České vysoké učení technické v Praze
Masarykův ústav vyšších studií
a
Vysoká škola ekonomická v Praze

Podnikání a komerční inženýrství v průmyslu

Martin Popelák

Využití metodiky SCRUM ve vývojových týmech

Diplomová práce

Praha 2015

Vedoucí diplomové práce:Doc. Ing. Dalibor Vytlačil, CSc.

Oponent diplomové práce:

Datum obhajoby:

Hodnocení:

ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
MASARYKŮV ÚSTAV VYŠŠÍCH STUDIÍ
a
VYSOKÁ ŠKOLA EKONOMICKÁ V PRAZE

Zadání diplomové práce

Školní rok: 2013/2014

Jméno a příjmení: Martin Popelák

Studijní program: Podnikání a komerční inženýrství v průmyslu

Obor studia: Podnikání a management v průmyslu

Forma studia: kombinovaná

Téma práce: Využití metodiky SCRUM ve vývojových týmech

Téma práce v anglickém jazyce: Usage of SCRUM in development teams

Zásady pro vypracování práce

Cíl práce (stručné vymezení zkoumaného problému): Cílem diplomové práce je popsat základní teoretické principy fungování metodiky SCRUM a obecně agilní filozofie při vývoji software. Na konkrétní modelové situaci prakticky zkoumat dopady změny fungování středně velké firmy při přechodu z rigorózní metodiky na metodiku agilní.

Teoretická východiska: Hlavním teoretickým východiskem je předmět Procesní řízení na MÚVS, který jako jeden z hlavních bodů diskutuje rozmanitost metodik pro řízení, nejen softwarových projektů. Dalším východiskem je předmět Projekt studie proveditelnosti, který již konkrétně analyzuje dopady při použití některé metodiky.

Metody práce: Pro praktickou část jsem si zvolil metodu pozorování a metodu srovnávání. Praktická část diplomové práce se zabývá pozorováním vývojového týmu před, během a po změně vývojové metodiky z rigorózní na agilní. Dále pak budu srovnávat naměřené metriky a klíčové ukazatele výkonu před a po změně metodiky.

Rámcová osnova:

1 Softwarové inženýrství

2 Rigorózní metodiky

3 Agilní metodiky

4 Analýza současného stavu procesů ve firmě

5 Návrh změny a optimalizace procesů

6 Exekuce a vyhodnocení změn procesů

Základní odborná literatura:

COHN, Mike. Succeeding with agile: software development using Scrum. Upper Saddle River, NJ: Addison-Wesley, c2010, xxviii, 475 p. Addison-Wesley signature series. ISBN 03-215-7936-4.

GOLDSTEIN, Ilan. Scrum shortcuts without cutting corners: agile tactics, tools. Upper Saddle River, NJ: Addison-Wesley, c2012, pages cm. ISBN 978-032-1822-369.

JOHNSON, By Chris Sims. Scrum: a breathtakingly brief and agile introduction. Upper Saddle River, NJ: Addison-Wesley, c2010, xxviii, 475 p. Addison-Wesley signature series. ISBN 978-193-7965-044.

KROLL, Per a Philippe KRUCHTEN. The rational unified process made easy: a practitioner's guide to the RUP. Boston: Addison-Wesley, c2003, xxxv, 416 p. Addison-Wesley signature series. ISBN 03-211-6609-4.

LACEY, Mitch. The scrum field guide: practical advice for your first year. 1st ed. Upper Saddle River, NJ: Addison-Wesley, c2012, xxxi, 378 p. ISBN 03-215-5415-9.

LAYTON, Mark. Agile project management for dummies: a practical guide to the most popular agile process. Hoboken, N.J.: Wiley, c2012, xiv, 346 p. Addison-Wesley signature series. ISBN 978-111-8235-850.

RUBIN, Kenneth S. Essential Scrum: a practical guide to the most popular agile process. Upper Saddle River, NJ: Addison-Wesley, c2012, xliii, 452 p. ISBN 01-370-4329-5.

SKARIN, Henrik Knibert a Forewords by Mary Poppendieck ANDERSON. Kanban and Scrum: making the most of both. S.l.: C4Media, Inc, 2010, xiv, 346 p. Addison-Wesley signature series. ISBN 978-055-7138-326.

Vedoucí práce: Doc. Ing. Dalibor Vytlačil, CSc.

Podpis vedoucího práce:

Datum odevzdání zadání: 6.12.2013

Datum odevzdání práce:

Podpis studenta stvrzující přijetí zadání práce:

Toto zadání platí tři po sobě jdoucí semestry od data odevzdání zadání.

Schválení zadání DP

27.1.2014

Datum a podpis vedoucího programu

Podpis ředitele MÚVS

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a že jsem uvedl všechny použité informační zdroje.

V Praze, 8. 6. 2015

……..
Martin Popelák

Poděkování

Rád bych poděkoval Doc. Ing. Daliboru Vytlačilovi, CSc. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování diplomové práce.

Identifikační záznam

Martin Popelák. *Využití metodiky SCRUM ve vývojových týmech.* Praha, 2015. 112 stran, 3 strany příloh. Diplomová práce. České vysoké učení technické v Praze, Masarykův ústav vyšších studií a Vysoká škola ekonomická v Praze, Podnikání a komerční inženýrství v průmyslu. Jméno vedoucího diplomové práce – doc. Ing. Dalibor Vytlačil, CSc.

Abstrakt

Tato práce se věnuje popisu a rozdělení agilních metodik používaných pro vývoj softwaru. Práce popisuje jejich historii a srovnání s tradičními metodikami, někdy označovanými jako rigorózní. Největší část teoretické části práce popisuje agilní metodiku Scrum. Metodika Scrum je nejpoužívanější agilní metodikou v oblasti vývoji softwaru. Práce se věnuje detailnímu popisu jednotlivých praktik, schůzek a rolí, které by měl tým dodržovat pro úspěšnou aplikaci metodiky. Teoretická část se také věnuje možným nástrahám a problémům, kterým tým může čelit, pokud se rozhodne svůj proces vývoje softwaru řídit kteroukoliv z agilních metodik. Praktická část je věnována popisu projektu v reálné společnosti. Popisuje kroky, které musel tým učinit pro aplikaci metodiky Scrum a zavedení agilních procesů. V závěru jsou vyhodnoceny konkrétní přínosy této metody při řešení konkrétního problému .

Abstrakt v anglickém jazyce

This diploma thesis deals with agile frameworks used for developing various software projects. Traditional and agile methodologies are describes in detail and the differences between those two approaches are mentioned as well. The theoretical chapters cover mainly the usage of agile framework Scrum, which is the most popular agile framework used today in software development. It describes the challenges that a team can face while implementing Scrum framework; as well as various roles, meetings and practices recommended by authors of the Scrum framework. Practical part of this diploma thesis describes a real case study, which was performed by the author of this thesis several years ago in a real company.

Klíčová slova

agilní metody, agilní myšlení, management vývoje softwaru, metodika, Scrum, Scrum Master, Software engineering, Vodopádový model, vývoj softwaru

Klíčová slova v anglickém jazyce

agile frameworks, agile mindset, framework, Scrum, Scrum Master, software development, software development management, software engineering, Waterfall method

Obsah

Předmluva

Téma agilních metodiky je mi velmi blízké. Vše začalo v roce 2007, kdy jsem plný nadějí z nastudovaných teoretických znalostí získaných v bakalářském studiu na vysoké škole nastoupil do svého prvního zaměstnání na pozici vývojáře. Na vlastní kůži jsem měl tu šanci zažít ten pocit, když jsem musel vyvíjet kód, o kterém jsem byl já i celý zbytek týmu na 100 % přesvědčen, že se nikdy nedostane k zákazníkovi, který si projekt objednal. Již tehdy bylo jasné, že se kód píše jen z důvodu chybného návrhu v přípravné fázi a bylo dohodnuté, že se příští měsíc tato funkcionalita bude opravovat. Na moji otázku, proč to neudělat rovnou správně, jsem byl projektovým manažerem zpražen, že takto je to naplánované a kdyby to tým nedodal, tak by byl problém s vedením společnosti, a co si vůbec dovoluji z pozice programátora komentovat, co je správně a co špatně. Po šesti měsících jsem ve firmě skončil. V té době mi můj bývalý spolužák nabídl práci ve své firmě. Byla to malá firma, kde každý znal každého a vývoj byl orientován hlavně na to, co chce zákazník. Zákazník byl velmi entuziastický a u vývojového týmu trávil každý den několik hodin a konzultoval, co by pro jeho projekt bylo nejlepší. Programovat mě nikdy moc nebavilo, a tak jsem velmi brzy zjistil, že moje silné stránky jsou především ve vedení lidí. Stal se ze mě projektový manažer, ale nechtěl jsem opakovat chyby, kvůli kterým jsem v předchozím zaměstnání skončil. Na internetu jsem objevil knihu Agile Project Management with Scrum od Kena Schwabera, která popisuje vedení projektů pomocí metodiky Scrum a agilní myšlení obecně. Velmi ovlivněn touto knihou jsem se pokusil nasadit metodiku Scrum na současném projektu. Výsledky se dostavily ihned, produktivita práce se okamžitě zvýšila a zákazník byl s projektem velmi spokojen.

Na agilních projektech se od té doby pohybuji dodnes. Již 7 let působím jako agilní kouč, Scrum Master nebo projektový manažer v různých českých, zahraničních, malých i velkých společnostech. Za tuto dobu jsem byl součástí více než 30 týmů. Agilní metodiky a metodiku Scrum vyučuji ve školicím centru PC-DIR, jehož zákazník je například Microsoft. Tyto prakticky nabité znalosti jsem využil v teoretické i praktické části této diplomové práce.

Úvod

Cílem této diplomové práce je zmapovat postupy používané při vývoji softwaru. V teoretické části této práce jsem se rozhodl věnovat popisu jednotlivých metodik, jejich základnímu dělení a popisu silných a slabých stránek vybraných metodik. Největší prostor je v teoretické části práce věnován dvěma hlavním používaným metodikám - rigorózní metodice RUP (Rational Unified Process) A agilní metodice Scrum. Scrum je dnes velmi populární. Je to nejpoužívanější metodika pro vedení softwarových projektů. V této práci také srovnávám výhody a nevýhody jednotlivých metodik a důvody, proč jsou agilní metodiky tak populární a co zapříčinilo jejich vznik. Praktická část této práce je věnována konkrétní implementaci metodiky Scrum na reálném projektu softwarové firmy. Po dobu více než půl roku jsem vedl, který se pokusil zavést metodiku Scrum pro řízení vývoje svého produktu. V praktické části této práce jsou tedy popsány konkrétní problémy a výzvy, které tým musel překonat, aby mohl metodiku Scrum používat. Agilní metodiky tvrdí, že tým, který je používá, je šťastnější tým. Pomocí měření spokojenosti zaměstnanců po dobu implementace metodiky Scrum jsem se pokusil toto tvrzení dokázat. Další z proklamovaných výhod agilních metodik je kvalita produktu. Tento parametr jsem se rozhodl zkoumat pomocí měření množství hlášených defektů při používání produktu.

1 Metodiky vývoje softwaru

Vzpomeňme si na doby, kdy byl počítač velký jako průměrný obývací pokoj a na jeho obsluhu bylo potřeba půl tuctu vyškolených a vysokoškolsky vzdělaných pracovníků v bílých pláštích. Již tenkrát bylo nutné tyto lidi nějakým způsobem koordinovat a řídit. Čím více se četnost a dostupnost počítačů zvyšovala, tím také rostly nároky na komplexnost softwaru, který na těchto počítačích běžel. Toto nové odvětví, tedy vývoj softwaru, bylo co se managementu a vedení lidí týče neprobádaným terénem. Bylo zcela odlišné od vedení zaměstnanců v továrně či kanceláři a poptávka po softwarových projektech exponenciálně stoupala. Díky této nezkušenosti se vývoj softwaru potýkal s řadou problémů. Počínaje zpožďováním doby doručení, přes mnohanásobné překračování plánovaného rozpočtu, nízkou kvalitu a konče množstvím chyb v doručeném softwaru. Dospělo to až do takového stavu, že v roce 1968 na konferenci o softwarovém inženýrství sponzorované organizací NATO byl současný stav označen jako „softwarová krize“. (Report on a conference sponsored by the NATO Science Commitee, online, cit. 2015-04-20) Touto krizí bylo míněno množství projektů, které v té době trpěly jedním, či více problémy popsanými výše. Jednou z největších kritik byl nedostatek systematického přístupu k vývoji. Lidé, kteří byli zainteresováni ve vývoji softwaru, volali po nutné změně, která by vedla k zlepšení odstrašujících výsledků, jež byly na konferenci prezentovány. (Report on a conference sponsored by the NATO Science Commitee, online, cit. 2015-04-20)

- Více než polovina softwarových projektů byla objednavateli doručena, ale nikdy nebyla úspěšně nasazena,
- 30 % projektů bylo zaplaceno, ale nikdy nebylo úspěšně doručeno,
- 20 % projektů bylo vyvinuto, ale pro velké změnové požadavky bylo rozhodnuto o jejich ukončení bez úspěšného nasazení,
- 3 % projektů byla po velkých úpravách nasazena a použita,
- 2 % projektů byla použita a nasazena bez zásadní úpravy po jejich dodání.

Hlavní příčinou byl dle přednášejících řečníků na konferenci nedostatek systematičnosti v řízení těchto projektů. Tedy špatné vedení, zadávání úkolů a vyhodnocování jednotlivých milníků v projektu. Poprvé se objevuje pojem nedostatek softwarového inženýrství. Jako softwarové inženýrství můžeme rozumět disciplínu

zabývající se metodami a systémy používanými pro analýzu, vývoj, testování a nasazení softwarových projektů. Softwarové inženýrství tedy zkoumá a porovnává praktiky používané při vývoji softwaru. Mezi tyto praktiky můžeme zařadit životní cyklus vývoje softwaru, objektové programování, modelovací nástroje nebo modularitu a koncept znovupoužití. (Report on a konference sponsored by the NATO Science Commitee, online, cit. 2015-04-20)

1.1 Frameworky metodik softwarového vývoje

V reakci na konferenci NATO spatřilo světlo světa hned několik metodik vývoje softwaru. Metodika je sada doporučení a pracovních postupů, jak „správně“ navrhovat, plánovat a řídit procesy, které jsou třeba pro vývoj softwarových řešení. (Akademický slovník cizích slov, 1997, strana 493) Je také důležité si uvědomit, že pro každý projekt je vhodná jiná metodika, popřípadě i její úprava. Většina metodik totiž vychází ze základního předpokladu, že metodika je upravována na základě potřeb konkrétní společnosti nebo projektu, na kterém je nasazena. I rozsah a úroveň detailu se u jednotlivých metodik může lišit. Některé metodiky určují každodenní práci všech rolí zúčastněných na projektu a některé pouze určují mantinely, ve kterých by se měl projekt v dané fázi pohybovat. Frameworky metodik typicky dělíme takto:

- Tradiční přístup. Někdy také označovaný rigorózní, pro svoji vysokou míru formalizace a důraz na sběr požadavků a business analýzu.
- Agilní přístup. Flexibilní přístup, kde je na prvním místě zákazník a jeho požadavky na kvalitní a funkční software.

1.1.1 Životní cyklus vývoje software

Pod pojmem životní cyklus se rozumí definice jednotlivých stádií, kterými software při vývoji musí projít. Rozdělujeme tři základní typy životních cyklů vývoje softwaru:

- lineární
- iterativní
- a jejich kombinaci.

Slovo lineární si můžeme vykládat jako „postup z jedné fáze do druhé za pomoci série jasných (přímých) kroků“. (Akademický slovník cizích slov, 1997, strana 461) Tato

definice přesně vystihuje tento životní cyklus vývoje softwaru. Tedy vše je předem naplánováno a jasně navrženo a plán se dodržuje za všech okolností. Často se tyto lineární životní cykly nazývají „plan-driven“, tedy řízeny plánem, protože ve většině případů vyžadují seznam požadavků (na software), které jsou známy již na začátku projektu. Tyto požadavky by měly být jasné, zřetelné a relativně stabilní. (Williams, 2007, strana 1)Jako hlavního představitele této kategorie můžeme uvést vodopádový model, kterému bude věnována vlastní kapitola této práce.

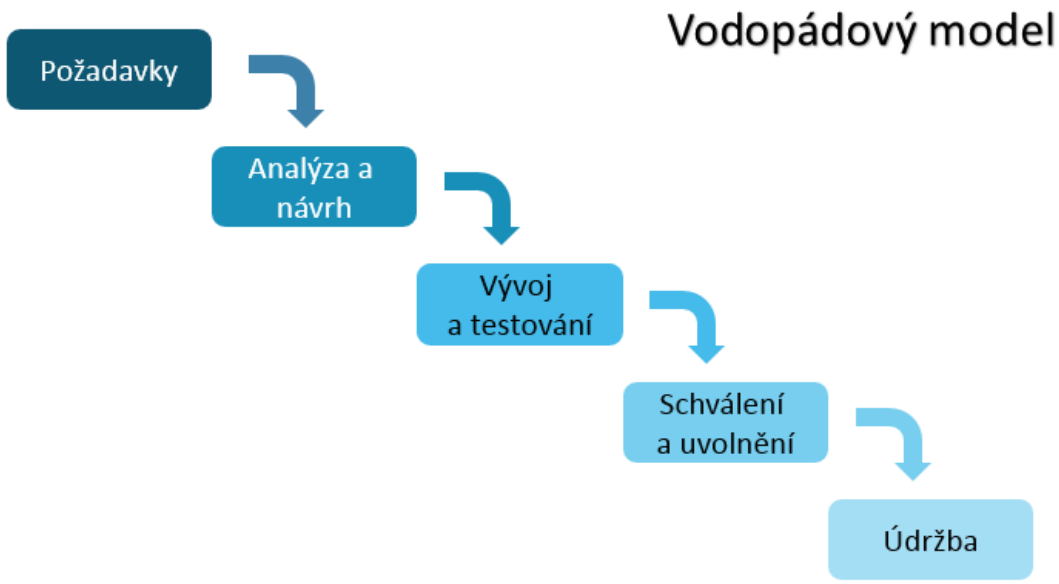
Iterativní životní cyklus vývoje softwaru je v tomto ohledu naprosto opačný. Tedy nevyžaduje, jako u lineárních životních cyklů, rozsáhlé plánování celého projektu již při jeho startu. Naopak toto nedoporučuje, protože v realitě se požadavky každý den mohou měnit a raději doporučuje v jednotlivých iteracích plán neustále revidovat a reagovat na změny a podněty od všech zainteresovaných stran projektu (tedy jak zadavatele, tak i vykonavatele projektu). V iterativních životních cyklech vývoje softwaru je práce často rozdělena do malých, ale funkčních celků, které jsou nazývány inkrementy. Tyto inkrementy pak jako celek tvoří výsledný produkt, pokud jsou správně poskládány dohromady. (Cocburn, 2008, strana 1-2) Iterativním modelům se bude věnovat velká část této práce, nejčastěji se používají metodiky Scrum nebo Extrémní programování. Dle výzkumu Larmana (2003) zlepšuje iterativní a inkrementální přístup proces vývoje a zvyšuje kvalitu výsledného produktu, protože díky častým revizím většinou výsledek z velké části koreluje s představou zadavatele projektu.

V praxi se můžeme setkat i s kombinací obou přístupů, tedy prolnutí iterativního životního cyklu s lineárním. Hlavními zástupci jsou modely Spirála, Rapid application development (RAD) a Extrémní programování. (Cocburn, 2008, strana 1) Dnes se u softwarových společností častěji setkáváme s iterativním přístupem nežli s lineárním, můžeme ho tedy označit za populárnější. Hlavními výhodami je velká míra flexibility v projektu a plánování. Představy a potřeby zákazníka se mohou znenadání měnit a dodržování stanoveného plánu, jak doporučuje vodopádový model, může vést k frustraci zákazníka, který vidí implementaci projektu, který díky nepředvídatelným okolnostem na trhu, na kterém působí, již třeba dávno nepotřebuje. (Williams, 2007, strana 2)

1.1.2 Vodopád

Základním konceptem vodopádového modelu je rozfázování velkého projektu na dílčí činnosti, které na sebe logicky navazují. Jako nutný vstup tyto činnosti používají výsledky fáze minulé. (Simsa Johnson, 2011, strana 14) Zde je tedy jasný původ názvu tohoto modelu, tedy vodopád, který čerpá ze zdrojů vytvořených procesem předešlým. V oblasti vývoje softwaru byl v roce 1970 představen tento model dr. Winstonem Roycem, který publikoval svoji práci *Managing the Development of Large Software Systems*. V této publikaci Royce (1970, strana 2-5)rozdělil velké softwarové projekty na pět základních fází:

- sběr požadavků,
- analýza a návrh,
- vývoj a testování,
- schválení a uvolnění,
- provoz a údržba..



Obrázek 1- Vodopádový model (zdroj: https://managementmania.com/cs/vodopadovy-model-waterfall-model, online, 2015)

Na první pohled se tento model zdá jednoduchý a účinný. Hlavně z důvodu jeho jednoduchosti je velmi oblíbený mezi manažery. Myšlenka vodopádu není žádnou novinkou a sám Royce přiznává inspiraci v jiných odvětvích. Tento model funguje velmi dobře například v architektuře. Každá stavba také potřebuje analýzu, například

studii proveditelnosti nebo geologický rozbor a podobné úkony, které musí být provedeny, než se může začít stavět. Každá stavba také vyžaduje architektonický plán. Tedy návrh, dle kterého bude stavba postavena. Až po schválení návrhu a opatření potřebných povolení může být zahájena samotná stavba, v korelaci s vodopádovým modelem tedy samotná implementace. Po dokončení stavby je potřeba objekt řádně verifikovat, zkolaudovat, prověřit, zda splňuje všechny stavební, požární a další vyžadované požadavky. Teprve potom se může stavba začít používat. Tento velmi zjednodušený popis jasně dokládá Roycovu inspiraci v jiných odvětví. V oblasti vývoje softwaru je tento model dnes již považován za překonaný. Ve srovnání se stavebnictvím, kde lze tento model v jistých obměnách používat dodnes, spočívá při vývoji softwaru rozdíl především v potřebě velkého množství dokumentace a plánování v rané fázi projektu. (Clinton, 2010) Je tedy potřeba vyspecifikovat všechny požadavky na software dopředu. Trendy ve vývoji softwaru se pohybují daleko rychleji než trendy ve stavebnictví. Pokud začnete stavět rodinný dům, který za rok dostavíte, je velmi pravděpodobné, že i bez velkých odchylek od plánu bude dokončená stavba stále moderní a funkční. U softwaru je tato situace jiná. Je velmi pravděpodobné, že za rok se nároky na funkčnost mohou změnit a díky vodopádovému přístupu, tedy že implementujete software dle plánu a ne dle potřeb zákazníka, bude tento software po nasazení nepoužitelný. Model postrádá flexibilitu a možnost reagovat na vzniklé události. Model také nedovoluje návraty k předchozím fázím projektu, a tak pokud vznikne chyba na začátku projektu, například v analýze, je velmi nákladné chybu v pozdějších fázích opravit.

1.1.3 Prototypování

Prototypování je přímou reakcí na vodopádový model, konkrétně na jeho nedostatečnou flexibilitu. V rámci prototypování dochází k vytváření nekompletních softwarových projektů nebo aplikací, které jsou následně prezentovány zákazníkovi. (Gibson, 2014, strana 25) V této aplikaci většinou funguje pouze několik konkrétních scénářů. Softwaroví vývojáři a vedoucí projektu díky těmto prototypům mohou získávat cennou zpětnou vazbu od zákazníka již ve velmi brzkém stádiu projektu. U vodopádového modelu se s reakcí zákazníka setkáváme až velmi pozdě, kdy je celý projekt „hotový“ a nasazený. Díky této zpětné vazbě si může zákazník jednoduše ověřit, že se projekt vyvíjí směrem, kterým očekává, a že funkcionalita v konečné fázi bude přinášet přidanou hodnotu (Arnowitz, 2007, strana 232). Jako příklad lze uvést

prototyp webového obchodu. Vytvoříme prototyp, kde jsou produkty napevno umístěné elementy na webové stránce bez možnosti jejich editace. Funguje pouze přidání produktu do košíku s tím, že na obrazovce košíku je vypočítána reálná cena za obsah košíku. Samotné vytvoření a dokončení objednávky je opět nefunkční nebo statické. S tímto prototypem si můžeme u zákazníka (vlastníka webového obchodu) ověřit, že je chování aplikace při vložení produktu do košíku správné a že je správný i proces výpočtu konečné ceny. Prototypování také pomáhá udržovat časový harmonogram a projektové milníky. Díky funkčnímu prototypu totiž můžeme poměrně snadno zjistit, jaká část projektu schází, pokud výsledný prototyp porovnáme se specifikací projektu. Tuto chybějící část projektu můžeme porovnat s již hotovou a časem, který jsme na jeho vytvoření potřebovali. Rozlišujeme několik druhů prototypování. Nejpopulárnějšími jsou

- throwaway prototyping,
- evolutionary prototyping,
- incremental prototyping,
- extreme prototyping.

Throwaway prototyping (z anglického „throwaway" – zahodit) vychází z konceptu rychlého vytvoření prototypu, který slouží k ověření zadání se zákazníkem. Většinou se jedná o prototyp aplikace, ve které funguje pouze základní průchod daného scénáře. Je kladen velký důraz na rychlost vytvoření prototypu, který je často velmi jednoduchý. Má za úkol pouze demonstrovat směr, kterým se opravdový vývoj bude ubírat. Tento prototyp je po akceptaci zákazníkem zahozen a nahrazen plně funkčním řešením. Pokud má zákazník připomínky k funkčnosti nebo návrhu prototypu, jsou požadavky upraveny dle jeho přání. Zákazníkům tento přístup často umožňuje uvědomit si, co vlastně chtějí. Mnohdy je pro zákazníka velmi složité předvídat, co vlastně chce, dokud software nevidí fungovat. Prototypování se toto snaží alespoň částečně funkčními nebo nefunkčními modely eliminovat. (Wiegers, 2003, strana 241) Často se s tímto typem prototypování setkáme ve spirálovém modelu, kde v analytické fázi je prototyp vytvořen, schválen, požadavky přepracovány a prototyp vyhozen. V další fázi cyklu je pak implementován v plné funkčnosti.

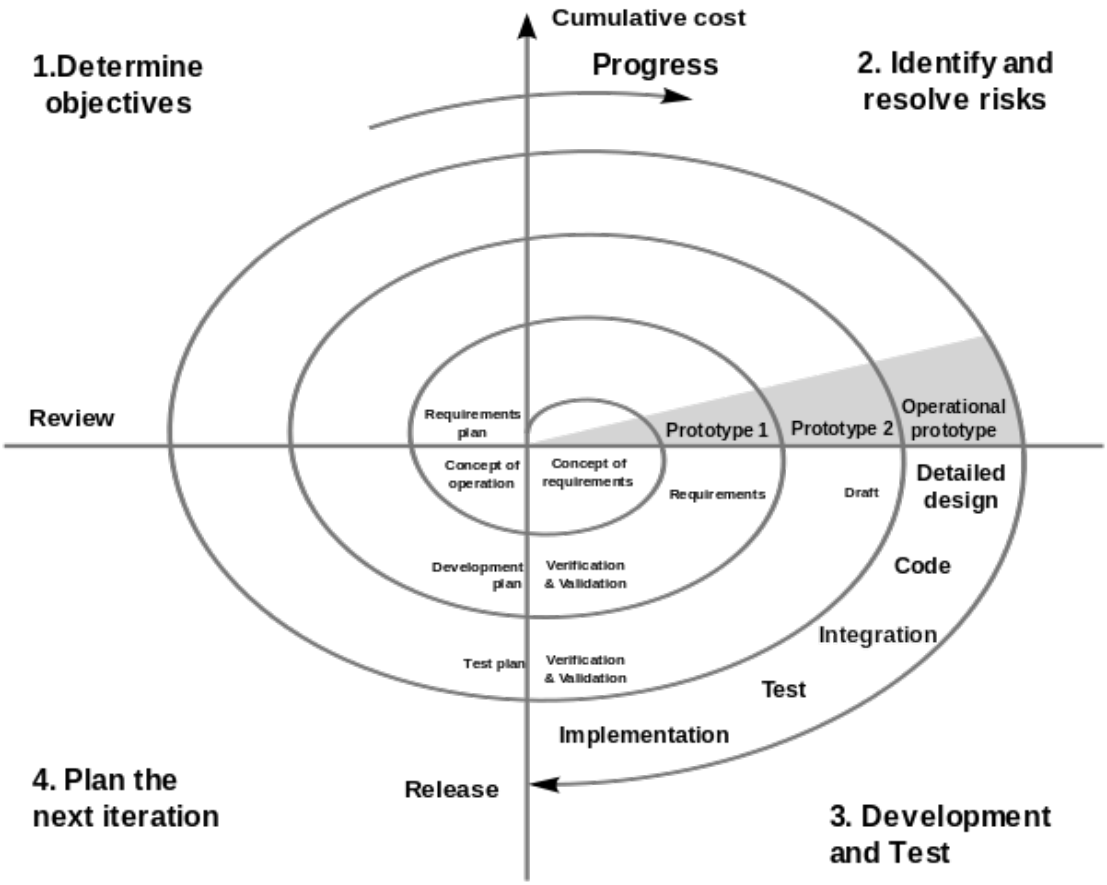
Evolutionary prototyping (z anglického „evolutionary" – evoluční) je koncept ve kterém se prototyp po dokončení nevyhazuje. Naopak prototyp je po konzultaci se

zákazníky přetvořen v konečné řešení. Tento prototyp je neustále udržován a jsou do něj přidávány další a další funkce které ve výsledku vytvoří koncový produkt. Hlavní výhodou je implementace funkčnosti tehdy, kdy je třeba. Tedy na začátku může být funkčnost vyspecifikována jen na velmi konceptuální úrovni a v průběhu projektu doladěna. Například již zmíněný internetový obchod počítá s tím, že v systému bude i zaznamenáno, který pracovník objednávku zpracoval, zabalil a odeslal. Tato funkčnost není detailně popsána ve specifikaci a bude naprogramována později. Nicméně prototyp již s touto funkcionalitou počítá a ukazuje fotku zaměstnance, který naposledy modifikoval objednávku na stránce objednávky.

Incremental prototyping je koncept velmi podobný evolutionary prototyping, ale finální produkt je vytvořen z několika spojených prototypů. Výhodou může být snadnější udržovatelnost než v případě jednoho obecného prototypu a také snazší paralelní zapojení několika vývojářů nebo týmů v jednom projektu. Samozřejmě nevýhodou mohou být potíže a případné konflikty při spojování několika prototypů do jednoho výsledného produktu.

Extreme prototyping je koncept používaný především při vývoji webových aplikací. Tento přístup je rozdělen na tři fáze. První, ve které po specifikaci projektu od zákazníka vývojáři vytvoří nefunkční prototypy jednotlivých obrazovek aplikace. Tento model je prezentován zákazníkovi a má především za úkol ověřit použivatelnost webové aplikace a funkční design. V následující fázi vývojáři tyto nefunkční stránky napojí na nefungující (mocking) aplikační logiku, která je následně ve třetí fázi zprovozněna. Tento přístup pomáhá zákazníkovi předcházet takzvanému „wow“ efektu, kde si zákazník uvědomuje, co vlastně potřeboval, až když vidí celé uživatelské prostředí aplikace najednou. Toto uživatelské prostředí je díky konceptu extreme prototyping dostupné hned v první fázi projektu a zákazník tedy může ověřit správnost zadání, popřípadě zadání upravit ve fázi, kdy se ještě nestrávil čas na jeho implementaci. (Arnowitz, 2007)

1.1.4 Spirála



Obrázek 2 - Model Spirála (zdroj: Spiral model. Wikipedia, online)

Model spirála je životní cyklus vývoje softwaru řízený na základě analýzy rizika. Tento model, na rozdíl například od vodopádu, byl určen především k vedení a řízení softwarových projektů. (Boehm, 2014) Na základě vyhodnocení rizik v jednotlivých fázích projektu doporučí spirálový model týmu zařadit do plánu jeden nebo více procesních modelů, například inkrementální vývoj, vodopád nebo prototypování. Jedná se tedy o takový mix různých vývojových metodik, zařazených v určitém pořadí, závisejících na konkrétním vyhodnocení rizik v projektu.

Poprvé světu představil svůj model spirály americký softwarový vývojář Barry Boehm ve své publikaci „A Spiral Model of Software Development and Enhancement“ v roce 1986. Boehm popsal spirálový model jako generátor procesního modelu. Model je rozdělen do jednotlivých cyklů, kterých může projekt obsahovat neomezeně mnoho. Každý cyklus pracuje s inkrementem vytvořeným v minulém

cyklu projektu. Tento inkrementální vývoj snižuje riziko projektu. Každý inkrement je konzultován se zákazníkem. Na těchto schůzkách je možno ovlivnit směr celého projektu. Každý cyklus je rozdělen na čtyři základní fáze (kvadranty) (Boehm, 2014, strana 8-15):

- stanovení cílů – diskuze alternativ a délky cyklu,
- vyhodnocení a řešení rizik – volba přístupu, který cyklus bude použit,
- vývoj a testování,
- plánování příštího cyklu – vyhodnocení dodaného inkrementu.

Spirálový model je vítanou alternativou k vodopádovému modelu. A především díky cyklickému přístupu se setkává s velmi dobrým ohlasem. Během let byl tento model často použit jako inspirace pro agilní metodiky vývoje softwaru, například SCRUM. Uplatnil se zejména iterativní a inkrementální přístup. Boehm upozorňuje na časté, mylně interpretované, chyby výkladu jeho modelu.

- Spirálový model je vnímán jako několik po sobě jdoucích vodopádových modelů. Zde upozorňuje na to, že spirálový model je navržen tak, že na začátku každého cyklu si vývojový tým může vybrat metodiku, která mu vyhovuje nejvíce.
- Všechny projektové aktivity musí probíhat sériově. Tedy, že spirálový model nedovoluje paralelní práci na projektu. Naopak Boehm doporučuje aby co nejvíce aktivit, pokud to je možné, probíhalo paralelně.
- Všechny projektové aktivity, které jsou zobrazeny na diagramu modelu, musí proběhnout. Model dovoluje některé procesy nebo fáze v jednotlivých stádiích vynechat. (Boehm, 2014, strana 17-22)

1.2 Rational Unified Process

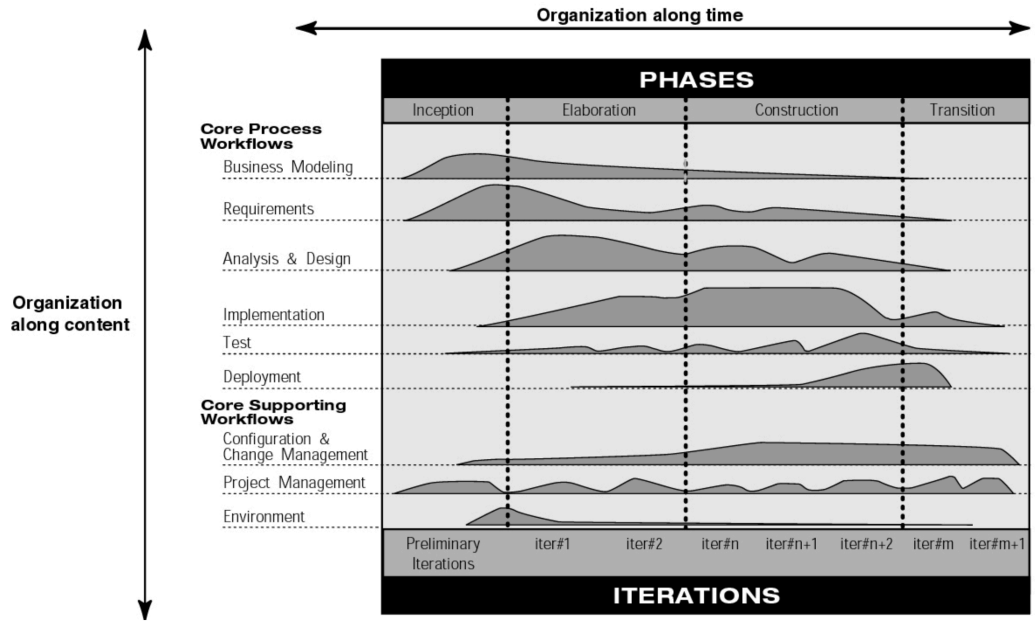
Jedna z nejznámějších procesních metodik je Rational Unified Process (RUP). Jedná se o iterativní metodiku vytvořenou v roce 2003 firmou Rational Software, divizí známé firmy IBM. RUP metodika je vlastně dokument, ve kterém jsou navzájem odkazované jednotlivé vzorové artefakty a detailní popisy pro různé druhy aktivit, které může softwarový projekt obsahovat. Tento dokument je velmi obsáhlý, jelikož obsahuje velké množství těchto artefaktů. Pomocí nástroje IBM Rational Model

Composer (RMC) je možné si sestavit proces podle potřeb jednotlivých projektů, a tak ho zjednodušit a sestavit ho takzvaně na míru. Výsledný ořezaný proces je i přes modifikace velmi obsáhlý a tedy pro malé projekty stále velmi komplexní. Právě velká komplexnost je hlavní výhodou a současně nevýhodou této metodiky. Metodika je vhodná pro komplexní projekty, jako jsou například projekty ve zdravotnictví nebo aviatice, kde jediná chyba v produktu může ovlivnit životy mnoha jedinců. Naopak se nehodí pro malé projekty, například jednoduchých webových obchodů. Metodika totiž určuje každý krok v projektu, od sběru požadavků, analýzy potřeb až po nasazení, školení a údržbu. RUP definuje šest základních pravidel pro vývoj softwaru, se kterými můžeme souhlasit i přesto, že například RUP pro jeho komplexnost pro daný projekt nepoužijeme:

- iterativní vývoj,
- aktivní správa požadavků,
- komponentová architektura,
- vizuální modelování,
- ověřování kvality softwaru,
- řízení změn.

Model Rational Unified Process může být popsán ve dvou dimenzích:

- Horizontální – reprezentuje čas a ukazuje dynamiku procesu. Na horizontální ose jsou znázorněny cykly, jednotlivé fáze, iterace a milníky projektu.
- Vertikální – reprezentuje jednotlivé procesy. Popisuje potřebné aktivity, artefakty, pracovníky a pracovní postupy. (Kruchten, 2010, strana 25-31).



Obrázek 3– Rational Unified Process (zdroj: Rational Unified Process Best Practices for Software Development Teams, online, 2001, strana 3)

Obvykle je projekt, který je řízen pomocí této metodiky, rozdělen do několika etap. Každý projekt musí mít alespoň jednu etapu. Můžeme se setkat i s projekty, které mají etap nekonečně mnoho. Jedná se převážně o projekty, které jsou vyvíjeny kontinuálně. Každá etapa je procesem rozdělena na čtyři po sobě jdoucí cykly (Rational Unified Process Best Practices for Software Development Teams, 2001, strana 3):

- zahájení (Inceptionphase),
- příprava (Elaborationphase),
- konstrukce (Constructionphase) a
- předání (Transitionphase).

Každá z těchto fází je zakončena jasně definovaným milníkem v projektu, tedy přesně určeným bodem na časové ose projektu. Ke každému milníku se váže sada klíčových úkolů, které musí být splněny, aby projekt mohl pokračovat do další fáze projektu (Boehm, 2014, strana 25).

1.2.1 Fáze zahájení –Inceptionphase

Během zahajovací fáze na projektu je definováno obchodní odůvodnění projektu (business case), tedy zdokumentování opodstatněnosti realizace projektu založené na odhadovaných nákladech ve srovnání s plánovanými přínosy za existence určitých rizik. (Prince 2, 2015) Dále je v zahajovací fázi projektu odhadnut rozsah projektu. Tyto informace můžeme definovat pouze za předpokladu, že jsme schopni definovat všechny externí vstupy, se kterými bude systém pracovat a způsoby, jakými budou tyto interakce probíhat. Tato definice zahrnuje především určení všech případů použití (use case) a detailní specifikaci těch nejdůležitějších. Obchodní odůvodnění projektu obsahuje stanovení podmínek úspěšnosti projektu, ale i zhodnocení všech rizik, které mohou během projektu, jeho nasazení a údržbě nastat. (Royce, 1970). Dále jsou odhadnuty zdroje, které jsou na realizaci projektu potřeba, a jsou navržena i data jednotlivých milníků v projektu. Výsledkem zahajovací fáze by měly být následující artefakty (Rational Unified Process, Best Practices for Software Development Teams, 2001, strana 4):

- **Vize** – Artefakt který obsahuje představu o budoucnosti projektu a jeho použití. Tedy definice požadavků, klíčových funkcí a omezení projektu.
- **Model případu použití** – Artefakt, který popisuje interakci jednotlivých uživatelů systému; definice funkcionality systému. V této fázi jsou popsány kritické případy užití, zhruba 20% systému.
- **Projektový glosář** – Artefakt který obsahuje definici jednotlivých pojmů v projektu. Může mít formu doménového modelu.
- **Obchodní odůvodnění projektu** – Artefakt který popisuje opodstatnění realizace projektu. Obsahuje kritéria úspěchu, analýzu konkurence, obchodního prostředí a finanční predikci projektu.
- **Analýza rizika** – Artefakt, který analyzuje potencionální hrozby projektu a identifikuje, jak jim předcházet.
- **Projektový plán** –Artefakt popisující důležitá předpokládaná data jednotlivých milníků, fází a iterací.
- **Obchodní model** – Artefakt popisující strategii, jak bude výsledný produkt projektu zapojen na trhu, popřípadě jak bude generovat zisk.
- **Jeden nebo více prototypů.**

Zahajovací fáze je zakončena prvním velkým milníkem projektu, tzv. milníkem cílů životního cyklu (Lifecycle Objectives Milestone), který hodnotí celou zahajovací fázi podle několika kritérií:

- Shoda zainteresovaných osob (stakeholders) na definici rozsahu, odhadované ceny a délky projektu.
- Porozumění požadavkům projektu jako důkaz proveditelnosti základních scénářů použití.
- Důvěryhodnost předpokládaného rozpočtu, naplánovaných priorit, hodnocení rizik a vývojového procesu.
- Vyhodnocení výsledků všech prototypů použitelné architektury projektu.
- Skutečný rozsah výdajů na zahajovací fázi v porovnání s plánovanými.

Díky posouzení těchto kritérií mohou zainteresované osoby rozhodnout o nutnosti přepracování projektu nebo o jeho úplném zrušení. Pokud je vše pozitivně vyhodnoceno, může projekt postoupit do fáze přípravy.

1.2.2 Fáze přípravy – Elaboration phase

Hlavním cílem fáze přípravy je analyzovat problémovou doménu projektu, vymodelovat projektovou architekturu, vytvořit projektový plán a vyloučit největší rizika, s nimiž se můžeme během zpracování projektu setkat. Toto je velmi náročná část projektu, která vyžaduje jeho velmi dobrou znalost. Kvalitní rozhodnutí o architektuře systému podmiňuje dostatek informací o projektu a kvalitně zpracované funkční a nefunkční požadavky na systém, jako je například požadovaný výkon nebo dostupnost. Ačkoliv se fáze přípravy může zdát jako nejméně důležitá, opak je pravdou. Architektura, která se nadefinuje v této fázi, je pak používaná po celý projekt. I přesto, že metodika RUP je metodikou iterativní, tedy eliminuje rizika neustálým ujišťováním se, že je projekt na správné cestě, je dobře nastavený architektonický plán základním stavebním kamenem v této metodice. Velká pozornost by měla být v této fázi věnována zajištění projektů, kdy je rozpočet a datum dodání většinou smluvně ukotveno a jejich porušení je nepřípustné. V této fázi již vstupují do plánu iterace. Pomocí různých druhů prototypování je v iterativním procesu vytvořen hlavní architektonický prototyp, který eliminuje co nejvíce budoucích rizik a odpovídá na většinu otázek ohledně architektury.(Kruchten a Kroll, 2003, strana 57). V této fázi již také může vznikat uživatelský manuál pro koncové

uživatele systému. Hlavním výstupem fáze přípravy jsou následující artefakty (Rational Unified Process, Best Practices for Software Development Teams, 2001, strana 5):

- **Model případu použití** – Artefakt, který popisuje interakci jednotlivých uživatelů systému, v této fázi je již skoro kompletní (80% systému), všechny scénáře byly identifikovány, stejně jako všichni uživatelé systému.
- **Analýza nefunkčních požadavků**–Artefakt, který popisuje nefunkční požadavky, tedy požadavky které nejsou pevně spjaté s žádným případem použití, například dostupnost nebo výkon systému či přístup přes mobilní zařízení.
- **Popis softwarové architektury** – Artefakt, který jak již název napovídá, popisuje architektonický plán pro vývoj projektu v dalších fázích.
- **Spustitelný architektonický prototyp** – Prototyp, na kterém se dá demonstrovat architektura z artefaktu Popis softwarové architektury. Tento prototyp by měl eliminovat co nejvíce rizik špatného návrhu v dalších fázích projektu.
- **Revidovaný seznam rizik a revidované obchodní odůvodnění projektu**
- **Vývojový plán** – Artefakt popisující jednotlivé iterace konstrukční fáze a jejich předpokládané výsledky. Tento artefakt také obsahuje hodnotící kritéria, kvantitativní a kvalitativní, pro jednotlivé iterace.
- **Specifikace konkrétní metodiky použité pro vývoj systému** – Artefakt popisující konkrétní metodiku pro samotný vývoj systému v konstrukční fázi.
- **Předběžný uživatelský manuál** – Artefakt dokumentující interakce koncových uživatelů se systémem. Díky dobře vyspecifikovaným požadavkům a funkčnímu prototypu by se již klíčové scénáře použití neměly měnit.

Na konci fáze přípravy nastává Milník architektury životního cyklu (Lifecycle Architecture Milestone). Hlavními hodnotícími elementy tohoto milníku jsou následující kritéria(Kruchten a Kroll, 2003, strana 43-45):

- Je vize projektu stabilní a realizovatelná?
- Je architektura stabilní a realizovatelná?

- Dokazuje funkční prototyp architektury to, že hlavní rizika projektu byla identifikována a eliminována?
- Důvěřují všechny zainteresované osoby (stakeholders) tomu, že současná vize může být realizována za pomoci navržené architektury a projektového/vývojového plánu?
- Je vývojový plán projektu dostatečně detailní a přesný? Jsou součástí tohoto plánu důvěryhodné odhady?
- Jak vychází porovnání současných nákladů na zdroje s plánovanými. Je výsledek tohoto porovnání akceptovatelný? Nepřevyšují výrazně reálné náklady ty plánované?

V této fázi mají zainteresované osoby možnost projekt zcela ukončit nebo vrátit k přepracování na začátek fáze přípravy nebo i zcela na začátek projektu.

1.2.3 Fáze konstrukce – Construction phase

V průběhu fáze konstrukce jsou všechny zbývající komponenty a funkce systému vyvinuty a integrovány do finálního produktu. Jsou také důkladně otestovány všechny případy použití. Fáze konstrukce je již výrobní proces samotný. Můžeme ho přirovnat k výrobnímu procesu v tovární hale. Tedy vývojáři, dělníci, dle plánů architektů vyvíjejí, montují, výsledný produkt. Právě toto je „kámen úrazu“, který je často této metodice z pohledu vývojářů vytýkán. V této fázi je již omezena jakákoliv vlastní iniciativa a kreativita. Vše by mělo být uděláno tak, jak bylo naplánováno. Díky iteracím se mohou, na rozdíl od vodopádového modelu, odvrátit krizové situace, nicméně každá odchylka od plánu má za důsledek prodloužení nebo prodražení projektu. Důraz je tedy kladen na dodržování vývojového plánu, který byl vytvořen v předešlých fázích projektu. Cílem této fáze je doručit nastavitelný výsledný systém, který do co největší míry odpovídá specifikovanému zadání (Rational Unified Process, Best Practices for Software Development Teams, 2001, strana 6).

Tento proces samozřejmě může být urychlen spuštěním sekvenčních iterací několika týmů naráz. Vzhledem k tomu, že za pomoci RUP jsou realizovány především větší projekty, tak se i v praxi často děje. Samozřejmě je jasné, že čím více paralelních iterací několika nezávislých týmů bude spuštěno, tím je složitější uhlídat celkovou orientaci projektu. To je možné zajistit jen díky kvalitní architektuře a rozsáhlé

dokumentaci (projektový a vývojový plán). Projektoví manažeři musí iterativně vyhodnocovat stav projektu a čerpání zdrojů.(Rational Unified Process, Best Practices for Software Development Teams, 2001, strana 7). V této fázi se totiž velmi jednoduše může stát, že je překročen finanční nebo časový rozpočet projektu.

Výsledkem fáze konstrukce je softwarový systém, který je možné nasadit a předat k používání koncovým uživatelům. Minimální sada požadavků na fázi konstrukce je tato:

- Dodání integrovaného softwarového produktu dle specifikace a na platformách, které byly nadefinovány se zainteresovanými osobami.
- Uživatelský manuál a možnost školení používání systému.
- Popis současného stavu, tedy popis případných závad, nedostatků a zpoždění projektu.

Tato fáze je zakončena třetím projektovým milníkem. Milníkem počáteční operační schopnosti (Initial Operational Capability Milestone). V této fázi projektu je potřeba posoudit, zda je produkt připraven pro nasazení v reálném provozu bez velkých chyb. Často je výsledek této fáze označován jako „beta“ verze, která by již měla být otestována oddělením pro ověřování kvality a nyní bude otestována reálnými uživateli systému. V této fázi jsou hodnotícími kritérii následující otázky (Rational Unified Process, Best Practices for Software Development Teams, 2001, strana 6):

- Je tato verze produktu dostatečně stabilní a použitelná, aby mohla být nasazena pro opravdové uživatele?
- Jsou všechny zainteresované osoby připraveny na tento posun?
- Odpovídá porovnání současných nákladů na zdroje plánovaným? Je výsledek tohoto porovnání akceptovatelný? Nepřevyšují výrazně reálné náklady ty plánované?

Pokud se zainteresované osoby rozhodnou že výsledky této fáze nejsou uspokojující, mohou se rozhodnout ukončit celou etapu a vrátit se přímo do fáze zahájení nové etapy a vynechat fázi předání.

1.2.4 Fáze předání –Transitionphase

Hlavním úkolem fáze předání je nasazení vytvořeného systému pro používání opravdovými uživateli. Toto „reálné“ testování systému s sebou většinou přináší odhalení chyb, na které se při vývoji nepřišlo nebo nebyly považovány za vážné. Proto je testování reálnými uživateli tak důležité. Tato fáze může nastat pouze tehdy, pokud je produkt dostatečně vyspělý a je připraven na tuto fázi. Pokud tato fáze nastane moc brzy, nedostaví se požadované výsledky, ale naopak nastane mezi reálnými uživateli, testery, frustrace z nefunkčního systému. Tato fáze může být triviálně jednoduchá a krátká, stejně tak může tato fáze trvat měsíce. Vše záleží na komplexnosti vyvíjeného systému. Pokud je nasazována jednoduchá webová aplikace, vše by mělo být jednoduché. Pokud se jedná například o aplikaci pro řízení letového provozu, je tato fáze velmi komplexní. Důležité je ověřit, že se výsledný produkt shoduje s vizí zadavatele a že uživatelé jsou schopni systém používat. Ať už za pomoci tréninku a školení, nebo jednoduše po pročtení uživatelské dokumentace (Kruchten a Kroll, 2003, strana 57).

Fáze předání je zakončena čtvrtým projektovým milníkem, a to vlastním vydáním produktu (Product Release Milestone). Tato etapa je poslední fází projektu a je na zainteresovaných osobách, aby se rozhodly, zda budou pokračovat s další etapou nebo tato etapa byla poslední. Hlavními hodnotícími kritérii jsou odpovědi na otázky:

- Je uživatel systému uspokojen?
- Vychází porovnání současných nákladů na zdroje s plánovanými. Je výsledek tohoto porovnání akceptovatelný? Nepřevyšují výrazně reálné náklady ty plánované?

1.2.5 Struktura procesu

Každá z výše popsaných fázích RUP může být dále rozdělena do iterací. Tyto iterace pracují s konceptem minimalizace rizika opakováním, a tím přibližováním se ke kýženému výsledku. Výsledkem iterace by měla být podmnožina kompletního produktu, až se v poslední iteraci stane výsledným produktem. Výhody iterativního přístupu jsou tyto (Kruchten a Kroll, 2003, strana 59):

- minimalizace rizik,
- lepší možnost reakce na změnu,

- znuvupoužitelnost,
- možnost získávání zkušeností a jejich aplikace během projektu (neopakování stejných chyb),
- lepší výsledná kvalita.

Pro znázornění aktivit a plánů v RUP je použito modelování jednotlivých procesů. Pro modelování je vyžadováno použití jazyka Unified Modeling Language. V těchto modelech můžeme rozlišit čtyři základní elementy (Rational Unified Process, Best Practices for Software Development Teams, 2001, strana 8):

- Pracovníky (workers) – odpověď na otázku „kdo?“.
- Aktivity (activities) – odpověď na otázku „jak?“.
- Artefakty (artefacts) – odpověď na otázku „co?“.
- Pracovní postupy (workflows) – odpověď na otázku „kdy?“.

Rational Unified Process rozděluje celý proces do devíti základních disciplín(šest základních a tři podpůrné). Tyto disciplíny seskupují pracovníky a jejich aktivity do logických celků. (Rational Unified Process, Best Practices for Software Development Teams, 2001, strana 8)

Základní disciplíny takzvané „technické“ (engineering):

- Tvorba modelu
- Správa požadavků
- Analýza a návrh
- Implementace
- Testování
- Nasazení

Doplňující disciplíny (supporting):

- Konfigurace a změny
- Řízení projektu
- Správa prostředí

1.3 Procesní meta modely

Metodiky popsané v předešlých kapitolách většinou odpovídají na otázky kdo, kdy, jak a co. Existují také procesní modely, které jsou určeny pro porovnání, vyhodnocování a zlepšování specifických oblastí procesů ve společnosti. Mezi nejvýznamnější zástupce patří:

- **ISO/IEC 12207** – mezinárodně uznávaná metoda pro porovnání výběru, implementace a monitorování zvolené metodiky pro vývoj softwaru a jeho životního cyklu.
- **CCMI** -Hodnotící model, který je založen na srovnání naplánované metodiky oproti jejímu aktuálnímu používání. CCMI nijak negarantuje, že projekt nebo výsledný produkt bude úspěšný. Garantuje však to, že produkt byl vyvinut za pomoci předepsané metodiky. (Hoyle, 2009)
- **ISO 9000** – Popisuje a verifikuje formálně organizovaný proces za použití jakékoliv metodiky. Původně bylo ISO 9000 navrženo jako kontrola kvality dodržování procesů ve výrobních podnicích. ISO 9000 stejně jako CCMI negarantuje úspěch projektu, ale pouze to, že projekt postupoval dle formalizovaného procesu. (Hoyle, 2009)
- **ISO/IEC 15504** – Mezinárodně uznávaná sada rad a doporučení (anglicky označovaná jako framework) pro posuzování procesu vývoje softwaru. Tento procesní model cílí na jasnou definici a hodnotící model porovnávání jednotlivých přístupů k vývoji softwaru. Výsledky slouží především pro odhalování slabých míst a mají za následek zlepšování procesů při vývoji softwaru. (Hoyle, 2009)

2 Agilní metodiky vývoje softwaru

Dnes je skoro nemožné setkat se se softwarovou společností, která o sobě alespoň netvrdí, že je řízena agilními procesy a metodikami. Často je označení agilní metodika považováno za *buzzword,* tedy *termín který vyjadřuje jakýkoliv nový, radikální nebo revoluční termín, který vešel v obecnou laickou známost.* (Wikipedia, online, cit. 2015-05-24) Důležité je si uvědomit chronologický vývoj v oblasti metodik vývoje softwaru. Předtím než se označení „agilní“ začalo tak hojně vyskytovat, již bylo publikováno několik metodik vývoje softwaru, které podporovaly základní myšlenky agilního vývoje, tedy inkrementální a iterativní vývoj. Například metodika Scrum byla představena v roce 1986, Rapid Application Development (RAD) v roce 1994 a Extrémní Programování (XP) v roce 1996 (Larman, 2003). Pojem Agilní metodiky tedy pouze spojuje dohromady již dříve vytvořené metodiky a sjednocuje myšlenkové procesy a postupy používané v těchto metodikách.

Označení „agilní“ bylo poprvé spojeno s vývojem softwaru v roce 2001, kdy se v americkém státě Utah sešlo 17 nadšenců působících v oblasti vývoje softwaru. Na tomto setkání diskutovali o možnostech nahrazení dosud hojně používaných metod vývoje softwaru, převážně vodopádového modelu, metodami, které by byly jednoduché a přímočaré. V obecné shodě o nutnosti změny vytvořili Agilní Alianci (Agile Alliance) a sepsali Manifest agilního vývoje softwaru.

2.1 Manifest agilního vývoje softwaru

Manifest agilního vývoje softwaru je soupis čtyř základních principů, na které by měl být brán zřetel při vytváření softwaru. Autoři ve svém prohlášení tvrdí, že si uvědomují vážnost principů na pravé straně každého bodu manifest ale domnívají se, že daleko důležitější je část levá. Manifest má následující znění:

- 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 (http://agilemanifesto.org/iso/cs/, online, cit. 2015-05-31)

Manifest agilního vývoje softwaru byl přímou reakcí na tehdejší dobu. Autoři manifestu tvrdí, že vývoj softwaru se zvrhl v neudržitelnou mašinérii, protože většina převážně velkých firem si vážila více svých nastavených procesů než jednotlivců, kteří tyto procesy vykonávali. (Apke, 2014, strana 32) Toto mělo mít za následek jednoduchou škálovatelnost a možnost jednoduše přidávat vývojáře do již rozjetého projektu. Díky nastaveným procesům měl hned každý vědět, co a jak dělat. Realita byla však opačná. Výsledek většinou vyústil ve spoutání jednotlivců řetězy v podobě procesů a v potlačení jakékoliv kreativní činnosti. Odůvodněním ve většině případů bylo, že dle procesu a konkrétního artefaktu je již vše takto naplánováno a nelze to změnit. Docházelo tedy k omezení jakékoliv invence a změn v průběhu procesu. Tento nápad vypadá na papíře realisticky a autoři uznávají, že paralela s výrobním závodem je hezká myšlenka, bohužel disciplína vývoje softwaru je daleko náročnější a hlavně nepředvídatelnější činnost než manuální výroba dílů v továrně. Vývojový tým je sestaven z individualit, kde každý má své vlastní silné a slabé stránky. Klasický proces počítá s průměrností všech pracovníků, kdežto agilní procesy staví na tom, že jsou využívány silné stránky každého z členů týmu na maximum a tým jakožto celek čelí problémům a slabým stránkám společně. V tomto bodě Manifestu je kladen důraz na motivační faktor. Každý člověk je daleko šťastnější a motivovanější, má-li možnost aktivně se zapojit do řešení konkrétních problémů, a ne pouze plnit úkoly od nadřízeného manažera. Proto jsou agilní metody dnes tak populární. Vývoji softwaru se většinou věnují inteligentní lidé. Pokud těmto jedincům nabídneme možnost podílet se na rozhodování o zvolených technologiích a budoucnosti projektu, je daleko pravděpodobnější, že se do projektu daleko více zapojí a práce je bude více bavit, a tím budou dosahovat lepších a hlavně kvalitnějších výsledků. (Stellman, 2014, strana 34)

Dalším bodem Manifestu je „Fungující software před vyčerpávající dokumentací“. Tento bod je jedním z nejčastěji nesprávně vysvětlovaných. Často se můžeme setkat s názorem, že tým se řídí agilními metodikami a tím pádem k projektu nemá vůbec žádnou dokumentaci. Dle Apkeho (2014, strana 66) je tento bod nešťastně napsaný. Tento princip by neměl vyjadřovat to, že není pro projekt třeba žádná dokumentace, ale že daleko větší hodnotou je funkční software než obsáhlá dokumentace popisující každý detail. A právě ve slově detail tkví meritum věci. Apke navrhuje, že daleko účelnější a jasnější by bylo, kdyby bod zněl takto: Fungující software před obsáhlou

specifikací požadavků a dokumentací architektury (Apke, tamtéž). Dle Cohna (2004) by měly být požadavky popsány v rovině interakce uživatele se systémem a na detailech a konkrétní implementaci by se měl tým dohodnout. Díky agilnímu myšlení, kde zákazník je na prvním místě, je dodán výsledek, který opravdu funguje (Cohn, tamtéž). Autoři Manifestu tímto bodem také zdůraznili nutnost doručení softwaru, který opravdu funguje a přináší přidanou hodnotu. Ve vodopádovém modelu se velmi často stávalo, že systém byl doručen v naprosté shodě s dokumentací, která vznikla na začátku projektu. Nicméně celý software neplnil očekávané funkce uživatelů, a tak nebyl ani přes dokonalou dokumentaci použitelný.

Agilní metodiky vývoje softwaru vyžadují důvěru na všech úrovních svého procesu. Jednou z těchto úrovní je i vztah zákazníka s vykonavatelem projektu. Tento vztah je většinou právně ukotven za pomocí různých smluv a kontraktů. Manifest upřednostňuje vzájemnou spolupráci a komunikaci před uzavíráním šibeničních smluv. Cohn (2004, strana 86) ve své knize popisuje důležitost edukace zákazníka a vysvětlování jednoduchého paradigmatu, že tak komplexní aktivita, jako je vývoj softwaru, nemůže být do detailu zachycena jakoukoliv smlouvou. A pokud se o to pokusíme, tak s velkou pravděpodobností jedna ze stran na konci projektu nebude spokojena. Přesně to je myšlenka třetího bodu Manifestu o přednosti spolupráce se zákazníkem před sjednáváním smluv. Autoři se tím opět nesnaží říct, že by se neměly uzavírat žádné smlouvy. Jen upozorňují na fakt, že pokud se pokusíme do smlouvy zakotvit všechny požadavky na výsledný software, tak bude nejen kontraproduktivní, ale také to může výsledný software vážně poškodit. (Apke, 2014, strana 88) Metodika Scrum toto například řeší za pomoci reprezentanta ze strany zákazníka, který je denně přítomen při vývoji projektu a má plnou pravomoc ovlivňovat, jakou cestou se projekt bude ubírat, a to díky velmi krátkým vývojovým iteracím.

Poslední bod Manifestu je autorovi této práce velmi blízký. I když se princip reakce na změnu před dodržováním plánu zdá jako naprosto racionální a samozřejmá věc, osobně se setkal s tím, kdy byl jako vývojář ve firmě používající metodiku RUP nucen programovat funkce, které naprosto odporovaly zdravému rozumu a použitelnosti pro zákazníka. Zákazník se zaštiťoval tím, že v dané fázi projektu je již vše naplánováno, čili nezbývá než postupovat podle plánu. Změna byla možná až v příští etapě naplánované za půl roku. To byl například jeden z konkrétních impulsů,

proč si autor oblíbil agilní metodiky. Agilní myšlení si uvědomuje nutnost reakce na změnu v jakékoliv fázi projektu. Tímto přímo podporuje jednu z hlavních myšlenek agilních metodik, a to přinášet zákazníkovi přidanou hodnotu vyvinutým softwarem. Zákazník je tedy na prvním místě a agilní metodiky se nesmí změn v průběhu projektu bát, ale musí s nimi dokonce počítat. Apke (2013, 85) ve své knize dokonce tvrdí, že změna by měla být vítaným prvkem, jelikož pokud zákazník své požadavky nemění, tak nereaguje na své obchodní požadavky a konkurenci.

2.2 Principy stojící za Manifestem agilního vývoje softwaru

Manifest byl vytvořen s cílem reprezentovat velmi jasnými a pochopitelnými body celé agilní hnutí. Na sympoziu v americkém Utahu v roce 2001 bylo také formulováno dvanáct základních principů, dle kterých by se agilní metodiky měly řídit (http://agilemanifesto.org/iso/cs/principles.html, online, cit. 2015-05-31).

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

- Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.
- Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

Těchto 12 bodů velmi úzce navazuje na Manifest a usnadňuje jeho čitelnost a obecně pochopitelnost agilního myšlení. Je zde velmi jasně naznačeno, že spolupráce se zákazníkem a fungující software jakožto přidaná hodnota je nejdůležitějším stavebním kamenem jakéhokoliv procesu vývoje softwaru.

Highsmith (2010, strana 67) ve své knize popisuje, že: „*agilní vývoj umožňuje daleko rychlejší a pružnější reakce na měnící se vývoj trhu a náhlé a nepředvídatelné taktické rozhodnutí konkurence"*. Ve Velkém anglicko-českém slovníku (1991. str. 70) je jako překlad slova „agile" uvedeno „*hbitý, čilý, agilní, svižný; bystrý*. Zde tedy vidíme, odkud získaly agilní metodiky svůj příhodný název.

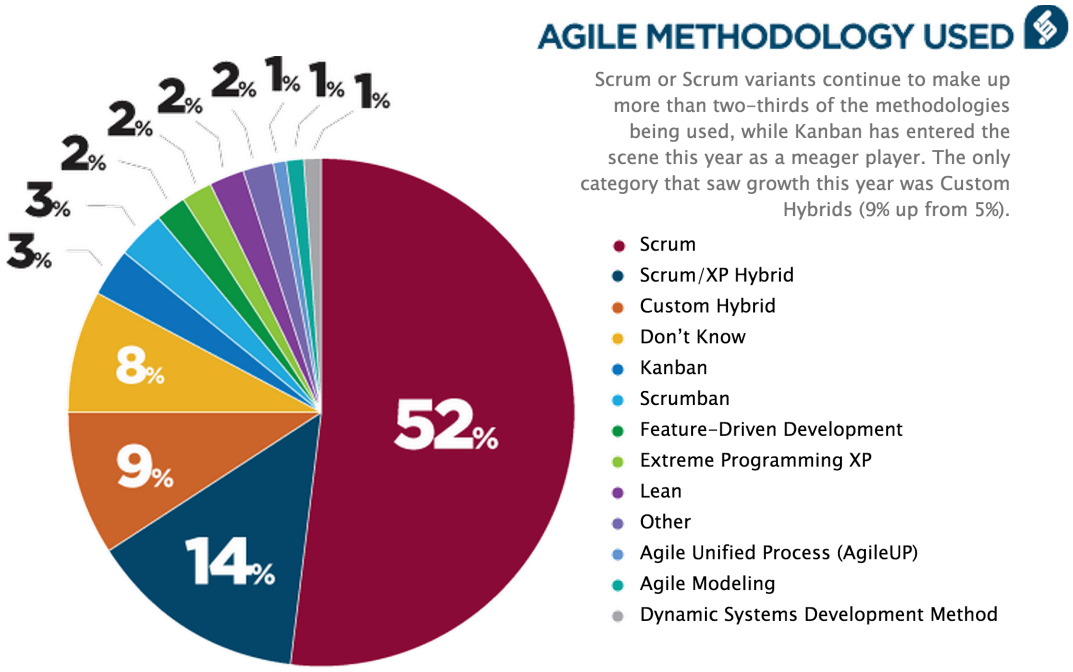
Na rozdíl od tradičního vodopádového přístupu, kde je projekt na začátku zcela definován a pak podle tohoto plánu vyvinut, se agilní přístupy spoléhají na iterativní vývoj, kde na konci každého cyklu je zákazníkovi prezentována plně funkční část (inkrement) produktu. Zákazník a vývojáři mohou pak na základě vizuálního ověření inkrementu navrhovat změny, které mohou být zahrnuty do příští iterace. Obsah příští iterace je plně v kompetenci zákazníka, který udává směr, jakým se projekt ubírá. Zákazníkovi je tímto dána možnost reagovat v jakékoliv fázi projektu pružně na aktuální potřeby. Vzhledem k tomu, že se dodává plně funkční inkrement produktu, může zákazník produkt začít používat již během samotného vývoje a nečekat až na jeho úplné dodání. Právě pomocí aktivního používání systému může přicházet na funkčnosti, které nutně potřebuje, ale na začátku projektu si je neuvědomoval. Předchází se tím tedy situaci, že zákazníkovi je dodán výsledný software, který vůbec nepotřebuje nebo který je v době dodání již zastaralý.

2.3 Popularita agilních metodik

Agilní metodiky, hlavně Scrum, Extrémní programování nebo jejich kombinace, jsou u softwarových firem dnes velmi populární. Dokazuje to výzkum společnosti VersionOne, který v roce 2011 zkoumal stav agilního vývoje (State of Agile

Development Survey Results, 2011, online, http://www.versionone.com/state_of_agile_development_survey/2011/). Tento výzkum dokázal, že 80% dotázaných respondentů potvrzuje používání jedné z agilních metodik pro vývoj softwarových projektů. 15% agilní metodiky nepoužívá a 5% si není jisto, zda agilní metodiky používá či ne. Jako hlavní důvod pro přijetí jedné z agilních metodik označilo 84% dotazovaných zlepšení schopnosti kontrolovat změnové požadavky klientů. Druhým důvodem, pro téměř 77% dotázaných, bylo zlepšení přehlednosti stavu projektu. 75% respondentů potvrdilo zlepšení v oblasti produktivity a týmové morálky.

Co se týče zastoupení jednotlivých metodik, tak je na čele s velkým náskokem 52 % metodika Scrum. Kombinace metodiky Scrum a Extrémního programování je na druhé pozici s 14% zastoupením. Zajímavou položkou je třetí příčka, kde 9 % respondentů uvedlo, že používá agilní metodiku, kterou si sami sestavili. V roce 2011, kdy byl tento výzkum publikován, byl zaznamenán nárůst těchto vlastních metodik z pěti na devět procent.



Obrázek 4 - Distribuce používání agilních metodik (zdroj: http://www.versionone.com/state_of_agile_development_survey/2011/, online)

Jedním z hlavních principů oblíbenosti agilních metodik je oproti klasickým (někdy označovaným také jako rigorózní) přístup k třem základním faktorům každého projektu:

- funkcionalita,
- rozpočet a
- časový plán.

Často jsou tyto faktory reprezentovány ve trojúhelníku a dle klasických manažerských pouček víme, že pokud měníme jeden z faktorů, tak to ovlivňuje nejméně jeden další faktor. Pokud měníme například rozsah funkčnosti, tak to přímo ovlivňuje rozpočet, časový plán nebo jejich kombinaci. Pro klasické metodiky je typické, že díky velkému plánování na začátku projektu je pevně ukotvena právě funkčnost projektu a druhé dva faktory, rozpočet a časový plán, se musí funkčnosti přizpůsobit. Oproti tomu u agilních metodik je typické, že je pevně daný rozpočet na projekt a díky krátkým iteracím i časový plán, jediným variabilním faktorem je tedy funkčnost. Teoreticky je totiž na konci každé iterace zákazníkovi předán funkční inkrement projektu a je pouze na zákazníkově rozhodnutí, jak bude v projektu dál pokračovat, jakou funkčnost naplánuje do další iterace nebo zda se rozhodne projekt ukončit s tím, že je s výsledným inkrementem spokojen a další vývoj již není potřeba, případně na něj není dostatek financí. Highsmith (2010, strana 229) toto uvádí jako jednu z velkých výhod agilního vývoje, protože teoreticky nikdy nemůže dojít k překročení rozpočtu projektu a projekt je vždy dodán ve smluvený termín. Jedinou neznámou je rozsah funkčnosti na konci projektu. Díky krátkým iteracím a reprezentantovi zákazníka přítomném denně na projektu má zákazník přímou možnost tuto výslednou funkčnost ovlivnit dle jeho přání.

Dalším markantním rozdílem, který přidává agilním metodikám na popularitě, je způsob řízení a zapojení pracovníků. V klasických metodikách je způsob řízení takzvanou metodou „top-to-bottom“, tedy že strategická rozhodnutí jsou činěna na vrcholku organizační struktury a tou procházejí dolů, tedy k lidem kteří opravdu vytvářejí zákazníkův produkt. V době, kdy k nim strategické rozhodnutí dorazí, již většinou mají velmi malou nebo nulovou šanci toto rozhodnutí nějak ovlivnit a vložit do něj svoji myšlenku. Toto je pro pracovníky velmi frustrující a demotivační. V agilních metodikách se toto paradigma otáčí a jedná se o takzvanou metodu

„bottom-up“, tedy že klíčová strategická rozhodnutí by měla přicházet od těch nejvíce zainteresovaných osob, tedy pracovníků samotných. Tento systém nejen že podporuje motivaci a zapojení pracovníků, ale také přímo ovlivňuje kvalitu jejich rozhodnutí. V klasické metodice o všem rozhoduje manažer, který je velmi často naprosto vzdálen od reálného dění v projektu a má pouhou iluzi o tom, co se děje. Kdežto pokud je do procesu rozhodování zapojen zaměstnanec, který je v každodenním kontaktu s projektem, tak ví daleko lépe, co zrovna projekt nejvíce potřebuje.

Další nespornou výhodou je proces neustálého zlepšování vývojového procesu. Agilní metody přímo vybízejí uživatele ke změnám procesu, pokud nějaká část nefunguje dle jejich představ. Po každé iteraci by se měly všechny zúčastněné strany projektu sejít a diskutovat o možnostech zlepšení nastaveného procesu. Tato diskuze by neměla být ovlivněna postavením v hierarchii organizační struktury a každý by měl mít možnost navrhnout změnu, která je následně týmem demokraticky odhlasována a je rozhodnuto o jejím přijetí nebo zamítnutí. Prvky demokracie a rovnosti jsou v agilních metodikách velmi silně přítomny a staví pozice jednotlivých účastníků procesu převážně na respektu oproti klasickým metodikám, které staví převážně na pozici v organizační struktuře.

2.4 Metodiky agilního vývoje software

Jak již bylo uvedeno v předešlé kapitole, můžeme agilní metodiky rozdělit do několika konkrétních variant. Většina těchto metodik již byla funkční daleko dříve, než byl pojem „agilní“ vůbec představen světu a tak hojně rozšířen. Každá z níže popsaných metodik pracuje s tím, že uživatelé metodik (vývojáři) mají agilní myšlení. Tedy, že si uvědomují hlavní základy agilních metodik jak je popisuje Manifest agilního vývoje softwaru a agilní principy. Hlavním rysem agilních metodik je to, že nejsou dogmatické jako jejich rigorózní předchůdci. Většina agilních metodik detailně nepopisuje každou z aktivit projektu, místo toho určuje spíše jakési hranice a dává za příklad osvědčené postupy z jiných projektů a doporučuje jejich aplikaci a přizpůsobení na prostředí dané společnosti nebo projektu (Beck, 2005, strana 205).

2.4.1 Extrémní programování

Kent Beck publikoval v devadesátých letech svoji metodiku Extrémní programování, která byla přímým protikladem vodopádového modelu a získala si velkou popularitu. Metodika stojí na jednoduché myšlence - inkrementální vývoj a jednoduchost. Metodika doporučuje krátké iterace, jejichž výsledkem je plně funkční inkrement, tyto operace je třeba provádět tak dlouho, dokud zákazník a tým nejsou spokojeni s výsledkem (Beck, 2005, strana 124). Extrémní programování přímo vyžaduje použití určitých vývojových technik, například „test-driven development“, což je technika kdy vývojář nejprve napíše automatické testy pro vyvíjenou funkčnost, které zpočátku selhávají. Takto má jistotu, že až funkčnost naimplementuje, tak testy proběhnou úspěšně. Také se tím vytvoří stoprocentní pokrytí aplikace testy, takže při budoucí změně jsou velmi přehledně vidět následky, jež změna způsobila, jelikož testy mohou v některé jiné části aplikace začít selhávat. Vývojář je díky vysokému zapojení automatizace procesů na tuto skutečnost upozorněn a kód nebo test upraví. Tímto je zajištěn velmi kvalitní produkt. Tato metoda také usnadňuje refactoring, tedy zjednodušování kódu, jelikož při pokusu o zjednodušení nebo optimalizaci mají vývojáři okamžitou zpětnou vazbu, že kód nefunguje (Meyer, 2014, strana 49). Extrémní programování se spoléhá na následující sadu principů (Beck, 2005, strana 78):

- **Krátké iterace**.
- **Párové programování**, tedy technika, kdy se dva vývojáři dělí o jednu klávesnici a myš a společně programují kód. Kent Beck tvrdí že zde platí známé pravidlo: „víc hlav, víc ví“. Vývojáři se mohou o postupu radit a konverzace nejen že přináší kvalitnější kód, ale díky střídání těchto dvojic také rozšiřuje znalost jednotlivých developerů, kteří se od sebe vzájemně učí novým technikám.
- **Uživatelské scénáře.** Tedy jednoduchý zápis požadavků na funkčnost, který je napsán ve formátu srozumitelné věty, jež popisuje, jak uživatel zachází se systémem. Například: Já, jakožto uživatel chci mít možnost načíst všechny tržby za zvolené období, abych je mohl předat účetní.
- **Refactoring.** Zjednodušování a upravování kódu.
- **Otevřené pracoviště.** Upřímnost, přehlednost a otevřenost na všech úrovních rozhodovacích procesů.

- **Kolektivní vlastnictví kódu.** Kód je vlastněn celým vývojovým týmem a nikoliv jednotlivcem a mělo by být v nejlepším zájmu každého vývojáře ho udržovat co nejpřehlednější a funkční.
- **Kontinuální integrace.** Vývojová technika, která spočívá v časté kompilaci zdrojového kódu projektu, po níž následuje spuštění testů. Tyto testy díky test-driven developmentu pokrývají velkou část aplikace a díky častému spouštěním testů se vývojový tým ihned dozví, že nastala chyba, kterou je nutno opravit
- **Test-driven development.** Psaní testů před psaním funkčního kódu, tím je ověřeno, že kód funguje a je zaručena kvalita produktu.

Extrémní programování je v současné době často používáno v kombinaci s některou jinou metodikou, nejčastěji s metodikou Scrum. Extrémní programování totiž velmi striktně předepisuje vývojové metody, které musí tým používat, což například Scrum vůbec neřeší. Extrémní programování představilo světu výše popsané vývojářské principy. Slovo „extrémní“ má popisovat rozsáhlost těchto principů, jelikož by se jimi v procesu měla řídit každá vyvíjená funkčnost v projektu (Chromatic, 2003).

2.4.2 Crystal

Agilní metodika Crystal byla představena jedním z původních sedmnácti signatářů Manifestu agilního vývoje softwaru Alistairem Cockburnem. Metodika Crystal v sobě obsahuje kombinaci několika různých metodik a projekt rozděluje na dvě dimenze. Dimenzi kritičnosti a dimenzi velikosti projektu. Každá z dimenzí je dále rozdělena na čtyři části. Od nejmenšího po největší projekt a od nejméně kritický po ten nejkritičtější. Metodika Crystal popisuje pro každý z takto vzniklé matice šestnácti elementů nejvhodnější metody a přístupy. (Meyer, 2014, strana 234). Metodika Crystal popisuje sedm základních principů, pomocí kterých by se měl projekt řídit (Cockburn, 2005, strana 114):

- **Časté doručování** funkčního a otestovaného kódu je nejdůležitější vlastnost každého projektu. Tento princip je častý ve všech agilních metodikách.
- **Reflexivní zlepšování** díky pravidelným schůzkám, na kterých se probírá, jak věci fungují a jak se dají zlepšit. Tyto schůzky by měly být časté, nejméně jedenkrát do měsíce.

- **Lidská komunikace,** tedy možnost otevřeně a upřímně komunikovat mezi všemi úrovněmi pracovníků nezávisle na jejich postavení v organizační struktuře.
- **Osobní bezpečí,** tedy například nemožnost vedoucích pracovníků nutit pracovat přesčas nebo trestat za otevřenost názoru. Možnost se vyjádřit o tom, že projekt v plně plánované funkčnosti není možné stihnout ve vymezeném čase.
- **Soustředění**. Metodika Crystal například říká, že by se vývojáři měli zaměřovat pouze na jeden úkol a nepřecházet mezi více úkoly. Stejně tak není správné zaměstnance neustále vyrušovat a nenechat je dokončovat dílčí části úkolů.
- **Dostupnost expertů**. Metodika Crystal klade důraz na možnost kdykoliv konzultovat jakoukoliv překážku s expertem v dané problematice. Tedy zajistit, aby byl například byl vždy dostupný expert zákazníka, který může okamžitě vysvětlit otázky ohledně implementace a předcházet tím zdržení projektu.
- **Zajištění technické excelence, automatického přístupu a časté integrace**. Relativně dlouhý princip ale poměrně sebevysvětlující. Tedy pro dobrý výsledek je potřeba zajistit zaměstnancům dobré vybavení. Často se integrací a automatizací zajistí odhalení problémů v raném stádiu a náklady na jejich opravení budou menší, než kdyby byly objeveny až na konci projektu.

Metodika Crystal není direktivní a nechává poměrně velkou volnost v implementaci. Metodika nepředepisuje žádné vývojové techniky jako například Extrémní programovaní, ani neříká, jakým způsobem se mají jednotlivé iterace plánovat jako například Scrum (Cockburn, 2005, strana 187).

2.4.3 DSDM

V roce 1994 byla představena metodika Dynamic systems development method, zkráceně DSDM. Tato agilní metodika vychází z již představených agilních hodnot. Metodika se řídí osmi základními principy (Stapleton, 1998):

- **Důraz na požadavky zákazníků**. Zákazník je na prvním místě a fungující software je to nejdůležitější, co v projektu je. Zaměření je především na kriticky důležité scénáře systému, které přináší zákazníkovi přidanou hodnotu.

- **Doručování včas**. DDSM doporučuje používat *timeboxing*, česky můžeme tuto metodu vyjádřit jako ohraničování časem. Tedy nehledě na to, co děláme, tak bychom si na to měli vymezit určitý čas a ten dodržet, předejdeme tím neplánovaným zpožděním. Doručování projektů včas a dodržování uzávěrek buduje vzájemnou důvěru mezi týmem a zákazníkem.
- **Spolupráce**. Zapojením zákazníka a vývojářů do projektu můžeme dosáhnout maximálně efektivních výsledků.
- **Vždy kvalitní.** Kvalitní software je základ. Frustrace z nekvalitního softwaru zkazí nadšení z projektu jak pro vývojáře, tak pro zákazníky. Metodika nabádá testovat často a ideálně automaticky a nikdy neslevit z kvality.
- **Inkrementální vývoj na solidních základech**. Funkční inkrement, který si zákazník může na konci každé iterace otestovat, je nejlepším prostředkem zajištění zpětné vazby a ověřování si správného směru projektu.
- **Iterativní vývoj**. Doručovat software v krátkých časových intervalech umožňuje pružnou reakci na změnu požadavků zákazníka.
- **Neustálá a přímá komunikace.** Pro efektivní proces je důležitá efektivní komunikace. Nikdo by neměl mít strach vyjádřit svůj názor v jakékoliv fázi projektu. Názor každého člena týmu by měl být demokraticky posouzen a zhodnocen.
- **Viditelnost stavu projektu.** Informovanost všech členů týmu je důležitá, stav projektu musí být tedy kdykoliv přístupný všem členům vývojového týmu, ale stejně tak i zákazníkovi.

Tyto principy neobsahují nic překvapivého, co by již nebylo zmíněno v jiné agilní metodice. Zajímavým prvkem, který metodika DSDM přinesla, je způsob hodnocení priorit, takzvanou MoSCoW metodu. Tato metoda určování priorit je rozložena do čtyř kategorií, které vycházejí z prvních písmen jejich anglického označení. Tato kategorizace je určena v časovém kontextu konkrétní etapy projektu a může se během této etapy měnit.

- **M – Must** (musí), tedy požadavky, které musí být doručeny aby mohl být projekt považován za úspěšný.
- **S – Should** (měly), reprezentuje kategorii požadavků, které mají středně velkou prioritu a měly by být ve výsledném projektu obsaženy. Prioritu však mají požadavky z první kategorie.

- **C – Could** (mohly), kategorie která spojuje požadavky, jež si zákazník přeje, ale nejsou pro něj kriticky důležité, a tak jejich implementace závisí na možnostech a kapacitách vývojového týmu.
- **W – Won't** (neměly), je skupina požadavků, které se po dohodě se zákazníkem nebudou v této etapě projektu implementovat. Neznamená to však, že během příští etapy se tyto požadavky nemohou posunout do vyšší kategorie.

3 Scrum

„Scrum: Týmový framework pro vytváření komplexních systémů a produktů“ - Scrum Aliance (Johnson, 2011, strana 100)

„Termín Scrum je definován tak, že neříká o vývoji softwaru nic konkrétního. Scrum je o řízení práce a týmové dynamice a může být použit i mimo softwarové prostředí.“ – Jeff McKenna (Johnson, 2011, strana 100)

Historicky první scrumový tým byl vytvořen v roce 1993, kdy Jeff Sutherland, tehdy technologický viceprezident společnosti Easel Corporation, byl již unaven stále neúspěšnými projekty a nefungujícími přístupy, jako je vodopádový model. Členy tohoto prvního scrumového týmu byli i Jeff McKenna a John Scumniolates. Všichni tři sdíleli myšlenku, že je na čase dělat vývoj softwaru jinak a v roce 2001 se stali signatáři Manifestu agilního vývoje softwaru.

Sutherland své nápady na změnu čerpal v knize Wicked Problems, Righteous Solutions: A Catalog of Modern Engineering Paradigms od Petra DeGrace a Leslie Hulet Stahla. Autoři této knihy byli ohromeni článkem z periodika Harvard Business Review z roku 1986, konkrétně článkem od Hirotaky Takeuchiho a Ikujira Nonaky, „The New New Product Development Game“, který přirovnával týmovou práci k mlýnici (anglicky „scrum“) z ragby. Článek popisuje holistický přístup, který by tým měl přijmout a stejně jako v mlýnici v ragby se do sebe zaklesnout a pohybovat se společně vpřed a přitom překonávat všechny nástrahy. Japonští autoři se však dle Sutherlanda až příliš inspirovali japonským přístupem ve výrobních továrnách, kde celý systém byl vytvořen jen v hlavách několika jedinců. A po jejich odchodu se systém budoval znovu. Sutherland zastával myšlenku, že tým musí fungovat jako jeden celek a úzce mezi sebou spolupracovat, aby odchod jednotlivce nezpůsobil kolaps celého procesu (Sutherland, 2014, strana 221).

Ředitel společnosti ve které Sutherland působil, si často stěžoval, že žádný plán ještě nepomohl k dokončení požadované funkčnosti ve slíbený čas. Prodlevy v projektech poškozují dobré jméno společnosti a způsobují finanční ztrátu. Sutherland navrhl

proces, který začal nazývat Scrum, ve kterém jsou úkoly naplánované na krátkou iteraci, kterou nazýval „sprint“. Záleží pouze na týmu, který musí velmi úzce spolupracovat, jakým způsobem se rozhodne, že bude nejlepší dosáhnout těchto cílů. Cíle musí být splnitelné a analogie s běžeckým sprintem je nasnadě (atlet při sprintu běží pouze velmi krátký úsek, v němž je schopen přesně určit každý svůj krok). Během sprintu tento tým nesmí nikdo vyrušovat, aby bylo možné doručit to co slíbil. Na konci sprintu je výsledkem software, který je demonstrován zákazníkovi a je plně funkční. Na konci každého sprintu je možné se rozhodnout, zda v projektu pokračovat a jakým směrem. Toto rozhodnutí dělá pouze zákazník a je tedy pouze na něm, co si vybere. Funkční kód buduje u zákazníka daleko větší důvěru než jeho pouhý popis v dokumentaci. Ředitel společnosti Sutherlandův proces přijal a výsledky se doručili během šesti měsíců. Zákazníci byly s výsledky projektů spokojení a dobré reference vedli ke zvýšení celkových tržeb (Sutherland, 2014).

O formální publikování metodiky Scrum se zasloužil i Kent Schwaber, který ve své firmě také praktikoval nápady převzaté z japonských výrobních továren. Schwaber byl zastáncem myšlenek, že projekty by se měly řídit pomocí empirického procesu (empirický = řídící se zkušenostmi; zkontroluj a zlepši), a ne pomocí definovaného procesu (naplánuj a proveď). Schwaber a Sutherland se znali již z dřívějších projektů a o procesech spolu diskutovali. V roce 1995 se sešli na konferenci o objektovém programování a domluvili se, že publikují metodiku Scrum, která bude jejich přístupy spojovat (Schwaber, 2004). Schwaber později založil Scrum Alliance, která má za úkol rozšiřovat povědomí o metodice a vytvořil systém certifikací pro jednotlivé role scrumového procesu.

3.1 Role Scrumu

Scrumový tým v sobě spojuje všechny tradiční role, které známe z klasických metodik. Ve scrumovém týmu najdeme jednotlivce s různými sadami zkušeností. Vývojáře, architekty, designéry, testery, produktové a projektové manažery. Každá z těchto rolí je v projektu potřebná a důležitá. Scrum se ale místo škatulkování jednotlivců do konkrétních rolí snaží sestavit homogenní tým, který dokáže čelit projektu společně. Scrum rozlišuje pouze tři role v projektu : Product Owner, Scrum Master a člen týmu (Schwaber, 2004). Anglické termíny jsou natolik zaužívané, že

byly v rámci této práce ponechány v původním znění (to znamená, že je respektováno oficiální názvosloví této metodiky).

3.1.1 Product Owner

Vývoj softwaru představuje pro zákazníka většinou velkou investici. Za tyto peníze očekává návratovou hodnotu, která zvýší přidanou hodnotu celé firmy. Hlavní rolí Product Ownera je zajistit co nejvyšší přidanou hodnotu, kterou firma za svoji investici získá. Product Owner je jediná zodpovědná osoba, která může měnit priority projektu a rozhodovat o složení jednotlivých iterací. Nejlépe funguje, pokud je tato role obsazena pracovníkem z týmu zákazníka (Johnson, 2011, strana 122). Product Owner přímo reprezentuje zákazníkovy zájmy a pouze on může určit, na čem bude scrumový tým pracovat. Tímto je projektový tým chráněn a může se soustředit na současnou iteraci. Další povinností je sběr požadavků a jejich komunikace mezi zákazníkem (stakeholders) a vývojový týmem. Zákazníci dost často neumí své požadavky správně komunikovat a často ani nevědí, jaký software vlastně potřebují. Rolí Product Ownera je tyto požadavky zjistit, přeložit je do jazyku, kterému rozumí vývojový tým a případně zodpovídat na otázky obou stran, tedy jak zákazníka, tak vývojového týmu. Tato fáze je důležitá, jelikož zabraňuje tomu, aby se vývojový tým zdržoval prací na úkolech, které zákazník vůbec nechce a nepřidávají projektu přidanou hodnotu (Schwaber, 2004, strana 233). Product Owner je zodpovědný za vytvoření a udržování projektové vize, tedy kterým směrem se bude projekt ubírat a jak ho budou uživatelé používat. Tato vize se samozřejmě může s vývojem produktu měnit a právě flexibilitou a změnou požadavků by se měl zabývat každý Product Owner. Product Owner je ta nejkritičtější role na projektu, jelikož pokud selže ve svých povinnostech, tak to má přímo vliv na zákazníka, který nedostane produkt, jenž si objednal. Product Owner zapisuje požadavky získané od zákazníka za pomoci uživatelských scénářů. Tedy jednoduchým a srozumitelným způsobem, který lidským jazykem popisuje, jak bude uživatel systém používat. Většinou se používá formát: *„Jako <typ uživatele> chci <funkčnost> aby<byla vytvořena nějaká hodnota>”* (Schwaber, 2004, strana 123).

Agilní kouč Simon Baker (2012) ve své knize popsal Product Ownera jako člověka který za pomoci psaní uživatelských scénářů a akceptačních testů; prioritizování scénářů dle přínosu přidané hodnoty; rozhodování které scénáře se budou

implementovat v příštím sprintu; a poskytování přímé zpětné vazby – řídí projekt a je zodpovědný za jeho výsledek. Jako hlavní strůjce úspěšného projektu musí být vidět, slyšet a musí být za všech okolností objektivní.

3.1.2 Scrum Master

„Scrum Master není projektový manažer.“ - Chris Sims (Rubin, 2012, strana 120)

Scrum Master působí ve scrumovém týmu jako kouč, který vede tým k ještě lepším výsledkům, spolupráci a organizovanosti. Hodnotícím výsledkem týmu je doručený produkt a šťastný zákazník. Hodnotícím výsledkem Scrum Mastera je lepší a schopnější tým. Hlavním úlohou Scrum Mastera je působit jako mentor a ochranář pro vývojový tým, nikdy by neměl být stavěn do role vedoucího pracovníka, který svoji roli vykonává z pozice moci a postavení ve struktuře organizace. Jeho úkolem je edukovat tým, ale i zákazníka v agilních hodnotách a hlídat dodržování procesu, na kterém se tým dohodl. Při schůzkách a diskuzích zaujímá většinou pozici moderátora a pomáhá udržovat směr diskuze a varuje tým, pokud se od směru diskuze odchyluje. Scrum Master není člověk, který by měl rozhodující pravomoci, tedy nemůže vývojový tým nutit například pracovat na konkrétním zadání ani používat konkrétní techniku. Scrum definuje tým jako samo-organizující se jednotku, která sama a demokraticky rozhoduje o tom, co je pro ni nejlepší (Schwaber, 2004, strana 233). Scrum Master musí být expert v agilním přístupu a metodice Scrum a může týmu navrhovat, jaké přístupy a techniky by měl přijmout. Rozhodnutí o jejich adopci je však pouze rozhodnutím týmu (Sutherland, 2014, strana 112).

Scrum Master by měl tým chránit před vnějšími vlivy a zajistit klid na práci a dodržování procesu. Měl by tedy například ochránit tým před zainteresovanými osobami, které v průběhu iterací ruší členy týmu zadáváním extra práce s tím, že toto je jen drobnost, kterou nutně potřebují (Sutherland, 2014, strana 223). Scrum Master by měl týmu pomáhat řešit všechny problémy, které mohou nastat a potencionálně zdržovat tým. Johnson ve svých publikacích označuje Scrum Mastera jako buldozér na problémy. Také musí zajistit, že každý člen týmu má k dispozici právě tolik informací, kolik ke své efektivní práci potřebuje. Například ví, kdy se koná která schůzka a kde najít pracovní postupy nebo na koho se obrátit, pokud někdo neví, jak má při práci na svém úkolu pokračovat., stejně tak pomáhá komunikovat členům

týmu. Občas se stává, že vývojář je špičkový odborník ve svém oboru a skvěle rozumí například programování. Tito jedinci mají občas problém s komunikací a trpí pocitem, že jim nikdo nerozumí a proto jsou často uzavření a nesdělují svoje názory nebo jsou jednoduše přehlušeni více aktivními lidmi, kteří mohou mít menší znalosti. Úlohou Scrum Mastera je vytvořit prostor pro komunikaci a dát každému členovi týmu stejnou možnost se vyjádřit. (Baker, 2012, strana 321)

Scrum Master nemusí být nutně dedikovaná role a můžeme se setkat se Scrum Mastery, kteří přispívají i jako vývojáři. U některých týmů toto funguje lépe a u některých hůře, hlavně ve chvíli, kdy se projekt dostane do kritické fáze a datum dodání se začne blížit. Scrum Master, který má i povinnosti vývojáře, se v této fázi může soustředit především na doručování kódu. Opravdu ho je ale potřeba v těchto těžkých situací na opačné straně, kde pomáhá týmu efektivně práci zvládnout a odstraňuje problémy ostatních. Takto nastavený proces vyžaduje velmi silné jedince, kteří jsou si dobře vědomi svých rolí a umí mezi nimi efektivně přepínat. (Johnson, 2011, strana 154)

3.1.3 Tým

Scrumové týmy jsou založeny na komunikaci, respektu a vzájemné důvěře. Důležitým prvkem těchto týmů je jejich schopnost vlastní organizace. Členové týmu mají naprostou kontrolu nad tím, jak a kdy bude práce v daném sprintu vykonána. Tým se také může rozhodnout, jaké nástroje a techniky pro řešení úkolů specifikované Product Ownerem na začátku sprintu použijí. Nikdo, ani ředitel společnosti, jim v tomto nemůže bránit nebo přikazovat jinak. Toto je základní pravidlo, které když je porušeno, tak nejen že odporuje metodice Scrum, ale také nabourává důvěru a respekt v týmu (Rubin, 2012, strana 78). To samé se týká odhadování délky a náročnosti práce ve Scrumu jednotlivých uživatelských scénářů. V ostatních neagilních metodikách určují odhady jednotlivých prací manažeři. Scrum říká, že odhadovat náročnost a délku mohou pouze ti, kteří práci ve výsledku dělají. V tomto případě členové vývojového týmu. Nikdo jim odhady nemůže nutit ani je zpochybňovat. Odhadování bude detailněji zpracováno v dalších kapitolách, ale je důležité říci, že odhadování, stejně jako jiné aktivity, probíhá v kolektivním duchu a názor každého má svou váhu. (Rubin, 2012, strana 85)

Ideální velikost scrumového týmu je sedm členů, plus minus dva. V týmu musí být zastoupeny všechny role, aby byl schopen splnit jakýkoliv uživatelský scénář v seznamu. Často se uvádí, že tým musí být takzvaně „cross-functional“, tedy že společnými silami musí být schopen doručit všechny disciplíny, které projekt vyžaduje. Od návrhu přes architekturu, programování až po testování a nasazení. Každý člen nemusí umět všechny tyto dovednosti, ale scrumové týmy nejsou o individualitách, ale o týmové práci. Pět až devět členu je ideální počet. Méně členů nemusí mít všechny potřebné dovednosti a více členů naopak může zabraňovat efektivní komunikaci (Sutherland, 2014, strana 321).

Zdlouhavé procesy a vyčerpávající dokumentace v rigorózních metodikách se snaží vytvořit iluzi, že členové týmu jsou nahraditelní. Schwaber (2004, strana 187) ve své knize říká, že opak je pravdou. Scrumový tým je unikátní a každý jeho člen je nenahraditelný pro svoji jedinečnost a sadu dovedností. Právě týmová práce, sociální vztahy a komunikace je to nejcennější, co tým má, a každá změna v týmu ovlivní jeho celkovou dynamiku. Pro měření rychlosti je ve Scrumu definovaná metrika „velocity“(rychlost), čili počet uživatelských scénářů, který je tým schopen doručit v určitém časovém období. Schwaber říká, že hodnota této metriky je přímo závislá na členech vývojového týmu a při jeho změně by se měla začít měřit od začátku.

3.2 Sprint

Sprint je základním stavebním metodiky Scrum. Sprinty udávají rytmus celému projektu. Koncept sprintů je obsažen ve všech agilních metodikách, kde je kladen důraz na iterativní vývoj a dokončování úkolů. Johnson ve své knize píše, že je nepodstatné pokud je tato krátká neustále se opakující doba nazývána sprintem, iteracím nebo cyklem. Důležitý je proces, kdy ukrajujete malé kousky z velkého koláče a nové si vezmete až po jejich dokončení a na konci každého sprintu máte funkční a nasaditelný inkrement.

Scrum přesně nedefinuje délku sprintu a nechává volnou ruku týmům, aby našly tempo, které jim nejvíce vyhovuje. Doporučuje však iterace delší než jeden týden a kratší než měsíc. Sprint je relativně krátký časový úsek, na jehož konec by měli být schopni členové týmu dohlédnout. Tedy představit si zcela konkrétně, jak projekt

bude vypadat na konci sprintu. Delší sprinty by dle Bakera (2012) mohly mít za následek nemožnost dohlédnout na konec této naplánované dráhy a mohly by mít za následek pouze stereotypní plnění úkolů. Kratší interval minimalizuje riziko toho, že výsledek nebude včas dodán..

Agilní kouč James Shore (2008, strana 54) ve své knize popisuje, že typický vývojový tým potřebuje šest sprintů, aby se naučil a pochopil metodiku Scrum a agilní myšlení. Podotýká, že u těchto šest sprintů nezávisí na jejich délce. Shore také doporučuje začít s krátkou jednotýdenní iterací a postupně sprinty prodlužovat.

Scrum doporučuje během sprintu organizovat různé schůzky. Ve shodě s běžnou praxí nejsou názvy schůzek v této práci počeštěny:

- Sprint Planning Meeting
- Daily Scrum
- Backlog grooming – Tato schůzka není v oficiální definici Scrumu, nicméně ji všichni autoři ve svých publikacích doporučují.
- Sprint Review
- Retrospective

3.2.1 Sprint Planning Meeting

Sprint Planning Meeting, česky plánovací schůzka, označuje začátek sprintu. Obvykle se dále dělí na dvě části. V první části schůzky se tým zaměřuje na definici toho, co na konci sprintu bude hotové. Tedy sadu hodnotících kritérií, která určí, zda byl sprint úspěšný nebo ne. V druhé fázi se tým dohodne, jak konkrétně k určené práci přistoupí, jak si ji rozdělí a jaké techniky bude po dobu sprintu používat. Doporučená délka této schůzky je jedna až dvě hodiny za každý týden sprintu. Pokud plánujeme dvoutýdenní sprint, pak bychom si měli vystačit se čtyřmi hodinami. Pokud je tento čas překročen, tak by se stejně jako u jakékoliv jiné scrumové aktivity měl tým poradit, jak tento proces zefektivnit. Díky této technice zvané *time-boxing* se nebudou schůzky prodlužovat nekonečně dlouho, každý by se měl snažit, aby byly co nejkratší, jelikož pokud je čas překročen, tak je schůzka ukončena.

První část: Co budeme dělat

V této části se tým a Product Owner společně domluví, jaké uživatelské scénáře budou na konci sprintu hotovy. Množství uživatelských scénářů závisí pouze na možnostech týmu a tým samotný si určuje, zda je počet malý, akorát nebo příliš velký s ohledem na to, aby na konci sprintu stoprocentně scénáře doručil. Tým nesmí nikdo nutit, aby do sprintu přijal scénář, u něhož si sám tým není jist, že ho může úspěšně dodat na konci sprintu. Prioritu jednotlivých scénářů, které jsou do sprintu naplánovány, určuje Product Owner. Určuje pouze jejich prioritu, nikoliv pořadí jejich implementace ve sprintu, toto rozhodnutí již záleží výhradně na týmu. V této části sprintu má Product Owner možnost vysvětlit členům týmu, jaká jsou akceptační kritéria scénáře a co vše musí fungovat aby on, jakožto reprezentant zákazníka, byl spokojen. Tým by uživatelský scénář neměl na této schůzce vidět poprvé. Scénář by měl být odhadnut a diskutován na schůzce Backlog grooming , na této by se mělo jen ověřit vzájemné porozumění mezi týmem a Product Ownerem.

Vzhledem k tomu, že Scrum je empirický proces, tak právě zkušenosti z minulých sprintů pomáhají naplánovat týmu vhodné množství práce do sprintu. Zmínili jsme zde již veličinu velocity, neboli kolik scénářů je tým schopen implementovat za určitý čas. Právě tato veličina by týmu měla být v plánování sprintu nápomocna, ačkoliv je pouze pomocníkem a tým se jí nemusí striktně držet.

Druhá část: Jak to budeme dělat

Ve druhé části této schůzky si tým dohodne, jaký přístup zaujme k vybraným uživatelským scénářům. I přesto, že uživatelské scénáře jsou poměrně malé části systému, tak si je tým rozdělí na ještě menší úkoly. Úkol může vypadat například jako přidání nového sloupce v databázi, přidání formulářových vstupů, návrh a implementace grafického rozhraní nebo funkční testování (Sutherland, 2014, strana). Product Owner je přítomen i na této části schůzky. Občas se totiž může stát, že týmu nejsou některé části uživatelského scénáře zcela jasné. Někdy může nastat situace, že po rozdělení uživatelského scénáře na dílčí úkony tým zjistí, že do plánu sprintu může přidat nebo naopak z něj musí odebrat další uživatelský scénář. Ačkoliv se tým snaží identifikovat všechny dílčí úkoly související s uživatelským scénářem již na plánovací schůzce, je možné že některé neobjeví. Poté co práce ve sprintu započne, jsou tyto úkoly objeveny a zařazeny do plánu sprintu. Tým ve svých odhadech s touto rezervou

počítá, a tak by nemělo být doručení plánu sprintu v ohrožení. Úspěšné scrumové týmy jsou schopny ve fázi plánování odhalit 60–70 % dílčích úkolů uživatelských scénářů. Poté co je sprint kompletně naplánován a Product Owner i tým s plánem souhlasí, je sprint zahájen.

3.2.2 Daily Scrum

Někdy je tato schůzka označována také stand-up meeting, z anglického stand-up = stát, protože členové týmu na schůzce stojí, aby se zamezilo zdlouhavému projednávání jednotlivých bodů. Tato schůzka je omezena patnácti minutami a každý člen týmu by měl odpovědět na tři jednoduché otázky:

- Co jsem udělal od našeho posledního Daily Scrumu?
- Co očekávám, že udělám do příštího Daily Scrumu?
- Jaké problémy mě blokují?

Tato schůzka má za účel informovat jednotlivé členy o vývoji práce celého týmu. Dává také možnost jednotlivým vývojářům požádat své kolegy o pomoc. Uzavření lidé mají občas problém přímo požádat některého ze svých kolegů o pomoc. Daily Scrum je díky třetí otázce nutí sdílet své problémy s celým týmem, a tak přijít na jejich řešení. Tato schůzka se koná, jak již název napovídá, denně. Metodika Scrum neurčuje přesný čas, ve kterém by se měla schůzka konat, toto rozhodnutí nechává na týmu podle jejich preference a zvyku. Je důležité tuto schůzku opakovat denně, aby se zamezilo tomu, že některé problémy nejsou řešeny, a tak by mohly ohrozit dodání sprintu včas. Daily Scrum by neměl sloužit k reportingu stavu sprintu pro manažery. Je určen přímo pro členy týmu a řešení jejich problémů.Za patnáct minut, které jsou pro schůzku určeny, není občas možné vyřešit problémy vážnějšího charakteru. Členové týmu se mohou domluvit, že se sejdou po Daily Scrumu a svůj problém vyřeší a nebudou tak zdržovat ostatní.

3.2.3 Backlog grooming

Celý tým a Product Owner se každý týden alespoň na jednu hodinu sejdou a diskutují o budoucích uživatelských scénářích, které jsou v plánu projektu. Product Owner týmu vysvětlí akceptační kritéria uživatelského scénáře. Pokud se týmu zdá uživatelský scénář příliš velký a obsáhlý, může požádat Product Ownera o jeho rozdělení do více uživatelských scénářů. Scrum říká, že by uživatelské scénáře měly

být co nejmenší, aby se minimalizovalo riziko neporozumění. Pokud se týmu zdá, že uživatelský scénář má akceptovatelnou velikost, společně odhadne jeho velikost a pracnost. Scrum konkrétně nepopisuje, jakým způsobem má tým scénář odhadovat. Mike Cohn (2006) ve své knize doporučuje tři přístupy, kterými se může tým řídit.

Odhadování v hodinách

Toto je tradiční přístup známý i z jiných metodik. Tým se vzájemně shodne na hodinové dotaci, kterou si myslí, že může daný uživatelský scénář trvat. Toto odhadování je velmi přirozené, ačkoliv v sobě skrývá úskalí. Tým se musí shodnout pouze na jednom čísle a v době odhadování neví, který konkrétní vývojář bude na scénáři pracovat. Časový odhad práce na uživatelském scénáři se může diametrálně lišit podle zkušeností jednotlivých členů týmu. Je rozdíl, pokud scénář odhaduje junior nebo senior. Tímto se součet těchto odhadů stává velmi nepřesným a subjektivním. Týmy tento problém řeší tím, že při plánování uživatelského scénáře do sprintu odhadnou každý dílčí úkol jednotlivě. Tento úkol je již přiřazen konkrétnímu vývojáři a odhaduje ho pouze on. Součtem všech úkolů ve sprintu se tým může ujistit, zda je plán na začátku splnitelný nebo ne. I během sprintu může díky kontinuálnímu odhadování kdokoliv a kdykoliv zjistit, v jakém stavu se sprint nachází. Tyto informace jsou vyjádřeny pomocí grafu Burndown Chart (graf úbytku práce) kterému bude věnována samostatná kapitola.

Odhadování v bodech

Tento přístup k odhadování uživatelských scénářů vyžaduje zkušenější scrumový tým a snaží se eliminovat problém odhadování v hodinách, kde je odhad velmi silně subjektivní a závislý na zkušenostech konkrétního vývojáře. Odhadování v bodech určí stupnici, ze které mohou vývojáři svoje odhady vybírat. Mike Cohn ve své knize doporučuje použít jako stupnici Fibonacciho posloupnost (1, 2, 3, 5, 8, 13, 21, 40 a otazník). Odhad již není subjektivní ale používá porovnávání uživatelských scénářů vzájemně mezi sebou. Z empirických zkušeností máme vzorové scénáře, které reprezentují každý bod na odhadovací stupnici, a vývojáři pouze srovnávají ke kterému vzorovému scénáři je prezentovaný scénář nejvíce podobný. V této disciplíně by měl junior i senior vybrat stejnou kategorii nehledě na čas, který by potřebovali k implementaci (Cohn, 2006, strana 301). Tým pak může sledovat, jaký počet bodů je

schopen za jeden sprint doručit, tedy zjistit svoji rychlost (velocity), kterou práce zmiňuje v předchozích kapitolách.

Odhadování v uživatelských scénářích

Může se zdát, že tento způsob odhadování ulehčuje práci scrumovým týmům. Pro odhadování totiž používá pouze počet uživatelských scénářů. Product Owner se snaží, aby všechny uživatelské scénáře byly stejně velké, takže tým může měřit svoji rychlost (velocity)pouze počtem ukončených scénářů v konkrétním sprintu. Tento způsob odhadování je určen pro velmi vyspělé týmy a pouze tehdy, pokud charakteristika uživatelských scénářů dovoluje jejich rozdělení na stejné nebo alespoň podobné jednotky. (Johnson, 2011).

Nehledě na to, jaký způsob odhadování si tým vybere, je důležité si uvědomit účel této činnosti. Záměrem odhadování je ověřit vzájemné pochopení požadavků všemi stranami. Pokud jeden vývojář odhaduje uživatelský scénář třinácti body a druhý dvěma, pak je nasnadě, že tým požadavky na funkcionalitu nepochopil a je potřeba je vysvětlit znovu nebo zodpovědět jejich otázky. Odhadování jednotlivých dílčích úkolů ve sprintu také slouží ke sledování stavu vývoje iterace a pokud se sprint dostane do zpoždění, může vývojový tým hledat společně s Product Ownerem kompromisní řešení.

3.2.4 Sprint Review

Na konci každého sprintu je potřeba předvést funkční inkrement produktu zákazníkovi. Na schůzku Sprint Review jsou pozvány všechny zainteresované osoby, které mají zájem vidět přírůstek nové funkčnosti (Shore, 2008). Tým by měl ukázat, které uživatelské scénáře byly na začátku sprintu naplánovány. Jeden po druhém pak předvést zákazníkovi na reálných příkladech z praxe. Uživatelské scénáře prezentuje vývojový tým sám, zde je také rozdíl oproti klasickým metodikám, ve kterých nové verze zákazníkům většinou předvádí obchodní zástupci firmy. Účelem této schůzky je získat zpětnou vazbu od zadavatele projektu. Úkolem Product Ownera je tuto zpětnou vazbu promítnout do budoucích plánů týmu tím, že jí v seznamu uživatelských scénářů přiřadí náležitou prioritu. Vývojový tým si může sám určit, zda chce uživatelský scénář prezentovat zákazníkovi, pokud není ve sprintu zcela dokončen, zde opět Scrum připouští, že i takováto situace může nastat, přece jen vývojáři jsou

pouze lidé (Baker, 2012, strana 272). Prezentace nedokončeného scénáře může mít přínos v podobě cenné zpětné vazby, která přijde v pravý čas. Scénář se bude pravděpodobně dokončovat v příštím sprintu a tým ve spolupráci s Product Ownerem může tuto zpětnou vazbu již zapracovat, místo aby čekal celou další iteraci na tuto zpětnou vazbu. Někdy ale může nastat situace, že se tým domluví, že rozpracovaný scénář prezentovat nebude, protože by prezentace přinesla zpětnou vazbu k problémům, o nichž tým stejně ví. Nezkušené scrumové týmy, kterým se nepovede dokončit ani jeden uživatelský scénář, mají tendenci tuto schůzku zrušit. Scrum je založen na transparentnosti procesů a komunikace, a proto by se tato schůzka konat měla. I když jediným bodem bude to, že tým bude zákazníka informovat o tom, že nic nebylo dokončeno. Tato schůzka by měla přinášet poučení pro každého účastníka, ať už z řad zákazníka nebo vývojového týmu (Sutherland, 2014). Pro zákazníka je navíc velmi motivující vidět svůj projekt po tak krátké době fungovat a u většiny zákazníků tato schůzka vzbuzuje důvěru v kompetence vývojového týmu. Ve vodopádovém modelu musel zákazník čekat na výsledek až skoro do konce celého projektu a často nebyl s výsledky spokojen. Díky častým schůzkám Sprint Review je zákazník kontinuálně informován o stavu projektu. Stává se jeho součástí a má možnost projekt komentovat a poskytovat zpětnou vazbu. (Johnson, 2011).

3.2.5 Retrospective

Metodika Scrum je založena na konstantním zkoumání, jak vývojový tým pracuje a jak by se mohl zlepšit. Pokračujme v paralele scrumového sprintu s tím atletickým. Atlet se na konci běhu zastaví a vydýchá se a začne se připravovat na nový sprint. Mezitím se může poradit s trenérem, prozkoumat video svého běhu a podle naměřených údajů svůj příští závod zaběhnout lépe, například po technické stránce nebo se může rozhodnout prodloužit krok. Tyto změny dělá proto, aby byl v příštím sprintu ještě rychlejší než v tom právě skončeném (Johnson, 2011). Vraťme se do prostředí metodiky Scrum, kde na konci každého sprintu se vývojový tým sejde a společně diskutuje, jak se právě skončený sprint povedl nebo nepovedl a jak tyto zkušenosti mohou ovlivnit sprint příští. Jako každá schůzka, tak i schůzka Retrospective, je ohraničena časovou dotací jedné až dvou hodin na každý týden sprintu. Retrospektiva se odehrává na konci každého sprintu, aby členové týmu mohli vyjádřit čerstvě nasbírané zkušenosti s průběhem sprintu. Během této schůzky má

každý člen týmu právo vyjádřit to, co mu na projektu vadí. Tato aktivita vytvoří například následující seznam problémů (Johnson, 2011, strana 288):

- Znovu jsme neměli na konci iteraci dostatečný čas vše stoprocentně otestovat.
- Integrace subsystému trvala déle a byla daleko náročnější, než jsme čekali.
- Zastaralý hardware zpomaluje kompilaci.

Scrum se však nespokojuje pouze se seznamem problémů, ale jde ještě o stupeň dál a ukládá týmu, aby se na schůzce Retrospective dohodl alespoň první krok, který napomůže řešení problému. Toto řešení může vzniknout z vášnivé debaty všech zúčastněných, ale vždy jeden člen, ne Scrum Master, by ho měl jasně definovat. První krok k řešení problému může vypadat následovně:

- Pořídit rychlejší hardware, hlavně pak pevný disk na kterém bude kompilace několikrát rychlejší.
- Přeřadit integrační story z osmibodové kategorie do kategorie třináctibodové.
- Zajistit stoprocentní kvalitu i za cenu, že se jiný uživatelský scénář nestihne.

Je pak úkolem Scrum Mastera, aby tyto akční body sledoval a připomínal je týmu během následujících sprintů, jelikož tým může mít tendenci dohodnuté věci zapomínat. (Baker, 2012, strana 344)

Schůzka Retrospective by neměla být pouze o hledání problémů, ale také o hledání pozitiv během vývoje. Tým by si měl neustále připomínat, které činnosti dělá dobře, popřípadě které problémy se mu již podařilo odstranit. Tato činnost by měla podporovat chuť měnit zaběhlé procesy a podporovat motivaci v týmu. (Schwaber, 2004).

Někdy bývá velmi obtížné přimět tým, aby se podělil o své dobré a špatné zkušenosti z proběhlého sprintu. Hlavně pak u začínajících agilních týmu, které byly zvyklé, že změna pochází především od nadřízených a že oni jakožto vývojáři nemají šanci proces nijak ovlivnit. Derby a Larsen ve své knize, která se věnuje pouze agilním retrospektivám, navrhují pro opravdové zapojení všech členů následující jednoduchá cvičení:

Starfish (hvězdice)

Každý člen vývojového týmu si vezme pět kartiček, na které napíše svůj názor na následující témata:

- začít dělat,
- přestat dělat,
- pokračovat v,
- více,
- méně.

Tyto názory jsou pak umístěné na společnou tabuli, kde reprezentují hvězdici všech pěti sekcí. Scrum Master jako moderátor diskuze vezme a přečte nahlas každou z kartiček s popisem, ve které sekci se nacházela. Toto čtení může probíhat i anonymně, pokud Scrum Master má za to, že v týmu je mezi členy až přehnaný respekt, který by bránil svobodnému vyjádření názoru. Tým o každém bodu z kartičky diskutuje a pokud shledá bod jako oprávněný, tak se snaží i přijít s návrhem na řešení. Toto řešení Scrum Master napíše na druhou stranu kartičky a vrátí na tabuli. Tým si na konci hry může vybrat jen určitý počet nebo třeba všechny, na které se bude příští sprint koncentrovat a které chce urgentně zlepšit. Důležité je i vyhodnocení na příštích retrospektivách, kdy se tým může k tabuli vracet a diskutovat, které návrhy byly splněny a které jsou stále problémem.

I appreciate (já oceňuji)

Lidé, hlavně pak v Čechách, dokážou být až přehnaně kritičtí a zapomínají na to, že i přes všechno špatné se udělalo i spousta dobrých a povedených věcí. Tato jednoduchá hra by měla podporovat právě hledání pozitivních věcí mezi členy týmu. Každý by měl na kartičku napsat větu ve formátu:

- *Já, oceňuji <jméno> za <důvod>.*

Tato jednoduchá a nenásilná forma nutí členy týmu se zamyslet, kdo pro ně v poslední iteraci udělal něco dobrého, co si zaslouží ocenění mezi ostatními členy týmu. Příklad:

- Já, oceňuji Janu za to, že mi pomohla opravit rozbitý build proces.
- Já, oceňuji Radka za to, že se zlepšil v přípravě uživatelských scénářů, které jsou nyní pochopitelnější.

- Já, oceňuji Honzu, že zachoval chladnou hlavu, když jsem byl na něj minulý týden nepříjemný.

Jednoduché, ale velmi účinné. Derby a Larsen ve své knize uvádí, že každá retrospektiva by měla pozitivními body začínat a končit, aby členové týmu byli méně frustrovaní z těch nepozitivních. (Derby a Larsen, 2006).

3.3 Artefakty Scrumu

Označení artefakt může mít pro mnohé týmy, které se Scrumem, začínají dost neblahý zvuk. Mohou si totiž vzpomenout na RUP (Rational Unified Process), který je složen z několika tisícovek různých artefaktů. Scrum staví především na své jednoduchosti a přehlednosti, a tak týmy neobtěžuje množstvím, někdy i zbytečných, artefaktů. Namísto toho jich definuje pouze 5. Tyto artefakty by měly podporovat jedno z hlavních hesel Scrumu, a to transparentnost. Díky omezenému množství artefaktů v nich není člen týmu ztracen. Naopak by měly být tak jednoduché a čitelné, že by je měli pochopit i lidé, kteří nejsou s týmem každý den. Zákazník, neplést s Product Ownerem, který současně realizuje několik projektů u různých týmů, by měl mít možnost z těchto artefaktů okamžitě zjistit, v jakém stavu je projekt a kam směřuje.

Scrum samozřejmě není dogmatický, a tak má každý tým možnost si artefakty upravit nebo přidat dle potřeby. Scrum jen upozorňuje na to, aby jich nebylo příliš mnoho, což by mohlo znepřehlednit jejich čitelnost. Metodika ve své originální verzi radí zřídit následující artefakty:

- Produktový Backlog
- Sprint Backlog
- Informační nástěnky
- Burn Charts
- Task Board

3.3.1 Produktový Backlog

Produktový Backlog je kumulativní seznam požadavků na systém. Pomocí uživatelských scénářů jsou tyto požadavky popsány a seřazeny podle přání Product Ownera. Tento seznam je v projektu nekonečně dlouhý, jelikož se může neustále

měnit, a každá role v týmu by měla mít možnost do tohoto seznamu své scénáře přidávat. Je pak na Product Ownerovi, který jediný může hýbat s pořadím (prioritou) scénářů, vybrat takové, které reflektují urgentní potřeby zákazníka. Scrum odmítá jasné definování požadavků na začátku projektu. Podobně je strukturován i Product Backlog. Nahoře jsou uživatelské scénáře, které jsou malé, jasně specifikované a odhadnuté. Čím se v dostáváme níž, tak se scénáře zvětšují, zobecňují a mají pouze orientační odhad. Postupem času, kdy se tyto scénáře dostanou výše (protože ty nahoře již tým naimplementuje), je tým rozdělí na menší, jasně definuje a přesněji odhadne. Většinou na konci Produktového Backlogu můžeme najít uživatelské scénáře, které mají velmi nízkou prioritu a pravděpodobně nikdy nedojde k jejich implementaci. Scrum přímo zakazuje plýtvat časem na detailnější specifikaci těchto scénářů. Tento prvek pochází z principů štíhlé výroby používané v japonských výrobních továrnách kde Scrum čerpal svoji inspiraci. Scrum je tak v některých svých částech podobný metodice Kanban a JIT (Just In Time). Produktový Backlog spravuje pouze Product Owner a on může rozhodnout, v jaké formě bude Produktový Backlog přístupný týmu. Může si vybrat, že se budou používat indexové kartičky nalepené na zdi, intranetový systém nebo cokoliv, co mu připadá dostatečně přehledné a čitelné pro tým.

3.3.2 Sprint Backlog

Sprint Backlog je seznam úkolů pro tým v konkrétním sprintu (Schwaber, 2004). Na rozdíl od Produktového Backlogu je tento seznam časově omezen délkou sprintu. Seznam zahrnuje všechny naplánované uživatelské scénáře a jejich dílčí úkoly, které se tým zavázal splnit na plánovací schůzce. Po skončení plánovací schůzky již Product Owner nemůže přidávat nebo substituovat jednotlivé naplánované scénáře. Toto zaručuje klid a možnost soustředění na práci. Tento základní prvek Scrumu umožňuje zákazníkovi, v zastoupení Product Ownera, měnit směr a priority projektu, ale pouze mezi jednotlivými iteracemi, a ne během nich. Pokud tým v průběhu sprintu zjistí, že si naplánoval moc nebo naopak málo uživatelských scénářů, může si s Product Ownerem vyjednat úpravu Sprint Backlogu v podobě přidání nebo odebrání scénáře. Tuto změnu může iniciovat pouze tým a nikdo jiný.

3.3.3 Informační nástěnky

Pokud vejdete do místnosti, ve které pracuje scrumový tým, většinou vás ohromí množství ručně napsaných indexových kartiček na zdi a kreslené grafy. Tyto informační nástěnky mají ihned odpovědět na otázku, v jakém stavu se sprint nachází a co zbývá udělat? Opět zde můžeme najít zřejmou inspiraci ve štíhlé výrobě, která pracuje s naprosto stejnými infomačními nástěnkami. Tyto artefakty samozřejmě mohou být přístupné i online, kde intranetový systém řeší jejich kalkulaci, řazení a zobrazování. Scrumové týmy v tomto ohledu bývají staromódní a Esther Derby ve své knize tvrdí, že vytváření těchto nástěnek společně, bez automatizace, pomáhá semknout tým dohromady, zlepšit vzájemnou komunikaci a zvýšit spolupráci. Uveďme si dva základní artefakty, které se mohou na informační nástěnce objevit:

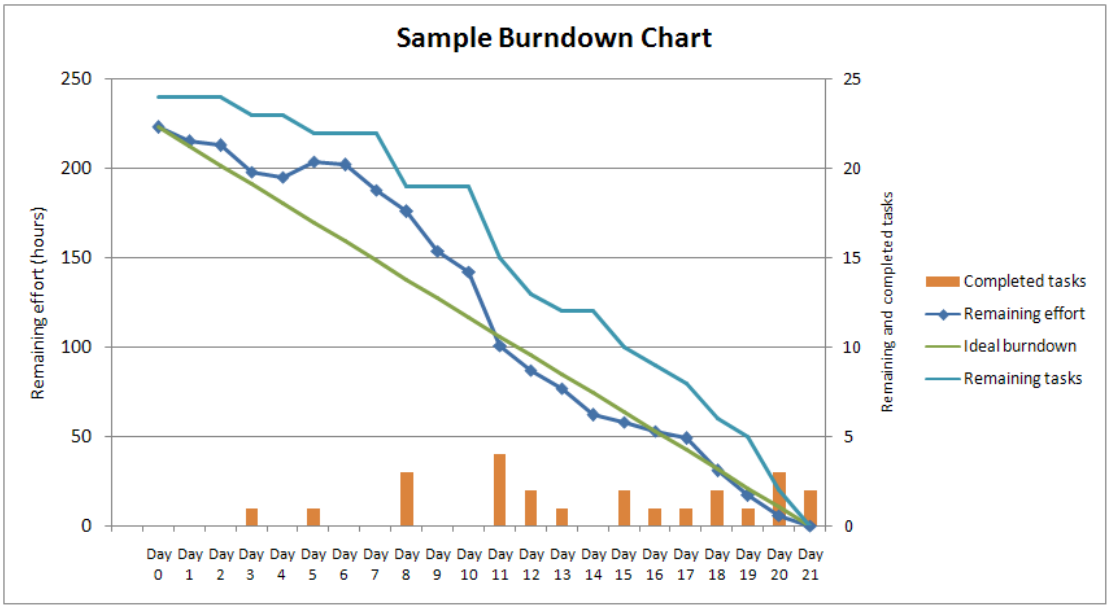
- Burn Charts,
- Task Board

3.3.4 Burn charts

„Pravděpodobně ta nejvíce sexy věc, kterou Scrum může nabídnout.“ – Kenneth Rubin (Rubin, 2012, strana 155)
„Burndown chart mi okamžitě dovoluje sledovat stav projektu.“ – Esther Derby (Derby, 2006, strana 209)

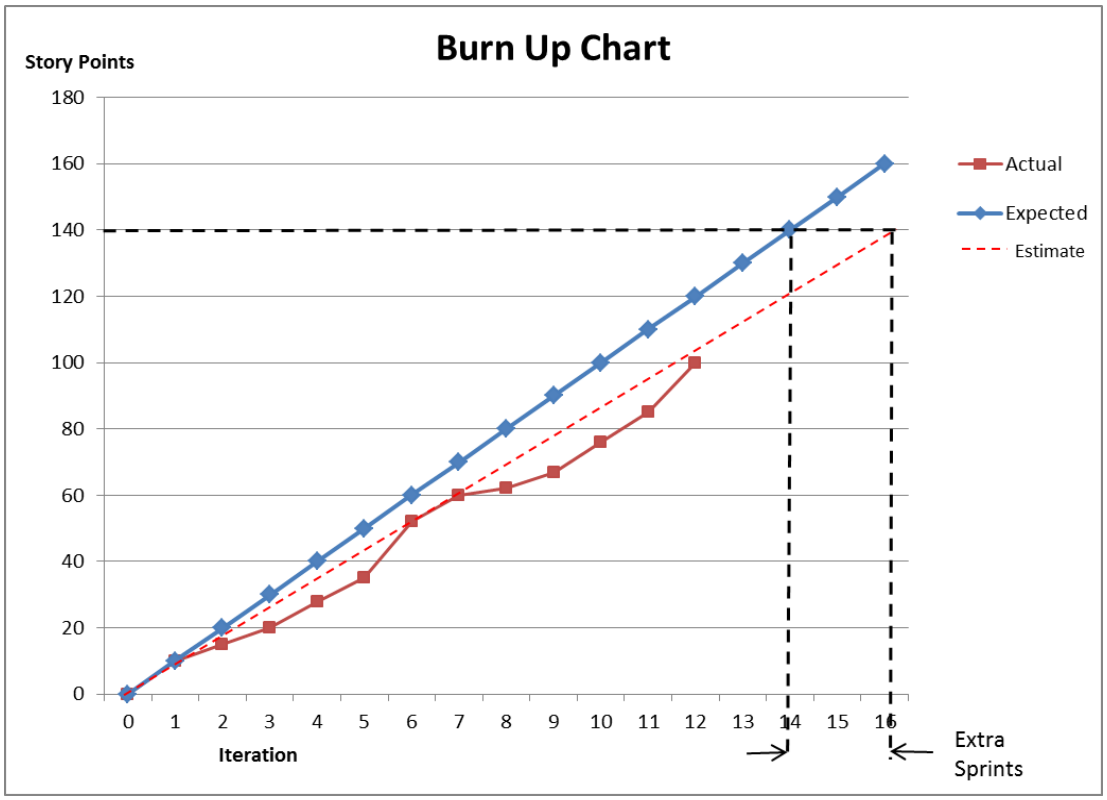
Burn chart, konkrétně v jeho variantě burndown chart, zobrazuje na liniovém grafu zbývající počet práce v každém časovém bodě sprintu. Protože je každá položka ve Sprint Backlogu odhadnutá díky odhadování dílčích odhadů, je na začátku iterace jednoduché zjistit celkový čas odhadovaný pro dokončení naplánované práce. Zbývající práce je vynesena na ose y. Každý den tuto zbývající práci může tým odhadovat na schůzce Daily Scrum a díky zaznamenávání do grafu sledovat průběh sprintu. Dny ve sprintu jsou vyneseny na ose x. Na liniovém grafu je také vedena přímka propojující začátek (suma všech odhadů na začátku sprintu) a konec (v poslední den sprintu by měla být zbývající práce nula) sprintu (Sutherland, 2014, strana 201). Díky aktuální pozici křivky průběhu práce může tým zjistit, zda naplánovaný sprint stíhá. Pokud se křivka aktuálního průběhu nachází pod křivkou ideálního průběhu sprint tým stíhá. Naopak nestíhá, pokud křivka aktuálního průběhu probíhá nad křivkou ideálního průběhu (Rubin, 2012, strana 95). Tým také může na

grafu burndown chart sledovat počet zbývajících scénářů nebo dílčích úkolů v určitém časovém bodě.



Obrázek 5 - Burndown chart

Burn chart má i svojí opačnou variantu burnup chart, který v jednom grafu označuje i celkový počet práce. (Goldstein, 2013, strana 305) Ve sprintu by se tento počet neměl často měnit, jelikož když je sprint naplánován, tak by se do něj práce již neměla přidávat. Pokud tým chce, může za pomoci těchto grafů sledovat i vývoj určité etapy nebo celého projektu. Burnup chart ukazuje dvě křivky na liniovém grafu. První ukazuje celkový počet naplánované práce a druhá celkový počet udělané práce. Na konci iterace by se měly obě křivky protnout. Dle rychlosti, jakou roste křivka udělané práce, můžeme predikovat, jestli tým projekt stíhá nebo ne.

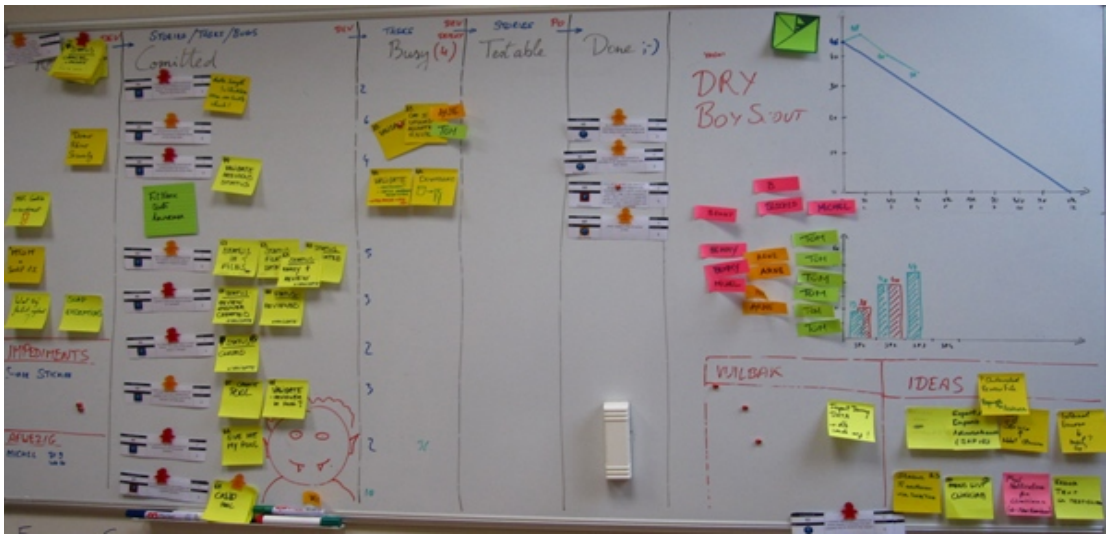


Obrázek 6–Burnup chart (zdroj: http://stephenwalther.com/archive/2012/08/17/scrum-in-5-minutes, online, cit. 2015-05-22)

3.3.5 Task Board

Všechny úkoly v projektu by měly mít v každém okamžiku určitý stav a ideálně přiřazenou osobu, která by tento úkol měla posunout dále v hierarchii stavů. Task Board by měl mít každý tým pro vizualizaci úkolů a seskupování dle jejich stavu. Tým, který se Scrumem začíná, může mít například následné stavy:

- Todo – Zbývá udělat.
- In Progress – Na úkolu se pracuje.
- Done – Hotovo.



Obrázek 7–Task Board (zdroj: http://stephenwalther.com/archive/2012/08/17/scrum-in-5-minutes, online, cit. 2015-05-22)

Zkušenější týmy mohou mít rozdílné workflow pro dílčí úkoly a pro uživatelské scénáře. Například mohou do svého procesu zahrnout stavy jako

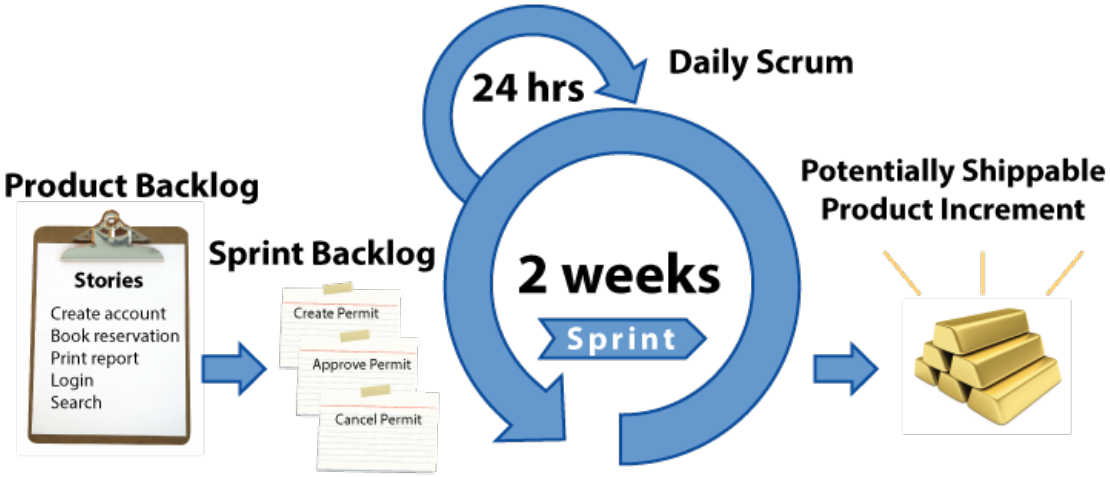
- Čeká na testování.
- Otestováno.
- Nasazeno.

Toto nastavení záleží pouze a jen na týmové domluvě. Task Board slouží k usnadnění práce týmu, viditelnosti a přehlednosti, nikoliv k byrokratickému procesu jako u jiných metodik. I zde se týmy velmi často uchylují k staré dobré tužce a papíru a Task Board si udržují nalepený na zdi. Kolem nástěnky Task Boardu se pak scházejí na svoje denní schůzky Daily Scrum. Obecně je lepší mít stav sprintu kdykoliv na očích a nemuset otevírat stránky na intranetu. (Derby, 2006, strana 209)

3.4 Scrum Proces

V předcházejících kapitolách jsme si představili jednotlivé role a artefakty v procesu Scrum. Obrázek 8 - Scrum Proces popisuje, jak vše zapadá do procesu jako celku. Na začátku vše začíná u Product Backlogu, jehož vlastníkem a správcem je Product Owner. Product Owner tento seznam uživatelských scénářů, které popisují funkčnost systému z pohledu uživatele, spravuje, prioritizuje jednotlivé scénáře mezi sebou a vybírá ty nejprioritněji z nich. Tento prioritní seznam pro časově omezenou iteraci tvoří Sprint Backlog. Velikost Sprint Backlogu pro daný sprint si určuje tým sám.

Zohledňuje možnosti a zkušenosti jednotlivých členů týmu a určí si pouze množství, které je reálné stihnout. Po dobu zvoleného časového úseku se každý den tým schází na schůzce Daily Scrum, aby zhodnotil vývoj sprintu a upravil svoji taktiku, jak dokončit všechny scénáře dohodnuté na začátku sprintu. Pokud nastane problém, konzultuje ho tým okamžitě s Product Ownerem a snaží se najít možný kompromis. Tým se skládá z erudovaných odborníků a je sebe-organizovaný a křížově funkční (tzv. „cross-functional“). Tedy dokáže se samostatně organizovat a zvládnout jakýkoliv úkol bez nutné externí pomoci.



Obrázek 8 - Scrum Proces

Na konci sprintu je výsledkem hotový, nasaditelný a otestovaný inkrement projektu. Tento výsledek prezentuje tým sám zákazníkovi a všem zainteresovaným lidem v projektu na schůzce Retrospective. Na této schůzce může kdokoliv průběh projektu komentovat a tím poskytovat zpětnou vazbu na funkcionalitu a použitelnost. Tyto body Product Owner přepíše do nových uživatelských scénářů, které zařadí a určuje priority v Product Backlogu. Tým se na konci sprintu schází za účelem zhodnocení sprintu po procesní stránce. Identifikuje, co tým dělal dobře a v čem měl rezervy a jak je zlepšit. Tyto body spravuje Scrum Master, který dohlíží na dodržování domluvených pravidel a postupů a upozorňuje členy týmu, pokud schází z dohodnuté cesty. Scrum Master také facilituje celý proces, moderuje diskuze a pomáhá řešit spory mezi členy týmu. Výsledkem scrumového procesu by měl být kvalitní a funkční produkt, spokojený zákazník, šťastní a motivovaní členové vývojového týmu.

4 Případová studie

Praktická část této diplomové práce je věnována optimalizaci procesů v konkrétní společnosti zabývající se vývojem softwaru. Zkoumaným objektem je softwarová firma X Software (jméno firmy bylo z důvodů smluvních vztahů autora práce a firmy změněno; dále jen firma), která vyvíjí a prodává informační systém zaměřený na správu požadavků velkých firem. Firma má své pobočky po celém světě a zaměstnává přes 250 zaměstnanců. Konkrétně se tato práce zabývá pražskou pobočkou, která se specializuje především na vývoj softwaru. Ostatní pobočky slouží spíše k obchodním účelům a žádný vývoj v nich neprobíhá. Centrála firmy je umístěna v německém městě Stuttgart. V této pobočce sídlí veškerý vyšší a produktový management, který komunikuje a zadává úkoly pražské pobočce. K firmě jsem se připojil v roce 2012 na pozici agilního kouče a mým hlavním úkolem bylo pomoci ve firmě rozšířit povědomí o agilních procesech a zavést jednotnou metodiku pro vývoj softwaru. Tento záměr bych chtěl, alespoň v hlavních bodech, zdokumentovat v této části práce, která tak prakticky naváže na informace v teoretické části. Na praktických ukázkách je zde znázorněno, jak může Scrum proces a hlavně pak agilní myšlení firmu zefektivnit. Odborné termíny již nejsou v této části práce vysvětlovány, neboť byly důkladně popsány v teoretické části práce.

Firma svůj produkt Y (dále jen produkt) vyvíjí již od roku 2005 a má více než stovku zákazníků. Rok od roku tržby a počet zákazníků stoupají a stejně tak se vyvíjí i počet zaměstnanců ve všech divizích firmy. Čím je více zákazníků, tím přibývá i požadavků na novou funkcionalitu. Firma si tuto poptávku uvědomuje¨, a tak investuje do rozšíření vývojového týmu nemalé finanční prostředky. Důsledkem toho je nárůst počtu zaměstnanců v pražské pobočce, která se vývojem zabývá. Ještě v roce 2009 byl vývoj postaven pouze na 10 zaměstnancích.

Firma má ve své pobočce v Praze 40 zaměstnanců, kteří se přímo podílejí na výrobě nebo podpoře vlajkového produktu. Tento produkt je prodáván všem koncovým zákazníkům ve stejné podobě a firma nabízí úpravu řešení přes svoji divizi Professional Services, která sídlí v Německu. Hlavním cílem autora bylo pomoci v organizaci práce a procesů hlavnímu vývojovému týmu, který společně

s produktovými manažery tvoří 24 zaměstnanců nebo kontraktorů. Pro potřeby této práce nebudeme na druh pracovní vztahu jednotlivých kontraktorů a zaměstnanců pohlížet s rozdílem a budeme všechny označovat jako zaměstnance firmy. Těchto 25 zaměstnanců bude pro zjednodušení v práci označeno jako vývojový tým společnosti. Jednotlivé kroky popsané v této práci se udály v časovém sledu tak, jak jsou v práci popsány, tedy od roku 2012 až do současnosti a o pohybu v čase vpřed bude čtenář informován.

4.1 Vývojový tým

Vývojový tým je tvořen 25 zaměstnanci firmy. Všichni zaměstnanci pracují na plný úvazek a vývoj probíhá ve standardních hodinách ve všední dny. Národnostní složení týmu je velmi pestré. Krom Čechů a Slováků je zde zastoupena většina východoevropských národností. Tým byl na začátku výzkumu, v únoru roku 2012, složen z následujících rolí

- 1x **Director of Prague R&D** – Ředitel pražské pobočky. Je zodpovědný za celých chod pobočky. Dohlíží na nábor a zaškolovací proces nových zaměstnanců. Dohlíží na projektové manažery, aby produkt postupoval kupředu a produktový tým v Německu byl s vývojem a výsledným produktem spokojen. Má na starosti rozpočet vývojového týmu a řeší spory, pokud se dvě strany nemohou dohodnout a je třeba je rozsoudit.
- 1x **Produktový/Projektový manažer** – Projektový manažer má za úkol komunikovat s německým produktovým oddělení a přetvářet jejich vágní požadavky na funkcionalitu na konkrétní a srozumitelné zadání pro vývojový tým. Projektový manažer také určuje, v jakých intervalech se bude produkt vydávat. Všechna tato rozhodnutí musí obhajovat v německé pobočce a před vyšším managementem.
- 1x **Quality Assurance manažer** – Manažer kvality. Je zodpovědný za výslednou kvalitu celého produktu. Koordinuje zbývající členy QA týmu. Občas se sám zapojuje do testování funkcionality.
- 2x **Senior Architekt** – Pomáhá projektovým manažerům s přetvářením vágního zadání na zadání srozumitelné vývojářům. Výsledkem jeho práce jsou databázové modely, funkční diagramy a třídní diagramy navrhovaného řešení problému.

- 4x **Senior Developer (starší vývojář)** – Vývojář, který má za úkol přetvářet zadání od architektů a projektových manažerů na výsledný produkt. Zaškoluje vývojáře juniory. Pokud má zájem, může být přizván k architektonickým řešením. 3 ze 4 starších vývojářů působí ve firmě více než 5 let.
- 9x **Junior Developer (mladší vývojář)** - Vývojář, který má za úkol přetvářet zadání od architektů a projektových manažerů na výsledný produkt. Průměrně má 2-3 roky zkušenost v oboru a technické vzdělání.
- 2x **UX Designer/HTML kodér** – Zodpovídá za návrh uživatelského rozhraní a vytváření HTML šablon. Produkt je webovým projektem. Spolupracuje s projektovými manažery a vytváří návrh pořadí a vzhled jednotlivých obrazovek systému. Nepodílí se na přípravě zadání.
- 1x **Senior Quality Assurance Engineer (starší QA)** – Tester, který kontroluje kvalitu výsledného produktu. Komunikuje s projektovými manažery ohledně očekávaného zadání a následně ověřuje jeho správnost. Jedná se o manuální black-box testování.
- 2x **Technical writer** – Má za úkol tvorbu projektové dokumentace. Když se projektoví manažeři dohodnou na konkrétní funkcionalitě, tak ji popíše, vytváří uživatelský manuál pro produkt.
- 2x **Junior Quality Assurance Engineer (mladší QA)** – Pomáhá s manuálním testováním. Provádí testovací scénáře navržené staršími testery (senior QA).

Vývojový tým je rozdělen na 2 samostatné dílčí týmy. Jeden tým tvoří Junior a Senior developeři a HTML kodéři , tým je označovaný jako „Vývoj“. Druhý označovaný jako „QA“ tvoří Senior a Junior Quality Assurance inženýři. Ostatní role ve vývojovém týmu pracují samostatně.

4.2 Seznámení se s prostředím

Firma nepoužívá pro svůj vývoj žádnou obecně známou vývojovou metodiku. Vzhledem k tomu, že firma prodává všem zákazníkům stejný software, tak požadavky na funkčnost přicházejí do pražské pobočky od produktového oddělení firmy sídlícího v Německu. Vedoucí pobočky v Praze nebyl schopný úplně přesně popsat autorovi práce, jakým způsobem je sběr požadavků od koncových zákazníků, kteří produkt opravdu používají, realizován. Firma vydává každý půl rok v únoru novou verzi

svého produktu. Vedoucí pobočky toto zdůvodňuje právě velikostí jednotlivých zákazníků (počtem uživatelů produktu), kdy v případě častějších vydání by rostly i náklady na instalaci a zaškolení. Tento seznam požadavků, který dorazí z Německa, je probrán s vývojovým týmem a ten po důkladné analýze, která trvá i několik týdnů, naplánuje, které z požadovaných funkčností je schopen do nového vydání dodat a které určitě nestihne. O prioritě těchto funkčností rozhoduje produktový tým v Německu. V průběhu roku se vedoucí pobočky a architekti pravidelně, každý měsíc schází s německým produktovým týmem a diskutují vývoj jednotlivých funkčností a pomocí Ganttova diagramu diskutují pokrok, případně zpoždění, projektu. Firma o sobě tvrdí, že je agilní a že dovoluje v průběhu měnit jednotlivé požadované funkčnosti. Tyto funkčnosti však může měnit v plánu pouze tehdy, pokud ještě nebyl započat vývoj na dané funkčnosti. Ředitel pobočky autorovi práce potvrdil, že za poslední 3 roky vývoje produktu se nastavený plán nestihl realizovat tak, jak byl naplánovaný a že toto byl jeden z hlavních důvodů, proč byl v roce 2010 ředitel pobočky odvolán a nahrazen novým. Jednotlivé funkčnosti se vyvíjejí dle priority, takže v případě zpoždění projektu se nedokončí ty, které mají prioritu nejnižší. Funkčnosti se velmi často dělají sekvenčně, takže problém dle ředitele pobočky nastane tehdy, pokud je před vydáním rozpracováno mnoho funkčností najednou a je nereálné je všechny dokončit. Pak nastane opravdový stres a musí se udělat mnoho kompromisů, aby alespoň nějaká funkčnost byla dokončena.

Jakmile je seznam funkčností přijat a schválen pražským týmem, tak nastane jejich detailní příprava a dělení do jednotlivých úkolů. Toto mají na starosti převážně projektoví manažeři, architekti a zkušenější vývojáři. Testeři zatím připravují pro jednotlivé funkčnosti testovací scénáře. Když je tato část přípravy dokončena, tak začíná samotná práce. Tým vývojářů postupně pracuje na jednotlivých úkolech. Tyto úkoly týmu přidělují projektoví manažeři. Ti také týmu vysvětlují zadání, pokud je nejasné. Architekti a projektoví manažeři detailně řídí každého vývojáře a konzultují s ním, jak a co bude dělat za úkol. Když je úkol nebo skupina úkolů dokončena, tak je nasazena na servery UAT (User Acceptance Testing, prostředí pro akceptační testy) a předána do QA týmu. QA tým prochází naplánovanou sadu, většinou manuálních, testů. Pokud je objevena závada, a to z 50 % je, tak je úkol vrácen zpět do vývojového týmu. Ředitel pobočky přiznává, že zde dochází k velkému zpožďování projektu. Pokud by testování prošlo na první pokus, tak se domnívá, že by funkčnosti dodali ve

stanoveném termínu, ale vzhledem k neustálému předávání úkolů mezi Vývojem a QA dochází k jeho prodlužování.

Každý měsíc se projektoví manažeři účastní schůzky s ředitelem pobočky a německým produktovým oddělením. Na této schůzce diskutují současný stav projektu, vyvinuté funkčnosti za minulý měsíc a plán na měsíc příští. Těmto měsíčním celkům říkají iterace. Ředitel pobočky potvrzuje, že čím více se blíží konec ročního vydání, tím se požadovaný plán iterací zvětšuje a někdy je naplánována iterace, která není reálně doručitelná. Ředitel pobočky přiznává, že hlavním důvodem nedodržování stanoveného plánu je rapidní nárůst zaměstnanců v pražské pobočce. Zákazníci i tržby rostou, a tak je ze strany managementu čím dál větší tlak na rozšiřování vývojového týmu. Tento tým se za poslední 3 roky rozrostl z 10 zaměstnanců na 24. Styl práce, který do té doby fungoval a byl založen především na vzájemné důvěře, s nárůstem zaměstnanců fungovat přestal a začaly se objevovat problémy v organizaci práce. Ředitel pobočky se také domnívá, že klesla morálka a motivace týmu, ačkoliv, nemá konkrétní metriku, která by toto potvrzovala. Dalším problémem je množství defektů, které jsou nahlašovány zákazníky. Za poslední dvě vydání produktu došlo k výraznému nárůstu těchto reportovaných defektů.

Autor při analýze situace věnoval velkou část svého času rozhovorům s týmem samotným. Při skupinových diskuzích i rozhovorech s jednotlivými zaměstnanci byly identifikovány tyto nedostatky:

- Plán je naplánován velmi formálně, a i přesto, že firma o sobě tvrdí že se řídí agilním myšlením, tak jakákoliv změna je zamítnuta produktovým oddělením.
- Komunikace v týmu je stále těžší. Tým je příliš velký.
- Nejsou domluveny praktiky spojené s vývojem – standardy kódu, automatické testování a kontinuální integrace.
- Mikro-management, tedy až přehnaně detailní řízení ze strany projektových manažerů a architektů. Žádná nebo velmi omezená možnost vlastní inovace.

Tato analýza současné situace vývojového týmu byla provedena v rozmezí tři měsíců. Autor pozoroval tým při práci a provedl individuální pohovory s jednotlivými členy

týmu. Také bylo provedeno několik týmových diskuzí, kde byla například využita „hra“ starfish, která je popsaná v teoretické části této práce.

4.3 Úkoly případové studie

Vývojová pobočka v Praze se dle pozorování autora práce potýkala s celou řadou méně či více závažných nedostatků, které by mohly být opraveny správným nastavením firemních procesů. Ve firmě chyběla formální definice metodiky vývoje a zodpovědná osoba, která by tento proces a metodiku udržovala aktuální. Firmu a zaměstnance velmi poznamenalo náhlé a nešetrné navýšení počtu vývojářů na pobočce. Autor této diplomové práce se rozhodl zaměřit se na tři konkrétní úkoly, navrhnout jejich možné řešení, představit řešení týmu, pokusit se ho implementovat a vyhodnotit výsledky.

4.3.1 Zefektivnění způsobu organizace práce a optimalizace procesů

Ředitel pobočky a většina vývojového týmu se shodla na nutnosti změnit způsob organizace práce. Ačkoliv se firma ve svých prohlášení prezentuje jako agilní, tak zaměstnancům samotným se agilní být nezdá. Autor práce si jako první úkol stanovil zefektivnění způsobu práce a optimalizace procesů. Konkrétně autor navrhl použití metodiky Scrum pro vývojový tým. Metodika Scrum nemůže být jednoduše představena a vynucována ze dne na den. Aby byl Scrum efektivní, tak musí být pozitivně akceptován samotnými zaměstnanci a samozřejmě i vedením, které musí upravit způsob zadávání požadavků projektu. Autor se tedy rozhodl, že bude implementaci Scrum provádět následovně:

- **Úvod do metodiky Scrum** – Autor v několika schůzkách vysvětlil zaměstnancům a vedení společnosti teoretické principy fungování metodiky Scrum.
- **Návrhy změn od samotných zaměstnanců** – Vzhledem k tomu, že v agilních metodikách nelze zaměstnancům diktovat, které změny pro zefektivňování procesů musí tým provést, je třeba dát prostor samotnému týmu, aby navrhoval změny v průběhu procesu na bázi demokratické volby. Díky tomu, že autor měl plnou podporu ředitele společnosti, nebyl mu tedy na úrovni managementu kladen odpor, který by mohl při nasazování agilních metodik

nastat. Obecná zkušenost doporučuje neměnit velké množství věcí od prvního dne, ale postupovat iterativně a měnit vždy jen malou část a tu vyhodnotit.

- **Implementace změn navržených zaměstnanci** – Autor práce působil v rámci projektu na pozici Scrum Mastera. Jeho hlavním úkolem bylo udržovat přehled o změnách a dohodnuté změny vyžadovat. Při implementaci metodiky Scrum se často můžeme setkat s tím, že se tým dohodne na změně na Retrospective schůzce, ale v průběhu iterace se vrátí do starých kolejí, protože je tak zvyklý. V každé iteraci se tým měl soustředit na implementaci změn navržených na předešlé Retrospective schůzce.
- **Vyhodnocení změn –** Na každé Retrospective schůzce by se měly vyhodnotit změny dohodnuté na schůzce předešlé a tým by měl odsouhlasit, jestli změna byla prospěšná a chce v ní pokračovat, a nebo nebyla a chce se vrátit k předešlému způsobu řešení.

Autor práce si uvědomuje, že podstatou metodiky Scrum je to, že procesy nebudou nikdy dokonalé a tým se neustále bude chtít zlepšovat. Pro potřeby této práce si autor nastavil období půl roku, ve kterém proces změny pozoroval a vyhodnocoval. Autor zvolil měsíční období pro navrhování a vyhodnocování změn (délka jednoho sprintu), proto je tato kapitola rozdělena do 6 podkapitol, které změny a vyhodnocení popisují.

4.3.2 Zlepšení spokojenosti zaměstnanců

Ředitel pobočky zmínil, že si myslí že, zaměstnanci dlouhodobě nejsou spokojeni se svojí prací a práce se pro ně stává rutinní a neinovativní. Proto se autor práce rozhodl během jednoho roku měřit spokojenost zaměstnanců s hypotézou, že úprava procesů bude mít pozitivní vliv na spokojenost zaměstnanců ve firmě. Měření bylo provedeno na začátku zkoumání, po šesti měsících a na konci zkoumání. Pro realizaci měření si autor práce vybral dotazník spokojenosti publikovaný Výzkumným ústavem práce a sociálních věcí (Výzkumný ústav práce a sociálních věcí, 2007, strana 6-9).Tento dotazník je rozdělen na dva okruhy. První sada je velmi jednoduchá a zkoumá pouze jednu otázku:

- Když zvážíte všechny okolnosti, jak jste celkově spokojen(a) se svou prací?

Druhá sada otázek již zkoumá dílčí aspekty pracovní spokojenosti. Pro účely tohoto výzkumu byl dotazník upraven tak, aby odpovídal jen na otázky, u kterých se dá předpokládat, že je úprava metodiky práce může změnit:

- Jste spokojen(a) se zajímavostí své práce?
- Jste spokojen(a) s délkou pracovní doby?
- Jste spokojen(a) s organizací pracovní doby?
- Jste spokojen(a) se vztahy s přímým nadřízeným?
- Jste spokojen(a) s pracovními podmínkami? (světlo, teplo, hluk)
- Jste spokojen(a) s pracovní zátěží?

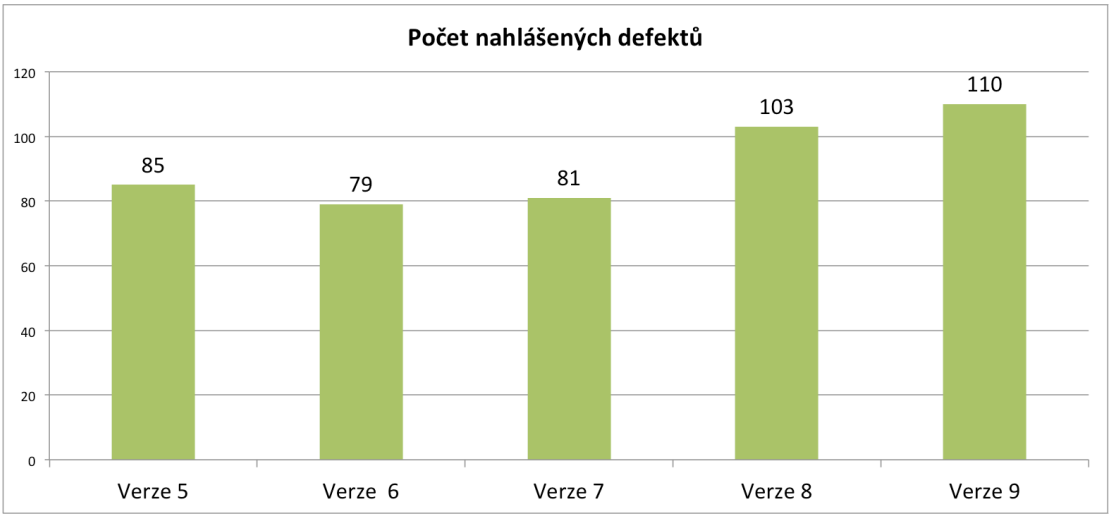
Na všechny otázky má respondent možnost odpovídat právě jednou z následujících možností:

- Rozhodně spokojen(a)
- Spíše spokojen(a)
- Ani spokojen(a), ani nespokojen(a)
- Spíše nespokojen(a)
- Rozhodně nespokojen(a)

Za spokojeného zaměstnance považujeme toho, který odpověděl na otázku jednou z možností rozhodně spokojen(a) nebo spíše spokojen(a).

4.3.3 Snížené množství defektů v budoucím vydání produktu

V dnešním vysoce konkurenčním prostředí je nepředstavitelné, aby produkt, který si zákazník zakoupil, nefungoval správně. Pokud se jedná o drobné chyby, které vyvažuje dobrá funkcionalita, tak je ještě možné být shovívavý. Pokud v produktu více věcí nefunguje než funguje, nebo nefungují jeho hlavní funkčnosti, tak se zákazník může rozhodnout přejít ke konkurenci. Proto je cílem zkoumané firmy velmi vysoká kvalita prodávaného produktu. Za poslední dvě vydání se však počet defektů nahlášených na oddělení péče o zákazníky zvyšoval. Dle ředitele společnosti byla tato metrika jedním z hlavních impulzů pro změnu organizace procesů ve vývojovém týmu. V tomto souhrnu jsou zahrnuty pouze defekty, které mají vážnost na deseti bodové stupnici (1 – nejméně vážný, 10 – nejvíce vážný) 5 a výše. Dalo se předpokládat, že pokud by byl produkt vyvíjen opravdu agilně, tak vzroste i kvalita produktu, což bude mít za následek pokles množství nahlášených defektů.



Graf 1 – Vývoj počtu nahlášených defektů – Začátek projektu (zdroj: autor práce)

Časový harmonogram projektu je nastaven tak, že nasazení metodiky Scrum probíhalo po dobu vývoje celého jednoho vydání, tedy půl roku, tak bylo možné změřit, kolik bude pro verzi 10 nahlášeno defektů a toto číslo vyhodnotit.

5 Práce s týmem

20. 3. 2012 – 23. 3. 2012 proběhlo praktické školení o metodice Scrum pro celý vývojový tým. Prezentaci, pomocí které představil autor práce agilní metodiky je možno najít na přiloženém CD. Na tomto školení byl celý tým seznámen s konceptem agilního myšlení, agilních metodik a konkrétně byla vysvětlena metodika Scrum. Celý jeden den byl věnován simulaci Scrum pomocí kostek Lego. Tento způsob simulace popsal Alexey Krivitsky ve svém projektu Scrum Simulation with LEGO. (http://www.lego4scrum.com/, online) Tým vytvoří menší scrumové týmy (zhruba 4–5 členů) a rozdělí si tři základní role: Scrum Master, Product Owner a zbytek členů je vývojový tým. Společnými silami se tým snaží vybudovat město postavené z kostek Lego. Zákazníka hraje agilní kouč a vymýšlí si zadání města. Jaké budovy chce postavit, kolik mají pater a také patřičně mění požadavky v průběhu jednotlivých sprintů, aby demonstrovaly prostředí reálného projektu. Tato simulace předpokládá, že účastnící mají alespoň teoretické znalosti Scrumu a zde si je mohou prakticky vyzkoušet. Vývoj města probíhá v jednotlivých iteracích, které počítají se všemi schůzkami jako v reálném procesu. Tým se akorát musí dohodnout, jak bude přistupovat k Daily Scrumu, který do iterace, které trvají kolem 30 minut jedna, není lehké vměstnat. Na plánovací schůzce tým řeší s Product Ownerem, které budovy z Product Backlogu postaví a které odloží na později. Také probíhají Sprint Review, kde tým prezentuje zákazníkovi, jakých výsledků ve sprintu dosáhl. Na Retrospective schůzce tým řeší jaký postup nebo nástroje jim mohou pomoci ve vývoji. Pokud se kouče, který je v roli moderátora, zeptají jestli jim může dát nástroj na rozebírání Lego kostek, tak jim ho moderátor dá. Pokud se nezeptají, tak musí kostky rozebírat ručně.

Je velmi důležité aby agilní myšlení a princip metodiky Scrum pochopil i management firmy. Pokud by totiž management nespolupracoval, tak byť by se tým snažil sebe víc, mohlo by vedení pořád očekávat doručení určité funkčnosti na konci cyklu projektu. Je důležité vysvětlit managementu principy agilního myšlení, protože v tomto směru musí agilní management respektovat rozhodnutí, které tým udělá, a to i v případě, že s nimi management nesouhlasí. Proto se autor práce rozhodl stejné

školení zopakovat, i s praktickou demonstrací za pomoci kostek Lego, v německém Stuttgartu, kde sídlí management firmy.

Oba týmy prezentace velmi zaujala a ohlas byl pozitivní. Byly zde zodpovězeny i otázky jednotlivých členů týmu a pražský produktový tým se rozhodl, že zkusí Scrum nasadit jako metodiku pro vývoj produktu. Tým se dohodl, že může kdykoliv Scrum zrušit, pokud se na tom shodne nadpoloviční většina členů.

5.1 Měření spokojenosti – Začátek projektu

„*Vysoká pracovní spokojenost významně ovlivňuje pro firmu tak zásadní jevy, jakými jsou fluktuace zaměstnanců či absentérství. Spokojení zaměstnanci vykazují obecně vyšší produktivitu práce než zaměstnanci nespokojení a zároveň jsou více motivováni k dobrému pracovnímu výkonu. Spokojení zaměstnanci bývají také k zaměstnavateli loajálnější.*" (Výzkumný ustav práce a sociálních věcí, 2007, strana)

Na začátku projektu byl provedeno počáteční měření spokojenosti zaměstnanců vývojového týmu. Výsledek toho testování měl sloužit jako výchozí bod pro budoucí srovnání se spokojeností po nasazení metodiky Scrum. Měření spokojenosti zaměstnanců je důležitá, a poměrně jednoduše, měřitelná metrika, kterou by měla každá firma sledovat. Může tím předejít případným problémům, které při nespokojenosti zaměstnanců mohou vzniknout, např. odchod klíčových zaměstnanců nebo nekvalitně odvedená práce. Tyto nedostatky mohou negativně ovlivnit vztah s existujícími nebo potencionálním klienty. Ve firmě doposud nebylo žádné měření spokojenosti prováděno.

Spokojenost byla měřena velmi jednoduchou otázkou:

- Když zvážíte všechny okolnosti, jak jste celkově spokojen(a) se svou prací?

Teoreticky by tato otázka stačila pro celé měření spokojenosti. Autor práce se domnívá, že pro potřeby správného směrování je dobré zjistit, v kterých konkrétních oblastech jsou zaměstnanci spokojení a v kterých nikoliv. Proto vybral i následující otázky:

- Jste spokojen(a) se zajímavostí své práce?

- Jste spokojen(a) s délkou pracovní doby?
- Jste spokojen(a) s organizací pracovní doby?
- Jste spokojen(a) s vztahy s přímým nadřízeným?
- Jste spokojen(a) s pracovními podmínkami? (světlo, teplo, hluk)
- Jste spokojen(a) s pracovní zátěží?

Na všechny otázky má respondent možnost odpovídat právě jednou z následujících možností:

- Rozhodně spokojen(a)
- Spíše spokojen(a)
- Ani spokojen(a), ani nespokojen(a)
- Spíše nespokojen(a)
- Rozhodně nespokojen(a)

n = 24	Rozhodně spokojen		Spíše spokojen		Ani spokojen ani nespokojen		Spíše nespokojen		Rozhodně nespokojen	
	Počet	%	Počet	%	Počet	%	Počet	%	Počet	%
Když zvážíte všechny okolnosti, jak jste celkově spokojen(a) se svou prací?	0	0%	2	8%	8	33%	12	50%	2	8%
Jste spokojen(a) se zajímavostí své práce?	0	0%	0	0%	15	63%	6	25%	3	13%
Jste spokojen(a) s délkou pracovní doby?	0	0%	7	29%	7	29%	10	42%	0	0%
Jste spokojen(a) s organizací pracovní doby?	0	0%	0	0%	14	58%	6	25%	4	17%
Jste spokojen(a) s vztahy s přímým nadřízeným?	0	0%	4	17%	9	38%	6	25%	5	21%
Jste spokojen(a) s pracovními podmínkami? (světlo, teplo, hluk)	12	50%	8	33%	3	13%	1	4%	0	0%
Jste spokojen(a) s pracovní zátěží?	5	21%	10	42%	7	29%	2	8%	0	0%

Tabulka 1 – Výsledky měření spokojenosti - Začátek projektu (zdroj: autor práce)

Hodnocení spokojenosti bylo realizováno za pomoci anonymního internetového dotazníku. Každý člen týmu byl požádán o vyplnění. Ačkoliv vyplnění nebylo povinné, tak se zúčastnilo všech 24 zaměstnanců pražské pobočky. Po hodnocení spokojenosti byl tým požádán o krátké hodnocení výsledků, které byly týmu prezentovány vždy jako konkrétní počet hlasů pro každou možnost a procentuální zastoupení. Tým měl za úkol pro každou z doplňujících otázek identifikovat, jaké

kroky by měla firma podniknout, aby byl výsledek při příštím měření lepší. Tyto body byly k nahlédnutí při všech Retrospective schůzkách v průběhu implementace metodiky Scrum.

Celkové výsledky hodnocení spokojenosti zaměstnanců dopadly dle očekávání ředitele pobočky. Pouze 8 % respondentů je celkově spokojeno (rozhodně spokojeno nebo spíše spokojeno, dále jen spokojeno) se svou prací, 33 % respondentů je v této otázce neutrální a nezaujímá žádné stanovisko, 58% zaměstnanců je celkově nespokojeno (rozhodně nespokojeno nebo spíše nespokojeno, dále jen nespokojeno) se svou prací. Tento klíčový faktor je velmi nízký a určitě je potřeba pracovat na jeho zlepšení. Konkrétní kroky tým navrhoval pouze pro dílčí otázky dotazníku, protože tato byla příliš obecná.

Výzkum poukázal na to, že zaměstnanci jsou spokojeni pouze s pracovními podmínkami. Firma si pronajímala moderní kanceláře v centru Prahy, a tak výsledky této otázky se daly předpovídat. Zaměstnanci by pouze uvítali, pokud by si mohli regulovat nastavení klimatizace.

Na otázku: „Jste spokojen(a) se zajímavostí své práce?“ odpovídala většina zaměstnanců neutrálně. 63 % respondentů uvedlo, že není ani spokojeno ani nespokojeno, 38% zaměstnanců bylo se zajímavostí své práce nespokojeno. Jako hlavní návrhy na zlepšení zaznělo větší zapojení vývojářů do rozhodování o tom, jak se budou konkrétní funkčnosti implementovat. Tehdy mohli funkčnost upravovat pouze architekti a projektoví manažeři na začátku projektu v analytické a přípravné části.

Výzkum identifikoval, že zaměstnanci nejsou zcela spokojeni s délkou pracovní doby. Pouze 29 % zaměstnanců bylo spokojeno, 42 % nespokojeno. Firma pravidelně žádá zaměstnance, aby pracovali přesčas, hlavně pak pokud se blíží vydání nové verze a tým nestíhá slíbené termíny. Tyto přesčasy jsou prezentovány jako dobrovolné, ale většina zaměstnanců má pocit, že je k těmto přesčasům nucena vedením. Pro růst spokojenosti v této oblasti by přesčasy neměly být tak často nařizovány.

Otázka: „Jste spokojen(a) s organizací pracovní doby?“ se setkala s neutrálním až negativním hodnocením. 58 % respondentů se nepřiklonilo ani k jedné straně. Bohužel zbylých 42 % účastníků bylo s organizací práce nespokojeno. Zaměstnanci navrhovali jako řešení menší míru detailního plánování a větší volnosti v rozhodování, jak budou některé úkoly provedeny.

Otázka: „Jste spokojen(a) s vztahy s přímým nadřízeným?“ byla vyhodnocena následovně. 17 % respondentů bylo spokojeno se svým nadřízeným, 38 % má neutrální postoj a 46 % je nespokojeno se vztahy s nadřízenými. Ředitel pobočky, projektový manažer a architekti mají dle respondentů vždy pravdu, a to občas i tehdy, pokud ji nemají.

Většina zaměstnanců je spokojena s pracovní zátěží. 63 % je spokojeno a pouze 8 % je nespokojeno. Zaměstnanci na toto téma neměli žádné konkrétní návrhy na zlepšení situace.

5.2 Organizace práce

Ihned po školení metodiky Scrum se tým rozhodl metodiku používat pro vývoj produktu, dokonce i projektový manažer a ředitel pobočky byli pro návrh implementace metodiky Scrum. Tým se rozhodl pracovat v pozměněných týmech než doposud. Metodika Scrum říká, že tým musí být schopen společnými silami doručit všechny disciplíny, které projekt vyžaduje. Tým se tedy rozhodl udělat dva scrumové týmy, které obsahují všechny role:

- Do role **Scrum Mastera** byl pro oba týmy jmenován autor práce. Původně autor práce zamýšlel, že Scrum Masterem bude ředitel pobočky, ten však pro nedostatek času odmítl. Ředitel pobočky se totiž musí starat i o ostatní zaměstnance v Praze, jako jsou například členové zákaznické podpory a marketingu.
- Roli **Product Ownera** měl pro oba týmy zastávat projektový manažer. Zde je potřeba zmínit to, že tým si s produktovým oddělení v Německu dohodl to, že budou navzájem ctít principy agilního plánování. Tedy že Product Owner bude zákazníka reprezentovat pro účely týmu a produktové oddělení bude

samotným zákazníkem. Společně vytvoří Produktový Backlog, který bude Product Owner udržovat. Na začátku projektu se do Product Backlogu umístí všechny nápady, které má v současnosti produktové oddělení připraveno. Odpadá tím fáze složitého plánování a analyzování, která by jinak musela proběhnout pro přípravu implementace této verze produktu. Produktové oddělení je stále zodpovědné za reprezentaci opravdových koncových zákazníků. Díky Product Backlogu ale může daleko obratněji určovat priority funkčností dle aktuálního přání zákazníků.

- Vývojový **tým** se rozdělil do dvou týmů. Tentokrát ne dle jejich expertiz na Vývoj a QA, ale rovnoměrně si rozdělil všechny role v týmu. V každém týmu byl 1x architekt, 2x starší vývojář, 4x mladší vývojář, 1x HTML kodér, 1x starší QA (QA manažer týmu pomáhal už jen přímo při ověřování kvality) a 1x mladší QA. Takto sestavený tým by měl mít všechny schopnosti a disciplíny potřebné pro kompletní doručení funkčnosti. Do této chvíli byl tým rozdělen do 3 oddělených místností. Tým se dohodl, že aby zlepšil vzájemnou komunikaci, tak každý tým bude sedět v jedné místnosti společně, kde budou umístěny všechny informační nástěnky jako Task Board a diagram Burndown chart.

Tým se rozhodl respektovat všechny schůzky, které Scrum definuje, tedy:

- Sprint Planning Meeting
- Daily Scrum
- Backlog
- Sprint Review
- Retrospective

Největší diskuze týmu byla o nutnosti Daily Scrumu v tak četném počtu opakování za sprint, tedy každý den. Tým navrhoval, že by tyto schůzky stačily pouze 2x do týdne. Dosud se tým scházel jen 1x týdně ve středu. Scrum Master navrhl dodržet tyto schůzky tak, jak je předepisuje Scrum s tím, že jejich četnost může být upravena po skončení první iterace, pokud se schůzky ukážou jako neúčinné. Hlavní obava týmu byla v tom, že byl zvyklý, že původní pravidelná schůzka trvala vždy až několik hodin a byla velmi vyčerpávající. Scrum Master představil termín „time-boxing“ tedy

ohraničování schůzek jasně vymezeným časem. Tato technika je používána přesně z tohoto důvodu, aby schůzky nebyly přítěží.

Aby mohla začít první iterace připravil Product Owner společně se Scrum Masterem uživatelské scénáře. Společně s týmem strávil několik hodin nad rozdělováním funkčností, které by si zákazník, německé produktové oddělení, přál. Připravili pouze tolik scénářů, kolik se zhruba vešlo do první iterace. Tým velmi ocenil, že nemusel strávit přípravou několik týdnů jako v minulosti.

Tým se rozhodl pro zjednodušení používat odhad v hodinách, s tím že ho velmi zaujal způsob odhadování v bodech a chtěl tyto odhady vyzkoušet. Hlavní změnou v odhadování týmu je to, že již práci neodhaduje projektový manažer a architekti, ale samotní členové týmu. Tým použil pro odhadování uživatelských scénářů pro první sprint metody takzvaného odhadovacího pokeru. Myšlenka této praktiky spočívá v tom, že Product Owner týmu prezentuje uživatelský scénář a jeho akceptační kritéria, tedy seznam požadavků, které musí fungovat, aby scénář mohl být akceptován Product Ownerem jako dokončený. Tým se může v průběhu prezentace ptát na detaily, které potřebuje vědět k odhadu pracnosti uživatelského scénáře. Když Product Owner dokončí prezentaci a tým nemá další otázky, tak každý člen týmu napíše na kartičku svůj odhad pracnosti pro daný uživatelský scénář. Důležité je, aby jednotliví členové týmu od sebe neopisovali. Tím se zaručí, že každý člen týmu dostane právo se vyjádřit k odhadu. Tišší jedinci totiž mohou být v hromadném odhadování opomíjeni. Na pokyn Scrum Mastera celý tým najednou otočí své index kartičky a diskutuje o výsledcích. Tým by se měl shodnout na jednom společném odhadu. Odhadování v hodinách má nevýhodu, již popisovanou v teoretické části této práce, že potřebný čas velmi závisí na konkrétním vývojáři, který bude úkol implementovat. Proto Scrum doporučuje odhadování v bodech a nikoliv v hodinách. Při odhadování v hodinách se tým musí dobře znát a vybrat průměrný odhad, který se v množství uživatelských scénářů implementovaných různě zkušenými vývojáři zprůměruje. Stejně tak si tým sám plánuje množství uživatelských scénářů, které má v plánu v iteraci dokončit. Předtím tento seznam neexistoval a tým věděl jen velmi nepřesně, co se očekává, že doručí.

5.3 Sprint 1

První iterace byla naplánována v délce jednoho měsíce a začala plánovací schůzkou. Tým znal uživatelské scénáře velmi dobře z jejich přípravy a odhadování, a tak byla schůzka velmi rychlá. Jedinou otázkou bylo množství, které by tým měl naplánovat. Na doporučení Scrum Mastera tým jednoduše sečetl svoji kapacitu, kterou mohl maximálně věnovat vývoji a testování uživatelských scénářů. Kapacita byla započítána pro všechny členy nově vzniklého týmu, tedy i pro junior a senior QA členy týmu. Každý člen týmu dostal koeficient výkonnosti dle své zkušenosti s vývojem. Starší vývojáři a QA si určili koeficient 1. Mladší vývojáři a QA měli koeficient od 0,5 do 0,7 dle své výkonnosti. Tento koeficient vyrovnává nepřesnost při odhadování v hodinách, která vzniká tím, že se celý tým musí dohodnout na konkrétním odhadu v hodinách. Tímto koeficientem byl vynásoben počet dní, které budou trávit vývojáři v práci, tedy nezapočítávají si dovolené a případně účasti na konferencích nebo školení. Poté Scrum Master správně připomněl, že tým nemůže počítat s tím že se bude 100 % svého času věnovat pouze vývoji. Tým musí započítat různé schůzky, které má na sprint naplánovány. Tým si určil, že se vývoji bude věnovat pouze 80 % možného času, takže vynásobením dospěl ke své plánované velocity, tedy metrice rychlosti, která týmu bude pomáhat při plánování sprintu. Tým samotný totiž určí, kdy je plán plný a již nemůže přidat do Sprint Backlogu další uživatelský scénář. Tým během plánovací schůzky pro první sprint naplánoval uživatelské scénáře v celkovém součtu odhadů 125 dní pro první tým a 122 pro druhý tým.

Plán pro první sprint je vždy naplánován pouze orientačně. Na konci se totiž ukáže, kolik uživatelských scénářů, respektive součet jejich odhadů, může tým dokončit. Scrum pak s touto hodnotu pracuje jako s orientační metrikou pro plánování dalšího sprintu. Jak sprinty pokračují, tak pokud tým vezme průměrnou hodnotu odhadů uživatelských scénářů, které na konci sprintu doručí, tak dostává velmi cennou metriku, která může být použita jednak pro plánování konkrétního sprintu, tak i pro odhad, kam se tým dostane s vývojem během několika následujících sprintů. Tato informace slouží však pouze jako odhad, nikoliv závazek.

5.3.1 Vyhodnocení sprintu

Vyhodnocení sprintu musíme porovnávat ze dvou úhlů pohledů – kolik uživatelských scénářů se povedlo doručit a jak se týmu pracovalo. S ohledem na to, že to byl první Scrum sprint týmu, tak se povedlo doručit 68 % naplánovaných uživatelských scénářů v týmu 1 a 59 % v týmu 2. Tým na společné Sprint Review schůzce prezentoval hotové uživatelské scénáře Product Ownerovi. Na schůzce byli přítomni i reprezentanti z produktového oddělení z Německa a dokonce i ředitel firmy. Velmi ocenili, že mohou ihned po prezentování uživatelského scénáře sdělovat svoji zpětnou vazbu na uživatelský scénář a pohovořit si o funkčnosti přímo s vývojovým týmem, který měl také řadu nápadů na zlepšení produktu. Na Retrospective schůzce tým hodnotil uplynulý sprint. Tým velmi pozitivně hodnotil volnost, kterou měl na rozdíl od předešlého systému vývoje, mohl si sám určit množství naplánované práce a také sám mohl rozhodnout, jakým způsobem bude postupovat při jeho řešení. Tým při Daily Scrumu dost často pokládal architektovi otázky, jak má být konkrétní uživatelský scénář naimplementován. Tento postup je považován za správný, tedy že tým o problému komunikuje a demokraticky rozhodne o nejlepším řešení místo toho, aby mu bylo nařízeno direktivně, jaký přístup má zvolit. Tým také hodnotil, z jakého důvodu nedokončil více než 30 % naplánované práce. Největší změna dle týmu bylo sloučení vývoje a ověřování kvality. Doposud byly tyto procesy oddělené, takže vývoj skončil tehdy, když vývojáři usoudili, že je funkčnost hotová. Následovala pauza, kdy QA testovalo jiné funkčnosti, a tak se tato konkrétní funkčnost dostala na testování až za několik dní. Často se stalo, že funkčnost byla vývoji vrácena z QA týmu, ale toto se již bralo jako nový úkol. Nyní bylo potřeba uživatelský scénář jak vyvinout, tak plně otestovat. Toto činilo týmu velký problém, jelikož předávky mezi vývojem a testováním byly okamžité. Tým seděl v jedné místnosti a vývojář mohl přímo přijít za testerem a požádat ho o jeho názor, popřípadě přímo o testování. Tým se dohodl, že pro zvýšení kvality produktu musí každý dílčí úkol a uživatelský scénář zkontrolovat další člen týmu. Scrum definuje, že tuto kontrolu by měl možnost udělat každý další člen týmu. Tým se shodl, že bude nejlepší, pokud kontrolu budou dělat členové týmu z původního QA týmu. Tým identifikoval jako největší problém situaci, kdy vývojář zodpovědný za implementaci uživatelského scénáře má dojem, že je hotový, ale přitom chybí mnoho drobností. Scrum Master doporučil týmu během příštího sprintu vytvořit dokument, který se v agilních metodikách nazývá Definition of Done (DoD).

DoD je seznam jednotlivých vlastností, které uživatelský scénář musí splňovat, aby mohl být považován za dokončený. Tento seznam musí být ve formátu kontrolního seznamu, kde si vývojář jednoduše může zkontrolovat, že jeho úkol obsahuje všechny náležitosti, tím se předejde zbytečné ztrátě času při předávkách. Tým společně souhlasil, že během příštího sprintu společně sepíše takovýto seznam, který bude sloužit ke snížení počtu předání mezi vývojem a testováním.

Během sprintu byl každý den dodržován Daily Scrum. Bylo pozitivní sledovat, jak tým o problémech diskutuje a snaží se je řešit. Tým si také během sprintu s Product Ownerem domluvil, že každý týden se na hodinu a půl sejde za účelem odhadování a diskuze o Product Backlogu, který Product Owner po celý sprint pečlivě připravoval.

5.4 Sprint 2

Tým při plánování druhého sprintu postupoval obezřetněji než na minulé plánovací schůzce. Měl v živé paměti ten nepříjemný pocit, kdy musel prezentovat hotové uživatelské scénáře před ředitelem společnosti a na konci schůzky musel přiznat, že více než 30 % naplánovaných uživatelských scénářů nestihl doručit. Tým se již na doporučení Scrum Mastera nezabýval počítáním přesné kapacity. Scrum doporučuje použití metriky průměrné velocity pro plánování sprintu. Toto samozřejmě platí pouze tehdy, pokud tým není nějak výrazně ovlivněn nemocemi nebo dovolenými zaměstnanců. Pro plánování sprintu použily oba týmy dosaženou velocity z minulého sprintu. Tým 1 velocity 97,5 dnů a tým 83 dnů. Na konci plánovací schůzky pro tým 2 se Product Ownerovi zdál seznam naplánovaných uživatelských scénářů malý, a tak se pokusil dle zažitých zvyků do plánu ještě jeden uživatelský scénář přidat. Scrum Master včas vše zastavil a vysvětlil Product Ownerovi, že tým již řekl, že Sprint Backlog je naplněn, a tak nemůže přijmout další scénář. Také mu vysvětlil, že pokud tým dokončí práci Sprint Backlogu dříve, tak si vždy může přidat další uživatelský scénář do plánu. Daleko těžší je scénář odebrat a je velmi nepříjemné na Sprint Review schůzce prezentovat, že se scénář nepodařilo stihnout. Product Owner souhlasil a oba týmy své plány přijaly. Důležité je také zmínit, že Scrum Master s týmem absolvoval i druhou část plánovací schůzky, kdy jednotlivé týmy procházejí Scrum Backlog a uživatelské scénáře dělí na dílčí úkoly. Tyto úkoly si pak tým

rozdělí mezi své členy a domluví si strategický postup, na které úkoly se zaměří jako první a které nechá na konec sprintu.

Během tohoto sprintu tým opět dodržoval pravidelné denní Daily Scrum schůzky. Scrum Master při těchto schůzkách začal pozorovat to, že tým přesahuje stanovený čas 15 minut na schůzku. Jeden den schůzka trvala 30 minut, tedy dvojnásobek povolené doby. Scrum Master se rozhodl ihned po schůzce s týmem konzultovat, proč se tak děje. Toto může být nevýhoda takto dlouhých sprintů. Při kratších sprintech by Scrum Master počkal do konce iterace a s týmem vše vyřešil až na Retrospective schůzce. Do Retrospective schůzky v tomto případě zbývalo 14 dní, a tak se Scrum Master s týmem rozhodl problém vyřešit ihned. Tým byl upozorněn, že přesahuje stanovenou dobu schůzky a zda-li může někdo identifikovat, proč se tak děje. Tým zjistil, že občas malá skupinka vývojářů na Daily Scrum řeší problém, který se týká pouze jich, a zabíhá do detailních technických detailů, ale nijak nepodporuje informovanost týmu. Proto Scrum Master navrhl, zda se při takových to problémech může skupina vývojářů domluvit a dořešit problém až po schůzce a tím nezdržovat ostatní. Na Daily Scrumu by se neměly řešit konkrétní technické detaily. Tím, že tým problém řeší v jedné místnosti, nepřichází ostatní o možnost zapojit se do diskuze, pokud si myslí, že mohou nabídnout nějaké řešení.

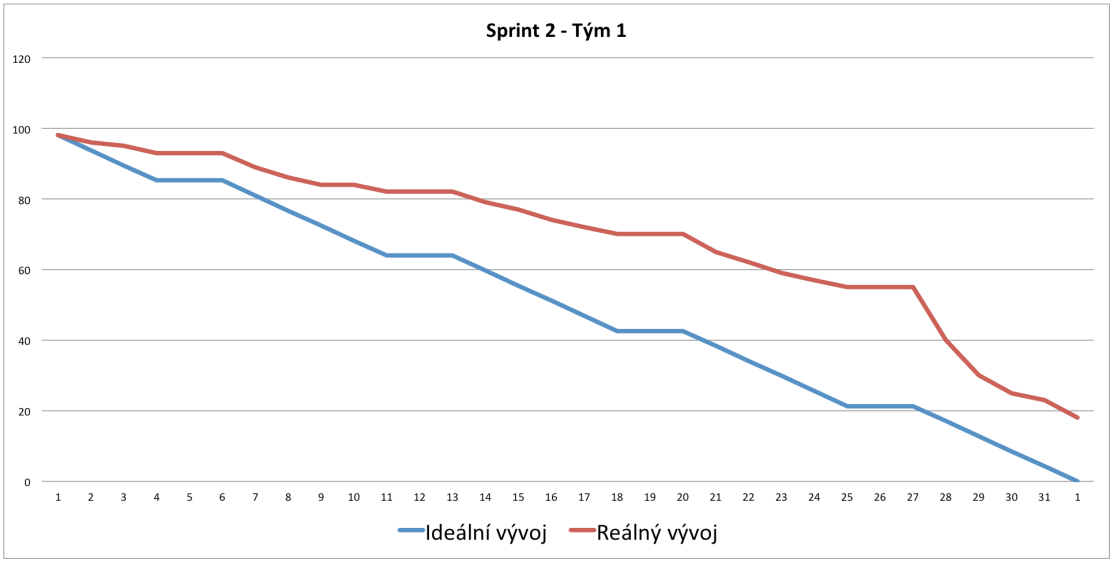
Tým se také rozhodl, že v tomto sprintu se třikrát sejde na maximálně jednu hodinu za účelem diskuze DoD seznamu, který si slíbil vytvořit na poslední Retrospective schůzce. Tým během diskuze identifikoval následující body, které každý vývojář musí projít dříve, než označí uživatelský scénář za hotový. Stejně tak tento seznam ještě musí projít pro každý uživatelský scénář další člen týmu z důvodu verifikace a zlepšování kvality:

- Všechny akceptační kritéria jsou splněna.
- Dílčí úkoly jsou otestovány automatickými Unit testy.
- Uživatelský scénář funguje ve všech podporovaných prohlížečích.
- Integrační testy nejsou chybové a běžely alespoň jednou po dokončení.
- Vývojář demonstroval funkčnost Product Ownerovi.
- Kód je nahrán do repositáře SVN.
- Kód odpovídá smluveným standardům (řádkování, odsazení, komentáře).

Tento seznam byl vytištěn pro každého člena týmu a umístěn na jeho stůl, aby ho měl pořád na očích.

5.4.1 Vyhodnocení sprintu

Tento sprint dopadl o něco lépe než sprint předešlý. Tým 1 splnil 81 % procent ze svých naplánovaných uživatelských scénářů. Tým 2 jich splnil 84 %. Tým vypracoval DoD seznam, který začal v tomto sprintu používat. Vzhledem k tomu, že seznam byl dokončen až těsně před koncem sprintu, bylo na jeho hodnocení brzo a tým se shodl, že bude jeho účinnost vyhodnocovat až na konci příštího sprintu. Tým se při Backlog Grooming schůzce shodl, že uživatelské scénáře připravené od Product Ownera jsou příliš velké a komplexní a je velmi těžké je odhadovat. Scrum Master zkoumal důvody, proč Burndown chart na konci sprintu měl tak strmý pád namísto toho, aby klesal rovnoměrně. Tým problém vysvětloval tím, že bylo těžké zcela dokončit jednotlivé uživatelské scénáře, jelikož byly obsáhlé a komplexní. Tým 1 si na začátku sprintu naplánoval doručit 11 uživatelských scénářů a na konci sprintu jich doručil 9. Tým navrhl Product Ownerovi, aby uživatelské scénáře pro příští sprint rozdělil na více menších a jednodušších uživatelských scénářů. Product Owner souhlasil a slíbil, že se pokusí do začátku příští iterace ještě uživatelské scénáře upravit.



Graf 2 - Sprint 2 - Tým 1 - Burndown chart (zdroj: autor práce)

Tým se na Retrospective schůzce shodl na následujících bodech:

- Rozdělit uživatelské scénáře na několik menších scénářů před začátkem dalšího sprintu.

- Tým požádal Scrum Mastera, aby během příštího sprintu vysvětlil odhadování v bodech a tým tuto praktiku zkusil používat.
- Tým přivítal větší moderaci Daily Scrum schůzek a dodržování časového ohraničení schůzky.
- Odhady jednotlivých scénářů se potvrdily jen náhodou, a to díky velkému množství podhodnocených a velkému množství nadhodnocených dílčích úkolů. Pokud se ve sprintu vyskytnou pouze podhodnocené úkoly, tak pak tým sprint nestihne.
- Nutnost zlepšení integračních testů, které díky výpadkům infrastruktury polovinu sprintu nefungovaly.

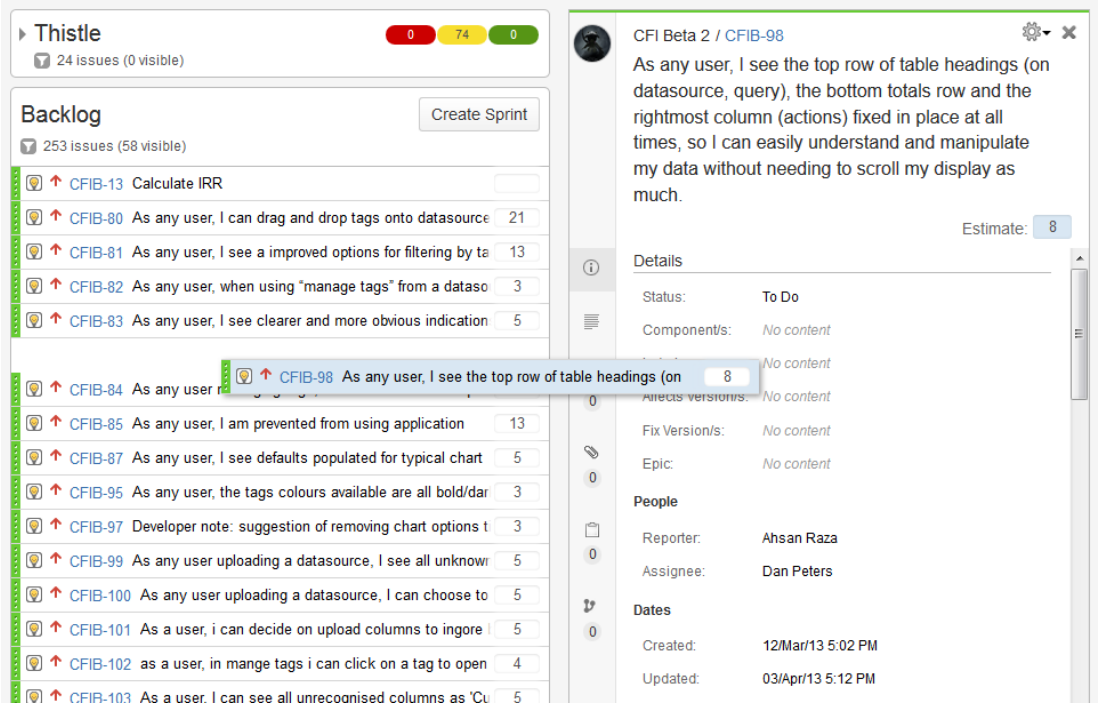
5.5 Sprint 3

Sprit začal plánovací schůzkou. Vzhledem k tomu, že si tým na své Retrospective schůzce odhlasoval, že by měly být uživatelské scénáře, které se budou do této iterace plánovat, menší a jednoduší na implementaci, tak tato schůzka trvala skoro celý den. Tým se pokoušel společně s Product Ownerem uživatelské scénáře rozdělit na co nejmenší. Oproti 11, respektive 12 naplánovaným uživatelským scénářům v minulém sprintu naplánoval tým na tuto iteraci uživatelských scénářů rovnou 20 pro tým 1, respektive 18 pro tým 2. Tým se rozhodl naplánovat uživatelské scénáře dle svojí průměrné velocity. Tým 1 si do svého Sprint Backlogu naplánoval uživatelské scénáře celkově odhadnuté na 89 dní a tým 2 si naplánoval scénáře v celkové hodnotě 77 dní.

Tým si pro tento sprint od Scrum Mastera vyžádal prezentaci a vysvětlení na téma odhadování v bodech, které týmu připadalo lepší než odhadování v hodinách. Scrum Master týmu vysvětlil teorii, která je popsána v teoretické části práce. Tým se rozhodl praktiku přijmout a odhadovat bodem. Scrum Master týmu vysvětlil, jakým způsobem začít. Odhadování v bodech je empirický proces, a tak je důležité podívat se zpět ideálně na již dokončené uživatelské scénáře. Tým si pro odhadování zvolil Fibonacciho posloupnost (1, 2, 3, 5, 8, 13, 21, 40 a otazník). Tým za poslední 2 sprinty dokončil celkem 33 uživatelských scénářů. Scrum Master dal týmu za úkol, aby tento seznam rozdělil od nejlehčího po nejtěžší uživatelský scénář. V tomto ohledu mohl týmu pomoci celkový čas strávený na uživatelském scénáři, který tým již

od začátku projektu měřil. Díky tomuto času nebylo těžké posloupnost scénářů vytvořit. Posloupnost nebyla však pouze seřazena dle stráveného času na scénáři, ale byl brán v potaz i vývojář, který scénář implementoval, a jeho zkušenost, popřípadě vzniklé technické problémy, které ovlivnily celkový čas vývoje scénáře. Scrum Master týmu doporučil, aby takto vzniklý seznam rozdělili na polovinu. Pak Scrum Master doporučil týmu, aby vytvořil hranici, která bude reprezentovat uživatelské scénáře v bodové hodnotě 8. Dalším půlením intervalu se tým dopracoval k bodové hodnotě 3 a 13. Scrum Master doporučil týmu, aby jako horní hranici počítali číslo 21, nikoliv 40. Toto číslo by mělo při odhadování označovat velký uživatelský scénář, který by měl být nadále rozdělen do více scénářů. Scrum Master tak připomíná týmu jeho dohodu o implementaci co nejmenších uživatelských scénářů. Posledním půlením intervalu vznikly hranice pro odhad 2 a 5, horní polovina byla již v této chvíli rozdělena. Scrum Master doporučil týmu, aby si do zasedací místnosti, kde probíhaly Backlog Grooming schůzky, vytvořil tabuli, na které budou jednotlivé scénáře a jejich bodová hodnota vyobrazeny. Díky tomu mohl tým kdykoliv vidět příklady scénářů v určité bodové hodnotě. Odhadování jednotlivých uživatelských scénářů pak probíhalo jako hledání podobného již dokončeného uživatelského scénáře a jeho bodová hodnota byla přiřazena odhadovanému scénáři. Tento postup byl realizován pomocí plánovacího pokeru, který byl vysvětlen výše. Tým ihned tuto praktiku vyzkoušel na následující Product Backlog Grooming schůzce. Tým si praktiku oblíbil a chtěl od příští iterace již používat pouze odhadování v bodech a ne v čase.

Product Owner si na pravidelné schůzce se Scrum Masterem chválil možnost změny pořadí uživatelských scénářů v Product Backlogu. Před plánováním této iterace se totiž objevil požadavek od koncového uživatele produktu na implementaci funkčnosti, která mu brání v používání produktu. Díky tomu, že funkčnost bude v příští verzi zákazníkovi dodána, tak učinil velkou objednávku licencí produktu.



Obrázek 9 - Změna priorit v Product Backlogu (zdroj: autor práce)

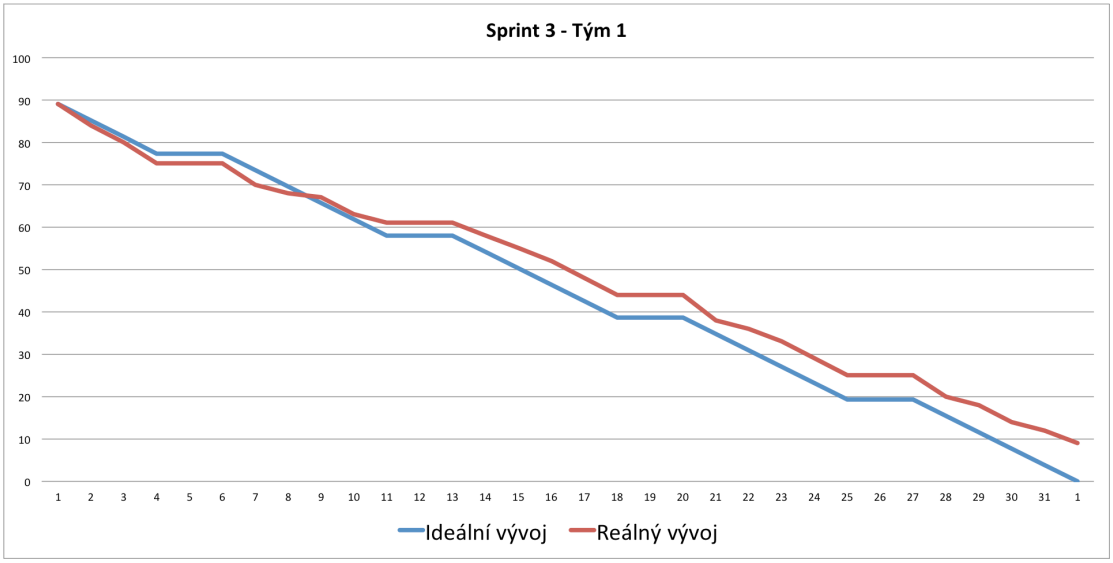
5.5.1 Vyhodnocení sprintu

Na Sprint Review schůzce byli opět přítomni zástupci produktového oddělení z Německa a ředitel firmy. Vývojový tým prošel všechny naplánované uživatelské scénáře a jednotliví členové týmu sami funkčnost prezentovali. Bylo možno pozorovat, jak se všichni zúčastnění snaží poskytnout zpětnou vazbu a Product Owner ji zaznamenával do Product Backlogu. Předešlo se tedy tomu, že by tyto nedostatky vyplynuly na povrch až v době dodání produktu, kdy by jejich oprava mohla být několikanásobně dražší, než když ji Product Owner zařadí na příští Sprint.

Tým také hodnotil použití DoD seznamu úkonů, které musí tým zkontrolovat, pokud chce uživatelský scénář prohlásit za hotový. Tuto praktiku hodnotili členové týmu pozitivně a rozhodli se v ní pokračovat i v příštích iteracích. Tým potvrdil, že se zmenšil počet nutných předání mezi QA a vývojáři.

V tomto sprintu se tým rozhodl začít uživatelské scénáře odhadovat v bodech. Také byl schopen doručit daleko větší počet uživatelských scénářů než v předešlých sprintech. Toto bylo důsledkem rozdělení jednotlivých scénářů na několik menších a jednodušších. Tým potvrdil, že odhadování menších scénářů bylo daleko jednodušší a

přesnější. Toto také jasně dokazuje Burndown chart týmu 1, který je tentokrát daleko lineárnější než minulý sprint.



Graf 3 - Sprint 3 - Tým 1 - Burndown chart (zdroj: autor práce)

Tým se na Retrospective schůzce shodl na následujících bodech:

- používání bodů pro odhad práce,
- nahrazení pomalého serveru pro integrační testy,
- párové programování pro ještě větší kvalitu produktu,
- pozitivně tým hodnotil vývoj sprintu a okamžité řešení problémů,
- lepší komunikace mezi vývojáři.

5.6 Měření spokojenosti – Polovina projektu

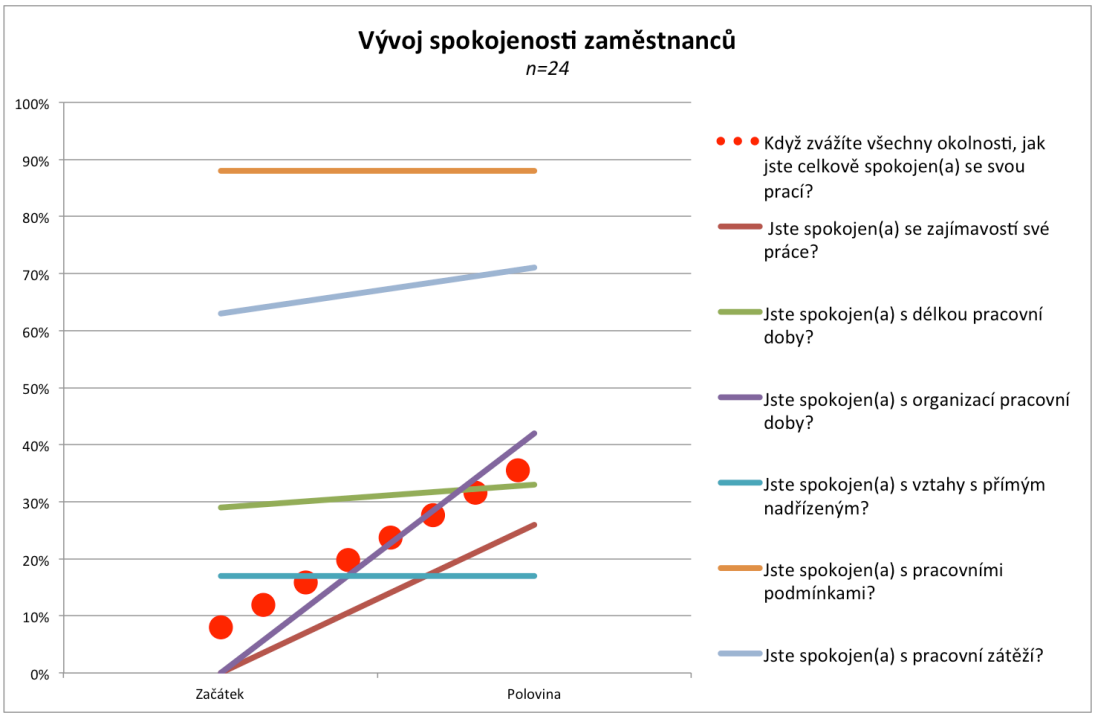
Projekt byl v polovině a autor projektu chtěl zjistit průběžnou hodnotu spokojenosti zaměstnanců. Tato metrika měla ukázat, jak jsou jednotliví členové týmu spokojeni se změnami, které se v týmu dějí. Spokojenost byla měřena za pomoci stejného dotazníku jako na začátku projektu.

n = 24	Rozhodně spokojen		Spíše spokojen		Ani spokojen ani nespokojen		Spíše nespokojen		Rozhodně nespokojen	
	Počet	%	Počet	%	Počet	%	Počet	%	Počet	%
Když zvážíte všechny okolnosti, jak jste celkově spokojen(a) se svou prací?	2	8%	7	29%	10	42%	4	17%	1	4%
Jste spokojen(a) se zajímavostí své práce?	1	4%	5	21%	11	46%	5	21%	2	8%
Jste spokojen(a) s délkou pracovní doby?	0	0%	8	33%	10	42%	6	25%	0	0%
Jste spokojen(a) s organizací pracovní doby?	3	13%	7	29%	8	33%	4	17%	2	8%
Jste spokojen(a) s vztahy s přímým nadřízeným?	0	0%	4	17%	12	50%	3	13%	5	21%
Jste spokojen(a) s pracovními podmínkami? (světlo, teplo, hluk)	12	50%	8	33%	4	17%	0	0%	0	0%
Jste spokojen(a) s pracovní zátěží?	5	21%	12	50%	5	21%	2	8%	0	0%

Tabulka 2 - Výsledky měření spokojenosti - Polovina projektu (zdroj: autor práce)

Výsledky průzkumu prokázaly, že se spokojenost zaměstnanců zlepšovala. Celková spokojenost (možnosti Rozhodně spokojen(a) a Spíše spokojen(a)) vzrostla o 29 % z 8 % na 37 %. Tým potvrdil, že se mu pracuje lépe a radostněji, avšak je ještě velmi brzo na hodnocení. Členové týmu velmi oceňovali, že se nějaká změna vůbec stala. Bylo to poprvé v historii firmy, kdy se takto velká změna procesů uskutečnila. Autor práce cítil v týmu pozitivní energii a velký potenciál.

Spokojenost s dílčími disciplínami dle očekávání vzrostla převážně v oblasti organizace pracovní doby a zajímavosti práce. Vzestupný trend byl velmi podobný vzestupu celkové spokojenosti. Konkrétně u organizace pracovní doby vzrostla spokojenost z 0 % na 42 % a u zajímavosti práce z 0 % na 26 %. Ostatní oblasti nezaznamenaly žádný nebo malý nárůst. Pozitivním znakem bylo i to, že v žádné z oblastí spokojenost neklesala.

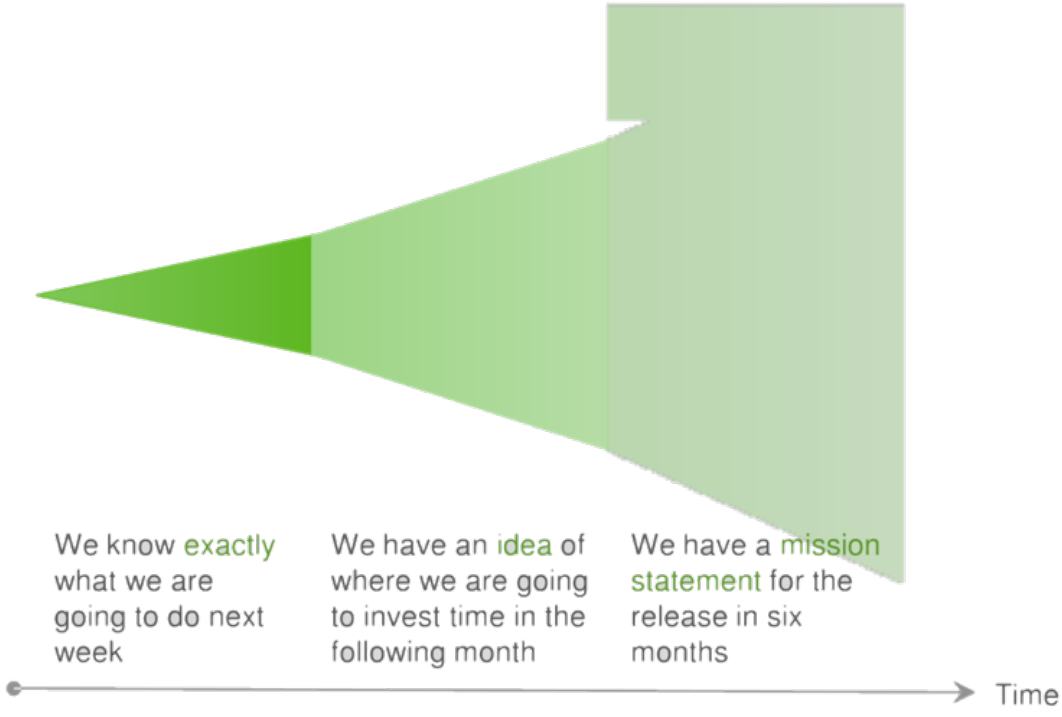


Graf 4 - Vývoj spokojenosti zaměstnanců - polovina projektu (zdroj: autor práce)

5.7 Sprint 4

Ve čtvrtém sprintu čekala tým velká změna v odhadování uživatelských scénářů. Z hodinových odhadů se tým rozhodl přejít na odhadování v bodech, které by mělo lépe sloužit týmu, kde je velký rozdíl ve výkonosti jednotlivých pracovníků. Drobné problémy nastaly již na začátku sprintu, když tým nevěděl, jak určit počet uživatelských scénářů, které do Sprint Backlogu naplánovat. Tým vsadil na své nabyté zkušenosti z minulých sprintů a jednoduše množství odhadl. Při počátečním odhadování pomocí bodů je velmi důležitá druhá část plánovací schůzky, kdy tým dále rozdělí uživatelské scénáře na dílčí úkoly, které přiřadí (na bázi dobrovolnictví) jednotlivým vývojářům a ti odhadnou jejich náročnost. Suma odhadů těchto dílčích úkolů by neměla přesáhnout celkovou kapacitu týmu. Odhadování v bodech je velmi často používáno pro výhled do budoucnosti. V tuto chvíli měl Product Owner připraven svůj Product Backlog v souladu s pokyny metodiky Scrum viz Obrázek 10 - Míra rozpracování uživatelských scénářů v Product Backlogu. V této chvíli by měl mít tým jasný výhled na následující 1-2 sprinty dopředu. Uživatelské scénáře v tomto krátkodobém výhledu by měly být jasně definovány a odhadnuty. Samozřejmě platí, že i když jsou jasně definovány a odhadnuty, tak Product Owner stále může měnit jejich pořadí a reagovat na požadavky zákazníka. Ve střednědobém výhledu, 3-4

sprinty dopředu, by tým měl mít uživatelské scénáře trochu rozpracované a mít povědomí o tom, co je v blízké budoucnosti čeká. Je velmi nepravděpodobné, že scénáře budou implementovány v určeném pořadí, takže tým ani Product Owner neztrácejí drahocenný čas na jejich detailní odhadování. 5 a více sprintů dopředu tým následuje vizi projektu, kterou udržuje Product Owner. Tým ví, jakým směrem se asi bude projekt ubírat, ale již vůbec neztrácí čas detailním odhadováním jednotlivých uživatelských scénářů, protože je jisté, že se jejich pořadí změní. Někdy tým má v této části namísto uživatelských scénářů takzvané epické uživatelské scénáře, které popisují celou funkčnost aplikace. Například: „Já jako zákazník mohu zaplatit za svoji objednávku“. Takto velké téma se v budoucnu rozdělí na několik konkrétních uživatelských scénářů. Aby Product Owner mohl směr projektu orientovat, může tým požádat o hrubý odhad těchto epik. Tým často používá pro odhad velikosti S, M, L, XL, aby pomohl Product Ownerovi v hrubé orientaci o velikosti epiky. Tento odhad není závazný, a proto se používají jiné jednotky, jelikož je zřejmé, že se bude epika ještě měnit a dále specifikovat.



Obrázek 10 - Míra rozpracování uživatelských scénářů v Product Backlogu (zdroj: prezentace Scrum; autor práce)

Tým 1 naplánoval na plánovací schůzce dokončení 180 bodů. Tým dva jich naplánoval 165. Jeden z členů týmu představil párové programování, tedy techniku, o

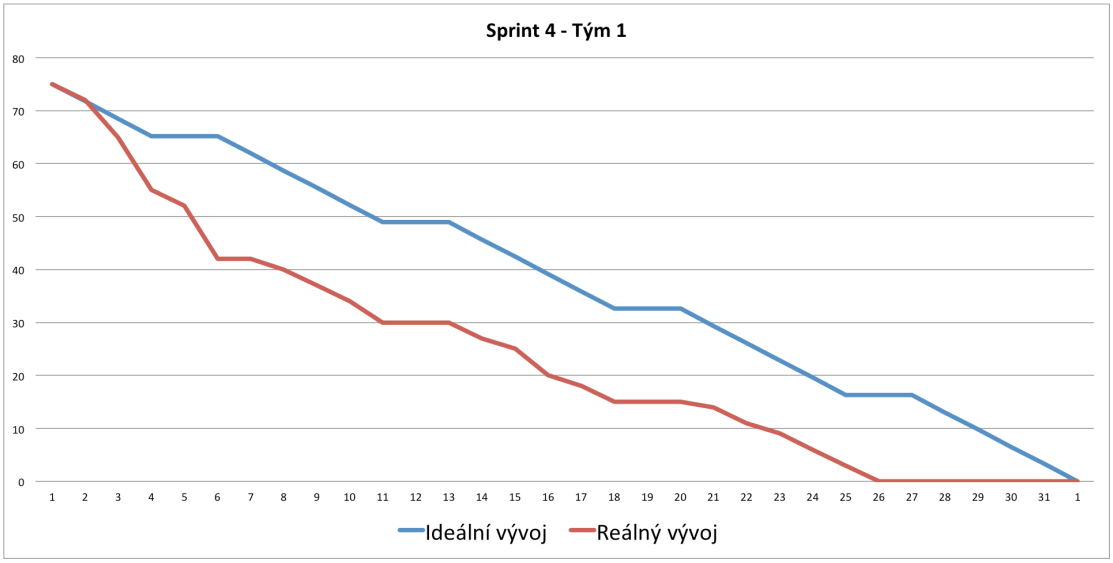
které si myslel, že by mohla přinést větší kvalitu produktu. Tento postup vychází z technik Extrémního programování a je popsán v příslušné kapitole teoretické části této práce. Týmu se nápad sdílení klávesnice líbil a rozhodl se, že ho od příštího sprintu začne používat.

Scrum Master dohodl s ředitelem pobočky, že bude zakoupen nový server na rychlejší pouštění integračních testů. Tyto testy tým zdržovaly, protože byly velmi pomalé. Testy jsou součástí DoD seznamu týmu, a tak je každý člen týmu musí pouštět pro úspěšné dokončení uživatelského scénáře. Toto je také jedno z poslání Scrum Mastera – snažit se zajistit tu nejlepší techniku pro tým a argumentovat, proč ji tým potřebuje.

5.7.1 Vyhodnocení sprintu

Tento sprint dopadl až nad očekávání dobře. Tým 1 na konci doručil 140 % naplánované práce a tým 2 doručil 120 % uživatelských scénářů, které si naplánoval. Oba týmy byly z výsledků nadšené. Scrum Master tým upozornil, že hlavní předností Scrumu je doručovat konzistentní výsledek v trvalém tempu. Samozřejmě je lepší, doručit o 40 % scénářů více než méně. Ale snahou týmu by mělo být doručit tolik scénářů, kolik bylo slíbeno. Díky grafu Burndown chart tým již dobře věděl, že se sprint vyvíjí dobrým tempem a že Product Owner může do sprintu nějaké uživatelské scénáře přidat. Pokud nastane tato situace, tedy že tým má volné kapacity v průběhu sprintu, tak je to opět Product Owner, který dle priorit určuje, který uživatelský scénář do sprintu doplní.

Pokud tým odhaduje scénáře v bodech, tak je Burndown chart konstruován z časových odhadů dílčích úkolů. Nelze tedy říci, že pokud je tým v polovině scénáře ohodnoceného 8 body, tak zbývají ještě 4 body. Místo toho se odhadne zbývající čas jako součet zbývajících odhadů dílčích úkolů.



Graf 5 - Sprint 4 - Tým 1 - Burndown chart (zdroj: autor práce)

Týmy se i v tomto sprintu snažily dělit uživatelské scénáře na co nejmenší, takže jich tým 1 doručil 23 a tým 2 rovných 20.

Tým se na Retrospective schůzce shodl na následujících bodech:

- pokračování odhadování scénářů v bodech,
- používání párového programování,
- lepší odhad dílčích úkolů.

5.8 Sprint 5

Tým se v tom sprintu dohodl, že bude používat metodu z Extrémního programování, konkrétně párové programování. Extrémní programování je známé tím, že používá své metody do extrému, tedy u každého uživatelské scénáře nehledě na jeho náročnost. Tým 1 v tomto sprintu naplánoval 225 bodů, což v součtu odhadů dílčích úkolů scénářů znamenalo naplánování 90 dnů. Tým 2 si naplánoval podle výsledků minulého sprintu jen 190 bodů v celkovém součtu dílčích úkolů 80 dní.

Tým dle plánu všechny uživatelské scénáře implementoval vždy ve dvou lidech a vývojáři si tento způsob práce velmi chválili. Ve sprintu byly opět naplánovány na každý týden Product Grooming schůzky, na kterých tým odhadoval budoucí uživatelské scénáře. Výhodou agilních sprintů je to, že na konci iterací musí být

funkční inkrement projektu, který je potencionálně možno odeslat zákazníkovi. Toto si tým a produktové oddělení velmi chválilo, protože předtím byla polovina funkčnosti rozpracovaná, ale nikdy ne dokončená, a tak se vždy čekalo až na poslední iteraci před vydáním nové verze, kdy se všechny rozdělané funkčnosti dokončovaly. Nyní má produktový tým možnost některým vybraným zákazníkům odeslat beta verzi, která obsahuje plně funkční scénáře, takže dle jejich reakce může jednotlivé funkčnosti dále upravovat, než vydá veřejnou verzi produktu.

5.8.1 Vyhodnocení sprintu

Tento sprint bohužel nedopadl z hlediska dokončených uživatelských scénářů moc dobře. Tým na Sprint Review musel prezentovat to, že z 21, respektive 19, uživatelských scénářů dokončil pouze 16, respektive 15. Hlavním důvodem nezdaru bylo zpomalení práce z důvodu aplikace párového programování. Scrum Master týmu naznačoval, že párové programování je technika hlavně určená pro zkvalitnění výsledného kódu, potažmo lepší kvality funkčností. Tým byl ale přesvědčen, že 4 ruce udělají práci dvakrát rychleji. Předpověď Scrum Mastera se však stala skutečností a tým 1 doručil pouze 70 % naplánovaných scénářů a tým 2 jen 65 %. Tým si však párové programování velmi chválil a rozhodl se v této metodě pokračovat i příští sprint, avšak ne tak dogmaticky. Oba týmy se chtěly samy rozhodnout, které úkoly budou chtít řešit pomocí párového programování, ty ostatní budou řešit samostatně jako doposud. Tým se také rozhodl, že bude používat větší odhad časové náročnosti pro výrobu těchto párových scénářů. Toto je velká výhoda agilních metodik. Tým si může použitou strategii a techniky upravovat tak, jak si myslí, že je pro něj nejlepší.

Tým se na Retrospective schůzce shodl na následujících bodech:

- Pokračovat v párovém programování. Vybírat si úkoly, které tak bude implementovat.
- Dokončit vydání nové produktové verze, kterou zákazníci očekávají na konci příštího sprintu.

5.9 Sprint 6

Scrum Master byl lehce zaskočen požadavkem týmu z poslední Retrospective schůzky, kde se tým shodl, že musí připravit produkt k vydání produktové verze. Tým

toto na plánovací schůzce vysvětlil jako zvyk z minulosti. Při vydání posledních verzí byl tým zvyklý, že většinu iterací zasvětil dopracování drobností, na které v průběhu vývoje nebyl čas. Díky Scrumu ale byla během posledních 5 sprintů věnována velká pozornost tomu, aby tým na konci iterace dosáhl potencionálně odeslatelného inkrementu produktu, takže tyto dodělávky již nebyly potřeba v tak velké míře jako při minulých vydání. Tým se v posledním sprintu rozhodl, že každý tým věnuje 40 % své kapacity na opravování defektů, které byly nashromážděny z minulých verzí a které tým objevil během vývoje této verze. Také byl na plánování sprintu přítomný zákazník, tedy produktové oddělení z Německa, které se pokusilo týmu přidělit dvakrát tolik práce, než co tým navrhl. Dle jejich odhadů by tým měl na konci toho sprintu vydat mnohem více funkčností, než plánuje. Scrum Master opět zákazníkovi vysvětlil, jak Scrum funguje a že i kdyby toto tým slíbil, tak je velká pravděpodobnost že navýšený seznam nedoručí. Tým 1 na tuto iteraci naplánoval doručit 21 uživatelských scénářů a tým 2 22 scénářů.

5.9.1 Vyhodnocení sprintu

Týmu se tento sprint povedl dle plánu. Na začátku sprintu si určil kritické úkoly, které si přál vyvíjet za pomoci párového programování. Těchto kritických úkolů bylo zhruba 35 % za všech úkolů. Tým si i v tomto sprintu tuto praktiku velmi chválil a věřil, že takto vyvinuté funkčnosti budou daleko kvalitnější a nebudou z nich vraceny žádné defekty zpět do týmu. Tým na konci sprintu byl schopen označit číslo úspěšného buildu (build je sestavení a přeložení všech zdrojových kódů aplikace a spuštění všech testů projektu), který byl předán marketingovému oddělení k distribuci pro koncové zákazníky. Product Owner si velmi chválil to, že dříve by se v tuto chvíli musely práce na projektu přerušit a začít plánovat novou produktovou verzi. Namísto toho díky Scrumu mají připravený Product Backlog, na kterém nepřetržitě pracovali a domnívali se, že vývoj bude moci pokračovat hned zítra dalším sprintem a nebude muset být přerušen. Tato kontinuální práce je velkou devizou agilních metodik.

5.10 Měření spokojenosti – Konec projektu

Na konci šestého sprintu, po úspěšném vydání nové verze produktu, byla naposledy pro účely této práce změřena spokojenost zaměstnanců. Spokojenost

byla měřena za pomoci dotazníku s totožnými otázkami jako na začátku a vprostřed projektu. Vynechala bych otázky a nechala bych jen tabulku.

n = 24	Rozhodně spokojen		Spíše spokojen		Ani spokojen ani nespokojen		Spíše nespokojen		Rozhodně nespokojen	
	Počet	%	Počet	%	Počet	%	Počet	%	Počet	%
Když zvážíte všechny okolnosti, jak jste celkově spokojen(a) se svou prací?	5	21%	9	38%	8	33%	2	8%	0	0%
Jste spokojen(a) se zajímavostí své práce?	4	17%	6	25%	9	38%	4	17%	1	4%
Jste spokojen(a) s délkou pracovní doby?	2	8%	9	38%	7	29%	6	25%	0	0%
Jste spokojen(a) s organizací pracovní doby?	5	21%	9	38%	7	29%	2	8%	1	4%
Jste spokojen(a) s vztahy s přímým nadřízeným?	0	0%	4	17%	12	50%	3	13%	5	21%
Jste spokojen(a) s pracovními podmínkami? (světlo, teplo, hluk)	12	50%	9	38%	3	13%	0	0%	0	0%
Jste spokojen(a) s pracovní zátěží?	8	33%	11	46%	3	13%	2	8%	0	0%

Tabulka 3 - Výsledky měření spokojenosti - konec projektu (zdroj: autor práce)

Tento výzkum spokojenosti prokázal pozitivní působení změn procesů ve firmě na zaměstnance. Celkem 58 % zaměstnanců potvrdilo, že jsou se svou prací spokojeni (možnosti Rozhodně spokojen(a) a Spíše spokojen(a)), což je nárůst o 21 % oproti minulému měření. Zbytek zaměstnanců, 33 %, zastávalo neutrální postoj s tím, že se báli, že toto byla jen taková přechodná fáze a vše se pomalu vrátí do starých kolejí. Pouze 8 % zaměstnanců uvedlo, že jsou se svou prací nespokojeni. Celkové srovnání bude tato práce hodnotit v následující kapitole.

V dílčích kategoriích měření byl zaznamenán nárůst ve spokojenosti s organizací pracovní doby (z 42 % na 58 %) a se zajímavostí práce (z 26 % na 42 %). Také byl velmi oceněn vliv agilních metodik na pracovní zatížení. Před změnou procesů bylo normální, že v iteraci před vydáním nové verze produktu byli zaměstnanci nuceni pracovat přesčas, aby vůbec byli schopni doručit funkční produkt. Díky tomu, že na konci každého sprintu je produkt funkční a plně otestovaný, tak v poslední iteraci nebylo nutno sáhnout k velkým změnám ani přesčasům.

6 Případová studie - Vyhodnocení

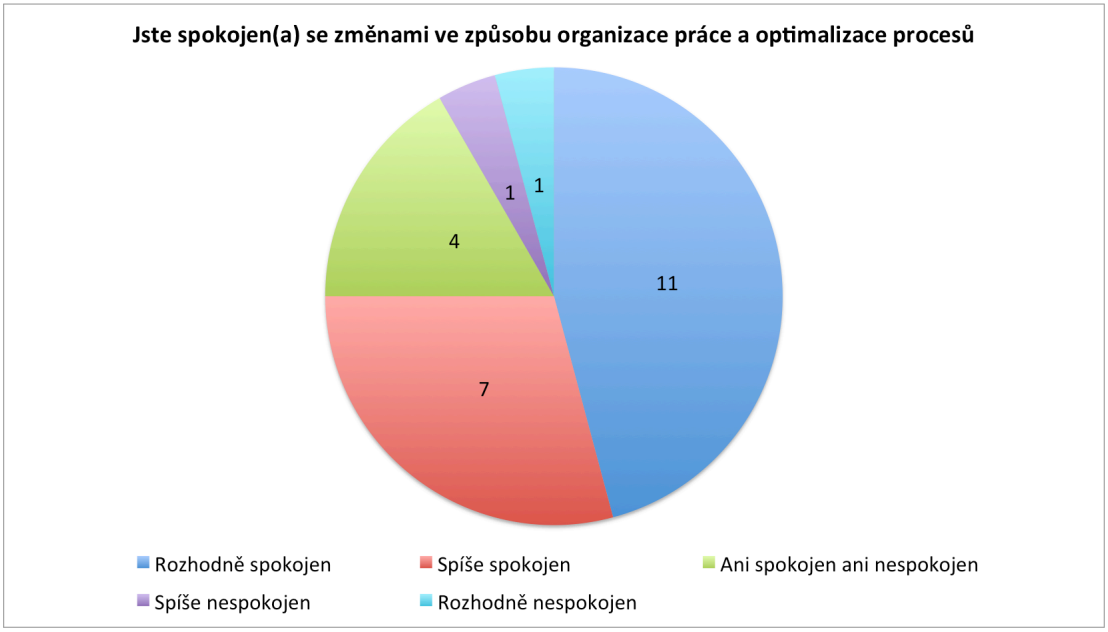
6.1 Zefektivnění způsobu organizace práce a optimalizace procesů

Je velmi těžké kvantitativně změřit, jestli byl způsob organizace práce a optimalizace procesů zlepšena. Pro potřeby této práce a kvantifikování provedených změn se autor rozhodl vytvořit dotazník, který zkoumal spokojenost zaměstnanců se změnou procesů ve firmě. Dotazník byl distribuován online a skládal se z jedné jednoduché otázky

- Jste spokojen(a) se změnami ve způsobu organizace práce a optimalizace procesů?

Každý zaměstnanec firmy měl odpovědět na tuto otázku jednou z následujících možností:

- Rozhodně spokojen(a)
- Spíše spokojen(a)
- Ani spokojen(a), ani nespokojen(a)
- Spíše nespokojen(a)
- Rozhodně nespokojen(a)



Graf 6 – Výsledky dotazníku – vyhodnocení změn způsobu organizace práce a optimalizace procesů (zdroj: autor práce)

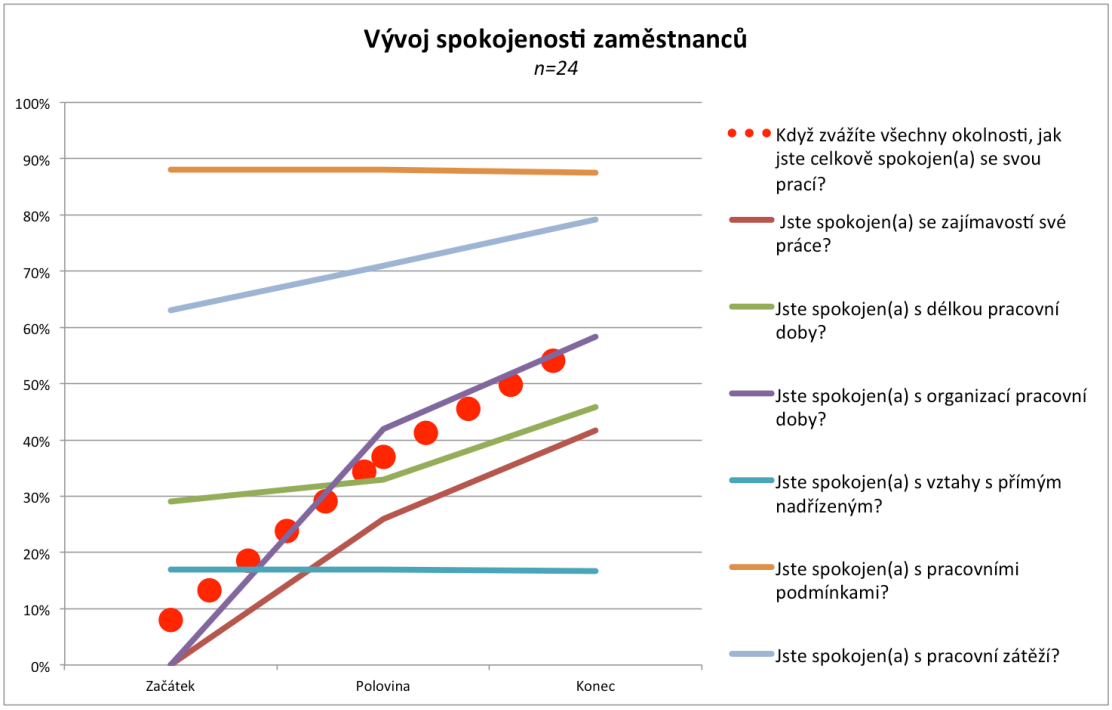
Na vyplnění dotazníku měli zaměstnanci firmy jeden den a výsledky dotazníku jsou následující:

- Rozhodně spokojen(a) – 11 zaměstnanců
- Spíše spokojen(a) – 7 zaměstnanců
- Ani spokojen(a), ani nespokojen(a) – 4 zaměstnanců
- Spíše nespokojen(a) – 1 zaměstnanec
- Rozhodně nespokojen(a) - 1 zaměstnanec

Celkem tedy 18 zaměstnanců potvrdilo že jsou spokojeni (možnosti Rozhodně spokojen(a) a Spíše spokojen(a)) se změnami provedenými v organizaci práce a optimalizaci procesů.

6.2 Zlepšení spokojenosti zaměstnanců

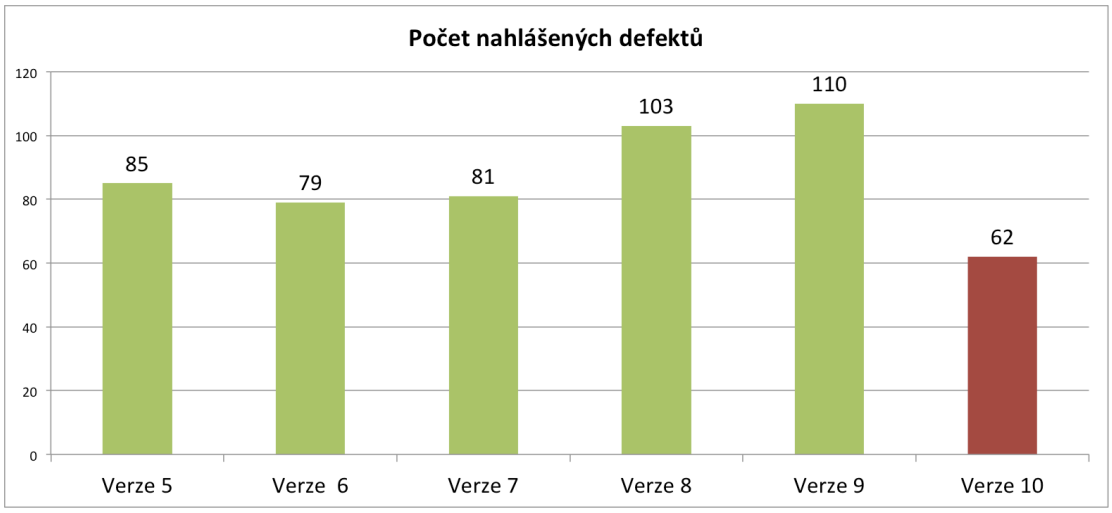
Jednou z hlavních cílů této práce bylo popsat optimalizaci organizace práce za pomoci nasazení metodiky Scrum. Jeden z dopadů této optimalizace by mělo být zlepšení spokojenosti zaměstnanců. Je nutno přiznat, že autor pozoroval to, že zaměstnanci byli na začátku projektu velmi demotivovaní a nespokojení. Autor práce měřil spokojenost pomocí dotazníku, který je popsaný v kapitolách konkrétní číslo. Díky zjištěným výsledkům můžeme konstatovat viditelné zlepšení spokojenosti. Tým ve skupinových diskuzích, které byli na toto téma pořádané, potvrdil, že nasazení metodiky Scrum bylo opravdu dobře načasováno a tým potřeboval v době začátku projektu své procesy změnit. Celková spokojenost zaměstnanců vzrostla z 8 % na 58 %. Tento dobrý výsledek podporuje nárůst spokojenosti zaměstnanců v dílčích otázkách.. Firma zaznamenala nárůst spokojenosti zaměstnanců v kategorii zajímavosti práce (z 0 % na 42 %) i délky pracovní doby (z 29 % na 46 %). Tým si velmi chválil jak si může organizovat práci sám a není v každém kroku nucen poslouchat příkazy projektového manažera a architektů. Namísto toho pracuje jako opravdový tým a volí si takový postup, který mu připadá nejlepší. Také si tým velmi chválil, že na konci poslední iterace před vydáním nové verze projektu odpadl stres, který byl přítomný před vydáním minulých verzí. V těchto iteracích nebyl tým nucen pracovat přes čas a všechny aktivity probíhaly v daleko větším klidu. Díky týmové práci a svobodě měl tým pocit, že stihl doručit daleko větší množství funkčností, než když byl vedoucím pobočky a manažerem nucen do nereálných plánů.



Graf 7 - Vývoj spokojenosti zaměstnanců - Konec projektu (zdroj: autor práce)

6.3 Snížené množství defektů v budoucím vydání produktu

Jedním z hlavních parametrů agilního vývoje je kvalita výsledného produktu, tedy fungující software. Cílem toho výzkumu bylo potvrdit, že pokud tým bude používat agilní metodiky, tak jeho produkt bude kvalitní. Firma v posledních vydáních byla konfrontována s narůstajícím počtem nahlášených defektů ve svém produktu. Od zavedení agilní metodiky Scrum si slibovala pokles této hodnoty, tedy, že počet defektů s vážností 5 a více (1 – nejméně vážný, 10 – nejvíce vážný) bude v nově vydané verzi menší než ve verzích předchozích. Toto hodnocení mohlo proběhnout až půl roku po vydání této verze. Statistika jasně ukazuje, že počet klesl ze 110 nahlášených defektů ve verzi předchozí na 62 v této verzi. I číslo 62 je poměrně vysoké na projekt řízený agilními metodikami a tým by měl v tomto ohledu ještě zapracovat.



Graf 8 – Vývoj počtu nahlášených defektů – konec projektu (zdroj: autor práce)

6.4 Závěrečná doporučení a budoucnost firmy

Nasazení agilní metodiky Scrum do firmy bylo úspěšné, ať již měřeno spokojeností zaměstnanců (spokojenější zaměstnanci jsou motivovanější a odvádějí kvalitnější práci), nebo spokojeností zákazníka, který díky této metodice dostal kvalitnější produkt v naplánovaném čase. I tato případová studie ukázala, jak je role Scrum mastera pro tým nenahraditelná a jak výrazně může Scrum master ovlivnit atmosféru ve firmě a vztahy mezi vývojáři a zákazníkem, potažmo úspěšnost produktu i celé firmy. Tým se rozhodl i nadále pokračovat v používání metodiky Scrum a agilních přístupů. Agilní metodiky říkají, že proces týmu není nikdy dokonalý a může být neustále zlepšován. Autor práce se dohodl s firmou na pokračování spolupráce a stal se stálým členem a Scrum Masterem týmu, na této pozici pracuje již třetí rok.

Závěr

Agilní metodiky jasně dominují organizaci práce při vývoji softwaru. Dnes je velmi těžké najít softwarovou společnost, která by o sobě alespoň netvrdila, že agilní procesy ve firmě používá. Popularitu si získaly hlavně pro způsob vedení, kdy vývojáři, tedy ti nejvíce informovaní lidé o dění v projektu, přímo ovlivňují rozhodnutí o vedení, organizaci a komunikaci v projektu. Již dávno jsou pryč doby, kdy se vývoj softwaru podobal výrobnímu závodu, kdy předák obrazně stál na vyvýšeném místě, udílel rozkazy a nutil zaměstnance k vyšším výkonům. Tento způsob vedení se postupem času ukázal u tak komplexního úkolu, jako je vývoj softwaru, jako neefektivní. To samé platí o specifikaci požadavků. V rigorózních metodikách je třeba celý projekt pečlivě vymyslet dopředu. Připravit všechny požadavky na funkčnost, analyzovat nefunkční požadavky a celý projekt zdokumentovat ještě dříve, než se napíše první řádka kódu. Toto vedlo k mnoha dnes již komickým situacím, kdy zákazník ve smluvený den dodání dostal naprosto odlišný nebo nefunkční projekt. Nefunkčnost však byla nahrazena velmi dobrou dokumentací, která celý projekt popisovala do nejmenšího detailu. Často se také stalo, že zákazník svůj objednaný projekt z důvodu zpoždění nedostal vůbec. Zákazník se dostal do krizové situace. Projekt nefungoval a ještě byl mnohem dražší, než čekal. Řešením bylo použití iteračního a inkrementálního vývoje, tedy základních stavebních kamenů všech agilních metodik. Díky orientaci na potřeby zákazníka a neustálému sledování, zda jde vývoj správným směrem, je zákazníkovi na konci projektu dodán produkt který mu přináší přidanou hodnotu, kterou očekával. Agilní vývoj zahazuje magické křišťálové koule, ze kterých architekti ve vodopádových modelech věští, co zákazník bude za rok potřebovat, a místo toho nařizují v projektu kontinuální spolupráci se zákazníkem a vyladění požadavků na produkt do nejmenších detailů podle jeho aktuálních potřeb. Agilní proces však není neomylný a vyžaduje opravdové lidi, kteří budou chtít o projektu přemýšlet a neustále ho zlepšovat, nikoliv lidské zdroje. Agilní procesy tvrdí, že je nelze jednou nastavit a pak několik let používat tak, jak jsou. Namísto toho nutí všechny zúčastněné k neustálé revizi procesu a úpravě dle toho, v čem tým aktuálně shledává největší nedostatky a problémy. Agilní metodiky nejsou dogmatické a poskytují členům týmu volnost pro maximální využití lidského potencionálu.

Seznam použitých zdrojů

Buzzword. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-06-07]. Dostupné z: http://en.wikipedia.org/wiki/Buzzword

A Spiral Model of Software Development and Enhancement. A Spiral Model of Software Development and Enhancement [online]. 1986 [cit. 2015-05-31]. Dostupné z: http://csse.usc.edu/csse/TECHRPTS/1988/usccse88-500/usccse88-500.pdf

Akademický slovník cizích slov: [A-Ž]. 1. vyd. Praha: Academia, 1997, 834 s. ISBN 80-200-0607-9.

APKE, Larry. Understanding The Agile Manifesto: A Brief & Bold Guide to Agile. Internet: Amazon Digital Services, 2014. ISBN ASIN:B00GR31ZDU.

ARNOWITZ, Jonathan, Michael ARENT a Nevin BERGER. Effective prototyping for software makers. 1st ed. Boston: Elsevier, 2007, xxxviii, 584 p. ISBN 978-012-0885-688.

BAKER, Simon. NO BULL. NO BULL. 2012. Dostupné z: http://www.energizedwork.com/no-bull

BECK, Kent a Cynthia ANDRES. Extreme programming explained: embrace change. 2nd ed. Boston, MA: Addison-Wesley, 2005, xxii, 189 p. ISBN 03-212-7865-8.

BOEHM, Barry W, Jo Ann LANE, Supannika KOOLMANOJWONG a Richard TURNER. The incremental commitment spiral model: principles and practices for successful systems and software. New York: Addison-Wesley Professional, 2014, xxii, 310 pages. ISBN 03-218-0822-3.

CHROMATIC.,. Extreme programming pocket guide. 1st ed. Farnham: O'Reilly, 2003, xv, 90 p. ISBN 05-960-0485-0.

COCKBURN, Alistair. Crystal clear: a human-powered methodology for small teams. Boston: Addison-Wesley, 2005, xxii, 312 p. ISBN 02-016-9947-8.

COCKBURN, Alistair. Using Both Incremental and Iterative Development. Using Both Incremental and Iterative Development [online]. 2008 [cit. 2015-05-31]. Dostupné z: http://static1.1.sqspcdn.com/static/f/702523/9242211/1288741989673/200805-Cockburn.pdf?token=fYl%2FZ3sz1lHaQLfYSXGMsni%2B1CY%3D

COHN, Mike. Agile estimating and planning. Upper Saddle River: Prentice Hall, 2006, xxx, 330 s. ISBN 01-314-7941-5.

COHN, Mike. User stories applied: for agile software development. Boston: Addison-Wesley, 2004, xxi, 268 p. ISBN 03-212-0568-5.

GIBSON, Jeremy. Introduction to game design, prototyping, and development: from concept to playable game-with Unity® and C#. New York: Addison-Wesley Professional, 2014, xxxi, 908 pages. ISBN 03-219-3316-8.

GOLDSTEIN, Ilan. Scrum shortcuts without cutting corners: agile tactics, tools. Calif.: Addison-Wesley Signature Series, 2013, 178 pages. ISBN 978-032-1822-369.

HAIS, Karel a Břetislav HODEK. Velký anglicko-český slovník: english-czech dictionary. 2nd ed. Praha: Academia, 1991, 755 s. ISBN 80-200-0065-81.

HIGHSMITH, James A. Agile project management: creating innovative products. 2nd ed. Upper Saddle River, NJ: Addison-Wesley, 2010, xxx, 392 p. Agile software development series. ISBN 03-216-5839-6.

HOYLE, David. ISO 9000 quality systems handbook: using the standards as a framework for business improvement. 6th ed. Amsterdam: Butterworth-Heinemann, 2009. ISBN 978-185-6176-842.

JOHNSON, by Chris Sims and Hillary Louise. The elements of scrum. 1st ed. Foster City, Calif: Dymaxicon, 2011. ISBN 978-098-2866-917.

JOHNSON, Hillary Louise a Chris SIMS. The elements of scrum. 1st ed. Foster City, California: Dymaxicon, 2011. ISBN 978-098-2866-917.

KEITH, Clinton. Agile game development with Scrum. Upper Saddle River, NJ: Addison-Wesley, 2010, xxv, 340 p. Addison-Wesley signature series. ISBN 978-032-1618-528.

KRIVITSKY, Alexey. Scrum Simulation with LEGO [online]. 2008 [cit. 2015-05-30]. Dostupné z: http://www.lego4scrum.com/

KROLL, Per a Philippe KRUCHTEN. The rational unified process made easy: a practitioner's guide to the RUP. Boston: Addison-Wesley, 2003, xxxv, 416 p. ISBN 03-211-6609-4.

KRUCHTEN, Philippe a [trad.: Francesco Bonfiglio .. et]. AL]. Rational unified process: introduzione. Milano: Addison Wesley Longman Italia, 2000. ISBN 978-887-1920-733.

LARMAN, Craig. Agile and iterative development: a manager´s guide. Boston: Addison-Wesley, 2003, xiv, 342 s. ISBN 01-311-1155-8.

LEFFINGWELL, Dean. Agile software requirements: lean requirements practices for teams, programs, and the enterprise. Upper Saddle River, NJ: Addison-Wesley, 2011, xxxv, 518 p. Agile software development series. ISBN 978-0321635846.

Manifest Agilního vývoje software [online]. 2001 [cit. 2015-05-22]. Dostupné z: http://agilemanifesto.org/iso/cs/

MEYER, Bertrand. Agile!: the good, the hype and the ugly. Cham: Springer, 2014, xix, 170 s. ISBN 978-3-319-05154-3.

Prince 2 - Obchodní případ - Business Case. Prince 2 [online]. 2015. [cit. 2015-05-20]. Dostupné z: http://prince-2.cz/index.php/index/page/1018_business-case-zduvodnene-projektu-obchodni-pripad

Rational Unified Process Best Practices for Software Development Teams. Rational Unified Process Best Practices for Software Development Teams [online]. 2001 [cit. 2015-05-31]. Dostupné z: http://www.ibm.com/developerworks/rational/library/content/03July/1000/1251/1251_bestpractices_TP026B.pdf

Report on a conference sponsored by the NATO Science Commitee. The NATO Software Engineering Conferences [online]. 1969 [cit. 2015-05-31]. Dostupné z: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

ROYCE, Winston W. MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS. MANAGING THE DEVELOPMENT OF LARGE SOFTWARE SYSTEMS [online]. 1970 [cit. 2015-05-31]. Dostupné z:

http://www.serena.com/docs/agile/papers/Managing-The-Development-of-Large-Software-Systems.pdf

RUBIN, Kenneth S. Essential Scrum: a practical guide to the most popular agile process. Upper Saddle River, NJ: Addison-Wesley, 2012, xliii, 452 p. ISBN 01-370-4329-5.

SCHWABER, Ken. Agile project management with Scrum. Redmond, Wash.: Microsoft Press, 2004, xix, 163 p. ISBN 07-356-1993-X.

SHORE, James a Shane WARDEN. The art of agile development. Sebastopol, CA: O'Reilly Media, Inc., 2008, xviii, 409 p. Theory in practice (Sebastopol, Calif.). ISBN 978-059-6527-679.

SIMS, Chris a Hillary Louise JOHNSON. The elements of scrum. 1st ed. Foster City, Calif: Dymaxicon, 2011. ISBN 978-098-2866-917.

Spiral model. Wikipedia: the free encyclopedia [online]. San Francisco (CA): Wikimedia Foundation, 2001- [cit. 2015-05-31]. Dostupné z: http://en.wikipedia.org/wiki/Spiral_model

STAPLETON, Jennifer. DSDM, dynamic systems development method: the method in practice. Reprint. Harlow [u.a.]: Addison-Wesley, 1998. ISBN 978-020-1178-890.

State of Agile Development Survey Results. State of Agile Development Survey Results [online]. 2011 [cit. 2015-05-22]. Dostupné z: http://www.versionone.com/state_of_agile_development_survey/2011/

STELLMAN, Andrew. Learning Agile. New York: Oreilly, 2014. ISBN 978-1449331924.

Vodopádový model. Vodopádový model [online]. 2015 [cit. 2015-05-31]. Dostupné z: https://managementmania.com/cs/vodopadovy-model-waterfall-model

VÝZKUMNÝ ÚSTAV PRÁCE A SOCIÁLNÍCH VĚCÍ. Spokojenost zaměstnanců: Manuál pro měření a vyhodnocení úrovně spokojenosti zaměstnanců. 1. vydání. Praha: VÚPSV, 2007. ISBN 978-80-87007-71-6.

WIEGERS, Karl Eugene. Software requirements. 3rd ed. Redmond, WA: Microsoft press, 2013, xxxii, 637 s. Best practices. ISBN 978-0-7356-7966-5.

WILLIAMS, Laurie. A Survey of Agile Development Methodologies. A Survey of Agile Development Methodologies [online]. 2007 [cit. 2015-05-31]. Dostupné z: http://agile.csc.ncsu.edu/SEMaterials/AgileMethods.pdf

WINSTON, Royce. Managing the Development of Large Software Systems. Managing the Development of Large Software Systems [online]. 1970 [cit. 2015-05-20]. Dostupné z: http://www.cs.umd.edu/class/spring2003/cmsc838p/Process/waterfall.pdf

Seznam obrázků, tabulek a grafů

Seznam obrázků

Seznam tabulek

Seznam grafů

Přílohy

Seznam příloh

Příloha A – Dotazník spokojenosti zaměstnanců

Dotazník spokojenosti zaměstnanců

Zakroužkujte právě jednu odpověď.

Otázka 1: Když zvážíte všechny okolnosti, jak jste celkově spokojen(a) se svou prací?

a) Rozhodně spokojen(a)
b) Spíše spokojen(a)
c) Ani spokojen(a), ani nespokojen(a)
d) Spíše nespokojen(a)
e) Rozhodně nespokojen(a)

Otázka 2: Jste spokojen(a) se zajímavostí své práce?

a) Rozhodně spokojen(a)
b) Spíše spokojen(a)
c) Ani spokojen(a), ani nespokojen(a)
d) Spíše nespokojen(a)
e) Rozhodně nespokojen(a)

Otázka 3: Jste spokojen(a) s délkou pracovní doby?

a) Rozhodně spokojen(a)
b) Spíše spokojen(a)
c) Ani spokojen(a), ani nespokojen(a)
d) Spíše nespokojen(a)
e) Rozhodně nespokojen(a)

Otázka 4: Jste spokojen(a) s organizací pracovní doby?

a) Rozhodně spokojen(a)
b) Spíše spokojen(a)
c) Ani spokojen(a), ani nespokojen(a)
d) Spíše nespokojen(a)
e) Rozhodně nespokojen(a)

Otázka 5: Jste spokojen(a) s vztahy s přímým nadřízeným?

a) Rozhodně spokojen(a)
b) Spíše spokojen(a)
c) Ani spokojen(a), ani nespokojen(a)
d) Spíše nespokojen(a)
e) Rozhodně nespokojen(a)

Otázka 6: Jste spokojen(a) s pracovními podmínkami? (světlo, teplo, hluk)

a) Rozhodně spokojen(a)
b) Spíše spokojen(a)
c) Ani spokojen(a), ani nespokojen(a)
d) Spíše nespokojen(a)
e) Rozhodně nespokojen(a)

Otázka 7: Jste spokojen(a) s pracovní zátěží?

a) Rozhodně spokojen(a)
b) Spíše spokojen(a)
c) Ani spokojen(a), ani nespokojen(a)
d) Spíše nespokojen(a)
e) Rozhodně nespokojen(a)

Příloha B – Dotazník vyhodnocení změn způsobu organizace práce a optimalizace procesů

Dotazník vyhodnocení změn způsobu organizace práce a optimalizace procesů

Zakroužkujte právě jednu odpověď.

Otázka: Jste spokojen(a) se změnami ve způsobu organizace práce a optimalizace procesů

a) Rozhodně spokojen(a)
b) Spíše spokojen(a)
c) Ani spokojen(a), ani nespokojen(a)
d) Spíše nespokojen(a)
e) Rozhodně nespokojen(a)

Evidence výpůjček

Prohlášení:

Dávám svolení k půjčování této diplomové práce. Uživatel potvrzuje svým podpisem, že bude tuto práci řádně citovat v seznamu použité literatury.

V Praze, 8.6. 2015.

Martin Popelák

Jméno	Katedra / Pracoviště	Datum	Podpis