Bakalářská práce

České
vysoké
učení technické
v Praze

F3

Fakulta elektrotechnická
Katedra počítačů

Systém pro ověřování účinnosti testovacích scénářů pro software

Aneta Volfová

Vedoucí práce: Ing. Miroslav Bureš, Ph.D.
Obor: Softwarové inženýrství a technologie
Květen 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Volfová** Jméno: **Aneta** Osobní číslo: **435280**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Systém pro ověřování účinnosti testovacích scénářů pro software

Název bakalářské práce anglicky:

System for verification of the software test cases effectiveness

Pokyny pro vypracování:

Navrhněte a implementujte systém pro ověřování účinnosti vytvořených testovacích scénářů. Systém bude založen na webové open-source aplikaci. V systému bude možné zapínat umělé chyby, jejichž aktivace uživatelem bude reportována pomocí speciálního logu. Pro tuto část bude možné využít výstup existujícího studentského projektu a dále jej upravit. Další částí systému bude sada znovupoužitelných objektů pro automatizované testy založené na platformě Selenium, které budou automaticky sestavovány do výsledného testu na základě parametrizované specifikace testu generované z nástroje Oxygen.

Seznam doporučené literatury:

COLLIN, Mark. Mastering Selenium WebDriver. Packt Publishing Ltd, 2015.
KOOMEN, Tim, et al. TMap next: for result-driven testing. Uitgeverij kleine Uil, 2013.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Miroslav Bureš, Ph.D., Software Engineering and Networking FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **16.01.2018** Termín odevzdání bakalářské práce: _____

Platnost zadání bakalářské práce: **30.09.2019**

Ing. Miroslav Bureš, Ph.D. podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	prof. Ing. Pavel Ripka, CSc. podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Studentka bere na vědomí, že je povinna vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____ _____

Datum převzetí zadání Podpis studentky

Poděkování

Chtěla bych poděkovat především svému vedoucímu semestrálního projektu a bakalářské práce Ing. Miroslavu Burešovi, Ph.D. za odborné vedení a za pomoc a rady při zpracování tohoto semestrálního projektu. Mé díky patří také rodině a příteli, který mě během mé práce podporoval a dodával mi motivaci k práci.

Prohlášení

Prohlašuji, že jsem předloženou práci vypracovala samostatně a že jsem uvedla veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací.

V Praze, 25. května 2018

Abstrakt

Obsahem práce je popis návrhu a implementace Systému pro ověřování účinnosti testovacích scénářů pro software v rámci semestrálního projektu a navazující bakalářské práce. V rámci návrhu jsou podle stanovených požadavků na systém analyzovány nástroje (technologie) vhodné pro implementaci tohoto systému a provedena rešerše existujících řešení. Analýzu doplňuje shrnutí zvolených nástrojů včetně zdůvodnění jejich výběru a srovnání s obdobnými technologiemi, které se v praxi používají nejčastěji. Práce se dále zabývá popisem zmapovaných procesů ve zvolené testovací aplikaci. Tato bakalářská práce pokračuje přehledovým modelem celého systému (tj. systému skládajícího se ze Systému pro ověřování účinnosti testovacích scénářů pro software a dalších aplikací, které byly v rámci tohoto projektu využity a popsány). Následující část je zaměřena na implementaci návrhu s ukázkami kódu a na testování systému.

Klíčová slova: automatizované testy, znovupoužitelné objekty, testovací scénáře

Vedoucí práce: Ing. Miroslav Bureš, Ph.D.

Abstract

The content of the thesis is description of the design and implementation of the System for verification of the software test cases effectiveness within the semester project and the following bachelor thesis. Within the design, the tools (technologies) appropriate for the implementation of the system are analyzed according to the specified system requirements. The analysis complements the summary of the selected tools, including the rationale for their selection and comparison with similar technologies that are used in practice the most frequently. The thesis also deals with description of mapped processes in selected test application. This bachelor thesis continues with an overview model of the whole system (i.e. a system consisting of System for verification of the software test cases effectiveness and other applications used and described within this project). The following section focuses on implementing a design with code samples and system testing.

Keywords: automated tests, reusable objects, test cases

Title translation: System for verification of the software test cases effectiveness

Obsah

Obrázky

Tabulky

Kapitola 1
Úvod

Hlavním cílem této a navazující bakalářské práce je navrhnout a implementovat Systém pro ověřování účinnosti testovacích scénářů pro software, který splňuje všechny požadavky uvedené v kapitole Analýza požadavků 3.1. Tento systém přijímá na vstupu vytvořené testovací scénáře pomocí aplikace Oxygen[1], ze kterých sestaví, ze sady znovupoužitelných objektů v Selenium[2], automatizované testy, kterými je otestována zvolená aplikace.

Systém pro ověřování účinnosti testovacích scénářů pro software je navrhován z důvodu usnadnění vytváření (sestavování) automatizovaných testů, jejichž tvorba je odvozena od testovacích scénářů. Samotné sestavování testů je převážně opakující se manuální práce, kterou je možné automatizovat.

Pro potřeby vypracování celé práce byly zajištěny potřebné informace o použitých technologiích, vyhledána a nastudována odborná literatura, s ohledem na zjištěná fakta navrhnuto samotné řešení a volba technologií a nakonec provedena implementace aplikace včetně testování. Pro názorné vysvětlení studované problematiky jsou přiloženy obrázky a schémata.

Výstupem bakalářské práce jsou kromě toho i diagramy procesů (založení projektu, reportování chyby, přidání uživatele, atp.) identifikované v Mantis BT, to znamená v aplikaci pro reportování a správu chyb nalezených při testování. Dalším výstupem jsou zvolené technologie na základě analýzy a také schéma architektury a komunikace aplikací (resp. jaké soubory aplikace přijímají a jaké vracejí) včetně popisu implementace projektu. Schéma je pro lepší pochopení popsáno a vysvětleno. Mezi výsledky práce patří také vypracování rešerše existujících řešení a v neposlední řadě vyhotovení Systému pro ověřování účinnosti testovacích scénářů pro software.

S ohledem na to, že se již téměř čtyři roky zajímám o problematiku testování, zvolila jsem si shora uvedené téma. Považuji za přínosné možnost využití svých zkušeností, jež jsem načerpala z praxe, absolvováním trainee programu, a dále prací na pozicích testera a test inženýra juniora v několika společnostech.

Testování považuji za neoddělitelnou součást tvorby každé aplikace. Klíčovou úlohu plní hlavně při vývoji aplikací, jejichž nefunkčnost by mohla mít pro okolí negativní až fatální následky. Příkladem jsou aplikace ve finanční

[1] Oxygen – viz kapitola 3.2.2 Aplikace pro kreslení modelů a generování test. scénářů

[2] Skripty v Selenium - viz kapitola 3.2.3 Nástroj pro sestavování automatizovaných testů

sféře, v průmyslu atd.

Následující kapitoly jsou rozděleny do sedmi celků – na část rešeršní, analytickou, návrhovou, implementační, část pojednávající o zvažovaných řešeních projektu a část popisující instalaci projektu a také testování. Rešeršní část je rozborem vědeckých článků a publikací s tématikou testování. Analytický oddíl práce je věnována analýze požadavků systému a shrnutí použitých nástrojů včetně zdůvodnění jejich výběru a srovnání s obdobnými nástroji, které se v praxi používají nejčastěji. Návrhový úsek se zabývá popisem zmapovaných procesů ve zvolené aplikaci pro testování a ukázkou přehledového modelu systému. Vše je doplněno obrázky s popisem. Implementační celek je zaměřen na popis vytváření projektu spolu s demonstrací kódu a vysvětluje, jak celý systém funguje. Kapitola Zvažované varianty řešení se zaobírá posuzováním možných způsobů tvorby projektu a objasňuje, proč byly tyto varianty zamítnuty. Poslední testovací část prezentuje jednoduché scénáře, kterými byl celý systém ověřován.

Kapitola 2

Rešerše existujících řešení

Problematika testování (převážně automatizovaného testování) je v současné době stále podceňována a přehlížena. Automatizace testů má přitom nesporné výhody - například úsporu prostředků, zvýšení kvality testovaného produktu a zamezení chybovosti z hlediska lidského faktoru.

Pokud se zaměříme na odborné články a vědecké publikace pojednávající o automatizovaném testování, zjistíme, že je lze rozdělit do tří obecných kategorií:

1. publikace zabývající se problémy v oblasti automatizovaného testování, jako jsou údržba testů a jejich použitelnost
2. články zaměřené na optimalizaci automatických testů
3. texty přinášející alternativní způsoby, jak snížit potřebu údržby testů

Tyto tři kategorie jsou dále rozpracovány.

Poslední částí této rešerše je analýza řešení zabývající se obdobnou tématikou jako tato bakalářská práce a závěrečné shrnutí prováděné rešerše.

2.1 Problémy v automatizovaném testování

Prvním z článků, který se pro tuto kategorii nabízí, je od autorů *Rafi, D.M. a kol.* [10]. Dokument přibližuje výhody a omezení automatizovaného testování založeného na studii získané spoluprácí s profesionálními testery a inženýry. Na základě analýzy potvrzují autoři dokumentu, že výhody spočívají ve znovupoužitelnosti a úspoře financí. Na závěr zmiňují, že téměř polovina respondentů sdělila, že jsou jimi využívané testovací nástroje nedostatečné.

Naopak *Kasurinen a kol.* [11] se zaměřili na nevýhody a limitace automatizovaných testů. Tito finští akademici se však názorově ztotožňují s předchozím článkem, že používané testovací nástroje nedosahují požadovaných vlastností a že pro testování nejsou vynaloženy potřebné finanční zdroje.

Další zpráva obdobné tématiky od *Bernera a kol.* [12] pojednává o zjištěních učiněných při automatizovaném testování. Práce se zmiňuje o zkušenostech získaných během testování a shrnuje doporučení pro budoucí generace.

Persson a kol. [13] řeší problematiku, jak se vyvarovat klasických chyb při testování, které vedou k neúspěchu projektu a zvažují, zda lze předejít neúspěchu využitím projektového řízení rizik. Jako důkaz předkládají své dva projekty, jejichž vhodným řízením autoři dospěli do požadovaného cíle.

S mírně odlišným řešením automatizace testů přicházejí autoři *Alégroth a kol.* [14], kteří v roce 2014 navrhli GUI[1] testovací nástroj pracující s rozpoznáváním obrázků nahrazující manuální testy uživatelů. Přestože přicházejí se zajímavým nápadem, neopomínají uvést, že mají podobně založená řešení stále ještě svá omezení.

Mezi klasické problémy automatizovaného testování lze zařadit přeceňování těchto testů. Veškeré části systému nemohou být otestovány bez uživatele, protože „počítači chybí rozum a také cit“ pro estetiku. Například text od *Fewstera* [15] zmiňuje dokonce prvotní zklamání z testů, jejichž efektivita narůstá až s vyšším počtem jejich spuštění.

V neposlední řadě je tu publikace taktéž spadající do této kategorie - zabývající se situací automatizovaných testů v Čechách a na Slovensku. Autor zde shrnuje aktuální trendy a zobecňuje výsledky pro Střední Evropu. Článek autora *Bureše* [16] je také analýzou dat sesbíraných od specialistů v oboru testování a přibližuje ekonomickou stránku automatizovaného testování.

2.2 Optimalizace automatických testů

Problematika optimalizace automatických testů není zdaleka tak známým tématem, jak by se dalo očekávat. Za účelem rešerše byly nalezeny dva články, které vhodně reprezentují tuto kategorii.

Filipský a kol. [17] se soustředí na tvorbu „chytrých” a znovupoužitelných testů, které vznikají z nahrávaných dat. Tato data jsou dále zpracovávána a na základě speciálně vyvinutých algoritmů identifikují posloupnosti kroků, které se opakují. Díky tomu nalézaní znovupoužitelných částí optimalizují testy a zamezují tvorbě duplicit.

V jiné studii [18] jsou získávána data nahráváním testů a ve své podstatě autoři nehledají jen posloupnosti kroků, ale přímo celé objekty, které by byly znovupoužitelné. Lze tedy říci, že tento článek pracuje s myšlenkou hledání a využívání Page objektů[2].

[1] GUI - grafické uživatelské rozhraní
[2] Vysvětleno v kapitole 5.1.1

2.3 Alternativní způsoby snížení potřeby údržby testů

Existuje množství způsobů, kterými lze docílit minimalizace finančních nároků na údržbu testů a zásahů do testů samotných - patří sem:

- sledování změn v projektu
- optimální architektura testů
- volba metrik

Další studie [19] se zabývá tématem snížení potřeby údržby založené na sledování změn v projektu. Cílem autora je automaticky identifikovat změny v projektu a s využitím naskenovaných změn a ručního vstupu tyto testy systematicky upravovat.

Bures [20] využívá také detekci změn, ale zároveň přichází s myšlenkou vytvoření optimální architektury testů obsahující znovupoužitelné objekty. Navrhovaný systém prochází jednotlivé elementy v testech a skládá je do speciální architektury neobsahující duplicity tak, aby byly testy znovupoužitelné.

Další dva články autora *Bureše* [21] a [22] definují a kategorizují metriky sloužící ke snížení ceny a potřeby údržby testů a metriky pro určování ceny vývoje a údržby testů. Oba texty se shodují na tom, že se kvalita testů odvíjí od kvality kódu front-endu[3] - jinak řečeno, pokud je například webová aplikace napsána dle standardů W3C[4], tak je znatelně jednodušší aplikaci otestovat a vytvořit stabilní a snadno udržovatelné testy. Tímto způsobem lze ušetřit čas i peníze a získat důkladně a efektivně otestovaný systém.

2.4 Obdobné řešení (framework)

Podobný framework, jako je vyvíjen v mé práci, popsali *Bures a kol.* [23]. Tato publikace se zabývá vytvořením frameworku založeném na Selenium WebDriver a složeném ze třech vrstev - z web-elementů, Page objektů a testovacích kroků. Rozdílem oproti zmiňovanému textu je výskyt web-elementů, které jsou v bakalářské práci zakomponovány přímo do Page objektů a fakt, že testovací kroky nejsou automatizovaně sestavovány.

[3] Front-end - stránka, kterou vidí běžný uživatel
[4] World Wide Web Consortium - konsorcium vyvíjející standardy pro web

2.5 Shrnutí rešerše

Při procházení výše zmiňovaných článků a jejich posuzování s ohledem na tématiku testování lze konstatovat, že většina autorů pouze identifikuje znovupoužitelné objekty, ale jen hrstka z nich vyloženě generuje a automatizuje testy. Jejich prací je spíše vytvářet objekty a poté ručně psát testy.

Na závěr tedy lze říci, že v současné době není dostupné duplicitní řešení, které by celkově pokrývalo stejnou oblast jako je tomu v Systému pro ověřování účinnosti testovacích scénářů pro software. Tento projekt bude dále využit ve výzkumné práci ověřující a dokazující funkčnost generování testovacích scénářů z Oxygen s efektivním pokrytím (libovolné aplikace) testy.

Kapitola 3
Analytická část

Část analytická je zaměřena na rozbor požadavků z analýzy a dále blíže rozvádí a srovnává zvažované technologie využitelné v rámci projektu.

3.1 Analýza požadavků

V této kapitole jsou popsány a rozvedeny požadavky na systém, které byly identifikovány během práce na semestrálním projektu, ze zadání a z konzultací s vedoucím práce Ing. Miroslavem Burešem, Ph.D.

Požadavky jsou rozděleny do dvou částí – na funkční (tj. co všechno musí systém umět, a které funkce budou podporovány) a nefunkční (tj. požadavky na design, výkonnost, použité technologie a kvalitu).

3.1.1 Funkční požadavky

Mezi funkční požadavky, které byly analyzovány, patří:

- Ověřování účinnosti testovacích scénářů – Systém otestuje zvolenou webovou aplikaci (Mantis BT) s využitím vytvořených testovacích scénářů.
- Umožnit zapínání umělých chyb – V systému bude možné libovolně zapínat umělé chyby konkrétním uživatelům aplikace.
- Aktivace umělých chyb bude logována – Tento log bude sloužit k reportování aktivace umělých chyb v aplikaci.
- Automaticky sestavovat testy – Automatizované testy budou automaticky sestavovány a generovány ze sady znovupoužitelných objektů z vytvořených testovacích scénářů.

3.1.2 Nefunkční požadavky

Požadavky, které patří mezi nefunkční, jsou:

- Systém bude založen na webové open-source[1] aplikaci – Jako předmět testování v rámci tohoto projektu bude vybrána libovolná open-source aplikace.
- Vytvořit sadu znovupoužitelných objektů – Pro možnost generování automatizovaných testů bude vytvořena sada znovupoužitelných objektů, ze kterých se budou automaticky sestavovat, na základě testovacích scénářů, samostatné testy.
- Testy budou založeny na platformě Selenium – Automatizované testy včetně znovupoužitelných objektů budou napsány v Selenium.

  (Poznámka: Tento požadavek byl přidán dodatečně po provedení analýzy technologií.)
- Vytvořit diagramy procesů v aplikaci – Na základě zkušeností s aplikací Mantis BT budou vytvořeny diagramy procesů v aplikaci tak, aby byly dostatečně objemné a pokryly nejčastěji využívané části aplikace.
- Vytvořit testovací scénáře - Z vytvořených diagramů procesů budou vytvořeny (resp. vygenerovány) testovací scénáře, které budou dále využity pro sestavování automatizovaných testů.

3.2 Použité technologie

V následujících podkapitolách jsou popsány a srovnávány technologie (resp. software) zvažované při práci na semestrálním projektu a použité v bakalářské práci. Konkrétně jsou vybírány tři kategorie technologií:

1. Systém, který bude testován.
2. Aplikace, ve které se budou kreslit modely (diagramy) a která vygeneruje testovací scénáře.
3. Nástroj, ve kterém se budou sestavovat automatizované testy.

[1] volně dostupná aplikace

3.2.1 Testovaný systém

Mantis BT

Mantis BT (neboli Mantis Bug Tracker) je volně dostupný webový software vhodný pro správu projektů, reportování chyb a další podporu při vývoji software, který spadá do první kategorie (tj. jde o systém, který bude testován). Mantis BT je dostupný v českém jazyce a je napsán v PHP [2]. Ke zprovoznění Mantisu je potřeba webový server a databáze [1].

Předností Mantisu je jednoduché uživatelské rozhraní, možnost zasílání notifikací, tedy upozornění na uskutečněnou aktualizaci, a přikládání souborů. Administrátoři mohou pro větší přehlednost upravovat rozložení a zobrazování jednotlivých sloupců na obrazovce, měnit workflow[3] procesů a nastavovat zasílání těchto notifikací.

Tento software podporuje více typů uživatelských rolí – konkrétně:

- reportéra – uživatel má jen právo reportovat chyby.
- recenzenta – uživatel může navíc přidávat poznámky k chybám.
- aktualizátora – uživatel může navíc upravovat chyby (editovat).
- vývojáře – uživatel může navíc měnit stav chyby a uzavírat chyby.
- vedoucího – uživatel může navíc zakládat projekty a mazat chyby.
- správce – hlavní role v aplikaci. Uživatel má neomezená práva.

Každá z těch rolí má jiná práva, přičemž administrátorem (tj. uživatelem s nejvíce právy) je tzv. správce, který je v aplikaci obvykle jeden jediný.

Mantis BT je vhodný jak pro testery a vývojáře, tak i pro manažery, jelikož si v něm lze zobrazit aktuální statistiky projektů – například to, které chyby jsou nepřiměřeně dlouho nevyřešeny, kolik bylo nalezeno zásadních a kritických chyb, který z vývojářů má nejvíce úspěšně vyřešených chyb, nebo který reportér má vysokou chybovost. Celkový přehled nalezených chyb na projektu je možné vyexportovat a dále zpracovávat.

Odůvodnění výběru Mantis BT

V rámci tohoto semestrálního projektu a navazující bakalářské práce je Mantis BT vybraným softwarem, který bude testován pomocí vytvořených testovacích scénářů.

Software Mantis BT byl zvolen ze tří hlavních důvodů. Prvním z nich je to, že se jedná o dostatečně robustní systém, který obsahuje řadu procesů, které lze zmapovat. Druhým a souvisejícím důvodem je dostupnost systému - je volně dostupný, zdarma a k jeho provozu stačí mít webový server a databázi.

[2] PHP - skriptovací jazyk
[3] Workflow - posloupnost kroků, postup

Posledním důvodem, neméně významným, je fakt, že existuje školní studentský projekt pracující s Mantis BT, který umožňuje zanesení zapínatelných chyb do systému a další správu těchto chyb. Tento studentský projekt se zapínatelnými chybami bude dále využíván a případně upravován.

3.2.2 Aplikace pro kreslení modelů a generování testovacích scénářů

Oxygen

Aplikace Oxygen je volně dostupný software vytvořený na ČVUT FEL na Katedře počítačů, který umožňuje vytváření testovacích scénářů [2]. Má jednoduché uživatelské rozhraní a není potřeba absolvovat školení pro zvládnutí jeho ovládání.

Veškeré testovací scénáře jsou automaticky generovány na základě nakreslené specifikace (tj. diagramu) a umožňují zvolit si úroveň pokrytí SUT [4]. Pro další zpracování si lze grafy (resp. diagramy) vyexportovat z aplikace [2].

Srovnání Oxygen s COVER Toolbox[5]

COVER Toolbox je na rozdíl od Oxygen webová aplikace, která je přístupná uživatelům (resp. zaměstnancům a zákazníkům firem Sogeti a Valori) pouze po přihlášení. Obě tyto aplikace umožňují generovat testovací scénáře z modelů (tj. z diagramů) a importovat výstupy do dalších testovacích aplikací [3].

Hlavním rozdílem mezi Oxygen a COVER Toolbox je v GUI[6]. Jak bylo zmíněno výše, Oxygen má jednoduché grafické uživatelské rozhraní, které se snadno používá. COVER Toolbox ale žádné grafické uživatelské rozhraní nemá. Veškerá data je potřeba specifikovat v textových programech. Tento fakt práci s COVER Toolbox uživatelům komplikuje.

Srovnání Oxygen s Graph Walker[7]

Graph Walker je volně dostupný nástroj založený na testování modelů grafů. Tyto grafy prochází a generuje z nich testovací scénáře. K provozu Graph Walker je potřeba Java JDK[8], Maven[9] a nástroj yEd[10]. Veškeré testování a modelování probíhá ve vývojovém prostředí a před prvním spuštěním vyžaduje značné množství nastavování konfiguračních souborů [4].

[4] System Under Test – aktuálně testovaná aplikace

[5] COVER Toolbox – oficiální stránky aplikace dostupné na WWW:<http://tstr.nl/coverhd/loginbasics.php>.

[6] Graphic User Interface – grafické uživatelské rozhraní

[7] Graph Walker - oficiální stránky aplikace dostupné na WWW:<http://graphwalker.github.io/>

[8] Java Development Kit – soubor základních nástrojů pro vývoj aplikací v Java

[9] Apache Maven – nástroj pro správu buildů a testování

[10] yEd – grafický editor

Srovnání Oxygen s Conformiq[11]

Aplikace Conformiq je placený a robustní nástroj, který je stejně jako Oxygen založen na generování testovacích scénářů z modelů grafů. Conformiq je zdokumentován, nadále vyvíjen a je používán množstvím firem, jako jsou Nokia, Honeywell, Oracle nebo Daimler. Jedná se tedy o průmyslově využívanou aplikaci [5].

Srovnáním Oxygen s Conformiq lze zjistit, že aplikace Conformiq generuje testovací scénáře z výrazně složitějších modelů. Dalším rozdílem je větší množství funkcionalit (funkcí) v Conformiq a fakt, že je Oxygen zdarma.

Odůvodnění výběru Oxygen

Pro kreslení diagramů procesů a následné vygenerování testovacích scénářů byl vybrán Oxygen. Hlavní výhody této aplikace oproti výše zmiňovaným, jsou:

- Existence grafického uživatelského rozhraní – COVER Toolbox ho nemá.
- Možnost pracovat v aplikaci bez přihlášení – na rozdíl od COVER Toolbox.
- Oxygen je zdarma – Conformiq je oproti ostatním aplikacím cenově na jiné úrovni.
- Jednoduché nastavení a spuštění – Graph Walker vyžaduje značné množství nastavování před prvním spuštěním.

Poznámka: Po zvážení výhod aplikace Oxygen v rámci semestrálního projektu byl do zadání bakalářské práce přidán požadavek na tvorbu diagramů v Oxygen.

3.2.3 Nástroj pro sestavování automatizovaných testů

Selenium WebDriver

Selenium WebDriver je nástroj pro automatizované testování webových aplikací, který je založen na programovacím jazyku Java a je nezávislý na platformě. Při psaní testů je nutné specifikovat, v jakém prohlížeči budou testy probíhat [6].

Jednou z výhod Selenium WebDriver je, že vývoj testů probíhá v Java – uživatel tedy může používat při programování testů objekty, vlastní proměnné a nemusí se omezovat pouze na jeden nástroj. Selenium jako takové je jedním ze standardů v automatizovaném testování [24].

[11] Conformiq - oficiální stránky aplikace dostupné na WWW:< https://www.conformiq.com/>

Srovnání Selenium WebDriver s Robot Framework

Robot Framework je framework[12] pro automatizované testování, který je založen na testování pomocí klíčových slov. Tato klíčová slova se musí uživatel naučit správně používat a psát testy s použitím tabulátorů anebo používat speciální vývojové prostředí pro Robot Framework [8].

Při porovnávání si lze všimnout, že u Robot Framework není potřeba znát programovací jazyk, ale uživatel se musí naučit konkrétní klíčová slova a jejich význam. Robot Framework se dá dále rozšiřovat pomocí knihoven, které jsou napsány ve většině případů pro programovací jazyk Python.

Srovnání Selenium WebDriver se Selenium IDE

Selenium IDE je integrované vývojové prostředí, které podporuje psaní automatizovaných testů. Jedná se o rozšíření (resp. doplněk) webového prohlížeče Mozilla Firefox. Toto rozšíření umožňuje „nahrávání“ akcí uživatele v prohlížeči a jejich přehrání spolu s převedením nahraných dat do automatizovaných testů napsaných v Java a v dalších programovacích jazycích.

Selenium IDE na rozdíl od Selenium WebDriver nezná a nepodporuje objekty a není schopno reagovat na změny v aplikaci. IDE je vhodné pro rychlé jednorázové otestování webové aplikace, avšak testy napsané v Selenium WebDriver jsou znovupoužitelné a udržovatelné lépe než testy v Selenium IDE.

Srovnání Selenium WebDriver s QF-Test

QF-Test je robustní software nezávislý na platformě, který je určen k vývoji automatizovaných testů. Je založen na Java a vzdáleně připomíná Selenium IDE, které podporuje nahrávání testů. QF-Test má vlastní grafické rozhraní, lze s ním otestovat i desktopové aplikace a podporuje vkládání skriptů napsaných v programovacím jazyku Jython[13]. Veškeré „programování“ zde probíhá klikáním přes grafické rozhraní, kdy si uživatel vybírá z předpřipravených šablon [7].

Hlavní rozdíl mezi Selenium WebDriver a QF-Test je ve výše zmíněném grafickém rozhraní. Zatímco v QF-Test uživatel prochází šablony, které se musí naučit používat, a nahrává kroky a komponenty (Poznámka: Komponenty QF-Test bez jejich prvotního nahrání není schopen lokalizovat.), při používání Selenium WebDriver je uživatel omezen „pouze“ svými znalostmi programování.

Odůvodnění výběru Selenium WebDriver

Selenium WebDriver byl vybrán jako nástroj, ve kterém se budou sestavovat

[12] Framework – softwarová struktura - podpora při programování a vývoji
[13] Jython – implementace programovacího jazyka Python v Java

automatizované testy. Mezi důvody, proč byl tento nástroj zvolen, patří:

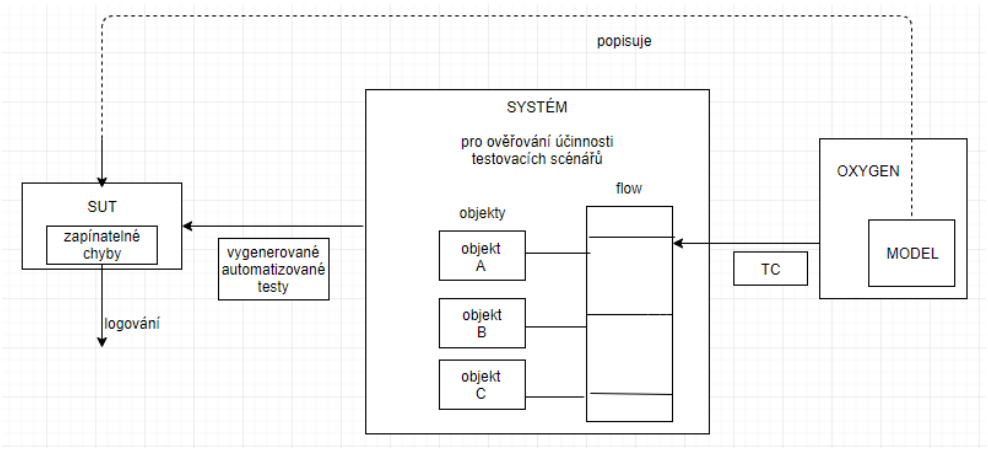
- Vývoj testů probíhá v Java – na rozdíl od QF-Test, který používá Jython, Selenium IDE nebo Robot Framework (testování pomocí klíčových slov).
- Uživatel může využívat objekty – QF-Test ani Selenium IDE objekty nezná.
- Testy mohou probíhat v libovolném webovém prohlížeči – rozdíl oproti Selenium IDE, ve kterém lze testy spouštět pouze v Mozilla Firefox.
- Lze začít hned programovat – není potřeba nahrávat komponenty jako u QF-Test.
- Jedná se o standard, který využívá Java.

Kapitola 4
Návrhová část

Část návrhová se zabývá popisem návrhu systému - tedy způsobem, jakým bude celý systém fungovat, jaké budou vstupy aplikací a vysvětluje roli diagramů procesů.

4.1 Přehledový model

Výsledný přehledový model Systému pro ověřování účinnosti testovacích scénářů včetně způsobu komunikace s ostatními aplikacemi (tj. typy vstupů a výstupů), je znázorněn na obrázku níže (viz Přehledový model 4.1).



Obrázek 4.1: Přehledový model

Model neboli diagram procesů, který popisuje nejčastěji využívané části aplikace Mantis BT (SUT), je zakreslen s pomocí aplikace Oxygen. Každý z modelů je zaměřen na jednu konkrétní část – například diagram založení projektu. Tento diagram zachycuje kroky (procesy), které uživatel může při daném úkolu, tedy zakládání projektu, provést.

Aplikace Oxygen zpracuje model a na základě zvolené úrovně pokrytí

(TDL[1]=1 nebo =2) vygeneruje konkrétní testovací scénáře, které by bylo vhodné otestovat, aby byla efektivně zkontrolována funkčnost testovaného systému. Tyto testovací scénáře budou předány Systému pro ověřování účinnosti testovacích scénářů.

Systém převezme testovací scénáře a podle nich sestaví ze sady znovupoužitelných objektů a předpřipravených funkcí automatizované testy v nástroji Selenium.

Znovupoužitelné objekty jsou v tomto případě objektyzastupující malé části aplikace Mantis BT – jednotlivé formuláře, položky v navigaci, tlačítka atp. Předpřipravené funkce, resp. předpřipravené seleniové skripty, jsou funkce, které budou vyplňovat jednotlivé kroky testu, jež nejsou zahrnuty ve znovupoužitelných objektech, tak, aby byl každý z testů schopen provést celý svůj testovací scénář během jednoho spuštění.

Takto vytvořené automatizované testy budou spuštěny na SUT (Mantis BT), který obsahuje uměle zapínatelné chyby, jejichž aktivace bude zaznamenávána do speciálního logu. (Poznámka: Projekt s uměle zapínatelnými chybami je stávající studentský projekt, který byl v rámci tohoto semestrálního projektu využit.)

Účinnost vytvořených testovacích scénářů bude ověřena a testovací scénáře uznány za dostačující v případě, kdy sestavené automatizované testy naleznou veškeré uměle zanesené chyby do aplikace Mantis BT.

4.2 Zmapované procesy

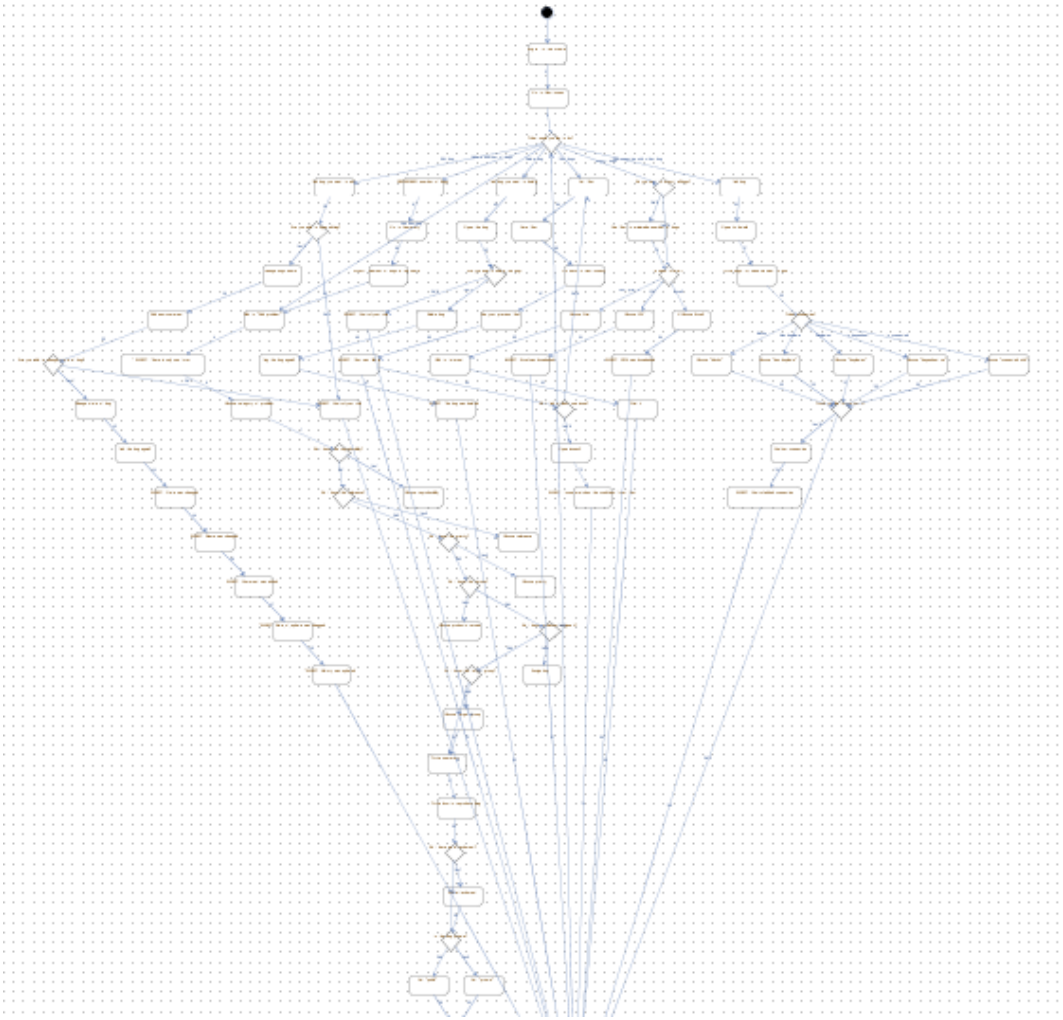
Na základě zkušeností s aplikací Mantis BT byly vybrány nejčastěji používané procesy, které byly zaznamenány do diagramů v aplikaci Oxygen. Pro uspořádání a snadnější dohledání určitých procesů byly diagramy rozděleny anebo naopak seskupeny podle oblasti, o kterou se jednalo - akce s uživatelským účtem, správa projektů, správa chyb, atp.

Do výsledných diagramů (resp. jednotlivých kroků) byly zařazeny i kontroly dat, nadpisů a přístupových práv. Z důvodu, že aplikace Oxygen nepodporuje export obrázků v jakémkoli formátu, jsou veškeré obrázky vytvořeny oddálením diagramů v aplikaci a pořízením snímku obrazovky – proto mají zhoršenou čitelnost. Na obrázku níže (Diagram správy chyby 4.2) je jeden z větších diagramů, z nichž bude Oxygen generovat testovací scénáře.

Tento diagram obsahuje 84 uzlů (kroků) a 106 hran. Aplikace Oxygen z něj vygeneruje zhruba 16 testovacích scénářů. Diagram správy chyby zachycuje procesy, mezi které patří založení nové chyby, úprava chyby, vymazání chyby, spojení chyby s jinou atp.

Hlavním důvodem pro zmapování procesů aplikace Mantis BT bylo to, že tyto diagramy jsou potřebným vstupem aplikace Oxygen, která na jejich základě generuje testovací scénáře. Vytvořené diagramy jsou nezbytnou součástí tohoto semestrálního projektu.

[1] Test Depth Level – úroveň pokrytí testu



Obrázek 4.2: Diagram správy chyby

Kapitola 5
Implementační část

Tato část představuje postup implementace projektu bakalářské práce s ohledem na strukturu systému, způsob její tvorby a objasnění funkcionalit systému jako celku.

5.1 Struktura systému

V následujících kapitolách jsou popsány a vysvětleny jednotlivé části projektu – tedy z jakých celků se samotný systém skládá, co bylo při programování využito za frameworky, a které balíčky projekt obsahuje. Podkapitoly jsou zaměřeny na tyto části:

- Page objects (neboli Page objekty) - návrhový vzor používaný při automatizovaném testování
- WebDriver (resp. Selenium WebDriver) a jeho využití při lokalizaci prvků
- TestSuite - objekt ve frameworku pro JUnit[1]

5.1.1 Page Object

Page Object neboli Page Object Model (POM) je návrhový vzor (tj. design pattern), který se využívá při automatizovaném testování [9]. Jedná se o způsob vytváření objektů jednotlivých HTML[2] stránek, jehož cílem je zamezit výskytu duplicitního kódu a zajistit, aby byl kód dobře udržovatelný. V případě, že během vývoje HTML stránky dojde ke změně identifikátoru libovolného prvku na stránce, je dohledání a úprava idetifikátoru v POM

[1] JUnit - framework pro jednotkové testy psaný v programovacím jazyce Java
[2] HTML - značkovací jazyk používaný pro tvorbu webových stránek

snadná. Samotný POM tedy slouží k oddělení abstrakce testů od testovacích skriptů.

Typická struktura Page Objectu je zobrazena níže. Každý Page Object zde odpovídá jedné HTML stránce.

```
// LoginPageObject.java

import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class LoginPageObject {

    public static WebDriver driver;

    By username=By.name("username");
    By passwd=By.name("password");
    By loginButton=By.cssSelector("#login-form>fieldset>"
    +"input.width-40.pull-right.btn.btn-success.btn-inverse");
    By logOutBanner=By.xpath("//*[@id=\"navbar-container\"]");
    By logOutButton=By.linkText("Odhlasit");

    public LoginPageObject(WebDriver driver){
        this.driver = driver;
    }
    public void setUsername(String newUsername){
        driver.findElement(username).sendKeys(newUsername);
    }
    public void setPasswd(String newPasswd){
        driver.findElement(passwd).sendKeys(newPasswd);
    }
    public void clickLoginButton(){
        driver.findElement(loginButton).click();
        WebDriverWait wait = new WebDriverWait(driver, 6);
        wait.until(ExpectedConditions."
        +"visibilityOfElementLocated"(By.xpath("//*[@id=\"
        +""navbar-container\"]")));
    }
    public void loginToMantis(String newUsername,
      String newPasswd){
        this.setUsername(newUsername);
        this.setPasswd(newPasswd);
        this.clickLoginButton();
    }
}
```

Z ukázky lze vidět, že se jedná o třídu LoginPageObject. Jako první

jsou zde deklarovány idetifikátory a lokalizátory prvků stránky, které jsou definovány pouze v této třídě – dále se s nimi pracuje za pomoci proměnných. Tato třída v sobě dále zahrnuje konstruktor objektu, který má na starosti předávání WebDriveru.

Ve třídě se dále nacházejí metody sloužící k provedení určité akce. Například metoda setUsername přijímá na vstupu řetězec znaků, který se pomocí WebDriveru zapíše do pole „uživatelské jméno“. Obdobným způsobem fungují i další metody.

5.1.2 WebDriver

Selenium WebDriver byl popsán již výše (3.2.3). Tato podkapitola se zabývá způsobem využití a nastavení WebDriveru v projektu. V rámci „Systému pro ověřování účinnosti testovacích scénářů“ je WebDriver (dále jen „driver“) nepostradatelnou součástí. Je využit jak pro simulaci uživatelského klikání a psaní, tak i pro lokalizaci prvků HTML stránky. Driver může lokalizovat prvky následovně s pomocí:

- atributu „name“ v HTML – například: By.name("username");
- CSS selektoru – například: By.cssSelector("login-form>fieldset>input.width-40.pull ");
- využití textu na stránce – například: By.linkText("Odhlásit");
- využití části textu na stránce – například: By.partialLinkText("Odhl");
- XPath – jedná se o absolutní cestu k prvku na stránce – například: By.xpath("//*[@id=n̈avbar-container]/div[2]/ul/li[3]/a/span");
- className By.className("Odhlásit");
- id – nejjednodušší způsob – předpokladem je výskyt unikátních ID na stránce – například: By.id("navbar-container ");
- tagName – používá se převážně při výběru prvků z nabídky – například: By.tagName("select");

Každá z těchto variant má jinou výhodu, ale nutno zmínit, že se častěji mění poloha prvku na stránce než jeho ID. Vzhledem k tomu, že driver neslouží jen k lokalizaci prvků, ale i k simulaci klikání a otevírání odkazů a prohlížeče, je vhodné si ho pro tyto účely nastavit. V projektu této bakalářské práce je driver nastaven takto:

```
// DriverSetting.java
public class DriverSetting {

    public static WebDriver driver;
```

```
    public static WebDriver newDriver() {
        System.setProperty("webdriver.chrome.driver",
                "C:\\Users\\Aneta\\Documents\\BAKALARKA\\"
            +"chromedriver_win32.zip\\chromedriver.exe");

        driver=new ChromeDriver();
        driver.get("http://147.32.80.239/mantisbt");
        driver.manage().window().maximize();

        return driver;
    }
    public static WebDriver getDriver() {
        if(driver==null) {
            driver=newDriver();
        }
        return driver;
    }
    public static void closeDriver() {
        driver.close();
    }
}
```

Driver zde má svoji vlastní třídu, která se stará o jeho inicializaci a nastavení. Metoda System.setProperty(); slouží k identifikaci umístění driveru. Dále je zvoleno, že se bude jednat o driver pro prohlížeč Chrome a při spuštění bude otevřena příslušná URL adresa (resp. adresa Mantis BT). Pro větší přehlednost je okno prohlížeče maximalizováno. Třída je ošetřena, aby se v případě volání o předání stavu driveru zamezilo případu, kdy driver ještě nebude inicializován.

5.1.3 TestSuite

TestSuite je objekt ve frameworku JUnit (tj. framework pro jednotkové testy psaný v jazyce Java). TestSuite v testování je obecně jakýsi kontejner, který obsahuje jednotlivé testovací případy a slouží k jejich shlukování a k dosažení větší přehlednosti – například při reportování stavu otestovanosti systému. Tento vztah je znázorněn na obrázku 5.1, kde je TestSuite zobrazen jako lineární sekvence (posloupnost) za sebou jdoucích a navazujících kroků konkrétních testů.

Testovací plán je rozdělen do několika Test Suite – v případě této bakalářské práce odpovídá jeden Test Suite jednomu diagramu procesů vytvořenému v Oxygen. Test Suite tedy obsahuje konkrétní vygenerované průchody diagramem, které je třeba otestovat, aby bylo docíleno požadovaného pokrytí testy [25].

Obrázek 5.1: Ukázka TestSuite

V tomto projektu jsou Test Suite vytvářeny dynamicky za pomoci JUnit4 TestAdapter. JUnit4TestAdapter je framework (resp. třída), který rozšiřuje java.lang.Object. Jeho vstupem je třída .class, která obsahuje testy. Hlavním důvodem využití JUnit4TestAdapter je, že kombinuje různé verze JUnit.

Na následujícím kódu je ukázán způsob použití TestSuite a JUnit4 TestAdapter.

```
// CSVRead.java
TestSuite suite=new TestSuite();

        Class c;

        for(int i=0; i<result.size(); i++) {

            if(result.get(i)=="END"){

                c=Class.forName("testClasses."
                +"LogOutFromMantisTestClass");

                System.out.println(c.getName());
                suite.addTest(new JUnit4TestAdapter(c));
                 }
        junit.textui.TestRunner.run(suite);
      }
```

Zde je vidět, že je nejprve inicializován nový objekt suite a nová pomocná obecná třída. Poté je for-cyklem dynamicky předáván třídě její název. Dále jsou JUnit4TestAdapter a daná třída vstupem, ze kterého se získá testovací metoda a ta se vloží do zmiňované suite.

Z důvodu, že do suite lze vložit pouze samotné testovací metody značené @Test, je nastavení WebDriveru prováděno samostatně (viz 5.1.2). Jedná se

o praktické řešení, jehož cílem je zajistit dobrou udržovatelnost projektu.

5.2 Implementace struktury systému pomocí balíčků (packages)

Projekt Systému pro ověřování účinnosti testovacích scénářů je rozdělen do tří hlavních balíčků:

1. pageObjects
2. testClasses
3. tests

Balíček pageObjects obsahuje jednotlivé HTML stránky (resp. Page Objekty), jak bylo vysvětleno v kapitole 5.1.1 a také soubor Variables.java zahrnující použité proměnné v testech.

TestClasses balíček pojímá třídy, které obsahují konkrétní testovací metody (resp. testovací kroky – test stepy). Každá ze tříd odpovídá jednomu kroku v diagramu procesů a lze ji naimplementovat například tímto způsobem:

```
// LoginTestClass.java

import org.junit.Test;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import pageObjects.LoginPageObject;
import pageObjects.Variables;
import tests.DriverSetting;

public class LoginTestClass {

  public static WebDriver driver;

  @Test
  public void loginToMantis() {

      driver=DriverSetting.getDriver();
      LoginPageObject loginPageObject=new
          LoginPageObject(driver);

      loginPageObject.loginToMantis(Variables.
          admin, Variables.adminPasswd);
    }
}
```

Zde je příklad - LoginTestClass. @Test metoda značí, že se jedná o test, který se předá do TestSuite. Nejprve je získán stav driveru. Dalším krokem je inicializace příslušného Page objektu, jež k dané testovací třídě náleží. Poté je na příslušném objektu zavolána požadovaná metoda – například metoda loginToMantis s potřebnými vstupy sestávajícími z uživatelského jména a hesla k účtu. Za účelem větší přehlednosti jsou tyto údaje udržovány v jednom souboru (třídě), kde je lze spravovat – konkrétně ve Variables. Ostatní TestClasses jsou strukturou obdobné této ukázkové.

Posledním z balíčků je tests, který je řídící jednotkou celého projektu. Obsahuje soubory:

- CSV read
- DriverSetting

Soubor *CSV read* se stará o čtení a ukládání dat z CSV[3] souborů do paměti, jak bylo vysvětleno v *Implementační části* a zároveň vytváří výsledný ArrayList[4] s testy, které se následně přidávají do TestSuite.

DriverSetting je zmiňovaným souborem zajišťujícím nastavování, inicializování a předávání driveru.

5.3 Popis funkcionality z implementačního pohledu

Tato část je zaměřena na popis systému – konkrétně na to, jak funguje od začátku - od vytvoření diagramů procesů do konce – tedy do provedení všech vytvořených automatizovaných testů. Kapitola se dále zabývá detailním vysvětlením způsobu fungování celého systému.

5.3.1 Od diagramu ke scénáři

Prvním a nejdůležitějším krokem, od kterého se odvíjel další postup, bylo vytvoření diagramů procesů Mantis BT v Oxygen – popsáno v kapitole 4.2.

Za účelem snadnější práce s aktivitami, rozhodovacími body a hranami v diagramech, je ke každému z „prvků“ přiřazena určitá předpona (prefix) dle těchto pravidel:

- A_(activity) – označení pro aktivitu (uzel)
- D_(decision) – označení pro rozhodovací bod
- E_(edge) – označení pro hranu vystupující z uzlu

[3] Comma-separated values - soubor s hodnotami oddělenými čárkami nebo středníky
[4] ArrayList - pole s dynamickou velikostí

- R_(result) – označení pro hranu, která vychází z rozhodovacího bodu

Z již vytvořených diagramů jsou vygenerovány testovací scénáře s následujícím nastavením předvoleb generování:

- zvoleno TDL=1
- vybrána funkce „Show edges in test cases“
- vybrána funkce „Show activity nodes in test cases“
- vybrána funkce „Show decision nodes in test cases“
- u ostatních předvoleb jsou ponechány defaultní (původní) hodnoty

Po potvrzení tohoto nastavení jsou vygenerovány testovací scénáře, které jsou dále vyexportovány jako CSV soubory a uloženy do zvoleného adresáře. Poté jsou také vyexportovány samotné diagramy procesů jako CSV soubory.

5.3.2 Propojení scénářů s Page objekty

Následujícím krokem je propojení testovacích scénářů s page objekty, za účelem dosažení plné funkčnosti systému. Aplikace Oxygen nepovolovala použití neunikátních názvů aktivit, rozhodovacích bodů a hran, proto se stejné funkce jmenují různě.

Zmiňované propojení (resp. mapování) je provedeno ruční úpravou vyexportovaných CSV souborů s jednotlivými diagramy. Do těchto souborů je vždy doplněn název konkrétní testovací třídy náležící k příslušnému kroku diagramu.. Tímto způsobem je docíleno namapování testovacích tříd na libovolný krok diagramu 5.2.

	A	B	C
1	A_	A_Log in to Mantis	LoginTestClass
2	A_	A_Choose project	ChangeProjectTestClass
3	A_	A_Go to View Issues	GoToViewIssuesTestClass
4	A_	A_Find bug you want to delete	GoToViewIssuesTestClass
5	A_	A_Open the bug	OpenIssueTestClass
6	D_	D_Are you able to delete the bug?	1
7	A_	A_ASSERT: Control your role	ControllUsernameTestClass
8	A_	A_Delete bug	ClickDeleteButtonTestClass
9	A_	A_Fing the bug again	GoToViewIssuesTestClass

Obrázek 5.2: Ukázka mapování diagramů v CSV

5.3.3 Načítání souborů do paměti

Pro účely dalšího zpracování CSV souborů s diagramy a s testovacími případy, jsou tyto dokumenty postupně načítány a ukládány do paměti – tuto funkci plní třída CSVRead.

Daný soubor je procházen a ukládán do ArrayListu. V projektu se nacházejí dva typy souborů:

1. soubor s diagramy
2. soubor s testovacími případy

Vzhledem k tomuto faktu je každý ze dvou typů souborů ukládán jiným způsobem.

První z nich - soubor s diagramem obsahující mapování - je procházen po řádcích a zapisován do ArrayListu stanoveným způsobem:

- Na každé sudé pozici včetně nuly se nachází název akce/rozhodovacího bodu/výsledek rozhodovacího bodu.
- Na každé liché pozici se vyskytuje název funkce (testovací metody), která je na daný krok diagramu namapována.

Druhým typem je soubor s testovacími případy, který je načítán také po řádcích, přičemž jeden řádek v souboru odpovídá jednomu testovacímu případu (resp. jednomu testovacímu scénáři). V ArrayListu se nacházejí za sebou uložené jednotlivé kroky testovacích scénářů (každý krok na jedné pozici) náležící vždy jednomu diagramu. Testovací scénáře jsou tedy ukládány za sebou a jsou od sebe vzájemně odděleny vložením slova „END”.

5.3.4 Vytvoření sekvence kroků

Cílem předchozích kapitol bylo vytvořit vhodnou „základnu” pro projekt. Tato kapitola je zaměřena již na samotné sestavování sekvencí kroků na základě dvou vytvořených ArrayListů.

Pro tuto výslednou sekvenci kroků je inicializován třetí ArrayList. Nejprve je procházen ArrayList s testovacími případy - do proměnné je uložen název první akce/rozhodovacího bodu/výsledku rozhodovacího bodu a tento název je hledán ve druhém Arraylistu. V případě jeho nalezení je poznamenán název namapované funkce (resp. název TestClass).

Tímto způsobem se projde celý první ArrayList a sestaví se výsledná sekvence kroků, která se bude dále zpracovávat.

5.3.5 Vytvoření TestSuite a spuštění testů

Sekvence kroků (testů) je dokončena, a proto začíná poslední část implementace - vytvoření TestSuite.

TestSuite je nejprve inicializována a poté je do ní postupně přidáván každý krok ze sekvence testů ze třetího Arraylistu. Z důvodu, že jsou kroky ukládány jako Stringy, jsou před vložením do TestSuite překonvertovány na třídy (tj. Classes). Tato testovací třída je poté uložena pomocí JUnit4TestAdapter jako samostatný test do TestSuite.

Nakonec je vytvořená TestSuite spuštěna a automatizované testy jsou aplikovány na testovaný systém Mantis BT.

Kapitola 6

Zvažované varianty řešení

Během zpracovávání bakalářské práce bylo zvažováno několik variant řešení celého projektu, které však po zakomponování do projektu a vyzkoušení nevykazovaly požadované chování nebo případně nezapadaly a zbytečně komplikovaly výslednou implementaci.

Pro ukázku jsou vybrány tři různé varianty, které byly nakonec zavrženy. Jednalo se o:

1. případ, kdy jedna třída TestClass obsahovala všechny @Test metody, které svojí příslušností spadaly k danému Page objektu
2. způsob namapování aktivit diagramu na testovací metody
3. neukládání CSV dokumentů do paměti

Varianta A - „případ, kdy jedna třída TestClass obsahovala všechny @Test metody, které svojí příslušností spadaly k danému Page objektu" byla původně v projektu zapracována a vyzkoušena na malém vzorku dat. Při testování této varianty řešení se vyskytlo několik problémů, jejichž mitigace by zabrala zbytečně mnoho času, proto se od varianty A ustoupilo. Mezi konkrétní vyvstalé problémy patřilo:

- omezení volání testovacích metod - resp. klasickou testovací metodu nelze samostatně volat.

Tuto situaci by sice vyřešilo použití TestSuite kontejneru umožňující volání:

```
// Example.java

        TestSuite suite= new TestSuite();
        suite.addTest(new MyTest("testDivideByZero"));
```

V tomto případě je „MyTest" název třídy a „testDivideByZero"zvolený test, který bude přidán do TestSuite.

Ponechání varianty A by však znamenalo, že by CSV soubory s mapováním musely obsahovat další informaci navíc (tj. název třídy včetně názvu testovací metody). Pokud se vezme v potaz, že se data ze souborů ukládají do paměti

a že by existovalo větší množství testovacích scénářů s mapováním, je zřejmé, že by toto řešení zatěžovalo paměť.

Řešení B - „způsob namapování aktivit diagramu na testovací metody" úzce souvisí s předchozím. Z toho důvodu bylo vytvořeno „odlehčené" mapování, které neobsahuje zbytečné informace navíc.

Možnost C - „neukládání CSV dokumentů do paměti" byla zamítnuta hned na začátku. Předpokladem pro toto rozhodnutí je, že je rychlejší data uložit do paměti jako ArrayList než při každém kroku procházet určené dva CSV soubory a neustále jimi iterovat.

Pro tyto zmíněné implementační nevýhody a problémy bylo vybráno řešení popsané v kapitole 5.

Kapitola 7

Testování systému

Základním stavebním kamenem správně fungující aplikace je testování. Abychom mohli říci, že Systém pro ověřování účinnosti testovacích scénářů pro software opravdu ověřuje účinnost testovacích scénářů, je potřeba ho otestovat. Problém nastává v případě, kdy testujeme aplikaci, která je určena k testování, což tento systém je.

Jako první se nabízí vytvořit automatizované testy, které by systém prošly a překontrolovaly. Jedná se o jednu z možností, ale v této situaci je výhodnější vytvořit si vlastní testovací scénáře a systém otestovat manuálně.

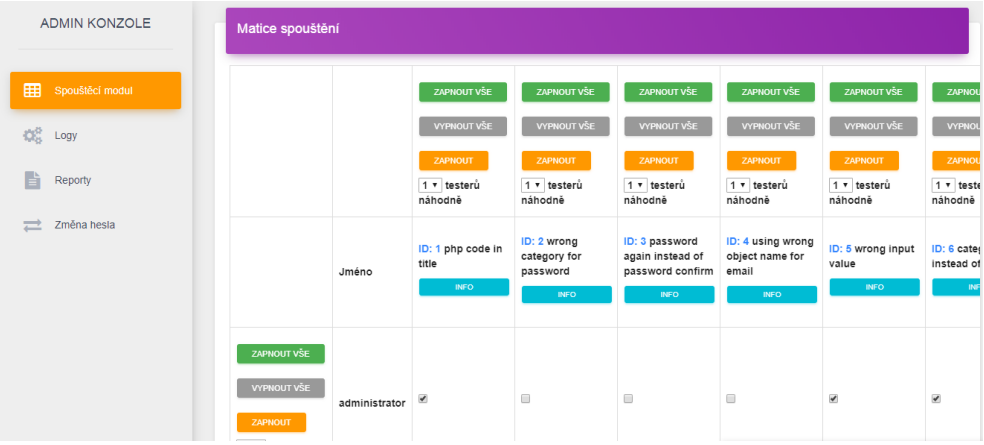
Cílem testování je nalézt veškeré aktivované zapínatelné chyby.

7.1 Aktivované zapínatelné chyby

V převzatém studentském projektu se zapínatelnými chybami a speciálním logem (Mantis BT), který je využit v bakalářské práci, byly zapnuty pro uživatele „administrátor" v administrátorské konzoli (7.1) tyto chyby:

- ID 1: php code in title - ve správě uživatelského účtu se objeví PHP kód.
- ID 5: wrong input value - v poli pro email se objeví heslo uživatele.
- ID 8: wrong title value - nadpis *Upravit ušivatele*.
- ID 9: negation in account verification/reset - uživatel může mít „prázdné"heslo.
- ID 11: password update message broken - špatná hláška.
- ID 14: negation in show category - nelze zvolit kategorii chyby.
- ID 17: missing 'Site Information' header - chybí název stránky ve *Správě*.
- ID 20: negation in product build - špatná verze produktu.
- ID 23: wrong code for bug severity - pole *Priorita* se v *Zobrazení chyby* ukáže dvakrát.

- ID 28: php code in by project - ve správě projektu se objeví PHP kód.



Obrázek 7.1: Administrátorská konzole ovládající zapínání chyb v Mantis BT

7.2 Zvolený testovací scénář

Prvním krokem testovacího scénáře je kontrola, zda se opravdu požadované zapínatelné chyby aktivovaly. Ověření provedeme pohledem do speciálního logu konzole.

Dále následuje spuštění automatizovaných testů (resp. Systému pro ověřování účinnosti testovacích scénářů pro software). Po určitém časovém úseku, kdy testy skončí a je nutno překontrolovat, zda testy proběhly do konce a zda našly nějaké chyby.

V případě, že byly nalezeny chyby, tak zkontrolujeme, jestli odpovídají zapnutým chybám v aplikaci Mantis BT.

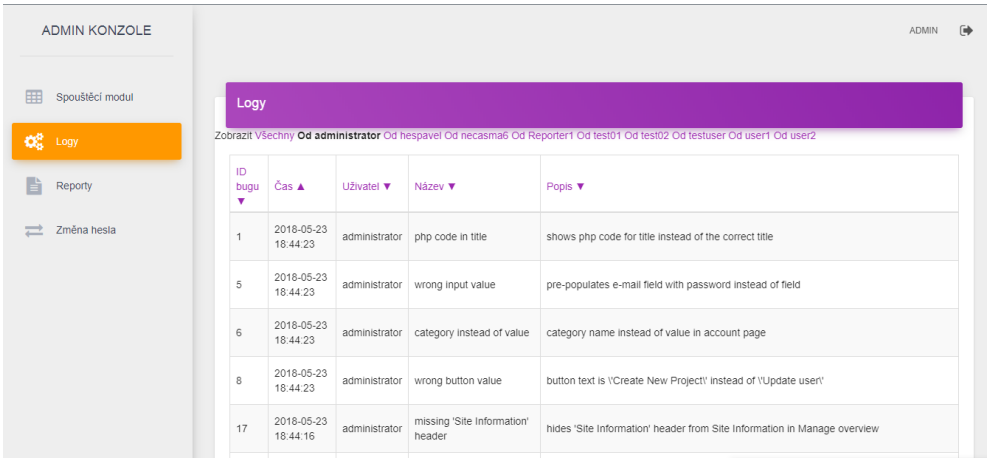
7.3 Výsledek testování

Z obrázku (7.2) je vidět, že byla informace o zapnutí chyb zanesena do logu. První krok je tedy splněn. Systém tedy musí chyby odhalit.

Automatizované testy doběhly do konce a log ve vývojovém prostředí ukazuje, že některé testy skončily s chybou.

Po průchodu výsledků testů s chybami a provedením kontroly, zda testy identifikovaly hledané chyby, je možné konstatovat, že byly všechny zapnuté chyby testy označeny.

Z dokončeného testování a nalezení aktivovaných chyb lze učinit závěr, že Systém pro ověřování účinnosti testovacích scénářů pro software funguje.



Obrázek 7.2: Speciální log zapisující aktivaci chyby

Kapitola 8

Instalace a spuštění projektu

Správné spuštění projektu vyžaduje tyto kroky:

1. stažení a naimportování si projektu do zvoleného vývojového prostředí - projekt byl konkrétně vyvíjen v NetBeans IDE 8.2[1], ale měl by být funkční i v jakékoli jiném prostředí.
2. stažení a instalaci Selenium WebDriver[2] pro CHROME prohlížeč - využita byla verze chromedriver_win32.zip. Důležité: Projekt vyžaduje existenci driveru pro Chrome!
3. nainstalování prohlížeče Chrome (nikoli Chromium) - v případě, že ho uživatel ještě nemá.

Nyní je nutné nastavit si cestu k .exe aplikaci Selenium WebDriver v souboru *DriverSetting.java* nacházejícího se v adresáři *BachelorThesisSelenium/test/-tests/* změnou na řádku:

```
// DriverSetting.java

System.setProperty("webdriver.chrome.driver",
        "C:\\Users\\Aneta\\Documents\\BAKALARKA\\"
        +"chromedriver_win32.zip\\chromedriver.exe");
```

Provedením předchozích kroků je tedy projekt připraven ke spuštění. Po volbě *RUN FILE* nebo *SHIFT+F6* nad souborem *BachelorThesisSelenium.java* (v mém případě je v C:/Users/Aneta/Documents/NetBeansProjects/BachelorThesisSelenium/test/BachelorThesisSelenium.java) je systém nastartován a začne se otevírat webový prohlížeč, ve kterém započne testování.

[1] Dostupný zde: https://netbeans.org/downloads/
[2] Dostupný zde: https://www.seleniumhq.org/download/

Kapitola 9

Závěr

Cílem bakalářské práce bylo navrhnout Systém pro ověřování účinnosti testovacích scénářů pro software (resp. navrhnout jeho schéma), vybrat a analyzovat technologie vhodné pro tento projekt a vytvořit diagramy procesů ve zvolené aplikaci tak, aby byly popsány nejčastěji používané části aplikace a bylo možné z nich vygenerovat testovací scénáře. Dalším z cílů bylo provést rešerši existujících řešení se zaměřením na obdobné projekty a implementace celého systému.

Výsledkem práce jsou vyhotovené diagramy, vybrané technologie, vytvořené schéma (přehledový model architektury), vypracovaná rešerše a naprogramovaný Systém pro ověřování účinnosti testovacích scénářů pro software.

Za účelem lepšího pochopení celé problematiky byly přiblíženy zvažované technologie, popsán obrázek se schématem architektury a přiložena ukázka diagramů a demonstrace kódů v implementační části.

Vybranými technologiemi v rámci semestrálního projektu tedy jsou – Mantis BT pro otestování, Oxygen pro vytváření diagramů a vygenerování testovacích scénářů a Selenium WebDriver pro sestavování automatizovaných testů za použití programovacího jazyka Java.

V průběhu bakalářské práce byly výsledky semestrálního projektu převzaty a začleněny do této práce a dále rozšířeny o rešeršní a implementační část a oddíly zaobírající se zvažovanými variantami řešení, instalací a testováním projektu.

Na závěr lze konstatovat, že byly všechny zmíněné cíle dosaženy - tj. Systém pro ověřování účinnosti testovacích scénářů pro software je vyhotoven, popsán a otestován a splňuje veškeré funkční a nefunkční požadavky včetně požadavků ze zadání bakalářské práce.

Literatura

[1] MantisBT Reviews: *Overview, Pricing and Features.* Software Reviews and Business Advice | FinancesOnline.com [online]. 2017 [cit. 2017-01-08]. Dostupné z WWW: <https://reviews.financesonline.com/p/mantis/>.

[2] Bures, M. (2015). *PCTgen: automated generation of test cases for application workflows.* In New Contributions in Information Systems and Technologies, Advances in Intelligent Systems and Computing, vol.353 (pp. 789-794). Springer.

[3] Model Voorbeelden. *COVER* [online]. 2017 [cit. 2017-01-09].Dostupné z WWW:<http://design2test.com/cover_16/examples_process.php?adress=naam@valori.nl>.

[4] Introduction into creation of a model for GraphWalker | *GraphWalker for testers. Getting started overview* | GraphWalker for testers[online]. Copyright ©2017 graphwalker.org. [cit. 2018-01-07]. Dostupné z WWW: <http://graphwalker.github.io/Model_design/>.

[5] Quick Start. *Conformiq* | Next generation in Agile Software Test Automation [online]. Copyright © 2018 [cit. 2018-01-10]. Dostupné z WWW: <https://www.conformiq.com/products/quick-start/>.

[6] Why do we use WebDriver instead of Selenium IDE? - Stack Overflow. Stack Overflow - Where Developers Learn, Share, and Build Careers [online]. [cit. 2018-01-07]. Dostupné z WWW: <https://stackoverflow.com/questions/19683100/why-do-we-use-webdriver-instead-of-selenium-ide>.

[7] QF-Test: Automated GUI Test Tool Java and Web - Quality First Software GmbH. 302 Found [online]. Copyright © 2018 QFS GmbH [cit. 2018-01-10]. Dostupné z WWW: <https://www.qfs.de/en.html>.

[8] Robot Framework. Robot Framework [online]. [cit. 2018-01-07].Dostupné z WWW: <http://robotframework.org/>.

[9] Set up Page Object Model (POM) in Selenium Automation Framework. QA Automation Tools Tutorial [online]. Copyright © 2013

[cit. 22.05.2018]. Dostupné z WWW: <http://toolsqa.com/selenium-webdriver/page-object-model/>.

[10] Rafi, D.M. et al. (2012). *Benefits and limitations of automated software testing: Systematic literature review and practitioner survey*, In 7th International Workshop on Automation of Software Test (AST), Zurich, Switzerland, pp. 36-42.

[11] Kasurinen, J., Taipale, O., & Smolander, K. (2010). *Software test automation in practice: empirical observations.* Advances in Software Engineering, 2010.

[12] Berner, S., Weber, R., Keller, R.K. (2005). *Observations and lessons learned from automated testing,* In Proceedings of the 27th international conference on Software engineering (ICSE '05), ACM, New York, pp. 571–579.

[13] Persson, C., & Yilmazturk, N. (2004). *Establishment of automated regression testing at ABB: industrial experience report on'avoiding the pitfalls'.* In Automated Software Engineering. Proceedings. 19th International Conference on (pp. 112-121). IEEE.

[14] Alégroth, E., & Feldt, R. (2014). *Industrial Application of Visual GUI Testing: Lessons Learned.* In Continuous Software Engineering (pp. 127-140). Springer.

[15] Fewster, M. (2001). *Common Mistakes in Test Automation.* White paper, Groove consulatns. Dostupné z WWW: <http://www.agileconnection.com/sites/default/files/article/file/2012/XDD2901filelistfilename1_0.pdf>.

[16] Bures, M. (2014). *Automated testing in the Czech Republic: the current situation and issues.* In Proceedings of the 15th International Conference on Computer Systems and Technologies (pp. 294-301). ACM.

[17] Filipsky, M., Bures, M., & Jelinek, I. (2015). *Creating Smart Tests from Recorded Automated Test Cases.* In New Contributions in Information Systems and Technologies, Advances in Intelligent Systems and Computing, vol.353 (pp. 773-780). Springer.

[18] Bures, M., Filipsky, M., & Jelinek, I. (2018). *Identification of Potential Reusable Subroutines in Recorded Automated Test Scripts.* International Journal of Software Engineering and Knowledge Engineering. 28(01), 3-36.

[19] Bures, M. (2015). *Model for evaluation and cost estimations of the automated testing architecture.* In New Contributions in Information Systems and Technologies, Advances in Intelligent Systems and Computing, vol.353 (pp. 781-787). Springer.

[20] Bures, M. (2014). *Change Detection System for the Maintenance of Automated Testing.* In IFIP International Conference on Testing Software and Systems, Lecture Notes in Computer Science, vol.8763 (pp. 192-197). Springer.

[21] Bures, M. (2015). *Framework for assessment of web application automated testability.* In Proceedings of the 2015 Conference on research in adaptive and convergent systems (pp. 512-514). ACM.

[22] Bures, M. (2015). *Metrics for automated testability of web applications.* In Proceedings of the 16th International Conference on Computer Systems and Technologies (pp. 83-89). ACM.

[23] Bures, M., & Filipsky, M. (2016). *SmartDriver: Extension of Selenium WebDriver to Create More Efficient Automated Tests.* In IT Convergence and Security (ICITCS), 2016 6th International Conference on (pp. 319-322). IEEE.

[24] Collin, M. (2015). *Mastering Selenium WebDriver.* Packt Publishing Ltd.

[25] Koomen, T., et al. (2013). *TMap next: for result-driven testing.* Uitgeverij kleine Uil.