Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvant. metod

Continuous Test Automation

Diplomová práce

Autor: Jakub Fečo
Studijní obor: Systémové inženýrství – Informační management
Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.
Hradec Králové, listopad 2020

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.
V Hradci Králové dne

Jakub Fečo

.....

Podpis

Poděkování:
Děkuji vedoucímu mé diplomové práce panu proděkanovi doc. Mgr. Tomáši Kozlovi, Ph.D. za jeho čas, cenné rady a odborné vedení. Děkuji společnosti Unicorn a Ing. Miroslavu Ježkovi za návrh tématu diplomové práce a jeho pomoc a ochotu v začátcích mého působení v oblasti testování softwaru.

Anotace práce:

Práce se zaměřuje na nástroje pro tvorbu a spouštění automatických testů, nástroje kontinuálního vývoje, nástroje pro test management a druhy automatizovaných testů, které lze implementovat a využít v rámci procesu testování při současném kontinuálním vývoji na projektu Core CC Tool 2.

První část práce vymezuje obor testování softwaru, jeho definici, rozebírá dělení testů, vymezuje test management, blíže popisuje automatizované testování, druhy a typy automatizovaných testů a metodologii DevOps s navazujícím tématem kontinuálního vývoje. Konec teoretické části se věnuje představení metodiky vytvoření automatických testů a jejich integraci s CI/CD a test management nástroji.

Druhá část práce se podrobně zabývá již definovanými druhy automatických testů a jejich integrací. Výsledkem této části práce je popis vytvoření variace automatických testů, které se zaměřují na určitou část testovaného softwaru a jejich integrace s dalšími nástroji.

Konec diplomové práce se věnuje sumarizaci nabytých teoretických a praktických znalostí vzešlých ze všech částí práce.

Annotation:

The work focuses on tools for creating and running automated tests, continuous development tools, test management tools and types of automated tests that can be implemented and used in the testing process with simultaneous continuous development on the project Core CC Tool 2.

The first part of the thesis defines the field of software testing, its definition, discusses the division of tests, defines test management, describes in more detail automated testing, types of automated tests and DevOps methodology with a related topic of continuous development. The end of the theoretical part is devoted to the introduction of the methodology of creating automatic tests and their integration with CI / CD and test management tools.

The second part of the thesis deals in detail with already defined types of automatic tests and their integration. The result of this part of the work is a description of the creation of a variation of automatic tests, which focus on a certain part of the tested software and their integration with other tools.

The end of the diploma thesis is devoted to the summary of the acquired theoretical and practical knowledge arising from all parts of the work.

Obsah

1 Úvod

V dnešním procesu testování se oblasti automatizovaného testování věnuje více času a prostředků, než tomu bylo dříve. Vývojové cykly v rámci sprintů se zrychlují a testování softwaru s nimi musí držet krok. Právě automatické testy a jejich zapojení do procesu testování je jednou z možností pro jeho zrychlení. Nezbytnost této disciplíny ještě více rezonuje v projektech, které si osvojily některou z agilních projektových metodologií. Ačkoliv je počáteční investice do automatizace značná, vyplatí se, pokud jsou automatické testy průběžně aktualizovány a používají se často v průběhu a na konci vývojového cyklu projektu.

Začátek diplomové práce nejdříve vysvětluje základní aspekty testování softwaru včetně jeho definice, jeho úrovní, rozdělení testů z více hledisek a poskytuje základní vhled do test managementu. Dále je představeno automatizované testování a jsou detailně popsány druhy automatických testů, které se mohou použít při testování na projektech společnosti Unicorn či jiných softwarových společností. Konec práce se věnuje metodologii DevOps a souvisejícímu tématu kontinuálního vývoje (continuous development) na projektech. Jsou vysvětleny jednotlivé fáze kontinuálního vývoje včetně definic a popisu úkonů, které se v nich činí a koncept CI/CD pipeline.

Další část práce poskytuje metodiku vytvoření sady automatických testů a popis jejich integrace s nástroji kontinuálního vývoje a test managementu. Metodika je poté uskutečněna v následujících kapitolách, ve kterých se dané testy prakticky implementují a integrují.

Tato diplomová práce a její výsledky slouží jako vhled či přehled možností automatizace testů společnosti Unicorn či týmům jiných softwarových společností, u kterých se projektové týmy rozhodnou pro zavedení DevOps metodologie s využitím CI/CD nástroje integrovaným s automatickými testy. Využít ji mohou také týmy testerů, které se rozhodnou využívat na projektu test management nástroj integrovaný s automatickými testy.

2 Cíl práce

Cílem diplomové práce je průzkum možností tvorby automatických testů ve vývoji softwaru a možností jejich integrace s nástroji kontinuálního vývoje a nástroji test managementu.

3 Testování softwaru

Testování je pro společnost klíčovou záležitostí a může jí ušetřit velkého poškození jména, ušetřit velké finanční náklady, přispět ke snížení rizika selhání během provozu systému a v neposlední řadě může také ušetřit životy. [1]

Jedním z pohledů, jakým lze na testování softwaru nahlížet, je pouhé hledání defektů v aplikaci. [2] Toto je ovšem pouze jedna z povinností v popisu práce desktopového, webového či mobilního testera softwaru. Obecně vzato je testování softwaru proces efektivního a řízeného zkoumání kvality softwaru, zaznamenávání defektů a kontinuálního reportingu o aktuálním stavu kvality.

Podle jiné definice je testování součástí procesu quality assurance (v překladu zajištění jakosti, zkráceně QA), který zajišťuje, že softwarové produkty a procesy v životním cyklu produktu odpovídají jejich specifickým požadavkům a dodržují jejich zavedené plány. [3]

Testování však samo o sobě kvalitu softwaru nezajišťuje. Testování pouze nachází a poukazuje na defekty, kontroluje, informuje a dohlíží na jejich nápravu.

V rámci tohoto dokumentu je testování softwaru chápáno jako „*Proces, který se skládá ze všech aktivit životního cyklu (dynamických i statických). Týká se plánování, přípravy a hodnocení komponenty nebo systému a souvisejících pracovních produktů s cílem určit, zda splňují specifikované požadavky a ukázat, že vyhovují účelu a najít defekty.*“ [4]

Testování je činnost různorodá a zahrnuje nejen samotné testování, ale v závislosti na testerské roli, může zahrnovat také činnosti manažerské, analytické či z části vývojářské. Samotný proces testování nemá jednotnou formu a může se velmi lišit v rámci použitých firemních metodik vývoje, technik testování či způsobu vykonávání. Právě automatizované testování, které je předmětem zkoumání v následujících kapitolách, je jednou ze způsobů vykonávání případů užití (use case).

3.1 Cíle testování

Určit cíle testování je jedním z požadavků pro to, aby všechny týmy sjednotily svá očekávání a aby se správně nastavila testovací strategie, která se následně využívá v rámci projektu.

Organizace ISTQB (International Software Testing Qualifications Board – nezisková organizace zabývající se testováním softwaru) určuje tyto typické cíle testování [2]:

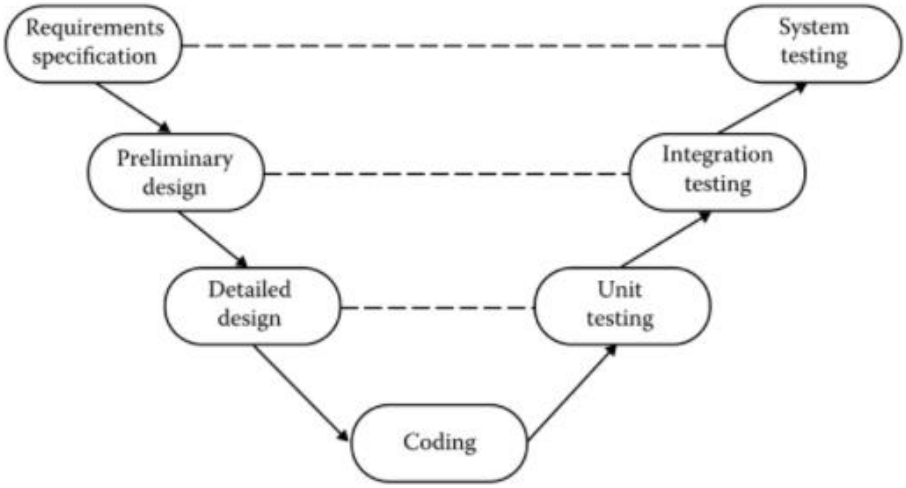
> *„• Předcházet vzniku defektů pomocí ohodnocení pracovních produktů, jako jsou požadavky, uživatelské scénáře, návrh a kód;*
>
> *• ověřit, zda byly splněny všechny specifikované požadavky;*
>
> *• zkontrolovat, že je testovaný objekt kompletní a validovat, že funguje tak, jak uživatelé a zainteresované strany očekávají;*
>
> *• vytvořit důvěru v danou úroveň kvality testovaného objektu;*
>
> *• odhalit defekty a selhání, a tím snížit úroveň rizika nízké kvality softwaru;*
>
> *• poskytnout informace zúčastněným stranám v dostatečné míře tak, aby mohly činit kvalifikovaná rozhodnutí, zejména pokud jde o úroveň kvality testovaného objektu;*
>
> *• dodržet smluvní, právní nebo regulatorní požadavky nebo normy a/nebo ověřit, zda testovaný objekt dosahuje shody s takovými požadavky nebo normami.“*

Další cíle testování mohou být přizpůsobovány na míru systému či testovanému modulu/části systému. [2]

3.2 Úrovně testování

Úrovně abstrakce v testování kopírují úrovně abstrakce obsažené ve vodopádovém modelu životního cyklu vývoje softwaru. V testování se úrovně abstrakce používají k jasnému odlišení úrovní testování a k určení cíle, kterým každá úroveň disponuje. [5]

Diagramová varianta vodopádového modelu, který organizace ISTQB nazývá „V-Model“ viz Obrázek 1, se zaměřuje na vztah mezi testováním a návrhovými úrovněmi systému. Dále z něj lze určit, že pro testování na nejnižší úrovni abstrakce je nejvíce vhodné takzvané unit testování, kdežto systémové testování je nejvíce vhodné pro testování na nejvyšší úrovni abstrakce, neboli testování požadavků na systém. [5]

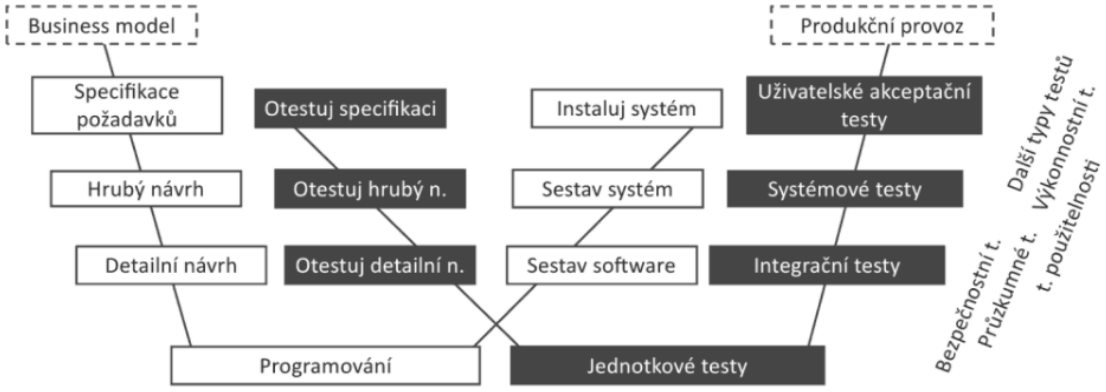


Obrázek 1 - V-model [5]

V praxi není za všechny úrovně testování zodpovědný pouze tester. O testování kódu na nejnižší úrovni abstrakce je většinou zodpovědný vývojář, který kód vytváří. Vývojář implementuje funkcionalitu a sám poté také implementuje související unit test. Následně může odevzdat svou práci společného repozitáře.

Problémem tohoto modelu je však jeho přílišná idealizovanost v případě provázání jednotlivých úrovní návrhu a testování. V mnoha společnostech nemusí k pravé části uvedené na Obrázek 1 vůbec dojít. Mnohdy je potřeba vycházet z více zdrojů specifikace, jelikož abstrakce pouze na základě explicitně vyjádřených potřeb zákazníka nemusí věrně obsáhnout vše důležité. Testeři se také mohou do procesu testování zapojit příliš pozdě. [6]

Tyto nedostatky eliminuje takzvaný „W-model“ viz Obrázek 2, který zohledňuje důležitost testerů již v raných fázích životního cyklu vývoje softwaru, ne až při systémovém a integračním testování. Výstižně také propojuje testerské aktivity s těmi vývojovými. [6]



Obrázek 2 - W-model [6]

3.3 Test Management

Test management znamená spravovat celý testovací proces. V rámci test managementu se spravují jednotlivé uživatelské případy užití v takzvaných test management nástrojích. Vytvářejí se v nich nové případy užití, které se přiřazují k zákaznickým požadavkům a popisují se jejich jednotlivé kroky do uceleného uživatelského scénáře. Seskupují se dle určitých parametrů či potřeb testovacího procesu. Zahrnují se do testovacích sad (test suite) přiřazených dané verzi softwaru. Zaznamenávají se u nich stavy případů užití po jejich vykonání v rámci testovací sady a případně se k nalezenému defektu přikládá ticket v project management nástroji. Vypracovávají se reporty a úvodní přehledy o stavu testovacího procesu.

3.4 Dělení testů

Testování softwaru je ze své podstaty komplexní záležitostí. Rozdělení všech druhů testů jen z jednoho úhlu pohledu, by bylo příliš komplikované. Teorie tedy dělí druhy testů, ve většině případů, do dvou různých dimenzí, ze kterých lze na testy pohlížet. Do dimenze časové a dimenze typové. Z hlediska těchto dimenzí lze poté určit další dělení testů dle modelů dimenzí kvality softwaru, dle fáze, dle realizace testů a dle používané techniky testování. Některé druhy testů se mohou objevovat ve více kategoriích současně.

3.4.1 Typy testů dle fáze testování

Časová dimenze rozděluje testy mezi fáze, které se na daném projektu v rámci testování provádějí a kopíruje výše zmíněné úrovně testování. Vývojový tým je zodpovědný za jejich implementaci a provádění unit testování. Testovací tým na straně dodavatele je zodpovědný za funkční, tovární, akceptační, systémové, regresní a performance testování. Testovací tým na straně zákazníka vykonává uživatelské akceptační testování, které prověřuje, že z pohledu zákazníka systém jako celek splňuje jeho požadavky a potřeby a lze ho použít na reálném či simulovaném provozním prostředí. [2]

Unit testy

Testování určité části kódu. Jsou nejblíže zdrojovému kódu softwaru. Testují se jednotlivé metody, funkce či třídy. [7]

Integrační testy

Testy pro ověření komunikace mezi jednotlivými moduly, ze kterých se skládá software, ověření komunikace mezi softwarem a operačním systémem či mezi softwarem a hardwarem. [8]

Funkční testy

Ověřují funkcionality požadované zákazníkem. Zjišťují, zda byly tyto funkcionality naimplementovány bez defektu a vykonávají přesně funkci či funkce, které pro ně byly zamýšleny (zahrnují smoke, regresní a další druhy testů).

Akceptační testy

Formální testy, které verifikují, zda software, či jeho nové funkcionality odpovídají požadavkům businessu. [7]

Systémové testy

Testování funkčnosti systému jako celku z pohledu zákazníka. Společně s integračními testy tvoří testovací fázi SIT (System Integration Tests). [8]

Performance testy

Testy nefunkčních výkonových vlastností softwaru. Mohou zjišťovat jeho chování při normální zamýšlené zátěži, při extrémní zátěži nebo mohou zjišťovat kde je zlomový bod zátěže, kterou již software není schopen unést.

Smoke testy

Rychlé funkční testy, které mají za úkol prověřit základní funkcionality softwaru a dát vývojářům a dalším zúčastněným stranám zpětnou vazbu o stavu softwaru, v co nejkratší době po jeho nasazení na čisté prostředí. [7]

Regresní testy

Tento druh testů zjišťuje, zda nově implementované funkcionality nezanášejí nové defekty do softwaru či zda nezpůsobují porušení funkčnosti u již dříve implementovaných a otestovaných funkcionalit.

3.4.2 Typy testů dle modelů dimenzí kvality softwaru

Jedno z elementárních dělení testů lze určit z modelu FURPS, který popisuje kvalitu softwaru. Tento model byl později doplněn firmou Hewlett Packard o typ +, který dal vzniknout novému modelu FURPS+. Podle tohoto modelu lze rozdělit testy na [9]:

1. Funkční (functionality);
2. použitelnostní (useablility);
3. spolehlivostní (reliability);
4. výkonnostní (performance);
5. testy na schopnost být udržován (supportability);
6. a testy dalších kategorií jako jsou designové omezení či požadavky na interface, požadavky na fyzické vlastnosti a požadavky na implementaci.

Pro porovnání lze uvést další z modelů, který popisuje kvalitu softwaru, je jím standard ISO/IEC 25010, podle tohoto standardu můžeme testy rozdělit do osmi typů [10]:

1. Funkční stability;
2. výkonnosti – efektivnosti;
3. kompatibility;
4. použitelnosti;
5. spolehlivosti;
6. bezpečnosti;
7. udržitelnosti;
8. přenositelnosti.

ISO/IEC 25010	
Funkční vhodnost	Úplnost, správnost, vhodnost
Výkonová účinnost	Chování v čase, využití zdrojů, kapacita
Kompatibilita	Koexistence, interoperabilita
Použitelnost	Rozpoznatelnost, učitelnost, estetika atd.
Spolehlivost	Zralost, dostupnost, odolnost proti chybám, obnovitelnost
Bezpečnost	Důvěrnost, integrita, odpovědnost atd.
Udržitelnost	Modularita, znovupoužitelnost, testovatelnost atd.
Přenositelnost	Adaptabilita, instalovatelnost, vyměnitelnost

FURPS+	
Funkčnost	Schopnost, znovupoužitelnost, bezpečnost
Použitelnost	Estetika, konzistence, citlivost atd.
Spolehlivost	Dostupnost, míra selhání, předvídatelnost atd.
Výkon	Rychlost, efektivita, propustnost, škálovatelnost atd.
Podporovatelnost	Testovatelnost, flexibilita, instalovatelnost atd.
+	Návrh, implementace, rozhraní, fyzické požadavky

Tabulka 1 - Porovnání modelů dimenze kvality softwaru [11]

3.4.3 Typy testů dle nutnosti spouštět testovaný software

Toto dělení od sebe odlišuje testy, které se dají vykonávat bez užití testovaného softwaru a testy pro které je nutné spustit a použít testovaný software.

3.4.3.1 Statické testování

Statické testování softwaru dokazuje, že testovat software lze i bez jeho použití a tím také potvrzuje, že by se testeři měli zapojit do testování již v raných fázích životního cyklu vývoje softwaru. Příkladem je analýza požadavků a designu ještě před následnou implementací. Techniky řadící se mezi statické testování jsou především statická analýza kódu prováděná automatickým nástrojem k tomu určenému, neformální a technická revize dokumentace a její připomínkování s členy týmu za účelem získání zpětné vazby. Mezi nesporné výhody využívání technik statického testování je především cena následné opravy nesrovnalostí v návrhu či dokumentaci oproti mnohonásobně dražší opravě chyby nalezené při testování v pozdějších fázích testování či na produkčním prostředí. [6]

Techniky statického testování jsou příkladem white-box techniky testování, které je předmětem dalších podkapitol.

3.4.3.2 Dynamické testování

Dynamické testování je pravým opakem statického testovaní a zahrnuje všechny ostatní techniky manuálního i automatizovaného testování. Určitými podkategoriemi, do kterých lze řadit další techniky testování, jsou black-box techniky, white-box techniky a grey-box techniky. Zařazení do těchto technik závisí na skutečnosti, zda dané testy využívají znalost vnitřního fungování zdrojového kódu softwaru, či nikoliv. Všechny black-box, white-box i grey-box techniky mohou být realizovány jak manuálně, tak automatizovaně. [12]

- **Black-Box technika**

  Tato technika testování v sobě zahrnuje všechny další techniky testování, které nevyužívají znalost vnitřního zdrojového kódu softwaru. Využívá se pouze vnitřní architektura softwaru a kontroluje se, že jsou všechny nutné vstupy korektně přijaty a software na ně reaguje poskytnutím korektního výstupu. [12]
- **White-Box technika**

  Protikladem k black-box technice je white-box technika, zahrnující techniky využívající znalost vnitřního zdrojového kódu softwaru. [12]
- **Grey-Box technika**

  Některé zdroje uvádějí ještě takzvané grey-box testy jakožto předěl mezi dvěma předešlými druhy testů. Při práci s touto technikou je testerům alespoň trochu známa vnitřní stavba softwaru. [12]

3.4.4 Typy testů dle jejich realizace

Dalším typem dělení testů je podle typu jejich realizace na automatizované a manuální. Tato diplomová práce se zaměřuje na průzkum automatických testů a jejich integrací s nástroji. Proto je manuální testování definováno pouze stručně a automatizovaným testováním a automatickými testy se práce zabývá v dalších kapitolách.

- **Manuální testování**
  Tester vykonává předem definovanou testovací sadu, ve které se nachází jednotlivé případy užití pro testování v daném běhu testování.

  Uvnitř každého z těchto případů užití je k nahlédnutí scénář celého testu a manuální tester tedy ví, jaké akce má vykonat a jaké reakce systému má očekávat. V momentu nalezení defektu softwaru může tester zaznamenat chybový krok scénáře a zároveň defekt propojit s project management nástrojem (Jira či jiné nástroje).
- **Automatizované testování**
  Automatizované testy jsou testy, které jsou realizované bez přičinění lidského faktoru. Vytvářejí se pomocí softwarového nástroje a vykonávají opakovaně testovací akce, které by v opačném případě musel tester vykonat manuálně. Jednotlivé akce jsou následně zaznamenány se statusem prošel (passed) či neprošel (failed), což určuje status celého automatického testu tak, jak ho tester naimplementuje. Tyto testy mohou být, ekvivalentně jako u manuálního testování, propojeny s rozličnými CI/CD nástroji či test management nástroji. Dále je automatizované testování popsáno následující kapitole.

4 Automatizované testování

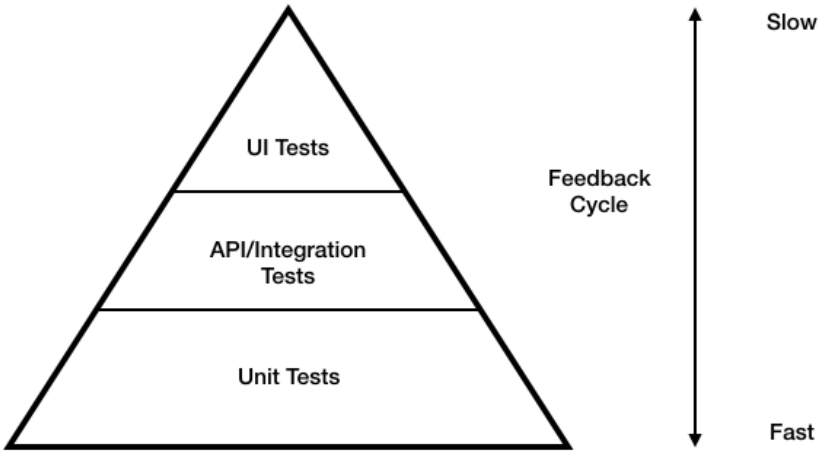
Automatizované testování se v posledních letech neustále se zrychlujícího procesu vývoje softwaru těší větší popularitě. V agilním prostředí je tento trend ještě markantnější, jelikož je opakovaně zapotřebí velkého množství regresního testování v rámci každého vývojového cyklu, které testerům zabírá podstatnou část času. Tu by naopak mohli využít lépe při testování nových funkčností. Dále poskytuje pro vývojáře po odevzdání kódu do repozitáře rychlou zpětnou vazbu a pro management poskytují alespoň přibližnou představu stavu sestavení (buildu). Automatizace testů je tedy jednou ze základních činností každého agilního procesu vývoje softwaru.

Před začátkem automatizace je potřeba si ujasnit, jaké mají všechny zainteresované strany očekávání. Příkladem špatně nastaveného očekávání je myšlenka automatizace všech případů užití softwaru. Bohužel, s přibývajícími automatickými testy přibývají také náklady na jejich údržbu a neustálou aktualizaci.

Dalším příkladem špatného očekávání je vidina nahrazení části manuálních testerů automatickými testy. Intuice a kreativita testera při hledání defektů jsou vlastnosti, které automatické testy nikdy mít nebudou (polemizovat lze nad moderními AI nástroji pro testování GUI). Testy dělají jen ty akce, které mají naimplementované a jejich podoba a důkladnost závisí na důvtipu a znalostech automatizačního testera, který má většinou za úkol automatizovat pouze happy day případy užití (základní případ užití funkcionality bez výjimkových či chybových situací).

Automatizace funkčního testování na straně dodavatele se nejčastěji týká vytvoření smoke testů, které mají za úkol zjistit stabilitu nasazeného sestavení (buildu) a regresních testů, které kontrolují, zdali nově naimplementované funkcionality zpětně nedestabilizovaly již hotové funkcionality. Případně se může jednat o automatizaci nefunkčního testování výkonnosti softwaru. V takové situaci se vytvářejí performance, stress či load testy, podle potřeb projektu.

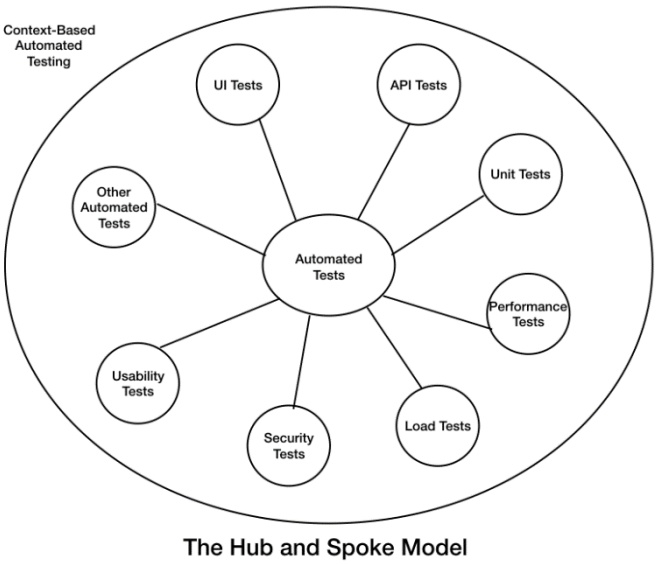
Existují tři úrovně automatizace testů, které jsou na projektech realizované. Úrovně znázorňuje Obrázek 3. Z automatizační pyramidy uvedené na obrázku lze odvodit přibližné množství každého druhu automatických testů prováděných při testování. [13]



Obrázek 3 - Automatizační pyramida [13]

Další možností, jak hledět na plánování automatizovaného testování je kontextově založený model, který se, spíše než na úrovně automatických testů, zaměřuje na typy automatických testů. Takzvaný „hub and spoke“ model plánování automatizovaného testování znázorněný níže, viz Obrázek 4. [14]

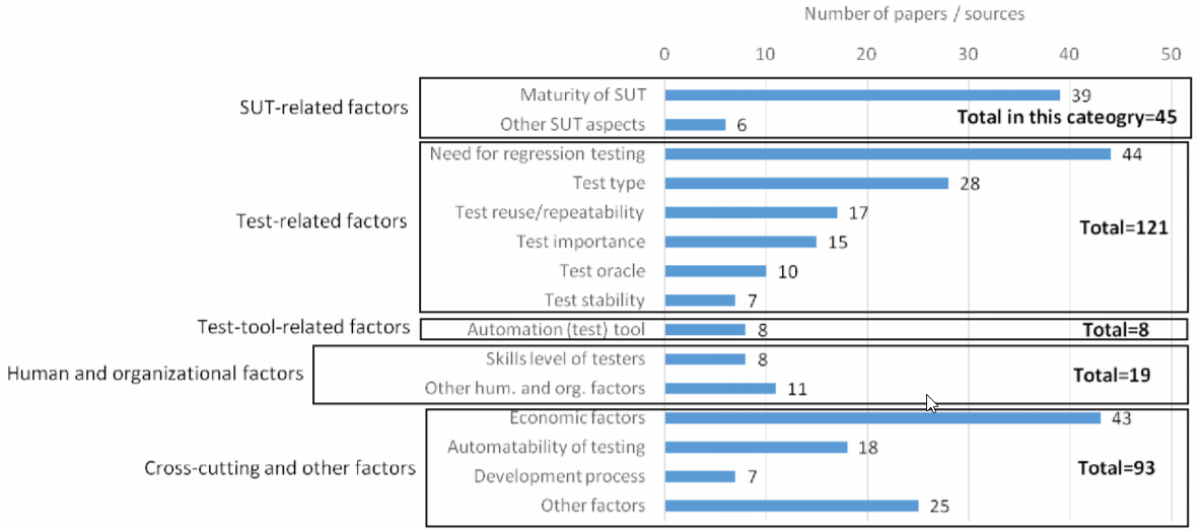
Druhy automatických testů jsou více probírány v dalších kapitolách.



Obrázek 4 - Hub and spoke model typů automatických testů [14]

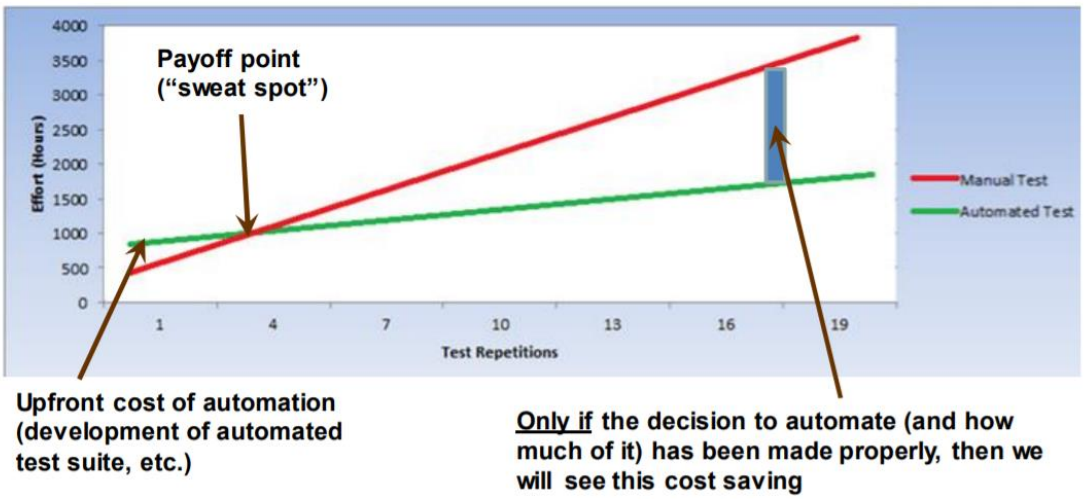
4.1 Kdy a co automatizovat

Studie zaměřené na podporu rozhodování v automatizaci testů poskytují většinou nesystematické a slabě empiricky podložené rady. Obecně má na rozhodování v automatizaci testů největší vliv patnáct faktorů, které společně vytváří pět odlišných skupin. Těmito skupinami jsou faktory spojené s testovaným softwarem, faktory spojené s testem samotným, faktory související s testovacími nástroji, lidské a organizační faktory a faktory průřezové. [15]



Obrázek 5 - Faktory – kdy a co by se mělo automatizovat [15]

Obrázek 5 již představuje ony konkrétní faktory v každé z pěti skupin. Největší vliv na rozhodnutí, zda se test bude automatizovat má faktor potřeby regresního testování následovaný ekonomickými faktory, pod kterými si lze představit návratnost investice do automatizace testů. Investice by se měla vrátit, pokud by se daný test měl provádět vícekrát, například v každé iteraci agilního projektu viz Obrázek 6. [15]



Obrázek 6 - ROI automatizace testů [15]

5 Automatické testy

Automatické testy jsou člověkem implementované skripty, skládající se z programových příkazů, které dohromady představují kroky určitých scénářů případů užití softwaru. Takto automatizované testy umožňují opakované spouštění přes software buď lokálně na počítači nebo vzdáleně pomocí nástrojů kontinuálního vývoje, které spouští software na serveru. Na základě jejich konečného stavu či stavů v průběhu vykonávání lze určit, zda daná funkcionalita, komponenta či nefunkční vlastnost testovaného softwaru, je funkční či nikoliv. Správně nastavené automatizované testování využívá automatické testy, které prověřují testovaný software ve více vrstvách. Danou komponentu či funkcionalitu například ověřuje jeden test na frontend (GUI) a druhý test ověřuje backend.

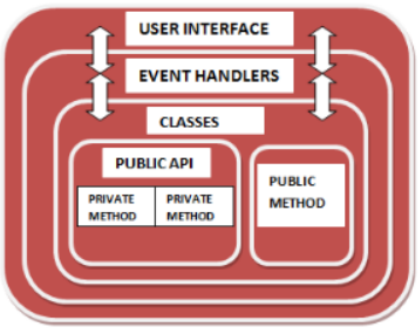
Výhodou automatických testů je možnost jejich opakovaného spuštění, rychlost a absence lidských chyb při jejich vykonávání. Při neustálém nárůstu počtu automatických testů jsou nevýhodou současně rostoucí náklady na jejich údržbu. [15]

5.1 Request-based testy

Request-based testy jsou typem automatických testů, které jsou založené na posílání requestu prostřednictvím vybrané http metody v určitém formátu na API (aplikační programovací rozhraní) a validace response ve stejném formátu, které API posílá nazpět. Stejně jako je to v request-response komunikaci mezi klientem a serverem. API se zde chová jako interface, který umožňuje komunikaci mezi programy.

„API jsou kolekce procedur a funkcí, které mohou ostatní aplikace využívat ke splnění svých funkcí.“ [16]

„Aplikační programovací rozhraní neboli API je část kódu, která umožňuje interakci dvou softwarových komponent.“ [17]



Obrázek 7 - API testing [16]

Způsobem posílání requestu a přijímání response jsou následně vytvářeny jednotlivé testy. Účelem tohoto typu testů je zjistit, jestli se lze připojit k API, zda poskytuje spolehlivé zabezpečení a zda funguje podle očekávání. [18]

Kromě využití ve funkčních testech, jako jsou například smoke testy, integrační testy či regresní testy, mohou být request-based testy využity i v nefunkčních testech, jako jsou performance, load či stress testy. Mohou se využít i v automatických testech pro grafické uživatelské rozhraní (GUI), pokud je třeba vykonat určité prerekvizity testu před započetím GUI testu. API testování je zásadní, pokud chce společnost dodat funkční, spolehlivý a bezpečný software. Stačí jedno nefunkční API a je ovlivněna funkčnost celého softwaru. [17]

Jelikož je manuální testování API ve větších společnostech velice náročné a rizikové, využívá se převážně automatizovaného přístupu. Výhodou použití API testování je relativní rychlost oproti GUI testování. V agilním prostředí je rychlost testů, a tím i odvíjející se rychlost zpětné vazby vývojářům zásadní. [19]

Dalšími výhodami request-based API testování jsou [19]:

- Rychlejší vyřešení defektů – při selhání testu v určité API metodě;
- lehčí údržba – oproti neustále se měnícímu GUI je API stabilnější a logika funkcí ani response, které se validují, se tolik nemění;
- rané testování – jakmile je logika navrhnuta a funkce implementovány, testy mohou být vytvořeny a mohou validovat response bez naimplementovaného GUI či bez propojení jednotlivých komponent izolovaně.

API rozhraní se dělí do dvou nejpopulárnějších druhů [17]:

1. SOAP (Simple Object Access Protocol);
2. REST (Representational State Transfer).

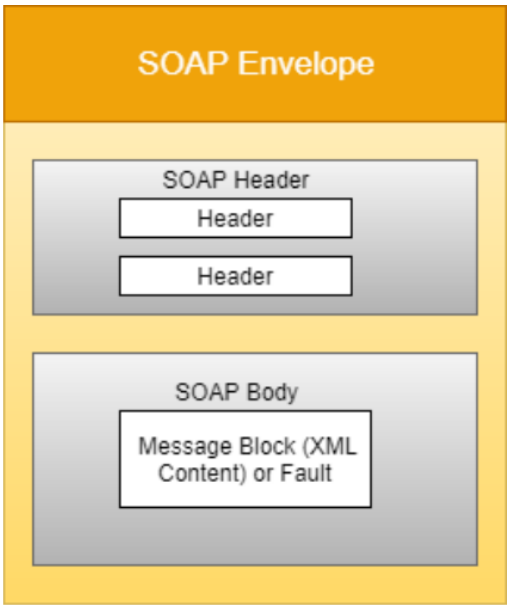
5.1.1 SOAP API

Simple Object Access Protocol je protokol pro zasílání zpráv. Tento protokol používá pro komunikaci po síti pouze formát XML. SOAP je základní součástí požadavků na architekturu orientovanou na služby (SOA) a požadavky webových služeb, které souvisejí se SOA. [20]

Narozdíl od REST striktně dodržuje určité standardy. Dodržuje strukturu zpráv, sadu pravidel kódování a standardy pro poskytování requestů a responsů a v určitých případech je bezpečnější než REST. Jednou z integrovaných funkcí SOAP je možnost vytvářet webové služby, které umožňují zpracovávat komunikaci a vytvářet response nezávislé na jazyce a platformě. [21]

Obrázek 8 znázorňuje SOAP formát zprávy.

- Envelope (obálka) – Určuje začátek a konec SOAP formátu zprávy a obsahuje hlavičku a tělo zprávy. Je u SOAP formátu zprávy nezbytná.
- Header (hlavička) – Volitelný element, který obsahuje například informace o autorizaci. SOAP formát zprávy definuje dva druhy hlaviček:
    1. Content-Type – specifikuje MIME typ pro zprávu
    2. Content-Length – určuje požadovaný počet bajtů v těle requestu a repsonse
- Body (tělo) – Nezbytný element obsahující XML data. Obsahuje request a response informace.
- Fault (chyba) – Element obsahující informace o případné chybě, která může nastat při poslání zprávy. [20]



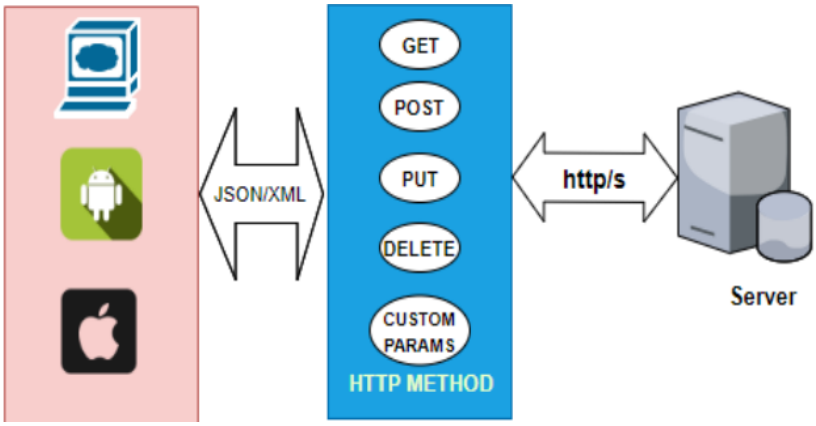
Obrázek 8 - Formát SOAP zprávy [20]

5.1.2 REST API

REST (Representational State Transfer) poprvé představil Roy Fielding jako architektonický styl pro zlepšování webových služeb. Jedná se o softwarovou architekturu, která se spoléhá na bezstavový komunikační protokol, nejběžnějším je HTTP. Oproti SOAP je více datově než funkčně zaměřený. [22]

K povolení komunikace jsou k dispozici typy requestu [19]:

- GET: Získání či čtení dat;
- POST: Přidání nových dat;
- PUT: Aktualizace již existujících dat;
- DELETE: Smazání existujících dat.



Obrázek 9 - REST Flowchart [20]

REST vytváří data v XML, YAML nebo jiných strojově čitelných formátech, ale JSON je nejběžněji používaný formát. [22]

5.1.3 Rozdíly mezi SOAP a REST API

REST a SOAP jsou dnes dvě nejpoužívanější API. REST je architektonický styl kdežto SOAP elementární protokol. Obě API se používají ke zajištění komunikace mezi programy a každé z nich má své vlastní výhody a nevýhody. [20]

SOAP má tyto výhody [23]:

- Nezávislý na jazyku, platformě i protokolu transportní vrstvy;
- lépe využitelný při testování distribuovaných systémů;
- standardizovaný;
- vbudované odstraňování chyb.

REST má oproti SOAP tyto výhody [23]:

- Více srozumitelný;
- rychleji naučitelný;
- efektivní (použití menšího formátu zpráv JSON);
- rychlý.

5.2 Selenium-based testy

„Selenium je záštitný projekt pro variaci nástrojů a knihoven, které podporují automatizaci webových prohlížečů.“ [24]

Selenium-based testy umožňují či vykonávají testy grafického uživatelského rozhraní (GUI) prostřednictvím automatizace instrukcí koncových uživatelů při používání webových prohlížečů. Těmito instrukcemi mohou být například klikání myší, vpisování textu do textových polí, výběr z nabídky, zaškrtávání, odkazování se na další stránky či dokumenty, scrollování myší atd. Může tak simulovat uživatelské chování.

5.2.1 Selenium IDE

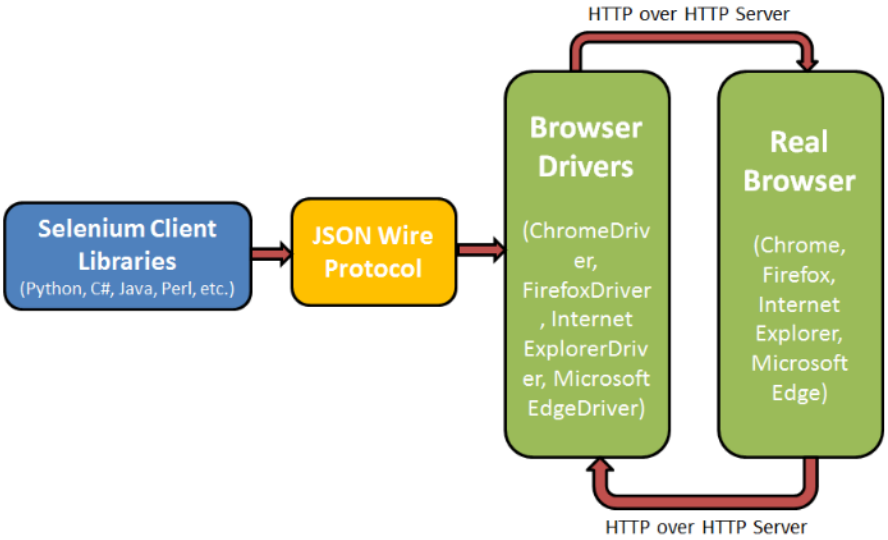
Rozšíření pro prohlížeče Chrome a Firefox, které dokáže nahrávat uživatelské akce a vytvářet test skript s uživatelskými akcemi přímo při jejich manuálním vykonávání v prohlížeči. [24]

5.2.2 Selenium WebDriver

Páteřním projektem Selenia je WebDriver, který poskytuje interface pro implementaci sad instrukcí, které mohou v nezměněné podobě běžet na více druzích prohlížečů na různých operačních systémech a druzích výpočetní techniky. [24]

WebDriver poskytuje určitou abstrakci nad rozdíly, které se vyskytují při vykonávání stejných instrukcí různými prohlížeči. Instrukce ve formě kódu nad touto abstrakcí mohou být implementovány jen jednou a budou funkční pro všechny druhy prohlížečů. [25]

Obrázek 10 znázorňuje architekturu Selenium WebDriveru.



Obrázek 10 - Selenium WebDriver architektura [26]

Selenium Client Libraries

Tester může psát kód pro použití Selenia ve více jazycích za pomoci Selenium Client Libraries, které si může stáhnout online. Například pro zápis Selenia pomocí jazyka Java si tester musí stáhnout Java Client Library. [26]

JSON Wire Protocol

Tento protokol představuje REST API, které zjednodušuje přenos dat mezi klientem a serverem. [26]

Browser Drivers

Drivery představují platformu pro komunikaci s různými druhy prohlížečů, která zajišťuje vrstvu zapouzdření, což udržuje abstraktnější úroveň ovládání. Implementace, architektura, funkcionality či obecně vnitřní logika prohlížečů není známa a je pro testery černou skříňkou. Každý prohlížeč nabízí svůj vlastní browser driver. Například Chrome nabízí vlastní ChromeDriver, Firefox nabízí GeckoDriver a Edge nabízí svůj EdgeDriver. [26]

Browsers

Všechny populární prohlížeče, které nabízí své drivery, které lze použít pro GUI testy. [26]

Příkladem: Chrome, Firefox, Internet Explorer, Edge, Safari, Opera.

5.2.2.1 Lokální a Vzdálený WebDriver

Selenium WebDriver lze použít jak lokálně přes výpočetní techniku, na které je umístěn test tak vzdáleně přes takzvaný vzdálený (remote) WebDriver. Vzdálený WebDriver má dvě části, klienta, což je WebDriver test a server, kterým je Java servet hostovaný na jakémkoliv Java EE aplikačním serveru. [27]

5.2.2.2 Selenium Grid

Selenium Grid umožňuje pouštět více WebDriver testů paralelně na více strojích a řídit tak různé verze a konfigurace prohlížečů centrálně. Může vyřešit mnoho delegačních a distribučních problémů. [28]

Dalším důvodem pro využívání Selenium Gridu je snížení času pro vykonání GUI testů. Umí též řídit uzly, ve kterých prohlížeče vykonávají testy a umožňuje provádět GUI testy na více operačních systémech. [29]

5.3 Testy s příkazy pro OS

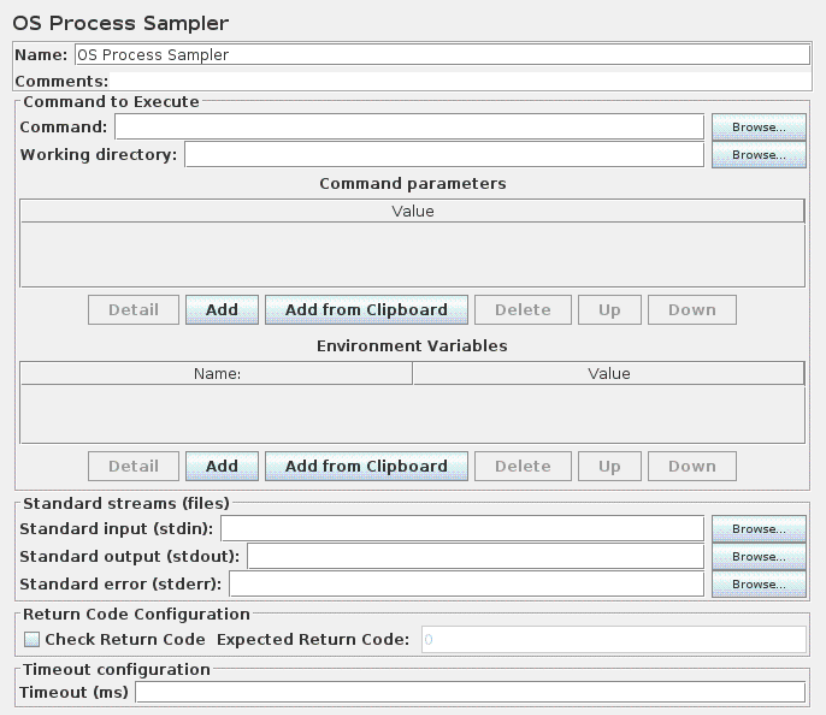
Automatické testy mohou vykonávat příkazy pro operační systém na lokálním stroji. Využití tyto testy naleznou především jako doprovodné k requestu-based testům či k Selenium-based testům. Dokážou provést určité příkazy operačního systému, které hlavní test potřebuje k úspěšnému pokračování, dokončení testu či pro kontrolu výsledků testu. Příkladem jsou:

- diff (na Linux) nebo fc (na Windows): pro porovnávání souboru s očekávaným souborem,
- du (na Linux) a dir (na Windows): pro zjištění velikosti souboru příkazem.

Dalšími možnostmi využití testů s příkazy pro OS jsou [30]:

- *„Spuštění skriptu;*
- *čtení logu;*
- *provedení změny konfigurace;*
- *nasazení / spuštění testované aplikace před zahájením testu a zastavení / odinstalování po skončení testu.*

Obrázek 11 obsahuje ukázku, jak lze vytvořit test s příkazem pro lokální operační systém v nástroji Apache JMeter.



Obrázek 11 - OS Process Sampler JMeter [31]

5.4 Testy s prací v databázi

Testují určitý databázový server. Pro přístup k relačním databázím se využívá Java Database Connectivity (JDBC), přístup k nerelační databázi poskytují standardní skriptovací API pro Java Virtual Machine (JVM) a jazyky jako je JSR223. [32] [33]

Pro vytvoření testu, který pracuje s databází je potřeba nejdříve nastavit jeho konfiguraci. JDBC, MongoDB či jiný driver podle testovaného databázového serveru musí být vložen do složky lib vybraného nástroje, ve kterém se test vytváří. Dále se musí nakonfigurovat URL adresa databázového serveru. Pokud je třeba také přihlašovací jméno a heslo. [34] [35] Poté již nic nebrání vytvoření testovacího requestu s databázovým dotazem. Tento test poté může prověřovat například:

- Vytvoření záznamu/dokumentu v databázi;
- aktualizaci záznamu/dokumentu v databázi;
- získání záznamu/dokumentu v databázi;
- získání více záznamů/dokumentů v databázi najednou;
- smazání záznamu/dokumentu v databázi.

5.5 Výkonnostní testy

Výkonnostní neboli performance automatické testy nalézají využití při testování nefunkčních parametrů softwaru.

Těmito parametry jsou například [36]:

- *„Rychlost;*
- *doba odezvy;*
- *stabilita;*
- *spolehlivost;*
- *škálovatelnost;*
- *využití prostředků softwaru při konkrétní pracovní zátěži.“*

Výkonnostní testování poskytuje důležité informace o rychlosti, stabilitě a škálovatelnosti softwaru při očekávaném zatížení všem zúčastněným stranám a managementu. Vývoji pomáhá určit, co by se mělo v rámci nefunkčních parametrů zlepšit v budoucích verzích softwaru. Bez výkonnostního testování, a tím také s nezjištěnými nefunkčními problémy softwaru, ztrácí jak společnost, tak i samotný software na prestiži. [36]

Nejvyužívanějšími typy výkonnostních testů jsou zátěžové testy (load testing) a stres testy (stress testing) [36]:

- Load testing – prověřuje schopnost softwaru fungovat pod očekávanou zátěží;
- stress testing – zjišťuje fungování softwaru pod velmi vysokou zátěží a zjišťuje zlomový bod, kdy aplikace již přestává zátěž zvládat.

Výkonnostní testování	**Zátěžové testování**	**Stres testování**
Nadmnožina zátěžového a stres testování.	Podmnožina výkonnostního testování.	Podmnožina výkonnostního testování.
Pomáhá nastavit benchmark a standardy pro aplikaci.	Pro zjištění, jak se software chová při normálním a vyšším zatížení.	Kontrola, jak se software chová při extrémním zatížení a jak se zotavuje z poruchy.
Cílem testování výkonu je získat informace o tom, jak se aplikace chová za běžných parametrů.	Generování zvýšené zátěže webové aplikace je hlavním cílem zátěžového testování.	Cílem zátěžového testování je zajistit, aby nedošlo k selhání serverů.

Využití zdrojů, dostupnost a spolehlivost produktu jsou ověřovány v rámci tohoto testování.	Atributy, které se kontrolují při zátěžovém testu, jsou výkon ve špičce a doba odezvy.	Tento druh testování kontroluje dobu odezvy stability a další nefunkční parametry.
Při testování výkonu je limit zatížení jak pod, tak nad prahovou hodnotou zlomového bodu.	Při testování zátěže je limitem zatížení zlomový práh.	Při zátěžovém testování je limit zatížení nad prahovou hodnotou zlomového bodu.

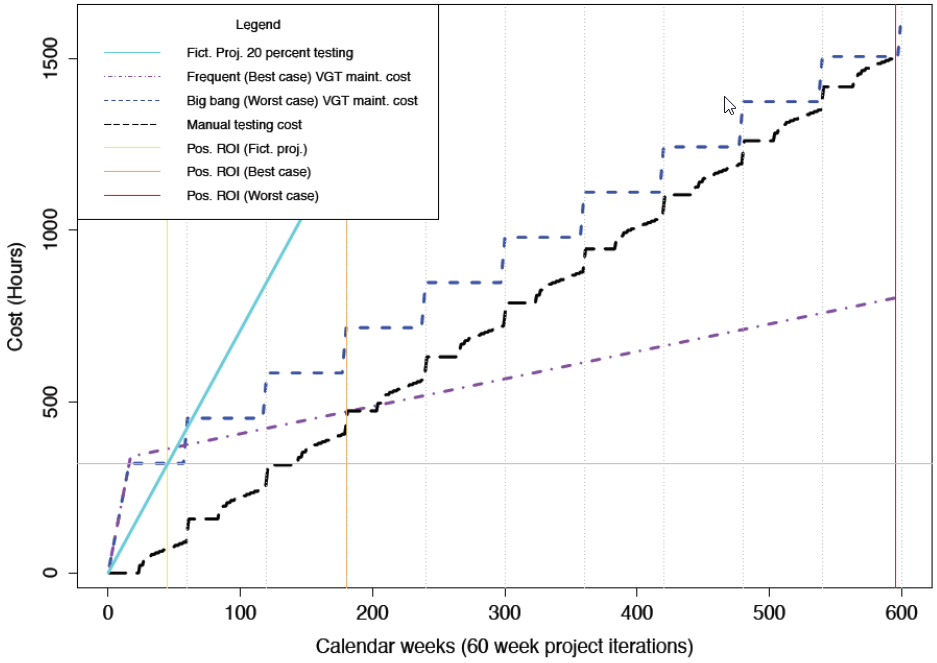
Tabulka 2 - Rozdíly mezi výkonnostním, zátěžovým a stres testováním [37]

5.6 Údržba automatických testů

Ukazuje se, že v případě automatických GUI testů jsou náklady největší na počátku jejich vývoje. Náklady na jejich údržbu jsou významně nižší a na základě frekvence úkonů údržby se náklady nepřímo zvyšují či zmenšují. Platí, že čím méně je frekventovaná průběžná údržba automatických GUI testů, tím jsou pozdější náklady na hromadnou údržbu vyšší. [38] Stejně tak může náklady na údržbu zvýšit zvětšování množství udržovaných automatických GUI testů.

Přesto jsou automatizované testy, za předpokladu jejich častého spouštění, finančně výhodnější než časté manuální testování stejných případů užití. [15] [38]

Obrázek 12 znázorňuje vývoj nákladů na vývoj, údržbu či vykonání v čase (týdny) za automatické GUI testy (udržované metodou big-bang najednou nebo frekventovaně) a manuální testy.



Obrázek 12 - Přibližné náklady na vývoj, údržbu či vykonání testů za čas [36]

6 DevOps a Continuous development

Pojem DevOps rozšiřuje agilní metodiku, zkratka je složenou slov development (Dev) a operations (Ops). V rámci DevOps se vykonávají tyto úkoly [39]:

- „*Build a release management;*
- *deployment management;*
- *administrace systému pro správu verzí;*
- *správa softwarových konfigurací;*
- *automatizace všech druhů;*
- *implementace continuous integration;*
- *implementace continuous testing;*
- *implementace continuous delivery;*
- *implementace continuous deployment;*
- *správa cloudu a virtualizace*.“

„*DevOps, představuje spojení lidí, procesů a technologií, jehož cílem je zajistit průběžné doručování kvalitních produktů a služeb zákazníkům. DevOps umožňuje dříve izolovaným rolím (vývoj, provoz IT, kontrola kvality a zabezpečení) vzájemnou spolupráci a koordinaci s cílem poskytovat lepší a spolehlivější produkty. Přechodem na kulturu DevOps spolu s využitím nástrojů a postupům týmy získávají schopnost lépe reagovat na potřeby zákazníků, zvýšit důvěru v aplikace, které vytvářejí, a rychleji dosahovat obchodních cílů.*“ [40]

DevOps inženýři vykonávají tyto úkoly pomocí nástrojů k tomu určeným. Existují nástroje pro continuous integration, continuous deployment nástroje a podobně.

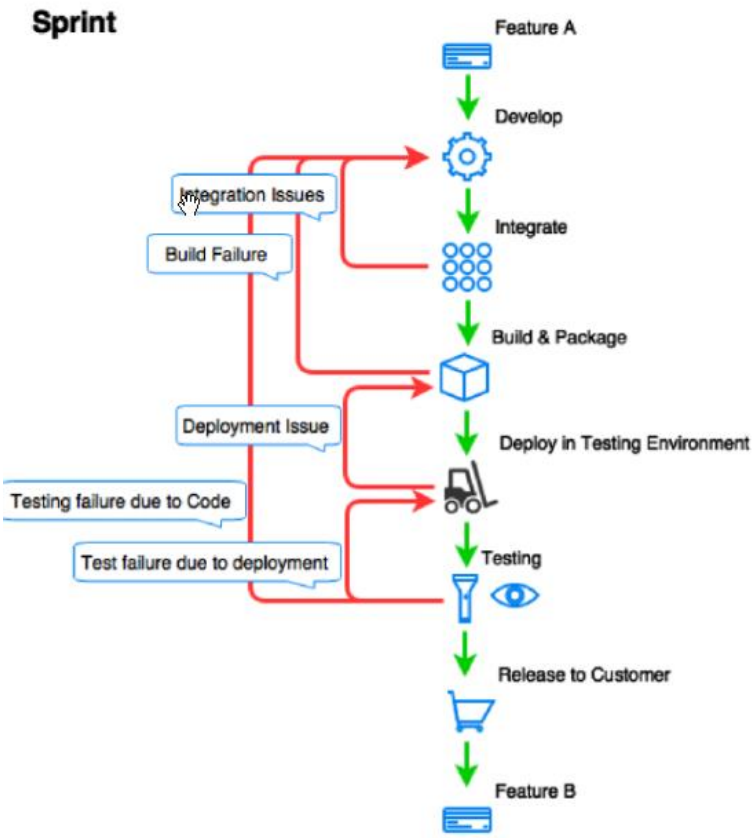
Kontinuální vývoj softwaru je obecný zastřešující pojem, který popisuje několik dalších faktorů agilního vývoje softwaru. Je jedním ze základních postupů pro DevOps. Zahrnuje pojmy continuous integration, continuous delivery, continuous testing a continuous deployment. [41]

6.1 Continuous integration (CI)

Continuous integration je vývojový postup pro který je nutné, aby vývojáři kontinuálně odesílali svůj kód, aby se mohl, alespoň jednou denně, integrovat a následně transformovat to spustitelné formy sestavení (buildu). Continuous integration pak dále zajišťuje, že změny v sestavení (potažmo vytvořený build) je vždy stabilní tím, že využívá automatické (unit, integrační atd.) testy, které se spustí po každém vytvořeném sestavení. [40]

Díky continuous integration problémy, se kterými se během integrace lze běžně setkat, jsou co nejdříve zjištěny. [39] Zapojení CI do projektů minimalizuje riziko, kdy se dvě různé větve od vývojářů nedají sloučit a zároveň zabraňuje vytvoření nestabilního sestavení. [42]

Obrázek 13 představuje příklad implementace continuous integration s jedním CI prostředím.

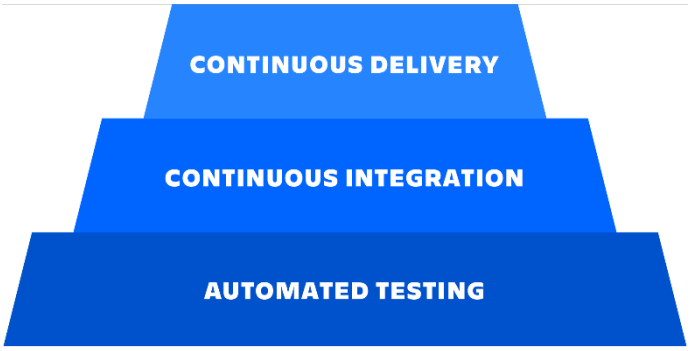


Obrázek 13 - Continuous Integration s jedním prostředím [39]

6.2 Continuous testing

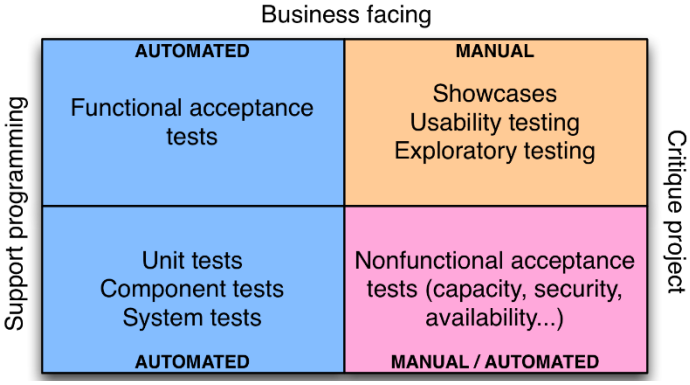
Spustitelné a otestované sestavení (build), které je získáno po continuous integration, je nutné před doručením do hlavního repozitáře (delivery) a nasazením uživatelům na produkční prostředí (deployment) nasadit na prostředí a otestovat ho.

V dnešním DevOps prostředí je nutné tento krok vykonat co nejrychleji. Pokud se testování vykonává pouze manuálně, není možné implementovat continuous delivery (CD). Po každém automatizovaném nasazení nové verze sestavení na testovací prostředí se spouští automatické testy, které ověřují, že nové změny nenarušují stabilitu již naimplementovaných funkcí nebo nezavlékají nové defekty. Dokud kroky continuous integration neprojdou automatickými testy, continuous delivery se nespustí. [43]



Obrázek 14 - Vztah mezi automatizovaným testováním, continuous integration a continuous delivery [43]

Cílem, který se testovací tým snaží zajistit, je provádění mnoha typů testů (automatických i manuálních), které se provádějí průběžně při procesu doručování softwaru, viz Obrázek 15. Pokud má testovací tým hotovou continuous integration a automatické testy může vytvořit CI/CD pipeline. [44]



Obrázek 15 - Typy testů, které se vykonávají v rámci CI [45]

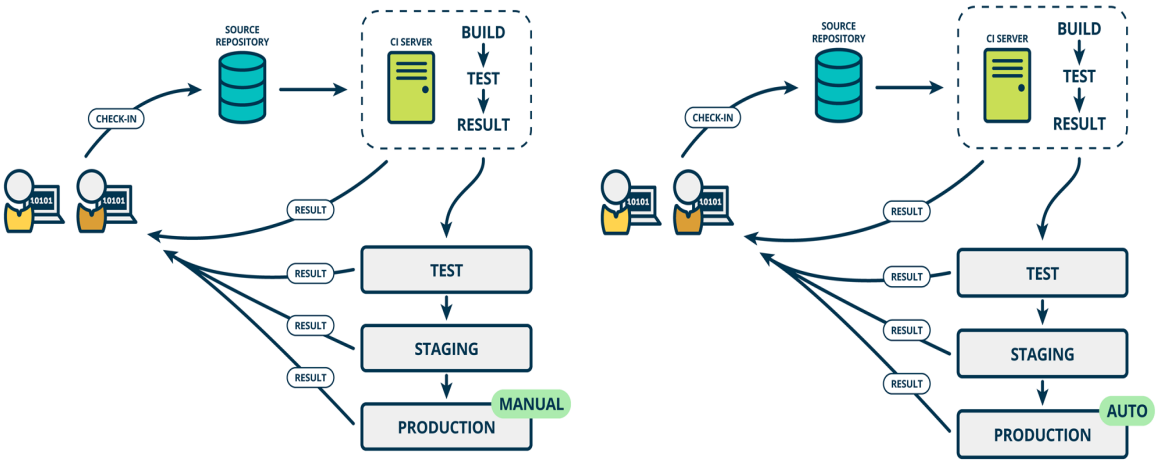
6.3 Continuous delivery a deployment (CD)

Continuous delivery a continuous deployment jsou dva podobné pojmy. Oba tyto pojmy jsou finálním krokem celé CI/CD pipeline, která se skládá z integration, delivery a deployment. [46] [47] Liší se v jejich přístupu k nasazování na produkční prostředí.

Continuous delivery je rozšířením pro continuous integration. Po automatizovaném testování je automatizovaný také proces nasazování úspěšně otestovaných změn kódu do hlavního repozitáře, odkud je lidská pracovní síla musí manuálně nasadit na produkční prostředí. Testovací tým se může rozhodnout, zda se bude do repozitáře nasazovat vícekrát za den, jednou denně či jednou týdně podle požadavků společnosti. [48]

Continuous deployment je rozšířením pro continuous delivery, které dále automatizuje také nasazení na produkční prostředí. Takto automatizovaná CI/CD pipeline nejvíce spoléhá na dobře navrhnuté automatické testy. Díky continuous deployment je možné vývojářovy změny nasadit, v případě, že projdou úspěšně automatickými testy na živé produkční prostředí během několika málo minut. [49]

Na Obrázek 16 je vidět rozdíl mezi continuous delivery a deploymentem. Nasazení na produkční prostředí je buď automatizované (deployment) či manuální (delivery).



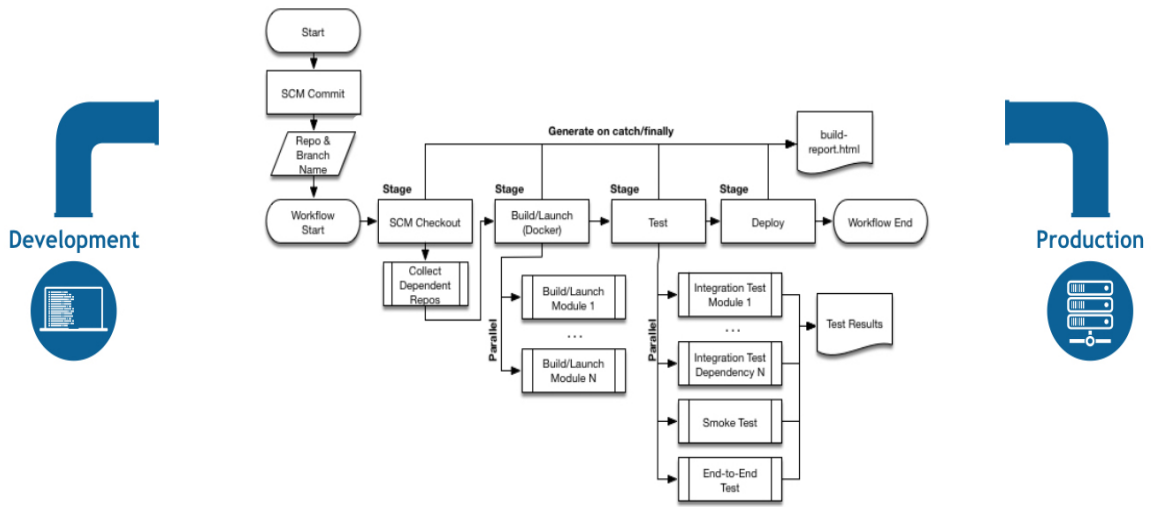
Obrázek 16 - Continuous delivery a continuous deployment [50]

6.4 CI/CD pipeline

Pipeline automatizuje sérii kroků od počátečního komitu vývojáře, přes vytvoření sestavení a continuous integration až po continuous deployment.

„CI/CD pipeline je automatizované vyjádření vašeho procesu pro získání softwaru od správy verzí až po vaše uživatele a zákazníky.“ [51]

Uživatel může vymodelovat CI/CD pipeline na míru projektu do přehledného Pipeline diagramu, který tento model představuje. Diagram se může skládat z více fází, které se skládají z jednoho nebo více kroků. Typickými fázemi CI/CD pipeline jsou fáze "Build", "Test" a "Deploy". Součástí celého Pipeline jsou také uzly, které vykonávají jednotlivé fáze a kroky. [51]



Obrázek 17 - CI/CD pipeline vymodelovaná v Jenkins Pipeline [51]

Chyby, které mohou nastat při častém nasazování nového či upraveného kódu, jsou díky automatizaci CI/CD eliminovány. Zároveň se šetří čas díky rychlejšímu nasazování a menší režii při provádění rutinních kroků. Tím, že dochází k častějšímu nasazování s menšími změnami v kódu, si jsou vývojáři při jeho spouštění více jistí. [40]

6.5 Metodika vytvoření sady automatických testů a jejich integrace s nástroji CI/CD a test managementu

Společnost Unicorn v roce 2020 využívá mnoho rozličných testovacích nástrojů a nástrojů podporujících testování.

Mimo jiných jsou těmito nástroji různé:

- Nástroje pro automatizaci testů;
- nástroje pro podporu průběžného vývoje;
- nástroje pro test management.

Z každé této kategorie nástrojů jsou vybrány dva až tři nástroje. Pomocí vybraných nástrojů pro automatizaci testů jsou v dalších kapitolách vytvořeny automatizované testy různých druhů.

Při vytváření automatických testů je pro každý druh testu vybrán nástroj, který podle zkušeností autora práce nejlépe vyhovuje pro vytvoření daného druhu testu a zároveň je v nějaké verzi využíván v rámci společnosti Unicorn.

Dále je předvedena ukázka integrace automatického testu se dvěma vybranými CI/CD nástroji a dvěma vybranými test management nástroji.

7 Automatizace testů

V této kapitole je popsán detailní manuál pro vytvoření různých druhů automatických testů. Pomocí dvou nástrojů na vytváření automatických testů je v následujících kapitolách vytvořen:

- Request-based automatický test, který může sloužit jako rychlý smoke test;
- selenium-based automatický test, který může sloužit jako regresní test;
- automatický test s příkazem pro OS, který může sloužit jako regresní test;
- automatický test úpravou databáze, který může sloužit jako regresní test;
- automatický výkonnostní test, který může sloužit jako regresní test pro prověření nefunkčních požadavků na software.

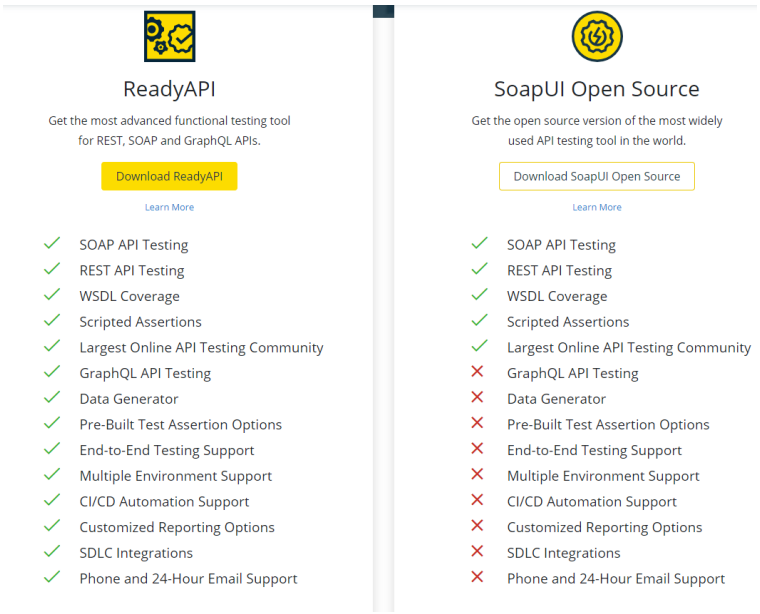
Nástroje jsou vybrány na základě jejich skutečného použití či možnosti jejich využití ve společnosti Unicorn na různých typech projektů. Liší se nejen účelem jejich používání, ale také cenou, jelikož některé vybrané nástroje jsou k dispozici ve verzi open-source a u jiných je třeba si zaplatit komerční licenci.

Request-based automatické testy jsou vytvořeny nástroji ReadyAPI, SoapUI a JMeter. První dva nástroje jsou primárně využívány pro funkční testování pomocí API, JMeter je v testerské praxi využíván převážně pro nefunkční testování jako je load testování a jiné druhy performance testování.

7.1 Request-based test

Automatický regresní request-based test je vytvořen pomocí nástrojů SoapUI ve verzi 5.6.0 a ReadyAPI ve verzi 3.4.0. Oba tyto nástroje jsou vyvíjeny společností SmartBear Software a jsou využívány pro vytváření funkčních a nefunkčních automatických testů. Nejvýraznějším rozdílem mezi oběma nástroji je, kromě množství podporovaných funkcionalit, které ukazuje Obrázek 18, že SoapUI existuje také ve open source verzi, zatímco ReadyAPI nabízí pouze časově omezenou trial verzi, kterou je později nutné přeinstalovat na plnou placenou verzi viz Obrázek 18.

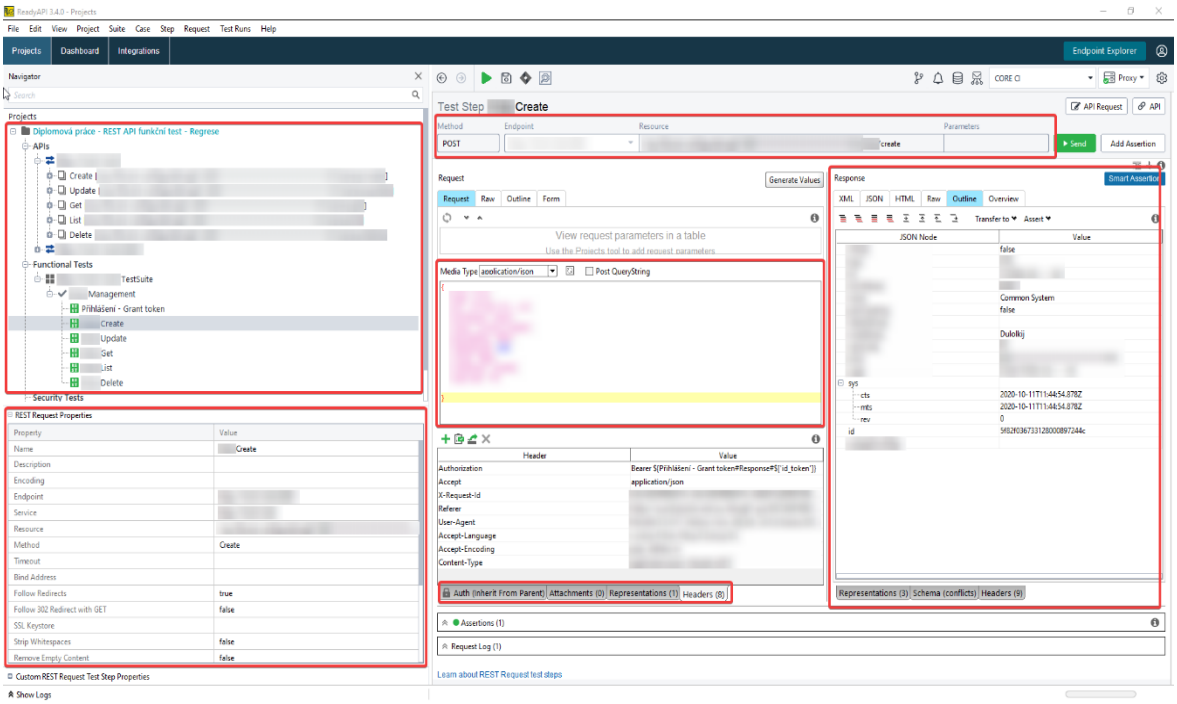
Po nainstalování nástrojů se, na úložiště počítače, ve kterém je nástroj spuštěn, vytvoří základní soubory pro workspace (xml dokument, ve kterém je definován projekt či projekty, které si uživatel vytvoří a dá se mezi nimi poté přepínat) a výchozí nastavení.



Obrázek 18 – Pricing strategie SoapUI a ReadyAPI [52]

V obou nástrojích se vytváření testu začíná vytvořením projektu, nastavením výchozích parametrů endpointů a nastavením výchozích parametrů requestů v levém dolním rohu uživatelského rozhraní obou nástrojů. V SoapUI se žádný výchozí projekt nevytváří. Oba nástroje umožňují vytvořit REST či SOAP projekt, ve kterých se následně mohou vytvořit celé testy. REST a SOAP requesty se dají kombinovat.

7.1.1 ReadyAPI



Obrázek 19 - UI ReadyAPI, Zdroj: Vlastní zpracování

Uživatelské rozhraní ReadyAPI se dělí na:

- Layout testu umístěného vlevo nahoře, který se skládá z definice API v horní části a samotného funkčního testu v hierarchii projektu – testovací sada (test suite) – případ užití (test case) – krok testu (test step);
- přehled vlastností každého jednotlivého requestu vlevo dole;
- nastavení metody, URL endpointu, proti kterému je request poslán, resource (vybraná metoda) a další parametry;
- vlastní tělo requestu s atributy uprostřed UI, které se s requestem posílají;
- nastavení autentizace, možnosti připojit přílohu k requestu či nastavení hlavičky requestu;
- response okno vpravo UI, které se dá zobrazit v různých formátech.

Autentizace

ReadyAPI poskytuje více metod autentizace od základních přes různé autorizační protokoly. Podporuje základní (Basic či Digest) metodu autentizace a protokol NTLM pro přihlášení v sítích se systémy s OS Windows. Dále protokoly OAuth 1.0, OAuth 2.0, OAuth 2.0 Azure, což jsou metody užívané pro přihlášení přes http protokol, či AWS Signature protokol pro autentizaci pro posílání requestů na Amazon Web Services. [53]

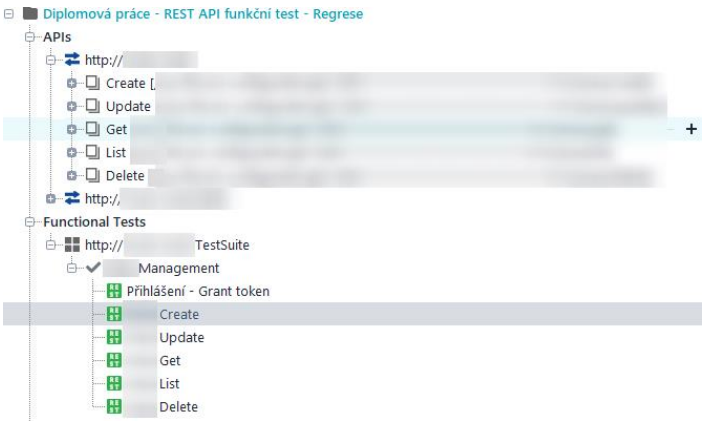
Ve vytvořeném funkčním testu viz Obrázek 19, nejsou využity integrované metody přihlašování pomocí předávání parametrů. V rámci test stepů je z response requestu pro přihlašování extrahován přihlašovací token a ten je následně vložen do hlaviček dalších test stepů.

Vytvoření funkčního testu

Pro vytvoření funkčního testu je nutné si nejdříve nadefinovat API, které se bude testovat. Definovat API lze více způsoby, jednodušším způsobem je import WSDL (Web Service Definition Language) či WADL (Web Application Description Language), což jsou xml soubory popisující metody, jakož i vstupy a výstupy, které nabízí webová služba. WSDL popisuje SOAP API webové služby a WADL popisuje REST webové služby. [54] [55] O něco složitější způsob definování API je ručně popsat všechny jeho parametry ručně, včetně URL, kde se nachází.

Když jsou naimportovány či sepsány definice všech metod, které webová služba poskytuje, můžou se na jejich základě vytvořit jednotlivé případy užití s requesty, které jsou k jeho otestování potřeba (test stepy) doplněním definice o tělo s atributy či jiné parametry.

Definice jednotlivých metod jsou poté k dispozici pod záložkou „APIs“ v layoutu testu a jednotlivé test stepy se nachází pod záložkou „Functional Tests“ viz Obrázek 20.



Obrázek 20 - Layout testu v ReadyAPI, Zdroj: Vlastní zpracování

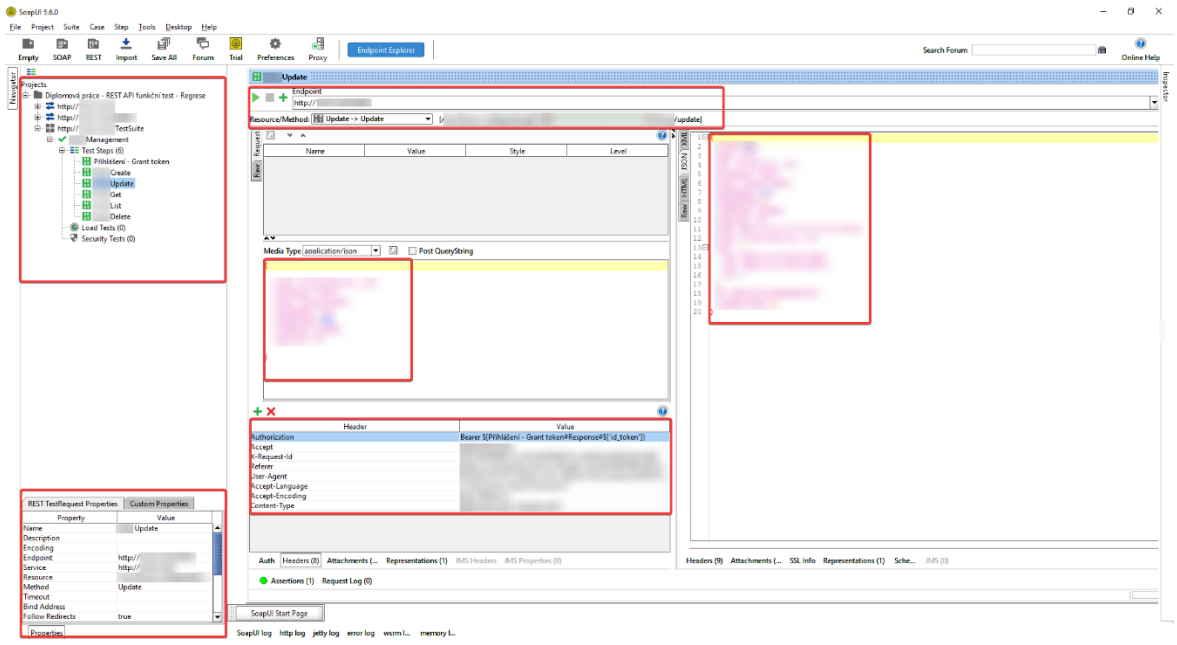
Aserce

Důležitou praktikou je u každého test stepu vytvořit aserci na nějaký aspekt či zdroj pro kontrolu správnosti jeho vykonání. V ReadyAPI lze vytvořit aserci na jakýkoli zdroj requestu test stepu, nejčastěji se však vytváří aserce na response daného test stepu. Aserce je nastavena na přítomnost atributu, počet, hodnotu určitého atributu a další zdroje. Aserce může vyhledávat určité hodnoty také pomocí regulárních výrazů.

7.1.2 SoapUI

Nástroje SoapUI a ReadyAPI jsou mezi sebou kompatibilní. Pokud se vytvoří projekt v jednom nástroji, může se otevřít a použít také v druhém.

Uživatelské rozhraní je s malými odchylkami podobné ReadyAPI s jedním větším rozdílem, tím je otevírání nového okna pro každé přepnutí mezi nastavením requestu.



Obrázek 21 - UI SoapUI, Zdroj: Vlastní zpracování

SoapUI uživatelské rozhraní se dělí na stejné části jako v případě ReadyAPI:

- Layout vlevo nahoře, rozdělený na API nahoře a funkčního testu v hierarchii projekt – testovací sada (test suite) – případ užití (test case) – krok testu (test step);
- parametry requestu vlevo dole;
- nastavení metody requestu, URL endpointu, samotná testovaná metoda;
- tělo requestu s atributy;
- nastavení autentizace, příloha k requestu a nastavení hlavičky requestu;
- response okno.

Autentizace

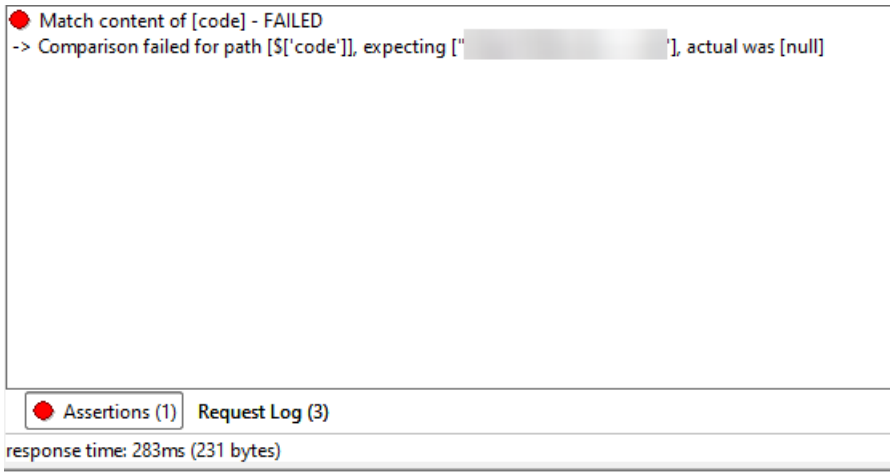
Autentizace je řešena stejným způsobem jako v ReadyAPI. Pomocí předání parametru token z response requestu pro přihlášení do hlaviček dalších requestů.

Vytvoření funkčního testu

Vytvoření testu probíhá přes import definice metod webové služby, přes WSDL nebo WADL či jejich manuálním sepsáním. Poté se mohou vytvořit jednotlivé test stepy doplněním definic metod o atributy, které se budou posílat v těle requestu.

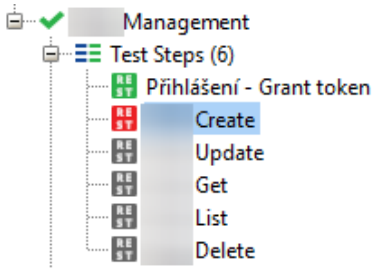
Aserce

Aserce je možné přidat na spodní straně okna pro úpravu requestu, viz Obrázek 22.



Obrázek 22 - Spadlá aserce v SoapUI, Zdroj: Vlastní zpracování

Kontrolovat se může existence atributu, hodnota v atributu či další aspekty. Pokud aserce zjistí rozpor mezi očekávanou a zjištěnou hodnotou, request se zabarví červeně, viz Obrázek 23. V případě, že se nepřidá k requestu aserce, je request červený, jen pokud se nepodaří získat adekvátní response.



Obrázek 23 - Spadlý request v SoapUI,
Zdroj: Vlastní zpracování

7.2 Selenium-based test

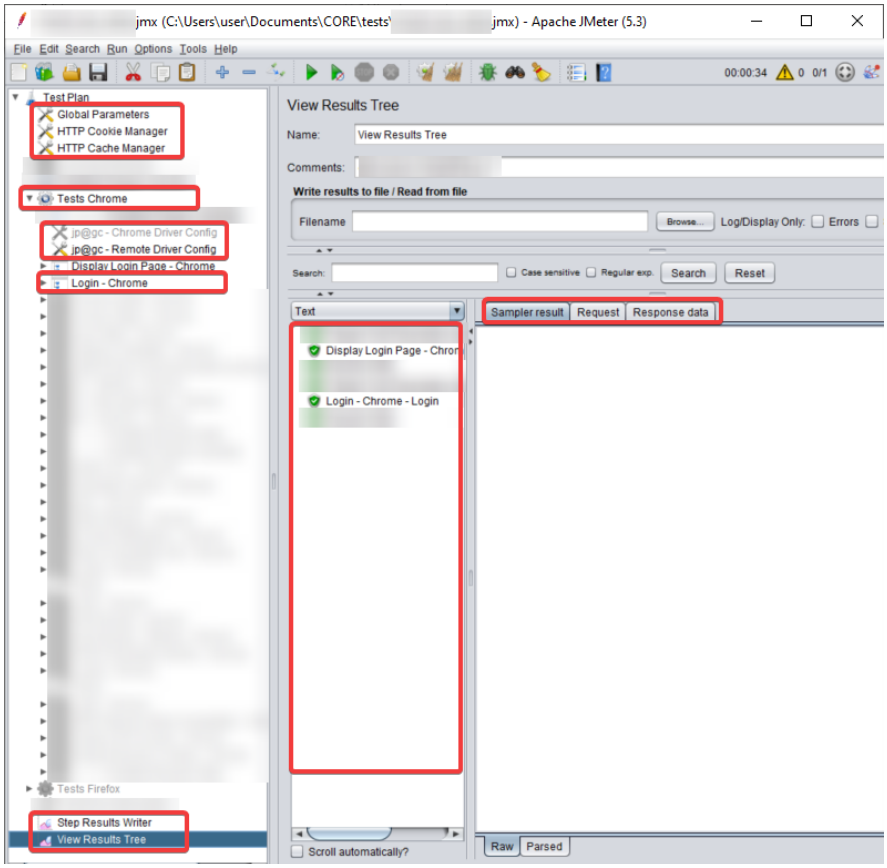
Automatický Selenium-based test prověřuje software z hlediska jeho uživatelského rozhraní. Využit je Chrome WebDriver (lze použít i WebDrivery pro ovládání jiných druhů prohlížečů), což je v podstatě knihovna Selenia, která obsahuje logiku všech dostupných metod (klikání, vepisování atd.). Tester, který vytváří Selenium WebDriver test, tedy může využít abstrakci WebDriveru a použít zjednodušené ovládání prohlížeče pro účely vytvoření GUI testu. Příklad nástroje, který umožňuje integrovat Selenium pro vytváření a spouštění Selenium-based testů ovládaných pomocí WebDriveru, je Apache JMeter.

Použit je JMeter ve verzi 5.3 a Chrome WebDriver pro Google Chrome prohlížeč ve verzi 86.0.4240.75. Oba tyto nástroje WebDriver i JMeter jsou k dispozici v open source formě.

JMeter je ve využíván pro funkční testování API, GUI testování i performance testování. JMeter není nutné instalovat, stačí jej stáhnout v zazipovaném souboru, rozbalit jej a spustit. WebDriver je po stažení nutné vložit do bin složky JMeteru.

Po spuštění nástroje JMeter se automaticky vytvoří test plan (projekt) do kterého je nejdříve nutné vložit konfigurační elementy pro práci s hlavičkami requestů a cache, vlákno (thread group, v testu „Tests Chrome“), ve kterém GUI test poběží, WebDriver konfigurační elementy pro Chrome Driver a Remote Driver a listenery (v testu „View Results Tree“, které umožňují zobrazení výsledků běhu jednotlivých GUI testů). Všechny tyto elementy zobrazuje Obrázek 24.

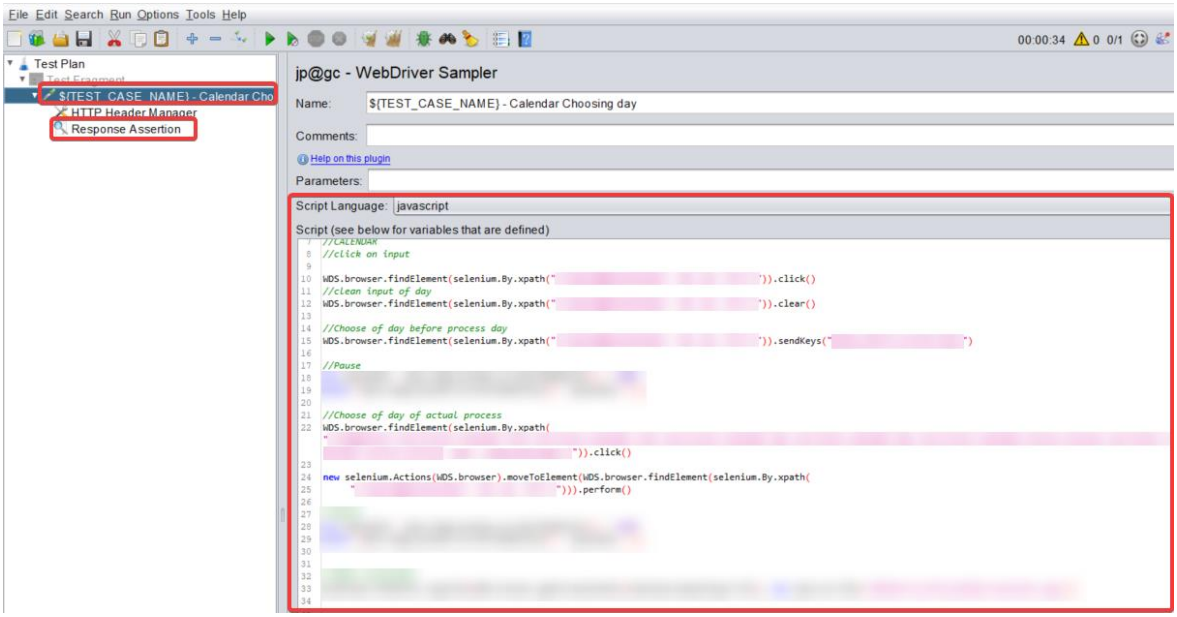
Obrázek 24 ukazuje také způsob prohlížení výsledků jednotlivých test stepů GUI testu uprostřed UI. Response každého test stepu se může prohlížet po kliknutí na vybraný test step a vybrání formátu, ve kterém se response zobrazí. Poté kliknutí na response záložku vpravo nahoře v UI, viz Obrázek 24. Na stejném místě je umístěna záložka pro prohlížení endpoint URL a metody s jejími atributy, které se společně s requestem poslaly a záložka „Sampler result“ s elementárními údaji o průběhu procesu poslání requestu a přijmutí response.



Obrázek 24 - UI JMeteru s GUI testem, Zdroj: Vlastní zpracování

Uvnitř skupiny vláken (Thread Group), ve kterém se nastaví požadovaný počet spuštěných vláken, se přidají samplery a logické ovladače (představují jednotlivé funkční či logické bloky), ze kterých se následně test skládá. Samplery představují v testu různé druhy requestů či dalších funkčních bloků, které podporují práci s e-mailem, LDAP, OS příkazy či samplery pro práci s parametry testu, které se dají v rámci jednotlivých test stepů lokálně přepisovat. V GUI testu je tím nejdůležitějším samplerem WebDriver sampler, který obsahuje veškerou logiku Selenium-based testu, viz Obrázek 25. Nejpoužívanějšími metodami Selenium WebDriveru jsou vyhledání elementu webové stránky přes jeho id či pomocí jazyka xpath, klik, hover, vepsání textu do elementu a další základní uživatelské akce při ovládání webového prohlížeče.

Stejně jako v ReadyAPI a SoapUI se můžou přidat aserce do jednotlivých test stepů, například pod WebDriver sampler „Calendar Choosing day“ můžeme přidat element assertion určitého druhu, viz Obrázek 25. Pod WebDriver samplerem se v response objevuje zdrojový kód webové stránky, nejvhodnějšími druhy aserce jsou tedy kontrola existence textu či xpath aserce (xpath je jazyk, pomocí kterého se můžou vybírat jednotlivé elementy z DOM webové stránky).



Obrázek 25 - WebDriver sampler v JMeteru, Zdroj: Vlastní zpracování

V případě výběru Chrome WebDriveru se po spuštění testu inicializuje kompletně automaticky ovládaný Chrome prohlížeč. Pokud se vybere použití Remote WebDriveru (ve kterém se může vybrat druh prohlížeče, pomocí kterého bude test vykonán), test je vykonán přes Selenium Grid na vzdáleném serveru a lokálně se žádný prohlížeč neinicializuje.

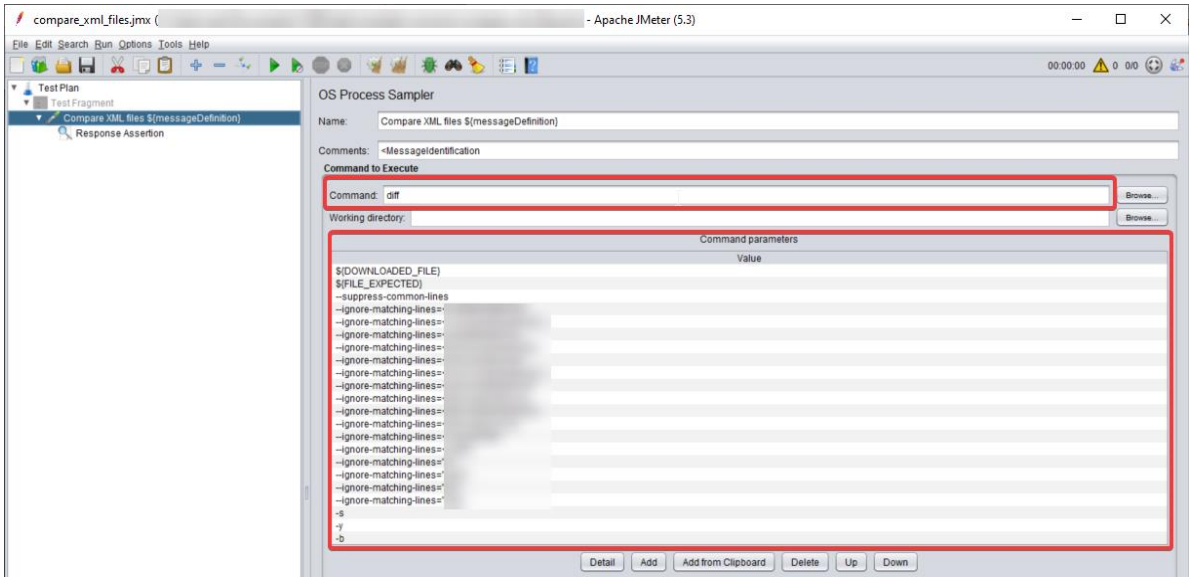
7.3 Test s příkazem pro OS

Testy s příkazem pro lokální operační systém se mohou využít jako pomoc při funkčním testování.

V případě nutnosti práce s velmi velkými daty, můžou testy s příkazem operačnímu systému pro přidání a odebrání souborů z a do archivu v rámci regresních či jiných funkčních testů pomoci, aby se data vešla na sdílené limitované úložiště. Například do GitHub úložiště.

V případě nutnosti porovnávat data generovaná softwarem s očekávanými daty se může využít test s OS příkazem pro porovnání souborů. Pokud je třeba kontrolovat v GUI testech obrazovky, existuje příkaz pro OS, který porovnává dva obrázky a vrací obrázek se zvýrazněnými změnami.

Nástrojem, který umožňuje využít příkazy lokálnímu operačnímu systému v rámci testu, je JMeter. Do vlákna se přidá sampler „OS Process Sampler“, viz Obrázek 26. [30]



Obrázek 26 - Diff příkaz pro OS v JMeter OS Process Sampleru, Zdroj: Vlastní zpracování

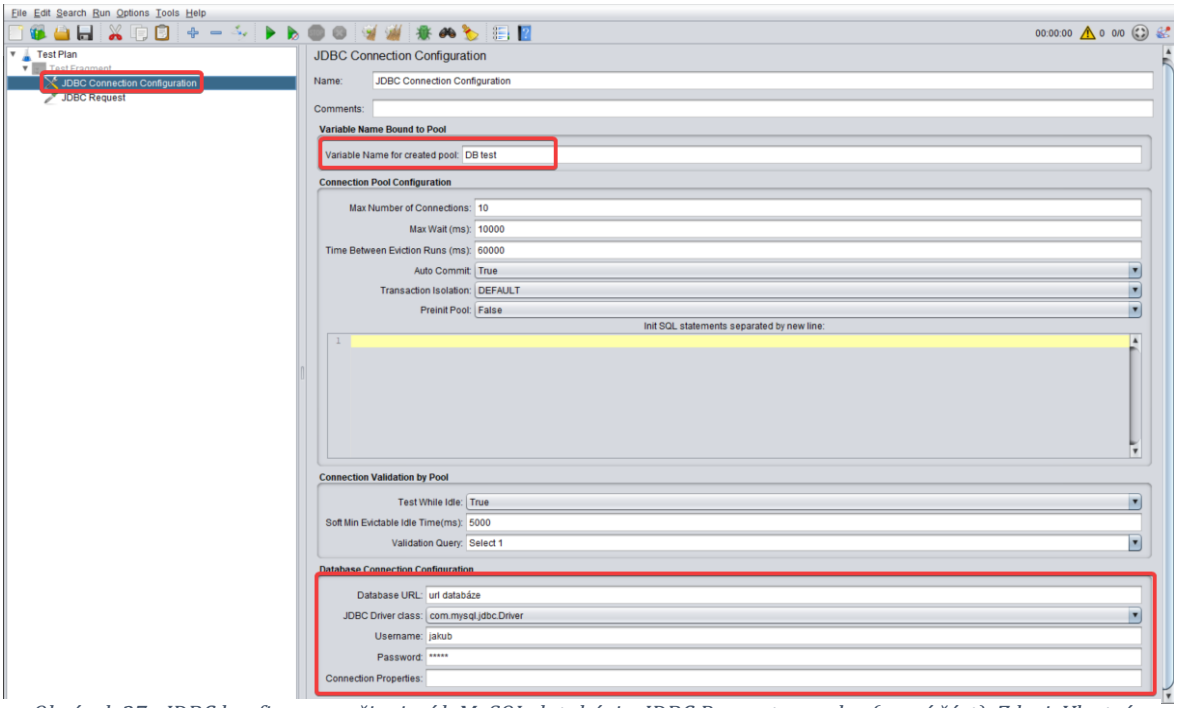
Do kolonky „command“ se vloží příkaz pro OS a pod něj případnou pracovní složku. Do okna „command parameters“ se přidají další nutné parametry příkazu (pokud existují). [30]

7.4 Test s úpravou databáze

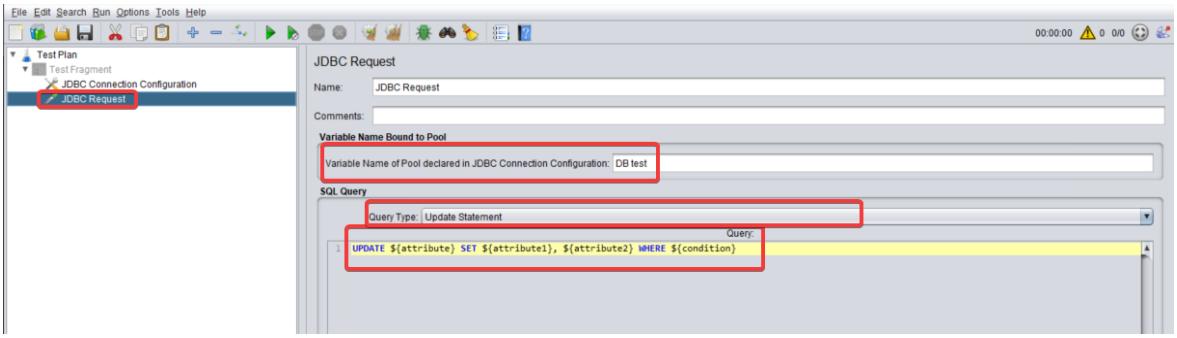
Pro vytvoření testu s úpravou MySQL databáze se nejdříve přidá vlákno (Thread Group), do něj se umístí konfigurační element „JDBC Connection Configuration “, který obsahuje všechny potřebné detaily pro inicializaci připojení mezi klientem a MySQL databází přes JDBC API. Mezi nutné parametry patří jméno poolu pro pozdější propojení konfigurace s JDBC request samplerem, URL databáze, uživatelské jméno, heslo a JDBC Driver class, viz Obrázek 27 (vrchní část). Pro práci s jinými databázemi jako je PostgreSQL postačí stáhnout příslušný driver class, opět ho přidat mezi ostatní knihovny ve složce lib a vybrat driver v konfiguračním elementu „JDBC Connection Configuration “. [57]

Tato knihovna se musí stáhnout ve formátu Java archive spustitelném souboru (.jar) a přidat do složky lib v kořenovém adresáři JMeteru. [56]

Poté se již může do vlákna přidat sampler „JDBC Request“, ve kterém se bude nacházet SQL dotaz. Tento sampler se také musí propojit s konfiguračním elementem pomocí stejného jména poolu, viz Obrázek 28. [57]



Obrázek 27 - JDBC konfigurace připojení k MySQL databázi a JDBC Request sampler (první část), Zdroj: Vlastní zpracování



Obrázek 28 - JDBC konfigurace připojení k MySQL databázi a JDBC Request sampler (druhá část), Zdroj: Vlastní zpracování

7.5 Performance test

Existuje více druhů performance testů, které mohou být implementovány a poté vykonávány pro ověření nefunkčních parametrů softwaru. Konkrétně load testy mohou být naimplementovány ve všech výše zmíněných nástrojích. V ReadyAPI a SoapUI existuje možnost transformovat již naimplementovaný funkční test v load test.

JMeter je však pro tento druh testování a vytváření performance testů používanější a podle autora práce vhodnější, jelikož umožňuje při load testování kooperaci s lokálním operačním systémem, snadnější práci s databázemi a nespornou výhodou je možnost simulovat reálné chování uživatelů při práci se softwarem nahráváním uživatelských akcí přímo z webového prohlížeče za pomoci Selenia. Akcemi mohou být klikání, scrollování myší, vepisování textu a další akce. Při nahrávání se však zaznamenává veškerý webový provoz, proto je nutné vybrat pouze ty akce a elementy s requesty a nastavením, které jsou pro load test důležité.

Před započetím vytváření load testu je důležité se ujistit, že stroj, na kterém bude test běžet, má [58]:

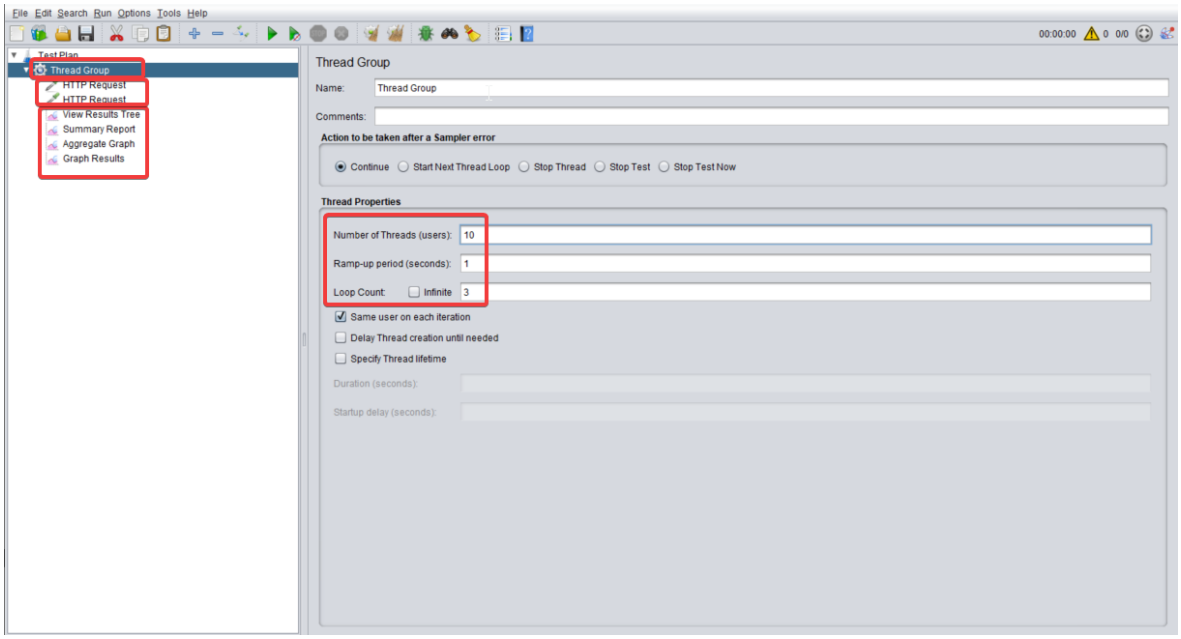
- *„Správné vlastnosti z hlediska CPU, paměti a sítě;*
- *nainstalovanou nejnovější verzi Javy podporované Jmeter;*
- *zvětšenou velikost Java haldy (ve výchozím nastavení běží JMeter s 1 GB Java haldou, to nemusí pro test stačit a záleží na testovacím plánu a počtu vláken, které chcete spustit).“*

Důležité je: *„Nepouštět load test přes GUI mód JMeteru“* [58] Jelikož tento způsob spuštění testu může zkreslit výsledky.

Na konci load testu spuštěného přes CLI mód (přes příkazový řádek) může JMeter vygenerovat HTML report, nicméně ve výchozím nastavení JMeter poskytuje výsledky již během testu. [58]

Load test může být vytvořen jako reálná simulace uživatelské zátěže při práci v softwaru jako Selenium-based test nebo může být složen z API http requestů jako request-based test.

Do elementu „skupina vláken“ (Thread Group) se nastaví počet požadovaných spuštěných vláken, počet cyklů testu a také náběhový čas testu. V rámci skupiny vláken se poté přidá logika load testu, představující všechny Selenium WebDriver skripty či všechny http requesty a logické či konfigureační elementy testu. Na konci je možné přidat listener, který zobrazuje výsledky testu v rámci GUI ve formě, jakou si zvolíme či jaká je pro daný test nejvhodnější. Vybrat si můžeme výpis výsledků ve formě grafu, tabulky či seznamu. Listenery u load testu však poslouží pouze pro kontrolu funkčnosti testu, jelikož při reálném spouštění nejsou potřeba. Všechny elementy zmíněné výše, viz Obrázek 29.



Obrázek 29 - Elementy load testu v JMeteru, Zdroj: Vlastní zpracování

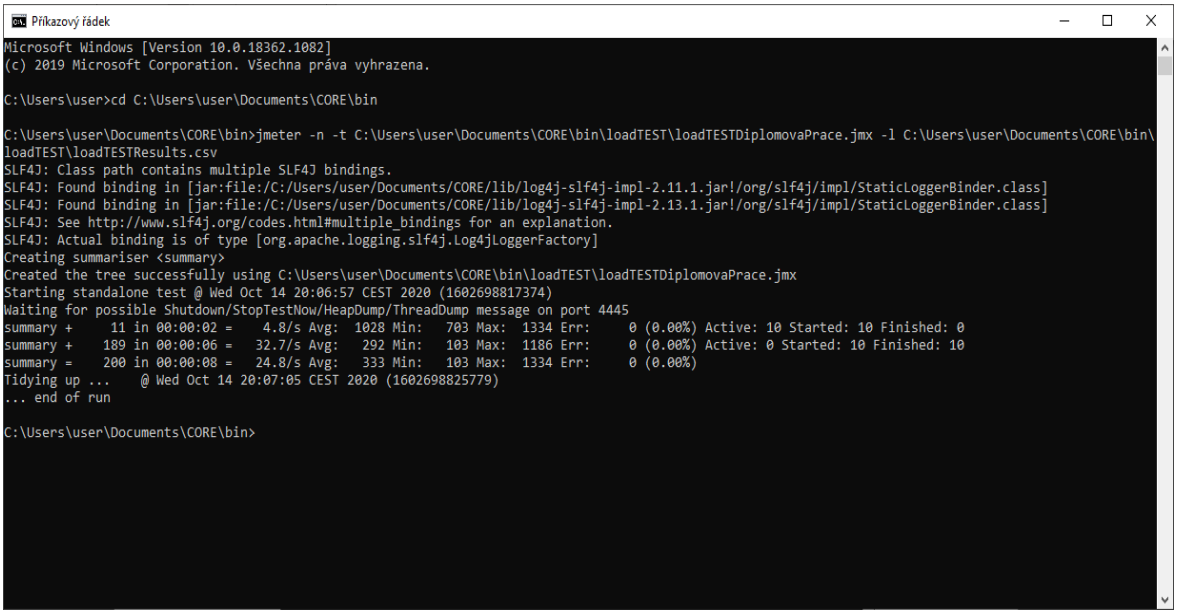
Pro spuštění load testu se využije, CLI (Command Line Interface) mód JMeteru. V příkazovém okně se musí nejdříve přesunout do složky bin v kořenovém adresáři

JMeteru (ve Windows příkazem cd a lokací k přemístění) odkud je možné spouštět testy. Poté je test spuštěn příkazem [58]:

„jmeter –n –t [lokace test skriptu] –l [lokace souboru s výsledky testu]"

-n znamená non-GUI mód; -t slouží k přidružení lokace test skriptu; -l ukazuje na umístění souboru s výsledky (většinou soubor.jtl) [58]

Příkaz může v praxi tedy vypadat takto:



Obrázek 30 - Spuštění load testu z příkazového řádku ve Windows, Zdroj: Vlastní zpracování

Pokud bude třeba vygenerovat report, který v grafické podobě ukáže všechny výsledky testu, je možnost využít dalšího příkazu, který je podobný, avšak liší se dvěma přidanými parametry –e; –o [58]:

„jmeter –n –t [lokace test skriptu] –l [lokace souboru s výsledky testu]" –e –o [lokace složky, kam bude uložen html report]"

–e po testu vygenerovat informační html report; –o určuje umístění složky, kam bude uložen html report. [58]

8 Integrace automatických testů s CI/CD nástroji

Integrace automatických testů s CI/CD nástroji znamená zařadit automatizované testování do procesu kontinuálního vývoje. Po fázi kontinuální integrace a kontinuálního nasazování je na řadě fáze kontinuálního testování, které zajišťují CI/CD nástroje, se kterými jsou integrovány nástroje pro vytváření a spouštění automatických testů.

Sestavení, které je po continuous integration k dispozici, se automaticky otestuje a poté se nasadí do repozitáře a uživatelům na produkční prostředí (v případě zavedeného kontinuálního nasazování).

Na trhu existuje mnoho open-source a komerčních nástrojů. Liší se od sebe uživatelskou přívětivostí, vestavěnými funkčnostmi, platformovou kompatibilitou, možnostmi hostingu a komerční nástroje také svou cenovou strategií.

	Jenkins	circleci	TeamCity	Bamboo	GitLab
Open source	Yes	No	No	No	No
Ease of use & setup	Medium	Medium	Medium	Medium	Medium
Built-in features	3/5	4/5	4/5	4/5	4/5
Integration	★ ★ ★ ★ ★	★ ★ ★	★ ★ ★ ★	★ ★ ★	★ ★ ★ ★
Hosting	On premise & Cloud	On premise & Cloud	On premise	On premise & Bitbucket as Cloud	On premise & Cloud
Free version	Yes	Yes	Yes	Yes	Yes
Build Agent License Pricing	Free	From $39 per month	From $299 one-off payment	From $10 one-off payment	From $4 per month per user
Supported OSs	Windows, Linux, macOS, Unix-like OS	Linux or MacOS	Windows, Linux, macOS, Solaris, FreeBSD and more	Windows, Linux, macOS, Solaris	Linux distributions: Ubuntu, Debian, CentOS, Oracle Linux

Obrázek 31 - Nejlepších 5 CI/CD nástrojů v roce 2020 [59]

V této diplomové práci jsou uvažovány pouze dva CI/CD nástroje, těmito nástroji jsou TeamCity a Jenkins, jelikož jsou oba tyto nástroje široce využívány a pokrývají nabídku CI/CD nástrojů na trhu z pohledu komerčnosti. TeamCity je nabízen jako komerční nástroj a Jenkins je od počátku implementován jako open-source nástroj.

8.1 TeamCity

Nejdříve je nutné TeamCity nainstalovat a vytvořit hlavní projekt. Hierarchie projektů v TeamCity začíná hlavním projektem, který se dále dělí na dílčí fáze (subprojekty), které vykonávají určité kroky sestavení. Každá fáze (subprojekt) poskytuje řadu parametrů k nastavení, které jsou rozdělené do několika sekcí.

Při sestavování pipeline pro hlavní projekt se vytvoří fáze (subprojekty) pro automatizované kontinuální testování. Tyto fáze v sobě obsahují jednotlivé kroky sestavení, které integrují nástroj pro spouštění automatických testů, který v rámci běhu sestavení vykonává určené automatické testy. U fází (subprojektů) zaměřených na spouštění automatických testů se, stejně jako u všech ostatních subprojektů, nastavují:

- General Settings (Obecné nastavení);
- version control settings (nastavení správy verzí);
- build steps (kroky sestavení);
- triggers (spouštěče);
- failure conditions (podmínky pro selhání sestavení);
- build features (funkce sestavení);
- dependencies (závislosti);
- parameters (parametry);
- agent requirements.

General Settings

Nejdůležitějším parametrem v subprojektu pro kontinuální testování jsou cesty artefaktů v sekci general settings (obecné nastavení). Artefakt je jakýkoliv soubor či jiný element vytvořený sestavením (buildem), který je uchován na TeamCity serveru (mohou to být soubory logů, srovnávacích obrázků či sobory generované testovaným softwarem). Parametr cesty artefaktů určuje, kde se mají ony vytvořené soubory v repozitáři spouštěcího agenta hledat a kde či jakým způsobem by se měly na TeamCity serveru uložit.

Version Control Settings

V sekci version control settings (nastavení správy verzí) se nastavuje, odkud a jakým způsobem se bude načítat zdrojový kód ze systému pro správu verzí softwaru (VCS – Version control system). Nastavena je URL repozitáře (například v GitHub) a další parametry, tím je vytvořeno spojení, odkud se načítají změny v testovací master větvi pravidelně v daném časovém intervalu. [60]

Build Steps

Stejně jako u subprojektu na nasazování aplikace se u subprojektu pro testování nastavují build steps (kroky sestavení), které se vykonávají sériově za sebou jeden po druhém. Kroků může být tolik, kolik jich daný subprojekt potřebuje. [61]

Prvním krokem v případě integrace Apache JMeteru, pro vykonávání automatických testů, je jeho čistá instalace. Například pomocí Apache Maven z repozitáře pomocí pom souboru, což je: „*Základní jednotka práce v Maven. Jedná se o soubor XML, který obsahuje informace o projektu a podrobnosti o konfiguraci, které Maven používá k sestavení projektu*“. [62]

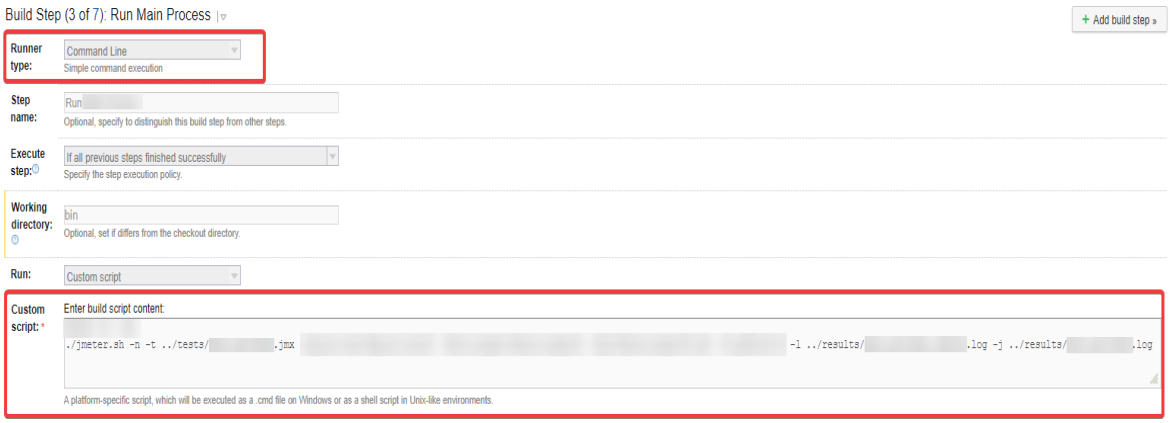
Dalším krokem je spuštění určitého automatického testu přes CLI mód příkazem pro nástroj pro vytváření a spouštění automatických testů, příkladem pomocí nástroje Apache JMeter.

Příkaz z předchozí kapitoly této práce vypadá takto [58]:

„jmeter –n –t [lokace test skriptu] –l [lokace souboru s výsledky testu]" -j [lokace logu průběhu vykonávání testu]

-n znamená non-GUI mód; -t slouží k přidružení lokace test skriptu; -l ukazuje na umístění souboru s výsledky (většinou soubor.jtl); -j ukazuje lokaci logu průběhu vykonávání testu [58]

Tento příkaz, po spuštění sestavení, spustí určitý test, uloží výsledky testu do formy logu v jakémkoliv zvoleném formátu a uloží log z průběhu vykonávání testu v jakémkoliv zvoleném formátu, viz Obrázek 32.



Obrázek 32 – Shell příkaz v kroku sestavení ke spuštění testu přes JMeter v TeamCity, Zdroj: Vlastní zpracování

Triggers

Poté co je vytvořena konfigurace sestavení se může spustit manuálně či pomocí triggers (spouštěčů).

Spouštěč sestavení je pravidlo, které iniciuje nové sestavení při určitých splněných událostech (jako je třeba úspěšně vykonané předchozí sestavení) nebo pravidelně s určitým konfigurovatelným intervalem (například každé 4 hodiny). Sestavení se vloží do fronty a počká na agenta, který by ho mohl spustit. [63]

Failure Conditions

V sekci failure conditions (podmínky pro selhání sestavení) jsou podmínky, které určují, že spuštěné sestavení je označeno jako selhané a zastaví se.

Běžnými podmínkami pro určení selhaného sestavení jsou tyto situace [64]:

- Sestavení se vytváří déle, než je nastavená povolená hodnota v minutách: výchozím nastavením je 0, což znamená žádný limit;
- proces spouštění sestavení končí s jiným kódem než 0;
- alespoň jeden z testů selže: pro subprojekt kontinuálního testování je tato podmínka dobře využitelná;
- spouštěč sestavení zaznamená chybovou hlášku;
- je zjištěn nedostatek paměti nebo selhání (pouze Java): Zaškrtnutím této možnosti označíte sestavení jako neúspěšné, pokud je zjištěn pád JVM, nebo Java má problémy s pamětí.

Build Features

V sekci build features (funkce sestavení) se přidávají funkce, kterými může dané sestavení disponovat. „Funkce sestavení“ je část funkce, kterou lze přidat do konfigurace sestavení, aby ovlivnila běžící sestavení nebo vykazování výsledků sestavení“. [65]

Jednou z těchto funkcí sestavení může být užitečné procesování XML report souborů, které byly generovány v externích nástrojích. [66]

Dependencies

Sekce nastavení dependencies (závislostí) nastavuje závislosti konfigurací sestavení jeden na druhém. Mohou být dva druhy závislostí konfigurací buildů [67]:

- Snapshot dependency: tato závislost velice úzce spojuje dvě sestavení, jedno sestavení je logickou součástí druhého sestavení;
- artifact dependency: artefakty vytvořené v rámci jednoho sestavení se objeví v druhém sestavení.

Parameters

Sekce parameters (parametry) je prostředkem pro sdílení nastavení a pohodlné předávání nastavení do sestavení. [68]

Existuj tři druhy parametrů, které se předávají do sestavení [68]:

- Parametry prostředí;
- systémové parametry;
- konfigurační parametry.

Agent Requirements

Sekce požadavků agent requirements (požadavky na agenty) obsahuje list požadavků, které by měli agenti splňovat, aby mohli spustit sestavení.

8.2 Jenkins

Opět je nutné nejdříve Jenkins nainstalovat a poté vytvořit hlavní projekt. Po zadání výchozí adresy http://localhost:8080/ do prohlížeče se v Jenkins na hlavní stránce vytvoří buď pipeline nebo jen freestyle projekt. Pro účely této práce je vybrán freestyle projekt, kde budou vykonána všechny potřebná nastavení sestavení jako v TeamCity. [69]

Sekcemi nastavení v Jenkins jsou [70]:

- General (Obecné nastavení);
- source code management SCM/VCS (řízení zdrojového kódu);
- triggers (spouštěče);
- build environment (prostředí sestavení);
- build (kroky sestavení).

General

Sekce general (obecné nastavení) poskytuje možnost popsat právě vytvářený projekt a sestavení. Dovoluje nastavit projektovou URL adresu do repozitáře (GitHub), zda se jedná o parametrizovaný projekt, zda budou artefakty ukládány, jestli je projekt aktivní. Dále lze nastavit další zdroje pro sestavení, požadovaný minimální čas mezi sestaveními a zda je aktivní možnost paralelně běžících sestavení. [71]

Source code management

V sekci source code management (řízení zdrojového kódu) se nastavuje odkud a jakým způsobem se načítá kód a odkud se kontrolují změny ve zdrojovém kódu softwaru ze systému pro správu verzí softwaru (SCM – source code management, v TeamCity VCM). Source code management se nemusí nastavovat vůbec. Pokud se však možnost SCM zaškrtne, je potřeba doplnit URL repozitáře. [72]

Triggers

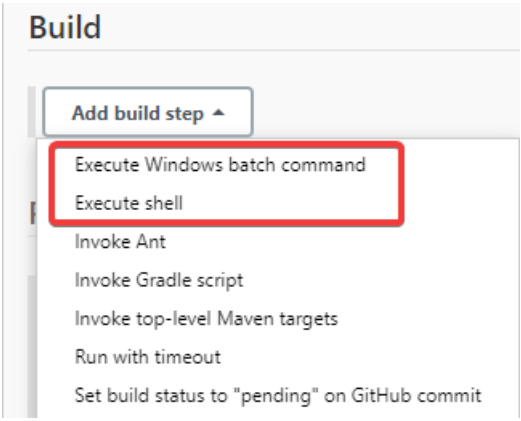
Triggers (spouštěče) nastavení aktivuje různé druhy dostupných spouštěčů buildu. Ve výběru existuje možnost pouštět buildy vzdáleně pomocí skriptu, možnost spustit build po dokončení jiného buildu, spouštět build pomocí nastaveného rozvrhu či další alternativy. [70]

Build Environment

V sekci build environment (nastavení prostředí) se nachází potřebné nastavení pracovního prostředí Jenkins, ve kterém sestavení běží. Jednou z možností je si pracovní prostředí pokaždé promazat než se spustí nové sestavení, zastavit sestavení, pokud běží příliš dlouho, přidat si časová razítka do výstupu konzole, vytvořit sestavení pomocí Apache Ant a další možnosti. [70]

Build

Build (sestavení) sekce umožňuje přidat vybrané kroky sestavení, které se budou vykonávat. Možností jsou pouštět příkazy pro lokální OS (Windows nebo Linux), vyvolat Apache Ant, vyvolat Gradle skript (build systém) a další kroky. Právě v této sekci se natavuje možnost spustit Windows či Linux příkaz pro spuštění automatického testu, viz Obrázek 33.



Obrázek 33 - Výběr kroků sestavení, Zdroj: Vlastní zpracování

V případě shell příkazu na Linuxu vypadá daný příkaz na vykonání určitého testu, uložení výsledků testu a průběhového logu, příkladem v JMeteru, stejně jako v TeamCity [58]:

> *„jmeter –n –t [lokace test skriptu] –l [lokace souboru s výsledky testu]" -j [lokace logu průběhu vykonávání testu]*

Po kliknutí na „Execute shell" přímo v Jenkins poté příkaz vypadá takto, viz Obrázek 34.

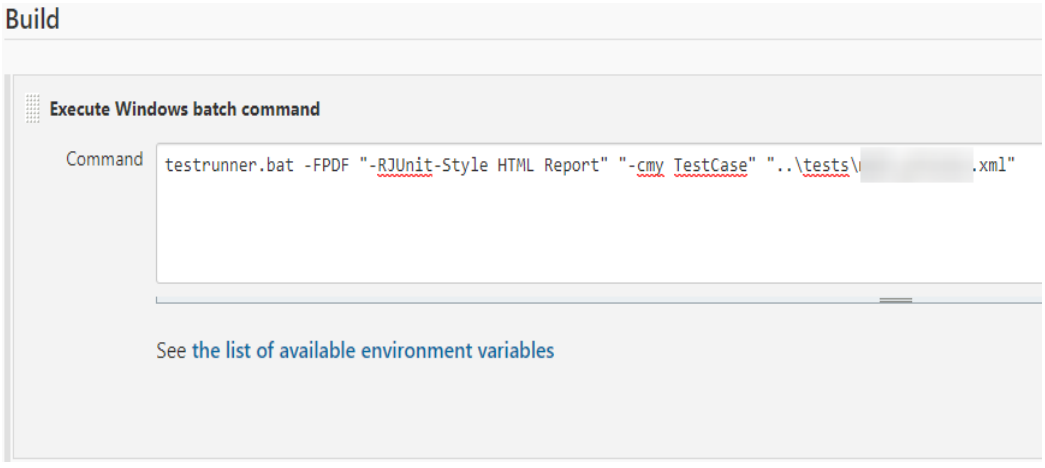
Obrázek 34 - Shell příkaz v build kroku ke spuštění testu přes Jmeter v Jenkins, Zdroj: Vlastní zpracování

Tento příkaz spustí automatický test, uloží výsledky v jakémkoliv zvoleném formátu a uloží log z průběhu vykonávání testu v jakémkoliv zvoleném formátu.

Pro spuštění testu na Windows, příkladem pro ReadyAPI, příkaz vypadá takto [73]:

„testrunner.bat [volitelné argumenty] <projekt test.xml>“

Po zadání příkazu jako „Execute Windows batch command“ přímo v Jenkins ve výběru kroků sestavení, vypadá krok poté takto, viz Obrázek 35. Tento příkaz spustí automatický test a vytvoří JUnit HTML report s vlastnostmi testu.



Obrázek 35 - Batch příkaz v build kroku ke spuštění testu přes ReadyAPI v Jenkins, Zdroje: Vlastní zpracování; [73]

9 Integrace automatických testů s test management nástroji

Integrací automatických testů s test management nástroji se rozumí posílání výsledků testů a dalších údajů určujících podrobnosti testů pomocí API http requestů (REST nebo SOAP) do test management nástrojů, kde se poté může s automaticky vyplněnými a otestovanými use casy dále pracovat. S využitím této integrace se mohou vytvářet automaticky vyplňované testovací sady, které se využijí při vytváření reportů. Reporty se poté mohou po každém spuštění automatických testů a poslání výsledků a informací o testech do test management nástrojů odevzdat všem zainteresovaným stranám.

Trh nabízí více test management nástrojů, tato práce se zaměřuje na nástroje PractiTest a Zephyr. Practitest se využívá na několika projektech společnosti Unicorn a Zephyr je jeden ze sady nástrojů, které vyvíjí společnost SmartBear, která implementuje také nástroje ReadyAPI a SoapUI. Oba nástroje jsou komerční. Jedním z užitečných příkladů integrace automatických testů s test management nástroji je automatizované vytvoření testovacích sad, do kterých jsou přidávány test instance případů užití (use casů), které jsou následně upraveny vyplněním detailů z jejich automatizovaného spuštění.

Tato integrace je ukázána v dalších kapitolách práce, pro její realizaci s PractiTestem a Zephyrem je využit nástroj Apache JMeter ve verzi 5.3. Samozřejmě je možné využít i jiné nástroje zmiňované v předešlých kapitolách této práce.

9.1 PractiTest

Practitest poskytuje své vlastní JSON API pro komunikaci s jinými nástroji. Pomocí tohoto API lze získat seznam všech testů (případů užití), které jsou v PractiTestu vytvořené v sekci test library, je možné získat určitý test, vytvořit si nový test, updatovat vybraný test, smazat vybraný test či spustit vybranou instanci testu z repozitáře testů (v případě test management nástrojů to znamená manuálně začít vyplňovat průběh případu užití testerem). [74]

Obdobnými metodami lze manipulovat také s kroky testu, instancemi testů v repozitáři testů, celými testovacími sadami či klientskými požadavky, které se mohou v PractiTestu propojit s use casy. [74]

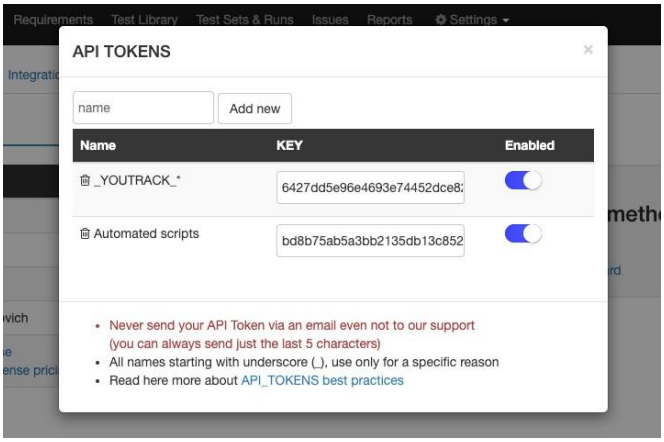
Před odesíláním informací o testu (use casu), který je vykonán v nástroji pro vytváření a spouštění automatických testů, je potřeba vykonat několik prerekvizit.

První prerekvizitou je nutnost do testu zakomponovat manipulaci s API tokenem PractiTestu. Nabízí se dva různé druhy tokenů [75]:

1. Osobní API token
2. Účtový API token

Osobní API token je unikátní jednomu jedinému uživateli a dává mu přístup k metodám a oblastem, ke kterým má přes API token přístup. Tento token je využíván častěji, protože jej může využívat pouze jeden uživatel, který si svůj osobní API token udržuje v utajení, a přitom nemá práva na všechny projekty či metody. Tento token má tedy výhodu v bezpečnosti. [75]

Účtový API token připisuje práva pro využívání CRUD metod v rámci všech projektů, které patří danému účtu. Stejný účtový API token může využívat více lidí, z tohoto důvodu a z důvodu velkých práv, které tento token poskytuje, by se tento druh tokenu měl uchovávat ve větším utajení než osobní API token. Oba druhy API tokenů se nejdříve musí aktivovat v nastavení účtu v PractiTestu viz Obrázek 36. [75]



Obrázek 36 - API Tokeny v nastavení účtu v PractiTestu [75]

Po získání API tokenu se tento token předá do konfiguračního elementu „HTTP Authorization Manager“ pro přihlášení do PractiTestu. Druhou prerekvizitou je vytvoření testovací sady, do které se poté budou přidávat všechny jednotlivé API a GUI testy. Request pro vytvoření testovací sady v PractiTestu vypadá takto [76]:

POST https://api.practitest.com/api/v2/projects//${id_projektu}/sets.json

POST data těla requestu [76]:

{"data":

```
        {"type": "sets", "attributes":
                {"name": "one", "priority": "highest", "custom-fields":
                        {"---f-22": "Windows", "---f-24": ["ClientA", "ClientB"]}
                }
        }
}
```

Třetí prerekvizitou je prvotní manuální či automatizované vytvoření testu (use casu) v test knihovně PractiTestu, ze kterého se získá id. Pomocí id je poté určitý případ užití v automatickém testu propojen se svým testem (případem užití) v knihovně testů. Čtvrtou prerekvizitou je poté přidání test instance do testovací sady, podle již zmíněného id z vytvoření testu. Nakonec jsou přidány všechny potřebné hodnoty a statusy test instancí v testovací sadě. Všechny potřebné hodnoty a konečné statusy pro instance se mohou extrahovat či předat jako parametr testu. Request pro vytvoření instance testu v testovací sadě vypadá takto [77]:

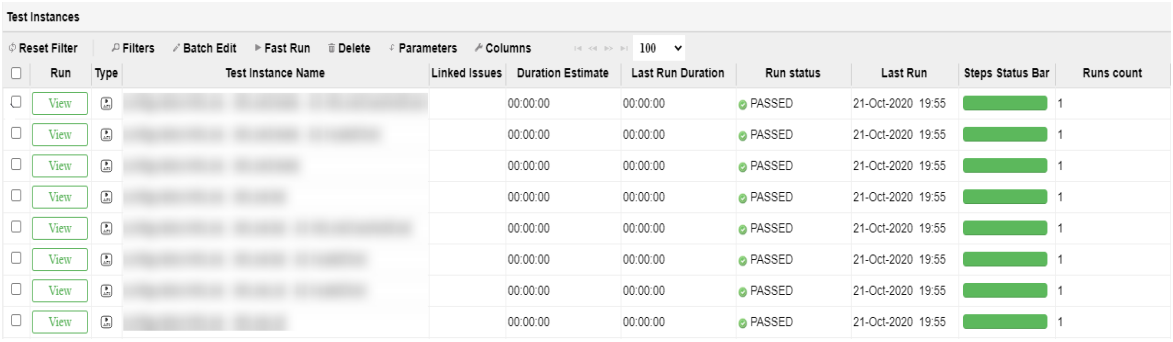
```
POST https://api.practitest.com/api/v2/projects/${id_projektu}/instances.json
POST data:
{"data":
        [{"type": "instances", "attributes":
                        {"test-id": ${id_testu_z_test_library},
                         "set-id": ${id_test_setu}}
        }]
}
```

Konečný request pro vyplnění instance potřebnými hodnotami atributů a statusem vypadá takto [78]:

```
POST https://api.practitest.com/api/v2/projects/11377/runs.json
POST data:
{"data": [{"attributes": {"instance-id": "${id_instance}","exit-code": 0,"run-duration": "0:0:0"},
```

```
"steps": {"data": [{
                "name": "${jméno_stepu}",
                "expected-results": "co se očekává",
                "actual-results": "co se opravdu stalo",
                "status": "${PASSED nebo FAILED}",
            {
                "name": ""${jméno_stepu}",
                "expected-results": "co se očekává",
                "actual-results": "co se opravdu stalo",
                "status": "${PASSED nebo FAILED}"
            }]}
        }]
}
```

V PractiTestu je po každém spuštění automatických testů v nástroji pro spouštění a vytváření automatických testů či přes CI/CD nástroj vytvářena testovací sada s vyplněnými a otestovanými instancemi use casů, které jsou napojené na testy (use casy) v knihovně testů PractiTestu, viz Obrázek 37.



Obrázek 37 – Testovací sada s automaticky vyplněnými test instancemi v PractiTestu, Zdroj: Vlastní zpracování

9.2 Zephyr

Zephyr je test management nástrojem od společnosti SmartBear, která vyvíjí také nástroje ReadyAPI a SoapUI. Nástroj je komerční a nabízí dvě možnosti jeho zapojení do testovacího procesu. Existuje možnost zapojit Zephyr jako rozšíření Jira bug tracking nástroje od společnosti Atlassian (nástroj umožňuje management chyb, úkolů, požadavků a dalších artefaktů projektu). Další možností, jak zapojit Zephyr, je zapojení jako samostatný nástroj, který lze také integrovat s Jira nástrojem. Samozřejmě také nástroj PractiTest lze integrovat s Jira nástrojem. [79] [80]

Tato diplomová práce uvažuje pouze verzi samostatného nástroje ve verzi 6.0. a vyšší. Zephyr umožňuje vytvořit klientské požadavky v záložce „požadavky". V záložce „repozitář testů" (v PractiTestu knihovna testů) lze vytvořit jednotlivé testy (případy užití), které jsou dále v záložce „plánování testů" (v PractiTestu test sets) naplánovány do projektových cyklů vybraného typu (sprint, fáze atd.). V další záložce „provedení testů" se naplánované testy v cyklech následně vykonávají a postupně se označují statusy jednotlivých kroků testu. Při nalezení defektu se může připojit Jira ticket k případu užití či jej lze přímo uvnitř Zephyru vytvořit. [81]

Stejně jako PractiTest také nástroj Zephyr poskytuje API pro další integrace s test nástroji jako jsou JMeter, ReadyAPI či SoapUI. Dále umožňuje přímou integraci s CI/CD nástrojem Jenkins přes plugin. [82] S metodami, které Zephyr poskytuje, lze vykonat tyto úkony [83]:

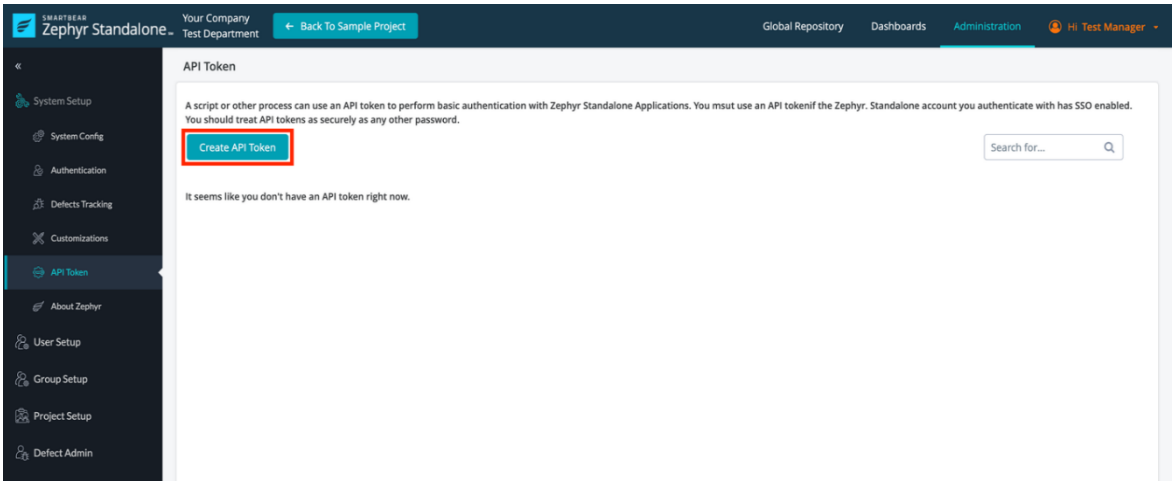
- *„Získat informace o uživatelích, projektech, verzích, repozitářích s testy (use casy) a základních složkách, testech (případech užití), cyklech provádění testů;*
- *vytvořit nové složky testů, testy (use casy), požadavky, prováděcí cykly, fáze provádění;*
- *vytvořit/upravit/smazat přílohy a fáze jednotlivých cyklů;*
- *přiřadit plány provádění testů;*
- *aktualizovat testy a stavy (statusy) provedení testů."*

Prerekvizitami pro nahrání informací o automatizovaném provedení určitého testu (use casu) jsou:

- Prvotní získání API tokenu;
- vytvoření testu (use casu) v test repozitáři;

- vytvoření cyklu (v PractiTestu test setu) do kterého jsou následně přidány instance testů z repozitáře;
- přidání testu do určitého cyklu;
- nahrání informací a statusu jednotlivých kroků testu společně se statusem celého testu.

Zephyr poskytuje individuální API token pro každý účet. Tento token je získán uvnitř nástroje po kliknutí na tlačítko „vytvořit API token“, viz Obrázek 38, nebo pomocí POST metody „http://localhost:80/flex/services/rest/latest/usertoken“ [84]. Tyto tokeny nemají expirační datum. Token se poté může využít v automatickém testu v rámci hlavičky každého následně využitého requestu. [85]



Obrázek 38 - Vytvoření API tokenu v Zephyru [84]

Nový test (případ užití) v repozitáři lze vytvořit jak automatizovaně, tak manuálně v záložce „repozitář testů“. Následně je nutné vytvořit nový cyklus. Pro jeho vytvoření má API Zephyru následující POST metodu cycle [86]:

POST http://localhost:80/flex/services/rest/latest/cycle

POST body:

```
{
 "id": 8,
 "name": "zephyr",
 "startDate": 1562610600000,
 "endDate": 1563733800000,
 "cycleStartDate": "07/09/2019",
 "cycleEndDate": "07/22/2019",
```

"status": 1,

"revision": 10,

"releaseId": 5,

"cyclePhases": [],

"createdOn": 1562668935969

}

Do tohoto cyklu je možné vytvořit fázi pomocí POST metody „http://localhost:80/flex/services/rest/latest/cycle/cycleid“. [87]

Dalším krokem je přidání testu do fáze uvnitř cyklu. Pro tento účel slouží POST metoda [88]:

POST
http://localhost:80/flex/services/rest/latest/assignmenttree/cyclephaseid/assign/bytree
POST body:

[{

"treeid": ID stromu, který je třeba přidat do fáze (ten který je v repozitáři testů),

"tctIds": [],

"isExclusion": false

},

{

"treeid": ID stromu, který je třeba přidat do fáze (ten který je v repozitáři testů),

"tctIds": [],

"isExclusion": false

}]

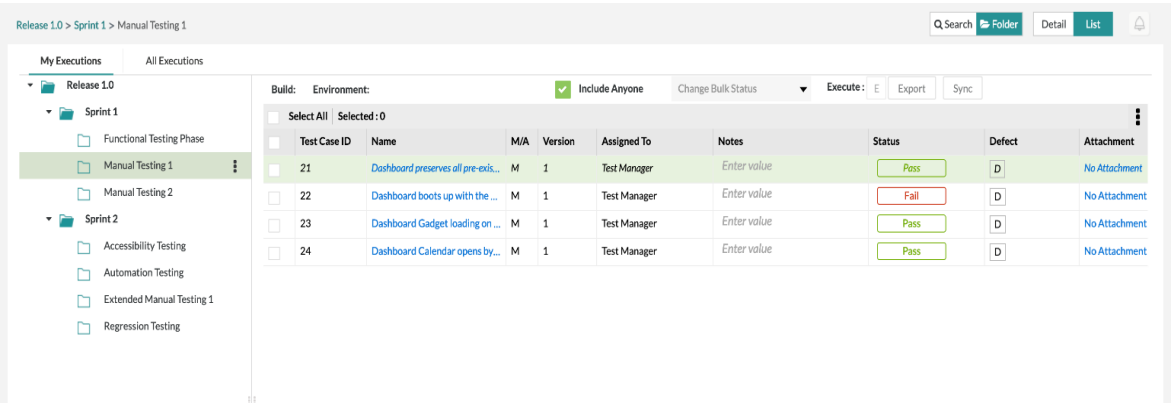
Na závěr jsou test instance, přiřazené ve fázi cyklu, hromadně vykonány za pomoci POST API metody Zephyru [89]:

POST http://localhost:80/flex/services/rest/latest/execution/bulk

POST body:

```
{
 "ids": [20,
 18,
 19],
 "selectedAll": 0,
 "serachView": false,
 "teststepUpdate": true,
 "teststepStausId": "2" (1 – PASS; 2- FAIL; 3- WIP; 4- Blocked)
}
```

Po hromadném vykonání se mohou případně všechny detaily jednotlivých testů upravit metodou: „http://localhost:80/flex/services/rest/latest/execution/statusandattachment“. [90] Všechna data pro předchozí requesty jsou buď získány z parametrů testu či se extrahují po vykonání testu z předcházejících requestů. Instance testů ve fázi v rámci určitého cyklu poté mají přiřazen svůj status a další detaily, které odpovídají jejich skutečnému automatizovanému vykonání přes nástroj pro vytváření a spouštění automatických testů, viz Obrázek 39.



Obrázek 39 - Záložka "vykonání testů" s automaticky vyplněnými detaily test instancí v Zephyru [91]

10 Výsledky práce

Cílem práce bylo prozkoumat možnosti tvorby automatických testů a možnosti jejich integrace. V diplomové práci byly identifikovány a definovány základní druhy testů, vhodné pro automatizaci ze strany testovacího týmu na projektech společnosti Unicorn či jiných společností. Jsou jimi smoke testy pro rychlé otestování softwaru a podávání průběžné zpětné vazby vývojářům, regresní testy pro zpětné hledání nových a zanesených defektů, systémové testy pro testování komplexních funkcionalit, integrační testy a performance testy pro ověření nefunkčních vlastností softwaru. Dále byly identifikovány nástroje kontinuálního vývoje a test managementu a možnosti jejich integrace s automatickými testy. Jednou z elementárních možností integrace automatických testů s nástroji kontinuálního vývoje je automatizované pouštění testů i s generováním artefaktů a logů přes CLI mód v rámci jednoho z kroků sestavení. Integraci automatických testů s test management nástroji je možné řešit pomocí API těchto nástrojů, které umožňují vytvářet případy užití i testovací sady s případy užití potřebnými v konkrétním vývojovém cyklu, které se poté, opět přes API, doplňují o detaily z jejich automatizovaného vykonání.

V průběhu zkoumání cílů této práce se vynořily důležité prerekvizity, které musí být brány v úvahu před započetím automatizace testů na projektu. Před započetím automatizace testů je důležité zjistit, které druhy je potřeba automatizovat, aby generovaly přidanou hodnotu pro projekt. Kromě vybraných druhů testů je nutné vybrat typy automatických testů, které se pokryjí. Budou se automatické testy zaměřovat pouze na testování pomocí API nebo se bude automaticky testovat také GUI softwaru? Zkoumané zdroje ukazují, že pro komplexní testování za účelem minimalizace defektů softwaru je třeba využívat oba přístupy. Budou se využívat automatické testy databáze a automatické výkonnostní testy?

Další otázkou, na kterou je třeba si před započetím automatizace odpovědět, je její rozsah. Počáteční investice do automatizovaného testování je významná, za předpokladu, že se testy využívají často a jsou průběžně aktualizovány, jsou však přínosy v průběhu času větší než náklady. Pro automatické GUI testování je obzvláště důležité testy průběžně aktualizovat, jelikož náklady na pozdní aktualizace mohou neúměrně narůstat. Pro automatizaci testů byly vybrány tři nástroje na základě jejich současného či minulého využití na projektech ve společnosti Unicorn. Další vybrané

nástroje jsou vyvíjeny stejnými společnostmi nebo jsou všeobecně známy a hojně využívány. Dále je nutné zjistit, zda test management nástroj využívaný testery na projektu, poskytuje API pro integraci automatických testů. Jejich integrací je možné docílit automaticky vytvářených a otestovaných instancí případů užití pro generování test reportů.

Pokud chce organizace začít využívat DevOps, je důležitým aspektem využívat CI/CD nástroje, do kterých jsou integrované automatické testy. Tyto testy je možné pouštět přes CLI mód nástroje využitého při jejich vytváření, přičemž jsou generovány výsledky testů do předem definovaného souboru daného formátu a další artefakty spuštěných testů. Všechny tyto soubory jsou poté uchovávány na serveru.

11 Závěr

Diplomová práce měla za cíl prozkoumat možnosti tvorby automatických testů ve vývoji softwaru a možnost jejich integrace s nástrojem kontinuálního vývoje a nástrojem test managementu. V kapitolách této práce je čtenář uveden do základních konceptů testování softwaru, test managementu, automatizovaného testování a automatických testů, metodologie DevOps a souvisejícího tématu průběžného vývoje jakož i do nástrojů, které se pro test management a průběžný vývoj používají.

Životní cyklus vývoje softwaru a s ním také související proces testování se, následkem zvyšujících se nároků ze strany zákazníků, neustále zrychluje. Rozhodnutí automatizovat proces testování v rámci CI/CD pipeline je správným krokem k jeho zrychlení. Obzvláště na projektech, které využívají agilní metodologii či přímo metodologii DevOps je tedy automatizace testování vhodná. Stejně vhodné je využití test management nástroje integrovaného s automatickými testy.

Autor této práce vycházel převážně z vlastních zkušeností z testování na projektech společnosti Unicorn, z dostupných elektronických příspěvků ve vědeckých databázích a dokumentacích jednotlivých využitých nástrojů. Na projektu Core CC Tool 2 je automatizace využívána a průběžně udržována od počátku procesu testování. Do dnešního data je na projektu automatizovaných 36 % všech případů užití a v budoucnu se toto procento bude zvyšovat. Během jednoho vývojového cyklu jsou automatizované případy užití vykonávány mnohokrát, a tak šetří čas manuálním testerům a také projektový rozpočet. Společně se zmíněnými výhodami automatizace testů na projektu nalézá toto testování také nové bugy. Během třetího kvartálu roku 2020 se díky nim nalezlo 68 z 384 nalezených defektů softwaru, což znamená 17,70 % z jejich celkového počtu s průměrem 13,66 nalezenými defekty měsíčně. Z těchto 68 nalezených defektů bylo 4,5 % úrovně A1 (nejvyšší priorita), 1,5 % A2 (vyšší priorita), 77 % B (střední priorita) a 17 % C (nižší priorita).

Práce přináší výsledky průzkumu možností automatizace testování na projektech společnosti Unicorn či projektech jiných softwarových společnostech. Poskytuje ucelený vhled či manuál pro jeho testovací týmy či testovací týmy jiných softwarových společností využívajících agilní projektové metodologie, které s automatizací procesu testování softwaru na projektu začínají nebo týmy, které již vlastní projektové automatické testy a chtějí je integrovat s nástrojem průběžného

vývoje či s test management nástrojem. Zároveň poskytuje důležité informace pro testery, kteří začínají nebo se budou věnovat oblasti vytváření automatických testů a automatizovaného testování.

12 Přehled použitých zdrojů

[1] LEVIN, Alan, Siddharth Vikram PHILIP a Christopher JASPER. Boeing Fixing New Software Bug on Max. Bloomberg [online]. 6. února 2020 [cit. 2020-09-12]. Dostupné z: https://www.bloomberg.com/news/articles/2020-02-06/boeing-identifies-new-software-problem-on-grounded-737-max-jet

[2] OLSEN, Klaus, Stephanie a Meile ULRICH. Certified Tester Foundation Level Syllabus. ISTQB® International Software Testing Qualifications Board [online]. 2018 [cit. 2020-09-12]. Dostupné z: https://www.istqb.org/downloads/send/2-foundation-level-documents/281-istqb-ctfl-syllabus-2018-v3-1.html

[3] FELDMAN, STUART. Quality Assurance: Much More than Testing: Good QA is not only about technology,but also methods and approaches. ACM Digital Library [online]. [cit.2020-09-12]. Dostupné z: https://dl.acm.org/doi/fullHtml/10.1145/1046931.1046943

[4] ISTQB Glossary. ISTQB® International Software Testing Qualifications Board [online]. 2018 [cit. 2020-09-13]. Dostupné z: https://glossary.istqb.org/cs/search/testov%C3%A1n%C3%AD

[5] JORGENSEN, Paul. Software testing: a craftsman's approach. Fourth edition. Boca Raton, [Florida]: CRC Press, Taylor & Francis Group, [2014]. ISBN 978-1-4665-6068-0.

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

[7] PITTET, Sten. The different types of software testing. ATLASSIAN CI/CD [online]. Atlassian, c2020 [cit. 2020-10-05]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

[8] HLAVA, Tomáš. Fáze a úrovně provádění testů. Testování softwaru [online]. testovanisoftwaru.cz, 21.8.2011 [cit. 2020-10-05]. Dostupné z: http://testovanisoftwaru.cz/tag/integracni-testovani/

[9] P. MIGUEL, José, David MAURICIO a Glen RODRÍGUEZ. A Review of Software Quality Models for the Evaluation of Software Products. International Journal of Software Engineering & Applications [online]. 2014, 5(6), 31-53 [cit. 2020-09-16]. ISSN 09762221. Dostupné z: doi:10.5121/ijsea.2014.5603

[10] ISO/IEC 25010. ISO 25000 Portal [online]. 2019, 2019 [cit. 2020-09-16]. Dostupné z: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010

[11] LARSSON, Jacob, Markus BORG a Thomas OLSSON. Testing Quality Requirements of a System-of-Systems in the Public Sector - Challenges and Potential Remedies [online]. 17 Feb 2016, 15 [cit. 2020-09-16]. Dostupné z: https://arxiv.org/ftp/arxiv/papers/1602/1602.05618.pdf

[12] JAN, Syed Roohullah, et al. An innovative approach to investigate various software testing techniques and strategies. International Journal of Scientific Research in Science, Engineering and Technology (IJSRSET), Print ISSN, 2016, 2395-1990.

[13] COHN, Mike. Succeeding with Agile: Software Development Using Scrum. Addison-Wesley, 2010. ISBN 0321579364.

[14] SUBRAMEYER, Raj. Context-Based Automated Testing: Rethinking the Test Pyramid. Ranorex [online]. Houston, TX 77092, USA, c2020, Mar 26, 2020 [cit. 2020-09-20]. Dostupné z: https://www.ranorex.com/blog/rethinking-test-automation-pyramid/

[15] GAROUSI, Vahid a Mika V. MÄNTYLÄ. When and what to automate in software testing? A multi-vocal literature review. Information and Software Technology [online]. 2016, 76, 92-117 [cit. 2020-09-21]. ISSN 09505849. Dostupné z: doi:10.1016/j.infsof.2016.04.015

[16] BANGARE, SUNIL, SEEMA BORSE, PALLAVI BANGARE a SHITAL NANDEDKAR. AUTOMATED API TESTING APPROACH [online]. Department of Information Technology, STES's Sinhgad Academy of Engineering, Pune-48, Maharashtra, India, February 2012, 4(02), 673-676 [cit. 2020-09-24]. ISSN 0975-5462. Dostupné z: https://www.researchgate.net/publication/267823533_AUTOMATED_API_TESTING_APPROACH

[17] ISHA, Abhinav SHARMA a M. REVATHI. Automated API Testing. In: 2018 3rd International Conference on Inventive Computation Technologies (ICICT) [online]. IEEE, 2018, 2018, s. 788-791 [cit. 2020-09-24]. ISBN 978-1-5386-4985-5. Dostupné z: doi:10.1109/ICICT43934.2018.9034254

[18] REICHERT, Amy. Testing APIs protects applications and reputations. SearchSoftwareQuality [online]. c2006-2020, 24 Mar 2015 [cit. 2020-09-24]. Dostupné z: https://searchsoftwarequality.techtarget.com/tip/Testing-APIs-protects-applications-and-reputations

[19] What Is API Testing? SMARTBEAR [online]. SmartBear Software, © 2020, 2020 [cit. 2020-09-27]. Dostupné z: https://smartbear.com/solutions/api-testing/

[20] API Features Individualizing of Web Services: REST and SOAP. International Journal of Innovative Technology and Exploring Engineering [online]. 2019, 8(9S), 664-671 [cit. 2020-09-28]. ISSN 2278-3075. Dostupné z: doi:10.35940/ijitee.I1107.0789S19

[21] What is SOAP: Formats, Protocols, Message Structure, and How SOAP is Different from REST. AltexSoft [online]. AltexSoft, c2020, 20 Aug, 2019 [cit. 2020-09-28]. Dostupné z: https://www.altexsoft.com/blog/engineering/what-is-soap-formats-protocols-message-structure-and-how-soap-is-different-from-rest/

[22] Getting Started with REST Testing in SoapUI: What is a RESTful API? SoapUI [online]. SmartBear Software, c2020 [cit. 2020-09-28]. Dostupné z: https://www.soapui.org/docs/rest-testing/

[23] SOAP vs REST 101: Understand The Differences: SOAP vs REST. SoapUI [online]. SmartBear Software, c2020, 2020 [cit. 2020-09-28]. Dostupné z: https://www.soapui.org/learn/api/soap-vs-rest-api/

[24] The Selenium Browser Automation Project [online]. Software Freedom Conservancy (SFC), c2013-2020 [cit. 2020-09-28]. Dostupné z: https://www.selenium.dev/documentation/en/

[25] The Selenium project and tools. The Selenium Browser Automation Project [online]. Software Freedom Conservancy (SFC), c2013-2020, 27 Sep 2020 [cit. 2020-09-28]. Dostupné z: https://www.selenium.dev/documentation/en/introduction/the_selenium_project_and_tools/

[26] HARSHIT, Paul. Selenium WebDriver Tutorial for Cross Browser Testing. LambdaTest: Cross Browser Testing Cloud [online]. 1390 Market Street, Suite 200 San Francisco CA 94102: LambdaTest, c2020, December 27, 2018 [cit. 2020-09-28]. Dostupné z: https://www.lambdatest.com/blog/selenium-webdriver-tutorial-for-cross-browser-testing/?utm_source=Sitepoint&utm_medium=blog1&utm_campaign=SitepointAd1&utm_term=Sitepoint

[27] Remote WebDriver. The Selenium Browser Automation Project [online]. Software Freedom Conservancy (SFC), c2013-2020, 30 Sep 2020 [cit. 2020-09-30]. Dostupné z: https://www.selenium.dev/documentation/en/remote_webdriver/

[28] Grid. The Selenium Browser Automation Project [online]. Software Freedom Conservancy (SFC), c2013-2020, 30 Sep 2020 [cit. 2020-09-30]. Dostupné z: https://www.selenium.dev/documentation/en/grid/

[29] When to use Grid. The Selenium Browser Automation Project [online]. Software Freedom Conservancy (SFC), c2013-2020, 30 Sep 2020 [cit. 2020-09-30]. Dostupné z: https://www.selenium.dev/documentation/en/grid/when_to_use_grid/

[30] TIKHANSKI, Dmitri. How to Run External Commands and Programs Locally and Remotely from JMeter. BlazeMeter [online]. Broadcom, c2020, Nov 24 2015 [cit. 2020-09-30]. Dostupné z: https://www.blazemeter.com/blog/how-run-external-commands-and-programs-locally-and-remotely-jmeter

[31] OS Process Sampler. The Apache Software Foundation [online]. Apache Software Foundation, c1999 – 2020 [cit. 2020-09-30]. Dostupné z: https://jmeter.apache.org/usermanual/component_reference.html#OS_Process_Sampler

[32] Getting started with the JDBC TestStep. SoapUI [online]. SmartBear Software, c2020 [cit. 2020-09-30]. Dostupné z: https://www.soapui.org/docs/jdbc/getting-started/

[33] JSR223 Scripting. OpenHAB [online]. openHAB Foundation e.V., c2020 [cit. 2020-09-30]. Dostupné z: https://www.openhab.org/docs/configuration/jsr223.html

[34] Building a Database Test Plan. The Apache Software Foundation [online]. Apache Software Foundation, c2020 [cit. 2020-09-30]. Dostupné z: https://jmeter.apache.org/usermanual/build-db-test-plan.html

[35] TONKOV, Konstantin. MongoDB Performance Testing with JMeter. BlazeMeter [online]. Apache Software Foundation, c2020, Dec 26 2018 [cit. 2020-09-30]. Dostupné z: https://www.blazemeter.com/blog/mongodb-performance-testing-with-jmeter

[36] Performance Testing Tutorial: What is, Types, Metrics & Example. Guru99 [online]. Guru99, c2020 [cit. 2020-10-01]. Dostupné z: https://www.guru99.com/performance-testing.html

[37] Load Testing vs Stress Testing vs Performance Testing: Difference Discussed. Guru99 [online]. Guru99, c2020 [cit. 2020-10-01]. Dostupné z: https://www.guru99.com/performance-vs-load-vs-stress-testing.html

[38] ALÉGROTH, Emil, Robert FELDT a Pirjo KOLSTRÖM. Maintenance of automated test suites in industry: An empirical study on Visual GUI Testing. Information and Software Technology [online]. 2016, 73, 66-80 [cit. 2020-10-01]. ISSN 09505849. Dostupné z: doi:10.1016/j.infsof.2016.01.012

[39] PATHANIA, Nikhil. Learning Continuous Integration with Jenkins: A beginner's guide to implementing Continuous Integration and Continuous Delivery using Jenkins [online]. Livery Place, Livery Street Birmingham B3 2PB, UK.: Packt Publishing, c2016 [cit. 2020-10-03]. ISBN 978-1-78528-483-0. Dostupné z: https://books.google.cz/books?id=hgRwDQAAQBAJ&pg=PA20&dq#v=onepage&q&f=false

[40] Co je DevOps? Microsoft Azure [online]. Seatte: Microsoft, c2020 [cit. 2020-10-03]. Dostupné z: https://azure.microsoft.com/cs-cz/overview/what-is-devops/#practices

[41] ROUSE, Margaret. Continuous software development. SearchSoftwareQuality [online]. TechTarget, March 2014 [cit. 2020-10-03]. Dostupné z: https://searchsoftwarequality.techtarget.com/definition/Continuous-Software-Development

[42] MOHAMMAD, Sikender Mohsienuddin. Continuous Integration and automation [online]. 4. Sr. Associate, KPMG LLP, Department of Information Technology, Wilmington University 419 V street, Apt D, Sacramento, CA USA: IJCRT, July 3, 2016 [cit. 2020-10-03]. ISSN 2320-2882. Dostupné z: https://papers.ssrn.com/sol3/papers.cfm?abstract_id=3655567

[43] PITTET, Sten. Automated Software Testing for Continuous Delivery. ATLASSIAN CI/CD [online]. Atlassian, c2020 [cit. 2020-10-04]. Dostupné z: https://www.atlassian.com/continuous-delivery/software-testing/automated-testing

[44] Continuous Testing. Continuous Delivery [online]. Jez Humble, c2010-2017 [cit. 2020-10-04]. Dostupné z: https://continuousdelivery.com/foundations/test-automation/

[45] MARICK, Brian. Continuous Testing. Continuous Delivery [online]. Jez Humble, c2010-2017 [cit. 2020-10-04]. Dostupné z: https://continuousdelivery.com/foundations/test-automation/

[46] PITTET, Sten. Overview - What is Continuous Deployment. ATLASSIAN CI/CD [online]. Atlassian, c2020 [cit. 2020-10-04]. Dostupné z: https://www.atlassian.com/continuous-delivery/continuous-deployment

[47] PITTET, Sten. How to get to Continuous Deployment - What is Continuous Deployment. ATLASSIAN CI/CD [online]. Atlassian, c2020 [cit. 2020-10-05]. Dostupné z: https://www.atlassian.com/continuous-delivery/continuous-deployment/how-to-get-to-continuous-deployment

[48] PITTET, Sten. Continuous integration vs. continuous delivery vs. continuous deployment. ATLASSIAN CI/CD [online]. Atlassian, c2020 [cit. 2020-10-05]. Dostupné z: https://www.atlassian.com/continuous-delivery/principles/continuous-integration-vs-delivery-vs-deployment

[49] DEVOPS - What is CI/CD? Red Hat - We make open source technologies for the enterprise [online]. Red Hat, c2020, c2020 [cit. 2020-10-05]. Dostupné z: https://www.redhat.com/en/topics/devops/what-is-ci-cd

[50] PRINCE, Suzie. The Product Managers' Guide to Continuous Delivery and DevOps. Mind the Product - Conferences, training, and content for the world's largest community of product managers, designers, and developers. [online]. Mind the Product, c2011-2020, February 11, 2016 [cit. 2020-10-05]. Dostupné z: https://www.mindtheproduct.com/what-the-hell-are-ci-cd-and-devops-a-cheatsheet-for-the-rest-of-us/

[51] Pipeline. Jenkins: Build great things at any scale [online]. Jenkins, [2020], [2020] [cit. 2020-10-05]. Dostupné z: https://www.jenkins.io/doc/book/pipeline/

[52] Download the Most Advanced API Testing Tool on the Market. SoapUI [online]. SmartBear Software, c2020 [cit. 2020-10-11]. Dostupné z: https://www.soapui.org/downloads/soapui/

[53] Authentication Types. SoapUI [online]. SmartBear Software, c2020, October 09, 2020 [cit. 2020-10-11]. Dostupné z: https://support.smartbear.com/readyapi/docs/requests/auth/types/

[54] HADLEY, Marc. Web Application Description Language. W3C [online]. 4150 Network Circle, Santa Clara, California 95054, U.S.A.: Sun Microsystems, 31 August 2009 [cit. 2020-10-11]. Dostupné z: https://www.w3.org/Submission/wadl/

[55] CHINNICI, Roberto, Canon Arthur RYMAN, Jean-Jacques MOREAU a Sanjiva WEERAWARANA. Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language. W3C [online]. W3C, c2007, 26 June 2007 [cit. 2020-10-11]. Dostupné z: https://www.w3.org/TR/wsdl/

[56] .JAR File Extension. ReviverSoft [online]. Corel Corporation, c2020 [cit. 2020-10-12]. Dostupné z: https://www.reviversoft.com/en/file-extensions/jar

[57] Building a Database Test Plan. Apache JMeter [online]. Apache Software Foundation, c1999 – 2020 [cit. 2020-10-12]. Dostupné z: https://jmeter.apache.org/usermanual/build-db-test-plan.html

[58] Getting Started. Apache JMeter [online]. Apache Software Foundation, c1999 – 2020 [cit. 2020-10-13]. Dostupné z: https://jmeter.apache.org/usermanual/get-started.html#test_plan_building

[59] Best 14 CI/CD Tools You Must Know. Katalon [online]. 1776 Peachtree Street NW, Suite 200N, Atlanta, GA 30309: Katalon, c2020 [cit. 2020-10-16]. Dostupné z: https://www.katalon.com/resources-center/blog/ci-cd-tools/

[60] MEGORSKAYA, Irina, YARKO, Yegor, ed. Configuring VCS Settings. TeamCity 10.x and 2017.x Documentation [online]. JetBrains, 2017, Feb 22, 2018 [cit. 2020-10-16]. Dostupné z: https://confluence.jetbrains.com/display/TCD10/Configuring+VCS+Settings

[61] HAMILTON, Christopher, ALEXANDROVA, Julia, ed. Configuring Build Steps. TeamCity 10.x and 2017.x Documentation [online]. JetBrains, 2017, Apr 05, 2018 [cit. 2020-10-17]. Dostupné z: https://confluence.jetbrains.com/display/TCD10/Configuring+Build+Steps

[62] Introduction to the POM: What is a POM? [online]. The Apache Software Foundation, c2002–2020, 16.10.2020 [cit. 2020-10-17]. Dostupné z: https://maven.apache.org/guides/introduction/introduction-to-the-pom.html

[63] HAMILTON, Christopher, SHER, Pavel, ed. Configuring Build Triggers. TeamCity 10.x and 2017.x Documentation [online]. JetBrains, 2017, Mar 20, 2018 [cit. 2020-10-17]. Dostupné z: https://confluence.jetbrains.com/display/TCD10/Configuring+Build+Triggers

[64] MAXIMOV, Kirill, YARKO, Yegor, ed. Build Failure Conditions. TeamCity 10.x and 2017.x Documentation [online]. JetBrains, 2017, Mar 29, 2017 [cit. 2020-10-17]. Dostupné z: https://confluence.jetbrains.com/display/TCD10/Build+Failure+Conditions

[65] KHALUSOVA, Maria, ALEXANDROVA, Julia, ed. Adding Build Features. TeamCity 10.x and 2017.x Documentation [online]. JetBrains, 2017, Jul 18, 2016 [cit. 2020-10-17]. Dostupné z: https://confluence.jetbrains.com/display/TCD10/Adding+Build+Features

[66] YARKO, Yegor, ALEXANDROVA, Julia, ed. XML Report Processing. TeamCity 10.x and 2017.x Documentation [online]. JetBrains, 2017, Jan 05, 2018 [cit. 2020-10-17]. Dostupné z: https://confluence.jetbrains.com/display/TCD10/XML+Report+Processing

[67] MEGORSKAYA, Irina, ALEXANDROVA, Julia, ed. Dependent Build. TeamCity 10.x and 2017.x Documentation [online]. JetBrains, 2017, Jan 11, 2018 [cit. 2020-10-17]. Dostupné z:

https://confluence.jetbrains.com/display/TCD10/Dependent+Build#DependentBuild-SnapshotDependency

[68] KHALUSOVA, Maria, ALEXANDROVA, Julia, ed. Configuring Build Parameters. TeamCity 10.x and 2017.x Documentation [online]. JetBrains, 2017, Apr 16, 2017 [cit. 2020-10-17]. Dostupné z: https://confluence.jetbrains.com/display/TCD10/Configuring+Build+Parameters#ConfiguringBuildParameters-ConfigurationParameters

[69] ALMEIDA, Leticia. 2 Ways to Integrate JMeter Tests into Jenkins. BlazeMeter [online]. Broadcom, c2020, Jul 05 2018 [cit. 2020-10-18]. Dostupné z: https://www.blazemeter.com/blog/two-ways-to-integrate-jmeter-tests-into-jenkins

[70] Jenkins. Jenkins 2.262 [software]. Říjen 2020.[přístup 19. 10. 2020]. Dostupné z: https://www.jenkins.io/download/ [Požadavky na systém: Java 8 až Java 11 32-bit i 64-bit, operační systémy 64-bit (amd-64) Windows Server, prohlížeč Google Chrome, Mozilla Firefox, Microsoft Internet Explorer, Microsoft Edge, Apple Safari, 1 GB+ RAM, 50 GB+ volné místo na disku]

[71] How to Create Builds with the Jenkins Freestyle Project. Guru99 [online]. Guru99, c2020 [cit. 2020-10-19]. Dostupné z: https://www.guru99.com/create-builds-jenkins-freestyle-project.html

[72] Source code management. ATLASSIAN Bitbucket [online]. Atlassian [cit. 2020-10-19]. Dostupné z: https://www.atlassian.com/git/tutorials/source-code-management

[73] Command-Line Arguments. SMARTBEAR Support [online]. SmartBear Software, c2020, October 14, 2020 [cit. 2020-10-19]. Dostupné z: https://support.smartbear.com/readyapi/docs/functional/running/automating/cli.html

[74] PractiTest API V2 [online]. PractiTest, [2020] [cit. 2020-10-21]. Dostupné z: https://www.practitest.com/api-v2/#practitest-api-v2

[75] API Tokens. PractiTest [online]. PractiTest, c2020 [cit. 2020-10-21]. Dostupné z: https://www.practitest.com/help/account/account-api-tokens/#Personal%20API%20Tokens%20(PAT)

[76] PractiTest API V2: Create a TestSet. PractiTest [online]. PractiTest, [2020] [cit. 2020-10-21]. Dostupné z: https://www.practitest.com/api-v2/#create-a-testset

[77] PractiTest API V2: Create an instance. PractiTest [online]. PractiTest, [2020] [cit. 2020-10-21]. Dostupné z: https://www.practitest.com/api-v2/#create-an-instance

[78] PractiTest API V2: Update a specific instance. PractiTest [online]. PractiTest, [2020] [cit. 2020-10-21]. Dostupné z: https://www.practitest.com/api-v2/#update-a-specific-instance

[79] Zephyr for Jira. SMARTBEAR Zephyr [online]. SmartBear Software, c2020 [cit. 2020-10-23]. Dostupné z: https://www.getzephyr.com/products/zephyr-for-jira#overview

[80] Zephyr Enterprise. SMARTBEAR Zephyr [online]. SmartBear Software, c2020 [cit. 2020-10-23]. Dostupné z: https://www.getzephyr.com/products/zephyr-enterprise#features

[81] PHAM, Kevin. Zephyr User Guide: Zephyr Technical Documentation [online]. Jan 31, 2019 [cit. 2020-10-23]. Dostupné z: https://zephyrdocs.atlassian.net/wiki/spaces/ZE61/pages/280756254/Zephyr+User+Guide

[82] Zephyr Enterprise Test Management. Jenkins: Plugins [online]. [cit. 2020-10-24]. Dostupné z: https://plugins.jenkins.io/zephyr-enterprise-test-management/

[83] PHAM, Kevin. Zephyr REST API: Zephyr Technical Documentation [online]. Feb 13, 2019 [cit. 2020-10-23]. Dostupné z: https://zephyrdocs.atlassian.net/wiki/spaces/ZE6/pages/148209973/API

[84] Generate token. Zephyr Enterprise REST API: UserToken [online]. [cit. 2020-10-24]. Dostupné z: https://zephyrenterprisev3.docs.apiary.io/#reference/usertoken/generate-token/generate-token

[85] PHAM, Kevin. Create and Manage API Tokens in Zephyr. Zephyr Technical Documentation [online]. SmartBear Software, c2020, Apr 15, 2020 [cit. 2020-10-23]. Dostupné z: https://zephyrdocs.atlassian.net/wiki/spaces/ZE/pages/1558445683/Create+and+Manage+API+Tokens+in+Zephyr

[86] Create New Cycle. Zephyr for JIRA API [online]. SmartBear Software, [2020] [cit. 2020-10-23]. Dostupné z: https://getzephyr.docs.apiary.io/#reference/cycleresource/create-new-cycle/create-new-cycle

[87] Create Cycle Phase. Zephyr Enterprise REST API: Cycle [online]. [cit. 2020-10-24]. Dostupné z: https://zephyrenterprisev3.docs.apiary.io/#reference/cycle/create-cycle-phase/create-cycle-phase

[88] Add Testcase to Free form phase. Zephyr Enterprise REST API: AssignmentTree [online]. [cit. 2020-10-24]. Dostupné z: https://zephyrenterprisev3.docs.apiary.io/#reference/assignmenttree/add-testcase-to-free-form-phase/add-testcase-to-free-form-phase-via-tree

[89] Execute testcases. Zephyr Enterprise REST API: Execution [online]. [cit. 2020-10-24]. Dostupné z: https://zephyrenterprisev3.docs.apiary.io/#reference/execution/execute-testcases/to-update-execution-details,-attachment-and-status

[90] To update execution details, attachment and status. Zephyr Enterprise REST API: Execution [online]. [cit. 2020-10-24]. Dostupné z: https://zephyrenterprisev3.docs.apiary.io/#reference/execution/to-update-

execution-details-attachment-and-status/to-update-execution-details,-attachment-and-status

[91] PHAM, Kevin. Test Execution. Zephyr Technical Documentation [online]. Apr 15, 2020 [cit. 2020-10-24]. Dostupné z: https://zephyrdocs.atlassian.net/wiki/spaces/ZE/pages/1558443308/Test+Execution

Příloha číslo 1

13 Seznam obrázků

Příloha číslo 2

14 Seznam tabulek

Příloha číslo 3

15 Zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor:	**Bc. Jakub Fečo**
Studium:	I1800752
Studijní program:	N6209 Systémové inženýrství a informatika
Studijní obor:	Informační management
Název diplomové práce:	**Continuous Test Automation**
Název diplomové práce AJ:	Continuous Test Automation

Cíl, metody, literatura, předpoklady:

Cílem práce je prozkoumat možnosti automatizace testů ve vývoji software a možnosti napojení automatizovaných testů na nástroje kontinuálního vývoje.

Osnova práce:

1. Úvod
2. Obecná problematika testování SW
3. Automatizované testování
4. Automatizace testování
    1. realizace
    2. napojení na nastroje CI/CD
    3. napojení na nástroje managementu
5. Výsledky
6. Závěr

Garantující pracoviště:	Katedra informatiky a kvantitativních metod, Fakulta informatiky a managementu
Vedoucí práce:	doc. Mgr. Tomáš Kozel, Ph.D.
Datum zadání závěrečné práce:	15.9.2020