Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií

Bakalářská práce

Agilní metodiky vývoje software ve vybrané společnosti

Radim Jirmus

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Radim Jirmus

Hospodářská politika a správa
Podnikání a administrativa

Název práce

Agilní metodiky vývoje software ve vybrané společnosti

Název anglicky

Agile methods of software development in selected company

Cíle práce

Hlavním cílem bude vymezit problematiku vývoje informačních systémů s hlavním zaměřením na agilní vývoj informačních systémů. Dílčím cílem bude navrhnout způsob testování nové funkcionality, která má být přidána do již existující a využívané aplikace vybrané banky působící na trhu v České republice. Tento test bude navržen za podmínek agilního vývoje vybrané společnosti a to podle již existující metodiky vývoje dané společnosti.

Metodika

Teoretická část bakalářské práce se bude zakládat na analýze a rešerši odborných informačních zdrojů. Následný postup spočívá ve zpracování interních podmínek agilního vývoje vybrané společnosti dle společností vydávaných dokumentů a metodik. Na základě takto zpracované literární rešerše pak bude navrženo testování v agilním prostředí konkrétního vývojového rámce při vývoji aplikací. Bude navržen soubor testů vhodný pro vývoj nové funkce přidávané do již existující a fungující aplikace.

Doporučený rozsah práce

30-40

Klíčová slova

Agilní metody, Scrum, Project Manager, Project Owner, Informační systém, Vývoj software

---

Doporučené zdroje informací

BRUCKNER, T. *Tvorba informačních systémů : principy, metodiky, architektury.* Praha: Grada, 2012. ISBN 978-80-247-4153-6.

BUCHALCEVOVÁ, A. *Metodiky vývoje a údržby informačních systémů : kategorizace, agilní metodiky, vzory pro návrh metodiky.* Praha: Grada, 2005. ISBN 80-247-1075-7.

KADLEC, V. *Agilní programování : metodiky efektivního vývoje softwaru.* Brno: Computer Press, 2004. ISBN 80-251-0342-0.

ŠOCHOVÁ, Z. – KUNCE, E. *Agilní metody řízení projektů.* Brno: Computer Press, 2019. ISBN 978-80-251-4961-4.

Předběžný termín obhajoby

2022/23 ZS – PEF

Vedoucí práce

Ing. Jana Hřebejková

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 14. 7. 2022

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 11. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 30. 11. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Agilní metodiky vývoje software ve vybrané společnosti" jsem vypracoval samostatně pod vedením vedoucího bakalářské 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 bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 30.11.2022 _____

Poděkování

Rád bych touto cestou poděkoval vedoucí bakalářské práce Ing. Janě Hřebejkové za cenné rady a pomoc při vypracování práce

Agilní metodiky vývoje software ve vybrané společnosti

Abstrakt

Bakalářská práce se zabývá problematikou vývoje informačních systémů. Cílem práce je vymezit metodiky vývoje informačních systémů, s hlavním důrazem na agilní metody. Dílčím cílem je zpracování interní metodiky vývoje software v agilním prostředí vybrané společnosti. Na základě těchto východisek si pak práce klade za cíl navrhnout způsob testování nové funkce přidávané do již existující a užívané aplikace. V teoretické části práce jsou zpracována teoretická východiska metodik vývoje informačních systémů. Jsou zde vymezeny jednotlivé tradiční a agilní metodické přístupy pro vývoj software, u kterých jsou uvedeny jejich výhody a nevýhody. V praktické části práce je pak zpracována agilní metodika vybrané banky působící na trhu v České republice a interní podmínky a způsoby testování v agilním prostředí vybrané společnosti. Na základě takto zpracovaných metodik je v druhé části Vlastní práce navrženo vylepšení informačního systému vybrané banky přidáním nového tlačítka a k tomuto vylepšení je navržen soubor testů, sloužící k ověření kvality dodávaného vylepšení.

Klíčová slova: Informační systém, Vývoj software, Agilní metody, Rigorózní metody, Scrum, Tým, Testování software

Agile methods of software development in selected company

Abstract

Bachelor thesis deals with problematics of software development. The aim of the thesis is to define software development methods with main attention to Agile methods. Partial goal it to describe agile software development methods of chosen company. According to the methods the aim is to suggest tests to make improvement in application which is already used by the Bank. In the theoretical part are defined theoretical basis of software development. It describes main traditional and agile methods of software development and its main advantages and disadvantages. The practical part describes agile methods and agile testing methods of the chosen company which operates on the market in the Czech Republic. In the second part of the practical part is suggested software improvement of the bank system, which adds a new function to the application. To this adjustment are suggested test which would be used to evaluate the delivered functionality.

Keywords: Information system, Software development, Agile methods, Rigorous methods, Scrum, Team, Software testing

Obsah

1 Úvod

U každé společnosti se setkáváme s nutností řízení a koordinování činností různých osob a činností obecně ve společnosti. Aby tato koordinace byla cílená a systematická je pro ni potřeba stanovit základní postupy a metody postupů. Právě za tímto účelem vznikají metody vývoje informačních systémů. Jejich účelem je poskytnutí základních postupů a pomáhají manažerovi projektu k jeho efektivnímu řízení a dodržení stanovených pravidel a norem.

Dříve byl nejčastěji používaným způsobem řízení projektů Vodopádový model, který je charakteristický postupnou prací na jednotlivých částech projektu, přičemž je vždy pracováno pouze na jednom konkrétním kroku. Z praxe je však tento postup stejně jako spirálový model nepříliš efektivní, a to zejména u větších projektů. Z toho je patrné že jsou hledány nové metodiky a postupy lépe vyhovující soudobým požadavkům.

V dnešní uspěchané době je kladen stále se zvyšující důraz na rychlost a kvalitu dodávaných informačních systémů. To má za následek uplatnění nových principů pro vývoj software. Jedním z těchto přístupů jsou Agilní metody vývoje, kterým je tato práce z velké části věnována. Jedná se o dnes velmi moderní způsob vývoje informačních systémů, a to zejména díky své rychlosti dodání a možnostem neustálých testů oprav a změn systému.

Jedním z podniků uplatňujících agilní vývoj software je i vybraná Bankovní instituce. Ta má na svém intranetu dostupnou vlastní agilní metodiku pro řízení projektů vývoje software. Tato agilní metodika je zpracována v části nazvané Vlastní práce a následně je, na základě těchto vnitřních metodik a obecně známých pravidel agilního vývoje, navržen soubor testů pro vývoj nové funkcionality v již fungujícím a využívaném systému.

2 Cíl práce a metodika

2.1 Cíl práce

Práce si klade za cíl čtenáře seznámit s problematikou vývoje informačních systémů a vyložit základní metodiky využívané pro této vývoj. Hlavní důraz je pak kladen zejména na Agilní vývoj informačních systémů, který je v dnešní době hojně uplatňován mnoha společnostmi.

Dílčím cílem je zpracování agilní metodiky vybrané Bankovní společnosti, za využití poskytnutých informací o vnitřních metodikách a obecně známých pravidel. Na základě zpracování těchto metodik a základních pravidel agilního programování je navržen soubor testů, který by měl společnosti poskytnout potřebnou zpětnou vazbu při vývoji nové funkce v již existující aplikaci.

2.2 Metodika

Metodický postup této práce lze rozdělit do dvou větších celků. Teoretická část práce je založena na přehledu řešené problematiky a teoretických východisek. Toho je docíleno pomocí analýzy a rešerše vybraných odborných informačních, zejména pak knižních zdrojů.

Následná vlastní práce spočívá ve zpracování vnitřních podmínek vybrané Bankovní společnosti. Ta je řízena vlastními platnými metodikami a dokumenty, které jsou zaměstnancům volně přístupné na intranetu společnosti. Tyto metodiky jsou stanovovány příslušným oddělením společnosti a jsou v souladu s obecně platnými normami.

Na základě takto zpracovaných pravidel agilního vývoje je navržen soubor testů, který by byl uplatněn při vývoji nové funkce, v již existující a využívané aplikaci. Tento soubor testů odpovídá obecně platným pravidlům agilního testování popsaným v teoretické části práce.

3 Teoretická východiska

3.1 Informační systém

Vzhledem k tomu, že se práce věnuje vývoji informačních systémů, je velmi vhodné si nejprve definovat samotný pojem informační systém, ze kterého práce vychází. Informační systém je celek složený z počítačového hardwaru, souvisejícího softwaru a lidí. Hardware lze obecně definovat jako veškeré pevné (hmatatelné) části počítače, tedy jeho technické vybavení. Softwarem pak nazýváme veškerá data a programy počítače. V této práci je pojem informační systém dále používán jako synonymum softwarového programu. (Buchalcevová, 2005)

3.2 Vývoj informačních systémů

Vývoj neboli budování Informačních systémů chápeme jako součást podnikové informatiky. Lze jej vymezit jako *„systém informačních a komunikačních technologií, dat a lidí, jehož cílem je efektivní podpora informačních a rozhodovacích procesů a procesů správy a využívání znalostí na všech úrovních řízení podniku.“* Samotný vývoj informačních systémů se pak řídí platnými metodikami, ať už samotných organizací či obecně platnou metodikou. Pojem metodika představuje v obecném smyslu souhrn postupů a metod pro realizaci určitého úkolu. (Voráček, Rosický, 1997)

3.2.1 Historie evoluce metodik pro vývoj softwaru

Jedním z nejdůležitějších produktů softwarového inženýrství jsou metodiky vývoje aplikací. Ty prošly zejména během posledních let velkou řadou změn za účelem přizpůsobení se novým trendům, moderní architektuře aplikací a potřebám uživatelů i samotných vývojářů. Jejich úkolem je vždy zrcadlit aktuální situaci v čas svého vzniku, tak aby byl vývoj přizpůsoben konkrétním požadavkům kladeným v současném čase a situaci. Dílčím cílem těchto metodik je také odstranění nežádoucích nedostatků vývoje software, které se v příslušném období testování nejvíce projevily. (Kadlec, 2004)

Před samotným zavedením metodik, které se začaly uplatňovat v 80. letech minulého století, tedy v období prvopočátku vývoje programů, byl používán model „Napiš a oprav. Na následujícím obrázku vidíme jednoduché schéma tohoto modelu, který *„spočíval*

v sepsání aplikace, v jejím následném předání do provozu (spuštění) a v opravování chyb.“ (Kadlec, 2004)

Obrázek 1: Model napiš a oprav

Zdroj: (Kadlec, 2004), vlastní zpracování

V této době vznikaly nepříliš rozsáhlé a složité programové systémy, přesto byl tento model značně neefektivní. Vytvářet systémy tímto způsobem nebylo dále možné, proto byl v roce 1957 definován tzv. „Stargewise model životního cyklu“. Tento model byl striktně založen na posloupnosti jednotlivých fází. Jako tyto fáze určil: Definice problému, specifikace požadavků, architektura a návrh, implementace, integrace, provoz. Největším problémem, který tento model skýtal byla naprostá absence jakékoliv zpětné vazby. Po dokončení jedné fáze se pokračoval dál bez jakéhokoliv testování a ověření výsledků. (Awad, 2005)

Tato nedostatečnost tak zavdala v roce 1970 vzniknout "Vodopádovému modelu“ z angl. „Waterfall“ jehož autorem byl Winston W. Royce. V roce 1988 pak byl v článku Barryho Boehma definován „Spirální model“ z angl. „Spiral model“ Tyto modely jsou mimo jiných, také označovány za „Rigorózní“ či „Těžké metodiky“. Pohlížejí na vývoj softwaru jako na inženýrskou disciplínu a jsou využívány do dnešní doby. Zejména v poslední době je však největší důraz kladen spíše na metodické přístupy nežli na samotné metodiky. To má v současné době velký vliv na prosazení moderních, či módních agilních metod. V současnosti tak lze říci že převládají dva hlavní základní proudy vývoje software, a to rigorózní metodiky a agilní metodiky, které jsou popsány dále. (Leffingwell, a další, 2010)

3.3 Rigorózní metodiky

Základním předpokladem pro tyto tradiční metody je předpoklad, že budování informačních systémů lze popsat, plánovat, řídit a měřit. Za těžké jsou také označovány, kvůli své velké objemnosti a složitosti, která je zapříčiněna snahou o přesnou definici procesů, činností a vytvářených produktů. Kladou také důraz na objemnou dokumentaci a

jsou specifické direktivním řízením. Zpravidla jsou založeny na sériovém vývoji, při němž probíhají jednotlivé fáze odděleně a sekvenčně za sebou. Toto pravidlo však neplatí vždy, existují i rigorózní metodiky, které jsou založené na iterativním a inkrementálním vývoji. Iterativní vývoj lze chápat jako opakování jednotlivých fází při vývoji informačních systémů. (Kadlec, 2004)

Tradiční metodiky lze také definovat předpokladem, že jednotlivé procesy probíhající při vzniku softwaru je možno plně a platně identifikovat a definovat, přičemž je lze konzistentně opakovat. Toto tvrzení tedy vychází z předpokladu, že je možné definovat a opakovat (Buchalcevová, 2005):

- Problém
- Řešení
- Nositele řešení – vývojáře
- Prostředí

Zároveň jsou vhodné a úspěšné za předpokladu, že jsou splněny následující požadavky (Buchalcevová, 2005):

- Reálná možnost definovat předem všechny požadavky
- Neměnnost definovaných požadavků
- Opakovatelnost procesů
- Opětovná aplikace modelů

Z výše uvedeného je patrné že existuje mnoho různých tradičních metodik, které se od sebe mohou značně lišit. Pro názornost a porovnání jsou v práci uvedeny dvě vybrané hojně využívané metodiky: Vodopádový životní cyklus a Spirálový vývoj.

3.3.1 **Vodopádový životní cyklus**

3.3.1.1 **Charakteristika**

Jedná se o nejtradičnější model vývoje softwaru, který však v současné době bývá mnohými zavrhován a zatracován, zejména kvůli své nízké variabilitě. Přesto je však nadále hojně využívaným modelem, který je vhodný zejména pro menší a jednodušší projekty. Mezi jeho nesporné výhody patří jednoduchost, dobrá kontrolovatelnost manažerem a jednoduché použití. *„Vodopádový model je definován jako model procesu vývoje softwaru, v němž*

všechny podstatné fáze, typicky definice problému, specifikace požadavků, návrh a architektura, implementace, testování a provoz, jsou prováděny ve stanoveném pořadí s žádnými nebo minimálními iteracemi.“ Typickým znakem pro vodopádový model, který ho zároveň i definuje, je tedy sekvenčnost jednotlivých fází. Jednoduše řečeno vodopádový model sestává z několika fází, které se postupně následují, přičemž musí být dodržena posloupnost jednotlivých fází, tedy žádná fáze nemůže předběhnout fázi předchozí. (Kadlec, 2004)

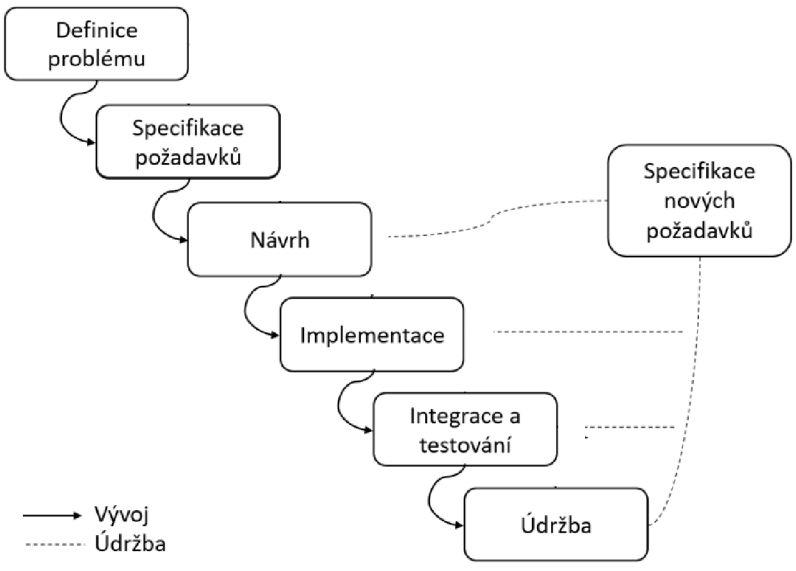
3.3.1.2 **Jednotlivé fáze vodopádového modelu a jejich charakteristika**

Vodopádový model se skládá z posloupnosti několika kroků a fází. Těmito fázemi jsou (Kadlec, 2004):

1. Definice problému, poznání zákazníka, proniknutí do cílové oblasti
2. Analýza a specifikace požadavků
3. Návrh systému
4. Implementace systému
5. Integrace a testování systému
6. Provoz a údržba

Tyto jednotlivé fáze jsou také patrné z přiloženého schématu zachycujícího základní strukturu vodopádového modelu životního cyklu.

Obrázek 2: Vodopádový model životního cyklu



Zdroj: (Kadlec, 2004), vlastní zpracování

Prvních pět fází životního cyklu lze označit za kategorii Definice a vývoj. V rámci těchto fází se lze pohybovat vždy pouze o jeden krok dopředu, nelze tedy žádnou fázi přeskočit. Teprve po dokončení všech těchto dílčích fází je produkt hotový a dochází předání produktu zákazníkovi. V této chvíli je pak možno se vrátit do kterékoliv jednotlivé fáze a provádět úpravy. (Despa, 2014)

Fáze definice problému

Hlavním úkolem v této fázi je definice záměru zákazníka. Ideálním prostředkem je návštěva zákazníka přímo na jeho pracovišti, a pochopení co zákazník od systému požaduje a očekává. Dále je potřeba pochopit jakou přesně agendu by měl systém zastat, tedy jakým způsobem a co by měl systém zákazníkovi usnadnit a jakým způsobem je tato agenda prováděna nyní, při absenci systému. Na základě této fáze je pak zpravidla definován dokument s názvem Úvodní studie, který by měl obsahovat veškeré zjištěné informace zmíněné v předchozím odstavci. V neposlední řadě by měl dále obsahovat informace o motivaci, proč by měl být systém dodán a také základní, stručný popis požadavků na systém. (Kadlec, 2004)

Fáze analýzy a specifikace požadavků

Hlavním úkolem Fáze analýzy je přesné pochopení problému zákazníka, který chce nasazením systému řešit. Dochází zde k podrobnému zkoumání, co přesně by měl systém dělat a zpracovávat a také jakým způsobem. Zprvu by se mohlo zdát, že jde o jednoduchou fázi vývoje, opak je ale pravdou. Největší důraz je zde kladen právě na analýzu požadavků zákazníka, jelikož při jejich nepochopení by mohlo dojít k vytvoření špatného či neúplného produktu, na což by se přišlo až při předávání finálního produktu zákazníkovi. V rámci specifikace požadavků určujeme, co by měl systém dělat, ale ne to, jakým způsobem. Jejím cílem je tedy popis aplikace v jazyku zákazníka. Výstupem z analýzy je dokument nazývaný Specifikace požadavků, na který jsou kladeny vysoké nároky. Tyto nároky jsou kladeny zejména na strukturu, formu, obsah a způsob vytváření. Základním východiskem vodopádového modelu je pak předpoklad, že na konci fáze analýzy budou jasně a přesně definovány požadavky, které se již do okamžiku odevzdání aplikace zákazníkovi nezmění. (Kadlec, 2004)

Fáze návrhu a vytváření architektury

Zjednodušeně řečeno se jedná o fázi, ve které dochází k budování řešení problému, který je popsán ve Specifikaci. Jejím úkolem tak je na základě specifikace požadavků navrhnout vhodnou architekturu systému pro daný projekt. Architekturou jsou veškerá rozhodnutí ohledně modulů, vrstev, technologií a vývojových prostředí která je potřeba učinit. Tato rozhodnutí jsou prováděna na základě několika kritérií. Dle zkušeností a schopností vývojového týmu, charakteru vybraného projektu a také prostředků vyhrazených na tento projekt, a to jak finančních, tak i lidských, časových, či technologických. (Despa, 2014)

Obecně lze postup návrhu architektury zhruba definovat takto (Kadlec, 2004):

1. Určení implementačního prostředí, vývojového nástroje, programovacího jazyka a všech potřebných technologií
2. Vytvoření architektury systému, logické rozdělení systému na subsystémy a další funkční celky
3. Výběr a návrh implementačních modulů, „namapování“ logického návrhu do fyzické – implementační – struktury
4. Definice chování modulů, specifikace práce s daty (uchovávání, formáty apod.)
5. Uspořádání a diskuze zúčastněných stran nad výsledky, upřesnění architektury

Fáze implementace

Fáze implementace je z hlediska její specifikace vcelku jednoduchá. Hlavním cílem této fáze je na základě požadavků naprogramovat aplikaci. Tedy požadavky „převést“ z jazyku zákazníka do zdrojového kódu. V této fázi nedochází k žádné změně či definici požadavků nýbrž pouze na základě dvou již dříve zmíněných a vytvořených dokumentů je vytvořen zdrojový kód přesně podle požadavků zákazníka. (Kadlec, 2004)

Fáze integrace a testování

Cílem Fáze testování je zajistit, aby aplikace jako celek i každá její drobná funkcionalita, či část kódu, fungovala tak, jak je požadováno. Vývojový tým se tedy nesmí zaměřit pouze na otestování základních funkcí, nýbrž i na každou podmínku a kombinaci

výstupních dat. V praxi však není dost dobře možné toto všechno otestovat a nasimulovat, proto existuje mnoho různých přístupů a způsobů k testování.

Pro jednoduchost lze naznačit, že v zásadě existují 2 druhy testování: (Kadlec, 2004)

- White-box (metoda bílé skříňky): testujeme funkce na základě znalosti vnitřní struktury (otestujeme všechny „cesty“ v programu)
- Black-box (metoda černé skříňky): testujeme funkce na základě očekávaného chovaní a specifikace (nezajímáme se o vnitřní implementaci, ale kontrolujeme výsledky)

Fáze provozu a údržby

Tato fáze probíhá za předpokladu, že zákazník vyvinutý systém akceptuje a je s ním spokojen. Probíhá od předání požadovaného systému zákazníkovi až do doby, kdy je vyřazen z provozu, zpravidla nahrazen novějším systémem. Během této fáze dochází k procesu neustálých úprav, oprav, vylepšování a ladění aplikace. (Kadlec, 2004)

3.3.1.3 Výhody a nevýhody vodopádového modelu

Přestože, jak bylo zmíněno, se jedná o dnes již poněkud odvrhovaný model, má několik nesporných výhod. Hlavní výhodou je jeho jednoduchost, kdy díky lineárnímu postupu skrze jednotlivé fáze je vždy jasně patrné, ve které předem přesně dané fázi se právě nachází. Navíc lze z hlediska vedení projektu jasně určit rámcový harmonogram pro postup po jednotlivých fázích a specifikace požadavků na výstupy a dokumenty. Z tohoto důvodu je také jednodušší pro řízení, protože je možná jednodušší orientace a alokace zdrojů. *„To však platí jenom zdánlivě, neboť vývojový tým při postupu podle vodopádu před sebou „tlačí“ dosud netušená rizika, která mohou všechny propočty rázem obrátit naruby.“* (Kadlec, 2004)

Ačkoliv byla v přechozím odstavci jednoduchost značena za největší výhodu, může být zároveň i nevýhodou. Zejména u větších a komplexnějších projektů je totiž Vodopádový model příliš jednoduchý a nepružný, jelikož není možné do vývoje nijak vstupovat a v průběhu měnit jeho zadání. Velkou nevýhodou je pak také dodání produktu zákazníkovi formou „Velkého třesku“ (Big Bang). Zákazník kromě zadání projektu není spojen s jeho samotným vývojovým cyklem, a je mu pouze na konci cyklu předán. To může mít za následek změnu požadavků či nespokojenost zákazníka s výsledkem ať už z hlediska funkcionalit, či uživatelského rozhraní. (Buchalcevová, 2005)

3.3.2 Spirálový vývoj

3.3.2.1 Charakteristika

Spirálový vývoj byl představen světu v roce 1985 a navázal tak na Vodopádový model. Jeho účelem byla eliminace některých chyb a nedostatků vodopádového cyklu. Hlavním rozdílem ve vývojovém cyklu, který Spirálový model přinesl je tzv. iterativní přístup. Celý projekt je tak rozložen do několika iterací, přičemž každá iterace v sobě nese posloupnost kroků podobných vodopádovému modelu. Jedná se o riziky řízený postup, tedy veškerý postup v projektu je založen především na opakovatelnosti a důslednosti provádění analýzy všech rizik. Analýza rizik má dva hlavní cíle (Kadlec, 2004):

- Zjistit předem možná ohrožení hladkého průběhu projektu
- Připravit reakce na tyto události

3.3.2.2 Posloupnost kroků a fází

Spirálový vývoj probíhá po spirále rozdělené do čtyř kvadrantů, což je patrné z následujícího obrázku. Levý horní kvadrant je v každé iteraci určen pro stanovení cílů. Stanovují se zde cíle, alternativy řešení, překážky a omezení dalšího postupu. V pravém horním kvadrantu probíhá analýza rizik vzhledem k následující iteraci. Postupy probíhající v následující pravém dolním kvadrantu lze souhrnně nazvat jako realizace. Poslední levý spodní kvadrant je vyhrazen pro zhodnocení iterace a plánování dalšího postupu. Vývoj podle spirálového modelu lze rozdělit do několika fází ve kterých probíhá. Jedná se o fáze (Buchalcevová, 2005):

1. Co nejpřesnější definice požadavků, cílů, alternativ a omezujících podmínek.
2. Předběžný návrh systému.
3. Zhotovení zjednodušeného prototypu systému dle zadaných požadavků.
4. Vyhodnocení prototypu a návrat k prvnímu kroku, tedy definice požadavků na nový prototyp.

V rámci spirálového vývoje existují tři základní druhy prototypů. Ilustrativní prototyp, který je využíván zejména při dialozích se zákazníkem, a klade hlavní důraz na vzhled uživatelského rozhraní. Funkční prototyp, který se naopak soustředí na funkční jádro systému. Nachází se zde minimum základních funkcí, které jsou následně přidávány. A Ověřovací prototyp, který se může zaměřit na jakoukoliv část systému, která je na řadě k

realizaci na základě analýzy rizik. Hlavním účelem je ujasnění požadavků a technologií, které mají být implementovány. (Kadlec, 2004)

3.3.2.3 **Výhody a nevýhody spirálového modelu**

Spirálový model je vhodný zejména pro velké, komplexní projekty, a to díky své celkové robustnosti a důrazu na plánování, ověřování a analýze rizik. Zároveň díky, v podstatě neomezenému počtu iterací, nabízí možnost zpracování prakticky libovolně velkého projektu. Tento model poskytuje vývojovému týmu časté možnosti změny požadavků mezi jednotlivými iteracemi, které jsou však ohroženy řadou rizik. Dle samotného autora modelu je hlavní výhodou nezávislost modelu na konkrétní metodice či strategii, díky čemuž jej lze vhodně využít pro řadu konkrétních předem definovaných postupů v závislosti na konkrétním projektu a organizaci. (Buchalcevová, 2005)

Nevýhodou Spirálového modelu, stejně jako u modelu vodopádového, je dodání softwaru jako celku vždy až po skončení posledního cyklu. V průběhu vývoje sice dochází k vývoji prototypů, nicméně ty jsou určeny pouze pro analýzu úzké části systému a nejsou zpravidla zahrnuty ve finálním produktu. Další nevýhodou je pak lidský faktor. Řízení a analýza rizik je velmi náročná činnost a je potřeba správně zjistit jejich zdroje. Je tedy nutné rozlišit rizika na ta závažná a méně důležitá, tak aby byla správně řešena. To může být do značné míry ovlivněno subjektivním názorem a může vést až ke krachu projektu. Velkým nedostatkem je absence propracované a detailní metodiky. Vývoj sice popisuje rizika a jejich analýzu, ale již nic neříká o konkrétních metodách jako jsou kontrakty, specifikace, dokumentace a milníky. Stejně jako u vodopádového modelu je i ten spirálový v současné době spíše na úpadku. Přestože jej lze považovat za jednoznačný milník vývoje softwarového inženýrství je v posledních letech využíván stále méně. To zejména kvůli novějším a novějším propracovaným metodikám a také částečné zastaralosti a nepružnosti, kvůli které není nejvhodnější volbou pro nové druhy aplikací. (Despa, 2014)

3.4 Agilní metodiky

V dnešní dynamické době, kdy změny probíhají takřka neustále, bylo potřeba aby došlo k požadovaným změnám také na poli metodik vývoje informačních systémů. Od dob kdy byly vynalezeny dříve popsané tradiční metodiky, se situace na trhu se softwarem

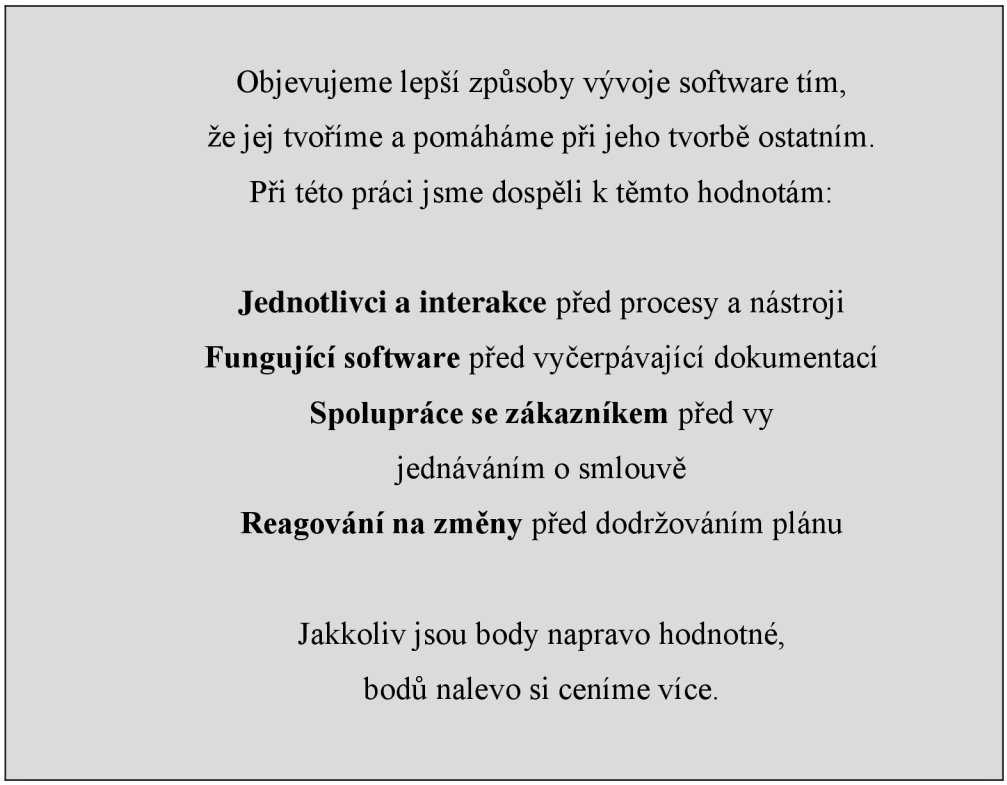
značně změnila. Lidé požadují stále složitější a komplexnější programy, přičemž odmítají dlouho čekat na dodání systému. Základní myšlenkou, kterou agilní metodiky předkládají, je vyvinutí systému v co nejkratším čase a předložení zákazníkovi. Na základě zpětné vazby zákazníka pak dochází k úpravám a opravám systému. Agilní systém se tak snaží o eliminaci zbytečné byrokracie a papírování a zjednodušení procesů změny. (Myslín, 2016)

Základním rozdílem oproti tradičním metodám vývoje software je rozdílný pohled na 3 základní proměnné pro vývoj aplikací. Jedná se o funkcionalitu, čas a zdroje. Zatímco tradiční metody berou funkcionalitu (požadavky zákazníka) jako fixní, a naopak zdroje a čas jako variabilní. Oproti tomu v agilním vývoji jsou za fixní veličiny brány čas a zdroje, které jsou na začátku projektu stanoveny zadavatelem a funkcionalita se v průběhu vývoje mění a přizpůsobuje se požadavkům zákazníka. Agilní metodika není pouze jedna, naopak je jich mnoho. Každá z agilních metodik je svým způsobem specifická, ale všechny jsou postaveny na stejném základním principu a hodnotách. V roce 2001 se představitelé agilních přístupů sešli, aby sjednotili základní myšlenky agilního vývoje. Vznikla „Aliance pro agilní vývoj softwaru“, jejíž členové podepsali i nově vzniklý „Manifest agilního vývoje softwaru“. (Beck, a další, 2001)

3.4.1 **Manifest agilního vývoje softwaru (Agile Manifesto)**

Agilní manifest, ačkoliv se nejedná o oficiální dokument, je základem celého agilního přístupu. Jedná se o soupis předpisů, jakési prohlášení, sepsané hlavními představiteli nových přístupů k vývoji softwaru. Tento manifest je volně dostupný na internetu a český překlad jeho formulace je následovný:

Obrázek 3: Manifest agilního vývoje



Zdroj: (Beck a spol., dostupné z: https://agilemanifesto.org)

Zároveň byly definovány základní principy stojící za agilním manifestem (Beck a spol., dostupné z: https://agilemanifesto.org):

1. Naší nejvyšší prioritou je vyhovět zákazníkovi časným a průběžným dodáváním hodnotného softwaru.
2. Vítáme změny v požadavcích, a to i v pozdějších fázích vývoje. Agilní procesy podporují změny vedoucí ke zvýšení konkurenceschopnosti zákazníka.
3. Dodáváme fungující software v intervalech týdnů až měsíců, s preferencí kratší periody.
4. Lidé z byznysu a vývoje musí spolupracovat denně po celou dobu projektu.
5. Budujeme projekty kolem motivovaných jednotlivců. Vytváříme jim prostředí, podporujeme jejich potřeby a důvěřujeme, že odvedou dobrou práci.
6. Nejúčinnějším a nejefektnějším způsobem sdělování informací vývojovému týmu z vnějšku i uvnitř něj je osobní konverzace.

7. Hlavním měřítkem pokroku je fungující software.
8. Agilní procesy podporují udržitelný rozvoj. Sponzoři, vývojáři i uživatelé by měli být schopni udržet stálé tempo trvale.
9. Agilitu zvyšuje neustálá pozornost věnovaná technické výjimečnosti a dobrému designu.
10. Jednoduchost, umění maximalizovat množství nevykonané práce, je klíčová.
11. Nejlepší architektury, požadavky a návrhy vzejdou ze samo-organizujících se týmů.
12. Tým se pravidelně zamýšlí nad tím, jak se stát efektivnějším, a následně koriguje a přizpůsobuje své chování a zvyklosti.

3.4.2 **Extrémní programování**

Extrémní programování, zkratka XP, je asi nejrozšířenější a nejpoužívanější agilní metodikou, i když v poslední době je stále častěji vytlačována metodou Scrum, které je věnována další kapitola. Tato metodika je vhodná zejména pro malé a střední firmy, zpravidla je využívána pro týmy v počtu jednotek programátorů. Jejím filozofickým východiskem je dle Kadlece přesvědčení, že: *„Jediným exaktním, jednoznačný, změřitelným, ověřitelným nezpochybnitelným zdrojem informací je zdrojový kód“*. Extrémní programování dává vývojářům možnost přizpůsobovat produkt přáním zákazníka, a to na základě komunikace se zákazníkem a jeho zpětné vazby, která probíhá neustále. Kolem vývoje produktu je tak postaven tým složený nejen z vývojářů, ale také manažerů a zástupců hájící zájmy a názory zákazníka. Tyto týmy jsou tak postaveny na hodnotách spolupráce, samoorganizace, zpětné vazby a respektu vůči ostatním členům týmu. (Kadlec, 2004)

Základní myšlenkou extrémního programování je jednoduchost se zaměřením na výsledek. Spoléhá se na věci, které se osvědčily a fungují a říká, že pokud něco dobře funguje, proč to tak nedělat pořád a stejně. Systém je tak ponechán v nejjednodušší podobě, ve které pracuje jak má, a vyhovuje požadavkům zákazníka. Jak bylo již dříve zmíněno, agilní programovaní se od tradičního odlišuje rozdílným pohledem na tři základní proměnné funkcionalitu, čas a zdroje. Extrémní programování k těmto třem proměnným přidává ještě jednu nazvanou šíře zadání. Ta označuje množinu funkcí požadovaných zákazníkem. Zároveň je z ní patrné, které funkce jsou pro zákazníka klíčové a které lze naopak upozadit, čímž se zadání může měnit a dodržuje se pouze určitá šíře a mantinely zadání. Pod těmito

základními čtyřmi hodnotami se nachází ještě pátá podprahová veličina, zvaná respekt. (Kadlec, 2004)

3.4.2.1 **Výhody a nevýhody extrémního programování**

Jednou z hlavních výhod extrémního programování je komunikace. K časté komunikaci dochází jak v rámci vývojového týmu, tak také pomocí zpětné vazby od zákazníka nebo jeho zástupců. Charakteristickou výhodou je také přímočarý postup k cíli bez lpění na formalitách, přičemž menší a rychle vytvořený plán bere v potaz, že bude docházet k plánování vývoje a jeho úpravám. (Despa, 2014)

Hlavní nevýhodou je riziko nedodání programového vybavení včas, nebo dodání nekompletního programového vybavení. Program pak musí být dodělán, či opraven což může vzhledem k časovému prodlení vést k nezájmu zákazníka, či irelevantnosti produktu. Nevýhodou či obtíží při tvorbě programu může být také důraz na jednoduchost praktické realizace. Pro nezkušené, či neznalé programátory může být složité dělat věci co nejjednodušeji, či sobě nebo ostatním přiznat neznalost nejzákladnějších principů. (Despa, 2014)

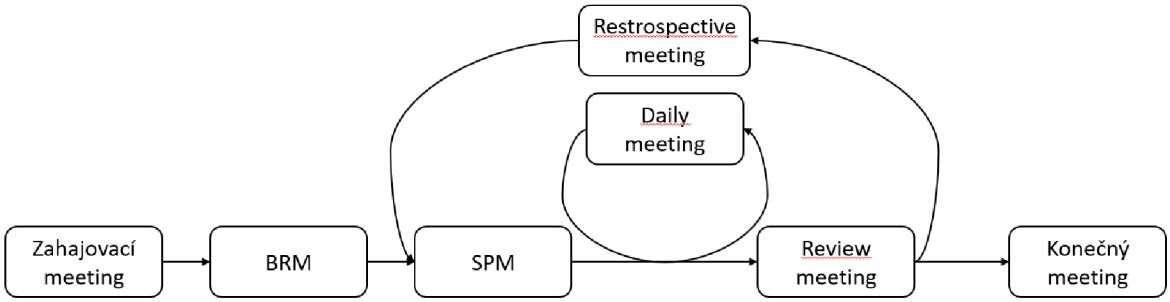
3.4.3 **Scrum**

„Scrum vychází z objektově orientovaného přístupu, díky němuž odpovídá každý vývojář za množinu objektů s jasně definovaným chováním a rozhraním.“ Základním členěním vývojového procesu ve scrumu jsou takzvané sprinty. Délka a počet jednotlivých sprintů záleží na konkrétní povaze projektu a velikosti týmu. Po každém sprintu je zákazníkovi ukázán výsledek, aby byla získána zpětná vazba. V rámci těchto sprintů nejsou definovány žádné konkrétní procesy, ale jsou požadovány schůzky na denní bázi, na kterých jsou určeny konkrétní činnosti. Tyto denní schůzky nahrazují centrální plánování. (Kadlec, 2004)

Základní veličinou pro Scrum je tým. U jednodušších projektů se jedná o jeden tým pracující na jednom projektu. Zde platí pravidlo zastupitelnosti, kdy nejsou nijak zásadně oddělené role analytiků, testerů a vývojářů, nýbrž lidé se mohou navzájem zastoupit. Členové tohoto týmu si navzájem pomáhají a organizují se, v ideálním případě pak pracují všichni na jednom místě, v jedné místnosti, aby se jim dobře spolupracovalo. Každý vývojový tým má právě jednoho Scrum Mastera a jednoho Product Ownera. Product Owner v projektu zastupuje zákazníka a jeho zájmy, tedy určuje, jak bude produkt vypadat a jeho

funkcionality. Scrum Master je manažerská pozice, ačkoliv se od klasických manažerských pozic značně liší. To zejména, jelikož Scrum předpokládá schopnost práce v týmu bez nadřízené osoby. Tento manažer se stará se o to, aby se programátoři mohli soustředit jen na práci, vyřizuje jejich požadavky a obstarává komunikaci s okolím. Zároveň také motivuje tým k lepším výsledkům. (Myslín, 2016)

Obrázek 4: Vývojový cyklus řízený metodikou SCRUM



Zdroj: (Myslín, 2016), vlastní zpracování

Celý tento cyklu lze zjednodušeně rozdělit do tří etap (Myslín, 2016):

1. **Zahájení** – Fáze projektu do počátku samotného vývoje, cílem je ujasnění formy spolupráce, definice toho, co se vlastně bude vyvíjet, rozdělení rolí a kompetencí, vzájemné seznámení vývojového týmu.
2. **Samotný vývoj** – Fáze ve které probíhá samotný vývoj aplikace. Tento vývoj je, jak už jsme několikrát naznačili, iterativní. Jednotlivé iterace se nazývají sprinty.
3. **Ukončení** – Fáze projektu, ve které je výsledný produkt finálně otestován, akceptován, nasazen a kdy je projekt finálně ukončen.

3.4.3.1 **Artefakty a pojmy ve Scrumu**

„SCRUM, stejně jako jakákoliv metodika, používá některé své pojmy a artefakty, se kterými dále pracuje. Na rozdíl od mnoha jiných metodik jich není naštěstí mnoho, ale o to je důležitější těch několik základních dobře pochopit.“ (Myslín, 2016)

Uživatelský příběh je uživatelem vytvořený popis, který nám udává, co by měl program umět neboli požadavky na něj. Jedná se tedy o obecnou informaci o jednotlivých funkcích, které by měl program umět a je napsán v jazyce zákazníka.

Product Backlog je kompletní seznam uživatelských požadavků setříděný podle priority pro zákazníka a náročnosti na vývoj. Vlastně jde o formu úplné specifikace vyvíjeného softwaru a zpravidla je vizualizován ve formě tabulky či seznamu.

Sprint Backlog je seznam úkolů, které je nutno implementovat v rámci aktuálního sprintu, čímž je podmnožinou Product Backlogu. Při přípravě následujícího sprintu je potřeba definovat Sprint Backlog, tedy uživatelské příběhy, kterými se v rámci tohoto sprintu budeme zabývat

Riziko je ve Scrumu velmi důležitou veličinou. Lze jej zařadit k metodám silně dbajícím na rizika, ta jsou revidována v každém postupu práce. Formují tak nejenom náplně iterací, ale i obsah dokumentů, funkcionalitu verzí a další důležité atributy

3.4.3.2 **Schůzky ve Scrumu**

V průběhu vývojového cyklu je potřeba mnoha pravidelných schůzek. Ty lze rozdělit na tři základní typy, které se liší jak v obsahu, tak četnosti. (Myslín, 2016)

Plánovací schůzky vždy předcházejí nějaké aktivitě, kterou se snaží naplánovat. Zpravidla se tým schází, aby naplánoval další sprint, případně celý projekt. Výsledkem této schůzky je logický plán, který může mít různé podoby

Hodnotící schůzky oproti plánovacím schůzkám probíhají vždy po dokončení, nějakého úkolu, sprintu, či celého projektu. Hodnotí se zde dosažení stanovených cílů a výsledků.

Každodenní schůzky probíhají, jak již název napovídá, každý den. Jejich cílem je zhodnocení práce za předchozí den a stanovení práce na další den. Zároveň je zde i upravován plán vytvořený v rámci plánovací schůzky. Členové týmu tak mají neustálý přehled o řešených problémech a mohou se zastoupit či si případně vypomoci.

3.4.3.3 **Výhody a nevýhody Scrumu**

Obrovskou a hlavní výhodou metody Scrum je schopnost pružně reagovat na jakékoliv změny požadavků v průběhu práce na projektu. Každodenně reflektuje na změny požadavků

a hledá rizika z nich vyplývající. Poskytuje tak vývojářům možnost navrhnout optimální řešení, a to dle svého uvážení. Další výhodou je týmovost, díky vysoké spolupráci a častým schůzkám pracují členové týmu efektivněji a jsou si bližší. (Šochová, a další, 2019)

Nevýhodou metodiky Scrum je její obtížné zavedení ve společnosti. Je jednodušší jej zavést ve společnosti, která již má zkušenosti s agilním vývojem a zůstane tak u stejného provádění činností, přičemž převezme pouze způsob vedení projektu. Scrum neodpovídá obvyklé hierarchii řízení v tradiční společnosti. Manageři si musí zvyknout na určitou ztrátu kontroly nad klasickým způsobem řízení úkol-report. To je také způsobeno tím, že metodika je spíše souhrn obecných vzorů, nežli specifikace konkrétních kroků a postupů. (Buchalcevová, 2005)

3.5 Testování v Agilním prostředí

Testování v agilním prostředí je typické tím, že k němu dochází velmi často, a to vždy okamžitě po napsání nové části kódu. Požadavek je tak v rámci dané iterace považován za dokončený vždy až tehdy, pokud byl řádně otestován. Jedná se o proces zajišťující zpětnou vazbu ohledně funkčnosti dodávaného softwaru, a to tak aby byl zákazníkovi doručen kvalitní a použitelný software. Tento software by měl být v ideálním případě otestován natolik aby byl připraven na všechny možné scénáře, nicméně z praxe toto není možné. (Buchalcevová, 2005)

3.5.1 Principy agilního testování

Použité metody pro testování v agilním prostředí se v zásadě příliš neliší od tradičních metod. Agilní metody nicméně volí jiný přístup a lépe uzpůsobené metody pro tento styl vývoje. Principy společně přináší týmu obchodní hodnotu a celý tým dohromady přebírá odpovědnost za dodání softwaru nejvyšší kvality, který bude odpovídat požadavkům zákazníka. Hlavní principy a postupy důležité pro agilní testery lze shrnout do těchto bodů (Crispin, et al., 2009):

- Provádět nepřetržitou zpětnou vazbu
- Dodávat zákazníkovi hodnotu a vyhovět jeho prioritám
- Umožnit osobní komunikaci
- Mít odvahu

- Udržovat jednoduchost ve všech směrech
- Neustále se zlepšovat a dělat práci lépe
- Pružně reagovat na změny
- Organizovat se samostatně
- Zaměřit se na lidi, jejich potřeby a udržovat respekt

3.5.2 Agilní týmy

Zákaznický tým (Customer Team)

Zákaznický tým obsahuje obchodní zástupce, product ownera, produktového manažera, zákazníka atd. Jedná se o všechny osoby podílející se na zadání projektu, který je zpracován vývojovým týmem. Komunikuje s vývojáři ohledně každé iterace, zodpovídá dotazy a píše příklady na tabuli.

Vývojový tým (Developer Team)

Každou osobu zodpovědnou za doručení produktového kódu nazýváme vývojářem a je součástí vývojového týmu. Agilní principy podporují členy týmu, aby si brali nejrůznější úkoly a byli schopni splnit jakékoliv zadání v rámci projektu.

3.5.3 Druhy testování

3.5.3.1 Dle fáze testování

Komponentové testy – Tyto testy píšou obvykle přímo programátoři, a jsou základním prvkem pro ověření funkčnosti kódu. Jedná se o testy zaměřené na testování zdrojového kódu do nejmenších detailů a podrobností. Testy jsou zpravidla automatizované, a slouží k kontrole vývojového týmu o funkčnosti kódu.

Integrační testy – Zaměřují se na testování kompatibility nově integrovaných komponent se samotnou aplikací, do které jsou začleňovány, a také vzájemnou interakci mezi jednotlivými systémy a komponentami aplikací.

Systémové testy – Ověřují funkčnost systému jako celku, jakým způsobem pracuje po celou délku daného procesu. Na základě těchto výstupů, které jsou prezentovány zákazníkovi, je následně vyhodnoceno, zdali bude software předán zákazníkovi.

Akceptační testy – Jsou realizovány zákazníkem a jejich cílem je ověřit hodnotu a funkčnost dodaného produktu.

3.5.3.2 Dle způsobu realizace testů

Manuální testy – Tento způsob testování je v dnešní době spíše nahrazován automatizovaným testováním, nicméně i přesto má stále své uplatnění. Testy provádí samotní testeři, a to dle předem definovaných testovacích scénářů.

Automatizované testy – Často jsou prováděny pomocí skriptovacích jazyků a testují opakující se scénáře a úkoly.

4 Vlastní práce

4.1 Interní podmínky agilních metodik vybrané bankovní společnosti

Cílem této metodiky je popsat způsob fungování agilních týmů a jejich vzájemnou koordinaci a vychází z doporučení a metodického rámce Scaled Agile Framework (SAFe). Tato agilní metodika je určena nejen pro osoby fungující v různých rolích v agilních týmech, ale i stakeholderům, kteří s těmito týmy spolupracují. Cílem agility je dosahování lepších business výsledků, a to prostřednictvím zaměření omezených zdrojů na dělání „správných věcí". „Správné věci" jsou takové, které přinášejí největší hodnotu pro banku a její klienty. Cílem agilního vývoje tedy není dodat více požadavků za časovou jednotku, ale dodat co největší hodnotu pro banku a její klienty.

Agilita není pouze jiný způsob dodávky projektů (s využitím jiných rolí a jiných typů koordinačních schůzek). Implementace agility znamená podstatnou změnu kultury banky spočívající ve změně hodnocení úspěšnosti změnových aktivit a v etablování autonomních agilních týmů schopných end-to-end dodávky řešení, nebo jeho části. Ve změně vztahu mezi managementem a těmito agilními týmy ve smyslu decentralizace některých typů rozhodování. Management stanovuje cíle a vytváří podmínky pro dobré fungování agilních týmů. Agilní týmy hledají cestu, jak stanovené cíle s omezenými zdroji co nejefektivněji naplnit.

4.1.1 ART (Agile Release Train)

ART představuje dlouhodobé stabilní seskupení více agilních týmů, zpravidla 5-12, které spravuje a neustále vylepšuje jeden či více produktů. Je postaven nad klíčovými strategickými iniciativami, které by měly end-to-end zajišťovat konkrétní zákaznickou potřebu. Cílem ART je zprostředkovávat vše od definování požadavků na produkty, vývoj až po jejich provoz. Agilní týmy využívají metodu Scrum. ART má přibližně 50-125 členů, kteří by měli být dedikováni full time, funguje velmi stabilně a dlouhodobě, a účelem jeho založení není jedna konkrétní dodávka.

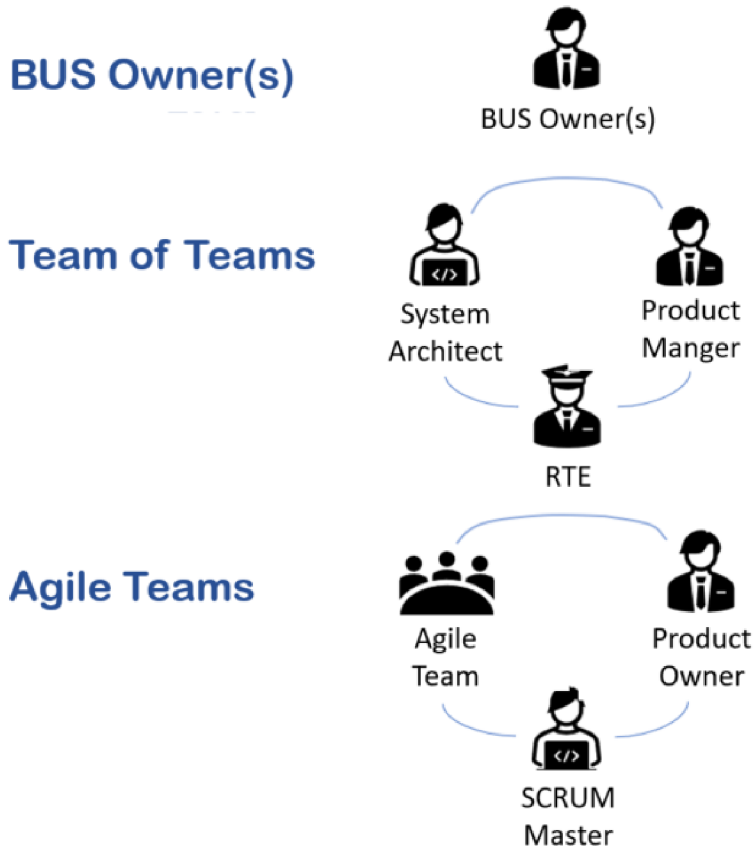
ART dodává v kvartálních cyklech, které se označují jako PI (Program inkrementy) a tradičně se skládají z 5-6 iterací. Agilní týmy pracují v iteracích, které trvají dva týdny. Celé ART dodržuje shodnou kadenci neboli začátek a konec všech iterací a celého Programového Inkrementu, a to z důvodu usnadnění spolupráce a koordinace aktivit. Toto nastavení je

dodržováno i mezi ostatními týmy ART, aby v případě rozsáhlejších řešení mohli spolupracovat a vytvořit tak Solution Train.

4.1.2 Agilní týmy a role v SAFe

Pro zachování kvalitního přenosu informací preferuje společnost zakládání menších týmů o velikosti 5-11 členů. Týmy by měly být samoorganizované a mít sílu o příslušných tématech samy rozhodovat. To v praxi znamená, že si členové týmu sami určují týmová pravidla a rozdělují veškeré kompetence a úkoly. U jednotlivých členů týmu je také důležité nastavení mysli (z angl. Mindset) a to zejména vůči spolupráci se svými kolegy, řešení problémů a podstupování výzev. V rámci agilního prostředí SAFe a agilních týmů jsou určeny základní role, které můžeme vidět dle příslušných úrovní na obrázku níže.

Obrázek 5: Role v prostředí SAFe



Zdroj: Vlastní zpracování

Product Manager - Odpovědností Product Managera je definice a rozvoj produktů, které jsou žádány ze strany zákazníka a zastřešuje společné očekávání ve spolupráci s týmem Product Ownerů.

System Architect – Je odpovědný za vydefinování odborné a architektonické vize ART, rozpracovává řešení, plánuje a vytváří Architektonickou mapu.

RTE (Release Train Engineer) – Vede a organizuje ART události a procesy, pomáhá odstraňovat překážky uvnitř i vně ART a asistuje týmům při plánovaných dodávkách.

Product Owner – Rozvíjí konkrétní produkt nebo jeho části a snaží se maximalizovat jeho hodnotu pro zákazníka. Formuluje Uživatelský příběh a stanovuje priority jednotlivých položek.

SCRUM Master – Má na starosti efektivní fungování a dodržování metody Scrum v rámci týmu. Koučuje tým, stará se o jeho rozvoj a motivuje členy týmu k lepším výsledkům.

Agilní tým – Zodpovídá za end-to-end dodávku funkčního řešení, které maximalizuje hodnotu pro zákazníka. Tedy modeluje, analyzuje, vyvíjí, testuje, nasazuje i provozuje.

4.1.3 SAFe Ceremonie

Klíčovou součástí agilních metodik SAFe jsou agilní ceremonie, ze kterých vychází všechny bankou využívané metodiky. SAFe ceremonie dělíme dle úrovní na:

- Team of Teams, kam spadá PI Planning,
- Inspect & Adapt,
- Scrum of Scrums,
- System Demo
- PO sync.

Druhou úrovní je týmová úroveň, pod kterou spadá:

- Itteration planning,
- Iteration Retrospective,
- Daily Stand-UP,
- Baclog Refinement
- Iteration Review

4.1.3.1 **PI Planning**

Program Increment Planning je stěžejní událostí celého programového inkrementu. Za organizaci ceremonie odpovídá RTE s pomocí svého týmu Scrum Masterů. RTE má na starost nejen organizační a koordinační stránku schůzky, ale dále odpovídá také za připravenost klíčových účastníků. Této aktivity se účastní všichni členové agilních týmů, Team of Teams a Business Owners. Očekávaným výsledkem je pak domluva všech zúčastněných na shodných očekáváních. Událost trvá dva dny a jejím účelem je sladění ohledně mise, vize i Businessových priorit a naplánování nadcházejícího Programového Inkrementu pro celý ART. Výsledkem PI Planningu jsou výstupy pro jednotlivé členy ARTs.

Tabulka 1: PI Planning

Výstupy z PI Planningu	Za přípravu odpovídá	Poznámky
Potvrzené PI Objectives	Agilní týmy	Business Owners přidělují PI Objectives Business value
Program Board	RTE	Board zobrazuje termíny dodání nových úkolů a významné milníky pro PI
Týmový plán a Týmový Backlog	Agilní týmy	Předběžný rozpad uživatelských příběhůdo itearcí do následujícího PI
Identifikovaná rizika	Agilní týmy	Tým identifikuje možná rizika, se kterými se následně pracuje
Identifikované závislosti	Agilní týmy	Tým identifikuje možné závislosti, se kterými se následně pracuje, případně je vede jako rizika
ART dokument	RTE	Dokument, který obsahuje všechny výstupy z PIPu, následně je předkládán k revizi

Zdroj: Vlastní zpracování

4.1.3.2 **Iteration Restrospective**

V rámci této ceremonie se řeší jak kvalita dodávek, tak fungování a nastavení týmu či ostatní vnější vlivy. Probíhá po Iteration Review a ideálně před Iteration Planningem, aby bylo možné konkrétní kroky pro zlepšení aplikovat již v další iteraci. Retrospektivy se účastní agilní tým včetně Product Ownera a Scrum Mastera, nicméně si dle potřeby můžou přizvat dalšího účastníka. Tým se zaměřuje na klíčové problémy, nicméně dlouhodobě se věnuje všem těmto oblastem:

- Zhodnocení konkrétních výstupů iterace / sprintu

- Fungování metody Scrum a jejích eventů
- Fungování agilního týmu a všech jeho rolí
- Fungování vztahů s organizací a okolím týmu
- Zefektivnění celkového fungování týmu

4.1.3.3 **Denní Stand-up**

Během Stand-upu se mají členové týmu vzájemně informovat o progresu jednotlivých uživatelských příběhů. Schůzka sleduje aktuální stav Iterace a případné problémy. Tým na Stand-upu zajišťuje synchronizaci svých aktivit a ověřuje, zda některý z členů týmu nepotřebuje pomoc ostatních. Všichni členové týmu si navzájem kladou následující otázky:

- Co jsem dělal/a včera?
- Co budu dělat dnes?
- Jsou nějaké překážky mé práce?

4.1.3.4 **Backlog Refinement**

Hlavním cílem Refinementu je pravidelná a průběžná aktualizace Backlogu a zároveň příprava přehledných a řádně popsaných uživatelských příběhů pro nadcházející iteraci tak, aby jim všichni členové Agilního týmu dobře rozuměli. Schůzka je dle potřeby opakována, dokud nejsou potřebné položky Backlogu dostatečně připraveny pro nadcházející schůzku Iteration Planning. Nejedná se tedy o jednorázovou událost, nýbrž o kontinuální proces pomáhající minimalizovat následné problémy vzniklé nepřesným odhadem, popisem, či vzájemným nepochopením zúčastněných stran. Složitější Uživatelské příběhy je tak vhodné diskutovat opakovaně předtím, než se tým zaváže k jejich splnění.

4.1.3.5 **Iteration Planning**

Hlavním cílem ceremonie je naplánování nadcházejícího sprintu na základě vstupů z Backlog Refinementu. Během schůzky agilní tým vytváří realistický plán iterace podle aktuálních týmových kapacit. Celý tým tak společně plánuje, kolik položek týmového Backlogu se zaváže dodat během nadcházející iterace. V rámci tohoto procesu je zohledněna složitost User Stories, jejich velikost a závislost na jiných týmech. Celou ceremonii organizuje agilní tým a při plánování vychází ze zkušeností načerpaných během předchozích

iterací. Zároveň zohledňuje zpětnou vazbu získanou od stakeholderů při ceremonii Iteration Review a podněty od ostatních týmů, se kterými spolupracuje.

4.1.3.6 **System Demo**

System Demo je ceremonie, na které Product Owners agilních týmů prezentují nové funkce. Ten následně prezentované dodávky akceptuje či vrací k dopracování. Cílem je získat zpětnou vazbu od Business Owners, sponzorů, zákazníků a dalších klíčových ART stakeholderů, kteří jsou na ceremonii přizváni. Díky tomu mohou agilní týmy včas a kontinuálně ověřit, zda se jejich práce neodchyluje od očekávání zákazníků. Ceremonie také umožňuje Product Managerovi ověřit, že vylepšení naplánované v Programovém Boardu byly skutečně dodány.

4.2 Interní strategie testování vybrané bankovní společnosti

Testování je životní cyklus plánování, přípravy, implementace a vyhodnocení, který probíhá s životním cyklem vývoje softwaru a ukazuje rozdíl mezi požadovaným a reálným stavem. Základním dokumentem stojícím na vrcholu hierarchie testovací dokumentace společnosti je dokument nazvaný Test Policy, který v následujících bodech popisuje Mise testování:

- Poskytnout reálný pohled na kvalitu SW produktů našim zákazníkům
- Podílet se na zlepšení kvality SW produktů včasnou identifikací defektů v rámci životního cyklu vývoje SW, a aktivně spolupracovat s vývojovými týmy na jejich vyřešení
- Přispět ke spokojenosti našich zákazníků a zvýšení jejich důvěry v SW produkty
- Testování je zapojeno do raných fází end-to-end procesu dodání
- Testovat nezávisle s vysokou efektivitou a účinností vzhledem k využitým zdrojům
- Budovat silné interní know-how klíčových bankovních systémů, stabilní a kvalitní testovací tým
- Používat jednotnou strategii testování a metodiku napříč projekty s možnosti řízených a kontrolovatelných výjimek

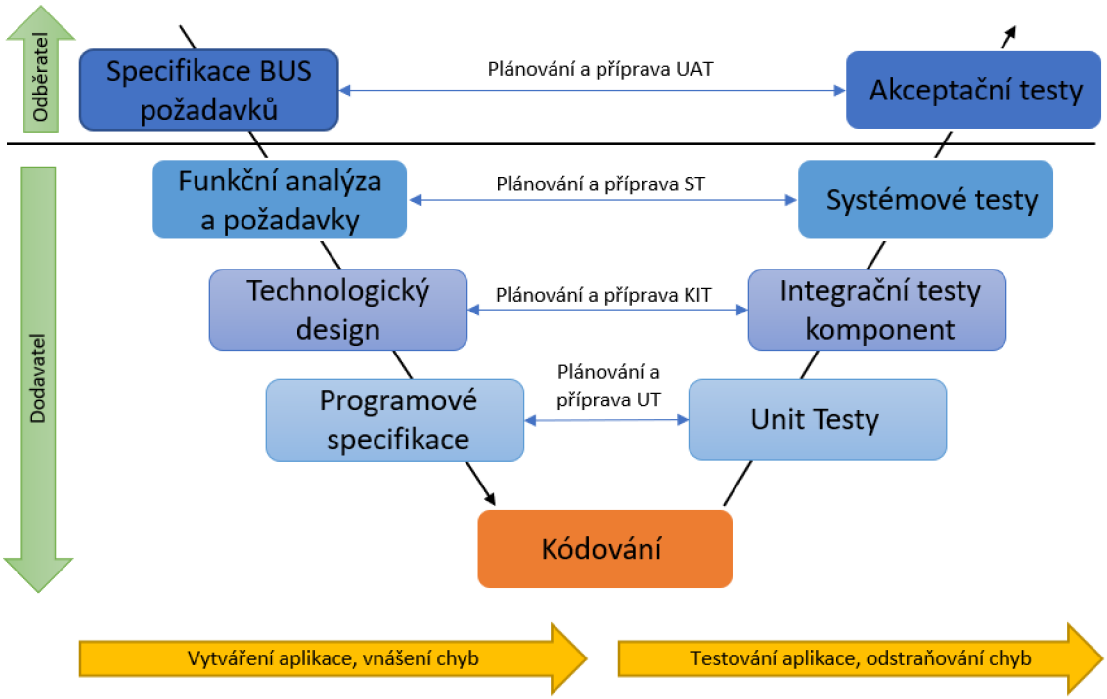
- Vyhodnocovat kvalitu a efektivitu testování a průběžně pracovat na zlepšování

V rámci metodiky SAFe uplatňované ve vybrané společnosti se nevytváří testovací strategie pro konkrétní projekty. Testovací strategie jednotlivých projektů je tak řízena pouze obecně platnou testovací strategií. Ta je určena všem, kdo se podílí na realizaci projektů v rámci společnosti a měli by tak s ní být seznámeni. Testovací strategie, a z ní plynoucí aktivity, však nejsou samoúčelné. Smyslem testování je podpořit zkvalitnění a úspěšné dodání projektu, nikoliv zbytečné zdržování a odvádění pozornosti a zdrojů od dalších důležitých součástí projektu.

4.2.1 Testovací úrovně

V rámci strategie testování ve vybrané společnosti rozlišujeme několik základních úrovní testů. Délka jednotlivých úrovní není pevně dána a může se lišit dle konkrétních potřeb projektu. Úrovně testování vychází ze sekvenčního „V-modelu“, který odpovídá zavedenému procesu vývoje.

Obrázek 6: V-model



Zdroj: Vlastní zpracování

Unit (komponentní) testy (UT)

Jedná se o nejnižší úroveň testování, během níž dochází k verifikaci fungování vyvinutých softwarových komponent, které jsou samostatně testovatelné. Plánování a řízení testů zajišťuje vedoucí vývojového týmu, který tento plán konzultuje se Scrum Manažerem. Analýza i návrh testovacích případů jsou následně plně v kompetenci vývojových týmů jednotlivých komponent. Ti také provádí implementaci a vykonání testů.

Integrační testy komponent (KIT)

Tato úroveň testování je zaměřena na integraci mezi komponentami. Ověřuje rozhraní mezi komponentami a interakce s různými částmi systému. Hlavními cíli této fáze jsou příprava integrovaných komponent na systémové testy a odhalení defektů nalezených v rámci realizace projektu, což představuje kvalitu dodávky do systémových testů. Za testování v rámci KIT odpovídá oddělení testování, přičemž hlavní zodpovědnou osobou je Test Manažer. Testování zpravidla probíhá v jednom kontinuálním cyklu a integrace je realizována po samostatně integrovatelných funkcionalitách. Přípravu testovacích případů a dat má na starost oddělení testování a vývoje, přičemž jsou sdíleny znalosti o scénářích v technickém testu.

Systémové testy (SIT)

Úroveň systémových testů je zaměřena na testování systému jako celku. Testují se business funkcionality a celkové end-to-end procesy. Hlavními cíli této fáze jsou ověření připravenosti pro akceptační testy a odhalení defektů. Odpovědnost za testování je na oddělení testování a vývoje, konkrétní zodpovědnou osobou je Scrum Manažer. Všechny navržené testy mají stanoveny priority a je definována sada scénářů.

Akceptační testy (UAT)

Během akceptačních testů se ověřuje, že systém splňuje všechny funkční i nefunkční požadavky a je připraven pro nasazení do produkce, přičemž hledání defektů není primárním účelem této fáze testování. Plánování a odpovědnost za provedení testů má Business (BUS), přičemž testy mohou probíhat i v několika cyklech a jejich plán by měl umožnit začít UAT i v případě, že SIT ještě nejsou kompletně dokončeny.

4.2.2 Typy testů

Funkční testy – Testují, zda software naplňuje specifikované funkční požadavky byznysu a podporuje jeho procesy

Security testy – Zaměřují se na zabezpečení testovaného systému a jeho dat, např ověřují, zda je systém přístupný pouze určeným uživatelům

Data testy – Testují datovou integritu a kontrolují datový model

Free testy – Funkční testy prováděné z aplikačního front-endu osobou s dobrou znalostí testované oblasti. Jsou doplňkem ke skriptovým testerům

Instalační testy – Testují, jak probíhá instalace a odinstalace produktu na daném prostředí. Po instalaci se ověřuje, jestli správně pracuje v prostředí s ostatními aplikacemi, nedochází ke konfliktu s nimi a je nainstalována správná verze aplikace

Performance testy – Testují výkonnost systému a výkonnost pod zátěží. Do této skupiny patří testy přetížení a stability systému

Usability testy – Testují snadnost ovládání aplikace pro uživatele. Zaměřují se tak především na testování uživatelského rozhraní.

Disaster Recovery – Testují chování systému během výpadku a schopnost obnovy funkčního stavu systému. Do této kategorie testů spadá i testováni backup & restore.

Regresní testy – Opakovaně testují již testovaný software s cílem najít všechny defekty, které byly zaneseny jako následek změn

Smoke testy – Krátký test nebo sada testů, které ověřují způsobilost softwaru pro další testování nebo fungování v produkci.

Následující matice testů uvádí přehled testů typicky prováděných na jednotlivých testovacích úrovních

Tabulka 2: Matice typů testů

Typ testů		**Úroveň testů**			
		UT	KIT	SIT	UAT
Funkční testy	Funkční testy	x	x	x	x
	Security testy			x	x
	Data testy	x	x	x	x
	Free testy			x	x
	Instalační testy			x	x
Nefunkční testy	Performance testy	x		x	x
	Usability testy			x	x
	Disaster recovery			x	x
Regresní testy	Smoke testy	x	x	x	x

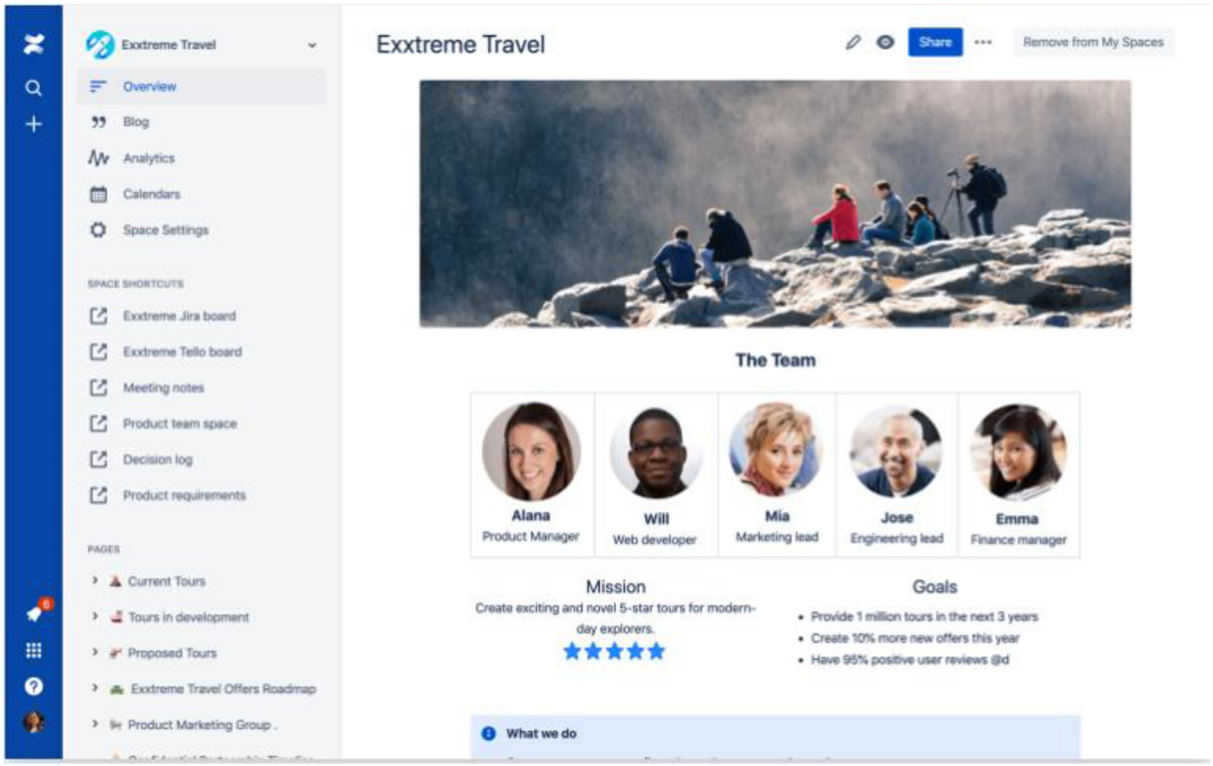
Zdroj: Vlastní zpracování

4.2.3 **Programy pro podporu agility**

4.2.3.1 **Confluence**

Confluence je softwarový cloudový nástroj vytvořený společností Atlassian, který slouží jako úložiště dokumentů a databáze knowhow společnosti. Zároveň usnadňuje spolupráci v rámci týmu a napomáhá online spolupráci. Dynamické stránky nabízí týmu možnost strukturovat, organizovat a sdílet práci, kterou si může kterýkoliv člen týmu kdykoliv zobrazit.

Obrázek 7: Příklad náhledu aplikace Confluence

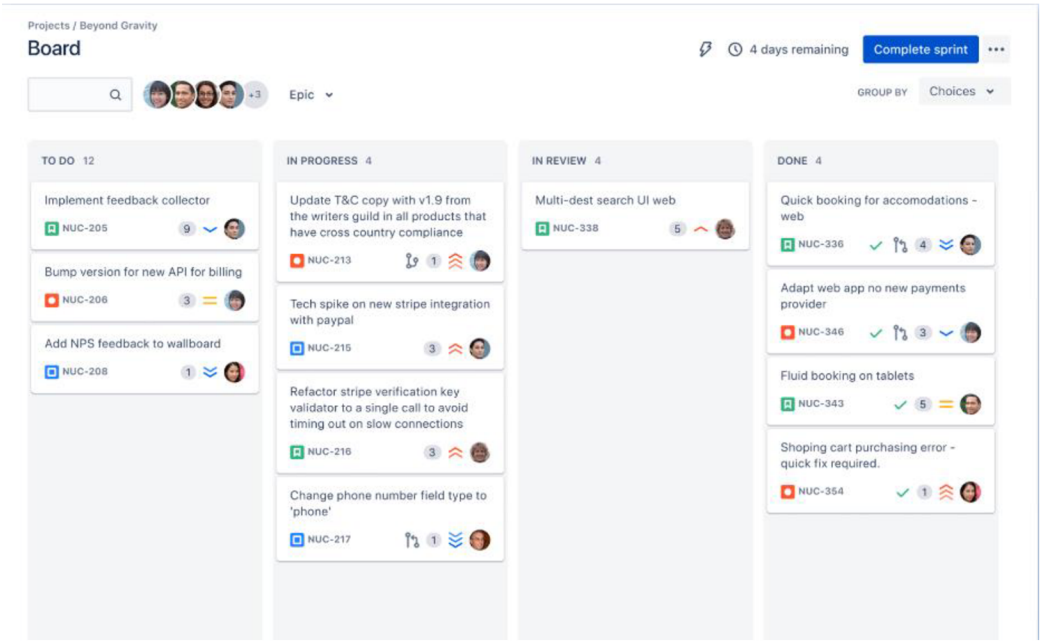


Zdroj: (Attlassian.com, 2022)

4.2.3.2 **JIRA**

JIRA je dalším softwarovým nástrojem od společnosti Atlassian. V současné době se jedná o jeden ze společnostmi nejvyužívanějších nástrojů pro podporu vedení projektů. Software umožňuje správu úkolů v rámci projektů a sledování chyb. Při zakládání nového projektu lze v JIRA vybrat z více předpřipravených šablon dle preferované metodiky agilního vývoje. Na výběr je šablona SCRUM, kterou společnost využívá u většiny projektů, umožňující pravidelný reporting a napomáhající k odevzdání práce dle pravidelného plánu.

Obrázek 8: Příklad náhledu aplikace JIRA



Zdroj: (Attlassian.com, 2022)

4.2.3.2.1 Zephyr scale for JIRA

Zephyr scale test management for JIRA je doplněk do aplikace JIRA, který nabízí lépe strukturované způsoby plánování, a testování uvnitř projektu. Poskytuje širokou knihovnu testů jak pro agilní metody, tak i pro klasický vodopádový vývoj. Je vhodný zejména pro týmy, které mají problém s optimalizací procesů, obecně napomáhá s prací týmu a pozitivně tak přispívá k zlepšení kvality dodávaného softwaru.

4.3 Návrh testování

V předchozích kapitolách práce byla shrnuta základní teorie vývoje software dle agilních metodik. V této kapitole je definován vzorový projekt, na základě kterého, je následně navržen soubor testů, který by byl využit při realizaci tohoto konkrétního projektu. Projekt je návrhem úpravy stávajícího systému společnosti, který by jejím zaměstnancům a klientům přinesl úsporu času a zjednodušení konkrétního procesu.

4.3.1 Uživatelský příběh

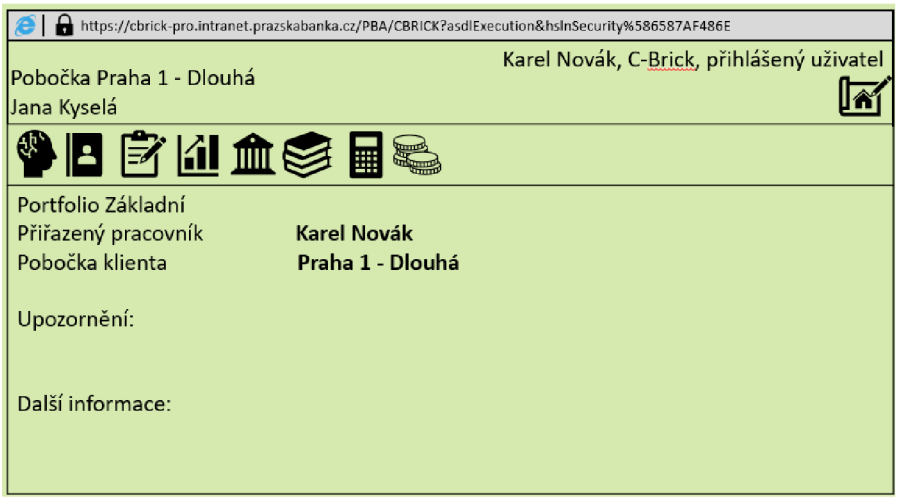
Úkolem projektu je přidání nové funkcionality do již existující aplikace C-Brick. Jedná se o jednu ze zásadních aplikací pro chod celé banky, jelikož právě v ní zaměstnanci banky spravují osobní údaje a účty klientů. V současné době, pokud si pracovník pobočky chce zobrazit transakční historii běžného účtu klienta, musí se přihlásit do jiné aplikace, kde si konkrétní účet klienta vyhledá a zobrazí transakční historii. Soubor testů je navržen pro proces vývoje nového tlačítka, které by umožnilo zobrazit transakční historii klienta za období 2 let přímo v aplikaci C-Brick, kde má pracovník klienta identifikovaného a může nahlížet na jeho produkty.

Zadavatelem tohoto projektu je fiktivní společnost Pražská banka, a.s. působící na trhu v České republice a zaměstnanci jejích retailových poboček. Společnost disponuje vlastními vývojovými týmy, řešitelem tak budou vlastní zaměstnanci pracující ve vývojářských týmech, konkrétně se jedná o ART C-Brick.

4.3.2 Definice požadavků a návrh arhitektury

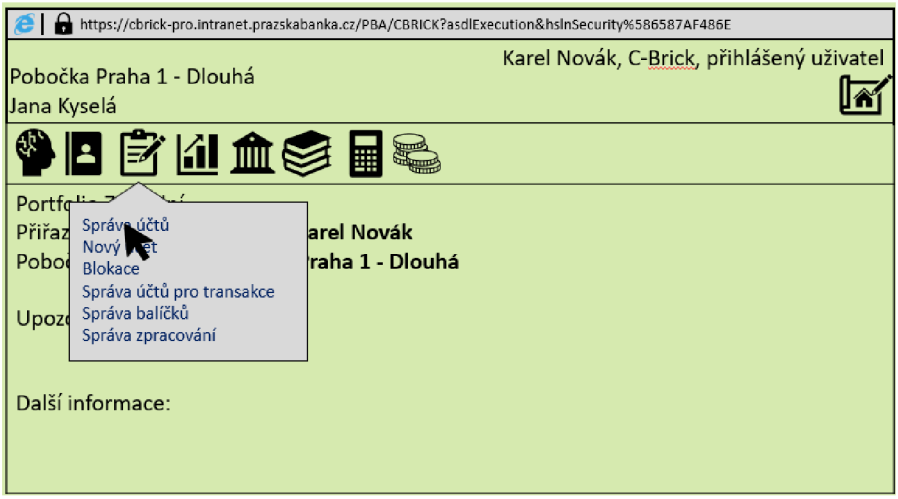
Požadavkem zákazníka je přidání tlačítka „Historie účtu“, po jehož zmáčknutí se pracovníkovi v novém okně zobrazí historie vybraného běžného či spořícího účtu. Na následujících obrázcích je stávající rozhraní aplikace C-Brick před a očekávané rozhraní po nasazení nové funkcionality:

Obrázek 9: Rozhraní aplikace C-Brick



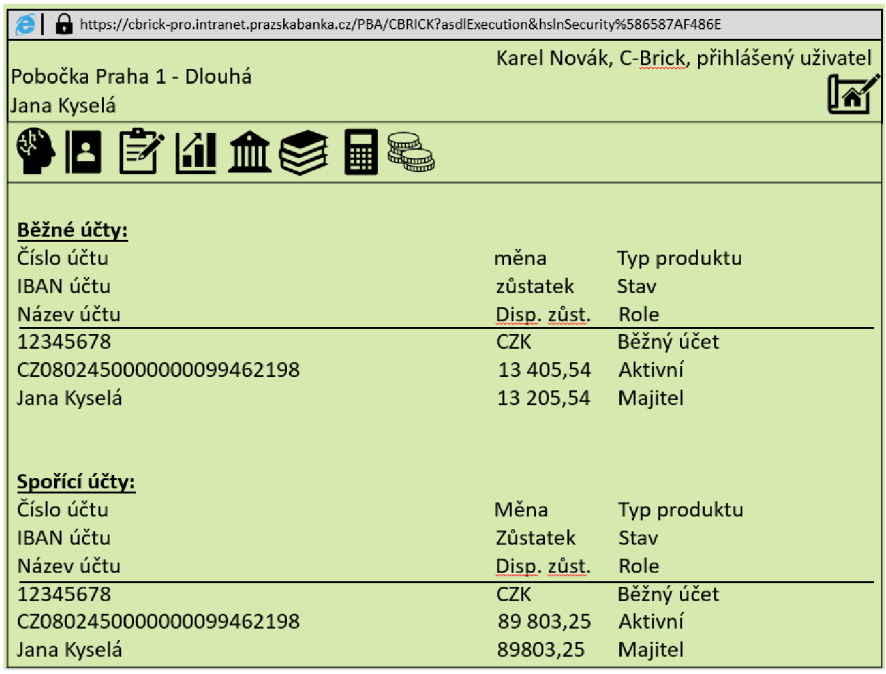
Zdroj: Vlastní zpracování dle aplikace vybrané banky

Obrázek 10: Zobrazení účtů klienta



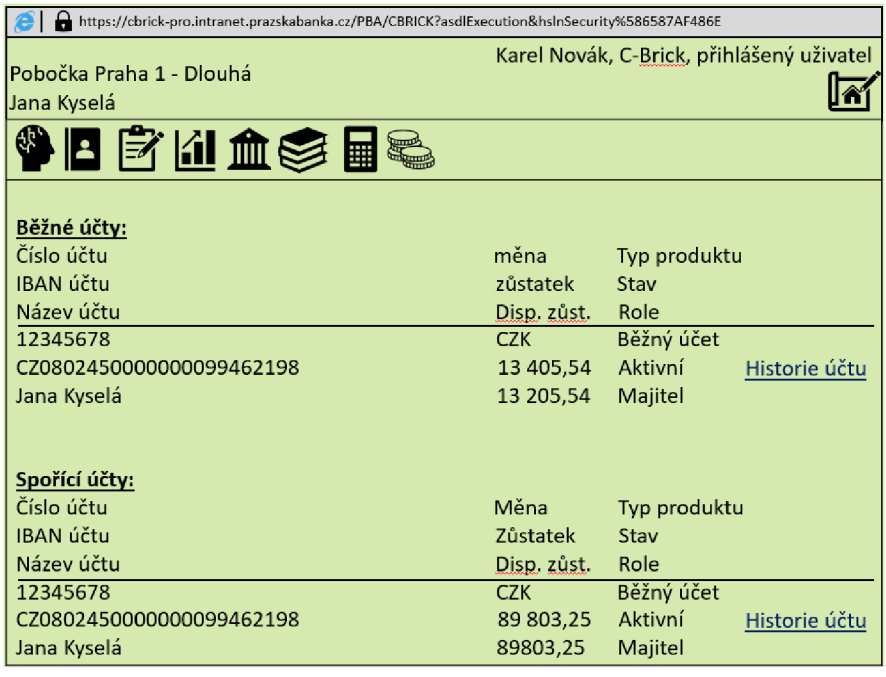
Zdroj: Vlastní zpracování dle aplikace vybrané banky

Obrázek 11: Účty klienta před nasazením změny



Zdroj: Vlastní zpracování dle aplikace vybrané banky

Obrázek 12: Účty klienta po nasazení nové funkce do produkce

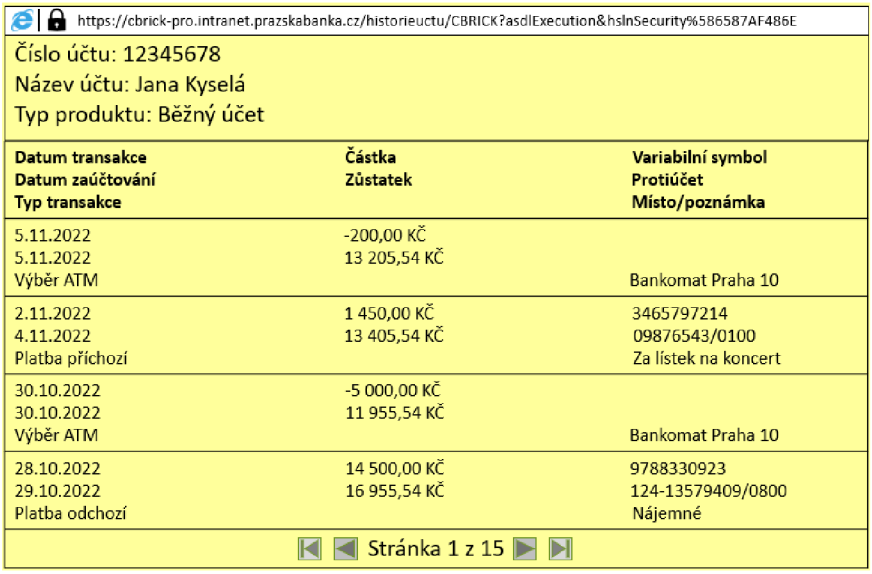


Zdroj: Vlastní zpracování dle aplikace vybrané banky

Po rozkliknutí nově přidaného tlačítka „historie účtu“ se objeví nové okno. V načteném okně je zobrazena historie účtu od nejnovějších plateb po nejstarší, a to za

období 2 let. U každé platby by mělo být uvedeno datum provedení, datum zaúčtování, typ transakce, částka, měna, variabilní symbol, místo transakce, protiúčet a konečný zůstatek účtu po zaúčtování platby. V hlavičce by dále mělo být uvedeno jméno a příjmení klienta, typ účtu a číslo tohoto účtu.

Obrázek 13: Transakční historie klienta



Zdroj: Vlastní zpracování dle aplikace vybrané banky

4.3.3 **PI Planning**

Základní agilní ceremonií je schůzka PI Planning, na které se sejdou všichni členové agilních týmů, Team of Teams a ostatní klíčoví stakeholdeři. Společně proberou akceptační kritéria dodaná zákazníkem, který je zastoupen Product Ownerem. Na schůzce jsou naplánovány jednotlivé uživatelské scénáře do všech Iterací a identifikovány závislosti a rizika. Na schůzce je domluveno, že nové tlačítko bude dostupné pouze pro pracovníky poboček, kvůli ušetření času klienta. Ostatní zaměstnanci banky musí historii účtu zobrazovat standartně v účetním systému. Odpovědná osoba za testování, tedy Testovací manažer, na základě metodiky testování vytvoří testovací plán, domluví rozsah potřebných testů a akceptační kritéria pro jednotlivé úrovně testů. Dle jednotlivých úrovní testů jsou navrženy testovací scénáře, které jsou předány testerům. Zároveň jsou domluveny typy prostředí pro konkrétní testování před nasazením do produkce.

Definice testovacích scénářů

Testovací scénář 1: Kontrola nového tlačítka

- Zobrazení tlačítka u všech typů variant běžných účtů
- Zobrazení tlačítka u spořícího účtu
- Zobrazení tlačítka pro základní uživatelské role
- Chování aplikace při neidentifikování klienta pracovníkem

Testovací scénář 2: Kontrola výstupu

- Zobrazení historie ke správnému číslu účtu
- Klient bez transakční historie na účtu
- Transakce a čísla jsou správně odřádkovány a zobrazují se správné hodnoty
- Transakce se řadí ve správném pořadí dle data zaúčtování

4.3.4 Identifikace úrovní a psaní testů

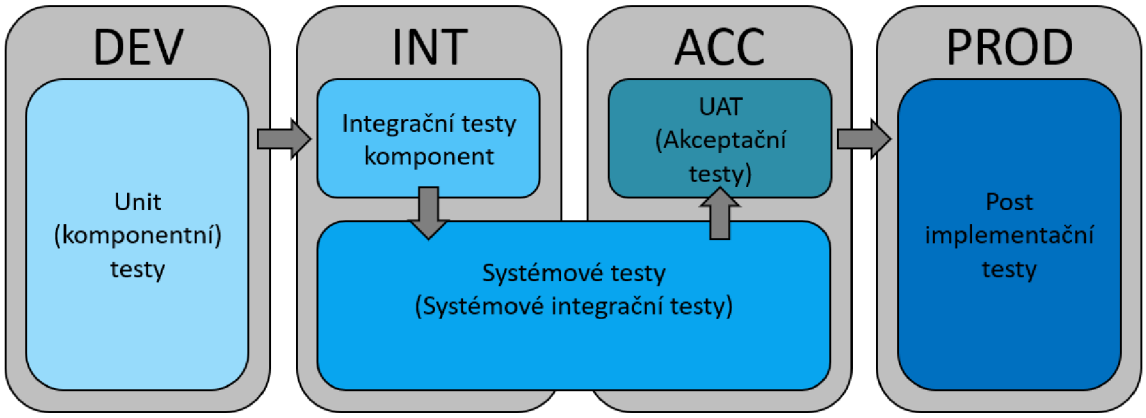
Na základě testovacích scénářů navrženého projektu jsou napsány testovací skripty pro jednotlivé úrovně testů, které mají za úkol ověřit kvalitu dodávaného softwaru během jeho vývoje a při jeho dodání. Testy jsou vytvořeny dle připravené matice úrovní a typů testů. V praxi by pak exekuce jednotlivých testů probíhala odlišně dle dané úrovně. Vybraná společnost má sice nastaveny podmínky agilního vývoje, nicméně i přesto je implementace agility velmi náročná a testování v celém procesu vývoje software není pouze agilní. Zejména se jedná o poslední úroveň uživatelské akceptační testy. Pro tuto úroveň jsou napsány všechny testovací scénáře a testy dopředu pro celý vývoj a ve fázi exekuce testů se již nemění ani nedopisují. Samotné testování pak probíhá na konkrétních prostředích dle přiloženého grafu.

Tabulka 3: Matice testů daného návrhu vylepšení

Typ testů		Úroveň testů			
		UT	KIT	SIT	UAT
Funkční testy	Funkční testy	x	x	x	x
	Security testy				x
	Data testy				x
	Instalační testy	x	x	x	x
Nefunkční testy	Performance testy				
	Usability testy			x	x
	Disaster recovery				
Regresní testy	Regresní testy			x	x
Smoke testy	Smoke testy			x	x

Zdroj: Vlastní zpracování

Obrázek 14: Testovací prostředí



Zdroj: Vlastní zpracování

DEV = Developerské prostředí
INT = Integrační prostředí
ACC = Akceptační prostředí
PROD = Produkční prostředí

4.3.4.1 **Unit testy**

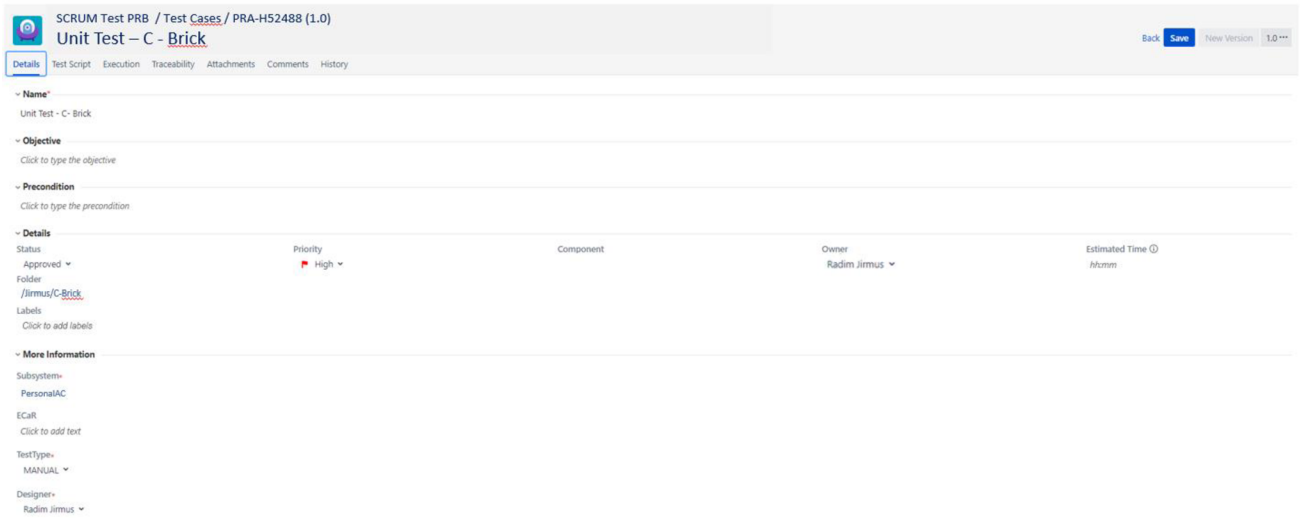
Nejnižší úrovní testovaní ve vývojovém procesu jsou Unit testy. Testy probíhají v developerském prostředí lokálně na počítači daného vývojáře a vytvářejí je sami vývojáři jednotlivých komponent. Předpokladem pro úspěšnou exekuci těchto testů je přihlášení k bankovním systémům v interní síti, případně vzdáleně pomocí VPN. Unit testů je v průběhu první fáze vývoje opravdu mnoho, proto je pro potřeby této práce zmíněn pouze jeden příklad takovéhoto testu.

Test č.1:

Name (Jméno)	UNIT Test – C- Brick
Objective (Cíl testu)	Ověření spuštění aplikace
Precondition (Předpoklady)	- Uživatel přihlášený na interní síti nebo VPN - Developerské prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má běžný účet Pražské konto
Defect priority (Priorita defektu)	Blocker
Test Type (Typ testu)	Manuální nebo automatizovaný test
Test name (Jméno testu)	Spuštění aplikace
Test Data (Data testu)	Bez dat
Test Script (Skript testu) 1	Otevři/Spusť verzi aplikace C-Brick 1.0
Expected Result (Očekávaný výsledek) 1	Aplikace se spustila bez chybových hlášek
Test name	Kontrola logu aplikace
Test Data	Bez dat
Test Script 2	Zkontroluj log aplikace
Expected Result 2	Není vidět žádný error

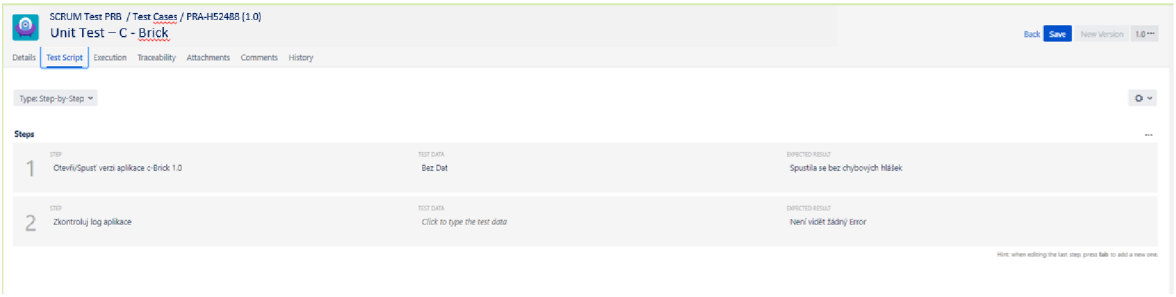
Následuje příklad zápisu testu v JIRA. Stejným způsobem se pak zapisují i ostatní testy pro další testovací úrovně.

Obrázek 15: Unit test - C-Brick



Zdroj: Vlastní zpracování

Obrázek 16: Unit Test - C-Brick Script



Zdroj: Vlastní zpracování

4.3.4.2 **Integrační testy komponent**

Na této úrovni dochází k testování integrace samotného tlačítka do aplikace C-Brick. Je potřeba otestovat zobrazení nového tlačítka u různých typů účtů a také pro různé uživatelské role.

Test č.2:

Name	KIT Test – C - Brick – Běžný účet Pražské konto
Objective	Kontrola zobrazení nového tlačítka u běžného účtu Pražské konto
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má běžný účet Pražské konto
Defect priority	Major
Test Type	Manuální
Test name	Tlačítko u Pražského konta
Test Data	Bez dat
Test Script	Otevři účty klienta v aplikaci C - Brick
Expected Result	U běžného účtu je zobrazeno tlačítko pro zobrazení historie

Test č.3:

Name	KIT Test – C- Brick – Běžný účet VIP konto
Objective	Kontrola zobrazení nového tlačítka u běžného účtu VIP konto
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má běžný účet VIP konto
Defect priority	Major
Test Type	Manuální
Test name	Tlačítko u VIP konta
Test Data	Bez dat
Test Script	Otevři účty klienta v aplikaci C - Brick
Expected Result	U běžného účtu je zobrazeno tlačítko pro zobrazení historie

Test č.4:

Name	KIT Test – C- Brick – Spořící účet
Objective	Kontrola zobrazení nového tlačítka u spořícího účtu
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má Spořící účet
Defect priority	Major

Test Type	Manuální
Test name	Tlačítko u Spořícího účtu
Test Data	Bez dat
Test Script	Otevři účty klienta v aplikaci C - Brick
Expected Result	U Spořícího účtu je zobrazeno tlačítko pro zobrazení historie

Test č.5:

Name	KIT Test – C- Brick – Role pracovníka C-Brick pokladna
Objective	Kontrola nezobrazení nového tlačítka u všech typů účtů
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pokladna - Klient má spořící a běžný účet
Defect priority	Minor
Test Type	Manuální
Test name	Pokladna bez zobrazení historie
Test Data	Bez dat
Test Script	Otevři účty klienta v aplikaci C - Brick
Expected Result	U žádného z klientových účtů není zobrazeno tlačítko historie účtu

Test č.6:

Name	KIT Test – C- Brick – Role pracovníka C-Brick Správa účtů
Objective	Kontrola nezobrazení nového tlačítka u všech typů účtů
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem správa účtů - Klient má spořící a běžný účet
Defect priority	Minor
Test Type	Manuální
Test name	Oddělení Správy účtů bez zobrazení historie
Test Data	Bez dat
Test Script	Otevři účty klienta v aplikaci C - Brick
Expected Result	U žádného z klientových účtů není zobrazeno tlačítko historie účtu

Test č.7:

Name	KIT Test – C- Brick – Zobrazení nového okna při kliknutí
Objective	Kontrola zobrazení nového okna po stisku nového tlačítka
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má spořící a běžný účet
Defect priority	High
Test Type	Manuální
Test name	Zobrazení nového okna po stisku tlačítka
Test Data	Bez dat
Test Script	- Otevři účty klienta v aplikaci C – Brick - Klikni na „historie účtu“ u běžného nebo spořícího účtu klienta
Expected Result	Objeví se nové okno

Test č.8:

Name	KIT Test – C- Brick – Neidentifikovaný klient
Objective	Kontrola nezobrazení nového tlačítka při neidentifikovaném klientovi
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient není identifikován
Defect priority	High
Test Type	Manuální
Test name	Neidentifikovaný klient
Test Data	Bez dat
Test Script	Otevři účty klienta v aplikaci C – Brick
Expected Result	Není zobrazeno tlačítko historie účtu

4.3.4.3 **Systémové integrační testy**

Systémové integrační testy mají za úkol ověřit načítání a propisování dat do databází a rejstříků. Následně také ověřují vrácená data a informace.

Test č.9:

Name	SIT Test – C- Brick – Načtení transakcí účtu

Objective	Kontrola načtení transakční historie
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má Spořící nebo běžný účet
Defect priority	Critical
Test Type	Manuální
Test name	Načtení detailu transakcí účtu
Test Data	Historie transakcí daného účtu
Test Script	- Otevři účty klienta v aplikaci C – Brick - Klikni na „historie účtu“ u běžného nebo spořícího účtu klienta
Expected Result	Objeví se nové okno s transakční historií klienta

Test č.10:

Name	SIT Test – C- Brick – Zobrazení správného čísla účtu
Objective	Kontrola zobrazení správného čísla účtu v nově otevřeném okně
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má Spořící nebo běžný účet
Defect priority	High
Test Type	Manuální
Test name	Zobrazení čísla účtu u transakční historie
Test Data	Historie transakcí daného účtu
Test Script	- Otevři účty klienta v aplikaci C – Brick - Klikni na „historie účtu“ u běžného nebo spořícího účtu klienta
Expected Result	Objeví se nové okno se správným číslem účtu v hlavičce

Test č.11:

Name	SIT Test – C- Brick – Zobrazení transakcí dle data zaúčtování
Objective	Kontrola řazení transakcí na účtu klienta
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má Spořící nebo běžný účet
Defect priority	Major
Test Type	Manuální

Test name	Řazení transakcí dle data zaúčtování
Test Data	Historie transakcí daného účtu
Test Script	- Otevři účty klienta v aplikaci C – Brick - Klikni na „historie účtu“ u běžného nebo spořícího účtu klienta
Expected Result	Objeví se nové okno, kde jsou transakce seřazeny dle data zaúčtování

Test č.12:

Name	SIT Test – C- Brick – Kontrola transakcí
Objective	Kontrola zobrazených transakcí klienta
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má Spořící nebo běžný účet
Defect priority	Critical
Test Type	Manuální
Test name	Správnost transakcí klienta
Test Data	Historie transakcí daného účtu
Test Script	- Otevři účty klienta v aplikaci C – Brick - Klikni na „historie účtu“ u běžného nebo spořícího účtu klienta
Expected Result	Objeví se nové okno se zobrazenými transakcemi klienta, které odpovídají transakcím klienta ze systému PersonalAC

Test č.13:

Name	SIT Test – C- Brick – Klient bez transakční historie (např nový účet)
Objective	Kontrola zobrazení chybové hlášky
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má Spořící nebo běžný účet
Defect priority	Minor
Test Type	Manuální
Test name	Zobrazení chybové hlášky pro klienta bez historie transakcí
Test Data	Historie transakcí daného účtu
Test Script	- Otevři účty klienta v aplikaci C – Brick - Klikni na „historie účtu“ u běžného nebo spořícího účtu klienta
Expected Result	Objeví se nové okno s chybovou hláškou „klient bez transakční historie“

4.3.4.4 **Uživatelské akceptační testy**

Uživatelské akceptační testy jsou nejvyšší úrovní testování, na níž dochází k ověření, zda je systém připraven k nasazení do produkce. Z praxe vybrané banky se způsob psaní těchto testovacích scénářů a testů liší od předchozích úrovní. Zatímco v předchozích úrovních jsou testovací scénáře psány a upravovány každých 14 dní dle agilní metodiky, psaní scénářů pro tuto testovací úroveň je řízeno vývojem dle Waterfall. Jsou tak vytvořeny testovací scénáře na celé 3měsíční období testování a po spuštění exekuce testů se jednotlivé scénáře příliš nemění a nejsou ani dopisovány nové. Vzhledem k tomu, že je tato práce zaměřena na agilní vývoj, uvádím tak pouze pro příklad jeden takový test, na jehož příkladu je demonstrováno, jakým způsobem by byly tyto testy zapsány.

Test č.14:

Name	UAT Test – C- Brick – Zobrazení transakční historie u Běžného účtu Pražské konto
Objective	Kontrola zobrazení transakční historie klienta u Pražského konta
Precondition	- Uživatel přihlášený na interní síti - Integrační prostředí - Uživatel přihlášen pod C–Brick uživatelem pobočka - Klient má Pražské konto s transakční historií
Defect priority	Critical
Test Type	Manuální
Test name	Zobrazení transakcí klienta pro Pražské konto
Test Data	Historie transakcí daného účtu
Test Script	- Otevři aplikaci KLM - Identifikuj klienta do systému - Otevři aplikace C – Brick proklikem z identifikovaného klienta v KLM - Klikni na ikonu účtů klienta - Klikni na tlačítko Správa účtů - Klikni na tlačítko Historie účtu - Zkontroluj načtená data v novém okně
Expected Result	Objeví se nové okno se správnými daty o účtu klienta a jeho historii

4.3.4.5 **Exekuce testů**

Po návrhu dané funkcionality a jejím vývoji přichází na řadu exekuce navržených testů. Zápis výsledků testů a případných defektů je evidován v JIRA, kde jsou jednotlivé

testy navázány k testovacím scénářům a úrovním. Pro postup do další úrovně vývoje a testování jsou vždy určena akceptační kritéria pro různé priority defektu. Tato kritéria jsou určována Test manažerem pro každou úroveň zvlášť a bez splnění těchto požadavků nelze postoupit na další úroveň testů. Následující tabulka zobrazuje základní priority defektů a jejich stručnou charakteristiku, dle které jsou priority přiřazeny jednotlivým testům.

Tabulka 4: Priority defektů

Priorita defektu	Popis
Blocker	Nejzávažnější chyba blokující pokračování v testování Chyba na klíčové funkcionalitě Není možné začít exekuci testovacích scénářů
Critical	Závažná chyba ovlivňující chování aplikace Není možné dokončit exekuci více testovacích scénářů
Major	Chyba ovlivňující základní funkcionalitu systému Není možné dokončit exekuci několika testovacích scénářů
High	Chyba ovlivňující funkcionalitu systému Není blokována exekuce dalších testovacích scénářů
Minor	Chyba bez dopadu na funkčnost systému Pouze mírné odlišnosti ve vzhledu či chování aplikace

Zdroj: Vlastní zpracování

5 Závěr

Bakalářská práce se zaměřuje na problematiku vývoje informačních systémů. Vymezuje základní tradiční a agilní metodiky vývoje software a jejich náležitosti. Cílem práce je vymezení základních metodik vývoje software a popsání agilní metodiky vývoje software vybrané společnosti. Na základě takto zpracovaných metodik je pak stanoven hlavní cíl, který spočívá v navržení přidání nové funkcionality do již existující a používané aplikace a navržení testů pro ověření kvality dodávaného vylepšení.

V teoretické části práce jsou definovány pojmy informační systém a vývoj informačních systémů, se kterými práce dále pracuje. Dále je popsána problematika vývoje informačních systémů z hlediska historie. Jsou definovány základní tradiční metodiky vývoje software a jejich výhody a nevýhody. Hlavní důraz je následně kladen na popsání agilních metodik, které jsou znalostním podkladem pro následné zpracování teoretické části.

Vlastní práce by se dala rozdělit na 2 tematicky oddělené části. První část praktické části práce se věnuje uplatnění agilních metodik v praxi. K tomu byla vybrána jedna z bank působících na trhu v České republice. Na základě interních pravidel agilního vývoje dostupných na intranetu společnosti, znalostí zaměstnanců a obecně platných pravidel je definována agilní metodika Banky. Za využití stejných zdrojů je také zpracována interní metodika testování software vybrané banky. Dle takto zpracované metodiky je následně navržen projekt a příslušné testy potřebné pro jeho spuštění do provozu.

V druhém oddílu praktické části práce je navrženo vylepšení aplikace, kterou používají zaměstnanci poboček. Zmíněné vylepšení umožňuje zaměstnancům zobrazení transakční historie účtu klienta jednodušším způsobem než doposud. Pro toto vylepšení jsou navrženy testovací scénáře a dle těchto testovacích scénářů jsou navrženy samotné testy, které by sloužily pro ověření kvality dodávaného software.

Hlavní přínos práce spočívá ve vypracovaném návrhu. Takto zpracovaný návrh vylepšení může být, pokud by se banka rozhodla projekt uskutečnit, jedním z podkladů pro případný vývoj. Zároveň může složit jako jeden teoretických podkladů pro podobné projekty. Případná realizace projektu by pak v případě nasazení do produkce měla vliv na obsluhu všech klientů banky a vzhledem k ušetřenému času klientů i pracovníků by přispívalo ke zvýšení obchodní hodnoty společnosti.

6 Seznam použitých zdrojů

Attlassian. *Confluence.* [Online] [Citace: 05. 11 2022.] https://www.atlassian.com/software/confluence.
Attlassian. *JIRA.* [Online] [Citace: 05. 11 2022.] https://www.atlassian.com/software/jira.
Attlassian. *Sprint Planning* [online]. [Citace: 07. 11 2022.]. https://www.agilealliance.org/glossary/sprint-planning/
Awad, M. A. 2005. *A Comparison between Agile and Traditional.* místo neznámé : The University of Western Australia, 2005.
Beck, Kent, a další. 2001. Agile Manifesto. *WEB Agile manifesto.* [Online] 11-13. Únor 2001. https://agilemanifesto.org.
Buchalcevová, Alena. 2005. *Metodiky vývoje a údržby informačních systémů.* Praha : Grada Publishing, a.s., 2005. 80-247-1075-7.
Crispin, Lisa and Gregory, Janet. 2009. *Agile testing.* Boston : Pearson Education, Inc., 2009. 978-0-321-53446-0.
Dean, Raymond a Drew, Jemilo. 2020. *The Scaled Agile story* [online]. [Citace: 07. 11 2022.]. https://www.scaledagile.com/about/about-us/
Despa, Mihai Liviu. 2014. Comparative study on software development methodologies. *Database Systems Journal.* 2014, 3.
Kadlec, Václav. 2004. *Agilní programování, Metodiky efektivního vývoje softwaru.* Brno :
Kitner, Radek. 2020. *Typy testování software (třídění testů)* [online]. [Citace: 07. 11 2022.]. https://kitner.cz/testovani_softwaru/typy-testovani-software-trideni-testu/ Computer Press, 2004. 80-251-0342-0.
Kumar, Manoj Lal. 2018. *Knowledge Driven Development.* Cambridge : University Press, 2018. 978-1-108-47521-1.
Leffingwell, Dean a Widrig, Don. 2010. *Agile Software Requirements: Lean Requirements Practices for Teams, Programs, and the Enterprise.* Boston : Addison-Wesley Professional, 2010. 978-0-321-63584-6.
Martin, Robert C. 2009. *Čistý kód.* [překl.] Jiří Berka. Brno : Computer Press, 2009. 978-80-251-2285-3.
Myslín, Josef. 2016. *Scrum, Průvodce agilním vývojem softwaru.* Brno : Computer Press, 2016. 978-80-251-4650-7.
Rehkopf, Max *User Stories with Examples and Template* [online]. [Citace: 07. 11 2022.]. https://www.atlassian.com/agile/project-management/user-stories
Šochová, Zuzana a Kunce, Eduard. 2019. *Agilní metody řízení projektů.* Brno : Computer Press, 2019. 978-80-251-4961-4.
Metodika vybrané banky dostupná z intranetu společnosti

7 Seznam obrázků, tabulek, grafů a zkratek

7.1 Seznam obrázků

7.2 Seznam tabulek

Odkazovaný seznam tabulek

7.3 Seznam použitých zkratek

ACC – Akceptační prostředí
ART – Agile release train
BUS – Bussiness
DEV – Developerské (Vývojářské) prostředí
E2E – end-to-end
INT – Integrační prostředí
KIT – Integrační testy komponent
PI – Program inkrement
PIP – Program inkrement planning
PO – Program
PROD – Produkční prostředí (Produkce)
RTE – Release train Engineer
SAFe – Scaled Agile Framework

SIT – Systémové testy
SW – Software
UAT – Akceptační testy
UT – Unit testy
XP – Extrémní programování