Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Strategie zajištění kvality softwaru

Aplikace ve výrobní společnosti

Diplomová práce

Autor: Bc. Aleš Stránský
Studijní obor: Informační management

Vedoucí práce: Mgr. Josef Horálek, Ph.D.

Pracoviště: FIM – Katedra informačních technologií

Hradec Králové Duben 2022

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

……..

V Hradci Králové dne 26.4.2022 Aleš Stránský

Poděkování:
Děkuji vedoucímu diplomové práce Mgr. Josef Horálek, Ph.D. za metodické vedení práce, ochotu, trpělivost a čas, který mi věnoval. Dále bych rád poděkoval své rodině za obrovskou toleranci a podporu, díky které mohla tato práce vzniknout.

Anotace

Práce se zabývá vytvořením testovací strategie v prostředí výrobní firmy založenou na microservices. Cílem je dosažená filozofie DevOps s požadavkem na ideální poměr cena a přidaná hodnota testováním. V teoretické části se zaobírá automatickým a manuálním testováním, testovací pyramidou, názvoslovím a pojmy nejenom z ověřování kvality, přístupy k vývoji a testování, jednotlivými druhy testů a jejich rozdíly. Praktická část je věnována vypracování samotné strategie podle posledních doporučení a zkušeností z ověřování kvality a samotného přístupu k vývojovému cyklu softwaru. Přináší přehled o všem spojeném s ověřením kvality a to nastavení rolí, druhy testů, procesů a výstupů z testování a celým životním cyklem vývoje softwaru od návrhu až po nasazení na produkční prostředí a následnou správou a monitoringem.

Annotation

Title: Software quality assurance strategy in a manufacturing company

The thesis deals with the creation of a testing strategy in a manufacturing company environment based on microservices. The goal is to achieve a DevOps philosophy with the requirement of an ideal cost/value ratio by testing. The theoretical part deals with automated and manual testing, the testing pyramid, terminology and concepts not only in quality assurance, approaches to development and testing, different types of tests and their differences. The practical part is devoted to the development of the actual strategy according to the latest recommendations and experience in quality assurance and the actual approach to the software development cycle. It provides an overview of everything related to quality verification, namely the setup of roles, types of tests, testing processes and outputs, and the entire software development life cycle from design to deployment to the production environment and subsequent management and monitoring.

Obsah

Seznam obrázků

Seznam tabulek

1 Úvod

Před začátkem každého vývoje vyvstane vždy otázka jak nejlépe, nejlevněji a bez chyb dodat novou funkcionalitu uživatelům. Leč na tuto otázku není jednoznačná odpověď. S novými poznatky a tím se měnícím přístupu k vývoji mění rok od roku. Jedna věc zůstává pořád stejná, a to čím dál větší důraz na kvalitu a především její automatizaci. Celý obor testování prošel vývojem od techniků, kteří se zabývali převážně debuggováním prvních strojů a hledali příčinu jejich selhání po oddělení těchto dvou odborů v sedmdesátých letech. Kdy se začalo spoléhat na dokola se opakující manuální testování podle předem daných scénářů v testovacích týmech oddělených od vývoje a s přesně danou strukturou od test manažera, přes test designera, až k manuálnímu testerovi. V průběhu této doby se začínají objevovat první pokusy o změnu ve vývojových týmech s cílem začít testování co nejdříve a tím dříve dokončit vývoj. Exploratory testování dostává svůj název a první testeři začínají rozvíjet svoje analytické schopnosti, a tím odhalovat více chyb než při klasické metodologii. Zásadní změna přichází s miléniem, a to agilní vývoj, kdy se testeři stávají součástí vývojových týmů, kde se dostávají do kontaktu s aplikací už ve fázi návrhu, automatizace regresních testů se stává běžnou praxí a testovatelnost aplikace je jeden z požadavků při návrhu na jakých technologiích a jak se bude aplikace vyvíjet. Završením, tedy prozatím, je filozofie DevOps, kde se z testerů stávají automatizační inženýři nebo spíše DevOps inženýři, a co je mezi požadavkem od zákazníka až po dodání je automatizované. To vše od prvopočátku testování bylo vždycky zaneseno v jednom dokumentu, a to testovací strategii. Přehled toho, jak se bude k ověření kvality při vývoji přistupovat, jaké budou role, procesy a podobně, což dává při každém novém vývoji možnost manažerovi kvality udělat průzkum trhu a zanalyzovat nové nástroje, přístupy nebo pohledy na testování. To je případ i této diplomové práce, která se zabývá vytvořením strategie testování softwaru ve výrobní společnosti, kde bylo a je testování přeneseno na uživatele a vývojový cyklus je dokončen při dodání nové funkcionality na testovací prostředí. V teoretické části se zaměří obecně na ověřování kvality a v praktické implementuje získané poznatky. Kde není jasný český překlad nebo není zažitý komunitou, bude používat originál.

2 Literární rešerše

Pro svoji diplomovou práci jsem se zaměřil na zdroje vydané nebo aktualizované v posledních několika letech, a to z důvodu toho, že se testování rapidně vyvíjí a chtěl jsem obsáhnout poslední trendy a zkušenosti.

Hlavní zdroj informací pro tuto práci je kniha s názvem Efektivní testování softwaru [1] v níž není asi ani jedna kapitola, ze které jsem nepoužil nějaké poznatky. Celá kniha je napsaná lidmi z praxe a zaměřuje se hlavně na efektivitu a dobré plánování. Bohužel zkušenosti autorů jsou většinou jen z vodopádového přístupu k vývoji, tak jsou agilní metodiky a DevOps zmíněny jenom okrajově. Jako jedna z mála také je v češtině.

Názvosloví a rozdělení beru podle standardu ISTQB, [6] a to hlavně z důvodu, že je mezi všemi testery dobře známé. Bohužel dále moc jako zdroj nevyužívám, protože si myslím, že jejich přístup, i když jsou pokusy jako certifikace agilní tester reagovat na poslední trendy pořád spadá do minulého století. Což nic nemění na tom, že si myslím, že základní certifikace je velice přínosná.

Kniha s poetickým názvem Umění testování [5] mi změnila myšlení ohledně provádění nebo spíše plánování testů, kde se nějaké postupy nebo analýzy dají považovat za kousek umění. Zmínil bych i její přístup k testování použitelnosti. Pohledu na agilní testování a podle mě velice dobře rozebranou kapitolu ohledně testování internetových aplikací.

Jedno z mých prvních setkání s pojmem continuous testing, což se dá přeložit jako průběžné testování bylo v této knize. [25] Ihned jsem si tento a DevOps přístup za velice přínosný a snažil se více zapracovat do této práce nebo aspoň jeho filozofii. Zmínil bych a doporučil kapitoly ohledně výše zmíněného doplněného o CI pipelines, testování webových API, vývoji automatizovaného frameworku, mockování dat jako součást průběžného testování a rozhodně shifted-left přístupu.

Budoucnost zajištění kvality softwaru [24] je kniha zaměřující se přímo na testování v DevOps a hodně jsem z ní o tom čerpal. Zákazník je v centru testování a specializované testy jsou další zajímavé kapitoly.

3 Testování softwaru

Software je všude kolem nás. Od jednoduchých kusů kódu v každé mikrovlnce nebo pračce, přes složitější systémy v našich chytrých zařízeních až po aplikace, které řídí jaderné elektrárny a bankovní systémy. Neexistuje obor, který s ním není aspoň okrajově spojen. Když se tato softwarová aplikace, komponenta, součást vyvine nebo se v ní provede nějaká změna, přichází fáze ověření kvality, aby při dodání na trh byla podle požadavků a bez nedostatků neboli testování softwaru. Testování může být dvojího druhu, a to manuální a automatizované. Selhání nebo neočekávanému chování se říká chyba (bug) a může způsobit ztrátu peněz, reputaci firmy, času, a dokonce i zranění nebo smrt. Předcházení chyb a selhání se zabývá testování softwaru. Definice z Economic Times:

„Účelem testování softwaru je identifikovat chyby, nedostatky nebo chybějící požadavky v porovnání se skutečnými požadavky.“ [9]

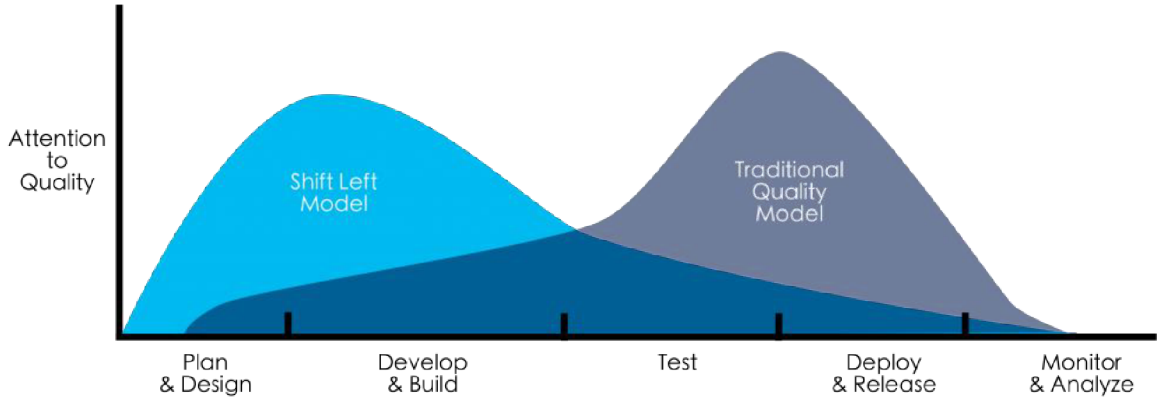
Podle mezinárodní testovací organice ISTQB:

“Testování softwaru je způsob, jak posoudit kvalitu softwaru a snížit riziko selhání softwaru v provozu.” [6]

V literatuře se objevuje samozřejmě i pojem Quality Assurence (QA), který je nadstavbou nad softwarovým testováním a zaměřuje se kromě softwaru i na celý proces ověřování kvality. [10]
První zmínky se začaly objevovat s prvními výpočetními stroji, a to v padesátých letech dvacátého století. V této době se stíral rozdíl mezi softwarovým testováním a debuggováním. Glenford J. Myers v roce 1979 pronesl a z toho se dá odvodit rozdělení těchto dvou disciplín:

„Úspěšný testovací případ je takový, který odhalil dosud neodhalenou chybu.“, [5]:6

Testování softwaru, jak ho známe dnes, se dá datovat do roku 1988, kdy William C. Hetzel nově rozdělil testování do fází plánování, návrh, tvorby, údržby, provedení testů a testovacího prostředí. [7] Testování softwaru se vyvíjelo a z absence v životním cyklu softwaru se stalo jeho nedílnou součástí. Když vezmeme vývoj softwaru jako přímku, kde na začátku je požadavek, respektive plánování a na konci monitorování produktu je vidět, jak se postupně fáze testování posunula doleva – Shift-left, jak je známo i z jiných oborů. Začalo se objevovat v předprodukční fázi a posunulo se až k samému začátku, a to do návrhu softwaru, kde se i požadavky dají testovat. [8]



Obrázek 1 - Shift left model
Zdroj: Shift Left. Home [34]

Manuální a automatizované testování jsou dva vzájemně se doplňující způsoby, kterými lze software testovat. Zatímco manuální testy jsou sestavovány na základě znalostí testujícího inženýra o testovaném systému, automatizované testování se spoléhá na výpočetní výkon stroje, který může provést mnohem více testů, než by dokázal člověk v omezeném časovém čase.

3.1 Manuální testování softwaru

Jak název napovídá, manuální testování je takové, při kterém je aplikace testována manuálně. Všechny testy jsou prováděny testerem, jeden po druhém a výsledky testování jsou porovnávány podle zadání nebo specifikace. Jedná se o základní formu testování a může být prováděna i testery se základní znalostí aplikace nebo

testování. Jedná se o nezbytný krok v cyklu vývoje softwaru, který byl bohužel upozaďován. Poslední trendy ukazují a tím pádem stále více odborníků dává přednost automatizovaným testům, ale stále existuje spoustu důvodů, které ospravedlňují použití manuálního testování. Základní použitelnost a vzhled může posoudit a vyhodnotit pouze člověk, protože je vyvíjen pro něj a dokáže se na celou aplikaci podívat v širším pohledu než jen stále se opakující naprogramovaný test. Automatizace je finančně náročná proto se vyplatí zejména, když se testy musí často opakovat. V případech jednorázových testů se stále vyplatí rychlá ruční validace. [5]

Testování použivatelnosti (Usability testing, UX testování), což je testování, zda je aplikace přívětivá a její používání je jednoduché, je také neproveditelné pomocí nějakých nástrojů.

Výhody manuálního testování

- Rychlá zpětná vazba
- Méně nákladné (při jednotkách opakování)
- Lidský faktor (úsudek a intuice) při testování

Nevýhody manuálního testování

- Člověk je náchylný k chybám
- Manuální testy nejsou znovu použitelné
- Specifické testovací požadavky nejdou provést

3.2 Automatické testování softwaru

Automatické testy jde definovat jako použití automatizačních nástrojů k provedení testů bez manuálního zásahu. Porovnává očekávané a skutečné výsledky testů a umí vytvořit podrobné zprávy. Cílem automatizace není zcela nahradit manuální testování, ale snížit počet ručních ověření kvality. Každá úprava aplikace vyvolává potřebu znovu otestovat předchozí funkcionalitu, v tomto směru jsou automatické testy nejvhodnější volba. Zavedení těchto testů vede k úspoře peněz a času. V případě implementace continuous integration je to naprostá nezbytnost. Jako automatické testování se dá použít nástroj nebo program na opakování

jednoduché operace. Jako třeba validace PSČ nebo e-mailové adresy, načtu si všechny vstupy a jeden po druhém je nechám aplikaci provádět místo zdlouhavého a neefektivního manuálního testování. Před vytvořením automatického testu musí být systém nebo aplikace otestována manuálně a provedena analýza, jaké budou nejlepší vstupy a co test pokryje. [1] Správné není napsat co nejvíce testů, ale napsat co nejméně testů, které pokryjí co největší část aplikace. V tomto případě se jeví ideální použití Paretova pravidla, kdy dvacet procent testů pokrývá osmdesát procent aplikace (80% test coverage). [1][15][11][12]

Výhody automatického testování

- Nachází více chyb než člověk
- Efektivnější a rychlejší proces
- Jdou opakovaně použít
- Můžou běžet kdykoli a jakkoli dlouho
- Snadno zvedne produktivitu
- Velký výběr automatizačních nástrojů
- Software nikdy nezapomene zkontrolovat i tu nejmenší část

Nevýhody automatického testování

- Neumí otestovat uživatelské přívětivost (UX)
- Některé nástroje jsou finančně náročné
- Nejsou 100% spolehlivé
- Testy potřebují údržbu - ta může být nákladná

3.3 Testovací a další pojmy

Testování jako každý jiný obor má svoje vlastní specifické názvosloví. Zde zmíním ty nejdůležitější a nejpoužívanější nejenom z oboru testování softwaru.

Equivalence Partitioning

Rozdělení ekvivalence je technika na rozdělení testovacích dat do ekvivalentních celků na validní a nevalidní, z kterých jdou odvodit testovací případy. Jako příklad jde uvést testování aplikace, že vstup může načíst jenom číslo o devíti znacích. Tak rozdělení ekvivalence bude do tří skupin. Menší než devět. Přesně devět. Více nebo rovno než deset. Můžu přidat i čtvrtou skupinu, kde vyzkouším jiný vstup, než čísla jako speciální znaky nebo písmena. [6]

Boundary Value Analysis

Jedná se o nadstavbu předchozího rozdělení ekvivalence. Podle této techniky jsou do testů zaneseny hraniční hodnoty testovacích hodnot. Jako příklad u předchozího budou hraniční hodnoty, devět jako správná hodnota, osm jako hodnota, od které níže je vstup špatný a analogicky deset od které výše je vstup špatně. [1]

Decision Table Testing

Jedná se o hojně používanou testovací techniku pro návrh testovacích případů. Různé kombinace vstupů a jejich příslušné chování je zachycené v tabulkové formě. Používá se k systematickému výběru vstupů k testování. Tato technika je vhodná pro testy, kde mají dva a více vstupů mezi sebou logický vztah. Jako příklad jsou uvedeny jednoduché podmínky při výběru z bankomatu. [6]

Tabulka 1 - Decision Table Testing

Podmínky			
Částka výběru <= Zůstatek	T	F	F
Aktivní kontokorent	T	T	F
Akce			
Povolen výběr	T	T	F

Zdroj: vlastní zpracování

State Transition Testing

Jedná se o testovací techniku, kde se sleduje, jak se změní stav aplikace při různých vstupech, zkouší se i záporné stavy. Podle vstupních hodnot může být stav aplikace připraven, nepřipraven, otevřen, neotevřen a podobně. [6]

Use Case Testing

Testování případů použití je testovací technika, která se používá k identifikaci případů k pokrytí celého testovaného systému. Transakci po transakci konkrétního použití systému. Účelem je najít „cestu" programem, která pokryje co nejvíce jeho funkcionality, respektive testovacích případů. [1][5]

Test coverage

Pokrytí testů je metodika používaná při testování softwaru. Ukazuje kolik procent zdrojového kódu je provedeno určitou sadou testů. Čím je procento větší, tím více kódu se spustí neboli otestuje. Jedná se o jednu z prvních metodik pro systematické testování softwaru. První zmínku o něm publikovali v roce 1963 Miller a Maloney. [16]

Pesticide paradox

Dokola opakující se stejné testy po nějaké době přestávají nacházet nové bugy a je potřeba změnit nastavení nebo přidat další testy. Název je v analogii k pesticidům, kdy po jejich dlouhodobé aplikaci přestávají být na škůdce účinné, tak jako na začátku. V některých případech, jako je automatizace regresních testů, je tento paradox přínosem. Chyba se objeví jen jednou, protože příště je už zkontrolována novým testem. [5][6]

Application programming interface (API)

Definuje způsob komunikace k propojení mezi počítači nebo počítačovými aplikacemi. Každé API má svojí specifikaci, jak se sestavuje nebo připojuje k ostatním. Výhoda je, že komunikace pomocí API je nezávislá na programovacím jazyku. Každá komponenta může být napsána v něčem jiném, ale když splňuje stejnou specifikaci tak není problém v komunikaci. [1][24][25]

White box nebo Black box testování

Testování černé nebo bílé skříňky je jedno ze základních rozdělení v testování softwaru. Název napovídá, o co se jedná. White box testuje, když má tester přístup ke kódu a přesně vidí, co se děje. Kde se načte, jaká proměnná, jaká část kódu se provádí. Oproti Black box testování – černé skříňce, kdy tester nevidí, co se děje uvnitř kódu programu a sleduje vstup a výstup aplikace. Někdy se objevuje i grey box testing, testování šedé skříňky. Jedná se o kombinaci předešlých dvou metod. Hlavním účelem tohoto testování je najít závady ve struktuře nebo nesprávným používáním. [1][5][6]

Statické nebo dynamické testy

Jedná se o rozdělení technik testování softwaru. Jak už název napovídá, statické testování je nad zdrojovým kódem, bez jeho spuštění. Dynamické testování se provádí nad spuštěným zdrojovým kódem a analyzuje vstupy a výstupy. Ke statickému testování se nejvíce používají check listy (kontrolní seznamy) a u dynamického test cases (testovací případy). [5][6][25]

Funkční nebo nefunkční testy

Další rozdělení je podle toho, na co se u testování zaměřujeme. Při funkčním testováním jde hlavně o funkční požadavky, které popisují funkce systému. Aplikace splňuje, co je v požadavcích. Správně počítá, otevře se správné okno, komunikuje s požadovanou službou a tak podobně. U nefunkčních testů se jedná o testy nefunkčních aspektů jako výkon, použitelnost, spolehlivost a třeba uživatelská přívětivost. [1][5][25]

Front-end a Back-end

Název sám o sobě je vypovídající. Front-end neboli uživatelské rozhraní je to, co je vidět, tedy vizuální stránka aplikace. Tam kde zadávám vstupy a dostávám informace zpět. Oproti back-endu kde se dějí všechny „zázraky“, co se zobrazují na front-endu. Jedná se o server, samotnou aplikace a třeba databázi. Vývoj front-endu vyžaduje úplně jiné znalosti oproti back-endu. Vývojář bez důkladného tréninku není schopen přecházet z vývoje jednoho k druhému. Trendem vývoje

front-endu poslední doby je vyvíjet ho na webových technologiích jako jsou HTML, CSS a JavaScript. [5][6][7]

Testovací artefakty

Anglicky testing artifacts nebo občas používané testing deliverables (výstupy testování) jsou reporty nebo dokumenty vytvořené před, při nebo po realizaci testování. Pomáhají zajistit, aby byli všichni zainteresovaní informováni o průběhu projektu. [5][6]

Entry criteria

Vstupní kritéria jsou souborem podmínek nebo požadavků, které je třeba splnit nebo dosáhnout, aby byly vytvořeny vhodné a příznivé podmínky pro testování. [6][24]

Exit criteria

Výstupní kritéria jsou důležitým dokumentem, který připravuje testovací tým, aby bylo možné dodržet stanovené termíny a přidělený rozpočet. Tento dokument specifikuje podmínky a požadavky, které je třeba dosáhnout nebo splnit před ukončením procesu testování softwaru. [6][24]

Definition of Ready (DoR)

Definice připraveného je seznam toho, co musí úloha splňovat, aby se na ní mohl začít vývoj. Příklad: úloha musí mít odhadnutou časovou náročnost, zadání a musí být schválen její vývoj od managementu. [24][25]

Definition of Done (DoD)

Definice hotového je dohodnutý seznam v týmu toho, co musí být splněno, aby se úloha mohla uzavřít. Příklad: aby mohla být úloha na novou funkcionalitu uzavřena, musí být funkcionalita vyvinuta, musí být po code review, napsané unit testy, otestováno, zdokumentováno; automatické testy musí být nasazeny a product owner souhlasí s nasazením na produkční prostředí. [24][25]

Remote Function Call (RFC)

Vzdálené volání funkcí je chráněné rozhraní SAPu. RFC obsahuje dvě standardní rozhraní, jedno pro komunikaci mezi SAP systémy a druhé pro přístup aplikací, které nejsou součástí SAP systému. Pokud nějaká aplikace chce získat data ze SAPu, musí využít Remote Function Call**.**

Vlastnost, ne chyba

Výrok "a feature, not a bug." (je to vlastnost, není to chyba). Používá se k popisu něčeho, co na první pohled může vypadat jako chyba, ale ve skutečnosti je to zamýšlené chování aplikace.

Traceability Matrix

Matice sledovatelnosti požadavků je matice zobrazující vztahy mezi požadavky a testovacími scénáři. Je to dokument, který mapuje testovací scénáře vůči požadavkům na vyvíjenou aplikaci. Pomáhá zjistit, jak úplné jsou testovací případy. [6]

Regresní testy

Jedná se o testy, které ověřují, že vyvinutá aplikace stále funguje. Většinou se spouští po tom, co se vyvine nějaká nová funkcionalita, před vydáním nové verze nebo v pravidelných iteracích, protože nejenom změna v aplikaci, ale i změna v prostředí může do systému zanést chybu. Regresní testy jsou vzhledem k tomu, že jsou pravidelně opakované ideální k automatizaci. Investované zdroje a čas se několikanásobně vrátí. [1][5][12]

User experience testy

Nedílnou součástí ověřování kvality jsou UX testy neboli testy uživatelského zážitku nebo uživatelské přívětivosti. V rámci těchto testů se zjišťuje, jestli je software pro uživatele přívětivý, najde hned co potřebuje, neztratí se, ovládání je podle zažitých standardů, jsou všechny ovládací prvky tam, kde jsou očekávané? Například jestli je scroll-bar napravo a na stisknutí pravého tlačítka se otevře kontextová tabulka. Zmáčknutí F1 vyvolá nápovědu a tak podobně. Nadstavbou je

UX design prováděný UX designerem, který si většinou vezme běžného uživatele a nechá ho procházet softwarem a zjišťuje, jestli jsou všechny prvky tam, kde je uživatel očekává. Jestli ví, co se po něm chce a kam má případné vstupy zadat.

Performance testy

Zátěžové testování, stress testing nebo výkonové testování ověřuje, že se aplikace nezhroutí pod zátěží. Jeden z testů zjišťuje, co se stane, když se v jednu chvíli připojí více uživatelů nebo když se nechá aplikace běžet v dlouhodobém horizontu, zdali nezačne vytěžovat více hardwarových zdrojů, než bylo očekáváno. Jak dlouho trvá zpracování požadavku a tak podobně. Výkonové testování by mělo být prováděno před každým nasazením na produkční prostředí nebo aspoň v pravidelných iteracích, aby se předešlo spadnutí aplikace v ostrém provozu. [6]

Smoke testy

Kouřový test je rychlý a jednoduchý test, zda je aplikace funkční. Nejvíce se využívá k ověření, jestli je build neboli nová verze aplikace stabilní. V případě, že nalezne nějaký bug, měly by následovat další testy. [8]

Exploratory testing

V češtině se objevuje jako průzkumné testování. Jedná se o manuální, dynamickou a black box techniku založenou na zkušenosti testera, která se dá popsat jako současné učení, navrhování a provádění testů. Poprvé termín zavedl Cem Kaner [11] na konci osmdesátých let, který zjistil, že testeři, kteří se neřídí podle přesně daného testovacího plánu, najdou více bugů než ti, kteří se jím řídili. Je to protipól k testování, kdy se nejprve nastuduje dokumentace, a pak se postupuje podle daného testovacího scénáře. Při exploratory testing závisí kvalita testování na dovednosti testera vymýšlet nové testovací případy a nacházet chyby. Čím více ví o produktu, jednotlivých metodách a přístupech testování, tím je testování kvalitnější. Při tomto testování jsou výsledky a očekávání otevřená. Kriticky zkoumá provoz, konfiguraci a výsledky. Reportuje chyby, neočekávané chování a výsledky, které se zdají být pravděpodobnou chybou. Software se testuje tím, že se s ním tester seznamuje a na základě těchto poznatků postupuje dále a činí tak

nová individuální rozhodnutí. Nikdy se neřídí pouze předem stanoveným postupem, ale osvědčený postup je si předem definovat na jakou část se testování bude zaměřovat. Z exploratory testování může vzniknout i dokumentace, co a jak bylo otestováno a pak použito jako testovací report. Jako dobrá analogie se jeví použití termínu "průzkum bojem". [1][11][5][6]

Supply Chain Management (SCM)

Do češtiny přeloženo jako: „Řízení dodavatelského řetězce" a jak název napovídá, jedná se o cestu výrobků, dat, peněz nebo služeb od nákupu surovin až k zákazníkovi na místo určení. Popis svádí k tomu, že se jedná převážně o logistiku, ale ta je jen jedna z jeho částí. Činnosti dodavatelského řetězce zahrnují nákup, řízení životního cyklu výrobků, plánování dodavatelského řetězce (včetně plánování zásob a údržby podnikových aktiv a výrobních linek), logistiku (včetně dopravy a správy vozového parku) a řízení objednávek. Zahrnuje činnosti obsahující globální obchod jako jeho dodavatele z celého světa a jejich výrobních procesů. Jeho existence se objevila už v dávných časech s prvním výrobkem nebo službou, která pak byla dále prodána. S příchodem industrializace se stal více sofistikovaný, a tím umožnil efektivnější procesování. Jako jeden z příkladů můžeme zmínit standardizaci výrobků a procesů, jak to udělal Henry Ford, což vedlo k masové výrobě, a tím uspokojení rostoucího počtu zákazníků. [22] Základem SCM je pět elementů a to plánování, což souvisí s vytvořením strategie pro celý řetězec. Zajištění dodávek materiálu je druhý. Výroba od prvního kontaktu s linkou až po zabalení je třetí. Správně nastavené dodávání výrobků je čtvrté. A vrácení jako reklamace defektů nebo vrácení nechtěného nebo nadbytečného zboží je pátá. SCM systém může být software pro sledování, kontrolu nebo řízení dodávek pro všechny zúčastněné strany jako jsou dodavatelé, výrobci, velkoobchodníci, dopravci a maloobchodníci. [2][22]

Manufacturing Execution System (MES)

Do češtiny se překládá jako: „Výrobní informační systémy". Jsou to převážně počítačové systémy pro podporu výroby, její sledování a případně reportování problémů nebo nesrovnalostí, což vede k vyšší efektivnosti výroby. Jedním ze

základů je, že pracují v reálném čase, jsou navázané na Enterprise Resource Planning (ERP, plánování podnikových zdrojů) a veškeré nástroje na automatizaci výroby. Není účelem této práce rozebrat všechny součásti systému, ale pro představu: Všechny potřebné informace jsou na jednom místě, jako naskenuj tento kód a přidej tuto součástku? Kolik materiálu je potřeba a jestli je jeho množství dostatečné pro celou zakázku. Mám přesná a správná data? Je proces efektivní? Je posloupnost akcí správná? Jsou odchylky proti standardu v normě? Všechna data o výrobě jsou na jednom místě a dá se dohledat, jaký materiál přišel do jakého výrobku? Má operátor znalosti pro výrobu a bude jich na směnu dost? Mají všichni přístup k potřebným informacím a jednotné místo nebo proces, kam mohou zadat problém nebo nápad pro vylepšení? [3][4]

Business inteligence (BI)

Nesouvisí přímo s výrobním procesem, vývojem nebo testováním softwaru, ale jejich přínos a vliv není určitě minoritní. Business inteligence jsou technologie, postupy, strategie a znalosti, které podniky používají k data miningu a správě dat pro rozhodování, pochopení trhu a dalších souvislostí. Dokáže zpracovávat velké objemy strukturovaných a nezřídka nestrukturovaných informací jako jsou elektronická pošta, obrázky, videa, dokumenty, zdravotní záznamy a další zdroje. Dá se rozdělit do několika kategorií od sběru, jejich integraci například v datových skladech, analýzu až k prezentaci. Nedílnou součástí je zapojení umělé inteligence jako prediktivní analytika a strojové učení. Ve firmách se používá pro nastavení správných výkonnostních ukazatelů, efektivní vytváření reportů, tak aby se při rozhodování používali aktuální a správné informace. Jedná se o jeden z nejvíce rostoucích oborů informačních technologií v posledních letech. [13][14][15]

Unit testing

Testování jednotky je white box statická technika testování a jedná se o první úroveň testů. Je prováděna bez spuštěného (sestaveného) programu. To znamená, že jsou prováděny jen nad zdrojovým kódem. Můžeme zařadit i statickou analýzu kódu nebo různé vulnerability testy pro kontrolu, jestli třeba není použitá nebezpečná open source knihovna. Bugy nalezené v této fázi vývoje jsou

nejlevnější a nejrychlejší na opravu, hlavně v porovnání s bugy nalezenými po vydání do produkce. Zde je největší přidaná hodnota automatizace a základ pro ostatní automatické testy. Jestli chceme začít s automatizací, tak automatické unit testy jsou základ a jsou zastoupeny až ze sedmdesáti procent všech testů. Když jsou implementovány, tak si je vývojář spustí pokaždé, když si chce ověřit, jestli jeho kus kódu pracuje správně. Analogicky si může spustit sadu testů kolegy nad celým kódem a zjistit, jestli jeho nově přidané řádky nezpůsobily nefunkčnost jiné části programu. Dnešní nástroje umožňují, že při každém požadavku na build se nejdříve automaticky spustí unit testy a vlastně všechny další testy statické analýzy, aby ověřily, že je všechno v pořádku před tím, než se investují zdroje na build aplikace. Když tento test selže, znamená to, že s novým kódem byla do aplikace včetně nové funkcionality zanesena i chyba. Jeden z přístupů pro vývoj softwaru doporučuje nejdříve napsat unit testy podle požadavků zákazníka, a pak teprve psát samotnou aplikaci, dokud nejsou všechny testy zelené – test driven development. Unit testy si píše vývojář sám nad svým kódem, ale praxe doporučuje nechat napsat testy kolegu a nechat tak další pár očí podívat se na novou funkcionalitu a zkontrolovat kód. Některé firmy si najímají externí společnosti na vytvoření unit testů s cílem najít místo, kde je třeba neošetřená výjimka, a tím pak předcházet nákladným opravám bugů v produkci. [1][6]

Component testing

Testování komponenty je zjednodušeně nadstavba nad unit testy. Je to testování izolované části systému na reálných datech bez integrace s ostatními komponenty. Jedná se o třídy, moduly, objekty a programy. Cílem je zkontrolovat různé aplikace a zabránit proniknutí bugů do vyšších úrovní testování. Bývá prováděno většinou vývojářem (developerem). Oprava nalezených bugů je v této fázi stále relativně levná. [6]

Integration testing

Testuje integraci mezi jednotlivými částmi systému. Zaměřuje se na správnou komunikaci mezi jednotlivými na sebe závislými součástmi. Používá se hlavně na ověření, že se data z jedné komponenty dostanou do druhé v nezměněné formě,

případně v požadované změněné formě. Nezaměřuje se na správnou funkcionalitu jednotlivých komponent. Jedná se o druhou úroveň testů, která přichází po unit testech, a pokud byly úspěšné funkční testy na všech komponentách. Při automatizaci jsou většinou zastoupeny z dvaceti procent a jedná se hlavně o volání různých API mezi sebou jako je výměna informací – vytvořit, editovat a smazat nebo jen zkouška, jestli jsou rozhraní mezi sebou dostupná. U těchto testů není potřeba mít sestavený front-end nebo jiné grafické uživatelské rozhraní. Testuje se zde hlavně funkčnost, shoda s předpisy a požadavky a bezpečnost komunikace. V případě webových služeb se testuje, jestli je požadavek (request) a odpověď (response) správně šifrovaná. [5][11][10]

Systémové (End-to-end) testy

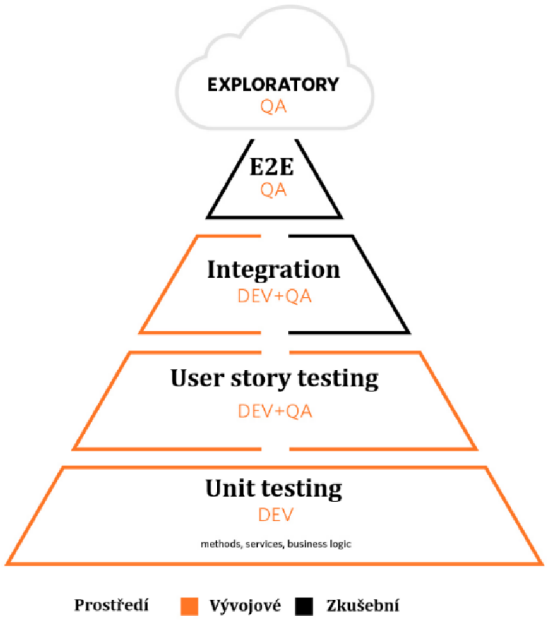
Systémové testování je třetí úroveň testů. Jedná se o testovací techniku, která testuje aplikaci od začátku až do konce, zda se chová tak, jak bylo požadováno, proto se jim také občas říká end-to-end (E2E) testy. Definují systémové závislosti produktu a zajišťují, aby všechny integrované části fungovaly společně. Jedná se o nejnáročnější sadu testů, a proto i nejmíň zastoupenou v automatizaci, přibližně asi deset procent, protože ke svému běhu potřebují mít sestavené grafické uživatelské rozhraní (GUI). Tyto testy jsou náročné na údržbu, protože se uživatelské prostředí velice často mění. Na druhou stranu se testuje přesně, co bude dodáno nebo jak se bude aplikace používat. Automatický test GUI bude napodobovat uživatelské chování, bude simulovat používání klávesnice nebo myši ke kliknutí nebo zadání vstupu do aplikace. Díky tomu se dají najít chyby co nejdříve, není potřeba manuálně vyplňovat rozsáhlé formuláře, které berou spoustu času, ale rychle, když je investován čas na napsání těchto testů nejlépe v počáteční fázi vývoje a s každým regresním testem si ověřit funkčnost. Základní rozdělení je na nástroje, kde se dá přesně nahrát testerovo chování jako jakýkoli pohyb myší nebo vstup klávesnice a na nástroje co umožní, aby prvky rozhraní byly viditelné pro nějaký nástroj jako třeba Selenium, a pak napsat test v požadovaném programovacím jazyku. [1][5][6][24]

User acceptance testing (UAT)

Jedná se o systémové testování na reálném produkčním prostředí nebo jeho simulaci. Testují reální uživatelé a potvrzují, jestli aplikace naplňuje jejich požadavky. [1][6][24]

3.4 Testovací pyramida

Pyramida testování je koncept, který rozděluje, většinou automatické softwarové testy do různých kategorií. Pro potřeby této práce budou tyto kategorie čtyři plus jedna. To pomáhá vývojářům a testerům zajistit vyšší kvalitu, zkrátit čas potřebný k nalezení příčiny chyb a vytvořit spolehlivější sadu testů.



Obrázek 2 - Testovací pyramida
Zdroj: vlastní zpracování

Dole jako základ pyramidy jsou unit testy tvořené vývojáři a podílející se na celkovém počtu většinou kolem sedmdesáti procent. Jsou první, nejlevnější, nejrychlejší a dokážou odhalit většinu chyb, protože se soustředí vždy jen na jednu část v kódu. Nad nimi jsou testy požadavků (jednotlivých user story), které se podílejí na celkovém počtu většinou dvaceti procenty a jsou prováděny převážně testery, ale v odůvodněných případech i vývojáři. Na řadě jsou vždycky až po tom, co jsou úspěšně provedeny unit testy a testují hlavně jednu funkcionalitu nebo požadavek. Jako třetí se na řadu dostávají integrační testy se svými přibližně deseti

procenty prováděné testery. Tyto testy jsou náročnější a tím i dražší, hlavně z důvodu, že potřebují mít připravené celé prostředí, na kterém se bude testovat. Výhodou je, že nepotřebují uživatelské prostředí jako testy následující, a to systémové testy. Tyto testy se podílejí na celkovém počtu kolem deseti procent, jsou prováděné převážně testery před a po nasazení na zkušební prostředí. Na automatizaci jsou náročné, tím i drahé a trvají dlouho, k jejich běhu je potřeba mít uživatelské rozhraní a ideálně simulovat, jak se chová i uživatel, proto nejdou nějaké kroky urychlit, případně provádět paralelně.[1][12][24][25]

Waterfall model (Vodopádový model)

Vodopád je technika založená na sériovém průběhu vývoje, kdy se nezačne další fáze vývoje, dokud není dokončena předchozí. Dokud nejsou sepsány požadavky, tak se nezačne s vývojem a analogicky se nezačne s testováním, dokud není aplikace naprogramovaná. Tento model z pohledu testování softwaru fungoval tak, že existovaly samostatné testovací týmy, kde byly striktně rozdělené role. Test manažer psal testovací scénáře, které si přebírali testeři a bod po bodu procházeli nebo automatizační inženýři podle zadání automatizovali. Touto metodikou ztráceli test manažeři kontakt s vlastním testováním nebo automatizační inženýři ztráceli zkušenosti s manuálním testováním a vlastní tester byl degradován na stroj na provádění testů. Neexistovala také přímá komunikace mezi vývojáři a testery. Software se předal k testování jako celek a vracel se otestovaný s test reportem a zadanými bugy. Toto se opakovalo, dokud software nesplňoval nastavené požadavky na kvalitu, nebo vedení neschválilo jeho nasazení na produkci. Jak se může tento model zdát v dnešní době trošku nepraktický, tak má stále opodstatnění. Třeba u velkých bankovních systémů, kde je přesně dané zadání a požadované vstupy a výstupy a není možnost rozdělit na logické celky nebo třeba u vývoje jádra antiviru. [24][32]

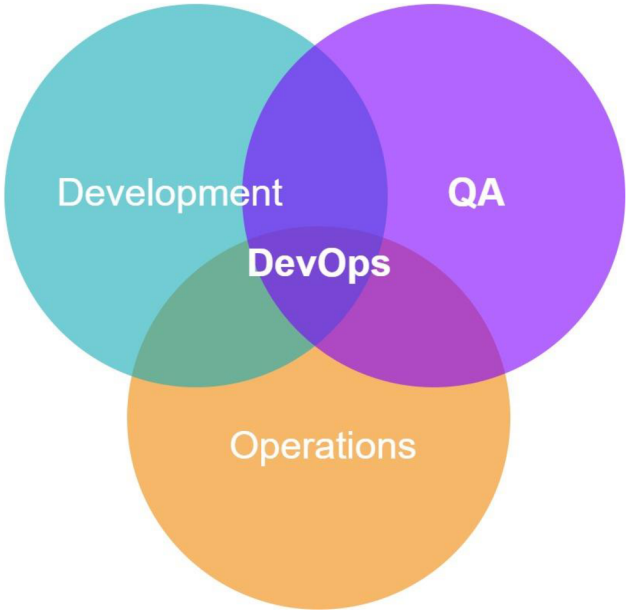
Agile (Scrum) model

Tento model se v poslední dekádě stal jedničkou v přístupu, jak vyvíjet software, a to hlavně kvůli jeho rychlosti. Tedy od prvních požadavků až k finálnímu dodání je potřebný čas minimalizován. Jeho největší výhodou je, že na rozdíl od

vodopádového modelu se k němu přistupuje paralelně. Když jsou známé první požadavky, může se začít s vývojem a to samé jakmile je hotový první kus kódu, tak může začít tester vykonávat svoji práci. V tomto modelu se také vytrácí striktní rozdělení týmů a rolí. V ideálním scrumu by se měl stírat i rozdíl mezi jednotlivými členy týmu a každý by měl být schopen dělat, co je aktuálně třeba. Největší výhodou je, že se vývoj rozděluje do jednotlivých iterací zvaných sprinty, které můžou mít různou délku, většinou čtrnáct dní. Tester si sám píše testovací scénáře, které pak i vykonává a píše automatické regresní testy. Každý sprint by měl končit dodávkou nové funkcionality pro uživatele, tím je snadno sledovatelné, co se podařilo vyvinout a zákazník si může na reálné aplikaci otestovat, jestli je to přesně podle jeho požadavků. V tomto modelu je nezbytná role scrum mastera, který plánuje práci na celý sprint, rozděluje aplikace do logických celků, které se dají dodat na konci každé iterace. Je tedy hlavní osoba v komunikaci s ostatními mimo tým, tak aby se vývojáři a testeři mohli naplno věnovat svojí práci. [5][24][25]

3.5 DevOps

Název DevOps vznikl ze spojení anglických slov development (vývoje softwaru) a operations (IT provozu). Jedná se o reakci na vzájemné se prolínání těchto dvou oborů. DevOps znamená propojení těchto dvou disciplín z pohledu filozofie, postupů a nástrojů. Cílem je vyvíjet a zlepšovat produkty rychleji než firmy využívající tradiční procesy vývoje a správy infrastruktury. I ten nejjednodušší proces je v DevOps popsán, a to hlavně z důvodu předvídatelnosti, produktivity, bezpečnosti a udržitelnosti údržby firemních systémů. Přesně vím, co mě čeká nebo následuje.



Obrázek 3 - DevOps
Zdroj: vlastní zpracování

V tomto modelu nejsou týmy vývoje, testování ani provozu rozděleny, ale u dokonalého DevOps se tyto pozice stírají a všichni by měli zvládnout teoreticky vše. Takovýto tým využívá co nejvíce vlastních zkušeností pro zjednodušení a automatizování procesů, které byly dříve manuální. [24][25]

Výhody DevOps

Ve srovnání s klasickým vývojem nebo i ve srovnání s agilním přístupem má DevOps nesporné výhody jako:

- **Rychlost**
  - Spojení microservices a continuous integration pomáhají v rychle se měnicím se prostředí být efektivní v plnění firemních požadavků.
- **Rychlé dodání**
  - Čím je rychlejší vydávání verzí od požadavku až po finální nasazení, tím lépe se reaguje na požadavky zákazníků a na případné opravy chyb v systému.

- **Spolehlivost**
    - Continuous integration a jejich dalších nadstaveb se každá změna nebo úprava otestuje před i po nasazení na produkční prostředí a tím se ujistí, že je změna funkční a bezpečná.
- **Škálování**
    - Při adopci microservices a DevOps filozofie se složité a rychle měnící systémy spravují efektivně s minimalizovaným rizikem selhání.
- **Vylepšená spolupráce**
    - Týmy podle metodiky jsou více zaměřené na odpovědnost a užší spolupráci mezi lidmi zapojenými do vývoje a údržby softwaru, sdílejí poznatky a předávají si pracovní postupy. Doba potřebná k předání od vývojářů k údržbě je minimalizována.
- **Bezpečnost**
    - Zrychlení vývoje není na úkor kvality. Nástroje pro bezpečnost se dají zautomatizovat nebo implementovat do nastavení CI pipelines. Jako příklad se dá uvést vulnerability skenování.

Continuous integration (CI)

Kontinuální integrace je vývojovou praxí, kde se každá část softwaru ukládá a verzuje v společném úložiště GIT (více o verzování [33]) a hned se integruje s kódem ostatních vývojářů. Předtím než se vývojář pustí do práce, musí vytvořit svojí vlastní kopii (větev) kódu upravovaného softwaru a na ní pracuje. Jak ostatní vývojáři odesílají svůj kód z vlastních větví do původní (hlavní) větve, tak kopie u prvního přestává odrážet všechny změny, co byly provedeny. S každou změnou kódu nebo přidání nové knihovny, které vytvářejí potencionální konflikty při pokusu odeslat vlastní kód do hlavní větve. [25]

Když se na větvi dlouho vyvíjí bez jejího sloučení do hlavní větve, tím více se tíhne k problémům způsobeným integračními konflikty, když je pak změna nahrávána do hlavní větve. Proto se před každým pokusem integrace vlastní a hlavní větve musí nejdříve vlastní větev zaktualizovat, tak aby odrážela všechny změny v hlavní větvi od vytvoření vlastní. Z toho důvodu se doporučuje při delším vývoji často

aktualizovat svojí vlastní větev podle změn v hlavní, jinak může dojít k takzvanému integračnímu peklu, kdy čas strávený integrací je větší, než čas investovaný do vývoje nové časti kódu. Je to rozdíl od vývoje, který se bohužel ještě dnes objevuje, a to manuální sbírání kódu jedním vývojářem od ostatních vývojářů a pak vlastní sestavení hlavní větve. V kontinuální integraci jsou tyto kroky zautomatizované nebo jsou na ně nastaveny jasné procesy.

Continuous Delivery (CD)

Kontinuální doručování je přístup ve vývoji softwaru, kde se na rozdíl od kontinuální integrace (CI) přidává k integraci kódu ještě jeho automatické otestování a sestavení verze. Vývoj pak jede v cyklech a umožňuje, že může být aplikace kdykoli a rychle nasazena na produkci. Tento přístup snižuje dobu potřebnou k dodávce nového kusu kódu do produkčního prostředí. To snižuje náklady, čas a riziko při vydávání jedné velké verze najednou. Místo toho je rozdělena do několika aktualizací.

Continuous Deployment (CD)

Kontinuální nasazení je stejné jako předchozí, kde se software vyvíjí v krátkých cyklech, ale s tím rozdílem, že se finální nasazení nedělá manuálně, ale pomocí automatizovaných nástrojů. To znamená, že od poslání vývojářova kódu do společného úložiště je veškerá integrace, testy a nasazení plně automatické bez zásahu člověka.

GITlab

GitLab je otevřená DevOps platforma určená pro vytvoření aplikace od začátku po nasazení. V jednom softwaru se dá spravovat celý vývojový cyklus umožňující vývoj, testování, správu bezpečnosti a možnost pracovat několika týmům simultánně na jednom projektu. Přes jedno uživatelské prostředí, přes jeden přístup. Jedná se o open source platformu, na které pracuje více než 1300 vývojářů v 68 zemích.

CI Pipelines

Jedná se o sérii kroků, které se musí provést před vydáním verze softwaru na produkční prostředí jako například integrace kódu, sestavení, testy a nasazení. Každý krok se dá provést manuálně, ale na síle dostávají s automatizací. Některé kroky můžou jet paralelně, třeba testy. [25][32]

Microservices

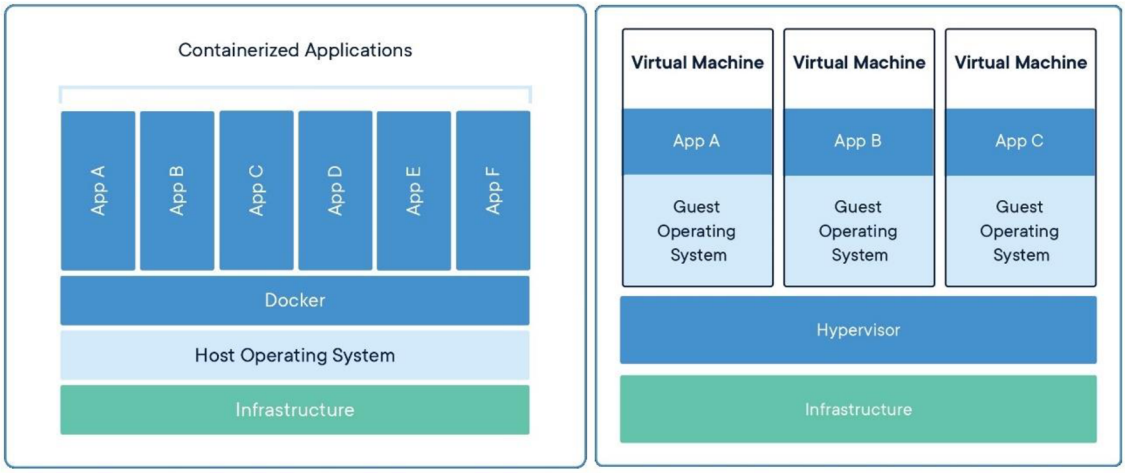
Mikroslužby jsou softwarová architektura při vývoji softwaru. Aplikace je soubor volně provázaných služeb, které mezi sebou komunikují podle jasně daného API. Jejich architektura umožňuje velmi dobrou škálovatelnost a zrychluje uvádění nových verzí do produkce. Oproti monolitické architektuře, kde je celá aplikace a v ní všechny procesy pevně spjaty a navenek se hlásí jako jedna služba. Takže pokud vznikne požadavek na růst jednoho procesu, musí se škálovat celá aplikace. Přidávání a též jakákoli vylepšení v monolitické aplikaci znamená nárůst její kódové základny. Tím její složitost omezuje různé experimentování a implementaci nových nápadů. Riziko nedostupnosti nebo selhání roste s každým novým řádkem kódu, protože i jeden chybný proces může vést k nedostupnosti celé aplikace. Jakékoli změny v programovacím jazyku nebo frameworku jsou velice složité, protože se musí změnit v celém monolitu, a nejen kde je potřeba. Oproti tomu v architektuře microservices je aplikace postavena na nezávislých komponentách, které spouštějí každý jednotlivý proces jako jednu službu. Tyto služby spolu komunikují přes jasně definované odlehčené API. Každá služba vykonává jednu funkci, v případě složitějších požadavků je zapojeno několik služeb. Jelikož jsou provozovány nezávisle, lze každou jednotlivou službu aktualizovat, nasadit, škálovat nebo vytvořit duplicitní služby, aby zvládla vyšší zátěž bez nutnosti měnit ostatní služby. Celé prostředí je pak nezávislé na tom, v jakém programovacím jazyce je napsáno a jaké nástroje, knihovny a tak podobně používají, pokud komunikují přes jednotné API. Pro představu, když chci nasadit změnu na jedné službě, tak změním a nasadím jenom ji a původní službu vypnu. V případě, že by neprošly testy s novou službou, přepojím celé prostředí na starou nebo vrátím (revert) změnu v nasazení. Samotné téma microservices je velice

rozsáhlé a není hlavním tématem této práce. Pro další informace doporučuji [19],[20],[21] a [31].

Virtualizace a kontejnerizace

Virtualizace je proces, při kterém vznikne další vrstva nad počítačovým hardwarem, která umožňuje rozdělit hardwarové prvky počítače do několika virtuálních strojů. Použitý software se nazývá hypervisor a umožňuje, aby vedle sebe běželo několik, klidně různých operačních systémů, které sdílejí stejný hardware.

Kontejnerizace je podobná virtualizaci, ale narozdíl od ní nevytváří virtuální počítač včetně včetně operačního systému, ale jen software nad operačním systémem, tím pádem nepotřebuje ani hypervisory. Kontejnery využívají určitou virtualizaci operačního systému, zjednodušeně řečeno využívají zdroje operačního systému hostitelské infrastruktury pro izolaci procesů a řízení jejich přístupu k procesoru, paměti a místu na diskovém prostoru. Pro představu, kontejnery mají velikost v megabajtech a spuštění je v sekundách oproti virtuálním strojům, kde je velikost v gigabajtech a můžou se spouštět až několik minut. To znamená, že na klasický server se vejde dvakrát, třikrát více aplikací v kontejnerech, než by tomu bylo s použitím virtualizace. [26][31]



Obrázek 4 - Virtualizace a kontejnerizace
Zdroj: Docker Inc. [26]

Nevýhodou je, že spuštěný kontejner je na stejném jádře jako jeho hostitelský počítač, takže není možné používat různé operační systémy (nelze spustit image Microsoft Windows na operačním systému Linux).

Docker

Docker se do češtiny překládá jako přístavní dělník a analogie k tomu k čemu se používá úplně vybízí. Jedná se o nástroj nebo spíš software pro správu kontejnerů (vytvoření, nasazení a provoz). Umožňuje vývojáři zabalit aplikace s tím, co je nezbytné pro její provoz jako knihovnami a závislostmi do jednoho image, tím zjednodušuje provoz aplikací na serveru. Základem každého docker image je docker file (soubor pro sestavení kontejneru).

Kubernetes

Zkráceně „K8S“ je clusterová technologie, která umožňuje spojit dohromady několik oddělených počítačů. Tyto spojené počítače se poté hlásí jako jeden počítač (cluster). Používá se hlavně na správu dockerových image a někdy se nazývá orchestrátor. [31]

- **Cluster**
  - Kubernetes cluster je virtuální propojení několika počítačů dohromady, které se na venek jeví jako jeden počítač.
- **Nodes**
  - Kubernetes nodes jsou jednotlivé počítače, ze kterých se cluster skládá. Není problém, že různé nodes nejsou v jednom datovém centru. Kubernetes se postará o jejich spojení a aby navenek působili jako jeden cluster a komunikovala spolu.
- **Pods**
  - Jedná se o instanci aplikace, co chceme provozovat.
  - Pod je nejmenší jednotka, co se dá nasadit (deployment). Jeden pod může obsahovat několik kontejnerů, ale běžně se používá, co jeden pod to jeden kontejner.

- **Container**
    - Kontejner je balíček, který obsahuje OS, potřebné knihovny a aplikaci, kterou chcete provozovat. Zjednodušeně lze říct, že kontejner je aplikace.

Vizualizace výsledků testů

Automatické testy ztrácí část své přidané hodnoty, když jejich výsledky nejsou jednoduše dostupné na jednom místě pro celý systém a jednoduše čitelné. Výsledky testů na zkušebním prostředí, a případně na produkčním prostředí, by měly být přístupné i business product ownerům a uživatelům, aby byli informování, že je aplikace rozbitá. Všechny výsledky testů, které selhaly, musí být též reportovány na pověřenou osobu nebo eskalovány, aby se co nejdříve začalo s analýzou. V případě business kritické funkcionality se musí na investigaci či opravě začít okamžitě.

Logování

Jak pro vývojáře, tak i testera je důležité, aby aplikace, případně automatický test logoval svoji aktivitu, aby se zpětně dalo najít kde a kdy přesně se stala chyba, a podle toho se dala najít a opravit chyba v aplikaci. Zkušenosti velí mít dva druhy logování, jedno vždy při běžící aplikaci do dočasné složky, kde se logují nejdůležitější akce, vstupy a procesy a pak druhé vývojové logování, velice podrobné, kde se dá podívat postupně na všechny, i ty nejmenší detaily. Logy je taky vhodné mít pro všechny systémy, a to i pro testy v jednotném formátu, aby se daly všechny otevírat v jedné aplikaci.

4 Testovací strategie

Navrhuji obecnou strategii pro firmu na základě analýzy a studia literatury, která bude platit pro všechny systémy jako MES, SCM, BI, integrační aplikace a podobně. Přístup k jejich testování bude rozpracován v jednotlivých testovacích plánech. Požadavky od managementu společnosti:

- Co nejnižší počet pracovníků

- Rychlé dodávání na produkci
- Kompatibilita s microservices
- Důraz na moderní technologie
- Kompatibilita s Průmysl 4.0
- Zaměření na uživatele

Ukázala jako nejvhodnější a všechny podmínky plnící strategii testování pomocí agilní metodiky s postupným přechodem na DevOps. Tím i postupné školení a vzdělávání z testovacích inženýrů na DevOps inženýry zaměřené na automatické testování a mentorující ostatní členy týmu ohledně správné techniky ověření kvality tak, aby testování mohlo být praktikováno kýmkoli z týmu. Samozřejmostí je kulturní změna spočívající v odpoutání se od vodopádové metodiky a projektového vývoje na produktové řízení, kdy je produkt, a hlavně uživatel středem všeho dění.

Obrázek 5 - Rozpad testovací strategie
Zdroj: vlastní zpracování

Záměr

Cílem je informovat všechny zúčastněné strany podílející se na testování, jeho přístupech, jednotlivých činností, včetně vztahů a závislostí z toho plynoucích systémech vyvinutých ve společnosti včetně aplikací dodaných od třetích stran a implementovány do firemní infrastruktury microservices.
Strategie testování softwaru popisuje všechny činnosti zapojené do ověřování kvality a nastavuje konzistentní přístup, jak se bude provádět ve společnosti.

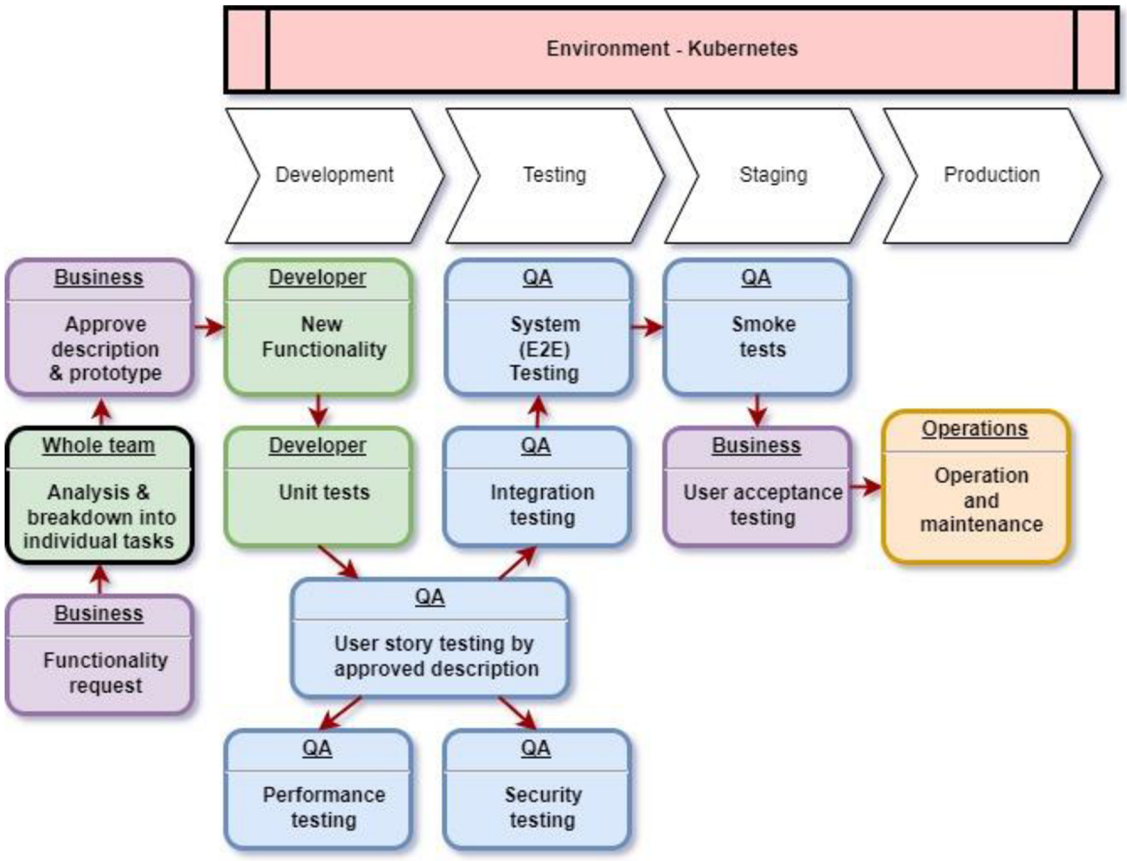
Cíle

Tento dokument definuje ověřování kvality softwaru ve společnosti a přístup k jednotlivým fázím průběhu testování, které jsou nutné ke kontinuálním dodávkám a akceptování dodaného řešení. Pokrývá všechny fáze a vydání verzí.

Rozsah

Strategie pokryje následující:

- Identifikuje jednotlivé druhy testování, které jsou potřeba provést v průběhu životního cyklu softwaru od návrhu až po provoz na produkčním prostředí.
- Detaily o probíhajícím testování nových služeb, jejich vylepšení a změnách.
- Oblasti jednotlivých testů.
- Stanoví způsob, jak mají být společná očekávání a standardy testování dosaženy pro všechny typy testování.
- Požadované technické znalosti, zdroje a prostředí nutné k implementaci.
- Nezbytné procedury testování a ověření kvality



Obrázek 6 - Průchod user story skrz prostředí a jednotlivé testy
Zdroj: vlastní zpracování

Životní cyklus je založen na široce rozšířeném agilním vývoji a tím i agilním testování s cílem implementovat DevOps. Strategie stanoví základní výstupy jak z vývoje, tak z testování a údržby, které by měly být v jednotlivých fázích testovány nebo případně monitorovány. Bude navržen v souladu s ostatními procesy společnosti. Oddělení Software delivery (SWD) bude zodpovědné za definovaní a provádění všech etap testování v průběhu celého životního cyklu v úzké součinnosti s product owners a uživateli jednotlivých aplikací.

4.1 Přístup k testování

K testování se bude přistupovat podle nejnovějších poznatků a standardů. Co nejvíce minimalizovat náklady na samotné testování pomocí automatizace a postupného přechodu do filozofie DevOps.

Jednoznačnost

Každý požadavek musí být jasně a přesně formulován. Měl by umožňovat jedinečný výklad. Požadavek musí být čitelný a srozumitelný. Pokud je požadavek obzvláště složitý, pak pro usnadnění pochopení může být doplněn přiloženým materiálem, jako jsou diagramy, obrázky nebo tabulky. Pokud jsou v user story použity přesvědčivé výrazy jako "je to zřejmé" nebo "samozřejmé", je dost možné, že se autor pokouší odvést vaši pozornost od toho či onoho nejednoznačného tvrzení.

Konzistence

Požadavky by neměly být v rozporu mezi sebou ani s platnými požadavky nebo normami. Pokud si požadavky vzájemně odporují, musí se následně podle nastavené priority konflikt vyřešit na nejbližším meetingu. Schopnost odhalení chyb vzniklých rozporem v požadavcích předpokládá dobrou znalost dokumentace a uživatelských požadavků, stávajících norem a celého produktu.

Dohledatelnost

Každé tvrzení musí mít jedinečný identifikátor, který umožňuje sledovat jeho průběh v celém životním cyklu. Ve výstupech testování, které se objevují v pozdějších fázích životního cyklu, jako je plán testování, musí být každý odkaz na požadavek nebo specifikaci dohledatelný.

Praktičnost

Každý požadavek by měl zahrnovat systémové zadání, poskytnout takové prostředky, které je ekonomicky vhodné vyvinout a udržovat. Pokud zákazník vznese nereálné požadavky z hlediska vynaloženého času a finančních prostředků na vývoj nebo požaduje vývoj funkcí, které budou nespolehlivé a nebezpečné pro používání, je třeba definovat rizika a přijmout příslušná opatření. Vyvíjený systém totiž musí být ekonomicky proveditelný, spolehlivý a snadno použitelný.

Testovatelnost

Pro každý požadavek bychom měli být schopni vyvinout ekonomicky životaschopné (nenáročná údržba a výkonové požadavky) a snadno použitelné testy, které prokážou, že testovaný softwarový produkt má potřebnou funkčnost, výkon a odpovídá současným standardům. To znamená, že každý požadavek musí být analyzován a jeho testování by mělo být provedeno za přijatelných podmínek poměru ceny a přidané hodnoty.

Testy budou rozděleny do následujících kategorii:

- Unit Testing
- User story testing
    - Testování podle specifikace/požadavků
    - Exploratory testing
- Integrační testování
    - Regresní integrační testy celého produktu
- Systémové testy
    - Testy uživatelského rozhraní
    - Exploratory testing
    - Regresní testy celého produktu
    - Bezpečnostní testování (Podle potřeby)
    - Výkonové testování (Podle potřeby)
- Uživatelské testování (UAT)
    - Smoke testy

Proces testování

Každému testování musí předcházet vize a cíl testování. Každá nová iterace procesu začíná s přidáním nové funkcionality určené k testování.

Požadavky testování

Hlavní požadavky testování jsou, že je aplikace bez přítomnosti chyb, dodána včas a za předem schválený rozpočet. Minimální požadavky jsou unit testy a smoke

testy, které potvrdí, že se aplikace nasadila, plní hlavní požadavek a je dostupná k používání. Plné požadavky znamenají, že je aplikace otestovaná celou sadou testů od unit testů, integračních, systémových a případně bezpečnostních a výkonových.

Specifikace

Pro proces testování je nezbytné umět identifikovat základní komponenty, komunikaci, systém a middleware. Pro testovací prostředí bude používán cluster v prostředí Kubernetes. V případě potřeby testovacího týmu se dá přidat kontejner s webovým prohlížečem pro provádění testů uživatelského prostředí nebo další kontejnerů pro výkonové testy.

Přístup k funkčnímu testování

Funkční testování je hlavní cíl testování, každá aplikace musí fungovat podle požadavků, standardů a specifikací. V případě, když je nalezeno chování, které neodpovídá požadavkům, je nežádoucí nebo je „podezřelé“, musí být vytvořen požadavek na opravu vývojovým týmem a ten být opatřen prioritou pro další vývoj, případně rovnou naplánován na opravu. Nefunkcionální testování je doplňkový cíl, který by ale neměl být upozaďován.

Manuální nebo automatické testování

Cílem testovacího týmu je mít všechny základní a pro systém kritické funkcionality pokryté automatickými testy. To neznamená, že se manuální testování upozadí. Vždycky bude každý požadavek testován prvně manuálně, kde se zjistí nepřítomnost chyb a jestli plní požadavky. Součástí tohoto testování je i analýza návratností investice do automatizace a vznik zadání pro vytvoření automatického testu.

4.2 Úrovně a typy testů

Všechny procesy a pravidla zde nastavená musí být dodržována, případnou výjimku může udělit jen testovací autorita (test lead, QA manager) případně architekt.

Unit testy

První a základní testy dodávané vývojářem s novým kódem. Není cílem vyvinout co nejvíce testů, ale testy, které opravdu ověřují funkčnost. U funkcionalit, kde je jedinou funkcí brát data z jednoho systému a posílat dále, není nutné investovat čas na vývoj z důvodu nákladné údržby testovacích dat, které se můžou lišit od produkčních.

Tabulka 2 - Unit testy

Zodpovědnost	Vývojář
Cíl	• Testování zdrojového kódu jednotlivých modulů a procedur k nim navázaným
Rozsah	• Testování zdrojového kódu • Reportování a správa chyb • Provádění testovacích případů • Vypracování výsledků testů a testovacích reportů • Vyvinout a případně přepracovat kód na základě výstupů z testů, provést opětovné testování až do jeho dokončení
Prostředí	Vývojové (Dev)
Testovací data	Nasimulovaná testovací data podle produkčních
Testovací cykly	S každým novým zdrojovým kódem
Časový horizont	V průběhu vývoje funkcionality
Formální dokumentace testu (ANO/NE)	Ano
Zaznamenávání závad a následná opatření (ANO/NE)	Ano
Hlášení o stavu (ANO/NE)	Ano

Zdroj: vlastní zpracování

- **Vstupní kritéria**
    - Dokončena fáze plánování

- Technická specifikace systému a dokumentace jsou analyzované a schválené
- Zákaznické požadavky jsou definovány a schváleny
- Moduly nebo zdrojový kód je připravený
- Testovací prostředí je dostupné

- **Vstupní artefakty**
  - Schválené požadavky, podmínky a specifikace
- **Výstupní kritéria**
  - Konec termínu nebo vyčerpaný rozpočet
  - Všechny testovací případy pro danou funkci jsou provedeny
  - Požadavky jsou splněny
  - Všechny chyby jsou reportované
  - Všechny kritické chyby jsou odstraněné, akceptují se jen závady nízké úrovně
- **Výstupní artefakty**
  - Unit Test Report
  - Chyby jsou reportované
  - Schválení IT architektem

User story testing

Testování uživatelských požadavků provádí tester podle požadavku zákazníka, zapisuje si postup, který použije jako test report. Ten následně bude sloužit jako zadání pro automatický test a případné chyby zadává do systému.

Tabulka 3 - User story testing

Zodpovědnost	Tester
Cíl	• Testování funkcionality aplikace podle požadavků • Ověřit shodu funkcí podle specifikací • Další nefunkční testy
Rozsah	• Provést akceptační testy s pozitivními a negativními scénáři

	• Management defektů • Provádění testovacích případů • Vypracování výsledků testů a testovacích reportů • Navrhnout vylepšení funkcionality • Kontrola, jestli je dokumentace aktuální • Vyvinout a nasadit automatické testy, případně založit do testovacího backlogu požadavek na vytvoření
Prostředí	Primární – Testovací (Test) Sekundární – Vývojové (Dev)
Testovací data	Testovací databáze (Kopie produkčních, případně externích dat), data co nejvíce odrážející produkční prostředí
Testovací cykly	Jeden sprint (Dva týdny)
Časový horizont	2 týdny
Formální dokumentace testu (ANO/NE)	Ano
Zaznamenávání závad a následná opatření (ANO/NE)	Ano
Hlášení o stavu (ANO/NE)	Ano

Zdroj: vlastní zpracování

- **Vstupní kritéria**
  - Unit testy jsou dokončeny včetně jejich případných oprav
  - Testovací prostředí je připravené a dostupné
  - Testovací účty jsou založené a funkční
  - Testovací data jsou dostupná v testovacím prostředí a nejsou zastaralá
  - Testovací scénáře jsou připravené
  - Všechny zainteresované strany jsou informovány

    - Plán testování je připraven a schválen
    - Jsou zřízeny přístupy do nástroje pro management chyb
    - Všechny potřebné systémy mají přístup do testovacího prostředí
- **Vstupní artefakty**
    - Schválená akceptační kritéria
    - Definice připravenosti (DoR) splněna
- **Výstupní kritéria**
    - Všechny testovací případy pro danou funkci jsou provedeny
    - Všechny kritické chyby jsou odstraněné, akceptují se jen závady nízké úrovně
    - Definice hotového (DoD) naplněna
- **Výstupní artefakty**
    - Sprint test report
    - Chyby jsou reportované
    - Aktualizované uživatelské akceptační testování (UAT)

Integrační testy

Tester provádí testy komunikace mezi jednotlivými rozhraní a systémy s důrazem na automatizaci všech možných kombinací.

Tabulka 4 - Integrační testy

Zodpovědnost	Tester
Cíl	• Otestovat požadované moduly a funkce na komunikaci, bezpečnost s ostatními systémy podle specifikace
Rozsah	• Spuštění automatických testovacích skriptů systému • Management chyb • Testování API (rozhraní) • Navrhnout vylepšení funkcionality • Kontrola, jestli je dokumentace aktuální

	• Vyvinout a nasadit automatické testy, případně založit do testovacího backlogu požadavek na vytvoření • Vypracování výsledků ze systémových testů a testování systémové integrace
Prostředí	Primární – Testovací (Test) Sekundární – Zkušební (Stag)
Testovací data	Testovací databáze (Kopie produkčních, případně externích dat), data co nejvíce odrážející produkční prostředí
Testovací cykly	Jeden sprint (Dva týdny)
Časový horizont	2 týdny
Formální dokumentace testu (ANO/NE)	Ano
Zaznamenávání závad a následná opatření (ANO/NE)	Ano
Hlášení o stavu (ANO/NE)	Ano

Zdroj: vlastní zpracování

- **Vstupní kritéria**
  - Unit testy jsou dokončeny včetně jejich případných oprav
  - Testovací prostředí je připravené a konektivita je otestovaná
  - Testovací sady jsou připravené
  - Jsou zřízeny přístupy do nástroje pro management chyb
  - Testovací účty jsou založené a funkční
  - Plán testování je připraven a schválený
  - Testovací data jsou dostupná v testovacím prostředí a nejsou zastaralá
  - Všechny další systémy mají přístup do testovacího prostředí
  - Všichni zainteresované strany jsou informovány

- **Vstupní artefakty**
    - Schválené integrační testovací případy
- **Výstupní kritéria**
    - Všechny integrační testy jsou provedeny
    - Všechny kritické chyby jsou odstraněné, akceptují se jen závady nízké úrovně
- **Výstupní artefakty**
    - Zpráva o integračním testování
    - Chyby jsou reportované
    - Schváleno architektem
    - Aktualizované integrační testy (SIT)

Systémové Testování

Také jinak jako E2E. Testy celého integrovaného systému přichází na řadu po unit testech a integračních testech. Jsou prováděné testerem a automatizované pomocí nástroje pro automatizaci uživatelského rozhraní. V případě požadavku se v této fázi dělají výkonové a bezpečnostní testy.

Tabulka 5 - Systémové testy

Zodpovědnost	Tester
Cíl	• Systémovým testováním testeři ověřují, že celá aplikace, a tím všechny součásti, fungují správně
Rozsah	• Testeři připraví systémové testy • Ověřit, že systém splňuje požadavky uživatelů • Podpora vývojářů a uživatelů pří řešení/zadávání chyb • Management chyb • Nefunkční testy
Prostředí	• Primární – Testovací (Test)

	• Sekundární – Zkušební (Stag)
Testovací data	• Testovací databáze (Kopie produkčních, případně externích dat), data co nejvíce odrážející produkční prostředí • Reálná data s pozitivními scénáři
Testovací cykly	Jeden sprint (Dva týdny)
Časový horizont	2 týdny
Formální dokumentace testu (ANO/NE)	Ano
Zaznamenávání závad a následná opatření (ANO/NE)	Ano
Hlášení o stavu (ANO/NE)	Ano

Zdroj: vlastní zpracování

- **Vstupní kritéria**
  - Unit testy jsou dokončeny včetně jejich případných oprav
  - Prostředí a testy jsou připraveny pro provedení systémového testování
  - Každý modul prošel unit testy a integračním testem
- **Vstupní artefakty**
  - Schválené systémové testy
- **Výstupní kritéria**
  - Integrační testy jsou úspěšně provedeny
  - Všechny funkce systému jsou ověřeny
  - Chyby s nejvyšší prioritou jsou opraveny
- **Výstupní artefakty**
  - Report o systémovém testování
  - Chyby jsou reportované
  - Schváleno architektem

Uživatelské akceptační testy

Testy jsou určené pro ověření funkcionality uživateli aplikace před nasazením na produkční prostředí. Jediné automatické testy budou smoke testy pro ověření správného nasazení. Může nastat i výjimečný případ, kdy se nejdříve nasadí a pak až otestuje s přímou podporou testera nebo vývojáře, který může případně změny vrátit zpět do původního stavu. Nastává po ukončení sprintu, kde se na live demo předvedou nové funkcionality, a business dostane jeden sprint na otestování a schválení nasazení na produkční, respektive zkušební prostředí.

- **Alfa testování**

Provádí převážně testovací tým, ale v některých specifických případech může být umožněn přístup i do této fáze testování, a to třeba jedná-li se o specifickou funkcionalitu, kde není žádoucí nebo by sebralo zbytečně moc zdrojů na sepsání požadavků na její funkčnost/otestování. Snaha je minimalizace rizika, které by představovalo testování na produkčním prostředí.

- **Beta testování**

Testují uživatelé na zkušebním (Stag) prostředí a výstupy z testování předávají na business product ownera nebo kanály na to určené – na testovací tým. Výstupy z těchto testů jsou procházeny jednou týdně na schůzce mezi testovacím týmem a zástupci businessu včetně business product ownera.

Tabulka 6 - Uživatelské akceptační testy

Zodpovědnost	Uživatel, Tester, IT podpora
Cíl	• Uživatelé nebo product owneři otestují aplikaci před ostrým nasazením na produkční prostředí
Rozsah	• Připravit seznam co bylo vyvinuto pro uživatele v rámci sprintu a je nutné

	otestovat • Sběr a analýza chyb nebo podmětů od uživatelů • Přímá technická podpora • Management chyb
Prostředí	• Primární – Zkušební (Stag) • Sekundární – Produkční (Prod)
Testovací data	• Testovací databáze (Kopie produkčních, případně externích dat), data co nejvíce odrážející produkční prostředí • Reálná data s pozitivními scénáři
Testovací cykly	Sprint + 1
Časový horizont	2 týdny
Formální dokumentace testu (ANO/NE)	Ano
Zaznamenávání závad a následná opatření (ANO/NE)	Ano
Hlášení o stavu (ANO/NE)	Ano

Zdroj: vlastní zpracování

- **Vstupní kritéria**
  - Uživatelské testovací případy jsou připravené k testování
  - Zkušební (Stag) prostředí je připraveno a dostupné pro uživatele
  - Tester dokončil integrační, systémové a regresní testování
  - Do fáze akceptačních testů se nemůže přenést žádná chyba vysoké priority, výjimku může schválit jen product owner
  - Do fáze akceptačních testů se můžou přenést maximálně dvě chyby střední priority, výjimku může schválit jen product owner
  - Do fáze akceptačních testů se může přenést maximálně tři chyby nízké priority, výjimku může schválit jen product owner

- **Vstupní artefakty**
    - Seznam funkcionalit vyvinutých v rámci sprintu
- **Výstupní kritéria**
    - Všechny nové funkcionality jsou otestované a schválené
    - Všechny kritické chyby jsou odstraněné
    - Závady nízké úrovně se akceptují jen se svolením businessu
- **Výstupní artefakty**
    - Výstupní report uživatelského akceptačního testování
    - Chyby jsou reportované
    - Schváleno product ownerem

4.3 Role a odpovědnosti

Každý v týmu má jasně definovanou roli a odpovědnosti z toho plynoucí. Doporučuje se, aby jeden pracovník měl přesně jednu roli.

- **Vývojář**
    - Píše zdrojový kód aplikace a unit testy
    - Účastní se scrumových meetingů, hlavně plánovacího
- **Tester**
    - Provádí testování
    - Píše automatické testy (Integrační, systémové, …)
    - Sbírá chyby od businessu (Je potřeba ověřit, než se předá vývojářům)
    - Podílí se na plánování testování
- **SWD Product owner**[1]
    - Spravuje a rozhoduje za aplikaci z pohledu SWD (Software delivery oddělení)

[1] Pozice SWD product ownera a business product ownera může být a většinou je vykonávána jednou osobou

- **Business product owner**
    - Spravuje a rozhoduje za aplikaci z pohledu businessu
- **Scrum master**
    - Řídí vývoj softwaru
    - Zastřešuje komunikaci mezi businessem a vývojovým týmem
- **Testovací autorita (QA manager/test lead)**
    - Schvaluje nasazování nových verzí
    - Plánuje testování
    - Kontroluje testování
    - Aktualizuje strategii testování
- **Architekt**
    - Designuje, jak bude aplikace vypadat
    - Účastní se týmových meetingů
    - Stará se o technickou stránku aplikace (Technický dluh)
- **Ostatní role**
    - UX Designer – Navrhuje prototypy uživatelského rozhraní
    - SAP Konsultant – Vývoj na straně SAP, převážně RFC
    - Database admin – Správa databází

Tabulka 7 - Zodpovědnosti při realizaci testů

Etapa/ Role	**Unit testing**	**User story testování**	**Integrační testy**	**Systémové testy**	**Uživatelské testy**
Prostředí	Dev	Test	Test	Test	Stag
Koordinuje	Vývojář	Tester	Tester + Vývojář	Tester + Vývojář	Tester
Provádí	Vývojář	Tester	Tester	Tester	Uživatel
Reportuje	Vývojář	Tester	Product owner	Product owner	Vývoj. tým
Podpora	Vývojář	Vývoj. tým	Vývoj. tým	Vývoj. tým	Vývoj. tým
Sponzor	Architekt	QA Lead	QA Lead	QA Lead	Product owner

Zdroj: vlastní zpracování

Každá role má odpovědnost v různé aktivitě nastavenou jako primární (jsem hlavní odpovědný) a sekundární (účastním se této aktivity, případně zastupuji).

Tabulka 8 - Matice odpovědnosti

Aktivita	Project Manager	Architekt	QA Manager (Test lead)	Tester
Plánování a časový odhad testů	S	S	P	
Schválení test plánu	P	P	P	
Dokumentace testování	S	S	S	P
Příprava a provedení testů			P	S
Nastavení testovacího prostředí	P	P		
Opravy chyb a vrácení k re-testingu		P		P
Průběžné reportování o testování			P	S
Závěrečná testovací zpráva			P	S

Zdroj: vlastní zpracování

P – Primární; S-Sekundární

Tabulka 9 - Raci matice odpovědností

Aktivita	Project Manager	Architekt	QA Manager (Test lead)	Tester
Plánování a časový odhad testů	I	C	A	R
Schválení test plánu	R	R	R	C
Dokumentace testování	I	I	A	R
Příprava a provedení testů	I	C	A	R
Nastavení testovacího prostředí	I	C	A	R
Opravy chyb a vrácení k přetestování	C	R	A	R
Průběžné reportování o testování	I	I	A	R

Závěrečná testovací zpráva	I	C	R	A

Zdroj: vlastní zpracování

4.4 Prostředí a jeho požadavky

Testeři mají přístup do vývojového, testovacího a zkušebního prostředí pro účely testování. Vývojové a zkušební prostředí se používá jenom, když testy nejdou z nějakého důvodu provést v prostředí testovacím.

- Každé prostředí má určeného svého správce
- Všichni, kdo do jednotlivých prostředí přistupují, mají nastavená práva podle své role
- Testovací prostředí musí být včas nastaveno a zkontrolována jeho dostupnost a funkčnost před zahájením testování
- Jsou k dispozici technici s vhodnými dovednostmi pro podporu testovacího prostředí
- Testeři (DevOps specialisté) jsou odpovědní za nastavení a údržbu testovacího prostředí. Zajišťují, že prostředí pro testování i všechny externí systémy jsou k dispozici a jsou synchronizovány
- Tester by měl mít připravené a správně nastavené nástroje pro testování a přístup do systému na řízení vývoje a zadávání chyb a být na ně správně proškolen
- Je definováno zálohování testovacích dat a strategii obnovy prostředí

Vývojové – Development (Dev)

Vývojové (špinavé) prostředí, kde jsou vypnuté automatické testy, kromě unit testů. Používá se převážně k vývoji, dost často se stává, že je nedostupné nebo nefunkční.

Testovací – Testing (Test)

Prostředí sloužící hlavně pro testovací team, prostředí, kde ladí svoje automatické testy. Provádí manuální testování bez toho, aby ovlivnili zákaznické testování na Stag prostředí.

Zkušební – Staging (Stag)

Zkušební prostředí je téměř přesná replika produkčního prostředí pro uživatelské testování aplikace. Má stejné nastavení jako produkční prostředí. V začátcích implementace této strategie bude napojena na testovací databázi s výhledovým přepojením v případě, že se nebudou objevovat chyby s kritickou prioritou na produkční databázi.

Produkční – Production (Prod)

Produkční prostředí připojené na produkční databázi. Vývojový tým a tím i testeři mají omezený přístup a případný přístup se vydává jen na omezenou dobu a jen na vyhrazené moduly. Jediné testy, které se budou provádět, budou po nasazení speciální smoke testy, a to hned po nasazení verze ze zkušebního na produkční prostředí, a to hlavně s cílem rychle ověřit, jestli bylo vše správně nasazeno a správně komunikuje.

4.5 Testovací nástroje

Strategie staví na automatizaci, což ale neznamená, že se bude vše bezmyšlenkovitě automatizovat. Před každým test plánem se musí udělat analýza pro vhodný nástroj a každá část modulu se musí analyzovat, jestli je vhodná investice do testování nebo je to business kritický proces a nemůže se stát, že nebude dostupný.

Rozhodování se pro automatizaci

Celý proces včetně jednotlivých kroků a komentářů bude předán ke schválení:

1. Výběr testovacího nástroje

   Cena? Open Source? Potřebné zkušenosti nebo školení?
2. Rozsah automatizace

   Co vše bude automatizované? Kolik času bude potřeba? Vyplatí se?
3. Plánování, návrh a vývoj

   Zapadá do harmonogramu vývoje nebo testování?

4. Spouštění testů

   Kdy? Jak? Pravidelně?

5. Údržba

   Kolik bude potřeba času na údržbu? Nebude na sebe vázat zbytečně lidské zdroje?

Požadavky

Na každého testera bude potřeba jeden klientský počítač s nainstalovaným příslušným softwarem. Všechny dokumenty SWD/Společnosti musí být testerům k dispozici.

Automation hub

Jedná se o aplikaci, která simuluje všechna hardwarová zařízení, potřebná na výrobní lince, jako jsou tiskárny, skenery a další zařízení. Při automatizaci MES by se musela vytvořit jedna speciální linka na simulaci všech těchto zařízení, což není proveditelné, proto se využívá tento nástroj, který se navenek chová, respektive vrací data stejná jako hardwarová zařízení bez jejich fyzické potřeby.

Potřebný software

Důraz kladen především na open source aplikace. Pro samotné testování není nutný žádný specifický software, ale doporučuje se nějaká aplikace na snímek obrazovky, nástroj na testování API a textový editor.

Testovací nástroje

Bude rozebráno v test plánu pro specifický produkt. Nástroje se mohou lišit produkt od produktu, ale doporučuje se mít jednotné nástroje a jazyk skrz celé oddělení. Standardem je programovací jazyk C# a nástroj pro unit testy a integrační testy XUnit, ale jejich použití není pravidlo.

4.6 Normy, standardy a požadavky

Vyvíjený software musí splňovat standardy, normy a požadavky zákazníků, které se přenáší na firmu jako dodavatele a kybernetické bezpečnosti. Toto téma je

velice rozsáhlé a vydalo by na vlastní diplomovou práci. Z pohledu testování by měly být tyto požadavky součástí zadání. Zmiňuji proto, aby se na ně myslelo při testování. Osoby zodpovědné za kybernetickou bezpečnost můžou zasahovat do požadavků na testování. Ve specifických situacích z pohledu bezpečnosti nebo utajovaným informacím může být testování předáno rovnou na uživatele a testeři fungují jako mentoři.
Zde uvádím důležité průmyslové normy, více se dá dozvědět na stránkách [28]

- ISO 9001
- ISO 14001
- ISO 50001
- ISO 45001

Normy kybernetické bezpečností více se dá dozvědět ze zdrojů [27],[29] a [30]

- ISO 27001
- ISO 27002
- ISO 27017
- ISO 27018
- ISO 15408

4.7 Výstupy testování

Všechny výstupy testování, včetně této strategie, budou vhodně uschovány s jejich verzí. Hlavní verze bude podléhat schválení testovací autority a vedlejší verze bude schvalována testerem zodpovědným za vypracování. Přehled změn bude uveden na začátku každého dokumentu spolu s datem schválení, jménem a komentářem.

Tabulka 10 - Ukázka verzování

Jméno	Pozice	Datum	Verze	Komentář
Test Testerovič Tester	Tester	2.22.2022	1.3	Doplněn test na testování testu a obejití dvou faktorového ověření přes speciální interní účet.

Zdroj: vlastní zpracování

Testovací strategie

Strategie testování je návod, který je třeba dodržet pro dosažení cílů testování a provedení jednotlivých typů testů uvedených v plánu testování. Zabývá se analýzou rizik, rozdělením kompetencí a cíli testů. Tvorba a dokumentace testovací strategie by měla být prováděna systematickým způsobem, aby bylo zajištěno, že všechny cíle jsou plně pokryty a pochopeny všemi zúčastněnými stranami. Měla by být také často přezkoumávána, zpochybňována a aktualizována s tím, jak se organizace a produkt v průběhu času vyvíjejí. Kromě toho by se strategie testování měla zaměřit také na sladění všech zúčastněných stran z hlediska terminologie, úrovní testů, integrace, rolí, odpovědností, sledovatelnosti, plánování zdrojů atd.

Testovací plán

Dává kolekci všech testů pro danou oblast, v našem případě epicu. Bude udávat, co a jak se bude testovat. Pro každou fázi testů bude v rámci plánování definováno konkrétní pokrytí a priority testů. Bude obsahovat a definovat cíle testování. Bude vydáván pro každou novou oblast a po každé verzi bude aktualizována podle implementace nových funkcionalit. Bude vždy vycházet z této testovací strategie a bude zohledňovat i případné rozdíly mezi prostředími a různými verzemi. Bude sloužit převážně pro implementaci automatických testů. Součástí bude i harmonogram jednotlivých prací ze strany testovacího týmu.

Testovací scénář

Testovací scénář je definován jako jakákoli funkčnost, kterou lze testovat. Jedná se o soubor testovacích případů, který pomáhá testovacímu týmu natavit pozitivní a negativní testovací scénáře projektu. Poskytuje rámcovou představu o tom, co potřebujeme testovat.

Test Status Report

Účelem tohoto reportu bude kontrola klíčové metriky, stanovení priority defektů, zkontrolovat aktuální stav a shrnout testovací plán pro další období. Bude prováděn minimálně jednou týdně v rámci každodenní schůzky.

Testovací specifikace

Bude obsahovat podrobný přehled jednotlivých testovacích scénářů, jak budou testovány, jak často budou testovány pro jednu danou funkcionalitu. Specifikace testů budou vypracovány pro každou fázi a verzi testování čímž vznikne vztah jedna ku jedné mezi plánem testování a jeho specifikací.

Testovací skript

Bude obsahovat soubor instrukcí napsaný pomocí skriptovacího nebo programovacího jazyka, který se bude provádět na testovaném systému pro ověření jeho funkcionality podle očekávání a požadavků. Testovací skripty se používají při automatizovaném testování. Každý test skript by měl obsahovat tyto atributy a být dobře okomentován z důvodu jednoduché správy a případných závislostí:

- Jedinečný identifikátor testovacího skriptu
- Název
- Účel testovacího skriptu
- Předpoklady (Pre-requisites)
- Potřebná data
- Jednotlivé kroky provedení testu v postupném pořadí
- U každého kroku uvést očekávané výsledky
- Označit komentářem chybějící/nedokončené kroky
- Regresní štítek (Označení testů, které budou tvořit balíček regresních testů)

Matice požadavků

V nástroji pro řízení vývoje bude propojen zákaznický požadavek s user story na vývoj (jediný druh user story co může zadávat i někdo mimo tým), který se rozbije na jednotlivé požadavky jako třeba vývoj funkcionality, prototyp uživatelského rozhraní, úpravu uživatelského rozhraní, požadavek na úpravu SAP RFC, požadavek na automatický test a případný požadavek na performance testy. To vše

bude zastřešeno pod jedním velkým požadavkem zvaným epic. Dokud nebude celý epic uzavřen a tím i otestovaný a případně pokryt automatickým testem nemůže se zavřít uživatelský požadavek. Tímto postupem bude zajištěno, že bude uživatelský požadavek vyvinut a otestován podle zadání. V průběhu celého životního cyklu vývoje bude mít zadavatel přístup k jednotlivým user story a může je komentovat.

Test report

Na konci testování je nezbytné shromáždit data a předložit report o testování. Za jeho vypracování odpovídá testovací autorita nebo jí pověřený zástupce. Zde jsou popsány oddíly, které budou jeho součástí:

- Souhrn pro management
- Výsledky testů v porovnání s testovacími scénáři
- Natavení systému a testovacího prostředí
- Soupis testů, které prošly
- Soupis testů, které neprošly s důvodem proč a odkazem na tiket, kde se bude řešit
- Seznam splněných/nesplněných výstupních kritérií
- Případné výstupy z výkonových testů
- Nedokončené testy spolu s plánem nápravných opatření

Provedení testů bude rozděleno na následující sady:

- Základní funkcionalita
- Podrobně nová funkcionalita
- Přetestování, jak oprav funkcionality, tak i regresního testování
- Závěrečné regresní testování
- Seznam aktuálních oproti očekávaným výsledkům

Testovací report bude průběžně aktualizován podle výsledků testů včetně informace, jestli byl test úspěšný, úspěšný s podmínkou nebo neúspěšný. Součástí by měl být i časový odhad, jak dlouho bude trvat oprava nalezených bugů.

Metriky testovací

Nepovinné, ale doporučené pro sledování, jak se v čase vyvíjí.

- Celkový počet testovacích případů vs. počet provedených testovacích případů
- Výsledky provedených testovacích případů (vyhověl/nevyhověl)
- Seznam závad podle závažnosti (Měsíčně? Trendy?)
- Seznam závad podle stavu (Měsíčně? Trendy?)

Získané zkušenosti (Lessons learned)

Na konci každého sprintu případně na konci testování testovací tým pošle zprávu o případných získaných zkušenostech, kterou dodá koordinátorovi testů, případně Scrum masterovi. Tato zpráva bude součástí test reportu v rámci hodnocení ukončeného sprintu. Jeden z dokumentů potřebný na retro meeting.

Nahrávky testování

Nejlepší test report je nahrávka celého testování. U automatických testů uživatelského rozhraní je běžné si zpětně pustit, jak probíhal test a podle toho vyvodit závěry případná opravná řešení.

Tato technika se důrazně doporučuje i při exploratory testing, kdy se nenahrává jenom obrazovka, ale kamera je za zády testera a nahrává i jeho chování před monitorem při vyhodnocení těchto testů a případné kroky pro reprodukci nějaké chyby jsou jednoduché. Proto je jeden z požadavků této strategie v pravidelných iteracích nastavit exploratory testing, zaměřit se na jeden modul, část funkcionalit, vyhradit si čas, třeba dvě hodiny a tento test provést. Výstupem by nemělo být samotné video nebo záznam zvuku, ale jeho analýza a zadané chyby do systému proto určeného.

4.8 Rizika a jejich mitigace

Zde jsou vybrána nejpravděpodobnější rizika s jejich návrhem mitigace a jejich vliv na celý proces testování.

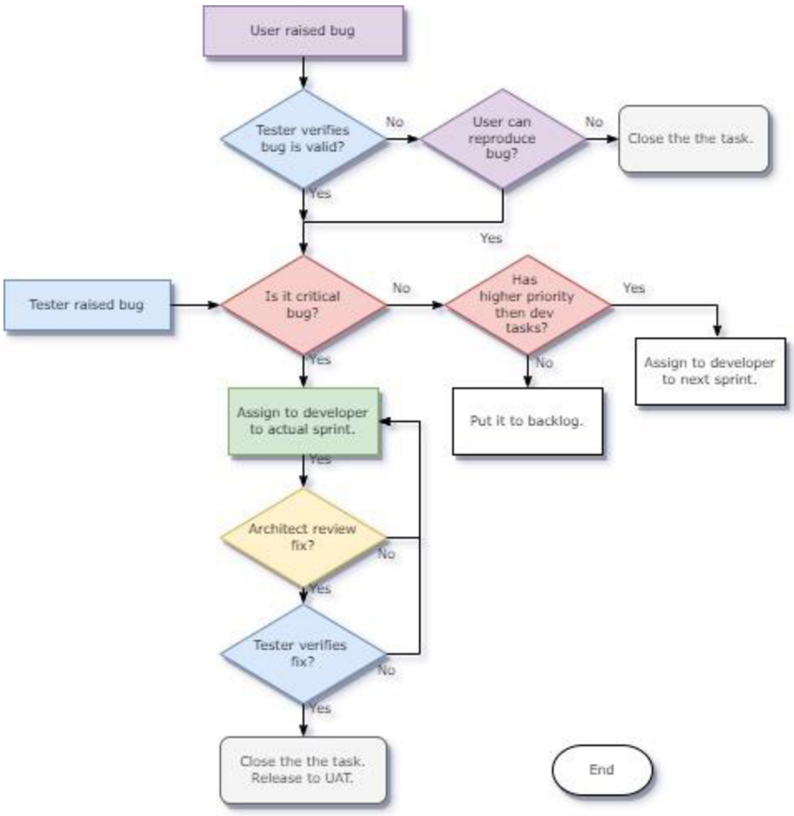
Tabulka 11 - Rizika

#	Risk	Mitigace	Vliv
1	Funkcionalita není vyvinutá v požadovaném čase	Pravidelná kontrola časového plánu a případná úprava zdrojů kde je potřeba	High
2	Testovací prostředí a nástroje nejsou připravené na začátek testování	Zainteresované týmy musí mít připravený i plán přípravy pro začátek testování	High
3	Zpoždění při opravě kritické chyby a následný požadavek na přetestování	Od vývojového týmu bude požadován důraz na řešení kritických chyb a jejich prioritního plánování	High
4	Vývojový, testovací, projektový tým vyžadují vzájemnou kooperaci. Pokud není někdo k dispozici zdržuje to projektový postup.	Všichni požadovaní musí být dostupní v kritických momentech nebo ustanovit jejich zástup	Med
5	Neočekávané závislosti mezi testovanými komponentami, které vyžadují revizi testovacích scénářů	Informace o závislostech jsou pravidelně aktualizované a komunikované, aby bylo možné upravit testovací scénáře	Low

Zdroj: vlastní zpracování

4.9 Nástroj pro reportování a správu

Ke sledování problémů, jejich oprav a řešení se bude používat systém vhodný systém – JIRA nebo jiný. Nepředpokládá se, že budou použity jiné testovací nástroje. Chyby při testování budou považovány za vzniklé, pokud testované produkty vykazují chování v rozporu s dohodnutým požadavkem nebo vykazují chování mimo něj. Každá taková odchylka bude případně v systému zaznamenaná jako chyba. Správa těchto testovacích chyb bude plně v kompetenci testovacího týmu, aby bylo zajištěno jejich řešení, opakované testování a případné uzavření.



Obrázek 7 - Workflow chyby
Zdroj: vlastní zpracování

Prioritizace

V systému se budou používat převážně vysoká, střední a nízká. Kritická a kosmetická jsou podkategorie vysoké a nízké.

Reportování a sledování chyb u každého projektu je jedna ze zásadních věcí pro úspěch projektu. Proto bude každé user story, případně chybě, přiřazena tato závažnost (severita) a priorita, podle které se bude k těmto požadavkům nebo chybám přistupovat. Budou tři druhy user story – chyba, uživatelský požadavek, analyzovaný požadavek určený k vývoji a epik, který bude schraňovat požadavky pro jednu funkcionalitu. Vše v jednom backlogu.

Tabulka 12 - Prioritizace user story

Závažnost (Severita)	**Priorita (Priority)**	**Definice závažnosti**	**Řešení/ Tolerance**
Kritická	Překážka dalšího postupu	• Závada má za následek selhání celého systému	• Všechny kritické závady musí být odstraněny před začátkem UAT.
Vysoká	Konec konkrétního testu	• Ovlivňuje kritické funkce v jedné nebo více oblastech • Není nouzového řešení • Zásadní vliv na výkon • Ovlivňuje další systémy	• Pokud není výslovně odsouhlaseno businessem, nesmí se do UAT dostat • Všechny závady s vysokou závažností musí být odstraněny před ukončením UAT
Střední	Konec konkrétního testu	• Nelze dokončit úlohu nebo uživatelské rozhraní výrazně mate uživatele • Není přesně implementován požadavek	• Chyby se střední závažností se můžou dostat do UAT • Všechny závady se střední prioritou musí být odstraněny před ukončením UAT
Nízká	Konec konkrétního testu	• Úloha jde s větším úsilím dokončit • Ovlivňuje konkrétní rozhraní, ale je zde	• Chyby s nízkou závažností se můžou dostat do UAT • Všechny chyby s nízkou

		náhradní řešení	závažností musí být odstraněny před ukončením UAT
Kosmetická	Malá úprava vzhledu	• Problém nemá vliv na funkčnost systému a lze jej snadno vyřešit	• Mohou se dostat do UAT • Všechny kosmetické chyby musí být odstraněny před ukončením UAT

Zdroj: vlastní zpracování

Každá zadaná user story nebo chyba bude nabývat tyto statusy podle jejich průchodem vývojovým cyklem softwaru.

Tabulka 13 - Stav user story

Status	**Popis**
New	Vytvořena nová user story.
To do	User story je přiřazena někomu z vývojového týmu, ale zatím na něm nezačal pracovat.
In Progress	Člen vývojového týmu začne na user story pracovat.
Review	User story je splněna a přesune se na nadřazenou autoritu architekta, test lead nebo product ownera aplikace na kontrolu.
Test	Funkcionalita nebo opravená chyba je nasazena na testovací prostředí a je připravena k testování.
Done	Funkcionalita je implementována, otestována a může být nasazena na zkušební prostředí.
Closed	Chyba byla opravena a testy byly úspěšné.
Reopen	Když byla user story uzavřena, ale chyba/požadavek se zase objevil.
Rejected	Zadaná user story nebude řešena. Nejedná se o chybu nebo není požadavek schválen.
Duplicate	User story se již objevuje v backlogu.

Out of Scope	Vyřešení user story náleží oddělení mimo SWD.
On Hold	Práce na user story je přerušena a čeká se na vstup někoho mimo vývojový tým.

Zdroj: vlastní zpracování

Pouze testovací autorita (QA manager nebo test lead) a SWD product owner může změnit user story do jednoho z těchto statusů:

- Rejected
- Duplicate
- Out of scope
- On hold

Každá user story bude označena štítkem neboli labelem v nástroji pro management projektu podle toho, na jaký tým je zamýšlen a mohlo se podle něj mezi user story filtrovat.

Tabulka 14 - Štítkování user story

Štítek (Label)	**Popis**
FE	User story je určena na tým vývoje uživatelského rozhraní (Front-end)
BE	User story je určena na tým vývoje funkcionalit (Back-end)
SAP	User story je určena na SAPový team. Potřeba upravit RFC.
QA	User story pro testovací tým (Quality assurance). Performance testy, požadavek na re-test nebo vývoj automatických testů.
DO	Zkratka pro DevOps. User story na vývoj/úpravu prostředí.

Zdroj: vlastní zpracování

Každá user story může spadat do několika kategorií z důvodu lepší správy a přiřazování na řešitele.

Tabulka 15 - Kategorie user story

Kategorie	Popis
Funkční	• Chybějící funkce, zobrazení nebo špatné nastavení (např. práva) • Problémy s funkčností aplikace
Integrační	• Problém s příchozími nebo odchozími daty mezi moduly • Špatně načtená počáteční data
Analytické	• Nejde se přihlásit do aplikace, i když je správně přiřazen přístup
Uživatelský problém	• Zadaná chyba nebo podmět od uživatele
Výkonové problémy	• Problémy s výkonem nebo během produktu
Chyba produktu	• Chyby a vady produktu
Nastavení testu	• Problémy s nastavením testu nebo nevhodné kroky testu
Chybná testovací data	• Nevalidní testovací data
Ostatní	• Všechny, co nejdou zařadit do jiné kategorie

Zdroj: vlastní zpracování

Požadavky user story

Každá user story, chyba nebo požadavek musí mít zadány tyto atributy. Platí, že čím více dat je vyplněno, tím rychleji se user story vyřídí a řešitel se nemusí doptávat na chybějící informace. Některé z nich automaticky vyplní nástroj pro řízení vývoje, ale je třeba zmínit pro případ, kdyby šel požadavek „mimosystémově".

- Jedinečný identifikátor (Automaticky)
- Název zadavatele (Automaticky)
- Datum a čas zadání problému (Automaticky)
- Popis testovaného problému (čím podrobněji tím lépe)
- Kroky k reprodukování (1. Spustit SW, 2. Otevřít, 3. …)
- Podpůrné informace, např. snímky obrazovky
- Fáze testu
- Číslo verze (označení buildu)
- Závažnost user story (High, Medium, Low)
- Případně při provádění, jakého testovacího scénáře bylo zjištěno

Eskalace user story

Všechny chyby, závady, připomínky nebo dotazy vzniklé během testování, které vzniknou v průběhu a nejdou rychle vyřešit „na místě" třeba pomocí informací z dokumentace nebo telefonátem, je třeba formálně předat k řešení konkrétnímu týmu případně rovnou na specifickou osobu.

V případě, že v průběhu řešení user story přibyla nějaká data nebo skutečnosti, které by mohly pomoci při procesování user story, budou přiloženy.

Prioritizace user story

- Pokaždé když vznikne nějaká user story, bude jí přiřazena jedna ze tří priorit, případně jejich pod varianta
- V případě potřeby od businessu, product ownera nebo i v případě, že vývojář nebo tester usoudí, že není priorita nastavena správně, může se její změna navrhnout na nejbližším denním meetingu

- Pokud se v průběhu testování objeví nějaký kritický problém, budou o něm informovány všechny příslušné strany

Retesting

Nasazování jednotlivých verzí může být běžné, to znamená naplánované anebo mimořádné, které se bude vydávat v případě nalezení kritické chyby aplikace a její potřeby okamžité implementace na produkční prostředí. Tyto mimořádné potřeby vývoje a jejich následné přetestování mají přednost před ostatními aktivitami a jejich potřeba přerušuje též probíhající testy do její opravy. Tato potřeba může nastat i mimo pracovní dobu!

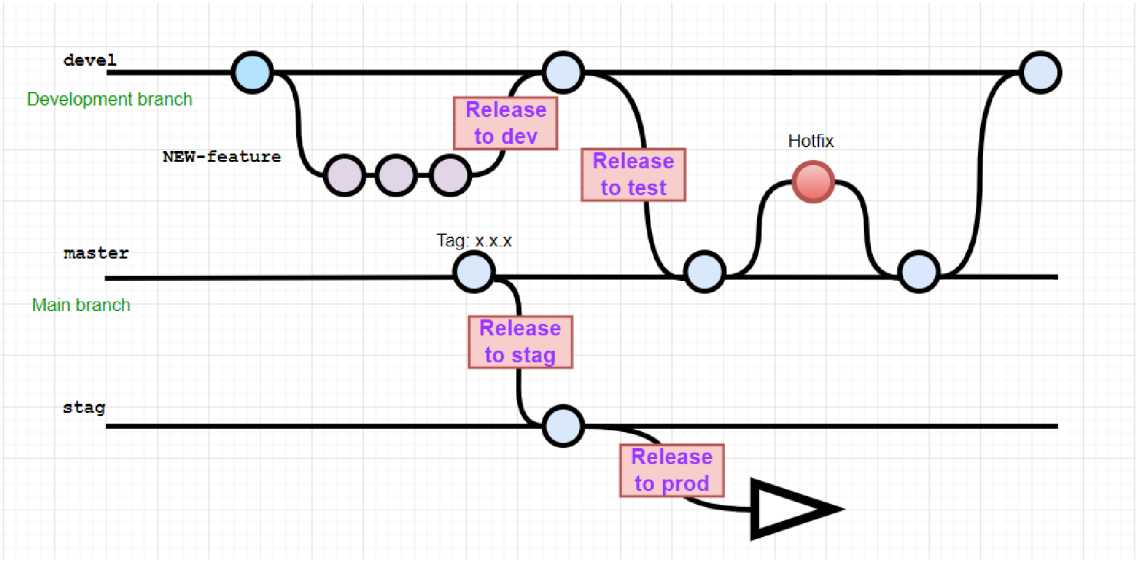
V případě nalezení jiné chyby než kritické nebo vysoké závažnosti se pokračuje v testování a nalezená chyba se podle priority zařadí do plánu vývoje. Ideálně do dalšího sprintu.

4.10 Management vydávání verzí

Nezbytná je kontrola verzí a dodržování jejího procesu/pravidel. Neplánované rychlé opravy (Hot fix & Quick fix) se nezakládají z vývojové (Dev) GIT větve, ale z master. V případě, že změny provedené v master nebudou implementovány i do vývojové větve se rozbije proces vydávání verzí, kvůli rozdílnému kódu v obou větvích (tok nové funkcionality „proti směru“). Nebo se v rámci nasazování verze na produkční prostředí nasadí funkcionalita, jenž nemá být dostupná nebo se špatným nastavením. Proto je důležité dodržovat pravidla:

- Všechny rychlé opravy zpětně nahrát i do development větve
- Kontrolovat, jestli kód ze všech uživatelských větví je integrován do společné větve
- Každé prostředí má mít vlastní konfigurační soubor
- Kontrolovat rozdíly mezi konfiguračními prostředími jednotlivých prostředí
- Jasně specifikováno, kdo vydává/schvaluje nasazení nové verze

- Dev – architekt po schváleném review
- Test – tester (SWD product owner)
- Stag, Prod – business product owner



Obrázek 8 - GIT větve
Zdroj: vlastní zpracování

Každá nasazené verze musí mít svůj unikátní identifikátor s přiloženým soupisem, co bylo v této verzi nasazeno. Je určeno pro kontrolu, jestli bylo všechno správně nasazeno a otestováno. Také se používá v případě, kdy se nasadí verze na produkci a je potřeba se vrátit na předchozí verzi,

4.11 Test a sprint management

Výčet schůzek a dalších aktivit v agilním vývoji z pohledu testovacího týmu.

Test status reporting

Během fáze testování se budou pravidelně konat schůzky ohledně jeho stavu. Účelem těchto schůzek bude přezkoumání klíčových metrik, stanovení prioritních defektů, přezkoumání aktuálního stavu a shrnutí plánu testování pro nadcházející týden. Všem zúčastněným stranám bude každý týden poskytnuta souhrnná zpráva. Účastníci schůzky:

- Testovací autorita

- Testeři
- Architekt
- Business product owner
- SWD product owner

Sprintové aktivity

Schůzky v rámci řízení vývoje pomocí agilních technik z pohledu testového týmu.

Daily stand up

Každodenní schůzka vývojového týmu, kde testeři reportují, jestli časové odhady náročnosti testování odpovídají plánu a reportují nově nalezené chyby.

Refinement

Schůzka, kde se před začátkem sprintu projdou znovu požadavky a revidují se časové odhady potřebné pro vývoj, respektive testování.

Review meeting

Schůzka, kde se prochází ukončený sprint z pohledu nových funkcí systému. Dodaných a nedodaných naplánovaných user story. V rámci schůzky by mělo být diskutováno z pohledu testového týmu:

- Aktivita jednotlivých členů
- Plánování a odhadování testů
- Přezkoumání a schválení plánu testování
- Dokumentace (výstupy) testování
- Příprava a provedení testů
- Nastavení testovacího prostředí
- Opravy chyb a vrácení testovacímu týmu k opakovanému testování
- Průběžné podávání zpráv o testech
- Souhrnné zprávy o testech
- Defect triage

- Rozřazování závad je proces nebo mechanismus, při kterém je každé chybě přiřazena priorita.

Retro meeting

Po ukončení sprintu nebo testování je naplánován retro meeting, kde se zpětně zhodnotí, co se povedlo, nepovedlo, co potřebuje zlepšení a co jsme se naučili.

5 Závěr a doporučení

Podle základních požadavků managementu na časově nenáročné a efektivní testování a na základě nastudované literatury byla vytvořena strategie testování softwaru. Soustředila se na obecný názor, že pro vývoj softwaru je velmi důležité řádné plánování a rozsáhlé testování podle robustní testovací strategie, jinak může snadno dojít k překročení rozpočtu. Neopomenula, že proces tvorby a aktualizace testovací strategie v agilním vývoji je komplikovaný, a proto je dobré se postupně zaměřit na každý aspekt. Výsledkem je strategie ověření kvality softwaru spoléhající na automatizaci, opírající se o agilní řízení s cílem v DevOps filozofii průběžného nasazování. Rozdělila testování do na sebe navazující pětici základních druhů testů, ve čtyřech specifických prostředích. Nastavila procesní pravidla, rámce a meze. Výstupy z testování rozebrala na jednotlivé na sebe navazující části a určila jejich časový rámec a smysl.

Pokud se její praktická aplikace podaří, bude se jednat o obrovský skok od monolitické aplikace bez automatického, a vlastně i manuálního testování k produktu v prostředí microservices a implementací průběžného nasazování. To znamená nejenom nové technologie, ale i přístup k vývoji, respektive testování. Strategie testování je jen jeden z kroků implementace DevOps. Při studiu se často vyskytoval i názor, že se musí implementovat i nové myšlení, a tím vlastně nastane něco jako kulturní revoluce ve vývoji. Pohled na testery se musí změnit a více je vnímat jako DevOps inženýry nebo mentory testování. Vývojáři, testeři i management jsou jeden tým a ten je zodpovědný za všechny testy. Odpovědnosti a specializace mezi členy vývojového týmu se stírají. Musí se změnit myšlení vývojářů a managementu od projektového přístupu k produktovému.

6 Seznam použité literatury

[1] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profes!onal. ISBN 978-80-247-5594-6.

[2] Supply Chain Management (SCM) Definition. Investopedia: Sharper insight, better investing. [online]. Dostupné z: https://www.investopedia.com/terms/s/scm.asp

[3] MES systém (Manufacturing Execution System). [online]. Dostupné z: http://www.mescenter.org/cz/clanky/5-co-je-to-mes-system

[4] 10 Functions Of An MES System I PINpoint. PINpoint Information Systems, Inc. [online]. Copyright ©2022 PINpoint Information Systems, Inc. [cit. 13.03.2022]. Dostupné z: https://www.pinpointinfo.com/blog/10-functions-of-an-mes-system-pinpoint

[5] MYERS, Glenford J., Tom BADGETT a Corey SANDLER. The art of software testing. 3rd ed. Hoboken, New Jersey: Wiley, c2012. ISBN 978-1118031964.

[6] GRAHAM, Dorothy, Rex BLACK a Erik van VEENENDAAL. Foundations of software testing: ISTQB certification. Fourth edition. Australia: Cengage, [2020]. ISBN 978-1-4737-6479-8.

[7] GELPERIN, D. a B. HETZEL. The growth of software testing.Communications of the ACM [online]. 1988, 31(6), 687-695 [cit. 2022-03-09]. ISSN 0001-0782. Dostupné z: doi:10.1145/62959.62965

[8] BJERKE-GULSTUEN, Kristian, Emil Wiik LARSEN, Tor STÅLHANE a Torgeir DINGSØYR. High Level Test Driven Development – Shift Left. LASSENIUS, Casper, Torgeir DINGSØYR a Maria PAASIVAARA, ed. Agile Processes in Software Engineering and Extreme Programming [online]. Cham: Springer International Publishing, 2015, 2015-5-16, s. 239-247 [cit. 2022-03-09]. Lecture Notes in Business Information Processing. ISBN 978-3-319-18611-5. Dostupné z: doi:10.1007/978-3-319-18612-2_23

[9] What is Software Testing? Definition of Software Testing, Software Testing Meaning - The Economic Times. Business News Today: Read Latest Business news, India Business News Live, Share Market & Economy News | The Economic Times [online]. Copyright © 2022 Bennett, Coleman [cit. 13.03.2022]. Dostupné z: https://economictimes.indiatimes.com/definition/software-testing

[10] What is Quality Assurance(QA)? Process, Methods, Examples [online]. Dostupné z: https://www.guru99.com/all-about-quality-assurance.html

[11]KANER, Cem, Jack L. FALK a Hung Quoc NGUYEN. Testing computer software. 2nd ed. New York: John Wiley, 1999. ISBN 978-0471358466.

[12]JORGENSEN, Paul. Software testing. Boca Raton: Auerbach. 2008. 416 s. ISBN 0-8493-7475-8

[13]DEDIĆ, Nedim a Clare STANIER. Measuring the Success of Changes to Existing Business Intelligence Solutions to Improve Business Intelligence Reporting. TJOA, A Min, Li Da XU, Maria RAFFAI a Niina Maarit NOVAK, ed. Research and Practical Issues of Enterprise Information Systems [online]. Cham: Springer International Publishing, 2016, 2016-11-18, s. 225-236 [cit. 2022-03-06]. Lecture Notes in Business Information Processing. ISBN 978-3-319-49943-7. Dostupné z: doi:10.1007/978-3-319-49944-4_17

[14]ARORA, Monika a Deepankar CHAKRABARTI. Application of Business Intelligence: A Case on Payroll Management. In: 2013 International Symposium on Computational and Business Intelligence [online]. IEEE, 2013, 2013, s. 73-76 [cit. 2022-03-13]. ISBN 978-0-7695-5066-4. Dostupné z: doi:10.1109/ISCBI.2013.22

[15]80-20 Rule Definition & Example (Pareto Principle). Investopedia: Sharper insight, better investing. [online]. Dostupné z: https://www.investopedia.com/terms/1/80-20-rule.asp

[16]MILLER, Joan C. a Clifford J. MALONEY. Systematic mistake analysis of digital computer programs. Communications of the ACM [online]. 1963, 6(2), 58-63 [cit. 2022-03-13]. ISSN 0001-0782. Dostupné z: doi:10.1145/366246.366248

[17]GÁLA, Libor, Jan POUR a Zuzana ŠEDIVÁ. Podniková informatika. 2., přeprac. a aktualiz. vyd. Praha: Grada, 2009. Expert (Grada). ISBN 80-247-2615-7.

[18]CHAPPLE, Mike, James Michael STEWART a Darril GIBSON. (ISC)^2 CISSP certified information systems security professional: official study guide. Eighth edition. Indianapolis, Indiana: John Wiley, [2018]. ISBN 978-1-119-47593-4.

[19]NEWMAN, Sam. Building microservices: designing fine-grained systems. Second edition. Beijing: O'Reilly, 2021. ISBN 978-1-492-03402-5.

[20]JAMSHIDI, Pooyan, Claus PAHL, Nabor C. MENDONCA, James LEWIS a Stefan TILKOV. Microservices: The Journey So Far and Challenges Ahead. IEEE Software [online]. 2018, 35(3), 24-35 [cit. 2022-03-15]. ISSN 0740-7459. Dostupné z: doi:10.1109/MS.2018.2141039

[21]MITRA, Ronnie a Irakli NADAREISHVILI. Microservices: up and running: a step-by-step guide to building a microservices architecture. Beijing: O'Reilly, 2020. ISBN 978-1-492-07545-5.

[22]What is Supply Chain Management? | Oracle Česká Republika. [online]. Dostupné z: https://www.oracle.com/cz/scm/what-is-supply-chain-management/

[23]AMMANN, Paul a Jeff OFFUTT. Introduction to software testing. SecondEdition. Cambridge, United Kingdom: Cambridge University Press, [2017]. ISBN 978-1-107-17201-2.

[24]GOERICKE, Stephan, ed. The Future of Software Quality Assurance [online]. Cham: Springer International Publishing, 2020 [cit. 2022-04-03]. ISBN 978-3-030-29508-0. Dostupné z: doi:10.1007/978-3-030-29509-7

[25]KINSBRUNER, Eran. Continuous testing for DevOps professionals: a practical guide from industry experts. [Velká Británie]: [CreateSpace Independent Publishing Platform], 2018. ISBN 978-1-7271-3217-5.

[26]Are Containers Replacing Virtual Machines? - Docker. Home - Docker [online]. Copyright © 2022 Docker Inc. All rights reserved [cit. 10.04.2022]. Dostupné z: https://www.docker.com/blog/containers-replacing-virtual-machines/

[27]About the OWASP Foundation | OWASP Foundation. OWASP Foundation, the Open Source Foundation for Application Security | OWASP Foundation [online]. Dostupné z: https://owasp.org/about/

[28]Jaké standardy se vztahují na strojírenský a zpracovatelský průmysl?[online]. Dostupné z: https://www.nqa.com/cs-cz/certification/sectors/engineering-manufacturing

[29]NCSC. NCSC [online]. Dostupné z: https://www.ncsc.gov.uk/section/about-ncsc/what-we-do

[30]About NIST | NIST. National Institute of Standards and Technology | NIST [online]. Dostupné z: https://www.nist.gov/about-nist

[31]MARTIN, Philippe. Kubernetes : Preparing for the CKA and CKAD Certifications. Berkley, United States: aPress, 2020. ISBN 9781484264935.

[32]AXELROD, Arnon. Complete Guide to Test Automation : Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. Berkley, United States: aPress, 2018. ISBN 9781484238318.

[33]About - Git. Git [online]. Dostupné z: https://git-scm.com/about

[34]Shift Left. Home [online]. Dostupné z: https://devopedia.org/shift-left

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Aleš Stránský**
Studium:	I2000205
Studijní program:	N0688A140001 Informační management
Studijní obor:	Informační management
Název diplomové práce:	**Strategie zajištění kvality softwaru**
Název diplomové práce AJ:	Software quality assurance strategy

Cíl, metody, literatura, předpoklady:

Cílem práce je navrhnout testovací strategii softwaru

V teoretické části se autor zaměří na obecné testování softwaru, jeho metodiku a automatické testy v microservices.

V praktické části autor vypracuje strategii testování podle posledních poznatků a literatury.

BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profes!onal. ISBN 978-80-247-5594-6.

MYERS, Glenford J., Tom BADGETT a Corey SANDLER. The art of software testing. 3rd ed. Hoboken, New Jersey: Wiley, c2012. ISBN 978-1118031964.

GRAHAM, Dorothy, Rex BLACK a Erik van VEENENDAAL. Foundations of software testing: ISTQB certification. Fourth edition. Australia: Cengage, [2020]. ISBN 978-1-4737-6479-8.

KINSBRUNER, Eran. Continuous testing for DevOps professionals: a practical guide from industry experts. [Velká Británie]: [CreateSpace Independent Publishing Platform], 2018. ISBN 978-1-7271-3217-5.

GOERICKE, Stephan, ed. The Future of Software Quality Assurance [online]. Cham: Springer International Publishing, 2020 [cit. 2022-04-03]. ISBN 978-3-030-29508-0. Dostupné z: doi:10.1007/978-3-030-29509-7

Garantující pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	Mgr. Josef Horálek, Ph.D.
Datum zadání závěrečné práce:	21.1.2020