Masarykova Univerzita
Fakulta informatiky

Funkční testování aplikace pro výuku zeměpisu

Bakalářská práce

Erik Sýkora

Brno, Podzim 2015

Místo tohoto listu vložte kopie oficiálního podepsaného zadání práce a prohlášení autora školního díla.

Prohlášení

Prohlašuji, že tato bakalářská práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Erik Sýkora

Vedoucí práce: Mgr. Jan Papoušek

Poděkování

Rád bych zde poděkoval zejména svému vedoucímu Mgr. Janu Papouškovi za cenné rady a odbornou pomoc při tvorbě této práce. Dále chci také poděkovat své rodině za jejich podporu a pochopení během tvorby této práce.

Shrnutí

This is the abstract of my thesis, which can span multiple paragraphs.

Klíčová slova

funkční testování, webové aplikace, Java, Arquillian Graphene, automatizace testování…

Obsah

Seznam tabulek

Seznam obrázků

1 Úvod

Vývoj softwaru je komplexní proces sestávající z mnoha kroků. Tento proces se liší dle použitého vývojového modelu, nicméně vždy by měl zahrnovat určitou formu testování. Náklady na opravu chyby totiž strmě rostou při jejím pozdním nalezení, v případě kritického softwaru může neošetřená chyba dokonce způsobit ztráty na životech.

Webové aplikace jsou dnes velice rozšířeným druhem softwaru. Testování webové aplikace zahrnuje spoustu oblastí, jmenovitě například funkční testování, testování kompatibility či validace a verifikace. Tyto způsoby testování se běžně používají pro testování softwaru obecně, nicméně testování webových aplikací má svá specifika. Metodami testování použitými pro testování konkrétní webové aplikace se bude zabývat druhá kapitola.

Představené postupy budou následně aplikovány při testování aplikace pro výuku zeměpisu – Slepé mapy[12]. Jelikož tato aplikace postrádá specifikaci softwaru, bude důležitou částí práce analýza aplikace, čemuž bude věnována třetí kapitola. Tato práce je zaměřena na funkční testování, proto zde bude také popsáno chování a funkčnost aplikace, jež bude následně otestována.

Pro tvorbu samotných testů je možno vybrat ze široké škály nástrojů. Pro testování aplikace Slepé mapy byla vybrána testovací platforma Arquillian a rozšiřující plugin Graphene. Tato platforma si zaslouží podrobnější popis, proto se čtvrtá kapitola bude zabývat představením tohoto a dalších použitých nástrojů. Na získaných znalostech o použitých technologiích bude stavět pátá kapitola, ve které bude vysvětlen postup při tvorbě testů. Vysvětlen bude kompletní postup použitý při testování webové aplikace Slepé mapy, a to od počáteční tvorby testového plánu, rozdělení aplikace na testovatelné části, návrh jednotlivých testových případů a následně jejich realizace pomocí výše zmíněných technologií.

Hotové testy budou spouštěny automaticky při aktualizování aplikace. Jelikož by bylo nepraktické po každé aktualizaci testy spouštěn manuálně, bude součástí práce automatizace testové sady. Způsob realizace automatického testování bude popsán v šesté kapitole.

Závěrem budou vyhodnoceny výsledky testování. Součástí výstupů této práce bude také zhodnocení současného stavu aplikace na

základě pokrytí testovými sadami. Vytvořená dokumentace společně s testovou sadou je dostupná v příloze.

2 Testování webové aplikace a použité přístupy

Testování softwaru je komplexní proces, jehož cílem je v programu či aplikaci nalézt softwarové chyby a ověřit, zda splňuje požadavky na něj kladené. Chybu lze obecně definovat jako neočekávané chování programu, lišící se od specifikace softwaru [4]. Za software obsahující chyby lze dle [2] považovat i obtížně srozumitelný, pomalý, či bude dle názoru testera z pohledu koncového uživatele jakkoliv nesprávný. Při testování softwaru je nutno mít na paměti, že je prakticky nemožné v rozumném čase kompletně otestovat byť jednoduchý program. I menší programy mají příliš mnoho možných vstupů, výstupů a možných cest skrze software, proto je třeba při testování zůžit tuto množinu na zvládnutelnou podmnožinu. Toho je v případě této práce docíleno vytyčením klíčových či rizikových částí aplikace a aplikováním vhodných testovacích metod.

Výběr vhodné strategie je klíčový pro optimální výsledek testování, proto v této kapitole bude představen výběr některých běžně používané přístupy pro testování webové aplikace. Většina zde uvedených postupů a metod byla využita v této práci při testování aplikace na procvičování zeměpisu – Slepé mapy. Je ovšem nutno zmínit, že seznam zde zmíněných principů testování webových aplikací není vyčerpávající. Je možné podrobit aplikaci důkladnějšímu testování pomocí dalších, zde nezmíněných metod, účelem této práce je ale provedení funkčního testování. Toto zúžení oblasti zájmu pouze na funkční testování umožňuje otestovat klíčové vlastnosti softwaru a podat tak informaci o funkčnosti – jednom z důležitých kritérií kvality softwaru. Nevýhodou je pak nemožnost otestovat některé ostatní kvality softwaru, jako například jeho spolehlivost, kompatibilitu či bezpečnost.

2.1 Typy testování

2.1.1 Funkční testování

Jelikož je stěžejním bodem této práce vypracování funkčních testů, je vhodné si uvést cíle tohoto typu testování. Cílem je otestování správ-

ného chování funkcí části aplikace či aplikace jako celku. Chování softwaru může být popsáno v jeho specifikaci. Pokud není specifikace k dispozici, tak je testováno předpokládané chování aplikace. [3]. Během funkčního testování tester nahlíží na aplikaci očima uživatele a jeho úkolem je ověřit, zda aplikace funguje správně. Ve webové aplikaci mohou být testovány například přechody mezi jednotlivými stránkami aplikace, ošetření zápisu neplatných hodnot do formulářů nebo ošetření chybových stavů aplikace [2].

Při funkčním testování je nutné nejprve identifikovat funkce testovaného softwaru a poté pro nalezené funkce vytvořit testy. V této práci nelze k identifikaci funkcí využít specifikaci softwaru, proto se třetí kapitola bude věnovat analýze webové aplikace a následného výběru funkcí vhodných k testování. Samotné tvorbě testů bude věnována pátá kapitola.

2.1.2 Jednotkové testování

Tento typ testování využívá malých, testovatelných částí aplikace. Touto takzvanou jednotkou mohou být například funkce, třídy, procedury či rozhraní. Cílem jednotkového testování je izolovat části programu a otestovat, zda splňují svou funkci a je možné je použít v celku, jehož jsou součástí. Výhodou tohoto přístupu je jednodušší objevení případných chyb v aplikaci. Jednotkové testování navíc bývá prováděno v ranných fázích vývoje softwaru, proto je také objevení chyb v programu výrazně levnější [5].

Některé programovací jazyky(Java, C#, Python, Ruby, PHP a další) mají přímou podporu jednotkového testování bez závislosti na externích knihovnách, nicméně běžně se používají aplikační rámce třetích stran, které usnadňují tvorbu jednotkových testů.

2.1.3 Statické a dynamické testování

Těmito pojmy lze rozlišit dvě základní možnosti, jakými můžeme k testování softwaru přistupovat. Statickým testováním je myšleno testování nespustitelné části programu, jde převážně o zkoumání dokumentace a specifikace softwaru. Tento přístup lze aplikovat ještě dříve, než je vytvořena spustitelná verze softwaru[2]. Hlavním cílem tohoto druhu testování je nalezení chyb v ranné fázi vývoje softwaru.

Odstranění chyb v softwaru díky včasnému odhalení nejasnosti ve specifikaci může být mnohem levnější, než opravování chyby v již hotovém kódu softwaru.

Dynamické testování naopak vyžaduje spustitelný kód, na základě jehož chování je software testován. Při testování zadáváme softwaru vstupní data, kontrolujeme reakci systému a ověřujeme korektnost výstupních dat. Dynamické testování lze také dále dělit na funkční a nefunkční testování. Funkční testování bylo v této kapitole popsáno, nefunkční testování se zabývá těmi aspekty softwaru, které přímo nesouvisí s jeho funkcionalitou. Pod nefunkčním testováním si lze představit například výkonnostní, zátěžové či bezpečnostní testování [10].

2.1.4 Regresní testování

Po opravě chyb v softwaru se může stát, že sice byla chyba opravena, ale jako vedlejší efekt se objevila chyba na jiném místě. Proces opakovaného spouštění testů při změně softwaru za účelem ověření existující funkcionality se nazývá regresní testování. Jeho cílem je zajištění, že změna kódu programu negativně neovlivnila funkčnost softwaru zavedením nových chyb [6]. Jelikož je testována funkčnost softwaru, tak lze regresní testování považovat za druh funkčního testování.

Regresní testování může probíhat manuálně, často ovšem bývá automatizováno. K testování je připravena testovací sada. Do této testovací sady je u větších projektů rozumné zahrnout pouze podmnožinu všech testů, jelikož spouštění velké testové sady vyžaduje větší množství času a zdrojů. V případě této práce jsou do testové sady zahrnuty všechny testy, což je díky jejich malému počtu rozumné.

2.2 Použité metody a postupy

2.2.1 Testování černé, šedé a bílé skříňky

Testování softwaru lze dělit pomocí pojmů testování černé, bílé a šedé skříňky. Při postupu testování černé skříňky má tester k dispozici pouze popis funkčnosti aplikace, nemůže využít zdrojového kódu softwaru. Lze tedy nesprávnou funkčnost pouze odpozorovat dle od-

lišného chování od specifikace či předpokládaného chování. Jako příklad testování černé skříňky může být uvedeno funkční testování.

Pokud je k testování využit i zdrojový kód programu nebo znalost vnitřních principů, jedná se o testování bílé skříňky. Testování založené na tomto principu bývá velmi důsledné, ale nemusí odpovídat realistickým scénářům užití. Tester využívající tuto techniku také musí mít programátorské znalosti, aby byl schopen porozumět kódu programu. Nevýhodou také je nemožnost testovat, zda nějaká funkcionalita chybí, jelikož tento přístup se zabývá testováním již napsaného kódu.

Třetí možností je testování šedé skříňky, která kombinuje předchozí dva postupy. Nejprve jsou testy navrhovány z pohledu uživatele (testování černé skříňky), následně je ale k návrhu testů využit také postupy testování bílé skříňky. Tento přístup zajistí efektivitu aplikace a dostatečné pokrytí kódu testy [1]. Postup testování šedé skříňky lze díky dostupnosti zdrojového kódu každé webové stránky využít i při testování webových aplikací.

2.2.2 Validace a verifikace

Oba tyto pojmy lze přeložit jako ověřování či kontrola, nicméně význam těchto dvou výrazů se liší a pro účely testování je tento rozdíl velmi důležitý. Verifikace je proces, jehož cílem je potvrzení, že testovaný software vyhovuje své specifikaci. Účelem procesu validace je kontrola, zda testovaný software vyhovuje požadavkům uživatele. V případě, že byla specifikace softwaru vytvořena nesprávně a nevyhovuje původním požadavkům uživatele, může dojít k situaci, kdy software sice splňuje verifikaci, ale validaci nikoliv [2].

2.2.3 Automatizace testování

Často se stává, že napsané testy bude nutné spuštět vícekrát, například v případě regresního testování. Nástroje automatizace testovacího procesu jsou vhodným řešením pro tyto situace. Mezi jejich hlavní výhody patří například rychlejší provádění testů, než je možné manuálně. Dále je testování efektivnější, jelikož při manuálním testování není možné provádět jinou činnost. Nespornou výhodou je také správnost a přesnost, jelikož člověk provádějící manuální tes-

tování může provést v testu chybu. Testovací nástroj je vždy ve své práci konzistentní [2]. Jednou z elegantních možností automatizace testování je využití systému průběžné integrace. Konkrétní postup využití systému průběžné integrace k automatizaci testování bude dále představen v šesté kapitole.

2.2.4 Průběžná integrace

Průběžná integrace je definována jako metoda vývoje softwaru, při které je práce každého vývojáře integrována v častých a pravidelných intervalech. K použití této metody je zapotřebí, aby vývojáři vkládali zdrojový kód do centrálního úložiště zdrojových kódů. V tomto repozitáři se nachází veškeré soubory nutné k sestavení projektu a po zaznamenání změny v repozitáři se provedou veškeré automatizované činnosti včetně sestavení projektu. Sestavení projektu může zahrnovat kompilaci, přesun souborů nebo stažení potřebných závislostí projektu [9]. Sestavení aplikace je pak v systému průběžné integrace automatizováno pomocí nástrojů jako například Ant, Maven, Gradle či Make. Nástroj Maven byl využit i v této práci a bude přiblížen ve čtvrté kapitole.

Dalším aspektem průběžné integrace je využití automatizovaného testování. Mezi výhody patří možnost nové chyby rychle odhalit a ihned informovat o chybě zainteresované osoby. Testovací sada by měla být spustitelná jednoduchým příkazem a měla by poukazovat na selhané testy. Automatické testování by také mělo při objevení chyby způsobit přerušení sestavení projektu [9].

Pro správné využití průběžné integrace je nutné udržovat repozitář projektu co nejaktuálnější, proto by každý vývojář měl alespoň jednou denně(ale pokud možno častěji) veškeré provedené změny ukládat do repozitáře. Čím častěji je repozitář aktualizován, tím rychleji a jednodušeji lze nalézt a vyřešit případný konflikt v kódu vývojářů. Okamžitá zpětná vazba je jednou z hlavních vlastností průběžné integrace [9].

3 Technologie pro funkční testování webové aplikace

V dnešní době existuje pro testování webových aplikací spousta různých nástrojů. Tyto technologie se liší jak v podpoře různých webových prohlížečů, skriptovacích jazyků nebo cenou. Hlavním cílem této kapitoly není představení velkého množství těchto nástrojů, ale pouze ty, které byly použity či s použitým řešením nějakým způsobem souvisí.

Hlavní technologií, která byla při vytváření testů použita, je testovací platforma Arquillian a její rozšíření Graphene a Drone. Velmi populární alternativou je testovací nástroj Selenium, jehož komponenta WebDriver je využita i ve dvou použitých rozšířeních platformy Arquillian. Kromě těchto nástrojů zde budou představeny i další specializované nástroje využité například k sestavení aplikace, průběžné integraci či verzování projektu.

3.1 Selenium

Selenium je jeden z nejpopulárnějších nástrojů sloužících k automatickému testování webových aplikací. Jedná se o open source aplikační rámec, jehož součástí je několik komponent, momentálně hlavně Selenium IDE a Selenium WebDriver, které budou dále představeny. Původní část projektu Selenium Remote Control je již zastaralá a jejím nástupcem je Selenium Webdriver, který také obsahuje schopnosti původně samostatné části Selenium Grid. Tento celek nabízí velice flexibilní platformu umožňující tvorbu testů pro širokou škálu webových prohlížečů a operačních systémů. Ve své podstatě Selenium umožňuje ovládat instance webového prohlížeče a emulovat jeho interakci s uživatelem. Touto interakcí mohou být různé běžně prováděné úkony, jako například pohyb myší, vyplnění textového pole nebo kliknutí na tlačítka či odkazy. Jedním z principů projektu Selenium je také jednotné rozhraní pro všechny běžné prohlížeče, což umožňuje jeden test spouštět na různých webových prohlížečích bez úprav. Selenium nástroje jsou také vhodné k dalšímu rozšíření jako součást jiných aplikací [8].

3.1.1 Selenium WebDriver

Po dlouhou dobu byl hlavním modulem Selenium projektu Selenium Remote Control (nazývaný také Selenium RC či Selenium 1.0), ten měl ale značné nedostatky. Tím hlavním je samotný princip fungování Selenium RC, a to vkládání funkcí psaných pomocí JavaScriptu do webového prohlížeče po jeho načtení. Tento princip činil práci s dynamickými webovými stránkami náročnou. Další nevýhodou bylo rozdílné chování JavaScriptu v různých webových prohlížečích, což mohlo způsobit neočekávané chování testů.

Přirozenou evolucí projektu Selenium RC se stal Selenium WebDriver (nazývaný také Selenium 2.0). Ten vznikl sloučením projektu WebDriver a původního Selenium 1.0 a oproti starší verzi nabízí mnohá vylepšení. Tím hlavním je využití tzv. WebDriver API[1], které využívá nativní podpory komunikovat přímo s webovým prohlížečem. Každý podporovaný webový prohlížeč má svůj vlastní WebDriver, který zajišťuje specifika práce s tímto prohlížečem. Uživatel pak už jen využívá WebDriver rozhraní, aniž by musel řešit rozdílné chování webových prohlížečů.

Podpora webových prohlížečů a programovacích jazyků

Výhodou Selenium WebDriveru je podpora běžných prohlížečů a programovacích jazyků. Podporováný jsou následující prohlížeče: Internet Explorer, Mozilla Firefox, Google Chrome, Safari, Opera, HtmlUnit a PhantomJS. Selenium WebDriver podporuje velké množství programovacích jazyků, mezi ty hlavní patří například Java, C#, Ruby, Python a JavaScript.

3.1.2 Selenium IDE

Druhým Selenium nástrojem pro tvorbu testů je Selenium IDE[2]. Jedná se o doplněk pro internetový prohlížeč Mozilla Firefox, díky čemuž je jeho instalace jednoduchá. Stejně tak je snadné i jeho použití, není nutné mít rozsáhlé programátorské znalosti. Tento nástroj slouží k nahrání interakce s internetovým prohlížečem. Tato interakce je ulo-

1. Application Programming Interface
2. Integrated Development Environment

žena jako posloupnost speciálních Selenium příkazů, nazývaných *Selenese*. Ty se skládají z příkazu a až dvou argumentů, většinou jde o identifikátor nějakého prvku na webové stránce a hodnoty, která je příkazu předávána. Posloupnost těchto příkazů je možné dále upravovat a vkládat další *Selenese* příkazy. Tato posloupnost příkazů tvoří jeden testový případ, který lze zpětně přehrát. Více testových případů lze seskupit do testovací sady. [7]

Jednoduchost tohoto nástroje je jeho silná stránka, nicméně není příliš vhodný k tvrobě složitějších testů. Pro větší množství robustnějších testů je vhodnější použít nástroj Selenium WebDriver. Další nevýhodou je také vázanost pouze na internetový prohlížeč Mozilla Firefox. Tuto vlastnost lze částečně obejít exportem testových případů do programovacího jazyka Java, Ruby, Python nebo C# a následným udržováním v některém z testovacích aplikačních rámců.

3.1.3 Selenium Grid

Za zmínku stojí také nástroj původně nazývaný Selenium Grid. Ve své podstatě se jedná o prostředí k distribuovanému provádění testů. To sebou přináší dvě hlavní výhody. Je možné testy současně provádět na více strojích, každý s různým webovým prohlížečem (případně jinými verzemi stejného prohlížeče) nebo operačním systémem. Další z výhod je snížení času potřebného k provedení testů díky paralelnímu zpracování na více strojích. Momentálně není tento nástroj dostupný samostatně, ale jeho funkcionalita je obsažena v nástroji Selenium RC, který je také součástí nástroje Selenium WebDriver [7].

3.2 Arquillian

Arquillian je testovací platformou vyvinutou za účelem tvorby automatizovaných integračních, funkčních a akceptačních testů v prostředí Java EE[3]. Zjednodušuje testování pomocí správy běhu programu, čehož dosahuje pomocí:

- Správy životního cyklu kontejneru (kontejnerů)

3. Java Platform, Enterprise Edition

- Zabalením testovacího případu, všech závislých tříd a zdrojů do ShrinkWrap [13] archivu (archivů)
- Nasazení archivu (archivů) do kontejneru (kontejnerů)
- Obohacení testového případu poskytnutím injekce závislostí a ostatních deklarativních služeb
- Provádění testů uvnitř (nebo proti) kontejneru
- Zachycení výsledků a jejich předání spouštěči testů, který o výsledcích reportuje

Projekt Arquillian se řídí třemi základními principy. Prvním z nich je přenositelnost testů do jakéhokoliv podporovaného kontejneru. Jelikož rozhraní specifické pro jednotlivé kontejnery není v testech používáno, lze verifikovat přenositelnost aplikace spuštěním testů v různých kontejnerech.

Dalším principem je možnost spouštět testy jak přímo z vývojového prostředí, tak pomocí nástroje pro sestavení aplikace. Z vývojového prostředí je při testování možné přeskočit krok sestavení aplikace a ušetřit čas. Navíc vývojář může pracovat ve svém vývojovém prostředí, s nímž je obeznámen. Tyto výhody neznemožňují využít testy v systému průběžné integrace.

Poslední z řídících principů je rozšiřování či integrování stávajících testovacích aplikačních rámců. Spouštění Arquillian testů je proto jednoduché jak přímo z vývojového prostředí výběrem „Run As > Test“ nebo spuštěním cíle „test“ v nástroji pro sestavení aplikace.

Aby nebylo zbytečně komplikováno sestavení aplikace, tak lze Arquillian integrovat s testovacími aplikačními rámci jako například JUnit nebo TestNG. Arquillian také nabízí několik rozšíření. Těmi hlavními pro účely této práce jsou Arquillian Graphene a Arquillian Drone, které budou nyní představeny.

3.2.1 Arquillian Graphene

Arquillian Graphene si klade za cíl rozšířit možnosti Selenium WebDriver technologie a přidává spousu vlastností umožňujících psát znovu použitelné a udržovatelné funkční testy. Důraz je také kladen

na jednoduché rozhraní, přenositelnost mezi webovými prohlížeči a psaní robustních testů podporujících AJAX[4] technologii. Arquillian Graphene je závislý na rozšíření Arquillian Drone, které se stará o životní cyklus použitých webových prohlížečů. Rozšíření Drone bude podrobněji popsáno níže.

Jednou z vlastností rozšíření Graphene je podporování ve využití abstrakcí webové stránky, konkrétně objekty stránky(Page Objects) a fragmenty stránky(Page Fragments). Objekt stránky zapouzdřuje strukturu testované stránky do jediného objektu, se kterým následně test komunikuje. Takto zapouzdřený objekt by měl vývojáři nabízet stejné služby jako modelovaná stránka. Zároveň by ale měl být jediným místem, které pracuje s vnitřní strukturou webové stránky. Jednoduše řečeno se jedná o rozhraní testované stránky. Díky tomu je při změně uživatelského rozhraní testované webové stránky nutné provést změnu jen na jednom místě v objektu stránky. Výhodou využití objektů stránky je také omezení duplicity a tvorba robustního kódu [7].

Druhou možnou abstrakcí webové stránky jsou fragmenty stránky. Jde o velice podobný koncept jako v případě objektů stránky. Také jde o zapouzdření struktury, ale na rozdíl od objektů stránky nejde o zapouzdření určité webové stránky (nebo její část). Fragment stránky zapouzdřuje pouze určitou malou součást stránky, která je znovu použitelná napříč všemi testovanými stránkami [14]. Příkladem může být navigační menu, které je součástí více webových stránek. Pokud by mělo být využito pouze objektů stránek, tak by struktura tohoto navigačního menu musela být opakovaně definována ve všech objektech stránky, které jej využívají. Lze ale definovat pouze jeden fragment stránky pro toto navigační menu a v objektech stránky pouze využívat tento fragment. Konkrétní použití obou druhů abstrakcí bude popsáno v páté kapitole.

Dále Graphene nabízí vylepšení synchronizace s webovým prohlížečem. Toto je velice důležitá vlastnost, jelikož WebDriver příkazy jsou vykonávány rychleji, než dokáže webový prohlížeč provést změnu stavu stránky. Pomocí *Waiting API* rozhraní lze test synchronizovat například čekáním, dokud se na stránce nezobrazí určitý element nebo dokud nebude splněna nějaká podmínka. Další mož-

4. Asynchronous JavaScript and XML

ností synchronizace je využití tzv. *Request Guards*. Ty umožňují ověřit, zda daná interakce s webovým prohlížečem vyvolala příslušný požadavek na server. Je možno ověřovat, zda došlo k HTTP nebo AJAX požadavku, případně zda nedošlo k žádnému požadavku. Vykonávání interakce s prohlížečem musí skončit v časovém intervalu příslušném druhu požadavku. Pokud skončí dříve, tak se pokračuje ve vykonávání testu.

Dalším vylepšením oproti Selenium WebDriver technologii je také rozšíření možností, jak nalézt požadované prvky testovaného uživatelského rozhraní. Kromě metody @FindBy, kterou využívá i WebDriver, podporuje Graphene také možnost vytvořit vlastní vyhledávací strategie. Metoda @FindByJQuery, která umožňuje vyhledávat na základě JQuery dotazu, je v Graphene již vytvořená a okamžitě použitelná.

3.2.2 Arquillian Drone

3.3 Ostatní nástroje

3.3.1 Git

3.3.2 JUnit

3.3.3 Maven

3.3.4 Travis CI

4 Aplikace Slepé Mapy a její analýza

Hlavní částí této práce je vytvoření funkčních testů pro aplikaci Slepé Mapy. V této kapitole bude tato aplikace přiblížena, výsledkem analýzy aplikace bude identifikování funkčních částí aplikace. Pro tyto testované scénáře budou následně napsány testy pomocí zvolených technologií.

5 Tvorba testů

6 Systém automatizovaného testování

7 Závěr

Literatura

[1] PAGE, Alan – JOHNSTON, Ken – ROLLISON, Bj. *Jak testuje software Microsoft.* Vyd. 1. Brno: Computer Press, 2009, 384 s. ISBN 978-80-251-2869-5.

[2] PATTON, Ron. *Testování softwaru.* Vyd. 1. Praha: Computer Press, 2002, xiv, 313 s. Programování. ISBN 80-7226-636-5.

[3] *What is Functional testing (Testing of functions) in software?* ISTQB Exam Certification [online]. 2015 [cit. 2015-04-12].Dostupné z: `http://istqbexamcertification.com/what-is-functional-testing-testing-of-functions-in-software/`

[4] *What is Software Testing?* ISTQB Exam Certification [online]. 2015 [cit. 2015-04-12]. Dostupné z: `http://istqbexamcertification.com/what-is-a-software-testing/`

[5] *What is Unit testing?* ISTQB Exam Certification [online]. [cit. 2015-11-06]. Dostupné z: `http://istqbexamcertification.com/what-is-unit-testing/`

[6] MYERS, Glenford J, Corey SANDLER a Tom BADGETT. *The art of software testing.* 3rd ed. Hoboken, N.J.: John Wiley & Sons, c2012, xi, 240 p. ISBN 1118133145.

[7] *Selenium documentation.* [online]. 12.11.2015 [cit. 2015-11-12]. Dostupné z: `http://docs.seleniumhq.org/docs/index.jsp`

[8] *Selenium documentation* [online]. 12.11.2015 [cit. 2015-11-12]. Dostupné z:`https://seleniumhq.github.io/docs/index.html`

[9] Fowler, Martin. *Continuous integration.* Martin Fowler. [online]. 22.11.2015 [cit. 2015-11-22]. Dostupné z: `http://www.martinfowler.com/articles/continuousIntegration.html`

[10] *Static Vs Dynamic Testing.* Guru99. [online]. © 2015 [cit. 2015-11-26]. Dostupné z: `http://www.guru99.com/static-dynamic-testing.html`

[11] *Arquillian.* Arquillian. [online]. © 2009-2015 [cit. 2015-12-08]. Dostupné z: http://arquillian.org/

[12] *Slepé Mapy.* Slepé Mapy. [online]. © 2015 [cit. 2015-12-10]. Dostupné z: http://slepemapy.cz/

[13] *Creating Deployable Archives with ShrinkWrap* Arquillian Guides. [online]. © 2009-2015 [cit. 2015-12-11]. Dostupné z: http://arquillian.org/guides/shrinkwrap_introduction/

[14] *Graphene 2.* Project Documentation Editor. [online]. 5.8.2014 [cit. 2015-12-13]. Dostupné z: https://docs.jboss.org/author/display/ARQGRA2