Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra Informačních Technologií

Metody testování softwaru

Bakalářská práce

Autor: Lukáš Váňa
Studijní obor: AI3

Vedoucí práce: doc. Ing. Hana Tomášková, Ph.D.

Hradec Králové Duben 2022

Prohlášení:

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

vlastnoruční podpis

V Hradci Králové dne 2. 5. 2022 Lukáš Váňa

Poděkování:

Děkuji vedoucí bakalářské práce doc. Ing. Haně Tomáškové, Ph.D. za metodické vedení, za pomoc a rady při zpracování této práce.

Anotace

Bakalářská práce se zaměřuje na metody a postup procesu softwarového testování. V práci jsou popsány jednotlivé fáze a metody softwarového testování, modely softwarového vývoje a životní cyklus softwaru. Zaměření je na charakteristiky jednotlivých metod a popis jejich výhod a nevýhod spojené s jejich použitím. Práce obsahuje i vlastní návrh postupu procesu softwarového testování a implementace automatizačních testů pro webové stránky. Proces je navržen podle agilní metodiky vývoje softwaru a čerpá ze znalostí uvedených v práci podle doporučených metod při vývoji webových aplikací. Implementace automatizovaných testů pro webové aplikace je realizována pomocí populárního Selenium frameworku, sloužící jako doporučený nástroj pro implementaci navrhovaného testovacího procesu.

Annotation

Title: Software testing methods

The bachelor thesis focuses on the methods and procedure of the software testing process. The thesis describes the different phases and methods of software testing, software development models and software life cycle. The focus is on the characteristics of each method and a description of the advantages and disadvantages associated with their use. The thesis also includes the actual design of the software testing process and the implementation of automation tests for websites. The process is designed according to the agile software development methodology and draws on the knowledge presented in the thesis according to the recommended methods in web application development. The implementation of automated tests for web applications is done using the popular Selenium framework, serving as a recommended tool for implementing the proposed testing process.

Obsah

Seznam obrázků

1 Úvod

Problematika softwarového testování je dnes běžnou součástí vývoje každého softwaru. Cílem testování je zaručení a ověření kvality produktu. Pro dosažení takového cíle existuje mnoho metodik a modelů vývoje softwaru které definují proces testování pro daný software. Porozumění těchto metod je klíčové pro správou volbu modelu k dosažení požadované kvality softwaru. Tato práce se soustředí právě na popis charakteristik jednotlivých testovacích metod a modelů vývoje softwaru. Spolu s návrhem vlastního postupu testovacího procesu.

Práce je tvořena do dvou částí, první část je teoretická která slouží pro charakterizaci jednotlivých testovacích metod a vývojových modelů softwaru. Ze začátku jsou vysvětleny jednotlivé fáze testování, rozdíl mezi statickým a dynamickým testování a jak je možné k testování přistupovat. Tyto metody slouží jako úvod do problematiky softwarového testování. Další podstatnou věcí jsou samotné modely vývoje softwaru, které s testováním úzce souvisí, protože testování je nejintenzivnější právě ve fázi vývoje softwaru. Důležité je i zvolení modelu vývoje softwaru podle kterého se rozhoduje mezi použitím jednotlivých testovacích metodik. Na závěr teoretické části je stručně představena BPMN notace, pro pochopení diagramu v další části práce.

V druhé části práce je představení vlastního procesu testování. Proces je navržen podle modelu agilního vývoje určený pro webové aplikace. Tento proces je dále do detailu popsán a vysvětlen. Cílem tohoto návrhu je předložit efektivní postup procesu testování webových aplikací pro vývojáře. Spolu s návrhem je představen Selenium framework jehož hlavním cílem je seznámit se s procesem automatizace testů webových aplikací. Tento framework je vhodný pro realizaci navrhovaného postupu procesu testování.

2 Teoretická část

2.1 Co je testování softwaru

Testování hraje klíčovou roli v hledání defektů, a zajišťování, zda se počítačový software chová podle zadaných požadavků. Jedná se o proces, který se snaží co možná nejvíce zajistit správnost fungování softwaru, a nalezení co možná nejvíce chyb. Testování softwaru nám poskytuje jistotu, zdali je produkt připravený pro zveřejnění na trh, pro potencionálního zákazníka. (1)

Testování nelze podcenit, jedná se o důležitou činnost ve vývoji každého softwaru. Nedostatečné testování vede k mnoha neodhaleným problémům, které se můžou projevit negativně v budoucnu. Chyby, které by se dali snadno vyřešit během brzkého vývoje, nemusí být snadné na odhalení a opravu v pozdějším stádiu vývoje. To vede k potencionálním finančním ztrátám, pracovního času vývojářů, reputace a v extrémních případech může vést i ke ztrátám lidských životů. (2)

2.2 Fáze a úrovně testování

Software prochází za svůj životní cyklus, různými fázemi vývoje. Softwarový vývoj jako takový, je dlouhý proces trvající měsíce i roky, během kterého je velice pravděpodobně, že se setkáme s různými chybami. Hlavním účelem, proč dělíme softwarový vývoj do úrovni je právě včasné odhalení chyb, a tím redukovat vznik chyb v pozdějším stadiu vývoje. Úrovně testovaní dělíme do 4 fází, kde každá fáze testuje integritu v různých stadií vývoje. Jedná se o unit testy (testy komponent), integrační testy, systémové testy, a nakonec akceptační testy.

2.2.1 Unit testování

Unit testování neboli testování jednotek, je první úroveň testování v procesu vývoje aplikace. Účelem je, aby základní stavební prvky programu byli samo o sobě funkční. Mezi testované prvky patří například jednotlivé moduly databáze, třídy, struktura kódu a dat. Na těchto prvcích je testováno, zda fungují správně, zdali jsou navrhnuty

správně a jestli není problém s tokem dat. Testování je obvykle prováděno izolovaně od zbytku systému a jsou běžně prováděny samotným vývojářem, který kód napsal. Hlavní výhodou unit testování je možnost pro vývojáře restrukturalizovat kód v pozdější fázi a zajistit že jednotlivé moduly budou stále fungovat správně. Běžnou procedurou je sepsat testovací případy pro všechny funkce a metody, takže kdykoliv změna způsobí chybu, lze ji rychle identifikovat a opravit. Avšak nemůžeme čekat že se podaří odhalit všechny chyby, protože unit testování zkoumá jednotlivé komponenty, a nikoliv jeho interakci s ostatníma komponentami. Testování je možné provést manuálně anebo automatizovaně. Obvykle se ale volí automatizovaná forma testu, protože pro manuální postup je potřeba sepsat dokument s návodem pro postup testování což zpomaluje samotný vývoj. (3) (4)

Pro automatizaci existují různé automatizační nástroje jako například:

1. Junit
2. NUnit
3. JMockit
4. EMMA
5. PHPUnit

2.2.2 Integrační testování

Po dokončení unit testů nastává druhá úroveň testování takzvaná integrační testování. Rozdíl spočívá v testování komponent jako celku, kde jednotlivé komponenty na sebe reagují. Důvod tohoto testu je odhalení chyb v komunikaci mezi moduly, které se u předešlého testování nedala odhalit. Obzvlášť u větších softwarových projektů, kde spolupráce jednotlivých vývojářů je nezbytná, hraje integrační testování velkou roli v udržování integrity a funkčnosti celého softwaru. Podstatou testování je testování systému nikoliv funkčnost jednotlivých modulů, toto testování je téměř celé z automatizované a je převážně prováděno testerem.
Integrační testování má 4 strategie přístupu, přičemž první 3 přístupy se řadí do inkrementálních postupů, které fungují systémem nabalování modulu na předešlé,

tento proces pokračuje, dokud nejsou všechny moduly integrovány a otestovány. Výhodou inkrementálního přístupu je snazší odhalení chyb za cenu pomalejšího vývoje. (4) (5)

Jednotlivé přístupy integračního testování:

1. Od shora dolů přístup
2. Od zdola nahoru přístup
3. Sandwich přístup
4. Big bang přístup

2.2.3 Systémové testování

V systémovém testování se posouváme od jednotlivých modulů na vyšší úroveň. Kde se k přihlíží na správný chod celého systému. Testují se zejména konkrétní funkčnosti v daném systému, které byli brány v potaz při návrhu softwaru se zákazníkem. Jde tedy o postup testování z pohledu zákazníka, kde se koukáme na celý systém jako na černou skříňku, kde nevidíme, jak vše funguje, ale zdali vše funguje. Pokud je objevena chyba je zapotřebí provést všechny testy, prováděné v této úrovni, od znova aby se udržela konzistence systému. Na základě těchto testů je potom vyhodnocováno, zda se daný software uvede na trh. (4) (6)
Mezi jednotlivé testy spadající pod systémové testování jsou například:

1. Usability testing
2. Load testing
3. Regression Testing
4. Recovery testing
5. Migration testing

Systémové testy jsou běžně prováděny nezávislým testerem, který postupuje podle takzvaných end-to-end scénáře, který popisuje postup chodu systému. Tato metoda se snaží co nejblíže napodobit interakci koncového uživatele s aplikací za účelem zajištění požadovaného výstupu a integraci dat. (4) (6)

2.2.4 Akceptační testování

V poslední fázi testování, takzvané akceptační testování se testuje už hotová aplikace. V této fázi se klade důraz na splnění potřeb a požadavků zákazníka k aplikaci. To znamená že akceptační testování se nesnaží najít defekty i přesto že to je možné, ale snaží se zajistit jistou satisfakci koncového uživatele s danou testovanou aplikací. Příliš nadměrné množství defektů odhalené v akceptačním testování je známkou nedostatečného předešlého testování a ukazuje na velký risk a nestabilitu celé aplikace. Testování je prováděné převážně koncovým uživatelem nebo samotným klientem. Akceptační testování se dělí na čtyři typy testování. (4) (7)

2.2.4.1 Uživatelské akceptační testování

Uživatelské akceptační testování se čistě zaměřuje na komfort uživatele s aplikací, kde se testuje, zda aplikace vyhovuje uživateli v běžném prostředí, a zda je schopen pracovat s co nejmenší obtížností, náklady na cenu a riziky. (4) (7)

2.2.4.2 Operační akceptační testování

Operační akceptační testování je zpravidla prováděno provozními pracovníky nebo správci systému. Testy se zaměřují na správnou funkčnost podpůrných operací jako například testování zálohy a obnovy systému, instalování, odinstalování, aktualizaci dílčích komponent, vyhledávání bezpečnostních rizik nebo testování výkonu samotného systému. (4) (7)

2.2.4.3 Smluvní a regulační akceptační testování

Smluvní a regulační testování zajišťuje, aby vyvíjený software byl v souladu s určitými kritérií a specifikací, které jsou předem definovány ve smlouvě a jestli jsou v souladu s předpisy. Projektový tým definuje kritéria a specifikace pro akceptaci ve stejnou dobu, kdy se celý tým dohodne na samotné smlouvě. Regulační akceptační testování se provádí na základě všech předpisů, které musí být dodrženy, jako například vládní, právní, nebo bezpečnostní předpisy. Jak smluvní, tak

regulační testování je prováděno nezávislými testery a v případě regulačních testů se můžou na testování podílet regulační orgány. (4) (7)

2.2.4.4 Alpha a Beta testování

Alpha a Beta testování je zejména používáno k poskytnutí zpětné vazby od koncových uživatelů. Alpha test je provozován přímo na místě vývoje aplikace a je proveden vybranými koncovými uživateli a výjimečně nezávislým vývojářem který co nejvíc napodobuje samotného uživatele. Beta test následuje po alpha testech, ale je možné přejít do fáze beta testu bez ohledu na to, jestli byl alpha test uskutečněn nebo ne. Beta testování je prováděno čistě koncovými uživateli na vlastních zařízení a je to poslední testování před uvedením aplikace na trh. (4) (7)

2.3 Statické testování

Statické testování neboli statická analýza zkoumá zdrojový kód aplikace bez toho, aniž by byla aplikace spuštěna a za určitých podmínek i bez existence samotného softwaru. Vzhledem k tomu že je testován nespuštěný kód, je nejefektivnější uplatnění statického testování v ranní fázi vývoje za účelem nalezení co nejvíce defektů, které můžou mít negativní dopad v pozdějším stádiu vývoje. Tento postup je v dnešní době z velké části automatizován, avšak můžeme se setkat i s manuální formou testů. Automatizace je realizována takzvanýma statickýma analyzátory, který provádějí analýzu na úrovni zdrojového kódu, objektového kódu či bajtkódu. Ty kontrolují daný kód a upozorňují na chyby ať už známé pro daný jazyk nebo definované přímo uživatelem. Pokud uživatel definoval vlastní pravidla kontroly statického analyzátoru je zapotřebí být velice opatrný za účelem vyhnutí falešného hlášení chyb. (8)

2.3.1 Testování dokumentace

Kromě statického testování kódu existuje i testování projektové dokumentace. Dokumentací je myšleno například manažerské výkazy, uživatelskou či technickou dokumentací a všechny dokumenty spojené s realizací softwaru nebo vzniklé během softwarového vývoje. Je testována správnost, úplnost a konzistence obsahu. (9)

2.3.1.1 Metody testování dokumentace

Testování samotné dokumentace prochází třemi kroky. První krok je ověření dodržení určených projektových šablon, podle kterých se daný dokument vytvářel. Druhý krok je odstranění všech možných pravopisných či gramatických chyb, kterých se mohl autor dopustit. (9)
A třetím krokem se volí alespoň jedna z těchto metod:

Neformální revize – Je neformální metoda, na které se podílí kolegové autora dokumentu spolu se samotným autorem dokument. Obě strany prochází obsahem dokumentu v pátrání po nepřesnostech či nedostatcích. Výhodou této metody je jednoduchost a efektivita kde není pořizován žádný záznam o nalezených chybách, protože jsou zpravidla opravovány přímo na místě. Nevýhodou je že efektivita revize je závislá na znalostech účastněných kolegů. (8) (9)

Procházení dokumentu – Jedná se o skupinovou aktivitu iniciovanou a řízenou autorem dokumentu. Má tedy větší formálnost a může nabývat například formy prezentace kde autor přestavuje svůj vypracovaný dokument. Dílčí osoby potom pokládají otázky nebo vyvolávají diskuzi o daném tématu. (8) (9)

Technická revize – Technická revize je ve své podstatě opak neformální revize. Jde tedy o formální validaci, která zkoumá, zda projekt splňuje požadavky zamýšlené v plánu projektu. Jednotliví účastníci jsou odborníci ve svých oborech díky čemuž mají velkou moc v rozhodování o dalším postupu. Výstupem této metody je seznam nalezených problémů, přepsání obsahu nebo akceptace či zamítnutí dokumentu. (8) (9)

Inspekce – Samotná inspekce je nejformálnější metodou testování dokumentace. Testování probíhá přesně v daný čas, kde každý účastník má předem definované role, jako například moderátor, průvodce, autor, oponenti a zapisovatel. Tato metoda je vysoce efektivní ale zase náročná na čas a zkušenosti jednotlivých zúčastněných. (8) (9)

2.3.2 Metody statického testování kódu

Metody statického testování pro kód se příliš neliší od metod pro testování dokumentace. Používají stejná hodnotící kritéria a jednotlivé výstupy testu si jsou podobné. Avšak u testování kódu požíváme tři hlavní metody:

Revize kódu – Revize kódu funguje podobně jako u dokumentace. Zkušenější kolega prochází zdrojový kód ostatních, většinou tak aby nenarušoval práci. Z důvodů dobré praktiky je třeba stanovit podmínky za kterých se má daný kód podrobit revizi, aby nedošlo k nadbytečnosti revizí. Na základě výstupu revize se rozhoduje, zda se kód přepracuje nebo postoupí na další úroveň testování. (9)

Párové programování – Je princip vývoje softwaru kde se na jednom úkolu podílí dva developeři na tom samém zařízení. Tento přístup vývoje zpravidla produkuje menší množství chyb, protože snadno opravitelné chyby jako překlepy a odchylky od zadání jsou přímo opravovány spolu vývojářem. I přesto se spíše jedná o extrémní způsob programování, který se dá využít pro začlenění nebo zaučení nového programátora do stávajícího projektu. Velkou nevýhodou je samozřejmě redundantní počet vývojářů, což u implementací jednoduchých komponent se může jevit jako plýtvání pracovní silou. (9) (10)

Automatické testování kódu – Psaní kódu je často realizováno v kódovacím prostředí, které už obsahuje automatické testování kódu ve výchozím stavu. Programové vývojové prostředí zpravidla kontroluje správnou syntaxi zápisu a upozorňuje na logické chyby. (9)

2.4 Dynamické testování

Na rozdíl od statického testování, dynamické testování je metoda zabývající se chováním softwarového kódu ve spuštěném stavu. Je testováno dynamické chování za účelem testovat dynamické proměnné které nejsou konstantní a mohli by

způsobit chyby za běhu softwaru. Pro využití dynamického testování je zapotřebí mít aplikaci v konzistentním stavu, z toho důvodu je dynamické tetování uplatňované především v pozdějším stádiu vývoje softwaru. Avšak to neznamená že dynamická metoda nahradí statickou, obě metody mají svoje cíle. Můžeme říci že statická metoda je proces verifikace a dynamická metoda je proces validace. Velmi účinnou strategií je použití obou metod, tedy kombinace dynamického a statického testování, především hlavně v oblasti bezpečnosti. (8) (11)

2.5 Testování podle viditelnosti kódu (White/Black/Grey box)

Vývoj softwaru v dnešní době je doprovázen různými specifikami, návrhovými dokumenty, dalšími fungujícími systémy a podobně. To znamená že i když neexistuje žádný specifický kód, z části víme, jak bude vypadat. Z této myšlenky plyne jeden koncept v oblasti testování softwaru. Je možné testovat různými metodami na různých úrovní viditelnosti. V testování softwaru máme tři základní skupiny technik, kde každá skupina funguje na jiné úrovni. Tyto přístupy jsou černá skříňka, bílá skříňka a speciální metoda šedá skříňka která je kombinací obou předchozích metod. Velká většina konkrétních testovacích technik vzchází právě z těchto metod. (12)

2.5.1 Černá skříňka

Metoda černé skříňky je založena na jednoduché analogii. Při poskytnutí vstupu do černé skřínky dostaneme nějaký výstup bez toho, aniž bychom věděli, jak jsem se k takovému výsledku dostali. Jednotliví testeři nemají žádné znalosti o tom, jak vnitřní struktura systému vypadá a řídí se jen specifikacemi funkčnosti systému. Jejich hlavním cílem je ověřit funkcionalitu systému, tedy jestli po zadání všech vstupů dostane požadované výstupy. Z toho důvodu je metoda černé skřínky označováno jako funkční testování, avšak tomu tak není vždy pravda. Výhodou této metody je že testeři nepotřebují znát vnitřní strukturu testovaného softwaru. A často ukazuje na možné vylepšení uživatelského rozhraní. Nevýhodou je že aby bylo

testování efektivní je třeba testovat malé části softwaru. Protože testování příliš velkých částí softwaru může být časově náročné a neefektivní. (12) (13)

2.5.2 Bílá skříňka

Opakem metody černé skřínky je logicky bílá skřínka, často označované jako strukturální testování. V této metodě oproti předchozí metodě, víme o vnitřní struktuře kódu a na jejím základu se určuje postup testování. Jde o velmi důkladný proces testování, se zaměřením na řízení toku dat. Prováděný zkušeným testerem, který má přehled o struktuře daného kódu. Vybaveným speciálním nástrojem na analýzu zdrojového kódu. Tato metoda je časově náročná a komplikovaná, za to ale velice účinná v ladění a optimalizování kódu. Odhaluje zbytečné části kódu a upozorňuje na takzvaná úzká hrdla (bottleneck), která způsobují zpomalení chodu celé aplikace. Tato metoda operuje na integrační úrovni, ale může se vyskytnou i na jednotkové úrovni. (12) (13)

2.5.3 Šedá skříňka

Šedá skříňka je spojení obou předešlých metod. Důvodem vzniku téhle metody je snaha o spojení výhod obou metod testování bez negativních efektů. Při návrhu testování jsou k dispozici návrhové dokumenty spolu do určité míry se zdrojovým kódem. Testování se podobá rozšířené metodě černé skřínky, kde se počítá s pokrytím celého systém bez narušení provozu. Tato metoda je nejvíce užitečná v integrační fázi vývoje, a můžeme se s ní setkat u testování webových aplikací. (12) (13)

2.6 Manuální testování

Manuální testování je proces testování prováděným testerem, který vykonává testovací případ manuálně. Tyto případy se zpravidla designují, aby co nejvíc napodobovali aktivitu koncového uživatele. Manuální testování je jedna z nejprimitivnějších metod testování ale taky ta nejdůležitější. Je na jejím základě postaveno mnoho odvozujících metod. Velká většina testů je právě manuální a bez

manuálních testů by automatizace nebyla možná. Prvotní testy každého softwaru jsou prováděny manuálně avšak nadměrné opakování testů je časově a finančně náročné. Jelikož se jedná o manuální metodu je velké riziko ve validaci testů, protože testování je prováděno lidmi, kteří jsou náchylní na chyby. Obecně je dobré většinu manuální testů z automatizovat ale existují typy manuálních testů u kterých se nevyplatí automatizace nebo by postrádala smysl. (14) (15)

Mezi tyto typy patří například:

Průzkumné testování – Je testování, které nemá předem definovaný testovací případ. Během samotného testu, testeři komunikují a zjišťují co a jak otestovat a podle toho určují testovací případ. Tato metoda vyžaduje jistou mírů kreativity a zkušenosti testerů. (14) (15)

Testování použitelnosti (Usability testing) – Zhodnocuje, jak moc je softwaru přívětivý pro koncového uživatele. Zda se dokáže orientovat v prostředí a jestli je software efektivní a pohodlný. Metodu testování použitelnosti lze testovat pouze manuálně, protože veškerá kritéria nelze měřit automaticky. (14) (15)

Ad-hoc testování – Se provádí bez jakékoliv přípravy nebo dokumentace. Obvykle se používá u testů, které nejsou tak rozsáhlé. Testování nepostupuje podle žádné struktury a hlavním účelem je hledání chyb pomocí náhodné kontroly. (14) (15)

2.7 Automatické testování

Automatické testování je jakékoliv testování prováděné softwarem. Je třeba říci že automatizace nahrazuje pouze některé činnosti testera, protože zpravidla nejde proces testování plně z automatizovat. Proto je automatizace nedílnou součástí manuálního testování, kde automatizujeme opakované činnosti pro ušetření času. Další výhodou je že automatické testování je velice spolehlivé a efektivní. V běžných testech hraje roli lidský faktor, kde je větší šance na chybné vyhodnocení testu.

Všechny tyto výhody přicházejí s velkými počátečními náklady. Automatizace je realizována pomocí skriptů, které je potřeba udržovat po celou dobu jejich využití. Je proto přínosné implementovat automatizaci už v brzké fázi vývoje softwaru, kde z dlouhodobého hlediska se automatizace vyplatí víc než čistě manuální metoda. (9) (12)

Mezi jednotlivé techniky automatizovaného testování jsou následující:

Zachycení a přehrávání aktivity uživatele – V principu jde o nahrávání aktivit uživatele testovacím nástrojem. Těmito aktivitami se rozumí například testovací případy, které lze po nahraní zpětně přehrát. Automatizační nástroj potom zhodnotí výstup testu podle určitých proměnných a porovná s očekávanými hodnoty po skončení testu. Je to velice efektivní metodu, která potencionálně ušetří mnoho času. Avšak jak už z názvu vyplývá, jedná se o metodu nahrávání, což znamená, jakou si citlivost na změny ovládacích prvků testované aplikace. Například když při nahrávání je použita ovládající jednotka, která se vymění za novější verzi. Je pravděpodobné že testy přestanou fungovat a je zapotřebí celí testovací případ znovu nahrát. (12)

Modifikace vygenerovaných skriptů – Ve své podstatě jde o upravenou předchozí metodu. Tedy vychází z nahraných skriptů, ale díky skriptovacím jazykům, je možné upravit a díky tomu vylepšit daný skript. Metoda modifikace vygenerovaných skriptů nabízí lepší znovu použitelnosti skriptů v podobně rozšíření obsahu testu a variability proměnných. Za cenu složitější implementace. (12)

Testování řízené daty – Je metoda, která kromě běžných testovacích skriptu používá tabulky s daty o vstupech a očekávaných výstupech. Daný testovací skript vykoná podle vstupních dat testovací případ a získané výstupy porovná s očekávanými výsledky z tabulky. Metoda testování řízené daty umožnuje snadné testování pozitivních a negativních případů. Tabulky jsou nahrávány po řádcích a obvykle bývají uloženy například v souborech typu CSV a XLS. (12)

Testování řízené klíčovými slovy – Je velice podobné s metodou testování řízené daty. Rozdíl spočívá ve struktuře datové tabulky. Kde místo vstupních dat jsou vstupem samotné operace a příkazy. Mezi tyto operace patří například „Kliknutí na tlačítko“ nebo „Ověření hodnoty“, kde tyto operace nazýváme jako klíčová slova. Pomocí těchto klíčových slov se sestavují jednotlivé testovací případy. Pro lepší představu se dá říct, že říkáme automatizačnímu nástroji, jaké příkazy a jak za sebou mají následovat pro uskutečnění testu. Výhodou této metody je dobrá udržovatelnost a snadné použití i bez znalosti skriptovacího jazyka. Nevýhodou je složitější implementace do systému a definování klíčových slov bez kterých nejde test uskutečnit. (12)

2.8 Životní cyklus vývoje softwaru

Pokud se bavíme o testování softwaru, je třeba zmínit, jak se takový software vyvíjí. Právě s vývojem softwaru se nedílně pojí softwarové testování.
Software za dobu svého života prochází různými etapami vývoje, ať už se jedná o fázi plánování, nebo fázi ukončení daného softwaru. Tyto fáze jsou součástí životního cyklu vývoje softwaru a definují jakým způsobem bude daný software vyvíjen.
Účelem je zlepšení kvality softwaru a obecného procesu vývoje.

Ve základní podobě existuje šest hlavních fází vývoje softwaru. Ale je možné se setkat s více nebo méně fázemi, kde jsou jednotlivý fáze spojeny do jedné nebo naopak rozděleny do více fází.

Plánování a analýza požadavků – První fáze se zaměřuje především na potřeby projektu. Patří sem například plánování nákladů nebo definování rozsahu aplikace. Během této fáze je běžně zhodnocena zpětná vazba od zákazníků, pro který je aplikace vyvíjena. (16)

Definování požadavků a návrh – Všechny požadavky ve fázi plánování jsou v této fázi upřesněny a jasně definovány pro implementaci aplikace. Cílem je se vyhnout

jakýmkoliv nepřesnostem které by mohli být v rozporu s původními požadavky zákazníka a vyústění tak v podobě nežádoucí aplikace. K tomu je využíván takzvaný SRS (Software Requirment Specification) dokument, který představuje kompletní sepis všech specifikací. (16)

Implementace a kódování – Po zpracování návrhového dokumentu začíná samotná implementace. Podle designu se zpracovává zdrojový kód aplikace a implementují všechny součásti softwaru. Některé designové prvky můžou být například architektura kódu, bezpečnost, prototyp uživatelského prostředí nebo zabezpečení aplikace a rozhodnutí na platformě. (17)

Testování – Testování má poskytnout jistotu, že všechny funkční operace fungují správně. Ačkoli testování probíhá po cele době životního cyklu, v této fázi je ta nejdůležitější část testů. Tyto testy kontrolují stav aplikace před tím, než bude zveřejněná zákazníkům a koncovým uživatelům. (16)

Nasazení – Po testech nastává nasazení aplikace na trh. V této fázi je software zveřejněn koncovým uživatelům. Podle rozsáhlosti projektu jsou náklady na uvedení aplikace na trh rozdílné. Je třeba brát ohled například na počet nových uživatelů, aby aktivita mnoha uživatelů nezpůsobila pád celé aplikace. Zpřístupnění softwaru je většinou realizováno automaticky, tedy zakoupení nebo stažení aplikace z webové stránky. (17)

Provoz a údržba – V době, kdy je software zveřejněn je zapotřebí udržovat plynulí stav chodu. Koncový uživatelé stále mohou najít chyby které je třeba co nejrychleji opravit. Taky v této fázi je místo k možnému rozšíření softwaru o další funkcionality a zlepšení kvality služeb. (17)

2.9 Modely vývoje softwaru

Vývoj software je složitý proces, kde každý software je zapotřebí vyvíjet jinak než ostatní. To právě řeší jednotlivé modely životního cyklu softwaru. Tyto metody

poskytují plán, podle kterého bude software vyvíjen. Odvíjí se od standartního vývoje k specifickým metodám. Je možné se setkat i s kombinovanou formou kde není přímo řečeno který metodě se vývoj softwaru blíží. (9) (18)

2.9.1 Vodopádový model

Vodopádový model je vůbec první model vývoje softwaru. Tato metoda je založena na úvaze připomínající obyčejný vodopád. Celý proces vývoje se skládá z jednotlivých fází, které na sebe navazuji. Tedy proto aby mohla nastat daná fáze je zapotřebí aby všechny předchozí fáze byly ukončeny. Samotný proces je jednosměrný a nedá se zpětně vracet do dřívějších fází. Už to ukazuje že tento model je zastaralý a neflexibilní. Z toho důvodu se v dnešní době vyplatí tento model využít u malých projektů. I přesto tento model nabízí jisté výhody jako například jednoduchost a jasnost procesu vývoje s minimálním plánováním. Kde v každém okamžiku je pevně dáno, v jaké fázi se projekt nachází. Tato metoda spolu nese řadu nevýhod, které můžou mít větší váhu než předem zmíněné výhody. Vzhledem k tomu že je proces neflexibilní je skoro nemožné, jakkoliv upravit software v pozdějších fází. To znamená že pokud je software špatně navržen je velice pracné celý proces upravit. Většinou je zapotřebí projí celý proces vývoje od znova. (12) (18)

2.9.2 Agilní vývoj

Hlavním důvodem vzniku agilní metodiky je odchýlení od tradičních neefektivních metod. Agilní metoda přestavuje adaptibilitu, flexibilitu a přizpůsobení k změnám ve vývojovém procesu. Tato metodika vnikla jako reakce na starší metodiky, které byli považovány za zastaralé z pohledu na realizaci změn k vývoji softwaru. Agilní přístup je silně orientován na interakci členů v týmu kde každý může vyjádřit svoje názory na přístup vývoje. Je upřednostňována práce před nadbytečnou dokumentací a definovaných procesních postupů. Všechny aktivity, které jsou považovány jako zbytečné jsou vyloučeny z procesu vývoje. Vývoj připomíná neformální styl vývoje, kde je kladen důraz na dokončení a kvalitu softwaru. Vývoj

je realizován formou iterace kde se po dobu krátkých časových intervalů, často několik týdnů, implementují požadavky zákazníka podle definované priority. (12)

Z agilního vývoje vycházejí další metodiky jako například SRCUM, extrémní programování a vývoj řízený vlastnostmi.

Extrémní programování – Tato metoda využívá principy ostatních agilních metodik, které dovádí do extrémů. Délka iterace u ostatních metodik může nabývat měsíce či týdny. U extrémního programování délka iterace můžou být dny nebo dokonce jediný den. Testování probíhá neustále. Právě díky krátké době iterací proběhne jak návrh, tak implementace, po kterých následuje rovnou nasazení do systémů. Komunikace mezi vývojovým týmem a zákazníkem je taky extrémní, kde komunikace mezi zákazníkem a členy týmu je natolik intenzivní, že se zákazník stává do jisté míry členem týmu. (12) (18)

SCRUM – Je metodika projektového řízení, která funguje na principech agilního vývoje. Vývojový tým má k dispozici produktový backlog, který představuje kolekci požadavků a funkcí na implementaci. Jednotlivé iterace vývoje jsou takzvaný sprinty. Jeden sprint trvá několik týdnů, během kterých se implementují funkce z takzvaného backlog sprintu. Ten se skládá z jednotlivých požadavků určených k implementaci z produkt backlogu v daném sprintu. Začátek každého dne v sprintu je doprovázený setkáním všech členů týmu pro z revidování dokončené práce a další zpětné vazby. (12) (18)

2.9.3 Spirálový model

Je model, jehož procesní postup připomíná graficky spirálu. Jednotlivé fáze modelu jsou vyjádřeny jako kvadranty, které spirála protíná, kde každý cyklus spirály nazýváme smyčkou. Tato smyčka reprezentuje jednu procesní iteraci modelu. Počet smyček je neznámý, protože každý projekt vyžaduje jiné množství iterací. Výhodou této metody je její vlastnost řešit rizika projektu. Riziky je myšleno libovolná situace, která má negativní vliv na projekt. Nakládaní s riziky je řešeno vytvořením

prototypové verze projektu v každé vývojové smyčce, které pomáhají určit potencionální rizika projektu. (19) (20)

Jednotlivé fáze spirálového modelu:
Plánování – Shrnutí požadavků, revize a určení proveditelnosti.
Analýza – Identifikace rizik, navrhnutí prototypové verze.
Vývoj – Implementace a testování požadavků.
Vyhodnocení – Zákazník zhodnotí nejnovější verzi softwaru.

2.9.4 V-model a W-model

V-model je obdobný model vycházející z vodopádového modelu. Jedná se o variaci vodopádového modulu, kde jednotlivé fáze orientující se na návrh a specifikace, korespondují s fázemi spojené s jednotlivými úrovněmi testování. Tyto páry dohromady tvoří diagram ve tvaru jednoduchého V. Tato metoda zdůrazňuje testování na odpovídající návrhové fáze, na rozdíl od vodopádového modelu, kde testování probíhá na konci implementace. Jedná se o složitější a dražší model, který díky procesu validace a verifikace nabízí větší kvalitu výsledného softwaru. Model je vhodný pro malé projekty s pevně danými požadavky. (9) (27)

W-model je vylepšení v-modelu. W-model funguje na stejným principu jako v-model, kde návrhové fáze stále tvoří páry s testovacími fázemi. Avšak návrhové a testovací procesy jsou na sebe nezávislé. To znamená že testování probíhá paralelně vůči implementačním procesům. Takovéto rozdělení nabízí testerům větší volnost v hledání chyb. (9) (21)

2.10 BPMN (Business Process Model and Notation)

BPMN je grafická notace určená pro modelování a analýzu firemních procesů. Vyskytuje se v podobě vývojového diagramu, který znázorňuje jednotlivé kroky obchodních procesů a snaží se vizuálně popsat jejich posloupnost. Za vývojem BPMN notace stojí BPMI organizace jejímž cílem je vytvoření notace, která bude snadno

čitelná a srozumitelná pro všechny účastníky obchodních procesů. Důvodem vzniku BPMN notace je snaha o vylepšení komunikace mezi jednotlivými členy návrhových a implementačních procesů. BPMN je užitečná i pro členy který se přímo nepodílí na návrhu a implementaci, jako například stakeholdeři kteří potřebují mít přehled o firemních procesech. (22) (23)

Pro reprezentaci BPMN notace existuje business process diagram (BPD), který zachycuje graficky firemní proces. Diagram se skládá z uzlů a přechodů připomínající spojitý graf. Jednotlivé prvky diagramu spadají do jedné ze čtyř základních skupin elementů: objekty skládající procesní vlákna (Flow Objects), propojovací objekty (Connecting Object), kontexty objektů (Swimlanes) a artefakty (Artifacts).

2.10.1 Základní objekty skládající procesní vlákna (Flow Objects)

Aktivity (Activities) – Aktivita je jakákoliv specifická činnost nebo úloha organizace v daném firemním procesu. Aktivita se může dále dělit na atomická a neatomická. V diagramu je aktivita reprezentovaná obdélníkem se zaoblenými rohy. (22) (23)

Události (Events) – Událost je spouštěč, který reaguje na danou podmínku a upravuje tok firemního procesu. Podmínkou je například zpráva, časovač, chyba nebo signál. Události lze kategorizovat do dvou skupin, vyvolávané (Throwing) nebo odchytávané (catching). Vyvolávané události slouží k modelování výstupní události která zasílá při spuštění danou událost, odchytávané události mají definovaný spouštěč pro odchycení vstupní události. Události jsou znázorněny prázdným kruhem se symbolem odpovídající danému spouštěči. Vyvolávané události se značí vyplněným symbolem a odchytávané se značí prázdným symbolem. (22) (23)

Brány (Gateways) – Brány jsou určené pro řízení toku sekvence. Představují rozhodovací bod, který upravuje cestu sekvence a připomínají logické funkce. Brány jsou graficky reprezentovány jako kosočtverec se symbolem definující logickou funkci brány. Exclusive brána přestavuje logický XOR značený prázdným

kosočtvercem nebo symbolem X, inclusive brána je logický OR značený kruhem, parallel brána představuje logický AND značený symbolem +, a complex brána je požitá výjimečně jako krajní řešení, pokud žádnou jinou bránu nelze použít, identifikovanou jako tři protínající se čáry uprostřed. (22) (23)

2.10.2 Propojovací objekty (Connecting Object)

Tok sekvence (Sequence Flow) – Určují návaznost aktivit ve firemním procesu. Sekvence toku je znázorněna přímkou s vyplněnou šipkou a každá sekvence má jeden zdroj a jeden cíl. (22) (23)

Tok zpráv (Message Flow) – Znázorňuje tok zpráv posílané mezi rozdílnými pool objekty. Zprávy jsou čistě informativní a zachycují komunikaci mezi dvěma účastníky. Tok zpráv je reprezentován přerušovanou přímkou s prázdnou šipkou. (22) (23)

Asociace (Association) – Definuje vztah mezi elementem diagramu a jeho popisem. Asociace není součástí toku sekvence a pouze asociuje artefakt nebo text s událostí nebo aktivitou. Asociace se vyznačuje tečkovanou čarou. Šipka asociace je pouze orientační a naznačuje, zda jde o zápis nebo čtení. (22) (23)

2.10.3 Kontexty objektů (Swimlanes)

Pool – Reprezentuje účastníka ve firemním procesu. Účastník je například společnost nebo organizace, účastníkem může být i samotná entita jako třeba zákazník, dodavatel nebo výrobce. Pool je reprezentován jako ohraničená oblast firemního procesu, kterou není možné překročit v rámci procesu. Mimo poolu lze pouze posílat zprávy jiným účastníkům. (22) (23)

Lane – Lane je dále rozdělená část poolu účastníka firemního procesu. Přestavuje roli v dané organizaci, kde se odehrává celý proces. Lanes poskytují kontext toho kdo je zodpovědný za danou část poolu. Tok sekvence firemního procesu nemůže

překročit ohraničení poolu, ale může zasahovat do rozdílných lanes v jednom poolu. (22) (23)

2.10.4 Artefakty (Artifacts)

Artefakty jsou podpůrné objekty, které mají za úkol přidat dodatečnou informaci a procesu bez toho, aby samotný proces ovlivňovali.

Datový objekt (Data Object) – Existuje více datových objektů, které mají vlastní reprezentaci v diagramu. Využití datových objektů se používá jen pouze pokud je zapotřebí specifikovat jaké data proces nebo aktivita využívá. (22) (23)

Skupina (Group) – Představuje možnost sjednocení více elementů do logického celku. Graficky reprezentovaný prázdným obdélníkem s přerušovanými hrany. (22) (23)

Textová anotace (Text Annotation) – Poskytuje možnost vložení textu pro bližší specifikace relevantní k procesu. Textová anotace je reprezentována textovým polem, které má z jedné části pole ohraničení. (22) (23)

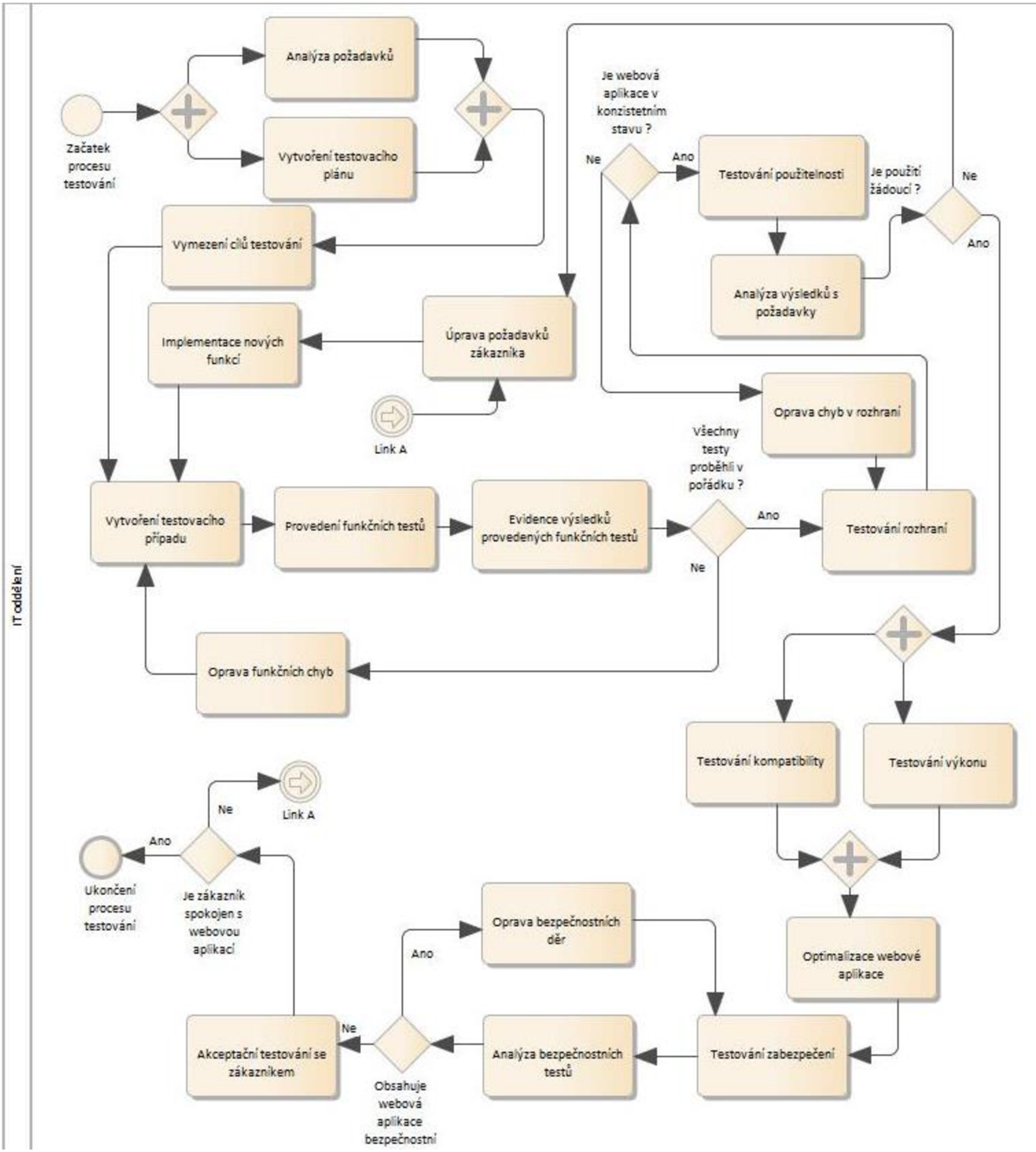
3 Praktická část

3.1 Proces softwarového testování

Testování softwaru je komplexní obor s mnoha metodami, jak k jistým problémům přistoupit. Spolu s procesem vývoje softwaru tvoří velkou část ve které když se chce někdo vyznat, tak se neobejde bez vše možných návodů a podrobných dokumentů s instrukcemi. U testování softwaru se takovému postupu, který definuje průběh testování říká životní cyklus softwarového testování a ten právě definuje, jak k softwarovému testování přistoupit. Jednotlivé etapy životního cyklu testování jsou podrobně definované a popsané, za účelem zajištění správného testování a docílit tak kvalitního softwaru. Součástí životního cyklu testování je například sběr požadavků potřebné pro zajištění správnosti testování, navrhování testovacích případů, provedení samotného testu a samotné vyhodnocení provedeného testu.

3.2 Návrh procesu testovaní webových aplikací

Účelem této práce je představit vlastní proces testování u webové aplikace. Proces je navržen v souladu s agilním vývojem softwaru. Agilní vývoj představuje flexibilní vývojový model, u kterého lze snadno upravit požadavky zákazníka bez velkých rizik. Celý proces se soustředí převážně na automatizované testy různých testovacích metod. Dohromady jsem vybral sedm nejpodstatnějších metod, které považuji za nejdůležitější pro dosáhnutí kvalitní webové aplikace. Mezi vybrané metody patří: funkční testy, interface testy, usability testy, testy výkonu a kompatibility, bezpečnostní testy, a nakonec akceptační testy. Navržený postup obsahuje i dodatečné doporučení úprav testovacího procesu.
Z business proces diagramu níže vysvětlím jednotlivé fáze a postupy v procesu testování.



Obrázek 1: Návrh procesu testování [Vlastní zpracování]

3.3 Analýza požadavků a vytvoření testovacího plánu

V počáteční fázi je potřeba si pevně definovat mezi členy podílející se na testování aplikace, jak při testování postupovat. Toho lze docílit pomocí testovacího plánu. Testovací plán je dokument, který přestavuje soupis všech použitých technologií a metod testování, kterého se každý člen musí držet při postupu testování.

Testovací plán obsahuje i bližší specifikace požadavků a výpis všech testovacích případů. Nicméně z vlastního usouzení bych nedoporučil vypisovat specifikace a testovací případy přímo do testovacího plánu. Protože testovací plán je velice rozsáhlí dokument, ve kterém změny způsobí akorát zmatek pro členy týmu. V agilním prostředí se předpokládá jistá míra změn a specifikace se mění v nepravidelných intervalech podle požadavků zákazníka. Z toho důvodu bych doporučil vytvořit zcela vlastní dokument pro požadavky a testovací případy, který bude přehledný pro provedené změny v požadavcích. Tento nový dokument obsahuje i testovací případy. Důvod je prostý, na základě požadavků definované zákazníkem se vytváří i testovací případy. Ty reflektují blízce požadavky od zákazník a tím tedy dává smysl jejich asociace v jednom dokumentu.

3.4 Cíle testování

Vymezení cílů slouží k jako odpověď na otázku: Co je podstatné, aby v systému fungovalo? Nestačí pouze říct že systém má fungovat, protože to je obecný přístup a nedává testerovi cíle, ke kterým má směřovat. Cílem tak může být například splňování norem dané úřadem. I když se na první pohled může zdát takovýto cíl jasný, v realitě to tomu tak být nemusí. Z toho důvodu před začátek testování je nejlepší si takové cíle kladené na systém vymezit.

3.5 Testovací případy

O testovacích případech už byla jistá zmínka nicméně je dobré si proces trošku blíže vysvětlit. Jak bylo již řečeno testovací případy vycházejí ze zadaných požadavků od zákazníka, které sami testují. Jedná se o jakýsi skript, podle kterého se tester musí držet, aby otestoval danou funkci. Je vhodné, aby byly veškeré požadavky znázorněny v testovacích případech. Výsledek testovacích případů je například ověření přihlášení uživatele, navigace po aplikaci nebo kontrola oprávnění uživatele při interakci s prvky webové aplikace. Po vytvoření všech klíčových testovacích případů může nastat první metoda testování.

3.6 Funkční testy (Functional testing)

Funkční testy jsou první testy v navrhovaném procesu testování. Funkční testy jsou klíčové pro chod celé aplikace, protože bez základních funkcí aplikace postrádá další testování smysl. Je tedy nutné pro další postup procesu, aby byla aplikace plně funkční. Tím se rozumí například správné odeslání požadavku při stisknutí tlačítka, nebo funkčnosti všech odkazů. Pokud se vyskytne chyba pří testování funkčnosti je potřeba chybu identifikovat, opravit a následně provést všechny testovací případy od znova abychom zachovali konzistenci aplikace. Tento proces je nejlepší z automatizovat například použitím selenium frameworku.

Další podstatnou součástí každého provedeného testu je evidence výsledků provedených testů. Je taky potřeba udržovat automatizované testy aktuální pro lepší chod procesu.

3.7 Testování rozhraní (Interface testing)

Po ukončení všech funkčních testů se vyhodnotí jejich výsledky. Pokud se neprojeví žádné chyby nastává fáze testování rozhraní. Testování rozhraní je důležité pro ověření plynulé komunikace mezi dvěma systémy. V principu jde o komunikaci například mezi webovou aplikací a databázovým serverem. Správná komunikace mezi systémy je klíčová pro chod celé aplikace. V kombinaci s funkčními testy máme

velkou jistotu v konzistenci celého systému a díky tomu můžeme přejít na další část procesu testování.

3.8 Testování použitelnosti (Usability testing)

Usability testování zkoumá, zda je daný software snadno použitelný pro zákazníka, jestli se v něm snadno naviguje, nebo jestli zobrazuje jen relevantní informace.
Při shromažďování požadavků od zákazníka nemusí být vždy jasné, jak má výsledný produkt vypadat nebo jak se v něm bude moct uživatel navigovat. Usability testování jsem zvolil jako první metodu testování po ověření stability systému. Důvodem je že v této fázi je možné prezentovat téměř hotový produkt, nebo aspoň hodně blízko podobné konečnému produktu. Právě v této fázi je vhodné komunikovat se zákazníkem o potencionálních změnách a upravit tak požadavky na celý systém, pokud je to žádoucí. V případě změny požadavků, je potřeba implementovat všechny požadované funkce a začít znova testovat funkční prvky systému pro udržení stability.

3.9 Testování kompatibility a výkonu (Compatibility and performance testing)

Po dokončení usability testů následuje testování kompatibility a výkonu. Jedná se o dvě rozdílné metody testování, které jsem sjednotil do dvou paralelních aktivit v procesu testování. Hlavním důvodem je ten že testování výkonu je v případě vývoje webových aplikací poměrně nenáročný. V komplexnějších systémech musíme dbát na hardware nároky, avšak u webových stránek jsou nároky relativně menší. To neznamená že je testování výkonu zbytečné. V dnešní době existují podpůrné frameworky které testují výkon webové stránky automaticky, a tím celý proces ulehčit.
Pro testování kompatibility je proces obdobný jako u testování výkonu. Jde v podstatě o to, aby webová aplikace byla dostupná i v jiných webových prohlížečích. A zpětně kompatibilní se staršími verzemi prohlížeče.
Na ukončení obou testovacích metod se podle dosáhnutých výsledků optimalizuje celá webová aplikace.

3.10 Bezpečnostní testy (Security testing)

V předposlední fázi procesu nastává testování bezpečnosti. Úkol metody testování bezpečnosti je jediný, a to zaručit, aby uživatel neměl přístup k citlivým datům, které mu nepatří. Testování bezpečnosti je poslední fáze testování, co se týče technických požadavků na aplikaci. Pokud je odhalena chyba v zabezpečení je potřeba jí co nejdříve opravit.

3.11 Ověřování požadavků a akceptační testování

V úplně poslední fázi nastává akceptační testování. Hlavní rolí akceptačního testování je zaručení spokojenosti zákazníka s konečným produktem. Pokud je zákazník nespokojen nebo aplikace nesplňuje některých z požadavků, je možné upravit požadavky a doimplementovat nedostatky. V případě že se implementují nové funkce nebo upravují ty stávající je potřeba provést revizi testovacích případů a projít celým procesem testování od znova.

3.12 Závěr návrhu

Tento proces je experimentální a slouží jako hrubý návrh procesu testování. Při návrhu postupu jsem se opíral o vědomosti zmíněné v této práci. Velkou většinu návrhu lze implementovat pomocí automatizačního frameworku pro webové stránky selenium.

3.13 Automatizované testování webových stránek pomocí Selenium frameworku

Selenium je open source framework určený pro vytváření automatizovaných funkčních testů webových stránek. Selenium se skládá ze tří komponent které implementují svoje dílčí funkce. Tyto komponenty jsou Selenium IDE, WebDriver a Selenium Grid. Selenium nabízí taky podporu pro různé programovací jazyky, jako například C#, Java, Python a Javascript.

Selenium IDE zkráceně pro Integrated Development Enviroment, je komponenta, která je určená pro nahrávání a zpětné zpuštění testů. Tyto testy nahrává uživatel přímo ve svém webovém prohlížeči, které může po nahrání editovat, ukládat a zpětně spouštět. Vytváření těchto testů je velice nenáročné, protože nejsou požadovány po uživateli žádné prvotní znalosti s programovacími jazyky.

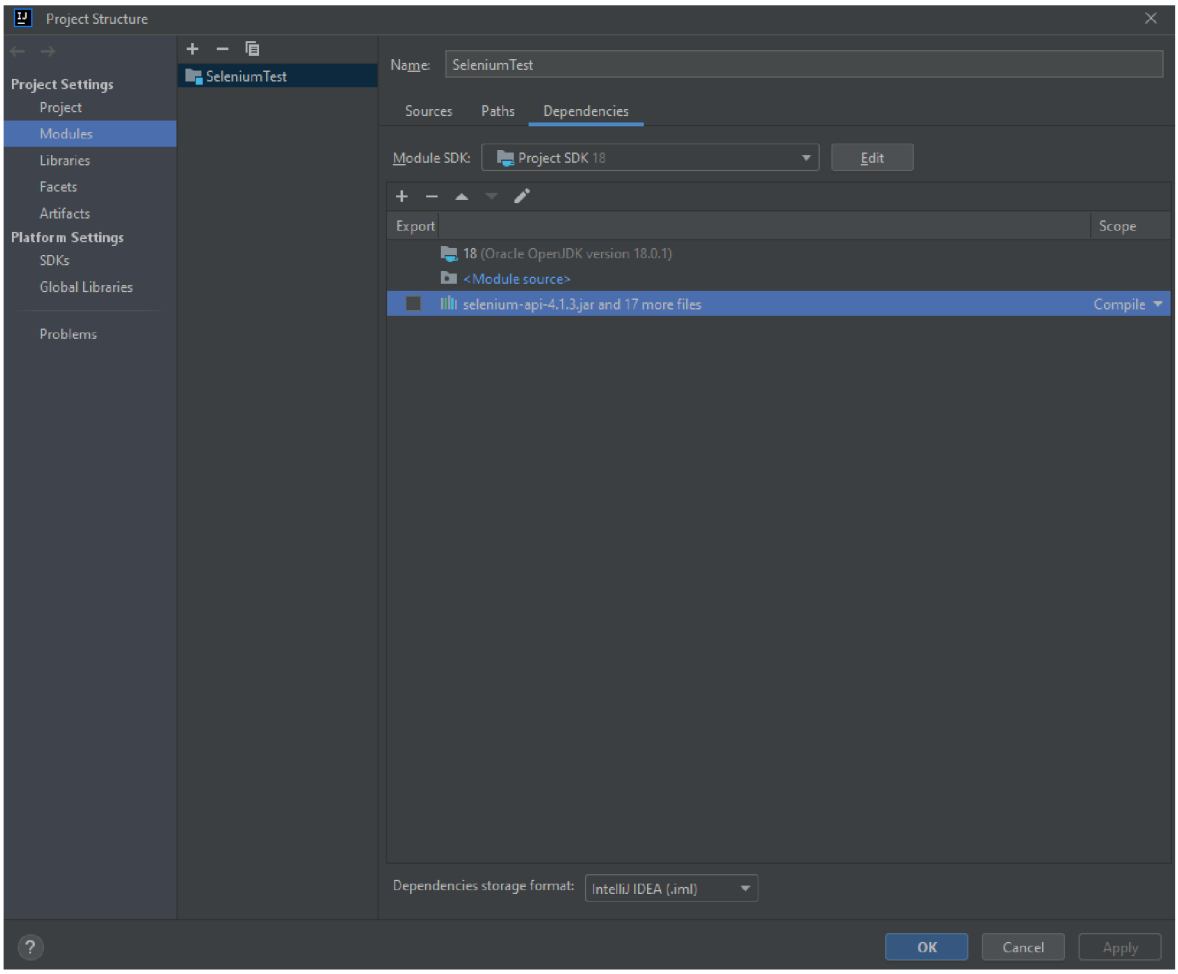
Selenium GRID je funkce která umožnuje spuštění testů paralelně. Testy bývají spouštěny na jiných zařízení, a i na jiných platformách. Můžeme tak využít pracovní sílu více zařízení k uskutečnění testů a testovat na různých platformách. Je tak možné například testovat funkce na jiných webových platformách v paralelním provedení navzájem nezávislých testů. Selenium GRID je velice užitečný například ve velkých firmách kde je vytvářeno mnoha testovacích případů které je potřeba pro efektivní práci rozdělit mezi více zařízení.

Selenium webdriver je samotná API knihovna, která se využívá k vytváření automatizovaných testů webových stránek. Dříve ještě existoval Selenium RC (Remote Control) který využíval komplexnější architektury kde bylo potřeba mít spuštěný server který fungoval jako prostředník mezi programovacím jazykem a samotným prohlížečem. V dnešní době se od selenium RC upustilo a webdriver kompletně převzal hlavní roli k tvorbě testů. Tyto testy potom simulují požadavky testů v prohlížeči.

3.13.1 Implementace Selenium testů

Abychom mohli začít vytvářet testy je zapotřebí zvolit programovací prostředí jako například intellj idea pro javu a propojit toto prostředí se selenium webdriver knihovnou. Jednotlivé testy jsou psány ve formě zdrojového kódu, kde specifikujeme, jaké akce má test s webovým prohlížečem provést.

V intellj idea importujeme nejdříve API knihovnu Selenium webdriveru. To je FILE>PROJECT STRUCTURE a dále v project structure okně MODULE>podsložka DEPENDENCIES a zde přídáme celou knihovnu webdriveru.



Obrázek 2: Import Selenium knihovny [Vlastní zpracování]

Po naimportování knihovny můžeme začít vytvářet první test. Založíme novou třídu v projektu a vytvoříme spouštěcí metodu která bude obsluhovat spuštěný kód. V této metodě budeme psát samotnou strukturu testů.

Pro demonstraci jsem si vybral jednoduchý test přihlášení studenta do studentského portálu UHK Oliva.

```
public static void main(String[] args) {


System.setProperty("webdriver.gecko.driver","E:/GeckoWebDriver/geckodr
iver.exe");

    //definování webového prohlížeče
    WebDriver driver = new FirefoxDriver();
    driver.get("https://oliva.uhk.cz/");

    //čekání na načtení elementu
    WebElement foo = new WebDriverWait(driver,
Duration.ofSeconds(3)).until(ExpectedConditions.elementToBeClickable(B
y.id("agree_button")));
    driver.findElement(By.id("agree_button")).click();

    //vyplnění přihlašovacích údajů
    driver.findElement(By.id("user_id")).sendKeys("vanalu1");
    driver.findElement(By.id("password")).sendKeys("UHK_heslo");

    //přihlášení uživatele
    driver.findElement(By.id("entry-login")).click();

    driver.close();
}
```

V tomto případě nejprve nastavíme prostředníka mezi selenium testy a samotným prohlížečem. V našem případě to je gecko driver. Ten se bude starat o zasílání inputů na simulovaný webový prohlížeč. Následně vybereme, na jakým webovým prohlížeči chceme test spouštět. V tomto konkrétním případě využíváme mozilla firefox.

3.13.2 Element lokátory

Při psaní kódu je potřeba specifikovat automatizačnímu nástroji, jak s jednotlivými elementy nakládat. Přesněji jaké elementy má využít pro dosažení výsledku testů. Těmto prvkům se říká lokátory, a jejich úkolem je selektovat elementy pro akce testů. K dispozici máme řadu lokátorů.

CSS ID (By.id) – Představuje nejlehčí způsob selektování elementu. Kde vyhledáváme element podle jeho CSS id atributu.

CSS class (By.className) – Podobné jako id lokátor jenom hledáme podle jména class atributu. Teoreticky každý element na webové stránce bude obsahovat aspoň jeden z těchto atributů. I tak může nastat situace kdy element nemá daný id atribut a class atribut sdílí s ostatními elementy, v tom případě je zapotřebí zvolit jinou metodu lokátoru.

CSS selektor (By.cssSelektor) – Jedná se o univerzální lokátor, který dokáže vybrat element podle potřebných atributů. Tyto atributy jsou například ID, class, potomek elementu nebo typ elementu. Například CSS selektor pro vybrání elementu, co má typ inputu vypadá následovně (=input) nebo element u kterého se v CSS vyskytuje celý řetězec („[name="jmeno"]"). Výhodou tohoto lokátoru je že můžeme vybrat téměř jakýkoli prvek dynamicky takže se nemusíme starat o to, jestli je daný element vykreslen u příslušného uživatele.

Name atribut (By.name) – Hledání elementu podle hodnoty name atributu. Tento lokátor vrátí první element, který obsahuje požadovanou hodnotu name atributu, pokud se na webové stránce vyskytuje více elementů se stejnou hodnotou name atributu. Výhoda i nevýhoda je že elementů který používají tento atribut není mnoho.

Xpath (By.xpath) – Pokud ani jeden lokátor nedokáže vybrat element efektivně. Je zapotřebí vyhledat element podle cesty. Vyhledávání podle cesty nabízí jednu velkou výhodu, a to hledat elementy dynamicky, což dává velkou flexibilitu v psaní testů. Nevýhodou je komplexnější identifikování cesty k webovému elementu. Můžeme hledat elementy podle absolutní nebo relativní cesty. S metodou absolutní cesty je třeba specifikovat celou cestu k elementu v závislosti celé webové stránky. Příklad absolutní cesty je například: /html//div/div/a/img, porovnání s relativní cestou, kde tento zápis vypadá takto: //img[@alt='test'].

Link text (By.linkText) – Tento lokátor vyhledává přesnou shodu textu u hypertextových odkazů. Efektivní vyhledávání, pokud chceme vyhledávat mezi hypertextovými odkazy.

Partial link text (By.partialLinkText) – Podobně jako předchozí lokátor vyhledává hypertextový odkazy. Avšak není potřeba zadávat úplný text odkazu. Riskujeme taky nalezení více elementů.

HTML tag name (By.tagName) – Poslední lokátor vyhledává podle názvu HTML tag. Můžeme tak vybírat mezi elementama jako třeba odkaz <a> nebo nadpis prvního řádu <h1>. Je třeba být opatrný při vybírání HTML tagu a zda je tento tag ojedinělí.

3.13.3 Finders

Vyhledávače slouží k identifikování elementů na webových stránkách. Právě díky těmto vyhledávačům se můžeme rozhodnout jaký lokátor použijeme pro efektivní nalezení potřebných elementů. Nejjednodušší způsob vyhledávání elementu je přímo v prohlížeči. Vyhledání elementu v DOM (Document Object Model), můžeme zpřístupnit pravým stisknutím myši na webové stránce a zvolení možnosti „prozkoumat prvek“. V novém okně můžeme prohlížet jednotlivé elementy a jejich atributy.

3.13.4 Interakce

Selenium nám dovoluje do jisté míry interagovat s vybranými elementy. Díky těmto interakcím dokážeme manipulovat s webovou stránkou v simulovaném prostředí. A dosáhnout tak simulace testů. Selenium umožnuje pouze pět akcí které můžeme s elementem provést.

Kliknutí – Kliknutí je primitivní operace, kterou lze vyvolat pomocí metody click(). Selenium tuto metodu provede nad vybraným elementem pouze pokud není nějak překrytý. Samotné kliknutí se aplikuje na střed elementu to může s jistými elementy být problematické a je třeba mít tuto vlastnost na vědomí.

Poslání vstupu – Posílání vstupu ať už se jedná o zasílání řetězců nebo poslání jedné klávesy, je další základní interakce s elementem. Tuto interakci lze vyvolat u

prvku pouze pokud je element editovatelného typu, to znamená třeba input element typu text, nebo element který má nastavený atribut „content-editable“. Vstup zasíláme podle metody sendKeys() kde do závorek vkládáme posílaný řetězec nebo specifickou klávesu.

Vyčistění pole – Příkaz indikovaný metodou clear() resetuje jakýkoliv obsah elementu. Operace vyčistění lze použít na element který má stejné podmínky jako u posílání vstupu a je nutné, aby element byl typu „resettable“.

Operaci submit – Submit operace se dříve používala jako poslání form elementu na webové stránce. Od Selenium 4 a více se od používání operace submit upustilo, proto se její používání nedoporučuje.

Operace select – Select je operace určená pro interakci dropdown a list elementů, který mají HTML tag <select>. K použití této operace je zapotřebí importovat pomocnou třídu ze Selenium knihovny.

3.13.5 Operace s prohlížečem

Selenium dovoluje testerovi provádět určité operace na webovém prohlížeči. Tyto operace představují možnosti s navigací mezi více webovými stránkami nebo získat informace o webové stránce.

Nastavení webového prohlížeče – Toho docílíme pomocí příkazu pro založení nové instance driveru. Dlouhý zápis tohoto příkazu je WebDriver driver = new FirefoxDriver(). Tím založíme novou instanci prohlížeče pro platformu firefox, je možné zvolit jinou platformu dle požadavků.

Informace o prohlížeči – Můžeme získat informace o titulní stránce pomocí příkazu getTitle(), nebo získat aktuální URL adresu webové stránky pomocí příkazu

getCurrentUrl(). Obě tyto operace vyvoláváme nad instancí driveru webového prohlížeče.

Základní navigace – První akci, kterou je potřeba udělat při vytváření nového testu, je specifikovat kterou webovou stránku chceme testovat. Toho lze dosáhnout pomocí příkazu get() kde do závorek zadáme adresu webové stránky. Avšak nemusíme zůstat pouze na jedné stránce, ale procházet mezi více stránkami. To nám Selenium umožnuje pomocí příkazu navigate().to() který po specifikaci URL adresy převede na danou webovou stránku. Pokud je zapotřebí přejít na předchozí nebo následující stránku to je možné provést příkazem navigate().back() pro předchozí stránku a navigate().forward() pro následující webovou stránku. Další akci kterou lze vyvolat je operace znovu načtení aktuální stránky, tuto operaci můžeme vyvolat příkazem navigate().refresh().

Alerty – V jistých okolností je potřeba se vypořádat s webovými alerty. I ty nám Selenium dovoluje manipulovat. Máme tři typy alertu, normální alert, potvrzovací alert a prompt alert. Normální alerty jsou vyskakovací okna, které sebou nesou text, většinou upozorňovací, a tlačítko kterým potvrdíme že alert zavíráme. Takové alerty lze potvrdit příkazem alert.accept(). Dalším druhem alertu je potvrzovací alert. Tento alert obsahuje kromě varovného textu a tlačítka ještě další tlačítko kterým můžeme zamítnout tento alert. Tedy můžeme vybrat, jestli alert potvrdíme nebo zamítneme, zamítnutí alertu lze dosáhnout pomocí příkazu alert.dismiss(). Poslední druh alertu je promt alert, ten se podobá potvrzovacímu alertu jenom s tím rozdílem že prompt alert vsobě obshaju textové pole, do které můžeme poslat znakové řetězce. Toho můžeme docílit podobně jako u elementů pomocí příkazem alert.sendKeys(), poté můžeme alert potvrzovacím příkazem schválit.

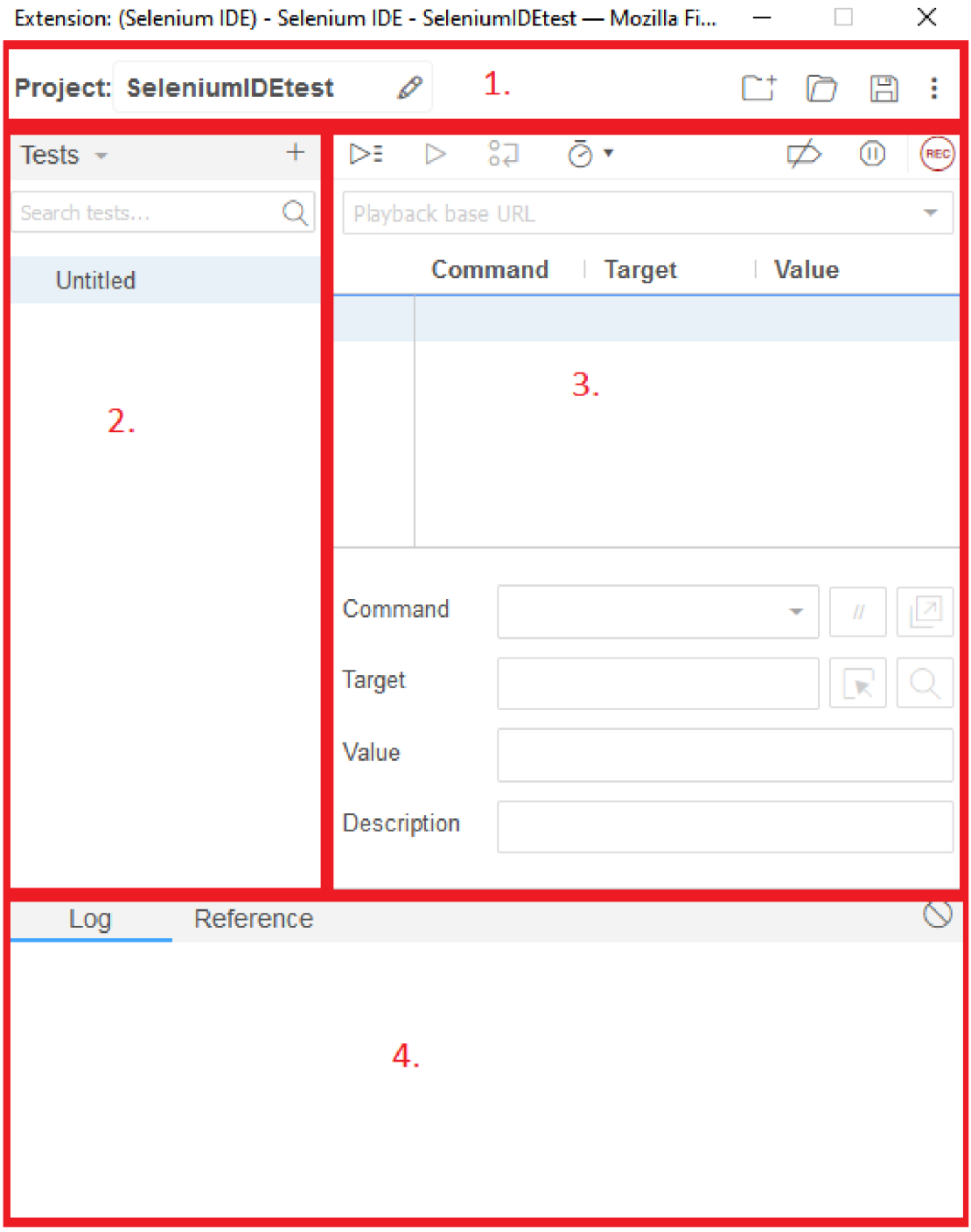
Cookies – Cookies jsou malé kousky dat které webová stránka ukládá lokálně na zařízení uživatele. Většinou jejich využití slouží k uchování informací uživatele pro snazší identifikování přes více návštěv stránky. Tyto data jsou typicky uchovány v JSON formátu. Selenium umožnuje manipulaci s těmito daty. Je možné přidat cookie data příkazem manage().addCookie() a poskytnutím hodnot ve formátu

„key“, „value“. Je možné získat data specifického cookie podle shodného jména. Tyto data získáme příkazem manage().getCookieNamed() a poskytnutí jména požadovaného cookie. Je možné získat hodnoty všech cookies najednou pomocí příkazu manage().getCookies(). Jako poslední akci, kterou můžeme s cookie soubory provést je jejich mazání. Můžeme mazat konkrétní nebo všechny cookies. Příkaz pro mazaní je manage().deleteCookie() a pro smazání všech coockie dat manage().deleteAllCookies().

3.13.6 Selenium IDE

Implementace Selenium IDE je velmi jednoduchá, stačí pouze přidat a naistalovat Selenium IDE rozšíření přímo do webového prohlížeče. Veškeré ovládání je realizováno přímo v samotném rozšíření, není potřeba žádných podpůrných knihoven nebo aplikací. Po přidání a naistalování rozšíření stačí pouze restartovat webový prohlížeč, po znovu načtení prohlížeče je možné spustit Selenium IDE rozšíření.

Po otevření rozšíření máme na výběr možnosti. Založení nového projetu, otevření již existujícího projektu nebo ukončení Selenium IDE rozšíření. Po vytvoření nového projektu je zpřístupněna samotná aplikace.



Obrázek 3: Uživatelské rozhraní Selenium IDE [Vlastní zpracování]

Uživatelské rozhraní aplikace je rozdělené do čtyřech částí:

1. První část aplikace identifikuje, v jakém projektu se nacházíme a umožnuje uživateli provádět základní operace s projektem. To zahrnuje založení

nového projektu otevření existujícího projektu a uložení aktuálně otevřeného projektu.

2. Levá část aplikace je určená pro procházení dílčích testů. V každém projektu je možné mít najednou více spustitelných testů. To je i podporováno i dalšíma funkcema aplikace. Při spuštění více testů máme přehled o tom, které testy byli úspěšné a které selhali.
3. Pravá část aplikace nabízí samotné ovládání testů. Jednotlivé funkce zleva doprava je spuštění všech testů v projektu, spuštění jediného testu, debugování aktuálního testu, nastavení rychlosti přehrávání testu, pozastavení nahrávání a spuštění nahrávání testu. O řádek níže je vložení URL adresy testované webové stránky. Při nahrávání testu jsou jednotlivé operace zapisovány do posloupnosti testu. Každá operace je nový řádek testu, který obsahuje informace o nahraném příkazu, o cíli příkazu a posílanou hodnotu. Při debugování testu se právě postupuje po jednotlivých řádcích a kontroluje se jejich vliv na test.
4. Dolní část aplikace je určená pro log aplikace, v logu je záznam o všech provedených operací v prováděném testu a v případě že se operace nepovedla log zobrazí chybovou hlášku spolu s výpisem proč test selhal.

4 Závěry a doporučení

Téma testování softwaru bylo pro mě v době výběru tématu bakalářské práce úplně nové. Nicméně bral jsem to jako příležitost si rozvinou svoje vědomosti a zjistit něco o testování softwaru. Proces testování mě přišel zajímavé a po vypracování této práce mě testování softwaru do jisté míry i zaujalo. Ze začátku jsem měl naivní představu o procesu testování jako „Vyzkoušej to, ať to funguje.“ A nemohl jsem se mýlit více. Celý proces je komplexní s pevně danými pravidly a spolu s nesčetným počtem metod pro testování softwaru dává tématu neuvěřitelnou hloubku. Každý software je unikátní a podle toho potřebuje mít proces testování udělaný na míru. Právě to mě vedlo k návrhu procesu testování softwaru.

Při návrhu jsem se opíral o získané vědomosti při vypracování teoretické části, a na jejich základě jsem se zamyslel jaký postup by byl ideální pro kvalitní vývoj webové aplikace. Někomu se může zdát, že navrhovat komplexní proces jako testování softwaru je poněkud ambiciózní úkol. Avšak i hrubý návrh může podnítit k vývoji něčeho většího. Proto bych doporučil i veterány oboru k zvážení mého návrh procesu testování a na základě zpětné vazby celý proces vylepšit. Proces je možné implementovat i jinými automatizačními nástroji, avšak zmíněný Selenium nástroj silně doporučuji. I pro někoho, kdo nemá s tématem automatizace testů žádné zkušenosti je tento nástroj snadný na použití a implementaci. V praxi je Selenium velmi oblíbené s mnoha pomocnými návody a velkou komunitou ze kterých lze čerpat.

Testování softwaru je široký obor s mnoha metody, tak mnoho že je nereálné popsat v této práci všechny. Cílem práce je představit obor testování a vysvětlit základní pojmy. Je důležité si uvědomit že testování není osamocený proces, ale představuje doplněk k vývoji softwaru. Dokud se proces vývoje softwaru dále vylepšuje bude potřeba vylepšit i příslušné testovací metody. Dohromady tak tvoří koloběh života softwaru.

5 Seznam použité literatury

1. **O'REGAN, Gerard**. *Introduction to Software Quality* [online]. Cham: Springer International Publishing, 2014 [cit. 2022-03-02]. Undergraduate Topics in Computer Science. ISBN 978-3-319-06106-1. Dostupné z: https://link.springer.com/content/pdf/10.1007%2F978-3-319-06106-1.pdf doi: 10.1007/978-3-319-06106-1
2. Co je testování softwaru. *Kitner* [online]. [cit. 2022-03-02]. Dostupné z: https://kitner.cz/testovani_softwaru/co-je-testovani-softwaru/
3. **HAMILTON, Thomas**. Unit Testing Tutorial: What is, Types, Tools & Test EXAMPLE. *Guru99* [online]. 08.10.2021 [cit. 2022-03-02]. Dostupné z: https://www.guru99.com/unit-testing-guide.html
4. **OLSEN, Klaus (chair), Meile POSTHUMA a Stephanie ULRICH**. Foundation Level Syllabus. *ISTQB* [online]. 2021 [cit. 2022-03-05]. Dostupné z: https://www.istqb.org/certifications/certified-tester-foundation-level
5. **HAMILTON, Thomas**. Integration Testing: What is, Types, Top Down & Bottom Up Example. *Guru99* [online]. 08.10.2021 [cit. 2022-03-02]. Dostupné z: https://www.guru99.com/integration-testing.html
6. **HAMILTON, Thomas**. What is System Testing? Types & Definition with Example. *Guru99* [online]. 26.02.2022 [cit. 2022-03-02]. Dostupné z: https://www.guru99.com/system-testing.html
7. **PEHAM, Thomas**. 5 Types Of User Acceptance Testing. *Usersnap* [online]. 2022 [cit. 2022-03-10]. Dostupné z: https://usersnap.com/blog/types-user-acceptance-tests-frameworks/
8. **ROUDENSKÝ, Petr**. *Kvalita softwaru: teorie a praxe*. Prostějov: Computer Media, 2016. ISBN 978-80-7402-294-4.
9. **BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ**. *Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu*. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.
10. **PEKAŘ, Lukáš**. Párové programování. *Bonsai* [online]. 2018, 13. 05. 2018 [cit. 2022-03-11]. Dostupné z: https://bonsai-development.cz/clanek/parove-programovani
11. **HAMILTON, Thomas**. What is Dynamic Testing? Types, Techniques & Example. *Guru99* [online]. 26.02.2022 [cit. 2022-03-12]. Dostupné z: https://www.guru99.com/dynamic-testing.html
12. **ROUDENSKÝ, Petr a Anna HAVLÍČKOVÁ**. *Řízení kvality softwaru: průvodce testováním*. Brno: Computer Press, 2013. ISBN 978-80-251-3816-8.
13. **SMITH, Andrew**. Difference Between Black-Box, White-Box, and Grey-Box Testing. *Dzone* [online]. 15. 5. 2020 [cit. 2022-03-13]. Dostupné z: https://dzone.com/articles/difference-between-black-box-white-box-and-grey-bo
14. **SYSTEMS, Apica**. Automated vs Manual Testing: Which Should You Use, and When?. *Apica* [online]. c2022, 6. 7. 2017 [cit. 2022-03-22]. Dostupné z: https://www.apica.io/difference-between-automated-manual-testing/
15. Manual Testing. *Javatpoint* [online]. c2011-2021 [cit. 2022-03-22]. Dostupné z: https://www.javatpoint.com/manual-testing

16. **GUPTA, Divyanshu**. Software Development Life Cycle. *Geeks for geeks* [online]. 5. 7. 2021 [cit. 2022-03-25]. Dostupné z: https://www.geeksforgeeks.org/software-development-life-cycle-sdlc/
17. **JEVTIC, Goran**. What is SDLC? Phases of Software Development, Models, & Best Practices. *PhoenixNAP* [online]. c2022, 15. 5. 2019 [cit. 2022-03-25]. Dostupné z: https://phoenixnap.com/blog/software-development-life-cycle
18. **MYSLÍN, Josef a Anna HAVLÍČKOVÁ**. *Scrum: průvodce agilním vývojem softwaru*. Brno: Computer Press, 2016. ISBN 978-80-251-4650-7.
19. **PAL, Sayan**. Software Engineering | Spiral Model. *Geeks for geeks* [online]. 9. 2. 2022 [cit. 2022-03-26]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-spiral-model/?ref=lbp
20. Spiral Model – What is SDLC Spiral Model. *Software testing help* [online]. c2022, 3. 4. 2022 [cit. 2022-04-10]. Dostupné z: https://www.softwaretestinghelp.com/spiral-model-what-is-sdlc-spiral-model/
21. **NAMIKO**. Difference between V-model and W-model | Software Testing. *Shift asia* [online]. 8. 7. 2020 [cit. 2022-04-01]. Dostupné z: https://shiftasia.com/column/difference-between-v-model-and-w-model-in-software-testing/
22. Business Process Model and Notation. *Omg* [online]. c2022, Prosinec 2010 [cit. 2022-04-27]. Dostupné z: http://www.omg.org/spec/BPMN/2.0/PDF
23. What is Business Process Modeling Notation. *Lucid chart* [online]. c2022 [cit. 2022-04-27]. Dostupné z: https://www.lucidchart.com/pages/bpmn#section_0
24. **RUNGTA, Krishna**. What is Selenium? Introduction to Selenium Automation Testing. *Guru99* [online]. 11. 12. 2021 [cit. 2022-04-15]. Dostupné z: https://www.guru99.com/introduction-to-selenium.html
25. What is Selenium Grid. *Javatpoint* [online]. c2011-2021 [cit. 2022-04-15]. Dostupné z: https://www.javatpoint.com/selenium-grid
26. **UNADKAT, Jash**. Getting Started with Selenium IDE. *BrowserStack* [online]. c2011-2022, 26. 3. 2021 [cit. 2022-04-16]. Dostupné z: https://www.browserstack.com/guide/what-is-selenium-ide
27. Software Engineering | SDLC V-Model. *Geeks for geeks* [online]. 3. 3. 2022 [cit. 2022-04-10]. Dostupné z: https://www.geeksforgeeks.org/software-engineering-sdlc-v-model/

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání bakalářské práce

Autor:	**Lukáš Váňa**
Studium:	I1800241
Studijní program:	B1802 Aplikovaná informatika
Studijní obor:	Aplikovaná informatika
Název bakalářské práce:	**Metody testování softwaru**
Název bakalářské práce AJ:	Software testing methods

Cíl, metody, literatura, předpoklady:

Analýza metod testování softwaru, jejich porovnání a aplikace.

- Roudenský, P. (2016). *Kvalita softwaru: teorie a praxe . Computer Media.*
- *Bureš, M., Renda, M., {\& Doležel, M. (2016). Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Grada Publishing as.*

Garantující pracoviště:	Katedra informačních technologií, Fakulta informatiky a managementu
Vedoucí práce:	doc. Ing. Hana Tomášková, Ph.D.
Datum zadání závěrečné práce:	15.10.2021