Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Testování softwaru v agilním prostředí

Bc. Karim Kelifa

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Karim Kelifa

Systémové inženýrství a informatika
Informatika

Název práce

Testování softwaru v agilním prostředí

Název anglicky

Software testing in an agile environment

Cíle práce

Cílem práce je nalézt a zhodnotit vhodný způsob testování softwaru v agilním prostředí. Práce se zaměřuje na problematiku testování softwaru v agilním prostředí v situaci, kdy je nedostatek kompletní specifikace softwarového produktu na začátku vývoje. Práce se zaměřuje na použití různých technik testování s ohledem na agilitu a jejich porovnání. Výsledek práce pomůže rozhodnout o tom, jaký způsob testování aplikovat na vlastní projekt. Tato diplomová práce může sloužit jako materiál pro výuku začínajících testerů.

Metodika

Analytická část práce se bude zakládat na rešerši a analýze odborných zdrojů.

V praktické části bude na základě zkušeností autora a poznatků zjištěných z analýzy odborných zdrojů použita vybraná metoda testování softwaru v agilním prostředí.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

testování, agilní metodika, test managment

Doporučené zdroje informací

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽAL a kolektiv. Efektivní testování softwaru. Praha: Grada, 2016. ISBN 978-80-247-5594-6.

CRISPIN, Lisa a Janet GREGORY. Agile Testing: A Practical Guide for Testers and Agile Teams. United States: Addison-Wesley, 20108. ISBN 978-0321534460.

LINZ, Tilo. Testing in Scrum: A Guide for Software Quality Assurance in the Agile World. United States: Rocky Nook Computing, 2014. ISBN 978-1-937538-39-2.

Předběžný termín obhajoby

2020/21 ZS – PEF (únor 2021)

Vedoucí práce

doc. Ing. Vojtěch Merunka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 19. 11. 2020

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 28. 02. 2021

Čestné prohlášení

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

V Praze dne 31.3.2021 _____

Poděkování

Rád bych touto cestou poděkoval vedoucímu této práce panu doc. Ing. Vojtěchovi Meruňkovi, Ph.D. za odborné vedení diplomové práce a cenné rady a připomínky, které mi při zpracovávání práce poskytl.

Testování softwaru v agilním prostředí

Abstrakt

Diplomová práce se zabývá problematikou testování softwaru v agilním prostředí. V teoretické části diplomové práce jsou vymezeny hlavní pojmy. Jedná se o definice testování softwaru, tradičního vývoje, agilního vývoje včetně agilního manifestu, vybraných agilních metodik a softwaru pro podporu agilního vývoje. Dochází ke srovnání tradiční a agilní metodiky vývoje softwaru.

Praktická část práce se věnuje charakteristice a rozboru fungování vybraných agilních projektů. U jednotlivých agilních týmů je proveden rozbor současného stavu testování. Na základě pozorování agilních týmů dochází k identifikaci slabých míst v testování softwaru jednotlivých týmů. Získaná data a informace v rámci zkoumání agilních týmů v oboru testování vedli k vytvoření návrhu a doporučením, které vedou ke zlepšení testování v agilním prostředí. Prvním návrhem je matice úrovní a typů testů, které by se měly provádět při vývoji softwaru v agilním prostředí. Jako druhý návrh je vytvořen příklad možného rozložení testovacích úrovní do sprintu během realizace agilního projektu.

Klíčová slova: testování softwaru, agilní metodika, Waterfall, Scrum, test management, techniky testování softwaru

Software testing in an agile environment

Abstract

The diploma thesis deals with software testing in an agile environment. The theoretical part of the thesis defines the main concepts. There are definitions of software testing, traditional development, agile development, including the agile manifest, selected agile methodologies, and software to support agile development. Traditional and agile software development methodologies are compared.

The practical part of the work is devoted to the characteristics and analysis of the functioning of selected agile projects. An analysis of the current state of testing is performed for individual agile teams. Based on the observations of agile teams and the author's experience, weaknesses in software testing of individual teams are identified. Subsequently, based on the information obtained in the theoretical part and the author's practical experience in the field of testing, a proposal and recommendations for improving testing in an agile environment are created. The first proposal is a matrix of levels and types of tests that should be performed when developing software in an agile environment. As a second proposal, an example of a possible division of test levels into a sprint during the implementation of an agile project is created.

Keywords: software testing, agile, Waterfall, Scrum, test management, software testing techniques

Obsah

Seznam obrázků

Seznam tabulek

Seznam použitých zkratek

ACC – Akceptační prostředí

AML – Anti Money Laundering

API – Application Programming Interface

BE – Backend

BUS – Business uživatel

DEV –Vývojové prostředí

INT – Integrační prostředí

KIT – Komponentní integrační testy

KYC – Know Your Customer

PROD – Produkční prostředí

PUI – Portal User Interface

SIT – Systémové integrační testy

UAT – Uživatelské akceptační testy

UT – Unit testy (jednotkové testy)

UX designer – User Expirience designer

WCM – Web Content Manager

1 Úvod

Testování funkčnosti je velice relevantní součástí vývoje a provozu softwaru. Včasné testování dokáže společnosti ušetřit spoustu peněz a nepříjemností s řešením větších či menších problémů s uživateli. Ideální testování by mělo probíhat ve všech fázích vývoje. Pro úspěšné otestování softwaru je důležité nejen samotné testování ale zejména správná příprava, tvorba dat pro testování, plánování, vyhodnocení provedených testů a správné reportování výsledků.

Cílem testování může být buď nalezení chyb v testované aplikaci nebo snaha o předcházení vzniku těchto chyb. Přičemž platí, že čím dříve se chyba odhalí tím snadněji a levněji je chyba opravena.

Testování je poměrně nákladná činnost, která může trvat i několik let, ale při vývoji softwaru hraje klíčovou roli a není radno ji zanedbávat. Je obecně známo, že neexistuje bezchybný software. Jestliže se software jeví jako bezchybný, tak to znamená, že se zatím žádná chyba neidentifikovala, avšak je pravděpodobné, že se ve zkoumaném softwaru chyba vyskytuje.

V dnešní uspěchané době existuje předpoklad, že firmy budou rychle reagovat na požadavky zákazníků, upravovat aplikace podle světových trendů a zároveň dodávat kvalitní a bezchybný software. Čím dál více firem přistupuje k agilní metodice vývoje softwaru, která se vyznačuje schopností rychlé reakce a je velice přizpůsobivá situacím.

Tato diplomová práce se zaměřuje na výběr vhodného způsobu testování v agilním prostředí.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je nalézt a zhodnotit vhodný způsob testování softwaru v agilním prostředí. Práce se zaměřuje na problematiku testování softwaru v agilním prostředí v situaci, kdy je nedostatek kompletní specifikace softwarového produktu na začátku vývoje. Práce se zaměřuje na použití různých technik testování s ohledem na agilitu a jejich porovnání. Výsledek práce pomůže rozhodnout o tom, jaký způsob testování aplikovat na vlastní projekt. Tato diplomová práce může sloužit jako materiál pro výuku začínajících testerů.

2.2 Metodika

Teoretická část práce se bude zakládat na rešerši a analýze odborné literatury a internetových zdrojů, které se vztahují k problematice testování softwaru a agilnímu vývoji softwaru. Teoretická část práce bude sloužit k seznámení se s problematikou, jako předpoklad pro sepsání praktické části této diplomové práce a k vyhodnocení vlastních návrhů a doporučení.

Pro tvorbu praktické části bude navázána spolupráce s vybranými projekty, které využívají agilní metodiky pro vývoj softwaru. V případě dohody zúčastněných stran budou poskytnuty informace a dokumenty potřebné pro praktickou část diplomové práce a bude umožněna praxe v rámci projektů. Na základě této praxe a spolupráce s projekty bude proveden sběr dat. Dojde k rozboru současného stavu jednotlivých projektů, a to především té části, která se zabývá testováním. Dojde k identifikaci problémů týkajících se testování v rámci agilního vývoje softwaru.

Na základě rozboru nedostatků v testování agilních týmů bude vytvořen obecný návrh a doporučení, které povede ke zlepšení procesu testování v agilním prostředí.

3 Teoretická východiska

V této kapitole jsou definovány hlavní pojmy týkající se testování softwaru, jako je definice testování softwaru a techniky testování softwaru. Dále se teoretická část práce věnuje agilnímu vývoji softwaru, agilnímu manifestu a vybraným metodikám vývoje softwaru v agilním prostředí. Na konci kapitoly jsou popsány vybrané nástroje pro podporu řízení agilního vývoje.

3.1 Testování softwaru

Testování softwaru je systematické a metodické zkoumání daného produktu za použití různých technik s úmyslem ukázat, že splňuje požadovaný nebo zamýšlený účel. Toto se provádí v prostředí, které je identické nebo téměř identické tomu, které bude použito v živém provozu. (Hambling, Morgan, Samaroo et al, 2019)

Tato definice je jen jedna z mála, protože všechny definice mají svá omezení. Definice závisí na cílech daného testování, nebo na testované aplikaci. Stejně tak je důležité vědět, že určené cíle na začátku procesu testování, se v průběhu životního cyklu vývoje softwaru mohou změnit.

Výchozí zásady testování (Hambling, Morgan, Samaroo et al, 2019):

- seznámení se s pracovním produktem,
- kontrola, zda byly splněny všechny požadavky a zda produkt pracuje tak, jak zúčastněné strany očekávají
- důvěřovat kvalitě testování,
- nalézt chyby a vady a zabránění jim v dosažení produkční verze softwaru
- poskytnout dostatečné informace k nápravě chyb, tak aby bylo možné rozhodnout, zda je produkt vhodný k vydání
- snížit úroveň rizika nedostatečné kvality softwaru (např. dříve nezjištěné poruchy vyskytující se v provozu),
- dodržovat regulační požadavky nebo normy.

3.1.1 Relevantnost testování

Podle současného trendu se díky neustálým změnám a rozvoji digitalizace život zlepšuje ve všech oblastech. Mění se také způsob práce. Lidé například přistupují k bance

online, nakupují online, objednávají si jídlo a mnoho dalších každodenních záležitostí řeší online. Všichni spoléhají na software a systémy. Ty se ovšem mohou prokázat jako vadné. Jedna malá chyba může mít obrovský dopad na podnikání z hlediska finanční ztráty a ztráty důvěry zákazníků. Aby se zajistilo dodání kvalitního produktu, je potřeba provádět testování v procesu vývoje softwaru. Důvody, proč se testování stává velmi významnou a nedílnou součástí oblasti informačních technologií, jsou následující (Roudenský, 2016):

1. Efektivita nákladů
2. Spokojenost zákazníků
3. Bezpečnost
4. Kvalita produktu

3.1.1.1 Efektivita nákladů

Ve skutečnosti nelze u žádného složitého systému nikdy zcela vyloučit vady návrhu. Není to proto, že vývojáři jsou neopatrní, ale proto, že složitost systému je neřešitelná. Pokud problémy s designem zůstanou nezjištěny, bude obtížnější dohledat vady zpětně a opravit je. Bude to nákladnější opravit. Někdy při opravě jedné chyby lze nevědomě přivodit další v nějakém jiném modulu. (Roudenský, 2016) Pokud je možné chyby identifikovat v raných fázích vývoje, jejich oprava stojí mnohem méně peněz. Proto je důležité najít vady v raných fázích životního cyklu vývoje softwaru. Jednou z výhod testování je nákladová efektivita. Je lepší začít s testováním dříve a zavést ho do každé fáze životního cyklu vývoje softwaru a je nutné pravidelné testování, aby bylo zajištěno, že aplikace bude vyvíjena podle požadavků.

3.1.1.2 Spokojenost zákazníků

V každém podnikání je konečným cílem zajistit co nejvyšší spokojenost zákazníků. Testování softwaru zlepšuje uživatelský komfort aplikace a poskytuje spokojenost zákazníků. Spokojení zákazníci znamenají pro podnik vyšší příjem. (Roudenský, 2016)

3.1.1.3 Bezpečnost

Bezpečnost je pravděpodobně nejcitlivější a nejzranitelnější část testování. Testování (penetrační testování a testování zabezpečení) pomáhá v zabezpečení produktu. Hackeři získávají neoprávněný přístup k datům a kradou informace o uživateli, které využívají je ve svůj prospěch a obohacení. Pokud se vyskytne bezpečnostní incident

na produktu budou se uživatelé jeho využívání vyhýbat z důvodu obav o svá data. Uživatelé vždy hledají důvěryhodné produkty. Testování pomáhá při odstraňování slabých míst v produktu. (Kitner, 2020)

3.1.1.4 Kvalita produktu

Testování softwaru je umění, které pomáhá posilovat reputaci společnosti na trhu tím, že dodává klientovi kvalitní produkt. Z těchto důvodů se testování softwaru stává velmi významnou a nedílnou součástí procesu vývoje softwaru.

3.1.2 Techniky testování softwaru

Existuje mnoho různých technik testování. Tyto techniky testování softwaru pomáhají navrhnout lepší testovací případy. Jaká technika bude použita při testování se vybírá a definuje při tvorbě testovacího plánu. (Kitner, 2020)

3.1.2.1 Statické a dynamické testování

Statické testování

Statické testování je ověřovací proces používaný k testování aplikace bez implementace kódu aplikace. Jedná se zejména o kontrolu (revizi) a prohlížení zkoumaného objektu. Mezi takovéto testování se řadí například kontrola kódu nebo dokumentace softwaru. Takovéto testování dokáže odhalit chybu hned na začátku vývoje a náklady a opravu jsou tak nízké. (Patton, 2002)

Dynamické testování

Dynamické testování je jednou z nejdůležitějších částí testování softwaru, které se používá k analýze dynamického chování kódu. Dynamické testování pracuje se softwarem tak, že se dle konkrétního testovacího případu zadají vstupní hodnoty a ověřuje se, zda software vrací očekávané výstupní hodnoty. Toto je možno učinit ručně, nebo pomocí automatizovaného procesu. Dynamické testování lze provést, když je kód spuštěn v běhovém prostředí. Jedná se o proces ověřování, při kterém se provádí funkční a nefunkční testování. (Patton, 2002)

3.1.2.2 Manuální a automatické testování

Manuální testování

Manuální testování je proces testování softwaru, při kterém se testovací případy provádějí ručně bez použití jakéhokoli automatizovaného nástroje. Všechny testovací případy jsou provedené testerem ručně podle pohledu koncového uživatele. Zajišťuje, zda aplikace funguje, jak je uvedeno v dokumentu s požadavky nebo ne. Testovací případy jsou plánovány a implementovány tak, aby dokončily téměř 100 procent softwarové aplikace.

Ruční testování je jedním z nejzásadnějších testovacích procesů, protože dokáže najít viditelné i skryté defekty (chyby) softwaru. Rozdíl mezi očekávaným výstupem a výstupem daným softwarem je definován jako defekt.

Ručně testovat aplikaci je důležité a nutné zejména při nově vyvinutém softwaru či jeho části. Poté lze k testování používat automatické testy. Toto testování vyžaduje velké úsilí a čas, ale poskytuje jistotu bezchybného softwaru. Ruční testování vyžaduje znalost technik manuálního testování, ale ne žádného automatizovaného testovacího nástroje. Ruční testování je nezbytné, protože jedním ze základních principů testování softwaru je, že není možná 100 % automatizace. (Patton, 2002)

Automatické testování

Provádění sady testovacích případů pomocí automatizovaných testovacích nástrojů, se označuje jako automatické testování. Proces testování se provádí pomocí speciálních automatizačních nástrojů k řízení provádění testovacích případů a porovnání skutečného výsledku s očekávaným výsledkem. Testování automatizace většinou vyžaduje obrovské investice zdrojů a peněz. Je tedy důležité se správně rozhodnout, zda je ekonomicky výhodné automatické testování při testování software zavádět.

Obecně se automatizace hodí při testování opakovaných testovacích případů, jako jsou regresní testy. Testovací nástroje používané při testování automatizace se používají nejen pro regresní testování, ale také pro automatizovanou interakci grafického uživatelského rozhraní, generování nastavení dat, protokolování defektů a instalaci produktu.

Cílem testování automatizace je snížit počet manuálních testovacích případů, ale ne eliminovat žádný z nich. Hlavní výhodou automatizace je rychlost, efektivita, správnost,

přesnost a neúnavnost. Stále se ale musí pamatovat na to, že automatizované nástroje nemohou v žádném případě nahradit testery. Pouze jim pomáhají dělat jejich práci snáze, lépe a kvalitněji. (Patton, 2002)

3.1.2.3 Funkční a nefunkční testy

Funkční testy

Jedná se o typ testování softwaru, který se používá k ověření funkčnosti softwarové aplikace, zda funkce funguje podle specifikace požadavku. Ve funkčním testování je každá funkce testována zadáním hodnoty, určením výstupu a ověřením skutečného výstupu s očekávanou hodnotou. Funkční testování se provádí jako testování černé skříňky, které slouží k potvrzení, že se funkce aplikace nebo systému chová tak, jak očekáváme. Provádí se k ověření funkčnosti aplikace. Funkční testování se také nazývá testování černé skříňky, protože se zaměřuje spíše na specifikaci aplikace než na skutečný kód. Tester musí testovat pouze program, nikoli systém. Účelem funkčního testování je zkontrolovat primární vstupní funkci, nutně použitelnou funkci, tok grafického uživatelského rozhraní obrazovky. Funkční testování zobrazuje chybovou zprávu, aby se uživatel mohl snadno pohybovat po celé aplikaci. (Kitner, 2020)

Nefunkční testy

Nefunkční testování je typ testování softwaru pro testování nefunkčních parametrů, jako je spolehlivost, zátěžové testy nebo výkonnostní testy softwaru. Primárním účelem nefunkčního testování je otestovat rychlost čtení softwarového systému podle nefunkčních parametrů. Parametry nefunkčního testování nejsou nikdy testovány před funkčním testováním. Funkční i nefunkční testování je u nově vyvinutého softwaru povinné. Funkční testování kontroluje správnost interních funkcí, zatímco nefunkční testování kontroluje schopnost pracovat v externím prostředí. Měření a metriky používané pro interní výzkum a vývoj jsou shromažďovány a vytvářeny nefunkčním testováním. Nefunkční testování poskytuje podrobné znalosti o chování produktu a použitých technologiích. (Kitner, 2020)

3.2 Agilní metodiky

Agilní vývoj je považován za relativně mladou disciplínu softwarového inženýrství, která vyplynula z průmyslových potřeb rychlého vývoje software. Hlavní pozornost by se měla zaměřit na zákazníka a jeho potřeby. Jedná se o metodiky vývoje softwaru, které se soustředí na myšlenku iterativního vývoje, kde se požadavky a řešení vyvíjejí ve spolupráci mezi samoorganizujícími se mezifunkčními týmy. Výhodou v agilním vývoji je to, že umožňuje týmům poskytovat hodnotu rychleji s vyšší kvalitou a předvídatelností, a hlavně s větší schopností reagovat na změny.

3.2.1 Tradiční vs Agilní metoda vývoje

Většina společností se dnes zaměřuje na poskytování kvality a získávání spokojenosti zákazníků. Aby toho bylo možné dosáhnout, musí si poradit s výběrem mezi tradičními vývojovými metodikami a metodami agilního vývoje.

Ačkoli oba tyto přístupy mají pozitiva i negativa, při zahájení nového projektu hraje při výběru klíčovou roli správná volba. Při výběru vývojové metodiky je třeba vzít v úvahu následující hlavní body (Bruckner, 2012):

- **obchodní potřeba** – dopad implementace specifikovaných požadavků na podnikání zákazníků,
- **zákaznické vnímání** – zákaznická perspektiva dopadu na podnikání,
- **časový rámec projektu** – definovaný časový rámec pro implementaci projektu v reálném čase.

3.2.1.1 Tradiční metodika vývoje software

Tradiční metodiky vývoje softwaru jsou založeny na předem organizovaných fázích životního cyklu vývoje software. Tok vývoje je zde jednosměrný, od požadavků k návrhu a poté k vývoji, poté k testování a údržbě. V klasických přístupech, jako je model Waterfall, má každá fáze konkrétní výstupy a podrobnou dokumentaci, které prošly důkladným procesem kontroly. (Geoffrey, 2004)

Tradiční přístupy jsou vhodné, jestliže jsou dobře pochopeny požadavky – například v průmyslových odvětvích, jako je stavebnictví, kde každý jasně rozumí konečnému

produktu. Na druhou stranu v rychle se měnících odvětvích, jako je IT, mohou tradiční vývojové postupy selhat při dosahování cílů projektu. (Geoffrey, 2004)

Tradiční metodika má několik přístupů k vývoji softwaru. Nejznámějším a nejpoužívanějším přístupem je Vodopádový model.

Vodopádový model

Vodopádový model, známý také jako lineární cyklus, je proces, který je postaven na plánované práci a je většinou vhodný pro projekty s jasně definovanými požadavky. Jasně nastiňuje řadu kroků nebo fází, které by měly být podniknuty při vytváření informačního systému. Tyto fáze obvykle sledují konkrétní pořadí s kontrolou na konci každé fáze. Po dokončení kontroly a splnění nezbytných požadavků lze zahájit další fázi. Níže na obrázku č. 1 je uveden základní příklad modelu vodopádu. (Boehm a Papaccio, 1988)

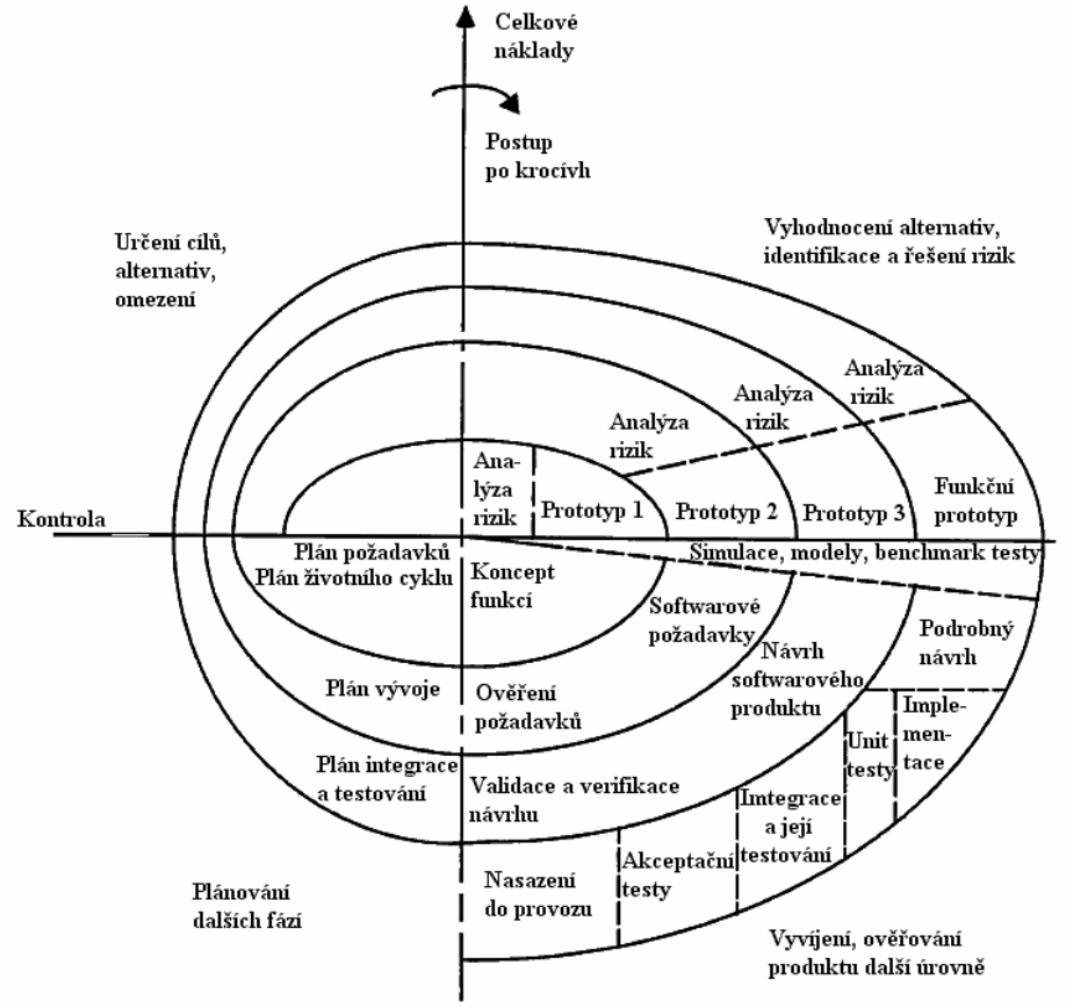
Obrázek 1: Proces vývoje softwaru ve Waterfallu

Zdroj: (Boehm a Papaccio, 1988), vlastní zpracování

Spirálový model

Spirálový model je považován za evoluční design v tom, že se jedná o systém vyvinutý z řady prototypů. Každý prototyp přidává vylepšení k předchozím. Softwarový projekt neustále prochází fázemi nebo iteracemi zvanými spirály. Tento model je vhodný zejména pro navrhování systémů experimentální povahy. Skládá se ze čtyř hlavních fází znázornění na obrázku č. 2. (Boehm, 2007)

Obrázek 2: Ukázka Spirálového modelu



Zdroj: (Hlava, 2011)

Hlavní nevýhody tradičních metod vývoje software:

- Business požadavky musí být definovány s dostatečným předstihem. Řešení je také třeba určit předem a nelze jej měnit ani upravovat. Celá sada požadavků musí být dána v počáteční fázi bez jakékoli šance na jejich změnu nebo úpravu po zahájení vývoje projektu.
- Uživatel nemůže provádět průběžná hodnocení, aby se ujistil, zda je vývoj produktu sladěn tak, aby konečný produkt splňoval obchodní požadavky.
- Uživatel získá systém založený na porozumění vývojáře, což nemusí vždy vyhovovat potřebám zákazníka.

- Vysoká priorita je kladena na dokumentaci a její tvorba je nákladná a časově náročná.
- Existuje méně šancí na vytvoření / implementaci opakovaně použitelných komponent.

Tyto nevýhody brání realizaci projektu z hlediska nákladů, úsilí, času, a nakonec mají zásadní dopad na vztahy se zákazníky.

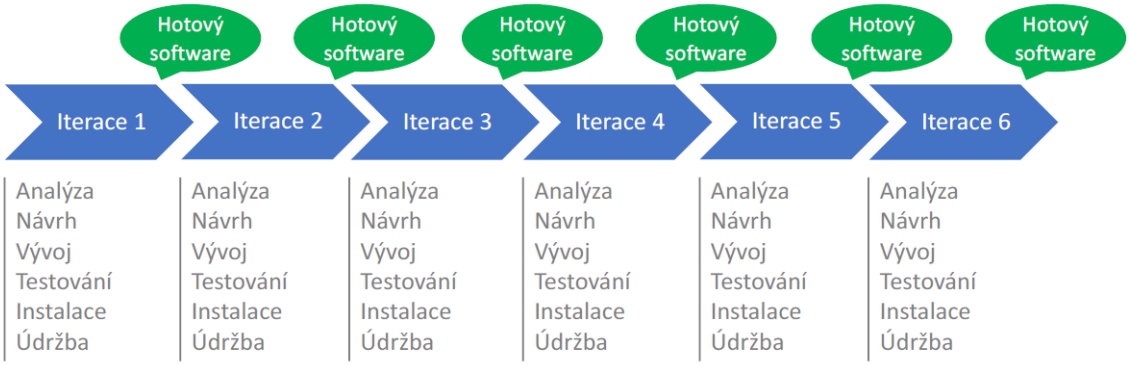
- Testování může začít až po dokončení procesu vývoje. Jakmile je aplikace ve fázi testování, není možné se vrátit a upravit cokoli, co by mohlo mít nepříznivý dopad na data dodání a náklady na projekt.
- Projekty se občas sešrotují, což vede k dojmu neefektivity a vede k zbytečnému úsilí a výdajům.
- Tradiční metodiky vývoje jsou vhodné, pouze pokud jsou požadavky přesné, tj. Když zákazník přesně ví, co chce, a může s jistotou říct, že během vývoje projektu nedojde k žádným zásadním změnám v rozsahu. Není vhodný pro velké projekty, jako jsou projekty údržby, kde jsou mírné požadavky a existuje velký prostor pro průběžné úpravy. (Haag a Cummings, 2012)

3.2.1.2 Agilní metodika vývoje softwaru

Na rozdíl od tradičních přístupů softwarového vývoje jsou agilní přístupy přesné a vstřícné k zákazníkům. Uživatelé / zákazníci mají možnost provádět úpravy během fází vývoje projektu. Mezi výhody Agile oproti tradičním metodikám vývoje patří:

- Ačkoli jsou business požadavky a řešení definovány předem, lze je kdykoli upravit.
- Požadavky / uživatelské příběhy lze poskytovat pravidelně, což znamená lepší šanci na vzájemné porozumění mezi vývojářem a uživatelem.
- Řešení lze určit rozdělením projektu do různých modulů a lze jej dodávat pravidelně.
- Uživatel dostane příležitost vyhodnotit moduly řešení, aby zjistil, zda je uspokojena obchodní potřeba, čímž jsou zajištěny kvalitní výsledky.
- Je možné vytvořit opakovaně použitelné komponenty.
- Dokumentace má menší prioritu, což má za následek menší časovou náročnost a výdaje.

Obrázek 3: Proces vývoje softwaru v agilním prostředí



Zdroj: (Crispin a Gregory, 2009), vlastní zpracování

3.2.2 Agilní manifest

Asi nejlépe vystihuje agilní vývoj manifest, který v roce 2001 sepsali osobnosti ve světě IT: Kent Beck, Mike Beedle, Arie van Bennekum, Alistair Cockburn, Ward Cunningham, Martin Fowler, James Grenning, Jim Highsmith, Andrew Hunt, Ron Jeffries, Jon Kern, Brian Marick, Robert C. Martin, Steve Mellor, Ken Schwaber, Jeff Sutherland, Dave Thomas. (Sutherland, Beck, Van Bennekum et al. 2001 a)

Tento manifest určuje priority a principy agilního vývoje. Jeho autoři v manifestu uvádějí, že objevují lepší způsoby vývoje software tím, že jej tvoří a pomáhají při jeho tvorbě ostatním. Při této práci dospěli ke čtyřem hodnotám. (Sutherland, BECK, Van Bennekum et al. 2001 a)

Jedná se o prohlášení, jehož cílem je zlepšit metodiky vývoje softwaru a přímo reaguje na neefektivitu tradičních vývojových procesů. Jmenovitě jejich spoléhání se na těžkou dokumentaci a příležitost k dohledu. (Sutherland, BECK, Van Bennekum et al. 2001 a)

Jednotlivci a interakce před procesy a nástroji

První hodnota agilního manifestu říká, že je potřeba si více vážit lidí než procesů nebo nástrojů. Jsou to totiž lidé, kteří reagují na obchodní potřeby a řídí proces vývoje. Pokud je vývoj řízen procesy nebo nástroji bude tým méně reagovat na změny a je méně pravděpodobné, že uspokojí potřeby zákazníků. Komunikace je příkladem rozdílu mezi oceňováním jednotlivců a procesem. V případě jednotlivců je komunikace plynulá

a probíhá v případě potřeby. V případě procesu je komunikace naplánována a vyžaduje konkrétní obsah. (Patton, 2002)

Fungující software před vyčerpávající dokumentací

Historicky bylo enormní množství času věnováno dokumentaci produktu pro vývoj a konečné dodání. Pro každou z nich jsou vyžadovány technické specifikace, technické požadavky, technický prospekt, dokumentace k návrhům rozhraní, test plány, plány dokumentace a schválení. Seznam byl rozsáhlý a byl příčinou dlouhých zpoždění ve vývoji. Agile nevylučuje dokumentaci, ale zefektivňuje ji ve formě, která dává vývojáři to, co je nutné k provedení práce, aniž by se zabředlo do detailů. Agilní požadavky na dokumenty jako uživatelské příběhy, které jsou dostatečné pro vývojáře softwaru k zahájení úkolu budování nové funkce. Agilní manifest hodnot hodnotí dokumentaci, ale více si váží pracovního softwaru.

Spolupráce se zákazníkem před vyjednáváním o smlouvě

Vyjednávání je období, kdy zákazník a produktový manažer vypracují podrobnosti o dodávce s body, které mohou být znovu vyjednány. Spolupráce je úplně jiný směr myšlení. S vývojovými modely, jako je Waterfall, zákazníci vyjednají požadavky na produkt, často velmi podrobně, před zahájením jakékoli práce. To znamenalo, že zákazník byl zapojen do procesu vývoje před zahájením vývoje a po jeho dokončení, ale ne během procesu. Agilní manifest popisuje zákazníka, který je zapojen a spolupracuje na celém procesu vývoje. Díky tomu je mnohem snazší během vývoje uspokojit potřeby zákazníka. Agilní metody mohou zákazníka zahrnovat v intervalech pro pravidelné ukázky, ale projekt může stejně snadno mít koncového uživatele jako každodenní součást týmu, který se účastní všech schůzek, což zajistí, že produkt splňuje obchodní potřeby zákazníka.

Reagování na změny před dodržováním plánu

Tradiční vývoj softwaru považoval změnu za výdaj, takže se jí bylo potřeba vyhnout. Záměrem bylo vypracovat podrobné a propracované plány s definovanou sadou funkcí a se vším, co má obecně stejně vysokou prioritu jako všechno ostatní, a s velkým počtem mnoha závislostí na dodávkách v určitém pořadí, aby tým mohl pracovat na další části projektu.

3.2.3 Dvanáct principů agilního manifestu

Dvanáct principů agilního manifestu jsou hlavními principy metodik, které jsou zahrnuty pod názvem „The Agile Movement.“ („Agilní hnutí“). Popisují kulturu, ve které je vítána změna, a zákazník se zaměřuje na práci. Rovněž prokazují záměr hnutí, jak ho popsal Alistair Cockburn, jeden ze signatářů Agilního manifestu, který má uvést rozvoj do souladu s obchodními potřebami.

Mezi dvanáct principů agilního manifestu patří (Sutherland, BECK, Van Bennekum et al., 2001 b):

1) **Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.** Zákazníci jsou spokojenější, když dostávají pracovní software v pravidelných intervalech, než aby čekali delší dobu mezi vydáním.
2) **Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.** Schopnost vyhnout se zpožděním při změně požadavku nebo funkce.
3) **Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.** Scrum vyhovuje tomuto principu, protože tým pracuje v softwarových sprintech nebo iteracích, které zajišťují pravidelné dodávání fungujícího softwaru.
4) **Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.** Když je obchodní a technický tým sladěn tvoří spolu lepší rozhodnutí.
5) **Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.** Je větší pravděpodobnost, že motivovaný tým dodá svoji nelepší práci na rozdíl od nespokojeného týmu.
6) **Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.** Je tedy lepší, aby týmy pracovaly vedle sebe.
7) **Hlavním měřítkem pokroku je fungující software**
8) **Agilní procesy podporují udržitelný rozvoj**. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo dodávání softwaru.

9) **Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.** Správné dovednosti a dobrý design zajišťují, že tým dokáže udržet tempo, neustále vylepšovat produkt a udržovat změny.
10) **Jednoduchost** – umění maximalizovat množství nevykonané práce-je klíčové.
11) **Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.** Kvalifikovaní a motivovaní členové týmu, kteří mají rozhodovací pravomoc, přebírají vlastnictví, pravidelně komunikují s ostatními členy týmu a sdílejí nápady, které dodávají kvalitní produkty.
12) **Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším** Sebezdokonalování, zlepšování procesů, rozvíjení dovedností a techniky pomáhají členům týmu pracovat efektivněji.

3.2.4 Scrum

Agilní metodika Scrum je systém řízení projektů založený na sprintu, jehož cílem je poskytnout zainteresovaným stranám nejvyšší hodnotu. Scrum je odlehčený rámec, který pomáhá lidem, týmům a organizacím generovat hodnotu prostřednictvím adaptivních řešení složitých problémů. Spolutvůrci Scrumu Ken Schwaber a Jeff Sutherland napsali „The Scrum Guide“, aby Scrum vysvětlili jasně a stručně. Tato příručka obsahuje definici Scrumu. Tato definice se skládá z odpovědností Scrumu, událostí, artefaktů a pravidel, která je spojují.

Tato metodika je o systému řízení projektů, který se spoléhá na postupný vývoj. Každá iterace se skládá ze dvou až čtyřtýdenních sprintů, kde cílem každého sprintu je nejprve vytvořit nejdůležitější funkce a vyjít s potenciálně dodávaným produktem. V následujících sprintech jsou do produktu zabudovány další funkce a jsou upraveny na základě zpětné vazby zainteresovaných stran a zákazníků mezi sprinty. Zatímco jiné metody řízení projektů kladou důraz na budování celého produktu v jedné iteraci od začátku do konce, scrum se zaměřuje na dodání několika iterací produktu, aby zúčastněným stranám poskytla nejvyšší obchodní hodnotu za co nejkratší dobu. Tato metodika má několik výhod. Podporuje rychlejší vytváření produktů, protože každá sada cílů musí být dokončena v časovém rámci každého sprintu. Vyžaduje také časté plánování a stanovování cílů, což pomáhá Scrum týmu soustředit se na cíle aktuálního sprintu a zvýšit produktivitu. (Ken, Sutherland, a Grüttner, 2020)

Scrum může být přínosem pro širokou škálu podniků a projektů, nejvyšší přínos je pro:

- Složité projekty: Metodika Scrum je ideální pro projekty, které vyžadují, aby týmy dokončily nevyřízené položky.
- Společnosti, které oceňují výsledky: Scrum je také přínosem pro společnosti, které oceňují výsledky v průběhu zdokumentovaného postupu procesu.
- Společnosti, které se starají o zákazníky: Scrum může pomoci společnostem, které vyvíjejí produkty v souladu s preferencemi a specifikacemi zákazníků.

Některé hlavní výhody scrumu:

- Pružnost a přizpůsobivost
- Kreativita a inovace
- Nižší náklady
- Zlepšení kvality
- Organizační synergie
- Spokojenost zaměstnanců
- Spokojenost zákazníků (Ken, Sutherland, a Grüttner, 2020)

Největší výhodou této agilní metodiky je její flexibilita. S modelem založeným na sprintu obdrží scrum tým po každém sprintu obvykle zpětnou vazbu od zúčastněných stran. Pokud dojde k jakýmkoli problémům nebo změnám, může scrum tým snadno a rychle upravit cíle produktu během budoucích sprintů a zajistit tak cennější iterace. Tímto způsobem jsou zúčastněné strany šťastnější, protože po zapojení na každém kroku dostanou přesně to, co chtějí. (Ken, Sutherland, a Grüttner, 2020)

Aby bylo možné implementovat agilní metodiku scrumu, musí být ve společnosti buď odborník na scrum, nebo externí konzultant pro scrum, který zajistí správné použití principů scrumu. Metodika agilního scrumu zahrnuje přesné provedení, které by mohlo vést ke katastrofě, pokud nebude provedeno správně.

3.2.4.1 Role ve Scrum

Se změnou přístupu k vývoji softwaru bylo potřeba přehodnotit i role a jejich povinnosti. Nově byli ve Scrumu definovány tři hlavní role: Scrum Master, Product Owner a Scrum tým.

Scrum master

Scrum master je zprostředkovatelem procesu vývoje scrumu. Jedná se o vůdce Scrum týmu a je zodpovědný za prosazování projektu, poskytování poradenství týmu a vlastníkovi produktu a zajišťování toho, aby členové týmu dodržovali všechny agilní postupy. Scrum master neřeší pouze všechny aspekty agilního procesu vývoje, ale slouží také podnikům, vlastníkům produktů, týmům a jednotlivcům a usnadňuje komunikaci a spolupráci mezi všemi těmito prvky. (Ken, Sutherland, a Grüttner, 2020)

Vzhledem k tomu, že role je ve spojení mezi obchodem, vlastníkem produktu, agilním týmem a jednotlivci, odpovědnost Scrum mastera se bude lišit v závislosti na jedinečných potřebách každého podniku a týmu. Někteří Scrum masteři také slouží jako týmový projektový manažeři. Někteří také plní roli agilního trenéra organizace.

Scrum master obecně plní tyto povinnosti (Ken, Sutherland, a Grüttner, 2020):

- Vedení a koučování organizace při jejím přijetí Scrumu
- Plánování implementace Scrumu v organizaci
- Tvořit změny, které zvyšují produktivitu Scrum týmu
- Spolupráce s dalšími Scrum Mastery na zvýšení efektivity Scrumu v organizaci

Může zdát, že scrum master je pouze jiný název pro projektového manažera známého z tradičních metod vývoje softwaru. Není tomu tak. Jak již bylo zmíněno, Scrum master může také plnit roli projektového manažera, ale je to jen část jeho definice zaměření. Níže v tabulce č. 1 jsou uvedeny rozdíly v rolích a odpovědností každého z nich:

Tabulka 1: Rozdíl mezi Scrum masterem a Projektovým manažerem

	Scrum master	Projektový manažer
Obecná role	• Pro projekty využívající agilní metodiky je klíčovou rolí Scrum master. Hraje roli zprostředkovatele a kouče pro agilní vývojové týmy při zajišťování dodávek produktů včas se stanovenou kvalitou.	• U většiny typů projektů má projektový manažer hlavní roli ve všech fázích a činnostech projektu, včetně plánování, vedení, řízení, monitorování a uzavírání projektů.
Zodpovědnost	• Podporuje vlastníky produktů při vývoji produktu. • Vede schůzky Scrum týmu a poskytujte týmovou podporu během plánování a provádění sprintu. • Poskytujte koučování agilním týmům. • Zajistěte, aby byly dodržovány agilní zásady. • Pomáhá týmům s určováním priorit a řízení sprintů, aby zajistil včasné a přesné dodání produktu. • Pomáhá týmům překonat překážky s cílem úspěšného dokončení produktu.	• Identifikuje a dokumentuje obchodní a projektové požadavky a plány pokroku. • Určuje, dokumentuje a spravujte rozsah projektu, úkoly, milníky, časové osy, rozpočet a zdroje. • Vede a mentoruje projektový tým. Určuje a přiřazuje úkoly a priority. Přiděluje, sleduje a spravujte prostředky projektu. • Nastavuje, sleduje a spravuje časové osy a parametry kvality projektu. • Vypracovává strategie pro řízení a sledování rizik. • Spravuje všechny zúčastněné strany a jejich očekávání. • Zajišťuje splnění cílů projektu.

Zdroj: (Schuurman, 2020)

Povinnosti a úkoly Scrum mastera se liší podle toho, se kterými členy týmu pracují. Na obchodní úrovni vytváří Scrum master vývojové prostředí, které je kreativní, bezpečné, produktivní a podpůrné a umožňuje spolupráci více směry. Na úrovni vlastníků produktů usnadňuje Scrum master plánování a pomáhá vlastníkům produktů porozumět technikám a postupům Scrumu a dodržovat je. Na úrovni týmu poskytuje mistr Scrum vedení, koučování, podporu a usnadnění a pomáhá odstraňovat překážky, se kterými se týmy mohou během cesty setkat. Na individuální úrovni podporuje Scrum master individuální úsilí, řeší všechny problémy, které se vyskytnou, a odstraňuje překážky, které pomáhají jednotlivcům soustředit se a být produktivní. (Schuurman, 2020)

Product Owner

Jak je popsáno v příručce Scrum Guide, Scrum Product Owener neboli vlastník produktu, je odpovědný za maximalizaci hodnoty produktu vyplývající z práce Scrum týmu. Způsob, jakým se to provádí, se může v různých organizacích a Scrum týmech velmi lišit. (Ken, Sutherland, a Grüttner, 2020)

Vlastník produktu zastupuje zúčastněné strany (Stakeholdery), kterými jsou obvykle zákazníci. Aby bylo zajištěno, že scrum tým vždy přináší hodnotu zúčastněným stranám a podniku, určuje vlastník produktu očekávání produktu, zaznamenává změny produktu a spravuje backlog, podrobný a neustále aktualizovaný seznam úkolů pro projekt ve Scrum metodice. Vlastník produktu je také zodpovědný za stanovení priorit cílů pro každý sprint na základě jejich hodnoty pro zúčastněné strany, takže v každé iteraci jsou zabudovány nejdůležitější a doručitelné funkce.

Scrum Tým

Scrum tým je samoorganizovaná skupina tří až devíti jednotlivců, kteří mají obchodní, konstrukční, analytické a vývojové dovednosti pro provádění práce, řešení problémů a výrobu dodávaných produktů. Členové Scrum týmu si sami spravují úkoly a jsou společně odpovědní za plnění cílů každého sprintu.

Vedlejšími členy týmu jsou také stakeholdeři, kteří jsou zapojeny do projektu, ale nejsou v týmu angažovaný. Pomocné role se obvykle skládají ze zákazníků, managementu a členů výkonného týmu, kteří jsou zapojeni za účelem konzultací, hlášení pokroku a shromažďování zpětné vazby, aby lépe pracovali na poskytování nejvyšší možné hodnoty. (Sutherland, 2020)

3.2.4.2 Hlavní pojmy

Agilní metodika také definuje pojmy, které nebyli v tradičních metodikách použity jedná se zejména o nástroje, které vedou ke zlepšení přehledu událostí a kontrolou nad Scrum projektem, ale také ke zrychlení práce a odstranění neefektivních procesů. Níže jsou popsány důležité pojmy tak, jak je vidí Průvodce Scrum (Ken, Sutherland, a Grüttner, 2020):

Scrumové události

Scrumové události se někdy též nazývají Scrumové ceremonie. Ve Scrumu existuje pět takových událostí. Jedná se o Sprint, Sprint planning, Daily Scrum, Sprint Review a Sprint Retrospective. Všechny tyto události mají předem danou a pevnou maximální jednotku času. Takovému omezení se říká timeboxing. Timeboxing je běžnou součástí mnoha metodik řízení projektů, protože timeboxing udržuje týmy zaměřené na splnění úkolu tím, že poskytuje jasnou definici hotového.

Sprint

Sprint je základním pojmem Scrumu. Jedná se o krátké, pravidelné iterace, které by neměli trvat déle než 2-4 týdny. Cílem sprintu je udržovat krátké iterace. Díky tomu se Scrum tým může soustředit na potenciální dodání přírůstku produktu. Sprinty by měly mít konzistentní délku. Sprint se skládá z plánování sprintu, Daily Scrums, Sprint Review a Sprint Retrospektivy. Podle pravidla by v rámci Sprintu neměl být měněn žádný cíl. Na konci každého sprintu by měl tým dodat funkční produkt nebo jeho funkční část, dle cílů, které byli na daný sprint naplánovány.

Sprint Planning

V agilním rámci Scrum je Sprint Planning schůzka událostí, která stanoví cíl vývoje produktu a plán nadcházejícího sprintu na základě přezkoumání nevyřízených produktů týmem. Úspěšný Sprint Planning přinese dvě důležité položky:

- **Cíl sprintu:** jedná se o krátké písemné shrnutí toho, co tým plánuje dosáhnout v následujícím sprintu
- **Nevyřízené položky sprintu:** Seznam příběhů a dalších položek nevyřízených produktů, na kterých se tým dohodl, že budou v nadcházejícím sprintu pracovat.

Daily Scrum

Ve Scrumu pořádá tým každý den sprintu denní schůzky s názvem denní scrum nebo též stand-up. Setkání se obvykle konají na stejném místě a každý den ve stejnou dobu. V ideálním případě se denní scrum meeting koná ráno, protože pomáhá nastavit kontext pro práci nadcházejícího dne. Scrum meetingy jsou přísně časově omezeny na 15 minut. Cílem tohoto omezení je zajistit svižnost a věcnost těchto schůzek. Existuje

pravidlo, které určuje, že během Daily Scrumu můžou mluvit pouze členové Scrum týmu. To znamená, že by se neměli těchto schůzek aktivně účastnit steakholdeři.

Sprint Review

Jedná se o schůzku, které se účastní Scrum tým, Scrum Master, Product Owener a Steakholder, na konci každého sprintu. Tým na schůzce předvádí ukázku produktu a určí, co je hotové a co ne. Účelem setkání Sprint Review je, aby tým ukázal zákazníkům a steakholderům práci, kterou během sprintu odvedli, a porovnal ji se závazkem daným na začátku sprintu.

Sprint Retrospective

V rámci Retrospektivy Sprintu se schází celý tým a ohlížejí se za uplynulým Sprintem. Tým zjišťuje, které činnosti zvládl dobře, ve kterých činnostech by měl pokračovat a co lze udělat pro to, aby byl další Sprint lepší, zábavnější nebo produktivnější. Délka takové schůzky záleží na faktorech, jako je počet lidí v týmu, jak nový je tým, nebo jestli lidé v týmu pracují vzdáleně. Existuje ovšem pravidlo, že v případě měsíčního sprint by takováto schůzka neměla trvat déle než tři hodiny. (Scrum.org, 2020 b)

Product Backlog

Tento pojem se nese napříč všemi Agile metodikami. Product Backlog je sestaven ze všech věcí, které je třeba udělat pro dokončení celého projektu. Právě toto je zdroj práce, kterou Scrum Tým provádí. Není to však jen jednoduchý seznam. Efektivní Product Backlog má rozdělenou každou z položek na seznamu do řady kroků, které pomohou vývojovému týmu. Musí zde být uvedena i doba trvání položky, aby tým věděl, kdy má zahájit úkol a jak dlouho mu zbývá, než jej musí dokončit. Tento proces lze urychlit pomocí softwaru pro správu úloh. V Product Backlogu je také stanoven produktový cíl.

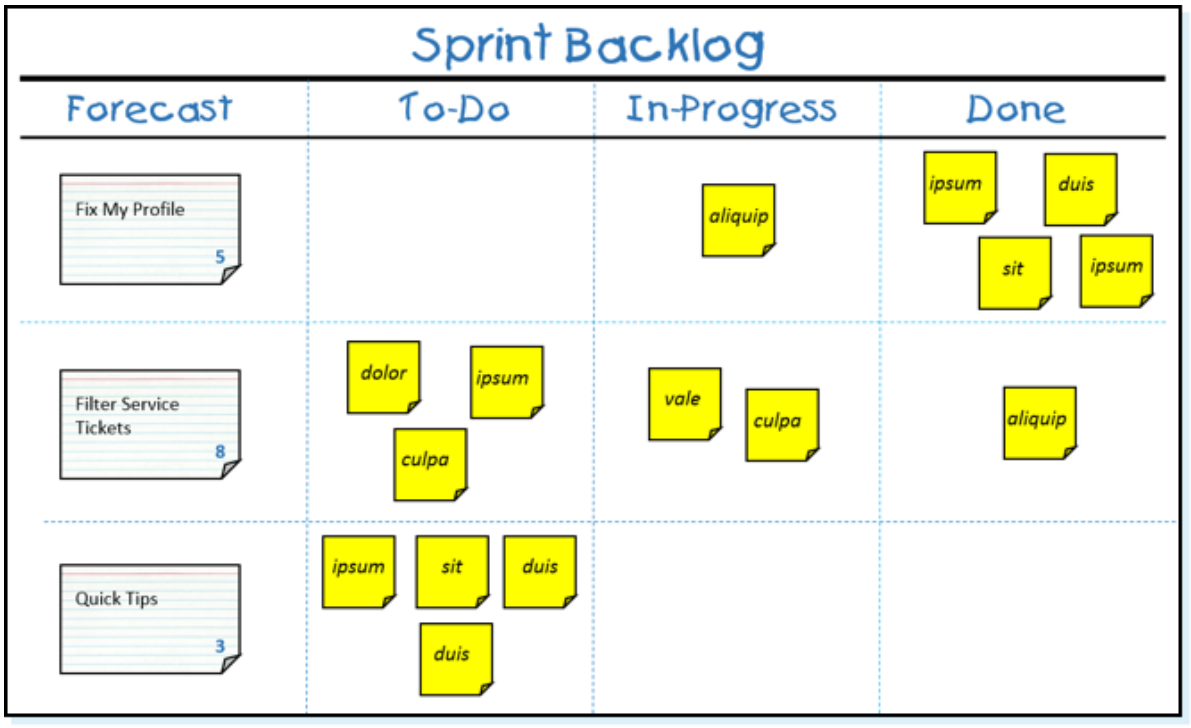
Obrázek 4: Ukázka Product Backlogu

	A	B	C	D	E	F
1	Project: SCRUM Test Project Query: Product Backlog					
2	Title	State	Backlog Priority	Story Points	Business Value	Iteration Path
3	As a Bank Customer I want to Register to the Online banking system so I can see my profile	Approved	9	5	9	\Release 1\Sprint 1
4	As a registered online banking cuotomer I want to login to online banking system so that I can View my profile	Approved	9	5	9	\Release 1\Sprint 1
5	As a Logged in Customer I want to my Account Transactions	Approved	9	8	9	\Release 1\Sprint 1
6	As a Logged In Customer I want to Create a New Account	Approved	7	8	7	\Release 1\Sprint 1
7	As a Logged In Customer I want to Pay my Creadit Card instalments	New	5	8	5	\Release 1\Sprint 2
8	As a Logged in Customer I want to Pay my Mobile Bill	New	5	8	5	\Release 1\Sprint 2
9	As a Logged in Customer I want to See Currency Rates	New	5	5	5	\Release 1\Sprint 2
10	As a Logged In Customer I want to Transfer Money between my accounts	New	4	8	4	\Release 1\Sprint 3
11	As a Logged In Customer I want to Add Local & International Beneficiaries	New	4	13	4	\Release 1\Sprint 3
12	As a Logged in Customer I want to Change my Password	New	4	3	4	\Release 1\Sprint 2
13	As a registered online banking cuotomer I want to recover my Password if I forget it	New	4	5	4	\Release 1\Sprint 3
14			Total	76		

Zdroj: (Kenda,2012)

Podmnožinou Product Backlogu je Sprint Backlog. Obsahuje pouze ty položky Product Backlogu, které lze dokončit během sprintu.

Obrázek 5: Ukázka sprint backlogu



Zdroj: (Scrum.org, 2020 a)

Epic

V agilním vývoji představuje Epic řadu příběhů uživatelů, které sdílejí širší strategický cíl. Když několik Epiců sdílí společný cíl, jsou seskupeny do stále širšího obchodního cíle, který se nazývá téma.

Dalším důležitým rozdílem je, že uživatelský příběh lze dokončit v časovém rámci agilního sprintu. Epic obvykle vyžaduje vývojovou práci zahrnující několik sprintů.

Story point

Tradiční softwarové týmy poskytují odhady v časovém formátu: dny, týdny, měsíce. Mnoho agilních týmů však přešlo na story pointy. Story pointy jsou měrnými jednotkami pro vyjádření odhadu celkového úsilí požadovaného k plné implementaci položky nevyřízeného produktu nebo jiné práce. Týmy přidělují body příběhu ve vztahu ke složitosti práce, množství práce a riziku nebo nejistotě. Hodnoty jsou přiřazeny tak, aby efektivněji rozdělovaly práci na menší části, aby mohly řešit nejistotu. To v průběhu času pomáhá týmům pochopit, kolik toho v daném časovém období mohou dosáhnout, a vytváří konsensus a závazek k řešení. Může to znít neintuitivně, ale tato abstrakce je ve skutečnosti užitečná, protože tlačí tým, aby učinil tvrdší rozhodnutí ohledně obtížnosti práce.

3.2.5 Kanban

Louis Raymond popisuje ve své knize Kanban jako nejednoduší doplňovací systém, kterému lze koncepčně porozumět, ale přesto je nejsložitější na úspěšnou aplikaci do podniku. Jak v knize dále uvádí Kanban zavedl poprvé do výrobního průmyslu Taiichi Ohno, který ho využil v automobilce Toyota. Tento koncept se osvědčil, proto ji v roce 2004 David J. Anderson zkusil poprvé uplatnit v oblasti IT, vývoje software a znalostních prací. (Louis, 2006)

Kanban má pouze tři pravidla, kterými jsou: Vizualizace, Minimalizace času průchodu a limitace Work in Progress. (Louis, 2006)

Kanban vs Scrum

Zatímco Scrum dává důraz na plánování a nepřeje si, aby v průběhu sprintu probíhaly změny v plánování tak podle metodiky Kanbanu je připuštěno, že změny můžou

probíhat častěji, tedy i v průběhu sprintu. Kanban neřeší délku trvání cyklu, nedefinuje role ani meetingy, proto se považuje za jednodušší metodiku, než je Scrum. (Vítek, 2012)

3.2.6 Extrémní programování

Extrémní programování je agilní framework pro vývoj softwaru, jehož cílem je produkovat kvalitnější software a vyšší kvalitu života pro vývojový tým. Extrémní programování je nejkonkrétnější z agilních frameworků, pokud jde o vhodné technické postupy pro vývoj softwaru. Extrémní programování vymyslel a v 90. letech představil, s cílem najít způsoby, jak rychle psát vysoce kvalitativní software a být schopen přizpůsobit se měnícím se požadavkům zákazníků, softwarový inženýr Ken Beck. Extrémní programování je sada technických postupů. Při provádění těchto postupů musí vývojáři jít nad rámec svých možností. Odtud pochází „extrém“ v názvu rámce. (Grossmann,2020)

3.2.6.1 Hodnoty extrémního programování

Extrémní programování se řídí pěti základními hodnotami. Jedná se o komunikaci, jednoduchost zpětnou vazbu, kuráž a respekt (Grossmann,2020):

Komunikace

Vývoj softwaru spoléhá na komunikaci při přenosu znalostí z jednoho člena týmu na všechny ostatní v týmu. Extrémní programování zdůrazňuje důležitost vhodného druhu komunikace – osobní diskuse pomocí bílé tabule nebo jiného mechanismu kreslení.

Jednoduchost

Účelem této hodnoty je vyhnout se plýtvání a dělat jen naprosto nezbytné věci, jako je udržovat návrh systému co nejjednodušší, aby byla snazší jeho údržba, podpora a revizi.

Zpětná vazba

Prostřednictvím neustálé zpětné vazby o svém předchozím úsilí mohou týmy identifikovat oblasti pro zlepšení a revidovat své postupy. Zpětná vazba také podporuje jednoduchý design.

Kuráž

Členové týmu se přizpůsobují změnám, které nastanou, a přebírají odpovědnost za svoji práci. Říkají pravdu o svém pokroku – neexistují žádné „lži“ ani výmluvy za to, že selhali. Není důvod se bát, protože nikdo nikdy nepracuje sám.

Respekt

Extrémní programování podporuje mentalitu „všichni za jednoho a jeden za všechny“. Každá osoba v týmu, bez ohledu na hierarchii, je za své příspěvky respektována. Tým respektuje názory zákazníků a naopak.

3.2.7 Vývoj řízený testy

Vývoj řízený testy neboli Test Driven Development je přístup k vývoji softwaru, ve kterém jsou vyvíjeny testovací případy, které specifikují a ověřují, co bude kód dělat. Zjednodušeně řečeno, testovací případy pro každou funkcionalitu jsou vytvořeny a testovány jako první, a pokud test selže, je napsán nový kód, aby vyhověl testu a aby byl kód jednoduchý a bez chyb.

Test-Driven Development začíná návrhem a vývojem testů pro každou malou funkčnost aplikace. TDD dává vývojářům pokyn, aby napsali nový kód, pouze pokud selhal automatický test. Tím se zabrání duplikaci kódu. Plnou formou TDD je vývoj zaměřený na testování. (Sirkovský, 2006)

3.2.8 SAFe

Metodika SAFe neboli Scaled Agile Framework byla představena v roce 2011. Za celým nápadem potřebným pro vznik stály Dean Leffingwell a Drew Jemilo. Jak uvádějí na svém webu (Dean a Jemilo,2020), původně byl tento framework představen v knize Leffingwella s názvem „Agile Software Requirements“ pod názvem „Agile Enterprise Big Picture“.

SAFe je navržen tak, aby poskytoval týmu flexibilitu a pomáhal zvládat některé výzvy, které mají větší organizace při agilním cvičení. Je koncipován nikoli jako jediná metodologie, ale jako široká znalostní základna osvědčených postupů, které týmy používaly k poskytování úspěšných softwarových produktů. Zatímco ostatní metodiky se soustředí na jednotlivé týmy SAFe se zaměřuje na agilní prostředí v celé organizaci a propojení jednotlivých agilních týmů. Agile je způsob práce, myšlení, Scrum je rámec do

značné míry založený na agilních principech a hodnotách a SAFe je na druhé straně škálovací rámec, který se používá pro implementaci agile na podnikové úrovni. (Dean a Jemilo,2020)

Principy SAFe metodiky

SAFe je postaven na devíti klíčových principech odvozených z agilních principů. Tyto principy definuje společnost Scaled Agile Framework takto (Scaled Agile Inc., 2021b):

1. Vezměte ekonomický pohled, abyste zajistili optimální dodací lhůtu, a přitom poskytli nejlepší kvalitu a hodnotu.
2. Implementujte systémové myšlení do všech aspektů vývoje.
3. Předpokládejte tržní a technickou variabilitu zachováním možností a podporou inovací.
4. Budujte postupně pomocí rychlých integrovaných cyklů učení, které umožňují zpětnou vazbu od zákazníků a snižují rizika.
5. Založte milníky na objektivním odhadu a vyhodnocení pracovních systémů, abyste zajistili ekonomický přínos.
6. Chcete-li povolit nepřetržitý tok, omezte množství nedokončené výroby, snižte velikosti dávek a spravujte délky front.
7. Použijte kadenci (načasování), synchronizujte s tvorbou více domén, abyste rozpoznali obchodní příležitosti a podle potřeby umožnili nápravná opatření.
8. Uvolněte skutečnou motivaci znalostních pracovníků k dosažení svého neviditelného potenciálu.
9. Decentralizovat rozhodování, aby se stala pružnější a efektivnější.

Program Increment planning

Program Increment planning, zkráceně PI planning je srdcem agilní SAFe metodiky. Událost PI planning je dvoudenní cílené plánování se všemi týmy, zúčastněnými stranami a vlastníky / manažery produktů na jednom místě, aby se zkontrolovalo nevyřízené množství Backlogu a určil se směr, kterým se bude program dále ubírat. Tato událost se obvykle odehrává každých osm až dvanáct týdnů a může být významnou výzvou pro velké týmy, které jsou rozmístěny po celé zemi nebo dokonce po celém světě. (Scaled Agile Inc., 2021a)

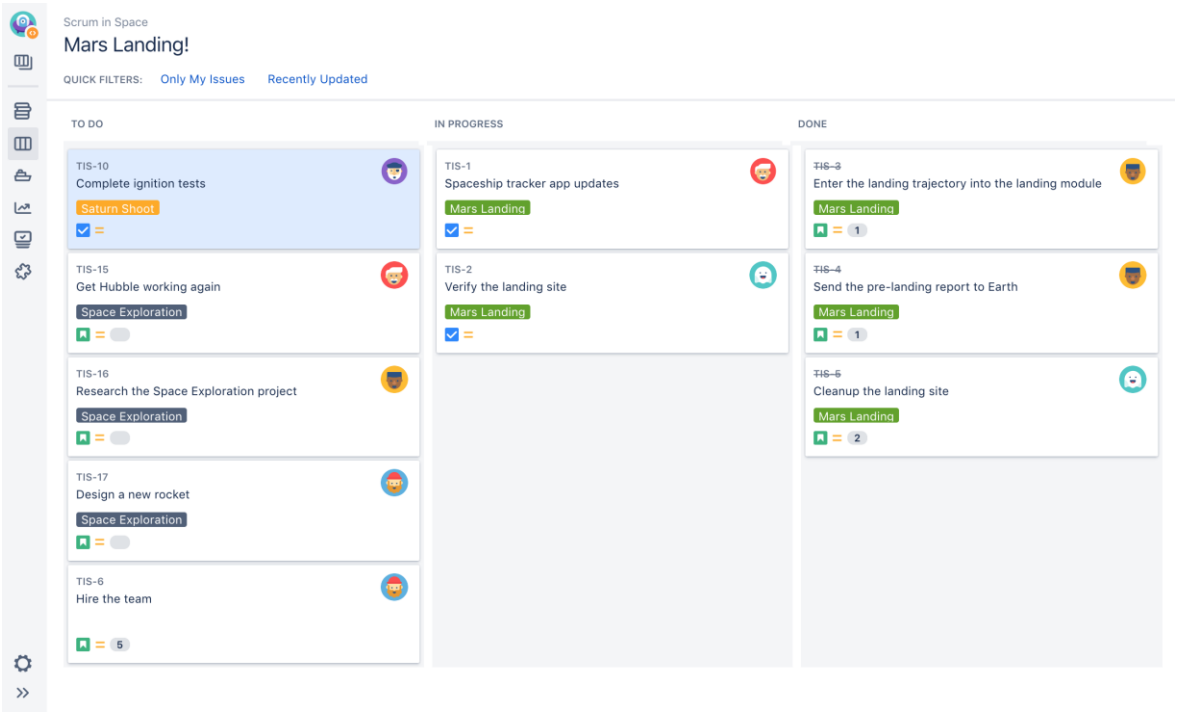
3.3 Programy pro podporu agility

Vedení agilního týmu není samo o sobě snadné. Pro splnění cílů je potřeba správné plánování. Aby toto plánování bylo co nejjednodušší a nejrychlejší je dobré mít po ruce správný software, který tuto práci usnadní.

3.3.1 JIRA

Jeden z nejznámějších software pro podporu vedení projektu. Software byl vyvinut australskou společností Atlassian, která uvádí, že se jedná o agilními týmy nejpoužívanější software. Tento software se používá pro sledování chyb, sledování problémů a správu projektů. Umožňuje transparentní sdílení úkolů, například díky přehlednému Scrum boardu (obrázku č. 6) a Kanban boardu, kde můžete vidět všechny prováděné činnosti a pokrok projektů. (Atlassian, 2020a)

Obrázek 6: Ukázka Scrum boardu v JIRA

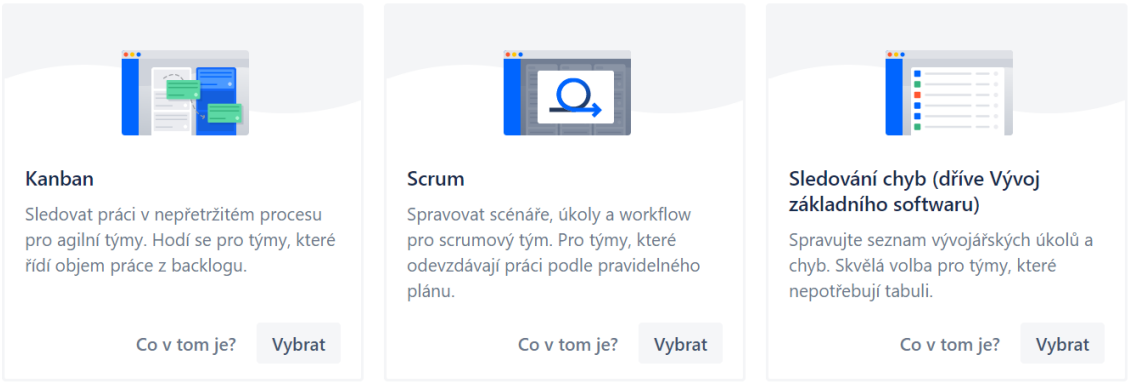


Zdroj: (Atlassian, 2020a)

Při tvorbě nového projektu si lze v JIRA vybrat šablonu (obrázek č. 7), podle typu agilní metodiky, kterou tým využívá. Vybírat lze šablonu Kanban, pro řízení práce

z backlogu, a šablony Scrum v případě nutnosti odevzdávání práce podle pravidelného plánu. Pokud tým nepotřebuje tabuli, může si vybrat šablonu pro sledování chyb.

Obrázek 7: Výběr šablony v JIRA



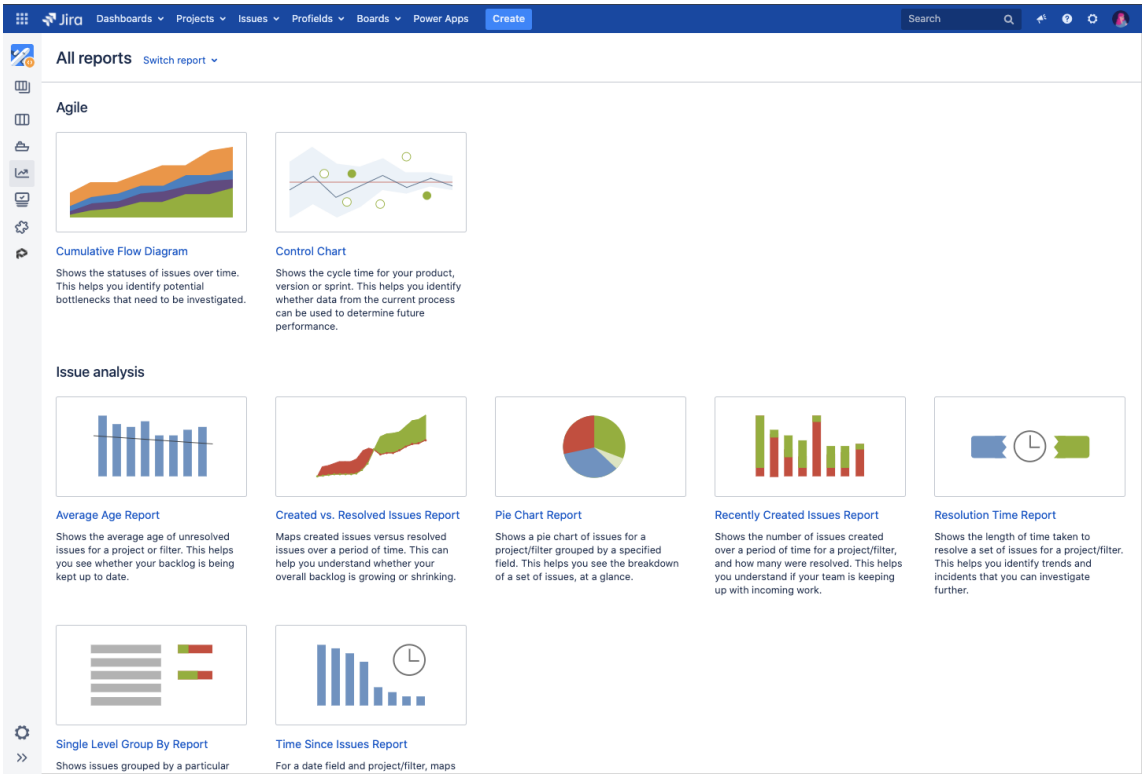
Zdroj: vlastní zpracování z aplikace JIRA

3.3.1.1 Reportování v JIRA

JIRA poskytuje v rámci projektu různé typy reportů. Tím pomáhá analyzovat pokrok, problémy, zastávky a včasnost jakéhokoli projektu. Pomáhá také analyzovat využití zdrojů. Reporty v JIRA jsou rozdělené dle výběru šablony.

Jak je zřejmé z obrázku č. 8, když nakonfigurována tabule Kanban je k dispozici v první řadě Kumulativní report a Control Chart report.

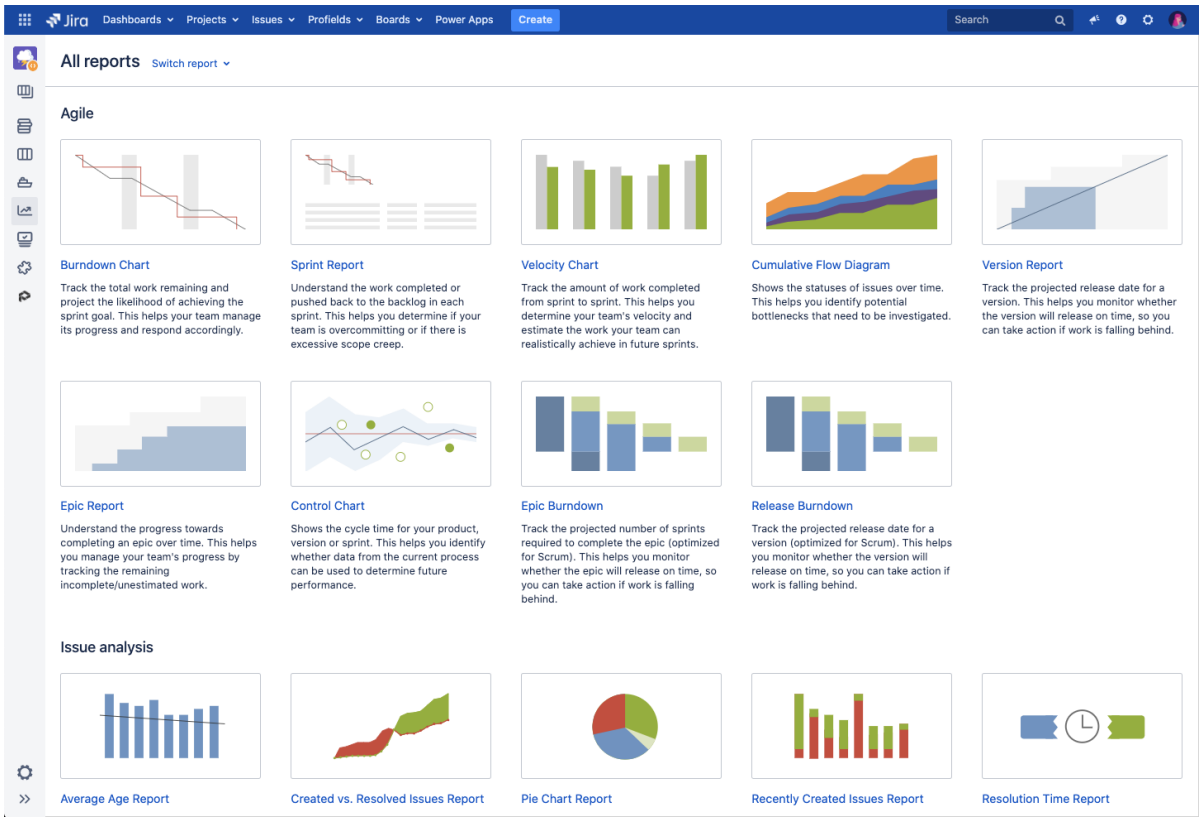
Obrázek 8: Ukázka reportů v Jira (Kanban šablona)



Zdroj: (Diaz, 2020)

Pro Scrum tabuli je k dispozici daleko větší množství reportů. Scrum reporty jsou určeny k poskytování informací o sprintech nebo rychlosti týmu, a pomáhají předvídat budoucí sprinty na základě odhadů z minulých sprint. K dispozici je Sprint report, graf rychlosti (Velocity chart), cumulative flow diagram, Epic report a mnoho dalších grafů, které lze vidět na obrázku č. 9. (Diaz, 2020)

Obrázek 9: Ukázka reportů v Jira (Scrum šablona)



Zdroj: (Diaz,2020)

Burndown Chart

Burndown chart je grafické znázornění toho, jak rychle tým pracuje na uživatelských příbězích zákazníka, agilní nástroj, který se používá k zachycení popisu funkce z pohledu koncového uživatele. Burndown graf ukazuje celkové úsilí oproti množství práce pro každou iteraci. (Sochová, 2008)

Množství zbývající práce se zobrazuje na svislé ose, zatímco čas, který uplynul od zahájení projektu, je umístěn vodorovně na grafu, který ukazuje minulost a budoucnost. Zobrazí se graf Burndown, aby jej mohli vidět všichni členové týmu, a je pravidelně aktualizován, aby byl přesný. Pro Burndown chart existují dvě varianty zobrazení. Sprint burndown slouží ke zobrazení zbývající práce v iteraci a Product Burndown se používá pro sledování zbývající práce na celém projektu. (Diaz, 2020)

Velocity Chart

Podle Velocity grafu se odhaduje výkonnost týmu. Graf pomáhá vizualizovat celkový stav práce a kolik práce může agilní tým dokončit v budoucích sprintech. Graf rychlosti je graf, který umožní snadno zobrazit odhadované Story pointy oproti skutečným dokončeným bodům příběhu. Storypointy se měří na svislé ose a dokončené sprinty se zobrazují na vodorovné ose. (Šimůnek, 2019)

Sprint report

Sprint report zobrazuje seznam problémů v každém sprintu. Jedná se o užitečný nástroj pro Sprint Retrospective schůzky a také pro kontroly průběhu ve sprintu. Zahrnuje burndown graf a souhrnnou tabulku se seznamem všech problémů ve sprintu a jejich stavů, jak je znázorněno na obrázku č. 10. (O'Reilly, 2020)

Obrázek 10: Ukázka Sprint Reportu

Zdroj: (O'Reilly, 2020)

Šedá čára je vodítko, které vychází z celkového odhadu problémů na začátku sprintu, na nula problémů na konci sprintu. Šedá čára zůstává mimo pracovní den plochá. Červená čára představuje skutečnou práci odvedenou během sprintu – čerpá se z celkového odhadu problémů na začátku sprintu a pohybuje se nahoru nebo dolů, podle toho, jak tým

pracuje na problémech ve sprintu. Červená čára zobrazuje aktuální celkový odhad nevyřešených problémů v kterémkoli bodě sprintu. Odráží také problémy, které jsou přidány nebo odebrány ze sprintu. (O'Reilly, 2020)

Kontrolní diagram

Kontrolní diagram pomáhá zjistit, zda lze data z aktuálního sprintu použít k určení budoucího výkonu. Čím menší je odchylka v době cyklu problému, tím vyšší je důvěra v použití průměru (nebo mediánu) jako ukazatele budoucího výkonu.

Pro výpočet klouzavého průměru se používá 20 % z právě zobrazených problémů (jedná se vždy o lichý počet a minimálně pět problémů). Modře stínovaná oblast kontrolního diagramu představuje směrodatnou odchylku – tj. Míru odchylky skutečných dat od klouzavého průměru. (Atlassian, 2020b)

Kumulativní diagram

Kumulativní vývojový diagram zobrazuje různé stavy pracovních problémů. Vodorovná osa X označuje čas a svislá osa Y označuje počet evidovaných problémů. Barevně jsou v grafu označeny stavy problémů. (Atlassian, 2020b)

3.3.2 Azure DevOps

Azure DevOps poskytuje vývojářům služby, které podporují plánování práce, spolupráci na vývoji kódu a vytváření a nasazování aplikací. Azure také spojuje jednotlivé vývojáře, projektové manažery a testery, tak aby co nejlépe dokončili vývoj softwaru. Dále umožňuje organizacím vytvářet a vylepšovat produkty rychlejším tempem ve srovnání s tradičními přístupy k vývoji softwaru.

Azure poskytuje integrované funkce, které lze využívat podle potřeby daného projektu. Azure Repos je sada nástrojů pro správu verzí, které vám umožňují v průběhu času sledovat veškeré změny provedené v kódu aplikace. Azure Pipelines automaticky vytváří a testuje kódy projektů, tak aby je zpřístupnil ostatním. Velkou výhodou je, že funguje téměř s jakýmkoliv jazykem, takže pomáhá nalézt chyby již na začátku vývojového cyklu. Další integrovanou funkcí je webová služba Azure Boards, kde mohou týmy spravovat své softwarové projekty. Poskytuje také bohatou sadu funkcí, včetně podpory pro Scrum a Kanban, přizpůsobitelné řídící panely a integrovaný reporting.

Kvalita manuálních a explorativních testů je zásadním aspektem softwarových systémů, proto je zde další důležitá funkce, a to funkce Azure Test Plans. Tato funkce představuje snadno použitelné řešení pro vytváření, provádění daných testů a shromažďování zpětné vazby od zúčastněných stran. (Microsoft, 2020)

3.3.3 Zoho Sprints

Jedná se o online agilní řešení pro správu projektů navržené tak, aby pomohlo agilním týmům naplánovat jejich projekt, sledovat jejich pokrok a včas dodat vhodný produkt. Zoho Sprints je součástí sady softwarových řešení od společnosti Zoho, která má podnikům pomoci v různých aspektech jejich provozu. V softwaru lze sledovat progres pomocí Scrum Boardu, plánovat schůzky a sledovat výkonost projektů pomocí grafů. V nástroji je k dispozici graf Velocity a Burndown. Software je také přístupný z mobilních zařízení se systémem Android a iOS. Bohužel není možné v Zoho Sprints tvořit testovací scénáře a cykly, ale pouze tvořit chyby. Tento software lze integrovat s testovacími nástroji jako je například JIRA. (Zoho, 2021)

4 Vlastní práce

Vlastní práce se skládá ze dvou částí. První část se zabývá rozborem fungování agilních týmů, kterých byl autor členem v rámci diplomové praxe. V každém týmu byla vykonávána praxe po dobu tří měsíců. V této části jsou analyzovány problémy týmů a problematika testování v těchto týmech. Vzhledem k požadavkům dotčených firem nebude v diplomové práci uveden název firem a budou anonymizovány názvy všech agilních projektů a pracovníků, kterých se práce týká. Ostatní informace budou uvedeny dle reality. Všechny vybrané agilní týmy jsou součástí společností fungujících v bankovním sektoru. V druhé části vlastní práce je vytvořen obecný návrh a doporučení pro zlepšení testování v agilních týmech.

4.1 Agilní tým č. 1

Agilní tým 1 je pracovní název pro tým zabývající se správou a aktualizací veřejného webu bankovní společnosti. Tento tým funguje pomocí metodiky SAFe. Jak bylo řečeno v teoretické části v užším agilním týmu by měli být pouze zaměstnanci alokovaní na 100 %, nicméně tato banka si určila, že ten, kdo má alokaci větší než 50 % je členem týmu a účastní se tak všech SAFe ceremonií.

4.1.1 Cíle projektu

Cílem projektu agilního týmu č. 1 je usnadnění a zrychlení online procesů přihlašování. K tomuto se využije zpracování identifikace klienta Anti Money Laundering (dále AML) výhody služeb Bank ID (zejména Know your customer – dále KYC). Za tímto účelem projekt vytvoří nezbytné technické prostředky, aby bylo možné využití Bank ID KYC jako přijímací banku (poskytovatel služby v terminologii Bank ID), tzn. že služba bude zavolána prostřednictvím standardního rozhraní, které Bank ID musí poskytnout a dále zpracovat data, jež služba vrací pro každého jednotlivého uživatele či klienta.

Vzhledem k tomu, že hlavním důvodem pro budování této schopnosti je zapojení klientů pomocí tohoto nového postupu, bude projekt integrovat tento způsob identifikace AML do procesu registrace Smartbanking a CAO (proces onboardingu webu). Integrace do jiných procesů není v rozsahu a rozpočtu tohoto projektu.

Postup (sken identifikačních dokumentů a mikroplatba z jiné banky) v procesu zůstane k dispozici a tento projekt v něm nebude aplikovat žádné změny kromě toho, že umožní uživateli vybrat, jaký způsob identifikace AML použít.

4.1.2 Členové agilního týmu

Dle teoretických východisek by mělo být v agilním týmu alokováno minimálně 5 a maximálně 9 členů. Tento tým, tj. agilní tým č. 1, je nastaven na větší počet lidí v týmu (disponuje 20 členy) z důvodu přetlaku práce. V případě, že by byl tento tým rozdělen na dva týmy, tak je velice pravděpodobné, že by v budoucí době týmy neměly dostatečné množství práce. Takto se může tým zbavit lidí, kteří již nejsou na projektu potřeba.

V tomto týmu je tedy alokováno 20 členů a každý z nich má vlastní roli, kterou zastává. Product Owner má na tomto projektu alokaci více jak 50 %, analyzuje potřeby ostatních projektů závislých na dodávce tohoto týmu a na základě se toho stará o prioritizaci tasků z backlogu a zaštiťuje rozpočet celého projektu. Nedílnou součástí agilní týmu je Scrum Master, který je též alokován na 50 % a stará se o všechny členy týmu, motivuje ostatní členy k efektivnosti plnění práce, tj. tato role má podobné charakteristiky jako z teoretických východisek.

Dalšími členy jsou čtyři Java vývojáři, kteří se dělí na pozice dle zkušeností, tzn. Junior/Middle/Senior. Senior Java vývojář ještě vykonává z 50 % funkci SAR, ostatní Java vývojáři se věnují své funkci. Analytik je též součástí týmu a má na starost rozpad epic na story. V týmu jsou dále tři členové Portal User Interface (dále PUI), jeden User Experience designer (dále UX designer) a Web Content Manager (dále WCM), který je alokován na 60 %, jehož hlavním úkolem je spravování obsahu na webu a jeho aktuálnosti, dále tento člen plní roli testera.

Členové, kteří disponují alokací 30 % a méně, tak tvoří širší sestavu týmu, jehož součástí jsou dva UX designéři (jejich alokace je definována na 20 %) a dva členové za Digital Tracking (získávají obsah o uživatelích) s alokací na 30 %. Součástí týmu jsou tři WCM a jeden aplikační inženýr – jejich celková alokace činí 30 %.

4.1.3 Nástroje a techniky agilního týmu č. 1

V tomto agilním týmu je definována hlavní výhoda, že tým využívá vlastní systém, a tím nezasahuje do práce ostatních týmů a není nijak omezován. Předchozí nastavení týmů

se totiž jevilo jako problematické kvůli špatné komunikaci. PI planning session jednou za 3 měsíce nestačil k tomu, aby se týmy mezi sebou domluvily.

Členové týmu začínají denním stand-upem neboli denním scrumem. Účastní se jich i Web Content Manažeři, kteří mají alokaci menší než 50 %, jelikož se starají o provozní část veřejných webů a denní aktualizace webových stránek, a tudíž potřebují mít přehled o tom, jak se projekt vyvíjí.

Tým má čtrnáctidenní sprint. Kapacitu týmu evidují ve story pointech. Ty mají škálu 1-20 a na začátku bylo určeno, že 1 je pracnost drobné úpravy na kalkulačce a 20 je dodání nové funkcionality na backendu, dále BE. Pokud se Story týmem nacení na více jak nad 20 story pointů, tak se rozdělí na dvě dodávky. Tým při plánování počítá s tím, že 30 % kapacity týmu je na řešení technického dluhu, dokumentace, vzdělání, bugy, incidenty a zdokonalení.

Aby ceremonie byla alespoň trochu dodržena tak prvních 15 minut mohou hovořit pouze základní členové a poté se mohou vyjadřovat lidé s nižší alokací. Zprvu byl problém se v tomto počtu lidí dostat pod půl hodinové schůzky, ale po zásahu Scrum Mastera jsou schůzky efektivnější a do patnácti minut se stihnou probrat všechny důležité věci. Po patnácti minutách je prostor pro dořešení otázek, dohod a problémů s časem a dodávkou. V případě, že není dostatek času a je potřeba aby někteří členové pracovali na dodávce děje se i to, že ze čtyř Java vývojářů v rámci jedné iterace dorazí na schůzi pouze jeden člen a probrané informace ostatním vývojářům sdělí.

4.1.4 Typy testů a úrovně testů

V této sekci bude popsáno samotné testování, typy testů a jejich úrovně. Úrovně testů jsou následující:

- UT – Unit testy
- KIT – Integrační testy komponent
- SIT – Systémové (Systémově-integrační) testy
- UAT – Uživatelské akceptační testy

Využití úrovní se může lišit dle konkrétního projektu, resp. jeho fáze dodávky.

4.1.4.1 Matice testů

Na základě pozorování týmu č. 1, a nahlížení do dokumentace k testování (Master Test Dokumentu a Testovacích scénářů) byla vytvořena matice testů, ve které byly využívány kombinace úrovní a typů testů v průběhu developmentu.

Obecně budou realizovány následující kombinace úrovní a typů testů:

Tabulka 2: Přehled využívaných typů a úrovní testů agilním týmem 1

	UT	KIT	SIT	UAT
Funkční testy	x	X	x	x
Security testy			x	x
Data testy		x	x	x
Performance testy			x	x
Usability testy			x	x
Regresní testy			x	x
Free testy			x	x

Zdroj: Vlastní zpracování

4.1.4.2 Detailní popis prováděných typů testů

Unit testy (dále UT) – tyto testy jsou plně v kompetenci vývojových týmů. Žádný z projektů neplánuje zapojení dedikovaného testovacího týmu do UT. Vývoj s každou dodávkou předává informace report s informacemi o:

- seznam funkcionalit
- seznam známých omezení
- seznam známých defektů

Funkční testy – v úrovni UT jsou v zodpovědnosti jednotlivých vývojových týmů. V úrovni KIT a SIT je provádí IT testovací tým a v úrovni UAT jsou v zodpovědnosti business user (dále jako BUS) testovacího týmu ve spolupráci s IT testovacím týmem.

Security testy – v první fázi projektu je alokován security analytik, dále SECAN, který dohlíží na security standardy.

Data testy – charakter dodávky nevyžaduje tento typ testu.

Performance testy (perf. testy) – performance testy jsou organizovány centralizovaně pod programem. Otestování dílčích projektových dodávek a následně také řešení jako celku zajišťuje dedikovaný performance tým.

Plánování a řízení spadá do kompetence a odpovědnosti performance test koordinátora. Je odpovědností každého Scrum Mastera oslovit programový performance tým a domluvit si testy své dodávky.

Usability testy – charakter dodávky nevyžaduje tento typ testu.

Disaster Recovery – projekt nepřidává žádný nový systém, proto v rámci projektu nebude prováděn tento typ testu.

Regresní testy – provádí IT testovací tým na INT prostředí a IT i BUS testovací tým na ACC prostředí.

Free testy – provádí IT testovací tým na INT prostředí a IT i BUS testovací tým na ACC prostředí.

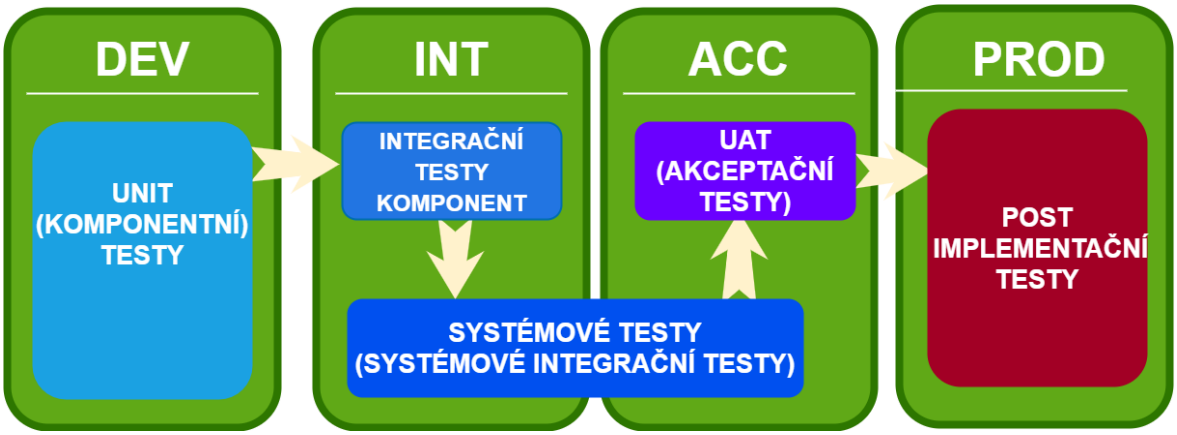
Bezpečnostní (security) testy – V první fázi projektu byl alokován SECAN, který dohlíží na security standardy. Ve fázi 2 bude rozhodnuta, zdali se finální security testy vykonají externě dohromady nebo interně.

4.1.5 Prostředí

- **INT** – V Integračním prostředí jsou testovány tyto úrovně: UT, KIT a SIT a tyto typy testů: funkční a regresní testy
- **ACC** – V Akceptačním prostředí jsou testovány tyto úrovně: SIT, UAT a tyto typy testů: funkční, regresní testy a performance testy
- **PROD** – smoke testy na produkci v rámci release víkendu

Využita budou všechna neprodukční prostředí DEV, INT, ACC a v poslední fázi programu též produkční prostředí v režimu PILOT.

Obrázek 11: Využití prostředí Agilním týmem 1



Zdroj: Vlastní zpracování

Vzhledem k předpokladu, že většina dodávek je svým rozsahem nad rámec časování standardního releasu, bude požadováno pro možnost kontinuálního testování dočasné dedikování/vybudování neprodukčních prostředí pouze pro potřeby programu. Konkrétní požadavky se vydefinují na projektové úrovni s ohledem na efektivitu.

4.1.6 Výstupy týmu

Tato sekce bude definovat výstupy a data agilního týmu č. 1, které budou dále detailněji popsány.

Pro řízení projektu a sledování problémů používá tým software JIRA ve verzi 8.13.2 s pluginem Zephyr, který klade důraz na testování. Díky tomuto pluginu je testování integrováno do projektových cyklů a umožňuje sledovat kvalitu softwaru. Tým v JIRA řídí své testovací cykly, sleduje protestovanost, eviduje chyby a pro řízení celého projektu využívá agilních nástrojů, jako je Backlog, aktivní sprinty a reporty, které Jira nabízí.

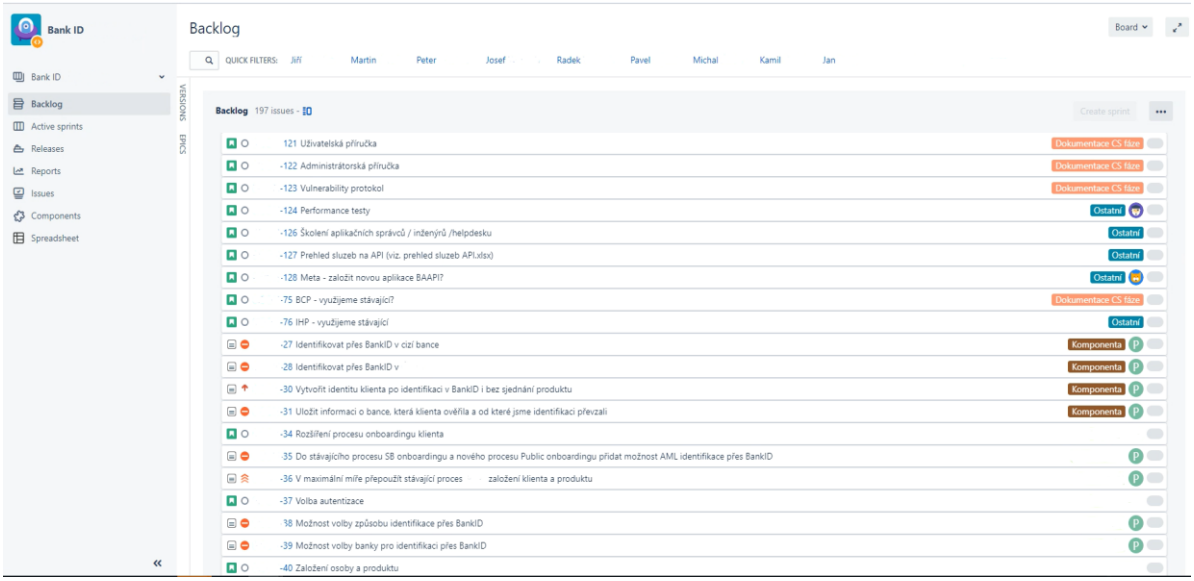
- Nástroje: JIRA, reportovací nástroje

4.1.6.1 Backlog

Jak můžeme vidět na obrázku č. 12 tým eviduje v backlogu všechny problémy projektu. Momentálně jich má 197. Kromě Story (zelený čtvereček) a požadavků jsou v Backlogu evidovány i chyby. U každého problému je vidět i jeho priorita, která se škáluje od nejdůležitější (tzv. Blocker) po nedůležité (tzv. Trivial). Jednotlivé problémy lze

přiřadit řešiteli, který je pak vidět u problému. Prioritu a zařazování do sprintu má na starosti Product Owener.

Obrázek 12: Backlog týmu 1

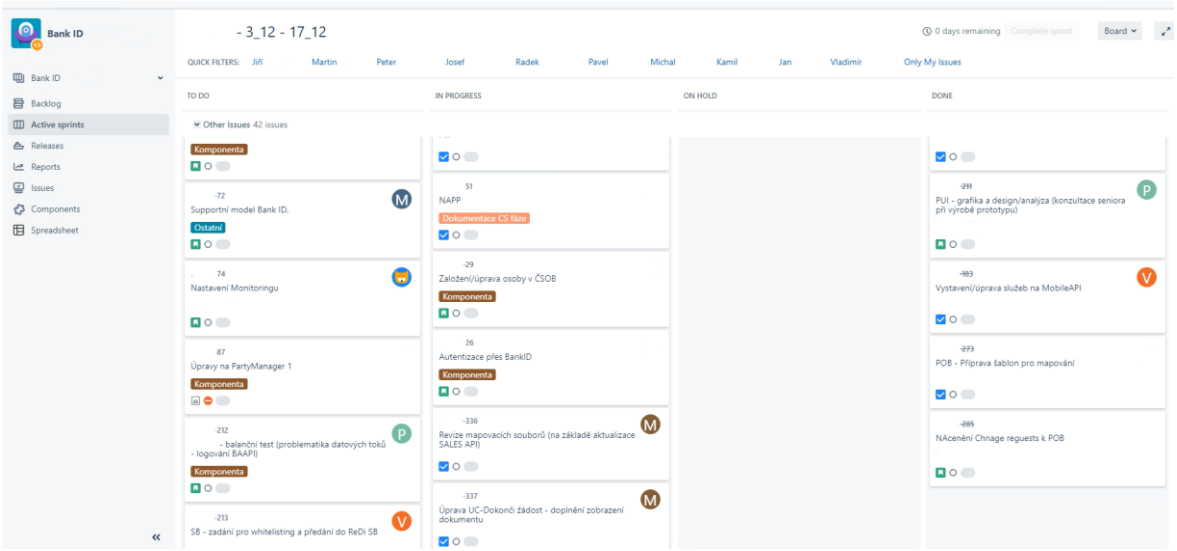


Zdroj:vlastní zpracování

4.1.6.2 Sprinty

Na obrázku č. 13 lze vidět naplánovaný právě aktivní čtrnáctidenní sprint. Tým tento přehled využívá pro rychlí přehled nad děním ve sprintu. Product Owner a Scrum Master na základě tohoto přehledu vědí, kde je potřeba zabrat a ihned směřují tým k lepším výsledkům a k práci nad důležitými problémy, které mají vyšší prioritu. Na tomto obrázku si lze všimnout, že v aktivních sprintech jsou zaznamenány problémy, které teprve mají být ve sprintu řešeny (TO DO), rozpracované problémy (IN PROGRESS), případně problémy, které momentálně nelze řešit, protože jsou z nějakého důvodu pozastaveny (ON HOLD) a problémy které jsou již vyřešeny (DONE). Každý problém má přiřazený svého řešitele (u některých musel být řešitel smazán kvůli osobním údajům). To pomáhá členům týmu v rychlé orientaci.

Obrázek 13: Ukázka aktivních sprintů týmu č. 1

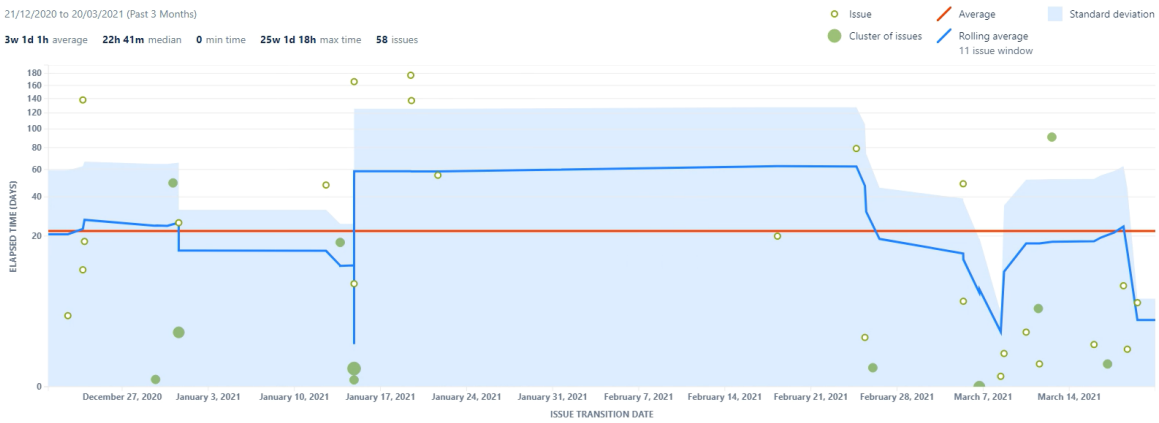


Zdroj: vlastní zpracování z platformy JIRA

4.1.6.3 Reporty

Každý tým by měl ke kontrole práce a zlepšování svých výsledků využívat reporty. Bohužel tento tým v JIRA nepracuje s nejdůležitějšími grafy jako je Velocity graf, který by jim mohl ukázat celkový stav práce a pomoci lépe nastavit cíle v budoucích sprintech. Podobný přehled může ovšem tým využít z kontrolního grafu. Na obrázku č. 14 lze vidět kontrolní graf pro předchozí tříměsíční období. Tento graf odpovídá předchozím šesti sprintům, které proběhly před PI planningem. graf týmu pomáhá retrospektivně analyzovat dosavadní výkon a zjistit jestli lze data z aktuálního období použít pro určení budoucího výkonu. Z grafu lze vyčíst, že průměrná doba vyřízení problému byla 27 dní (červená čára). Dle rychlosti řešení jednotlivých problémů, které jsou v diagramu znázorněny jako jednotlivé problémy (nevyplněné zelené kolečka) nebo v případě, že byli problémy řešeny ve stejnou dobu a podobně rychle, jako skupina problémů (plné zelené kolečka), lze analýzou predikovat jak rychle budou řešeny podobné problémy.Pro lepší odhad se v grafu využívá klouzavý průměr, který je přesnější.

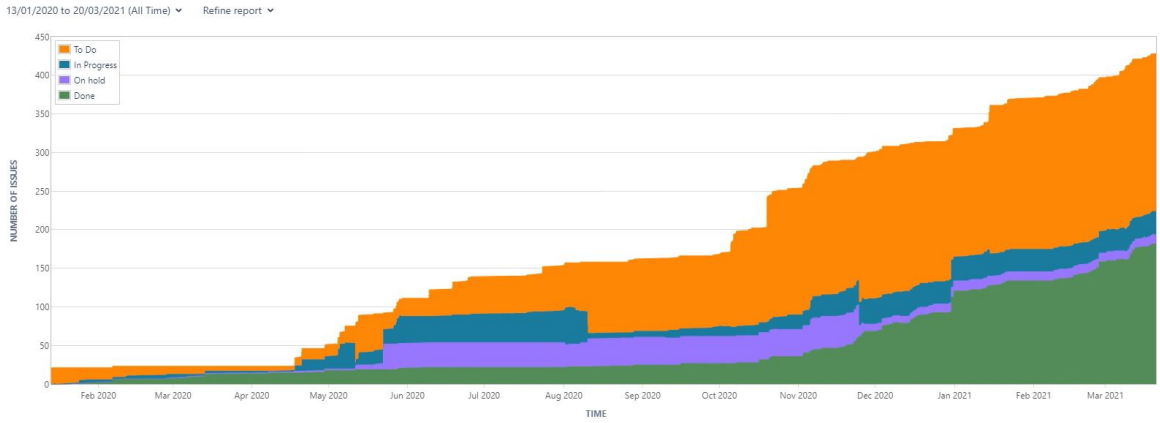
Obrázek 14: Kontrolní graf týmu č. 1



Zdroj: Vlastní zpracování

Obrázek č. 15 znázorňuje Cumulative flow diagram, který ukazuje množství vykonané práce. Jak lze vidět týmu za tři měsíce objem práce neustále narůstá. Diagram odpovídá tomu, že je projekt teprve na začátku, kdy se dotváří cíle a projekt se rozrůstá.Jako pozitvní může tým hodnotit neustále se zvyšující objem dokončených problémů.

Obrázek 15: Cumulative flow diagram tým č. 1



Zdroj: Vlastní zpracování

Tabulka 3: Přehled agilního týmu 1

	Iterace 1	Iterace 2	Iterace 3	Iterace 4	Iterace 5	Iterace 6
Počet programátorské práce v MD	25	25	25	25	25	25
Story ve sprintu	13	20	16	19	15	18
Dokončených story ve sprintu	8	12	9	12	11	14
Nedokončených story	5	8	7	7	4	4
Chybovost (%)	20	15	15	5	0	0
přepoužitého kódu (%)	0	0	0	0	0	0

Zdroj: Vlastní zpracování

V tabulce č. 3 je znázorněn průběh iterací v době pozorování. Jak je uvedeno v kapitole 3.2.2 v agilním prostředí je kladen důraz na komfort týmu. Komfortu je mimo jiné dosaženo správným řízením týmu. Tým se nesmí zbytečně zahlcovat úkoly které by nemusel dosáhnout. Proto se bere jako měřítko úspěšnosti sprintu počet dokončených a malé množství nedokončených story ve sprintu. V tabulce č. 4 je možné vidět, že sprinty byly naplánovány tak, aby měl tým malé množství nedokončených story ve sprintu. V průběhu iterací se také zmenšil počet chyb odhalených v produkčním prostředí. Dle vyjádření vývojářů nebyl v dodávkách použit žádný původní kód.

4.1.7 Analýza testingu agilního týmu č. 1

V rámci zkoumání byl autor pozorovatelem součástí týmu č. 1 po dobu tří měsíců, kde na základě dat bylo zjištěno:

Tým využívá širokou škálu testů. Aby neopoměl na žádné testy byl vytvořen Master Test Document, který Jak bylo uvedeno v teoretické části zejména unit testy, které tým využívá, jsou v agilním prostředí vhodné k úspoře času a odhalení chyb již při vývoji. Vzheldem k délce trvání projektu přistoupil projekt k využívání automatických testů v regresním testování. Tento typ testů má sice relativně velké náklady na počáteční vývoj,

ale tým si spočítal, že se jejich zavedení z důvodu opakované využívání v rámci testování v jednotlivých sprintech vyplatí. Automatické testování, týmu ušetřilo spoustu času a odhalilo velké množství chyb. Vzhledem k povaze projektu a bezpečnostnímu riziku byl tým nucen využít i Security testy, které nejsou zajišťovány přímo projektem, ale speciálním týmem dohlížejícím na bezpečnost banky. V bance je bezpečnost dodávaných produktů na prvním místě a je na ni kladen velký důraz již při vývoji testů, proto tento typ testů většinou neodhalí velké množství chyb, ale je prováděn zejména pro kontrolu. V Rámci Integračních testů komponent bylo testováno:

- rozhraní,
- infrastruktura,
- konfigurace systému,
- databázová implementace
- data konfigurace

Hlavními cíly této fáze jsou příprava integrovaných komponent na systémové testy. Integrační testy komponent odhalilo v průměru 30% všech defektů nalezených v rámci sledovaných iteracích. Tento typ testů se projevil jako velice užitečný. Včasné odhalení těchto chyb vedlo ke snížení nákladů na opravu a rychlejší dodávce kvalitního produktu.

Jak je znázorněno v tabulce č. 4 snížil se počet chyb odhalených UAT testy. Toto nebylo způsobeno omezením UAT testů, ale vytvořením automatických regresních testů, které testovali komponenty vytvořené v předchozích sprintech. Toto vedlo i ke zvýšení počtu dokončených story, protože testři měli více času na přípravu a testování nově vyvinutých funkcionalit, které se tak mohli rychleji a s větší otestovaností dodat.

Z výsledné tabulky č. 3 matice testů je zřejmé že tým, který dodává novou funkcionalitu nepodceňuje důležitost testování a toto se mu vyplácí jak pozitivní odezvou od uživatelů tak snížením nákladů na opravu chyb.

4.2 Agilní tým 2

Agilní tým 2 je smyšlený název pro bankovní tým, který se zabývá vývojem backend rozhraní pro integraci Documenta, platformy pro správu podnikového obsahu. Tento tým využívá metodiku SAFe. Součástí týmu jsou i lid s alokací menší než 0,3 FT.

4.2.1 Cíle projektu

Cílem projektu agilního týmu č. 2 je implementovat backend pro správu podnikového obsahu, tak aby bylo možné bezpečně vytvářet spravovat šablony smluv a dokumenty klientů. Banka se rozhodla pro správu těchto dat využít platformu Documentum společnosti OpenText. Z důvodu bezpečnosti dat klientů banka přistoupila k využití vlastních backendových technologií. Aktuálně tým pracuje na vylepšené stávajících služeb pro jediného konzumenta DMS Services, kterého využívá na 1IB platformě zbytek banky. V budoucnu se plánuje využívání nové architektury. Banka má přejít z SOAP operací na REST služby a využití Apache Kafka.

4.2.2 Členové agilního týmu

Tým se skládá z šesti vývojářů, dva vývojáři zastávají juniorskou pozici. Ostatní vývojáři na ně dohlíží a kontrolují jimi odvedenou práci. Jeden hlavní vývojář před releasem připravuje balíček nově vyvinutých nebo opravených služeb pro nasazení do akceptačního případně produkčního prostředí.

V týmu se dále nachází dva IT analytici, kteří vytvářejí analýzu potřeb a procesů pro nově vyvíjené funkcionality. V rámci analýzy mapují pomocí schématických diagramů informační systémy a zjišťují dopady na všechny možné systémy, kterých by se mohl vývoj nebo oprava nových funkcionalit dotknout. Analytici zároveň tvoří dokumentaci, ze které čerpají jak vývojáři, tak i testeři při vytváření testovacích scénářů.

Důležitou součástí týmu je agilní kouč, který zároveň v podstatě zastává funkci delivery manažera. Jeho úkolem je dohlížet na správné využívání agilní metodiky a pomáhat firmě s vylepšením produktivity. Agilní kouč řídí agilní ceremonie, řeší konflikty v týmu a připravuje podklady, na základě kterých přesvědčuje vedení, že nové funkcionality jsou připravené k nasazení do produkčního prostředí. V týmu se také nachází

specialista na Documentum, který má pouze 25 % alokaci. Tento člověk řeší konfigurační věci ohledně této platformy.

Nedílnou součástí celého týmu je i Test Engineer, který se stará o otestování vyvinutých funkcionalit, resp. jednotlivých problémů. Test Engineer má také na starost průběžnou přípravu automatických testů, aby pokrývali co největší oblast. Tento člen má zároveň zodpovědnost za správně provedené testy a správné výsledky testů na základě kterých vytváří report pro vedení. Report obsahuje seznam provedených testů, seznam Story, které jsou připravené k nasazení a seznam chyb, které je potřeba řešit.

Prioritizaci Story a jejich přiřazování do sprintu má v tomto týmu na starosti Product Owner. Ten také řeší požadavky zasahující do jiných týmů, účastní se skoro všech ceremonií a řeší SLA (Service-level agreement), dovolené členů týmů a jejich celkové hodnocení. V týmu se tedy celkově nachází deset lidí.

4.2.3 Nástroje a techniky agilního týmu č. 2

Z předchozí kapitoly víme, že na správné fungování týmu a zachování agility týmu dohlíží agilní kouč. Ten se v tomto týmu také stará o hladký průběh všech schůzek a zachování jejich podstaty.

Členové týmu se každé ráno účastní Stand-upu na kterém dle zavedeného pořádku každý člen říká co dělal předchozí den a na čem plánuje pracovat nyní. Každý by se měl se svojí řečí vejít do jedné minuty. V případě, že má někdo rozsáhlejší problém tak agilní kouč domlouvá řešitelskou schůzku.

Po každém sprintu, který trvá čtrnáct dní, naplánuje agilní kouč Retrospektivu. Dříve tato schůzka fungovala tak, že měl každý postupně slovo a měl sdělit, jestli nemá něco na srdci. Jen zřídka se stávalo, že si někdo stěžoval, nebo někoho chválil. Schůzka se teď odvíjí tak, že každý má prostor se přihlásit a sdělit své problémy. Schůzka se tedy spíše omezila na pochvaly od Product Ownera za uplynulý sprint.

Tým má Sprint Review schůzky spojené i s Planingem další iterace. Schůzky probíhají tak, že se kontrolují otevřené problémy z právě ukončené iterace. Člen týmu, který má problém v řešení vysvětlí, v jakém je stavu, na co se čeká a jestli má být problém přesunut do následující iterace. V další části schůzky se vybírají problémy, které je potřeba zařadit prioritně do následující iterace a na koho budou přiřazeny. Členové týmu si mohou

také vybrat, který problém chtějí do sprintu zařadit. Testing si občas nechává do sprintu zařadit problémy, které se týkají úpravy automatických testů nebo přípravu dashboardu.

Jelikož je potřeba zpracovávat i požadavky ostatních týmů jsou pořádány i takzvané Product Backlog Refinement. Jedná se o schůzku za účelem zpřesnění Produktového Backlogu. Tato schůzka se koná před každým zahájením iterace. Probírá se agenda, kterou navrhne Product Owner, většinou se jedná o problémy, jejichž řešení je vyžadováno jinými týmy. Tyto schůzky občas suplují řešitelské schůze.

4.2.4 Typy testů a úrovně testů

Agilní tým 2 využívá následující způsob rozdělení testů do úrovní:

- Vývojové testy – testy prováděné vývojáři ve vývojovém prostředí
- UAT – Uživatelské Akceptační testy prováděné v Akceptačním prostředí
- Canary release – testování na podskupině uživatelů v Produkčním prostředí
- A/B testing – Testování na polovině uživatelů v Produkčním prostředí

Využití úrovní testování se liší dle typu Softwarových změn.

4.2.4.1 Matice testů

Matice testů dle typů a úrovní využívaný agilním týmem č. 2, který se nachází v tabulce č. 4, byl vytvořen na základě tříměsíčního pozorování práce tohoto týmu a využívání jeho dokumentace. Jedná se o matici testů, které byli využívány při vývoji nového systému. V případě oprav chyb (incidentů) z produkce se neprováděl plný rozsah těchto testů.

Tabulka 4: Přehled využívaných typů a úrovní testů agilním týmem 2

	Vývojové testy	UAT	Canary release	A/B testing
Health checks testy	X	X		
Testy Funkcionality		X	X	X
Test kompatibility		X	X	X
Performance testy		X		
Security testy		X		
Smoke testy		X		
Regresní testy		X		

Zdroj: vlastní zpracování

4.2.4.2 Detailní popis prováděných typů testů

Použití Unit testů není v tomto týmu vyžadováno a dle zjištění se nestává, že by toto mohlo odhalit některé chyby dříve, ale dle jejich názoru je to zpomaluje ve vývoji a úpravě kódu a nemyslí si, že by přinesli větší užitek, než jakou dají pracnost.

Health cheks testy – tento krátký typ testu provádí vývojáři. Jedná se o pouhou kontrolu stavu, díky které se zjistí, zda vyvinutá služba je dostupná a komunikuje. Tento test se provádí před předání k testování. Výhodou tohoto testu je, že si vývojář ihned může ověřit, že služba komunikuje a tím ušetřit čas jak sobě, tak testerovi, kterému by trvalo déle zjistit proč s ním služba nekomunikuje. Zároveň se jedná zatím o jediný test, který vývojáři musí provádět. Do budoucna se plánuje vývojářům přidat povinnost provádět drobné testy komponent, které ověří správnou základní funkčnost.

Funkční testy – na úrovni vývojových testů zatím nejsou prováděny. Vzhledem ke špatným výsledkům chybovosti bude na této úrovni brzo zavedeno. Funkční testy jsou prováděny na úrovni UAT testů, a to jak testerem, tak business uživatelem. Tento typ testů se také provádí přímo v produkčním prostředí, a to dvěma způsoby v závislosti na potřebách týmu, respektive celé banky. Prvním způsobem je Canary release, kdy banka dá

k dispozici novou verzi aplikace svým zaměstnancům, rodinným příslušníkům zaměstnanců nebo zákazníkům kteří chtějí sami být zapojeni v pilotní verzi. Banka následně sleduje chování v aplikaci a hlášené incidenty. Následně se rozhodne, zda novou verzi nasadit všem, nebo zda pracovat na úpravách nebo opravách nové verze. Dalším způsobem je tzv. A/B Testing, kdy banka postupně zavede na produkci novou verzi aplikace přibližně polovině uživatelů a dále postupuje jako u Canary release metody.

Test kompatibility – jedná se o ověření funkčnosti aplikace či nově vyvinuté funkcionality na různých operačních systémech, prohlížečích a různých zařízeních v různých rozlišeních. Tento způsob testů provádí v Akceptačním prostředí jak Bussines uživatelé v rámci UAT testů, tak tester týmu. Na Produkční úrovni se sbírají a vyhodnocují data z Canary releasu či A/B testingu podobně jako u funkčních testů.

Performance testy – výkonnostní testy jsou prováděny speciálním performance týmem. Jejich plánování a řízení má na starosti koordinátor, s kterým se domluvená Scrum Master. Domluvené termíny performance testů jsou zaneseny do speciálního kalendáře, dle kterého se orientují i ostatní týmy.

Security testy – Security testy jsou vykonávány v případě, že speciální bezpečnostní tým banky vyhodnotí, že je to potřeba. Bezpečnostní tým tento typ testů zajišťuje a vyhodnocuje na úrovni UAT testování. Provádí se zejména penetrační testy a vulnerability scan. Defekty z bezpečnostních testů nejsou a nesmí být zaznamenávány v nástrojích pro podporu testování (tj. v JIRA, kterou tým využívá).

Smoke testy – Smoke testy jsou prováděny automatickým testováním. Spouštějí se po každém nasazování do akceptačního prostředí. Jedná se o testy základních funkcionalit, které ověřují, zda bylo správně nasazeno a tester může začít testovat nově dodané funkcionality. V případě že by Smoke testy neproběhly úspěšně tester neztrácí čas testováním nových funkcionalit, protože pravděpodobně nastal problém v nasazování.

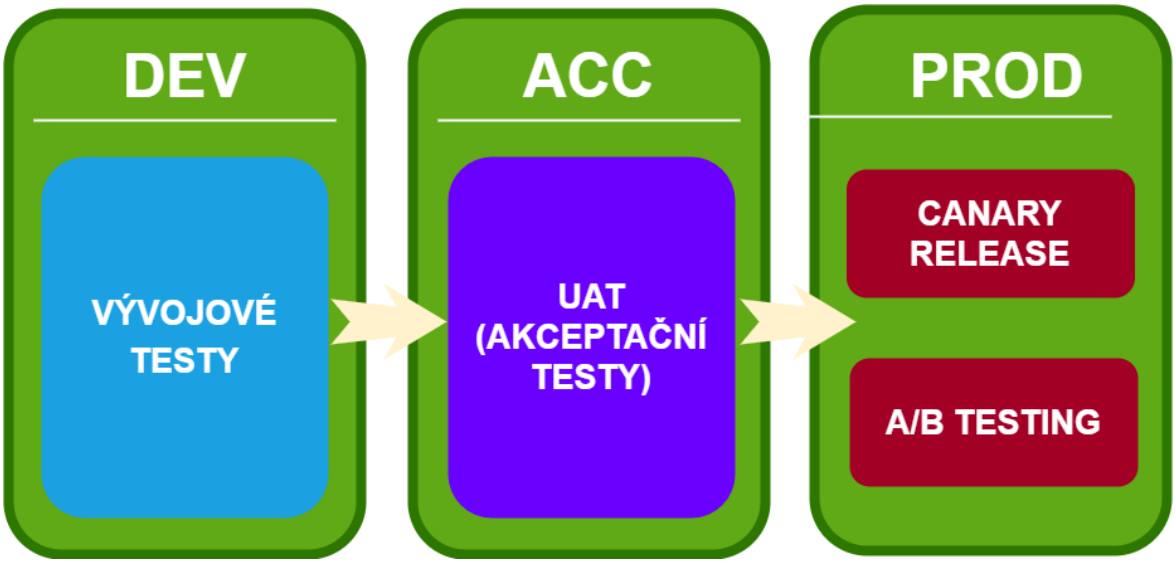
Regresní testy – tým každou iteraci regresně testuje vyvinuté funkcionality z předchozích iterací. V rámci iterace se také tvoří automatické regresní testy funkcionalit z předchozích sprintů. Toto ulehčuje práci testera, který se může věnovat testování nových věcí. Regresní testování se provádí pouze v Akceptačním prostředí.

4.2.5 Prostředí

- **DEV –** probíhají zde testy na úrovni Vývojových testů, konkrétně jde o health cheks testy
- **ACC –** V akceptačním prostředí jsou prováděny UAT testy, konkrétně se jedná o všechny typy testů uvedené v matici testů v tabulce č. 5
- **PROD -** V produkci probíhá testování na úrovni: Canary release a A/B testů a tyto typy testů: kompatibility a funkcionality

Tento tým nemá k dispozici integrační prostředí, proto naplno využívá možnosti testovat v Akceptačním prostředí. Velký důraz se klade také na výsledky testů z produkčního prostředí.

Obrázek 16: Využití prostředí Agilním týmem č. 2



Zdroj: Vlastní zpracování

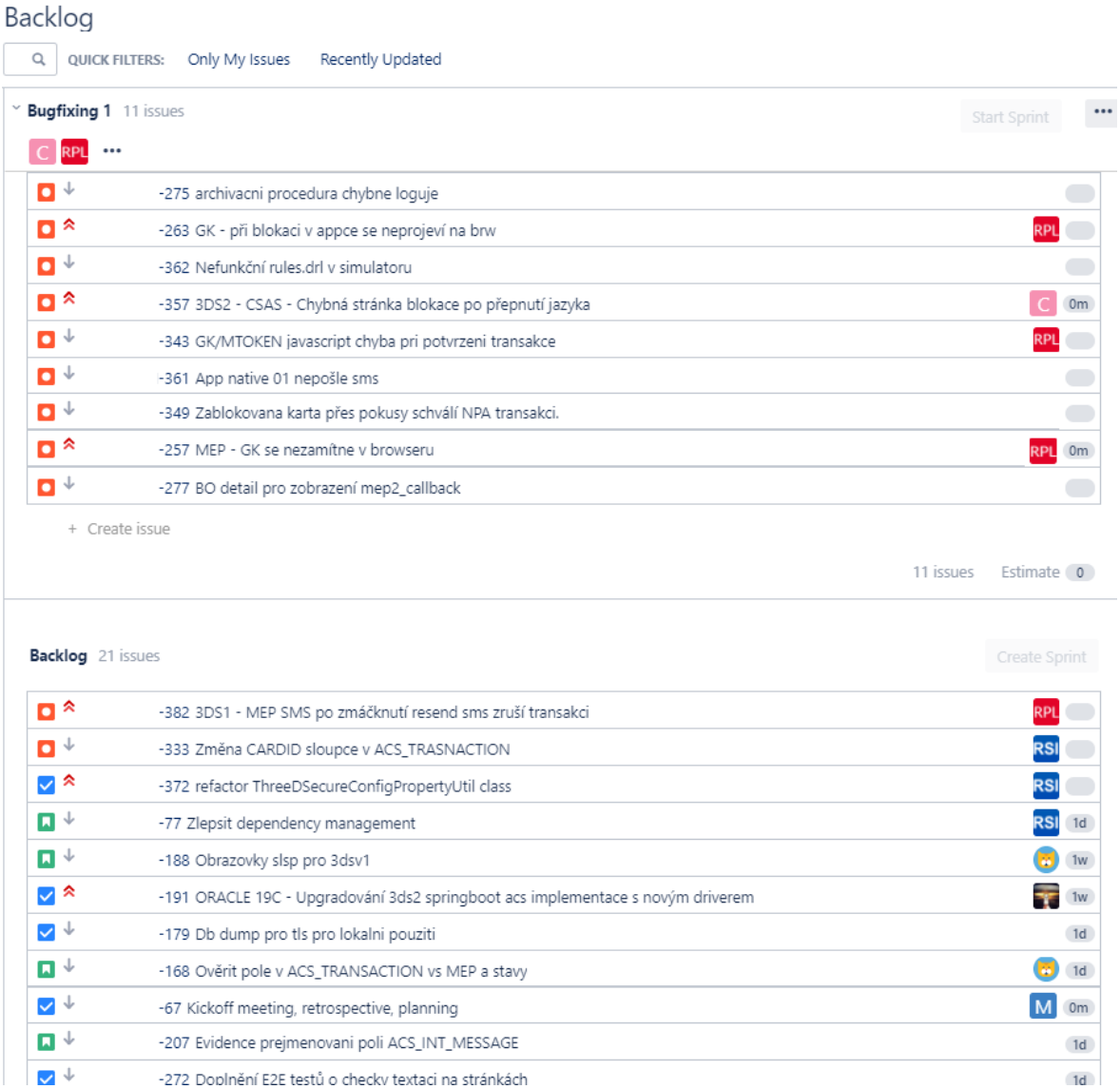
4.2.6 Výstupy týmu

Agilní tým č. 2 využívá pro správu projektu a sledování problémů JIRA software ve verzi 7.13.3. Tým využívá šablony SCRUM. Jira je v týmu využívána pro řízení celého projektu, správu backlogu, aktivních sprintů, testovacích scénářů a hlášení všech defektů, které se netýkají bezpečnosti.

4.2.6.1 Backlog

Jak lze vidět na obrázku č. 17 tento tým má rozdělený backlog na dvě části. První část, označená jako Bugfixing 1 se věnuje pouze chybám, kterých v současné chvíli tým eviduje 11. tyto chyby podle priority tým zařazuje do sprintů. Chyby, které mají prioritu Major (dvě červené šipky) a vyšší budou pravděpodobně zařazeny do nejbližšího sprintu, protože je nutné tyto chyby co nejdříve odstranit. Chyby nižší priority budou do sprintu zařazeny podle uvážení Product Ownera.

Obrázek 17: Backlogu teamu 2



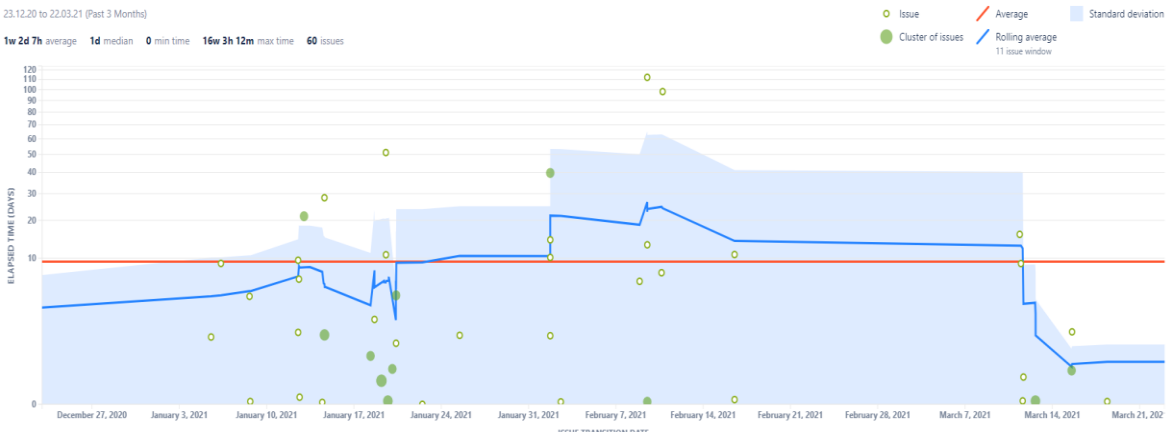
Zdroj: vlastní zpracování z JIRA

Druhá část Backlogu zobrazuje Tasky a Story, které zbývají v rámci projektu dodělat. Tým má v této části Backlogu zařazené i nové, zatím neroztříděné chyby, které se do první části Backlogu musí přiřazovat ručně. V této části Backlogu eviduje tým 21 problémů.

4.2.6.2 Reporty

Tým má k dispozici pro odhad následujících sprintů Kontrolní graf. Z obrázku č. 18 je patrné, že týmu trvá jeden problém vyřídit v průměru 9 dní. Dle toho by se dalo předpokládat, že tým přizpůsobí počet problémů ve sprintu tak, aby na konci sprintu byl co nejnižší počet nedokončených problémů. Nicméně, jak bude vysvětleno později Product Owner se statistikami neřídí a stanovuje počet Story ve sprintu dle vlastního uvážení.

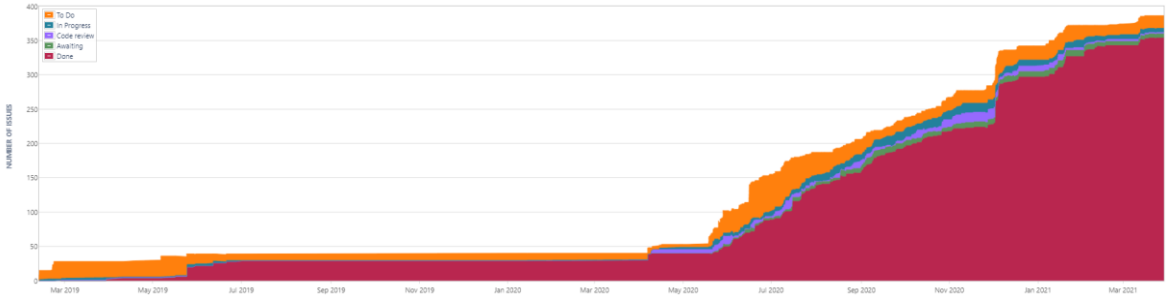
Obrázek 18: Kontrolní graf týmu č.2



Zdroj: Vlastní zpracování

Obrázek č. 19 zobrazuje množství vykonané práce za celou dobu existence projektu. Je vidět, že se týmu daří držet malý počet problémů, které je potřeba řešit (TO DO). Z diagramu se dá vyčíst, že již počet nových problémů nepřibývá tak razantně. Je to dáno tím, že projekt se pomalu blíží do finální fáze a nově se k řešení objevují pouze defekty.

Obrázek 19: Cumulative flow diagram Týmu č.2



Zdroj: Vlastní zpracování

Z tabulky č. 5, kde je zobrazen průběh iterací v době pozorování týmu, lze usoudit, že tým nebral ohled na velký počet nedokončených story ve sprintu. Tým si neustále plánoval nesplnitelný počet úkolů ve sprintu. Tímto nepřesným plánováním sprintů mohl vznikat zbytečný nátlak na vývojáře a ostatní členy týmu a být jedním z faktorů poměrně vysoké chybovosti odhalené Akceptačními testy ve vyvíjeném softwaru. Dle odhadu vývojáři využívali zhruba 20 % kódu z předchozích projektů.

Tabulka 5: Přehled agilního týmu 2

	Iterace 1	Iterace 2	Iterace 3	Iterace 4	Iterace 5	Iterace 6
Dní programátorské práce v MD	48	48	48	48	48	48
Story ve sprintu	56	62	48	50	50	53
Dokončených story ve sprintu	18	12	12	12	15	14
Nedokončených story	38	50	36	38	35	39
Chybovost (%)	30	25	34	28	35	31
přepoužitého kódu (%)	20	20	20	20	20	20

Zdroj: Vlastní zpracování

4.2.7 Analýza testingu Agilního týmu č. 2

Při pozorování testingu týmu č. 2 bylo zjištěno, že banka nemá žádná obecná pravidla pro testování softwaru na projektech a vše závisí na zkušenostech jednotlivých týmů. Banka si tento problém dlouhodobě uvědomuje a v rámci transformace celé banky nesoucí název „Nová Banka“ se chystá tento problém napravit.

Tým se snaží pracovat s automatickými testy, ale tester je nestíhá sám psát. Bylo by dobré, kdyby se vývojáři více zapojili do testování a sami používali alespoň unit testy. Tým by mohl také jednoho z vývojářů na několik MD ve sprintu pověřit psaním automatických testů. Tyto testy pomůžou odhalovat chyby vzniklé úpravou již stávajících funkcionalit za účelem implementace nových. Zavedením integračních testů alespoň na úrovni akceptačního prostředí by tým snížil náklady na opravu defektů. Integračními testy by sám tester mohl analyzovat kde vznikla chyba. Nemusel by tak zapojovat větší počet lidí do řešení této chyby, ale pouze vývojáře dané komponenty, na které chyba vznikla.

4.3 Agilní tým 3

Posledním pozorovaným objektem byl Agilní tým 3. Tým se zabývá tvorbou agregační platformy umožňující přístup k účtům různých bank prostřednictvím jedné platformy. Tým uvádí, že pracuje pomocí SCRUM metodiky.

4.3.1 Cíle projektu

Cílem projektu agilního týmu č. 3 je zpřístupnit služby, které jsou definované evropskou směrnicí PSD2 a propojení bank pomocí jediného API. Tato služba je prostřednictvím obchodních zástupců firmy nabízena bankám, úvěrovým společnostem, finančním poradcům, platebním agregátorům a firemním zákazníkům.

Tým pracuje na možnosti zprostředkovat bankovní identitu a určování ekonomického profilu zákazníka na základě transakční historie. Cílem tedy je vytvořit univerzální produkt který bude společnost prodávat ostatním subjektům.

4.3.2 Členové agilního týmu

Zajímavostí je, že si tým na začátku zvolil, že bude používat agilní metodiku, ale toto se neprojevilo ve složení týmu. Tým zachovává názvy pozic známé z tradičních metod vývoje SW. Jedním z důvodů je, že tým je téměř jediný, kdo v rámci firmy agilní metodiku využívá. Členové týmu si chtěli tento způsob vývoje sami vyzkoušet.

V současné době tvoří tým čtyři vývojáři, kteří mají na starost integrování již vyvinutého řešení subjektům, kteří projevili zájem o tento produkt. Jeden z vývojářů funguje jako šéf vývoje a dohlíží na pokrok, dodržování plánu a vytváření změnových

požadavků. Tento vývojář je v agilním týmu od samého počátku. Má tak největší přehled o problematice vývoje produktu.

Analýzu systémů, do kterých se bude produkt implementovat mají na starosti dva Systémoví Architekti. Jejich práce spočívá v seznámení se s procesy a potřeby firmy pro kterou tým produkt zavádí. Ti navrhují optimalizaci procesů a způsob jakým bude produkt zaváděn.

Jelikož se stává, že v jedné chvíli je potřeba produkt zavádět do více firem má tým k dispozici tři Projektové Manažery. Jejich úkolem je komunikace s odpovědným vedením firem, kterým se produkt dodává. Musejí se zároveň mezi sebou domlouvat na tvorbě plánu, tvoření Backlogu a přidáváním problémů do Sprintu.

Tým má k dispozici jediného testera, který má za úkol úpravu testovacích scénářů pro potřeby dané problematiky. Jeho nejdůležitějším úkolem je získání a příprava testovacích dat. Ty by měl získat od společností pro které je produkt implementován.

Zřejmě nejdůležitější součástí týmu je Produktový manažer. Produktový manažer má na starosti nabízení a prodej výrobku. Zároveň sbírá poznatky od uživatelů a sleduje trendy, které by měli vést k vylepšení nabízeného produktu.

4.3.3 Nástroje a techniky agilního týmu č. 3

V předchozí kapitole bylo řečeno, že tento tým je ve firmě jediný, který využívá agilní metodiku. Toto se podepisuje i na způsobu využití agility. Tým nemá k dispozici agilního kouče, který by ho navedl správným směrem. Členové týmu využívají znalosti, které se naučili na školení, ale nedodržují Scrum metodiku důsledně.

Projekt využívá ze Scrumových ceremonií pouze denní scrum. Těmito schůzky zahajuje tým každý pracovní den. Projektový manažeři se střídají v tom, kdo schůzku vede. Na schůzce se řeší pracovní náplň jednotlivých členů týmu pro tento den. Občas se na schůzce řeší rozsáhlejší problém, pak může schůzka protáhnout i na hodinu.

Mimo tuto ceremonii se konají nepravidelné schůzky se zákazníky, kterých se účastní především projektový manažeři a systémový architekti. Na těchto schůzkách se domlouvá podoba služby, kterou bude tým dodávat a zkoumá se dopad služby na ostatní systémy.

Tým v počátku vývoje produktu pracoval ve čtrnácti denních sprintech. V době pozorování týmu pracoval pouze v týdenních sprintech. Od tohoto kroku si sliboval rychlejší přizpůsobování se požadavkům zákazníků.

4.3.4 Typy a úrovně testů

Z důvodu povahy nabízené služby a omezeného přístupu k datům zákazníků má tým v testování omezené možnosti. Tým pracuje pouze na těchto úrovních testů:

- Vývojové testy – testy prováděné ve vývojovém prostředí
- SIT – Systémově integrační testy
- UAT – Uživatelské akceptační testy

Využití úrovní testování tým rozlišuje dle typu Softwarových změn

4.3.4.1 Matice testů

Podobně jako u předchozích týmů byla sestavena matice testů (tabulka č. 6), která mapuje využití typů a úrovní testů tímto agilním týmem.

Tabulka 6: Přehled využívaných typů a úrovní testů agilním týmem č. 3

	Vývojové testy	SIT	UAT
Funkční testy		X	X
Regresní testy		X	X
Smoke testy	X	X	X

Zdroj: vlastní zpracování

4.3.4.2 Detailní popis prováděných typů testů

Funkční testy – jsou prováděny na úrovni SIT testerem agilního týmu a na rovni UAT jsou uskutečňovány business uživatelem s podporou tohoto agilního týmu.

Regresní testy – provádí tester agilního týmu a business uživatel pouze na akceptačním prostředí

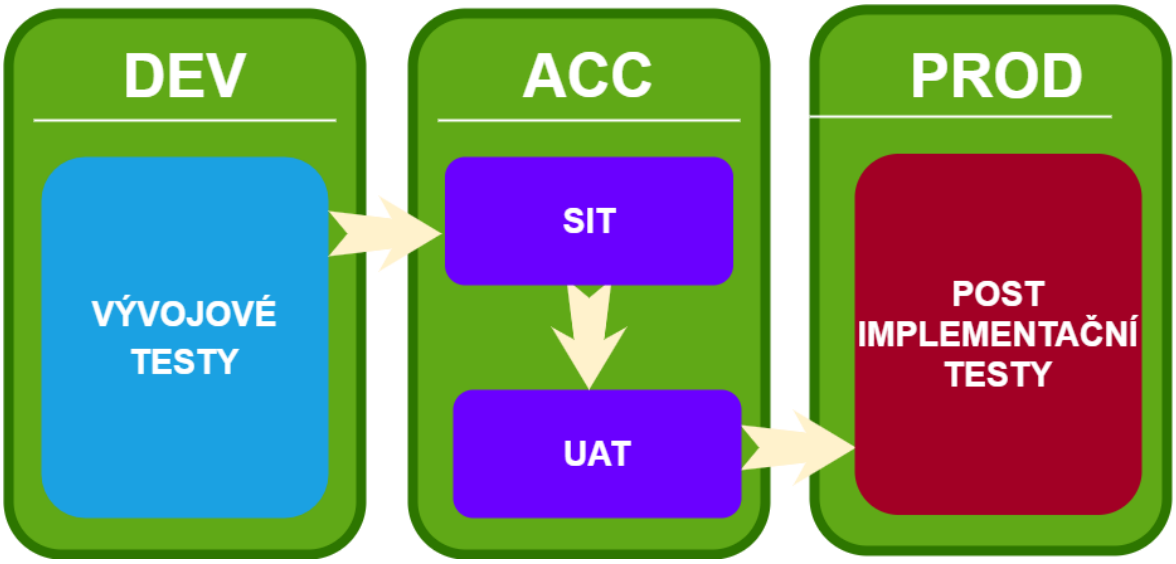
Smoke testy – Smoke testy ve vývojovém prostředí provádí vývojáři v Akceptačním prostředí ho po každém nasazování provádí tester agilního týmu a před každým prováděním Business testů ho provádí Business uživatelé.

4.3.5 Prostředí

- **DEV –** vývojáři v DEV prostředí provádí Smoke testy na úrovni vývojových testů

- **ACC –** V akceptačním prostředí jsou prováděny na úrovni SIT a UAT funkční, regresní a smoke testy
- **PROD** – v produkčním prostředí probíhají po nasazení nové funkcionality smoke testy

Obrázek 20: Využití prostředí Agilním týmem č. 3



Zdroj: Vlastní zpracování

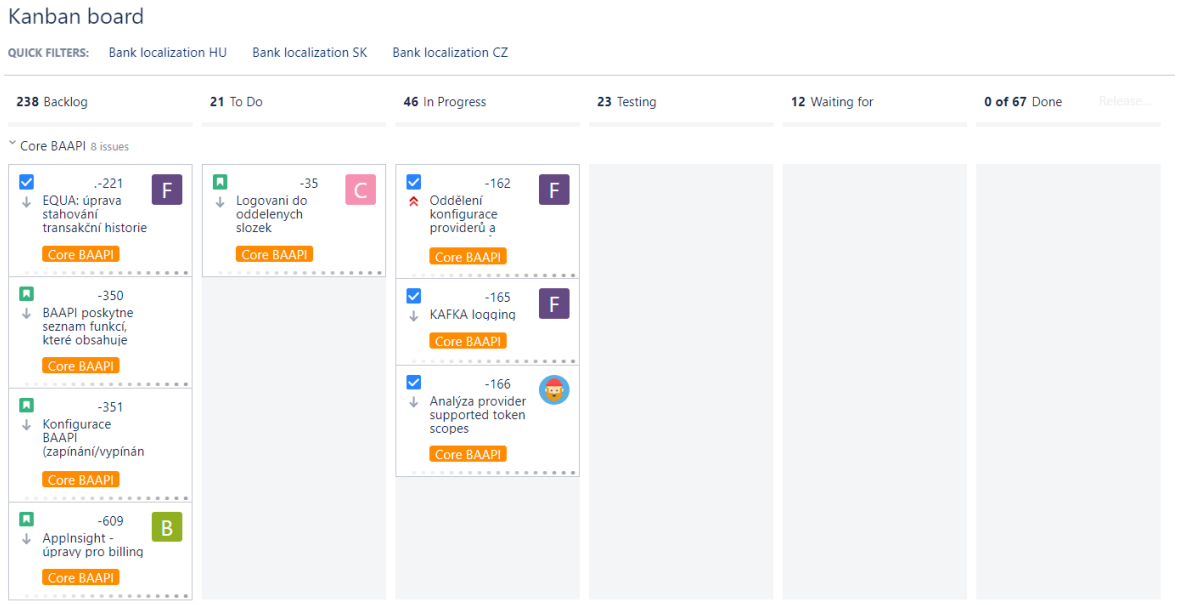
4.3.6 Výstupy týmu

Tento tým pracuje se softwarem JIRA, využívá ho ale pouze ke správě Testovacích scénářů, defektů a evidenci Backlogu. Aktivní sprinty eviduje v excelu. Tým si je vědom možnosti evidence Sprintů v excelu. Snaží se postupně s tímto nástrojem začít pracovat.

4.3.6.1 Backlog

Tým má v JIRA, kvůli vybrání špatné šablony při tvoření projektu, k dispozici Kanban board, jak můžeme vidět na obrázku č. 21 v backlogu je evidováno 238 problémů. Tyto problémy jsou rozděleny do 29 podoblastí dle typu problému (např. Technologický dluh, online úvěry, support, nové banky atd.). tento Kanban board vznikl zhruba před rokem ve snaze začít využívat JIRA nástroje pro větší efektivitu týmu, ale po pár měsíců se nástroj přestal používat.

Obrázek 21: Kanban board Agilního týmu č. 3

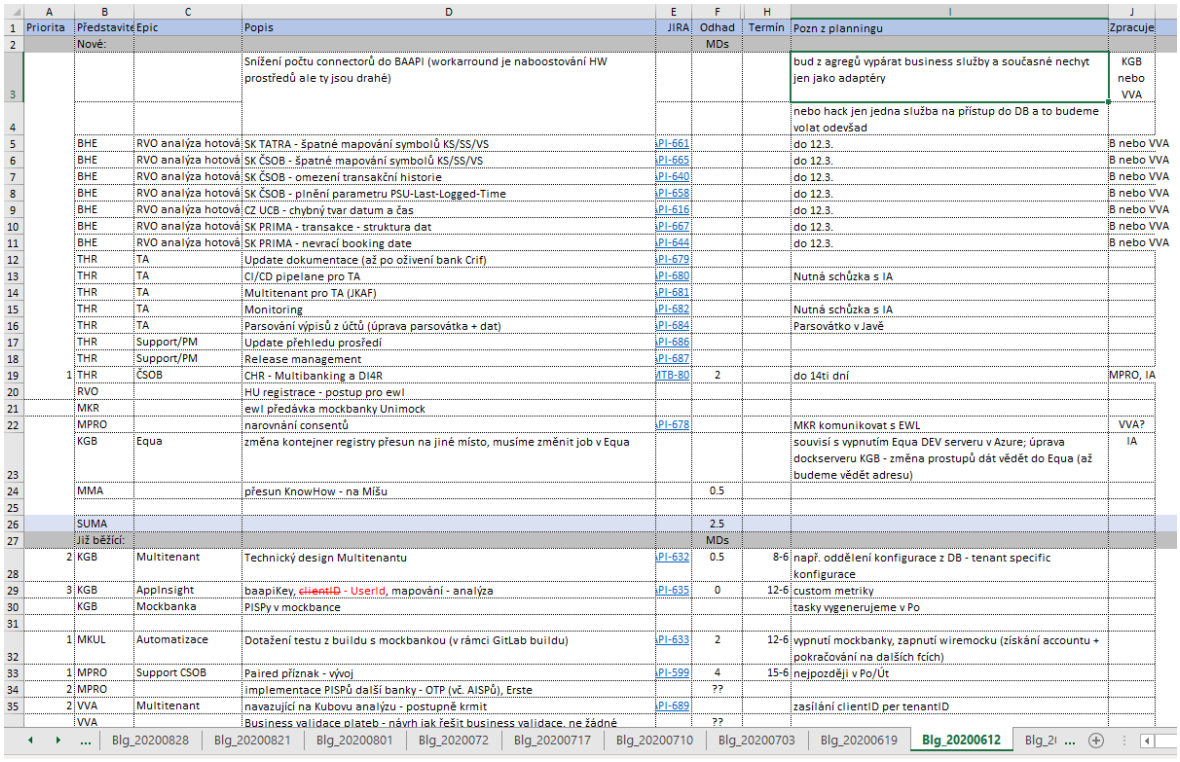


Zdroj: vlastní zpracování

4.3.6.2 Sprinty

Zajímavostí agilního týmu č. 3 je, že své sprinty neřídí pomocí nástroje JIRA, ale pomocí tabulkového procesoru Microsoft Excel. Ukázka toho, jak Sprint v Excelu vypadá je na obrázku č. 22. Projektový manažer vyplňoval každý týden na konci sprintu úkoly, které je potřeba v průběhu následujícího sprintu zpracovat. Problémy jsou rozdělovány na již běžící z předchozích sprintů a na nově přidané do sprintu. U většiny problémů je odkaz, pod kterým je úkol evidován v JIRA. Sprinty jsou evidované v jediném excelu a každý sprint představuje jednu záložku. Odhadu časové náročnosti se nevěnovalo příliš pozornosti.

Obrázek 22: Ukázka plánování sprintu týmem č. 3



	A	B	C	D	E	F	H	I	J
1	Priorita	Představitel	Epic	Popis	JIRA	Odhad	Termín	Pozn z planningu	Zpracuje
2		Nové:				MDs			
3				Snížení počtu connectorů do BAAPI (workarround je naboostování HW prostředů ale ty jsou drahé)				bud z agregů vypárat business služby a současně nechyt jen jako adaptéry	KGB nebo VVA
4								nebo hack jen jedna služba na přístup do DB a to budeme volat odevšad	
5		BHE	RVO analýza hotová	SK TATRA - špatné mapování symbolů KS/SS/VS	JPI-661			do 12.3.	B nebo VVA
6		BHE	RVO analýza hotová	SK ČSOB - špatné mapování symbolů KS/SS/VS	JPI-665			do 12.3.	B nebo VVA
7		BHE	RVO analýza hotová	SK ČSOB - omezení transakční historie	JPI-640			do 12.3.	B nebo VVA
8		BHE	RVO analýza hotová	SK ČSOB - plnění parametru PSU-Last-Logged-Time	JPI-658			do 12.3.	B nebo VVA
9		BHE	RVO analýza hotová	CZ UCB - chybný tvar datum a čas	JPI-616			do 12.3.	B nebo VVA
10		BHE	RVO analýza hotová	SK PRIMA - transakce - struktura dat	JPI-667			do 12.3.	B nebo VVA
11		BHE	RVO analýza hotová	SK PRIMA - nevrací booking date	JPI-644			do 12.3.	B nebo VVA
12		THR	TA	Update dokumentace (až po oživení bank Crif)	JPI-679				
13		THR	TA	CI/CD pipelane pro TA	JPI-680			Nutná schůzka s IA	
14		THR	TA	Multitenant pro TA (JKAF)	JPI-681				
15		THR	TA	Monitoring	JPI-682			Nutná schůzka s IA	
16		THR	TA	Parsování výpisů z účtů (úprava parsovátka + dat)	JPI-684			Parsovátko v Javě	
17		THR	Support/PM	Update přehledu prostředí	JPI-686				
18		THR	Support/PM	Release management	JPI-687				
19	1	THR	ČSOB	CHR - Multibanking a DI4R	ITB-80	2		do 14ti dní	MPRO, IA
20		RVO		HU registrace - postup pro ewl					
21		MKR		ewl předávka mockbanky Unimock					
22		MPRO		narovnání consentů	JPI-678			MKR komunikovat s EWL	VVA?
23		KGB	Equa	změna kontejner registry přesun na jiné místo, musíme změnit job v Equa				souvisí s vypnutím Equa DEV serveru v Azure; úprava dockserveru KGB - změna prostupů dát vědět do Equa (až budeme vědět adresu)	IA
24		MMA		přesun KnowHow - na Míšu		0.5			
25									
26		SUMA				2.5			
27		Již běžící:				MDs			
28	2	KGB	Multitenant	Technický design Multitenantu	JPI-632	0.5	8-6	např. oddělení konfigurace z DB - tenant specific konfigurace	
29	3	KGB	AppInsight	baapiKey, ~~clientID~~ - UserId, mapování - analýza	JPI-635	0	12-6	custom metriky	
30		KGB	Mockbanka	PISPy v mockbance				tasky vygenerujeme v Po	
31									
32	1	MKUL	Automatizace	Dotažení testu z buildu s mockbankou (v rámci GitLab buildu)	JPI-633	2	12-6	vypnutí mockbanky, zapnutí wiremocku (získání accountu + pokračování na dalších fcích)	
33	1	MPRO	Support CSOB	Paired příznak - vývoj	JPI-599	4	15-6	nejpozději v Po/Út	
34	2	MPRO		implementace PISPů další banky - OTP (vč. AISPů), Erste		??			
35	2	VVA	Multitenant	navazující na Kubovu analýzu - postupně krmit	JPI-689			zasílání clientID per tenantID	
		VVA		Business validace plateb - návrh jak řešit business validace, ne žádné		??			

Blg_20200828 | Blg_20200821 | Blg_20200801 | Blg_2020072 | Blg_20200717 | Blg_20200710 | Blg_20200703 | Blg_20200619 | **Blg_20200612** | Blg_2...

Zdroj: Vlastní zpracování

4.3.6.3 Reporty

Tým řádně nevyužíval pro svoji práci software podporující agilní metody a řízení projektu. Vytváření reportů by tak dalo projektovým manažerům příliš práce. Přehledy proto tvořili pouze pokud si to od nich vyžádalo vedení. Projektový manažeři nevěnují pozornost tomu, že by jim vytváření reportů pomohlo se zlepšením plánování práce a ušetřilo čas. Nevylučují se ani chyby, které mohli členové tým utvořit při přepisování problémů mezi jednotlivými sprinty, nebo při případném zanesení výsledku zpátky do JIRA.

V tabulce č. 7 byl na základě poznatků a informací získaných v průběhu pozorování vytvořen přehled Agilního týmu č. 3. Tým byl pozorován tři měsíce, v průběhu kterých proběhlo dvanáct iterací. Náhodný počet story ve sprintu a počet nedokončených story ve sprintu odpovídá tomu, že tým nevěnoval pozornost odhadu náročnosti jednotlivých úkolů a sledování reportů. Příčinou většího počtu chyb je dle pozorování a vyjádření týmu problém s akceptačním prostředím, které neodpovídá tomu produkčnímu.

V současné době pracují na implementaci již vyvinutého produktu do různých bankovních prostředí. Přepoužívá tak téměř 100 % původního kódu, který upravuje dle potřeb klienta.

Tabulka 7: Přehled agilního týmu č. 3

	Iterace 1	Iterace 2	Iterace 3	Iterace 4	Iterace 5	Iterace 6	Iterace 7	Iterace 8	Iterace 9	Iterace 10	Iterace 11	Iterace 12
Dní programátorské práce v MDs	20	20	20	20	20	20	20	20	20	20	20	20
Story ve sprintu	29	30	31	30	31	22	21	31	38	30	25	34
Dokončených story ve sprintu	20	20	19	19	23	22	14	11	25	16	15	20
Nedokončených story	9	10	12	11	8	0	7	20	13	14	10	14
Chybovost (%)	50	45	51	25	41	12	10	40	44	30	51	30
Přepoužitého kódu (%)	100	100	100	100	100	100	100	100	100	100	100	100

Zdroj: Vlastní zpracování

4.3.7 Analýza testingu Agilního týmu č. 3

Při pozorování testingu bylo zjištěno, že Agilní tým má velký problém s počtem chyb odhalených v produkčním prostředí. Tento problém je způsoben dvěma faktory. Část chyb vzniká nekonzistencí prostředí na straně klienta a vývoj tomuto nemůže v podstatě nijak zabránit. Není-li poskytnuta dostatečná součinnost zákazníka budou tyto chyby neustále vznikat. Agilní tým má toto ošetřené SLA dohodami, takže mu nehrozí postih za nekvalitní dodávku způsobenou nekonzistencí. Dalším problémem jsou chyby, které vznikají při úpravě kódu. Stává se, že vývojář při úpravě nevědomky smaže část kódu nebo pozmění chování nějaké části implementace. Tento problém by mohli odhalit detailnější vývojářské testy, které by navíc šetřili náklady na opravu. Vývojáři by se měli zapojit do testování využíváním unit testů. Vzhledem k povaze projektu by měl tým zvážit využití automatických testů. Tester využívá pro správu testovacích scénářů, exekuci testů a evidování defektů JIRA software díky kterému má nejen on ale i celý tým přehled o problémech týkajících se testování.

4.4 Návrhy a doporučení

Analýzou chování pozorovaných týmů bylo zjištěno časté odchylování se od podstaty agilního vývoje softwaru. V průběhu pozorování bylo zjištěno, že v týmech stále převládá rivalita mezi testováním a vývojem. V omezeném počtu členů týmu, kdy testování zastává mnohdy pouze jeden člověk, je potřeba se zaměřit na větší spolupráci členů týmu. Vedení firem by mělo zajistit správné nástroje pro podporu agilních metodik, které ušetří čas a pomohou zefektivnit práci. Tyto nástroje je vhodné používat pro odhad náročnosti jednotlivých problémů a zlepšení plánování sprintů. Správný odhad může pomoci s větší spokojeností členů týmu. Pokud je sprint naplánovaný tak, že v jeho průběhu nebude mít některý z členů již přiřazený žádný problém snižuje se efektivita týmu. V případě, že budou mít členové týmu na sobě ve sprintu přiřazeno spoustu problémů může to vést ke stresu, který snižuje efektivitu a může zapříčinit větší chybovost jedince.

Z informací získaných v průběhu pozorování práce týmu a zkušeností autora této diplomové práce, kterou získal v průběhu pěti let strávených v prostředí testování aplikací, vznikl obecný model úrovní a typů testů (tabulka č. 8), které by mohly využívat agilní týmy

pro představu a vytvoření vlastní sady testů potřebných k úspěšnému otestování softwaru dle typu softwarových změn.

Tabulka 8: Návrh matice úrovní a typů testů při vývoji nového SW

	UT	KIT	ST	UAT
Funkční testy	x	x	x	x
Security testy		x	x	x
Performance testy			x	x
Disaster Recovery			x	x
Regresní testy	x	x	x	x
Free testy				x

Zdroj: Vlastní zpracování

Unit testy (UT)

Vzhledem k tomu, že v agilním týmu je často pouze jeden tester měl by se do procesu testování větší měrou zapojovat i vývojový tým. Ten by měl již na úrovni Unit testů (UT) implementovat Funkční a Regresní testy na základní vyvíjenou funkcionalitu formou tzv. happy day scénářů, tedy testů takových dat, u kterých je očekáváno, že budou fungovat. Plánování a řízení testů by měl zajistit nejzkušenější vývojář týmu. Náklady na vývoj unit testů by měli být zahrnuty do odhadů na vývoj komponent. Implementaci a vykonání testů provádí vývojáři jednotlivých komponent. Předpokládá se použití emulátorů umožňujících simulaci jiných FE a BE komponent, aby daná komponenta byla testovatelná izolovaně. Regresní unit testy by měl vývojář testovat před předáním každé nově vyvinuté komponenty testerovi. Testy jsou primárně vykonávány na vývojovém prostředí (DEV). Za předpokladu, že bude defekt odhalený unit testy vývojářem ihned opraven nemusí být evidován.

Integrační testy komponent (KIT)

Příprava testovacích případů a dat na úrovni KIT by měla být na zodpovědnosti testera v součinnosti s vývojáři. Vývoj by se měl na test analýze a přípravě testovacích scénářů přímo podílet, nebo by měl být seznámen se sadou happy day scénářů před zahájením testování. Při návrhu testovacích scénářů musí být mimo jiné zohledněna také dostupnost jednotlivých

komponent pro integraci a případná nutnost použití emulátorů pro chybějící frontend nebo backend. Funkční testy na této úrovni provádí tester. Ten by měl provedené testy evidovat. V případě, že jsou v rámci integračního testování nalezené defekty, které nejsou ve spolupráci s vývojářem hned opraveny měli by být evidovány v agilním boardu. Při zvolení vhodných nástrojů pro testování integračních testů (například platformy ReadyAPI od společnosti Smartbear) lze jednoduše vytvořit sada automatických testů, které lze využívat v regresním testování, které by mělo být na úrovni KIT také prováděno. Vyžaduje-li povaha projektu prováděni Security testů měli by být testy prováděny již na této úrovni. Tyto testy by měli být prováděny až po exekuci funkčních testů a odstranění evidovaných chyb na komponentě.

Systémové testy (ST)

Úroveň systémových testů je zaměřena na testování systému jako celku. Testují se business funkcionality a celkové E2E procesy. Přípravu testovacích případů a testovacích dat pro ST má v kompetenci tester. Testovací případy by měli být navrhovány tak, aby pokrývaly business požadavky a byly použitelné pro oddělení BUS v rámci akceptačních testů. Z toho důvodu je vhodná součinnost někoho z businessu při navrhování TC. Testy by měli být vykonávány na integračním (INT) a akceptačním (ACC) prostředí. Na ACC prostředí by měla být provedena minimálně sada testů určená následně pro UAT, aby byla ověřena připravenost na akceptační testování. Po nasazení na ACC prostředí je dobré provést sadu sanity testů pro ověření připravenosti pro ST. Na začátku každého testovacího cyklu je vhodné vykonat sadu smoke testů ověřujících základní funkčnost a připravenost na další testování. V rámci jednotlivých testovacích cyklů se provádí také regresní testy, které ověřují, že nebyla ovlivněna již dříve otestovaná funkčnost. Regresní testy je dobré i za pomoci Vývojářů automatizovat. Všechny testy a jejich výsledky by měli být evidované. Nalezené defekty je potřeba přiřadit do agilního boardu, kde jim Product owener určí prioritu, dle které je přiřadí do následujících sprintů k opravě.

V rámci Systémových testů by se mělo po otestování funkčních testů a opravě všech defektů přistoupit k testování nefunkčních požadavků. V akceptačním prostředí by měli být otestován pomocí performance testů výkon a stabilita systému. Zejména u velkých systémů by měl být naplánován test chování systému během výpadku a schopnosti obnovit funkční stav systému.

Akceptační testy

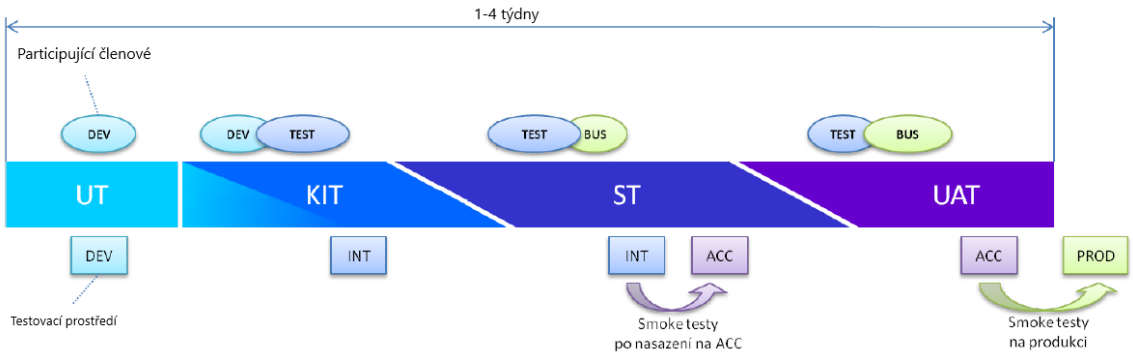
Během akceptačních testů se ověřuje, že systém splňuje všechny funkční i nefunkční požadavky a je připraven pro nasazení do produkce. Hledání defektů není primárním účelem této úrovně testování. Označení UAT neznamená omezení pouze na oblast User Acceptance testů, ale zahrnuje všechny potencionální typy akceptačních testů:

- Uživatelské akceptační testy
- Provozní akceptační testy
- Smluvní a regulační akceptační testy
- Alfa a beta testy

Odpovědnost za provádění těchto testů by měl mít business tým. Pro testování se používají vybrané testovací scénáře z úrovně systémových testů. Testování probíhá v Akceptačním prostředí.

Na obrázku č. 23 je zobrazen příklad možného rozložení testovacích úrovní během realizace projektu. Pro zvýšení efektivity testování se předpokládá překryv jednotlivých úrovní v čase. Délka jednotlivých úrovní není pevně dána a závisí na konkrétních potřebách projektu. Cílem je, aby doba od začátku vývoje po nasazení do produkce nepřesáhla 4 týdny. Jedná se o maximální stanovenou dobu trvání sprintu v Agilní metodice.

Obrázek 23: Harmonogram exekuční fáze testovacích úrovní



Zdroj: vlastní zpracování

5 Závěr

Diplomová práce se zaměřuje na problematiku testování softwaru v agilním prostředí. Cílem diplomové práce je nalézt a zhodnotit vhodný způsob testování softwaru v agilním prostředí.

V teoretické části diplomové práce jsou definovány základní pojmy, které se týkají testování softwaru a techniky testování softwaru. V kapitole testování softwaru je popsána relevantnost testování ve spojitosti s efektivitou nákladů, bezpečnosti a kvality produktu. Dále jsou v této diplomové práci popsány agilní metodiky a agilní manifest. V následující kapitole jsou vysvětleny agilní metody vývoje softwaru a jejich srovnání s tradičními metodami. Poslední kapitolou jsou programy pro podporu agilního vývoje, kde jsou charakterizovány softwary, které využívají vybrané agilní týmy.

V analytické částí byly na základě rozboru fungování agilních týmů a jejich způsobu testování zjištěny nedostatky týkající se především špatného nastavení úrovně testování. Byla shledána nedostatečná spolupráce vývojářů a testerů v agilních týmech a malé, nebo žádné zapojení vývojářů do procesu testování. Toto vedlo k problémům s odhalením chyb, které se v důsledku toho dostávaly do produkční verze softwaru. V této diplomové práci byl navržen obecný model úrovní a typů testů, které mohou využít agilní týmy pro představu a vytvoření vlastní sady testů potřebných k úspěšnému otestování softwaru dle typu softwarových změn. Byl také navržen harmonogram exekuční fáze testovacích úrovní, do kterého jsou zapojeny vývojáři, testeři i business uživatelé. Zlepšením komunikace a spolupráce těchto skupin může vést ke zkvalitnění vyvíjeného produktu. V rámci diplomové práce bylo zjištěno, že v agilním prostředí, ve kterém funguje správná spolupráce, není nutné vytvářet obsáhlou dokumentaci.

Výstupy diplomové práce jsou schopné rozhodnout o tom, jaký způsob testování aplikovat na vlastní projekt v agilním prostředí.

6 Seznam použitých zdrojů

ATLASSIAN. *Using Active sprints* [online]. 2020 a Dostupné z: https://confluence.atlassian.com/jirasoftwareserver/using-active-sprints-938845497.html

ATLASSIAN. *View and understand the control chart* [online]. 2020 b [cit. 2020-12-02]. Dostupné z: https://support.atlassian.com/jira-software-cloud/docs/view-and-understand-the-control-chart/

BOEHM, Barry a P.N. PAPACCIO. *Understanding and Controlling Software Costs*. IEEE Transactions on Software Engineering. Redondo Beach: TRW, 1988. ISBN 0098-5589.

BOEHM, Barry. *Software engineering: Barry W. Boehm's lifetime contributions to software.* Wiley-IEEE Computer Society Press. Redondo Beach: John Wiley & Sons, 2007. ISBN 978-0-470-14873-0.

BRUCKNER, Tomáš. *Tvorba informačních systémů: principy, metodiky, architektury*. Praha: Grada, 2012. Management v informační společnosti. ISBN 978-80-247-4153-6.

CRISPIN, Lisa a Janet GREGORY. *Agile testing: a practical guide for testers and agile teams*. Upper Saddle River: Addison-Wesley, 2009. Addison-Wesley signature series. ISBN 978-032-1534-460.

DEAN, Raymond a Drew JEMILO. *The Scaled Agile story* [online]. 2020 Dostupné z: https://www.scaledagile.com/about/about-us/

DÍAZ, Leo. *Creating reports in Jira: 6 Different ways to generate them* [online]. 2020 Dostupné z: https://blog.deiser.com/en/creating-reports-in-jira-6-ways-to-generate-them

GEOFFREY, Elliot. *Global Business Information Technology: an integrated systems approach.* New York: Addison-Wesley, 2004. Addison-Wesley signature series. ISBN 0-321-27012-6.

GRAHAM, Dorothy, Rex BLACK a Erik VAN VEENENDAAL. *Foundations of software testing: ISTQB certification.* Fourth edition. Andover, Hampshire: Cengage, [2020]. Addison-Wesley signature series. ISBN 978-1-4737-6479-8.

GROSSMANN, Lukáš. *Lekce 5 - Extrémní programování* [online]. 2020. Dostupné z: https://www.itnetwork.cz/navrh/metodiky/extremni-programovani

HAAG, Stephen a Maeve CUMMINGS. *Management Information Systems for the Information Age*. Irwin/McGraw-Hill, 2012. ISBN 978-0-07-337685-1.

HAMBLING, Brian, Peter MORGAN, Angelina SAMAROO, Geoff THOMPSON a Peter WILLIAMS. *Software Testing: An ISTQB-BCS Certified Tester Foundation guide - 4th edition* [online]. 2019 [cit. 2021-01-12]. ISBN 978-1-78017-494-5.

HLAVA, Tomáš. *Spirálový model* [online]. 2011 Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/modely-zivotniho-cyklu-softwaru/spiralovy-model/

KEN, Schwaber, Sutherland JEFF a Iveta GRÜTTNER. *Průvodce Scrum* [online]. 2020, s. 1-15. Dostupné z: https://scrumguides.org/docs/scrumguide/v2020/2020-Scrum-Guide-Czech.pdf

KENDA, Pittari. *WHAT IS SCRUM? – PART 2* [online]. 2012 Dostupné z: https://wafinettest.wordpress.com/2012/02/25/what-is-scrum-part-2/

KITNER, Radek. *Typy testování software (třídění testů)* [online]. 2020 [cit. 2021-02-05]. Dostupné z: https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/

LOUIS, Raymond. Custom Kanban: *Designing the System to Meet the Needs of Your Environment*. University Park: Productivity Press, 2006. ISBN 978-1-56327-345-2.

MICROSOFT AZURE. *Azure DevOps* [online]. 2020 [cit. 2020-12-02]. Dostupné z: https://azure.microsoft.com/cs-cz/services/devops/

O'REILLY. *The sprint report* [online]. 2020 Dostupné z: https://www.oreilly.com/library/view/jira-software-essentials/9781788833516/78724b42-e94c-4217-a977-7c3766f80644.xhtml

PATTON, Ron. *Testování softwaru*. Praha: Computer Press, 2002. Programování. ISBN 80-7226-636-5.

ROSENBERG, Doug a Matt STEPHENS. *Extreme Programming Refactored: The Case Against XP*. Apress, 2003. ISBN 978-1-59059-096-6.

ROUDENSKÝ, Petr. *Kvalita softwaru: teorie a praxe*. Prostějov: Computer Media, 2016. ISBN 978-80-7402-294-4.

SCALED AGILE INC.. *PI Planning* [online]. 2021a [cit. 2021-03-30]. Dostupné z: https://www.scaledagileframework.com/pi-planning/

SCALEDAGILE INC. *SAFe Lean-Agile Principles* [online]. 2021b [cit. 2020-12-02]. Dostupné z: https://www.scaledagileframework.com/safe-lean-agile-principles/

SCHUURMAN, Robbin. *Scrum Master vs Project manager: An overview of the tifferences* [online]. 2020 [cit. 2021-01-30]. Dostupné z: https://medium.com/the-value-maximizers/scrum-master-vs-project-manager-an-overview-of-the-differences-73104d0264ab

SCHWABER, Ken. *SCRUM Development Process: Advanced Development Methods*. 131 Middlesex Turnpike Burlington: Cengage, 2005. Dostupné také z: http://jeffsutherland.com/oopsla/schwapub.pdf

SCRUM.ORG. *What is a Sprint Backlog?* [online]. 2020 Dostupné z: https://www.scrum.org/resources/what-is-a-sprint-backlog

SCRUM.ORG. *What is a Sprint Retrospective?: Learn About the Sprint Retrospective Event* [online]. 2020 [cit. 2021-03-19]. Dostupné z: https://www.scrum.org/resources/what-is-a-sprint-retrospective

ŠIMŮNEK, David. *Co je Velocita v Agilním vývoji* [online]. 2019 Dostupné z: https://www.davidsimunek.com/post/co-je-velocita-v-agilnim-vyvoji

SIRKOVSKÝ, Marek. *Testováním řízený vývoj* [online]. 2006 [cit. 2021-01-30]. Dostupné z: http://programujte.com/clanek/2006051006-testovanim-rizeny-vyvoj/

SOCHOVÁ, Zuzana. *Burndown graf* [online]. 2008 Dostupné z: https://soch.cz/blog/management/agile/burndown-graf/

SUTHERLAND, Jeff, Kent BECK, Arie VAN BENNEKUM, et al. *Manifest Agilního vývoje software* [online]. 2001a [cit. 2021-03-19]. Dostupné z: https://agilemanifesto.org/iso/cs/manifesto.html

SUTHERLAND, Jeff, Kent BECK, Arie VAN BENNEKUM, et al. *Principy stojící za Agilním Manifestem* [online]. 2001b [cit. 2021-03-19]. Dostupné z: https://agilemanifesto.org/iso/cs/principles.html

SUTHERLAND, Jeff. *The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework*. One Broadway, 14th Floor: Cengage, 2020. Dostupné také z: https://web.archive.org/web/20150814201800/http://jeffsutherland.com/ScrumPapers.pdf

VÍTEK, Václav. Kanban: *Tahový systém řízení výroby* [online]. 2012 Dostupné z: https://www.svetproduktivity.cz/slovnik/Kanban.htm

ZOHO. *See how you can keep sprinting smoothly with Zoho Sprints!* [online]. [cit. 2021-03-31]. Dostupné z: https://www.zoho.com/sprints/