Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra systémového inženýrství

Diplomová práce

Defekt management v agilním prostředí České spořitelny, a. s.

Bc. Jan Flachs

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jan Flachs

Projektové řízení

Název práce

Defekt management v agilním prostředí České spořitelny, a. s.

Název anglicky

Defect management in the agile environment of Česká spořitelna, a. s.

Cíle práce

Hlavním cílem této diplomové práce je vytvoření nové metodiky pro práci s defekty v České spořitelně, a. s. a návrh životního cyklu defektu pro implementaci do nového softwaru Jira určenému k agilnímu řízení práce.

Dílčími cíli práce jsou: popis defektu a jeho vlastností, charakteristika agilních metodik, vytvoření analýzy pro nový defekt management a porovnání stávajícího nástroje HP ALM a nového nástroje Jira, pomocí dotazníkového šetření.

Metodika

V teoretické části práce jsou vysvětleny základní pojmy spojené s defekt managementem, jako je například význam priority a severity defektu, root cause analýza a popis životního cyklu defektu. Současně jsou zde vytyčeny základní vlastnosti stávajícího nástroje HP ALM v České spořitelně, a. s. Kapitola je doplněna o popis vlastností a funkcí budoucího jednotného nástroje Jira.

Praktická část je zaměřena na charakteristiku společnosti Česká spořitelna, a. s., důvody pro přechod defekt managementu do budoucího nástroje a je popsána nově vytvořená metodika pro defekt management. Metodika obsahuje zejména nový životní cyklus defektu s detailním popisem jednotlivých stavů a charakteristiku polí využívaných pro definici vlastností defektu. Praktická část je doplněna o dotazníkové šetření, které se týká srovnání stávajícího a budoucího nástroje z pohledu uživatelského hlediska, přívětivosti a funkčnosti při práci s defekty.

Doporučený rozsah práce

60 – 80 stran

Klíčová slova

defekt, defekt management, workflow, metriky, agile, Jira, priorita, severita

---

Doporučené zdroje informací

Agilní manifest, 2001. Agilní manifest. [Online] Available at: http://agilemanifesto.org/iso/cs/manifesto.html

Doležal, J., 2016. Projektový management: Komplexně, prakticky a podle světových standardů. První vydání editor Praha: Grada Publishing.

Kerzner, H., 2009. Project management : a systems approach to planning, scheduling, and conrolling. 10. editor New Jersey: John Wiley & Sons.

Kunce, E. & Šochová, Z., 2014. Agilní metody řízení projektů. 1. vydání editor Brno: Computer Press.

PMI, 2017. PMBOK® GUIDE. 6.vydání editor Newtown Square: Project Management Institute, Inc.

Předběžný termín obhajoby

2020/21 LS – PEF

Vedoucí práce

Ing. Jiří Fejfar, Ph.D.

Garantující pracoviště

Katedra systémového inženýrství

Elektronicky schváleno dne 16. 3. 2021

doc. Ing. Tomáš Šubrt, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 23. 3. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 23. 03. 2021

Čestné prohlášení

Prohlašuji, že svou diplomovou práci „Defekt management v agilním prostředí České spořitelny, a. s.“ jsem vypracoval samostatně pod vedením vedoucího diplomové práce, 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 28. 3. 2021 _____

Poděkování

Rád bych touto cestou poděkoval Ing. Jiřímu Fejfarovi, Ph.D. za odborné vedení a cenné rady při zpracování diplomové práce. Mé poděkování patří i kolegům ze společnosti Česká spořitelna, a. s. za cenné rady a konzultace při realizaci této práce. V neposlední řadě bych chtěl také poděkovat své rodině a blízkým přátelům za podporu, pomoc a trpělivost při mém studiu.

Defekt management v agilním prostředí České spořitelny, a. s.

Abstrakt

Zadáním diplomové práce je vytvoření metodiky pro defekt management v novém agilním nástroji Jira pro společnost Česká spořitelna, a. s.

První část práce, literární rešerše, se věnuje uvedení do problematiky defekt managementu a obecnému popisu defektu a jeho životního cyklu, který je nezbytné stanovit pro správné předávání defektu mezi jednotlivými řešiteli. Jsou zde popsány pojmy, jednotlivé stavy, které by defekt měl mít, a základy agilního řízení, stěžejní pro přizpůsobení defekt managementu agilnímu řízení. Nezbytnou součástí práce je také grafické znázornění výše zmíněných procesů, neboť je důležitým nástrojem pro pochopení uvedené problematiky.

V praktické části se obsah zabývá stručným popisem agilního řízení a nově vytvořených rolí v České spořitelně, majících vliv na defekt management. Dále je uvedena analýza, která předcházela převedení defekt managementu do nového nástroje. Pro defekt management je vytvořeno nové workflow s jednotlivými stavy a přechody.

Nedílnou součástí této části jsou také sledované metriky, vytváření defektu, testování obrazovek, které jsou důležité pro zobrazení detailu defektu nebo jeho založení, popis vzniklých polí, charakterizujících povahu defektu, dále vybrané náměty pro zlepšení uživatelského komfortu při používání defekt managementu a vyhodnocení dotazníku, jenž byl položen uživatelům.

Klíčová slova: defekt, defekt management, workflow, metriky, agile, Jira, priorita, severita

Defect Management in the agile environment of Česká spořitelna, a. s.

Abstract

The content of the thesis is the creation of a methodology for defect management in the new Jira agile tool for Česká spořitelna, a. s.

The first part, a literary research, focuses on the introduction into the issue of defect management and the general description of the defect and its life cycle, which must be determined. It describes the concepts, individual conditions that the defect should have and the basics of agile management, crucial for adapting the defect management to agile management. The essential part is also a graphical representation, which is important for understanding the above.

In the practical part, the content is oriented towards a brief description of agile management and roles in Česká spořitelna, a. s. that affect defect management. The analysis that preceded the transfer of the management defect to the new tool is listed below. The new workflow with individual states and transitions is created for defect management. An integral part of this section are also monitored metrics, creating a defect, testing screens for a defect, describing the resulting fields, selected ideas to improve user comfort when using defect management and evaluating a questionnaire that was submitted to users.

Keywords: defect, defect management, workflow, metrics, agile, Jira, priority, severity

Obsah

Seznam obrázků

Seznam tabulek

Seznam grafů

Seznam použitých zkratek

HP ALM – Hewlett-Packard Application Lifecycle Management

RCA – Root Cause Analysis

GUI – Graphical User Interface

ID – Identifikace

1 Úvod

V současné době je velmi rozšířen trend využívání agilního řízení ve firmách hlavně v oblastech projektového řízení a IT projektů. Právě nové komunikační nástroje dovolují řešitelům projektu zůstávat ve stálém kontaktu se zákazníkem a zadavatelem projektu.

Aby byl projekt dokončen zcela úspěšně a v souladu s přáním zákazníka, je důležité nepozbýt schopnost bezprostředně reagovat na změny. V roce 2001 si tuto nezbytnost uvědomilo 17 expertů, již na základě dynamického reagování na požadavky společně položili základy agilního řízení. Některé metodiky, které využívaly agilního přístupu, existovaly ještě před okamžikem sepsání Agilního manifestu. Avšak díky sepsání Agilního manifestu teprve došlo ke sjednocení základů pro tyto metodiky. Agilní manifest také položil základ pro nejznámější agilní metodiky Scrum a Kanban.

Práce se zaměřuje na přizpůsobení defekt managementu agilnímu řízení v jedné z největších bank na českém trhu – České spořitelně, a. s. Metodiku defekt managementu je důležité správně nastavit, jelikož ovlivňuje fungování celé oblasti IT, převážně pak podoblasti testování. Aby banka mohla fungovat agilně a jednotně, bylo potřeba přejít do nového sjednoceného nástroje.

Práce se nezabývá projektem přechodu defekt managementu do nového nástroje, nýbrž vytvořením nové metodiky pro použití defekt managementu v agilním prostředí a v novém nástroji. Zároveň vyhodnocení dotazníku v této práci bude podkladem pro zlepšení jednotlivých oblastí metodiky, případně pro zlepšení uživatelského komfortu při práci s defektem.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem této diplomové práce je vytvoření nové metodiky pro práci s defekty v České spořitelně, a. s. a návrh životního cyklu defektu pro implementaci do nového softwaru Jira, určenému k agilnímu řízení práce.

Dílčími cíli práce jsou: popis defektu a jeho vlastností, charakteristika agilních metodik, vytvoření analýzy pro nový defekt management a porovnání stávajícího nástroje HP ALM s novým nástrojem Jira pomocí dotazníkového šetření.

2.2 Metodika

V teoretické části práce jsou vysvětleny základní pojmy spojené s defekt managementem, jakými jsou například: význam priority a severity defektu, root cause analýza a popis životního cyklu defektu. Současně jsou zde vytyčeny základní vlastnosti stávajícího nástroje HP ALM v České spořitelně, a. s. Kapitola je doplněna o popis vlastností a funkcí budoucího jednotného nástroje Jira.

Praktická část se zaměřuje nejprve na charakteristiku společnosti Česká spořitelna, a. s., dále na důvody pro přechod defekt managementu do budoucího nástroje a je zde také popisována nově vytvořená metodika pro defekt management. Metodika obsahuje zejména nový životní cyklus defektu s detailním popisem jednotlivých stavů a charakteristiku polí využívaných pro definici vlastností defektu. Praktická část je doplněna o dotazníkové šetření, které se týká srovnání stávajícího a budoucího nástroje nejen z hlediska uživatele, ale také z hlediska přívětivosti a funkčnosti při práci s defekty.

3 Teoretická východiska

3.1 Defekt

Při testování softwaru je možné se setkat s označením chyb různých typů. Používají se termíny defekt (defect), selhání (failure), chyba (error) či závada (fault). Pro správnou definici defektu je nutné tyto pojmy přesně definovat. Tato pojetí jsou si blízká, avšak stále mezi nimi existuje určitý rozdíl.

Dodnes je možné se setkat se záměnou i v některých literárních zdrojích (Petr Roudenský, 2013). Níže je uveden popis výše zmíněných termínů:

- **Selhání (failure):** Selhání je možné vnímat jako odchylku softwaru od jeho účelu. Tohoto stavu je možné dosáhnout, pokud chybná část kódu vede ke špatnému stavu, který se dostane do výstupu. Selhání nastane tehdy, když programátor pochybí a napíše špatnou část kódu. Tester nebo programátor tento typ chyby naleznou často až při aktuálně provedeném spuštění testu a následkem je softwarové selhání. (Magalhães, 2015) (Graham, 2020)
- **Chyba (error):** Jedná se o lidskou aktivitu, jež má za následek chybný výsledek. Pro programátora se tato chyba nazývá „error“. Může nastat díky chybnému porozumění požadavkům na software, nepřesným výpočtům, mylné interpretaci hodnoty apod. (Magalhães, 2015)
- **Závada (fault):** Zřídkakdy se potkáme s překladem tohoto termínu do českého ekvivalentu „závada“. Zde je překlad uveden výhradně pro porozumění odlišnosti termínů. Užívá se pro vyjádření více chyb najednou. (Graham, 2020)
- **Defekt (defect):** Defekt je stav softwaru, který nesplňuje požadavek, jak je uveden ve specifikaci nebo očekávání koncového uživatele. Jde tedy o chybu v logice či kódu, která způsobuje selhání softwaru nebo jeho neočekávané chování. (ISTQB, 2020)

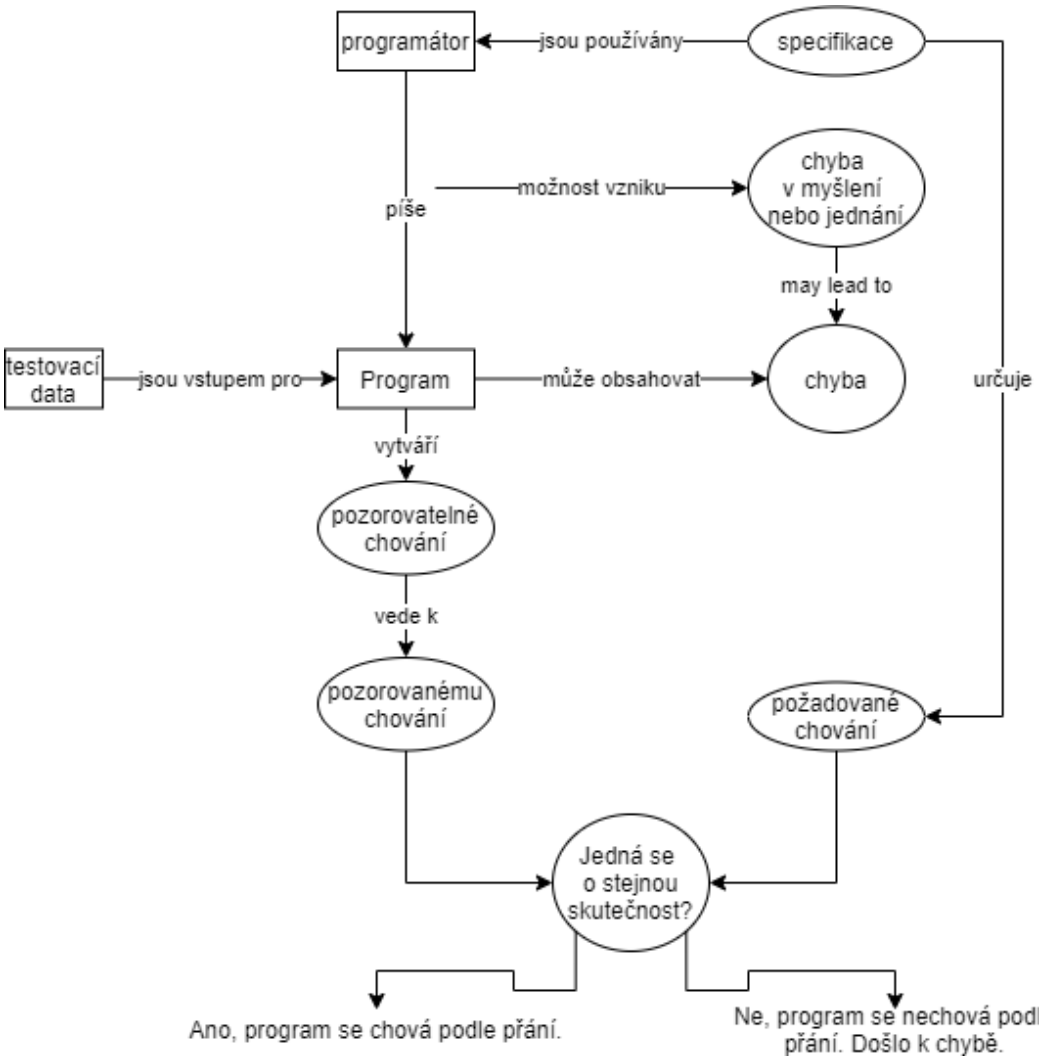
V odborných článcích, odborné literatuře nebo při debatách s odborníky je také možné setkat se s pojmem „bug“. Prostým překladem tohoto anglického slova do češtiny dostaneme slovo „brouk“. Bug nemusí být pouze softwarovou chybou, ale může se jednat i o chybu v hardwaru.

Není náhodou, že slovo „bug“ se začalo používat právě jako označení pro chybu, neboť odkazuje na opravdový hmyz, jenž byl nalezen v počítači, kde způsoboval jeho nefunkčnost. Po jeho odstranění následně dochází k jeho opětovné funkčnosti.

Proces odstranění se začal nazývat „debugging“. Následně došlo k přenosu významu na softwarové programování. Je možné tedy říci, že slova defekt a bug jsou v tomto vzájemným ekvivalentem. (Čermák, 2016)

Propojení výše uvedených definic je možné vyjádřit graficky. Díky definicím je zřejmé, že defekt je odborným termínem pro výskyt chyby v softwaru na základě předem určených kritérií.

Obrázek 1 Chyby, závady a selhání



Zdroj: Mathur, 2011, s. 6

3.1.1 **Příčiny vzniku defektu**

Důvodů vzniku defektu může být celá řada a nemusí vždy jít ani o chybu programátora. Realizátor může například udělat chybu při aktivitách, jakými jsou sestavení business modelu, hrubého návrhu, detailního návrhu apod.

Nejtypičtějšími zdroji chyb jsou:

- nesprávně pochopené potřeby zákazníka, které vedou k chybné specifikaci požadavků,
- chybně nebo jen částečně vymezené požadavky,
- požadavky jsou chybně popsány do funkčního designu,
- nepřesně interpretovaný funkční design, kvůli kterému dojde k odlišnému technickému designu,
- mylně pochopený technický design, jenž vede k nesprávně naprogramovanému softwaru,
- nedostačující výkon systému, uživatelské nepohodlí, špatná ovladatelnost a další problémy, týkající se především nefunkčních požadavků,
- chybné pochopení požadavků nebo funkčního designu, vedoucí k defektům v testovacích skriptech nebo scénářích.

Původcem vzniku defektu může být kterákoli činnost či událost, popsaná ve výše uvedených bodech, a to samostatně nebo dokonce i všechny najednou – kdy dojde k jedné chybě a ta následně přibírá další a další. (Bureš a kol., 2016)

3.1.2 **Životní cyklus defektu**

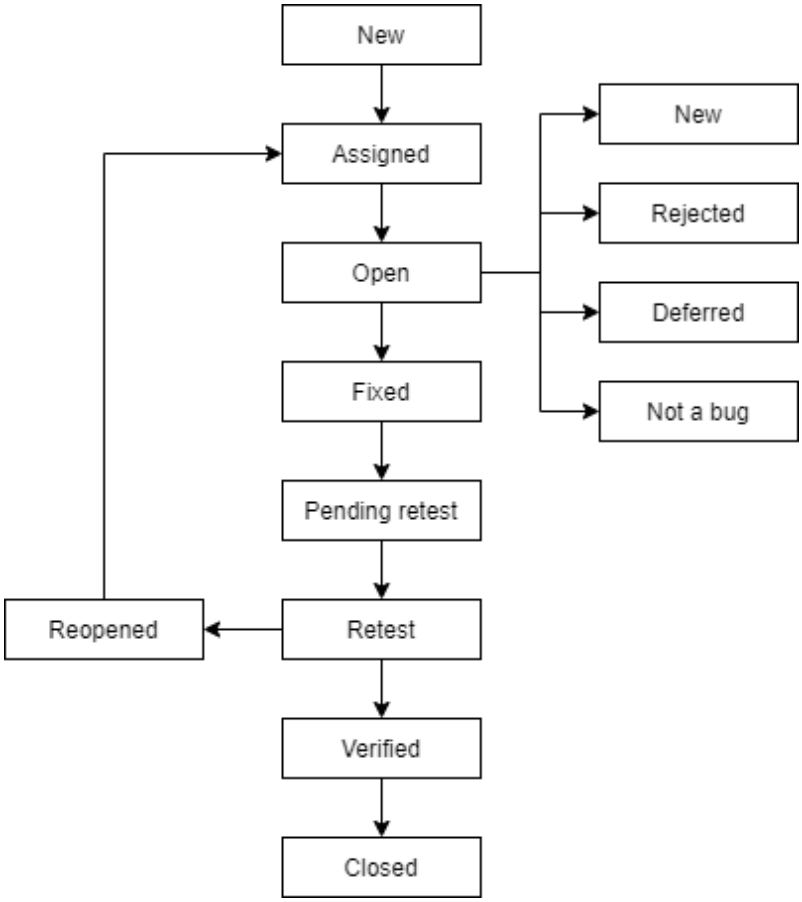
Životní cyklus defektu je chápán jako cyklus, kterým defekt projde od jeho nalezení, přes založení, vyřešení až po jeho uzavření. Defekt je spojen s chybou, která je nalezena při testování softwaru.

Během svého životního cyklu defekt nabývá různých stavů, které se v průběhu mění podle toho, jak postupuje jeho řešení.

Průběh životního cyklu defektu může být různý v každé organizaci a zároveň pro každý projekt podle potřeb. Níže na obrázku č. 2 je zobrazen cyklus, který objasňuje,

co se děje s defektem, a díky kterému je zároveň jednodušší nastavit metriky tak, aby byl defekt později lépe analyzovatelný. (Kumar, 2018)

Obrázek 2 Životní cyklus defektu



Zdroj: Kumar, 2018

Definice stavů defektu v jeho životním cyklu dle metodiky ISTQB (2020):

- **New:** Stav, kdy je defekt poprvé objeven, zaznamenán a je mu přiřazen unikátní název.
- **Assigned:** Stav, v němž po zveřejnění defektu testerem schválí vedoucí testerů, že se jedná o chybu reálnou, a přiřadí ji příslušnému vývojáři či vývojářskému týmu.
- **Open:** Stav, kdy vývojář zanalyzuje chybu a pracuje na její opravě.

- **Fixed:** Stav, kdy vývojář provede určité kroky k odstranění nalezené chyby v kódu, ověří jeho správnou funkčnost a předá opravený kód testerovi nebo testovacímu týmu.
- **Pending retest:** Stav, kdy opravený defekt byl předán testerovi.
- **Retest:** Stav, kdy tester provede opakované testování změněného kódu vývojářem, aby zkontroloval opravy nalezené chyby.
- **Verified:** Stav, kdy je kód testerem opětovně otestován pro potvrzení, že díky opravě kódu byla chyba skutečně odstraněna. V případě, že se chyba nadále neprojevuje, je stav defektu upraven na „verified“.
- **Reopened:** Stav, v němž je chyba stále přítomna, a to i po její opravě vývojářem. Defekt prochází znovu životním cyklem od stavu Open/Assigned.
- **Closed:** Stav, kdy je chyba opravena a otestována jak testerem, tak vývojářem. Pokud nebyla chyba testerem následně zreprodukována, považuje se defekt za opravený, otestovaný a schválený, což udává stav „closed“.
- **Duplicate:** Stav, kdy byla chyba hlášena jinak. Toto hlášení je tedy duplicitou k jinému.
- **Rejected:** Stav, kdy vývojář odmítá, že se jedná o chybu v kódu.
- **Deferred:** Stav, kdy se defekt odkládá, jelikož jeho priorita není tak vysoká, a očekává se, že bude opraven v příštích verzích softwaru. K tomuto stavu může dojít, pokud se vyskytuje mnoho chyb s vyšší prioritou, ale není k dispozici dostatek lidských zdrojů pro jejich opravu a zároveň chyba nemá zásadní vliv na funkci softwaru.
- **Not a bug:** Stav, kdy dojde například ke změně barvy textu, o kterou požádal zákazník. Nejedná se tedy o chybu softwaru ani jeho změnu.

3.1.3 Náležitosti defektu

Nalezené defekty mají různou prioritu řešení a různý dopad na samotný software či na spolupracující okolní systémy. Defekt, který má dopad i na okolní systémy, se nazývá integrační.

Jelikož tedy mohou existovat i integrační defekty, je nutné, aby ve společnosti byl nastaven jednotný způsob zaznamenávání defektů a jejich nezbytných náležitostí, jejichž

příklady jsou uvedeny v tabulce č. 1. Žádná metodika nepoukazuje na jednotnou škálu klasifikace a zaznamenávání defektů. Poskytují pouze doporučení, které údaje je vhodné mít u defektu vyplněné. (Graham, 2020)

Tabulka 1 Náležitosti defektu dle IEEE Computer Society

Atribut	Definice
Defekt ID	Unikátní identifikátor defektu.
Description	Popis defektu – co je špatně, co je vhodné doplnit nebo opravit.
Status	Jaký je současný stav defektu – kde se nachází ve svém životním cyklu.
Asset	Systém nebo jeho část, kde se defekt objevuje.
Artifact	Konkrétní softwarový pracovní produkt, ve kterém se defekt objevuje.
Version detected	Verze systému, ve které byl defekt nalezen.
Version corrected	Verze systému, ve které byl (bude) defekt opraven.
Priority	Priorita defektu, udává jeho závažnost pro porovnání s ostatními nalezenými defekty pro efektivní utilizaci lidských zdrojů k urgentní opravě defektů s vlivem na funkčnost systému.
Severity	Jaký je možný dopad defektu na funkčnost systému. Důležitý aspekt, který zajímá vývojáře.
Probability	Pravděpodobnost opakující se chyby způsobené defektem.
Effect	Třída požadavku, která je ovlivněna poruchou způsobenou defektem.
Type	V jaké části kódu byl defekt nalezen.
Mode	Vyjádření, čím byl defekt způsoben – špatnou implementací, přidáním funkčnosti, která nebyla potřebná apod.
Insertion activity	Činnost, při které byl defekt vložen.
Detection activity	Činnost, při které byl defekt nalezen.
Failure reference(s)	Identifikátor poruchy způsobené defektem.
Change reference	Identifikátor změny požadavku pro opravu defektu.
Disposition	Finální dispozice reportu o uzavření defektu.

Zdroj: IEEE Standard Classification for Software Anomalies, 2009

3.1.4 **Priorita defektu**

Priority defektů určují pořadí, v jakém následují jejich opravy. Čím vyšší priorita je danému defektu přiřazena, tím dříve je nezbytné defekt opravit. Defekty s nastavenou vysokou prioritou je potřeba zpracovávat urgentně a bezodkladně.

Defekty, které způsobují nemožnost používání systému, mají větší prioritu než defekty, které ovlivňují jen částečné či zanedbatelné funkcionality systému. (Rungta, 2020)

Priorita defektu může být v jednotlivých společnostech různě nastavena. Níže je uveden příklad členění na čtyři úrovně:

- **Blocker:** Chyba s touto prioritou značí, že může být důvodem znemožnění dalšího vývoje či testování.
- **Vysoká:** Vysoká priorita značí, že určité části/moduly systému mohou být po nasazení nefunkční. Funkční jsou pouze určité části systému. Systém není funkční jako celek.
- **Střední:** Chyby s označením „střední“ je nutné opravit, ale v případě potřeby (například nedostatku kapacit lidských zdrojů) je možné tyto defekty odložit ke zpracování do další verze softwaru.
- **Nízká:** Defekty s nízkou prioritou vyžadují nejnižší pozornost. Nelze je ignorovat, ale zpravidla se odbavují až poté, co jsou zpracovány defekty s vyšší prioritou.

Každá úroveň priority defektu má své číslo: 1 – naléhavá, 2 – vysoká, 3 – střední, 4 – nízká. (Yadav, 2020)

3.1.5 **Severita defektu**

Severita neboli závažnost defektu značí, jaký je celkový dopad konkrétní chyby na systém. Určuje totiž, jak chybný vliv bude mít na systém jako celek. Tato kategorie reprezentuje měřítko schopnosti chyby narušit software. Severitu defektu určuje tester, který ji nalezne a má požadované znalosti o systému. (Rungta, 2020)

Stejně jako priorita i severita má může mít různé úrovně v rozdílných společnostech.

Níže je uveden obecný popis všech 3 úrovní:

- **Kritická:** Chyba s tímto označením může znemožnit používání dílčích částí systému.
- **Důležitá:** Defekt se střední prioritou způsobuje nežádoucí chování, ale systém je stále funkční.
- **Méně důležitá:** Takto označená chyba nezpůsobí žádné neočekávané chování systému při jeho používání. (Yadav, 2020) (Software Testing Fundamentals, 2020)

3.1.6 **Vztah mezi severitou a prioritou defektu**

V tabulce č. 2 jsou uvedeny klíčové rozdíly mezi prioritou a severitou defektu.

Tabulka 2 Klíčové rozdíly priority a severity defektu

Priorita	Severita
Priorita defektu určuje, v jakém pořadí mají být defekty odbaveny a vyřešeny.	Severita defektu definuje velikost dopadu defektu na systém.

Zdroj: Rungta, 2020

Pro pochopení je na obrázku č. 3 graficky znázorněn vztah mezi prioritou a severitou. Zároveň v obrázku jsou uvedeny obecně možné příklady, kdy je konkrétní hodnota těchto dvou charakteristik defektu použita.

Obrázek 3 Vztah mezi severitou a prioritou

Zdroj: Rungta, 2020

Při určení kritické severity by se dalo očekávat, že automaticky bude použita také nejvyšší možná priorita. Toto nemusí být vždy pravda. (Petr Roudenský, 2013)

Příkladem výše uvedeného stavu může být například uvolnění beta verze s novými funkcemi na webu sociálních sítí pro mnoho aktivních uživatelů. Pokud bude nalezena nějaká chyba, která ovlivní koncové uživatele, je jí přiřazen vysoký stupeň severity, ale naopak nízký stupeň priority, jelikož je možné tuto chybu opravit až v další verzi. (Seela, 2020)

Zvláštní rozdíl mezi severitou a prioritou je ve stavech, kterými defekt prochází ve svém životním cyklu. Severita se většinou nemění po celou dobu existence defektu. Naopak priorita je od začátku přiřazována testerem, který defekt objevil, a její změna závisí na více faktorech, jakými jsou například kapacity lidských zdrojů, čas, fáze projektu a podobně. (Petr Roudenský, 2013)

3.2 **Defekt management**

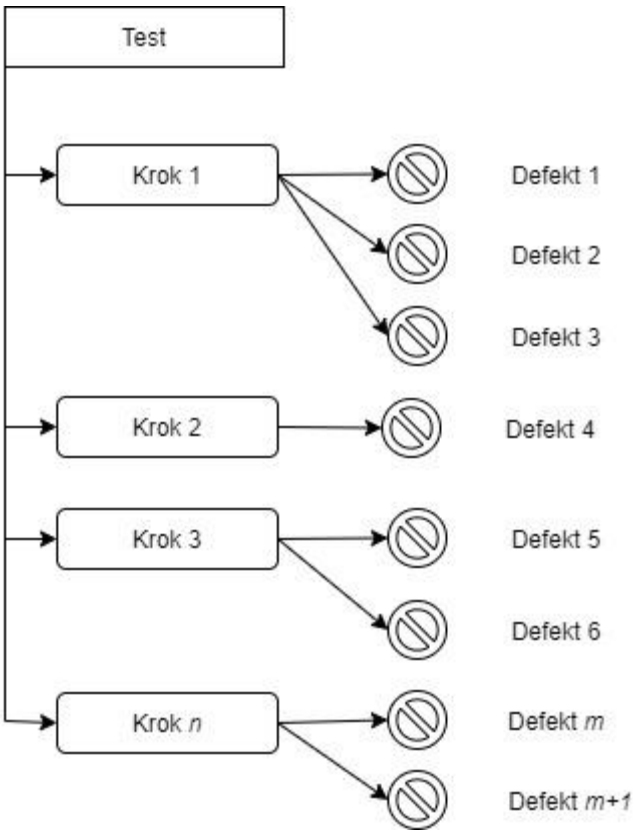
Defekt management popisuje správný postup při nalezení a opravě chyby neboli defektu. Správný postup vychází z metodik, které popisují práci vývojářského týmu a testerů. Pro korektní fungování defekt managementu je předpokladem existence systému

s databází chyb. Jako systém, který má představovat databázi chyb, se dá použít helpdeskový systém, v němž se chyby z testů dají jednoduše oddělit.

Nejvhodnějším řešením pro větší společnosti je používat pokročilejší systémy, které v sobě obsahují moduly jak pro řízení systémových testů, tak defektů. V těchto systémech je možné defekt napojit přímo na konkrétní krok testu a díky němu efektivněji specifikovat, kde chyba nastala.

Jelikož ke každému kroku může existovat nespočet defektů, znamená to, že ke každému testu může také existovat nespočet defektů. (Otta, 2016)

Obrázek 4 *Defekty na testu a testovacích krocích*



Zdroj: Vlastní zpracování

V jednotlivých společnostech se jednotlivé části procesu spadající pod defekt management mohou lišit, jelikož si ho každá společnost upravuje dle svých potřeb.

Při realizaci každého projektu v různých organizacích je nezbytně nutné seznámit se s procesem defekt managementu. (Mathur, 2011)

3.2.1 Nástroje defekt managementu

HP Application Lifecycle Management

Nástroj Aplication Lifecycle Management je dodáván výrobcem Micro Focus. Jedná se o nástroj pro evidenci jak testovacích scénářů, tak pro evidenci defektů z nich vzniklých. Tento nástroj je velice rozšířený v oblasti testování, pokud je ve firmě testování prováděno ve vodopádové podobě. Nástroj umožňuje podrobné sledování defektů ve všech jejich stavech.

K defektu je možné přiložit nespočet typů příloh, například snímek obrazovky či odkaz na webovou stránku přímo k testovacímu kroku.

Stavy defektu je možné upravovat přímo na přehledu defektů a současně v něm nechybí ani historické informace o provedených změnách. Celou databázi defektů je možné exportovat do několika formátů, z nichž uživatel později může upravit shrnující záznamy pro své potřeby.

Uživatelsky může být tento nástroj pro někoho nepřijatelný nebo nevýhodný, neboť je kompatibilní pouze s prohlížečem Microsoft Explorer. (Micro Focus, 2020)

Jira

Jira je dalším možným nástrojem pro evidenci defektů od výrobce Atlassian. Avšak nevýhodou je, že pokud společnost chce evidovat ve stejném nástroji i testovací případy, které budou uživatelsky přívětivé, musí dokoupit doplněk i pro tuto část. Jira disponuje nativní funkcí, která umí evidovat defekty a jiné potřebné entity.

Aplikace Jira je přizpůsobena fungování v agilním prostředí firem a nabízí širokou nabídku pro evidenci agilních entit, na jejichž základě jednotlivé týmy ve společnostech fungují. (Atlassian, 2020)

3.2.2 Komunikace defektu na projektu

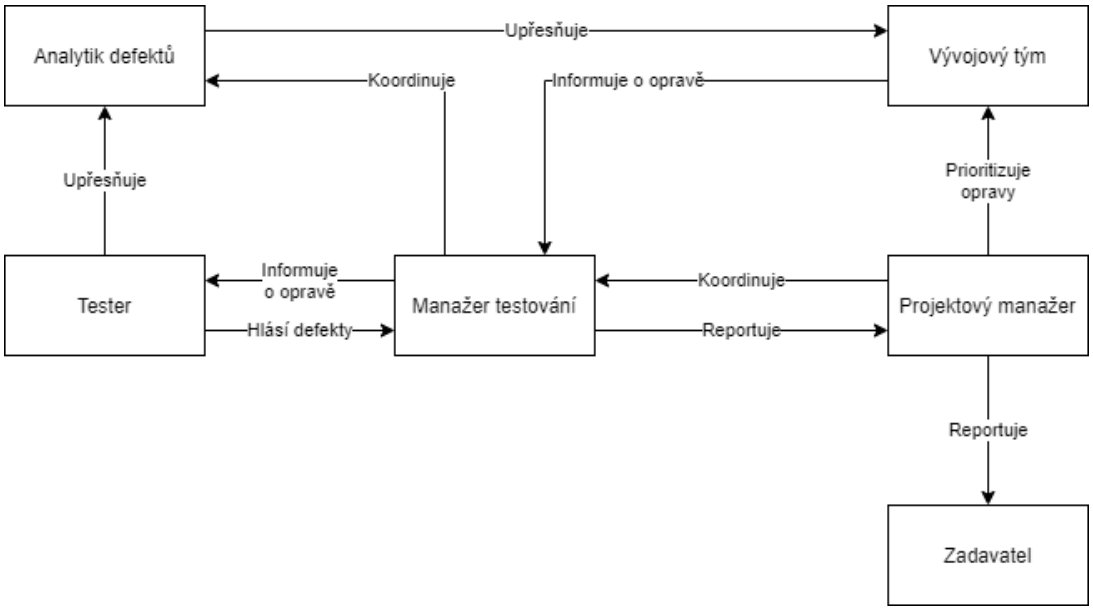
Když se blíží dokončování testů na projektu, může být situace mezi členy týmu dost napjatá. Efektivní prevencí nejrůznějších možných problémů v komunikaci je dobře připravená komunikační a eskalační mapa.

Komunikační mapa je vhodným materiálem pro nastavení efektivní komunikace mezi všemi stranami, které jsou na projektu zainteresovány. Mapa vyobrazuje informační toky

mezi stranami. Na projektové úrovni tuto mapu připravuje a odpovídá za ni projektový manažer. Vstupy do komunikační mapy za oblast testování poskytuje manažer testování.

Mapa by měla být představena členům týmu v prvotních fázích projektu. Zároveň by měla být volně přístupná všem. (Bureš a kol., 2016)

Obrázek 5 Příklad komunikační mapy pro správu defektů



Zdroj: Bureš a kol., 2016

Eskalační mapa je dalším důležitým dokumentem pro správnou a efektivní práci celého týmu v případě nálezu defektu. Určuje, na koho se obrátit v případě nutnosti při obtížích či napjatých situacích. Stejně jako komunikační mapa by měla být připravena před začátkem projektu. Každý člen týmu musí mít k tomuto dokumentu přístup a být s ním seznámen před zahájením prací nad daným projektem, ke kterému se konkrétní komunikační mapa vztahuje. Tato mapa obvykle nepracuje s konkrétními jmény, nýbrž s názvy rolí, které jsou pro každou společnost individuální.

Obecně je nejvhodnější konfliktům a jejich eskalacím předcházet, avšak na druhou stranu by se neměl případný kritický stav projektu zamlčovat, aby bylo možné se mu včas věnovat a škoda mohla být co nejdříve minimalizována. V případě eskalace je nutné dodržet profesionální postoj, věcnou a neemotivní rovinu komunikace. (Bureš a kol., 2016)

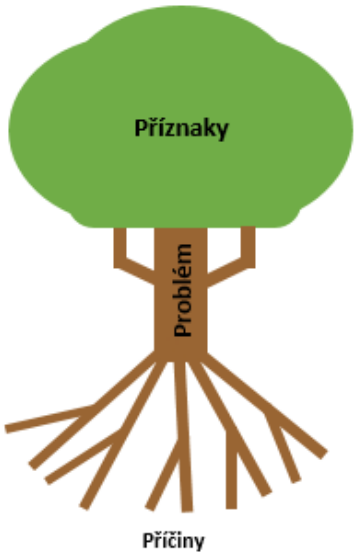
3.2.3 **Root cause analysis**

Root Cause Analysis (RCA) je technika, která se využívá k nalezení odpovědi, proč došlo k danému problému. V češtině je tato metoda nazývána analýzou kořenových příčin. Jedná se o opatření kontroly kvality, umožňující zjištění, co se stalo a pochopení proč k problému došlo. Díky této identifikaci je možné upravit proces, který zajistí snížení pravděpodobnosti opakování stejného problému.

RCA identifikuje, zda chyba byla způsobena testováním, vývojem či chybou v zadání. (Boog, 2020)

Název této metody je založen na imaginárním stromu, kde listy reprezentují příznaky problémů a kmen samotný problém. Tyto dva aspekty se nachází nad zemí, tudíž jsou viditelné. Naopak kořeny, které představují příčiny problému, jsou pod zemí a nejsou viditelné. Rostou čím dál hlouběji a mohou se rozšířit dále, než je žádoucí. (Seela, 2020)

Obrázek 6 *Grafické znázornění RCA*



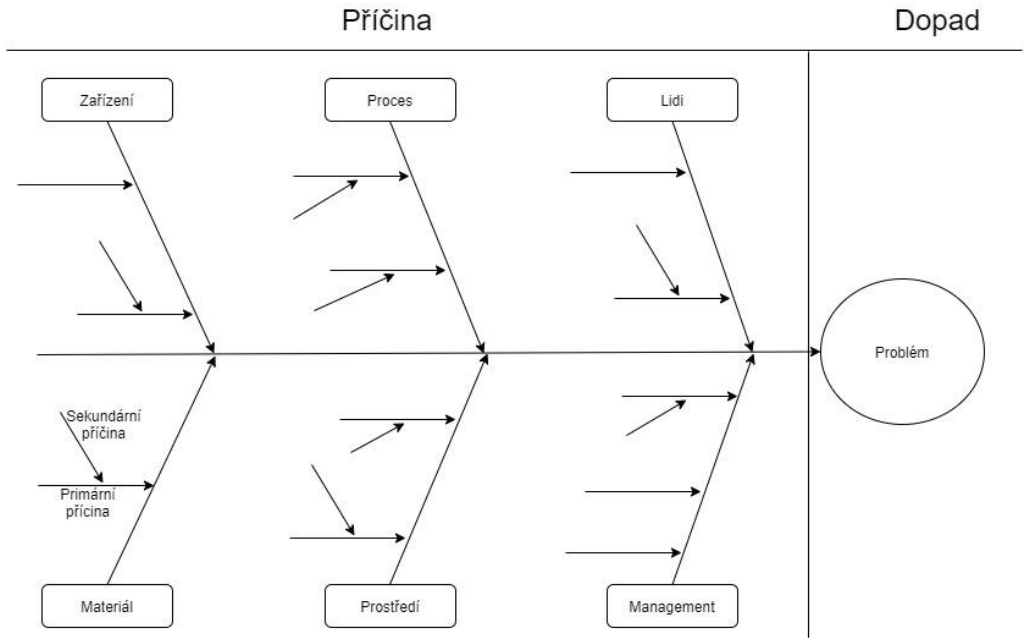
Zdroj: zpracováno dle Rahim, 2019

Jedna z nejoblíbenějších grafických podob je Iskawův diagram. Do grafu se uvádějí nejčastější možné faktory, které jsou příčinami a postupují směrem k vytváření problému.

Ne všechny problémy jsou nutně vytvářeny stejnými faktory. V některých případech se může jednat o velice rozsáhlý diagram. Vždy je potřeba zvážit, zda časová náročnost,

vynaložená na vytvoření složitého diagramu, přináší výhodu. V mnoha případech stačí i jednoduchá verze, která může být pro tým dostatečná. (Huston, 2020)

Obrázek 7 Ishikawův diagram



Zdroj: zpracováno dle Huston, 2020

3.2.4 **Metriky týkající se defektů a reportingu**

Z pohledu defektů musí být defekt koordinátor schopný poskytnout aktuální přehled všech defektů na daném projektu včetně jejich trendů, rozdělení, přiřazených řešitelů a rozdělení podle jejich oblasti testování.

V praxi se nejčastěji používá pohled na nevyřešené defekty, které jsou seřazené podle priority, aby si firma mohla vytvořit obrázek, jak tým zvládá opravovat defekty, nebo zda týmu pouze nenarůstá jejich počet.

V tabulce níže jsou uvedené nejčastěji sledované metriky aktuálních hodnot defektů (Bureš a kol., 2016):

Tabulka 3 Metriky aktuálních hodnot defektů

Otevřené defekty	
Popis	Zde jsou zahrnuty veškeré defekty, které doposud nejsou vyřešeny. Je možné je seřadit dle různých atributů např. stav, priorita atd.
Forma	koláčový graf, tabulka

Opravené defekty	
Popis	Seznam defektů, které byly opraveny a jsou připraveny na opakované testování týmem testerů.
Forma	tabulka
Report při předání do produkce	
Popis	V seznamu jsou uvedeny veškeré defekty, které byly řešeny. Stejně tak jsou zde uvedeny i neopravené defekty s dohodnutým postupem opravy.
Forma	tabulka
Hustota zbytkových chyb	
Popis	Jedná se o počet defektů, jež se dostaly do produkčního prostředí, vztažených k počtu řádků kódu (v tisících). Tato metrika vypovídá o kvalitě testování. Výpočet: $$Hustota = \frac{(počet\ defektů\ na\ produkčním\ prostředí)}{počet\ řádků\ kódu} \times 1000$$ Obvyklá hodnota na jeden Kloc (tisíc řádků kódu) jsou 2 defekty.
Forma	tabulka
Body zájmu (spotmapa)	
Popis	Tato metrika je vhodná pro business procesy. Reprezentuje jednotlivé fáze procesů, včetně jejich množství opakovaných testů na dané části kódu a stavu. Fáze jsou obarveny barvami semaforu. Defektům jsou přiřazena identifikační čísla.
Forma	tabulka

Zdroj: Bureš a kol., 2016, s. 159

Tabulka 4 Metriky trendů defektů

Vývoj defektů v čase	
Popis	Křivka nalezených defektů v čase by měla mít průběh ve tvaru písmene „S“. Na začátku testování se vyskytuje nejvíce defektů (přístup do testovacího prostředí, seznamování se se systémem apod.).
Forma	spojnicový graf, případně tabulka

Průměrné stáří defektů	
Popis	U otevřených defektů je zjišťováno, jak dlouho jsou otevřené. Z této hodnoty určíme průměr, a pokud tato hodnota v čase narůstá; znamená to, že opravy defektů se prodlužují nebo že vývojový tým má problém s opravami.
Forma	spojnicový graf nebo tabulka
Trend otevřených defektů	
Popis	Obsahuje veškeré defekty, které nejsou uzavřené. Pokud je zde rostoucí trend, mohou nastat problémy s opravami nalezených defektů.
Forma	spojnicový graf nebo tabulka

Zdroj: Bureš a kol., 2016, s. 160

3.3 Agilní metodiky

Agilní metodiky umožňují rychle a pružně přizpůsobovat řešení projektů. Jedná se o metodiky, které začaly vznikat na počátku druhé poloviny 90. let. Prosazují myšlenku, že jediná možnost, jak správnost systému ověřit, je celý nebo jeho část co nejrychleji vyvinout a prezentovat zákazníkovi. Díky rychlé zpětné vazbě od zákazníka na počátku vývoje je možné systém postupně upravovat dle přání zákazníka.

Jednotlivé agilní metodiky jsou do určité míry specifické, ale zároveň jsou všechny založeny na stejných hodnotách a principech. Na základě společných znaků se představitelé jednotlivých metodik rozhodli sepsat „Manifest agilního vývoje software“ a tím vytvořili „Alianci pro agilní vývoj software“. (Buchalcevová, 2005)

3.3.1 Hlavní principy agilních metodik

Manifest agilního vývoje softwaru definuje základní principy, jež je nezbytné při vývoje softwaru uplatňovat. Těchto principů vzniklo celkem 10:

1. *„Nejvyšší prioritou je včas a kontinuálně dodávat software, který zákazníkům přináší hodnotu.“*

Pro zákazníky nemají grafy ani dokumenty při vývoji softwaru hodnotu. Nejvíce je zajímá, zda dostanou funkční systém a zda budou uspokojeny jejich potřeby. Je důležité, aby hodnota byla stále prověřována, jelikož je předmětem změny.

2. *„Změnu požadavků je možné provést i v pozdějších fázích vývoje, protože tím může zákazník získat konkurenční výhodu.“*

V agilním přístupu se prosazuje snaha o efektivní realizaci a řízení rizik negativních dopadů. Je žádoucí realizovat krátké iterace (2 týdny až 4 měsíce), na jejichž konci je dodáno fungující řešení. Agilní metodiky dávají důraz na zkrácení cyklu pro dodání přírůstku.

3. *„Uživatelé a vývojáři spolupracují denně na projektu.“*

Agilní přístupy vycházejí z přesvědčení, že není možné definovat veškeré požadavky na začátku projektu. Definují na začátku pouze hrubé návrhy požadavků, které se mění na základě denních jednání s uživateli, díky nimž se specifikují nebo i mění.

4. *„Motivovaní jedinci, kteří mají vytvořeny podmínky pro práci a mají podporu vedení, jsou klíčovým faktorem úspěchu projektu.“*

Projekt musí být podporován všemi zapojenými stranami. Zároveň je nutné pracovat se skutečností, že práce závisí především na lidech, a je proto nezbytné jejich motivaci soustavně udržovat.

5. *„Nejefektivnějším způsobem přenosu informací v rámci vývojového týmu je osobní komunikace.“*

Dokumentace není stěžejním základem projektu. V rámci týmu se podporuje přímá komunikace se zainteresovanými stranami směřující k efektivnímu a rychlejšímu nalezení řešení.

6. *„Primární mírou úspěchu je fungující software.“*

Pro úspěch projektu je požadován fungující systém, jelikož ani plnění či neexistující dokumentace jej nezajistí.

7. *„Agilní procesy předpokládají „zdravý“ vývoj.“*

Pro vývojáře a jisté členy týmu je potřeba jasně definovat hranice, aby se jejich produktivita nesnižovala. Například je nezbytné vymezit pracovní dobu tak, aby v jejím rámci zůstal tým v dobré kondici.

8. *„Perfektní technické řešení i návrh.“*

Důraz je kladen na kvalitu návrhu, která je nutná pro realizaci změn. Návrh je iterativní činnost, která je prováděna v průběhu projektu.

9. *„Zásadním požadavkem je jednoduchost řešení, tj. umění maximalizovat množství neudělané práce.“*

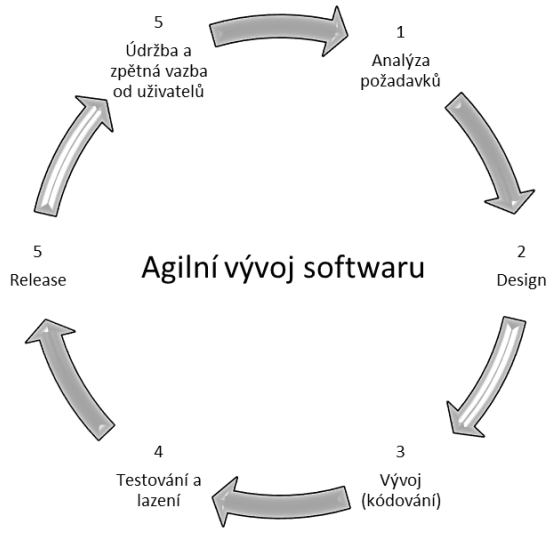
Je možné použít větší množství metod a postupů – zejména těch jednoduchých.

10. *„Nejlepší architektury, požadavky a návrhy vznikají ze samoorganizujících se týmů.“*

V tomto bodě je zdůrazněna dennodenní komunikace, přizpůsobování metodiky a kreativita lidí. (Buchalcevová, 2005) (Beck, et al., 2001)

Na obrázku 8 je znázornění agilního vývoje softwaru.

Obrázek 8 *Agilní vývoj softwaru*



Zdroj: zpracováno dle Sweeney, 2020

3.3.2 **Scrum**

Scrum je v dnešní době nejpoužívanější agilní metodikou. Tato metodika byla vytvořena v roce 1990 a jejími autory jsou Ken Schwaber a Jeff Sutherland.

Název byl odvozen z ragbyového termínu „scrumage“ neboli „mlýn“, který reprezentuje situaci při hře, kdy celý tým se shromáždí ve mlýně a společně se snaží získat/udržet míč. V tomto herním okamžiku musí celý tým tlačit stejným směrem.

Scrum je celý založen na třech pilířích, jimiž jsou: transparentnost, kontrola a adaptace. Jedná se o to, že všichni členové týmu by měli mít jasný přehled o tom, co se dělá, jak se to dělá, proč se to dělá a v jakém je to stavu.

Využívají se krátké iterace, kterým se říká „sprinty“, a jsou typicky dlouhé 1 až 2 týdny, avšak ne delší než 1 měsíc. Na konci jednotlivých sprintů by měla být hotova část produktu, na niž tým může získat zpětnou vazbu od zákazníka. Existuje trvalá zpětná vazba na produkt díky opakování cyklu vývoje v neustálém kruhu. (PM Consulting, 2021)

3.3.2.1 **Scrum tým**

Jelikož scrum je agilní metodikou, tak i role jsou oproti klasickému vodopádovému modelu odlišné. Aby tým správně fungoval, je nutné zavést nové role – „Scrum Master“ a „Vlastník produktu“.

Scrum Master je osoba, která pomáhá celému týmu, ale nejedná se o klasického manažera týmu. Funguje jako tzv. „mezičlánek“ mezi týmem samotným a jakýmkoliv elementem zvenku, který by mohl narušovat práci členů.

Jeho hlavní náplní je vytvoření samostatného, efektivního a spokojeného samoorganizovaného týmu – dokáže dokončit práci bez pomocné síly. Zároveň zabezpečuje dodržování principů agilních metodik a scrumu, současně však také dohlíží na vzdělávání týmu v oblasti používání agilních nástrojů. Jeho cíle a povinnosti jsou definovány v následujících bodech:

- pomáhá dosáhnout vytyčených cílů celému týmu,
- odstraňuje vzniklé problémy při práci členů týmu,
- motivuje členy týmu k lepším výsledkům.

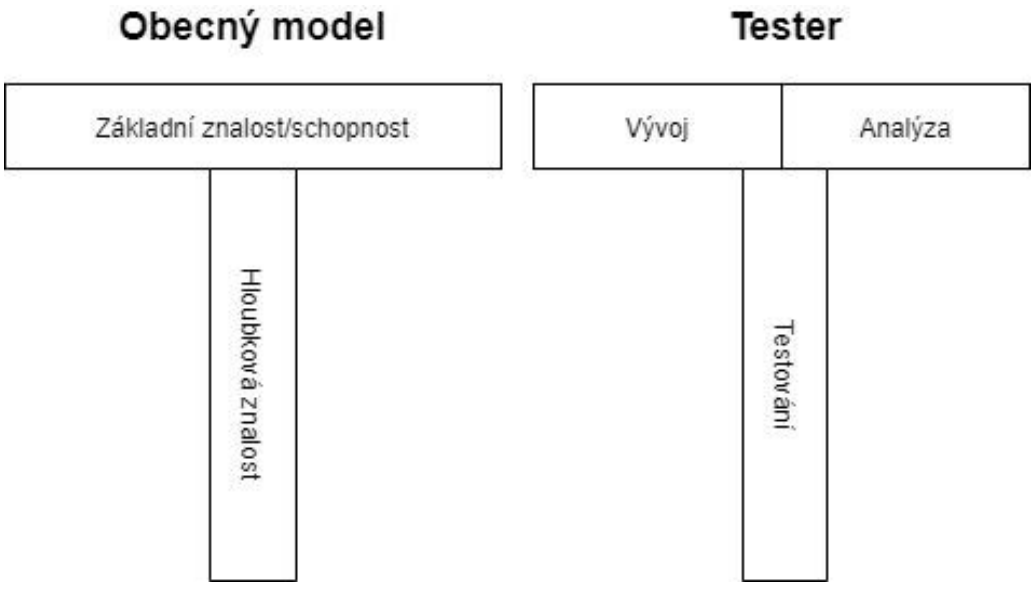
Vlastník produktu se stará o definování vize projektu a transparentnost komunikace směrem k týmu, zákazníkům a firmě. Stanovuje priority požadavků, díky čemuž může rozhodovat, v jakém pořadí se bude pracovat na funkcionalitách systému. Do jeho zodpovědnosti spadá celý product backlog. Na rozdíl od Scrum Mastera netráví s týmem tolik času, ale naopak se pravidelně schází se zákazníky, aby dokázal správně pochopit jejich požadavky a správně se rozhodoval, jakou je ta pravá hodnota pro zákazníka.

Jeho zodpovědnost je klíčová pro hlavní čtyři fáze, které zahrnují: plánování sprintu, denní scrum, recenzi sprintu a retrospektivu.

Členové týmu vyvíjí a zlepšují produkt na základě prioritizovaného backlogu. Mezi členy týmu řadíme i vývojáře a testery. K projektu mohou být přizváni i specialisté, avšak nejefektivnější týmy jsou složeny z lidí, kteří jsou nejen všestranně schopní, ale také dokáží efektivně organizovat své pracovní postupy a kroky.

Důraz je kladen na vzájemnou zastupitelnost mezi členy týmu. Na rozdíl od vodopádového modelu, kdy na jednotlivé oblasti jsou alokováni specialisté, se scrum snaží vyhnout vytváření takovéto závislosti. Jelikož si členové týmu navzájem pomáhají a sdílí spolu zkušenosti, učí se tím novým dovednostem, stávají se všestrannějšími a prohlubují funkčnost celého týmu. Tento přístup se zobrazuje pomocí obráceného písmene „T“, a díky tomu se mu říká T-shape. (Šochová a Kunce, 2014)

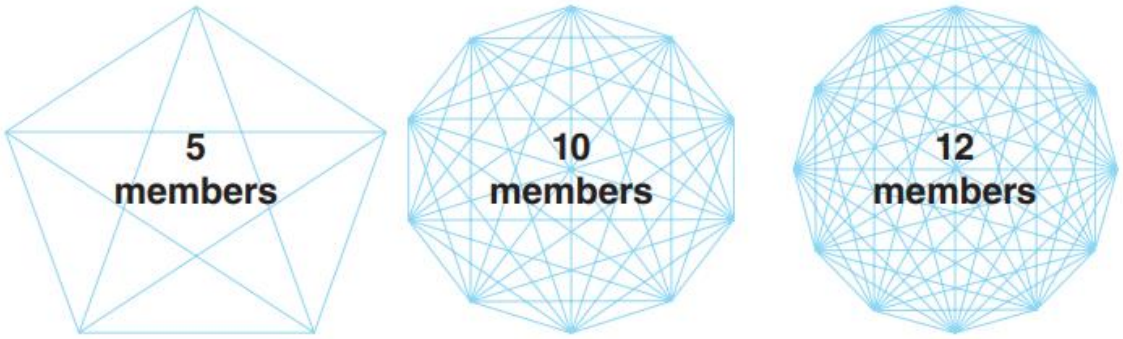
Obrázek 9 T-shape model



Zdroj: zpracováno dle Thorn, 2016

Komunikace mezi členy celého týmu při dodržování metodiky scrum je velmi důležitá. Dle Kerznera (2018) komunikační kanály v týmu rostou exponenciálně na základě vztahu: $\boldsymbol{velikost\ týmu \times (velikost\ týmu - 1)/2}$.

Obrázek 10 Týmové komunikační kanály v závislosti na velikosti týmu



Zdroj: Kerzner, 2018

3.3.2.2 **Týmové ceremonie**

Scrum definuje základních pět ceremonií, přičemž každá z nich má jinou náplň. Tyto ceremonie jsou velice důležité pro udržení morálky, efektivity týmu a při otevírání hranic komunikace.

Standup, nebo také Scrum Meeting, je jedna z nejznámějších praktik využívaná scrum metodikou. Meeting není složitý na implementaci, proto s ním většina týmů může začít ihned. Schůzka se koná každý den, kdy se všichni členové týmu ráno sejdou na jakémkoli jiném místě než svém pracovním a sdílí mezi sebou informace, na čem pracují, na čem budou pracovat daný den, případně zda mají nějaké problémy, o nichž by se tým měl dozvědět.

Scrum Master zde vystupuje v roli moderátora. V případě, že se chce přijít podívat někdo z organizace, kdo není členem týmu, jeho vstup není zakázán. Je zde pak pouze v roli pozorovatele a nezasahuje do dění schůzky.

Každý člen týmu by během standupu měl zodpovědět základní otázky:

- co jsem dokončil včera,
- co dokončím dnes,
- jaké mám problémy.

Celá schůzka by z důvodu efektivity neměla být delší než 15 minut. Pokud je nutné nějakému tématu věnovat větší pozornost, svolá se další schůzka pouze se zainteresovanými stranami. (Ingeno, 2018, s. 35)

Retrospektiva je nástrojem, který přináší zpětnou vazbu. Jde o zpětnou vazbu v jakémkoli kontextu pro zlepšování nebo inovace práce jak týmu, tak jednotlivce. Událost trvá zhruba hodinu a je naplánována na konec sprintu.

Všichni členové týmu by zde měli identifikovat klíčové problémy, překážky nebo naopak vyzdvihnout, co bylo ve sprintu dobře. Mělo by se jednat o konstruktivní připomínky k práci.

Schůzka je rozdělena do několika fází, kterými prochází:

1. Úvod: Připomenou se pravidla retrospektivy a projde se seznam akcí z minulé retrospektivy a potvrdí se program.

2. Sběr dat: Časový prostor pro získání zpětné vazby a informací o tom, co funguje, a na co je třeba se zaměřit.

Jsou zde položeny otázky, které odpoví každý člen sám za sebe na lístek. Obvykle otázky mají následující charakter:

- V čem chci pokračovat a co se mi líbilo?
- S čím chci přestat a co se mi nelíbilo?
- Co nového bych rád zavedl?

3. Porozumění získaných informací: po zodpovězení otázek Scrum Master vybere lístky, a poté tým hlasuje o položkách, které jsou nejzávažnějšími a v dalším sprintu je chtějí změnit.

4. Brainstorming možností: pro vybrané položky se navrhne řešení a na další retrospektivě se znovu zhodnotí, zda se povedlo problém odstranit.

5. Shrnutí konkrétních akcí: schůzka končí shrnutím konkrétních kroků ke zlepšení či změně. Pokud by zde nedošlo ke shrnutí konkrétních aktivit, stalo by se v rámci retrospektivy pouze „povídáním“ o problémech bez možnosti změny. (Šochová a Kunce, 2018, s. 70-71)

Plánování sprintu (Sprint Planning) je událost, na které se podílí celý tým. Provádí se první ráno nového sprintu. Schůzka by měla trvat mezi 1-2 hodinami za každý týden sprintu, což v případě čtrnáctidenního sprintu znamená, že by tato schůzka měla trvat 2-3 hodiny. Se zvyšujícími se schopnostmi a znalostmi agilních metodik se doba plánování může zkracovat.

Pro odhadnutí náročnosti jednotlivých požadavků jsou použity story pointy, jež vycházejí ze škály Fibonacciho posloupnosti a udávají komplexitu daných aktivit. Pro odhad jednoho požadavku by nemělo být vynaloženo více než pět minut.

Cílem této schůzky je definovat úkoly, na kterých se ve sprintu bude pracovat.

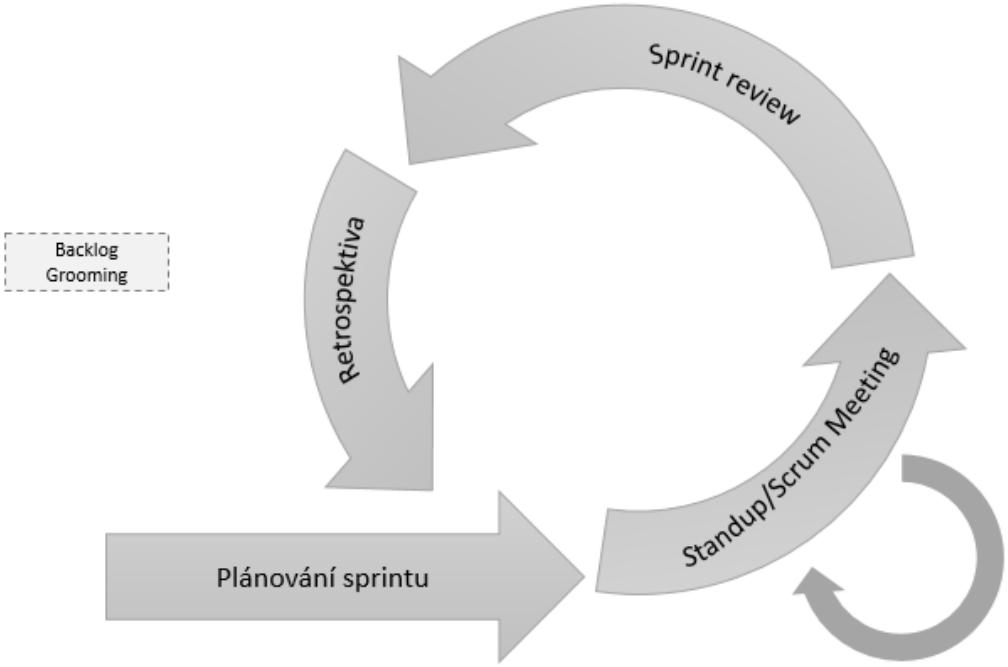
Sprint review je setkání v posledním dni sprintu. Předvádí se dokončené projekty a je nutné získat zpětnou vazbu od zákazníků. Na schůzce jsou přítomni členové týmu a zároveň jsou vítáni investoři a zadavatelé. Časově sprint review není nijak časově ohraničeno.

Posbíraná zpětná vazba je zapsána a evidována k požadavkům budoucího rozvoje.

Backlog Grooming je velmi důležitá schůze pro týmy, jež pracují v komplexním prostředí, které má složitou architekturu a mnoho integrací. Nestačí dělat průběžnou přípravu v probíhajícím sprintu, ale je potřeba konzistentní čas pro porozumění jednotlivým user stories.

Běžně se plánuje do poloviny sprintu, v případě potřeby i několikrát v jeho průběhu. Na Backlog Groomingu se schází vlastník produktu s týmem a ujasňují si, co každá user story znamená a jak lze rozeznat, že je již hotová. (Šochová a Kunce, 2018, s. 79 - 83) (Radigan, 2020)

Obrázek 11 Scrum ceremonie v čase



Zdroj: zpracováno dle Visual Paradigm, 2018

3.3.2.3 **Arftefakty**

Sprint je iterace, která by měla trvat v rozmezí 1-4 týdnů. Zpravidla bývá nejčastěji využívána délka 2 týdnů. Během jednotlivých sprintů je vytvářena část nové funkcionality nebp bývá upravována stávající. Cílem každého sprintu je dodat novou hodnotu systému.

Product backlog je seřazený seznam veškerých požadavků, které je potřebné vytvořit pro zlepšení produktu. Po celou dobu trvání projektu je seznam neustále udržován. Z něho se následně vybírají aktivity, které se zařazují do následného sprintu, v němž budou zpracovány. Každá jednotlivá aktivita má přiřazenou prioritu, podle níž je řazena v backlogu.

Vzhledem k využití agilní metodiky není nikdy seznam úkolů úplný. Na počátku obsahuje jen jasné základní požadavky. V průběhu projektu jsou přidávány nové úkoly, dle samotného vývoje produktu a prostředí. Product backlog je tedy dynamicky se měnící seznam úkolů, jenž se mění za účelem maximalizace hodnoty výsledného produktu.

Sprint backlog obsahuje aktivity, které mají být odbaveny během sprintu. Činnosti sem zařazené se vybírají z produktového backlogu.

Jednotlivé činnosti jsou ohodnoceny časově v podobě story pointů a přidělené konkrétnímu řešiteli. Každý člen týmu má na starost několik činností v průběhu sprintu. Pořadí řešení činností si vybírá dle svého uvážení a případné návaznosti na další aktivitu.

User Story je často používaný pojem v agilních týmech. Jde o vyjádření, co chceme udělat, pro koho a hlavně proč. Utváří jednotný obrázek pro vývojové týmy, co se po nich požaduje.

Done kritéria jsou kritéria, která určují, jak se pozná, že konkrétní úkol je hotový. Přestože jsou používány různé nástroje, v nichž zachyceny průběhy jednotlivých úkolů, je nezbytné, aby poslední krok byl shodný se všemi projekty. (Šochová a Kunce, 2018) (Schwaber a Sutherland, 2017) (Sutherland, 2012)

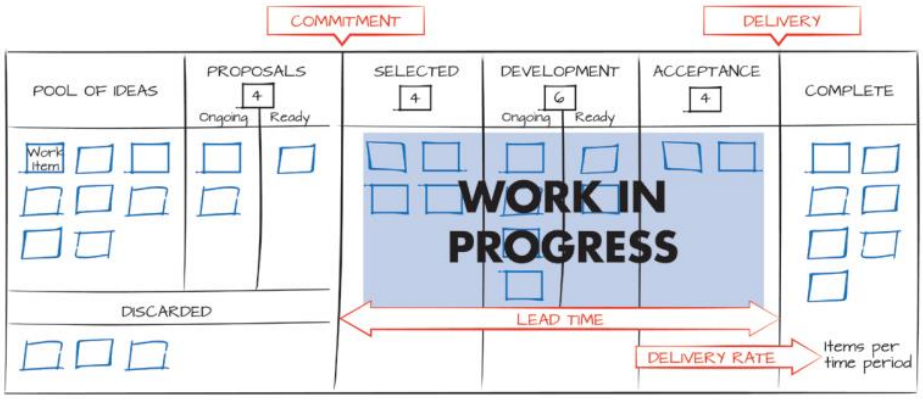
3.3.3 **Kanban**

Kanban je agilní metodika, která vizualizuje práci a pomáhá omezit rozpracované úkoly a maximalizovat efektivitu. Cílem této metodiky je identifikace potenciálních překážek v procesu a jejich následná oprava tak, aby práce mohla protékat efektivním způsobem.

Metodu nejvíce rozvíjela společnost Toyota ve 40. letech 20. století. Při výrobě byl využíván systém lístků a jejich tok se stal zdrojem zpětných informací.

V současnosti je kanban využíván pro vývoj softwaru. V překladu kanban znamená tabule, která prezentuje postup jednotlivých úkolů, které dohromady vytvářejí požadovanou funkcionalitu. Každý lístek představuje jednotlivý úkol, jenž se v průběhu řešení posouvá ve sloupcích na tabuli dle jeho aktuálního stavu. Jednotlivé sloupce kanbanové tabule se mohou lišit dle potřeb projektu. (Anderson a Carmichael, 2016) (Scotland, 2010)

Obrázek 12 Příklad kanbanové tabule



Zdroj: Anderson a Carmichael, 2016

Mezi základní stádia pro vývoj softwaru by mělo patřit:

- Backlog – schránka lístků s úkoly, které jsou následně posouvány podle stavu dále.
- In progress – úkoly, které jsou rozpracované.
- To test – vývoj dané funkcionality (resp. úkol) je vyřešen a je možné ji otestovat na správnou funkčnost.
- Done – funkcionalita je vyvinuta a otestována a může být předána zákazníkovi. (Anderson, 2020)

Kanban se opírá o pět klíčových vlastností:

1. Vizualizace workflow

Veškeré procesní kroky jsou vyobrazeny na fyzické či virtuální kanbanové tabuli. Tabule může být jednoduchá až velmi komplikovaná v závislosti na složitosti procesu.

2. Omezení množství rozpracované práce

Záměrem kanbanu je využití veškerých alokovaných kapacit na aktuální projekt a zabránění jejich zahlcení. Díky vytvořenému backlogu si vývojáři přebírají jednotlivé požadavky podle svých volných kapacit – nedochází tak k jejich zahlcení.

3. Řízení toku

Cílem řízení toků je sledování a reportování toku rozpracovaných lístků v celém projektu a zajistit tak co možná nejplynulejší a nejrychlejší průchod veškerých lístků projektem za snížení rizika jakéhokoli prodlení.

4. Zjednodušení procesu

V oblasti vizualizace procesu dává smysl explicitně definovat a vizualizovat zásady, podle nichž je práce vykonávána. Vytváří se tak společný základ, díky němuž by všichni účastníci procesu měli pochopit, jak práci v systému provádět.

5. Společná práce na zlepšení

Tato metoda je také dynamickým procesem zlepšování. Podporuje přijímání malých změn, vedoucích k postupnému zvyšování tempa ve stanoveném objemu práce tak, aby ji tým lehce zvládal. (Anderson, 2020) (Anderson a Carmichael, 2016) (Scotland, 2010)

4 Vlastní práce

4.1 Česká spořitelna, a. s.

Česká spořitelna, a. s. patří v současné době mezi největší společnosti v oblasti bankovnictví. V srpnu roku 2020 Česká spořitelna, a. s. vykázala okolo 4,6 mil. klientů. Dále vlastní největší síť poboček a bankomatů na území České republiky.

Česká spořitelna, a. s. je od roku 2000 součástí finanční skupiny ERSTE, působící v 11 zemích východní a střední Evropy.

Mimo jiné se Česká spořitelna, a. s. zabývá také například životním prostředím a jeho ochranou, podporuje studenty a nabízí jim možnost propojit znalosti nabyté ve škole s praxí, snaží se pozvednout úroveň finanční gramotnosti již u dětí a je partnerem českých vysokých škol.

V roce 2018 Česká spořitelna, a. s. začala procházet transformací způsobu řízení projektů z vodopádového přístupu na agilní řízení. Tato transformace si klade za cíl zvednutí klientské spokojenosti a zefektivnění fungování bankovního institutu jako celku. Tato změna se dotkla významně pracovníků na centrále, kteří museli upravit svůj styl práce. Některé pracovní pozice úplně zanikly, avšak proti tomu vznikala zcela nová pracovní místa. Současně vzniklo i nové uspořádání jednotlivých pracovních týmů.

4.1.1 Agilní uspořádání společnosti Česká spořitelna, a. s.

Vlivem transformace vznikla nová organizační struktura agilního typu. V rámci jednotlivých velkých sekcí vzniklo několik tribů, které se dále rozdělily na squady. Veškeré role jsou v bance soustředěny do rozdílných chapterů.

4.1.1.1 Tribe

Tribe - česky kmen - je organizační uspořádání, zaměřující se na rozvoj a podporu produktů v rámci určité bankovní oblasti. Squady v rámci jednoho tribu se dohromady zaměřují na splnění stanovených cílů pro tribe. V každém tribu je zhruba 50 až 150 členů s různými specializacemi. Česká spořitelna, a. s. je rozdělena do více než 20 tribů.

Zároveň na stejné úrovni jako triby se nachází centra expertíz, která mají za úkol podporovat triby a obchodní síť. Centra expertíz jsou specializována na určitou oblast,

například na testing nebo data science. V České spořitelně jsou centra expertíz označována zkratkou „*CoE*“, tudíž centrum expertízy za testing se jmenuje „*CoE Testy*“.

4.1.1.2 **Squad**

Výše zmíněné triby se dělí na jednotlivé squady se specifickým zaměřením. Squady jsou autonomními skupinami, které směřují ve své činnosti k vývoji produktů nebo funkcionalit. Členy do squadů vybírá Tribe Lead ve spolupráci s Chapter Leadem a agilními kouči.

Squad obsahuje přibližně 10 pracovníků, navzájem spolu řešícími úkoly daných sprintů. Všichni členové jsou si rovni a mají stejně důležitý hlas při rozhodování. Mezi sebou si samostatně rozdělují úkoly z prioritizovaných úkolů od vlastníka produktu. Každý squad má svého produktového vlastníka a členy, jež jsou z různých chapterů a jsou specializováni na různé oblasti. Česká spořitelna, a. s. interně rozděluje pozice dle specializace na:

- Designer
- Vývojář
- Analytik
- Expert na zákaznické cesty
- Expert na závazek (Engagement expert)
- Testing expert
- Product Owner

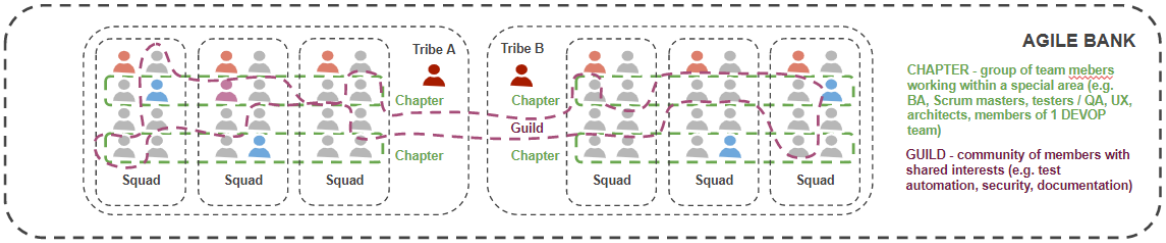
4.1.1.3 **Chapter**

Chapter – v českém jazyce kapitola – je shromáždění expertů s určitou specializací (řemeslem), jenž mohou být z rozdílných squadů. Jejich rozvoj zajišťuje pracovník s pozicí Chapter Lead na úrovni tribu, který má za úkol mentorovat je a rozvíjet v jejich znalostech specializace. Chapter Lead se sám neustále vzdělává, rozvíjí svou specializaci a pomáhá najít metody, které mohou pomoci zefektivnit práci v zaměřené oblasti.

Nad Chapter Leady, tedy nad úrovní tribu, figuruje osoba s rolí Area Chapter Lead, který společně s Chapter Leady vytváří minimální standardy a kvality pro využívané metodiky v dané specializaci napříč celou bankou.

Defekt Management spadá pod chapter testing a Area Chapter Leadem je osoba určená pro testing.

Obrázek 13 Chapter v bance



Zdroj: vlastní zpracování

4.1.1.4 Agilní kouč

V České spořitelně jsou dva typy agilních koučů – na úrovni squadu a na úrovni tribu. Agilní koučové mají za úkol rozvíjet, šířit a pomáhat pochopit agilní procesy a principy. Squad agilní kouč funguje pro určité squady, zpravidla pro dva až tři. Znalost a zkušenost agilních koučů je klíčová pro fungování týmů v agilním procesu. Squad agilní kouč dohlíží na dodržování stanovených agilních metodik, stanovených ceremonií a postupů. Měli by utvářet vyhovující atmosféru ve squadu a pomáhat odstranit nežádoucí vliv okolního prostředí, které mohou squad zdržovat v jejich práci.

Hlavním cílem agilního kouče je podpora vytvoření samostatně a efektivně fungujícího týmu. Agilní kouč není nadřízeným ostatním členům, nýbrž je jen navádí správným směrem svými radami a návrhy.

4.2 Nový defekt management

V rámci celé transformace České spořitelny, a.s. bylo rozhodnuto o používání jednotného systému pro celou banku. Důvodem byly finance a zároveň lepší přizpůsobení agilnímu řízení. Pro tuto skutečnost byl vybrán systém Jira, který v bance byl již dříve využíván pro evidenci požadavků a v současné době nabízí i možnosti pro agilní fungování společnosti. Systém Jira je v současnosti v bance již využíván pro evidenci požadavků. Interně je systém Jira označován jako „*FUJIRA*“ (Future Jira).

Jira definuje jednotlivé týmy jako projekty, jimž náleží určití členové. Z důvodu technické náročnosti byla přijata varianta, že veškeré defekty (vývojové, nevývojové,

integrační i neintegrační) budou evidovány v jednom projektu s názvem „*Defect Management“*. Uživatelé pro práci s defekty budou využívat dashboardy, které si nakonfigurují vlastním filtrem podle potřebného pohledu.

4.2.1 **Analýza pro nový defekt management**

V této kapitole jsou popsané požadavky, které vznikly na základě interní analýzy pro vznik nového defekt managementu. Na počátku stály tři hlavní důvody, na jejichž základě vznikla tato aktivita:

- Systém HP ALM je již pomalý, technicky zastaralý, nepodporuje nynější agilní způsob práce a tím pádem ani nevyhovuje potřebám banky.
- Počínaje rokem 2021 bylo rozhodnuto o úsporách v rámci licenční podpory systému HP ALM.
- Nemalá část současných týmů již interně pracuje s defekty v Jira, a tím vznikla potřeba práci sjednotit tak, aby došlo k využití jednoho nástroje pro management požadavků i pro defekt management.

4.2.1.1 **Požadavky na systém**

Konzultacemi s osobami odpovědnými za proces defekt managementu byly zároveň definovány základní požadavky na systém. Požadavky jsou uvedeny v tabulce 5 – Definice požadavků na systém. Definice priority požadavků vychází z prioritizační metody MoSCoW. Jednotlivé požadavky jsou označené písmenem z anglických spojení *Must have (M), Should have (S), Could have (C)* a *Won’t have (W).* Na základě doporučení, vyplývajících z této metody, by rozvržení úsilí nezbytného pro splnění předem definovaných požadavků mělo být následující:

- *Must have (M)*: Maximálně 60 % celkového plánovaného úsilí na projekt.
- *Should have (S)*: Maximálně 20 % celkového plánovaného úsilí na projekt (v součtu s prioritou *Must have* tvoří 80 % celkové práce na projektu).
- *Could have (C)*: Zbylých 20 % tvoří časový buffer, pro potřeby vzniknutí nepředvídaných problémů.
- *Won’t have (W)*: Požadavky, u kterých bylo rozhodnuto, že budou přesunuty do další fáze vývoje.

Tabulka 5 Definice požadavků na systém

ID	Název	Priorita
P-01	Funkční defekt management v nástroji Jira	M
P-02	Jednotný způsob práce s defekty napříč bankou	M
P-03	Vytvoření nového workflow pro defekty	M
P-04	Unifikace a minimalizace polí	S
P-05	Zajištění schopnosti reportingu	S

Zdroj: vlastní zpracování

- **P-01 Funkční defekt management v nástroji Jira** – Požadavek spočívá v přesunutí celého defekt managementu z nástroje HP ALM do systému Jira. Nově vytvořená entita defektu a celý proces defekt managementu v nástroji Jira musí být funkční od 17. srpna 2020.
- **P-02 Jednotný způsob práce s defekty napříč bankou** – Vzhledem k jednotnému fungování celé banky je potřeba zajistit sjednocený způsob práce s defekty napříč všemi odděleními a týmy v bance, a to z důvodu zajištění schopnosti včasného odbavování jak integrační tak neintegračních defektů.
- **P-03 Vytvoření nového workflow pro defekty** – Z pohledu sjednocení práce je třeba vytvořit nové jednotné workflow a jednotlivé stavy defektu v jeho životním cyklu.
- **P-04 Unifikace a minimalizace polí** – Z důvodu spolehlivosti, menší zátěže na systém a jednoduššího používání je nutné minimalizovat počet používaných polí a propagovat jejich unifikaci.
- **P-05 Zajištění schopnosti reportingu** – Z pohledu žádaného reportingu je důležité se zaměřit na schopnost práce s jednotnými metrikami a vytváření jednotlivých reportů.

4.2.1.2 **Přínosy**

Od 17. srpna 2020, kdy se defekt management má spustit v novém systému, se Jira stane jednotným místem pro práci s požadavky, testy i defekty se vzájemnými propojením.

Dříve byly požadavky v systému Jira oddělené od testů a defektů, které byly evidovány v systému HP ALM. Systém Jira využívá doplněk Xray pro evidenci a práci s testy, což usnadní práci i s defekty, převážně vznikající právě z těchto testů.

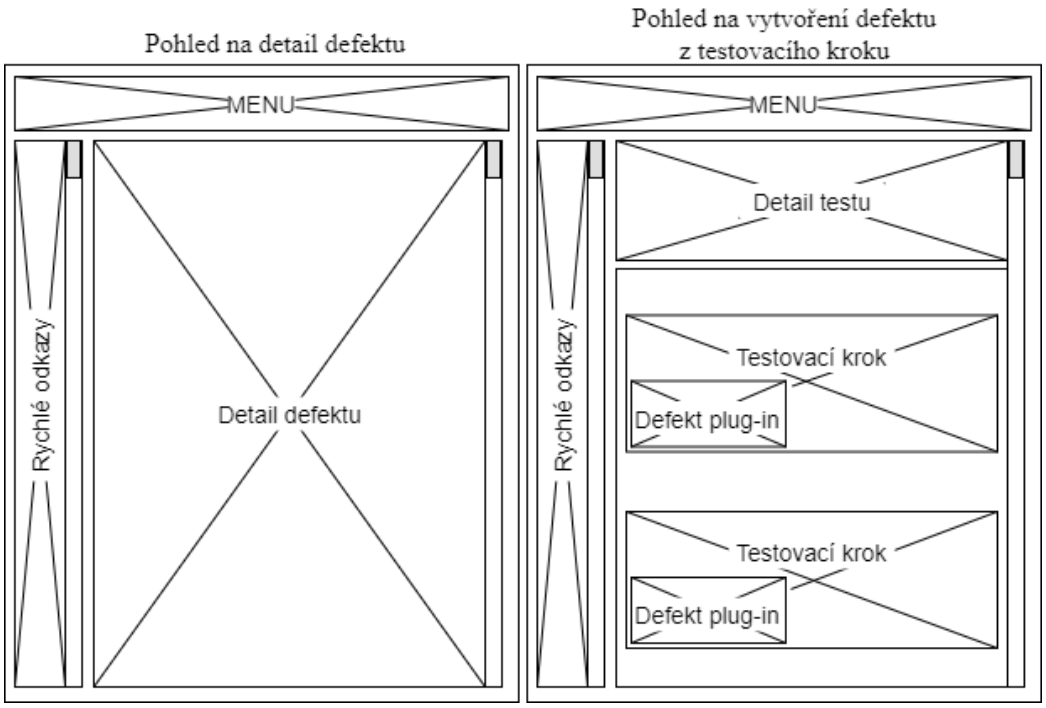
Stejně tak vznikne i jednotný proces pro práci s defekty bez ohledu na tým a roli, díky čemuž nebudou nastávat jakákoli nedorozumění a jiná chápání významů termínů jako je tomu v současné době u jiných objektů v Jira.

4.2.1.3 **Wireframe (návrhový vzor)**

Návrh vzhledu a přidání nového tlačítka byl proveden pomocí drátového modelu (wireframe). Wireframe je jednoduchý model, který definuje obsah, rozložení a funkce aplikace. Jedná se o architektonický návrh, ve kterém není potřeba používat barvy, obrázky ani žádný jiný obsah. Návrhový vzor byl vytvořen pro ujasnění rozvržení elementů a funkcí v systému pro přesnější představu zákazníkem.

Z pohledu frontendu je systém rozvrhnut do několika funkčních oken, jež se vztahují k samotné funkcionalitě systému. Celkem jsou navrhnuta čtyři okna dle vlastních funkcionalit. Základním oknem je *Pohled na detail defektu*. Dalšími okny jsou: P*ohled na vytvoření defektu z testovacího kroku, Pohled na vytvoření defektu z požadavku* a *Pohled na vytvoření defektu z tlačítka CREATE.*

Obrázek 14 Drátový model GUI systému – pohled na detail defektu a pohled na vytvoření defektu z testovacího kroku



Zdroj: vlastní zpracování

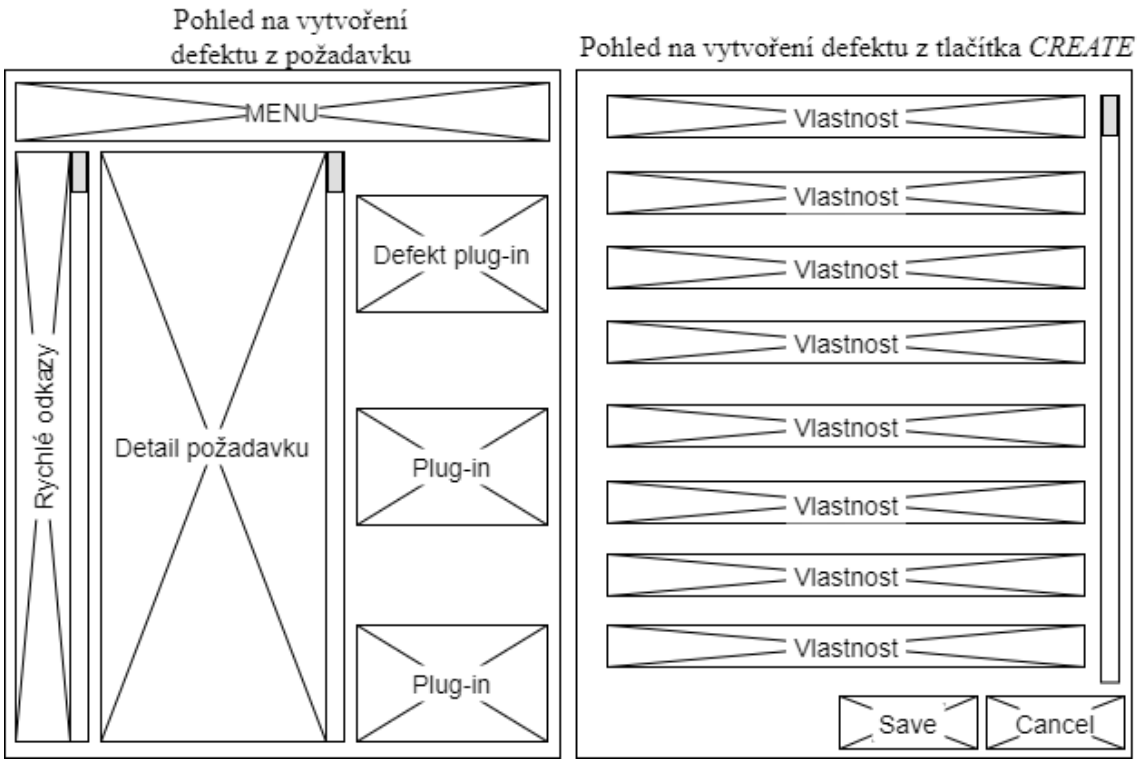
Na obrázku 14 je vyobrazeno okno *Pohled na detail defektu*, které se otevírá při spuštění aplikace Jira v projektu *Defect Management*. V tomto okně je možné prohlédnout si veškeré detaily k vybranému defektu. K detailům defektu také náleží vazba na jiný požadavek, přílohy a komentáře. Některá pole, která se zde nalézají, jsou zároveň i linkami, které umožňují rychlý a jednoduchý pohyb v aplikaci.

Horní menu je základním prvkem na každé obrazovce, pomocí něhož uživatel může překliknout na požadovaný projekt, nalézt hledaný požadavek, ukázat nástěnku nebo přejít k testům pomocí doplňku Xray.

V levé části je možné využít rychlé odkazy, mezi které patří například backlog, board nebo sprint.

Pro vytvoření defektu z testovacího kroku je nutné přejít do doplňku Xray. Okno *Pohled na vytvoření defektu z testovacího kroku* na obrázku 14 znázorňuje pohled na detail test exekuce s detaily jednotlivých testovacích kroků. Pro zefektivnění práce při testování bylo navrženo vytvoření tlačítka pro jednotlivé testovací kroky pro založení defektu z této obrazovky.

Obrázek 15 Drátový model GUI systému – pohled na vytvoření defektu z požadavku a pohled na vytvoření defektu z tlačítka CREATE



Zdroj: vlastní zpracování

Obrázek 15 zobrazuje okno *Pohled na vytvoření defektu s požadavkem*. Aby bylo dosaženo většího komfortu při práci, bylo i zde navrženo vytvoření tlačítka pro založení defektu z tohoto okna. Pod doplňkem pro založení defektu je možné přidat další doplňková okna pro rozšíření informací o daném defektu v tomto okně.

V levé části okna v tomto pohledu se také nachází možnost využití rychlých odkazů.

Pomocí drátového modelu bylo také upřesněno okno pro založení defektu za pomocí tlačítka *CREATE* na obrázku 15 (*Pohled na vytvoření defektu z tlačítka CREATE*), které se nachází v menu horní liště v každém okně. V této obrazovce uživatel vyplňuje atributy defektu dle požadavků, ze kterých defekt vznikl. Současně se ve spodní části obrazovky nachází tlačítko *Save* pro uložení a vytvoření defektu a tlačítko *Cancel* pro opuštění formuláře a vymazání zadaných hodnot do všech polí na této obrazovce.

4.2.2 **Definovaná severita defektu**

V České spořitelně byla jasně definována jednotná označení a pravidla pro severitu zadávaných defektů, vzniklých z testů:

1. **A – Fatal:** Případ užití pro většinu klientů nefunguje, v operaci nelze pokračovat ani jiným způsobem, resp. testovací případ nelze dokončit.

Příklady:

a. Nelze založit žádost o novou hypotéku.
b. Nelze vyhledat klienta.
c. Zobrazená výše zůstatku na účtu se řádově liší.

2. **B – Critical:** Případ užití nefunguje, ale je možné jej dokončit jiným (zpravidla složitějším a zdlouhavějším) způsobem, případně nefunguje pouze pro určitou velmi specifickou datovou variantu.

Příklady:

a. Lze založit žádost o novou hypotéku, ale zpracování je potřeba dokončit manuálně na backendové části aplikace.
b. Nelze založit hypotéku pro cizince, který je vdovec s více než čtyřmi dětmi.
c. Některé specifické klienty nelze vyhledat.
d. Zůstatek se pro cizince zobrazuje v nesprávné měně (avšak ve správné výši).

3. **C – Major:** V rámci případu užití nefunguje některá funkčnost. Operace může být pro většinu případů dokončena, ale v některých specifických případech nepracuje správně.

Příklady:

a. Hypotéku lze založit, ale pro klienta, který je cizinec a vdovec, je třeba zpracování dokončit manuálně.
b. Při vyhledávání klientů se v seznamu některé položky objeví dvakrát.
c. Zůstatek se v německé verzi průchodu aplikací zobrazuje v angličtině (částka je správná).

4. **D – Minor:** Kosmetické chyby, drobné nedostatky.

Příklady:

a. Ve smlouvě o hypotéce je logo v černobílé verzi a má být barevné.

b. Chyba v textu na obrazovce bankomatu.

c. Nezarovnané prvky na obrazovce.

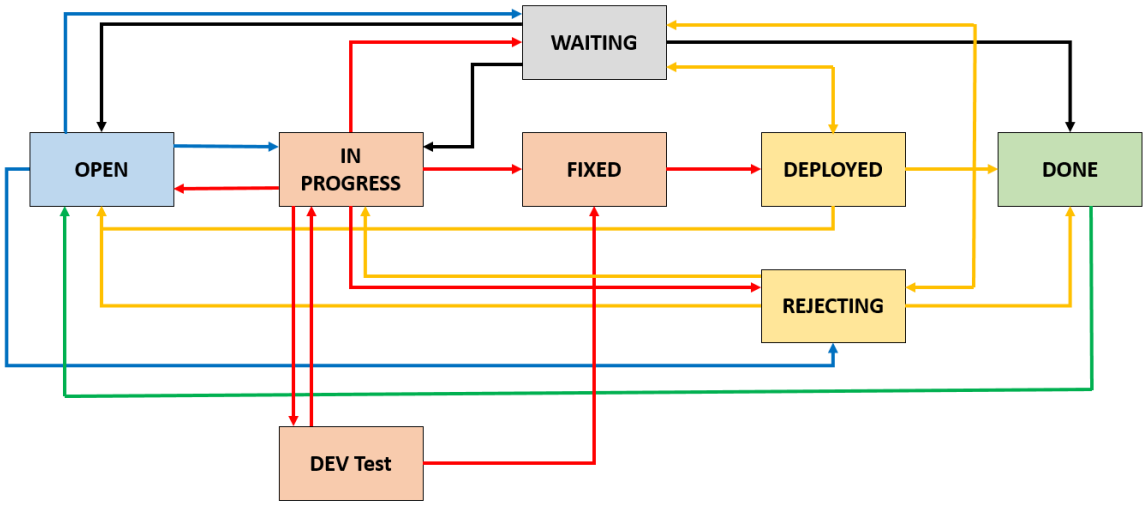
4.2.3 Nové workflow a stavy defektu v Jira

Pro vytvoření nového workflow defektu se vycházelo z několikaletého užívání flow v systému HP ALM a spojením flow požadavků v Jira.

Obdélníky v obrázku vymezují jednotlivé stavy v životním cyklu defektu. Stav charakterizuje situaci, kdy konfigurace defektu v systému je již ustálená. Změny stavu defektů jsou řízeny vnější vstupem neboli reakcí na vznik události (specifikace něčeho významného, co se stane v daném prostoru a čase).

Šipky pak znázorňují možnost přechodu mezi danými stavy defektu. Pokud mezi nějakými stavy šipky nejsou znázorněny, je tím znemožněn přechod defektu.

Obrázek 16 *Nové workflow v Jira*



Zdroj: vlastní zpracování

Popis jednotlivých stavů:

1. **OPEN –** Založení defektu. Defekt je přiřazen na aplikaci a čeká na převzetí týmem, který bude daný defekt řešit.
2. **IN PROGRESS** – Defekt byl převzat řešitelským týmem k řešení.
3. **FIXED –** Defekt je opraven a čeká se na nasazení opravy.
4. **DEPLOYED –** Defekt je opraven a nasazen. Zadavatel defektu má provést retestování testovacího kroku, v němž byl defekt nalezen.
5. **REJECTING –** Pokud je defekt odmítnut z jakéhokoli důvodu, nastaví odpovědná osoba *REJECTING* a zároveň musí vyplnit důvod, ze kterého byl defekt odmítnut.
6. **WAITING –** Defekt někdy potřebuje jiné řešení nebo se bude řešit později. Pro tyto účely lze využít stav *WAITING*, přičemž musí být vyplněn důvod čekání.
7. **DEV Test –** Tento stav by měl být ideálně využit pouze u produkčních defektů, u nichž musí proběhnout interní test před nasazením opravy do produkce. V případě potřeby je možné tento stav využít pro interní předtestování defektu. Stav *DEV Test* nenahrazuje stav *DEPLOYED*.
8. **DONE –** Stav *DONE* je jediným stavem, který označuje ukončení života defektu v jeho životním cyklu.

4.2.3.1 Praktický popis možných přechodů mezi stavy

1. **Ze stavu *OPEN* je možné přejít do stavů (modré šipky viz. obrázek 16):**

a. *IN PROGRESS* – Začíná se pracovat na opravě defektu.
b. *REJECTING* – Defekt se neuznává jako defekt a vyplňuje se důvod zamítnutí.
c. *WAITING* – Čeká se na další nezbytné prerekvizity. Defekt se odsouvá k pozdějšímu vyřešení a vyplňuje se důvod.

2. **Ze stavu *IN PROGRESS* je možné přejít do stavů (červené šipky vedoucí z tohoto stavu viz obrázek 16):**

a. *OPEN* – Defekt se vrací na začátek svého životního cyklu. Je možné změnit aplikaci a tím přeřadit defekt do jiného systému, resp. změnit řešitele vlastníka.
b. *FIXED* – Defekt je opraven a čeká se na nasazení opravy.

c. *WAITING* – Čeká se na další nezbytné prerekvizity. Defekt se odsouvá k pozdějšímu vyřešení a vyplňuje se důvod.
d. *REJECTING* – Defekt se není uznán jako defekt a vyplňuje se důvod zamítnutí.
e. *DEV Test* – Je nezbytné provést předtestování například produkčního defektu na vývojovém prostředí.

3. Ze stavu *FIXED* je možné přejít pouze do stavu (červená šipka vedoucí z tohoto stavu viz. obrázek 16):

a. *DEPLOYED* – Defekt je opraven, nasazen a je třeba provést retest.

4. Ze stavu *DEPLOYED* je možné přejít do stavů (žluté šipky vedoucí z tohoto stavu viz. obrázek 16):

a. *WAITING* – Čeká se na další nezbytné prerekvizity. Defekt se odsouvá k pozdějšímu řešení a vyplňuje se důvod.
b. *OPEN* – Defekt není opraven. Jedná se znovuotevření defektu („reopen“).
c. *DONE* – Defekt je vyřešen (opraven, nasazen, retestován). Uzavírá se životní cyklus defektu.

5. Ze stavu *REJECTING* je možné přejít do stavů (žluté šipky vedoucí z tohoto stavu viz. obrázek 16):

a. *WAITING* – Čeká se na další nezbytné prerekvizity. Defekt se odsouvá k pozdějšímu řešení a uvádí se důvod.
b. *OPEN* – Není uznáno odmítnutí defektu nebo jsou dodávány doplňující informace k defektu a dochází k jeho znovuotevření („reopen“).
c. *IN PROGRESS* – Není uznáno odmítnutí defektu nebo se dodávají doplňující informace a defekt se znovu otevírá, ale přiřazuje se přímo vývojářům, aniž by bylo nezbytné vyčkat na další zaplánování defektu k řešení.
d. *DONE* – Defekt je vyřešen (opraven, nasazen, retestován). Uzavírá se životní cyklus defektu.

6. **Ze stavu *WAITING* je možné přejít do stavů (černé šipky viz obrázek 16):**

a. *OPEN* – Řešení defektu bylo z jakéhokoli důvodu pozastaveno a nyní je možnost ho zařadit do backlogu.
b. *IN PROGRESS* – Defekt již není pozastaven a pracuje se na jeho opravě.
c. *DEPLOYED* – Již není nutno čekat, je možné defekt retestovat a uzavřít. Ve stavu *WAITING* byl například z toho důvodu, že byl blokován jinou chybou nebo se musely zajistit věci týkající se retesování.
d. *REJECTING* – Chyba již není blokována. Funkcionalita, na kterou byla vystavena, je zprovozněna, a proto je možné chybu odmítnout.
e. *DONE* – Defekt už není potřeba řešit (byl vyřešen v nové verzi) a uzavírá se jeho životní cyklus.

7. **Ze stavu *DEV Test* je možné přejít do stavů (červené šipky vedoucí z tohoto stavu viz obrázek 16):**

a. *IN PROGRESS* – Defekt není opraven, proto se vrací do stavu *IN PROGRESS.*
b. *FIXED* – Defekt je opraven a čeká se na nasazení opravy.

8. **Ze stavu *DONE* je možné přejít do stavu (zelená šipky viz obrázek 16):**

   a. *OPEN* – Defekt byl omylem uzavřen. Tímto přechodem je možné opravit administrativně uzavřený defekt.

4.2.4 **Domluvené (doporučené) časy při řešení defektů**

V bance byly domluveny časy pro převzetí, řešení a odbavení defektu zvlášť pro řešitelský tým a zvlášť pro testery:

Řešitelský tým:

Tabulka 6 Časování převzetí k řešení defektu

Severita	**Priorita**	
	0 - Blocker	**Ostatní**
A – Fatal	2h	4h
B – Critical	4h	6h
C – Major	-	12h
D – Minor	-	16h

Zdroj: vlastní zpracování

Převzetí k řešení defektu se počítá jako čas, po který je defekt ve stavu *OPEN* na stejném týmu.

Tabulka 7 Časování vyřešení defektu

Severita	**Priorita**	
	0 - Blocker	**Ostatní**
A – Fatal	8h	16h
B – Critical	16h	24h
C – Major	-	48h
D – Minor	-	64h

Zdroj: vlastní zpracování

Vyřešení defektu je čas přechodu mezi stavy *OPEN* a *DEPLOYED*, resp. čas, po který se defekt nachází ve stavech *IN PROGRESS* a *FIXED.*

Testeři:

Tabulka 8 Časování odbavení defektu

Severita	Priorita	
	0 - Blocker	**Ostatní**
A – Fatal	4h	8h
B – Critical	4h	8h
C – Major	-	16h
D – Minor	-	40h

Zdroj: vlastní zpracování

Odbavení defektu je čas, který defekt stráví ve stavech *REJECTING* nebo *DEPLOYED.*

4.2.5 **Metriky**

Metriky jsou využívané pro kontinuální zvyšování kvality verzí, protože by měly poukazovat na potenciálně problematická místa ať již z pohledu délky řešení nebo množství defektů.

Pro vytváření reportů (ohledně sledování práce na defektech a vývoje defektů v čase) je potřeba mít jasně stanovené metriky. Pro reportování jsou použity staré metriky, které je možné uzpůsobit novému systému, a současně jsou vytvářeny další metriky, které může pro sledování defektů nový systém Jira nabídnout.

Aby bylo možné reporty, vyžadované vedením projektu, vytvářet, bylo zapotřebí některá pole uvést jako povinná již při zakládání defektu, aby se zamezilo jejich nevyplňování, protože jsou klíčové pro poskytnutí přehledové zprávy o defektech.

V tabulce 9 je uveden příklad metrik, které se v bance sledují:

Tabulka 9 Příklad sledovaných metrik v České spořitelně

ID	Metrika
M-01	Počet defektů a jejich severita a priorita
M-02	Časové trendy mezi stavy
M-03	Příčina zamítnutí defektu
M-04	Porušování workflow

ID	Metrika
M-05	Změna severity
M-06	Počet znovuotevřených defektů
M-07	Počet neuzavřených defektů, jež jsou nasazeny do produkce s releasovou verzí

Zdroj: vlastní zpracování

- **M-01 Počet defektů a jejich severita a priorita** – Tento report vykazuje, kolik vzniklo v určitém období v bance defektů bez ohledu na jejich povahu (z testů, produkčních defektů apod.) a zároveň prezentuje i jejich severitu a prioritu. Současně se z těchto dat tvoří i trendový graf vývoje nalezených defektů za jednotlivá sledovaná období. Je díky tomu možné vytvořit vzájemná porovnání období a vyhodnotit, které z nich bylo z hlediska četnosti chyb nejvíce rizikovým.
- **M-02 Časové trendy mezi stavy –** Report vyjadřuje, jak defekt v průměru postupoval v čase mezi jednotlivými stavy. Časy mezi stavy se měří v době stanoveného pracovního okna od 8.30 do 17.00 hodin. Mezi sledované průchody patří:
    a. Čas ze stavu *OPEN* do *DONE* – Vyjadřuje, jak dlouho trvalo defektu projít životní cyklus od jeho založení po jeho zánik.
    b. Čas ze stavu *OPEN* do *IN PROGRESS* – Časový údaj ukazuje, jak dlouho trvalo než si defekt, který byl vytvořen na konkrétní aplikaci, převzal zodpovědný tým.
    c. Čas ze stavu *IN PROGRESS* do *FIXED* – Tento časový přechod prezentuje „práci vývojáře“, resp. jak dlouho trvalo, než se daný defekt opravil.
    d. Čas ze stavu *FIXED* do *DEPLOYED* – Pomocí tohoto ukazatele je možné vyhodnotit časový úsek od opravení defektu do jeho nasazení.
    e. Čas ze stavu *DEPLOYED* do *ANY* (*OPEN* nebo *DONE*) – Jedná se o „práci testera“ neboli jak dlouho trvalo, než se otestovala daná funkcionalita, kterou znemožňoval zadaný defekt.
    f. Čas ze stavu *REJECTING* do *ANY* (*OPEN* nebo *DONE*) – Ukazuje, jak dlouho trvalo zadavateli se věnovat zamítnutému defektu – např. upravit zadání nebo se již nepodařilo chybu nasimulovat.

- **M-03 Příčina zamítnutí defektu** – Metrika prezentuje, jaké jsou nejčastější důvody zamítnutí defektu. Při bližší analýze je možné zjistit, jaké byly příčiny zamítnutí defektu a skutečnost napravit například správným proškolením testerů. Zároveň je vytvářen časový trend, umožňující porovnání zamítnutých defektů v jednotlivých sledovaných obdobích.
- **M-04 Porušování workflow –** Je možné zjistit, který uživatel nebo tým záměrně porušuje stanovené jednotné workflow defektů pro celou banku.
- **M-05 Změna severity –** Vysoký nárůst změny severity ve sledovaném období značí špatné chápání tohoto atributu na straně vývojáře případně testera.
- **M-06 Počet znovuotevřených defektů** – Je třeba věnovat pozornost defektům, které se znovuotevírají k řešení. Pokud počet znovuotevřených defektů vzroste, vzniká potřeba provést podrobnější analýzu a zajistit nápravu, neboť může dojít ke zpomalování práce**.**
- **M-07 Počet neuzavřených defektů, jež jsou nasazeny s releasovou verzí –** Metrika sleduje, jaký je počet neopravených defektů, které byly nasazeny s verzí do produkce. Umožňuje mít přehled o možných rizicích na produkci, které mohou ohrozit klienta. Zároveň se může projevovat nedisciplinovanost uživatelů, kteří administrativně neuzavřou defekt v systému. Tato skutečnost vede ke znehodnocování čistoty dat v systému pro defekt management.

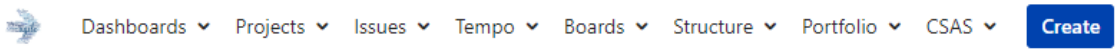
4.3 Defekt v prostředí Jira

Pro zjednodušení veškeré práce se zakládáním defektu byly definovány tři možnosti jeho založení:

1. **Založení defektu pomocí tlačítka *CREATE***

Tlačítko *CREATE* je nativní funkcí Jira a nachází se v horní liště. Umožňuje zakládání veškerých možných entit v tomto systému, a proto ani defekt není výjimkou.

Obrázek 17 Horní lišta Jira s tlačítkem CREATE



Zdroj: vlastní zpracování

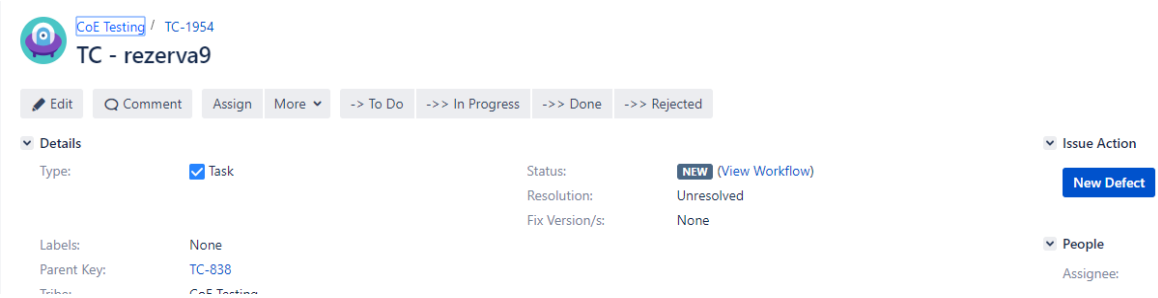
Tato varianta zakládání defektu je doporučována k používání minimálně, jelikož při jejím využití nevznikne žádná vazba na jiný požadavek, resp. nedojde k automatickému prolinkování s entitami Epic, User Story, Task apod. a uživatel již nevidí, ke kterému požadavku defekt patří. Avšak je tuto variantu možné využít například pro tým aplikační podpory, která nezakládá defekty z požadavků.

2. **Založení defektu pomocí interně vyvinutého tlačítka *New Defect* z libovolné podporované entity**

Pro zjednodušené vytvoření defektu bylo interně vyvinuto tlačítko *New Defect*, které je zobrazeno na všech entitách, na kterých je možné defekt vystavit. Defekt nemůže být vystaven na entitu Iniciativa a Sub Task, tudíž zde ani není zobrazeno toto tlačítko.

Pokud je využito vytvoření defektu pomocí *New Defect* tlačítka, dojde k automatickému vytvoření vazby na entitu, ze které je tlačítko použito. Zároveň je automaticky zařazen do projektu pro všechny defekty – *Defect Management*. Tato varianta zakládání defektu je doporučována z důvodu čistoty dat.

Obrázek 18 *Interně vyvinuté tlačítko New Defect na Tasku*



Zdroj: vlastní zpracování

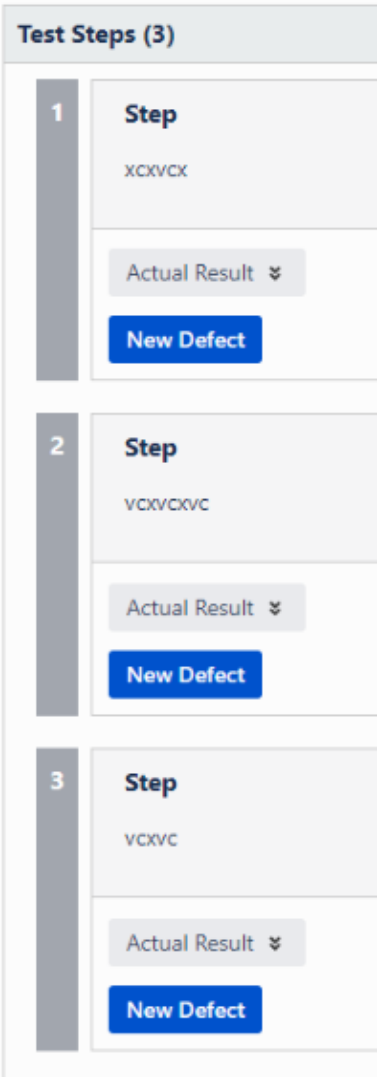
3. **Založení defektu z testovacího kroku pomocí interně vyvinutého tlačítka *New Defect***

Uživatel může pracovat v doplňku Jira v Xray. Vzhledem k této skutečnosti bylo toto interně vyvinuté tlačítko umístěno přímo na jednotlivé testovací kroky. Uživatel tedy může vytvořit defekt přímo na konkrétní testovací krok, v němž chybu nalezl.

Tato varianta je také doporučena v případě, že uživatel pracuje v doplňku Xray. Stejně jako ve variantě 2. zde vznikne vazba přímo s testem a případně i s entitou, pod kterou je daný test zařazen.

Současně je zde výhodou předvyplnění některých polí, které se zobrazí při zakládání defektu. Do pole *Description* se i předvyplní testovací krok, ve kterém defekt byl nalezen. Tuto skutečnost předvyplněných polí lze samozřejmě libovolně upravovat.

Obrázek 19 *Interně vyvinuté tlačítko New Defect na testovacím kroku*

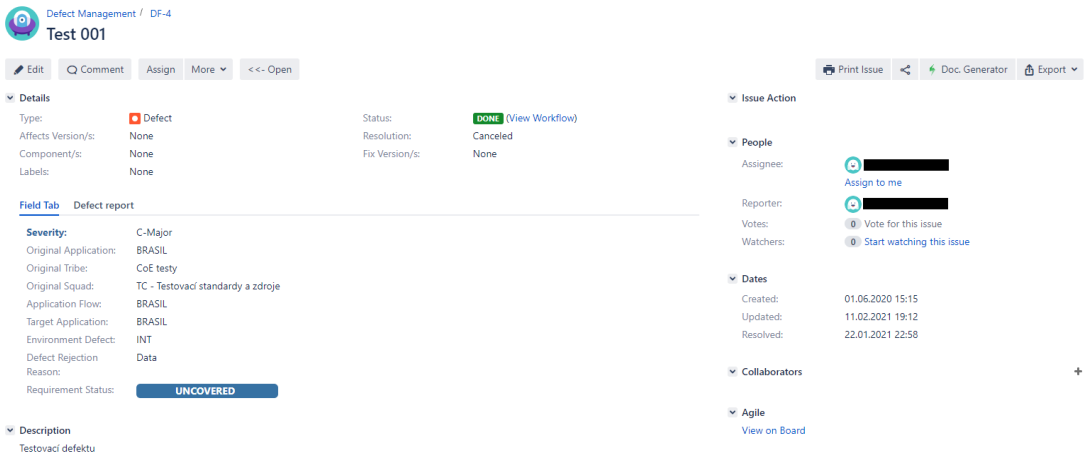


Zdroj: vlastní zpracování

4.3.1 **Pole u defektu**

Při zakládání defektu je potřeba naplnit určitá pole hodnotou. Pole byla rozdělena do tří kategorií: na povinná, nepovinná a ostatní, s nimiž se uživatel může v průběhu života defektu setkat. Povinná pole byla stanovena podle potřeby, kterou definuje Jira pro založení defektu. Zároveň další pole byla nově označena povinně u entity *Defect* z důvodu možnosti reportingu. Některá pole jsou pro potřeby typu „combobox“ – možnost vybírat hodnoty z připojeného seznamu, a multivalue – možnost vyplnění více hodnot.

Obrázek 20 Ukázka polí u založeného defektu



Zdroj: vlastní zpracování

4.3.1.1 **Povinná pole při zakládání defektu**

Povinná pole při zakládání defektu jsou pro lehčí orientaci uživatele označena červenou hvězdičkou.

1. **Project** – Toto pole musí být vždy naplněno hodnotou „*Defect Management*“, jinak není možné defekt založit.
2. **Issue Type** – Pro defekt je nutné v tomto poli mít vyplněnou hodnotu „*Defect*“.
3. **Original Tribe** – Jedná se o pole typu combobox a vyplňuje se hodnota tribu, odkud se defekt zadává.
4. **Original Squad** – Jedná se o pole typu combobox a vyplňuje se hodnota squadu, odkud se defekt zadává. Hodnoty v comboboxu jsou filtrované na základě vybrané hodnoty v poli *Original Tribe*.

5. **Original Application** – Pole začne samo nabízet hodnoty z databáze aplikací banky, jakmile do něj uživatel začne psát.
6. **Affects Version/s** – Jedná se o multivalue pole typu combobox, kde uživatel vybírá hodnotu releasu, ve kterém defekt nalezl (jedná se o hodnotu systému, který defekt nalezl). Pole přebírá hodnoty z projektů v poli *Fix Version*.
7. **Target Application** – Pole začne samo nabízet hodnoty, jakmile do něj uživatel začne psát. Toto pole určuje, zda se jedná o integrační defekt. Pokud se nejedná o integrační defekt, uživatel zadává stejnou hodnotu jako v poli *Original Application.*
8. **Summary** – Textové pole, které vymezuje krátký název defektu.
9. **Description** – Textové pole, ve kterém uživatel vyplňuje popis defektu.
10. **Environment Defect** – Jedná se o multivalue pole, kde je možné vybrat z hodnot PROD, PREPROD, PRS, INT, INT2, EDU, DEV podle toho, v jakém prostředí byl defekt nalezen.
11. **Severity** – Jedná se o pole typu combobox s hodnotami A-Fatal, B-Critical, C-Major, D-Minor.
12. **Priority** – Jedná se o pole typu combobox s hodnotami 0-Blocker, A-High, B-Medium, C- Low, D-Lowest.

4.3.1.2 Nepovinná pole při zakládání defektu

1. **Tribe** – Jedná se o pole typu combobox, kde uživatel vyplní, který tribe defekt opravuje.
2. **Squad** – Jedná se o pole typu combobox, kde uživatel vyplní squad, který defekt opravuje. Hodnoty v comboboxu jsou filtrované na základě vybrané hodnoty v poli *Tribe*.
3. **Fix Version/s** – Jedná se o multivalue pole typu combobox, kde uživatel vyplní release, ve kterém bude defekt opraven. Toto pole je důležité zejména pro defekty, které se nestihnou opravit v releasu, v němž je defekt nalezen.
4. **Component/s** – Jedná se o multivalue pole typu combobox, které může být využito pro moduly aplikace a jiné určení defektu. Hodnoty v comboboxu jsou filtrované na základě vybrané hodnoty v poli *Original Tribe*.

5. **Story Points** – Pole pro zaznamenávání odhadu. Využívané jen v některých squadech pro zaznamenávání odhadu objemu práce. Nemá na nic vliv a při předání na jinou aplikaci se hodnoty automaticky odstraní.
6. **Labels** – Klasické pole pro Jira pro potřebu „oštítkování“ defektu a zařazení pod libovolnou kategorii z důvodu lepšího filtrování.
7. **Attachment** – Pole, kam uživatel může přidávat libovolné přílohy, které jsou pro defekt důležité.
8. **Sprint** – Pole, kde uživatel může vyplnit, ve kterém sprintu s defektem bude pracovat.
9. **Technical Version Name** – Textové pole, kde uživatel může vyplnit technickou verzi svého tribu.
10. **ServiceNow ID** – Pole pro zaznamenání ID produkčního defektu. Pole se stává linkem do ServiceNow, což je systém pro zaznamenávání produkčních defektů, incidentů a řízení releasů.
11. **Required Date** – Jedná se o pole s datem, které určuje nasazení opraveného defektu do testu.

4.3.1.3 Ostatní pole, se kterými se uživatel může setkat v průběhu života defektu

1. **Asignee** – Jedná se o pole, kde uživatel přiřazuje řešitele defektu. Pole se při přechodu do stavu *IN PROGRESS* stává povinným.
2. **Application Flow** – Pole se naplňuje samo. Jedná se o pole, kam se postupně zapisují všechny aplikace, přes které defekt prošel, resp. se zaznamenává historie.
3. **Comment** – Standardní Jira pole, kam uživatel může napsat komentáře k defektu.
4. **Reporter** – Standardní Jira pole, kde uživatel vidí zadavatele defektu.
5. **Rejected to OPEN** – Pole, které se vyplní automaticky početní hodnotou znovuotevřeného defektu.
6. **Deployed to OPEN** – Pole, které se vyplní automaticky početní hodnotou znovuotevřeného defektu.
7. **Reason for Waiting** – Jedná se o textové pole, kam uživatel uvádí důvod posunu defektu do stavu *WAITING*.
8. **Resolution** – Pole, ve kterém je u defektu celý jeho životní cyklus hodnota *Unresolved*. Tato hodnota se změní při přechodu do stavu *DONE* z jakéhokoli jiného

stavu než ze stavu *REJECTING* na hodnotu *Resolved.* V případě, že uživatel přepíná defekt ze stavu *REJECTING* do stavu *DONE*, vyplní se pole hodnotou *Canceled.*

9. **Defect Rejection Reason** – Jedná se o pole typu combobox, kde uživatel vybírá důvod zamítnutí z níže uvedených hodnot:
    a. **Test (Correct Behaviour) -** Byl špatně napsán nebo proveden testovací případ. Ve skutečnosti se tedy jedná o chybu testera, nikoli o chybu aplikace.
    b. **Data** – Tester měl chybná testovací data.
    c. **Cannot Reproduce** – Chybu nelze na straně vývojáře nasimulovat, tím pádem není možné určit její původ.
    d. **Insufficient Description** – Záznam defektu neobsahuje dohodnuté informace, na základě kterých lze efektivně zahájit analýzu chyby.
    e. **Outage** – Chyba byla způsobena nedostupným testovacím prostředím (plánovaná odstávka, neplánovaný výpadek, havárie…).
    f. **Duplicity** – Stejné selhání aplikace bylo zadáno dvěma defekty. Druhý se tedy zavře jako duplicita a řeší se pouze první defekt.
    g. **Won't fix** – Nebude se na hledání příčiny dále pracovat. Defekt bude uzavřen např. z důvodu neefektivity, resp. malé pravděpodobnosti výskytu chyby na produkčním prostředí.

4.4 Testování a zhodnocení nového defekt managementu

V této kapitole je rozebíráno akceptační testování nového defekt managementu vybranou skupinou uživatelů. Současně jsou zde uvedeny nově vzniklé požadavky ze strany uživatelů, kteří s defektem denně pracují. Následuje vyhodnocení dotazníku, který byl předán vybraným uživatelům k vyplnění.

4.4.1 Akceptační testování

Akceptačního testování se zúčastnily průběžně osoby, které jsou zodpovědné za opravu defektů v jednotlivých tribech. Zároveň tyto osoby mají několikaleté zkušenosti v oblasti testování i s prací v systému Jira, tudíž jsou jejich názory podloženy i jejich pracovními zkušenostmi.

Před samotným testování byl účastníkům vysvětlen postup zakládání defektů a jeho životní cyklus se stavy a přechody. Následně jim byla i poskytnuta dokumentace, v níž se nachází veškerá metodika, týkající se nového defekt managementu.

Pro akceptační testování jsou připraveny 3 testovací scénáře, které navádí uživatele k práci se samotným zakládáním defektu.

Tabulka 10 Testovací scénář 1 – založení defektu pomocí tlačítka CREATE

ID	**Testovací krok**	**Očekávaný výsledek**
1	Přihlaste se do aplikace Jira	Přihlášení proběhlo v pořádku.
2	Spusťte založení požadavku pomocí tlačítka *CREATE* v pravé části horního menu.	Po stisku tlačítka se zobrazí okno pro založení požadavku.
3	V horní části obrazovky v poli *Project* vyberte hodnotu *Defect Management*	Hodnota je zobrazena a je možné ji zvolit.
4	Následující pole vyplňte hodnotami uvedenými níže: Original Application = SB Original Tribe = CoE Testy Original Squad = TC – Testing Delivery Affects Version/s = 2021 03 Target Application = SB Summary = Test01 Description = Uživatelský test1 Environment Defect = INT2 Severita = B-Critical Priority = B-Medium	U polí byla umožněna editace a hodnoty byly zapsány dle popisu kroku.
5	Stiskněte tlačítko *Create.*	Defekt byl vytvořen.

Zdroj: vlastní zpracování

Tabulka 11 Testovací scénář 2 – Založení defektu pomocí interně vyvinutého tlačítka New Defect z libovolné podporované entity

ID	Testovací krok	Očekávaný výsledek
1	Přihlaste se do aplikace Jira	Přihlášení proběhlo v pořádku.
2	Vyhledejte libovolné požadavek ze skupiny Story, Epic, Task nebo T-Task	Požadavek byl vyhledán a otevřen.
3	V pravé části obrazovky uprostřed klikněte na tlačítko *New Defect*.	Zobrazí se okno s poli k vyplnění na založení defektu.
4	Následující pole vyplňte hodnotami uvedenými níže: Original Application = SB Original Tribe = CoE Testy Original Squad = TC – Testing Delivery Affects Version/s = 2021 03 Target Application = SB Summary = Test02 Description = Uživatelský test2 Environment Defect = INT2 Severita = B-Critical Priority = B-Medium	U polí byla umožněna editace a hodnoty byly zapsány dle popisu kroku.
5	Stiskněte tlačítko *Save.*	Defekt byl vytvořen.

Zdroj: vlastní zpracování

Tabulka 12 Testovací scénář 3 – Založení defektu z testovacího kroku pomocí interně vyvinutého tlačítka New Defect

ID	Testovací krok	Očekávaný výsledek
1	Přihlaste se do aplikace Jira	Přihlášení proběhlo v pořádku.
2	Vyhledejte požadavek typu Test Execution.	Požadavek byl vyhledán a otevřen.
3	V části obrazovky s testovacími kroky u libovolného testovacího kroku stiskněte tlačítko *New Defect*.	Zobrazí se okno s poli k vyplnění na založení defektu.
4	Následující pole vyplňte hodnotami uvedenými níže: Original Application = SB Original Tribe = CoE Testy Original Squad = TC – Testing Delivery Affects Version/s = 2021 03 Target Application = SB Summary = Test03 Description = Uživatelský test3 Environment Defect = INT2 Severita = B-Critical Priority = B-Medium	U polí byla umožněna editace a hodnoty byly zapsány dle popisu kroku.
5	Stiskněte tlačítko *Save.*	Defekt byl vytvořen.

Zdroj: vlastní zpracování

4.4.2 Požadavky na zlepšení nového defekt managementu

Níže je uveden výběr a popis požadavků (pojednávaných v předešlých kapitolách) na defekt management, které navrhli uživatelé v rámci testů a pilotního ověřování nového defekt managementu:

Tabulka 13 Vybrané požadavky na zlepšení práce s defektem od uživatelů

ID	Název
P-01	Multivalue pole *Components*
P-02	Pole Labels při zakládání defektu

ID	Název
P-03	Výmaz pole *Tribe* a *Squad* při přechodu do stavu REJECTING
P-03	Výmaz pole *Tribe* a *Squad* při přechodu do stavu REJECTING
P-04	Formátování textu při zakládání defektu
P-05	Předvyplnit pole *Tribe* a *Squad* při zakládání defektu
P-06	Rozšíření číselníku v poli *Rejecting*
P-07	Přidání příloh při zakládání defektu
P-08	Úprava notifikace
P-09	Zrušení povinnosti vyplňovat pole *Assignee* při přechodu do stavu *FIXED* nebo *DEV Test*
P-10	Přidat pole pro očekávané vyřešení defektu
P-11	Výmaz pole Sprint při přechodu do stavu *OPEN*

Zdroj: vlastní zpracování

- **P-01 Multivalue pole *Components* –** Pole je typu combobox, přičemž nabízené hodnoty se filtrují na základě vybrané hodnoty v poli *Original Tribe*. Vznikl požadavek od uživatelů o rozšíření o možnost do tohoto pole zadat více než jednu hodnotu, jelikož jeden defekt může mít vliv na více komponent systému.
- **P-02 Pole *Labels* při zakládání defektu** – Uživatelé by rádi měli možnost přidat si label neboli štítek k defektu již při jeho zadávání. Než tento požadavek byl vyřešen, uživatel musel defekt založit a poté v editovacím módu label přidat.
- **P-03 Výmaz pole *Tribe* a *Squad* při přechodu do stavu *REJECTING* –** Tento požadavek byl mířený pouze na integrační defekty. Pokud je vystaven defekt a přejde do cílové aplikace, měl by se po změně stavu na *REJECTING* vrátit zpět na zakladatele defektu, aby ho měl možnost spatřit ve svém filtru na nástěnce a mohl ho přiřadit do správných tribe a squad.
- **P-04 Formátování textu při zakládání defektu** – Pro větší komfort uživatelů a zároveň zrychlení práce při zakládání defektu je požadována možnost formátovat text již při zakládání defektu. Tato možnost je v současné době dostupná uživatelům, pouze pokud defekt zakládají pomocí tlačítka *CREATE*, jelikož se jedná o nativní funkci systému Jira. Naopak tlačítko *New Defect* je interně vyvinuté a tato možnost zde chybí. Z důvodu zastaralé verze Jira, kterou již dodavatel nepodporuje,

je požadavek zaparkován do doby, než se přejde na novou verzi (upgrade je plánován na duben 2021).

- **P-05 Předvyplnit pole *Tribe* a *Squad* při zakládání defektu –** Uživatelé by rádi při zakládání měli předvyplněná základní pole dle jejich příslušnosti k danému tribu a squadu. Požadavek byl vyřešen pomocí personalizace nastavení jednotlivých uživatelů. Uživatel si tedy v nastavení vlastního profilu může tyto informace nastavit a při zakládání defektu mu budou hodnoty automaticky vyplněny.
- **P-06 Rozšíření číselníku v poli *Rejecting* –** Je potřeba přidat hodnotu plánované odstávky do pole *Rejecting.* Mezi hodnoty byla teda zařazena i hodnota *Outage*.
- **P-07 Přidání příloh při zakládání defektu –** Možnost přidávat přílohy přímo při zakládání defektu by zrychlilo a usnadnilo práci. Stejně jako v požadavku P-04 je tato akce umožněna pouze při zakládání defektu přes tlačítko *CREATE*. Momentálně je požadavek také zaparkován z důvodu čekání na novou verzi systému. Poté se požadavek otevře a bude vytvořena analýza pro přidání možnosti pro tlačítko *New Defect.*
- **P-08 Úprava notifikace –** Notifikace o jakýchkoliv změnách na defektu dostávají zakladatelé defektu nebo lidé, kteří si nastaví sledování daného defektu. Funkce notifikací je nativní funkcí Jira, ale přesto by ji uživatelé uvítali lépe strukturovanou a s více informacemi. Byla tedy provedena aktualizace a stále je požadavek sledován, zda se vyskytla potřeba přidat/ubrat nebo upravit informace.
- **P-09 Zrušení povinnosti vyplňovat pole *Assignee* při přechodu do stavu *FIXED* –** Pole v tomto přechodu stále zůstává, ale bylo označené jako nepovinné pole. Přechod do stavu *FIXED* je možné pouze ze stavů *IN PROGRESS* a *DEV Test*. Z metodického hlediska není potřeba mít při těchto přechodech povinné pole *Assignee.*
- **P-10 Přidat pole pro očekávané vyřešení defektu** – Mezi veškerými informacemi o defektu chybělo pole *Required Date*. Očekávané vyřešení defektu bylo možné zapisovat do pole *Description*, kde se však údaj mohl snadno ztratit mezi dalšími informacemi.
- **P-11 Výmaz pole Sprint při přechodu do stavu *OPEN* –** Pokud defekt přejde zpět do stavu *OPEN*, bylo požádáno o výmaz pole *Sprint*, aby nový řešitel defektu mohl defekt naplánovat do vlastního sprintu, ve kterém bude řešen.

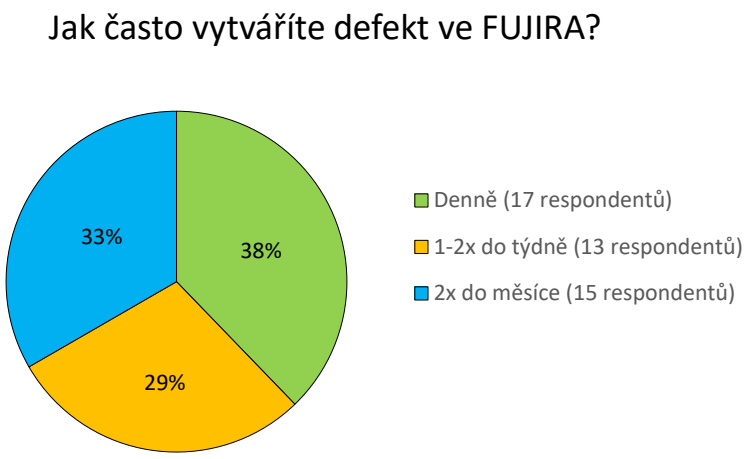
4.4.3 **Dotazník**

Účelem dotazníkového šetření je zjistit, jak uživatel vnímá nový defekt management v novém nástroji. Dotazník dovoluje uživateli vyjádřit anonymně svůj názor. Poslední otázkou byla respondentovi mimo jiné nabídnuta odpověď v podobě vlastního textu. Díky dotazníku je vedení CoE Testy poskytnuta zpětná vazba od uživatelů defekt managementu v systému Jira. Vyhodnocení tohoto dotazníku bude podkladem pro zlepšení defekt managementu.

Dotazník obsahuje 6 otázek a byl vytvořen pomocí aplikace Google Forms (viz příloha E a F). Následné zpracování dotazníku bylo provedeno pomocí softwaru Microsoft Excel. Dotazník vyplnilo 45 respondentů.

1. Otázka – „*Jak často vytváříte defekt ve FUJIRA?*“

Graf 1 Otázka „Jak často vytváříte defekt ve FUJIRA?“

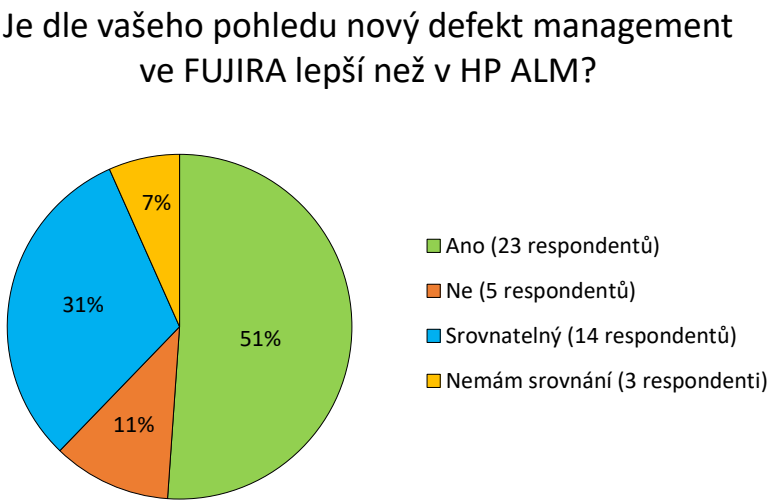


Zdroj: vlastní zpracování

V této otázce bylo zjišťováno, jak často uživatelé zakládají defekty. Vzhledem k povaze dat je zřejmé, že 30 respondentů zakládá defekt velice často. Zbylých 15 respondentů zakládá znatelně méně defektů.

2. **Otázka – „*Je dle vašeho pohledu nový defekt management ve FUJIRA lepší než v HP ALM?*“**

Graf 2 Otázka „Je dle vašeho pohledu nový defekt management ve FUJIRA lepší než v HP ALM?“



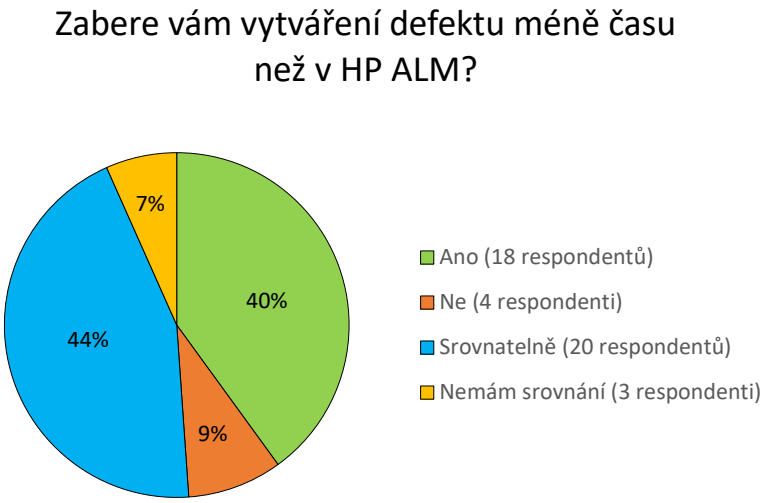
Zdroj: vlastní zpracování

Z grafu 2 lze vyčíst, že 51 % respondentů hodnotí defekt management v novém systému FUJIRA jako pozitivní změnu oproti předcházejícímu nástroji HP ALM. Celkem 31 % respondentů zastává názor, že defekt management v novém nástroji je srovnatelný.

Vzhledem k převládajícím pozitivním odpovědím je možné konstatovat, že převod defekt managementu do nového nástroje je pro společnost krokem pozitivním.

3. Otázka *„Zabere vám vytváření defektu méně času než v HP ALM?“*

Graf 3 Otázka „Zabere vám vytváření defektu méně času než v HP ALM?“

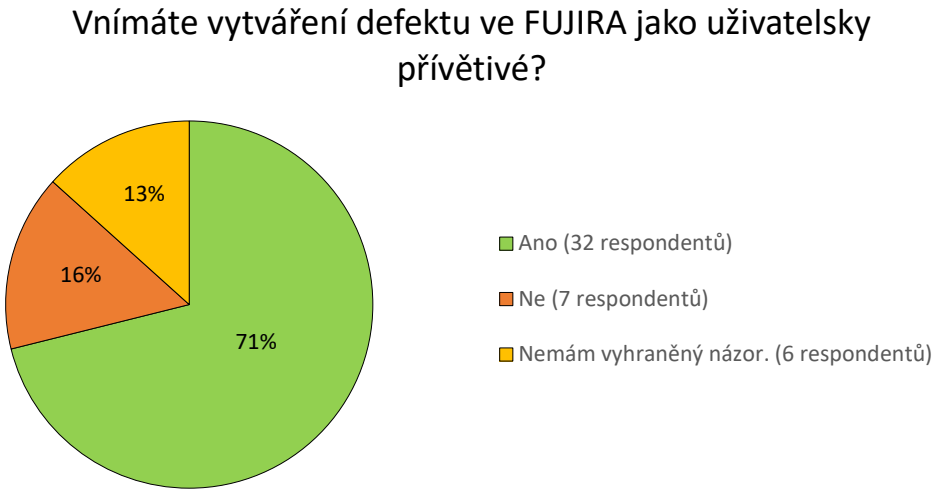


Zdroj: vlastní zpracování

Graf 3 zobrazuje, jak uživatelé vnímají časovou náročnost zakládání defektu v novém nástroji ve srovnání se starým. Zobrazení na grafu znázorňuje důležitý poznatek, a sice že 88 % respondentů nevnímá negativně časovou náročnost zakládání defektu v novém nástroji. Skupina uživatelů, která nemá srovnání, jsou například zaměstnanci, kteří neměli možnost se se systémem HP ALM setkat v současné práci či v jejich předešlé praxi.

4. Otázka „*Vnímáte vytváření defektu ve FUJIRA jako uživatelský přívětivé?*“

Graf 4 Otázka „Vnímáte vytváření defektu ve FUJIRA jako uživatelsky přívětivé?“



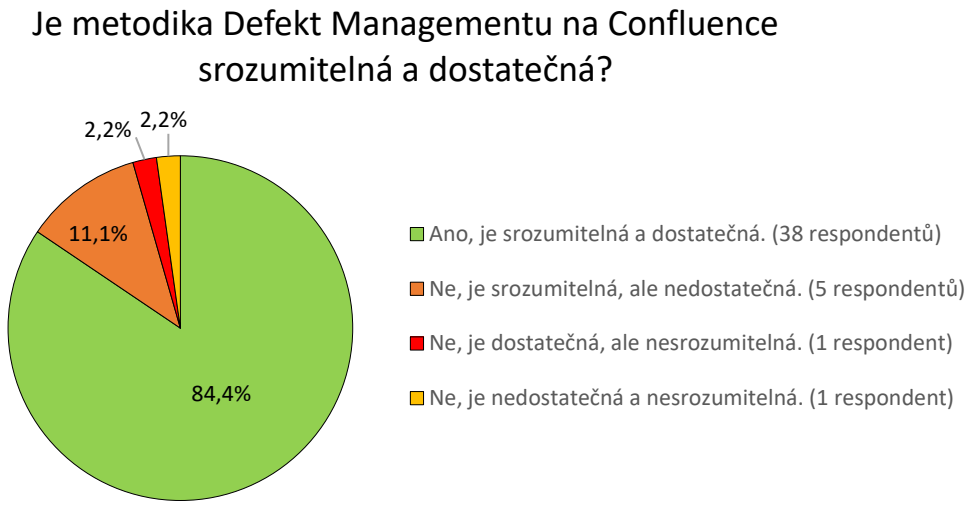
Zdroj: vlastní zpracování

Procentuální rozdělení v grafu 4 dle odpovědí uživatelů vyjadřuje, že 71 % respondentů, kteří vytváří defekt v novém nástroji, je spokojeno se vzhledem a chováním systému Jira při zakládání nového defektu. Zbylých 29 % se dělí na respondenty, kteří nemají vyhraněný názor (13 %), nebo jim zakládání defektu přijde uživatelsky nepřívětivé (16 %).

Jelikož 71 % respondentů je spokojeno se zadáváním defektu, je zřejmé, že technické nastavení zadávání defektu, které bylo provedeno na základě metodiky, je s ohledem na uživatelský komfort vytvořené správně.

5. **Otázka – *„Je metodika defekt managementu na Confluence srozumitelná a dostatečná?“***

Graf 5 Otázka „Je metodika defekt managementu na Confluence srozumitelná a dostatečná?“



Zdroj: vlastní zpracování

Confluence je nástroj, který se v České spořitelně používá pro evidenci jednotlivých dokumentací. Dokumentace k defekt managementu je také uložena v tomto nástroji, proto byla uživatelům položena otázka, do jaké míry jsou se zpracováním dokumentace k defekt managementu v nástroji Jira spokojeni.

Z šetření vyplývá, že 84,4 % (38) respondentů je spokojeno s vytvořenou dokumentací pro metodiku defekt managementu. Zbylých 15,6 % respondentů spatřuje v dokumentaci nějaké nedostatky, týkající se její srozumitelnosti nebo objemu informací.

6. Otázka – „*Co by vám pomohlo ve zlepšení práce s defektem?*“

Graf 6 Otázka „Co by vám pomohlo ve zlepšení práce s defektem?“

Zdroj: vlastní zpracování

V poslední otázce měli respondenti možnost vyjádřit se k tomu, co by potřebovali, aby se jejich práce s defektem zlepšila. Z grafu 6 vyplývá, že 64 % respondentů má vše co potřebuje, jen se musí zdokonalit v práci s defektem.

Naopak 36 % (16) respondentů ke zlepšení práce něco chybí. Odpovědi, týkající se skutečností, které by respondenti vylepšili, nebo vyjádření, co jim chybí, jsou uvedeny v tabulce 14 níže.

Tabulka 14 Individuální odpovědi na otázku „Co by vám pomohlo ve zlepšení práce s defektem?“

ID	Odpověď
1	Možnost zadání defektu přímo z instance "sub-exekuce".
2	Předvyplnění polí, co jdou předvyplnit – např. tribe, squad, prostředí atp. (protože to ve více jak 80 % případů vyplňuji vždy stejně). Možnost přiložit screenshot a přiřadit defekt na člena týmu hned z X-Ray modálu, abych pak nemusel chytat oznámení o vytvoření a dělat tyto změny později. Formátování texty popisu defektu v X-Ray modálu stejné jako v defektu (např. abych úryvky kódu mohl vložit do bloku hned, a ne až z detailu defektu a aby se to lépe četlo). Zjednodušit proces nalezení cílového squadu/tribu – je to úporné tipovat, na koho tak zhruba defekt vystavit.

ID	Odpověď
3	Ideálně bezchybné implementace, aby defekt nebylo třeba zadávat.
4	Možnost zadat defekt rychle bez vyplnění množství polí a zjišťování toho, kdo ho má řešit, nebo kdo je zadavatelem nebo řešitelem. Mělo by to chtít max. 3 povinné informace.
5	Vyplňuji spoustu polí, které mi nepřináší žádnou přidanou hodnotu.
6	Úprava workflow dle reálného workflow.
7	Jednoznačná metodika pro práci s integračním defektem.
8	Kdyby zodpovědní řešitelé defekt přejali na sebe a řešili, a nenechávali ho ve stavu jako otevřený. Případně aby ho sami přeposlali na jiného kompetentního řešitele.
9	Simplifikace zadávání a znepovinit položky, které bohužel CoET považuje za mandatorní.
10	Chtěl bych, aby si uživatel mohl nastavit automatické vyplňování polí hodnotou, kterou si sám určí. Důležité to je hlavně pro pole Description.
11	Malé změny, např.: změna Target Application i v jiném stavu než OPEN, mít na jednom místě i PROD def. (ne v SNOW), atd…
12	Zeštíhlení počtu polí a zjednodušení flow ve smyslu vyskakovacích obrazovek na jednotlivých transitions. Také bych některá pole udělala jako optional, nemyslím, že je potřeba, aby téměř každé pole na defectu bylo jako mandatory.
13	Přívětivější prostředí pro zadávání defektu
14	Možnost přidávat screen obrazovky už ve vytváření defekt. Ne v situaci až je defekt vytvořen.
15	Automatické vyplnění některých polí na základě příslušnosti ke Squadu (Original Application, Original Tribe, Original Squad).
16	Možnost předvyplnění většího počtu polí – ať už formou, jako to bylo v HP ALM (nastavení hodnot konkrétních polí a pak kliknutí na jedno tlačítko) nebo třeba přebírání hodnot z jiného defektu – když vystavuji defekt z jedné funkční oblasti a z jedné user story, tak většina polí je stejná

Zdroj: vlastní zpracování

Veškeré odpovědi, které respondenti uvedli, budou podrobeny následné analýze, na jejímž základě bude rozhodnuto, zdali je možné je efektivně zapracovat, případně budou navržena taková technická řešení, která by snížila uživatelskou nespokojenost při práci s defektem.

5 Závěr

Hlavním cílem diplomové práce bylo vytvoření nové metodiky pro práci s defekty v novém jednotném systému Jira. Bylo vytvořeno nové workflow defektu a nová metodika pro práci s defekty v České spořitelně, a. s.

V teoretické části práce byly na základě prostudovaných publikací definovány veškeré pojmy, které jsou pro pochopení problematiky defekt managementu nezbytné. Dále byl uveden obecný životní cyklus defektu a jeho různé stavy. Současně v této části byl stručně charakterizován jak software, který byl dříve v České spořitelně využíván pro správu defektů, tak i nový nástroj, ve kterém se bude s defekty pracovat.

Praktická část diplomové práce se věnuje obecnému popisu části agilního fungování společnosti Česká spořitelna, a. s., které bylo nutné charakterizovat s ohledem na jejich dopad do defekt managementu.

Byly rozebrány důvody a přínosy převedení defekt managementu do nového nástroje. Následně po vymezení teoretických poznatků o společnosti bylo vytvořeno nové workflow defektu s jeho stavy, které byly také podrobně popsány. Zároveň byla charakterizována a detailně vytyčena jednotlivá pole, která se u defektu využívají.

V závěru celé práce je zpracován dotazník, který byl odeslán uživatelům nového defekt managementu k vyplnění. Na základě zpracovaného dotazníku je možné uvést, že přechod do nového nástroje Jira a úpravy v procesu defekt managementu byly pro banku pozitivním posunem dopředu. Výsledky dotazníkového šetření a veškeré připomínky poslouží k dalšímu vývoji systému.

Autor této diplomové práce zastřešuje v bance pozici defekt koordinátora a byl jednou z pověřených osob, které zajišťovaly vytvoření nové metodiky pro práci s defektem v souvislosti s přechodem do nového nástroje.

6 Seznam použitých zdrojů

P. MATHURS, Aditya, 2011. Foundations of Software Testing. Indie: Pearson. ISBN 978-81-317-5908-0.

J. ANDERSON, David. The Principles and General Practices of the Kanban Method. David J Anderson School of Management [online]. [cit. 2021-02-04]. Dostupné z: https://djaa.com/principles-and-general-practices-of-the-kanban-method/

J. ANDERSON, David a Andy CARMICHAEL, 2016. Essential kanban condensed. Seattle: Lean Kanban University Press. ISBN 978-0-9845214-2-5.

ATLASSIAN. How to Use Jira | Official Buyer & User Guide [online]. [cit. 2020-12-14]. Dostupné z: https://www.atlassian.com/software/jira/guides

DOLEŽAL, Jan. SCRUM – rámec agilního přístupu. PM Consulting [online]. [cit. 2021-02-02]. Dostupné z: https://www.pmconsulting.cz/pm-wiki/scrum/

BECK, Kent, Mike BEEDLE, Arie VAN BENNEKUM, et al., 2001. Manifest Agilního vývoje software [online]. [cit. 2021-02-02]. Dostupné z: http://agilemanifesto.org/iso/cs/manifesto.html

BOOG, Jason, 2020. 5 Root Cause Analysis Tools For Better Testing & QA. The QA Lead [online]. [cit. 2021-02-01]. Dostupné z: https://theqalead.com/topics/root-cause-analysis/

BUCHALCEVOVÁ, Alena, 2005. Metodiky vývoje a údržby informačních systémů. Praha: Grada Publishing, 163 s. 1. vydání. ISBN 978-80-247-1075-4.

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Grössl ZDENĚK, Martin KOMÁREK, Ondřej MACEK a Mlynář RADOSLAV, 2016. Efektivní testování softwaru. Praha: Grada Publishing, 232 s. ISBN 978-80-247-5594-6.

Defect Severity – SOFTWARE TESTING Fundamentals [online]. [cit. 2021-12-14]. Dostupné z: https://softwaretestingfundamentals.com/defect-severity/

GRAHAM, Dorothy, Rex BLACK a Erik VAN VEENENDAAL, 2012. Foundations of software testing: ISTQB Certification. 4. vydání. Hampshire: Cengage, 256 s. ISBN 9781408044056.

HOUSTON, Tom. What is Root Cause Analysis? Smart Bear [online]. [cit. 2021-03-12]. Dostupné z: https://smartbear.com/learn/performance-monitoring/what-is-root-cause-analysis/

IEEE COMPUTER SOCIETY. IEEE Standard Classification for Software Anomalies. 2009.

INGENO, Joseph, 2018. Software Architect's Handbook. Birmingham: Packt Publishing. ISBN 978-1-78862-406-0.

ISTQB Glossary [online]. [cit. 2021-12-13]. Dostupné z: https://glossary.astqb.orh/search/defect

MAGALHÃES, Ivan Luizio, 2015. Difference between Error, Fault, Defect and Failure LinkedIn [online]. [cit. 2021-12-14]. Dostupné z: https://www.linkedin.com/pulse/difference-between-error-fault-defect-failure-ivan-luizio-magalh%C3%A3es/

2016. Clever and Smart [online]. [cit. 2021-12-14]. Dostupné z: https://www.cleverandsmart.cz/je-bezpecnostni-chyba-error-bug-flaw-defekt-a-hole-totez/

KERZNER, Harold, 2018. Project Management Best Practices: Achieving Global Excellence. 4. vydání. Wiley. ISBN 978-1-119-46885-1.

KRISHNA, Rungta, 2020. Severity & Priority in Testing: Differences & Example [online]. [cit. 2021-12-14]. Dostupné z: https://www.guru99.com/defect-severity-in-software-testing.html

MICROFOCUS. ALM Help Center [online]. [cit. 2021-12-14]. Dostupné z: https://admhelp.microfocus.com/alm/en/15.0-15.0.1/online_help/Content/Resources/_TopNav/_TopNav_Home.htm

OTTA, Jiří. Testování v procesu implementace informačního systému [online]. 2016 [cit. 2021-12-14]. Dostupné z: https://www.systemonline.cz/erp/testovani-v-procesu-implementace-informacniho-systemu.htm

RADIGAN, Dan. Four agile ceremonies, demystified. Atlassian [online]. [cit. 2021-02-03]. Dostupné z: https://www.atlassian.com/agile/scrum/ceremonies

ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ, 2013. Řízení kvality softwaru. Brno: Computer Press.

SCHWABER, Ken a Jeff SUTHERLAND, 2017. The Scrum Guide: The Definitive Guide to Scrum: The Rules of the Game. Dostupné také z: https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

SCOTLAND, Karl, 2010. Methods & Tools – Summer 2010. 78.

SEELA, Swati, 2020. Software Testing Help. Defect Severity and Priority in Testing with Examples and Difference [online]. [cit. 2021-12-14]. Dostupné z: https://www.softwaretestinghelp.com/how-to-set-defect-priority-and-severity-with-defect-triage-process/

ŠOCHOVÁ, Zuzana a Eduard KUNCE, 2014. Agilní metody řízení projektů. Brno: Computer Press. ISBN 978-80-251-4194-6.

SUTHERLAND, Jeff, 2012. The Scrum Papers: Nut, Bolts, and Origins of an Agile Framework [online]. [cit. 2021-02-03]. Dostupné z: http://jeffsutherland.org/scrum/ScrumPapers.pdf

SWEENEY, Michael. Agile vs Waterfall: Which Method is More Successful? - Clearcode Blog. Clearcode [online]. [cit. 2021-02-02]. Dostupné z: https://clearcode.cc/blog/agile-vs-waterfall-method/

YADAV, Rahul, 2020. Understanding Priority vs Severity. QAtechnicals [online]. [cit. 2021-12-14]. Dostupné z: https://qatechnicals.wordpress.com/bug-and-bug-tracking/understanding-priority-vs-severity/

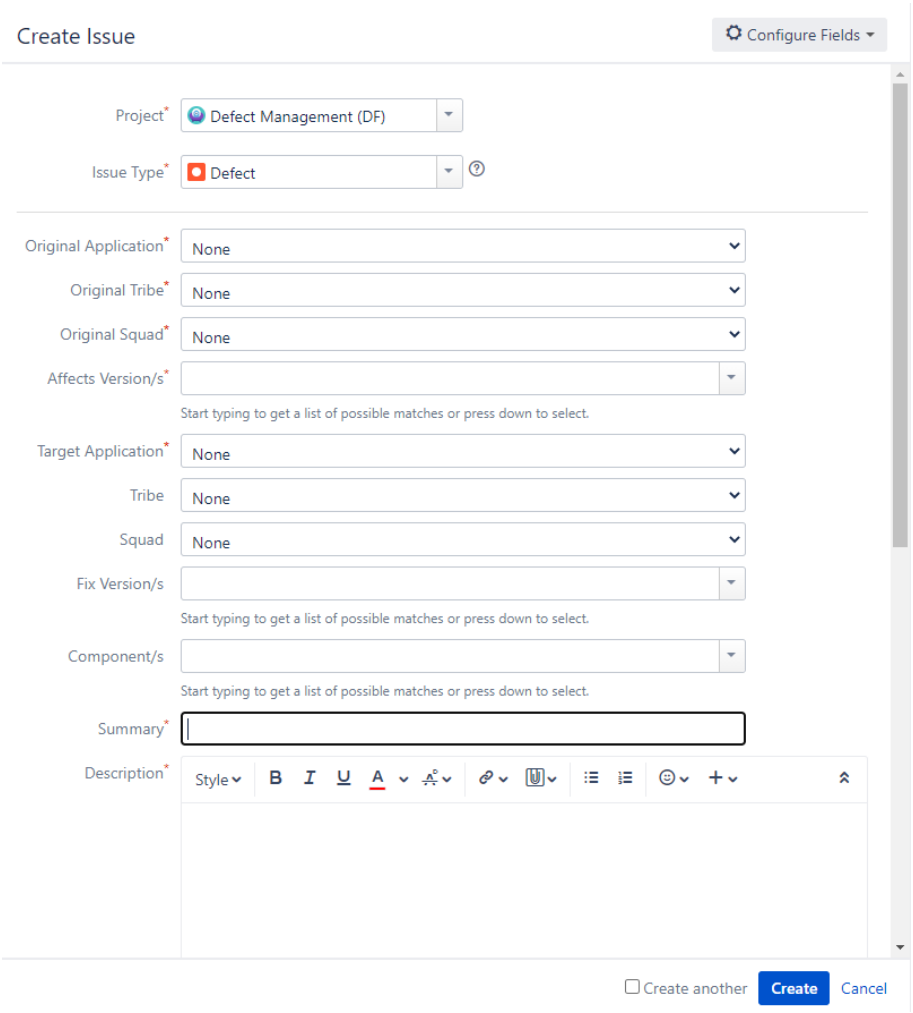
RAHIM, Riz, 2019. Root Cause Analysis: The 5 Best Tools with Examples and Steps. Upskill Nation [online]. [cit. 2021-02-04]. Dostupné z: https://upskillnation.com/root-cause-analysis/

THORN, Mary, 2016. T-Shaped People: Time to Get in Shape for Your Testing Future [online]. [cit. 2021-02-04]. Dostupné z: https://www.slideshare.net/TechWellPresentations/tshaped-people-time-to-get-in-shape-for-your-testing-future

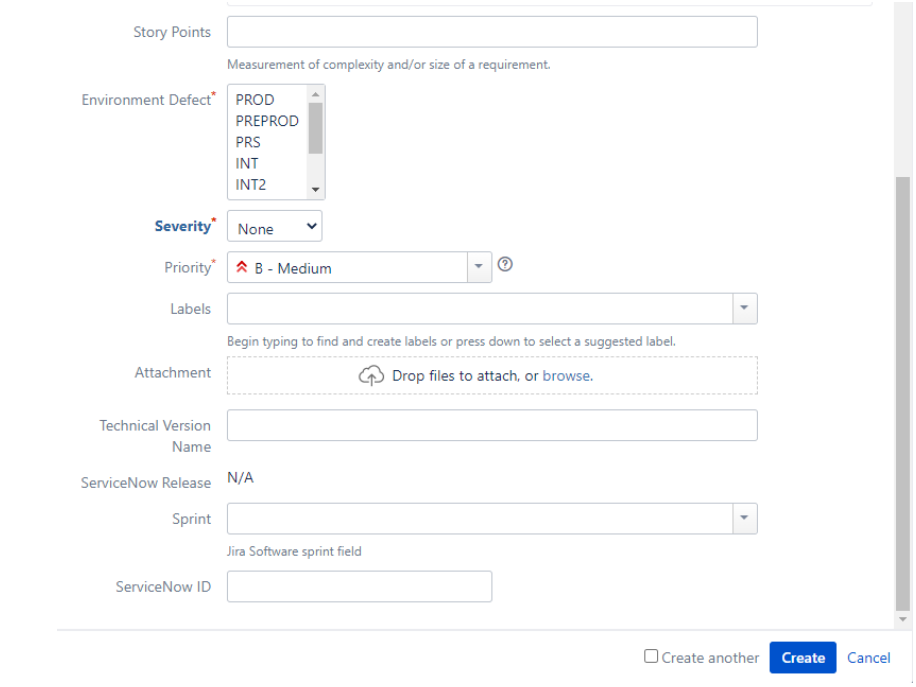
What are Scrum Ceremonies?, 2018. Visual Paradigm [online]. [cit. 2021-02-04]. Dostupné z: https://www.visual-paradigm.com/scrum/what-are-scrum-ceremonies/

7 Seznam příloh

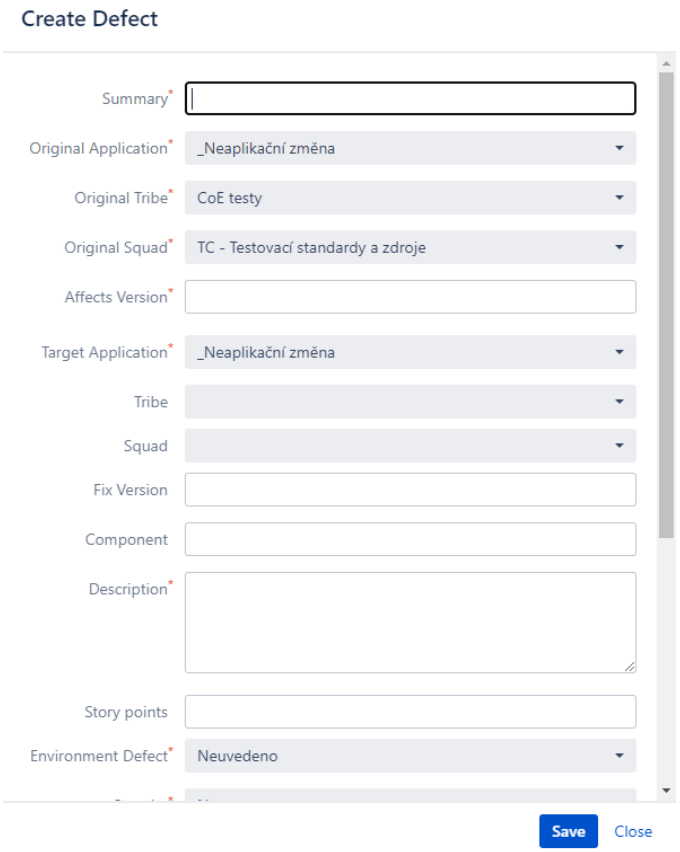
Příloha A *Ukázka polí při zakládání defektu z tlačítka CREATE 1/2*



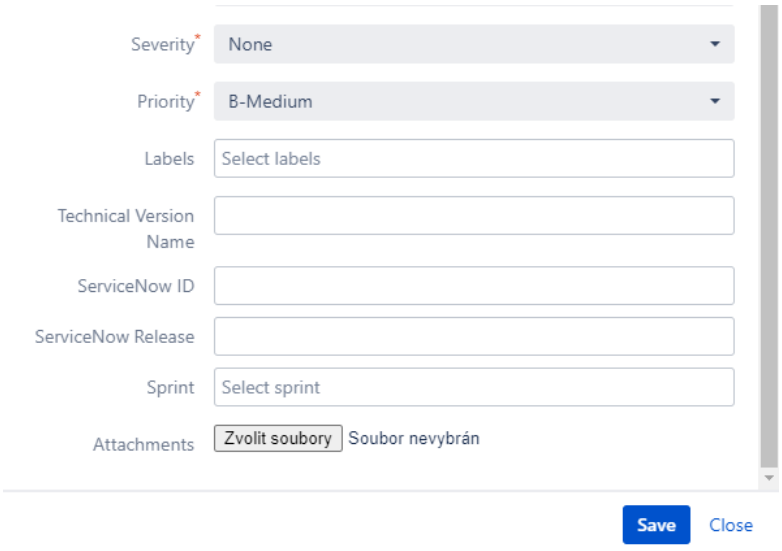
Příloha B *Ukázka polí při zakládání defektu z tlačítka CREATE 2/2*



Příloha C *Ukázka polí při zakládání defektu z tlačítka New Defect 1/2*



Příloha D *Ukázka polí při zakládání defektu z tlačítka New Defect 2/2*



Vnímání nového Defekt Managementu ve FUJIRA

Dobrý den,

věnujte, prosím, několik minut svého času pro vyplnění následujícího dotazníku, který je zcela anonymní a nezabere Vám více jak 5 minut.

Dotazník je vytvořen pro použití v mé diplomové práci a zkoumá vnímání nového Defekt Managementu (ve FUJIRA) uživatelem. Získaná data budou použita k výzkumným účelům.

Děkuji za Váš čas,
Jan Flachs

*Povinné pole

1. Jak často vytváříte defekt ve FUJIRA? *

Označte jen jednu elipsu.

- ◯ Denně
- ◯ 1-2x do týdne
- ◯ 2x do měsíce
- ◯ Jiné: _____

2. Je dle vašeho pohledu nový defekt management ve FUJIRA lepší než v HP ALM? *

Označte jen jednu elipsu.

- ◯ Ano
- ◯ Ne
- ◯ Srovnatelný
- ◯ Nemám srovnání

Vnímání nového Defekt Managementu ve FUJIRA

3. Zabere Vám vytváření defektu méně času než v HP ALM? *

Označte jen jednu elipsu.

- ◯ Ano
- ◯ Ne
- ◯ Srovnatelně
- ◯ Nemám srovnání

4. Vnímáte vytváření defektu ve FUJIRA jako uživatelsky přívětivé? *

Označte jen jednu elipsu.

- ◯ Ano
- ◯ Ne
- ◯ Nemám vyhraněný názor.

5. Je metodika Defekt Managementu na Confluence srozumitelná a dostatečná? *

Označte jen jednu elipsu.

- ◯ Ano, je srozumitelná a dostatečná.
- ◯ Ne, je srozumitelná, ale nedostatečná.
- ◯ Ne, je dostatečná, ale nesrozumitelná.
- ◯ Ne, je nedostatečná a nesrozumitelná.

6. Co by vám pomohlo ve zlepšení práce s defektem? *

Označte jen jednu elipsu.

- ◯ Mám vše, co potřebuji.
- ◯ Jiné: _____

2/2