VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA PODNIKATELSKÁ
FACULTY OF BUSINESS AND MANAGEMENT

ÚSTAV MANAGEMENTU
INSTITUTE OF MANAGEMENT

ZAVEDENÍ SCRUM V PROJEKTU
IMPLEMENTATION OF SCRUM IN THE PROJECT

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Karin Koyšová
AUTHOR

VEDOUCÍ PRÁCE — Ing. et Ing. Pavel Juřica, Ph.D.
SUPERVISOR

BRNO 2022

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA PODNIKATELSKÁ

Zadání diplomové práce

Ústav:	Ústav managementu
Studentka:	**Bc. Karin Koyšová**
Vedoucí práce:	**Ing. et Ing. Pavel Juřica, Ph.D.**
Akademický rok:	2021/22
Studijní program:	Strategický rozvoj podniku

Garant studijního programu Vám v souladu se zákonem č. 111/1998 Sb., o vysokých školách ve znění pozdějších předpisů a se Studijním a zkušebním řádem VUT v Brně zadává diplomovou práci s názvem:

Zavedení SCRUM v projektu

Charakteristika problematiky úkolu:

Úvod
Vymezení problému a cíle práce
Teoretická východiska práce
Analýza problému a současné situace
Vlastní návrhy řešení, přínos návrhů řešení
Závěr
Seznam použitých zdrojů

Cíle, kterých má být dosaženo:

Cílem diplomové práce je aplikace metodiky SCRUM a návrh na zlepšení procesu vývoje softvéru na vybraném projektu.

Základní literární prameny:

CRAIG, L. a B. VODDE. Large-Scale Scrum. Pearson Education, 2016. ISBN 978-0321985712.

DOLEŽAL, J. Projektový management: komplexně, prakticky a podle světových standardů. Praha: Grada Publishing, 2016. ISBN 978-80-247-5620-2.

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

ŠOCHOVÁ, Z. a E. KUNCE. Agilní metody řízení projektů. Brno: Computer Press, 2014. ISBN 978-8-251-4194-6.

ŠOCHOVÁ, Z. Skvělý ScrumMaster. Brno: Computer Press, 2018. ISBN 978-80-251-4927-0.

Termín odevzdání diplomové práce je stanoven časovým plánem akademického roku 2021/22

V Brně dne 28.2.2022

L. S.

doc. Ing. Vít Chlebovský, Ph.D. garant	doc. Ing. Vojtěch Bartoš, Ph.D. děkan

Abstrakt

Diplomová práca sa zaoberá problematikou agilného vývoja softvéru. Teoretická časť práce je venovaná popisu vývoja softvéru, tradičnému a agilnému prístupu k projektovému riadeniu. Analytická časť popisuje súčasný stav projektového riadenia a procesu vývoja softvéru na vybranom projekte. Návrhovú časť práce tvorí aplikácia metodiky Scrum a LeSS, návrhy na zlepšenie procesu vývoja, finančné zhodnotenie, prínosy a limity návrhu.

Kľúčové slová

projektové riadenie, vývoj softvéru, agilné metodiky, Scrum

Abstract

The proposed thesis deals with agile software development. The teoretical part is devoted to the description of software development, traditional and agile approaches of project management. The analytical part describes current situation of project development and process of software development. Proposal of solution consists of the application of Scrum and LeSS methodology, proposals to improve the development process, financial evaluation, benefits and limits of the proposal.

Key words

project management, software development, agile methodologies, Scrum

Bibliografická citácia

KOYŠOVÁ, Karin. *Zavedení SCRUM v projektu* [online]. Brno, 2022 [cit. 2022-05-09]. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/142924. Diplomová práce. Vysoké učení technické v Brně, Fakulta podnikatelská, Ústav managementu. Vedoucí práce Pavel Juřica.

Čestné prehlásenie

Prehlasujem, že predložená diplomová práca je pôvodná a spracovala som ju samostatne. Prehlasujem, že citácie použitých prameňov sú úplné, že som vo svojej práci neporušila autorské práva (v zmysle Zákona č. 121/2000 Sb., o práve autorskom a o právach súvisiacich s právom autorským).

V Brne dňa 9. mája 2022 .....

podpis autora

Poďakovanie

Chcela by som touto cestou veľmi poďakovať Ing. et Ing. Pavlovi Juřicovi, Ph.D. za vedenie mojej diplomovej práce. Ďalej ďakujem mojim kolegom za všetko čo ma za posledný rok v práci naučili. A v neposlednom rade chcem poďakovať môjmu snúbencovi Petrovi a mojej rodine, za ich podporu počas celého štúdia vysokej školy.

Obsah

Úvod

V dnešnej dobe, kedy sa veľkou rýchlosťou menia požiadavky na produkt ako klienta, tak i zákazníka, má čoraz väčšie miesto agilné riadenie projektov. Práve agilné riadenie je schopné reagovať na meniace sa požiadavky a držať tak krok s klientovou víziou o produkte. To je dôvod, prečo sa agilné metodiky v posledných rokoch tešia veľkej popularite. Čoraz viac firiem využíva či už agilné riadenie projektov, alebo konkrétnu agilnú metodiku. Prípadne aj ich kombináciu, ktorá vyhovuje ich potrebám.

Problém však nastáva u firiem, ktoré nevyužívajú potenciál agilných metodík naplno. Môže za to buď prirýchle nasadenie bez hlbšieho premyslenia, malá informovanosť o metodikách, alebo neochota niečo zmeniť. Pritom správne nasadenie napríklad agilnej metodiky Scrum môže firmám skutočne pomôcť k dosahovaniu lepších výsledkov na projekte, väčšej spokojnosti zákazníka i zamestnancov.

V teoretickej časti tejto diplomovej práce je najskôr vysvetlený vývoj softvéru, jeho životný cyklus a projekt vývoja softvéru. To celé pre lepšie porozumenie procesom v IT. Následne je v teoretickej časti popísaný tradičný prístup riadenia softvéru vo všeobecnosti a sú popísané i konkrétne metodiky. Najväčšia časť však patrí agilnému prístupu a agilným metodikám ako je Scrum, Extrémne programovanie, či LeSS.

Analytická časť obsahuje analýzu súčasného stavu riadenia projektu vývoja softvéru vo vybranej spoločnosti. Je zanalyzovaná oblasť rolí, udalostí, artefaktov a samotný proces vývoja softvéru.

V následnej návrhovej časti bola navrhnutá správna aplikácia metodiky Scrum. Okrem toho návrhová časť obsahuje návrhy pre zlepšenie procesu vývoja, ale aj návrh metodiky pre viacero tímov.

Vymedzenie problému a cieľ práce

Cieľom diplomovej práce je aplikácia metodiky Scrum a návrh na zlepšenie procesu vývoja softvéru na vybranom projekte.

Dôvodom spracovania vybranej témy je skutočnosť nedostatkov v riadení projektu a v celkovom procese vývoja. Na vybranom projekte je využívaný agilný proces riadenia projektu s malými podobnosťami metodiky Scrum. Niektoré časti metodiky sú nesprávne použité, a niektoré chýbajú úplne. V niektorým prípadoch má riadenie projektu dokonca znaky tradičného vodopádového modelu. Sú tak spôsobené problémy v plánovaní vývoja, doručovaní produktu, komunikácii a tiež v iných oblastiach projektu. Tím a produkt sa za posledných pár mesiacov rozrástol, a ďalší nárast sa očakáva v najbližšom roku vývoja. Takto veľký tím už potrebuje presnejšie stanovené procesy, a to je aj dôvod spracovania tejto diplomovej práce.

Táto práca sa venuje návrhom riešenia vo vybraných oblastiach ako je tímová štruktúra, roly, udalosti, artefakty a praktiky. Okrem toho obsahuje aj návrhy na zlepšenie samotného procesu vývoja, návrh na aplikáciu LeSS metodiky i finančné zhodnotenie návrhu.

Všetky návrhy riešenia vychádzajú z analýzy, ktorá sa venuje rovnakým oblastiam ako návrhová časť. Na analýzu boli využité metódy a techniky na zber informácií, konkrétne išlo o priame pozorovanie s aktívnou participáciou na procesoch. Rovnako boli pre zber informácií využívané aj rozhovory s členmi tímov.

1.Teoretické východiská práce

Teoretická časť diplomovej práce sa najskôr venuje vývoju softvéru, vysvetleniu základných pojmov ako je softvér a projekt vývoja softvéru a aj samotnému životnému cyklu vývoja softvéru. V ďalšej časti sa podrobnejšie zameriava na opis tradičných metód riadenia projektu a agilných metód riadenia projektu. Podrobne opisuje konkrétne klasické i agilné metodiky.

1.1 Vývoj softvéru

Softvér je vlastne programové vybavenie počítača, to znamená, že za softvér sa dajú považovať všetky programy, ktoré môžu byť nainštalované v počítači. Ide o všetko s čím pracujeme na obrazovkách počítačov, notebookov, mobilov, tabletov, ale samozrejme aj zariadenia, ktoré displejom a komunikačným prostredím nedisponujú. Základné rozdelenie softvéru je na Systémový a Aplikačný softvér. Na rozdiel od hardvéru má softvér nemateriálnu podobu. Bez softvéru by hardvér bol len súborom elektronických súčiastok, ktoré by nemohli spĺňať svoju funkciu a byť tak užitočné. Práve softvér má na starosti to, aby tieto elektronické súčiastky robili to, čo po nich užívateľ chce (Myslín, 2016).

Produkcia takéhoto softvéru sa nazýva vývoj. Vývoj softvéru je iteratívny logický proces vývoja individuálneho softvéru, pomocou špecifického programovacieho jazyka. Vývoj softvéru stojí niekde medzi tvorbou a výrobou. Výroba sa vyznačuje tým, že sa jedná o rutinu a je viac-menej jasné od začiatku, čo má vzniknúť. Pri vývoji sa nejedná o rutinu, pretože v tomto prípade ide o jedinečné riešenia pre zákazníka. Tvorba zasa postráda systematickosť, ktorá je však pri vývoji dôležitá (Myslín, 2016).

Vývoj softvéru má od výroby hardvéru niekoľko ďalších odlišností:

- Pri vývoji je dôležitejší dizajn a ergonómia. Menej sa kladie dôraz na po predajnú ekonomickú prevádzku, alebo koľko zaberie softvér miesta (toto pravidlo neplatí vždy, výnimkou je napr. softvér pre mobilné zariadenia).
- Pri vývoji je potrebné myslieť viac do budúcna. Musí sa počítať s modifikáciou, rozšírením a prispôsobením softvéru pre väčší počet užívateľov.

- Softvér je potrebné viac individualizovať. Podstatné je, aby riešenia boli šité vždy na konkrétneho zákazníka a boli tak lepšie prispôsobené konkrétnemu prostrediu, kde bude softvér nasádzaný.
- Vývoj nekončí odovzdaním. Je nutné mať na vedomí, že k dokumentom, analýzam a návrhom sa tím bude musieť opakovane vracať.
- Musí sa klásť väčší dôraz na kompatibilnosť. Predovšetkým v minulosti boli softvéry neschopné spolu komunikovať a podpora medzi nimi bola nulová. Dnes musia vývojári dodržiavať pravidlá, konvencie a zásady pri vývoji softvéru (Kadlec 2004).

Na vývoji softvéru pracujú tímy ľudí, ktorých je nutné koordinovať. Každý člen tímu by mal vedieť čo má spraviť, ako je jeho práca integrovaná s prácou kolegov, ale aj termín do kedy musí byť jeho práca dokončená. K tomuto účelu slúžia rôzne formy koordinácie činností, ktorých nástrojom sú práve rôzne metodiky riadenia vývoja (Martinů a Čermák, 2018).

1.1.1 Životný cyklus vývoja softvéru

Životný cyklus vývoja softvéru, častokrát v literatúre označovaný anglickým názvom Software development life cycle (SDLC) je rámec, ktorý pozostáva z niekoľkých krokov. Poskytuje štruktúrovanú sekvenciu fáz, ktoré spoločnosti pomáhajú produkovať vysokú kvalitu softvéru, ktorý je otestovaný a pripravený na "život". Je dôležité pochopiť, čo ktorá fáza obnáša a prečo je dôležitá pre vývoj softvéru ako celku (Newgenapps.com, 2021).

Fázy životného cyklu vývoja softvéru môžu byť v rôznych literatúrach trochu odlišné, ale ich základ je rovnaký. Typický životný cyklus vývoja softvéru sa skladá zo 6 fáz: analýza a plánovanie, definícia požiadavkou, návrh, vývoj, testovanie, nasadenie a údržba.

Analýza požiadavkou a plánovanie

Analýza požiadavkou a plánovanie sú stavebným kameňom životného cyklu vývoja softvéru. Počas analýzy sa získavajú vstupy od zákazníkov, odborníkov z odboru, informácie od zainteresovaných oddelení, prieskumy trhu a iné. Tieto vstupy sú ďalej využité k plánovaniu. Výstupom tejto fázy je štúdia realizovateľnosti (education-wiki.com, 2020)

Definícia požiadavkou

Vo fáze definovania je nutné všetky požiadavky zdokumentovať a následne predložiť zákazníkovi/analytikovi/zodpovednej osobe na schválenie a to v podobe dokumentu-špecifikácia softvérovej požiadavky. Tento dokument obsahuje všetky požiadavky na produkt, ktoré majú byť navrhnuté a vyvinuté počas životného cyklu projektu (education-wiki.com, 2020)

Návrh

Práve špecifikácia softvérovej požiadavky z predchádzajúcej fázy slúži produktovým dizajnérom/architektom na návrhy produktu. Pre tieto účely slúži dokument špecifikácia dizajnu. Takáto špecifikácia je ďalej kontrolovaná a analyzovaná odpovedajúcimi osobami a je vybraný najlepší prístup k dizajnu. Vybraný dizajn obsahuje všetky technické detaily potrebné na implementáciu produktu (education-wiki.com, 2020).

Vývoj

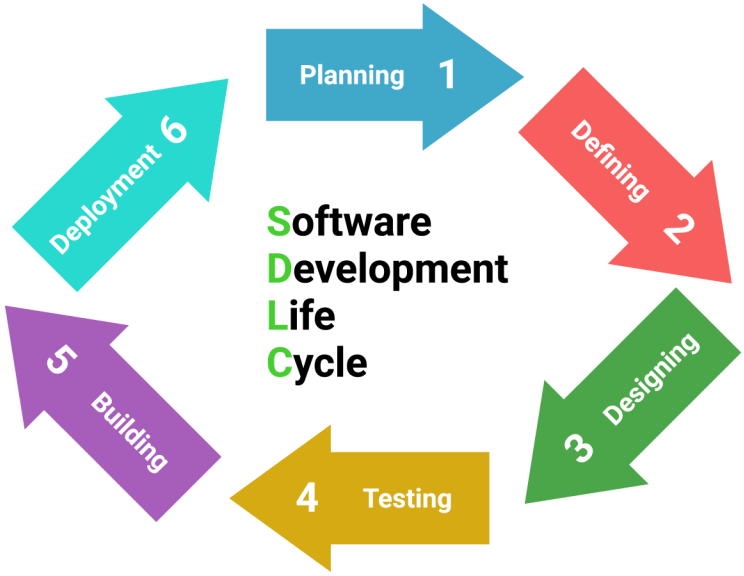
Ako už napovedá názov fázy, v tejto časti sa začína vývoj softvéru. Čím jasnejšia je dokumentácia špecifikácia softvérového požiadavku a špecifikácia dizajnu z predošlých fáz , tým menej problémov nastane pri generovaní kódu. Vývojári by mali robiť všetko pre to, aby bol kód čitateľný a čo najprehľadnejší (education-wiki.com, 2020).

Testovanie

Testovanie je fáza, v ktorej sa zisťujú, nahlasujú a opravujú vady na produkte. Cieľom testovania je teda overenie kódu a detekcia chýb v kóde. Tester vytvára testy a sleduje správanie konkrétneho programu za určitých podmienok. Fáza testovania je veľmi dôležitá, pretože ak tester neotestuje softvér, dá sa povedať, že ho otestujú až zákazníci a vtedy už je príliš neskoro hasiť škody (Kadlec, 2004).

Nasadenie a údržba

Nasadenie softvéru sa vzťahuje na proces spustenia produktu na serveri alebo zariadení. Softvér môže byť počas životného cyklu vývoja softvéru nasadený niekoľkokrát v závislosti od jeho fungovania a výsledkov kontroly chýb. Je zhromažďovaná spätná väzba od prvých používateľov a ak sa objavia nejaké chyby, prípadne požadované funkcie, v rámci údržby ich tím vývojárov opraví (Kadlec, 2004).



Obrázok č. 1: Životný cyklus vývoja softvéru
(Zdroj: vlastné spracovanie)

1.1.2 Projekt vývoja softvéru

„Podľa štandardu IPMA je projekt jedinečný, časovo, nákladovo a zdrojovo obmedzený proces realizovaný za účelom vytvorenia definovaných výstupov v požadovanej kvalite a v súlade s platnými štandardmi a odsúhlasenými požiadavkami“ (Doležal, 2016).

Projekt vývoja softvéru má za cieľ vyvinúť a dodať požadovaný softvér. Pre úspešnú implementáciu projektového riadenia je nutné porozumieť jedinečnému projektovému prostrediu. Preto je dôležité využívanie systémového prístupu, porozumenie spoločnosti, riadenie zainteresovaných strán, správne pochopenie kontextu informačných projektov a prispôsobenie životného cyklu produktu-softvéru (Schwalbe, 2010).

Každý projekt by mal spĺňať nasledovné kritériá:

- jedinečnosť cieľa – nejde o rutinu;
- vyhranenosť – stanovený termín, zdroje a rozpočet;
- potreba realizácie projektovým tímom – potreba špecializovaných pracovníkov;
- komplexnosť a zložitosť – nejde o triviálny problém;
- nadpriemerné riziko – vyplýva zo všetkých vyššie spomínaných atribútov (Doležal, 2016).

Softvérový projekt delíme podľa veľkosti na:

- Malý projekt – trvanie projektu do 3 mesiacov, do 5 ľudí v tíme.
- Stredný projekt – trvanie projektu 3-6 mesiacov, 5-25 ľudí v tíme.
- Veľký projekt – trvanie projektu 6 a viac mesiacov, 25 a viac ľudí v tíme (Doležal, 2016).

Riadenie projektu v akomkoľvek odbore je vždy o troch oblastiach, ktoré spolu úzko súvisia:

- Čas – spočíva v naplánovaní jednotlivých činností tak, aby bol dodržaný termín projektu.
- Náklady – spočíva v riadení 3 oblastí, ktorými sú odhad nákladov, vytvorenie rozpočtu a riadenie nákladov s cieľom dodržať rozpočet.
- Kvalita – týka sa dosiahnutia funkcionality (Komzák, 2013).

1.2 Tradičné metódy riadenia projektov

Tradičné metodiky sa nazývajú tradičné z dôvodu, že tieto metodiky uplatňujú tradičný prístup k vývoju softvéru. Ich označenie za tradičné vychádza aj z toho, že väčšina z nich vznikla skutočne pred dnešnými modernejšími agilnými metodikami. Medzi tieto tradičné prístupy patrí hlavne snaha o čo najmenšiu nepresnosť alebo nejasnosť. Ide v prvom rade o to, aby bolo čo najlepšie možné určiť jednotlivé termíny a požiadavky. Takáto vágnosť sa v tradičných metodikách nenachádza ani v samotných rolách, ktorých je mnoho. Ľudia majú svoju špecializáciu, ktorej sa venujú a mimo ich roly sa v iných činnostiach vývoja softvéru veľmi neangažujú. Striktne stanovené roly sa dajú považovať za výhodu i nevýhodu. Výhodou je, že člen tímu je vysoko špecializovaný a môže sa naplno venovať rozvíjaniu vo svojej role. Za nevýhodu sa dá považovať skutočnosť, že pre splnenie úlohy je nutná kooperácia a komunikácia veľkého počtu ľudí, čo predlžuje a komplikuje vývojový proces (Myslín, 2016).

Vo všeobecnosti sa tradičné metodiky vyznačujú precíznou dokumentáciou. Musí byť zdokumentované všetko od komunikácie so zákazníkom, cez zber požiadavkov a vývoj až po odovzdanie a údržbu. Väčšinou platí, že vývojári začínajú vyvíjať produkt až v okamžiku, keď disponujú kompletnou analýzou a návrhom. Pre zákazníkov to predstavuje nevýhodu v tom, že od samotného zadania nevidia žiadny progres, ani

čiastočné riešenie softvéru. Pre vývojárov je tento proces zasa zdĺhavý z pohľadu častej komunikácie so zákazníkom, spisovaniu a čítaniu dlhej dokumentácie.

Nespornou výhodou tradičných metodík a taktiež aj dôvod ich stáleho využívania je ich poriadok, istota a predvídateľnosť (Myslín, 2016).

Využívanie tradičných metodík dáva zmysel na projektoch:

- väčšieho rozsahu;
- kde spolupracuje viacero tímov;
- kde sú tímy geograficky rozdelené;
- kde sa integruje viac rôznych technológií a systémov;
- kde je potrebné presne dodržať funkcionalitu;
- kde je potrebné presne dodržať termíny;
- kde je presne daný rozpočet (Myslín, 2016).

1.2.1 Vodopádový model

Vodopádový model vznikol v sedemdesiatych rokoch a patrí medzi najstaršie metodiky vývoja softvéru. Priniesol základné členenie softvérového procesu na jednotlivé aktivity a ich logickú postupnosť a v dobe svojho vzniku predstavoval obrovský prevrat v pohľade na vývoj softvéru. Existuje viacero verzií vodopádového modelu, ktoré sa však prakticky líšia len konkrétnou množinou fáz, ktoré definujú. V súčasnosti sa vodopádový model považuje za zastaralý a je využívaný minimálne, avšak z jeho aktivít a ich postupností sa v moderných metodikách vychádza dodnes (Kadlec, 2004).

Metodiky môžeme vo všeobecnosti charakterizovať týmito 3 bodmi:

- Lineárny priebeh – každá fáza nasleduje po predošlej a nie je možné sa vracať späť.
- Jednoznačnosť – vždy je jasné v akej fáze sa tím nachádza.
- Úplné zadanie – do ďalšej fázy je možné vstúpiť, len ak je predošlá fáza s výstupmi dokončená (Myslín, 2016).

Všetky podstatné fázy, ako definícia problému, špecifikácia požiadavkou, návrh a architektúra, implementácia, testovanie a prevádzka, sú vykonávané v poradí

so žiadnymi alebo len minimálnymi iteráciami. Po skončení každej fázy nastáva nasledujúca fáza a nie je možné vykonávať viac fáz naraz.

Fáza definície problému

Vo fáze definície problému je hlavnou úlohou spoznať zákazníka a pokúsiť sa čo najviac vcítiť do jeho role. Je dôležité zistiť jeho potreby, požiadavky a očakávania. V skutočnosti sa v tejto fáze zvyčajne navštívi zákazník na jeho pracovisku a pomocou rozhovoru sa zisťuje v čom by mu mal nový systém uľahčiť prácu, na čo konkrétne ho potrebuje, čo od systému očakáva, kto so systémom bude pracovať a ako vyzerá súčasná situácia bez systému.

Výstupom prvej fázy je dokument Úvodná štúdia, ktorá obsahuje zistené informácie o zákazníkovi, výstupy z rozhovorov o jeho potrebách a motivácii k novému systému. Súčasťou je aj hrubý popis požiadaviek na systém, ktorý je však len orientačný a nenahradzuje samotnú špecifikáciu, ktorá nastáva až v ďalšej fáze.

Fáza analýzy a špecifikácie požiadaviek

Zatiaľ čo prvá fáza mala zmapovať približný popis požiadaviek, druhá fáza sa venuje podrobnejšiemu preskúmaniu toho, čo presne by mal systém spĺňať. Cieľom špecifikácie požiadaviek by mal byť popis aplikácie v jazyku zákazníka a v žiadnom prípade by nemala obsahovať časť o implementácii. Keď je po analýze vypracovaná špecifikácia, vo vodopádovom modeli to znamená, že sa nezmení až do odovzdania systému.

Fáza návrhu a vytvárania architektúry

Fáza návrhu sa zaoberá preštudovaním špecifikácie požiadaviek a na jej základe sa navrhuje vhodná architektúra systému. Architektúra je napríklad rozdelenie aplikácie do vrstiev, ale aj voľba vhodnej technológie a vývojového prostredia. Dá sa povedať, že vo fáze návrhu sa buduje riešenie problému, ktorý bol popísaný v špecifikácii požiadavky.

Fáza implementácie

Predlohou pre implementáciu je špecifikácia, preto by mala byť čo najlepšie zapracovaná a zrozumiteľná. Cieľom je, aby vývojový tím naprogramoval systém tak, aby bol zákazník spokojný. Vo fáze implementácie nie je možné nijako meniť alebo vymýšľať nové funkcie, ktoré nie sú súčasťou špecifikácie.

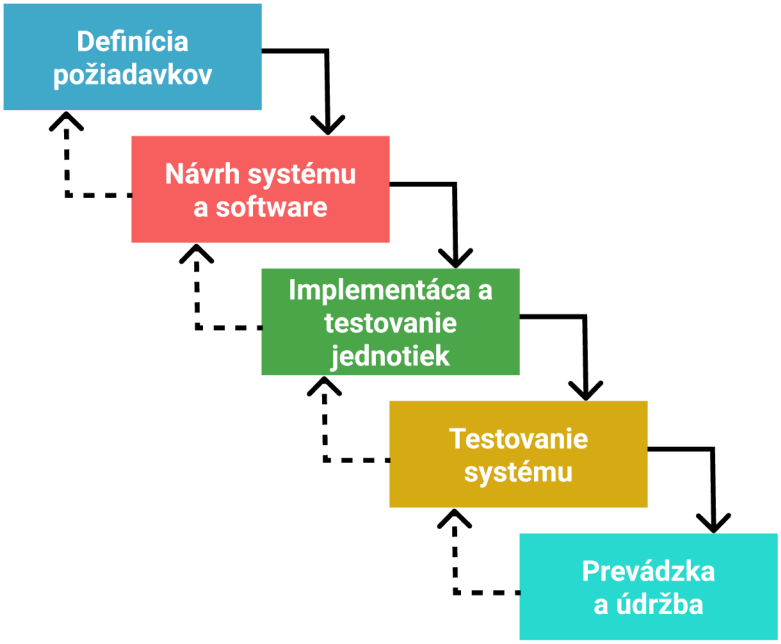
Fáza integrácie a testovania

Cieľom fáze testovania by malo byť, aby softvér fungoval správne a aby robil to, čo od neho zákazník očakáva. V zásade existujú dva druhy testovania:

- metóda bielej skrinky (white-box) – sú testované funkcie na základe znalosti vnútornej štruktúry;
- metóda čiernej skrinky (black-box) – sú testované funkcie na základe očakávaného chovania a špecifikácie.

Fáza prevádzky a údržby

Keď sa softvér dostane u užívateľa do prevádzky, väčšinou s používaním prichádzajú požiadavky na opravu, úpravu, vylepšenie a ladenie softvéru. Týmto sa práve zaoberá posledná fáza údržby.



Obrázok č. 2: Fázy vodopádového modelu
(Zdroj: Vlastné spracovanie podľa: Kadlec, 2004, s. 57)

Ako je vidieť na obrázku hore, prvých šesť fáz je v kategórii definícia a vývoj. V rámci týchto šiestich fáz je možné sa po vodopádovom modeli pohybovať vždy o jednu fázu vpred a vzad. Najdôležitejšie však je, že po vykonaní úprav musia všetky dokumenty so zmenami prejsť opäť schvaľovacím procesom. V tomto prípade sa jedná o jediný prejav flexibility vo vodopádovom modeli.

1.2.2 Špirálový model

Špirálový model vyvinul Barry Boehm v roku 1985 a mal za cieľ eliminovať niektoré nedostatky vodopádového modelu. Boehm do procesu vývoja zaviedol dva koncepty, ktorými sú iteratívny prístup a opakovaná, dôsledná analýza rizík. Vychádzal z niekoľkých modelov, ktoré už nestačili pre vývoj rozsiahlejších projektov. Model patrí do skupiny tzv. rizikami riadených prístupov, čo znamená, že musia byť dôsledne analyzované riziká a problémy. Na základe takýchto analýz sa ďalej rozhoduje o ďalšom smere vývoja.

Pri špirálovom modeli sa jedná o sekvenciu cyklov, kde špirála reprezentuje náklady na vývoj softvéru (čas aj financie). Špirála ďalej ukazuje, akými krokmi produkt prechádza. Pravouhlé súradnice zas ukazujú vývoj vykonávaný v jednotlivých cykloch špirály.

Vývoj podľa špirálového modelu prebieha po špirále od stredu von a špirála je rozdelená na 4 kvadranty. Prvý kvadrant nachádzajúci sa v ľavom hornom rohu je dôležitý pre stanovovanie cieľov. Keď sa vývoj dostane do tohto kvadrantu, je na čase stanoviť ciele, alternatívy a podmienky ďalšieho postupu. Pravý horný kvadrant sa týka analýzy rizík smerom k následnej iterácii. Pravý spodný kvadrant je vyhradený pre realizáciu a posledný kvadrant je vždy nahradený pre nejakú formu plánovania ďalšieho postupu.

Vývoj prebieha tak, že sa v úvodnej fáze identifikujú tri základné skupiny parametrov: ciele, alternatívy a obmedzujúce podmienky. V druhej fáze je potrebné vyhodnotiť alternatívy vzhľadom k obmedzujúcim podmienkam, ktoré vzišli z prvej fázy. Cieľom ďalšej fáze je vytvorenie dostatočne funkčného prototypu. Existujú tri typy prototypov:

- ilustratívny prototyp;
- funkčný prototyp;
- overovací prototyp.

Každá úroveň špecifikovania požiadaviek musí byť vždy zakončená procesom overenia a schválenia. Pri vývoji potom nasleduje pravý dolný kvadrant, v ktorom sa realizuje samotná náplň príslušnej iterácie. Na základe analýz sa tu vykoná špecifikácia požiadavky, návrh, implementácia či testovanie.

1.2.3 Metodika RUP a UP

Metodika Rational Unified Precess je veľmi rozsiahla iteratívna metodika vývoja softvéru. Ide o komerčný produkt, ktorého cena je oproti iným metodikám, ktoré sú prevažne zdarma, naozaj vysoká. RUP podrobne odpovedá na otázky súvisiace s procesom vývoja softvéru, ktoré začínajú na kto, čo, kedy a ako. Ide o komplexný produkt, ktorý poskytuje metodické pokyny, inštrukcie, šablóny, príklady, modely, postupy a tak ďalej. Metodika Rational Unified Precess je skutočne stará metodika, ktorej začiatky siahajú k roku 1987.

RUP zavádza 4 základné fázy vývoja, pričom každá je organizovaná do niekoľkých iterácií. Každá iterácia sa delí na štyri fázy. V prvej fáze zahájenia musí vývojár definovať účel a rozsah projektu. V druhej fáze, ktorou je projektovanie, je úlohou vývojárov analýza potrieb projektu a zákazníka a definovanie architektúry. Fáza realizácie je zas zameraná na vývoj dizajnu aplikácie a tvorbu zdrojových kódov a v poslednej fáze ide o predanie projektu zákazníkovi. RUP definuje množstvo dokumentov a iných formálnych náležitostí, ktoré sa nazývajú medziprodukty a s ich pomocou sú definované činnosti a zodpovednosti pracovníkov.

Veľmi podobnou metodikou je metodika Unified Software Development Process (UP), ktorá sa člení na totožné štyri fázey a definuje päť základných pracovných procesov (stanovenie požiadaviek, analýza, návrh, implementácia a testovanie). Narozdiel od metodiky Rational Unified Precess, je metodika Unified Software Development Process menej prepracovaná a nie tak detailná. Jej veľkou výhodou je, že oproti RUP je úplne bezplatná a tak ju môžu používať spoločnosti s menším rozpočtom.

1.3 Agilné metódy riadenia projektov

Za vznikom agilných metodík stojí skupinka ľudí, z ktorej najznámejšie mená sú: Ken Schwaber, Jeff Sutherland, Alistar Cockburn, Kent Beck, Martin Fowler, Ward Cunningham a ďalší. Títo ľudia sa samostatne snažili o vznik novej metodiky už dlhší čas, avšak agilné metodiky vznikli po ich spolupráci až v roku 2001, konkrétne v štáte Utah (Kadlec, 2004).

Agilné metodiky vychádzajú zo slova “agilný“, čo podľa slovníku znamená:

- vyznačujúci sa pripravenosťou a schopnosťou rýchleho pohybu;
- majúci vynaliezavý a prispôsobivý charakter (Doležal, 2016).

Dá sa povedať, že agilita je iný spôsob života, uprednostňujúci iné hodnoty. Byť agilný znamená žiť podľa agilnej filozofie a je to úplne odlišná firemná kultúra a nálada. Agilná transformácia je pomerne náročná premena, pri ktorej ide o zmenu myslenia a celkového prístupu. Nejde len o nejaký nový proces, rolu alebo nástroj, ide o zmenu kultúry v celej organizácii (Šochová a Kunce, 2019).

Agilný prístup sa snaží o elimináciu chýb tradičného vývoja, ako je zbytočná byrokracia, zbytočná dokumentácia, a o uľahčenie procesov, ktoré vedú ku zmene. Častým kameňom úrazu je nepochopenie agilnej transformácie a v týchto prípadoch nastáva anarchia vo vývoji. Agilný prístup v žiadnom prípade neznamená neporiadok a stratu kontroly nad projektom, ale práve pri správnom nasadení nastáva zefektívnenie práce. Princípom agilných metodík je, že chceme dodať kvalitný softvér a všetko ostatné tomu musíme podriadiť.

Základom celého agilného prístupu je Agilný manifest, ktorý vytvorila už spomínaná skupinka ľudí, ktorá stojí za vznikom Agilného prístupu k vývoju. Nejedná sa o nejaký oficiálny dokument, ale ide skôr o prehlásenie, ktorým by sme sa v agilnom svete mali viesť (Myslín, 2016).

Agilný manifest pozostáva zo 4 častí:

Jednotlivci a interakcie pred procesmi a nástrojmi

Pre agilné metodiky sú dôležitejší jednotlivci, ich potreby a samozrejme aj interakcie. Rôzne nástroje a procesy slúžia len k tomu, aby uspokojili potreby jednotlivcov a aby sa uskutočnili jednotlivé plánované interakcie. Práve toto je prvý rozdiel oproti tradičnému prístupu, ktorý si zakladá na procesoch, ktoré sú častokrát bez ohľadu na to, či sú ľudia na projekte spokojní (Myslín, 2016).

Na druhú stranu Agilný manifest samozrejme nehovorí, že procesy a dohody by nemali existovať, ani že by sa nemali využívať nástroje. Upozorňuje na to, že by si tím mal možnosť nástroje vybrať a používať len tie, ktoré im vyhovujú a pomáhajú im naplniť ich ciele (Šochová a Kunce, 2019).

Fungujúci softvér pred vyčerpávajúcou dokumentáciou

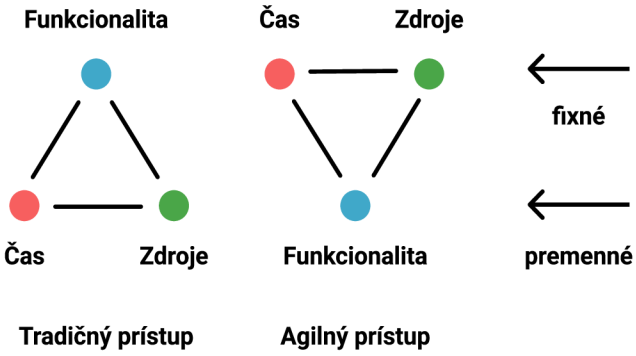
V agilných metodikách je preferovaný fungujúci softvér pred rozsiahlou dokumentáciou, čo samozrejme neznamená, že dokumentácia by mala byť úplne vypustená. Napríklad interná dokumentácia by mala byť stručnejšia a obsahovať kľúčové informácie. Ostatné veci je obvykle efektívnejšie dokumentovať priamo v kóde. Dokumentáciu funkcionalít je možné úplne vypustiť a nahradiť dobrou komunikáciou medzi testermi, analytikmi a vývojármi. Cieľom agilných metodík je teda dokumentáciu len obmedziť na minimum, aby pomer medzi písaním dokumentácie odpovedal hodnote, ktorá je jej čítaním nadobudnutá. Dokumentácia by mala primárne slúžiť ako referencia pre oblasti, ktoré nie sú ľahko pochopiteľné (Šochová a Kunce, 2019).

Spolupráca so zákazníkmi pred vyjednávaním o zmluve

Zmluvy sú určite dôležité i v agilnom svete, ale nemali by nahradzovať komunikáciu a spoluprácu. Pri agilných metodikách je ústna komunikácia s ľudským prístupom braná ako štandardná komunikácia. Spolupráca so zákazníkom by mala byť na pravidelnej úrovni a nie si len odkazovať záležitosti po právnych zástupcoch, v horšom prípade prostredníctvom súdov (Šochová a Kunce 2019).

Reagovanie na zmeny pred dodržiavaním plánu

Pri vývoji sa môže stať, že zákazník príde s úplne inou požiadavkou na produkt, ako pôvodne chcel, a ktorá nebola naplánovaná. Na rozdiel od agilného prístupu, tradičný prístup často takéto zmeny úplne blokuje. V agilnom svete je dôležité ľudské jednanie, ktorého výsledkom je dohoda na zmenách tak, aby bol zákazník spokojný. Samozrejme, že projektové plány sú dôležité, ale fungujú len ako vodítko. Každé plány sa menia a neustále dodržiavanie pôvodných plánov za každú cenu potom vedie k väčšiemu problému, ako ich postupné prispôsobovanie danej situácii (Šochová a Kunce 2019).



Obrázok č. 3: Tradičný vs. Agilný prístup k projektom
(Zdroj: Vlastné spracovanie podľa: Kadlec, 2004, s. 119)

Na obrázku vyššie je na ľavej strane znázornený tradičný prístup, kde tradičné metodiky vychádzajú z nutnosti za každú cenu vyhotoviť dokument špecifikácia požiadaviek. Často sa však stáva, že sú náklady navýšené a projekt sa oneskorí. V tomto prípade je teda funkcionalita fixná a premennými sú čas a zdroje. V prípade tradičných metodík je pre zákazníka dôležitejšie splnenie všetkých požiadaviek než dodržanie rozpočtu (Doležal, 2016).

Na pravej strane obrázku je zobrazený agilný prístup, ktorý naopak pokladá čas a zdroje za fixné a funkcionalita je premenná. Je to z dôvodu, že projekt sa spustí často len s hrubou predstavou o produkte a požiadavky bývajú menené a doplňované. Preto musí mať agilný projekt od začiatku stanovené náklady a čas (Doležal, 2016).

Na to, aby sa dalo riadenie projektu považovať za agilné, nemusí byť aplikovaná žiadna z metodík. Rozhodujúce je, či sú uplatňované nasledujúce princípy:

- Iteratívny vývoj – celková práca na projekte je rozdelená do etáp, ktoré trvajú vždy rovnakú dĺžku. Zvyčajne je dĺžka iterácie 2-4 týždne.
- Inkrementálne dodávky – nové funkcie na produkte sú dodávané postupne. Projektový tím sa sústreďuje vždy na menšiu časť, ktorá je potom dodaná zákazníkovi. Každý inkrement by mal predstavovať funkcionalitu, ktorá má pridanú hodnotu pre zákazníka.
- Multifunkčné tímy – Pri agilnom riadení je vyžadovaná úzka spolupráca medzi členmi tímu, a pramení to z dôvodu veľmi obmedzeného času na dodanie výstupov. Preto je podstatná vnútro tímová komunikácia.

- Zapojenie zákazníka – V agilnom projekte sa očakávajú zmeny v projekte ai v zadaní, a preto je veľmi dôležitá úzka spolupráca medzi zákazníkom a projektovým tímom. Zákazník by mal byť k dispozícii po celú dobu projektu a dávať pravidelnú spätnú väzbu tímu. V tradičnom modeli je to naopak a častokrát sa stáva, že zákazník je prítomný len pri zadávaní a preberaní projektu na jeho konci.
- Agilné chovanie – Všetci členovia tímu by mali agilite porozumieť a je od nich očakávané napĺňanie agilných princípov. Takéto správanie vyžaduje určitú osobnú vyzretosť ľudí v tíme.
- Pravidelná revízia požiadaviek – Tento princíp úzko súvisí s predošlým princípom a vychádza z toho, že pri agilných projektoch často na začiatku neexistuje presná predstava o produkte, prípadne sa často mení alebo upresňuje. Aby boli včas podchytené nové požiadavky, je dôležité robiť pravidelnú revíziu scopu a ujasniť si tak požiadavky so zákazníkom (Doležal, 2016).
- Opakované a priebežné testovanie – Vzhľadom k už spomínaným častým zmenám je podstatné aj priebežné overovanie správností systému. Testy by mali byť automatizované a testovacie scenáre by mali byť napísané ešte pred implementáciou (Kadlec, 2004).

1.3.1 Extrémne programovanie

Extrémne programovanie, alebo často v literatúre pod skratkou XP je prístup, ktorý je dnes už menej využívaný a je nahradzovaný SCRUM metodikou. Kent Beck s ním oficiálne prišiel v roku 1999, kedy ho po prvý raz definoval v rozhovore pre magazín C++. Extrémnemu programovaniu predchádzala Beckova nespokojnosť s nesprávnym vývojovým procesom v spoločnosti Crysler Corp. Preto sa začal venovať návrhu metodiky, ktorá by mala umožniť flexibilnejší a prispôsobivejší vývojový proces (Kadlec, 2004).

Táto metodika odvodzuje svoj názov od toho, že privádza do extrému mnohé princípy, či už tradičných alebo agilných metodík. Častokrát, keď tímy prechádzajú na túto metodikyu z tradičného prístupu, prídu im princípy a myšlienky extrémneho programovania až šokujúce a naozaj extrémne. Avšak cieľ extrémneho programovania je ten istý ako pri hociktorom inom prístupe a to je dodať fungujúci softvér (Myslín, 2016).

Prvým veľkým rozdielom je dĺžka iterácie. Pri tradičných modeloch sú tímy zvyknuté na iterácie trvajúce niekoľko mesiacov a pri agilných projektoch sa jedná o niekoľko týždňov. V extrémnom programovaní trvá jedna iterácia niekoľko dní, v tých najextrémnejších prípadoch len jeden deň. To znamená, že každý deň sa môže skončiť iterácia a do produkčného prostredia je implementovaná nová funkcia, niekedy až niekoľkokrát denne. Ďalšou veľkou odlišnosťou hlavne oproti tradičnému prístupu je testovanie. V tradičných modeloch sa testuje až na konci iterácie, avšak pri extrémnom programovaní sa testuje prakticky hneď a stále. Funkcia je navrhnutá, naimplementovaná, otestovaná a hneď nasadená. Veľkým rozdielom je aj zapájanie zákazníka, keďže pri XP sa zákazník stáva súčasťou vývojového tímu. Asi najextrémnejšou odlišnosťou extrémneho programovania od hociktorej inej metodiky sú zmeny. Pri extrémnom programovaní je bežné, že sa niekoľkokrát za deň zmenia požiadavky (Myslín, 2016).

Extrémne programovanie je založené na 4 hodnotách:

1. Komunikácia – členovia tímu spolu osobne komunikujú všetky záležitosti. Pracujú spoločne na návrhu riešení problémov, navrhovaní, kódovaní aj na testovaní. V rámci XP sú definované postupy, ktoré priamo nabádajú tím ku komunikácii.
2. Jednoduchosť – tím pracuje a sústreďuje sa len na veci, ktoré sú pre nich v prítomnosti podstatné a je potrebné ich urobiť. Neplánujú príliš dopredu a nepremýšľajú nad tým, čo by mohol zákazník ešte navyše chcieť. Postupnými menšími krokmi, pravidelnou spätnou väzbou a dôverou tomu v čo veria, sa snažia minimalizovať chyby.
3. Spätná väzba – počas krátkych iterácií je pre tím podstatná spätná väzba a prípadné zmeny zapracováva podľa požiadaviek.
4. Odvaha – XP vyžaduje od celého tímu kus odvahy. Či už sa jedná o odvahu prijať zodpovednosť za zlý odhad, odvahu prijať zmeny, alebo odvahu na robenie vecí inak (Šochová a Kunce, 2019).

Z vyššie spomínaných hodnôt vychádza z Extrémneho programovania aj niekoľko princípov. Jedným z nich je párové programovanie, kde kód určený pre produkčné prostredie vždy spoločne vyvíjajú dvaja programátori. Jeden z nich píše kód a druhý všetko sleduje a premýšľa ako veci vylepšiť. Ďalším princípom je napríklad stabilný

štyridsaťhodinový pracovný týždeň, ktorý vychádza z toho, že pre Extrémne programovanie sú dôležití odpočinutí vývojári, ktorí nemajú časté pracovné nadčasy. Zaujímavým princípom je spoločné vlastníctvo, ktoré je presným opakom väčšiny metodík. V XP sú za kód rovným dielom zodpovední všetci, a ktokoľvek môže čokoľvek upravovať. Toto samozrejme platí pri dodržaní nejakých stanovených pravidiel. Veľmi dôležitým, ale určite nie posledným princípom je testovanie, ktoré je považované v Extrémnom programovaní za základ. Platí, že ak máme čokoľvek naprogramovať, musíme najprv vymyslieť to ako to otestujeme (Myslín, 2016).

1.3.2 Lean Development a Kanban

Lean Development je ďalšou agilnou metodikou, lepšie povedané súborom pravidiel, ktorých používanie by malo zefektívniť a zrýchliť vývojový proces. Na rozdiel od ostatných metodík, ktoré boli vytvorené pre vývoj softvéru, Lean Development má korene vo výrobnej sfére. Lean Development a rovnako aj Kanban pôvodne pochádzajú z Japonska. Kanban bol pôvodne využívaný na riadenie počtu ľudí v chráme. Lean Manufacturing alebo aj štíhla výroba zas vznikla v osemdesiatych rokoch v prostredí firmy Toyota (Kadlec, 2004).

Samotný Lean Development spočíva v desiatich pravidlách a siedmich princípoch, ktorých dodržovanie sľubuje efektívnejší a rýchlejší proces vedúci k lepšiemu uspokojeniu zákazníkových požiadaviek. Táto metodika naznačuje, akým spôsobom by mal vývoj prebiehať, ako by mal byť vykonávaný, čoho by sa mal vyvarovať a kam by mal smerovať.

Štíhly vývoj je často kombinovaný aj s Kanban procesom, o ktorom sa dá povedať, že je jedným z nástrojov Lean Developmentu. Kanban je veľmi flexibilný proces a v podstate nič neprikazuje. Stačí, ak sú dodržané 3 princípy:

- Obmedziť rozpracovanú prácu (work in progress)
- Minimalizovať čas priechodu (lead time)
- Vizualizovať progres.

Podstatné je vytvoriť prehľadnú Kanban tabuľu a rozdeliť ju na časti, napríklad: Backlog, In progress, Dones, a obmedziť počet lístočkov v každej časti. Ak je kapacita naplnená, musí sa najprv dokončiť vec, ktorá už je rozpracovaná a až potom je možné pridať novú.

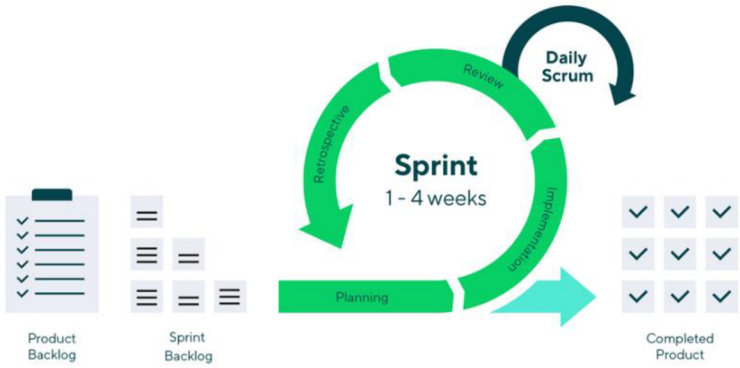
V úspešných implementáciách sa nevyužíva len Kanban, ale vždy v kombinácií s niečím iným. Čistý princíp Kanbanu sa doplňuje o pravidelnú Retrospektívu, Daily Scrum, Scrum (Kanbam) Mastrom, Product Ownerom a tak ďalej. Princípy z Kanbanu ako je obmedzená rozpracovanosť User stories, alebo vizualizácia Kanban tabuľou, sa často využívajú aj pri Scrum procese (Šochová a Kunce 2019).

1.3.3 Scrum

Ken Schwaber a Jeff Sutherland prvýkrát predstavili Scrum na konferencii OOPSLA v roku 1995. Okrem nich na vytváraní tejto najobľúbenejšej agilnej metodiky, tak ako ju poznáme dnes, pracovali aj Jeff McKenn, John Scumniotales, Mike Smith a ďalší. Ich cieľom bolo vyvinúť flexibilnú metodiku, ktorá sa dokáže lepšie vyrovnať s meniacimi sa požiadavkami a ktorá vylepší produktivitu procesu vývoja softvéru (Kadlec, 2004).

Scrum je pomerne jednoduchý framework, ktorý definuje len časti potrebné k implementácii jeho teórie. Je založený na kolektívnej inteligencii, empirisme a štíhlom myslení. Pri empirizme konkrétne znalosti vychádzajú zo skúseností a štíhle myslenie má na starosti obmedziť plytvanie a zamerať sa len na to dôležité. Scrum využíva iteratívny, inkrementálny prístup k optimalizácii predvídateľnosti a k riadeniu rizika.

V Scrume existuje niekoľko základných prvkov: postupy, role, artefakty a udalosti. Tento framework využíva iterácie, ktoré sú nazývané šprinty a majú určitú dobu trvania. Na začiatku každého scrumového projektu sa zbierajú požiadavky na produkt a ich prioritizácia. Miesto, ktoré obsahuje tieto všetky požiadavky sa nazýva Product backlog. Položky z Product backlogu sú potom podľa ich prioritizácie počas plánovania presunuté do aktuálneho Sprint Backlogu. Behom Sprintu sa tím stretáva na pravidelných daily stand-up poradách. Na konci každého Sprintu by mal byť hotový produkt (časť produktu), ktorý prinesie hodnotu zákazníkovi. Sprint končí predaním produktu (Review) a tímovou Retrospektívou. Scrum je framework vhodný pre 1-2 tímy. Ak na produkte pracuje viacero tímov, je vhodné zvoliť nejakú zo škálovaných metodík (Doležal, 2016).



Obrázok č. 4: Vizualizácia Sprintu
(Zdroj: wrike.com)

Scrum roly

Základnou jednotkou Scrumu je malý scrumový tím, založený na princípe samo organizovanosti, transparentnej komunikácii a otvorenej kultúre. Scrumový tím sa skladá z vývojárov, Product Ownera a Scrum Mastera a jeho veľkosť by nemala presahovať 9 členov tímu. V tíme neexistuje žiadana hierarchia a všetci sú zodpovední za všetky činnosti súvisiace s produktom (Šochová a Kunce 2019).

Vývojári

Za Scrum team membera alebo aj „radového" člena tímu sú považovaní vývojári. Jeho veľkosť býva väčšinou do sedem ľudí a v rámci tohto tímu nie sú stanovené ďalšie role. V prípade, že sa počet členov rapídne zväčšuje, nastáva rozdeľovanie tímu do menších celkov tak, aby sa zachovali prínosy osobnej interakcie. Vývojový tím musí byť samo-organizovaný a multifunkčný, to znamená, že sú v ňom zastúpené všetky potrebné odbornosti, aby boli činnosti paralelné. Tím sa snaží dosiahnuť čo najväčšiu flexibilitu, efektivitu a zdieľanú znalosť(Doležal, 2016).

Vývojári musia vždy:

- vytvárať plán Sprintu a Backlog Sprintu;
- zaistiť kvalitu dodržovania Definition of Done (Definíca hotového);
- prispôsobiť svoj plán smerom k cieľu Sprintu (Schwaber a Sutherland, 2020).

Product Owner

Rolu Vlastníka produktu zastrešuje vždy len jedna osoba a jeho úlohou je zastupovať zákazníka. Samotný Product owner definuje ako bude produkt vyzerať, jeho funkcionality, vízie a čo vlastne bude robiť. S konečnou platnosťou schvaľuje hotový produkt, alebo akceptáciu produktu odmieta. Product Owner je zodpovedný za efektívnu správu Backlogu, z toho vyplývajú tieto úlohy:

- rozvíja a otvorene komunikuje Cieľ produktu
- vytvára, riadi, prioritizuje a komunikuje položky Backlogu (Schwaber a Sutherland, 2020).

Scrum Master

Scrum Master je rola, ktorá zodpovedá za zavedenie Scrumu a pomáha porozumieť jej teórii a praxi nielen v tíme, ale aj v celej organizácii. Na rozdiel od Product Ownera musí Scrum Master disponovať predovšetkým mäkkými dovednosťami. Scrum Master tím podporuje, motivuje, chráni a smeruje tím tam, kde ho chce mať. Odpovedá za efektívnosť tímu a v jeho náplni je:

- odstraňuje prekážky, ktoré tímu bránia v dosiahnutí cieľa;
- facilituje udalosti a zabezpečuje ich produktivitu;
- koučuje členov tímu v multi-odborovosti a seba riadení;
- pomáha zefektívňovať všetky artefakty;
- podporuje Product Ownera v tom, aby sa kvalitne ujal svojej role (Schwaber a Sutherland, 2020).

Udalosti

Každá udalosť v Scrume predstavuje formálnu príležitosť na empirické pozorovanie a následné prispôsobenie Scrumových artefaktov. Tieto udalosti sú špecificky navrhnuté a chronologicky zoradené tak, aby umožňovali potrebnú transparentnosť.

Sprint Planning

Sprint je zahájený plánovaním šprintu na rozvrhnutie práce, ktorá má byť v Sprinte vykonaná. Tento výsledný plán spoluvytvára celý Scrumový tím. Ide o veľmi dôležitý míting, ktorého cieľom je pripraviť plán Sprintu. Plán sa skladá z definície cieľa Sprintu, vytvorenia Sprint Backlogu a definície potrieb. Cieľ Sprintu navrhuje a schvaľuje Product Owner. Na základe tohto cieľa si vývojári zostavujú Sprint Backlog. Scrum

Master moderuje míting a spolu s vývojovým tímom pripomienkuje návrhy Product Ownera (Myslín, 2016) .

Zaujímavou technikou, ktorá sa využíva pri plánovaní alebo pri Refinemente je plánovací poker. Využíva sa na ohodnotenie jednotlivých user stories. Najprv sa predstaví user story, vznikne priestor na otázky, dovysvetlenia a následne sa pomocou kartičiek ohodnocuje náročnosť danej story. Tí účastníci s najvyššou hodnotou vysvetlia dôvod svojho výberu a prebieha diskusia až sa príde k záveru. Ak je to potrebné, hra sa môže opakovať aj viackrát pre jednu user story (Doležal, 2016).

Daily Scrum meeting

Daily Scrum meeting alebo aj Stand-up je krátka udalosť, ktorá trvá cca 15 minút pravidelne každý deň. Účelom je preskúmať postup smerom k cieľu Sprintu a vytvorenie akčného plánu na ďalší pracovný deň. Prítomnosť členov tímu, narozdieľ od Scrum Mastera a Product Ownera, je povinná. Tím si môže zvoliť ľubovoľnú štruktúru a techniku tejto udalosti, ale zvyčajne sa využíva tzv. Koliesko, kde každý člen tímu v skratke predstaví:

- čo robil predchádzajúci deň a s akým výsledkom;
- čo plánuje robiť aktuálny deň;
- či vie o nejakej prekážke, ktorá by ho brzdila v plynulej práci (Schwaber a Sutherland, 2020).

Sprint Review

Sprint Review alebo aj vyhodnotenie Sprintu je jedným z dvoch udalostí uskutočňujúcich sa na konci Sprintu. Je to zároveň aj jediná udalosť, ktorej sa môžu, alebo skôr sa očakáva, že sa budú zúčastňovať Stakeholederi (zákazník, investor, produktový manažér). A to z dôvodu, že na Review majú možnosť vidieť výsledok Sprintu a môžu si produkt aj vyskúšať. Úlohou Product Ownera spolu so Scrum Masterom je spolu prezentovanie výsledku Sprintu a zodpovedanie otázok a Scrum Master okrem toho samozrejme moderuje celý míting. Vývojový tím zodpovedá otázky a zainteresované strany majú možnosť testovať funkcionalitu a vznášať námietky k produktu (Myslín, 2016).

Retrospektíva

Retrospektíva slúži na to, aby sa tím pozrel na uplynulý Sprint kriticky a definoval, aké procesy je potrebné zlepšiť. Trvanie je väčšinou 1-2 hodiny (podľa veľkosti tímu) a môže a nemusí nadväzovať na Review meeting. V rámci Retrospektívy je možné hodnotiť plnenie úloh, ale aj prístup jednotlivých členov tímu. Pri tomto meetingu je veľmi podstatná jeho pravidelnosť a je dôležité, aby si členovia tímu zvykli na pravidelnú spätnú väzbu a naučili sa meniť procesy, praktiky a zvyklosti (Šochová a Kunce, 2019).

Rola Product Ownera je zhodnotiť priebeh Sprintu, vyjadriť sa k hodnoteniu a navrhnúť opatrenia. Scrum Master celý meeting moderuje a taktiež hodnotí, vyjadruje sa k hodnoteniu a navrhuje zmeny. Členovia vývojového tímu prichádzajú s opatreniami a vyjadrujú sa k hodnoteniu (Myslín, 2016).

Existuje množstvo formátov, ako viesť Retrospektívu a je dobré tieto formáty čas od času obmieňať. Obľúbeným formátom je „koliesko“, kde každý člen postupne odpovie na to, čo sa mu páčilo, čo sa mu nepáčilo, a čo by chcel zaviesť nové. Populárne sú Retrospektívy s lepiacimi papierikmi, loď, ESVP, časová os a iné. Dnes sú veľmi populárne aj online nástroje, ktoré ponúkajú mnoho šablón s rôznymi formátmi najrôznejších Retrospektív (Šochová a Kunce, 2019).

Backlog Refinement

Backlog Refinement pri dobre zabehnutých tímoch väčšinou neprebieha formou meetingu, ale ako aktivita, na ktorej tím pracuje distribuovane, podľa toho, kto má zrovna čas. V prípade, že tím s Backlog Refinementom len začína, je dobré z neho spraviť meeting 1-2 krát za Sprint, podľa potrieb tímu. Jeho cieľom je to, aby celý tím porozumel Product Backlogu, vízii, položkám v Backlogu a ich prioritám. Je dôležité, aby z Backlog Refinementu vypadli položky, ktoré sú ohodnotené story pointmi a spĺňajú definition of ready. Vo firmách, kde už majú skúsenosti s Refinementom prebieha zväčša raz za kvartál, kde sa stretnú na workshope tímy spolu so stakeholdermi a pracujú spoločne na Backlogu. Ak je Refinement správne zvládnutý, z plánovania Sprintu sa stáva len formalita a jeho dĺžka je maximálne 30 minút (Šochová 2018).

Artefakty a praktiky

Artefakty (nástroje) a praktiky Scrumu reprezentujú prácu a jej hodnoty rôznymi spôsobmi, ktoré sú užitočné pre poskytovanie transparentnosti a ďalej sú príležitosťou

pre kontrolu a adaptáciu. Podľa Scrum sprievodcu existujú 3 Artefakty: Product Backlog, Sprint Backlog a Inkrement.

Product backlog

Product Backlog je súhrn všetkých informácií o produkte a skladá sa z položiek Backlogu. Takýto zoznam čo chceme na produkte zhotoviť má na starosti Product Owner, ale musí byť prístupný všetkým členom tímu. So zostavovaním a udržovanám Produktového Backlogu mu pomáhajú stakeholderi a členovia tímu. Product Owner má však posledné slovo a je zodpovedný za jeho zmysluplnosť a prioritizáciu (Šochová a Kunce, 2019).

Záväzkom Product Backlogu je cieľ produktu, ktorý popisuje budúci stav produktu, voči ktorému tím plánuje. Backlog produktu zas v čase definuje čo splní stanovený produktový cieľ. Takýto cieľ je dlhodobou úlohou pre Scrumový tím a najprv musí byť cieľ naplnený, aby sa mohol stanoviť nový (Schwaber a Sutherland, 2020).

Na spravovanie Backlogu sú využívané rôzne informačné systémy, Medzi najznámejšie patrí Jira, Azure DevOps alebo Redmine.

Sprint Backlog

Od produktového Backlogu je odvodený Šprint Backlog a to tým že na začiatku šprintu behom plánovania sa do neho presunú položky s najvyššou prioritou. Keďže sa plánuje šprint, musí byť stanovený aj už spomínaný cieľ šprintu. Pre správu Backlogu sa aj v tomto prípade využívajú už spomínané informačné systémy. Sprint Backlog je tvorený vývojovým tímom. (Doležal, 2016).

Inkrement

Inkrement alebo aj novo nasaditeľný produkt zahŕňa všetky položky Product Backlogu, ktoré boli doteraz implementované. Patrí do neho každá doposiaľ realizovaná položka Sprint Backlogu a tiež tie funkcionality, ktoré boli implementované v poslednom Sprinte. Všetky tieto funkcionality musia spĺňať definíciu „done“, aby aj novo nasaditeľný produkt bol považovaný za hotový a bol použiteľný zákazníkom tak, aby mu prinášal hodnotu.

Sprint

Sprint je špeciálny názov iterácie pre Scrum a je to teda cyklus, ktorý sa počas vývoja softvéru opakuje viackrát. Ide o zvyčajne 2-6 týždňov dlhý fixný časový úsek. Mal by byť aspoň tak dlhý, aby tím stihol dokončiť a odprezentovať nejakú čas na Review. Cieľom Sprintu nie je dodať všetky položky zo Sprint Backlogu, ale dosiahnuť cieľ Sprintu (Sprint Goal). Sprint Goal je taká malá vízia na jeden Sprint. Správny cieľ Sprintu by mal adresovať potreby zákazníkov, mal by byť zameraný na hodnotu a nie na funkcionalitu. Cieľ Sprintu si stanoví Product Owner spolu s tímom na plánovaní Sprintu a počas aktuálneho Sprintu sa cieľ nemení. Počas Sprintu však je možné meniť položky v Backlogu, ak to pomôže tímu k naplneniu cieľa (Šochová a Kunce, 2019).

Položka Backlogu

Každá položka Backlogu predstavuje samostatný pracovný balík. Položky Backlogu tvoria také funkcionality, ktoré môžu priniesť zákazníkovi hodnotu. Bez ohľadu, či sú položky zaznamenávané pomocou User stories, alebo iným spôsobom, musia byť prezentovateľné zákazníkovi a samozrejme na ne potom musí tím získať spätnú väzbu. Jednotlivé položky sú potom priradené členom v tíme, ktorí na nich budú počas Sprintu pracovať (Šochová a Kunce, 2019).

Use story a Task

Najčastejšie sa jednotlivé položky Backlogu definujú pomocou tzv. User story, ktorú je možné preložiť ako príbeh užívateľa. Z takejto funkčnej špecifikácie by malo byť možné zistiť, kto je príjemcom hodnoty, čo má produkt robiť a aké očakávanie je tým plnené. User stories by mali byť unikátne, prínosné, odhadnuteľné a testovateľné. Rozpadnutím User story na menšie celky dostaneme tasky (úlohy) (Doležal, 2016).

Epic a Témy

Ak je potrebné udržať kontext a jednotlivé User stories pohromade, využíva sa na to tzv. Epic. Pri definovaní Produkct Backlogu je lepšie začať práve od takýchto väčších celkov. Epiky sa ako také nedajú naplánovať do Sprintu a ani ohodnotiť, pretože sú príliš veľké. V Scrume sa v rámci Refinementu Epiky delia na menšie, konkrétnejšie položky Backlogu-User stories. Takéto väčšie celky ako sú Epiky sa spojujú dohromady do Tém. Témy idú naprieč funkcionalitami a združujú jednotlivé položky Backlogu logicky z pohľadu businessu (Šochová a Kunce, 2019).

Bug

Bug je označenie pre chybu. Je súčasťou Backlogu a Product Owner by mal Bugy prioritizovať podľa biznisovej hodnoty. Takéto Bugy do Backlogu pridávajú väčšinou testery po testovaní, zákazník alebo hociktorý člen scrumového tímu (Šochová, 2018).

Definition of Done a Definition of Ready

Definition of Done alebo aj definícia hotového definuje globálne podmienky, ako sa pozná, že je položka Backlogu hotová. Vzniká dohodou medzi tímom a Porduct Ownerom a platí na všetky položky Produktového Backlogu. Položka, ktorá splní Definition of Done je pripravená na Review. Takéto kritéria musí splňovať každá hotová funkcionalita z Backlogu a je pre všetky položky rovnaká. Definition of Ready alebo aj definícia pripraveného nám zas udáva, aké podklady musia byť poskytnuté a vyjasnené predtým, než sa začne pracovať na danej úlohe. (Doležal, 2016).

Story points

V klasickom projektovom riadení sa na odhad pracnosti úloh využívajú väčšinou man/days (človeko-dni). V Scrume je objem práce stanovený na bodovacej škále pomocou tzv. story pointov. Najvyužívanejšia škála je 2, 4, 8, 16 a tak ďalej. Pri vyšších číslach je však vhodnejšie rozpadnúť User story na menšie celky a tie ohodnotiť.

Scrum Tabuľa

Scrumová tabuľa slúži pre dobrú vizualizáciu Sprintového Backlogu. Vychádza z Kanbanovej tabule a tiež sa snaží minimalizovať rozpracované úlohy. Všeobecne stačí, keď má tabuľa 3 stĺpce: Sprint Backlog, In progress (prebieha) a Done (hotové). Scrumová tabuľa slúži okrem iného aj na vizualizáciu Taskov, ktoré vypadli z rozpadu User story. Položky sú presúvané do stĺpcov, podľa ich aktuálneho stavu a než sa začne pracovať na novej úlohe, musí byť úloha dokončená alebo predaná ďalej. Každý člen tímu sa môže odlišovať inou farbou pre lepšiu prehľadnosť tabule (Šochová a Kunce 2019).

Burndown chart

Takýto graf znázorňuje, akú veľkú časť práce scrumový tím už vykonal, a koľko mu ešte zostáva dokončiť do konca šprintu. Burndown chart znázorňuje tempo práce a pomáha tímu udržať rýchlosť dodávky na vysokej úrovni.

Velocity

Velocita je súčet ohodnotenia dokončených User stories za dokončený Sprint. Nedokončené User stories sa do velocity nezapočítavajú, a musia byť presunuté do Product Backlogu. Výška velocity by mala byť pre Sprinty viac-menej konštantá a podľa nej sa dá plánovať záväzok na ďalší Sprint. Táto rýchlosť sa nepočíta za jednotlivých členov tímu ale dokopy za tím a Sprint (Šochová a Kunce, 2019).

Churn a Balance

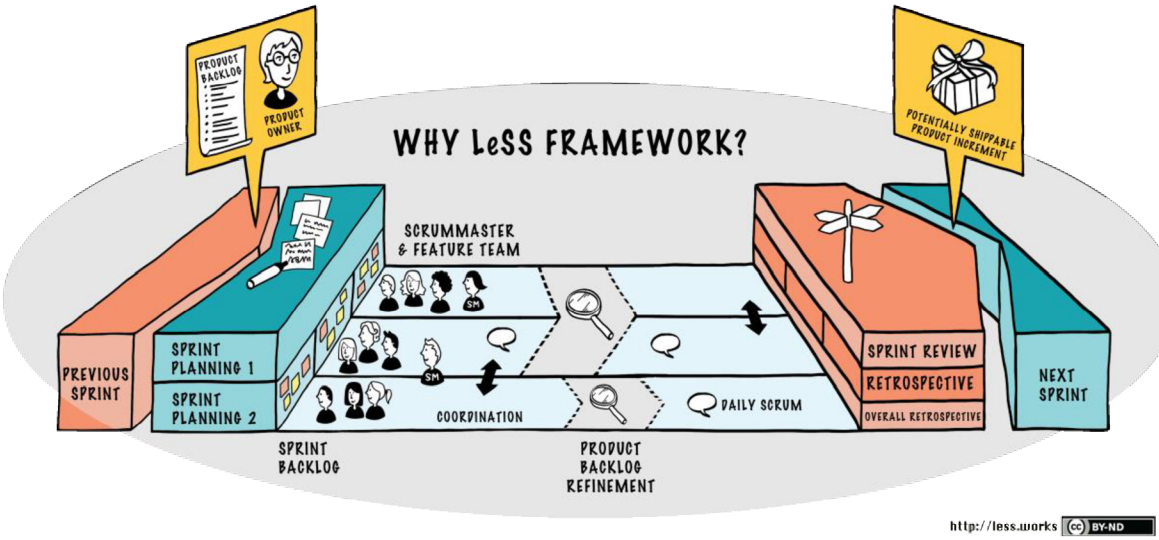
Churn je metrika využívajúca sa v agilných metodikách, ktorá predstavujúca narušenie Sprintového Backlogu. Niekedy je takéto narušenie Sprintu nevyhnutné, avšak nemalo by k nemu dochádzať často a v príliš veľkej miere. Balance je zase metrika, ktorá sleduje to, že keď je Sprint narušený a boli v jeho priebehu pridané user stories, malo by byť rovnaké množstvo aj odobrané. Táto metrika je teda rozdiel medzi pridanými a odobranými položkami, podelené záväzkom na začiatku. Z toho vyplýva, že narušenie Sprintového Backlogu je v poriadku, ak sa dodrží istý balanc (Myslín, 2016).

1.3.4 LeSS

Keď je produkt príliš veľký a pracuje na ňom viacero tímov, Scrum začína byť nedostatočný. V takomto prípade je vhodné zvoliť framework pre škálovaný Scrum. Jedným z nich je Large-Scale Scrum, skrátene LeSS. Vznikol niekedy v roku 2005 a teší sa veľkej popularite.

LeSS rovnako ako aj pri klasickom Scrum má len jednoho Product Ownera. Jeden Scrum master môže slúžiť pre 1-3 vývojové tímy a zameriava sa tam na celkový organizačný systém. Prvý rozdiel nastáva pri Sprint planningu, ktoré je rozdelené na Sprint planning 1 a Sprint Planning 2. Prvého plánovania sa okrem Product Ownera zúčastňujú všetci členovia tímov. Sami si rozdeľujú položky z Product Backlogu a diskutujú o možnostiach spolupráce. Druhé plánovanie už prebieha pre každý tím samostatne. Daily Scrum meeting tiež vedie každý tím nezávisle, hoci člen tímu 1 môže napríklad sledovať denný scrum tímu 2, aby sa zvýšilo zdieľanie informácií. Jedinou požiadavkou v LeSS je Product Backlog Refinement pre jeden tím, rovnako ako v Scrume s jedným tímom. Ale bežnou a užitočnou variáciou je Product Backlog Refinement pre viac tímov, kde sú dva alebo viac tímov spolu v rovnakej miestnosti, aby sa zlepšilo učenie a koordinácia. Ďalšou

udalosťou je Sprint Review, ktorá okrem vlastníka produktu zahŕňa ľudí zo všetkých tímov, relevantných zákazníkov/používateľov a ďalšie zainteresované strany. Okrem klasickej Retrospektívy pre každý tím v LeSS je aj celková Retrospektíva, ktorej účelom je skôr preskúmať zlepšenie celkového systému, než sa zamerať na jeden tím. Maximálne trvanie je 45 minút za týždeň Sprintu. Zahŕňa vlastníka produktu, Scrum Masters a rotujúcich zástupcov z každého tímu (Larman a Vodde, 2017).



Obrázok č. 5: LeSS framework
(Zdroj: less.works)

1.4 Zhrnutie teoretických východísk práce

Celá teoretická časť práce slúži ako základ pre spracovanie analýzy súčasného stavu a následne vlastného návrhu riešenia. Na začiatku teoretickej časti bol v krátkosti rozobratý vývoj softvéru, jeho životný cyklus a aj to, čo je projekt vývoja softvéru. Ďalšia časť sa zameriava na tradičné metodiky riadenia vývoja softvéru, aj s opisom 4 konkrétnych metodík. Posledná a najdlhšia časť je vyhradená pre agilné metodiky riadenia softvéru.

Práve v agilných metodikách vidím veľký potenciál a preto som sa rozhodla venovať sa im aj v návrhovej časti, konkrétne Scrum a Less metodike. Tieto metodiky sú v súčasnosti veľmi obľúbené a ich popularita svedčí aj o ich skvelom prístupe k vývoju softvéru. Najviac sa stotožňujem s prístupom ku Scrum metodike autorov Šochovou a Kuncem, ktorí opisujú celú metodiku na veľmi praktických príkladoch. Vo svojej knihe sa

nevenujú len čisto Scrum metodike, ale aj rôznym postrehom a informáciami z ich dlhoročnej praxe. Práve aj ja sa budem v návrhovej časti snažiť priviesť okrem správneho nasadenia Scrum metodiky aj návrhy z praxe, ktoré by mohli riadenie projektu vo vybranej spoločnosti vylepšiť.

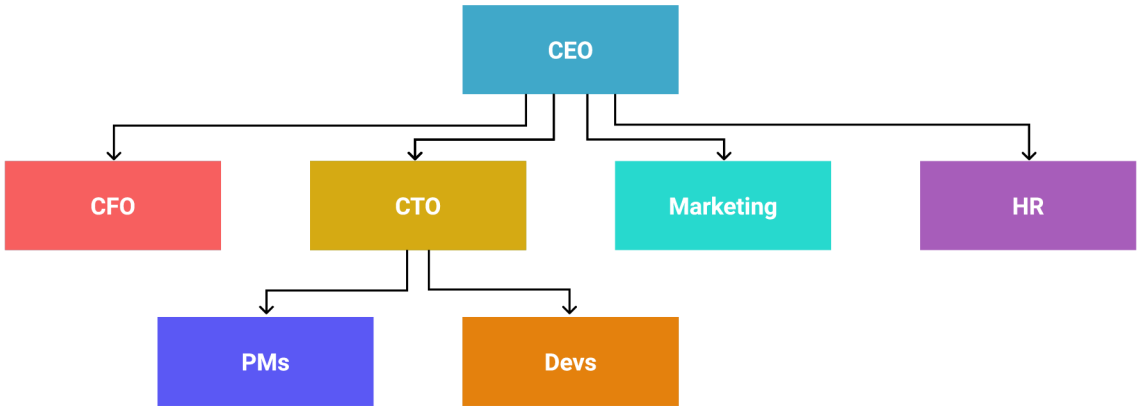
2. Analýza problému a súčasná situácia

Kapitola analýza problému a súčasná situácia opisuje spoločnosť a vybraný projekt vývoja softvéru v tejto spoločnosti. Má za cieľ zmapovať celý proces vývoja na tomto projekte, role participujúce v tíme, udalosti a aktivity, artefakty a aj informačné systémy, ktoré tím využíva.

2.1 Predstavenie spoločnosti a projektu

Vybraná spoločnosť pôsobí na trhu zákazkového vývoja informačných systémov od roku 2008. Zameriava sa na veľké projekty pre zákazníkov z celého sveta. Firma disponuje pobočkami v Brne i v Prahe a do budúcna plánuje expandovať do zahraničia. V súčasnosti vo firme pracuje približne 100 ľudí. Za posledných 10 rokov spoločnosť vydala viac ako 100 softvérových riešení pre zákazníkov a pýši sa viac ako 20 miliónmi unikátnych stiahnutí.

Spoločnosť sa zameriava predovšetkým na vývoj mobilných aplikácií pre Android a iOS, tvorbu webových aplikácií a webových stránok, či riešeniami operujúcimi s veľkým objemom dát. Tímy využívajú koncept agilného vývoja softvéru s podobnosťami metodike SCRUM. Veľkosť tímov sa rozlišuje podľa alokácie a potreby daného projektu, ale zväčša sa jedná o 5-7 zamestnancov na projekt.



Obrázok č. 6: organizačná štruktúra vybranej spoločnosti
(Zdroj: vlastné spracovanie)

Každý projekt je trochu odlišný, či už štýlom riadenia, veľkosťou tímu, alebo prístupom k zákazníkovi. Táto diplomová práca sa bude zameriavať na jeden konkrétny projekt, na ktorom firma v súčasnosti pracuje. Jedná sa o produkt zahraničného klienta, ktorý pôsobí na trhu s automobilmi.

Spoločnosť poskytuje pre klienta komplexné softvérové riešenie pre mobilnú a webovú platformu, pričom sú zastrešované spoločnou dátovou vrstvou. V súčasnosti pracuje tím na softvéri pre interné použitie v klientskej spoločnosti, a neskôr plánuje začať pracovať aj na časti riešenia, ktorá bude určená k používaniu pre koncových zákazníkov. Celý produkt sa vyvíja v programovacom jazyku Kotlin. Na projekt je alokovaných 8 zamestnancov. V 2. kvartále roku 2023 plánuje spoločnosť alokáciu navýšiť o ďalších minimálne 6 vývojárov, a to z dôvodu pokrytia vývojového tímu pre nové požiadavky na produkt. Jedná sa o veľmi dôležitý projekt pre túto spoločnosť, a preto je firma otvorená zmenám v projektovom riadení. Tak aby uspokojila klienta i maximalizovala úspešnosť riadenia projektov do budúcna.

2.2 Roly

Nasledujúca podkapitola obsahuje opis rolí na vybranom projekte.

Projektový manažér

Prvým z členov tímu na projekte je projektový manažér, ktorého úlohou je predovšetkým riadenie projektu a backlogu, avšak tu jeho úloha nekončí a mimo práce projektového manažéra zastáva aj iné funkcie. Dôležitou úlohou, ktorú vykonáva, je samozrejme komunikácia so zákazníkom, kde spolu rozoberajú očakávania, víziu a špecifikácie projektu. Manažér následne analyzuje položky, niekedy ide o hĺbkovú analýzu a niekedy len o predbežnú analýzu.

Projektový manažér má pod správou celý backlog projektu, vytvára nové položky, prioritizuje ich, častokrát ohodnocuje a rozdeľuje prácu pre developerov v aktuálnom Sprinte.

Čo sa týka tímových udalostí, projektový manažér má na starosti ich plánovanie, organizovanie aj moderovanie. Jedná sa napríklad o Daily, Retrospektívu, alebo plánovanie Sprintu.

Do kompetencií projektového manažéra na tomto pozorovanom projekte spadá aj občasné testovanie funkcionality produktu, ktoré pramení z jeho bývalej práce na pozícii testera na inom firemnom projekte. Potreba testovania zo strany projektového manažéra vychádza z nedostatku času testera, ktorý pracuje na trištvrtečný úväzok. Úlohy na testovanie sa kopia predovšetkým na konci Sprintu. Veľkou časťou jeho práce je zabezpečovanie odovzdania novej verzie softvéru a získavanie spätnej väzby od zákazníka. Má konečné slovo pri schvalovaní produktu.

Projektový manažér je vo veľkej miere otvorený zmenám v riadení vývoja, avšak žiadne predchádzajúce skúsenosti s využívaním metodiky Scrum nemá. Veľká časť podobností, ktorú aktuálny štýl riadenia má so Scrumom sú práve na jeho popud a iniciatívu.

Vývojári

Vývojový tím pozostáva z 5 developerov, z ktorých 1 zastrešuje backend (dátová vrstva, API), 2 frontend (webová stránka a webové aplikácie) a 2 developeri sa špecializujú na mobilný vývoj (Android). Developeri sú skúsení a ich tím sa do budúcna plánuje rozširovať. Vývojový tím sídli v oboch pobočkách firmy a niektorí členovia pracujú na diaľku z domu. Z tohto dôvodu je pre tím veľmi dôležitá komunikácia pomocou informačných systémov.

Vývojári pracujú na úlohách zo Sprint Backlogu a sú zodpovední za správu priradených úloh, hĺbkovú analýzu špecifikovaných požiadavkou, návrh riešenia, implementáciu navrhnutého riešenia, spísanie dokumentácie zdrojového kódu a manuálne testovanie vlastných riešení. Pri backend a frontend vývoji sa vytvárajú aj automatizované testy. Vývojári, spolu s projektovým manažérom, sa občas podieľajú na stretnutiach s klientom, pomáhajú odhadovať, naceňovať projekt, analyzovať a samozrejme vymýšľať funkcionalitu produktu.

Kvôli dôležitosti klienta firma alokovala na tento projekt tím so seniornými znalosťami a práve vďaka zvýšenej seniorite sa produkt neustále vylepšuje aj na popud vývojárov.

Dizajnér

Aby výsledný produkt bol vizuálne prívetivý, je potrebná práca UI dizajnéra (User Interface). Avšak aby užívateľské rozhranie bolo zmysluplné a bol umocnený užívateľský zážitok, je potrebná práca UX dizajnéra (User Experience).

Na tomto projekte pracuje dizajnér, ktorý má na starosti UI aj UX. Dizajnér úzko spolupracuje s vývojovým tímom kvôli ujasneniu požiadaviek a overeniu technologických možností. Vizuálnu časť svojej práce spracováva pomocou grafického nástroju Figma. Tento nástroj používa aj vývojový tím, pričom navrhnuté užívateľské rozhranie im slúži ako náhľad pre implementáciu vizuálnej stránky aplikácií. Design však nie je jeho jediná úloha, rovnako sa podieľa na analyzovaní požiadaviek, niekedy dokonca programuje v jazyku Kotlin, rovnako ako vývojári.

Tester

Koncovou pozíciou pred posunutím nových funkcionalít do produkcie je pozícia testera. Tester má na starosti rozdielne platformy a musí sa vysporiadať aj s variabilitou prostredí jednotlivých platforiem. Pre mobilné a webové aplikácie je tak potrebné zvlášť otestovať napríklad jednotlivé krajiny, pre ktoré sú aplikácie určené.

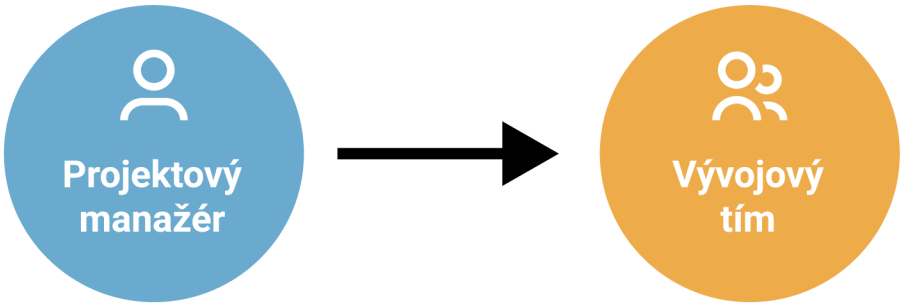
Tester sa pri testovaní riadi špecifikáciou uvedenou v samostatných taskoch. Ak tester objaví nezrovnalosť v nových alebo upravovaných funkcionalitách, zaznamená problém do konkrétneho tasku a úlohu opäť priradí na zodpovednú osobu. Čas od času tester narazí aj na neočakávaný stav mimo testovanej úlohy. V tomto prípade tester zakladá takzvaný bug a jeho priradenie konzultuje s projektovým manažérom. Automatizované testy zvyčajne nemá na starosti tester, o jej správu sa stará vývojový tím.

Okrem testovania novej a upravenej funkcionality po častiach sa tester stará aj o celkovú kontrolu aplikácií pred ich zverejnením. Rolu testera, ako už bolo spomenuté, čiastočne zastupuje aj projektový manažér. V súčasnosti tester pracuje len na ¾ úväzok z dôvodu školských povinností, ale to by sa malo v najbližších mesiacoch zmeniť. Častokrát nastáva problém, že tester nemá na začiatku Sprintu dlhšiu dobu čo otestovať. Potom sa ku koncu Sprintu hromadia veci v stave „pripravené na testovanie“.

Zákazník

Rola zákazníka je pri tomto konkrétnom projekte nesmierne dôležitá. Zákazník v minulosti využíval už podobné informačné systémy pre jeho prácu a z toho dôvodu úzko spolupracuje s projektovým manažérom a podieľa sa na detailnom špecifikovaní zadania.

Zákazník sa podieľa aj na testovaní aplikácií. Pre tieto účely vytvoril vývojový tím špeciálne prostredie, ktoré nie je súčasťou ani vývojového prostredia a ani produkčného prostredia. Zákazník toto prostredie používa na zaškolenie nových pracovníkov pre predstavenie nových funkcionalít ešte pred zverejnením. Jedinou udalosťou, ktorej sa zákazník zúčastňuje je projektový all-hands míting a samozrejme stretnutia s projektovým manažérom, kde komunikujú víziu, ale aj prioritizáciu položiek. Nepochopenie občas nastáva, keď zákazník nešpecifikuje čo by potreboval vyriešiť za problém, ale rovno prichádza s riešením. Takýto postup prináša často zmätok na oboch stranách.



Obrázok č. 7: Aktuálne členenie tímu
(Zdroj: vlastné spracovanie)

2.4 Udalosti

V nasledujúcej podkapitole sú opísané udalosti a aktivity, ktoré v tíme na tomto projekte prebiehajú. Udalosti sú založené na metodike Scrum, ale veľká časť z nich sú len inšpirované a so Scrumom majú spoločný predovšetkým názov. Z analýzy pomocou pozorovania vyplynulo, že tím sa nepravidelne zúčastňuje udalostí: Daily, Sprint Review, Retrospektíva, plánovanie a projektový all hands meeting.

Daily Scrum

Trikrát za týždeň (pondelok, streda a piatok), vždy v rovnaký čas, sa tím stretáva na standup mítingu. Schôdzka sa uskutočňuje online a na tieto účely využívajú v tíme Google meets. Mítingu sa zúčastňuje celý tím. Daily je naplánované od 8:30 do 9:00, ale väčšinou sa predĺži o 20 až 30 minút. Míting vedie a moderuje projektový manažér a to tým spôsobom, že menovite vyzýva ľudí z tímu a tak im dáva priestor na prehovorenie. Každý zo zúčastnených na tomto mítingu informuje ostatných kolegov o tom, na čom

pracoval predošlý deň, na čom plánuje pracovať aktuálny deň, s akými prekážkami sa stretol a v akom stave sa nachádza jeho task.

Cieľom je, aby si určili závislosti a prekážky a spoločne ich odstránili. Projektový manažér prichádza na standup míting s novinkami ohľadom produktu, prípadne s internými firemnými informáciami. Debata sa častokrát skĺzne mimo účelu toho mítingu a to zdržiava ostatných zúčastnených. Niektoré dni sa projektovému manažérovi prekrýva tento míting s iným neplánovaným mítingom. V takom prípade je daily standup zrušený.

Plánovanie

Plánovanie Sprintu sa odohráva zakaždým posledný deň Sprintu a plánuje sa vždy nasledujúci Sprint. Túto udalosť vedie projektový manažér, ktorý vyberá podľa prioritizácie položky z Product Backlogu a presúva ich do Sprint Backlogu, ktorý je práve vo fáze plánovania. Ešte než sa začne s priraďovaním jednotlivých user stories, projektový manažér si overí dostupnosť členov tímu na ďalší Sprint. Jednotlivé user stories sa priraďujú konkrétnym ľuďom, ktorí za ne majú zodpovednosť a na nasledujúcom Sprinte na nich budú pracovať.

Keď má každý člen tímu rozdelenú prácu na ďalší Sprint, nastáva ohodnocovanie user stories story pointmi (väčšina položiek už je ohodnotená projektovým manažérom v predstihu). Pri tejto aktivite sa názory na počet story pointov veľmi líšia a preto musí často rozhodovať projektový manažér. Pri plánovaní sa projektový manažér snaží vždy o podobný počet story pointov na každý Sprint, samozrejme v závislosti od dostupnosti vývojárov, ich dovoleniek, prípadne štátnych sviatkov, kedy sa produkt nevyvíja.

Sprint Review

Táto udalosť býva spojená s Retrospektívou a teda koná sa rovnako ako Retrospektíva v nepravidelných intervaloch raz za dva až tri Sprinty po plánovaní Sprintu. Sprint Review býva naplánový na 45 minút a odohráva sa v posledný deň Sprintu. Jeho dĺžka však väčšinou presiahne hodinu čistého času a to zapríčiní skrátenie vyhradenej doby na Retrospektívu.

Sprint Review moderuje projektový manažér a zúčastňujú sa ho všetci členovia tímu. Každá priradená zodpovedná osoba sa k danej user story v krátkosti vyjadrí a prípadne to

otvorí hlbšiu diskusiu. Backlog sa prechádza vždy postupne najprv backend, frontend a nakoniec vývojári. Cieľom je prejsť si jednotlivé položky zo Sprint Backlogu, overiť ich stav a pozatvárať user stories, prípadne nedokončené tasky presunúť do iného Sprintu.

Retrospektíva

Retrospektíva sa pri tomto projekte uskutočňuje nepravidelne, každý druhý-tretí Sprint. To, či je Retrospektíva potrebná, alebo nie, vyhodnocuje projektový manažér vždy na konci Sprintu. Retrospektívy sa zúčastňuje celý tím a vedie ju projektový manažér. Má za cieľ zistiť pocity členov tímu a získať ich názor na rôzne témy. Z dôvodu, že sa tím nestretáva fyzicky ako celok, sú Retrospektívy riešené online formou a to za pomoci softvéru Miro a komunikácia prebieha cez Google meets.

Projektový manažér využíva vždy jednu šablónu z tohto softvéru, ktorá je rozdelená na 4 oblasti. Prvá oblasť je „veci čo boli dobré“, kde sa majú všetci zúčastnení pomocou lístočkov vyjadriť k skutočnostiam, ktoré sa im na poslednom Sprinte páčili, s čím boli spokojní, prípadne nejaké pozitívne postrehy. Druhou oblasťou je „veci čo boli zlé“, kde sa zbierajú názory na to čo sa im nepáčilo počas Sprintu a s čím nesúhlasili. Treťou časťou zvolenej formy Retrospektívy sú „nápady“. Táto časť slúži na vyjadrenie postrehov, čo by sa malo zmeniť, čo by mal tím začať robiť inak ako doposiaľ a rôzne nápady ako posunúť tím vpred. Poslednou časťou sú „akčné body“, ktoré má na starosti projektový manažér ako facilitátor tohto mítingu. Akčné body slúžia na zaznamenanie ďalších krokov, ktoré vyplynuli z Retrospektívy a ktorým sa tím ako celok, alebo jednotlivci bude venovať.

Udalosť/Deň sprintu	1	2	3	4	5	6	7	8	9	10
Daily Scrum	X		X		X	X		X		X
Plánovanie										X
Review										X
Retrospektíva										X

Obrázok č. 8: Aktuálny kalendár udalostí
(Zdroj: miro.com)

Tabuľka vyššie obsahuje kalendár Sprintu s udalosťami podľa toho v ktorom dni sa uskutočňujú. Ako už bolo spomenuté, Review a Retrospektíva sa uskutočňuje nepravidelne, to znamená nie v každom Sprinte.

Projektový all-hands meeting

Raz za kvartál sa uskutočňuje tzv. projektový all-hands meeting, ktorého sa zúčastňuje zákazník, majiteľ vývojárskej spoločnosti a všetci členovia tímu z tohto projektu. Zákazník spolu s projektovým manažérom predstaví všetkým zainteresovaným stranám, ako sa produktu darí, a predstaví plány na ďalší kvartál. Posledné kvartál sa do agendy pridala aj prezentácia produktu zákazníkovi a tým pádom aj zozbieranie jeho spätnej väzby.

Zamestnanci softvérovej firmy berú tento míting ako veľmi zaujímavý a užitočný. Spoločnosť tak chce poďakovať tímu pracujúcemu na projekte a motivovať všetkým tým, že ukáže reálne čísla a úspechy produktu. Na konci je priestor na otázky a komunikáciu.

2.3 Artefakty a praktiky

Analýzou boli zistené a následne rozpísané v nasledujúcej podkapitole artefakty a praktiky, s ktorými členovia tímu na projekte prídu do styku.

Sprint

Pri tomto konkrétnom projekte trvá Sprint vždy 2 týždne. Výnimku tvoria len sviatky v období Vianoc, kedy sa Sprint môže natiahnuť z dôvodu dovoleniek a prispôsobí sa koncu kalendárneho roka až na trojtýždňový Sprint. Sprint začína v stredu a končí v utorok o 2 týždne. Číslo Sprintu sa počíta vždy od prvého v novom kalendárnom roku a za číslom sa označuje aktuálny rok (napr. 3/2022).

Prvý deň Sprintu odštartuje ranný daily standup, ktorý sa väčšinou natiahne na dlhší čas, než je vyhradený. Posledný deň Sprintu sa uskutočňujú plánovanie a následne či už pravidelné alebo nepravidelné ceremónie ako Sprint Review a Retrospektíva. Po poslednom Sprinte v kvartáli sa uskutočňuje projektový all-hands meeting.

Product Backlog

Produktový Backlog je vlastne zoznam úloh, ktoré je potrebné na produkte zrealizovať. Pre zaznamenávanie, správu a kontrolu týchto úloh používa tím na tomto projekte, ale aj

celá spoločnosť, softvér Jira od spoločnosti Atlassian. Do produktového Backlogu majú prístup nie len všetci členovia tímu na tomto konkrétnom projekte, ale aj všetci zamestnanci z celej vývojárskej spoločnosti. Primárne do product Backlogu pridáva položky projektový manažér, ktorý aj nesie zodpovednosť za správu celého Backlogu.

Sprint Backlog

Rovnako ako pre správu produktového Backlogu je využívaný softvér Jira aj pre Srintový Backlog. Sprint Backlog je zoznam úloh na ktorom tím pracuje v danom Sprinte. Väčšinou obsahuje aj úlohy, ktoré boli presunuté z predchádzajúceho Sprintu, bugy od zákazníka a samozrejme nové úlohy, ktoré boli naplánované počas Sprint planningu.

Položky v Sprintovom Backlogu sú zoradené podľa priority, kde na najvyššej priečke je úloha s najvyššou priotitou a na konci sú položky s najnižšou prioritou. Jednotlivé položky v Sprintovom Backlogu by mali obsahovať popis, dokumentáciu, označenie stavu v akom sa nachádza a priradenú osobu, ktorá je zodpovedná za úlohu. Nie každá položka spĺňa všetky vyššie spomínané parametre a to často spomaľuje prácu na vývoji produktu. Sprint Backlog tímu zostavuje a predstavuje projektový manažér. Sprint Backlog býva narušený a to pridaním väčšieho objemu story pointov počas priebehu Sprintu. Tieto položky neboli zohľadňované pri plánovaní. So Sprint Backlogom sa po pridaní nenaplánovaných položiek nič viac nerobí. Pôvodne naplánované položky v Sprintovom Backlogu zostávajú.

Tím pre lepšiu vizualizáciu používa Kanbanovu alebo teda Scrum tabuľu. Viackrát už sa stalo, že bol Sprint Backlog nedopatrením narušený niekým z iného vývojového tímu a položky sa zo stĺpčekov stratili.

Epic

Pre označenie globálneho kontextu v rámci požiadaviek na implementáciu sa využíva Epic. Tím využíva Epic pre pomenovanie najväčšej časti funkcionality a samotný nemusí obsahovať žiaden popis. Epici sú priraďované samostatným user sories, taskom, bugom a tím ich využíva pre lepšiu orientáciu pri vývoji. V softvéri Jira má Epic nasledovné stavy: future, later, now, archived, done.

User story

User story je používaná pre označenie širšieho celku, pričom existuje minimálne jedna pod úloha typu task, ktorá z nej vychádza. Rovnako ako task, user story taktiež obsahuje popis návrhu zadania a popis návrhu riešenia. Častokrát sa stáva, že opisy niesú súčasťou user story a to zapríčiňuje rôzne komplikácie a nejasnosti. Hlavným rozdielom pri použití user story je kvôli možnosti spoločného popisu požiadavku pre viaceré pod úlohy, ktoré sú určené jednotlivým platformám. Každá user story je pridelená k jednému Epicu, nikdy nie opačne. User story sa vyznačuje nasledujúcimi stavmi: draft, úlohy, blocked, rozpracované, merge request, dev testing, archived a done.

Task

Základným typom požiadavky je task a slúži pre spracovanie úlohy. Task obsahuje popis návrhu zadania a popis návrhu riešenia. Task môže byť vytvorený všetkými členmi tímu, pričom musí byť stanovená zodpovedná osoba pre jeho vypracovanie. Tasky sú usporiadané podľa priority, kde task s najvyššou prioritou je na začiatku Backlogu.

Task môže byť vytvorený ako súčasť User story, alebo aj samostatne. Tím je zvyknutý využívať stavy: úloha, blocked, rozpracované, in review, ready to deploy, archived a done

Bug

Bug využíva tím pre označenie požiadavky reprezentujúcu chybu. Niekedy bug obsahuje špecifikáciu ako:

- Popis chyby
- Postup pre vyvolanie chyby
- Uvedenie na akej platforme chyba nastala
- Názov verzie v ktorej chyba nastala
- Možné riešenie pre opravu chyby

Častokrát sa stane, že bug neobsahuje všetky náležitosti a riešiteľ chyby si musí dohľadávať potrebné informácie od tvorcu bugu.

Takýto Bug môže vytvoriť ktorýkoľvek člen tímu na popud vlastného nálezu, alebo nálezu od používateľa. Bug býva väčšinou daný do aktuálneho Sprintového Backlogu a nie je vyhodnotená jeho prioritizácia.

V informačnom systéme Jira rovnako ako aj pri Tasku je Bug označený aktuálnym stavom. Týmito stavmi sú: úloha, blocked, rozpracované, in review, ready to deploy, archived a done.

2.5. Proces vývoja softvéru

V podkapitole proces vývoja softvéru je opísaný celkový proces vývoja softvéru na tomto projekte.

Požiadavka na funkcionalitu

Základným vstupom do procesu vývoja je požiadavka na funkcionalitu. Požiadavky prichádzajú prevažne od zákazníka, ale existujú aj také, ktoré prichádzajú od tímu pre testovanie a od vývojového tímu. Požiadavky na funkcionalitu rozdeľujú na nové požiadavky, požiadavky pre úpravu funkcionality, požiadavky pre opravu funkcionality a požiadavky pre vylepšenie funkcionality.

Každý nový projekt začína s radou nových požiadaviek na funkcionalitu. Nové požiadavky prichádzajú vo forme textového dokumentu (Google Doc), kde zákazník definuje, ako by sa mala nová funkcionalita správať, a čo by mala spĺňať. Ideálnym prípadom popisu novej požiadavky je popis problému. Problém musí konkrétne popisovať o čo sa jedná a predostrieť očakávaný stav, nie riešenie. Príkladom problému pre projekt zaoberajúci sa výkupom vozidiel je problém nemožnosti nafotiť vykupované vozidlo s automatickým nahrávaním do databázy. Nová požiadavka tak neopisuje konkrétnu funkcionalitu, ale iba problém, ktorý by mala nová funkcionalita riešiť. Avšak nové požiadavky prichádzajú od zákazníka nie formou popisu problému, ale formou popisu konkrétnej funkcionality, o ktorú má záujem. Nastáva tak stav, kedy zákazník získa funkcionalitu, ktorú chce a nie funkcionalitu, ktorú potrebuje.

Požiadavka na úpravu funkcionality je typ, kedy súčasný stav nie je vyhovujúci, a je potrebné existujúcu funkcionalitu upraviť. Zaujímavé je, že aj menšia úprava funkcionality môže spotrebovať viac času na vývoj než nová požiadavka. Je to z dôvodu náročnejšej analýzy aktuálneho stavu. Rovnako ako predošlý typ požiadavky, aj tento typ býva zväčša popísaný v textovom dokumente. V inom prípade dochádza k popisu požiadavky na úpravu priamo formou nového tasku prostredníctvom systému JIRA, kedy nastáva vynechanie analýzy požiadavky a priame popísanie návrhu riešenia. Pri menších

úpravách môže tento úkon ušetriť čas potrebný na realizáciu požiadavky, ale môže aj spôsobiť nechcenú úpravu, ktorá vyžaduje ďalšiu úpravu.

Typ požiadavky pre opravu nemá jasne definovanú podobu, akou by mala byť spísaná. K nálezu dochádza primárne počas testovania testerom pred zavedením do produkcie, no stáva sa, že požiadavka pre opravu vznikne aj na popud problému vzniknutom po nasadení. Pri náleze problému testerom je oprava popísaná formou bugu zadanom v systéme JIRA. Tester v ňom popíše o aký nevyhovujúci stav sa jedná, a špecifikuje stav, pri ktorom chyba nastala. Následnú analýzu a návrh riešenia preberá vývojový tím, ktorý s projektovým manažérom v ojedinelých prípadoch konzultuje, či má byť oprava vykonaná v aktuálnom Sprinte, alebo presunutá do Backlogu. Pokiaľ dôjde k nálezu chyby na strane zákazníka, popis opravy býva odovzdaný vývojovému tímu. Vývojári overia, či chyba nastáva a pri kladnej odpovedi majú na starosti prevziať postup testera a pre danú opravu vytvoriť bug.

Požiadavka pre opravu môže prísť aj z informačného systému Firebase, ktorý zbiera reporty priamo z produkcie a vývojový tím notifikuje o vzniknutých chybách, ktoré by mohlo byť potrebné opraviť. V tomto prípade sa postupuje rovnako, ako v predošlom.

Posledným typom požiadavky na zmenu je návrh na vylepšenie od vývojového tímu. Vývojári získavajú počas vývoja detailné informácie o zadanom probléme, a niekedy tak prídu na riešenie, ktoré by mohlo byť efektívnejšie než to, ktoré majú vyvinúť ale už vyvinuli. Zadaný návrh na vylepšenie konzultuje projektový manažér so zákazníkom, ktorý rozhoduje o zaradení schválení požiadavky.

Analýza požiadavky

Cieľom analýzy požiadavky je overiť relevantnosť z hľadiska pridanej hodnoty pre zákazníka. Nedielnou súčasťou analýzy je tiež zváženie času a náročnosti konkrétnej požiadavky na základe stanovených cieľov. Výsledkom analýzy požiadavky je schválenie alebo zamietnutie. Schválená požiadavka je zaradená do plánu pre vývoj aj na základe času a náročnosti požiadavky. Zodpovedná osoba pre analýzu sa mení v závislosti na type požiadavky.

Pri analýze novej požiadavky je potrebná dôkladná analýza na strane projektového manažéra z dôvodu overenia jej potreby. Ak sa jedná o zložitú alebo technickú

požiadavku, projektový manažér ju konzultuje s vývojovým tímom. Po schválení projektový manažér určuje zodpovednú osobu pre návrh riešenia. Častokrát však novú požiadavku navrhuje sám projektový manažér s čiastočnou pomocou od dizajnéra.

Pre analýzu požiadavky na úpravu nie je stanovený presný proces. Pri požiadavke na úpravu od zákazníka sa postupuje rovnakým spôsobom, ako pri požiadavke na novú funkcionalitu. Pri zadaní požiadavky na úpravu priamo do systému JIRA formou nového tasku môže dôjsť k preskočeniu analytickej časti. Zodpovedná osoba predpokladá pri návrhu riešenia uskutočnenú analýzu danej požiadavky. Z dôvodu možnosti vytvorenia nového tasku kýmkoľvek z tímu môže nastať vynechanie analýzy. Tento prípad má za následok navýšenie iterácií potrebných pre dokončenie požiadavky a tím sa k nemu musí viac krát vracať.

Analýza pre požiadavky na opravu sa rozdeľujú na 2 typy. Prvým je požiadavka na opravu zo strany testera. Tester z návrhu riešenia dospeje k záveru, že požiadavka nevyhovuje zadaniu a je potrebné oprava. V tomto prípade analýza nie je potrebná, pretože prebehla pred vývojom a jej výsledok nijako nevyvracia požiadavku na opravu. Druhým typom je požiadavka na opravu zo strany zákazníka. Tento krát je opätovná analýza potrebná, pretože pokiaľ tester potvrdil správnosť požiadavky počas testovania a zároveň zákazník vyžaduje opravu, mohlo dôjsť k nesprávnemu návrhu riešenia plynúcemu z analýzy a popisu. Projektový manažér má za úlohu overiť potrebu vykonania opravy a potrebu vykonania opätovnej analýzy.

Analýza požiadavky na vylepšenie prebieha na strane iniciátora vylepšenia s konzultáciou tímu a predovšetkým s projektovým manažérom. Schválenie požiadavky po analýze závisí od obtiažnosti zavedeniu vylepšenia.

Návrh riešenia

Návrh riešenia priamo vychádza z analýzy požiadavky. Pri návrhu je dôležité zostať v súlade s architektúrou systému, a preto musí mať zodpovedná osoba čo najviac informácií o danom probléme. V začiatočných fázach vývoja sa pre zápis návrhu využíval informačný systém Confluence, avšak už po krátkom vývoji sa zápis presunul do systému Jira. Návrh riešenia obsahuje zadanie pre návrh a špecifikáciu samotného návrhu. Návrh riešenia slúži z veľkej časti aj pre testovanie, kedy tím pre testovanie berie návrh ako presný popis, ako sa má riešenie správať a ako má fungovať.

Zodpovednú osobu pre návrh určuje projektový manažér pri vytváraní úloh v Jira, kde má za úlohu definovať minimálne názov úlohy a zadanie problému, ktorý má návrh riešiť. Ak je zadanie návrhu potrebné špecifikovať detailnejšie, je povinnosťou osoby vytvárajúcej zadanie priložiť všetky potrebné dokumenty a informácie pre vytvorenie návrhu. Pokiaľ sa jedná o komplexné riešenia v rámci datovej vrstvy, vytvorenie úlohy má na starosti seniorny backend vývojár na popud od projektového manažéra. Zastúpenie projektového manažéra je z dôvodu náročnosti po technickej stránke.

Typ úlohy v Jira určuje zodpovedná osoba pri vytváraní zadania návrhu. V prípade zložitých a časovo náročnejších riešení sa používa typ úlohy Story, ktorá sa ďalej delí na menšie úlohy, ktoré môžu byť implementované separátne. Tento typ úlohy sa využíva prevažne pre návrh riešenia nových požiadaviek na funkcionalitu. Pre menšie úlohy alebo požiadavky na úpravu býva použitý typ úlohy Task. Jediným rozdielom v návrhu riešenia medzi typom úlohy Task a Story je v tom, že návrh riešenia býva špecifikovaný len v Story a v príslušných pod-úlohách typu Task je špecifikácia prázdna. Samostatné úlohy typu Task špecifikáciu obsahujú. Typ úlohy Bug sa pre návrh riešenia používa v prípadoch požiadaviek na opravu funkcionality. A okrem základného zadania pre návrh, a návrhu riešenia obsahuje aj kroky, akými je možné chybu vyvolať a zariadenie/prostredie, kde sa chyba vyskytla.

Forma návrhu riešenia sa rozlišuje podľa vývojového prostredia. Pre prezentačnú vrstvu (frontend) musí návrh obsahovať odkaz na design a rozdelenie požadovaných funkcionalít pre jednotlivé obrazovky. Pre zadanie návrhu riešenia sa využívajú prípady použitia tzv. use-cases, v ktorých býva popísané chovanie funkcionality zo strany používateľa. Chovanie jednotlivých platforiem v závislosti na požiadavke na funkcionalitu nie je v návrhu zohľadnené, pokiaľ sa jedná o štandardné chovanie platformy. Typickým príkladom je spracovanie a zobrazenie chybovej hlášky. V takomto prípade je za dodržanie štandardov v rámci platformy zodpovedný vývojár a návrh ich obsahovať nemusí. Na druhú stranu, návrh musí obsahovať špecifické chovanie platformy. Príkladom môže byť synchronizácia dát na pozadí systému bez ovplyvnenia používateľa. Pre vrstvu operujúcu s dátami (backend) musí návrh obsahovať jasnú formu dát, ktoré sú na vstupe a výstupe, validáciu dát a popis úspešného a chybového stavu. Pre veľký počet možných technológií na využitie musí návrh obsahovať zoznam technológií, ktoré sa majú využiť práve pri implementácií daného riešenia.

V ojedinelých prípadoch neobsahuje vytvorená úloha žiadne zadanie pre návrh riešenia a je definovaná iba názvom a zodpovednou osobou. V takomto prípade musí pridelená osoba dohľadať všetky potrebné informácie pre vypracovanie návrhu riešenia, zapísať ich ako zadanie pre návrh riešenia a následne návrh vypracovať.

Priorita a časová náročnosť implementácie nebýva určená počas návrhu riešenia. Tieto kritériá sa špecifikujú počas tímových stretnutí určených k plánovaniu nasledujúceho Sprintu.

Pre schválenie návrhu nie je definovaný žiaden postup. Ak má navrhovateľ riešenia všetky potrebné informácie pre vypracovanie návrhu, po jeho vytvorení návrh neprechádza žiadnou validáciou. V prípade nedostačujúcich zadaných informácií prebieha konzultácia s osobou zakladajúcou zadanie návrhu počas jeho spracovania a po jeho vypracovaní.

Vývoj

Ďalším krokom v procese vývoja softvéru je vývoj. Vývoj bez výnimky vychádza z návrhu riešenia. Osobu zodpovednú za vývoj určuje projektový manažér. Častokrát má na starosti vytvorenie návrhu aj implementáciu tá istá osoba.

Progres vývoja je znázornený v systéme Jira pomocou stavov jednotlivých úloh. Vďaka nemu má projektový manažér jasnú predstavu o tom, v akom stave sa práve vývoj nachádza. Úvodný stav úlohy nesie označenie „Pripravené na vývoj“ a vývojár vidí úlohy s týmto označením pre aktuálny Sprint. Tento stav reprezentuje úlohu s pripraveným návrhom riešenia. Ak implementácia nemôže začať z dôvodu nadväznosti na ešte nedokončenú úlohu, nesie pomenovanie „Blokované“. V opačnom prípade vývojár presúva úlohu do stavu „Rozpracované“ a začína s návrhom implementácie. Po dokončení implementácie presúva vývojár úlohu do stavu „Kontrola“ v rámci ktorej kontroluje zdrojový kód iný vývojár z tímu danej platformy. Po dokončení kontroly sa úloha môže vrátiť naspäť do stavu „Pripravené na vývoj“ s pridaným komentárom popisujúcim, čo je potrebné pridať, odobrať alebo opraviť. Po úspešnej kontrole nasleduje stav „Pripravené na testovanie“. V tomto stave sa úloha dostáva do bodu, kedy vývojár vydá novú verziu určenú na testovanie. Vo väčšine prípadov k vydaniu novej verzie dochádza naraz pri väčšom počte úloh v stave „Pripravené na testovanie“. Všetky úlohy obsiahnuté v novej verzii prechádzajú do stavu „Testovanie“. V tomto bode prechádza

pridelená osoba na testovací tím zodpovedná za danú platformu. V prípade negatívneho výsledku pri testovaní vráti testovací tím testovanú úlohu opäť do stavu „Pripravené na vývoj“ s opätovným pridelením vývojára k úlohe. Pri pozitívnom výsledku testovania prechádza úloha do stavu „Hotovo“. Tento stav reprezentuje úlohy pripravené na vydanie do produkcie. Do každého stavu musí byť úloha presunutá ručne a žiadna zmena stavu sa nevykoná automaticky.

Implementáciu riešenia vždy predchádza návrh implementácie. Tento návrh nie je zapísaný v žiadnom dokumente alebo úlohe. Návrh implementácie slúži ako plán pre vývojára, pomocou ktorého postupuje pri písaní zdrojového kódu. Návrh vývojárovi pomáha pri ujasnení kritérií z návrhu riešenia, a pripravuje ho do zásahu do aktuálneho zdrojového kódu. Pre úlohy s menším zásahom vývojári návrh implementácie nepoužívajú.

Pri implementácii vývojár dodržiava architektúru danej platformy a zavedené konvencie formátovania zdrojového kódu. Tímy jednotlivých platforiem majú definované vlastné konvencie formátovania zdrojového kódu nezávisle od projektu. Zaužívané konvencie umožňujú vývojárom absenciu dokumentácie, pretože zdrojový kód je veľmi dobre čitateľný, avšak nástup nového vývojára je pri neexistujúcej dokumentácii náročnejší. Implementácia testov sa využíva predovšetkým pri vrstve operujúcou s dátami a pri prezenčnej vrstve sa využíva len ojedinele. Pre písanie automatických testov vývojári využívajú externé knižnice poskytujúce rozsiahle funkcie pre testovanie zdrojového kódu. Počas implementácie vývojár využíva testovacie prostredie, ktoré nijakým spôsobom neovplyvňuje prostredie určené pre zákazníka a pre produkciu.

Pre správu verzií zdrojového kódu je využívaný nástroj Git, z ktorého vývojár využíva Commit, Branch a Merge Request. Pred implementáciou developer vytvorí Branch, čo je možné si predstaviť ako kópiu aktuálneho stavu pre vykonávanie zmien. Pri implementácii vývojár vytvára Commit, ktorý obsahuje zdrojový kód nových a zmenených súborov a krátku informačnú správu ohľadom uskutočňovanej zmeny. Každý Commit si nástroj Git zvlášť ukladá aj s informáciou o dátume vzniku, autorovi zmeny a pridelenej Branch. Vďaka tomu vie každý vývojár zistiť, kto a kde zdrojový kód zmenil a s akým zámerom. Po dokončení všetkých zmien potrebných pre implementovanie danej požiadavky vytvára vývojár Merge Request. Toto označenie

reprezentuje požiadavku o zavedení zmien zo všetkých Commitov danej Branch do aktuálnej verzie zdrojového kódu, z ktorej sú vytvárané verzie pre testovanie, pre zákazníka a pre produkciu. Pre Merge Request je pridelená osoba, ktorá ho vytvorila a osoba, ktoré má zavádzanú zmenu skontrolovať. Osoba kontrolujúca zdrojový kód vykonáva kontrolu kódu, tzv. Code Review. Kontrolór môže k jednotlivým riadkom zdrojového kódu pridať komentár a rozhoduje o schválení alebo zamietnutí nových zmien. Zároveň pri vytvorení Merge Request prebieha skúška prekladu zdrojového kódu. Táto skúška je druhým ukazovateľom pre schválenie. Po schválení kontrolórom sa kód uloží do aktuálnej verzie.

Testovanie

Posledným krokom pred nasadením novej verzie je testovanie. Pred vykonaním manuálneho testovania sú spustené automatizované testy zdrojového kódu. Ich spustenie prebieha automaticky ako súčasť procesu vydania novej verzie pre testovacie prostredie.

Manuálne testovanie vykonáva tím podľa úloh v stave „Testovanie“ v systéme Jira. Tester má na starosti všetky platformy a úlohy potrebné na otestovanie. Vybavuje ich podľa priority a vzájomným nadväznostiam. Manuálne testovanie vychádza z návrhu riešenia špecifikovanom v danej úlohe. Pokiaľ tester pri testovaní zistí nezrovnalosti v testovanej verzii, má za úlohu oznámiť pridelenému vývojárovi o aké nezrovnalosti sa jedná pomocou komentára a zmeny stavu úlohy na stav „Pripravené na vývoj“. V prípade, kedy tester nájde chybu nesúvisiacu s testovanou úlohou, zakladá novú úlohu typu Bug s popisom, o akú chybu sa jedná, ako chybu vyvolať, v akej verzii bola chyba vyvolaná a prípadne návrh na opravu. Pre úspešné otestovanie tester presúva úlohu do stavu „Hotovo“.

Okrem manuálneho testovania má tester na starosti aj testovanie verzie pred vydaním do produkcie. Toto testovanie obsahuje testovacie scenáre základných funkcií aplikácie, ktorých funkčnosť sa overuje pred každým vydaním do produkcie bez ohľadu na to, či sa do príslušnej funkcionality zasahovalo, alebo nie. Testovacie scenáre sú spísané v tabuľkách Google Sheets vo forme očíslovaných krokov reprezentujúcich jednotlivé úkony.

Nasadenie

Poslednou fázou procesu vývoja softvéru je nasadenie. Pod pojmom nasadenie je myslené zavedenie novej verzie vyvíjaného systému do stavu dostupného pre zákazníka a používateľov. Za nasadenie zodpovedajú vývojári jednotlivých platforiem separátne a nasadenie sa vykonáva na popud od projektového manažéra. Nasadenie býva naplánované na konkrétny deň, pričom zmena termínu je možná len vo výnimočných situáciách.

Keďže mobilné aplikácie je potrebné aktualizovať na každom zariadení zvlášť, pre distribúciu nových verzií je využívaný služba App Distribution nástroja Firebase. Prístup k novej verzii majú len vopred definovaní používatelia stanovení zákazníkom a oznámenie o novej verzii prebieha formou emailu a notifikácie na mobilné zariadenie. V prípadoch, kedy je pri novej verzii mobilnej aplikácie nutná aktualizácia na každom zariadení, musí projektový manažér upraviť minimálnu podporovanú verziu špecifikovanú tiež v nástroji Firebase. Predošlá verzia mobilnej aplikácie tak získa informáciu o nutnosti vykonať aktualizáciu, a túto možnosť užívateľovi automaticky ponúkne.

Pre ostatné platformy nie je potrebná distribúcia novej verzie, pretože nie je potrebné vykonávať aktualizáciu na konkrétnych zariadeniach.

2.6 Informačné systémy

Tím počas celého vývoja softvéru využíva množstvo informačných systémov, ktoré sú podrobnejšie opísané v nasledujúcej podkapitole. Jedná sa o softvér na riadenie projektu, na uchovávanie dokumentov, správu kódu, komunikáciu, vykazovanie práce, ale samozrejme aj na samotný vývoj.

Jira

Softvér Jira od spoločnosti Atlassian využíva tím na riadenie vývoja projektu. Ide o webovú aplikáciu, ktorá tímu pomáha s celkovým procesom vývoja, so správou Backlogu, komunikáciou, reportami, štatistikami a mnoho iného.

Kanbanova tabuľa je jedna z funkcií softvéru Jira, ktorú využíva celý tím na projekte pre lepšiu vizualizáciu aktuálneho Sprintu. Kanbanovu tabuľu majú členovia tímu rozdelenú do šiestich stĺpcov:

- To do – položky ktoré sú v Backlogu
- In progress – položky na ktorých sa pracuje
- In review – položky určené na kontrolu
- Ready to deploy – položky pripravené na testovanie
- Dev testing – položky, ktoré sú v stave testovania
- Done – hotové položky pripravené na vydanie do produkcie

Jednotlivé položky sú presúvané vždy podľa ich aktuálneho stavu do prislúchajúceho stĺpčeka.

Confluence

Pre vytváranie, kolaboráciu a organizovanie dokumentov využíva projektový tím softvér Confluence, taktiež od spoločnosti Atlassian. V tomto online pracovnom priestore sú zaznamenané informácie ako prístupové účty, základné informácie o projekte, kontakty, dokumentácia, alebo napríklad testovacie scenáre slúžiace testerovi. Všetci členovia tímu môžu spoločne tento priestor upravovať.

Gitlab

Gitlab je webový nástroj využívaný na verzovanie a správu zdrojového kódu. Prístup je prideľovaný jednotlivcom pracujúcim na danom projekte a každá platforma má pre svoj zdrojový kód oddelené miesto uloženia. Vďaka tomuto nástroju je na projekte dostupná kompletná história zdrojového kódu a taktiež umožňuje kontrolu zdrojového kódu iným vývojárom pred aktualizovaním súčasnej verzie. Súčasne obsahuje históriu všetkých vydaných verzií, či už pre testovanie, pre zákazníka a aj pre produkciu.

Tempo

Pre zaznamenávanie času práce strávenej na projekte sa využíva nástroj Tempo. Aj keď umožňuje synchronizáciu so systémom Jira a tak vytvára možnosť zaznamenávať čas strávený na konkrétnych úlohách, táto funkcionalita sa nevyužíva.

V druhom rade slúži Tempo na fakturačné účely a projektový manažér pomocou neho plánuje obsadenosť jednotlivých členov vývojového tímu.

Slack

Slack je platforma určená na obchodnú komunikáciu. Slúži pre komunikáciu celého tímu, pričom obsahuje kanály určené pre všetkých členov tímu na zdieľanie všeobecných

informácií a kanály určené pre jednotlivé platformy pre špecifické informácie. Okrem skupinových kanálov umožňuje komunikáciu aj medzi jednotlivými členmi tímu a zároveň uskutočnenie video hovorov.

Google Workspace

Google Workspace je balík nástrojov slúžiacich pre zaznamenávanie textu a ukladanie dôležitých dokumentov.

Pre ukladanie všetkých zdieľaných dokumentov je využívaný nástroj Google Drive so špecifikovanými právami na prístup. Jednotlivé dokumenty je následne možné upravovať pomocou Nástrojov Google Documents a Google Sheets.

Google Drive slúži taktiež pre vývojárov ako úložisko rôznych textových kľúčov potrebných pre vydávanie nových verzií.

Firebase

Informačný systém Firebase sa na tomto projekte využíva pre distribúciu nových verzií mobilnej aplikácie a zároveň pre zaznamenávanie a správu chybových hlásení všetkých platforiem. Pokiaľ dôjde k detekcii chýb na jednotlivých platformách, dochádza k automatickému zaslaniu správy o tejto chybe a možnosti jej zobrazenia v systéme Firebase súčasne s emailovým upozornením pre zodpovedných vývojárov.

Ďalším využitím je uloženie informácie o minimálnej podporovanej verzie mobilnej aplikácie, ku ktorej majú aplikácie prístup a slúži za účelom vykonania automatickej aktualizácie.

Figma

Figma je grafický nástroj umožňujúci prácu s vektorovými objektami. Na tomto projekte sa nástroj Figma využíva pre návrh užívateľského rozhrania mobilnej aplikácie, webovej stránky a webovej aplikácie. Design užívateľského rozhrania v ňom vytvára designér a k náhľadu majú prístup všetci členovia vývojového tímu, ktorí môžu na konkrétnych grafických prvkoch zanechať komentár. Pri vývoji je Figma využívaná pre získanie presných rozmerov a tvarov grafických komponent a hodnôt konkrétnych farieb a štýlov písma.

Vývojové prostredie

Vývojových prostredí je pri tomto projekte využívaných viac, ale najhlavnejšími sú Android Studio využívané pri vývoji mobilnej aplikácie pre platformu Android, PhpStorm využívaný pri vývoji datovej vrstvy a IntelliJ Idea využívaná pri vývoji prezentačnej vrstvy webovej stránky a webovej aplikácie.

2.7 Vyhodnotenie analýzy

Z analýzy vyplynnulo, že projektové riadenie má isté podobnosti so Scrum metodikou, avšak na to, aby bolo možné túto metodiku zaviesť, budú potrebné viaceré zmeny.

Roly

Čo sa projektového manažéra týka, takáto rola v metodike Scrum neexistuje. Dá sa povedať, že v súčasnosti projektový manažér zastáva až tri role spojené do jednej. Zčasti ide o rolu Scrum Mastera (ktorá na prejekte chýba), tým, že moderuje mítingy. Ďalej je súčasťou vývojového tímu, pretože zastupuje úlohu testera. A nakoniec sa dá povedať, že vystupuje aj ako Product owner, pretože má pod správou Backlog. Na to, aby bola Scrum metodika použitá správne, nemôžu byť 3 role zastupované jednou osobou.

Ak teda budeme brať do úvahy, že v tomto prípade projektový manažér je po správnosti Product owner, spĺňa prvú náležitosť vlastníka produktu, a tou je to, že spravuje Backlog a pridáva doň položky. Porduct owner môže a nemusí byť zamestnancom zákazníka, ale musí zastupovať jeho požiadavky, takže v tomto prípade je to tiež v poriadku. Súčasťou práce projektového manažéra je teda komunikácia so zákazníkom a analýza jeho požiadaviek. V Scrume má product owner na starosti predbežnú analýzu, avšak hĺbková analýza by mala byť prácou niekoho iného. V tomto konkrétom prípade projektový manažér robí veľakrát aj celkovú hĺbkovú analýzu, čož je z pohľadu metodiky Scrum nesprávne. Rovnako aj vytváranie testovacích scenárov nie je úlohou Product ownera, ale vývojového tímu. Na vybranom projekte rozdeľuje úlohy v sSprint Backlogu projektový manažér. Z pohľadu Scrumu by si mali Sprint Backlog vytvárať členovia tímu sami a to na základe prioritizácie od Product ownera. V tomto prípade je nesprávne, keď projektový manažér vyberá kto a na čom bude pracovať v nadchádzajúcom Sprinte. Projektový manažér niekedy ohodnocuje položky backlogu story pointmi a má posledné slovo pri tomto ohodnocovaní. Po správnosti by mali ohodnocovať položky vývojári a ak

sa nevedia zhodnúť na počte story pointov, je na mieste využiť nejaký nástroj. A nakoniec konečné slovo pri schvalovaní produktu, ktoré v tomto prípade má projektový manažér (product owner), odpovedá metodike Scrum.

Vývojový tím, na to, aby sme ho mohli označiť za Scrumový, musí spĺňať multifunkčnosť a samoorganizovanosť. Ako už bolo spomenuté v teoretickej časti, v scrumovom tíme niesú ďalej rozdelené role. Vo vybranom projekte tím pozostáva z viacerých špecifickejších rolí ako je tester, či dizajnér. V tomto prípade je to však v poriadku, pretože multifunkčnosť tímu je zabezpečená zdieľanými znalosťami. Aj keď sa testovaniu primárne venuje tester, nie je táto úloha hodená len na neho. Vývojári takisto manuálne testujú. Rovnako sa podieľajú aj na iných úlohách ako je pomoc projektovému manažérovi s analýzou a tak ďalej. Dizajnér ich vie zastúpiť v analyzovaní, alebo aj vývoji menších položiek. Týmto tím splňuje zastupiteľnosť, ktorá je pri Scrumovom tíme podstatná. Jediný člen vývojového tímu, pri ktorom je potrebné zapracovať na multifunkčnosti je tester. Ďalej v rámci multifunkčnosti tímu bude nutné navrhnúť menšie zmeny, hlavne v oblasti plných úväzkov zamestnancov, nahromadenej práce testera na konci Sprintu a spoločnej zohratej práci na každej položke Bbacklogu už od začiatku.

Čo sa samoorganizovanosti týka, tímy by si mali sami vybrať na čom budú pracovať v aktuálnom Sprinte, samozrejme tak, aby to sedelo s prioritizovanými položkami. Taktiež je miesto na zlepšenie komunikácie v tíme a tímovej spolupráci, tak aby tím bol čo najefektívnejší. Príkladom je nekonanie sa mítingov bez projektového manažéra, aj keď jeho prítomnosť nie je vyžadovaná. Toto určite nie je známka samoorganizovaného tímu.

Artefakty a praktiky

Čo sa sSrintu týka, na projekte je dĺžka sSrintu odpovedajúca pravidlám Scrumu. Backlog je využívaný taktiež správne a je splnená aj podmienka, že Product owner (v tomto prípade projektový manažér) zaň nesie zodpovednosť. Tím využíva Scrum board, ale problém nastáva pri miznúcich položkách z tohto boardu. Probém je určite v samotných položkách Backlogu, ktoré nespĺňajú Definition od Ready a Definition od Done. Tím tieto definície nemá ani stanovené a to spôsobuje problémy v inkrementoch. Ako už bolo spomenuté, veľkou odchýlkou od metodiky Scrum, je zostavovanie Sprint Backlogu projektovým manažérom a nie členmi tímu.

Z analýzy ďalej vyplynulo, že prebiehajúci Sprint je častokrát narušený a to pridaním položiek s veľkým počtom Story pointov. Problém je, že sa do Sprintu pridávajú nové veci a nič sa z neho neodstráni pre zachovanie balancu. Ďalšou nepresnosťou od Scrum metodiky je absencia Sprint goalu, ktorý zabezpečuje sústredenosť na dodanie dôležitej časti. Rozpad položiek Backlogu na Epik, User story a Task je v poriadku a odpovedá metodiek Scrum. Taktiež aj Bugy sú správne využívané na označenie chyby, ale častokrát sú hodené do aktuálneho Sprint Backlogu a nie je prioritizované ich vyriešenie. To má za následok vyriešenie malého, menej podstatného bugu na úkor veľkej chyby, ktorá má vyššiu biznisovú hodnotu.

Udalosti

Z analýzy vypadlo niekoľko nepresností aj čo sa udalostí týka. Veľkým problémom je nepravideľnosť udalostí a z toho prameniaca ich dlhá doba trvania. Príkladom je Daily míting, ktorý sa odohráva 3x za týždeň, ale častokrát je zrušený z dôvodu časovej vyťaženosti projektového manažéra. Keď už táto udalosť prebehne, má neprimeranú dĺžku a častokrát sa debata skĺzne úplne mimo agendu stretnutia. Plánovanie Sprintu je podla Scrumu prvá udalosť Sprintu, ale v tomto prípade sa uskutočnuje posledný deň Sprintu pred Review a Retrospektívou. Problémom plánovania je aj fakt, že User stories by už mali byť ohodnotené ale nie projektovým manažérom, ako to je zvykom na tomto projekte. Review je udaosť, ktorej by mal byť súčasťou zákazník alebo stakeholderi, v tomto prípade sa odohráva len v kruhu tímu. Retrospektíva je nepravideľná, a tým, že ju vedie projektový manažér, nastáva konflikt. Preto je nutné, aby Retrospektívu viedla „tretia strana“ a to Scrum Master.

Ďalej z analýzy vyplynula skutočnosť, že tímu chýba aktivita, kde by prechádzali spoločne Backlog, ohodnocovali ho, riešili nezrovnalosti v položkách, a tak sa uistili, že všetci rozumejú Backlogu. Poslednou udalosťou je projektový All-hands meeting, ktorý sa v metodike Scrum nenachádza, avšak tímu vyhovuje. Časť jeho agendy by sa ale mala určite stať súčasťou udalosti Review.

Proces vývoja

Prvým bodom súhrnu analýzy procesu vývoja softvéru je postup popisu nových požiadaviek od zákazníka. Požiadavky častokrát prichádzajú od zákazníka nie formou popisu problému, ale formou popisu konkrétnej funkcionality, o ktorú má záujem.

Dochádza tak k preskočeniu niekoľkých krokov procesu vývoja a môže to spôsobiť stav, kedy zákazník získa funkcionalitu, ktorú chce a nie funkcionalitu, ktorú potrebuje.

S podobným problémom sa stretáva aj typ požiadavky určenej na úpravu existujúcej funkcionality. Keďže tento typ nemá jasne definovanú štruktúru a spôsob, ako s týmto typom postupovať v rámci procesu, nastáva vynechanie analýzy požiadavky a priame popísanie návrhu riešenia.

Pri type požiadavky určenej na opravu sa existujúce procesy od vyššie spomenutých značne odlišujú. Z analýzy vyplýva absencia definície štruktúry tohoto typu a zároveň nie je možné využiť procesy z ostatných typov požiadaviek kvôli rôznym odlišnostiam.

Čo sa týka určenia právomocí v rámci tímu, projektový manažér často zasahuje do kompetencií, ktoré by správne podľa metodiky Scurm nemal vykonávať. Na základe nových požiadaviek od klienta vykonáva projektový manažér analýzu, ktorá nie je len povrchová a ktorej časti by mal vykonávať vývojový tím. Rovnako aj pri analýze požiadaviek pre úpravu existujúcej funkcionality neexistuje žiadna ucelená forma pre jej priebeh a prakticky ktokoľvek v tíme môže takýto typ požiadavky vytvoriť, a tým vzniká riziko vynechania analýzy.

Na rozdiel od požiadaviek od klienta, návrh riešenia nebýva definovaný v samostatnom súbore, ale býva súčasťou jednotlivých úloh v Jira. Pri samotnom vývoji je síce návrh jednoducho dohľadateľný, ale pri úprave alebo oprave je náročnejšie návrh dohľadať.

Prípad, kedy je v systéme Jira vytvorená úloha bez popisu očakávajúca pre dopracovanie návrhu priradenej zodpovednej osoby, by nemal nastať. Pokiaľ je k úlohe priradená zodpovedná osoba, a úloha neobsahuje žiaden popis, ukazuje to na absenciu krokov potrebných vykonať pred zahájením vývoja.

Avšak jedným z najväčších problémov vyplývajúcich z analýzy procesu vývoja softvéru je chýbajúci proces schvaľovania návrhu. Tím nemáva žiadne stretnutie, pri ktorom by spolu prechádzali návrhy jednotlivých úloh pred vývojom, takže môže nastať chyba návrhu spôsobená vykonávateľom návrhu riešenia. Taktiež osobu zodpovednú za vývoj by po správnosti podľa metodiky Scrum nemal určovať projektový manažér. Takéto roztriedenie práce by mohli byť napríklad súčasťou stretnutia určeného pre plánovanie nasledujúceho Sprintu.

Stavy úloh v systéme Jira sú dôležitou súčasťou pri sledovaní priebehu vývoja softvéru. Stav s označením „Blokované“ môže určiť ktokoľvek a zároveň bez určenia príčiny. Dôvod, prečo sa úloha dostane do tohoto stavu, ostáva pre zvyšok tímu neznámy.

Dilema, s ktorou sa stretne každý vývojár, je odpovedať na otázku, či k zdrojovému kódu písať aj dokumentáciu. Tím má na rozličných platformách stanovené konvencie a pravidlá, ktoré síce kód značne sprehľadňujú, ale pre nových členov pri rastúcom tíme je stále zložité a časovo náročné pochopenie existujúcej logiky.

Posledným bodom súhrnu analýzy procesu vývoja softvéru je vydávanie novej verzie. Tím čakáva na nahromadenie väčšieho počtu úloh v stave „Pripravené na testovanie“, kým vykoná vydanie novej verzie a aktualizáciu týchto úloh do stavu „Testovanie“. Tímu nič nebráni vo vydávaní nových verzií pre testovanie po dokončení každého samostatného celku a zároveň uskutočniť integráciu s verzovacím systémom, ktorá umožňuje automatickú aktualizáciu stavov pri vydaní novej verzie podľa identifikačného čísla.

Informačné systémy

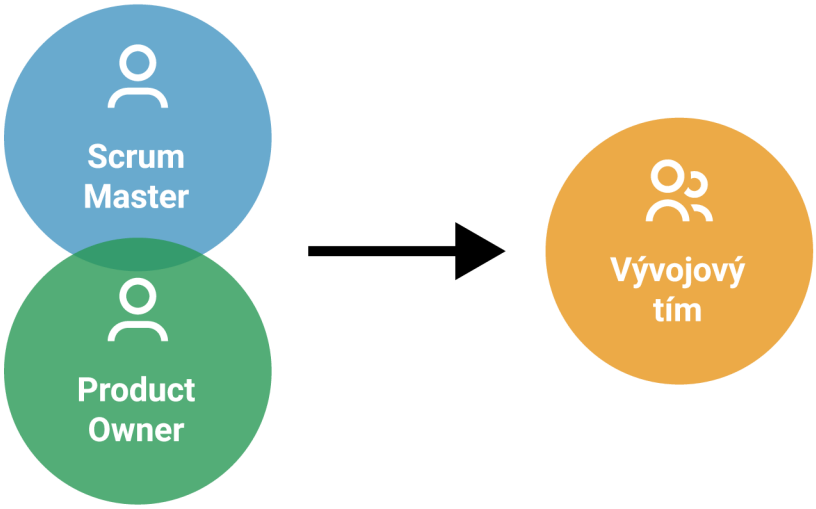
Tím na projekte využíva veľké množstvo moderných informačných systémov, ktoré im uľahčujú prácu. Keďže sa v ich prípade jedná o tím, ktorý nesedí spoločne v jednej kancelárii, využívajú informačné systémy i na komunikáciu medzi sebou. Konkrétme systém Miro je v tomto tíme využívaný pre jednu šablónu na počas Retrospektívy. Tu vidím veľký potenciál na zlepšenie a využitie tohto nástroja naplno.

3. Vlastné návrhy riešenia

Táto kapitola predstavuje návrh zmien, ktoré by mali byť prijaté na dosiahnutie správneho zavedenia Scrum metodiky na vybranom projekte. Návrhová časť vychádza z analýzy a bude aplikovať teóriu Scrum metodiky z teoretickej časti tejto diplomovej práce. Návrh sa bude venovať zmenám v tímových rolách, udalostiach, artefaktoch a v celkovom procese vývoja. Ďalej bude počítané aj s do budúcna sa zväčšujúcim tímom, pre ktorý bude nutné využiť škálovnú metodiku. V neposlednom rade návrh zmien pozostáva z finančného zhodnotenia a prínosov návrhu.

3.1 Tímová štruktúra

V Scrum metodike je odporúčaný maximálny počet členov scrumového tímu 9. Je to z dôvodu, aby dostal každý člen tímu dostatočný priestor, práca v tíme bola prehľadná a ľahko kontrolovateľná a aby stretnutia netrvali príliš dlho. V návrhu zostavenia scrumového tímu je tento počet členov dodržaný. Tím bude pozostávať zo Scrum Mastra, Product Ownera a 7 členov vývojového tímu. Na obrázku nižšie je vidno nová tímová štruktúra, po zavedení role Scrum master.



Obrázok č. 9: Členenie tímu po zavedení Scrum
(Zdroj: Vlastné spracovanie)

3.2 Roly

Scrum Master

Prvým návrhom z oblasti Rolí je najatie Scrum Mastra. Aj keď je táto rola v Scrum metodike nepovinná, v tomto prípade by tímu veľmi pomohla a odbremenila Product ownera. Vlastník produktu by sa tak mohol plne venovať svojim povinnostiam, ktoré pramenia z jeho role.

Scrum Master bude dohliadať na pravidelnosť a efektivitu všetkých mítingov tak, aby sa neopakovali súčasné chyby. Tím sa tak vyhne zbytočne dlhým mítingom, ktorých obsah častokrát skĺzava mimo agendu mítingu. Postará sa o pravidelné uskutočňovanie udalostí, ich plánovanie a efektívnosť. Dohliadne na to, aby všetky dôležité osoby boli na jednotlivých mítingoch prítomné, prevezme od Projektového manažéra vedenie mítingov a zpoznámkuje mítingy, ktoré si vyžadujú zápis. Následne zápis z mítingov bude poskytnutý relevantným osobám, a tak sa predíde zabudnutiu výstupov zo stretnutí.

Jednou z prvých úloh Scrum Mastra bude spoznať bližsie každého člena tímu, aby si získal ich dôveru. Na začiatku by mal Scrum Master zorganizovať školenia pre členov tímu. Hlavnou témou školenia je určite Scrum metodika a ďalšie témy vyplynú zo slabých miest, ktoré tím má, a v ktorých by sa mal zlepšovať.

Scrum Master bude radiť a pomáhať v správnom zavedení Scrumu aj vlastníkovi produktu. Jeho úlohou bude motivovať, podporovať všetkých členov tímu a starať sa o odstraňovanie prekážok, ktoré sa tímu naskytnú.

Z role Scrum mastera vyplýva aj sledovanie artefaktov a ich neustále vylepšovanie podľa potrieb tímu. Scrum Master bude sledovať a vysvetľovať metriky, ktoré tímu pomôžu zefektívniť aktuálny Sprint. Kokrétne metriky sú navrhnuté v časti artefakty.

Product owner

Ako už bolo spomenuté v analýze, rola projektového manažéra by mala byť nahradená rolou vlastníka produktu, ktorá odpovedá metodike Scrum. Je na mieste, aby zamestnanec prešiel školením pre product ownerov, aby tak lepšie pochopil svoju rolu a povinnosti. Scrum Master ho samozrejme bude podporovať a pomáhať mu v prechode na novú rolu

a prevezme časť jeho doterajších zodpovedností na seba. Ako vlastník produktu už samozrejme nebude ani vykonávať zastupujúcu rolu testera na tomto projekte. Takéto odbremenenie od dvoch úloh umožní product ownerovi plné sústredenie len na produkt.

Samozrejmou úlohou, ktorú bude zastávať, je zastupovanie zákazníka, komunikovať s ním, preberať jeho vízie a viesť ho celým procesom. Vlastník produktu už viac nebude robiť hĺbkovú analýzu požiadavky, ale jeho úlohou bude pripraviť len prvotnú analýzu a zvyšok prenechá vývojovému tímu. Aj naďalej bude spravovať Backlog a pridávať položky do Backlogu. Ako vlastník produktu bude mať na starosti to, aby všetci členovia tímu rozumeli položkám, bude ich vysvetľovať a predstavovať tímu produktovú víziu. Novou úlohou bude nastavovanie Sprint goalu alebo teda cieľa Sprintu. Product owner musí tento cieľ Sprintu komunikovať smerom k tímu, uistiť sa, že všetci mu rozumejú a dohliadať na to, že k nemu aj smerujú. Podstatná bude prioritizácia položiek, podľa ktorej si budú členovia tímu, ponovom už sami, plánovať Sprint.

Product ownerovi odpadne úloha ohodnocovania user stories story pointmi. Predsalen najlepšie vedia ohodnotiť položky samotní vývojári. A pre lepšie zvládanie tohto ohodnocovania bude prítomný Scrum Master, ktorý im pomocou vhodne zvoleného nástroja pomôže zhodnúť sa na konečnom počte story pointov. Okrem ohodnocovania user stories odpadne vlastníkovi produkutu aj úloha zostavovania Sprint Backlogu a prideľovania jednotlivých úloh konkrétnym členom tímu. Podľa Scrum metodiky budú mať na starosti rozdeľovanie položiek na prácu samotní vývojári.

Vývojový tím

Vývojový tím podľa návrhu bude patriť spoločne k radovým členom tímu. Počet členov vývojového tímu bude v tomto prípade 7, čo spĺňa aj odporúčanie zo Scrum metodiky, ktoré je uvedené v teoretickej časti tejto práce. Ako už bolo spomenuté, Scrumový tím musí byť samoorganizovaný a multifunkčný.

Čo sa multifunkčnosti týka, členovia tímu sa budú aj naďalej zastupovať a zdieľať svoje znalosti. Prvým návrhom, aby sa úplne predišlo tomu, že tester na začiatku Sprintu nemá dosť práce a ku koncu zas priveľa práce, s ktorou mu musí pomáhať Projektový manažér (Product owner), je lepšie spolupracovať ako tím na položkách Backlogu vždy úplne od začiatku. Túto skutočnosť spôsobuje to, že vývojový tím občas skĺzne do tradičného waterfallu. Najprv pracujú na analýze, potom príde vývoj a až na konci sa všetko otestuje.

Po zavedení Scrum metodiky by sa mal analytik (hociktorý člen vývojového tímu odpovedajúci pri vybranej položke za analýzu) zamýšlať nad tým, ako bude daná vec fungovať a tester sa na to zároveň musí pozerať z pohľadu, ako by sa daná funkcionalita dala rozbiť a rovno prichádza s prípadnými chybovými scenármi.

Ešte pre lepšiu multifunkčnosť tímu bude do budúcna dôležitá alokácia development tímu len na plný úväzok. Tomuto neodpovedá len jediný člen tímu-tester. Ako už bolo spomenuté v predošlej analýze, testerovi by sa mal zmeniť úväzok na plný v najbližších mesiacoch, po absolvovaní vysokej školy. Na zdieľanej znalosti bude možné v rámci tímu zapracovať pomocou rôznych workshopov, ktoré sa doposiaľ medzi členmi tímu nekonali.

V rámci samoorganizovanosti je navrhnutá lepšia komunikácia v tíme počas stretnutí tak, aby neskĺzla mimo agendu a taktiež vyhýbanie sa rušeniu mítingov, len z dôvodu, že Product owner nemôže byť prítomný, i keď jeho prítomnosť je dobrovoľná. Toto by mal zabezpečiť práve už spomínaný Scrum Master. Samoorganizovanosť sa vyznačuje aj tým, že si vývojový scrumový tím vyberá, plánuje a zostavuje Sprintový Backlog sám. Doteraz túto funkciu mal Projektový manažér, avšak v rámci Scrum metodiky je táto úloha čisto na vývojom tíme, rovnako aj s prideľovaním jednotlivých položiek konkrétnemu členovi tímu. Toto plánovanie sprintového Backlogu budú vývojári robiť na základe prioritizácie a sprint goalu, ktoré stanovuje vlastník produktu.

Zákazník

I keď zákazník nie je členom scrumového tímu, a jeho prítomnosť, záujmy a vízie zastrešuje product owner, zákazník má určite miesto v celom procese a aj tu je priestor na zlepšenia. Prvým návrhom je vylepšiť proces popisu nových požiadaviek od klienta, ktorý bude podrobnejšie rozpracovaný v návrhu požiadavky na funkcionalitu.

Určite prínosné ako pre zákazníka, tak i pre scrumový tím, je prítomnosť zákazníka na udalosti Review. Táto udalosť bude ideálna pre zapojenie zákazníka do tímu. Vývojový tím bude na Sprint Review predstavovať zákazníkovi funkcionalitu a tak od neho dostanú kvalitnejšiu a hlavne včasnejšiu spätnú väzbu, nie až na projektovom All-hands mítingu, ktorý je na konci kvartálu. Review sa môže zúčastniť nie len zákazník ako jeden predstaviteľ spoločnosti, ale aj zamestnanci z tejto spoločnosti, ktorí reálne produkt využívajú.

3.3 Udalosti

Na to, aby udalosti a mítingy, ktoré sa uskutočňujú na projekte odpovedali metodike Scrum, je nutné zaviesť zmeny. Predovšetkým pôjde o zmeny v pravidelnosti, dĺžke a priebehu udalostí, konkrétne ide o Backlog Refinement a Review. Návrh obsahuje aj pridanie udalostí tak, aby korešpondovali s vybranou metodikou. Nakoniec budú v tejto časti navrhnuté i udalosti/mítingy, ktoré sa v Scrume metodike nenachádzajú, ale pre projekt a tím by mohli byť prínosné.

Plánovanie

Plánovanie je udalosť, ktorá sa aktuálne na projekte uskutočňuje v posledný deň Sprintu a jej dĺžka je 30-60 minút. Prvým návrhom je presunutie tejto udalosti na prvý deň Sprintu, tak ako to definuje Scrum metodika. Na plánovaní sa budú zúčastňovať vývojári, vlastník produktu a moderovanie udalosti bude mať na starosti Scrum Master. Cieľom plánovania je naplánovať aktuálny Sprint a zostaviť Sprint Backlog. Zostavovanie Sprint Backlogu bude na vývojovom tíme. Položky si počas plánovania budú vyberať. Taktiež si medzi sebou vyberú aj to, kto bude na ktorých položkách pracovať.

Plánovanie bude samozrejme vychádzať z prioritizácie od vlastníka produktu tak, aby zohľadňoval cieľ Sprintu. Pri plánovaní je potrebné zohľadniť aj dostupnosť členov tímu na aktuálny Sprint. Od toho sa bude odvíjať aj počet story pointov naplánovaných v Sprinte. Plánovanie by malo byť rýchlejšie ako doposiaľ, pretože položky Backlogu už budú ohodnotené story pointmi. Toto ohodnocovanie bude po novom prebiehať na Backlog groomingu, ktorý sa bude konať pravidelne, približne v polovici Sprintu. Na plánovaní vlastník produktu schváli Sprintový Backlog, a tak potvrdí záväzok tímu dodať naplánovaný Sprint.

Tým, že časť bývalej agendy plánovania, t.j. ohodnocovanie user stories prebehne už v predstihu na inom mítingu, plánovanie bude podstatne kratšie. O kratšie plánovanie sa určite zaslúži aj Scrum Master, ktorý bude na celý proces dohliadať a moderovať túto udalosť tak, aby bola čo najefektívnejšia. Podľa kalendáru tímu by sa plánovanie mohlo uskutočňovať vždy prvý deň Sprintu-v stredu o 9:00 a jeho dĺžka bude naplánovaná na 30 minút.

Daily Scrum

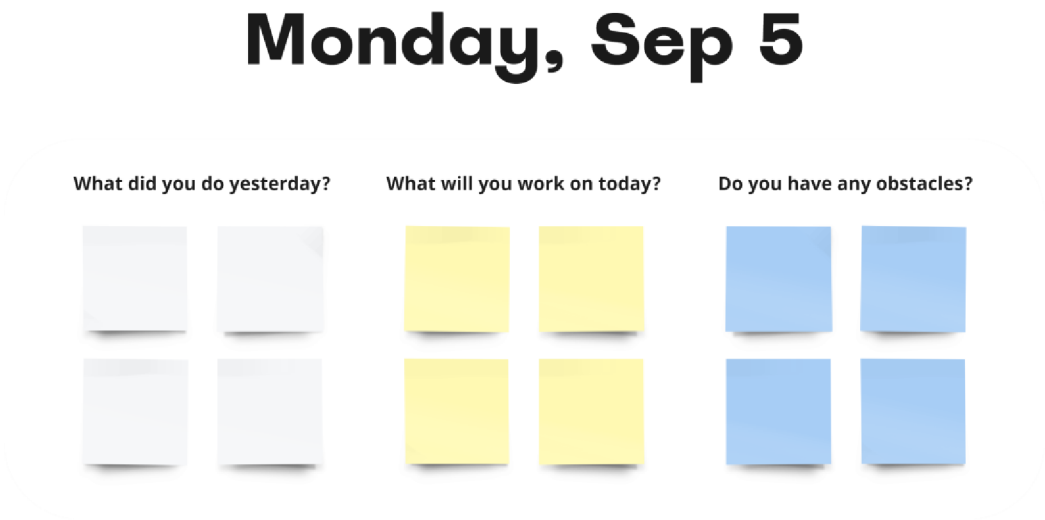
Daily scrum sa stane pravideľnou udalosťou, konajúcou sa každý deň ráno od 8:30 do 9:00. Dĺžka udalosti bude stanovená na 15 minút. Účasť vlastníka produktu bude dobrovoľná a bude na jeho zvážení, či je potrebný na daily, alebo nie, prípadne, či mu to jeho pracovná vyťaženosť dovolí. Podstatnými a povinnými účastníkmi budú členovia vývojového tímu. Účasť Scrum mastra ja tiež dobrovoľná, avšak zo začiatku by sa daily mal zúčastňovať, aby dohliadol na priebeh, dĺžku a na to, či sa táto udalosť pravidelne uskutočňuje. Už by sa nemalo stávať, že daily scrum sa zruší pre neprítomnosť Product ownera.

Členova vývojového tímu môžu využívať techniku koliesko, to znamená, že sa každý postupne dostane k slovu a odpovie na otázky:

- čo robil predchádzajúci deň a s akým výsledkom;
- čo plánuje robiť aktuálny deň;
- či vie o nejakej prekážke, ktorá by ho brzdila v plynulej práci.

Každý člen vývojového tímu dostane rovnaký čas na odpoveď na tieto otázky. Ak by z ich odpovedí vypadli nejaké požiadavky alebo prekážky, tím má možnosť prísť na riešenie počas tohto mítingu. Samozrejme sa nemôže stať, že sa opäť debata skĺzne len medzi napríklad 2 účastníkov a ostatní členovia by sa len prizerali. Aj z tohto dôvodu bude vhodná prítomnosť Scrum Mastra. Tieto otázky budú k dispozícii v popise každej udalosti Daily scrum v kalendári, tak aby vývojári mali tieto otázky kedykoľvek k nahliadnutiu. Prípadne, aby si mohli pred začatím mítingu jednoducho pomocou toho premyslieť, čo budú hovoriť. V 7. deň Sprintu pribudne do agendy Daily Scrum kontrola akčných bodov z Retrospektívy. Scrum Master si overí v akom stave je ich riešenie, prípadne pomôže s ich riešením on sám.

Scrum Master, prípade dobudúcna iná zvolená osoba počas Daily scrum mítingu otvorí aktuálny Sprint Backlog pre lepšiu vizualizáciu členov tímu. Prípadné poznámky alebo výstupy z daily, na ktoré tím nesmie zabudnúť bude vhodné zapísať. Na tento účel tím bude využívať nástroj Miro, na ktorý sú zvyknutí z Retrospektívy. Miro obsahuje aj niekoľko predpripravených šablón, vhodných na poznámky z Daily scrum mítingov.



Obrázok č. 10: Šablóna na Daily Scrum
(Zdroj: miro.com)

Na obrázku je zobrazená šablóna z informačného systému Miro, ktorú je možné využívať na Daily Scrum mítingoch. Každý člen si bude môcť predom napísať heslovite na lístočky svoje odpovede.

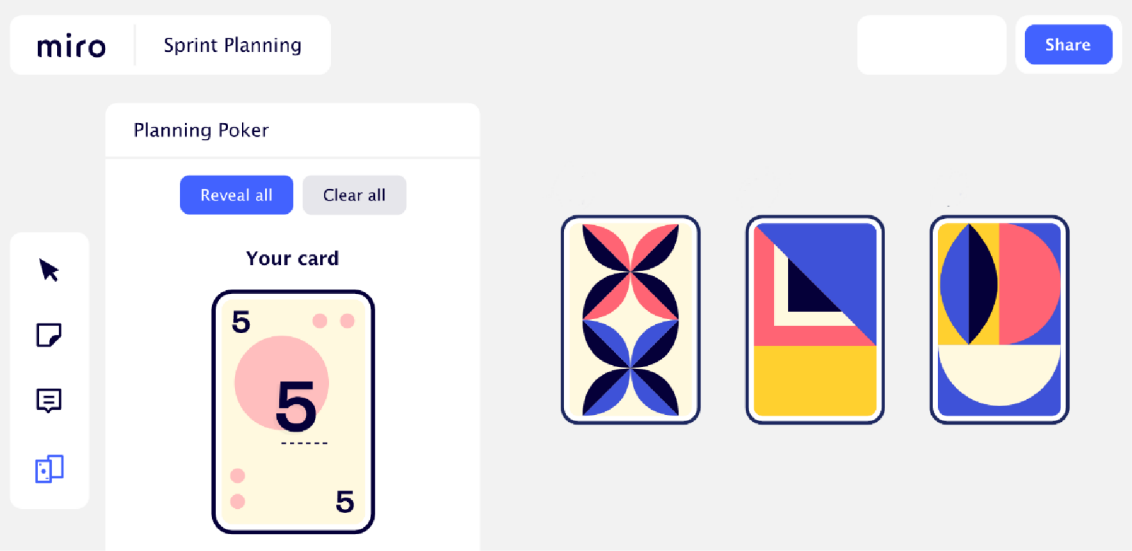
Backlog Refinement

Tímu úplne chýba priestor na rozpadanie väčších celkov na menšie, priestor na vysvetlenie položiek a ich lepšie porozumenie. Priestor na tieto aktivity dostanú na Backlog Refinemente. Bude sa tu preberať vízia produktu a spoločne sa budú venovať jednotlivým položkám a prioritám, ktoré stanoví Product owner.

Míting bude facilitovať Scrum master, a členovia vývojového tímu spolu s Product ownerom budú postupne prechádzať Backlog. Vlastník produktu bude k dispozícii na odpovedanie otázok vývojárom tak, aby položky Backlogu boli porozumené. Backlog Refinement bude užitočná aj ako predpríprava na plánovanie.

Jednou z aktivít, ktoré budú prebiehať je ohodnocovanie položiek story pointmi. Je vhodné, aby položky boli ohodnotené na cca 2 Sprinty dopredu. Aby sa predišlo predchádzajúcim nezhodám pri ohodnocovaní, tím využije metódu planning poker. Keďže sa tím nenachádza spoločne na jednom mieste a nie je v silách tímu sa vždy stretnúť, využije sa opäť nástroj Miro. V Mire je množstvo šablón na zjednodušenie a vizualizáciu Refinementu. Rovnako vie tento nástroj poslúžiť aj ako funkcia planning pokeru. Planning poker bude moderovať Scrum Master a zúčastňovať sa na ňom bude len

vývojový tím. Product owner bude v tomto prípade v roli pozorovateľa, prípadne odpovedať na vzniknuté otázky. Vzhľad funkcie planning poker v informačnom systéme Miro je zobrazený na obrázku nižšie.



Obrázok č. 11: Šablóna na Planning poker
(Zdroj: miro.com)

Na to, aby mohla byť položka pripravená na vývoj, musí spĺňať Definition of Ready. Práve tieto podmienky bude tím prechádzať na Refinemente a kontrolovať, či položka obsahuje všetko potrebné a či je tak pripravená na vývoj.

Backlog Refinement sa bude konať 6. deň Sprintu a bude naplánovaný na 2 hodiny. V prípade, že by jeden Backlog Refinement nestačil a tímu by viac vyhovovala frekvencia 2 Backlog Refinementy za Sprint, je možné pridať ďalší Refinement napríklad v 3. deň Sprintu. Do budúcna, keď sa tím viac zabehne v aktivitách Refinementu, je možné z neho spraviť workshop raz za niekoľko Sprintov a vývojári by Refinement mali ako samostatnú aktivitu počas Sprintu. V tom prípade by túto aktivitu vykonávali distribuovane, podľa toho kedy má kto čas.

Review

Prvou vecou, čo sa musí zmeniť na udalosti Review, je jeho pravidelnosť. V súčasnosti sa Sprint Review koná zväčša raz za 2 až 3 Sprinty podľa časovej vyťaženosti a iniciatívy Projektového manažéra. Podľa Scrum metodiky musí byť Sprint Review pravidelné a to na konci každého Sprintu. Sprint Review bude naplánovaná na posledný deň Sprintu v poobedných hodinách na 1 hodinu. Účastníkmi udalosti okrem vlastníka produktu,

Scrum mastra a vývojového tímu, bude aj zákazník. Zákazníkom sa v tomto prípade myslí hlavný zadávateľ (predstaviteľ zákazníka), ale aj všetci zamestnanci zo zákazníckej firmy, ktorí pracujú, alebo budú pracovať s vyvíjaným produktom.

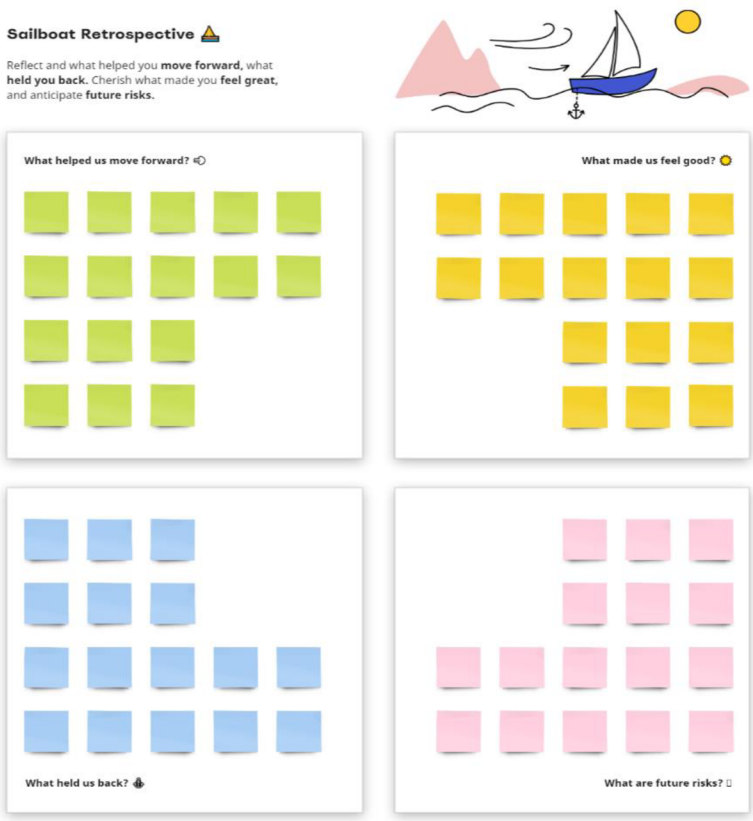
Na Sprint Review bude vývojový tím predstavovať zákazníkovi celý inkrement produktu. Súčasťou predvádzanej hodnoty budú len dokončené položky. Za dokončené položky sa v tomto prípade považujú len tie, ktoré spĺňajú Definition of Done. Ponovom už nebudú vývojári detailne prechádzať položku po položke a vyjadrovať sa k nej. Sprint Review bude zameraný predovšetkým na cieľ Sprintu. Zákazníka predsa nezaujíma každá položka, pretože takéto technické Review by mu ani neprinášalo žiadnu hodnotu.

Cieľom Sprint Review bude získať spätnú väzbu od zákazníka. Zákazník si bude môcť produkt vyskúšať a bude sa tam cítiť viac zapojený do procesu. Pre tím bude prezentácia produktu zároveň motiváciou a lepšie tak pochopia zákazníkove požiadavky.

Retrospektíva

Rovnako ako aj Sprint Review bude Retrospektíva pravideľná, vždy v posledný deň Sprintu. Keďže Retrospektíva uzatvára celý Sprint, bude sa konať po Sprint Review. Účastníkmi budú vlastník produktu, Scrum Master a celý vývojový tím. Scrum Master bude moderovať udalosť a využije na tieto účely opäť nástroj Miro. Doposiaľ bola využívaná len jedna šablóna v Miro, ale keďže tento nástroj ponúka veľké množstvo šablón, je vhodné ich občas obmieňať. Prípadne by Scrum Master mal zistiť, ktorá šablóna tímu najviac vyhovuje. Výmena šablón vonkoncom nebude len o vzhľade, pretože každá šablóna si pokladá trošku iné otázky a to donúti tím zamýšľať sa nad Sprintom vždy z trochu iného uhla. Zaujímavou voľbou bude Retrospektíva “loď“, kde budú odpovedať na otázky:

- Čo nám pomohlo sa posúvať vpred?
- Vďaka čomu sme sa cítili dobre?
- Čo nás zdržiavalo?
- Čo predstavuje budúci risk?



Obrázok č. 12: Šablóna na sailboat Retrospektívu
(Zdroj: miro.com)

Scrum Master okrem moderovania bude mať na starosti aj zapisovanie tzv. akčných bodov. Akčný bod bude mať nasledujúcu štrktúru:

- označenie z ktorého podnetu akčný bod vyplynul;
- krátky opis akčného bodu;
- člena/členov, ktorých sa akčný bod týka a budú sa mu venovať.

K týmto akčným bodom sa budú počas nasledujúceho Sprintu vracať a bude súčasťou agendy na Daily Scrum, vždy v 7. deň Sprintu. Na Daily Scrum sa skontroluje ako sa riešenie akčných bodov pohlo.

Meeting pre Product ownera a Scrum mastra

Míting medzi Scrum Mastrom a Product Ownerom nie je definovaný v Scrum guide, avšak pre tím, ktorý začína so Scrum metodikou, bude vhodný. Na tomto mítingu si budú môcť predať Scrum Master a Product owner informácie, nápady, stratégiu. Môže slúžiť aj ako čas, kedy Scrum Master pomáha Product Ownerovi s jeho rolou, dáva mu tipy, prípade spätnú väzbu na jeho úlohu v Scrumovom tíme. Môžu spoločne naplánovať

workshopy a školenia, ktoré tím posunú ďalej a pomôžu všetkým lepšie pochopiť Scrum. Tento míting bude vhodný prevažne zo začiatku ich spolupráce a bude naplánovaný na 15 minút raz za Sprint.

Projektový all hands meeting

Ani projektový all-hands meeting nie je definovaný ako udalosť v Scrum guide. Táto udalosť však členom tímu vyhovuje a dozvedia sa tam vždy zaujímavé veci. Je preto vhodné, aby aj naďalej bol tento míting súčasťou práce na produkte. Zákazník bude informovať celý Scrumový tím o biznise, ktorého súčasťou je ich vyvíjaný produkt, budú sa preberať plány a tímové kapacity do budúcna. Časť agendy projektového all-hands meetingu, kde bol zákazníkovi prezentovaný produkt bude presunutá do Sprint Review, tak ako je to stanovené v Scrum metodike. Aj naďalej sa bude tento míting konať raz za kvartál a bude naplánovaný na 1 hodinu.

Udalosť/Deň sprintu	1	2	3	4	5	6	7	8	9	10
Daily Scrum	X	X	X	X	X	X	X	X	X	X
Plánovanie	X									
Review										X
Retrospektíva										X
Backlog Refinement						X				

Obrázok č. 13: Kalendár po zavedení Scrum
(Zdroj: Vlastné spracovanie)

Na obrázku je zobrazený kalendár udalostí po aplikácii zmien. Zmenené boli frekvencie Daily Scrum mítingov, presunuli sa niektoré udalosti na iný deň a bol pridaný Backlog Refinement. Míting medzi Product ownerom a Scrum mastrom nie je zobrazený v kalendári, pretože ide o nepovinný míting len medzi týmito dvoma účastníkmi.

3.4 Artefakty a praktiky

V nasledujúcej časti budú navrhnuté zmeny v artefaktoch a praktikách. Bude sa jednať nielen o zmeny v rámci správneho nasadenia Scrum metodiky, ale taktiež o zmeny, ktoré vylepšia využívanie týchto artefaktov a praktík.

Definition of Ready a Definition of Done

Na vybranom projekte úplne chýbajú zostavené podmienky, ktoré musí spĺňať položka Backlogu pripravená na vývoj – Defnition of Ready a položky Backlogu, ktorá je hotová – Definition of Done.

Návrh Definition of Ready:

- Názov User Story – Obsahuje meno funkcionality
- Popis
- Story points – ohodnocovacia škála je 2, 4, alebo maximálne 8
- Priorita – určuje Product owner
- Test info – testovacie scenáre
- Návrh riešenia – popis návrhu vychádzajúceho z analýzy
- Preferovaný UI dizajn

Návrh Definition of Done:

- Napísaný kód – vytvorený merge request
- Code review – schválenie merge requestu pridelenou osobou
- Otestované – položka musela prejsť testovacími scenármi
- Dokumentácia – obsahuje internú a uužívateľskú dokumentáciu
- Akceptované Product ownerom
- Nasadené – dostupná nová verzia

Všetky takto spísané parametre by mali byť ľahko prístupné každému členovi tímu na Confluence pod zložkou DoR a DoD. Pre rýchly prístup k definícii pripraveného je vhodné vložiť jednotlivé body do udalosti Refinement v kalendári. Tím si samozrejme časom môže pridať ďalšie kritériá, podľa ich potrieb.

Scrum board

Tím využíva k lepšej prehľadnosti aktuálneho stavu Sprintového Backlogu Scrum board, alebo teda Scrum tabuľu. Toto je samozrejme v rámci Scrum metodiky v poriadku. Jediný problém, ktorý vyplynul z analýzy, je občasné nechcené presunutie, prípadne vymazanie položky niekým iným. Takéto Scrumové boardy sú využívané aj v rámci iných projektov v spoločnosti a tak je na mieste obmedziť prístupy k boardom. Prístup do scrumového boardu na tomto projekte bude mať len Scrumový tím zložený zo Scrum mastra, Product ownera a 7 vývojárov.

Sprint Backlog

Ako už bolo spomínané, Sprintový Backlog by si posprávnosti v Scrum metodike mali zostavovať členovia vývojového tímu. Preto je súčasťou návrhu pridelenie tejto úlohy samotným vývojárom. Vývojový tím si bude sám vyberať položky, na ktorých chce pracovať v Sprinte, ale samozrejme v rámci priorít a Sprintového cieľa.

Sprint goal

Z analýzy vyplynulo, že na projekte nie je stanovovaný žiaden cieľ Sprintu. Práve Sprint goal je však najdôležitejšiu časťou Scrumu, pretože predstavuje víziu a adresuje potreby zákazníka. Podstatné je aj správne tento cieľ Sprintu sformulovať. Nemôže byť príliš konkrétny, upnutý len na jednu funkcionalitu, ale musí predstavovať väčšiu funkcionalitu z Backlogu. Príklad správnej formulácie Sprint goalu pre tento konkrétny produkt:

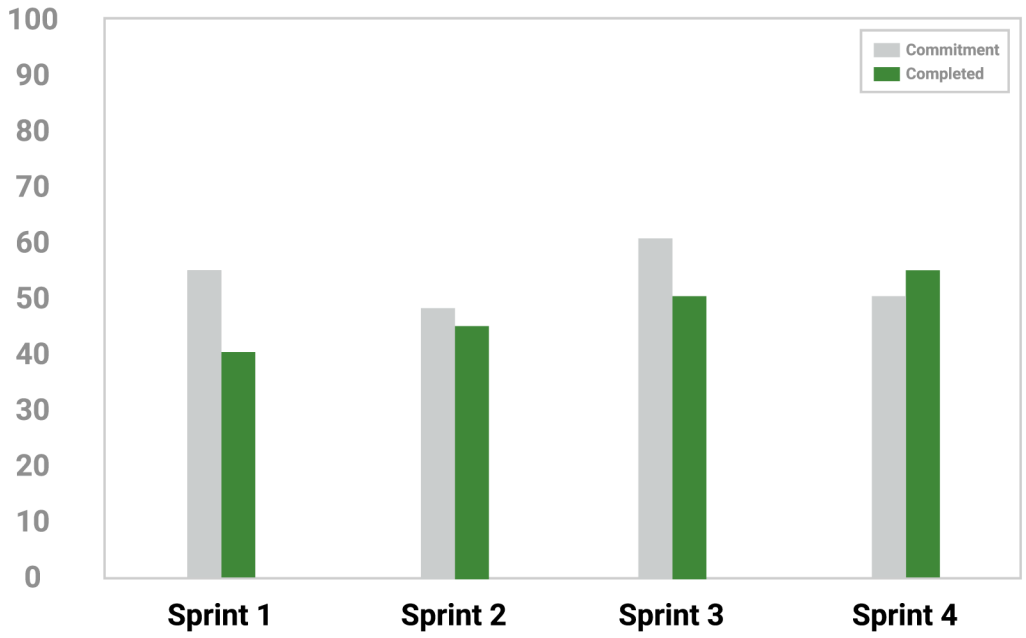
- Cieľ Sprintu: Zrýchliť výkup vozidiel
- Vybrané položky Backlogu: implementácia vlastného sprievodcu pre nafotenie vozidla, tlačenie dokumentov pre výkup priamo z mobilnej aplikácie, vlastný číselník pre manuálne zadávanie rozmerov pneumatiky.

Metriky

S lepším plánovaním Sprintu pomôžu tímu metriky. Metriky bude mať na starosti Scrum master, bude ich sledovať, vysvetľovať ich podstatu tímu, a bude učiť tím ako s nimi pracovať. Metriky nebudú len ďalšia pridaná práca pre tím, naopak by im mali pomôcť s hladším priebehom Sprintu, nie byť pre nich na obtiaž.

Prvou metrikou je velocity – rýchlosť. Táto metrika predstavuje počet story pointov, ktorý sa tím zaviazal dodať (commitment) a skutočný počet dodaných story pointov. Takéto poznatky im pomôžu lepšie naplánovať nasledujúci Sprint. Keď uvidia, že sa zaviazali

doručiť oveľa viac, než skutočne doručili, znamená to, že Sprint bol preplánovaný. V prípade, že doručia omnoho viac, než bol ich commitment, znamená to, že nasledujúci Sprint naplánujú viac. Na výsledky sa môžu pozrieť napríklad na Review, prípade na Retrospektíve. Na Respektíve potom môžu prebrať čo spôsobilo ich preplánovanie, alebo ich väčší počet dodaných story pointov. Graf sumarizujúci velocitu je súčasťou softvéru Jira, ktorý tím využíva.



Graf č. 1: Metrika velocity v informačnom systéme Jira

(Zdroj: Vlastné spracovanie podľa: Jira)

Ďalšie metriky, ktoré sú pre tím odporúčané, sú churn a balance. Keďže z analýzy vyplynulo, že tím má problém s narušovaním Sprintu, tieto metriky im pomôžu lepšie s narušením pracovať. Metrika churn im ukáže ako veľmi bolo do Sprintu zasiahnuté, a koľko story pointov bolo pridaných. Balance im zase ukáže, ako pracujú s narušením Sprintu, a či je tam balanc medzi pridanými a odobranými položkami. Bude vhodné, ak si tím nastaví vlastné percentá, ktoré sú pre narušenie Sprintu v ich prípade ešte akceptovateľné, a kedy už je narušenie Sprintu neprípustné. S týmto nastavením im pomôže Scrum master, ktorý bude monitorovať vybrané metriky.

Bug

Bug je správne využívaný pre označenie chyby v produkte. Problém nastáva s jeho prioritizáciou, ktorá v súčasnosti prakticky neexistuje. Väčšina bugov, ktorá vznikne po

vytvorení testerom, alebo vývojárom, je automaticky hodených do aktuálneho Sprintu. Návrhom je, aby bol každý bug po jeho vytvorení prioritizovaný Product Ownerom. Tie bugy, ktoré prinášajú veľkú biznisovú hodnotu poputujú do aktuálneho Sprintu. Tie, ktorých vyriešenie až tak nehorí, počkajú v Backlogu na ďalší Sprint. Súčasťou návrhu je aj návrh štruktúry ohlásenej chyby, ktorý je spísaný nižšie v návrhoch procesu vývoja.

3.5 Proces vývoja

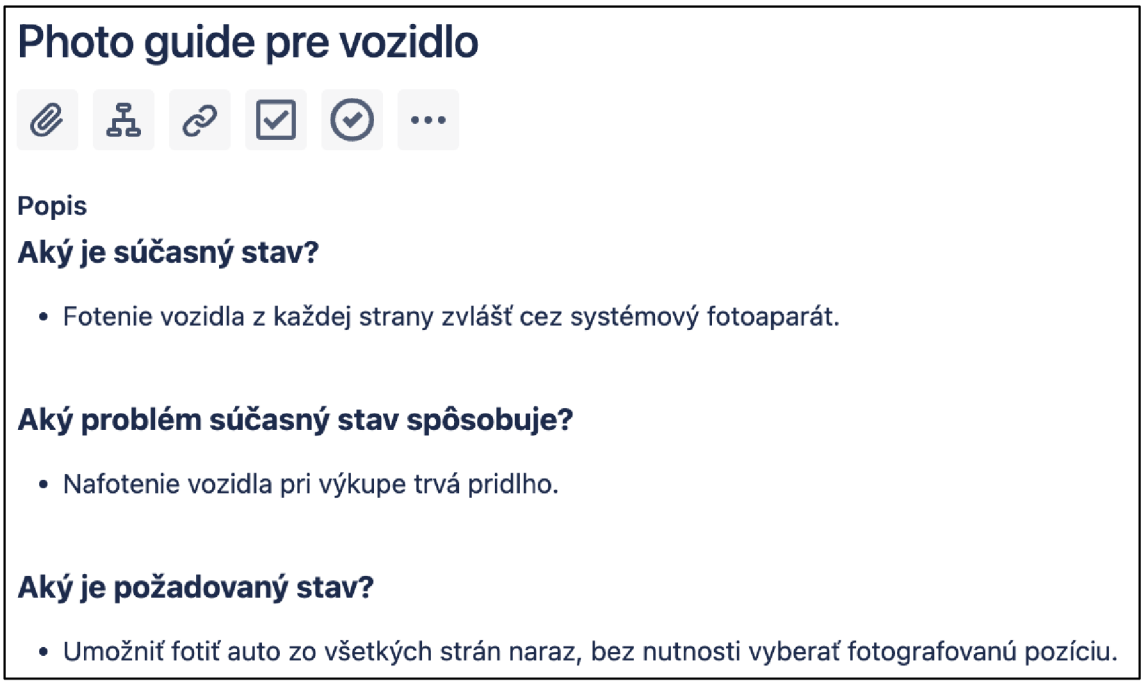
Požiadavka na funkcionalitu

Prvý návrh pre proces vývoja softvéru sa týka nových požiadaviek na funkcionalitu. Osvedčenou metódou popisu nových požiadaviek od klienta je popis problému, nie priamo jeho návrh na riešenie problému. Vývojové tímy majú s vývojom ďaleko väčšie skúsenosti než klient, a zároveň zaužívané konkrétne procesy, ktorými by mala každá požiadavka prejsť, než sa dostane do vývoja. Pokiaľ klient v popise popíše ihneď konkrétnu funkcionalitu, ktorú chce od tímu vypracovať, dochádza tak k preskočeniu analýzy a návrhu riešenia. Návrhom je začať popisovať novú požiadavku na funkcionalitu zodpovedaním troch základných otázok:

- Aký je súčasný stav?
- Aký problém súčasný stav spôsobuje?
- Aký je požadovaný stav?

Vďaka odpovediam na tieto 3 základné otázky získa Product Owner a tím dostatočný počet informácií o novej požiadavke, vďaka ktorým môžu vypracovať analýzu a návrh riešenia. Dodatočný popis pre novú požiadavku je taktiež v poriadku, ale nesmú chýbať odpovede na 3 spomínané otázky.

Obrázok nižšie pozostáva z návrhu šablóny v Jire pre požiadavku na funkcionalitu. Obsahuje aj konkrétne vyplnenie otázok tak, ako by to pre správne použitie a urýchlenie práce na požiadavke malo vyzerať.



Obrázok č. 14: Šablóna na vytvorenie požiadavky v informačnom systéme Jira
(Zdroj: Vlastné spracovanie)

V súčasnosti typ požiadavky určenej na úpravu existujúcej funkcionality nemá žiadnu definovanú štruktúru pre popis. Rovnakú štruktúru popisu môže využiť aj tento typ požiadavky. Pri novej požiadavke bude najčastejšou odpoveďou absencia riešenia v súčasnom stave, ale pri požiadavke na úpravu bude už táto odpoveď ďaleko konkrétnejšia, a bude popisovať reálny stav. Pokiaľ bude táto štruktúra zachovaná pri oboch typoch požiadaviek, tímu to sprehľadní vývoj a skráti čas potrebný na pochopenie popisu požiadavky.

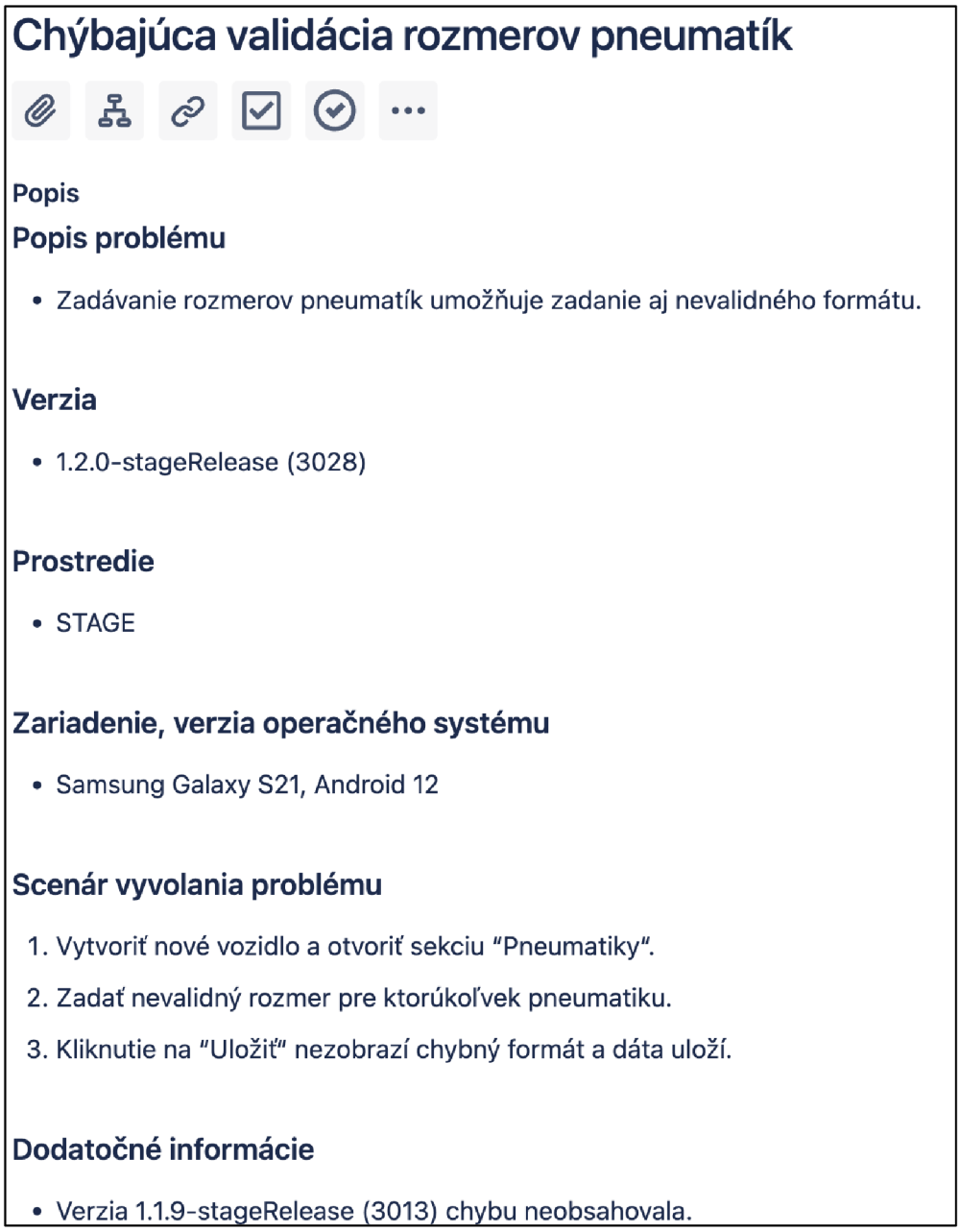
Návrh pre vytvorenie štruktúry popisu požiadavky určenej na opravu funkcionality (chyby) sa od predchádzajúcich dvoch typov odlišuje. Spomínané základné otázky sú síce aj pre tento typ požiadavky vyhovujúce, avšak musia byť rozšírené o oveľa detailnejšie a konkrétnejšie informácie. Aby vývojový tím mohol úspešne realizovať hĺbkovú analýzu a návrh riešenia pre požiadavku určenej na opravu, musí vedieť, v akej verzii problém nastal. Pretože počas testovania vývoj nestojí a môže sa stať, že od nálezu problému na opravu boli vydané už ďalšie verzie. Pokiaľ číslo (identifikátor) verzie neobsahuje informáciu, o aké prostredie (vývojové, produkčné, ...) sa jedná, je potrebná špecifikácia prostredia. Veľmi nápomocné je aj vedieť, aké akcie neočakávanému chovaniu softvéru predchádzali. Vývojári sa stretávajú s nemožnosťou vyvolať chybový stav, pretože

nepoznajú konkrétny scenár, ktorý k tomuto stavu viedol. Nápomocná je aj znalosť dát (prihlásený užívateľ, aktuálne zobrazená stránka informačného systému, ...), pri ktorých chyba nastala. V dnešnej dobe býva informačný systém využívaný na obrovskom počte rozdielnych zariadení od rozdielnych výrobcov s rozdielnymi operačnými systémami. Preto sa nesmie zabudnúť ani na toto kritérium pri popise požiadavky na opravu. V neposlednom rade môže pomôcť aj popis, kedy problém naopak nenastal. Vývojový tím tak môže získať stopu pre odhalenie príčiny problému.

Z dôvodov popísaných v predchádzajúcom odstavci, návrh štruktúry popisu chyby pozostáva z nasledujúcich bodov:

- Popis problému
- Verzia
- Prostredie
- Zariadenie, verzia operačného systému
- Scenár vyvolania problému
- Dodatočné informácie

V nasledujúcom obrázku je zobrazená šablóna pre popis bugu v informačnom systéme Jira. Návrh štruktúry popisu chyby je rovno zobrazený na konkrétnom príklade, ako by mal takýto popis vyzerať.



Obrázok č. 15: Šablóna na vytvorenie bugu v informačnom systéme Jira
(Zdroj: Vlastné spracovanie)

Analýza požiadavky

Návrhom pre analýzu požiadavky je rozdelenie analýzy na povrchovú a hĺbkovú. Celkovú analýzu už naďalej nebude vykonávať projektový manažér, pretože by to nemalo spadať do jeho kompetencií a zároveň v návrhu táto rola prestane existovať a nahradí ju Product Owner. Povrchovú analýzu bude vykonávať spomínaný Product Owner na základe požiadaviek od klienta alebo vývojového tímu. Hĺbkovú analýzu bude vykonávať člen vývojového tímu, ktorý bude za túto analýzu zodpovedný.

Zároveň vytvorenie požiadavky pre úpravu alebo opravu existujúcej funkcionality bude aj naďalej môcť vytvoriť ktokoľvek z tímu, avšak požiadavka bude prioritizovaná vlastníkom produktu a až podľa tejto prioritizácie bude zaradená buď do Sprint Backlogu alebo Backlogu, kde bude čakať na ďalšie Sprinty.

Návrh riešenia

Novým návrhom pre definovanie návrhu riešenia je umiestniť špecifikáciu do informačného systému Confluence. V Confluence je možné vytvoriť samostatné stránky a k nim ďalšie podstránky, čím bude zabezpečená dostatočná hierarchická úroveň pre tento typ dokumentácie. Každá samostatná stránka obsahuje nadpis a popis, ktorý môže byť ľubovoľne formátovaný a obsahovať napríklad odkazy na design. Zároveň je možné tento informačný systém jednoducho prepojiť s požiadavkami v Jira a pri prepájaní je možné aj vyhľadávanie podľa nadpisu stránky. Správa prístupu do Confluence je jednoducho konfigurovateľná a v prípade, kedy klient nemá prístup do Jira, stále môže mať prístup do Confluence a čítať (prípadne komentovať) jednotlivé stránky.

Podobnú formu zápisu návrhu poskytuje aj rozhranie systému Jira v jednotlivých požiadavkách, avšak spätné dohľadanie návrhu v budúcnosti je už značne komplikovanejšie.

Ďalším návrhom je preddefinovanie šablóny pre jednotlivé typy požiadaviek. Prázdny popis v takomto prípade už nikdy nenastane a priradená zodpovedná osoba dopracuje návrh až po doplnení podľa tejto šablóny na základe vykonanej analýzy. Šablóny budú vychádzať z navrhovaných otázok a bodov popísané v návrhovej časti pre požiadavku na funkcionalitu na začiatku tejto kapitoly.

Najdôležitejším návrhom tejto časti je zavedenie procesu schvaľovania návrhu prostredníctvom stretnutia tzv. Refinement.

Vývoj

Keďže požiadavku v systéme Jira môže do stavu s označením „Blokované“ presunúť ktokoľvek, je potrebné vo vývojovom tíme stanoviť pravidlo, čo je k tomuto úkonu neodbytnou súčasťou. Návrh definuje pravidlo o odôvodnení zmeny stavu požiadavky na stav s označením „Blokované“ pomocou prepojenia s inými požiadavkami, alebo prostredníctvom komentáru. V prípade, že je požiadavka blokovaná inou požiadavkou

(napríklad frontend čaká najprv na implementáciu logiky od backend), je potrebné vytvoriť prepojenie dvoch alebo viacerých požiadaviek. Veľkou výhodou je existencia prepojenia v systéme Jira. V prípade, kedy nie je možné využiť prepojenie požiadaviek (napríklad pri čakaní na vyjadrenie od podpory využívanej služby), bude využitá sekcia komentárov.

Druhým návrhom pre vývoj je vytvorenie programovej dokumentácie priamo v zdrojovom kóde. S rastúcimi požiadavkami vzniká dopyt po nových členoch tímu, ktorým dokumentovaný zdrojový kód výrazne uľahčí a urýchli pochopenie existujúcich procesov a logiky.

Testovanie

Posledným návrhom procesu vývoja je častejšie vydávanie novej verzie určenej na testovanie. Tím nie je technologicky obmedzený počtom vydaných verzií, a preto je navrhované automatizovať vydávanie novej verzie systému vždy pri zavedený nových zmien vo verzovacom systéme Gitlab prostredníctvom Merge Request. Gitlab umožňuje jednoducho nastaviť automatické vydávanie novej verzie hneď v niekoľkých možnostiach (napríklad pri spomínanom Merge Request).

Dodatočným návrhom je aj sprostredkovanie integrácie systémov Jira a Merge Request, čím bude docielené automatické presunutie požiadaviek do stavu „Testovanie“. Pre uskutočnenie takejto integrácie je nutné každý Merge Request označiť prostredníctvom unikátneho čísla, ktorým disponuje každá požiadavka v systéme Jira. Na základe tohoto označenia bude možné automatizáciu uskutočniť.

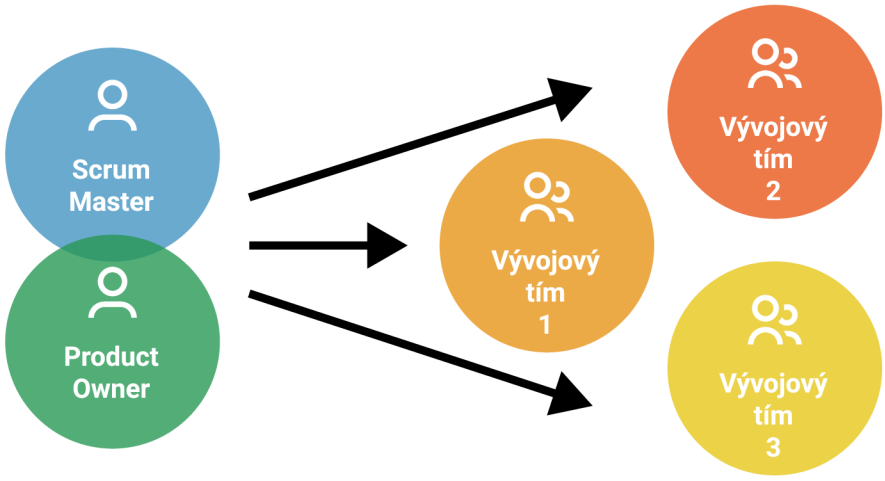
3.6 Škálovanie

Z analýzy vyplynulo, že produkt má v pláne rásť, a je naplánovaný vývoj časti produktu určený pre koncových užívateľov. S takýmto rozsiahlym vývojom by sa malo začať v druhom kvartále roku 2023 a vývojárska spoločnosť bude na projekt alokovať ďalších minimálne 6 zamestnancov. To znamená, že v budúcnosti by celkovo na produkte malo byť alokovaných minimálne 15 zamestnancov.

Z teoretickej časti je známe, že 15 členov scrumového tímu nie je v súlade so správnym využívaním metodiky Scrum. Na takéto účely je vhodnejšie využiť škálovanú metodiku LeSS (Large Scale Scrum).

V tomto prípade budú vývojári rozdelení do 3 tímov, ale spoločne budú pracovať na jednom produkte. Tímy budú rozdelené podľa častí produktu:

- 1. tím – výkup
- 2. tím – predaj
- 3. tím – sklad



Obrázok č. 16: Tímová štruktúra po zavedeni LeSS
(Zdroj: Vlastné spracovanie)

Riadenie projektu bude musieť prejsť istou transformáciou, pretože medzi Scrum metodikou a Less sú isté rozdiely. V prvom rade sa zavedie Sprint planning 1 a Sprint planning 2.

Sprint planning 1 sa budú zúčastňovať všetci členovia 3 tímov, Product owner i Scrum master. Prípadne bude možné vybrať 2 zástupcov z každého tímu, a len tí sa na Sprint planningu 1 zúčastnia. Tímy sa spoločne stretnú a Product owner im predstaví prioritné položky Backlogu. Samotné tímy sa dohodnú, ktorý bude pracovať na ktorej položke z Backlogu. Zodpovedajú sa prípadne otázky, ktoré neboli vyriešené na Refinemente a definuje sa Sprint goal. Sprint planning 2 má každý tím zvláš,ť a na ňom už si vývojový tím vyberá položky, ktoré za daný Sprint vyhotoví. To znamená, že každý tím bude mať svoj vlastný Sprint Backlog.

Daily Scrum bude prebiehať rovnako ako pri klasickom Scrume. To znamená, že členovia tímu budú odpovedať na 3 otázky:

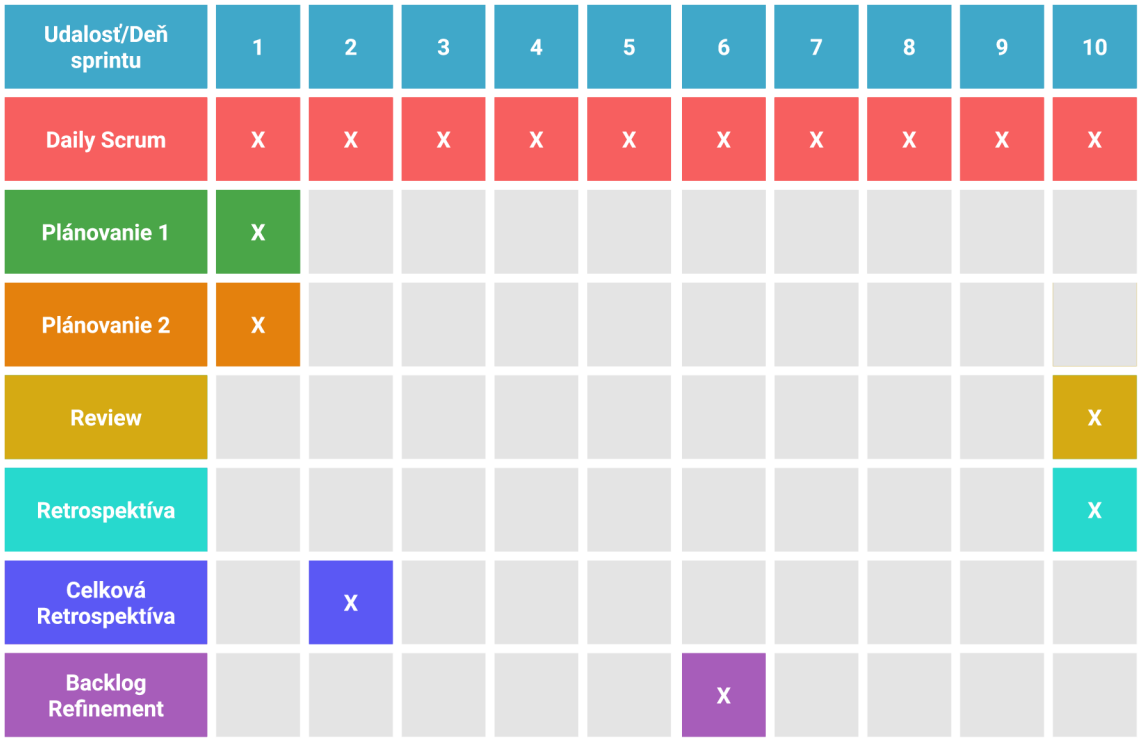
- Na čom som pracoval včera?

- Na čo budem pracovať dnes?
- Čo ma brzdí v práci?

Každý tím bude mať svoj vlastný daily scrum. V prípade, že bude potrebná väčšia spolupráca tímov, pre lepšie zdieľanie informácií, môžu navštevovať jednotlivci daily scrum ostatných tímov.

Product Backlog Refinement bude v tomto prípade prebiehať pre každý tím zvlášť. Budú tu rovnako ako v Scrume rozoberať väčšie položky na menšie, prechádať položky tak, aby im všetci rozumeli, a ohodnocovať položky. Review bude prebiehať spoločne pre všetky tímy naraz, a bude naplánované na 1,5 hodiny. Zúčastnia sa ho celé vývojové tímy, Porduct owner a moderovať ho bude Scrum master. Okrem nich bude pozvaný aj zákazník a všetky zainteresované strany.

Retrospektívy sa budú konať až dve. Prvá Retrospektíva bude pre každý tím zvlášť a bude prebiehať rovnako ako v klasickom Scrume. Druhá Retrospektíva bude spoločná pre všetky tímy. Cieľom bude rozobrať medzitímové a organizačné problémy. Na spoločnej Retrospektíve sa bude zúčastňovať Product owner, Scrum master a dvaja zástupcovia z každého tímu. V dôsledku toho, že sa v posledný deň uskutočňuje viacero udalostí, celková Retrospektíva sa bude konať až v druhý deň nasledujúceho Sprintu. Jej účastníci tak budú oddýchnutejší a budú mať čas sa na celkovú Retrospektívu pripraviť.



Obrázok č. 17: Kalendár po zavedení LeSS
(Zdroj: Vlastné spracovanie)

3.7 Finančné zhodnotenie návrhu

Z návrhu riešenia nevyplynuli žiadne náklady na nové informačné systémy. Zvýšenie nákladov po zavedení Scrum predstavuje zamestnanie nového člena tímu, konkrétne na rolu Scrum mastra. Pre odhad nákladov na mesačnú mzdu mediorného až senioreného Scrum mastra boli využité aktuálne ponuky práce.

Tabuľka č. 1: Náklady na nového zamestnanca
(Zdroj: vlastné spracovanie)

Rola	Mesačné náklady pre zamestnávateľa	Mesačná hrubá mzda
Scrum master	66 900 Kč	50 000 Kč

Ďalším nákladom bude nákup kurzu pre Product ownera. Z dôvodu, že predtým zastupoval rolu projektového manažéra, bude potrebné, aby absolvoval kurz, ktorý ho naučí lepšie zvládať jeho novú rolu. Kurzom by mal prejsť ešte pred aplikáciou Scrum metodiky na vybranom projekte. Vybraný kurz je od spoločnosti Scrum.org a trvá 2 dni. Po dokončení kurzu Product owner absolvuje skúšobný test.

Tabuľka č. 2: Náklady na kurz

(Zdroj: vlastné spracovanie)

Kurz	Cena s DPH	Cena bez DPH
Professional Scrum Product owner	32 670 Kč	27 000 Kč

Náklady na zamestnanie Scrum mastra sú udané mesačne. Vybraný kurz pre Product ownera bude pre firmu jednorázový náklad.

3.8 Prínosy a limity návrhu

Implementácia navrhovaných riešení, ktoré sa zameriavajú na aplikovanie Scrum metodiky a na zlepšenie procesu vývoja na vybranom projekte, by mala priniesť výhody v nasledujúcich oblastiach:

- Efektívnosť – bude dosiahnutá konkrétnejším definovaním právomocí a zodpovedností rolí. Dôjde k stanoveniu presných úloh jednotlivých rolí, a to umožní lepšie zameranie tímu na prácu. Efektívnosť pozdvihne aj hlbšie využívanie nástroja Miro, logickejšie nadväznosti mítingov, presná agenda udalostí aj správne navrhnutie artefaktov.
- Plánovanie a realizácia – nová forma plánovania umožní efektívnejšiu prioritizáciu úloh a bugov, rýchlejší vývoj, menej narušené Sprinty. Vďaka metrikám môžu vyhodnotiť naplánovaný predchádzajúci Sprint a tak eliminovať chyby v plánovaní aktuálneho Sprintu. K urýchleniu plánovania prispeje aj nová udalosť Backlog Refinement, vrámci ktorej dôjde ku zmysluplnému rozpadu väčších celkov aj k ich presnejšej estimácii. Vylepšenie plánovania prispeje k eliminovaniu rizika zapadnutiu položiek v Backlogu.
- Zákazník – vďaka častejšiemu zapojeniu zákazníka dôjde k presnejšiemu splneniu jeho vízie a požiadaviek, prispeje tomu aj nová štruktúra zadávania požiadaviek. Zákazníkova pravidelná účasť na Review skvalitní spätnú väzbu, ktorú dá vývojovému tímu. Vďaka kvalitnejšiemu dodanému produktu narastie celková zákazníkova spokojnosť. Jeho spokojnosť sa môže odraziť na referenciách o firme.

- Kvalita produktu – K celkovej kvalite prispeje definovanie Definition of Done a Definition of Ready, testovanie pomocou častejších buildov, informovanosť tímu. Nové návrhy na vývoj prinesú detailnejšiu analýzu problémov. Tým sa dosiahne vyššia kvalita konečného riešenia.
- Tím – celkovo sa tímu zlepší pozornosť a chápanie vízie produktu aj zásluhou zavedenia Sprint goalu, či lepšiemu vysvetľovaniu položiek Porduct ownerom. Pomocou Retrospektívy dosiahnu pravidelné zisťovanie problémov vo vnútri tímu a predchádzanie problémov vďaka celkovej častejšej komunikácii. Toto všetko prispeje tiež k lepšej atmosfére v tíme.
- Nové projekty – návrh môže poslúžiť ako predloha k zavedeniu Scrum metodiky do celej vývojárskej spoločnosti. Prínosom je i zohľadnenie nárastu produktu a tímu pomocou škálovania.

Okrem prínosov má predložený návrh aj svoje limity:

- Neochota ku zmenám – môže nastať neochota ku zmenám zo strany vývojového tímu, alebo zo strany spoločnosti.
- Náklady – spoločnosť nebude chcieť vynaložiť dodatočné náklady na zamestnanie Scrum mastra, či zaplatenie kurzu pre Product ownera.

Záver

Cieľom diplomovej práce bola aplikácia metodiky Scrum a návrhy na zlepšenie procesu vývoja softvéru na vybranom projekte. Diplomová práca pozostáva z týchto hlavných častí: teoretické východiská práce, analýza problému a súčasná situácia, vlastné návrhy riešenia.

Toretické východiská práce mali za úlohu v prvom rade vysvetliť vývoj softvéru, jeho životný cyklus a projekt vývoja softvéru. Následne bol popísaný tradičný prístup k riadeniu softvérových projektov a jeho konkrétne metodiky ako vodopádový model, špirálový model a metodiky RUP a UP. V poslednej časti teórie bol vo všeobecnosti popísaný agilný prístup k riadeniu softvérových projektov. Rozobraté bolo extrémne programovanie, Lean development a Kanban. Podrobnejšie bola opísaná metodika Scrum so všetkými jej náležitosťami ako sú roly, udalosti, artefakty a praktiky. Okrem Scrum metodiky sa teoretická časť venovala i škálovanej metodike LeSS pre viacero tímov.

Kapitola analýza problému a súčasná situácia opisuje spoločnosť a vybraný projekt. Ďalej sa venuje zanalyzovaniu všetkých rolí na projekte, udalostí, artefaktov a praktík. Rozobratý a opísaný bol aj proces vývoja softvéru na vybranom projekte, ktorý pozostáva z častí: požiadavka na funkcionalitu, analýza požiadavky, návrh riešenia, vývoj, testovanie a nasadenie. Ďalej boli opísané informačné systémy, ktoré využívajú zamestnanci a v poslednom rade vyhodnotenie analýzy. Vyhodnotenie analýzy obsahovalo predovšetkým opis nedostatkov spojených s procesom vývoja a rozdielnosti od metodiky Scrum.

V návrhu riešenia boli zvolené rovnaké oblasti ako pri analýze a to pre lepšiu prehľadnosť návrhovej časti. Keďže sa v návrhovej časti jednalo predovšetkým o aplikáciu metodiky Scrum, najprv bola navrhnutá tímová štruktúra. Následne boli navrhnuté zmeny v oblasti rolí, vrátane pridania roly Scrum mastra. Ďalej v časti artefakty a praktiky boli navrhnuté zmeny tak, aby bola dodržaná metodika Scrum. Rovnako boli navrhnuté zmeny v udalostiach, nielen aby odpovedali metodike, ale vychádzalo sa aj zo skúseností z praxe. V procese vývoja boli navrhnuté zmeny, ktoré vedú k celkovému zlepšeniu procesu od požiadavky až po nasadenie softvéru. V návrhovej časti bolo počítané i s narastajúcim tímom, v tom prípade by Scrum metodika

už nestačila a bola navrhnutá pre tieto prípady škálovaná metodika LeSS. Finančné zhodnotenie návrhu obsahuje mesačné náklady na nového člena tímu a náklady na absolvovanie kurzu Professional Scrum Product owner. Nakoniec boli v časti prínosy a limity návrhu sformulované prínosy z oblasti efektívnosti, plánovania, zákazníka, kvality produktu, tímu i nových projektov a samozrejme aj limity návrhu.

Táto diplomová práca slúži ako podklad pre správnu aplikáciu Scrum metodiky na vybranom projekte a ako podklad pre zlepšenie procesu vývoja softvéru. Do budúcna môže firma využiť tento návrh pre aplikáciu metodiky Scrum na zvyšných svojich projektoch, prípadne aj škálovanú metodiku LeSS, ak by projekt narástol, tak ako je naplánované. Cieľ práce aplikácia Scrum metodiky a návrhy pre zlepšenie procesu vývoja tak boli naplnené.

Zoznam literatúry

DOLEŽAL, Jan. *Projektový management: Komplexně, prakticky a podle světových standardů*. Praha: Grada Publishing, 2016. ISBN 978-80-247-5620-2.

Education.wiki. *Co je vývoj softwaru?* [online]. 2020 [cit. 2022-04-04]. Dostupné z: https://cs.education-wiki.com/9575463-what-is-software-development

KADLEC, Václav. *Agilní programování*. Brno: Computer Press, 2004. ISBN 80-251-0342-0.

KOMZÁK, Tomáš. *Řízení IT projektů pro úplné začátečníky* [online]. Brno: Computer Press, 2013 [cit. 2022-09-03]. ISBN 978-80-251-3791-8. Dostupné z: https://docplayer.cz/11633426-Tomas-komzak-rizeni-it-projektu-pro-uplne-zacatecniky.html

LARMAN, Craig a Bas VODDE. *Large-Scale Scrum*. 2017. Indiana: Pearson Education. ISBN 978-0-321-98571-2.

MARTINŮ, Jiří a Petr ČERMÁK. *Metodiky vývoje software* [online]. Olomouc: Moravská vysoká škola Olomouc, 2018 [cit. 2022-09-03]. Dostupné z: https://dl1.cuni.cz/pluginfile.php/864918/mod_resource/content/1/Metodiky-v%C3%BDvoje-software-studijn%C3%AD-text.pdf

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

Newgenapps.com. *Fáze a modely vývoje softwaru LifeCycle* [online]. 2021 [cit. 2022-04-04]. Dostupné z: https://www.newgenapps.com/cs/blogs/phases-and-models-of-software-development-lifecycle/

SCHWABER, Ken a Jeff SUTHERLAND. *Scrum Guide* [online]. In: . 2020 [cit. 2022-04-04]. Dostupné z: https://scrumguides.org/index.html

SCHWALBE, Kathy. *Řízení projektů v IT: Kompletní průvodce*. Brno: Computer Press, a. s., 2010 [cit. 2022-04-04]. ISBN 978-80-251-2882-4.

ŠOCHOVÁ, Zuzana a Eduard KUNCE. *Agilní metody řízení projektů*. Brno: Computer Press, 2019. ISBN 978-80-251-4961-4.

ŠOCHOVÁ, Zuzana. *Skvělý Scrum Master.* Brno: Computer Press, 2018. ISBN 978-80-251-4927-0.

Zoznam obrázkov

Zoznam grafov

Zoznam tabuliek