ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra počítačů

Framework pro rekonstrukci modelu aplikací pro chytrou televizi

Framework for Model Reconstruction of Smart Television Applications

bakalářská práce

Studijní program: Softwarové inženýrství a technologie

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

Jan Richter

Praha 2018

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Richter** Jméno: **Jan** Osobní číslo: **420069**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra počítačů**

Studijní program: **Softwarové inženýrství a technologie**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Framework pro rekonstrukci modelu aplikací pro chytrou televizi

Název bakalářské práce anglicky:

Framework for Model Reconstruction of Smart Television Applications

Pokyny pro vypracování:

Navrhněte, vytvořte a otestujte framework, který bude automaticky procházet aplikaci pro chytrou televizi a rekonstruovat model jejího uživatelského rozhraní. Tento model bude založen na orientovaném grafu, kde uzly budou představovat obrazovky aplikace a hrany přechody mezi těmito obrazovkami. Model bude možné exportovat ve formátu akceptovaném aplikací Oxygen, vyvíjené na katedře počítačů ČVUT FEL. Pro rozsáhlé aplikace bude framework umožňovat nastavení kritéria, kdy bude možno procházení aplikace ukončit. Funkčnost frameworku otestujte na třech vybraných aplikacích pro chytrou televizi.

Seznam doporučené literatury:

Rafi, D.M. et al.: 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, 2012, pp. 36--42.
Alégroth, E., & Feldt, R. (2014). Industrial Application of Visual GUI Testing: Lessons Learned. In Continuous Software Engineering (pp. 127-140). Springer.

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Miroslav Bureš, Ph.D., laboratoř inteligentního testování softwaru FEL

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

Datum zadání bakalářské práce: **24.11.2017** Termín odevzdání bakalářské práce: **25.05.2018**

Platnost zadání bakalářské práce: **30.09.2019**

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

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen 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.

23/5/18

Datum převzetí zadání Podpis studenta

Prohlášení

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

Praha, 24. 5. 2018

Podpis autora práce

Poděkování

Chtěl bych poděkovat panu Ing. Miroslavu Burešovi, Ph.D. za velmi užitečné rady a konzultace, které mi pomohly při vypracování mé bakalářské práce. Dalším, kterému bych chtěl poděkovat je Dr. Bestoun S. Ahmed, Ph.D., který mi během práce poskytl cenné metodické konzultace k tématu.

Abstrakt

Práce se zabývá vývojem softwaru usnadňující automatické testování aplikací pro chytré televize. Chytré televize jsou relativně novou platformou, která se stává běžnou součástí domácností. A dostává se čím dál více do hledáčku vývojářů, kteří by pro tuto platformu chtěli vyvíjet aplikace. V dnešní době existuje velice málo nástrojů pro testování SmartTV aplikací, což je důvodem pro vznik frameworku, o kterém pojednává tato práce.

Práce je rozdělena do čtyř částí. První část popisuje dostupná řešení a rozebírá studie o automatizovaném testování. Druhá část se zabývá analýzou softwaru. Popisuje specifika, která souvisí s vývojem aplikací pro SmartTV, dále popisuje požadavky, které musí software splnit, a nakonec samotný návrh softwaru. Ve třetí části je rozebrána implementace softwaru a použité technologie. Ukazuje jednotlivé problémy, které je nutné vyřešit pro správné fungování softwaru a jejich řešení. A nakonec samotné fungování vyvíjeného softwaru. Poslední část se zabývá testováním na třech vybraných aplikacích.

Abstract

The paper deals with development of software for automated testing of smart TV applications. SmartTVs are a new platform becoming a common part of our homes. Software engineers are likely interested in developing applications for this platform. Nowadays, there are lack of tools for testing SmartTV applications, which is motivation for the creation of this work.

The paper is divided into four parts. The first part describes existing solutions and discusses about automated testing. The second part deals with software analysis. It describes the specifics that are related to the development of SmartTV applications, next it describes the requirements of software, and finally the software design itself. The third part describes the software implementation and used technology. It shows the individual issues that need to be solved for the proper function of the software and its solutions. And finally, the actual function of the software. The last part deals with testing on three selected applications.

Obsah

Seznam obrázků

Seznam scriptů

Použité zkratky a akronymy

SUT	System under test (testovaný software)
GUI	Graphical User Interface (grafické rozhraní aplikace)
SmartTV	Chytrá televize
DOM	Document Object Model (Objektový model dokumentu)
HTML	HyperText Markup Language (jazyk popisující internetové stránky)

1. Úvod

V poslední době jsou chytré televize (SmartTV) stále běžnějším zařízením v domácnostech. Proto se o oblast chytrých televizí začíná zajímat více vývojářů, kteří tvoří nové a nové aplikace. Protože SmartTV je relativně nová technologie, je jen velice málo nástrojů, které by byly zaměřené na platformu Smart TV a zabývaly se problematikou kolem testování jejích aplikací. Přitom je stále více vývojářů, kteří vyvíjejí aplikace pro tuto platformu. Tento projekt si klade za cíl vytvořit testovací nástroj, který by vývojářům usnadnil testování jejich vyvinutých SmartTV aplikací.

Tato práce se zabývá vytvářením modelů pro automatické testování, které se hojně využívají při testování webových aplikací. Pro webové aplikace funguje velice známý nástroj Selenium WebDriver, který se používá pro návrh automatických testů a následné automatické testování. Často se pro práci s webovými aplikacemi využívá metoda zvaná crawling, kdy počítačový program zvaný crawler pročítá stránky webové aplikace a na základě získaných dat vytváří testovací model. Tato práce se inspiruje metodou webového crawlingu a snaží se vytvořit crawler, který by nepracoval s webovými aplikacemi, ale uměl by pracovat s aplikacemi pro chytré televize.

Hlavním cílem této práce je vytvořit framework, který bude přistupovat k aplikaci bez znalosti její vnitřní struktury (black box). Automaticky nalezne aktivní elementy uvnitř aplikace a vytvoří model, který umožní generování testovacích scénářů. Vytvořený model bude možné nahrát do aplikace Oxygen pro další zpracování. Framework by měl usnadňovat testerům vytváření automatických testů pro chytré televize.

Pro vytvoření frameworku je potřeba analyzovat jakým způsobem fungují SmartTV aplikace. Následně navrhnout způsob jakým bude framework komunikovat s testovanou aplikací a s aplikací Oxygen a jakým způsobem bude vytvářet model testované aplikace. Nakonec je nutné vytvořený framework důkladně otestovat na několika vybraných aplikacích.

2. Existující řešení

Testování softwaru je velice důležitou součástí softwarových projektů, jelikož zajišťuje kvalitu vytvářených produktů a snižuje nadbytečné výdaje, které musejí vývojáři vydat na opravu chyb uvnitř již vydaných produktů. Zvyšování efektivity testovacích procesů má za následek zvyšování kvality vydávaných softwarů a snižování nákladů na jejich vývoj. Jedno z možných řešení, jak zvýšit efektivitu testování je zavedení automatizace do částí testovacích procesů. Díky čemuž se testeři mohou zaměřit na kritické části softwaru, a jednoduché a opakující se úkony mohou nechat na automatickém systému. Automatické testy se snažíme vytvořit tak, aby byla jejich použitelnost při testování daného softwaru co nejdelší. Poměr ceny a výkonu automatických testů závisí na tom, kolikrát bude možné test zopakovat, aniž by byl upraven kvůli změně v SUT. Automatické testy se používají zpravidla na opakující se úlohy, protože vytvořit nebo upravit automatický test je náročnější než daný test provést jednou manuálně.

Jednou z možností, jak vytvářet automatické testy, je metoda capture/replay. Jedná se o metodu, kdy testovací nástroj nahrává pohyby myší nebo stisknutí tlačítek uvnitř SUT, a poté je schopen tyto události zopakovat a sledovat výstupy z SUT. Testy se tedy vytváří tak, že tester zapne nahrávání a aplikaci projde podle předem vytvořeného testovacího scénáře. Testovací nástroj poté může procházet SUT podle uložených dat. Pro testování webových aplikací pomocí capture/replay existuje framework Selenium WebDriver[1] a pro SmartTV byl vytvořen framework suitest[2]**.** Problém tohoto přístupu je v tom, že je náchylný na změny v grafickém rozhraní testované aplikace. Pokud je provedena změna v grafickém rozhraní, nahraná data už nebudou sedět s očekávanými výsledky, tím pádem je nutné testovací scénáře nahrát znovu, čímž se tester připraví o výhodu použití automatických testů. Vysoká údržba automatizovaných testů je brána jako jeden z nevýznamnějších problémů této technologie a nejedná se pouze o individuální případové studie[1].

Existují různé experimentální projekty, které se pokouší tento problém automatizovaně řešit, například detekci změn v systému [8], použitím znovupoužitelných objektu [9] včetně identifikace takových objektu na základě analýzy kódu[11].

Další možný přístup, jak vyřešit tento problém, je crawling GUI testované aplikace. Jedná se o přístup, kdy nástroj zvavý crawler prochází SUT a vytváří model testovaného systému. Crawling se používá i například pro zhodnoceni automatizovatelnosti scénářů[12].

[1] Dostupné z: http://www.seleniumhq.org/
[2] Dostupné z: https://suite.st/

To ale není jediné možné použití. Pomocí crawlingu jsme dokonce schopni vytvořit i nové testovací scénáře. Rozdíl oproti capture/replay je, že crawlingem projdeme mnohem větší rozsah aplikace.

Jak je uvedeno výše, při testování webových aplikacích se crawling běžně používá, ovšem pro testování SmartTV aplikací nebyly nalezeny studie, které by popisovaly využití crawlingu pro testování SmartTV aplikací. Z tohoto důvodu byl vytvořen nástroj, který je popsán v této práci a který pro testování SmartTV aplikací crawling využívá.

3. Analýza

Tato kapitola se nejprve zabývá aplikacemi pro chytré televize a jejich odlišností od jiných aplikací. Poté analyzuje požadavky, které jsou kladeny na vytvářený systém. Z požadavků jsou vytvořeny a popsány případy užití systému. A ke konci kapitoly je popsán návrh vyvíjeného systému.

3.1. Specifika SMART TV aplikací

Tato kapitola se zabývá konceptem aplikací pro chytré televize. Popisuje, co to chytré televize jsou a jak fungují. Dále se zabývá single-page aplikacemi, které jsou velice podobné SmartTV aplikacím a popisuje v čem jsou si podobné a v čem naopak rozdílné. Ke konci této kapitoly je popsán způsob testování aplikací pro SmartTV.

3.1.1. Smart TV

SmartTV [14] můžeme jednoduše popsat jako televizi připojenou k internetu. Jedná se zařízení, které kombinuje vlastnosti počítače a televize. SmartTV má v sobě integrovaný internet a díky tomu je schopná kromě zpracování a přehrávání klasického televizního signálu také přehrávat nejrůznější multimediální obsah, jako například internetové televize, streamování videa z internetu nebo přehrávání filmů přes domácí síť. SmartTV také uživatelům povoluje instalovat a používat nejrůznější aplikace od vývojářů třetích stran, které zvyšují schopnosti daného televizoru. Každá chytrá televize obsahuje operační systém, ve kterém veškeré aplikace běží. Tato práce je zaměřená na operační systém Tizen od společnosti Samsung, který je jádrem všech Samsung SmartTV.

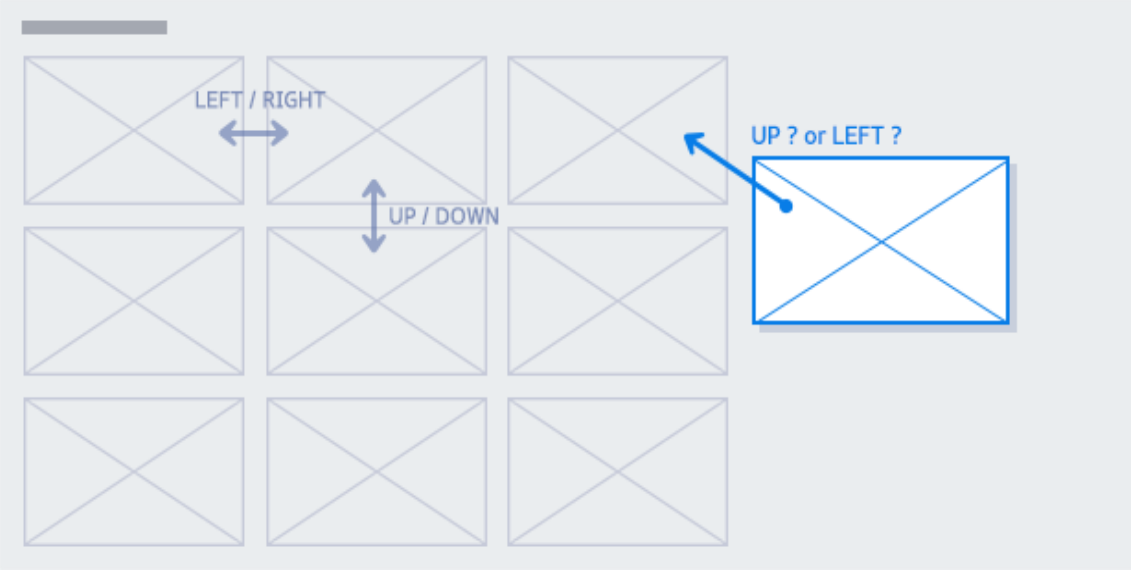
3.1.2. Single-Page aplikace

Single page [16] aplikace je typ webové aplikace, která interaguje s uživatelem za pomoc dynamické změny webové stránky na rozdíl od klasické webové aplikace, která interaguje pomocí načítání nové verze stránky ze serveru. Většinu práce musí obstarat javascript na straně klienta. Single page aplikace využívá technologie AJAX. Tato technologie zajišťuje komunikaci se serverem bez načtení nové webové stránky. Pomocí AJAX klient se serverem synchronizují pouze změny v datech a díky tomu je aplikace uživatelsky přívětivější a uživatel není zdržován načítáním nových stránek. Single page aplikace vytváří pocit klasické desktopové aplikace, ale běží ve webovém prohlížeči.

3.1.3. SmartTV aplikace

SmartTV aplikace [17] funguje velice podobně jako singl page aplikace v internetovém prohlížeči. Jejich hlavní rozdíl je v ovládání a grafickém rozhraní. Obrazovka televize je většinou větší než obrazovka počítače. Na druhou stranu uživatel při ovládání televize sedí a relaxuje na pohodlné pohovce ve svém obývacím pokoji a je mnohem více vzdálen od obrazovky než uživatel u počítače. Proto je grafické rozhraní SmartTV aplikace přizpůsobeno těmto podmínkám. Jednotlivé komponenty jsou větší, aby byly snadno viditelné. Aplikace pro SmartTV mají velice zjednodušené grafické rozhraní, aby se daly snadno ovládat pomocí dálkového ovládání.

Zatímco u aplikací na internetu máme možnost používat myš a klávesnici, SmartTV aplikaci ovládáme pomocí televizního dálkového ovládání, které má značně omezené možnosti. Pomocí dálkového ovládání je možné se pohybovat mezi objekty grafického rozhraní pouze ve čtyřech směrech: nahoru, dolů, doprava a doleva. Navíc není možné některé objekty přeskakovat, čili z jednoho objektu je možné se přesunout pouze na sousední objekty. Tento způsob ovládání je velice odlišný oproti ovládání myší, kdy je možné kliknout na jakýkoliv zobrazený objekt. Z tohoto důvodu je vzhled SmartTV aplikací přizpůsoben tak, aby byly snadno ovladatelné pomocí dálkového ovládání. Jednotlivé objekty jsou vedle sebe uspořádány do neviditelné tabulky tak, aby bylo uživateli na první pohled jasné, který objekt je nahoře, který dole, který vpravo a který vlevo (viz. Obrázek 1).



Obrázek 1 - Uspořádání objektů v aplikaci

3.1.4. Testování SmartTV aplikací

Každá aplikace musí být před svým zveřejněním řádně otestována a případně opravena, aby se zajistila dostatečná kvalita vydané aplikace. Je tomu tak i u aplikací pro SmartTV[15]. Možnosti testování aplikace jsou prakticky nekonečné, a proto se pro testování vytvářejí nejrůznější strategie, tak aby vybrané testy nalezly co možná nejvíce chyb za co možná nejkratší dobu. Aby byla aplikace řádně otestována, je nutné testovat ji na různých úrovních, od testování čistě jednotlivých funkcí v aplikaci až po testování aplikace jako celku. Účelem této práce je testování aplikace pomocí automatického testování. V dnešní době je nejznámějším softwarem pro testování webových aplikací pomocí automatických testů Selenium3. Pro automatické testování SmartTV aplikací se používá aplikace Suitest4. Obě aplikace používají stejný přístup. Nejdříve spustí testovanou aplikaci a nahrávají události, které se v aplikaci odehrávají, a poté jsou schopny tyto události zopakovat. V praxi to vypadá tak, že tester zapne Suitest, spustí nahrávání a prokliká se aplikací předem připraveným způsobem. Poté nahrávání uloží jako test a Suitest je schopen spustit danou sekvenci automaticky znovu, kdykoliv si to bude tester přát. Tento přístup nutí testera, aby předem připravil průchody danou aplikací a ty poté pomocí Suitest nahrál, a tím vytvořil dané testy, což může být u rozsáhlých aplikací velice náročná práce.

Framework popsán v této práci si klade za cíl zautomatizovat i část vytváření testů. Čili místo aby framework jen nahrával ovládání testera, měl by být schopen sám rozeznat, které elementy jsou důležité, na které lze kliknout a na které nikoliv. Ovšem stejně jako Suitest i tento framework musí být schopen aplikaci testovat bez znalosti vnitřní struktury aplikace – black box. Tento přístup nám zásadně komplikuje způsob, jaký jsou elementy uvnitř webových aplikací a tím pádem i u SmartTV aplikací tvořeny. Protože jakýkoliv element uvnitř webové aplikace muže interagovat s uživatelem a záleží jen na naprogramovaném ovládání u daného elementu. Z tohoto důvodu musí být framework schopen chovat se podobně jako uživatel. Musí umět ovládat aplikaci pomocí ovladače a musí být schopen vyhodnotit, na který element v danou chvíli kliknul. Podobně jako to dělá uživatel, když ovládá televizi. Pokud bude framework schopen simulovat chování uživatele, tak snadno nalezne v aplikaci všechny elementy, na které lze kliknout.

[3] Dostupné z: http://www.seleniumhq.org/
[4] Dostupné z: https://suite.st/

3.2. Požadavky na řešení

1. Vytvoření modelu

Framework musí umět vytvořit model testované aplikace. Framework nemusí znát vnitřní strukturu dané aplikace. Měl by být schopen model vytvořit pouze pomocí interakce s aplikací a na základě výstupu dané aplikace, podobně jako normální uživatel.

2. Model

Vytvořený model testované aplikace musí obsahovat informace dostačující k vytvoření testovacích scénářů pro automatické testování.

3. Omezení modelu

Framework musí umožnit testerovi omezit velikost vytvořeného modelu testované aplikace. Tester musí mít možnost určit počet elementů v modelu.

4. Export dat

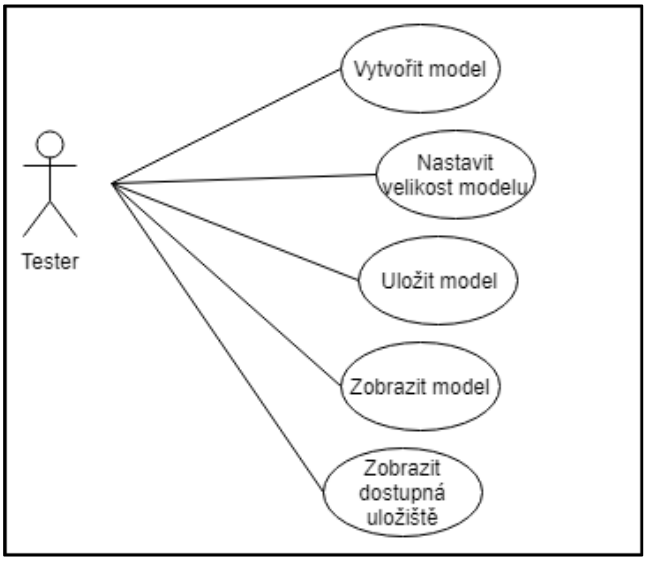
Framework umožní exportovat model testované aplikace ve formátu CSV.

5. Prostředí

Framework musí být kompatibilní s webovými aplikacemi pro operační systém chytrých televizí Tizen od společnosti Samsung.

6. Technilogie

Framework musí být vytvořen pomocí programovacího jazyku javascript bez přidaných knihoven.



Obrázek 2 - Diagram případů užití

3.3. Případy užití

Vytvořit datový model:

1. Tester pomocí tlačítek najede na element, od kterého chce začít aplikaci prohledávat.
2. Tester zavolá funkci crawlerStart().
3. Systém začne prohledávat testovanou aplikaci a vytvářet její model.
4. Po skončení systém vypíše „finish“ a vrátí odkaz na vytvořený model.

Nastavit velikost modelu:

1. Tester pomocí tlačítek najede na element, od kterého chce začít aplikaci prohledávat.
2. Tester zavolá funkci crawlerStart(stopCriterium). Tester do funkce vloží počet elementů na jedné stránce
3. Systém začne prohledávat testovanou aplikaci a vytvářet její model. Dodržuje počet elementů na jedné stránce
4. Po skončení systém vypíše „finish“ a vrátí odkaz na vytvořený model.

Uložit model:

1. Systém již dokončil vytváření modelu a má ho k dispozici.
2. Tester zavolá funkci saveModel(Path).
3. Systém vytvoří CSV soubor s názvem model_date v adresáři, který zadal tester.

Zobrazit model:

1. Tester použije nástroj Oxygen
2. Tester klikne na import graph in CSV a nahraje model z crawleru
3. Systém vygeneruje graf
4. Tester klikne na auto arrange graph
5. Systém upraví graf
6. Systém zobrazí graf

Zobrazit dostupná uložiště:

1. Tester zavolá funkci storage().
2. Systém vypíše všechny připojené externí uložiště, které lze využít.

3.4. Návrh systému

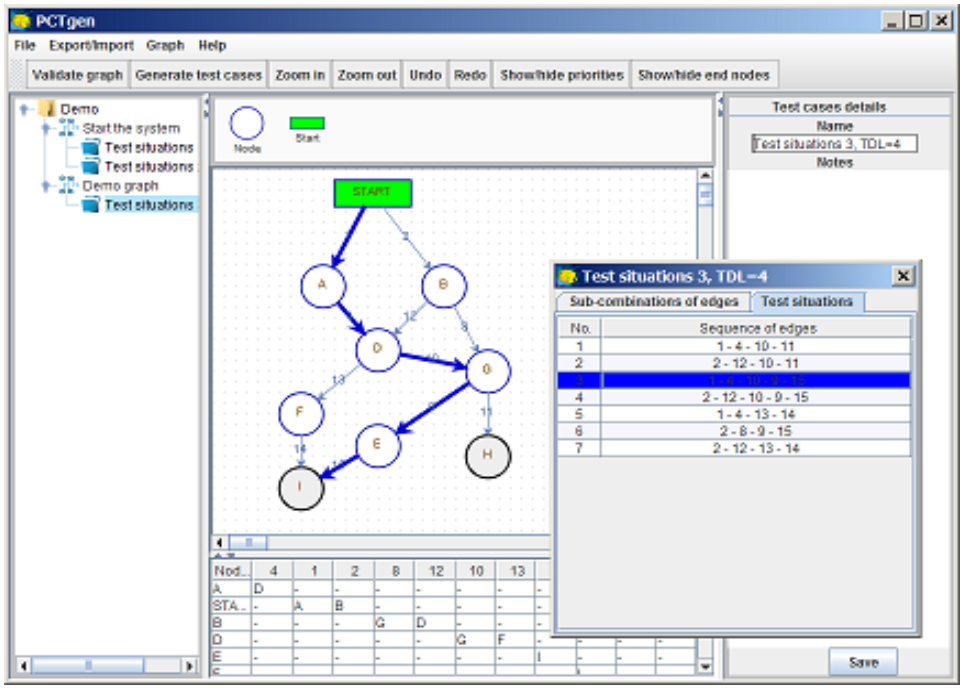
Framework se skládá ze dvou částí. První je crawler, který vytváří model testované aplikace. Druhou částí je externí aplikace Oxigen, která pracuje s modelem a vytváří testovací scénáře. Crawler je konzolová aplikace, se kterou se může pracovat v konzoli uvnitř nástroje web inspektor. Crawler rozšiřuje testovanou aplikaci o funkce usnadňující její testování.

3.4.1. Modul Crawler

Crawler je nástroj na vytváření modelu testované aplikace. Crawler nemanipuluje se zdrojovým kódem testované aplikace, pouze s ní interaguje, jako by byl normální uživatel. Čili pouze simuluje ovládání dálkovým ovladačem a poté čte výstupy z aplikace. Výstupem crawleru je model testované aplikace, který zobrazuje aktivní elementy v aplikaci. Daný model je více popsán v kapitole 4.2.2. Datový model je možné exportovat ve formátu CSV. Exportovaný soubor je kompatibilní s nástrojem pro tvorbu testovacích scénářů Oxygen.

3.4.2. Nástroj Oxygen

Oxygen[5] [18]je open-source nástroj pro automatické generování testovacích scénářů pro aplikační procesy, vyvíjený v jazyce Java. Oxygen vytvořila a sále vyvíjí Software Testing IntelLigent Lab[6] na katedře počítačů fakulty elektrotechnické ČVUT v Praze. V nástroji se dají vytvářet grafy popisující průchody testovanou aplikací, které oxygen graficky zobrazuje. Grafy je možné také importovat ve formátech CSV a XML. Na základě vytvořených grafů je Oxygen schopen vygenerovat testovací scénáře pro automatické testování testované aplikace (viz. Obrázek 3). Testovací scénáře je možné exportovat ve formátech CSV, XML a JSON.



Obrázek 3 – Ukázka z uživatelského rozhraní Oxygen

[5] http://still.felk.cvut.cz/oxygen/
[6] http://still.felk.cvut.cz/

4. Implementace

Tato kapitola nejprve rozebírá nástroje a technologie, které bylo k vývoji použity. Poté popisuje architekturu vyvinutého systému a datové modely pro reprezentaci dat. Dále řeší technické problémy, které bylo nutné během vývoje vyřešit. A nakonec popisuje způsob nasazení a ovládání systému.

4.1. Použité technologie

Tato kapitola popisuje použité technologie a nástroje během vývoje frameworku. Zabývá se operačním systémem, ve kterém běží SmartTV aplikace. Dále popisuje emulátor, pomocí kterého byl framework vytvořen a testován. A v neposlední řadě se zabývá programovacími jazyky, které byly použity pro vývoj frameworku.

4.1.1. Tizen

Tizen [20] je open source operační systém od společnosti Samsung, který podporuje inovace a dodává uživatelsky přívětivý design. Tizen není vytvořen pouze pro chytré televize nebo mobilní zařízení. Podporuje mnoho zařízení od chytrých telefonů a tabletů přes televize až po zařízení pro automobilový průmysl a tím zasahuje skoro do každého odvětví. Tizen je založený na linuxovém jádře a softwarové knihovně GNU C Library. Aplikace běžící v sytému Tizen jsou podobné webovým aplikacím, protože jsou psané za pomocí jazyků JavaScript a HTML5, které se používají pro vytváření internetových aplikací a stránek. Tizen podporuje pro vývoj aplikací i javascriptové knihovny jQuery a jQuery mobile.

4.1.2. Samsung TV Emulator

TV emulátor [21] znázorněn na Obrázek 4 je nástroj, který vytváří virtuální SmartTV na počítači. Slouží k testování aplikací bez nutnosti mít reálné zařízení. Emulátor poskytuje většinu možností jako reálná televize a přidává některé další, jako je virtuální dálkové ovládání, simulované připojené USB zařízení jako například flash disk a virtuální síťové připojení. Jelikož emulátor pouze emuluje reálné televizní zařízení, některé funkce nejsou dostupné. V emulátoru nemůžeme pracovat s některými zařízeními a funkcemi, které SmartTV podporují, jako je kamera, Bluetooth, hlasové ovládání nebo 3D zobrazení. Ovšem pro potřeby tohoto projektu je TV emulátor naprosto dostačující.

Obrázek 4 – Ukázka z aplikace TV emulátor

4.1.3. HTML

HTML (HyperText Markup Language) [22] je hlavním jazykem pro tvorbu webových stránek. Vznikl jako jednoduchý formát pro tvorbu hypertextových dokumentů, které se snadno dají přenášet z platformy na platformu a začal se používat pro přenos internetových stránek mezi serverem a clientem. HTML definuje a popisuje strukturu a obsah dokumentu. Jak se vyvíjí internet, tak HTML také neustále prochází vývojem, aby splňoval požadavky internetového publika. Celý proces vývoje a udržování jazyka řídí organizace W3C. HTML využívá strukturu tagů pro interpretování obsahu, který poté zobrazuje internetový prohlížeč. Prohlížeč dokument přečte a převede text do viditelné podoby. Pomocí tagů v dokumentu prohlížeč zjisti, jestli text má zobrazit jako například odstavec, tabulku, odkaz, nebo obrázek.

4.1.4. JavaSrcipt

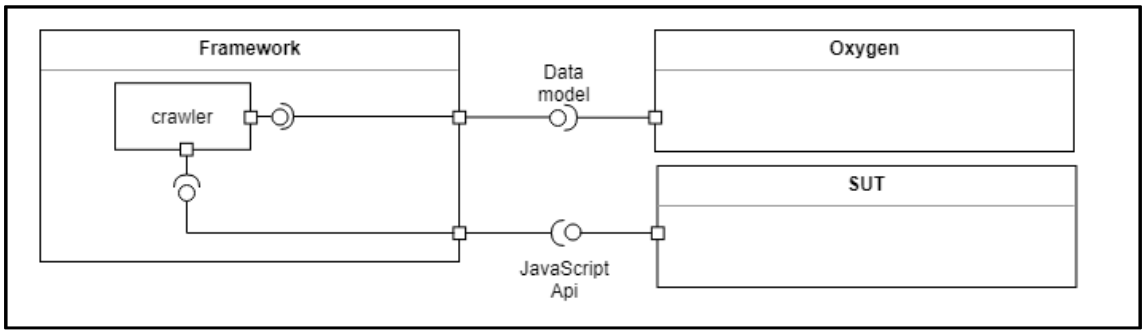
JavaScipt [23] je nenáročný interpretovaný objektově orientovaný jazyk, který se převážně využívá jako skriptovací jazyk pro webové stránky a aplikace. Program v JavaScriptu se spouští na straně klienta na rozdíl od ostatních jiných interpretovaných programovacích jazyků (např. PHP a ASP) a používá se k ovládaní klientské části webových aplikací. JavaScript má API pro práci s textem, polem, daty, regulárními výrazy, a má funkce, které podporují manipulaci s document object modelem (DOM) webové HTML stránky.

4.2. Architektura vyvinutého řešení

Framework funguje jako javascriptový script, který se musí vložit do hlavní HTML stránky testované aplikace, stejně jako se používají scripty třetích stran ve webových aplikacích, například google Analytics. Po vložení framework testovanou aplikaci obohatí o testovací funkce, které pomohou testerovi vytvořit model testované aplikace.

4.2.1. Komponenty

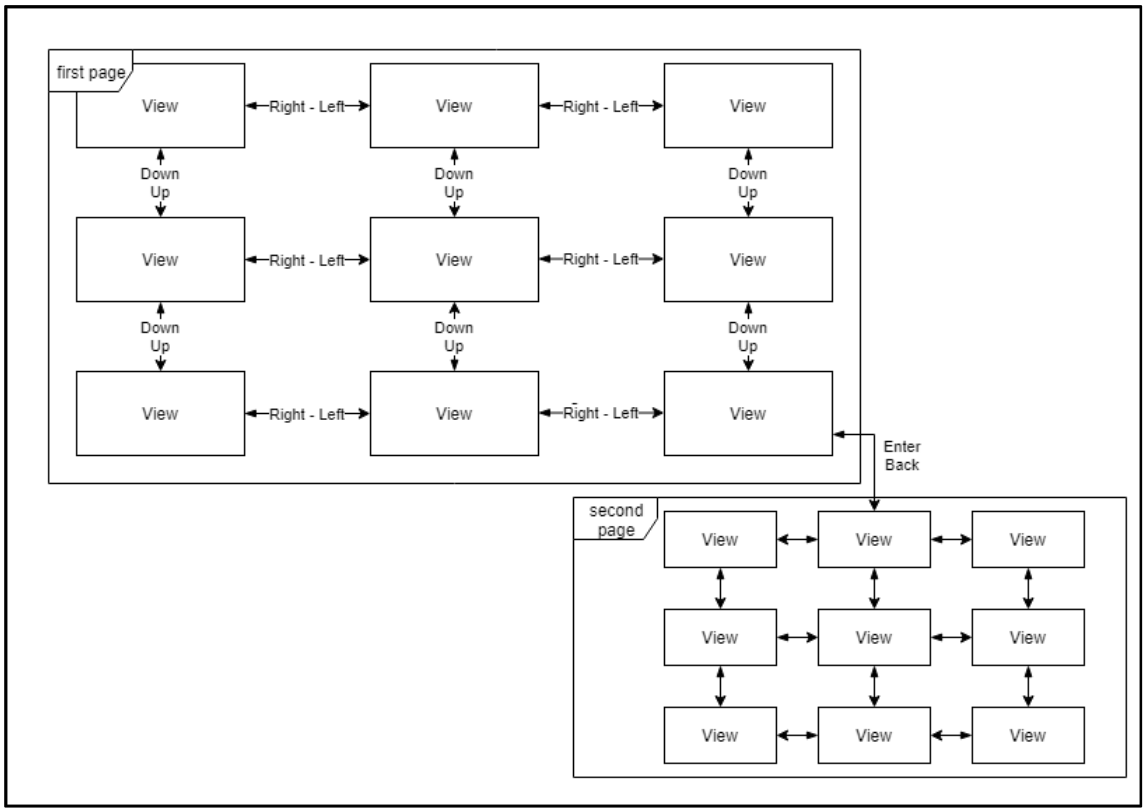
Framework je rozdělen na dvě části, na crawler a externí aplikaci oxygen. Toto rozdělení znázorňuje Obrázek 5. Crawler je součástí scriptu, který se vkládá do testované aplikace a jeho funkce rozšiřují aplikaci o automatické testování. Crawler slouží k vytváření modelu testované aplikace a získaný model poskytuje aplikaci Oxygen pro další zpracování. Oxygen je externí aplikace, která zpracovává model testované aplikace a podle něj vytváří testovací scénáře. Aplikace Oxygen je popsána v kapitole 3.4.2.



Obrázek 5 - Diagram struktury frameworku

4.2.2. Datový model testovaného systému

Pro ukládání nalezených aktivních elementů byl vytvořen datový model, který představuje upořádání elementů uvnitř aplikace. Celá aplikace je reprezentována pomocí grafu, kdy jednotlivé vrcholy v grafu představují nalezené aktivní elementy (viz. Obrázek 6). Každý vrchol je vytvořen pomocí třídy View. Třída View má atributy value, up, down, left, right, enter, back a slouží pouze k ukládání informací o aktivních elementech. Atribut value obsahuje DOM element, který byl nalezen jako aktivní. Atributy up, down, left, right, enter a back obsahují referenci na třídu View, jejíž hodnota se stane aktivní po stisknutí tlačítka nahoru, dolu, doleva, doprava, ok nebo zpět. Jednotlivá views, která jsou spojena vazbami up, down, left a right, považujeme za prvky jedné stránky. Vazby enter a back jsou vazbami mezi dvěma stránkami. Spojením jednotlivých views se vytvoří graf, který reprezentuje všechny aktivní elementy uvnitř aplikace.



Obrázek 6 - Schéma datového modelu

4.2.3. Struktura formátu pro export modelu

Aplikace Oxygen přijímá pro import grafů formáty XML a CSV. Pro exportování modelů vytvořených tímto frameworkem byl zvolen formát CSV, tak aby mu aplikace Oxygen rozuměla. Grafy aplikace Oxygen mají několik specifik. Každý graf má jeden počáteční uzel s názvem START, do kterého nevstupují žádné hrany. Každý graf má alespoň jeden koncový uzel, ze kterého žádné hrany nevystupují. A nakonec žádné dvě hrany nesmí mít stejné jméno. Při vytváření modelu se každému view přiřadí jedinečný identifikátor (view1; view2; …), který se připojí k názvu jeho hran (up1; down1; left1; right1; up2; down2; …). Tím se vyřeší problém s duplicitními hranami. Koncové uzly a uzel START se vytváří při exportu. Nejprve se k modelu přidá uzel START, který se vloží před první vytvořené view, a jako vazba mezi START a view se použije vazba enter. Poté se vytvoří koncové uzly, ty se připojí vazbou enter ke všem views, které tuto vazbu dosud nemají. Takto upravený model se převede do tabulky ve formátu CSV, kde sloupce znamenají jednotlivé hrany a řádky jednotlivé uzly grafu.

4.3. Vyřešené technické problémy

Tato kapitola popisuje největší problémy, které bylo nutné vyřešit, aby mohl framework správně fungovat. Každá část se zabývá jedním problémem. Popisuje, o jaký problém se jedná a také způsob, jakým je problém ve frameworku vyřešen.

4.3.1. Simulace ovládání ovladačem

Framework přistupuje k testované aplikaci jako k black box modelu, takže nezná vnitřní strukturu aplikace, ale musí být schopen vytvořit model popisující objekty uvnitř aplikace. Aby framework objevil objekty uvnitř aplikace, musí na ně kliknout podobně, jako by to udělal uživatel. Proto je nutné, aby framework uměl simulovat zmačknutí tlačítka na ovladači pro pohyb nahoru, dolů, doleva, doprava a kliknutí na daný objekt. V java scriptu se rozpoznávají akce od uživatele pomocí eventů. Eventy vyvolané pomocí televizního ovladače jsou typu keydown. Každý event obsahuje atributy, které určují, jaké tlačítko event vyvolalo – keyCode, a který DOM element je cílovým pro vyvolaný event – target. Atribut keyCode může pro různá zařízení určovat různá tlačítka, proto je v systému Tizen zabudovaná funkce *tizen.tvinputdevice.getKey(‚KEY').code*, která pro zadané tlačítko vrátí keyCode kompatibilní se zařízením na kterém aplikace v danou chvíli běží. Java script umožňuje vytvářet vlastní eventy a nastavovat jim požadované vlastnosti. Takto vytvořené eventy se dají programově vyvolat pomocí funkce *dispatchEvent€.* Díky těmto vlastnostem java scriptu, lze vytvořit keydown event a nastavit mu atribut keyCode pomocí *tizen.tvinputdevice.getKey(‚KEY').code* a atribut target na právě aktivní element, poté ho vyvolat pomocí funkce *dispatchEvent€*. Tímto způsobem lze simulovat stisk daného tlačítka na ovladači.

4.3.2. Nalezení aktivního elementu

SmartTV aplikace musí být schopna uživateli jasně zobrazit, který element je v danou chvíli aktivní. Protože uživatel při ovládání televize nepoužívá myš, ale televizní ovladač. Aby mohl uživatel televizi ovládat, musí vědět, na kterém elementu se zrovna nachází a kam se dostane stiskem tlačítka vpravo, vlevo, nahoru nebo dolů. Z tohoto důvodu je aktivní element vždy nějakým způsobem graficky odlišen od ostatních elementů na obrazovce, např. změnou barvy, odstínu, nebo ohraničení daného elementu. Pro definování zobrazení elementů ve webových aplikacích se používá jazyk CSS. Jazyk CSS využívá systém selektorů, podle kterých vybere element na stránce a definuje pro něj zadaný vzhled. Mezi selektory patří například, typ elementu, atribut id nebo atribut třída. Třída se často využívá ke změně vzhledu elementu za běhu programu, protože každý element může obsahovat více tříd a díky tomu mohu měnit jenom část vzhledu. Z tohoto důvodu je jednou z možností pro zobrazení aktivního elementu

vytvořit třídu s názvem “active”, která má definovaný vzhled v CSS, a při změně aktivního elementu tuto třídu přesunout k novému elementu. Jelikož jméno pro aktivní třídu není nijak standardizováno, a aplikace může využívat více tříd pro definování aktivního elementu, musí názvy aktivních tříd zadat do frameworku jeho uživatel. Java Script usnadňuje vyhledání elementů podle třídy díky funkci *document.getElementsByClassName (className)*. Pokud framework nalezne více elementů s aktivní třídou, musí určit, který z nich se právě zobrazuje uživateli. Elementy zobrazující se na obrazovce nesmí mít vlastnost dispaly nastavenou na none a vlastnost visibility na hidden. Pokud má element definovanou aktivní třídu a splňuje podmínku pro zobrazení, jedná se o aktivní element.

Druhou možností pro zobrazení aktivního elementu je využití vlastnosti DOM activeElement. DOM nám umožňuje nastavit jeden element jako aktivní a to pomocí funkcí *focus(), blur() a activeElement().* Funkce *element.focus()* nastaví daný element jako aktivní, funkce *element.blur()* změní element na neaktivní a *document.activeElement()* vrátí právě aktivní element. Využitím funkcí *focus() a blur()* je možné dosáhnout podobného výsledku jako při používání tříd v první možnosti. V tomto případě je pro framework nalezení aktivního elementu jednodušší, protože se stačí dotázat pomocí funkce *document.activeElement().*

Často vývojáři používají kombinaci obou výše zmíněných možností, kdy DOM activeElement využívají pro zachytávání eventů od uživatele a k zobrazení aktivního elementu využívají třídy. Z tohoto důvodu framework využije funkce *document.activeElement()* pouze v případě, že nenalezne žádný element, který by měl aktivní třídu a zároveň byl viditelný.

4.3.3. Vyhledávání elementů

Pro vytvoření modelu aplikace je potřeba systematicky projít testovanou a aplikaci a zaznamenat všechny elementy. Vyhledávání funguje následujícím způsobem. Po spuštění vyhledávání se uloží právě aktivní element a framework začne simulovat stisk tlačítek na ovladači bez použití tlačítka enter. Po každém stisknutí tlačítka musí framework počkat, na změnu elementů uvnitř DOM, a poté vyhledat nový aktivní element, který uloží. Takto pokračuje, dokud nenalezne všechny elementy na stránce. Toto omezení je nastaveno kvůli aplikacím, které načítají data z internetu a počet elementů by tam mohl převýšit kapacity frameworku. Takovýmto případem je mimo jiné seznam videí na https://www.youtube.com/. Jakmile nalezne všechny elementy na dané stránce, začne pro každý element simulovat stisk tlačítka enter. Pokud se po stisku tlačítka enter přesune na novou stránku, opakuje se vyhledávání elementů na dané stránce. Tento celý proces běží, dokud existují nové nenalezené objekty. Po skončení práce framework vrátí uživateli pole Views s nalezenými elementy.

4.3.4. Detekce změn uvnitř DOM

Každá aplikace potřebuje na změnu uvnitř DOM nějaký čas. Pokud chceme po stisku tlačítka detekovat nový aktivní element musíme počkat, dokud aplikace neprovede všechny změny. Nejjednodušší způsob je nastavit frameworku nějakou konstantní dobu, kterou bude po každém kliknutí čekat, než začne vyhledávat aktivní element. Ovšem toto řešení má zásadní problém, a to je určení doby čekání. Různým aplikacím trvají změny uvnitř DOM různě dlouho a různé změny ve stejné aplikaci mohou být také různě dlouhé. Jak tedy definovat dobu, po kterou by měl framework čekat, aby se všechny změny stihly provést? Můžeme například říct, že existuje doba, za kterou by každá aplikace měla dokončit všechny změny, jinak by se stala pro uživatele prakticky nepoužitelnou. Například kdyby uživatel musel po každém stisku tlačítka počkat 20 sekund, než by mohl pokračovat, aplikaci by nikdo nechtěl používat. S tímto předpokladem má framework jistotu, že pokud vyčká 20 sekund, budou všechny změny provedeny, a tím pádem by neměl mít problém nalézt nový aktivní element. Ovšem toto řešení framework velice zpomalí, protože většina změn je provedena v rámci stovek milisekund a jen opravdu složité změny trvají déle. Z tohoto důvodu je lepší detekovat, zda se provádějí změny uvnitř DOM či nikoliv a podle toho pokračovat.

V javascriptu existuje nástroj MutationObserver[7], který pozoruje DOM a reaguje na změny uvnitř DOM. V MutationObserveru můžeme nastavit funkci, která se zavolá při změně DOMu. Díky tomu jsme schopni detekovat, zda se právě provádí změny DOMu či nikoliv. Ve frameworku pracuje MutationObserver tak, že pokud zaznamená změnu uvnitř DOMu, tak změní značku isModified na true. Detekci změny představuje [Script 1]. Framework po stisknutí tlačítka čeká 200 milisekund a poté zjišťuje, jestli se DOM mění, pokud ano čeká dalších 200 milisekund. Tímto způsobem čeká tak dlouho dokud se DOM nepřestane měnit. Díky tomuto přístupu je framework schopen snadno detekovat okamžik, kdy se dokončily změny v DOMu a on může dále pokračovat.

```
function detectDOMChange(){
    if (isModified) {
       isModified = false;
       wait(200);
       detectDOMChange()
    }
    else{
       resolve()
    }
}
```

Script 1 - Detekce změn uvnitř DOM

[7]Více na https://developer.mozilla.org/en-US/docs/Web/API/MutationObserver

4.3.5. Detekce dotazů na server

Některé aplikace nemají všechna data, proto aby mohly zobrazovat nové informace, musí během svého běhu komunikovat se serverem, který jim nová data poskytne. Tato komunikace je většinou časově náročná. Aby nezdržovala běh celé aplikace, často probíhá asynchronně. Framework musí umět pracovat i s daty, která přicházejí ze serveru. Aby mohl pracovat i s těmito daty musí být framework schopen detekovat dotazy aplikace na server a počkat, dokud server nová data neposkytne. Jelikož framework nepracuje s kódem aplikace, je problém komunikaci se severem zachytávat, protože v javascriptu neexistuje nástroj, který by umožnoval detekci komunikace mezi aplikací a serverem. Framework musí ten nedostatek obejít, aby mohl komunikaci detekovat.

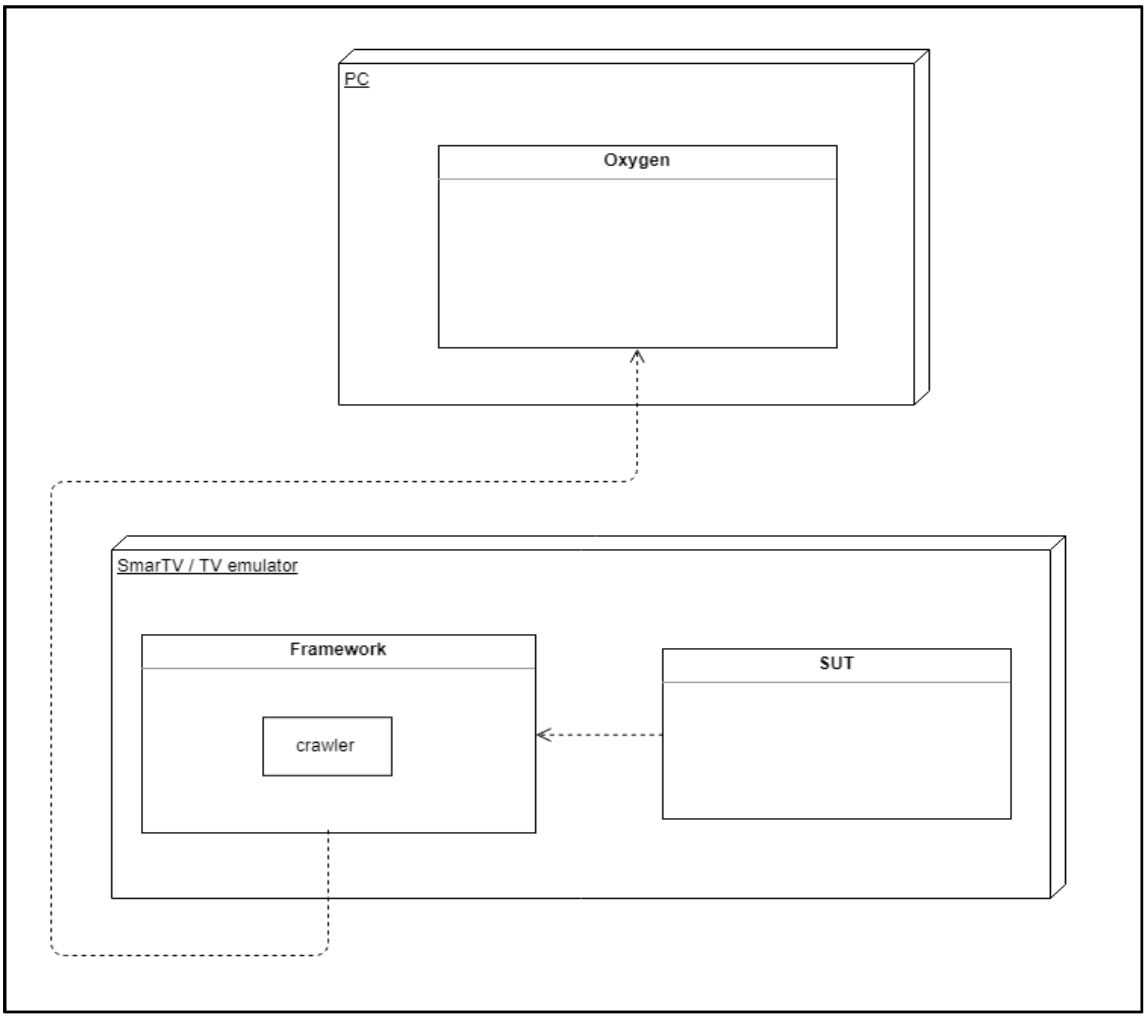
Je několik způsobů, jakým může aplikace komunikovat se serverem. Velmi používané řešení je pomocí nástroje XMLHttpRequest a proto je framework zaměřený na detekci těchto dotazů. Nástroj XMLHttpRequest umožňuje odesílat http dotazy na server a reagovat na odpověď. Aby mohl framework detekovat tyto dotazy, využívá vlastnosti javascriptu, ve kterém jsou všechny třídy a funkce sdíleny globálně. Díky tomu je možné javascriptové nástroje pozměnit. Framework přímo do nástroje XMLHttpRequest přidává počítadlo, které počítá komunikaci mezi aplikací a serverem a díky tomu je framework schopen detekovat, jestli aplikace čeká na nějaká data či nikoliv.

4.3.6. Ukončení vyhledávání pro velké aplikace

Některé aplikace mohou pořád dokola načítat data z internetu, i když pro model jsou tato data nepotřebná (například seznam videí na https://www.youtube.com). Anebo je aplikace moc velká a potřebujeme namodelovat jen některé její části. Z těchto důvodů framework umožňuje omezit počet namodelovaných aktivních elementů na jedné stránce a také umožňuje modelovat jen některé části testované aplikace. Počet elementů na jedné stránce může tester nastavit při spouštění modelování. Framework poté počítá, kolik elementů na dané stránce již objevil a pokud se dostane na zadanou hodnotu, další elementy na této stránce už nezaznamenává a pokračuje v modelovaní na další stránku. Pokud chce tester namodelovat pouze část aplikace stačí mu spustit modelování na stránce s danou částí aplikace. Framework modeluje stránky pouze směrem dolů. Všechny stránky, které jsou nad vybranou stránkou (pod tlačítkem back) nebudou namodelovány. Například na Obrázek 6 vidíme dvě stránky, pokud modelování spustíme z některého view na druhé stránce, views na první stránce nebudou namodelovány.

4.4. Návod na nasazení

Framework se vkládá jako script přímo do testované aplikace, která běží buď na vývojářském počítači uvnitř emulátoru nebo uvnitř chytré televize, které je spuštěna v debug modu. Framework vytváří model pro nástroj Oxygen. Oxygen je klasická desktopová aplikace, čili běží na počítači a ne uvnitř chytré televize. Jak jsou jednotlivé komponenty uspořádány zobrazuje diagram nasazení na Obrázek 7.



Obrázek 7 - Diagram nasazení

4.4.1. Prerekvizity

1. Samsung Tizen Studio s rozšířením pro TV
2. Samsung TV emulator je součástí Tizen Studia viz kapitola 4.1.2
3. Oxygen viz kapitola 3.4.2

4.4.2. Nasazení

1. Vložte javascriptový soubor tester.js s frameworkem do složky s ostatními scripty aplikace.
2. Do úvodního html souboru (většinou se jménem index.html) vložte následující script:

```
<script>
var TesterFocusClass = ["focused", "active", "active-button"];
</script>
<script src="Tester.js" id="test_sctipt">test</script>
```

Script 2 - Načtení testeru do aplikace

3. Do pole TesterFocusClass vepište všechny třídy, které definují aktivní elementy uvnitř aplikace. Pokud toto pole zůstane prázdné bude framework vyhledávat elementy pouze podle vlastnosti focus.
4. Do atributu src vepište cestu k souboru Tester.js, který jste uložili v kroku 1.
5. Aplikaci povolte práva pro práci se soubory a dálkovým ovladačem:
    a. Do souboru config.xml přidejte následující oprávnění:

```
<tizen:privilege name='http://tizen.org/privilege/filesystem.read'/>
<tizen:privilege name='http://tizen.org/privilege/filesystem.write'/>
<tizen:privilege name='http://tizen.org/privilege/tv.inputdevice'/>
```

Script 3 - Přidání práv aplikaci

6. Můžete aplikaci spustit
    a. Pokud aplikaci pouštíte na reálném zařízení, musíte jí spustit v debug modu[8], abyste mohli připojit web Insector. K zařízení je nutné připojit externí paměť např. flash disk, do které se budou ukládat výsledky.
    b. Pokud aplikaci pouštíte v emulátoru musíte v control panelu emulovat externí USB zařízení, do kterého se budou ukládat výsledky.

4.5. Ovládání implementovaného frameworku

Framework přidává k testované aplikaci několik funkcí, které se dají použít přímo ve scriptech testované aplikace, anebo se mohou spouštět jednotlivě z konzole uvnitř nástroje Web inspektor. Tyto funkce slouží k procházení aplikace s vytvořením modelu a k uložení modelu ve formátu CSV do externí paměti. Pro správné fungování crawleru je velice důležitá proměnná "TesterFocusClass", která se vytváří při nasazení frameworku. Jedná se o pole, které musí obsahovat názvy všech html tříd, které jsou použity pro definování právě aktivního elementu.

[8] https://developer.samsung.com/tv/develop/getting-started/using-sdk/tv-device

Pokud bude toto pole špatně definováno, crawler bude špatně vyhodnocovat aktivní elementy a vytvořený model aplikace bude neúplný.

4.5.1. Crawling

Pro vytváření modelu se používá funkce crawlerStart(stopCriterium). Crawler se snaží načíst celou aplikaci. Pokud jsou v aplikaci například seznamy napojené na databázi, může se stát, že crawler bude načítat nová a nová data ze seznamu, dokud nenamodeluje celou databázi. Z tohoto důvodu funkce crawlerStart obsahuje parametr stop kritérium, kterým se nastaví počet elementů na jedné stránce, například po spuštění crawlerStart(20) bude model na každé stránce obsahovat 20 elementů. Před spuštěním této funkce přejděte v testované aplikaci na element, od kterého chcete začít. Crawler začne vyhledávání od vámi zvoleného elementu. Tento element považuje za součást nejvyšší vrstvy aplikace, takže pokud se vámi vybraný element nenachází například na úvodní stránce, crawler vytvoří model až od vrstvy, na které jste element vybrali a vyšší vrstvy nebudou v modelu zaneseny. Tuto vlastnost můžete využít, pokud nechcete zaznamenat model celé aplikace, ale jen její část. Po spuštění uvidíte, že crawler „proklikává“ aplikaci. V této době nijak nezasahujte do běhu aplikace, moli byste ovlivnit výsledný model. Po skončení prohledávání crawler vypíše do konzole „FINISH“ a zobrazí model testované aplikace.

4.5.2. Ukládání modelu

Pro ukládání výsledného modelu se využívá externího uložiště, aby bylo snadné model přenést z televize do počítače pro další zpracování. Pokud pro testování používáte reálné zařízení musíte k televizi připojit externí uložiště. Pokud pro testování požíváte emulátor na počítači, je možné externí uložiště také emulovat. Jak pracovat s emulátorem je popsáno v dokumentaci k TV emulátoru [21]. Jestli je externí uložiště připojeno zjistíte pomocí funkce storag(), která zobrazí všechna dostupná externí uložiště. Po vytvoření modelu a připojení uložiště, můžete model uložit pomocí funkce saveModel(path). Jako parametr použijte cestu k adresáři, do kterého chcete model uložit např. saveModel(“externDevice/models/test_app”). Tato funkce vytvoří v zadaném adresáři CSV soubor s názvem model_date.csv. Tento soubor lze importovat jako model do aplikace Oxygen pro další zpracování.

5. Testování

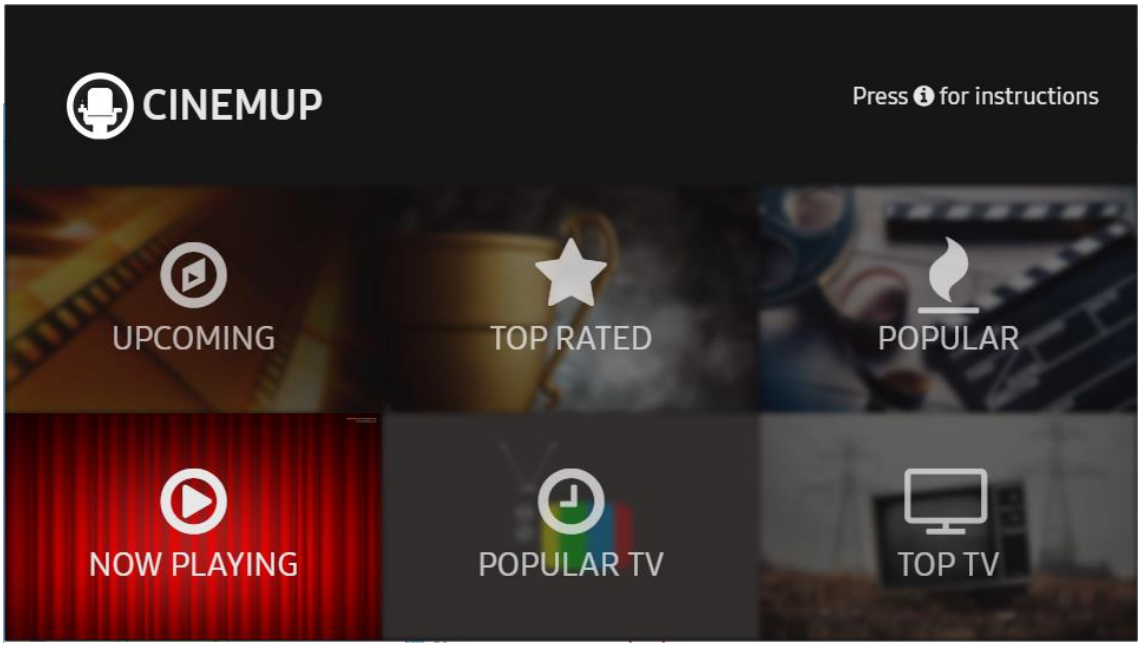
5.1. Výběr testovacích aplikací

Pro vývoj testovacího frameworku je zapotřebí mít vhodnou aplikaci, na které se budou testovat funkce frameworku. Jedna z možností je naimplementovat aplikaci tak, aby splňovala požadavky potřebné pro testování frameworku. Druhou možností je najít open-source aplikaci, která bude mít zveřejněné zdrojové kódy a následně jí upravit pro testování frameworku. Pro tuto práci byla vybrána druhá možnost, protože vyvíjet novou aplikaci, i když zjednodušenou, je mnohem náročnější než upravit existující aplikaci. A přínosy k projektu jsou oproti druhé možnosti minimální. Navíc aplikace vytvořená takzvaně na míru testovacího frameworku, může vytvořit příliš přívětivé podmínky pro framework a naším cílem je, aby framework fungoval obecně nejen na speciálně vytvořené aplikaci. Následující tři aplikace byli vybrány pro otestování funkčnosti framewoku.

5.1.1. Cinemup

Aplikace Cinemup[9] je vytvořena pro vyhledávání nejpopulárnějších a nejnovějších filmů nebo televizních seriálů (viz. Obrázek 8). Umožňuje vytřídění podle nejrůznějších kategorií a všechna data stahuje z filmové databáze https://www.themoviedb.org/. Aplikace je velice vhodná pro naše použití, jelikož má několika stupňové grafické rozhraní, takže je dostatečně rozsáhlá, používá asynchronní načítání dat z externí databáze a podle získaných dat asynchronně mění grafické rozhraní. Navíc její vnitřní struktura je velice vhodná pro testování frameworku, protože pro ovládání aktivních elementů používá několik přístupů. Díky tomu je možné framework přizpůsobit všeobecnému použití. Pro ovládání aktivních elementů aplikace využívá framework caph-jquery, což je velice používaný framework vytvořený společností Samsung pro ovládání aplikacích běžících na systému Tizen. Dále má aplikace zabudovaný vlastní systém ovládání, který používá na automaticky generované prvky. Tyto dva systémy fungují na principu přiřazování aktivní třídy danému elementu uvnitř DOM. Poslední technologie, kterou aplikace využívá je změna aktivního elementu uvnitř DOM pomocí javascriptových funkcí *focus()* a *blur().* Díky těmto vlastnostem aplikace představuje vhodné testovací prostředí pro vyvíjený Framework.

[9] https://github.com/daliife/Cinemup



Obrázek 8 – Ukázka z aplikace Cinemup

5.1.2. ChessLab

Aplikace ChessLab[10] je aplikací pro šachové hráče, kteří chtějí zlepšit své schopnosti. Aplikace obsahuje různé herní situace a jejich možnosti řešení tak, aby hráč došel k vítězství (viz. Obrázek 9). Pro testování frameworku byla vybrána, protože na rozdíl od aplikace Cinemup se neskládá pouze z jednoho html souboru, který je pomocí javascriptu upravován, ale každá herní situace je zobrazována na nové stránce. Z tohoto důvodu se nejedná o úplně klasickou SmartTV aplikaci a tím pádem je vhodné ji využít k testování tohoto frameworku.

Obrázek 9 – Ukázka z aplikace ChessLab

[10] https://github.com/PabloEzequiel/Tizen/tree/master/ChessLabTV

5.1.3. Memoryyyyyy

Aplikace Memoryyyyyy[11] je velice jednoduchou aplikací. Jedná se pouze o hru pexeso s náhodně generovanými pořadím hracích karet (viz. Obrázek 10). Důvodem, proč je aplikace Memoryyyyyy použita pro testování frameworku je, že se jedná o nedokončenou aplikaci s chybami v ovládání. Pokud z jedné karty přejdeme na jinou kartu pomocí tlačítka down, měli bychom se vrátit zpět pomocí tlačítka up, ovšem uvnitř této aplikace to takto nefunguje, a proto nelze aplikaci systematiky projít. Aplikace má rozbitý datový model aktivních elementů, a proto se hodí tento jev otestovat.

Obrázek 10 – Ukázka z aplikace Memoryyyyyy

5.2. Průběh testů

Pro testování frameworku byly vytvořeny testovací scénáře popsané v kapitole 5.2.1. Z každé testované aplikace byl model vytvořen 5krát vždy s jiným stopovacím kritériem a od různých počátečních elementů. Některé počáteční elementy nebudou z hlavní stránky, aby se otestovala schopnost frameworku vytvářet modely vybraných částí aplikace. Po vytvoření modelů se porovnávali testované aplikace se svým modelem na základě testovacích scénářů. Rozdíly mezi modelem a testovanou aplikací jsou považovány za chybu.

[11] https://github.com/wissalKhalfi/Brain-Up---Tizen-smart-TV/tree/master/Memoryyyyyy

5.2.1. Testovací scénáře

1. **Zaznamenání konkrétní elementu do modelu**

   Tento testovací scénář má za úkol zjistit, zda se elementy objevující se v aplikaci zaznamenali do modelu. Nejprve vybereme element z aplikace a poté zjistíme, zda se element nachází i uvnitř modelu.

2. **Kontrola, že názvy elementů odpovídají testované aplikaci**

   Tento testovací scénář má za úkol zjistit, zda jsou elementy v modelu správně označeny a zda element v modelu jednoznačně určuje stejný element v aplikaci. Vybereme element z aplikace a zkontrolujeme, zda popis vybraného elementu je stejný s popisem jeho protějšku v modelu.

3. **Zaznamenání přechodu mezi dvěma elementy do modelu**

   Tento testovací scénář má za úkol zjistit, zda jsou jednotlivé elementy v modelu správně propojeny. Vybereme dva mezi sebou propojené elementy z aplikace a poté zjistíme, jestli se tyto dva elementy nachází v aplikaci a zdali jsou propojeny stejnou vazbou.

4. **Zaznamenání vrácení se na již objevenou obrazovku**

   Tento testovací scénář má za úkol zjistit, jestli všechny elementy, ze kterých se lze vrátit na předchozí obrazovku jsou v modelu zaznamenány s vazbou back. Vybereme element z předchozí obrazovky a zkontrolujeme, zda je spojen vazbou back se všemi elementy, ze kterých se v aplikaci lze tlačítkem back vrátit na vybraný element z předchozí obrazovky. Tento testovací scénář budeme testovat pouze na aplikaci Cinemup, protože aplikace ChessLab a Memoryyyyyy pracují pouze s jednou obrazovkou a vazba back u nich neexistuje.

5. **Kontrola, že se systém zastavil podle zadaného kritéria**

   Tento testovací scénář má za úkol zjistit, zda správně funguje stopovací kritérium. Každý model vytvoříme s jiným stopovacím kritériem a budeme sledovat, zda velikost vytvořeného modelu odpovídá stopovacímu kritériu.

6. Kontrola, schopnosti vytvořit pouze částečný model

Tento testovací scénář má za úkol zjistit, zda je systém schopen vytvořit model pouze vybrané části z aplikace. Při vytváření modelu bude vybírat počáteční element z různých částí aplikace a budeme sledovat, jestli vytvořený model neobsahuje některé elementy navíc. Tento testovací scénář budeme testovat pouze na aplikaci Cinemup, protože aplikace ChessLab a Memoryyyyyy pracují pouze s jednou obrazovkou.

7. Kontrola, že cesta zaznamenaná v modelu lze projít v testované aplikaci

Tento testovací scénář má za úkol zjistit, zda jsou jednotlivé elementy v modelu propojeny stejnými vazbami jako elementy v aplikaci. V grafu modelu vytvoříme cestu a poté se tuto cestu pokusíme projít v aplikaci a budeme sledovat, zda jsme schopni touto cestou projít.

5.3. Výsledky testů

Během tetování bylo vytvořeno 5 modelů pro každou testovanou aplikaci, přičemž každý model byl vytvářen s jinými vstupními parametry. Na každém modelu byly provedeny všechny testovací scénáře, kromě scénářů 4 a 6, ty byly prováděny pouze na modelech aplikace Cinemup. Pro aplikace Cinemup a ChessLab testovací scénáře neobjevily žádné chyby, jednotlivé modely vždy odpovídaly reálnému stavu uvnitř testovacích aplikací.

Framework měl problém s vymodelováním aplikace Memoryyyyyy. Při testování bylo zjištěno, že aplikace obsahuje 13 aktivních elementů ovšem vytvořené modely obsahovaly pouze 11 aktivních elementů. Framework nebyl schopen nalézt všechny aktivní elementy uvnitř aplikace Memoryyyyyy. Z tohoto důvodu testovací scénáře 1 a 3 u aplikace Memoryyyyyy neprošly. Důvodem této chyby je rozpadlý datový model aplikace popsaný v kapitole 5.1.3. Framework si s tímto případem neumí poradit, a proto není schopen model Memoryyyyyy vytvořit správně.

6. Závěr

Hlavním cílem této práce bylo vytvořit framework pro nalezení aktivních elementů uvnitř SmartTV aplikací a zefektivnit tak samotné testování chytrých televizí.

Práce nejprve popisuje různé přístupy k testování aplikací a dochází k zjištění, že zatím neexistuje mnoho řešení pro testování chytrých televizí, a proto je dobré vytvořit systém, který bude testování aplikací pro chytré televize usnadňovat. Po této úvaze byly stanoveny požadavky na systém a byl vytvořen návrh, jakým způsobem by měl systém fungovat.

Během implementace frameworku byly nalezeny nejrůznější technické překážky, které bylo potřeba pro vytvoření frameworku překonat. Pro překonání těchto problémů bylo nutné dopodrobna analyzovat fungování aplikací pro chytré televize. Vyřešení těchto problémů bylo nejsložitější částí celé práce a výsledná řešení přináší cenné poznatky pro práci s chytrými televizemi.

Následně po vytvoření byl framework otestován na třech rozdílných aplikacích. Každá aplikace byla vybrána tak, aby její implementace byla rozdílná od ostatních testovacích aplikací. Jednotlivé testy měly za úkol zjistit, zda je framework schopen vytvořit model odpovídající testovací aplikaci. Testy objevily, že pokud má aplikace mezi sebou špatně propojené elementy, framework nemusí nalézt všechny elementy a vytvoří pouze neúplný model aplikace. V ostatních případech framework funguje správně.

Cíl práce byl úspěšně naplněn. Výsledkem je funkční framework, který dokáže nasimulovat ovládání chytré televize ovladačem, jako by ho užíval sám uživatel. Během procházení aplikace nalezne a uloží aktivní elementy. Framework je schopen po zadání kritérií aktivního elementu v dané aplikaci, vytvořit schéma aplikace popisující vnitřní strukturu jejích elementů a získaná data poskytnout aplikaci Oxygen k dalšímu zpracování. Tento framework funguje obecně na aplikacích běžících na SmartTV televizorech se systémem Tizen.

Framework je dále možné rozšířit o testovací část. Jelikož framework již umí ovládat testované aplikace, nabízí se možnost rozšířit jeho funkce i o automatické testování. V budoucnu by mohl být framework schopen načítat testovací scénáře vytvořené pomocí aplikace Oxygen a pomocí nich následně testovat zadanou aplikaci.

Zdroje

[1]. Rafi, D.M. et al.: 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, 2012, pp. 36--42.

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

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

[4]. 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.

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

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

[7]. Fewster, M., Common Mistakes in Test Automation. White paper, Groove consulatns, 2001. http://www.agileconnection.com/sites/default/files/article/file/2012/XDD2901filelistfilename1_0.pdf

[8]. 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.

[9]. Fewster, M., Graham, D.: Software Test Automation: Effective Use of Test Execution Tools. Addison-Wesley Professional. ACM Press Books (1999)

[10]. 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.

[11]. 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.

[12]. 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.

[13]. 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.

[14]. Bulankina, V., Ogeenko, P. Y., & Morozova, T. I. (2014). Smart TV–Advanced Features.

[15]. Bestoun S. Ahmed, Miroslav Bures, "Testing of Smart TV Applications: Key Ingredients, Challenges and Proposed Solutions", Future Technologies Conference (FTC2018), In Press, 2018.

[16]. Mikowski, M. S., & Powell, J. C. (2013). Single page web applications. *B and W*.

[17]. Tv design principles. Samsung developers [online]. [cit. 2018-01-09]. *Dostupné z: http://developer.samsung.com/tv/design/design-principles.*

[18]. Miroslav Bures: 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, p. 789-794, Springer, 2015.

[19]. Miroslav Bures, Tomas Cerny and Matej Klima: Prioritized Process Test: More Efficiency in Testing of Business Processes and Workflows. In ICISA 2017, LNEE vol.424, p. 585-593, Springer, 2017.

[20]. Matulac, J. S. Case Study of Tizen Operating System.

[21]. TV Emulator. Samsung Developers [online]. [cit. 2018-05-11]. Dostupné z: https://developer.samsung.com/tv/develop/getting-started/using-sdk/tv-emulator

[22]. Shannon, R. (2007). What is HTML. YourHTMLSource, c2000-2009 [cit. 2009-04-24]. *Dostupný z: http://www.yourhtmlsource.com/starthere/whatishtml.*

[23]. Flanagan, D. (2006). *JavaScript: the definitive guide*. " O'Reilly Media, Inc.", kapitola 1 “Intoduction to JavaScript”.

Seznam příloh

- **Zdrojové kódy:** Javascriptový soubor s frameworkem
- **Psaný projev:** Elektronická verze bakalářské práce