Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra systémového inženýrství

Bakalářská práce

Zavedení testovací metodiky v projektu ve vybraném podniku

Tomáš Kuryl

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Tomáš Kuryl

Informatika

Název práce

Zavedení testovací metodiky v projektu ve vybraném podniku

Název anglicky

Test methodology implementation in selected company

Cíle práce

Hlavním cílem bakalářské práce je analýza existujících přístupů k testování v rámci řízení projektů, výběr vhodných metod pro potřeby vybraného projektu a jejich implementace do metodiky testování.

1. Popis artefaktů testování – Testovací příprad, Testovací scénář, Test plán, Testovací strategie
2. Vymezení činností testování, popis aktivit uvnitř testovacího cyklu a definice vstupů a výstupů.
3. Zaměření na proces tvorby metodiky z pohledu řízení projektu a jeho velikosti.
4. Rozebrání nástrojů určených pro podporu testování.
5. Analýza as-is stavu vybraného projektu z pohledu testování a určení slabých míst.
6. Určení takových metod, které je účelné implementovat pro zlepšení kvality dodávané aplikace ve vybraném projektu.
7. Vyhodnocení realizace / plánu realizace opatření, která byla nebo budou přijata k implementaci do vybraného projektu.

Metodika

Metodika řešené problematiky je založena na studiu a analýze odborných informačních zdrojů. Teoretická část bude obsahovat základní charakteristiky testování, popis testovacího procesu a metod testování vzhledem k životnímu cyklu vývoje software, řízení testování jako jedné z disciplín testování. Tato část se zaměří i na nástrojovou podporu testování a řízení chyb. V praktické části bude řešeno sestavení ucelené metodiky na základě zjištěných skutečností z teoretické části. Dále pak úprava použitých metod na základě požadavků daného projektu.

Doporučený rozsah práce

30-60 stránek

Klíčová slova

Testovací metodika, Testovací proces, Testování, Řízení testování, Agilní projekt

---

Doporučené zdroje informací

BUREŠ M., RENDA M., DOLEŽEL M. a kolektiv: Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016, ISBN 978-80-247-5594-6

HAMBLING B., MORGAN P., SAMAROO A., THOMPSON G., WILLIAMS P.: Software Testing: An ISTQB-BCS Certified Tested Foundantion – 4th edition. London: British Computer Society, 2019, ISBN 1780174926

KUNCE E., ŠOCHOVÁ Z.: Agilní metody řízení projektů. Praha: Computer Press, 2014, ISBN 978-80-251-4194-6

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

SCHWALBE K.: Řízení projektů v IT: Kompletní průvodce. Praha: Computer Press, 2007, ISBN 978-80-251-1526-8

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Petra Pavlíčková, Ph.D.

Garantující pracoviště

Katedra systémového inženýrství

Elektronicky schváleno dne 25. 2. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 8. 3. 2022

doc. Ing. Tomáš Šubrt, Ph.D.

Děkan

V Praze dne 15. 03. 2023

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Zavedení testovací metodiky v projektu ve vybraném podniku" jsem vypracoval samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 14.3.2023 _____

Poděkování

Rád bych touto cestou poděkoval paní doktorce Petře Pavlíčkové za pomoc a cenné rady při realizaci této práce a své manželce za podporu.

Zavedení testovací metodiky v projektu ve vybraném podniku

Abstrakt

Bakalářská práce se zabývá analýzou možných přístupů k testování a jejich uplatněním v rámci projektu. Teoretická část se věnuje procesu testování jako systematické činnosti, která patří do životního cyklu vývoje software a která pomáhá zlepšovat kvalitu a spolehlivost softwarových aplikací tím, že odhaluje chyby a nedostatky. V práci jsou definovány základní koncepty testování, vyjasňuje také návaznosti mezi artefakty testování, technikami, úrovněmi a typy testů. V druhé části práce je představen stávající proces testování ve vybraném reálném projektu. Zjištěné skutečnosti jsou následně porovnány s teoretickými východisky uvedenými v první části práce. Na základě tohoto srovnání jsou poté navržena opatření, která pomohou testování zlepšit, a to včetně plánu jejich realizace.

Klíčová slova: testování, testovací metodika, řízení testování, projektové řízení, kvalita software, testovací nástroje, artefakty testování

Test methodology implementation in selected company

Abstract

The bachelor's thesis deals with the analysis of possible approaches to testing and their application within a project. The theoretical part focuses on the testing process as a systematic activity that belongs to the software development life cycle and helps to improve the quality and reliability of software applications by detecting defects and deficiencies. The thesis defines the basic concepts of testing, clarifies the relationships between testing artifacts, techniques, levels, and types of tests. In the second part of the thesis, the existing testing process in a selected real project is presented. The findings are then compared with the theoretical foundations outlined in the first part of the thesis. Based on this comparison, measures are proposed to improve testing, including a plan for their implementation.

Keywords: testing, testing methodology, test management, project management, quality of software, testing tools, testing artifacts

Obsah

1 Úvod

Software hraje v dnešní době podstatnou roli ve všech myslitelných oblastech lidského konání. V průmyslu a obchodu je software nezbytný pro automatizaci procesů a optimalizaci výroby. Díky automatizaci se zvyšuje efektivita, rychlost a dochází k úspoře nákladů. Přínos automatizace ve formě konkrétní softwarové aplikace je přímo úměrný její kvalitě a proto platí, že čím kvalitnější software máme, tím existuje větší pravděpodobnost, že aplikace přinese očekávané efekty. Testování software je činnost, která podstatným způsobem přispívá ke kvalitě výsledného řešení. V současnosti je pevnou součástí vývojového cyklu, který je realizován v projektech.

Provedení testů je pouze špičkou ledovce celého testovacího procesu. Aby mohl být tento proces prohlášený za dostatečně efektivní, je potřeba přihlédnout k požadavkům a cílům projektu a zároveň vzít v úvahu nejlepší existující přístupy k testování. Zavést takovou metodiku testování je nelehký úkol a obvykle se tak přistupuje k nastavení, které v plné šíři nereflektuje skutečné potřeby zadavatele nebo organizace. V již běžících projektech je minimální vůle cokoli měnit, pokud k tomu není dostatečný důvod, nicméně pokud tyto důvody nastanou, může být revize stávajících metodik, jejich úprava a případné nové nastavení vhodným krokem, který dokáže výrazně ovlivnit výslednou kvalitu. Správně zvolená metodika testování určuje rámec toho, jak testování v projektu probíhá, jak je testování řízeno a organizováno a jaké metody se používají. Pomáhá tak najít společnou řeč v interním projektovém týmu i mezi ostatními stakeholdery.

2 Cíl práce a metodika

2.1 Cíl práce

Hlavním cílem bakalářské práce je analýza existujících přístupů k testování v rámci řízení projektů, výběr vhodných metod pro potřeby vybraného projektu a jejich implementace do metodiky testování.

1. Popis artefaktů testování - Testovací případ, Testovací scénář, Test plán, Testovací strategie
2. Vymezení činností testování, popis aktivit uvnitř testovacího cyklu a definice vstupů a výstupů
3. Zaměření na proces tvorby metodiky z pohledu řízení projektu a jeho velikosti
4. Rozebrání nástrojů určených pro podporu testování
5. Analýza as-is stavu vybraného projektu z pohledu testování a určení slabých míst
6. Určení takových metod, které je účelné implementovat pro zlepšení kvality dodávané aplikace ve vybraném projektu
7. Vyhodnocení realizace / plánu realizace opatření, která byla nebo budou přijata k implementaci do vybraného projektu

2.2 Metodika

Metodika řešené problematiky je založena na studiu a analýze odborných informačních zdrojů. Teoretická část bude obsahovat základní charakteristiky testování, popis testovacího procesu a metod testování vzhledem k životnímu cyklu vývoje software, řízení testování jako jedné z disciplín testování. Tato část se zaměří i na nástrojovou podporu testování a řízení chyb. V praktické části bude řešeno sestavení ucelené metodiky na základě zjištěných skutečností z teoretické části. Dále pak úprava použitých metod na základě požadavků daného projektu.

3 Teoretická východiska

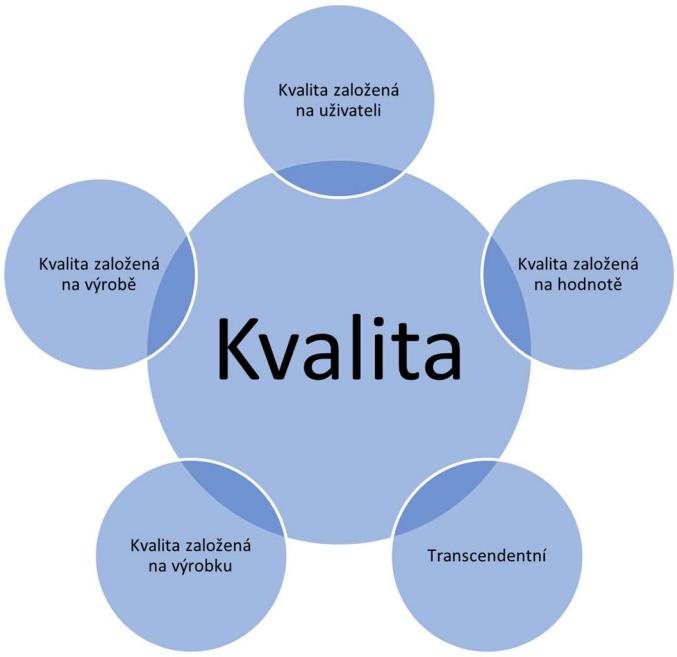
Abychom byli schopni zadefinovat pojmy, které souvisí s testováním, musíme nejprve poznat skutečnosti, které nám do testování vstupují a nějakým způsobem i ovlivňují přístup k celé problematice.

3.1 Kvalita a zajištění kvality

3.1.1 Kvalita software

Pokud se podíváme do oxfordského slovníku, zjistíme, že kvalita je charakterizována jako úroveň porovnání něčeho s něčím dalším ve smyslu, jak dobré nebo špatné to je. (Oxford Learner's Dictionaries, b. r.)

Klasifikace a definice pojmu kvalita je v praxi vnímána rozdílně. Pod termíny „kvalita", „kvalitní" či „nekvalitní" si většina z nás představí něco odlišného. Rozdílné chápání tohoto pojmu ilustruje studie, kterou zpracoval David Garvin. Hovoří o různých pohledech, kdy na jedné straně něco kvalitního poznáme na první pohled, aniž bychom rozumově byli schopni uvést důvody (transcendentní pohled). Na druhé straně zmiňuje technické hledisko, kdy výrobek dle rozsahu splňuje danou specifikaci (výrobkový pohled). Zároveň uvádí i další perspektivy jako např. cenu, kterou je zákazník ochoten za něco kvalitního zaplatit (hodnotový pohled). Atributový pohled předpokládá, že míra kvality jednotlivých dílů ovlivní výsledný produkt nebo míra naplnění uživatelských potřeb a požadavků a jejich vazba na zamýšlené použití produktu (uživatelský pohled). (Roudenský and Havlíčková, 2013)



Obrázek 1 - Perspektivy kvality

Zdroj: Gregory (2020), zpracování vlastní

S posledně jmenovanou charakteristikou souhlasí i Lewis, Dobbs a Veerapillai (2009), kteří tvrdí, že kvalitou se rozumí souhrn vlastností a charakteristik výrobku nebo služby, které mají vliv na schopnost uspokojovat stanovené nebo předpokládané potřeby.

3.1.2 Zajištění kvality

„Zajištění kvality je systematická činnost, která poskytuje důkazy o vhodnosti celého softwarového produktu k použití.“ (Lewis, Dobbs and Veerapillai, 2009, p. 16)

Je potřeba si uvědomit, že se jedná o nikdy nekončící úsilí, kdy jsou všechny procesy, které působí na výsledný produkt neustále vyhodnocovány a zlepšovány. Z pohledu softwarového vývoje mluvíme o souboru činností, které se používají k řízení projektu tak, aby s vysokou mírou jistoty byly splněny stanovené cíle.

3.1.3 Kontrola kvality

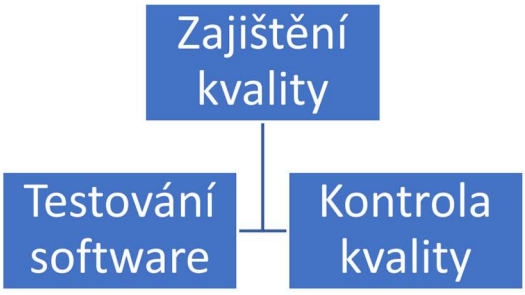
Kontrola kvality je činnost, kdy se porovnává jakost produktu proti platným normám. Primárně se zaměřuje na revize a odstraňování chyb před tím, než výsledný produkt opouští výrobní prostory nebo hardware výrobce. Z pohledu softwarového vývoje se kontrola

kvality zaměřuje na kontrolu kódu, revizi specifikací a dokumentů, které souvisí s vyvíjeným softwarem. Protože se jedná o systematickou činnost, jsou předem stanoveny milníky, při kterých se kritéria daná kontrolou kvality aplikují. (Lewis, Dobbs and Veerapillai, 2009)

3.1.4 Vztah zajištění kvality a testování

Z praxe víme, že termíny testování a zajištění kvality spolu souvisejí, ale často dochází k jejich chybné interpretaci a záměně. Hlavním rozdílem je zejména přístup k řešení nalezených chyb a problémů. Zatímco zajištění kvality představuje proaktivní přístup, který nám zaručuje, že správné nastavení všech procesů eliminuje budoucí selhání a chyby, testování je zaměřeno na reaktivní přístup, kdy různými technikami vyhledáváme v softwarovém produktu chyby, které do něj již byly zaneseny. (Page, Johnston and Rollison, 2009)

Zde je třeba zmínit, že zajištění kvality je rámec, který je součástí oboru nazývaného řízení kvality. Testování jako proces nebo sada aktivit je součástí zajištění kvality. Do této hry nám ještě vstupuje kontrola kvality. Někteří autoři jako například Lewis, Dobbs a Veerapillai (2009) tvrdí, že kontrola kvality a testování jsou aktivity, které spolu souvisejí, ale stále se jedná o relativně nezávislé činnosti.

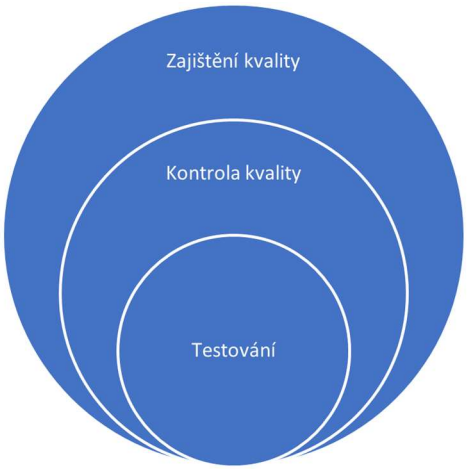


Obrázek 2 – Testování software a kontrola kvality jako nezávislé činnosti

Zdroj: Lewis, Dobbs and Veerapillai (2009), zpracování vlastní

Jiní autoři, mezi něž patří např. Hambling (2015) uvádí, že testování software je součástí kontroly kvality a ta je součástí zajištění kvality. Kontrola kvality ověřuje, zda se dosahuje požadované úrovně kvality. Testování je dle jeho mínění jedním z nástrojů,

které kontrola kvality používá, nicméně může použít i jiné techniky – např. dotazování nebo měření.



Obrázek 3 - Testování jako součást kontroly kvality

Zdroj: Hambling (2015), zpracování vlastní

3.2 Projekt a řízení projektu

„Projekt je dočasná organizace vytvořená za účelem dodání jednoho nebo více obchodních produktů podle dohodnuté obchodní strategie. Projektové řízení je plánování, delegování, monitorování a kontrola všech aspektů projektu (vč. motivace zúčastněných) za účelem dosažení cílů projektu v očekávaném čase, nákladech, kvalitě, rozsahu, přínosu a očekávaných rizicích.“ (Office Of Government Commerce, 2012, p. 3)

Dle PMI (2017) je projekt definován jako dočasná snaha o vytvoření hodnoty prostřednictvím jedinečných produktů, služeb a výsledků. Řízení projektů pak definuje jako využití specifických znalostí, dovedností, nástrojů a technik na projektové činnosti s cílem splnit požadavky na projekt.

3.2.1 Klasifikace projektu

Klasifikace projektů umožňuje rozdělit projekty do skupin podle společných vlastností. Takto klasifikace pak pomáhá identifikovat nejlepší postupy pro každý typ projektu a lépe tak projekt řídit. Výslednou klasifikaci (velikost) projektu určuje celá řada kritérií. Pro potřeby práce jsou zmíněna taková, která nejvíce ovlivňují IT projekty.

3.2.1.1 Trvání

Trvání projektu je důležitý indikátor, který určuje výslednou velikost projektu. Delší trvání koreluje s vyšší potřebou zdrojů, vyššími náklady, množstvím chyb a riziky selhání.

3.2.1.2 Rozpočet

Pravděpodobně nejdůležitější kritérium společně s trváním projektu. Náklady jsou nejsledovanějším indikátorem, který je v průběhu projektu sledován, protože jakýkoliv neúspěch má negativní dopad na finanční zdroje a obchod.

3.2.1.3 Složitost

Složitost určuje velikost projektu svým nárůstem. Podstatné je její měření. Může být dána množstvím funkcionalit, které výsledný produkt obsahuje, případně počtem řádků kódu nebo množstvím týmů, které jsou do projektu zapojeny.

3.2.1.4 Obchodní hodnota

Faktor, který sleduje, jakou hodnotu projekt přináší ve formě financí, případně k postavení společnosti v konkurenčním prostředí na trhu. Pokud má projekt výrazný dopad na stav peněžních toků, měl by mu být věnována patřičná pozornost.

3.2.1.5 Použitá metodika vývoje

Vybraná metodika řízení projektu určuje, jakými kroky a pokyny se musí projekt řídit a jak musí být personálně zajištěn. (Neumeyer, 2022)

3.3 Metodiky vývoje software

Z pohledu vývojového cyklu software mluvíme o řadě metodik, které jsou použitelné s ohledem na specifika IT projektu. Některé metodiky se liší pouze v drobnostech, jiné jsou tak odlišné, že je velmi obtížné je adaptovat s ohledem na kulturní nebo sociální zvyklosti. Pro účely bakalářské práce jsem zvolil tři nejčastější metodiky, které na projektech potkáváme v našem civilizačním okruhu. Všechny z nich mají samozřejmě své výhody a nevýhody.

3.3.1 Vodopád

Model vývoje softwaru popsal již v 70. letech 20. století Dr. Winston Royce. Je zajímavé, že ve své práci popisuje tuto metodiku jako více-iterativní. Podle jeho názoru je vhodné tento model projít alespoň dvakrát s tím, že první iterace je prototypovací, pro lepší pochopení požadavků, technologie a zákaznických potřeb.

Vodopád se skládá z několika na sebe navazujících bloků, sekvenci kroků shora dolů. Všechny kroky jsou provedeny jednou a každý další krok může začít pouze v případě, že předešlý krok byl ukončen. (Palmquist *et al.*, 2013)

V literatuře se můžeme setkat s různou úrovní detailu a z toho odvozeného počtu kroků, které metodika obsahuje. Pro účely práce jsem vybral kroky, které uvádějí Page, Johnston a Rollison (Page, Johnston and Rollison, 2009):

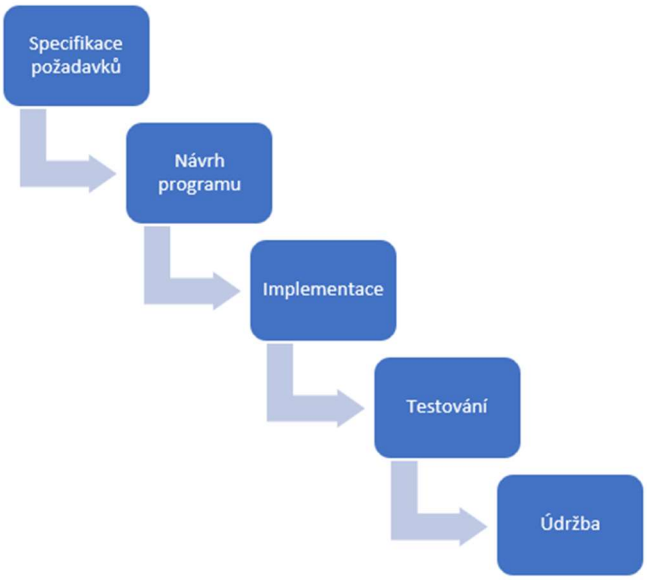
1. Specifikace požadavků
2. Návrh programu
3. Implementace / Programování
4. Testování
5. Údržba

Výhody:

- Slouží jako podklad pro detailní projektový plán
- Zákazník ví, co přesně může očekávat a kolik ho to bude stát
- Lidské zdroje na projektu je možné jednoduše plánovat dle fází

Nevýhody:

- Testování začíná až na konci fáze – chyby v návrhu nelze jednoduše ostranit
- Nízká flexibilita – změny jsou implementovány s obtížemi
- Finální výsledek je k dispozici až na konci projektu



Obrázek 4 - Metodika vodopád

Zdroj: Page, Johnston and Rollison (2009), zpracování vlastní

3.3.2 Iterativní vývoj

Na rozdíl od Vodopádu vzniká výsledný software přírůstkově (inkrementálně) v rámci cyklů. Na začátku není nutné rozpracovat a uzavřít všechny požadavky, ale pouze takové, které jsou poplatné pro danou fázi. Následně se pokračuje přes všechny milníky, které jsme si definovali u Vodopádu. Projekt se dělí na cykly, které mají jasně definované náklady a časový rámec. Po ukončení testování a vyhodnocení začíná další cyklus. Tento proces se opakuje, dokud není vytvořen zcela funkční systém. Zcela klíčové je zapojení koncových uživatelů do testování, tím se minimalizuje riziko, že na konci vývoje vznikne nevyhovující systém. (Hambling, 2015) Stejně jako ostatní přístupy má i tento způsob vývoje dle Allana (2019) výhody a nevýhody:

Výhody:

- Relativně snadná implementace změn
- Cykly zrychlují proces vývoje
- Nákladově efektivní
- Jednodušší testování v rámci iterací

Nevýhody:

- Sekvenční přístup – iterace se nesmí překrývat
- Definice přírůstků bývá náročná na zdroje
- Může dojít ke zvýšení nákladů, pokud se zanedbá plánování

Obrázek 5 - Metodika iterativní vývoj

Zdroj: Hambling (2015), zpracování vlastní

3.3.3 Agilní přístup

Vznikl jako reakce na procesně náročné modely životního cyklu software. Tam, kde model Vodopádu nebo Spirálový model selhává pro přílišnou rigiditu, dostávají šanci metodiky, které řadíme pod tzv. Agilní přístup. V dnešní době zažívají boom, protože potlačují nevýhody tradičních modelů a umožňují dosahování hmatatelných výsledků v relativně krátkém časovém intervalu. Práce probíhá v krátkých časových iteracích v týmech společně se zákazníkem, klade se důraz na komunikaci a rychlé rozhodování. (Page, Johnston and Rollison, 2009)

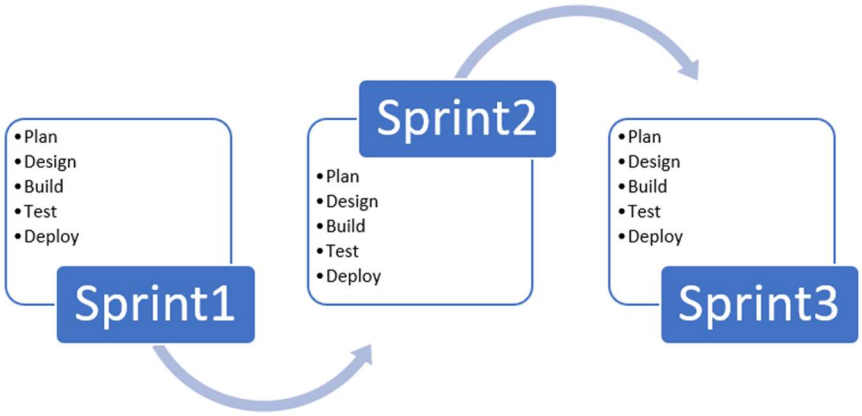
Stejně jako Vodopád, má Agilní přístup řadu výhod a nevýhod. Dle Lonergana (2016) se jedná o následující:

Výhody:

- Vysoká míra flexibility – zapracování změn je relativně jednoduché
- Zákazník obdrží první výsledky relativně rychle

Nevýhody:

- Vysoké nároky na lidské zdroje
- Dokumentace existuje v omezené míře – riziko do budoucna
- Obtížný odhad celkových nákladů
- Nechuť zákazníka být součástí týmu



Obrázek 6 - Agilní přístup

Zdroj: Page, Johnston and Rollison (2009), zpracování vlastní

Tabulka níže porovnává jednotlivé metodiky vývoje software ve vybraných klíčových oblastech. Jak je již na první pohled zřejmé, vodopádový a agilní přístup jsou v těchto ukazatelích v podstatě protikladné. Iterativní vývoj často stojí mezi nimi a jeho zařazení na jednu nebo druhou stranu škály není ve většině případů jednoznačné.

	Metodiky vývoje software		
	Vodopád	Iterativní vývoj	Agilní vývoj
Jasně definované požadavky			
Včasné odhalení potenciálních problémů v projektu			
Flexibilní zapracování změn			
Detailní dokumentace, která může sloužit jako podklad pro projektový plán			
Odhadnutelné náklady na projekt			
Snadné plánování lidských zdrojů			
Aktivní zapojení zákazníka v průběhu celého vývoje a testování			
Potřeba intenzivní spolupráce a komunikace v rámci týmu			
Dodávání průběžných výsledků			

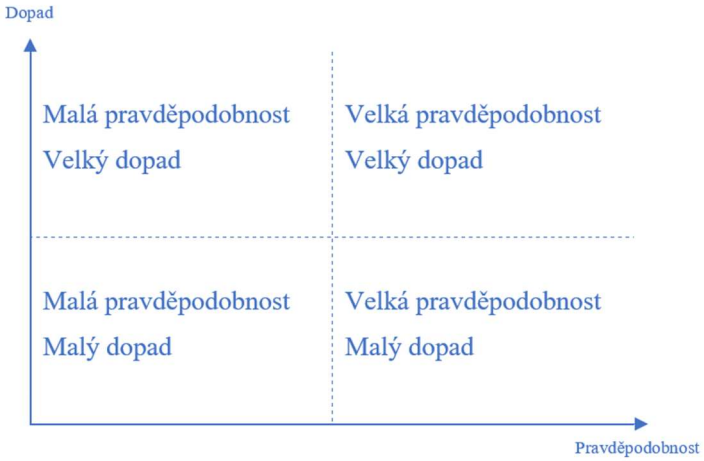
Tabulka 1 - Porovnání metodik vývoje software

Zdroj: Allan (2019), Lonergan (2016), zpracování vlastní

3.4 Rizika

„Riziko je možnost, že se někdy v budoucnu stane něco špatného, situace, která by mohla být nebezpečná nebo mít špatný výsledek.“ (Oxford Learner’s Dictionaries, b. r.)

Pro riziko můžeme najít celou řadu dalších definic. Veskrze se shodují v tom, že se jedná o budoucí událost, která může nastat s různou pravděpodobností a jejíž vyústění může mít takové následky, které mohou ovlivnit (pozitivně i negativně) předpokládaný cíl. Abychom byli schopni jednoduše rizika zaznamenat, může nám pomoci matice rizik, která přispívá k hodnocení rizika tím, že do ní můžeme vynést výši dopadu a pravděpodobnosti jeho vzniku. Rizika pak můžeme snáze řídit. (Hopkin, 2017)



Obrázek 7 - Matice rizik

Zdroj: Hopkin (2017), zpracování vlastní

Řízení rizik by mělo být přirozenou součástí životního cyklu vývoje software, protože rizika vznikají dynamicky v celém jeho průběhu, a proto je nutné na ně včas reagovat.

3.4.1 Produktová rizika

Rizika spojená s kvalitou vyvíjeného produktu. Produkt může být chybový, může dávat uživateli nesprávné výsledky, je uživatelsky nepřívětivý, nedostatečně responzivní nebo pomalý. Právě tato rizika pomáhá snižovat testování software, protože rizika snižuje na úroveň, která je pro zadavatele akceptovatelná.

3.4.2 Projektová rizika

Rizika, která mohou mít dopad na dodávku projektu. Obvykle zasahují oblasti, které nějakým způsobem ovlivňují chod projektu. Rizikem tudíž může být nedostatek lidských zdrojů, nízká seniorita specialistů, zdržení analytické části subdodávky nebo nedostatečná analýza klientských požadavků.

3.4.3 Přístup k řízení rizik

Riziku je nejlépe se zcela vyhnout, ale pokud to není možné, je potřeba s ním dále pracovat.

Ve fázi detekce rizika dochází k zaznamenání a ohodnocení rizika dopadem a pravděpodobností a jeho zaznamenání do tabulky rizik. Mluvíme o tzv. analýze rizik,

kdy z výsledného indexu daného součinem dopadu a pravděpodobnosti odvozujeme finální prioritu rizika. Čím vyšší ohodnocení, tím vyšší je priorita a nutnost ho přednostně řešit.

Jakmile jsou rizika ohodnocena, přichází rozhodnutí o jejich řešení. V ideálním případě se daří rizika eliminovat pomocí testování – díky němu dochází k jejich identifikaci a následně k odstraňování chyb v software. Jedná se logicky o rizika, která odstranit jdou. Část rizik může být přenesena i na jinou část vyvíjeného modulu, rizika s nízkou prioritou jsou komunikována uživateli formou zpráv přímo v uživatelském rozhraní software. Je potřeba zmínit, že některá rizika odstranit nelze, proto je nutné je diskutovat se zadavatelem a navrhovat další alternativní řešení.

Pokud lze riziko kompletně nebo částečně vyřešit, hovoříme o fázi zotavení z rizika. Při kompletním vyřešení se odebírají z tabulky rizik, částečné řešení může výslednou prioritu snížit. V některých případech je nutné navrhnout specifická řešení ve formě nových funkcionalit v software, která rizika mohou eliminovat. (Agarwal, Tayal and Gupta, 2010)

3.5 Verifikace a validace

Uživatelský předpoklad při používání aplikace je ten, že software, který byl vyvinutý bude poskytovat správnou funkcionalitu, ideálně po neomezenou dobu. V rámci zajištění kvality mluvíme o činnostech, které nazýváme Verifikace a Validace.

3.5.1 Verifikace

Činnost Verifikace ověřuje shodu softwarového systému s jeho specifikacemi. Odchylky jsou považovány za chyby nebo selhání v případě, že jsou v rozporu s návrhovými vzory nebo standardy kódování. Selhání se projevuje nesprávným chováním aplikace. Verifikace zahrnuje například ověření chování komponent při vzájemné interakci, které nemusí být postřehnutelné na úrovni pozdějšího použití aplikace ze strany uživatele. Verifikační činnosti pomáhají odhalit chyby při nesprávném použití navržených algoritmů nebo datových struktur.

3.5.2 Validace

Činnost Validace ověřuje, zda software obsahuje všechny funkcionality, které byly definovány zákazníky. Za odchylku se považuje též stav, kdy aplikace obsahuje

funkcionalitu, kterou zákazník nedefinoval. Ta může totiž negativně ovlivnit zbylé funkcionality. Do validačních činností zahrnujeme:

- Systémové testování
- Akceptační testování
- Statistické testování
- Ověření odolnosti software proti chybám
- Činnosti zajištění bezpečnosti systému

Do validačních činností lze zařadit i inspekci kódu nebo další činnosti zaměřené na prevenci problémů ve specifických provozních prostředích zákazníků. (Tian, 2005)

3.6 Testování software

3.6.1 Historie testování software

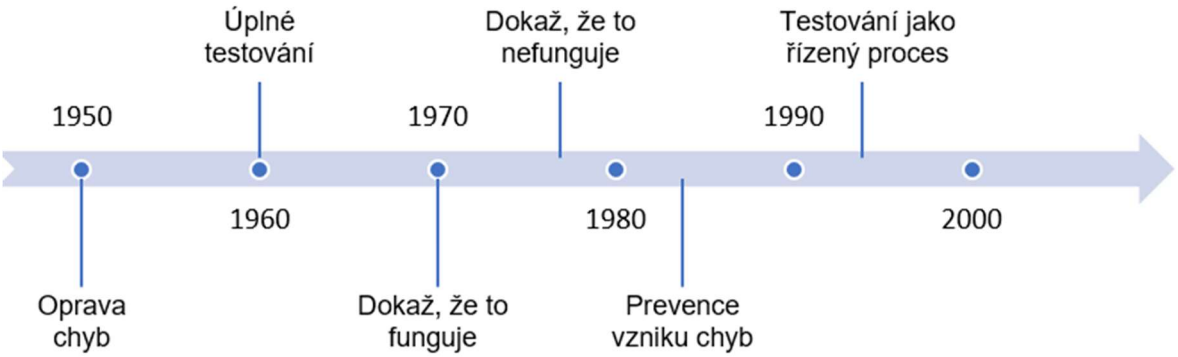
Testování software jde historicky ruku v ruce s vývojem software. Již v 50. letech 20. století mluvíme o testování jako o hledání chyb ve vyvíjených aplikacích. Tuto činnost měli na starosti tvůrci aplikací. V 60. letech 20. století se objevuje koncept, který uvažuje o testování všech průchodů za použití kompletního setu dat. Z dnešního pohledu jsou tyto techniky nepředstavitelné, ale je přirozené, že k tomuto zjištění museli odpovědní pracovníci teprve dospět. Až časová a finanční náročnost tohoto počínání donutila k hledání dalších cest. V 70. letech 20. století se objevuje koncept, který nazýváme: „Průkaz správnosti“. Teorie nám říká, že software se považuje za formálně správný v případě, že za všech myslitelných okolností software dělá, co dělat má. Tento přístup je u jednoduchých a na kód nenáročných aplikací myslitelný, ale u složitějšího software s tisíci řádků kódu zdlouhavý a neefektivní. Nicméně i tak tento koncept do dnešního dne přežil a můžeme ho pro určitý typ testů použít.

Na konci 70. let 20. století se začínáme setkávat s přístupem, který tvrdí, že testování je takové použití programu, jehož cílem je nalézt chybu. Jedná se o přesný opak konceptu Průkazu správnosti. Setkáváme se s návrhem principů testování, které v drobných obměnách existují dodnes a které budou ještě v následujících kapitolách práce zmíněny.

Až v 80. letech 20. století začínáme o testování uvažovat nikoli jen jako o aktivitě ověřující již naprogramovaný kód, ale i o aktivitě, která slouží k prevenci vzniku chyb. Testování začíná pronikat i do ostatních fází vývojového cyklu – testem prochází kromě

aplikace také požadavky, návrh, kód. Pracuje se s tezí, že včasná oprava v návrhu je levnější než následné opravy již naprogramované komponenty. Jsou navrženy první metodiky testování. V téže době se začínají objevovat i automatizované testy jako náhrada testů manuálních.

Přerod testování z pouhé aktivity do řízeného procesu přichází v 90. letech 20. století. Testování je nově definováno jako plánování, návrh, sestavování, provádění a údržba testů a testovacího prostředí. Tento pohled na testování vychází ze zajištění kvality. Objevují se pokročilé nástroje pro řízení testování, reportování výsledků testů, mapování testů na požadavky a předávání chyb. Přichází nástroje pro měření výkonosti aplikací. Přístup k testování tak, jak byl definován v 90. letech 20. století je tu s námi dodnes. Změny, které přicházejí a které se logicky testování dotýkají, souvisí s konceptem samotného vývoje, kdy se od iterativních metodik posouváme k metodikám agilním. (Lewis, Dobbs and Veerapillai, 2009)



Obrázek 8 - Historie testování

Zdroj: Lewis, Dobbs and Veerapillai (2009), zpracování vlastní

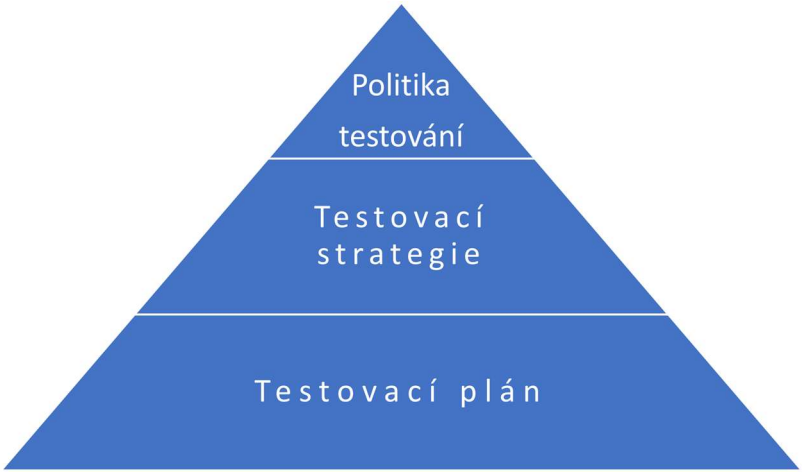
3.6.2 Účel testování

Účelem testování je ověření kvality vyvíjeného software a zjišťování nedostatků, které snižují jeho kvalitu. Testování systematicky zkoumá komponenty nebo celý systém a zjišťuje chyby, které následně předává společně s podklady pro jejich opravu zpět do vývoje nebo k analýze. Nacházením chyb a potvrzením správnosti chování systému se zvyšuje kvalita software a snižuje riziko jeho selhání. (Hambling, 2015)

Jak již bylo zmíněno, vyhledávání chyb je pro testování zásadní, je však potřeba dodat, že toto vyhledávání a nalezení by mělo probíhat co nejdříve, a to již v raných fázích vývojového cyklu. Představme si to tak, že již ve fázi Specifikace, před tím, než začne samotné kódování, dochází k zapojení testování, a to z toho důvodu, že včasná oprava chyby na úrovni Specifikace je násobně levnější než oprava kusu kódu, který nefunguje vinou chyby, která byla ve Specifikaci ponechána. Kromě nalezení chyby je však nutné zajistit i její nápravu a to tak, že poskytneme vývoji potažmo analýze maximum vstupů pro její reprodukci. (Patton, 2002)

3.7 Artefakty testování

Artefakty testování jsou dokumenty a výstupy, které vznikají v rámci procesu testování. Slouží pro posouzení kvality software a jako důkaz o tom, že byl proces testování řádně proveden.



Obrázek 9 - Vztah testovacích dokumentů

Zdroj: Bakanoğlu (2017), zpracování vlastní

3.7.1 Politika testování

Dokument, který vymezuje testování na nejvyšší úrovni v rámci společnosti nebo organizace. Popisuje hlavní cíle a zásady testování. Několika větami je uvedena hodnota, kterou testování přináší. Pokud společnost neprochází vážnějšími organizačními změnami, tak se z dlouhodobého pohledu politika nijak nemění

3.7.2 Strategie testování

Strategie testování je dokument, který popisuje přístup k životnímu cyklu vývoje software. Popisuje obecné požadavky, jak se k testování v konkrétní společnosti přistupuje a základní podrobnosti o testování, obecné metodiky testování. Vymezuje řízení chyb, vysvětluje metriky a měření v rámci reportování testování. Na úrovni společnosti se jedná o víceméně statický přehled toho, jak je testování vymezeno. (Quinn, 2022)

Bureš a spol. (2016) definují testovací strategii jako obecný popis úrovní testování, tak jak jsou prováděny v konkrétní organizaci.

3.7.3 Testovací plán

„Dokument popisující rozsah, přístup, zdroje a harmonogram zamýšlených testovacích činností. Identifikuje testované položky, funkce, které mají být testovány, úkoly testování, kdo bude jednotlivé úkoly provádět, a případná rizika vyžadující plánování nepředvídaných událostí.“ (Institute of Electrical and Electronics Engineers *et al.*, 2008, p. 42)

Norma (Institute of Electrical and Electronics Engineers *et al.*, 2008) uvádí, že testovacích plánů existuje více typů.

3.7.3.1 Master test plán

Hlavní (Master) test plán je dokument, který zastřešuje plánování a řízení testování a pokrývá kompletní cyklus testování. Stanovuje cíle pro každou část, z pohledu času a zdrojů stanovuje dělbu práce, určuje rizika, předpoklady a standardy, které musí být vzaty v úvahu. Identifikuje kontrolní mechanismy v rámci testovacího úsilí, počet úrovní testování, úkoly, které mají být provedeny a požadavky na dokumentaci.

Úvodní část dokumentu zasazuje test plán do kontextu životního cyklu projektu. Kromě úsilí, které bude testování věnováno, popisuje, jak bude testování organizováno, přibližuje harmonogram testování, věnuje se zdrojům, které budou na testování použity, vymezuje odpovědnosti a přibližuje nástroje a techniky, které budou použity.

Druhá část se věnuje procesům testování, požadavkům na dokumentování testů, reportování během testování. Vymezuje též úrovně testování, kterým se následně věnují ve větším detailu test plány pro jednotlivé úrovně.

Poslední část obsahuje slovníček pojmů, zkratek a dále pak proces, jakým dochází ke změnám v test plánu.

3.7.3.2 Úrovňový test plán

Pro každou z úrovní testování vzniká specificky zaměřený test plán, který upřesňuje rozsah, přístup, zdroje a harmonogram testovacích činností. Dále identifikuje testovací položky, testované funkcionality, odpovědnosti za jednotlivé úkoly. Pro každou z úrovní může vzniknout různě podrobný test plán i proto, že požaduje odlišné zdroje nebo testovací prostředí.

Úvodní část zmiňuje především scope testování pro zvolenou úroveň, testovací podmínky a tzv. třídy testování, které můžeme chápat jako testování pozitivních, negativních, nebo různě hraničních hodnot a způsob, jakým bude systém reagovat.

Následující část popisuje pro danou úroveň položky, které mají být testovány, představuje mapu požadavků a jejich pokrytí jednotlivými testy a akceptační kritéria pro každou z úrovní.

Předposlední část se věnuje řízení testování, infrastruktuře, která bude pro testování použita, odpovědnostem a pravomocím, zdrojům testování. V případě, že na úrovni Master test plánu nejsou definovaná rizika, najdeme je právě v této části.

V poslední části najdeme postupy a metriky zajištění kvality, historii změn dokumentu a případný slovníček pojmů pro danou úroveň. (Institute of Electrical and Electronics Engineers *et al.*, 2008)

3.7.4 Testovací případ

„Testovací případ je soubor vstupních hodnot, předpokladů provedení, očekávaných výsledků a postpodmínek provedení, vytvořený pro určitý cíl nebo podmínku testu, například pro průchod určité cesty programu nebo pro ověření shody s určitým požadavkem.“ (Hambling, 2015, p. 80)

Jestliže nám test plán říká „co“ a „proč“, tak testovací případ (test case) odpovídá na otázku „jak“. Při tvorbě případů je potřeba se dohodnout na definici hranic, které budou pokrývat. Jedná se o nelehké rozhodnutí, protože jde o subjektivní činnost, která je závislá na zkušenostech s testováním. Obecně se doporučuje identifikovat nejmenší obchodní činnosti, které software podporuje a pro každou z činností následně definovat sadu

testovacích případů. Postupuje se směrem nahoru od nejmenších činností k větším, případně se činnosti propojují dle obchodního procesu a na ně se pak vážou další případy. Everett a McLeod (2007) dále připomínají, že správný testovací případ obsahuje celou řadu náležitostí:

1) Identifikátor (ID) testovacího případu
   - Mělo by se jednat o jedinečný řetězec čísel nebo písmen. Definuje se obvykle už v test plánu během návrhu testů.
2) Jedinečný název
   - Krátké, výstižné označení, ideálně unikátní, které naznačuje účel testovacího případu.
3) Stručný popis
   - Rozšiřuje název a uvádí další podrobnosti o účelu testovacího případu.
4) Vývojová fáze, ve které se test case provádí
   - Připomíná, v jaké části vývoje by měl být kontrolován kód
5) Cíle testování
   - Kvantifikuje specifické cíle testování a způsob, jakým bude zjištěno, že těchto cílů bylo dosaženo
6) Testovací data
   - Určuje minimální množství testovacích dat pro provedení testovacího případu. Obvykle stačí reprezentativní vzorek dat platný pro obchodní činnost, která je ověřována.
7) Testovací nástroje, které se použijí k vykonání
   - Mohou být uvedeny nástroje pro automatizované testování, které jsou vhodné pro vykonání testu. Výběr závisí na znalostech testovacího týmu
8) Postup spuštění testu
   - Jedná se o sadu předpokladů, které musí být provedeny, aby bylo možné spustit první krok testu. Dokumentuje, jaký hardware, software a data musí být k dispozici pro zahájení testu.
9) Postup ukončení testu
   - Procedura, která dokumentuje, jak ukončit spuštěný software
10) Postup resetu testu pro opakované spuštění

- Dokumentuje sadu podmínek, které je nutné splnit mezi ukončením testu a jeho dalším spuštěním. Obvykle se jedná o obnovení testovacích dat do původního stavu jako bylo před prvním spuštěním.

11) Kroky provedení testu
    - Obsahuje přesný seznam akcí, které krok za krokem tester provádí společně se seznamem očekávaných výsledků v každém z kroků.
12) Čas a datum posledního spuštění testu
13) Počet spuštění testu
    - test case se během testovacího cyklu spouští obvykle vícekrát.
14) Seznam chyb, které tento případ odhalil
    - Chyby jsou navázány na konkrétní testovací případ, aby bylo možné následující opravy opětovně jednoduše testovat.

3.7.5 Testovací scénář

Sady souvisejících testovacích případů lze označit jako testovací scénáře. Návrh a následná implementace takových scénářů může pomoci s uskutečněním regresního testování, případně při testování změn. Mohou být použity i pro školení nových uživatelů. (Institute of Electrical and Electronics Engineers *et al.*, 2008)

Hamilton (2020a) posouvá význam pojmu do trochu odlišné roviny. Tvrdí, že testovací scénář je určitým způsobem nadřazený testovacímu případu, protože scénář definuje, co je potřeba na vyvíjené funkcionalitě testovat tzv. high-level, přičemž testovací případ již obsahuje všechny podmínky, kroky, data a očekávané výsledky, které je potřeba dosáhnout k úplnému otestování. Vztah testovací scénář a testovací případ si lze představit jako relaci 1:M, kdy k jednomu scénáři obvykle náleží jeden a více testovacích případů.

3.7.6 Chyba

„Chyba je odchylka od obchodních nebo technických požadavků“ (Lewis, Dobbs and Veerapillai, 2009, p. 301)

Chyba je definována jako chyba nebo nedostatek softwarového produktu, který způsobuje, že produkt nesplňuje specifikaci nebo neplní jeho účel. (Institute of Electrical and Electronics Engineers *et al.*, 2008)

V literatuře se lze setkat s anglickým označením chyby – Bug nebo Defect. Tyto pojmy jsou většinou zaměňovány, nicméně některé zdroje uvádějí, že Bug je chyba odhalená během vývoje nebo testování, zatímto Defect je chyba, která je zjištěna až při produkčním používání software (Rakjumar, 2022).

V souvislosti se software se můžeme setkat i s pojmy jako selhání a incident. Jejich definice se od pojmu chyba liší. Incident je jakákoliv událost, kterou je nutné prozkoumat, zatímco selhání je následek nalezené chyby.

3.7.6.1 Příčiny chyb

V každé fázi cyklu vývoje software může dojít k zanesení chyby do výsledného řešení. Příčinou může být špatné pochopení požadavků zadavatele, nesprávné interpretace obchodních požadavků a z toho vyplývající chyba ve funkční nebo technické specifikaci, chyba v samotném kódu vyvíjeného software nebo nedokonalost v testovacím případu nebo scénáři. (Bureš *et al.*, 2016)

Nejvíce chyb vzniká v raných fázích vývoje software. Zhruba polovina chyb se do software zanese v rámci tvorby specifikace. Důvodů může být celá řada – specifikace vůbec nemusí existovat, neustále se mění, není dostatečně podrobná nebo ji nemusí tým dostatečně chápat. Zhruba čtvrtina chyb vzniká ve fázi návrhu, který bývá často uspěchaný nebo se opětovně mění bez dostatečného prodiskutování v rámci týmu. Zbytek chyb jde na vrub chybám v kódu a jiným příčinám. Kód je složitý, není dostatečně dokumentovaný, některé chybné přístupy jsou zvoleny z důvodu časové tísně. Jiné příčiny mohou obsahovat chyby v samotném testování nebo nesprávné předpoklady.

3.7.6.2 Náklady na chybu

Náklady rostou exponenciálně s fázemi vývojového cyklu. Oprava chyby ve specifikaci stojí podstatně méně než oprava chyby, která byla objevena až během testování. Z toho vyplývá, že by se kromě dynamických technik nemělo zapomínat na statické techniky testování, které významně šetří náklady na následné opravy v pozdějších fázích vývoje software. (Patton, 2002)

3.7.6.3 Záznam o chybě

Aby byl záznam o chybě kompletní, je potřeba zapsat celou řadu atributů, které usnadní její zatřídění, opravu a následný retest. Podle atributů můžeme hledat příčiny chyb, závažnost a míru dopadu nebo stav oprav. Atributy jsou obvykle přednastaveny (a následně doplněny) implicitně dle použitého nástroje podporujícího evidenci chyb, nicméně závisí na specifikaci konkrétního projektu, jaké atributy finálně použije.

Níže jsou uvedeny obvykle používané atributy:

1. Datum a čas nalezení
2. Tvůrce záznamu
3. Naléhavost
4. Dopad
5. Priorita
6. Aktuální výsledek
7. Očekávaný výsledek
8. Stav
9. Vazba na specifikaci
10. Vazba na testovací případ

Míra a závažnost dopadu, naléhavost a priorita jsou obvykle určeny předem domluvenou škálou, která může být vyjádřena slovně nebo číselně. Velice často dochází ke zjednodušení, kdy se používá pouze priorita.

Záznam o chybě by měl být dostatečně podrobný, aby umožnil nasimulování stávajícího chování a jasně poukázal na chování, které je očekávané, a stejně tak umožnil jednoduchý retest funkcionality po provedené opravě. (Bureš *et al.*, 2016)

Do atributů lze dále zařadit i kroky vedoucí k reprodukci chyby, přiřazení k řešiteli, dostatečně výstižný název nebo oblast funkcionality, kde se chyba vyskytla. (Page, Johnston and Rollison, 2009)

3.7.6.4 Životní cyklus chyby

Cyklus může nabývat různých modelů od jednoduchých po složitější, nicméně všechny mají společné prvky a těmi jsou stavy a přechody. Podle stavů je možné určit v jaké fázi se chyba nachází. Například stav „Nový“ dokumentuje chybu, která byla právě nalezena,

stav „Vyřešen“ chybu opravenou a stav „Uzavřen“ chybu uzavřenou. Přechody pak určují, do jakých stavů se chyba může dle navrženého workflow dostat, resp. jak mezi sebou jednotlivé stavy souvisejí.

V praxi je pak složitost životního cyklu zvolena dle potřeb projektového týmu, nutností spolupráce s dalšími týmy z důvodů integrace nebo složitosti vyvíjeného řešení. (Roudenský and Havlíčková, 2013)

3.8 Metodika testování

Dle stránky Software Testing Help (2023a) jsou metodiky testování metodami, které je možné použít pro otestování funkčních a nefunkčních požadavků. Metodiky tak můžeme považovat za sadu mechanismů (technik, typů a úrovní testování) v rámci zvoleného cyklu vývoje software. Výběr vhodné metodiky testování je považován za jádro procesu testování.

3.8.1 Techniky testování

Testování jako takové je možné založit na rozdílných technikách, tedy v podstatě přístupech, kterými lze k testování přistoupit. Vhodnost jejich použití se může lišit v závislosti na předmětu testování a jeho komplexitě, na fázi vývoje nebo například na zkušenostech testera. V principu je možné dělit techniky na dvě základní skupiny – statické a nestatické.

3.8.1.1 Statické techniky

Předmětem testování jsou požadavky, specifikace nebo kód aplikace, ale bez nutnosti jeho spouštění. Tato technika se uplatňuje v raných fázích vývoje a je velmi vhodná k použití právě proto, že dokáže odhalit příčiny selhání, které by se do kódu mohly dostat a objevit se jako chyba v pozdějších fázích vývoje. Do statických technik řadíme revize, které se zaměřují na specifikaci a další dokumenty související s vývojem, a dále pak statickou analýzu, která se věnuje strukturálním chybám a programovým nedostatkům v samotnému kódu. (Hambling, 2015)

3.8.1.2 Dynamické techniky

Základem testování je spuštěný kód a běžící aplikace. Jde o nejběžnější podobu testování, kdy jsou v běžící aplikaci hledány a reportovány chyby.

Testování bílé skříňky

Testování bílé skříňky je testování založené na kódu, často se o něm mluví též jako o strukturálním testování. Tester vidí do útrob aplikace a na základě toho, jak je funkce implementovaná, dokáže identifikovat vhodné testovací případy. Právě nutnost pochopení, jak je funkcionalita naprogramovaná, klade na testera vysoké nároky. Ten by měl mít povědomí o teorii grafů, aby mohl přesně popsat, co testuje. (Myers, Sandler and Badgett, 2012)

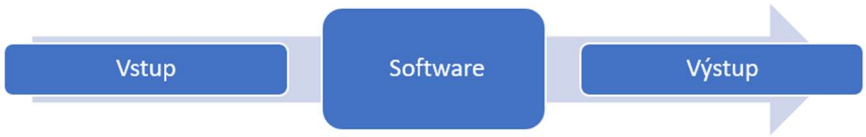
Testování je orientováno na zdrojový kód se zaměřením na tok řízení (přechod řízení z jedné instrukce na druhou) a tok dat (přechod dat z jedné proměnné nebo konstanty do jiné proměnné). (Naik and Tripathy, 2008)

Obrázek 10 - Testování bílé skříňky

Zdroj: Patton (2002), zpracování vlastní

Testování černé skříňky

Testování černé skříňky je na rozdíl od testování bílé skříňky zaměřeno na ověřování vstupů a výstupů. Tester nemá povědomí o vnitřním fungování aplikace, a proto se zaměřuje na podmínky, které jsou dané ve specifikaci. Výhodou tohoto přístupu je, že testování je zcela nezávislé na implementaci a vývoj testů může probíhat paralelně s vývojem software. (Jorgensen, 2014)



Obrázek 11 - Testování černé skříňky

Zdroj: Patton (2002), zpracování vlastní

Testování šedé skříňky

Testování šedé skříňky kombinuje techniky bílé a černé skříňky. Tester rozumí specifikaci a se znalostí struktury aplikace mu velká část testů odpadne, protože danou funkcionalitu může otestovat pouze jednou a nemusí tak zbytečně v různých částech aplikace testovat totožný kód. (Lewis, Dobbs and Veerapillai, 2009)

Testování na základě zkušenosti

Technika testování na základě zkušenosti se uplatňuje v případě, že neexistuje specifikace, ze které by bylo možné vydefinovat vhodné Testovací případy. Prostor dostává zkušenost a znalost testera nebo uživatele k určení slabých míst implementovaného systému. V tomto případě lze uplatnit zjednodušené přístupy ad-hoc testování, nebo více systematické přístupy blížící se průzkumnému testování. Při tom vznikají testovací scénáře během samotného testování. (Hambling, 2015)

3.8.2 Typy testování

Typy testování slouží k vyhodnocení naplnění specifických cílů úrovní testování.

3.8.2.1 Funkční testování

Funkční testování vyhodnocuje funkčnost zkoumaného systému. Nazývá se též testováním na základě specifikace. (Hambling, 2015)

Crispin a Gregory (2009) uvádějí, že cílem funkčního testování je ověření, že software pracuje dle očekávání, neobsahuje chyby a splňuje všechny funkční požadavky na něj kladené.

Lewis, Dobbs a Veerapillai (2009) stejně jako Jorgensen (2014) tvrdí, že mezi funkční testování a testování černé skříňky lze dát rovnítko. Podle nich jde o stejný typ testování, kdy se dle specifikace hodnoty zadané na vstupu ověřují proti hodnotám, které vznikají na výstupu.

3.8.2.2 Nefunkční testování

Testují se aspekty chování systému v souladu s požadavky ve specifikaci. Součástí jsou výkonnostní testy nebo testování použitelnosti. (Hambling, 2015)

Výkonnostními testy se zabývá samostatná podkapitola. Cílem testování použitelnosti je zjistit, zda je aplikace snadno použitelná pro uživatele, jaké jsou jeho reakce během testování aplikace na uživatelské rozhraní, ovládací prvky a celkovou přehlednost. (Roudenský and Havlíčková, 2013)

3.8.2.3 Strukturální testování

Strukturální testování se zaměřuje na testování interních struktur software – kód, architektura, návrh. Zaměřuje se na testování zdrojového kódu software a provádí se pomocí metrik, které berou v úvahu pokrytí kódu a složitosti kódu. (Hambling, 2015)

Stejně jako vztah funkčního testování a testování černé skříňky, je i strukturální testování zaměňováno s technikou bílé skříňky, které zmiňují Lewis, Dobbs a Veerapillai (2009) a Jorgensen (2014).

3.8.2.4 Regresní testování

„Určuje rozsah testů, které je potřeba opakovat při změnách v již prozkoumaných softwarových produktech. Posuzuje povahu změn s cílem určit následné dopady na další aspekty systému. Test se opakuje na základě změn nebo oprav, aby se odhalily chyby vzniklé v důsledku těchto zásahů.“ (Institute of Electrical and Electronics Engineers *et al.*, 2008, p. 95)

Na rozdíl od retestování, pokrývá regresní testování všechny podstatné funkcionality, a to z toho důvodu, aby se zjistilo, že změny nijak nenarušily stávající funkcionalitu. Důležitost tohoto testování vzrůstá z pohledu času s přibývající funkcionalitou. Je zřejmé, že nová funkcionalita může ovlivňovat funkcionalitu, která již byla implementována v předcházejících cyklech, proto je potřeba vybrat vhodnou sadu regresních testů, kterou je potřeba vždy naplánovat tak, aby doplnila sadu testů nové funkcionality. (Institute of Electrical and Electronics Engineers *et al.*, 2008)

Regresní testy mohou být realizovány v různých úrovních testování. Své místo mají v testování jednotek a komponent – každé další spuštění kódu je vlastně regresním testem, zároveň dochází k automatickému spouštění testů v rámci kontinuální integrace. V rámci systémového testování se jedná o testovací případy nejkritičtějších funkcionalit. Uplatňují se v rámci integrací i akceptačních testů. Obecně platí, že regresní testování by mělo alespoň

z části podléhat automatizaci, protože se jedná o testy, které se opakují v každém cyklu. (Bureš *et al.*, 2016)

Techniky regresního testování

Zajištění vhodné sady regresních testů je zásadní podmínkou pro zajištění kvality v oblasti stávajících funkcionalit vyvíjeného software. Existuje několik cest, jak takovou sadu identifikovat.

Jsou-li provedeny veškeré existující testy v jednom běhu, hovoříme o regresním testu všech funkcionalit. Tato technika je ze své podstaty obtížně realizovatelná a nákladná, protože vyžaduje velké množství času a zdrojů. Teoreticky může být vhodná v situacích, kdy existuje jednoduchý software s minimem funkcionalit.

Další možností je regresní testovací sada, kdy se vyberou se takové testovací případy, které ověřují dopady změn vyvolaných modifikovaným kódem.

Posledním přístupem je volba takových testovacích scénářů, které ověřují kvalitu vybraných funkcionalit s ohledem na jejich kritičnost. Provedou se takové testy, které jsou z pohledu obchodních požadavků nejpodstatnější. Výběr takové sady podstatně redukuje množství testů a z toho vyplývající pracnost testování. (Hamilton, 2020b)

3.8.2.5 Výkonnostní testování

Jde o typ testu nefunkčního testování, který uvádím kvůli rozsahu v samostatné podkapitole. Z názvu výkonnostní je patrné, že podstatou je posouzení dostatečnosti nebo nedostatečnosti výkonnosti vyvíjeného software. Základní otázka tedy zní, kdy je aplikace dostatečně výkonná? Molyneaux (2014) naznačuje, že odpověď je vždy ve vnímání. Z praxe, kterou má se zákazníky je to taková výkonnost, kdy lze provádět činnosti bez zbytečného zpoždění, bez negativních pocitů, ideálně tak, aby nevznikaly prostoje a nebyla zbytečně rozptylována pozornost. Výkonnostní testování ověřuje naplnění nefunkčních požadavků. V souvislosti s tím se hovoří o indikátorech, které pomáhají výkonnost posuzovat.

Indikátory pro posouzení výkonnosti

Pro posouzení výkonnosti se hodnotí následující ukazatele

- Dostupnost

Množství času, kdy je aplikace dostupná koncovému uživateli. I malý výpadek může pro kritické aplikace způsobit ztráty ve formě dodatečných nákladů.

- Odezva

Doba nebo množství času, kterou software potřebuje pro reakci na uživatelský požadavek. Z pohledu výkonnostního testování je tento čas měřený jako rozdíl mezi časem, kdy uživatel zadal požadavek a časem, kdy aplikace zareagovala a vrátila uživateli požadovaná data.

- Propustnost

Množství dat, které se v určitý časový úsek přenese na infrastruktuře jako odpověď na požadavek klienta. Může se počítat jako požadavky za sekundu, návštěvy za sekundu apod.

- Využití

Obvykle se vyjadřuje jako procento využití kapacity zdroje. Zdrojem může být procesorová kapacita, kapacita paměti nebo šířka pásma sítě, případně disková kapacita apod.

- Ostatní indikátory

V literatuře se lze setkat s dalšími indikátory. Thakar (2019) navíc uvádí chybovost (množství chyb generovaných během testování) nebo konkurenční uživatele (množství konkurenčních uživatelů, kteří využívají server v určitý čas. Většinu indikátorů lze však vždy zařadit mezi 4 základní, které jsou vyjmenovány výše.

Typy výkonnostního testování

Existuje celá řada testů, kterými můžeme posuzovat výkon. Nejčastěji využívaným typem výkonnostního testování je tzv. zátěžové (Load) testování. Sledovaný software je vytížen paralelními dotazy konkurenčních uživatelů a následně se posuzuje obecné chování, odezva nebo propustnost. Cílem je naplnit stanovené nefunkční požadavky, mezi které může patřit např. maximální čas, do kdy musí software zareagovat a vrátit požadovaná data.

Pokud je třeba ověřit, jak software reaguje při velkém zatížení a překročení navržených limitů, využívá se stres (Stress) testování. V tomto případě se sleduje, kdy systém přestane vykonávat požadované operace a při jaké zátěži ji dojde k selhání. Hledá se tzv. bod zvratu, který říká, že při dané zátěži již systém přestává plnit navržené nefunkční požadavky.

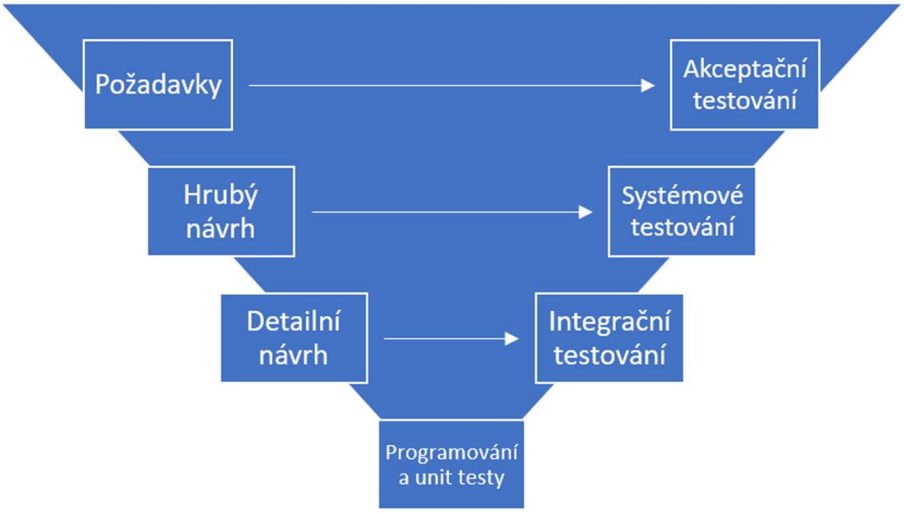
Při testování škálovatelnosti se postupně zvyšují zdroje, které jsou software přiděleny a následně se zjišťuje, jak dobře software škáluje dle přidělených zdrojů. Test může sloužit k následnému rozhodnutí, jak efektivně plánovat infrastrukturu, na které software poběží.

Testování špiček ověřuje, jak dobře se software vypořádá s náhlými nárůsty provozu. Test je modelován pro náhlý nárazový provoz. Reálně může jít o přípravu na marketingovou kampaň nebo plánovanou událost nebo funkcionalitu, která s novou verzí software může přijít na trh. (Cohnen, 2021)

3.8.3 Úrovně testování

Testování aplikace se provádí na různých úrovních vyvíjeného aplikačního systému. Postupuje se od jednotkové úrovně k testování integrace komponent, přes systémové testy až k testům akceptace. Každá úroveň testování sleduje odlišný cíl, celek pak vytváří komplex, který zvyšuje efektivitu testování (Naik and Tripathy, 2008)

V souvislosti s úrovní testování hovoříme o tzv. V-modelu. Jistým způsobem se jedná o variaci vodopádového životního cyklu vývoje software. Rozdíl je v tom, že ke každému milníku náleží jedna z úrovní testování. (Bureš *et al.*, 2016)



Obrázek 12 - V model

Zdroj: Bureš et al. (2016), zpracování vlastní

3.8.3.1 Unit / Component

Testování jednotek se zabývá ověřování částí kódu, procedur, metod a funkcionalit. Obvykle ho zastávají samotní vývojáři, kteří kód vyvíjejí. Někdy v souvislosti s jednotkovým testováním hovoříme o programování řízené testy, kdy jako první vznikají jednoduché testy a na základě již vyvinutých testů se implementuje kód.

3.8.3.2 Integrační testování

Integrační testování ověřuje rozhraní komponent vyvíjeného systému. Cílem je otestovat tok dat mezi závislými moduly a vyhodnotit správnost jejich komunikace. Následuje po testování jednotek a jejich úspěch je podmínkou pro úspěšný start Systémových testů

3.8.3.3 Systémové testování

Systémová úroveň obsahuje celou řadu funkčních testů, testování zátěže, testování zabezpečení a spolehlivosti. Jedná se o poslední úroveň testování, kterou obvykle provádí tým na straně dodavatele software. Z pohledu pracnosti se jedná o nejsložitější část, která obvykle zabírá v exekuci testů nejvíce času.

3.8.3.4 Akceptační testování

Zákazník provádí své vlastní testování poté, co byl systém úspěšně otestován zhotovitelem aplikace. Ověřuje splnění smluvených akceptačních kritérií stanovených požadavků a posuzuje výslednou kvalitu produktu. (Naik and Tripathy, 2008)

3.9 Testování podle způsobu realizace

Testování lze svěřit lidským zdrojům, které test provedou a vyhodnotí, nebo stroji, který je naprogramován tak, aby byl schopen provést stejnou práci bez lidského zásahu. Realizaci testování lze proto rozdělit podle toho, zda test provádí uživatel nebo předem definovaný program.

3.9.1 Manuální testování

Testování probíhá ručním procházením vyvinuté aplikace za použití předem definovaných Testovacích případů nebo méně sofistikovanými metodami více či méně náhodného průchodu. V projektech obvykle tento způsob převažuje, protože počáteční investice jsou ve srovnání s automatickým testováním relativně nízké.

3.9.2 Automatizované testování

Již z názvu je patrné, že testování vykonává s podporou testera speciální software, který je určen právě pro tento způsob testování. Testy vznikají za použití record and replay nástrojů nebo se upravují již vygenerované skripty. Obrovskou výhodou je snížení nákladů na údržbu systému, protože se automatizují testy, které se opakují (regresní testy). Dochází k úspoře času, protože vykonání testu trvá násobně nižší čas oproti manuálnímu způsobu. Dalšími benefity může být zvýšení účinnosti některých testů, protože můžeme použít mnohem větší množství testovacích dat. Teorie nám říká, že jakékoliv úlohy, které se během testování opakují se vyplatí automatizovat, nicméně praxe ukazuje, že údržba testů je přímo závislá na změnách, které v softwaru probíhají (změna API vyžaduje úpravu testu), proto by úvaha o zavedení automatizace měla obsahovat i časové hledisko, protože návratnost investice do automatizace přichází až ve střednědobém nebo dlouhodobém horizontu. (Roudenský and Havlíčková, 2013)

3.10 Nástroje

Na úvod této podkapitoly je potřeba podtrhnout, že nástroje, které se pro podporu testování používají nejsou zárukou úspěchu. Žádný nástroj sám o sobě nemůže zajistit, že testování dopadne dobře, nicméně mohou výrazně usnadnit práci a pomoci dosáhnout definovaných cílů. Jednou z prerekvizit každého projektu je rozumné zvážení, které nástroje je vhodné pro podporu testování zvolit. (Crispin and Gregory, 2009)

Před tím, než rozebereme vlastnosti a funkce jednotlivých nástrojů, měli bychom si odpovědět na otázku, jakou metodikou vývoje software je projekt, pro který je nástrojová podpora zvolena, řízen a jaké testování podle způsobu realizace bude zvoleno. Část nástrojů je použitelných ve všech metodikách i realizacích – např. systém pro evidenci chyb, některé nástroje jsou však poplatné více agilní metodice - typicky nástroje pro řízení CI Pipelines

apod. Níže uvedené dělení jistě není konečné, nicméně pro účely této práce je zmíněna právě tato množina, protože pokrývá podstatnou část činností během testování.

3.10.1 Nástroje podporující řízení testování

Tyto nástroje pomáhají řídit testy, zjednodušují jejich spouštění, vykonávání a následné vyhodnocení. Můžeme je chápat jako jednotné centrální úložiště všech testovacích případů, které jsou následně organizovány do testovacích případů dle potřeb projektu. Dle konkrétní implementace se liší způsoby, jakým testování pojímají a dále pak dalšími vlastnostmi, které můžou jednotlivé role v testování využít.

Jak bylo z míněno výše, základem jsou testovací případy, testovací scénáře, které je možné libovolně přepoužívat v rámci testovacích cyklů a dle potřeb je dále hierarchizovat. (Mckinley, 2022)

Čím se zásadně odlišují jednotlivé implementace je množství doprovodných funkcionalit, modulů a rozšíření – evidence chyb, mapování na požadavky, test plán, API integrace, kvalita reportů atd. Bohatost těchto funkcionalit je logicky vykoupena náklady na pořízení. (Software Testing Help, 2023b)

3.10.2 Nástroje podporující evidenci chyb

Nalezení a identifikace chyby je podstatná část procesu testování. Pokud se testerovi podaří chybu zachytit, měl by následovat další krok a tím je nutnost chybu správně zaevidovat, předat do vývoje a její opravu následně retestovat. Pokud dokáže tester chybu identifikovat, měl by ji zaevidovat do nástroje nebo systému pro sledování chyb. Základním stavebním kamenem těchto nástrojů je tzv. workflow, kdy lze definovat stavy chyb, jejich přechody vč. vzájemných návazností. Toto celé může být automatizováno vč. přidělování relevantním příjemcům. Nástroje tohoto typu podstatným způsob zefektivňují proces testování a mají následující výhody.

3.10.2.1 Přehled o kvalitě

Obvyklým cílem projektu je dodání kvalitního softwarového řešení ideálně bez funkčních závad. Pokud projekt ví, jaké chyby ve vyvíjeném software existují, v jakém jsou stavu, jaký dopad (v negativním smyslu) mají do kvality řešení, jakou naléhavost

je nutné zvolit pro jejich řešení, má z poloviny vyhráno, protože může zvolit správné strategie pro jejich řešení. Nástroje podporující evidenci chyb existují právě z toho důvodu. Poskytují jednoznačné odpovědi na otázky, v jakém stavu je aktuálně vyvíjený software, jaké chyby jsou opravené, jaké zbývá opravit, kolik chyb je aktuálně otevřených a kolik se jich podařilo uzavřít. Kromě projektu poskytují přehled i všem ostatním stakeholderům

3.10.2.2 Zlepšuje týmovou spolupráci

Usnadňuje práci členům týmů od testerů po vývojáře. Všichni zúčastnění jsou notifikováni o jakýchkoliv změnách, které se při řešení chyb podařilo dosáhnout. Není nutné se složitě doptávat, v jakém stavu chyby jsou a zda jsou již opravené či nikoliv. Projektový manažer může díky přehledu o chybách jednodušeji řídit projektový tým.

3.10.2.3 Reporting

Nahlášené chyby zůstávají v evidenci i po jejich vyřešení. Lze se k nim zpětně vracet a provádět doprovodné analýzy – např. pracnost, která s odbavením chyb souvisela v jednotlivých fázích vývojové cyklu nebo trendy, kdy se do software zaneslo více chyb apod.

3.10.2.4 Uživatelská zkušenost

Soudobé nástroje jsou již tak uživatelsky přívětivé, že je možné je integrovat i do stávajících aplikací, kdy mohou dokonce koncoví uživatelé hlásit nalezené chyby z prostředí aplikace. (Sharma, 2019)

3.10.3 Nástroje podporující výkonnostní testování

Na trhu existuje celá řada nástrojů, která podporují tento typ testování. Je potřeba vzít v úvahu náklady na pořízení licencí, požadavky na hardware, podporované typů protokolů, podporu a zejména požadavky projektu, který tento nástroj bude používat.

Nástroje tohoto typu se primárně zaměřují na testování zátěže – typicky simulují množství uživatelů, kteří mohou vyvíjenou aplikaci používat v určitý časový úsek. Testování s použitím těchto nástrojů se může zaměřovat na vytížení backendové nebo frontendové části aplikace. Zároveň existují nástroje, které ověřují zátěž i na mobilních platformách.

Na trhu existuje celá řada řešení, které mají své výhody a nevýhody. Mezi ně může patřit uživatelská přívětivost, nutnost znalosti programování, možnosti analýzy dat a výsledků testování možnosti skriptování, všeobecná rozšířenost, komunita apod. (Galeza, 2022)

3.10.4 Nástroje podporující monitoring a logování

Aby bylo možné řešit problémy, které souvisejí s nefunkčností vyvíjeného software efektivně a systematicky, dává smysl využít takové nástroje, které pomáhají s analýzou logů. Díky nástrojům tohoto typu je možné získat ze surových dat takové informace, které umožní určit příčinu chyby, která se v software vyskytuje. Zároveň je možné jít ještě dál a získat i trendy a patterny, které mohou zjednodušit další rozhodování.

Nástroje umožňují nahrávání, vyhledávání, filtrování a analyzování logů na různých zařízení v reálném čase. Řada z nich poskytuje informace, které umožňují porovnat log s využitím infrastruktury. Umožňují též tvorbu dashboardů pro lepší vizualizaci nebo nastavení alertů na logovaná data. (Kuć, 2023)

3.11 Role v testování

S přibývající komplexitou a velikostí projektů vyvstala potřeba specializace rolí, které zajišťují testování. Tam, kde v minulosti bylo potřeba jednoho člověka, který testování odřídil, odbavil analytickou část i část exekuční jsou dnes potřeba specifické role, které jednotlivé aspekty testování pokryjí.

V praxi se setkáváme s různým označením rolí, které za testování odpovídají. Některé zdroje zmiňují role test architekt (Page, Johnston and Rollison, 2009) nebo test engineer (Sheremetov, 2022), nicméně pro účely této práce budeme pracovat s rolemi, které kategorizují Roudenský a Havlíčková (2013).

3.11.1 Tester

Pracovník v roli tester má na starosti spuštění vyvinutých testovacích případů. Provádí jejich kroky a v případě, že naleznete chybu, reportuje její výskyt do předem definovaného nástroje pro správu chyb. Po opravách provádí retesty a v případě potřeby reportuje chyby nadřízenému.

3.11.2 Analytik testování

Analýzou funkčních i nefunkčních požadavků navrhuje testovací případy, které dále kategorizuje a prostřednictvím doplnění všech náležitostí systematicky dotváří. Jeho úkolem je projít funkční specifikaci ve formě případů užití, UML diagramů, návrhů obrazovek a dalších specifikací a pokrýt tak tyto vstupy relevantními testy. Logicky pak musí všechny testy prioritizovat s přihlédnutím k omezením ve formě rozpočtu a času.

3.11.3 Manažer testování

Stejně jako projektový manažer je odpovědný za řízení projektu, je manažer testování (též nazývaný jako test leader) odpovědný za řízení testování. Jelikož se jedná o řídící roli, odpovídá se přímo právě projektovému manažerovi, reportuje mu výstupy z testování a finálně odpovídá za navržený testovací plán a další dokumenty související s testováním. Má na starosti cíle testování, stanovení akceptačních kritérií a následné hodnocení jejich naplnění. Koordinuje a řídí testovací tým, podílí se na složení testovacího týmu, vyjednává za testovací tým se všemi stakeholdery vč. zákazníka.

3.11.4 Porovnání kompetencí rolí

V mnoha projektech bývají role testera a analytika testování sloučeny a vykonává je jeden pracovník. To zpravidla platí pro menší projekty, kde je požadavek na minimalizaci nákladů. Jednotlivé kompetence tak mohou mezi rolemi přecházet dle aktuálních potřeb. Manažer testování může v určitých situacích převzít část kompetencí analytika testování, vše závisí na aktuálních potřebách projektu. Rozdělení kompetencí je proto potřeba brát jako doporučení, nikoliv jako dogma. (Roudenský and Havlíčková, 2013)

Kompetence	Role		
	Tester	Analytik testování	Manažer testování
Návrh a vytvoření testů		✓	
Provádění testů	✓		
Reporting			✓
Řízení testování			✓
Řízení testovacího týmu			✓
Vytváření chyb	✓		

Tabulka 2 - Porovnání kompetencí rolí

Zdroj: Roudenský and Havlíčková (2013), zpracování vlastní

3.12 Řízení testování

Řízení testování zahrnuje dle Institute of Electrical and Electronics Engineers (2008) celou řadu aktivit od sledování a hodnocení plánů spouštění testů, analýzu odchylek oproti testovacímu plánu, reportování pokroku v rámci dílčích procesů testování, hodnocení skutečných výsledků oproti očekávání k ověření, zda jsou úkoly dokončené a výsledky úplné.

V kontextu řízení projektů má řízení testování své neoddiskutovatelné místo, protože podává jasné informace dovnitř a vně projektu o stavu dodávky, pomáhá reagovat na situaci, která ovlivňuje vyvíjený produkt a naznačuje možné důsledky, které plynou z učiněných rozhodnutí. V roli test managera vystupuje osoba s jasnou kompetencí a odpovědností, která zastřešuje testování jako celek, komunikuje se zadavateli a ostatními stakeholdery projektu. Zároveň vede testovací tým a určuje strategii, jakou se testování bude v intencích projektu ubírat. (Pinkster *et al.*, 2004)

Řízení testování obsahuje celou řadu dílčích aktivit, které podporují projekt po celou dobu jeho vývojového cyklu.

3.12.1 Risk analýza a testovací strategie

Během projektu musí manažer testování uvažovat oba typy rizik – produktová i projektová. Protože připravuje test plán jako základní dokument, který zastřešuje testování, musí identifikovaná rizika včlenit do plánu a nastavit takovou strategii, která daná rizika bude minimalizovat. Strategie testování jako prerekvizita a součást testovacího plánu vymezuje jaká rizika jsou pokryta, jakými testy. Testovací plán, který vzniká v této fázi by měl být v souladu se Strategií testování.

3.12.2 Naplánovat testování

Plánování úsilí, které bude testování věnováno je ovlivněno charakterem projektu, formou dodávky a metodikou vývoje software. Vstupem plánování je analytická a vývojová dokumentace ve formě požadavků, technických a funkčních specifikací a projektový plán.

Součástí této fáze je odhadování pracnosti testování. Existuje celá řada metod, jak odhadování koncipovat, přičemž nelze říct, že některá cesta je správnější než jiná. Vše závisí na okolnostech a vstupech, které lze pro odhadování použít. Pokud v minulosti existoval nějaký projekt, který se podobá tomu současnému můžeme extrapolovat odhady na základě minulé zkušenosti (Analogie). Podobný způsob (Shora dolů) je používán zkušenými projektovými manažery, kteří přihlížejí k předchozí zkušenosti a z ní predikují předpokládané náklady a trvání na větší celky a poté odhady zpřesňují u podřízených aktivit. Pokud existuje skutečně podrobná analýza, detailně popsané úkoly, můžeme použít techniku, kdy tyto dílčí úkoly relativně snadno odhadneme a výsledné číslo dá celkový odhad (Zdola nahoru). Chybí-li jakékoliv informace na základě kterých by odhad mohl být stanoven, můžeme testování odhadnout na základě trvání projektu nebo vývojové fáze (Procenta). Existuje celá řada dalších metod, která je použitelná – odhadování na základě

story pointů, případů užití atd. Logicky lze tyto metody kombinovat a dosahovat tak vyšší míry přesnosti.

Výstupem plánování a odhadování je následně obohacený test plán o časové hledisko ve formě harmonogramů, zdrojů testování a odpovědností, dále pak jak bude testování organizováno. (Pinkster *et al.*, 2004)

3.12.3 Navrhnout testy

„Návrh testů (test design) je dokumentace, která specifikuje podrobnosti přístupu k testování funkcionalit software nebo jeho kombinací funkcionalit a identifikace souvisejících testů.“ (Institute of Electrical and Electronics Engineers *et al.*, 2008, p. 11)

V této fázi dochází k k identifikaci podmínek testování, testovacích případů a testovacích dat. Je potřeba rozhodnout, které techniky testování budou použity, jaké způsoby realizace a jak budou navržené případy podporovat úrovně testování. Abychom si byli jisti, že jsme testování zaměřili správně, musíme ověřit pokrytí testů tak, že pro každou funkcionalitu zvažujeme testování s přihlédnutím k závažnosti a dopadu funkcionality na výsledný produkt. (Pham, 2017)

3.12.4 Vyvinout testy

Pod pojmem vyvinout testy se rozumí aktivita, kdy dochází k finálním dopracování navržených testovacích případů, zajištění vhodných testovacích dat a konfiguraci testovacích prostředí. Výstupem je aktualizovaný test plán, vyvinuté testovací případy a nakonfigurované testovací prostředí.

Pokud existuje pro daný testovací případ pouze jediná vstupní hodnota, je možné ke každému z kroků testovacího případu tuto hodnotu doplnit. Velmi často však potřebujeme sadu testovacích dat, protože je nezbytné ošetřit všechny možné testovací hodnoty. V tomto

případě přistupujeme k datům jako k proměnné, přičemž každá proměnná je definována jako sada testovacích dat.

Pro správně navržený testovací případ je potřeba jednoznačně stanovit, kdy je test považovaný za úspěšný a kdy nikoliv, jaké podmínky musí být naplněny pro to, aby mohl být test spustitelný. Testovací případy se doplní o testovací data a sadu kroků, které ověří správnost výstupů testovaného systému.

Pro správné otestování funkcionalit vyvíjeného software je potřeba zajistit testovací prostředí, může se jednat o takové instance prostředí, které budou podporovat různé úrovně testování. Pro Unit / Component testování se nepředpokládá vysoká míra integrace, protože je potřeba testovat pouze vyvíjený kód, ale pro Akceptační testování by se prostředí co možná nejvíce mělo podobat finálnímu produkčnímu prostředí vč. integrací. (Lewis, Dobbs and Veerapillai, 2009)

3.12.5 Provést testy

Testování je spuštěno a jsou nacházeny chyby. Pro reportování chyb se používají předem definované nástroje, které slouží k jejich evidenci a mnohdy i k evidenci požadavků. Chyby jsou předávány a odbavovány dle dohodnutých priorit v rámci definovaného workflow, které usnadňuje jednoznačnou identifikaci, kdo je za řešení chyby odpovědný. Workflow pomáhá určit v jaké fázi se chyba nachází a komu byla přidělena. Testování je odpovědné za ověření, že opravy nalezených chyb jsou v souladu s požadavky.

Během této fáze se pravidelně ověřuje, které testy byly spuštěny a s jakým výsledkem. Pravidelně se vyhodnocuje množství nalézaných chyb. Pokud je množství nalézaných chyb konstantní, předpokládá se, že se v aplikaci nachází stále velké množství chyb. S tím, jak se snižuje záchyt chyb (pokud jsou správně zaměřené testy), existuje předpoklad, že se množství chyb v aplikaci snižuje.

Výstupem této fáze je aktualizovaný test plán a další reporty z testování – seznam spuštěných testovacích případů, seznam známých chyb apod. (Naik and Tripathy, 2008)

3.12.6 Vyhodnotit testování

V poslední fázi vypracuje test manager závěrečnou zprávu, která by měla obsahovat následující body:

- Porovnání deklarovaných cílů testování se skutečně dosaženými cíli. Porovnání následně pomůže určit, jaké cíle byly splněny zcela, jaké byly splněny částečně a jaké nikoliv.
- Informace o dosažené kvalitě vyvinutého software. Jsou zmíněna produktová rizika a jakými úrovni testů byla pokryta, zda byla splněna akceptační kritéria a jaká rizika byla případně nevyřešena s návrhem, jakým způsobem mohou být v následujících fázích dále zmírněna.
- Náklady vyložené na jednotlivé fáze testování.
- Kvantifikace přínosů testování.
- Přehled všech dodávek, které testování zajistilo
- Doporučení ohledně nasazení software do produkčního prostředí vč. počátečního provozu

Zprávu následně předloží ke schválení roli projektový manager. (Pinkster *et al.*, 2004)

4 Vlastní práce

4.1 Projekt

4.1.1 Popis a cíle projektu

Projekt, který jsem podrobil analýze v této bakalářské práci se zabývá sledováním a vyhodnocením emisí CO2 v nejmenované automobilce. Problematika CO2 je v dnešní době aktuální, protože emise tohoto a dalších skleníkových plynů jsou pod drobnohledem vlád a orgánů Evropské unie. Automobilky jsou povinny dodržovat platné limity emisí CO2, které se v průběhu let postupně snižují, a proto musí mít dokonalý přehled o produkovaných emisích na úrovni svých flotil. V případě, že automobilka nesplní zákonné limity, může být penalizována ve formě pokut. Protože automobilky obvykle vyrábí více druhů / tříd vozidel, stojí před rozhodnutím, jakým způsobem plánovat svůj výrobní program a strategii prodeje s přihlédnutím k maržím a potenciálním pokutám.

Cílem projektu je dodávka softwarového řešení pro krátkodobé řízení emisí CO2, které automobilce umožní získat následující přehled:

1. množství prodaných a plánovaných vozidlech za sledované období kalendářního roku
2. technické parametry pro každé vyrobené a plánované vozidlo – CO2 emise, hmotnost vozidla
3. prodejních indikátory – náklady na výrobu, marže, prodejní cena za každý automobil
4. pokuta za každý plánovaný a vyrobený automobil vč. celkové pokuty za celou flotilu vozidel

Řešení dále umožní provádět takové změny v plánovaných prodejích, které dovolí sledovat dopady do výsledné pokuty. Mezi takové změny patří následující:

1. Úprava objemů ve výrobním programu pro plánované měsíce
2. Úprava technických parametrů v případě, že v důsledku modernizace výrobního programu dojde ke změnám hmotnosti nebo vypouštěných emisí CO2 na konkrétním vozidle.

4.1.2 Technologie

Aplikace je realizovaná třívrstvým architektonickým modelem. Prezenční vrstva běží ve webovém prohlížeči a je implementovaná s využitím knihoven REACT. Aplikační vrstva je implementovaná v jazyce JAVA, využívá rozhraní REST API pro komunikaci s prezenční vrstvou, s datovou vrstvou pak komunikuje prostřednictvím MDX dotazů. Datová vrstva, která se stará o uložení a procesování dat je implementovaná v nástroji IBM Planning Analytics. Na rozdíl od standardních relační databáze, používá IBM Planning Analytics datové struktury ve formě dimenzí a kostek, které pak tvoří multidimenzionální prostor, ve kterém je možné data analyzovat z různých úhlů pohledu.

4.1.3 Organizace projektu

Zadavatel (automobilka) vystupuje v projektu jako zákazník a delegovala svou odpovědnost na svého zaměstnance, který v projektu zastává roli Product owner. Ten definuje požadavky a jejich prioritu v souladu s přáním zákazníka, průběžně komunikuje s dodavatelem.

Dodavatel má na starosti kompletní cyklus vývoje software. Tým tvoří cross-funkční tým, kde každý člen týmu má přidělené specifické role a odpovědnosti:

1. Projektový manažer – řídí tým a komunikuje se zadavatelem
2. Analytický tým – připravuje analytické podklady pro vývojový a testovací tým. Tým tvoří 2 analytici
3. Front-end tým – vyvíjí prezenční vrstvu. Tým tvoří 3 vývojáři
4. Middleware tým – vyvíjí aplikační vrstvu. Tým tvoří 1 vývojář
5. Datový rým – vyvíjí datovou vrstvu. Tým tvoří 3 vývojáři
6. Testovací tým – připravuje a vykonává testy. Tým tvoří 3 testeři

Projekt si zvolil agilní přístup k řízení projektu. Převzal některé postupy z metodiky SCRUM – vývoj probíhá ve sprintech, probíhají některé ceremonie obvyklé pro agilní vývoj – stand-up, sprint planning, sprint review, demo. Analýza požadavků probíhá s předstihem min. jednoho sprintu. Existuje backlog s požadavky, které jsou prioritizovány rolí Product Owner a v souladu s odhady pracnosti za každý požadavek následně plánovány

do implementačních sprintů. Součástí sprintu je samotný vývoj, testování v testovacím prostředí, demo a následné nasazení do produkčního prostředí.

4.2 Současný stav testování v projektu

4.2.1 Testovací tým

Testovací tým tvoří role test leader. V rámci svých odpovědností má na starosti:

- Řízení týmu testerů
- Přidělování úkolů a dohlížení na jejich plnění
- Koordinaci nalezených chyb
- Plánování, exekuci a vyhodnocení testování
- Komunikaci za tým směrem k projektu a zákazníkovi
- Další úkoly, které zastává role test analytik

Zbytek testovacího týmu z role test analytik, kteří vykonávají též roli tester. Jejich náplní práce je:

- Spolupráce při odhadování pracnosti testování
- Vytváření testovacích případů a testovacích dat
- Spouštění testů
- Zadávání chyb a retest oprav

Testovací tým pracuje společně v otevřené kanceláři a sdílí prostor i s vývojovým týmem. Práce z domova je umožněna, reálně je poměr práce z domova a v otevřené kanceláři 50:50.

4.2.2 Nástroje

4.2.2.1 Confluence

Platforma od společnosti Attlassian se v projektu používá pro správu požadavků. Jedná se o webovou aplikaci, která umožňuje uživatelům vytvářet a sdílet veškeré dokumenty a zobrazovat je formou wiki stránek.

Na projektu slouží tento nástroj jako jednotné místo pro uložení požadavků. Hierarchická struktura pak umožňuje jejich zobrazení v návaznosti na proběhlé či plánované sprinty. Součástí analyzovaných požadavků je funkční i technická specifikace, kolaborativní

vlastnost nástroje pak umožňuje případné komentáře všech členů týmu. Prostředí nástroje umožňuje pohodlné vyhledávání, označování příspěvků a jejich verzování. Nespornou výhodou celého řešení je trvalý a snadný přístup ke všem informacím, které tým potřebuje a dále pak integrace na další nástroj od stejné společnosti - JIRA

4.2.2.2 JIRA (Zephyr plugin)

Hlavním účelem nástroje je umožnění organizace, sledování a řízení celého projektu. Nástroj pracuje s úlohami, kterým lze nadefinovat workflow pro řízení jejich životního cyklu (dle typu úlohy). Workflow specifikuje, jakým způsobem se úlohy pohybují od vytvoření až po uzavření – má jasně definované stavy a přechody. Určuje tak, který tým nebo osoba má právo na úpravu, jaké jsou potřebné akce pro dokončení a kdo musí akci provést. Pro potřeby projektu na řízení CO2 emisí se pracuje s následujícími typy úloh:

1. User Story – reprezentuje požadavek, jehož přesné znění a popis je uložený v nástroji Confluence.
2. Sub-task – reprezentuje dílčí úlohu User Story, které je podřízen. Sub-tasky jsou organizovány dle funkčních týmů, obvykle tak existuje min. jeden vývojový Sub-task navázaný na jeden z vývojových týmů a jeden „testovací“ Sub-task, který je navázaný na testovací tým.
3. Nested-bug – reprezentuje chybu, která byla nalezena ve funkcionalitě implementované User Story a z toho důvodu je dané User Story podřízena.
4. Bug – reprezentuje chybu, která byla nalezena mimo funkcionalitu nově implementovaných User Stories, nejčastěji během regresního testování. Hierarchicky je na stejné úrovni jako User Story.

Testovací tým kromě výše uvedených úloh používá funkcionality rozšíření s názvem Zephyr. Tento plugin poskytuje robustní a rozsáhlé funkce pro správu testování software. Umožňuje vytvářet, plánovat, spouštět manuální testovací scénáře (specifická forma úlohy) a následně je vyhodnocovat a poskytovat tak podklady vedení projektu formou reportů a online dashboardů. Testovací scénáře jsou plně v režii testovacího týmu a podporují úrovně systémového a akceptačního testování. Díky možnosti vazeb na User Stories je pak možné sledovat pokrytí testů.

Z pohledu použití zastává aplikace JIRA nástroj podporující řízení testování a nástroj podporující evidenci chyb. Jeho nespornou výhodou, že obě tyto funkcionality jsou úzce prointegrované.

4.2.2.3 SoapUI

Pro potřeby integračního testování byl vybrán nástroj SoapUI od společnosti Smartbear. Nástroj podporuje testování webových služeb různých rozhraní vč. REST, které je použito ve vyvíjené aplikaci. Testovací scénáře, které vznikají v tomto nástroji se zaměřují primárně na ověření integrace prezenční a aplikační vrstvy. Tyto vrstvy si mezi sebou vyměňují zprávy v JSON formátu, které jsou pak v nástroji kontrolovány prostřednictvím vestavěných assertion, což je mechanismus pro ověřování, zda odpověď na požadavek splňuje očekávání. Pro vytváření složitějších testů se používají Groovy skripty, které umožňují sofistikovanější vícekrokové scénáře vč. generování dat.

Testy jsou organizovány v kolekcích a spouštěny vždy po nasazení nových verzí aplikace. Protože běží v poloautomatickém režimu, lze dosažené výsledky analyzovat poměrně rychle bez výrazného zásahu testera.

4.2.2.4 IBM Planning Analytics

Z pohledu testování probíhá v Planning Analytics kromě kontroly uložených dat též kontrola výpočtů dle specifikovaných algoritmů. Aplikace se používá pro kontrolu výsledků během systémového testování.

4.2.3 Proces testování

4.2.3.1 Odhadování pracnosti

Odhadování pracnosti probíhá mimo sprint resp. náročnost požadavků je odhadována v dokončovací fázi analytického procesu. Role test leader odhaduje pracnost ve spolupráci s leadery ostatních vývojových týmů na společné schůzce, kde analytik představuje navrženou User Story. Odhad pracnosti, který je stanoven v člověkodnech vychází ze zkušenosti z předchozích sprintů. Do odhadu se promítá předpokládaná složitost – množství implementovaných formulářů, nově implementované webové služby, dopady do databázové části aplikace. Pro stanovení předpokládané pracnosti se používá tříbodový

odhad, kdy se stanovuje optimistická (nejméně pracná), pesimistická (nejvíce pracná) a realistická varianta. Z tohoto rozpětí se následně vypočte pravděpodobná pracnost. Při odhadování se bere v potaz předpokládaný čas strávený přípravou testovacích scénářů a předpokládaná doba strávená testováním. Odhady jsou následně schvalovány zástupcem zadavatele – rolí Product Owner. V prostředí automobilky jsou odhady považovány za skutečnou pracnost, což v prostředí vývoje s agilními prvky vede k občasným nedorozuměním a následně se musí skutečná pracnost (pokud je vyšší než plánovaná) složitě vysvětlovat.

4.2.3.2 Design a vývoj testovacích scénářů

Jakmile je sprint naplánován a je zřejmé jaké User Stories se budou implementovat, dojde na společné schůzce testovacího týmu k seznámení s implementovanými funkcionalitami. Role test leader seznámí tým s plánem sprintu a všemi podstatnými milníky – předpokládané termíny dokončení vývoje, startu UAT a nasazení aplikace do produkčního prostředí. Současně test leader přerozdělí plánované User Stories do týmu k následné test analýze. Již na této úvodní schůzce mohou vznikat na základě základního rozboru User Stories první připomínky, které si mohou vyžádat úpravy v analytické dokumentaci.

Každý ze členů týmu následně analyzuje přidělené User Stories. Funkční a technická specifikace je podrobena statickým technikám testování, kdy jsou nalézány první chyby nebo podněty k zapracování.

Testovací scénáře pro potřeby integračního automatického testování vznikají v nástroji SoapUI, testovací scénáře pro manuální systémové a akceptační testování v nástroji Zephyr, který je součástí JIRA. V obou případech jde funkční testy. Role test analytik tak kromě nových scénářů pracuje i na regresní sadě testů, kterou upravuje v souladu s požadavky předchozího sprintu. Všechny scénáře, které vyvíjí testovací tým vznikají formou černé skříňky, testování bílé skříňky si zajišťuje vývoj pomocí jednotkových testů.

Testovací tým provádí konzultace ostatními týmy s cílem ověřit, zda jsou testovací scénáře řádně navrženy a vytvořeny.

Na konci fáze jsou tak dopracovány testy, které obsahují cíle testování, podmínky pro jejich spuštění, všechny potřebné kroky a očekávané chování. Testy jsou zařazeny do kolekcí (SoapUI) a sad (Zephyr) a tím jejich vývoj končí.

4.2.3.3 Provedení testů na straně dodavatele

Provádění testů začíná, jakmile je dokončen vývoj alespoň jedné User Story nebo jejího dílčího Sub tasku. Zcela v souladu s agilním přístupem se tak nečeká na dokončení kompletního vývoje všech funkcionalit. Role tester spouští testy a vyhodnocuje jejich výsledky. Pokud test selže, je u konkrétního kroku testovacího scénáře proveden záznam o chybě. K analýze chyb jsou používány logy, které generují všechny vrstvy. Cílem je poskytnou vývoji maximum informací. Do záznamů o chybách se kromě povinných náležitostí přidávají záznamy z logů, kroky k vyvolání stávajícího chování a případně i nahraná videa formou gif videí. Chyby jsou na základě dohod předávány v nástroji JIRA do vývoje a po opravách zpět do testování. Retest chyb probíhá, jakmile jsou opravy nasazeny do testovacího prostředí.

Na základě dohod se zadavatelem je testování na straně dodavatele ukončeno, jakmile jsou dokončeny testy a opraveny a retestovány chyby s nejvyššími prioritami (číselné označení 1 a 2). Chyby s nižší prioritou (3 a 4) by se ideálně měly vypořádat do konce sprintu. Pokud se to nepodaří, jsou reprioritizovány a přesunuty do backlogu nebo je jejich oprava naplánována do dalšího sprintu.

Během některých sprintů je chybovost vývoje větší a proto dochází k prodloužení doby testování. To vede k posunům následujících milníků – předání aplikace do UAT a nasazení aplikace do produkčního prostředí.

4.2.3.4 Provedení testů na straně zadavatele

Uživatelsky akceptační testy probíhají v samém závěru sprintu. Zadavatel si formou free testů ověřuje naplnění akceptačních kritérií, které jsou součástí specifikace User Stories. Testovací tým dodavatele je k dispozici jako podpora pro nalezené chyby nebo jako podpora pro vysvětlení nové funkcionality.

Akceptace User Stories probíhá průběžně podle toho, jak zadavatel stíhá dokončit své testy. Z dosavadních zkušeností byly vždy akceptovány všechny dodané User Stories. V některých případech však s některými výhradami, které byly napraveny v následujících sprintech.

4.2.3.5 Nasazení aplikace do produkčního prostředí

Poslední fáze sprintu, na které participuje i testovací tým. Po nasazení aplikace do produkčního prostředí probíhají smoke testy, které ověřují, že nasazení proběhlo úspěšně. Definovaná sada smoke testů je spouštěna po každém nasazení. Cílem je ověřit kritické funkcionality aplikace. Jakmile je testování ukončeno, informuje pracovník v roli test leader všechny týmy vč. zákazníka o úspěšném nasazení.

4.3 Analýza současného stavu testování v projektu

Pro analýzu stavu testování v projektu jsem porovnal stávající proces testování s procesem, který je definován v odborné literatuře a jehož výstupy jsou součástí teoretické části této bakalářské práce. Samozřejmostí musí být i přihlédnutí ke specifikům projektu. Součástí porovnání bude i návrh opatření.

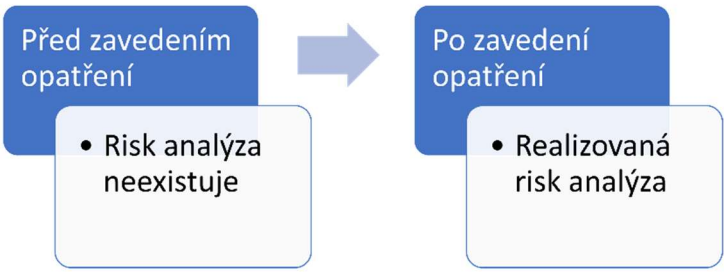
4.3.1 Porovnání stávajícího procesu testování s nejlepšími možnými přístupy

4.3.1.1 Risk analýza

Risk analýza slouží jako vstupní dokument pro plánování a návrh testování software. Na základě identifikovaných rizik a jejich pravděpodobností se rozhoduje, jaká testovací opatření budou použita a jaký bude rozsah a hloubka testování nově implementovaných funkcionalit. Na základě rizik se též určí priority testovacích aktivit a případně se naplánují další opatření pro minimalizaci rizik. Risk analýza by měla být součástí agilního vývoje a měla by být prováděna v rámci každé iterace. Ve sledovaném projektu není risk analýza vytvářena a proto i procesu testování chybí základní vstupy, na základě kterých by mohl dosáhnout větší efektivity.

Návrh opatření

Realizovat risk analýzu po dohodě se zadavatelem jako průběžný proces na začátku každého sprintu. Do analýzy zapojit všechny členy týmu vč. role Product Owner. Umožnit tak adaptaci na nová rizika s každou iterací, kdy vývoj probíhá.



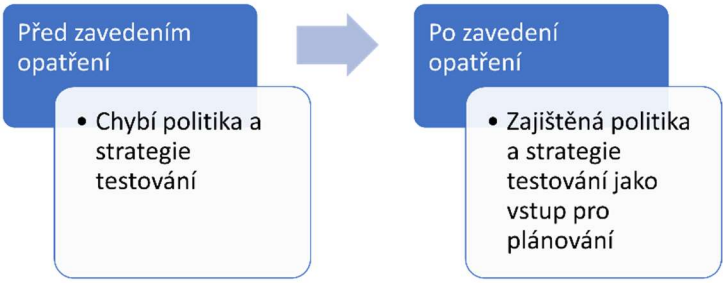
Obrázek 13 - Opatření týkající se risk analýzy

4.3.1.2 Politika a strategie testování

Politika a strategie testování mají výrazný dopad na proces testování v projektu tím, že stanovují směr a cíle pro celý testovací proces a poskytují základ pro rozhodování o konkrétních činnostech a metodách testování. Kvalita politiky a strategie testování výrazným způsobem ovlivňují všechny aktivity, které jsou součástí testovacího procesu. Projekt, který je implementován v prostředí automobilky je ovlivněn chybějící politikou a strategií testování. Na úrovni automobilky je testování roztříštěno mezi různé týmy a není nikterak centralizováno prostřednictvím jasně definované politiky a z ní vyplývající strategie.

Návrh opatření

Ověřit přes roli Product Owner, který je součástí týmu zadavatele, zda v automobilce existují dokumenty, které vymezují politiku a strategii testování. Pokud dokumenty existují, použít je jako vstup pro testovací plán.



Obrázek 14 - Opatření týkající se politiky a strategie testování

4.3.1.3 Plánování testování

Testovací plán jako dokument, který popisuje plánovaný přístup k testování software v projektu není formalizován. Všechny testovací aktivity, které jsou v projektu realizovány se iterativně vylepšují, ale chybí globální přehled a přesah. Plánování testování se v aktuálním nastavení skládá z designu a vývoje testovacích scénářů, jsou specifikovány nástroje podporující testování, testovací prostředí, přístup k řízení chyb i zdroje, které testování zajistí. Je možné, že některé aktivity, které by měly být realizovány, chybí nebo jsou pokryty nedostatečně.

Návrh opatření

Vytvořit testovací plán i přes to, že projekt je již v plném proudu a zpřístupnit ho celému týmu na Confluence. Testovací plán v první iteraci doplnit o stávající proces testování. Provést revizi úrovní testování a pro každou úroveň analyzovat, zda testování rozšířit o další typy testů jako je výkonnostní testování nebo testování uživatelského rozhraní. Případné rozšíření typů testů je nutné komunikovat s rolí Product Owner, protože jejich vývoj a provádění může mít teoreticky dopad do rozpočtu projektu a potřeby dalších testovacích zdrojů. Testovací plán pak udržovat aktuální a upravovat ho každý sprint s ohledem na změny a nové požadavky. Design a vývoj testovacích scénářů přesunout do dalších fází

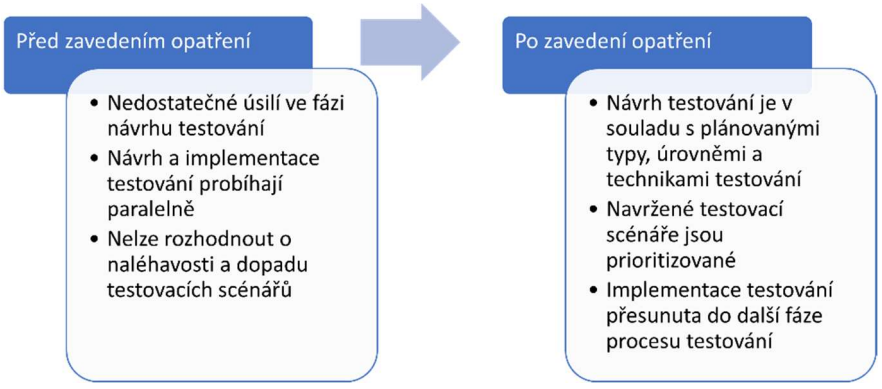
Obrázek 15 – Opatření týkající se plánování testování

4.3.1.4 Návrh testování

Návrh testování probíhá ruku v ruce s jejich vývojem. Protože běží obě aktivity paralelně dochází k opomenutí některých případů a testování tak není efektivně zaměřeno a chyby jsou tak nalezeny až během testování zadavatelem nebo v produkčním prostředí. I když Zephyr plugin umožňuje sledovat pokrytí testů, dochází kvůli neefektivnímu návrhu k nedostatečnému pokrytí všech scénářů a to opětovně vede k neodhalení maximálního množství chyb již během raných fází testování. Zcela chybí prioritizace testovacích scénářů. Během spouštění testů tak není snadné rozhodnout o tom, které testy by z pohledu priority měly být spuštěny jako první.

Návrh opatření

Oddělit návrh a implementaci testů. Návrh testování předřadit implementaci a věnovat tak této fázi zvýšené úsilí. To může pomoci k lepšímu zaměření testů a zefektivnění procesu testování. Již během návrhu testů zvažovat použití vhodných úrovní a technik testování. Součástí návrhů testů musí být jejich správná priorita. Věnovat jednu iteraci zjištění zpětné vazby k navrženým testům od analytického týmu.



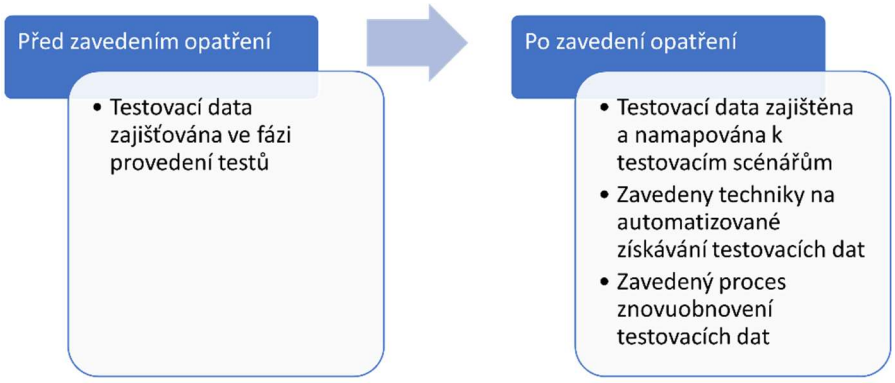
Obrázek 16 - Opatření týkající se návrhu testování

4.3.1.5 Implementace testů

Během vývoje testů se zapomíná na vhodné zajištění testovacích dat. Tyto jsou získávány až během spouštění testů a to vede ke zbytečným prostojům. Mnohdy je velice obtížné vhodná testovací data získat, protože v testovacím prostředí neexistují.

Návrh opatření

Přesunout implementaci testování z části návrhu testování. Zajistit vhodná testovací data již během vývoje testů. Pro jednotlivé testovací scénáře definovat, jaká testovací data musí být použita a prostřednictvím této specifikace pak zajistit vytvoření dat samostatně nebo s podporou vývojového týmu. Zvážit použití nástrojů, které se zabývají generováním testovacích dat. Zvážit techniky, které umožní znovuobnovení již použitých dat – restore prostředí, degradace produkčních dat.



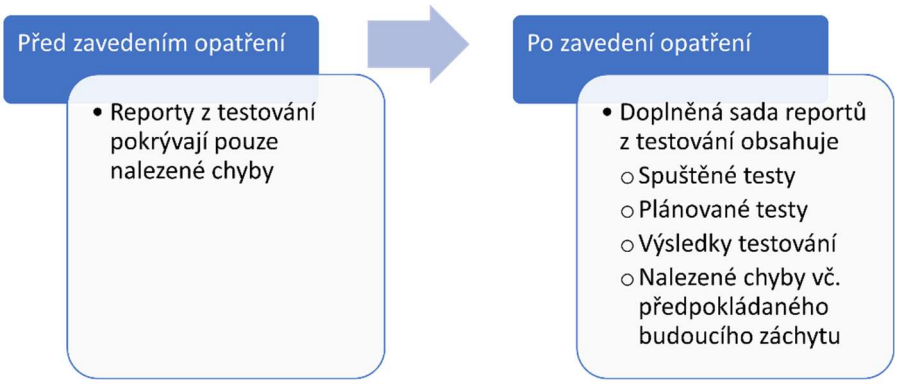
Obrázek 17 - Opatření týkající se implementace testů

4.3.1.6 Provedení testů na straně dodavatele

Ve fázi spouštění testů dochází k nedostatečnému reportování výsledků testování směrem k vedení projektu. Je sice zřejmé, jaké chyby byly odhaleny, ale k celkovému obrazu o tom, jak je testování úspěšné chybí pohled na to, které testy byly spuštěny a s jakým výsledkem, jaké testy mají být ještě spuštěny vč. předpokládaného harmonogramu. Zároveň chybí reporty o záchytu chyb, které mohou implikovat množství ještě nenalezených chyb

Návrh opatření

Vytvořit sadu jednoduchých průběžných reportů, které zajistí projektu jednoznačný pohled na úspěšnost testování – reporty o průběhu testování, reporty o pokrytí testování, reporty o výsledcích testování.



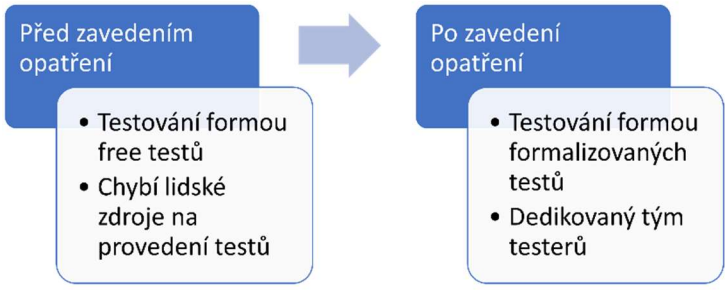
Obrázek 18 - Opatření týkající se provedení testů na straně dodavatele

4.3.1.7 Provedení testů na straně zadavatele

Protože testy probíhají formou free testů, nemusí být akceptační kritéria ověřena správným testováním. Reálně se stává, že se chyba, která projde do produkce mohla odhalit během testování na straně zadavatele, nicméně zadavatel zvolil chybný způsob testování. Zároveň na straně zadavatele chybí lidské zdroje na testování, testy nemá kdo provést a zadavatel tak nemusí ověřit naplnění akceptačních kritérií.

Návrh opatření

Pokusit se zajistit zdroje testování s podporou role Product Owner. Na straně zákazníka existující testovací týmy, je proto možné, že by pro projekt bylo možné získat testovací tým, které by pomohl zajistit akceptační testy. Zároveň bude vhodné formou osvěty pomoci zadavateli ze strany dodavatele s lepší formalizací testů.



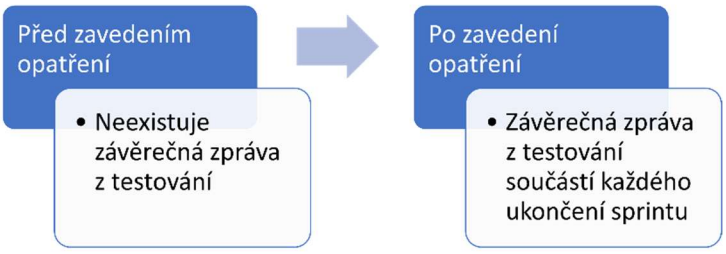
Obrázek 19 - Opatření týkající se provedení testů na straně zadavatele

4.3.1.8 Vyhodnocení testování

Vyhodnocení testování je aktuálně omezeno na vyhodnocení úspěšnosti testování. Na základě nalezených chyb na straně zadavatele dochází k vyhodnocení záchytu chyb z předcházející fáze testování. Zcela chybí zhodnocení výsledků testování na úrovni testovacího a celého týmu. Závěrečná zpráva z testování není požadovaná. S případnými nedostatky se nepracuje systémově – chybí jakékoliv hodnocení zkušenosti, se kterými by se mohlo pracovat s následujících sprintech.

Návrh opatření

Zaměřit se na hodnocení zkušeností. Aktuální zjištění zachytit na Confluence a definovat proces, jakým způsobem se bude se zkušenostmi v projektu dále pracovat. Každý sprint připravit závěrečnou zprávu z testování, která bude předložena projektu k odsouhlasení.



Obrázek 20 - Opatření týkající se vyhodnocení testování

5 Výsledky a diskuse

V předcházející části byla navržena některá opatření, která by mohla pomoci s optimalizací procesu testování v projektu. Zda a jaká opatření má smysl realizovat bude potřeba ověřit nejprve na úrovni interního týmu a následně se zadavatelem, nicméně ještě před samotnou realizací doporučuji následující postup.

5.1.1 Identifikace klíčových oblastí pro zlepšení

Z pohledu testování byla popsán proces testování, provedena as-is analýza metod testování a v souladu s nejlepšími možnými postupy navrženy další kroky, které mohou vést k zefektivnění procesu testování. Je zřejmé, že bude potřeba zvážit, jaké metody vylepšit a jaké do stávajícího procesu implementovat jako nové.

5.1.2 Získat zpětnou vazbu od zadavatele

Důležitá prerekvizita pro realizaci opatření je získání zpětné vazby od zadavatele projektu. Měli bychom klást otázky, zda je vyvíjená aplikace dostatečně kvalitní, zda požadavky na ni kladené byly skutečně naplněny z pohledu funkcionality i nefunkčních požadavků. Zadavatel by měl primárně zapojit roli Product Owner i reálné uživatele, kteří aplikaci používají. Pokud se ukáže, že existuje prostor ke zlepšení a zadavatel uzná za vhodné provést některé změny v testování, jsme schopni mu nabídnout navržená řešení.

5.1.3 Stanovení rozpočtu

Navržená opatření doporučuji ohodnotit jak z pohledu dopadů do rozpočtu, tak z pohledu náročnosti na další lidské zdroje. Je možné, že část opatření nebude vyžadovat navýšení investic. Tyto pak mohou být implementovány okamžitě bez potřeby složitého vyjednávání se zadavatelem. Opatření, která budou vyžadovat jednorázovou nebo pravidelnou investici bude potřeba ocenit a stanovit k nim jednoznačné přínosy, aby bylo možné zadavateli poskytnout relevantní vstupy pro následné rozhodnutí. Kromě finančního ocenění bude potřeba vzít v úvahu časovou náročnost, aby bylo jasné, jak dlouho implementace potrvá a kdy očekávat první výsledky.

5.1.4 Prezentace návrhu opatření zadavateli

Zadavateli budou prezentována opatření v souvislosti se získanou zpětnou vazbou. Jednotlivé body zpětné vazby budou jasně strukturovány a propojeny s navrženými opatřeními. Prezentace bude obsahovat vysvětlení, proč jsou návrhy důležité a jak pomohou k dosažení stanovených cílů. Zadavatel bude informován o ceně a harmonogramu realizace. Součástí pak bude diskuse a zodpovězení všech otázek, které se řešení budou týkat.

6 Závěr

Bakalářská práce definuje testování jako součást procesu zajištění kvality, tedy všech činností, které mají za cíl naplnění potřeb a očekávání uživatele týkajících se dodávaného produktu. Testování software je aktivita, jejímž účelem je identifikace všech nedostatků ve formě chyb, které mohou výslednou kvalitu software ovlivnit.

Aby bylo možné zavést vhodné metody testování v projektu zabývajícím se vývojem software, je třeba se nejprve zorientovat v možných přístupech, technikách a obecně pojmech, které s testováním souvisí. Metodika testování je rámec, do kterého lze začlenit celou řadu technik, typů a úrovní testování. Bylo by chybou domnívat se, že testování zahrnuje pouze přímé ověření vlastností kódu aplikace nebo nepřímé ověření funkcionality pomocí vhodných vstupů a očekávaných výstupů. Naopak, do testování je zahrnuta celá řada činností, které pomáhají odhalit chyby již ve fázi definice požadavků, jejich analýzy a tvorby specifikace. Využití těchto technik již v rané fázi vývoje pomáhá šetřit čas i náklady, protože k odhalení potenciálních nedostatků dochází mnohem dříve. Efektivnímu využívání zdrojů napomáhá i správná volba úrovně testování. Díky té je možné systematicky ověřovat konkrétní funkcionalitu, začít u méně komplexních testů a postupně zvyšovat složitost a rozsah testování. Každá úroveň se potom zaměřuje na jiný aspekt aplikace. U jednotkových testů se předpokládají testy kódu, v integračních testech se zkoumá interoperabilita vyvinutých komponent a během systémových testů se ověřuje funkcionalita na základě funkční specifikace.

Pro zavedení vhodných metod testování byly zvažovány i způsoby, jakými jsou v současnosti řízeny projekty vývoje software. Dle zjištění se objevují vodopádové, iterativní a agilní přístupy, z nichž každý z nich má své výhody a nevýhody. Nebylo nicméně prokázáno, že by některý z uvedených přístupů významně ovlivňoval použití specifických metod testování.

V rámci práce je testování zkoumáno i z procesního pohledu. Touto problematikou se zabývá oblast nazývaná řízení testování. Ta definuje celou řadu činností, které do takto definovaného procesu patří, a které na sebe navazují prostřednictvím specifikovaných vstupů a výstupů. Za základní dokument, prolínající se celým procesem, je považován testovací plán. Ten si lze představit jako projektový plán, určený pro potřeby testování. Zahrnuje veškeré informace o testování od plánování, harmonogramu, zajištění všech nezbytných

zdrojů společně s metodami, které budou pro testování použity. Testovací plán nicméně není jediným artefaktem testování, který bylo třeba v rámci práce specifikovat. Dalšími podstatnými artefakty jsou například testovací případy a testovací scénáře, které určují, jakým způsobem bude vyvíjené řešení otestováno. V rámci návrhové a vývojové fáze testování jsou případy a scénáře specifikovány a dopracovány, zatímco ve fázi provedení jsou testy dle těchto artefaktů spouštěny a vyhodnocovány.

V praktické části práce byla provedena analýza nastavení stávajícího procesu testování ve vybraném projektu. Projekt zabývající se vývojem řešení pro krátkodobé řízení CO2 emisí v automobilovém průmyslu, je řízený agilním přístupem a má již implementovanou celou řadu testovacích metod, které pomáhají zajišťovat kvalitu dodávaného software. Podařilo se zmapovat testovací proces, realizovaný převážně interním projektovým týmem a částečně týmem zadavatele ve fázi akceptačních testů. Analýza stávajícího procesu odhalila, že existuje prostor ke zlepšení, a to zejména v oblastech chybějící risk analýzy, politiky a strategie testování. Dále se podařilo zjistit, že všechny aktivity testovacího cyklu je možné upravit v souladu s doporučeními, které byly zjištěny v teoretické části práce.

Aby byl realizován praktický přínos práce, byl navržen další postup, který by měl zajistit implementaci zjištěných opatření do stávajícího procesu testování. Důležitým předpokladem je zpětná vazba od zadavatele, která určí podobu a rozsah implementace. Všechna opatření tak budou ohodnocena z pohledu pracnosti a náročnosti na zdroje včetně předpokládané doby trvání a očekávaných přínosů. Ve formě prezentace pak budou návrhy představeny zadavateli jako vstup pro rozhodnutí následné realizace.

7 Seznam použitých zdrojů

Agarwal, B.B., Tayal, S.P. and Gupta, M. (2010) *Software engineering & testing: an introduction*. Sudbury, Mass.: Jones and Bartlett.

Allan, M. (2019) 'What is Iterative Development? - An Easy Guide for the Beginners', *GoodCore Blog*, 25 October. Available at: https://www.goodcore.co.uk/blog/iterative-development/ (Accessed: 9 March 2022).

Bakanoğlu, S. (2017) *Test Documentation: Test Policy, Test Strategy, Test Plan….* Available at: https://www.icterra.com/test-documentation-test-policy-test-strategy-test-plan/ (Accessed: 12 March 2023).

Bureš, M. *et al.* (2016) *Efektivní testování softwaru: Klíčové otázky pro efektivitu testovacího procesu*.

Cohnen, S. (2021) 'Types of Performance Testing', *StormForge*, 26 April. Available at: https://www.stormforge.io/blog/types-performance-testing/ (Accessed: 18 February 2023).

Crispin, L. and Gregory, J. (2009) *Agile testing: a practical guide for testers and agile teams*. Upper Saddle River, NJ: Addison-Wesley (The Addison-Wesley signature series).

Everett, G.D. and McLeod, R. (2007) *Software testing: testing across the entire software development life cycle*. [Piscataway, NJ] : Hoboken, N.J: IEEE Press ; Wiley-Interscience.

Galeza, A. (2022) *9 Recommended Performance Testing Tools in 2022*. Available at: https://www.netguru.com/blog/performance-testing-tools (Accessed: 17 February 2023).

Gregory, J. (2020) *Part 3: Testing vs Quality Management - What is Quality*, *DragonFire Inc.* Available at: https://janetgregory.ca/part-3-testing-vs-quality-management/ (Accessed: 5 March 2022).

Hambling, B. (2015) *Software testing: an ISTQB-BCS certified tester foundation guide*.

Hamilton, T. (2020a) *Test Case vs Test Scenario – Difference Between Them*. Available at: https://www.guru99.com/test-case-vs-test-scenario.html (Accessed: 29 November 2022).

Hamilton, T. (2020b) *What is Regression Testing? Test Cases (Example)*. Available at: https://www.guru99.com/regression-testing.html (Accessed: 18 February 2023).

Hopkin, P. (2017) *Fundamentals of risk management: understanding, evaluating and implementing effective risk management*. Fourth Edition. E-book edition. New York, NY: Kogan Page.

Institute of Electrical and Electronics Engineers *et al.* (2008) *IEEE standard for software and system test documentation*. New York, NY: Institute for Electrical and Electronics Engineers.

Jorgensen, P. (2014) *Software testing: a craftsman's approach*. Fourth edition. Boca Raton, [Florida]: CRC Press, Taylor & Francis Group.

Kuć, R. (2023) *10+ Best Log Analysis Tools of 2023 [Free & Paid Log Analyzers]*, *Sematext*. Available at: https://sematext.com/blog/log-analysis-tools/ (Accessed: 17 February 2023).

Lewis, W.E., Dobbs, D. and Veerapillai, G. (2009) *Software testing and continuous quality improvement*. 3rd ed. Boca Raton: CRC Press.

Lonergan, K. (2016) *Agile Versus Waterfall?*, *PMIS Consulting Limited*. Available at: https://www.pmis-consulting.com/agile-versus-waterfall/ (Accessed: 8 March 2022).

Mckinley, N. (2022) *The Eleven Must-Have Features for Test Management Tools*, *Medium*. Available at: https://productcoalition.com/the-eleven-must-have-features-for-test-management-tools-d60d04c926a4 (Accessed: 30 January 2023).

Molyneaux, I. (2014) *The art of application performance testing*. Second edition. Beijing: O'Reilly (Theory in practice).

Myers, G.J., Sandler, C. and Badgett, T. (2012) *The art of software testing*. 3rd ed. Hoboken, N.J: John Wiley & Sons.

Naik, K. and Tripathy, P. (2008) *Software testing and quality assurance: theory and practice*. Hoboken, N.J: John Wiley & Sons.

Neumeyer, A. (2022) *Defining a Project Classification That Works For Your Organization*, *Tactical Project Manager*. Available at: https://www.tacticalprojectmanager.com/defining-a-project-classification-that-works-for-your-organization/ (Accessed: 19 February 2023).

Office Of Government Commerce (ed.) (2012) *Managing successful projects with PRINCE2*. 5. ed., 2. impr. London: TSO.

Oxford Learner's Dictionaries (b. r.) *Oxford Learner's Dictionaries*. Available at: https://www.oxfordlearnersdictionaries.com/definition/english/quality_1?q=quality (Accessed: 11 September 2022).

Page, A., Johnston, K. and Rollison, B. (2009) *Jak testuje software Microsoft*. Brno: Computer Press.

Palmquist, S. *et al.* (2013) *Parallel Worlds: Agile and Waterfall Differences and Similarities*. report. Carnegie Mellon University. Available at: https://doi.org/10.1184/R1/6576047.v1.

Patton, R. (2002) *Testování softwaru*. Praha: Computer Press.

Pham, R. (2017) ‘Categories of Test Design Techniques’, *ISTQB Foundation*, 18 September. Available at: https://istqbfoundation.wordpress.com/2017/09/18/categories-of-test-design-techniques/ (Accessed: 17 November 2022).

Pinkster, I. *et al.* (2004) *Successful Test Management: an Integral Approach*. Berlin, Heidelberg: Springer Berlin Heidelberg.

Project Management Institute (ed.) (2017) *A guide to the project management body of knowledge / Project Management Institute*. Sixth edition. Newtown Square, PA: Project Management Institute (PMBOK guide).

Quinn, F. (2022) *What is a Test Strategy and How to Build One*, *TestLodge Blog*. Available at: https://blog.testlodge.com/what-is-test-strategy/ (Accessed: 17 November 2022).

Rakjumar (2022) *Difference between defect, bug, error and failure*. Available at: https://www.softwaretestingmaterial.com/difference-between-defect-bug-error-and-failure/ (Accessed: 28 November 2022).

Roudenský, P. and Havlíčková, A. (2013) *Řízení kvality softwaru: pruvodce testováním*. Brno: Computer Press.

Sharma, S. (2019) *Why Are Bug Tracking Tools so Important for Testing Teams? - DZone*, *dzone.com*. Available at: https://dzone.com/articles/why-is-bug-tracking-tool-so-important-for-the-test (Accessed: 16 February 2023).

Sheremetov, D. (2022) *What Does a Software Test Engineer Do [Role & Responsibilities]*. Available at: https://onix-systems.com/blog/what-does-a-software-test-engineer-do-during-software-development (Accessed: 28 November 2022).

Software Testing Help (2023a) *Software Testing Methodologies For Robust Software Delivery*, *Software Testing Help*. Available at: https://www.softwaretestinghelp.com/software-development-testing-methodologies/ (Accessed: 18 February 2023).

Software Testing Help (2023b) *Top 11 Test Case Management Tools [Latest 2023 Ranking]*, *Top 11 Test Case Management Tools*. Available at: https://www.softwaretestinghelp.com/test-case-management-tools/ (Accessed: 30 January 2023).

Thakkar, M. (2019) ‘Key Performance Metrics for Effective Performance Testing’, *QA Testing Service Provider Company Australia | KiwiQA*. Available at: https://www.kiwiqa.com.au/blogpost/key-performance-metrics-for-effective-performance-testing/ (Accessed: 17 February 2023).

Tian, J. (2005) *Software quality engineering: testing, quality assurance, and quantifiable improvement*. Available at: http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&db=nlabk&AN=131934 (Accessed: 1 March 2022).

8 Seznam obrázků, tabulek, grafů a zkratek

8.1 Seznam obrázků

8.2 Seznam tabulek

8.3 Slovníček

Backlog – seznam požadavků, které mají být vyvinuty v rámci projektu
CO2 emise – uvolňování oxidu uhličitého do atmosféry při spalování fosilních paliv
Degradace dat – proces záměrného narušení integrity dat s cílem jejich anonymizace
Demo – předvedení hotového výstupu sprintu zadavateli
IBM Planning Analytics – software pro plánování, předpovědi a anlýzy
JAVA – objektově orientovaný programovací jazyk
Krátkodobé řízení CO2 emisí – řízení CO2 emisí v rámci kalendářního nebo fiskálního roku
MDX dotaz – jazyk pro vytváření dotazů a analýzu dat v multidimenzionálním prostoru
Multidimenzionální prostor – prostor, který obsahuje více než tři rozměry a který umožňuje analyzovat vztahy a data z různých pohledů
REACT – open-source knihovna v JavaScriptu vyvinutá společností Facebook
REST API - architektura webových služeb, která umožňuje klientům získávat, aktualizovat a mazat data ze serveru pomocí standardních HTTP metod.
Restore prostředí – obnova prostředí po ukončení testovacího cyklu
Stand-up – krátké denní setkání týmu v rámci agilní metodiky SCRUM
Sprint – časový rámec v rámci agilní metodiky SCRUM, ve kterém probíhá vývojový cyklus
Sprint planning – první fáze každého sprintu v rámci agilní metodiky SCRUM
Sprint review – fáze sprintu, kdy se prezentují výsledky a splnění cílů, které byly definovány na začátku sprintu
SCRUM – jedna z agilních metodik projektového řízení
Třívrstvá aplikace – softwarová architektura skládající ze tří vrstev (prezentační, aplikační, datové), které jsou na sobě nezávislé
Workflow – soubor definic stavů a přechodů mezi nimi