ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

MASARYKŮV ÚSTAV VYŠŠÍCH STUDIÍ

DIPLOMOVÁ PRÁCE

Zavedení kontroly kvality softwaru ve startupu

Quality Assurance Implementation in the Startup Company

2024

Bc. Dominik Krautstengel

Studijní program: Projektové řízení inovací

Vedoucí práce: Ing. Mgr. Tomáš Sadílek, Ph.D.

ZADÁNÍ DIPLOMOVÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Krautstengel** Jméno: **Dominik** Osobní číslo: **470380**

Fakulta/ústav: **Masarykův ústav vyšších studií**

Zadávající katedra/ústav: **Institut manažerských studií**

Studijní program: **Projektové řízení inovací**

II. ÚDAJE K DIPLOMOVÉ PRÁCI

Název diplomové práce:

Zavedení kontroly kvality softwaru ve startupu

Název diplomové práce anglicky:

Quality Assurance Implementation in the Startup Company

Pokyny pro vypracování:

CÍL: Zdokumentování zavedení systému kontroly kvality softwaru vytvářeného ve startupové společnosti zabývající se zprostředkováním VPN Cloudových služeb a Zero Trust Networku.
PŘÍNOS: Přínosem práce je návrh inovativních procesů pro moderní systém kontroly kvality (Quality Assuarance, QA) softwaru.
OSNOVA: Úvod, 1. Teoretická část, 2. Quality Assurance, 3. Trendy v QA, 4. BDD, TDD a jiné přístupy ke QA, 5. Životní cyklus vývoje softwaru, 6. Post mortem testování, 7. Praktická část, Závěr

Seznam doporučené literatury:

GRAHAM, Dorothy, Rex BLACK a Erik VAN VEENENDAAL. Foundations of software testing: ISTQB certification. Fourth edition. Andover, Hampshire: Cengage, [2020]. ISBN 978 1473764798.
RASMUSSON, Jonathan. The way of the web tester: a beginner's guide to automating tests. Raleigh, North Carolina: The Pragmatic Bookshelf, [2016]. Pragmatic programmers. ISBN 978 1680501834.
WHITTAKER, James A., Jason ARBON a Jeff CAROLLO. How Google tests software. Upper S addle River, NJ: Addison Wesley, c2012. ISBN 978 0321803023.

Jméno a pracoviště vedoucí(ho) diplomové práce:

Ing. Mgr. Tomáš Sadílek, Ph.D. Masarykův ústav vyšších studií ČVUT v Praze

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) diplomové práce:

Datum zadání diplomové práce: **08.12.2023** Termín odevzdání diplomové práce: **25.04.2024**

Platnost zadání diplomové práce: _____

Ing. Mgr. Tomáš Sadílek, Ph.D. podpis vedoucí(ho) práce	Ing. Dagmar Skokanová, Ph.D. podpis vedoucí(ho) ústavu/katedry	prof. PhDr. Vladimíra Dvořáková, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Diplomant bere na vědomí, že je povinen vypracovat diplomovou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v diplomové práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

KRAUTSTENGEL, DOMINIK. Zavedení kontroly kvality softwaru ve startupu. Praha: ČVUT 2024. Diplomová práce. České vysoké učení technické v Praze, Masarykův ústav vyšších studií.

Prohlášení

Prohlašuji, že jsem svou diplomovou práci vypracoval samostatně. Dále prohlašuji, že jsem všechny použité zdroje správně a úplně citoval a uvádím je v přiloženém seznamu použité literatury.

Nemám závažný důvod proti zpřístupňování této závěrečné práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Praze dne: 22. 04. 2024

Poděkování

Rád bych poděkoval Ing. Mgr. Tomáši Sadílkovi, Ph.D., vedoucímu mé diplomové práce za vedení i pod časovým nátlakem, který se práce vyžádala. Energie a čas věnována asistenci mi velice pomohla a ochotný přístup byl ideálním partnerem v dokončení práce. Dále bych rád poděkoval mé rodině za neskonalou podporu během mého studia. Další poděkování patří týmu ze společnosti GoodAccess s.r.o. za důvěru a poskytnutí všech prostředků pro potřeby této práce.

Abstrakt

Diplomová práce s názvem „Zavedení kontroly kvality softwaru ve startupu“ popisuje zavedení procesů kontroly kvality do společnosti GoodAccess, která si dala za cíl zdokonalení celého procesu a s tím navázaných aktivit. Cílem je úprava vývojového procesu a procesu kontroly kvality jako takového. V práci je popsáno vypracování testovací databáze a užití prostředků pro manuální a automatizované testování. Autor popisuje, jak vypadají testovací scénáře a automatizované postupy testování dle nejmodernějších trendů a zavádí BDD a TDD přístup k testování. Společně se zadavatelem byly vypracovány jednotlivé cíle jejich vypracování dostalo časový rámec jednoho roku, kdy byly postupně výsledky analyzovány po jednotlivých kvartálech. Jednotlivé cíle jsou součástí diplomové práce. Teoretická část se věnuje popisu, co je kontrola kvality, jak funguje a jak se využívá v moderním přístupu k vývoji softwarových produktů. Popisuje jednotlivé druhy testování, které jsou následně využity v části praktické. V praktické části čtenář nalezne popis a využití jednotlivých nástrojů podporujících kontrolu kvality v softwaru, stejně tak jsou uvedeny příklady jednotlivých testovacích scénářů nebo automatických testů.

Klíčová slova

Quality Assurance, QA, startup, testování, kontrola kvality, test, automatické testy, manuální testy, GoodAccess, VPN, Zero Trust Network, testing, kvalita, Cypress, VSCode, Qase, Google Workspace, Control Panel, ZTNA

Abstract

The thesis entitled "Quality Assurance Implementation in the Startup Company" describes the introduction of quality control processes in GoodAccess, which aimed to improve the whole process and related activities. The goal is to modify the development process and the quality control process as a whole. The paper describes the development of a test database and the use of resources for manual and automated testing. The author describes how test scenarios and automated testing procedures look like according to the latest trends and introduces BDD and TDD approach to testing. Together with the client, the individual objectives were developed with a timeframe of one year, where the results were analyzed quarter by quarter. The individual objectives are included in the thesis. The theoretical part describes what quality control is, how it works and how it is used in the modern approach to software product development. It describes the different types of testing, which are then used in the practical part. In the practical part, the reader will find a description and use of individual tools supporting quality control in software, as well as examples of individual test scenarios or automated tests.

Keywords

Quality Assurance, QA, startup, testing, quality control, test, automated tests, manual tests, GoodAccess, VPN, Zero Trust Network, testing, quality, Cypress, VSCode, Qase, Google Workspace, Control Panel, ZTNA

Obsah

Úvod

Informační technologie jsou každým dnem vyvíjeny s exponenciálně se zrychlujícím tempem. Tato digitální éra přinesla revoluci, se kterou mohou malé i velké společnosti řešit problémy lidstva na úrovni, kterou jsme doposud nezažili. Mohou na trh přivádět převratné produkty s nebývalou rychlostí. Tento spěšný vývojový cyklus je zásadní pro získání silné pozice na trhu a následné úpravy a vylepšení pro neustále se měnící potřeby zákazníků. Tento pokrok sebou přináší významné výzvy, a to zajištění vysoké kvality a dostupnosti služeb, a tedy softwarových produktů. Tento problém se řeší pomocí nasazení expertů pro řízení kvality softwaru – QA. Řízení kvality nelze přeceňovat a jeho význam pro zásadní udržování softwaru je stále důležitější pro zajištění úspěchu společností a jejich produktů.

Zajištění kvality bylo tradičně považováno za poslední kontrolní bod v životním cyklu vývoje softwaru, momentálně se však střetáváme s agilním přístupem k vývoji softwaru. QA je dnes integrováno do všech fází vývojového procesu, přičemž se uplatňuje neustále iterovaní testovacích cyklů již od počátečních fází vývoje. Tento posun klade důraz nejen na odhalování chyb po dokončení vývoje, ale na prevenci chyb prostřednictvím průběžného testování a zlepšování procesů již od fáze návrhu softwaru.

Pokud se zaměříme na startupové prostředí, představuje proces zajištění kvality často velkou výzvu pro společnosti. A to vzhledem k omezeným zdrojům a častému upřednostňování rychlosti uvedení produktů na trh a jejich následné dynamické a rychlé inovace. Tento přístup je výhodná pro rychlé iterace, ale dlouhodobá kvalita softwarových produktů a jejich udržitelnost může být ohrožena. Proto zavádění QA procesů do podobných firem vyžaduje detailní přístup s důrazem na zachování dostatečné kvality a stejně tak zachování agility vývoje produktu. Jinými slovy QA nesmí čekat na vývoj a vývoj naopak na QA.

Tato práce se věnuje praktickému zavedení QA procesů do startupové společnosti GoodAccess s.r.o., která si uvědomila kritickou potřebu zajištění robustních procesů kontroly kvality vzhledem k nátuře dodávaného produktu. Společnost se zabývá vývojem SaaS řešení pro dodávání zabezpečeného připojení do firemní sítě pomocí VPN. Software sám o sobě vyžaduje minimální chybovost, vzhledem k práci s citlivými údaji v rámci firemních sítí. Služba obsahuje mnoho podpůrných funkcí, které jsou pro zákazníky přístupné podle volby předplatného.

Cílem práce je zhodnocení stávajícího stavu a zavedení komplexních procesů kontroly kvality v rámci startupové společnosti, ve webové aplikaci GoodAccess Central Dashboard. V rámci zhodnocení stávajícího stavu se bude porovnávat stav před zavedením projektu a jeho konečný výsledek po úpravách procesů a zavedení inovativních postupů řešení kontroly kvality ve společnosti na softwarovém produktu. Cílem je splnění cílů určených zadavatelem (společnost GoodAccess) v určeném časovém rámci. Následně bude porovnána výchozí hodnota požadavků s koncovou hodnotou po uplynutí časového rámce. Cílem je konkrétně vypracování QA postupů, Testovací dokumentace obsahující veškeré testovací scénáře a testovací plány, zdokonalení manuálních testu a konečně zavedení automatizovaných testů pro vybrané sety testovacích scénářů.

TEORETICKÁ ČÁST

1 Quality Assurance

Při aktuálním trendu v požadavcích na rychlé doručení a inovace si již mnoho lidí od projektových manažerů po vývojáře softwaru nedovolí zpochybňovat roli kontroly kvality při vývoji softwarových produktů. V některých společnostech se volí cesty, kdy kontrolu kvality provádí samotný vývojáři. V těchto případech často dochází ke kolizi v oborech a vývojáři nevyužívají systematické postupy, leč dokážou kvalitně ovládat jednotlivé testovací nástroje. Mnoho vývojářů zároveň zastává názoru, že testování a kontrola kvality je jakási zpětná vazba, která jim má ulehčit či zlepšit práci. To je z velké míry pravda, avšak realita je taková, že testování v kontextu kontroly kvality má především za úkol hledat chyby v softwaru a zároveň, což je možná ještě důležitější, má předcházet vůbec tvorbě chyb již při vývoji. To se provádí zavedením správných přístupů, nástrojů a metodik (Tian, 2005).

Dalším požadavkem, který je potřeba spolehlivě řešit jsou samozřejmě požadavky koncových uživatelů. Ty se dají shrnout jednoduše do dvou kategorií:

- Je potřeba aby software plnil funkce, které má
- Je potřeba aby tyto funkce splnil bezchybně

Pro jednoduchou představu může být uveden příklad: "Hotelový rezervační systém má bezchybně vyřizovat rezervace zákazníků. Nemá vyřizovat objednávku surovin do hotelové kuchyně. Přičemž první požadavek validuje kritéria žádaných funkcí, tedy těch, co se věnují rezervaci hotelového pokoje zákazníkem A druhý požadavek validuje, že všechny tyto funkce fungují bezchybně tak jak mají a jak uživatel očekává. Tedy, že rezervace pokoje místo pokoje neobjedná surovinu do hotelové restaurace (O'Regan, 2022; Tian, 2005 str. 3-6).

Na základě těchto požadavků koncových uživatelů můžeme rozpoznat další nutné aktivity, které musí quality assurance, a tedy QA inženýři vykonávat. Tyto úkoly je potřeba provádět takových způsobem, aby jak jednotlivci na pracovních pozicích, kteří je vykonávají, tak organizace jako takové dokázaly spolehlivě zákazníkům zaručit, že dodávaný software splňuje požadavky na kvalitu. Tyto aktivity zahrnují:

- plánování kontroly kvality
- budování souboru testovacích scénářů
- vykonávání manuálních a automatických testů
- vykonávání vybraných QA technik pro validaci a verifikaci
- měření a analýza nasbíraných dat pro neustále zlepšování kvality dodávaného softwaru (O'Regan, 2022)

1 Testování a kontrola kvality

Je důležité si porovnat používané pojmy testování a kontrola kvality. Testování jako takové je aktivita vyvíjená za účelem nalezení chyb v softwaru nebo validace požadovaných kritérií. U testování jako takového je potřeba, si uvědomit jedno ze základních pravidel, a to, že žádný počet provedených testů nezaručí kvalitní software. Chybovost je důsledek mnoha aktivit, které předcházejí, a i následují po samotném testování. Často se můžeme setkat například s problémy vycházejících z DevOps procesů a podobně. Testování jako takové je samozřejmě pravděpodobně ta nejvíce viditelná a potažmo také nejdůležitější součást kontroly kvality softwaru. Dopady testování jsou nevíce znatelné a nejlépe měřitelné, je tedy možné vyvodit zajímavé statistiky, které mohou napomáhat k neustálému zlepšování kvality softwaru (O'Regan, 2022).

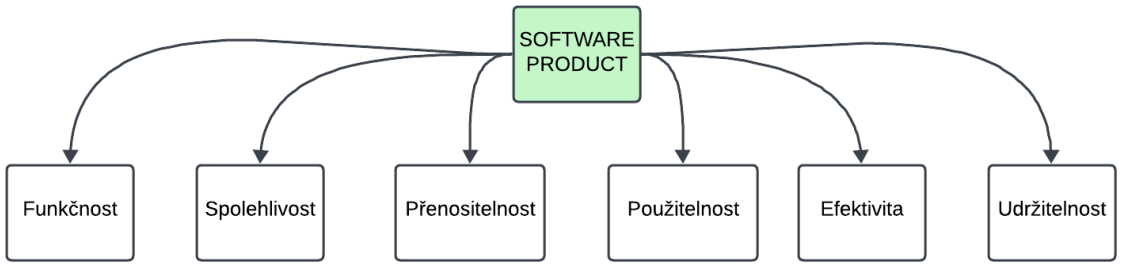
Naproti tomu pod kontrolou kvality je zahrnut soubor všech aktivit, které jednak zahrnují testování samotné a zároveň všechny podpůrné kvality, které se věnují také plánování, návrhu procesů, monitorování vývoje a neustálému zlepšování. Kontrola kvality tedy představuje komplexní přístup k zajištění, že výsledný produkt nebo služba splňuje předem stanovené požadavky a očekávání jak zákazníka, tak stakeholderů (Galin, 2004; Aniche, 2022).

Pro další potřeby této práce se budou tyto dva pojmy vzájemně propojovat, tedy pokud budeme referovat k termínu testování je na něj ve většině případů navázáno ve spojení s aktivitami, které jsou prováděny obecně v procesech kontroly kvality softwaru. Tento fakt je matoucí ale v obecné rovině se tyto termíny volně zaměňují v mnoha pramenech i populárně naučných zdrojích a koneckonců i v praxi (O'Regan, 2022).

1.1 Kvalita softwaru

V rámci kontroly kvality se neustále setkáváme s termínem kvalita softwaru. Je tedy správné se ptát na to, co za sebou termín kvalita softwaru skrývá. Termínem rozumíme především to, co má daný software splňovat, tak jak jsme si uvedli v předešlých kapitolách. Kvalita samotná je soubor několika vlastností. Mezi ně řadíme funkčnost, spolehlivost, použitelnost, efektivitu, udržitelnost a přenositelnost (podle ISO/IEC 25010:2011). Je pravděpodobné, že podle toho, koho se konkrétně ptáme, bude soubor vlastností lehce odlišný. Problém je v tom, že neexistuje jednotná definice a většinou jde o to, jak si je nastaví daná společnost, jež vyvíjí software. Mezi další spíše okrajové, vlastnosti můžeme dále zmínit kompatibilitu a zabezpečení. V kontextu této práce se posledním dvou zmíněným však nebudeme věnovat, jelikož tyto vlastnosti jsou často validovány za využití externích zdrojů. Nyní si jednotlivé vlastnosti přiblížíme.

OBRÁZEK 1 - KVALITA SOFTWARE, ZÁKLADNÍ VLASTNOSTI, ZPRACOVÁNÍ AUTOR, 2024



(Bourque a Fairley, 2014; ISO/IEC 25010:2011)

1.1.1 Funkčnost

Vztahuje se k míře schopnosti softwarového produktu plnit své zamýšlené funkce, tak jak je vymezeno jeho specifikací. Produkt je považován za funkční, že plní funkce a operace takové, které vyhovují jak požadavkům, které jsou v jeho samotné definici, tak zároveň vyhovuje daným normám a předpisům, tam kde je produkován a kde bude také vydáván. Funkčnost je hodnocena posouzením úplnosti softwaru, počínaje odpovídající dokumentací jak jeho funkcí, tak průběhu jeho vývoje, kde je správně zdokumentováno, jaké kroky byly provedeno za účelem splnění všech požadavků. Je tedy jednoduše řečeno posuzována shoda požadavků s jejich reálnou implementací (Tian, 2005).

1.1.2 Spolehlivost

Tato vlastnost softwaru souvisí se schopností softwarového produktu plnit požadované funkce a operace za stanovených podmínek po stanovenou dobu bez výskytu chyb, které vedou k nesprávnému nebo žádnému výsledku. Do této vlastnosti kvality softwaru zahrnujeme několika možných tolerancí, které se dají pohybovat dle potřeby podle konkrétního softwaru. To znamená, že každý software může mít jinou toleranci pro chybovost v určitých samostatných částech/ modulech. Dá se tedy pracovat s tím, že některé moduly mohou mít menší prioritu než jiné, u kterých je ve své podstatě možno tolerovat určitý počet a druh chyb. Některé chyby nemusí vést k nesprávnosti výsledků, ale například mohou jenom ovlivnit vzhled softwaru, takové chyby nemají nutně dopad na funkčnost, nicméně spolehlivost je samozřejmě ovlivněna (O'Regan, 2022; ISO/IEC 25002:2024).

1.1.3 Přenositelnost

Přenositelnost se zabývá flexibilitou vyvíjeného softwaru, při přenosu a přizpůsobení se novým prostředím, jednoduše řešeno, že je možno software například instalovat na různých operačních systémech, které mohou mít stovky konfigurací nebo že funguje na různě velkých rozlišeních monitorů apod. Patří sem i adaptabilita a schopnost softwaru se těmto novým prostředím s minimálním úsilím. Zároveň je také důležitá kompatibilita s ostatním instalovaných softwarem v daných prostředích, často jde například o koexistenci s antivirovými aplikacemi (Tian, 2005).

1.1.4 Použitelnost

Dalším kritickým aspektem kvality softwarových produktů je použitelnost. Jde o přímou interakci koncových uživatelů se softwarem, kdy je naším cílem co nejvíce uživatelům usnadnit komunikaci, naučení se funkcí a dosažení požadovaných výsledků s dostatečnou efektivitou. Patří sem tedy celková spokojenost uživatelů, z čehož každá jednotlivá složka hraje významnou roli při výběru vhodného, „úspěšného", softwaru na trhu. Tato vlastnost se na první pohled nemusí zdát jako něco, co by souviselo s kontrolou kvality jako takovou v klasickém pohledu kdy se „QA = testování". V moderním pojetí QA, se setkáváme právě s tím faktem, že se posuzuje vše, nejenom chybovost jako taková, a proto mnoho pracovníků v QA analyzuje a vyhodnocuje právě i podobné vlastnosti jako je použitelnost nebo i vizuální líbivost softwaru (Tian, 2005; ISO/IEC 25002:2024).

1.1.5 Efektivita

Efektivitou rozumíme schopnost softwaru vykonávat požadované operace s ohledem na co nejvýhodnější využití zdrojů. Snahou je maximalizace výkonnosti při minimalizaci potřebného výpočetního výkonu. To zahrnuje optimalizaci výkonnostních charakteristik softwaru, jako je rychlost, škálovatelnost a odezva a zajištění toho, aby software operoval za podobného využití výkonu při různých situacích a konfiguracích prostředí, ve kterém operuje (Mendez, 2023).

1.1.6 Udržitelnost

Pro dlouhodobou životaschopnost softwaru je zásadní udržitelnost, tedy snadnost, s jakou lze software upravovat a spravovat ho za účelem odstranění závad, zlepšení výkonu nebo přizpůsobení se změnám prostředí. Zahrnuje úvahy o modularitě, architektuře softwaru, která usnadňuje aktualizace a srozumitelnost; čitelnosti, která vývojářům umožňuje snadné pochopení a orientaci v kódové základně; a testovatelnosti, míře, do jaké je software možno otestovat do dostatečné míry a verifikovat a validovat všechna testovaná kritéria. Dalším moderním přístupem, který je v tomto směru vhodné zmínit je, že při využití Agilního vývojového přístupu jsou již často samotní QA pracovníci zařazeni do vývoje tak, aby zajistili, že nově produkovaný software bude testovatelný do maximální možné míry. Představit si tento aspekt můžeme jako zaručení testerem toho, že software dává jednoduše řečeno smysl z pohledu testera a není přespříliš zbytečně komplikovaný a podobně (Galin, 2018; ISO/IEC 25002:2024).

1.2 Úskalí definice kvality softwaru

V momentě, kdy se snažíme definovat co je kvalita softwaru narážíme hned na několik problémů. Tím pravděpodobně nejvýznamnějším je celkově definice samotná. Neexistuje oficiální definice a například fakt, že se zdá, že je definována například mezinárodními normami, tak při bližším zkoumání zjistíme, že samotné normy se v definicích často rozchází (O'Regan, 2022).

Kvalita v kontextu softwaru přesahuje pouhou funkčnost a výkonnost a zahrnuje aspekty, jako je použitelnost, udržovatelnost a spolehlivost, z nichž každý může být různými stakeholdery interpretován a hodnocen odlišně. Například vývojář může upřednostňovat udržitelnost a efektivitu a zaměřit se na čistý, dobře zdokumentovaný kód, který usnadňuje budoucí úpravy. Naproti tomu koncový uživatel může na prvním místě oceňovat použitelnost a funkčnost a hledat intuitivní rozhraní a bezproblémový výkon. Tento rozdíl v prioritách komplikuje vytvoření jediné, univerzálně použitelné definice kvality softwaru (O'Regan, 2022).

Rychlý vývoj softwarových technologií a metodik navíc přináší další vrstvu složitosti. S tím, jak se objevují nové technologie a stávající se vyvíjejí a inovují, vyvíjejí se odpovídajícím způsobem i parametry, podle kterých se kvalita softwaru měří a vyhodnocuje. To, co bylo před deseti lety považováno za kvalitní software na základě tehdejších technologických standardů a očekávání uživatelů, nemusí vyhovovat dnešním kritériím. Tento neustálý vývoj zpochybňuje zavedení pevné definice kvality softwaru, protože každá taková definice musí být aktuální v kontextu inovací a změn na straně koncových uživatelů, tedy jejich požadavků na software (Stamelos, Sfetsos, 2007).

Jak již bylo zmíněno, různé interpretace kvality softwaru v mezinárodních normách a standardech jsou dalším příkladem obtížnosti stanovení univerzální definice. Různé normalizační orgány a průmyslové skupiny často zdůrazňují různé aspekty kvality

softwaru. Například norma ISO/IEC 25010 poskytuje model kvality softwaru, který mezi charakteristiky kvality zahrnuje vhodnost, přesnost, interoperabilitu a bezpečnost. Mezitím jiné rámce mohou upřednostňovat aspekty, jako je kvalita a jednoduchost kódu, škálovatelnost nebo dokonce etické aspekty v systémech založených na umělé inteligenci. Tento rozdíl v jednotlivých aspektech hodnocených na jak existujících a nově vyvíjených produktech je dalším důkazem nemožnosti aplikovat jednu definici na všechen software. Je obrovský rozdíl v softwaru využívaném v kritické infrastruktuře a ve spotřebitelské aplikaci na mobilním zařízení koncového uživatele (Stamelos, Sfetsos, 2007).

Kvantifikace a měření kvality softwaru navíc představují praktické výzvy. Zatímco některé aspekty kvality, jako například výkonnostní metriky, lze objektivně měřit, jiné, jako například spokojenost uživatelů nebo čitelnost kódu, jsou ze své podstaty subjektivní a kvantifikaci se brání. To vede k obtížím při stanovení standardizovaných metrik pro hodnocení kvality softwaru, což dále komplikuje definici kvality (Petrasch, 1999).

Konečně lze tedy říci, že definování kvality softwaru je mnohotvárný problém, jehož kořeny tkví v subjektivitě toho, co kvalita znamená pro různé zúčastněné strany, v dynamické povaze softwarových technologií a v rozmanitosti aplikací a kontextu, v nichž se software používá. Absence všeobecně přijímané definice odráží tyto složitosti a zdůrazňuje potřebu flexibilního, kontextově citlivého přístupu k chápání a dosahování kvality softwaru (Tian, 2005).

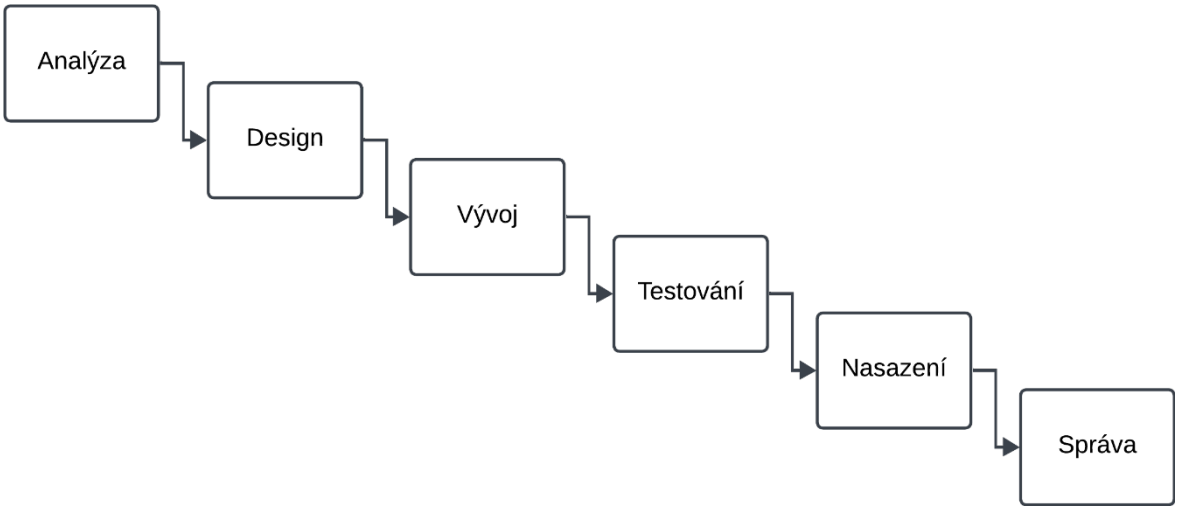
1.3 Modely vývoje softwaru a testování

Historicky se management kvality obvykle soustředil na testování kódu a funkcí až v pozdních fázích vývojového cyklu softwaru, po etapě implementace. Zodpovědnost za vývoj softwaru a za kvalitu dodávaného softwaru přitom spadala pod různé osoby. Oddělení zajišťující kvalitu nebylo zapojeno do fáze implementace, a jeho primárním úkolem bylo ověření, zda se software chová v souladu s požadavky. Týmy zabývající se kvalitou a vývojové týmy často měly odlišné cíle; zatímco cílem vývojového týmu bylo dodat produkt v určeném termínu a rozpočtu, úkolem týmu kvality bylo zajištění kvality produktu, což mohlo ovlivnit rychlost uvedení produktu na trh. V kontextu řízení celého projektu je však dodržení plánovaného data uvedení produktu na trh klíčovým faktorem, což v některých případech vedlo ke zkrácení času vyhrazeného na testování produktu. V tento moment přicházíme s jednotlivými modely testování, které se používají nebo používaly v testování softwaru, v rychlosti si je představíme. Těmi nejvýznamnějšími je model vodopádový, z moderních přístupů si znázorníme SCRUM a celkově přechod na Agile development (Silhavy, 2020).

1.3.1 Vodopádový model

Vodopádový model se vyznačuje lineárním a sekvenčním přístupem k vývoji softwaru. Vyžaduje, aby požadavky byly plně definovány a pochopeny před přechodem do fáze návrhu, což zajišťuje, že každá fáze životního cyklu vývoje je důkladně dokončena před přechodem do další. V tomto modelu začíná testování až po dokončení celého procesu vývoje, přičemž žádná fáze se neprotíná s jinou. Plánování úkolů v rámci jednotlivých fází je přísně kontrolováno, s konkrétními termíny dokončení, aby byla zachována kvalita projektu (Senerath, 2021).

V tradičním vodopádovém přístupu dochází k postupu z jednoho kroku do dalšího až po úplném schválení nebo "zmrazení" aktuálního kroku. Například návrhové činnosti nezačnou, dokud nejsou požadavky pevně stanoveny, a vývoj nezačne, dokud není ukončena fáze návrhu. Tato rigidní struktura může vést ke značným zpožděním a zvýšeným nákladům, protože vady mohou být odhaleny až v pozdní fázi životního cyklu projektu kvůli nedostatečnému včasnému zapojení testovacího týmu. Testeři jsou zapojeni výhradně ve fázi testování, což může vést k neefektivitě a prodloužení lhůt vývoje (Das, Sinha, 2021; Levin, 2019).



OBRÁZEK 2 - WATERFALL MODEL – ZPRACOVÁNÍ AUTOR, 2024, ZDROJ: (LEVIN, 2019)

Když jsou požadavky poskytnuté klientem dokončeny, znamená to začátek životního cyklu vývoje podle vodopádového modelu, bez možnosti úprav, jakmile projekt překročí fázi požadavků. Toto striktní dodržování původních požadavků charakterizuje strukturovanou povahu modelu (Silhavy, 2020).

Mezi výhody vodopádového modelu patří jasné a dobře definované požadavky před zahájením vývoje, dokončení každé fáze vývoje ve stanoveném časovém rámci, jednoduchost implementace, minimální nároky na zdroje a důkladná dokumentace v každé fázi, která zajišťuje kvalitu vývoje. (Das, Sinha, 2021; Dima, Maasen 2018).

Vodopádový model však není bez nevýhod. Problémy v rámci určité fáze často zůstávají nevyřešeny a po jejím skončení se obvykle objevují komplikace. Kromě toho nelze vyhovět případným požadavkům klienta na změnu požadavků, jakmile proces vývoje pokročí za fázi požadavků, což představuje problém při přizpůsobování se vyvíjejícím se potřebám spotřebitelů (Silhavy, 2020).

1.3.2 SCRUM

1.3.2.1 Teorie

Scrum je moderní agilní metoda která je navržena pro rychlé doručování kvalitního softwaru. Scrum jako takový je metodologický soubor, který stanovuje pravidla, aktivity, týmové složení a procesů. Sám o sobě, avšak neudává konkrétní postupy. To je důležité zmínit, jde spíš o metodologický přístup založený na tom, že je následně upraven na konkrétní potřeby společností. To znamená, že jedna společnost může například využívat naprosto odlišné vývojové ceremonie oproti společnosti druhé, která je nemusí mít vůbec (Goericke, 2022; Baumgartner 2021).

Metodika se vyznačuje výraznou orientací na týmovou práci. Podporuje týmy v tom, aby se učily ze zkušeností, samy se organizovaly kolem problémů a neustále reflektovaly své úspěchy a neúspěchy, a tím podporovaly zlepšování. Tato metodika je sice běžně spojována s týmy zabývajícími se vývojem softwaru, ale je použitelná v různých týmových pracovních prostředích, což přispívá k jejímu širokému rozšíření. Je založen na třech pilířích: transparentnosti, kontrole a přizpůsobení. Transparentnost zajišťuje, že každý člen týmu má přístup ke všem potřebným informacím týkajícím se jeho úkolů, což podporuje jasnou komunikaci a definované cíle. Rámec klade důraz na průběžnou kontrolu procesů a artefaktů, aby byl zajištěn soulad s cíli projektu a udržením kvality. Přizpůsobení v rámci Scrumu umožňuje úpravy procesů v reakci na potřeby projektu, což usnadňuje reakci na změny v požadavcích zákazníka nebo ve směřování projektu (Goericke, 2022).

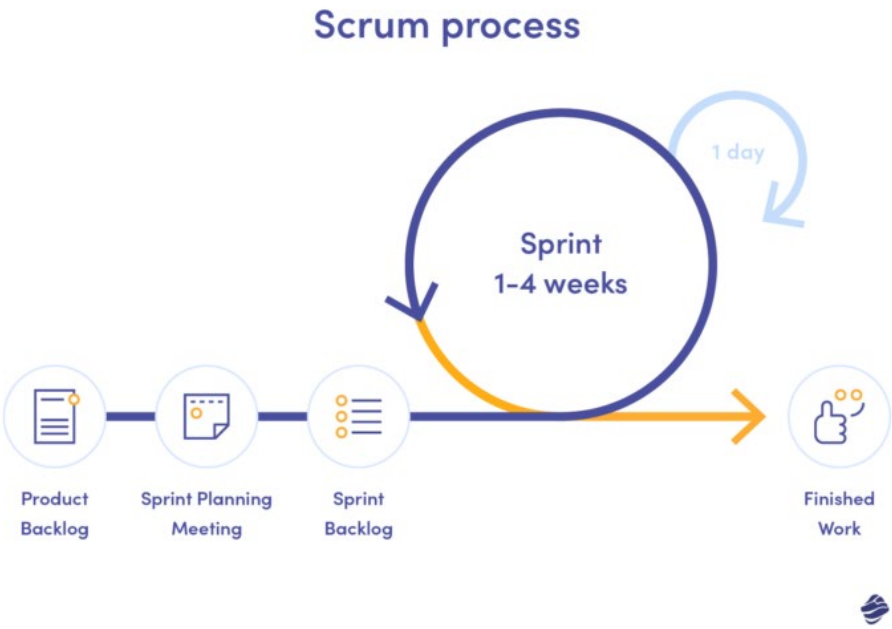
Je potřeba také dát na pravou míru vztah Scrum a Agile přístupy. Mnoho lidí vnímá tyto dvě věci jako rozdílné pojmy. Avšak pravdou je, že oboje je provázáno. Scrum je praktická metodologie využitelná při vývoji softwaru, zatímco Agile je filosofie jako taková. Scrum tuto filozofii ztělesňuje svým přístupem, který se zaměřuje na učení se ze zkušeností a klade důraz na základní postupy namísto zbytečně zdlouhavých procesů (Baumgartner, 2022; Alami, Krancher 2022).

1.3.2.2 Praxe

V praxi Scrum usnadňuje rozdělení složitých úkolů do menších jednodušeji zvládnutelných souborů, které jsou vypracovávány v krátkých cyklech známých jako sprinty, zpravidla trvající 1-4 týdny. V těchto sprintech se následně dokola iteruje několika fází:

1. Revize product a sprint backlogu
2. Plánování sprintu
3. Samotný sprint – zde se iteruje nad fázemi, design, vývoj, testování
4. Sprint review

Tato metoda zajišťuje rychlé dodání hodnoty zákazníkům, podporuje týmovou spolupráci a adaptivní a iterativní zlepšování produktů a procesů (Miquido, 2023; Schwaber, Sutherland, 2020; Alami, Krancher 2022).



OBRÁZEK 3 - DIAGRAM SCRUM METODOLOGIE; MIQUIDO, 2023 DOSTUPNÉ Z: HTTPS://WWW.MIQUIDO.COM/BLOG/FOSTER-GREATNESS-EMBRACE-SCRUM/

1.4 Metody testování

Metodami testování rozumíme jednotlivé strategie a přístupy využívané pro testování softwarových produktů, tak aby splňovali požadavky tak jak jsme si je definovaly v předešlých kapitolách. Obecně lze metody testování dělit podle způsobu kdy a jakým způsobem jsou testy provedeny. Nemůžeme mluvit o hierarchii metod, jelikož každá metodologie je využívána pro jiné účely. Tento celý soubor je zastřešen od testování za pomoci statických tak dynamických testů (Saini, 2023).

Dynamické testování je způsob testování softwaru, který zahrnuje přímé využití softwaru, od instalace až přes spuštění jednotlivých funkcí a vyhodnocování jednotlivých operací. Software je spuštěn v konkrétním prostředí a na něm jsou prováděny testy, za pomoci zadávání vstupů a zkoumání výstupů. Hlavním cílem dynamického testování je zajistit, aby software během instalace i po ní řádně fungoval a aby byla zajištěna jeho stabilita bez výrazných nedostatků (Saini, 2023).

Výhodou dynamického testování je bezesporu testování v rozličných prostředích, kdy jsme schopni odhalovat problémy v rámci výkonnosti aplikace a s ohledem na využívání zdrojů prostředí. Můžeme tedy konkrétně odhalit například úniky výpočetního výkonu, bezpečnostní problémy nebo problémy v samostatném chodu aplikace. Takové problémy není možné odhalit v rámci statického testování. U dynamického testování si můžeme uvést několika příkladů, v rámci této práce se budeme v dalších kapitolách věnovat právě dynamickému testování (O'Regan, 2022):

- White Box Testing – testování s kompletní znalostí kódu
- Black Box Testing – testování bez znalosti kódu
- Funkční testování – testování funkčnosti softwaru

Statické testování je typ testování softwaru, při kterém je software testován bez spuštění kódu. Za účelem nalezení chyb se provádí ruční nebo automatizované kontroly kódu, dokumentů s požadavky apod. Hlavním cílem statického testování je zlepšit kvalitu softwarových aplikací nalezením chyb v raných fázích procesu vývoje softwaru (Hamilton, 2024).

Statické testování zahrnuje ruční nebo automatizované revize dokumentů. Tato revize se provádí v počáteční fázi testování, aby se zachytily chyby v rané fázi životního cyklu vývoje softwaru (SDLC). Zkoumá pracovní dokumenty a poskytuje připomínky k revizi. Říká se mu také nevýkonové testování nebo ověřovací testování. Jako příklady statických testů si můžeme uvést například (Leach, 2019):

- Revize dokumentů – týmu pracovníků je dodán soubor všech žádaných dokumentů k přezkoumání, dokumenty jsou zrevidovány a každý pracovník vypracuje vlastní posudek nezávisle na ostatních, na základě těchto poznatků se přistoupí k přepracování dokumentace, pokud se uzná za vhodnou
- Code review – revize kódové základny softwaru.

- Inspekce/ technical review – nezávislí hodnotitelé zkoumají na základě checklistu software a vypracovávají posudek, většinou jde o technickou inspekci softwaru

Oba přístupy jsou pro QA relevantní a důležité. Je třeba si ale uvědomit, že statické testy často nutně nemusí být prováděny QA specialisty, nýbrž interními, zainteresovanými nebo externími pracovníky. Z určitého důvodu jde o zachování objektivity, jelikož u častého testování, což se může rychle proměnit v rutinní aktivity, především v rámci Agile se můžeme setkat s termínem, kterým je tzv. „QA Bias". Jde o jakousi slepost testerů, která nastává při opakovaném rutinním testování stejného kusu softwaru s minimálními změnami. Následně je velice jednoduché mnohé prvky testovaných součástí přehlížet (Hamilton 2024; Dima 2019).

1.4.1 Manuální testování

Přestože stoupá významnost automatizovaného testování, manuální testování zůstává nepostradatelné a dává nám možnost se na software pohledět napřímo. Lidský pohled na věc je nezbytný pro posouzení všech vlastností testovaného softwaru jako je právě použitelnost, přístupnost a ve výsledku pak celková uživatelská zkušenost. Manuální testování zahrnuje ruční provádění testovacích scénářů bez využití automatizovaných nástrojů s cílem zachytit vady softwaru. Jednou z výhod manuálního testování je flexibilita. Testeři mohou s velikou rychlostí a kadenci upravovat prováděné testovací scénáře a rychle reagovat na změny v softwaru. U automatizovaných testů je často časově náročné je upravovat dostatečně rychle a například pokud se provádí opravy softwaru s vysokou prioritou, tester zásadně využije manuální testování (Patton 2006; Kaner 2001) (Myers, 2015).

Kromě toho je manuální testování nezbytné pro testování scénářů, které je obtížné automatizovat, jako je posuzování vizuálních aspektů aplikace, včetně rozvržení a grafického designu. Hraje také klíčovou roli při testování aplikací, kde se uživatelské rozhraní často mění, což činí údržbu automatizovaných testů nákladnou a časově náročnou. (Fewster a Graham, 1999).

Manuální testování je však ze své podstaty pomalejší a náchylnější k lidským chybám, zejména v opakujících se a nudných testovacích scénářích. Proto je nejefektivnější, když je doplněno automatizovaným testováním, které využívá silné stránky obou přístupů. (Desikan a Ramesh, 2006; Leach, 2019).

1.4.2 Automatizované testování

S tím, jak se vývoj softwaru rychle posouval bylo přirozené, že vznikla potřeba testování zautomatizovat. Automatické testování je jednoduše řečeno programaticky zajištěné testování za pomoci různých nástrojů. Podstata automatizovaného testování spočívá v jeho schopnosti provádět opakované úlohy s vysokou přesností, čímž se minimalizuje prostor pro lidskou chybu. To je výhodné zejména při regresním testování, kdy je třeba často provádět stejnou sadu testů, aby se odhalily případné nezamýšlené důsledky změn provedených v kódové základně (Dustin, 2002).

Automatizované testování navíc usnadňuje kontinuální integraci/kontinuální nasazení (CI/CD) a umožňuje týmům častěji a efektivněji integrovat změny kódu. Tato integrace

umožňuje včasné odhalení chyb, což výrazně snižuje náklady, a především čas potřebný k dodání softwaru. Právě kontinuální integrace/ nasazení je jedním z prvkům, kterým se vyznačuje moderní vývoj softwaru, který je silně dynamický, a tedy potřebuje i robustní systém kontroly kvality. (Fowler a Foemmel, 2006).

Implementace automatizovaného testování však není bez svých výzev. Počáteční nastavení vyžaduje značné investice z hlediska času a zdrojů. Kromě toho je účinnost automatizovaného testování závislá na kvalitě testovacích skriptů a přizpůsobivosti testovacího rámce změnám v uživatelském rozhraní nebo funkčnosti aplikace. Je potřeba aby testovací nástroje byly dobře pochopeny a samozřejmě tyto nástroje musí být kompatibilní s prostředím, ve kterém pracujeme. Typů nástrojů, pro automatické testování je mnoho, a v různých formách. Některé mají vlastní grafické uživatelské rozhraní, jiné fungují čistě z příkazového řádku. Pomocí některých testujeme frontend a další zase slouží pro backend. Je proto potřeba před implementací automatizovaného testování se opravdu objektivně podívat na celý náš projekt a správně navrhnout kombinaci testovacích nástrojů tak, aby se nepromarnily žádné zdroje. To je komplikovaná disciplína a je jednou z hlavních aktivit QA specialistů (McDonough, 2021).

Je důležité si však uvědomit, že automatizované testování není nic, co by fungovalo samo o sobě, a zmizela tak potřeba testování manuálního. Automatizace nenahradí analýzu a vědomosti pracovníků. Je nutná vhodná kombinace jak manuálního, tak automatizovaného testování. Navzdory těmto problémům jsou výhody automatizovaného testování – jako je vyšší přesnost, rychlejší doba provedení a možnost paralelního provádění testů nepostradatelným nástrojem v moderních metodikách zajištění kvality softwaru (Dustin, 2002; Jose, 2021).

1.4.3 Strukturální testování (white box – code testing)

Metodou white box, nebo také strukturální testování rozumíme testování takové, které pracuje napřímo s kódovou základnou softwaru. Tester napřímo zná strukturu, design a způsob implementace prvku, který testuje. Účelem testování za kompletní znalosti kódu je ověřování vnitřního fungování aplikace. Testování se zaměřuje na samostatnou logiku zanesenou do kódu. Zaměřujeme se na samotné funkce a spustitelné příkazy v aplikaci jako takové. Tyto testy jsou z logiky věci prováděny samotnými programátory (O'Regan, 2022).

Výhodou unit testování je jeho přesnost, jelikož pracuje přímo s kódem aplikace a může tak odhalit chyby, které se nedají jednoduše zjistit bez jeho znalosti. Můžeme se například setkat s etickými hackery, kteří testují zabezpečení aplikace po tom, co důkladně zrevidují kód aplikace. Můžeme odhalit matematické problémy ve funkcích apod (O'Regan, 2022).

White box testing má však i své nevýhody. Neověřuje správnost samotných specifikací, místo toho se zaměřuje pouze na vnitřní logiku programu, aniž by ověřovalo jeho soulad se zamýšlenými specifikacemi. Nemůže bez podrobných informací o specifikaci odhalit chybějící cesty nebo chyby citlivé na data. Navíc není možné provést všechny možné logické cesty programem kvůli astronomicky velkému počtu potenciálních testů (Stamelos, Sfetsos, 2007).

Dalším aspektem white-box testování je použití softwarových nástrojů pro testování pokrytí, jako jsou debuggery, které pomáhají sledovat chování programu. To umožňuje testerům zjistit, zda byl konkrétní příkaz proveden a zda bylo výsledné chování

očekávané. To sice usnadňuje lokalizaci a opravu chyb, ale má to omezený rozsah, protože nelze snadno odhalit chyby v návrhu jako takovém (Stamelos, Sfetsos, 2007).

1.4.4 Acceptance testing

Akceptační testování je proces, při kterém se ověřuje akceptace prvků, vlastnosti a požadavků na dodávaný software, a to z hlediska, zda splňuje tyto požadavky ze strany businessu a potažmo ze strany koncových uživatelů. V mnoha případech se akceptační testování provádí v koncových fázích vývoje, jelikož se snaží ověřit korelaci testovaných součástí softwaru společně s cíli a záměry společnosti. Je však nutné se na akceptační testování patřičně připravit, a to se naopak může dít již v prvotních fázích, jelikož zde je již možno testerem, nebo jiným pracovníkem odhalit nějaké odchylky od cílů a akceptačních kritérií (de Oliveira, 2023).

Metodika akceptačního testování se často velmi liší a zahrnuje jak manuální, tak automatizované přístupy, které vyhovují různým potřebám projektu. Bez ohledu na přístup závisí úspěch akceptačního testování na jasných a měřitelných akceptačních kritériích, včasném zapojení zúčastněných stran a vývoji realistických testovacích scénářů, které napodobují skutečné podmínky používání. Důkladná dokumentace a otevřená komunikace v průběhu celého procesu testování jsou nezbytné pro řešení problémů a zajištění toho, že software skutečně splňuje potřeby uživatelů a kritérií společnosti (de Oliveira, 2023).

Souhrnně řečeno, akceptační testování je nezbytné pro ověření, že softwarový produkt nejen funguje tak, jak má, ale také splňuje provozní požadavky a obchodní cíle. Díky efektivnímu plánování, provádění a spolupráci slouží akceptační testování jako zásadní krok k dosažení dokonalosti softwaru a zajišťuje, že je konečný produkt připraven k úspěšnému nasazení a vydání na trh (Tian, 2005; Rose, 2022).

1.4.5 Regresní testování

Regresní testování zajišťuje, že dříve vyvinutý a otestovaný software funguje správně i po jeho změně nebo propojení s jiným softwarem. Změny mohou zahrnovat aktualizace, vylepšení nebo opravy chyb. Hlavním cílem regresního testování je identifikovat chyby, které mohly být zavedeny do dříve stabilních funkcí softwaru.

Regresní testování lze provádět ručně nebo automatizovaně, přičemž automatizované regresní testování je pro nás zajímavější díky své efektivitě a možnosti provádět testy opakovaně a neustále, jejich rychlost nám následně pomáhá šetřit zdroje společnosti (Botto-Tobar, 2022).

Efektivní strategie regresního testování zahrnuje výběr správných testovacích scénářů, které mají být opakovaně prováděny. Není nutné pokaždé spouštět všechny testovací scénáře, spíše by měl být vybrána vhodný soubor scénářů relevantní pro nedávné změny nebo ty, které pokrývají kritické funkcionality aplikace. Rozsah a četnost regresního testování závisí na povaze projektu a dostupných zdrojích. Navzdory problémům při správě sady testů a zajištění včasného provedení testů jsou přínosy nepopiratelné (Botto-Tobar, 2022; Williams, 2005).

1.4.6 Exploratory testing

Na rozdíl od tradičních testovacích metod, které se ve velké míře spoléhají na předem připravené testy, exploratory testování klade důraz na kombinaci nauky o softwaru a neustálém prozkoumávání. Tato metoda není jen o hledání chyb, ale je to postup založený na zvědavosti, který vyžaduje, aby se tester neustále přizpůsoboval a učil se z testovaného softwaru (Hendrickson, 2013).

Jádrem exploratory testování je myšlenka, že testování softwaru se podobá zkoumání neprobádaných území. Jde o proces, při kterém se tester podobně jako průzkumník pouští do hlubin softwaru, aby nejen potvrdil to, co je známo, ale aby odhalil neznámé. Testování je založeno na dovednosti testera pozorovat a interpretovat chování softwaru. Vyžaduje určitou úroveň pozornosti a vnímavosti, aby si všiml anomálií a vzorců, které mohou naznačovat skryté problémy. Kromě toho vyžaduje myšlení, které je otevřené překvapením a ochotné sledovat, kam software vede, což je obzvláště efektivní v agilních a rychlých vývojových prostředích, kde se požadavky a funkce mohou rychle vyvíjet (Hendrickson, 2013).

Jedna z hlubokých předností exploratory testování spočívá v jeho přizpůsobivosti. Lze jej použít v kterékoli fázi vývojového cyklu, což z něj činí neocenitelný nástroj pro průběžný feedback a potencionální zlepšení softwaru. Tester je schopen vnést do projektu vlastní poznámku na uživatelský zážitek z celé aplikace nebo například zhodnotit vizuální líbivost a své poznatky rychle předat vývojářům, prodiskutovat a případně zanést do celkové podoby softwaru. Doplňuje ostatní metody testování tím, že vyplňuje mezery, které skriptované testování nemusí pokrýt, zejména v oblastech, kde není známo chování softwaru ve složitých nebo nepředvídaných scénářích (Hendrickson, 2013).

1.5 Softwarové chyby

Čím se rozumí softwarové chyby, „bugy"? Softwarové chyby představují nesrovnalosti mezi zamýšleným a skutečným chováním testovaného softwaru. Tyto anomálie, často označovány jako vady, chyby, problémy nebo poruchy, vznikají z různých zdrojů, včetně lidské chyby při kódování, během vzniklých chyb v návrhu nebo problémy vzniklé v chodu softwaru jako takovém, ať už jde o nechtěné interakce mezi jednotlivými součástmi, chyby v logice kódu nebo slepé cesty. Pochopení projevů chyb v softwaru má zásadní význam pro vývoj účinných strategií pro zajištění kvality, testování, ladění a v konečném důsledku pro zvýšení spolehlivosti a výkonnosti softwaru. Dopad softwarových chyb se značně liší, od drobných nepříjemností až po závažné problémy, které mohou ohrozit bezpečnost, integritu dat a uživatelský komfort. Řešení softwarových chyb zahrnuje komplexní přístup, který zahrnuje strategie prevence, detekce a nápravy (O'Regan, 2022):

- Prevence je minimalizovat výskyt chyb prostřednictvím důsledného návrhu a vývoje softwaru, včetně revizí kódu a dodržování vhodných konvencí při vývoji
- Detekce zahrnuje různé metodiky testování určené k odhalení chyb v různých scénářích.

- Náprava se zaměřuje na opravu zjištěných chyb a na zajištění toho, aby prostřednictvím regresního testování bylo zamezeno chybám tvořeným během dodatečných změn apod (Shatnawi, Alazzam, 2022).

1.5.1 Ekonomické a praktické aspekty

Pochopení ekonomických přínosů zajištění kvality a efektivní správy chyb je klíčové. Investice do procesů a nástrojů pro zajištění kvality může výrazně snížit dlouhodobé náklady spojené s opravou chyb, zejména těch, které byly zjištěny později ve vývojovém cyklu nebo po vydání produktu. Efektivní správa chyb navíc přispívá ke spolehlivosti softwaru a spokojenosti uživatelů, které jsou rozhodující pro udržení konkurenceschopnosti. Je potřeba si uvědomovat, že kvalitně navržené a samozřejmě provedené QA procesy mohou společnostem ušetřit významné zdroje. Mnoho společností se však potýká s určitou skepsí vůči větším investicím do procesů kontroly kvality. Významnost tohoto odvětví se nepochybně zvětšuje, avšak skepse stále přetrvává. Kvalitní kontrola kvality softwaru může ušetřit také mnoho času, jelikož pokud je v testovacím procesu zařazeno samostatné testovací oddělení, ušetří čas vývojovému oddělení chyby hledat, analyzovat a až teprve následně opravovat. Vhodná spolupráce testerů a vývojářů šetří čas a tím i zdroje (Capers, Olivier, 2012).

1.5.2 Priorita x severita

Během testování je důležitým pojmem priorita a severita. Jsou zpravidla využívány pro určení závažnosti chyby a na základě toho se vyhodnotí další postupy jako je urgentnost řešení vyskytlé chyby nebo její zařazení do vývojového rámce. Tester určuje tyto dva pojmy na základě své vlastní expertizy a musí je kvalitně komunikovat (Tozzi, 2023).

Priorita je pojem, který určuje významnost nalezené chyby a na základě jejího určení se stanoví urgentnost řešení problému. Urgentnost se určuje primárně dle potřeb zákazníka. Tester sděluje vývojářskému týmu, jak rychle je problém potřeba vyřešit, tedy problém se posouvá v příčkách posloupnosti všech problémů zařazených k řešení (Tozzi, 2023).

Severita je výraz pro určení dopadu, který má závada nebo problém na fungování softwarové aplikace. Je to míra toho, jak vážně chyba ovlivňuje funkčnost, použitelnost nebo celkové fungování softwaru. Závažnost se obvykle klasifikuje nezávisle na četnosti nebo snadnosti reprodukce chyby a zaměřuje se výhradně na potenciální důsledky chyby (Bose, 2023).

2 Vývoj softwaru a QA

2.1 Životní cyklus

Životní cyklus vývoje softwaru = „Software Development Life Cycle = SDLC" je rámec, který definuje proces používaný organizacemi při vývoji dodávaného softwarového produktu. SDLC popisuje jednotlivé fáze procesu vývoje softwaru od počáteční koncepce až po nasazení a údržbu, čímž zajišťuje, že kvalitní software je dodáván dostatečně rychle a za co nejlepší možné kvality. SDLC má standartně několika fází, které si nyní uvedeme (Silhavy, 2020):

Analýza požadavků: Tato počáteční fáze zahrnuje shromažďování a analýzu potřeb koncových uživatelů, aby bylo zajištěno, že nový systém splní jejich očekávání. Ve většině případů je zapojena téměř celá organizace od business developmentu až po testery.

Návrh: V této fázi se vytváří celková architektura a návrh softwaru, včetně databází, jednotlivých součástí softwaru anebo například uživatelského rozhraní. Fáze návrhu převádí požadavky na plán toho, jak bude daný software ve výsledku vypadat. I v této fázi mohou být zainteresování testeři, jelikož dokáží zhodnotit, jak moc testovatelná aplikace v budoucnu bude a mohou tak svou zpětnou vazbou ovlivnit celkový produkt.

Implementace: Zde probíhá vlastní vývoj softwaru. Vývojáři používají programovací jazyky k vývoji nebo úpravám stávajícího softwaru na základě návrhových dokumentů a specifikací z předchozí fáze.

Testování: Jakmile je software vyvinut, prochází důkladným testováním, aby se zjistily a opravily případné chyby. Testování zajišťuje, že software splňuje všechny stanovené požadavky a funguje podle očekávání ve všech zamýšlených prostředích, tak jak jsme si definovali v předchozích kapitolách

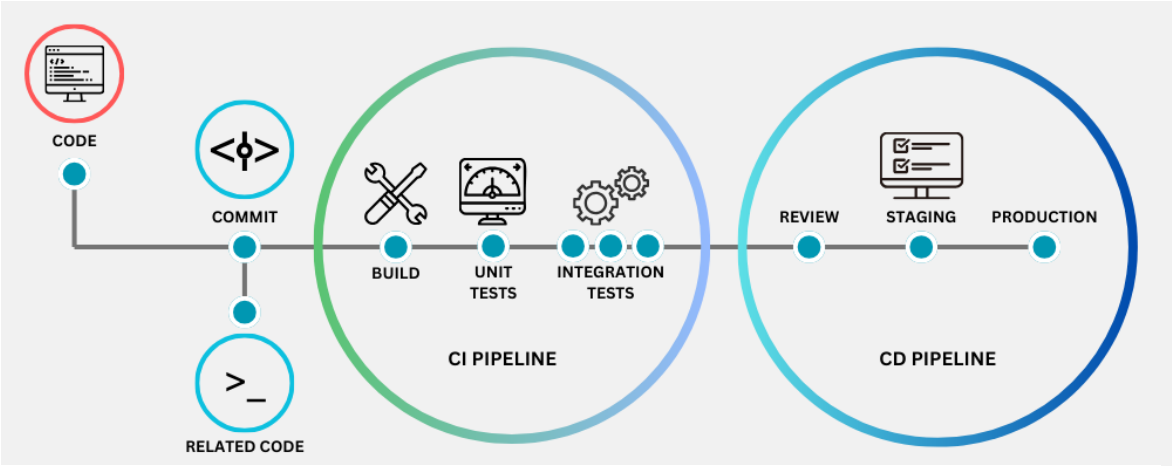
Nasazení: Po testování je software nasazen do produkčního prostředí, kde k němu mají přístup koncoví uživatelé. Nasazení může probíhat v několika fázích, ale rozhodné je, že na konci této fáze je software v rukou koncových uživatelů.

Údržba: Závěrečná fáze zahrnuje provádění aktualizací, aby se zajistilo, že software bude i nadále vyhovovat potřebám uživatelů, a také opravu případných chyb, které se objeví po nasazení. V tento moment testeři provádějí post-mortem testování. Tedy snaží se replikovat problémy, které vznikly v produkci a urychleně je komunikují s vývojářským týmem kdy se společně snaží je co nejrychleji nasadit.

Co se různých modelů SCLD týče, můžeme si připomenou model vodopádový a metodiky v rámci Agile proces je stejný (Tuteja, 2012).

2.2 Neustálé zlepšování – CI/ CD + DevOps

Dnešní dynamicky se proměňující svět vývoje softwaru si klade neustále se zvyšující nároky na dodávku softwaru, která je častější a s větší kvalitou. Ale přesně tyto dva nároky jsou v neustálém konfliktu. Neustálé změny a vylepšení softwaru jsou podstatně náchylnější k přinášení nových a nových chyb. To v důsledku znesnadňuje snahy o kvalitní přetestování a zaručení stability celého systému. Proto bylo nutno přinést řešení, a to se nachází v podobě CI/CD – kontinuální integrace a kontinuální nasazení. Přináší automatizované řešení mnoha problémů, které je podchyceno kvalitním monitoringem. Cílem CI/CD je usnadnit práci vývojářů aby mohli zajistit neustálou dodávku nových a zároveň otestovaných součástí softwaru. Tzv. CI/CD pipeliny dokáže automaticky sestavit a přeložit celý kód který i následně nasadí na specifikované prostředí, ať už jde o prostředí určené pro testování nebo přímo na produkci. Dokáže zároveň postupně nasazovat na jedno prostředí a po dokončení automatizovaných testů sama vyhodnotí výsledky a nasadí následně software na produkci pro užívání koncovým uživatelům. Je sestavena z mnoha kroků, které jsou definovány vývojáři a dalšími pracovníky, kteří mají podíl na vývoji jako jsou testeři, pracovníci infrastruktury (komunikace se servery, databázemi apod.) (RedHat, 2022).



OBRÁZEK 4 - SCHÉMA CI/CD, GOLUBEV 2022, DOSTUPNÉ Z: HTTPS://TESTRIGOR.COM/BLOG/WHAT-IS-CICD/

2.3 Test driven development

Test driven development, zkráceně „TDD“. Se v posledních letech stal populárním přístupem k vývoji softwarových produktů. Ve zkratce se tento přístup vymyká klasickému vývoji, kde se vyvíjí, implementuje a až teprve následovně se testuje a místo toho využívá praxe, kdy se nejdříve vytváří na základě požadavků projektu testy, které defacto reprezentují vyvíjenou funkci nebo modul softwaru. Tento test samozřejmě nejdříve přirozeně vyhodnotíme jako neúspěšný, jelikož nově vyvíjený kód zatím ještě nebyl implementován. Druhou fází je tedy napsání jednoduchého kódu tak, aby test byl vyhodnocen jako úspěšný. V poslední fázi, po tom, co je test úspěšně vyhodnocen se vrátím k vytvořenému kódu a zrevidujeme ho, tzv. refactorujeme (Maximilien, Williams, 2024; George, Williams, 2004).

Celý proces se dá představit na jednoduchém příkladu:

„Představte si, že stavíte stavebnici ze známých kostiček, ale místo toho, abyste se rovnou pustili do spojování dílků podle obrázku na krabici, nejprve se podíváte do návodu, abyste zjistili, jaký má být další krok. Pak tento díl postavíte, zkontrolujete ho podle návodu, zda je správný, a teprve potom přejdete k dalšímu kroku. Postupně tak sestavíte celou stavebnice. Vývoj řízený testy (TDD) je něco podobného, ale pro programování, rozdíl je ještě takový, že návod si často napíšete sami."

2.4 Behaviour driven development

Vývoj řízený chováním (Behaviour-Driven Development, BDD) je rozšířením vývoje řízeného testy (TDD), které klade důraz na spolupráci mezi vývojáři, odborníky na kontrolu kvality a netechnickými zainteresovanými stranami. Tento přístup podporuje týmy v používání jednoduchého jazyka specifického pro danou oblast k definování chování softwaru, což zajišťuje jasnost a porozumění všem členům týmu. BDD se zaměřuje spíše na zkušenosti uživatelů a chování systému než na technickou implementaci, což z něj činí vysoce efektivní metodiku v prostředí agilního vývoje softwaru. BDD se točí kolem vytváření uživatelských příběhů a s nimi spojených scénářů. Uživatelský příběh je definice funkce softwaru napsaná z pohledu koncového uživatele, která se zaměřuje na hodnotu, kterou funkce přináší zákazníkovi. Scénáře naproti tomu poskytují konkrétní příklady toho, jak by se měl software chovat v různých situacích, často vyjádřené ve formátu Given-When-Then. Tento formát je užitečný pro definování jasných kritérií přijatelnosti funkcí před zahájením vývoje (Cucumber, 2024).

K BDD je nutno dodat, že se dá velice jednoduše aplikovat společně s TDD přístupem. U BDD jde spíše o styl zápisu testovacích scénářů zatímco u TDD jde o přístup k vývoji softwaru jako takovému. V rámci praktické části této práce se bude využívat právě kombinace těchto dvou přístupů. Týmy mohou využívat formát "dáno-když-tedy" pro tvorbu testovacích scénářů, které jsou snadno pochopitelné a přehledně strukturované. Tento přístup umožňuje inženýrům efektivně transformovat obchodní požadavky na funkční software a zlepšuje spolupráci s všemi zainteresovanými stranami projektu. Tento přístup rovněž usnadňuje technicky méně zkušeným zaměstnancům orientaci v dění kolem softwarového projektu. Implementací behaviorálně řízeného vývoje (BDD) do testovacího procesu je možné výrazně posílit spolupráci mezi klíčovými účastníky a zvýšit transparentnost celého vývojového procesu. Pojďme se podívat na některé základní rysy BDD testování (Matsinopoulos, 2020):

- Testy jsou psány v přístupné a popisné formě anglického jazyka
- Testy se primárně zaměřují na uživatelské požadavky
- Testy jsou organizovány do specifikačních souborů, což zefektivňuje vývoj

Před zahájením projektu se klíčový stakeholder setkává s vývojáři a testery, aby probrali všechny produktové požadavky, které jsou poté zformulovány do scénářů. Tyto scénáře jsou pak připojeny v nástrojích pro automatizované testování jako funkční soubory, reprezentující automatizované akceptační testy. Každý BDD scénář obvykle obsahuje tři klíčové prvky: Given (udává kontext). When (znázorňuje provedení akce), Then (určuje výsledek předešlé akce). Pro dodatečné detaily ve scénáři lze přidat deskriptor "And", který umožňuje vložení dalších informací do každého z kroků (Poliarush, 2022).

Příklad BDD testovacího scénáře za využití Gherkin syntaxu:

```
Feature: Login
  As a new user
  I want to log in to the website
  So that the system can remember my data

  Scenario #1: Successful Log in to the website
    Given A user brings up the login pop-up
    When A user clicks Sign-in
    And A user enters a valid email <email> and password <password>
    And A user clicks Sign-in
    Then A user should be successfully logged into the site

  Scenario #2: Unsuccessful Log in to the website
    Given A user brings up the login pop-up
    When A user enters an invalid email <email> and password <password>
    And A user clicks Sign-in
    Then A user should not be successfully logged into the site
```

OBRÁZEK 5 - PŘÍKLAD BDD TESTOVACÍHO SCÉNÁŘE, DOSTUPNÉ Z: HTTPS://TESTOMAT.IO/BLOG/WRITING-BDD-TEST-CASES-IN-AGILE-SOFTWARE-DEVELOPMENT-EXAMPLES-BEST-PRACTICES-TEST-CASE-TEMPLATES/

PRAKTICKÁ ČÁST

3 Výchozí stav společnosti

V praktické části práce jsem se věnuji postupnému zavádění procesů kontroly kvality při vývoji softwaru ve společnosti GoodAccess s.r.o. Společnost se zabývá vývojem SaaS řešení pro Zero Trust Network (ZTNA). Zprvu bylo potřeb vyhodnotit a důkladně zanalyzovat výchozí stav co se zavedených procesů jak ve vývoji software, tak v kontrole kvality samotné týče. Bylo tedy potřeba vyhodnotit v jakém stavu se tyto aspekty nachází, následně je zdokumentovat a se stakeholdery se vymezí chtěný budoucí stav, tedy cíl.

Praktická část je postavena na přímé zkušenosti, jelikož působím jako kontraktor v rámci vykonávání procesů kontroly kvality a zodpovídání za konečnou kvalitu vyvíjeného softwaru, který je dodáván koncovým uživatelům. K vypracování jsem využil praktické znalosti nabyté zkušenostmi, kdy se oboru věnuji šest let a zároveň využívám teoretické znalosti z udávaných pramenů. Dalším zdrojem pro vypracování byly rozhovory s pracovníky společnosti, především pak s těmi, se kterými jsem během vypracování interagoval nejvíce. Konkrétně pak jde především o jednotlivé vývojáře, scrum mastera a v neposlední řadě také product owneři.

Zároveň jsem stavěl na již etablovaných postupech, které ve firmě byly již zajištěny, a proto jsem je využil jako zdroj pro správné nastavení postupů a procesů, k tomuto účelu posloužila existující dokumentace a informace od pracovníků.

3.1 Popis společnosti

Společnost GoodAccess s.r.o. je kyberbezpečností firma zabývající se nabízením uceleného SaaS řešení zaměřené na jednoduché nasazení zero-trust architektury (ZTNA). Hlavním cílem společnosti je zaměření se na malé a střední společnosti a nabízet jim rychle a efektivně zabezpečit přístup k firemním sítím, datům nebo systém z jakéhokoli místa a kdykoliv.

Společnost zastává názoru, že možnost zabezpečeného přístupu do interních zdrojů by měla mít každá společnost i při omezených zdrojích. V dnešním moderním IT podnebí se hrozby násobně množí a každý by měl být spolehlivě zabezpečen. Díky spolehlivosti, jednoduchosti nasazení a použití služby společnosti GoodAccess využívá více jak 1300 zákazníků po celém světě.

Zero Trust Network je síť založená na principu "nulové důvěry", jde o bezpečnostní model, který funguje na předpokladu, že žádné zařízení ani uživatel by neměly být automaticky důvěryhodné, a to bez ohledu na to, zda se nacházejí uvnitř nebo vně firemní sítě. Princip nulové důvěry vyžaduje, aby každý uživatel a zařízení prošly ověřením a autentizací před získáním přístupu k jakýmkoliv firemním zdrojům. Nabízená řešení sestávají z několika klíčových funkcí a technologií, které díky GoodAccess řešení umožňují efektivní implementaci zero trust architektury:

Centralizovaná kontrola – Umožňuje správu uživatelských účtů a jejich práv, definici chráněných systémů, dohled nad síťovou aktivitou a zabezpečení zařízení z jednoho webového rozhraní. Automatizace a škálovatelnost – Automatizace rutinních úkolů, definice přístupových práv a blokace nebezpečných zařízení, což umožňuje rychlé škálování. Dedikovaná cloudová brána – Klíčová komponenta pro správu a bezpečnost,

která zajišťuje, že všechny přístupy jsou kontrolovány a zabezpečené. Multifaktorová autentizace – Posiluje bezpečnost tím, že vyžaduje více metod ověření totožnosti uživatele a zajišťuje lepší sledování a reakci na bezpečnostní incidenty. Device Posture Check – Díky nastavení jednotlivých politik, mají administrátoři možnost omezit přístup do systému, jako je verze operačního systému nebo aktivní antivirus na zařízení. Filtrování DNS a Kontrola přístupu – Zajišťuje, že všechny žádosti o přístup jsou bezpečné a legitimní, což minimalizuje riziko útoků.

3.2 Proces vývoje

Procesem vývoje rozumíme v rámci společnosti GoodAccess to, jak je celkově zpracováván projekt od návrhu po nasazení. Projektem jsou rozuměny větší celky obsahující inovaci produktu, jsou to nové funkce, které mají větší dopad na celkový produkt. Zároveň jsou vyvíjeny menší úpravy a funkce, ty však nevnímáme jako samostatné projekty a nevyžadují projití stejným procesem jako celé projekty. Společnost vyvíjí produkt na několika frontách. Aplikace funguje ve webovém prostředí, na mobilních zařízeních a na desktopových strojích. Zároveň se také spravuje webová prezentace společnosti. V rámci této práce se budeme věnovat vývoji a testovacím procesům v rámci aplikace fungující ve webovém prostředí.

Ve společnosti GoodAccess je proces softwarového vývoje založen na metodice Agile Scrum, která je známá svou flexibilitou a adaptabilitou. Tento přístup umožňuje týmu rychle reagovat na změny a efektivně spolupracovat na vývoji produktů které společnost nabízí. GoodAccess se zaměřuje na dynamický a rychlý vývoj s pravidelným nasazováním úprav, oprav a inovací, což zajišťuje neustálé zlepšování a inovace. Tým pracuje ve čtrnáctidenních sprintech. Každý sprint začíná plánovacím setkáním, kde se definují cíle sprintu a úkoly, které je třeba dokončit. Tyto úkoly jsou vybírány z backlogu, který je pravidelně aktualizován a na základě potřeb zákazníků a strategických cílů společnosti je neustále prioritizován. Samostatnou aktivitou je pak fáze objevovací. Tato část není vyloženě součástí sprintu a její výsledky jsou do sprint backlogu a následně do samotných sprintů teprve zařazeny.

3.2.1 Objevovací fáze (Discovery)

Projekty, které jsou v rámci webové aplikace vytvářeny procházejí procesem vývoje. Počátek je objevovací fáze (Discovery), v této fázi vezmou zainteresované strany podnět pro inovaci produktu a za pomoci brainstormingových metod jsou objevovány možnosti řešení nebo úpravy řešení stávajícího. Můžeme zmínit například metody rychlých nápadů nebo myšlenkové mapy. Všechny nápady a podněty jsou následně řádně dokumentovány do jednotného dokumentu, který zahrnuje informace o účastnících, o součástech aplikace, kterých se inovace může dotýkat, datum sezení a další podstatná metadata. Tento dokument slouží jako základní kámen pro další fáze vývojového procesu a je zásadní pro zajištění, že všechny nápady jsou řádně zváženy a hodnoceny. Proces objevování má několik iterací, dokud není dostatečně naplněn vhodnými kritérii, která se mohou následně rozvinout do přesnější podoby v dalších fázích vývoje. Hlavním cílem této fáze je prozkoumat a identifikovat nejlepší možné řešení, které by mohlo být implementováno v rámci produktu.

Discovery fáze je zásadní pro úspěch projektů v GoodAccess, neboť zajišťuje, že vývoj produktu je řízen skutečnými potřebami trhu a očekáváními uživatelů. Díky pečlivému prozkoumání, brainstormingu a dokumentaci může společnost efektivně přistupovat k

inovacím a zajišťovat, že každý nový produkt nebo funkce je relevantní, inovativní a hodnotný pro její zákazníky.

Po ukončení této objevovací fáze a v případě, že je kladně vyhodnocena, přistupuje tým k zařazení do produktového backlogu. Po vyhodnocení časového rámce, kdy bude vhodné nový projekt začít vyvíjet je následně přesunut do plánovací fáze a poté zařazen do sprint backlogu a na projektu se začíná pracovat ve zvoleném sprintovém období. Projekt není nutné dokončit v jednotlivých sprintech, je podstatné že se projekt celkově hýbe směrem k dokončení vývoje, v rámci sprintu se spíše vyhrazuje potřebný čas pro práce na projektu, avšak vyloženě očekávání dokončení projektu není nutné, samozřejmě záleží na velikosti projektu a plánovací tým se vždy snaží projekty nastavit tak aby byly realisticky splnitelné během jednoho sprintovacího období.

3.2.2 Plánovací fáze

Další fází v rámci vypracování projektu ve společnosti GoodAccess je fáze plánovací. Do této fáze se dostáváme po zdárném zařazení do product backlogu a následně se zařadí do sprint backlogu. Tato fáze zahrnuje přesun nápadů a požadavků z produktového backlogu do podoby takové, kdy je projekt připraven na implementaci a je tudíž zařazen do sprintového backlogu, kdy se následně vyhodnotí vhodný moment pro spuštění projektu v daném sprintu.

V rámci metodologie SCRUM, kterou GoodAccess využívá se plánovací fáze dá také rozdělit na high-level plánování a následně sprint-level plánování. Vezmeme tedy projekt z produktového backlogu a snaha bude projít high-level plánování, které nám dovolí připravit projekt pro sprint. Při high-level plánování se tým opírá o nápady nastřádané během Discovery fáze. Tým složený z vývojářů, product ownera, scrum mastera a dává dohromady celkový cíl projektu. Do tohoto plánování jsou zařazeny všechny požadavky počínající business požadavky pro koncové uživatele, přechází se přes technické požadavky nutné pro implementaci až po design a samotné fungování vyvíjených funkcionalit. Cílem plánovací fáze je příprava projektu do formy, kdy je kompletně připraven pro implementaci. To obnáší důkladné zvážení každého požadavku, jeho prioritizaci a zařazení do jednotlivých částí projektu a plánu vývoje jako takového. V tomto bodě se také vyhodnocuje vhodný okamžik zařazení projektu do sprint backlogu a následně do samotného sprintu jako takového. Tento výběr je zásadní, jelikož musí být reflektována vytíženost vývojového týmu a samozřejmě také strategické cíle společnosti.

Po uceleném vyhodnocení projektu a zařazení do sprint-backlogu se přechází do sprint level plánování, což je mikro pohled na konkrétní postavení projektu v samostatném sprintu. V rámci tohoto plánování je kladen důraz na detailní rozpracování požadavků, včetně technických specifikací a designu v rámci časových náročností, tak aby se bezproblémově naplnily požadavky v časových bariérách sprintu. Tým společně rozebírá každý aspekt projektu, od business požadavků pro koncové uživatele až po specifikace jednotlivých funkcionalit a jejich uživatelských scénářů. Při tomto plánování se také přihlíží k rizikům a možným komplikacím, což umožňuje proaktivní přístup k řízení projektu v rámci sprintu, kdy se může také objevit případ re-prioritizace a přesunutí celého nebo součástí projektu do dalších sprintových oken.

Je zásadní, aby každý člen týmu měl jasně definovanou roli a odpovědnost. Product owner má na starosti komunikaci s ostatními stakeholdery a zajišťuje, že tým má jasné porozumění požadavkům z obchodní perspektivy. Scrum master zajišťuje plynulost

procesů a eliminuje překážky, které by mohly narušit efektivitu práce týmu. Vývojáři se zaměřují na technickou stránku realizace požadavků a jejich převod do funkčního softwaru.

Implementační fáze je defacto fáze samotného vývoje požadovaných funkcionalit, úprav nebo kompletně nových dodatků do softwarového produktu. Tato fáze začíná v momentě, kdy je dokončeno plánování a projekt je zařazen do sprint backlogu. Po určení časového rámce je projekt vhodně zařazen do samostatného sprintu jako takového. Jedno sprintové období ve společnosti GoodAccess trvá čtrnáct dní, na jehož konci probíhá vyhodnocení sprintu. V průběhu sprintu je tým zaměřen na dosažení specifikovaných cílů projektu v rámci doby sprintu, pokud se tak nestane nastanou dva momenty. Tým si provede retrospektivní vyhodnocení a zjistí a zanalyzuje proč nebyly naplněny cíle během jednoho sprintu, následně je projekt aktualizován, co se priorit týče a dodatečné nedokončené cíle jsou zařazeny do dalšího sprintu. Tato varianta není vyloženě negativní, ale poukazuje na větší rozsah projektu s neočekávanými překážkami.

Implementační fáze spočívá v samotné technické implementaci řešení. Vývojáři jsou během této fáze neustále zapojeni do kódování, testování a integrace nových funkcí. V rámci GoodAccess se vývojový tým neustále vzájemně kontroluje a podporuje, snaha je na zajištění správného dodržení postupů, čistého kódu a následování odpovídající architektury. Product owner má na starosti udržování vizí a požadavků koncových zákazníků a ujišťuje se, že výsledky práce týmu odpovídají obchodním cílům. Testeři hrají rovněž klíčovou roli v implementační fázi. Jsou zodpovědní za přípravu a provedení testů, jejichž úkolem je ověřit funkčnost a stabilitu softwaru před jeho nasazením do provozu. Tato aktivita zahrnuje různé formy testování, od unit testů po komplexní manuální a automatizované testy. V rámci výchozího stavu společnosti před zavedením inovovaných postupů se setkáme s nasazením testera v poslední fázi vývoje, který v tomto momentě spíše připomíná waterfall. Tester jako takový se neúčastní plánovací fáze a je zařazen až po výsledné implementaci nově vyvíjených součástí softwaru.

Celá fáze zároveň následuje všechny podstatné Agile vývojové ceremonie. Nejvýznamnější jsou každodenní stand-up meetingy, kdy se sejde celý vývojový tým a každý den si společně projdou splněné úkoly a úkoly na kterých budou dále pracovat. Proberou se možné překážky a blokující okolnosti, které e následně pokoušejí co nejdříve odbavit. Pro standupy jsou využívány aplikace GitLab (vývojový ticketing systém a správa projektu) a následně Google Meets pro online meetingy, případně se ještě využívá project management systém Asana. V průběhu implementační fáze jsou tedy propojeny technické dovednosti a projekt management metody s cílem vytvořit produkt, který nejen splňuje technické požadavky, ale který je také plně v souladu s obchodními cíli a potřebami uživatelů.
Vzhledem k tomu, že se stále dokola bavíme o Agile SCRUM tak je nutno zmínit, že sprint jako takový je vnímán jako jednotlivá iterace produktu. V některých pramenech a v metodologii SCRUM obecně se dočteme, že jeden sprint obsahuje jednotlivé úkoly z product backlogu, tedy jenom součásti celkového projektu. Ve společnosti GoodAccess se projektem nazývají jednotlivé ucelené sety úkolů pro určitou část projektu, respektive celého produktu. Tedy jinými slovy celý produkt je „hlavní“ projekt a v rámci sprintů se pracuje na projektech, které jsou jednotlivé ucelené součásti celého produktu, „hlavního“ projektu. V rámci sprintu se iteruje následně dle potřeb vývoje, kromě standupů se svolávají ad-hoc schůzky, pokud je třeba upřesnit nějaké požadavky projektu, technické výzvy nebo například doladit součásti technické architektury a podobně.

3.2.3 Vyhodnocovací fáze

Po ukončení sprintu se následně tým dostává do vyhodnocovací fáze. Vyhodnocovací fáze ve společnosti GoodAccess je závěrečným a klíčovým segmentem každého sprintu. Tato fáze zahrnuje dva hlavní komponenty: sprint review a sprint retrospective. Její průběh je zásadní pro udržení agilního procesu a neustálého zlepšování týmové práce a samotného produktu. Je to považováno za jakousi závěrečnou ceremonii v rámci vývojového týmu.

3.2.3.1 Sprint Review

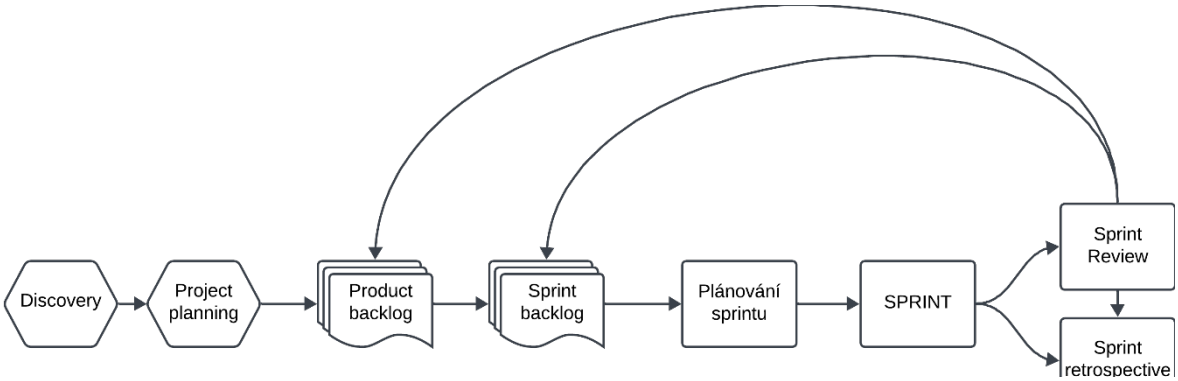
Sprint review je setkání, které se koná na konci každého sprintu. Hlavním cílem je prezentovat stakeholderům dosažené výsledky a shromáždit jejich zpětnou vazbu. Toto setkání není jen formalitou, ale představuje prostor pro diskusi o tom, co bylo během sprintu vyvinuto. Tým, včetně vývojářů, scrum mastera, a product ownera, představí, které funkce byly implementovány a zda byly nasazeny na produkční prostředí, případně zda jsou finálně připraveny na nasazení. V případě, že některé cíle nebyly dokončeny, jsou identifikovány důvody a diskutovány s ostatními členy týmu i dalšími účastníky. Tato transparentní komunikace pomáhá všem zainteresovaným stranám pochopit, kde tým stojí vůči cílům produktu a co je potřeba zlepšit pro další sprint. Během review jsou také upraveny sprint backlog a product backlog. Výsledky tohoto setkání mají přímý vliv na prioritizaci a aktualizaci těchto backlogů, což umožňuje týmu přizpůsobit se měnícím se požadavkům a zajistit, že následující sprint bude zaměřen na nejdůležitější úkoly.

3.2.3.2 Sprint Retrospective

Sprint retrospective se obvykle koná bezprostředně po sprint review. Je to čas pro sebereflexi a hledání možností, jak zlepšit pracovní procesy. Tým analyzuje, co fungovalo dobře a co by se dalo udělat lépe, jak na úrovni technických schopností (hard skills), tak v oblasti týmové práce, komunikace a dalších měkkých dovedností (soft skills). Retrospektiva je otevřená diskuse, kde každý má možnost vyjádřit se k jakémukoli aspektu práce – od procesů přes nástroje až po osobní interakce. Cílem je vytvořit bezpečné a podpůrné prostředí, kde je možné otevřeně mluvit o úspěších i neúspěších, bez obav z odsouzení. Tým se zaměřuje na konstruktivní zpětnou vazbu a pracuje společně na nalezení řešení pro identifikované problémy. Je zásadní, aby se retrospektiva neomezovala pouze na negativní aspekty, ale aby uznávala a oslavovala úspěchy týmu. Toto pozitivní uznání motivuje tým a přispívá k budování silné týmové dynamiky. Výstupy z retrospektivy jsou zdokumentovány a stávají se akčními body pro zlepšení v nadcházejících sprintech. Tato opatření mohou zahrnovat technické úpravy, změny v komunikaci nebo nové způsoby řešení problémů. Scrum master hraje klíčovou roli v zajištění, že tyto akční body jsou integrální součástí plánování dalšího sprintu.

3.2.3.3 Kontinuální Zlepšování

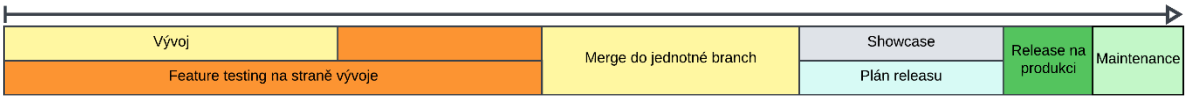
Hlavním cílem vyhodnocovací fáze je poskytnout týmu příležitost k neustálému zlepšování. Jak sprint review, tak retrospektiva jsou základními kameny pro tento proces. Tým se učí z minulých zkušeností a neustále se přizpůsobuje. Důraz na retrospektivu pomáhá nejen v identifikaci oblastí pro zlepšení, ale také v rozvoji týmu jako celku. Společným cílem je nejen vytvářet kvalitní software, ale také posilovat týmové vazby a profesionální růst každého jednotlivce. Výsledkem je nejen vývojový tým, který je efektivní a spokojený, ale i produkt, který je v souladu s očekáváními a potřebami koncových zákazníků.



OBRÁZEK 6 - PROCES VÝVOJE, ZPRACOVÁNÍ AUTOR 2024

3.3 Proces vydávání

Nedílnou součástí vývojového procesu je nasazování na produkci, tzv. proces vydávání. Společnost GoodAccess v rámci agilového vývoje využívá přístupu CI/CD a tedy nasazování probíhá pravidelně a často. Pravidelné procesy vydání jsou vykonávány s kadencí jednou za týden nebo jednou za 14 dní. Procesy probíhají například dvakrát během jednoho sprintu. Důvodem je, že během jednoho sprintu se pracuje klidně na vícero projektech najednou, a proto je možno vydávat vícekrát v rámci více projektů. Pravidelná vydávání se vyznačují také tím, že jejich součástí nejsou jenom nově vyvíjené součásti nebo inovace, ale jsou takto pravidelně vydávány právě opravy nalezených chyb či lehké úpravy. Zde je nutno zmínit, že občas se nasazuje také mimo pravidelný vydávací proces. Tak se stává v případě oprav kritických chyb, kde se využívá tzv. „hotfixu". Tento proces obchází schvalovací a testovací procesy a takové nasazení je následně testováno zpětně. Jde zpravidla například o chyby v kritické infrastruktuře aplikací nebo nejdůležitější funkce jako je například přihlášení do aplikace nebo připojení ke vzdálené bráně VPN. U těchto funkcí se tým vývojářů snaží zabezpečit jejich stoprocentní dostupnost, a proto se v těchto případech přistupuje k okamžitým opravám.



OBRÁZEK 7 - VÝCHOZÍ PROCES VYDÁVÁNÍ, ZPRACOVÁNÍ AUTOR 2024

Z obrázku (7) je patrné, že proces vydávání nastupuje až v pozdějších fázích celého vývoje, respektive počíná plánem, odsouhlasením, a nakonec samotným nasazením. Ve výchozím stavu se setkáváme s testováním na straně vývojového týmu, kdy samostatně každý vývojář sám testuje svou vyvíjenou feature. Ve výchozím stavu se vydávání plánuje na základě vzájemné domluvy týmu, není zde tedy přítomna osoba zodpovědná za vydávání na produkční prostředí. Součástí procesu vydávání je také navázaná dokumentace, jež obsahuje changelog provedených změn nebo nově přidaných součástí produktu. Tento changelog se v minulosti zveřejňoval, momentálně však funguje jako interní záležitost a zákazníci jsou informování zkrácenou verzí.

Proces vydávání také obsahuje vybrané metriky. Vývojový tým měří rychlost nasazení a celkovou dobu odstávky služeb, ta je však vázána na to, jak velké změny jsou nasazovány a jaký je celkový vliv na systém. Menší změny vedou ke kratším prostupům, naopak velké zásahy mohou silně ovlivňovat dostupnost služeb. To se odvíjí od toho, zda je ovlivněna například infrastruktura nebo pouze UX/ UI elementy produktu. Komplexní kontrola kvality může kladně ovlivnit celý proces nasazení na produkční prostředí, jelikož se dá předcházet chybám, které ovlivní dostupnost služeb. Další metrikou je také chybovost po nasazení, tedy to, kolik nových chyb se nestihlo odhalit před nasazením na produkci, a tedy probublaly směrem nahoru na produkční prostředí, kde se s nimi setkali koncoví zákazníci.

3.4 Používané nástroje

Pro efektivní spolupráci a koordinaci svého týmu se společnost spoléhá na soubor nástrojů, které jsou základem jejich každodenní práce, a které odráží její moderní přístup k řízení projektů a kvality softwarového vývoje.

3.4.1 Komunikace a spolupráce

3.4.1.1 Slack

Komunikace týmu je zásadně ovlivněna využíváním nástroje Slack, což je platforma, která umožňuje okamžitou výměnu informací, efektivní komunikaci v reálném čase, a integraci s množstvím dalších aplikací používaných v rámci společnosti. Slack přináší možnost strukturovat komunikaci do jednotlivých kanálů, což umožňuje týmům se soustředit na specifické projekty nebo témata bez zbytečného rušení. Zároveň je Slack využíván jak s interními, tak externími integracemi jako jsou například Google Calendar nebo Asana. Týmy napříč celou společností mají rychlou možnost komunikace a možnost rychlé reakce v rámci produktu, jako je například feedback od zákazníků a podobně.

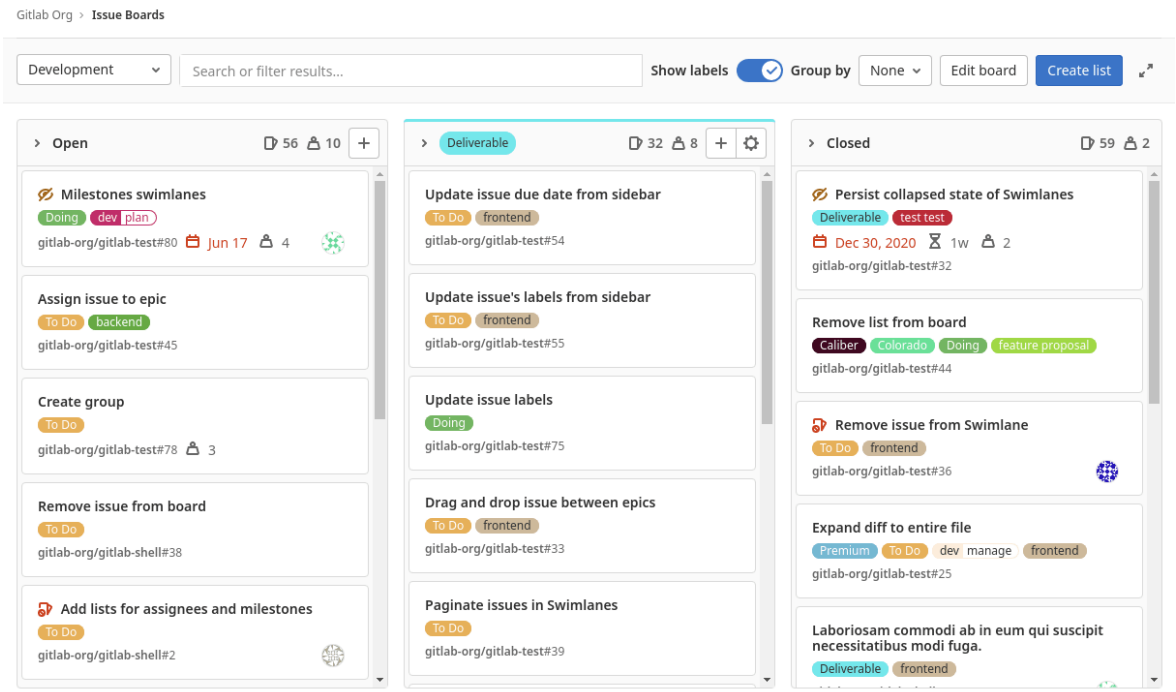
3.4.1.2 Google workspace – calendar, meets

Dále, společnost využívá Google Workspace, jehož součástí je Google Drive pro sdílení a společnou práci na dokumentech, tabulkách a prezentacích, které jsou nezbytné pro průběh projektů. Google Calendar je používán pro koordinaci schůzek a rozvrhů, zatímco Google Meets, slouží jako nástroj pro online schůzky, které jsou klíčové v dnešním světe vzdálená spolupráce. Google Workspace se přenáší také do dalších součástí řízení projektu. Součástí Google Workspace je také aplikace Gmail, emailový klient, který společnost využívá a celkově zde má přehled nad emailovými účty pracovníků ve společnosti, jelikož se pomocí nich přihlašuje do rozličných aplikací třetích stran včetně aplikací vyvíjených přímo společností GoodAccess, kdy se využívá SAML protokolu pro bezpečné přihlášení.

3.4.2 Správa projektu

3.4.2.1 GitLab

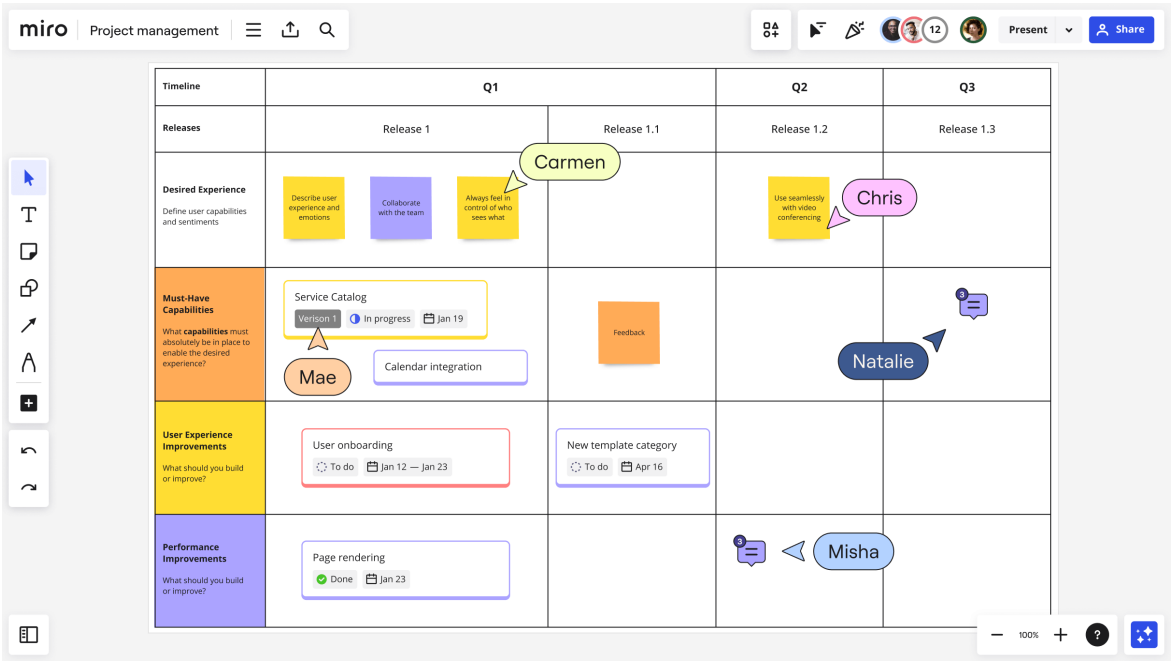
Pro projektový management a správu verzí se GoodAccess spoléhá na GitLab, platformu, která integruje správu kódu s funkcemi CI/CD. To umožňuje týmu kontinuálně pracovat na vývoji a testování nových funkcí bez narušení stabilní verze produktu. GitLab, s jeho uživatelsky přívětivým rozhraním a robustními funkcemi, se stal kritickým nástrojem nejen pro správu kódu, ale i pro celý životní cyklus softwarového vývoje. V rámci GitLabu je nejvíce využívána jako možnost Kanbanových tabulek. Probíhá zde odbavování jednotlivých ticketů souvisejících s vývojem. Jednotlivé tickety je možno přiřazovat vývojářům, podávat zpětnou vazbu nebo je označovat „labely“, které slouží pro jednoduché rozpoznání, čeho se daná kartička týká.



OBRÁZEK 8- GITLAB KANBAN BOARD. DOSTUPNÉ Z: HTTPS://DOCS.GITLAB.COM/EE/USER/PROJECT/ISSUE__BOARD.HTML

3.4.2.2 Miro

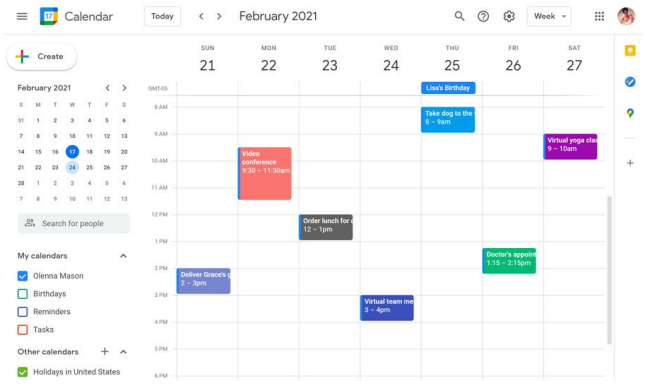
Dalším využívaným nástrojem je Miro, což je projekt managementový systém, který podporuje mnoho standardů pro vývojové diagramy a podobně, jako je například BMPN nebo UML. Zároveň nabízí mnoho managementových prvků. Jeho nespornou výhodou je možnost kolaborace, kdy jednu nástěnku může spravovat více uživatelů najednou.



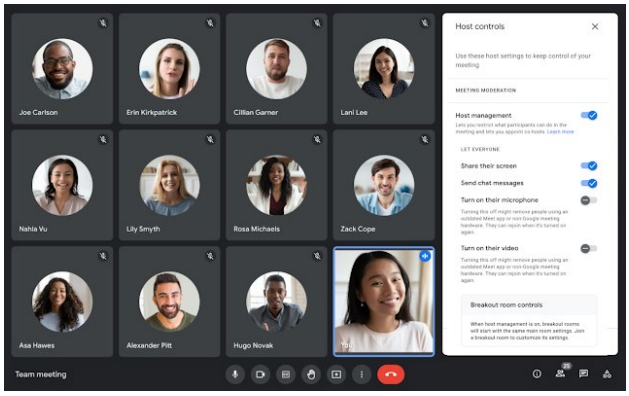
OBRÁZEK 9 - MIRO WORKSPACE, DOSTUPNÉ Z: HTTPS://MIRO.COM/

3.4.2.3 Google workspace

Jak bylo zmíněno v kapitole (4.4.1), pro správu projektu se využívá další aplikace z Google Workspace, tím je v tomto případě Google Drive společně s aplikacemi Google Docs a Google sheets. Drive je aplikace simulující harddisk v cloudu, jde tedy o on-linové řešení úložiště dat a dokumentů. Docs je aplikace pro zpracování textu podobně jako je například MS Office Word a podobně. Sheets je naproti tomu editor tabulek opět po vzoru MS Office Excel. Všechny tyto aplikace společně synergicky spolupracují a dokážou se vzájemně pohodlně integrovat. Jelikož společnost GoodAccess využívá kompletního spektra aplikací Google Workspace dává smysl je používat všechny dohromady.



OBRÁZEK 11 - GOOGLE CALENDAR, DOSTUPNÉ Z: HTTPS://EDU.GCFGLOBAL.ORG/EN/GOOGLE-TIPS/GETTING-STARTED-WITH-GOOGLE-CALENDAR/1/



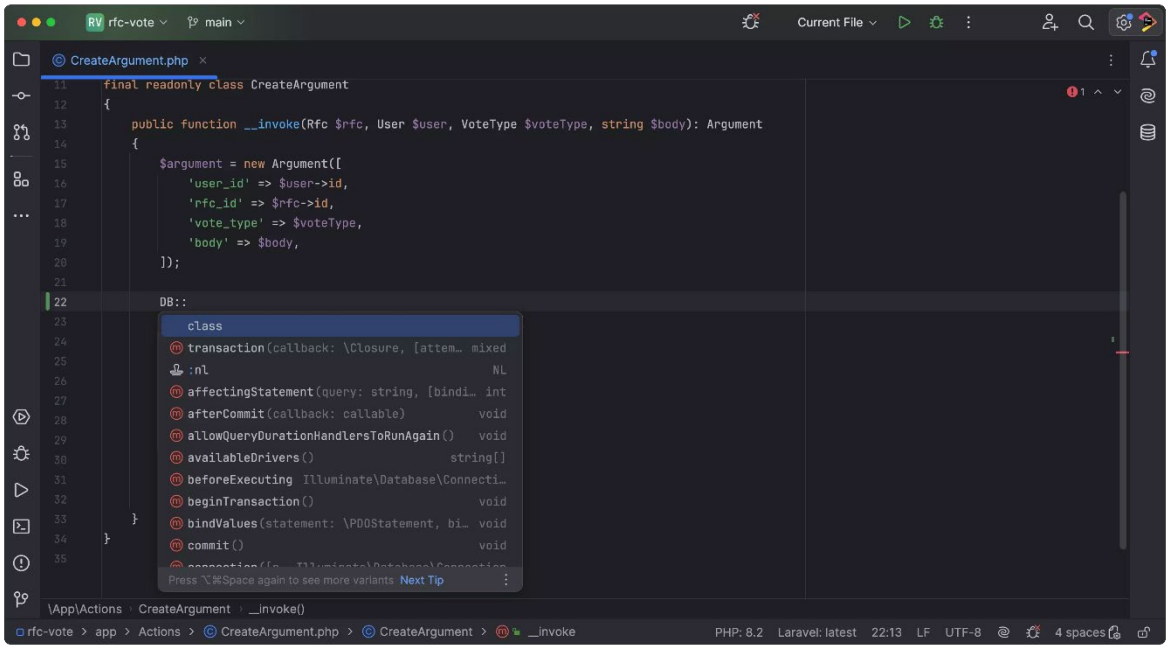
OBRÁZEK 10 - GOOGLE MEET, DOSTUPNÉ Z: HTTPS://MEET.GOOGLE.COM

3.4.2.4 Asana

Posledním nástrojem pro spravování projektu je Asana. Ta je využívána především produktovým týmem, businessem nebo marketingovým týmem. V rámci této diplomové práci se věnujeme vývojovému a QA procesu, avšak Asana zasahuje i do těchto aktivit a to především v rámci evidence a dokumentace product a sprint backlogu. Zároveň byla Asana využita pro nově zavedené QA procesy a v databázi aplikace byly vedeny dokumenty a tickety související s QA aktivitami, plněním cílů apod.

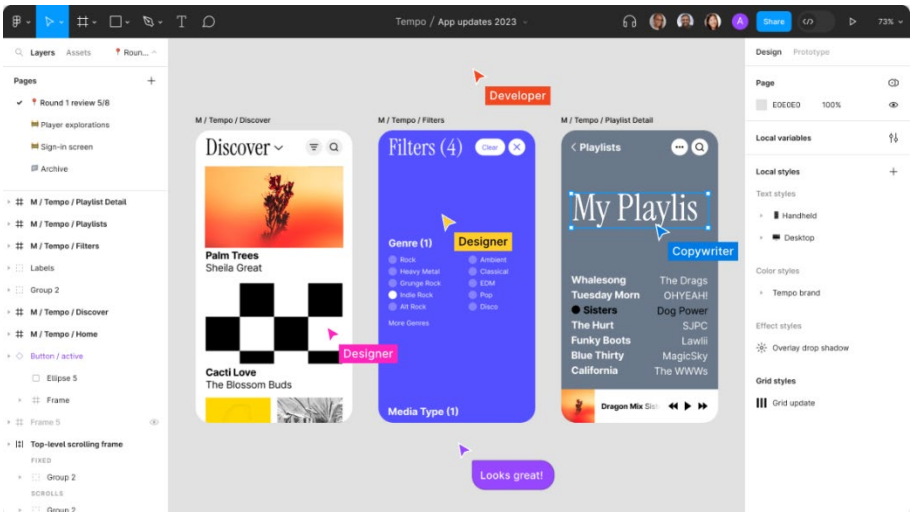
3.4.3 Vývoj a testování

Vývojáři v GoodAccess mají k dispozici vývojové prostředky jako PHPStorm od JetBrains, což je inteligentní IDE (Integrated Development Environment) specializované pro PHP, které nabízí rozsáhlé možnosti pro návrh, psaní a ladění kódu. Dále tým využívá GitHub Copilot, AI nástroj, který slouží, jako inteligentní asistent programátora, pomáhající s doporučeními kódu a zvýšením efektivity vývoje. Výhodou PHPStorm je jeho specializované určení pro jazyk PHP a jeho frameworky, čímž se uceluje možnost efektivního vývoje.



OBRÁZEK 12 - PHP STORM IDE NÁHLED PRACOVNÍ PLOCHY - DOSTUPNÉ Z: HTTPS://WWW.JETBRAINS.COM/PHPSTORM/FEATURES/

Dalším hojně užívaným nástrojem je Figma, ta se ve společnosti GoodAccess využívá pro designování UX/ UI první aplikací, které jsou vyvíjeny. Figma je nástrojem, jehož síla spočívá stejně jako u Mira ve spolupráci více uživatelů nad jednou nástěnkou kdy mohou vzájemně kolaborovat na vytvářených designech. Figma podporuje celý designový proces, avšak vývojový tým ji využívá především jako vodítko při vývoji a samozřejmě také jako pomoc, jelikož dokáže vzájemně propojovat design a kód.



OBRÁZEK 13 - FIGMA HOMEPAGE, DOSTUPNÉ Z: HTTPS://WWW.FIGMA.COM

3.5 Proces kontroly kvality

Kontrola kvality ve výchozím stavu do určité míry kopírovala sprintový přístup k vývoji, avšak testing nebyl zařazen do všech fází a v majoritě případů se s testováním začínalo až v rámci zakončení implementační fáze a spoléhalo se výhradně na manuální testování. Problém tkvěl také v nedostatečné personální kapacitě v rámci kontroly kvality, kdy byl přítomen pouze jeden tester, který se věnoval především mobilním a desktopovým aplikacím. V rámci webové aplikace se přistupovalo k testování samotným týmem vývojářů a product ownerem, případně dalšími kolegy. Tento přístup nesl mnohá úskalí, jelikož se vystavujeme možnosti „slepoty tvůrce" kdy je pro vývojářský tým jednoduché přehlížet chyby, jelikož testy nejsou tak objektivní jako v případě vykonávání kontroly kvality samostatným pracovníkem, který má pohled na software z jiného úhlu. Testy byly vykonávání především manuálně, následuje také absence unit testů a jejich integrace do procesu CI/CD dále podkopává schopnost rychlé a efektivní reakce na změny v kódu, které jsou v agilním vývojovém prostředí nezbytné. Unit testy představují důležitý nástroj pro ověřování jednotlivých částí aplikace v izolaci, což zajišťuje stabilitu a funkčnost kódu. Tedy panoval přístup, který byl v mnoha ohledech neadekvátní pro agilní vývojové procesy a nestál v souladu s nejlepšími praktikami. Bylo zřejmé, že pro zajištění vyšší kvality softwarového produktu a efektivity testovacích procesů je nezbytné provést revizi a zlepšení v oblastech automatizace testů, dokumentace a integrace testů do CI/CD procesů.

Samostatná dokumentace jak vývojová a projektová dokumentace existuje v dostatečně kvalitní míře, avšak testovací dokumentace nebyla vytvořena v potřebném detailu, opět vzhledem k nedostatečné personální kapacitě ve společnosti. Absence testovací dokumentace je velice častým problém pro menší vývojářské týmy/ softwarové společnosti, jelikož je náročná jak na personální, tak na finanční zdroje a její potřeba se začne zvětšovat s růstem společnosti a celkově také s rostoucí komplexností produktu.

3.6 Požadavky na zlepšení

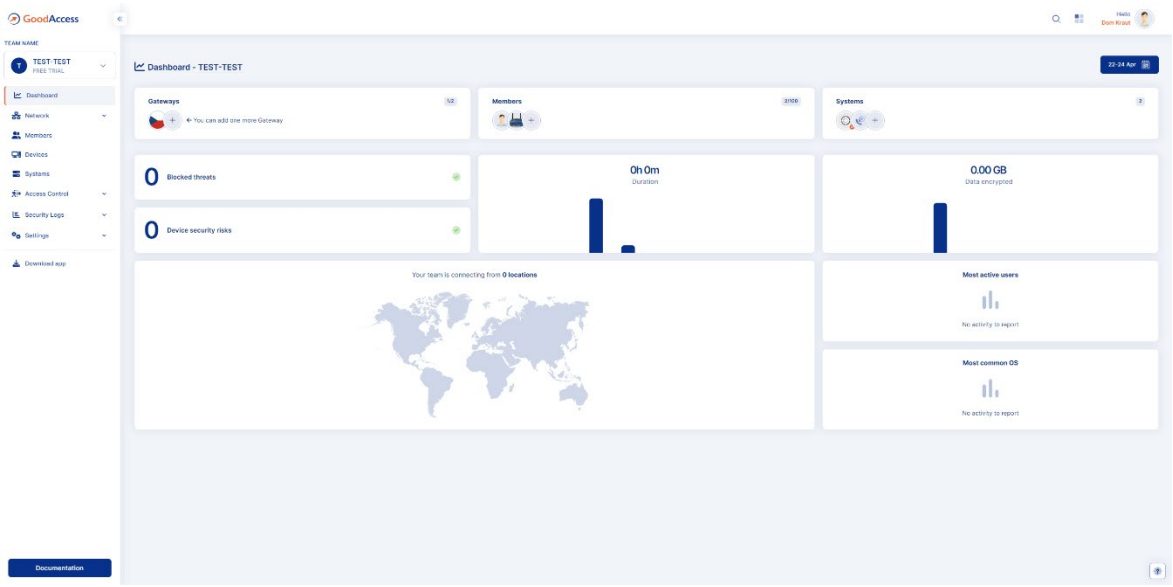
Společnost GoodAccess přistupuje se záměrem zlepšení celkového procesu kontroly kvality. Ten v důsledky pozmění a vylepší celkový vývoj produktu ve společnosti. Kontrola kvality byla doposud spíše upozaděna, jak z popisu výchozího stavu vyplývá. Společnost dosáhla bodu, kdy se mohou zvýšit kapacity pro vytvoření spolehlivého kompletního „fully-featured" procesu kontroly kvality vyvíjeného softwarového produktu. Ve společnosti GoodAccess byla rozpoznána potřeba reformovat a vylepšit stávající postupy QA, což je odrazem jejího závazku k inovaci. Tradičně byly aktivity QA ve společnosti vnímány jako sekundární, avšak analýza výchozího stavu naznačuje, že pro udržení konkurenční pozice a kvality produktu je potřeba dát kontrolu kvality do centra pozornosti. GoodAccess dosáhla fáze ve svém vývoji, kdy je žádoucí nejen rozšíření kapacit, ale i vytvoření robustního a plně integrovaného procesu kontroly kvality. Tato iniciativa je v souladu s moderními trendy softwarového průmyslu.

S přihlédnutím k těmto cílům se společnost rozhodla k radikální transformaci svých QA procesů a ke spolupráci s novým testerem, autorem této práce, jehož znalosti a zkušenosti v oblasti Quality Assurance jsou nezbytné pro dosažení těchto ambiciózních cílů. K dosažení těchto cílů společnost plánuje zavedení automatizovaných testů a rozvoj pokročilých manuálních testů, které budou spolupracovat s cílem zajistit nejvyšší možnou úroveň kvality softwaru. Důraz bude kladen na vypracování komplexní testovací dokumentace, která poslouží jako klíčový nástroj pro sledování, hodnocení a

dokumentaci testovacích aktivit. Následujícím krokem je vypracování testovacího plánu, který bude detailně mapovat všechny fáze vývoje softwaru, od počátečních fází specifikace požadavků, přes design a implementaci, až po uvedení produktu na trh. Testovací plán bude také obsahovat metodiku pro integraci automatizovaných testů, což zahrnuje výběr nástrojů, definování testovacích scénářů a stanovení metrik pro hodnocení úspěšnosti testů. Důležitým aspektem plánu je určení, které komponenty webové aplikace budou podrobeny automatizovanému testování a jak bude tento výběr ovlivněn podnikovými prioritami a riziky. Toto vyústí v doporučení pro budoucí rozvoj, které pomohou společnosti udržet své QA postupy v souladu s nejlepšími praxemi a inovačními standardy odvětví. Na základě provedených analýz a výsledků implementace nových QA procesů bude tato práce prezentovat komplexní zhodnocení a navrhne další kroky, které by společnost GoodAccess měla podniknout, aby udržela a nadále rozvíjela úroveň kvality. Výhodou je již existující Agilový přístup společnost, a není nutno vytvářet proces vývoje kompletně od počátku, jelikož tým již využívá moderní postupy tohoto rámce.

3.7 Povaha testované aplikace

Aplikace, pro kterou jsou požadované změny v kontrole kvality společností GoodAccess vykonávány se nazývá GoodAccess Central Dashboard. Jedná se o manažerský panel, který pracuje se správou uživatelů a dostupných funkcí v desktopových aplikacích společnosti GoodAccess, které slouží k zabezpečenému přístupu interních systému společností. Tento panel nabízí správcům zabezpečení ve společnostech spravovat přístupy uživatelů k daným funkcím a zároveň vidí důležité statistiky. Jako je počet přenesených dat uživatelů, jejich lokace a na jaké systémy se přihlašují. Zároveň je zde možnost sledovat zablokované hrozby, které odchytila služba GoodAccess. (GoodAccess, 2024)



OBRÁZEK 14 - GOODACCES CENTRAL DASHBOARD, GOODACCESS 2024

4 Navrhované řešení

Navrhované řešení je dle specifikací společnosti. V rámci zapracování komplexního procesu kontroly kvality do vývoje je potřeba začít s celkovým procesním managementem a optimalizací procesů stávajících. Na základě popisu výchozího stavu společně s požadavky společnosti jsem přistoupil k návrhu řešení, který popisuji v této kapitole. Následně se přešlo k samotné implementaci, a nakonec k vyhodnocení. Celá implementace probíhala v několika fázích. Pro určení významnosti jednotlivých součástí nově vypracovaných procesů a metrik byla vytvořena jednoduchá tabulka, která zohledňuje určené cíle společností společně s jednotlivou vahou určující prioritu pro zadavatele (společnost). Tyto priority byly vypracovány spolu s managementem společnosti. Byly určeny procentuální výchozí hodnoty a následně požadované cíle. V tabulce je procentuálně určena výchozí hodnota pokrytí a následně cílová kýžená hodnota pokrytí. Vzhledem k tomu, že jde o budování celkově nového přístupy jsou rozdíly hodnot relativně vysoké.

Požadavek/ Cíl	Priorita	Výchozí hodnota	Cílová hodnota
Robustní QA proces	Vysoká	25 %	80 %
Dokumentace test. scénářů	Střední	10 %	95 %
Automatizované testy	Vysoká	0 %	75 %
Manuální testy	Vysoká	65 %	100 %
Dokumentace QA procesů	Nízká	0 %	50 %
Dokumentace testovacího plánu	Střední	0 %	80 %
Integrace QA do vývojového procesu	Vysoká	30 %	100 %

TABULKA 1 - CÍLE PROJEKTU, AUTOR 2024

Na základě tabulky je možno vyvodit dopady, které mohou změny mít na celkové fungování společnost. Od schopnosti spolupráce mezi jednotlivými týmy až po hodnotící metriky v rámci sprintů. Cíle byly určeny s ohledem na jejich splnitelnost a následně si je společnost spolu s testerem rozřadí do jednotlivých časových rámců a opatří se jakožto splnitelné KPI cíle, které budou pravidelně vyhodnocovány. Jakožto časový rámec se navrhuje dosažení cílů v rámci jednoho roku, kdy se budou jednotlivé úkoly plnit ve čtvrtletních celcích označených Q1, Q2, Q3, Q4. Pro jednotlivá čtvrtletí budou zařazeny kratší úkoly, které jsou vyhodnoceny jako dílčí úkoly z celého projektu, který se takto postupně dosáhne. Z tabulky jsou také patrné jednotlivé oblasti, kterým se v rámci projektu autor následně plánoval věnovat. Jednotlivé sekce jsou následně rozebrány v kapitolách věnujících se samostatné implementaci. Konkrétně jde o kapitoly sekce 5.1.

4.1 Implementace řešení

Navrhované řešení bylo implementováno v časovém rozmezí, které bylo vyhodnoceno jako jeden rok od startu projektu. Celý projekt implementace by dodatečně rozdělen na dvě základní časová období, první polovina roku, která sloužila jako základní fáze, ta se soustředila především na zavedení, optimalizaci a standardizaci procesů. Zároveň se první polovina roku silně věnovala vytvoření testovací dokumentace, testovacího plánu a s tím navázané manuální testy. S přípravou dokumentace se váže příprava pro automatizaci vybraných testů. V druhé polovině roku se prováděla především

implementace automatizovaných testů a dodatečné úpravy nově zavedených procesů. Dále se v druhé polovině časového rámce pracovalo s optimalizací procesu vydávání (proces vydávání, kapitola 4.3).

Pro řízení projektu byla zvolena platforma Asana, která umožnila detailní rozpis cílů do jednotlivých kvartálů. Každý kvartál (Q1, Q2, Q3, Q4) měl specifické milníky a cíle, které byly pravidelně monitorovány a hodnoceny. Tyto cíle zahrnovaly:

První kvartál (Q1):

- Vypracování a schválení QA procesního dokumentu.
- Vývoj a zavedení počáteční sady testových scénářů pro klíčové funkce produktu.
- Kompletní seznámení testera s produktem a interních nástrojů.
- Nastavení spolupráce s vývojovým týmem.
- Účast testera na všech ceremoniích.

Druhý kvartál (Q2):

- Zavedení pravidelných manuálních testovacích cyklů pro kontrolu nově implementovaných funkcí a součástí
- Implementace prvotních automatizovaných testů.
- První vyhodnocení efektivity nových procesů a úprava plánu na základě získaných dat.
- Vývoj a dokumentace testovacího plánu pro další kvartály.

Třetí kvartál (Q3):

- Další rozvoj a optimalizace automatizovaných testů s důrazem na regresní a náročnější testovací scénáře.
- Implementace pokročilých analytických nástrojů pro hodnocení kvality a reporting.
- Průběžné školení QA týmu a rozvoj jejich dovedností v oblastech jako testování, programování a používání nových nástrojů.

Čtvrtý kvartál (Q4):

- Závěrečné hodnocení dosažených výsledků ve srovnání s původními cíli.
- Feedback loop s vývojovým týmem
- Vývoj plánu pro následující rok na základě lekcí získaných během roku.
- Zavedení procesů pro průběžnou integraci a doručování (CI/CD).

Během celého procesu implementace byl kladen velký důraz na transparentnost, což umožňovalo všem členům týmu mít přehled o postupu a výsledcích. Projektový manažer pravidelně sdílel aktualizace s celým týmem a managementem společnosti, což zajišťovalo, že všechny zainteresované strany jsou informovány a mohou poskytovat zpětnou vazbu.

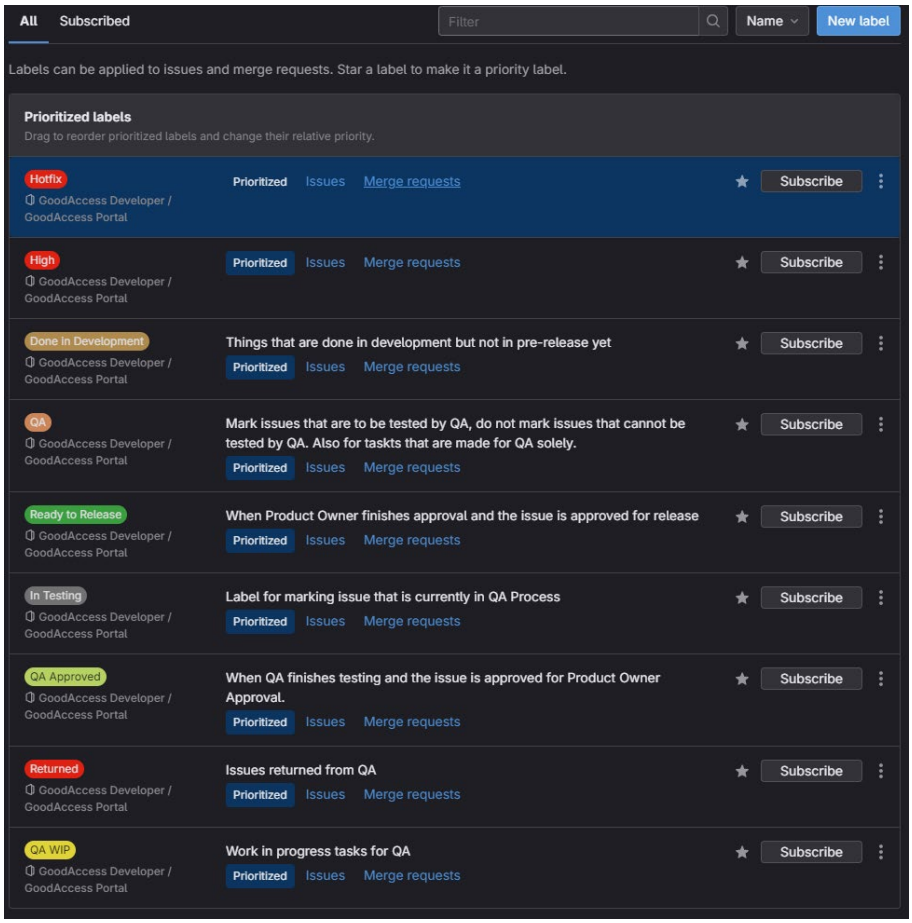
Konec prvního roku byl charakterizován vyhodnocením dosažených výsledků a sestavením plánu pro následující rok. To zahrnovalo revizi úspěšně implementovaných postupů a identifikaci oblastí, které potřebují další zlepšení. Strategie pro další rok byla založena na těchto hodnoceních a získaných poznatcích s jasným zaměřením na neustálé zlepšování a inovaci. Podrobnější vyhodnocení a doporučení pro další vývoj jsou uvedena v nadcházejících kapitolách níže.

4.1.1 Úprava procesu vývoje

Integrace komplexních procesů kontroly kvality přinesla vývojovému procesu řadu zásadních změn, které se projevily na několika úrovních. Úpravy byly promyšlené s cílem zvýšit efektivitu, spolupráci mezi týmy a celkovou kvalitu výsledného produktu.
Začlenění testera do všech setkání a meetingů bylo prvním klíčovým krokem. Tento přístup umožnil testerovi zapojit se do projektu od samého začátku, což vedlo k lepšímu pochopení požadavků a celkového procesu vývoje, stejně tak jako se mohl seznámit lépe s týmem na personální úrovni. Tester tak měl možnost provádět analýzu testovatelnosti již v rané fázi vývoje a přispívat tak k návrhům, které jsou vhodné k testování a zároveň intuitivní pro uživatele. Díky tomuto přístupu bylo možné identifikovat potenciální problémy dříve a ušetřit tak čas i zdroje v pozdějších fázích vývoje.

Dalším významným krokem bylo posílení spojení mezi vývojovým týmem a product ownerem. Zlepšení komunikace bylo dosaženo zahrnutím testera do pravidelných setkání, kde byly diskutovány a řešeny otázky UX/UI, což zajišťovalo, že všechny požadavky a očekávání jsou jasné a vzájemně porozuměny. Všechny strany si vzájemně předají zpětnou vazbu a tím je docíleno větší efektivity, která doposud nebyla zahrnuta do procesu.

V prostředí GitLabu došlo k přepracování workflow vývoje pomocí upravených labelů a boardů na kanban nástěnce. Nově zavedené značky odpovídaly specifickým krokům QA procesu a jejich účelem bylo zajistit, aby byly úkoly a tikety správně kategorizovány a přiřazeny. Byla stanovena jasná pravidla pro označování tiketů, což usnadnilo nejen sledování stavu úkolů, ale také určení priorit a zodpovědnosti. Vývojáři byli instruováni, aby tikety přiřazovali testerovi k dalšímu testování, což zajistilo, že každá nová feature byla před nasazením do produkčního prostředí řádně a systematicky otestována. Ve výchozím procesu byl již kanban systém využíván v rámci Gitlabu a vývoje, byly pouze přidány nové nástěnky a upraveny samostatné labely, kterým byl přidán vhodný popisek. Začaly se používat specifické QA štítky, uvedeny na obrázku a tabulce níže.



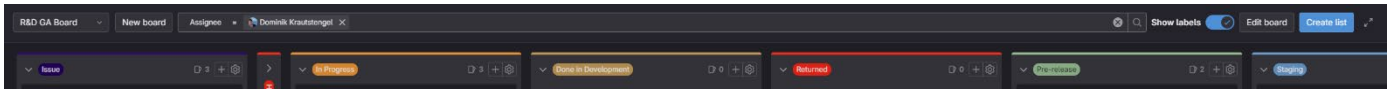
OBRÁZEK 15 - ŠTÍTKY V PROSTŘEDÍ GITLAB (PRIORITIZOVANÉ) - ZDROJ: GOODACCESS

Štítek v GitLabu	Popis
QA	Obecný štítek pro označení ticketů, které souvisejí s testerem
In Testing	Označení ticketu, který je právě v testovacím procesu
QA Approved	Ticket odsouhlasený testerem jako splněný a otestovaný
QA WIP	Označení ticketu, který je právě vypracováván v rámci QA
Blocked	Označení ticketu, který je blokován
QA Done	Označení hotových ticketů, vztahujících se k projektům čistě za QA oddělení
QA Automation	Označení QA projektů vztahujících se k automatizaci
Testing projects	Označení ticketů souvisejících s čistě QA projekty/ úkoly

TABULKA 2 - ŠTÍTKY V GITLABU, GOODACCESS 2024

Všechny tyto změny vyústily ve vývojový proces, který je nyní více centrován na kvalitu a který umožňuje týmu rychle reagovat na problémy a efektivně pracovat na jejich řešení. Nově implementovaný systém přidání značek a systematického přiřazování tiketů vedl k transparentnosti a snadnému sledování pokroku úkolů. To vše společně podporuje kulturu neustálého zlepšování a přizpůsobení se novým výzvám ve světě softwarového vývoje.

Kanbanová tabulka v nástroji GitLab následně funguje klasickým způsobem, kdy se postupně odbavují jednotlivé kartičky a vývojáři s testerem začali vzájemně kartičky přeřazovat jak mezi sebou, tak mezi jednotlivými nástěnkami a jejich štítky.



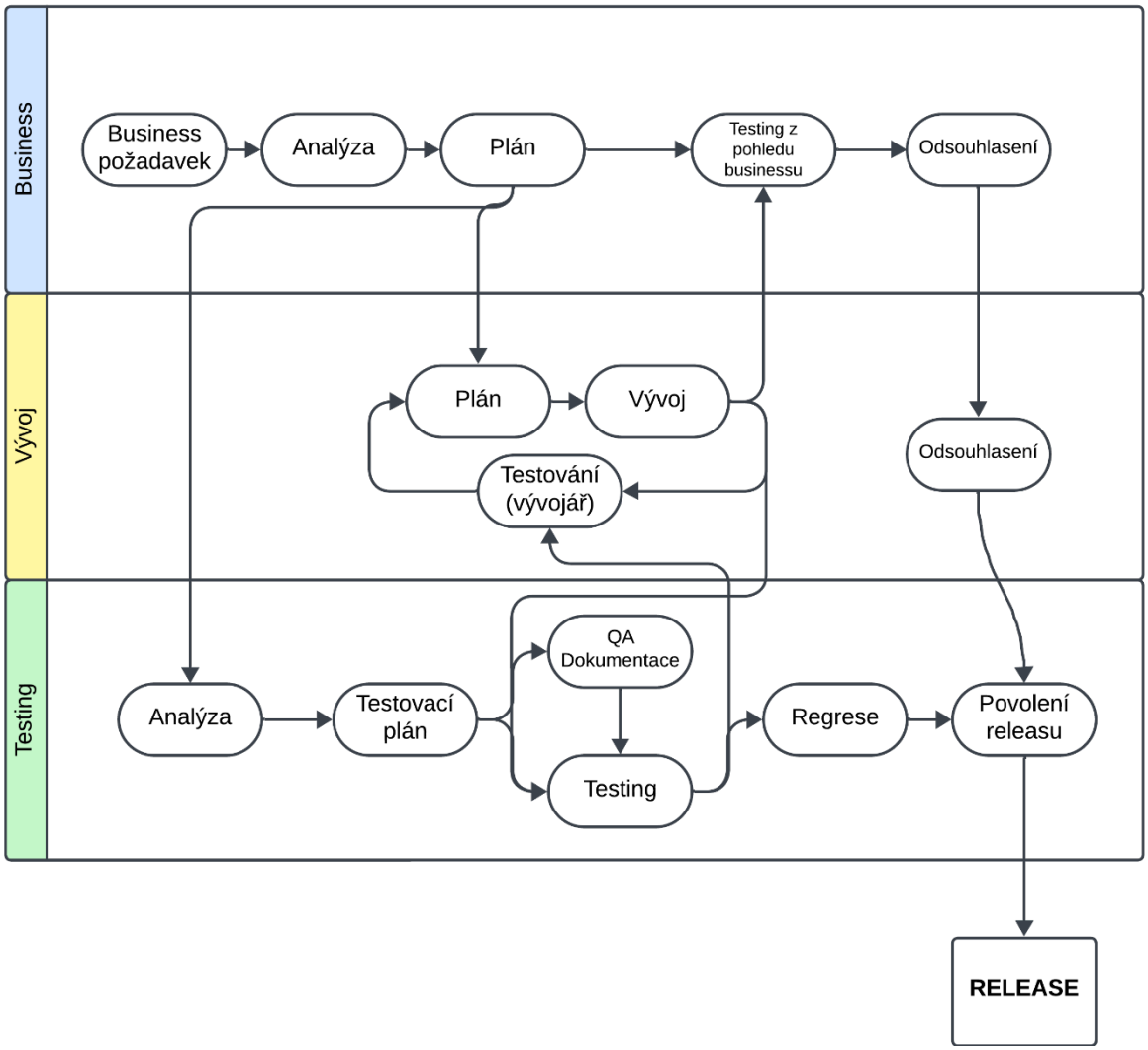
OBRÁZEK 16 - JEDNOTLIVÉ NÁSTĚNKY V GITLAB PROSTŘEDÍ, AUTOR 2024

Další provedenou změnou v rámci vývoje jako takového byla evoluce v komunikaci mezi jednotlivými vývojáři a testerem. Doposud se pracovalo na spíše vodopádovém stylu předávky jednotlivých kartiček s úkoly k testingu. Tento přístup se změnil za pomoci takzvané „shift-left" metodologie. Tato metoda spočívá v aktivnějším zapojení testera do vývoje, kdy vstupuje do procesu ještě před dokončením vývoje jednotlivých features. To znamená že tester začne s testováním téměř okamžitě a snaží se testovat zatím ještě nedopracovaný kus aplikace/ změn nebo jednotlivých features. V tento moment spolupráce funguje na bázi, kdy vývojář vyvíjí společnost s testerem a dynamicky a rychle si předávají vzájemné informace a status updaty. Docílí se tím tak mnohem efektivnějšího vývoje, kdy na sebe vzájemně nemusejí vyčkávat a tento čas je ušetřen. Tato úzká spolupráce umožňovala průběžnou výměnu informací a stavových aktualizací, což zvyšovalo rychlost iterací a zlepšovalo kvalitu vývoje. Statistiky ukazují, že aplikací "shift-left" se proces vývoje zrychlil přibližně o 30-40 % ve srovnání s předchozím stavem.

V důsledku tohoto přístupu bylo zaznamenáno značné snížení chyb objevených v pozdějších fázích vývoje, což vedlo k úspoře přibližně 25 % člověkohodin, které byly dříve vynaloženy na opravu chyb po hlavních vývojových milnících. Díky tomu bylo možné uvolnit zdroje a zaměřit se na vývoj dalších funkcí nebo vylepšení stávajících vlastností. Nehledě na snížení náporu na customer support a zpětné replikování objevených chyb.

Další významnou změnou, která doprovázela přechod k "shift-left" metodologii, bylo začlenění testera do procesu vydávání softwaru. Tester, který převzal roli release managera, se stal nejen posledním filtrem před uvedením produktu na trh, ale také hlavním rozhodčím pro posouzení, zda je produkt opravdu připraven pro produkční prostředí. Tento krok vedl k dalšímu zvýšení odpovědnosti a zlepšení kvality, protože release manager měl komplexní přehled o všech aspektech produktu. Tato role částečně také přenesla odpovědnost z vývojového týmu na testera a koncentrovala se tak do jednoho člověka, který dokáže posoudit připravenost produktu na vydání ke koncovým uživatelům. Zároveň se také začalo využívání procesu, kdy se některé změny zpřístupnili pouze některým uživatelům, kdy bylo domluveno testování nových funkcí z uživatelského hlediska na jejich straně.

Výsledkem těchto úprav je vývojový proces, který je v souladu s nejlepšími praxemi v oblasti QA, a který zvyšuje důvěru v kvalitu softwaru dodávaného společností GoodAccess s.r.o.



OBRÁZEK 17 - AKTUÁLNÍ VÝVOJOVÝ PROCES, ZJEDNODUŠENO, AUTOR 2024

4.1.2 Úprava procesu kontroly kvality

Proces kontroly kvality bylo primární zaměření v rámci zavedení komplexních QA postupů do společnosti GoodAccess s.r.o. Vedle procesu vývoje jsme se společně s týmem soustředili právě na zavedení těchto postupů. Klíčovým cílem bylo nejen vylepšení existujících procesů, ale také zavedení nových metod a nástrojů, které by umožnily efektivnější práci QA týmu a celkově zlepšily kvalitu vývoje. Tyto změny byly důležité pro udržení konkurenceschopnosti v dynamickém prostředí technologického trhu. V následujících odstavcích se podrobně zaměříme na jednotlivé aspekty těchto změn.

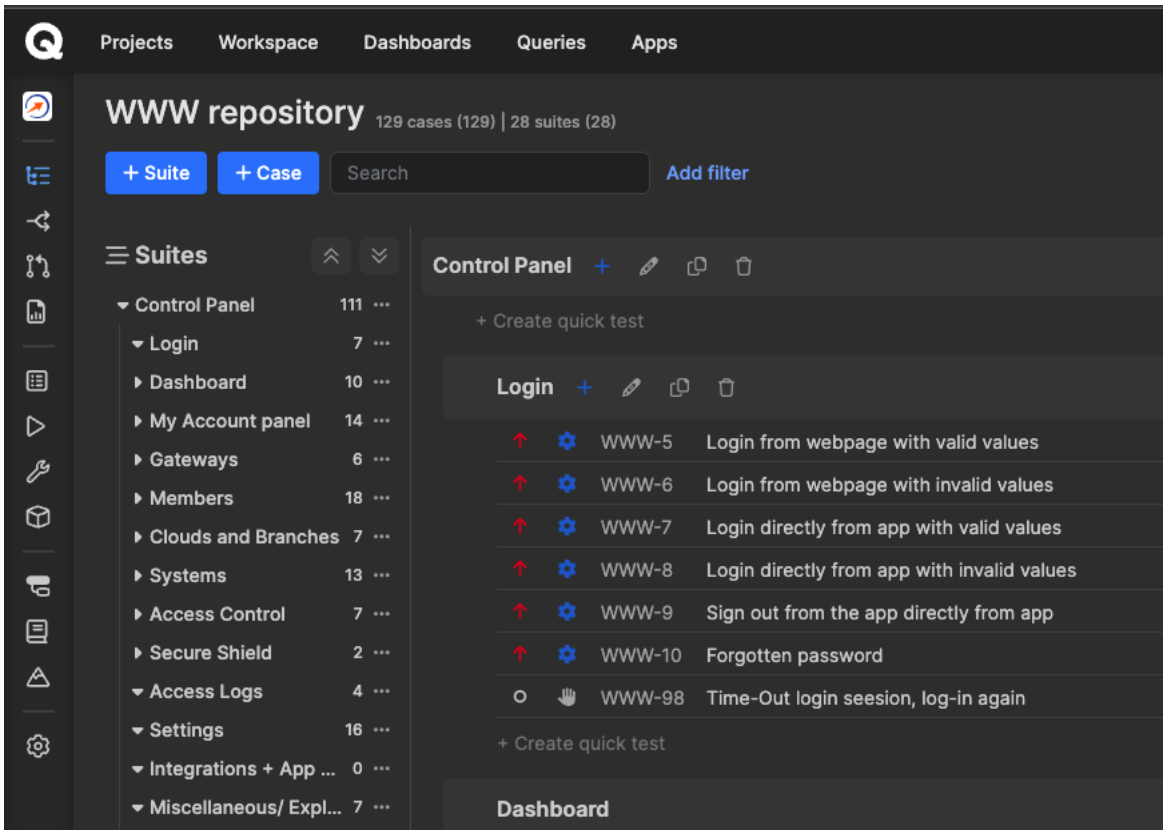
4.1.2.1 Testovací dokumentace

Jedním z prvních kroků bylo zavedení pečlivé dokumentace testovacích scénářů. Tento proces zahrnoval vytvoření detailních testů, které byly důkladně plánovány a kategorizovány podle priority a severity. Vznikla tak komplexní testovací databáze, který

umožnil QA týmu systematicky přistupovat k testování a zvyšovat efektivitu detekce a opravy chyb. Pro testovací databázi byl vybrán nástroj Qase.

Qase je test management systém (TMS), který nabízí moderní správu databáze testovacích scénářů. Zároveň nabízí nástroje pro organizaci, sledování a reportování testování a je navržen tak, aby umožnil co nejrychlejší reportování výsledků QA týmu. Mezi klíčové funkce Qase patří organizace testovacích případů do tzv. "Test Suites", což jsou sady testů, kde můžete definovat vlastnosti jako jsou závažnost a priorita testů. Dále je možné popsat před-podmínky a post-podmínky pro jednotlivé testovací případy a kroky pro reprodukci scénářů. Tyto funkce umožňují QA týmům systematický přístup k testování. Při samotném spuštění testů ("Test Execution") Qase nabízí "Smart Wizard", který vás provede tvorbou testovacích plánů a umožní vám kontrolovat každý testovaný případ najednou. Výsledky testů jsou přehledně zobrazeny, včetně úspěšnosti, chybových logů a doby trvání každého testu. Dalšími funkcemi Qase je schopnost analyzovat a sdílet testovací data. Díky přizpůsobitelným dashboardům je možné získat okamžitý a jasný přehled o testovacích výsledcích, což pomáhá jakémukoli členu společnosti porozumět výsledkům testů a využívat je pro další rozhodování. Qase zároveň klade velký důraz na bezpečnost a spolehlivost.

Za použití tohoto nástroje byl postupně vystaven repositář s testovacími scénáři, které obsahují celou webovou aplikaci GoodAccess. Příkladem mohou být testy pro přihlášení do aplikace a využívání všech funkcí aplikace. V případě vývoje nových funkcionalit se repositář testovacích scénářů pravidelně doplňuje o nové testovací scénáře.



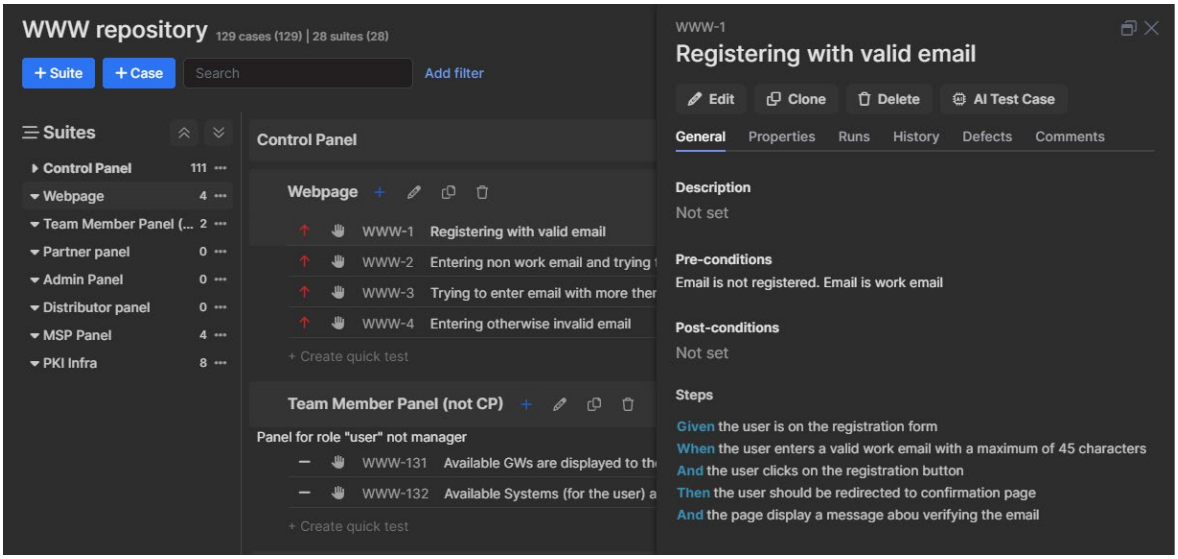
OBRÁZEK 18 - QASE REPOSITÁŘ, AUTOR 2024, PROJEKT GOODACCESS

Dokumentace testovacích scénářů se stala živým dokumentem, který byl neustále aktualizován a reflektoval nejen současný stav aplikace, ale i plánované změny a rozšíření. Každý test byl označkován s přesnou specifikací, zda se jedná o exploratory, regresní nebo jiný typ testu. Tato praxe umožnila lepší orientaci v testech a ulehčila novým členům týmu zorientovat se v procesech.

V rámci nástroje a vytváření testovací dokumentace se využívalo BDD přístupu ve tvoření testovacích scénářů psaných pomocí Gherkin syntaxu. Snahou bylo vytvořit jednoznačné a jednoduše srozumitelné testovací scénáře tak, aby je pochopil každý, i méně technicky znalý spolupracovník. Tato metodika je vhodná v komunikaci s business oddělením ve společnosti, kde je velice jednoduché upravovat testovací scénáře dle potřeb všech kolegů. Testovací scénáře byly zároveň vytvořeny tak, aby si zachovali relativně krátkou délku, byly tedy zahrnuty dílčí funkce místo větších funkčních celků. Tím se zajití izolovanost testů a ve výsledku větší efektivita ve vyhledávání defektů v produktu. Testovací scénáře jsou zpravidla vytvářeny v anglickém jazyce, pro jejich univerzálnost.

Příklad testovacího scénáře, přihlášení uživatele do aplikace:

Given the user is on the registration form
When the user enters a valid work email with a maximum of 45 characters
And the user clicks on the registration button
Then the user should be redirected to confirmation page
And the page display a message abou verifying the email

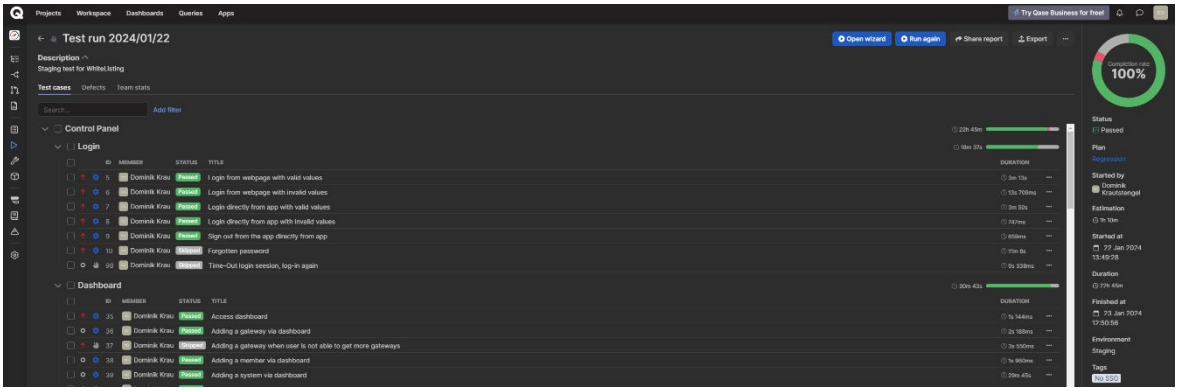


OBRÁZEK 19 - TESTOVACÍ SCÉNÁŘ, BDD STYL, AUTOR 2024

Další funkce nástroje Qase umožňují sestavování testovacích plánů. Je velice jednoduché založit jednotlivé sety, kdy Qase funguje jednoduše tak že se do testovacího plánu postupně zařadí jednotlivé testy, které uživatel vybere. Pomocí tohoto jednoduchého postupy byly následně vybrány testy, které se věnují regresním, smoke testům a podobně. Nejpoužívanějším setem je testovací plán obsahující testy regresní. Regresní testy jsou následně využívány v rámci testování předcházející nasazení na produkční prostředí. Výhodou je dodatečná možnost štítkování jednotlivých setů testů i v rámci

testovacích plánů. To znamená, že můžeme selektivně vybrat množinu testovacích scénářů z repositáře

Dalším využitým prvkem nástroje Qase je schopnost reportování. V rámci spojení repositáře testovacích scénářů a jejich následné propojení s testovacími sety/ plány se po samotném spuštění testovacích setů automaticky vypracovává statistika provedených testovacích scénářů. Ve statistice je na základě výsledků testů uvedeno jak dlouhý časový úsek byl potřeba na vypracování jednotlivých testů, a statistika ohledně úspěšnosti testů. V případě využití automatizovaných testů je nástroj automaticky napojen na runner a čas testů je automaticky synchronizován.



OBRÁZEK 20 - QASE REPORT, TEST RUN. AUTOR 2024

V případě, že některý z testovaných scénářů vyhodnotíme jako neúspěšný, nástroj automaticky vytváří záznam o defektu, který je následně možno integrovat s dalšími službami jako je například GitLab. V tento moment se tedy vytvoří ticket pro vývojářský tým a tester tým informuje o jeho založení. Vývojář následně ticket vyhodnotí a přejde k vypracování nápravy. Následně se pokračuje klasickým procesem tak jak je popsán výše v kapitole 5.1.1. Jednotlivé reporty jsou následně zpracovány testerem a reportovány do aplikace Slack, kde jsou výsledky komunikovány se zbytkem týmu a dalších zainteresovaných stran. Pro tyto potřeby byl vytvořen samostatný komunikační kanál (channel) v aplikaci s názvem „qa-test-results“.

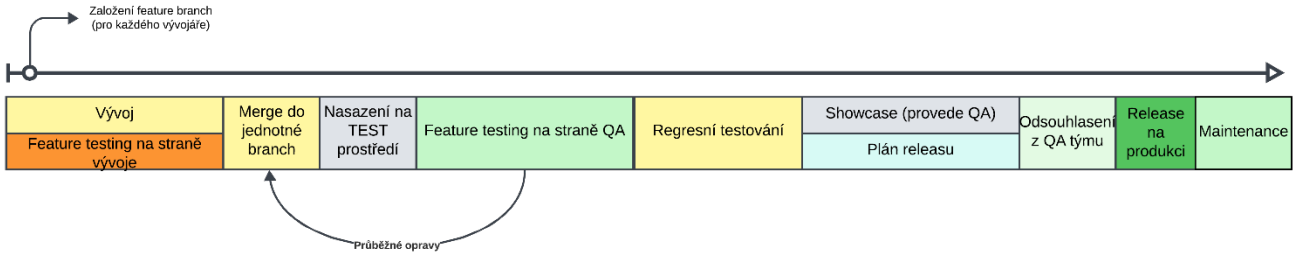
Součástí testovací dokumentace jsou také protokoly vypracovávané v rámci jednotlivých sprintových setkání. Každý sprintový meeting má přiřazen vlastní dokument s protokolem. V novém procesu kontroly kvality jsem zmiňoval, že tester se účastní všech těchto meetingů a analyzuje testabilitu vyvíjených prvků. To znamená že tester je také zařazen do vypracovávání protokolu z meetingu, kdy jsou přidávány jednotlivé body se zpětnou vazbou testera, ke které se následně přihlíží při vývoji. Pro tyto potřeby je využíván nástroj Google Docs. Každý dokument obsahuje soupis zúčastněných na meetingu. Popis s jednotlivými nově zadanými úkoly, jejich status, a nakonec jsou přidávány poznámky. Jde o velice jednoduchý dokument, který přináší vysokou hodnotu do vývoje produktu.

4.1.3 Manuální testy

Manuální testování prošlo menšími úpravami v rámci projektu GoodAccess, jelikož byl proces do určité míry již úspěšně nastaven. Úpravy spočívali především v propojení

procesu provádění manuálních testů s testovou dokumentací. Jak bylo v předešlé kapitole popsáno, šlo o kompletní vystavení testovací dokumentace. Testy byly v dokumentaci roztříděny podle priority a severity a štítkovány podle určení testu (regresní testy, smoke testy a podobně.). Dokumentace musela být zahrnuta do praxe jak manuálních, tak automatizovaných testů. Tedy manuální testy byly nově prováděny na základě dokumentace obsažené v aplikaci Qase, to znamená že pokud měl být spuštěn manuální test nejprve byl otevřen korespondující testovací scénář a podle něj proveden manuální test. Po ukončení manuálního testu bylo vše zdokumentováno a reportováno do týmu.

Další součástí evoluce manuálních testů byla úprava procesu vydávání, který je uveden v kapitole 3.3 a je konkrétně znázorněn na obrázku 7. Proces vydání byl upraven do následující podoby:

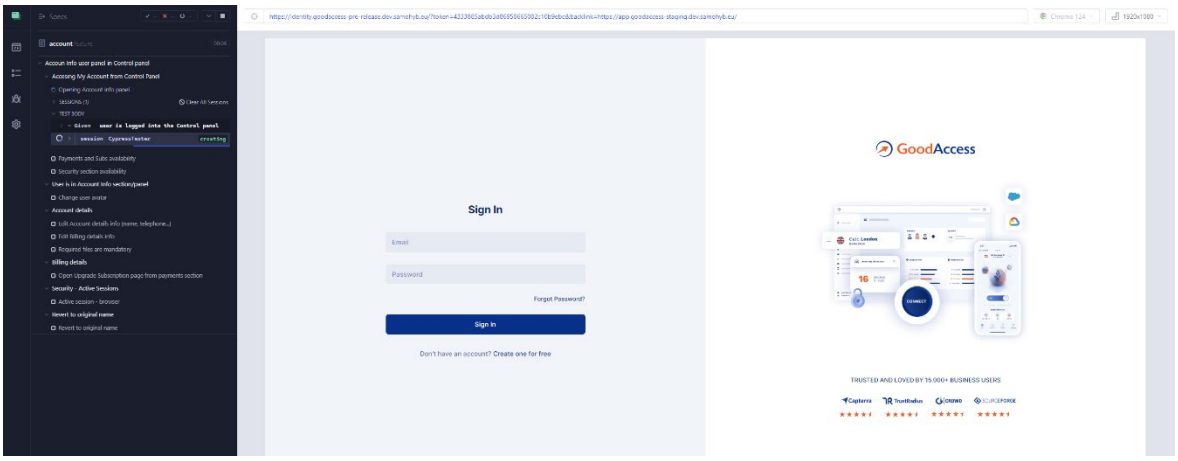


OBRÁZEK 21 - AKTUALIZOVANÝ PROCES VYDÁVÁNÍ, AUTOR 2024

4.1.4 Automatizované testy

V rámci projektu implementace QA procesů ve společnosti GoodAccess byla důležitým krokem selekce testů vhodných pro automatizaci z nově vypracované testovací dokumentace zanesené do Qase. Po této selekci byl pro vypracování automatizovaných testů zvolen framework Cypress. Výběr testů vhodných pro automatizaci je velice důležitý, jelikož ne všechny testovací scénáře jsou pro podobné použití vhodné. Zvolené testy musejí splňovat především to, že se dají stále opakovat bez větších zábran jako na přednastavení systému a zanesení testovacích dat. Minimalizace potřeb čištění testovacích dat po protestování systému je zásadní pro rychlý chod celého testovacího setu. Čištěním dat rozumíme jednoduše řešeno navrácení stavu systému do původního stavu, tak jak byl před početím testovacích scénářů. Toto platí jak pro manuální, tak pro automatizované testy, ale u automatizovaných je důležitost ještě větší v důsledku náročnosti programatického zapracování.

Cypress, založený na jazyce JavaScript, je špičkový nástroj pro automatizované testování webových aplikací, s hlavním zaměřením na UI/UX testování. V GoodAccess byl Cypress začleněn do procesu (CI/CD) v Gitlabu. Tato integrace znamenala, že každá změna kódu nebo přidání nové funkce spustí sérii automatizovaných testů bez potřeby manuálního zásahu. Tím se zjednodušilo testování a umožnilo neustálé dodávání kvalitního softwaru s minimálním rizikem chyb. Vzhledem k limitům GitLabu však bylo automatizované testování omezeno na proces vydávání, jelikož by jinak takové testování bylo příliš náročné na finanční zdroje společnosti.



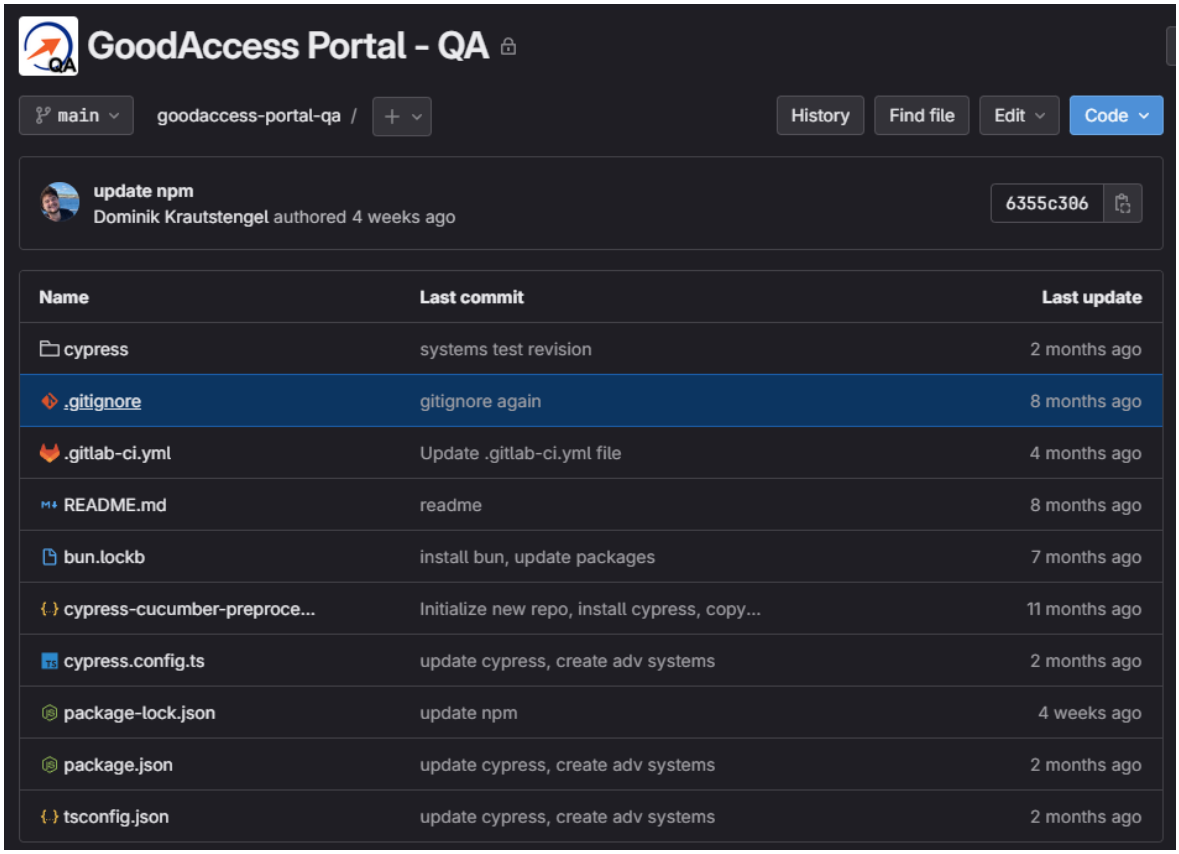
OBRÁZEK 22 - NÁHLED DO DASHBOARDU CYPRESS, AUTOR 2024

Automatické reportování výsledků testů do kanálu "qa-test-results" na Slacku je další klíčovou komponentou implementace Cypressu. Tento krok zaručil, že výsledky jsou snadno dostupné a přehledné pro celý vývojový tým, což zefektivnilo procesy řešení problémů a zrychlilo komunikaci.

Propojení Cypressu s testovacím management systémem Qase umožnilo synchronizaci testovacích scénářů a testovacích sad s testovacími plány vytvořenými v Qase. Tento propojený přístup poskytl kompletní přehled o testovacích aktivitách, umožnil automatizované sledování pokroku a poskytl hlubší vhled do kvality kódu a aplikace. S Qase můžou být výsledky testů spojeny s konkrétními scénáři a sadami, což napomáhá lepšímu pochopení výsledků a identifikaci oblastí, které vyžadují další pozornost nebo vylepšení.

4.1.4.1 Implementace automatizovaných testů pomocí Cypress

Automatizované testy byly implementovány za pomoci frameworku Cypress, jak bylo uvedeno v kapitole výše, tento jazyk je jak bylo zmíněno založen na jazyku JavaScript. Pro implementaci automatizovaných testů jsem zvolil vývojářské prostředí (IDE) VSCode. VSCode od společnosti Microsoft nabízí robustní možnosti úprav programu dle potřeb programátora a nabízí velké množství pluginů podporující BDD a TDD přístupy k programování automatizovaných testů. Po instalaci aplikace VSCode bylo potřeba založit samostatný projekt pro uložení všech nutných souborů pro automatizované testy. Vzhledem k využití GitLabu celou společností byl projekt vytvořen zde s názvem „Goodaccess – QA".



OBRÁZEK 23 - NÁHLED PROJEKTU GOODACCESS QA V GITLAB, GOODACCESS 2024

README.md

GoodAccess QA

This is a repository for GoodAccess Quality Assurance operations. This project utilises Cypress for E2E testing of the web App GoodAccess - Control Panel.

Contributors

Dominik Krautstengel

Automated tests in Cypress/Cucumber

Cypress-cucumber automated tests for GoodAccess. Testing scenarios are defined using Gherkin syntax (from Cucumber) and saved in Feature files. Each step of a Feature file has corresponding step definition written using Cypress.

Steps to run tests locally:

- Clone this repository
- Run `npm install` to install all dependendies
- Set up `.env` file -- there is an **example dotenv** file included in the repository
- Environment is now GoodAccess Pre-Release Dev environment on default, but you can change where tests run by either
  - Changing `baseUrl` in cypress.config.ts
  - Running script with `--config baseUrl="yourChosenURL"`
    - for example: `npm cypress run --browser chrome --config baseUrl=https://something.dev/ --env tags='@public'`
- Run `npm cy:open` to open Cypress GUI and see all tests
- Run `npm cy:run:dev:smoke` to run smoke tests in headless mode on chrome in dev environment

What makes it work:

- Cypress - browser automation tool
- Cucumber - BDD tool for "Living documentation"
- Cypress cucumber preprocessor - package making Cypress and Cucumber work together
  - To understand cypress-cucumber integration better, you can follow this guide.

OBRÁZEK 24 - README PROJEKTU GOODACCESS QA V GITLABU, GOODACCESS 2024

Následně se propojil GitLab projekt s mým repositářem uloženým na mém pracovním stroji, kde jsem pracoval v aplikaci VSCode, propojení proběhlo pomocí SSH klíče vygenerovaného na GitLab a uloženého v rámci Git klienta na mém stroji. Následně je možno vzájemně nahrávat jednotlivé verze a úpravy mezi mým strojem (lokálním klientem) a mezi servery, na kterých je uložen projekt v rámci GitLab nástroje. Pro práci mezi lokálním strojem a GitLabem jsou využívány standartní Git příkazy uvedené v tabulce níže:

Příkaz	Popis
git clone „URL“	Klonuje repozitář z URL na lokální stroj.
git status	Zobrazí stav lokálního repositáře
git add	Připraví všechny změny na nahrání
git commit -m „zpráva“	Vytvoří commit se zprávou
git push	Odelšle změny na server (GitLab)
git pull	Stáhne aktuální stav ze serveru (GitLab)
git branch	Zobrazí větve
git checkout -b „jmeno_vetve“	Vytvoří novou větev s názvem
git merge „jmeno_vetve“	Sloučí aktuální větev s další větví

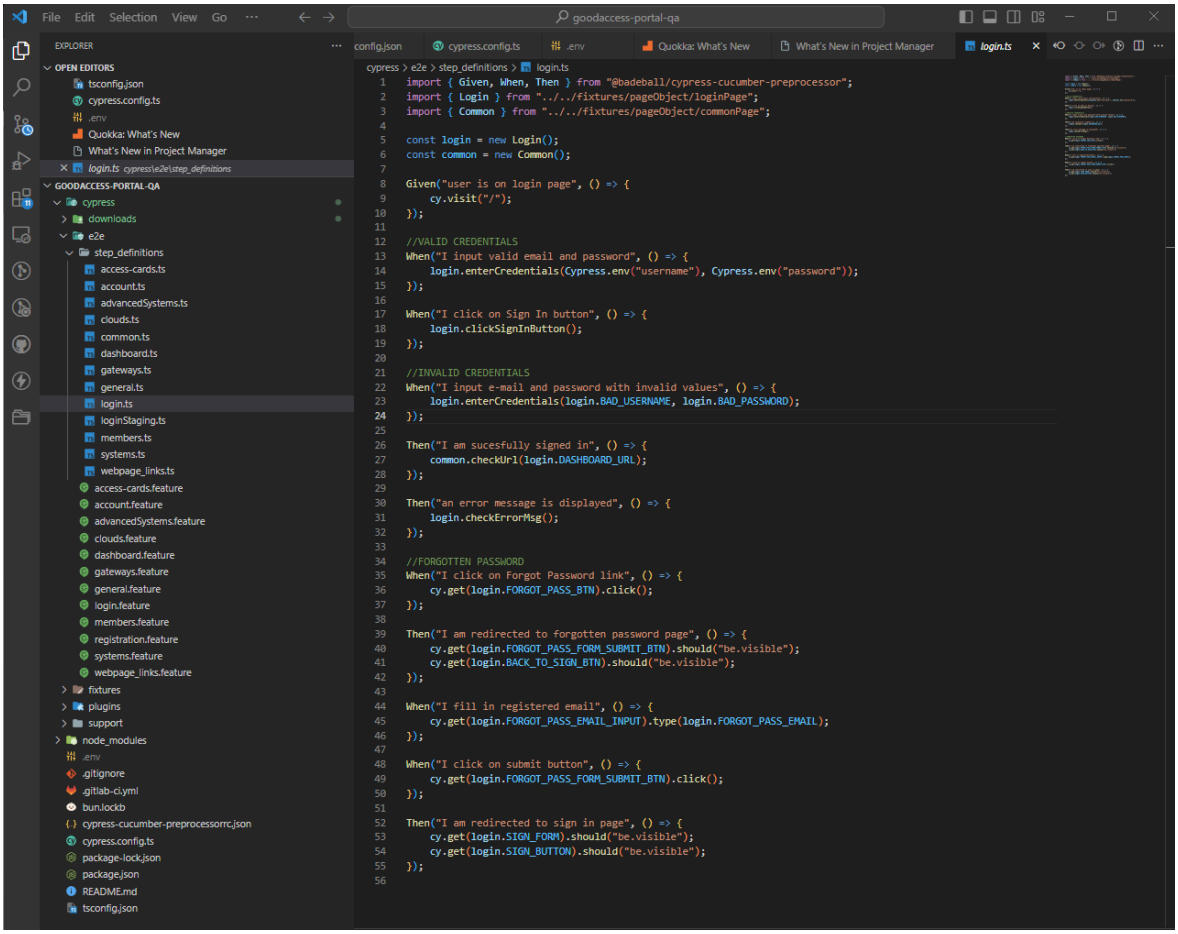
TABULKA 3 - GIT PŘÍKAZY, AUTOR 2024

V GitLabu se pracuje s větvemi, každý nový automatizovaný set testů dostane při vývoji svou vlastní větev, která je po ukončení vývoje pomocí příkazu „git merge jmeno_vetve“ sloučena s hlavní projektovou větví „MAIN“. Následně jsou změny nahrány na vzdálený repositář v prostředí GitLab. V rámci vývoje automatizovaných testů byla zvolena následující struktura projektu:

```
└── REPOSITÁŘ/
    ├── cypress instalace/
    │   └── e2e/
    │       ├── step_definitions/
    │       │   ├── automaticky_test1
    │       │   ├── automaticky_test2
    │       │   └── ...
    │       ├── test_scénář1
    │       ├── test_scénář2
    │       ├── ...
    │       └── page_object
    ├── závislosti_instalace
    ├── env
    └── cypress_konfigurace
```

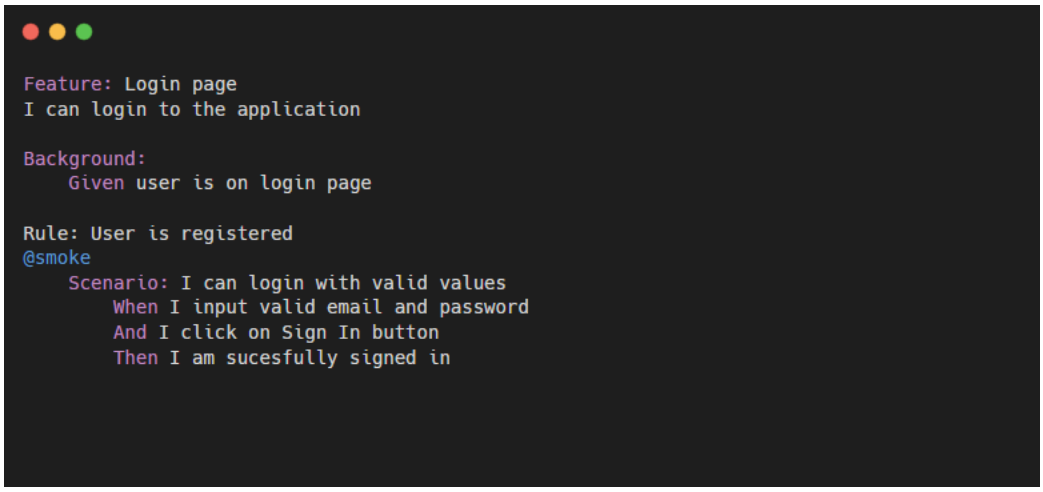
Ze struktury projektu vidíme několik důležitých poznatků o fungování celého projektu. Všechny soubory související s automatizovanými testy jsou umístěny ve složce „cypress“ ta obsahuje složku „e2e“ a „page_object“. Testovací scénáře ve formátu BDD jsou umístěny v samostatné složce „cypress“, soubory obsahující kód automatizovaných testů jsou umístěny ve složce „e2e“ (zkratka pro end to end testing). Soubory testovacích scénářů a automatizovaných testů jsou následně nakonfigurovány pomocí pluginy pro Cypress – „cypress-cucumber-preprocessor“, který soubory vzájemně propojí a dodá frameworku Cypress vlastnosti, které dokážou zpracovat BDD syntax (Given-When-Then) to zapříčiní že Cypress při vykonávání automatizovaných testů vyhledává text (string) ze souborů testovacích scénářů a přiřazuje ho jednotlivým součástem kódu v souborech automatizovaných testů. Jednoduše řečeno testovací scénáře jsou napárovány na odpovídající kód automatizovaných testů. (Bahmutov, 2024) (Hric, 2023)

Automatizované testy jsou tedy programovány v jazyce JavaScript, kdy se využívají metody implementované pomocí Cypress, k tomu je napárována nadstavba BDD pomocí Gherkin syntaxu.



OBRÁZEK 25 - VSCODE, AUTOMATIZOVANÝ TEST PRO LOGIN, GOODACCESS 2024

Pro praktický příklad využiji jednoduchý automatizovaný test pro přihlášení do webové aplikace GoodAccess. Pro tento test potřebujeme podle popisu výše dva soubory vyjma instalačních souborů Cypressu. Potřebujeme soubor obsahující testovací scénář a soubor s kódem automatizovaného testu. Soubor obsahující testovací scénář v BDD stylu vypadá takto:



OBRÁZEK 26 - BDD SCÉNÁŘ PRO PŘIHLÁŠENÍ DO APLIKACE, GOODACCESS 2024

Na uvedeném kusu kódu lze vidět jeho jednoduchá struktura testovacího scénáře zapsaného v BDD metodě. Feature označuje název celého testovacího setu a poté následuje krátký popis. Background označuje prerekvizity testu, to jak má být systém nastaven před provedením testu. V tomto případě nastavujeme, že uživatel je přítomen na stránce pro přihlášení. Následně je dopsán scénář jako takový s jednotlivými kroky, které by měl uživatel následovat, pokud některý z kroku vyústí v chybu nebo je blokovaný, test je vyhodnocen jako neúspěšný a následně se přejde k reportu chyby.

Kód automatizovaného testu pro případ přihlášení uživatele pomocí Cypress vypadá takto:

OBRÁZEK 27 - KÓD AUTOMATIZOVANÉHO TESTU PRO PŘIHLÁŠENÍ, GOODACCESS 2024

Kód v tomto případě využívá metody zabudovány do frameworku Cypress a zároveň je navázán na soubor s testovacím scénářem, to je patrné z textu v závorkám za metodami „Given, When, When“. Ve stejném duchu jsou sepsány všechny automatizované testy pro projekt GoodAccess. Implementováno bylo zhruba 40 %

všech testovacích scénářů. Což zhruba proporčně odpovídá standartu v oboru a vzhledem k povaze testované aplikace.

Automatizované testy následně fungují tak, že framework Cypress čte zapsané metody z kódu automatizovaných testů a na základě zabudovaných metod následně vykonává jednotlivé úkony, při kterých simuluje akce jako kdyby byl reálný živý uživatel aplikace. To znamená že například metoda „visit" navštíví danou webovou stránku. Například příkaz „click" simuluje klikání kurzorem myši na vybraný HTML element na webové stránce. V tabulce níže jsem uvedl nejpoužívanější příkazy z frameworku Cypress. Tyto příkazy jsou majoritně využívány v testovacím kódu v rámci projektu GoodAccess, a tedy této diplomové práce.

Příkaz	Popis
Cy.visit	Navšítív zadanou URL adresu
Cy.get	Vybere HTML element a ověří existenci
Cy.contains	Najde element obsahující podmínku
Cy.click	Simuluje kliknutí kurzorem
Cy.type	Simuluje psaní na klávesnici
Cy.url	Ověří vlastnosti URL odkazu
Cy.scroll	Simuluje pojezd kolečkem myši

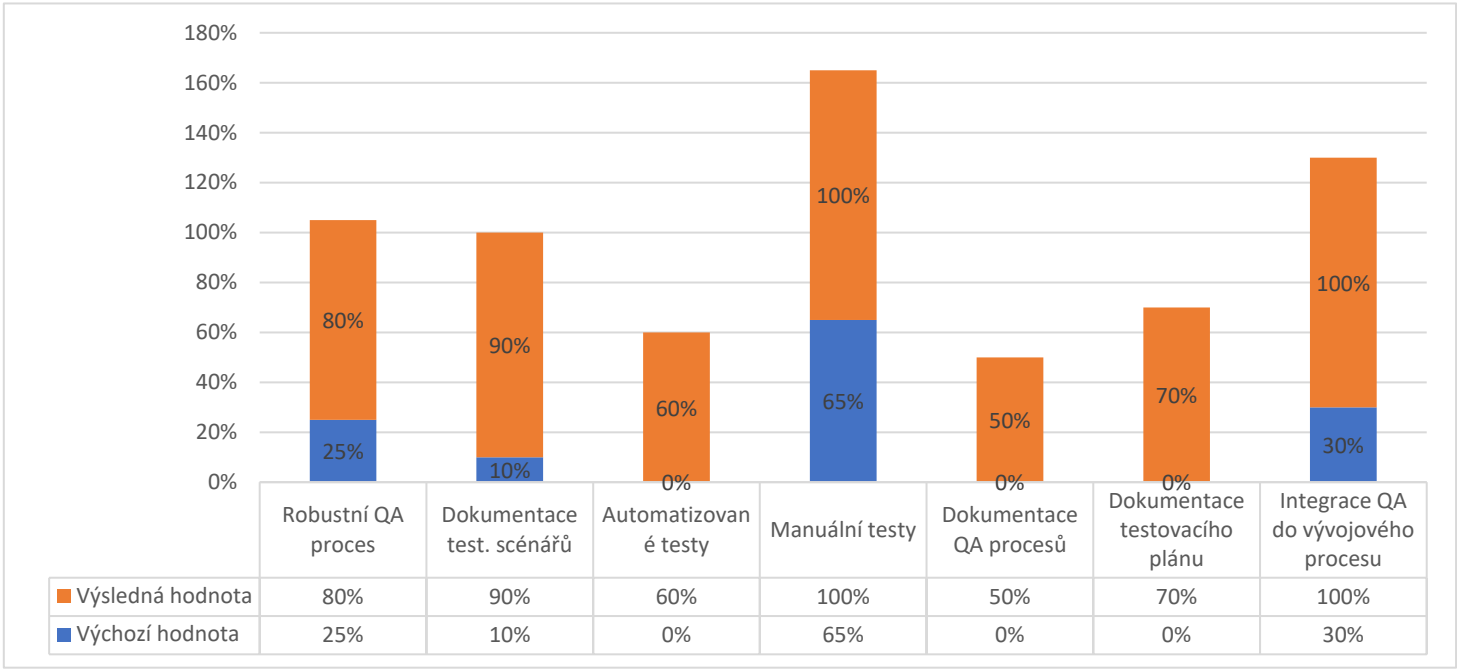
TABULKA 4 - NEJPOUŽÍVANĚJŠÍ CYPRESS PŘÍKAZY, AUTOR 2024

5 Vyhodnocení stavu

Projekt trval v časovém rámci jednoho roku, kdy byly postupně vyhodnocovány dílčí cíle po jednotlivých kvartálech. Vedení společnosti GoodAccess podal na základě tabulky s požadavky na projekt aktualizovanou statistiku, která popisuje procentuální naplnění jednotlivých cílů z pohledu společnosti. Celkové hodnocení projektu je jako kladné, jelikož všechny dílčí úkoly projektu byly úspěšně splněny. Očekávání z naplnění cílů nebylo dosaženo v pokrytí automatizovanými testy a u zavedení kompletní pokrytí v rámci CI/CD. Důvodem pro nenaplnění těchto cílů byl vstup dalších projektů, které zbrzdili vývoj v ohledu na QA a společnost využila zdroje pro věnování se dalším projektům. Nicméně cíle jsou společností považovány jako úspěšně splněné i přes nižší než očekávané pokrytí. Aktuální reportované výsledky projektu jsou uvedeny v tabulce níže:

Požadavek/ Cíl	Cílová hodnota	Výsledná hodnota
Robustní QA proces	80 %	80 %
Dokumentace test. scénářů	95 %	90 %
Automatizované testy	75 %	60 %
Manuální testy	100 %	100 %
Dokumentace QA procesů	50 %	50 %
Dokumentace testovacího plánu	80 %	70 %
Integrace QA do vývojového procesu	100 %	100 %

TABULKA 5 - VÝSLEDKY PROJEKTU, AUTOR 2024



GRAF 1 - VÝCHOZÍ A VÝSLEDNÁ HODNOTA NAPLNĚNÍ CÍLŮ PROJEKTU

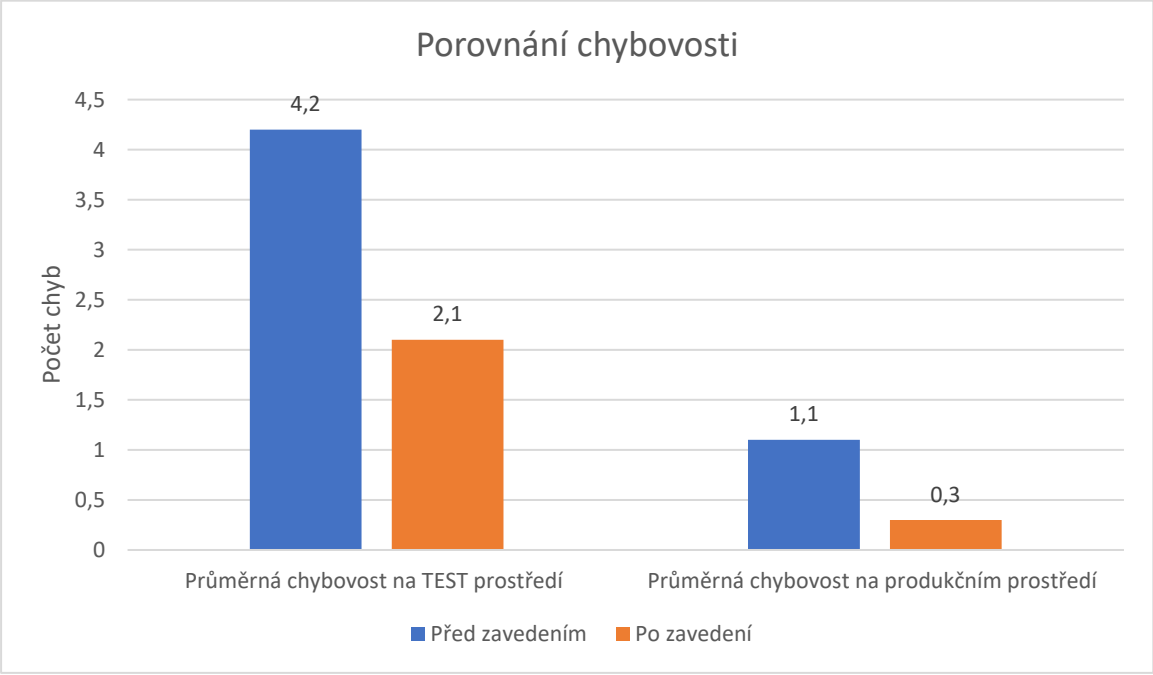
Splnění jednotlivých cílů dle kvartálů:

Kvartál	Splnění cílů
Q1	ANO
Q2	ANO
Q3	ANO
Q4	NE

TABULKA 6 - VYHODNOCENÍ SPLNĚNÍ CÍLŮ V KVARTÁLECH, AUTOR 2024

Z tabulky výše je patrné že dílčí cíle projektu byly úspěšně naplněny v prvních třech kvartálech, poslední kvartál nebyl zcela splněn. Konkrétně se jedná pro dokončení automatizace všech vybraných testovacích scénářů a dokončení automatizace pomocí CI/CD. Tyto cíle byly přesunuty do cílů pro nadcházející rok a je tedy počítáno s prodlevou projektu. Vzhledem k celkově většímu počtu splněných cílů je však společnost GoodAccess spokojena s celkovým výsledkem projektu a shledává efektivnější proces kontroly kvality ve vývoji softwaru.

Další metrikou úspěšnosti projektu byla určena chybovost na produkčním prostředí. Byly porovnány průměrné hodnoty nalezených chyb před a po skončení projektu a následně porovnány, výsledná data jsou dostupná na grafu níže:



GRAF 2 - POROVNÁNÍ CHYBOVOST, GOODACCESS 2024

Jako velice kvalitní byla společností GoodAccess nově vytvořená testovací dokumentace, která následuje nejmodernější trendy v rámci tvoření testovacích scénářů a postupů. Nástroj Qase se prověřil jakožto velice kvalitní nástroj ulehčující práci týmu kontroly kvality. Momentálně GoodAccess reportuje plné pokrytí aplikace testovacími scénáři na základě, nichž se provádí pravidelné manuální a částečně automatizované testování celé aplikace. Samotnou součástí je propojení dokumentace s procesem kontroly kvality jako takové. Společnost si pochvaluje aktivní účast testera na všech relevantních pracovních

setkáních a hodnotí jeho přínos vývoji jako velice efektivní a díky tomu si aplikace zachovává testovatelnou podobu.

Úprava procesu vydávání se ukázala dle GoodAccess jako velice efektivní, jelikož se díky ní upustilo od tzv. guerilla vývoje a celkový proces tak dostal pravidelný a kontrolovaný řád.

Nákladovost na celý projekt je z pohledu zadavatele přiměřená, hlavní metrikou pro výpočet nákladů na zavedení komplexní QA struktury je především strávený čas testera nad celým projektem, kde byl dohodnutý vynaložený náklad na bázi hodinové sazby. Tester vypracovával projekt v rámci 1 „FTE" – ekvivalent plného pracovního úvazku. Do nákladů nebyly započítány náklady provozní – tj. elektřina na provoz testovacího zařízení nebo spotřeba vody. V rámci časového rámce jednoho roku bylo vypočítáno, že projekt jevil časovou náročnost 1462 hodin, jež byly reportovány testerem přímo společnosti GoodAccess za použití měřícího nástroje „Toggl track". Po vynásobení upravenou průměrnou sazbou vyšel náklad projektu na 515 624,- Kč. Reportování bylo prováděno jednou měsíčně. Nespornou výhodou bylo navázání spolupráce se zkušeným testerem a díky tomu se projekt obešel bez nákladů na vzdělávání pracovníka.

V případě že následně porovnáme nákladovost nalezených chyb, můžeme vypočítat úspornost celého projektu oproti výchozímu stavu, kdy porovnáme náročnost opravy chyby v člověkohodinách. Na jednu nalezenou chybu je v průměru potřeba 1,5 člověkohodiny člena vývojového týmu a k tomu je potřeba ekvivalent za jednoho testera. Náklad na nalezené chyby ve výchozím stavu byl stanoven jako 1275,- Kč na chybu. Vzhledem k chybovosti na TEST prostředí v průměru 4,2 se celková nákladovost během jednoho testovacího období rovná v průměru 5355,- Kč / nalezená chyba. Po nasazení procesu kontroly kvality se chybovost na TEST prostředí snížila na 2,1 chyby za testovací období, celkový náklad na chyby v jednom testovacím období nově činí 2677,- Kč / nalezená chyba.

Dalším prvkem v rámci výpočtu nákladovosti chyb je výskyt chyb na produkčním prostředí. V tomto případě se setkáváme jak s nákladem na vyřešení nalezených chyb, tak s možností, kdy je systém mimo provoz pro koncové zákazníky, což vytváří ušlý potenciální zisk společnosti a může být poškozeno dobré jméno společnosti. V tomto případě je výpočet proveden na základy změn chybovosti uvedených na grafech výše. V potaz bereme průměrné hodnoty získané od společnosti GoodAccess. Parametry jsou takové, že na vyřešení jedné nalezené chyby na produkčním prostředí stráví vyřešením problému developer cca 1,5 člověkohodiny a k tomu potřebuje asistenci testera, který stráví prací stejný ekvivalent člověkohodin. Po propočtech vyšla nákladovost na jednu chybu na produkčním prostředí na 1275,- Kč, po úpravě na celkový průměrný počet nalezených chyb před zavedením QA procesů vychází na 1402,- Kč. Naproti tomu zavedení QA procesů snížilo celkový počet nalezených chyb na produkčním prostředí a celkový náklad se sníží na 382,- Kč. Celkově je úspora po zavedení QA procesů v rámci chyb na produkčním prostředí 1020,- Kč v rámci jednoho vydávacího okna. Celková úspora zavedení QA procesů jak v rámci TEST prostředí a produkce společně s celou nákladovostí projektu je následující:

	Výchozí stav	Aktuální stav
N na chyby na TEST prostředí	5355,0 Kč	2677,50 Kč
N na chyby na Produkčním prostředí	1402,50 Kč	382,50 Kč
CELKEM	**6757,50 Kč**	**3060,0 Kč**
	Výchozí stav	Aktuální stav
Náklady na chybovost	6757,50 Kč	3060,0 Kč
Investice	- Kč	515624,0 Kč
N na chyby za jeden rok	351390,0 Kč	159120,0 Kč
Úspora za jeden rok	- Kč	192270,0 Kč
N na chyby za 5 let	1756950,0 Kč	795600,0 Kč
Úspora za 5 let	- Kč	961350,0 Kč
Návratnost projektu	**2,7 let**	

TABULKA 7 - NÁKLADY NA PROJEKT, ZPRACOVÁNÍ AUTOR 2024, ZDROJ GOODACCESS S.R.O. 2024

5.1 Budoucí vývoj

Vzhledem ke zbývajícím cílům projektu, které byly přesměrovány do dalšího roku, a tedy celkově prodloužen projekt se dá očekávat dokončení těchto cílů v budoucnost. Tyto cíle zahrnují kompletní pokrytí určených částí automatizovanými testy a dokončení CI/CD automatizovaného procesu testování a reportingu. Základní sety testovacích scénářů a plánů byly úspěšně vystaveny, nicméně vzhledem k povaze vývoje softwaru v aktuálním stavu vytvořené dokumenty nebudou s časem dostačující. Proto se do budoucího vývoje musí počítat s neustálou údržbou všech dokumentů a dalšímu vývoji procesů. S ohledem na sběr a analýzu dat z testování by se mohl v GoodAccess objevit trend využívání pokročilých analytických nástrojů. Tyto nástroje by umožnily hlubší vhled do procesů vývoje a údržby. Nově nastavený reportingový proces v rámci aplikace Qase a Slack je momentálně dostačující, ale s rostoucím zájmem o produkt společnosti bude potřeba sáhnout k dokonalejším a komplexnějším nástrojům, který trh nabízí.

Umělá inteligence a strojové učení se stávají stále více relevantními pro oblast QA. Integrace těchto technologií by mohla vést k lepší předvídavosti a proaktivní identifikaci rizik, což by mohlo GoodAccess umožnit předcházet problémům ještě před jejich vznikem. Paralelně s tím by integrace uživatelského testování mohla přinést GoodAccess bezprostřední a cennou zpětnou vazbu přímo od koncových uživatelů. Tento přímý kontakt s uživateli by umožnil získávat hlubší pochopení jejich potřeb a preferencí, což by vedlo k neustálému a cílenému vylepšování produktů a služeb. Uživatelské testování by také poskytlo realistický pohled na to, jak skuteční uživatelé interagují s produktem, což by mohlo odhalit nečekané problémy nebo možnosti pro inovace, které by mohly uniknout běžným testovacím procesům. Nehledě na to, že nespornou výhodou uživatelské testování je podpora BDD a TDD přístupu k vývoji, který byl nově zaveden. Koncoví uživatelé dokážou týmu lépe ilustrovat chování koncového uživatele co by ve výsledku bylo implementováno do testovací strategie.

Rozvoj a školení QA týmu by se měly stát prioritou, neboť kvalifikovaný a vzdělaný tým je klíčem k úspěchu v moderním vývoji softwaru. Investice do odborné přípravy by mohly zvýšit efektivitu a schopnosti týmu implementovat nejnovější metody a nástroje v oblasti

zajištění kvality. Dá se očekávat potřeba pro dokonalejší a hlubší znalosti v programovacích jazycích pro celý QA tým ve společnosti. Proto je vhodné přemýšlet o investicích právě do vzdělávání celého týmu, aby zůstal relevantní v neustále se měnícím IT prostředí. Všechny tyto kroky by mohly představovat další fáze v evoluci QA procesů ve společnosti GoodAccess.

Závěr

Diplomová práce s názvem „Zavedení kontroly kvality softwaru ve startupu“. Popsala kompletní vypracování projektu pro zadavatele GoodAccess, který spočíval ve zhodnocení výchozí a stávajícího stavu společnosti v rámci kontroly kvality softwarového produktu a zavedení komplexních procesů kontroly kvality do společnosti startupového prostředí. Práce představila aplikaci postupů pro zavedení inovací do softwarové společnosti na základě moderních metodik, které byly těženy ze získaných znalostí vyučovaných během bakalářského a magisterského studia na MÚVS ČVUT. Tyto znalosti byly spojeny s praktickými zkušenostmi autora z prostředí informačních technologií a s tím spjatých aktivit ohledně vývoje softwaru. Výsledkem je popis zavedení kompletního procesu a způsobu kontroly kvality ve společnosti za použití moderních metod. Práce popsala teoretický základ pro pochopení, jak funguje kontrola kvality v rámci softwarových společností a v praktické části popsala jednotlivé kroky k úspěšnému vypracování cílů vypracovaných společně se zadavatelem.

Naplnění cílů probíhalo v několika fázích během časového rámce jednoho roku, kdy byly postupně dílčí úkoly vyhodnocovány kvartálně. V prvním kvartále se pracovalo především na dokumentaci a počátečních QA procesech. Tester (Autor) se také v prvním kvartále usilovně seznamoval s kompletních systémem aplikace, která byla objektem zájmu celého projektu. V následujících kvartálech se přešlo k usilovné implementaci navrhovaného řešení, konkrétně šlo o dopracování testovací dokumentace a na ni navázané pravidelné manuální testování, následně bylo implementováno automatizované testování v rámci určených testovacích setů sestavených z nově vytvořených testovacích scénářů. V rámci implementace se také pracovalo na zdokonalení a ustálení již existujících vývojových procesů ve společnosti. Z testera se stal release manager a jakožto entita odpovědná za kontrolu kvality byl zdokonalen celý release proces vydávání na produkční prostředí ke koncovým zákazníkům. Ve výsledku byla celkově shrnuta úspěšnost projektu a bylo analyzováno, zda byly naplněny cíle projektu a v jaké míře. Ve spolupráci s týmem pracovníků GoodAccess bylo vyhodnoceno procentuální naplnění jednotlivých cílů a porovnání s výchozím stavem. Lze konstatovat že projekt je hodnocen jako úspěšný, avšak s drobnými rezervami, kdy nebyly kompletně dokončeny dva dílčí úkoly, a to kompletní implementace automatizovaných testů a implementace CI/CD automatizovaných procesů. Společnost tyto cíle přesunula do dalšího roku a jsou zahrnuty v budoucím plánu vývoje QA. Velký posun byl shledán v testovací dokumentaci, kde se ukázalo, že kvalitní testová dokumentace napomáhá k zachování celkové kvality softwarového produktu, a i k satisfakci koncových uživatelů.

Úspěch projektu lze přičíst k pečlivému plánování a zapojení týmu na každé úrovni vývoje. Významnou roli hrálo začlenění rychlých zpětných vazeb z testování do vývojového cyklu, což umožnilo rychlé identifikování a opravování chyb před dalšími fázemi vývoje. Toto proaktivní zapojení nejenže zefektivnilo testovací procesy, ale také podpořilo kulturu neustálého zlepšování, což je klíčové pro dynamické prostředí startupu. Integrace automatizovaných testů a CI/CD procesů byla sice částečně odložena, ale základní struktura a procesy byly již vytvořeny a testovány. Příští fáze projektu bude zahrnovat jejich plnou implementaci a optimalizaci, což povede k dalšímu zlepšení efektivity a kvality výstupů. Tento přístup kvality nejenže splnil očekávání zadavatele, ale také poskytl pevný základ pro budoucí rozvoj a inovace v GoodAccess.

Seznam použité literatury

1. Alami, A., & Krancher, O. (2022). *How Scrum adds value to achieving software quality?* doi:https://doi.org/10.1007/s10664-022-10208-4
2. Alina Mihaela Dima, M. A. (2018). From Waterfall to Agile software: Development models in the IT sector. *Journal of International Studies*, stránky 315-325.
3. Bahmutov, G. (2024). *https://github.com/badeball/cypress-cucumber-preprocessor*. Načteno z GitHub.
4. Baumgartner, M. K. (2021). *Agile testing : The agile way to quality.* Essen: Springer International Publishing AG.
5. Bose, S. (21. Duben 2023). *App & Browser Testing Made Easy.* Načteno z BrowserStack: https://www.browserstack.com/guide/bug-severity-vs-priority
6. Botto-Tobar, M. S.-E. (2022). *Trends in Artificial Intelligence and Computer Engineering : Proceedings of ICAETT.* Springer International Publishing AG.
7. Capers, J., & Olivier, B. (2012). *The Economics of Software Quality.* Boston: Pearson Education, Inc.
8. Das, A. S. (2021). Agile Methodology Vs. Traditional Waterfall SDLC: A case study on Quality Assurance process in Software Industry,. *5th International Conference on Electronics, Materials Engineering & Nano-Technology (IEMENTech)*, (stránky 1-4). Kolkata.
9. de Oliveira, R. B.-N. (Leden 2023). Characterizing the Software Acceptance Testing and the Inclusion of People with Disabilities by Means of a Systematic Mapping. *IEEE Latin America Transactions*, stránky 35-46.
10. Dustin, E. (2002). *Effective software testing : 50 specific ways to improve your testing.* Boston, MA, USA: Pearson Education, Inc.
11. Fairley, P. B. (2014). *Guide to the Software Engineering Body of Knowledge, Version 3.0.* IEEE Computer Society. Získáno 20.03.2024. Březen 2024, z www.swebook.org
12. Forewords, M., Van Deursen, A., & Freeman, S. (nedatováno). *Effective Software Testing.*
13. Galin, D. (2004). *Software Quality: From theory to implementation.* Harlow, Spojené Království: Pearson Education Ltd.
14. Galin, D. (2018). *Software Quality: Concepts and Practice: Concepts and Practice.* John Wiley & Sons.
15. George, B., & Williams, L. (2004). A structured experiment of test-driven development. *Information and Software Technology*(5). doi:https://doi.org/10.1016/j.infsof.2003.09.011.
16. GoodAccess. (2024). *GoodAccess*. Načteno z GoodAccess - Central Dashboard: https://www.goodaccess.com/features/central-dashboard
17. GRAHAM, D. E. (2008). *Foundations of software testing: ISTQB certification. Rev. ed. Australia: Course.*
18. Hendrickson, E. (2013). *Early Praise for Explore It!* Pragmatic Bookshelf.
19. Hric, F. (21. Březen 2023). *filiphric*. Načteno z https://filiphric.com/cucumber-in-cypress-a-step-by-step-guide
20. ISO/IEC 25002:2024. (3 2024). *Systems and software engineering*. International Organization for Standardization, Geneva, Switzerland. Načteno z https://www.iso.org/standard/78175.html
21. Jose, B. (2021). *Test automation : A manager's guide.* BCS Learning & Development Limited.
22. KUBÁLEK, T. a. (2015). *Program pro tvorbu diagramů Microsoft Visio 2013.* Praha: Nakladatelství Oeconomica. Načteno z https://min.vse.cz/visio
23. Leach, R. J. (2019). *Introduction to software engineering.* CRC Press LLC.

24. Levin, M. A. (2019). *mproving product reliability and software quality : Strategies, tools, process and implementation.* John Wiley & Sons, Incorporated.
25. Maneela Tuteja, G. D. (2012). *n importance of Testing and Quality Assurance in Software Development Life Cycle (SDLC) Models .* International Journal of Soft Computing and Engineering (IJSCE).
26. Matsinopoulos, P. (2020). *Practical test automation: Learn to use jasmine, RSpec, and cucumber effectively for your TDD and BDD.* Karatea: Springer Science+Business Media New York.
27. McCall, J. A., Richards, P. K., & Walters, G. F. (Srpen 1977). Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality. Sunnyvale, California, USA.
28. McDonough, J. E. (2021). *Automated unit testing with abap : A practical approach.* Apress L. P.
29. Mendez, D. (2023). *Software Quality: Higher Software Quality through Zero Waste Development.* Mnichov: Springer Nature Switzerland.
30. Miquido. (23. Září 2023). Načteno z https://www.miquido.com/blog/foster-greatness-embrace-scrum/
31. Myers, G. (2015). *The Art of Software Testing.* Willey.
32. Naik, K., & Tripathy, P. (2008). *Software Testing and Quality Assurance: Theory and Practice.*
33. O'Regan, G. (2022). *Concise Guide to Software Engineering From Fundamentals to Application Methods.* Brighton: Springer International Publishing.
34. Petrasch, D. R. (1999). The Definition of ‚Software Quality': A Practical Approach.
35. Poliarush, M. (17. Červen 2022). *Testomat.io*. Načteno z https://testomat.io/blog/writing-bdd-test-cases-in-agile-software-development-examples-best-practices-test-case-templates/
36. *RedHat*. (2022). Načteno z https://www.redhat.com/en/topics/devops/what-is-ci-cd
37. Rose, R. F. (2022). *oftware development activity cycles : Collaborative development, continuous testing and user acceptance.* Apress L.P.
38. Schwaber, K., & Sutherland, J. (2020). *The 2020 Scrum GuideTM.*
39. Schwager, M. (2010). *Masarykova univerzita Fakulta informatiky DIPLOMOVÁ PRÁCE Automatizace funkčních testů (HP QuickTest Professional).*
40. Shatnawi, & Alazzam. (Duben 2022). An Assessment of Eclipse Bugs' Priority and. Jordán, Jordánsko.
41. Silhavy, R. (2020). *Comparative analysis of Software development life cycle models (SDLC).* Springer International Publishing AG.
42. Stamelos, I., & Sfetsos, P. (2007). *Agile software development quality assurance.* Information Science Reference.
43. Tian, J. (2005). *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement.* New Jersey: John Wiley & Sons, Inc.
44. TOPOLOVÁ, I. M. (2017). *Manažerská informatika. Textový procesor Microsoft Word 2016.* Praha: Nakladatelství Oeconomica.
45. Tozzi, C. (28. Prosinec 2023). *Understanding Bug Severity vs. Priority: Key Differences and Best Practices.* Načteno z SauceLabs: https://saucelabs.com/resources/blog/bug-severity-vs-priority
46. Williams, L. (2005). *Software Testing and Continuous Quality Improvement.* AUERBACH PUBLICATIONS.

Seznam obrázků

Seznam tabulek

Seznam grafů