

Vysoká škola ekonomická v Praze

Fakulta informatiky a statistiky



Metodika vývoje mobilních aplikací

DIPLOMOVÁ PRÁCE

Studijní program: Aplikovaná informatika

Studijní obor: Podniková informatika

Autor: Bc. Lukáš Růžička

Vedoucí diplomové práce: Ing. Václav Oškrdal, Ph.D.

Praha, prosinec 2020

Prohlášení

Prohlašuji, že jsem diplomovou práci Metodika vývoje mobilních aplikací vypracoval samostatně za použití v práci uvedených pramenů a literatury.

V Praze dne 6. prosince 2020

.....

Lukáš Růžička

Poděkování

Rád bych tímto poděkoval Ing. Václavu Oškrdalovi, Ph.D., za vedení mé diplomové práce a cenné rady, které ve značné míře pomohly směřování k její současné podobě. Poděkování patří také mé přítelkyni a rodině za podporu při celém studiu, bez které bych se do tohoto bodu nedostal.

Abstrakt

Předmětem této práce je vytvoření metodiky vývoje vhodné pro prostředí mobilních aplikací. Základu pro tvorbu metodiky je dosaženo pomocí analýzy specifík prostředí a existujících metodik. Na základě této analýzy je vybrána nejbližší metodika, která je dále upravena tak, aby vyhovovala specifikům prostředí.

Práce obsahuje také analýzu prostředí subjektu, který se zabývá vývojem mobilních aplikací a je vhodným kandidátem k osvojení navržené metodiky. Je tedy popsán možný přínos zavedení této metodiky a následně navržen i jeho postup, včetně návodu pro konfiguraci nástrojů pro podporu řízení.

Cílem této práce je vytvoření komplexní metodiky vývoje mobilních aplikací a navržení jejího jednoduchého zavedení a řízení v tomto prostředí.

Pokud se čtenář věnuje vývoji mobilních aplikací, přináší mu tato práce teoretický i praktický návod, jak řídit tento vývoj a může na základě získaných informací implementovat metodiku i ve vlastním prostředí.

Klíčová slova

Mobilní aplikace, řízení vývoje, agilní řízení.

Abstract

The subject of this thesis is creation of development methodology suitable for mobile application environment. Base for the methodology is obtained from analysis of specifics of the environment and already existing methodologies. The closest methodology is selected based on the analysis and is then modified to suit the environment specifics.

The thesis also contains analysis of environment of subject which is engaged in mobile app development and is therefore appropriate candidate for adoption of the methodology. The possible benefits of implementation of the methodology are described. Then process of the implementation is designed, including configuration of management support tools.

Goal of this thesis is to create complex mobile application development methodology and design easy implementation and management of it.

If reader's area of interest includes mobile application development, this work provides him with theoretical and practical instructions for managing the development. Based on the acquired information he can adopt the methodology and use it in his own environment.

Keywords

Mobile application, development management, agile management.

Obsah

Úvod.....	13
Cíle práce.....	13
Použité metody	13
1 Analýza prostředí a existujících metodik	15
1.1 Specifika prostředí vývoje mobilních aplikací	15
1.1.1 Rozdíly oproti vývoji softwaru pro jiné než mobilní platformy.....	15
1.1.2 Specifika prostředí z časového pohledu	16
1.1.3 Specifika prostředí z pohledu řízení a členění týmu	16
1.2 Výčet existujících metodik pro řízení vývoje softwaru	19
1.2.1 Rigorózní	19
1.2.2 Agilní.....	22
2 Příprava pro tvorbu metodiky	30
2.1 Porovnání existujících metodik se specifiky prostředí.....	30
2.2 Očekávané vlastnosti metodiky	30
3 Výběr nejbližší metodiky	32
3.1 Ohodnocení kritérií pro výběr metodiky	32
3.2 Vyhodnocení výběru metodiky	34
3.3 Nevyhovující části vybrané metodiky	34
4 Návrh metodiky	36
4.1 Role.....	36
4.1.1 Product Owner	37
4.1.2 Agile Coach	37
4.1.3 Vývojový tým	37
4.1.4 Solution Expert.....	38
4.2 Typy úkolů a jejich workflow	39
4.2.1 Epic	40
4.2.2 Story	41
4.2.3 Implementační sub-task	43
4.2.4 Neimplementační sub-task	45
4.2.5 Task / Bug	46
4.3 Pohledy jednotlivých rolí na workflow	47
4.3.1 Product Owner a Agile Coach	47
4.3.2 Vývojový tým	48

4.3.3 Solution Expert	49
4.4 Rozdělení projektu v čase	50
4.4.1 Přepoččet na časovou náročnost	50
4.4.2 Iterace.....	51
4.4.3 Události	51
4.4.4 Projekce iterací.....	52
4.4.5 Speciální případ iterace: Hotfix	53
4.5 Dokumenty a ustanovení	54
4.5.1 Co je třeba definovat před začátkem projektu?	54
4.5.2 Report iterace	56
4.5.3 Zápisy z pravidelných událostí.....	56
4.6 Metriky.....	57
4.6.1 Výkonnost týmu.....	58
4.6.2 Komplettnost analýzy úkolů.....	59
4.6.3 Odstraňování překážek	59
4.6.4 Příčina nepřímo související s projektem	60
5 Implementace metodiky	61
5.1 Představení subjektu	61
5.1.1 Popis prostředí subjektu	61
5.1.2 Současné řešení řízení projektů.....	62
5.2 Analýza subjektu	63
5.2.1 Problémy a nedostatky současného řešení	63
5.2.2 Řešení problémů a nedostatků pomocí metodiky	64
5.2.3 Kroky zavedení metodiky	65
5.2.4 Metriky pro ověření účinnosti zavedení metodiky.....	68
5.3 Konfigurace nástrojů pro podporu řízení metodiky	68
5.3.1 Nástroje.....	68
5.3.2 Vzory dokumentů a ustanovení.....	80
5.3.3 Simulace modelových situací	86
5.4 Shrnutí	90
Závěr	91
Použitá literatura	92
Přílohy	I
Příloha A: Veřejný repozitář s digitálními zdroji práce	I
Příloha B: Manuál pro role: Product Owner.....	I

Příloha C: Manuál pro role: Agile Coach	I
Příloha D: Manuál pro role: Mobilní vývojář	II
Příloha E: Manuál pro role: Tester	III
Příloha F: Manuál pro role: Master Solution Expert	III
Příloha G: Manuál pro role: Platform Solution Expert	IV

Seznam obrázků

Obrázek 1: Grafické znázornění „toku“ mezi jednotlivými fázemi v rámci vodopádového přístupu (Half, 2014)	20
Obrázek 2: Grafická reprezentace spirálového modelu a jeho cyklů (Boehm, 1986).....	21
Obrázek 3: Ukázka podoby digitálního Kanban Boardu (Rehkopf, 2020)	25
Obrázek 4: Diagram strukturalizace úkolů	39
Obrázek 5: Workflow úkolů typu Epic.....	40
Obrázek 6: Workflow úkolů typu Story	42
Obrázek 7: Workflow úkolů typu Implementační sub-task	44
Obrázek 8: Workflow úkolů typu Neimplementační sub-task.....	45
Obrázek 9: Workflow úkolů typu Task nebo Bug.....	46
Obrázek 10: Jednoduchý příklad implementace roadmapy	53
Obrázek 11: Jednoduchý příklad projekce hotfix iterace a změn, které vyvolává v roadmapě	54
Obrázek 12: Současné workflow úkolů v Synetechu.....	62
Obrázek 13: Konfigurace typu úkolů v nástroji Jira.....	70
Obrázek 14: Vlastní položky karet úkolů vytvořené v nástroji Jira.....	70
Obrázek 15: Konfigurace schématu typů úkolů a odpovídajících karet v nástroji Jira	73
Obrázek 16: Schéma workflows pro typy úkolů v nástroji Jira	73
Obrázek 17: Konfigurace sloupců Boardu pro role Product Owner a Agile Coach v nástroji Jira.....	75
Obrázek 18: Příklad filtrů pro zjednodušení práce s Boardy v nástroji Jira	75
Obrázek 19: Příklad konfigurace „Swimlanes“ v nástroji Jira.....	76
Obrázek 20: Automatizační pravidlo pro posun Epicu do stavu <i>IN PROGRESS</i> v nástroji Jira.....	76
Obrázek 21: Automatizační pravidlo pro posun Epicu do stavu <i>DONE</i> v nástroji Jira	77
Obrázek 22: Automatizační pravidlo pro kopírování hodnoty <i>Components</i> při vytváření implementačních sub-tasků v nástroji Jira	77
Obrázek 23: Automatizační pravidlo pro změnu stavu z <i>IMPLEMENTED</i> na <i>READY FOR TESTING</i> po přidání hodnoty pro položku <i>Testing Environment</i> v nástroji Jira	78
Obrázek 24: Automatizační pravidlo pro výpočet souhrnného odhadu pracnosti pro Iteraci ve Story Pointech (počítáno pro úkoly pro platformu iOS s nízkou úrovní rizikovosti) v nástroji Jira.....	79
Obrázek 25: Ukázka projekce roadmapy v nástroji Easy Agile Roadmaps	80
Obrázek 26: Vzor projektové stránky – úvodní část a rozcestník	81
Obrázek 27: Vzor projektové stránky – ustanovení	82
Obrázek 28: Vzor projektové stránky – kontakty.....	83
Obrázek 29: Vzor reportu iterace – seznam úkolů.....	84
Obrázek 30: Vzor reportu iterace – výkonnost týmu	85
Obrázek 31: Vzor reportu iterace – nedodělky	85
Obrázek 32: Vzor zápisu ze Standupu	86
Obrázek 33: Vzor zápisu z retrospektivy	86
Obrázek 34: Modelová situace pohledu na Board pro roli Solution Expert před započítáním iterace	87

Obrázek 35: Modelová situace pohledu na Board pro mobilní vývojáře při započetí iterace	87
Obrázek 36: Modelová situace pohledu na Board pro roli Solution Expert po nalezení nedostatku sub-tasku vývojářem.....	88
Obrázek 37: Modelová situace pohledu na Board pro mobilní vývojáře po doplnění specifikace sub-tasku.....	88
Obrázek 38: Modelová situace pohledu na Board pro roli Solution Expert po dokončení implementace všech sub-tasků Story	88
Obrázek 39: Modelová situace pohledu na Board pro testera po přiřazení testovacího prostředí implementované Story.....	89
Obrázek 40: Modelová situace pohledu na Board pro mobilní vývojáře po nalezení nedostatku v implementaci Story během jejího testování	89
Obrázek 41: Modelová situace pohledu na Board pro role Product Owner a Agile Coach v průběhu iterace	90

Seznam tabulek

Tabulka 1: Praktiky XP (Extreme Programming)	26
Tabulka 2: Vícekriteriální rozhodování výběru z vyčtených agilních metodik	33
Tabulka 3: Stavy úkolů typu Epic	40
Tabulka 4: Stavy úkolů typu Story	42
Tabulka 5: Stavy úkolů typu Implementační sub-task (bez stavů „BLOCKED BY..“, které jsou vysvětleny v textu pasáže)	44
Tabulka 6: Pohled rolí Product Owner a Agile Coach na stavy jednotlivých typů úkolů	47
Tabulka 7: Pohled role Mobilní vývojář na stavy jednotlivých typů úkolů	48
Tabulka 8: Pohled role Tester na stavy jednotlivých typů úkolů	48
Tabulka 9: Pohled role UX/UI designér na stavy jednotlivých typů úkolů	49
Tabulka 10: Pohled role Backend vývojář na stavy jednotlivých typů úkolů	49
Tabulka 11: Pohled role Solution Expert na stavy jednotlivých typů úkolů	50
Tabulka 12: Formy zápisů z pravidelných událostí	56
Tabulka 13: Stručný popis kroků zavedení metodiky v prostředí analyzovaného subjektu	65
Tabulka 14: Seznam položek karet pro jednotlivé typy úkolů v nástroji Jira	71

Seznam zkratek

CI	Continuous Integration (kontinuální integrace) je praktika založená na rychlé a spolehlivé integraci nově vytvořeného kódu do hlavních vývojových větví.
CD	Continuous Delivery (kontinuální doručení) je rozšířením CI, které se stará o to, že kód v hlavních větvích je vždy nejen spustitelný, ale zároveň také připraven k nasazení.
JIT	Just In Time (právě včas) – jedná se o princip doručování produktu nebo jeho částí v rozsahu a čase požadovaném jeho poptávkou.
MADM	Pracovní název navržené metodiky. Vychází ze zkratky <i>Mobile Apps Development Methodology</i> , což v překladu znamená: metodika vývoje mobilních aplikací.
TDD	Test Driven Development (vývoj řízený testy) je přístup k vývoji, který je založen na tom, že jsou nejdříve napsány testy a teprve poté samotná implementace, která je považována za hotovou ve chvíli, kdy jsou všechny testy provedeny úspěšně.
UX	User Experience (uživatelská zkušenost) představuje oblast vývoje, která se zaměřuje na použitelnost a intuitivnost aplikace z pohledu koncového uživatele.
WIP	Work In Progress (rozpracováno) představuje označení úkolu, který je momentálně ve fázi rozpracování.

Úvod

Trh s mobilními aplikacemi neustále roste. Chytrý telefon vlastní v dnešní době již více než „2,7 miliard lidí“ (Milijic, 2020) a toto číslo stále roste. Mobilní aplikace dnes pronikají již do všech myslitelných oblastí a známé rčení „There’s an App for That“¹ (Chen, 2010), které zaznělo v reklamních spotech propagující iPhone již v roce 2009, dnes již není jen marketingovým tahákem, ale přímo realitou. Význam trhu mobilních aplikací stále roste, stejně tak jako počty vývojářů a vývojářských studií, které se zabývají právě mobilními platformami.

Ač je prostředí vývoje mobilních aplikací značně specifické, neexistuje komplexní metodika vývoje mobilních aplikací, kterou by bylo možné jednoduše zavést a řídit. Účelem této práce je tedy vytvoření takové metodiky tak, aby ji bylo možné zavést bez nutnosti analýzy, adaptace a ověřování celé škály existujících metodik a rámců pro řízení vývoje softwaru.

Cíle práce

Hlavním cílem práce je vytvoření komplexní metodiky vývoje mobilních aplikací, která je vytvořena s ohledem na specifika tohoto prostředí a je jednoduše implementovatelná.

Tohoto cíle je dosaženo pomocí následujících podcílů:

- Analýza specifik prostředí vývoje mobilních aplikací.
- Analýza stávajících metodik pro řízení vývoje softwaru.
- Definice stěžejních problémů, které by metodika měla vyřešit.
- Výběr nejbližší existující metodiky a její adaptace na dané prostředí.
- Návrh metodiky v plném rozsahu.
- Představení a analýza subjektu, pro který je navržena implementace metodiky.
- Návrh postupu implementace metodiky v prostředí představeného subjektu.

Použité metody

Práce je rozdělena na 3 hlavní části. V první části je analyzováno prostředí vývoje mobilních aplikací a jsou představeny existující metodiky pro řízení vývoje softwaru. V druhé části jsou zhodnoceny a porovnány výsledky první části a na základě tohoto zhodnocení je navržena metodika. Poslední část je věnována návrhu implementace metodiky v reálném prostředí.

Práce předpokládá určitý přehled a zkušenosti s vývojem softwaru na straně čtenáře (minimálně jako člen vývojového týmu). Hlavním zaměřením je navržení komplexní

¹ Překlad: „Na to je aplikace“. Toto rčení značí, že na všechno existuje aplikace, která to za nás vyřeší.

metodiky, která ovšem není v plném rozsahu pochopitelná, pokud si ji čtenář nedokáže sám představit v prostředí reálných procesů vývoje softwaru.

1 Analýza prostředí a existujících metodik

V této kapitole je analyzováno prostředí vývoje mobilních aplikací a jsou vyčteny existující metodiky pro řízení vývoje softwaru. Výstupy z této části jsou dále použity v následující kapitole návrhu metodiky.

1.1 Specifika prostředí vývoje mobilních aplikací

V rámci této sekce je analyzováno prostředí vývoje mobilních aplikací a jsou definována jeho specifika, která je nutno zohlednit při tvorbě metodiky řízení projektů v tomto prostředí.

1.1.1 Rozdíly oproti vývoji softwaru pro jiné než mobilní platformy

Vysoká proměnlivost

Hlavním specifikem prostředí vývoje mobilních aplikací je zejména jeho proměnlivost. Technické vybavení koncových zařízení stále roste a zvyšují se tak standardy, které by aplikace měly splňovat. Časté aktualizace operačních systémů, které v nemalém počtu případů vyžadují i změny na straně samotných aplikací, jsou v tomto prostředí také na denním pořádku.

A ta vůbec nejproměnlivější část je samotný trh mobilních aplikací. Každý den je vydáno v průměru 1 118 nových verzí aplikací² pro platformu iOS a neuvěřitelných 3 699 aplikací pro platformu Android ([42matters, 2020](#)).

Způsob nasazení a distribuce aplikací

Jednotlivé aplikace jsou umisťovány do virtuálních obchodů s aplikacemi: App Store pro platformu iOS a Google Play pro platformu Android. Mobilní platformy nejsou jediné, které poskytují distribuci softwaru přes virtuální obchody s aplikacemi, ovšem pokud jsou uvažována specifika těchto platforem, nelze tuto skutečnost opomenout. O to spíše, že platforma iOS je unikátní v tom, že v jejím prostředí je obchod s aplikacemi opravdu jediným zdrojem, odkud může koncový uživatel produkční aplikaci získat.

Společnost Apple si svoje prostředí hlídá i tím, že aplikace před uveřejněním v App Store musí projít přezkoumáním, čímž je zajištěno, že se uživatel jejich platformy nikdy nemůže dostat k nefunkční nebo závadné aplikaci. Zároveň to ovšem představuje zvýšenou zátěž na straně vývojářů a tento krok je nutno uvažovat v rámci procesu nasazení aplikace nebo její aktualizace. Ke kroku přezkoumávání aplikací před jejich publikováním do oficiálního obchodu se v poslední době ubrala také společnost Google (pro svoji platformu Android),

² Do tohoto čísla jsou započítány jak úplně nové aplikace, tak aktualizace existujících aplikací.

ovšem stále se nejedná o tak striktní revizi, jako praktikuje společnost Apple ([Toombs, 2019](#)).

Samostatnost aplikace

Není to pravidlem, ovšem ve většině případů neexistuje samotná aplikace, ale souběžně s ní je v provozu i její webová podoba. Tato skutečnost je typická např. pro segment e-commerce, ve kterém hojně vznikají mobilní aplikace sloužící jako podpora webové aplikace. V některých případech je aplikováno i propojení webu a mobilní aplikace tím, že části aplikace pouze zobrazují webové stránky místo toho, aby byly duplicitně implementovány v totožné podobě.

Rozsah zpracovatelných informací o uživateli a prostředí

V neposlední řadě je nutné zmínit, že chytré telefony poskytují značně obsáhlejší informace o uživateli a jeho prostředí, které je možné odrazit v UX aplikace. To je také jedním z důvodů, proč je UX pro mobilní aplikace velmi důležité. Jeho správné aplikování se často stává rozhodující konkurenční výhodou. Pro lepší pochopení tohoto specifika může být jednoduchým příkladem to, že se v aplikaci s jízdními řády MHD na prvním místě nabídne nejbližší zastávka na základě pozice získané z GPS senzoru.

1.1.2 Specifika prostředí z časového pohledu

Jak již bylo zmíněno v předchozí sekci, u mobilních aplikací existuje jistá prodleva mezi žádostí o publikování verze aplikace a jejím samotným publikováním, která je způsobena prováděním revize před uvedením do obchodu. Tento fakt zvyšuje důležitost ověřování kvality aplikace před jejím uvedením, jelikož pokud je nalezen bug³ v nasazené produkční verzi, tak, i pokud je vyřešen jako hotfix⁴, není možné jeho opravu okamžitě publikovat.

Je také nutné vzít v potaz to, že, opět zapříčiněním proměnlivosti prostředí, aplikace rychle zastarávají a je tedy nutné je průběžně udržovat. Nejde ovšem pouze o fakt, že by aplikace vypadala nemoderně, ale i o to, že po určité době nemusí být její aktualizace schválena k publikování do obchodu z důvodu použití zastaralých knihoven nebo nesplňování nových požadavků. Pokud je tedy aplikace neudržována a po nějaké době se objeví požadavek na přidání jednoduché funkcionality navíc, může se stát, že změna zabere i několikánásobek času kvůli ošetření dalších nedostatků, které v průběhu času vznikly.

1.1.3 Specifika prostředí z pohledu řízení a členění týmu

Z pohledu podoby týmu je nutné uvažovat dvě skutečnosti. Jednou z nich je, že pro uvedení mobilní aplikace na trh je v naprosté většině případů nutné vytvořit dvě aplikace pro obě stěžejní platformy Android a iOS. Druhá skutečnost se týká toho, že aplikace v nemalém počtu případů vznikají jako podpora existující např. webové aplikace, a tudíž aplikace

³ Chyba nebo nedostatek v aplikaci.

⁴ Rychlá oprava např. kritického bugu, který je nalezen v nasazené produkční verzi aplikace.

nevzniká „na zelené louce“ a je zasazována do již existující infrastruktury. Obě skutečnosti jsou více rozebrány v následujících oddílech.

Přístup k vývoji na obě stěžejní platformy

Existuje několik způsobů, jak přistoupit k vytváření mobilní aplikace, která bude použitelná na obou stěžejních platformách.

Nejspolehlivějším řešením je samostatný nativní vývoj pro každou platformu zvlášť. Toto řešení zahrnuje určitou duplicitu, ovšem to za cenu vyšší stability, větších možností, které jednotlivé platformy nabízí, a také přirozenějšího pocitu z aplikace pro uživatele.

Protikladem je multiplatformní vývoj za pomoci nástrojů jako je Flutter, React Native nebo Unity (používané zejména pro hry). To umožňuje vytvoření aplikace pro obě platformy s použitím jedné codebase⁵. Nevýhodou tohoto přístupu je, že jsou obě aplikace v podstatě totožné, což nemusí být žádoucí stav, jelikož i jednotlivé platformy mají svá specifika a jejich uživatelé očekávají v určitých případech rozdílné chování. Obecně je tedy tento přístup doprovázen snížením UX hodnoty aplikace. Zároveň je také tímto přístupem směřováno k tomu, že je aplikace vytvořena tak, aby ji bylo schopno provozovat nejslabší možné zařízení, na což mohou doplatit uživatelé druhé platformy. Jedním z těchto případů může být nucené omezení funkcionalit, které by jinak obecně výkonnější zařízení platformy iOS zpracovala bez problémů.

Posledním přístupem je hybridní řešení, které je založené pouze na částečném sdílení zdrojového kódu. Tento přístup je momentálně možné použít buď kombinováním předešlých přístupů nebo pomocí nově vznikajících nástrojů jako je např. Kotlin Multiplatform. Tento přístup eliminuje nevýhody předešlých dvou a umožňuje sdílení částí kódu, který je funkcionalitou totožný (typicky byznys logika a zpracování dat), se zachováním nativní implementace uživatelského rozhraní, a tudíž se zachováním maximálního možného UX, a s otevřenými možnostmi použít vše, co jednotlivé platformy nabízí. Znamená to ovšem také zvýšenou zátěž na řízení.

Rozsah oblasti vývoje

Pro rozsah oblasti vývoje je stěžejní, jestli služba poskytovaná aplikací již existuje a je pouze rozšiřována na další platformy, nebo jestli se jedná o vývoj úplně nové služby. V případě rozšiřování je nutné počítat s tím, že s největší pravděpodobností již existuje určitá infrastruktura, což znamená, že při vývoji aplikace není uvažován např. vývoj webového backendu⁶ nebo designový návrh, který může vycházet z již existujícího řešení.

Zahrnutí designového návrhu do vývoje aplikace je sporné zejména při zakázkovém vývoji, kdy je předmětem zakázky pouze aplikace jako podpora ke stávajícímu řešení a o stránku

⁵ Souhrn zdrojového kódu, který je potřeba k sestavení určitého softwaru nebo jeho částí.

⁶ Značí architektonickou vrstvu, která operuje s daty (typicky pracuje s databázemi a vystavuje rozhraní, čímž poskytuje zpracovaná data dalším vrstvám). V kontextu mobilních aplikací se jedná o implementaci vzdáleného poskytovatele dat aplikace.

vzhledu a UX by se měl tedy starat spíše tým, který měl na starosti původní řešení. Toto samozřejmě není pravidlem, ale je to často užívaný přístup.

1.2 Výčet existujících metodik pro řízení vývoje softwaru

S rozvojem komplexnosti informačních technologií se zvyšovala potřeba metodického řízení vývoje softwaru od jeho návrhu, přes implementaci a testování až po jeho nasazování a následnou údržbu. Řízení vývoje si již prošlo značnou evolucí a současné metodiky se značnou měrou liší od těch prvních, které vznikaly v druhé polovině minulého století. Tím nejzásadnějším posunem v myšlení se stalo uvědomění, že vývoj softwaru nelze do detailu popsat před jeho samotným počítím. Tento posun také rozděluje existující metodiky na dvě hlavní kategorie: **rigorózní** a **agilní**.

V tomto oddílu jsou vyčteny existující metodiky v obou hlavních kategoriích a nastíněny všechny možnosti, dle kterých lze k řízení vývoje softwaru přistoupit.

1.2.1 Rigorózní

Rigorózní nebo také tradiční přístup k řízení vývoje softwaru je z obou přístupů tím starším. Základní myšlenkou, o kterou se opírá, je, že vývoj lze předem popsat a naplánovat v jeho plném rozsahu. Přístup neumožňuje změny v průběhu vývoje (nebo pouze ve velmi malém množství) a striktně dodržuje předem plánované scénáře.

I přesto, že ho lze, vzhledem k výše zmíněnému posunu v myšlení na to, že vývoj softwaru nelze předem popsat, považovat za zastaralý, stále může být vhodným řešením pro určité typy projektů. Mezi takové projekty je možné zařadit např. software velmi malého rozsahu a velmi specifického účelu nebo vývoj identického existujícího softwaru pro jinou podobnou platformu. Samozřejmě nelze říct, že pro takové projekty je vždy nejvhodnějším řešením, nicméně v určitých situacích se může stát první volbou.

Metodiky řazené do kategorie rigorózního přístupu si prošly určitou evolucí a některé z nich již nestaví takový odpor ke změnám nebo třeba rozdělují celý vývoj na iterace, což je již považováno za jedno ze specifik agilního přístupu. V dalších částech jsou popsány nejznámější metodiky, resp. modely životního cyklu, spadající do kategorie rigorózních.

Vodopádový model

Tzv. „waterfall“ je nejstarším modelem životního cyklu vývoje softwaru a také nejznámějším rigorózním modelem (lze ho také označit za ryze rigorózní). Je založen na sekvenčním přístupu, kdy na sebe jednotlivé fáze vývoje přímo navazují a práce jimi tedy „protéká“ směrem dolů, stejně jako u vodopádu.



Obrázek 1: Grafické znázornění „toku“ mezi jednotlivými fázemi v rámci vodopádového přístupu⁷ (Half, 2014)

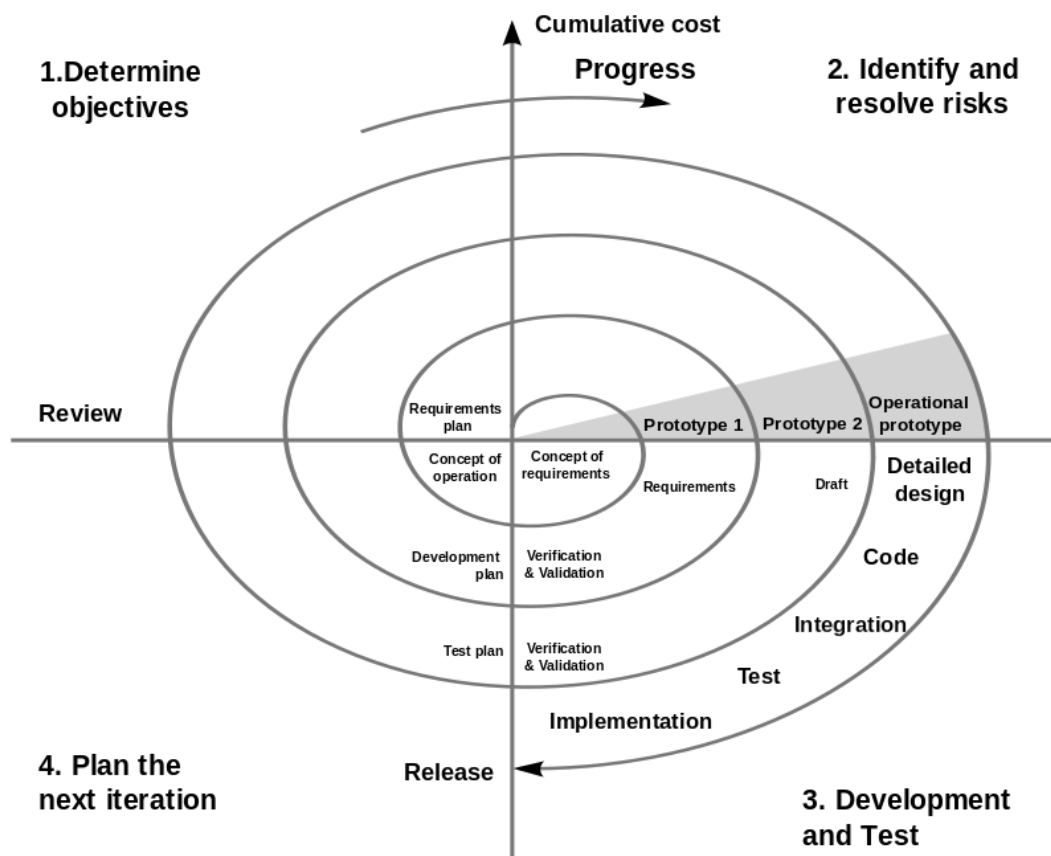
Za autora první formální zmínky o tomto modelu je považován Winston W. Royce, který článek s popisem modelu publikoval v roce 1970 (ovšem jednalo se pouze o popis, název „vodopád“ zmíněn nebyl). Zajímavostí je, že model byl v článku označen jako vadný a nefunkční, ale i přesto se stal na několik dalších dekád hojně využívaným. (Wikipedia, 2020)

Jak lze vytušit z popisu nebo přímo z grafického znázornění na obrázku 1, změna v průběhu vývoje v podstatě není možná. I malá změna by znamenala kompletní přeplánování projektu a bylo by nutné opět „vystoupat na vrchol vodopádu“.

Spirálový model

Z vodopádového modelu vychází spirálový model, který je mimo něj inspirován také prototypovým přístupem. Pomocí tohoto přístupu lze řešit problém nereálnosti vytvoření přesné a detailní specifikace všech požadavků před začátkem vývoje. Celý projekt je rozdělen do cyklů a během každého cyklu se opakují fáze analýzy, hodnocení, vývoje a plánování dalšího cyklu. Tento přístup již umožňuje méně nákladné změny v průběhu projektu.

⁷ Překlad: Requirements = Požadavky; Design = Návrh; Implementation = Implementace; Verification = Ověření; Maintenance = Údržba



Obrázek 2: Grafická reprezentace spirálového modelu a jeho cyklů⁸ (Boehm, 1986)

Na konci každého cyklu vzniká prototyp softwaru, na základě kterého lze přehodnotit další postup a kompletizovat původní specifikaci požadavků. Takto lze ve velké míře snížit rizika vývoje. Zároveň to umožňuje zapojení klienta v průběhu vývoje, kdy je mu poskytnuta možnost „osahat“ prototyp výsledného softwaru a případně ho připomínkovat. (Wikipedia, 2020)

UP (Unified process)

UP je rámec, který, na rozdíl od spirálového modelu, vychází z inkrementálního přístupu. Ten se od prototypového liší zejména tím, že předem rozděluje celý projekt na části, které postupně plní. V nejjednodušší formě si lze základní inkrementální přístup představit jako sérii po sobě jdoucích vodopádových modelů.

Rámec rozděluje projekt na 4 fáze: **zahájení**, **příprava**, **konstrukce** a **předání**, které jsou následně děleny na jednotlivé časově ohraničené iterace. Fáze zahájení je tou nejkratší a slouží k úvodní specifikaci projektu. V následné fázi přípravy jsou identifikována rizika projektu a je navržena architektura systému. Jednou ze základních praktik UP je použití

⁸ Překlad: Cumulative cost = Kumulativní náklady; Progress = Pokrok; Determine objectives = Určení cílů; Identify and resolve risks = Identifikace a vyřešení rizik; Development and test = Vývoj a testování; Plan the next iteration = Plánování další iterace

UML⁹ a v této fázi jsou tedy také vytvořeny diagramy, které UML definuje. Jakmile je příprava hotova, přechází se k nejobsáhlejší fázi konstrukce, v rámci které je tvořena samotná implementace. Projekt končí fází předání, která zahrnuje nasazení softwaru do produkčního prostředí a případné opravy a úpravy na základě zpětné vazby od uživatelů.

UP je obecně spíše rámcem určeným k rozšíření a adaptování na konkrétní prostředí. I proto vzniklo několik jeho variací. Mezi nejznámější variace patří RUP (Rational Unified Process), vlastněný společností IBM, nebo OpenUP, která je jednodušší open source¹⁰ formou RUP. UP se dokonce dočkalo i agilní variace AUP. ([Wikipedia, 2020](#))

1.2.2 Agilní

Agilní přístup je v mnohých ohledech naprostým opakem rigorózního přístupu. Změny v průběhu vývoje, maximální zapojení klienta a zaměření na přidanou hodnotu jsou základními stavebními kameny všech agilních metodik. Lidé jsou bráni jako klíčový faktor úspěchu a agilní metodiky tedy upřednostňují týmovou práci a schopnosti jedinců před striktně dodržovanými procesy. I proto jsou tyto metodiky označovány jako „lehké“.

Lean myšlení je podstatou agilního přístupu a značí soustředění na přidanou hodnotu a eliminaci „odpadu“. Přístup je zaměřen na to, aby byla dodána maximální hodnota, která je zpravidla skryta v nevelké části celého rozsahu projektu. Na tuto myšlenku lze skvěle aplikovat Paretovo pravidlo 80/20 – 80% hodnoty je skryto ve 20% rozsahu.

Agilní týmy jsou stavěny jako multifunkční a měly by zahrnovat všechny role a profese potřebné pro celý průběh vývoje. Týmy jsou tedy soběstačné a mohou do důsledku táhnout za jeden provaz (lze eliminovat situaci, kdy se přehazuje vina mezi jednotlivými týmy různých zaměření).

Nedílnou součástí agilních metodik je také kontinuální zlepšování. Krátké iterace umožňují nejen přehodnocení samotného produktu, ale i procesů a technik používaných pro vývoj. V rámci týmů tak lze zefektivnit třeba procesy předávání jednotlivých částí mezi členy týmu (např. předání návrhu k implementaci nebo hotové implementace k testování). Právě kontinuální zlepšování a multifunkční týmy umožňují velmi značné zvýšení celkové efektivity vývoje.

V současné době existuje nemalé množství agilních metodik a rámců. V dalších částech jsou představeny ty nejvíce používané.

Scrum

Scrum je vůbec nejznámějším a nejpoužívanějším agilním rámcem, dokonce je někdy jeho jménem mylně pojmenováván celý agilní přístup k řízení projektů. Je založen na krátkých časově ohraničených iteracích. Nejedná se o metodiku, ale pouze o rámec, který by měl být

⁹ Grafický jazyk pro specifikaci a návrh systémů.

¹⁰ Přístup známý zejména z prostředí vývoje softwaru, který značí, že jsou všechny zdroje veřejné a mohou být opravovány a vylepšovány komunitou.

adaptován na konkrétní prostředí. V dnešní době nalézají uplatnění nejen ve světě vývoje softwaru.

Hlavními hodnotami Scrumu, které by měly být ztělesněné v týmu, jsou: *závazek, kuráž, soustředění, otevřenost a respekt* (Sutherland, 2017). Tyto hodnoty obecně znamenají to, že celý tým táhne za jeden provaz a snaží se společně o dosažení cílů.

Důležitou součástí Scrumu je tzv. definice „Done“ („Hotovo“). Tato definice by měla být sdílena všemi členy týmu a měla by popisovat, co znamená, že je nějaký úkol dokončen (např. úkol musí být implementován, otestován a integrován). Tím je zajištěna konzistence a také je jasné a transparentní, co se od každého člena týmu očekává.

Scrum definuje určité **role, události a artefakty**.

Mezi **role** patří Product Owner a Scrum Master (pro řízení projektu a týmu) a vývojový tým, který obecně zastřešuje všechny členy, kteří se přímo podílí na samotné tvorbě produktu. Všechny tyto role dohromady tvoří tzv. „Scrum tým“.

Product Owner (v překladu vlastník produktu) je role, která je zodpovědná za maximalizování hodnoty produktu. Vytváří a řadí úkoly v tzv. Product Backlogu (vysvětleno níže) a určuje tím postup vývoje produktu.

Scrum Master má za úkol řídit vývojový tým a pomáhat mu. Měl by nejlépe rozumět teorii a praktikám Scrumu a měl by hlídat jejich dodržování v rámci celého Scrum týmu. Jeho úloha není pouze vést tým, ale také mu pomáhat zbavovat se překážek, na které tým v průběhu vývoje narazí.

Vývojový tým je skupina profesionálů, kteří se starají o samotnou tvorbu produktu. Měl by zahrnovat všechny role a profese potřebné k dokončení úkolů. Scrum nepřipouští žádné „podtýmy“ (jako např. vývojáři nebo testéři) a všichni jsou tedy na stejné lodi.

Události jsou vyčteny zejména „pro vytvoření pravidelnosti a k minimalizování potřeby schůzí, které nejsou definované Scrumem“ (Sutherland, 2017). Vynecháním některé z těchto událostí může dojít ke snížení transparentnosti, která je podstatnou součástí Scrumu.

Základní událostí je *Sprint*, který představuje jednu iteraci vývoje. Jedná se o opakující se, časově ohraničené období, na konci kterého vzniká potencionálně doručitelný inkrement produktu. Délka Sprintu je předem definována a měla by trvat mezi jedním a čtyřmi týdny.

Každému sprintu musí předcházet *Sprint Planning* neboli plánování Sprintu. Vstupem této události je projektovaná kapacita týmu na další Sprint. V rámci plánování se vybere množina úkolů, kterou je možné dokončit v rámci kapacity týmu. Dokončení této množiny úkolů se stává cílem následujícího Sprintu a tým se zavazuje k naplnění tohoto cíle. Samotná schůzka se skládá ze dvou částí:

- *Co?* – vyberou se úkoly, které budou dokončeny v následujícím Sprintu
- *Jak?* – diskutuje se, jakým způsobem budou tyto úkoly dokončeny

Celý Sprint Planning by neměl být delší než 8 hodin pro 4týdenní Sprint. Součástí plánování je typicky i odhad pracnosti jednotlivých úkolů. Na to jsou používány tzv. *Story Points*, které představují jednotku práce. Na jejich základě je poté možné měřit celkovou výkonnost týmu, která je reprezentována počtem Story Points, které tým zvládne dokončit během Sprintu.

V průběhu Sprintu by se každý den měl konat *Daily Scrum*, jinak označovaný také jako Standup. Jedná se o krátkou (maximálně 15 minut) a pravidelnou schůzku, na které se synchronizují jednotliví členové týmu. Schůzka může mít různé podoby, ovšem tradičně probíhá tak, že postupně každý odpoví na následující otázky:

- Co jsi dělal včera pro dosažení cíle Sprintu?
- Co budeš dělat dnes pro dosažení cíle Sprintu?
- Existují nějaké překážky, které brání dosažení cíle Sprintu?

Na konci každého Sprintu by se mělo konat *Sprint Review* (shrnutí Sprintu). Této schůzky se účastní nejen Scrum tým, ale i stakeholdéři. Je během ní shrnuto, co bylo dokončeno během Sprintu (ideálně i s demo ukázkou), a je diskutováno nad tím, co by mělo být zpracováno dále tak, aby byla optimalizována hodnota produktu.

Po Sprint Review (ale zároveň před dalším plánováním) by měla následovat *Sprint Retrospective* (retrospektiva), která představuje prostor, v kterém může tým zhodnotit pozitiva a negativa událostí a situací, které nastaly během Sprintu a případně vznést námět na další zlepšení. Zlepšení může samozřejmě být zavedeno kdykoliv, ovšem tato schůzka poskytuje formální možnost k tomu se zastavit a zaměřit se na věci, které by mohly být vykonávány efektivněji.

V neposlední řadě jsou Scrumem popsány **artefakty**. Jejich definováním a udržováním je zajištěna transparentnost.

Prvním artefaktem je *Product Backlog*, což představuje řazený seznam úkolů, které vycházejí ze specifikace produktu. Jak již bylo zmíněno výše, o Product Backlog se stará Product Owner. Řazení úkolů by mělo odpovídat jejich přidání hodnotě a tedy předpokládaným pořadím jejich dokončení. Znamená to také, že typicky jsou úkoly na vrchních pozicích Product Backlogu popsány detailněji než ty na jeho konci.

V rámci plánování Sprintu jsou úkoly z Product Backlogu přesouvány do *Sprint Backlog*. Ten tedy představuje množinu úkolů, které by měly být dokončeny v rámci Sprintu. Důležité je, že by po dobu Sprintu měl zůstat neměnný. Pouze vývojový tým může odsouhlasit jeho změnu během Sprintu.

Posledním artefaktem je *Increment*, což představuje inkrement produktu, který vzniká na konci každého Sprintu a je potenciálně doručitelný. ([Sutherland, 2017](#))

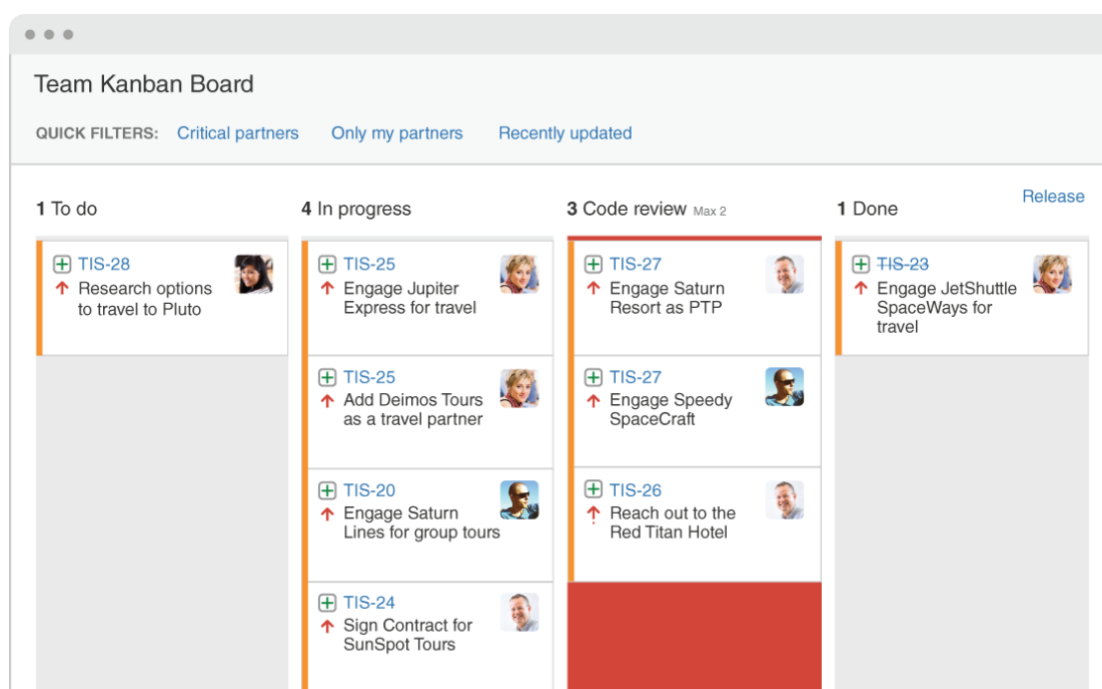
Kanban

Kanban, jako metodika řízení práce, vznikl již ve 40. letech minulého století v Toyotě. Ta se ve snaze o optimalizaci svých procesů inspirovala modelem doplňování zboží v obchodech. Obchody plní své regály takovým množstvím zboží, aby přesně pokryly poptávku zákazníků.

Stejný JIT (just in time – právě včas) princip implementovala Toyota do svých výrobních procesů a započala tak vznik Kanbanu.

Dnešní podoba Kanbanu ve spektru vývoje softwaru je stále založena na stejném principu. JIT princip je odražen ve WIP (work in progress – rozpracováno) limitech, které zaručují, že je každý maximálně soustředěn na svou činnost a odbavování jednotlivých úkolů je plynulé.

Základem rámce je tzv. *Kanban board*, který zajišťuje vizualizaci práce a jejího toku. Tento board může být fyzický, ovšem dnes již existuje nespočet nástrojů, které lze použít pro vytvoření jeho digitální podoby. Díky vizualizaci zajišťuje Kanban board také standardizaci workflow¹¹ a snazší identifikování potencionálních překážek nebo závislostí, které je nutné vyřešit.



Obrázek 3: Ukázka podoby digitálního Kanban Boardu (Rehkopf, 2020)

Kanban board obsahuje tzv. *Kanban cards*, které reprezentují pracovní úkoly a které jsou posouvány dle jejich aktuálních stavů v horizontálním směru po boardu. Tyto karty obsahují kritické informace o konkrétním úkolu, jako např. jeho popis, akceptační kritéria, osobu, které je přidělena zodpovědnost za dokončení úkolu, pracnost nebo hodnotu úkolu, a případně zdroje potřebné k jeho dokončení.

Jak již bylo zmíněno, Kanban je založen na kontinuálním toku a projekt tedy není dělen do iterací. V praxi to funguje tak, že po dokončení jednoho úkolu si pracovník vezme další úkol z vrchu backlogu (v tom případě se jedná o první sloupec „To Do“) a pokračuje dále. Není tedy třeba řešit uzavřenost Sprintu, ale backlog je možné přerazovat kdykoliv, čímž je

¹¹ Tok práce – jedná se v podstatě o stavový diagram, který odráží jednotlivé fáze pracovního úkolu.

zajištěno, že každý další úkol, který se začne zpracovávat má v momentální situaci nejvyšší hodnotu.

Na rozdíl od Scrumu není možné měřit výkonnost týmu dle počtu jednotek práce dokončených během Sprintu. Kanban místo toho definuje tzv. *Cycle time*, který reprezentuje čas, který zabere dokončení jedné jednotky práce.

Jelikož je v rámci kontinuálního toku nutné také časté doručování malých inkrementů produktu, bere si Kanban na pomoc techniku CD (continuous delivery – kontinuální doručení), která nejenže zajišťuje, že je zdrojový kód vždy spustitelný, ale pomáhá také automatizaci doručování sestaveného softwaru, čímž je možné rapidně zvýšit frekvenci doručování. ([Rehkopf, 2020](#))

XP (Extreme Programming)

Extrémní programování je rámec, který je, na rozdíl od ostatních zmíněných, přímo určen pro řízení vývoje softwaru a obsahuje techniky, které jsou méně abstraktnější a aplikovatelné pouze na prostředí vývoje. Celý rámec je založen na aplikování 5 hodnot a 13 praktik, zbylé části jsou poněkud volněji specifikované.

Projekt je dělen na cykly vydání, které jsou dále děleny na týdenní cykly. Na začátku každého týdne se sejde celý tým, včetně klienta, a rozhodují, které úkoly by měly být dokončeny během nastávajícího týdne. Na konci týdne se opět sejdou, aby zhodnotili postup a rozhodli o dalším pokračování.

Zajímavou technikou, kterou XP využívá je vytváření tzv. *Spiků* v případě, že existuje úkol, který není možné časově odhadnout kvůli nedostatečnému množství informací o technické stránce implementace řešení. Může tak pomoci například při implementaci nové technologie, se kterou se řešitelé dosud nesetkali. Spike je pouze speciálním typem úkolu, jehož obsahem je vyzkoumat určitou oblast a navrhnout možná řešení problému včetně jejich hrubých odhadů pracnosti. Jakmile je Spike dokončen, může se opět přistoupit k původnímu úkolu, odhadnout jeho pracnost a zařadit ho do dalšího cyklu.

Základními hodnotami XP jsou *komunikace*, *jednoduchost* (podporuje lean myšlení), *zpětná vazba* (konstantní a frekventovaná), *kuráž* a *respekt*.

Praktiky XP se od původních 12 z první verze rámce částečně liší. XP popisuje tyto praktiky jako vzájemně provázané a dohromady tvořící celý rámec. Jednotlivé praktiky je možné použít i izolovaně, ovšem nelze pak dosáhnout maximální úrovně rámce XP. Všechny praktiky jsou popsány v [tabulce 1](#).

Tabulka 1: Praktiky XP (Extreme Programming)

Praktika	Popis
Sednout si spolu	Komunikace je jednou ze základních hodnot XP a konverzace „face to face“ je tou nejefektivnější formou, proto je dobrou praktikou nechat si tým sednout spolu do jedné místnosti a diskutovat.

Celý tým	Tato praktika odráží to, že týmy by měly být multifunkční a měly by zahrnovat všechny role a profese potřebné k dokončení úkolů.
Informativní pracovní prostor	Všechny informace musí být v rámci týmu správně a dostatečně komunikovány pro zajištění transparentnosti.
Energická práce	Jedinec je nejvíce efektivní, pokud je maximálně soustředěn a není vyrušován. U programování toto platí dvojnásob. Mělo by být zajištěno, že pracovníkovi je umožněno se do takového stavu dostat a že zároveň není přetěžován tak, aby se do stavu soustředění dostat nemohl.
Párové programování	Tato praktika znamená, že je implementace řešení úkolu vykonávána dvěma lidmi sedícími za jedním strojem. Tím je zajištěno, že je kód průběžně revidován a riziko, že se vývojář „zasekne“ na řešení určitého problému je minimalizováno. Je ověřeno, že aplikování této praktiky neznamená dvojnásobný nárůst času stráveného implementací úkolu, jelikož dva lidé dokážou dojít ke správnému řešení problému dříve.
Stories (překlad: příběhy)	Jednotlivé implementační úkoly by měly být popisovány formou „příběhů“ uživatele. Typickým popisem je např. „Jako uživatel chci, aby se po akci A ukázala obrazovka B“. Touto formou je snazší komunikovat jednotlivé funkce systému s netechnicky zaměřeným člověkem.
Týdenní cyklus	Popsáno výše, jedná se o rozdělení vývoje na týdenní iterace. Na začátku týdne se koná plánování, na konci týdne zhodnocení.
Čtvrtletní cyklus	Je synonymem k cyklu vydání. XP nespecifikuje pevnou délku na čtvrtletí, jen tímto naznačuje jeho větší objem oproti týdenním cyklům.
Slack	Slack v rámci XP znamená to, že jsou do jednotlivých cyklů přidávány úkoly s nízkou prioritou, které mohou být z cyklu vyřazeny, pokud nastane skluz na úkolech s vyšší prioritou.
Desetiminutové sestavení	Cílem je sestavit celý systém a spustit všechny testy během 10 minut. Pokud by tento proces trval delší dobu, stává se méně pravděpodobné, že bude probíhat pravidelně.
CI	Continuous Integration (kontinuální integrace) je technika, která zajišťuje, že je zdrojový kód automaticky sestaven a otestován před jeho sloučením s hlavními vývojovými větvemi. Tento přístup značně zvyšuje výslednou kvalitu produktu a zajišťuje jeho konstantní funkčnost.
TDD	Test Driven Development (vývoj řízený testy) je technika, která je založená na tom, že jsou nejdříve napsány testy a následně samotná implementace. Kompletní dokončení implementace je pak zajištěno tím, že jsou všechny testy úspěšně splněny.

XP samotné nespecifikuje žádné role, ovšem většinou je rámec asociován s rolemi **zákazník, vývojář, stopař a kouč**.

Zákazník je zodpovědný za obchodní rozhodnutí týkající se projektu. Mezi jeho úkoly patří:

- specifikovat, co by měl produkt umět,
- definovat akceptační kritéria,
- hlídat rozpočet a
- řadit jednotlivé úkoly dle priorit a jejich přidané hodnoty.

Vývojář je obecně člen vývojového týmu, tak jak je znám z prostředí Scrumu. Nezáleží na jeho konkrétní roli v týmu nebo profesi, ale jedná se obecně o osobu, která je schopna pomoci ve snahách o dokončení jednotlivých úkolů.

Stopař není povinná role a většinou ji zastupuje jeden z vývojářů. Tato role má za úkol sledovat relevantní metriky a ukazatele a hledat prostory pro zlepšení týmu.

Kouč má funkci poradce a měl by pomáhat týmu správně aplikovat techniky XP. Tato role by se dala připodobnit ke Scrum Masterovi ve Scrumu, ovšem v tomto případě kouč nemá na starosti řízení týmu, ale představuje pouze konzultanta pro správné aplikování rámce. (Agile Alliance, 2020)

Scrumban

Scrumban, jak jeho název značí, je evolucí Scrumu, která je obohacena o techniky Kanbanu. Je značně detailnější než Scrum, ovšem stále se jedná pouze o rámec, nikoli preskriptivní metodiku.

Co se týče rozdělení v čase, Scrumban se inspirované Kanbanovým kontinuálním tokem a nedefinuje tedy žádné dělení na iterace. Odůvodňuje to větší možností flexibility. Ostatní události Scrumu zachovává, ovšem nejsou vztaženy ke Sprintu, ale jsou vyvolány, když je potřeba (nepočítaje Daily Scrum, který se také koná každý den).

Pro rozdělení projektu na úkoly využívá Stories, tak jak jsou popsány v rámci XP. Velmi specifické ovšem je to, že navrhuje jejich jakékoli odhady pracnosti a předpokládá, že jsou všechny Stories v průměru stejně rozsáhlé. Odhadování pracnosti tedy považuje za ztrátu času.

Scrumban Board je typicky rozsáhlejší a definuje více stavů pro jednotlivé úkoly. Je založen na tzv. „pull systému“, který značí, že by úkoly neměly být tlačeny směrem ke konci workflow, ale naopak by měly být „tahány“ z opačného směru. Názorným příkladem může být to, že vývojář po implementaci úkol nepřesune do stavu „Testování“, ale do stavu

„Připraveno k testování“, odkud si ho do stavu „Testování“ přesune sám tester ve chvíli, kdy na něm začne pracovat.

Board také obsahuje WIP limity (stejně jako u Kanbanu), které zajišťují plynulý tok úkolů skrze workflow.

Hlavní metrikou užívanou ve Scrumbanu je *Lead Time*, který značí dobu mezi specifikováním úkolu, resp. Story, zákazníkem a schválením jeho dokončení od zákazníka. Nejedná se o celkový čas strávený nad řešením úkolu, ale o časový úsek mezi identifikováním a dokončením úkolu. Pro zlepšení přesnosti predikcí nepočítá průměrný Lead time ze všech Stories, ale rozděluje je na vysoce, středně a nízkorizikové a počítá průměrný lead time pro každou z těchto kategorií.

Scrumban obecně definuje značné množství metrik a podporuje tedy rozsáhlejší analýzy ukazatelů a metrik. Lead time lze zredukovat na *Service time* (čas služby), který končí ihned po dokončení úkolu (a tedy před schválením klientem). Service time lze dále rozdělit na:

- *Acceptance time* (čas akceptace) – čas mezi zadáním úkolu klientem a jeho vytvořením (a specifikováním) v backlogu
- *Queue time* (čas ve frontě) – čas mezi vytvořením úkolu v backlogu a počtím jeho zpracování
- *Flow time* (čas toku) – čas mezi počtím zpracování úkolu a jeho dokončením
- *Delivery time* (čas doručení) – čas mezi dokončením úkolu a jeho předáním klientovi

Všechny tyto metriky značí, že se Scrumban nesoustředí pouze na optimalizaci samotného vývoje, ale všech podpůrných procesů, které projekt zahrnuje.

Scrumban nedefinuje konkrétní role, ale pouze předkládá, že je třeba tým a k tomu všechny role, které jsou potřeba. V hodně ohledech nechává volný prostor k adaptaci a doporučuje rozšíření o praktiky Scrumu dle potřeby konkrétního prostředí. (Reddy, 2016)

Shrnutí

Nutno podotknout, že je běžnou praktikou používat hybridní řešení inspirované několika metodikami. Typickým a dost často užívaným příkladem je použití Scrumu, který je rozšířen o Kanban board a o některé praktiky z XP.

2 Příprava pro tvorbu metodiky

V této kapitole jsou zhodnoceny výstupy z první části a na jejich základě jsou definovány očekávané vlastnosti navrhované metodiky.

2.1 Porovnání existujících metodik se specifiky prostředí

Jako první a hlavní specifikum prostředí vývoje mobilních aplikací byla zmíněna vysoká proměnlivost. To samo o sobě eliminuje jakékoliv metodiky z kategorie rigorózních. Pořád mohou existovat případy, kdy by předem kompletně naplánované řešení mohlo být vhodné, nicméně obecně nelze považovat rigorózní přístup za adekvátní v kontextu tohoto prostředí.

Kvůli zmíněné proměnlivosti prostředí a poněkud složitějšímu procesu nasazování nových verzí je potřeba také, aby byla reakční doba snížena na absolutní minimum. Nutno podotknout, že i týdenní Sprint bez možnosti přerušení může být určitou překážkou. Z tohoto pohledu by se tedy jevila jako vhodná možnost přistoupit ke kontinuálnímu toku.

Na druhou stranu, byla naznačena také potřeba rychlé zpětnovazební smyčky, která by značila spíše potřebu ohraničených iterací, jelikož validace každého úkolu zvlášť se může stát velmi neefektivní a výsledně kontraproduktivní. Ohraničená iterace může být také vhodná, pokud je vzato v úvahu, že je potřeba synchronizovat vývoj na obou platformách, který se může v čase rozcházet. Ohraničení by mohlo pomoci s lepším plánováním kapacit s ohledem na postup pro každou z platforem.

Jedním ze specifik bylo také to, že v nemalém množství mobilní aplikace nevznikají „na zelené louce“, ale je nutno je implementovat v rámci existujících infrastruktur či designových návrhů. To naznačuje, že je potřeba klást větší důraz na analýzu jednotlivých úkolů před jejich samotnou implementací. Toto tvrzení také podporuje fakt, že aplikace musí projít důkladnou revizí před jejím samotným nasazením a je tudíž nutné analyzovat, zda navržené řešení je možné aplikovat, či zda nemusí být podloženo ošetřením dalších okolností, opět ideálně před samotnou implementací.

Je jasné, že metodika pro vývoj mobilních aplikací by měla být agilní. Ovšem žádná z uvedených agilních metodik neodpovídá výše zmíněným specifikům v jejich plném rozsahu. V následujících oddílech je tedy pokračováno v hledání metodiky či rámce, který je nejvhodnějším vstupním bodem k tvorbě celé metodiky.

2.2 Očekávané vlastnosti metodiky

Před výběrem je důležité shrnout, co je vlastně od výsledné metodiky očekáváno vzhledem ke specifikům prostředí:

- Možnost velmi rychlé reakce na změny
- Možnost rychlé zpětnovazební smyčky, která je vykonávána na smysluplném množství změn
- Možnost synchronizování postupu vývoje pro obě platformy
- Možnost důkladnější analýzy úkolů před jejich samotnou implementací

Nutno podotknout, že výsledná metodika by měla zahrnovat také metriky pro zvýšení možností analýzy a přehodnocení přístupu, ovšem, vzhledem k tomu, že takové metriky lze nastavit nad jakoukoli metodiku, není tato vlastnost považována za kritérium výběru.

3 Výběr nejbližší metodiky

Cílem této kapitoly je najít nejvhodnější metodiku pro vývoj mobilních aplikací z těch, které byly vyčteny v první kapitole. Výstupem je zároveň i nalezení částí vybrané metodiky, které nejsou vhodné nebo si žádají úpravu pro použití v prostředí vývoje mobilních aplikací.

3.1 Ohodnocení kritérií pro výběr metodiky

V předchozí kapitole byly prakticky vyřazeny všechny možné existující metodiky. Ovšem s takovou skutečností bylo předem počítáno. V rámci tohoto oddílu je nalezena metodika, která je potřebnému řešení nejbližší, či je na toto řešení nejlépe adaptovatelná.

Pro výběr nejbližší metodiky je použita metoda vícekritériálního rozhodování. Na základě předchozí analýzy jsou hodnoceny tyto vlastnosti: **rychlá reakce na změny**, **zpětnovazební smyčka** (rychlá, ale na smysluplném množství změn), **synchronizace vývoje obou platforem** a **důkladnější analýza úkolů před implementací**. Každá z vlastností je hodnocena dle dvou kritérií: **míra naplnění** existující metodikou (dále jen MN) a **náročnost změny** existující metodiky (dále jen NZ).

Jednotlivé vlastnosti mají přiřazeny váhy na stupnici 1-3, které značí jejich důležitost (čím vyšší váha, tím je vlastnost důležitější). Nejvyšší váhy mají vlastnosti „rychlá reakce na změny“ a „synchronizace vývoje obou platforem“, jelikož jsou naprosto nezbytné pro fungování metodiky. Správná podoba „zpětnovazební smyčky“ je neméně důležitá, ovšem vzhledem k tomu, že agilní metodiky již z principu vyžadují krátký průběh této smyčky, nelze ji považovat za rozhodující faktor výběru. Nejnižší vahou je ohodnocena vlastnost „důkladnější analýza úkolů před implementací“, jelikož ji nelze považovat za nezbytnou a jedná se spíše o optimalizaci metodiky více než základ, bez kterého metodika nemůže vzniknout.

Obě kritéria mají stejnou váhu a jsou hodnocena na stupnici 1-5, podobně jako ve škole (1 značí nejlepší hodnocení, 5 značí nejhorší hodnocení). Hodnoty v kontextu konkrétních kritérií jsou popsány níže.

Pro míru naplnění (MN) značí:

- hodnota 1, že je vlastnost metodikou v plné míře naplněna.
- hodnota 5, že vlastnost není metodikou naplněna vůbec.

Pro náročnost změny (NZ) značí:

- hodnota 1, že zakomponování změny do metodiky je nejjednodušší a implikuje minimální změny celé metodiky.

- hodnota 5, že zakomponování vlastnosti do metodiky je velmi těžké a s velkou pravděpodobností i nemožné, jelikož by změna implikovala zásadní změnu celé koncepce a myšlenky metodiky.

Nejvhodnější metodika by měla mít nejnižší celkový součet hodnocení vlastností, které je násobeno příslušnými vahami.

Tabulka 2: Vícekriteriální rozhodování výběru z vyčtených agilních metodik

Metodika	Vlastnost	Rychlá reakce na změny		Zpětnovazební smyčka		Synchronizace vývoje obou platform		Důkladnější analýza úkolů před implementací		Celkem
	Váha	3		2		3		1		
	Kritérium	MN	NZ	MN	NZ	MN	NZ	MN	NZ	
XP		2	2	1	1	3	4	4	5	46
Kanban		1	1	4	5	5	5	5	5	64
Scrumban		1	1	4	3	5	3	5	5	54
Scrum		3	2	1	1	2	3	3	1	38

Rámec **XP** by splňoval podmínky rychlé zpětnovazební smyčky a částečně i velmi rychlou reakci na změny (v případě nutnosti by se přerušil týdenní cyklus a nedokončené úkoly by se přenesly do dalšího cyklu). Synchronizace vývoje je teoreticky možná pomocí „Spiků“, které by vždy stačilo přiřadit vývojářům platformy, která je napřed, ovšem toto nelze považovat za obecné řešení. Problémem je to, že metodika definuje plánování a výhled pouze na aktuální týden, což přináší minimální možnosti synchronizace. Hlavní nevýhodou ovšem je to, že plánování na aktuální týden v podstatě vylučuje možnost důkladnější analýzy úkolů předem, jelikož by buď značnou mírou narušovala cyklus předchozí, nebo by bylo vynucované její dokončení před začátkem cyklu (bez jejího dokončení by všechny práce pro daný cyklus byly blokovány).

Kanban je perfektním kandidátem pro velmi rychlou reakci na změny. Ovšem, zpětnovazební smyčka je příliš krátká a pro vyřešení pokrytí této i zbylých vlastností by bylo potřeba velké množství nadstavby už tak velmi lehkého rámce. Tento rámec tedy také není příliš vhodný pro tvorbu metodiky v prostředí mobilních aplikací.

Scrumban je velmi zajímavou evolucí a posunem Scrumu, který rozhodně stojí za inspiraci, ovšem jeho finální podoba se nezdá jako nejvhodnější pro dané prostředí a žádala by si rozsáhlé úpravy pro to, aby vyhovovala všem požadovaným specifikům. Stejně jako Kanban, je velmi příhodná pro velmi rychlou reakci na změny, ovšem pokrytí zbylých vlastností by si kvůli kontinuálnímu toku vyžadovalo větší zásah. Ovšem, na rozdíl od Kanbanu, který je na kontinuálním toku postaven, je v tomto případě implementace nějaké formy ohraničených iterací jednodušší a nenarušuje myšlenku celé metodiky. Největším problémem by byla změna k vyhovění možnosti důkladnější analýzy úkolů před jejich samotnou implementací, jelikož to implikuje jejich prodlužování, což jde přímo proti myšlence Scrumbanu, který se snaží o zkrácení času mezi zadáním a doručením úkolu.

Posledním zbývajícím je **Scrum**, který díky svým Sprintům vyhovuje možnostem zpětnovazební smyčky. Rychlá reakce na změny v něm není tak úplně možná, ovšem nabízí se jednoduché řešení přerušení Sprintu a následné vyřazení úkolů, které odpovídají časové náročnosti přerušení, ze zbylé části Sprintu. Synchronizace vývoje obou platforem je o něco přístupnější než v případě XP díky možnosti delších než týdenních cyklů, ovšem obecné řešení by si také vyžadovalo mechaniku dlouhodobějšího plánování, která ve Scrumu není zakořeněna. Scrum zahrnuje analýzu úkolů během Sprint planningu, ovšem jedná se pouze o povrchní analýzu. Větší detailnost analýzy lze zajistit zvětšením rozsahu Sprint planningu nebo vyžadováním důkladné individuální přípravy před samotnou událostí.

3.2 Vyhodnocení výběru metodiky

Dle výsledků z předchozího oddílu je nejbližší vhodnou metodikou **Scrum**. Jelikož je celkové hodnocení i této nejbližší metodiky velkou mírou vzdáleno od ideálního výsledku (nejnižší možné hodnocení je 18, Scrum dosáhl hodnoty 38), je nutné ve finálním návrhu počítat s nemalými změnami oproti této metodice.

Praktiky a postupy ostatních metodik jsou zohledněny jako inspirace a při vytváření metodiky je použit výše zmíněný hybridní přístup. Jako základ ovšem slouží Scrum.

3.3 Nevyhovující části vybrané metodiky

Před počítáním tvorby samotné metodiky je nutné definovat, které části výchozí metodiky je nutné upravit, tak aby vyhovovala očekávaným vlastnostem. Mezi problematické oblasti patří **Sprint Planning** a **Sprint**.

Součástí Sprint Planningu je z definice také analýza úkolů a diskuze nad tím, jak budou tyto úkoly řešeny v rámci Sprintu. Ovšem pokud by mělo dojít k hluboké analýze, mohl by rozsah schůzky plánování značnou měrou narůst. Důkladnou analýzu je tedy nutné řešit jiným způsobem.

Sprint, jakožto časově ohraničená iterace, se může stát překážkou pro rychlé reakce na změny. Jak bylo zmíněno výše, i týdenní iterace může být v tomto ohledu problematická. Scrum samotný obsahuje mechaniku k přerušení Sprintu, ovšem je brána pouze jako krajní

řešení a měla by být použita pouze v případě, že dosažení cíle Sprintu ztratilo smysl, což neodpovídá tomuto případu. Koncept Sprintu je tedy nutné přehodnotit tak, aby bylo umožněno rychle reagovat na změny.

Vzhledem k tomu, že je zasahováno do základních částí rámce, je nutné zmínit, že tyto změny se mohou odrazit také ve změně jiných částí, které nejsou v tomto oddílu zmíněny.

4 Návrh metodiky

V rámci této kapitoly je navržena kompletní metodika pro vývoj mobilních aplikací. Základem pro metodiku je agilní rámec Scrum, který je upraven na základě analýzy z předchozí kapitoly.

Jak již bylo zmíněno, metodika by měla být velmi flexibilní, měla by umožňovat hlubší analýzu úkolů a měla by obsahovat mechaniky pro synchronizaci vývoje jednoho produktu pro dvě různé platformy.

Nedílnou součástí metodiky by měly být také metriky, které by umožnily analýzu, vytvoření trendů a ukazatelů a případně přehodnocení jednotlivých částí metodiky. Metriky jsou vhodné nejen pro hodnocení celé metodiky a její přínosnosti, ale i k průběžnému měření výkonnosti týmů a jejich výkyvů, které je nutné podchytit a řešit včas.

Samotný návrh je strukturován dle základních elementů Scrumu, kterými jsou role, události a artefakty. K těmto elementům jsou přidány také formální ustanovení a metriky, které ve Scrumu definovány nejsou.

Role jsou popsány v [oddílu 4.1](#). Na ně navazuje rozdělení a struktura typů úkolů, které dávají do kontextu zodpovědnosti definovaných rolí. Následně jsou pro jednotlivé role popsány jejich pohledy na workflow v takové formě, aby získávaly ze stavů jednotlivých úkolů pouze pro ně relevantní informace.

Události jsou popsány formou rozdělení projektu v čase v [oddílu 4.4](#). Artefakty jsou zachovány v původní formě tak, jak jsou definovány Scrumem.

V neposlední řadě jsou metodikou definována formální ustanovení pro jasnost hodnot, které z velké části vychází z hodnot Scrumu. Pro tyto potřeby jsou definovány dokumenty a ustanovení v [oddílu 4.5](#).

Poslední částí návrhu metodiky jsou metriky, pomocí kterých je možné hodnotit jak efektivitu metodiky samotné, tak výkonnost a efektivitu týmu v čase.

4.1 Role

Vzhledem k potřebě hlubší analýzy úkolů, kterou již kvůli jejímu rozsahu není možné provádět během Sprint Planningu, je nutné definovat novou roli, která bude mít tuto analýzu na starost. Tato role je pojmenována *Solution Expert* a je více přiblížena v [oddílu 4.1.4](#).

Nová metodika již není čistým Scrumem a i z toho důvodu nedává název role Scrum Master smysl. Nově je tedy přejmenována na *Agile Coach*.

Následuje popis funkcí a zastoupení jednotlivých rolí.

4.1.1 Product Owner

Role Product Ownera (dále jen PO) není nijak odlišná od té definované Scrumem. Jedná se o roli, která má převážně na starosti organizaci backlogu: vytváření úkolů, jejich řazení a určování priorit a hodnot jednotlivých úkolů.

Měla by to být osoba, která má nejbližší ke klientovi produktu a přetváří jeho představu do konkrétních kroků, které jsou definovány společným jazykem tak, aby byly pochopitelné i netechnicky zaměřeným člověkem, ale i tak, aby je dokázal identifikovat a pochopit vývojový tým v dostatečném rozsahu.

Samotné úkoly by měly být popisovány formou Stories, tak jak je popisuje XP. Nedílnou součástí Stories by měla být akceptační kritéria, jejichž vytvoření je také zodpovědností PO.

Vzhledem k tomu, že PO není jedinou rolí, která může založit úkol do backlogu (např. tester může založit úkol k vyřešení nalezeného bugu), je nutné, aby používala frekventovaně techniku *backlog grooming*, která značí řazení úkolů v backlogu dle jejich priorit a hodnot.

PO by měl také projektovat postup projektu v čase, a tudíž jeho další odpovědností je udržování roadmapy, která by měla jasně značit, kam se bude projekt dále ubírat. Technika projekce pomocí roadmapy je více přiblížena v sekci 4.4.4 Projekce iterací.

4.1.2 Agile Coach

Agile Coach (dále jen AC) by měl odpovídat roli Scrum Mastera ve Scrumu. Jeho hlavní úlohou je podpora týmu a validace dodržování procesů definovaných metodikou. Této úlohy dosahuje pomocí řízení komunikace v týmu, vedení a plánování schůzek týmu a odstraňování překážek.

Jednou z hlavních odpovědností této role je řízení vnitřní efektivity týmu, čehož by měl dosáhnout pomocí užití technik a praktik definovaných metodikou. Vnitřní efektivitě týmu by měla také pomoci jeho funkce prostředníka, která značí, že on by měl být první osobou snažící se o vyřešení překážky objevené v průběhu vývoje. Tím je zajištěno, že vývojový tým není zbytečně vyrušován a může se maximálně soustředit na vykonání svých činností.

Je také zodpovědný za analýzu ukazatelů výkonnosti týmu a měl by navrhopvat oblasti, ve kterých by se tým mohl zlepšit. Tyto návrhy by dále měly být prodiskutovány v rámci schůzek retrospektivy.

Role AC nemusí nutně být zastoupena jednou osobou, která nemá jinou roli v týmu. Je možné, aby například jedna osoba zastupovala roli PO i AC. Ovšem osoba vykonávající roli AC by nikdy neměla být součástí vývojového týmu.

4.1.3 Vývojový tým

Scrum definuje vývojový tým jako skupinu osob všech profesí a rolí, které jsou potřebné ke zpracování úkolů projektu a nejsou dále děleny dle jejich zaměření. Ovšem vzhledem ke zmíněným variacím rozsahu vývoje (aplikace může být vyvíjena s již existujícím backendem

nebo designovým návrhem) je v rámci této metodiky nutné vývojový tým dle zaměření rozdělit. Toto rozdělení na druhou stranu poskytuje možnost větší specializace částí procesů na konkrétní role a zaměření, která může vést ke zvýšené efektivitě.

Vývojový tým lze tedy rozdělit na následující role:

- Mobilní vývojář: Programátor vyvíjející samotnou mobilní aplikaci (bez ohledu na zaměření na konkrétní platformu)
- Tester
- UX/UI designér (může a nemusí být součástí týmu, záleží na situaci)
- Backend vývojář (může a nemusí být součástí týmu, záleží na situaci)

I přesto, že je vývojový tým dělen, nic to nemění na tom, že by měl táhnout za jeden provaz a navzájem si pomáhat v dosahování cílů celého týmu.

4.1.4 Solution Expert

Hlavním úkolem nově vytvořené role Solution Experta (dále jen SE) je analyzování úkolů a návrh postupu jejich řešení. Jak bylo zmíněno v předchozí kapitole, kvůli okolnostem prostředí vývoje mobilních aplikací je nutná hlubší analýza úkolu před samotným počítáním jejich implementace. Tato činnost se tedy může stát časově náročnou a její začlenění do Sprint Planningu již nedává smysl.

Roli SE nemusí zastupovat nezávislá osoba, naopak je doporučeno, aby tuto roli zastupoval některý ze členů vývojového týmu, jelikož má největší přehled o kontextu projektu a zároveň nutné technické znalosti. Tato role by měla být zastoupena nejzkušenějším vývojářem, případně by se měla průběžně předávat mezi zkušenými vývojáři v rámci týmu pro rozprostření celkové zátěže, která se s vykonáváním této role nese.

S ohledem na rozsah vývoje může být potřebné vytvořit více SE rolí. Tato potřeba nastává již při nativním vývoji aplikací pro obě platformy zvlášť. Tehdy musí být role rozdělena na *Master Solution Experta* (dále jen MSE), který má za úkol prvotní analýzu, a *Platform Solution Experta* (PSE), který má za úkol hotovou analýzu adaptovat na prostředí své platformy.

PSE rolí může být několik a může zastupovat: druhou mobilní platformu oproti MSE, vývoj backendu nebo UX/UI designový návrh. Pokud jsou úkoly komplexní nebo MSE není dostatečně kvalifikován pro komplexní prvotní analýzu, je vhodné, aby byla uspořádána schůzka všech SE, na které všichni společně úkoly analyzují. Je potřeba, aby tato schůzka nastávala pouze v nejnutnějších případech a v nezbytném rozsahu, jinak může mít negativní vliv na celkovou efektivitu týmu.

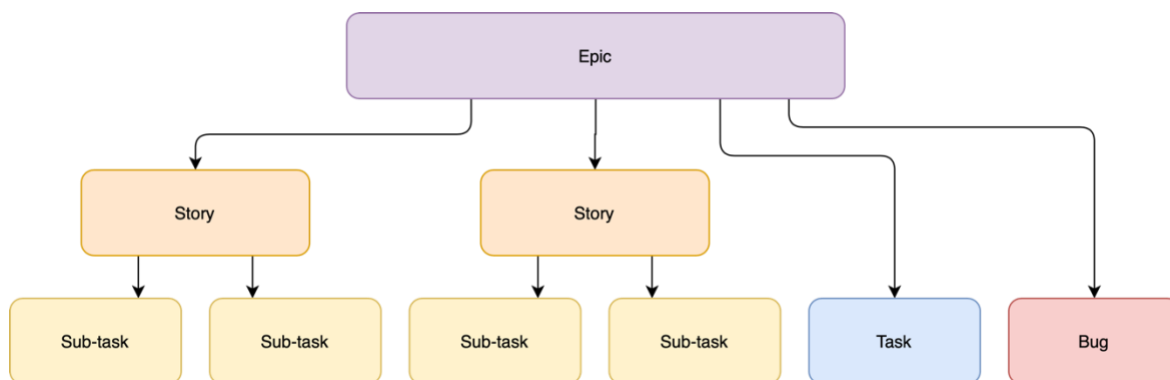
Pokud v rámci organizace nejsou jasně zavedené postupy řešení úkolů, doporučuje se nezavádět roli SE ihned při startu projektu. V takovém případě je žádoucí přistupovat k analýze úkolů v rámci Sprint Planningu tak, jak je definováno Scrumem, a až po ustálení konvencí pro řešení roli SE zavést.

4.2 Typy úkolů a jejich workflow

Nutnost důkladnějších analýz a stanovení dalších rolí (zejména role Solution Expert) implikuje také potřebu propracovanější strukturalizace úkolů, pro které musí být definované workflow, které odráží zodpovědnosti jednotlivých rolí.

Pro lepší strukturalizaci úkolů je nutné definovat jejich typy. Jednotlivé typy mají odlišnou funkci a tím pádem i odlišné parametry a workflow. Typy úkolů jsou rozděleny do tří úrovní:

1. Na nejvyšší a zároveň nejabstraktnější úrovni jsou definovány tzv. Epicy, které označují větší logické celky aplikace.
2. Epicy lze následně rozdělit na menší části: Stories. Ty by měly být definované formou, která je (alespoň povrchně) srozumitelná i pro netechnicky zaměřeného člověka.
3. Nejnižší úrovní jsou Tasky, Buggy a Sub-tasky (podúkoly). Ty obsahují specifikaci řešení, se kterou již přímo pracují členové vývojového týmu.



Obrázek 4: Diagram strukturalizace úkolů

Hrubší členění není kvůli proměnlivosti prostředí a průměrným velikostem projektů mobilních aplikací nutné. Ovšem pokud je to vzhledem k povaze, a hlavně rozsahu projektu, vhodné, je možné mezi první a druhou úroveň vložit další úroveň (např. Feature). Tyto úrovně pak mají totožné chování jako úroveň první a definují stav úrovně, která se v hierarchii nachází přímo nad ní.

Každý úkol je specifikován svou kartou, která má různé parametry. Pro jednotlivé typy úkolů jsou definovány jejich specifické parametry, ovšem všechny úkoly (kromě Epiců) by měly sdílet následující:

- Identifikátor
- Název
- Popis
- Zadavatel (ten, který úkol vytvořil)
- Řešitel
- Platforma (iOS/Android nebo jiné)
- Rizikovitost (nízká/střední/vysoká, zejména dle zkušeností s technologiemi pro řešení úkolu)
- Priorita (nízká/střední/vysoká)

- Související úkoly (převážně výpis těch, které musí být dokončeny před řešením daného úkolu)

Pokud existují nějaké zdroje k úkolu, jako např. sada ikon, měly by být samozřejmě také součástí karty úkolu.

Workflow každého typu úkolu je specifické. Jediným společným znakem všech úkolů je, že každý z nich lze z jakéhokoli stavu uzavřít.

4.2.1 Epic

Jak již bylo zmíněno v úvodu oddílu, *Epic* je větším logickým celkem aplikace. Neměl by představovat příliš velký objem práce (např. roční časová náročnost je přespříliš), ovšem stále by se jeho časová náročnost měla pohybovat alespoň v řádu člověkotýdnů. Epic lze následně dělit na Stories. Epicy se používají pro plánování z dlouhodobějšího pohledu.

Jednoduchým příkladem Epicu může být například „Nákupní košík“ nebo „Přihlášení“.

Karta

S ohledem na to, že Epic je pouze množinou úkolů, nemusí jeho karta obsahovat tolik informací. Nicméně stále by měla obsahovat: identifikátor, název, krátký popis a případně prioritu, která může být nápomocná při jeho plánování. Pokud nějak souvisí s jiným Epicem, měl by tento stav být v kartě také specifikován.

Workflow

Epic podléhá nejjednodušší formě workflow, která výsledně odráží pouze stav jeho podúkolů.



Obrázek 5: Workflow úkolů typu Epic

Tabulka 3: Stavy úkolů typu Epic

Stav	Význam
NOT DEFINED (nedefinováno)	Epic je založen, ovšem nejsou definovány jeho podúkoly.
READY (připraveno)	Všechny podúkoly Epicu (které jsou momentálně známé) jsou vytvořeny.
IN PROGRESS (probíhá)	První z podúkolů Epicu se začal zpracovávat.

4.2.2 Story

Termín *Story* byl již několikrát zmíněn a představuje funkcionalitu aplikace, která je popsána ve formě „příběhu“. Slouží k rozdělení Epiců a jedná se o styčný bod porozumění mezi vývojovým týmem a okolím. Story by měla být úkolem, jehož zadání dokáže interpretovat všechny zaujaté strany. Časová náročnost k dokončení Story by nikdy neměla přesáhnout jeden člověkotýden, aby ji bylo možné rozumně plánovat.

Story by měla být specifikována formou popisu situace z pohledu uživatele, což umožní jednodušší pochopení z pohledu netechnicky zaměřeného člověka a zároveň umožní větší pochopení produktového pohledu na aplikaci vývojovým týmem. Pro vyvolání ještě znatelnějšího produktového citění je dobrou praktikou nepopisovat situaci z pohledu běžného uživatele, ale přímo z pohledu konkrétního zástupce cílového segmentu (např. „Jako teenager Filip.“ nebo „Jako babička Božena.“). Větší identifikací vývojářů s produktem lze podchytit stavy, kdy se objevují nedostatky v řešení kvůli špatné interpretaci zamýšlené funkcionality.

Od chvíle definování Story je k ní přiřazen jako řešitel Solution Expert, který je zodpovědný za její dokončení.

Karta

Mimo základních parametrů by měla Story obsahovat hlavně rozsáhlejší specifikaci, a to včetně všech potřebných zdrojů, odkazů na designový návrh nebo specifikaci API apod. Je nutné si uvědomit, že Story je jediným místem předání informací mezi PO a zbytkem týmu a touto cestou by tedy měla být všechna relevantní data předána.

Nedílnou součástí Story musí být také kritéria, dle kterých lze potvrdit, že je úkol dokončen, tedy **akceptační kritéria**. Typicky jsou popsána formou Given – When – Then, což znamená, že je popsán stav, akce, která v tomto stavu nastane, a očekávaný výsledek. Příkladem může být akceptační kritérium „Nacházím se na detailu produktu – Kliknu na `Přidat do košíku` – Produkt je přidán do košíku“.

Karta musí obsahovat také číslo verze aplikace, ve které by daná funkcionalita měla být obsažena. Toto číslo by měl přiřadit, případně aktualizovat, Product Owner nejpozději před započítáním iterace.

Story je zároveň prvním bodem strukturalizace, kde již dává smysl odhadovat pracnost úkolu. Zachování Story Points, jakožto jednotek práce, jak jsou popsány Scrumem, je v tomto případě rozumné.

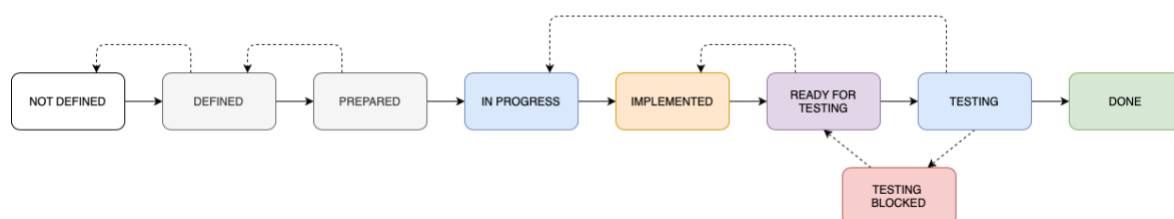
Po analýze Story jsou k její kartě přidány také sub-tasky, jejichž postup definuje stav samotné Story.

Jakmile je dokončena implementace, měla by být také doplněna informace, kde je možné řešení otestovat. Může se jednat čistě jen o číslo verze aplikace, o link ke stažení nebo klidně jen o informaci o tom, koho má tester požádat o sestavení aplikace k testování, pokud tedy nejsou správně nastaveny procesy CD (toto je krajní řešení, které se samozřejmě negativně projeví na efektivitě).

V případě, že je během testování Story nalezen nedostatek nebo chyba, je testerem vytvořen nový sub-task, který popisuje nalezený nedostatek nebo chybu a vyžaduje jeho opravu.

Workflow

Jak bylo zmíněno, stav Story je převážně odražen ve stavu jejich sub-tasků. Sama ovlivňuje svůj stav pouze ve fázi testování.



Obrázek 6: Workflow úkolů typu Story

Tabulka 4: Stavby úkolů typu Story

Stav	Význam
NOT DEFINED (nedefinováno)	Počáteční stav Story, který značí, že ještě není plně definována.
DEFINED (definováno)	Story je detailněji specifikována, obsahuje akceptační kritéria a všechny potřebné zdroje. Pokud něco z těchto parametrů chybí, může být Story vrácena do stavu NOT DEFINED.
PREPARED (připraveno)	Tento stav značí, že byla Story analyzována a rozdělena na jednotlivé sub-tasky. Pokud analýza objeví nedostatek ve specifikaci, může být vrácena do stavu DEFINED.
IN PROGRESS (probíhá)	Začalo zpracování minimálně jednoho ze sub-tasků.
IMPLEMENTED (implementováno)	Všechny sub-tasky byly dokončeny.
READY FOR TESTING (připraveno k testování)	Story je připravena k otestování. Znamená to také, že obsahuje informaci, kde je možné implementaci Story otestovat. Pokud tuto informaci neobsahuje nebo neodkazuje na existující verzi, může být Story vrácena zpět do stavu IMPLEMENTED.
TESTING (testování)	Probíhá testování. V případě odhalení nedostatku je založen nový sub-task a Story je vrácena do stavu IN PROGRESS.

TESTING BLOCKED (testování blokováno)	Stav, který odráží, že Story není možné testovat. Důvody mohou být různorodé a nelze specifikovat jejich obecné řešení. Tento stav by měl nastávat výjimečně a tester sám by ho měl operativně řešit s Agile Coachem jako překážku, kterou je nutné odstranit.
DONE (dokončeno)	Všechny části Story byly implementovány a úspěšně otestovány. Story je považována za dokončenou.

4.2.3 Implementační sub-task

Sub-tasky vznikají během analýz SE a jsou podmnožinou jednotlivých Stories. Na rozdíl od Story jsou technicky popsány a místo abstraktního popisu funkcionality popisují přímo konkrétní řešení problému. Jedná se o typy úkolů, se kterými přímo pracují vývojáři.

Karta

Sub-task by měl stejně jako Story obsahovat odhad pracnosti. Tento odhad může být částečně zkreslený odhadem rodičovské Story, ovšem i přesto může sloužit k upřesnění celkových odhadů. Tento odhad přiřazuje Solution Expert během analýzy Stories. Pokud součet odhadů všech sub-tasků není roven odhadu Story, měl by být odhad ve Story upraven.

Není to povinností, ovšem je vhodné v kartě používat tagy¹², které odráží např. architektonickou vrstvu aplikace pro lepší pochopení a filtrování úkolů ze strany vývojářů.

Pro potřeby analýzy a snazší pochopení obsahu řešitelem by implementační sub-tasky měly být ještě označeny tagy odrážející jejich typy: **standardní**, **změnový** (značí změnu, která byla přidána až po dokončení prvotní analýzy) a **chybový** (značí, že tester našel chybu nebo nedostatek během testování implementace rodičovské Story).

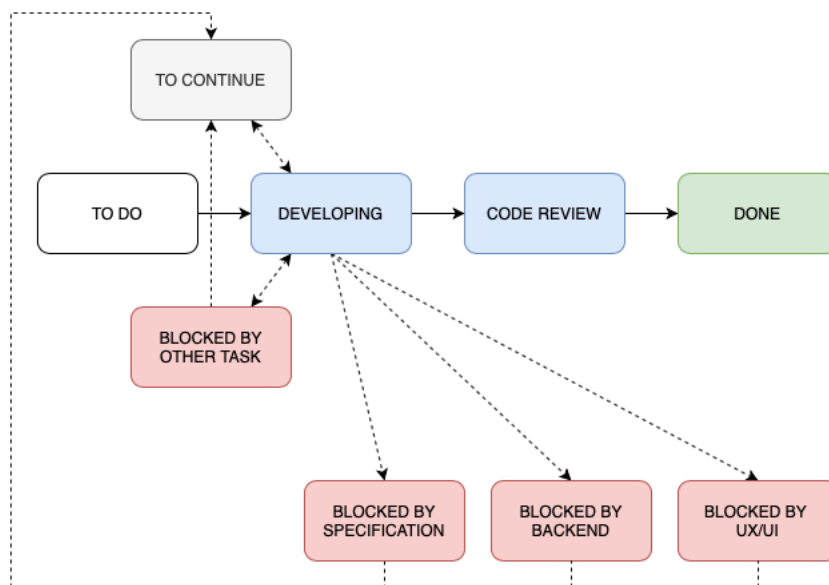
Workflow

Žádoucí tok úkolu definuje typický proces vývoje. Specifickou částí workflow je několik existujících stavů „BLOCKED BY..“, které kategorizují typy nedostatku informací nebo prostředků pro implementaci daného úkolu. Rozdělením zdánlivě jednoho stavu do několika je zajištěna jasná odpovědnost za odblokování úkolu:

- BLOCKED BY SPECIFICATION (blokováno specifikací) – úkol může odblokovat SE doplněním specifikace úkolu
- BLOCKED BY BACKEND (blokováno backendem) – úkol může odblokovat Backend vývojář
- BLOCKED BY UX/UI (blokováno UX/UI) – úkol může odblokovat UX/UI designér

¹² Štítek nebo značka. Standardně určuje zařazení.

Tyto stavy nejsou povinné a záleží na konkrétním rozsahu vývoje (např. pokud se aplikace vytváří s již existujícím backendem, a tím pádem bez role Backend vývojář, nedává smysl stav „BLOCKED BY BACKEND“, a v případě existujícího nedostatku backendu se jedná o stav nedostatečné specifikace). Druhou možností je zachování těchto stavů při jakémkoli rozsahu (pro usnadnění analýz), ovšem za těchto okolností musí být implementace workflow doprovázena změnou rolí odpovědných za odblokování.



Obrázek 7: Workflow úkolů typu Implementační sub-task

Tabulka 5: Stavy úkolů typu Implementační sub-task (bez stavů „BLOCKED BY..“, které jsou vysvětleny v textu pasáže)

Stav	Význam
TO DO (ke zpracování)	Úkol byl založen a je připraven ke zpracování.
TO CONTINUE (k pokračování)	Tento stav značí, že zpracování úkolu již začalo, ale momentálně na něm není pracováno. Důvodem k odložení do tohoto stavu je většinou potřeba odblokování úkolu jinou osobou než je řešitel. Do tohoto stavu je možné se dostat také odložením úkolu během jeho řešení, kvůli naléhavější situaci, kterou je nutné řešit. K tomuto scénáři by ovšem mělo docházet výjimečně a větší výskyt těchto situací naznačuje existující problém v procesu vývoje.
BLOCKED BY OTHER TASK (blokováno jiným úkolem)	Po začátku zpracování úkolu bylo objeveno, že je závislý na dokončení jiného souvisejícího úkolu. Tato skutečnost by měla být odhalena již v rámci analýzy Stories, a proto by tento stav neměl nastávat a jeho častý výskyt značí existující problém, který je nutno řešit.
DEVELOPING (vyvíjeno)	Úkol je momentálně zpracováván.

CODE REVIEW (revize kódu)	Úkol byl implementován a čeká na revizi kódu. Toto je běžná praktika vývoje, kdy jiný vývojář prochází nově implementovaný zdrojový kód a ověřuje jeho validitu. Tato metodika definuje různé úrovně Code Review, což je více přiblíženo v sekci Rozsah testování kvality .
DONE (dokončeno)	Úkol je implementován a ověřen. Je považován za dokončený.

4.2.4 Neimplementační sub-task

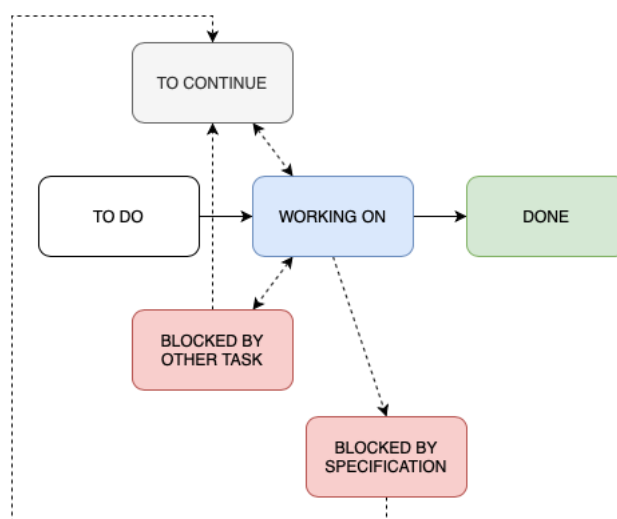
Úkoly typu Neimplementační sub-task také vznikají během SE analýzy a jsou také podmnožinou jednotlivých Stories. Řešení těchto úkolů ovšem nezahrnuje implementaci, ale spíše další podpůrné činnosti. Zpracování Neimplementačních sub-tasků typicky předchází samotné implementaci.

Karta

Karta by v podstatě měla odpovídat kartě Implementačních sub-tasků. Stejně je taky vhodné použít tagy, které by v tomto případě indikovaly oblast zaměření sub-tasku (např. UX výzkum).

Workflow

Jedná se o redukovanou formu workflow Implementačních sub-tasků a významy jednotlivých stavů jsou tedy zachovány.



Obrázek 8: Workflow úkolů typu Neimplementační sub-task

4.2.5 Task / Bug

Úkoly typu *Task* a *Bug* se od *Story* liší zejména tím, že nepodléhají SE analýze. *Task* je úkol, který je tak malý, že není potřebné ho podrobovat postupu tvorby *Story* (např. změna barvy nebo umístění tlačítka). *Bug* je specifický tím, že je vytvořen na základě nalezené chyby, ovšem v rámci procesu se chová stejně jako *Task*, a proto byly tyto dva typy z pohledu workflow sjednoceny. Pokud se ovšem stane, že je vytvořený *Task* nebo *Bug* velkého rozsahu (tzn. je objemem blíže rozsahu *Story* než sub-tasku), měl by být uzavřen a znovu vytvořen jako *Story* tak, aby podlehл všem jejím náležitostem, včetně SE analýzy.

V neposlední řadě je nutné zmínit, že *Bug* jako samostatný úkol může vzniknout pouze při testování, které přímo nesouvisí s nově implementovanou *Story*. Znamená to tedy, že je nalezena chyba nebo nedostatek např. při systémovém testování před vydáním nebo přímo v produkčním prostředí. Pokud je nalezena chyba nebo nedostatek při testování *Story*, měl by být založen další implementační sub-task s tagem „chybový“.

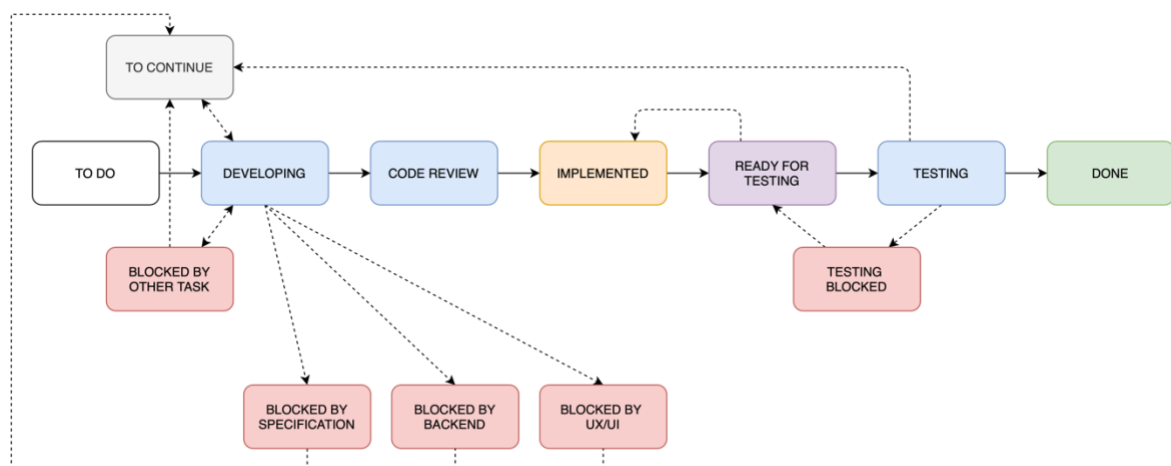
Karta

Oproti základním parametrům může karta *Tasku* nebo *Bugu* obsahovat také odhad pracnosti, pokud to dává smysl. *Bug* by měl ještě speciálně obsahovat popis prostředí, ve kterém byl nalezen (např. verzi aplikace, testované zařízení a verzi OS).

Jelikož tyto typy úkolů se, stejně jako *Story*, podrobují testování, měla by k nim po implementaci být doplněna informace o tom, kde je možné řešení úkolů otestovat.

Workflow

Workflow *Tasku* a *Bugu* je jakýmsi hybridním řešením mezi *Story* a Implementačními sub-tasky, jelikož obsahuje jak fázi vývoje, tak fázi testování. Jeho tok je nezávislý.



Obrázek 9: Workflow úkolů typu *Task* nebo *Bug*

Vzhledem k tomu, že je workflow složeno ze stavů a částí workflow ostatních typů úkolů, není potřebné znovu tyto stavy objasňovat.

4.3 Pohledy jednotlivých rolí na workflow

Vzhledem k relativně vysoké komplexnosti všech workflow je nutné, aby jednotlivým rolím byly poskytnuty uzpůsobené pohledy a byla pro ně přehlednější a čitelnější jejich část celého procesu. Workflow navržená v přechozím oddílu v podstatě znemožnila možnost použití fyzického Boardu. Implementace pohledů rolí tuto možnost vylučuje úplně a pro implementaci této metodiky je tedy nutné použít digitální Board.

Každá role má definované stavy, které jsou pro ni relevantní. V případě PO a AC je relevantnost definována potřebou informovanosti, pro ostatní role znamená relevantnost v tomto kontextu schopnost a možnost změnit aktuální stav úkolu. Jelikož stav z pohledu role může odrážet více stavů různých typů úkolů, jsou tyto skutečnosti zapsány v tabulce formou matice.

Epicy jsou kvůli jejich specifitě a nízké nutnosti sledování jejich stavu na denní bázi z těchto pohledů vynechány.

4.3.1 Product Owner a Agile Coach

Tyto dvě role mají spíše řídicí funkci, a tudíž potřebují agregovaný pohled, který by měl jen informovat o aktuálním stavu, ve kterém se tým nachází.

Tabulka 6: Pohled rolí Product Owner a Agile Coach na stavy jednotlivých typů úkolů

Product Owner, Agile Coach	TO DEFINE (k definování)	BLOCKED (blokováno)	DEVELOPING (vyvíjeno)	TESTING BLOCKED (testování blokováno)	TESTING (testováno)	DONE (dokončeno)
<i>Story</i>	NOT DEFINED		DEFINED, PREPARED, IN PROGRESS, IMPLEMENTED	TESTING BLOCKED	READY FOR TESTING, TESTING	DONE
<i>Impl. sub-task</i>		BLOCKED BY..	TO DO, TO CONTINUE, BLOCKED BY OTHER TASK, DEVELOPING, CODE REVIEW			DONE
<i>Neimpl. sub-task</i>		BLOCKED BY..	TO DO, TO CONTINUE, BLOCKED BY OTHER TASK, WORKING ON			DONE

<i>Task/Bug</i>		BLOCKED BY..	TO DO, TO CONTINUE, BLOCKED BY OTHER TASK, DEVELOPING, CODE REVIEW, IMPLEMENTED	TESTING BLOCKED	READY FOR TESTING, TESTING	DONE
-----------------	--	--------------	---	--------------------	-------------------------------------	------

4.3.2 Vývojový tým

V kontextu tvorby pohledů je přínosné rozdělení vývojového týmu podle zaměření, jelikož pohled každé z těchto podrolí je značně odlišný. V následujících tabulkách jsou tedy představeny pohledy pro všechna zaměření vyskytující se ve vývojovém týmu.

Tabulka 7: Pohled role Mobilní vývojář na stavy jednotlivých typů úkolů

<i>Mobilní vývojář</i>	TO DO (ke zpracování)	BLOCKED (blokováno)	TO CONTINUE (k pokračování)	IN PROGRESS (probíhá)	CODE REVIEW (revize kódu)	IMPLEMENTED (implementováno)
<i>Impl. sub-task</i>	TO DO	BLOCKED BY OTHER TASK	TO CONTINUE	DEVELOPING	CODE REVIEW	
<i>Task/Bug</i>	TO DO	BLOCKED BY OTHER TASK	TO CONTINUE	DEVELOPING	CODE REVIEW	IMPLEMENTED

Tabulka 8: Pohled role Tester na stavy jednotlivých typů úkolů

<i>Tester</i>	TO DO (ke zpracování)	BLOCKED (blokováno)	IN PROGRESS (probíhá)
<i>Story</i>	READY FOR TESTING	TESTING BLOCKED	TESTING
<i>Task/Bug</i>	READY FOR TESTING	TESTING BLOCKED	TESTING

Tabulka 9: Pohled role UX/UI designér na stavy jednotlivých typů úkolů

UX/UI designér	TO DO (ke zpracování)	TO UNBLOCK (k odblokování)	BLOCKED (blokováno)	TO CONTINUE (k pokračování)	IN PROGRESS (probíhá)
<i>Impl. sub-task</i>		BLOCKED BY UX/UI			
<i>Neimpl. sub-task</i>	TO DO		BLOCKED BY OTHER TASK	TO CONTINUE	WORKING ON
<i>Task/Bug</i>		BLOCKED BY UX/UI			

Tabulka 10: Pohled role Backend vývojář na stavy jednotlivých typů úkolů

Backend vývojář	TO DO (ke zpracování)	TO UNBLOCK (k odblokování)	BLOCKED (blokováno)	TO CONTINUE (k pokračování)	IN PROGRESS (probíhá)	CODE REVIEW (revize kódu)	IMPLEMENTED (implementováno)
<i>Impl. sub-task</i>	TO DO	BLOCKED BY BACKEND	BLOCKED BY OTHER TASK	TO CONTINUE	DEVELOPING	CODE REVIEW	
<i>Task/Bug</i>	TO DO	BLOCKED BY BACKEND	BLOCKED BY OTHER TASK	TO CONTINUE	DEVELOPING	CODE REVIEW	IMPLEMENTED

4.3.3 Solution Expert

Solution Expert potřebuje stejně jako PO a AC agregovaný pohled, ovšem nejsou pro něj relevantní všechny stavy všech typů úkolů. Pro tuto roli je nejdůležitější sledování stavu úkolů typu Stories, jelikož za tyto úkoly má zodpovědnost. Např. za přesun ze stavu IMPLEMENTED do stavu READY FOR TESTING má odpovědnost právě Solution Expert.

Tabulka 11: Pohled role Solution Expert na stavy jednotlivých typů úkolů

<i>Solution Expert</i>	TO ANALYSE (k analýze)	TO UNBLOCK (k odblokování)	IN PROGRESS (Story pobíhá)	IMPLEMENTED (implementováno)
<i>Story</i>	DEFINED		PREPARED, IN PROGRESS	IMPLEMENTED
<i>Impl. sub-task</i>		BLOCKED BY SPECIFICATION		
<i>Neimpl. sub-task</i>		BLOCKED BY SPECIFICATION		
<i>Task/Bug</i>		BLOCKED BY SPECIFICATION		

4.4 Rozdělení projektu v čase

S ohledem na zmiňovanou potřebu velmi rychlých reakcí na změny není možné dále držet koncept časově ohraničených Sprintů. Jako základ tedy dává smysl využít kontinuální tok, nad který ovšem musí být nastaveno určitou formou iterací tak, aby zpětnovazební smyčka dosáhla úměrně smysluplné délky.

Z těchto důvodů metodika definuje koncept **rozsahem ohraničených iterací**. Nejedná se o iterace, které jsou časově ohraničené, ale které jsou **ohraničené objemem práce** v nich obsažené. Jejich délka v čase je tedy definována časovým odhadem všech úkolů, které jsou do iterace zahrnuté. Pro tyto účely jsou tedy potřeba přesnější odhady náročností úkolů.

4.4.1 Přepočet na časovou náročnost

Jak již bylo zmíněno, jednotlivé Stories jsou odhadované pomocí Story Points (dále jen SP). Časovou náročnost dle hodnoty SP lze dopočítat užitím ukazatele **Story Point Time** (dále jen SPT), který určuje koeficient převodu SP na člověkohodinu. Tento koeficient by měl v počátku projektu představovat odhadovanou výkonnost týmu dle schopností jednotlivých členů a mělo by být primárně vycházeno z historických dat. Pokud se jedná o úplně první projekt s použitím této metodiky, je nutné tuto hodnotu odhadnout dle prvních odhadovaných náročností ve Story Points a odhadu týmu o reálné časové náročnosti úkolů. Tento koeficient by měl být přehodnocen dle aktuálních dat po skončení každé iterace.

Přesnějším odhadům by mělo napomoci určování stupně rizikovosti jednotlivých úkolů (nízká/střední/vysoká). Ukazatel SPT by tedy měl být určen třemi hodnotami pro každý

stupeň rizikovitosti. Časová náročnost úkolů je tedy určena vynásobením SP s odpovídající hodnotou SPT dle stupně rizikovitosti úkolu.

4.4.2 Iterace

Jednotlivé iterace by měly obsahovat logické celky, které lze nejlépe odrazit příslušností řadových úkolů k nejvyššímu stupni úkolu, tedy k Epicu. Iterace by zároveň měla dosahovat **minimální délky jednoho týdne, maximálně však 4 týdnů**, stejně jako je definována délka Sprintu Scrumem (doporučený maximální rozsah pro nejvyšší míru efektivity je 2 týdny). Pokud není možné naplnit minimální délku iterace v rámci jednoho Epicu, může být obsahem i více Epiců, případně může být iterace doplněna o úkoly, které přímo nespadají pod specifický Epic (v tomto kontextu se bude jednat zejména o úkoly typu Task nebo Bug).

Tím, že iterace nejsou pevně zakořeněny v časovém rozložení projektu, je možné jednoduše zajistit jejich přerušení nebo částečné přerušení v případě, že nastane scénář, který si žádá okamžitý zásah (více popsán v sekci 4.4.5). Tuto mechaniku může vyvolat pouze role Product Owner a měla by být využívána vždy jen v nejnutnějších případech, protože je spojena s prodloužením přerušené iterace kvůli ztrátě kontextu.

4.4.3 Události

Rozsahem ohraničené iterace zachovávají jednu z velmi podstatných hodnot Scrumu, kterou je *závazek*. Tým se v rámci plánování zaváže k dokončení daného objemu úkolů, který je ihned projektován do časově ohraničeného rámce. Tím je zachován princip závazku a tým tak může společně směřovat k dosažení cíle iterace.

Zároveň je zachován i časový sled událostí definovaných Scrumem, které jsou žádoucí i v kontextu této metodiky s minimální potřebou obměny.

Iteration Planning

Každé iteraci předchází schůzka plánování, která nese příhodnější název *Iteration Planning* a zahrnuje pouze první část plánování (tak jak ho popisuje Scrum) a je během ní tedy pouze vybráno, respektive validováno, co bude obsahem následující iterace. Druhá část není potřebná, jelikož tomuto plánování již předchází analýza Solution Experta a tudíž otázka postupu řešení úkolů je již zodpovězena.

Standup

V průběhu iterace probíhá každý den *Standup*, který může mít poněkud lehčí formu. Vzhledem k tomu, že postup jakýchkoli úkolů je jednoduše dohledatelný v jejich digitální podobě, není nutné, aby jednotliví členové týmu vyjmenovávali, na čem konkrétně pracovali a budou pracovat. Hlavním zaměřením této schůzky by mělo být jen odhalování překážek. Každý by tedy měl říct, jaké oblasti se bude věnovat a jaké vidí potenciální riziko nebo reálnou překážku, která by neumožnila dokončení dalších úkolů. Agile Coach by měl všechny překážky, které během schůzky zazní, zaznamenávat a aktivně řešit jejich

odstranění před dalším Standupem. Z jeho úst by tedy mělo v rámci Standupu zaznít, jaký je aktuální stav všech překážek.

Iteration Review

Každá iterace by měla být, stejně jako ve Scrumu, zakončena Review, v tomto případě *Iteration Review*, v rámci kterého je představen výsledek iterace ve formě dema a je zhodnocen postup týmu. V případě, že některé z úkolů nemohly být v průběhu iterace dokončeny, je toto předmětem diskuze v rámci této schůzky. Schůzky by se měl účastnit také klient.

Iteration Retrospective

V neposlední řadě by po každé iteraci měla následovat retrospektiva, opět s upraveným názvem *Iteration Retrospective*. Kromě jména by ovšem měly být zachovány všechny aspekty této schůzky, kterou Scrum definuje jako Sprint Retrospective.

4.4.4 Projekce iterací

Koncept rozsahem ohraničených iterací dává možnost projektovat jejich plán do budoucna. Vzhledem k agilitě přístupu a proměnlivosti prostředí je možné dosáhnout pouze velmi hrubého plánu a na krátké období, ovšem zajištěním pravidelných aktualizací projekce lze dosáhnout reálného obrazu vývoje projektu v čase odrážejícího aktuální podmínky.

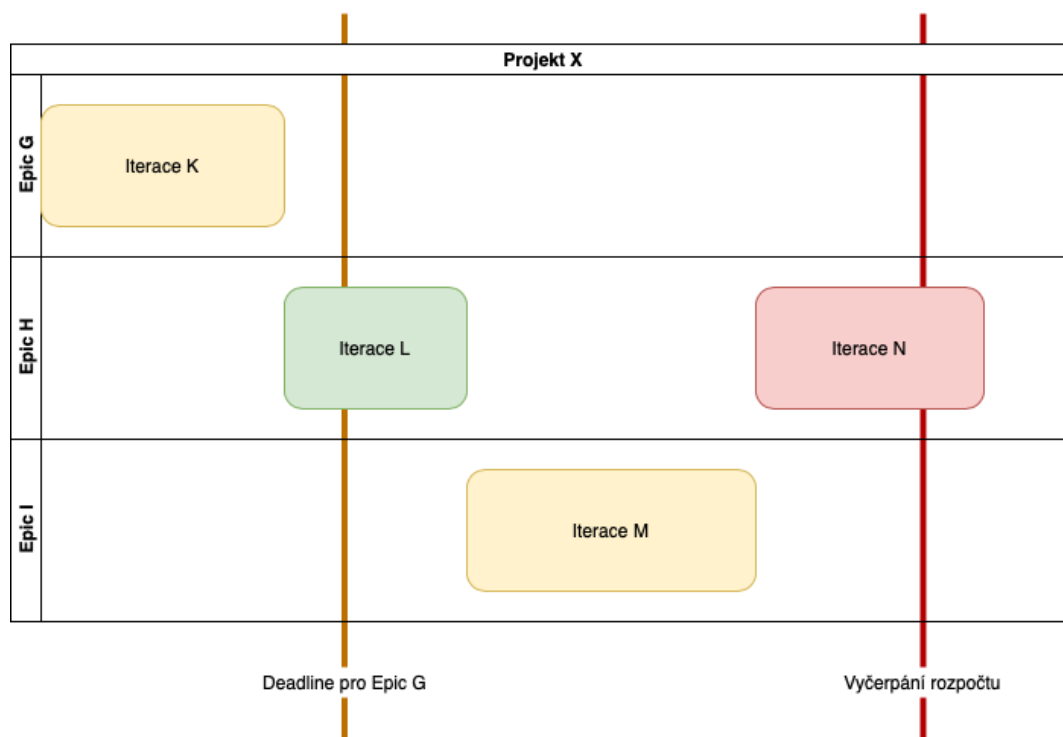
Technikou projekce iterací v čase je jejich zobrazování na **roadmapu**. Tato mapa představuje časový vývoj jednotlivých iterací a měla by být vždy veřejně dostupná všem stakeholderům projektu. Za pomoci tohoto projektování lze docílit nejen lepší představy o vývoji projektu z externího pohledu, ale také lepšího plánování interních kapacit.

Zakomponováním projekce dalších okolností projektu do roadmapy může být docíleno dřívějšího uvědomění, a tedy i dřívější reakce na blížící se překážku. Tou překážkou může být *deadline*¹³ pro implementaci konkrétní funkcionality nebo vyčerpání rozpočtu projektu, což je riziko, které je potřeba aktivně řídit v rámci agilního přístupu otevřenosti ke změnám.

Reálná implementace roadmapy může být variována s ohledem na situaci, ovšem musí vždy dodržovat zásady **pravidelné aktualizace a plné transparentnosti a dostupnosti**.

Následující příklad ukazuje jednoduchou implementaci roadmapy. Iterace K je poslední iterací Epicu G a její konec je projektován před *deadline* Epicu, ovšem zahrnuje pouze malou rezervu (což naznačuje žluté zbarvení). V podobném stavu je také iterace M, která se blíží k vyčerpání rozpočtu. V průběhu iterace N je projektováno vypršení rozpočtu, tudíž je tato iterace zbarvená červeně, což značí, že ji nebude možné dokončit a neměla by být tedy v tomto stavu odstartována.

¹³ Nejzazší termín, do kterého musí být práce dokončena.



Obrázek 10: Jednoduchý příklad implementace roadmapy

Právě projekce iterací řeší další z požadovaných vlastností definovaných v předchozí kapitole, kterou je možnost synchronizace vývoje obou platforem.

4.4.5 Speciální případ iterace: Hotfix

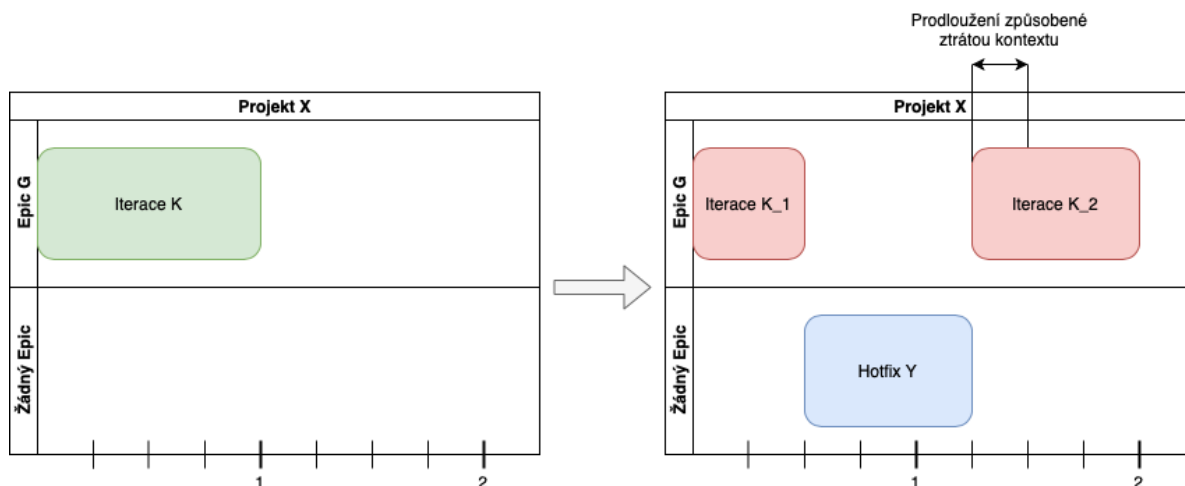
Jedním ze specifíků prostředí vývoje mobilních aplikací je složitý a zdoluhavý proces nasazení nové verze aplikace. Z tohoto důvodu je naprosto nezbytné na nalezený kritický bug v produkční verzi aplikace reagovat okamžitě. A přesně pro tyto scénáře metodika poskytuje mechaniku zvanou **Hotfix iterace**.

V popsaném případě je nezbytné okamžitě započít novou iteraci a pozastavit nebo alespoň částečně pozastavit aktuálně probíhající iteraci. Částečné pozastavení je zde zmíněno z důvodu, že kritický bug se nutně nemusí týkat obou platforem, a tudíž nemusí být nutné pozastavit práci celého týmu. V takovém případě by běžely obě iterace společně s tím, že čas původní iterace by se prodloužil úměrně délce hotfix iterace.

V rámci konceptu rozsahem ohraničených iterací je taková operace velmi jednoduchá. Je ovšem důležité zodpovědně a přesně tuto skutečnost zaznamenat do roadmapy a upravit projekci všech následujících iterací.

Jedno úskalí s sebou ale aplikování hotfix iterace nese. Tím je prodloužení času původní iterace, které s sebou nese snížení efektivity způsobené ztrátou kontextu. Toto je jeden z hlavních důvodů, proč je nutné tuto techniku využívat pouze v naprosto nezbytných případech.

Následující jednoduchý příklad zobrazuje hotfix iteraci v plném rozsahu, kdy je nutno kompletně přerušit stávající iteraci.



Obrázek 11: Jednoduchý příklad projekce hotfix iterace a změn, které vyvolává v roadmapě

4.5 Dokumenty a ustanovení

Ač se jedná o agilní metodiku, je nutné, aby byly v průběhu projektu vytvářeny a řízeny určité dokumenty, které pomáhají nejen udržení transparentnosti, ale i větší informovanosti všech stakeholderů o aspektech projektu.

4.5.1 Co je třeba definovat před začátkem projektu?

Před samotným začátkem projektu je nutné se sestaveným týmem odsouhlasit konvence a společné závazky, které se týkají vykonávání částí procesů projektu. Všichni členové týmu musí s navrženými parametry souhlasit proto, aby byl vytvořen společný cíl a směr, kterým se bude celý projekt ubírat. Parametry projektu jsou popsány v následujících sekcích.

Rozsah oblasti vývoje

Jak již bylo zmíněno ve specifikách prostředí vývoje mobilních aplikací, aplikace může být vyvíjena např. s již existující infrastrukturou nebo s existujícím designovým návrhem. Tuto část samozřejmě není možné rozporovat ze strany týmu, ovšem všichni by měli odsouhlasit, že si jsou těchto skutečností vědomi a že přijímají zodpovědnost, která je s danou situací spojena.

Rozsah testování kvality

Prostředí vývoje mobilních aplikací tak, jak je popsáno v úvodních částech práce, vyžaduje vysokou úroveň kvality před samotným nasazením, zejména kvůli značně omezené reakční době na přehlédnuté nedostatky aplikace objevené po jejím nasazení. Základní mechanismy pro udržení kvality tedy musí být nedílnou součástí metodiky.

Testování úkolů typu Story je zakomponováno v metodice, tudíž není přípustná další variace.

Vyšších úrovní kvality aplikace lze dosáhnout ve dvou oblastech: **systémové testování** a **code review**.

Realizace systémového testování před nasazením nové verze aplikace by měla být podmínkou. Ovšem otázkou je, zda by toto rozsáhlé testování mělo probíhat pouze před vydáváním nové verze nebo po každé iteraci.

Druhou formou variace kvality je code review, které může dosahovat několika úrovní:

1. Nejnižší úroveň obsahuje pouze statickou analýzu kódu.
2. Druhá úroveň zahrnuje mimo statické analýzy i testovací sestavení kódu posuzovatelem.
3. Třetí úroveň značí, že posuzovatel nejen kód spustí, ale i ručně otestuje novou funkcionalitu.
4. Nejvyšší úrovní je to, že posuzovatel sám napíše automatické testy pro otestování nové funkcionality. Pokud projdou úspěšně, je implementace nové funkcionality schválena.

Samozřejmě platí, že čím vyšší úroveň kvality, tím vyšší časová náročnost.

Alternativním přístupem může být použití TDD, které by značilo, že testy budou napsány ještě před samotným code review. Za předpokladu, že jsou nastaveny procesy CI a testy se spouštějí automaticky, nezáleží na rozsahu code review, jelikož žádná z definovaných úrovní nedokáže za této situace znatelně zvýšit úroveň kvality aplikace.

Výsledné rozhodnutí o rozsahu testování kvality je ovlivněno spíše rozpočtem projektu a požadavkem klienta, ovšem opět je nutné, aby celý tým potvrdil, že si je rozsahu testování kvality a zodpovědností s ním spojených vědom.

Definice „Připraveno“

Celý tým by se měl shodnout na své vlastní definici stavu „Připraveno“, který by měl odrážet stav úkolů, který lze považovat za kompletní proto, aby mohl být vpuštěn do procesu vývoje. Výstupem by měla být věta nebo série vět popisující kompletně připravený úkol.

Příkladem definice „Připraveno“ může být „Každý úkol musí mít popisný název, jasné pochopitelná akceptační kritéria a musí obsahovat zdroje ve vhodném formátu pro použití na mobilních platformách.“

Definice „Hotovo“

Stejně tak jako definice „Připraveno“, i definice „Hotovo“ by měla mít formu věty nebo série vět, která shrnuje stav úkolů, tentokrát ve stavu, který tým bude považovat za dokončený. Tato definice se bude částečně odrážet od definovaného rozsahu testování kvality, ovšem přesto je nutné, aby celý tým vědomě odsouhlasil, co je cílem jejich snažení.

Příkladem definice „Hotovo“ může být „Každý úkol je zodpovědně implementován a otestován v plném rozsahu dle specifikace. K návrhu i samotnému provedení bylo

přístupováno s myšlenkou maximálně podpořit uživatelskou zkušenost při používání aplikace.“

4.5.2 Report iterace

Výstupem každé iterace by měl být formalizovaný dokument, který má funkci reportu o průběhu iterace. Jeho základním obsahem je seznam dokončených úkolů v rámci iterace. Měl by také obsahovat hodnoty metrik naměřených během iterace s případným odůvodněním jejich výkyvu oproti standardním hodnotám.

V kontextu zhodnocení průběhu iterace je nutné vzít v potaz také možnost, že některé z vybraných úkolů nebyly dokončeny nebo musely být kvůli různým okolnostem dokončeny v omezeném rozsahu. Tyto skutečnosti by měly být popsány v rámci reportu, včetně jejich odůvodnění.

Tento dokument dále slouží jako zdroj pro tvorbu následných iterací v dané doméně (typicky v rámci Epicu, pokud nebyl iterací uzavřen).

Seznam překážek

Jak bylo zmíněno, v rámci každodenních Standupů by měl vznikat seznam překážek, které mohou ohrozit postup týmu v rámci iterace. Nedílnou součástí reportu iterace by tedy měl být i seznam těchto překážek včetně stavu jejich řešení. Pokud některé z těchto překážek vyřešeny nebyly, měly by podlehnout analýze, zda mohou ohrozit tým v budoucnu a případně definovat akční kroky k jejich odstranění.

4.5.3 Zápisy z pravidelných událostí

Pro všechny události by měl existovat formalizovaný zápis, který by měl být dodržován tak, aby bylo možné správně interpretovat výstupy schůzek a ty mohly být historicky drženy a analyzovány. Jejich forma je popsána v následující tabulce.

Tabulka 12: Formy zápisů z pravidelných událostí

Událost	Forma zápisu
Iteration Planning	Nepotřebuje speciální zápis, jelikož jeho výstup, kterým je připravený Backlog iterace, je dostatečně vypovídající.
Standup	V rámci této metodiky je Standup jen o hledání a řešení překážek, a tudíž jeho zápis je pouze seznamem překážek, který je aktualizován. Agile Coach průběžně mění stavy řešení překážek a při Standupu vždy shrne, jaký je jejich aktuální stav. Následně jsou do seznamu přidány další překážky, které někdo v týmu objevil. Po skončení iterace jsou zbylé překážky přeneseny do reportu a další iterace začíná s novým prázdným seznamem.

Iteration Review	Výstupem z této schůzky je report iterace, respektive podklady pro jeho tvorbu. Vypovídající hodnotu má až tento report, a proto není potřeba formalizovat zápis, který slouží pouze jako vstup k jeho tvorbě.
Iteration Retrospective	Forma retrospektivy není striktně definována, ovšem vždy by jejím výstupem měl být výčet akčních bodů, které je potřeba vykonat pro zvýšení celkové efektivity týmu. Tyto výstupy by měly být zapisovány a historicky drženy, včetně zápisu postupu jejich řešení. Zodpovědnost za udržování těchto dokumentů má Agile Coach.

4.6 Metriky

V tomto oddílu jsou popsány metriky, na základě kterých je možné hodnotit jak výkonnost a efektivitu týmu, tak efektivitu metodiky samotné nebo funkčnost jejich technik. Analýzou těchto metrik by mělo být možné odhalit možné hrozby a překážky, které by mohly negativně ovlivnit výkonnost týmu či výslednou kvalitu produktu.

„Metrika je přesně vymezený finanční nebo nefinanční ukazatel nebo hodnotící kritérium, které jsou používány k hodnocení úrovně efektivnosti či jakosti konkrétní oblasti IS/ICT nebo k hodnocení podnikového výkonu a úrovně jeho podpory prostředky IS/ICT. Přesným vymezením metriky se rozumí definovaný postup, který se použije pro získání hodnoty metriky (metoda měření) a definice způsobu, jakým budou získané hodnoty mezi sebou porovnávány (měřicí stupnice).“ (Novotný, 2003 cit. dle Oškrdala, 2012)

Metriky lze v základní formě rozdělit na **kvantitativní**, které umožňují číselnou reprezentaci, a **kvalitativní**, které využívají nečíselných vyjádření.

Lépe interpretovatelné a čitelnější jsou metriky kvantitativní, jelikož nejsou ovlivněny subjektivností hodnocení a dosahují tedy vždy objektivních hodnot. Jsou vždy opřené o konkrétní data, na základě kterých jsou dále analyzovány, zejména z pohledu vývoje jejich hodnot v čase.

Kvalitativní metriky jsou oproti kvantitativním založené na subjektivním hodnocení a nejsou interpretovatelné na základě číselných dat. Pro jejich měření lze využít binárních hodnot (tj. „ano“ nebo „ne“) nebo výběru stupně na škále (např. známkování ve škole). Pro zvýšení výpovědní hodnoty měření je vhodné rozšířit tyto metriky o kvantitativní formulaci, což znamená, že zařazení do škály je podloženo objektivními hodnotami dat. Příklad známkování ve škole můžeme takto rozšířit o to, že je známka zvolena na základě procentuální úspěšnosti v testu.

Vzhledem k povaze řešené oblasti nejsou čistě kvalitativní metriky považovány za směrodatné, jelikož existence nemalého množství velmi odlišných pohledů může tyto hodnoty značnou mírou zkreslit a snížit tak jejich vypovídající hodnotu. Jinými slovy, nelze určit jednu roli, která by mohla nezávisle určovat hodnoty těchto metrik. Z tohoto důvodu

jsou i kvalitativní metriky rozšířené o kvantitativní formulaci a umožňují tedy reprezentaci na základě dostupných objektivních dat.

Následující metriky jsou rozděleny do kategorií dle jejich zaměření. Jsou formulovány pouze pomocí slovního popisu a jejich výsledná podoba není metodikou striktně vyžadována, jelikož je značně závislá na prostředí a potřebách podniku, ve kterém je metodika implementována. Následující metriky mají tedy spíše povahu doporučení, jak by k této oblasti mělo být přistupováno a je zhruba navržena podoba měření a řízení jejich hodnot.

4.6.1 Výkonnost týmu

Základem pro měření výkonnosti týmu je výpočet hodnot všech forem ukazatele **Story Point Time** (dále jen SPT) značící koeficient pro přepočtení odhadu ve Story Pointech na časovou náročnost v člověkohodinách. Koeficient je více popsán [v sekci 4.4.1](#).

Tento ukazatel by měl mít pro každou z vyvíjených platforem 3 různé hodnoty – pro každou z úrovní rizikovosti (nízká/střední/vysoká). Je nutné oddělit tyto ukazatele pro jednotlivé platformy, jelikož by mohla být zkreslena hodnota rozdílností zastoupení vývojového týmu na každé z platforem (např. pokud by v týmu byly 2 seniorní iOS vývojáři a Android vývojáři by byli v poměru 1 senior a 2 junioři). Zároveň je takto umožněna synchronizace postupu obou platforem a dlouhodobější plánování kapacit vývojového týmu.

Samotné metriky na základě těchto ukazatelů (ve všech formách) by měly zahrnovat

- *poměr celkového průměru a trendové hodnoty pro **analýzu dlouhodobé výkonnosti***
- *a poměr trendové hodnoty a aktuálního průměru (za poslední iteraci) pro **analýzu aktuální výkonnosti**.*

Trendová hodnota označuje průměr za poslední 4 iterace (tzn. minimálně za poslední měsíc).

Analýza dlouhodobé výkonnosti by měla být prováděna pravidelně Agile Coachem a na jejím základě by měly být navrhovány změny procesů a základních technik používaných pro řízení projektu.

Analýza aktuální výkonnosti musí být prováděna na konci každé iterace a představena v rámci události *Iteration Retrospective*, na které by měla být metrika kriticky zhodnocena a měla by vyvolat diskuzi o možných zlepšeních či o validaci změn provedených v předešlém průběhu projektu.

Tato analýza v podstatě koreluje s porovnáváním odhadované náročnosti iterace vůči realitě a tyto hodnoty tedy není potřeba podrobovat žádné další analýze.

Pokud v prvních iteracích projektu hodnoty ukazatelů značně vzrostou, je nutné si uvědomit, že interpretací tohoto vzrůstu není s největší pravděpodobností zhoršující se výkonnost týmu, ale spíše existující problém s tvorbou úvodních hodnot ukazatelů (tedy

bud' špatný výpočet dle složení týmu a historických dat nebo špatný úvodní odhad přepočtu).

4.6.2 Komplettnost analýzy úkolů

Tato kategorie se zabývá spíše identifikací možných problémů s funkčností technik definovaných metodikou, konkrétně technik analýzy úkolů, a tedy práce role Solution Expert. Řešením těchto problémů je validace správného využití těchto technik.

Jednou z indikací nedostatečné analýzy může být zvýšený **výskyt úkolů ve stavech *BLOCKED BY*** (včetně *BLOCKED BY OTHER TASK*), jelikož do těchto stavů jsou úkoly posouvány pouze v případě, že obsahují nedostatek informací, které by měly být získány právě v průběhu analýzy.

Důvodem detailnější analýzy v této metodice je zejména zvýšení kvality a snížení chybovosti aplikace. Zvýšení chybovosti je tedy indikací toho, že metodika či její techniky nejsou dostatečně efektivní. Pro tento kontext je vhodné sledovat metriku **poměru vrácených Stories ze stavu *TESTING***, která pravděpodobně přímo souvisí s tím, že byla během analýzy opomenuta skutečnost, kterou následně tester objevil jako nedostatek implementace. Určité procento je samozřejmě v normě a neměla by být sledována absolutní hodnota této metriky, ale měla by spíše být porovnávána v čase. Hodnota by měla být měřena odděleně pro každou iteraci.

S předchozí metrikou souvisí také **počet vzniklých úkolů typu Bug**, které také napovídají, že analýza nebyla dostatečně detailní. Rozdíl oproti poměru vrácených Stories je v tom, že ani tester daný nedostatek či chybu neobjevil. Stejně je také nutné tuto metriku sledovat v čase a tolerovat určitou hladinu. Doporučuje se sledovat tuto metriku jako trendovou hodnotu za poslední měsíc, jelikož nemusí mít přímý vztah k probíhajícím iteracím.

Metrikou s dlouhodobější formou v této oblasti je **poměr Hotfix iterací vůči iteracím celkově**. Hodnoty této metriky nedává smysl sledovat frekventovaně, ovšem je vhodné několikrát do roka tuto metriku analyzovat, jelikož její vysoká hodnota by značila zásadní problém, který může být spojený nejen s nedostatečnou analýzou, ale také s nedostatečně kvalitním testováním. Pokud jsou všechny ostatní hodnoty metrik této kategorie v běžných hodnotách a nedosahují značných výkyvů, pravděpodobně je problémem spíše způsob a kvalita testování.

4.6.3 Odstraňování překážek

Funkcí role Agile Coach by mělo docházet ke spolehlivějšímu a rychlejšímu odstraňování překážek v průběhu iterace. Poněkud subjektivnější indikací toho, že funkce této role není spolehlivá, může být to, že se **opakují hlášené překážky při pravidelných událostech**. Tato metrika není objektivně měřitelná, jelikož velmi závisí na složitosti řešení daných překážek, ovšem pokud se komukoli z týmu zdá, že nějaké překážky nejsou řešeny a zaznívají pravidelně, měl by tuto skutečnost nadnést na některé z odpovídajících událostí (Standup nebo Iteration Retrospective, v závislosti na povaze překážky).

Objektivní a měřitelnou metrikou v této kategorii je **průměrný čas úkolů ve stavu *BLOCKED BY*** (mimo stav *BLOCKED BY OTHER TASK*). Vyšší hodnota této metriky značí, že jsou překážky v průběhu iterace odbourávány nedostatečnou rychlostí. Absolutní hodnota této metriky je velmi specifická v ohledu na kontext projektu, ovšem je vhodné měřit její výkyvy a analyzovat jejich důvody.

V neposlední řadě je vhodné sledovat také vývoj **délky události *Iteration Retrospective***. Ta by se na základě průběžného řešení překážek měla v průběhu projektu postupně snižovat. Zvyšování její délky je indikací, že jsou překážky odstraňovány pomalu, v nedostatečné kvalitě nebo neorganizovaně (nejsou odstraňovány prioritnější překážky na úkor těch s nižší prioritou). V úvodních fázích projektu se běžná délka pohybuje kolem jedné hodiny, postupně by se měla snižovat až na půl hodinu. Pokud se délka pohybuje pod těmito hranicemi, jsou menší výkyvy v délce normální a neměla by jim být prisuzována vysoká váha.

4.6.4 Příčina nepřímo související s projektem

Následující metrika nesouvisí přímo s projektem a nemůže být odstraněna změnou řízení projektu či procesu vývoje. Jedná se o **poměr úkolů s vysokou nebo střední rizikovostí vůči nízkorizikovým**. Tento poměr je samozřejmě závislý na konkrétním předmětu vývoje a může být nárazově ovlivněn určitou (např. experimentální) částí projektu. Krátkodobý nárůst této hodnoty je tedy normální. Indikací problému je ovšem to, že se tento poměr stabilně drží ve vyšších hodnotách. Tento problém může značit chybějící nebo nedostatečné firemní know-how, což by mělo být reportováno odpovědným osobám mimo samotný projektový tým.

5 Implementace metodiky

V této kapitole je představen subjekt, ve kterém by měla být metodika zavedena. Je analyzován jeho současný stav a jsou vyjmenovány problémy a nedostatky, které v současném stavu má. Nedostatky jsou porovnány s navrhovanou metodikou a je zhodnocena její přínosnost v kontextu těchto nedostatků.

V další části je navržena implementace metodiky v prostředí daného subjektu. Tato implementace je rozvržena do několika kroků, které umožňují postupné zavádění metodiky, a tedy i možnost včasné revalidace a případné úpravy částí metodiky, pokud by po jejich implementaci nebyl zaznamenán očekávaný přínos.

Následně je popsána věcná implementace metodiky, která se týká zejména konfigurace nástrojů pro podporu řízení metodiky. Návodů pro přípravu těchto nástrojů jsou detailně popsány a v poslední části jsou zobrazeny simulace modelových situací projektu s použitím těchto nástrojů.

5.1 Představení subjektu

Vybraným subjektem je startup *Synetech*, který je na trhu již 5 let a věnuje se zakázkovému vývoji mobilních aplikací. Dnes disponuje velikostí zhruba 30 zaměstnanců. I kvůli malé velikosti zde doposud nebylo metodicky řešeno řízení projektů a vývoje a je tudíž vhodným kandidátem k analýze a zavedení metodiky, která je uzpůsobena konkrétnímu zaměření podniku.

5.1.1 Popis prostředí subjektu

Synetech se věnuje hlavně nativnímu vývoji mobilních aplikací a jsou zde tedy dva týmy vývojářů pro každou z mobilních platform. Částečně se *Synetech* věnuje také vývoji webových aplikací a je zde i tým webových vývojářů, ovšem navrhovaná metodika se týká pouze vývoje mobilních aplikací a tento tým tedy pro tuto práci není relevantní. Vedle vývojářských týmů stojí také tým testerů.

Momentálně se také začíná tvořit tým UX/UI designérů, který je ovšem stále v raných fázích. Zároveň je také, v případě nutnosti, možné dedikovat backend vývojáře, který je dostupný v rámci týmu webových vývojářů.

O řízení projektů se stará tým projektových manažerů, kteří mají na starost jak řízení týmů, tak komunikaci s klienty a celkové řízení produktu. Součástí užšího vedení je také oddělení prodeje, marketingu atd., ovšem tato oddělení jsou pro následující analýzu irelevantní.

Podnik se momentálně nachází v situaci nedostatku komerčních projektů, která je způsobena mimo jiné i současnou situací ve světě v důsledku koronavirové krize. Následující analýza tedy odráží poslední standardní stav, kdy existoval dostatek komerčních projektů a

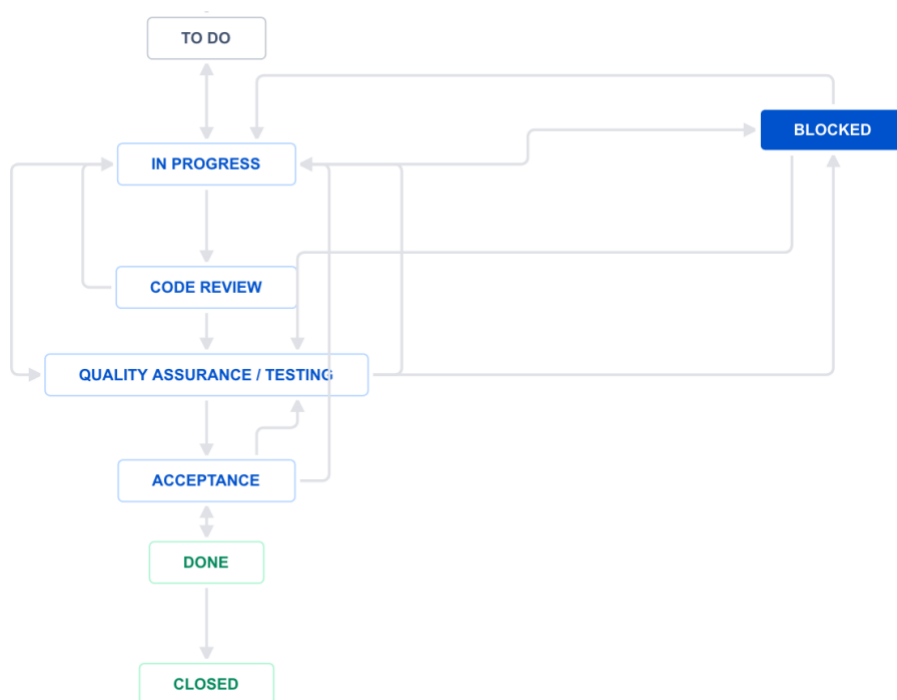
podnik fungoval v běžném režimu. Tato situace je také důvodem, proč samotná metodika není v rámci této práce zavedena a je pouze navržena její implementace. Podnik si aktuálně nemůže dovolit experiment nasazení neověřené metodiky. Interpretace její efektivity by také mohla být zkreslena okolnostmi.

5.1.2 Současné řešení řízení projektů

Pokud se objeví nová zakázka, je vytvořen projektový tým takovým způsobem, že projektový manažer, který zakázku vyjednával, vytvoří vývojový tým z dostupných vývojářů pro obě platformy a k tomu přiřadí, typicky jednoho, testera. V případě nutnosti jsou přiřazeny také další role, ovšem standardně se technická infrastruktura nachází na straně klienta a vizuální návrh aplikace také dodává klient sám, případně je využito externího dodavatele.

Pro řízení projektu se používá některých praktik Scrumu – projekt je dělen do, standardně dvoutýdenních, Sprintů, každému Sprintu předchází plánování a po jeho skončení se koná retrospektiva (není úplně pravidlem, někdy se k retrospektivě přistupuje až po skončení projektu). Několikrát do týdne je také konán Standup, který je pravidelně plánován, ovšem jeho frekvence se liší podle konkrétních projektů. Na konci každého Sprintu jsou nové iterace aplikací otestovány a předány zákazníkovi. Způsob vydávání nových verzí aplikací se liší dle klienta.

Tým i produkt řídí jen jedna role projektového manažera, která se stará o veškerou komunikaci v týmu. Pro tvorbu úkolů jsou používány Stories, které jsou zařazeny pod určité Epicy. Jejich řízení je realizováno pomocí běžných Scrumových artefaktů (Produktový backlog a Sprint backlog). Pro podporu řízení úkolů je standardně používán nástroj Jira, ve kterém je vytvořen Kanban board odrážející jednoduché workflow úkolů (bez ohledu na jejich typ), které je zobrazeno na obrázku 12.



Obrázek 12: Současné workflow úkolů v Synetechu.

Efektivita týmu a jejich členů není nijak řízena a měřena. Pracnost úkolů je odhadována jen pomocí člověkohodin potřebných k jeho dokončení. Tyto odhady jsou přiřazovány během plánování Sprintu, jehož obsah je následně naplněn dle dostupných časových kapacit. Úspěšnost Sprintu je založena na kompletizaci všech úkolů, kterými byl naplněn Sprint backlog, a výsledně tedy odráží spíše přesnost odhadů než výkonnost týmu. Plánování Sprintu také nezahrnuje žádnou analýzu Stories, což může způsobovat zkreslení odhadů náročnosti.

Shrnutí předchozího Sprintu z pohledu posunu projektu se odehrává v první části plánování a zahrnuje v podstatě jen odsunutí nedokončených úkolů do Sprintu následujícího. Ovšem co se týče samotných aktivit na projektu, ty jsou validovány v rámci retrospektivy, během které jednotliví členové týmu vznášejí návrhy na zlepšení procesů, praktik a technik pro zvýšení efektivity týmu. Tyto podněty jsou následně přetransformovány v akční body, ke kterým jsou přiřazeny odpovědné osoby. Ovšem běžně se stává, že tyto body nejsou vyřešeny a na retrospektivách se tedy opakují.

Pravidelný Standup probíhá formou kolečka, kdy každý z členů týmu říká, čemu se věnoval od posledního Standupu a čemu se bude věnovat dále. Pokud si je vědom nějaké překážky, kterou je potřeba odstranit, je tato skutečnost také vznesena. Vyřešení těchto překážek ovšem občas vázne a často se stává, že stejné překážky zaznívají opakovaně.

5.2 Analýza subjektu

V tomto oddílu je analyzováno současné řešení řízení vývoje v prostředí subjektu představeného v předchozím oddílu. Jsou vyjmenovány nedostatky a navrženo řešení těchto nedostatků pomocí navrhované metodiky.

Jelikož kompletní zavedení nové metodiky je náročné a je velmi těžko aplikovatelné například na již probíhající projekty, je navržen postup zavedení jednotlivých kroků, které mohou alespoň částečně pomoci zlepšit současný stav a připravit podnik na jednodušší přechod na novou metodiku. Pro průběžné ověřování efektivity metodiky jsou také navrženy metriky, pomocí kterých je možné posun vpřed analyzovat a případně včas zarazit a revalidovat postup zavádění.

5.2.1 Problémy a nedostatky současného řešení

Nejočividnějším problémem je **absence formalizované metodiky**, která implikuje to, že se řízení může zásadně lišit mezi jednotlivými projekty. Tato nekonzistence může způsobovat výkyvy ve výkonnostech jednotlivých týmů a může velkou měrou ztěžovat případný přechod členů týmů mezi projekty, o to spíše pokud by se jednalo o samotnou roli projektového manažera.

Za tohoto stavu je také v podstatě nemožné zavádět jakékoli metriky pro měření výkonnosti a efektivity týmů, jelikož by nebyly mezi jednotlivými týmy porovnatelné a mohly by porovnávat pouze jeden tým na jednom projektu v čase – a i to je s otazníkem, protože je možné, že se v průběhu projektu změní část řízení, která může hodnoty metrik zkreslit.

Dalším nedostatkem je **absence analýzy úkolů**, která stěžuje práci členů týmů, zejména vývojářů, a narušuje možnost jejich plného soustředění na svou vlastní činnost. Jak bylo zmíněno již v předchozích kapitolách, úkoly si v prostředí mobilních aplikací z více důvodů žádají důkladnější analýzu, která se ovšem za současného stavu neděje.

Viditelným problémem jsou také **nejasné zodpovědnosti rolí**, což je způsobeno zejména jednoduchým obecným workflow. Pokud takové workflow existuje, mělo by v každém jeho stavu být jasné, kdo je zodpovědný za vyřešení tohoto stavu. Některé ze stavů jsou příliš abstraktní a mohou odrážet různé situace, což znamená, že mohou odrážet také potřebu jiné role pro vyřešení.

Tento problém také úzce souvisí s **rolí projektového manažera**, která je velmi abstraktní a pokrývá přespříliš velké množství činností, což může mít za důsledek to, že jsou některé upozaděny na úkor těch, které mají momentálně vyšší prioritu. Tato skutečnost může způsobovat to, že je jednání s klientem upřednostňováno před řízením týmu samotného.

Abstraktní role projektového manažera pravděpodobně souvisí s **absencí řízení efektivity týmu**, což je velmi zásadní nedostatek. Tím, že není vymezena role, která by výkonost týmu řídila a neexistuje žádná metrika, která by dokázala výkonost týmu analyzovat a hodnotit, je nemožné tuto efektivitu zvyšovat. Výkonost týmu je za tohoto stavu tvořena pouze individuální efektivitou jednotlivých členů týmů.

Efektivitu týmu ovšem také není možné řídit za aktuálního stavu odhadování pracovní pomoci člověkohodin, které nijak nereflektují finálního řešitele úkolu (který může být na jiné úrovni než ten, který udával odhad). Tato skutečnost odhaluje další nedostatek – **chybějící variace přístupu dle schopnosti jednotlivců**. Pokud by byla výkonost jednotlivce řízena stejným způsobem bez ohledu na jeho schopnosti, byly by výsledky zavádějící a pravděpodobně i demotivující pro daného jednotlivce. Tato skutečnost musí být v řízení odražena a odhad pracovní pomoci musí být objektivní.

V neposlední řadě **schází řešení překážek v průběhu Sprintu**. Bylo zmíněno, že překážky či náměty na zlepšení se opakují, což by se stávat nemělo a zejména překážky, které se objeví v průběhu Sprintu musí být vyřešeny okamžitě.

5.2.2 Řešení problémů a nedostatků pomocí metodiky

Absenci formalizované metodiky lze samozřejmě vyřešit zavedením této metodiky. Opomenout se ovšem nesmí také to, že musí být v rámci zavádění připraveny nástroje pro podporu řízení a do podnikových směrnic či dokumentace musí být zavedeny také návody, postupy a vzory, díky kterým bude možné metodiku užívat konzistentně v jejím plném rozsahu a bez nutnosti dalších zaškolování.

Zavedením rolí tak, jak jsou definované metodikou, lze vyřešit hned několik problémů. **Role projektového manažera** již nebude tak přetížena, jelikož jeho zodpovědnosti budou rozděleny do rolí *Product Owner* a *Agile Coach*.

S rolí *Agile Coach* také úzce souvisí řešení pro **nedostatečné řešení překážek v průběhu Sprintu**. Odpovědností této role je postarat se o to, aby překážky byly průběžně odstraňovány a tým se tak mohl posouvat dále. To se samozřejmě týká i řešení námětů pro zlepšení fungování týmu.

Agile Coach se také stará o **vnitřní efektivitu týmu**, a tedy i o její **řízení**, což bylo také vyčteno mezi nedostatky aktuálního řešení. K tomu je potřeba zavést další část metodiky, která je v současném řešení opomíjena - metriky a jejich řízení. S těmito nástroji může *Agile Coach* řídit výkonnost týmu a pomáhat k jejímu navyšování.

Další role, která by byla oproti současnému stavu novinkou, je *Solution Expert*, který společně se zavedením strukturalizace úkolů a zvýšení komplexnosti jejich workflow, dokáže jednoduše vyřešit **absenci analýzy úkolů**.

Strukturalizace a workflow úkolů je zároveň řešením **nejasné zodpovědnosti rolí**, která se díky komplexnosti workflow stane mnohem jasnější a všechny stavy, kterých mohou úkoly dosáhnout, budou mít jasného řešitele.

V neposlední řadě je nutné opustit odhady pracnosti v člověkohodinách a zavést odhady pomocí Story Pointů. Díky nim, nejenže bude možné lépe řídit efektivitu díky mnohem více vypovídajícím metrikám, ale také bude možné vytvořit **variaci přístupu dle schopnosti jednotlivců**. Během odhadu pracnosti tedy nebude záležet na tom, kdo bude finálně úkol zpracovávat, ale pouze na objemu práce potřebnému k vyřešení úkolu. Výslednou časovou náročnost bude možné dopočítat dle schopností a výkonnosti řešitele.

5.2.3 Kroky zavedení metodiky

Jak již bylo zmíněno v úvodu oddílu, přechod na novou metodiku je náročný a pro jeho zjednodušení je vhodnější postupně zavádět jeho části a průběžně ověřovat jejich přínosnost a správnost nasazení. Zavedení je tedy rozděleno do tří kroků, které jsou stručně popsány v následující tabulce.

Tabulka 13: Stručný popis kroků zavedení metodiky v prostředí analyzovaného subjektu

Krok	Popis
1	Úvodní krok se týká zejména zavedení rolí a definovaných typů úkolů, včetně jejich workflow. V tomto kroku by mělo být odstraněno nejvíce vytyčených nedostatků bez potřeby zásadních změn myšlení.
2	Druhý krok je zaměřen na řízení výkonnosti, s čímž je spojena změna způsobu odhadování náročnosti z časové na relativní. V rámci tohoto kroku je tedy zaveden odhad pomocí Story Pointů a přiřazování stupňů rizikovitosti úkolů.
3	Poslední krok obsahuje organizačně nejnáročnější část, což je rozdělení projektu v čase za pomoci rozsahem ohraničených iterací.

K zavedení dalšího kroku by se mělo vždy přistoupit až ve chvíli, kdy je předchozí krok implementován v plném rozsahu a je ověřeno, že je zaveden správně a přináší očekávané výsledky.

V jednotlivých krocích není zmiňováno zavádění odpovídajících **dokumentů a ustanovení**, jelikož nevyžadují změnu fungování průběhu projektu. Ty, které by měly vzniknout před začátkem projektu by měly být zavedeny se startem nového projektu, případně s dokončením plné implementace metodiky (pro stávající projekty). Ostatní přímo souvisí se zaváděním technik metodiky a je tedy považováno za samozřejmost, že zavedení techniky je doprovázeno zavedením odpovídající dokumentace – např. zavedením iterací se předpokládá, že bude zaveden také vznik reportu z iterací.

První krok

V prvním kroku by se mělo co možná nejméně zasahovat do běžných zvyklostí, a tudíž se tedy vůbec netýká rozdělení projektu v čase. Změna odhadů pracnosti si vyžaduje určitou změnu myšlení, a také pro tento krok není vhodná. Zároveň je žádoucí, aby se ihned prvním krokem odstranilo co nejvíce nalezených problémů a nedostatků.

V rámci něj tedy navrhuji zavedení **rolí a strukturalizace úkolů a jejich workflow**.

Co se týče rolí, nebylo by v této fázi vhodné vynucovat přítomnost nových členů v týmu, a proto navrhuji pouze formální ustanovení rolí a jejich zodpovědností. Role projektového manažera by se tedy měla rozdělit na roli *Product Owner* a *Agile Coach*, jejichž zástupci by si měli být vědomi všech zodpovědností, které roli provází. Obě tyto role může zastávat osoba, která doteď zastávala roli projektového manažera, případně může být role *Agile Coache* delegována na dostatečně kompetentního člena vývojového týmu.

Role *Solution Experta*, resp. *Solution Expertů*, by měla být přiřazena nejzkušenějším členům vývojového týmu. Tyto osoby by měly být obeznámeny s povinnostmi této role a měly by jim být vyhrazeny kapacity pro vykonávání těchto povinností.

Další bodem je zavedení strukturalizace úkolů a jejich workflow, které se do důsledku týká pouze upravení podpůrného nástroje pro podporu řízení (v tomto případě Jira) a seznámení všech členů týmu s touto skutečností. Workflow, včetně definovaných pohledů na workflow, je dostatečně deskriptivní pro to, aby bylo pro jednotlivé role samo o sobě jednoduše pochopitelné. Je ovšem žádoucí, aby popis typů úkolů a jejich workflow byl zařazeno do podnikové dokumentace, aby si případně mohl každý člen týmu ověřit, zda s novým prostředím nakládá správně. Pomoci by v tomto směru měly manuály rolí pro práci v metodice, které jsou k dispozici v přílohách B-G.

Tento krok není provázán se zavedením metrik, a tudíž nelze kvantitativně měřit jeho úspěšnost. Hodnocení úspěšnosti tohoto kroku je tedy spíše subjektivní a mělo by vycházet z individuálního vnímání jednotlivých členů týmů. Indikací správného zavedení by měly být zejména tyto skutečnosti: minimalizace odbíhání od kontextu, možnost větší koncentrace na vykonávání své činnosti a rychlejší odbourávání překážek.

Druhý krok

V druhém kroku je nutné se zaměřit na **řízení výkonnosti**. Z toho důvodu je potřeba zavést odhadování pracnosti pomocí Story Pointů. S tím souvisí zejména změna myšlení při odhadech tak, aby se staly opravdu objektivními. Tomuto by mělo napomoci vytvoření referenční tabulky, která obsahuje dobře představitelné úkoly společně s jejich odhady pracnosti. Na základě této tabulky bude jednodušší odvodit pracnost jiných úkolů. Pro odhady doporučuji využití delfské metody¹⁴.

Společně s odhady pomocí relativní metriky je potřeba také zavést hodnocení rizikovitosti úkolů, které pomůže s přepočtem Story Pointů na reálnou časovou náročnost úkolů. Rizikovitost by vždy měla být přiřazena až po přidání odhadu, aby subjektivně nezkracovala jeho hodnotu.

Odhadováním pomocí Story Pointů a hodnocením rizik se otevírá možnost zavedení metrik, které by měly být měřeny ihned s nasazením nové formy odhadů. Jejich analýza bude v prvních fázích nevypovídající, jelikož nabyté hodnoty nebude s čím srovnávat, ovšem v průběhu času se stanou již mnohem více vypovídajícími.

V rámci tohoto kroku je doporučeno analyzovat fungování rolí zavedených v předchozím kroku a případně obměnit osoby, které role zastávají. Pokud byl v prvním kroku ustanoven *Agile Coach* člen vývojového týmu, musí být v tomto kroku nahrazen nezávislou osobou tak, jak je vyžadováno metodikou.

Úspěšnost zavedení kroku nelze měřit dle absolutních hodnot metrik, ovšem úspěšným nasazením by mělo být možné ve vývoji hodnot metrik vidět trendy, které opravdu odrážejí realitu. Částečně to tedy opět odráží subjektivní hodnocení, ovšem tentokrát je toto subjektivní hodnocení porovnáváno s objektivními hodnotami. Pokud hodnoty neodpovídají subjektivním hodnocením, pak nelze považovat tento krok za splněný.

Třetí krok

Třetím a posledním krokem je již zavedení metodiky v plném rozsahu. Poslední zbývající částí je **rozdělení projektu v čase**. Vzhledem k tomu, že zavedení rozsahem ohraničených iterací vyžaduje také jejich projekci do budoucnosti a odhady ve Story Pointech s relevantními přepočty na časové kapacity, doporučuji začít s přípravou tohoto kroku minimálně měsíc před samotným zavedením.

V této fázi by již projekt měl být kompletně připraven na zavedení iterací na základě objemu práce a měl by mít již zavedeny všechny metriky, které jsou potřebné k tomu, aby projekce byla velmi blízko reálnému průběhu. Ovšem je nutné tento krok neuspěchat z důvodu získání více objektivních hodnot metrik pro reálnější hodnoty přepočtu odhadů na časovou náročnost.

¹⁴ Expertní metoda hledání řešení. Je založena na tom, že každá osoba запиše svůj odhad a nestranná osoba poté všem sdělí průměr všech odhadů. Tento postup se opakuje, dokud nedojde k alespoň přibližné shodě.

Úspěšnost tohoto kroku lze již objektivně měřit pomocí zavedených metrik. Měření účinnosti metodiky pomocí těchto metrik je více popsáno v následující sekci.

5.2.4 Metriky pro ověření účinnosti zavedení metodiky

Nejdůležitější metrikou pro ověření účinnosti je samozřejmě výkonnost týmu. Jelikož absolutní hodnotu není možné porovnávat s předchozím stavem, je nutné sledovat vývoj této metriky v čase, v tomto případě se jedná zejména o měření **dlouhodobé výkonnosti**, která by měla mít **rostoucí tendenci**. Aktuální výkonnost může být v tomto případě zavádějící, jelikož může být ovlivněna nestandardními okolnostmi.

Pokud je možné zpětně získat data o **poměru vrácených Stories ze stavu TESTING** (resp. ze všech typů úkolů a testování úkolů obecně) a o **počtu vzniklých úkolů typu Bug**, které budou dostatečně relevantní, je možné měřit tyto hodnoty s hodnotami aktuálními, které musí být značně nižší. Toto porovnání je podmíněno ověřením správné implementace metodiky, které může být zajištěno analýzou všech metrik této kategorie (zejména jejich vývoje v čase) tak, jak je popsáno metodikou. Pokud jmenované metriky dosahují hodnot podobných nebo vyšších a historická data jsou dostatečně objektivní, lze hodnotit metodiku jako méně efektivní v oblasti kvality a je vhodné vyvolat její přezkoumání.

Vzhledem k tomu, že v současném stavu je odstraňování překážek značným problémem, lze považovat jakýkoli pozitivní trend v metrikách souvisejících s odstraňováním překážek (popsaných v [sekci 4.6.3](#)) za ověření přínosnosti zavedení metodiky.

5.3 Konfigurace nástrojů pro podporu řízení metodiky

Tento oddíl popisuje konfiguraci nástrojů, tvorbu vzorů a dalších náležitostí pro implementaci navržené metodiky v prostředí analyzovaného subjektu. V předchozím oddílu byl navržen postup zavedení v jednotlivých krocích, ovšem zde je popsán pouze finální stav, který by měl nastat po dokončení posledního kroku. Jelikož jednotlivé kroky jsou rozděleny pouze na postupné přejímání technik, bylo by zbytečné i tento oddíl rozdělovat do těchto kroků a mohlo by to naopak způsobit to, že by čtenáři uteklo logické propojení jednotlivých technik.

Po představení celkové implementace metodiky z věcné stránky je, pro lepší představu pochopení fungování nových procesů, ukázáno několik simulací modelových situací včetně popisu jejich průběhu a vizualizace pomocí použitých nástrojů.

5.3.1 Nástroje

Analyzovaný subjekt již používá nástroj pro podporu řízení a pro snadnější přechod je na místě využít pro implementaci tento nástroj, pokud je to možné. V této sekci je tedy popsán postup implementací v prostředí nástrojů pro podporu řízení tak, aby byly splněny všechny náležitosti a minimalizovala se manuální administrativa, která by jinak se zavedením komplexnější metodiky byla spojena.

Pro možnost převodu odhadů pracnosti na čas je také potřebný nástroj pro sledování času stráveného nad určitým úkolem. Analyzovaný subjekt již používá pro tyto účely nástroj zvaný *Toggl*, který je funkčně implementován včetně integrace s nástrojem pro podporu řízení. Zavádění tohoto nástroje je tedy v této sekci vynecháno.

Stejný případ je nástroj pro správu testovacích případů a scénářů. Pro tyto účely má subjekt zaveden nástroj *TestRail*.

Pro konfigurace je primárně využíván anglický jazyk, aby všechny nástroje byly připraveny na použití pro mezinárodní nebo zahraniční projekty, případně aby byly připraveny na zapojení člena týmu nemluvícího českým jazykem.

Veškeré vzory a konfigurace, které bylo možné exportovat, jsou k dispozici ve veřejném repozitáři této práce, jehož odkaz lze nalézt v [příloze A](#).

Nástroj pro podporu řízení – Jira

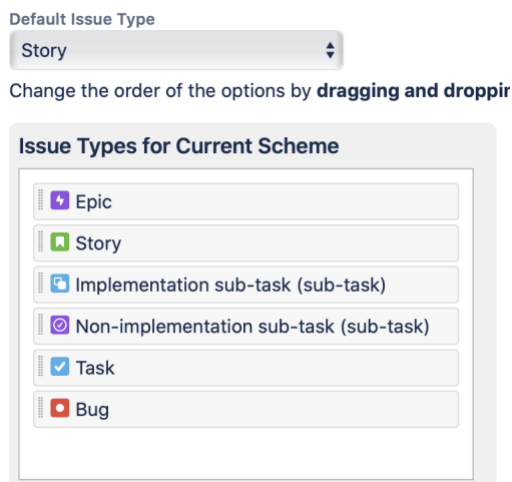
Nástroj Jira bohužel neumožňuje export celých konfigurací, a je proto potřeba prakticky vše nastavit ručně. Jediná možnost exportu je pro workflow, nicméně jedná se pouze o zlomek celé konfigurace. Následuje tedy návod popisující jak nakonfigurovat nástroj Jira, aby v něm bylo možné používat navrženou metodiku.

Většinu konfigurací je naštěstí možné nastavit globálně a není tedy potřeba následující kroky opakovat pro každý projekt. Jedinou výjimkou jsou Boardy, které musejí být vytvořeny a nakonfigurovány na každém projektu zvlášť.

Pro několik prvních kroků je potřeba se pohybovat v nastavení *Issues* (tímto termínem označuje Jira obecně jakýkoli úkol). Do tohoto nastavení se lze dostat kliknutím na ikonu nastavení (v pravém horním rohu) a výběrem položky „Issues“.

V první řadě je potřeba definovat typy úkolů. Základní výběr obsahuje skoro všechny potřebné typy a je tedy nutné provést pouze dvě změny. Je potřeba změnit typ „Sub-task“ na „Implementation sub-task“ a přidat nový typ úkolu s názvem „Non-implementation sub-task“. Je důležité zatrhnout u typu úkolu „Sub-Task Issue Type“, aby se úkol opravdu choval jako sub-task.

Dále je potřeba přiřadit všechny potřebné typy úkolu ke schématu úkolů. V případě, že v konfiguraci nejsou jiná schémata, jsou všechny typy automaticky přiřazeny výchozímu schématu. Doporučuji také nastavení výchozího typu úkolu na Story.



Obrázek 13: Konfigurace typu úkolů v nástroji Jira

V dalším kroku je potřeba nastavit položky na kartách typů úkolů. Primárně je využíváno standardních položek, které jsou nabízeny nástrojem. Oproti standardním položkám je ovšem potřeba vytvořit několik vlastních tak, aby bylo možné přidat na karty úkolů všechny náležitosti definované metodikou.

Je potřeba vytvořit 6 položek, které není možné pokrýt standardními položkami, které Jira nabízí. Jedná se o *akceptační kritéria*, *prostředí chyby* (popis prostředí, ve kterém byla nalezena chyba), *rizikovitost*, *doména* (oblast zaměření) *sub-tasku*, *typ sub-tasku* (myšleno rozdělení na standardní, změnový a chybový) a *prostředí pro testování* (číslo verze, link na stažení nebo podobný zdroj aplikace, na které může být implementace úkolu otestována).

Acceptance Criteria MADM: Acceptance Criteria in form of GIVEN - WHEN - THEN.	Text Field (multi-line)
Bug Environment MADM: Environment in which the bug was found (e.g. app version, tested device ...)	Text Field (multi-line)
Risk MADM: It represents the possible risk of lack of knowledge with solving this type ...	Select List (single choice)
Sub-task domain MADM: Domain of non-implementation sub-task, e.g. UX research.	Select List (single choice)
Sub-task type MADM: Reflects whether sub-task was created initially, added as change request...	Select List (single choice)
Testing Environment MADM: Application version or build number, link for download or any other form ...	Text Field (single line)

Obrázek 14: Vlastní položky karet úkolů vytvořené v nástroji Jira¹⁵

¹⁵ Překlad: Acceptance Criteria = akceptační kritéria; Text Field = textové pole; multi-line = víceřádkový; Bug Environment = prostředí chyby; Risk = rizikovitost; Select List (single-choice) = výběrový seznam s výběrem jedné možnosti; Sub-task domain = doména / oblast zaměření sub-tasku; Sub-task type = typ sub-tasku; Testing Environment = testovací prostředí; single line = jednořádkový. Popisy položek není potřeba překládat, jelikož pouze parafrázuji popis, který je součástí návrhu metodiky.

Akceptační kritéria, prostředí chyby i testovací prostředí mají přiřazené obyčejné textové pole, jelikož mohou dosahovat různých forem a rozsahu a svazování striktní formou může být spíše na škodu. *Rizikovost* a *typ sub-tasku* mají pevný výčet možností, ze které jen jedna může být správná, a proto mají přiřazen seznam s možností výběru jedné varianty. Stejnou formu má i položka *doména sub-tasku*, u které je předpokládáno, že je možné v kontextu možností a schopností subjektu vyčíst všechny možné hodnoty této položky. A opět, jen jedna varianta by měla odpovídat jednomu úkolu (pokud by se hodilo více variant, je to možné brát jako indikaci toho, že by mělo vzniknout i více úkolů).

Pro parametr *platforma* není vytvořena vlastní položka, ovšem je pro něj použita standardní položka s názvem „Component“ (komponenta). Tato položka poskytuje lepší možnosti filtrování a seskupování úkolů a v podstatě v daném kontextu kopíruje sémantický význam tohoto parametru.

V prostředí nástroje Jira také není možné přejmenovat časově ohraničené úseky na iterace a musí tedy být pracováno s pojmem Sprint. Tuto položku je nutné přiřadit všem úkolům typu Story, Task a Bug.

Ted' už jen stačí přiřadit odpovídající položky k jednotlivým typům úkolů. V nástroji Jira jsou karty úkolů pojmenovány jako Screens. Všechny typy úkolů s přiřazenými položkami jsou zobrazeny v následující tabulce.

Tabulka 14: Seznam položek karet pro jednotlivé typy úkolů v nástroji Jira¹⁶

Epic	Story	Impl. sub-task	Neimpl. sub-task	Task	Bug
Epic Name	Summary	Summary	Summary	Summary	Summary
Summary	Issue Type	Issue Type	Issue Type	Issue Type	Issue Type
Issue Type	Description	Description	Description	Description	Description
Description	Reporter	Reporter	Reporter	Reporter	Reporter
Priority	Assignee	Assignee	Assignee	Assignee	Assignee
Linked Issues	Components	Components	Components	Components	Components
	Risk	Risk	Risk	Risk	Risk
	Priority	Priority	Priority	Priority	Priority







¹⁶ Překlad: Epic Name = název Epicu; Summary = shrnutí (v prostředí nástroje lze reprezentovat jako název úkolu); Issue Type = typ úkolu; Description = popis; Priority = priorita; Linked Issues = Související úkoly; Reporter = Zadavatel; Assignee = Řešitel; Components = komponenty (v tomto kontextu se jedná o platformy); Epic Link = odkaz na Epic; Epic Color = barva Epicu; Attachment = příloha; Fix versions = fixní čísla verzí; Story point estimate = odhad ve Story Pointech

	Linked Issues	Attachment	Attachment	Linked Issues	Linked Issues
	Epic Link	Linked Issues	Linked Issues	Epic Link	Epic Link
	Epic Name	Sub-task type	Sub-task domain	Epic Name	Epic Name
	Epic Color			Epic Color	Epic Color
	Acceptance Criteria			Attachment	Attachment
	Attachment			Fix versions	Fix versions
	Fix versions			Story point estimate	Story point estimate
	Story point estimate			Testing Environment	Bug Environment
	Testing Environment			Sprint	Testing Environment
	Sprint				Sprint

Jelikož nástroj pracuje jinak s příslušností k Epicům, je pro odpovídající typy úkolů (které mají přímý vztah k Epicu) nutné přiřadit také 3 položky, které označují odkaz, název a barvu Epicu. Příslušnost k Epicu je následně velmi jednoduše identifikovatelná.

Jednotlivé karty musí mít asociovány také svá schémata, takže je potřeba pro každou z vytvořených karet vytvořit také schéma (zvaná Screen schemes), které má přiřazenou danou kartu jako výchozí.

Po konfiguraci všech potřebných karet úkolů a jejich schémat je potřeba vytvořit schéma, v rámci kterého jsou karty přiřazeny k typům úkolů. V sekci „Issue type screen schemes“ je možné vytvořit nové schéma. Po vytvoření je potřeba přidat konkrétní typy úkolů a k nim přiřadit odpovídající Screen scheme. Konfigurace schématu by následně měla vypadat stejně jako je zobrazeno na následujícím obrázku.

Issue Type	Screen Scheme
Default Used for all unmapped issue types.	MADM: Task
 Epic	MADM: Epic
 Task	MADM: Task
 Story	MADM: Story
 Bug	MADM: Bug
 Implementation sub-task	MADM: Implementation sub-task
 Non-implementation sub-task	MADM: Non-implementation sub-task

Obrázek 15: Konfigurace schématu typů úkolů a odpovídajících karet v nástroji Jira

Pro nastavení správné funkce úkolů již zbývá jen připravit jejich workflow. Tato sekce je strukturována podobně, jako předešlé – je nutné nakonfigurovat jednotlivé workflow a poté vytvořit schéma, ve kterém jsou workflow přiřazeny ke konkrétním typům úkolů.

Workflows jsou podrobně graficky znázorněny v oddílu 4.2 (včetně všech možných přechodů mezi stavy), a tudíž není potřeba zde procházet postup jejich tvorby. Je ovšem nutné nezapomenout na to, že u všech typů úkolů musí existovat stav *CLOSED*, do kterého je možné se dostat ze všech ostatních stavů, ač není součástí žádného z diagramů.








Workflow je také jediná položka, kterou je možné z Jiry vyexportovat, a je tedy k dispozici v [repozitáři práce](#).

Jakmile jsou konfigurovány, případně importovány, všechna workflows, stačí jen vytvořit schéma s přiřazením typů úkolů, které by mělo vypadat tak, jak je zobrazeno na následujícím obrázku.

MADM: Workflow scheme

Workflow scheme for Mobile App Development Methodology

Add Workflow ▾

Workflow	Issue Types
MADM: Epic View as: Text Diagram	 Epic ×
MADM: Implementation sub-task View as: Text Diagram	 Implementation sub-task ×
MADM: Non-implementation sub-task View as: Text Diagram	 Non-implementation sub-task ×
MADM: Story View as: Text Diagram	 Story ×
MADM: Task/Bug View as: Text Diagram	 All Unassigned Issue Types  Bug ×  Task ×

Obrázek 16: Schéma workflows pro typy úkolů v nástroji Jira

Na globální úrovni je již vše nastaveno a nyní je potřeba přiřadit konkrétní konfigurace přímo k projektu. Stačí otevřít seznam projektů a u daného projektu otevřít jeho nastavení (po kliknutí na tři tečky v odpovídajícím řádku napravo). Následně je potřeba přiřadit vytvořená schémata v sekcích *Issue types*, *Workflows* a *Screens*.

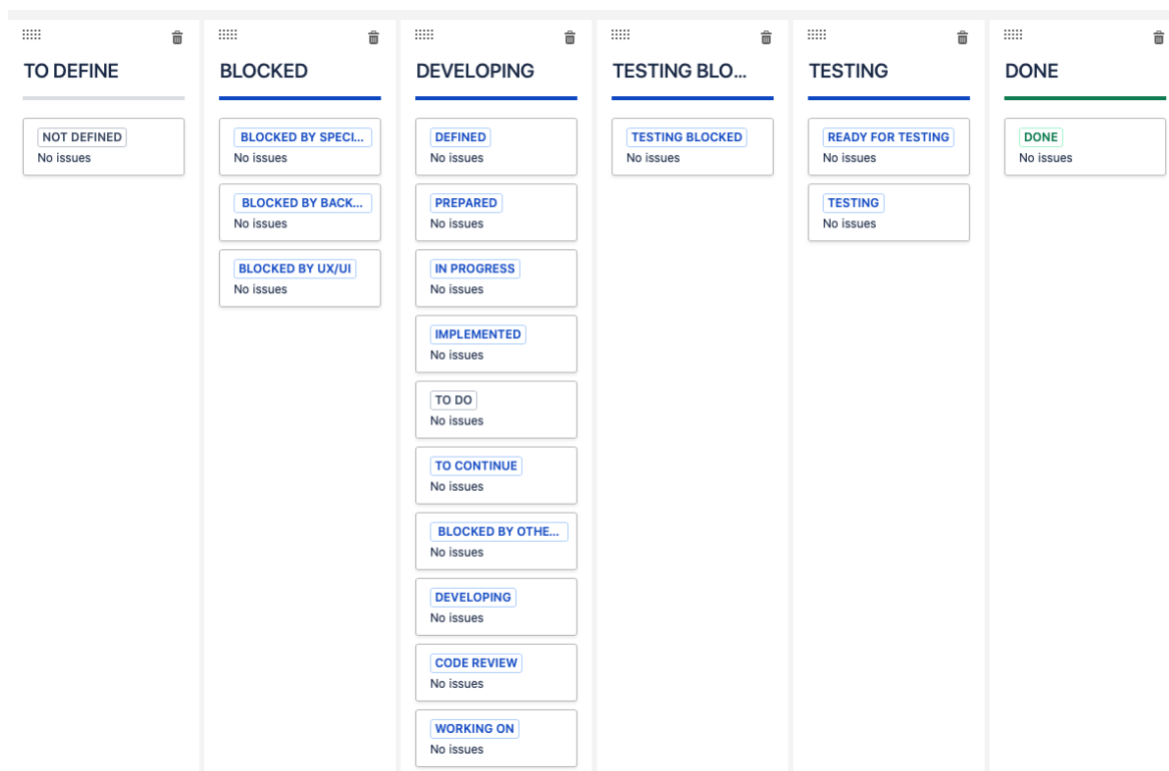
K přípravě základní funkcionality nástroje pro podporu řízení je už třeba jen vytvořit Boardy pro jednotlivé role v týmu, které představují jejich pohledy na workflow. Tuto část nelze konfigurovat na globální úrovni a musí tedy být vytvořena pro každý projekt zvlášť.

Board lze vytvořit kliknutím na druhou položku v levém sloupci otevřeného projektu (tato položka by měla obsahovat název aktuálně otevřeného Boardu), což vyvolá otevření listu Boardů, na konci kterého je položka „Create board“, pomocí které lze vytvořit nový Board. Následně je potřeba vybrat „Scrum board“ a zaškrtnout volbu „Board from an existing project“, která zajistí, že bude Board vytvořen v již existujícím projektu. Pak stačí už jen Board pojmenovat a vybrat ze seznamu aktuální projekt.

Jedinou výjimkou je Board pro roli Solution Expert, která potřebuje ke své činnosti „Kanban board“, jelikož musí přistupovat k úkolům i před započatím jejich Iterace.

Pro možnost využití technik metodiky je potřeba vytvořit 4-6 Boardů (v závislosti na rozsahu vývoje – záleží na tom, zda projekt zahrnuje i UX/UI návrh nebo vývoj Backendu). U každého z Boardů je poté nutné nakonfigurovat odpovídající sloupce tak, jak jsou popsány v [oddílu 4.3](#). K nastavení Boardu se lze dostat otevřením příslušného Boardu, kliknutím na tři tečky v pravém horním rohu a výběrem „Board settings“. Přímo k nastavení sloupců se poté lze dostat výběrem položky „Columns“ v levém sloupci.

Sloupce a odpovídající stavy úkolů jsou dostatečně deskriptivně zobrazeny v [oddílu 4.3](#), a pro představu je tedy na následujícím obrázku zobrazen pouze jeden příklad, který je zároveň tím nejsložitějším, a to je pohled pro role Product Owner a Agile Coach.



Obrázek 17: Konfigurace sloupců Boardu pro role Product Owner a Agile Coach v nástroji Jira

Vzhledem k tomu, že názvy jednotlivých stavů nejsou polysémantické, není nutné rozlišovat mezi typy úkolů a ve sloupcích stačí pouze přiřadit stavy bez ohledu na jejich příslušnost k určitým workflows.

Pro zpřístupnění práce s úkoly na základě jejich typů a vlastností je ovšem potřeba nastavit ještě několik detailů pro jednotlivé Boardy.

Všechny Boardy by určitě měly obsahovat filtr pro oddělení jednotlivých platform. Tyto filtry je možné přidat v sekci „Quick Filters“ v nastavení Boardu. Na následující obrázku je příklad filtrů pro jednotlivé mobilní platformy.



Obrázek 18: Příklad filtrů pro zjednodušení práce s Boardy v nástroji Jira

Pro Board role Solution Expert je vhodné průběžně přidávat tzv. „Swimlanes“, které horizontálně rozdělují Board. Je tak možné oddělit úkoly pro jednotlivé iterace a zajistit tak, že se nebudou míchat úkoly z různých iterací. Příklad konfigurace „Swimlanes“ lze vidět na následující obrázku (iterace je nutné odkázat pomocí jejich identifikátoru).

Přihlášení	linkedIssue in (DP-3)
Registrace	linkedIssue in (DP-4)

Obrázek 19: Příklad konfigurace „Swimlanes“ v nástroji Jira

Základní konfigurace nástroje pro podporu řízení by mělo být tímto připraveno a je možné přejít k pokročilejším částem.

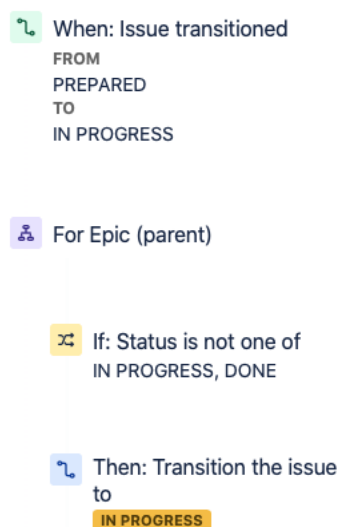
Automatizace v nástroji Jira

Některé z operací, které by se v procesu měly dít, jsou již z principu automatizovatelné a absence jejich automatizace by mohla způsobit určitou nekonzistenci v odpovídajících stavech úkolů. Jedná se zejména o vztahy dítě-rodíč, kdy se úkol řídí stavem jeho podúkolů.

Nástroj Jira sám o sobě nabízí možnosti automatizace. Globálně je lze nakonfigurovat v nastavení pod položkou „System“ a následném výběru „Automation rules“ na konci seznamu v levém sloupci.

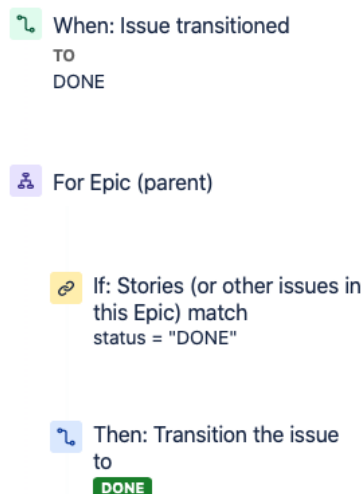
Automatizační pravidla je možné exportovat a importovat a jsou tedy k dispozici [v repozitáři práce](#).

Epic by měl měnit svůj stav na *IN PROGRESS*, když je práce na prvním z jeho Stories. Do konečného stavu *DONE* by měl být přesunut v moment, kdy jsou všechny jeho podúkoly dokončeny. Vytvořená automatizační pravidla pro tyto dvě operace jsou zobrazená na následujících obrázcích.



Obrázek 20: Automatizační pravidlo pro posun Epicu do stavu *IN PROGRESS* v nástroji Jira¹⁷

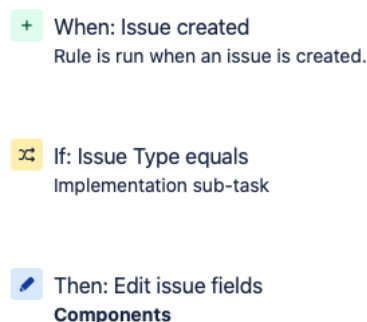
¹⁷ Slovní popis: Pokud je úkol přesunut ze stavu *PREPARED* do stavu *IN PROGRESS* (odpovídající přechod stavů pro úkoly typu Story), změň stav jemu přiřazenému Epicu na *IN PROGRESS* (v případě, že stav Epic již není *IN PROGRESS* nebo *DONE*)



Obrázek 21: Automatizační pravidlo pro posun Epicu do stavu *DONE* v nástroji Jira¹⁸

Na podobném principu lze vytvořit automatizace pro přechod do stavu *IN PROGRESS* a *IMPLEMENTED* pro Story a příslušné sub-tasky.

Další vhodnou automatizací je kopírování hodnoty *Components* (odpovídající platformě) při vytváření implementačních sub-tasků pro Story. Je to forma zajištění toho, že se tato hodnota nezapomene přiřadit, což by mohlo způsobit, že vývojář daný sub-task neuvidí na svém Boardu. Podobu automatizačního pravidla lze vidět na následujícím obrázku.

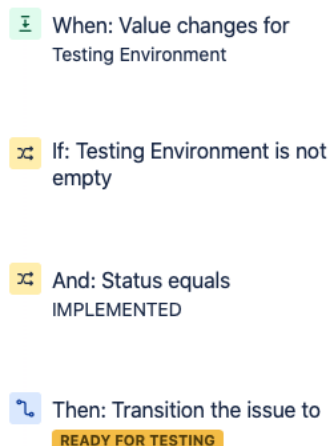


Obrázek 22: Automatizační pravidlo pro kopírování hodnoty *Components* při vytváření implementačních sub-tasků v nástroji Jira¹⁹

V neposlední řadě je také vhodné automatizovat posun úkolů ze stavu *IMPLEMENTED* do *READY FOR TESTING*. Tento přechod je podmíněn pouze přidáním hodnoty *Testing Environment* (značící, kde je možné funkcionalitu otestovat). Pravidlo je opět zobrazeno na následujícím obrázku.

¹⁸ Slovní popis: Pokud je úkol přesunut do stavu *DONE*, změň stav jemu přiřazenému Epicu za předpokladu, že všechny Stories a jiné podúkoly Epicu jsou ve stavu *DONE*.

¹⁹ Slovní popis: Pokud je úkol typu Implementační sub-task vytvořen, změň hodnotu položky *Components* (není zobrazeno v přehledu, ovšem poslední krok má nakonfigurováno, že změna hodnoty má být na hodnotu rodičovského úkolu).



Obrázek 23: Automatizační pravidlo pro změnu stavu z *IMPLEMENTED* na *READY FOR TESTING* po přidání hodnoty pro položku *Testing Environment* v nástroji Jira²⁰

Nástroj pro řízení roadmapy

Pro řízení roadmapy byl vybrán nástroj *Easy Agile Roadmaps*, který je k dispozici jako doplněk pro nástroj Jira (nalézt ho je možné v Atlassian obchodu). Po instalaci je možné vytvořit roadmapu otevřením Boardu projektu (konkrétně Boardu pro Project Ownera a Agile Coache), kliknutím na tři tečky vpravo nahoře a zvolením položky *Roadmap*.

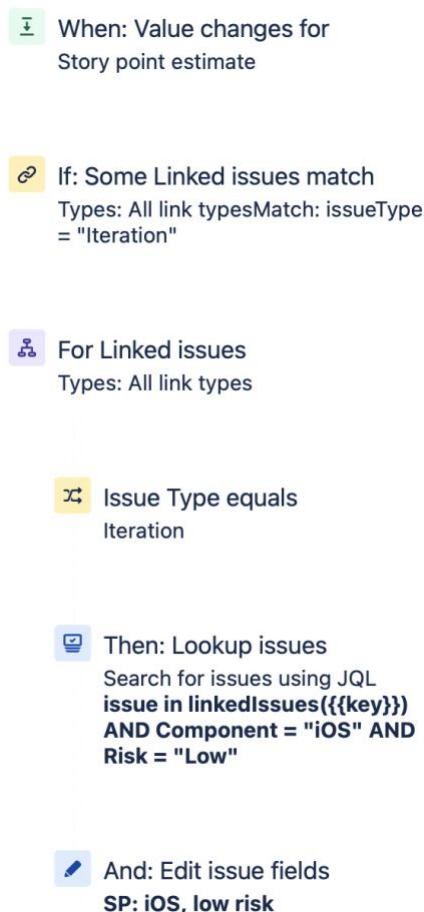
Tento nástroj je postaven na projekci Epiců (případně jiných typů úkolů) na roadmapu, ovšem v případě této metodiky je potřeba zobrazovat celky, které nemusejí mít konzistentní stav k Epicu (iterace by správně měla zahrnovat určitou část Epicu, ovšem může zahrnovat také více Epiců). Pro tyto účely je tedy nutné vytvořit nový typ úkolu – Iterace.

Tento typ úkolu by měl obsahovat pouze základní položky – identifikátor, název, datum začátku, datum ukončení a celkový počet Story Points pro jednotlivé platformy a stupně rizikivosti.

Vzhledem k tomu, že není potřeba řídit stav tohoto typu, postačí jednoduché workflow se dvěma stavy: *CREATED* (vytvořen) a *FINISHED* (dokončen).

Pro výpočet počtu Story Points úkolů přiřazených dané iteraci je potřeba vytvořit automatizační pravidla. Na následujícím obrázku je jeden z výpočtů pro celkový počet Story Pointů z úkolů platformy iOS s nízkou rizikovostí. Na stejném principu pak lze vytvořit pravidla i pro zbylé potřebné hodnoty.

²⁰ Slovní popis: Pokud se změní hodnota položky *Testing Environment* na neprázdnou (tzn. je přidána hodnota, nikoli odebrána) a stav úkolu je *IMPLEMENTED*, změň stav úkolu na *READY FOR TESTING*.



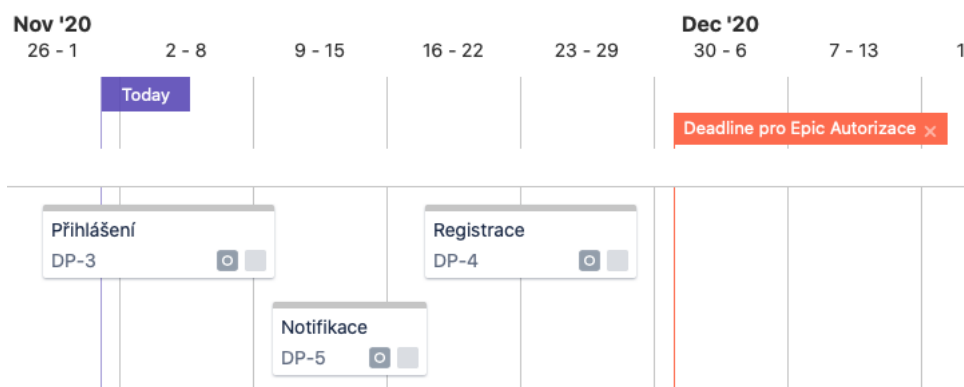
Obrázek 24: Automatizační pravidlo pro výpočet souhrnného odhadu pracnosti pro Iteraci ve Story Pointech (počítáno pro úkoly pro platformu iOS s nízkou úrovní rizikovosti) v nástroji Jira²¹

Jak bylo zmíněno již v předchozí sekci, automatizační pravidla jsou k dispozici také v repozitáři práce a mohou tak být přímo importovány bez nutnosti jejich znovuvytváření.

Product Owner poté může jednoduše dopočítat časovou náročnost iterace na základě součtu součinů hodnot odhadů ve Story Pointech a aktuálních hodnot ukazatelů SPT.

Jakmile jsou iteraci přiřazena data začátku a ukončení, tak se zobrazí na roadmapě. Na ní je následně možné s iterací volně posouvat s tím, že se hodnoty dat interaktivně mění. Do roadmapy je možné přidat milníky (označované jako *Markers*). Může jim být přiřazen popis a barva pro snazší identifikaci a také je možné je volně posouvat po mapě.

²¹ Slovní popis: Pokud se změní odhad ve Story Pointech a úkol je přiřazen k Iteraci, pak pro všechny přiřazené Iterace spočítej sumu odhadů všech přiřazených úkolů, které mají položku *Component* s hodnotou iOS a nízké riziko (není zobrazeno v přehledu, ovšem poslední krok obsahuje výpočet sumy položek „Story point estimate“ pro všechny nalezené úkoly).



Obrázek 25: Ukázka projekce roadmapy v nástroji Easy Agile Roadmaps

To vše by mělo poskytnout dostatečné programové podklady pro řízení projektu s navrženou metodikou.

5.3.2 Vzory dokumentů a ustanovení

Jak již bylo zmíněno v předchozí sekci, primárně je vycházeno z nástrojů, které již analyzovaný subjekt používá. V případě podnikové dokumentace je používán nástroj *Notion* a veškeré vzory dokumentů a ustanovení jsou tedy vytvořeny ve formátu vhodném pro prostředí tohoto nástroje. Pro příklad je použit přímo tento nástroj.

Vzory jsou v digitální podobě k dispozici ve veřejném repozitáři této práce ([příloha A](#)).

Projektová stránka

Každý projekt by měl mít projektovou stránku, která slouží jako zdroj všech informací, které na projektu vzniknou, a jako rozcestník pro všechny zdroje s projektem spojené.

V úvodu by měl mít krátký a stručný popis. Následně by měl odkazovat na seznam iterací (obsahující všechny náležité dokumenty) a měl by poskytnout odkazy na všechny externí zdroje – v tomto případě se jedná o odkazy na: roadmapu, Jira projekt, jednotlivé repozitáře pro každou z platforem a UX/UI design.

Projekt

Stručně popiš účel projektu.


Rozcestník

 Iterace


Důležité odkazy

Přidej odpovídající odkazy.


Roadmapa


 Add a web bookmark

Jira projekt


 Add a web bookmark

Repozitáře

 Add a web bookmark

 Add a web bookmark

UX/UI návrh

 Add a web bookmark

Obrázek 26: Vzor projektové stránky – úvodní část a rozcestník

Tato stránka by měla obsahovat také všechny ustanovení a náležitosti, které metodika vyžaduje před začátkem projektu. Týká se to rozsahu vývoje, rozsahu testování a definicí „Připraveno“ a „Hotovo“. Jejich formu je možné vidět na následujícím obrázku zachycujícím poskytovaný vzor stránky.

Rozsah projektu

Komponenty

Aa Komponenta	☰ Řešitel	+
Mobilní aplikace	Náš tým	
UX/UI design	Externí dodavatel	
Backend	Klient	
+ New		
COUNT 3		

Rozsah testování

Frekvence systémového testování: **po každé iteraci / před vydáváním nové verze** (vyber jednu z možností)

Odsouhlasená úroveň testování: **vyber z následujících**

Úroveň 1 – Statická analýza kódu během code review.

Úroveň 2 – Statická analýza kódu + sestavení posuzovatelem během code review.

Úroveň 3 – Posuzovatel během code review spustí kód a ručně otestuje novou funkcionalitu.

Úroveň 4 – Posuzovatel během code review napíše automatické testy pro otestování nové funkcionality.

TDD (Test Driven Development, Vývoj řízený testy) – Řešitel úkolu nejprve napíše automatické testy a pak až samotnou implementaci.

Ustanovení

Kdy je úkol připraven?

Doplň definici podle úvodní dohody. Příklad:

„Každý úkol musí mít popisný název, jasně pochopitelná akceptační kritéria a musí obsahovat zdroje ve vhodném formátu pro použití na mobilních platformách.“

Kdy je úkol hotov?

Doplň definici podle úvodní dohody. Příklad:

„Každý úkol je zodpovědně implementován a otestován v plném rozsahu dle specifikace. K návrhu i samotnému provedení bylo přistupováno s myšlenkou maximálně podpořit uživatelskou zkušenost při používání aplikace.“


Obrázek 27: Vzor projektové stránky – ustanovení

V neposlední řadě by na stránce měly být vyčteny také veškeré kontakty všech zainteresovaných osob na projektu. Měli by zde být vyčtení jak členové vlastního týmu, včetně jejich rolí, tak kontaktní osoby na straně klienta, také včetně jejich pozice či role.

Kontakty

Označ jednotlivé členy týmu v následujícím seznamu a přiřaď k nim jejich roli/role.

Tým

 Jméno (začni psát se @)

iOS

Android

Product Owner

Agile Coach

QA

Solution

+ New

Přidej zástupce na straně klienta, včetně jejich kontaktů.

Klient

 Jméno

@ E-mail

 Telefonní číslo

CTO

+ New

Obrázek 28: Vzor projektové stránky – kontakty

Report iterace

Jak je vyčteno již v návrhu, report by měl primárně obsahovat seznam dokončených, částečně dokončených a nedokončených úkolů. U částečně dokončených a nedokončených by mělo být poskytnuto také odůvodnění. Pro částečně dokončené je vhodné uvést, jaká část byla dokončena.

iOS

Vypiš všechny úkoly, které byly v rámci iterace plně dokončeny.

✅ Dokončené úkoly

ID	Název	
+ New		
COUNT 3		

Vypiš všechny úkoly, které byly v rámci iterace dokončeny částečně. Odůvodni, proč nebyly dokončeny v plném rozsahu. Pokud takové úkoly nejsou, tabulku smaž.

⚠ Částečně dokončené úkoly

ID	Název	Co nebylo dokončeno?	Odůvodnění	
+ New				
COUNT 3				

Vypiš všechny úkoly, které v rámci iterace nebyly dokončeny. Odůvodni, proč nebyly dokončeny. Pokud takové úkoly nejsou, tabulku smaž.

❌ Nedokončené úkoly

ID	Název	Odůvodnění	
+ New			
COUNT 3			

Obrázek 29: Vzor reportu iterace – seznam úkolů

Seznam úkolů musí samozřejmě vzniknout pro každou platformu zvlášť.

Další náležitostí reportu jsou hodnoty metrik. Měly by být vyčteny hodnoty naměřené v aktuální iteraci a zároveň i trendové hodnoty naměřené za poslední období. Pokud jsou mezi těmito hodnotami značné rozdíly, měl by být tento výkyv odůvodněn.

Výkonnost týmu

Doplň hodnoty metrik za aktuální iteraci a trendové hodnoty. Pokud jsou mezi těmito hodnotami značné rozdíly, odůvodni výkyv hodnot.

Hodnoty metrik

Aa Název metriky	# Trendová hodnota	# Hodnota v poslední iteraci	☰ Komentář	+
+ New				

COUNT 3

Obrázek 30: Vzor reportu iterace – výkonnost týmu

V neposlední řadě by měl report obsahovat seznam překážek, které se objevily v průběhu iterace, ovšem nebyly vyřešeny. Tyto překážky by měly být analyzovány, zda představují hrozbu i pro následující iterace.

Nedodělký

Doplň překážky, které vyvstaly v průběhu iterace, ale nebyly vyřešeny. Analyzuj zda je potřeba je nadále řešit a případně je přiřaď do další iterace.

Neodstraněné překážky

Aa Name	☑ Stále aktuální?	+
 Název překážky	Ano	
 Název překážky	Ne	
+ New		

COUNT 3

Obrázek 31: Vzor reportu iterace – nedodělký

Je také vhodné uvést datum dokončení iterace a zpracovatele reportu, aby nebyla ztracena výpovědní hodnota i po exportu mimo nástroj Notion (např. exportem pro tisk ve formátu pdf).

Zápis ze Standupu

Tento zápis by díky redukované formě Standupů definované metodikou měl obsahovat pouze seznam překážek. Tento seznam by měl být na začátku iterace prázdný a poté by do něj měly přibývat další objevené překážky, u kterých je průběžně měněn stav.

Stránka se zápisem by pro každý Standup měla vzniknout nově pro případ, že by bylo potřeba zapsat i jiné poznámky. Proto je doporučeno vždy předchozí stránky zkopírovat, aktualizovat datum a pokračovat se zápisem z aktuální schůzky.

xx.xx.xxxx (doplň datum)

Překážky

 <u>Název překážky</u>	Identifikována
 <u>Název překážky</u>	Řeší se
 <u>Název překážky</u>	Vyřešena
+ New	

Obrázek 32: Vzor zápisu ze Standupu

Zápis z retrospektivy

Jak již zaznělo v návrhu, forma retrospektivy není nijak striktně definována a sám tým by si měl zvolit formu, která mu bude nejvíce vyhovovat. Ovšem vždy by výstupem této schůzky měl být seznam akčních bodů. Měl by být také aktualizován jejich stav, jelikož poslední seznam akčních bodů by měl být vždy představen na začátku následující retrospektivy.

Retrospektiva

Doplň formu retrospektivy, která bude vyhovovat vašemu týmu.

Akční body

 <u>Název akčního bodu</u>	Vytvořen
 <u>Název akčního bodu</u>	Řeší se
 <u>Název akčního bodu</u>	Vyřešen
+ New	

Obrázek 33: Vzor zápisu z retrospektivy

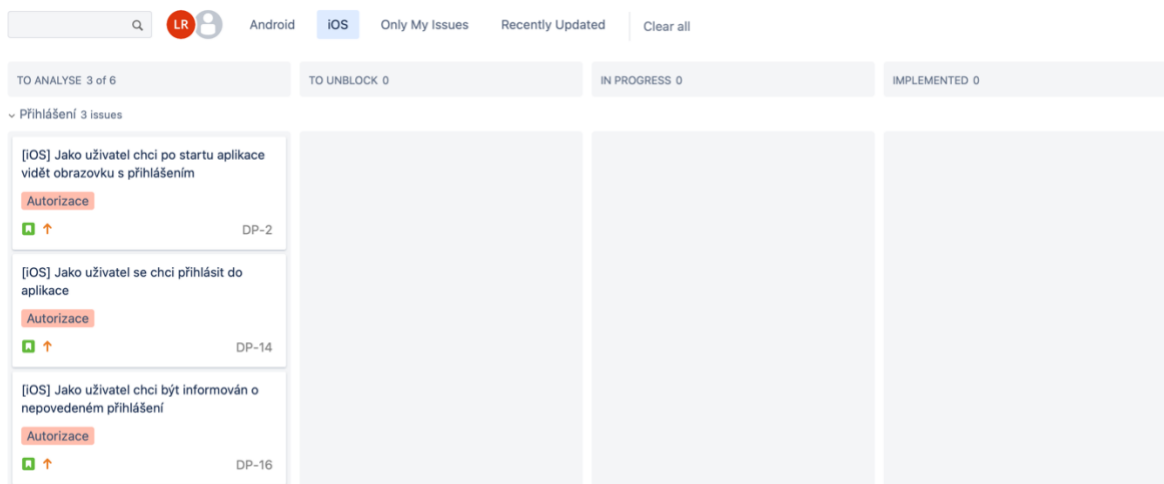
5.3.3 Simulace modelových situací

V této sekci je představeno několik modelových situací, na kterých je ukázána funkcionality konfigurovaných nástrojů a jsou popsány stavy a osoby odpovědné za jejich řešení. Tyto situace by měly pomoci s pochopením propojenosti nástrojů a přidáné hodnoty užívání navržených praktik.

Pro větší přehlednost a jednodušší pochopení je v modelových situacích uvažován pouze rozsah vývoje nezahrnující vývoj backendu a UX/UI návrh.

Před započítáním iterace

Než je samotná iterace započatá, musí všichni zástupci rolí Solution Expert analyzovat Stories a rozdělit je na konkrétní sub-tasks. Jejich Board by tedy před započítáním iterace měl vypadat podobně jako na následujícím obrázku.

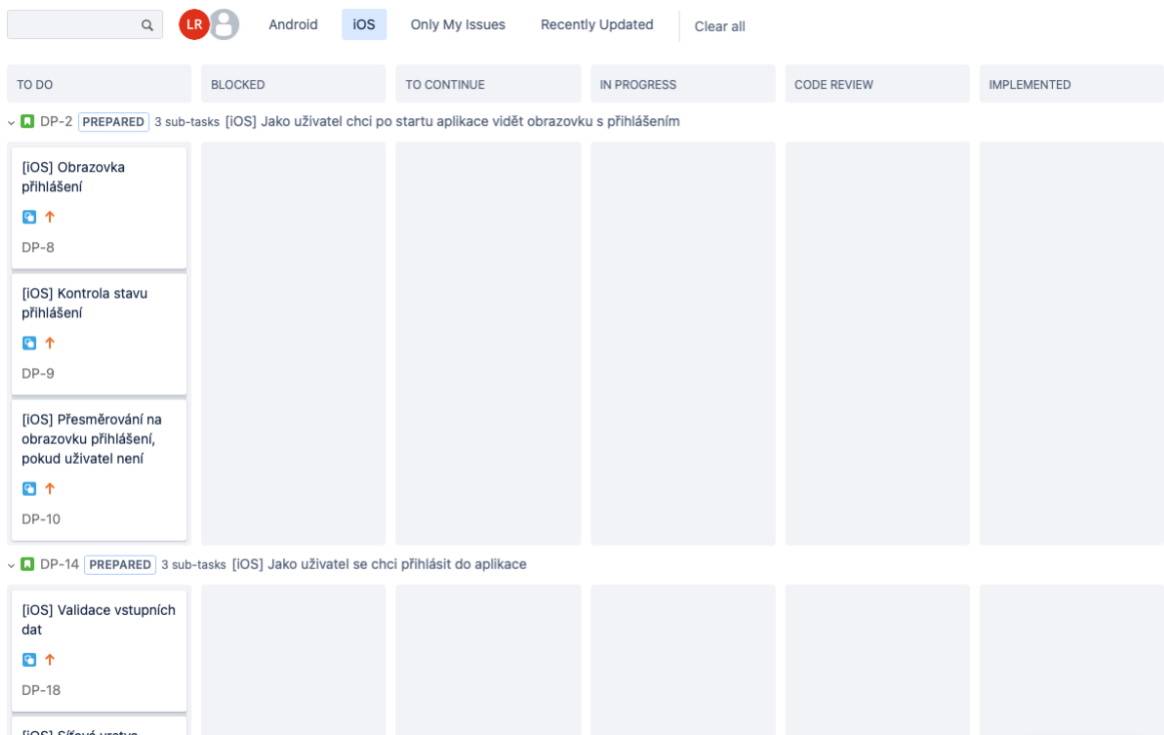


Obrázek 34: Modelová situace pohledu na Board pro roli Solution Expert před započítím iterace

Po vykonané analýze jsou všechny Stories přesunuty do stavu PREPARED a na tomto Boardu jsou tedy přesunuty do sloupce IN PROGRESS. Od započetí iterace samotné by se nikdy nemělo stát, že by se jakýkoli úkol nacházel v prvním sloupci tohoto Boardu. To se může stát vizuální indikací toho, že byla fáze analýzy zanedbána.

Započetí iterace

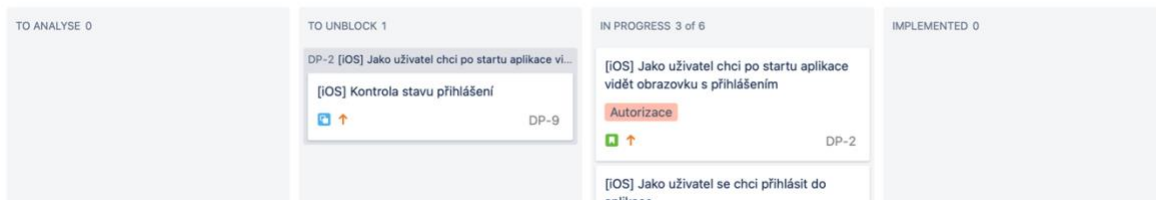
V moment, kdy je započata iterace, by měl Board pro mobilní vývojáře vypadat stejně jako na následujícím obrázku. Všechny implementační sub-tasks by měly být připraveny v prvním sloupci TO DO. Jakmile začne vývojář na prvním z nich pracovat, přesune ho do sloupce IN PROGRESS. Board pro Testera by měl být v tuto chvíli prázdný.



Obrázek 35: Modelová situace pohledu na Board pro mobilní vývojáře při započítí iterace

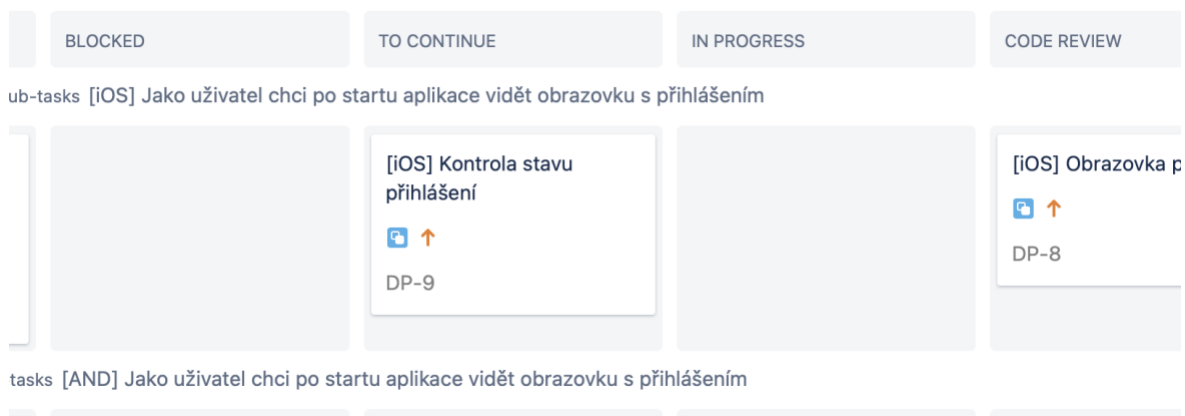
Nalezení nedostatku vývojářem

Pokud vývojář nalezne nedostatek ve specifikaci sub-tasku, přesune ho do stavu **BLOCKED BY SPECIFICATION**. V tuto chvíli úkol mizí z Boardu mobilních vývojářů a objevuje se v druhém sloupci na Boardu pro roli Solution Expert, jak je zobrazeno na následujícím obrázku.



Obrázek 36: Modelová situace pohledu na Board pro roli Solution Expert po nalezení nedostatku sub-tasku vývojářem

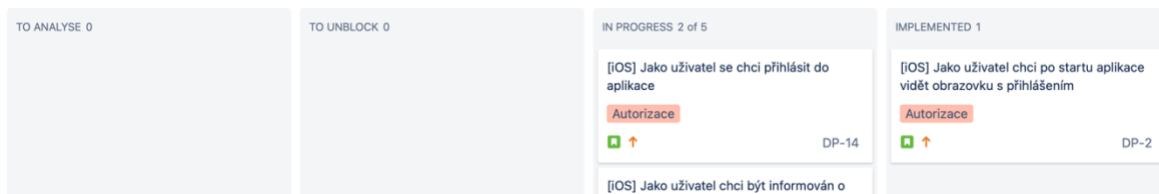
Jakmile je specifikace úkolu doplněna, Solution Expert přesouvá úkol do stavu **TO CONTINUE** a ten se tedy opět objevuje na Boardu mobilních vývojářů.



Obrázek 37: Modelová situace pohledu na Board pro mobilní vývojáře po doplnění specifikace sub-tasku

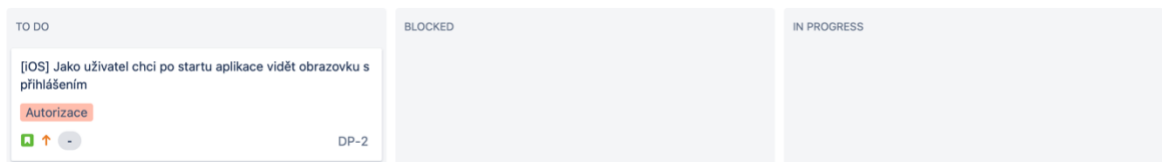
Dokončení implementace Story

Jakmile jsou dokončeny všechny sub-tasky Story, je automaticky přesunuta do stavu **IMPLEMENTED**, a tedy i do posledního sloupce Boardu pro roli Solution Expert.



Obrázek 38: Modelová situace pohledu na Board pro roli Solution Expert po dokončení implementace všech sub-tasků Story

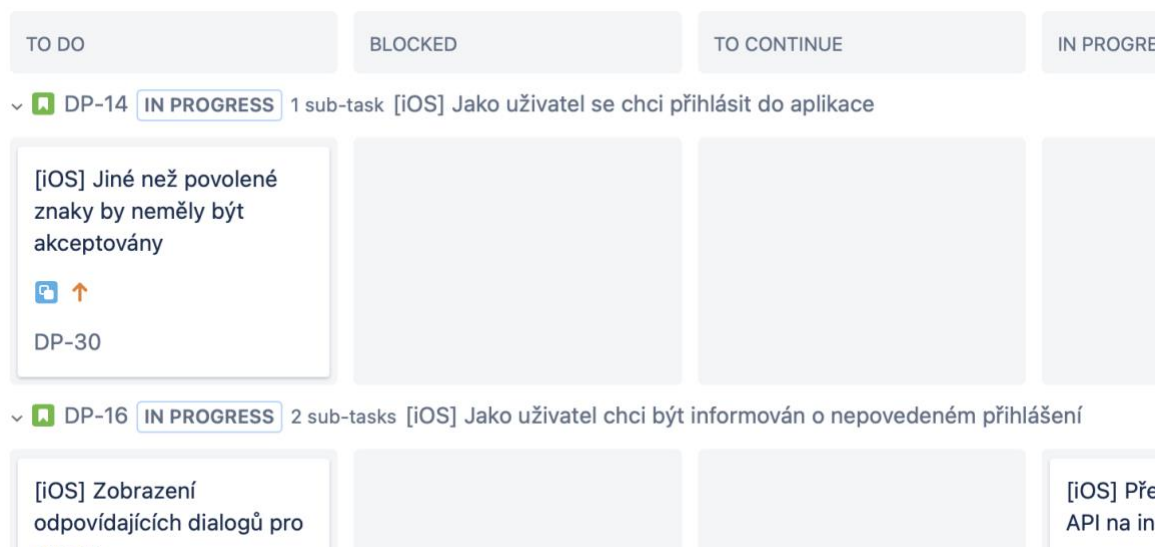
Ve chvíli, kdy je v ní vyplněna položka testovacího prostředí, je přesunuta do stavu **READY FOR TESTING** a objevuje se na Boardu testera.



Obrázek 39: Modelová situace pohledu na Board pro testera po přiřazení testovacího prostředí implementované Story

Návrat Story kvůli nedostatku

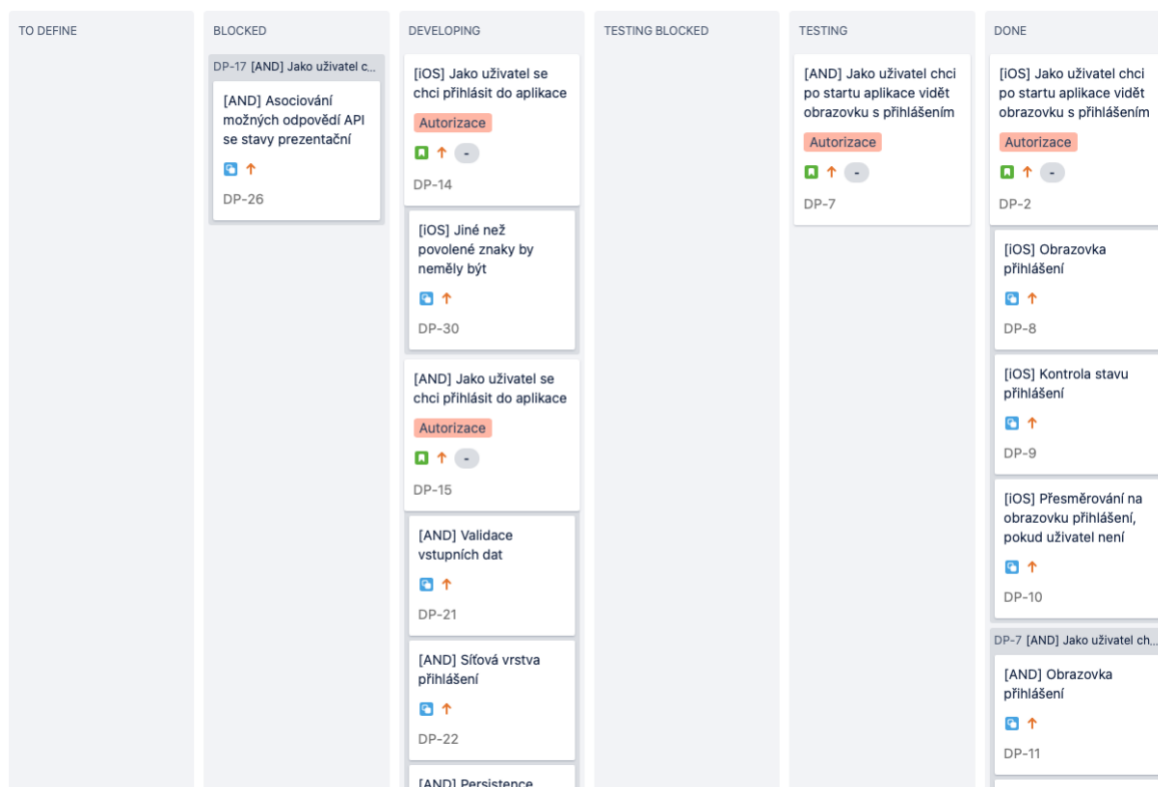
Pokud tester objeví nedostatek při testování implementace Story, zakládá nový sub-task a Story vrací do stavu IN PROGRESS. Ta se tedy opět objevuje na Boardu pro roli Solution Expert, zatímco nový sub-task se objevuje na Boardu mobilních vývojářů.



Obrázek 40: Modelová situace pohledu na Board pro mobilní vývojáře po nalezení nedostatku v implementaci Story během jejího testování

Pohled rolí Product Owner a Agile Coach v průběhu iterace

Vzhledem ke specifickým pohledům jednotlivých rolí nelze z výše zmíněných Boardů vyčíst obecný postup všech úkolů dané iterace. Tento pohled poskytuje pouze Board pro role Product Owner a Agile Coach. Na obrázku níže je zobrazen pohled na tento Board v průběhu iterace.



Obrázek 41: Modelová situace pohledu na Board pro role Product Owner a Agile Coach v průběhu iterace

5.4 Shrnutí

Na základě navrženého postupu implementace metodiky včetně návodů pro konfiguraci jednotlivých nástrojů by měl být nyní čtenář schopen metodiku zavést a řídit. Postup by měl být dostatečně podrobný tak, aby nevyžadoval hlubší znalosti práce s vybranými nástroji.

Pro vyšší úspěšnost zavedení metodiky je nutné se držet navržených kroků postupu (pokud není metodika zaváděna v novém prostředí). Ovšem není nutné se striktně držet navržených nástrojů a k aplikaci metodiky je možné využít jakýchkoli nástrojů, které je možné správně konfigurovat dle specifik metodiky. Výběr jiných nástrojů je tedy doporučen pouze v případě, že je čtenář dostatečně obeznámen s jejich možnostmi.

I v případě použití jiných nástrojů je vhodné postupovat dle kroků konfigurací nástrojů vybraných v této práci, jelikož postup by měl zhruba odpovídat s použitím jakéhokoli nástroje stejného typu.

Závěr

Hlavního vytyčeného cíle práce, včetně všech jeho podcílů, bylo dosaženo. Byla vytvořena komplexní metodika vývoje mobilních aplikací založená na specifikách prostředí, která byla v rámci práce analyzována. Práce obsahuje také detailní postup implementace metodiky v prostředí analyzovaného subjektu, a tudíž splňuje i stanovený cíl možnosti jednoduché implementace.

Navržená implementace metodiky má dále prostor pro optimalizaci. První položkou na seznamu je určitě propracovanější integrace mezi nástrojem pro správu roadmapy a nástrojem pro podporu řízení projektu. Další automatizace jako například generování reportu z iterace by měly následovat.

V návrhu také existuje prostor pro optimalizaci testovacích procesů. Tester by mohl přiřazovat relevantní testovací případy jednotlivým úkolům, na základě čehož by bylo možné generovat testovací scénáře pro jednotlivé verze či iterace.

I přesto, že je zde velký prostor k optimalizování, považuji i současné řešení za dostatečně efektivní s dostatečným přínosem pro subjekt, ve kterém bude tímto způsobem metodika zavedena.

Práce je zároveň psána dostatečně deskriptivně a metodika samotná není závislá na prostředí analyzovaného subjektu, tudíž ji lze považovat za přínosnou pro jakéhokoli čtenáře zabývajícího se problematikou vývoje mobilních aplikací a může se pro něj stát návodem pro řízení tohoto vývoje ve vlastním prostředí.

Použitá literatura

42MATTERS. Google Play vs the iOS App Store | Store Stats for Mobile Apps. In: *42matters* [online]. 2020. Dostupné z: <https://42matters.com/stats>

AGILE ALLIANCE. What is Extreme Programming (XP)? In: Agile Alliance [online]. 2020. Dostupné z: <https://www.agilealliance.org/glossary/xp>

BOEHM, Barry. A spiral model of software development and enhancement. In: ACM Digital Library. 1986. Dostupné z: <https://dl.acm.org/doi/10.1145/12944.12948>

CHEN, Brian X. Apple Registers Trademark for „There’s an App for That“. In: *WIRED* [online]. 2010. Dostupné z: <https://www.wired.com/2010/10/app-for-that/>

HALF, Robert. Waterfall Methodology 101: the Pros and Cons. In: *Robert Half* [online]. 2014. Dostupné z: <https://www.roberthalf.com/blog/salaries-and-skills/waterfall-methodology-101-the-pros-and-cons>

KNIBERG, Henrik. *Scrum and XP from the Trenches*. 2. vyd. USA: C4Media, 2015. ISBN 978-1-329-22427-8.

MILIJIC, Marko. 29+ Smartphone Usage Statistics: Around the World in 2020. In: *Leftronic* [online]. 2019. Dostupné z: <https://leftronic.com/smartphone-usage-statistics/>

OŠKRDAL, Václav. *Využití principů, postupů a nástrojů procesního řízení při vedení projektů softwarového vývoje* – doktorská disertační práce, Praha: Vysoká škola ekonomická, 2012.

REDDY, Ajay. *The Scrumban [R]Evolution: Getting the Most Out of Agile, Scrum, and Lean Kanban*. USA: Pearson Education, 2016. ISBN 978-0-13-408621-7.

REHKOPF, Max, Claire DRUMOND a Dan RADIGAN. The Agile Coach. In: *Atlassian Agile Coach* [online]. 2020. Dostupné z: <https://www.atlassian.com/agile>

SUTHERLAND, Jeff. *Scrum: The Art of Doing Twice the Work in Half the Time*. London: Random House Business Books, 2015. ISBN 978-1-84794-110-7.

SUTHERLAND, Jeff a Ken SCHWABER. *The Scrum Guide* [online]. 2017. Dostupné z: <https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf>

TOOMBS, Cody. Google Play Store silently extends app review to all submissions, fails to inform developers. In: *Android police* [online]. 2019. Dostupné z: <https://www.androidpolice.com/2019/09/03/google-play-expands-store-app-review/>

WIKIPEDIA. Metodika vývoje softwaru. In: *Wikipedia* [online]. 2019. Dostupné z: [https://cs.wikipedia.org/wiki/Metodika_vývoje softwaru](https://cs.wikipedia.org/wiki/Metodika_v%C3%BDvoje_softwaru)

WIKIPEDIA. Software development process. In: *Wikipedia* [online]. 2020. Dostupné z: https://en.wikipedia.org/wiki/Software_development_process

WIKIPEDIA. Unified Process. In: *Wikipedia* [online]. 2020. Dostupné z: https://en.wikipedia.org/wiki/Unified_Process

Přílohy

Příloha A: Veřejný repozitář s digitálními zdroji práce

Veškeré digitální zdroje práce jsou k dispozici ve veřejném repozitáři dostupném na adrese <https://github.com/lukysnupy/metodika-vyvoje-mobilnich-aplikaci.git>.

Tento repozitář obsahuje

- finální verzi práce v digitální podobě,
- konfigurace a vzory pro nástroje pro podporu řízení,
- dokumentaci pro jednodušší orientaci ve zdrojích.

Příloha B: Manuál pro role: Product Owner

Hlavní zodpovědností této role je komunikace s klientem a transformace jeho vize do reálných návrhů a možností. Jeho hlavním úkolem v procesu je tedy vytváření Stories a odpovídajících akceptačních kritérií dle požadavků klienta. Na základě těchto požadavků a také směřování projektu by měl být schopen určovat priority úkolů podle jejich přidáné hodnoty. Za pomoci těchto informací by měl pravidelně pomocí techniky *backlog grooming* řadit úkoly dle jejich priorit a hodnoty.

V komunikaci s klientem musí také plánovat časové rozložení projektů, a tudíž druhou z jeho hlavních zodpovědností je tvorba a udržování roadmapy. Tato role je zodpovědná na rozdělování iterací, včetně rozhodování o jejich obsahu a objemu. Také by měla být schopna rozhodnout o vhodné návaznosti úkolů. Má na starosti také hlídání deadlinů a jiných milníků, které by měla projektovat do roadmapy a včas na ně reagovat.

Tato role se účastní všech definovaných pravidelných událostí, ovšem zastává zde spíš roli konzultanta a pozorovatele. Případně může dle svého uvážení průběžně informovat tým o budoucích plánech projektu.

Příloha C: Manuál pro role: Agile Coach

Jeho hlavní úloha je podpora týmu a hlídání dodržování procesů vyžadovaných metodikou. Měl by se starat o správné řízení komunikace v týmu i odstraňování překážek a obecně by se měl snažit o to, aby členové týmu nebyli zatíženi řešením okolností, které přímo nesouvisí s jejich vykonávanou činností.

Má na starosti řízení vnitřní efektivity týmu. Nejen, že by se měl starat o spokojenost týmu a o to, aby jednotliví členové mohli být sami o sobě na maximu své produktivity, ale má také na starost posun efektivity týmu jako celku. Toho by měl být schopený dosáhnout analýzou

metrik a jejich dosahovaných hodnot v čase. Na základě těchto analýz by měl sám navrhovat možnosti, jak zefektivňovat komunikaci, předávání pracovních úkonů mezi členy týmů a celkově optimalizovat procesy tak, aby tým fungoval jako jedna ruka. Všechny tyto skutečnosti by měl aktivně a transparentně komunikovat.

Konkrétní zodpovědností je zpracování analýz metrik do obecně pochopitelné podoby a jejich představení týmu v rámci schůzky Iteration Retrospective. Měl by přednést týmu aktuální čísla, kterých tým dosáhl a porovnat je s trendovými hodnotami, aby byla vyvolána diskuze nad možnými výkyvy těchto hodnot.

Je facilitátorem²² všech událostí definovaných metodikou, i jiných vyvolaných schůzek týmu, které se týkají průběhu projektu. Měl by být schopen vést věcnou diskusi, dělat zápisy ze schůzek a starat se o jejich pravidelný a konzistentní průběh.

Jeho zodpovědností je také odstraňování překážek, které jsou průběžně identifikovány. Pokud na řešení překážky není dostatečně kompetentní, dokáže delegovat řešení problému odpovědným osobám, ovšem jeho zodpovědností stále zůstává skutečnost, že bude překážka opravdu odstraněna. Na všech pravidelných schůzkách, kde překážky nebo akční body vznikají, musí vždy shrnout aktuální stav těch, které byly vzneseny na schůzkách předcházejících.

Příloha D: Manuál pro role: Mobilní vývojář

V důsledku okolností, které metodika definuje, by tato role měla být opravdu úzce zaměřena pouze na samotnou implementaci řešení úkolů. Neměla by v průběhu své vlastní činnosti řešit analýzu úkolů, případně si dokonce sama získávat informace, které v zadání úkolu chybí. Díky naprosté minimalizaci nutnosti odbíhání od kontextu by měla být zajištěna její nejvyšší možná produktivita.

Za běžných okolností se tato role pouze účastní všech pravidelných událostí a stará se o pohyb úkolů po vlastním Boardu. V případě, že nalezne v úkolu nedostatek, předává úkol dál odpovědné osobě (posunem stavu úkolu) a pokračuje v práci na dalším úkolu. Pokud narazí na překážku, ať už aktuální nebo možnou budoucí, oznámí to na příslušných schůzkách (Standup či Iteration Retrospective, dle povahy věci) a o samotné vyřešení překážky se nestará (pokud není přímo oslovena Agile Coachem s žádostí o její vyřešení).

Pokud je zástupce této role přiřazen jako posuzovatel jinému vývojáři, má také na starosti provádění code review ve smlouveném rozsahu kvality.

²² Moderátor schůzek a diskuzí.

Příloha E: Manuál pro role: Tester

Tato role by se, podobně jako role Mobilní vývojář, díky okolnostem definovaných metodikou měla být schopna úzce zaměřit na svou vlastní činnost bez nutnosti odbíhání od kontextu v důsledku odstraňování překážek.

Na začátku každé iterace by měla tato role analyzovat všechny Stories, které jsou součástí iterace, a jejich akceptační kritéria, na základě kterých by měla tvořit nové, případně upravovat stávající, testovací případy a scénáře.

V průběhu iterace se, mimo účasti na všech pravidelných schůzkách, stará o pohyb úkolů po vlastním Boardu, kde vidí již implementované úkoly, které jsou připravené k testování. V případě, že narazí na překážku, kvůli které není možné úkol otestovat, předá tento úkol odpovědné osobě (buď vývojářům k doplnění informací nebo Agile Coachi k odstranění této překážky).

V případě, že v průběhu testování narazí na chybu nebo nedostatek, koná následovně:

- Pokud je nalezena chyba během testování Story, založí nový sub-task Story s tagem *chybový*. Story by měla být automaticky posunuta zpět do stavu *In Progress*.
- Pokud je nalezena chyba během systémového či jiného nezávislého testování, založí úkol typu Bug a informuje Product Ownera, aby přiřadil Bugu prioritu a zařadil ho na odpovídající místo v Backlogu, případně aby mohl iniciovat vytvoření Hotfix iterace, pokud se jedná o chybu v produkčním prostředí.

Tato role má samozřejmě také na starosti provádění systémových testů, které jsou pravidelně prováděny ve frekvenci, která je definována smluveným rozsahem kvality.

Příloha F: Manuál pro role: Master Solution Expert

Hlavní zodpovědností této role je analýza Stories a rozdělování jejich řešení na jednotlivé sub-tasky. Tato role musí být zastoupena dostatečně kompetentní technicky zaměřenou osobou, která dokáže definovat odpovídající řešení pro alespoň z jednu z platform, pro které je aplikace vyvíjena.

Tato role by tedy měla v dostatečném předstihu (nejpozději do půlky předešlé iterace) analyzovat všechny Stories, které jsou obsahem dané iterace a rozdělit je na sub-tasky pro svou platformu. Zároveň také přiřazuje odhad náročnosti sub-tasků a v případě, že součet odhadů není roven odhadu náročnosti Story, změní i hodnotu její náročnosti. Odhady náročnosti mohou být rozporovány týmem při Iteration Planningu, kterého se tato role musí povinně účastnit.

Jakmile je analýza hotova, předává výsledky dalším Platform Solution Expertům, kteří na základě této analýzy vytvoří sub-tasky pro vlastní platformu.

Pokud osoba zastupující tuto roli není dostatečně kompetentní pro kompletní analýzu některé ze Stories, má možnost vyvolat schůzku všech Solution Expertů, v rámci které jsou dané Stories analyzovány a rozděleny celou skupinou.

V průběhu iterací má na starosti sledovat vlastní Board a starat se o vyřešení stavů *Blocked by..*, pokud se do nich některý ze sub-tasků dostane. Zároveň by měl také přesouvat úkoly ze stavu *Implemented* pomocí doplnění testovacího prostředí. Tento Board by měla kontrolovat alespoň dvakrát denně.

Příloha G: Manuál pro role: Platform Solution Expert

Tato role úzce souvisí s rolí Master Solution Expert (dále jen MSE). Má na starosti tvorbu sub-tasků pro svou platformu, a tedy i částečnou analýzu Stories na základě prvotní analýzy vytvořené MSE. V případě, že MSE vyvolá skupinovou analýzu, účastní se také této schůzky.

V průběhu iterací má stejné zodpovědnosti jako MSE, což je sledování Solution Expert Boardu a řešení stavů *Blocked by..* a *Implemented*. Board by měla kontrolovat alespoň dvakrát denně.