VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

NÁSTROJ PRO PODPORU MANUÁLNÍCH GUI TESTŮ

GUI TEST DEVELOPMENT SUPPORT TOOL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **MARIE BEDNÁŘOVÁ**
AUTHOR

VEDOUCÍ PRÁCE — **ADAM ROGALEWICZ, doc. Mgr., Ph.D.**
SUPERVISOR

BRNO 2021

Vysoké učení technické v Brně
Fakulta informačních technologií

Ústav inteligentních systémů (UITS) Akademický rok 2020/2021

Zadání bakalářské práce

Studentka: **Bednářová Marie**
Program: Informační technologie
Název: **Nástroj pro podporu manuálních GUI testů**
Nástroj pro podporu manuálních GUI testů
Kategorie: Analýza a testování softwaru

Zadání:

1. Nastudujte principy automatizovaného testování grafického uživatelského rozhraní, zaměřte se na testování aplikací pomocí služby WinAppDriver.
2. Navrhněte aplikaci usnadňující tvorbu reportů z výsledků GUI testů. Aplikace by měla automaticky generovat report o jednotlivých krocích testů. Report by měl zahrnovat prováděné uživatelské akce (např. kliknutí na objekt, psaní klávesnice) nad testovaným subjektem a snímky uživatelského rozhraní v průběhu testu.
3. Implementujte aplikaci v jazyku C#. Aplikace by měla využívat definice regresních testů z WinAppDriver Recorder.
4. Ověřte základní funkcionalitu pomocí automatických testů.

Literatura:

- Domovská stránka projektu WinAppDriver: https://github.com/microsoft/WinAppDriver
- SOJČÁK, Juraj. *Generátor testovacích běhů nad GUI*. Brno, 2019. Diplomová práce. Vysoké učení technické v Brně, Fakulta informačních technologií. 2019-06-18.

Pro udělení zápočtu za první semestr je požadováno:

- První dva body zadání

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Rogalewicz Adam, doc. Mgr., Ph.D.**
Vedoucí ústavu: Hanáček Petr, doc. Dr. Ing.
Datum zadání: 1. listopadu 2020
Datum odevzdání: 12. května 2021
Datum schválení: 11. listopadu 2020

Abstrakt

Manuální testování aplikací je účinným, ale časově náročným procesem. Tato práce se bude zabývat metodami a technologiemi, které by umožnily zopakovat manuální postup bez přítomnosti člověka a následně z průběhu vytvořit report se zhodnocením průběhu. V řešení byla zvolena metoda testování na základě znalosti aplikace. Spolu s technologií WinAppDriver bylo možné navrhnout framework, který umožní vykonávat nahrané postupy v podobě testovacích běhů, sbírat informace z jejich průběhu a vytvářet report. Výsledkem této práce je dvojice nástrojů, která umožní automaticky spouštět speciálně vytvořené testy nad danou aplikací. Zároveň byly zahrnuty i akce, které jsou potřebné vykonat před spuštěním testů tak, aby bylo možné celý proces provádět bez přítomnosti člověka.

Abstract

Manual testing of application is very effective, but at the same time very time consuming process. This thesis will deal with methods and technologies, which would allow computers to not only record and rerun the process without the human assistence or intervention but also make an outcome of this process, which will be in form of report with results of the task run. In the final solution the method of testing was chosen based on the knowledge of application. Within this thesis the design of framework was created with the help of the WinAppDriver technology, which is able to execute the recorded methods as test runs during which the informations about the process itself are being collected. The final report is made, based on the collected information. The outcome of this thesis is a pair of tools, which can automatically execute specifically designed GUI application tests. There were simultaneously included actions, which are necessary to perform before executing the tests. Therefore the program will be able to operate without the need of human assistance.

Klíčová slova

GUI.regresní testování.WinAppDriver.report.konzole.framework

Keywords

GUI.regression testing.WinAppDriver.report.console.framework

Citace

BEDNÁŘOVÁ, Marie. *Nástroj pro podporu manuálních GUI testů.* Brno, 2021. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce Adam Rogalewicz, doc. Mgr., Ph.D.

Nástroj pro podporu manuálních GUI testů

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracovala samostatně pod vedením pana Adama Rogalewicze doc. Mgr., Ph.D.. Další informace mi poskytl Ing. Dalibor Lupínek. Uvedla jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpala.

.....
Marie Bednářová
7. května 2021

Poděkování

Chtěla bych tímto poděkovat mému vedoucímu Adamu Rogalewiczi doc. Mgr., Ph.D. za vedení práce a rady při jejím psaní. Dále bych chtěla poděkovat mému kolegovi, týmovému vedoucímu a konzultantovi Ing. Daliboru Lupínkovi za rady a nápady ohledně zadání a řešení práce.

Obsah

Kapitola 1

Úvod

Grafické uživatelské rozhraní (anglicky Graphical User Interface, dále GUI) je prostředníkem, kterým lze ovládat počítač díky interaktivním grafickým ovládacím prvkům. Jde také o způsob abstrahování informací obsažených v systému tak, aby jim rozuměl uživatel. Poprvé se koncept GUI objevil v roce 1973 v laboratořích firmy Xerox ve formě systému WIMP (window, icon, menu, pointer). Největší boom však nastal až v roce 1984 s příchodem počítačů Macintosh firmy Apple.[26] Dnes se s GUI setkáváme nejen na počítačových zařízeních, ale i na přenosných zařízeních (mp3, chytré telefony), herních zařízeních, kancelářských a průmyslových vybaveních, domácích spotřebičích, ale dokonce i na chytrých hodinkách.

Tato práce se zabývá řešením, které má pomoci s testováním desktopových aplikací skrze jejich grafické rozhraní. V kapitole 2 bude stručně představeno testování jako obor, jeho rozdělení, důležité pojmy z tohoto oboru a nejznámější metody testování nejen GUI. Kapitola 3 přiblíží GUI testování na základě znalosti aplikace a mimo jiné zde budou představeny existující technologie a frameworky. V kapitola 4 bude krátce zmíněno zadání, které specifikovalo téma celé této práce a důvod proč se práce takovému tématu věnuje. V kapitole 5 bude provedena analýza požadavků a představeno řešení, které má požadavky splňovat. V kapitole 6 budou zmíněny některé implementační detaily řešení. A v kapitole 7 bude představeno testování vytvořených nástrojů.

Kapitola 2

Testování a GUI

S používáním GUI a jeho vývojem v rámci softwarového inženýrství přichází i nutnost jeho testování. GUI je zásadní částí softwaru, může být jeho ovladačem nebo pouhým prostředníkem při zobrazování dat. Co je zajímavější, přes GUI se dají odhalit chyby a nedostatky i na úrovni samotného systému. Následující kapitola se bude věnovat nejen testování, ale i teorii testování softwaru obecně a samotného GUI. V této kapitole budou představeny techniky a způsoby, které se dnes využívají a které budou v rámci řešení inspirací.

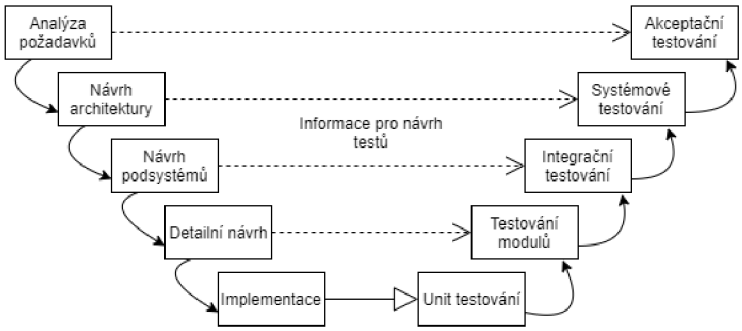
2.1 Testování softwaru

Testování ve vývoji softwaru hraje důležitou roli. Lze jej definovat jako systematické prozkoumávání softwaru za účelem odhalení chyb. Ty by poté měly být zdokumentovány a nahlášeny vývojovému týmu na opravu.[23] Testování softwaru také úzce souvisí s tzv. verifikací. Verifikaci lze chápat jako aktivitu, která využívá znalosti konkrétního softwaru, jeho struktury, návrhu a především specifikace.[17] Tyto informace jsou pak využity k systematické kontrole softwaru, zda je splňuje. To, jak bude kontrola testování prováděna, se odvíjí od „testovací strategie“. Testovací strategie je formální výstup přípravné fáze systému a je vytvářena na základě popisu zkoumaného systému. Popisuje co a jak bude v systému testováno, jak často, jak se budou výsledky testů interpretovat a metrika podle které se bude hodnotit správnost systému.[14] Poté lze aplikovat různé přístupy a metody.

V následujících částech budou zmíněny některé z metod verifikace, které budou v rámci práce stěžejní.

2.1.1 Přístupy verifikace a V - model

Testování softwaru je v rámci přístupů verifikace řazené mezi *Dynamickou analýzu*. Dynamickou analýzou jsou ověřovány vlastnosti softwaru na základě vykonávání kódu.[35] Testování je prováděno za účelem zvýšení kvality softwaru jeho spouštěním. Software je spouštěn s reálnými vstupními hodnotami a studuje se jeho chování. Ve většině literatury je pojem „testování“ spojováno právě s tímto způsobem. A v dalších částech této práce bude pojem používán stejně.



Obrázek 2.1: V-model jako znázornění fází vývoje softwaru souvisejících úrovní testování. Převzato z [17]

Jak je ale vidět například u V-modelu (viz obrázek 2.1), lze takovou analýzu provádět až od určité fáze vývoje.[17] Například pokud není aplikace ve spustitelném stavu, není možné na ni provádět některou z dynamických testovacích metod. Lze však použít některou z metod Statické analýzy. Ta zkoumá software bez toho, aniž by byl spuštěn. Například inspekcí kódu, kterou může provádět člověk, nebo některou z metod Formální verifikace, ale tou se tato práce zabývat nebude.

I přes to je doporučováno vytvářet a navrhovat testy konkrétně s každou vývojovou aktivitou. Například lze již při návrhu explicitně specifikovat případy užití podsystému nebo způsob komunikace mezi moduly.[17] Tyto informace pak poslouží jako plány a návrhy k vytvoření jednotlivých testovacích procesů, nebo lze ještě neimplementované části nahradit simulací jejich chování.[37] Tento přístup je klíčový v případě chyb, které by se v softwaru jinak zdály jako očekávané. A čím dříve je chyba v návrhu odhalena a opravena, tím méně času a peněz bude nutné investovat do zkvalitnění vytvářeného softwaru.

Důležitou částí testování softwaru, kterou V-model nezobrazuje, je regresní testování.[37] Každá nová vývojová fáze přináší nové změny do již existujícího softwaru. Je nutné se ujistit, že se spolu s nimi nezanesly do řešení nové chyby. Regresní testování, prováděné na různých úrovních, má tyto hrozby odhalit.

2.1.2 Důležité pojmy

V rámci testování softwaru existují pojmy, které budou v práci dále používány.

Testovaný software je také nazýván jako *Testovaný systém* (System under test), zkráceně *SUT*. Podobně se dá nahlížet i na testování aplikace (AUT) nebo testování samotných jednotek (Unit testing).[17] K němu bývá k dispozici *specifikace*. Specifikace je popisem systému, jeho funkcí a korektního chování, spolu s popisem vstupních a výstupních parametrů.[35] Pokud systém vykazuje neočekávané nebo možné nežádoucí chování, nazývá se tento jev jako *bug*. Bug je obecným termínem vyjadřující problém se systémem nebo v systému.[35] Problém může způsobovat *vada*. Vadou se v softwaru považuje statický defekt. Příkladem vady může být špatná manipulace s polem hodnot typu číslo (integer).[35] Vada se pak může (i nemusí) projevit *chybou*, která označuje nekorektní vnitřní stav systému.[35] Fakt, že se chyba projevit nemusí, záleží na *dosažitelnosti* míst v kódu, které chyby produkují.[33] Pokud je jedno z takových míst dosaženo a provedeno, nastává *infikování*. Po provedení se systém musí nacházet v nekorektním stavu.[17] V tomto případě se chyba stále neprojevila externě. Může se však systémem rozšířit *propagaci* až na výstup systému a tím způsobit

jeho *selhání*.[35] Selhání je projevem nespecifikovaného nebo špatného chování systému, vzhledem k jeho specifikaci.[35]

Popsaný průběh možného selhání je v teorii testování znám jako model vada/selhání. Tři již zmíněné faktory, tedy dosažitelnost, infekce a propagace, jsou nutné pro to, aby systém kvůli vadě nakonec selhal. Pokud je ale vada neznámá, je pravděpodobné díky selhání vadu zpětně „vystopovat“. Tomuto procesu se jinak říká tzv. Ladění (Debbuging). Využitím znalosti selhání se hledá samotná vada. Náročnost a čas nutný pro její nalezení pak závisí na její podstatě.[17]

2.1.3 Testování v kontextu se SUT

Testování je v tomto případě chápáno jako proces vyhodnocování SUT sledováním jeho běhů.[35] Pokud testovací běh neskončil selháním systému, bude se jednat o test, který prošel (Test Passed). V opačném případě půjde o test, který selhal (Test Failure).

Každým testovacím během se spouští SUT, nebo aspoň jeho část. Aby se celý běh dokončil, je nutné předat v rámci testů i vstupní hodnoty. Ty jsou nazývány jako Hodnoty testovacího případu (Test case values). Z těchto hodnot, které systém zpracuje, mohou vyjít výstupní hodnoty. Aby bylo možné testy vyhodnotit, je nutné výstupní hodnoty porovnat s očekávanými výstupními hodnotami (Expected results).[17]

V některých případech test může vyžadovat určitý počáteční stav a je nutné SUT do takového stavu dostat. K tomuto procesu se dají využít tzv. *Prefixové hodnoty* (Setup values). Tyto hodnoty mají uvést SUT do vyžadovaného stavu, aby bylo možné test uskutečnit. Podobně jako předpřipravení testu, může být nutností provést některé aktivity po testu. Tyto hodnoty jsou tzv. *Postfixové hodnoty* (Teardown values). Ty je nutné SUT poslat hned potom, co byly odeslány Hodnoty testovacího případu. Všechny tyto typy hodnot tvoří *Testovací případ* (Test case). Více testovacích případů pak tvoří *Testovací sadu* (Test set).[17]

2.1.4 Automatizace testových aktivit

Testování je časově a zdrojově náročným procesem. Proto jeden z cílů testování softwaru je hledat řešení, která by některé procesy ulehčila. Jedním z kandidátů na automatizaci je kompilace zdrojového kódu, vygenerování vstupních hodnot testů. Taková řešení mohou snížit cenu testování, ušetřit čas a lidskou pracovní sílu. Zároveň se dá snížit pravděpodobnost zanesení lidské chyby do výsledků testů a ulehčit tak i provádění regresního testování. Ne však každý proces je možné jednoduše automatizovat, pokud vůbec.[17]

2.2 Testování grafického uživatelského rozhraní

Během vývoje moderních aplikací může na grafické rozhraní připadat i více jak polovina objemu zdrojového kódu. I přes to existuje jen málo technik a způsobů, jak tento objem kódu otestovat.[33] A o to těžší je testování automatizovat. Největším problémem u GUI je tzv. stavová exploze.[17] Pokud bude cílem otestovat aplikaci přes její grafické rozhraní, bude nutné nejdříve nalézt všechny stavy, ve kterých se systém může ocitnout. Každá interakce s GUI systém dostane do nového stavu. A zaznamenat všechny takové stavy může i na jednoduchém grafickém rozhraní vzít spoustu času a práce. Proto se následující části budou popsány způsoby a principy testování GUI.

2.2.1 Kategorie GUI testování

Grafické uživatelské rozhraní lze testovat z různých hledisek. Prvním je testování použitelnosti. Použitelnost je definována jako míra toho, jak použitelná je aplikace pro specifickou skupinu uživatelů. Charakteristickými znaky, které jsou stěžejní, jsou efektivnost, účinnost a spokojenost uživatelů v daném kontextu.[25] Jeden ze způsobů kontroly použitelnosti bude představen v části 2.2.2.

Oproti tomu funkční testování je bližší obecnému principu testování. Jde o kontrolu, zda aplikace jako celek funguje podle specifikace. Funkční testování lze rozdělit do více částí, ve kterých se důvod testování dále specifikuje.[33] Jednotlivé způsoby jsou představeny v následující tabulce:

Způsob testování	**Účel testování**
Systémové GUI testování	testování systému skrze GUI
Regresní testování	testování UI po nových změnách
Testování validity vstupů	schopnost softwaru rozpoznat a reagovat na nevalidní vstupy
GUI testování	jak dobře GUI funguje

Tabulka 2.1: Způsoby funkčního testování Převzato z [33]

Jak je vidět z tabulky 2.1, lze skrze GUI otestovat velkou část systému. Není pravidlem, že se každá chyba v systému musí objevit ve formě selhání. Lze však najít řešení, která budou úpravu GUI zahrnovat. Například skrýt nevhodné tlačítko nebo zavést povinný parametr ve formuláři. A tím nedovolit, aby se systém do chybového stavu dostal.

2.2.2 Manuální testování

Manuální testování je testovací metoda prováděná člověkem. Jako jediná nevyužívá žádných nástrojů k automatizaci. Můžeme se s ním setkat na začátku prozkoumávání systému, nebo u úkolů, které nelze vykonávat strojem. Manuální testování může provádět tester, specializovaný člen vývojového týmu, nebo uživatel(v rámci Beta testování).

Tester provádí testování vykonáváním testovacích případů. Ty jsou rozděleny do kroků, které tester vykonává. Na konci testu je manuálně výstup porovnán s očekávaným výstupem testu. Ten může být ve formě specifikace stavu, ve kterém se má aplikace na konci testu nacházet.

Nevýhodou tohoto přístupu je v časové náročnosti. To souvisí i s množstvím peněz, které je nutné do této práce investovat. Navíc provádění všech testovacích případů může být pro testera fyzicky i psychicky vyčerpávající. Lze tak zanést lidskou chybu do výsledků. Zároveň se tester může dopustit chyby i špatným vyhodnocením testu. I přes zmíněná negativa jde o činnost, u které lze najít i takové chyby, které by stroj nedokázal odhalit.[37]

Beta testování je typem akceptačního testování. Je prováděné uživateli, kteří budou v budoucnu software používat. Skrze používání je možné odhalit neočekávané chyby. Ale i případné nedostatky, kterých si vývojář nemusel všimnout. Výsledky testů, chyby a návrhy na vylepšení jsou zaznamenávány a předkládány vývojovému týmu.

2.2.3 Metoda zaznamenej a přehraj

Metoda spočívá ve využití lidského elementu k provedení akce nad GUI, která se v rámci testu bude opakovat. Akce je příslušným nástrojem nahrána a převedena do sekvence kroků

například ve formě metod. Tato akce bude referencí správného a předpokládaného chování systému. Metoda je nejběžnějším nástrojem pro regresní testování.[17] Existující nástroje tento způsob zaznamenávání kroků poskytují. Jako příklad může být WinAppDriverUIRecorder[1] (viz sekce 3.3.2).

2.2.4 Testování na základě znalosti aplikace

Tato metoda využívá automatizace skrze rozhraní, která jsou poskytována dodavateli daných systémů/webových prohlížečů. Díky těmto rozhraním je možné nahlížet do struktury právě spuštěných aplikací a ovládat je. Nahlížení do struktury může probíhat přístupem k tzv. *Document Object Model*[2] (DOM). V této struktuře jsou pak jednotlivé objekty vyhledávány (např. jazykem *XPath*[3]) a získávány jejich modely.[37] Nad nimi pak lze provádět uživatelské akce, nebo se dotazovat na jejich vlastnosti. Lze tak simulovat uživatele nad aplikací, nebo testovat jejich vlastnosti na obrazovce.

Jedním z největších představitelů této metody je projekt Selenium[4] (viz sekce 3.1).

2.2.5 Testování na základě rozpoznávání objektů GUI

Podobným způsobem jako v předchozí metodě lze vyhledávat i manipulovat s GUI pomocí jeho grafické reprezentace. Tento způsob vyhledávání se více blíží tomu, co by normální uživatel na obrazovce viděl.[37] Lze metodu využít i v případě, kdy je složité získat vnitřní strukturu GUI nebo zdrojové kódy aplikace.[10]

Jako příklad takového nástroje, je projekt *SikuliX*. S využitím obrazového rozpoznávání díky knihovně *OpenCV*[5] SikuliX rozpoznává grafické objekty na obrazovce. A pro schopnost rozpoznávání textu využívá i knihovny *Tesseract*[6].[10]

[1] `https://github.com/Microsoft/WinAppDriver/wiki/WinAppDriver-UI-Recorder`
[2] `https://www.w3schools.com/js/js_htmldom.asp`
[3] `https://www.w3schools.com/xml/xpath_intro.asp`
[4] `https://www.selenium.dev/`
[5] `https://docs.opencv.org/master/d1/dfb/intro.html`
[6] `https://github.com/tesseract-ocr/`

Kapitola 3

Nástroje pro automatizaci GUI testů

V této kapitole budou popsány některé existující nástroje a řešení pro testování grafického uživatelského rozhraní využívající metody znalosti aplikace. Mimo jiné bude představen framework Selenium. Tento projekt byl jeden z prvních svého druhu a je dnes největším a nejpoužívanějším nástrojem na trhu.[31]

Budou představeny hlavní principy, které jsou využity v dalších Seleniu podobných nástrojích. Dále bude představen framework pro automatizaci mobilních aplikací Appium (viz sekce 3.2). A jako poslední WinAppDriver, představitel automatizace desktopových aplikací systému Windows (viz sekce 3.3).

3.1 Selenium

Počátek projektu Selenium 1 se datuje na rok 2004. Touto dobou, ve společnosti *ThoughtWorks* v Chicagu, autor projektu Jason Huggins pracoval na nástroji „JavaScriptTestRunner“. Nástroj měl ušetřit čas strávený manuální verifikací chování uživatelského rozhraní webových aplikací. S využitím jazyka *JavaScript* bylo možné nahlédnout do struktury DOM prohlížeče a vykonávat některé z uživatelských aktivit. Řešení však vyžadovalo injektování kódu do kódu webových aplikací pro možnost přístupu ke driveru prohlížeče, což bylo nevyhovující a způsobovalo problémy.[31]

V roce 2011 byla zveřejněna nová verze Selenium 2.0. V rámci ní bylo představeno WebDriver API, které implementovalo ovladače tehdy nejpoužívanějších webových prohlížečů, včetně prohlížečů na mobilních platformách Android a iPhone. Jelikož tyto ovladače byly implementovány samotnými poskytovately platforem, není již nutné injektovat a upravovat kód testovaných aplikací.[20]

Projekt se dále vyvíjel a s příchodem verze 3.14.0 se stal jedním z W3C standardů[1] doporučený pro ovládání webových prohlížečů.[37]

Poslední verze, zveřejněná v březnu 2020, přinesla Selenium 4.0.0 Alpha 5, které zveřejnilo úplné standardizování Selenium WebDriver.[27]

[1] `https://www.w3.org/TR/webdriver1/`

3.1.1 Princip a filosofie Selenium

Jako otevřený software je Selenium testovacím frameworkem pro automatizaci a testování webových aplikací. Umožňuje připojit se k webovému prohlížeči a ovládat jej ať už lokálně, tak i vzdáleně. Lze tak simulovat aktivity, které by prováděl skutečný uživatel.

V rámci automatizace Selenium nevyžaduje injektování nebo změnu kódu testované aplikace. Vytváření testů není závislé na jediném jazyku či frameworku, což umožňuje přenositelnost vytvořených testů jak mezi prohlížeči, tak i operačními systémy (viz sekce [27]).

Selenium je souborem několika nástrojů:

- Selenium IDE je nástroj pro tvorbu testovacích případů. S verzí 4 je možné jej použít ve všech prohlížečích, které Selenium podporuje. Nástroj funguje na principu zaznamenej a přehraj.[30] (viz sekce 2.2.3)
- Selenium WebDriver je soubor API, která jsou používána k ovládání webových prohlížečů. Každý prohlížeč má specifický *driver* (ovládací rozhraní). Tím není potřeba měnit zdrojový kód testovaných aplikací. (viz sekce 3.1.2)
- Selenium Grid, také *Selenium Standalone Server*, lze využít ke spuštění testovacích běhů aplikace na vzdáleném zařízení, na více zařízeních zároveň, na zařízeních s různými operačními systémemy, nebo v různých webových prohlížečích[30].

3.1.2 WebDriver

Selenium podporuje všechny hlavní používané prohlížeče, jako je Chrom(ium), Firefox, Opera, Safari a Internet Explorer.[2] K jejich samotnému ovládání Selenium používá WebDriver představující API a protokol, který zajišťuje ovládání prohlížečů. Architektura protokolu WebDriver je klient- server.

Lokální uzel představuje klientskou část protokolu. Klient je implementovaný samotným vývojářem jednou ze Selenium poskytnutých klientských knihoven [2]. K dispozici jsou knihovny psané v nejpoužívanějších jazycích, jako: Java, Python, C#, Ruby, JavaScript nebo Kotlin.

Vzdálený uzel hostuje serverovou stranu protokolu. Ten může představovat dva typy uzlů, se kterými klient komunikuje:

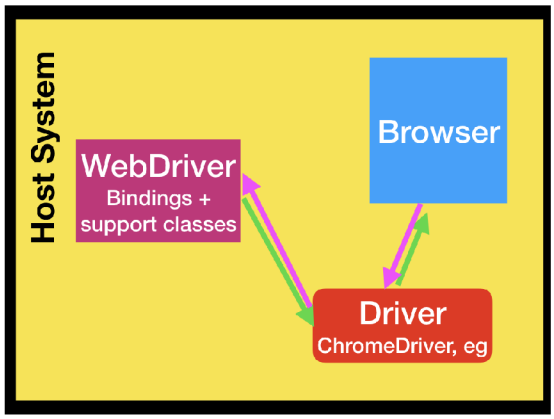
- *Prostředník* - představuje proxy server, který v komunikaci vystupuje jak jako lokální, tak vzdálený uzel.
- *Koncový uzel* - uzel, na kterém běží web prohlížeč.

Lokální uzel nemá přístup k implementaci žádného ze vzdálených uzlů. Jeví se pro něj jako černé skřínky.[15]

Koncový uzel je implementovaný *user agentem*, který pak vykonává akce nad aplikací. Akce jsou diktované klientem, přeneseny skrz WebDriver až k driveru prohlížeče, a nad ním pak akce vykonávány. Výsledky akcí jsou přenášeny stejnou cestou zpět klientovi, zpracovány a interpetovány.[39] Celá komunikace probíhá přes protokol HTTP, který přenáší data ve formátu JSON.[21]

Obrázek 3.1 představuje nejjednoduší přímou komunikaci mezi WebDriver (klienská strana) a prohlížečem. (*Binding* viz 3.1.3).

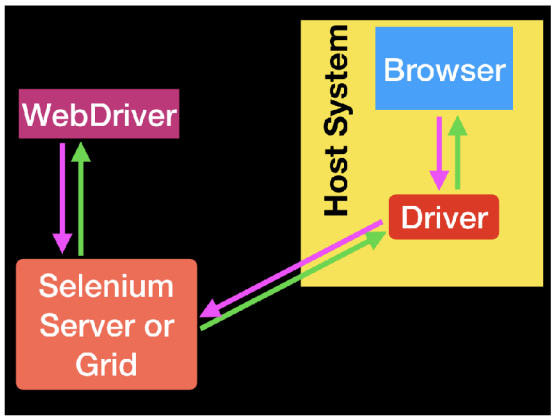
[2] `https://www.selenium.dev/documentation/en/selenium_installation/installing_selenium_libraries/`



Obrázek 3.1: Schéma nejjednoduší komunikace mezi WebDriverem, driverem prohlížeče, a prohlížečem. Převzato z [29]

Driver je zde specifický podle typu prohlížeče. Prohlížeč a jeho driver se vždy nacházejí na stejném systému.[29]

To nemusí platit pro implementovaného klienta. Další obrázek představuje komunikaci mezi klientem a driverem prohlížeče přes Selenium Server, který zde funguje jako prostředník. V tomto případě může jít o vzdáleného klienta, který zasílá příkazy prohlížeči na lokálním systému.



Obrázek 3.2: Připojení WebDriveru přes Selenium Server.[29]

Pro nalezení elementu na stránce poskytuje WebDriver několik typů již zabudovaných funkcí [3]. Příklad 3.1.2 zdrojového kódu je způsob získání elementu podle vlastnosti `Id`.[19] (Názvy objektů a funkcí se mohou lišit vzhledem k použité jazykové knihovně. Zde byl použit jazyk C#.)

```
IWebElement element = driver.FindElement(By.Id("button_edit"));
```

Vystupují zde dvě základní instance objektů, které WebDriver implementuje:

- `WebDriver (driver)`: je objekt instance WebDriver a představuje prohlížeč. Je zároveň kořenem struktury DOM.

[3] `https://www.w3.org/TR/webdriver1/#element-retrieval`

- `WebElement (element)`: je objekt představující konkrétní DOM uzel (tlačítko, textbox, okno uvnitř prohlížeče, atd.). WebElement dále implementuje atributy a vlastnosti, které reprezentují daný objekt. Jsou k dispozici i funkce, kterými lze získat velikost objektu, jeho polohu, stav, nebo na něm vykonávat akce a dotazy (Click, IsVsible, ...) [4].

Metoda `findElement(By)`[5] vrací instanci objektu `WebElement`, reprezentuje objekt v prohlížeči. Pokud takových elementů bude nalezeno více, tato funkce vrací první výskyt. Pokud není nalezen žádný, vrací funkce hodnotu **null**.

Elementy lze vyhledávat podle různých vlastností. Ty jsou implementovány rozhraním *By*, které bylo použito v příkladu 3.1.2. Toto rozhraní podporuje několik lokalizačních strategií. Následující tabulka obsahuje všechny, které jsou ve WebDriver implementované:

Lokátor	**Popis**
class name	Lokalizace elementů, jejichž třída obsahuje hledanou hodnotu
css selector	Lokalizace elementů odpovídající vlastnosti CSS (např. barva pozadí)
id	Lokalizace elemntů, jejichž ID atribut se shoduje s hledanou hodnotou
name	Lokalizace elementů, jejichž NAME atribut se shoduje s hledanou hodnotou
link text	Lokalizace elementů s odkazem, jejichž viditelný text se shoduje s hledanou hodnotou.
partial link text	Lokalizace elementu s odkazem, který ve viditelném textu se shoduje s hledanou hodnotou. Pokud je takových elementů nalezeno více, vrátí první nalezený.
tag name	Lokalizuje elementy, jejichž tag name(button, window, panel, textbox...) se shoduje s hledanou hodnotou.
xpath	Lokalizuje elementy, jejichž XPath se shoduje s hledanou hodnotou.

Tabulka 3.1: Tabulka se seznamem všech atributů, podle kterých lze elementy vyhledávat

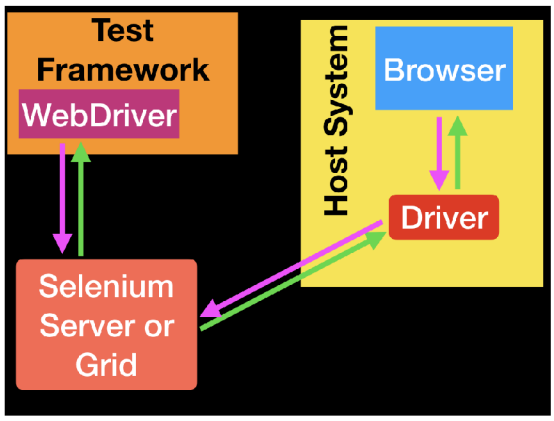
3.1.3 Testovací frameworky

V části 3.1.2 byl stručně vysvětlen princip použití Selenium WebDriver pro ovládání a interakce s potenciálním SUT. Zároveň i možné využití Selenium Grid (viz sekce 3.1.1) pro vzdálenou komunikaci. Nyní záleží na samotném vývojáři či testerovi, jakým způsobem bude dané nástroje využívat.

Selenium sice poskytuje automatizaci a k tomu příslušné knihovny a nástroje. K samotnému testování je nutné využít frameworků pro to určené. Ty také budou zodpovědné za spouštění WebDriveru, vytvoření spojení a za interpretaci zpráv, které budou klientovi přicházet. Jak bylo vidět na obrázku 3.1, WebDriver se nacházel v modulu, kde bylo zajištěné tzv. vazebné aplikační rozhraní (*Binding*). Toto rozhraní poskytuje lepidlový kód.[7] Zde přichází na řadu poskytnuté knihovny, které WebDriver aplikační rozhraní poskytuje pro řadu jazyků (viz sekce 3.1.2). Není tedy nutné před začátkem komunikace specifikovat jazyk, ve kterém budou testy a klient vytvořené.

[4] `https://www.w3.org/TR/webdriver1/#element-state`
[5] `https://www.w3.org/TR/webdriver1/#find-element`

Obrázek 3.3 zobrazuje finální využití nástrojů Selenium. Využité testovací frameworky mohou být ty, které jsou psané v jednom z podporovaných jazyků. Těmi jsou například: NUnit pro .NET, JUnit pro Java, nebo RSpec pro Ruby.



Obrázek 3.3: Zapojení testovacího frameworku. Převzato z [29]

3.2 Appium

Tento otevřený software je nástroj pro automatizaci nativních, mobilních webových a hybridních aplikací systémů iOS, Android a Windows desktop.[39] Nejvíce se však zaměřuje právě na mobilní aplikace. Filosofie Appia spočívá ve 4 bodech:

- Není třeba překládat ani měnit zdrojový kód aplikace ve smyslu automatizace.

Appium, podobně jako Selenium a podporované prohlížeče, používá a již implicitně obsahuje frameworky, které jsou vytvářeny a spravovány poskytovately platforem.

- Nebýt závislí na jednom jediném jazyku a frameworku, ve kterém budou psány a spouštěny testy.

Všechny podporované frameworky jsou obsaženy v jednom API. Tím je již dříve zmíněný WebDriver (viz sekce 3.1.2).

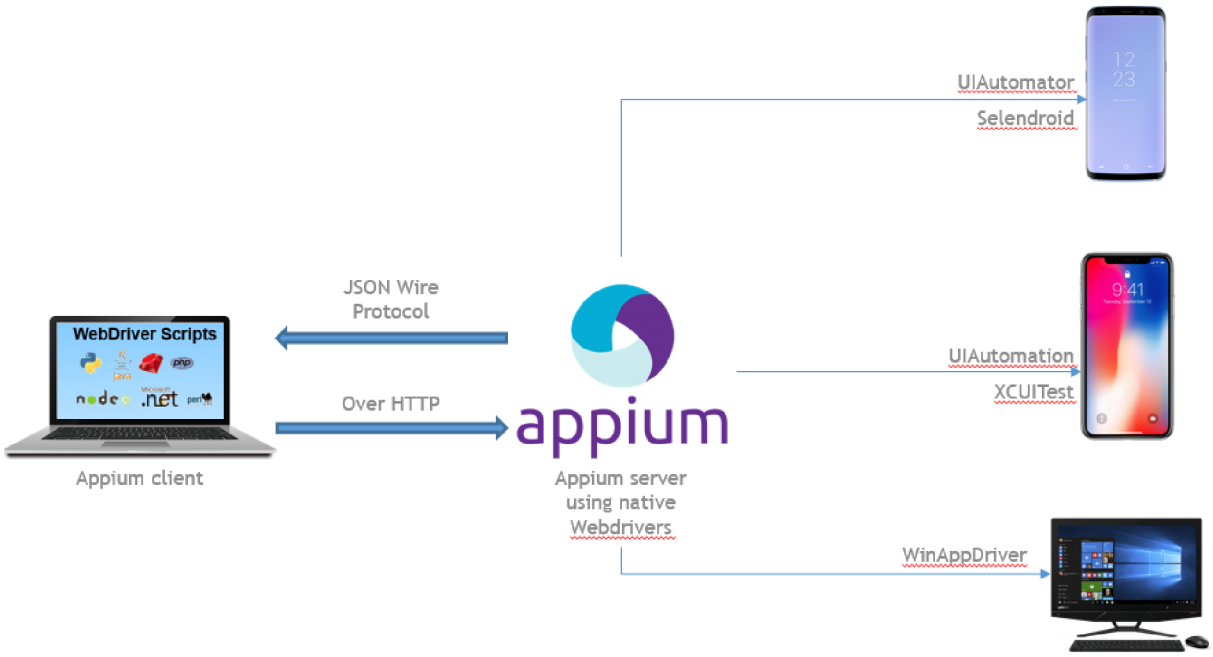
- Využít již dostupných technologií a frameworků pro automatizaci mobilních aplikací.

Appium využívá WebDriver API ke komunikaci a ovládání aplikací. Využívá se tedy řešení, které již existuje. Zároveň Appium rozšiřuje API o metody, které jsou třeba pro automatizaci mobilních aplikací.[39]

- Tyto frameworky a nástroje poskytovat jako otevřený software.

Dokumentace k nástroji je k dispozici zde: `https://appium.io/docs/en/about-appium/intro/`.

3.2.1 Koncept



Obrázek 3.4: Schéma komunikace mezi Appium, testovacím frameworkem a mobilní aplikací. Převzato z [18]

Koncept Appia se příliš neliší od konceptu Selenia a Selenium WebDriver. Appium jde ale v automatizaci a testování o kousek dál. Spojuje webový server poskytující REST API a WebDriver.[39]

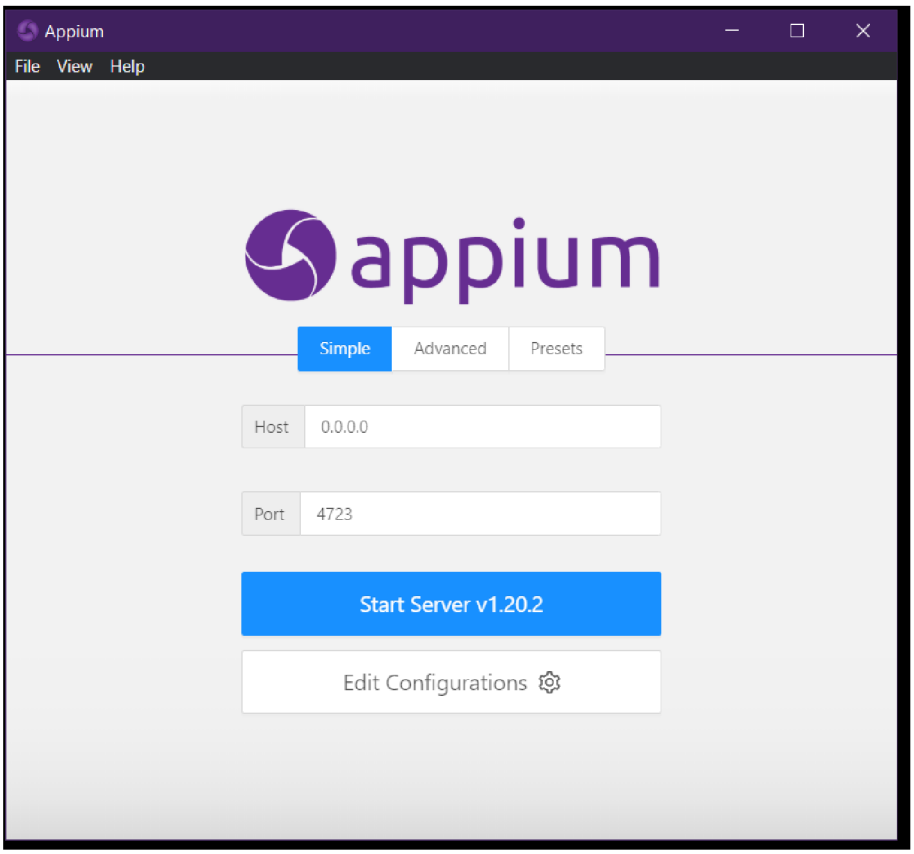
Architektura klient/server je stejná jako u projektu Selenium. Samotný server je webovým REST API, které komunikuje s klientem na jedné straně, a provádí příkazy nad mobilní aplikací, na straně druhé. Již se zde neodděluje server zařizující spojení a WebDriver, ovládající driver zařízení. Appium poskytuje obojí. Pro psaní testů jsou k dispozici knihovny psané v jazycích, které podporuje samotný WebDriver. K tomu Appium rozšiřuje tuto sadu o knihovny jazyků PHP[6] a RobotFramework[7]. Všechny podporované knihovny je možné nalézt na adrese: `https://appium.io/docs/en/about-appium/appium-clients/index.html`.

Princip spojení mezi klientem a Appiem, se také nazývá *session*. Klient iniciuje spojení připojením se k serveru a zasláním tzv. *desired capabilities* (požadované schopnosti) ve formě JSON. Tyto požadavky obsahují klíčové hodnoty, které specifikují vlastnosti spojení, které je od Appia požadováno. Lze tak nastavit např. jméno testované platformy a její verzi. Server tento objekt přijme a zahajuje spojením s odpovědí, která nese identifikační číslo spojení. Appium pak vykonává akce diktované klientem. Aby mohlo Appium zařízení ovládat, používá k tomu automatizované frameworky, spravované poskytovateli dané platformy. Těmito poskytovately jsou mimo jiné Microsoft, Apple nebo Google.[34] Následující seznam obsahuje poskytovately spravované frameworky[39]:

[6] `https://github.com/appium/php-client`
[7] `https://github.com/serhatbolsu/robotframework-appiumlibrary`

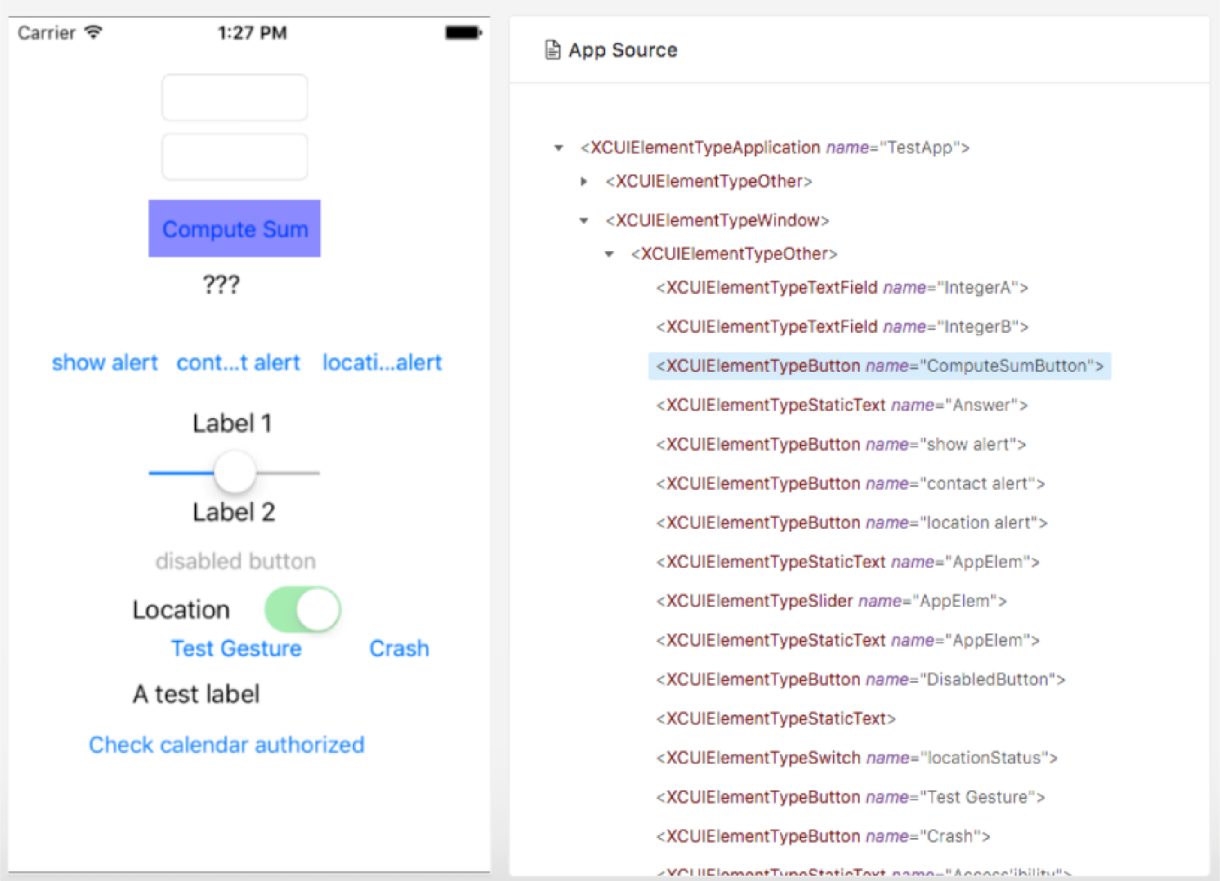
- iOS 9.3 a vyšší: Poskytovatel Apple XCUITest[8]
- iOS 9.3 a nižší: Poskytovatel Apple UIAutomation[9]
- Android 4.3+: Poskytovatel Google UiAutomator/UiAutomator2[10]
- Windows: Poskytovatel Microsoft WinAppDriver[11], viz také 3.3

Kromě knihoven je možné využít i desktopové aplikace Appium Desktop[12]. Aplikace jsou přehledným grafickým obalem serveru Appium (viz sekce 3.5). Jako podpora pro vytváření testů je v aplikaci k dispozici i tzv. *Inspector*. Ten umožní zobrazit hierarchii objektů na obrazovce aplikace.

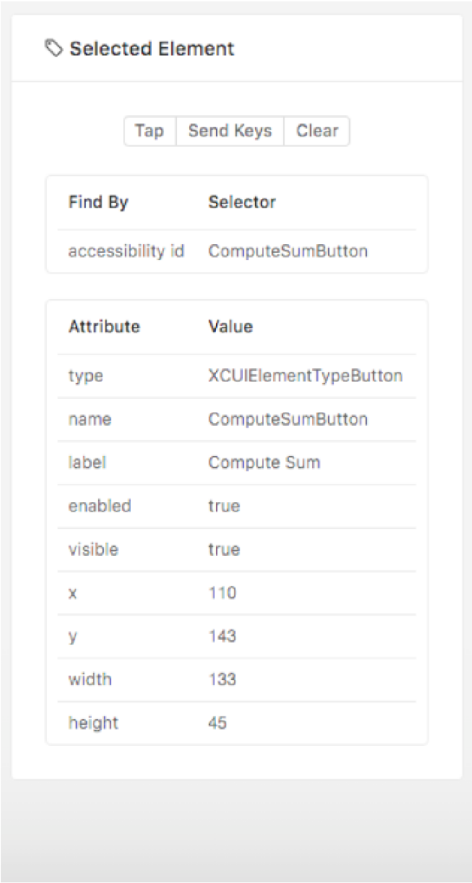


Obrázek 3.5: Snímek vzhledu Appium desktop pro Windows.

[8] https://developer.apple.com/documentation/xctest
[9] https://web.archive.org/web/20160425114149/https://developer.apple.com/library/ios/documentation/DeveloperTools/Reference/UIAutomationRef/
[10] https://developer.android.com/training/testing/ui-automator
[11] https://github.com/microsoft/winappdriver
[12] https://github.com/appium/appium-desktop/blob/master/README.md



Obrázek 3.6: Levá a prostřední část okna nástroje Inspector v aplikaci Appium. V levé části je zobrazen vzhled obrazovky zařízení. V pravé části (uprostřed obrazovky aplikace) je zobrazena hierarchie obrazovky zařízení ve formátu XML. Převzato z [28].



Obrázek 3.7: Pravá část obrazovky nástroje Inspector aplikace Appium. Zde je vidět detail vybraného elementu na obrazovce (viz obrázek 3.6). Převzato z [28].

3.3 WinAppDriver

Windows Application Driver (WinAppDriver) je testovací framework, vyvinutý firmou Microsoft. Jde o otevřený software navržený jako podpora automatizace testování desktopových aplikací systému Windows. Inspirace pro tento projekt jsou projekty Selenium a Appium (viz sekce [24]). Představuje další rozšíření již existujícího API WebDriver[13].

WinAppDriver byl navržen tak, aby umožňoval interakci s aplikacemi, které jsou implementovány frameworky Universal Windows Platform (UWP), Windows Forms (WinForms), Windows Presentation Foundation (WPF) a Classic Windows (Win32)[36].

UI automatizace desktopových aplikací je oproti webovému UI poměrně odlišná. Webové stránky jsou dokumenty reprezentovány formou HTML. Tato HTML podoba lze převést na strukturu DOM, která dovoluje s dokumentem manipulovat a měnit jej. V této struktuře existují konkrétní objekty v podobě uzlů a jejich atributů, představující vlastnosti objektu. Na tomto principu pracuje nástroj WebDriver, který touto strukturou dokáže procházet, vyhledávat v ní a zjišťovat informace o jednotlivých elementech (viz sekce 3.1.2). DOM je tedy technologií, která umožňuje interagovat s grafickými elementy webu.[4]

[13] `https://www.nuget.org/packages/Microsoft.WinAppDriver.Appium.WebDriver/1.0.1-Preview`

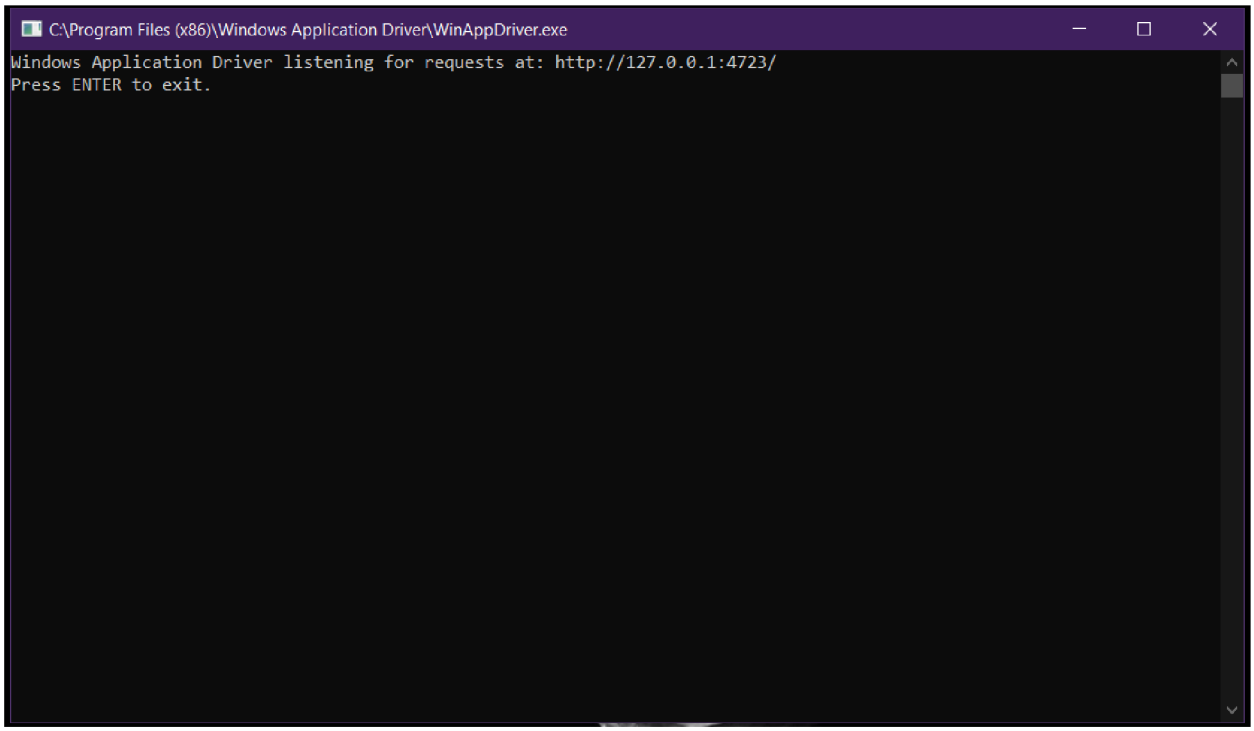
U desktopových aplikací systému Windows toto neplatí. Již zmíněné technologie UWP a WPF, jsou založené na jazyku XAML. WinForms a Win32 byly vyvinuty v jazyce C++. Pro interakci s aplikacemi vytvořené ve jmenovaných frameworcích, je nutné využít technologií UI Automation (UIA) nebo Microsoft Active Accessibility (MSAA[14]). Obě tyto technologie WinAppDriver podporuje a tím umožňuje přistupovat ke grafickému rozhraní aplikací stejným způsobem, jako Selenium přistupuje k objektům webového prohlížeče.[24]

WinAppDriver byl vyvinut pro systém Windows. Proto se pro vytváření testů doporučuje využít technologií a testovacích frameworků firmy Microsoft. Jako je například MSTest (viz sekce 6.1.3) a Visual Studio.[24]

3.3.1 Princip WinAppDriveru

WinAppDriver je poskytnut firmou Microsoft jako aplikace, i jako knihovna. Nejnovější verzi WinAppDriver lze stáhnout z oficiálních stránek[15]. Pro stažení knihovny lze využít například balíčkový manažer NuGet[16] pro technologii .NET.

Po instalaci a spuštění je WinAppDriver standardně připojen na localhost na portu 4723 (127.0.0.1 : 4723). K této adrese se bude později připojovat klient při zahajování spojení (viz výpis 3.1).



Obrázek 3.8: Zobrazené logovací okno aplikace WinAppDriver.

Na obrázku 3.8 je vidět logovací okno. Veškerá aktivita WinAppDriveru je zde zobrazena ve formě JSON výpisů. Spojení je navazováno mezi klientem, WinAppDriverem a testovanou aplikací. Proto se při vytváření spojení musí WinAppDriveru informace předat. K tomu

[14] `https://docs.microsoft.com/en-us/windows/win32/winauto/microsoft-active-accessibility-and-ui-automation-compared`

[15] `https://github.com/Microsoft/WinAppDriver/releases`

[16] `https://github.com/Microsoft/WinAppDriver/releases`

poslouží tzv. požadované schopnosti (*desiredcapabilities*, viz sekce 3.2.1), kterým se spojení definuje. WinAppDriver pak testovanou aplikaci spustí a může ji začít ovládat. Umožňuje však i připojení k již běžící aplikaci[17].

Implementace klienta lze provést například pomocí projektu MSTest (viz sekce 6.1.3). Tento framework poskytuje nástroje pro spouštění a vyhodnocení testů. Za použití metod implementované v rámci WinAppDriver API lze simulovat interakci uživatele s aplikací.

Následující část zdrojového kódu (viz výpis 3.1) představuje vytvoření spojení mezi klientem, WinAppDriverem a aplikací Notepad.

```
DesiredCapabilities appCap = new DesiredCapabilities();
appCap.SetCapability("app",@"C:\Windows\System32\notepad.exe");
appCap.SetCapability("appArguments",@"MyTestFile.txt");
appCap.SetCapability("appWorkingDir",@"C:\MyTestFolder\");
NotepadSession = new WindowsDriver<WindowsElement>(new
       Uri("http://127.0.0.1:4723"), appCap);
```

Výpis 3.1: Vytvoření spojení s aplikací Notepad využitím knihovny OpenQA.Selenium.Remote[19]

V příkladu je vytvořena instance objektu *DesiredCapabilities* a jsou do pole schopností vloženy následující hodnoty. Atribut *app* s hodnotou string, představující absolutní cestu ke spustitelnému souboru aplikace Notepad. Atribut *appArguments* představující vstupní parametry pro spouštěnou aplikaci. Zde jde o jméno souboru, který bude v Notepad otevřen. A atribut *appWorkingDir*, představující cestu ke složce se souborem, který bude otevřen. Poté následuje navázání spojení vytvořením instance *NotepadSession*. Pro její vytvoření je nutné předat Uri serveru, ke kterému se bude připojovat (viz sekce 3.8) a popis spojení díky objektu *appCapabilities*.

Jeden z rozdílů oproti Appiu je v typu tříd, se kterými WinAppDriver pracuje. WinAppDriver implementuje třídy WindowsElement a WindowsDriver. Tyto třídy jsou implementované rozhraním tříd AppiumWebElement a AppiumDriver. Protože rozhraní tříd, které používá Appium framework, jsou rozšířením aplikačního rozhraní WebDriver API, používá WinAppDriver stejné rozhraní, jaké poskytuje samotný WebDriver. Práce s objekty je tedy stejná, jako tomu je u frameworku Selenium.

```
WindowsElement element = NotepadSession.FindElementByClassName("Edit");
element.SendKeys("Hello World!!");
```

Následující tabulka představuje schopnosti a jejich popis, podle kterých lze spojení specifikovat:

[17] https://github.com/Microsoft/WinAppDriver/wiki/Frequently-Asked-Questions#when-and-how-to-attach-to-an-existing-app-window

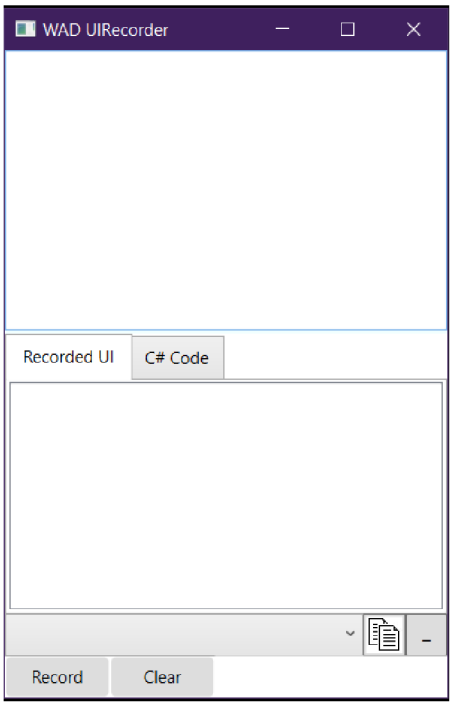
[19] https://www.selenium.dev/selenium/docs/api/java/org/openqa/selenium/remote/package-summary.html

Schopnost	Popis
app	Identifikátor spouštěné aplikace nebo absolutní cesta k souboru .exe
appArguments	Vstupní argumenty spouštěné aplikace
platformName	Jméno cílené platformy
platformVersion	Verze cílené platformy

Tabulka 3.2: Seznam některých definovatelných atributů spojení.

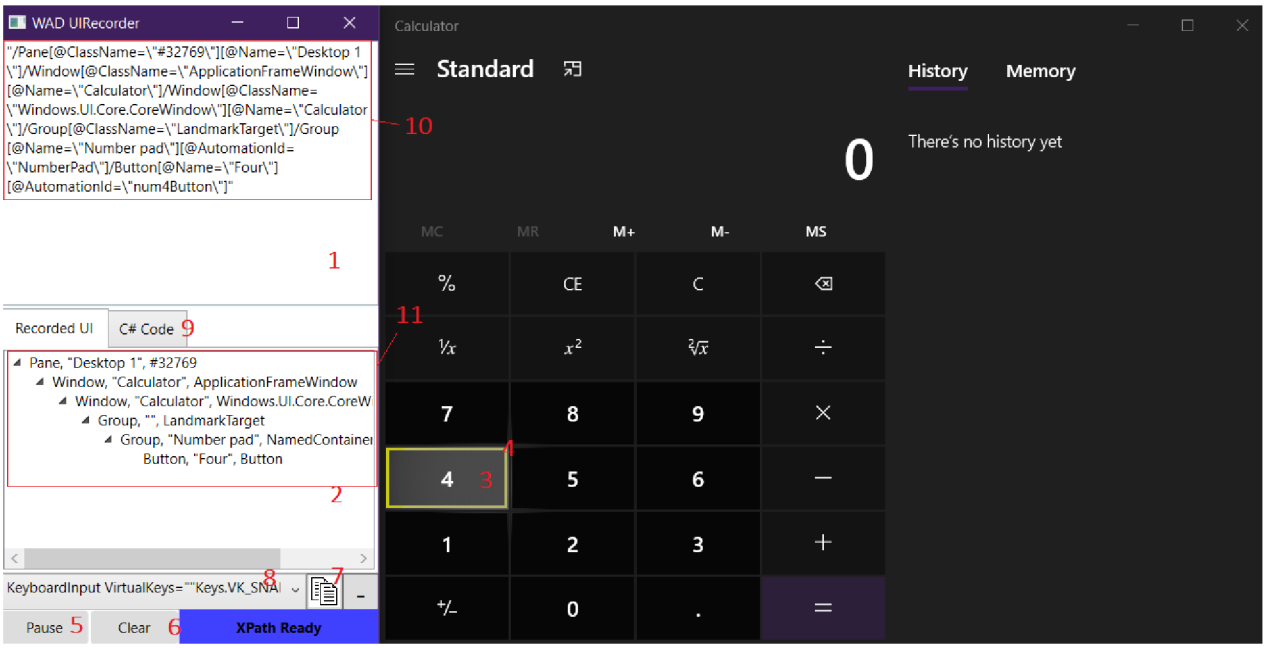
3.3.2 WinAppDriverUIRecorder

Podobně jako u projektu Appium a nástroje Inspector (viz obrázek 3.6), WinAppDriver poskytuje nástroj UIRecorder. Aplikace je volným softwarem, který nevyžaduje instalaci. Jde o nástroj, který umožňuje zaznamenání aktivit uživatele pro pozdější přehrání pomocí WinAppDriver.



Obrázek 3.9: Snímek aplikace UI Recorder.

Obrázek 3.9 představuje grafické rozhraní nástroje UIRecorder. Použití vyhledání elementu na obrazovce je zobrazeno na obrázku 3.10.



Obrázek 3.10: Ukázka inspekce elementu tlačítko v aplikaci Kalkulačka na systému Windows.

Na obrázku 3.10 je demonstrováno nahrání postupu na aplikaci Kalkulačka na systému Windows 10. Princip v zaznamenávání aktivit spočívá ve sledování ukazatele myši a událostí na klávesnicí. Celý proces zaznamenání a nahrání postupu probíhá v několika krocích:

1. Po spuštění nástroje UIRecorder(5) začne nástroj sledovat pohyb myši na obrazovce a aktivitu klávesnice. Pokud se myš přesune nad oblast, kde je například tlačítko(3), nástroj vykreslením jeho okrajů(4) dá najevo, že element zachytil a zobrazí XPath řetězec daného lokalizovaného elementu(10) na horní obrazovce(1).

2. Po zachycení elementu je možné provést akci, která bude interpretována ve formě XML záznamu(11). Na dolní obrazovce (2) se pak zobrazí jeho podoba.

3. První dva body lze opakovat. Jednotlivé aktivity jsou ukládány(8) a lze je upravovat.

4. Po ukončení nahrávání(znovu 5) lze převést formát nahrávky do formátu C# metod(9) a zkopírovat do schránky(7). Nebo je možné celý postup smazat(6) a nahrát aktivity nové.

Výstupem celého procesu je nahraný postup aktivit nad GUI aplikace ve formátu XML elementů, nebo C# metod. Tento formát lze pak vložit do testovacího projektu jako testovací případy. Metody se skládají z funkcí, které používají aplikační rozhraní WindowsDriver a WindowsElement pro interakci s aplikací postupem, jaký byl před tím nahrán. Tím je možné automatizovat některé stereotypní úkony, nebo otestovat stále se opakující proces.

3.3.3 Vývoj

Vývojáři WinAppDriveru provozují stránky s podporou a repozitář, kde jsou umístěny všechny zdrojové kódy. Kromě odkazu na instalační stránku WinAppDriver a UI Recorder,

je zde k dispozici i soubor příkladů pro vytvoření testů a testovacích skriptů, použití jednotlivých method WinAppDriver API, a také příslušná dokumentace. Nástroj je poměrně nový a stále se vyvíjí.

3.4 Limity a nedostatky nástrojů pro automatizaci testování GUI

Hlavním problémem těchto nástrojů, je zejména v úplné náhradě myšlení uživatele strojem. Největším problémem je čas, který je nutný pro načítání nového okna. Pokud není v testech explicitně zavedeno čekání, nástroje si s problémem neporadí a mohou tak být některé z testů zkresleny a vyhodnoceny špatně.

Autor testů navíc musí počítat s tím, že nástroje ve skutečnosti netuší, s čím pracují. Pro ně jsou objekty na obrazovce pouze uzly s atributy ve stromové struktuře. V ní se postupně vyhledává. Proto se vyhledávání objektů implementuje nejčastěji ve formě cyklu s nastaveným časováním. Periodicky se kontroluje, jestli nebyl požadovaný element nalezen. Toto vyhledávání se provádí jen v určitém počtu nastavených pokusů. Samozřejmě, čím delší se nastaví časování a počet pokusů, tím bude zvýšena i pravděpodobnost, že bude vyhledávání úspěšné. Je však neefektivní 10 krát prohledávat celou strukturu hledáním elementu, který na obrazovce ve skutečnosti není.

Při rozpoznávání objektů v rámci nahrávání může dojít ke komplikacím spojené se složitostí GUI. Proto se musí vývojář ujistit, že bude daný test skutečně fungovat. Například se bude muset zvolit jiná funkce, nebo jako hledanou hodnotu využít jiného atributu elementu.

Pro vytváření testů je k dispozici několik nástrojů. Absolutní cesty XPath, které vygenerují jako reprezentaci elementu na obrazovce, jsou většinou komplikované a špatně se odlaďují. Doporučuje se tedy před nasazením testů, je nejdříve vyzkoušet.

3.5 Zhodnocení nástrojů pro automatizace GUI testů na základě znalosti aplikace

Následující tabulka obsahuje srovnání představených frameworků v cílených technologiích, kompatibilních systémech, driverů, které jsou použity, a typu distribuce.

Framework	Technologie	Kompatibilita	Kořenový objekt
Selenium	web	Chrom(ium), Firefox, Opera, Safari, Internet Explorer	WebDriver
Appium	nativní, mobilní, hybridní, desktop	iOS, Android, Windows	AppiumDriver
WinAppDriver	desktop	Windows	WindowsDriver

Tabulka 3.3: Tabulka obsahuje srovnání všech zmíněných nástrojů ve vybraných aspektech. Kompatibilita označuje systém nebo webový prohlížeč, na kterém se nachází SUT.

Kapitola 4

Analýza problému

V této části bude představen zákazník. V části 4.2 a 4.3 bude popsáno prostředí systému, kterého se bude řešení týkat, spolu s aktuálním stavem, i snímkem pro ucelení představy o SUT. Na základě těchto informací budou v další části předloženy požadavky.

4.1 Zadavatel

Process Automation Solutions[1] je Německou společností, založenou roku 1986. Dnes se jedná o mezinárodní korporát působící ve 14 zemí napříč světem. V České republice společnost působí již 20 let. Se svými pobočkami v Praze, Brně, Lovosicích a Mladé Boleslavi poskytují své produkty a služby nejen českým firmám.[1]

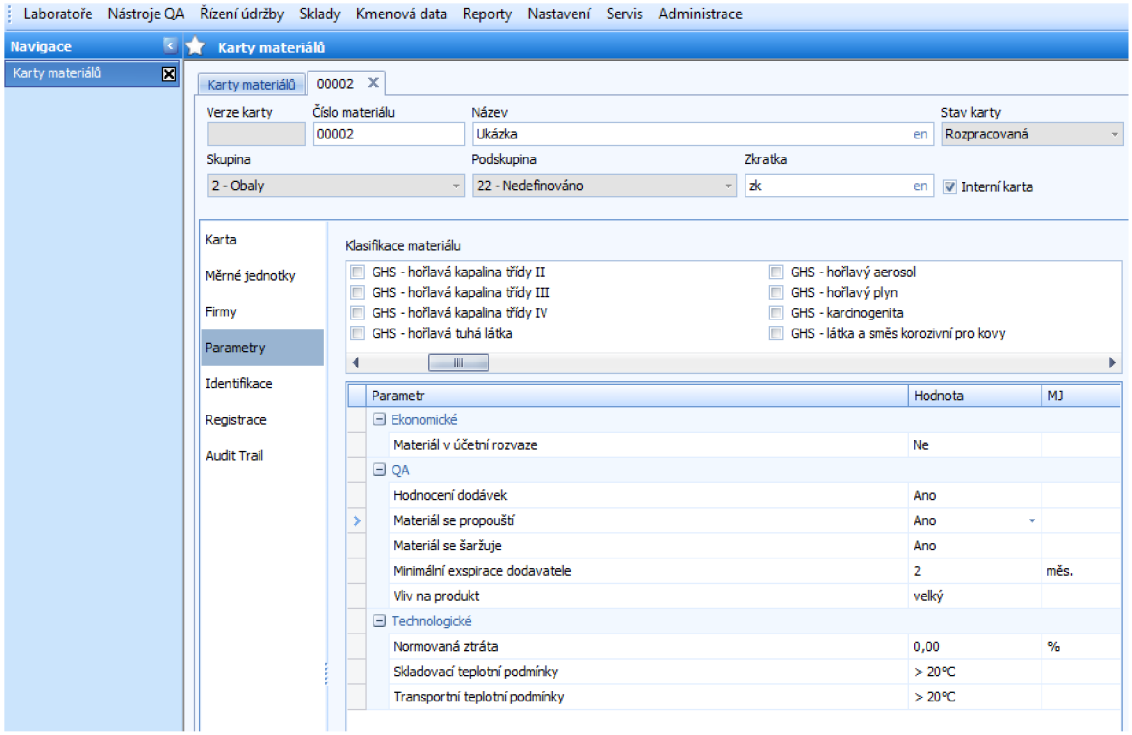
Společnost poskytuje automatizační řešení pro technologické a výrobní procesy. Zabývá se implementací průmyslových řídících systémů a jejich integrací do podnikových procesů. Těmi jsou zejména chemický průmysl, farmacie a biotechnologie, potravinářský průmysl, ropa a plyn, či automobilový průmysl.[1]

4.2 Popis SUT

Software, kterého se řešení dotýká, je informační systém MES (anglicky Manufakturing Execution Systems). Tyto systémy jsou používané ve výrobních podnicích pro řízení a monitoring výrobních procesů.[16] Zahrnují přehledy a seznamy laboratoří, skladů, výroben a dalších objektů a faktorů specifické pro daného zákazníka. Informační systémy jsou vyvíjené jako desktopové aplikace na systému Windows. Implementovány jsou v jazyce rodiny .NET s využitím knihovny WinForms pro tvorbu grafického rozhraní.

Každý ze zákazníků má specificky upravený systém podle svých potřeb. Rozdíly jsou však pouze v konfiguraci. Systémy jsou ve většině případů robustní a složité. Obsahují často velké množství formulářů a dialogových oken.

[1] `https://pa-ats.com/cz-cz`



Obrázek 4.1: Ukázka grafického rozhraní SUT s formulářem, který představuje objekt materiálové karty v systému.

4.3 Aktuální stav

Systémy několika zákazníků jsou přiděleny oddělení s osmi členy, které tvoří: konzultant, jeden vedoucí/senior, dva senioři, dva programátoři a dva junioři/brigádníci. Pokud je dokončena oprava chyby nebo implementace nové vlastnosti, před uzavřením jednoho cyklu je výsledný stav aplikace otestován. Testování provádí kdokoli jiný z týmu, až na samotného autora. Pokud změnu provede brigádník, nesmí po něm změnu kontrolovat nikdo jiný, než jeden ze seniorů.

Tým pracuje na systému Windows ve vývojovém prostředí Visual Studio[2]. Veškeré projekty jsou zálohované a sdílené přes verzovací systém Git[3]. Data, která jsou v systémech zobrazována, jsou uložena v relačním databázovém a analytickém systému Microsoft SQL server, verze 2019[4], dále jako MSSQL. Testování probíhá manuálně. Testuje se funkčnost, použitelnost, i kontrola kódu. Pokud řešení nesplňuje požadavky na změnu nebo mění funkčnost systému vzhledem ke specifikaci, je úkol vrácen zpět autorovy na opravu. Testy pro kontrolu průběžné integrace nejsou aktuálně příliš využívány.

Pro možnost regresního testování existuje také dokumentace s kvalifikačními testy. Podle nich lze otestovat funkcionality každého systému, nebo vytvořit testy, které se již budou jen spouštět. Míra automatizace testů není však dostatečná a je vhodné ji zvýšit.

[2] `https://visualstudio.microsoft.com/cs/downloads/`
[3] `https://git-scm.com/`
[4] `https://www.microsoft.com/cs-cz/sql-server/sql-server-2019`

Kapitola 5

Analýza požadavků a návrh nástroje pro podporu manuálních GUI testů

V této kapitole bude popsán vývoj nástroje pro podporu manuálních GUI testů. V části 5.1 jsou specifikované požadavky, které by měli být splněny ve výsledném řešení. Poté bude popsán konceptuální návrh obou nástrojů. V části 5.3 bude popsán strukturní návrh konzolové aplikace AuTeR. Poté bude následovat část 5.4 se strukturním návrhem reportovacího frameworku AuTeReporter. V této části bude také detailněji popsán algoritmus například vyhledávání elementů, který bude ve frameworku použit. Poslední dvě části se budou věnovat návrhu řešení pro interpretaci testů, viz 5.4.5 a také návrh samotného reportu 5.4.6.

5.1 Analýza požadavků a návrh řešení

Id	Popis	Část
REQ.1	Řešení musí umožňovat spouštění testů bez přítomnosti testera.	-
REQ.2	Pokud bude po nástroji vyžadován stažení změn z repozitáře, provede jej.	konzole
REQ.3	Předem vytvořené testy budou moci využívat funkcí pro vytvoření reportu.	framework
REQ.4	Report bude ve formě souboru, který je čitelný a přehledný pro člověka.	framework, report
REQ.5	Výsledný report bude možné vložit do předem definovaného souboru. Pokud nebude složka specifikována, bude uložen do složky k testům.	framework, konzole
REQ.6	Nástroj bude kompatibilní s testy využívající framework MSTest nástroje Visual Studio.	framework
REQ.7	Nástroj bude schopný pracovat s aplikacemi využívající frameworků WinForm, WPF a DevExpress.	framework
REQ.8	Nástroj bude podporovat nástroje firmy Microsoft a bude spustitelný na systému Windows.	framework, konzole
REQ.9	Pro funkci reportování bude možné vytvářet testy s co nejmenší režií.	framework
REQ.10	Umožnit po doběhnutí testů rozesílat výsledný report přes elektronickou poštu na předem specifikované adresy.	konzole
REQ.11	Pokud se bude pracovat s elementem typu tlačítko, vytvoří se snímek obrazovky a tlačítka.	framework, report
REQ.12	Pro specifikaci testů bude možné vkládat do reportu poznámky.	framework, report
REQ.13	Pokud řešení bude zahrnovat i práci s databází, tak databáze bude připravena před samotným spuštěním testů a obnovena po jejich provedení.	-

Tabulka 5.1: Tabulka obsahuje seznam požadavků, podle kterých má být nástroj navržen.

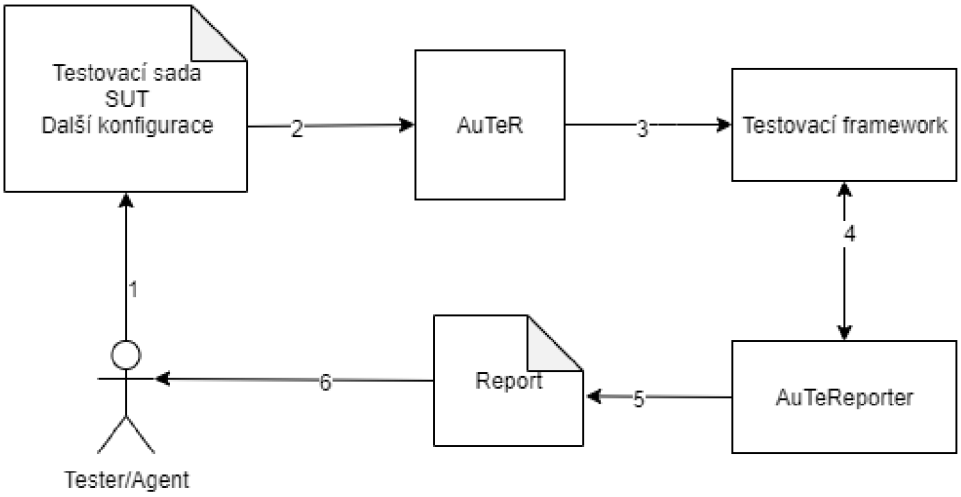
5.2 Konceptuální návrh

Na základě požadavků bylo navrženo řešení, které by umožňovalo automatické spuštění testů bez přítomnosti testera a z jejich průběhu vytvářet report. Zároveň má řešení zahrnovat možnost snadného vytváření GUI testů, pro jejich použití v rámci regresního testování. Řešení bylo navrženo jako soubor dvou samostatných částí, které budou v průběhu práce vyvíjeny. První část je konzolová aplikace, která na vstupu přijme informace o testech, cesty k potřebným souborům a cestu ke spustitelnému souboru SUT. Tato aplikace byla nazvána AuTeR (Automated TesteR). Druhá část řešení je reportovací framework. Ten má nejen generovat samotný report, ale má zejména zajišťovat interakci s grafickým rozhraním SUT a částečně řídit průběh testů. Byl nazván AuTeReporter (Automated Tester and Reporter).

V rámci požadavků bylo nutné najít řešení, které by zajišťovalo práci s databází (viz REQ.1 a REQ.13 v tabulce 5.1). Systém MSSQL (viz sekce 4.3) poskytuje možnost vytváření skriptů, které mohou dané akce zajistit a nemuset jejich vykonání přesouvat mimo pro-

středí databázového systému. Spolu s vytvářením skriptů navíc poskytuje služby tzv. *SQL Server Agenta*. Jde o službu, která spouští naplánované akce a procesy, jinak také nazývané jako *jobs*.[11] Tímto způsobem lze například spustit aplikaci, která se postará o spuštění a průběh testů. Záloha databáze bude provedena před samotným spuštěním testů a obnovení databáze po jejich ukončení.

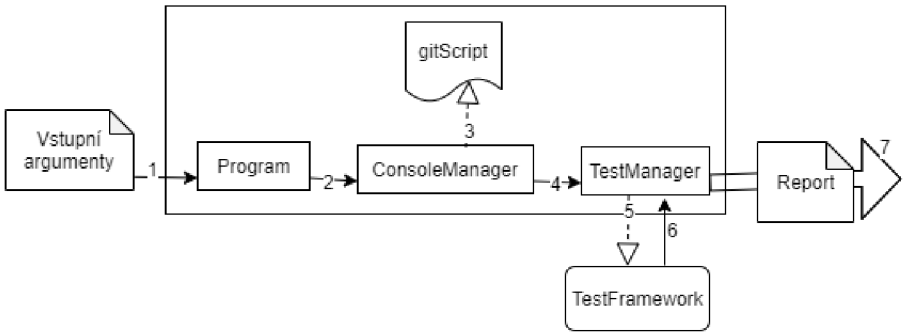
Jak je vidět na diagramu 5.1 (vlevo dole), bude možné nástroj spouštět jak testerem, tak automatizovaně jiným procesem.



Obrázek 5.1: Zde je vidět konceptuální řešení všech částí nástroje pro automatizaci GUI testů, spolu s možností generování reportu. Testovací framework obsahuje předem vytvořené testy. Využitím modulu AuTeReporter je potom z testů vytvořený report. Diagram nezahrnuje akce spojené s databází. O ty se nebude starat žádný z navrhovaných nástrojů.

5.3 Strukturní návrh konzolové aplikace AuTeR

V této části bude popsán návrh konzolové aplikace AuTeR. Aplikace typu konzole byla vybrána proto, aby bylo snadné ji spouštět například z příkazového řádku. Na diagramu 5.2 jsou zobrazeny nejdůležitější moduly konzole. Zároveň je pro úplnost zobrazena i část s Testovacím Frameworkem (viz sekce 6.1.3), který zde využívá reportovacího frameworku AuTeReporter. Vytvořený report bude pak možné podle specifikace odeslat na předem zadané adresy elektronické pošty.



Obrázek 5.2: Diagram zobrazuje vstup(1) a výstup(6 respektive 7). Výstupem systému může být pouze vytvoření reportu(6), nebo jeho následné odeslání(7). Může také pouze dojít ke spuštění testovacího běhu(5), pokud by nešlo o testovací projekt využívající reportovacího frameworku a následně ukončení aplikace AuTeR. Hrana (1) označuje vstupy, které jsou zpracovány modulem *Program*. Poté jsou hodnoty předány modulu *ConsoleManager*(2), který provede kontrolu hodnot a spouští skript *gitScript*(3). Tím jsou zajištěny nutné přípravy zahrnující stažení změn z repozitáře a procesy sestavující projekty. Po kontrole je pomocí funkce *Run()* modulu *TestManager*(4) spouštěn projekt s testy(5). Po dokončení testů je k dispozici vygenerovaný *Report*, který je možné díky funkci *SendReport()* v modulu *TestManager* odeslat na zadané adresy(7) přes elektronickou poštu.

5.3.1 Modul Program

Modul *Program* je prvním spouštěným modulem aplikace AuTeR. Jeho úkolem je ověřit správnost vstupů při spuštění nástroje a ověřit správné formáty hodnot. Poté spouští modul ConsoleManager funkcí *Process()*.

5.3.2 Modul ConsoleManager

Tento modul má za úkol zkontrolovat řetězce představující cesty k souborům z modulu *Program* a ověřit jejich existenci v systému, na kterém jsou testy spouštěny. Modul také bude spouštět skript *gitScript*, díky kterému budou sestaveny potřebné soubory. Pokud je na vstupu specifikován projekt SUT, podle požadavků má modul být schopný provést i stažení nových změn z repozitáře.

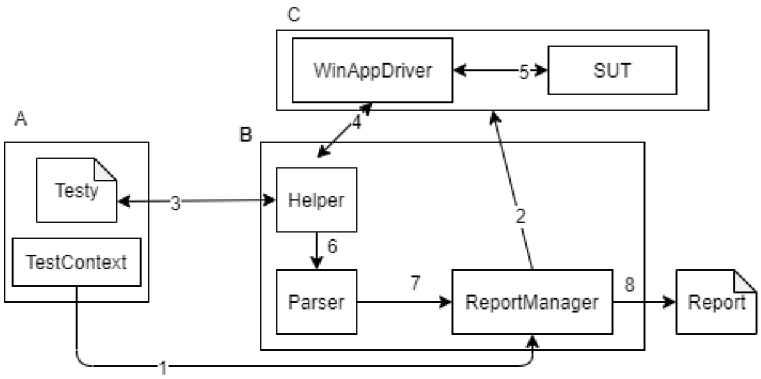
5.3.3 Modul TestManager

Jeden z dalších požadavků bylo automatizované spuštění testů. Modul obsahuje metodu *Run()*, která tuto akci bude vykonávat. Po dokončení spuštěných testů bude možné očekávat report. Ten má být možné odeslat přes elektronickou poštu na adresy specifikované v souboru, jehož cesta byla uvedena jako jeden ze vstupních argumentů konzole. Tento proces bude prováděn metodou *SendReport()*. Po dokončení odesílání, nebo samotném ukončení testů, bude proces konzole ukončen.

5.4 Strukturní návrh frameworku AuTeReporter

V následující části budou popsány a představeny hlavní moduly reportovacího frameworku AuTeReporter. Úkolem tohoto frameworku však nebude jen sbírání informací z testů a ge-

nerování reportu. Framework zajistí interakci s grafickým rozhraní SUT a umožní tak provádění akcí testovacích případů obsažených ve spouštěných testovacích projektech. Diagram 5.3 znázorňuje vztahy mezi moduly. Hlavním modulem je *ReportManager*, který inicializuje práci reportovacího frameworku. Vstupními hodnotami jsou hodnoty z *TestContext* třídy, která popisuje prostředí a hodnoty spouštěných testů, spolu s cestami k cílovým souborům. *ReportManager* je také modulem, který spouští jak server *WinAppDriver*, tak spustitelný soubor SUT a zajištuje spojení mezi nimi.



Obrázek 5.3: Následující diagram znázorňuje nejdůležitější moduly frameworku AuTeReporter. První část diagramu **A** představuje spouštěný projekt s testy. Prostřední část **B** představuje framework AuTerReporter. A poslední část **C** jsou spouštěné aplikace. Modul *ReportManager* je spuštěním testů inicializován a jako parametr je mu předána instance objektu *TestContext*(1). Ten je implementován testovacím frameworkem a jsou v něm obsaženy důležité informace o vykonávaných testech. Touto cestou je předána informace o absolutní cestě ke spustitelnému souboru *SUT*. Následně je modulem *ReportManager* spuštěn *WinAppDriver server* a je přípojen k *SUT*(2). Poté probíhá vykonávání testů. Testy využívají rozhraní modulu *Helper*(3), který posílá příkazy *serveru*(4), a ten provádí akce nad *SUT*(5). Výsledky akcí se pak stejnou cestou vracejí k modulu *Helper*. Ten předá informace modulu *Parser*(6), který informace vkládá do instance objektu, který představuje výsledný *Report* a je jedním z vlastností modulu *ReportManager*(7). Po dokončení testů je tento objekt převeden metodou *HtmlCreater()*, do podoby výsledného Reportu(8).

5.4.1 Modul ReportManager

Modul má za úkol inicializovat potřebné zdroje a instance frameworku AuTeReporter. Jak již bylo zmíněno v části 5.4, bude jím zajištěno spuštění serveru WinAppDriver i aplikace, nad kterou budou testy prováděny. Pro zjištění cesty ke spustitelnému souboru SUT bude použit objekt *TestContext*, který je implementován testovacím frameworkem (viz sekce 6.1.3). Tento objekt obsahuje informace o prostředí, ve kterém jsou testy spuštěny, spolu s dalšími předem definovanými parametry (viz sekce 6.2.1).

Pro interakci se SUT bude použit objekt *desktopSession*, který obsahuje instanci WindowsDriver. Ten bude představovat grafické rozhraní SUT (viz sekce 3.1.2). Instance je implementována WindowsDriver aplikačním rozhraním a rozhraním modulu *Helper*. Tento návrh byl převzat z dokumentace WinAppDriver, viz [36].

Vzhledem ke způsobu spojení mezi SUT a WinAppDriver, modul *ReportManager* bude implementovat několik metod, kterými bude řízen průběh vykonávání testů. Tyto metody

je nutné použít v projektu s testy nejen pro funkci generování reportu, ale hlavně kvůli ovládání SUT přes WinAppDriver server. Převzato z [36].

- Metoda *SetUp(TestContext context)* zajišťuje připojení WinAppDriver serveru k SUT. Zde se inicializuje proměnná *desktopSession*.
- Metoda *TearDown()* zajišťuje ukončení spojení mezi WinAppDriver serverem a SUT. V této metodě je *desktopSession* zrušena.
- Metoda *Initialize(TestContext context)* inicializuje framework přistoupením k hodnotám v proměnné *context* a získává informace o SUT. Spouští server WinAppDriver a inicializuje novou instanci *ReportModel*, představující report. Tato funkce musí být obsažena v testovací sadě v metodě s konkrétním atributem. V kombinaci s konzolí AuTeR jde o součást automatizovaného spuštění.
- Metoda *Initialize(string _application)*, jde o přetížení metody *Initialize()*. Tuto metodu lze použít v případě ladění testovací sady. Bude spuštěn server, ale nebude se generovat report.
- Metoda *Initialize(string _application, string _resultDirectory)* je psoledním typem metody pro inicializaci frameworku. Tato metoda spolu se spuštěním a provedením testů umožňuje vytvářet i report. Tím pádem lze pro generování reportu a provádění testů použít pouze tento framework.
- Metoda *FinalTasks()* je provedena jako poslední z celého testovacího běhu. Tato metoda ukončuje proces serveru a SUT, a popřípadě generuje report do výsledné podoby.

5.4.2 Modul Helper

Tento modul je prostředníkem mezi testovacím frameworkem a *WinAppDriver* serverem, díky kterému lze SUT ovládat. Splněním požadavku snadného používání pro generování reportu *Helper* bude reimplementovat a rozšiřovat již existující metody *WindowsDriver API*. Pokud budou testy vytvořené za pomoci například nástroje WinAppDriverUIRecorder, nebude nutné provádět větší změny, aby reportovací framework fungoval. Samotná kontrola vyhledaných elementů bude pak provedena metodami modulu *Parser* (viz sekce 5.4.4).

Dále má *Helper* podle požadavků obsahovat i metody pro vkládání poznámek mezi testovací případy pro zachování kontextu testů.

5.4.3 Vyhledání elementu

V této části bude blíže popsán pseudoalgoritmus, jakým jsou navrženy metody v modulu *Helper*. Každá z metod přijímá jinou vlastnost, podle které je element vyhledáván. Na začátku je vytvořen prázdný objekt elementu WindowsElement. Každá z metod má v sobě cyklus, ve kterém se pokusí díky *WindowsDriver API* element vyhledat. Na vyhledání je zadaný počet pokusů, aby se předešlo případnému zablokování. Pokud je hledání úspěšné nebo dojde počet pokusů, cyklus se ukončí. Poté je provedena kontrola elementu. Kontrolu provádí modul *Parser*. Po dokončení kontroly je řízení vráceno modulu *Helper*, který vrací objekt element testovacímu frameworku.

Input: XPath|AccessibilityId|Name
Output: element

```
1:  WindowsElement element = null;
2:  pocet_pokusu = 3;
3:  while pocet_pokusu > 0 do
4:      element =
          WindowsDriver.FindElementBy(XPath|AccessibilityId|Name);
5:      if element != null then
6:          ukonči cyklus;
7:      end if
8:      else
9:          Počkej 0.5s;
10:     end if
11:     Získej informace o elementu a vlož je do objektu TestCaseModel;
12:     pocet_pokusu = pocet_pokusu −1;
13: end while
14: return element
```

5.4.4 Modul Parser

Tento modul implementuje metody ukládající informace o elementech. Získané informace jsou ukládány do instancí třídy *TestCaseModel*. Jednou z hlavních metod modulu je *SetInfo()*, která jako argumenty přijímá vyhledaný element, vlastnost, podle které byl získán, a objekt WindowsDriveru (desktopSession) pro získání dalších informací.

Podle algoritmu 1 má kontrola dva scénáře.

1. Pokud je z modulu *Helper* předán element, který není prázdným objektem - hledaný objekt na obrazovce byl úspěšně nalezen - vytvoří se nová instance objekt *TestCaseModel* s atributem výsledek = úspěšný. Do ní budou uloženy detaily o metodě, kterou byl element nalezen, jeho typ (tlačítko, okno, textové pole...), Id elementu, velikost, poloha a další. V případě, že se jedná o element typu tlačítko, bude podle požadavků získán snímek obrazovky a samotného tlačítka.

2. Pokud je z modulu *Helper* předán element, který je prázdným objektem - hledaný objekt na obrazovce nebyl nalezen - vytvoří se nová instance objekt *TestCaseModel* s atributem výsledek = neúspěšný. Do instance bude uložena vlastnost, podle které se WindowsDriver pokusil objekt nalézt. A bude vytvořen snímek aktuální obrazovky pro případné zachycení příčiny selhání.

5.4.5 Interpretace testů

Reportovací framework interpretuje každé použití některé z metod modulu *Helper*, které používá WindowsDriver API, jako jeden testovací případ. Kromě případu metody pro vytvoření poznámky. Jak bylo popsáno v algoritmu 1, získávání elementů na obrazovce funguje na principu: najdi a vrať. Pokud bude procesem hledaný element nalezen, testovací případ byl úspěšný a podle toho také modul *Parser* v příslušné nové instanci *TestCaseModel* bude nastaven atribut výsledku jako „úspěšný“. V opačném případě se bude brát testovací případ za neúspěšný a hodnota atributu bude „neúspěšný“.

5.4.6 Report

Tato část se bude věnovat návrhu reportu. V něm mají být obsaženy detaily o testech, jejich průběhu a informace o akcích, které byly provedeny nad SUT. Po dokončení testů budou tyto informace převedeny do formy HTML souboru pomocí funkce *HtmlCreater()*. Celý průběh vytváření reportu probíhá ve frameworku AuTeReporter.

Základní atomickou částí reportu je třída *TestCaseModel*. Tato třída představuje objekt informace, která přichází z jednoho vyvolání některé z funkcí modulu *Helper*. Třída obsahuje několik atributů. Tím nejdůležitějším je typ akce, díky které byl objekt vytvořen a při vzniku nové instance je tato informace do něj uložena. Těmito akcemi jsou v dosavadním řešení: **note** (poznámka) a **find** (nalézt). Tedy pokud bude vyvolána akce vyhledání elementu, bude vytvořena instance *TestCaseModel*, které bude nastaven atribut akce na **find**. Dále budou uloženy informace o elementu, jako jeho typ (tlačítko, okno, textové pole,. . .), jeho Id, popřípadě text, který je v něm zobrazený. Pokud se bude jednat o tlačítko, budou pořízeny dva snímky: samotného elementu a obrazovky před provedením akce kliknutí. Pokud je vyhledání neúspěšné nebo je uložena do atributu *info* poznámka o hodnotě vlastnosti, podle které se pokusil WindowsDriver element najít. Pokud půjde o vytvoření nové poznámky, je do atributu akce nastavena hodnota **note** a do atributu *info* řetězec, který byl vyvoláním metody předán jako parametr.

Nad třídou *TestCaseModel* je třída *TestMethodModel*, která představuje jednu testovací metodu. Tato třída obsahuje několik atributů, jako pořadí v rámci reportu, jméno metody a atribut *cases*. Tento atribut je polem instancí objektů *TestCaseModel*. Zde jsou uloženy všechny testovací případy, které byly v rámci metody vykonány. Pokud je tedy vykonána nová akce a byla dokončena kontrola, je nová instance uložena do tohoto pole. Metoda v rámci jednoho reportu je minimálně jedna.

Každá instance *TestMethodModel* je součástí jedné instance *ReportModel*. Jde o poslední třídu v hierarchii testu. *ReportModel* obsahuje atributy se jménem testovacího projektu, cestou k tomuto projektu a jméno SUT, nad kterým jsou testy reportovány. Dále obsahuje atribut *methods*, který obsahuje pole instancí třídy *TestMethodModel*.

Po dokončení testovacího běhu, nebo po jeho přerušení, je z objektu *ReportModel* generován soubor ve formátu HTML. Všechny objekty jsou převedeny do podoby HTML řetězců, které jsou po té vpisovány do nového souboru, kterému je automaticky vygenerováno jméno s aktuálním časem a datem. Ten je pak uložen do složky, která byla specifikována buď na začátku testů, nebo získáním z instance objektu třídy *TestContext* (viz sekce 5.4.1).

Kapitola 6

Implementace

V následující kapitole budou popsány implementační detaily obou vyvíjených nástrojů. Budou zmíněny využité existující technologie a balíčky. Volba technologií úzce souvisí s prostředím, ve kterém zadavatel pracuje (viz sekce 4.3).

Konzolová aplikace i framework jsou implementované v jazyce C# s použitím frameworku a počítačové platformy .NET. Projekty byly vyvíjeny na operačním systému Windows 10. Úprava zdrojových kódů byla prováděna na platformě Visual Studio.

6.1 Vybrané nástroje

V této části jsou představeny již existující nástroje, které byly v řešení použity. Většina z nich je buď součástí základního balíčku Visual Studio Community 2019[1], dostupná přes balíčkovací systém NuGet[2] nebo volně dostupná na webových stránkách GitHub[3].

6.1.1 WinAppDriver

Tento nástroj byl zvolen zejména díky kompatibilitě se systémem Windows a s podporou práce s aplikacemi, využívající grafického rozhraní vytvořené pomocí frameworků WinForms, WPF nebo DevExpress. Je jednou z částí modulu AuTeReporter (viz sekce 6.2.2).

Kromě samotného serveru je také využit nástroj WinAppDriverUIRecorder (viz sekce 3.3.2). Ten bude možné použít k nahrání testovacích běhů. Vygenerované testy bude možné uložit do projektu testů MSTest. Po menších úpravách je možné testy automatizovaně nad aplikací SUT provádět a generovat výsledný report (viz sekce 6.2.5).

6.1.2 Automatizace spuštění testů

Visual Studio poskytuje několik prostředí příkazové řádky pro vývojářskou práci. Jedním z nich je VsDevCmd [32]. Jde o standardní příkazový řádek s výchozím umístěním ve složce `C:\ProgramFiles(x86)\MicrosoftVisualStudioV.w\Common7\Tools`. Ten je použit pro spuštění projektu s testy. Spolu s využitím balíčků *xUnit* a třídy systémové knihovny *System.Diagnostic.Process* (viz sekce 6.2.3), je při spuštění testů vytvořen nový proces. Ten je přesunut do složky s příkazovým řádkem VsDevCmd a zde spuštěn.

[1] `https://visualstudio.microsoft.com/cs/vs/community/`

[2] `https://www.nuget.org/`

[3] `https://github.com/`

Pro úspěšné spuštění testů přes VsDevCmd je nutné předat důležité vstupní argumenty. Příkazový řádek umožňuje spouštět testy v různých konfiguracích, s větším množstvím testovacích projektů, nebo v různých režimech [32]. V rámci řešení byl zvolen nejjednodušší a nejméně náročný způsob, aby spuštění bylo spolehlivé a bylo možné proces úspěšně zautomatizovat. Vstupní argumenty jsou obstarány nástrojem AuTeR (viz sekce 6.2.1).

Proces při spouštění příkazového řádku předává vstupní argumenty. Těmi jsou absolutní cesty k souborům typu *.sln*, *.dll*, které jsou součástí projektů s testy (viz sekce 6.2.1), a soubor *.runsettings* (viz sekce 6.2.1).

6.1.3 Testovací framework MSTest

MSTest[4] je rozhraní Microsoft Test Framework. Je určený pro všechny jazyky rozhraní .NET, rozšiřitelné a kompatibilní se sadou Visual Studio. MSTest je v kombinaci s čistě Microsoft technologiemi nejvhodnější. Proto byl v řešení zvolen. Jde o jeden z frameworků kolekce *xUnit* (Unit Testing Frameworks) – Jednotkové testovací frameworky. [22]

Testovací běhy jsou tvořené projekty, který obsahují kromě souborů s knihovnami a konfiguračními soubory, také soubor se jménem testovacího projektu typu *.sln*. Po přeložení a sestavení tohoto souboru je vytvořena složka s binárními soubory a soubory typu *.dll*. Soubor *.sln* obsahuje zdrojový kód testovacích metod a funkcí. Při vytváření tohoto souboru je nutné dodržet syntax, který specifikuje MSTest framework a je na něm závislá implementace frameworku AuTeReporter.

Syntax testů Syntax souborů s testy určují funkce, objekty a atributy, které jsou implementovány frameworkem MSTest [9]. Následující představené objekty jsou součástí jmenného prostoru *Microsoft.VisualStudio.TestTools.UnitTesting* (viz [8]). Nejdůležitějším objektem je třída s atributem *TestClass*. Tato třída představuje celý jeden testovací běh, obsahuje všechny testovací případy a je v projektu povinná. Tříd s tímto atributem může být v rámci jednoho celého testovacího projektu více.

Každá třída s atributem *TestClass* obsahuje metody s atributem *TestMethod*. Metoda má představovat jeden testovací případ, nebo ucelenou skupinu funkcí a proměnných v rámci jednoho testovacího případu.

Kromě atributů, které specifikují logické části testů, lze díky dalším takovým specifikovat prostředí testů, nebo připravit stav SUT pro jejich provedení. Tyto atributy jsou přiřazovány metodám, které jsou spuštěny před provedením, nebo hned po provedení dané části testovacího projektu. V rámci testování nejsou povinné, ale mohou být užitečné například v případě uvolnění využitých zdrojů po provedení testů. Těchto atributů je velké množství. Ale v následujících odstavcích budou představeny stěžejní v rámci řešení frameworku AuTeReporter.

Jak bylo zmíněno v odstavcích výše, může být v rámci jednoho projektu specifikováno více testovacích tříd typu *TestClass*. Všechny tyto třídy jsou označované jako shluk (*Assembly*). Pro specifikování prostředí a proměnných v rámci celého projektu, lze použít atributy *AssemblyInitializeAttribute* a *AssemblyCleanUpAttribute*. S těmito atributy mohou být specifikované pouze jednou v rámci celého projektu a každá je provedena právě jednou.

Pro specifikaci prostředí a proměnných v rámci jedné testovací třídy, lze použít atributy *ClassInitializeAttribute* a *ClassCleanUpAttribute*. Podobně jako u atributů *AssemblyInitialize* a *AssemblyCleanUp*, jsou tyto atributy specifikovány v rámci třídy jen jednou.

[4] `https://docs.microsoft.com/cs-cz/dotnet/core/testing/#mstest`

V rámci celého testovacího běhu lze získávat informace o projektu s testy skrze tzv. *TestContext* [12]. Tato třída je k dispozici po celou dobu běhu testů. Umožňuje přístup k informacím jako je absolutní cesta k testovacímu projektu, jeho shluku, jméno a cesta pracovního adresáře. Kromě toho obsahuje i proměnnou *Properties* [13]. Jde o list typu slovník, který lze nastavovat před spuštěním samotných testů pomocí konfiguračního souboru *.runsettings*. Tento konfigurační soubor je využíván konzolovou aplikací AuTeR pro předávání důležitých informací reportovacímu frameworku (viz sekce 6.2.1).

6.2 Implementované části

Zde budou popsány implementované části řešení. Budou popsány implementační detaily konzolové aplikace i reportovacího frameworku. Tato část bude zmiňovat i podskupinu implementovaných a splněných požadavků, stejně tak i plánovaná vylepšení.

6.2.1 Konzole AuTeR

Aplikace AuTeR byla vyvíjena jako konzolová aplikace. Projekt byl vytvořen a vyvíjen na vývojové platformě Visual Studio, které poskytuje předpřipravené šablony různých typů[5]. Jednou z nich je i konzolová aplikace, viz sekce [6]. Inspirací v interakci s uživatelem a vzhledu byl příkazový řádek VsDevCmd (viz sekce 6.1.2). Cílem bylo vyvinout aplikaci, která by nejvíce zapadala do prostředí systému Windows.

Aby bylo možné spouštět konzoly automatizovaně a nebyla nutná přítomnost člověka, je celá práce prováděna bez dalších nutných uživatelských vstupů. Všechny důležité hodnoty jsou povinné při spuštění aplikace. Konzole AuTeR tedy definuje několik vstupních argumentů:

- **/TestSolutionPath:** – povinný – tímto argumentem je specifikována absolutní cesta k souboru typu *.sln* projektu s testy. Hodnota je využita k sestavování testů, vytváření binárních souborů a generování souboru typu *.dll*.
- **/TestAdapterPath:** – povinný – tento argument je absolutní cestou složky, ve které bude možné po sestavení testovacího projektu nalézt soubor *.dll*.
- **/TestResultsDirectory:** – volitelný – argumentem lze definovat cestu k místu v systému, kde bude po dokončení testů a vygenerování reportu umístěn výsledný report. Pokud nebude tento argument specifikován, bude report uložen ve složce *TestResults* v projektu s testy. Pokud taková složka nebude v daném projektu existovat, bude vytvořena.
- **/Executable:** – povinný – argument představuje absolutní cestu ke spustitelnému souboru *.exe* SUT.
- **-p [SUTSolutionPath]** – volitelný – přepínačem je specifikován požadavek stažení změn z repozitáře SUT. Hned po přepínači je nutné specifikovat absolutní cestu k souboru typu *.sln* projektu SUT.
- **-h** tímto přepínačem bude na výstup vypsána nápověda k aplikaci AuTeR.

[5] `https://docs.microsoft.com/cs-cz/dotnet/core/tutorials/with-visual-studio`

Automatizace překladu projektů Za účelem sestavování SUT a projektů s testy byl nejprve navržen skript *gitScript* (viz sekce 5.3). Ten byl později rozdělen do několika příkazů, které jsou spouštěny přímo z kódu pomocí třídy *System.Diagnostic.Process*. Touto akcí je spouštěn také nástroj Devenv, který v kombinaci s integrovaným prostředí *Visual Studio Community*[6] umožňuje zajistit automatizované sestavení a přeložení projektů.[38]

Automatizace stažení změn z repozitáře Podobně jako v části 6.2.1 byla nejdříve akce, která zajišťovala stažení změn projektů, navržena jako součást skriptu *gitScript* (viz sekce 5.3). Pro vykonání stažení je použita příkazová řádka *cmd.exe*, která provede příkaz **pull** v dané složce s projektem.[40]

Předávání informací spouštěným testům Aplikace AuTeR umožňuje spouštět automatizovaně testy. K tomu, aby byly spuštěny s takovými hodnotami, které byly vloženy na vstup aplikace, je použit soubor *.runsettings*. Tento soubor je konfigurace unit testů. Pro každý testovací běh je generován nový a při spuštění testů je předán jako jeden z argumentů příkazového řádku VsDevCmd, kterým budou testy spuštěny. Tento soubor obstarává aplikace AuTeR pomocí třídy *RunSettingFileManager*. Díky speciální syntaxi je možné předat testům informace, ke kterým pak lze přistupovat přes instanci objektu třídy *TestContext*[3]. Tímto způsobem je například předána cesta k souboru, do kterého bude později uložen výsledný report.

6.2.2 Framework AuTeReporter

Jedním z požadavků byla integrace reportovacího frameworku do platformy .NET. Proto byl AuTeReporter vytvořen jako Knihovna tříd .NET. Stejně jako konzole AuTeR, poskytuje Visual Studio předpřipravené šablony projektů, které mají plnit roli knihoven, viz [5].

Pro zjednodušení práce s typy testů v rámci počátečního vývoje frameworku, byl testovací framework MSTest zvolen jako prozatím jediný, který AuTeReporter podporuje. Proto se předpokládá, že všechny projekty s testy, které budou reportovací framework používat, budou zároveň používat i framework MSTest.

Vykonávání testů je řízeno frameworkem AuTeReporter. V rámci implementovaných metod je nejen prováděna kontrola elementů, ale také se rozhoduje, zdali budou testy pokračovat, nebo budou přerušeny. Protože jsou testy prováděny automatizovaně, a tudíž budou s největší pravděpodobností prováděny bez přítomnosti člověka, nebylo navrženo prozatím řešení, kterým by se dalo snadno rozhodnout, zda se SUT bude nacházet v takovém stavu, který by umožňoval v testování pokračovat. Proto bylo řízení testů navrženo tak, aby byl jejich běh přerušen hned jak selže první testovací případ. Aby bylo zajištěno vygenerování reportu s případným odhalením příčiny selhání, je tento proces ukončování řízen frameworkem.

Vytváření reportu Po dokončení nebo přerušení testovacího běhu, je vygenerován report na základě informací, které byly během testů získány. Report je převáděn do formy HTML prvků, které jsou vkládány do výsledného souboru. Místo, kam jsou soubory s reporty ukládány, lze definovat při spuštění frameworku jednou z metod (viz sekce 5.4.2).

[6] `https://visualstudio.microsoft.com/cs/vs/community/`

Pokud jsou testy spouštěny automatizovaně díky aplikaci AuTeR, jsou tyto informace obsaženy v souboru *.runsettings* jako konfigurace, který je pro každé nové spuštění testů generován jako nový soubor. Při spouštění příkazového řádku VsDevCmd je soubor předán jako jeden ze vstupních argumentů, (viz sekce 6.1.2). Podle konfigurace jsou v testovacím běhu vytvořeny proměnné, které jsou poté k dispozici v probíhajících testech přes objekt *TestContext*. Tímto způsobem je předána automatizovaně informace o místu, kam se má vytvořený report uložit.

6.2.3 Použité balíčky a knihovny

Kromě standardních systémových knihoven byly v projektech AuTeR a AuTeReporter použity následující balíčky:

AuTeR

- xunit.abstractions v2.0.3 a xunit.runner.utility v2.4.1 – Obě knihovny poskytují nástroje a funkce pro spouštění testů.

AuTeReporter

- Appium.WebDriver v4.1.1 – Balíček je rozšířením Selenium WebDriver. Obsahuje také implementaci pro ovládání aplikace WinAppDriver.
- MSTest.TestFramework v2.1.2 – Balíček s implementací rozhraní MSTest projektů. Díky tomuto balíčku je AuTeReporter kompatibilní s testy vytvořené díky frameworku MSTest.

Ze standardních systémových knihoven byly použity následující třídy:

- System.Diagnostic.Process v4.7.2 – třída byla použita pro spouštění akcí zahrnující stažení změn z repozitářů a překlad projektů. Také byla použita při spouštění testů.

6.2.4 Rozhraní frameworku AuTeReporter

Framework AuTeReporter implementuje funkce, které je možné použít v testovacích projektech MSTest. Díky znalosti aplikačního rozhraní MSTest framework, byly funkce navrženy tak, aby využily co nejvíce informací, které MSTest framework o testech bude poskytovat.

Následující metody jsou implementací modulu Helper (viz sekce 5.4.2). Tyto metody jsou použity pro interakci s grafickým rozhraním SUT. Argumenty metod–lokátory, pro vyhledání elementů odpovídají těm, které jsou definovány rozhraním WebDriver (viz tabulka 3.1).

- *FindByAbsoluteXpath(string xPath)* je metodou, která vyhledává element na základě jeho xPath vlastnosti, tedy cesty ve stromě objektů grafického rozhraní. Metoda je implementována metodou WindowsDriver rozhraní *FindElementByXpath(string xPath)*, která vrací první výskyt takového elementu typu WindowsElement. Pokud není žádný takový nalezen, vrací metoda hodnotu **null**.
- *FindByAccessibilityId(string accessibilityId)* je metodou, která vyhledává element podle hodnoty vlastnosti přístupový identifikátor. Tato metoda je implementována metodou aplikačního rozhraní WindowsDriver *FindElementByAccessibilityId(string accessibilityId)*. Jako výsledek je vrácen první výskyt elementu typu WindowsElement s takovou vlastností. Jinak je vrácena hodnota **null**.

- *FindByName(string name)* je metodou, která vyhledává elementy na obrazovce pomocí jejich jména. Implementací metodou *FindElementByName(string name)* rozhraní WindowsDriver je vrácen buď první výskyt elementu typu WindowsElement s takovou hodnotou atributu jméno, nebo hodnota **null**.
- *Note(string note)* je funkce, která umožňuje na dané místo vložit poznámku. Jako jediná neodpovídá žádné z metod, kterou implementuje rozhraní WindowsDriver a pouze dovoluje zpřehlednit informace, které jsou z testů poskytovány.

Všechny vyjmenované metody obsahují kromě metod aplikačního rozhraní WindowsDriver i metodu pro inspekci každého elementu, který bude nalezen. Kontrola je prováděna modulem Parser (viz sekce 5.4.4) přes instanci objektu *ReportManager*.

6.2.5 Příprava vygenerovaných testů pro možnost generování reportu

V následující části bude popsán postup přípravy testovacího běhu pomocí aplikačního rozhraní, které poskytuje AuTeReporter a projektu s již připravenými testy využívající frameworku MSTest. Pro vygenerování testů lze použít nástroj WinAppDriverUIRecorder (viz sekce 3.3.2). Vygenerované testy bude možné převést do podoby metod v jazyce C#. Z toho důvodu bude ukázka představena ve stejném jazyce.

Po vygenerování testovacích případů lze testy překopírovat do vytvořeného projektu s testy. Vzhled jednotlivých metod bude vypadat tak, jak je vidět na obrázku 6.1. Pro přehlednost byly zvýrazněny nejdůležitější části. Tento zdrojový kód by vykonal přesně ten samý nahraný proces, ale nebylo by možné zároveň generovat report. Funkce rozhraní WindowsDriver komunikují přímo se serverem WinAppDriver. Pro možnost zachycení komunikace mezi těmito body, byly v řešení použity reimplementované a upravené funkce, které vykonávají stejnou práci, jako metody původní a zároveň jsou sbírány informace do výsledného reportu. Na obrázku 6.2 je vidět použití nových metod, se stejným rozhraní.

Po úpravě testovacích metod je nyní potřeba nastavit nezbytné metody, které budou spouštěny před a po provedení testů. Díky možnosti použití atributů, které definuje a implementuje MSTest framework (viz sekce 6.1.3) AuTeReporter poskytuje funkce pro řízení testů a generování reportu (viz sekce 5.4.1). Na obrázku 6.3 je představen pseudoalgoritmus, který znázorňuje syntax testovacího projektu.

```
// LeftClick on Button "Jedna" at (78,38)
Console.WriteLine("LeftClick on Button \"Jedna\" at (78,38)");
string xpath_LeftClickButtonJedna_78_38 = "/Pane[@ClassName=\"#32769\"][@Name=\"Desktop 1
\"]/Window[@ClassName=\"ApplicationFrameWindow\"]
[@Name=\"Kalkulačka\"]/Window[@ClassName=\"Windows.UI.Core.CoreWindow\"][@Name=
\"Kalkulačka\"]/Group[@ClassName=\"LandmarkTarget\"]
/Group[@Name=\"Číselná klávesnice\"][@AutomationId=\"NumberPad\"]/Button[@Name=\"Jedna\"]
[@AutomationId=\"num1Button\"]";
var winElem_LeftClickButtonJedna_78_38 =
desktopSession.FindElementByAbsoluteXPath(xpath_LeftClickButtonJedna_78_38);
if (winElem_LeftClickButtonJedna_78_38 != null)
{
   winElem_LeftClickButtonJedna_78_38.Click();
}
else
{
   Console.WriteLine($"Failed to find element using xpath: {xpath_LeftClickButtonJedna_78_38}");
   return;
}

// LeftClick on Button "Dva" at (38,31)
Console.WriteLine("LeftClick on Button \"Dva\" at (38,31)");
string xpath_LeftClickButtonDva_38_31 = "/Pane[@ClassName=\"#32769\"][@Name=\"Desktop 1
\"]/Window
[@ClassName=\"ApplicationFrameWindow\"][@Name=\"Kalkulačka\"]/Window[@ClassName=
\"Windows.UI.Core.CoreWindow\"][@Name=\"Kalkulačka\"]
/Group[@ClassName=\"LandmarkTarget\"]/Group[@Name=\"Číselná klávesnice\"][@AutomationId=
\"NumberPad\"]/Button[@Name=\"Dva\"][@AutomationId=\"num2Button\"]";
var winElem_LeftClickButtonDva_38_31 =
desktopSession.FindElementByAbsoluteXPath(xpath_LeftClickButtonDva_38_31);
if (winElem_LeftClickButtonDva_38_31 != null)
{
   winElem_LeftClickButtonDva_38_31.Click();
}
else
{
   Console.WriteLine($"Failed to find element using xpath: {xpath_LeftClickButtonDva_38_31}");
   return;
}
```

Obrázek 6.1: Vzhled zkopírovaného vygenerovaného kód z nástroje WinAppDriverUIRecorder. Světle modré zvýrazněné části jsou výpisy do konzole WinAppDriver (viz sekce 3.3). Červeně jsou zvýrazněny řetězce představující xPath daných elementů (tlačítko 1 a 2 na Kalkulačce). Světle zeleně jsou zvýrazněny funkce rozhraní WindowsDriver, které bylo v rámci řečení upraveno (viz sekce 5.4.2).

```
// LeftClick on Button "Jedna" at (78,38)
Console.WriteLine("LeftClick on Button \"Jedna\" at (78,38)");
string xpath_LeftClickButtonJedna_78_38 = "/Pane[@ClassName=\"#32769\"][@Name=\"Desktop 1
\"]/Window[@ClassName=\"ApplicationFrameWindow\"]
[@Name=\"Kalkulačka\"]/Window[@ClassName=\"Windows.UI.Core.CoreWindow\"][@Name=
\"Kalkulačka\"]/Group[@ClassName=\"LandmarkTarget\"]
/Group[@Name=\"Číselná klávesnice\"][@AutomationId=\"NumberPad\"]/Button[@Name=\"Jedna\"]
[@AutomationId=\"num1Button\"]";
var winElem_LeftClickButtonJedna_78_38 =
desktopSession.FindByAbsoluteXPath(xpath_LeftClickButtonJedna_78_38);
if (winElem_LeftClickButtonJedna_78_38 != null)
{
   winElem_LeftClickButtonJedna_78_38.Click();
}
else
{
   Console.WriteLine($"Failed to find element using xpath: {xpath_LeftClickButtonJedna_78_38}");
   return;
}

// LeftClick on Button "Dva" at (38,31)
Console.WriteLine("LeftClick on Button \"Dva\" at (38,31)");
string xpath_LeftClickButtonDva_38_31 = "/Pane[@ClassName=\"#32769\"][@Name=\"Desktop 1
\"]/Window
[@ClassName=\"ApplicationFrameWindow\"][@Name=\"Kalkulačka\"]/Window[@ClassName=
\"Windows.UI.Core.CoreWindow\"][@Name=\"Kalkulačka\"]
/Group[@ClassName=\"LandmarkTarget\"]/Group[@Name=\"Číselná klávesnice\"][@AutomationId=
\"NumberPad\"]/Button[@Name=\"Dva\"][@AutomationId=\"num2Button\"]";
var winElem_LeftClickButtonDva_38_31 =
desktopSession.FindByAbsoluteXPath(xpath_LeftClickButtonDva_38_31);
if (winElem_LeftClickButtonDva_38_31 != null)
{
   winElem_LeftClickButtonDva_38_31.Click();
}
else
{
   Console.WriteLine($"Failed to find element using xpath: {xpath_LeftClickButtonDva_38_31}");
   return;
}
```

Obrázek 6.2: Upravený zdrojový kód. Nové funkce jsou vyznačeny sytě žlutou barvou.

```
namespace TestEditMaterializeCard
{
  [TestClass]
  public class CalculatorTests : ReportManager.ReportManager
  {
    [AssemblyInitialize()]
    public static void AssemblyInitialize(TestContext context)
    {
      Initialize(context);   <--- inicializace frameworku AuTeReporter
    }

    [AssemblyCleanup()]
    public static void AssemblyCleanup()
    {
      FinalTasks();   <--- zajištění vygenerování reportu po provedení všech testů
    }

    [ClassInitialize()]
    public static void ClassInitialize(TestContext context)
    {
      Setup(context); <--- zajištění spojení v každé testovací metodě
    }

    [ClassCleanup()]
    public static void ClassCleanup()
    {
      TearDown();  <--- uvolnění zdrojů po dokončení testovací metody
    }

    [TestMethod]
    public void MainTestMethod()
    {
      .
      .          <--- Testovací případy
      .
    }
}
```

Obrázek 6.3: Pseudoalgoritmus objektů v testovacím projektu pro správný běh nástroje AuTeReporter. Jsou zde využity metody rozhraní modulu *ReportManager* (viz sekce 5.4.1)

6.3 Implementované požadavky

V následující části bude představen přehled požadavků, které byly splněny v rámci implementace, a které budou splněny v rámci plánovaných rozšíření.

6.3.1 Přehled implementovaných požadavků

V následující tabulce budou vyjmenované požadavky, které byly implementací splněny.

Id	Popis	Část
REQ.2	viz sekce 6.2.1	konzole
REQ.3	Reportovací framework AuTeReporter byl vyvíjen jako knihovna metod, které bude možné v testech použít, viz sekce 6.2.4 a 5.4.1	framework
REQ.4	Výsledný report je generován do formy HTML souboru, viz sekce 5.4.6.	framework
REQ.5	viz sekce 6.2.1	konzole
REQ.6	Nástroj bude kompatibilní s testy využívající framework MSTest nástroje Visual Studio.	framework
REQ.7	viz sekce 3.3	framework
REQ.8	Oba nástroje byly vyvíjeny a testovány na operačním systému Windows 10.	framework, konzole
REQ.9	Pro vytváření testů nad GUI je možné použít nástroj WinAppDriverUIRecorder, viz sekce 3.3.2.	framework
REQ.11	viz sekce 6.2.4	framework, report
REQ.12	viz sekce 6.2.4	framework, report

Tabulka 6.1: Tabulka obsahuje seznam implementací splněných požadavků s krátkým popisem. Všechny identifikátory se odkazují na tabulku 5.1.

6.3.2 Plánovaná rozšíření

V této části budou jmenovány plánovaná rozšíření jak v aplikaci AuTeR, tak ve frameworku AuTeReporter. Také budou zmíněny některé požadavky, které budou splněny v rámci rozšíření.

Jedním z požadavků byla možnost odesílání vygenerovaných reportů na příslušné adresy využitím služeb elektronické pošty, viz REQ.10 v tabulce 5.1. Tento požadavek bude implementován jako jedna z funkcí konzole AuTeR, spolu se vstupním argumentem, kterým bude specifikována cesta k souboru se seznamem adres. Tato funkcionalita bude spojena s možností vytváření konfiguračních souborů testovacích běhů. Tyto soubory budou obsahovat argumenty stejné, jaké jsou nyní vkládány na vstup aplikaci AuTeR. Tyto soubory budou uloženy a uchovány v příslušné složce aplikace, aby byly vždy k dispozici a bylo možné vytvářet dávky testů, které budou jednotlivě spouštěny.

V rámci metod, kterými lze vytvářet report ve frameworku AuTeReporter, jsou plánovaná rozšíření o metody, které by umožňovaly pokračovat v provádění testů i po selhání některého z testovacích případů. Také jsou plánovaná rozšíření o metody, které by více prohledávaly jednotlivé elementy, nebo ověřovaly konkrétní vlastnosti elementů.

V rámci navazování spojení mezi frameworkem, SUT a WinAppDriver serverem bude možné použít v testech metody, které umožní připojit se k již spuštěné aplikaci. Toto rozšíření bude umožňovat spouštět více testovacích projektů nad jedním aktuálním procesu SUT, ale také vytvářet sekvence, které uvedou SUT do potřebného stavu před spuštěním testů, po jejich spuštění, nebo umožní regeneraci SUT po případném selhání testů.

Kapitola 7

Testování a Evaluace

V této kapitole bude popsán průběh testování obou nástrojů. Pro ověření správnosti bylo použito jednotkové a integrační testování s využitím testovacího frameworku MSTest. MSTest framework byl zvolen v rámci samotného integračního testování s frameworkem AuTeReporter, protože se touto cestou ověřila i funkčnost s daným typem testů podle požadavku REQ.6 v tabulce 5.1.

Pro každý nástroj byl vytvořen nový projekt, AuTeRTests a AuTeReporterTests, kde každý obsahuje několik sad testů. Oba projekty jsou uloženy na paměťovém médium. Testování probíhalo na dvou zařízení se stejným prostředím, tedy operační systém Windows 10. Oba nástroje byly testovány na aplikaci Kalkulačka systému Windows a na desktopové aplikaci zadavatele.

7.1 Jednotkové testování

V této části bude popsán způsob jednotkového testování obou nástrojů.

7.1.1 Jednotkové testy konzolové aplikace AuTeR

Projekt s testy pro nástroj AuTeR, AuTeRTests, se skládá ze dvou sad testů. První sada pokrývá implementaci modulu *Program*, který ověřuje validitu vstupních hodnot. Pro pokrytí všech možných kombinací vstupů při spuštění aplikace byly vytvořeny testy ověřující:

- prázdné vstupy
- neznámé řetězce
- parametry bez definovaných hodnot, nebo s hodnotami ve špatném formátu
- neznámé parametry
- špatné pořadí parametrů

Druhá sada ověřuje, zda aplikace dokáže rozpoznat nevalidní zadané hodnoty, které jsou ve správném formátu. Těmito testy byla pokryta implementace instance *ConsoleManager*, která ověřuje existenci všech zadaných cest a souborů. Také jsou ověřeny funkce, které zajišťují čištění projektů, sestavování a případné stahování nových změn z repozitáře. Proto byly vytvořeny testy s předpokládanými výstupními hodnotami, kterými byly ověřeny:

- stavy, kdy zadaná cesta k souboru typu *.exe* nebo *.sln* neexistuje

- stavy, kdy zadaná cesta ke složce se shluky nebo složce, do které má být uložen výsledný report, neexistuje
- stav, kdy projekt nebyl úspěšně sestaven
- stav, kdy se nepovedlo stažení změn projektu

7.1.2 Jednotkové testy frameworku AuTeReporter

Projekt s testy pro framework AuTeReporter, AuTeReporterTests, obsahuje kromě integračních testů i sadu jednotkových testů. Testy v této sadě je ověřena funkčnost metod vytvářející reporty. Testovacími případy a očekávanými hodnotami je ověřeno, jestli jsou objekty testovacích případů vytvářeny správně, se správnými hodnotami, jak se ukládají informace do nových instancí a jak jsou tyto instance ukládány do struktury reportu.

```
ReportManager.ActualReportModel.NewCase(uiTarget, xPath: xPath);
Assert.IsEqual(1, ReportManager.ActualReportModel.actualMethod.cases.Count());
```

Obrázek 7.1: Snímek zobrazuje testovací případ vytvoření nového záznamu v reportu.

7.2 Integrační testování

V této části bude popsán způsob integračního testování frameworku AuTeReporter. Integračním testováním se ověřuje, zda komunikace mezi aplikacemi nebo nástroji probíhá správně. I přes to, že lze oba vyvíjené nástroje použít společně, jejich interakce neprobíhá přímo. Testy, které mohou využívat frameworku AuTeReporter, jsou nástrojem AuTeR spouštěny pomocí příkazové řádky VsDevCmd (viz sekce 6.1.2). Při spouštění jsou předávány informace přes příslušné soubory (viz sekce 6.2.1), které jsou stěžejní uvnitř probíhajících testů. O to důležitější je ověření správného chování, pokud bude vynechán nástroj AuTeR, protože spolu s tím nebude provedena kontrola, která je jinak v konzoli prováděna. Proto byly vytvořeny integrační testy, které ověřují funkčnost rozhraní frameworku AuTeReporter.

Integrační testy tvoří několik sad testů uvnitř projektu AuTeReporterTests. Ty ověřují funkčnost komunikace mezi AuTeReporter, testovacím frameworkem MSTest a prostředím, kde jsou testy prováděny. Podle funkcí rozhraní, která byla představena v sekci 5.4.1 a 5.4.2, byly testy navrženy tak, aby ověřovali jejich funkčnost. První sada pokrývá rozhraní instance *ReportManager*. Metody tohoto rozhraní umožňují ovládat běh testů, spouštění SUT a spouštění WinAppDriver serveru. Před spuštěním samotných testů jsou ověřeny a ošetřeny následující stavy:

- cesta k souboru SUT typu *.exe* neexistuje
- cesta ke složce, kde má být uložen výsledný report neexistuje
- nelze najít a spustit server WinAppDriver.exe

Pokud některý z těchto stavů nastane, není možné dále testy provádět.

Testování rozhraní instance *Helper* probíhalo již od začátku vývoje. Díky opakovanému používání a ladění v testovacích projektech pro aplikace Kalkulačka, podléhalo rozhraní kontrole nejdéle ze všech částí vyvíjených nástrojů.

Kapitola 8

Závěr

Cílem této práce bylo navrhnout a implementovat nástroj, kterým bude možné ulehčit vykonávání GUI testy nad aplikací v rámci regresního testování a vytvářet o jejich průběhu report. Po prostudování možných technologií a frameworků z oblasti testování grafického uživatelského rozhraní, byl zvolen framework WinAppDriver a navrženy dva nástroje. Pro samotné vykonání testů byl vyvinut framework AuTeReporter, který umožnil interakci s GUI právě díky zvolené technologii WinAppDriver a následné generování reportu. Pro automatizaci spouštění testů a automatizované vykonávání dalších potřebných akcí byl vyvinut nástroj AuTeR, který umožňuje testy připravit a spustit.

Díky možnosti využití technologie WinAppDriver, která je navíc otevřeným softwarem, se podařilo navrhnout a vytvořit rozhraní, které lze v testech snadno použít a zároveň zajistit sbírání informací. Díky již existujícímu aplikačnímu rozhraní WindowsDriver a WebDriver lze sbírat informace o elementech na obrazovce, jejich snímky i stavy, a ty pak vkládat do generovaného reportu.

Naopak se nepodařilo zcela nahradit a reimplementovat již existující metody bez toho, aby nebylo nutné přepisovat již vygenerované testy pomocí nástroje WinAppDriverUIRecorder. Další fáze vývoje se ale tomuto problému budou hlouběji věnovat. V budoucnu bude rozhraní frameworku AuTeReporter rozšiřováno a vylepšováno podle potřeb a požadavků testerů.

Implementací konzole AuTeR se podařilo automatizovat přípravu projektů před spuštěním testů, a tím umožnit urychlení a zautomatizování nejen testování. V dalších fázích vývoje bude snaha o rozšíření funkcionality o odesílání vygenerovaného reportu přes elektronickou poštu, možnosti vytváření sad testů, nebo vytváření logických scénářů, a tím zefektivnit a urychlit regresní testování.

Literatura

[1] Process Automation Solutions [cit. 2021-05-05]. Dostupné z: https://pa-ats.com/cz-cz.

[2] *Browsers* [online]. [cit. 2021-04-15]. Dostupné z: https://www.selenium.dev/documentation/en/getting_started_with_webdriver/browsers/.

[3] *Configure unit tests by using a .runsettings file* [online]. Microsoft [cit. 2021-05-02]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/test/configure-unit-tests-by-using-a-dot-runsettings-file?view=vs-2019.

[4] *Introduction to the DOM.* [cit. 2021-04-16]. Dostupné z: https://developer.mozilla.org/en-US/docs/Web/API/Document_Object_Model/Introduction.

[5] Kurz: Vytvoření knihovny tříd .NET pomocí sady Visual Studio. [online]. Microsoft. [cit. 2021-04-26]. Dostupné z: Kurz:Vytvořeníknihovnytříd.NETpomocísadyVisualStudio.

[6] Kurz: Vytvoření konzolové aplikace .NET pomocí sady Visual Studio. [online]. Microsoft. [cit. 2021-04-26]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/core/tutorials/with-visual-studio.

[7] Language binding. [online]. [cit. 2021-04-26]. Dostupné z: https://en.wikipedia.org/wiki/Language_binding.

[8] *Microsoft.VisualStudio.TestTools.UnitTesting Namespace* [online]. Microsoft [cit. 2021-05-04]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting?view=mstest-net-1.3.2.

[9] Most Complete MSTest Unit Testing Framework Cheat Sheet. [online]. AutomateThePlanet. [cit. 2021-04-26]. Dostupné z: https://www.automatetheplanet.com/mstest-cheat-sheet/.

[10] *SikuliX by RaiMan's* [online]. [cit. 2021-04-14]. Dostupné z: http://sikulix.com/.

[11] *SQL Server Agent* [online]. Microsoft [cit. 2021-05-05]. Dostupné z: https://docs.microsoft.com/en-us/sql/ssms/agent/sql-server-agent?view=sql-server-ver15.

[12] *TestContext Class* [online]. Microsoft [cit. 2021-05-01]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting.testcontext?view=mstest-net-1.3.2.

[13] *TestContext.Properties Property* [online]. Microsoft [cit. 2021-05-01]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualstudio.testtools.unittesting.testcontext.properties?view=mstest-net-1.3.2.

[14] *Testovací strategie – test plán* [online]. [cit. 2021-05-04]. Dostupné z: http://testovanisoftwaru.cz/dokumentace-v-testovani/test-plan/.

[15] *WebDriver - W3C Recommendation 05 June 2018*. 2018 [cit. 2021-04-17]. Dostupné z: https://www.w3.org/TR/webdriver1.

[16] *MES systém (Manufacturing Execution System)* [online]. MES center, 2021 [cit. 2021-03-08]. Dostupné z: http://www.mescenter.org/cz/clanky/5-co-je-to-mes-system.

[17] AMMANN, P. a OFFUTT, J. *Introduction to software testing*. 32 Avenue of the Americas, New York, NY 10013-2473, USA: Cambridge University Press, First published 2008 [cit. 2021-03-07]. ISBN 978-0-521-88038-I.

[18] AVRAM, L. Mobile Automation Made Easy with Robot Framework and Appium. [online]. Dostupné z: https://www.pentalog.com/blog/mobile-automation-with-robot-framework-and-appium.

[19] BURNS, D. a HARSHA, S. *Locating elements*. Mar 2021 [cit. 2021-04-17]. Dostupné z: https://www.selenium.dev/documentation/en/webdriver/locating_elements/.

[20] DEVELOPERS, S. Selenium 2.0: Out Now! July 2011, [cit. 2021-04-17]. Dostupné z: https://seleniumhq.wordpress.com/2011/07/08/selenium-2-0/.

[21] DUA, A. What is JSON wire protocol in selenium? [online]. 2019, [cit. 2021-04-14]. Dostupné z: https://www.tutorialspoint.com/what-is-json-wire-protocol-in-selenium.

[22] DYKSTRA, T. Testing in .NET. [online]. Microsoft. [cit. 2021-04-26]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/core/testing/.

[23] HAMBLING, B., MORGAN, P. a SOCIETY, B. C. *Software testing : an ISTQB-ISEB foundation guide*. 2nd ed. London : British Computer Society, 2010 [cit. 2021-03-07]. ISBN 9781906124762 (pbk.).

[24] HERNANDEZ MAYNEZ, E. WinAppDriver and Desktop UI Test Automation. [online]. Microsoft Testing Team. [cit. 2021-04-16]. Dostupné z: https://techcommunity.microsoft.com/t5/testingspot-blog/winappdriver-and-desktop-ui-test-automation/ba-p/1124543.

[25] *ISO/IEC 25010:2011 Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models* [online]. Březen 2011 [cit. 2021-04-09]. Dostupné z: https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?start=3.

[26] KERŠLÁGER, M. *Grafické uživatelské rozhraní* [online]. [cit. 2021-04-09]. Dostupné z: https://en.wikipedia.org/wiki/Graphical_user_interface.

[27] KINSBRUNER, E. What to Expect in the Latest Version of Selenium. [online]. 2020, [cit. 2021-04-14]. Dostupné z: https://www.perfecto.io/blog/selenium-latest-version-selenium-releases.

[28] LIPPS, J. [online]. Jul 2017 [cit. 2021-05-02]. Dostupné z: https://github.com/appium/appium-desktop/blob/master/docs/images/screen-inspector.png.

[29] MOLINA, D. *Understanding the components.* 2018 [cit. 2021-04-17]. Dostupné z: https://www.selenium.dev/documentation/en/webdriver/understanding_the_components/.

[30] MOLINA, D. *Quick tour.* 2019 [cit. 2021-04-17]. Dostupné z: https://www.selenium.dev/documentation/en/getting_started/quick/.

[31] MOLINA, D. a KUMAR, M. *The Selenium project and tools.* Oct 2019 [cit. 2021-04-17]. Dostupné z: https://www.selenium.dev/documentation/en/introduction/the_selenium_project_and_tools/.

[32] NYFFENEGGER, R. [online]. [cit. 2021-04-16]. Dostupné z: https://renenyffenegger.ch/notes/Windows/dirs/Program-Files-x86/Microsoft-Visual-Studio/version/edition/Common7/Tools/VsDevCmd_bat.

[33] O. GALITZ, W. *The Essential Guide to User Interface Design: An Introduction to GUI Design Principles and Techniques.* 3rd. Wiley, 2007 [cit. 2021-3-7]. ISBN 9780470053423.

[34] SHAH, M. Appium Tutorial for Complete Beginners. [online]. [cit. 2021-04-16]. Dostupné z: https://dzone.com/articles/appium-tutorial-for-complete-beginners.

[35] SMRČKA, A. *Testování a dynamická analýza - Úvod a pojmy v testování.* [cit. 2021-04-10].

[36] Society, M. Microsoft [cit. 2021-04-16]. Dostupné z: https://github.com/microsoft/WinAppDriver.

[37] SOJČÁK, J. *Generátor testovacích běhů nad GUI* [online]. [cit. 2021-03-09]. Diplomová práce. Dostupné z: https://www.vutbr.cz/studenti/zav-prace/detail/121953?zp_id=121953.

[38] TerryGLee a olprod. Devenv - přepínače příkazového řádku. [online]. 2018, [cit. 2021-04-17]. Dostupné z: https://docs.microsoft.com/cs-cz/visualstudio/ide/reference/devenv-command-line-switches?view=vs-2019.

[39] VIRANI, M. [online]. [cit. 2021-04-16]. Dostupné z: https://appium.io/docs/en/about-appium/intro/.

[40] ZHANG, T. *Process.StandardOutput Property* [online]. Microsoft, Mar 2017 [cit. 2021-04-30]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.process.standardoutput?redirectedfrom=MSDN&view=net-5.0#System_Diagnostics_Process_StandardOutput.

Příloha A

Obsah paměťového média

- Zdrojové kódy nástroje AuTeR
- Zdrojové kódy frameworku AuTeReporter
- README k aplikaci AuTeR
- README k frameworku AuTeReporter
- Ukázkový projekt s testy na aplikaci Kalkulačka
- Ukázkový report
- Projekt s testy AuTeRTests
- Projekt s testy AuTeReportTests