Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačních technologií.

Bakalářská práce

Automatizace procesů v IT

Denisa Tomková

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Denisa Tomková

Informatika

Název práce

Automatizace testování softwaru

Název anglicky

Software automated testing

Cíle práce

Hlavním cílem práce je navrhnout implementaci automatizace testování softwaru v korporátním procesu s ohledem na specifické požadavky a kriteria jako např. cena, výkon, bezpečnost. Dílčí cíle práce jsou:
- analýza testovacích platforem a nástrojů na základě studia dostupných odborných informačních zdrojů
- porovnání vhodných řešení dle zvolených kritérií
- výběr nejvhodnější varianty a navržení její konkrétní implementace v praxi

Metodika

Metodika teoretické části práce je založena na studiu odborných informačních zdrojů v oblasti testování a kvality softwaru a automatizace. V praktické části práce bude provedeno porovnání zvolených platforem a nástrojů pro automatizaci pomocí vícekriteriální analýzy variant. Bude vybráno nejvhodnější řešení s ohledem na specifické požadavky konkrétního prostředí a navržen způsob implementace do praxe. Na základě výsledků teoretické a praktické části budou formulovány závěry práce.

Doporučený rozsah práce

40-50

Klíčová slova

Automatizace testování, agilní vývoj, vodopádový model, průběžná integrace

---

Doporučené zdroje informací

Bureš, M., Renda M., Doležel M. a kol. Efektivní testování softwaru – Klíčové otázky pro efektivitu testovacího procesu. Grada, 2016. ISBN: 978-8024755946

Crispin, L. Gregory J. Agile Testing: A Practical Guide For Testers And Agile Teams. Addison-Wesley Professional, 2008. ISBN: 978-0321534460

Graham, D., Fewster, M. Experiences of Test Automation – Case Studies of Software Test Automation. Addison-Wesley Professional, 2012. ISBN: 978-0321754066

Graham, D., Fewster, M. Software Test Automation – Effective use of test execution tools. Addison-Wesley Professional, 2000. ISBN: 978-0201331400

Myers, G. J., Sandler C., Badgett T. The Art of Software Testing. 3rd Edition. Wiley, 2011. ISBN: 978-1118031964

Roudenský, P. Kvalita software – Teorie a praxe. Computer Media, 2018. ISBN: 978-8074023224

Předběžný termín obhajoby

2021/22 LS – PEF

Vedoucí práce

Ing. Jan Pavlík

Garantující pracoviště

Katedra informačních technologií

Elektronicky schváleno dne 17. 8. 2021

doc. Ing. Jiří Vaněk, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 5. 10. 2021

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 15. 03. 2022

Čestné prohlášení

Prohlašuji, že svou bakalářskou práci "Automatizace procesů v IT" jsem vypracoval(a) samostatně pod vedením vedoucího bakalářské práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené bakalářské práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 15.3.2022 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) Ing. Janu Pavlíkovi za laskavé vedení bakalářské práce.

Automatizace procesů v IT

Abstrakt

Cílem této práce je výběr, návrh a implementace automatizace testování v prostředí korporátní organizace. Obsahem teoretické části je analýza trhu platforem pro automatizaci, jejich porovnání a SWOT analýza. Věnujeme se komerčním i nekomerčním produktům. Vybereme nejvhodnější řešení pro dané nasazení. Kritérii jsou cena, jednoduchost implementace a naplnění všech bezpečnostních a výkonových kritérií organizace. Téma automatizace testování je aktuálním problémem řešeným napříč celým trhem s informačními technologiemi, snažíme se pro to vybrat co nejmodernější technologie. Aktuálním trendům se věnuje i kapitola v teoretické části práce. Zde se zabývám také rolí automatizace v agilním vývoji. V praktické části práce se věnuji implementaci zvoleného řešení a vyhodnocuji jeho přínos korporaci.

Klíčová slova:
Automatizace, testování, development, agilní, waterfall, test, performance, continuous integration

Test automation in IT

Abstract

The aim of this work is the selection, design and implementation of testing automation in the corporate organization. The content of the theoretical part is the market analysis of automation platforms, their comparison and SWOT analysis. We deal with commercial and non-commercial products. We will choose the most suitable solution for the given deployment. The criteria are price, ease of implementation and fulfillment of all security and performance criteria of the organization. The topic of test automation is a current problem solved across the entire information technology market, we try to choose the most modern technologies. The chapter in the theoretical part of the thesis also deals with current trends. Here I also deal with the role of automation in agile development. In the practical part of the work I focus on the implementation of the chosen solution and evaluate its contribution to the corporation.

Keywords:

Automation, testing, development, agile, waterfall, test, performance, continuous integration

Obsah

Seznam obrázků

Seznam tabulek

Slovník použitých pojmů a zkratek

Zkratka	Pojem	Význam
UML	Unified Modeling Language	Grafický jazyk pro vizualizaci, specifikaci, navrhování a dokumentaci programových systém
RUP	Rational Unified Software	Propracovaná metodika vývoje softwaru
XP	Extrémní programování	Metodologie vývoje software
SIT	System Integration Testing	Systémové a integrační testování – testování z pohledu funkčnosti a integrity systému
UAT	User Acceptance Testing	Uživatelské a akceptační testování – testování z pohledu zákazníka
GUI	Graphic User Interface	Grafické uživatelské rozhraní
API	Application Programming Interface	Rozhraní pro programování aplikací
IDE	Integrated Development Environment	Software usnadňující práci programátorů, většinou zaměřené na jeden konkrétní programovací jazyk. Obsahuje editor zdrojového kódu, kompilátor, případně interpret a většinou také debugger.
SOAP	Simple Object Access Protocol	Protokol pro výměnu strukturovaných informací po síti (je často využíván u webových služeb).
GIT	Global Information Tracker	Název systému správy verzí při vývoji software

1 Úvod

Automatizace testování je v současnosti velmi skloňované téma. Se vzrůstající závislostí procesů na informačních systémech a postupující digitalizací se stáváme čím dál náchylnější a zranitelnější vůči chybám v používaných systémech. S přibývajícími objemy dat a zpracováváním složitějších problémů narážíme na limity možností testovacích týmů (a nejen jich) manuálně otestovat všechny potřebné scénáře včas a dostatečně kvalitně.

Tuto oblast také dále ovlivňuje postupný přechod na agilní metodiky vývoje a odklon od vodopádového řešení. V turbulentní době a při zvyšujícím se tempu změn již vodopádové modely nejsou konkurenceschopné. Použití iteračních modelů přináší velké množství repetitivní práce, kterou je potřeba vykonat před každým nasazením produktu na produkci, ale i jindy. Tato práce je velmi vhodným adeptem pro automatizaci, která zajistí uvolnění cenných zdrojů pro testování nových funkcionalit. Pokud se vybere vhodný soubor testů, dokáže zajistit dobrou stabilitu aplikace a testovat i zpětnou kompatibilitu.

Vývoj je překotný a s ním i vývoj nástrojů a aplikací pro automatizaci testování. Na výběr je celá škála produktů jak open-source, tak i poměrně drahých profesionálních řešení. Liší se nejen implantovanými technologiemi, podporou, ale kladou i velmi různé nároky na znalosti členů testovacích týmů, které je budou používat. Někdy stačí jen znalost použití webového prohlížeče, pro použití jiných je nutná znalost programování. Stejně tak nám tyto nástroje dokážou generovat různé výsledky a hodí se pro použití v trochu jiných podmínkách.

Výběr vhodného řešení pro automatizaci specifického produktu se tak stává opravdovou výzvou. Zorientovat se na trhu, stanovit kritéria výběru a navrhnout nasazení daného nástroje do praxe je hlavním cílem této bakalářské práce. Vedlejším cílem je zpracování teoretických východisek jako podkladu pro tento výběr.

2 Cíl práce a metodika

2.1 Cíle práce

Hlavním cílem práce je navrhnout implementaci automatizace testování softwaru v korporátním procesu s ohledem na specifické požadavky a kritéria jako např. cena, výkon, bezpečnost. Dílčí cíle práce jsou:

- analýza testovacích platforem a nástrojů na základě studia dostupných odborných informačních zdrojů
- porovnání vhodných řešení dle zvolených kritérií
- výběr nejvhodnější varianty a navržení její konkrétní implementace v praxi.

2.2 Metodika

Metodika teoretické části práce je založena na studiu odborných informačních zdrojů v oblasti testování a kvality softwaru a automatizace. V praktické části práce bude provedeno porovnání zvolených platforem a nástrojů pro automatizaci pomocí vícekriteriální analýzy variant. Bude vybráno nejvhodnější řešení s ohledem na specifické požadavky konkrétního prostředí a navržen způsob implementace do praxe. Na základě výsledků teoretické a praktické části budou formulovány závěry práce.

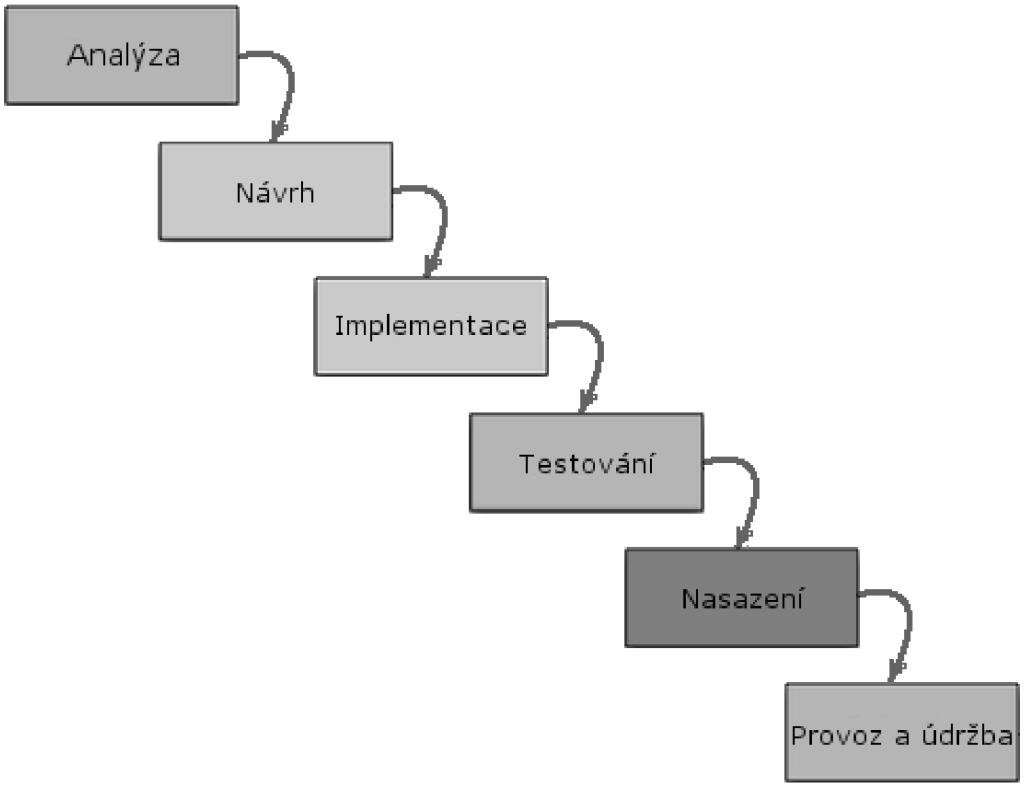
3 Teoretická východiska

3.1 Testování softwaru a jeho hodnota

Návrh a implantace softwarových produktů je poměrně složitý proces. Vstupuje do něj mnoho aktérů a je při něm potřeba zajistit množství úkolů. Nástrojem pro koordinaci těchto činností jsou různé metodiky vývoje softwaru. Mnoho metodik je založených na tzv. modelování, kde se využívá různých nástrojů k tomu určených – např. UML. Modelování slouží k návrhu vztahů mezi dílčími částmi navrhovaného systému, k jejich zobrazení, k návrhu vstupů a výstupů jednotlivých komponent a k celkové analýze vyvíjeného softwaru. Modelování je účinnou metodou, jak zabránit chybám vycházejícím ze slabě analyzovaných částí kódu. Poskytuje také podpůrný prostředek pro tvorbu dokumentace. Usnadňuje práci jasným vymezením úkolů.*9*

3.1.1 Životní cyklus vývoje softwaru

Pojem životní cyklus znamená návaznost jednotlivých etap procesu vývoje [9]. Budeme se podrobněji věnovat pěti etapám, znázorněným na Obrázek 1- Životní cyklus vývoje software



Obrázek 1- Životní cyklus vývoje software
Zdroj: https://www.soom.cz/data/PTWA_SDLCvodopad.png

3.1.2 Analýza a specifikace

V této etapě se provádí sběr požadavků na nový produkt a jejich dokumentace. Je vyhotovena analýza koncepce a návrh produktu. Připravujeme analýzu rizik a akceptační testy. Rozhodnutí vzniklá během této etapy mají největší dopad na výsledný produkt [13].

3.1.3 Návrh systému

Provádíme technické práce na návrhu – činnosti týkající se zajištění bezporuchovosti, udržovatelnosti a ochrany proti vlivům prostředí. Naplánujeme rozdělení na podproblémy, specifikujeme jejich funkcionality a rozhraní mezi nimi. Plánují se podrobnější testy systému. Je vhodné naplánovat postup nasazení. Zamýšlíme se nad jednotlivými moduly, jejich algoritmy a datovými strukturami. Také nad způsobem jejich testování. Výstupem by měl být odhad ceny a nároků na lidské zdroje a čas [13].

3.1.4 Implementace a testování součástí

V tomto období je produkt na základě připravené dokumentace zhotoven a jednotlivé moduly otestovány. Jsou provedeny konfirmační testy. Je zhotovena uživatelská dokumentace *13*.

3.1.5 Integrace a nasazení systému

Jednotlivé moduly jsou začleněny v rámci finálního systému, provádíme integrační testy. Jsou provedeny uživatelské akceptační testy a revidována dokumentace. Produkt je instalován na místo svého provozu. Je zkontrolována a dokumentována možnost údržby systému [13].

3.1.6 Provoz a údržba

Jedná se o nejdelší období životního cyklu produktu. Provádíme školení obslužného personálu. Řeší se provozní problémy a opravují chyby. Dále provádíme modernizaci produktu. Konec této fáze nastává, pokud je provoz dále nehospodárný vlivem zvýšených nákladů na údržbu produktu nebo vlivem jiných faktorů [13].

3.2 Modely životního cyklu softwaru

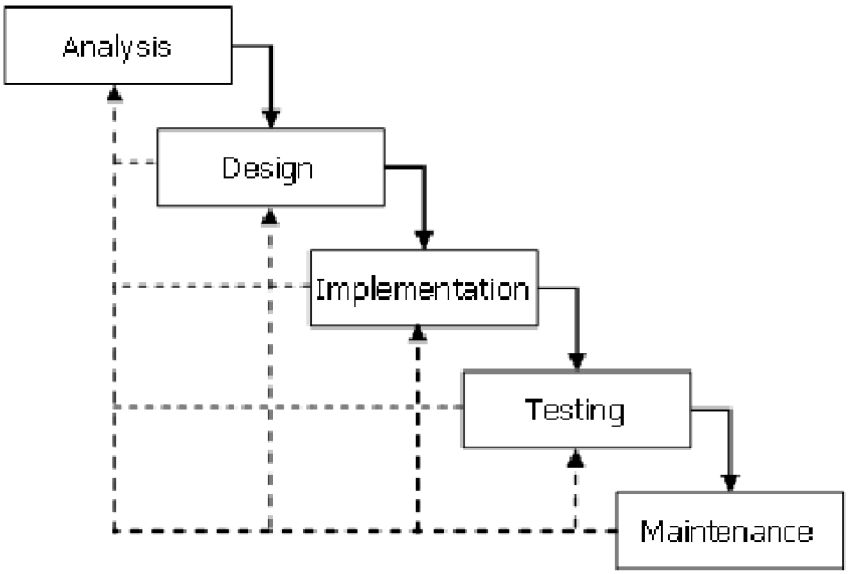
Model životního cyklu softwaru popisuje vzájemné vztahy mezi fázemi jeho životního cyklu. Každý model obsahuje vlastní metodiku, jak zajistit dostatečnou kvalitu produktu.
Výběr vhodného modelu je klíčový pro úspěch celého projektu. V současné době je těchto modelů velké množství. Představíme si hlavní v současnosti používané modely [13].

3.2.1 Vodopádový model

Jedná se o nejstarší model životního cyklu softwaru. Byl vyvinut s cílem lépe se vyrovnat s rostoucí složitostí produktů leteckému průmyslu. Model vychází ze sekvenčního přístupu k jednotlivým fázím. Je charakterizován tím, že vstoupit do další fáze mohu až v momentu, kdy je předchozí fáze kompletně dokončena a uzavřena. Model se dá v praxi použít jen tehdy, pokud je počátečním fázím věnován dostatek času. „Odhalení a odstranění chyby, která vznikne na počátku životního cyklu, je mnohem levnější, než kdybychom tuto chybu opravovali později“[2]. Pokud tento model chceme použít, musíme si být na konci každé fáze maximálně jistí její validací a kompletností. Vzhledem k tomu, že požadavky klientů se mohou v průběhu realizace projektu měnit, nelze prakticky dokončit jednu fázi a zahájit další, beztoho aniž bychom se k ní v budoucnu opět vrátili.

Finální verze aplikace je klientovi představena až v momentu, kdy se nelze vrátit do procesu oprav a úprav. To může být důvodem neúspěchu celého projektu. Fáze testování nastává až po samotné implementaci, ve chvíli, kdy je produkt téměř připraven na předání zákazníkovi. Všechny změny je třeba zapracovat do specifikace a veškeré dokumentace. „Chyba v analýze může vyústit v kompletní přepracování celého projektu“[8]. Tento model je jednoduchý a snadno pochopitelný. Pro ilustraci je přiložen Obrázek 2 - Vodopádový model Zpravidla se používá u menších projektů, kde se nepočítá s pozdějšími změnami funkcionalit či vlastností celého výsledného programu.

Z původního modelu vychází spousta modifikací, které se snaží odstranit nedostatky původního modelu, kvůli kterým se v praxi příliš nepoužívá. Příkladem může být V-model, Sashimi a další.



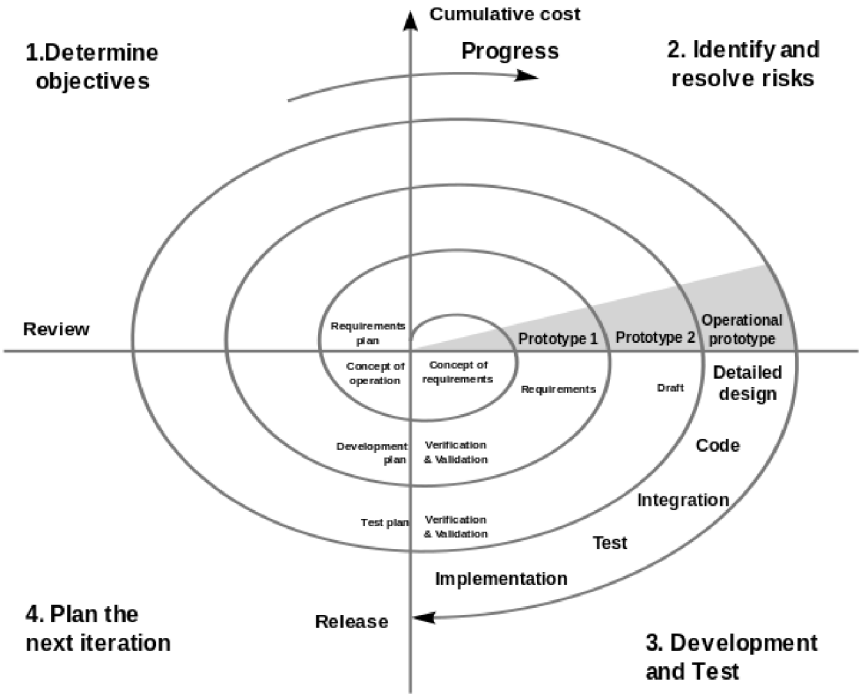
Obrázek 2 - Vodopádový model
Zdroj: https://arxiv.org/ftp/arxiv/papers/1205/1205.6904.pdf

3.2.2 Spirálový model

Tento model se snaží vyřešit nedostatky vodopádového modelu. Náleží do skupiny přístupů řízených riziky. Postup do další fáze závisí na analýze všech rizik a možných problémů. Rizika zde chápeme v nejobecnějším smyslu, např. vztah k legislativě či marketingu. Model je založen na iterativním přístupu. Lépe se tak vyrovnává s pozdější úpravou požadavků. Je proto vhodný i pro větší projekty. Model probíhá v několika krocích, které se neustále opakují, do doby dokončení produktu. Nové části se nasazují na již prověřený základ.
Z počátku vyvíjíme na základě hrubé specifikace, která se každou iterací upřesňuje[3].
Během každého cyklu spirály jsou spouštěny čtyři základní fáze (kvadranty) – viz. Obrázek 3 - Spirálový model

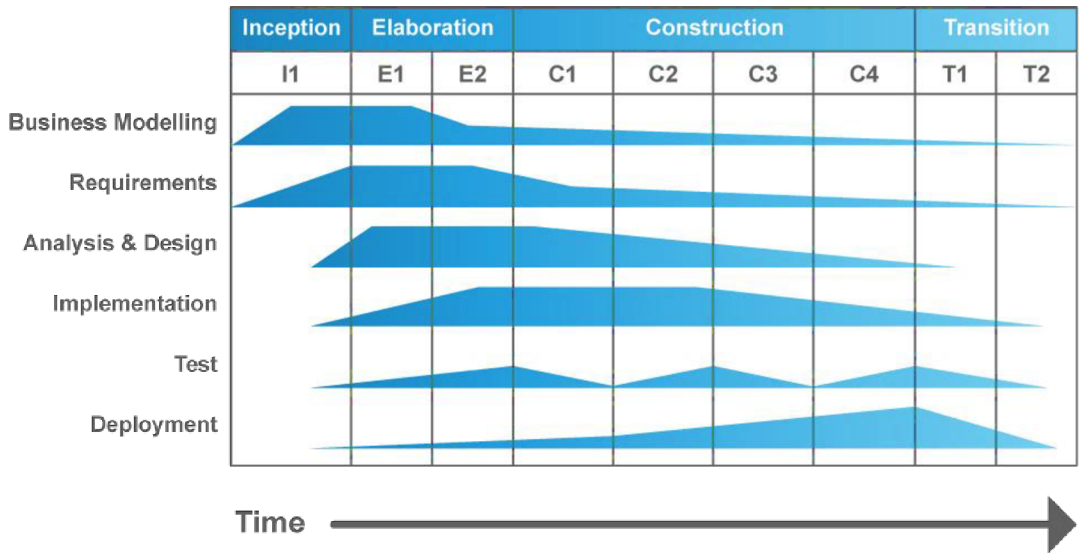
- Analýza – stanovení cílů, alternativ a rozsahu iterace
- Vyhodnocení – vyhodnocení alternativ, identifikace a řešení rizik
- Vývoj – vývoj produktu a kontrola očekávaných výsledků
- Plánování – plán pro příští iteraci

Po každé fázi následuje testování, hodnocení a předání dílčích výsledků. Produkt je pravidelně testován již od raných fází. Díky tomuto přístupu je vhodné využít automatizovaných testů. Ty při každé iteraci pouze upravujeme podle aktuálních testovacích případů a scénáře. Aplikaci je možné testovat i po částech. Díky tomu dochází ke včasnému (a levnému) odhalování chyb.



Obrázek 3 - Spirálový model
Zdroj: https://commons.wikimedia.org/wiki/File:Spirale_(Boehm,_1988).svg

Jedná se o objektově orientovaný iterativní přístup k vývoji software. Náleží do skupiny tzv. přístupů řízených případy použití (use-casedriven approach). Pro modelování procesů se využívá prostředků jazyka UML. Na testování je kladen důraz během celého modelu. Nejvíce však těsně před předáním produktu zákazníkovi, tedy ke konci implementace.
Obsahuje celkem čtyři základní fáze. Každá fáze obsahuje několik dalších iterací – viz. Obrázek 4 - RUP Před započetím nové iterace musí být splněna dříve definovaná kritéria předchozí iterace. Fáze zahájení (inception) definuje účel, rozsah projektu a jeho obchodní kontext. Ve fázi projektování (elaboration) je potřeba analyzovat požadavky zákazníka, celého projektu a definovat základy architektury. Realizační fáze (construction) je nejdelší probíhá zde tvorba zdrojových kódů. V poslední fázi předání (transition) může být projekt předán zákazníkovi nebo do dalšího cyklu. Testování prochází mnoha fázemi, proto lze testy neustále rozvíjet[4].



Obrázek 4 - RUP
Zdroj: https://www.eps.ch/assets/components/phpthumbof/cache/eps-website-rup-1000.11255b4d9ea2c21d0eb60c6b83ffed0b.png

3.2.3 Agilní vývoj

Jedná se o skupinu metodik vývoje softwaru založených na iterativním vývoji, kde se požadavky a řešení postupně vyvíjejí těsnou spoluprací mezi samo organizujícími se multifunkčními týmy. Termín se objevil v roce 2001, kdy byl formulován Agilní manifest[11].

Agilní metodiky jsou založeny na iterativním vývoji, ale usilují o odlehčenější a více na lidi zaměřený přístup než tradiční metodiky. Agilní procesy v základu zahrnují iteraci a neustálou zpětnou vazbu, které vedou k postupnému zjemňování a zpřesňování produktu.
Agilní přístup není omezen pouze na programování, ale našel své uplatnění také ve finančnictví, telekomunikacích, marketingu i v oblasti řízení lidských zdrojů.
Priority agilního programování:

- Lidé a jejich spolupráce před procesy a nástroji
- Fungující software před obsáhlou dokumentací
- Spolupráce se zákazníkem před sjednáváním smluv
- Reakce na změnu před dodržováním plánu

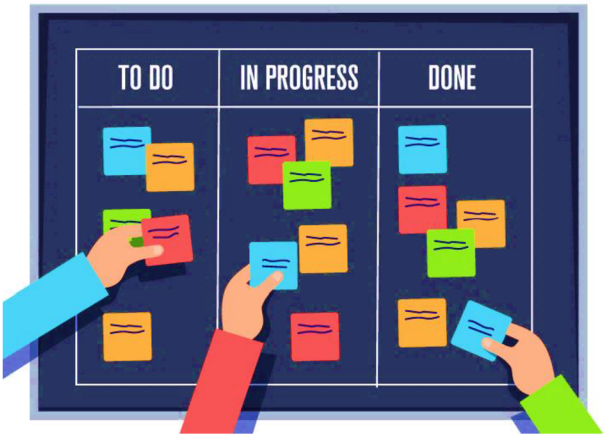
Většina vývoje dnes probíhá agilní metodikou[5]. Agilní přístup je bezpečnější pro dodavatele i zákazníka. Klient již od počátku nemusí mít zcela jasno v tom, jak bude výsledný produkt vypadat a může dělat v jeho průběhu změny, a dodavatel produktu nemusí být již od počátku schopen odhadnout cenu celé zakázky.
Dále není problém díky rozdělení vývoje na sprinty průběžně nějaké funkce upravit, vynechat nebo přidat a jednoduše se na to v dalším sprintu soustředit. Vyhneme se tak drahým změnám na konci projektu. Nevýhoda tohoto způsobu je zásadní nutnost zapojit klienta do procesu vývoje.
Existuje mnoho typů této metodiky.

3.2.4 XP (Extreme programming)

Jedná se o časté dodávky software v krátkých vývojových cyklech. Využívá párové programování, unit testy, propaguje použití jednoduchého a jasného kódu. Používá se zde společné vlastnictví kódu a neustálý refaktoring. Klade důraz na komunikaci v rámci týmu i s klientem.

3.2.5 Kanban

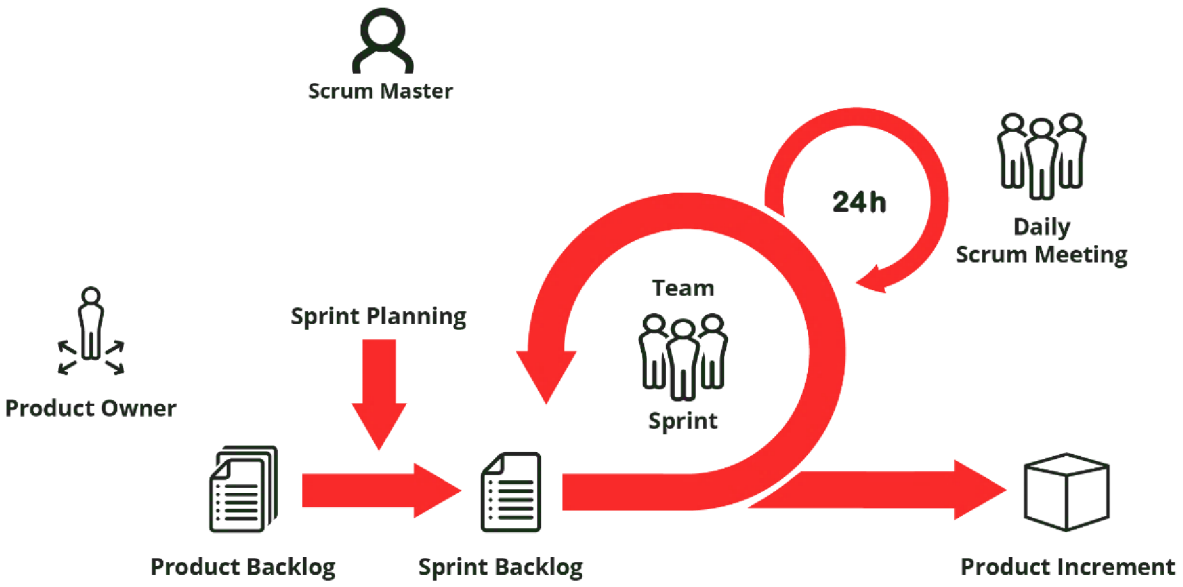
Koncept spjatý se s principem výroby „Just in time Kanban je nejčastěji znázorňován jako tabule, která má několik sloupců. Každý sloupec představuje jednu etapu životnosti úkolu. Úkoly se pak přesouvají od levého sloupce k pravému. Mezi typické životní cykly úkolů patří Zásobník, Schváleno, Pracuje se, Dokončeno (volné překlady z anglických termínů Backlog, To Do, In Progress, Done)[14]. Tabule je znázorněna na Obrázek 5 - Kanban
Tradičně se používají při agilním programování, ale je možné ho použít v jakémkoliv oboru, případně i denním životě.



Obrázek 5 - Kanban
Zdroj: https://www.integrify.com/site/assets/files/2473/kanban-project-management.png

3.2.6 Scrum

Klíčovou částí metodiky jsou každodenní krátká setkání týmu, nazývaná daily standups. Každý člen zde referuje o své činnosti z minulého dne, o tom, co bude dělat dnes, a na jaké problémy narazil. Metodika prosazuje iterativní vývoj. Období iterace se nazývá Sprint a trvá 1-4 týdny. Výsledkem Sprintu je demo vzniklých úprav, které je předvedeno stakeholderům (zákazníkovi, sponzorovi, management boardu, investorům)[7]. Ti poskytují zpětnou vazbu, což umožňuje rychle reagovat na změny v požadavcích. Jsou zde rozeznávány tři role – Product Owner má za úkol komunikovat se zákazníkem a definici co nejlepšího produktu, bývá nazýván hlasem zákazníka. Správné fungování vývojového týmu zajišťuje Scrum Master. Člen vývojového týmu se nazývá Scrum Team Member.



Obrázek 6 - Scrum
Zdroj: https://www.etventure.de/wp-content/uploads/2018/01/Digitallearning-Scrum.png

3.3 Fáze a úrovně provádění testů

3.3.1 Black box vs White box

Testování může být rozděleno do dvou základních kategorií: black box a white box. Zatímco black box testování je metodou, při které nepotřebujeme znát vnitřní strukturu projektu, při white box testingu známe strukturu a design implementovaného softwaru.

Black box	**White box**
Nemusíme znát testovaný produkt s výjimkou specifikace	Spoléhá na analýzu zdrojového kódu
Používá se v integračním a systémovém testování	Používá se při unit testování
Typicky prováděno uživateli	Typicky prováděno programátory

Tabulka 1 - Blackbox vs whitebox
Zdroj: vlastní zpracování na základě https://cs.education-wiki.com/4775748-black-box-testing

3.3.2 Testování programátorem (Developer testing)

V praxi jsou tyto testy označovány jako „Assembly tests“. Většinou si však programátor netestuje svoji část kódu, ale realizuje se tzv. „test čtyř očí“[12]. To znamená, že kód testuje jiný programátor než ten, který jej napsal. Program je v tomto stupni kontrolován na úrovni zdrojového kódu. V praxi je bohužel tento stupeň testování často podceňován. Přitom opravy chyby v této části testování software je nejméně nákladná.

3.3.3 Testování jednotek (Unit testing)

U objektově orientovaného programování se jedná o testování jednotlivých tříd a metod. Testovanou jednotkou v tomto případě rozumějme samostatně testovatelnou část aplikačního programu. Testy těchto jednotek se zapisují ve formě programového kódu. Proto jej z pravidla obsluhují vývojáři[12]. Pro vytváření testů se využívá nástrojů na bázi frameworků. Testy jednotek se velmi špatně aplikují na již zaběhlých projektech.

3.3.4 Funkční testy (Functional testing)

Testují se všechny funkce v aplikaci, ověřuje se, že fungují správně, a že odpovídají požadavkům zákazníka. Na výslednou spolehlivost aplikace mají velký vliv. Aplikace nemusí být testována na plně integrovaném prostředí. Splnění akceptačních kritérií na konci FAT je nutnou podmínkou pro vstup do následující fáze. [12]

3.3.5 Integrační testování (Integrativní testing)

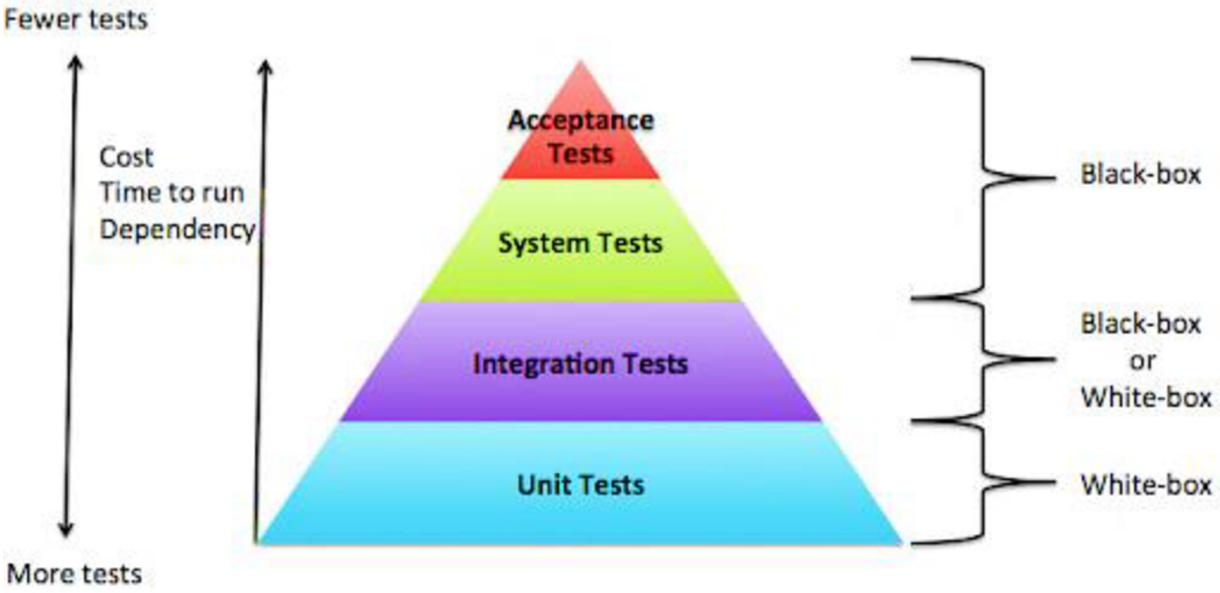
Ověřuje bezchybnou komunikaci mezi jednotlivými komponentami uvnitř aplikace. Integraci však lze ověřovat nejen mezi komponentami, ale také mezi komponentou a operačním systémem, hardwarem či rozhraním různých systémů. V této fázi se tak testuje integrace dosud jednotlivě ověřených částí. [12]

3.3.6 SIT – Systémové testování (Systém testing)

Během těchto testů je aplikace ověřována jako funkční celek. Tyto testy jsou používány v pozdějších fázích vývoje. Ověřují aplikaci z pohledu zákazníka. Podle připravených scénářů se simulují různé kroky, které v praxi mohou nastat. Součástí této úrovně jsou jak funkční, tak nefunkční testy. Poslední úroveň testů, které se provádějí před předáním produktu zákazníkovi, jsou tedy systémové testy. [12]

3.3.7 UAT – Akceptační testování (Acceptance testing)

Jedná se o akceptační testy na straně zákazníka. Pokud všechny předchozí etapy testů proběhly bez větších nedostatků, je možné předat aplikaci zákazníkovi. Testy probíhají na testovacím prostředí u zákazníka. Nalezené nesrovnalosti mezi aplikací a specifikací, jsou reportovány zpět vývojovému týmu. Opravené chyby jsou nasazeny zpět na prostředí u zákazníka. Jsou podmínkou úspěšného ukončení projektu. [12]

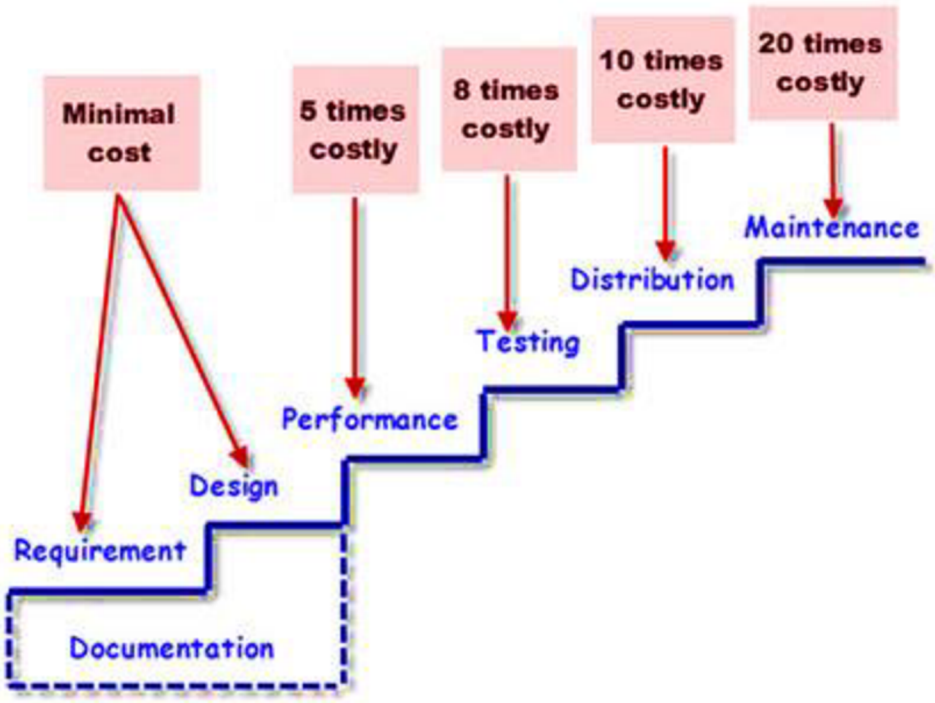


Obrázek 7- Testovací fáze
Zdroj: http://softwaretesting.natashaleite.com/wp/wp-content/uploads/2014/02/Screen-Shot-2014-02-09-at-10.43.34.png

3.4 Hodnota testování

Nekvalita testovacího procesu může mít pro firmu velice závažný dopad. Hodnotíme jednak přímé finanční dopady – výpadek objednávek (nefunkční e-shop), letadla sedící na zemi a ve skladech (Boeing 737 MAX), ale stejně závažné jsou problémy reputační, které mohou být o řád až dva větší. Dále musíme započítat náklady na analýzu a opravu chyby, regresní, zátěžové a uživatelské testy. Těmto nákladům, které mohou jít až do několika milionů USD (v závislosti na vyvíjeném produktu) se ale pravděpodobně mohlo předejít vhodně navrženým a implantovaným automatizovaným testem. Náklady na takovýto test jsou ve srovnání s výše uvedenými náklady a (hlavně) ztrátou reputace u zákazníků zanedbatelné. Přesto se v praxi s takovýmto druhem testování pravidelně nesetkáváme. Náklady na defekt, který je nalezen až v produkci, jsou podle průzkumu v průměru až12,5krát vyšší než náklady na odstranění defektu ve vývojářské fázi. Pokud jsou testovány i analytické dokumenty, může být oprava chyb levnější až 50krát[1].

Cena za nalezení chyby je podrobně ilustrována na Obrázek 8- Cena za nalezení chyby V počátečních stadiích vývoje je cena nejnižší, s postupem času významně roste a nejdražší je ve fázi údržby. Nejvyšší úspory dosáhneme ve fázi analýzy a designu, proto by jí měla být věnována patřičná pozornost.



Obrázek 8- Cena za nalezení chyby

Zdroj: https://qatestlab.com/assets/Start-Software-Testing-On-Early-Stages4.-How-Does-This-Affect-Cost.jpg

3.5 Cena za kvalitu

Pro stanovení ceny testů je výhodné se ptát: „Čeho chceme pomocí testů dosáhnout? Jaká je hodnota, kterou testy přinesou?“[1] Odpověď na tuto otázku nám mohou dát jen objednavatelé projektu. V praxi se osvědčila diskuse ohledně očekáváné hodnoty testování, která definuje i cíle testování. Dále odhadneme pracnost a validujeme ji se zákazníkem.

Dle ISTQB je tzv. „Cost of quality“ definována následujícími kategoriemi nákladů:

- náklady na prevenci defektů (např. školení, zavedení kódovacích standardů)
- náklady na nalezení defektu (všechny testovací aktivity počínaje plánováním)
- náklady vyvolané řešením defektů během testování (znovu nasazení, retesty)
- náklady vyvolané selháními v produkčním prostředí (přímé náklady na opravu, ale i sankce a náklady na ztrátu reputace)

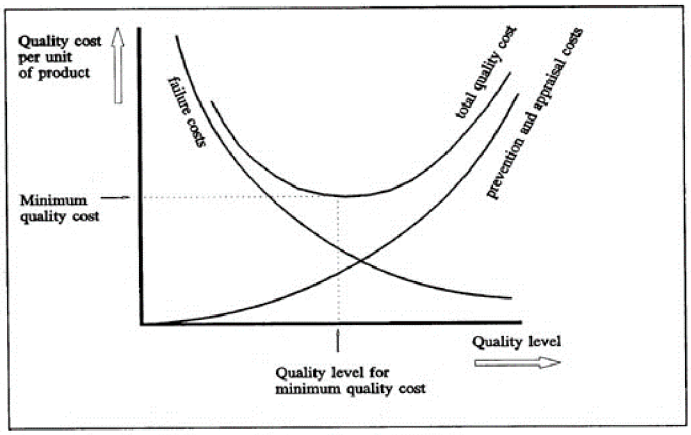
Náklady na zajištění kvality před nasazením do produkčního prostředí jsou dané součtem prvních tří položek:

Cena zajištění kvality = Náklady na prevenci + Náklady na nalezení defektů + Náklady na opravu

Přínos těchto investic před uvedením systému do produkčního prostředí vůči možným nákladům, které by bylo nutné vynaložit, kdybychom tyto aktivity nerealizovali pak ukazuje cena za kvalitu:

Cena za kvalitu = Náklady vyvolané možným produkčním selháním – Cena zajištění kvality

Testovat má smysl, dokud jsou náklady na testování nižší než náklady na řešení případných produkčních incidentů[1].



Obrázek 9 - Cena za kvalitu

Zdroj: https://online-accounting.net/wp-content/uploads/2020/10/image-MlbvaANSDI15T7gR.png

3.6 Automatizace testování

Velkým tématem současnosti je automatizace testovaní. Ta může uspořit mnoho zdrojů. Zároveň ale může mnoho nákladů přinášet, pokud se nepoužije správně. Za automatizované testování softwaru se považuje, když je část testovacího procesu, nebo i celý tento proces, prováděn bez přímého působení člověka, a to pomocí specializovaného softwaru[15]. Existuje mnoho nástrojů, softwaru a způsobů využití automatizace. Otázkou zůstává, kdy se vyplatí a co je vhodné automatizovat. Testy má smysl automatizovat, pokud se dostatečné množství testů neměnně, nebo jen s malými změnami, opakuje dostatečně často. Testy je vhodné používat nad neměnnou částí aplikace. Je vhodné automatizovat části, kde již proběhly manuální testy. Automatizace je výhodná u regresních testů, které mají ověřit stálou funkčnost částí aplikace.

Velký význam má automatizace u nefunkčních (např. performance) testů. Velké množství těch testů není bez užití nástrojů ani možné provést. Vhodné je automatické testy použít u smoke testů, které mají ověřit správnost instalace a konfigurace aplikace. Používá se také u testů založených na datech, většina nástrojů je na to stavěna. Automatizace testů je zásadní pro průběžné doručování (continuous delivery) a průběžné testování (continuous testing). Rostoucím trendem ve vývoji softwaru je používání testovacích frameworků, např. xUnit (například JUnit a NUnit), které umožňují provádění jednotkových testů. Časté opakování automatických testů může být důvodem pro tzv. „pesticide paradox“, kdy přestanou detekovat chyby jdoucí nad rámec jejich rozhraní. [16]

Obrázek 10 - Manuální versus automatizované testování
Zdroj: https://bitbar.com/wp-content/uploads/old_testdroid/2015/07/Manual-vs.-Automation.jpg

3.6.1 Výhody a nevýhody automatizovaného testování

Oba přístupy k testování (manuální x automatické) mají svá pro a proti a hodí se na jiné typy testovacích případů. Nejpodstatnější výhody automatizovaného testování jsou spolehlivost, rychlost a opakovatelnost[6]. Průběh testu je vždy spolehlivý a nikdy nevynechá žádné detaily. Je vždy objektivní – dopadne buď úspěšně nebo neúspěšně. Testy lze pravidelně opakovat, například každou noc anebo po každé změně repozitáře. Výhodou je možnost spouštět stejné testy s různými testovacími daty. Po implementaci testu každé jeho další spuštění stojí jen strojový čas.

Nevýhodou automatizovaného testování je nutnost neustálé údržby. Čím častěji se aplikace mění, tím větší jsou náklady na údržbu. Může to vést k tomu, že se automatizace stane méně efektivní než manuální testování. Automatizací nedokážeme najít chyby rozvržení a použitelnosti. Nemůžeme využít implicitní znalosti a nenajdeme chyby, které nesouvisí s funkcionalitou, kterou ověřujeme. Tyto testy také nedokážou najít mnoho nových chyb. Mohou nám dávat falešný pocit bezpečí.

3.6.2 Vhodnost aplikace pro automatizaci

Vhodnost aplikace pro nasazení autotestů posoudíme dle několika kritérií. Především by daná aplikace měla být stabilní. To znamená, aby se v rámci vývoje pracovalo se stejnými názvy a strukturou objektů a metod. Dále je vhodné, aby byla daná aplikace objektová. Čím jednodušší je aplikace, tím snadnější je potom automatizace. Automatizační nástroje se mohou prodražit, a tak se obvykle používají v kombinaci s manuálním testováním. Je třeba ale myslet na to, že je nutné je manuálně udržovat.

3.6.3 Vhodný nástroj pro automatizaci

Volba vhodného nástroje je úspěch automatizace klíčová. Často se dává přednost ekonomickým aspektům před funkčními. Volba by se měla odvíjet od technických požadavků. Zvolený nástroj by měl být schopen pracovat se všemi objekty aplikace.

3.6.4 Testování na základě GUI

Výhodou tohoto přístupu je, že nevyžaduje znalost kódování ani vývoje software. Existují nástroje poskytující funkce záznamu a přehrávání uživatelských akcí (např. kliku myší). Aplikace musí poskytovat grafické uživatelské rozhraní. Jakákoli změna v něm, např. umístění tlačítek vyžaduje opětovné zaznamenání testu. Tomu se lze vyhnout použitím nástroje, který pracuje s DOM událostmi.

Tato varianta je vhodná např. pro testování webových stránek. Můžeme využívat prohlížeče bez grafického uživatelského rozhraní, nebo nástroje založené na Selenim Web Driver.

3.6.5 Testování na základě API

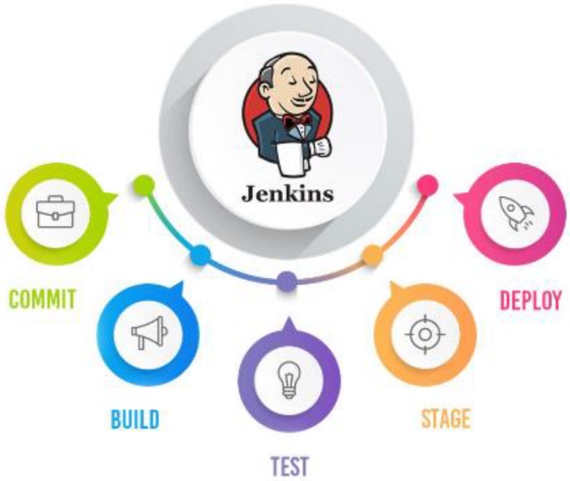
Toto řešení umožňuje testovat funkce nezávisle na jejich implementaci v GUI. Tyto testy se obvykle používají od raných fází vývoje a umožňují udržovat čistotu kódu. Také je využíváme při integračních testech a při testech týkajících se funkčnosti, spolehlivosti, výkonu nebo zabezpečení. Bez těchto testů se neobejdeme u continuous integration[6].

3.7 Nástroje pro automatizaci testování

Volbě vhodného nástroje pro automatizaci musíme věnovat dostatečnou pozornost. Nástroje můžeme rozdělit podle způsobu fungování. Existují nástroje open-source, které lze použít zdarma, stejně tak jako drahé profesionální komerční nástroje. Analýzou dostupných nástrojů na trhu si představíme hlavní možnosti – jejich vlastnosti, rozsah podpory, cenovou politiku a vhodné případy užití. Kritérium výběru do přehledu byla také dostupnost licence pro uvažované řešení.

3.7.1 Jenkins

Jenkins je free a open-source server napsaný v Javě. Slouží k automatizaci procesu vývoje softwaru v podobě automatizace průběžné integrace. Umožňuje průběžné sestavování a testování projektů. Umí spouštět a monitorovat naplánované úkoly. Tak se perfektně hodí pro použití jako jádro automatizovaného řešení. Jenkins disponuje množstvím rozšíření, a tak může být nastaven pro velké množství použití a integraci s mnoha nástroji a technologiemi. Využijeme ho v kombinaci se Seleniem.[17]



Obrázek 11 - Jenkins
*Zdroj: https://miro.medium.com/max/1107/1*BPDI1zBzsM5i43DAE8_1Jg.png*

3.7.2 Selenium IDE

Selenium IDE slouží k tvorbě testovacích nástrojů v GUI. Používá se jako doplněk/plugin k prohlížeči. Testy se vytváří „nahráváním“ kroků, které uživatel provede v GUI. Uložené testy lze editovat, stejně jako psát úplně nové. Nevýhodou je omezená možnost editace a použitých funkcionalit. Výhodou je snadná použitelnost a to, že není nutné používat psaní kódu. Vytvořené testy mohou být exportovány a upravovány např. v Javě a Pythonu.[18]

3.7.3 Selenium 2 (Webdriver)

Jde o nástroj umožňující vytváření automatických testů ve velké škále programovacích jazyků (např. Java, C#, Ruby, Python, JavaScript, PHP). WebDriver volá každý prohlížeč sám o sobě, a tak je nutné upravovat testy pro každý prohlížeč zvlášť. Podpora je poskytována pro Internet Explorer, Google Chrome, Operu, Firefox, Safari a mobilní prohlížeč operačního systému Android. Velkou výhodou je možnost spouštět testy vzdáleně na serveru a velká variabilita použití. Pro použití je nutná základní znalost kódování. [18]

Obrázek 12 - Selenium Suite
Zdroj: https://www.simplilearn.com/ice9/free_resources_article_thumb/selenium-suite.JPG

3.7.4 SoapUI

SoapUI od firmy SmartBear je velmi používaný opensource automatizační nástroj. Existuje i v placené PRO verzi, která obsahuje podporu a rozšířené možnosti reportingu.
SoapUI pracuje s technologiemi SOAP/WSDL, REST, HTTP, AMF, JDBC a JMS. Obsahuje přehledné uživatelské prostředí, kde lze nahrávat komunikaci klient-server, testy mohou být také psány v Javascriptu. Testy se dají spouštět pomocí příkazové řádky, je možná integrace s mnoha dalšími nástroji. Specialitou je možnost security testování pomocí simulace SQL INJECTION, XML BOMB a dalších.

3.7.5 JMeter

JMeter je opensource nástroj vyvinutý k testování funkčnosti a měření výkonu webových aplikací. Je to nástroj, který dokáže simulovat velkou zátěž na server nebo skupinu serverů. Podporuje následují technologie: HTTP, HTTPS, SOAP, FTP, DB (JDBC), TCP a další. Můžeme ho ovládat pomocí GUI rozhraní. Scripty se také mohou psát v Javě. Je dostupné velké množství různých pluginů (open source i placených). Výsledky jsou ukládány do textové podoby, ale pomocí GUI pluginů si je můžeme zobrazit graficky.[20]

Obrázek 13 - Apache Jmeter
Zdroj: https://upload.wikimedia.org/wikipedia/commons/2/22/Apache_JMeter.png

3.7.6 IBM Rational Functional Tester

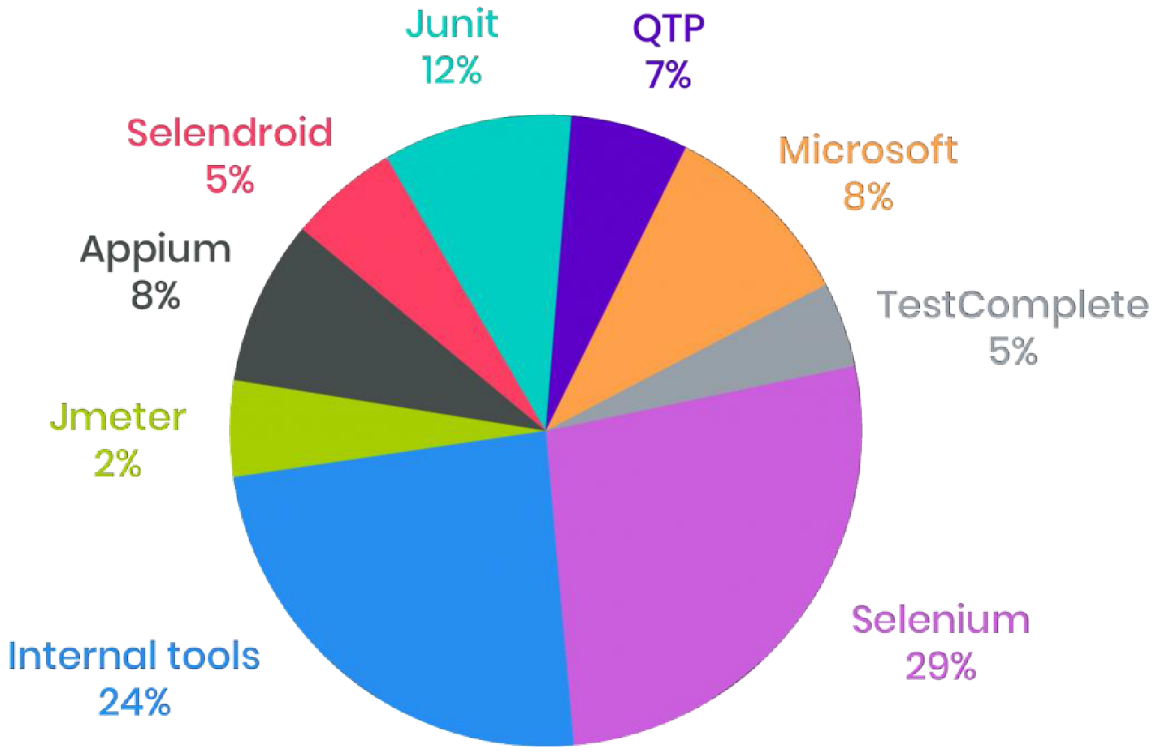
Komerční IBM RFT je nástroj pro automatizované funkční a regresní testování. Podporuje velkou řadu aplikací, např. technologie .Net, Java, Siebel, SAP, emulované terminálové aplikace a PowerBuilder. Je možné i nahrávání klasických uživatelských akcí a jejich pozdější úprava pomocí skriptování v Javě nebo VisualBasicu. Je dobře integrovatelný s jinými IBM Rational produkty jako Rational Team Concert, Rational ClearCase, Rational Quality Manager či Rational TestManager. Jedná se o robustní nástroj, nevýhodou je vysoká cena licence.[21]

3.7.7 HP Unified Functional Testing (Quick Test Professional)

Komerční nástroj firmy HP prošel dlouhým vývojem a změnou názvu. Je dobře integrovaný s oblíbenými nástroji HP Quality Center a HP Quality Management. Je možné GUI testování i http-level testy. Podporuje technologie jako JAVA, .NET, WPF, SAP, Oracle, Flex, Siebel, Delphi, emulátor terminálu, Silverlight, Web Services a mobilní technologie (Windows mobile, Android emulátor). Testy je podobně jako u jiných nástrojů možné nahrávat a později editovat pomocí skriptování ve Visual Basicu. Pro technicky méně zdatné testery je možné testy tvořit pomocí klíčových slov. [22]

3.7.8 Axe Enterprise Test Automation Platform

Komerční nástroj firmy Odin je nástroj pro automatizaci testů a generování dokumentace, cílící na testery a business uživatele. Je možné navrhovat testy v tabulkovém editoru. Je velmi dobře integrovaný s ALM a Visual Studiem. Je možné ho využívat pro testovaní desktopových, webových, mobilních, ERP a SOAP aplikací. Výhodou je možnost tvorby testů netechnickými uživateli. Specialitou je zaměření na accesibility testing.[23]



Obrázek 14 - Podíl na trhu UI Automation toolů v r. 2016
Zdroj: https://www.businessprocessincubator.com/wp-content/uploads/2020/04/www.indiumsoftware.comRanorex-Statics-1024x667-fade12f1cdb10ac66e102dd2a35ea52c2e80bf92.png

3.8 Vícekriteriální analýza variant

Metodou, která pomáhá v rozhodování, kde je nutné respektovat mnoho kritérií pomocí matematického aparátu je vícekriteriální analýza variant. Řešení problému je často nejednoznačné. Výsledkem rozhodování je výběr optimální varianty ze souboru variant potenciálně realizovatelných v dané situaci. Volba tzv. optimální varianty je dosti individuálním počinem, neboť záleží na postoji rozhodovatele a jeho preferencích[24]. Preference jsou vyjádřeny jako soubor kritérií. Správná volba kritérií je důležitým krokem k objektivnímu posouzení všech variant, stejně jako stanovení vah, které vyjadřují důležitost jednotlivých kritérií[24].

3.8.1 Pojmy vícekriteriální analýzy variant

Cíl rozhodování je stav, kterého by mělo být dosaženo. Cílů může být i více a mohou být vyjádřeny kvantitativně i kvalitativně.

Kritéria hodnocení slouží k výstižnému posouzení jednotlivých variant. Existují maximalizační (výnosová) a minimalizační (nákladová) kritéria. Pro maximalizační preferujeme vyšší hodnoty, pro minimalizační preferujeme nižší hodnoty. Mohou být vyjádřena kvalitativně i kvantitativně.

Subjekt rozhodování je ten, který rozhoduje – volí variantu. Může to být jednotlivec anebo skupina lidí.

Objekt rozhodování je oblast pro rozhodování. Může to být například výrobní program, finanční rozvoj firmy atd.

Varianta rozhodování představuje možnost způsobu jednání, která vede ke splnění stanoveného cíle nebo cílů.

Důsledek rozhodování je dopad, který nastal po volbě variant a lze jej vyjádřit příslušnými hodnotami kritérií.

Stav světa chápeme jako situaci, která může nastat pro realizaci varianty.

Vícekriteriální hodnocení variant použijeme, existuje-li množina přípustných variant ve formě konečného seznamu.

Vícekriteriální programování se používá v případech, kdy je množina přípustných variant vymezena souborem podmínek. Jelikož může mít nekonečný počet varant, je pro řešení nezbytné použít vhodný software.

3.8.2 Dostupné informace

Toto členění se odvíjí od typu informace, která je k dispozici rozhodovateli o preferencích mezi kritérii a variantami. Preference mezi kritérii je vyjádřena pomocí vah a preference mezi variantami pomocí hodnot jednotlivých kritérií[24].

Žádná informace o preferencích je přípustná pouze co se týče kritérií. Kdyby totiž neexistovala preference mezi variantami, nebylo by možné vybrat nejlepší variantu[24].

Nominální informace o preferencích se týká opět pouze kritérií. Je vyjádřena pomocí tzv. aspiračních úrovní neboli nejhorších možných hodnot kritérií, při nichž může být varianta akceptována. Aspirační úroveň tak varianty rozděluje na přijatelné a nepřijatelné v rámci určitého kritéria[24].

Ordinální informace vyjadřuje uspořádání kritérií podle důležitosti nebo pořadí variant z hlediska jednotlivých kritérií[24].

Kardinální informace má kvantitativní charakter. V případě kritérií se jedná o číselné vyjádření důležitosti kritéria (váhy) a v případě variant jde o číselné hodnocení variant podle kritérií (např. pomocí bodové stupnice) [24].

3.8.3 Vícekriteriální rozhodování za jistoty

Při rozhodování za jistoty se rozhodovatel může zcela oprostit od vnějšího prostředí, jehož stav je známý a nehrozí tak žádné riziko negativního dopadu na vybrané řešení. Při rozhodování za jistoty lze v podstatě postupovat následujícím způsobem: 1. tvorba variant; 2. stanovení kritérií rozhodování a jejich vah; 3. hodnocení jednotlivých variant; 4. určení výsledné varianty[24].

3.8.4 Kritéria rozhodování a stanovení jejich vah

Výběr vhodných kritérií umožňuje jednoduché a jasné ohodnocení variant podle těchto kritérií. Přirozeně jednodušší je pracovat s kritérii kvantitativního charakteru. Ne vždy je ale možné použít pouze je. Každému kritériu je nutné přidělit váhu, která určuje jeho důležitost. Váhy se stanovují těmito způsoby[24]:

- Nemáme-li **žádné informace** o preferencích mezi kritérii, přidělíme každému kritériu stejnou váhu
- Máme-li **ordinální informaci** o kritériích, tedy můžeme určit pořadí důležitosti kritérií, lze použít metodu pořadí a Fullerovu metodu

- Máme-li **kardinální informace**, tedy známe pořadí i rozestupy preferencí mezi jednotlivými kritérii, použijeme bodovací metodu nebo Saatyho metody

Bodovací metoda patří mezi nejjednodušší metody vyžadující kardinální informaci. Spočívá v přiřazení bodů z určené stupnice každému kritériu. Stupnice může mít v podstatě jakýkoli rozsah, je však běžné používat např. stupnici 1–10. Čím více bodů, tím více je kritérium preferované. Výhodou je to, že je možné přiřadit i několika kritériím stejný počet bodů. Bodovací metodu a metodu pořadí lze použít, pokud kritéria hodnotí více rozhodovatelů[24].

3.8.5 Metody hodnocení variant

Dominovaná varianta je varianta, ke které existuje tzv. dominující varianta, která je podle všech kritérií hodnocena stejně nebo lépe než dominovaná varianta, přičemž alespoň podle jednoho kritéria lépe.

Nedominovaná varianta je varianta, která není dominovaná žádnou jinou variantou, nazývá se též efektivní či paretovská.

Ideální varianta, ať už je hypotetická či opravdu existuje, dosahuje ve všech kritériích nejlepších hodnot.

Bazální varianta naopak dosahuje ve všech kritériích nejhorších hodnot.

Kompromisní varianta, jak již bylo řečeno, vychází z toho, že při použití různých metod výpočtu mohou být určeny různé optimální varianty. Existuje více možností, jak stanovit kompromisní variantu, vždy však musí splňovat podmínku nedominovanosti[24].

Bodovací metoda je velmi jednoduchá metoda, která nevyžaduje znalost vah kritérií. Spočívá v ohodnocení variant podle jednotlivých kritérií prostřednictvím zvolené stupnice, která musí být pro všechna kritéria stejná. Jako kompromisní varianta je pak vybrána ta, která má největší počet bodů. Metodu lze rozšířit i o váhy kritérií, pokud jsou rozhodovateli známy. Hodnoty se pak vypočítají jako vážené součty[24].

Příklad - bodovací metoda s vahami

Č.	Kritérium	Váha	V1	V2	V3	V4	V5
1	Reference	0,30	1,5	2,4	2,1	0,9	1,2
2	Platební podmínky	0,25	1,5	1,75	2	1	1,75
3	Záruky za jakost	0,15	1,2	1,4	0,6	0,15	0,9
4	Lhůta plnění	0,10	0,5	0,6	0,5	0,8	1
5	Podíl vlastních kapacit	0,20	1,6	1	0,6	1,8	2
	Celkem	1	6,3	7,1	5,8	4,7	6,9
	Pořadí		**3.**	**1.**	**4.**	**5.**	**2.**

Obrázek 15 - Příklad bodovací metody s vahami
Zdroj: https://slideplayer.cz/slide/3331095/11/images/19/P%C5%99%C3%ADklad+-+bodovac%C3%AD+metoda+s+vahami.jpg

4 Vlastní práce

4.1 Analýza požadavků

Již bylo uvedeno, že výběr vhodného automatizačního nástroje je klíčovým faktorem úspěchu každého projektu. Je velice důležitý také pro co nejnižší náklady na tvorbu a údržbu testů. S přihlédnutím k bezpečnosti byl výběr zúžen na povolené aplikace, které jsou vypsané v předchozí kapitole. Další aplikace nebudou brány na zřetel. Provedením analýzy na projektu byly získány následující informace.

Automatizací je třeba hlavně vyřešit tyto situace:

- Nedostatek testovacích kapacit v závěrečných fázích projektu
- Zamezení zanášení regresních chyb na produkci
- Umožnění integračního testování v každém momentu vývojového cyklu (stabilní kód)
- Zvýšení spolehlivosti aplikace a tím důvěryhodnosti u klienta
- Umožnit performance testování

Následující druhy testování není třeba prozatím do automatizace zahrnovat:

- Funkční testování
- Integrační testy
- UAT testování
- Nefunkční testy s výjimkou performance

Při výběru aplikace musíme brát na zřetel i následující kritéria:

- Snadnost a spolehlivost použití
- Znalosti členů týmu vzhledem k již používaným technologiím na projektu
- Bezpečností požadavky projektu
- Ekonomické aspekty jako je např. cena licence nebo cena potřebných školení
- Možnosti integrace s dalšími aplikacemi již používanými pro vývoj a řízení projektu

Pro lepší orientaci zde uvádím, které technologie jsou v současnosti již na projektu používány:

- JAVA
- Python
- HTTP
- JUnit
- Bash
- Oracle
- SOAP

Bylo by vhodné, aby vybraný nástroj umožňoval integraci s již nasazenými aplikacemi:

- GIT
- Artifactory
- Jenkins
- IBM Rational Team Concert

4.2 Testovací tým

V projektovém týmu je pět testerů, jejichž znalosti se liší. 3 testeři techničtějšího typu dokážou programovat, znají Javu, Python a tak dále. 2 juniornější testeři testují hlavně manuálně a pro ně je nejdůležitější uživatelská přívětivost a preferují nástroje, ke kterým není třeba znalosti programovat. Dále se procesu testování účastní 2 business analytici a také testuje skupina zhruba 5 superuserů. Výběr automatizačního nástroje se těchto osob tolik netýká, ale byli přítomni při odborných debatách v rámci pravidelných týmových porad. Na jejich názor tak byl také brán zřetel, projevuje se to hlavně preferencí nástrojů s GUI.

4.3 Stanovení kritérií a jejich vah pro výběr

Cílem rozhodovacího procesu je nalezení nejvhodnějšího automatizačního nástroje pro použití na daném projektu. V předchozí kapitole byla provedena analýza požadavků – na jejím základě a na základě odborné diskuse byla vybrána následující kritéria hodnocení:

1. Znalost nástroje a technologií nutných k jeho použití (dále jako Znalostní báze)
2. Obsluhuje využívané technologie (dále jako Kompatibilita)
3. Cena licence
4. Integrační potenciál
5. Technická podpora
6. Možnost performance testování

Byla použita kritéria kvantitativního i kvalitativního charakteru, přičemž některá z nich jsou maximalizační, jiná minimalizační. Podrobná charakteristika každého kritéria následuje níže v textu.

4.3.1 Kritérium 1 – Znalostní báze

Kritérium bylo obodováno v následující tabulce. Znalost technologií jednotlivých členů týmu byla zjištěna v analytické části. V potaz bereme znalost technologie nutné k vytvoření testů. Kritérium je maximalizační. Pro obodování jsme diskutovali a expertně stanovili následující pořadí důležitosti jednotlivých technologií (od nejdůležitější):

1. Java

2. GUI
3. SOAP
4. Python
5. JavaScript
6. Keywords
7. Excel
8. VisualBasic

Počet bodů ke každému nástroji byl přiřazen na základě klíče 7 nejlepší, 1 nejhorší.

Nástroj	GUI	Java	Python	JavaScript	SOAP	VisualBasic	Keywords	Excel	Body
Selenium	ne	ano	ano	ne	ne	ne	ne	ne	**3**
Selenium IDE	ano	ano	ano	ne	ne	ne	ne	ne	**7**
SOAP UI	ano	ne	ne	ano	ano	ne	ne	ne	**6**
JMeter	ano	ano	ne	ne	ne	ne	ne	ne	**4**
IBM RFT	ano	ano	ne	ne	ne	ano	ne	ne	**5**
HP UFT	ano	ne	ne	ne	ne	ne	ano	ne	**1**
AXE	ano	ne	ne	ano	ne	ne	ne	ano	**2**

Tabulka 2 - Ohodnocení kritéria 1 - Znalostní báze
Zdroj: vlastní zpracování

4.3.2 Kritérium 2 – Kompatibilita

Kritérium bylo obodováno v následující tabulce. Využívání technologií v projektu bylo zjištěna v analytické části. Kritérium je maximalizační. Pro obodování jsme diskutovali a expertně stanovili následující pořadí důležitosti jednotlivých technologií (od nejdůležitější):

1. Java
2. HTTP
3. SOAP
4. Python
5. Oracle
6. Junit
7. Bash

Počet bodů ke každému nástroji byl přiřazen na základě klíče 7 nejlepší, 1 nejhorší.

Nástroj	Java	Python	JUnit	Bash	Oracle	HTTP	SOAP	Body
Selenium	ano	ano	ano	ne	ne	ano	ano	**7**
Selenium IDE	ne	ne	ne	ne	ne	ano	ne	**1**
SOAP UI	ne	ne	ne	ne	ano	ano	ano	**4**
JMeter	ne	ne	ne	ne	ano	ano	ano	**4**
IBM RFT	ano	ne	ano	ano	ano	ano	ano	**6**
HP UFT	ano	ne	ano	ano	ano	ano	ne	**2**
AXE	ano	ne	ne	ne	ano	ano	ano	**5**

Tabulka 3 - Ohodnocení kritéria 2 - Kompatibilita
Zdroj: vlastní zpracování

4.3.3 Kritérium 3 – Cena licence

Cena licence je uvedena za jednoho uživatele a platí v lednu 2022. Cena je orientační, může záviset na počtu uživatelů a servisní smlouvě. Kritérium je maximalizační.

Počet bodů ke každému nástroji byl přiřazen na základě klíče 7 nejlepší, 1 nejhorší.

Nástroj	Cena licence (Kč)	Body
Selenium	0	**7**
Selenium IDE	0	**7**
SOAP UI	12000	**3**
JMeter	0	**7**
IBM RFT	200 000	**1**
HP UFT	68 000	**2**
AXE	10 000	**4**

Tabulka 4 - Ohodnocení kritéria 3 - Cena licence
Zdroj: vlastní zpracování

4.3.4 Kritérium 4 – Integrační potenciál

Kritérium bylo obodováno v následující tabulce. Vyžadované integrace byly zjištěny v analytické části. Kritérium je maximalizační. Počet bodů ke každému nástroji byl přiřazen na základě klíče 7 nejlepší, 1 nejhorší. Pro obodování jsme diskutovali a expertně stanovili následující pořadí důležitosti jednotlivých technologií (od nejdůležitější):

1. Jenkins
2. GIT
3. IBM RTC
4. Artifactory

Nástroj	GIT	Artifactory	Jenkins	IBM RTC	Body
Selenium	ano	ano	ano	ano	**7**
Selenium IDE	ne	ne	ano	ne	**3**
SOAP UI	ano	ne	ano	ano	**5**
JMeter	ne	ne	ano	ne	**3**
IBM RFT	ano	ano	ano	ano	**7**
HP UFT	ano	ano	ano	ne	**4**
AXE	ne	ne	ano	ne	**3**

Tabulka 5 - Ohodnocení kritéria 4 - Integrační potenciál
Zdroj: vlastní zpracování

4.3.5 Kritérium 5 – Technická podpora

Toto kritérium bylo hodnoceno na základě klíče 7 nejlepší, 1 nejhorší. Kritérium je maximalizační.

Nástroj	Technická podpora	Body
Selenium	ne	**1**
Selenium IDE	ne	**1**
SOAP UI	ano	**7**
JMeter	ne	**1**
IBM RFT	ano	**7**
HP UFT	ano	**7**
AXE	ano	**7**

Tabulka 6 - Ohodnocení kritéria 5 - Technická podpora
Zdroj: vlastní zpracování

4.3.6 Kritérium 6 – Performance testování

Toto kritérium bylo hodnoceno na základě klíče 7 nejlepší, 1 nejhorší. Kritérium je maximalizační.

Nástroj	Performance testy	Body
Selenium	ne	**1**
Selenium IDE	ne	**1**
SOAP UI	ano	**7**
JMeter	ano	**7**
IBM RFT	ne	**1**
HP UFT	ne	**1**
AXE	ano	**7**

Tabulka 7 - Ohodnocení kritéria 6 – Performance testování
Zdroj: vlastní zpracování

4.4 Vyhodnocení variant

Vyhodnocení vícekriteriální analýzy bylo provedeno pomocí expertního stanovení dílčích ohodnocení. Váhy odpovídají pořadí stanovenému v analýze a byly stanoveny expertním odhadem na základě odborné diskuse. Každé jednotlivé kritérium bylo ohodnoceno body v závislosti na pořadí v rámci kritéria. V níže uvedené tabulce bylo provedeno finální porovnání všech variant. Pro porovnání kritérií využíváme bodovou stupnici 1-7, přičemž 7 je nejlepší. Všechna kritéria již byla převedena na maximalizační.

Kritérium	Váha	Selenium		Selenium IDE		SOAP UI		Jmeter		IBM RFT		HPUFT		AXE	
K1	0.3	0.9	3	2.1	7	1.8	6	1.2	4	1.5	5	0.3	1	0.6	2
K2	0.25	1.75	7	0.25	1	1	4	1	4	1.5	6	0.5	2	1.25	5
K3	0.14	0.98	7	0.98	7	0.42	3	0.98	7	0.14	1	0.28	2	0.56	4
K4	0.12	0.84	7	0.36	3	0.6	5	0.36	3	0.84	7	0.48	4	0.36	3
K5	0.11	0.11	1	0.11	1	0.77	7	0.11	1	0.77	7	0.77	7	0.77	7
K6	0.08	0.08	1	0.08	1	0.56	7	0.56	7	0.08	1	0.08	1	0.56	7
Celkem		4.66		3.88		5.15		4.21		4.83		2.41		4.1	
Pořadí		2		5		1		4		3		6		5	

Tabulka 8 - Vyhodnocení variant
Zdroj: vlastní zpracování

4.5 Výběr optimální varianty

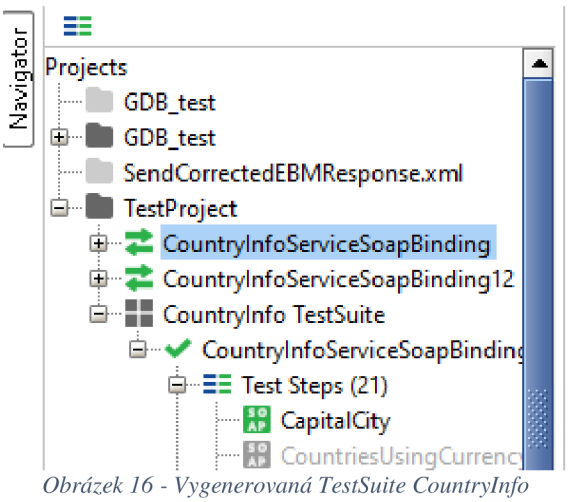
Porovnáním všech ohodnocených kritérií s přihlédnutím k váze byla jako optimální varianta vybrána varianta třetí, SOAP UI. Jako velmi dobré alternativní řešení se nabízí Selenium a IBM Rational. U Selenia je výhodou, že je zdarma, nástroj od IBM je poměrně drahý, ale nabízí nejvíce funkcí a nejlepší možnosti integrace. Z tohoto pohledu je SOAP UI dobré ve všech ohledech, není tak drahé a je široce použitelné. Je nutné si uvědomit, že je výběr silně ovlivněn výběrem kritérií a stanovením vah. Po úpravě jednoho nebo druhého by se patrně změnilo i pořadí, a tedy i optimální varianta. V další části se práce již bude zabývat pouze vybranou variantou.

4.6 Ukázkové případy užití SOAP UI

Představíme si dva ukázkové případy užití nástroje SOAP UI, jednou otestujeme webovou službu, v dalším případě otestujeme. Vzhledem k možnému vyzrazení obchodního tajemství nemohu použít příkladu nasazení, k ukázce tedy bude použita jiná webová služba.
K testování jsem vybrala službu
http://webservices.oorsprong.org/websamples.countryinfo/CountryInfoService ne je veřejná a pro účely prezentace dobře poslouží. Volím ji, protože v cílové testovací aplikaci jsou podobné informace (ISO kódy zemí a měst) často využívané.
Také se detailněji nebudu zabývat instalací a nastavením samotného nástroje, bylo k tomu již vydáno několik diplomových prací (k nalezení v seznamu zdrojů) a dá se nalézt množství dalších materiálů zabývajících se touto problematikou. Dobrým startem je například kurz samotného výrobce SoapUI, Smartbearu: https://www.soapui.org/gettingnestarted/soapnetest/.
Ukážeme si tvorbu functional test case a load test case, třetí typ, bezpečnostní testy prozatím vynecháme, budoucí testovaná aplikace je vystavena ve vnitřní síti a tyto testy tedy nejsou tak důležité a nebyly ani kritériem výběru testovací aplikace.

4.6.1 Test suite Countryinfo

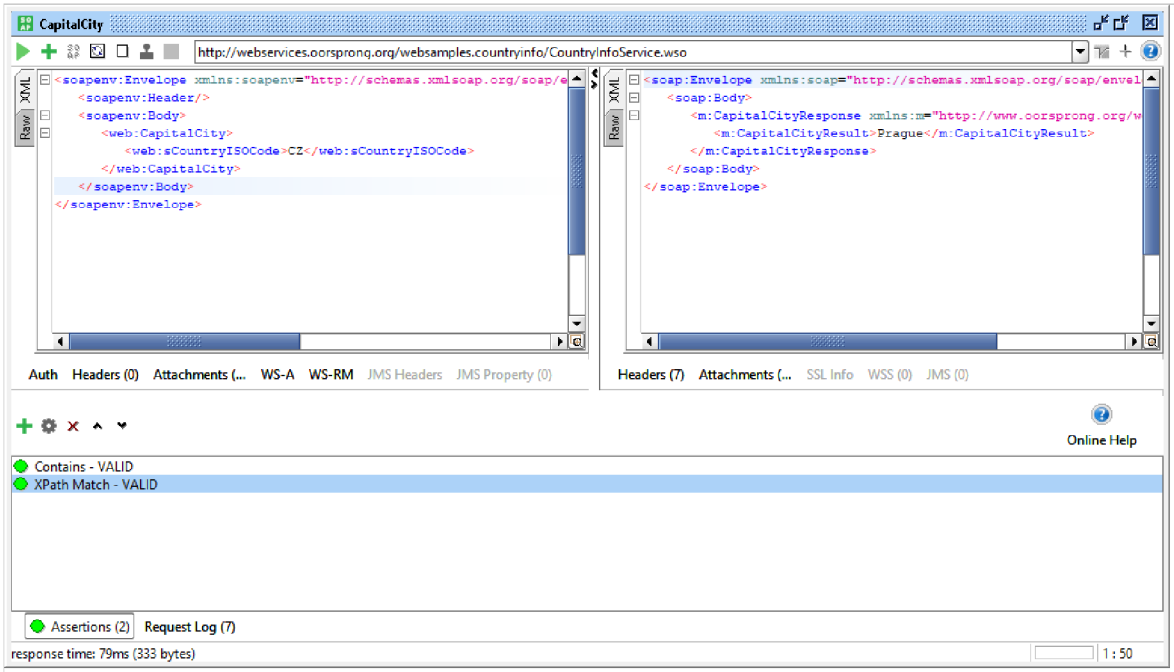
V programu SOAP UI byla vytvořena TestSuite CountryInfo. Pro lehčí práci jsme si automaticky vygenerovali TestCase.



Obrázek 16 - Vygenerovaná TestSuite CountryInfo
Zdroj: vlastní zpracování

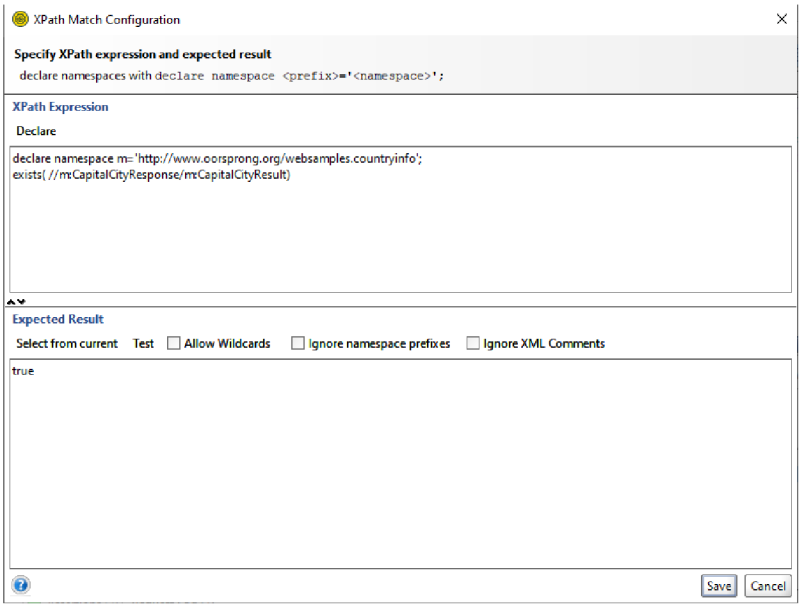
4.6.2 Vytvoření TestCase CapitalCity

Dotaz na webovou službu je náš testovací případ. Pro správnou funkčnost testcase musíme definovat co má být ve výsledku dotazu a vytvořit **Assertion**.



Obrázek 17 - SoapUI – Assertions Status
Zdroj: vlastní zpracování

Vytvořili jsme dvě assertion, jedna je druhu **Contains** a druhá **XpathMatch**. Po spuštění testcase program barevně rozliší, zda byla daná podmínka (assertion) splněna (zeleně) nebo nesplněna (červeně). Pokud jsou všechny podmínky splněny, je testcase ve stavu Passed. V opačném případě je testcase Failed.

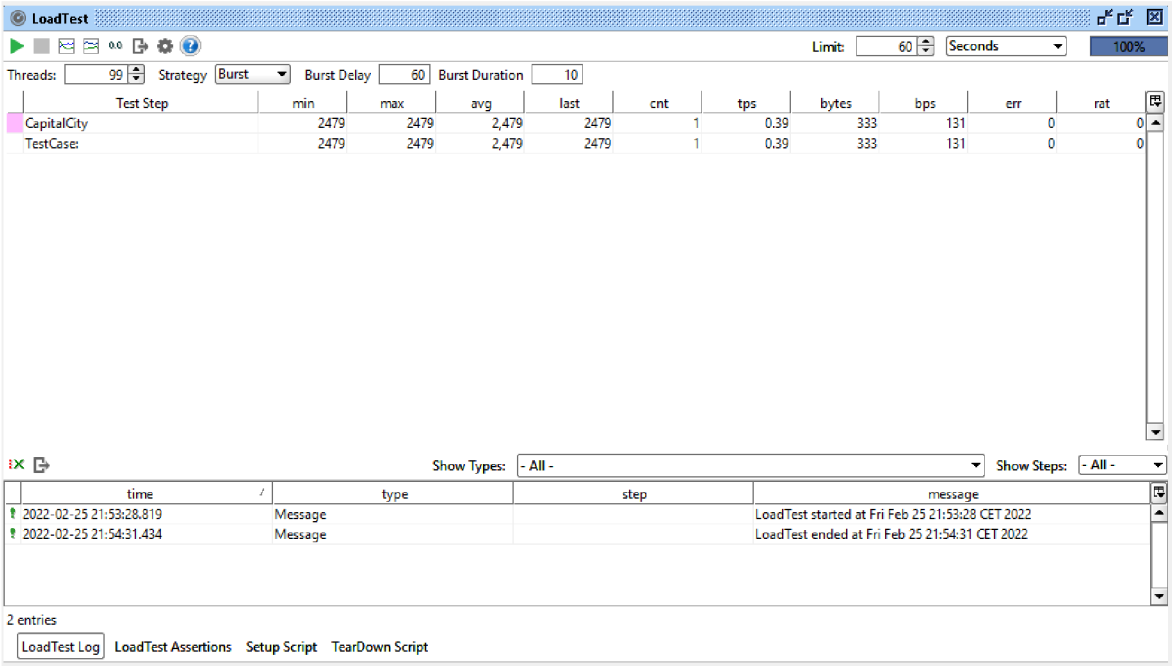


Obrázek 18 - SoapUI – Assertion Configuration
Zdroj: vlastní zpracování

4.6.3 Vytvoření TestCase LoadTest

SoapUI umožňuje automatické vytvoření Load Testu, pokud již máme vytvořený funkční test. Po kliknutí na náš TestSuite CountryInfo ho vytvoříme kliknutím na volbu **NewLoadTest**. Nastavíme parametry testu – test poběží 60 vteřin, použije 99 vláken (**Threads**). Použijeme volbu **Burst**, která vygeneruje krátkodobou vysokou zátěž. Další detaily nastavení LoadTestu jsou součástí dokumentace SoapUI. Po spuštění a dokončení testu se zobrazí sledované parametry:

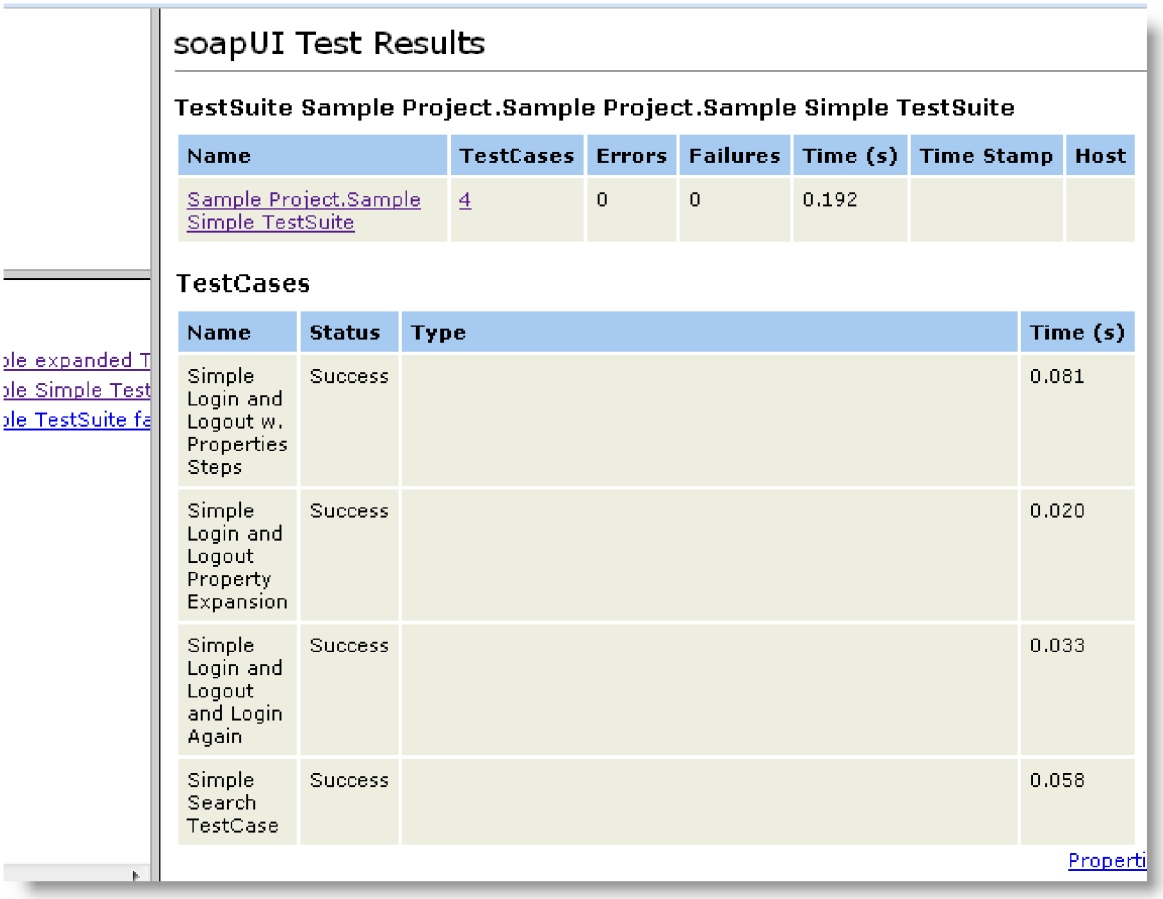
- **min** – minimální naměřená doba odpovědi testovacího kroku (v ms)
- **max** – maximální naměřená doba odpovědi testovacího kroku (v ms)
- **avg** – průměrná naměřená doba odpovědi testovacího kroku (v ms)
- **last** – průměrná naměřená doba odpovědi posledního běhu (v ms),
- **cnt** – kolikrát byl daný testovací krok proveden
- **tps** – počet transakcí za vteřinu
- **bytes** – kolik bytů bylo přesunuto
- **bps** – počet bytů za vteřinu
- **err** – počet chyb, které se běh testu objevily
- **rat** – procento neúspěšných dotazů



Obrázek 19 - SoapUI – LoadTest
Zdroj: vlastní zpracování

4.7 Návrh způsobu implementace do praxe

Zavést do fungujícího procesu nový nástroj či technologii je vždy ošemetné. Mohou nás potkat jak technické problémy, tak i problémy s širším akceptováním nástroje a ochotou ho používat. Je proto důležité si postup důkladně naplánovat a tým proškolit ohledně způsobu používání a výhod daného nástroje. Pro přehlednost si postup rozdělíme do několika fází.



soapUI Test Results

TestSuite Sample Project.Sample Project.Sample Simple TestSuite

Name	TestCases	Errors	Failures	Time (s)	Time Stamp	Host
Sample Project.Sample Simple TestSuite	4	0	0	0.192		

TestCases

Name	Status	Type	Time (s)
Simple Login and Logout w. Properties Steps	Success		0.081
Simple Login and Logout Property Expansion	Success		0.020
Simple Login and Logout and Login Again	Success		0.033
Simple Search TestCase	Success		0.058

Obrázek 20 - SoapUI Test Results
Zdroj: https://www.soapui.org/soapui/media/images/stories/reporting/htmlnereportnetestsuiteneview.png

4.7.1 Analýza test casů

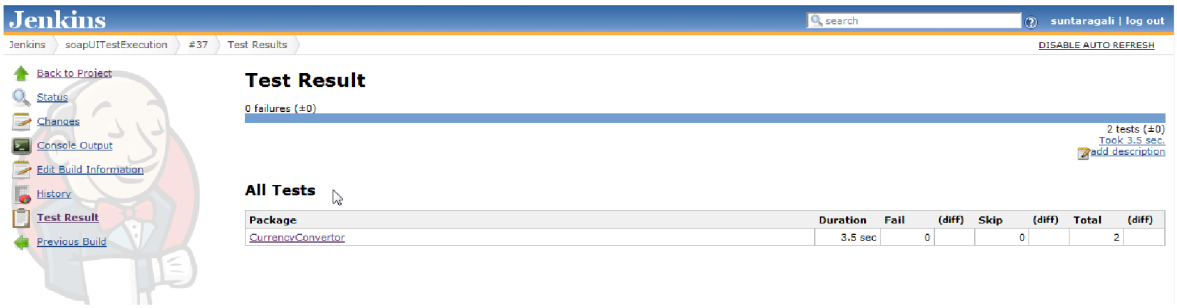
Projedeme testovací scénáře doteď používané a uložené v databází nástroje RTC. Identifikujeme scénáře vhodné pro automatizaci. Bude se jednat hlavně o vybrané funkční testy, dále regresní, integrační a performance testy. Zaměříme se na ty test casy, které spouštíme velice často a zaměřují se na nejdůležitější části aplikace, a dále všechny testy, které je třeba spouštět před každým releasem. Vzhledem k tomu, že se releasuje několikrát do roka, půjde o velkou úsporu času a testovacích kapacit. V současné době hlavně performance testy a regresní testy vytěžují velkou část kapacit po dlouhý čas.

4.7.2 Vytvoření testovacích scénářů

Vybrané testy projdeme a optimalizujeme pro spouštění v programu SoapUI. Připravíme testovací data, kde je to nutné a nastavíme správné Assertions. Využijeme hlavně XPath. SoapUI má svá specifika, je tedy nutné se na ně připravit. Pro rychlejší přechod navrhuji, aby tuto práci vykonávalo více testerů paralelně a současně, aby se každý nový test case, který bude vhodný rovnou automatizoval. Po měsíci bychom už měli mít solidní základnu testů, která bude přispívat ke stabilitě aplikace.

4.7.3 Integrace SoapUI do Jenkins

Jenkins může být se SoapUI integrovaný velice snadno. Stačí použít utilitu TestRunner.bat. Výsledky testů mohou být zobrazeny v grafické podobě a nebo je můžeme automaticky parsovat pomocí xml nebo csv. Potom mohou být testy přidány k jakémukoli buildu bude potřeba, můžeme nastavit noční spouštění pro LoadTesty, a tak se stanou neopomenutelnou součástí projektu.



Obrázek 21 - Jenkins test results
Zdroj:https://1.bp.blogspot.com/neqaNA6oURibY/UWp5fVkyubI/AAAAAAAAAaw/V2Fnea0qx_s4/s1600/TestResult.png

4.7.4 Školení testovacího týmu

Pro úspěch zavedení automatizace do testování je klíčové, aby všichni z testovacího týmu danou technologii a nástroj dobře znali a chtěli ho používat. Proto je nezbytné je v používání software proškolit, vytvořit základní scénáře a pomocí nich demonstrovat zjednodušení práce a zlepšení kvality celého software. Stejně tak je toto třeba obhájit i u managementu, aby na přípravu, a hlavně údržbu automatických testů byla vyhrazena dostatečná kapacita. V minulosti již podobné snahy proběhly a tato fáze byla podceněna.

5 Výsledky a diskuse

Výběr nejvhodnějšího testovacího nástroje nebyl tak jednoduchý, jak by se na první pohled mohlo zdát. Do hry vstupuje poměrně dost i protichůdných faktorů a kritérií, kdy např. nízká cena vylučuje širokou integraci a co nejjednodušší použitelnost. Přesto je již na trhnu dostatek vyhovujících nástrojů. Pro samotnou analytickou práci a vyhodnocení kritérií se Vícekriteriální analýza variant zdá jako velmi dobrá volba. Svojí strukturou a postupem navede k dosažení optimálního výsledku a pomáhá se na problém podívat strukturovaně a určit si kritéria pro všechny dílčí součásti rozhodovacího problému. Proto bych tuto možnost zvolila i příště při řešení podobného analytického problému či při rozhodování např. na manažerské úrovni. Nástroj se nehodí na rychlé reaktivní rozhodování, ale je určitě vhodný pro stanovování dlouhodobých cílů a strategií.

Seznámení se s nástrojem Soap UI a tvorba prvních testcasů proběhla dobře a jednoduše, použití je částečně intuitivní, i když samozřejmě pomáhá znalost testovací terminologie a procesů. Pro tento nástroj je k dispozici dostatek různých tutoriálů a návodů, a tak práci s ním zvládá i juniornější tester. Prvních 20 testcasů bylo vytvořeno během týdne a schváleno a zrevidováno testovacím týmem bez výhrad.

Performance testování jde s tímto nástrojem nastavit velice jednoduše, dobrá je možnost zkonvertovat na performance jakýkoli již vytvořený testcase. Tato možnost velice zefektivní před-releasovou přípravu, protože peformance se měří po každé změně a dosud bylo obtížné testy pokaždé přizpůsobovat nové funkcionalitě. Nyní se můžeme vždy zaměřit na specifickou změnu a k tomu pustit několik regresních testů.

Testy pomocí SoapUI jsme prováděli v rámci měsíce a tento způsob se osvědčil. Již použité a uložené testy se dají jednoduše revidovat a tvořit podle nich další testy. Integrace do Jenkins proběhla dobře, nyní je možné vybrané testy spouštět v rámci buildu a do budoucna se počítá s rozšířením a nastavením tzv. night testů a vytvoření robustních před-releasových regresních testů. Výhodou je, že na rozdíl např. od testů v JUnit většina testerů nepotřebuje podporu developerů a jsou schopni testy generovat sami a v požadované kvalitě.

Do budoucna očekáváme rozvoj a přibývání různých testů, pro každou novou funkcionalitu a aktualizace po každé změně stávajících. Časem se nejspíše stane výzvou údržba testů, protože v první fázi jsme se změnami až tak nezabývali. Je třeba najít a nastavit systém, jak vyhradit pravidelný čas na tuto údržbu a rozvoj regresních testů, tak abychom se vyhnuli tzv. falešnému pocitu bezpečí, kdy ty samé testy již po nějaké době nejsou schopny odhalovat nové chyby. Před touto výzvou ale stojí každý, kdo se pro automatizaci ve větším měřítku

rozhodne. Věříme, že do budoucna bude automatizace čas šetřit, a že také usnadní příchod nováčků do týmu a významně nám zjednoduší každodenní repetitivní práci.
Přínos aplikace SOAP UI hodnotím velmi dobře, stejně tak jako použitelnost tohoto produktu, jeho podporu a uživatelskou komunitu, která ohledně něj vznikla. Doporučila bych nasazení i na jiné projekty, kam by se z hlediska použitých technologií hodil.
Pokud bych měla zhodnotit samotné zpracování této práce, musím připustit, že mě v začátcích zaskočilo, jak je poměrně obtížné stanovit kritéria pro vícekriteriální analýzu a jak je správně ohodnotit, obodovat. Pokud bych tuto práci dělala znovu, od začátku bych věnovala více času důkladné analýze projektu a kritérií před započetím samotné vícekriteriální analýzy.
Přínos této práce do praxe je určitě v tom, že každý projekt, který hledá svůj automatizační nástroj může s drobnými modifikacemi použít stejný postup. Pravděpodobně se budou lišit technologie, možná přibude nějaké kritérium, ale jinak je tento postup univerzální a dovede nás k dobrému výsledku. Užitečná je také teoretická část práce, kde je rešerše důležitých pojmů z automatizace testování a přehled často používaných současných automatizačních nástrojů.

6 Závěr

Hlavním cílem práce bylo navrhnout implementaci automatizace testování softwaru v korporátním procesu s ohledem a specifické požadavky a kritéria jako je např. cena, výkon, bezpečnost. Tohoto cíle jsme dosáhli s použitím vícekriteriální analýzy variant, kdy na základě expertních odhadů a debaty v týmu byla stanovena kritéria pro výběr vhodného nástroje. Těmi byly znalost daného programu a technologií, kompatibilita se stávajícími užívanými technologiemi, cena za licenci, možnost integrace s dalšími využívanými aplikacemi, technická podpora a možnost performance testování.

V teoretické části byly pomocí rešerše odborných informačních zdrojů představeny pojmy a procesy týkající se automatizace testování softwaru. Kapitola 3.1 se zaměřila hlavně na cyklus vývoje softwaru a modely životního cyklu, které přímo souvisí se zvoleným způsobem testování na projektu. Dále byla připomenuta hodnota testování a v kapitole 3.6 byly představeny přínosy, ale i nevýhody automatizace testování. V kapitole 3.7 byl proveden výběr a představení nejvhodnějších a nejpoužívanějších automatizačních nástrojů. V kapitole 3.8 byly představeny nejdůležitější pojmy Vícekriteriální analýzy variant. Tím byl splněn první dílčí cíl práce, kterým byla analýza testovacích platforem a nástrojů a literární rešerše. Dalším cílem bylo porovnání vhodných řešení dle zvolených kritérií. Každé kritérium bylo vyhodnoceno pro každý zvažovaný nástroj a výsledky jsou shrnuty v kapitole 4.3. Zvolen byl pro jednoduchost bodovací systém. Na základě porovnání těchto kritérií byl splněn poslední dílčí cíl a vybrána nejvhodnější variantu nástroje pro použití na projektu. Tímto vítězným nástrojem se stalo SOAP UI. Hlavními klady této aplikace je univerzálnost použití, dobrá kompatibilita a široké možnosti integrace s dalšími aplikacemi a platformami a možnost jednoduché konfigurace performance testování.

V kapitole 4.6 byly pro tento nástroj připraveny 2 ukázkové případy užití, a navržena strategie implementace do praxe. Ta zahrnuje analýzu test casů, vytvoření testovacích případů v aplikaci, integraci těchto testů do Jenkins a školení celého testovacího týmu.

Hlavním přínosem této bakalářské práce je nalezení vhodného nástroje pro automatizaci testování ve specifické situaci. Díky návrhu způsobu implementace do praxe se začne s implementací tohoto nástroje do praxe. Velkým přínosem může být také teoretická část práce, kde nalezneme přehledně zpracované informace, důležité pro rozhodnutí o zavedení automatizace do praxe a také přehled automatizačních nástrojů. Část s vícekriteriální analýzou variant může být s drobnými úpravami kritérií či jejich hodnocení využita pro jakýkoli podobný výběr testovacího nástroje na libovolném projektu a může být také vhodně rozšířena.

7 Seznam použitých zdrojů

[1] BUREŠ, M., RENDA M., DOLEŽEL M. A KOL. Efektivní testování softwaru ne Klíčové otázky pro efektivitu testovacího procesu. Grada, 2016. ISBN: 978ne8024755946

[2] REŠ, RADIM. Zajištění kvality webových aplikací pomocí nástrojů automatického testování. Brno: VUT, 2014

[3] BOEHM, BARRY W. A Spiral Model of Software Development and Enhancement. TRW Defense Syst. Group, Redondo Beach, CA, roč. 21, č. 5, August 1988. ISSN: 0018ne9162

[4] KRUCHTEN, P. The rational unified process: an introduction. 3rd ed. Upper Saddle River: AddisonneWesley, 2004, xviii, 310 s. ISBN 03ne211ne9770ne4

[5] TOMÁNEK, MARTIN. Současný stav používání agilních metodik ve světě a v ČR. Acta informatica pragensia, roč. 4, č. 1, 2015. DOI: 10.18267/j.aip.48

[6] KRAUS, VOJTĚCH. Automatizované testování v moderním DevOps prostředí. Bakalářská práce. Praha: České vysoké učení technické v Praze, Fakulta informačních technologií, 2019.

[7] BUSCHOVÁ, LUDMILA. Zavedení agilní metody Scrum v IT organizaci. Praha: Vysoká škola ekonomie a managementu, Management firem, 2020, 134 s. [Citace 20. 2. 2022] https://theses.cz/id/wjaxlq/341447_bpdp_final.pdf

[8] ROYCE, WINSTON. 1970. Managing the development of large software systems. [Online] 1970.[Citace 2. 2 2022.] https://wwwnescf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf

[9] MÜLLER, THOMAS, DEBRA FRIEDENBERG A KOLEKTIV. Certifikovaný tester: Učební

[10] Osnovy pro základní stupeň [online]. B.m.: ISTQB. 2011 [Citace 12.3.2022]. http://castb.org/wpnecontent/uploads/2013/11/ISTQBneCTFLneSyllabusv2011neCZneBeta1.pdf

[11] BECK, K., BEEDLE, M., BENNEKUM, A. VAN, COCKBURN, A., CUNNINGHAM, W., FOWLER, M., ET AL. Manifest Agilního vývoje software. [Online] 2001.[Citace 4. 2 2022.] https://agilemanifesto.org/iso/cs/manifesto.html

[12] HLAVA, TOMÁŠ. Fáze a úrovně provádění testů. [online]. 21.08. 2011 [Citace 18.2.2022] http://testovanisoftwaru.cz/metodikatestovani/druhynetypyneanekategorienetestu/fazenetestu/

[13] The Software Development Process: A Complete Guide. [Online] 4.11.2021. [Citace 4. 2 2022.] https://taazaa.com/whitepaper/softwarenedevelopmentneprocess/

[14] APUTIME. Lean, kanban a APUtime [Online] 2020. [Citace 4. 2 2022.] https://www.aputime.cz/blog/leannekanbanneaneaputime

[15] BCT Services. Testing. [online]. [Citace 18.2.2022] https://www.bctservices.cz/testing/

[16] Základy automatizace testování. [online]. [Citace 18.2.2022] http://test.swtestovani.cz/index.php?option=com_content&view=article&id=40:zakladyneautomatizacenetestovani&catid=3:zaklady&Itemid=11

[17] Jenkins. [online]. [Citace 18.2.2022] https://www.jenkins.io/

[18] Seleniumhq.org [online]. [Citace 16. 2. 2022]. http://seleniumhq.org/docs/02_selenium_ide.html#seleniumnecommandsneselenese

[19] Soapui.org [online]. [Citace 16. 2. 2022]. https://www.soapui.org/

[20] Co je JMeter? [online]. [Citace 16. 2. 2022]. https://cs.educationnewiki.com/9105507newhatneisnejmeter

[21] What is IBM® Rational® Functional Tester? [online]. [Citace 16. 2. 2022]. https://www.ibm.com/products/rationalnefunctionalnetester

[22] GUI Test Drivers [online]. [Citace 16. 2. 2022]. http://testingfaqs.org/tnegui.html

[23] Axe Platform [online]. [Citace 16. 2. 2022]. http://www.qatestingtools.com/testingnetool/axe_platform

[24] DOUBRAVOVÁ, HANA BC. Vícekriteriální analýza variant a její aplikace v praxi. Diplomová práce. České Budějovice: jihočeská , univerzita v českých budějovicích, ekonomická fakulta, 2009.