Univerzita Pardubice
Fakulta ekonomicko-správní
Ústav systémového inženýrství a informatiky

Automatizované testování SW pomocí nástroje Apache JMeter

Bc. Sebastian Vacek

Diplomová práce

2019

Univerzita Pardubice
Fakulta ekonomicko-správní
Akademický rok: **2018/2019**

ZADÁNÍ DIPLOMOVÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Bc. Sebastian Vacek**
Osobní číslo:	**E17529**
Studijní program:	**N6209 Systémové inženýrství a informatika**
Studijní obor:	**Informatika ve veřejné správě**
Název tématu:	**Automatizované testování SW pomocí nástroje Apache JMeter**
Zadávající katedra:	**Ústav systémového inženýrství a informatiky**

Zásady pro vypracování:

Cílem práce je analýza možností tvorby automatizovaného testu SW aplikace s využitím nástroje Apache JMeter a porovnání časové náročnosti manuálního a automatizovaného testování.

Práce bude obsahovat:

- základní pojmy k problematice;
- metody testování softwaru;
- Apache JMeter;
- nástroje pro management softwaru;
- praktický příklad;
- zhodnocení.

Rozsah grafických prací:

Rozsah pracovní zprávy:

Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam odborné literatury:

BUREŠ, Miroslav a kol. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.
HASS, Anne Mette Jonassen. Guide to advanced software testing. Boston: Artech House, c2008. ISBN 978-1-59693-285-2.
PATTON, Ron a David KRÁSENSKÝ. Testování softwaru. Praha: Computer Press, 2002. ISBN 80-7226-636-5.
PATTON, Ron. Testování softwaru: automatické i ruční testování, testování použitelnosti, lokalizace i kompatibility produktů nejen pro manažery softwarových projektů a testery, praktická cvičení na konci kapitol. Praha: Computer Press, c2002. Programování. ISBN 80-7226-636-5.
STEPHENS, Matt a Doug ROSENBERG. Testování softwaru řízené návrhem. Brno: Computer Press, 2011. ISBN 978-80-251-3607-2.

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

Datum zadání diplomové práce: **3. září 2018**
Termín odevzdání diplomové práce: **30. dubna 2019**

doc. Ing. Romana Provazníková, Ph.D.
děkanka

L.S.

doc. Ing. Pavel Petr, Ph.D.
vedoucí ústavu

V Pardubicích dne 3. září 2018

Prohlašuji:

Tuto práci jsem vypracoval samostatně. Veškeré literární prameny a informace, které jsem v práci využil, jsou uvedeny v seznamu použité literatury.

Byl jsem seznámen s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne 29. 4. 2019

Sebastian Vacek

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce doc. Ing. Pavlu Petrovi, Ph.D. za metodické vedení práce. Dále děkuji svému vedoucímu ve společnosti Unicorn Systems a.s. Ing. Miroslavu Ježkovi za vstřícnost a ochotu při praktickém vedení diplomové práce.

ANOTACE

Tato diplomová práce je zaměřena na nástroj Apache JMeter, který je využit pro tvorbu automatizovaných testů během vývoje a servisu mezinárodních energetických aplikací ENTSO-E Transparency Platform (ETP) a Nordic Unavailability Collection System (NUCS). Popisuje jednotlivé části daného nástroje i jeho rozšíření, které umožňují lepší využití v praxi.

Diplomová práce zároveň přináší praktický příklad tvorby automatizovaného testu pro dané aplikace a porovnání časové náročnosti manuálního a automatizovaného testování jednotlivých systémů. Hlavním cílem je poukázat na možnost propojení nástroje pro automatizované testování s dalšími aplikacemi, které usnadňují management vyvíjeného softwaru. Zde se také nachází praktický příklad propojení nástroje Apache JMeter s několika dalšími nástroji.

KLÍČOVÁ SLOVA

Apache JMeter, automatizované testování, software, testování, nástroj, management softwaru

TITLE

Automated Software Testing via Apache JMeter.

ANNOTATION

This thesis describes Apache JMeter tool, which is used for creation of automated tests during the development and maintenance of the international energy applications ENTSO-E Transparency Platform and Nordic Unavailability Collection System. It describes separate parts of automated tool including its plugins, which are used in practice.

The thesis includes the practical example of creation of automated test for these energy applications and time comparison of manual and automated testing on the individual systems. The main goal is a description of connectivity of tools for automated testing with other tools, which ease the software development. In addition, there is a practical example of connection between Apache JMeter and other tools.

KEYWORDS

Apache JMeter, automated testing, software, testing, tool, management of software

OBSAH

SEZNAM OBRÁZKŮ

SEZNAM TABULEK

SEZNAM ZKRATEK

ENTSO-E	European Network of Transmission System Operators for Electricity
ETP	ENTSO-E Transparency Platform
NUCS	Nordic Unavailability Collection System
TSO	Transmission System Operator
ROI	Return on Investment
BVT	Build Verification Testing
DevOps	Development & Operations
ASF	The Apache Way
CI	Continuous Integration
API	Application Programming Interface
RegEx	Regular Expression Extractor
US	Unicorn Systems

ÚVOD

Tématem diplomové práce je návrh automatizovaného testu pro základní testovací případy webové aplikace Nordic Unavailability Collection System (NUCS). Tento test je realizován za předpokladu dlouhodobé vývojové a servisní činnosti u daného softwaru. Téma je zvoleno záměrně, jelikož v současné době jsou automatizované testy velmi často využívány firmou Unicorn Systems (US) pro různé typy projektů.

Hlavním cílem této diplomové práce je analýza možností tvorby automatizovaného testu SW aplikace s využitím nástroje Apache JMeter a porovnání časové náročnosti manuálního a automatizovaného testování. Ke splnění hlavního cíle je nutné splnit tyto parciální cíle:

- definování základních pojmů v oblasti testování a specifikace testovacího procesu,
- vysvětlení rozdílu mezi manuálním a automatizovaným testováním,
- představení různých automatizovaných nástrojů, které jsou určeny pro testování softwaru s detailním zaměřením na nástroj Apache JMeter.

V závěru práce je poukázáno na možnost propojení nástroje Apache JMeter s nástroji, které usnadňují řízení softwaru. Dále je proveden časový rozbor na konkrétním testovacím případu aplikace NUCS.

1 TESTOVÁNÍ SOFTWARU

Testování softwaru je podle [1] nedílnou součástí vývoje softwaru. V průměru zabírá přibližně 20 % až 40 % pracnosti. Pokud se uvažuje o vývoji řídícího softwaru, který má vysoké požadavky na spolehlivost, může pracnost testování přesáhnout i pracnost samotného vývoje.

Podle [2] kdysi bývalo testování softwaru jen jakousi popelkou, na kterou zbýval čas až na konci vývoje softwaru. Jelikož se v té době dělaly produkty malé a nebyly příliš komplikované a zároveň na nich pracovalo jenom malé množství lidí. Chyby proto neznamenaly nijak podstatný problém. Postupem času se doba změnila. Softwarový průmysl pokročil do etapy, kdy vyžaduje profesionální softwarové testery. Jelikož vytvořit chybný software v dnešní době je zkrátka příliš drahé.

Testování lze chápat jako proces, při kterém se zjišťuje, že reálné vlastnosti softwaru odpovídají předem definovaným požadavkům. Tyto požadavky se zpravidla získávají během analytické fáze a jsou detailně popsány v dokumentaci, která je součástí vyvíjeného produktu. Pokud má být testování efektivní, musí být k němu přistupováno jako k samostatné disciplíně. Podobně jako k managementu projektu (projektovému řízení), analýze, architektuře či samotnému vývoji.

1.1 Základní pojmy

Tato kapitola charakterizuje vybrané základní pojmy, se kterými se lze setkat během testovacího cyklu. Jednotlivé pojmy jsou velice důležité, jelikož mají přímý vliv na výslednou kvalitu vyvíjeného systému. Tyto pojmy jsou vysvětlovány podle postavení, povinností a odpovědnosti ve společnosti Unicorn Systems. Testovací cyklus je detailněji popsán v kapitole 1.2. Více podrobnějších informací o základních pojmech lze dohledat v [2] až [6].

V první řadě je nutné charakterizovat testovací tým. Ten je tvořen určitým počtem osob v závislosti na velikosti projektu. Osoby zastávají určité pozice ve firmě

US a jsou odpovědné za jednotlivé činnosti během testovacího procesu. Mezi tyto pozice lze zařadit:

- test managera,
- test analytika,
- test architekta,
- testera.

V závislosti na praxi a velikosti projektu, který je v této diplomové práci prezentován v kapitole 4, jsou nejpodstatnější pozice test architekt a tester.

Test architekt

Test architekta lze charakterizovat jako zkušenou osobu, jejíž hlavní úlohou je navrhnout správná řešení, jak testovat vyvíjenou aplikaci. Tato řešení jsou prováděna pomocí kontextuálně relevantního procesu a praxe s využitím nástrojů, technologií a soft dovednostních aplikací, jako je například efektivní komunikace a mentorování týmu, využití automatizovaných nástrojů apod. [4]

Zároveň je odpovědný za individuální úkoly, které vyplývají z jeho pozice. Mezi tyto základní klíčové úkoly patří:

- sestavení testovací strategie,
- vytvoření testovací dokumentace,
- vyhodnocení časové náročnosti manuálního testování,
- vyhodnocení automatizace jednotlivých testovacích případů.

Prvním úkolem test architekta je sestavit testovací strategii, ve které jsou popsány jednotlivé fáze testování a osoby, které jsou odpovědné za správný průběh jednotlivých fází. Jsou zde také podrobně popsány akceptační kritéria. Ta vyznačují například maximální schválený počet nalezených chyb v dané fázi. Pokud je celkový počet nalezených chyb v příslušné fázi menší nebo roven počtu chyb zapsaných v dokumentaci, nic nebrání ve spuštění další fáze vyvíjeného produktu. Podrobnější popis tohoto dokumentu je popsán níže.

Druhým a nejdůležitějším úkolem test architekta je vytvoření testovací dokumentace. Ta je opravdu velice důležitá, jelikož se skládá z několika hlavních částí, která slouží ke správnému otestování aplikace. Podle této dokumentace provádí svou činnost tester. Pokud je v testovací dokumentaci chyba, může to vést až k situaci, kdy vydaný software nepracuje podle zadané produktové specifikace. Opravit takovýto software je poměrně finančně náročné. Detailnější popis testovací dokumentace je možné nalézt v následující kapitole Testovací dokumentace.

Dalším z úkolů test architekta ve společnosti Unicorn Systems je posouzení všech testovacích případů, které byly navrhnuty v rámci testovací dokumentace. Test architekt musí vyhodnotit pracnost manuálního testování a možnost vytvoření automatizovaného testu pro daný testovací případ. Musí přitom brát v úvahu časovou náročnost jednotlivých testů a frekvenci testování daného případu. Pokud test architekt usoudí, že daný případ by bylo efektivnější zautomatizovat, označí ho a předá ho testerovi. Rozdíl mezi a automatizovaným a manuálním testováním je možné nalézt v kapitole 1.3.

Tester

Tester je osoba, která je zodpovědná za správné testování zadaných testovacích případů. Tuto činnost může provádět manuálně nebo pomocí automatizovaných testů. Ověřuje, zda jednotlivé funkčnosti produktu splňují zadané požadavky a případně zjišťuje a následně i popisuje neadekvátní chování vyvíjeného softwaru. [5]

Zjednodušeně by se dalo říci, že cílem testera je zjistit co nejvíce chyb, které se vyskytují v produktu. Toto vnímání může být z několika důvodů zavádějící. Například pokud tester nenašel chybu nebo závadu, neznamená to, že nedělá svou práci správně. V praxi mohou testeři najít mnoho defektů v produktu, které se vyvíjí. Cílem však je konečné odstranění všech nesrovnalostí, které narušují produkční běh softwaru.

Testovací dokumentace

Nejdůležitější pro testera je testovací dokumentace, podle které je schopný kvalitně otestovat vyvíjený software. Testovací dokumentace se skládá z několika částí a zároveň se jednotlivé části vzájemně doplňují. Tyto části se dají stručně shrnout do několika následujících bodů:

- **Testovací případ (angl. Test Case)** je množina instrukcí (kroků – angl. Test Steps), které jsou provedeny na testovaném softwaru s cílem zjistit, zda tento systém pracuje dle očekávání. Tyto případy určuje test architekt z produktové specifikace, jež je nedílnou součástí vyvíjeného softwaru. Specifikace popisuje jednotlivé funkcionality produktu i chování produktu jako celku. [6]

  Skupiny testovacích případů simulují reálné používání daného produktu zákazníkem. Důležité je, aby test architekt nezapomněl na možné alternativní případy, které by mohly nastat během provozu softwaru, a tím ohrozit celkovou kvalitu produktu.
- **Testovací data (angl. Test Data)** jsou souborem dat, která jsou speciálně identifikovaná pro využití v rámci testovacího případu. Jednotlivá data jsou vždy na vstupu do aplikace analyzována a je k nim určen předpokládaný výsledek, kterého by měl zadaný software docílit. Dosažený výsledek ze softwaru je posléze porovnáván s předpokládaným výsledkem, který měl z příslušných dat vzniknout. [6]

  Doporučení: Optimální je začít pracovat na testovacích datech ještě před zahájením testování, jelikož je na tvorbu dat více času a zároveň je známa veškerá potřebná dokumentace, která je nutná pro správné vytvoření.
- **Mapa pokrytí (angl. Traceability Matrix)** uvádí, zda ke každému požadavku existuje minimálně jeden test, který ověří správnou implementaci. Jednotlivé požadavky jsou identifikovány již z produktové specifikace. [6]
- **Výsledky testu (angl. Test Report)** je detailní soubor informací obsahující minimálně identifikátor testu a jeho jednoznačný výsledek. Součástí mohou být

komentáře, grafy nebo tabulky o reálném testovacím kole (běhu) daného softwaru, je-li třeba i další upřesňující informace. Všechny tyto informace jsou doplňovány individuálně podle požadavků zákazníka. Jednotlivé součásti testovacího reportu se mohou lišit také podle přístupu různých firem. [6]

Poznámka: Ideální testovací report by měl obsahovat, popis jednotlivých testovacích případů včetně testovacích kroků. Dále kdo tyto případy testoval, kdy tyto případy testoval a zda test proběhl v pořádku či nikoliv. Pokud test neproběhl podle očekávání, je součástí testovacího reportu i popis softwarové chyby. Pro přehlednost zákazníka je důležité, aby veškeré informace byly zobrazeny v grafu, který ukazuje ucelený pohled na příslušné testovací kolo.

Testovací strategie

Testovací strategie, jak již bylo řečeno, je dokument, který popisuje metodiku testování použitou v rámci projektu. Stanovuje primární okruhy aktivit, odpovědnosti rolí a výstupní kritéria testování. Jsou zde definovány například cíle testování, fáze testování a jejich milníky, typy testů, role a jejich odpovědnosti, testovací prostředí a data, defect tracking[1], přístup k test analýze nebo identifikace rizik a způsobů, jak je eliminovat. Dobrá testovací strategie by měla mít následující vlastnosti [6]:

- Rozmanitá – měla by obsahovat různorodé kombinace typů testů a testovacích technik. Tato podmínka zajišťuje, že všechny kvalitativní oblasti systému budou ověřeny.
- Zaměřená na rizika – pokud se testování zaměří na rizika, umožňuje vhodně identifikovat priority testů pro plánování.
- Specifická na produkt – každý vyvíjený produkt má jiné vlastnosti, proto se musí také jinak testovat.

[1] Je proces sledování zaznamenaných chyb v aplikaci od založení až po uzavření této chyby.

- Praktická – testovací aktivity musí být efektivní. Musí být vybrány způsoby, jak zajistit jednoduché a účinné testování v rámci daného rozpočtu.
- Obhájitelná – všechny testovací aktivity musí být zdůvodněny, proč jsou navrženy a prováděny tak, jak byly popsány.

Poznámka: Tato dokumentace je velice důležitá. Definuje hranice, které jsou kritické pro spuštění softwaru v reálném provozu. Proto musí být schvalována zákazníkem, který svým podpisem souhlasí s nastavením jednotlivých kritérií.

Softwarová chyba

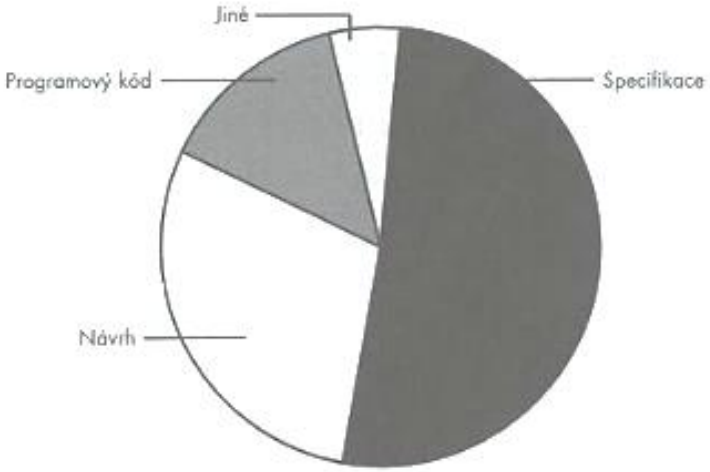
Softwarovou chybu je možno chápat jako určité selhání softwaru, ať už se jedná o chybu v počítačové hře, která může způsobit určité nepohodlí, nebo také katastrofu, která znamená třeba i ztráty na životech. Ve všech těchto případech bylo jasné, že software nepracoval v souladu s výchozím zadáním či původním záměrem.

V běžném životě se může člověk setkat s různými pojmy, které pojmenovávají určitou chybu, jako je například vada, závada, problém, omyl, odchylka, selhání, nekonzistence, chyba apod. Jelikož existuje mnoho výrazů pro popis softwarové chyby, závisí skutečně pouze na kultuře dané firmy, jaký výraz upřednostní.

Jak uvádí [2], chyba se dá nejjednodušeji definovat, pokud je splněna jedna nebo více z následujících podmínek.

- Software nedělá něco, co by podle specifikace produktu dělat měl.
- Software dělá něco, co by podle specifikace produktu dělat neměl.
- Software dělá něco, o čem se produktová specifikace nezmiňuje.
- Software nedělá něco, o čem se produktová specifikace nezmiňuje, ale měla by se zmiňovat.
- Software je obtížně srozumitelný, těžko se s ním pracuje, je pomalý, nebo podle názoru testera softwaru, jej koncový uživatel nebude považovat za správný.

Chyby vznikají z několika příčin. Podle řady studií, které byly provedeny nad projekty různých velikostí, od velmi malých až po extrémně velké, jsou příčiny vždy stejné. U softwarových chyb je hlavní příčinou obsah specifikace. Ten podle [2] zapříčiňuje přes 50 % nalezených chyb. Jednotlivé příčiny vzniku chyb odráží obrázek 1.



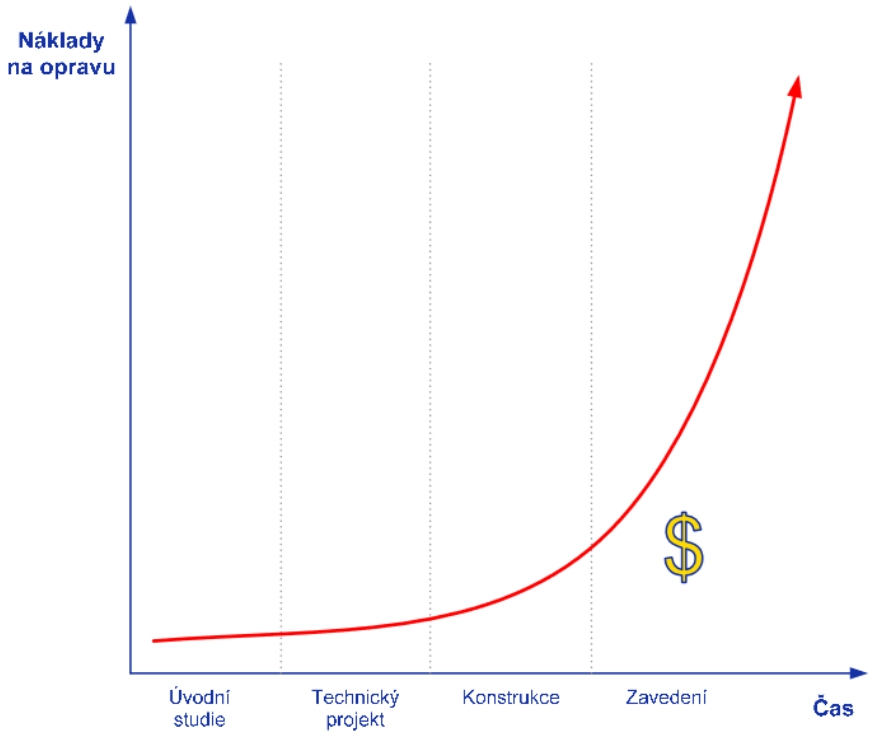
Obrázek 1 - Rozdělení příčin vzniku chyb. *Zdroj: [2]*

To, že je obsah specifikace hlavní příčinou vzniku chyb, má několik důvodů. Jednou z příčin může být to, že není dostatečně detailně popsán, nebo se neustále mění, nebo nebyl konzultován se všemi členy vývojového týmu. Plánování softwaru je tak opravdu nejdůležitější. Pokud se neprovede správně, hrozí, že se do softwaru zanesou chyby.

Důležitou otázkou zůstává, proč vůbec vyvíjený software testovat? Na tuto otázku existuje zcela jasná odpověď, a tou je hledisko nákladů na opravu chyb daného softwaru. Čím dříve je chyba v dané aplikaci nalezena, tím levnější je její oprava a naopak.

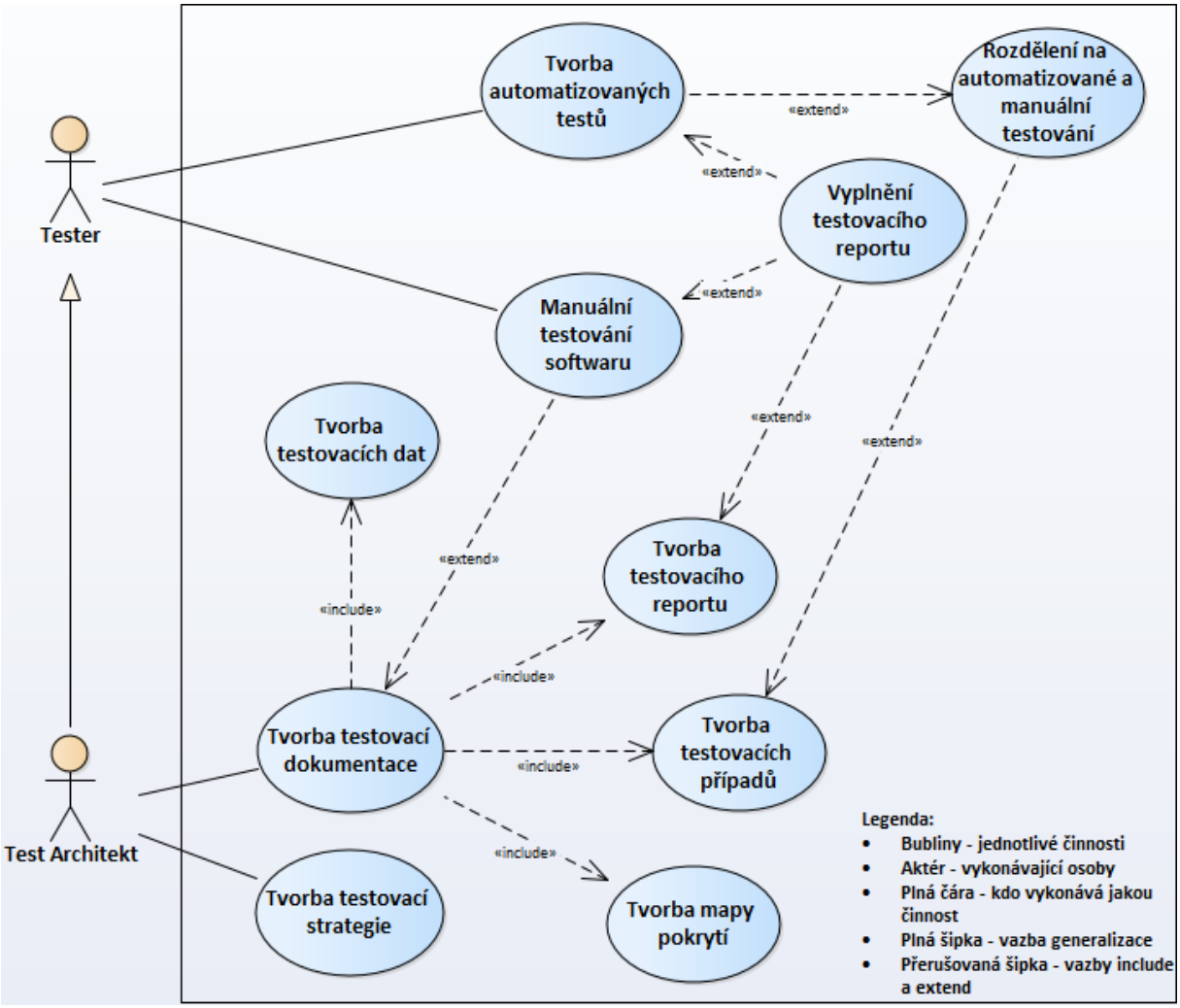
Pokud se chyba nalezne až v produkční verzi, náklady na opravu rostou, a zároveň to může způsobit i určitou finanční ztrátu pro samotného zákazníka, který ji následně bude vymáhat od dodavatele softwaru. Naopak, jestliže je zjištěna určitá nekonzistence už během úvodní studie projektu, náklady na opravu takto nalezené chyby jsou minimální. Jak se zvyšují jednotlivé náklady na opravu defektu v průběhu

času, od úvodní studie systému až po jeho zavedení do produkce, je znázorněno na obrázku 2.



Obrázek 2 - Náklady na opravu v čase. *Zdroj: [6]*

Poznámka: K přehlednějšímu pochopení jednotlivých základních pojmů může pomoci následující obrázek 3, který ukazuje souvislosti mezi popisovanými pojmy. Dále zobrazuje, která osoba je odpovědná za jakou činnost. Mezi test architektem a testerem existuje vazba generalizace, jelikož test architekt může zastávat činnosti testera, ale nikoliv opačně. Vazba extend charakterizuje například vyplnění testovacího reportu, který rozšiřuje manuální testování softwaru. Zároveň platí, že manuální testování softwaru nevyžaduje vyplnění testovacího reportu. Zatímco vazba include znázorňuje, že jedna činnost je nedílnou součástí druhé činnosti. Obě činnosti musí být splněny pro správný průběh testování.



Obrázek 3 - Provázání jednotlivých činností. *Zdroj: Autor*

1.2 Testovací proces

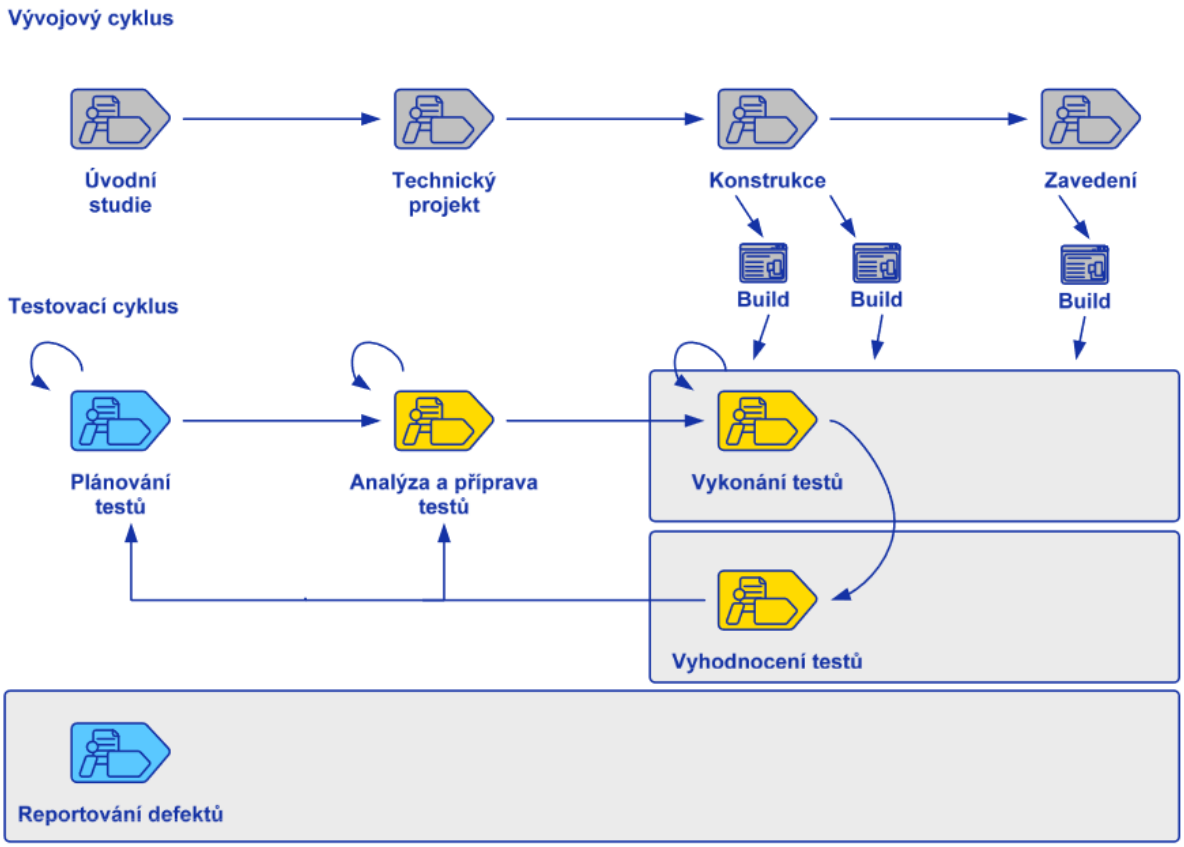
Testovací proces je postup určité skupiny lidí, který lze definovat v několika bodech. Tyto body charakterizují jednotlivé aktivity, které jsou potřeba vykonat pro správné a důkladné testování vyvíjeného softwaru. Podobně jako u ostatních disciplín, je i u testování nutné dbát ohled na velikost projektu. Z tohoto důvodu je potřeba přizpůsobit i testovací proces tak, aby vyhovoval dané situaci. Celý postup je popsán podle metodiky a standardů ve firmě Unicorn Systems za pomoci [1], [6] až [16].

U testovacího procesu je nutné definovat [6]:

- Vstupy a výstupy z jednotlivých aktivit, lze to také shrnout jako „**CO**“ je vstupem a výstupem u dané aktivity.

- Odpovědnostní role v rámci celého procesu, tzn. „**KDO**“ je odpovědný za danou aktivitu.
- Všeobecně uznávané normy a standardy, jinými slovy řečeno způsob, „**JAK**“ se bude postupovat.
- Vlastní životní cyklus, tzn. posloupnost aktivit, které je potřeba provést. Tato vlastnost by se dala shrnout jako „**KDY**“ se jednotlivé aktivity uskutečňují.

Pokud se uvažuje, že existuje určitý životní cyklus softwarového projektu, potom testovací proces je jeho součástí. Tím se zajišťuje kompletní portfolio služeb v oblasti testování softwaru a kontroly kvality. Jednotlivé části testovacího procesu jsou inicializovány právě v určitých částech vyvíjeného produktu. Detailní popis životního cyklu softwarového projektu ve firmě Unicorn Systems je možné nalézt v [3]. Obrázek 4 zobrazuje obecný a jednoduchý pohled na propojení obou cyklů.



Obrázek 4 - Životní cyklus testování jako součást vývojového cyklu. *Zdroj: [6]*

1.2.1 Plánování testů

Jedná se o soubor činností, pro které je nutné zjistit přesnou definici projektu. Dále co by se mělo kontrolovat a jak se budou vyhodnocovat nastalé situace. Sepsat testovací strategii, nalézt zdroje (finanční, kapacitní atd.) pro testy a přípravu testovacího plánu. Následně je nutné identifikovat a odhadnout pracnost všech testovacích požadavků.

Odhadování pracnosti je poměrně důležitá činnost v testovacím procesu. Pokud ve fázi plánování nezískáme na jednotlivé testy dostatečné zdroje, příprava a průběh testů budou nepříznivě ovlivněny. V krajním případě hrozí riziko, že nebudou splněny požadavky na testy, které je potřeba provést. Při odhadování pracnosti záleží na tom, co se v danou chvíli musí testovat. Zda se jedná o úplně nový produkt, nebo pokračování stávajícího produktu – například pouze nová verze stávajícího softwaru.

Pro nový projekt je zásadní zvolit vhodnou metriku. Nad malou reprezentativní částí produktu uskutečnit detailní odhad. Při tomto odhadu by se nemělo zapomenout na započítání času pro jednotlivá testovací kola a opakované testy (retesty) nalezených defektů. Existuje případ, kdy plánované aktivity jsou rozebírány ve velké úrovni detailu, což může vést k nadsazování odhadů. Tento efekt vzniká souhrou několika vlivů, mezi které se například řadí [1]:

- Zaokrouhlování: Pokud se daná aktivita má odhadovat s určitou přesností (například na dny nebo na hodiny), přirozeně je tento odhad raději zaokrouhlován nahoru.
- Rezerva: Strach z případných negativních scénářů vede k tomu, že je k daným aktivitám přidávána rezerva v podobě navýšení odhadu pracnosti.

Odhadování pracnosti v nízké úrovni detailu může vést k podcenění. Například malé množství času k podrobné analýze všech potřebných dílčích aktivit zapříčiní nedostatek informací ke správnému odhadu. Dále je potřeba identifikovat rizikové oblasti a separátně odhadnout čas přímo pro ně.

Pro odhad u stávajícího produktu se použijí stejné metriky jako v předchozí verzi produktu s tím, že známe reálné časy, které se získaly pomocí měření. Je nutné si vyhradit dostatečný čas na rizikové oblasti. Jestliže funguje správně odhad pracnosti z vývoje, lze dopočíst pouze konstantu pro odhad testů.

Kontrola jednotlivých odhadů se může provádět pomocí několika různých postupů. Výběr správného postupu závisí na velikosti vyvíjeného softwaru a jeho potencionální rizikovosti. Jednotlivé postupu je možné nalézt v následující tabulka 1.

Tabulka 1 - Postupy nezávislé kontroly odhadů. *Zdroj: [1]*

Postup	**Pracnost**	**Riziko zkreslení**
Zcela nezávislý odhad druhým specialistou (nemá k dispozici odhad prvního specialisty).	Velká.	Nejmenší (není příležitost k efektu ukotvení[2]).
Revize jednoho odhadu druhým specialistou – jiná metoda odhadování.	Střední.	Střední (efekt ukotvení, který se týká výsledku odhadu).
Revize jednoho odhadu druhým specialistou – stejná metoda.	Malá.	Velké (efekt ukotvení, který se týká výsledku odhadu i parametrů použité metody).
Revize odhadu za přítomnosti osoby, která dělala původní odhad.	Nejmenší (efektivita komunikace, možnost hned vyjasnit skutečnosti z revidovaného odhadu).	Největší (kromě efektu ukotvení, snaha obhájit svůj původní odhad, obava z negativní zpětné vazby, vymýšlení argumentů, co k původnímu odhadu vede, když to nemusí být relevantní nebo významné).

[2] Efekt ukotvení je jev, který poukazuje na psychologický experiment, který provedl Tversky a Kahneman v sedmdesátých letech. Více informací o tomto jevu lze dohledat v [1].

1.2.2 Analýza a příprava testů

Ve fázi analýzy testů se provádí návrh jednotlivých testovacích případů, které ověřují, zda jsou všechny požadavky na systém splněny. Doporučená délka jednoho testovacího případu je maximálně 20 kroků, protože při větším počtu kroků dochází k nepřehlednosti. Také dochází i ke ztrátě detailu, který bylo potřeba důsledně testovat. Daný případ musí být jednoznačně reprodukovatelný a musí obsahovat odpovídající míru detailu.

Již během přípravy testů je důležité, aby byla identifikována a připravena potřebná testovací data. Jelikož už i v této fázi jsme schopni, podle specifikace, zajistit správné sestrojení příslušných testovacích dat. Pokud by se v této fázi příprava testovacích dat přeskočila, mělo by to za následky nesprávné provedení a vyhodnocení testovacích případů. V nejhorším případě by nastalo velké časové zdržení, jelikož by se testovací data musela připravit až v následující fázi. Tam už je postupně ztrácen přehled o funkční specifikaci vyvíjeného softwaru a testeři se řídí pouze podle připravených testovacích případů.

Rovněž je nutné specifikovat jednotlivé testovací techniky, které budou využity během samotného testování. Mezi tyto techniky lze zařadit například:

- **Smoke testy** – Tato testovací technika se využívá v okamžiku, kdy je dokončen vývoj aplikace a je možné ji spustit. Jde o krátký test, který slouží jako rychlé a okamžité ověření, zda je vyvíjená aplikace připravena pro další fázi testování. Obvykle se provádí jednoduché ověření všech částí aplikace, zda jdou implementovány, nainstalovány a spuštěny. [11]
- **Funkční testy** – Tyto testy se dají zařadit do kategorie systémových testů. Ověřují, že aplikace správně plní všechny úkoly, pro které je navržena. Obecně se testují všechny funkce, které jsou v aplikaci implementovány. Zároveň se kontroluje, zda odpovídají požadavkům zákazníka. [12]

- **Integrační testy** – U těchto testů je nutné rozlišovat mezi vnitřními testy, které spočívají v testování propojení jednotlivých částí vyvíjené aplikace a vnějšími testy, kde se testuje integrace jednotlivých softwarů do větších provozuschopných oddílů. Integrační testy se tedy zaměřují na přímou komunikaci jednotlivých částí (modulů). Hlavním úkolem těchto testů je ověření, že modul umí odesílat a přijímat komunikaci s dalšími moduly. [13]
- **Regresní testy** – Tyto testy jsou velice důležité, jelikož mají za úkol ověřit, zda nové zásahy do aplikace nenarušily správné chování těch částí aplikace, které těmito zásahy neměly být ovlivněny. To znamená, že je testováno to, že oprava nějaké chyby nebo přidání nové funkčnosti aplikace nezpůsobí nový defekt v již vytvořených a funkčních částech aplikace. [13]
- **Zátěžové testy** – Cílem tohoto typu testování je prověřit vlastnosti aplikace, zda je stabilní během určitého zatížení. Toto zatížení lze simulovat například velkým počtem uživatelů, kteří využívají daný software ve stejnou dobu či spuštění několika procesů v aplikaci současně. Výsledkem těchto testů by mělo být ověření, zda jednotlivé testy nemají vliv na výkon systému. [14]

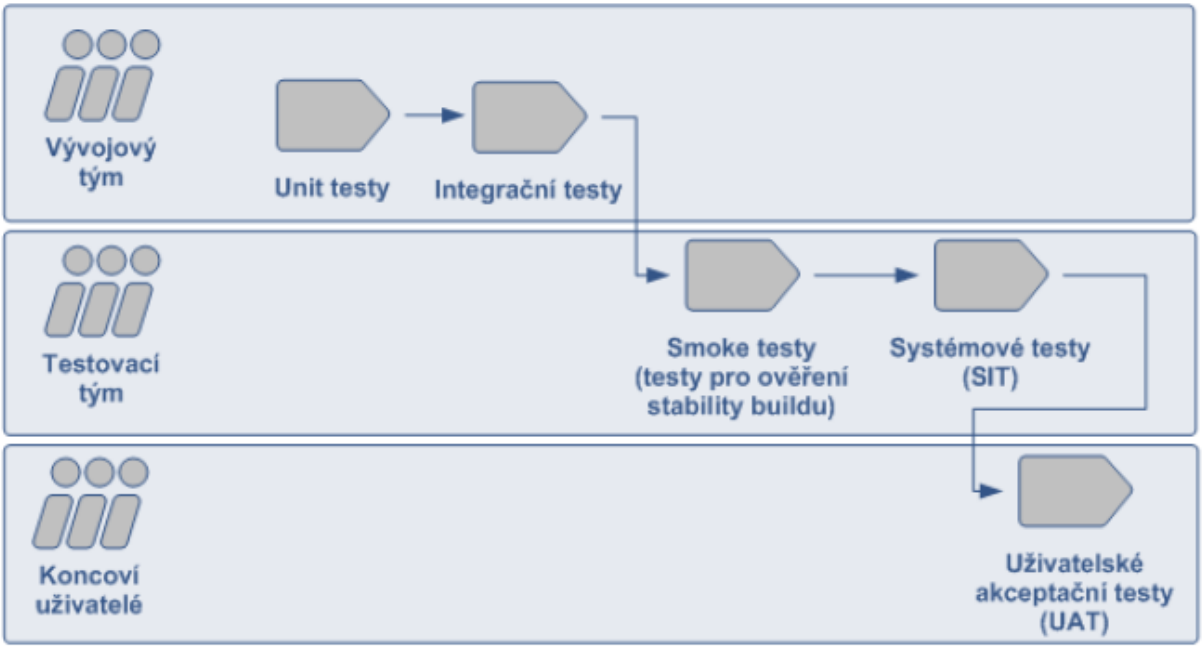
V tomto kroku je nutné dávat pozor, zda se provádí analýza a příprava testů již nad finální revidovanou vstupní dokumentací. Pokud tomu tak není, hrozí, že vytvořené testovací případy nemusí odpovídat konečným požadavkům od klienta. Dalším nebezpečím je malá diverzita použitých technik. Pokud není použito dostatečné množství testovacích technik, může například nastat případ, kdy aplikace je sice plně funkční, ale při přihlášení více uživatelů dojde k jejímu výpadku. To znamená, že testovací tým zvolil pouze funkční testování aplikace a nevěnoval dostatečnou pozornost zátěžovým testům. Další hrozbou je příliš detailní testovací případ. To může zapříčinit obtížnou udržovatelnost jednotlivých případů, malou volnost pro kreativitu testera anebo malý prostor pro „náhodu“. [6]

1.2.3 Provedení a vyhodnocení testů

Jedná se o soubor činností, které vedou ke správnému provedení a vyhodnocení testů. Jak již bylo řečeno, samotné provádění testů je předem naplánováno v testovací strategii na určitou dobu. Tato doba většinou okamžitě navazuje na dokončení implementace samotného softwaru, ale reálně jsou tyto odhady pouze orientační. Ve firmě Unicorn Systems začíná testování už během částečné implementace softwaru, a to z několika důvodů.

Jedním z důvodů je pomoc vývojovému týmu k odstranění základních chyb, které by mohly narušit celý testovací běh, a tím i časovou osu vyvíjeného softwaru. V tomto případě se tedy testuje „volně“ podle zkušeností testera s danou aplikací. Počet osob, které testují, je minimální (většinou pouze jedna osoba). Zároveň má tester nastudovanou vstupní dokumentaci a může testovat i nové implementační řešení. Již ze zkušenosti je zjištěno, že většina nových implementací má v sobě skrytou chybu, která je sice snadná na opravu, ale způsobuje například nefunkčnost důležitých částí aplikace. Pokud by se tato chyba našla až během prvního kola testování, čas na opravu a opětovné nasazení aplikace by způsobil, že testovací tým by nemohl v tuto dobu pracovat. Dalším důvodem jsou náklady na opravu chyby. Jak již bylo zmíněno, čím dříve se daná chyba nalezne, tím levnější je její oprava.

Existují různé typy testů a zároveň jednotlivé typy vykonávají odpovědné týmy, nebo uživatelé. Obrázek 5 zobrazuje rozdělení jednotlivých testů ve společnosti Unicorn Systems a kdo tyto testy vykonává. Detailnější popis jednotlivých testů a jejich význam lze dohledat v [3].



Obrázek 5 - Odpovědnost za jednotlivé testy. *Zdroj: [10]*

Z hlediska tématu této práce jsou pro provádění a vyhodnocování testů nejdůležitější smoke testy a funkční testy (řadí se mezi systémové testy). Pro první zmíněné testy je optimální využít automatizované testy. Ty simulují reálné chování testera a ověřují, zda nasazená aplikace je ve stabilním stavu. Dále kontrolují základní funkčnosti, které byly již dříve implementovány, jestli fungují správně. Pokud jsou využity právě automatizované testy, dochází u nich k automatickému vyhodnocování podle parametrů, které jsou zadané. Samotné vyhodnocení je následně automaticky propisováno k příslušným testovacím případům do nástroje, který se stará o sledování defektů u individuálních testovacích případů.

Systémové testy se provádějí manuálně, jelikož se ve většině případů jedná o nové funkcionality softwaru. Jednotlivé testy se provádějí pomocí kroků, které jsou obsaženy u každého testovacího případu. Tester sám vyhodnocuje každý krok a následně zapisuje výsledný stav do nástroje, který se stará o sledování defektů u individuálních testovacích případů.

1.2.4 Sledování a správa defektů

Už bylo charakterizováno, co je chyba neboli defekt. Nyní se podíváme na to, jak defekt nahlásit, vyhodnotit jeho prioritu, řídit a sledovat.

Z pohledu řízení testování je nutné, aby bylo nahlášení defektů co nejvíce vyvážené ze dvou pohledů [1]:

- Tester by neměl strávit příliš mnoho času při nahlašování defektu. Podle dosavadních zkušeností je optimální doba do 1 minuty, samozřejmě záleží na složitosti zadávané chyby.
- Zadaný defekt musí být úplný a srozumitelný, jelikož je potřeba s defektem dále pracovat a případně jej opravit.

V této situaci mohou pomoci nástroje pro správu defektů. Ty umožňují efektivně a jednoduše zadat nalezenou chybu. Podle dosažených zkušeností můžeme říci, že optimální popis defektů se skládá z hlavních částí. První částí je stručné a jednoznačné pojmenování chyby, dále v bodech sepsaný postup testování a třetí částí je aktuálně zjištěný výsledek a očekávaný výsledek. Tyto jednotlivé body lze pro kvalitní nahlášení defektu doplnit o informace, které usnadňují opravu a simulaci chyby. Mezi ně lze zařadit logovací informace[3], snímky obrazovky, které zobrazují popisovanou chybu nebo použitá data během testovacího postupu. Pokud se během testování využívá více prostředí s různými verzemi aplikace, je nutné zaznamenat na jakém prostředí, a na jaké verzi aplikace se daná chyba vyskytuje. Jestliže tester zapomene některé důležité informace zadat – ať už logy, snímky obrazovky nebo přesný popis, jak defekt reprodukovat, znamená to, že tyto nekvalitně zadané defekty vedou k prodloužení opravy, případně k zbytečným pracím.

[3] Informace o jednotlivých činnostech v daném prostředí, které se automaticky zaznamenávají do patřičného souboru.

Důsledek chybně zadaného defektu může být – kromě nutného času na vyjasnění – i nesprávně reportovaný stav projektu. To znamená buď falešný klid (pokud jsou chyby nahlášeny jako marginální a nedůležité), nebo falešná hysterie (pokud jsou naopak všechny defekty nahlášeny jako kritické).

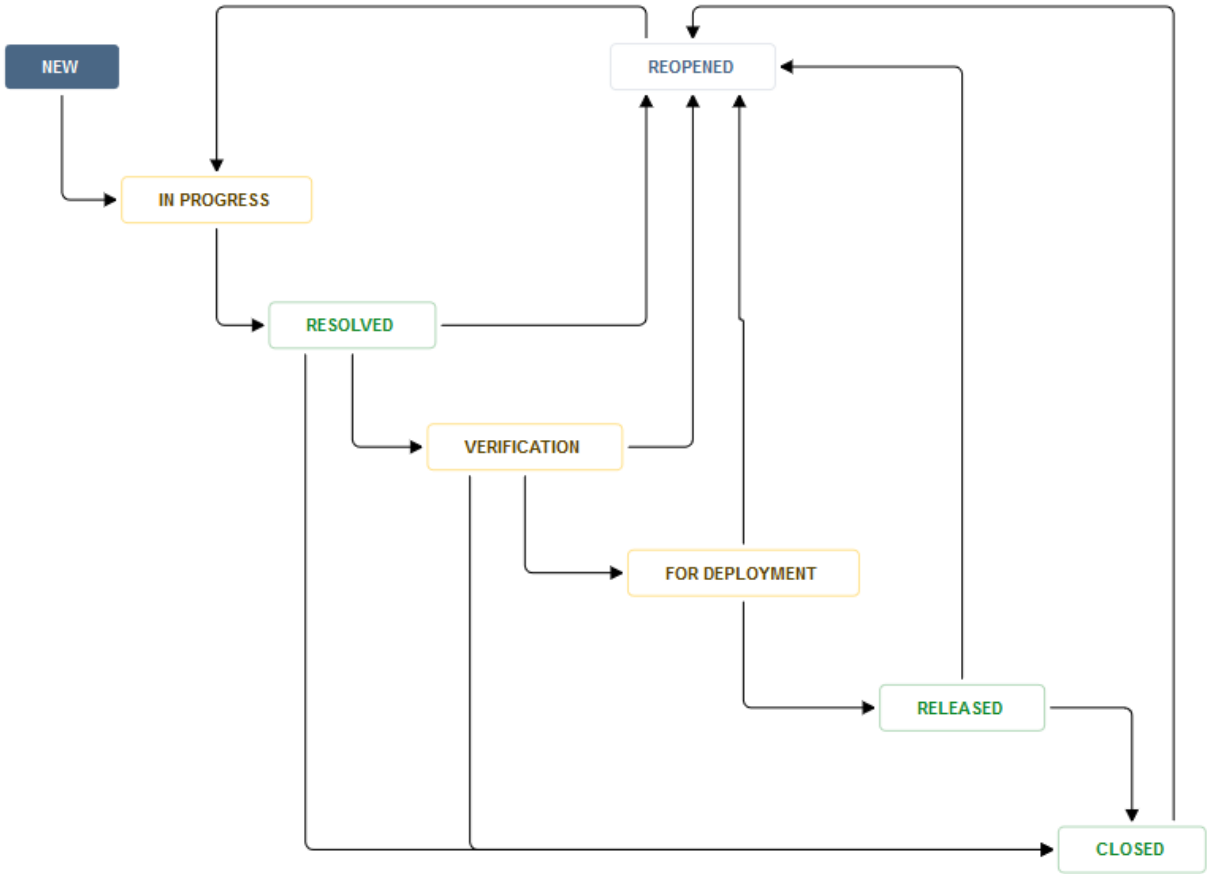
V tomto bodě se dostáváme do situace, kdy musíme správně určit závažnost chyby. Veškeré informace o zadávání defektů je možné dohledat v interním dokumentaci [15] společnosti Unicorn Systems. Jednu z možností pro určování závažnosti chyb vyjadřuje následující škála [15]:

- Priorita A1 (Blocker), přičemž tato chyba má nejvyšší prioritu a znamená, že systém nebo jeho část zcela nefunguje, a tudíž se jedná o nejzávažnější chybu, kterou je nutné co nejdříve vyřešit.
- Priorita A2 (Critical) znamená, že nefunguje nebo zcela chybí určitá funkčnost systému a není nastaven žádný postup, jak tuto chybu obejít tzv. workaround.
- Priorita B (Major) značí, že se vyskytla chyba, na kterou lze použít workaround.
- Priorita C (Minor), tato priorita neznamená kritickou chybu, ale jedná se spíše o drobné chyby způsobené například překlepem ve slově, špatným formátem textu nebo rozdílně zarovnaná data v tabulce. Tyto chyby jsou řešeny jako poslední, jelikož se ve většině případů jedná pouze o „kosmetické chyby“.

Každý defekt má svůj životní cyklus. Stavy tohoto cyklu defektu a přechody mezi nimi vystihuje anglický termín workflow. Je nutné, aby v každém okamžiku byl kdokoliv schopen určit nejen stav konkrétního defektu, ale je také potřeba mít možnost dohledat předchozí stavy a kdo jednotlivé stavy měnil. Dále je důležité nalézt kolik defektů je v konkrétním stavu a kdo je odpovědný za daný defekt.

Příliš velký počet stavů v životním cyklu sice dokáže zacílit i na ty nejdetailnější rozdíly ve stavech defektů, ale správa chyb bude nepřehledná. S jednotlivými defekty se bude špatně pracovat. V konečném důsledku může právě složitá administrativa spojená se správnou změnou stavů defektů vést k prodražení oprav.

Stavy a jejich povolené přechody bývají většinou přednastaveny v nástroji pro správu chyb. Vyspělejší nástroje dovolují životní cyklus defektu modifikovat. Příklad konkrétního životního cyklu ilustruje následující obrázek 6, jenž je pořízen z výstupu interního informačního systému společnosti Unicorn Systems. Tento příklad životního cyklu na projektu NUCS lze zjednodušit i rozšířit. Například možnost rozšířit dané workflow lze v případě, kdy je nejprve nutné schválit nově zadanou chybu. Pro tento případ by bylo potřeba přidat nový stav mezi stavy „New“ a „In progress“, který by charakterizoval schválení zadaného defektu nebo naopak, přímé zamítnutí. Jednotlivé šipky potom charakterizují přechody z jednoho stavu do druhého, podle toho, jak je s daným defektem pracováno.



Obrázek 6 - Životní cyklu defektu na projektu NUCS. *Zdroj: Autor s využitím výstupu informačního systému.*

Jak již bylo řečeno, tento konkrétní životní cyklus je z projektu NUCS. Každý stav je zde z určitého důvodu. Pokud je chyba ve stavu „New“ znamená to, že tento defekt byl nalezen poprvé a podle zadané priority by měl být vyřešen. V tuto chvíli přichází

na řadu analýza zadané chyby a stav se změní na „In progress“ tzn., že se na dané chybě pracuje. Pokud osoba k tomu určená rozhodne, že se skutečně jedná o chybu a priorita dané chyby je správná, daný defekt se přidělí pracovníkovi, který tuto chybu opraví. Jestliže ale rozhodne o tom, že se systém chová podle specifikace, přepne stav do „Vyřešen – Resolved“ a pak upozorní zadavatele na vyřešení chyby. Ten tuto chybu může zavřít s tím, že se skutečně o chybu nejedná nebo ji znova otevřít s tím, že analýza zadané chyby proběhla špatně, tudíž se jedná o chybu. Tím se chyba dostává do stavu „Reopened“, a následně probíhá předchozí postup znovu.

Pokud je chyba opravena, přechází do stavu „Resolved“, následně přidělena osobě, která je odpovědná za správné propojení opravy a ostatního kódu aplikace. Jestliže je oprava nasazena na testovací prostředí, opět se změní stav. Nyní je defekt ve stavu „Verification“. To znamená, že je chyba přiřazena na testovací tým, který provede ověření na testovacím prostředí, zda oprava defektu funguje správně. Pokud nefunguje správně, defekt se může znovu otevřít a celý postup se opakuje. Jestliže oprava funguje správně, je stav chyby změněn na „For deployment“, kde čeká na nasazení na testovací prostředí zákazníka. Pokud je defekt nasazen na testovací prostředí zákazníka, dostává stav „Released“ a zákazník je upozorněn, že oprava defektu je nasazena a může sám ověřit, zda je oprava vyřešena správně nebo nikoliv. Pokud je vyřešena správně, stav defektu se naposledy změní. Nový stav defektu je „Closed“, čímž se dává na vědomí, že celý životní cyklus defektu je u konce. Jestliže oprava není vyřešena z pohledu zákazníka správně, může opět chybu otevřít a celý postup se opakuje. V krajních případech může nastat, že již vyřešená chyba se opět projeví. Z tohoto důvodu lze přejít ze stavu „Closed“ na „Reopened“ a řešit nastalý defekt znovu.

Při vytváření workflow by měl mít každý na paměti kontext konkrétního projektu a podle toho zjednodušit nebo rozšířit stavy defektů. Životní cyklus defektu by měl být odsouhlasen se zástupci vývoje, s analytiky, zadavatelem, i s projektovým

managementem. Jakmile je zvolený životní cyklus jednou zaveden, je dobré mít jej po celou dobu trvání projektu zafixovaný.

1.2.5 Akceptace a uzavření testování

Už v rané fázi je důležité mít specifikované pokrytí testovacích případů nad testovaným systémem. Zároveň je nutné věnovat pozornost plánovaní budoucích reportů. Z tohoto důvodu je potřeba nastavit testovací nástroje nebo metodiky jejich používání tak, abychom z nich dostali potřebná data.

Záleží na dohodě se zákazníkem, v jaké formě mu budou sděleny veškeré informace o testovaní nad daným systémem. Dle dosavadní zkušenosti, je pro zákazníka standardem, dostat veškeré informace v těchto dvou dokumentech:

- Release Notes,
- Test Report.

Release Notes

Release Notes je komunikační dokument, který je sdílený se zákazníky a klienty organizace. Uvádějí změny nebo vylepšení funkcí služeb nebo produktu, který společnost poskytuje. Tento dokument je tedy obvykle rozeslán pouze po důkladném testování a schválení produktu nebo služby podle specifikace poskytnuté vývojovým týmem. [16]

Release Notes mají striktní strukturu, která je charakterizována v následujících bodech:

- hlavička dokumentu,
- výstupy,
- přehled opravených defektů,
- přehled zamítnutých defektů,
- přehled nevyřešených defektů.

V tomto dokumentu musí být charakterizována hlavička. Ta obsahuje informace o produktu, nasazované verzi a datu nasazení této verze. Dále je zde uvedena URL adresa, která informuje o tom, kam je daná verze nasazována. Poslední součástí hlavičky je účel, za jakým je daná verze nasazována. Součástí Release Notes je kapitola, která obsahuje informace, které jsou dodávány spolu s nasazenou verzí. Například informace o tom, že součástí této verze je i administrační návod, návod k instalaci apod. Veškeré tyto jednotlivé body lze nalézt na reprezentativním obrázku 7, který je pořízen z výstupu interního informačního systému firmy US.

Header

Product	NUCS (Nordic Unavailability Collection System)
Version	NUCS 1.1.0.744
Release Date	07.12.2018
URL	Installed on UAT Environment [redacted]
Purpose	NUCS 1.1 PROD RC - Release Candidate for PROD from last week of UAT.

Deliverables

Following deliverables **were delivered** in the package:

- NUCS 1.1.0.744 deployed on UAT environment
- NUCS 1.1.0.744 - Release Notes.pdf
- NUCS 1.1.0.744 - Test Report

Following deliverables **were NOT delivered** in the package:

- Merge of Change Requests Specifications into project documentation
- NUCS - Administration Guide_GUI Guideline
- NUCS - Installation Guide

Obrázek 7 - Ukázka hlavičky v Release Notes. *Zdroj: Autor s využitím výstupu informačního systému.*

Další kapitolou v Release Notes je přehled defektů, které jsou opraveny a dodány, jejich typ, identifikátor, jednoduchý popis chyby a zadaná priorita. V této kapitole se mohou vyskytovat i informace o defektech, které byly zamítnuty, a tím pádem se

o chyby nejednalo. Příklad takto dodaných opravených a neopravených chyb je možné shlédnout na obrázku 8, který je pořízen z výstupu interního informačního systému firmy US.

Content of Delivery

Issue Type	Key	Summary	Priority
Bug	NUCS-419	Multiple Parallel UMM - wrong computation of individual xml	Blocker
Bug	NUCS-432	Missing Values after Synchronization run	Critical
Bug	NUCS-433	NTC Publication to ETP	Critical
Bug	NUCS-426	Internal Links are rejected for NTC computation	Major
Bug	NUCS-411	Provider filter/search not working correctly	Minor
Bug	NUCS-412	Number of messages per page is not consistent	Minor
Change Request	NUCS-303	"Enter" should not publish an unavailability	Major
Task	NUCS-434	Missing borders	Blocker

Won´t Fix

Issue Type	Key	Summary	Priority
Bug	NUCS-286	Filtering by market participant	Major
Bug	NUCS-414	Too many borders / areas visible	Major
Bug	NUCS-422	XML, CSV and Excel export shows wrong UMM dates	Major

Obrázek 8 - Přehled dodaných defektů. *Zdroj: Autor s využitím výstupu informačního systému.*

V poslední části Release Notes jsou informace o zbývajících nevyřešených defektech a limitující informace, které určitým způsobem ovlivňují nasazení aplikace. Podle těchto dodaných informací se porovnávají výsledky s akceptačními kritérii, které jsou definovány vždy na začátku projektu.

Test Report

Druhým dokumentem je Test Report, který už nemusí mít tak striktní strukturu. Jednotlivé části tohoto dokumentu jsou upravovány podle informací, které je nutné předat zákazníkovi. Pro konkrétní projekt NUCS je struktura následující:

- sloupec Test, který značí ID testovaného případu,
- sloupec Name, který popisuje testovací případy,

- sloupec Last Run, který znázorňuje datum a čas, kdy byl daný testovací případ testován,
- sloupec Run Status, který udává stav daného testovacího případu,
- sloupec JIRA Issue ID, který zobrazuje ID nalezeného defektu v externím nástroji.

V tomto dokumentu jsou sepsány všechny testovací případy. Jednotlivé případy musí pokrývat celistvou aplikaci a nevyřešené defekty by se měli odrážet ve výsledcích těchto případů. Pokud se objeví v daném testovacím případě hodnota „Passed“, je tento případ v pořádku a daná část aplikace funguje dle specifikace. Jestliže se objeví testovací případ s hodnotou „Failed“ znamená to, že během testování určitá část nefunguje dle specifikace. Tento testovací případ je posléze v testovacím reportu doplněn o identifikátor chyby, který je zákazník schopen dohledat v Release Notes, případně v nástroji pro správu chyb. Test Report s konkrétními testovacími případy a jejich výsledky je možné nalézt na obrázku 9.

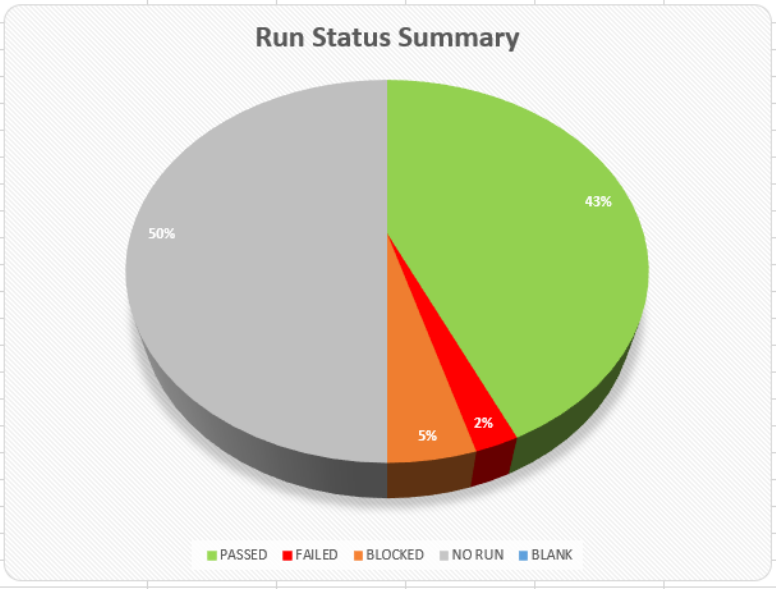
Test:	Name	Last Run	Run Status	JIRA Issue ID
141	15.1.A - Migration - Planned Unavailability of Generation Units	07-Dec-2018 16:08	PASSED	
142	15.1.B - Migration - Changes in Actual Availability of Generation Units	07-Dec-2018 16:08	PASSED	
143	15.1.C - Migration - Planned Unavailability of Production Units	07-Dec-2018 16:08	PASSED	
144	15.1.D - Migration - Changes in Actual Availability of Production Units	07-Dec-2018 16:08	PASSED	
145	10.1.A - Migration - Planned Unavailability In The Transmission Grid	07-Dec-2018 16:08	PASSED	
146	10.1.B - Migration - Changes in Actual Availability in the Transmission Grid	07-Dec-2018 16:08	PASSED	
147	7.1.A - Migration - Planned Unavailability of Consumption Units	07-Dec-2018 16:08	PASSED	
148	7.1.B - Migration - Changes in Actual Availability of Consumption Units	07-Dec-2018 16:08	PASSED	
149	10.1.C - Migration - Changes in Actual Availability of Off-shore Grid Infrastructure	07-Dec-2018 16:08	PASSED	
150	10.1.C - Replication - Changes in Actual Availability of Off-shore Grid Infrastructure	07-Dec-2018 16:08	PASSED	
151	7.1.B - Replication - Changes in Actual Availability of Consumption Units	07-Dec-2018 16:08	PASSED	
152	7.1.A - Replication - Planned Unavailability of Consumption Units	07-Dec-2018 16:43	FAILED	NUCS-435
153	10.1.B - Replication - Changes in Actual Availability in the Transmission Grid	07-Dec-2018 16:08	PASSED	
154	10.1.A - Replication - Planned Unavailability In The Transmission Grid	07-Dec-2018 16:08	PASSED	
155	15.1.A - Replication - Planned Unavailability of Generation Units	07-Dec-2018 16:08	PASSED	
156	15.1.B - Replication - Changes in Actual Availability of Generation Units	07-Dec-2018 16:08	PASSED	
157	15.1.C - Replication - Planned Unavailability of Production Units	07-Dec-2018 16:08	PASSED	
158	15.1.D - Replication - Changes in Actual Availability of Production Units	07-Dec-2018 16:08	PASSED	
159	Transmission Grid Unavailability with Internal Area and External Area	07-Dec-2018 16:07	BLOCKED	NUCS-355
160	Transmission Grid Unavailability without Transmission Assets		NO RUN	
161	Unavailability Message with more Reason Texts A95		NO RUN	
162	Halving of Time Intervals - 200 documents limit		NO RUN	

Obrázek 9 - Testovací případy v Test Reportu. *Zdroj: Autor*

Součástí Test Reportu bývá i popis, kdo tento dokument vytvořil a co obsahuje. Rovněž obsahuje zobrazení všech výsledků v podobě grafu. Takovýto graf znázorňuje dosažené stavy jednotlivých testovacích případů, což umožnuje lepší přehled o kvalitě dodávaného systému. To je možné vidět i na obrázku 10, který znázorňuje ucelený pohled na jedno testovací kolo u vyvíjené aplikace NUCS.

Title	NUCS 1.1 - UAT Test Report - Increment
Entity Type	Instance
Report Type	Tabular
View	NUCS 1.1. - UAT NEW
Created by	Sebastian Vacek
Created at	07-Dec-2018 16:53 (UTC)

Run Status	Count
PASSED	19
FAILED	1
BLOCKED	2
NO RUN	22
BLANK	0



Obrázek 10 - Graf v Test Reportu. *Zdroj: Autor*

Takto zpracované a vyplněné dokumenty jsou předány zákazníkovi. Ten je povinen tyto dokumenty schválit, nebo vrátit s připomínkami.

1.3 Rozdělení testů

Existuje několik způsobů, jak testy dělit. Ať už podle způsobu testování (White box, Grey box nebo Black box testování) nebo podle doby, kdy a kdo aplikaci testuje (Akceptační testy, systémové testy apod.). Veškerá tato rozdělení lze případně dohledat v bakalářské práci [3]. V rámci této diplomové práce se zaměříme na rozdělení testů podle toho, kdo tyto testy vykonává. To znamená, že testy rozdělíme na manuální nebo automatizované.

Manuální testování

Manuální testování je pro někoho pouze „klikání“ do aplikace. Toto tvrzení není zcela správné. Za manuálním testováním stojí hodně práce. Samotné testování provádí

přímo tester a využívá k tomu veškeré své dosavadní zkušenosti. Postupuje podle zadaného scénáře a vyhodnocuje testy podle předpokládaných výsledků. Nicméně během manuálního testování hraje velkou roli zkušenost a kreativnost testera. Díky těmto dvěma aspektům se přichází na spoustu defektů, které by zadané scénáře neodhalily. [17]

Zároveň veškeré vyhodnocování je prováděno rovněž manuálně. Tester se díky tomu dokáže zaměřit jak na cílený výsledek, tak je zároveň schopný si všimnout drobných chyb, které vyplynou během testování daného scénáře na povrch.

Manuální testování zabírá hodně času, a to hned z několika důvodů. Prvním důvodem může být situace, kdy tester přijde k testování až těsně před začátkem testovacího kola a nemá možnost se důsledně seznámit s danou aplikací. Samozřejmě by se měl řídit podle navržených scénářů, ale i ty mohou být příliš obecně napsané a nemusí být jasné, jak má vypadat správný výsledek. Dalším důvodem je lidský aspekt, člověk nedokáže kontrolovat všechny informace okamžitě, musí se na každou informaci soustředit zvlášť, a zároveň není schopen „klikat" tak rychle jako automatizovaný test.

Automatizované testování

Automatizované testování softwaru má hodně podob a v dnešní době je stále více využíváno. Hlavním cílem je časová úspora při opakovaném spouštění jednotlivých testů. Obecně se dá říci, že automatizace testů softwaru má za účel zefektivnit prováděný postup, uspořit prostředky a samozřejmě zvýšit kvalitu výsledného produktu. Pokud jsou testy vykonávány plně automaticky, tím je myšleno zcela bez zásahu lidského faktoru, výrazně se tím snižuje možnost chybného provedení určitého scénáře, a tím také pravděpodobnost bezporuchového provozu softwaru. [18]

Jedná o testy, které jsou prováděny pomocí specializovaných softwarových nástrojů. Ty simulují reálné používání vyvíjené aplikace, zároveň dokáží napodobit obrovské množství uživatelů. Dá se předpokládat, že uživatel používá stejné postupy (scénáře) během skutečného užívání aplikace. Proto je vhodné automatizovat ty testy, které je

možné ověřovat opakovaně. To znamená, že jsou především neměnné a pokrývají části aplikace, které se často nemění. Z tohoto důvodu se nejvíce automatizují Smoke testy, Regresní testy, ale také například Nefunkční testy. Těmito testy se snažíme pokrýt co možná největší část vyvíjené aplikace. U aplikací, které mají krátkou životnost (v řádech měsíců, jedna verze) se nám nevyplatí vytvářet tyto testy. Naopak u aplikací, kde je předpokládáno s delší životností (řádově roky, více verzí), tak tam se zcela jistě realizace automatizovaných testů vyplatí. Nesmíme zapomínat, že automatizované testy se nemusejí využívat jenom u vývojového týmu, ale lze tyto testy převést také do servisního týmu, který má za úkol udržovat a opravovat produkční verze aplikace. Při realizaci testů se však musí přihlédnout k tomu, zda část aplikace nebude často upravována. Jelikož by to znamenalo častou úpravu automatizovaných testů, a tím také spojené zvyšování nákladů na testování. [19]

1.3.1 Využití v praxi

V praxi se nejvíce setkáváme s kombinací obou metod testování. Tato kombinace je zcela logickým řešením. Pokud uvažujeme, že pro danou aplikaci se vyplatí vytvořit automatizované testy, tak ušetření času v budoucích kolech testování znamená, že celkové náklady na testování budou menší, než kdyby se počítalo pouze s manuálními testy. Zároveň není vždy možné, aby veškeré funkčnosti aplikace byly pokryty automatizovanými testy na 100 %. Během testování by měl hrát roli i lidský faktor, jelikož kombinací obou metod je dosaženo nejvyšší kvality vyvíjeného softwaru s minimálními náklady a časem na testování. Tato kapitola je zpracována za pomoci [19] až [21].

Využití manuálního testování

Manuální testování přichází v úvahu v okamžiku, kdy je nutné průzkumné (náhodné) testování. Například při práci s ne zcela podrobně napsanou dokumentací nebo při nedostatku času. V tomto typu testování je zapotřebí zkušenost testera s kvalitní

znalostí domény spolu se silnými analytickými dovednostmi, kreativitou a intuicí. Tento přístup vyžaduje minimální plánování a maximální provádění testů.

Další možné využití manuálního testování je při testování použitelnosti. Toto testování pomáhá zjistit uživatelskou přívětivost aplikace. Cílem tohoto testování je zjistit spokojenost koncového uživatele s aplikací, proto je zde lidské pozorování je důležité. Ruční přístup je pro toto testování ideálním řešením.

Ad-hoc testování je taktéž prováděno pomocí manuálních testů. Tento typ testování se provádí bez jakéhokoliv plánování či dokumentace. Testování se neřídí žádným strukturovaným způsobem testování a provádí se náhodně na kterékoliv části aplikace. Zároveň závisí především na testerových znalostech o vyvíjené aplikaci. Hlavním cílem tohoto testování je nalezení závad náhodnou kontrolou. Ad-hoc testování může být provedeno v jakémkoliv okamžiku, ať už je to začátek, střed nebo konec testování projektu. Může je využíváno, pokud je čas na testování velmi omezený a zároveň je potřeba velmi podrobné ověření aplikace. [21]

Doporučení: Podle dosavadních zkušeností je nejlepší volbou využít ad-hoc testování na konci testovacích kol. Jelikož tester má detailnější přehled o testované aplikaci a zároveň může využít své zkušenosti k nalezení různých druhů defektů.

Podle [20] lze mezi výhody manuálního testování zařadit získávání rychlejší a přesnější vizuální zpětné vazby. Další výhodou je lidský úsudek a intuice, ty vždy přinášejí užitek během testování. Při testování malé změny by automatizované testy vyžadovaly kódování, které by mohlo být časově náročné, zatímco manuální testování lze provádět ihned.

Mezi nevýhody ručního testování je možné zařadit i menší spolehlivost, která se váže na testy vedené člověkem. Ten může způsobovat vlastní chyby během testování. Manuální test se ve většině případů nezaznamenává, to způsobuje, že ho nelze znovu použít.

Využití automatizovaného testování

Automatizované testování se velmi často využívá během regresního testování. Je vhodné, jestliže dochází k častým změnám kódu, které vyžadují časté testování. Toto testování se provádí tak, aby se zajistilo, že nové změny kódu, které jsou implementovány, neovlivní exitující funkce systému.

Testování zatížení je rovněž testováno pomocí této metody. To pomáhá určit výkon systému v podmínkách zatížení v reálním produkčním stavu. Toto testování zjišťuje, jak se systém chová, když k němu přistupuje více uživatelů současně.

Testování výkonu pomáhá zjistit rychlost, škálovatelnost a stabilitu aplikace. Cílem je odstranit případné nedostatky ve výkonu. To vyžaduje simulaci tisíců souběžných uživatelů. Automatizované testování je proto upřednostňováno pro tento typ testů.

Jak uvádí [20], mezi výhody automatizovaných testů lze zařadit rychlý a efektivní proces. Pomáhají najít další chyby v porovnání s lidským testerem. Probíhá pomocí softwarových nástrojů, takže pracuje bez únavy, na rozdíl od lidí při manuálním testování. Může snadno zvýšit produktivitu, protože poskytuje rychlý a přesný výsledek testování. Podporuje různé aplikace a lze spouštět více testů paralelně.

Naopak, mezi nevýhody uvádí [20] tyto skutečnosti. Bez lidského prvku je obtížné získat přehled o vizuálních aspektech vyvíjeného uživatelského rozhraní jako jsou barvy, písmo, rozměry, kontrast nebo velikost tlačítek. Nástroje pro běh automatizovaného testování mohou být drahé, což může zvýšit náklady na testovací fázi. Každý automatizovaný nástroj má své omezení, které snižuje rozsah automatizace. Ladění testovacího skriptu je dalším závažným problémem, jelikož údržba jednotlivých testů je velmi nákladná.

Jak uvádí [22], pokud jsou stále opakovány tytéž testy, je velice pravděpodobné, že časem, podle dosavadních testovacích případů, nebudou nalezeny žádné nové defekty. Tento jev je nazýván: „Pesticidní paradox“. K překonání tohoto jevu je potřeba, aby byly existující testovací případy pravidelně revidovány a upravovány.

Proto je žádoucí, aby byly napsány nové a odlišné testy pro případné odhalení dalších defektů. Následující tabulka 2 a tabulka 3 detailněji zobrazují rozdíly mezi manuálním a automatizovaným testováním.

Tabulka 2 - Rozdíly manuálního a automatizovaného testování – část I. *Zdroj: [20]*

Parametr	Manuální testování	Automatizované testování
Definice	Testovací případy jsou prováděny člověkem a softwarem.	Testovací případy jsou prováděny pomocí automatizačních nástrojů.
Doba zpracování	Ruční testování je časově náročné a zabírá lidské zdroje.	Je podstatně rychlejší než manuální přístup.
Průzkumné testování	Umožňuje náhodné testování.	Neumožňuje náhodné testování.
Počáteční investice	Počáteční investice do manuálního testování je poměrně nižší. ROI je v porovnání s automatizovaným testováním v dlouhodobém horizontu nižší.	Počáteční investice do automatizovaného testování je vyšší. Nicméně, ROI je dlouhodobém horizontu lepší.
Spolehlivost	Není tak spolehlivou metodou, kvůli možnosti lidských chyb.	Je spolehlivou metodou, protože se provádí pomocí nástrojů a skriptů.
Změna uživatelského rozhraní	Malé změny, jako je například změna v id, třídě atd., by neměly znemožnit provedení ručního testu.	Pro dokonce i triviální změny uživatelského rozhraní je nutné upravit automatizované testovací skripty tak, aby fungovaly podle očekávání.
Investice	Investice jsou potřebné pro lidské zdroje.	Investice je nutná pro zkušební nástroje (pokud není zdarma) i pro lidské zdroje.
Nákladově efektivní	Není nákladově efektivní pro regresi s vysokým objemem.	Není nákladově efektivní pro regresi s malým objemem.
Zobrazení výsledků testů	Jsou obvykle zaznamenávány v aplikaci Excel nebo Word a výsledky testů nejsou snadno dostupné.	Všechny zúčastněné strany se mohou přihlásit do systému a zkontrolovat výsledky testování.

Tabulka 3 - Rozdíly manuálního a automatizovaného testování – část II. *Zdroj: [20]*

Parametr	Manuální testování	Automatizované testování
Lidské pozorování	Umožňuje lidské pozorování, což může být užitečné při nabízení uživatelsky přívětivého systému.	Nezahrnuje lidskou úvahu, takže nikdy nemůže poskytnout jistotu uživatelské přívětivosti a pozitivní zkušenosti zákazníků.
Testování výkonu	Testování výkonu není proveditelné ručně.	Testy výkonu, jako zátěžové testování, zkoušení ve špičce atd. Musí být testováno pomocí automatizovaného nástroje.
Paralelní spouštění	Mohou být prováděny paralelně, ale je třeba zvýšit lidský zdroje, což je drahé.	Toto testování může být provedeno na různých operačních platformách paralelně a zkracovat dobu provádění testů.
Programovací znalosti	Není třeba programování při ručním testování.	Programovací znalosti jsou nezbytným předpokladem.
Opakované testování	Opakované ruční provádění testů může být nudné a náchylné k chybám.	Prováděno nástrojem. Vždy je přesný a není ovlivněn lidským pochybením.
Ověření správnosti nasazeného buildu	Testování je velmi časově náročné.	Jsou ideální pro ověření stability nasazeného buildu, tzv. Build Verification Testing (BVT).
Použití	Je vhodné pro zkoumání, použitelnost a testování Ad-hoc. Využívá se také za situace, kdy se často mění aplikace.	Je ideální pro regresní testování, testování výkonu, zátěžové testování nebo pro funkční testovací případy.

2 AUTOMATIZOVANÉ NÁSTROJE

Existuje množství nástrojů, které slouží k automatizovanému testování. Každý je v něčem odlišný, a proto je nutné zvolit správný nástroj, který splňuje požadavky na testování softwaru. Požadavky na rychlejší poskytování kvalitního softwaru neboli „Quality at Speed“ vyžaduje, aby organizace vyhledávaly řešení v metodách „Agile“, „Continuos Integration“ a „DevOps“. Automatizované testování je nezbytnou součástí těchto aspektů. Už v roce 2017 byly zveřejněny informace o nejdůležitějších nástrojích pro automatizaci testů, u nichž se věřilo, že nejlépe vyřeší problémy v automatizaci v příštích několika letech. Kritéria, která byla vybrána pro následující seznam automatizovaných nástrojů, jsou podpora testování API a služeb, nabízení některých funkcí inteligence, strojového učení nebo analytické schopnosti a popularita a vyspělost nástroje. Mezi deset nejlepších automatizovaných nástrojů právě tyto [23]:

- Selenium,
- Katalon Studio,
- UFT,
- TestComplete,
- SoupUI,
- IBM Rational Functional Tester,
- Tricentis Tosca,
- Ranorex,
- Postman,
- Apache JMeter.

Selenium

Je považován za průmyslový standard pro automatizaci uživatelských rozhrání testovaných webových aplikací. Je určen pro vývojáře a testery, kteří mají zkušenosti a dovednosti v oblasti programování a skriptování. Selenium nabízí flexibilitu, která je nevídaná v mnoha jiných nástrojích pro automatizaci testů. Uživatelé mohou psát testovací skripty v mnoha různých jazycích (např. Java, Grooby, Python, C#, PHP, Ruby a Perl), které běží ve více systémových prostředích (Windows, MacOS, Linux) a prohlížečích (Chrome, Firefox, IE atd.).

Jedná se o open source nástroj. Uživatelé musí mít pokročilé programovací schopnosti a musí věnovat značné množství času na vybudování automatizačních rámců

a knihoven potřebných pro automatizaci. To je hlavní nevýhoda Selenia, která je řešena v integrovaných nástrojích, jako je například Katalon Studio, Apache JMeter a jiné. Více detailních informací o tomto nástroji je možné dohledat v [23] a [24].

Katalon Studio

Je výkonné a komplexní řešení pro automatizaci testování API, webových a mobilních aplikací. Má bohatou sadu funkcí pro tyto typy testování a podporuje několik platforem včetně Windows, MacOS a Linuxu. S využitím Selenia a Appia nabízí jedinečné prostředí pro testy, které mají potíže s jejich integrací. Zároveň umožňuje nasazováním různých rámců a knihoven s těmito pluginy. Dále obsahuje inteligentní schopnost nahrávání testů, které generuje pružné a udržovatelné lokátory. To umožňuje, aby zkušební skripty fungovaly i při změně testovací aplikace. Aplikace je zdarma a obsahuje kompletní sadu funkcí pro automatizaci. Podrobnosti o tomto nástroji lze nalézt v [23] a [25].

UFT

Je oblíbený komerční nástroj pro testování webových a mobilních aplikací. Byl rozšířen tak, aby obsahoval dobrý soubor schopností pro testování API. Poskytuje více platforem pro testovanou cílovou aplikaci. Dále zprostředkovává několik pokročilých funkcí pro detekci inteligentních objektů a detekci objektů založených na snímcích. Tato aplikace je placená a stojí ročně 3 200 $. Další informace o tomto nástroji je možné dohledat v [23] a [26].

TestComplete

Obsahuje seznam výkonných a komplexních funkcí pro testování webových, mobilních a desktopových aplikací. Testeři mohou používat JavaScript, VBScript, Phyton nebo C++ Script pro psaní testovacích skriptů. Stejně jako UFT, dokáže přesně detekovat prvky dynamického uživatelského rozhraní. Testeři mohou snadno

používat záznamové a přehrávací funkce. Mohou vložit kontrolní body do testovacích kroků k ověření výsledků. Jako produkt SmartBear může být TestComplete snadno integrován s dalšími produkty nabízenými touto společností. Tato aplikace je placená a stojí ročně 2 399 $ za uživatele. Podrobnosti o tomto nástroji na automatizované testování lze nalézt v [23] a [27].

SoupUI

Nejedná se přímo o testovací automatizační nástroj pro testování webových nebo mobilních aplikací, ale může být nástrojem pro testování API a webových služeb. Jedná se funkční testovací nástroj speciálně navržený pro testování API. Podporuje služby REST a SOAP. Testeři, kteří automatizují rozhraní API mohou používat verzi open source nebo pro. Pro vydání má uživatelsky přívětivé rozhraní a několik pokročilých funkcí. Mezi pokročilé funkce lze zařadit generování testů pomocí drag and drop, point-and-click nebo asynchronní testování. Verze „Open-source“ je zdarma, verze „Pro“ začíná na částce 659 $ ročně. Více detailních informací o tomto nástroji je možné dohledat v [23] a [28].

IBM Rational Functional Tester

IBM Rational Functional Tester je testovací automatizační nástroj určený pro testování aplikací, které jsou vyvinuty pomocí jazyků a technologií, jako jsou Web, .NET, Java, Visual Basic, Siebel, SAP, PowerBuilder, Adobe Flex a Dojo Toolkit. Je také datovou řízenou zkušební platformou pro funkční a regresní testování. Dále poskytuje funkci nazvanou „testování storyboardu“, která pomáhá vizualizovat a upravovat testy pomocí snímků obrazovky přirozeného jazyka a aplikací. Funkce ScriptAssure umožňuje testerům vytvářet zkušební skripty, které jsou odolné změnám v uživatelském rozhraní. Tento nástroj lze také integrovat s dalšími nástroji IBM pro správu životního cyklu aplikací, jako je IBM Rational Team Concert a Rational Quality

Manager. Nástroj má pouze komerční licenci. Další informace o tomto nástroji je možné dohledat v [23] a [29].

Tricentis Tosca

Existuje několik testovacích platforem, které poskytují komplexní sady nástrojů pro podporu testovacích činností, od testování a automatizace testů až po testovací zprávy a analýzy. Tricentis Tosca je jednou z nich. Tento nástroj má mnoho funkcí, jako jsou dashbordy, analýzy, integrace a provedení, které podporují nepřetržitou integraci a postupy DevOps. Navíc nabízí přátelské uživatelské rozhraní a bohatou sadu funkcí pro navrhování, implementaci, provádění, správu a optimalizaci testů API. Licence je opět komerční. Podrobnosti o tomto nástroji lze nalézt v [23] a [30].

Ranorex

Po dobu mnoha let poskytuje společnost Ranorex komplexní a profesionální sadu funkcí pro testování webových, mobilních a desktopových aplikací. Využitím svých dosavadních zkušeností s automatizací testů má společnost Ranorex pokročilé funkce pro identifikaci, úpravu a správu prvků uživatelského rozhraní. Stejně jako Studio Katalon je návrh automatizovaných testů pro testery snadný, jelikož využívají intuitivní GUI rozhraní, které slouží pro záznam, přehrávání a generování skriptů. Tento software je placený, cena je 690 € za doplněk a 2290 € pro licenci Premium. Více detailních informací o tomto nástroji je možné dohledat v [23] a [31].

Postman

Postman je další automatizační nástroj určený pro testování API. Uživatelé mohou tento nástroj nainstalovat jako rozšíření prohlížeče nebo pracovní plochy. Je podporován všemi systémovými prostředími jako je MacOS, Linux nebo Windows. Je populární nejen mezi testery pro testování automatizace API, ale také vývojáři, kteří

tento nástroj používají k vývoji a testování rozhraní API. Licence tohoto softwaru je komerční. Více detailních podrobností o tomto nástroji lze dohledat v [23] a [32].

Apache JMeter

Apache JMeter je open source software, který byl vyvinut společností Apache Software Foundation. Tato společnost poskytuje služby a podporu mnoha komunitním softwarovým projektům, které se rozhodly připojit k ASF. ASF je známá jako „The Apache Way“ a přináší vývoj volně dostupného softwaru pro podnikovou sféru, veškerý software je pod licencí Apache. [33]

Software Apache JMeter je implementován pomocí jazyka Java a slouží převážně pro testování funkčního chování a výkonnostního testování. Původně byl tento software navržen pouze pro testování webových aplikací, ale novější verze tohoto softwaru umožňují rozšíření o další funkčnosti. Proto se dá říci, že tento software se nejvíce používá pro testování webových dynamických aplikací. Dále se využívá k simulaci těžké zátěže na serveru, skupině serverů nebo na samotné síti, aby bylo možné otestovat jejich sílu nebo analyzovat celkový výkon systému. [34]

Apache JMeter zahrnuje schopnost načíst a testovat výkon různých typů aplikací, serverů a protokolů, jako jsou HTTP / HTTPS, SOAP / REST webové služby, FTP, databáze přes JDBC, mail pomocí SMTP / POP3 / IMAP apod. Plně vybavený testovací modul umožňuje rychlé nahrávání testovacího plánu přímo z prohlížeče. Zároveň lze spustit test z libovolného Java kompatibilního operačního systému (Linux, Windows, Mac OSX, …). Více detailních informací o všech schopnostech lze dohledat v [34].

JMeter funguje na úrovni protokolů, kdy se pro jednotlivé činnosti posílají jednotlivé žádosti (Requests), proto jedním z důležitých rozšíření je možnost využití Selenia. Toto rozšíření umožňuje využívat WebDriver pro konkrétní webový prohlížeč a pomocí vytvořeného kódu simulovat reálné využití aplikace v daném prohlížeči. Toto rozšíření skutečně odráží reálné chování uživatele, jelikož daný software následně kliká na požadovaná tlačítka v prohlížeči a nevolá pouze jednotlivé metody

v pozadí. Bohužel čas běhu takto vytvořeného testu je znatelně delší než čas testu, který je vytvořen pouze pomocí žádostí, jelikož zde hraje roli více vnějších faktorů, jako je obsah zobrazované stránky, délka stisknutí tlačítka apod. [34]

Tento software je využit ve firmě Unicorn Systems z několika důvodů. Hlavním důvodem je jeho jednoduchost. Nezkušený člověk dokáže vytvořit funkční Smoke Test už během jednoho týdne, a to bez znalosti jakéhokoliv programovacího jazyka. S tímto důvodem lze spojit také myšlenku lehkého předání vytvořených automatizovaných testů jiné osobě, a to bez znalosti aplikace, pro kterou se tyto testy vytvářely.

Další podmínkou při výběru softwaru pro automatizované testy byl požadavek na testování frontendu aplikace. Tuto schopnost dociluje Apache JMeter pomocí rozšíření o Selenium. Rozšíření využívá sekvence programovacího kódu. Pokud se uživatel naučí základní programovací příkazy, tak pro většinu základních testovacích scénářů jsou tyto příkazy plně dostačující. Pokud nastane speciální testovací případ, kdy znalost programovacího kódu není dostatečná, lze využít rozsáhlou dokumentaci nebo pomoc komunity na internetu. To byly také další důvody, proč Apache JMeter je hojně využíván ve společnosti Unicorn Systems.

Velice podstatnou roli při vybírání softwaru hrála jeho kompatibilita s vybranými nástroji. Pro firmu US je důležité, aby software na automatizované testování umožňoval propojení s nástrojem TeamCity. Ten umožňuje spouštět automatizovaný test bez jakékoliv znalosti softwaru Apache JMeter. Dále se požaduje, aby software dokázal zapisovat své výsledky automatizovaných testů do nástroje PractiTest. Více detailních informací o nástrojích, které jsou využívány firmou US a jednotlivých propojení je popsáno v kapitole 3.

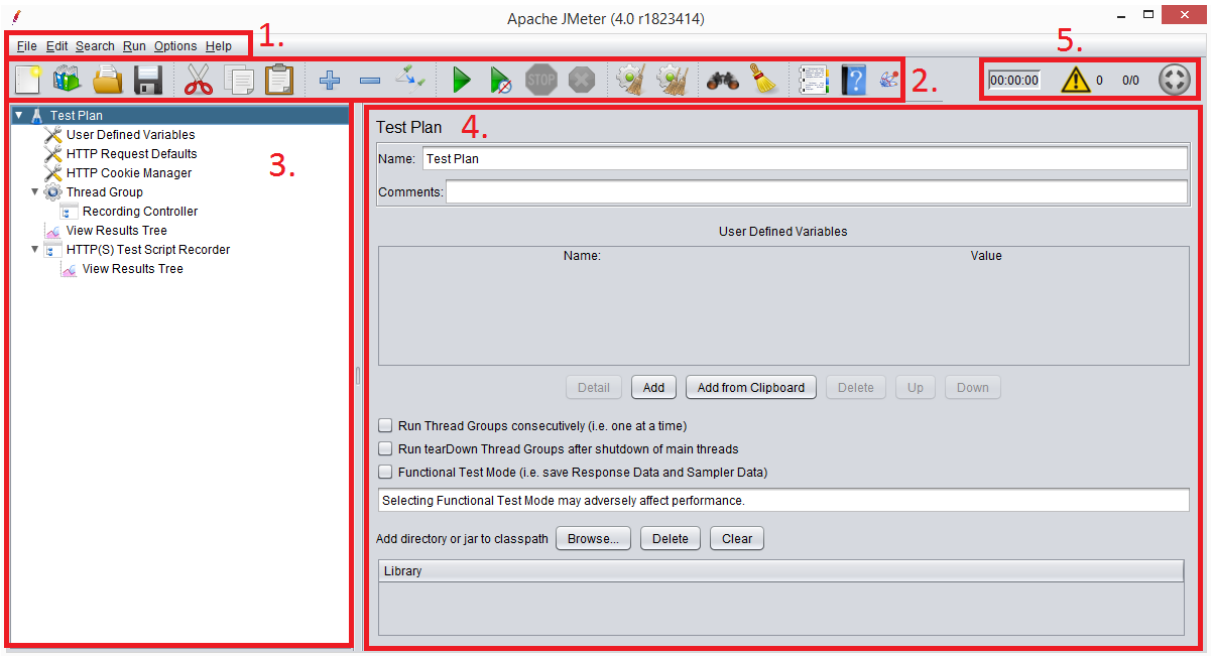
V poslední řadě hrála důležitou stránku také cena za tento software. Apache JMeter je zcela zdarma a nevyžaduje žádné placené rozšíření, které by bylo nutné pořídit k plné funkčnosti tohoto softwaru.

2.1 Uživatelské rozhraní

Předtím než se podíváme na samotné rozhraní nástroje Apache JMeter, je nutné daný software nainstalovat. Jelikož se jedná o open-source software, je stažení nástroje zcela zdarma. Instalační soubory je možné nalézt na internetových stránkách https://jmeter.apache.org/.

Instalace JMeteru je opravdu jednoduchá. Po stažení balíčku z internetových stránek společnosti, stačí pouze tento balíček rozbalit. Ten v sobě obsahuje dvě základní složky. První je složka „lib“ (library), ve které najdeme celou základní knihovnu, která je plnohodnotná k používání daného softwaru. Pokud by bylo zapotřebí rozšířit tento nástroj o určité funkcionality, je potřeba si stáhnout daný plugin a jeho obsah vložit právě do složky lib. Druhým základním prvkem je složka „bin“. Ta obsahuje všechny ostatní soubory. Pro spuštění JMeteru je potřeba jmeterw.cmd nebo jmeter.sh soubor. Záleží, jaký druh operačního systému je využit. Jedinou podmínkou pro správné spuštění aplikace je nainstalovaná Java na daném zařízení.

Jak je možné vidět na obrázku 11, po spuštění nástroje se zobrazí okno se základním rozložením funkcí. Aplikace má jednoduché a uživatelsky srozumitelné rozhraní, které lze rozdělit na pět částí. Novější verze JMeteru umožňují přizpůsobit vzhled podle předdefinovaných stylů. Proto se můžete setkat s tmavou variantou tohoto softwaru nebo i napodobeninou retro stylu, který byl využíván u operačního systému Windows.



Obrázek 11 - Základní okno nástroje Apache JMeter. *Zdroj: Autor*

Pod první částí si lze představit typické záložky, jako je nabídka File, Edit a podobně, které slouží k provozu základních vlastností nástroje. V části číslo dvě se nachází panel zobrazující ikony pro rychlí přístup k jednotlivým funkcím nástroje, které se nejčastěji využívají během automatizovaného testování. Mezi tyto funkčnosti je možné zařadit například ikony pro spouštění, ukládání a otevírání automatizovaného testu, pomocná ikona nebo tlačítko, pomocí kterého lze vyčistit výsledky testu před spuštěním dalšího kola. Toto pole je možné editovat a zvolit si jednotlivé ikony podle potřeby.

Třetí část slouží k zobrazení stromové struktury vytvořeného automatizovaného testu. Stromová struktura se skládá z jednotlivých modulů, které plní požadované procesy v automatizovaném testu. Po spuštění nástroje obsahuje stromová struktura pouze rodičovský uzel. Jak je vidět na obrázku níže, jedná se o uzel s názvem Test Plan. Z tohoto uzlu potom vycházejí jednotlivé moduly.

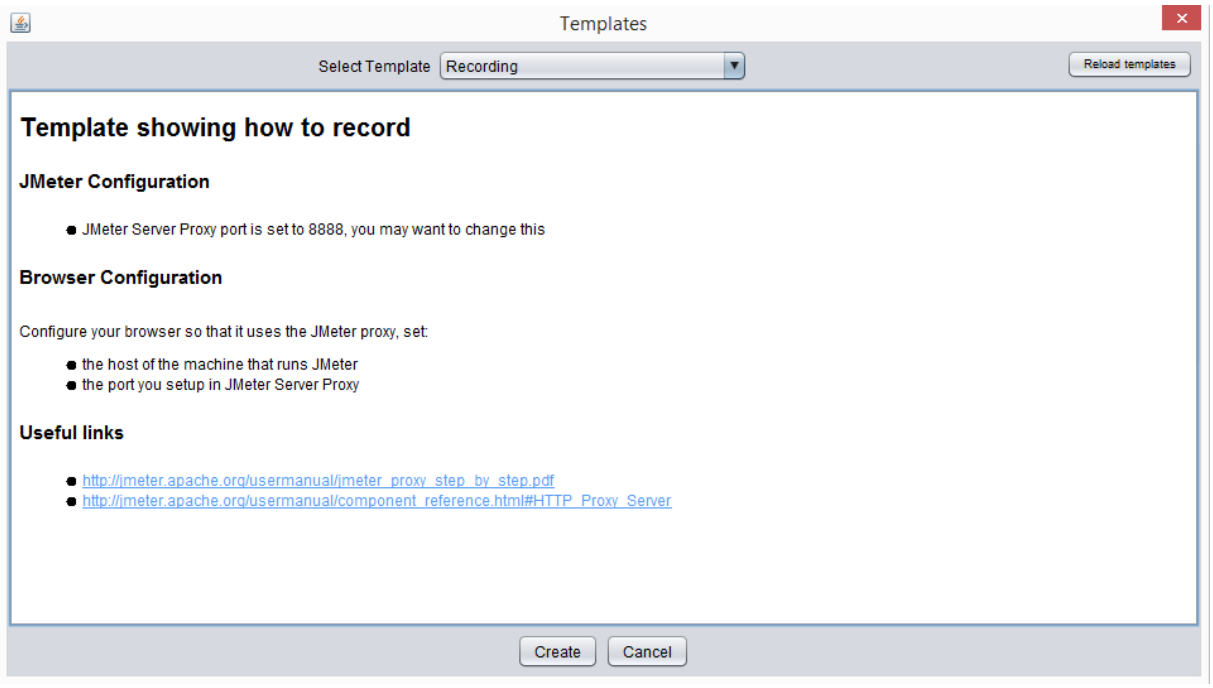
Oblast čtyři zobrazuje veškeré detailní informace, které se týkají vybraného modulu ve stromové struktuře. Tato část se tedy dynamicky mění v závislosti na výběru uživatele. Dále slouží k doplňování, upravování a správě jednotlivých modulů. Více názorných ukázek je možné dohledat u praktického přikladu, který je součástí této

diplomové práce. Pátá oblast, zároveň je to poslední část, je zaměřena na detailní informace o vyskytnutých chybách. Pod jednotlivými ikonami se nachází logovací funkce JMeteru. Pomocí jednotlivých zaznamenávaných informací lze snáze vyřešit nastalý problém.

2.2 Nastavení pro tvorbu automatizovaného testu

Tento nástroj umožňuje nahrávat kroky uživatele, čímž posléze dokáže reprodukovat daný testovací scénář. Hlavní výhodou tohoto nástroje je jednoduchost nahrávání. Právě tento postup, jak vytvořit automatizovaný test, je popisován v této kapitole.

Po spuštění nástroje je zobrazena úvodní obrazovka nástroje. Ta obsahuje funkčnost výběru z předem připravených šablon. Z jednotlivých šablon vybereme tu, která slouží k nahrávání skriptu. Šablona Recording obsahuje základní popis konfigurace a užitečné linky. Tuto šablonu charakterizuje obrázek 12, který zároveň popisuje jednotlivé konfigurace.

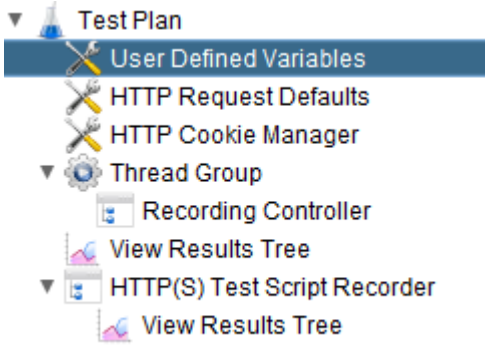


Obrázek 12 - Šablona Recording. *Zdroj: Autor*

Jak je možné vidět na obrázku 13, po vytvoření vzniknou ve stromové struktuře základní moduly, které jsou potřebné pro nahrání daného skriptu. Jednotlivé moduly

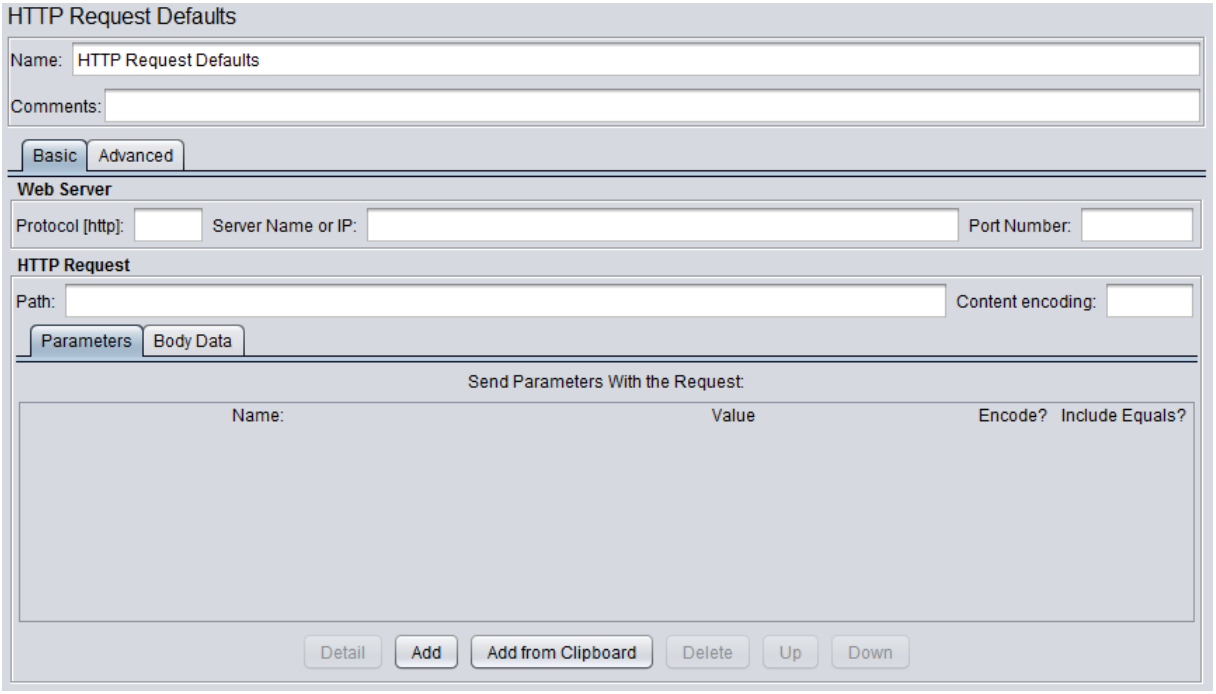
mají přesně danou funkci. Test Plan, jak již bylo řečeno, slouží k prezentaci samotného testu. User Defined Variables definuje proměnné, které se využívají v průběhu celého testu. HTTP Request Defaults slouží pro definici společných parametrů pro všechny requesty, jako je protokol, IP adresa, port apod. HTTP Cookie Manager ukládá a odesílá cookies, pracuje s cookies, stejně jako prohlížeč. Thread Group, tímto modulem začíná daný testovací skript. Definuje počet uživatelů, kteří budou využívat daný test v jeho dceřiných uzlech. V tomto uzlu se nacházejí základní Logic Controllers, ty určují pořadí, v jakém jsou jednotlivé vzorky zpracovávány. Pro nahrávání postačuje základní Recording Controller, který, jak již název napovídá, slouží k uchování záznamu daného testovacího postupu.

HTTP(S) Test Script Recorder, který slouží k propojení internetového prohlížeče a JMeteru. Kdekoliv ve stromové struktuře skriptu se může objevit View Reults Tree, ten je využíván k zobrazení všech výsledků testu v dané grupě, controlleru nebo nahrávání.



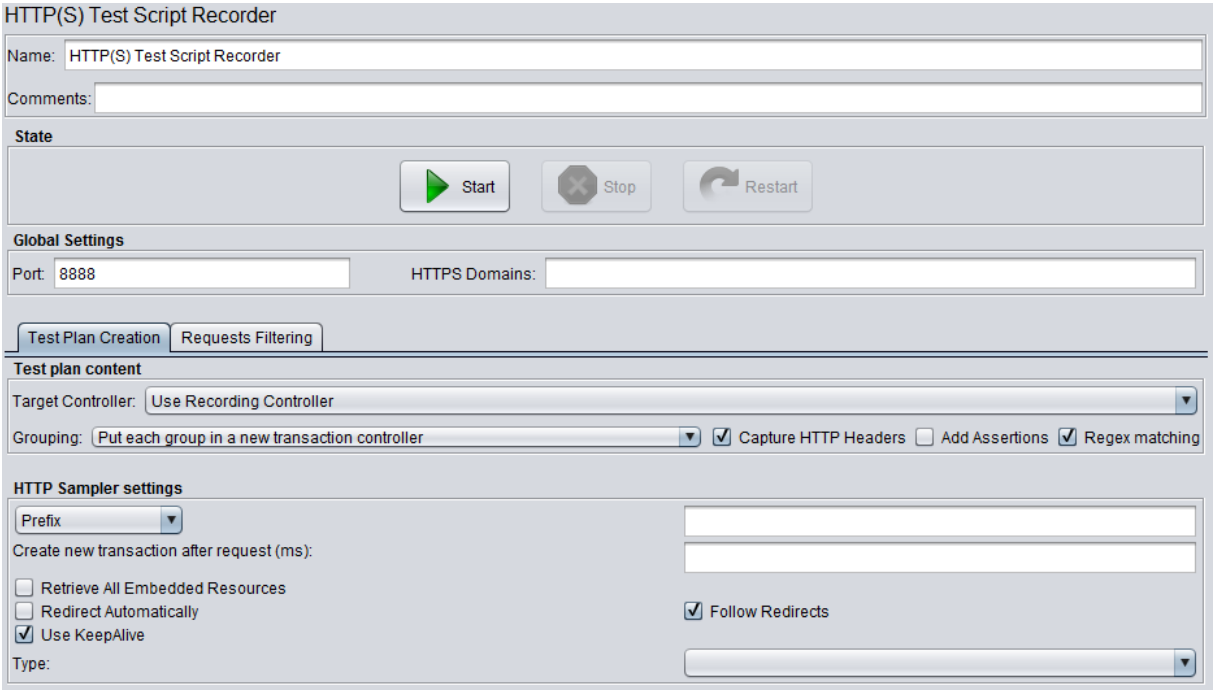
Obrázek 13 - Stromová struktura skriptu. *Zdroj: Autor*

Pro správné nahrávání scénáře je nutné, aby byly jednotlivé části odpovídajícím způsobem nastaveny, a aby plnily to, k čemu byly předurčeny. Pro HTTP Request Defaults, který je zobrazen na obrázku 14, je nutné nastavit Web Server. Do této kolonky se zadá webová adresa nebo je možné zadat přímo IP adresu příslušné stránky či aplikace, kterou je potřeba testovat. Položka Path potom slouží k upřesnění cesty, ale není ji nutné vyplňovat.



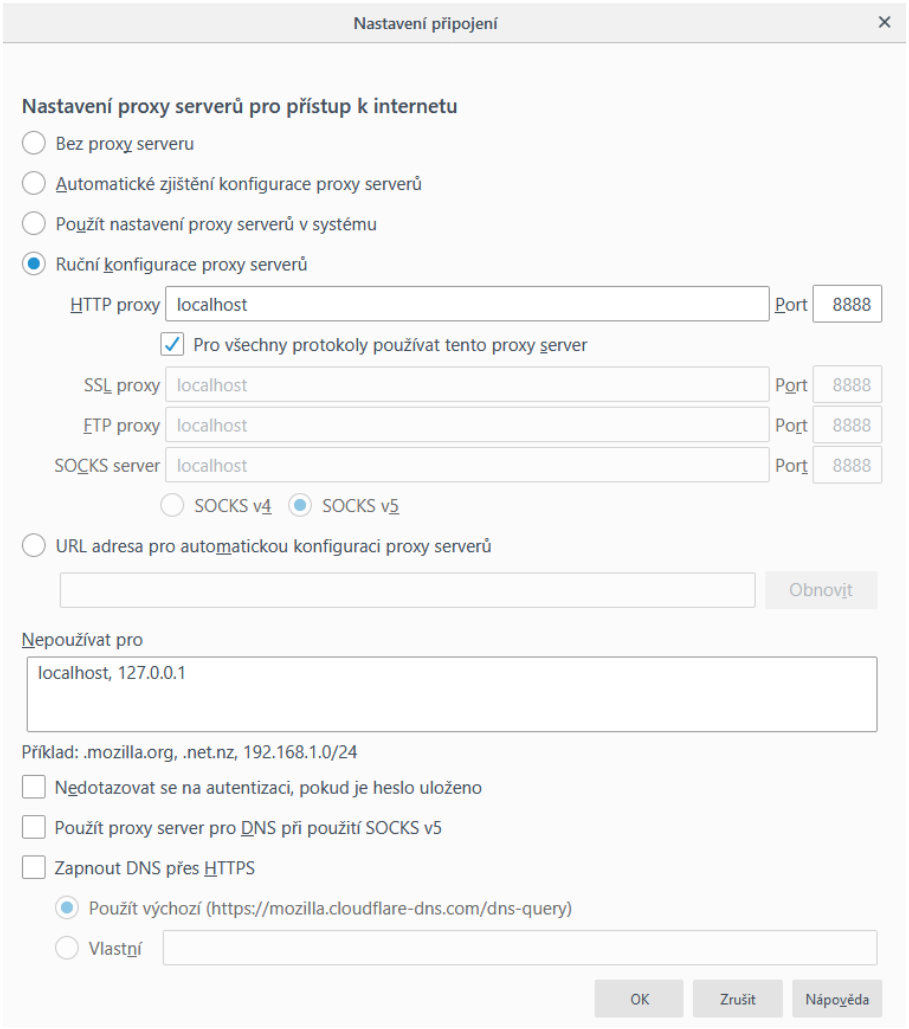
Obrázek 14 - HTTP Request Defaults. *Zdroj: Autor*

HTTP(S) Test Script Recorder je nutné nastavit pro propojení internetového prohlížeče a JMeteru. V kolonce Grouping se zvolí hodnota „Put each group in a new controller“. To umožňuje alespoň základní seskupování jednotlivých nahraných requestů. Potom je nutné nastavit Port. Zde je podstatné zadat takovou hodnotu, která není využívána jinými aplikacemi v počítači. V tomto případě se osvědčil port 8888, ten se musí shodovat s hodnotou portu, který se vyplňuje v internetovém prohlížeči. Ostatní hodnoty mohou zůstat v základním nastavením. Takto nastavený HTTP(S) Test Script Recorder lze zhlédnout na obrázku 15.



Obrázek 15 - HTTP(S) Test Script Recorder. *Zdroj: Autor*

Poslední důležitou částí je správné nastavení webového prohlížeče. Ve webovém prohlížeči v nastavení připojení je důležité zvolit možnost ruční konfigurace proxy serverů. Do kolonky HTTP Proxy zadejte například localhost, port se musí shodovat s tím v JMeteru, tedy 8888 a zaškrtněte možnost pro všechny protokoly používat tento proxy server. Toto je poslední nutné nastavení a nyní je možné začít nahrávat testovací skript. Takto vyplněné nastavení webového prohlížeče je možné vidět na obrázku 16.



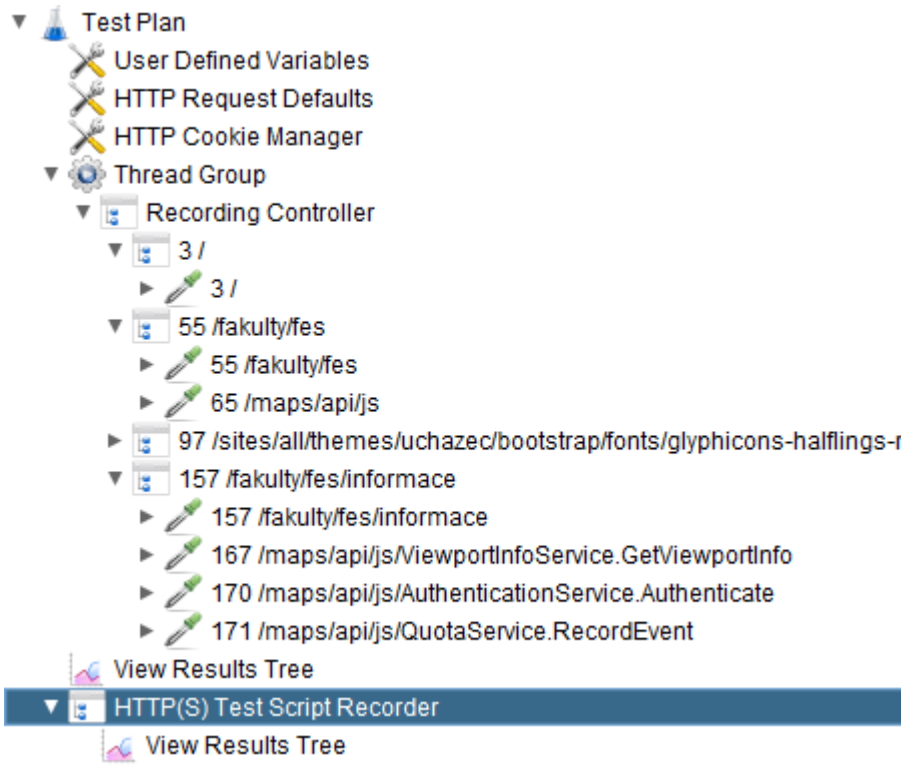
Obrázek 16 - Nastavení webového prohlížeče. *Zdroj: Autor*

2.3 Vytvoření automatizovaného testu

Nahrání testovacího skriptu podle určitého scénáře je v tuto chvíli poměrně jednoduché. Pokud jsou nastaveny všechny dílčí kroky z předchozí kapitoly správně, stačí pouze v HTTP(S) Test Script Recorderu kliknout na tlačítko Start. Tím se spustí sledování uživatele, který v tento moment může provádět testovací scénář v daném internetovém prohlížeči. Tester provádí svou činnost bez jakéhokoliv ohledu na automatizovaný nástroj. Po dokončení svého scénáře stačí opět ve stejném modulu kliknout na tlačítko stop. Tím je ukončeno sledování uživatele v prohlížeči. Na příkladu níže je znázorněno, jak jednoduché je vytvořit základní automatizovaný skript podle určitého scénáře. Scénář zahrnoval následující kroky:

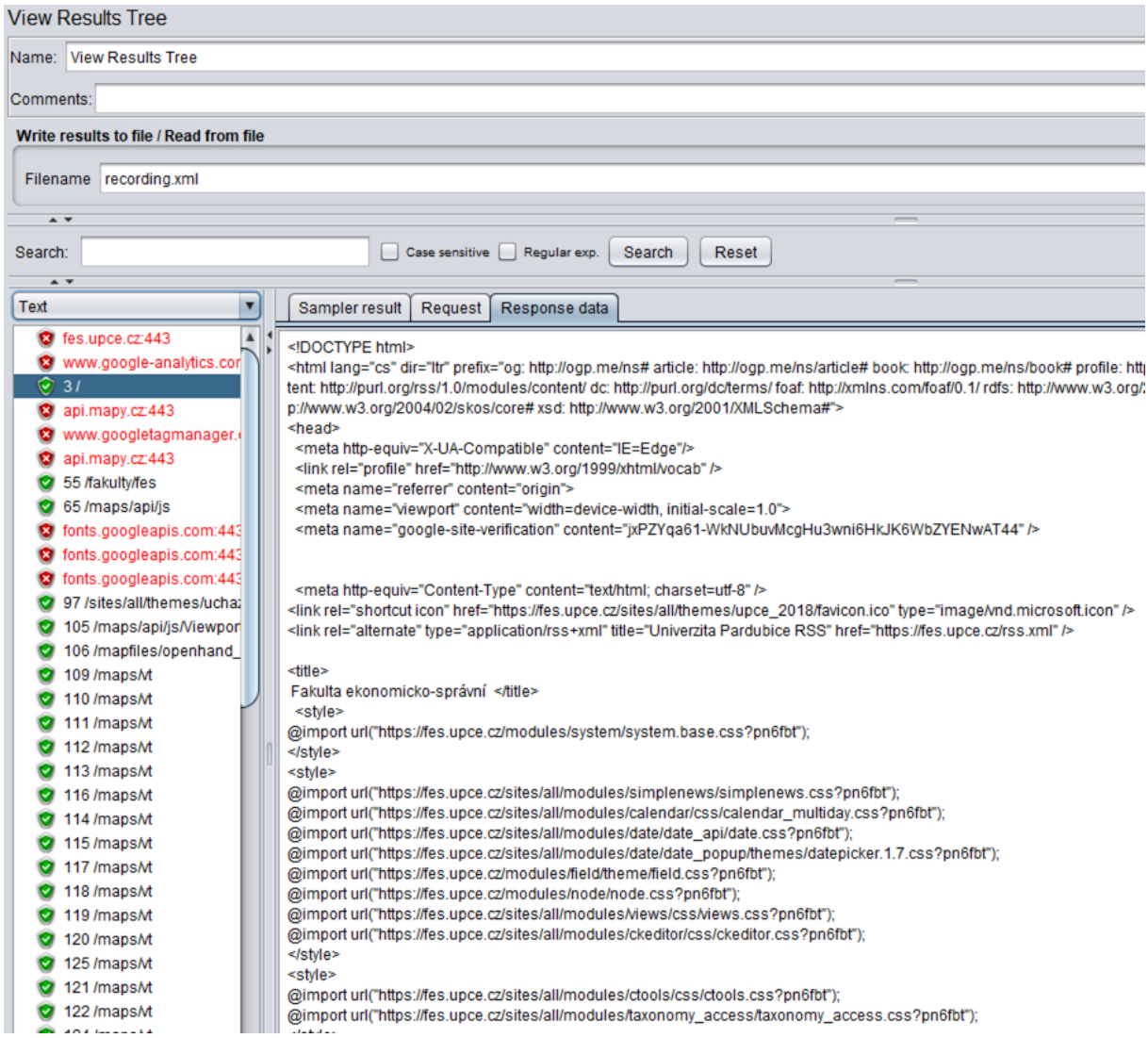
- zobrazení hlavní stránky Univerzity Pardubice a kontrolu vybraných názvů,
- zobrazení hlavní stránky fakulty ekonomicko-správní a kontrolu vybraných názvů,
- zobrazení stránky „Fakulta ekonomicko-správní > spravovat a řídit hodnoty“ a kontrolu vybraných názvů,
- zobrazení stránky „Informace k přijímačkám na FES“ a kontrolu vybraných názvů.

Jelikož je na začátku zvolen určitý typ seskupování, všechny nahrané kroky jsou zaznamenány do Recording Controlleru v určité struktuře. V tomto případě jednotlivé stránky a jejich kompletní obsah je roztříděn do separátních podmnožin. Doporučuje se rozdělit testovací scénář na menší nahrávací celky. Jak si lze všimnout na obrázku 17, během nahrávání se zaznamenávají i všechny obrázky, mapy a ostatní doplňky, které jsou na dané stránce obsaženy. Ty jsou pro tento konkrétní automatizovaný test ve většině případů zcela nepotřebné a je možné je odfiltrovat. Může se stát, že jednotlivé popisky jsou zcela zavádějící, jako to je v tomto případě. Například request pod názvem „3 /“ obsahuje krok, kdy se uživatel dostal na hlavní stránku fakulty ekonomicko-správní, což není z názvu nahraného requestu jasné. Navíc je tento krok velice důležitý z hlediska testovacího scénáře, jelikož je cílem právě tuto stránku automatizovaně testovat. Z tohoto důvodu je důležité mít při nahrávání modul, který zobrazí všechny výsledky během testerova postupu.



Obrázek 17 - Nahraný skript UPCE. *Zdroj: Autor*

View Results Tree, který je obsažen právě v HTTP(S) Test Script Recorderu, slouží ke získání všech důležitých informací, které jsou vráceny během nahrávání. Tedy obsahuje všechna data, která jsou zobrazena na dané stránce. Záleží čistě na uživateli JMeteru, jak budou jednotlivá data prezentována. Může si zvolit zobrazení formou textu, jak je možné vidět na obrázku 18, nebo zvolit jinou dostupnou možnost, například zobrazení v základní HTML struktuře, jako je to například na obrázku 20.



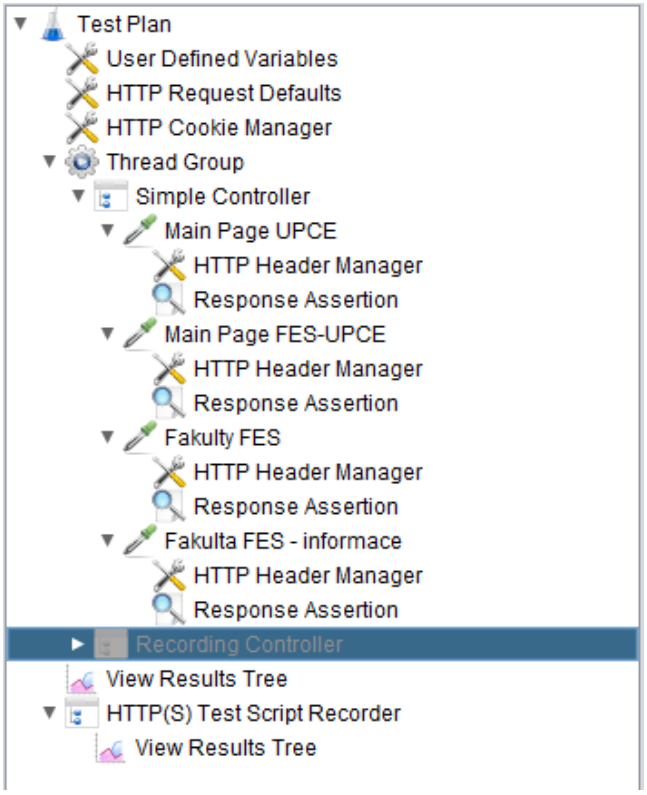
Obrázek 18 - View Results Tree s nahraným skriptem. *Zdroj: Autor*

Nyní máme dostupná všechna data, která jsou potřebná pro vytvoření jednoduchého automatizovaného testu. Nejprve je nutné pročistit a uspořádat stromovou strukturu tak, aby odpovídala testovacímu scénáři. Do stromové struktury byl přidán Simple Controller, který je základním uzlem navrhovaného scénáře. Požadované requesty jsou přesunuty z Recording Controlleru do Simple Controlleru a přejmenovány, kvůli lepší přehlednosti. Zároveň přejmenování slouží k orientaci během jednotlivých testovacích kol, kde konkrétní název vypovídá o aktuální testované části systému.

Jak si lze všimnou na obrázku 19, ke každému requestu byl zároveň přidán Response Assertion. Tento článek testovacího skriptu je velice důležitý, jelikož se stará o kontrolu dotazovaných stránek. Zjišťuje, zda testovaná stránka obsahuje konkrétní

informace, jako je například nadpis s daným textem, číselná hodnota, obrázek a podobně. Z tohoto důvodu byly doplněny jednotlivé requesty o tento článek s odpovídajícím textem, který se má kontrolovat.

Pro stránku „Fakulta FES – Informace“ je přidána validace, že tato internetová stránka obsahuje nadpisový text „<h1 class="page-header">Informace k přijímačkám na FES</h1>“. Tento text bude kontrolován v každém spuštěném běhu automatizovaného testu. Zároveň je nutné, aby nepotřebné kroky byly zablokovány a nespouštěly se, jelikož nejsou během testovacího scénáře důležité. V tomto případě je Recording Controller vyřazen a jednotlivé kroky, které obsahuje z předchozího nahrávání, nebudou zahájeny. Nebo je možné odstranit celý tento modul, ale tím bychom přišli o veškeré získané informace, které jsou nutné, pokud je sestavený skript špatně implementován a nesplňuje zadaný testovací cíl. Pokud dojde k odstranění a zároveň je testovací skript špatně navržen, nezbývá nic jiného, než provést nahrání všech kroků znovu a následně postupovat podle předchozího popisu od začátku.



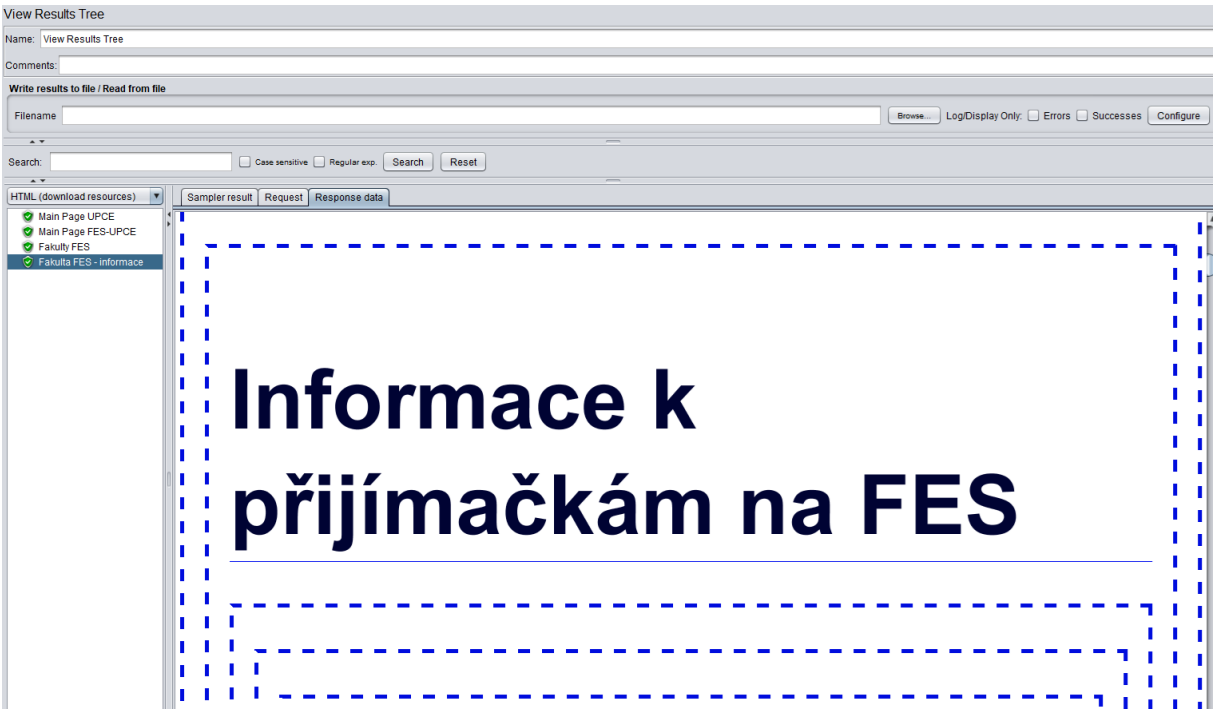
Obrázek 19 - Upravená stromová struktura testu. *Zdroj: Autor*

V tuto chvíli je základní automatizovaný test hotov. Stačí jej pouze spustit a zjistit, zda dosahuje požadovaných výsledků a odpovídá testovacímu scénáři.

2.4 Získané výsledky

Po spuštění skriptu, přes tlačítko v horním menu, získáváme informace o průběhu jednotlivých kroků. Opět jsou všechny výsledky zaznamenány do View Results Tree. Na obrázku 20 lze vidět, všechna zaznamenaná data od spuštění. Ty mají zelenou ikonu, která označuje, že daný test proběhl správně. To, že proběhl správně, je zajištěno validačním textem, který byl vložen do Response Assertion. Zároveň si lze ověřit správnost pomocí záložky Response data, kde uživatel vidí strukturu webové stránky, pro kterou byl odeslán daný request. Pokud si uživatel JMeteru zvolí zobrazení v základní HTML struktuře, může zároveň vidět jednotlivé vnořování textu. To se většinou využívá, pokud potřebujeme zjišťovat načtené hodnoty v tabulkách, správně oddělený text, nebo se s tím setkáváme v jiných vizuálních případech.

Současně si lze všimnout, že po spuštění testovacího skriptu nebyly zároveň spuštěny requesty, které se nacházejí ve vyřazeném Recording Controlleru. Jelikož průběh testu je realizován systematicky odshora dolů podle jednotlivých kroků, tak na kroky, které se nacházejí právě ve vyřazeném Recording Controlleru už nedošlo, protože celý tento uzel byl přeskočen.



Obrázek 20 - Výsledky skriptu UPCE. *Zdroj: Autor*

Celý testovací scénář na kontrolu webových stránek Univerzity Pardubice trval pomocí nástroje Apache JMeter pouze 18 sekund. Jednotlivé kroky i celý scénář lze libovolně opakovat a simulovat tím různé případy využívání. Dokonce byl proveden i scénář, kde tyto 4 kroky provádělo 1000 uživatelů najednou. Takovéto zátěžové testování je pomocí JMeteru velice jednoduché a výsledky jsou okamžitě viditelné. V důsledku tohoto zátěžového pokusu byly stránky univerzity dočasně nedostupné.

3 NÁSTROJE PRO MANAGEMENT SOFTWARU

Jak již bylo řečeno, je velmi důležité efektivně a správně řídit testovací proces. Proto se tato kapitola věnuje vybraným nástrojům, které pomáhají řídit testovací proces u vyvíjeného softwaru. Jednotlivé nástroje jsou velice důležité, jelikož bez nich je značně finančně a časově obtížné vytvořit a spravovat aplikaci, která by splňovala normy firmy Unicorn Systems.

Každý popisovaný nástroj plní jinou funkci, ale zároveň všechny dohromady tvoří celek, bez kterého by automatizované testování bylo obtížnější, složitější a zdlouhavější. Firma Unicorn Systems využívá dle dosavadních zkušeností právě tyto nástroje:

- Jira,
- Confluence,
- TeamCity,
- PraciTest,
- Sourcetree.

3.1 JIRA

Jira Software je účelově postavený nástroj vytvořený společností Atlassian. Jedná se o platformu, která pomáhá týmům naplánovat, přiřadit, sledovat a řídit práci. Je speciálně navržen pro softwarové týmy. Spojuje kvalitně zpracované Jira workflow s nejdůležitějšími prvky agilního vývoje, jako jsou flexibilní scrum a kanban boardy, reporty v reálném čase a podobně.

Jira je k dispozici ve dvou variantách hostingu: cloud a vlastní správa (self-managed). Pro vlastní správu si mohou zákazníci vybrat mezi serverovým a datovým centrem. Cloud je obecně nejlepší možnost pro týmy, které chtějí začít rychle a snadno. A také pro týmy, které nezvládnou technickou složitost hostingu. S programem Jira Sofware Sever je Jira Software hostován na vlastním hardwaru a dokáže se přizpůsobit podle

požadovaného nastavení. To je nejlepší možnost pro týmy, které potřebují spravovat všechny detaily, mají přísnější požadavky na správu dat a nevadí jim dodatečná složitost hostingu. S datovým centrem může být hostován Jira Software na vlastním hardwaru nebo s prodejci laaS jako AWS a Azure. Tato volba je nejlepší pro podnikové týmy, které potřebují nepřetržitý přístup k Jira Softwaru a zajištění určité výkonnosti.

Tento software umožňuje zadávat chyby, žádosti, tikety nebo úkoly (angl. issues, requests, tickets nebo tasks). Jednotlivé typy je možné sledovat od vytvoření až po dokončení. Lze je přiřazovat osobám a zadat jim prioritu řešení. Pokud si tým stanoví priority jednotlivých úkolů, přináší to efektivní a flexibilní činnost celého týmu. Jednoduchá manipulace pomocí „drag and drop" umožňuje vytvářet organizovanou práci a plánování jednotlivých sprintů.

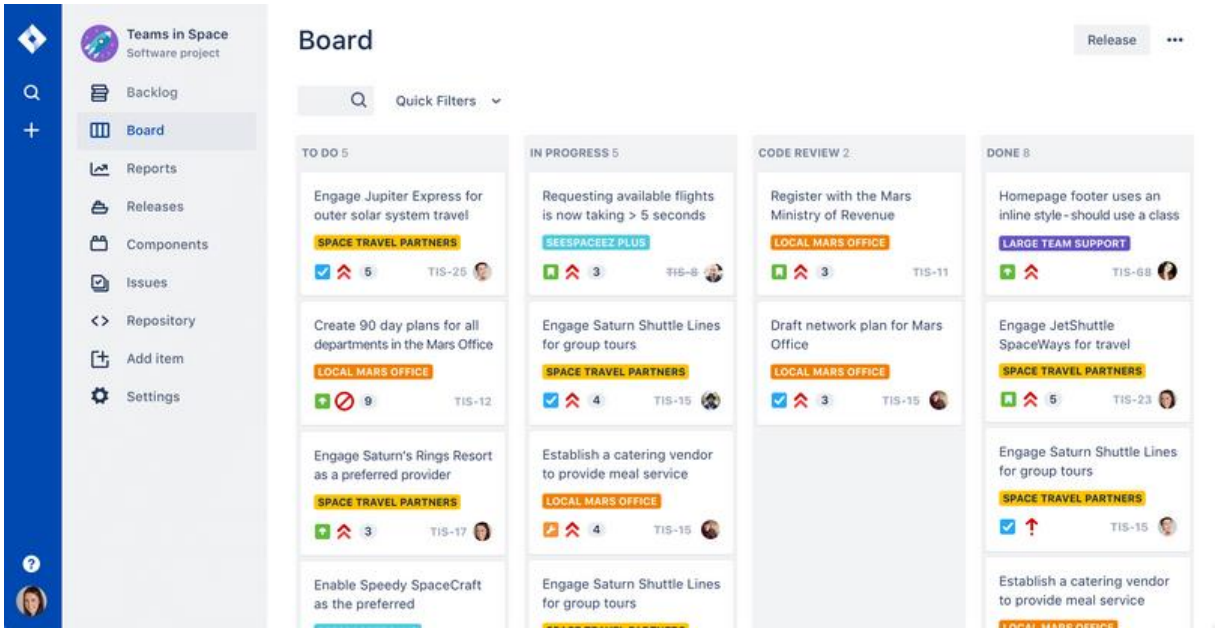
Jira je ve firmě Unicorn využívána z několika důvodů. Prvním důvodem je agilní přístup. Ten zdůrazňuje interativní přístup k práci, který využívá zpětnou vazbou od zákazníků, kde dochází k přírůstkové a kontinuální dodávce. Ideální agilní tým je flexibilní, rychlý a dokáže se přizpůsobit měnícím se požadavkům od zákazníka, aniž by se zbytečně zdržel. Jira podporuje tento způsob například pomocí kanban boardu.

Pro testování je tento nástroj velice důležitý, jelikož se do něj zaznamenávají všechny defekty, které jsou nalezeny v testované aplikaci. Pomocí konkrétního workflow jsou defekty individuálně řešeny, příklad konkrétního workflow je zobrazen na obrázku 6. Testovací tým dokáže pomocí agilního přístupu efektivně retestovat jednotlivé defekty. Výhodou tohoto softwaru je možnost propojení s nástrojem Apache JMeter. Vytvořený automatizovaný test je možné nakonfigurovat tak, že po nalezení chyby v automatizovaném testovacím scénáři sám dokáže zadat tento defekt do softwaru Jira. To přináší úsporu času a lidských zdrojů.

Bohužel jedna z nevýhod tohoto softwaru je jeho pořizovací cena. Pokud zákazník zvolí cloudové řešení, cena se pohybuje okolo 10 dolarů jako měsíční paušální poplatek. Tato speciální cena je pouze pro malé týmy do 10 osob. Jestliže je zapotřebí více osob (11-100 uživatelů), cena se pohybuje v částce 7 dolarů za osobu měsíčně. Tato

cena je určena pro větší a rozšiřující se týmy. Když si zákazník zvolí možnost self-managed, tak za serverové řešení je jednorázová platba 10 dolarů za 10 uživatelů. Cena tohoto řešení se může vyšplhat až ke 45 500 dolarů za 10 001+ uživatelů. Datové centrum začíná na částce 12 000 dolarů za 500 uživatelů ročně a může dosahovat až 450 000 dolarů za 50 001+ uživatelů ročně.

Jak vypadá práce s tímto nástrojem je možné vidět na obrázku 21. Více detailnějších informací o tomto nástroji je možno dohledat v [36] a [37].



Obrázek 21 - Ukázka Jira Software. *Zdroj: [35]*

3.2 CONFLUENCE

Tento nástroj je další publikovaný produkt od australské softwarové společnosti Atlassian. Confluence je otevřený a sdílený pracovní prostor. Umožňuje zákazníkům vytvářet, spolupracovat a uchovávat všechnu jejich práci a na jednom místě. Narozdíl od jakýkoliv nástrojů ke sdílení souborů a dokumentů je Confluence otevřený a přístupný software, který umožňuje týmu a společnosti odvést kvalitnější práci.

Jak již bylo řečeno, tento software je výkonným nástrojem pro týmovou práci. Realizuje cokoli od zápisů ze schůzek a produktových požadavků až po marketingové

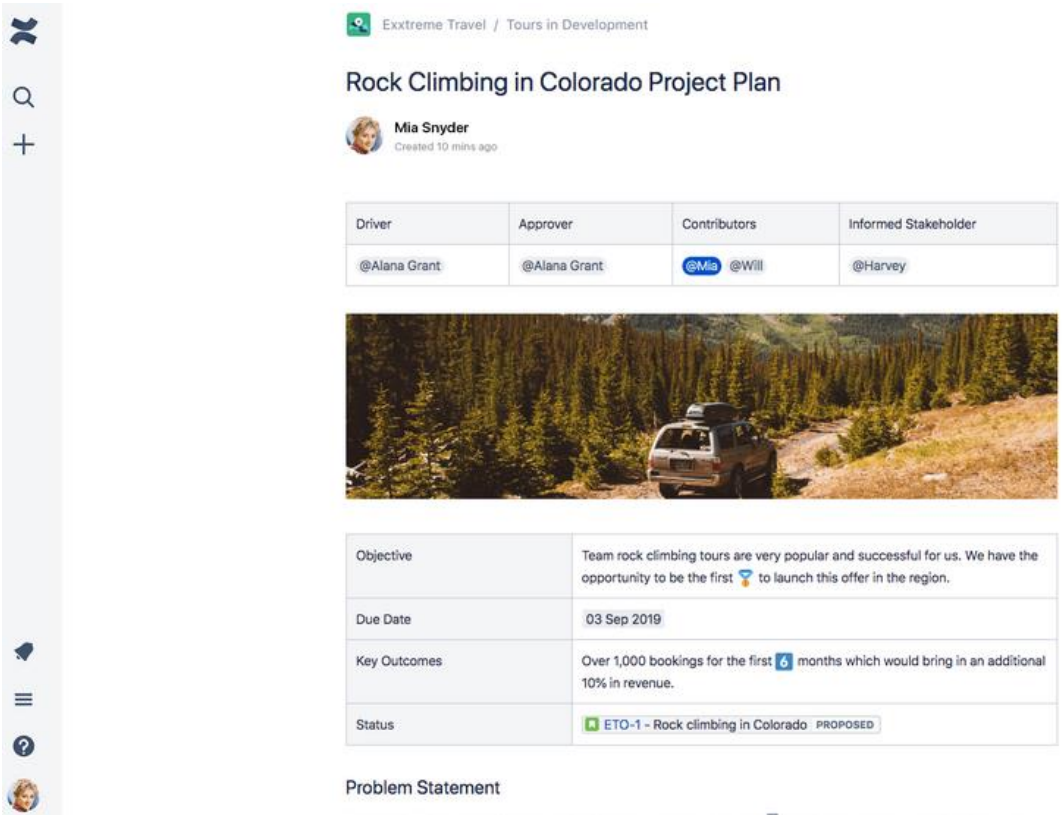
plány a zásady. Jednotlivé stránky lze přizpůsobovat podle potřeby, nebo je možné využít již předem připravené šablony. Dále umožňuje přidávat osobní prvky pomocí obrázků, videí a gifů.

Slouží k udržení organizované práce pomocí seskupováním souvisejících stránek ve vyhrazeném prostoru, ke kterému mohou mít přístup pouze oprávněné osoby, nebo kdokoliv. Pomocí výkonného vyhledávání a strukturovaných stromů stránek je zajištěno, že obsah půjde vždy snadno najít. U tohoto softwaru je možné využít i zpětnou vazbu. Pomocí vložených komentářů a funkce „@ zmiňování členů týmu“, lze získat větší vhled do rozhodování, které je potřeba učinit.

Obrovskou výhodou je propojení s ostatními nástroji od této společnosti. Pomocí propojení s Jira je Confluence často využíván k plánování a přehledu jednotlivých releasů (release management), k uchování dokumentace a k přehledu jednotlivých vývojových, servisních nebo produkčních prostředí. Další pozitivní funkcí je, že Confluence umožňuje propojení například i s PractiTestem, pomocí kterého lze zobrazovat jednotlivé grafy o rozsahu testování vyvíjené aplikace. To umožňuje managerovi sledovat a řídit daný projekt na jednom místě, a mít tak přehled o všech aktivitách, které jsou v danou chvíli v činnosti.

Patrně jedinou nevýhodou toho softwaru je, že všechny jeho služby a funkce jsou placené. Princip placení funguje stejně jako u Jira softwaru. Opět se rozděluje na možnost využití cloudu nebo vlastní správu (Self-managed). Pokud uživatel zvolí cloudovou možnost, zaplatí buďto 10 dolarů měsíční pevný poplatek až za 10 uživatelů, nebo 5 dolarů na uživatele za měsíc pro 11-100 uživatelů. Za serverovou variantu zaplatí jednorázový poplatek 10 dolarů za 10 uživatelů, ale rovněž se cena může vyšplhat až na 35 400 dolarů za 10 001+ uživatelů. V případě datového centra je pro 500 uživatelů vyžadováno 12 000 dolarů ročně, a v případě 40 001+ uživatelů až 200 000 dolarů ročně.

Ukázku vizualizace v nástroji Confluence je možné nalézt na obrázku 22. Více detailních informací je možné nalézt v [38].



Obrázek 22 - Ukázka nástroje Confluence. *Zdroj: [38]*

3.3 TEAMCITY

TeamCity je kontinuální integrační nástroj napsaný v jazyku Java, který podporuje vytváření a nasazování různých typů projektů. Jedná se o nástroj společnosti JetBrains a řadí se mezi nástroje kontinuální integrace (CI). Umožňuje až 100 konfigurací a tři volné agenty v bezplatné verzi.

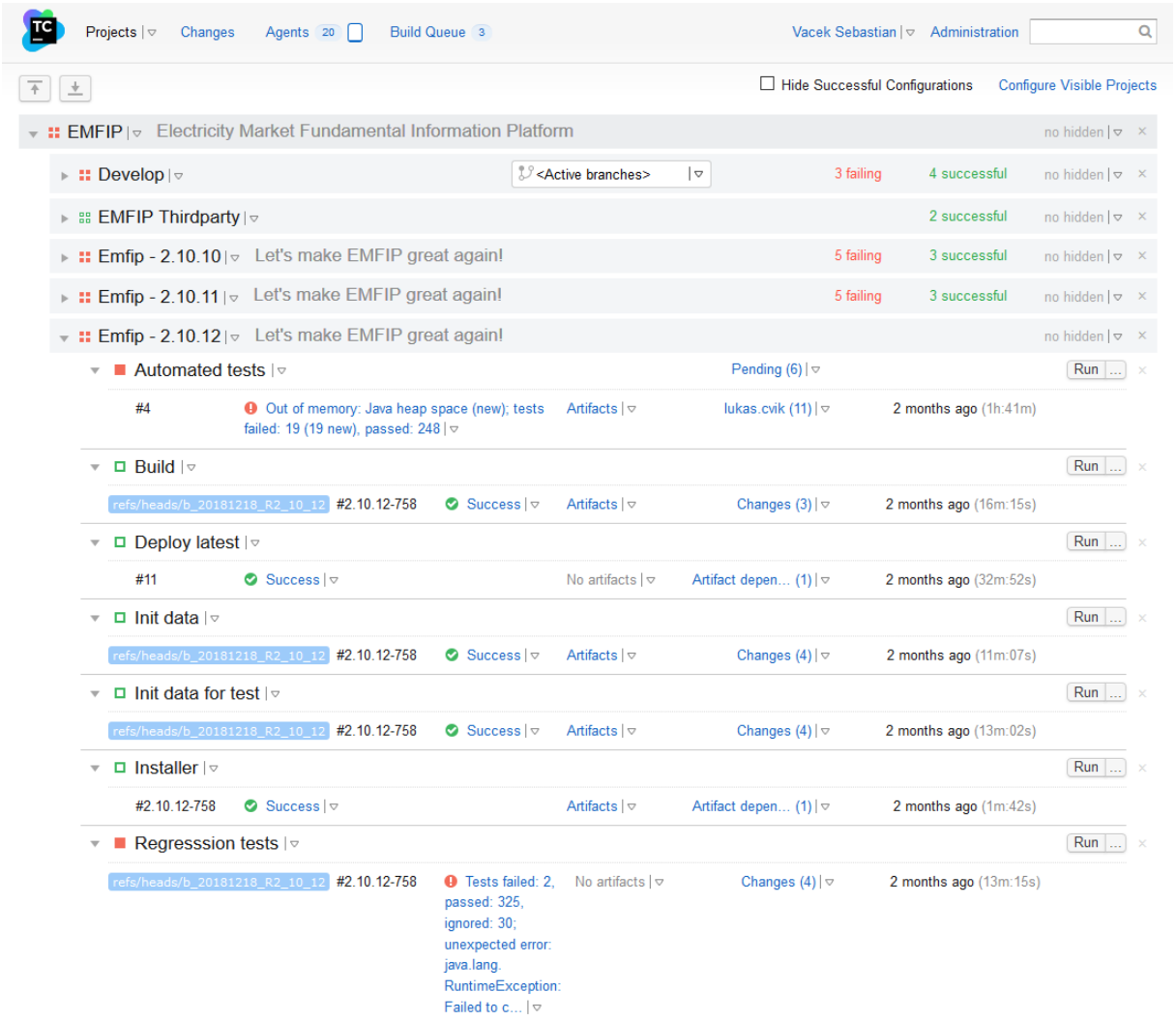
Tento software využívá inteligentní konfiguraci, pomocí které lze vytvořit konkrétní hierarchii projektu. To znamená, že projekt bude zobrazován ve stromové struktuře tak, aby dědil výchozí nastavení a oprávnění. Dále je možné vytvářet šablony se společným nastavením a dědit z nich libovolný počet konfigurací, nebo rozdělit postup nasazování na jednotlivé části, které lze spouštět buďto paralelně, nebo postupně.

Jednotlivé funkce se rozdělují na několik skupin. Hlavní skupinou je „Technologická informovanost“. Jedním z důvodů, proč se TeamCity nazývá „Inteligentní server CI“,

je jeho přístup k integraci. Tento nástroj podporuje všechny možné případy. Jako příklad je zde vysvětlována podpora projektů Visual Studio, kde poskytuje automatickou detekci verze nástrojů, podporuje testování frameworku, pokrytí kódem, analýzu statických kódů a další. Celou tuto podporu získává zákazník bez instalace pluginů a bez úprav skriptů.

TeamCity dokáže flexibilně přizpůsobit správu uživatelů, včetně přiřazování uživatelských rolí, třídění uživatelů do skupin, různé způsoby ověřování uživatelů a protokolu se všemi uživatelskými akcemi pro průhlednost veškeré činnosti na serveru. Více popisovaných skupin, které charakterizují veškeré výhody používání tohoto nástroje je popsáno v [39] a [40].

V rámci testování je TeamCity využíván pro spouštění automatizovaných testů. Po správném nakonfigurování uživateli stačí pouze stisknout tlačítko „Run“, vyplnit povinná pole jako IP adresa testované aplikace a spustit. Celý proces testování je pro něj black box a vystačí si pouze s dosaženými výsledky a logy. To je obrovskou výhodou, jelikož během agilního vývoje je tento proces využívám několikrát denně a nezáleží na osobě, která ho spouští. Získané výsledky jsou vždy v přehledné formě, a pokud není zcela jasné, proč nastala určitá chyba, přidružené logovací soubory poskytují dostatečně detailní informace o průběhu testovacího skriptu. Využívání tohoto nástroje je možné vidět na obrázku 23.



Obrázek 23 - Ukázka nástroje TeamCity. *Zdroj: Autor s využitím výstupu informačního systému.*

3.4 PRACTITEST

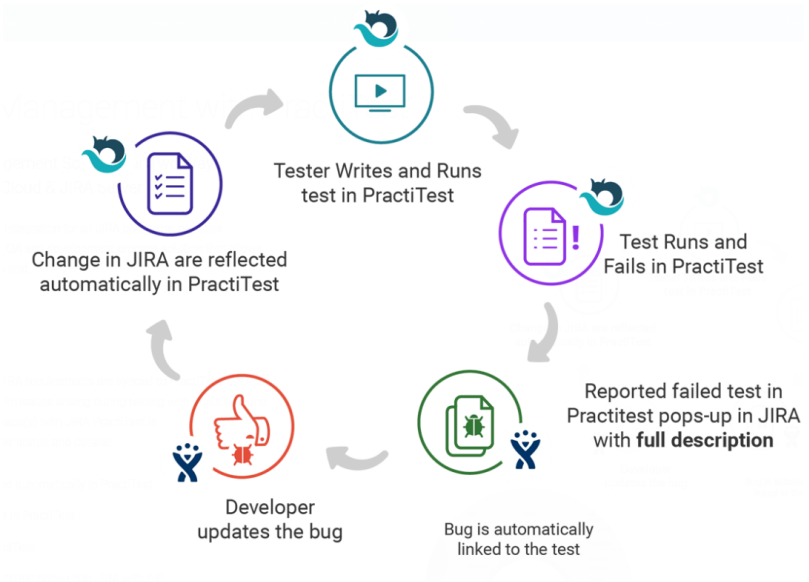
Jedná se o nástroj, který je určen pro QA & Test Case Management. Spravuje proces ověřování kvality a testování, zároveň umožňuje kontrolovat své testovací úkoly, a získávat tím úplný přehled o všech výsledcích.

Umožňuje vytvořit manuální testy a organizovat je na základě cyklů, sprintů atd. Tím podporuje agilní přístup k testování. Bezproblémově integruje manuální testování s procesy automatizace a CI. Realizuje opětovné testování a korelace výsledků mezi různými vydáními a produkty. Zajišťuje reálný pohled na projekt pomocí přizpůsobitelných dashbordů a reportů. Dokáže vložit své živé panely do externích systémů, jako jsou wikis, confluence nebo webové portály. Exportuje své informace,

které je možné zpracovávat dalšími rozsáhlými statistickými a reportovacími softwary.

PractiTest je uzpůsoben k zachycení a organizaci jednotlivých požadavků. Pomocí této aplikace lze propojit požadavky zákazníka s testy, aby bylo dosaženo lepšího zviditelnění stavu testovaného projektu. Obrovskou výhodou tohoto softwaru je správa chyb v PractiTestu nebo integrace na externí systémy, jako jsou Jira, Pivotal Tracker, Redmine atd. To znamená, že tester dokáže vytvořit chybu přímo ze zkušebních běhů na základě zjištěných informací.

V praxi je tento způsob velice často využíván. Tester postupuje podle zadaného testovacího případu v PractiTestu. Po nalezení defektu, umožní PractiTest zadat chybu do externího systému, v tomto případě do nástroje Jira. Po zadání a uložení chyby v systému Jira je tester schopen dále pokračovat v testování daného případu. Zároveň je u daného scénáře vytvořeno propojení, které odkazuje na nalezený defekt v externím systému. Proces propojení lze vidět na obrázku 24.

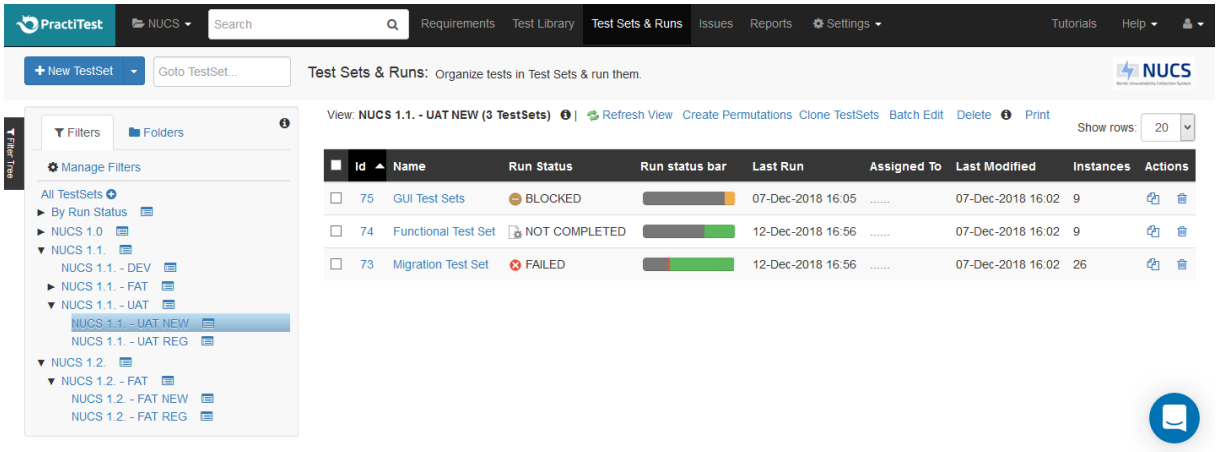


Obrázek 24 - Proces zadání chyby do externího systému. *Zdroj: [42]*

Veškeré rozhraní v PractiTestu je tvořeno pomocí inteligentních filtrů. Pomocí nich lze uspořádat testy, issues a požadavky na základě různých kritérií současně – pomocí komponent, verze, typu nebo jakéhokoliv jiného kritéria, které je vyžadováno. To

zároveň umožňuje identifikovat a odstranit duplicitní informace. Nástroj také dokáže sledovat trvání testovacích běhů tím, že tester spustí daný test. Poté se začne počítat doba provádění tohoto scénáře až do jeho dokončení. Více doplňujících informací o funkčnostech a propojení s dalšími nástroji je možné dohledat v [41] a [42].

Během testování ve firmě Unicorn Systems je tento nástroj využíván nepřetržitě. Nejenže umožňuje zobrazit v přehledných grafech reálný pohled na rozsah testování systému, ale zároveň dokáže mapovat požadavky na jednotlivé testovací scénáře, což umožňuje managementu zkontrolovat pokrytí systému jednotlivými testy. Tento software rapidně zvyšuje rychlost, kvalitu, efektivitu a flexibilitu testování. Příklad využití tohoto nástroje na projektu NUCS je možné nalézt na obrázku 25.



Obrázek 25 - Ukázka nástroje PractiTest. *Zdroj: Autor s využitím výstupu informačního systému.*

3.5 SOURCETREE

Tento nástroj je zde zmiňován převážně kvůli automatizovaným testům. Sourcetree je nástroj od společnosti Atlassian a jedná se bezplatného klienta Gitu pro Windows a Mac. Zjednodušuje interakci s úložišti Git a zároveň vizualizuje a spravuje repozitáře pomocí jednoduchého rozhraní Git GUI.

Jelikož všechny automatizované skripty jsou uloženy v Gitu, je potřeba nástroj, který umožní spravovat činnosti nad těmito repozitáři, k tomu právě slouží nástroj Sourcetree. Ten umožňuje grafický přehled nad jednotlivými změnami testovacích

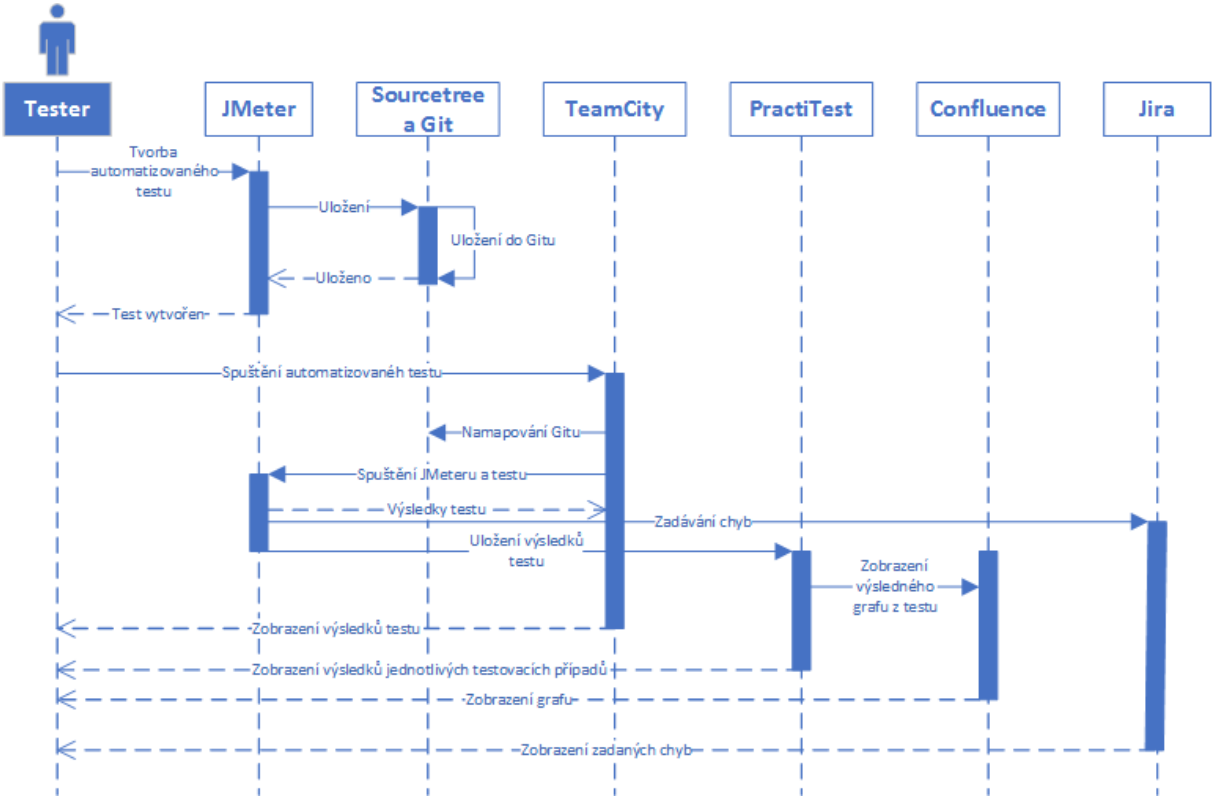
skriptů, zobrazuje datum změň a osobu, která dané změny provedla. Zároveň zvyšuje produktivitu týmu pomocí služby Bitbucket. Bitbucket dává týmům jedno místo pro plánování projektů, spolupráci na kódování, testování a nasazení. Více informací o tomto softwaru je možné nalézt v [43].

3.6 Propojení jednotlivých nástrojů

Jednotlivé nástroje je možné propojit, což zajištuje efektivní manipulaci i řízení testovacího procesu. Pokud tester chce vytvořit automatizovaný test, využívá k tomu nástroj JMeter, po vytvoření tohoto testu dojde přes software Sourcetree k uložení do Git repozitáře. Tím je proces tvorby automatizovaného testu a jeho následné uložení dokončeno.

Pokud kdokoliv chce spustit automatizovaný test, využije k tomu služeb TeamCity, kde je provedeno mapování na konkrétní automatizovaný skript. Pomocí tlačítka Run spustí TeamCity agenta, ten si z Gitu stáhne aktuální data a spustí Apache JMeter s daným automatizovaným skriptem. Tento test je proveden a jednotlivé výsledky jsou vráceny do TeamCity. Zároveň se jednotlivé výsledky propisují do PractiTestu, kde jsou zhotoveny jednotlivé grafy konkrétního běhu. Tyto grafy se dále propisují do Confluence, kam mají přístup manažeři celého projektu. Ti jsou schopni zkontrolovat kvalitu dodávaného softwaru.

Pokud je v rámci automatizovaného testu nalezena chyba, JMeter dokáže automaticky zadat chybu do nástroje Jira. Tato funkce se v praxi využívá zřídka, jelikož ne vždy se jedná o adekvátní chyby, které by měly být zadány (například výpadek sítě apod.). Pro lepší názornost, obrázek 26 detailně zobrazuje propojení jednotlivých nástrojů a činností, které byly popisovány v této ukázce.



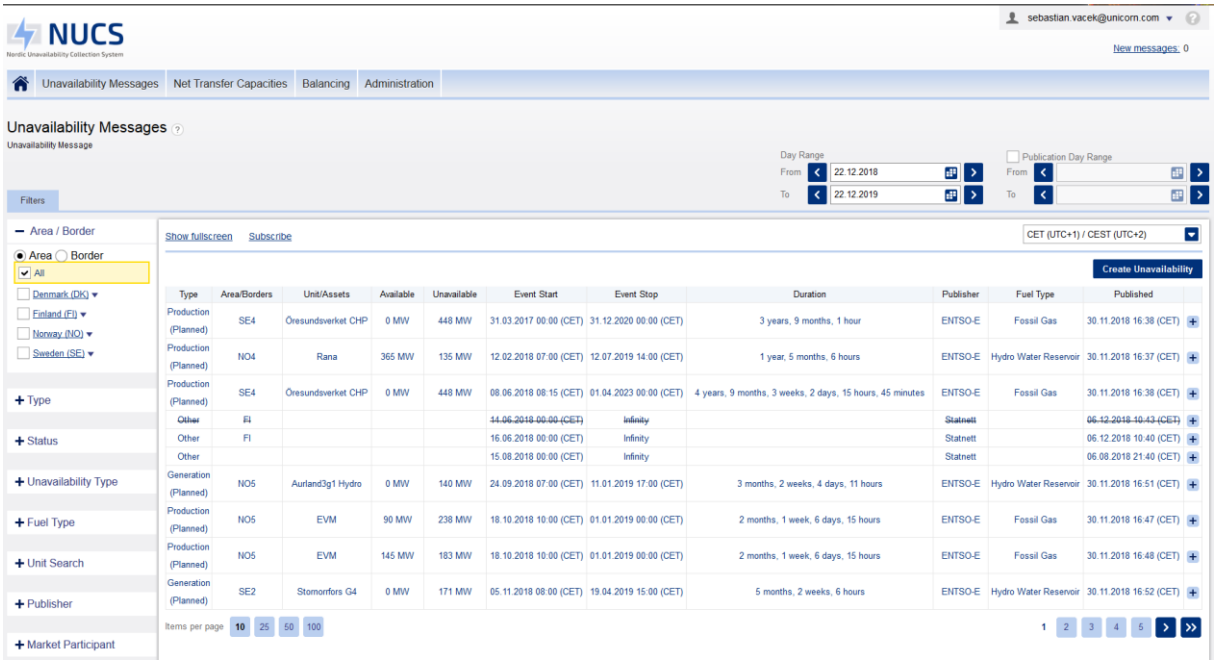
Obrázek 26 - Schéma propojení jednotlivých nástrojů. *Zdroj: Autor*

4 PRAKTICKÝ PŘÍKLAD

V této kapitole je uveden praktický příklad tvorby automatizovaného testu, jeho propojení s vybranými nástroji a závěrečné vyhodnocení tohoto testu. Zároveň také poukazuje na časovou náročnost manuálního testování a automatizovaného testování. Tento příklad je v rámci iniciativy NUCS, která je podrobněji popsána v následují části diplomové práce.

Účelem systému Nordic Unavailability Collection System (NUCS) je poskytovat službu provozovatelům přenosových soustav a účastníkům trhu, kteří zajišťují shromažďování informací ze severského trhu s elektrickou energií (angl. Nordic Electricity Market). Výpočet předpokládaných hodnot přenosových kapacit je založený na získávání informací o nedostupném přenosu, tyto hodnoty jsou následně přeposílány do ENTSO-E Transparency Platform za účelem plné transparentnosti dle nařízení Evropské unie. Systém Nordic Collection poskytuje grafické uživatelské rozhraní, které umožňuje anonymnímu uživateli prohlížet publikované informace v systému. Provozovatelé přenosových soustav a účastníci trhu navíc mohou vytvářet nebo měnit informace v systému. [44]

Jak si lze všimnout z obrázku 27, vyvíjená aplikace se skládá z několika oblastí. Pokud je uživatel přihlášen, může vidět tři základní záložky. Unavailability Messages doména slouží k vytváření a evidenci jednotlivých nedostupností. Net Transfer Capacities je doména, která slouží k agregovanému pohledu na předpokládané přenosové kapacity. Důležitou záložkou je Administration, jelikož slouží uživatelům ke správné konfiguraci této aplikace.



Obrázek 27 - Náhled aplikace NUCS. *Zdroj: Autor s využitím výstupu informačního systému.*

Prvním úkolem bylo navrhnout rozsáhlé testovací případy (angl. Test Cases), které pokryjí hlavní funkčnosti softwaru. K tomuto kroku byla využita funkční specifikace, která je nedílnou součástí vyvíjeného produktu. Podle finální verze testovací dokumentace je celkový počet případů 125. Jednotlivé případy pokrývají hlavní funkčnosti softwaru. Nejvíce takto pokrytou částí je právě doména Unavailability Messages (Outages), protože právě tato doména je nejdůležitější z hlediska požadavku zákazníka.

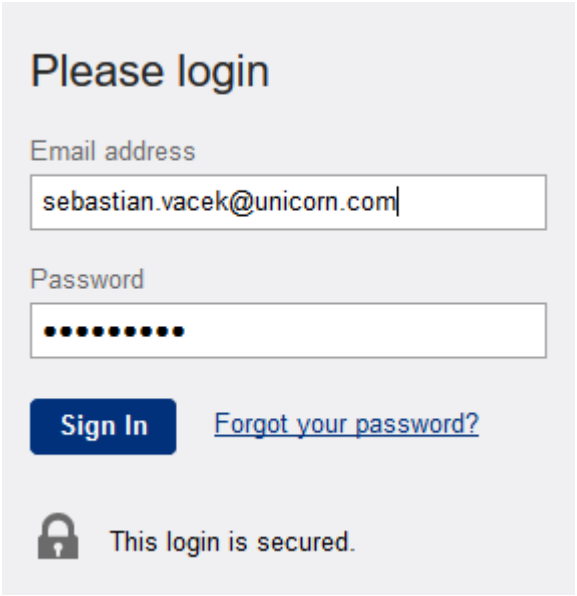
Následnou činností bývá zpravidla samotné testování aplikace. Pokud se vyvinou jakékoliv nové testovací případy, vždy se nejprve musí testovat manuálně, aby byla zajištěna nutná kvalita softwaru.

4.1 Manuální test

Manuální testovaní softwaru NUCS lze znázornit na reprezentativním příkladu jednoho testovacího případu. Nejdůležitější součástí této kapitoly je zhodnocení časové náročnosti daného reprezentativního případu, který je zároveň vyvinut v nástroji JMeter jako plně automatizovaný test.

Obsahem scénáře tohoto testovacího případu je v první řadě nutnost přihlášení do systému. Dále následují tyto kroky. Nakonfigurovat daný software dle potřeby, v tomto případě tak, aby provozovatel přenosové soustavy mohl zadat plánovanou nedostupnost přenosové sítě (angl. Planned Transmission Grid Unavailability). Poté se přihlásit do systému jako provozovatel přenosové soustavy a zadat určitou nedostupnost. Dalším krokem tohoto scénáře je kontrola zpracování a následné správné publikování dat, které charakterizují právě tuto nedostupnost. Posledním článkem, který je zároveň nad rámec tohoto testovacího případu a není z hlediska funkčnosti aplikace důležitý, je stažení všech možných souborů (Excel, CSV a XML), které reprezentují danou nedostupnost. Tento testovací scénář je označován jako „Happy Day“, jelikož kontroluje hlavní funkcionalitu systému a simuluje pouze reálné používání dané aplikace.

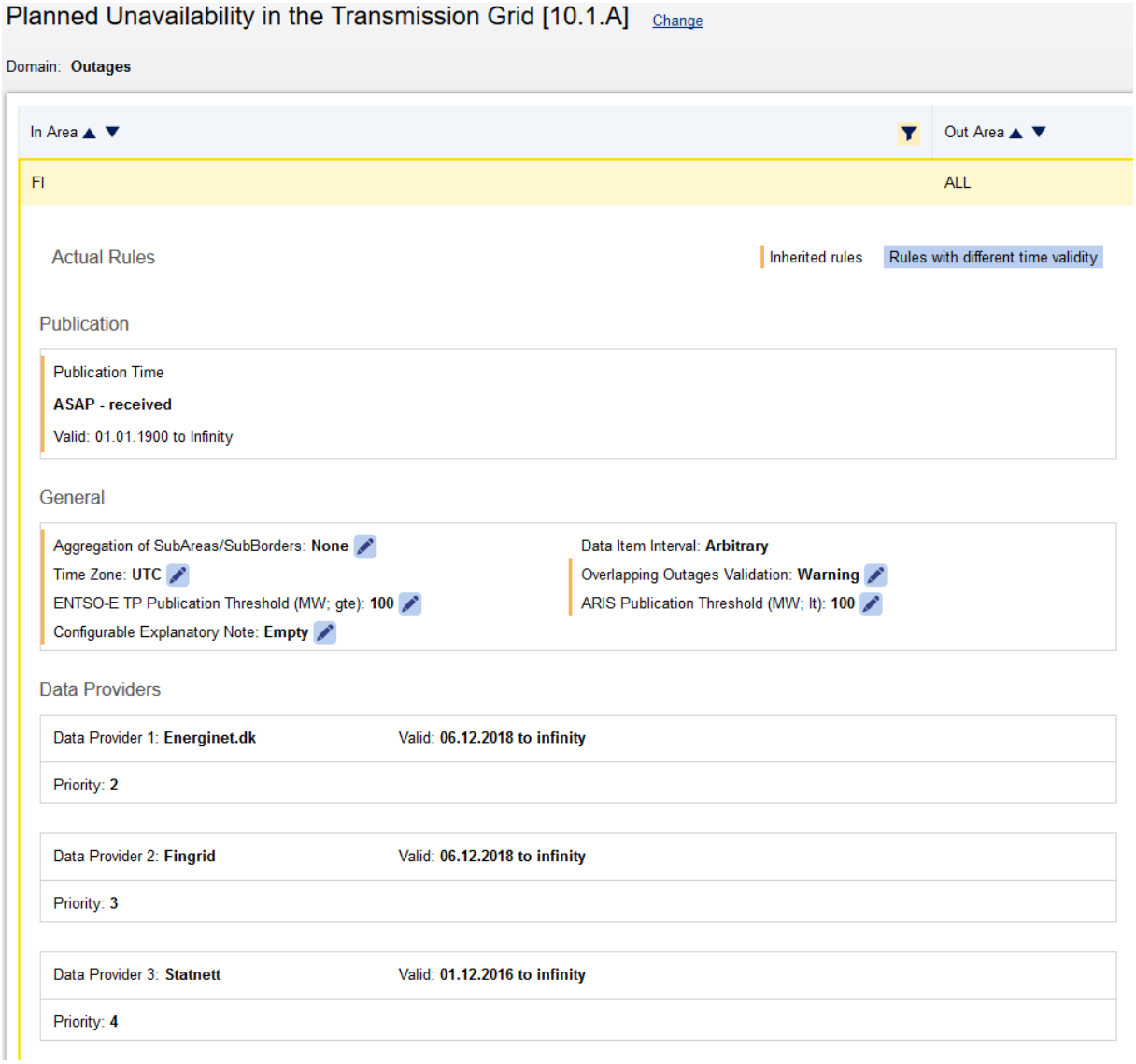
Přihlášení do systému je jednoduchou činností, jak si lze všimnout z obrázku 28, pokud uživatel je již registrován, pak stačí pouze vyplnit přihlašovací údaje a přihlásit se. Celý tento krok nezabere více než pár sekund. Pro tento testovací případ je nutné, aby přihlašovatel do systému měl administrativní práva.



Obrázek 28 - Přihlášení do systému. *Zdroj: Unicorn Systems*

Po přihlášení do systému je nutné nastavit aplikaci tak, aby byla schopná zpracovávat soubory, které jsou vytvářeny provozovatelem přenosové soustavy. Tento postup

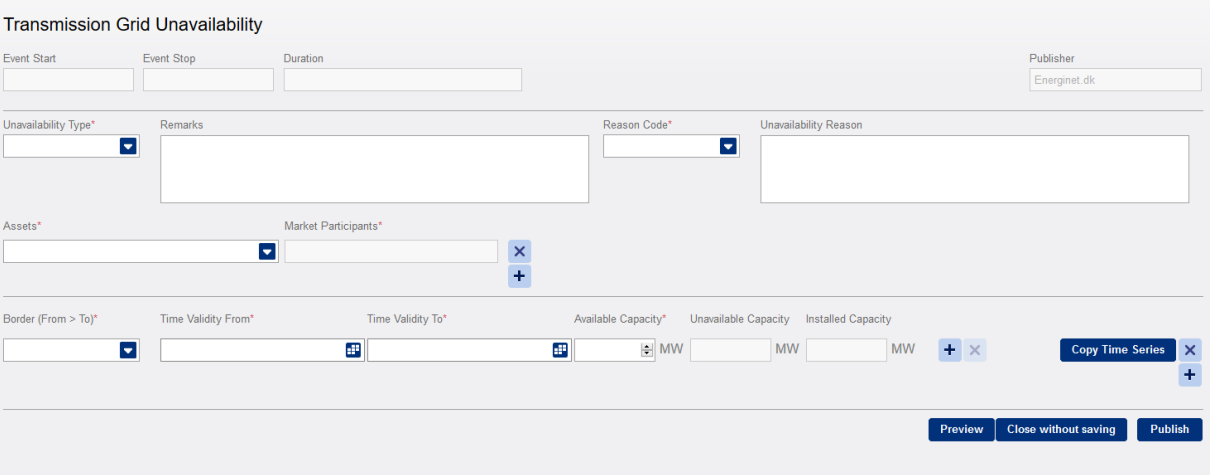
zahrnuje přidělení území k provozovateli, který bude moci zadávat hodnoty za danou zemi. Jednoduše řečeno, provozovateli ve Finsku jsou přiřazeny finské hranice a vnitřní rozdělení země. Dále je nutné povolit provozovateli možnost zadávat informace pro daný typ nedostupnosti. Celý tento proces zabere při manuálním testování přibližně 15 až 30 minut. Typický příklad přidělení konkrétního provozovatele k danému typu nedostupnosti je možné zhlédnout na obrázku 29.



Obrázek 29 - Příklad přidělení provozovatele k typu nedostupnosti. Zdroj: Autor s využitím výstupu informačního systému.

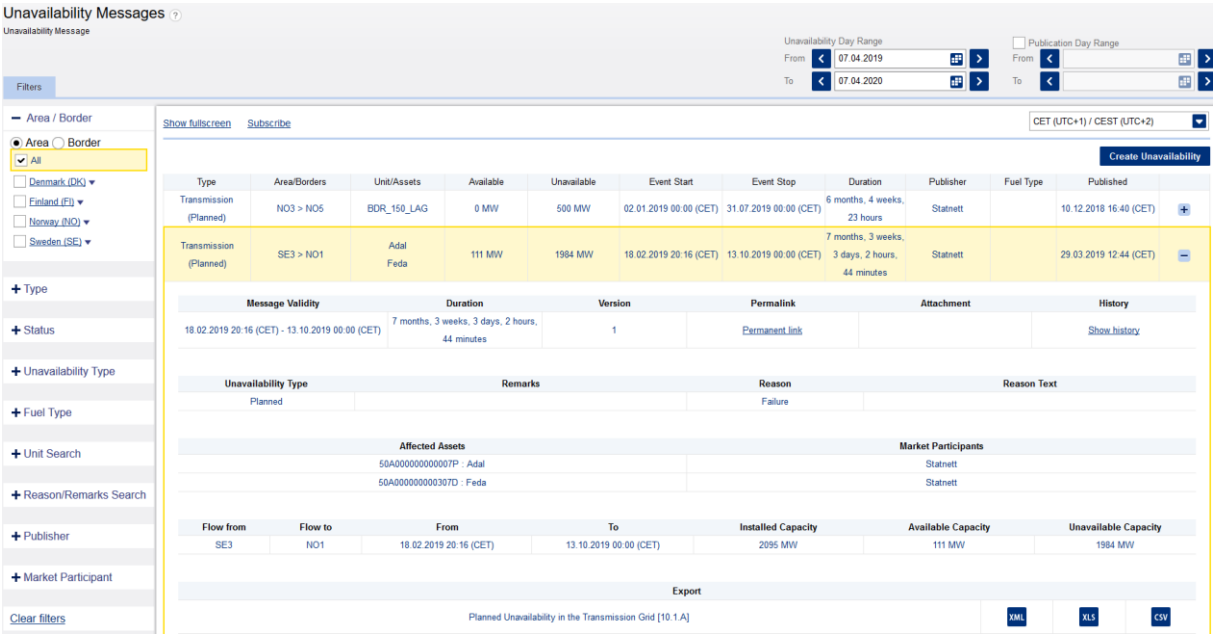
Nejdůležitější funkčností systému je vytvoření samotné nedostupnosti. To se provádí pomocí GUI, které je k tomu přizpůsobeno. Není dán přesný obsah, který je nutné vyplnit pro vytvoření nedostupnosti. Jednotlivé testy se mohou lehce lišit. Průměrné

testování tohoto scénáře zabere 5 až 10 minut. Z vyplněné šablony, kterou lze vidět na obrázku 30, vznikne XML soubor. Ten je následně zpracováván a publikován v aplikaci NUCS.



Obrázek 30 - Náhled šablony pro vytvoření nedostupnosti. *Zdroj: Autor s využitím výstupu informačního systému.*

Výsledná kontrola zpracování a následného publikování dat záleží na velikosti a množství informací obsažených v dané nedostupnosti. Průměrné časové hodnoty této činnosti jsou opět mezi 5 až 10 minutami. Jak si lze všimnout na obrázku 31, všechny publikované informace o jednotlivých nedostupností se zobrazují v záložce Unavailability Messages.



Obrázek 31 - Náhled publikovaných hodnot. *Zdroj: Autor s využitím výstupu informačního systému.*

Jednotlivé informace o průběhu testovacích případů jsou zaznamenávány ručně do aplikace PractiTest, která slouží k řízení veškerých testů během iniciativy NUCS. Z této aplikace následně vycházejí veškeré dosažené statistiky o výsledném testovacím kole. Pokud bereme v úvahu zapisování do PractiTestu, jedná se opět pouze o zlomky minut, kdy uživatel označí stav daného testu, ale i to je už čas strávený navíc. Takto označené testovací případy, které souvisí s vyvíjenou aplikací NUCS je možné nalézt na obrázku 32.

Test Instances

Reset Filter | Filters | Batch Edit | Fast Run | Delete | Parameters | Columns | 60

Run	Type	Test Instance Name	Duration Estimate	Last Run Duration	Run status	Last Run	Steps Status Bar	JIRA Issue ID
Run		Message is successfully computed and published	00:00:00	00:17:24	PASSED	12-Apr-2018 18:51		
Run		Message is successfully computed and published	00:00:00	00:00:16	PASSED	12-Apr-2018 18:51		
Run		Message is successfully computed and published	00:00:00	00:00:13	PASSED	12-Apr-2018 18:52		
Run		Message is successfully received	00:05:00	00:00:28	PASSED	12-Apr-2018 18:07		
Run		Message is successfully received	00:05:00	00:00:19	PASSED	12-Apr-2018 18:08		
Run		Message is successfully received	00:05:00	00:01:00	PASSED	20-Apr-2018 10:36		
Run		Message is successfully received	00:05:00	00:00:12	PASSED	12-Apr-2018 18:10		
Run		Message is successfully received	00:05:00	00:01:16	PASSED	12-Apr-2018 18:10		
Run		Message is successfully received	00:05:00	00:00:11	PASSED	12-Apr-2018 18:11		
Run		Message is successfully received	00:05:00	00:00:10	PASSED	12-Apr-2018 18:11		
Run		Message is successfully received	00:05:00	00:00:12	PASSED	12-Apr-2018 18:12		
Run		Message is successfully received	00:05:00	00:00:11	PASSED	12-Apr-2018 18:12		
Run		Message is successfully uploaded	00:00:00	00:08:27	PASSED	10-Apr-2018 12:45		
Run		10.1.A&B - Export	00:05:00	00:00:00	N/A	20-Apr-2018 08:11		OUT OF SCOPE
Run		10.1.C - Export	00:05:00	00:00:00	N/A	20-Apr-2018 08:11		OUT OF SCOPE
Run		7.1.X - Export	00:05:00	00:00:00	N/A	20-Apr-2018 08:11		OUT OF SCOPE
Run		15.1.X - Export	00:05:00	00:00:00	N/A	20-Apr-2018 08:11		OUT OF SCOPE
Run		Export	00:05:00	00:00:32	PASSED	10-Apr-2018 21:01		
Run		Export	00:05:00	00:00:16	PASSED	11-Apr-2018 06:10		
Run		Export	00:05:00	00:00:12	PASSED	11-Apr-2018 06:11		
Run		Create message via GUI	00:05:00	00:00:00	FAILED	22-Apr-2018 12:23		NUCS-172, NUCS-175

Obrázek 32 - Příklad zapisování výsledků do PractiTestu.[4] *Zdroj: Autor s využitím výstupu informačního systému.*

Celkové vyhodnocení časové náročnosti pro daný testovací případ se pohybuje okolo 30 až 50 minut. Pokud se uvažuje o manuálním testování celé aplikace, jedná se přibližně o týden až týden a půl práce. Nicméně pro řádné srovnání s automatizovanými testy byl proveden časový rozbor pro konkrétních 18 testovacích případů, pro které byla vyhodnocena časová náročnost manuálního testování na 12 až 16 hodin.

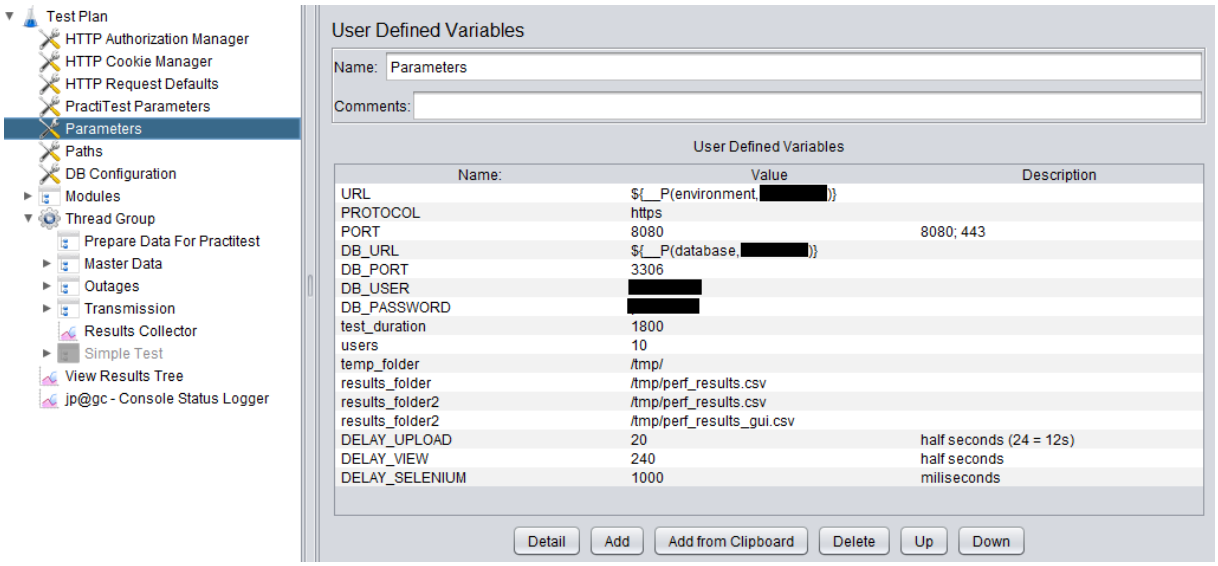
[4] Jednotlivé hodnoty sloupce „Duration“ neodpovídají skutečným hodnotám naměřeným během manuálního testování, jelikož tyto hodnoty jsou získávány pouze z doby, kdy byla spuštěna a ukončena daná instance testu v PractiTestu.

4.2 Automatizovaný test

Po důkladné analýze všech testovacích případů bylo rozhodnuto, že je nutné automatizovat základní funkčnosti dané aplikace. Využito přitom bylo zkušeností z automatizovaných testů, které se vyvíjely pro podobnou aplikaci ENTSO-E Transparency Platform. Z tohoto projektu byly vybrány vhodné automatizované skripty, které se následně využily i při tvorbě automatizovaného testu pro projekt NUCS.

Jedním ze zvolených testovacích případů byl totožný scénář pro zakládání nedostupnosti přenosové sítě, který byl popsán během manuálního testování. To umožňuje identifikovat časovou náročnost obou těchto metod a dosáhnout tak potřebných výsledků k jednotlivým analýzám.

Prvním úkolem během vývoje automatizovaného testu bylo nutné identifikovat základní parametry, které se budou během celého testu využívat. Jedná se především o URL adresu testované aplikace a URL adresu její databáze, z důvodu bezpečnosti jsou na obrázku níže tyto informace nedostupné. Rovněž i přístupové údaje, které jsou zapotřebí k přístupu na jednotlivá prostředí. Jednotlivé URL adresy jsou navrženy tak, že jejich hodnotu je možné získávat dynamicky z nástroje TeamCity a to pomocí dvou proměnných ${__P(**environment**, default value) a ${__P(**database**, default value). Veškeré tyto parametry se nacházejí v již zmiňovaném User Defined Variables. Všechny jsou načtené na začátku každého běhu a využívají se v každé části automatizovaného testu. Identifikování těchto parametrů zabere pár sekund, ale mapování do jednotlivých částí je poměrně více časově náročné. Časová náročnost je odhadována na 5 až 10 minut u individuálního scénáře, samozřejmě záleží na složitosti. Takto sestavené parametry je možné vidět na obrázku 33.



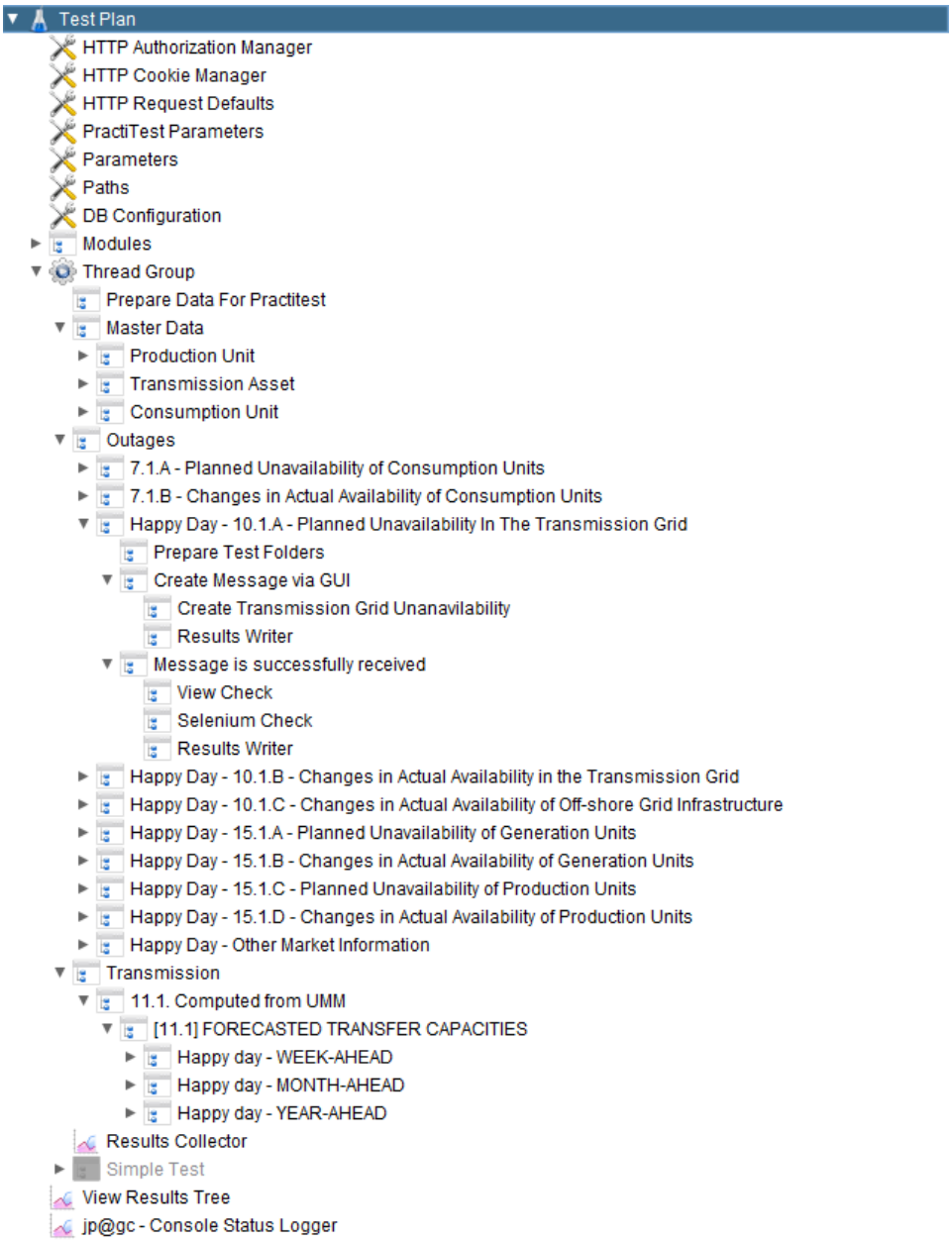
Obrázek 33 - Základní parametry. *Zdroj: Autor*

Po zajištění všech podstatných parametrů je možné spustit nahrávání testerova postupu podle scénáře, který byl identifikován během manuálního testu. Po nahrání dílčích kroků, byly tyto kroky transformovány do podoby, která lépe charakterizuje jejich účel. Zároveň byly rozděleny do separátních testovacích skript (z historického řešení zvané Modules, tedy moduly) kvůli znovupoužitelnosti a jednodušší detekci případné chyby.

Vznikla tak struktura, která reprezentuje jednotlivé testovací scénáře, a která umožňuje několikanásobné použití jednotlivých částí. Zároveň je rozdělena do několika úrovní podle testovaných sekcí (Master Data, Outages a Transmission). Individuální sekce využívají parametry, které jsou pro danou skupinu klíčové. Všechny podsoubory právě dědí tyto informace z rodičovského uzlu. Pro tento konkrétní scénář je nejdůležitější Outages sekce, která v sobě obsahuje testovací scénář pro vytvoření nedostupnosti přenosové sítě. Detailní pohled na danou strukturu je zobrazen na obrázku 34.

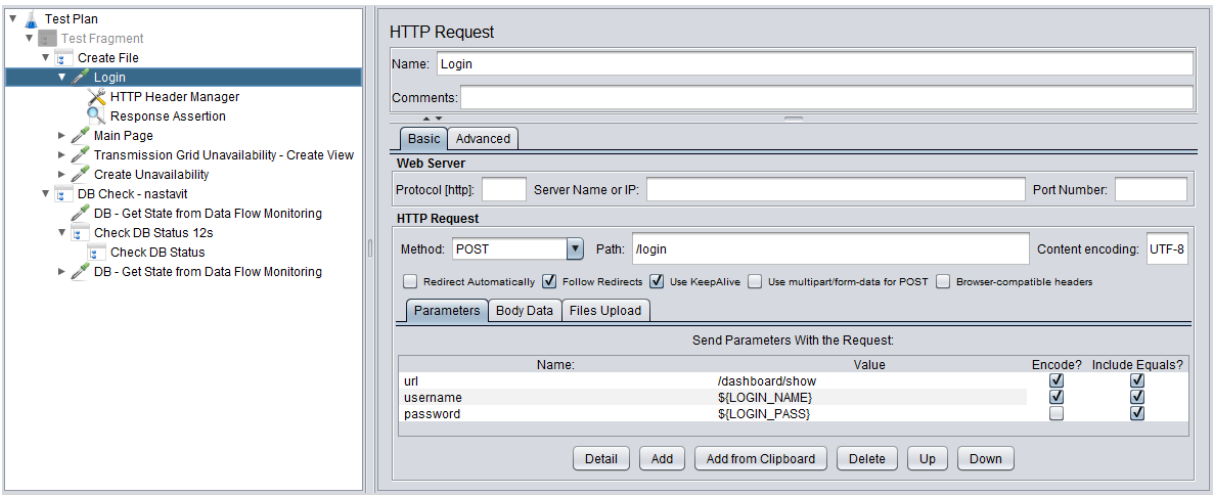
Happy Day scénář na vytvoření plánované nedostupnosti přenosové sítě obsahuje konkrétní moduly. První modul slouží k vytvoření složky, do které se budou dílčí výsledky ukládat. Následující modul je určen přímo k vytvoření dané nedostupnosti, spolu se zapisovačem výsledků z dané části testu. V posledním oddílu tohoto testu se

nachází kontrola vytvořené a zároveň publikované nedostupnosti. Tato validace je provedena pomocí dvou metod. První z nich je ověření pomocí requestu, který vrací data z kontrolované stránky. Druhou metodou je použití pluginu Selenia, kdy systém pracuje přímo jako tester v internetovém prohlížeči. Zobrazí si požadovanou stránku, na které se nacházejí konkrétní informace a ověří je. Více detailních informací o jednotlivých metodách je popsáno v další části této práce. Součástí tohoto modulu je rovněž zapisovač výsledků, který je prvním náznakem znovupoužitelnosti jednotlivých částí skriptu.



Obrázek 34 - Struktura nově vytvořeného automatizovaného testu. *Zdroj: Autor*

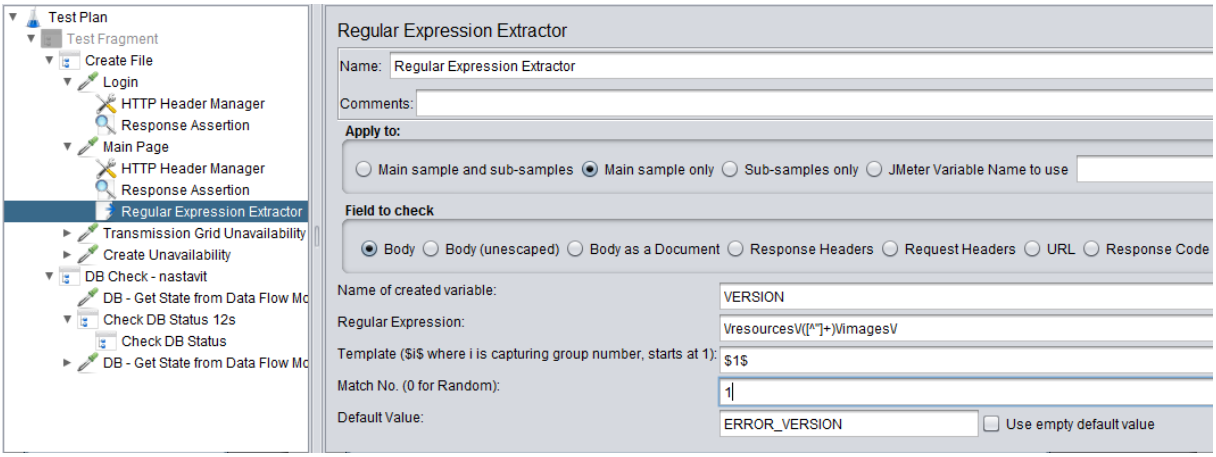
Vytvoření nedostupnosti přenosové sítě je v separátním testu. Tento skript zahrnuje postup, kdy jako první se ověřuje přihlašování do systému. Pomocí dynamických parametrů ${LOGIN_NAME} a ${LOGIN_PASS}, které jsou posílány do aplikace, se zjišťuje, zda přihlášení proběhlo podle očekávaného výsledku. Pokud aplikace vrátí výsledek „ok“, test pokračuje dalším krokem. Časová náročnost této části testu se pohybuje okolo půl hodiny, jelikož je žádoucí, aby přihlašování bylo možné použít i v jiných částech vytvářeného skriptu. Výhodou je, že během přihlašování se mění pouze přihlašovací údaje, ostatní parametry zůstávají shodné. Takto vytvořený skript je možné vidět na obrázku 35.



Obrázek 35 - Login. *Zdroj: Autor*

Další krok obsahuje pouze zobrazení hlavní stránky testované aplikace. Na této stránce se nachází text „Unavailability Messages“, který je kontrolován. Dále je součástí této stránky informace o verzi dané aplikace. Ta je z hlediska následujícího kroku, kde jsou kontrolovány názvy formuláře pro vyplnění nedostupnosti, velice důležitá. Verze je totiž přidávána do URL adresy, na kterou daný request směřuje. Pro zjištění informace je využit Regular Expression Extractor (RegEx). Ten je přidán do HTTP Requestu na zobrazení hlavní stránky. RegEx dokáže pracovat s regulárními výrazy. Do proměnné s názvem VERSION je schopný uložit hodnotu, která se nachází ve zdrojovém kódu webové stránky mezi /resources/ a /images/. Pokud taková hodnota neexistuje, je do proměnné vložen chybový text ERROR_VERSION.

Zobrazení a kontrola hlavní stránky je poměrně časově nenáročné. Doba vytvoření se pohybuje okolo 10 minut. Nicméně součástí tohoto kroku je i RegEx, který musí být správně nastaven a otestován tak, aby vracel vždy požadovanou hodnotu. Tím se doba vytvoření této části testu zvyšuje a je odhadována na 45 minut. Takto sestavený Regular Expression Extractor lze vidět na obrázku 36.

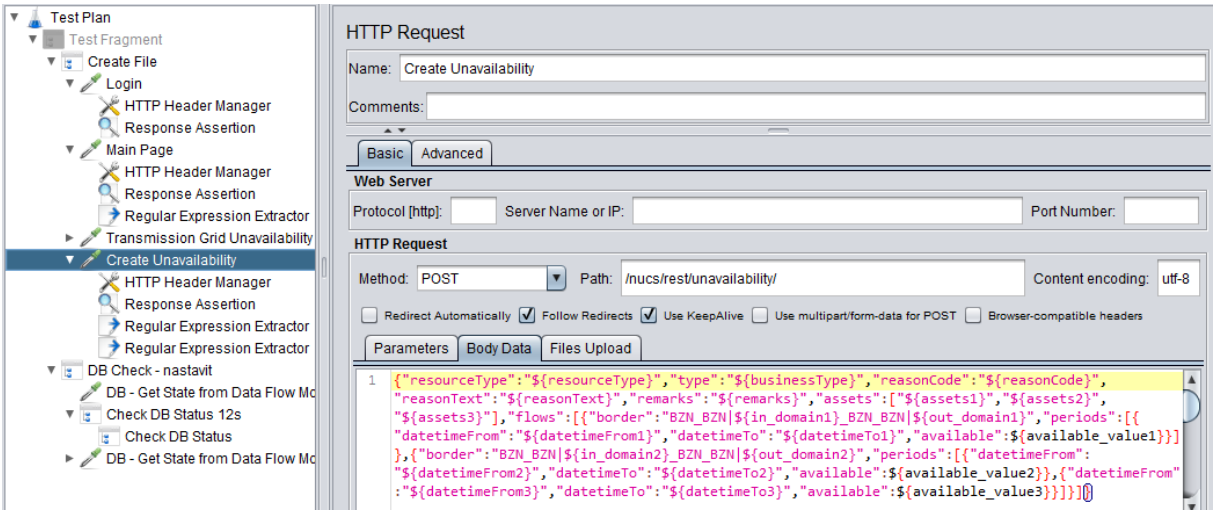


Obrázek 36 - Regular Expression Extractor. *Zdroj: Autor*

Pro další část testovacího scénáře je vytvořen request se specifickou cestou, která je v následujícím tvaru: „/resources/${VERSION}/angular/views/outage-domain/umm-form-transmission.html“. Tato cesta odkazuje na stránku s formulářem. V ní je použit dynamický parametr ${VERSION} a jeho hodnota je získávána z předchozího kroku pomocí RegExu. Jednotlivé názvy musí být shodné, aby následovalo správné vložení získané informace. Pokud je cesta náležitě sestavena, aplikace vrátí výsledek, u kterého jsou kontrolována jednotlivá textová pole formuláře na založení nedostupnosti. Doba tvorby této části skriptu se pohybuje okolo 30 minut. Jelikož parametry závisí na předchozím kroku, je tím vývoj časově navyšován. Rovněž je zde kontrolováno obrovské množství informací, což také zvyšuje dobu vzniku tohoto requestu.

Následující krok obsahuje situaci, kdy pomocí metody POST je konstruován request, který vyplňuje a odesílá hodnoty pro vytvoření nedostupnosti přenosové sítě. Jednotlivé parametry musí být dynamické, proto se zde vytvořily z pevně daných

hodnot proměnné, které jsou vyplňovány například už během spouštění kořenového modulu. Takto vytvořený request je zobrazen na obrázku 37.



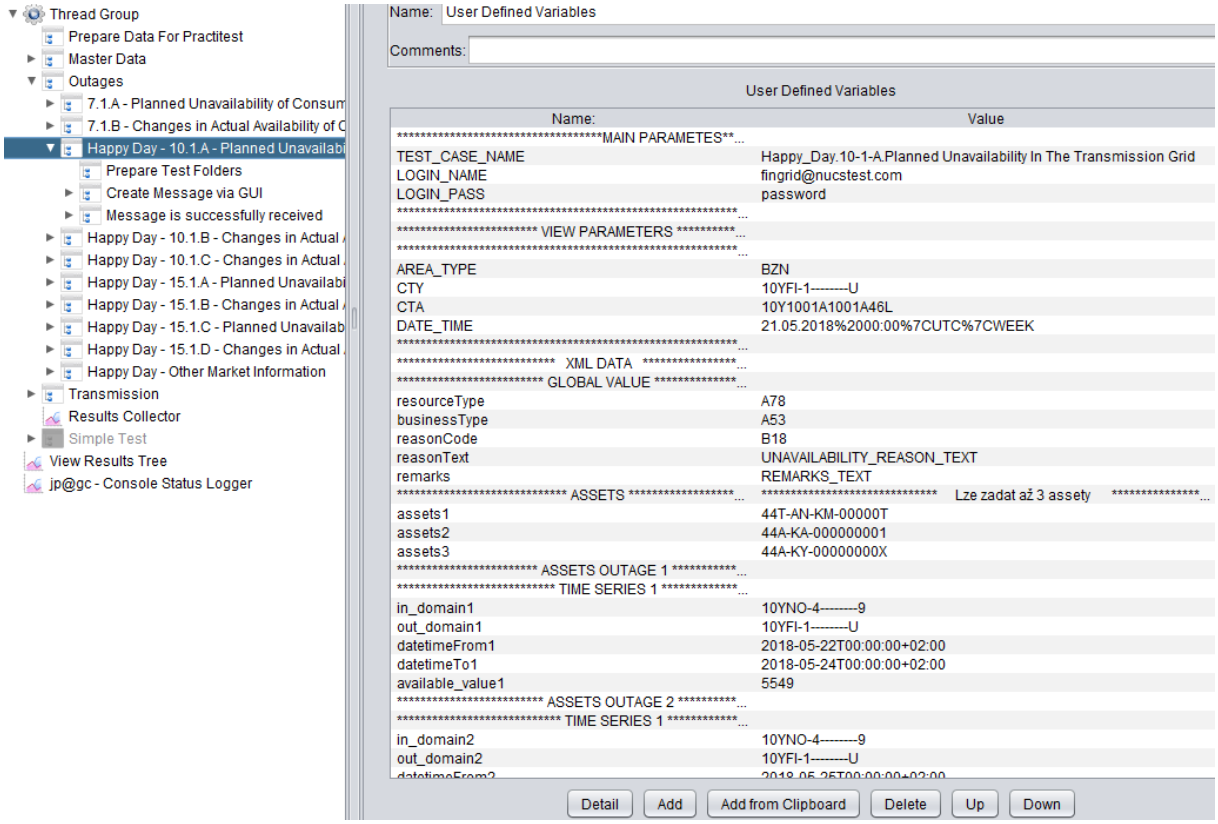
Obrázek 37 - Vytvoření nedostupnosti s dynamickými parametry. *Zdroj: Autor*

Pokud jsou použity dynamické parametry, usnadňuje to práci se vstupními hodnotami, které mohou být předávány i například pomocí csv souboru. Osoba, která spouští tento test, nemusí vůbec zasahovat do vytvořeného skriptu a může změnit hodnoty pouze ve zvoleném souboru nebo pomocí vyplněných parametrů u spouštěcího nástroje, jako je TeamCity apod.

Celý tento postup je složitější na vývoj, nicméně přidaná hodnota takto upraveného testu se projeví vždy, kdy je nutné měnit jednotlivé parametry. Jak si lze všimnout na obrázku 38, jednotlivé hodnoty jsou v tomto případě přidávány už během spouštění Parameterized Controlleru v hlavním testovacím skriptu. U tohoto typu testů je to žádoucí, jelikož všechny parametry se vyskytují na jednom místě a není tak potřeba otevírat jiné testovací skripty. Po odeslání takto vyplněného formuláře vzniká pomocí aplikace NUCS xml soubor, který je následně validován a zpracován v této aplikaci.

Časová náročnost této dynamické části testovacího skriptu se pohybuje okolo 2 až 3 hodin. Je opravdu složité správně nastavit jednotlivé parametry nedostupnosti, a následně je dynamicky předávat v daném requestu. Nicméně tento request je použit v několika testovacích scénářích, kde už stačí pouze vyplnit požadované hodnoty, a tím se doba takto vytvořeného testu rapidně snižuje. Proto se časová náročnost již

vytvořených dynamických částí skriptu pohybuje maximálně okolo 30 minut (v závislosti na vyplňovaných parametrech).



Obrázek 38 - Parameterized Controller. *Zdroj: Autor*

V poslední části separátního skriptu na vytvoření nedostupnosti je databázová kontrola, zda je takto vyplněný a odeslaný formulář správně zpracován v aplikaci NUCS. Jedná se pouze o databázové dotazy, které opakovaně kontrolují stav daného xml dokumentu, dokud nenabude stavu PROCESSED nebo REJECTED. Takové dotazy a prostup do databáze se vytvářejí přibližně 30 minut. Požadovaný výsledek, který je zároveň kontrolován, je PROCESSED. Pokud výsledek nabude opačného stavu REJECTED, je testovací scénář ukončen.

Tímto krokem je dokončen separátní testovací skript pouze pro vytvoření nedostupnosti přenosové sítě. Celková časová náročnost takto utvořeného skriptu se přibližně pohybuje okolo 3-5 hodin. Do tohoto časového rozhraní jsou zahrnovány i externí vlivy na testování, jako je výpadek testované aplikace, průběžné nasazování a čistění aplikace apod.

Nyní je potřeba ověřit, zda finální publikování na GUI aplikace NUCS doběhlo úspěšně. To je z hlediska businessu zákazníka klíčové. Proto bylo využito dvou metod, které zajišťují tuto verifikaci. První z nich opět využívá dotazování přes request, který byl doposud používán u všech předchozích kroků. V separátním testovacím skriptu je vytvořena kontrola stránky všech nedostupností, kde pomocí filtru, který je nastaven v URL cestě, vyfiltruje vytvořenou nedostupnost. Response Assertion posléze zjistí, zda zadané hodnoty odpovídají těm publikovaným. Pokud ano, testování pokračuje druhou metodou.

Druhá metoda, která se používá pro kontrolu, je plugin Selenium. Pomocí tohoto rozšíření je možné simulovat jednotlivé scénáře přímo v daném prohlížeči (Chrome, Firefox či Internet Explorer). Pomocí jednoduchého kódování je možné vytvořit požadovaný postup. V tomto případě jde o tyto konkrétní kroky:

- Nejprve je spuštěn zvolený prohlížeč. K tomuto kroku je nutné specifikovat Driver Config, který odkazuje na nainstalované drivery. U tohoto scénáře je využit driver pro prohlížeč Chrome. Prohlížeč je spuštěn s přesně daným rozlišením – 1920x1080.
- Do prohlížeče je vložena URL adresa, která odkazuje na konkrétní vytvořenou nedostupnost. V této adrese je opět využita proměnná, která je dynamicky doplňována z předchozích kroků.
- Po načtení adresy následuje zavření dialogového okna, které se zobrazí po prvním navštívení vyvíjené aplikace.
- V další části skriptu se nachází kliknutí na tlačítko, které umožňuje rozbalení detailních informací u dané nedostupnosti. Ne všechny nedostupnosti obsahují toto tlačítko.
- Poslední, a zároveň jedna z nejdůležitějších částí z hlediska kontroly publikované nedostupnosti je vytvoření screenshotu celé obrazovky prohlížeče. Tento obrázek obsahuje pohled na konkrétní nedostupnost a slouží ke kvalitní kontrole publikovaných dat. V tuto chvíli je visuální kontrola prováděna

prostřednictvím lidské manuální činnosti, nicméně je předpokládáno, že v budoucnosti bude tato kontrola provedena pomocí porovnávání vytvořeného screenshotu s předem známou šablonou. Tento postup je v současné chvíli teprve v počátcích vývoje. Po dokončení by měl být využit na všech dosavadních energetických projektech.

Jednotlivé kroky jsou zaznamenávány ve WebDriver Sampleru, kde je možné využít různé druhy skriptovacích jazyků. Tento modul rovněž obsahuje Response Assertion, který validuje hodnoty z poslední zobrazené stránky. Ověřovány jsou stejné hodnoty, jako v předchozí kontrole, která je prováděna pomocí requestu. Detailnější pohled na možnost využití Selenia v nástroji JMeter je možné nalézt na obrázku 39. Doba k vytvoření dynamické kontroly jednotlivých testů se pohybuje okolo 2 až 3 hodin.

Script Language: javascript

Script (see below for variables that are defined)

```
var selenium = JavaImporter(org.openqa.selenium, java.lang.Integer, org.apache.commons.io)
var delay = selenium.Integer.parseInt("${DELAY_SELENIUM}")

WDS.sampleResult.sampleStart()
try {
//START-----------------------------------------------------------------------------------------------------
//Window resize
WDS.browser.manage().window().setSize(new selenium.Dimension(1920, 1080))

//Go to View URL
WDS.log.info("log: ${VIEW_URL_1}")
WDS.log.info("log: ${VIEW_URL_2}")
WDS.log.info('https://${URL}/outage-domain/unavailability-messages/show?ummId=${UNAVAILABILITY}')

//Go to View URL
WDS.browser.get("https://${URL}/outage-domain/unavailability-messages/show?ummId=${UNAVAILABILITY}")

//Pause to load
var pauseEnd = java.lang.System.currentTimeMillis() + delay
while (java.lang.System.currentTimeMillis() < pauseEnd ) {}

//Close Dialog about first time on Emfip site
WDS.browser.findElement(selenium.By.xpath("//div[@role='button']")).click()

//Display View Detail if there is collapsed detail
WDS.browser.findElement(selenium.By.xpath("//a[@title='Detail' and contains(@class, 'operation-detail-expand')]")).click()

//Take screenshot
selenium.FileUtils.copyFile(WDS.browser.getScreenshotAs(selenium.OutputType.FILE),
new java.io.File('${RESULTS_PATH_OK}${TEST_NAME_SEL}${TEST_NAME}.png'))

//END-------------------------------------------------------------------------------------------------------
```

Obrázek 39 - Selenium kontrola. *Zdroj: Autor*

Tímto krokem končí konkrétní testovací scénář na vytvoření nedostupnosti přenosové sítě. Celková časová náročnost se v tomto případě odhaduje na 5 až 8 hodin práce. Výhodou takto náročného testovacího scénáře je, že jednotlivé dynamicky vytvořené

skripty se dají použít i u ostatních testovacích scénářů. Tím lze ušetřit drahocenný čas, který lze využít například pro jiné testovací činnosti. Nicméně podle dosavadních zkušeností, bylo zjištěno, že na jeden automatizovaný testovací scénář pro aplikaci NUCS v průměru odpovídá přibližně 5 hodin práce.

Kompletní regresní testy se skládají z těchto jednotlivých částí a pokrývají přes 18 testovacích scénářů pro aplikaci NUCS. Odpracovaná doba na regresních testech se pohybuje kolem týdnu a půl až dvou týdnů práce (60 až 80 hodin). Po spuštění celého balíčku testů je doba automatizovaného testování přibližně 30 až 45 minut, což je stejný čas, který dokáže využít tester pro jeden testovací scénář.

4.3 Propojení automatizovaného testu s nástroji pro management aplikace

Důležitou vlastností a výhodou nástroje Apache JMeter je možnost propojení tohoto softwaru s nástroji, které usnadňují manipulaci s automatizovanými testy a řízení testovací činnosti na daném projektu.

Propojení s nástrojem TeamCity je poměrně snadné a efektivní. TeamCity dokáže spustit vlastního agenta pro určitý operační systém. Tento agent umožní stažení JMeteru z úložiště Git a následně jeho spuštění. To vše pomocí základní konfigurace. Pro správné spuštění daného automatizovaného nástroje, stačí pouze vyplnit příkazový skript. Ten lze vidět na obrázku 40 a značí:

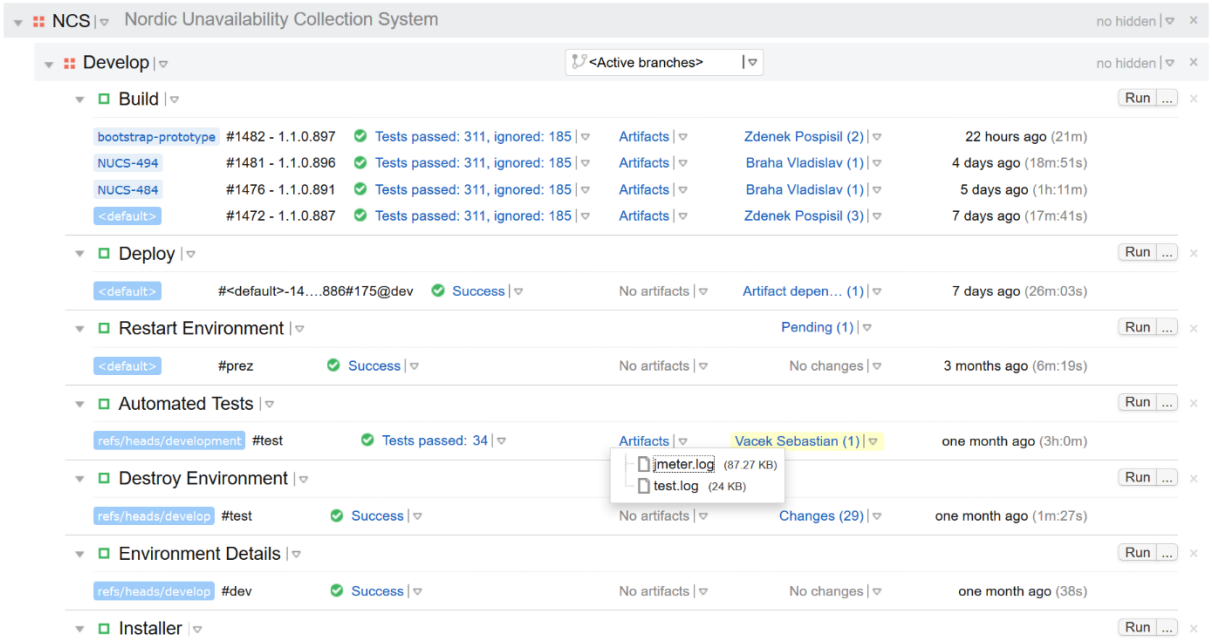
- Jaký nástroj se spouští. Tím je soubor pro operační systém Linux ./jmeter.sh, jelikož v současné době jsou všichni agenti nastaveny na tento systém.
- S jakými parametry se spouští. Prvním parametrem je odkaz na vytvořený regresní test, který se má spouštět, ten má příponu jmx. Další dva parametry odkazují na dynamicky se doplňující adresy systému. Pokud jsou v TeamCity vyplněny hodnoty db.ip a wp.ip, jsou propsány až do JMeteru. To usnadňuje manipulaci s testem, kdy není potřeba lokálně otevírat daný test a měnit ručně jeho parametry.

- Kam se ukládá výstup. Všechny výstupy z regresního testu se ukládají do test.log. Tento log lze po dokončení testu stáhnout nebo zobrazit v TeamCity, jako jeden z možných výstupů. To umožňuje rychlou kontrolu vykonaného automatizovaného testu.

Obrázek 40 - TeamCity, skriptovací příkaz. *Zdroj: Autor s využitím výstupu informačního systému.*

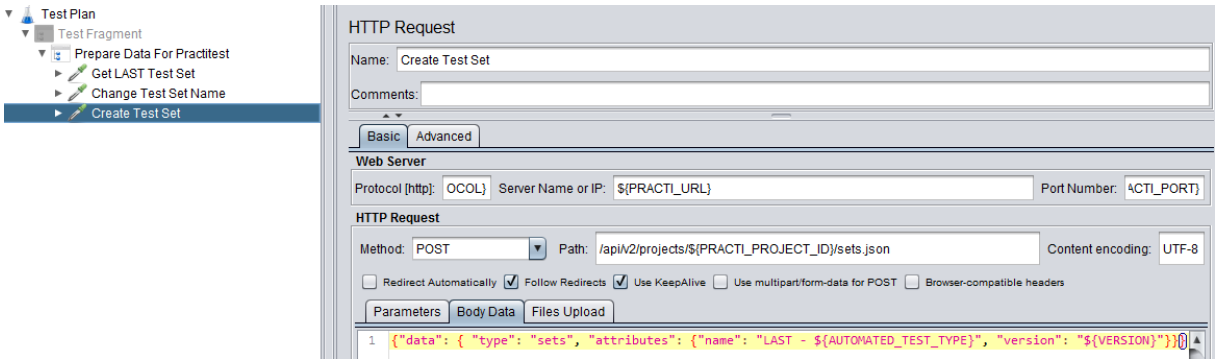
Další výhodou je možnost opakovaného spouštění jednotlivých automatizovaných testů. V TeamCity může automatizované testy spouštět kdokoliv, kdo má na danou činnost právo. Pokud uživatel má veškerá oprávnění, jsou mu umožněny všechny činnosti, které jsou zobrazeny na obrázku 41. Pokud se zaměří na sekci Automated Tests, která slouží ke spouštění testů pro aplikaci NUCS, pak uživateli stačí pouze kliknout na tlačítko Run. Po kliknutí na toto tlačítko se zobrazí formulář s parametry, které je nutné vyplnit pro správný běh automatizovaného testu. Po vyplnění opět postačuje kliknutí na tlačítko Run a daný test je spuštěn.

Test běží na agentovi, tím pádem není nutné ztrácet čas sledováním testu a lze pokračovat v jiných činnostech. Celé testování je pro uživatele skryté (tzv. black box) a jediné co je potřeba, jsou výsledky z daného běhu. Ty se zobrazí po skončení testu v TeamCity, zároveň jsou přehledněji propsány i do nástroje PractiTest.



Obrázek 41 - Automatizovaný test v TeamCity. *Zdroj: Autor s využitím výstupu informačního systému.*

Propojení s nástrojem PractiTest už tak snadné není. Základní podmínkou pro připojení je nastavení URL adresy PractiTestu a přihlašovacích údajů v JMeteru. To se provádí pomocí HTTP Authorization Manager. Po správném nastavení těchto údajů bylo nutné vytvořit moduly, které slouží ke komunikaci s tímto externím softwarem. První modul slouží k vytvoření Test Setu v PractiTestu. Pro tento postup byl opět využit request s metodou post. Jak lze doložit na obrázku 42, dochází k vytvoření nového Test Setu za pomoci JSONu, kde je jasně specifikován název a verze daného setu.

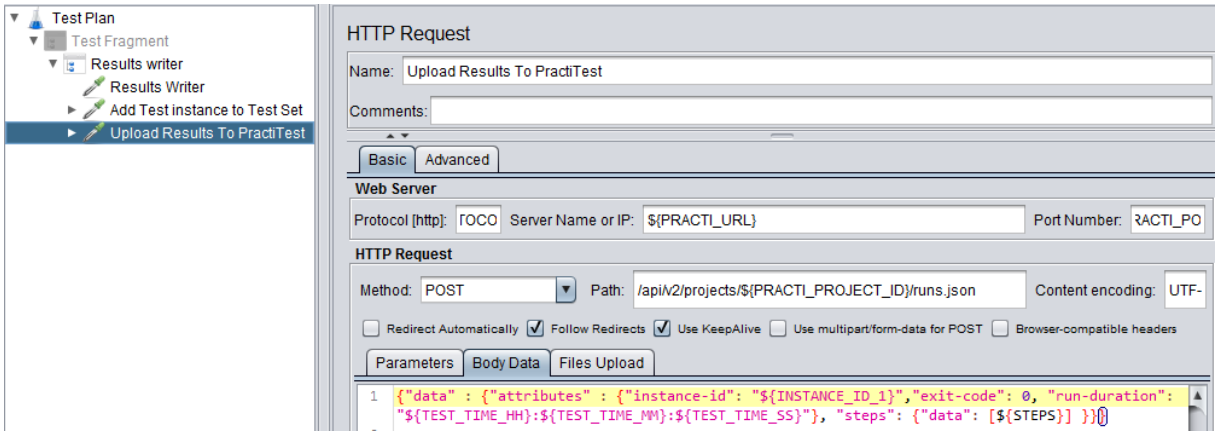


Obrázek 42 - Create Test Set. *Zdroj: Autor*

Po vytvoření Test Setu je důležité správně tuto složku naplnit. K tomuto slouží další modul, který se stará o přidávání dílčích testovacích instancí a následné nahrání

výsledků přímo do PractiTestu. Nejprve je nutné, aby veškeré testovací scénáře byly vytvořeny v PractiTestu manuálně. Po vytvoření totiž získávají identifikátor, který je vkládán jako parametr u příslušného testovacího scénáře v JMeteru.

Dále přichází na řadu vyplnění testovacích výsledků dané instance. To je provedeno rovněž pomocí requestu. Jak lze vidět na obrázku 43, podle dosažených předchozích výsledků z testovacího skriptu v JMeteru, se nahrává stejný výsledek k dané instanci v PraciTestu. Zároveň jsou zde vyplněny jednotlivé kroky, které instance obsahuje. Také se zde nachází časový údaj o délce trvání daného testu.



Obrázek 43 - Nahrání výsledků. *Zdroj: Autor*

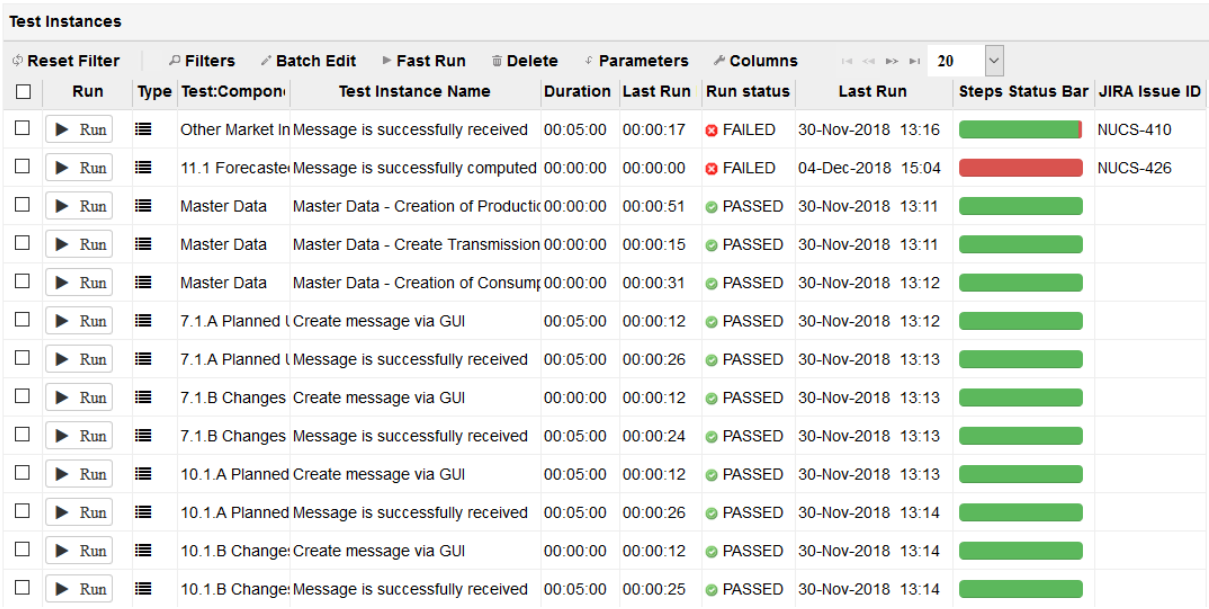
Toto propojení je převzato z jiných již vytvořených projektů ve firmě Unicorn Systems, jelikož je kvalitně a dynamicky zpracováno. To umožňuje opakované využití v různých projektech, které jsou založené na automatizovaném testování pomocí nástroje Apache JMeter. Stačí pouze přizpůsobit jednotlivé proměnné, pak je propojení zcela hotové. Jednotlivé výsledky jsou zapisovány zcela automaticky přímo do PractiTestu, jak je možné vidět v kapitole 4.4 Vyhodnocení testování.

4.4 Vyhodnocení testování

Vyhodnocování automatizovaných testů lze rozdělit na dvě skupiny:

- vyhodnocení testů spuštěných na lokálním zařízení,
- vyhodnocení testů spuštěných v nástroji TeamCity.

Pokud uživatel spouští automatizované testy na lokálním zařízení, všechny výsledky vidí v reálném čase přímo v nástroji Apache JMeter, podobně jako je zobrazeno na obrázku 20. Zároveň se veškeré pořízené snímky obrazovky v rámci testu ukládají do složky „result“ spolu s veškerými logy z nástroje. Tuto složku je možné nalézt u složek „bin“ a „lib“ nástroje Apache JMeter. Pokud existuje spojení s nástrojem PractiTest, jsou rovněž všechny výsledky zaznamenávány do tohoto systému. Popisovanou situaci znázorňuje obrázek 44, na kterém je možné vidět úspěšně ukončené instance testovacích scénářů, ale zároveň i dva neúspěšně ukončené. Ty jsou posléze manuálně revidovány. Pokud se skutečně jedná o chybu systému, tester vytvoří v nástroji Jira defekt, který je následně odkázán i v PractiTestu přes vyplnění políčka JIRA Issue ID.



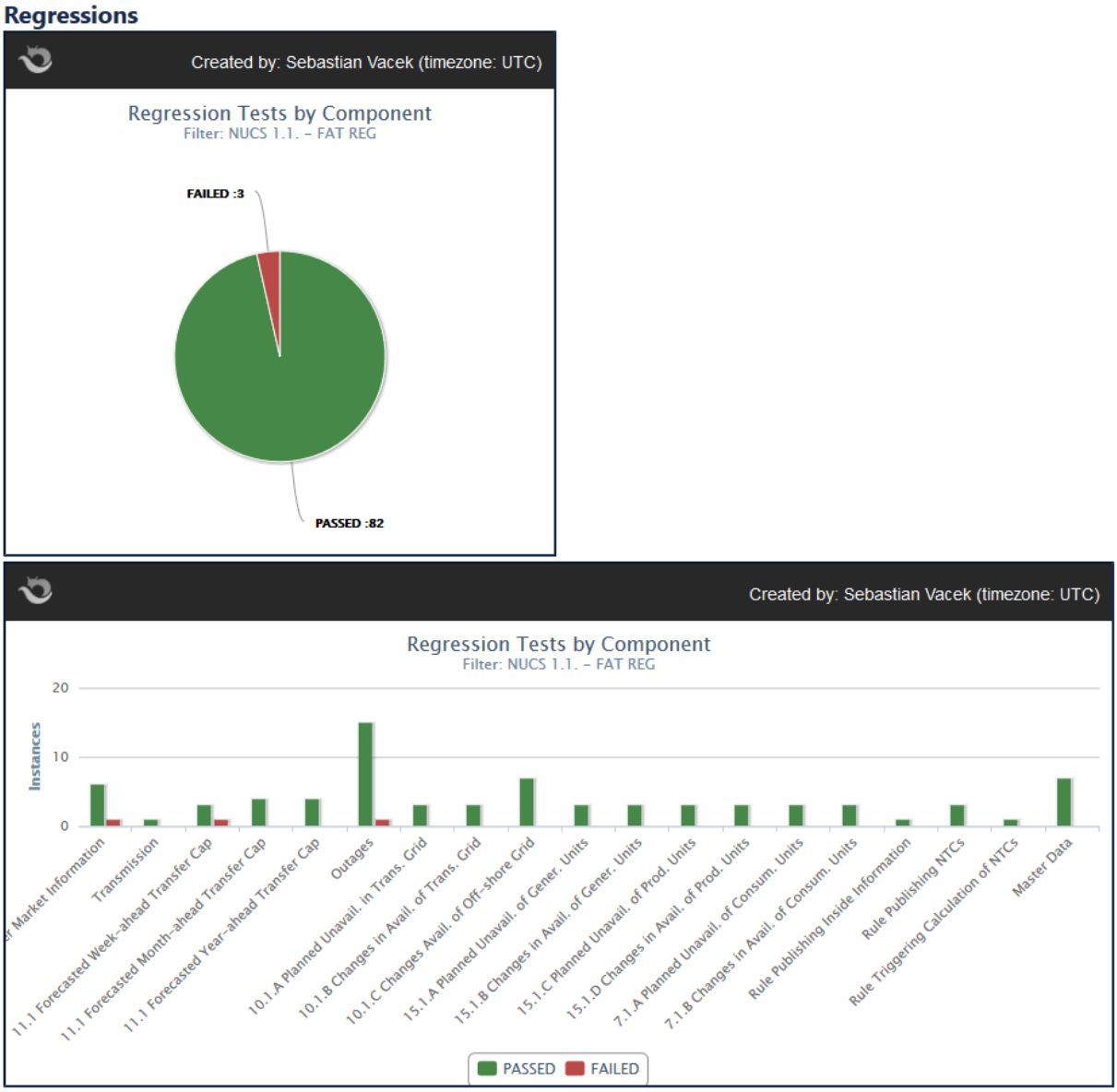
Test Instances

Reset Filter | Filters | Batch Edit | Fast Run | Delete | Parameters | Columns | 20

Run	Type	Test:Compon	Test Instance Name	Duration	Last Run	Run status	Last Run	Steps Status Bar	JIRA Issue ID
Run		Other Market In	Message is successfully received	00:05:00	00:00:17	FAILED	30-Nov-2018 13:16		NUCS-410
Run		11.1 Forecaste	Message is successfully computed	00:00:00	00:00:00	FAILED	04-Dec-2018 15:04		NUCS-426
Run		Master Data	Master Data - Creation of Productic	00:00:00	00:00:51	PASSED	30-Nov-2018 13:11		
Run		Master Data	Master Data - Create Transmission	00:00:00	00:00:15	PASSED	30-Nov-2018 13:11		
Run		Master Data	Master Data - Creation of Consump	00:00:00	00:00:31	PASSED	30-Nov-2018 13:12		
Run		7.1.A Planned I	Create message via GUI	00:05:00	00:00:12	PASSED	30-Nov-2018 13:12		
Run		7.1.A Planned I	Message is successfully received	00:05:00	00:00:26	PASSED	30-Nov-2018 13:13		
Run		7.1.B Changes	Create message via GUI	00:00:00	00:00:12	PASSED	30-Nov-2018 13:13		
Run		7.1.B Changes	Message is successfully received	00:05:00	00:00:24	PASSED	30-Nov-2018 13:13		
Run		10.1.A Planned	Create message via GUI	00:05:00	00:00:12	PASSED	30-Nov-2018 13:13		
Run		10.1.A Planned	Message is successfully received	00:05:00	00:00:26	PASSED	30-Nov-2018 13:14		
Run		10.1.B Change	Create message via GUI	00:00:00	00:00:12	PASSED	30-Nov-2018 13:14		
Run		10.1.B Change	Message is successfully received	00:05:00	00:00:25	PASSED	30-Nov-2018 13:14		

Obrázek 44 - Výsledky v nástroji PractiTest. *Zdroj: Autor s využitím výstupu informačního systému.*

Veškeré záznamy z jednotlivých testovacích kol jsou přetvářeny do přehledných grafů, které charakterizují stav vyvíjené aplikace. Tyto grafy jsou mapovány do nástrojů Confluence, Jira a podobně, které pomáhají řídit vývoj aplikace NUCS. Na obrázku 45 je zřejmé, že testovací kolo „NUCS 1.1. - FAT REG“ obsahuje 3 testovací scénáře, ve kterých byl nalezen defekt. To umožňuje managementu mít alespoň určitou představu o kvalitě dodávaného softwaru. Zároveň jim to umožňuje včasně reagovat na aktuální situaci a případně lépe řídit jednotlivé činnosti svého týmu.



Obrázek 45 - Grafy v Confluence. *Zdroj: Autor s využitím výstupu informačního systému.*

5 SHRNUTÍ ZÍSKANÝCH VÝSLEDKŮ

Hlavním cílem této diplomové práce byla analýza možností tvorby automatizovaného testu SW aplikace s využitím nástroje Apache JMeter a porovnání časové náročnosti manuálního a automatizovaného testování.

Nejprve bylo nutné teoreticky vymezit základní pojmy a činnosti spjaté s testováním. Tato oblast byla zpracována s ohledem na dosavadní zkušenosti na projektech ENTSO-E Transparency Platform a Nordic Unavailability Collection System ve firmě Unicorn Systems. Posléze byly zpracovány hlavní rozdíly mezi manuálním a automatizovaným testováním, které přinášejí detailní informace o výhodách a nevýhodách jednotlivých typů testování.

Dále byl proveden rozbor nástroje Apache JMeter se zaměřením na jednoduchost tvorby automatizovaných testů pomocí tohoto nástroje. Zde byl demonstrován příklad tvorby testu, který ověřoval správnost webových stránek Univerzity Pardubice. Jako nadstavba tohoto nástroje byla popisována možnost propojenosti mezi nástrojem Apache JMeter a nástroji, které usnadňují řízení softwaru.

Porovnání časové náročnosti manuálního a automatizovaného testování bylo provedeno na jednom konkrétním testovacím případu aplikace NUCS. Tento případ obsahoval kroky, které vedly k vytvoření plánované nedostupnosti přenosové sítě a pokrývaly tím základní funkčnosti vyvíjené aplikace. Zároveň zde byl proveden rozbor časové náročnosti nad kompletním balíčkem testovacích případů aplikace NUCS. Během manuálního testování bylo zjištěno, že doba strávená nad testováním jednoho testovacího případu byla okolo 30 až 50 minut, přičemž manuální testování 18 testovacích případů aplikace zabralo přibližně 12 až 16 hodin. Na rozdíl od toho, vytvoření jednoho automatizovaného testovacího případu zabralo odhadem 5 až 8 hodin práce. Vytvoření kompletního balíčku základních 18 automatizovaných testovacích případů zahrnovalo až dvoutýdenní pracovní činnost. Nicméně po spuštění celého balíčku testů je doba automatizovaného testování přibližně 30 až 45

minut, což je stejný čas, který dokáže využít tester pro jeden testovací scénář. Kompletní porovnání výsledků manuálního a automatizovaného testování je možné nalézt v tabulce 4.

Tabulka 4 – Přehled časové náročnosti. *Zdroj: Autor*

	Manuální testování	**Automatizované testování**
Čas realizace jednoho testovacího případu.	30–50 minut.	5–8 hodin.
Čas realizace 18 testovacích případů.	12–16 hodin.	60–80 hodin.
Čas opakované realizace testovacího případu.	30–50 minut.	1–5 minut.
Čas opakované realizace 18 testovacích případů.	12–16 hodin.	30–45 minut.

Ze získaných údajů je zřejmé, že na vytvoření základního balíčku 18 automatizovaných testů je zapotřebí daleko více času než na manuální testování stejného balíčku testovacích případů. Avšak po vytvoření automatizovaných testů se doba opakovaného testování snížila z původních 12 až 16 hodin na 30 až 45 minut. Při agilním vývoji, který předpokládá nasazování nových verzí i několikrát za den, jsou tyto testy nepřetržitě využívány. Tímto způsobem se počáteční investice na tvorbu automatizovaných testů několikanásobně vrací v průběhu celého vývoje softwaru. Nicméně pokud by se předpokládalo, že počet testovacích kol má být malého počtu (do 6 kol), tvorba automatizovaných testů by se s největší pravděpodobností nevyplatila, proto by bylo upřednostněno opakované manuální testování vyvíjené aplikace.

Jelikož existuje možnost propojit nástroj Apache JMeter s jinými nástroji, které usnadňují řízení testovacího procesu a zároveň i samotného vývoje aplikace, je tím zajištěna ještě větší efektivita automatizovaných testů. Propojení s jednotlivými

nástroji šetří čas i finance, které by byly potřeba na doplnění všech informací do těchto nástrojů.

Získané znalosti a následná doporučení

Během vývoje automatizovaných testů byla získána znalost o dynamicky se naplňujících parametrech, a to i z externích nástrojů jako je TeamCity, PractiTest a jiné. To umožňovalo efektivně používat určité části automatizovaných skriptů opakovaně, a tím ušetřit čas na tvorbu kompletního balíčku testovacích případů.

V současné době se parametry pro vytváření nedostupností přenosové sítě získávají ze stromové struktury v nástroji Apache JMeter. Je doporučeno, aby jednotlivé parametry, které se nastavují vždy po spuštění daného skriptu, bylo možné nastavit mimo nástroj Apache JMeter. Toho se dá docílit například pomocí externího excel souboru, který by obsahoval konkrétní hodnoty pro dynamické parametry. Tento soubor by mohl upravovat kdokoliv před spuštěním automatizovaného testu, a to bez znalosti nástroje Apache JMeter. Následně by se vyexportoval do csv souboru, ze kterého by Apache JMeter získal a přiřadil hodnoty k jednotlivým parametrům už zcela samostatně

Další variantou je možnost vkládat jednotlivé parametry během spouštění automatizovaného testu v nástroji TeamCity. Jelikož je těchto parametrů velké množství, vyplňovaný formulář by byl velice obsáhlý a nepřehledný, z tohoto důvodu je upřednostňována první varianta.

Rovněž byla získána znalost o regulárních výrazech, které se využívají během dynamického doplňování informací z výsledků automatizovaných testů. Tyto výrazy umožňují efektivnější a rychlejší vývoj těchto testů.

ZÁVĚR

Výsledkem této práce je ucelený automatizovaný test v nástroji Apache JMeter, který je pravidelně využíván během vývoje aplikace NUCS. Tento test simuluje činnost testera a pokrývá základní funkčnosti vyvíjené aplikace. Obrovskou výhodou je poměrně rychlý průběh automatizovaného testu spjatý s autonomním chováním, což zaručuje podstatnou úsporu času pracovníka a rovněž i úsporu financí firmy Unicorn Systems. Předpokládá se, že tento test bude i nadále využíván během servisních činností u aplikace NUCS.

Systém NUCS se nadále rozvijí a v průběhu několik měsíců bude spuštěn do reálného provozu. Spolu s rozvojem systému se rozšiřují a udržují i automatizované testy. Cílem společnosti je tak nadále zvyšovat procento pokrytí aplikace automatizovanými testy, a tím i zvyšovat kvalitu dodávaného softwaru.

Automatizované testování má v dnešní době nepostradatelný význam. Zejména s čím dál více využívaným agilním přístupem během vývoje softwaru a větší složitostí softwaru je potřeba, aby byl takovýto software kvalitně, správně, a hlavně rychle testován. Pomocí automatizovaného testování je tato potřeba zajištěna, a to s minimálními náklady. Počáteční investice do tvorby automatizovaného testu se v průběhu vývoje několikanásobně vrátí. Je pravděpodobné, že v nadcházejících letech bude na automatizované testování kladen ještě větší důraz, než je tomu dnes. To by mohlo přinést tvorbu nových technologií, například s vyspělou umělou inteligencí nebo větší počet nástrojů pro tento typ testování.

POUŽITÁ LITERATURA

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

[2] PATTON, Ron a David KRÁSENSKÝ. Testování softwaru. Praha: Computer Press, 2002. ISBN 80-7226-636-5.

[3] VACEK, Sebastian. *Automatizované testování SW a aplikace vybraných nástrojů v praxi*. Hradec Králové, 2017. Bakalářská práce. Univerzita Hradec Králové. Vedoucí práce doc. Mgr. Tomáš Kozel, Ph.D.

[4] DARUKA, Chandan. What is a Test Architect in Software Testing?. *Quora* [online]. [cit. 2018-10-27]. Dostupné z: https://www.quora.com/What-is-a-Test-Architect-in-Software-Testing

[5] HOLLAND, Bill. How to Define a Software Tester. *Sitepoint* [online]. 2012 [cit. 2018-10-27]. Dostupné z: https://www.sitepoint.com/how-to-define-a-software-tester/

[6] ERTL, Otakar. *Testovací praktiky* [online]. In: . Praha: Profinit EU, 2018, 17. 01. 2018, s. 62 [cit. 2018-11-07]. Dostupné z: https://profinit.eu/temata-a-cviceni/07_Testing_public.pdf

[7] ZELINKA, Bořek. *Testování softwaru* [online]. In: . Unicorn Systems, 2013, 10. 04. 2013, s. 38 [cit. 2018-10-27]. Dostupné z: http://d3s.mff.cuni.cz/teaching/commercial_workshops/previous/1213/zelinka-zajisteni_kvality_softwarovych_produktu.pdf

[8] *Testování & QA* [online]. Praha: Unicorn, 2018 [cit. 2018-11-21]. Dostupné z: https://unicorn.com/cz/testing-and-qa

[9] *Testing as a Service: Přístupné, flexibilní a cenově výhodné řešení pro ověření kvality softwaru*. Praha: Unicorn Systems, 2016. Dostupné také z: https://unicorn.com/cz/testing-and-qa

[10] HLAVA, Tomáš. ISTQB - přednáška testování. *Příprava na test ISTQB FL.* Praha : Unicorn - Top Gun, 16. 06 2015.

[11] HLAVA, Tomáš. Smoke testy. *Testování softwaru* [online]. 2011 [cit. 2019-03-29]. Dostupné z: http://testovanisoftwaru.cz/tag/smoke-testy/

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

[13] Druhy testování v procesu vývoje SW. *Testování software* [online]. 2011 [cit. 2019-03-29]. Dostupné z: http://test.swtestovani.cz/index.php?option=com_content&view=article&id=18:druhy-testovani-v-procesu-vyvoje-sw&catid=3:zaklady&Itemid=11

[14] HLAVA, Tomáš. Zátěžové testy software. *Testování softwaru* [online]. 2012 [cit. 2019-03-29]. Dostupné z: http://testovanisoftwaru.cz/testing/zatezove-testy-software/

[15] JEŽEK, Miroslav, Michal BEDNÁŘ a Ondřej ŠTVERÁK. *NUCS - Test Strategy*. 2. Praha: Unicorn Systems, 2018.

[16] What is RELEASE NOTE?. *The Law Dictionary: Featuring Black's Law Dictionary Free Online Legal Dictionary 2nd Ed.* [online]. [cit. 2019-03-30]. Dostupné z: https://thelawdictionary.org/release-note/

[17] Manuální testování. *Testování softwaru* [online]. [cit. 2019-03-30]. Dostupné z: http://testovanisoftwaru.cz/manualni-testovani/

[18] Automatizované testování. *Testování softwaru* [online]. [cit. 2019-02-02]. Dostupné z: http://testovanisoftwaru.cz/automatizovane-testovani/

[19] A quick guide to Manual Testing Vs Automated Testing. *ReQtest* [online]. ReQtest, 2018, 4. července 2018 [cit. 2019-02-08]. Dostupné z: https://reqtest.com/testing-blog/manual-testing-vs-automated-testing/

[20] Automation Testing Vs. Manual Testing: What's the Difference?. *Guru99* [online]. [cit. 2019-02-07]. Dostupné z: https://www.guru99.com/difference-automated-vs-manual-testing.html

[21] KAPOOR, Kush. Ad Hoc Testing Explained. *WebOmates* [online]. 2018 [cit. 2019-03-30]. Dostupné z: https://www.webomates.com/blog/ad-hoc-testing/

[22] STRHARSKÝ, Jaroslav. Základní principy testování podle ISTQB: Pesticidní paradox. *Computer World* [online]. 2013 [cit. 2019-03-30]. Dostupné z: https://computerworld.cz/ness-up-ideas/zakladni-principy-testovani-podle-istqb-pesticidni-paradox-50361

[23] Brian. Best Automation Testing Tools for 2019 (Top 10 reviews). *Medium* [online]. 2017, 26. října 2017 [cit. 2019-02-08]. Dostupné z: https://medium.com/@briananderson2209/best-automation-testing-tools-for-2018-top-10-reviews-8a4a19f664d2

[24] What is Selenium?. *SeleniumHQ* [online]. [cit. 2019-04-02]. Dostupné z: https://www.seleniumhq.org/

[25] *Katalon Studio* [online]. Atlanta: Katalon, 2018 [cit. 2019-04-02]. Dostupné z: https://www.katalon.com/

[26] Unified Functional Testing (UFT): Functional Test Automation Software. *MICRO FOCUS* [online]. Micro Focus, 2019 [cit. 2019-04-02]. Dostupné z: https://www.microfocus.com/en-us/products/unified-functional-automated-testing/overview

[27] TestComplete. *SMARTBEAR* [online]. SmartBear Software, 2019 [cit. 2019-04-02]. Dostupné z: https://smartbear.com/product/testcomplete/overview/

[28] *SoapUI* [online]. SmartBear Software, 2019 [cit. 2019-04-02]. Dostupné z: https://www.soapui.org/

[29] Rational Functional Tester. *IBM* [online]. [cit. 2019-04-02]. Dostupné z: https://www.ibm.com/us-en/marketplace/rational-functional-tester

[30] *Tricentis* [online]. Tricentis, 2019 [cit. 2019-04-02]. Dostupné z: https://www.tricentis.com/products/

[31] Test Automation for All. *Ranorex* [online]. Ranorex, 2019 [cit. 2019-04-02]. Dostupné z: https://www.ranorex.com/

[32] *Postman* [online]. Postman, 2019 [cit. 2019-04-02]. Dostupné z: https://www.getpostman.com/

[33] The Apache Software Foundation. *The Apache Software Foundation* [online]. [cit. 2019-02-09]. Dostupné z: https://www.apache.org/foundation/

[34] Apache JMeter™. *The Apache Software Foundation* [online]. 2018 [cit. 2019-02-09]. Dostupné z: https://jmeter.apache.org/

[35] *ATLASSIAN: Jira Software* [online]. ATLASSIAN, 2019 [cit. 2019-04-07]. Dostupné z: https://cs.atlassian.com/software/jira

[36] Jira Software: Jira Product Guides & Tutorials. *Atlassian* [online]. Atlassian, 2019 [cit. 2019-02-23]. Dostupné z: https://www.atlassian.com/software/jira/guides/getting-started/overview

[37] Jira Software. *Onlio* [online]. [cit. 2019-04-29]. Dostupné z: https://www.onlio.com/-/atlassian/produkty-a-reseni/jira-software

[38] Confluence. *Atlassian* [online]. Atlassian, 2019 [cit. 2019-02-25]. Dostupné z: https://cs.atlassian.com/software/confluence

[39] TeamCity. *JetBrains* [online]. JetBrains, 2019 [cit. 2019-02-25]. Dostupné z: https://www.jetbrains.com/teamcity/

[40] Features. *JetBrains* [online]. JetBrains, 2019 [cit. 2019-02-25]. Dostupné z: https://www.jetbrains.com/teamcity/features/

[41] QA & Test Case Management for Professional Testing. *PractiTest* [online]. PractiTest, 2019 [cit. 2019-02-25]. Dostupné z: https://www.practitest.com/product/

[42] Discover The Best Saas Testing Tool To Ensure Seamless Integration. *PractiTest* [online]. PractiTest, 2019 [cit. 2019-02-25]. Dostupné z: https://www.practitest.com/integrations/

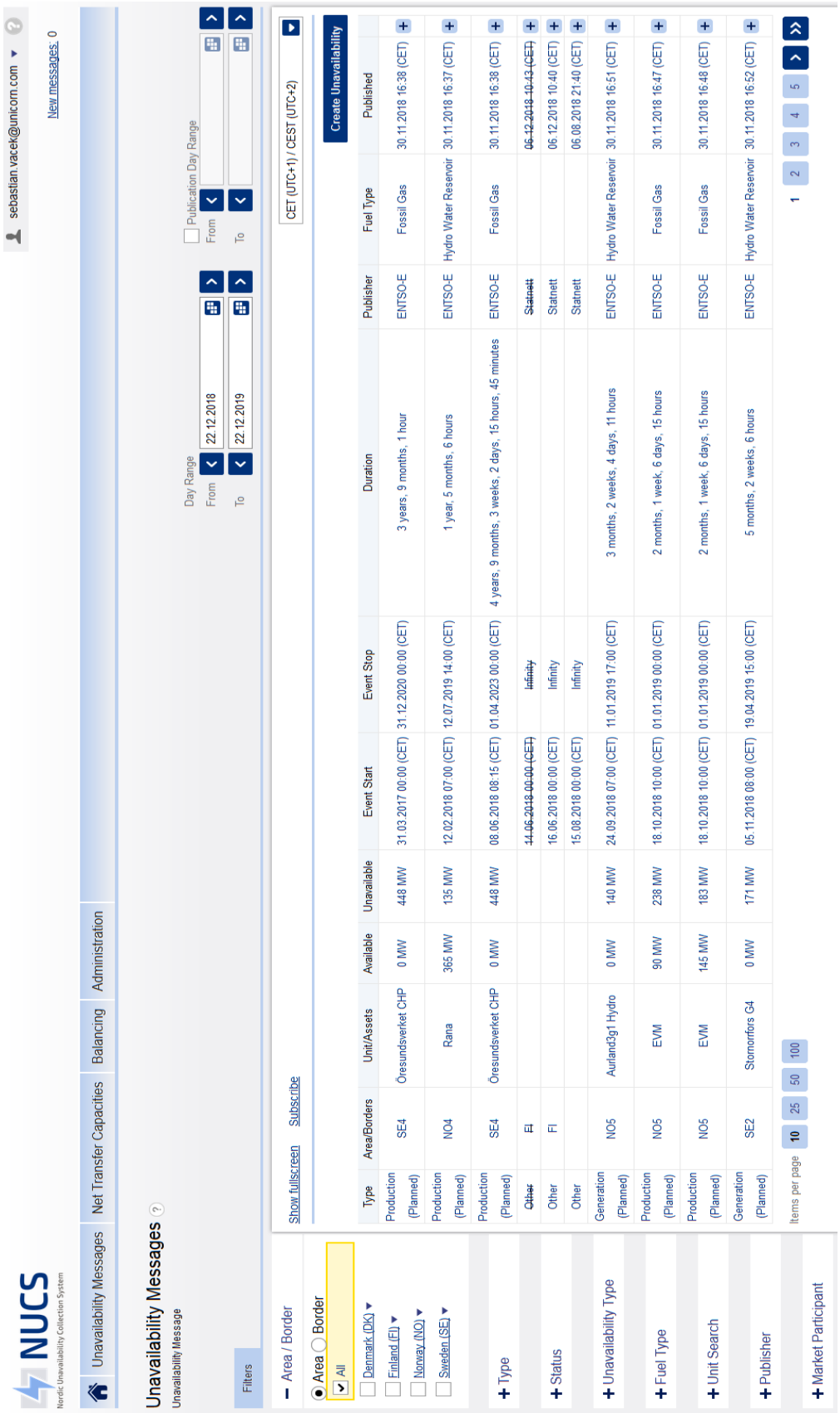
[43] A free Git client for Windows and Mac. *Sourcetree* [online]. Atlassian, 2019 [cit. 2019-03-29]. Dostupné z: https://www.sourcetreeapp.com/

[44] HATKA, Martin a Jiří DUDEK. *NUCS - High Level System Design*. 3. Praha: Unicorn Systems, 2017.

PŘÍLOHY

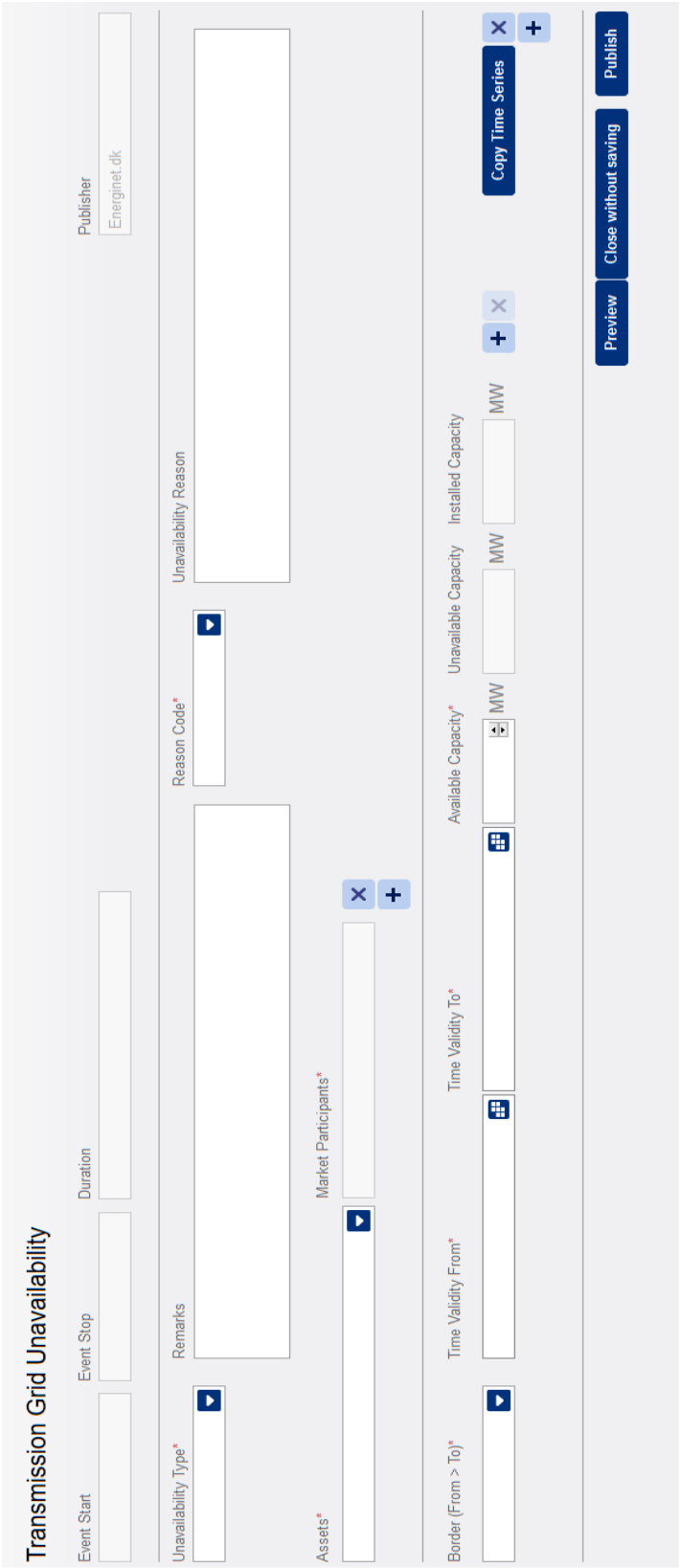
PŘÍLOHA A – APLIKACE NUCS

Tato příloha obsahuje obrázek 27 ve větší velikosti, pro kvalitnější náhled.



PŘÍLOHA B – ŠABLONA PRO VYTOVŘENÍ NEDOSTUPNOSTI

Tato příloha obsahuje obrázek 30 ve větší velikosti, pro kvalitnější náhled.



PŘÍLOHA C – PUBLIKOVANÁ DATA

Tato příloha obsahuje obrázek 31 ve větší velikosti, pro kvalitnější náhled.