Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informatiky a kvantitativních metod

Automatizované testování SW a aplikace vybraných nástrojů v praxi

Bakalářská práce

Autor: Sebastian Vacek
Studijní obor: Aplikovaná informatika

Vedoucí práce: doc. Mgr. Tomáš Kozel, Ph.D.

Odborný konzultant: Ing. Karel Schejbal
Unicorn Systems a.s.

Hradec Králové Duben 2017

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně a s použitím uvedené literatury.

V Hradci Králové dne 20.4.2017 Sebastian Vacek

Poděkování:

Děkuji vedoucímu bakalářské práce doc. Mgr. Tomáši Kozlovi, Ph.D. za metodické vedení práce. Dále děkuji svému vedoucímu, ve společnosti Unicorn Systems a.s. Ing. Karlu Schejbalovi, za vstřícnost a ochotu při praktickém vedení bakalářské práce.

Anotace

Tato bakalářská práce je zaměřena na testování, jako nedílnou součást životního cyklu vyvíjeného softwaru. Popisuje celý životní cyklus softwaru a základní pojmy, vyskytující se dále u testování jednotlivých skupin, které se objevují v cyklu. Bakalářská práce přináší základní rozdělení testů a jednotlivé způsoby testování, se kterými se lze setkat v praxi při testování vyvíjeného softwaru.

Hlavním cílem je poukázat na možnost vytvořit automatizovaný test, jako jednu z variant při opakovaném testování vyvíjeného softwaru a následný popis základního nastavení nástroje, který slouží k tvorbě automatizovaného testu.

Poslední částí je návod, jak vytvořit určitý automatizovaný test a následný praktický příklad, který ukazuje jeho využití v praxi. Zde se také nachází kritická úvaha nad pravidelným využitím automatizovaných testů, jako jednou z možností testování softwaru.

Annotation

Title: Automated Software Testing and Implementation of selected Tools in Practice

The bachelor thesis deals with testing, as an integral part in life cycle of exerted software. There is described a whole life cycle of software and a basic concepts which appeared in the testing of each group in the cycle. This bachelor refers to basic split of tests and a different ways of testing.

The main goal is to show a way how to create an automated test as a one of option on retesting of exerted software. It is description of basic settings tool which is used in creation of automated test.

In the last part of bachelor thesis is description how to create a certain automated test and a practical example which is showing their using in the practise.

Obsah

1 Úvod ... 1

1.1 Cíl práce ... 1

2 Životní cyklus softwarového projektu ... 2

2.1 Integrace ... 2

2.2 Vývoj ... 3

2.3 Provoz ... 4

2.4 Podpora ... 5

3 Testování softwaru ... 7

3.1 Základní pojmy ... 7

3.2 Dělení testů ... 8

3.2.1 Unit testy ... 9

3.2.2 Assembly testy ... 9

3.2.3 Systémové a integrační testy ... 10

3.2.4 Akceptační testy ... 11

3.2.5 Testy po akceptaci ... 12

3.2.6 Regresní testy ... 12

3.3 Způsoby testování ... 13

3.3.1 White box testování ... 13

3.3.2 Black box testování ... 14

3.3.3 Grey box testování ... 15

4 Automatizované testování ... 17

4.1 Fáze vývoje automatizovaného testu ... 17

4.1.1 Seznámení se softwarem ... 18

4.1.2 Vytvoření hrubého automatizovaného testu ... 18

4.1.3 Doladění detailů v testu ... 18

4.1.4 Integrace do celého projektu 20
5 Nástroje pro testování 21
5.1 Firebug 21
5.2 Visual Studio Test Professional 21
5.2.1 Microsoft Test Manager 22
5.2.2 Release Management 22
5.3 TestComplete 23
5.4 Apache JMeter 25
5.5 Selenium 26
5.6 Nástroje pro správu chyb 28
5.6.1 Mantis 28
5.6.2 Jira 28
5.6.3 TestLink 29
6 Vlastní příklad automatizovaného testu 30
6.1 Základní používání nástroje JMeter 30
6.1.1 Instalace nástroje 30
6.1.2 Základní přehled GUI nástroje 30
6.1.3 Základní nastavení 32
6.1.4 Nastavení jednotlivých prvků 33
6.1.5 Nahrání testovacího skriptu 35
6.1.6 Kontrola správnosti skriptu 37
6.2 Využití v praxi 38
6.2.1 Seznámení se systémem 38
6.2.2 Příprava testu 39
6.2.3 Automatizovaný test pro PCR 40
6.2.4 Doplňující informace 47

7 Výsledky a závěr 48

8 Citovaná literatura 49

9 Přílohy 53

Seznam obrázků

Obrázek 1 - Schéma životního cyklu projektu, který se využívá ve firmě Unicorn a.s. - vychází z vodopádového modelu v (1) ... 6
Obrázek 2 - V model (8) ... 8
Obrázek 3 - W model (8) ... 9
Obrázek 4 - Postup testů podle (6) ... 13
Obrázek 5 - Desktop module (23) ... 23
Obrázek 6 - Web module (25) ... 24
Obrázek 7 - Mobile module (26) ... 25
Obrázek 8 - Selenium plugin v JMeteru (Zdroj: vlastní) ... 26
Obrázek 9 - Selenium IDE (Zdroj: vlastní) ... 27
Obrázek 10 – Hlavní panel nástrojů (Zdroj: vlastní) ... 30
Obrázek 11 – Hlavní stromová struktura (Zdroj: vlastní) ... 31
Obrázek 12 – První pohled do Apache JMeter (Zdroj: vlastní) ... 32
Obrázek 13 – Rozdělení ve stromové struktuře (Zdroj: vlastní) ... 32
Obrázek 14 – HTTP Request Defaults (Zdroj: vlastní) ... 34
Obrázek 15 – HTTP(S) Test Script Recorder (Zdroj: vlastní) ... 34
Obrázek 16 – Konfigurace webového prohlížeče (Zdroj: vlastní) ... 35
Obrázek 17 - Nahraný skript bez úpravy (Zdroj: vlastní) ... 36
Obrázek 18 - Skript po úpravě (Zdroj: vlastní) ... 37
Obrázek 19 - Skript po úpravě názvu (Zdroj: vlastní) ... 37
Obrázek 20 - Zobrazené výsledky (Zdroj: vlastní) ... 38
Obrázek 21 – Stromová struktura na projektu PCR (Zdroj: vlastní) ... 39
Obrázek 22 -Rozšířená stromová struktura na projektu PCR (Zdroj: vlastní) ... 40
Obrázek 23 – Response Assertion (Zdroj: vlastní) ... 41
Obrázek 24 – BeanShell Listener (Zdroj: vlastní) ... 41
Obrázek 25 – If Controller (Zdroj: vlastní) ... 41
Obrázek 26 – Selenium ve stromové struktuře (Zdroj: vlastní) ... 42
Obrázek 27 – Kód Selenia 1 (Zdroj: vlastní) ... 42
Obrázek 28 – User Defined Variables (Zdroj: vlastní) ... 43
Obrázek 29 – Kód Selenia 2 (Zdroj: vlastní) ... 43

Obrázek 30 – Detail kódu Selenia (Zdroj: vlastní) 44
Obrázek 31 – Xpath ve stromové struktuře (Zdroj: vlastní) 44
Obrázek 32 – Xpath Extractor (Zdroj: vlastní) 44
Obrázek 33 – BeanShell Sampler (Zdroj: vlastní) 45
Obrázek 34 – Nahrání dat pomocí Selenia (Zdroj: vlastní) 46
Obrázek 35 – While Controller (Zdroj: vlastní) 46
Obrázek 36 – Podmínka ve stromové struktuře (Zdroj: vlastní) 47

1 Úvod

Úkolem této bakalářské práce bylo navrhnout automatizovaný test pro základní testovací scénář, jelikož se předpokládá dlouhodobá servisní činnost na vyvíjeném softwaru.

Dále tato práce popisuje testování, jako nedílnou součást celého životního cyklu softwaru. Následně je zde poukázáno na jednotlivé druhy testů a způsoby testování, které se v praxi využívají. V poslední části je představen automatizovaný test, jako jedna z možností, jak se testuje software. Následuje popis nástroje JMeter a tvorby automatizovaného testu.

1.1 Cíl práce

Cílem této práce je detailnější seznámení s problematikou testováním softwaru, využití automatizovaného testování a ukázka konkrétního použití automatizovaného testu v praxi.

2 Životní cyklus softwarového projektu

Životní cyklus softwarového projektu lze rozdělit do několika částí, které na sebe navazují. Jedná se o skupiny procesů, jež se mohou lišit podle strategie jednotlivých firem. V následujícím textu budou popsány jednotlivé fáze cyklu, vycházející ze základních metod a postupů firmy Unicorn Systems a.s., podle kterého se řídí jednotliví pracovníci. Tento postup je detailněji popsán v (1).

2.1 Integrace

První fází lze nazvat integrací. Jedná se o činnost mezi zákazníkem a výrobcem softwaru.

Primárním úkolem integrace je seznámení se se softwarem, který zákazník vyžaduje, a jeho začlenění v celkovém prostředí zákazníka. Postup při tvorbě softwaru začíná již prvotními rozhovory mezi klientem a výrobcem. Ty zjednodušeně nastiňují strukturu a provoz daného projektu. Po těchto diskusích má výrobce za úkol tento systém namodelovat, a přijít tak s návrhem výsledného softwaru. Výrobci v této fázi pomáhají již zmíněné informace od klienta. Tyto informace lze chápat jako soubor odpovědí na konkrétní otázky, jež výrobce klade zákazníkovi. Zároveň, klient předkládá svoje požadavky, které musí systém splňovat.

Výrobce musí získat přehled o tom, kdo tento systém bude využívat a za jakých okolností. Zda má být tento software propojován s jinými aplikacemi, či to bude samostatná jednotka, která bude zcela nezávislá. Dále je nutné zjistit, jaké vstupy (různé konfigurace, data, tabulky, dokumenty) budou do aplikace vkládány a co se s těmito vstupy dále stane.

Již v této části můžeme uvažovat o prvních testech celého dodávaného řešení. Tyto testy se skládají především z toho, zda zákazník má reálné požadavky, které je výrobce schopen naplnit. Pokud výrobce přijde na to, že zákazník podal nereálný požadavek, je mu navrhnut jiný postup na vyřešení tohoto problému. Nicméně může se stát, že ani nově takto vytvořený model systému nebude správně fungovat, jelikož se na všechny problémy nemusí přijít hned na začátku životního cyklu.

S integrací se můžeme setkat v jakékoliv části životního cyklu. Jedná se totiž o činnost, která je nedílnou součástí každého zkoumaného prvku, který má být do softwaru implementován. Integrace jako proces však nekončí specifikací a dodáním požadovaného řešení. Pokračuje přes celý životní cyklus daného softwaru, protože během případných změnových požadavků či požadovaných úprav je nutné znovu zvážit dopad do celého IT prostředí u zákazníka.

2.2 Vývoj

Druhou fází v softwarovém cyklu je vývoj dané aplikace. Jedná se především o návrh a implementaci požadovaného systému.

Návrh systému předchází samostatnému vývoji. Čím lépe je systém popsán během technického projektu, tím přesněji může být poté implementován jednotlivými vývojáři.

Úkolem vývojářů[1] je poté naprogramovat namodelovaný systém do podoby, který zákazník akceptuje dle předem domluvených parametrů. Namodelovaný systém je většinou ztvárněn v nástroji, jenž využívá objektového modelování. Pokud lze, je tento model dodáván s dokumentem, který charakterizuje funkční a nefunkční specifikace tohoto systému. To napomáhá k reálnější představě, jak tento systém naprogramovat, aby byl uživatelsky nejpřívětivější a pokrýval veškeré potřeby zákazníka.

Práce na vývoji je rozdělena mezi jednotlivé osoby. Každá osoba (vývojář) programuje svoji část aplikace. Po dokončení svého úseku je tento zdrojový kód integrován k ostatním částem. To znamená, že vývojář musí svůj kód aplikace psát tak, aby byl slučitelný s celkovým zdrojovým kódem aplikace.

Zdrojový kód aplikace musí obsahovat všechny požadavky, které jsou sepsány ve funkční specifikaci, popřípadě jsou namodelovány pomocí objektového modelování.

Už v této části se setkáváme s jedním typem testování softwaru. Tento typ testování se nazývá „Unit testy“. Jedná se o testování, které provádí sám vývojář. Po

---

[1] Osoby, které vyvíjí danou aplikaci

navržení svého kódu a implementaci do celkového systému se musí tento kód otestovat. Vývojář testuje správnou funkčnost svého navrženého kódu i to, zda je tento kód kompatibilní s celkovým kódem systému. Tím ale testování nekončí.

Po testování vývojářů přichází testování celé aplikace jako celku. Zde jsou zapojeny osoby (testeři), které mají na starost již zmíněné testování celého systému. Podle navrženého komplexního modelu se vytvoří testovací scénáře, podle nichž testeři postupují.

Základní testy na straně výrobce jsou FAT testy (Factory Acceptance Testing). Testy ze strany zákazníka se nazývají SAT testy (Site Acceptance Testing). Testy, které kontrolují, zda systém funguje správně pro uživatele, se nazývají UAT testy (User Acceptance Testing). Případně lze ještě využít SIT testy (System Integration Testing), ty se využívají ke kontrole funkčnosti vyvíjeného systému po zapojení k ostatním systémům.

Pokud aplikace nefunguje dle specifikace, je vrácena zpět k vývojáři na odstranění daného problému. Po úspěšném otestování funkčnosti zdrojového kódu je systém schopen provozu.

2.3 Provoz

Předposlední fází v životním cyklu je provoz systému. Zákazník má sice k dispozici hotovou aplikaci, ale pokud je riziko uvedení do reálného provozu příliš vysoké (kritická aplikace, potenciální finanční ztráty, výpadky výroby atp.), je nutné spustit zkušební provoz. V něm se testuje simulace reálného použití. V aplikaci se provádí naprosto stejný denní proces, ale výsledky nejsou použity, pouze se testuje jejich shoda s výstupy z reálného provozu. Software se testuje v takzvaném pilotním provozu. V této fázi mohou nastat dva případy. Buď zákazník je spokojen s navrženým systémem a nenašel žádné chyby, nebo našel závažné chyby systému. Ty by měly být odhaleny již v SAT nebo UAT testech, ale mohlo se stát, že se na tyto chyby nepřišlo. Pokud tedy zákazník našel nějaké chyby, jsou reportovány zpět výrobci aplikace a je požadováno opravení těchto chyb, jelikož se jedná o chybný nebo nestabilní provoz aplikace. Jestliže je aplikace bez závažnějších chyb a běží stabilně, může přejít do další úrovně.

Po úspěšném pilotním provozu následuje rutinní provoz. Zde je výsledný systém spuštěn a postupně se k němu připojují reální uživatelé. Jakmile do systému vstoupí uživatel, měl by být software v bezchybném stavu a měl by správně fungovat, jelikož prošel širokou škálou testování v průběhu jeho životního cyklu. Nicméně ani to nezajišťuje, že systém je zcela bez chyb.

Ke správnému a úspěšnému provozu je zapotřebí komplementární proces podpory a údržby.

2.4 Podpora

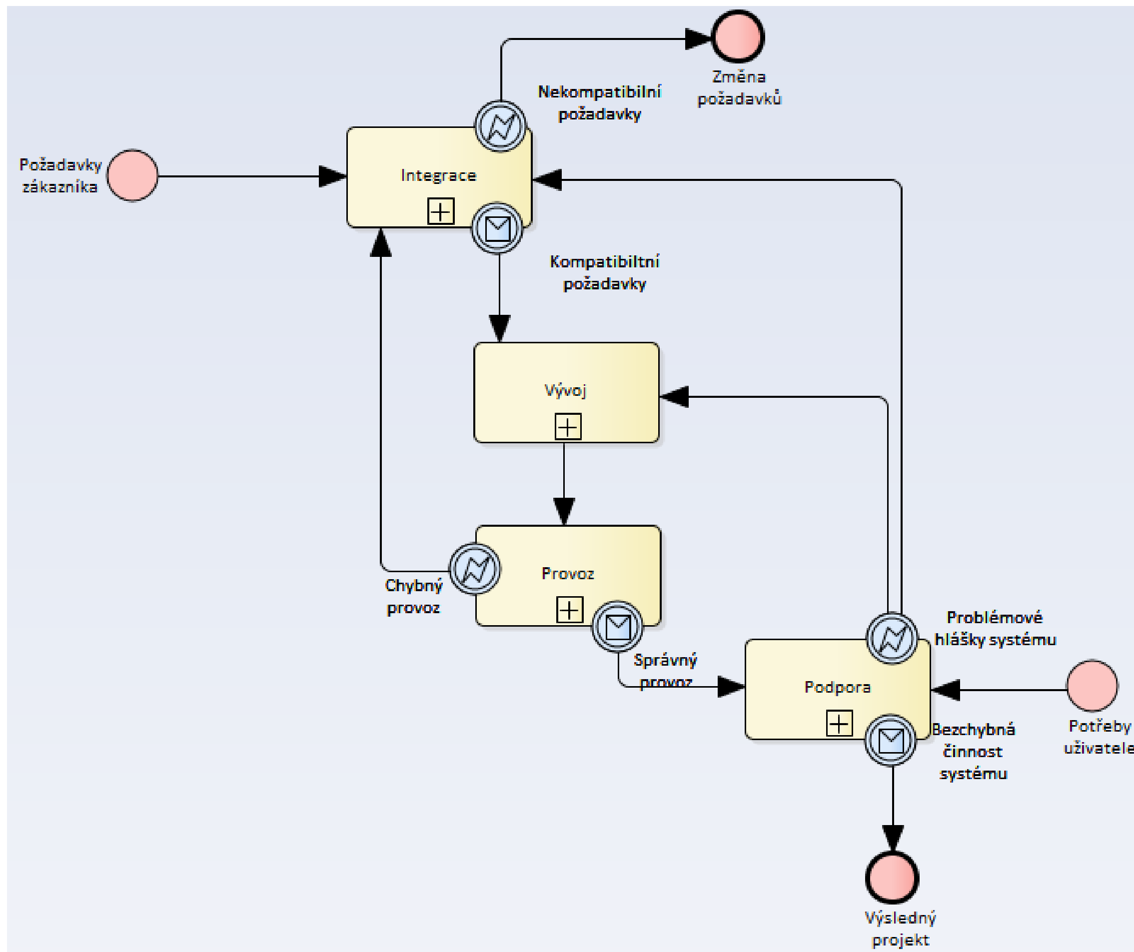
Závěrečnou fází celého životního cyklu je podpora a údržba. Ta je zaměřena na pomoc při používání aplikace. Podpora je důležitou součástí cyklu, jelikož se jedná o činnost výrobce systému, která napomáhá ke správnému chodu aplikace. Podporu využívají především uživatelé aplikace, jelikož ti aktivně používají vyvinutý systém.

Podporu můžeme rozdělit na několik variant. Jak již bylo řečeno, i výsledný systém, který začínají používat uživatelé, nemusí být zcela bezchybný. Právě tyto chyby, které nalezli uživatelé, jsou reportovány k podpoře aplikace. Ta má za úkol tyto chyby analyzovat a následně je opravit.

Další variantou jsou připomínky uživatelů. Uživatelé svou praxí v systému, umožňují zefektivnit danou aplikaci. Svoje nápady na lepší zpracovávání softwaru opět posílají na podporu, která tyto nápady konzultuje se zákazníkem a vývojáři. Pokud je požadavek schválen, je implementován a integrován do celého systému.

Díky těmto variantám se cyklus začíná opakovat a vznikají tak nové verze systému, které slouží k lepšímu a efektivnějšímu fungování celé aplikace.

K přehlednějšímu pochopení celého životního cyklu softwarového projektu může pomoci tento obrázek, který ukazuje souvislosti mezi již zmíněnými skupinami.



Obrázek 1 - Schéma životního cyklu projektu, který se využívá ve firmě Unicorn a.s. - vychází z vodopádového modelu v (1)

3 Testování softwaru

V této části bakalářské práce jsou popsány základní pojmy automatizovaného testování softwaru. Více podrobnějších informací, které popisují jednotlivé pojmy, lze nalézt v (2), (3), (4) a (5).

3.1 Základní pojmy

Vývojář – osoba, která vyvíjí (programuje) namodelovaný software.

Tester – osoba, která testuje vyvinutý software.

Test architekt – osoba, která je zodpovědná za navržení testovací strategie a řízení testovacího procesu.

Testovací nástroj – software, který slouží testerovi k otestování dané aplikace.

Testovací dokumentace – soubor dokumentů, který je potřeba pro kvalitní proces testování.

Testovací případ (Test case) – dokument, který je vytvořen při vývoji softwaru. V něm jsou sepsány jednotlivé kroky, které určují postup ke správnému otestování daného softwaru.

Testovací sada (Test Suite) – soubor testů, které spolu určitým způsobem souvisí. Testy, které jsou zařazeny do testovací sady, určuje tester, případně test architekt.

Testovací kroky (Test Step) – jednotlivé činnosti při testování softwaru, jež mají být vykonány v určitém pořadí, které určuje Test Case.

Testovací skript (Test Script) – soubor, v němž jsou uvedené jednotlivé kroky, jež mají být otestovány. Pro každý krok je sepsán výsledek, kterého by měl tester dosáhnout. Testovací skript musí obsahovat všechny vstupní požadavky pro své správné vykonání. Nedílnou součástí jsou vstupní data. Tester využívá tato data k jednotlivým krokům testovacího skriptu. Posléze zjišťuje u každého kroku, zda proběhl dle očekávání.

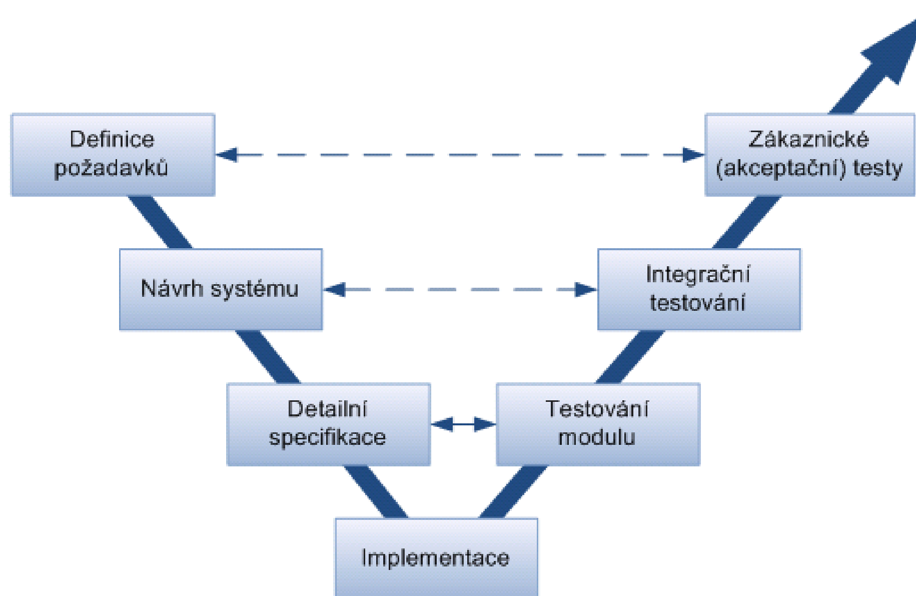
Testovací scénář (Test Scenario) – scénář, který je vytvořen spojením několika testovacích skriptů. Jednotlivé části odpovídají reálnému používání softwaru ze strany zákazníka. Jednotlivé testovací skripty na sebe navazují tak, že konec prvního skriptu tvoří začátek skriptu následujícího.

Testovací data – soubory dat, které simulují reálné použití aplikace. Jsou potřebná k úspěšnému nebo neúspěšnému dokončení testovacího scénáře.
Regulární výraz (Zkráceně také regexp nebo regex) – speciální řetězec znaků, který představuje určitou šablonu pro textové řetězce.
Softwarová chyba – chyba testovaného softwaru.

3.2 Dělení testů

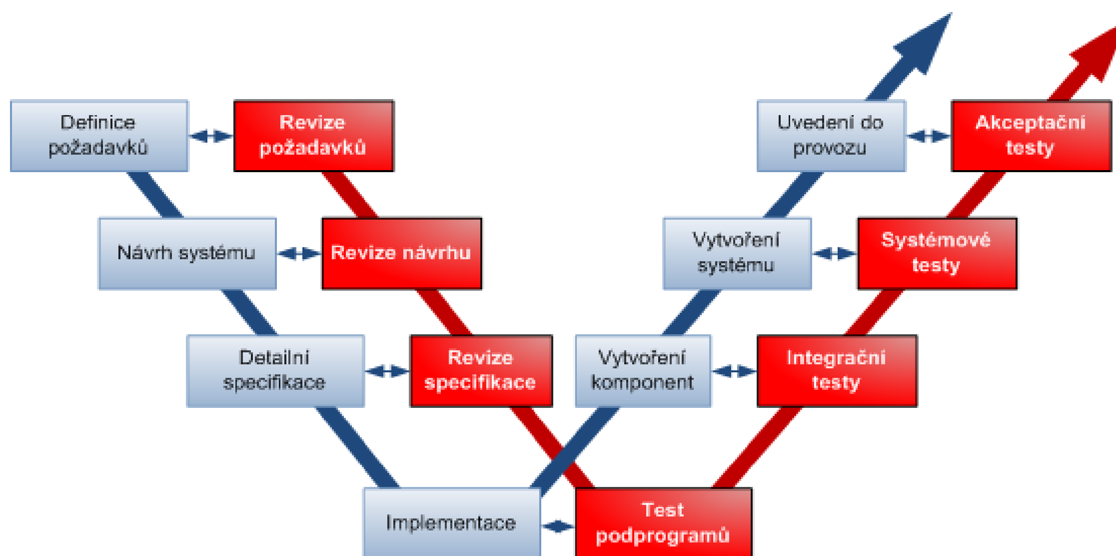
Testování softwaru není jednorázový proces. Je spjato s celým vývojem softwaru. Jeho úkolem je zkontrolovat vytvořený software, a v něm nalézt (v ideálním případě všechny) softwarové chyby. Obecně totiž platí, že čím dříve je chyba v softwaru během vývoje nalezena, tím levnější je její oprava. Proto je kladen důraz na to, aby byl software maximálně otestován, než je předán zákazníkovi. Tato kapitola využívá školící prezentace od firmy Unicorn a.s. (6).

„Různé druhy testování slouží v průběhu procesu vývoje k ověření, že daná fáze vývoje proběhla správně a že výsledky odpovídají očekávání. Vztah mezi fázemi vývoje a druhy testování se často zobrazuje v podobě tzv. ***V modelu****.“* (7)



Obrázek 2 - V model (8)

Levá část obrázku poukazuje na jednotlivé fáze, kterými si prochází vývoj softwaru. V pravé části pak naleznete jednotlivé druhy testování, které se využívají právě v jednotlivých fázích vývoje. K detailnějšímu pohledu na tuto strukturu se využívá tzv. **W model** (8). Ten znázorňuje charakteristické prvky u jednotlivých částí.



Obrázek 3 - W model (8)

3.2.1 Unit testy

Unit testy se využívají v první fázi testování. Typicky se testují nejmenší části aplikace, především komponenty na úrovni základního kódu jako jsou moduly, objekty a třídy. Tyto testy obvykle provádějí samotní vývojáři. Ti se snaží dokázat, že jejich implementovaný kód v daném softwaru funguje správně a dle specifikace. Tento druh testování nebývá zahrnut do plánů na otestování softwaru. Pomáhá rovněž ověřit základní konzistentnost a správnost při následném rozšiřování funkčnosti.

3.2.2 Assembly testy

Assembly testy lze je zařadit mezi Unit testy, jež testují vývojáři, i mezi testy, které provádějí testeři. Úkolem je ověřit, zda jednotlivé části softwaru lze pospojovat do funkčního celku. Jedná se o testy integrace jednotlivých komponent. Provádí je zpravidla vývojář, který se snaží sestavit kód do provozuschopného uskupení.

3.2.3 Systémové a integrační testy

Systémové a integrační testy jsou nejdůležitější a nejpodstatnější částí během testování softwaru. Tyto testy lze podle (7) rozdělit do několika základních podskupin:

1) **Smoke testy –** Předtím, než je započato samotné testování softwaru je dobré si ověřit, že tento software lze vůbec testovat. K tomuto slouží právě Smoke testy. Zjišťují stav aplikace, například po jejím nasazení. Jde o rychlé testy, které obsahují jednoduchý průchod základními funkčnostmi softwaru (typicky zobrazení GUI, přihlášení, zpracování základní sady dat, apod). Většinou je provádí tester nebo osoba zodpovědná za nasazení softwaru na server.

   Smoke testy je dobré zautomatizovat, protože se často opakují – dokáží rychle ověřit stav aplikace při každém upgradu softwaru (iterativní vývoj aplikace, oprava chyb, apod). Zautomatizováním těchto testů je docíleno toho, že software je otestován pokaždé stejným postupem.

2) **Integrační testy –** Lze je rozdělit na dva typy. Vnitřní testy spočívají ve vzájemné komunikaci jednotlivých částí softwaru. Vnější testy slouží k propojování jednotlivých softwarů do větších provozuschopných oddílů. Jak u vzájemné komunikace, tak u propojování jednotlivých softwarů je nutné provádět integrační testy. Mají tedy za úkol zrevidovat korektní komunikaci jednotlivých modulů (aplikací).

   Právě integrace je nejkritičtější místo v jednotlivých fázích vývoje. Proto jsou také integrační testy velice podstatné u každého testování softwaru a nedají se nahradit jinými testy.

   Zahrnují nejprve jednotlivé moduly a posléze směřují ke složitějším celkům. První fází těchto testů je prověřit rozhraní jednotlivých modulů. V této fázi lze využít tzv. „Fake moduly“. Jde o simulátory, které napodobují komunikaci ostatních modulů, ale bez jejich funkčnosti. S

pomocí výše zmíněných modulů lze tedy zjistit správnou komunikaci softwaru.

Druhou fází je jejich spojování. Tyto moduly se spojují do větších celků. Spojují se pouze ty, které mají z pohledu testování nějaký důvod.

Poslední fází integračních testů je otestování kompletního softwaru. Z pohledu vnější integrace se jedná o náročné testování. V úvahu se musí vzít to, že k integraci softwaru může docházet od různých výrobců. Tato fáze je proto velice náročná na spolupráci všech vývojových týmů. Na kvalitě součinnosti všech týmů záleží nejvíce.

3) **Systémové testy** – v dnešní době se využívají nejvíce. Slouží k ověření funkčnosti aplikace jako celku. Zákazník vytvoří spolu s vývojovou společností funkční specifikace, ve kterých je popsáno správné chování aplikace. Tester posléze postupuje při testování pomocí tohoto dokumentu. Zjišťuje, zda aplikace plní požadavky ze strany zákazníka a zda vrací správné výstupy. Testují se i nestandartní situace, na které by mohl uživatel narazit při každodenním používání. Systémové testy probíhají v několika fázích. Pokud se nalezne chyba, tester reportuje tuto chybu typicky v tiketovacím systému[2]. Chyba se posléze opraví a musí být znovu přetestována, pro ověření správného chování.

3.2.4 Akceptační testy

Akceptační testy využívá zákazník k ověření výsledné kvality softwaru. Pomocí nich se ověřuje, zda aplikace splňuje všechny podmínky a funkčnosti, které zákazník stanovil. Obrovskou výhodou, téměř nutností, je, aby danou aplikaci otestoval tým ze strany zákazníka, případně nezávislá autorita. Není to však podmínkou, proto akceptační testy mohou provádět i týmy, které realizovaly systémové testy. Následné posouzení kvality softwaru podléhá tzv. Akceptačním kritériím – předem definované kvalitě softwaru, nutné pro akceptaci zákazníkem. Tato kritéria typicky

[2] Systém pro správu chyb

obsahují seznam funkčností, testovacích scénářů a maximální množství chyb různých kategorií, které může testovaný software obsahovat, aby prošel těmito kritérii. Dále mohou obsahovat například výkonové a zátěžové požadavky. Pokud testy projdou danými podmínkami, vývoj aplikace může být ukončen a aplikaci přebírá zákazník (tím však životní cyklus nekončí, aplikace přechází do produkčního provozu s definovanou podporou).

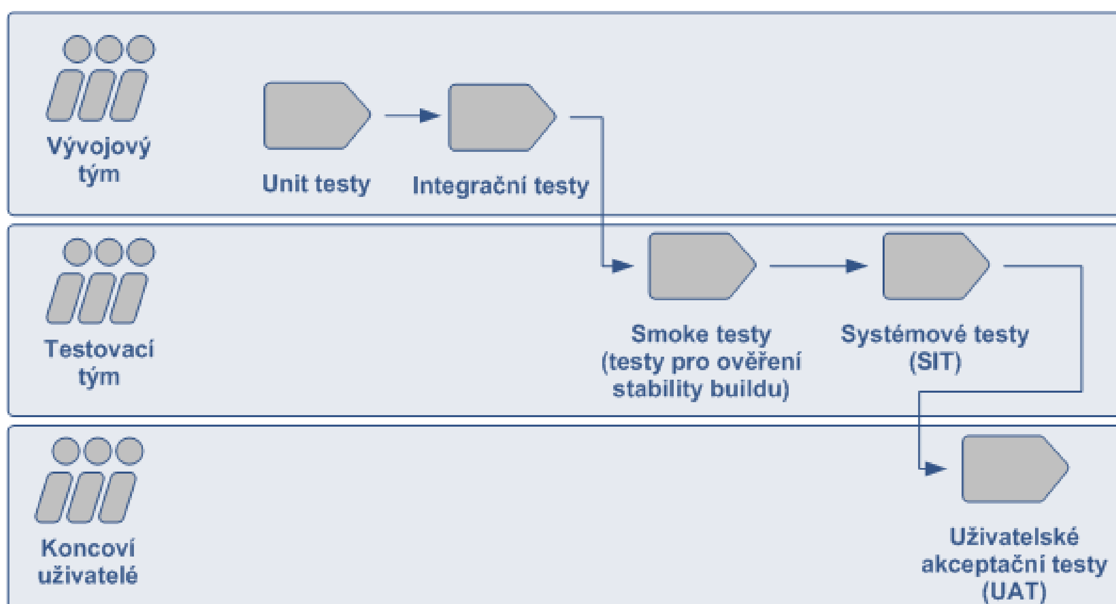
3.2.5 Testy po akceptaci

Je dobré podotknout, že ani po akceptačních testech nemusí být vývoj softwaru u konce. Přestože akceptační testy projdou požadavky, je velice pravděpodobné, že aplikace bude obsahovat skryté nebo doposud neodhalené chyby. Proto je v zájmu vývojového týmu nalézt a opravit je co nejdříve. Je nutné si uvědomit, že chyba v produkčním prostředí může kromě nekonzistence dat způsobit zákazníkovi i nezanedbatelné finanční ztráty. Testy po akceptaci se soustředí na analýzu provozních událostí, logů atp. s cílem odhalit skryté chyby nebo podezřelé chování. S touto činností souvisí i potřeba opětovného testování. Další možností je, že zákazník může rozšířit funkčnost aplikace. V obou těchto variantách je potřeba ji znovu a znovu testovat. Lze přitom využít již zmiňované testy, ale v tuto chvíli se nejčastěji využívá další způsob testování tzv. regresní testy.

3.2.6 Regresní testy

Regresní testy se využívají po opravě chyb softwaru nebo po přidání nové funkčnosti. Tyto testy kontrolují, zda zásahy, které se uskutečnily v aplikaci, nezpůsobily nesprávné fungování vyvíjeného softwaru. Dále se ověřuje, jestli nově přidané funkčnosti pracují správně v rámci celého systému.

Regresní testy lze zautomatizovat, jelikož se jedná o opakovatelné operace se známým výsledkem. Cílem těchto automatizovaných testů je zjistit, zda v průběhu změny aplikace nevznikla odchylka mezi výstupem získaným před změnou a výstupem po změně.



Obrázek 4 - Postup testů podle (6)

3.3 Způsoby testování

V této kapitole jsou popsány způsoby testování, podle již zavedených termínů. Na tyto termíny se odkazuje (9). Následný text v jednotlivých způsobech je zpracován za pomoci (10), (11) a (12).

3.3.1 White box testování

Používané též jako glass box, clear box, open box či strukturální box. White box testování předpokládá, že tester bude znát vnitřní strukturu softwaru.

Podrobněji řečeno, white box testování vyžaduje znalost vnitřních datových struktur. Nedílnou součástí pro tento typ testování jsou i znalosti o programových strukturách a dále je třeba znát, jak je daná aplikace naimplementována. Tester je zaškolen a poté mu jsou předány všechny informace, které potřebuje k otestování. To znamená, že mu je poskytnuta příslušná dokumentace, binární a zdrojový kód celého softwaru. Součástí tohoto testování je porozumění poskytnutého kódu a jeho analyzování. Tento způsob je občas označován jako audit zdrojového kódu. Testování může probíhat zcela automaticky nebo se dá provádět ručně. Existují dva typy analytických nástrojů. Prvním typem jsou nástroje, které potřebují zdrojový kód. Dalším typem jsou ty nástroje, které jsou schopny provést automatickou

dekompilaci binárního kódu. Zdrojový kód je vyroben automaticky a po částech je analyzován.

White box testování se používá zpravidla na počátku vývojového cyklu. Tento typ je velice vhodný k použití u webových služeb. Vývojář může spolupracovat s testerem. Společně tak mohou nalézat nežádoucí chyby. White box test se hlavně využívá na testování odolnosti. Ověřuje se, zda je daný software ochráněn před neautorizovaným přístupem a zda je chráněn přístup k jednotlivým segmentům softwaru či k uloženým datům.

Mezi výhody můžeme zařadit podle (10) například:

- Včasné odhalování chyb – Díky analýze zdrojového kódu lze odhalit chyby, kterých se programátor dopustil při vývoji softwaru.
- Odhalení nežádoucího kódu – Díky znalosti zdrojového a binárního kódu lze otestovat kód, zda neprovádí nežádoucí operace (špionáž, možný únik citlivých dat, manipulace s finančními toky apod.).

Mezi nevýhody můžeme zařadit podle (10) například:

- Náročnost na testování – Mimo jiné vyžaduje znalost zdrojového a binárního kódu, testovacích nástrojů a programovacích jazyků.
- Časová náročnost – Důkladné testování zabere mnoho času, jelikož se testuje velké množství kódu.
- Vysoké náklady – Kvůli časové náročnosti a specializovaným nástrojům rostou i náklady.

3.3.2 Black box testování

Označované též jako opaque box, closed box, behavioral nebo funkční box. Tento typ testování nepředpokládá znalost interních datových a programových struktur.

Black box testy slouží k testování softwaru bez znalosti binárního a zdrojového kódu. K tomuto typu testování poslouží testerovi testovací scénáře. Tyto scénáře si může vytvořit sám tester. Podle chování aplikace a vstupních dat lze určit i hodnotu, kterou by měla výstupní data obsahovat. To pomáhá testerovi porovnat, zda daný test byl úspěšný nebo ne. Black box testy mohou být testovány ručně, nebo

automatizovaně za použití vhodných nástrojů. Tento typ testu lze využít v aplikacích, kde známe vstupní a výstupní hodnoty.

Mezi výhody můžeme zařadit podle (12) například:

- Snadnost – Není potřeba znalost programovacích jazyků.
- Rychlost – Lze otestovat rozsáhlé systémy v krátkém čase.
- Transparentnost – Je srozumitelný pro zákazníka. Dokáže ho otestovat sám.
- Testovací scénáře mohou být napsány až v okamžiku, kdy je dokončena specifikace daného softwaru.
- Po změně programovacího jazyka, operačního systému a hardwaru, nedochází ke změně testovacích scénářů.
- Tester dokáže otestovat daný software bez znalosti zdrojového a binárního kódu.

Mezi nevýhody můžeme zařadit podle (12) například:

- Nižší kvalita kódu – Netestuje se efektivita napsaného kódu.
- Nežádoucí chování – Netestuje se nežádoucí chování kódu.

3.3.3 Grey box testování

Používané též jako translucent box. Tento typ testování předpokládá menší znalost interních datových a programových struktur. Využívá se na navržení vhodných testovacích scénářů.

Grey box je střední cesta mezi black box a white box testováním. Nelze říci, že se jedná o black box, protože tester zná některé vnitřní struktury. Nelze ani říci, že se jedná o white box testování, jelikož znalost vnitřních struktur není úplná. Grey box poskytuje velice jednoduchý proces testování. Testerovi stačí znát, jak software funguje uvnitř, a následně ho může detailněji otestovat zvenku. Stejně jako black box testování, je grey box testování prováděno zvenku. Nicméně tester má větší informace o tom, jak jednotlivé části produktu pracují, a jak je software provázán.

Grey box test se využívá při integračním testování dvou modulů od dvou různých dodavatelů. Dalším případem využití těchto testů je při testování vícevrstvé

aplikace, kde tester má kontrolu nad vstupem, výstupem a přímým vstupem do databáze. Tímto způsobem tak lze otestovat všechny tři hodnoty: vstup, hodnotu v databázi a výstup. Lze zjistit, kde dochází k manipulaci s výsledky. Může se stát, že manipulace je na straně klienta, aplikace nebo na straně databáze. V databázi může docházet k chybě v zápisu a čtení, nebo k chybě přímo v databázi, kde například trigger způsobí spuštění nějakého uloženého procesu. Dále se dají otestovat HTML formuláře, analyzovat skripty nebo práci s cookie. Grey box se také využívá na testování, pokud nechceme poskytnout zdrojové ani binární kódy. Současně nechceme, aby tester ztrácel čas se skenováním sítě a inventarizací. V tomto případě jsou testerovi poskytnuty potřebné informace.

Mezi výhody můžeme zařadit podle (11) například:

- Slučuje black box i white box přístup.
- Neintrusivní – Tester nepotřebuje přístup ke zdrojovému ani binárnímu kódu. Stačí mu znát pouze funkční specifikace, rozhraní a architekturu aplikace.
- Inteligentní testy – Díky znalostem testera, byť omezenými, dokáže lépe odhadnout okrajové testovací scénáře.

Mezi nevýhody můžeme zařadit podle (11) například:

- Neúplné otestování – Tester nezná zdrojový ani binární kód. To mu zabraňuje otestovat všechny datové toky.
- Kvalita kódu – Kvůli neznalosti zdrojového kódu a binárního kódu nelze říci, zda kód je efektivní, či zdali aplikace neobsahuje žádný nežádoucí kód.

4 Automatizované testování

Automatizované testování softwaru je podle (13) stále velmi podceňovanou disciplínou. Nicméně začíná se čím dál více využívat, jelikož se prokazuje jeho efektivita a nezávislost. Pokud se testuje manuálně, je zapotřebí pracovníka, který postupuje podle určitého návrhu. Zabere to čas a celkové náklady rostou s opakujícím se testováním. Firmy si stále nedokáží uvědomit, že manuální test, který se opakuje a má stejné výsledky, lze zautomatizovat.

Zde přichází obrovská výhoda automatizovaných testů. Pokud se firma rozhodne, že se určitý postup zautomatizuje, bude výsledkem v průběhu času podstatné snížení nákladů, než kdyby se muselo testovat manuálně. I když se zpočátku zdá, že na vytvoření automatizovaného testu se spálí více času, než by zabralo manuální otestování, jeho budoucí využití nám zajistí časovou úsporu a ušetření peněz. Po vytvoření automatizovaného testu a implementování do systému, který dokáže použít většina uživatelů, lze testovat, aniž by to zabralo čas testera.

Automatizace má i určité nevýhody. Za jednu z nevýhod lze považovat neustálou údržbu. I přes snahu mít automatizované testy vždy plně soběstačné, ne vždy se to daří. Pokud je do aplikace zařazena změna, která má vliv na postup testu, který je zautomatizován, musí být upraven i tento test. Pokud se tak nestane, veškeré testy, které budou spuštěny, doběhnou neúspěšně. Další nevýhodou je, že se nemusí přijít na všechny chyby, jelikož automatizovaný test má jasný výsledek, kterého se snaží dosáhnout. Když ho dosáhne, skončí úspěšně, nicméně se může stát, že i po úspěšném dokončení se během testu vyskytla chyba, která nebyla odhalena. Tato chyba nemusí být nějak závažná, jelikož se na ni test sám o sobě nezaměřuje. Test se dá nastavit tak, aby kontroloval důležité body během testu, ale i to nezabraňuje nějaké jiné chybě, kterou nepředpokládáme.

4.1 Fáze vývoje automatizovaného testu

V této kapitole je popsán detailnější postup při vytváření a implementaci automatizovaného testu. Podle předchozích popisů se může jednat například o vytvoření automatizovaného Smoke testu za pomocí Black box testování.

4.1.1 Seznámení se softwarem

Prvním úkolem každého pracovníka, který vytváří automatizované testy, je seznámit se s vyvinutým softwarem. Každý tester musí pochopit fungování testovaného softwaru. K tomu pomáhá testovací dokumentace, ze které je nejdůležitější složkou pro testera dokument, obsahující jednotlivé Test Casy.

Náplní práce každého pracovníka je zhodnotit jednotlivé Test Casy a vytipovat testy, které má smysl automatizovat. Test architekt dané aplikace potvrdí, zda tyto testy budou automatizovány či nikoli.

Při rozhodování musí brát v úvahu časovou náročnost na testování jednotlivého Test Casu, počet testů (každý update[3] softwaru, který musí být testován) a čas, který zabere vývoj automatizovaného testu. Matematicky si musí ověřit, že počet hodin strávený na vývoji a údržbě tohoto testu bude menší než počet hodin každého testera, který bude testovat stejný Test Case po každém updatu. Na základě toho se aktualizuje testovací plán, podle kterého se daná aplikace testuje.

4.1.2 Vytvoření hrubého automatizovaného testu

Po schválení navržených Test Casů se prvotně vyberou tzv. Basic Flows[4]. Každý pracovník, který vytváří hrubou kostru jednotlivých automatizovaných testů, se snaží o co nejefektivnější postup. Základní navržení hrubé kostry znamená uvědomění si jednotlivých částí, které mohou mít stejnou či podobnou strukturu. Jedná se o to, aby se ušetřilo co nejvíce času a námahy najednou. Zároveň to umožňuje opravovat chyby v hrubé kostře pouze na jednom místě. To přináší obrovskou výhodu v implementaci jednotlivých testů. Hrubý automatizovaný test musí zvládnout všechny kroky, které jsou uvedené v navrženém scénáři v Test Casu.

4.1.3 Doladění detailů v testu

Je rozdíl, zda byl automatizovaný test vytvořen na Frontend testování nebo na Backend testování. Pokud se test vytvářel na Frontend testování, neboli na testování například GUI celého systému, test je převážně navržen na kontrolu správného

[3] Aktualizace, zdokonalení
[4] Základní operace softwaru

fungování jednotlivých tlačítek. Pokud byl test vytvořen na Backend testování, tedy na testování například Web Service nebo API funkčností, je zde převážně využito odkazování na jednotlivé soubory. Vše závisí na individuálním přístupu k jednotlivým Test Casům, tedy jak přesně se má kontrolovat výsledek jednotlivého Test Casu.

V případě, že automatizovaný test projde všechny kroky podle scénáře, neznamená to ještě, že je optimální. Hrubý test se většinou odkazuje na určitá ID[5] jednotlivých tlačítek či souborů. Jedná se totiž o nejjednodušší způsob odkazování. Mohlo by se stát, že poslední aktualizace softwaru měla za následek prohození identifikačních zkratek jednotlivých tlačítek (např. prohození hodnot v tabulce nebo v menu), což by mohlo mít za následek, že v hrubém automatizovaném testu selhal určitý krok.

V optimálním testu se snažíme vyvarovat odkazování pomocí jednotlivých ID. Doporučuje se odkazovat přímo na určité názvy, jelikož ty se nejméně často mění. V případě prohození míst nehrozí, že krok v testu selže.

Nejčastějším způsobem, který slouží k odkazování názvů, nahrazování textu a k vyhledávání, je využití již zmiňovaných regulárních výrazů. Především se jedná o výrazy, které usnadňují práci s textem v dané aplikaci. Tento způsob funguje jako nahrazování určitého textu pomocí znaků (tzv. metaznaků). Nejčastěji se lze setkat s variantou, kdy je známa část textu, za kterým následuje proměnná, která je hledána. Může se jednat o náhodné názvy, identifikátory či hodnoty, které jsou ověřovány nebo používány dále. Využívá se jednoduchého postupu. Pokud je známa část textu, za kterým následuje proměnná, zadáním speciálních znaků místo této proměnné a následném spuštění se zobrazí výsledná hodnota v daném čase.

Mezi další nejčastěji používanou technikou je vyhledávání přes XPath. Pomocí něho se lze odkazovat na potřebné části přímo v daném HTML. Využití této metody je detailněji popsáno v kapitole 6.2.3.

---

[5] Zkratka pro identifikaci

4.1.4 Integrace do celého projektu

Poslední částí je integrace testů. Vytvořený a odzkoušený automatizovaný test se musí upravit tak, aby jej mohl využít každý, kdo potřebuje. Tím se zamezí tomu, že by musel znát nástroj, který byl využit k vytvoření automatizovaného testu. Většinou se automatizovaný test integruje do vývojového prostředí, resp. řetězce nástrojů, které používají všichni pracující na daném projektu. Stisknutím jednoho tlačítka v aplikaci se spustí celý test. Spuštění spočívá v tom, že se vytvoří agent, na kterém se spustí nástroj s automatizovaným testem. Ten musí být nastaven tak, aby všechny výsledky postupně zaznamenával a dále je předával zpět do aplikace. V ní se potom výsledky shodují s těmi, co nalezl automatizovaný test. Daný tester následně může dohledat zcela jednoduchým způsobem, kde se nalézá daná chyba.

Další možností automatizovaného testu je využití zápisu dat do grafu. Jednotlivé grafy se týkají například využití paměti při manipulaci s daty z databáze do aplikace a obráceně v reálném čase. Některé nástroje dokonce dokáží nasimulovat větší počet uživatelů. Tím se provádí kontrola softwaru při reálném zatížení. To pomáhá při důsledné kontrole a optimalizaci celého navrženého softwaru.

5 Nástroje pro testování

Existuje spousta nástrojů, které jsou užitečné při testování. Tyto nástroje většinou zefektivňují a usnadňují testerovi práci. Tato kapitola je zaměřena pouze na popis vybraných testovacích nástrojů a jejich základní využití.

5.1 Firebug

Firebug je opensource nástrojem pod licencí BSD[6], který je rozšířením webového prohlížeče Mozzila Firefox. Lze použít také verzi Firebug Lite, kterou lze spustit i v internetových prohlížečích Internet Explorer, Google Chrome, Safari nebo Opera.

Jeho instalace je jako u všech doplňků Firefoxu jednoduchá. Stačí si pouze stáhnout požadovaný soubor a instalace proběhne automaticky.

Tento nástroj je univerzální a lze jej využít v mnoha případech. Jeho schopnosti lze popsat několika způsoby. První funkcí, kterou nástroj dokáže, je kontrola a úprava HTML kódu. Dále práce s CSS, kde lze zobrazit jednotlivé styly a případně lze uskutečnit editaci jednotlivých elementů. Dovede zobrazit měřítka na obrazovce, sledovat všechny dotazy, které stránka vyvolá (síťový monitoring). Firebug také obsahuje ladící nástroj pro JavaScript, včetně konzole pro příkazy, okno pro zobrazení nalezených chyb či stromově procházet DOM[7].

Tento nástroj je především určen pro vývojáře, ale dá se také právě využít při testování webových aplikací.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (14), (15), (16) a (17).

5.2 Visual Studio Test Professional

Balíček od společnosti Microsoft, jehož nástrojem je například Microsoft Test Manager, je skvělým řešením, pro správu životního cyklu projektu. Jeho obrovskou výhodou je propojení s ostatními nástroji této společnosti, jako jsou například Visual Studio Team Foundation Serveru, Visual Studio Online a spousty jiných nástrojů,

---

[6] Licence pro svobodný software.
[7] Záložka DOM, ve kterém se nachází inspector DOM a Ajax.

které se podílejí na testování softwaru. To umožňuje koordinovat všechny činnosti týkající se testování a zefektivnit tak celkový postup při testování softwaru.

Nástroj se využívá při průzkumných a ad hoc testech. U těchto testů nejsou potřeba žádné předdefinované testovací kroky ani případy. Nicméně, tento nástroj dokáže zaznamenávat akce, komentáře s poznámkami, zachytávat data či nahlašovat chyby. Ze záznamů nahodilých testů může posléze vytvářet testovací případy a začleňovat je do testovacích plánů.

Více detailnějších informací o tomto nástroji se naléźá v odkazu (18) a (19).

5.2.1 Microsoft Test Manager

Tento testovací nástroj je dalším příspěvkem od společnosti Microsoft. Spolupracuje s již zmiňovaným Visual Studio Test Professional. Jeho hlavním úkolem je spouštění, zaznamenávání a opakování ručních testů.

Tento nástroj je plně konfigurovatelný. Spolu s Test Runnerem spravuje podrobnější záznamy o již uskutečněných krocích. Je zaznamenáno chování nebo stav jednotlivého kroku. Test je možné kdykoliv přerušit za účelem nahlášení chyby. Tato nahlášená chyba se dostane k vývojovému týmu spolu se všemi technickými údaji, které slouží k opakovatelnému nasimulování dané chyby. Podstatnou výhodou je, že jednotlivé testy lze zaznamenávat a později je přehrát. To pomáhá urychlit testování.

Více detailnějších informací o tomto nástroji se naléźá v odkazu (20).

5.2.2 Release Management

Nedílnou součástí každého testování je nástroj, který slouží k zaznamenávání všech nalezených chyb. Právě Release Management je jedním z řešení, do kterého se dají zasílat tyto chyby. Tento nástroj spolupracuje s již zmíněnými nástroji společnosti Microsoft. Vzájemná komunikace je obrovskou výhodou, jelikož systémy se dají propojit tak, že mnoho činností je zcela automatických.

Pokud tester nalezne chybu, sepíše ji a následně uloží. Propojení mezi testovacím a tiketovacím nástrojem zajistí, že tato chyba bude uložena do přehledu o vyvíjeném softwaru.

Tento nástroj slouží i k získání přehledu o jednotlivých nasazeních. U tohoto nástroje lze celé nasazení nové verze softwaru zautomatizovat. Jednotlivé nasazení lze detailněji prozkoumat a zobrazit výsledky o úspěšném či neúspěšném pokusu.

K jednotlivým nasazením lze přiřadit grafy, které se skládají z průběhu testování a určují tak kvalitu nasazené verze softwaru.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (21).

5.3 TestComplete

Jedná se o testovací nástroj, který vyvinul SmartBear Software. Nástroj se především využívá při automatizovaném testování. Vytváří spolehlivé GUI testy. Lze si vybrat, zda vytvoříte skripty pomocí JavaScript, Python, VBScript nebo pomoci spousty jiných skriptovacích jazyků. Tento nástroj podporuje Oracle Forms a integruje například nástroj SoapUI pro automatizaci testů. Tento nástroj jde rozdělit do tří základních modulů.

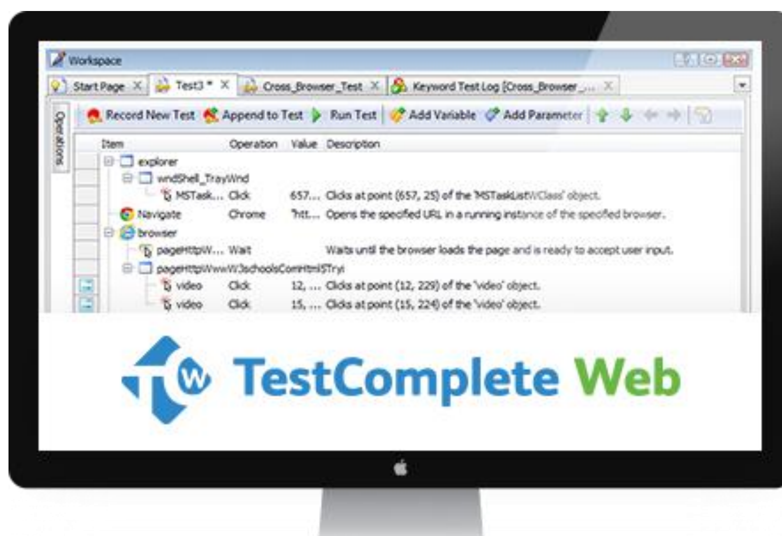
První modul je pro Windows Desktop Applications & Packaged Apps neboli pro desktopové Windows aplikace a balíčkové aplikace. Podporuje velké množství desktopových aplikací, například .NET, C++, Java nebo Oracle Forms. Využívá možnosti vytvářet UI testy pomocí metody „nahrát a spustit", nebo vytvoření přes skriptovací jazyky. Obsahuje inteligentní mechanismus, který rozpoznává objekty a tím tak zajišťuje, že veškeré testy budou snadno udržovatelné.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (22), (23) a (24).

Obrázek 5 - Desktop module (23)

Druhý modul je určen k vytváření automatizovaných testů pro standartní a mobilní web. Výhodou tohoto modulu je spouštění na všech možných prohlížečích. Pro vytvoření tohoto typu testů lze využít šesti jazyků, včetně Pythonu nebo JavaScriptu. Dále tento modul podporuje i začlenění Selenia, které je popsané níže. Modul využívá dalšího inteligentního mechanismu, který rozpoznává objekty u dynamických prvků a technologií. Nalezne Flex, Flash, AJAX, JavaScript, HTML5 a AngularJS. Pokud využijeme možnosti propojení se serverem, řídícím zdrojem systému a nástrojem pro správu chyb, lze spouštět tyto testy po celý den, a to bez přítomnosti fyzické osoby.

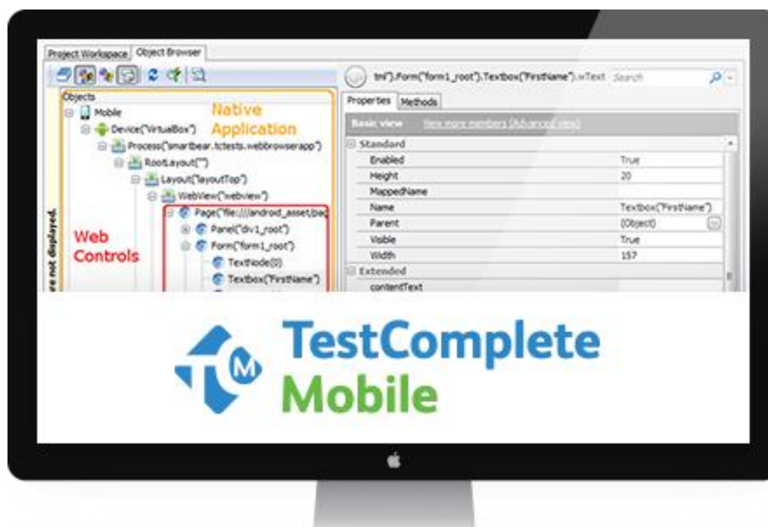
Více detailnějších informací o tomto nástroji se nalézá v odkazu (25).



Obrázek 6 - Web module (25)

Poslední modul je provázán s testováním nativních a hybridních mobilních aplikací. Tento nástroj podporuje jak testování operačního systému Android, tak operačního systému iOS. K otestování nejnovější verze iOS je zapotřebí Swift, což je nový programovací jazyk od společnosti Apple. Proto i tento jazyk je součástí tohoto modulu. Další výhodou je vytvoření jednoho testu, který lze spustit na více zařízeních – ty se mohou odlišovat například velikostí obrazovky, což zaručuje otestování na reálných telefonech a tabletech, které zákazník využívá. Podobně jako u webového modulu, lze si vybrat ze šesti skriptovacích jazyků. Další zajímavou funkcí je záznam gest, která se provádějí při testování. Tato gesta jsou zaznamenávána a posléze si je lze přehrát pro přesné nasimulování testovacího postupu.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (26).



Obrázek 7 - Mobile module (26)

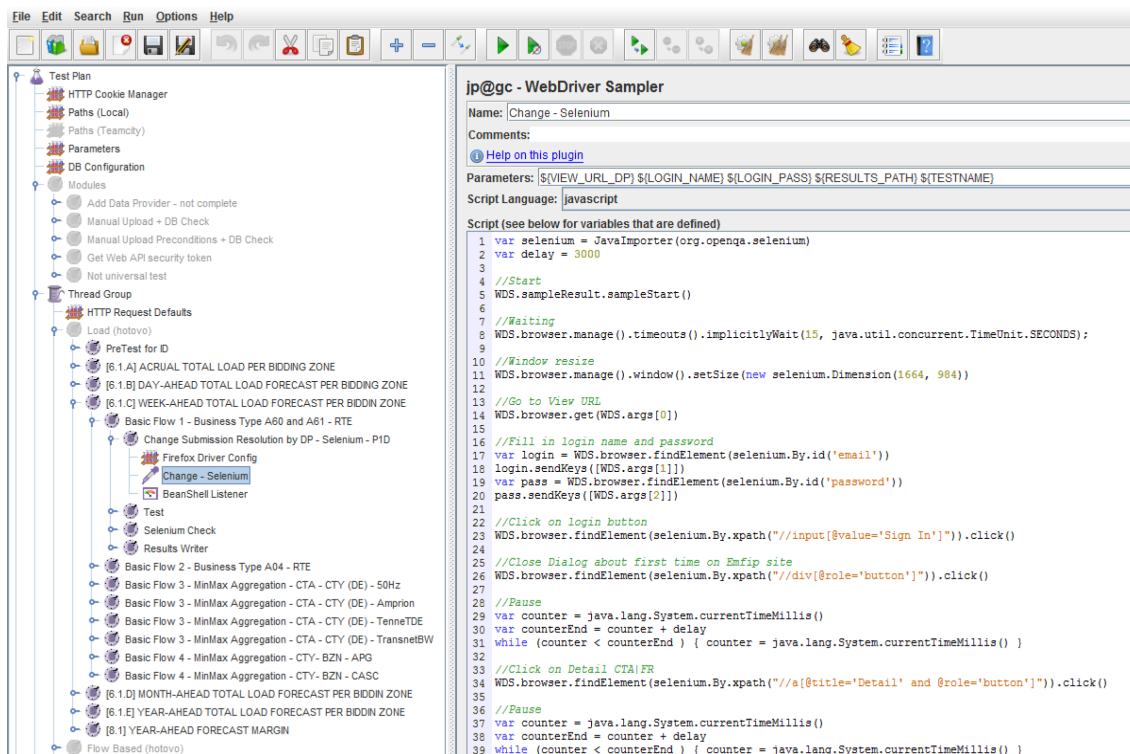
5.4 Apache JMeter

Tento open source software od společnosti Apache Software Foundation je dalším často využívaným nástrojem, který je určen pro testování vyvinutých aplikací. Tento nástroj dokáže vyvinout i zátěžové testy. Pomocí něj lze prozkoumat databázi nebo vykreslit různé grafy, které pomáhají k přehlednějšímu zobrazení dosaženého výsledku. JMeter se dá rozdělit na dvě části.

První část je zaměřena na samotnou funkci tohoto nástroje. Po stažení JMeteru stačí pouze vytvořit přehledný strom s požadovanými elementy, které jsou potřeba k nahrání určitého testovacího postupu. Po propojení s požadovaným webovým prohlížečem stačí pouze tento nástroj zapnout. V prohlížeči se nakliká požadovaný testovací postup a je skoro hotovo. JMeter každý krok, který se provede, nahraje. Poté stačí tyto kroky optimalizovat a automatizovaný test je připraven ke spuštění.

Druhou částí je spojení JMeteru s dostupnými pluginy. Ty dokáží rozšířit základní nástroj a umožnit tak kombinaci s jinými prvky. Jednoduchá implementace do nástroje zaručuje uživatelsky přívětivé rozhraní. Po stažení pluginu a začlenění do knihovny JMeteru se změny projeví okamžitě, a to v nabídce možností, které si vybíráme v kořenové struktuře nástroje. Jedním z těchto rozšíření je Selenium. Toto rozšíření umožňuje využívat WebDriver pro konkrétní webový prohlížeč. To umožňuje vytvářet automatizované testy ve více variantách.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (27).

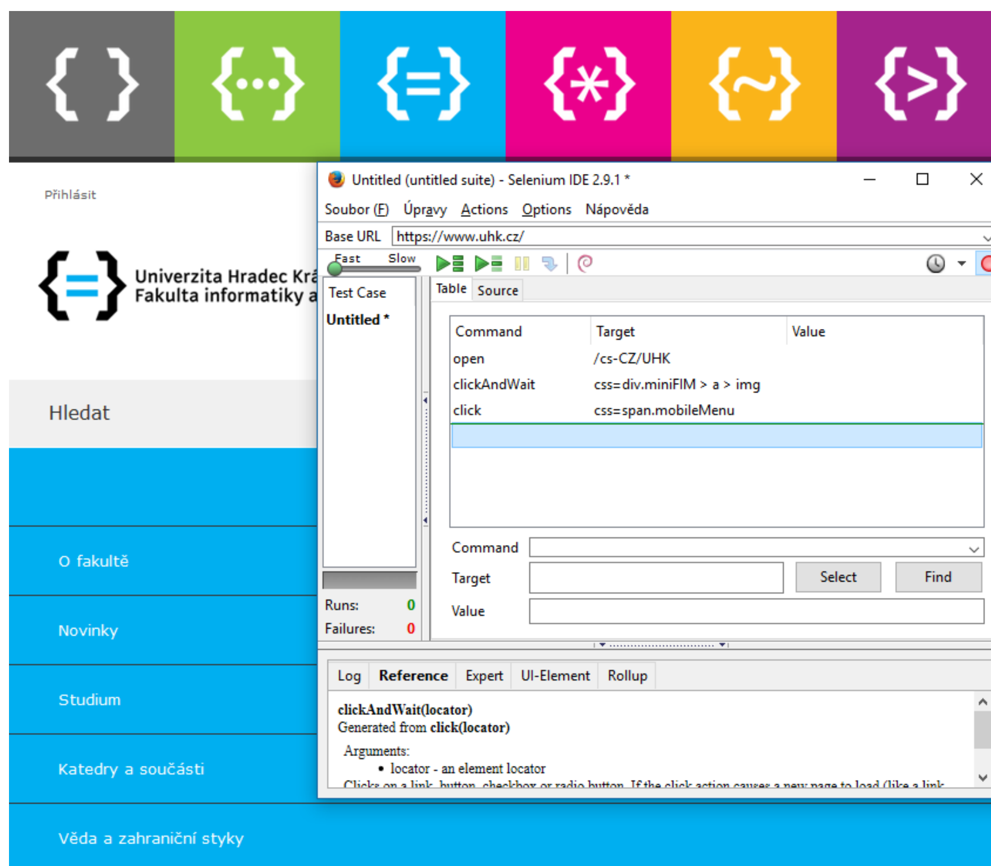


Obrázek 8 - Selenium plugin v JMeteru (Zdroj: vlastní)

5.5 *Selenium*

Jedná se o nástroj pro testování webových aplikací. Vyvinut je v programovacím jazyku Java, a jak již bylo řečeno, je možné ho integrovat i do jiných nástrojů. Selenium se dá rozdělit do několika komponent.

První komponentou je Selenium IDE. Původně se jednalo o plugin do internetového prohlížeče Mozilla Firefox. Pomocí tohoto způsobu lze nejjednodušeji nahrát jednotlivé kroky testu a tím tak následně vytvořit automatizovaný test. Jedná se o to, že nástroj sleduje jednotlivé kroky, které uživatel dělá v prohlížeči. Nahrává je a po dokončení těchto kroků vzniká automatizovaný test, který si lze spustit a nasimulovat tak do detailu předchozí postup. Nevýhodou je, že tuto komponentu lze pouze spouštět právě v již zmiňovaném webovém prohlížeči Mozilla Firefox.



Obrázek 9 - Selenium IDE (Zdroj: vlastní)

Další komponentou je Selenium RC. Jedná se o nástroj, který využívá více programovacích jazyků k vytváření testů. Další změnou oproti Seleniu IDE je, že testy se dají spouštět na více internetových prohlížečích. Selenium RC využívá Selenium server, který překládá příkazy z testu, posílá je pomocí JavaScriptu Injection a výsledek testu zobrazuje zpět testovanému programu. Využity jsou přitom dotazovací metody GET a POST jazyka HTML.

Selenium WebDriver je další komponentou tohoto nástroje. Základním prvkem je jednodušší přístup k vytváření automatizovaných testů. Není nutné používat Selenium server ke spouštění těchto testů. WebDriver volá každý prohlížeč sám o sobě a to znamená, že je nutné upravovat tuto komponentu podle používaného prohlížeče. Právě tyto Drivery využívá již zmiňovaný Apache JMeter.

Poslední komponentou je Selenium Grid. Jedná se pouze o možnost spouštět automatizované testy na více strojích a prohlížečích současně. To napomáhá k ušetření času, a tím také peněz.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (28) a (29).

5.6 Nástroje pro správu chyb

Jak již bylo řečeno, nedílnou součástí nástrojů pro testování jsou i nástroje, ve kterých můžeme nalezené chyby reportovat. Jedná se o správu chyb. Po nalezení chyby, je tato chyba popsána a uložena právě do nástroje, který provádí detailnější přehled o testování. To napomáhá k přehlednějšímu informování o celkovém stavu testovaného softwaru.

5.6.1 Mantis

Jedná se o open source bug tracker, neboli o nástroj pro správu chyb. Hlavní výhodou této aplikace je snadné používání. Stačí si pouze nainstalovat danou aplikaci na server a všichni uživatelé k němu budou mít přístup přes webový prohlížeč.

Uživatele lze rozdělit podle rolí ve firmě – vývojář, analytik, vedoucí projektu. To umožňuje využít řízeného přístupu, který zamezuje otevření určité části v této aplikaci osobě, pro kterou není dané oprávnění. Další výhodou je flexibilita, kterou tato aplikace disponuje. Uživatel si může nastavit své workflow podle vlastních potřeb. Upozornění na nějakou změnu lze dostávat pomocí e-mailu nebo existuje i aplikace do chytrého zařízení.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (30) a (31).

5.6.2 Jira

Tento nástroj je vytvořen firmou Atlassian. Jedná se o velice univerzální a všestrannou aplikaci, která slouží mimo jiné i k získání přehledu o chybách a k jejich správě. Obrovskou výhodou je propojenost i s ostatními produkty této společnosti. To napomáhá k efektivnějšímu přístupu během celého životního cyklu vyvíjeného softwaru.

Nástroj jde využít v několika případech. Pomocí této aplikace lze vytvořit plán o budoucího testování, celkový přehled o předchozích testech nebo zobrazit podpůrné grafy o stavu vyvíjené aplikace. Je možné oddělit i uživatelské role. To umožňuje zasílat chyby, jak ze strany zákazníka, tak ze strany vývojového týmu, který například nechce, aby zákazník viděl nalezené chyby.

Po nalezení chyby, uživatel založí novou zprávu v Jiře a detailně ji popíše. K popisu slouží již předdefinovaná šablona, kterou stačí vyplnit. Následně uživatel, který chybu nalezl a sepsal, převede tento spis (Report) na kompetentní osobu. Jira umožňuje i další komunikaci mezi zákazníkem a servisním týmem. Uživatel nemusí zakládat pouze chyby (Bug), ale lze založit i různé úkoly (Task). Všem těmto možnostem se dá určit jejich priorita, která hraje důležitou roli při upřednostňování oprav. K jednotlivým reportům lze psát různé komentáře a přidělovat je do příslušných verzí (Release) spravovaného systému.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (32).

5.6.3 TestLink

Poslední uvedený nástroj pro správu a organizaci testování softwaru je Testlink. Primárně je určen pro tvorbu Test Casů, ale dá se použít i pro tento účel.

Tento nástroj má obrovskou výhodu a v tom, že je zcela zdarma. Jedná se o svobodný software, který je šířený pod licencí GNU GPL. Je psaný v PHP a jeho kód si lze upravit podle své potřeby. Dá se propojovat s jinými nástroji pro správu (Mantis, JIRA, apod.).

Bohužel se zde najdou i jisté nevýhody. Za zmínění stojí především vazba na rodiče[8] při kopírování. Pokud si zkopírujete určitý požadavek a upravíte něco v něm, už nedochází k úpravě u rodiče. Proto se musí upravovat na více místech.

Více detailnějších informací o tomto nástroji se nalézá v odkazu (33) a (34).

[8] Původní zdroj nebo instance, ze které dochází ke kopírování.

6 Vlastní příklad automatizovaného testu

Před popisem vlastního příkladu, bude popsán postup vytváření automatizovaného testu v nástroji JMeter. Tento nástroj se dá použít k testování funkčních i nefunkčních požadavků dané aplikace.

6.1 Základní používání nástroje JMeter

Ukázka je prováděna v internetovém prohlížeči Mozilla Firefox. Dále je zde využita internetová adresa Univerzity Hradec Králové - www.uhk.cz.

6.1.1 Instalace nástroje

Instalace JMeteru je jednoduchá, stačí si pouze stáhnout balíček z internetových stánek a rozbalit ho. Tento balíček v sobě obsahuje dvě základní části. Složku lib a bin. Ve složce lib najdeme celou základní knihovnu, která je použita u tohoto nástroje. Pokud bychom potřebovali rozšířit tento nástroj o nějakou funkcionalitu, stačí si stáhnout příslušný plugin a všechny soubory překopírovat do složky lib. Složka bin obsahuje vše ostatní. Nejdůležitější soubory v této složce jsou jmeterw.cmd a jmeter.sh. Jedná se o spouštěcí soubory. Pro operační systém Windows skript jmeterw.cmd, pro systém Linux lze využít druhý spouštěcí soubor jmeter.sh.

6.1.2 Základní přehled GUI nástroje

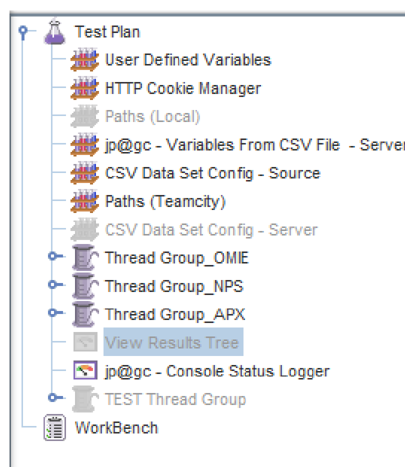
Okno aplikace se skládá ze tří hlavních částí. První částí je horní menu, které obsahuje základní operace, jež se dají využít v tomto nástroji.

Obrázek 10 – Hlavní panel nástrojů (Zdroj: vlastní)

Obrázky odpovídají funkcím, které lze najít i v boxech nad těmito obrázky. V první řadě je k dispozici možnost založení nového testu, využití různých šablon, otevření již vytvořeného testu, zavření aktuálního testu a možnost uložit aktuální test. Druhá řada obsahuje funkce, které slouží k zefektivnění tvorby testu, jako je například kopírování, možnost udělat krok zpět a podobně. Třetí řada pomáhá při práci se stromovou strukturou testu. Čtvrtá a pátá řada spouští, zastavuje nebo

ukončuje test či testy, které jsou aktuálně vyvíjené. S tím souvisí i pátá řada. Jelikož je nutné kontrolovat výsledky testu, tlačítka v páté řadě pomáhají k promazávání těchto výsledků. To slouží k lepšímu přehledu při testování. Poslední dvě řady slouží k vyhledávání určitého výrazu v celé stromové struktuře vyvíjeného automatizovaného testu a k pomoci objasnit funkcionality různých využitých modulů. Oranžový trojúhelník s vykřičníkem uprostřed zobrazuje chyby (Errors). Zobrazuje počet chyb s možností rozkliknout konzoly – kliknutí na ikonku trojúhelníku. Poslední symbol udává status, který ukazuje počet aktivních uživatelů z celkového počtu.

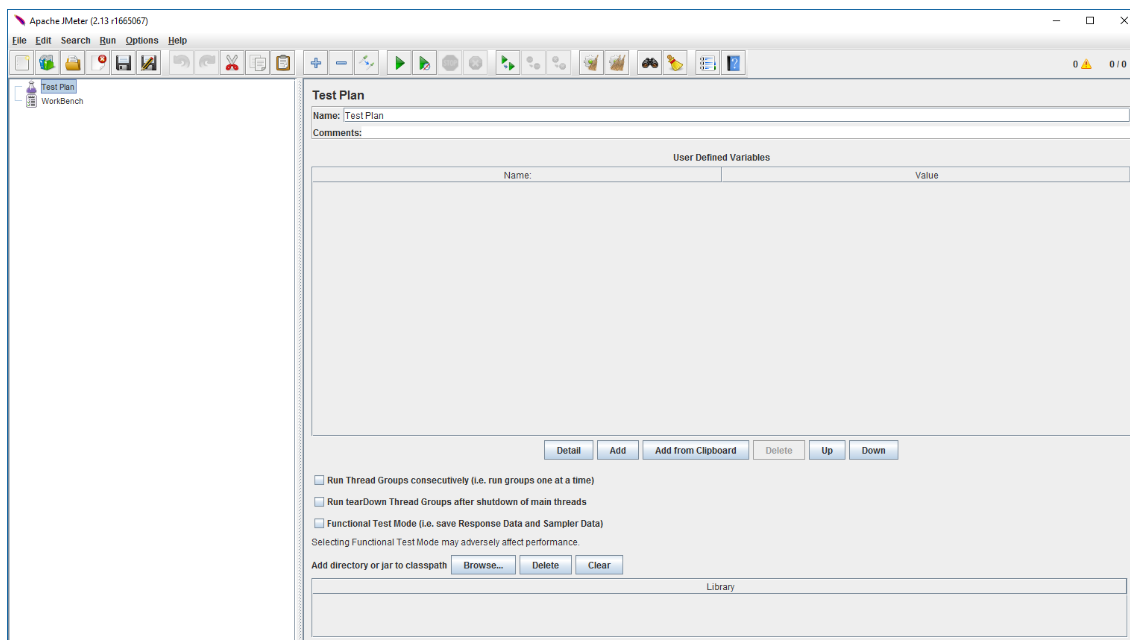
Druhou hlavní částí okna, kterou můžete nalézt v této aplikaci je zobrazení stromové struktury všech využitých modulů.



Obrázek 11 – Hlavní stromová struktura (Zdroj: vlastní)

Po prvním otevření nástroje obsahuje tato struktura pouze dvě komponenty. Test Plan, který slouží k nadefinování automatizovaného testu a WorkBench, který se používá při nahrávání určitého postupu.

Poslední částí je prostor pro zobrazení jednotlivých modulů, které JMeter nabízí. To se mění podle toho, co se zrovna používá.

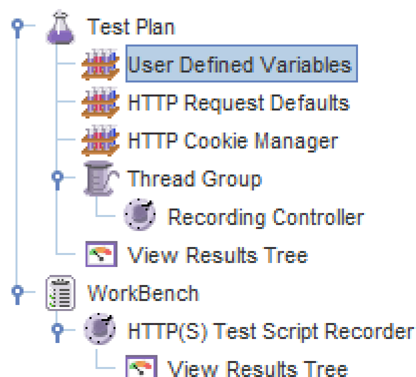


Obrázek 12 – První pohled do Apache JMeter (Zdroj: vlastní)

6.1.3 Základní nastavení

Tento nástroj, jak již bylo řečeno, slouží k nahrávání a automatizaci určitého postupu. Tato kapitola zobrazuje základní nastavení aplikace tak, aby bylo možné zaznamenávat kroky, které se uskutečňují pomocí aktivního využívání webové aplikace uživatelem.

Vytvoří se nový test. K tomuto postupu lze využít již předdefinovanou šablonu, kterou tento nástroj obsahuje. Pomocí tlačítka Templates a vybráním příslušné situace (pro tento účel – Recording) se vytvoří základní stromová struktura.



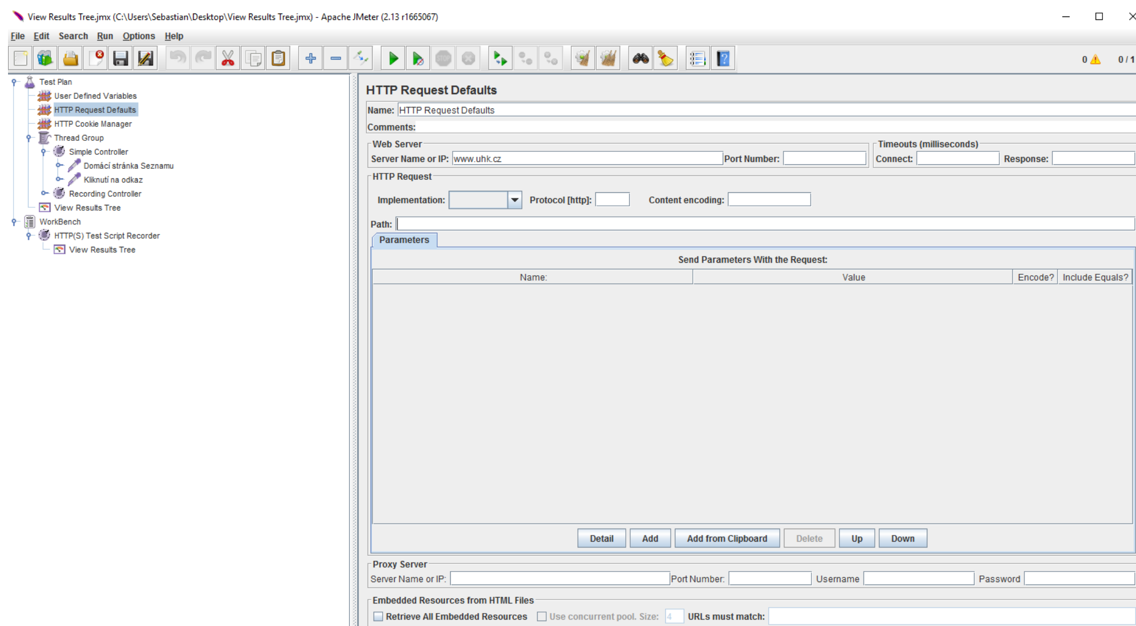
Obrázek 13 – Rozdělení ve stromové struktuře (Zdroj: vlastní)

Stromová struktura obsahuje následující prvky:

- Test Plan – slouží k prezentaci samotného testu.
    - User Defined Variables – definuje proměnné, které se mohou využívat v průběhu celého testu.
    - HTTP Request Defaults – slouží pro definici společných parametrů pro všechny requesty.
    - HTTP Cookie Manager – ukládá a odesílá cookies, stejně jako prohlížeč.
    - Thread Group – definuje uživatele.
        - Recording Controller – je nutný k zaznamenávání postupu uživatele.
- WorkBench – slouží k nahrání skriptu
    - View Results Tree – zde jsou zobrazeny všechny výsledky testu nebo nahrávání.
    - HTTP(S) Test Script Recorder – slouží k propojení internetového prohlížeče s JMeterem.

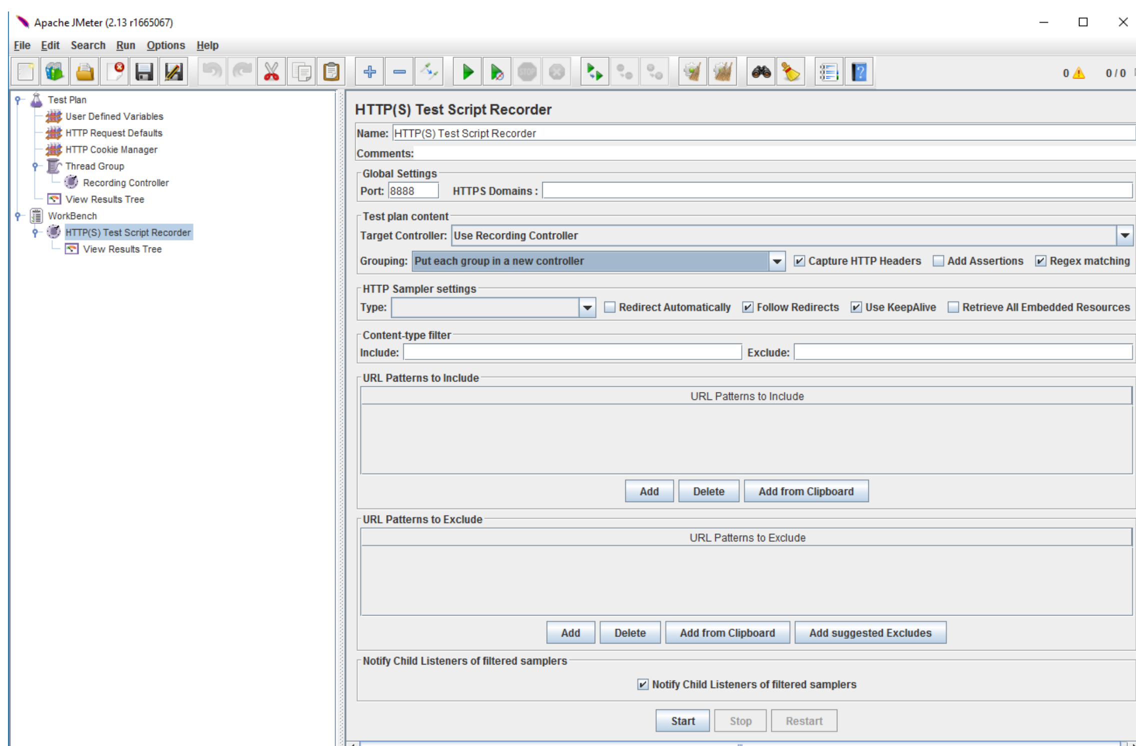
6.1.4 Nastavení jednotlivých prvků

HTTP Request Defaults – Pro spuštění testu je nutné nastavit Web Server. Do této kolonky se zadá webová adresa nebo IP adresa příslušné stránky nebo aplikace, kterou chcete testovat. Položka Path potom slouží k upřesnění cesty, ale není ji nutné vyplňovat.



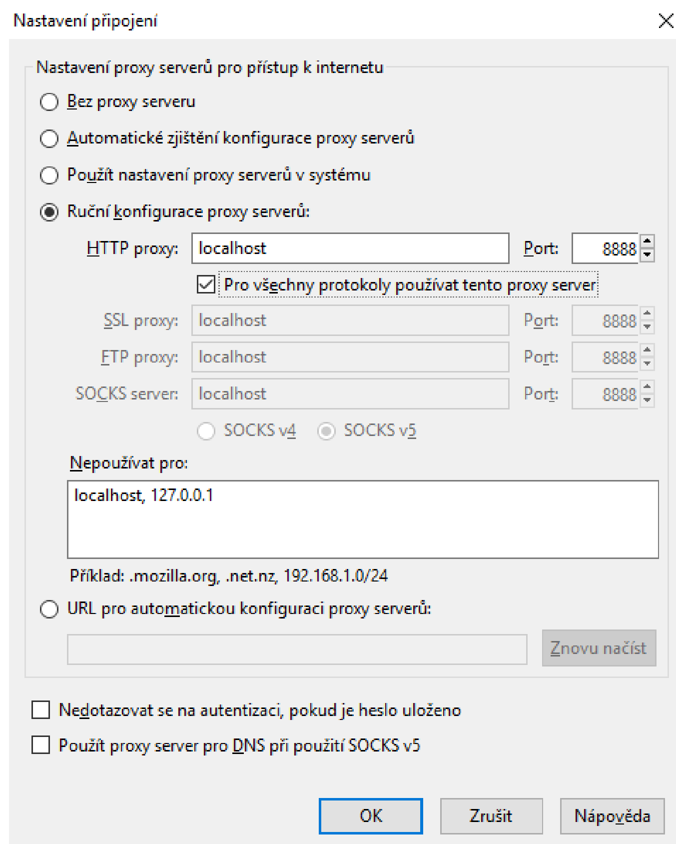
Obrázek 14 – HTTP Request Defaults (Zdroj: vlastní)

HTTP(S) Test Script Recorder – Pro nahrání testovacího skriptu musíte propojit internetový prohlížeč s JMeterem. V kolonce Grouping zvolte hodnotu Put each group in a new controller, to umožňuje alespoň základní rozdělení všech requestů. Potom nastavení Portu. Zvolte hodnotu například 8888, tato hodnota se musí shodovat s hodnotou, která se nachází i v nastavení internetového prohlížeče.



Obrázek 15 – HTTP(S) Test Script Recorder (Zdroj: vlastní)

Webový prohlížeč – Ve webovém prohlížeči v nastavení připojení zvolíte možnost ruční konfigurace proxy serverů. HTTP proxy zadáte localhost, port 8888 a zaškrtnete možnost pro všechny protokoly používat tento proxy server. Tím veškeré nastavení končí a lze začít nahrávat testovací skript.

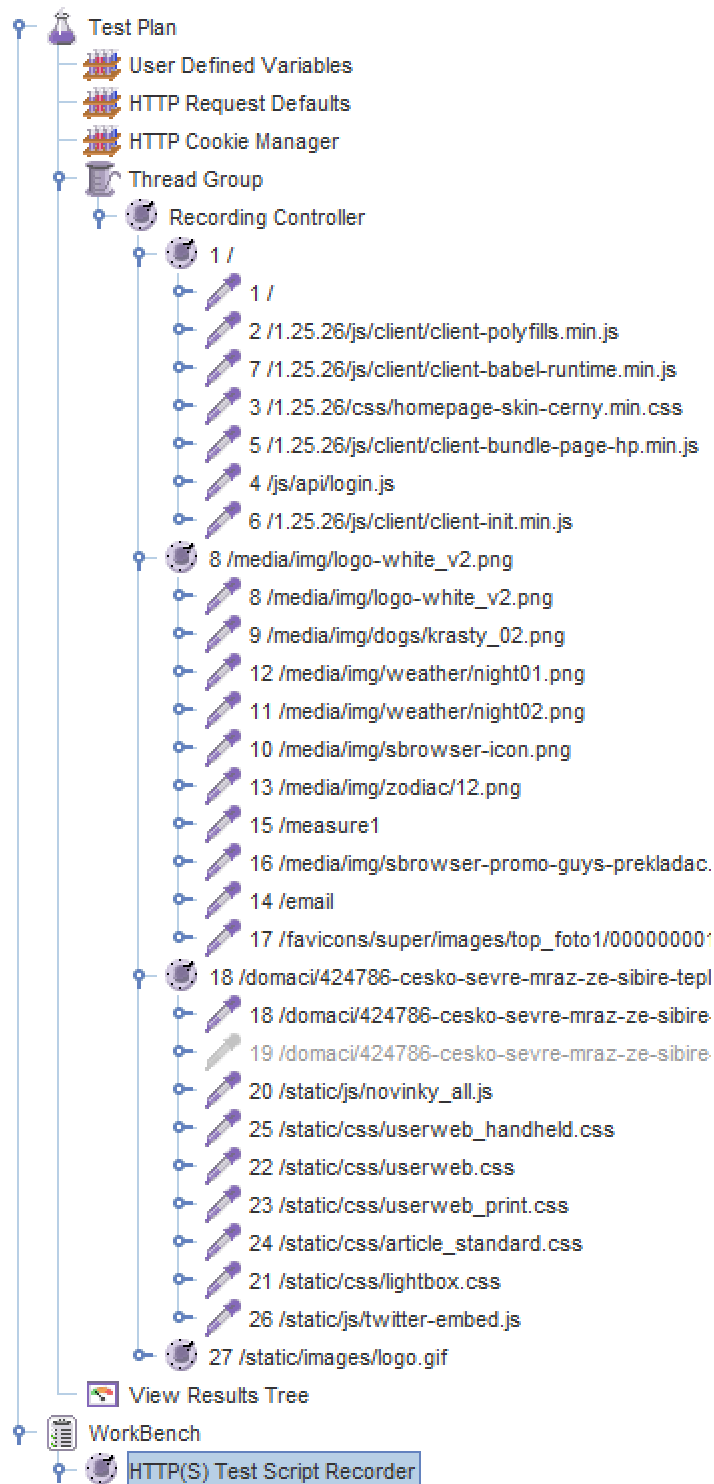


Obrázek 16 – Konfigurace webového prohlížeče (Zdroj: vlastní)

6.1.5 Nahrání testovacího skriptu

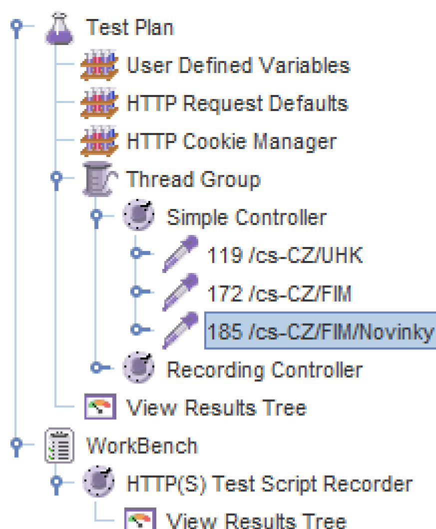
Po dokončení nastavení je vše připravené k nahrání zvoleného testovacího postupu. U HTTP(S) Test Script Recorder se zmáčkne tlačítko Start. Potom uživatel kliká postup ve webovém prohlížeči a JMeter tyto kroky automaticky nahrává do Recording Controller. Po dokončení testovacího postupu se nahrávání zastaví tlačítkem Stop. Celý postup je uložen v Controlleru.

Do Contolleru se nahrává úplně všechno. Tudíž je za potřebí výsledný testovací skript upravit do podoby, odpovídající tomu, co je očekáváno. Jak je vidět, nahrávají se nejen kroky stisknutí tlačítka, ale nahrály se například všechny obrázky, které na stránce naleznete. Proto je nutné tuto strukturu upravit a promazat.



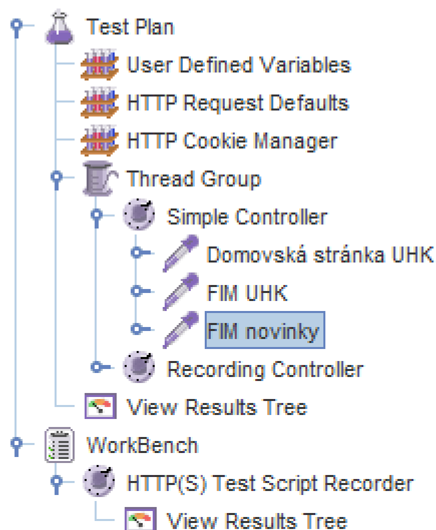
Obrázek 17 - Nahraný skript bez úpravy (Zdroj: vlastní)

Po úpravě je výsledný model velice odlišný, nicméně splňuje všechny požadavky, které jsou očekávány. Ze všech dat, která jsme měli nahraná, vypadá výsledný skript takto. Simple Controller jsme použili pouze na oddělení potřebných a nepotřebných dat.



Obrázek 18 - Skript po úpravě (Zdroj: vlastní)

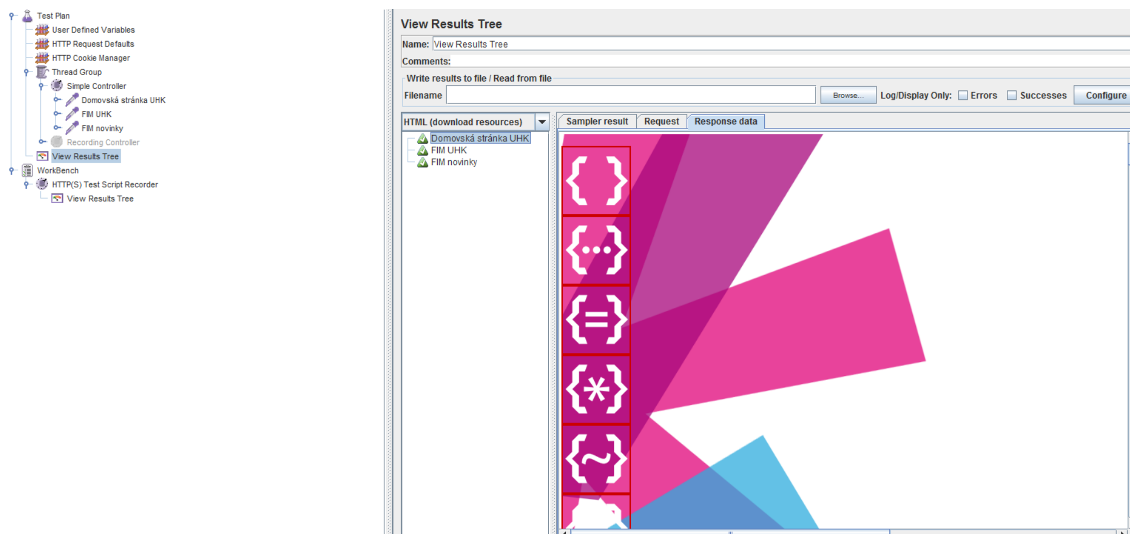
Po této úpravě si lze ještě jednotlivé kroky pojmenovat, kvůli přehlednosti.



Obrázek 19 - Skript po úpravě názvu (Zdroj: vlastní)

6.1.6 Kontrola správnosti skriptu

Ke kontrole nám slouží View Results Tree. Po spuštění testu se zde zobrazí výsledky. Pokud se zvolí varianta HTML nebo HTML (download resources), lze vidět v oddělení Response Data výsledky jednotlivých kroků. Recording Controller zakomentujeme z důvodu, že nechceme spouštět jeho vnitřní kroky.



Obrázek 20 - Zobrazené výsledky (Zdroj: vlastní)

6.2 Využití v praxi

V tomto odstavci je popsán postup při vytváření automatizovaného testu pro společnost Unicorn a.s. a v rámci iniciativy PCR (Price Coupling of Regions), která bude podrobněji popsána v následující kapitole.

6.2.1 Seznámení se systémem

Řešení propojení evropského denního trhu s elektřinou umožňuje burzám určovat ceny elektřiny jednotným způsobem a ve stejnou dobu, což představuje revoluční krok k jednotnému evropskému trhu s elektřinou.

Evropští spotřebitelé mohou efektivněji využívat evropskou elektrifikační soustavu, což se projevuje vzájemným přiblížením cen elektřiny na integrovaných trzích. Energetické burzy EPEX, GME, Nord Pool Spot, OMIE, OTE a další, spolupracující v iniciativě Price Coupling of Regions, obchodují se 75 % evropské spotřeby elektřiny (tj. více než 2000 TWh).

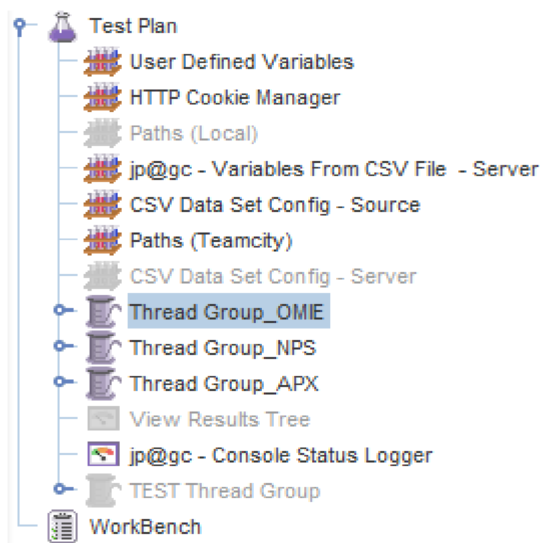
Aplikace PCR Matcher-Broker (zkráceně PCR nebo PMB) vyvinutá společností Unicorn Systems, je klíčovou součástí nového řešení PCR a sdružuje informace zúčastněných evropských energetických burz do decentralizovaného Broker Cloudu (každá burza vlastní svou instanci PMB. Ty pak mezi sebou vzájemně komunikují v rámci cloudu). Ceny a čisté přenosy jsou určovány jednotným výpočtem pomocí

algoritmu na základě evidence objednávek a dostupných přenosových kapacit z jednotlivých regionů.

Díky decentralizované architektuře nabízí PMB vysokou úroveň zabezpečení a spolehlivosti. Zároveň umožňuje burzám pracovat v autonomním módu nezávisle na ostatních burzách, pokud by to situace na denním trhu vyžadovala. Více detailnějších informací o tomto systému se nalézá v odkazech (35), (36), (37) a (38).

6.2.2 Příprava testu

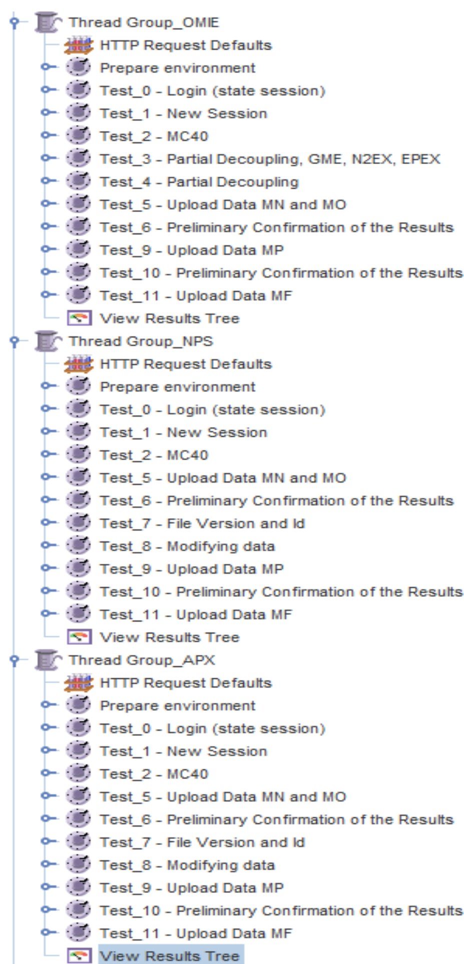
Automatizovaný test musí odpovídat přesnému postupu, který provádí tester manuálně. Při testování se využívají tři aplikace, které spolu komunikují. Stejný model musí splňovat i automatizovaný test. Tři Thread Group představují jednotlivé instance PMB aplikace (simulace tří různých burz).



Obrázek 21 – Stromová struktura na projektu PCR (Zdroj: vlastní)

6.2.3 Automatizovaný test pro PCR

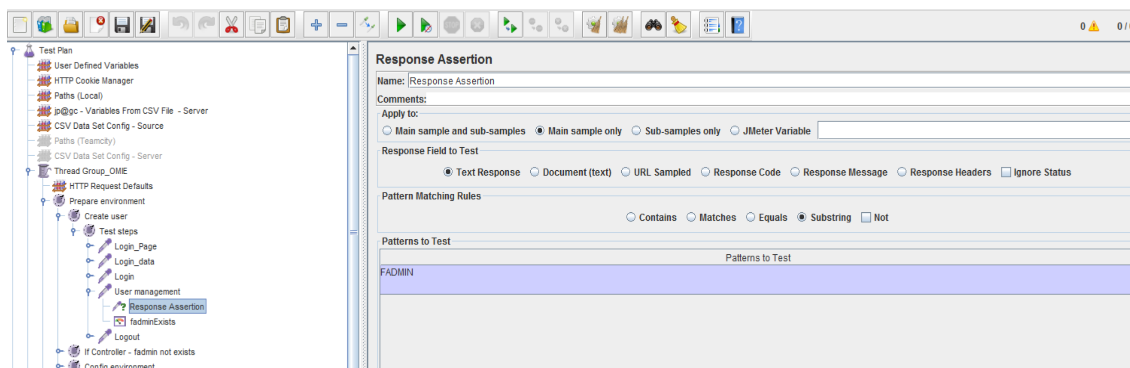
Jednotlivé grupy obsahují skoro identické kroky.



Obrázek 22 -Rozšířená stromová struktura na projektu PCR (Zdroj: vlastní)

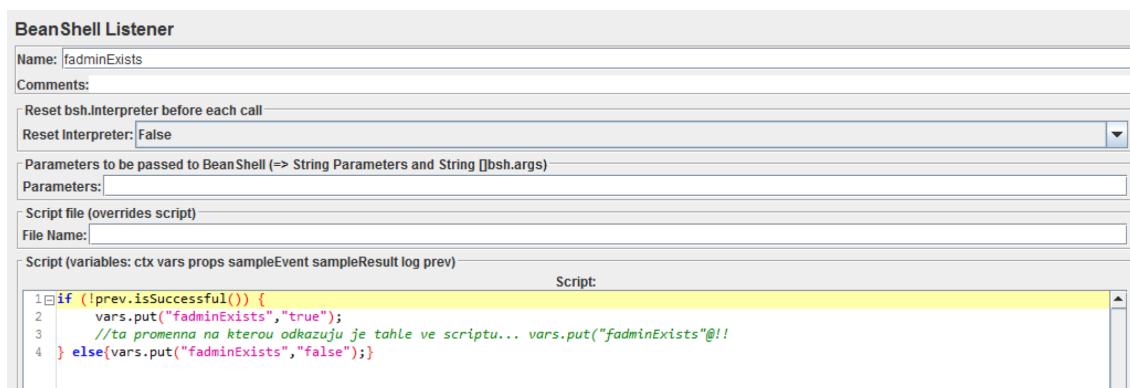
Před nahráním konfigurací, které jsou potřebné pro správné fungování aplikace, je vytvořen nový uživatel. Tím se kontroluje správné fungování tvorby uživatelů v aplikaci.

Nejprve se kontroluje, zda není uživatel už vytvořený, pokud ano, je tento krok přeskočen. K této kontrole můžeme využít Response Assertion, který dokáže na zobrazené stránce nalézt požadovaný text.



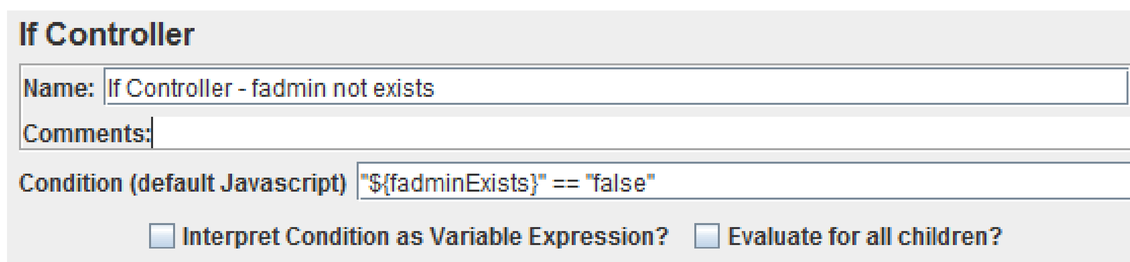
Obrázek 23 – Response Assertion (Zdroj: vlastní)

Pokud se tento text nevyskytuje na požadované stránce, znamená to, že příslušný krok je označen červeně a nabývá hodnoty „false“. K odchycení této hodnoty je používám BeanShell Listener, který naplňuje proměnnou fadminExists.



Obrázek 24 – BeanShell Listener (Zdroj: vlastní)

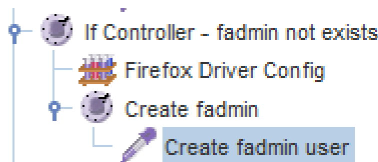
Jestliže je použita podmínka v následujícím kroku, využívá se tzv. If Controller. Pokud proměnná nabývá požadované hodnoty, je tento krok spuštěn, pokud ne, je přeskočen.



Obrázek 25 – If Controller (Zdroj: vlastní)

Pro vytvoření nového uživatele je použit jeden z pluginů, který JMeter umožňuje implementovat. Jedná se o možnost využití Selenia. V tomto automatizovaném testu se využití Selenia vyskytuje na více místech, jelikož se jedná v dané chvíli o lepší řešení. K tomu, aby bylo možné Selenium využít, musí se začlenit do kořenové

struktury Driver Config pro webový prohlížeč, který se využívá na propojení Selenia a příslušného webového prohlížeče. V tomto případě využíváme Firefox Driver Config.



Obrázek 26 – Selenium ve stromové struktuře (Zdroj: vlastní)

Druhá věc potřebná k začlenění Selenia je jp@gc – WebDriver Sampler. Právě do něho se vypisuje skript, který definuje Selenium. V kolonce Parameters se definují parametry, které se posléze využívají v kódu. Číslují se od nuly a předávají se pomocí WDS.args[číslo].

jp@gc - WebDriver Sampler

Name: Create fadmin user

Comments:${LOADBALANCER_URL_OMIE}

Help on this plugin

Parameters: ${PROTOCOL} ${LOADBALANCER_URL_OMIE} ${PMB_ADMIN} ${PMB_ADMIN_PASS} ${LOGIN_NAME} ${LOGIN_PASS} ${source}

Script Language: javascript

Script (see below for variables that are defined)

```
1  var selenium = JavaImporter(org.openqa.selenium)
2  var delay = 3000
3  var delayLogin = 9000
4  var decoupling_mode = 30000
5  var waiting = 240000
6  var wait = 90000
7
8  //Start
9  WDS.sampleResult.sampleStart()
10
11 //Waiting
12 WDS.browser.manage().timeouts().implicitlyWait(45, java.util.concurrent.TimeUnit.SECONDS);
13
14 //Main page
15 WDS.browser.get(WDS.args[0] + '://' + WDS.args[1] + '/broker/')
16
17 //Window resize
18 WDS.browser.manage().window().setSize(new selenium.Dimension(1280, 720))
19
20 //Fill in login name and password
21 var login = WDS.browser.findElement(selenium.By.id('loginForm:j_username'))
22 login.sendKeys([WDS.args[2]])
23 var pass = WDS.browser.findElement(selenium.By.id('loginForm:j_password'))
24 pass.sendKeys([WDS.args[3]])
25
26 //Click on login button
27 WDS.browser.findElement(selenium.By.xpath("//input[@value='LOGIN']")).click()
28
29 //Pause
30 var counter = java.lang.System.currentTimeMillis()
31 var counterEnd = counter + delayLogin + delay
32 while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }
33
34 //Click on Yes, if user is already logged in
35 if(WDS.browser.findElement(selenium.By.id('myModalFormFix:mdyes')).isDisplayed()){
36    WDS.browser.findElement(selenium.By.id('myModalFormFix:mdyes')).click()
37 }
38
39 //Pause
40 var counter = java.lang.System.currentTimeMillis()
41 var counterEnd = counter + delayLogin + delay
42 while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }
43
44 //Click on User Management
45 WDS.browser.executeScript("arguments[0].setAttribute('style','display: block;')",WDS.browser.findElement(selenium.By.xpath("(//span[@class='subMenu'])[2]")))
46 WDS.browser.findElement(selenium.By.xpath("//a[.='User Management']")).click()
```

Obrázek 27 – Kód Selenia 1 (Zdroj: vlastní)

Jak je vidět, parametry jsou předávány pomocí proměnných, které lze definovat na začátku každého testu pomocí již zmíněného User Defined Variables.

User Defined Variables

Name: Paths (Local)

Comments:${PROTOCOL} ${LOADBALANCER_URL_OMIE} ${LOGIN_NAME} ${LOGIN_PASS} ${SCREEN_PATH} ${LOADBALANCER_URL_JO} ${LOGIN_NAME_JO} ${LOGIN_PASS_JO} ${ABORTED_PLACE_A} ${SUBMENU}

User Defined Variables

Name:	Value	Description
SCREEN_PATH	..\\PCR_Auto_mode\\	
PROTOCOL	http	
LOADBALANCER_URL_OMIE		
LOGIN_NAME	fadmin	
LOGIN_PASS		
LOADBALANCER_URL_NPS		
LOADBALANCER_URL_APX		
NoPlayPreOMIEc	false	
NoPlayPreNPSc	false	
NoPlayPreAPXc	false	
NoPlayPreAPXa	false	
NoPlayPreOMIEa	false	
NoPlayPreNPSa	false	
Alternative1	true	
Alternative2	true	
ClickOnBroker1	false	
ClickOnBroker2	false	
BeanShell1	false	
BeanShell2	false	
fadminExists	false	
PMB_ADMIN	pmb_admin	
PMB_ADMIN_PASS		
transSCFupload	false	
DCFupload	false	
LCFupload	false	
SCFupload	false	
ENVIRONMENT	Branch	Alternative value is: Test
source	C:\Users\user\Desktop	
DATE	12.8.2016	
Shared	false	
Shared2	false	
Shared3	false	

Obrázek 28 – User Defined Variables (Zdroj: vlastní)

Následně co je uživatel vytvořen, přihlásí se do aplikace a začne nahrávat požadovanou konfiguraci. Nejdříve je zkontrolováno, zda konfigurace nejsou už nahrané, pokud ne, začnou se nahrávat. K nahrání dat je opět využito Selenium, kde pomocí relativní cesty je vloženo XML, které obsahuje potřebné konfigurace.

```
//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delayLogin
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Click on Shared Configuration
WDS.browser.executeScript("arguments[0].setAttribute('style','display: block;')",WDS.browser.findElement(selenium.By.xpath("(//span[@class='subMenu'])[2]")))
WDS.browser.findElement(selenium.By.xpath("//a[.='Shared Configuration']")).click()

//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delay
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Click on Import new configuration
WDS.browser.findElement(selenium.By.xpath("//input[@value='Import new configuration']")).click()

//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delay
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Select file:
WDS.browser.findElement(selenium.By.xpath("//input[@type='file' and @accept='*.xlsx']")).sendKeys(WDS.args[5] + "\\data\\PCR-Tests_Automation\\1.PCR_Process_Dat

//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delay
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Click on Next
WDS.browser.findElement(selenium.By.xpath("//input[@value='Next']")).click()

//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delayLogin + delayLogin
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Click on Complete
WDS.browser.findElement(selenium.By.xpath("//input[@value='Complete']")).click()
```

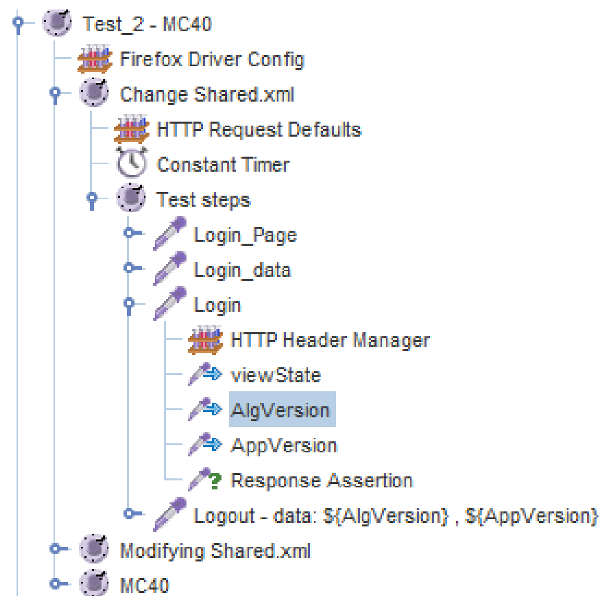
Obrázek 29 – Kód Selenia 2 (Zdroj: vlastní)

Kontrola se provádí také pomocí Selenia, kde příslušný kus kódu vytvoří obrázek aplikace v reálném čase a uloží jej do složky.

```
//Make screenshot
var screenshot = WDS.browser.getScreenshotAs(selenium.OutputType.FILE)
var file = new java.io.File(WDS.args[4] + 'OMIE_01_HomePage.png')
screenshot.renameTo(file)
```

Obrázek 30 – Detail kódu Selenia (Zdroj: vlastní)

Po nahrání těchto konfigurací je spuštěn celý proces. Jakmile je založena nová Session, začnou se v tomto případě nahrávat první data. Zde dochází k prvnímu problému. Jelikož data musí obsahovat název algoritmu, který je v aplikaci využit a musí rovněž obsahovat i verzi aplikace, je zde použita metoda, která dokáže na zobrazené stránce nalézt požadovaný text. Byl zde využit XPath Extractor.



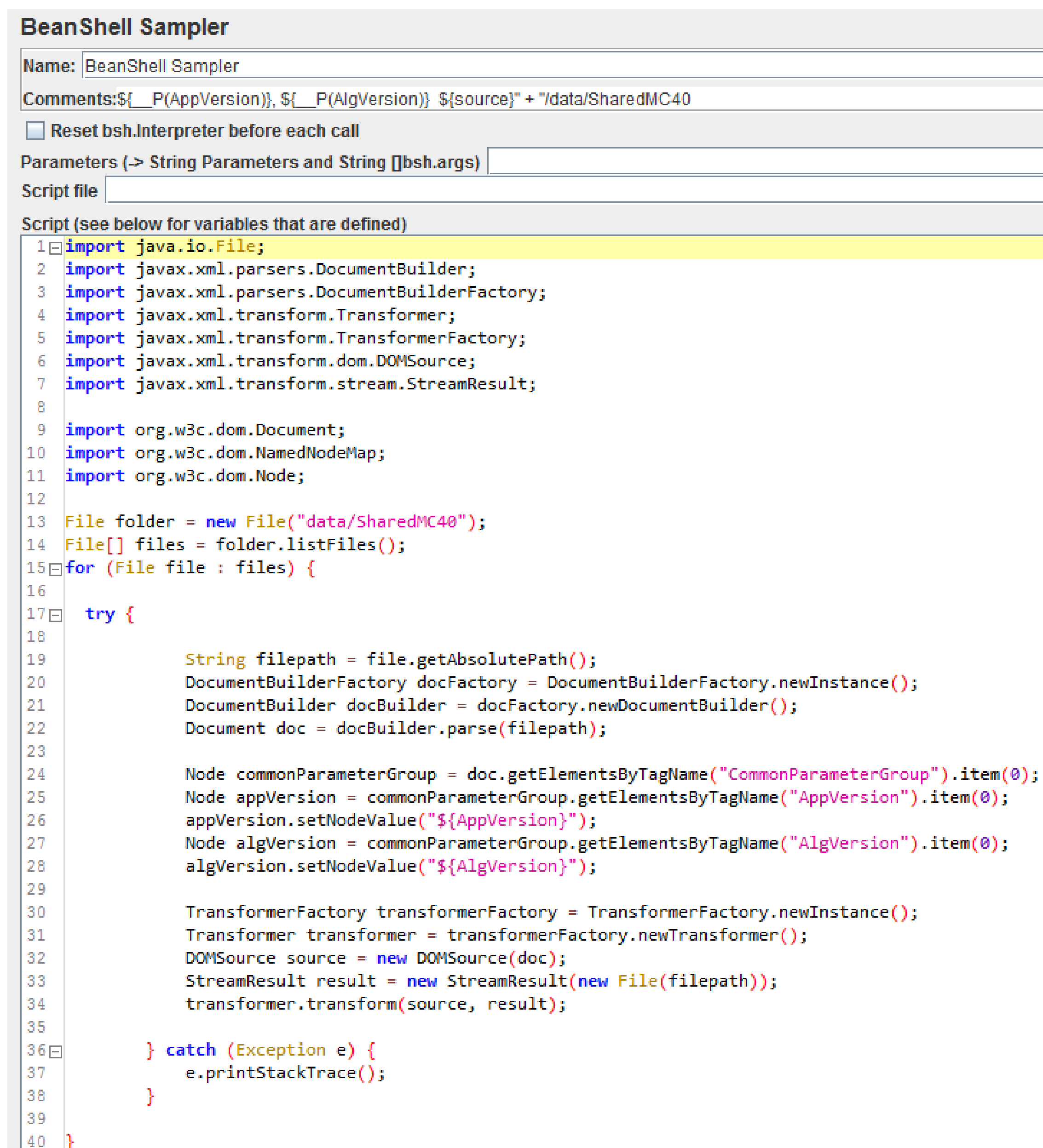
Obrázek 31 – Xpath ve stromové struktuře (Zdroj: vlastní)

Pomocí něho se zjistí v okamžiku zobrazení stránky login, příslušný algoritmus a verze aplikace. Využívá tzv. XPath query, které dokáží objevit požadovaný text, aniž by se vědělo, jak vypadá. Tento text hledá v příslušném HTML kódu aplikace.

XPath Extractor
Name: AlgVersion
Comments:
Apply to:
Main sample and sub-samples | Main sample only | Sub-samples only | JMeter Variable
XML Parsing Options
Use Tidy (tolerant parser) | Quiet | Report errors | Show warnings
Use Namespaces | Validate XML | Ignore Whitespace | Fetch external DTDs
Return entire XPath fragment instead of text content?
Reference Name: AlgVersion
XPath query: (//td[contains(.,'Euphemia ')])[2]
Default Value: Error_AlgVersion

Obrázek 32 – Xpath Extractor (Zdroj: vlastní)

Po nalezení proměnných, jsou tyto hodnoty zapsány do příslušných XML. K tomuto scriptu, který nám zobrazí XML, nalezne hodnotu, kterou má přepsat. K přepsání a následnému uložení využíváme opět BeanShell Sampler.



Obrázek 33 – BeanShell Sampler (Zdroj: vlastní)

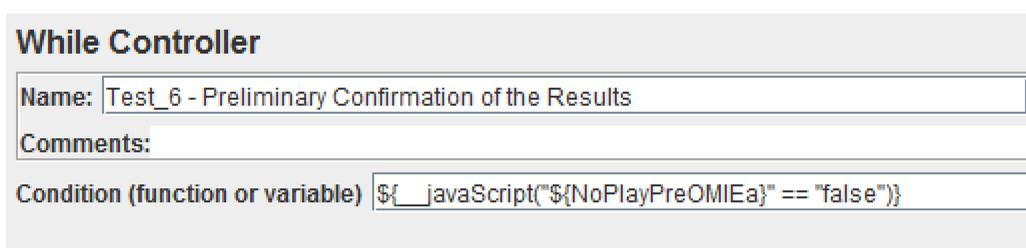
Po dokončení úpravy těchto dat dochází k jejich nahrání. K tomuto postupu je opět využito Selenium. Pomocí něj je vybrán požadovaný typ dokumentu, určená oblast a cesta, kde je tento dokument uložen. Poté je tento soubor nahrán a následná kontrola se provádí pomocí kontrolní hlášky, která se objeví po nahrání dat do

aplikace. Ke kontrole této hlášky se, v tomto případě, využívá již zmiňovaný Response Assertion.

```
//Select Data Type:
WDS.browser.findElement(selenium.By.id(WDS.args[6] +":manualUploadForm:typeInput")).click()
WDS.browser.findElement(selenium.By.xpath("//div[.='MC40 Shared configuration']")).click()
WDS.browser.findElement(selenium.By.id(WDS.args[6] +":manualUploadForm:typeInput")).sendKeys(selenium.Keys.ENTER)

//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delay
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Select Sender:
WDS.browser.findElement(selenium.By.id(WDS.args[6] +":manualUploadForm:sendersInput")).click()
WDS.browser.findElement(selenium.By.xpath("//div[.='BROKER - EPEX']")).click()
WDS.browser.findElement(selenium.By.id(WDS.args[6] +":manualUploadForm:sendersInput")).sendKeys(selenium.Keys.ENTER)

//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delay
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Select Select file:
WDS.browser.findElement(selenium.By.xpath("//input[@type='file' and @accept='*.xml']")).sendKeys(WDS.args[7] +"\\data\\SharedMC40\\Shared.xml")
//WDS.browser.findElement(selenium.By.id("j_idt63:manualUploadForm:fileupload")).sendKeys("C:\\Users\\user\\Desktop\\Práce\\PCR\\PCR - Tests Auto

//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delay
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Click on Upload file
WDS.browser.findElement(selenium.By.xpath("//input[@value='Upload file']")).click()

//Pause
var counter = java.lang.System.currentTimeMillis()
var counterEnd = counter + delay
while (counter < counterEnd ) { counter = java.lang.System.currentTimeMillis() }

//Make screenshot
var screenshot = WDS.browser.getScreenshotAs(selenium.OutputType.FILE)
var file = new java.io.File(WDS.args[4] + 'OMIE_09_BROKER - EPEX.png')
screenshot.renameTo(file)

//Click on Close
WDS.browser.findElement(selenium.By.xpath("//input[@value='Close']")).click()
```

Obrázek 34 – Nahrání dat pomocí Selenia (Zdroj: vlastní)

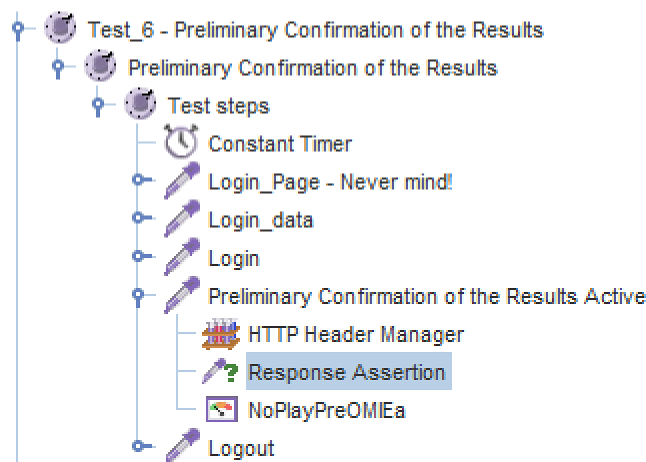
Po nahrání všech dat a po úpravách konfigurace v aplikaci je spuštěna kalkulace. Výpočty probíhají určitou dobu. Nikdy však neprobíhají stejnou dobu a může se stát, že výpočty ani nedoběhnou kvůli nějaké chybě. Další krok lze uskutečnit pouze až po doběhnutí kalkulace, zde je využit další způsob kontrolování výsledku. While Controler nám slouží k činnosti, kterou chceme opakovat.



Obrázek 35 – While Controller (Zdroj: vlastní)

Pokud nadefinuje proměnnou (v tomto případě NoPlayPreOMIEa) na začátku v User Defined Variables na false, tak se nám vždy tento cyklus spustí. Neboť uvnitř tohoto cyklu máme požadavek na kontrolu stavu určité činnosti. Jestliže nalezený text na konkrétní stránce se neshoduje s textem, který hledáme, zůstane proměnná stále false. Až nastane situace, že se text na stránce našel, změní se hodnota

proměnné z false na true a vyskočí z tohoto cyklu. V rámci cyklu je využit tzv. Constant Timer, který nám určuje rychlost vykonávání jednotlivých kroků.



Obrázek 36 – Podmínka ve stromové struktuře (Zdroj: vlastní)

Po dokončení všech zbývajících kroků je kontrolováno, zda Session je ve stavu Completed.

6.2.4 Doplňující informace

V rámci začlenění testu do systému, který slouží k manipulaci s podobnými nástroji, je využito ještě několik metod, které usnadňují práci s automatizovaným testem.

V tomto testu je použit CSV Data Set Config, který slouží k nalezení požadovaného souboru na místě určení. Přečte si informaci z tohoto souboru a uloží si ji do proměnné, kterou následně může využívat JMeter. S tím je spojeno jeho rozšíření jp@gc – Variables Drom CSV File. To slouží k podobné věci, ale jeho výhodou je, že si lze přímo otestovat, co tento soubor vrací za hodnoty a do jakých proměnných jsou tyto hodnoty vloženy.

Lze využívat hodnoty, které jsou do JMeteru zaslány z jiných aplikací, a to přes proměnné, které mají jinou syntaktickou podobu. JMeter potom s nimi pracuje stejně jako s normálními proměnnými. Místo použití normální proměnné ${název}, se využívá tato varianta ${__P(název)}.

7 Výsledky a závěr

Automatizované testování je v dnešní době nezbytnou součástí vývoje každého softwaru. Vytvářejí se čím dál složitější aplikace, zvyšuje se jejich propojenost a nároky na jejich kvalitu – tím se výrazně zvyšují i nároky na správné a rychlé otestování. Automatizované testování tak poskytuje ideální kombinaci rychlého a přesného testování. Počáteční investice do vývoje automatizovaného testu se v průběhu vývoje a vlastně celého životního cyklu software mnohonásobně vrátí.

Výsledkem této práce je automatizovaný test, který je pravidelně využíván v servisním prostředí při nasazení nové verze spravované aplikace. Tento test simuluje činnost testera a probíhá pouze pro určený scénář dané aplikace. Výhodou tohoto testu je autonomní chování, což zaručuje podstatnou úsporu času pracovníka i úsporu financí firmy. Po zkušebním provozu je zařazen do pravidelného postupu při testování servisního projektu PCR ve firmě Unicorn Systems a.s.

Systém PCR se nadále rozvíjí pravidelnými servisními verzemi a přibližně jednou za rok novou major verzí, obsahující důležité businessové změny. Spolu s rozvojem systému se udržují a rozšiřují automatizované testy. Cílem společnosti je nadále zvyšovat procento pokrytí aplikace těmito testy, což výrazně usnadňuje, zpřesňuje a zefektivňuje celý proces testování. Postupně tak dochází ke zvyšování kvality dodávaného softwaru.

8 Citovaná literatura

1. Unicorn Systems a.s. Unicorn Systems - Příručka uživatele. Praha : Unicorn, 2015.

2. Použití nástrojů v testování. *SW Testování.* [Online] 18. 03 2011. [Citace: 08. 07 2016.] http://www.swtestovani.cz/index.php/testovaci-nastroje/35-pouiti-nastroj-v-testovani.

3. Testovací dokumentace - plán, scénář, případ. *SW Testování.* [Online] 07. 02 2011. [Citace: 08. 07 2016.] http://www.swtestovani.cz/index.php?option=com_content&view=article&id=15:testovaci-dokumentace-plan-scena-pipad&catid=3:zaklady&Itemid=11.

4. SoapUI prakticky III. - Test Suite, Test Case. *SW Testování.* [Online] 08. 10 2011. [Citace: 08. 07 2016.] http://www.swtestovani.cz/index.php/testovaci-nastroje/56-soap-ui-prakticky-iii-test-suit-test-case-test-step.

5. PECKA, Miroslav. REGULÁRNÍ VÝRAZY. [Online] 2005-2016. [Citace: 14. 07 2016.] http://www.regularnivyrazy.info/.

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

7. Druhy testování v procesu vývoje SW. *SW Testování.* [Online] 10. 02 2011. [Citace: 19. 07 2016.] http://www.swtestovani.cz/index.php/uvod-do-testovani/18-druhy-testovani-v-procesu-vyvoje-sw.

8. Vývojové modely. *Diagnostika a testování elektronických systémů.* [Online] [Citace: 10. 01 2017.] http://www.umel.feec.vutbr.cz/bdts/index.php/embedded-systemy/vyvojove-modely.

9. ČERMÁK, Miroslav. Způsoby testování. *Clever and Smart.* [Online] 25. 12 2008. [Citace: 03. 11 2016.]

10. —. White box test. *Clever and Smart.* [Online] © 2008 - 2016, Miroslav Čermák, 23. 12 2008. [Citace: 25. 07 2016.] http://www.cleverandsmart.cz/white-box-test/.

11. —. Grey box test. *Clever and Smart.* [Online] © 2008 - 2016, Miroslav Čermák, 25. 12 2008. [Citace: 09. 08 2016.] http://www.cleverandsmart.cz/grey-box-test/.

12. —. Black box test. *Smart and Clever.* [Online] © 2008 - 2016, Miroslav Čermák, 25. 12 2008. [Citace: 09. 08 2016.] http://www.cleverandsmart.cz/black-box-test/.

13. Automatizované testování. *Testování softwaru.* [Online] [Citace: 10. 09 2016.] http://testovanisoftwaru.cz/automatizovane-testovani/.
14. Firebug. *Firebug.* [Online] Mozilla, 2005-2010. [Citace: 02. 12 2016.] http://getfirebug.com/.
15. Firebug. *What is Firebug?* [Online] Mozilla, 2005-2010. [Citace: 01. 12 2016.] http://getfirebug.com/whatisfirebug.
16. Firebug. *Wikipedie.* [Online] 26. 08 2014. [Citace: 10. 09 2016.] https://cs.wikipedia.org/wiki/Firebug.
17. HLAVA, Tomáš. Firebug. *Testování softwaru.* [Online] 07. 03 2012. [Citace: 09. 10 2016.] http://testovanisoftwaru.cz/nastroje/firebug/.
18. Předplatné Visual Studio Test Professional. *Microsoft store.* [Online] Microsoft, 2017. [Citace: 06. 01 2017.] https://www.microsoftstore.com/store/mseea/cs_CZ/pdp/productID.509437430 0.
19. Visual Studio Test Professional. *Visual Studio.* [Online] Microsoft, 2017. [Citace: 05. 01 2017.] https://www.visualstudio.com/cs/vs/test-professional/.
20. Testování aplikace pomocí nástroje Microsoft Test Manager. *MSDN Microsoft.* [Online] Microsoft, 04 2016. [Citace: 01. 12 2016.] https://msdn.microsoft.com/cs-cz/library/jj635157.aspx.
21. Release Management. *Visual Studio.* [Online] Microsoft, 2017. [Citace: 06. 01 2017.] https://www.visualstudio.com/cs/team-services/release-management/.
22. Automated Software Testing. *SmartBear.* [Online] SmartBear Software, 2017. [Citace: 10. 01 2017.] https://smartbear.com/product/testcomplete/overview/.
23. Desktop Application Testing. *SmartBear.* [Online] SmartBear Software, 2017. [Citace: 04. 01 2017.] https://smartbear.com/product/testcomplete/desktop-module/overview/.
24. Test Complete. *Wikipedia.* [Online] 22. 01 2017. [Citace: 04. 01 2017.] https://en.wikipedia.org/wiki/TestComplete.
25. Automated Web Application. *SmartBear.* [Online] SmartBear, 2017. [Citace: 10. 01 2017.] https://smartbear.com/product/testcomplete/web-module/overview/.

26. Mobile Application Test Automation. *SmartBear.* [Online] SmartBear Software, 2017. [Citace: 04. 01 2017.] https://smartbear.com/product/testcomplete/mobile-module/overview/.
27. Apache JMeter™. *The Apache Software Foundation.* [Online] Apache Software Foundation , 1999 – 2016 . [Citace: 11. 01 2017.] http://jmeter.apache.org/.
28. Selenium IDE. *SeleniumHQ.* [Online] [Citace: 10. 01 2017.] http://www.seleniumhq.org/projects/ide/.
29. Selenium 1 (Selenium RC). *SeleniumHQ.* [Online] Selenium Project, 10. 03 2017. [Citace: 10. 03 2017.] http://www.seleniumhq.org/docs/05_selenium_rc.jsp.
30. MantisBT makes collaboration with team members & clients easy, fast, and professional. *Mantis Bug Tracker.* [Online] MantisBT Team. [Citace: 10. 01 2017.] https://www.mantisbt.org/.
31. HLAVA, Tomáš. Mantis Bug Tracker. *Testování softwaru.* [Online] 07. 03 2012. [Citace: 10. 01 2017.] http://testovanisoftwaru.cz/nastroje/mantis-bug-tracker/.
32. JIRA Software. *Atlassian.* [Online] Atlassian . [Citace: 10. 01 2017.] https://www.atlassian.com/software/jira.
33. HLAVA, Tomáš. TestLink – správa a organizace testování softwaru. *Testování softwaru.* [Online] 13. 08 2012. [Citace: 10. 01 2017.] http://testovanisoftwaru.cz/nastroje/testlink/.
34. TestLink. *Wikipedie.* [Online] 16. 01 2017. [Citace: 10. 03 2017.] https://cs.wikipedia.org/wiki/TestLink.
35. Unicorn Systems Participates in a New Solution to Interconnect. *Unicorn.* [Online] Unicorn Systems a.s. , 28. 04 2014. [Citace: 10. 03 2017.] http://test.unicornsystems.eu/en/news/press-release/unicorn-systems-participates-in-a-new-solution-to-interconnect-european-day-ahead-electricity-markets.html.
36. PCR - Price Coupling of Regions . *OTE.* [Online] OTE, 2010. [Citace: 10. 03 2017.] http://www.ote-cr.cz/kratkodobe-trhy/integrace-trhu/pcr-price-coupling/pcr-price-coupling-of-regions.
37. Market Coupling PCR: Price Coupling of Regions. *EPEX SPOT.* [Online] EPEX SPOT, 2017. [Citace: 10. 03 2017.] https://www.epexspot.com/en/market-coupling/pcr.

38. Unicorn Systems pomáhá propojit evropský denní trh s elektřinou. *Hospodářské noviny.* [Online] Economia, a.s., 20. 05 2014. [Citace: 20. 03 2017.] http://archiv.ihned.cz/c1-62212100-unicorn-systems-pomaha-propojit-evropsky-denni-trh-s-elektrinou. ISSN 1213-7693.

39. 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. ISBN 978-80-247-5594-6.

40. STEPHENS, Matt a Doug ROSENBERG. *Testování softwaru řízené návrhem.* Brno : Computer Press, 2011. ISBN 978-80-251-3607-2.

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

42. —. *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. ISBN 80-7226-636-5.

43. Testovací nástroje a služby. *Visual Studio.* [Online] Microsoft, 2017. [Citace: 05. 01 2017.] https://www.visualstudio.com/cs/team-services/testing-tools/.

44. ZAPLETAL, Lukáš. Regulární výrazy v příkladech. *ROOT.CZ.* [Online] 28. 01 2003. [Citace: 11. 10 2016.] https://www.root.cz/clanky/regularni-vyrazy-v-prikladech/.

45. Nástroje. *Testování softwaru.* [Online] [Citace: 10. 09 2016.] http://testovanisoftwaru.cz/nastroje/.

46. List of unit testing frameworks. *Wikipedia.* [Online] 08. 03 2017. [Citace: 10. 01 2017.] https://en.wikipedia.org/wiki/List_of_unit_testing_frameworks.

47. BSD licence. *Wikipedie.* [Online] 01. 03 2015. [Citace: 15. 11 2016.] https://cs.wikipedia.org/wiki/BSD_licence.

9 Přílohy

1) Posudek od divizního ředitele servisní divize v Hradci Králové.

Sebastian Vacek se v průběhu praxe dokázal seznámit s několika testovanými systémy. Ve spolupráci s analytiky a test architekty navrhl sadu testovacích scénářů, které postupně automatizoval za pomocí nástrojů JMeter a Selenium. Díky automatizaci testů došlo k výrazné úspoře času při regresních testech a průběžném testování vyvíjených verzí systémů PCR Matcher-Broker a ENTSO-E Transparency Platform.

Automatizaci testů prováděl iterativně, takže bylo možné postupně spouštět dílčí testy a již v průběhu vývoje odhalit chyby vzniklé při implementaci nových funkcionalit. Velký důraz byl kladen na stabilitu testů a opakovanou použitelnost. Stabilitou se myslí rezistence testů vůči změnám v aplikaci a hlavně GUI. V této oblasti využil Sebastian zkušenosti z předchozích implementací testů a navrhl strukturu testovacích bloků tak, aby je nebylo nutné v budoucnu měnit. U specifických nastavení aplikace zavedl parametrizaci testů a tím docílil přenositelnost testů mezi verzemi i různými prostředími (vývojové, testovací).

Nedílnou součástí jeho práce bylo začlenění automatizovaných testů do standardního vývojového cyklu. Automatizované testy uložil do verzovacího systému GIT a nakonfiguroval jejich spouštění automaticky nebo na vyžádání z TeamCity (Build management and continuous integration server). To umožnilo jednoduché spuštění testů a transparentní sdílení výsledku testů. Vývojáři díky tomu dostali možnost snadno provést průběžnou integraci svých změn a během několika minut je komplexně otestovat na pozadí bez fyzické přítomnosti a součinnosti testera. Celý projektový tým tímto dostává okamžitou zpětnou vazbu o průběžném stavu a kvalitě vyvíjené verze.

Ing. Karel Schejbal

Oskenované zadání práce

Univerzita Hradec Králové
Fakulta informatiky a managementu
Akademický rok: 2016/2017

Studijní program: Aplikovaná informatika
Forma: Prezenční
Obor/komb.: Aplikovaná informatika (ai3-p)

Podklad pro zadání BAKALÁŘSKÉ práce studenta

PŘEDKLÁDÁ:	ADRESA	OSOBNÍ ČÍSLO
Vacek Sebastian	Skuhrov nad Bělou 148, Skuhrov nad Bělou	I14157

TÉMA ČESKY:

Automatizované testování SW a aplikace vybraných nástrojů v praxi

TÉMA ANGLICKY:

VEDOUCÍ PRÁCE:

doc. Mgr. Tomáš Kozel, Ph.D. - KIKM

ZÁSADY PRO VYPRACOVÁNÍ:

Cíl: Návrh a využití automatizovaného testování ve vývojovém cyklu informačního systému. Jeho praktické použití pro různé typy testů a implementace pomocí vybraných nástrojů.

Osnova:
1. Úvod
2. Životní cyklus softwarového projektu
3. Testování softwaru
4. Automatizované testování
5. Nástroje pro testování
6. Vlastní příklad automatizovaného testu
7. Výsledky a závěr

SEZNAM DOPORUČENÉ LITERATURY:

-

Podpis studenta: Sebastian Vacek .... **Datum:** 13.10.16

Podpis vedoucího práce: .... **Datum:** 14.10.16

(c) IS/STAG , Portál - Podklad kvalifikační práce , I14157 , 12.10.2016 12:41