Univerzita Pardubice

Fakulta ekonomicko – správní

Návrh postupu pro proces testování softwaru ve vybrané organizaci

Diplomová práce

2023 Bc. Laura Betcherová

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: 2022/2023

ZADÁNÍ DIPLOMOVÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Laura Betcherová**
Osobní číslo:	**E19966**
Studijní program:	**N0688A140007 Informatika a systémové inženýrství**
Specializace:	**Informatika ve veřejné správě**
Téma práce:	**Návrh postupu pro proces testování softwaru ve vybrané organizaci**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování

Cílem práce je navrhnout postup a procesní mapu, která by usnadňovala firmě zabývající se vývojem a servisem software rozhodování, jak nastavit proces testování pro danou dodávku software.

Osnova:

- Požadavky na software – typy požadavků, nástroje pro správu
- Testování software
- Popis současného stavu a cílového stavu
- Návrh a ověření postupu

Rozsah pracovní zprávy: **50**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

GLENFORD J. MYERS, TOM BADGETT, a COREY SANDLER. The Art of Software Testing. 3. B.m.: Word Association, Inc., nedatováno. ISBN 978-1-118-13313-2.
JEZ HUMBLE a DAVID FARLEY. Continuous Delivery. Boston: Pearson Education, Inc., nedatováno. ISBN 978-0-321-60191-9.
GENE KIM, JEZ HUMBLE, PATRICK DEBOIS, a JOHN WILLIS. The DevOps Handbook. B.m.: IT Revolution Press, LLC, nedatováno. ISBN 978-1-942788-07-2.
ISO/IEC/IEEE 29119-1:2013 Software and Systems Engineering – Software Testing – Part 1: Concepts and Definitions

Vedoucí diplomové práce: **doc. Ing. Hana Kopáčková, Ph.D.**
Ústav systémového inženýrství a informatiky

Datum zadání diplomové práce: **1. září 2022**
Termín odevzdání diplomové práce: **30. dubna 2023**

L.S.

prof. Ing. Jan Stejskal, Ph.D. v.r.
děkan

prof. Ing. Jitka Komárková, Ph.D. v.r.
garant studijního programu

V Pardubicích dne 1. září 2022

Prohlašuji:

Práci s názvem Návrh postupu pro testování softwaru ve vybrané organizaci jsem vypracovala samostatně. Veškeré literární prameny a informace, které jsem v práci využila, jsou uvedeny v seznamu použité literatury.
Byla jsem seznámena s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 7/2019 Pravidla pro odevzdávání, zveřejňování a formální úpravu závěrečných prací, ve znění pozdějších dodatků, bude práce zveřejněna prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 15.12.2023

Bc. Laura Betcherová v. r.

PODĚKOVÁNÍ

Chtěla bych vyjádřit svou upřímnou vděčnost všem, kteří mě podporovali během mé cesty při psaní této diplomové práce, zejména mému partnerovi, přátelům, kolegům a rodině. Vím, že to se mnou v tomto období nebylo jednoduché.

Ráda bych poděkovala hlavně mé vedoucí práce, doc. Ing. Kopáčkové, PhD., za její vedení a odborné rady.

Bc. Laura Betcherová

ANOTACE

Cílem práce je navrhnout postup a procesní mapu, která by usnadňovala firmě zabývající se vývojem a servisem software rozhodování, jak nastavit proces testování pro danou dodávku software.

KLÍČOVÁ SLOVA

testování, software, procesy, nástroje, plánování, požadavky

TITLE

Procedure Proposal for the Software Testing Process in the Selected Organization

ANNOTATION

The goal of the work is to propose a procedure and a process map that would make it easier for a company dealing with software development and maintenance to decide how to set up the testing process for a given software delivery.

KEYWORDS

testing, software, processes, tools, planning, requirements

OBSAH

SEZNAM ZKRATEK A ZNAČEK

BR – Business Request

CR – Change Request

FAT – Factory Acceptance Testing

MHR – Man-hour

NFR – Non-functional Request

PQDD – Portal for Quality of Data Display

RCC – Regional Coordination Center

SAT – Site Acceptance Testing

SIT – Software Integration Testing

SW – Software

TC – Test Case

TSO – Transmission System Operator

UAT – User Acceptance Testing

UC – Use case

Úvod

Testování software je proces nebo sada procesů, jejichž cílem je zajistit, aby počítačový kód dělal to, k čemu byl navržen a naopak, že nedělá nic nezamýšleného. Software by měl být předvídatelný a konzistentní a neměl by uživatele nijak překvapit. [1]

Testování software je nezbytnou součástí jeho samotného vývoje. Pokud je správně naplánováno a poté i podle daného plánu vykonáno, dokážeme tak díky němu výrazně snížit rizika potenciálních incidentů, eliminovat čas, a tudíž i finance na opravy chyb hlášených zákazníkem a také získat, příp. udržet si spokojeného klienta, který dostane kvalitní software.

Pojďme si výše zmíněná tvrzení potvrdit čísly. Na základě statistických údajů zveřejněných [2]:

- Na 1 000 řádcích kódu se průměrně nachází 70 chyb
- 15 z těchto chyb se dostane až ke zákazníkovi
- Oprava jedné chyby trvá 30krát déle než napsaní jednoho řádku kódu
- 75 % pracovního času programátora je tvořeno opravou chyb

Abychom byli schopni eliminovat čas věnovaný opravám chyb a nespokojenost na straně zákazníka, musíme se více zaměřit již na první kroky směřující k vytváření software a tím je analýza následovaná plánováním. Díky pečlivé analýze veškerých požadavků na software a jejich souvislostí s aspekty, které ani nebyly vyřčeny samotným klientem, nabudeme reálnou představu o náročnosti naše tvorby. Na základě toho jsme schopni sestavit tým lidí, rozpad jednotlivých pracovních částí a jejich časovou náročnost.

V realitě se často potkáváme s tím, že testování je poněkud opomíjenou disciplínou a už během fáze plánování na ní není alokovaný potřebný čas, příp. alokovaný čas je pak zkrácen časem, o který byl prodloužen vývoj.

Cílem této práce je analyzovat procesy práce s požadavky, plánování testů a proces samotného testování v konkrétní nejmenované společnosti v rámci projektu PQDD. Na základě zjištěných informací poté navrhnout změny v těchto procesech a následně je implementovat a vyzkoušet při plánování reálného dodání nové verze software.

V rámci projektu PQDD probíhá kontinuální vývoj stejnojmenného software, přičemž nové verze software se dodávají dva až tři krát ročně. V případě kontinuálního vývoje je první vydání software na produkční prostředí klienta pouze prvním stadiem jeho životního cyklu. Software se dál a dál vyvíjí formou několika nových verzí ročně, v našem případě dvou až tří. Je tedy důležité, aby se s kontinuálním vývojem software také vyvíjel a měnil i proces dodávání nových verzí s ohledem na měnící se okolí. Správně nastavený proces testování je právě klíčovým prvkem úspěchu kontinuálního vývoje. [3]

Začátkem práce si více přiblížíme samotnou oblast práce s požadavky na software a poté testování software, kde si projdeme jednotlivé aspekty, které musíme při plánování zvážit. Dále se budeme zabývat současnou situaci na projektu PQDD a oblastem, ve kterých vidíme příležitost ke zlepšení.

1. POŽADAVKY NA SOFTWARE

Požadavek je cokoliv, co ovlivňuje rozhodování při návrhu (Brian Lawrence, 1997).

Sběr a správné zpracování požadavků na software je jednou z nejdůležitějších částí IT projektů a její podcenění může zvyšovat rizika, že nebude splněn cíl projektu. Typy požadavků a způsob jejich zpracování významně ovlivňují způsob, jakým se bude plánovat a provádět testování software, proto si v následující kapitole stručně uvedeme typy požadavků a jejich zpracování.

Každý požadavek by měl být jasný, konkrétní a stručný a měl by obsahovat:

- Identifikátor
- Název požadavku
- Popis
- Kritéria akceptace

Základní dělení požadavků na software definuje funkční a nefunkční požadavky.

1.1 Funkční požadavky

Funkční požadavky popisují chování software za konkrétních podmínek a tím pomáhají zachytit zamýšlené chování systému. Zpravidla jsou koncovými uživateli vysloveně požadovány jako základní vybavení systému. Funkční požadavek tak definuje konkrétní akci, kterou musí software provést a akceptační kritéria, která určují, jak bude měřena úspěšnost provedení požadavku.[4]

Příklady oblastí, na které se může funkční požadavek vztahovat:

- Uživatelské funkce
- Administrativní funkce
- Zobrazovací funkce
- Integrace a rozhraní
- Zpracování dat
- Výpočty
- Komunikace a notifikace

1.2 Nefunkční požadavky

Nefunkční požadavky, označovány i jako nebehaviorální požadavky nebo požadavky kvality, jsou definovány jako kvalitativní omezení, které musí software splňovat. Často krát jsou ovlivněny vnějšími faktory jako třeba legislativa nebo provázanost s jiným softwarem.

Tento typ požadavků se zaměřuje na oblasti:

- Výkonnost
    - Rychlost odezvy, zatížení systému
- Spolehlivost
    - Obnovení po selhání, doba mezi poruchami
- Bezpečnost
    - Ochrana dat, audity
- Dostupnost
    - Doba dostupnosti služby
- Podpora různých platforem
    - Kompatibilita s operačními systémy nebo jiným softwarem
- Uživatelská spokojenost
    - Dostupnost podpory
- Škálovatelnost
    - Škálovatelnost na základě zátěže
- Flexibilita
    - Flexibilita v možnosti konfigurace a přizpůsobení různým potřebám a prostředím
- Udržitelnost
    - Udržitelnost v průběhu času (snadno rozumět, upravovat, rozšiřovat a opravovat kód)
- Znuvupoužitelnost
    - Znuvupoužitelnost kódu v různých částech software případně v jiných projektech, modulární kód s minimálním množstvím závislostí [5]

Nefunkční požadavek obvykle představuje sám o sobě akceptační kritérium. Na rozdíl od funkčních požadavků je snáz měřitelný a kvantifikovatelný což usnadňuje jeho vyhodnocení během akceptačních testů i v samotném provozu. [4]

1.3 Nástroje pro správu požadavků

Stejně důležité jako správná definice požadavku na software je pak i jeho následné řízení. Zejména při větším množství požadavků je nezbytné jejich správné řízení, které spočívá v třídění, prioritizaci, pravidelných aktualizacích reálného stavu a v neposlední řadě i archivování. Těchto nástrojů existuje celá řada, my si uvedeme několik z nich, se kterými má zkušenost tým naši nejmenované společnosti, se kterým v rámci této práce spolupracujeme.

1.3.1 JIRA

Nejvíce využívané jsou tzv ticketovací nástroje, které umožňují přehlednou správu požadavků během celého jejich životního cyklu pro všechny zainteresované strany anebo jen pro účely interního řízení. Jsou z velké míry konfigurovatelné, aby odpovídaly požadavkům jednotlivých projektů a jejich využitelnost je díky tomu velice pestrá. Jsou vhodné jak pro tradičně, tak i agilně řízené projekty, tzn umožňují správu jak dílčích sprintů, tak i celých dodávek softwarových verzí.

Nejrozšířenějším nástrojem je v této oblasti produkt JIRA od společnosti Atlassian.

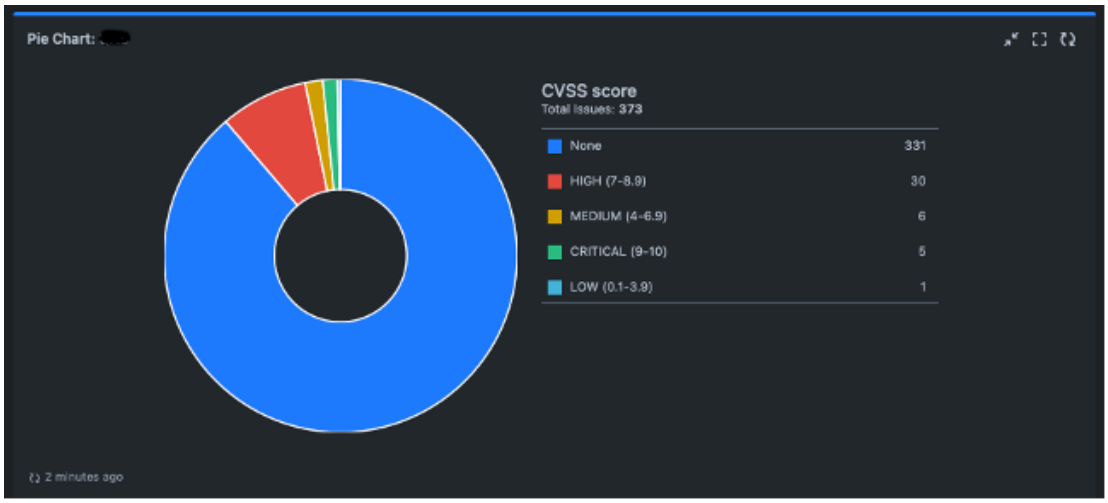
Označení „ticketovací“ nástroj plyne z hlavního prvku tohoto nástroje, čímž je ticket. Na každý dílčí úkol, požadavek, chybu, incident, problém, riziko, vulnerabilitu a mnoho dalších položek může být vytvořen ticket adekvátního typu. Ticket pak může obsahovat veškeré náležitosti, které se dané oblasti týkají – emailové komunikace, přílohy, odkazy na související tickety, které ovlivňují daný ticket anebo jsou ním ovlivněny a mnoho dalších vazeb. Každý typ ticketu má definované své workflow, které si lze nakonfigurovat podle vlastních kritérií pro každý projekt a určuje životní cyklus daného typu ticketu (možné stavy ticketu a jejich návaznost). Tickety mají definovanou svoji prioritu, severitu a urgenci, na základě čeho je řízení a prioritizace úkolů mnohem snazší. Nabízí také řadu možností na vytváření vlastních dashboardů (včetně Kanbanu), které můžou být využity jak pro interní přehled na projektu, tak i pro externí reportování stavu vedení anebo klientovi.

Využití takového ticketovacho nástroje je vhodné pro veškeré fáze životního cyklu IT projektu včetně servisu software. Kromě svých klasických funkcionalit nabízí také řadu doplňkových pluginů, které umožňují například měření SLA, Service Level Agreement, na dobu reakce (jak dlouho po vytvoření ticketu klientem na něj

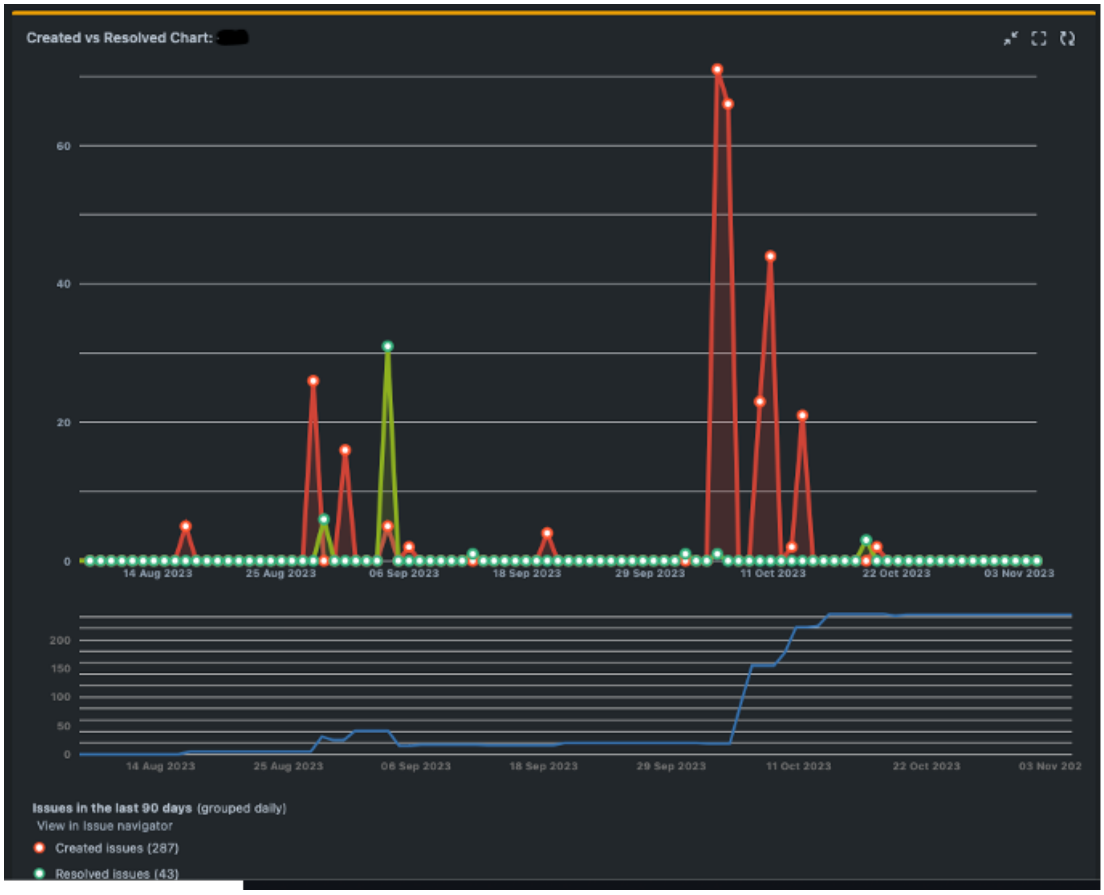
zareagoval servisní pracovník) nebo na dobu vyřešení (jak dlouho od první reakce servisního pracovníka trvalo vyřešení incidentu).

JIRA je velkým pomocníkem i pro testování software, či už pro jeho řízení anebo samotné vykonávání testů. Pomocí různých typů označení lze zařadit tickety do konkrétního sprintu anebo verze software, což nám zajistí kontrolu, že každá část byla řádně otestovaná. Díky označením výrazně zjednodušuje práci s backlogem a minimalizuje tím riziko, že by určitý nápad na vylepšení anebo oprava minoritní chyby, kterou nešlo zakomponovat do předchozích verzí, byla zapomenuta.

Následující obrázky představují ukázky dashboardů vytvořených v JIRA prostředí, znázorňují grafy počtu ticketů podle jejich severity a počtu ticketů, které byly založeny a uzavřeny v rámci zkoumaného období. [6]



Obrázek 1: JIRA Dashboard – ukázka 1



Obrázek 2: JIRA Dashboard – ukázka 2

1.3.2 Confluence

Confluence, také od společnosti Atlassian, je vhodný nástroj pro sdílení požadavků a projektových informací a pro udržování dokumentace. Její propojení s nástrojem Jira umožňuje jednoduché udržování aktuálních informací. Například v dokumentu o nové verzi anebo sprintu, můžeme mít relevantní informace přímo z Jira, které reflektují aktuální stav bez nutnosti přepínání mezi těmito nástroji.

Podobně jako Jira, i Confluence má množství možností rozšíření pomocí doplňkových pluginů.

V oblasti testování je oblíbené využití dokumentace testovacích scénářů právě v Confluence, kde pak lze přímo odkázat testovací případ z Jira. Také může sloužit pro uchování testovacích dat tak, aby byly uloženy bezpečně a měli k nim přístup pouze oprávnění lidi. [7]

1.3.3 Další nástroje

Jira a Confluence jsou dva celosvětově nejvíc využívané nástroje pro správu požadavků a řízení IT projektů jako takových. Jako další příklady můžeme stručně uvést:

- Trello
    - Trello je dalším nástrojem od společnosti Atlassian, který pomáhá řídit požadavky, úkoly, schůzky a jiné náležitosti, které potřebujeme spravovat v rámci (nejen) IT projektu. Umožňuje propojení s dalšími nástroji z rodiny Atlassian jako je výše zmíněna Jira a Confluence.
    - Mimo tento teamový účel, je Trello skvělým pomocníkem i na osobní time management, kde umožňuje vytvářet různé nástěnky, třeba i Kanban. [8]
- Asana
    - Asana je nástroj, který usnadňuje plánování, sledování a spolupráci na různých projektech a úkolech. Poskytuje různé způsoby vizualizace včetně plánovače. Dokáže pracovat s termíny pro splnění úkolů a posílat upomínky uživatelům ohledně nedodržení úkolů. Nad dílčím úkoly může probíhat diskuse celého teamu což umožňuje sdílet informace skupinově. [9]
- IBM Engineering Requiremenent Management DOORS
    - Nástroj DOORS využívá umělou inteligenci k vylepšení kvality požadavků a pomáhá optimalizovat komunikaci a spolupráci napříč týmy a stakeholdery za účelem maximalizace produktivity a kvality.
    - Mezi klíčové vlastnosti se řadí možnosti definování a organizování požadavků, možnosti textových i grafických popisků, sledovatelnost požadavků, spolupráce včetně revizí a schvalování, reporting a dashboardy.
    - Také nabízí možnost konfigurace nástroje podle konkrétních potřeb a představ uživatelů. [10]

2. TESTOVÁNÍ SOFTWARE

Testováním software vzniká přidaná hodnota zvyšováním spolehlivosti software, a to hledáním a odstraňováním chyb. Proto netestujeme software abychom demonstrovali, že funguje správně, naopak, vycházíme z předpokladu, že software obsahuje chyby a chceme odhalit co nejvíce z nich. [1]

V následující části si objasníme, co a proč je vlastně účelem testování software:

- Odhalování chyb – co umožňuje jejich následovné odstranění a tím zvýšení kvality software
- Shromažďování informací o testované oblasti – testování generuje informace, které mohou sloužit různým účelům, například:
    - Vývojáři mohou použít informace k odstranění chyb, zvýšení kvality kódu anebo se naučit vytvářet kvalitnější kód v budoucnu
    - Testeři mohou použít informace k vytvoření lepších testovacích scénářů
    - Manažeři mohou použít informace k rozhodnutí, kdy ukončit testování, příp. jakou změnu v plánování testování mají v budoucnu udělat
    - Uživatelé mají prospěch z vyšší kvality software
- Budování důvěry a rozhodování – poskytnutím důkazu, že testovaná položka funguje za specifických podmínek správně, se zvyšuje důvěra stakeholdera, že testovaná položka bude fungovat správně v provozu. S dostatečnou důvěrou se mohou stakeholdeři rozhodnout testovanou položku vydat. [11]

Pro oblast testování software existuje soubor principů, které bychom měli brát v potaz při plánování a vykonávání testů software:

1. Nezbytná část testovacích případů pozůstává z definice očekáváných vstupů anebo výsledků.
2. Programátor by se měl vyhýbat testování svého vlastního kódu.
3. Společnost, která software vyvinula by neměla být jediná, která bude software testovat.
4. Každý testovací proces by měl zahrnovat důkladnou kontrolu výsledků každého testu.
5. Testovací případy musí být připraveny pro vstupní podmínky, které jsou nevalidní a neočekávané, stejně tak, jak pro ty validní a očekáváné.

6. Vykonávání testů za účelem ověření, jestli software nedělá to, co by měl dělat je pouze polovinou úspěchu, druhá polovina spočívá v ověření, jestli software dělá něco, co by neměl.
7. Vyhněme se jednorázovým testovacím případům, pokud software skutečně není na jedno použití.
8. Neplánujme náročnost testů na základě předpokladu, že nebudou nalezeny žádné chyby.
9. Pravděpodobnost přítomnosti více chyb v konkrétní části software je úměrná počtu chyb, které již byly v této oblasti nalezeny.
10. Testování je kreativní a intelektuálně náročný úkol. [1]

2.1 Vstupy

Abychom mohli řádně naplánovat a zahájit testování, potřebujeme mít k dispozici dostatečné informace, materiály a zdroje. V následujících podkapitolách shrneme vstupy do procesu testování, které zahrnují širokou škálu informací, od požadavků na software až po technické specifikace a vše, co se nachází mezi nimi. Bez těchto vstupů by bylo těžké provádět testování systematicky a efektivně. Z toho důvodu je jejich správné pochopení a správné využití nezbytné pro každého, kdo se na testovaní podílí, či už jako tester, manager anebo vývojář. Následující seznamy a popisy byly připraveny na základě fungování naší nejmenované společnosti a tzv. best practices.

2.1.1 Požadavky na software

Požadavkům na software jsme se věnovali již v předchozí kapitole, proto zde jen připomeneme jejich klíčovou roly jakožto zdroje pro tvorbu testovacích scénářů, testovacích případů a pro plánování testů celkově.

2.1.2 Softwarová specifikace

Dokument anebo sada dokumentů, které obsahují detailní popis software, jeho architektury, rozhraní, datových struktur a jiných informací.

Obvykle se jedná o sadu dokumentů, které jsou rozdělené podle svého obsahu a podle cílové skupiny. V této sadě dokumentů definujeme:

- Uživatelská příručka
    - Tento dokument je připraven pro potřeby uživatele software, a proto neobsahuje technické detaily, ale je psaný čtivou, jednoduchou formou

s množstvím obrázků, jednouchých use casů (akce uživatele a reakce software), doporučení na hardware a software, který by se měl používat pro optimální fungování a další.
    - Uživatelská příručka cílí na potřeby uživatele, nicméně, je užitečná i pro účely testování, kdy tester simuluje akce, které budou v reálnem provozu vykonávané uživatelem.
- Software design
    - Software design je připraven typicky business analytikem anebo architektem a software architektem. To poukazuje na to, že technická úroveň tohoto dokumentu je vyšší a jde do mnohem větších detailů než předchozí uživatelská příručka. Obsahuje zejména informace o architektuře softwaru, datových modelech a strukturách, integrací, infrastruktuře, komponentách a nárocích na hardwarové a softwarové požadavky (opět ve větším detailu, než to je v případě uživatelské příručky).
    - Z pohledu požadavků tento dokument popisuje spíše nefunkční požadavky, zatímco uživatelská příručka je zaměřená na požadavky funkční.
    - Z externího pohledu, software design cílí na potřeby IT oddělení klienta, tzn lidi, kteří software nepoužívají jako uživatelé, ale zajišťují jeho správné nasazení a obsluhu.
    - Z interního pohledu je software design nezbytnou součástí pro analytiky a vývojáře, kteří potřebují softwaru rozumět do detailu. Pro testery může být zajímavý pro porozumění fungování systému a pro vymyšlení testovacích případů, které se pokouší software „rozbít“.
- Technická specifikace
    - Mezi technickou specifikací a software designem existuje tenká hranice a jde hlavně o domluvu s klientem a v rámci týmu na tom, co do kterého z těchto dokumentů zařadit.
    - Z úrovně detailu lze říct, že technická specifikace jde opět ještě do většího detailu, než je u software designu. Může obsahovat seznam všech use casů, které v software probíhají, přičemž nepopisují use casy z funkčního hlediska, jak je tomu v uživatelské příručce, ale z hlediska

software – jak je která komponenta a datové toky využity, aby daný use case proběhnul v pořádku.
    - Co se týče cílové skupiny tohoto dokumentu, zajímavý a užitečný je zejména pro vývojáře a business architekty. [12]

2.1.3 Testovací strategie

Testovací strategie je dokument, který definuje, jak bude probíhat testování a jeho organizace. Záleží na nastavení fungování konkrétní společnosti a domluvy s klientem, co je jejím obsahem. Obvykle obsahuje následující kapitoly:

- Terminologie
    - Vysvětlení pojmů použitých v rámci dokumentu, popis kategorizace chyb/požadavků a kategorizace severit (A, B, C)
- Fáze testování
    - název,
    - jak dlouho bude fáze trvat,
    - jaké typy testů budou vykonané,
    - kdo zodpovídá za přípravu, vykonání, revizi a akceptaci,
    - jaké prostředí bude v dané fázi využito,
- Akceptační kritéria
    - maximální možný počet chyb jednotlivých severit, se kterou se může verze předat klientovi. Například:
        - 0 kritických chyb (A),
        - 0 majoritních chyb (B),
        - 10 minoritních chyb (C),
- Testovací software
    - Seznam softwarových nástrojů použitých za účelem podpory testování a jeho účel. Například:
        - Jira – nástroj pro zaznamenávání nalezených chyb,
        - PractiTest – online nástroj pro správu testovacích případů a výsledků jejich provedení, který je možné integrovat s Jira pro okamžité sdílení výsledků testování nad konkrétním Jira ticketem,
        - MS Excel – nástroj pro správu testovacích případů a výsledků jejich provedení,

        - TeamCity – nástroj pro přípravu a spuštění unit testů, automatické kompilace zdrojového kódu,
        - ShareFile – nástroj pro sdílení dokumentů,
        - Apache JMeter – nástroj pro výkonnostní a automatizované funkční testy,
        - Nmon / Nmonchart – nástroj pro měření a vizualizaci hardwarových parametrů během výkonnostních testů,
- Testovací prostředí
    - Seznam prostředí, které budou pro testování využity, kde pro každé prostředí definujeme:
    - Účel – pro jaké testy bude využito,
    - Kdo prostředí spravuje,
    - Parametry prostředí
    - Počet serverů,
    - Seznam komponent,
    - Informace o CPU,
    - Informace o RAM,
    - Informace o HDD,
    - Seznam povolených portů,
- Typy testů
    - Seznam testů, kterou budou vykonány, pro každý typ je uveden název, popis a kdo je za jeho vykonání zodpovědný. Může se jednat o:
    - Unit testy,
    - Smoke testy,
    - Funkční testy,
    - Integrační testy,
    - Instalační testy,
    - Regresní testy,
    - Testy stability,
    - Specifika pro vybrané testovací případy
        - Testovací strategie neobsahuje kompletní seznam testovacích případů, ten je součástí příloh k dokumentu, nicméně, pro komplikovanější testovací scénáře zde můžeme uvést a upřesnit

jak přesně a s jakými daty bude daný testovací případ vykonán. [13] [14]

2.1.4 Plán testování

Plán testování je dalším důležitým vstupem, který je dobré mít připravený před zahájením testování. Je míň detailní než předchozí testovací strategie a definuje oblasti jako:

- Reporting
    - Jak bude probíhat reportování stavu testů vůči klientovi,
    - Jaké reporty a v jaké formě klient obdrží po ukončení jednotlivých typů testů,
    - Součástí může být i šablona reportu,
- Testovací data
    - Jaká data budou pro testy použita – reálná data od klienta anebo námi vytvořená testovací data,
- Workshopy a schůzky
    - V rámci plánu testů si definujeme, jak často mají probíhat schůzky s klientem, kde ho budeme informovat o stavu testů a kdo se těchto schůzek bude účastnit,
    - Určíme jestli, jak a kdy budeme organizovat workshop pro klienta a uživatele software, kde budeme prezentovat nové funkcionality aplikace a kdo se workshopu bude účastnit,
    - Jaké testovací případy budou během workshopu prezentovány. [13] [14]

2.1.5 Testovací případy

Testovací případy poskytují detailní popis, instrukce k tomu, jak otestovat dílčí část software. Měly by být napsány tak, aby je byl schopen provést i tester, který s daným softwarem nemá zkušenosti a nezná jeho detaily, tzn měly by obsahovat veškeré potřebné informace a neměly bychom při jejich přípravě počítat s tím, že si tester určitou část domyslí. Takhle popsané testovací případy pak snižují riziko toho, že testy nebudou probíhat podle plánu v případě nedostatku pracovních sil na našem projektu, protože tester bude moct být zastoupen.

Testovací případ obsahuje:

- ID
    - Jednoznačné unikátní číslo anebo kombinace čísel a písmen, které nám umožní jednoduchou a jednoznačnou identifikaci a odkazování na daný testovací případ,
- Název
    - Jednoduchý, jedinečný a krátký název, který identifikuje oblast testování daného testovacího případu,
- Popis
    - Stručný popis toho, co bude daným testovacím případem ověřováno, co je cílem této části testu,
- Prerekvizity
    - Stručná a jednoznačná definice prerekvizit anebo předpokladů k provedení testu,
    - Například pod jakou uživatelskou rolí je uživatel přihlášen, jaký prohlížeč a v jaké verzi má být použit,
- Postup testování
    - Sekvence jednoduše popsaných kroků, které má tester vykonat, jeden krok odpovídá právě jedné akci,
- Očekávaný výsledek
    - Ke každému kroku náleží očekáváný výsledek po provedení tohoto kroku,
    - Krok = akce provedená testerem,
    - Očekávaný výsledek = reakce software,
- Reálný výsledek
    - Při přípravě testovacích případů můžeme připravit rovnou část s reálnými výsledky, kde tester během testování zaznamená reálný výsledek, pokud není v souladu s uvedeným očekáváným výsledkem,
- Testovací data
    - K testovacímu případu je vhodné přidat rovnou odkaz na testovací data pro daný případ,
- Komentáře
    - Pokud chceme uvést k testovacímu případu něco, co nezapadá do předchozích kategorií, můžeme vytvořit sekci na komentáře, kde tuto

informaci uvedeme. Může se jednat například o obrázek, screenshot, očekáváného stavu pro usnadnění ověření a jiné. [1] [13] [14]

Ke každé dílčí části softwaru by měli existovat alespoň dva testovací případy, „happy day“ a alternativní. Happy day testovací případ má za cíl nasimulovat pozitivní výsledek, zatímco alternativní testovací případ má za cíl nasimulovat negativní výsledek. [1]

Příkladem happy day testovacího případu může být úspěšné přihlášení uživatele do software. Alternativní případ k němu by byl, zadání nesprávného hesla uživatelem, které vyvolá chybovou hlášku zobrazenou softwarem a uživatel se nepřihlásí.

2.1.6 Testovací data

Testovací data jsou data, které používáme pro nasimulování konkrétního testovacího případu. Může se jednat o data, která do software nahráváme anebo o data, která vyplňujeme například do formuláře v softwaru. V případě, kdy jde pouze o vyplňování, můžou být tato data uvedena přímo u testovacího případu, pokud je ale jejich obsah velký, je lepší udržovat je mimo soubor s testovacími daty abychom nezpůsobili nepřehlednost tohoto souboru. V případě, že se jedná o data, která do software nahráváme, doporučuje se mít je roztříděné do složek, kde každá složka nese název korespondující s ID testovacího případu. Takto snížíme riziko použití nesprávného datového souboru a ušetříme testerům čas. [1]

2.1.7 Testovací prostředí

Testovací prostředí je prostředí, které je obvykle ve správě dodavatele software, přičemž klient na něj může, ale nemusí mít přístup. Testovací prostředí by mělo co nejvíce odpovídat klientovu produkčnímu prostředí abychom u testů měli stejné, anebo co nejvíc podobné podmínky, v jakých bude software fungovat v reálném, produkčním, provozu. Je to důležité zejména pro testování nefunkčních požadavků, případně pro vykonávání výkonnostních a zátěžových testů. V případě, kdy testovací prostředí nemá shodné parametry s produkčním prostředím, nemají tyhle typy testů dostatečnou vypovídající hodnotu. [1]

Obvykle se pro testy využívají tři prostředí, může jich být ale víc i míň, záleží na domluvě s klientem, velikosti projektu, financích na provoz prostředí a jiných faktorech. Rozlišujeme hlavně:

- Vývojové prostředí
    - Jak ze samotného názvu plyne, vývojové prostředí je primárně pro vývojáře na testy nasazení dílčích částí software, instalační testy, rychlé testy ze strany vývoje. Na základě toho, toto prostředí nemusí splňovat tak přísná kritéria na podobnost s produkčním prostředí, jelikož zde neprobíhají tak náročné testy.
    - Klient na toto prostředí nemá přístup.
- Interní testovací prostředí (FAT)
    - Interní testovací prostředí je využíváno zejména pro funkční testy na straně dodavatele software, tzv FAT (Factory Acceptance Test). Na toto prostředí jsou již kladeny větší nároky, co se týče podobnosti s produkčním prostředím. Pokud to není možno splnit, je důležité tuto skutečnost zdůraznit v rámci testovací strategie u popisu prostředí, jelikož to může u určitých testů ovlivnit výsledky testování na straně dodavatele a na straně klienta.
    - Klient na toto prostředí nemá přístup.
- Integrační prostředí (SIT)
    - Integrační prostředí je ještě o level výše naproti předchozím prostředím a jak z názvu plyne, jedná se o prostředí, které je integrované s dalším softwarem tak, jak to je na produkčním prostředí. Toto prostředí by mělo být tedy totožné s produkčním prostředí, co se týče veškerých parametrů.
    - Obvykle je ve správě klienta, ale nemusí to být pravidlem. Přístup na integrační prostředí má strana dodavatele i strana klienta.
- Klientské testovací prostředí (UAT)
    - Toto prostředí je ve správě klienta a zpravidla na něj strana dodavatele nemá přístup. Přístupy se zřizují ve výjimečných případech, a to převážně v případě dohody, že strana klienta bude poskytovat určitý druh podpory během klientských neboli uživatelských testů.

- Co se týče parametrů, platí pro toto prostředí podobné podmínky jako pro interní testovací prostředí (FAT), tzn mělo by se co nejvíc přibližovat produkčnímu prostředí, ale není to podmínkou. [1] [15]

2.2 Výstupy

Výstupy z testování mají odpovídat tomu, k čemu se zavážeme v rámci přípravy testovací strategie (2.1.3) případně plánu testování (2.1.4).

Obvykle se jedná o reporty neboli protokoly z každého typu testů, které byly vykonány, kromě unit testů vývojářů.

2.2.1 Reporty

Každý vykonaný typ testu by měl obsahovat výstupní report neboli zprávu o jeho provedení, která bude sloužit buď to jen interním účelům anebo bude poskytnuta i klientovi.

Nejdůležitějším výstupem je FAT report, který bývá povinnou položkou v rámci samotné dodávky software. Obvykle má formu protokolu s informacemi o tom:

- jaké typy testů byly provedeny,
- jaké testovací případy byly provedeny a s jakým výsledkem,
- grafické znázornění počtu provedených testovacích případů a výsledků
- zprávy o chybách nalezených během testování (pokud má klient přístup do projektového ticketovacího nástroje, stačí poskytnout odkaz na přehled těchto chyb),
    - počet chyb, které se nepovedlo opravit před samotnou dodávkou software a jejich severita,
        - srovnání tohoto počtu s počtem uvedeným v akceptačních kritériích,
- doporučení pro zlepšení
    - zde můžeme uvést návrhy na vylepšení softwaru na základě skutečností zjištěných během testování, případně i návrhy na zlepšení testovacího procesu. [1] [13] [14]

2.2.2 Scan kvality kódu

Obvyklou součástí dodávky bývá i scan kvality kódu, který je výstupem statické analýzy kódu. Nabízí se řada nástrojů, které tuto analýzu provedou a výsledky z ní se pak zpracují do dalšího reportu, který je součástí výstupů.

Jedná se o scan kódu, který ověřuje, zda jsou splněny standardy pro daný programovací jazyk, přítomnost syntaktických chyb, detekce zranitelností, duplicit kódu a další. Nástroje jsou z velké míry konfigurovatelné a metriky si zvolí každý projekt podle svých potřeb, případně podle domluvy s klientem. Součástí může být i analýza toho, jak velká část kódu je pokryta unit testy. V ideálním případě by se mělo jednat cca o 80 % pokrytí. [16]

2.2.3 Release notes

Poznámky k vydání software, označovány i v češtině jako release notes, obsahují popis všeho nového, upraveného anebo odstraněného ze software v jeho nejnovější verzi.

Začátek dokumentu zpravidla obsahuje informace o datumu vydání software a jeho přesnou verzi včetně čísla buildu.

Úvod dokumentu pak shrnuje popis software a iniciativu za jeho novou verzi, kdo zadal změnové požadavky a co přináší. Dále jsou zmíněné minimální hardwarové a softwarové požadavky, které jsou nutné pro správné fungování aplikace a seznam verzí software třetí strany, který je s danou verzi nového software kompatibilní. Součástí dokumentu může být i popis instalace anebo aktualizace software. Pokud je ale jedná o složitejší proces, bývá většinou popsán v separátním dokumentu, instalační příručce.

Hlavní část dokumentu pak obsahuje seznamy a popisy změn v software od poslední verze. Obvykle nejsou změny popisovány do detailů, ale obsahují odkaz na změnový požadavek anebo na specifikaci.

Závěr dokumentu obvykle obsahuje seznam všech dokumentů, reportů a jiných náležitostí, které jsou součástí vydání software a odkaz na místo, kde je čtenář může najít. [13] [14]

2.3 Typy testů podle změn v software

Na základě toho, jaké změny v software jsou obsahem nové verze, definujeme několik typů testů. Pokud například nová verze software obsahuje pouze nefunkční změny, není potřeba testovat všechny testovací případy, které prověřují veškeré funkcionality v software. Z tohoto důvodu si v následujících podkapitolách shrneme dělení typů testů podle toho, jaké změny byly v software implementovány.

2.3.1 Testy funkčních požadavků

Funkční změny neboli funkční požadavky, jsme si již představili v kapitole 1.2. Jedná se o změny, které přímo ovlivňují funkčnost softwaru z pohledu koncového uživatele. Jejich testy jsou pokryté zejména testovacími případy, které jsme si objasnili v kapitole 2.1.5. Jedná se tedy o testování chování software z perspektivy reálného uživatele, tudíž funkční testy by měly pokrývat veškeré možnosti, které uživatel může v software vykonat.

V rámci projektu PQDD členíme funkční testy do několika oblastí:

- Testy uživatelského rozhraní (GUI)
    - Testovací případy zaměřující se na konzistentnost uživatelského rozhraní (zda rozložení jednotlivých prvků odpovídá specifikacím), správnost navigace mezi obrazovkami (zda přesměrování mezi jednotlivými obrazovkami funguje dle specifikace) a interakce s uživatelskými prvky (zda akce uživatele nad daným prvkem vyvolá reakci software podle specifikace)
- Testy funkcionalit
    - Testovací případy zaměřující se na veškeré funkcionality software
- Testy kompatibility
    - Testy kompatibility jsou pokryty testovacími případy na uživatelské rozhraní i na testy funkcionalit. Kompatibilitu ověřujeme pomocí různých předpokladů k daným testovacím případům čímž ověříme fungování i zobrazení software za použití všech podporovaných webových prohlížečů a operačních systémů

2.3.2 Testy nefunkčních požadavků

Testy nefunkčních požadavků můžou být částečně pokryty v rámci testovacích případů, kdy pomocí předpokladů k testovacímu případu ověřujeme stejnou funkcionalitu za použití různých webových prohlížečů a/nebo operačních systémů.

Další části nefunkčních požadavků nelze otestovat z pohledu koncového uživatele a je potřeba využít jiné typy testů. Může se jednat například o požadavek, že software splňuje nefunkční požadavky na dobu odezvy i v případě, že je ve stejnou chvíli aktivně přihlášeno 50 uživatelů. Manuálním testem bychom požadavek tohoto typu ověřili jen velice těžko a výsledek by i tak nemusel být přesný.

Pro takové typy požadavků využíváme **zátěžové neboli výkonnostní testy**. Pomocí kódu se snažíme nasimulovat podmínky vysokého zatížení software a zároveň probíhá detailní monitoring software. Takový test nám dokáže poskytnout detailní kvalitativní výsledky, se kterými můžeme vyhodnotit splnění nefunkčních požadavků, ale také můžeme odhalit hranice výkonu systémů. Například při kolika aktivně přihlášených uživatelích software přestává zvládat splňovat nefunkční požadavky na odezvu, jak se prodlužuje doba odezvy software při každém dalším aktivně přihlášeném uživateli nad určitých počtem, jaký je limitní počet aktivně přihlášených uživatelů (kdy se software začne chovat tak, že práce s ním není uživatelsky přívětivá, příp. není schopen plnit své funkční požadavky atd.). Kromě aktivně přihlášených uživatelů se může jednat i o objemy dat, které jsou nahrávány do software a jiné.

Další části nefunkčních požadavků na software můžou být bezpečnostní požadavky. Požadavky na bezpečnost můžou být definovány v rámci nefunkčních požadavků anebo můžou tvořit samostatní kategorii požadavků. Jak již bylo nastíněno v kapitole 1.2, bezpečností požadavky jsou definované s cílem zajištění ochrany dat a uživatelů a odolnosti software vůči útokům. Některé části bezpečnostních požadavků lze pokrýt i testovacími případy, jako například testy autentizace, autorizace a správy hesel. Nicméně požadavky na šifrování dat, ochrany proti útokům, zabezpečení komunikace a konfigurace a jiné, nelze otestovat z pohledu koncového uživatele. Proto se pro danou oblast testů využívají **penetrační testy** neboli etický hacking. Penetrační testy můžou mít množství podob a stupňů, tím nejvyšším je tzv. red team, který simuluje reální útok na organizaci jako takovou za využití fyzické, sociální a technické taktiky.

Provádí se na objednávku vedení organizace samotné a je zaměřen na celkové dodržování bezpečnostních norem v celé organizaci, nejen na software.

Běžně prováděné penetrační testy po vývoji software anebo implementaci nových částí mají podobně jako zátěžové testy formu kódu, tzn. není potřeba aby tester manuálně zkoušel hacknout software. Tento kód simuluje bezpečností rizika a monitoruje chování software. Cílem penetračních testů je identifikovat bezpečností slabiny a potenciální zranitelnosti, které by mohly být zneužity útočníky. [17]

Kromě penetračních testů jako takových je důležité myslet i na software třetích stran, který je využívány a provádět tzn. security scany. Security scany nám pomůžou odhalit zranitelnosti v těchto částech a lze je řešit updatem na vyšší verzi, pokud již je dostupná. Pro tyto scany je velice užitečná Národní Databáze zranitelností (NVD, National Vulnerability Database), která pravidelně zveřejňuje nalezené zranitelnosti v softwarových produktech. Každá zranitelnost obsahuje detailní informace o tom, jaké konkrétní knihovny a verze se týká, co může způsobit a jak ji lze vyřešit (jestli už je dostupná verze, která danou zranitelnosti neobsahuje, případně kdy je očekáváná). [18]

2.3.3 Smoke testy

Dalším důležitým typem testů jsou tzv. smoke testy. Jedná se o testy, které pokrývají základní funkcionality software. Sada testovacích případů pro smoke testy by měla pokrývat nejdůležitější části software. Tyto testovací případy by měli být schopny otestovány rychle a efektivně, a to po každé změně, opravě chyby anebo aktualizaci ve software, proto je zde na místě automatizace těchto testů. Testovací případy obsažené v této sadě jsou pouze happy day typu (2.1.5), tzn. obsahují pouze pozitivní scénáře využívání software, nezkoumají okrajové případy anebo podmínky. Jejich cílem je potvrzení, že základní funkcionality fungují, a tudíž je možné provádět základní operace.

Hlavním cílem smoke testů je rychlé ověření, zda nové změny, opravy nebo aktualizace nemají zásadní dopady na základní funkcionalitu software. [1]

2.3.4 Regresní testy

Regresní testy, funkční testy a smoke testy mezi sebou dělí tenká hranice. Regresní testy většinou představují určitou vybranou množinu z funkčních testů a jejich

testovacích případů. U funkčních testů jde o ověření celkové funkčnosti software a všech jeho případů užití, u smoke testů o ověření, že základní funkce fungují správně po implementaci změny anebo opravy.

Regresní testy se zaměřují na konkrétní funkce software, ověření jejich stability po implementaci změn a jejich cílem je prevence odchylek existujících funkčností během průběhu vývoje. Většinou se jedná o tzv. selektivní testování, přičemž jsou vybrány části software, které mohli být ovlivněny implementovanou změnou anebo opravou a testy se poté zaměřují na testovací případy identifikované oblasti.

Podobně jako smoke testy by měli být regresní testy vykonané po každé změně, opravě anebo aktualizaci v software. Zatímco smoke testy hodnotí, že klíčové funkcionality fungují správně, regresní testy ověří, že poslední implementace neovlivnili ostatní funkcionality software. [1]

2.3.5 Unit testy

Unit testy, jak již z názvu plyne, jsou zaměřené na otestování nejmenší testovatelné jednotky kódu. Ve většině případů je možné tyto testy spustit bez spuštění celého software. Jejich cílem je ověřit, že každá část kódu pracuje správně, a to i izolovaná od zbytku systému včetně databáze anebo sítě. Díky logice izolovanosti pak víme, že výsledky testů jsou nezávislé na stavu jiných části kódu. Každý testovací případ ze sady unit testů, obsahuje konkrétní vstupy a očekáváné výstupy. [1] [3]

Unit testy jsou automatizované testy, které se spouští po každé změně v dané části kódu. Existuje několik frameworků, které lze pro unit testy použít na základě jazyku kódu:

- JUnit – Java
- NUnit – .NET
- Mocha – JavaScript
- PyTest – Python
- …

2.4 Typy testů podle provedení

Dalším důležitým dělením testů je podle toho, jak jsou vykonány – manuálně anebo automaticky. V následujících dvou podkapitolách si shrneme principy, výhody a nevýhodou obou typů.

2.4.1 Manuální testy

Manuální testy jsou vykonány testery, na základě připravených testovacích případů. Jak již bylo zmíněno, manuálně lze vykonávat funkční testy a malou množinu nefunkčních testů.

Tester prochází testovací případy a jejich dílčí kroky a výsledky zaznamenává do nástroje, který je ke správě testovacích případů použitý (např. MS Excel anebo PractiTest).

Nevýhodou manuálních testů je jejich časová náročnost a možná chybovost způsobená lidskou nepozorností. Na druhé straně, manuální testy jsou pořád efektivnější v hledání chyb. Ze všech nalezených chyb na konci testování, až 70 % z nich bývá odhaleno během manuálních testů. [1]

2.4.2 Automatizované testy

Automatizované testy jsou vykonávány pomocí automatizovaných nástrojů anebo skriptů. Jeden z typů automatizovaných testů jsme si již zmínili – Unit testy (2.3.5), které se zaměřují na nejmenší testovatelnou jednotku. Automatizované testy vytvořené ve speciálních nástrojích ale umožňují vytvoření a zautomatizování testů celého software či už se jedná o testy uživatelského rozhraní anebo nefunkčních požadavků.

Nevýhodou u automatizovaných testů je časová náročnost jejich přípravy. Je nutno zvážit, jestli se vzhledem k velikosti projektu a náročnosti testů oplatí testy automatizovat anebo ne.

Výhodou je rychlost jejich provedení a eliminace lidských chyb spojených s manuálním testováním. Nicméně, lidské chyby se můžou objevit samozřejmě i v automatizovaném testu, který je taky vytvářen lidskými zdroji. Další výhodou je možnost paralelního provedení testů na více zařízeních nebo prostředích což opět vede k ušetření času stráveným testováním. Naopak od manuálních testů, pomocí automatizovaných jsme schopni pokrýt větší množinu nefunkčních požadavků. [3] [19]

2.4.3 Kombinované testy

Často využívaným kompromisem mezi manuálními a automatizovanými testy je jejich kombinace. Jestliže velká část testovacích případů obsahuje opakující se kroky, které jsou časově náročné (např. nahrání dat do software), můžeme automatizovat vykonání

těchto kroků a ostatní pak vykonat manuálně. Výhodou takové kombinace je úspora času testera a příprava ani údržba takového automatizovaného scénáře není až tak časově náročná.

2.5 Akceptační typy testů

V předchozích částech textu jsme si definovali různé typy testů podle jejich obsahu anebo způsobu provedení. V následující kapitole si definujeme typy testů podle toho, kdo a kdy je provádí. Jak již bylo uvedeno, každý software má od zákazníka definovaná určitá akceptační kritéria, která by měla být součástí specifikace a testovací strategie. Jedná se podmínky, za kterých bude software zákazníkem schválen a tím bude splněn důležitý milník v životném cyklu vydání software.

2.5.1 Factory acceptance testy

Factory acceptance testy, FAT, jsou testy, které probíhají na straně dodavatele software. Interní testery testují na interním prostředím. Z hlediska typů testů by v této fázi měli proběhnout celkové funkční (regresní) testy a testy nefunkčních požadavků, které můžou obsahovat i zátěžové a penetrační testy. Konkrétní obsah testování v této fázi je vždy definován již během přípravy testovací strategie, která je poté schválena se zákazníkem.

Na základě výstupů ze všech testů provedených v rámci FAT se zhodnotí, jak jsou splněny akceptační kritéria na software. V případě, že jsou kritéria splněna, můžeme FAT fázi ukončit, software dodat zákazníkovi spolu se zpracovanými výstupy z testů a pokračovat další fázi, kterou je SAT. [1] [11] [20]

2.5.2 Site acceptance testy

Site acceptance testy, SAT, jsou testy, které můžou probíhat na straně dodavatele nebo na straně klienta. Může se jednat o společné testování, případně testování klienta s podporou dodavatele. K testování v této fázi bývá využito SIT prostředí, které obsahuje integrace na další software a komponenty, které se software komunikují.

SAT fáze testování není povinnou fázi a může být vynechána, záleží na komplexitě software nebo projektu celkově, složení týmů na straně klienta apod. Často platí, že strana klienta není zároveň i stranou uživatele. V takovém případě by měli být SAT nezbytnou části.

Na základě výsledků testů v této fázi opět probíhá vyhodnocení neboli srovnání s akceptačními kritérii. Může se stát, že během SAT fáze jsou případně nalezené chyby rovnou opravovány a nasazovány na SIT prostředí formou hot fixů anebo patche. [1] [11] [20]

2.5.3 User acceptance testy

User acceptance testy, UAT, jsou nejdůležitější z fáze testování. Jedná se o testy klientem (budoucím uživatelem) na jeho vlastním prostředí, bez zásahů týmu dodavatele. Klient testuje se svými vlastními testovacími případy, daty, případně i nástroji. Na základě vyhodnocení těchto testů se rozhodne, zda se bude pokračovat v plánu a dodá se software na produkční prostředí klienta. Ukončení fáze UAT je důležitým milníkem v dodávce software celkově, jelikož je zpravidla část platby, anebo platba celá, uhrazena právě po úspěšném ukončení UAT, tzn. s výsledky testů, které splňují akceptační kritéria. [1] [11]

3. POPIS SOUČASNÉ SITUACE

V následujících kapitolách si zmapujeme, jak probíhají jednotlivé procesy spojené s testováním software na projektu PQDD v nejmenované společnosti. Analyzujeme si současné procesy tak, abychom z nich vytáhli silné i slabé stránky, na kterých budeme dále stavět při přípravě návrhů úprav v těchto procesech.

Než začneme s konkrétními procesy, představíme si stručně projekt PQDD nejmenované společnosti. V rámci projektu PQDD byl vyvinutý stejnojmenný software, který je od roku 2013 nasazený na produkčním prostředí klienta a naše nejmenovaná společnost poskytuje podporu tohoto software a jeho kontinuální vývoj, který probíhá formou dvou až tří nových dodávek verzí ročně.

PQDD zpracovává a zobrazuje data o kvalitě a kvantitě elektrické energie na téměř celém území Evropy. Takto zpracovaná data (agregované modely) jsou poté výchozím bodem pro následné podnikové procesy včetně plánování. V rámci této práce, zejména z důvodu anonymizace, nebudeme zacházet do detailů funkcionalit software a jeho uživatelů. Na tomto místě je pro nás důležité vědět, že uživateli software jsou operátoři pracující pro jednotlivé evropské TSO (Transmission Systém Operator, operátor přenosové soustavy) a RCC (Regional Coordinator Center, regionální koordinační centra). PQDD je integrován s dalšími čtyřmi systémy, které poskytují do PQDD relevantní data. Mezi hlavní funkcionality PQDD patři validace přijatých dat, jejich vizualizace z různých pohledů detailu, reporting, dokument management systém a správa uživatelů.

Tým PQDD se skládá z následujících rolí:

- Projekt Manager (1)
- Software Architekt (1)
- Vývojáři (2–3)
- Business Analytik (1)
- Tester (1-2)

3.1 Práce s požadavky na SW

Požadavky na změny v PQDD anebo na implementaci nových funkcionalit softwaru PQDD jsou zadávány průběžně, a to buď ze strany samotných uživatelů software, klientem anebo členy PQDD týmu na straně dodavatele. Jak jsme již zmínili při

představení samotného PQDD projektu, je integrován s dalšími čtyřmi systémy. Klient musí při každé změně v těchto systémech zvážit i dopad na PQDD. Z těchto integračních důvodů plyne značná část změnových požadavků od klienta. Vzhledem k tomu, že se pohybujeme v energetickém sektoru, značná část změnových požadavků vyplívá z legislativních změn, které ovlivní například data, která vstupují do PDQQ software a vyústí tak ve změny, aby mohla být data správně zpracována a zobrazena. Další změny pramenící z legislativních úprav bývají třeba změny ve výpočtech, vyhodnocování určitých hodnot anebo úprava výstupných reportů, které se můžou v PDQQ generovat.

Jak jsme již zmínili, PQDD je na produkčním prostředí klienta aktivně využíván od roku 2013. Každým rokem roste objem vstupujících dat, přistupujících uživatelů, náročnost výpočtů a množství vyhodnocovaných a zobrazovaných aspektů dat. Kvůli tomu je nezbytné pracovat i na nefunkčních požadavcích, aby PQDD fungoval v definované kvalitě i při pořád rostoucím objemu dat, četnosti jejich nahrávání a zobrazovaných funkcích.

3.1.1 CR Dokument

Na základě toho, odkud pramení změnový požadavek, liší se často i jeho forma. Zadávání změnových požadavků od klienta obvykle probíhá formou **CR (change request) dokumentu**, který obsahuje informace o změně na různých úrovních detailu. Dokument má své identifikační číslo, úvodem obsahuje pozadí toho, co stojí za požadovanou změnou (např. změna v legislativě), jaký je dopad na software (např. zmíněna změna v legislativě vyžaduje úpravu vstupných dat), popis současné implementace a popis nově požadované implementace. Dokument obsahující zmíněné části je předán na klíčové lidi PQDD projektu, kteří požadavek zanalyzují a na základě toho dokument doplní o části jako výstupy z analýzy, obsah změny (co všechno bude potřeba kvůli požadované změně upravit), doporučení (PQDD nemůže sám upravovat popis požadované změny, může ale dodat doporučení, co by bylo vhodné spolu s touto změnou implementovat, např. úprava GUI aby byl nový typ dat zobrazen přehledně). Dokument je takhle i několik krát vyměněn mezi klientem a PQDD týmem. V bodě, kdy jsou všechny strany spokojené s popisem dané změny, PQDD tým přidá do dokumentu kapitolu s časovými, a tudíž i finančními odhady na implementaci změny podle finálního zadání. Takhle finální dokument je poté podepsán klíčovými osobami a již se nemůže upravovat. CR dokumenty jsou spravovány na platformě Sharefile

během celého jejich životního cyklu. Na základě důležitosti změny a finančních odhadů se poté klient rozhodne, do které verze PQDD bude chtít změnu implementovat.

3.1.2 JIRA Ticket

Další formou, kterou jsou požadavky zadávány je **JIRA ticket**. Ty jsou využívanou formou zejména ze strany samotných uživatelů. Buď to se jedná o přímý požadavek na změnu anebo ústí z jiného ticketu, který byl založen třeba na podezření na chybu, případně je výsledkem diskuse s PQDD týmem. Takto založený požadavek nemá vždy dostatečnou formu, na rozdíl od zmíněného CR dokumentu. Veškeré diskuse týkající se požadované změny a získávání více informací ze strany zadavatele se odehrává v ticketu, aby byly relevantní informace na jednom místě. Bohužel, často to končí nepřehledným ticketem. V bodě, kdy už ticket na změnu obsahuje i informace o časové náročnosti, předá se na klienta a opět probíhá na jeho straně zvážení, do které verze PQDD bude změna implementována a jestli vůbec. Tímto stylem vzniká v JIRA tzv. backlog, skupina ticketů typu změnového požadavku, které byly schváleny klientem, nicméně, ještě nepadlo rozhodnutí, do které verze PQDD bude implementován.

Forma JIRA ticketu je využívána i při zadávání podnětů na vylepšení anebo změny od PQDD týmu samotného. Proces probíhá stejně jako v předchozím případě, ticket končí v sekci backlog.

Plánování obsahu nové verze poté probíhá následovně. Klient na základě toho, jaký budget má k dispozici na novou verzi a jaká je priorita jednotlivých požadavků stanoví, které změny jsou tzv. must-have. Pokud po započtení těchto změn zbude budget i na jiné změny, přecházíme k revizi sekce backlog a PQDD tým společně s klientem diskutuje o dalších změnách, které by bylo možné a vhodné do nové verze zařadit. Zařazování změn z backlogu může mít spojitost i s tím, jakou část software budeme upravovat a tím pádem i více testovat, kvůli již schváleným změnám. Tudíž některé změny by v určité kombinaci zabrali míň času, než kdyby byly implementovány separátně.

3.1.3 Silné a slabé stránky

Takhle nastavený proces práce s požadavky má svoje silné i slabé stránky:

- Silné stránky
    - CR dokumenty – jejich úroveň detailu a kooperace mezi klientem a dodavatelem při jejich přípravě,
    - Backlog – možnost průběžného zadávaní změnových požadavků a jejich pravidelná revize s klientem,
- Slabé stránky
    - CR dokumenty (Sharefile) vs JIRA tickety
        - nekonzistentnost ve formě zadávání a uchovávaní požadavků,
    - JIRA tickety – vzhledem k množství informací a zachycených diskusí se často stávají nepřehledné.

3.2 Příprava a plánování testů SW

Proces přípravy a plánování testů v rámci projektu PQDD probíhá na základě charakteru a ovlivněných oblastech změnovými požadavky, které byly začleněny do nové verze.

Již při vývoji první verze PQDD byly sepsány testovací případy na každou dílčí funkcionalitu software, které jsou dále každou změnou upravovány a doplňovány. Testovací případy jsou uchovávané v souboru xslx, která je pro uchovávání a aktualizaci testovacích případů v pořádku, nicméně, můžou nastat problémy způsobeny verzováním souboru, kdy více členů týmu provede v souboru změny lokálně ve svém počítači a tyto změny pak nejsou sjednoceny do jednoho souboru. Za udržování testovacích případů je primárně odpovědný business analytik, ale úpravy provádí i tester.

Smoke testy obsahují výběr několika testovacích případů ze zmíněného excelovského souboru. Unit testy jsou připravovány vývojářem přímo během fáze vývoje nové verze v nástroji gitlab.

Tudíž na základě obsahu nové verze PQDD probíhá definice toho, jaké testy bude potřeba provést což je zdokumentováno formou testovací strategie, kterou jsme si již představili v kapitole 2.1.3.

Při plánování samotném jsou odhady na testy většinou přidávány na základě konzultace mezi testerem, software architektem, business analytikem a projektovým manažerem. Není aplikováno konkrétní pravidlo, které by definovalo, že na testy by se mělo vyhradit určité procento času odhadovaného na vývoj. Pracuje se zde hlavně

s tím, že víme, že celkové funkční testy (všechny testovací případy) zaberou testerovi přibližně 35 hodin. Při odhadech se bere v potaz i čas na úpravu a přípravu nových testovacích případů pramenících z nově přidané funkcionality anebo naopak, z vyloučení určité funkcionality.

Výstupem z plánování testů je sada dokumentů, které jsme si představili již v kapitole 2.1. Patří mezi ne testovací strategie, test plán, testovací případy, testovací data a testovací prostředí.

3.2.1 Silné a slabé stránky

Pro tento proces si taky na základě předchozího popisu uvedeme jeho silné a slabé stránky:

- Silné stránky
    - Zapojení všech klíčových rolí na projektu do procesu plánování (projekt manager, software architekt, business architekt, tester)
    - Využívání a správa prostředí pro vývoj a testy
- Slabé stránky
    - Testovací případy v Excelu – stává se, že jednotliví členové týmu nemusí mít aktuální verzi a že všechny úpravy se nedostanou do sjednoceného souboru
    - Absence role test managera – stává se, že jeden člen týmu připravuje nové anebo upravuje stávající testovací případy, které poté sám testuje a vyhodnocuje

3.3 Testování SW

Základ procesu testování nové verze PQDD je při každé verzi téměř stejný. Spočívá v Unit testech vývojářů, testech nových změn a oprav samostatně, poté se pokračuje regresními testy pokrývajícími celou aplikaci a na závěr jsou provedeny scany kvality kódu.

Při samotném programování vývojáři připravují a spouštějí unit testy na kontrolu upravené anebo nově vytvořené části kódu. V bodě, kdy je připravena kompletní změna, anebo její samostatně testovatelná část v případě rozsáhlejších změn, vývojáři nasadí tuto část na testovací prostředí, kdy ji můžou testeři otestovat podle relevantních testovacích případů. I když projekt funguje tradičně řízeným způsobem (vodopád),

tímto způsobem je eliminováno riziko velkého počtu nalezených chyb během testovací fáze. Když je připravená kompletní nová verze, připraví se instalační balíček, který si testeři nasadí na testovací prostředí samy, jelikož i manuální instalace software je součástí testovacích případů. Poté se začíná smoke testem verze a pokračuje se dalšími testy podle obsahu nové verze, jak jsme si již popsali v kapitole 2.3. V případě potřeby penetračních a/nebo výkonnostních testů, nasazuje se verze i na prostředí k tomu určené. V našem případě se jedná o PERF prostředí, které je interní a integrované s nástroji na vykonání těchto testů.

Smoke, funkční i regresní testy jsou vykonávaný manuálně testery. Tudíž automatizování testovacích případů na projektu PQDD zatím neproběhlo. Vzhledem k tomu, že PQDD validuje a zobrazuje data přijaté z jiné komponenty, 80 % testovacích případů začíná krokem nahráním dat. Komponenta, která data odesílá umožňuje nahrání pouze jednoho souboru na jedno odeslání. Při každém nahrávání je nutné určit typ zprávy, kterou posíláme a cílové prostředí. Modely, které PQDD zobrazuje jsou většinou agregací třech až sedmy souborů. Tudíž, nahrávání dat je nejvíc náročným krokem u většiny testovacích případů.

Níže si určíme, jaké silné a slabé stránky definujeme u stávajícího procesu testování SW:

- Silné stránky
    - Testování dílčích částí nové verze již během fáze vývoje
    - Dedikované interní prostředí pro vývoj, funkční testy a výkonnostní nebo penetrační testy
- Slabé stránky
    - Kompletně manuální vykonávání funkčních testů, a to i kroků, které se opakují v 80 % testovacích případů
    - Absence role test managera – jeden člen týmu vykonává testy, které poté sám i vyhodnocuje

4. CÍLE NOVÉHO POSTUPU

Cílem nového postupu je zejména optimalizace procesů spojených s testováním. Usnadnění a urychlení procesu plánování obsahu nové verze PQDD a tím získat více času na samotný vývoj a testy nové verze. Zvýšení přehlednosti, a to na takovou úroveň, že při změně osob na klíčových rolích projektu nebude obtížné je do projektu zapojit. Toho chceme docílit hlavně eliminováním výjimek a nekonzistencí, které momentálně v procesech jsou a striktním rozdělením odpovědností mezi členy týmu PQDD.

Na základě stávajících procesech spojených s testováním nové verze software si nyní stanovíme, co by bylo vhodné změnit u kterého z již definovaných procesů abychom byli schopni dosáhnout zmíněných cílů. V přechozích kapitolách jsme si u každého z procesů definovali silné a slabé stránky. Při sestavování upravených procesů se budeme soustředit primárně na eliminaci slabých a posílení silných stránek.

5. NÁVRH NOVÉHO POSTUPU

Následující kapitoly obsahují popis úprav, které navrhujeme v stávajících procesech na projektu PDQQ za účelem dosažení cílů, které jsme si definovali v předchozí kapitole.

5.1 Práce s požadavky na SW

V kapitole 3.1 jsme představili stávající proces práce s požadavky a sestavování obsahu nové verze PQDD. Pomocí úprav v procesu budeme cílit na odstranění slabých stránek prostřednictvím posílení silných stránek.

Silnými stránkami jsou již využívané CR dokumenty, které mají detailní a zároveň přehlednou strukturu a vznikají formou spolupráce mezi klientem a klíčovými lidmi na projektu PQDD. Tímto stylem je eliminováno riziko, že se na určité aspekty anebo vlivy změny u analýzy a odhadů pracnosti zapomene. Proto chceme takhle definované CR dokumenty zavést i pro změny, které jsou navrženy uživateli a momentálně jsou zakládány formou JIRA ticketů. Vzhledem k tomu, že příprava kvalitního CR dokumentu je časově náročná a toho, že JIRA tickety na změny jsou zakládány uživateli bez předchozí konzultace s relevantními lidmi na straně klienta, příprava CR dokumentu pro každý ticket typu změnového požadavku by nebyla optimálním řešením. Z tohoto důvodu přidáme do procesu potvrzení změnového požadavku relevantní osobou na straně klienta, které bude podmínkou pro přípravu CR dokumentu. Takhle bychom eliminovali jednu ze slabých stránek, kterou je nekonzistence mezi změnovými požadavky. Obdobně chceme přistupovat i k CR dokumentům, které jsou udržovány v rámci platformy Sharefile a ke každému z nich zakládat odpovídají JIRA ticket. Tímto způsobem bude řízení backlogu a obsahu verze v JIRA transparentnější a přehlednější. Takhle založený JIRA ticket nemusí obsahovat popis změnového požadavku do detailu, stačí základní popis a odkaz na CR dokument. Tímto způsobem bude pak možné lépe řízení vývoje i testů pomocí dashboardů v nástroji JIRA, které jsou již využívané, ale nejsou objektivní vzhledem ke zmíněné nekonzistenci.

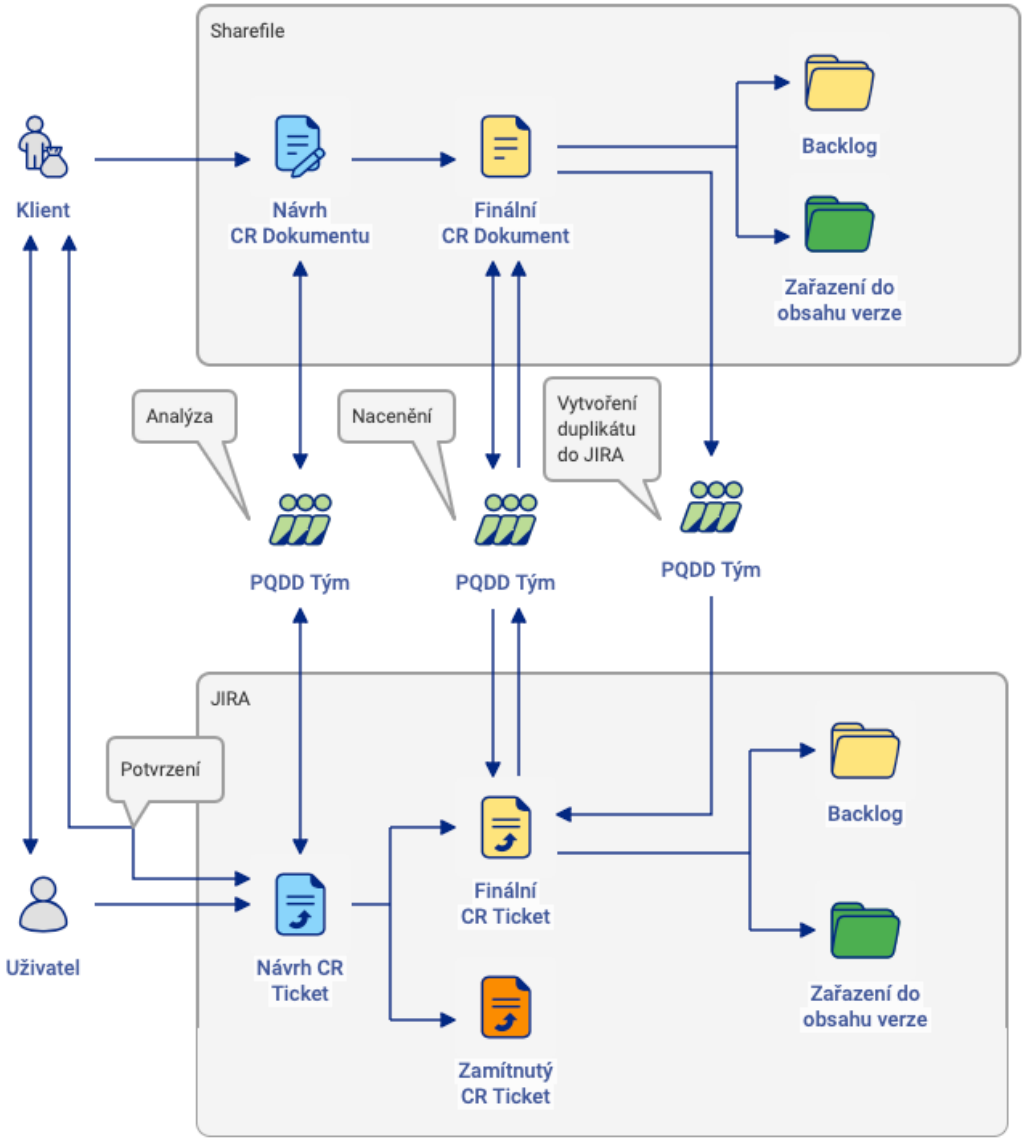
5.1.1 Časová náročnost

Zmíněné změny nepředstavují velkou časovou náročnost, ale sníží množství času na řízení přípravy nové verze a backlogu pro budoucí verze. Vytvoření JIRA ticketu pro

již existující CR dokument je úkol v řádu minut. V opačném případě, kdy bude CR dokument vznikat na základě JIRA ticketu to bude trochu náročnější, ale v průměru mluvíme o jedné hodině, vzhledem k tomu, že již v JIRA ticketu by měl být změnový požadavek popsaný v dostatečné úrovni detailu. Tudíž půjde zejména o přepsání informací do jiné struktury.

5.1.2 Procesní mapa

Následující diagram graficky popisuje proces práce s požadavky na PQDD software spolu s navrženými úpravami. Jak je znázorněno, PQDD tým koriguje konzistentnost mezi Sharefile, kde jsou spravovány CR dokumenty, a JIRA tickety. Klient připraví návrh změnového požadavku formou CR dokumentu, který zveřejní na Sharefile a informuje o něm PQDD tým, který CR dokument zreviduje a doplní o výstupy z analýzy, doporučení, náročnost změny a případné komentáře. Takto upravený CR dokument PQDD tým zveřejní jako další verzi na Sharefile. Klient upravený dokument zreviduje, případně zapracuje navržené změny anebo komentáře od PQDD týmu a vznikne tak finální verze CR dokumentu, který se opět dostává do rukou PQDD týmu kvůli doplnění finančních odhadů na danou změnu. Změnový požadavek je buď to začleněn do části Backlog anebo je rovnou zařazen do konkrétní verze PQDD, a to podle jeho priority a finanční náročnosti. K finálnímu CR dokumentu poté PQDD tým vytvoří odpovídající ticket v JIRA. V dalším případě návrh na změnu přichází od uživatele PQDD formou JIRA ticketu, který je poté analyzován PQDD týmem a schválen klientem. Na základě toho je ticket buď to zamítnutý anebo se z něj stává finální ticket, na kterém se dále pracuje obdobně jako v případě CR dokumentu. PQDD informace z tohoto ticketu vloží do šablony CR dokumentu, který zveřejní na Sharefile.



Obrázek 3: Diagram procesu práce s požadavky na SW

5.1.3 Matice odpovědností

Tabulka 1: Práce s požadavky – RACI matice

Aktivita / Role	Klient	Uživatel	PM	SWA	BA	DEV	TST
Návrh změnového požadavku formou CR dokumentu	R; A	C	I	I	I	I	I
Doplnění výstupů z analýzy do CR dokumentu	I	I	A	R	R	C	C
Doplnění časových odhadů do CR dokumentu	I	I	A	R	R	I	I
Nacenění změnového požadavku do CR dokumentu	I	I	R; A	C	C	I	I
Vytvoření odpovídajícího ticketu na CR dokument	I	I	I	C	R; A	I	I
Zařazení CR do backlogu anebo konkrétní verze	R; A	I	C	C	C	I	I
Potvrzení změnového požadavku v JIRA	R; A	C	I	I	I	I	I
Příprava CR dokumentu podle JIRA ticketu	I	I	A	C	R	I	I

5.2 Příprava a plánování testů SW

Ve stávajícím procesu přípravy a plánování testů identifikujeme slabiny, především v souvislosti s udržováním testovacích případů v excelovském souboru a v absenci striktního rozdělení odpovědností mezi členy týmu PQDD.

Existuje několik možností, jak tuto oblast zefektivnit a zajistit bezpečnější a přehlednější správu testovacích případů. Mezi tyto možnosti patří využití produktu PractiTest od společnosti Atlassian, který poskytuje integrované možnosti a umožňuje propojení s produktem JIRA, používaným v projektu PQDD k evidenci změn, chyb a dalších aktivit.

Implementace PractiTestu do projektu PQDD přináší výhody v podobě širší dostupnosti členů týmu, což usnadňuje úpravy testovacích případů a zároveň poskytuje manažerovi projektu a vedení přehled o stavu testů. Vzhledem k partnerství naší společnosti s Atlassian, které vychází z využívání produktu JIRA, a díky dřívějšímu úspěšnému nasazení PractiTestu v jiných projektech, je cena této integrace přijatelná a zřízení nového projektu v této platformě je časově nenáročné. Proto jsme se rozhodli migrovat testovací případy právě do tohoto nástroje.

PractiTest nejen eliminuje riziko ztráty úprav, ale také usnadňuje analýzu výsledků testů. Díky propojení s JIRA projektem PQDD je možné výsledky testů automaticky přiřazovat k relevantnímu ticketu, což je zejména u dílčích testů opravy chyb nebo nových implementací velmi efektivní. Grafy a přehledy provedených testů v rámci regresních nebo smoke testů umožňují vytvářet výstupní reporty s klíčovými informacemi pro celý tým.

Druhá identifikovaná slabá stránka spočívá v nejasném rozdělení odpovědností. Stejně tak, jak není vhodné, aby programátor testoval sám svůj kód, není vhodné ani aby jeden člověk testy naplánoval, připravil, vykonal i vyhodnotil. Vzhledem k tomu, že business analytik má detailní know-how o tom, jak má software fungovat, přijde nám vhodné, aby byl za přípravu testů a závěrečné vyhodnocení testů odpovědný právě on.

5.2.1 Časová náročnost

Testovací případy z excelu do PractiTestu není možné importovat, ale je nutné je přepsat. Momentálně obsahuje testovací sada 107 testovacích případů. Některé z nich je možno přepsat za jednotky minut, jiné můžou trvat i 15 až 30 minut. Na základě toho odhadujeme, že migrace testovacích případů ze současně využívaného excelovského souboru do PractiTestu zabere přibližně 27 člověkohodin. Vzhledem k tomu, že PQDD probíhá formou kontinuálního vývoje, tudíž nečekáme, že dodávání nových verzí software by mělo být v horizontu následujících 3 let ukončeno, investovaných 27 hodin do migrace se na tomto místě vyplatí.

Jelikož manuální migrace testovacích případů není nijak náročná, může se stát úkolem pro nového juniorního testera, který si tímto úkolem osvojí testovací případy a rychleji získá deatilnější povědomí o fungování software PQDD. Dalším neméně důležitým aspektem je to, že jeho sazba je pro společnost na takový úkol mnohem výhodnější než v případě, kdyby migraci dělal seniorní člen týmu.

5.2.2 Procesní mapa

Vzhledem k tomu, že využívání PractiTestu není změna, která by měnila nastavení procesu, následujícím diagramem jsme zformalizovali proces plánování testování software PQDD.

Graficky znázorňujeme to, co již bylo řečeno v předchozích kapitolách. Jak typy požadavků vstupují do procesu plánování testů, jaké typy testů je nutné provést a co je výstupem plánování – test plán a testovací strategie. Takhle připravený diagram usnadní definování potřebných oblastí, nad kterými je potřeba se při plánování zamyslet.

Odpovědnosti za zmíněné aktivity v diagramu jsou definovány v matici odpovědností, která je v následující kapitole.

Horní část mapy zobrazuje aktivity a výstupy, které se připravují na základě charakteru požadavku na software. V případě **funkčního typu požadavku** (BR) se musíme zamyslet nad tím, jaký má vliv na případy užití (UC). Tyto informace jsou obvykle výstupem z analýzy požadavků. Pokud se jedná o nové případy užití, které musí být vytvořeny v rámci implementace změny, musí se to zohlednit i v testovacích případech (TC). Každému případu užití by měli odpovídat minimálně 2 testovací případy (happy

day a alternativní) a každý z nich má obsahovat i data na jejich nasimulování. Tudíž, pokud požadavek definuje nové případy užití, musíme počítat s tím, že je potřeba připravit nové testovací případy a nová testovací data. V případě, že požadavek nedefinuje zcela nové případy užití, ale mění ty stávající, je potřeba tyto změny zohlednit i v adekvátních testovacích případech a datech. Ty jsou pak použity zejména pro funkční testy.

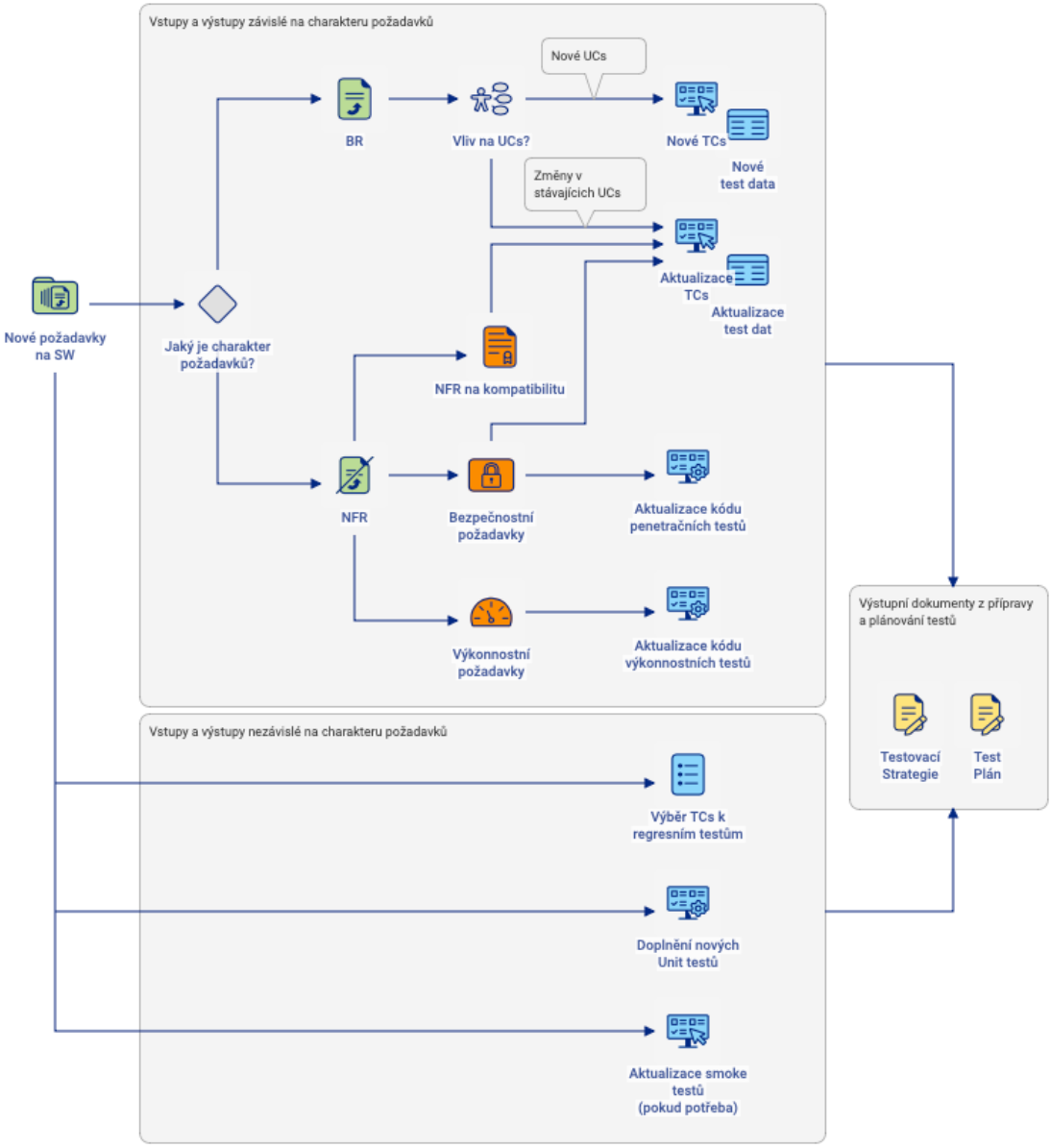
V případě **nefunkčního typu požadavku** (NFR) se musíme dále zamyslet, co je jeho primárním cílem. Pokud požadavek cílí na **změnu v kompatibilitě**, měli bychom to zohlednit pomocí aktualizací v odpovídajících testovacích případech. Jak jsme již zmínili, součástí testovacích případů jsou i předpoklady, které by měli obsahovat i informace o použitém software třetí strany. Například v případě nefunkčního požadavku na kompatibilitu s webovým prohlížečem Safari v jeho 3 posledních verzí, by měly testovací případy, které probíhají v prostředí webového prohlížeče, obsahovat předpoklad, že je použita každá z těchto verzí Safari. V případě požadavků na kompatibilitu software s webovými prohlížeči Chrome, Firefox, MS Edge a Safari a jejich třemi posledními verzemi to znamená, že každý testovací případ musí proběhnout 12krát, s každou podporovanou verzi uvedených webových prohlížečů. Jelikož nám to proces testování může znatelně prodloužit, je proto důležité na to nezapomenout.

Dalším častým typem nefunkčního požadavku je požadavek, který cílí na **bezpečnost software**. Zde je potřeba se zamyslet, o jaký který aspekt bezpečnosti přesně jde. Mezi bezpečností požadavky může patřit například politika hesel (například heslo musí mít minimálně 8 a maximálně 14 znaků, musí obsahovat velká i malá písmena, alespoň jeden speciální znak a alespoň jedno číslo). Takový typ požadavku můžeme zohlednit také v testovacích případech, kde jeden testovací případ ověří chování software při zadání hesla, které respektuje dané podmínky a pak alespoň dva testovací případy, které některou z podmínek nerespektují abychom ověřili, že takové heslo neprojde validací. Na druhou stranu, bezpečnostní požadavek na šifrování dat neotestujeme pomocí klasických, manuálních testovacích případů. V tomto případě je potřeba postupovat úpravou v kódu na spouštění penetračních testů.

Podobně je to u požadavků na **výkon software**, ten nelze pořádně otestovat pomocí testovacích případů, proto je zde nutné upravit kód výkonnostních anebo zátěžových testů podle kritérií daných požadavkem.

Spodní část mapy obsahuje grafické shrnutí aktivit, testů a výstupů, které musí být součástí každého testování nové verze software, a to bez ohledu na charakter požadavků, které jsou implementovány. Jak jsme již zmiňovali v kapitole 2.3, smoke testy (2.3.4), regresní testy (2.3.3) a unit testy (2.3.5) musí být provedeny po každé změně v implementaci. Musí jim samozřejmě předcházet příprava a aktualizace. V případě **regresních testů** si musíme určit, jaké testovací případy jsou anebo by mohly být ovlivněny implementací v rámci nové verze. V tomto případě netestujeme celou sadu testovacích případů, ale jen vybranou množinu. Pokud máme dokumentaci k software pořádně připravenou a udržovanou, tento výběr není náročný vzhledem k tomu, že výstupem z analýzy by mělo vždy být zmapování požadavků, které ukazuje na již zmíněné případy užití anebo část software, které se požadavky dotýkají.

Poslední, avšak neméně důležitá součást je **scan kvality kódu**, který se spustí ve využívaném nástroji po tom, co je hotová implementace a máme finální verzi kódu.



Obrázek 4: Diagram procesu přípravy a plánování testování SW

5.2.3 Matice odpovědností

Jak již bylo zmíněno, PQDD tým nemá člověka na pozici test managera anebo test architekta, proto aktivity, které by měla zastávat tato pozice budou v našem případě vykonávaný business analytikem. V současném procesu se často stává, že tester sám upravuje testovací případy, co chceme eliminovat.

Na druhou stranu software architekt v našem případě zastává pozici leadera pro vývoj a technické části projektu. Jeho aktivity v rámci vývoje spočívají zejména v revizi kódů, unit testů a konzultacích s vývojáři a průběžném hodnocení kvality připraveného kódu formou scanu kvality. Proto právě on získá primární odpovědnost za přípravu náročnějších automatizovaných testů jako jsou penetrační a výkonnostní testy.

Tabulka 2: Příprava a plánování testování – RACI matice

Aktivita / Role	PM	SWA	BA	DEV	TST
Definice nových UCs a/nebo změn ve stávajících UCs	I	C	A; R	I	I
Příprava nových testovacích případů a/nebo změn ve stávajících	I	C	A; R	I	C
Příprava nových testovacích dat a/nebo změn ve stávajících	I	C	A; R	I	C
Aktualizace kódu penetračních a/nebo výkonnostních testů	I	A; R	C	I	I
Příprava nových a/nebo aktualizace stávajících unit testů	I	A	C	R	I
Výběr sady testovacích případů k regresním testům	I	C	A; R	I	I
Příprava testovací strategie a plánu testování	A	C	R	I	I
Příprava testovacího prostředí	I	A; R	I	R	I

5.3 Testování SW

V současném procesu testování identifikujeme slabou stránku neboli příležitost pro zlepšení, v manuálním provedení veškerých testovacích případů, které jsou součástí smoke, funkčních i regresních typů testů. Na základě výše zmíněného a na základě hlavních funkcionalit software PQDD zde navrhujeme automatizaci nahrávání dat do PQDD a tím eliminovat zmíněnou slabou stránku.

Další slabá stránka v současném nastavení procesu testování z našeho pohledu spočívá v nejasném rozdělení odpovědnostní za vykonání a vyhodnocení testů, a to jak funkčních, tak nefunkčních. Opět bude tedy změna spočívat v detailnějším rozdělení odpovědností formou RACI matice. Za funkční testy navrhujeme rozdělení odpovědností mezi testera a business analytika. Za nefunkční testy rozdělení mezi vývojáře a software architekta.

5.3.1 Časová náročnost

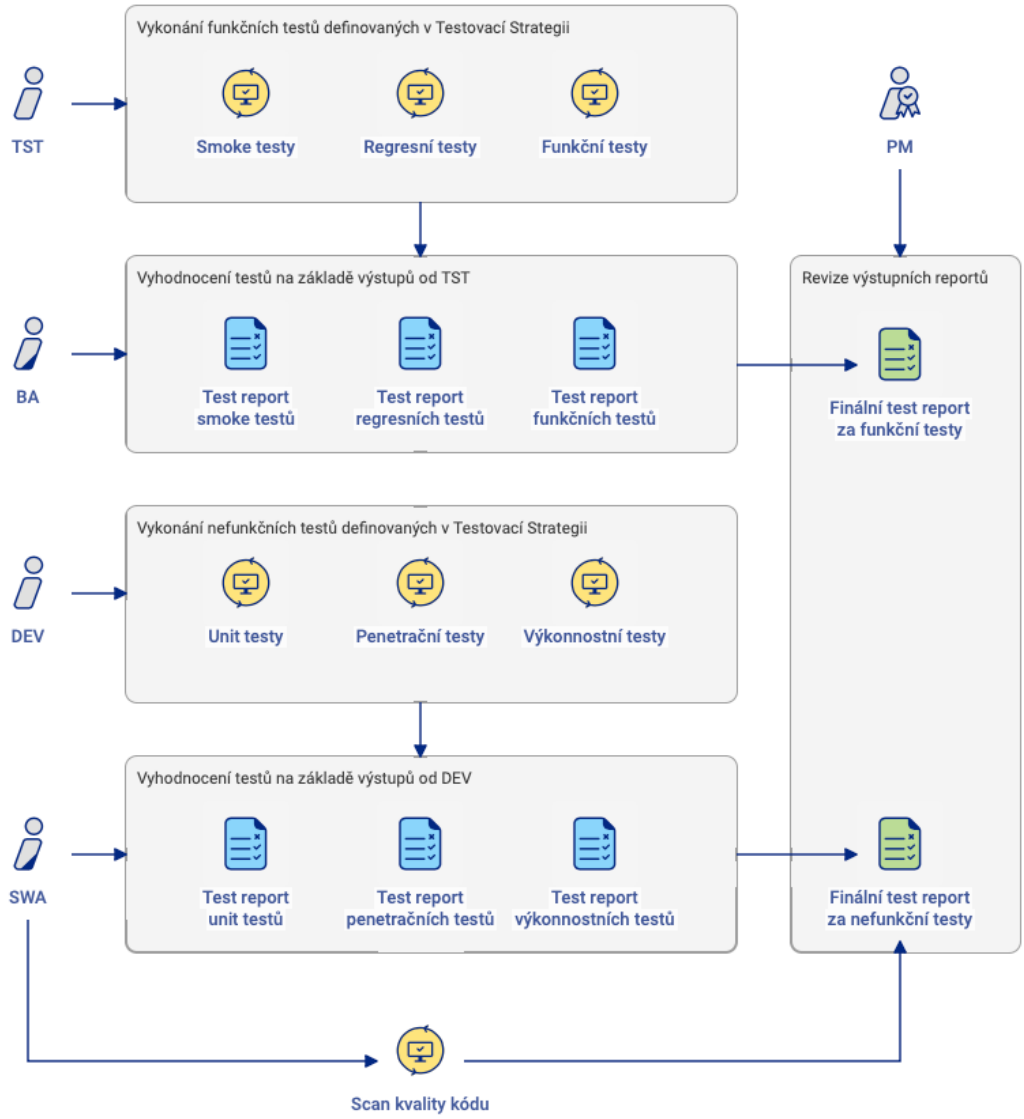
Vzhledem k tomu, že vývojáři poctivě pracují s Unit testy, nebude příprava automatizace této části tak časově náročná, jelikož součástí Unit testů je i nahrávání dat. Rozdílem je, že Unit testy nahrávají data přímo do PQDD, ne přes komponentu, která je k tomu využívaná na produkčním prostředí a momentálně i pro interní testy. Potřebujeme tedy otestovat, že PQDD správně přijímá data z této komponenty. K takovému ověření nám ale stačí provést několik manuálních testů a při dalších testovacích případech je možno data nahrát již automaticky prostřednictvím vybraného nástroje. Pro správnou komunikaci s dalšími komponenty slouží, k tomu určené, integrační testy (SIT).

Vzhledem k tomu, že lidí z projektového týmu mají licenci na nástroj IntelliJ IDEA, bylo rozhodnuto, že pro automatizaci bude využít právě tento nástroj. Tudíž, nevytvoříme dodatečné náklady za nové licence. Na trhu je množství nástrojů pro automatizované testy, které jsou vyvinuty přímo pro tento účel a nabízejí i propojení s PractiTestem a JIRA, nicméně, pro automatizaci nahrávání a vzhledem k potenciálně vzniklým nákladům, je zde nebudeme uvažovat. Co se týče časové náročnosti na přípravu tohoto testu, pohybujeme se v jednotkách hodin. Jelikož členové týmu s nástrojem mají zkušenosti, tudíž čas na zaškolení testerů bude také minimální.

Manuální nahrávání dat trvá v průměru 90 sekund na jeden testovací případ. Nahrávání dat pomocí IntelliJ IDEA trvá v průměru 15 sekund. Před spuštěním se změní pouze odkaz na složku, ze které si má nástroj data vzít, jelikož máme data rozdělená ve složkách ke každému testovacímu případu, tento čas je minimální. Když uvažujeme, že krok nahrání dat obsahuje 75 testovacích případů, které momentálně trvá 90 vteřin, ušetříme pomocí automatizace tohoto kroku téměř 94 minut (5 625 sekund). Tudíž čas věnovaný přípravě automatizovaného nahrávání dat se nám vrátí již během testování jedné verze.

5.3.2 Procesní mapa

V rámci přípravy a plánování testů vznikl plán testů a testovací strategie, které přesně určují, jaké typy testů budou vykonány pro danou verzi PQDD software. Diagram níže zobrazuje, jak je rozděleno jejich vykonání, vyhodnocení a revize. Obsah jednotlivých testů jsme si představili již v předchozí části této práce (2.3). Díky tomu, že jsme do procesu testování zavedli využívání produktu PractiTest, tester nemusí během vykonávání testů připravovat žádný speciální dokument, ale business analytik si výsledky jednotlivých testů najde v PractiTestu a na základě těchto výsledků, grafů a statistik zpracuje dílčí reporty, které poté shrne ve finálním reportu za funkční část testů. Dílčí reporty jsou připravovány zejména z důvodu časové následnosti, jelikož logika posloupnosti typů testů jde zleva doprava. Tester tedy začíná smoke testy, poté pokračuje regresními testy a na závěr vykoná funkční testy. Obdobně je to u nefunkčních testů provedených vývojářem, kde nejdříve probíhají unit testy poté penetrační a výkonnostní (pokud byly zařazeny do testovací strategie a plánu). Nezávisle na zmíněných testech, software architekt spustí scan kvality finálního kódu, kterého vyhodnocení zapracuje do finálního reportu za nefunkční testy. Finální test reporty z obou částí jsou pak revidovány projektovým managerem.



Obrázek 5: Diagram procesu testování SW

5.3.3 Matice odpovědností

Jak již bylo zmíněno u procesu plánování testů, považujeme za vhodné rozdělit aktivity spojené s funkčním testováním mezi testera a business analytika. Business analytik testy plánuje a připravuje, tester je vykonává a na základě výstupů z testů poté business analytik vykonané testy vyhodnotí.

Obdobným způsobem navrhujeme postup i při nefunkčních testech, které jsou vykonávány vývojářem, připraví a vyhodnotí je software architekt.

Na závěr pak probíhá revize finálních reportů projektovým managerem, jak je graficky znázorněno v předchozím diagramu.

Je nutno zdůraznit neustálou komunikaci mezi všemi členy týmu. O průběhu a výsledku každého procesu a testu jsou informování všichni členové týmu.

Tabulka 3: Testování software – RACI matice

Aktivita / Role	PM	SWA	BA	DEV	TST
Vykonání smoke, regresních a funkčních testů	I	I	C	I	A; R
Příprava reportů ze smoke, regresních a funkčních testů	I	I	A; R	I	C
Vykonání penetračních a výkonnostních testů	I	C	I	A; R	I
Příprava reportů z penetračních a výkonnostních testů	I	A; R	I	C	I
Vykonání unit testů	I	C	I	A; R	I
Příprava reportů z unit testů	I	A; R	I	C	I
Příprava, spuštění a vyhodnocení scanu kvality kódu	I	A; R	I	C	I
Revize finálních test reportů	A; R	C; I	C; I	I	I

6. IMPLEMENTACE NOVÉHO POSTUPU

V následující části práce si zavedeme upravené procesy související s testováním software na plánování konkrétního vydání nové verze PQDD.

6.1.1 Obsah nové verze SW

Obsah nové verze software v rámci projektu PDQQ, který byl domluvený klientem i naši nejmenovanou společností. Z důvodu anonymizace nejsou uvedeny detaily požadavků, pouze jejich kódy, které indikují, o jaký typ změny se jedná:

- CR = Change Request, je primárně funkčním požadavkem,
- NFR = nonFunctional Request
- BF = Bug Fix, oprava chyby, která může být jak funkčního, tak nefunkčního charakteru
- SSDLC = Secure Software Development Life Cycle, pod tímhle označením jsou evidovány nalezené zranitelnosti software, které jsou pak eliminovány v každé nové verzi SW. Jsou rozděleny na základě jejich rizika do kategorií critical, high, medium, low. Každá detekovaná zranitelnost má tento stupeň definovaný v databázi zranitelností (NVD).

U každé položky jsme si již připravili odhady pracnosti, které jsou rozdělené do částí:

- ANA = analýza, tzn. člověkohodiny odhadované na analýzu daného požadavku,
- V rámci analýzy probíhá příprava designu změny, definice případů užití, které budou ovlivněny změnou a jak, určení nových případů užití a aktualizace relevantních dokumentů.
- DEV = development, tzn. člověkohodiny odhadované na vývoj daného požadavku,
- MNG = management, tzn. člověkohodiny odhadované na řízení zapracování požadavku a jeho vydání,

Po naplánování testů nám do odhadů ještě přibude položka:

- TST = test, tzn. člověkohodiny odhadované na přípravu, provedení a vyhodnocení testů.

Obsah nové verze SW:

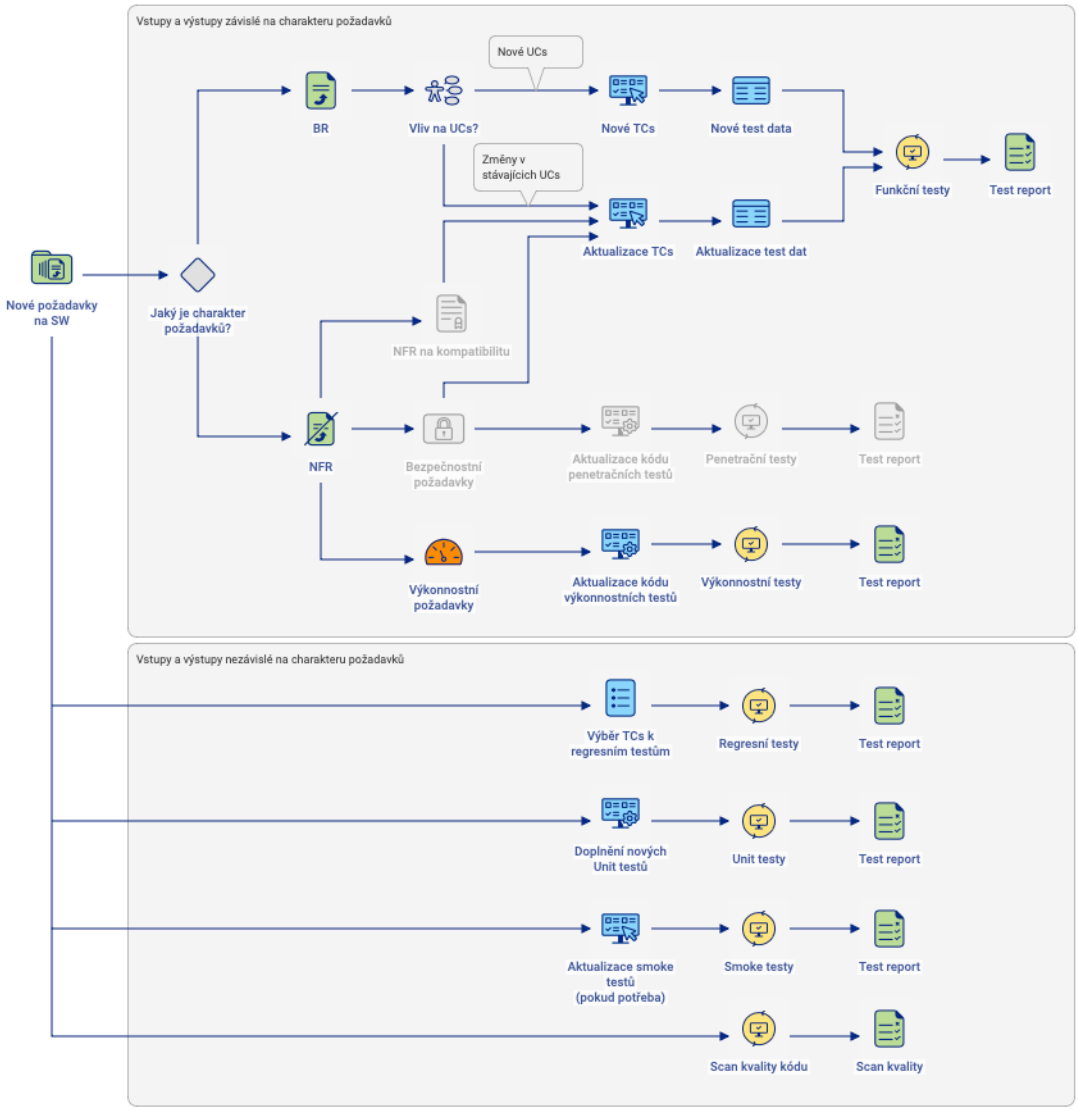
- CR_2024_006
    - Odhad pracnosti:
        - ANA: 16,00 Mhrs
        - DEV: 95,00 Mhrs
        - MNG: 10,00 Mhrs
    - Ovlivněné oblasti:
        - Zobrazení dat, Filtrování dat, Export dat, GUI – dvě obrazovky
        - Stávající případy užití: 6
        - Nové případy užití: 2
- CR_2023_036
    - Odhad pracnosti:
        - ANA: 7,00 Mhrs
        - DEV: 38,00 Mhrs
        - MNG: 4,00 Mhrs
    - Ovlivněné oblasti:
        - GUI – celkový layout
        - Stávající případy užití: 28
        - Nové případy užití: 0
- BF_1292
    - Odhad pracnosti:
        - ANA: 2,00 Mhrs
        - DEV: 6,00 Mhrs
        - MNG: 1,00 Mhrs
    - Ovlivněné oblasti:
        - Reporting – šablona pro jeden report
        - Stávající případy užití: 0
        - Nové případy užití: 0
- NFR_1295
    - Odhad pracnosti:
        - ANA: 4,00 Mhrs
        - DEV: 22,00 Mhrs
        - MNG: 2,00 Mhrs
    - Ovlivněné oblasti:

        - Rychlost odezvy GUI
        - Stávající případy užití: 0
        - Nové případy užití: 0
- NFR_1296
    - Odhad pracnosti:
        - ANA: 2,00 Mhrs
        - DEV: 16,00 Mhrs
        - MNG: 2,00 Mhrs
    - Ovlivněné oblasti:
        - Aplikační logování
        - Stávající případy užití: 1
        - Nové případy užití: 0
- SSDLC
    - Critical: 3
        - Odhad pracnosti
            - ANA: 1,00 Mhr
            - DEV: 2,50 Mhrs
    - High: 4
        - Odhad pracnosti
            - ANA: 1,00 Mhr
            - DEV: 3,00 Mhrs
    - Medium: 42
        - Odhad pracnosti
            - ANA: 4,00 Mhrs
            - DEV: 14,00 Mhrs
    - Low: 1
        - Odhad pracnosti:
            - ANA: 0,25 Mhr
            - DEV: 0,50 Mhr

6.1.2 Plánování testů

V rámci plánování testů byl použitý diagram, který spojuje diagramy připraveny v kapitolách 5.2.2 a 5.3.2. Na diagramu níže, jsou barevně odlišeny (šedě) aktivity

a výstupy, kterým se nemusíme věnovat vzhledem ke charakteru požadavků, se kterými pracujeme.



Obrázek 6: Definice obsahu testů

Na základě toho, že požadavky, které máme obsahují funkční změny, které ovlivňují případy užití a také definují i nové případy užití a jsou mezi nimi také i nefunkční požadavky, můžeme z aktivit a vstupů vyřadit pouze penetrační testy a úpravy testovacích případů kvůli nefunkčním požadavkům na kompatibilitu.

Odhady na celkové trvání stávajících testů máme připravené na základě předchozích vykonávání kompletních testů, které jsme snížili o čas, který bude ušetřen díky automatizace nahrávání dat. Budeme tedy pracovat s těmito odhady (označeny jako „základ“). K tomu si přičteme čas potřebný na jejich úpravu, tvorbu nových testovacích případů a testovacích dat.

- Funkční testy:
    - Základ: 34 Mhrs
        - S ohledem na automatizaci nahrávání dat: 32,5 Mhrs
    - Příprava test reportu: 2 Mhrs
    - Změny na základě 6.1.1: 30,50 Mhrs
        - *2 nové UCs -> 6 nových TCs s daty*
            - *6 Mhrs*
        - *6 upravených UCs -> 18 upravených TCs s daty*
            - *9 Mhrs (0,5 Mhr na úpravu jednoho TC s daty)*
        - *28 upravených UCs (GUI) -> 61 upravených TCs bez dat*
            - *15,5 Mhrs (0,25 Mhr na úpravu jednoho TC bez dat)*
- Výkonnostní testy:
    - Základ: 20 Mhrs
    - Příprava test reportu: 8 Mhrs
- Regresní testy:
    - Základ: N/A (obsah je pokaždé odlišný)
        - Na základě 6.1.1 odhadnuto na 6 Mhrs
            - S ohledem na automatizaci nahrávání dat: 5,25 Mhrs
    - Příprava test reportu: 1 Mhrs
- Smoke testy:
    - Základ: 3 Mhrs
        - S ohledem na automatizaci nahrávání dat: 2,75 Mhrs
    - Příprava test report: 1 Mhr
- Scan kvality kódu:
    - Základ: 1 Mhrs
    - Příprava test reportu: 4 Mhrs
- Unit testy
    - Jelikož jsou unit testy vytvářené a spouštěné během vývoje, odhady jsou započítané již v hodinách za vývoj (DEV)

Na základě dílčích odhadů odhadujeme celkové trvání testů na **108 Mhrs**. Vzhledem k tomu, že máme dva testery na plný úvazek (40 hodin týdně) a že výkonnostní testy a scan kvality kódu budou provedeny vývojáři a software architektem to znamená,

že na testy bychom si měli vyhradit 2 pracovní týdny. Jako rezervu na opravy nalezených chyb a protestování jejich oprav si zpravidla přidáváme jeden týden.

Celkově jsou naše celkové odhady následující:

- Analýza: 37,25 Mhrs
- Management: 19 Mhrs
- Vývoj: 197,00 Mhrs
- Testy: 108,00 Mhrs (s rezervou na opravy zaokrouhleno na 3 týdny)

Složení týmu:

- Projekt manager – 1
- Business analytik – 1
- Software architekt – 1
- Vývojář - 2
- Tester – 2

Celkové odhady s respektem na složení týmu:

- Analýza – 1 pracovní týden
- Vývoj – 3 pracovní týdny
- Testy – 3 pracovní týdny
- Management – hodiny projektového managera budou alokovány průběžně během všech fázi, nemají na plán vliv, pouze na cenu, kterou v této práci řešit nebudeme

Na základě výše zmíněných odhadů a milníků, které byly domluveny s klientem, jsme připravili následující plán dodání nové verze software v rámci projektu PDQQ.

Obrázek 7: Plán dodání nové verze

7. ZÁVĚR

Cílem této práce bylo zmapovat a analyzovat procesy související s testováním software v konkrétní nejmenované společnosti. V této nejmenované společnosti jsme spolupracovali s týmem projektu PQDD, v rámci kterého probíhá kontinuální vývoj, tudíž má své specifika naproti klasickému vývojovému projektu a to zejména v periodicitě a obsahu dodávek nových verzí. Naproti vývojovým projektům trvajícím několik měsíců, zde probíhá vývoj opakovaně dva až tři krát ročně po menších částech. I to je jeden z důvodů proč je potřeba procesy pravidelně revidovat a snažit se nacházet nové příležitosti ke zlepšení a zefektivnění dodávek nových verzí.

Iniciativu najít prostory ke zlepšení v procesu testování jsem si vybrala zejména proto, že tato oblast je v IT světě pořád často opomíjenou oblastí, které není věnován dostatečný čas, příprava a podmínky.

V rámci práce jsme si definovali a stručně přiblížili veškeré aspekty, které mají vliv na testování software. Uvedli jsme, jak se může lišit zadávání a práce s požadavky na software v rámci IT projektů. Uvedli jsme pár nástrojů, které dokážou usnadnit práci s řízením testů, ale i s řízením projektu celkově.

V další části práce už jsme se věnovali testování jako takovému a definovali si různé typy testů na základě toho, jaké mají vstupy anebo výstupy, jak jsou plánované anebo na jakém prostředí probíhají a kdo je vykonává. Co by měli které vstupy a výstupy obsahovat, aby měli jasnou představu a očekávání lidi na straně dodavatele, klienta i budoucí uživatele software.

Testování a celkově vývoj software je tvůrčí proces, a proto neexistuje postup anebo přesná kalkulačka, která by nám po zadání požadavků sestavila konkrétní plán, který by byl 100 % reálně proveditelný a efektivní ve všech typech organizací. Existuje řada postupů a každá organizace anebo vedení projektu si z nich může vzít to, co vyhovuje právě jejich pracovnímu stylu, produktu, vztahu s klientem apod.

Procesní mapy a matice odpovědností, které byly v rámci práce připraveny jsou založeny na fungování konkrétního týmu v konkrétní společnosti. Může však sloužit i jako vodítko pro jiné týmy, aby v rámci pár jednoduchých krocích ověřili, že se nic nevynechalo, že plán na testování počítá se všemi potřebnými vstupy, aktivitami a výstupy.

Po představení procesů a jejich změn proběhlo i následní ověření na plánování konkrétní dodávky nové verze PQDD naší nejmenované společnosti. Na tomto příkladu jsme si dokázali, že nejde o pomůcku, která plánování udělá za nás, ale která nás navede správným směrem a eliminuje riziko, že některé části budou opomenuté. Na základě toho, považuji cíl této práce za splněný.

8. LITERATURA

[1] GLENFORD J. MYERS, TOM BADGETT, a COREY SANDLER. *The Art of Software Testing*. 3. B.m.: Word Association, Inc., [vid. 2023-11-04]. ISBN 978-1-118-13313-2.

[2] ASSARAF, Ariel. This is what your developers are doing 75% of the time. *Coralogix* [online]. 19. únor 2015 [vid. 2023-11-04]. Dostupné z: https://coralogix.com/blog/this-is-what-your-developers-are-doing-75-of-the-time-and-this-is-the-cost-you-pay/

[3] JEZ HUMBLE a DAVID FARLEY. *Continuous Delivery*. Boston: Pearson Education, Inc., [vid. 2023-11-04]. ISBN 978-0-321-60191-9.

[4] *Software Requirements and their types - Coding Ninjas* [online]. [vid. 2023-11-04]. Dostupné z: https://www.codingninjas.com/studio/library/software-requirements-and-their-types

[5] Non-functional Requirements: Examples, Types, Approaches. *AltexSoft* [online]. 26. červen 2022 [vid. 2023-12-14]. Dostupné z: https://www.altexsoft.com/blog/non-functional-requirements/

[6] ATLASSIAN. Introduction to Jira Family. *Atlassian* [online]. [vid. 2023-12-14]. Dostupné z: https://www.atlassian.com/software/jira/guides/more/jira-family

[7] ATLASSIAN. What is Confluence: A Brief Overview. *Atlassian* [online]. [vid. 2023-12-14]. Dostupné z: https://www.atlassian.com/software/confluence/resources/guides/get-started/overview

[8] *Trello 101: How to Use Trello Boards & Cards | Trello* [online]. [vid. 2023-12-14]. Dostupné z: https://trello.com/guide/trello-101

[9] ASANA. Asana Work Management - Features, Uses & Product • Asana. *Asana* [online]. [vid. 2023-11-04]. Dostupné z: https://asana.com/product

[10] *Engineering Requirements DOORS | IBM* [online]. [vid. 2023-11-04]. Dostupné z: https://www.ibm.com/products/requirements-management

[11] ISO/IEC/IEEE 29119-1:2013 Software and Systems Engineering -- Software Testing -- Part 1: Concepts and Definitions

[12] *Software Documentation Best Practices [With Examples]* [online]. [vid. 2023-12-14]. Dostupné z: https://helpjuice.com/blog/software-documentation

[13] *Planning Software Testing: 6 Test Documentation Templates | Scribe* [online]. [vid. 2023-12-14]. Dostupné z: https://scribehow.com/library/test-documentation-templates

[14] Test Documentation in QA: Principles and the Best Practices. *Custom Software Development Company* [online]. [vid. 2023-12-15]. Dostupné z: https://maddevs.io/blog/test-documentation-in-software-testing/

[15] MARGET, Adam. Development and Test Environments: Understanding the Different Types of Environments. *Unitrends* [online]. 23. červenec 2021 [vid. 2023-12-15]. Dostupné z: https://www.unitrends.com/blog/development-test-environments

[16] *Code Quality | GitLab* [online]. [vid. 2023-12-01]. Dostupné z: https://docs.gitlab.com/ee/ci/testing/code_quality.html

[17] *Understand Pentesting vs. Red Teaming* [online]. [vid. 2023-12-01]. Dostupné z: https://www.cyderes.com/blog/penetration-testing-vs-red-teaming

[18] *NVD - General* [online]. [vid. 2023-12-15]. Dostupné z: https://nvd.nist.gov/general

[19] GENE KIM, JEZ HUMBLE, PATRICK DEBOIS, a JOHN WILLIS. *The DevOps Handbook*. B.m.: IT Revolution Press, LLC, nedatováno. ISBN 978-1-942788-07-2.

[20] *FAT and SAT: What is the difference and why are they so important?* [online]. [vid. 2023-12-06]. Dostupné z: https://blog.pqegroup.com/commissioning-qualification/fat-and-sat