z empiricky ověřených příkladů, protože žádné neexistovaly. Dílčím problémem bylo podcenění hrozeb a rizik, kdy se doufalo, že se daná chyba v budoucnu přehlédne.

V době softwarové krize se ještě o softwarovém inženýrství jako oboru vůbec nehovořilo, právě softwarová krize byla jedním z hlavních spouštěčů další diskuze o založení nového oboru, která vyústila v samostatné téma na konferenci NATO (Kadlec, 2004).

2.4 Softwarový proces

S všelijakými procesy se obyčejný smrtelník setkává každý den. *„Proces je po částech uspořádaná posloupnost aktivit vedoucí od daného počátečního stavu k danému cíli.“* Například takový proces oblíkání, počátečním stavem je, že ráno vstaneme a potřebujeme se obléct do práce, do školy, kamkoliv. Cíle dosáhneme tehdy, kdy stojíme oblečeni před prahem dveří připraveni odejít. Během počátečního stavu a konečného stavu jsme museli podstoupit po částech uspořádané aktivity, abychom to všechno zvládli. Po částech uspořádané právě proto, že nejde prohodit fázi oblékání spodního prádla a kalhot, i když by se určitě někteří „umělci“ našli (Myslín, 2016).

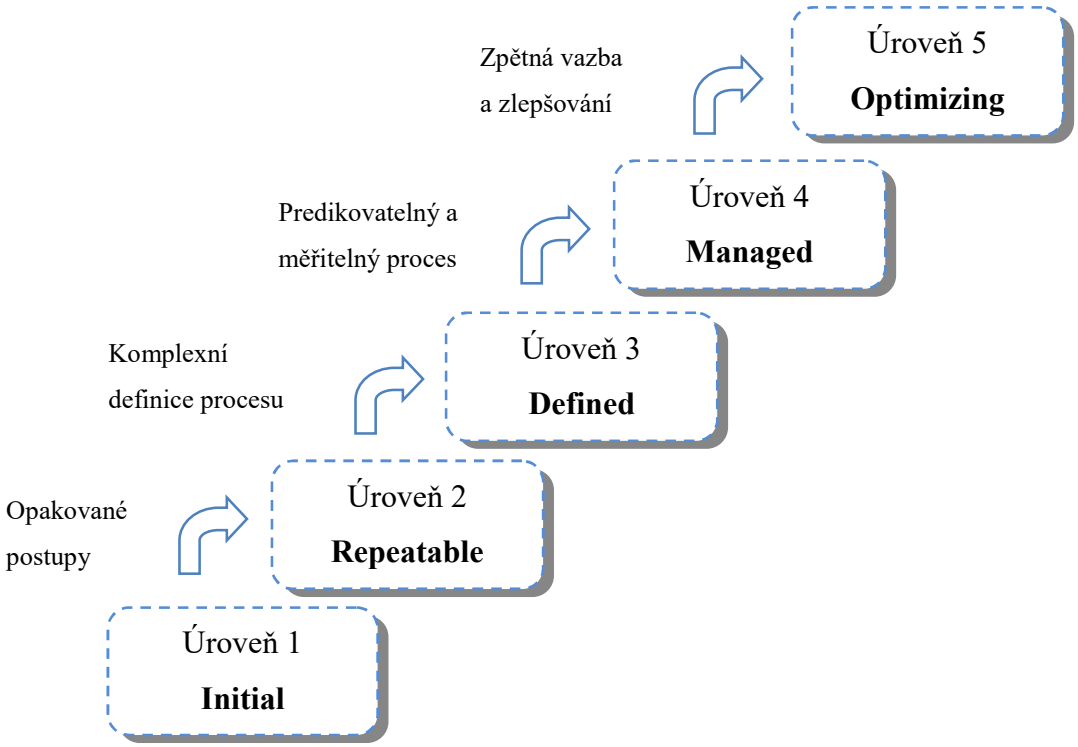
U softwarového procesu je to podobné, pomocí postupných aktivit docílíme produkce softwarového produktu. U softwarových procesů hovoříme o těchto čtyřech aktivitách:

1) **Specifikace softwaru** – zákazníci si společně s techniky definují budoucí funkcionality softwaru a omezení jeho činnosti.
2) **Vývoj softwaru** – software je navržen a poté naprogramován.
3) **Validace softwaru** – dochází ke kontrole softwaru a k otestování zákazníkem, zda zakázka odpovídá jeho požadavkům na začátku procesu.
4) **Evoluce softwaru** – vzhledem k měnícím se požadavkům zákazníka a trhu je povinností být připraven software udržovat a upravovat (Sommerville, 2013).

2.5 Model zralosti CMM

Možností, jak měřit kvalitu vývoje procesu softwaru ve firmě, je celá řada, nejznámějším a nejjednodušším zůstává model Capability Maturity Model (CMM),

který můžeme doslova a do písmene přeložit jako model vyspělosti dovedností. I když model není náročný na pochopení, pomůžeme si jednotlivé úrovně rozklíčovat pomocí následujícího obrázku (Myslín, 2016).



Obr. 1: Úrovně zralosti modelu CMM

Zdroj: vlastní zpracování na základě: (Myslín, 2016, s. 12)

2.5.1 Úroveň 1 – Initial (výchozí)

Na této úrovni nehovoříme o žádném vyvíjení kvalitního softwaru, úroveň procesů je výchozí. Společnost vyvíjející software stojí na samotném začátku a proces je pro ni úplně nový. Procesy na této úrovni nejsou nijak definovány, jsou chaotické a úkoly jsou řešeny způsobem ad hoc. Zároveň chybí řízení lidských zdrojů a jakékoliv plánování. U větších zakázek nelze přesně předpokládat bez definovaných procesů kolik projekt bude stát, za jak dlouho bude dodán a v jaké kvalitě (Myslín, 2016).

2.5.2 Úroveň 2 – Repeatable (opakovatelná)

Přechod na úroveň číslo 2 většinou probíhá samovolně a bez úmyslu vývojářského týmu. Pokud jsme řešili množinu podobných úkolu na úrovni 1 způsobem ad hoc, tak při řešení dalšího obdobného úkolu můžeme aplikovat postup, který se nám již osvědčil

při předešlých úkolech. Poté opakujeme postupy v dalších projektech či fázích téhož projektu. Stále v této fázi nemáme popsán proces jako takový, pouze využíváme určitých zkušeností na dílčí části. Nevýhodou je, že nikde nemáme zaručeno, že takový způsob je optimální. Ostatní části projektu jsou stále řešeny ad hoc přístupem (Myslín, 2016).

2.5.3 Úroveň 3 – Defined (definovaná)

Teprve třetí úroveň je brána něco jako systematický vývoj softwaru. Oproti předešlým dvěma úrovním, které popisovaly pouze části procesu, definovaná úroveň popisuje celý proces vývoje od začátku až do konce. Softwarový tým se neřídí podle náhodných postupů, ale řídí se dle předem nastavené metodiky vývoje softwaru. Jako záruka kvality softwaru slouží certifikát o řízení systému jakosti podle normy ISO 9001 – předpokládá komplexní popis procesů v dané firmě. Pro zisk této certifikace je nutné mít nastavené procesy alespoň na třetí úrovni. Neznamená to ale, že dosáhnout třetí úrovně je totéž jako zisk certifikátu dle normy ISO 9001. V dnešní době je větší tužba po onom certifikátu než po skutečném reálném zavedení systému řízení jakosti, a to z důvodu, že bez certifikátu se firmy nemohou účastnit většiny výběrových řízení (Myslín, 2016).

2.5.4 Úroveň 4 – Managed (řízená)

Předchozí úroveň byla minimem pro vyvíjení rozsáhlého moderního softwaru. Sem tam se někomu může povést vytvořit slušný software při nemetodickém postupu, ale v tomto případě jde pouze o náhody. Na třetí úrovni máme tedy rozsáhlý informační systém, který je propojený s dalšími softwarovými a hardwarovými systémy. Je něco, co přístupu na třetí úrovni chybí? Ano, je to měřitelnost a vyhodnocování. I když tvoříme dobrý software, nevíme, jestli je jeho postup jediný možný. Současný vývoj může produkovat zbytečné chyby a ztráty. Řízená úroveň kromě standardních procesů zainteresovává systémy pro měření kvality těchto procesů ve všech projektech. Systémy pro měření kvality hodnotí, zda postup vede ke správnému a úplnému výsledku a zda není vhodné změnit postup, který povede ke stejnému výsledku s vynaložením menšího úsilí. Pokud jsme schopni definovat sadu ukazatelů a k nim nadefinovat měřicí stupnici a následně provádět pravidelná systematická měření a poté je dokázat správně

vyhodnocovat, tak lze říci, že vyspělost softwarového procesu je na čtvrté úrovni (Myslín, 2016).

2.5.5 Úroveň 5 – Optimizing (optimalizace)

Nyní máme správně nastavené procesy a ukazatele, které dovolují hodnotit procesy, zda jsou dobře nastavené či nikoliv a stále je to málo. Samotné měření nám nemůže stačit, pokud nevede k zefektivnění celého procesu. Za předpokladu diagnostikování neefektivity našeho procesu se musíme postarat o implementaci nápravového mechanismu, který zajistí návrat odchylky reálného stavu do očekávaného stavu. Důležitý faktor hraje čas, pokud chybu odhalíme a opravíme včas, tak se vyhneme vážnějších následků (Myslín, 2016).

2.6 Metodiky vývoje softwaru

Předchozí kapitola nás provedla tím, jak by měl vypadat proces vývoje softwaru, který povede ke spokojenosti zákazníka. Abychom k těmto procesům dospěli, potřebujeme mít dokonale popsané, kdo, kdy a jak má jednotlivé činnosti dělat od počátku do konce projektu. K tomu nám napomáhá správně zvolená metodika vývoje softwaru. V týmu mohou být sebezkušenější „borci“, kteří mohou namítat, že daný úkol zvládnou bez jakékoliv metodiky. Může se tak stát u jednoduchých projektů, ale u rozsáhlejších se bez systematických a metodických kroků ztratí i skutečný „profík“. Takže schopné lidi je dobré mít v týmu, ale zároveň je nutné mít schopného člověka, který bude tým vést a koordinovat. A není to pouze o lidech v týmu, neméně důležité je nastavit komunikaci se zákazníkem, pravidla dokumentace a mnoho dalších (Myslín, 2016).

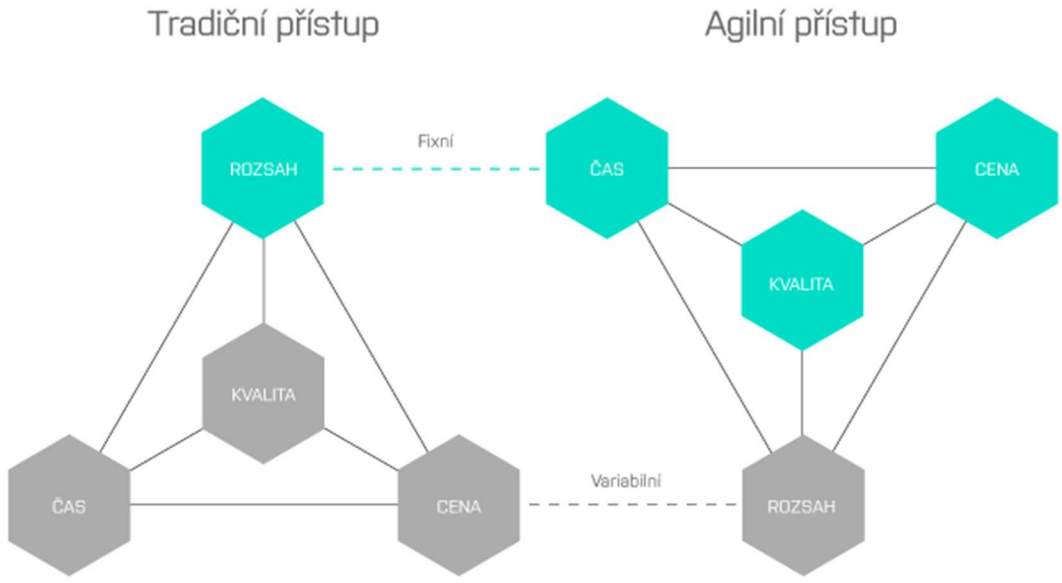
„Metodika není náboženství, metodika není dogma.“ Metodiku nemůžeme brát jako recept na bábovku, kde postupujeme přesně krok po kroku. Software totiž vyvíjíme, tudíž na začátku procesu nemáme zdání, jak bude na konci vypadat, přičemž u bábovky máme jasnou vizi finální podoby. Nelze tedy nastudovat některou z metodik a bezhlavě ji implementovat do svého projektu. Každý z projektů je jinak velký a svým způsobem specifický, proto neexistují univerzální a naprosto dokonalé postupy. Správný projektový manažer by měl uvážit jednotlivé metodiky a rozhodnout se pro tu nejvíce vyhovující pro následující projekt. Tedy nemějme metodiky za dogmata, ale za průvodce, který nás dovede ke spokojenosti všech zúčastněných stran (Myslín, 2016).

2.6.1 Typy metodik vývoje softwaru

V první kapitole jsme si stručně popsali historii softwarového inženýrství, obdobně se vyvíjely i metodiky vývoje softwaru. Přicházející metodiky byly odrazem aktuální situace v čase svého vzniku. Vývoj softwaru byl podmaněn poptávkou na software v dané době.

První skupinou metodik, které si podrobněji popíšeme v následující kapitole, jsou tradiční metodiky. Ty vznikly reakcí na softwarovou krizi, tudíž velký důraz byl kladen na fáze analýz, specifikací, testů apod., absence těchto činností vedla právě k již zmíněné krizi.

Současná uspěchaná doba měla vliv na vznik druhé skupiny metodik, která má za společný cíl dodat software za co nejkratší dobu v požadované kvalitě. Řeč je o agilních metodikách, které se snaží rychle a flexibilně vyvíjet aplikace. Společnost, která vyvíjí agilně, klade důraz na interakci se zákazníkem, aktivní komunikaci, snaží se zkrátit dobu pro analýzu a návrh, klade důležitost na implementaci, pravidelné vyvíjení prototypů a důsledné testování (Kadlec, 2004).



Obr. 2: Projektový trojúhelník

Zdroj: http://www.web-integration.info/cs/blog/agilni-projekty-zpohledu-zakaznika/Contents/0/agilni-projekty_schema_cz.png

Na obr. 2 vidíme model projektového trojúhelníku, který znázorňuje rozdíly mezi tradičním a agilním přístupem vývoje softwaru.

Tradiční metodiky si zakládají na přesně specifikovaných požadavcích na počátku projektu, které jsou smluvně zafixovány a lze s nimi těžce hýbat. Cena a doba dodání softwaru podléhá odhadům a jsou domlouvány na úrovni přání. Veškeré potencionální funkcionality podléhají detailním analýzám, aby se náhodou na něco nezapomnělo. Na základě těchto výstupů se realizační tým řídí a nebere zřetel na ostatní ovlivňující faktory, což mívá dopady na kvalitu dodávky. Nekvalitní dílo se samozřejmě klientovi nelíbí, a tak dodavatel navrhne opravu, která projekt prodraží a prodlouží, z čehož vyplývá, že čas a peníze jsou proměnné.

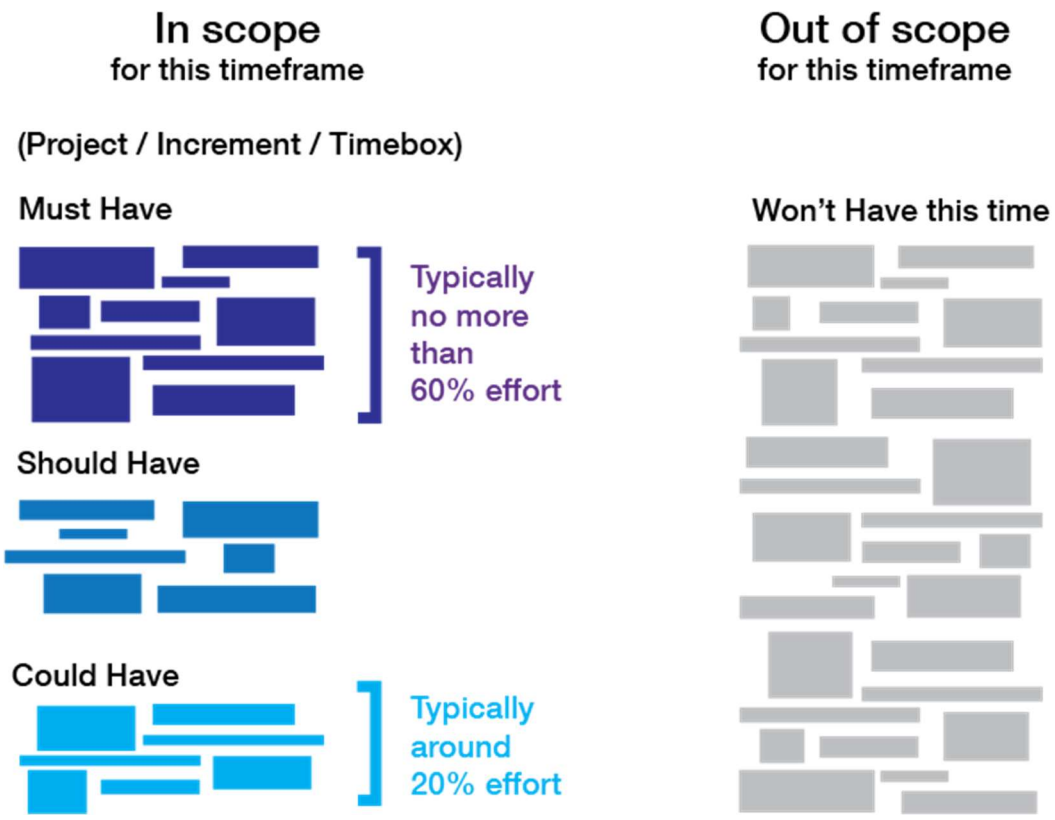
Na pravé straně obr. 2 máme opačný projektový trojúhelník, který reprezentuje agilní přístup a na rozdíl od tradičního přístupu říká, že projekt má jasně vymezený čas a cenu, kterou musíte a chcete dodržet při určité úrovni kvality produktu. To byly všechno neměnné parametry, jediné variabilní kritérium je rozsah funkcí, které je softwarová firma schopna dodat v předem daném časovém a finančním rozmezí. Agilní přístup kvituje návrhy na změny požadavků, reflektují totiž změny okolností, priorit a vnáší do projektu nové nápady. Alfou a omegou je tedy komunikace se zákazníkem, který návrhy předkládá a přisuzuje jim prioritu. Výsledkem poté je, že beze změny termínu dodání a navýšení rozpočtu se rozhodnete důležitější věci upřednostnit před těmi méně podstatnými. Leckdo by si mohl říct, že se vymanil od dokumentace, ale tomu tak není. Sice dokumentová náročnost není zdaleka tak vysoká jako u tradičních metod, ale úplně bez ní to taky nejde. Snahou je popsat základní „mantinely“ a základy projektu, kam právě budou spadat fixní části rozpočtu a času. V této fázi je potřeba se spolehnout na expertní odhady a nelhat si do kapsy. Rozmezí času a peněz musí být nadefinováno tak, aby developerský tým byl vůbec schopný reálně vytvořit do provozu fungující verzi. Pro takové případy není od věci využít techniky MoSCoW, která rozděluje požadavky dle priorit (‘Philosophy and Fundamentals’, c2019; Procházka, 2014)

2.6.2 MoSCoW prioritizace požadavků

V projektu, který je řízen agilním přístupem, je nezbytné pochopit důležitost práce, která má být provedena, aby bylo možné dodržet lhůtu dodání aplikace v předem domluveném čase. Úkoly a požadavky na systém v agilním světě nazýváme odborně

jako User Stories, což jsou uživatelské příběhy. MoSCoW technikou třídíme business požadavky do jednotlivých kategorií na základě jejich důležitosti pro konečný produkt. Název MoSCoW je odvozen z počátečních písmen čtyř prioritních skupin:

- **M**ust Have (musí mít) – tyto požadavky poskytují tzv. Minimum Usable SubseT (MUST), které představují minimální požadavky, které projekt garantuje k dodání. Bez dodání těchto požadavků ztrácí projekt pointu a není bezpečný. Pokud se situace vyvine, že jste schopni nalézt jiné a bolestivější řešení požadavku, pak se jedná o Should Have a Could Have požadavky. Změna kategoraziace na Should Have a Could Have neznamená, že nebude požadavek dodán, jen toto dodání není zaručeno.
- **S**hould Have (měl by mít) – jsou pro projekt důležité, ale ne zásadní. Když se v projektu vynechají, nemá to fatální dopad v podobě zrušení projektu. Jedním ze způsobů, jak odlišit důležitost Should Have požadavků od Could Have požadavků, je přezkoumání, jak velký dopad mělo nesplnění požadavků z hlediska obchodní hodnoty nebo zasažených osob.
- **C**ould Have (dobré mít) – lze je charakterizovat jako chtěné, žádoucí, ale méně důležité. V případě vynechání mají mírnější dopad pro projekt než Should Have. Většinou se jedná o jednotlivé vychytávky, které slouží ke spokojenosti uživatele. K realizaci těchto požadavků dochází pouze v případě nejlepšího scénáře, že projekt nemá časový skluz a stihne být dodán v termínu. V ostatních případech, kdy je projekt časově ohrožen, se Could Have požadavky vynechávají.
- **W**on't Have this time (v tuto chvíli nebude mít) – projektový tým si odsouhlasí, že se bez těchto požadavků obejde. Díky tomu je zabráněno opětovnému zavedení později. Dále napomáhají řídit očekávání, že nějaké požadavky zkrátka nebudou do projektu implementovány ('MoSCoW Prioritisation', c2019).



Obr. 3: MoSCoW – vyvážení priorit

Zdroj: Výstřižek z ('MoSCoW Prioritisation', c2019)

Schéma vyobrazené výše nám udává doporučené procentuální rozdělení mezi jednotlivé skupiny požadavků. Pro dosažení úspěšného projektu by požadavky Must Have neměly přesahovat 60 % celkového plánovaného úsilí na projektu. Za určitých okolností může být procento úsilí u skupiny Must Have nižší než 60 % a brána jako výhoda za předpokladu, že projektový tým poskytuje co největší flexibilitu k optimalizaci hodnoty skrze více požadavků Should Have.

Obecně tedy můžeme říct, že požadavky Must Have a Should Have by měli tvořit maximálně 80 % předpokládaného úsilí projektu. Zbylých 20 % vyplňují požadavky Could Have, které jsou zrealizovány za předpokladu dokončení Must Have a Should Have požadavků.

Na přesném rozdělení plánovaného úsilí se musí dohodnout každý projektový tým. Efektivní MoSCoW metoda je o vyvážení rizik a předvídatelnosti u každého projektu ('MoSCoW Prioritisation', c2019).

3 Tradiční vývoj softwaru

Ačkoliv v současné době je tradiční přístup spíše na ústupu, tak si zaslouží svoji kapitolu, protože na vývoj softwaru měl významný podíl. Představíme si nejznámější a dodnes používaný Vodopádový model životního cyklu, dále Boehmův spirálový model, který začal využívat iterativních prvků a metodiku Rational Unified Process.

Tradiční nazýváme proto, abychom dokázali odlišit od dnes modernějšího agilního přístupu. To ovšem neznamená, že metodika, která vznikne dnes, je automaticky agilní. Datum vzniku metodiky není rozhodující, záleží, jaký přístup metodika vyznává. Typickou vlastností tradičního přístupu je co nejpřesnější určení termínů a jednotlivých požadavků. Každá osoba v týmu zastává danou roli, má tak svoji specializaci, od které se neodchyluje a nevměšuje se do činností jiných spolupracovníků. Příkladem takových rolí v tradičních metodikách může být architekt, analytik, programátor, tester, projektový manažer, grafik, kodér, databázový specialista apod. Jak už to bývá, toto rozdělení má své pro i proti. Na jedné straně je dobré, že každý je specialistou ve své konkrétní činnosti, ale na straně druhé vázne komunikace a kooperace mnoha lidí, která je nutná pro splnění úkolu, a tak je celý proces prodlužován.

Další důraz při vývoji tradičním způsobem je kladen na pečlivě vedenou dokumentaci od komunikace se zákazníkem, sběru požadavků, vývoje softwaru, testování, až po předávání a údržbu. Vývojáři tak poměrně značnou dobu vyčkávají, než dostanou návrh toho, jak by měl systém vypadat. Rovněž zákazník čeká delší dobu na první verze systému.

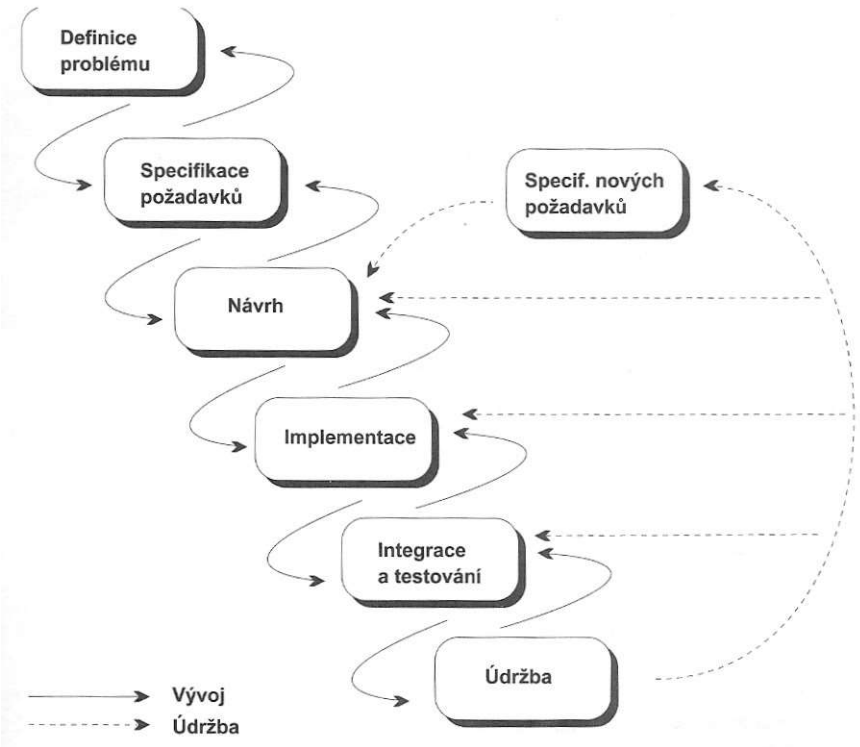
Ač to může vypadat, že tradiční přístup je již nepoužitelný, není tomu tak. Tomu nasvědčuje fakt, že je stále hojně využíván často v různých podobách hybridu. Projektoví manažeři totiž oceňují u tradičních metod jejich řád, pořádek, jistotu a předvídatelnost. *„Funkcionalita je dána před zahájením implementace, změny jsou minimální a podmíněny tím, že projdou poměrně značným sítem byrokracie, schvalování a hodnocení“* (Myslín, 2016).

3.1 Vodopádový model životního cyklu

Vodopádový model pochází z roku 1970 a mezi modely a metodikami vývoje softwaru je považován za stařík. V době svého vzniku představoval převrat ve způsobu, jak vyvíjet software. Z tohoto modelu se mnohdy stále vychází v různých modifikacích dodnes.

Hlavním charakteristickým znakem vodopádového modelu je sekvenčnost jednotlivých fází. Nutností je předem naplánovat všechny aktivity procesu a poté je možné spustit životní cyklus softwaru, kde po skončení jedné fáze lze zahájit následující fázi vývoje. Nelze spustit dvě či více fází najednou. To má svou výhodu, že v průběhu projektu vždy víme, v jaké fázi se nacházíme a co máme dělat. Pro svou jednoduchost byl a je stále populární. Co už však model postrádá, jsou iterace, nepočítá s prototypy, a především chybí průběžná komunikace se zákazníkem.

Při pohledu na obr. 4 vidíme fáze životního cyklu vývoje softwaru, který postupuje formou kaskády, a proto model dostal přívlastek vodopádový (Kadlec, 2004; Sommerville, 2013).



Obr. 4: Schéma vodopádového životního cyklu

Zdroj: (Kadlec, 2004, str. 57)

Dle schématu na předchozí straně vidíme jakési možné kroky ve vývoji i opačnou stranou. Po vodopádovém modelu se tak můžeme pohybovat vždy jen o krok dopředu či dozadu. Je to jediný flexibilní prvek tohoto modelu. Pokud tedy zjistíme ve fázi implementace, že něco v návrhu nesedí nebo některý prvek architektury je obtížné zrealizovat, musíme se vrátit o krok zpět do fáze návrhu a podniknout nápravu. S každou provedenou úpravou v jakékoliv fázi souvisí i následná úprava dokumentace dané fáze. Bez schválených a podepsaných dokumentů se vodopádový model neobejde (Kadlec, 2004).

Základní fáze vodopádového modelu:

1) **Definice problému** – jedná se o první interakci se zákazníkem nejlépe na jeho pracovišti, kde bude později systém nasazen. Cílem je pochopit záměr zákazníka, vcítit se do jeho situace a porozumět jeho potřebám a požadavkům. Vzhledem k tomu, že vodopádový model neobsahuje moc komunikace se zákazníkem, tak je zapotřebí s ním co nejvíce mluvit v této fázi. Měli bychom se snažit pochopit, proč systém potřebuje, v čem mu usnadní práci, co od něj očekává a jak problém řeší dosud bez požadovaného systému. Informace o zákazníkovi shrnuje dokument Úvodní studie.
2) **Analýza a specifikace požadavků** – v první části této fáze analyzujeme problém zákazníka, který by měl být odstraněn nasazením systému. Potřebujeme zjistit, co by měl systém dělat a jak. Případné nepochopení zákazníkových požadavků mívá fatální důsledky. Je tedy vhodné hovořit zákazníkovým jazykem, nikoliv programátorskou hantýrkou, která by zákazníkovi akorát zamotala hlavu. Naopak od zákazníka je vyžadující perfektní součinnost, protože k dalšímu slovu se dostane až při převzetí výsledného produktu. Z důkladné analýzy dostáváme dokument Specifikace požadavků, který popisuje, co by měl systém umět, nikoliv, jak budou požadavky implementovány. Řeč o programovacím jazyce v této fáze stále není na pořadu dne.
3) **Návrh a vytvoření architektury** – v této fázi je hlavním úkolem projít detailně specifikaci požadavků a na jejím základě navrhnout celkovou architekturu systému s přidělenými požadavky na hardware a software. Při tvoření návrhu

musíme vzít v potaz dostupné prostředky (ať už lidské, finanční, časové nebo technické a technologické). Zároveň se řídíme dle pravidel, které jsou nastaveny ve vývojovém týmu a uplatňujeme své zkušenosti a zvyklosti. Pouze na základě dobře odvedené práce analytika a architekta je programátor schopen celý návrh zrealizovat ve funkční systém.

4) **Implementace** – v ideální případě jsou specifikované požadavky přepsány do zdrojového kódu. Dokumenty by měly systém natolik dobře popisovat, aby programátoři věděli, co mají dělat. Jaký si zvolí programovací jazyk je čistě na rozhodnutí vývojového týmu.
5) **Integrace a testování** – cílem testování nemůže být nic jiného než ověření, že aplikace funguje správně a splňuje zákazníkovy požadavky. Integrují se a testují jednotlivé programové jednotky jako kompletní systém. Po úspěšném testování je systém připraven k předání zákazníkovi.
6) **Provoz a údržba** – probíhá instalace systému a zákazník přebírá finální produkt k používání. Tím nám práce nekončí a nastává nekonečná fáze údržby, ve které opravujeme chyby z předchozích fází, kde byly přehlédnuty. Dále zlepšujeme implementaci systémových jednotek a přijímáme nové požadavky pro zjednodušení systémových služeb zákazníka (Kadlec, 2004; Sommerville, 2013).

Model se dostal do obliby vedoucích pracovníků především pro snadné řízení. Určí se harmonogram projektu a přechody mezi fázemi probíhají vždy po schválení fáze předchozí. Vedení tak může lehce kontrolovat, v jaké fázi se projekt právě nachází, jestli nedochází k časové prodlevě, zda je plněna specifikace požadavků a zda není překračován rozpočet pro projekt.

Achillovou patou vodopádového modelu je nedostačující komunikace se zákazníkem. Dodavatel systému přichází do kontaktu se zákazníkem pouze v úvodu při definování požadavků na aplikaci a poté až na konci při předání hotové aplikace. Pokud tedy analytik nesprávně uchopí zákazníkovy potřeby, nebo je zákazník chybně formuluje, nedorozumění tak vyplyne na povrch až při předání produktu, čímž způsobí především ztráty finančního charakteru. Pouze tedy za předpokladu, že v celém cyklu nejsou

průběžně dodávány prototypy verzí, které by chyby a případné odklonění od definovaných požadavků odhalily.

Ačkoliv vodopádový model může užívat zaslouženého důchodu, nikdo mu již nevezme prvenství a revoluční pojetí ve vývoji softwaru. Díky tomuto modelu bylo realizováno nespočet softwarových projektů a dodnes se s jeho prvky můžeme setkat (Kadlec, 2004).

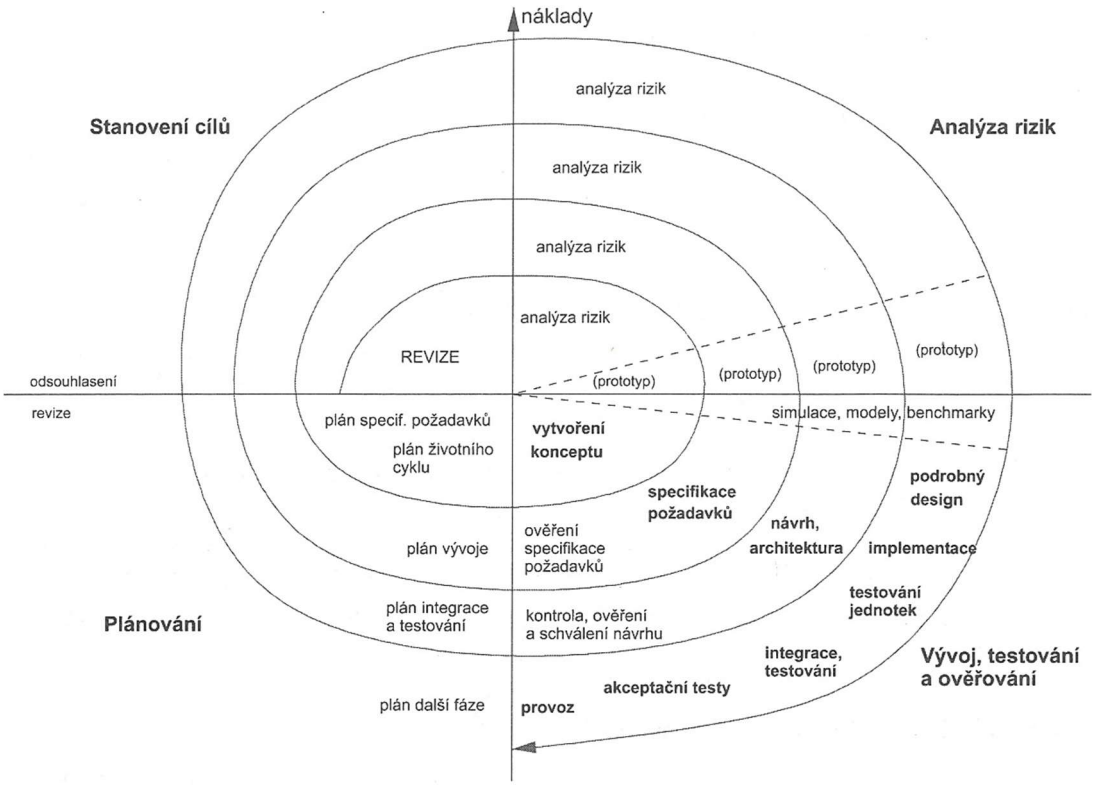
3.2 Boehmův spirálový model

Na nedostatky vodopádového modelu zareagoval Barry Boehm, který obohatil vývojářský svět o spirálový model životního cyklu v roce 1986, kdy poprvé popsal spirálový model, veřejnosti svoji publikaci (A Spiral Model of Software Development and Enhancement) představil o dva roky později. Do softwarového procesu byly zařazeny dva průlomové prvky. Jedním z nich je iterativní přístup a druhým je opakovaná a důsledná analýza rizik, na které je proces založen.

Jako efektivnější cesta se ukázala být nedefinování všech požadavků na začátku projektu a začít obecnou specifikací a architektury systému. Na základě obecného rámce lze vyvinout prototyp a ten konzultovat se zákazníkem, zda se projekt ubírá správným směrem či naopak. Zpětná vazba nás usměrní v následujících krocích spirálového modelu. Jak jsme uvedli výše, vývoj využívá tzv. iterací, což jsou opakované cykly, které napomáhají zpřesňovat specifikaci a zdokonalovat produkt dle konkrétní situace.

Na začátku každé iterace je prováděna analýza všech rizik a možných problémů. Ta předpokládá možné hrozby, které by mohly ohrozit průběh projektu. Vývojářský tým tak není příliš zaskočen, když potenciální problém nastane. Důsledkem každé analýzy rizik je rozhodnuto, kam bude směřovat další vývoj aplikace, jak bude vypadat konkrétní postup, nebo může dojít i k zastavení celého projektu.

Pro lepší představu je spirálový model znázorněn na obr. 5, na kterém vidíme sekvenci cyklů. Jednoduše můžeme říci, že čím robustnější spirála, tedy čím více cyklů model má, tím vyšší jsou časové a finanční náklady (Kadlec, 2004).



Obr. 5: Schéma spirálového modelu

Zdroj: (Kadlec, 2004, str. 68)

Struktura spirálového modelu je rozdělena do 4 kvadrantů:

1) **Stanovení cílů –** levý horní kvadrant stanovuje cíle pouze pro následující iteraci. V kterémkoliv cyklu této fáze se určují cíle (výkonnostní požadavky, požadavky na funkčnost), alternativy (může být stanoveno více možných způsobů designu modulu), omezující podmínky (cena, plán projektu) následujícího postupu.
2) **Analýza rizik –** hodnotí se alternativy a požadavky vzešlé z předchozího kvadrantu. V případě, že se objeví nějaké nepřesně definované oblasti, které by mohly představovat rizika, je zapotřebí formulovat strategie eliminující tato rizika. Příkladem může být sestrojení analytického modelu, vytvoření prototypu, simulace nebo dotaz na zákazníka.
3) **Vývoj, testování a ověřování –** dochází ke transformaci náplně příslušné iterace. Z předchozích analýz je postupně provedena specifikace požadavků, podrobný návrh architektury, implementace a otestování aplikace.

4) **Plánování –** dochází k přezkoumání projektu a přijímá se rozhodnutí, zda projekt bude rozšířen o další smyčku spirály. Pokud ano, je sestaven plán pro další fázi projektu (Kadlec, 2004; Sommerville, 2013).

Spirála obsahuje tolik životních cyklů, kolik jen potřebuje. Čím více se spirála natahuje, tím je dodání aplikace prodlužováno a dochází k prodražení celkové ceny. Můžeme definovat 4 základní cykly. U prvního cyklu jde především o to si definovat globální rizika, zpracovat koncept vývoje a rozhodnout o konkrétních metodách. Druhý cyklus specifikuje požadavky na systém. Cílem třetího cyklu je navrhnout architekturu systému. Celý cyklus je zakončen čtvrtou fází, kde probíhá implementace, testování a integrace. Všechny cykly obsahují nezbytné fáze plánování, určení cílů, alternativ a omezujících podmínek a analýzu rizik.

Původní originální spirálový model neobsahoval žádné formální náležitosti ani dokumenty. To Boehm v roce 1996 napravil a definoval tři základní mezníky:

- cíle životního cyklu (Life Cycle Objectives, LCO)
- architektura životního cyklu (Life Cycle Architecture, LCA)
- počáteční funkční vlastnosti (Initial Operational Capability, IOC)

Dokumentace neslouží pouze pro potřeby vývojového týmu, ale je konzultována a poskytována zákazníkovi (Kadlec, 2004).

3.3 Metodika Rational Unified Process

Rational Unified Process, zkráceně RUP, je objektově orientovaná, iterativní metodika vývoje softwaru. RUP lze pořídit jako sadu internetových stránek tvořících znalostní bázi nebo bylo vydáno nepřeberné množství knih a odborných článků o této metodice. Ač metodika byla vyvinuta společností Rational Software, tak nyní metodika spadá pod společnost IBM, která firmu Rational zakoupila (Kadlec, 2004).

RUP poskytuje disciplinovaný přístup k přiřazování úkolů a odpovědností v rámci vývojářského podniku. Podobně jako u většiny metodik je cílem kvalitní výroba softwaru, která bude naplňovat zákazníkovy potřeby v rámci předvídatelného rozpočtu a časového plánu. RUP slouží jako návod, jak efektivně používat Unified Modeling Language (UML). UML je grafický jazyk, který nám umožňuje jasně komunikovat

požadavky, architektury a návrhy systému. RUP je zároveň podporován nástroji, které automatizují velké části procesu (*Rational Unified Process; Best Practices for Software Development Teams*, 1998).

RUP zvyšuje produktivitu týmu tím, že každému členovi týmu poskytuje snadný přístup ke znalostní bázi s přesnými pokyny, šablonami a nástroji pro všechny důležité vývojové aktivity. Metodika definuje, jak efektivně nasadit následujících šest obecných osvědčených postupů při vývoji softwaru:

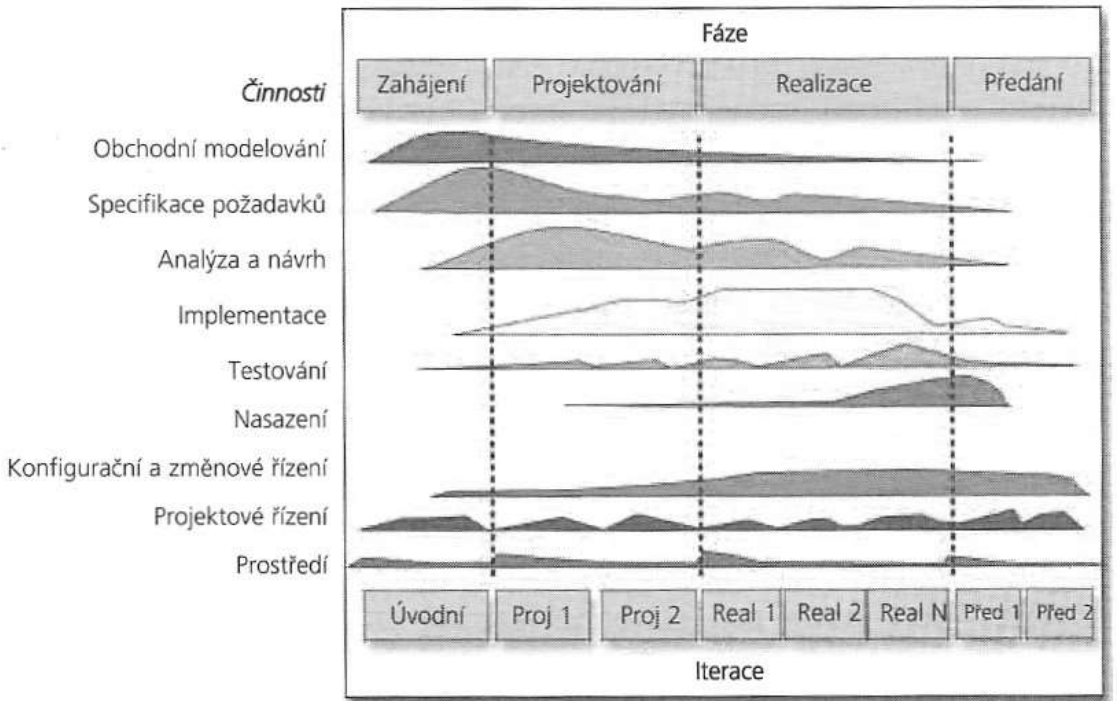
- **Iterativní vývoj softwaru**
- **Správa a řízení požadavků**
- **Použití komponentové architektury**
- **Vizuální modelování softwaru**
- **Průběžné zajišťování a ověřování kvality**
- **Řízení změn**

RUP proces popisuje **kdo, co, kdy** a **jak** dělá a je reprezentován pomocí těchto čtyř nejdůležitějších elementů modelování:

1) **Pracovníci** (Workers) – odpovídají na otázku **kdo**. Pracovník definuje chování a odpovědnosti jednotlivce nebo skupiny jednotlivců, kteří pracují jako tým. V metodice je pracovník považován s nadsázkou jako „klobouk“, který může být jednotlivci nošen v projektu. Každému jednotlivci může být nasazeno více různých klobouků najednou. Pracovník je brán jako role, která říká, jak by práce měla být správně vykonána.
2) **Činnosti** (Activities) – odpovídají na otázku **jak**. Činnost pracovníka je jednotkou práce, kterou by měla být jednotlivcem provedena. Každá činnost je přiřazena konkrétnímu pracovníkovi. Dále má jasný účel, většinou jako vytvoření nebo aktualizace některého meziproduktu.
3) **Meziprodukty** (Artifacts) – odpovídají na otázku **co**. Meziprodukt je část informace, která je vyráběna, upravována nebo používána v procesu. Artefakty jsou hmatatelnými výsledky projektu. Mohou nabývat různých forem jako je: model, element modelu, dokument, zdrojový kód nebo spustitelná aplikace.

4) **Pracovní procesy** (Workflows) – odpovídají na otázku **kdy**. K popsání celého procesu nestačí výčet všech pracovníků, činností a meziproduktů. K tomu potřebujeme způsob, jak popsat smysluplně posloupnosti činností a stanovit interakce mezi pracovníky, které vyústí k vytvoření hodnotného produktu (*Rational Unified Process; Best Practices for Software Development Teams*, 1998).

Obr. 6 znázorňuje jeden vývojový cyklus v metodice RUP. Jeden právě proto, že celý cyklus může být libovolně opakován. Proces je rozdělen mezi dvěma osami. Horizontální osa představuje čas a je vyjádřena v podobě fází a iterací. Vertikální osa představuje statický aspekt procesu, tj. jak je popsán z hlediska pracovníků, činností, meziproduktů a pracovních procesů (*Rational Unified Process; Best Practices for Software Development Teams*, 1998).



Obr. 6: Vývojový cyklus v metodice RUP

Zdroj: (Kadlec, 2004, str. 89)

Rational Unified Process rozděluje jeden vývojový cyklus do čtyř po sobě následujících fází:

1) **Zahájení** (Inception) – snaha o dosažení shody všech vymezených cílů projektu napříč všemi účastněnými entitami. Dohoda by měla zahrnovat kritéria úspěšnosti, hodnocení rizik, odhadované finanční potřebné zdroje a hrubý časový harmonogram.
2) **Projektování** (Elaboration) – cílem je analyzovat rizikové oblasti, specifikovat vlastnosti a navrhnout vhodnou architekturu systému.
3) **Realizace** (Construction) – během realizace jsou všechny komponenty a funkce aplikace vyvíjeny a integrovány do produktu. Následně jsou všechny funkce důkladně otestovány.
4) **Předání** (Transition) – účelem je předání softwarového produktu koncovému uživateli a jeho školení. Obvykle dochází k více iteracím, protože se objeví nějaké nedostatky, které jsou potřeba doladit před nasazením další verze.

První vývojový cyklus je nazýván úvodní vývojový cyklus. Po jeho dokončení docílíme funkčního softwarového systému. První cyklus je dále rozšiřován o další vývojové cykly, které jsou nazývány jako rozvíjející cykly. Počet rozvíjejících cyklů není nijak omezen a záleží pouze na robustnosti zadání projektu (*Rational Unified Process; Best Practices for Software Development Teams*, 1998).

„RUP podporuje cyklický vývoj, iterativní a inkrementální přístup, objektově orientované vidění světa a modelování v UML.“ Je hoden spíše pro projekty většího rázu, velké firmy a software developerské týmy, které umí precizně stanovit a důsledně zdokumentovat vývojový proces (Kadlec, 2004).

4 Agilní metodiky vývoje softwaru

Konečně se dostáváme k hlavní náplni této práce, a to k agilnímu pojetí vývoje softwaru. Tato kapitola má za úkol nás seznámit, co vůbec slovo agilní znamená, jaké hodnoty agilní vývoj zastává a dojde k popsání nejpoužívanějších metodik v praxi.

Pod pojmem agilní si lze představit synonyma jako dynamický, hbitý, interaktivní, přizpůsobivý, iterativní, zábavný etc. Pokud si organizace zvolí jít agilní cestou, musí upřednostnit jiné hodnoty a začít žít agilní filosofii. Nefunguje to jako u kuchařky, že byste si koupili knihu a tupě postupovali krok za krokem. Agilním nestačí být, musíte tak i myslet, a především se agilně chovat. Celý tým dělá v danou chvíli to, co má smysl a nejlépe jak umí. Vývojářský tým si modifikuje pravidla hry dle vlastní potřeby tak, aby se cítil dobře a mohl být co nejvíce produktivní, efektivní a dodal zákazníkovi kvalitní systém v co nejkratší dobu.

Agilní přístup se snaží o eliminaci zbytečné byrokracie, zbytečného dokumentování každé sebeméně významné události či aktivity, o zjednodušení procesů změny. Bylo by chybou spojovat agilní vývoj s rysy anarchistického chování. Agilně správně řízené projekty by měly vykazovat všechny znaky toho, že vámi využívaný softwarový proces je minimálně na třetí, lépe však na vyšší úrovni modelu vyspělosti dovedností (CCM). V agilním přístupu najdeme stejný pořádek jako v tradičním pojetí. Jedná se pouze o jiný přístup k rozumně chápaným procesům, jiný přístup k efektivitě. Lidé, kteří správně pochopili agilní myšlení a využili jej ve vývoji softwaru, si nemohou vynachválit tento přístup a odprošťují se od názorů s anarchistickými prvky. Jak již to bývá, tak i agilní metodiky vzešly do módy a tímto trendem se chce ubírat většina firem. Častou chybou však je, že se implementování agilního přístupu chopí lidé, kteří mu nerozumí a snaží se je aplikovat na nevhodné projekty, u kterých se zkrátka příslušná metodika nehodí. Můžeme při takovém selhání projektu vedený pomocí agilního vývoje vůbec hovořit o selhání agilního vývoje? Určitě ne, ve většině případů je nástroj nesprávně pojatý a jedná se spíše o selhání lidského faktoru.

Alfou a omegou agilního přístupu je kvalitní dodávka softwaru, zbytek je pouhým nástrojem. Základním stavebním kamenem celého agilního přístupu je Agilní manifest (Agile Manifesto) (Myslín, 2016; Šochová & Kunce, 2014).

4.1 Agilní manifest

U zrodu Agilního manifestu stála skupina 17 zkušených softwarových inženýrů, kterým nebyl lhostejný způsob vývoje softwaru a cítili tak potřebnou změnu. Všichni se sešli v roce 2001 v lyžařském středisku Snowbird v Utahu, aby si společně zalyžovali, ale především, aby našli společné hodnoty, které vyznávají. Výsledkem se stal Manifest Agilního vývoje software (Manifesto for Agile Software Development) obsahující čtyři hodnotné výroky, které tvoří základ pro agilní vývoj softwaru. V následujících měsících autoři rozšířili původní čtyři myšlenky o dvanáct principů agilního vývoje softwaru (‘Agile 101’, c2019, p. 101).

Originální znění bylo zatím přeloženo celkově do 68 jazyků včetně jazyka českého (‘Manifesto for Agile Software Development’, 2001).

„Objevujeme lepší způsoby vývoje software tím,
že jej tvoříme a pomáháme při jeho tvorbě ostatním.
Při této práci jsme dospěli k těmto hodnotám:

- **Jednotlivci a interakce** před procesy a nástroji
- **Fungující software** před vyčerpávající dokumentací
- **Spolupráce se zákazníkem** před vyjednáváním o smlouvě
- **Reagování na změny** před dodržováním plánu

Jakkoliv jsou body napravo hodnotné,
bodů nalevo si ceníme více.“

Výše uvedené čtyři základní hodnoty doprovází následujících dvanáct principů, kterými by se agilně vyvíjející firma měla řídit:

1) Nejvyšší prioritou je vyslyšet slova zákazníka brzkým a průběžným dodáváním kvalitního softwaru.
2) Veškeré změny v požadavcích jsou vítány, a to i v průběhu vývoje. Agilní procesy převedou změny v konkurenční výhodu zákazníka.

3) Dodávat fungující software v intervalech týdnů až měsíců. Preferuje se kratší doba dodání.
4) Lidé z obchodu musí spolupracovat s vývojáři na každodenní bázi po celou dobu projektu.
5) Utvářet tým kolem motivovaných jedinců. Dodat jim podporu, vhodné prostředí a také důvěru, že práci odvedou na maximum.
6) Nejefektivnější cestou, jak si vývojový tým může z vnějšku i uvnitř týmu vyměňovat informace, je osobní konverzace.
7) Fungující software je primárním měřítkem progresu vývoje.
8) Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet nasazené tempo.
9) Neustále klást pozornost technické výjimečnosti a dobrému návrhu zvyšuje agilitu.
10) V jednoduchosti je krása, snaha maximalizovat množství nevykonané práce.
11) Nejlepší návrhy, požadavky a architektury vychází ze samo-organizujících se týmů.
12) Vývojový tým se pravidelně zabývá otázkou, jak být ještě efektivnějším, na jejímž základě modifikuje své chování a postupy (‘Principles behind the Agile Manifesto’).

4.2 Lean Software Development

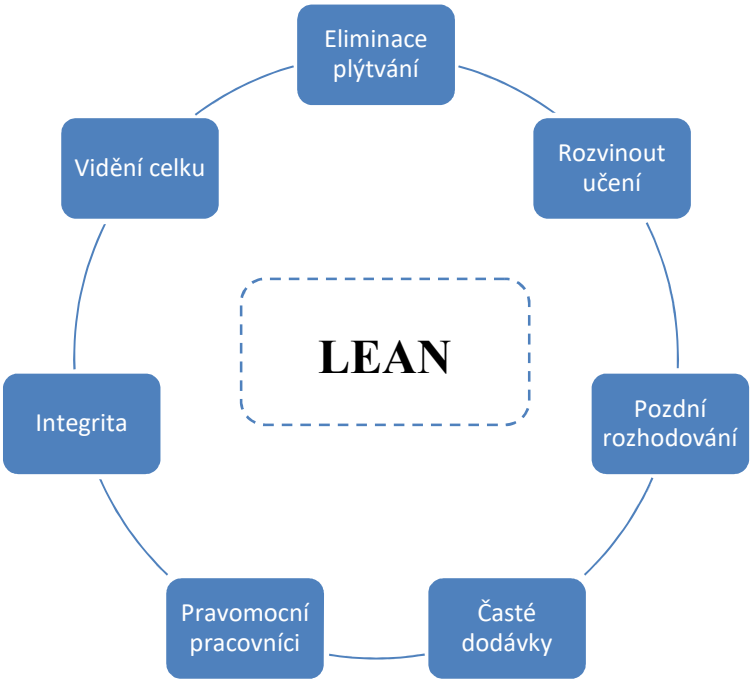
Kořeny Lean metodiky pocházejí z výrobního odvětví, konkrétně automobilního. Metodika Lean Development byla postavena na myšlenkách výrobního procesu Lean Manufacturing. S tímto procesem je úzce spjat Číňan Taiichi Ohno, který se zabýval produktivitou a efektivitou výrobních procesů v japonské automobilce Toyota. Cíl byl prostý – eliminovat vše zbytečné, co v průběhu výroby vznikalo a snižovalo efektivitu (Kadlec, 2004)

Lean v překladu znamená štíhlý. Metodika Lean Development tedy pojednává o štíhlé výrobě, kde je hlavním záměrem odstranit všechny nepotřebné a zdržující kroky bránící rychlejší výrobě. Úspěchu docílíme děláním věcí pouze, když jsou potřeba – tzv. just in time. Podobně jako „agile“, tak „lean“ si nelze představovat jako kuchařku a slepě

implementovat jeho principy. V obou případech je nutné pochopit celou filosofii, koneckonců oba principy se vzájemně podobají a prolínají (Šochová & Kunce, 2014)

4.2.1 Principy metodiky Lean Development

Jak jsme se mohli dočíst o pár řádků výše, samotná metodika původně sloužila ve výrobním odvětví. Základních deset myšlenek procesu Lean Manufacturing se zalíbilo manželům Poppendieckovým, kteří původní pravidla aplikovali na vývoj softwaru a nechali tak vzniknout metodiku Lean Software Development. Ta je založena na sedmi konkrétních principech (viz. obr. 7). Principy jsou doplněny o nástroje, které mají být průvodcem při realizaci v praxi (Kadlec, 2004).



Obr. 7: Principy metodiky Lean Software Development

Zdroj: vlastní zpracování na základě: (Kadlec, 2004, s. 173)

1. **Eliminace plýtvání** – snaha vyhnout se úkonům, které nemají smysl. Identifikovat pro nás důležité hodnoty a stát se tak efektivnějším.
2. **Rozvinout učení** – pokud opakujeme stejný proces dokola bez zpětné vazby, může se stát, že vykonáváme jednu a tu samou chybu opakovaně. Cílem je ověřovat vykonané procesy a poučit se do následujících iterací.

3. **Rozhodovat co nejpozději** – s odkladem rozhodnutí máme možnost načerpat co nejvíce informací. Záměrem je mít zadní vrátka v případě, že dojde k zásadním změnám požadavků.
4. **Dodávat co nejrychleji** – ač fázi rozhodování můžeme protáhnout, fungující software musíme dodat co nejrychleji. Díky tomu dostaneme dříve zpětnou vazbu a v následující iteraci ji můžeme promítnout. Zároveň při rychlém dodání produktu nemá zákazník času nazbyt modifikovat své požadavky.
5. **Pravomocní pracovníci** – dodat svému týmu důvěru a zodpovědnost. Pracovníci v týmu budou tak motivovanější.
6. **Integrita –** úspěšný vývojový proces musí být jednotný. Začínat by měl specifikací požadavků a pokračovat úkony, které jsou potřeba pro splnění zadání. Nástroje integrity kladou důraz na testování a zabývájí se refaktorizací.
7. **Vidět celek** – k chybám pochopitelně bude vždy docházet, důležité je se z nich poučit. *„Think big, act small, fail fast; learn rapidly“* – v překladu: Myslete dopředu, začněte u malých věcí, ty vyhodnoťte a poučte se z nich rychle. Není v našem zájmu systém rozdělovat na malé části a ty optimalizovat, důležité je zajistit celkový běh systému.

Jednou z metod, které využívají výše uvedené principy je Kanban. Ten se kromě „leanu“ opírá také o agilní zásady a Kanbanu se budeme věnovat v jedné z následujících kapitol (Kadlec, 2004; Poppendieck & Poppendieck, 2010; Šochová & Kunce, 2014).

4.3 Artefakty agilního programování

V následující podkapitole bude popsáno pět agilních praktik programování, bez kterých by se samotné metodiky neobešly. I když většina artefaktů pochází z metodiky Extreme Programming, tak jsou hojně využívány i u jiných (Šochová & Kunce, 2014).

4.3.1 Pair Programming

Již z názvu párové programování můžeme tušit, že se bude jednat o techniku, které se účastní dva programátoři. Společně sdílejí jednu obrazovku, klávesnici a myš. Ten programátor, co zrovna píše kód je nazýván jako „řidič“ a jeho napomáhající kolega je nazýván jako „navigátor“. Programátoři se velice často střídají u klávesnice. Jeden tak píše kód a druhý přemýšlí a kontroluje ho. Ve dvou objeví rychleji chyby a ihned je opraví (‘Pair Programming’, c2019).

4.3.2 Review

Další rychle účinnou praktikou je review. Jedná se o provedení kontroly toho, co již někdo udělal. Review neplatí pouze u psaní kódu, ale také u testování, psaní dokumentace, designu a analýzy, zkrátka u všeho. Nejpoužívanější je code review, kdy podobně jako u párového programování je cílem odhalit chyby co nejrychleji, což ve výsledku přinese úsporu času = peněz. Review provádí libovolný člen týmu. Při kontrolování seniorem se dá předpokládat, vzhledem k jeho zkušenostem, že chybu nalezne sám. Pokud review dělá mladší člen týmu, tak mu změny v kódu musí vysvětlit ten, kdo je psal. Tým si tak rychle předává své znalosti od zkušenějších po méně zkušenější (Šochová & Kunce, 2014).

4.3.3 Coding Standard

Coding Standard je zavedení stejného stylu psaní kódu pro všechny. Tato praktika může vyvolávat otázky, k čemu je to dobré, hlavní přeci je, když rozumí kódu programátor. Takto to v agilním světě nefunguje a fungovat nemůže, protože jedním z dalších artefaktů agilního programování je sdílený kód, kde všichni můžou dělat změny všude. Při dodržení stejného stylu tak docílíme přehlednosti pro všechny, i když ten kód nepsali. Zvláště ze začátku se může zdát tato praktika jako velice nepříjemná, ale je zapotřebí vydržet a změna k lepšímu se dostaví (Šochová & Kunce, 2014).

4.3.4 Continuous Integration

Průběžná integrace je agilní praktika vývoje softwaru, kde členové týmu často integrují svou práci. Obvykle se integraci věnuje každý pracovník alespoň jednou denně, což přináší vícenásobnou integraci za den. Každá integrace je ověřena automatickým buildem včetně automatického otestování, aby došlo k případnému odhalení chyb integrace co nejdříve. Po zavedení praktiky neustálého integrování zjišťuje mnoho týmů, že dochází k redukci integračních problémů a že je schopen vyvíjet software rapidně rychleji (‘Continuos Integration’, 2006).

Pro zálohu zdrojového kódu nám slouží tzv. repositor – úložiště zdrojového kódu. V něm může každý člen týmu přehledně editovat hotový kód. V případě potřeby lze zjistit, kdo repositor nejvíce využívá, nahlédnout do historie souborů a spousty dalších funkcí. Ze známých systémů pro správu verzí si můžeme uvést CVS, SVN, GIT, Team Foundation atd. (Šochová & Kunce, 2014).

4.3.5 Test Driven Development – TDD

Do češtiny lze TDD volně přeložit jako programování řízené testy. A tento název nám již napovídá, jaká je jeho hlavní náplň. TDD označuje styl programování, kdy je zapotřebí nejprve napsat test ještě před fází psaní samotného kódu. V době, kdy máme testovací kód hotov, nastává chvíle naprogramování zdrojového kódu za dodržení pravidla, že implementujeme přesně takové množství programového kódu, kolik dokáže projít testem. Z předchozích řádků víme, že kvalita testování se výrazně podepíše na celkové spokojenosti zákazníka. Zákazník je sice dobrý tester a přijde na většinu chyb, ale toho se chceme právě vyvarovat (Kadlec, 2004; 'TDD', c2019).

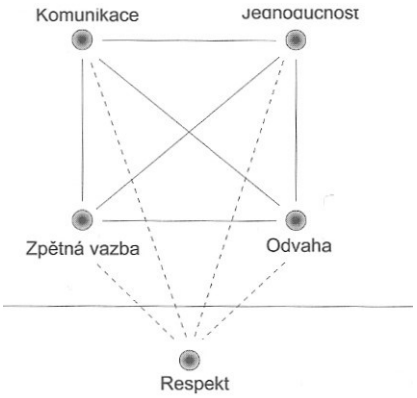
Spousta testerů odkládá testování s výmluvou, že vývojáři jim dodávají kód pozdě. Koncept TDD této výmluvě předchází a začíná s testováním ještě dříve, než je kód napsán. Tester po diskuzi s analytikem ví už na začátku, jak by se měla daná funkcionalita chovat a napíše test na danou situaci dopředu. Nemusí tak čekat, až vývojář práci započne. To umožňuje spolupracovat analytikovi, vývojáři i testerovi najednou, což přesně vyznává agilní myšlení (Šochová & Kunce, 2014).

4.4 Extreme Programming

Metodiku Extreme Programming přivedl na svět Kent Back v roce 1999 a představil v ní základní myšlenky. Extrémní programování můžeme znát také pod zkratkou XP, která je hojně využívána. *„XP je lehký, účinný, nepříliš rizikový, pružný, předvídatelný, vědecký a zábavný způsob, jak vyvíjet software."* Metodika je určena pro malé a středně velké týmy, které se často musí vypořádávat s proměnlivými nebo nejasnými požadavky. Beck se setkal se spoustou pozitivních ohlasů na tuto metodiku především díky tomu, že užívá „zdravý rozum". Proč tedy extrémní? Přívlastek extrémní využívá proto, že běžně známé principy a postupy vyšperkuje do extrémů. Například jestliže se osvědčuje testování, budou všichni nepřetržitě testovat. Zaběhnuté zvyklosti tak přesouvá do vyšších dimenzí (Beck & Makovec, 2002; Kadlec, 2004).

4.4.1 Čtyři hodnoty XP

Vývojový tým vyznávající Extreme Programming může být úspěšný pouze za předpokladu dodržování čtyř hodnot, na kterých metodika staví. Dle obr. 8 vidíme, že se jedná o komunikaci, jednoduchost, zpětnou vazbu a odvahu. Jak ale dále můžeme vidět, tak pod nimi leží pátá, podprahová hodnota – respekt (Beck & Makovec, 2002; Kadlec, 2004).



Obr. 8: Základní hodnoty v XP

Zdroj: (Kadlec, 2004, str. 128)

- **Komunikace –** kdykoliv se objeví nějaký problém nebo zpoždění, tak je většinou příčina ve špatné komunikaci. Obvykle určitá osoba zapomene, nebo se bojí sdělit důležitou informaci někomu jinému. XP má snahu udržet aktivní komunikační toky na všech frontách, využívá k tomu kouče, který odhaluje výpadky komunikace a snaží se je obnovit na správnou úroveň.
- **Jednoduchost –** kouč XP se svého týmu ptá: „Co je nejjednodušší věc, která by ještě mohla fungovat?“ Nepřemýšlíme, co bude zítra, za týden a příští měsíc. Lepší je implementovat jednoduchou věc hned a zaplatit o něco více za případnou změnu než vymyslet složitou věc, která se nemusí vůbec využít.
- **Zpětná vazba –** neboli feedback zprostředkovává informace o aktuálním stavu vyvíjeného produktu, o průběhu vývoje, o situaci ve vývojovém týmu, o požadavcích zákazníka a o dalších věcech důležitých pro vývoj. XP uvádí, že programátoři trpí nemocí z povolání a to optimismem, že jejich kód je napsán bezchybně a bude funkční. Tuto „nemoc“ léčí zpětná vazba, kterou může být například testování.
- **Odvaha –** může nastat situace, že z nějakého důvodu nevede cesta dál a tým musí opravit situaci za každou cenu. I kdyby to mělo znamenat zahodit

dosavadní napsaný zdrojový kód, nebo přepracovat celý návrh s architekturou. Taková odvaha v kontextu s předchozími hodnotami se stává nedocenitelnou.

- **Respekt –** členové týmu by k sobě měli chovat vzájemnou úctu a zajímat se jeden o druhého, jak by mu mohli pomoci. V případě, že se pracovníci budou chovat jako sóloví hráči, tak nemá XP šanci uspět (Beck & Makovec, 2002).

Všechny popsané hodnoty by nefungovaly bez dvanácti postupů, které vedou ke tvorbě kvalitního systému pomocí metodiky XP. Řeč je o těchto praktikách: Plánovací hra, Malé verze, Metafora, Jednoduchý návrh, Testování, Refaktorizace, Párové programování, Společné vlastnictví, Nepřetržitá integrace, Čtyřiceti hodinový týden, Zákazník na pracovišti, Standardy pro psaní zdrojového textu (Beck & Makovec, 2002).

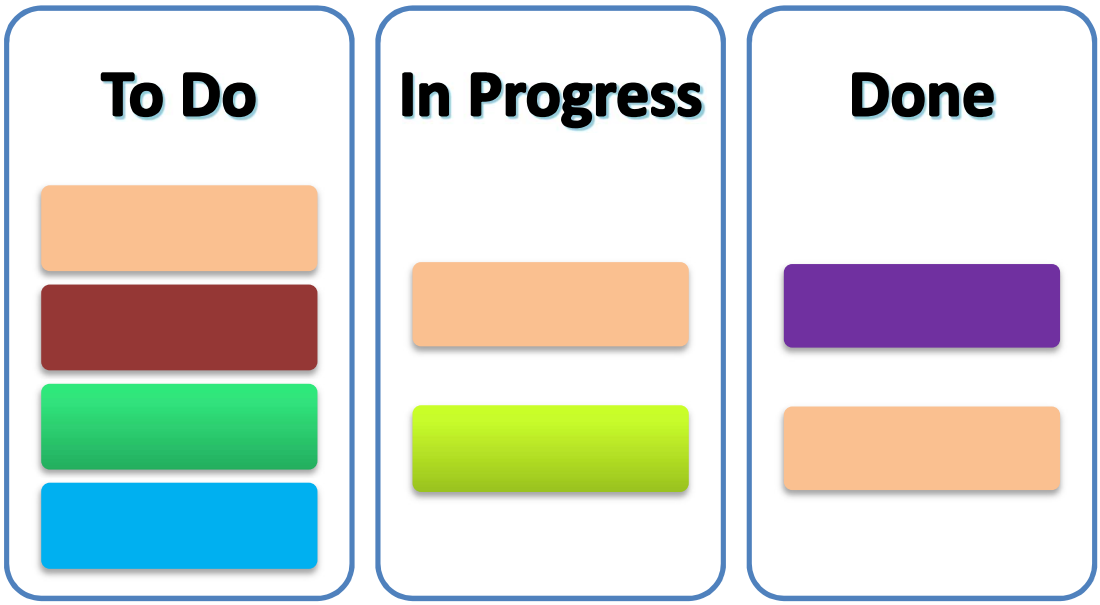
Extrémní programování není v současné době tak hojně využíváno jako v prvních deseti letech 21. století. Metodika, označována také jako XP, je v dnešní době nahrazována populárním Scrumem, nutno ale říci, že Scrum aplikuje spoustu agilních praktik, které vyšly právě z XP (Šochová & Kunce, 2014).

4.5 Kanban

Při popisu Lean Software Development jsme uvedli, že myšlenky „leanu“ využívá společně s agilními praktikami Kanban metoda. Ta zapustila své kořeny v Japonsku, kde se podle ní řídil způsob návštěvy chrámu. Před vchodem dovnitř dostal dotyčný lísteček, který si po dobu strávenou v chrámu uschoval a následně při odchodu odevzdal zpátky. Celkový počet lístečků byl kapacitně omezen, a tak v jednu chvíli nemohlo být uvnitř chrámu větší počet osob než lístečků. Později tento princip použily továrny v Japonsku pro řízení výroby (Šochová & Kunce, 2014).

Pro úspěšné zavedení je potřeba dodržet následující tři principy:

- **Vizualizovat** – vizualizace pomocí Kanban tabule
- **Omezit rozpracovanou práci** – nastavení vhodného WIP limitu (Work In Progress)
- **Minimalizovat čas průchodu** – dodávka nových funkcí co nejrychleji



Obr. 9: Kanban tabule

Zdroj: vlastní zpracování na základě: (Šochová, Kunce, 2014, s. 106)

Správné vizualizace dosáhneme zavedením tzv. Kanban tabule, což je přehledná tabule, která obsahuje lístečky s jednotlivými úkoly nacházejícími se v příslušné fázi procesu. Sloupců tabule může obsahovat více, nám pro vysvětlení postačí tři základní fáze – „To Do – In Progress – Done“. Dále je zapotřebí nadefinovat maximální počet lístečků v každé fázi. Ve fázi „To Do“ se nachází lístečky s názvy úkolů, které je potřeba udělat. Různé barevné kombinace indikují důležitost úkolu, úkoly s nejvyšší prioritou se dostávají navrch, odkud si členové týmu vybírají úkoly. Sloupec „In Progress“ zobrazuje na čem se momentálně pracuje a kým je úkol řešen. Po dokončení daného úkolu se lísteček přesouvá do sekce „Done“ a člen týmu si vybírá nový lísteček z „To Do“, nebo vypomůže na některém rozdělaném úkolu. Kamenem úrazu může být nesprávné nastavení limitu WIP (Work In Progress), tedy na kolika úkolech může vývojový tým pracovat najednou, aby byl co nejefektivnější. Při nastavení nižšího limitu minimalizujete čas průchodu funkce a o to nám v Kanbanu jde. Příliš nízký WIP limit nebude tolik efektivní a příliš obsáhlý WIP limit způsobí, že členové budou přeskakovat mezi úkoly a dodávka bude opožděna (‘Kanban’, c2019; Šochová & Kunce, 2014).

Kanban nepřiděluje žádné role a lze s ním začít kdykoliv. Běžně se využívá v call centrech a maintenance týmech. Pro vývoj softwaru je samotný Kanban nevhodný a je lepší ho doplnit jednou z agilních metodik, ať už je to Scrum či XP (Šochová & Kunce, 2014).

4.6 Scrum

Pokud tyto řádky čtou ragbyoví fanoušci snažící se najít pravidla zavedení míče zpět do hry, tak budou zklamáni. Přesto zde nejsou náhodou, „scrum“ totiž v překladu znamená skrumáž a v ragby ji známe mimo jiné pod terminologií „mlýn“, kdy se týmy protivníků přetlačují, aby dosáhly zisku uvedeného míče do hry. Tuto myšlenku poprvé rozpracovali Takeuchi a Nonaka ve svém článku pro Harvard Business Review, kde proces vývoje produktu vychází z neustálé interakce mezi řádně vybraným týmem, ve kterém jeho členové týmu spolupracují od začátku až do konce. I když v ragby jde o to společným úsilím dostat míč za požadovanou čáru, u vývoje softwaru jde o podobný princip, a to společně vyprodukovat fungující software, se kterým bude zákazník spokojen (‘Agile Practices Timeline’, c2019; Takeuchi & Nonaka, 1986).

I když Takeuchi a Nonaka značně ovlivnili metodiku Scrum, tak oficiálně ji představili v roce 1995 Ken Schwaber a Jeff Sutherland na konferenci OOPSLA. Oba dva autoři si vyzkoušeli vyvíjet software v několika společnostech a týmech, kde zjistili, že jim daná metodika nevyhovuje, a proto chtěli vyvinout flexibilnější metodiku, která rychle reaguje na měnící se požadavky a zvýší produktivitu celého týmu při procesu vývoje softwaru (‘Agile Practices Timeline’, c2019; Schwaber & Sutherland, 2017).

Scrum metodika slouží jako procesní rámec k řízení vývoje složitých produktů. Pro správné fungování Scrumu je nezbytné, aby se skládal ze Scrum týmů a přidružených rolí, činností, artefaktů a pravidel. Schwaber společně se Sutherlandem definují Scrum jako jednoduchý, srozumitelný, ale extrémně obtížný pro dokonalé zvládnutí. (Schwaber & Sutherland, 2013)

4.6.1 Role v týmu

Když se zeptáte projektového manažera, co považuje za největší kapitál, tak se budou odpovědi s největší pravděpodobností odlišovat. Může to být firemní know-how, všelijaké patenty či dostatečný finanční kapitál, se kterým se může firma pustit do

velkých projektů. Pak tu máme vlastní lidské zdroje, které Myslín ve své knize považuje za největší kapitál. Můžete mít totiž všechno zmíněné, ale bez kvalitních lidí se nepohnete kupředu. Koneckonců metodika Scrum je především o kooperaci pracovníků uvnitř týmu (Myslín, 2016).

Pigs & Chickens

Jednotlivci, kteří se podílejí na projektu řízeném metodikou Scrum, jsou rozděleny do dvou hlavních skupin. Jedná se o dvě skupiny Pigs (prasata) a Chickens (kuřata). Nejedná se o žádné hanlivé označení či zesměšnění. K tomuto rozdělení se váže příběh o sporu názvu restaurace prasete a kuřete. Kuře navrhuje Ham & Eggs (proslulý hemenex), což se pochopitelně praseti nelíbí, jelikož je v názvu obětováno a kuře pouze součástí, která mu neublíží. Na tomto obyčejném příkladu, který pochopí i malé dítko, jsou rozděleni lidé v rámci projektu (Myslín, 2016).

- Pigs – pracovníci, kteří se napřímo podílí na projektu, podílejí se na projektových pracích, jsou součástí projektového týmu a jsou odpovědni za výsledek.
- Chickens – projekt se těchto pracovníků „pouze“ týká, ale nejsou přímou součástí projektového týmu a nenesou odpovědnost za výsledek.

Která ze skupin je důležitější, se říct nedá. Obě jsou významné. Pigs tvoří projektový tým, jsou přímo zapojeni v projektu a tvoří projekt, za který nesou odpovědnost. Chickens jsou většinou označování jako koncoví uživatelé, pro které se projekt tvoří. V následujících řádcích si popíšeme jednotlivé role projektového týmu, jsou celkem tři. Tak nízký počet je z důvodu, že Scrum se snaží role minimalizovat a zaměřuje se na univerzálnost projektového týmu (Myslín, 2016).

Samoorganizovaný tým

Samoorganizovaný tým přebírá odpovědnost za svá rozhodnutí a dokáže si řešit své každodenní úkoly. Členové týmu se umí mezi sebou dohodnout, kdo bude pracovat na kterém úkolu. Pokud někdo s něčím nesouhlasí má právo se ozvat a konstruktivně to řešit s celým týmem, cílem by měla vždy být smysluplná diskuze, která vede k vzájemnému pochopení vedoucí ke změně jejich spolupráce. Dobrý tým by se měl dívat na řešené problémy uvnitř týmu pohledem „my“, nikoliv „já“. Dohled nad tímto

pravidlem udržuje tzv. ScrumMaster, který podporuje členy týmu, aby pomáhali ostatním členům a nezaměřovali se pouze na své osobní úkoly a cíle. Pokud tedy některému z členů nelze vyřešit úkol, tak ho nepřehlížíme a nenecháváme ho v tom „vykoupat“. Říct si, mám svých úkolů habaděj, to je jeho problém, je cesta do záhuby. Pouze nabídnutím pomocné ruky lze dosáhnout samoorganizovaného týmu (Šochová & Daněk, 2018).

Product Owner

Jedná se o člena projektového týmu, který je zástupcem zákazníka. Product Owner definuje funkcionality a udává směr, jak bude produkt vypadat. Product Ownera si vybírá zákazník, může si zvolit z vlastních řad, nebo si najmout člověka z externího prostředí. Obojí má své výhody a nevýhody. Definice vize je klíčová událost, mnohdy totiž ani zákazník neví, co přesně chce a stále jen opakuje, že chce zvýšit zisk. K tomu mu však pouze kvalitní software nepomůže, prvně musí mít stanovenou strategii a software mu může cestu k zisku pouze ulehčit. Pokud zákazník nemá jasnou představu, co od softwaru očekává, tak nemá cenu s ním začít rozjíždět jakýkoliv byznys. Základem je tedy slyšet od zákazníka: co se má vytvořit, proč se to má vytvořit a jaké jsou reálné možnosti.

Product Owner by měl: definovat vizi, definovat úkoly (tvorba Backlogu), definovat priority, respektovat tým a komunikovat se zákazníkem. Má také svaté právo v podobě zrušení sprintu, ať už kvůli vnitřním, nebo vnějším problémům, k takovému kroku však musí mít vážné důvody.

Další rolí Product Ownera je stanovení priorit v projektu. Kromě nastavení základních priorit také musí reagovat na případné změny okolo nich. Musí priority měnit podle toho, jak se mění svět. Rozlišujeme dva základní typy změn: změny vnitřní (např. management firmy změní obchodní cíle firmy) a změny vnější (legislativní změny, změny na trhu, změny technologické). Klíčem k úspěchu je vzájemná spolupráce projektového týmu a zákazníka. Tedy nefungující firma zákazníka logicky znamená, že projekt nebude zrealizován, alespoň tedy ne pomocí agilního vývoje.

Dále by měl Product Owner respektovat technologie a postupy, které tým pod vedením ScrumMastera zvolil. Product Owner má rovněž jedno významné právo, avšak by s ním

rozhodně neměl plýtvat či zneužívat ho. Jedná se o zrušení sprintu. Může se stát, že celý projekt ovlivní nějaké jevy z vnějšího prostředí, což je ta lepší varianta. S tou prostě Product Owner nic nenadělá a v tomto případě je zrušení sprintu vítané v celém projektovém týmu, jelikož nebude plýtváno jejich časem. Například může se jednat o některou z legislativních změn, které značně ovlivňují podnik, změnu u konkurence, technologickou změnu, nebo změnu v obchodních plánech společnosti. Horší situace nastane, když Product Owner zruší sprint kvůli vnitřním problémům, zde se jedná o závažný problém, jelikož problémy vycházejí zevnitř projektového týmu. Většinou k takovému rozhodnutí dojde, když se ukáže, že současný projektový tým není schopen pokračovat v práci. Product Owner musí mít k takovému důvodu opravdu pádný důvod (Myslín, 2016).

ScrumMaster

I když se jedná o manažerskou roli, tak si pod ScrumMasterem nepředstavujme typického šéfa, jak ho známe. Protože jak víme, tak tým ve Scrumu je samoorganizovaný a je schopen pracovat bez klasického šéfování. ScrumMaster má za úkol podporovat, motivovat a chránit všechny členy týmu. Neměl by v týmu zastávat klasického programátora, ale určitě se nic nestane, když sem tam vypomůže s kódem (Myslín, 2016).

Scrum tým

Ve Scrum týmu figurují „řadoví“ členové týmu, jedná se o vývojáře. Neměli by být zahlceni administrativní a provozní činností ve firmě, měli by se čistě soustředit na vývojářskou práci. Nedělíme role na architekta, analytika, testera a kodéra. Každý člen týmu je prostě vším. Členové týmu sice nemají přidělenou práci napřímo, ale je sestaven tzv. backlog (seznam úkolů), ze kterého si vybírají, na čem budou pracovat.

Úspěch, či selhání týmu je snadno měřitelné, buď nám software funguje, nebo nefunguje. Z neúspěchu se nelze nikterak vypovídat. K plnění úkolů je potřebná uvědomělost všech členů týmů, bez té spolupráce nebude fungovat. To znamená, že při výběru úkolů z backlogu není ideální, když si senior vývojář vybere snadný úkol a na juniora vývojáře nechá úkol těžšího rázu. Tato demokratičnost nepovede k výsledku a celý projekt nemůže být úspěšný (Myslín, 2016).

Zákazník

Ačkoliv zákazník není přímým článkem projektového týmu, tak na základě jeho satisfakce s výsledkem můžeme zhodnotit, zda byl projekt úspěšný, či nikoliv. Za zákazníka považujeme toho, kdo software objednává a platí (právní zákazník) nebo toho, kdo software bude používat (faktický zákazník).

Zákazník ve Scrumu je součástí týmu dle potřeby, není nezbytné, aby byl neustále k dispozici, ale pokud ho vývojáři potřebují, tak je dobré, aby byl v rozumný časový rámec schopen pomoci svou přítomností a součinností. Hlavní prostředník mezi zákazníkem a vývojářským týmem je a musí být Product Owner, který je jedním ze tří základních pilířů projektového týmu (Myslín, 2016).

4.6.2 Artefakty ve Scrumu

Sprint

Sprintem se ve Scrumu rozumí každá iterace opakující se při vývoji softwaru. Jedná se o největší část ve vývoji softwaru. Název sprint vychází se sportovní terminologie, kdy před samotným sprintem probíhá jistá příprava. Samotný sprint pak není příliš řízený, prostě běžíme a snažíme se vydat maximum, není čas, aby trenér vysílal pokyny zpoza dráhy. Až poté je čas na zhodnocení a vytvoření dalšího plánu.

Cílem každého sprintu je vytvořit spustitelnou aplikaci, která bude validovatelná a testovatelná. Tedy něco, co zákazník bude schopen spustit, otestovat a říci, zda mu funkcionality vyhovují, či nikoliv.

Žádná příručka vám neřekne, jak má být sprint dlouhý. Autoři Scrumu uvádějí jako délku trvání jeden měsíc, popřípadě kratší dobu, ale mohou se objevit projekty, kterým budou vyhovovat týdenní sprinty a některým projektům naopak šestitýdenní sprinty, je to opravdu individuální a délka se liší projekt od projektu. Avšak měli bychom se vyhnout jistým extrémům, není správné plánovat sprinty po dvou dnech, protože za dva dny se spíše zabýváme plánováním a vyhodnocováním a na vývojářskou práci nezbude ani moc času. Dlouhé sprinty jsou také kontraproduktivní, zákazník pak čeká delší dobu na každou verzi a případné problémy se odhalí později (Myslín, 2016; Schwaber & Sutherland, 2017).

V ideálním případě se sprint řídí sám od sebe, sice probíhají každodenní schůzky, ale ty jsou spíše informativní. Vývojáři si vybírají úkoly ze Sprint Backlogu, který by měl zůstat na konci sprintu prázdný.

Backlog

Backlog je seznam s nevyřízenými úkoly (User Stories), které je potřeba implementovat do systému. Ve Scrumu rozlišujeme dva základní backlogy:

- **Product Backlog** – u každé projektu někam zaznamenáváme, co všechno plánujeme udělat, ve Scrumu se tento seznam nazývá Product Backlog a odpovídá za něj Product Owner. I když při jeho tvoření mu pomáhají ostatní členové, tak poslední slovo má vždy Product Owner. Jednotlivé požadavky na systém (funkcionality) se zapisují formou User Story s přidělenou prioritou.
- **Sprint Backlog** – je součástí Product Backlogu. Jsou v něm zahrnuty User Stories, u kterých se tým zavázal dodat je do konce daného sprintu. Sprint Bakclog si vybírá tým dle stanovených priorit Product Ownerem. Výsledky jsou prezentovány na Sprint Review (Šochová & Kunce, 2014).

User Story

Pokud chceme vytvořit kvalitní aplikaci, tak se vývojáři neobejdou bez vytyčení požadavků svého zákazníka. Vývojáři by se měli přizpůsobit řeči zákazníka a snažit se pochopit jeho potřebám. Je to tedy uživatelský popis toho, co by měl systém dělat. Uživatelský příběh má tři základní části:

1) Definice role – musí být určeno, kdo bude příběh používat.
2) Definice cíle – jedná se o specifikaci úkolu, říkáme, jakou činnost chceme systémem vyprodukovat. Cíl by měl být krátký a jasný.
3) Definice užitku – jedná se o nepovinnou část, ale není od věci užitek do příběhu doplnit, může být pro vývojáře pomocným prvkem.

User Story může vypadat například takto: „Jako administrátor chci, abych mohl vypsat na obrazovku seznam uživatelů tak, aby bylo možné najít duplicity.“ Užitkem v tomto případě je pro administrátora vyhledání duplicit (Myslín, 2016).

4.6.3 Meetingy ve Scrumu

Plánovací schůzka

Schůzka začíná tím, že Product Owner představí celému týmu User Stories, které připravil. Tým se domlouvá, které příběhy si vybere do následujícího sprintu a dává je do Sprint Backlogu. Plánovací schůzka by neměla trvat déle než osm hodin, závisí na délce sprintu. Scrum Master řídí průběh schůzky a snaží se o to, aby účastníci pochopili její účel (Schwaber & Sutherland, 2017; Šochová & Kunce, 2014).

Denní schůzka (Daily Scrum)

Již z názvu lze usoudit, že schůzka probíhá každý den. Vzhledem k tomu, že tým je řízen demokraticky, tak členové týmu mají obvykle pružnou pracovní dobu. Není tedy rozumné začínat s denní schůzkou v sedm hodin ráno. Je potřeba, aby se členové stihli alespoň trochu připravit před začátkem. Jako optimální se zdá být devátá hodina ranní. Pokud probíhá denní schůzka standardně bez větších komplikací, tak trvá 10 až 15 minut. Hlavním úkolem je především informovanost ostatních členů týmu, kdo na jaké funkcionalitě pracuje a v jakém stádiu je jejich práce. Dalším cílem je včasné podchycení problémů a následný návrh řešení vyskytnutých problémů. Nemělo by tedy docházet k tomu, že problémy budou zameteny pod stůl. Proslov Scrum Mastera by měl být věcný a co nejvíce stručný. Dále dostávají slovo všichni členové týmu, kde by měli říct, čemu se věnovali předcházející den a co plánují dělat v daném dni. Na denní schůzce Product Owner nehraje téměř žádnou roli, je pouhým pozorovatelem. Ovšem pokud přijde s věcnou myšlenkou, může vstoupit také do diskuze. Měl by však vycítit, zda je jeho podnět opravdu vázaný k této schůzce. Výsledkem by měla být zvýšená informovanost v projektovém týmu (Myslín, 2016).

Vyhodnocení sprintu (Sprint Review)

Jedná se o slavnostní chvíli, kdy dochází k představení výsledné práce zákazníkovi. Během vyhodnocení se snažíme dostat zpětnou vazbu na dokončené User Stories. Product Owner vyhodnocení sice uvádí, ale je dobré, aby jednotlivé User Story předváděli členové týmu, oni sami vědí, jak celý systém funguje. Prezentovat by měli

pouze User Stories, které jsou dokončené a byly akceptovány Product Ownerem (Šochová & Kunce, 2014).

Retrospektiva sprintu

Retrospektiva probíhá po vyhodnocení sprintu, ale ještě před další plánovací schůzkou. Retrospektiva je efektivní technika pro získání zpětné vazby od členů týmu. Účelem retrospektivy je identifikovat hlavní aspekty uplynulého sprintu, které fungovaly dobře a které je zapotřebí vylepšit. Běžná retrospektiva vypadá tak, že tým usedne u kulatého stolu a vytvoří tzv. kolečko. Moderátor schůzku vede a požádá všechny členy, aby zodpověděli na následující otázky:

- Co se mi líbilo a v čem bych chtěl pokračovat?
- Co se mi nelíbilo a chtěl bych s tím přestat?
- Co bych zavedl nového?

Moderátor nechá poslat dokola např. tužku a vždy mluví pouze ten, kdo ji má před sebou, ostatní mlčí a se svými připomínkami čekají na závěr schůzky. Cílem je posbírat více možností řešení problému do dalšího sprintu (Schwaber & Sutherland, 2017; Šochová & Kunce, 2014).

5 Analýza současné situace vývojářského týmu

Následující kapitola představí softwarovou společnost, její procesy v týmu, zainteresované role v procesu a nástroje, které společnost používá.

5.1 Představení společnosti

Vybrána byla hradecká softwarová a poradenská společnost GIST, s.r.o., která se dočkala založení v roce 1994. V současné době firmu pohání cca 90 zaměstnanců, kteří jsou zaměření na tři produkty v jejich portfoliu:

- **Business Intelligence (BI) –** vlastní systém GIST Intelligence, který nabízí komplexní podporu controllingových činnosti.
- **Consulting –** s projekty BI je nabízeno business poradenství pro zdokonalení controllingu.
- **Software –** vývoj softwaru na míru dle požadavků zákazníka.

Organizační struktura není schválně uvedena, jelikož není pro náš záměr důležitá. Jedná se o projektově řízenou firmu, kdy jeden člověk může spolupracovat na pěti projektech najednou. Momentálně v organizaci běží zhruba dvacet projektů společně i s těmi interními.

V rámci této práce se zaměříme pouze na divizi consultingu, která je pro společnost majoritní. Divize consultingu má na starost produkty GIST Intelligence (GI) a GIST Controlling (GC), pro které vydává nové verze (buildy a patche).

5.2 Zákazník

Zákazníkem může být takřka kdokoliv, kdo vyhledává služby BI a consultingu. Především ale jejich cíloví zákazníci jsou střední podniky. Malým podnikům by se koupě systému ekonomicky nevyplatila. Jedná se o organizace, jak v komerční sféře, tak ve veřejné správě (kraje, města, nebo státem zřizované společnosti). Nemají žádné preference, do kterého segmentu jdou, i když v jejich referencích se nejčastěji objevují výrobní společnosti, avšak jejich klientem je i ZOO Dvůr Králové. Zajímavějším

zákazníkem se stává takový, který nevyžaduje systém pouze pro reportování čísel, ale chce nějakou přidanou hodnotu. Například nasazení controllingu.

Fakturovaná cena zákazníkovi se liší podle obsahu řešení. Zákazník platí licence a služby, ale konečná cena je případ od případu rozdílná, záleží na robustnosti obsahu řešení. Na celé objednávce je nejdražší realizační část.

5.3 Složení týmu

- **Produktový ředitel** – osoba jakožto v TOP managementu, se zabývá vizí podniku a strategickými cíli.
- **Vedoucí interního projektu** – zastřešuje celou verzi projektu.
- **Tester / Analytik** – role testera a analytika zastupuje jeden člověk. Navrhuje zadání pro programátora a následně testuje, zda je naprogramováno tak, jak on navrhl. Poté zpracovává dokumentaci sloužící uživatelovi. Role je takto přizpůsobena z důvodu, že daný člen týmu testuje něco, co sám navrhl a má k tomu již citové pouto a chce návrh dotáhnout dokonce.
- **Programátor –** transformuje návrh zadání do zdrojového kódu. Píše výhradně v programovacím jazyce Java.
- **Konzultant** – plní defacto roli zákazníka, který společně s týmem sedí v jedné budově. Konzultant skrze zákazníka přebírá jeho požadavky na vylepšení produktu a poté námět prezentuje na poradě. Má na starosti implementační část a školí koncového uživatele.

5.4 Fáze procesu vývoje nové verze

5.4.1 Definice interního projektu vývoje verze GIST Intelligence

O vývoji nové verze rozhoduje vedoucí projektu GIST Intelligence. Definuje členy týmu odpovědné za definici zadání, za programování, za testování a za tvorbu uživatelské dokumentace. Rozhodnutí o vývoji nové verze musí být zaznamenáno v Zápise z vývojové porady. Detailnější předmět projektu bude specifikován tím, že v Produktovém webu nebo v aplikaci Helpdesk nebo v aplikaci TFS budou označeny příslušné náměty, Analýzy požadavků a Definice zadání číslem příslušné verze SW produktu.

5.4.2 Definice zadání

Nejprve je zpracována Analýza požadavků a na produktové poradě proběhne ověření smysluplnosti jednotlivých požadavků, vyřazení některých požadavků, případně jejich přesnější formulace. Poté je zpracována detailní Definice zadání, která je již úplným zadáním pro programátora.

Vedoucí projektu schvaluje Analýzu požadavků a Definici zadání na poradách. Zadavatel analytik / tester je zodpovědný za rozepsání Definice zadání do dílčích úkolů v aplikaci TFS.

5.4.3 Vývoj části aplikace

Po schválení zadání vedoucí projektu určí programátora, který je odpovědný za vývoj příslušné části aplikace dle Definice zadání a dle dílčích úkolů v aplikaci TFS. V případě nejasností v zadání se programátor obrací na analytika / testera, který navrhl řešení a případně i na zákazníka, když to situace vyžaduje.

Programátor je povinen provést autorské testování funkčnosti dle testovacích scénářů v Definici zadání a dle vlastního úsudku. Zároveň doplní testovací scénáře o další oblasti, které by mohly být dotčeny a je třeba je otestovat.

5.4.4 Testování části aplikace

Z podkapitoly Složení týmu víme, že firma slučuje analytika a testera do jedné osoby. Analytik / tester ověřuje správnost realizace všech požadavků uvedených v Definici zadání. Testováním garantuje výsledky práce příslušného programátora. Testování může probíhat už v průběhu vývoje a musí být především provedeno na konci vývoje. Dále analytik / tester zodpovídá za „uživatelské“ testování zpravidla spojené s tvorbou dokumentace. Analytikovi / testerovi slouží k testování konzultant, který ztvárňuje roli zákazníka.

Zjištěné vady se zadávají do aplikace TFS. Programátor je odpovědný za co nejrychlejší odstraňování vad zjištěných v rámci testování části aplikace.

5.4.5 Tvorba uživatelské dokumentace

Uživatelskou dokumentaci tvoří analytik / tester již při testování části aplikace. Pokud se tak nestalo, musí vedoucí projektu obstarat její vytvoření ještě před testováním

produktu jako celku. Uživatelskou dokumentací se rozumí zpravidla Help (případně jiná dokumentace definovaná smlouvou se zákazníkem). Vytvořená uživatelská dokumentace je testována zároveň s produktem při testování softwarového produktu jako celku.

5.4.6 Testování verze jako celku

Testování verze jako celku musí zabezpečit jednak testování platforem a jednak testování na reálných databázích vybraných zákazníků. Všechny testovací scénáře se nestihnou projít kvůli časovým nárokům.

Testování platforem nebo testování na zákaznických projektech provádí analytik / tester. Testuje se dle úplného testovacího scénáře GC nebo GI. Případně vedoucí projektu může rozhodnout o testování vybraných relativně samostatných celků, ale pouze v případě, že má jistotu, že ostatní části aplikace jsou bezproblémové.

5.4.7 Uvolňování verze

Po řádném otestování rozhoduje vedoucí projektu, jestli dojde k uvolnění nové verze. Může dospět i k rozhodnutí o uvolnění pouze pro konkrétního zákazníka nebo pouze pro konkrétní platformu. Rozhodnutí o uvolnění verze produktu musí být zaznamenáno v Zápise z vývojové porady. Dále je provedena záloha a dochází k přípravě distribučního balíčku nové verze. Vedoucí projektu připravuje seznam provedených změn v buildu a informuje emailem zákazníka, že je dostupná nová verze. Paralelně se už pracuje na dalších požadavcích, které budou promítnuty v následujícím balíčku.

5.4.8 Údržba verze

Vedoucí projektu je odpovědný za údržbu dané verze po jejím uvolnění. Údržba spočívá v odstraňování vad zjištěných na základě námětů zákazníků a uživatelů.

Požadavky na opravy či drobné změny probíhají skrze aplikaci Helpdesk, odtud se automaticky promítají do aplikace TFS. Zde se uvažují pouze opravy a změny týkající se nové verze, nikoliv chyby implementace. Vedoucí projektu zodpovídá za ověřování, zda se jedná opravdu o vady.

Vedoucí projektu připravuje seznam úkolů, jež navrhuje k realizaci v následujícím buildu nebo patchi. Patchem se rozumí co nejrychleji vytvořený a distribuovaný build

obsahující pouze opravu závažné vady, která nemohla počkat na pravidelný build, tedy na novou verzi.

5.5 Zavedené procesy

5.5.1 Porady

Jednou měsíčně probíhají porady interních projektů, kde se vedoucí interních projektů zodpovídají TOP managementu.

V kalendáři je na každý týden naplánováno týdenní review, kde vedoucí projektu BI rekapituluje současný stav. Vzniknuvší problém většinou „vybublá“ nečekaně a řeší tak hned v kanceláři, že se členové týmu sejdou. Úkoly jsou zadávány operativně, nečeká se na týdenní review.

Programátoři mají jednou měsíčně programátorskou poradu, která se nese spíše v obecnějším duchu. Probíhá zde vzájemná inspirace a sdílejí si navzájem novinky z programátorského světa.

5.5.2 Monitorování chyb

Chyby jsou hlášeny skrze aplikaci Helpdesk, což je aplikace, kam zákazníci a uživatelé zadávají nové náměty. Někdy se stává, že nahlásí vadu, ale jedná se o požadavek pro další realizaci. Konzultant rozhoduje, zda se jedná o chybu vývoje (závažné chyby, které se musí řešit ihned a vydává se nový patch), nebo o chybu v realizaci (chybně naimplementováno).

5.5.3 Vydávání verzí

Jedná se o dlouhodobý interní projekt systému Business Intelligence, který je rozdělený do jednotlivých iterací, kde výsledkem každé iterace je příslušný build (verze). Jedna iterace trvá průměrně dva měsíce, což znamená, že každé dva měsíce je zpřístupněna pro zákazníky nová verze.

5.6 Používané nástroje

- **Helpdesk** – aplikace založená na bázi Microsoft CRM, do které externí zákazníci mohou zapisovat vypozorované chyby.
- **Team Foundation Server (TFS)** - nástroj, který umí řídit kód. V případě, že programátor provede nějakou změnu v kódu, tak ji musí přidělit k příslušnému

úkolu. Pak má možnost sledovat, jakou změnu udělal a vidí úkoly, které jsou již hotové a ostatní, na které se musí vrhnout.

- **Visual Studio** – programátoři v něm píší kód, nástroj je integrován do prostředí TFS.
- **Word** – používají pro zadání velké funkcionality, při menším rozsahu psaní zadání přeskakují.
- **Sharepoint –** je nástroj pro týmovou spolupráci. Jsou zde sdílena veškerá zadání. Ve chvíli, kdy je zadání schválené, tak se přenáší do vývojového nástroje TFS.
- **Generátor nápovědy** – speciální aplikace, ve které analytik / tester vytváří nápovědu.
- **Power BI** – vlastní datový sklad, který slouží pro přehled kolik úkolů (funkcionalit) zbývá k dořešení. Programátoři a analytici / testeři přehledně vidí, kolik úkolů mají za splněných a kolik jim zbývá dořešit. Vedoucím pracovníkům slouží aplikace pro jejich přehled, jak si jejich tým vede.
- **Outlook** – používají na klasickou poštu, ale komunikace může probíhat i v rámci chatu v programu TFS.

6 Případová studie

Případová studie se bude zabývat používáním agilních metodik a jejich nástrojů po celém světě. Ke zpracování nějakých závěrů nám poslouží data z dotazníkového šetření, které provádí americká společnost CollabNet VersionOne. Jsou to právě oni, kteří přišli s touto myšlenkou na trh a již od roku 2006 provádí každoroční šetření globálního rázu s názvem „State of Agile Report“, který se zaměřuje na otázky spojené s agilním vývojem softwaru. Jedná se o nejrozsáhlejší a nejdéle běžící dotazník s agilní tématikou na světě. Odpovědi na tyto dotazníky zpracovává a analyzuje nezávislá společnost Analysis.Net Research.

Naposledy dostupná data jsou z minulého roku, kdy vyšel „12th Annual State of Agile Report“ za rok 2017. V době psaní této práce bohužel ještě nebyly k dostání výsledky z dotazníkového šetření za rok 2018. Výstupy z těchto dotazníků jsou vydávány zhruba v polovině následujícího roku od skončení dotazování.

Nutno podotknout, že odpovědi mohou být poněkud zkreslené, jelikož jsou sbírány především od lidí, kteří již nějaké zkušenosti s agilními metodikami nasbíraly (Tománek, 2015).

V této kapitole si klademe za cíl vysledovat aktuální trendy v agilním prostředí a zaznamenat, jak se za poslední roky mění přístup agilně vyvíjejících firem. Výstupy budou sloužit jako nápomocný podklad při budoucím zavádění agilní metodiky ve společnosti GIST, s.r.o., kterou jsme v minulé kapitole podrobili analýze.

Cíl této studie bude dosažen pomocí srovnání jednotlivých dotazníků, které si drží obdobnou strukturu dotazování každý rok. Porovnány budou pouze vybrané otázky a odpovědi na ně budou zpracovány do časových řad. Z hodnot zanesených do grafů se pokusíme určit, zda časové řady vystihují nějaký trend. K analýze budou podrobeny následující body:

- Počet respondentů
- Zkušenosti podniku s agilními přístupy
- Doba používání agilního přístupu

- Agilně řízené projekty
- Důvody pro zavedení agilních metodik
- Výhody po zavedení agilních metodik
- Používané agilní metodiky
- Používané agilní techniky
- Nástroje pro řízení agilních metodik

Obr. 10: Ukázka z prvního reportu 2006

Zdroj: Výstřižek z (VersionOne, 2007)

Obr. 11: Ukázka z dvanáctého reportu 2007

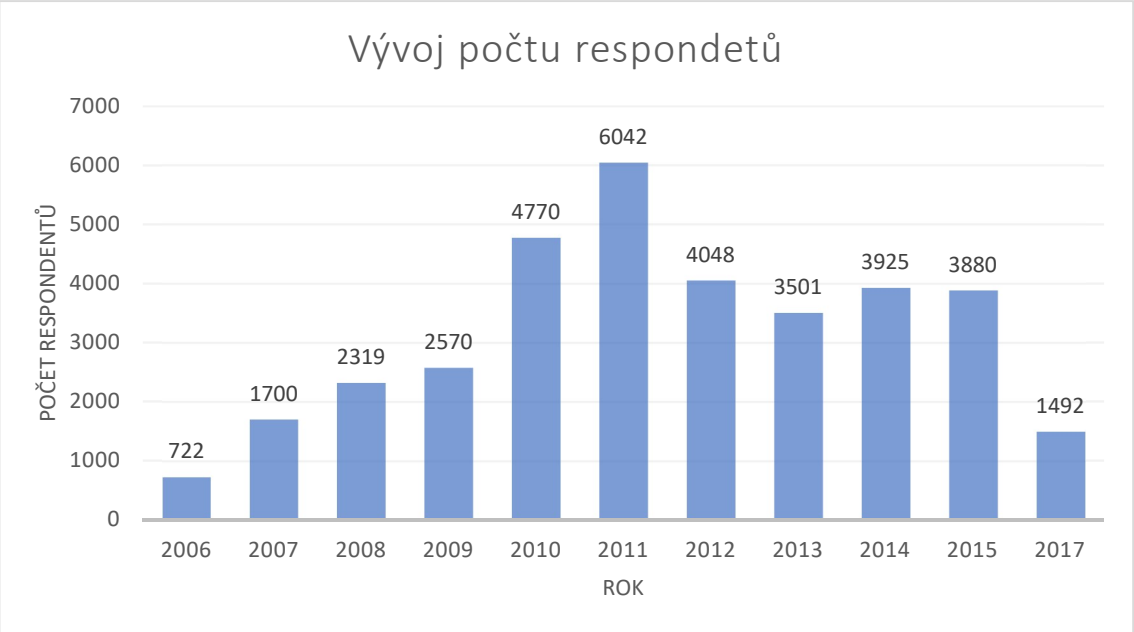
Zdroj: Výstřižek z (VersionOne, 2018)

Dva výše uvedené obrázky nám demonstrují ukázku výsledků z prvního dotazníku (Obr. 11) a ukázku z aktuálně posledního dostupného dotazníku (Obr. 12). Již od pohledu můžeme vidět, že zpracování posledního ročníku zaznamenalo jistý progres od prvního ročníku alespoň po vizuální stránce.

6.1 Počet respondentů

Graf 1 znázorňuje vývoj počtu respondentů, kteří se zúčastnili dotazníkových šetření od společnosti CollabNet VersionOne. Počty respondentů jsou zkoumány od prvního ročníku z roku 2006 až po poslední dostupný z roku 2017. Ti bystřejší si jistě všimli, že je vynechán rok 2016, je to z důvodu toho, že v reportu tato informace zkrátka chybí. Společnosti CollabNet VersionOne byl odeslán požadavek na tuto informaci, avšak bez odezvy, bylo by asi překvapením, kdyby odpověděli.

Pohled na graf 1 může v člověku vyvolávat pocit, že hledí na pyramidu. Špičku té pyramidy tvoří právě rok 2011, kdy se dotazníku podrobil rekordní počet – 6042 respondentů. Následující roky se nesly ve standardních počtech, avšak rok 2017 zaznamenal hluboký pokles. Velkým otazníkem tedy je, zda se počet dotazujících za rok 2018 vzpružil na hodnotu kolem tří až čtyř tisíc respondentů.

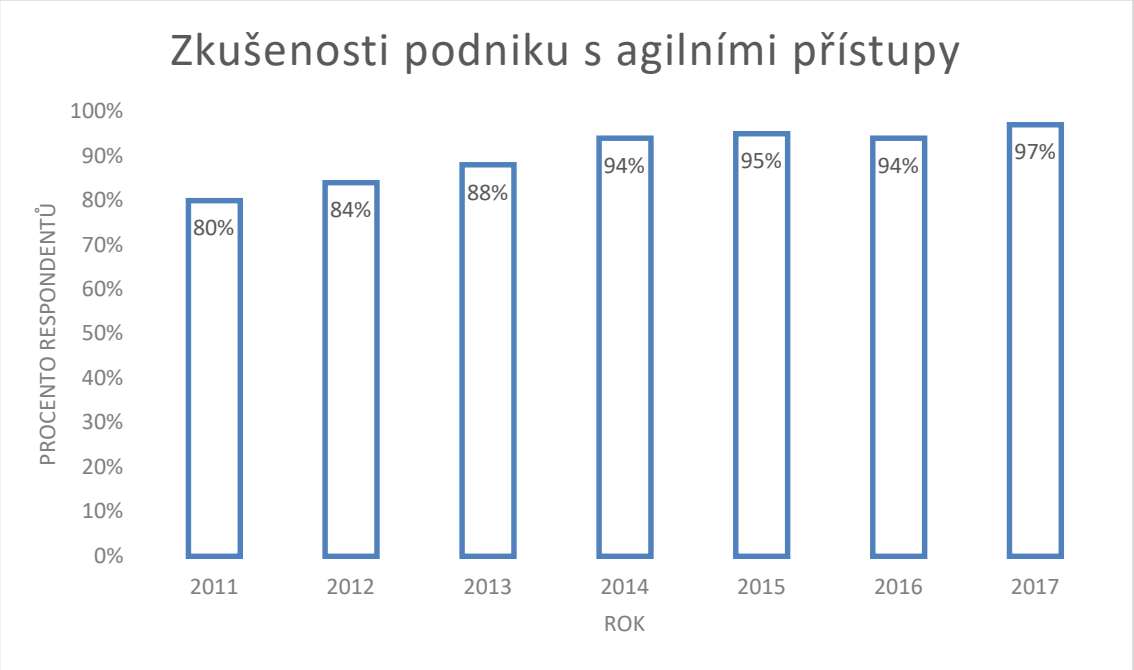


Graf 1: Vývoj počtu respondentů

Zdroj: Vlastní vypracování z dat (VersionOne, 2007–2018)

6.2 Zkušenosti podniku s agilními přístupy

Graf 2 popisuje, jak vysoké procento respondentů odpovědělo, že ve firmě, kde pracují, používají agilní přístup. Můžeme vidět, že v posledních čtyřech letech je agilní přístup již zaběhlý téměř v každé firmě, procento nad 95 % můžeme očekávat i za rok 2018. Ve firmách je tedy „agile“ populární, avšak nelze opomenout, co bylo zmíněno v úvodu této kapitoly, a to, že výsledky mohou být lehce zkreslené.

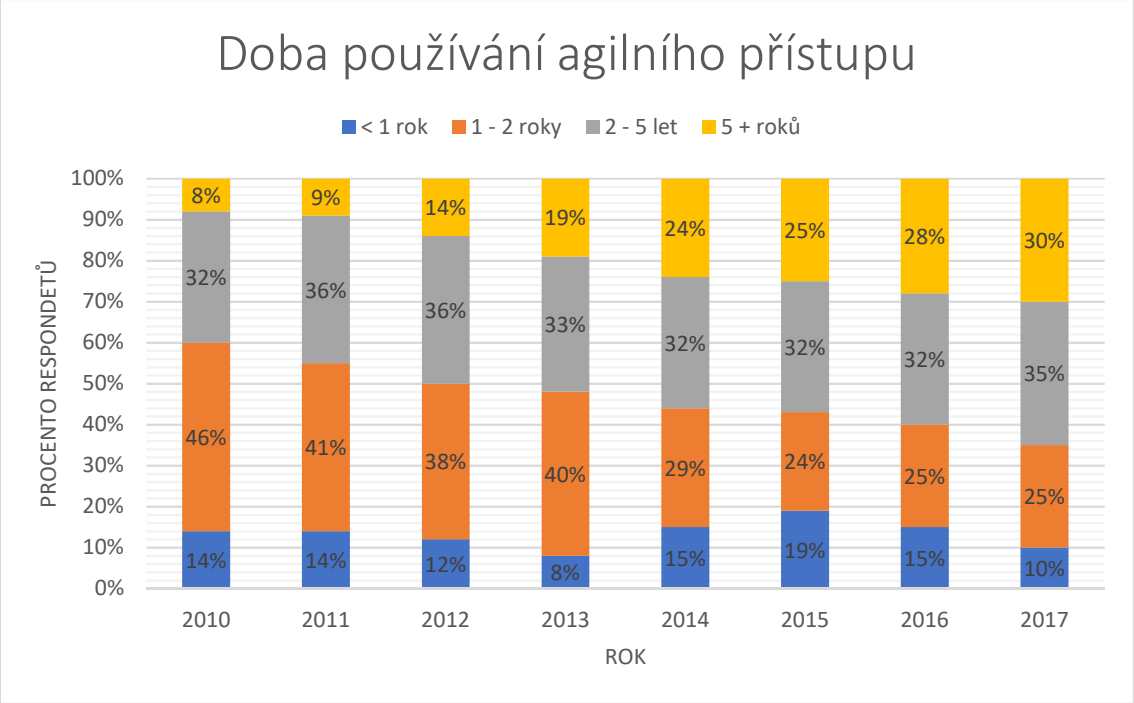


Graf 2: Zkušenosti podniku s agilními přístupy

Zdroj: Vlastní vypracování z dat (VersionOne 2012-2018)

6.3 Doba používání agilního přístupu

Dalším bodem v pořadí je průzkum, jak dlouhou dobu firmy praktikují agilní přístup. Relativní četnosti doby používání agilních metodik jsou uvedeny v grafu 3. Tuto otázku společnost CollabNet VersionOne zařadila do dotazníku v roce 2010, od kterého naše srovnání začíná. Odpovědi rozčleňují organizace do čtyř kategorií dle jejich délky praxe s „agile“.



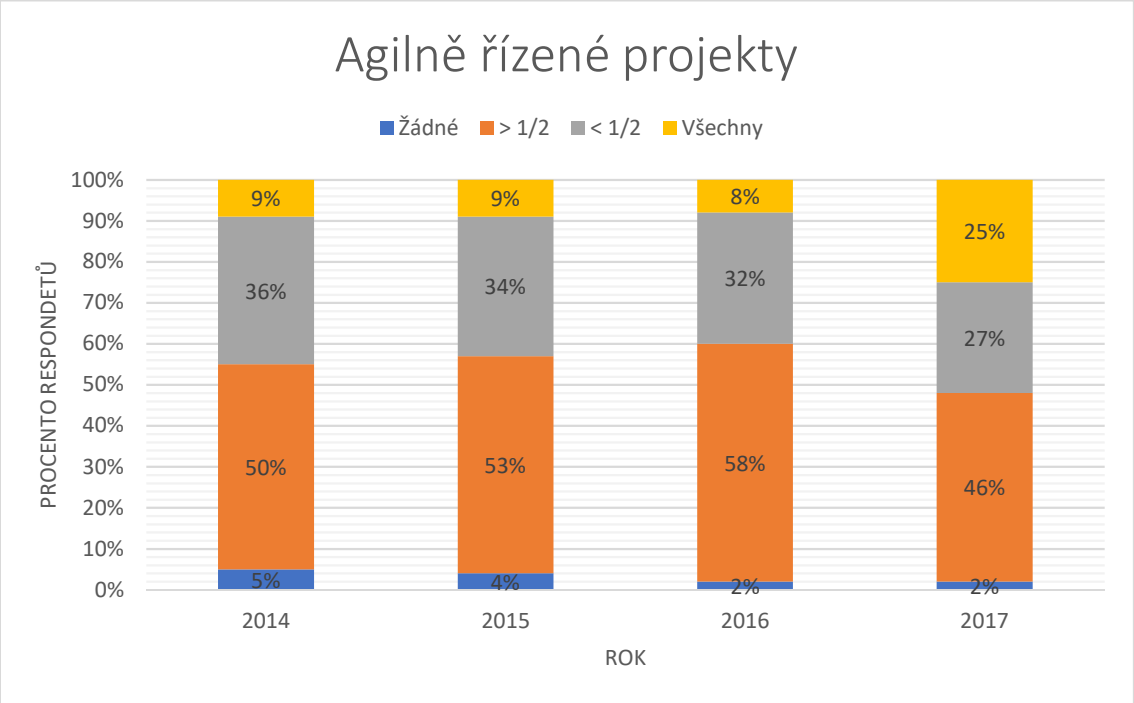
Graf 3: Doba používání agilního přístupu

Zdroj: Vlastní vypracování z dat (VersionOne 2011-2018)

Zavádění agilních metodik a jejich oblíbenost potvrzuje fakt, že si je firmy čím dál více osvojují a řídí se podle nich. Narůstající trend je reflektován v grafu výše, kdy v roce 2010 existovalo 60 % firem, které nepoužívaly agilní metodiky déle než dva roky, zatímco v posledních letech se karta obrátila a od roku 2016 je používá přes 60 % déle než dva roky, což má za následek, že kategorie 1–2 roky se postupně snižuje. Rovněž nelze přehlédnout nárůst v roce 2015 u firem, které měly čerstvé zkušenosti (méně než jeden rok) s agilním světem, a to jen dokazuje neupadající zájem o agilní metodiky.

6.4 Agilně řízené projekty

Následující graf 4 zobrazuje relativní četnosti projektů, které jsou řízeny ve firmě pomocí agilních metodik. Pro porovnání byly vybrány pouze poslední čtyři dostupné roky, a to z důvodu, že od roku 2014 byla lehce upravená struktura dotazníku a nešly by tak pochopitelně porovnávat stejné proměnné. Respondenti si mohli vybírat mezi žádným agilně řízeným projektem, méně než polovinou řízených projektů, více než polovinou agilně řízených projektů, nebo všemi agilně řízenými projekty.



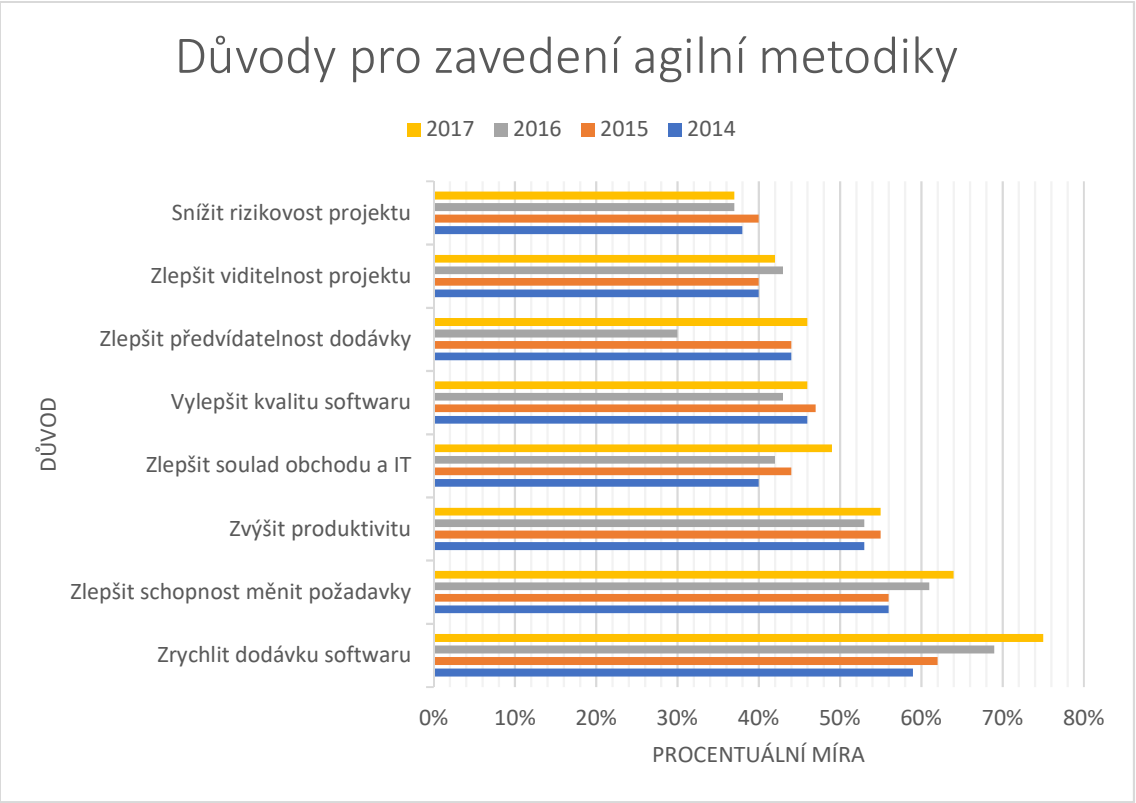
Graf 4: Agilně řízené projekty

Zdroj: Vlastní vypracování z dat (VersionOne 2015-2018)

Za zmínku rozhodně stojí vyhodnocení ročníku 2017, kde můžeme pozorovat meziroční nárůst u všech agilně řízených projektech o 17 %. Také poprvé v roce 2017 vidíme, že miska vah, kdy ve firmě jsou z více než poloviny řízeny projekty agilně, se přehoupla přes 50 %. V minulých letech byl trend opačný.

6.5 Důvody pro zavedení agilní metodiky

Další část v dotazníku patří hlavním důvodům, proč firmy přecházejí na agilní vývoj. Jejich očekávání jsou zanesena v grafu 5. Sledovány byly poslední čtyři roky. V tomto případě celkové procento v jednom roce nabývá vyšší hodnoty než 100 %, jelikož respondentům bylo umožněno vybírat více možností (důvodů).



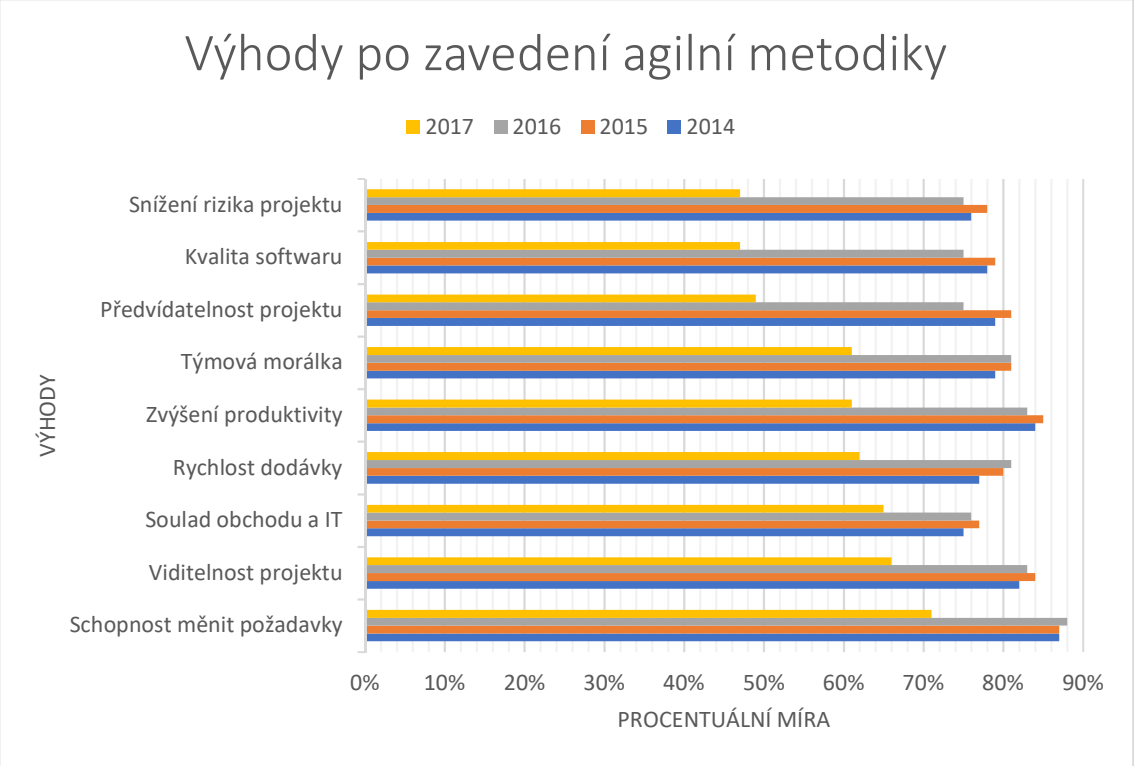
Graf 5: Důvody pro zavedení agilní metodiky

Zdroj: Vlastní vypracování z dat (VersionOne 2015-2018)

Všechny čtyři zkoumané roky potvrzují, že se hlavní důvody pro zavedení agilního vývoje nemění. Mezi tři nejuváděnější důvody patří snaha o zrychlení dodávky softwaru, zlepšení schopnosti měnit požadavky a zvýšení produktivity. Největší meziroční nárůst byl zaznamenán u bodu zlepšení předvídatelnosti dodávky, v roce 2017 tento důvod uvedlo o 16 % procent více než v roce 2016.

6.6 Výhody po zavedení agilní metodiky

S důvody pro zavedení agilních metodik z předchozí strany je zajímavostí, porovnat je s reálnými výhodami, které po zavedení agilního přístupu podnikům přinesly. Podobně jako u důvodů byly vybrány pro porovnání poslední čtyři dostupné reporty. Respondenti opět mohli zaškrtávat více možností, v tomto případě výhod. Srovnání výhod napříč roky 2014 až 2017 vyobrazuje graf 6.



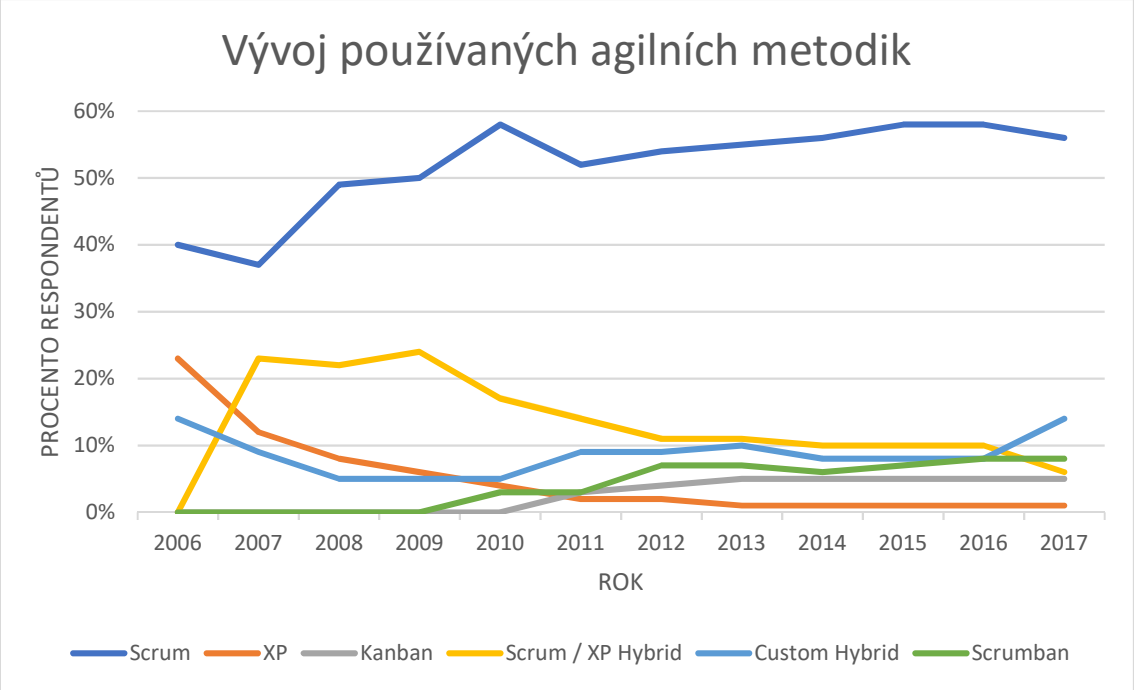
Graf 6: Výhody po zavedení agilní metodiky

Zdroj: Vlastní vypracování z dat (VersionOne 2015-2018)

Rok 2017 poprvé přinesl změnu na stupních vítězů za nejuváděnější výhody po zavedení agilní metodiky. V předešlých letech respondenti nejvíce vnímali jako výhody schopnost měnit požadavky, zvýšení produktivity a viditelnost projektu. Právě zvýšení produktivity vystřídal benefit souladu obchodu a IT. Dále si nelze nevšimnout všech žlutých pruhů, reprezentujících rok 2017, které dosáhly podstatně nižších hodnot než v předešlých letech. Proč tomu tak doopravdy je, bohužel nevíme a bude tak zajímavostí vyhlížet výsledky za rok 2018, zda se nižší hodnoty budou opakovat.

6.7 Používané agilní metodiky

Agilních metodik existuje v různých modifikacích v dnešní době opravdu nemalé množství. Pro lepší přehlednost grafu byly vybrány pouze ty nejpoužívanější agilní metodiky. Jedná se o metodiky Scrum, Extrémní programování (XP), Scrum / XP Hybrid, Kanban, Custom Hybrid a Scrumban. Graf 7 promítá vývoj používání těchto metodik ze všech aktuálně dostupných reportů, tj. od roku 2006 do roku 2017.



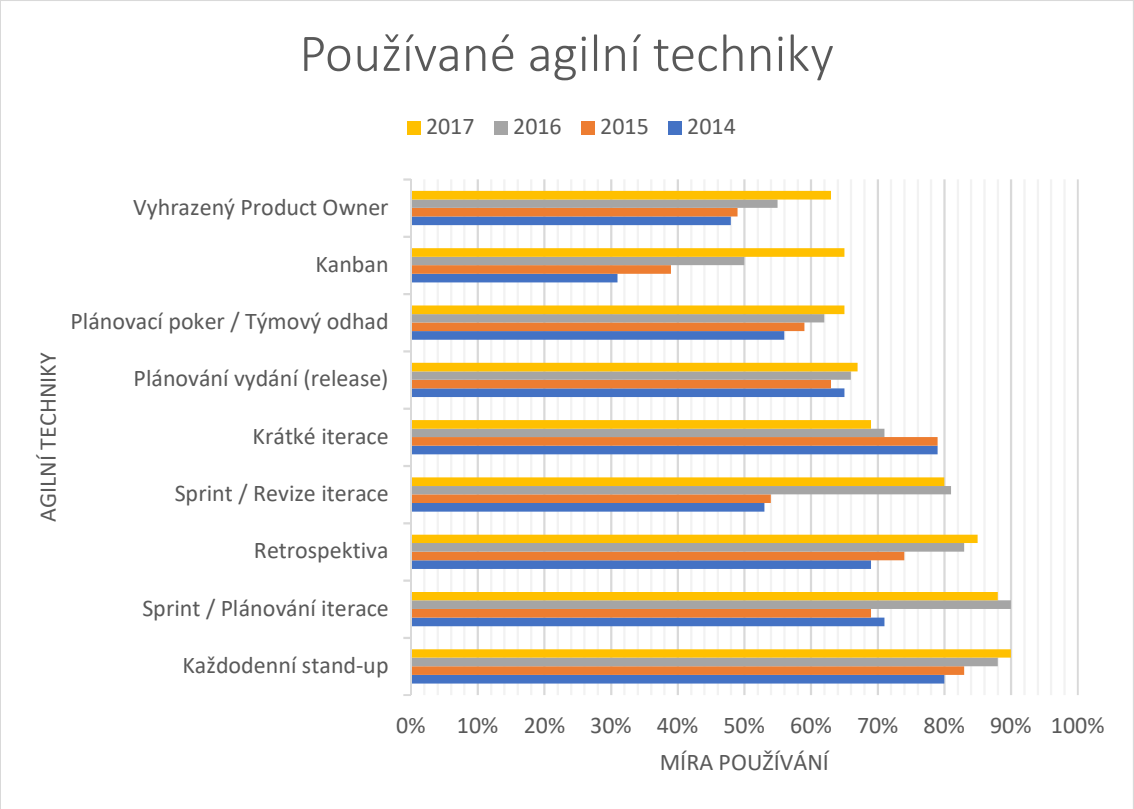
Graf 7: Vývoj používaných agilních metodik

Zdroj: Vlastní vypracování z dat (VersionOne 2007-2018)

Hned na první pohled je zcela zřejmá dominance agilní metodiky Scrum, která válcuje ostatní metodiky na plné čáře. Největší rozkvět tato metodika zaznamenala v roce 2010 a svou popularitu si drží nadále. V roce 2017 uvedlo 56 % respondentů, že řídí projekty dle metodiky Scrum. Pokud bychom k samotnému Scrumu přidali další dvě jeho variace Scrum / XP Hybrid (Scrum kombinovaný s Extrémním programováním) a Scrumban (Scrum kombinovaný s Kanbanem), tak od roku 2008 se pohybujeme kolem 70% hranice využívání metodik založených na metodice Scrum. Nejvýraznější vývojový pokles zaznamenala metodika XP, která ještě v roce měla 23 % příznivců, kdežto v roce 2017 podle XP postupuje pouze 1 %.

6.8 Používané agilní techniky

Graf 8 představuje výběr aktuálně nejpoužívanějších agilních technik a mapuje jejich vývoj co do míry používání ve firmách. Sledovány byly čtyři poslední dostupné ročníky, tj. 2014 až 2017. Respondentům byl pochopitelně umožněn výběr více agilních technik.



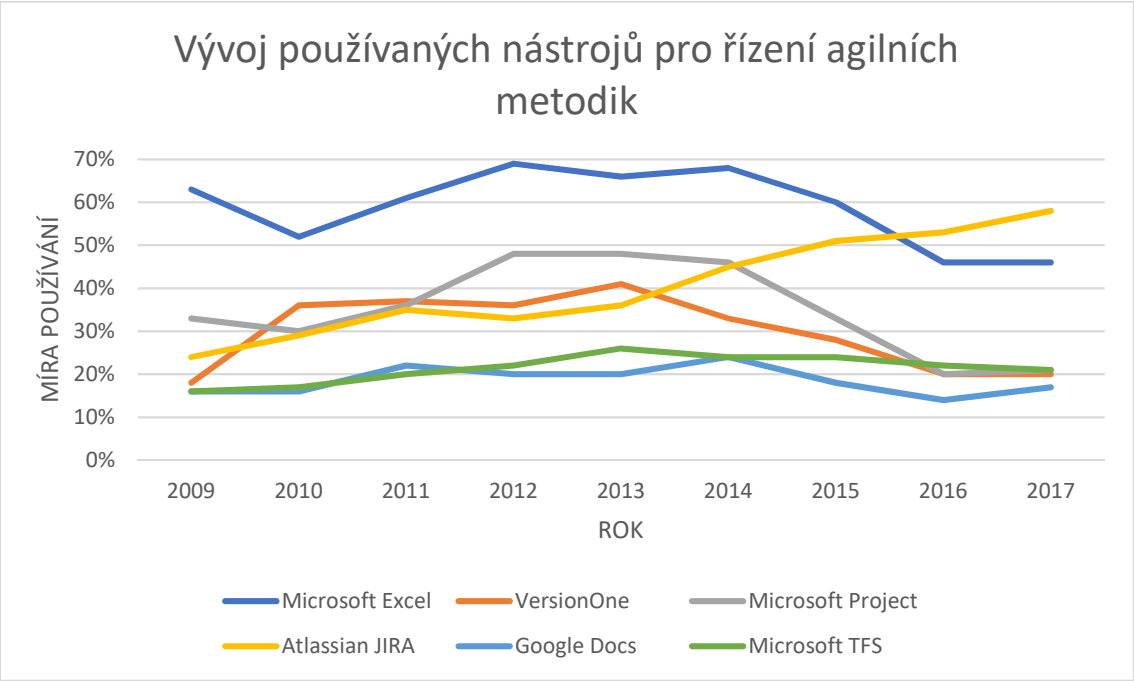
Graf 8: Používané agilní techniky

Zdroj: Vlastní vypracování z dat (VersionOne 2015-2018)

Čím dál používanější technikou se stává Kanban, který známe jako agilní metodiku. V tomto případě se jedná o techniku, která spočívá ve vizualizaci a omezení práce v podobě barevných lístečků. V roce 2014 uvedlo, že používá Kanban techniku 31 % respondentů, o tři roky později to je již 65 %. Největší meziroční nárůst jsme zaznamenali u techniky revize iterace, která mezi roky 2015 a 2016 vzrostla o 24 %. Mezi nejpoužívanější agilní techniky za poslední dva zkoumané roky patří každodenní stand-up, dále plánování iterace a do třetice retrospektiva, bez kterých by se neobešlo téměř 90 % agilně vyvíjejících týmů.

6.9 Nástroje pro řízení agilních metodik

Závěrečná část dotazníku se respondentů dotazuje, jaké nástroje používají ve firmě pro správu agilních metodik. Přehledný vývoj užívaných nástrojů je znázorněn v grafu 9, do kterého byla vybrána data z reportů od roku 2009 do roku 2017.



Graf 9: Vývoj používaných nástrojů pro řízení agilních metodik

Zdroj: Vlastní vypracování z dat (VersionOne 2010-2018)

Od samotného počátku dotazování byl Microsoft Excel uváděn jako nejpoužívanější nástroj, než ho v roce 2016 vystřídal na vedoucí pozici nástroj JIRA od společnosti Atlassian a vypadá to, že nastolený trend bude pokračovat i v nadcházejících letech. Za rok 2017 zaškrtlo JIRA 58 % respondentů, přitom v roce 2009 ho používalo pouze 24 %. Stagnující Excel spadl na 46 %. Podívejme se na rok 2013, kdy nástroje VersionOne (41 %) a Atlassian JIRA (36 %) byly konkurenceschopné. V roce 2014 nastal zlomový okamžik a JIRA meziročně stoupla o 9 %, za to VersionOne klesl meziročně o 13 % a od té doby pozorujeme, že Atlassian postupně přebíral zákazníky od VersionOne.

Všimněme si nástroje Microsoft TFS, který nám je známý ze společnosti GIST, s.r.o., která ho pro své řízení projektů používá. Ačkoliv jasně vzestupující trend zaznamenal nástroj JIRA, tak TFS od společnosti Microsoft je dobrou volbou, což prokazuje jeho používání po celém světě.

7 Výběr vhodné agilní metodiky

Následující kapitola se zaměří na výběr vhodné agilní metodiky vývoje softwaru, která bude dále doporučena k implementaci do podniku GIST, s.r.o. Ve výběru vhodné agilní metodiky nám pomůže software pro podporu rozhodování Expert Choice 2000, který je volně dostupný pro naše studenty Fakulty informatiky a managementu.

Nejprve dojde k výběru vhodných kritérií, které následně pracovníci GIST, s.r.o. v rámci dotazníkového šetření mezi sebou porovnají a stanoví jim jejich vnímanou důležitost. Z výsledků dotazníků budou pomocí Saatyho matice stanoveny váhy jednotlivých kritérií. Právě Thomas Saaty byl spoluzakladatelem společnosti Expert Choice, která aplikuje jím vytvořený proces AHP (The Analytic Hierarchy Process). Následovně dojde ke zvolení tří agilních metodik, které budou hodnoceny v závislosti na vybraných kritériích. Expert Choice 2000 na konci procesu seřadí jednotlivé metodiky od nejlepší po nejhorší (Fiala, Jablonský, & Maňas, 1994; Fotr & Švecová, 2016).

Všechny agilní metodiky vyznávají podobné hodnoty a principy, které stanovuje Manifest agilního vývoje softwaru, i přesto ale nejsou totožné a v konkrétních situacích se liší.

7.1 Kritéria výběru

Zvolená kritéria byla pečlivě vybrána a diskutována s produktovým ředitelem společnosti GIST, s.r.o. Hledali jsme takové body, které se úzce dotýkají a ovlivňují vývoj aplikace. Vybrali jsme následujících sedm kritérií, na jejichž základě bude vybrána nejvhodnější agilní metodika vývoje softwaru:

- **Míra složitosti metodiky** – čím složitější metodika je, tím více času trvá adaptace na ni.
- **Míra dokumentace** – přiměřená míra dokumentace, vymanit se psaní zbytečné a nepotřebné dokumentace.
- **Míra chybovosti** – nadbytečná chybovost je neefektivní a zabírá tak volnou kapacitu.

- **Specifikace požadavků zákazníka** – porozumění zákazníkových požadavků pro jeho spokojenost. Chybně pochopené požadavky jsou příčinou zbytečně vykonané práce.
- **Rychlost dodávky** – vydání buildu v domluveném termínu.
- **Definování zodpovědnosti** – přidělení zodpovědností jedincům, či celému týmu.
- **Komunikace napříč týmem** – vzájemná interakce zvyšuje efektivitu celého týmu a problémy jsou řešeny hned.

7.2 Stanovení vah jednotlivých kritérií

Pro stanovení vah určených kritérií bylo provedeno dotazníkové šetření, které proběhlo uvnitř podniku. Před samotným šetřením byla za spolupráce produktového ředitele provedena pilotní studie, která ověřila proveditelnost dotazníku.

Po skončení pilotní studie byly dotazníky distribuovány produktovým ředitelem mezi jeho spolupracovníky. Dotazník byl zpracován elektronicky a následně vytištěn do papírové podoby. V rámci celého průzkumu byla zachována anonymita respondenta. Struktura a obsah dotazníku je k nahlédnutí v příloze A. Celkově jsme se dočkali návratnosti dotazníků od pěti respondentů.

Jako metoda pro vícekriteriální rozhodování byla vybrána Saatyhoda metoda, která srovnává páry kritérií mezi sebou. Respondenti tedy byli vyzváni, aby učinili tento krok a ohodnotili vždy dvě kritéria mezi sebou. Vzhledem k tomu, že bylo získáno celkově pět hodnocení, tak prvně proběhlo stanovení vah kritérií pomocí Saatyho matice pro všech pět dotazníků zvlášť a následně byl z odlišných vah stanoven aritmetický průměr pro každé kritérium. Výsledné stanovené váhy jsou zaneseny v tabulce 1. Analýza jednotlivých dotazníků v podobě Saatyho matic je dostupná v příloze B.

Kritérium	Váhy
Míra složitosti metodiky	**0,031**
Míra dokumentace	**0,054**
Míra chybovosti	**0,227**
Specifikace požadavků zákazníka	**0,315**
Rychlost dodávky	**0,179**
Definování zodpovědnosti	**0,068**
Komunikace napříč týmem	**0,126**
∑	**1,000**

Tabulka 1: Stanovené váhy kritérií

Zdroj: vlastní zpracování

Za nejdůležitější kritérium respondenti považují specifikaci požadavků zákazníka. Zvolili jej proto, že správné porozumění zákazníkovi vnímají jako nejdůležitější faktor, s chybnou specifikací požadavků vyjde práce celého týmu vniveč.

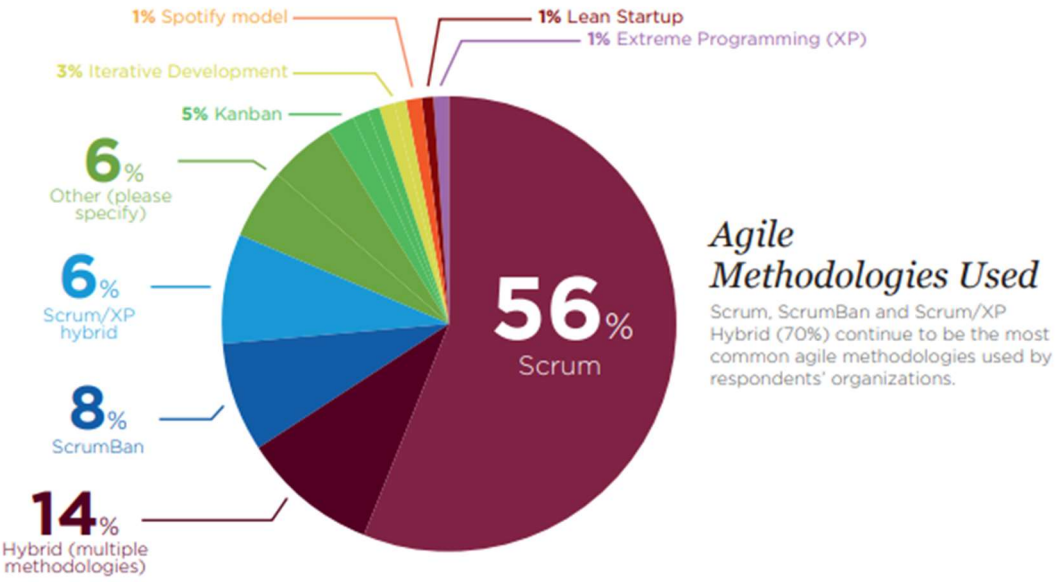
Jako druhé nejdůležitější kritérium skončila míra chybovosti. Podobně jako u chybné specifikace, tak u vysoké chybovosti dochází k neefektivitě vykonané práce. Snaha o eliminaci chybovosti v rámci celého týmu je promítnuta ve vnímané důležitosti tohoto kritéria.

Bronzovou medaili, co do seřazení kritérií dle důležitosti, obdržela rychlost dodávky. Pro společnost je žádoucí vydat novou verzi systému bez většího časového prodlení.

Komunikace napříč týmem skončila na čtvrtém místě a předhonila kritéria definování zodpovědnosti, míru dokumentace a míru složitosti metodiky. Že je komunikace brána za důležitou je dobře, protože bez ní to v agilním světě nejde. Naopak za nejméně důležité kritérium spolupracovníci uvádějí míru složitosti metodiky.

7.3 Zvažované metodiky

Důležitá kritéria pro výběr vhodné metodiky máme již zvolena a nyní zbývá vybrat vhodné adepty z řad agilních metodik, které budou přisuzovány jednotlivým kritériím dle vhodnosti dané metodiky. Pro takový výběr jsme využili předešlé kapitoly, která se věnovala trendu agilních metodik po celém světě. Do výběru jsme tedy zařadili metodiky: Scrum, Kanban a Extrémní programování (XP), právě tyto 3 zmíněné metodiky byly uvedeny jako nejpoužívanější za rok 2017 (viz obr. 13), vyjímaje hybridních metodik.



Obr. 12: Nejpoužívanější agilní metodiky v roce 2017

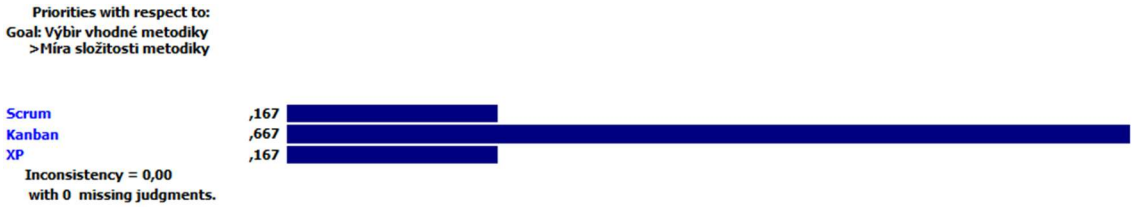
Zdroj: Výstřižek z (VersionOne, 2018)

7.4 Ohodnocení metodik podle kritérií

Pro zjednodušení a pro lepší znázornění nám poslouží program Expert Choice 2000, ve kterém jsme již stanovili váhy jednotlivých kritérií a nyní dojde k ohodnocení metodik na základě vybraných kritérií. Jak bylo uvedeno v předešlé podkapitole, tak na výběr k hodnocení byly vybrány tři agilní metodiky: Scrum, Kanban a Extrémní programování (XP). Ke správnému ohodnocení poslouží kapitola 4, která byla věnována agilním metodikám a popisuje právě tři zmíněné metodiky.

7.4.1 Míra složitosti metodiky

Ačkoliv smyslem agilních metodik je právě jednoduchost, tak nám se jako nejjednodušší jevila metodika Kanban, která se mimo jiné řadí mezi štíhlou výrobu (Lean Development). Metodika Kanban ve své podstatě nic nenařizuje ani nezakazuje, pouze musí dodržet tři principy: vizualizovat práci, minimalizovat čas průchodu a omezit rozpracovanou práci. Metodiky Scrum a XP vnímáme podobně složité, jelikož už musí dodržovat určité postupy, nicméně za složité se rozhodně považovat nedají.



Obr. 13: Ohodnocení kritéria Míra složitosti metodiky

Zdroj: vlastní zpracování (výstřižek z Expert Choice 2000)

7.4.2 Míra dokumentace

I když jedna základní hodnota Agilního manifestu praví, že preferuje fungující software před vyčerpávající dokumentací, tak to neznamená, že lze dokumentaci zcela vypustit a nevěnovat ji žádnou pozornost. Míra dokumentace se odvíjí od nároků zadavatele, ten, pokud si umane, že chce vytvořit dokumentaci na určité části systému, tak mu ji musíme dodat. Z tohoto pohledu vyšla jako nejvíce vyhovující metoda Extrémního programování, jelikož jednou ze dvanácti praktik jsou standardy pro psaní zdrojového textu, které se musí vývojáři držet. Zdrojový kód totiž v metodice XP představuje část dokumentace. Díky stanovenému řádu pro psaní kódu pak není problém vyznat se v jednotlivých řádcích, i když je psal kolega.

Obr. 14: Ohodnocení kritéria Míra dokumentace

Zdroj: vlastní zpracování (výstřižek z Expert Choice 2000)

7.4.3 Míra chybovosti

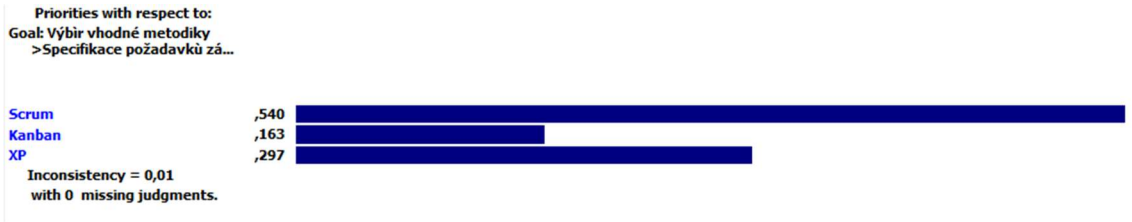
Chybovost je ožehavé téma a v první řadě záleží na lidech, které máte v týmu. Jsou faktory, které mohou chybovost ovlivnit, ať už je to motivace zaměstnance nebo jeho schopnosti. My shledali výhodu opět v metodice Extrémního programování, která se řídí praktikou čtyřiceti hodinového týdne, která značnou částí přispívá ke snížení chybovosti. Pracovníci tak nedělají chyby, které by pramenily z jejich únavy, ledaže by si v předchozím dni „vyhodili z kopýtka“.

Obr. 15: Ohodnocení kritéria Míra chybovosti

Zdroj: vlastní zpracování (výstřižek z Expert Choice 2000)

7.4.4 Specifikace požadavků zákazníka

U tradičního vývoje docházelo běžně k tomu, že požadavky vzešlé od zákazníka nebyly dobře definovány, nebo došlo k nepochopení ze strany výrobce aplikace. Vinu za to nesla především nedostatečná komunikace mezi oběma stranami. I když všechny vybrané agilní metodiky nabádají pro aktivní komunikaci mezi členy týmu a jejich zákazníkem, my jsme přesto zvolili jako nejcennější metodiku Scrum. Učinili jsme tak z důvodu, že Scrum definuje důležitou roli Product Ownera, který je součástí týmu a zodpovídá za jednotlivé požadavky na systém. V metodice XP je rovněž zákazník přidělen na pracoviště k vývojářům, ale nemá přesně stanovené úkoly, je pouze k dispozici programátorům.

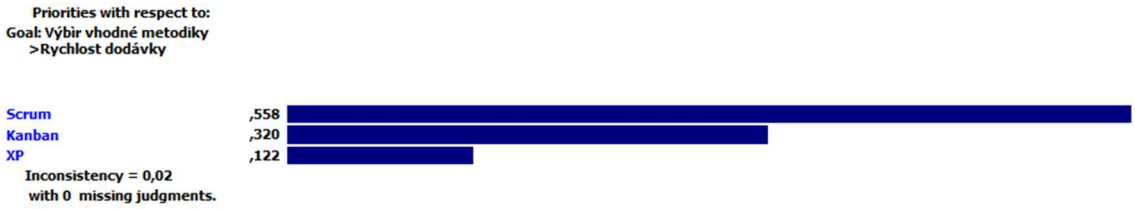


Obr. 16: Ohodnocení kritéria Specifikace požadavků zákazníka

Zdroj: vlastní zpracování (výstřižek z Expert Choice 2000)

7.4.5 Rychlost dodávky

U rychlosti dodávky nám šlo především o to, aby nedošlo k opoždění vydání nové verze, proto se jeví jako nejvhodnější metodika Scrum, která vývoj rozděluje na více iterací (v případě Scrumu sprintů), kdy na konci každého sprintu je fungující software, čímž zamezíme opoždění dodání. Hned v závěsu se umístila metodika Kanban, u které platí princip minimalizování času průchodu, takže by mělo dojít k dodávce co nejrychleji.



Obr. 17: Ohodnocení kritéria Rychlost dodávky

Zdroj: vlastní zpracování (výstřižek z Expert Choice 2000)

7.4.6 Definování zodpovědnosti

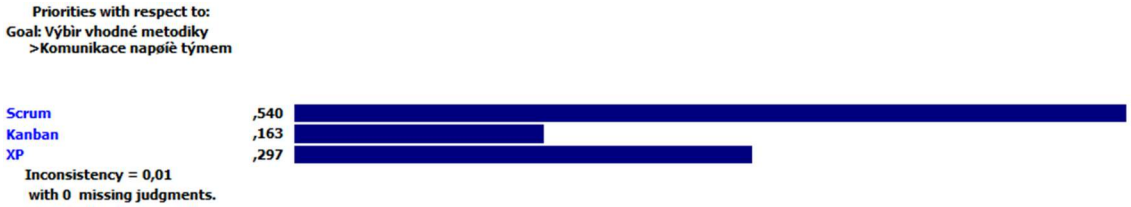
U tohoto kritéria je obtížné vybírat vhodnou metodiku, poněvadž u všech tří metodik je sdílená zodpovědnost v rámci celého týmu. Když má někdo problém, tak ho nemá jednotlivec, ale mají ho všichni, v tom tkví týmová hra, o které agilní metodika bezesporu je. Přeci jen jsme vyhodnotili jako více vyhovující metodiku Scrum, protože role Product Ownera zodpovídá za všechny požadavky na systém (Product Backlog).

Obr. 18: Ohodnocení kritéria Definování zodpovědnosti

Zdroj: vlastní zpracování (výstřižek z Expert Choice 2000)

7.4.7 Komunikace napříč týmem

Komunikace je jedním ze základních pilířů všech agilních metodik. Metodiky jsme hodnotili s ohledem na to, jakou přidanou hodnotu v podobě komunikace nabízí. Nejlépe byla ohodnocena metodika Scrum, která v celém procesu zahrnuje spoustu setkání, plánováním počínaje a retrospektivou konče. Největší přínos však mají denní schůzky, na kterých si členové vyměňují informace. XP metodika sice nestanovuje žádné schůzky, ale využívá praktiky párového programování, při kterém jsou programátoři v neustálém komunikačním kontaktu.

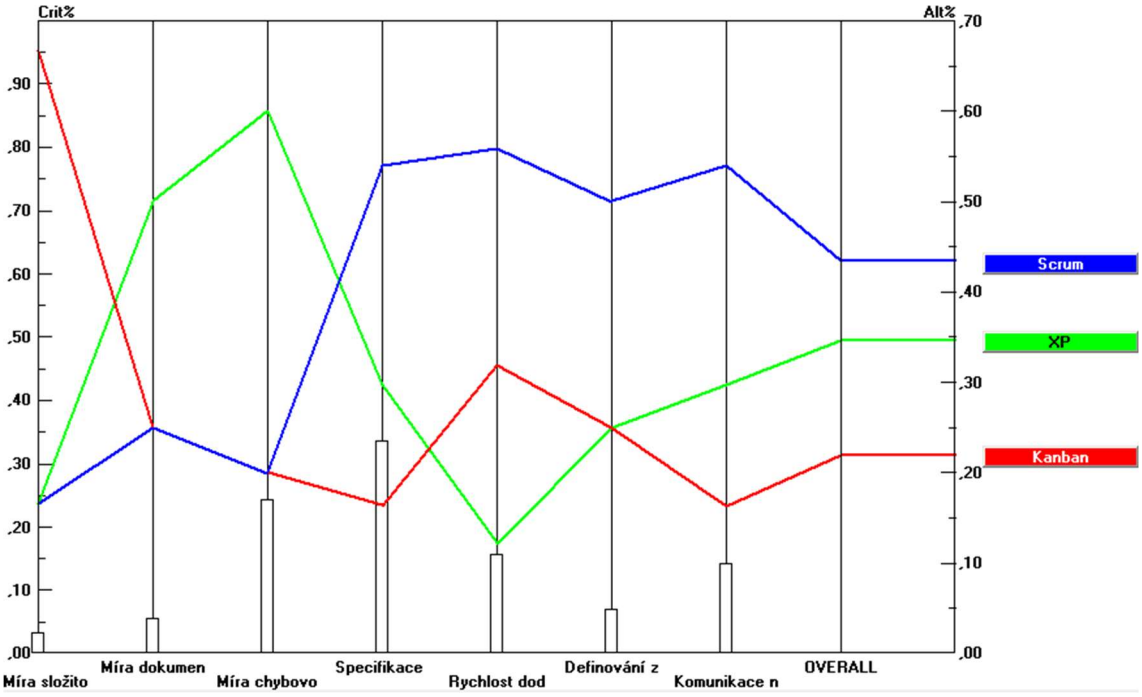


Obr. 19: Ohodnocení kritéria Komunikace napříč týmem

Zdroj: vlastní zpracování (výstřižek z Expert Choice 2000)

7.5 Rozhodnutí o výběru agilní metodiky

Na základě vícekriteriální analýzy a pomocí manažerského nástroje pro podporu rozhodování Expert Choice 2000 byl sestaven výsledný graf, který vybírá co nejvhodnější agilní metodiku pro implementaci do společnosti GIST, s.r.o. Výsledek tohoto výběru si lze prohlédnout v grafu 10.



Graf 10: Výsledný graf s výběrem agilní metodiky

Zdroj: vlastní zpracování (výstřižek z Expert Choice 2000)

V našem grafickém výstupu vidíme, že první příčku obsadila metodika Scrum. Stalo se tak především díky její detailnější propracovanosti metodiky. Dominanci Scrumu potvrzuje i fakt, že se jedná o nejpoužívanější agilní metodiku na světě.

V závěsu na druhé příčce vidíme metodiku XP (Extrémní programování), která sice na své oblíbenosti ubrala, ale její základní principy vyhovují pro stanovená kritéria.

Jako nejméně vyhovující ze zkoumaných metodik se umístil Kanban, který působí velice abstraktně a je vhodnější spíše jako doplňující metodika.

V rámci další diskuze by určitě nebylo od věci zapřemýšlet nad zkombinováním dvou metodik s tím, že základní kostru by tvořila metodika Scrum a byla by obohacena některými principy z jiné metodiky. Koneckonců, jak jsme mohli vidět v minulé kapitole, zavádění všelijakých hybridů je v módě.

8 Návrh na zavedení agilní metodiky

V minulé kapitole jsme pomocí vícekriteriální analýzy dospěli k tomu, že jako nejvhodnější pro implementaci ve společnosti GIST, s.r.o. je agilní metodika Scrum.

Záměrem této kapitoly je navrhnout základní principy Scrumu podpořené radami od odborníků z praxe. Jednou takovou je agilní koučka Zuzana Šochová, jejíž odborné publikace byly při zpracování této práce nápomocné.

Jelikož s řízením Scrumu ani jiných agilních metodik nemá ve firmě nikdo žádné zkušenosti, tak je doporučeno takového člověka najít. Nabízí se možnost hledat někoho z externího prostředí, ale je nutností počítat s vyššími náklady na jeho zaplacení. Nebo svěřit tento úkol někomu z vlastních řad a investovat do jeho sebevzdělávání v podobě různých školení a kurzů s agilní tématikou. Základním předpokladem všeho je chtít něco změnit a myslet jinak – agilně.

8.1 Definování rolí

8.1.1 Product Owner

V našem případě nelze zvolit učebnicového Product Ownera ze zákaznické komunity, nebo z externího prostředí, jelikož roli zákazníka nám v týmu představuje konzultant, který skrze zákazníka přebírá požadavky na vylepšení produktu. Mimo to produkt je tvořen pro více zákazníků najednou, tudíž konzultant nevyslýchá požadavky pouze od jednoho zákazníka. Návrhem tedy může být transformace jednoho konzultanta na Product Ownera, který bude shromažďovat požadavky od ostatních konzultantů a úzce s nimi spolupracovat.

8.1.2 ScrumMaster

Role ScrumMastera je ve Scrumu nezaměnitelná a pro správné fungování vyžadující. Pokud dojde k výběru ScrumMastera z vlastních řad, tak by dotyčná osoba měla mít především zájem o tuto roli a měla by mít přirozený respekt. Ten si nezíská povyšováním nad ostatními nebo konfliktními situacemi. Úplně postačí, když půjde ostatním příkladem a bude vkládat důvěru do celého týmu. Hlavním cílem ScrumMastera je, aby z týmu vytvořil samoorganizovaný tým. Přechod na jakoukoliv

agilní metodu vyžaduje velkou změnu ve způsobu myšlení. ScrumMaster by měl být průvodcem takové změny a měl by být vždy o jeden krok před týmem, aby dokázal lidi neustále motivovat a vytahovat je z jejich stereotypů. Skvělý ScrumMaster se stane leaderem tehdy, kdy pomáhá lidem kolem sebe stát se také leadery. Jedním z přístupů, který by ScrumMaster mohl aplikovat, je State of Mind model, který je popsán v následující podkapitole (Šochová & Daněk, 2018).

8.1.3 Scrum tým

Správný tým by neměl být složený pouze ze seniorských pozic, ale je dobré týmu dát jistý balanc a vyvážit tým natolik, aby dohromady zvládl jakoukoliv překážku. Agilně začínající týmy často mají za nešvar vytvářet malý vodopádový model v rámci sprintu tím, že začínají analýzou, poté ji vývojáři naprogramují a na konec testeři otestují. Správný Scrum tým by měl spolupracovat na každé User Story již od začátku. Analytik se zamýšlí nad návrhem a programátor se ho ptá, jak se má funkcionalita chovat. Pracují tak na ní současně. Doplňuje je tester, který se zamýšlí, jak by se daná funkcionalita dala rozbít a rovnou identifikuje chybové scénáře. V důsledku lepší komunikace by Scrum tým měl sedět pohromadě v jedné kanceláři, aby se členové týmu mohli v nesnázích poradit s ostatními. V agilním světě totiž platí, že můj nebo tvůj problém je náš problém (Šochová & Kunce, 2014).

8.1.4 Projektový manažer

Zavedením Scrumu se role projektového manažera neruší. Ten se stará o správné reportování projektu ve všech systémech firmy, dále se stará o sledování rozpočtu a může sledovat odpracovaný čas na produktu. Co by už projektový manažer neměl dělat je zasahování do práce týmu a jeho řízení (Šochová & Kunce, 2014).

8.2 State of Mind model

ScrumMaster by měl poznat, v jakém stavu se tým nachází a podle toho přizpůsobit svůj přístup. Vybírat může ze čtyř základních přístupů, které popisuje model State of Mind:

- **Vysvětlování, učení a sdílení zkušeností –** zejména na začátku agilní transformace je zapotřebí týmu neustále opakovat, proč vůbec Scrum dělá a co od takové změny lze očekávat. Jakmile tým nabude základní zkušenosti, tak ScrumMaster sdílí své zkušenosti s ostatními a tým učí nové praktiky.

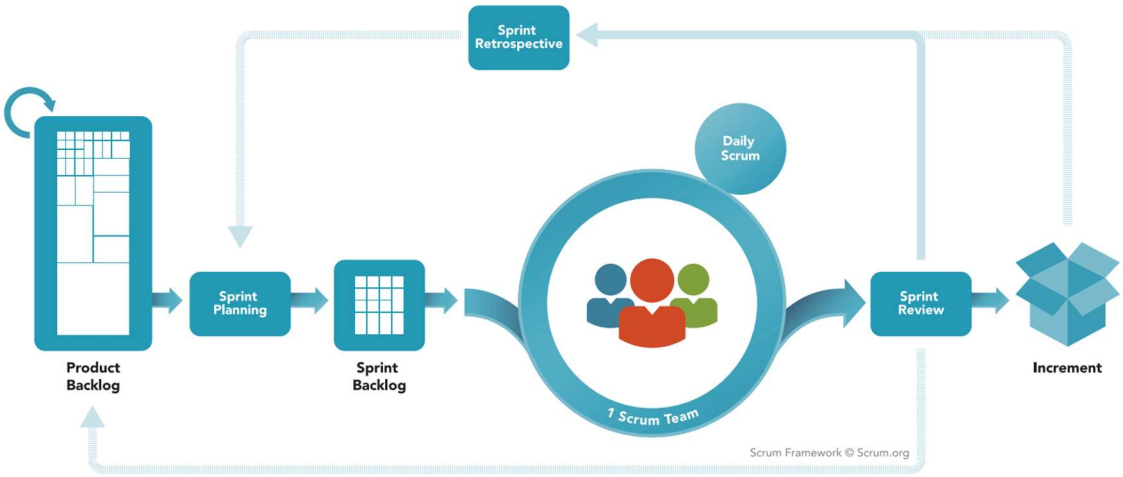
- **Odstraňování překážek –** ScrumMaster by měl pomáhat týmu odstraňovat zbytečné překážky v cestě a stát se efektivnějším. Odstraňuje je způsobem, že zapojí do řešení celý tým, aby se naučil řešit problémy sám. V žádném případě ScrumMaster nepřebírá zodpovědnost za tým.
- **Facilitace –** díky facilitaci je komunikace efektivnější. Každý meeting by měl mít jasně definovaný cíl, výstup a očekávaný výsledek.
- **Koučování –** jedná se o nejdůležitější dovednost. ScrumMaster potřebuje hodně praxe, aby zvládl tým koučovat ve správnou chvíli.

Záleží na povaze ScrumMastera, který z přístupů mu je v dané situaci bližší. Měl by použít takový přístup, který podpoří dosažení aktuálního cíle a přispěje k posílení samoorganizace týmu.

Jakkoliv jsou zmiňované čtyři přístupy důležité, tak bez jednoho základního přístupu se ScrumMaster neobejde. Je jím pozorování. State of Mind model učí ScrumMastera nekonat horlivě, ale zachovat chladnou hlavu a chvíli jen pozorovat a poté se rozhodnout pro vhodný přístup. „Pozorování, naslouchání a schopnost dát týmu prostor jsou nejdůležitějšími aspekty práce skvělého ScrumMastera“ (Šochová & Daněk, 2018).

8.3 Vývojový proces řízený metodikou Scrum

Na další straně se nachází obr. 20, který znázorňuje průběh vývojového cyklu řízený metodikou Scrum. Celý cyklus lze rozdělit do tří fází. V první fázi dojde k sestavení celkového Product Backlogu, tedy toho, co bude v rámci projektu vytvořeno. Druhá nejdelší fáze je Sprint, který probíhá dokola do té doby, než z Product Backlogu zmizí všechna User Stories. V poslední fázi by se měl produkt finálně otestovat a v případě, kdy je produkt akceptován Product Ownerem, tak může být vydána, v případě hradecké společnosti, nová verze. Jednotlivé artefakty a důležité meetingy byly již rozepsány v podkapitole 4.6, nyní bude cílem je doplnit a uvést pár „best practices“ Scrumu vhodných zkoumané společnosti.



Obr. 20: Životní cyklus metodiky Scrum

Zdroj: Výstřižek z ('The Scrum Framework Poster')

8.3.1 Sprint

Sprint ve Scrumu je každá iterace, která se opakuje při vývoji softwaru, dokud není Product Backlog prázdný. Vydání nové verze aplikace GIST Intelligence v současné době trvá zhruba dva měsíce, pro začátek tedy můžeme navrhnout třítýdenní délku sprintu.

8.3.2 Daily Scrum

Ve zkoumané společnosti je nyní pouze na každý týden naplánováno týdenní review, kde vedoucí projektu GIST Intelligence rekapituluje současný stav. Proto by byla vhodná zavést denní schůzka, někdy nazývána jako Stand-up, která se koná každý den. Jedná se o krátkou schůzku členů týmu, kde ve stoje rekapitulují, co dělali minulý den a co budou dělat v probíhajícím dni. Stand-up by neměl přesáhnout délku patnácti minut. Mělo by být docíleno zvýšení informovanosti týmu.

8.3.3 Plánovací poker

Ve chvíli, kdy je Product Owner spokojený s nastaveným Product Backlogem, který obsahuje všechny funkcionality v podobě User Stories, tak přichází na řadu ohodnocení náročnosti dokončení každé funkcionality. V rámci Scrumu lze použít metodu Planning Poker, kde Product Owner nejdříve představí funkcionalitu tak, aby všichni členové týmu pochopili její podstatu. Následně každý člen týmu zvolí jednu kartu s bodovým ohodnocením a položí ji na stůl číslem dolů. Planning poker karty obvykle obsahují: 0, ½, 1, 3, 5, 8, 13, 20, 40, 100, ?, nekonečno a kartu s kávou. Karta nekonečna slouží pro

příliš komplexní funkcionality, kartu s otazníkem členové použijí v případě, že ještě neví a rádi by položili doplňující otázku Product Ownerovi. Karta s kávou poslouží pro pěti minutovou přestávku.

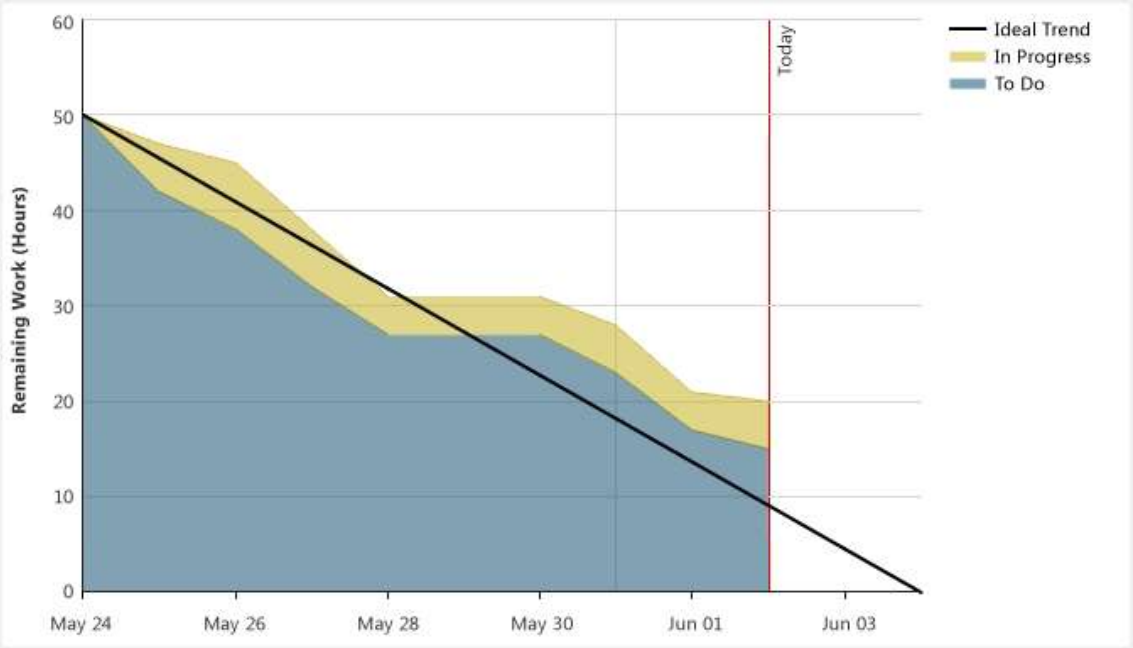
Pokud mají všichni členové týmu vybranou kartu, tak ScrumMaster zavelí pro jejich otočení. ScrumMaster se poté ptá člena týmu s nejvyšším hodnocením, proč si myslí, že se jedná o tak náročný požadavek a zároveň se ptá člena týmu s nejnižším hodnocením, proč si myslí, že bude daná funkcionalita příliš jednoduchá. Oba členové vysvětlí, proč tak učinili a snaží přehodnotit bodová hodnocení svých kolegů. Cílem všeho je se shodnout na stejném bodovém ohodnocení a stanovit všem User Stories bodové ohodnocení (Šochová & Kunce, 2014).

8.3.4 Rychlost

Jakmile skončí Sprint, tak dochází k zúčtování, přesněji řečeno tým sečte bodové ohodnocení všech User Stories, které tým za uplynulou iteraci stihl dokončit. Výsledkem je rychlost neboli velocity. Funkcionality, které tým nestihl dokončit se přesouvají zpět do Product Backlogu. Na začátku se dá očekávat, že rychlost bude oscilovat, je vhodné proto volit průměrnou rychlost ze tří posledních Sprintů. S přibývajícími zkušenostmi se rychlost týmu stabilizuje a je tak předvídatelný. Velké výkyvy rychlosti jsou příznakem nedostatečné spolupráce. Vždy se měří rychlost za celý tým a Sprint, nikoliv za jednotlivce (Šochová & Kunce, 2014).

8.3.5 Burndown graf

Pro zobrazení stavu celého projektu a predikci jeho dokončení slouží tzv. Burndown graf. Jedná se o sloupcový graf, ze kterého tým v rámci Sprintu „pálí“ určité množství funkcionalit a které byly předtím ohodnoceny v rámci Planning Pokeru. Postupem času a vykonanou prací se sloupce snižují a množství dokončené práce každým Sprintem roste. Jak rychle se bude Produckt Backlog „spalovat“ záleží na odhadu rychlosti (velocity). Příklad Burndown grafu je znázorněn v grafu 11. Jedná se o graf z nástroje Team Foundation Server od společnosti Microsoft, který zkoumaná společnost používá (Šochová & Kunce, 2014).



Graf 11: Burndown graf

Zdroj: (KathrynEE, n.d.)

V tomto případě horizontální osa znázorňuje dny ve sprintu a vertikální osa ukazuje zbývající práci v hodinách. Černá čára vykresluje ideální situaci, kdy tým v rámci Sprintu „spálí“ všechny požadavky za konstantní rychlosti. Skupina In Progress ukazuje, kolik hodin zbývá pro úkoly, které jsou rozpracovány v daném Sprintu. Skupina To Do zobrazuje, kolik hodin zbývá pro úkoly, které je potřeba v rámci Sprintu dodělat. Skupiny In Progress a To Do jsou vykresleny na základě skutečného vývoje plnění požadavků (KathrynEE, n.d.).

8.3.6 Ohodnocování týmu

V současné situaci jsou jednotlivci sledování a hodnoceni dle tvrdých (hard) metrik. Příkladem může být hodnocení vývojáře podle řádků napsaného kódu nebo počtu chyb. Tester bývá hodnocen na základě počtu nalezených chyb. Ovšem toto není hodno agilního myšlení a pro zaměstnance bývá deformující. Ve Scrumu se naopak používají měkké (soft) metriky. Po každém sprintu jsou členové týmu vyzváni ScrumMasterem, aby ohodnotili sami sebe na stupnici od jedné do deseti a poté dostanou zpětnou vazbu od ScrumMastera. Je vhodné vyvarovat se diskuzím, proč je to pětka, lepší je vysvětlit, co musí udělat proto, aby to byla např. devítka. Product Ownera hodnotí celý tým opět

na stupnici od jedné do deseti, kde jedna znamená „nic moc“ a deset „super“ (Šochová & Kunce, 2014).

Radou na závěr je zvýšit pozitivní myšlení uvnitř týmu, protože na problémy se dá dívat i z té lepší stránky. Každý jistě zná, že o sklenici se dá říct, že je poloprázdná, či poloplná. Tým by si měl připomínat, co dobrého se mu povedlo a na co je pyšný. Jako pomůcka k zvýšení pozitivity slouží tzv. zeď pozitivity, která zviditelňuje naše úspěchy a které by se měli společně zapít (Šochová & Daněk, 2018).

9 Shrnutí výsledků

Diplomová práce se zaměřila na přehledné a srozumitelné popsání tradičního a agilního vývoje softwaru. Hlavním cílem bylo vybrat a navrhnout vhodnou agilní metodiku pro hradeckou softwarovou společnost GIST, s.r.o., která by ráda změnila řízení svých projektů a přešla tak z vodopádového modelu životního cyklu na některou z agilních metodik.

Aby bylo možné navrhnout některou agilní metodiku, tak bylo zapotřebí v prvé řadě načerpat vědomosti z odborné literatury. Popsán byl obor softwarové inženýrství, který zažil vzestup v šedesátých letech a na který nebyl evidentně připraven. Zakázky na vývoj softwaru rostly rychlým tempem a nestíhaly se dokončovat. Toto období je označováno softwarovou krizí, která byla hlavní událostí pro vznik softwarového inženýrství.

Pro navržení vhodné agilní metodiky musela být nejprve provedena analýza samotné firmy, kterou najdeme v páté kapitole. Dozvěděli jsme se, jaké má firma produkty ve svém portfoliu, kdo je jejím zákazníkem a také jsme si popsali jednotlivé role v podniku. Především byl popsán celý proces vývoje nové verze pro své zákazníky, který postupuje vodopádovou metodou. I když se vývoj drží více tradičního postupu, tak firma využívá některých agilních prvků a nástrojů, jako např. nástroj Team Foundation Server (TFS), který podporuje agilní řízení projektů.

Případová studie se věnovala používáním agilních metodik po celém světě, kde jsme se snažili zachytit nastupující trendy a zaznamenat, jak se vyvíjí agilní myšlení. Ke zpracování výsledků posloužila data z reportů, které provádí společnost CollabNet VersionOne každým rokem již od roku 2006. Závěry byly přehledně zachyceny pomocí grafů. Zkušenosti s agilním přístupem ve firmě uvedlo 97 % respondentů za rok 2017, v roce 2011 jich bylo 80 %. Narůstající trend jsme rovněž zaznamenali u doby používání agilního přístupu, ještě v roce 2010 bylo 60 % firem, které nepoužívaly agilní metodiky déle než dva roky, v roce 2017 se karta obrátila a 65 % firem používá agilní přístup déle než dva roky. Mezi hlavní důvody pro zavedení agilní metodiky je dlouhodobě zrychlení dodávky softwaru a zlepšení schopnosti měnit požadavky. Všem

agilním metodikám vévodí Scrum, který používá ve všech různých modifikacích kolem 70 % podniků. Nástrojům pro řízení projektů vévodí JIRA od společnosti Atlassian.

Dále se aplikační část zaměřila na výběr vhodné agilní metodiky vývoje softwaru, která byla doporučena k implementaci v podniku GIST, s.r.o. Výběr byl proveden vícekriteriální analýzou za podpory programu Expert Choice 2000. Na počátku pět zaměstnanců (expertů) firmy párově ohodnotilo vybraná kritéria a na základě těchto výsledků byly pomocí Saatyho matice stanoveny jednotlivé váhy. Z hodnocených tří agilních metodik (Scrum, Kanban a XP) byla vybrána jako nejvíce vyhovující vůči vybraným kritériím agilní metodika Scrum.

Diplomová práce splnila deklarovaný cíl, hlavním přínosem je výběr a návrh vhodné agilní metodiky pro vybranou společnost GIST, s.r.o.

10 Závěry a doporučení

Téma diplomové práce bylo vypracováno na základě zadání hradecké společnosti GIST, s.r.o, která vycítila potřebu zvýšit efektivitu při řízení procesu vývoje aplikace GIST Intelligence. V průběhu zpracování tématu probíhala úzká spolupráce s produktovým ředitelem Ing. Filipem Ježkem.

Výzkum byl opřen o výsledky získané z literatury, na které navázalo vlastní šetření. Doporučení z případové studie není nikterak omezeno, výsledky jsou určeny všem subjektům, které zajímá vývoj agilního přístupu ve světě.

Lze konstatovat, že jsme zemí, kam většina trendů přichází se zpožděním. Jinak tomu není ani u agilních metodik, které přichází především ze Spojených států amerických, kde se zrodil Agilní manifest a pochází odtud většina metodik. Díky tomuto celosvětovému průzkumu je možné predikovat vývoj agilního přístupu i u nás. V rámci dalšího průzkumu by bylo zajímavé provést dotazníkové šetření se zachovanou strukturou dotazníku VersionOne pouze na našem území a poté porovnat výsledky s celým světem. Nápad již jednou zrealizovala společnost Etnetera v roce 2013, kdy 171 respondentů odpovídalo na otázky zaměřené na agilní vývoj.

Výběr vhodné metodiky se vztahoval pouze na zkoumanou společnost a její produkt, kde došlo k expertnímu vybrání kritérií za spolupráce produktového ředitele firmy. Doporučení výběru metodiky Scrum je tak omezeno pouze pro společnost GIST, s.r.o.

Dospěním k výběru metodiky Scrum se potvrdil fakt, že se jedná o nejpoužívanější agilní metodiku na světě. V rámci dalšího doporučení může být rozšíření metodiky Scrum o další principy plynoucí z jiných agilních metodik. Řadě společností umožňuje zavedení agilního přístupu být konkurenceschopnými, zvyšovat tržní hodnotu firmy a stabilizovat klíčové zaměstnance.

11 Seznam grafů

12 Seznam obrázků

13 Seznam tabulek

14 Seznam příloh

15 Seznam použité literatury

Agile 101. (c2019). Retrieved from Agile Alliance website: https://www.agilealliance.org/agile101/

Agile Practices Timeline. (c2019). Retrieved from Agile Alliance website: https://www.agilealliance.org/agile101/practices-timeline/

Beck, K., & Makovec, P. (2002). *Extrémní programování*. Praha: Grada Publishing.

Continuos Integration. (2006, 5). Retrieved from Martin Fowler website: https://www.martinfowler.com/articles/continuousIntegration.html

Fiala, P., Jablonský, J., & Maňas, M. (1994). *Vícekriteriální rozhodování: Určeno pro stud. všech fakult VŠE Praha*. Praha: Vysoká škola ekonomická.

Fotr, J., & Švecová, L. (2016). *Manažerské rozhodování: postupy, metody a nástroje*.

Kadlec, V. (2004). *Agilní programování: metodiky efektivního vývoje softwaru*. Brno: Computer Press.

Kanban. (c2019). Retrieved from Agile Alliance website: https://www.agilealliance.org/glossary/kanban/

KathrynEE. (n.d.). Sprint Burndown (Scrum) - TFS. Retrieved 26 April 2019, from https://docs.microsoft.com/en-us/azure/devops/report/sql-reports/sprint-burndown-scrum

Manifesto for Agile Software Development. (2001). Retrieved from https://agilemanifesto.org/

MoSCoW Prioritisation. (c2019). Retrieved from Agile Business website: https://www.agilebusiness.org/content/moscow-prioritisation

Myslín, J. (2016). *Scrum: průvodce agilním vývojem softwaru*.

Pair Programming. (c2019). Retrieved from Agile Alliance website: https://www.agilealliance.org/glossary/pairing/

Philosophy and Fundamentals. (c2019). Retrieved from Agile Business website: https://www.agilebusiness.org/content/philosophy-and-fundamentals

Poppendieck, M., & Poppendieck, T. (2010). *Lean software development: an agile toolkit* (Nachdr.). Boston: Addison-Wesley.

Principles behind the Agile Manifesto. (n.d.). Retrieved from https://agilemanifesto.org/iso/en/principles.html

Procházka, J. (2014, 8). Agilní projekty z pohledu zákazníka. Retrieved from Webová integrace website: http://www.web-integration.info/cs/blog/agilni-projekty-zpohledu-zakaznika/

Rational Unified Process; Best Practices for Software Development Teams. (1998). Retrieved from https://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

Schwaber, K., & Sutherland, J. (2013). *The Scrum Guide, Průvodce Scrumem: Pravidla hry*. Retrieved from https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-CS.pdf

Schwaber, K., & Sutherland, J. (2017). *The Scrum Guide, The Definitive Guide to Scrum: The Rules of the Game*. Retrieved from https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

Šochová, Z., & Daněk, M. (2018). *Skvělý ScrumMaster*.

Šochová, Z., & Kunce, E. (2014). *Agilní metody řízení projektů*. Brno: Computer Press.

Software engineering history. (2008). Retrieved from http://www.SoftwareEngineering-9.com/Web/History/

Sommerville, I. (2013). *Softwarové inženýrství*. Brno: Computer Press.

Takeuchi, H., & Nonaka, I. (1986). The New New Product Development Game. *Harvard Business Review 64, No. 1*, *(January-February 1986)*.

TDD. (c2019). Retrieved from Agile Alliance website: https://www.agilealliance.org/glossary/tdd/

The Scrum Framework Poster. (n.d.). Retrieved 25 April 2019, from Scrum.org website: https://www.scrum.org/resources/scrum-framework-poster

Tománek, M. (2015). Současný stav používání agilních metodik ve světě a v ČR. *Acta Informatica Pragensia*, *4*(1), 4–17. https://doi.org/10.18267/j.aip.48

VersionOne. (2007). *The State of Agile Development*. Retrieved from https://www.versionone.com/pdf/2006-state-of-agile-survey.pdf

VersionOne. (2008). *2nd Annual Survey 'The State of Agile Development'*. Retrieved from https://www.versionone.com/pdf/2007-state-of-agile-survey.pdf

VersionOne. (2009). *3rd Annual Survey: 2008 'The State of Agile Development'*. Retrieved from https://www.versionone.com/pdf/2008-state-of-agile-survey.pdf

VersionOne. (2010). *4th Annual State of Agile Survey: 2009*. Retrieved from https://www.versionone.com/pdf/2009-state-of-agile-survey.pdf

VersionOne. (2011). *5th Annual State of Agile Survey: 2010*. Retrieved from https://www.versionone.com/pdf/2010-state-of-agile-survey.pdf

VersionOne. (2012). *6th Annual State of Agile Survey: 2011*. Retrieved from https://www.versionone.com/pdf/2011-state-of-agile-survey.pdf

VersionOne. (2013). *7th Annual State of Agile Development Survey*. Retrieved from https://www.versionone.com/pdf/2012-state-of-agile-survey.pdf

VersionOne. (2014). *8th Annual State of Agile Survey*. Retrieved from https://www.versionone.com/pdf/2013-state-of-agile-survey.pdf

VersionOne. (2015). *9th Annual State of Agile Survey*. Retrieved from https://www.versionone.com/pdf/2014-state-of-agile-survey.pdf

VersionOne. (2016). *10th Annual State of Agile Report*. Retrieved from https://www.versionone.com/pdf/2015-state-of-agile-survey.pdf

VersionOne. (2017). *11th Annual State of Agile Report*. Retrieved from https://www.versionone.com/pdf/2016-state-of-agile-survey.pdf

VersionOne. (2018). *12th Annual State of Agile Report*. Retrieved from https://www.versionone.com/pdf/2017-state-of-agile-survey.pdf

16 Přílohy

Příloha A – Vzor dotazníku párového ohodnocení

Dotazník k diplomové práci

Vítejte u krátkého dotazníku, který byl vytvořen za účelem zjištění Vámi vnímané důležitosti jednotlivých kritérií u vývoje aplikace. Průzkum poslouží pro **výběr vhodné agilní metodiky** vývoje softwaru. Tento dotazník má za cíl pouze správně nadefinovat jednotlivé váhy kritérií, ke kterým budou následně přiděleny vybrané agilní metodiky a mezi sebou posouzeny.

Dotazník je zcela anonymní a výsledky budou zveřejněny pouze v diplomové práci.

Pro výběr vhodné metodiky byla vybrána následující kritéria:

- **Míra složitosti metodiky** – čím složitější metodika je, tím více času trvá adaptace na ni.
- **Míra dokumentace** – přiměřená míra dokumentace, vymanit se psaní zbytečné a nepotřebné dokumentace.
- **Míra chybovosti** – nadbytečná chybovost je neefektivní a zabírá tak volnou kapacitu.
- **Specifikace požadavků zákazníka** – porozumění zákazníkových požadavků pro jeho spokojenost. Chybně pochopené požadavky jsou příčinou zbytečně vykonané práce.
- **Rychlost dodávky** – vydání buildu v domluveném termínu.
- **Definování zodpovědnosti** – přidělení zodpovědností jedincům, či celému týmu.
- **Komunikace napříč týmem** – vzájemná interakce zvyšuje efektivitu celého týmu a problémy jsou řešeny hned.

V následující části budou proti sobě postavena vždy dvě kritéria. **Zakroužkujte**, prosím, takovou hodnotu, o kterou si myslíte, že je jedno kritérium důležitější než to druhé.

Vysvětlivky: 1 = stejně důležitá kritéria, 3 = slabě preferované kritérium, 5 = silně preferované kritérium, 7 = významně preferované kritérium, 9 = absolutně preferované kritérium

Míra složitosti metodiky vs. Míra dokumentace

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra složitosti metodiky vs. Míra chybovosti

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra složitosti metodiky vs. Specifikace požadavků zákazníka

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra složitosti metodiky vs. Rychlost dodávky

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra složitosti metodiky vs. Definování zodpovědnosti

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra složitosti metodiky vs. Komunikace napříč týmem

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra dokumentace vs. Míra chybovosti

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra dokumentace vs. Specifikace požadavků zákazníka

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra dokumentace vs. Rychlost dodávky

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra dokumentace vs. Definování zodpovědnosti

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra dokumentace vs. Komunikace napříč týmem

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra chybovosti vs. Specifikace požadavků zákazníka

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra chybovosti vs. Rychlost dodávky

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra chybovosti vs. Definování zodpovědnosti

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Míra chybovosti vs. Komunikace napříč týmem

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Specifikace požadavků zákazníka vs. Rychlost dodávky

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Specifikace požadavků zákazníka vs. Definování zodpovědnosti

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Specifikace požadavků zákazníka vs. Komunikace napříč týmem

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Rychlost dodávky vs. Definování zodpovědnosti

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Rychlost dodávky vs. Komunikace napříč týmem

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Definování zodpovědnosti vs. Komunikace napříč týmem

9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

Děkuji Vám za vyplnění dotazníku a přeji mnoho úspěchů a zdraví do dalších let.
Bc. František Linder

Příloha B – Saatyho matice a dopočtené váhy kritérií

Vysvětlivky:

K_1 – Míra složitosti metodiky

K_2 – Míra dokumentace

K_3 – Míra chybovosti

K_4 – Specifikace požadavků zákazníka

K_5 – Rychlost dodávky

K_6 – Definování zodpovědnosti

K_7 – Komunikace napříč týmem

Kritérium	K_1	K_2	K_3	K_4	K_5	K_6	K_7	Geometrický průměr	Výsledné váhy
K_1	**1**	1	1/5	1/3	1/3	1/3	1/3	0,424	0,048
K_2	1	**1**	1/5	1/3	1/5	1/3	1/5	0,367	0,041
K_3	5	5	**1**	1	1	5	5	2,508	0,282
K_4	3	3	1	**1**	1	5	1	1,723	0,194
K_5	3	5	1	1	**1**	5	1	1,853	0,208
K_6	3	3	1/5	1/5	1/5	**1**	1/5	0,546	0,061
K_7	3	5	1/5	1	1	5	**1**	1,472	0,166
∑								**8,893**	**1,000**

Tabulka 2: Saatyho matice – respondent č. 1

Zdroj: vlastní zpracování

Kritérium	K_1	K_2	K_3	K_4	K_5	K_6	K_7	Geometrický průměr	Výsledné váhy
K_1	**1**	1/4	1/5	1/7	1/5	1/2	1/5	0,282	0,031
K_2	4	**1**	1/3	1/7	1/5	1	1/3	0,536	0,059
K_3	5	3	**1**	1/3	1/3	1/2	1	0,974	0,107
K_4	7	7	3	**1**	5	3	3	3,514	0,387
K_5	5	5	3	1/5	**1**	1	1	1,472	0,162
K_6	2	1	2	1/3	1	**1**	1	1,042	0,115
K_7	5	3	1	1/3	1	1	**1**	1,258	0,139
∑								**9,079**	**1,000**

Tabulka 3: Saatyho matice – respondent č. 2

Zdroj: vlastní zpracování

Kritérium	K_1	K_2	K_3	K_4	K_5	K_6	K_7	Geometrický průměr	Výsledné váhy
K_1	**1**	1/7	1/5	1/8	1/5	1/2	1/3	0,275	0,028
K_2	7	**1**	1	1/5	1/3	5	1/4	0,926	0,095
K_3	5	1	**1**	1	1/5	5	5	1,584	0,162
K_4	8	5	1	**1**	1	5	3	2,494	0,255
K_5	5	3	5	1	**1**	7	7	3,231	0,331
K_6	2	1/5	1/5	1/5	1/7	**1**	1/5	0,333	0,034
K_7	3	4	1/5	1/3	1/7	5	**1**	0,923	0,095
∑								**9,766**	**1,000**

Tabulka 4: Saatyho matice – respondent č. 3

Zdroj: vlastní zpracování

Kritérium	K_1	K_2	K_3	K_4	K_5	K_6	K_7	Geometrický průměr	Výsledné váhy
K_1	**1**	1/6	1/7	1/6	1/5	2	1/5	0,316	0,029
K_2	6	**1**	1/6	1/7	1	1/3	1/5	0,514	0,048
K_3	7	6	**1**	1	8	8	3	3,615	0,335
K_4	6	7	1	**1**	8	6	6	3,830	0,355
K_5	5	1	1/8	1/8	**1**	1/5	1/5	0,439	0,041
K_6	1/2	3	1/8	1/6	5	**1**	1	0,767	0,071
K_7	5	5	1/3	1/6	5	1	**1**	1,319	0,122
∑								**10,801**	**1,000**

Tabulka 5: Saatyho matice – respondent č. 4

Zdroj: vlastní zpracování

Kritérium	K_1	K_2	K_3	K_4	K_5	K_6	K_7	Geometrický průměr	Výsledné váhy
K_1	**1**	1/7	1/8	1/8	1/8	1/5	1/7	0,187	0,017
K_2	7	**1**	1/7	1/6	1/8	1/8	1/9	0,312	0,028
K_3	8	7	**1**	1/5	5	5	5	2,815	0,249
K_4	8	6	5	**1**	5	5	5	4,361	0,386
K_5	8	8	1/5	1/5	**1**	6	3	1,728	0,153
K_6	5	8	1/5	1/5	1/6	**1**	1/5	0,658	0,058
K_7	7	9	1/5	1/5	1/3	5	**1**	1,228	0,109
∑								**11,289**	**1,000**

Tabulka 6: Saatyho matice – respondent č. 5

Zdroj: vlastní zpracování

Vysvětlivky:

R_1 – respondent č. 1

R_2 – respondent č. 2

R_3 – respondent č. 3

R_4 – respondent č. 4

R_5 – respondent č. 5

Kritérium	R_1	R_2	R_3	R_4	R_5	Průměr
K_1	0,048	0,031	0,028	0,029	0,017	**0,031**
K_2	0,041	0,059	0,095	0,048	0,028	**0,054**
K_3	0,282	0,107	0,162	0,335	0,249	**0,227**
K_4	0,194	0,387	0,255	0,355	0,386	**0,315**
K_5	0,208	0,162	0,331	0,041	0,153	**0,179**
K_6	0,061	0,115	0,034	0,071	0,058	**0,068**
K_7	0,166	0,139	0,095	0,122	0,109	**0,126**
∑	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**	**1,000**

Tabulka 7: Stanovení vah pomocí aritmetického průměru

Zdroj: vlastní zpracování

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2018/2019

Studijní program: Systémové inženýrství a informatika
Forma: Prezenční
Obor/komb.: Informační management (im2-p)

Podklad pro zadání DIPLOMOVÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Bc. Linder František	Na Břehách 397/26, Hradec Králové - Třebeš	I1600933

TÉMA ČESKY:

Zavedení agilní metodiky vývoje softwaru

TÉMA ANGLICKY:

Implementing Agile Software Development Methods

VEDOUCÍ PRÁCE:

doc. Ing. Hana Mohelská, Ph.D. - KM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl diplomové práce:
Cílem práce je popsat agilní metodiky řízení vývoje softwaru, jejich hlavní rozdíly od vodopádových metodik a navrhnout postup, jak by měla softwarová společnost postupovat při přechodu od vodopádové k agilní metodice.

Osnova:
1) Úvod
2) Cíl práce a metodika
3) Literární rešerše
4) Analýza současné situace vývojářského týmu
5) Porovnání metod a výběr optimální agilní metodiky
6) Návrh na zavedení vybrané agilní metodiky
7) Závěry a doporučení
8) Seznam literatury
9) Přílohy

SEZNAM DOPORUČENÉ LITERATURY:

[1] KADLEC, Václav. Agilní programování: metodiky efektivního vývoje softwaru. Brno: Computer Press, 2004. ISBN 80-251-0342-0.
[2] MARTIN, Robert C. Agile software development, principles, patterns, and practices. Harlow, Essex: Pearson, 2014. ISBN 978-1-292-02594-0.
[3] MEYER, Bertrand. Agile!: the good, the hype and the ugly. New York: Springer, 2014. ISBN 978-3-319-05154-3.
[4] MYSLÍN, Josef. Scrum: průvodce agilním vývojem softwaru. Brno: Computer Press, 2016. ISBN 978-80-251-4650-7.
[5] PROCHÁZKA, Jaroslav a Cyril KLIMEŠ. Provozujte IT jinak: agilní a štíhlý provoz, podpora a údržba informačních systémů a IT služeb. Praha: Grada, 2011. Průvodce (Grada). ISBN 978-80-247-4137-6.
[6] ŠOCHOVÁ, Zuzana a Eduard KUNCE. Agilní metody řízení projektů. Brno: Computer Press, 2014. ISBN 9788025141946.
[7] ŠOCHOVÁ, Zuzana. Skvělý ScrumMaster. Přeložil Milan DANĚK. Brno: Computer Press, 2018. ISBN 978-80-251-4927-0.
[8] WYSOCKI, Robert K. Effective project management: traditional, agile, extreme. 5th ed. Indianapolis, IN: Wiley Publishing, c2009. ISBN 978-0-470-42367-7.

Odborné vědecké články v databázích WoS, Scopus a Medline.

Podpis studenta: ..... Datum: 25.2.2019

Podpis vedoucího práce: ..... Datum: 25. 2. 2019