

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

**APLIKOVANIE NÁSTROJOV AUTOMATICKÉHO
TESTOVANIA DESKTOPOVÝCH APLIKÁCIÍ S GRAFICKÝM
ROZHRANÍM**

Diplomová práca

2020

Bc. Matúš Kokinda

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

**APLIKOVANIE NÁSTROJOV AUTOMATICKÉHO
TESTOVANIA DESKTOPOVÝCH APLIKÁCIÍ S GRAFICKÝM
ROZHRANÍM**

Diplomová práca

Študijný program: Informatika
Študijný odbor: 9.2.1 Informatika
Školiace pracovisko: Katedra počítačov a informatiky
Školiteľ: doc. Ing. Zdeněk Havlice, CSc.
Konzultant: Ing. Ján Trembulák
Ing. Iveta Kindernayová

2020 Košice

Bc. Matúš Kokinda

Abstrakt

Objektom skúmania tejto diplomovej práce je problematika automatického testovania desktopových aplikácií s grafickým používateľským rozhraním. Dôležitou súčasťou práce je analýza súčasného procesu testovania v spoločnosti R-SYS s.r.o. a zhodnotenie možností automatizácie manuálnych testov prostredníctvom voľne dostupných automatizačných nástrojov. Vzhľadom na zložitosť vyvíjaných systémov, bol vypracovaný návrh metodiky automatického testovania, odlišujúcej sa od štandardných postupov aplikovaním automatizačných nástrojov, využívajúcich počítačové videnie. Metodika bola experimentálne overená a vyhodnotená v procese testovania aplikácie C2D, ktorá je súčasťou pasívneho sledovacieho systému VERA-NG. Výsledkom riešenia je definovaný metodický postup automatického testovania, ktorý je možné vo všeobecnosti aplikovať na desktopové aplikácie s grafickým používateľským rozhraním. Zavedením metodiky do praxe je možné efektívne pristupovať k automatizácii testovania, a to nezávisle od zdrojového kódu testového systému a bez navýšených finančných nákladov na projekt.

Kľúčové slová

GUI, automatické regresné testovanie, desktopové aplikácie, testovacie nástroje, metodika

Abstract

The object of research of this thesis is the test automation issue of desktop applications with graphical user interface. An important part of this thesis is the analysis of the current testing process for R-SYS Ltd. company and evaluation of manual test automation options through open source automation tools. Due to the complexity of the developed systems, a proposal of methodology for automatic testing was developed, which differs from standard procedures by the application of automation tools using computer vision. The methodology was experimentally verified and evaluated in the process of C2D application testing, which is part of the VERA-NG passive tracking system. The result of the solution is a defined method of automatic testing that can be generally applied to desktop applications with a graphical user interface. By applying the methodology in practice, it is possible to effectively approach testing automation, independently of the source code of the system under test and without increasing project costs.

Keywords

GUI, automated regression testing, desktop applications, testing tools, methodology

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY
Katedra počítačov a informatiky

ZADANIE DIPLOMOVEJ PRÁCE

Študijný odbor: **Informatika**

Študijný program: **Informatika**

Názov práce:

Aplikovanie nástrojov automatického testovania desktopových aplikácií s grafickým rozhraním
Applying Automated Testing Tools for Desktop Applications with Graphical Interface

Študent: **Bc. Matúš Kokinda**
Školiteľ: **doc. Ing. Zdeněk Havlice, CSc.**
Školiace pracovisko: **Katedra počítačov a informatiky**
Konzultant práce: **Ing. Ján Trembulák, Ing. Iveta Kindernayová**
Pracovisko konzultanta: **R-SYS, Bielerkerovská 29, 040 22 Košice**

Pokyny na vypracovanie diplomovej práce:

1. Analyzovať aktuálny stav v oblasti automatizácie vybraných testovacích postupov a scenárov.
2. Definovať model životného cyklu vývoja aplikácie najmä z pohľadu regresného testovania a použitia metódik testovania pre validáciu aplikácie.
3. Navrhnuť, implementovať a integrovať nástroje a mechanizmy pre automatizovanie vybraných testovacích postupov a scenárov.
4. Demonštrovať využitie navrhnutého a implementovaného riešenia a porovnať ho s inými dostupnými riešeniami.
5. Vypracovať dokumentáciu podľa pokynov vedúceho práce a konzultantov (hlavná časť v rozsahu 50-70 strán a prílohy, tlačенá forma v nerozoberateľnej väzbe a e-forma na DVD).

Jazyk, v ktorom sa práca vypracuje: slovenský
Termín pre odovzdanie práce: 04.05.2020
Dátum zadania diplomovej práce: 31.10.2019



A handwritten signature in blue ink, appearing to be "L. Vokorokos".

prof. Ing. Liberios Vokorokos, PhD.

dekan fakulty

Čestné vyhlásenie

Vyhlasujem, že som celú diplomovú prácu vypracoval samostatne s použitím uvedenej odbornej literatúry.

Košice, 05. mája 2020

.....

vlastnoručný podpis

PodĎakovanie

Touto cestou chcem vysloviť poĎakovanie predovšetkým svojej rodine a najbliĎším priateľom za trpezlivosť a podporu počas celého štúdia. Ďalej by som rád vyjadril poĎakovanie doc. Ing. Zdeňkovi Havlicemu, CSc. za odbornú pomoc, cenné rady a pripomienky pri vypracovaní diplomovej práce. Svoje poĎakovanie by som tiež rád venoval spoločnosti R-SYS s.r.o., predovšetkým Ing. Ivete Kindernayovej za odborné konzultácie a interné materiály na účely vypracovania diplomovej práce, a taktieĎ Ing. Jánovi Trembulákovi za poskytnutie príležitosti realizovať výskum v oblasti automatického testovania.

Obsah

Zoznam obrázkov	9
Zoznam tabuliek	10
Zoznam symbolov a skratiek	11
Úvod	12
1. Analýza problematiky automatického testovania softvéru	13
1.1. Definovanie základných pojmov	13
1.1.1. Procesné modely testovania softvéru	15
1.2. Proces testovania softvéru	17
1.3. Vhodnosť automatizácie testov	19
2. Vývoj softvéru z pohľadu regresného testovania	21
2.1. Regresné testovanie	21
2.2. Automatizácia regresného testovania	22
2.3. Techniky používané v automatickom testovaní	23
2.3.1. Základné princípy automatického ovládania GUI	26
2.4. Aktuálny stav testovania desktopových aplikácií	28
2.5. Zhodnotenie možností integrácie nástrojov a mechanizmov pre automatizovanie testovacích postupov a scenárov	30
2.5.1. Squish for Windows	30
2.5.2. TestComplete	31
2.5.3. Autolt a AutoHotKey	31
3. Návrh metodiky efektívneho aplikovania a integrácie nástrojov automatického testovania desktopových aplikácií s GUI	33
3.1. Návrh integrácie testovacích nástrojov a mechanizmov	33
3.1.1. SikuliX	34
3.1.2. JMeter	35
3.2. Model integrácie vybraných testovacích nástrojov a mechanizmov	37
3.3. Proces automatického testovania VSBT	40

3.3.1.	Analýza rizík a cieľov pri nasadení automatického testovania.....	41
3.3.2.	Príprava testovacieho prostredia.....	41
3.3.3.	Prepísanie testovacích prípadov do testovacích skriptov	42
3.3.4.	Generovanie reportov.....	47
3.3.5.	Údržba automatických testov v postprojektových fázach	48
4.	Experimentálne overenie metodiky pri automatizácii regresného testovania aplikácie C2D .	50
4.1.	C2D	50
4.2.	Konfigurácia C2D a simulačných nástrojov	51
4.3.	Implementácia a exekúcia testov.....	52
4.4.	Vyhodnotenie výsledkov automatických testov	55
5.	Vyhodnotenie výsledkov metodiky	57
	Záver.....	60
	Zoznam použitej literatúry	61
	Prílohy	66

Zoznam obrázkov

Obr. 1 V-model [26].....	15
Obr. 2 W-model [26]	16
Obr. 3 Základný proces testami riadeného vývoja [28]	17
Obr. 4 Základný testovací proces [9].....	18
Obr. 5 Použité testovacie nástroje v procese manuálneho testovania	28
Obr. 6 Model integrácie testovacích nástrojov a mechanizmov.....	33
Obr. 7 Vizualne skriptovanie a rozpoznávanie obrázkov	34
Obr. 8 Architektúra nástroja SikuliX [45].....	35
Obr. 9 Príklad spustenia SikuliX skriptu v nástroji JMeter	36
Obr. 10 Model integrácie vybraných automatizačných nástrojov a mechanizmov.....	37
Obr. 11 <i>HTTP Request Sampler</i> pre vypnutie súradnicovej mriežky v mapovom poli	38
Obr. 12 Procesný model testovania s využitím navrhovanej metodiky	41
Obr. 13 Štruktúra automatického testu ERA2U_3 v nástroji JMeter	44
Obr. 14 Nastavenie licencie pre test ERA2U_3	45
Obr. 15 Vygenerovanie obrázkov pri výskyte chyby	46
Obr. 16 Ukážka zobrazenia základného GUI aplikácie C2D.....	50
Obr. 17 Dialógové okno pri výskyte chyby	55
Obr. 18 Aktuálny a očakávaný stav AIS cieľa	55
Obr. 19 Nesprávne zobrazenie zakresleného ArcBy3Points objektu	56
Obr. 20 Návravnosť investície do automatizácie (SUT sa nemení).....	58
Obr. 21 Návravnosť investície do automatizácie (SUT sa mení).....	58
Obr. 22 Priemerná doba vykonania testu pred a po automatizácii	59

Zoznam tabuliek

Tab. 1 Zdrojová tabuľka pre testy riadené dátami [22]	25
Tab. 2 Príklad tabuľky pre testovanie kľúčovými slovami	26
Tab. 3 Požiadavky na systém automatického testovania.....	33
Tab. 4 Tabuľa HTTP požiadaviek pre testovací interface aplikácie C2D [31]	40
Tab. 5 Príklad testovacieho prípadu v nástroji TestLink.....	43
Tab. 6 Príklad časti vygenerovaného testovacieho reportu.....	48
Tab. 7 Príklad tabuľky pre evidenciu zmien v testoch.....	49
Tab. 8 Príkladná štruktúra testu ERA2U_519_STEP3 – Filtre a profily.....	53
Tab. 9 Výhody a obmedzenia automatického testovacieho systému.....	57

Zoznam symbolov a skratiek

AIS	Automatic Identification System (automatický identifikačný systém)
API	Application programming interface (rozhranie pre programovanie aplikácií)
CSV	Comma-Separated Values (hodnoty oddelené čiarkami)
FTP	File Transfer Protokol (protokol prenosu súborov)
GPL	General Public Licence (všeobecná verejná licencia)
GUI	Graphical User Interface (grafické používateľské rozhranie)
HTTP	HyperText Transfer Protocol (hypertextový prenosový protokol)
HTTPS	HyperText Transfer Protocol Secure (zabezpečený hypertextový prenosový protokol)
IDE	Integrated Development Environment (integrované vývojové prostredie)
IFR	Instrument Flight Rules (pravidlá letu podľa prístrojov)
IP	Internet Protocol (internetový protokol)
MBT	Model Based Testing (testovanie založené na modeloch)
REST	Representational State Transfer (reprezentačný prenos stavu)
SDLC	Systems Development Life Cycle (životný cyklus vývoja systému)
SUT	System Under Test (testovaný systém)
SVFR	Special Visual Flight Rules (špeciálne pravidlá letu za viditeľnosti)
SLC	System Life Cycle (životný cyklus systému)
TDD	Test-Driven Development (testami riadený vývoj)
UFE	Unified Format ERA (jednotný formát ERA)
URI	Uniform Resource Identifier (jednotný identifikátor prostriedku)
VFR	Visual Flight Rules (pravidlá letu za viditeľnosti)
VPN	Virtual Private Network (virtuálna privátna sieť)
VSBT	Visual Script Based Testing (testovanie založené na vizualizačných skriptoch)

Úvod

Nevyhnutnosťou vývoja každého softvéru je jeho dôkladné testovanie, ktoré si dáva za cieľ detailne overiť funkčnosť, výkonnosť, kompatibilitu a bezpečnosť navrhnutých riešení. Testovanie vo všeobecnosti zvyšuje dôveru používateľov v kvalitu systému, čo platí aj pre širokú škálu softvérových produktov s rôznym používateľským rozhraním [2]. Súčasný trend vývoja softvérových aplikácií kladie dôraz na intuitívne ovládanie systému, prostredníctvom grafického používateľského rozhrania (GUI, *Graphical User Interface* angl.). Zložitosť GUI, ako aj rôznorodosť použitých technológií pre jeho zobrazenie, môže pri automatizácii testovania predstavovať potenciálne časové a technické riziko. Zvlášť je problém citeľný v automatickom ovládaní GUI desktopových aplikácií. Súčasnne voľne dostupné testovacie nástroje sú buď vysoko závislé od použitej GUI technológie, alebo nepodporujú všetky GUI komponenty použité v testovanom systéme.

Cieľom tejto diplomovej práce je navrhnúť vhodnú metodiku automatického testovania, ktorú by bolo možné vo všeobecnosti aplikovať na desktopové aplikácie s grafickým používateľským rozhraním. Metodika by mala popisovať integráciu viacerých voľne dostupných automatizačných nástrojov, ktoré majú ako celok vytvárať komplexný testovací systém. V rámci teoretickej časti je analyzovaná problematika automatizácie testovania najmä z pohľadu regresného testovania a existujúcich techník automatického ovládania GUI. Následne je zhodnotený aktuálny stav testovania v spoločnosti R-SYS s.r.o. a analyzované existujúce nástroje na trhu, v oblasti testovania desktopových aplikácií. Praktická časť obsahuje samotný návrh metodiky a jej aplikovanie pri automatizácii testovania aplikácie C2D [39], ktorá je určená pre zobrazenie vzdušnej situácie z radarových senzorov, nad mapovým podkladom.

Výsledkom práce je ucelená metodika automatického testovania, ktorú je možné v budúcnosti aplikovať v rámci vývoja a údržby softvérových produktov spoločnosti R-SYS s.r.o. K výsledku práce taktiež patria sady príkladných automatických testovacích skriptov, ktoré môžu byť zahrnuté do testovacieho repozitára projektu C2D.

1. Analýza problematiky automatického testovania softvéru

Pre mnohé softvérové projekty sa automatizácia testov stáva dôležitou súčasťou vývojového procesu. Hlavným cieľom tohto prístupu je zjednodušenie úsilia vynaloženého na testovanie prostredníctvom testovacích nástrojov, ktoré realizujú vopred napísané testovacie skripty nad testovaným systémom. Aplikovaním týchto nástrojov možno efektívnym spôsobom generovať a simulovať testovacie dáta, ovládať používateľské rozhranie alebo vyhodnocovať a porovnávať výsledky testov.

1.1. Definovanie základných pojmov

Testovanie predstavuje súbor procesov, prostredníctvom ktorých sú analyzované časti testovaného softvéru, za účelom odhalenia rozdielov medzi existujúcimi a požadovanými podmienkami [24]. V rámci analýzy sú zistené informácie o kvalite testovaného softvéru alebo jeho komponentov v rôznych fázach životného cyklu vývoja.

S testovaním sú úzko späté pojmy validácia a verifikácia. Oba pojmy v terminológii testovania softvéru znamenajú niečo úplne odlišné a je potrebné medzi nimi vnímať rozdiel. **Verifikáciu** je možné chápať ako proces, ktorého cieľom je overenie správnosti systému vzhľadom k formulovaným požiadavkám [25]. V procese verifikácie zisťujeme, či je systém vytvorený správne, teda či sú v návrhu zakomponované riešenia všetkých problémov vyplývajúcich z požiadaviek. **Validácia** naopak predstavuje proces, ktorého cieľom je overenie správnosti systému vzhľadom k reálnym požiadavkám. V procese validácie zisťujeme, či je systém správny, teda či systém vykonáva to, na čo bol určený.

Automatické testovanie je definované ako časť testovacieho procesu alebo celý proces, ktorý je vykonávaný bez priameho pôsobenia človeka, a to pomocou špecializovaného softvéru [1]. Existuje viacero programov, ktorých úlohou je samostatne vykonávať niektorú z činností testovacieho procesu. Medzi najznámejšie programy patria tie, ktoré dokážu nahradiť manuálne klikanie testera v GUI testovaného systému. Existujú taktiež programy, ktoré sú určené napríklad na automatizáciu inštalácie a konfigurácie testovaného systému. Ďalšie programy sú schopné vzájomne porovnávať výsledky opakovaných testov a informovať o prípadných rozdieloch. Medzi programy podporujúce automatické testovanie patria aj tie nástroje, ktoré dokážu generovať a vykonávať testovacie prípady z analýzy a návrhu systému. Príkladom toho je modelom riadené testovanie (MBT, *Model Based Testing* angl.), použité napríklad v publikovanej metodike [23] využitia modelov a prototypov pri testovaní softvéru.

Vo všeobecnosti sú **automatizované testovacie nástroje** programy, ktoré umožňujú používať vysoko úrovňový jazyk alebo kľúčové slová pre písanie testovacích skriptov. Okrem toho zabezpečujú riadenie vykonávania testovacích skriptov a spôsob, akým prebieha komunikácia s testovaným systémom. Na základe skutočností, čo všetko dokážu pokryť jednotlivé špecializované testovacie nástroje, je možné podľa zdroja [1] konštatovať, že vhodná kombinácia testovacích nástrojov umožňuje automatizovať takmer celý proces testovania systému.

O **testovacom skripte** hovoríme ako o postupnosti inštrukcií, ktoré je potrebné automaticky vykonať v testovanom systéme. Definuje akcie a kritériá, podľa ktorých testovací nástroj vyhodnocuje jeho úspešnosť alebo zlyhanie. Testovacie skripty sú často priamou súčasťou testovacích prípadov.

Testovací prípad (*Test Case* angl.) popisuje konkrétne kroky vykonávané v určitom testovanom softvérovom komponente a ich očakávané výsledky. Môže obsahovať odkazy na externé zdroje údajov, testovacie skripty alebo konfiguračné súbory. Súbor viacerých testovacích prípadov určených k testovaniu určitého softvérového komponentu tvorí **testovaciu sadu** [2].

Rámec automatického testovania (*Test Automation Framework* angl.) stanovuje pravidlá a postupy pre automatizáciu testovania konkrétneho softvérového produktu, s cieľom dosiahnuť čo najväčšiu efektivitu pri opakovanom použití [3]. Tento rámec integruje viaceré funkčné knižnice, zdrojové testovacie dáta a rôzne opakovane použiteľné moduly. Následne tieto komponenty fungujú ako malé stavebné bloky, ktoré je potrebné zostaviť na mieru konkrétnemu testovanému systému.

Testovanie môže prebiehať v rôznych fázach životného cyklu vývoja softvéru. Vo fáze špecifikácie používateľských požiadaviek a návrhu systému sa uplatňuje takzvané **testovanie na úrovni modelov** (MBT, *Model Based Testing* angl.). Pri tomto testovaní sú využívané existujúce modely a prototypy vyvíjaného systému, prostredníctvom ktorých prebieha overovanie navrhovaných riešení problémov vyplývajúcich z požiadaviek [23]. Overovanie môže prebiehať ešte pred samotnou implementáciou, a to pomocou prototypov a modelov.

Testovanie na úrovni implementačného kódu je uplatňované vo fázach realizácie, testovania a údržby, kde sú testovacie prípady vykonávané nad testovaným systémom (SUT, *System Under Test* angl.), ktorý je implementovaný v špecifickom programovacom jazyku [8]. Tieto dva prístupy sa môžu vzájomne prekrývať a to prostredníctvom metodík, ktoré umožňujú generovanie a vykonávanie testovacích prípadov prostredníctvom modelov a prototypov vyvíjaného systému.

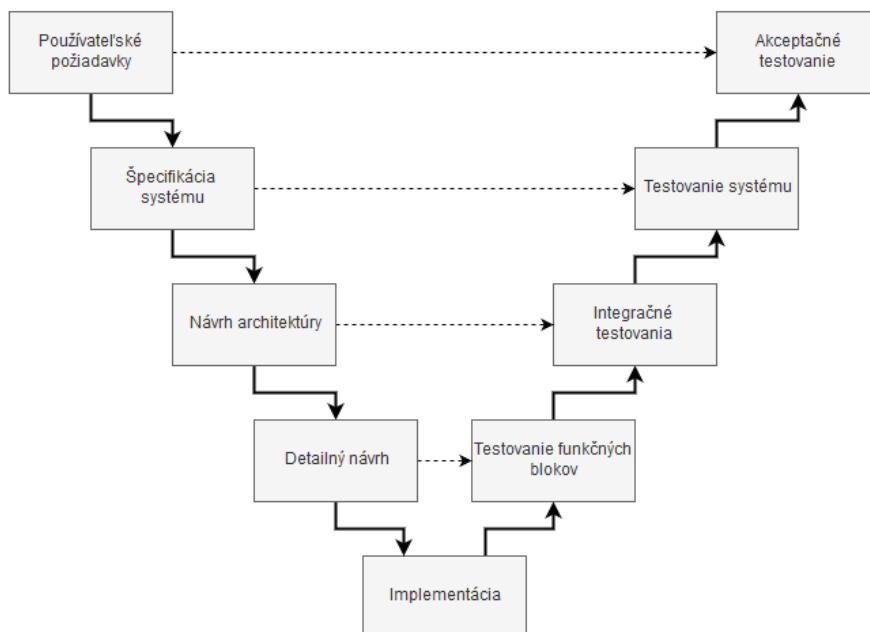
V závislosti od toho, ktoré časti testovaného systému majú byť overené, rozdeľujeme testy na progresné a regresné. **Progresné testy** (*Progression Tests* angl.) overujú prvotnú funkčnosť nových implementovaných funkcií alebo vlastností systému. Opakom toho sú **regresné testy** (*Regression Tests* angl.). Ako naznačuje názov (regres - návrat k predošlému stavu), využívajú sa pri spätnom testovaní systému z dôvodu overenia, či vykonané zmeny alebo implementácia nových vlastností v systéme nenarušila správnu funkčnosť existujúcich častí systému [13]. Regresné testy sú súčasťou **regresného testovania** (*Regression Testing* angl.), ktoré je bližšie špecifikované v kapitole 2.1.

1.1.1. Procesné modely testovania softvéru

V súčasnosti existuje viacero modelov, ktoré definujú aktivity potrebné počas vývoja softvérového produktu a zároveň definujú aj postup ich vykonávania. Medzi najznámejšie patria napr. vodopádový alebo špirálový model [8]. Aplikovanie takéhoto modelu do životného cyklu vývoja systému (SDLC, *Systems Development Life Cycle* angl.) zefektívňuje celý proces, a to najmä z hľadiska riadenia a plánovania zdrojov. Tieto modely však nedefinujú, akým spôsobom majú byť jednotlivé aktivity vykonávané, a preto môžu byť použité pri rôznych prístupoch k vývoju softvéru.

Popri modeloch, ktoré všeobecne popisujú proces vývoja softvéru existujú aj modely, ktoré sa venujú procesu testovania softvéru. Medzi najviac používané patria [7]:

- **V-model** jednoduchým a zrozumiteľným spôsobom zobrazuje súvislosti medzi testovacími aktivitami na jednej strane a vývojovými aktivitami na strane druhej [27]. Ako je zrejme z obrázka 1, tvar písmena „V“, ktorý model zobrazuje, je tvorený ľavou a pravou vetvou.

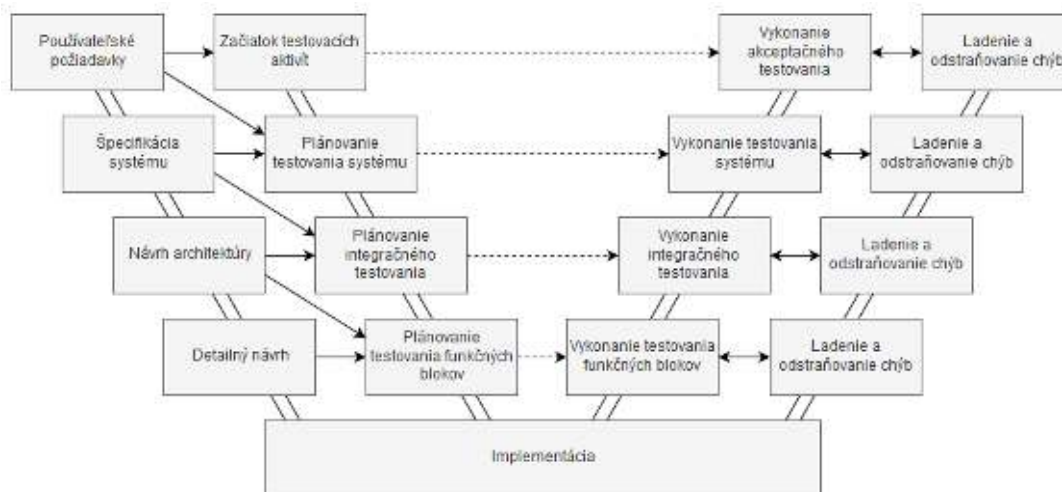


Obr. 1 V-model [26]

Ľavá vetva modelu obsahuje vývojové aktivity s chronologickou postupnosťou procesu vývoja softvéru zhora nadol. Pravú vetvu tvoria testovacie aktivity, ktoré majú chronologickú postupnosť testovacieho procesu zdola nahor. Šípky smerujúce z ľavej vetvy do pravej zobrazujú, ktorá vývojová aktivita je základom danej testovacej aktivity.

Proces vývoja softvéru začína definovaním používateľských požiadaviek v ľavej vetve. Požiadavky sa neskôr premenia do špecifikácie systému, na základne ktorej sa vykoná návrh architektúry [27]. Ďalej nasleduje vytvorenie detailného návrhu a implementovanie systému. Z modelu vyplýva, že vyvíjaný systém nestačí iba správne navrhnuť a implementovať, ale v určitých časových intervaloch je potrebné zaradiť testovanie ako nástroj pre jeho overenie [7]. Tieto časové intervaly sú inak označované ako úrovne testovania (*Test Levels* angl.).

- **W-model** je založený na rozšírení predchádzajúceho V-modelu. Podľa Spillnera [26] je V-model nepostačujúci z dvoch dôvodov. Prvý je ten, že testovacia aktivita je na určitej úrovni spustená až po začatí vývojovej aktivity na danej úrovni, resp. súčasne s ňou. Druhým dôvodom je skutočnosť, že V-model bližšie nešpecifikuje jednotlivé fázy testovacieho procesu na konkrétnych úrovniach modelu. Na základe toho bol vytvorený rozšírený W-model, ktorého obsahom sú aj aktivity, ktoré so samotným testovaním úzko súvisia.



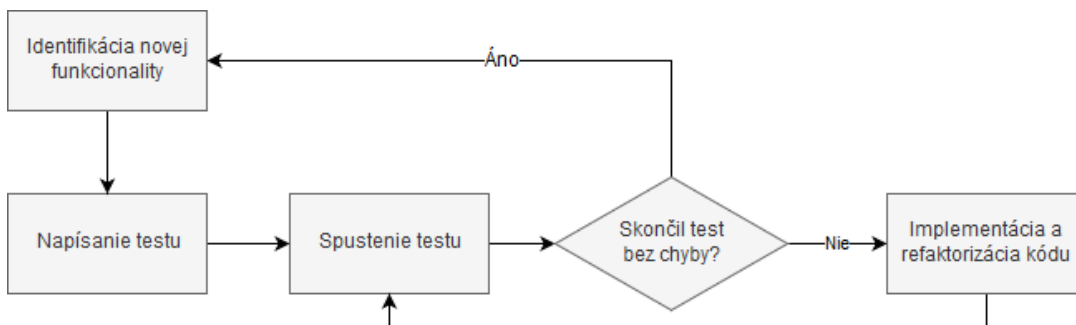
Obr. 2 W-model [26]

Medzi tieto aktivity patrí plánovanie testovania, vykonávanie testovania a následný proces ladenia a odstraňovania chýb [7]. Ako znázorňuje obrázok 2, W-model je zložený z dvoch vetiev na pravej a ľavej strane. Vo vonkajšej ľavej vetve sú znázornené vývojové aktivity softvéru rovnako ako pri V-modeli. Vedľa nej sa nachádza vetva s konštruktívnymi aktivitami testovania, čo zahrňuje plánovanie, tvorbu a návrh testovacích prípadov. Vzájomné závislosti medzi vývojovými a testovacími aktivitami sú znázornené šípkami.

Pravá vnútorná vetva obsahuje deštruktívne testovacie aktivity, teda činnosti na vykonávanie testov. Vonkajšia vetva zobrazuje aktivity zamerané na ladenie a odstraňovanie chýb v testovanom softvéri. Úzko spätá väzba medzi aktivitami v oboch vetvách je znázornená obojstrannými šípkami.

Výhoda oproti V-modelu spočíva v oddelení plánovania testovacích aktivít od ich vykonávania [7]. Na druhej strane, menšou nevýhodou je bližšie nešpecifikovaná spolupráca pri ladení a odstraňovaní chýb s ostatnými aktivitami.

- **Testami riadený vývoj** (TDD, *Test-Driven Development* angl.) - Hoci táto metodika nepopisuje celý životný cyklus vývoja softvéru, je jednou z kľúčových stratégií programovania, ktorá si získava čoraz väčšiu pozornosť. Hlavný princíp tohto prístupu spočíva v písaní automatických testov ešte pred samotným implementovaním kódu, v malých rýchlych iteráciách [29].



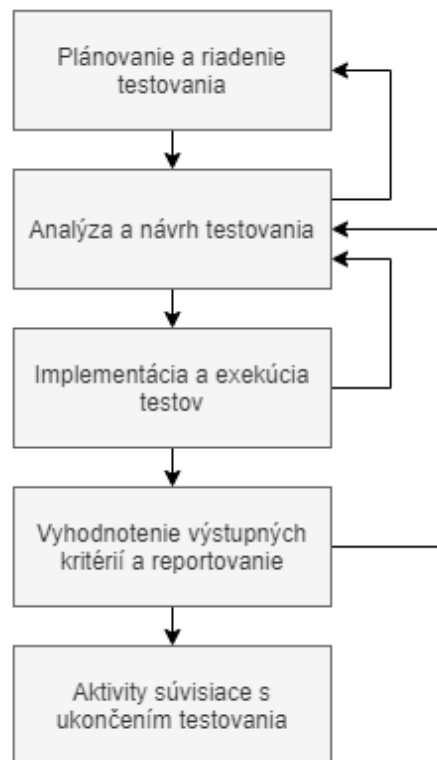
Obr. 3 Základný proces testami riadeného vývoja [28]

V rámci jednej iterácie programátor najskôr vytvorí testy, ktoré overujú požiadavky na funkcionality danej časti kódu. Pri prvom spustení testov sa overuje samotná funkčnosť testov a to tak, že by každý test mal skončiť neúspešne. Po overení funkčnosti je implementovaný kód, ktorý úspešne prejde existujúcimi testami. Pokiaľ to je potrebné, programátor môže optimalizovať svoj kód s ohľadom na efektívnosť [5]. S využitím automatických testov je tak možné overiť, či v procese vývoja nebola zanesená chyba do kódu, alebo nedošlo k zmene požiadaviek na funkčnosť výsledného kódu.

1.2. Proces testovania softvéru

Testovanie je proces aktivít, ktoré na seba sekvenčne nadväzujú, prekrývajú sa a opakujú, s cieľom zvýšiť kvalitu softvéru. Kvalita a efektívnosť testovania softvéru závisí od kvality použitých prístupov a techník [8]. Preto pri vytváraní automatických testov, pre zložitejšie systémy, je potrebné dbať na rôzne aspekty v každej fáze procesu testovania.

V závislosti od toho, či sú automatické testy vytvárané už pre vyvinutý systém, alebo sa s nimi počíta už od začiatku vývoja, záleží spolupráca medzi jednotlivými členmi vývojového tímu. Ak proces testovania počíta s aplikovaním automatizácie od začiatku vývoja systému, je podľa Pagea, Rollisona a Johnstona [1] vhodné, aby sa špecialista pre automatické testovanie podieľal na návrhu architektúry a spôsobe vývoja, s cieľom zabezpečiť dobrú testovateľnosť výsledného systému.



Obr. 4 Základný testovací proces [9]

Webový portál Tryqa.com [10], rozdeľuje aktivity testovania do piatich hlavných krokov. Celkový proces je nazývaný ako „Základný testovací proces“ (obr. 4) pričom začína plánovaním a končí reportovaním chýb a uzatvorením celého testovania.

- **Plánovanie a riadenie testovania** – Proces plánovania určuje ako sa testy budú vykonávať, čo bude testované a ako to dosiahneme. Z pohľadu automatických testov je potrebné rozhodnúť, aké typy automatických testov budú vytvárané a prečo. Taktiež je dôležité zistiť pre aké systémy a procesy budú nasadené automatické testy [8]. Riadením testovania je možné určiť, či testovanie prebieha správne a čo je potreba robiť v prípade, že testovanie neprebieha podľa plánov.
- **Analýza a návrh testovania** – V tejto fáze sú využité informácie z plánovania ako napríklad čo bude testované, alebo ako a kedy sa bude testovať. Tieto informácie sú transformované

do testovacích prípadov, ktoré by mali byť navrhnuté tak, aby pokryli čo najviac požiadaviek. Ak automatizujeme už existujúce manuálne testy, je potrebné určiť, ako budú automatické testy štruktúrované. Rovnako je dôležité vybrať vhodné nástroje, frameworky a určiť, ktoré typy testov automatizovať [8].

- **Implementácia a vykonávanie testov** – Táto fáza pozostáva z vytvorenia a stanovenia priorít testovacích procedúr, nastavenia verzovania testov a synchronizácie verzií testov s verziou testovaného systému. Ďalej zahrňuje vytvorenie testovacích dát, písanie testovacích skriptov, opakované vykonávanie testovacích procedúr a porovnávanie zaznamenaných výsledkov s očakávanými výsledkami testov.
- **Vyhodnotenie výstupných kritérií a reportovanie** – Podmienky, ktoré boli stanovené počas fázy plánovania, sa v tomto bode overujú a zisťuje sa, či boli splnené. Vyhodnocujú sa potrebné dodatočné testy, aktualizácia testovacích skriptov, prípadne sa uvažuje o zmene výstupných testovacích kritérií. Ďalej nasleduje spísanie výsledkov testovacích aktivít vykonaných nad testovaným systémom pre zvýšenie dôvery v kvalitu systému.
- **Aktivity súvisiace s ukončením testovania** – Ide o kontrolné aktivity, ktoré uisťujú, že všetko prebehlo podľa plánov [10]. Medzi tieto aktivity patrí uzatváranie vyriešených problémov, kontrola správnosti dokumentácie, archivácia testov a testovacieho prostredia, vyhodnotenie procesu testovania a jeho optimalizácia pre ďalšie podobné projekty.

1.3. Vhodnosť automatizácie testov

Nie je možné jednoznačne určiť, kedy pristúpiť k automatizovaniu testov. Existuje však všeobecné kritérium, ktoré sa používa pri kritických rozhodnutiach v rámci vývoja softvéru: Musí sa to vyplatiť.

Testovanie má zmysel automatizovať vtedy, keď sa dostatočné množstvo testov v nemennej podobe opakuje dostatočne často [4]. To, aké množstvo testov je dostatočných a aký je dostatočný počet opakovaní, závisí od zložitosti, rozsahu a kritickosti systému. Pri tomto rozhodovaní musí projektový resp. testovací manažér brať do úvahy aj nasledujúce aspekty:

- počet testerov potrebných pre manuálne vykonanie testov,
- rozsiahlosť a detailnosť testov,
- zložitosť testovanej aplikácie,
- finančné náklady na zabezpečenie a údržbu automatizačných testovacích nástrojov,
- možnosť aplikovania metodiky automatizácie aj na iné projekty [5].

Opakované automatické testovanie musí byť vykonané nad nemennou časťou testovaného systému. Automatizované nástroje, na rozdiel od práce testera, nedokážu pružne a rýchlo reagovať na zmeny v systéme. Každá zmena v testovanom systéme si nutne vyžaduje prepracovať testovacie skripty alebo upraviť konfiguráciu testovacieho nástroja. Možnosť automatizácie konfigurovateľnosti testov, podľa realizovaných zmien v testovanom systéme, je značne obmedzená. Preto je vhodné automatizovať testy tej časti systému, nad ktorými už prebehli bežné manuálne testy.

Na základe vyššie uvedených podmienok je možné povedať, že automatizácia je vhodná pre **regresné testovanie**, ktoré vyžaduje opakované vykonávanie testov po každej zavedenej zmene v testovanom systéme. Rovnako tak je vhodná automatizácia pri testoch, ktoré overujú zmeny dát v systéme [4]. Napríklad u viacjazyčných verziách aplikácie.

Automatizácia ma taktiež zmysel pri **nefunkčných testoch**, ktoré spočívajú v overení tých vlastností systému, ktoré priamo nesúvisia s jeho funkciami, ale zároveň sú podstatné pre jeho správne fungovanie. Medzi tieto testy radíme **výkonnostné testovanie**, ktoré overuje správne fungovanie aplikácie pod záťažou. Napríklad simuláciou zvýšeného počtu používateľov systému v rovnakom čase alebo zvýšeným objemom dát. Medzi nefunkčné testy rovnako patria testy správania sa testovaného systému vzhľadom k hardvéru, s ktorým systém pracuje. Napríklad či systém zbytočne nezaťažuje procesor alebo pamäť [1].

Ďalšou vhodnou oblasťou pre automatizáciu sú tzv. **overovacie testy zostavenia** známe pod pojmom (*Smoke testing* angl.), ktoré overujú, či je testovaný systém správne nainštalovaný a nakonfigurovaný ešte pred samotným spustením testov [8]. Cieľom týchto overovacích testov nie je nachádzať chyby, ale kontrolovať pripravenosť systému. V rámci testov sa zvyčajne realizujú iba základné prechody testovaným systémom a overujú sa základné funkčnosti, ktoré sú navyše stále rovnaké [1]. Úspešné vykonanie testov obvykle slúži ako vstupné kritérium pre spustenie ďalších fáz testovania.

2. Vývoj softvéru z pohľadu regresného testovania

Aj malé zmeny vo vyvíjanom softvéri, spôsobené napríklad modifikáciou alebo integráciou s novými komponentami, môžu mať neočakávané dôsledky. Testovanie každej zmeny, ktorá by mohla negatívne vplývať na funkčnosť systému, sa môže zdať ako nákladný a časovo náročný proces. Pre tento účel je vhodné navrhnuť účinnú stratégiu regresného testovania, ktorá v plnej miere pokryje potreby vyvíjaného systému a zaisť kvalitu za optimálnu cenu.

2.1. Regresné testovanie

Regresné testovanie je definované ako typ softvérového testovania, ktoré je vykonávané za účelom overenia, či zmena zdrojového kódu v softvéri nemá vplyv na existujúcu funkčnosť nemodifikovaných častí systému [11]. Toto testovanie zabezpečuje, že systém funguje tak, ako pred implementovaním novej funkčnosti, modifikáciou existujúcej funkčnosti alebo opravou existujúcej chyby.

Regresné testovanie je častokrát zamieňané s pojmom „konfirmačné testovanie“. Rozdiel medzi týmito dvoma spôsobmi testovania spočíva v rozsahu testov, ktoré sú vykonávané po modifikácii testovaného systému.

- **Konfirmačné testovanie** – Inak nazývané ako pretestovanie alebo re-testovanie, je založené na opakovanom vykonaní testovacieho prípadu, ktorý v minulosti skočil chybou [12]. Cieľom tohto testovania je potvrdiť, že chyba, ktorá bola úspešne opravená, sa po opakovanom vykonaní testovacieho prípadu neprejavuje.
- **Regresné testovanie** – Overuje, či sa pri zmenách v zdrojovom kóde, alebo konfigurácii prostredia neprejavili chyby vo funkčnosti, prípadne iných aspektoch vyvíjaného systému, ktoré pred aplikovaním zmien fungovali podľa očakávania [12].

Softvérová chyba, ktorá spôsobí určitý druh nepredvídanej nefunkčnosti pri modifikácii testovaného systému, je nazývaná aj ako regresia softvéru. Existujú tri kategórie regresných chýb softvéru [7]:

- **Lokálne chyby** – Ak sa chyba prejavuje v tom istom softvérovom komponente, ktorý bol modifikovaný.
- **Vzdialené chyby** – Ak sa chyba prejavuje v inom softvérovom komponente ako v tom, ktorý bol modifikovaný. Zvyčajne ide o chyby spôsobené integráciou.
- **Odkryté chyby** – Ak modifikácia softvérového komponentu spôsobí aktiváciu chyby, ktorá sa už v komponente vyskytovala, ale nespôsobovala zlyhanie.

Vyššie spomínané chyby sú spôsobené prevažne zmenami v testovanom softvéri. Zmena je kľúčovým konceptom regresného testovania a dôvody týchto zmien môžeme vo všeobecnosti rozdeliť do štyroch kategórií [13]:

- **Implementovanie novej funkcionality** – Ide o najbežnejší dôvod zmeny softvéru, pri ktorom je vhodné aplikovať regresné testovanie. Po zavedení novej funkcionality musí byť existujúci a novo implementovaný kód plne kompatibilný. Pravdepodobnosť výskytu regresných chýb závisí aj od skúsenosti vývojára s daným systémom. Neznalosť internej štruktúry a možné dôsledky zásahov do kódu môžu produkovať viac regresných chýb.
- **Modifikácia existujúcej funkcionality** – Modifikácia môže v sebe zahŕňať revidovanie, odstránenie alebo úpravu existujúcej funkcionality. Regresné testovanie v týchto prípadoch kontroluje, či bola príslušná funkcionality odstránená alebo upravená, bez narušenia zvyšnej funkčnosti. Aby sa čo najmenej zamedzilo výskytu regresných chýb, je dôležité dbať na prehľadnosť kódu, nezávislosť komponentov a správne logické štruktúrovanie funkcionality do metód [7].
- **Integrácia** – V tomto prípade regresné testovanie overuje bezchybnú funkčnosť softvéru po integrácii testovaného systému s novými komponentami. Regresné chyby môžu byť taktiež spôsobené nekompatibilitou s aktuálnymi verziami externých knižníc alebo zmenami na úrovni komponentov operačného systému.
- **Oprava chýb** – Snaha vývojára o rýchlu nápravu chyby môže vo výsledku generovať ešte viac regresných chýb [7]. Tieto chyby sa najčastejšie vyskytujú pri nedostatočnej analýze častí softvéru alebo dát, ktoré využívajú opravenú časť kódu.

2.2. Automatizácia regresného testovania

Vo všeobecnosti je najčastejším dôvodom aplikovania automatizácie zvýšenie efektívnosti vykonávania testovacích prípadov. Návrh, vykonanie ako aj analýza testovacích prípadov, patria z pohľadu manuálneho testovania k časovo náročným procesom. Samotné regresné testovanie si vzhľadom na svoju širokú podmnožinu testovacích prípadov vyžaduje dostatočne dlhú dobu, aby boli všetky testy správne vykonané a vyhodnotené.

Nie je pravidlom, že manuálne regresné testovanie je vhodné pre malé projekty a automatické regresné testovanie pre stredné a veľké projekty. Pre projekty akejkoľvek veľkosti môžeme aplikovať obidve metódy [17]. Dôležité je vybrať vhodný pomer manuálneho a automatického testovania tak, aby čo najlepšie vyhovoval potrebám projektu. Ak nie je stanovený správny pomer, môže väčšie množstvo manuálnych alebo automatizovaných regresných testov predstavovať riziká spojené s nedodržaním časového a finančného plánu projektu [18].

Aby sa znížila pravdepodobnosť výskytu ľudskej chyby a čas pri vykonávaní manuálnych regresných testov, je možné automatizovať viaceré činnosti, ktoré súvisia priamo s vykonávaním testu alebo prípravou testovacieho prostredia. Medzi tieto činnosti patrí:

- generovanie vstupných testovacích dát,
- extrahovanie výstupných dát,
- porovnávanie výsledkov,
- ovládanie GUI,
- konfigurácia testovacieho prostredia.

Automatizácia môže byť realizovaná prostredníctvom špecializovaných testovacích nástrojov alebo vlastných implementovaných skriptov. Konzistentné automatické vykonávanie týchto činností má viacero výhod [8]:

- **Nižšie náklady na údržbu systému** – Množstvo prostriedkov, ktoré je potrebné vynaložiť na opravu chýb v systéme, sa zvyšuje úmerne s časom [1]. Hoci si automatizácia vyžaduje vyššie vstupné náklady, z dlhodobého hľadiska vedie častejšie stabilné vykonávanie regresných testov k zvýšeniu pravdepodobnosti objavenia chýb v čo najkratšom čase.
- **Vyššia účinnosť testov** – Platí to najmä pre tie testy alebo sady testov, ktoré pre svoju komplexnosť vyžadujú dlhú interakciu s testovaným systémom. Ak sú tieto testy úplne automatizované, môžu prebiehať počas noci alebo prostredníctvom vzdialeného virtuálneho počítača.
- **Testovanie na rôznych platformách** – Správne navrhnutá architektúra skriptov a použitie vhodných testovacích nástrojov umožňuje spúšťanie testov vo viacerých typoch operačných systémov a webových prehliadačov [7].
- **Vyvinutý testovací framework** – Framework, pomocou ktorého sú riadené procesy a nástroje pre návrh a vykonávanie automatických regresných testov, môže byť použitý pri budúcich projektoch, kde by sa jeho samostatný vývoj nevyplatil.

2.3. Techniky používané v automatickom testovaní

Existuje viacero techník, pomocou ktorých je možné vytvárať nové automatické testy alebo automatizovať existujúce manuálne testy. Vzájomne sa odlišujú v rôznych aspektoch, v závislosti od technických znalostí, jednoduchosti použitia, modularity vygenerovaných skriptov a ich udržateľnosti [8]. Medzi najviac využívané techniky patria:

- **Zaznamenávanie a prehrávanie aktivít (record and playback)** – Jedná sa o najznámejšiu formu automatizácie, ktorá je realizovateľná aj bez väčších technických znalostí. Jej princíp pozostáva z dvoch fáz:

- *Zaznamenávanie* – Testovací nástroj zaznamenáva aktivitu používateľa vykonávajúceho testovací prípad v používateľskom rozhraní testovaného systému. Zaznamenávané sú ako vstupy od používateľa, tak aj reakcie systému.
- *Prehrávanie* – Test je spustený prostredníctvom rovnakého testovacieho nástroja, ktorý ho vykonáva podľa vopred zaznamenaného testovacieho scenára a porovnáva vygenerované premenné s ich očakávanými hodnotami.

Podľa Kenta [19] sa táto technika najviac využíva pri testovaní webových aplikácií, pričom využíva objekty používateľského rozhrania ako napríklad textové polia, tlačidlá, zaškrtávacie polia, rozbaľovacie zoznamy a podobne.

Výhoda tohto prístupu spočíva v jednoduchosti a rýchlej produkcii testovacích skriptov. Nevýhodou je vysoká citlivosť na akékoľvek funkčné či technické zmeny systému. Aj menšia obmena staršieho ovládacieho prvku za novší môže spôsobiť nefunkčnosť testov. Medzi ďalšie nevýhody patrí statický prístup pri zaznamenávaní vstupných hodnôt, nízka modularita a komplikované ošetrenie chybových stavov [20]. Vhodné využitie tejto techniky je v neskorších fázach regresného testovania, kedy dochádza k minimálnym zmenám systému [8].

- **Modifikovanie vygenerovaných skriptov** – Technika vychádza zo zaznamenávania a prehrávania aktivity používateľa, ale vďaka programátorským znalostiam je rozšírená o úpravu a rozširovanie vygenerovaných skriptov [19]. Prostredníctvom zásahov do skriptov je možné eliminovať viaceré obmedzenia, ktoré vychádzajú z predchádzajúcej techniky a dosiahnuť tak lepšiu udržateľnosť a znovupoužiteľnosť jednotlivých testov.

Modifikácia skriptov umožňuje implementovať zložitejšiu logiku testovacích prípadov a rozšíriť pritom ich obsah, ako aj variabilitu využívaných premenných [8]. Existujú prípady, pri ktorých je manuálna úprava zaznamenaných testov nevyhnutná. Napríklad pri spracovaní meteorologických dát z externých súborov je potrebné overiť, či systém pracuje s aktuálnym časom a dátumom. Pri zázname testu bude síce zaznamenaný aktuálny dátum, ale v budúcnosti by tento test nefungoval správne. Bude preto potrebné nahradiť statický zaznamenaný čas a dátum funkciou, ktorá v čase vykonávania testu vráti aktuálnu hodnotu.

- **Testovanie riadené dátami** – Pri automatickom testovaní systému môže byť niekedy potrebné opakovane pretestovať rovnakú funkcionálnosť, s väčším variabilným objemom

vstupných údajov. V takýchto prípadoch nie je vhodné ukladať testovacie údaje do skriptu, ale odporúča sa tieto údaje uchovávať v externej databáze mimo testovacích skriptov [21].

Vstupné údaje		Očakávané výstupy
Viditeľnosť (km)	Oblačnosť (ft)	atribút podmienky
6	1800	VFR
6	900	SVFR
2	900	SVFR
6	400	IFR
2	300	IFR

Tab. 1 Zdrojová tabuľka pre testy riadené dátami [22]

Príklad, ktorý zobrazuje tabuľka č. 1, slúži ako zdroj údajov pre testy riadené dátami k overeniu správneho priradenia atribútov letových podmienok pre správy METAR (pravidelná letecká správa o počasí). Zdrojové údaje môžu byť uložené v databázovej tabuľke alebo v súboroch typu CVS a XLS [22]. Testovací skript údaje načítava riadok po riadku, vykonáva nad nimi potrebné operácie a výsledky porovnáva s očakávanými výstupmi, ktoré sú taktiež dostupné v zdrojovom súbore.

Hlavnou výhodou tejto techniky je nezávislosť testovacích skriptov od testovacích údajov, čo výrazne znižuje celkový počet skriptov potrebných na pokrytie všetkých možných kombinácií testovacích prípadov. Nevýhodou je zložitosť a vysoké vynaložené úsilie pri integrácii s testovanou funkcionalitou, bez zbytočných doplňujúcich medzikrokov [21].

- **Testovanie riadené kľúčovými slovami** – Ide o techniku, ktorá rozširuje dátovo riadené testovanie. Napriek rovnakej štruktúre testov medzi oboma prístupmi, testy riadené kľúčovými slovami neobsahujú v zdrojových tabuľkách len vstupné a výstupné údaje, ale aj príkazy, z ktorých pozostáva testovací skript [8]. Tieto príkazy sú označované ako kľúčové slova, teda sekvencie akcií, ktoré majú byť vykonané.

Číslo kroku	Kľúčové slovo	Parameter_1	Parameter_2	Popis
1	Kliknutie_na_objekt	GUI.Object("drawing")		Zobrazenie kresliacich nástrojov
2	Vykreslenie_objektu	GUI.Drawing("circle")	20	Vykreslenie kružnice s polomerom 20NM

3	Overenie_vykreslenia	GUI.Map(o,o)	Circle_1.jpg	Overenie vykreslenia kružnice
4	Overenie_textu	GUI.Object("circle")	"Oblasť 1"	Overenie názvu kružnice

Tab. 2 Príklad tabuľky pre testovanie kľúčovými slovami

V zdrojovej tabuľke testu, overujúceho správne vykreslenie oblasti v mapovom poli aplikácie, sú kľúčové slová za behu načítavané a spracované, čím sa dynamicky vytvára samotná logika testu. Skript alebo automatizovaný nástroj číta každý riadok zo súboru, kde je definované, aká inštrukcia má byť použitá, nad akými objektami a s akými vstupnými a výstupnými údajmi [21]. Keďže návrh testu je oddelený od nízkoúrovňovej implementácie skriptov, tvorba testovacích prípadov spočíva vo vyskladaní jednotlivých krokov prostredníctvom kľúčových slov a ich parametrov. Táto technika má niekoľko výhod [8]:

- vysoká miera abstrakcie testov,
- odolnosť voči zmenám systému,
- dobrá udržateľnosť testov,
- príprava návrhov testov už v ranných fázach vývoja.

Ako uvádza webový portál Softwaretestinghelp.com [21], medzi nevýhody patrí náročná príprava a taktiež nutnosť zavedenia spoľahlivého podporného frameworku, ktorý môže byť implementovaný samostatne alebo integrovaný prostredníctvom existujúcich testovacích nástrojov, s dostatočným časovým predstihom. Ak totiž nie sú definované všetky potrebné kľúčové slová, nie je ich možné použiť pri návrhu testovacieho prípadu.

Hybridný prístup – Ako už názov napovedá, táto technika je kombináciou viacerých prístupov používaných v automatickom testovaní. V priebehu vývoja softvérového projektu môže nastať situácia, kedy je prvotne zvolená technika nepostačujúca a musí byť doplnená ďalšou [8]. Vtedy je odporúčané kombinovať viaceré techniky s cieľom eliminovať ich slabé stránky a vyťažiť tak z automatizácie maximálny úžitok.

2.3.1. Základné princípy automatického ovládania GUI

Prostredníctvom grafického používateľského rozhrania testovaného systému je testerami realizovaná väčšina manuálnych testov. K rovnakému rozhraniu je preto potrebné pristupovať aj pri vytváraní automatických testov, ktoré simulujú akcie manuálneho testera.

Vzhľadom na rozdielny prístup k používateľskému rozhraniu a použitie rozdielnych technológií, ktoré jednotlivé typy testovaných systémov využívajú, rozlišujeme technologické princípy ovládania GUI [8] pre webové, desktopové a mobilné aplikácie. Nižšie vymenované technologické princípy s príznakom **WEB** značia, že sú použiteľné pre systémy typu klient-server, ktoré majú webové používateľské rozhranie. Do tejto kategórie spadajú aj webové aplikácie optimalizované pre mobilné zariadenia. Príznak **NATIVE** definuje použitie pre desktopové (natívne) aplikácie, ktoré sú určené priamo pre konkrétnu platformu. Rovnako tak majú svoje špecifické označenie aj mobilné aplikácie s príznakom **MOBILE**.

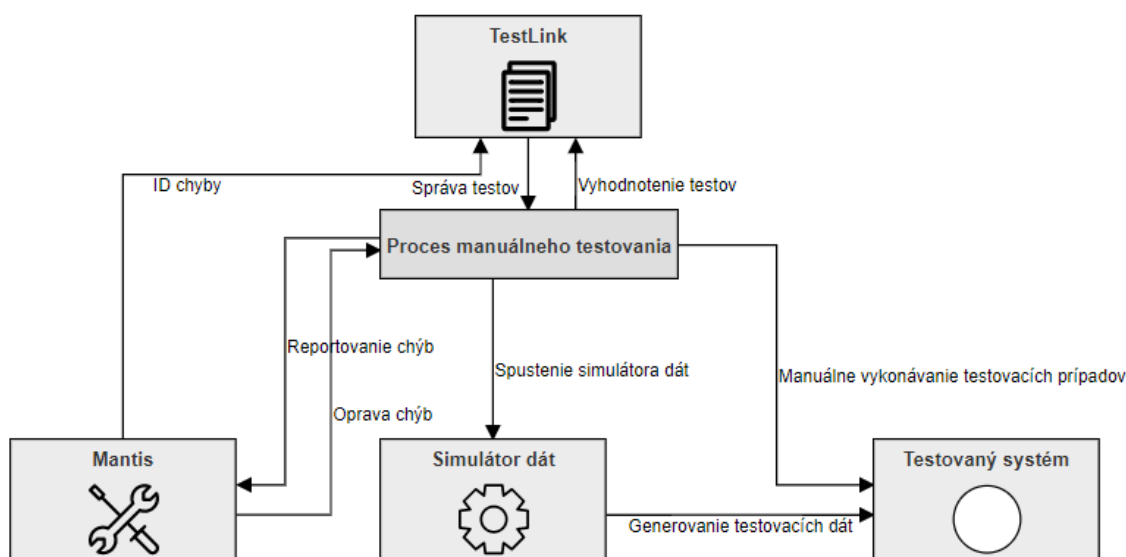
- **Klikací automat (WEB, NATIVE, MOBILE)** – Ide o jeden z najjednoduchších spôsobov ovládania GUI automatickým testom. Pomocou automatizovaného nástroja je nahratá sekvencia posunov a klikov myši pomocou súradníc, kombinované stláčanie klávesov, manipulácia s oknami a ovládacími prvkami. Medzi najznámejší nástroj patrí AutoIt [30]. V súčasnosti sa už tento princíp nevyužíva z dôvodu nižšej spoľahlivosti.
- **Sekvencia HTTP požiadaviek (WEB, NATIVE)** – Využíva sekvencie HTTP (*HyperText Transfer Protocol* angl.) požiadaviek, ktoré sú nahrané na server a následne prehrávané. Tento princíp je vhodnejší pre záťažové testovanie, ale môže byť taktiež použitý pre ovládanie GUI aplikácie, pokiaľ je v testovanom systéme implementované testovacie rozhranie [31] pre ladiace a automatizačné účely. Tento princíp je použitý napríklad vo voľne dostupnom nástroji Apache JMeter [32].
- **Rozhranie pre webový prehliadač (WEB)** – V tomto prípade sa k testovanému systému pripojujeme prostredníctvom rozhrania, vytvoreného pre konkrétny webový prehliadač, pričom sú k dispozícii rôzne podporné nástroje vo forme rozšírení [8]. Ide o veľmi populárny spôsob automatizácie GUI webových aplikácií vzhľadom na jeho rýchlosť a jednoduchosť, bez nutnosti implementácie testovacích skriptov. Tento princíp uplatňuje viacero voľne dostupných webových testovacích nástrojov ako napríklad Katalon Recorder [33] alebo Selenium WebDriver [34].
- **Rozhranie pre prostredie desktopových aplikácií (NATIVE, MOBILE)** – Pomocou rozhrania na úrovni operačného systému je možné pristupovať k objektom používateľského rozhrania testovaného systému. To si vyžaduje špecializované rozhranie pre každú technológiu, v ktorom je desktopová aplikácia implementovaná. V súčasnosti čiastočne túto možnosť ponúka komerčný testovací nástroj TestComplete [35], avšak použitie voľne dostupných nástrojov je v tomto prípade značne obmedzené.
- **Rozpoznanie grafického objektu (WEB, NATIVE, MOBILE)** – Ide o princíp ovládania GUI, ktorý je nezávislý od typu aplikácie a použitých technológií, s ktorými systém pracuje.

Objekt v používateľskom rozhraní testovaného systému je lokalizovaný na základe grafickej reprezentácie. Aktuálna grafická reprezentácia objektu nemusí byť v absolútnej zhode s očakávaným obrazom, čo závisí od nastavenia presnosti rozpoznania objektu podľa vlastných potrieb [8]. Umožňuje jednoduchšie zameranie objektov GUI a vyššiu stabilitu automatických testov pri často sa meniacom testovanom systéme. Princíp založený na optickom rozpoznávaní znakov (OCR, *Optical Character Recognition* angl.) je napríklad súčasťou voľne dostupného nástroja SikuliX [36].

Väčšina vyššie spomenutých testovacích nástrojov využíva techniku zaznamenávania a prehrávania aktivít v GUI testovaného systému. Výsledkom toho sú vygenerované skripty uložené v konkrétnom programovacom jazyku. Alternatívnym riešením k ovládaniu používateľského rozhrania je preto aj **manuálny skriptovací princíp** [8], ktorý vďaka aktívnemu využitiu skriptovacích jazykov (napr. Groovy, Python), umožňuje vytvárať efektívnejšie a stabilnejšie prechody testovaným systémom. Vývojár automatických testov okrem iného rozhoduje, akým spôsobom pristupovať k objektom GUI a aké atribúty majú byť použité pri mapovaní objektov, nad ktorými sú volané akcie používateľa [33]. Skriptovací princíp je zväčša súčasťou testovacích nástrojov vo forme editačného módu.

2.4. Aktuálny stav testovania desktopových aplikácií

V súčasnosti spoločnosť R-SYS s.r.o. uplatňuje pri vývoji svojich desktopových aplikácií s grafickým používateľským rozhraním, prevažne princíp manuálneho vykonávania integračných, systémových a akceptačných testov.



Obr. 5 Použité testovacie nástroje v procese manuálneho testovania

Ako zobrazuje obrázok 5, vo fáze testovania sú v súčasnosti využívané tri základné nástroje, ktoré slúžia na správu testovacích prípadov, evidenciu chýb a simuláciu testovacích dát:

- **Mantis** – Je webový systém [37] so všeobecnou verejnou licenciou (GPL, *General Public Licence* angl.), ktorý slúži na sledovanie a evidenciu chýb pri vývoji softvéru. Prístup do systému je podmienený pripojením sa k virtuálnej privátnej sieti (VPN, *Virtual Private Network* angl.). Z pohľadu konfigurácie stavov, ktorými chyby pri svojom spracovaní prechádzajú, umožňuje Mantis spravovať viaceré činnosti:
 - *Správa jednotlivých stavov* – Umožňuje vytvárať, editovať a rušiť stavy tak, aby mohol byť proces sledovania a evidencie chýb prispôsobený potrebám projektu.
 - *Správa prechodov medzi stavmi* – Poskytuje možnosť konfigurácie vzájomnej nadväznosti jednotlivých stavov na seba.
 - *Správa prístupu používateľských rolí k jednotlivým stavom* – V systéme Mantis sú každému používateľovi pridelené prístupové práva, na základe ktorých je možné vykonávať iba jemu prislúchajúce úkony. Spravidla sú tieto prístupové práva pre jednotlivých členov vývojového tímu stanovené prislúchajúcim projektovým manažérom.
- **TestLink** – Je webovo orientovaný nástroj s GPL licenciou, ktorý je určený pre manažment testovania. Ide teda o nástroj, ktorý obsahuje komponenty pre evidenciu softvérových požiadaviek, ako aj komponenty pre vytváranie, správu a vyhodnocovanie testov [38]. Štruktúra testov je uchovávaná v databáze na firemnom serveri, ktorá pracuje obdobne ako databáza chýb Mantis. Prístup do databázy je realizovaný cez VPN prostredníctvom ľubovoľného webového prehliadača.
 - Súčasťou TestLink-u je taktiež možnosť prepojenia s bugtrackingovými nástrojmi (v našom prípade systém Mantis), pre priamu evidenciu nájdených chýb. Takto je zabezpečená krížová kontrola TestLink-Mantis pre potreby sledovania stavu chýb a pokrytia chýb testovacími prípadmi.
- **Simulátor dát (Prehrávač)** – Simulátor dát predstavuje sadu interných pomocných testovacích nástrojov (utilít), ktoré umožňujú posielanie prehrávaných alebo generovaných testovacích dát na testovaný systém pre ich ďalšie spracovanie. Väčšinou sú vo forme spustiteľných skriptov písaných v jazyku Perl a Python alebo majú podobu komplexnejších GUI aplikácií s možnosťou vlastnej konfigurácie. Vzhľadom na to, že spoločnosť R-SYS vyvíja prevažne systémy, ktoré spracúvajú špecifické dáta z oblasti riadenia leteckej premávky, sú využívané hlavne simulácie Asterix dát nad mapovým podkladom. Tieto dáta predstavujú

ciele (lietadlá, lode), ktoré sú buď generované pomocou existujúcich knižníc, alebo prehrávané prostredníctvom prehrávača z nahrávok reálnej dopravy.

Samotný proces vykonávania testov prebieha manuálne, simuláciou jednotlivých krokov testovacích prípadov za pomoci priložených testovacích dát, ktoré sú potrebné pre úspešný priebeh testu.

2.5. Zhodnotenie možností integrácie nástrojov a mechanizmov pre automatizovanie testovacích postupov a scenárov

V procese analýzy súčasne dostupných softvérových produktov, ktoré riešia problematiku testovania GUI natívnych aplikácií, boli overené viaceré testovacie nástroje s cieľom vyhodnotiť ich efektívnosť, použiteľnosť a možnosti vzájomnej integrácie. Testovanie nástrojov prebiehalo štandardne v prostredí operačného systému Windows 10 s nainštalovanou aplikáciou C2D [39], ktorá je v súčasnosti vyvíjaná spoločnosťou R-SYS. Jednotlivé nástroje boli skúšobne integrované do procesu testovania, na základe ktorého bolo potrebné overiť nasledujúce vlastnosti:

- automatické spustenie testovanej aplikácie,
- automatické spustenie podporných simulačných nástrojov,
- základný prechod testovanou aplikáciou a identifikovanie známych chýb,
- automatické vygenerovanie reportov.

2.5.1. Squish for Windows

Komerčný testovací nástroj Squish for Windows [40], vyvíjaný spoločnosťou Froglogic, je primárne určený pre natívne aplikácie v operačnom systéme Windows, s GUI založenom na Windows Forms a novších. Pri týchto aplikáciách vie nástroj pomerne spoľahlivo mapovať jednotlivé GUI objekty, čítať obsah a pokiaľ sú editovateľné aj meniť ich stav.

Princíp fungovania nástroja je podobný princípu vytvárania Excel makier v MS Office [41]. Základná štruktúra testu je založená na nahrať činnosti používateľa v spôsobe simulovaného ovládania testovanej aplikácie pomocou myši a klávesnice. Následne je test upravený a doplnený o očakávané reakcie testovanej aplikácie na dané podnety, vyvolané ovládacími prvkami GUI. Posledným krokom je spustenie celého testu a overenie, či funguje podľa zadania.

Počas testovania produktu Squish for Windows v prostredí aplikácie C2D, začal nástroj vykazovať viaceré nedostatky. Spustenie aplikácie a simulačných nástrojov prebehlo v poriadku, ale problém nastal pri základnom prechode testovanou aplikáciou, kedy Squish

nedokázal nájsť špecifické prvky GUI. Tieto chyby nastávajú z toho dôvodu, že prvky sú mapované na základe pomenovania textu v záhlaví okna, ktorý nemusí byť vždy rovnaký. Squish ponúka pre takéto prípady alternatívu vo forme virtuálneho posúvania myši a vykonávania akcií na konkrétnych súradniciach obrazovky. Tento princíp je pri podobných projektoch prakticky nepoužiteľný, keďže výsledok testu závisí od parametrov testovacieho stroja, ktoré sa môžu v priebehu vývoja meniť.

2.5.2. TestComplete

Ďalší známy komerčný produkt, ktorý ponúka možnosť testovania natívnych aplikácií, je testovací nástroj TestComplete vyvíjaný spoločnosťou SmartBear Software [35]. Testy v nástroji je možné vytvárať pomocou skriptovacieho jazyka, záznamu akcie používateľa alebo kombináciu oboch možností. Pre vytváranie testov pomocou záznamu a ich následnej údržbe je k dispozícii funkcia Test Visualizer. Každá akcia v skripte má priradený zodpovedajúci interaktívny snímok obrazovky, v ktorom je zvýraznený použitý GUI objekt.

Overenie základnej funkcionality nástroja TestComplete v GUI aplikácie C2D nebolo úspešné. Záznam základného prechodu testovanou aplikáciou prebehol úspešne, pričom bol vygenerovaný skript, ktorý po spustení ihneď ukončil priebeh testovania s chybným výpisom o nenájdenných objektoch GUI. Prvý problém, ktorý spôsobil zlyhanie testu, boli všeobecne pomenovania GUI objektov, na základe ktorých nebolo možné presne mapovať objekty v prostredí aplikácie. Druhým problémom bolo neúspešné prepojenie nástroja s GUI C2D prostredníctvom API (*Application Programming Interface* angl.) knižnice Qt [42].

2.5.3. Autolt a AutoHotKey

Ide o dva odlišné produkty, ktorých princíp fungovania je úplne rovnaký. Autolt [43] a AutoHotKey [44] sú voľne dostupné skriptovacie nástroje, ktoré nie sú primárne určené pre testovacie účely, ale slúžia skôr k automatizácii rutinných činností v prostredí operačného systému Windows. To znamená, že nástroje neobsahujú komponenty pre zachytávanie a reportovanie chýb. Na druhej strane môžu byť tieto komponenty nahradené skriptami, ktoré kontrolujú výstupy aplikácie a vygenerujú výsledky testov. Napriek týmto obmedzeniam bola odskúšaná použiteľnosť nástrojov v GUI aplikácie C2D a výsledok bol podobný ako pri komerčných testovacích nástrojoch. Skriptovací jazyk, ktoré oba nástroje používajú síce indikoval väčšinu aktívnych prvkov GUI, ale problém nastal pri akciách typu „drag and drop“, kde sú využívané konkrétne súradnice objektov. Nástroje taktiež

neposkytujú grafickú verifikáciu objektov v mapovom poli, čím je ich využitie v projekte značne obmedzené.

Vzhľadom na zistenia, ktoré vyplynuli z analýzy vyššie uvedených testovacích nástrojov bolo vyhodnotené, že v súčasnosti nie je k dispozícii taký testovací nástroj, ktorý by vyhovoval komplexným testovacím požiadavkám projektu. Preto bolo navrhnuté alternatívne riešenie s využitím viacerých voľne dostupných nástrojov, ktoré medzi sebou vzájomne interagujú a vytvárajú tak ucelený testovací systém, ktorý je nezávislý od prístupu ku GUI, ako aj od samotnej platformy, pre ktorú je aplikácia určená.

3. Návrh metodiky efektívneho aplikovania a integrácie nástrojov automatického testovania desktopových aplikácií s GUI

Spoločnosť R-SYS s.r.o., ktorá je zameraná na vývoj aplikácií v oblasti riadenia letovej prevádzky, v rámci svojho portfólia vyvíja systémy bez grafického rozhrania, ale aj systémy určené pre koncové grafické pracovné stanice. Na základe požiadaviek spoločnosti (tab. 3) bolo potrebné navrhnúť vhodné riešenie práve v oblasti testovania grafických aplikácií, kde je dostatočný priestor pre zefektívnenie testovania formou aplikovania voľne dostupných automatizačných nástrojov.

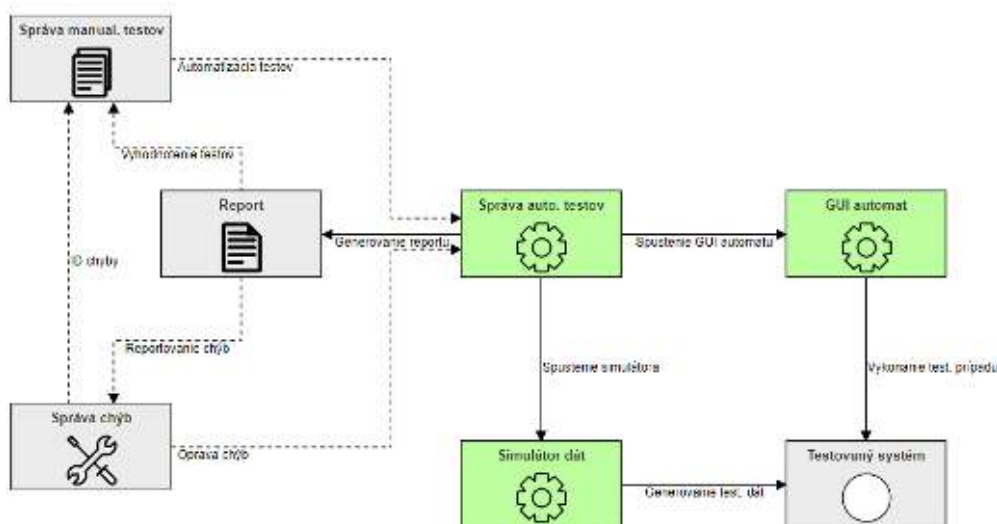
#	Systémové požiadavky	Priorita
Rq1	Systém musí využívať voľne dostupné automatizačné nástroje a mechanizmy	Vysoká
Rq2	Systém musí byť všeobecne aplikovateľný na desktopové aplikácie s grafickým rozhraním v OS Windows 10	Vysoká
Rq3	Systém musí umožniť vykonávať všetky možné akcie nad GUI objektami SUT	Stredná
Rq4	Systém musí po ukončení testovania automaticky vyhodnotiť výsledky testov a vygenerovať report vo formáte CSV resp. HTML	Vysoká
Rq5	Systém musí byť integrovateľný s existujúcimi testovacími nástrojmi a mechanizmami v spoločnosti	Stredná
Rq6	Systém musí podporovať viaceré spôsoby ovládania GUI	Nízka
Rq7	Systém musí podporovať základnú prácu so súborovým systémom, automaticky spúšťať interné simulačné nástroje (skripty) a aplikácie tretích strán	Stredná

Tab. 3 Požiadavky na systém automatického testovania

3.1. Návrh integrácie testovacích nástrojov a mechanizmov

Súčasťou navrhovaného riešenia sú tri hlavné nástroje, ktoré prostredníctvom vzájomnej integrácie umožnia automatizovať proces manuálneho testovania na rôznych úrovniach. Medzi tieto nástroje patrí:

- nástroj pre automatické ovládanie GUI (GUI automat),
- nástroj pre správu, spúšťanie a vyhodnocovanie testov (Správa auto. testov),
- nástroj pre simuláciu testovacích dát (Simulátor dát).



Obr. 6 Model integrácie testovacích nástrojov a mechanizmov

Model zobrazený na obrázku 6, vychádza zo súčasného modelu použitých testovacích nástrojov v procese manuálneho testovania (obr. 5). Zelenou farbou sú označené automatizačné nástroje, ktoré sú súčasťou navrhovaného testovacieho systému. Šípky vychádzajúce z týchto nástrojov predstavujú činnosti, ktoré budú plne automatizované a pri svojom vykonávaní nebudú vyžadovať manuálny zásah. Naopak, prerušovanými šípkami sú vyznačené tie úlohy, ktoré budú v určitej miere vyžadovať manuálny prístup. Príkladom môže byť proces vyhodnocovania vygenerovaných reportov alebo samotná automatizácia manuálnych testov.

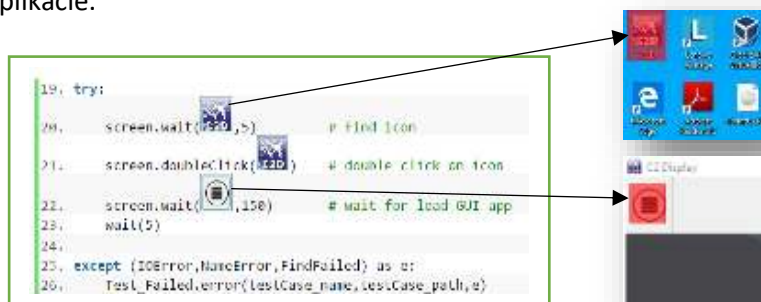
Testovací systém počíta s aplikovaním princípu automatického ovládania GUI na základe rozpoznávania grafických objektov. Táto funkcionality bude zabezpečená prostredníctvom voľne dostupného automatizačného nástroja SikuliX [36]. Princíp založený na optickom rozpoznávaní objektov bol navrhnutý z dôvodu:

- viacerých neúspešných pokusov o napojenie sa ku GUI testovanej aplikácii,
- testovania aplikácií vyžadujúcich prácu s objektami nad mapovým podkladom,
- testovania aplikácií v rôznych operačných systémoch,
- použiteľnosti pri testovaní natívnych aj webových aplikácií,
- možnosti grafickej verifikácie objektov.

Správu automatických testov, spúšťanie testov, generovanie reportov a vykonávanie príkazov v operačnom systéme, umožní nástroj JMeter [32] od spoločnosti Apache Software Foundation. Nástroj JMeter bude taktiež prepojený s internými simulačnými nástrojmi, vďaka čomu bude možné v testovanej aplikácii zobrazovať aktuálne dáta pre špecifické typy testov.

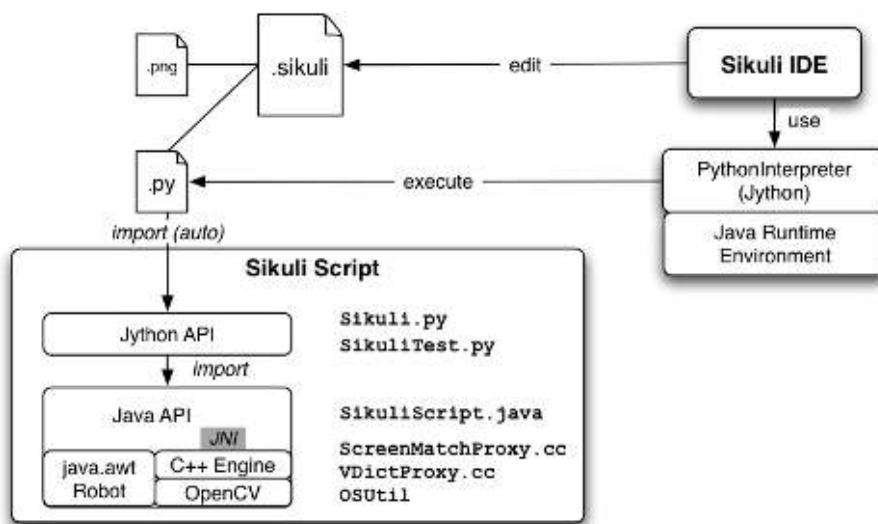
3.1.1. SikuliX

SikuliX je voľne dostupný softvér s otvoreným zdrojovým kódom [47] (MIT licencia), ktorý v rámci výskumu vyvinul tím Sikuli Lab na Univerzite v Colorade (University of Colorado Boulder). Ide o automatizačný nástroj, ktorý používa technológiu rozpoznávania obrazu na identifikáciu a kontrolu prvkov GUI, bez potreby znalosti zdrojového kódu testovanej aplikácie.



Obr. 7 Vizualné skriptovanie a rozpoznávanie obrázkov

Nástroj podporuje skriptovanie prostredníctvom integrovaného vývojového prostredia (Sikuli-IDE, *Integrated Development Environment* angl.). Súčasťou nástroja je API podporujúce skriptovanie prostredníctvom vizualizačných skriptov (obr. 7) a syntaxe programovacieho jazyka Jython, ktorý dovoľuje spustiť kód programovacieho jazyka Python pomocou JVM (*Java Virtual Machine* angl.). Jazyk Jython obsahuje všetky moduly, ktoré sú štandardnou súčasťou distribúcie jazyka Python.



Obr. 8 Architektúra nástroja SikuliX [45]

Jadrom SikuliX skriptu je knižnica Java, ktorá sa skladá z dvoch častí. Pomocou triedy *java.awt.Robot* sú implementované akcie udalostí vykonané pomocou myši alebo klávesnice. Funkcie na rozpoznávanie obrazu sú implementované prostredníctvom OpenCV [46] knižníc počítačového videnia v jazyku C++. Samotný projektový adresár (*.sikuli*) pozostáva zo zdrojového kódu Python (*.py*) a všetkých obrazových súborov (*.png*), ktoré zdrojový súbor používa. Všetky obrazové vzory použité v skripte sú označené cestou k (*.png*) súboru v projektovom adresári. Preto môže byť zdrojový súbor editovateľný ľubovoľným textovým editorom.

SikuliX podobne ako AutoIt a AutoHotKey neobsahuje komponenty pre vytváranie testovacích plánov, vyhodnocovanie testov a generovanie reportov. Tieto funkcie bude zabezpečovať testovací nástroj JMeter.

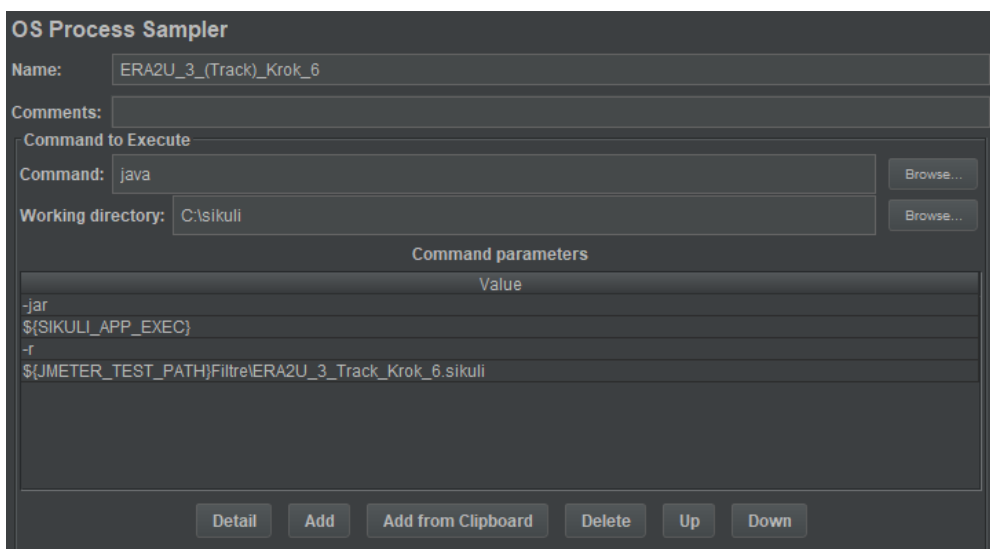
3.1.2. JMeter

Apache JMeter [32] je voľne dostupný automatizačný testovací nástroj, určený primárne pre záťažové testovanie, meranie výkonnosti a testovanie funkčnosti. Vznikol pre účely webového testovania, no dnes už zahŕňa aj komponenty pre testovanie FTP (*File*

Transport Protocol angl.), databáz, mailov a pod. Nástroj je napísaný a vyvíjaný v programovacom jazyku Java.

JMeter pracuje na základe testovacieho plánu, ktorý mu zostaví používateľ. Pre celú skupinu testov platia zadefinované premenné v komponente *User Variables*, ktoré využívajú jednotlivé úlohy v testovacích cykloch. Počas týchto cyklov poskytujú testovacie funkcie dynamický vstup do testu alebo manipuláciu so získanými hodnotami. Testovací plán predstavuje zoznam krokov, ktoré budú vykonané počas testovania. Plán môže obsahovať logické ovládače, vzorkovače, časovače a iné konfiguračné elementy. Všetky tieto elementy patria do určitej hierarchie a preto ich umiestnenie v testovacom pláne má vplyv na priebeh testovania.

Základný komponent, ktorý budeme v rámci navrhovaného testovacieho systému využívať, je *OS Process Sampler*. Tento komponent zabezpečí spúšťanie SikuliX skriptov a simulačných skriptov v nástroji JMeter prostredníctvom vykonaných príkazov na lokálnom počítači. Príklad spustenia SikuliX skriptu v *OS Process Sampler* je zobrazený na obrázku 9.

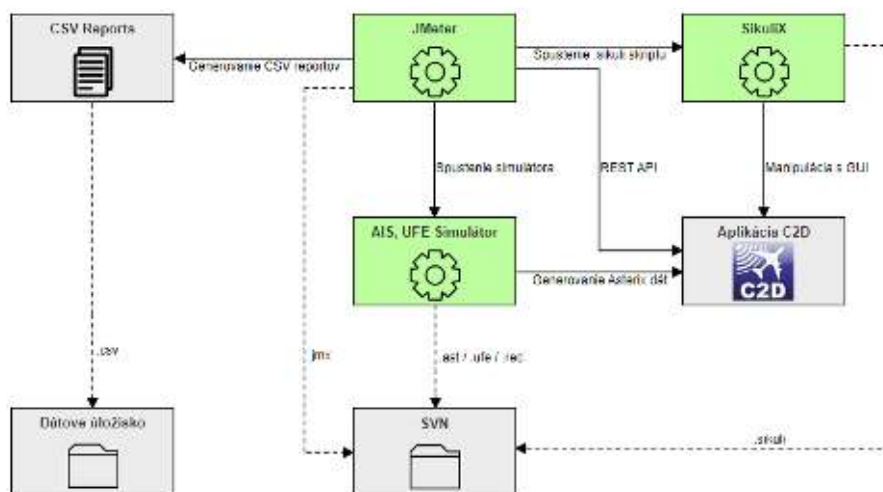


Obr. 9 Príklad spustenia SikuliX skriptu v nástroji JMeter

V testovacom pláne bude mať každý automatický SikuliX test priradený práve jeden *OS Process Sampler*. Spustením procesu sa automaticky vykoná simulácia prostredníctvom nástroja SikuliX. Po ukončení simulácie JMeter vyhodnotí výsledok testu na základe logov, ktoré vznikli v priebehu vykonávania skriptu. Výsledky testov budú sumarizované v komponente *View Results Tree*, odkiaľ sa automaticky vygenerujú do výsledného reportu vo formáte CSV (*Comma-Separated Values* angl.).

3.2. Model integrácie vybraných testovacích nástrojov a mechanizmov

Testovací framework bol navrhnutý na základe voľne dostupných riešení, s prihliadnutím na použiteľnosť vo viacerých projektoch. Modularita systému umožňuje jednoduchú integráciu komponentov a nastavenie automatizácie testovania podľa konkrétnych požiadaviek projektu. V súlade s návrhom bol zostavený model integrácie vybraných testovacích nástrojov a mechanizmov (obr. 10), ktorý zohľadňuje aktuálny stav aplikácie C2D a požiadavky kladené na automatické vykonávanie testovacích prípadov.



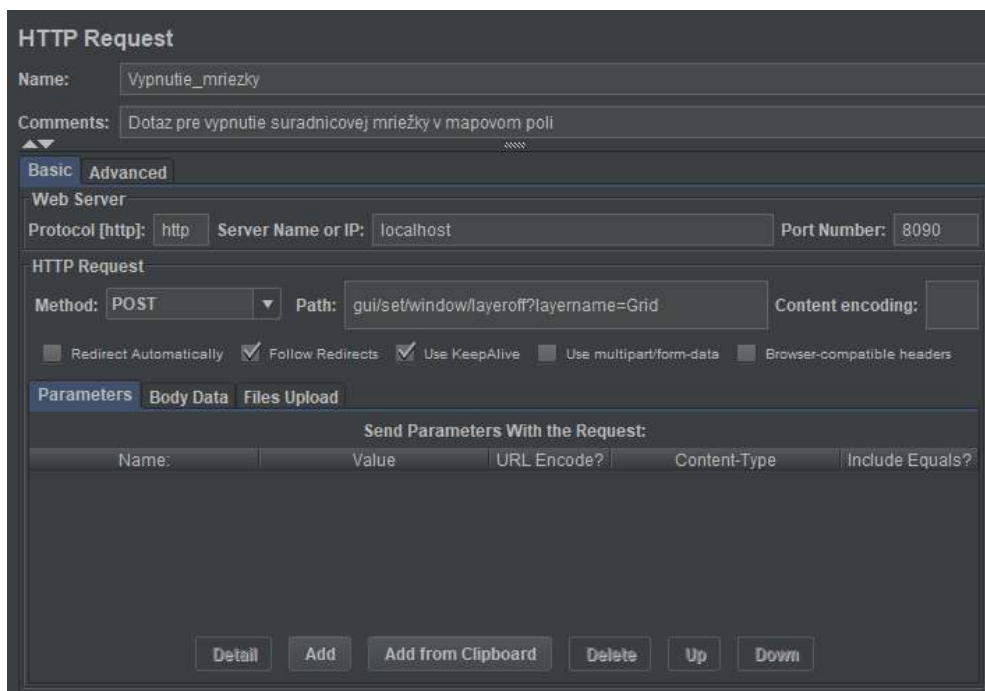
Obr. 10 Model integrácie vybraných automatizačných nástrojov a mechanizmov

Základ systému tvoria tri vybrané automatizačné nástroje označené zelenou farbou. Ich vzájomnou interakciou sú súčasne spúšťané procesy s presne určenou časovou postupnosťou, ktorá zodpovedá pripravenému testovaciemu plánu (viď kap. 4.3. tab. 8). Zdrojové súbory, ako aj vzorové dáta týchto nástrojov sú verzované prostredníctvom verzovacieho systému TortoiseSVN [48]. Vygenerované reporty vo formáte CSV sú ukladané samostatne na testovacie dátové úložisko reportov, pričom štruktúra dátového úložiska zodpovedá aktuálnemu číslovaní verzií testovanej aplikácie. V prípade výskytu chýb v testovanej aplikácii je report doplnený o samostatný adresár, ktorý disponuje aktuálnymi snímkami obrazovky v čase výskytu chýb a ich očakávanými výsledkami. Týmto spôsobom je zabezpečená rýchlejšia identifikácia chýb v testovanej aplikácii, prípadne iné príčiny, ktoré spôsobujú zlyhanie automatických testov.

Model reprezentovaný na obrázku 10, zobrazuje priamu interakciu nástroja JMeter s testovanou aplikáciou C2D, prostredníctvom REST (*Representational State Transfer* angl.) API rozhrania. Táto funkcionality je dostupná na základe existujúceho interného ladiaceho rozhrania aplikácie C2D [42], ktorá rozširuje možnosti ovládania a monitorovania jednotlivých procesov, najmä pre ladiace účely a automatické testovanie. Aplikácia C2D s ladiacim rozhraním

podporuje možnosť pripojenia a posielania požiadaviek pomocou HTTP protokolu. Posielanie HTTP požiadaviek zabezpečuje nástroj JMeter prostredníctvom vzorkovača *HTTP Request*, ktorého príklad je zobrazený na obrázku 11. Je v ňom potrebné vyplniť nasledovné parametre:

- typ protokolu,
- názov servera alebo IP adresu,
- port servera (daný konfiguračne),
- typ metódy pre prístup k zdrojom,
- jednotný identifikátor zdroja (URI, *Uniform Resource Identifier* angl.).



Obr. 11 *HTTP Request Sampler* pre vypnutie súradnicovej mriežky v mapovom poli

V prípade potreby posielania viacerých HTTP/HTTPS požiadaviek na tú istú IP (*Internet Protocol* angl.) adresu s rovnakým portom, je výhodné použiť *HTTP Request Defaults* komponent, v ktorom sú zadefinované predvolené hodnoty. Následne už potom nie je potrebné vyplňať rovnaké parametre pre jednotlivé HTTP požiadavky.

V URI HTTP požiadavky je špecifikovaná skupina resp. príkaz a následne parametre. Ak nie je definované inak, HTTP požiadavky využívajú metódu GET. Tabuľa 4 popisuje základné akcie, ktoré je možné vykonať odoslaním jednotlivých typov správ.

Popis	URI	Parametre	Návratová hodnota	Podporované od verzie
Práca s cieľmi (dátovým modelom cieľov)				
Získanie celkového počtu trackov v internej tabuľke trackov	track/get/count	n.a.	JSON formát: - true: { "Request Status" : 1, "All Tracks" : 0, "Dead Tracks" : 0, "Live Tracks" : 0 } - false: { "Request Status" : 0 }	V5.0
Reset (zmazanie) tabuľky trackov	track/reset	n.a.	JSON formát: - true: { "Request Status" : 1 } - false: { "Request Status" : 0 }	V5.0
Ovládanie GUI				
Nastavenie stredu zobrazenia a mierky	gui/set/window/view/main	center=<lon>,<lat>, <scale>	JSON formát: - true: { "Request Status" : 1, "Position Lat" : 48.2661, "Position Lon" : 19.3822, "Scale" : 10000 } - false: { "Request Status" : 0, "Request Text" : "Center map position is not specified" }	V5.0
Zatvorenie okna v aplikácii podľa jeho titulku	gui/performanceaction/window/close	title=TITULOK	n.a.	V5.2
Zapnutie/vypnutie vrstvy zobrazenia	gui/set/window/layeron alebo gui/set/window/layeroff	layername = <name>	JSON formát: - true: { "Request Status" : 1, "Request Text" : "Layer Analyses activated" } - false: { "Request Status" : 0, "Request Text" : "Layer Analyse not found" }	V5.0
Nástroje a pomocne príkazy				
Získanie kópie obrazovky hlavného okna	tools/screenshot/main	- bez parametrov vráti screenshot hlavného okna - parameter box=X,Y,width,height definuje výrez	screenshot vo formate .png JSON formát: - false: { "Request Status" : 0 }	V5.0

		obrazu z hlavného okna		
Import konfigurácie aplikácie	tools/set/app/config/dynamic	path=/path/to/dynamicConfig.zip - ZIP s dynamickou konfiguráciou musí byť na stanici spolu s C2D - Aplikácia C2D sa ukončí s kódom 100.	JSON formát: - true: { "Request Status" : 1, "Request Text" : "Dynamic config is importing from file /path/to/dynamicConfig.zip" } - false: { "Request Status" : 0, "Request Text" : "Dynamic config file /path/to/dynamicConfig.zip does not exist!" }	V5.2

Tab. 4 Tabuľa HTTP požiadaviek pre testovací interface aplikácie C2D [31]

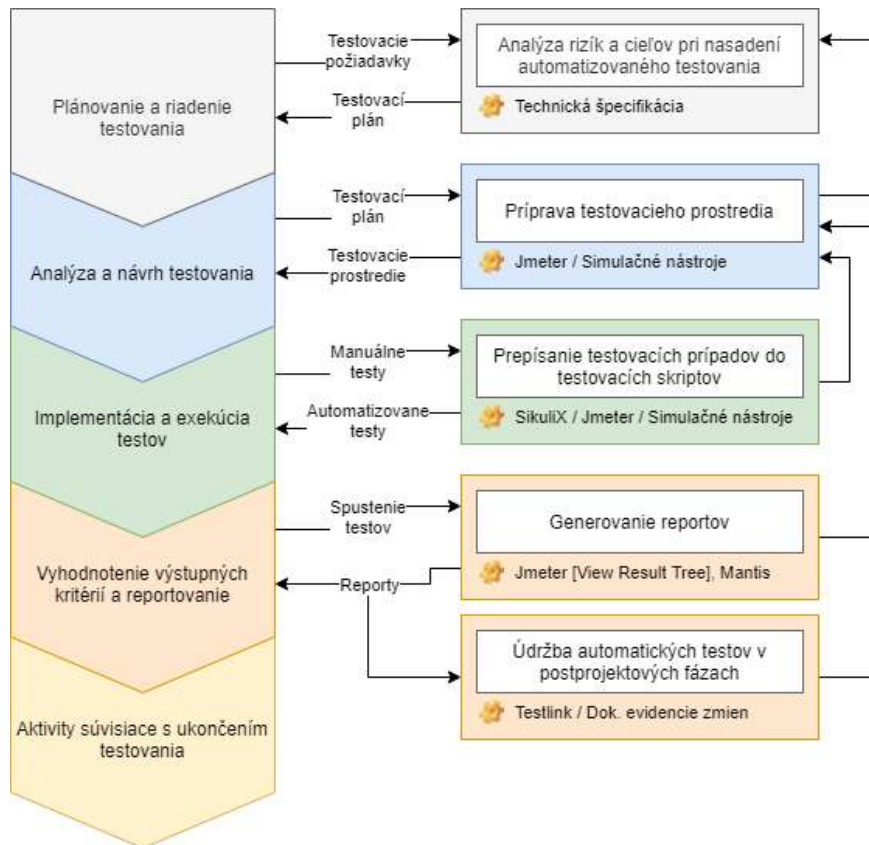
Kompletný zoznam HTTP požiadaviek, ktoré podporuje aplikácia C2D, je popísaný v internej projektovej dokumentácii aplikácie C2D [31] (viď príloha D). Pre overenie návratových hodnôt, ako v príklade získania celkového počtu trackov v internej tabuľke cieľov, slúži v nástroji JMeter komponent *JSON assertion*. Prostredníctvom tohto komponentu je možné porovnať získaný špecifický reťazec alebo kód návratovej odpovedi servera s očakávanými výsledkami.

V prípade, ak by desktopová aplikácia nedisponovala testovacím rozhraním, môžu byť akcie popísané v tabuľke 4 nahradené SikuliX skriptami. Avšak, ak sú k dispozícii obidva spôsoby, je výhodnejšie využívať metódy interného ladiaceho rozhrania, a to najmä z dôvodu rýchlosti, údržby a jednoduchosti testov.

3.3. Proces automatického testovania VSBT

Základné kroky obsiahnuté v metodike VSBT (*Visual Script Based Testing* ang.), sú zapracované v procesnom modeli, ktorý je zobrazený na obrázku 12. Model pozostáva z dvoch častí. Ľavá časť diagramu reprezentuje základný testovací proces. V pravej časti sú pod sebou definované základné fázy metodiky:

- analýza rizík a cieľov pri nasadení automatického testovania,
- príprava testovacieho prostredia,
- prepísanie testovacích prípadov do testovacích skriptov,
- generovanie reportov,
- údržba automatických testov v postprojektových fázach.



Obr. 12 Procesný model testovania s využitím navrhovanej metodiky

3.3.1. Analýza rizík a cieľov pri nasadení automatického testovania

Pri úvahe automatizovať testovací proces je dôležité počítať s rizikami, ktoré môžu negatívne ovplyvniť vývoj celého projektu. Ešte pred samotnou aplikáciou automatického testovania je nutné zhodnotiť:

- vhodnosť testovaného systému k automatizácii,
- výber použitých automatizačných a simulačných nástrojov,
- reálnu úsporu zdrojov pri automatizácii.

Bližší popis k jednotlivým rizikám a cieľom je obsiahnutý v prílohe B: Metodická príručka.

3.3.2. Príprava testovacieho prostredia

Pre úspešné aplikovanie metodiky v projekte je potrebné stiahnutie a inštalácia dvoch základných voľne dostupných programov:

- Apache JMeter [32]
- SikuliX [36]

Tieto nástroje môžu byť doplnené o ďalšie podporné simulačné nástroje alebo simulačné skripty. V našom prípade budeme pre simuláciu využívať simulačné skripty písané v jazyku Perl (*ais_player.pl*) a JavaScript (*gen-basic-track.js*). Okrem toho budeme taktiež využívať internú aplikáciu *ERA Record Player* vo verzii 4.0.4. Podporné simulačné nástroje a skripty budú slúžiť pre generovanie a vizualizáciu dát vzdušnej situácie nad mapovým podkladom.

Samozrejmosťou je taktiež inštalácia samotnej testovanej desktopovej aplikácie a všetkých potrebných komponentov. Vzhľadom na to, že budeme v rámci testov využívať skripty písané v jazyku JavaScript a Perl, je zároveň potrebné stiahnuť a nainštalovať prostredie pre ich spúšťanie:

- Node.js [49] – serverový framework pre JavaScript,
- ActivePerl [50] – implementácia programovacieho jazyka Perl pre Windows.

Ak sú všetky potrebné nástroje a komponenty nainštalované, je možné pristúpiť ku konfigurácii testovanej aplikácie, prípadne aj samotných automatizačných nástrojov. V prípade testovanej aplikácie C2D je pred začiatkom testovania potrebná základná konfigurácia kanálov pre prijímanie dát zo simulačných nástrojov a skriptov. Bližšie informácie o konfigurácii aplikácie C2D sú uvedené v kapitole 4.2.

3.3.3. Prepísanie testovacích prípadov do testovacích skriptov

Štruktúra testovacieho prípadu v nástroji Testlink (viď kap. 2.4.) pozostáva z viacerých častí. Príkladný testovací prípad v tabuľke 4 zobrazuje základnú štruktúru testu pre vytváranie logických kombinácií "dočasných" filtrov, s použitím operátorov AND a OR, v aplikácii C2D.

Test Case ERA2U-3: RA.6.2 Umožniť vytváranie logických kombinácií "dočasných" filtrov s použitím operátorov AND a OR. [Version : 2]	
Author:	Pavol Bednárík - 11/04/2017 10:24:23
Last edit by:	Stanislav Valica - 18/12/2019 14:50:01
<p>Summary:</p> <p>Umožniť vytváranie logických kombinácií filtrov ako dvojstupňové logické výrazy s použitím AND a OR tak, ako je to riešené v pôvodnej aplikácii ERA Display.</p> <p>Na základe tejto požiadavky dôjde k zmene editačného okna tak, ako je riešené v aplikácii Display. Toto riešenie okrem iného prinesie možnosť vytvárania logických kombinácií filtrov pomocou operátora „OR“ a AND.</p> <p>Okno umožní vytvoriť tzv. „Anonymný“ filter. Ide o filter, ktorý nebude uložený pod menom, t.j. po jeho definícii sa priamo aktivuje (tlačidlo „Set“) v príslušnej funkcii, z ktorej bol vyvolaný. Po ukončení aplikácie C2D sa definícia tohto filtra stratí.</p>	

<u>Preconditions:</u>		
-Na vykonanie testu potrebujeme bežiacu aplikáciu C2D_DPET.		
<u>Test Inputs:</u>		
-Nastavený aktívny kanál.		
-Dáta trackov, AIS a AIS_UFE musia byť prijímané.		
<u>#:</u>	<u>Step actions:</u>	<u>Expected Results:</u>
1	<ol style="list-style-type: none"> 1. V hornej lište zapnúť tlačidlo Track. 2. Skontrolovať položku Filter. 3. Kliknúť na položku Edit vedľa comboboxu filter v hornej lište. 4. Označiť [temporary] a potvrdiť set. 5. Opäť aktivovať Track Filter Definition a vo filtri [temporary] zadefinovať nejaké logické operátory. Po použití set skontrolovať funkčnosť filtra. 6. Reštartovať aplikáciu c2d a overiť vynulovanie filtra [temporary]. 7. Postup opakovať aj pre AIS filtre. (bod 1 prepnúť AIS) 	<ol style="list-style-type: none"> 1. Track je zapnutý a aktívny (filtre platia pre tracky) 2. Položka Filter obsahuje --no filters--. 3. Aktivuje sa okno Track Filter Definition. 4. Položka Filter obsahuje [temporary]. 5. Je možné definovať filter a ten správne zobrazí v mape tracky podľa nastavených logických operátorov. 6. Po reštarte aplikácie sa logické operátory a nastavenie filtra vynuluje do prednastaveného stavu. 7. Výsledok je rovnaký a platný aj pre AIS.
<u>Execution type:</u>	Manual	
<u>Exec. duration (min):</u>	30	
<u>Importance:</u>	Medium	
<u>Action on System Halt:</u>	-reštart aplikácie c2d -zopakovať celý testovací postup od začiatku	
<u>Recording and reduction analysis:</u>	-zálohovať dump súbor -analýza vytvoreného dump súboru -analýza vstupných dát v jednotlivých krokoch testovacieho postupu	
<u>Requirements</u>	RA.6.2: Logické kombinácie filtrov ako logické výrazy s použitím AND a OR	
<u>Keywords:</u>	None	

Tab. 5 Príklad testovacieho prípadu v nástroji TestLink

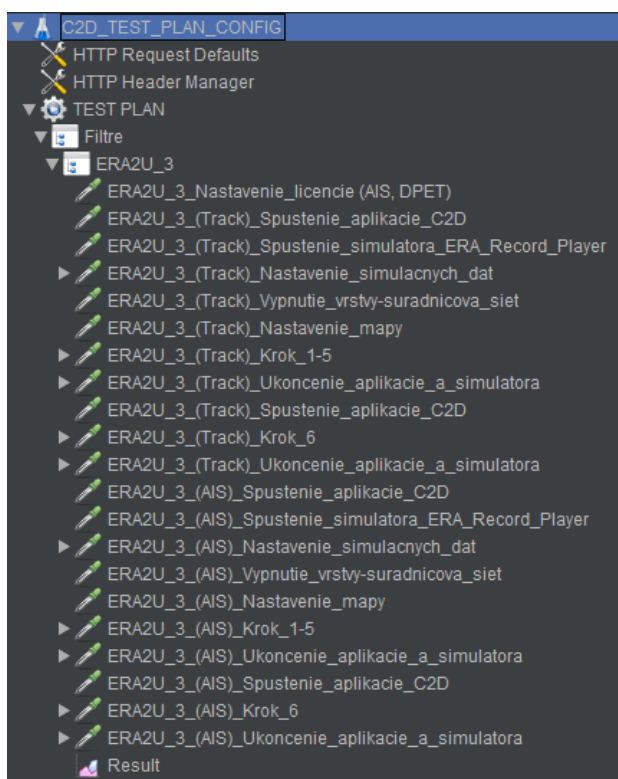
Ako je zřejmé z testovacích predpokladov a vstupov, test funkcionality bude vyžadovať spustenú aplikáciu C2D s licenciou DPET a nakonfigurovaným kanálom pre príjem a zobrazenie informácií AIS (*Automatic Identification System* angl.) a AIS_UFE (*Unified Format ERA* angl.) nad mapovým podkladom. Krok 1.7 testovacieho prípadu popisuje opakované vykonanie celého testu s dátami typu AIS. V našom prípade to znamená, že nami vytvorený automatický test bude rozdelený na dve časti. V prvej časti budú pri vykonávaní

testu spustené simulačné dáta typu AIS_UFE (informácie o lietadlách) a v druhej časti budú pri vykonávaní testu spustené simulačné dáta typu AIS (informácie o plavidlách/lodiach).

Pre vytvorenie automatického testu je potrebné v nástroji JMeter otvoriť predpripravenú testovaciu šablónu *TestPlan_C2D.jmx*, ktorá je umiestnená v priloženom adresári súborov [Automaticke_testy\Projekt_C2D\]. Šablóna obsahuje 4 základné komponenty:

- **C2D_TEST_PLAN_CONFIG** – Konfigurácia ciest k spustiteľným súborom pre nástroj SikuliX a testovanú aplikáciu,
- **HTTP Request Defaults** – Konfigurácia predvolených parametrov pre posielanie HTTP požiadaviek (viď popis k obr. 11),
- **HTTP Header Manager** – Konfigurácia hlavičiek pre HTTP požiadavky,
- **TEST PLAN** – Vlákno pre vytvorenie kompletnej štruktúry automatických testov.

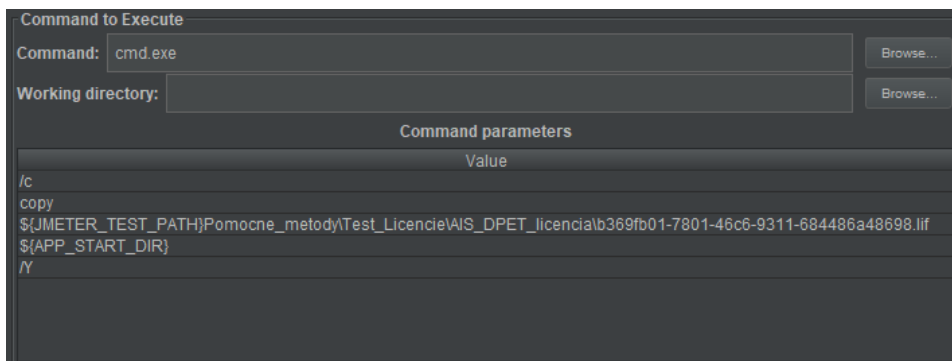
Na základe zistených informácií z testovacieho prípadu vytvoríme základnú štruktúru automatického testu, ktorý pozostáva z nasledovných častí:



Obr. 13 Štruktúra automatického testu ERA2U_3 v nástroji JMeter

Nastavenie licencie – Na začiatku testu je prostredníctvom komponentu *OS Process Sampler* zabezpečené správne nastavenie softvérovej licencie pre testovanú aplikáciu.

Nastavenie licencie prebieha prekopírovaním predpripraveného licenčného súboru z testovacích dát do adresára s nainštalovanou testovanou aplikáciou C2D. Táto operácia je vykonaná prostredníctvom MS-DOS príkazu *copy* nasledovne:



Obr. 14 Nastavenie licencie pre test ERA2U_3

Premenné `${JMETER_TEST_PATH}` a `${APP_START_DIR}` sú definované v komponente `C2D_TEST_PLAN_CONFIG`.


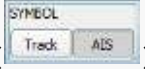



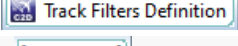
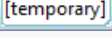

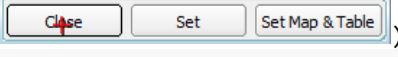
Spustenie aplikácie a simulátora – Pre spustenie aplikácií alebo simulátorov môžeme využiť komponent *BeanShell Sampler*, ktorý obsahuje skript pre vykonávanie príkazov prostredníctvom metódy `Java.lang.Runtime.exec()`.

- Spustenie aplikácie: `Runtime.getRuntime().exec("cmd.exe /c start " + vars.get("APP_START_PATH"));`
- Spustenie simulátora: `Runtime.getRuntime().exec("cmd.exe /c start " + vars.get("JMETER_TEST_PATH") + vars.get("SIMULATOR_START_PATH"));`

Odoslanie HTTP požiadaviek – V teste boli taktiež použité základné funkcie interného ladiaceho rozhrania aplikácie (tab. 4), medzi ktoré patrí *Vypnutie_vyrstvy* a *Nastavenie_mapy*. Odoslaním týchto HTTP požiadaviek pomocou *HTTP Request Sampler* (viď obr. 11), bude v aplikácii vypnutá súradnicová sieť pre lepšiu identifikáciu trackov a nastavené presné zobrazenie mapy podľa definovanej mierky a súradníc.

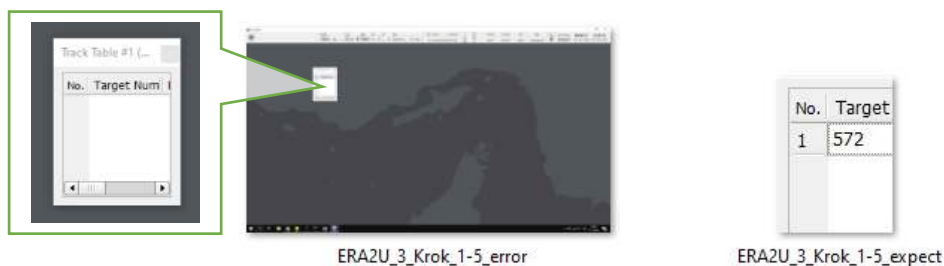
Spustenie SikuliX skriptov – SikuliX skripty sú spúšťané prostredníctvom komponentu *OS Process Sampler* (viď kap. 3.1.2. obr. 9). Po ich spustení sa v prostredí testovanej aplikácie vykoná hlavný testovací scenár (krok 1-7). Okrem toho môžeme prostredníctvom SikuliX skriptov vykonať aj konfiguračné nastavenia, ako napríklad nastavenie simulačných dát. Nasledujúci príklad zobrazuje časť SikuliX skriptu pre test ERA2U_3 (krok 6), ktorý overuje vynulovanie filtra a logických operátorov do prednastaveného stavu, po reštarte aplikácie C2D.

```

12. try:
13.     #Reštartovať aplikáciu c2d a overiť vynulovanie filtra [temporary].
14.     #Po reštarte aplikácie sa logické operátory a nastavenie filtra vynuluje
        do pôvodného stavu.
15.     screen.wait(,200)
16.     wait(3)
17.     if(screen.exists(,5):
18.         screen.click()
19.         wait(2)
20.         screen.find()
21.         screen.click()
22.         screen.wait(,5)
23.         screen.click()
24.         wait(2)
25.         screen.find()
26.         screen.click()
27.
28. except (IOError,NameError,FindFailed) as e:
29.     Test_Failed.error(testCase_name,testCase_path,e)

```

Test na začiatku očakáva úspešné spustenie a načítanie GUI testovanej aplikácie. Ak je na obrazovke identifikovaný objekt hlavného menu, následne zisťujeme, či je aplikácia v požadovanom režime *Track* pomocou funkcie `exists()`. Po kliknutí na položku *Track* musia byť logické operátory a nastavenie filtra v prednastavenom stave. To overíme prostredníctvom funkcie `find()` v časti *Filter*, a rovnako tak aj v samotnom okne pre definovanie filtrov. Ak vykonanie skriptu prebehne bez chyby, JMeter vyhodnotí komponent *ERA2U_3_(Track)_Krok_6* ako úspešný (označí ho zelenou farbou). V opačnom prípade je do adresára `[..\Projekt_C2D\Pomocne_metody\Test_Errors]` vygenerovaná aktuálna snímka obrazovky v čase výskytu chyby a očakávaný hľadaný objekt podobne, ako na obrázku 15.



Obr. 15 Vygenerovanie obrázkov pri výskyte chyby

JMeter vyhodnocuje výsledok testu na základe získaných logov z vykonaného SikuliX skriptu pomocou komponentu *BeanShell Assertion*. Ten obsahuje BeanShell skript, ktorý prehľadáva získané logy a hľadá v ňom kľúčové slovo *TestFailed*. Záznam informujúci o chybe pri vykonaní skriptu je zároveň pridaný do vygenerovaného reportu v stĺpci *FailureMessage*.

3.3.3.1. Pomocné metódy

Aby sme predišli často sa opakujúcim krokom v jednotlivých SikuliX skriptoch, sú v adresári [Automaticke_testy\Projekt_C2D\Pomocne_metody] vytvorené pomocné SikuliX skripty. Pomocné SikuliX skripty obsahujú funkcie, v ktorých sú implementované často sa opakujúce čiastkové operácie v testovanej aplikácii. Volanie týchto funkcií v skriptoch zvyšuje prehľadnosť testu a zjednodušuje jeho údržbu.

Adresár *Pomocne_metody* taktiež obsahuje:

- *Test_Licence* – súbor s testovacími licenciami,
- *Test_Configuration* – súbor s predvolenou konfiguráciou,
- *Test_Simulator* – súbor so simulačnými nástrojmi a dátami,
- *Test_Errors* – súbor s vygenerovanými obrázkami pri výskyte chýb,
- *Test_Results* – súbor s vygenerovanými reportmi testov.

3.3.4. Generovanie reportov

Generovanie reportov je v nástroji JMeter zabezpečené prostredníctvom komponentu *View Result Tree* (viď obr. 13 - Result). Po spustení testu tlačidlom *Start* sú v komponente postupne zobrazované výsledky jednotlivých krokov automatického testu. Po ukončení testovania je následne vygenerovaný report vo formáte CSV, ktorý je uložený v adresári [..\Projekt_C2D\Pomocne_metody\Test_Results].

label	responseCode	threadName	success	failureMessage
ERA2U 3 (Track) Spustenie aplikacie C2D	200	TEST PLAN 1-1	true	
ERA2U 3 (Track) Krok 6	0	TEST PLAN 1-1	false	ERA2U_3_Krok_6 TestFailed FindFailed: 1578263984061.png: (266x29) in R[0,0 1920x1080]@S(0) Unable to find expected object (..\Test_Errors\ERA2U_3_Krok_6_error.png) on screen (..\Test_Errors\ERA2U_3_Krok_6_expect.png)
ERA2U 3 (Track) Ukoncenie aplikacie C2D	0	TEST PLAN 1-1	true	

ERA2U 3 (Track) Ukoncenie simulatora	0	TEST PLAN 1-1	true	
--	---	------------------	------	--

Tab. 6 Príklad časti vygenerovaného testovacieho reportu

Ako zobrazuje príklad v tabuľke 6, výsledný vygenerovaný testovací report obsahuje nasledujúce informácie:

- Label – názov kroku v automatickom testovacom scenári,
- ResponseCode – návratový kód pri odoslaní požiadavky,
- ThreadName – názov vlákna, v ktorom bol test vykonaný,
- Success – výsledok testu,
- FailureMessage – záznam o výskyte chyby pri vykonaní testu.

3.3.5. Údržba automatických testov v postprojektových fázach

Metodika VSBT sa taktiež uplatňuje aj v postprojektových fázach údržby životného cyklu systému (SLC, *System Life Circle* angl.), ktorých súčasťou sú zmeny, opravy a rozšírenia. Úpravy softvéru sú častokrát zdrojom znefunkčnenia existujúcich automatických testov. Nápravu testov docielime:

- nahradením zastaralých testovacích prípadov novými,
- aktualizovaním existujúcich testovacích prípadov.

Tieto aktivity sú súčasťou prípravnej fázy testovania každej novej verzie testovaného systému, pričom musí byť stanovená vhodná stratégia regresného testovania (viď príloha B - kap. 1.1.3.).

Aby boli automatické testy neustále udržiavané v aktuálnom stave, je nevyhnutná ich aktualizácia po každej vykonanej zmene v manuálnych testovacích prípadoch. Evidencia zmien je popísaná v internej online dokumentácii testov podobne, ako zobrazuje príkladná tabuľka 6. Dokument evidencie zmien v testoch je prístupný pre celý testovací tím.

SADA	VYSLEDOK	TEST	MOD	AUTOR	REV.	EDITOVAL	ZMENA REV.	KOMENTAR
Filter	PASS	ERA2U-21	AUTO	VAL	1			

Filter	PASS	ERA2U-2	MANUAL	BED	3	VAL	4	Zmena krokov 1-8
Filter	FAIL	ERA2U-3	AUTO	KOK	1	VAL	2	Zmena krokov 1-4
Filter	PASS	ERA2U-8	MANUAL	VAL	1			

Tab. 7 Príklad tabuľky pre evidenciu zmien v testoch

Ak autor testu realizuje úpravy v aktuálne platnom testovacom scenári, je povinný o nich informovať prostredníctvom online dokumentu, v ktorom poznačí skratku svojho mena, aktuálnu verziu manuálneho testu a stručný komentár o vykonaných úpravách. Automatizovaný tester tak bude informovaný o všetkých úpravách v testovacích prípadoch a na základe týchto informácií prevedie potrebné zmeny v automatických testoch.

V prípade vizualizačných skriptov je dôležité využívanie pomocných metód (viď príloha B - kap. 1.3), prostredníctvom ktorých vieme výrazne rýchlejšie vykonať zmeny v často sa opakujúcich krokoch testovacieho prípadu. Taktiež je potrebné zamedziť duplikovaniu zhodných častí testov v rámci projektu. Vo všeobecnosti platí, že čas strávený údržbou automatického testu by nemal prevyšovať čas potrebný na jeho manuálne vykonanie.

Bližšie informácie k jednotlivým krokom a činnostiam metodiky sú obsiahnuté v prílohe B: Metodická príručka.

4. Experimentálne overenie metodiky pri automatizácii regresného testovania aplikácie C2D

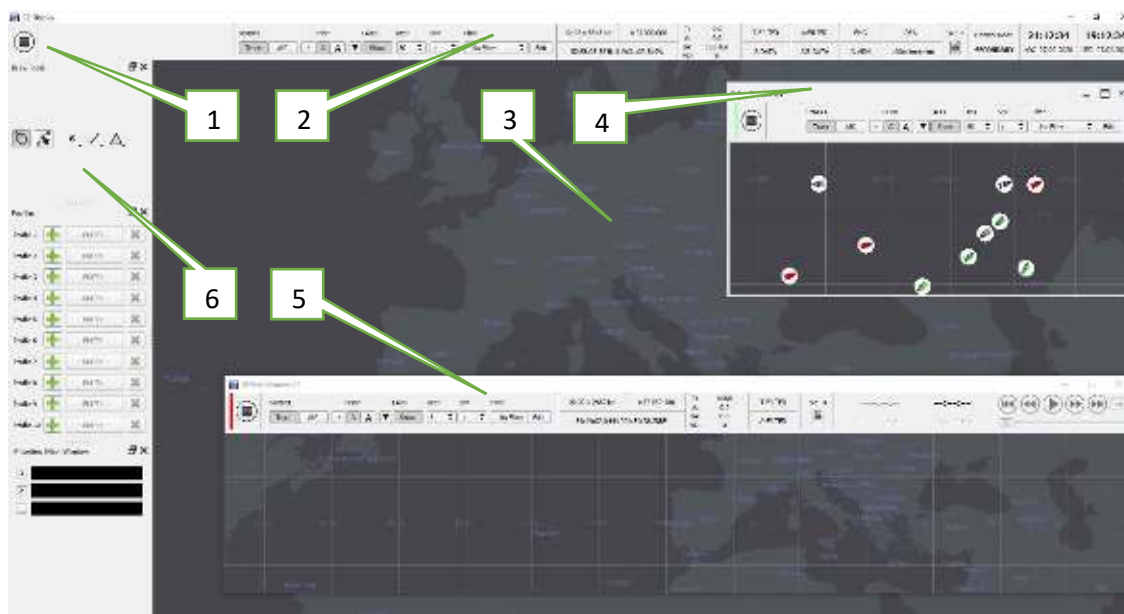
V rámci experimentálneho overenia metodiky VSBT, popísanej v kapitole 3, bola automatizovaná sada regresných testov: *ERA2U-519:Release test pred odoslaním setup-u_5.4 a 5.5* (viď príloha C). Tieto testy overujú v jednoduchých krokoch základnú funkčnosť naprieč celou aplikáciou C2D. Testy sú v súčasnosti vykonávané manuálne, v pravidelných iteráciách pred odoslaním setup-u do produkcie.

4.1. C2D

Aplikácia C2 Display (ďalej len C2D) slúži na prezentáciu UFE, ASTERIX a AIS informácií nad mapovým podkladom. Primárne je určená pre obsluhujúci personál pasívnych sledovacích systémov VERA-NG [54], pričom umožňuje používateľom pomocou GUI (v online móde) meniť parametre zobrazenia na základe atribútov pozorovaných objektov a nastavení vlastností prehľadového okna.

Aplikácia C2D je určená pre OS Windows a spúšťa sa štandardným spôsobom prostredníctvom ikony C2D zobrazenej na pracovnej ploche stanice. Aplikácia C2D môže pracovať v dvoch režimoch: režim pre bežného užívateľa ("User mode") a režim pre administrátora ("Admin mode") [51]. Pri spustení aplikácie je zistený režim, v ktorom je aktuálne aplikácia spustená a podľa toho sprístupní svoje funkcie. V prípade administrátora sú sprístupnené všetky jej funkcie a v prípade bežného užívateľa sú niektoré funkcie nedostupné.

Jednotlivé funkcie aplikácie C2D sú ovládané alebo aktivované prostredníctvom GUI aplikácie. Ukážka zobrazenia základného GUI aplikácie je uvedená na nasledujúcom obrázku.



Obr. 16 Ukážka zobrazenia základného GUI aplikácie C2D

Základné GUI aplikácie C2D pozostáva z nasledujúcich častí:

1. tlačidlo (ikona) na zobrazenie hlavného menu aplikácie,
2. ovládacia lišta aplikácie,
3. hlavné online prehľadové okno,
4. vedľajšie online prehľadové okno,
5. vložené offline prehľadové okno,
6. dokovacie okná.

V aplikácii C2D sa taktiež nachádzajú štandardne používané ovládacie prvky GUI, akými sú napríklad tlačidlá, prepínače, zaškrŕavacie a textové polia, ponukové zoznamy, stromy, záložky a pod.

4.2. Konfigurácia C2D a simulačných nástrojov

Pred samotným začiatkom prepísania testovacích prípadov do testovacích skriptov, bolo potrebné vykonať základnú konfiguráciu pre korektný príjem testovacích dát. Konfigurácia C2D prebiehala prostredníctvom C2D Offline Editora ("Admin mode"), pričom boli nastavené nasledujúce parametre:

- root -> c2d -> Common: Povolený **TestInterfacePort** s portom 8090
- root -> c2d -> DataChannels: Pridaný **TrackChannel[1]**
 - (Name: TEST1, Address: 127.0.0.1, Port: 50050, Category: FLOW)
- root -> c2d -> DataChannels: Pridaný **TrackChannel[2]**
 - (Name: TEST2, Address: 127.0.0.1, Port: 50000, Category: CAT 248 v1.15)
- root -> c2d -> DataChannels: Pridaný **AISChannel[1]**
 - (Name: TEST3, Address: 127.0.0.1, Port: 10001, ConnectionType: UDP)

Ďalej bola potrebná konfigurácia simulačných nástrojov a zoskupenie simulačných dát. Dáta spolu so simulačnými nástrojmi, ktoré sú dostupné v priloženom adresári [..\Projekt_C2D\Pomocne_metody\Test_Simulator], boli rozdelené do troch kategórii:

- **Generátor_AIS:** Obsahuje prehrávač *ais_player.pl* so zaznamenanými simulačnými nahrávkami AIS dát.
- **Generátor_AIS_UFE:** Obsahuje aplikáciu *ERA Record Player* so zaznamenanými simulačnými nahrávkami AIS_UFE dát.
- **Generátor_UFE:** Obsahuje aplikáciu *ERA Record Player* so zaznamenanými simulačnými nahrávkami UFE dát.

Automatické spustenie simulačných nástrojov *ERA Record Player* a *ais_player.pl* je popísané v kapitole 3.3.3 – *Spustenie simulátora*. Samotné nastavenie a spustenie simulačných dát zabezpečuje funkcia **set_simulation_data(MyDataPath)**, kde *MyDataPath* predstavuje cestu k potrebnej nahrávke (viď kap. 4.3. – STEP3_1.sikuli: 21. riadok). Výsledný výber metódy simulácie testovacích dát závisí od konkrétnych požiadaviek jednotlivých testovacích prípadov.

4.3. Implementácia a exekúcia testov

Vzhľadom na rozsiahlosť testovacieho scenára, ktorý je príkladne popísaný v prílohe B - Metodická príručka, bol automatický test rozdelený na 8 častí:

- ERA2U_519_STEP1 – Základné zobrazenie cieľov,
- ERA2U_519_STEP2 – Label layout,
- ERA2U_519_STEP3 – Filtre a profily,
- ERA2U_519_STEP4 – Inšpektor a exportovanie dát,
- ERA2U_519_STEP5 – Export a import konfigurácie,
- ERA2U_519_STEP6 – Nahrávanie a kreslenie objektov,
- ERA2U_519_STEP7 – Profily, priority, warning DB,
- ERA2U_519_STEP8 – Offline okno.

Jednotlivé časti testu obsahujú sadu samostatných krokov (viď napr. tab. 8 - Filtre a profily). Každá časť automatického testu je nezávislá od ostatných, preto môže byť podľa potrieb vykonaná iba určitá podmnožina testov.

	Označenie a popis	Názov kroku	Použitá služba	
TEST PLAN - ERA2U_519	ERA2U_519_STEP1	Základné zobrazenie cieľov		
	ERA2U_519_STEP2	Label layout		
	ERA2U_519_STEP3	Filtre a profily		
	Predpoklady pre vykonanie kroku STEP3_1	Odstránenie nepotrebných licencií	Vymazanie licencie (AIS, DPET)	JMeter – OS Process Sampler
			Vymazanie licencie (EMPTY)	JMeter – OS Process Sampler
		Nastavenie licencie (AIS, DPET)	JMeter – OS Process Sampler	
		Spustenie aplikácie C2D	JMeter – OS Process Sampler, SikuliX	
		Spustenie simulátora ERA	JMeter – BeanShell Sampler	
		Vypnutie vrstvy – súradnicová sieť	JMeter – HTTP Request	
		Nastavenie mapy	JMeter – HTTP Request	
	Spustenie GUI simulátora	STEP3_1	JMeter – OS Process Sampler, SikuliX	
Predpoklady pre vykonanie kroku	Nastavenie mapy	JMeter – HTTP Request		

STEP3_2			
Spustenie GUI simulátora	STEP3_2		JMeter – OS Process Sampler, SikuliX
Predpoklady pre vykonanie kroku STEP3_1_AIS	Ukončenie aplikácie C2D		JMeter – OS Process Sampler, SikuliX
	Ukončenie simulátora ERA		JMeter – OS Process Sampler, SikuliX
	Spustenie aplikácie C2D		JMeter – OS Process Sampler, SikuliX
	Spustenie simulátora AIS		JMeter – BeanShell Sampler
	Vypnutie vrstvy – súradnicová sieť		JMeter – HTTP Request
	Nastavenie mapy		JMeter – HTTP Request
Spustenie GUI simulátora	STEP3_1_AIS		JMeter – OS Process Sampler, SikuliX
Predpoklady pre vykonanie kroku STEP3_2_AIS	Nastavenie mapy		JMeter – HTTP Request
Spustenie GUI simulátora	STEP3_2_AIS		JMeter – OS Process Sampler, SikuliX
Predpoklady pre ukončenie testu	Ukončenie aplikácie C2D		JMeter – OS Process Sampler, SikuliX
	Ukončenie simulátora AIS		JMeter – OS Process Sampler, SikuliX
ERA2U_519_STEP4	Inšpektor a exportovanie dát		
ERA2U_519_STEP5	Export a import konfigurácie		
ERA2U_519_STEP6	Nahrávanie a kreslenie objektov		
ERA2U_519_STEP7	Profily, priority, warning DB		
ERA2U_519_STEP8	Offline okno		
RESULT	Generovanie CSV reportu		JMeter – View Results Tree

Tab. 8 Príkladná štruktúra testu ERA2U_519_STEP3 – Filtre a profily

Tabuľka 8 zobrazuje časť navrhutej štruktúry automatického testu v súbore *TestPlan_C2D.jmx*, ktorý overuje funkčnosti filtrov a profilov pre UFE a AIS ciele. Jednotlivé kroky testu je možné rozdeliť do dvoch kategórií:

- **Predpoklady pre vykonanie a ukončenie testu** – kroky súvisiace s prípravou testovacieho prostredia.
- **Spustenie GUI simulátora** – kroky súvisiace s vykonávaním simulačných SikuliX skriptov v GUI aplikácii C2D.


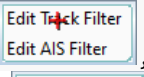
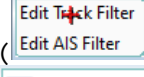
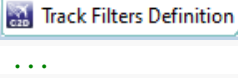
Pre efektívnejšiu údržbu testov a spracovanie výsledkov boli vytvorené knižnice:

- **Test_Failed.sikuli** – obsahuje funkcie pre spracovanie výsledkov v prípade výskytu chýb v testovanom systéme.

- **Configuration_settings.sikuli** – obsahuje funkcie pre simuláciu často sa opakujúcich krokov v testovanom systéme.

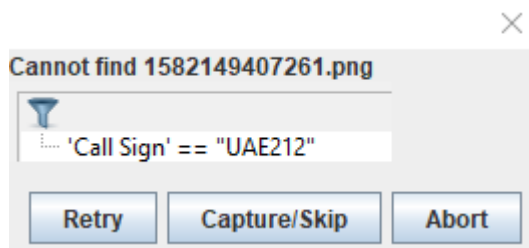
Aby boli jednotlivé testy vykonávané v rovnakej počítačovej konfigurácii (rozloženie mapy, vynulovanie filtrov, vynulovanie profilov atď.), je na začiatku každého testu (napr. STEP3_1.sikuli) importovaná predpripravená konfigurácia pomocou funkcie **set_default_configuration()**.

```

1. import re
2. import os
3. import shutil
4. from sikuli import *
5. import sys
6.
7. myScriptPath = os.path.dirname(getBundlePath()) + "\\..\Pomocne_metody"
8. if not myScriptPath in sys.path: sys.path.append(myScriptPath)
9. import Test_Failed
10. import Configuration_settings
11.
12. screen = Screen(0) # 0 primary monitor, 1 secondary monitor
13. screen.setFindFailedResponse(PROMPT) # want to be prompted if TestFailed
14. testCase_name = "ERA2U_519_STEP3_1"
15. testCase_path = getBundlePath()
16. testCase_dirname = os.path.dirname(getBundlePath())
17.
18. try:
19.     Configuration_settings.set_default_configuration()
20.     Settings.MoveMouseDelay = 0.8
21.     Configuration_settings.set_simulation_data("\\..\Pomocne_metody\Test_Simulat
or\generator_UFE\ADSB\20151018_140000_Cat248_ADSB.ufe")
22.     wait(10)
23.     if screen.exists()
25.         screen.rightClick()
26.         screen.wait(, 5)
27.         screen.click()
28.         screen.wait(, 5)
29.         # next steps ...
30.
31. except (IOError, NameError, FindFailed) as e:
32.     Test_Failed.error(testCase_name, testCase_path, e)

```

Ak si testovací prípad vyžaduje simuláciu cieľov, je prostredníctvom funkcie **set_simulation_data()** nastavená a spustená nahrávka, podľa potreby testu. V prípade ak dôjde k zlyhaniu testu, objaví sa dialógové okno (obr. 17), ktoré informuje testera o nenájdennom očakávanom objekte na obrazovke, nad ktorým má byť vykonaná určitá akcia.



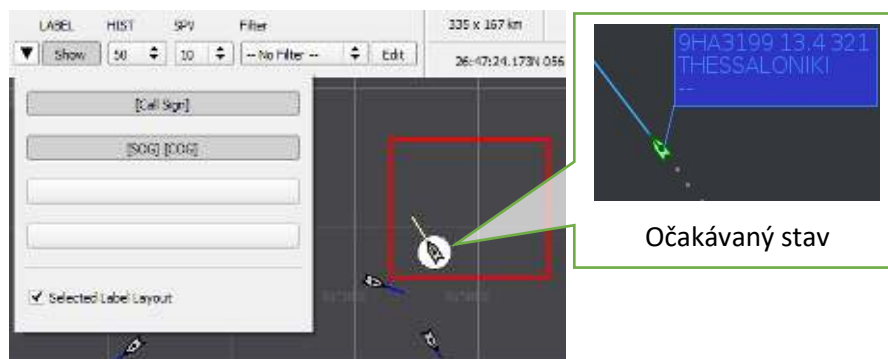
Obr. 17 Dialógové okno pri výskyte chyby

V takomto prípade má tester možnosť skúsiť akciu zopakovať, preskočiť-upraviť daný krok alebo prerušiť vykonávanie testu. Ak je prerušené vykonávanie testu, dôjde k vygenerovaniu a uloženiu aktuálnej snímky obrazovky v čase výskytu chyby a očakávaného hľadaného objektu (viď kap. 3.3.3. obr. 15).

4.4. Vyhodnotenie výsledkov automatických testov

Po spustení vytvorenej sady automatických systémových release testov *ERA2U-519* boli vo verzii 5.5.54468 aplikácie C2D zistené nasledujúce chyby:

- **V mapovom poli sa nezobrazuje štítok (label angl.) pre AIS cieľ** – V prípade, ak je aplikácia C2D spustená s AIS licenciou a obsahuje informácie o AIS cieľoch, nie je v mapovom okne zobrazený štítok s nakonfigurovanými položkami (obr. 18).



Obr. 18 Aktuálny a očakávaný stav AIS cieľa

- **Pád aplikácie pri nahrávaní súborov v offline okne** – V offline okne aplikácie C2D je prostredníctvom funkcie *Record Manager* načítaná a spustená prvá nahrávka (*c2d_ais_uae_2016-10-19-135917.rec*). Pri pokuse o načítanie ďalšieho súboru (*c2d_2016-10-14-082357.rec*) prostredníctvom tlačidla *Load all*, dôjde k pádu aplikácie.
- **Nesprávne zobrazenie zakresleného ArcBy3Points objektu** – Pri pokuse o úpravu ArcBy3Points objektu v mapovom poli, nie je objekt vykreslený podľa očakávania. Po opätovnom kliknutí na objekt nie je označený samotný objekt, ale iba jeho zvyšná časť

(obr. 19). Pri presunutí objektu na iné súradnice, sa zobrazuje už iba jeho zvyšná časť, ktorá by sa podľa očakávania nemala zobrazovať.



Obr. 19 Nesprávne zobrazenie zakresleného ArcBy3Points objektu

Po ukončení testovania bol do adresára [...\Pomocne_metody\Test_Results] automaticky vygenerovaný report vo formáte CSV (viď kap. 3.3.4.), prostredníctvom ktorého boli analyzované vyššie spomínané chyby. Tie boli následne reportované prostredníctvom systému pre správu chýb Mantis (viď kap. 2.4.).

5. Vyhodnotenie výsledkov metodiky

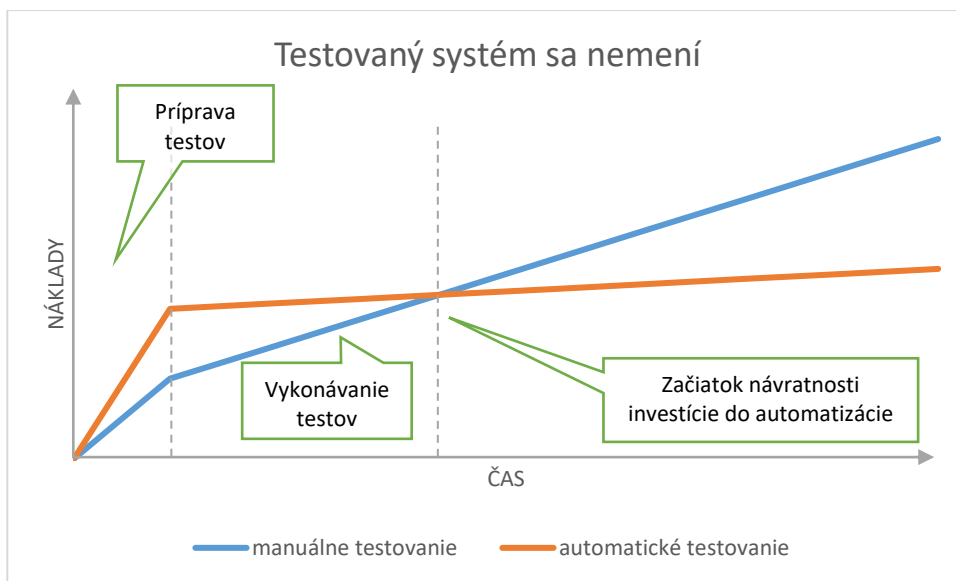
V procese experimentálneho aplikovania metodiky VSBT v projekte C2D, boli zistené určité obmedzenia a overené viaceré výhody automatického testovacieho systému.

Výhody	Obmedzenia
<ul style="list-style-type: none"> • Založený na použití voľne dostupných automatizačných nástrojov • Nezávislý od zdrojového kódu testovanej aplikácie • Použiteľný pre testovanie desktopových aj webových aplikácií • Umožňuje presné rozpoznávanie GUI objektov • Jednoduchosť použitia a adaptácie na testovaný systém • Modulárnosť – systém je možné rozširovať a upravovať podľa potrieb jednotlivých projektov • Podporovaný na platformách Windows, Mac, Linux • Umožňuje základnú prácu so súbormi v OS • Podporuje ovládanie GUI pomocou sekvencie HTTP požiadaviek a počítačového videnia 	<ul style="list-style-type: none"> • Počas testovania si akcie vyvolané SikuliX skriptami držia interakciu nad testovaným systémom (nie je možné na testovacom stroji vykonávať paralelne viacero činností) • Presnosť rozpoznávania GUI objektov je závislá od rozlíšenia obrazovky (odporúča sa vytvárať automatické testy pri rozlíšení 1920 × 1080 a viac) • Potrebná rozsiahla aktualizácia automatických testov pri zásadnej zmene GUI

Tab. 9 Výhody a obmedzenia automatického testovacieho systému

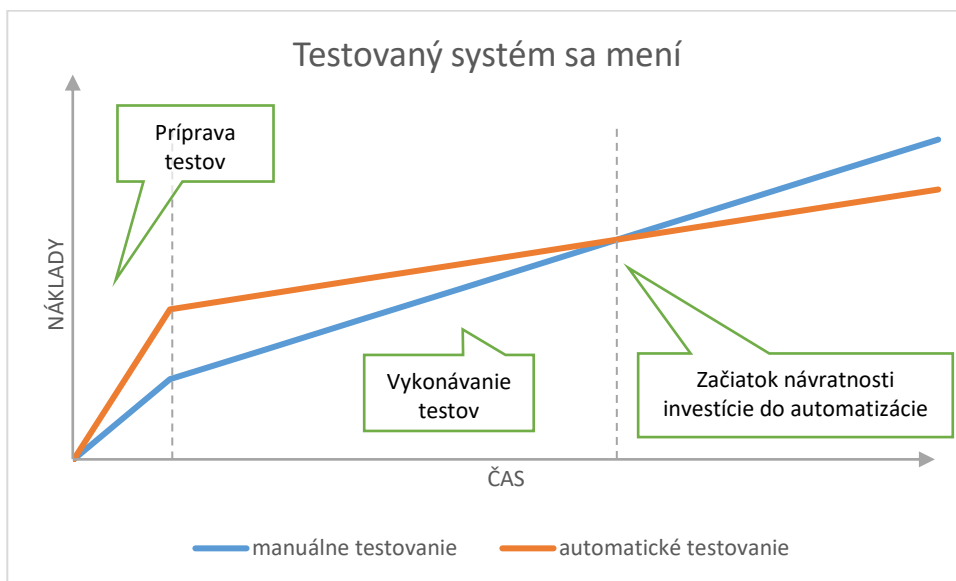
Keďže navrhnutý testovací systém využíva pri mapovaní techniku grafického rozpoznávania objektov na obrazovke, a teda je nezávislý od platformy a použitých technológií, bolo možné metodiku experimentálne aplikovať aj na iné systémy vyvíjané spoločnosťou R-SYS s.r.o. Konkrétne to bola webová aplikácia IXO [52], ktorá je v súčasnosti testovaná prostredníctvom techniky rozpoznávania HTML objektov a desktopová aplikácia ERA_PCM [53], ktorá na automatické testovanie využíva techniku sekvencie HTTP požiadaviek pomocou skriptov v jazyku Perl. V oboch prípadoch bol proces nasadenia metodiky a priebeh testovania bezproblémový. V prípade webovej aplikácie IXO sa v súčasnosti odporúča ponechať existujúci zaužívaný spôsob automatického testovania. Pre efektívnejšie a rozsiahlejšie pokrytie testovacích prípadov, môže byť existujúci testovací nástroj Katalon [33] rozšírený o použitie vizualizačných SikuliX skriptov, a to hlavne v semi-automatických testoch. V prípade aplikácií C2D a ERA_PCM, je vhodné pre jednoduchosť a lepšiu údržbu komplexne zjednotiť proces automatizácie testov prostredníctvom navrhnutej metodiky a nahradiť tým existujúce čiastkové riešenia.

Automatické testovanie metodikou VSBT stojí menej úsilia ako manuálne vykonávanie testovacích prípadov. V ideálnom prípade je návratnosť investície do automatického testovania už po niekoľkých opakovaníach testov (viď obr. 20). To platí hlavne pri automatických testoch, ktoré sú vykonávané nad nemennou časťou testovaného systému.



Obr. 20 Návratnosť investície do automatizácie (SUT sa nemení)

Presnejší odhad návratnosti investície do automatického testovania je zobrazený v grafe na obrázku 21, v ktorom sú zakomponované aj postprojektové fázy SLC, súvisiace s údržbou testovaného systému.

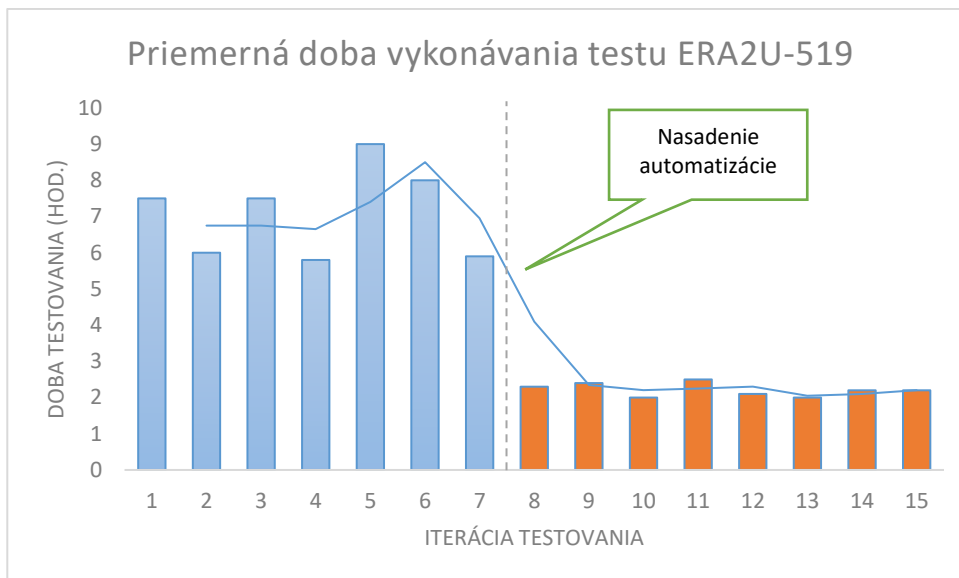


Obr. 21 Návratnosť investície do automatizácie (SUT sa mení)

Ako sa testovaný systém v priebehu životného cyklu mení, automatické testy je potrebné taktiež udržiavať v zhode s aktuálnou verziou testovaného systému. Tým pádom je návratnosť investície do automatického testovania o čosi dlhšia.

Doba manuálneho vykonania testovacieho prípadu *ERA2U-519* (viď príloha C) je v priemere 6 až 8 hodín. Keďže ide o špecifický test, ktorý obsahuje všeobecne popísané testovacie kroky, celková doba vykonania testu závisí od zložitosti a rozsahu testovaných atribútov. Aplikovaním

automatizácie metodikou VSBT (viď kap. 4.), sa priemerná doba vykonania testu skrátila na 2 až 2,5 hodín (obr. 22), čo z dlhodobého hľadiska predstavuje značnú úsporu zdrojov.



Obr. 22 Priemerná doba vykonania testu pred a po automatizácii

Vzhľadom na to, že je metodika VSBT založená na použití voľne dostupných automatizačných nástrojov, nie je potrebné pri jej zavádzaní do praxe navýšenie finančných nákladov na projekt. Technika ovládania GUI prostredníctvom počítačového videnia je náchylná na prípadné zmeny v používateľskom rozhraní testovanej aplikácie. Preto je potrebné brať do úvahy, že každá zmena dizajnu alebo úprava GUI komponentov si bude vyžadovať aktualizáciu testovacích skriptov. Výsledný čas a kvalita vytvorených testov závisí od počtu a skúseností automatizovaných testerov. Dôležitá je pritom znalosť štruktúrovania testov, mapovanie jednotlivých GUI objektov, ako aj skúsenosti s implementáciou samotných vizualizačných SikuliX skriptov v jazyku Jython.

Záver

Cieľom tejto diplomovej práce bolo definovať metodiku automatického testovania, ktorú je možné vo všeobecnosti aplikovať na desktopové aplikácie s grafickým používateľským rozhraním v OS Windows 10. Návrhu metodiky predchádzala analýza existujúcich komerčných a voľne dostupných testovacích nástrojov, primárne určených pre desktopové aplikácie. Z vyhodnotenia použiteľnosti jednotlivých testovacích nástrojov vyplynulo, že v súčasnosti nie je k dispozícii také riešenie, ktoré by komplexne pokrývalo testovacie požiadavky pre systémy vyvíjané spoločnosťou R-SYS s.r.o. Vzhľadom na zložitosť týchto systémov bol navrhnutý model integrácie viacerých testovacích nástrojov a mechanizmov, ktorý je možné aplikovať nezávisle od použitia GUI technológie v jednotlivých testovaných aplikáciách.

Na základe znalostí a skúseností s použitými automatizačnými nástrojmi bol vytvorený metodický postup, ktorý v konkrétnych krokoch popisuje aplikovanie metodiky VSBT v projekte. Metodika bola experimentálne overená v prostredí aplikácie C2D, určenej pre zobrazenie vzdušnej situácie nad mapovým podkladom na základe informácií prijímaných z rôznych radarových senzorov. Proces automatického testovania spočíval v paralelnom vykonávaní procesov v nástrojoch JMeter, SikuliX a interných simulačných nástrojoch, s presne určenou časovou postupnosťou, ktorá zodpovedala pripravenému testovaciemu plánu. Automatické testovacie skripty boli odvodené z manuálne vytvorených testovacích prípadov popísaných v systéme TestLink. Testovaním boli zistené viaceré chyby funkčnosti aplikácie C2D, ktoré boli následne reportované prostredníctvom systému Mantis.

Navrhnutá metodika automatického testovania sa od štandardných postupov odlišuje aplikovaním automatizačných nástrojov využívajúcich počítačové videnie. Tieto nástroje sa výrazne líšia prístupom k danej problematike, na rozdiel od nástrojov využívajúcich zdrojový kód testovanej aplikácie. Možným predmetom ďalšieho skúmania je využitie počítačového videnia v súčasných semi-automatických testoch webových aplikácií, čo by mohlo viesť k zvýšeniu pokrytia a efektivity vykonávania automatických testov.

Zoznam použitej literatúry

- [1]. PAGE Alan; ROLLISON, B; JOHNSTON Ken. Jak testuje software Microsoft. Computer Press, Albatros Media as, 2017. ISBN 978-80-251-2869-5.
- [2]. POULOVA Petra; KLIMOVA Blanka. *Automated Software Testing—A Case Study* [online]. Advanced Science Letters, 2018, 24.4: 2578-2581 [cit. 2019-3-2]. Dostupné na internete: <<http://ptgmedia.pearsoncmg.com/images/9780321754066/samplepages/0321754069.pdf>>.
- [3]. GURU99.com. Automation testing Tutorial: Process, Planning & Tools [online]. 2017 [cit. 2019-3-2]. Dostupné na internete: <<https://www.guru99.com/automation-testing.html>>.
- [4]. SMARTSHEET.com. Automated Software Testing Isn't Automatic: Introduction to the Basics [online]. 2018 [cit. 2019-3-2]. Dostupné na internete: <<https://www.smartsheet.com/automation-testing-software>>.
- [5]. CROSSBROWSETESTING.com. Moving From Manual to Automated Testing [online]. [cit. 2019-3-5]. Dostupné na internete: <<https://crossbrowsertesting.com/crossbrowsertesting/media/pdfs/moving-from-manual-to-automated-testing.pdf>>.
- [6]. SIMPSON Jim; WISNOWSKI Jim. Automated Software Testing Implementation Guide [online]. STAT T&E Center of Excellence, 2017 [cit. 2019-3-5]. Dostupné na internete: <https://www.afit.edu/stat/statcoe_files/Automated_Software_Testing_Implementation_Guide.pdf>.
- [7]. BUREŠ Miroslav, RENDA Miroslav; DOLEŽEL Michal. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Grada Publishing as, 2016. ISBN: 978-80-247-5594-6.
- [8]. ROUDENSKÝ Peter; HAVLÍČKOVÁ Anna. Řízení kvality softwaru. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-3816-8.
- [9]. SOFTSERVEINC.com. Fundamental test process [online]. 2014 [cit. 2019-3-10]. Dostupné na internete: <<https://en.ppt-online.org/38967/>>.
- [10]. TRYQA.com. What is fundamental test process in software testing? [online]. 2017 [cit. 2019-3-20]. Dostupné na internete: <<http://tryqa.com/what-is-fundamental-test-process-in-software-testing/>>.
- [11]. GLOSSARY.istqb.org. Regression testing [online]. 2018 [cit. 2019-5-4] Dostupné na internete: <<https://glossary.istqb.org/en/term/regression-testing-1>>.

- [12]. SOFTWARETESTINGCLASS.com. Difference between Regression Testing vs Retesting? [online]. 2013 [cit. 2019-5-04] Dostupné na internete:
<<https://www.softwaretestingclass.com/difference-between-regression-testing-vs-retesting/>>.
- [13]. BESSONOVA Tatiana. Regression Testing Strategies: an Overview [online]. 2018 [cit. 2019-5-4]. Dostupné na internete:
<<https://www.infoq.com/articles/regression-testing-strategies>>.
- [14]. RANOREX.com. Regression Testing Guide [online]. [cit. 2019-5-5]. Dostupné na internete:
<<https://www.ranorex.com/resources/testing-wiki/regression-testing/>>.
- [15]. SOLANSKI Kamna Jyoti. A Comparative Study of Five Regression Testing Techniques: A Survey [online]. International journal of scientific & technology research volume 3, August 2014 [cit. 2019-5-5]. Dostupné na internete:
<<https://www.ijstr.org/final-print/aug2014/A-Comparative-Study-Of-Five-Regression-Testing-Techniques-A-Survey.pdf>>.
- [16]. SOFTWARETESTINGHELP.com. Regression Testing Complete Guide: Tools, Method, and Example [online]. April 2019 [cit. 2019-5-5]. Dostupné na internete:
<<https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/>>.
- [17]. ALSHEUSKI Viktor. Regression testing example [online]. ScienceSoft 2017. [cit. 2019-6-10]. Dostupné na internete:
<<https://www.scnsoft.com/blog/regression-testing-example>>.
- [18]. McPEAK Alex. What is Regression Testing? [online]. CrossBrowserTesting 2017 [cit. 2019-6-10]. Dostupné na internete:
<<https://crossbrowsertesting.com/blog/test-automation/regression-testing-automation/>>.
- [19]. KENT John. Test Automation: From Record/Playback to Frameworks [online]. 2014 [cit. 2019-6-14]. Dostupné na internete:
<<http://www.simplytesting.com/software-testing-papers/Test%20Automation%20-%20From%20Record-Playback%20To%20Frameworks.pdf>>.
- [20]. GURU99.com. Test Automation Frameworks - Stuff you must Know! [online]. [cit. 2019-6-14]. Dostupné na internete:
<<https://www.guru99.com/test-automation-framework.html>>.
- [21]. SOFTWARETESTINGHELP.com. Most Popular Test Automation Frameworks with Pros and Cons of Each [online]. 2019 [cit. 2019-6-14]. Dostupné na internete:
<<https://www.softwaretestinghelp.com/test-automation-frameworks-selenium-tutorial-20/>>.

- [22]. R-SYS s.r.o. Spracovanie Meteo – Interná projektová dokumentácia SESAR_PJ014 [online]. 2019. [cit. 2019-6-14]. Dostupné z:
<http://wiki.irsys.sk/index.php/SESAR_PJ14:FS1:Funkcie:Meteo>.
- [23]. KOKINDA Matúš. Použitie modelov a prototypov pre testovanie softvéru [online]. [cit. 2019-6-21]. Dostupné na internete:
<<https://opac.crzp.sk/?fn=detailBiblioForm&sid=5301BA0C1F12FE31B44E4CA915FD>>.
- [24]. TUTORIALSPPOINT.com. Software Testing Tutorial [online]. [cit. 2019-6-23]. Dostupné na internete:
<https://www.tutorialspoint.com/software_testing/software_testing_tutorial.pdf>.
- [25]. PATTON Ron. Testovanie softvéru. Computer Press 2017 [cit. 2019-6-23]. ISBN 80-7226-636-5.
- [26]. SPILLNER Andreas. The W-model Strengthening the Bond Between Development and Test [online]. STAReast 2002. [cit. 2019-6-26]. Dostupné na internete:
<<https://www.cmcrossroads.com/sites/default/files/article/file/2014/The%20W%20Model%20Strengthening%20the%20Bond%20Between%20Development%20and%20Test.pdf>>.
- [27]. GOLDSMITH F. Robin, GRAHAM Dorothy. The Forgotten Phase [online]. Sdmagazine.com 2002. [cit. 2019-6-26]. Dostupné na internete:
<<https://www.cs.du.edu/~snarayan/sada/teaching/COMP3705/lecture/p1/07sd.Gold45-47.pdf>>.
- [28]. SOMMERVILLE Ian. Software engineering 9th Edition [online]. Edwards Brothers 2011. [cit. 2019-6-26]. Dostupné na internete:
<https://edisciplinas.usp.br/pluginfile.php/2150022/mod_resource/content/1/1429431793.203Software%20Engineering%20by%20Somerville.pdf>.
- [29]. BECK Kent. Test-Driven Development By Example [online]. Three Rivers Institute 2002. [cit. 2019-6-26]. Dostupné na internete:
<https://www.eecs.yorku.ca/course_archive/2003-04/W/3311/sectionM/case_studies/money/KentBeck_TDD_byexample.pdf>.
- [30]. AUTOIT.com. Freeware BASIC-like scripting language designed for automating the Windows GUI [online]. [cit. 2019-8-30]. Dostupné na internete:
<<https://www.autoitscript.com/site/autoit/>>.
- [31]. R-SYS s.r.o. C2D:TEST:TestInterface – Interná projektová dokumentácia projektu C2D [online]. 2019. [cit. 2019-8-30]. Dostupné z:
<<http://wiki.irsys.sk/index.php?title=C2D:TEST:TestInterface&oldid=42561>>.

- [32]. JMETER.APACHE.org. The Apache JMeter™ open source software [online]. [cit. 2019-8-30]. Dostupné na internete:
<<https://jmeter.apache.org/>>.
- [33]. KATALON.com. Katalon Studio: Free automation testing solution developed by Katalon LLC [online]. [cit. 2019-8-31]. Dostupné na internete:
<<https://www.katalon.com/>>.
- [34]. SELENIUMHQ.org. Selenium: Browser automation [online]. [cit. 2019-8-31]. Dostupné na internete:
<<https://www.seleniumhq.org/>>.
- [35]. SMARTBEAR.com. TestComplete: Automated UI Testing Tool [online]. [cit. 2019-8-31]. Dostupné na internete:
<<https://smartbear.com/product/testcomplete/overview/>>.
- [36]. SIKULIX.com. SikuliX: Automate what you see on a computer monitor [online]. [cit. 2019-8-31]. Dostupné na internete:
<<http://www.sikulix.com/>>.
- [37]. MANTIS.org. Mantis – open source bug tracker [online]. [cit. 2019-9-19]. Dostupné na internete:
<<https://www.mantisbt.org/index.php>>.
- [38]. ROSEN Chad and TestLink community. TestLink - User manual. TestLink Community 2010. Dostupné na internete:
<https://wiki.openoffice.org/w/images/1/1b/Testlink_user_manual.pdf>.
- [39]. R-SYS s.r.o. ERA C2D UPG 5 v5.X – Interná dokumentácia projektu C2D [online]. Michal Vlasák 2019. [cit. 2019-10-5]. Dostupné z:
<http://10.10.100.1/home/Projects/ERA_C2D_UPG51/FS0/FS0_C2D-07119-01-RFR_Upgrade_V5.X.pdf>.
- [40]. FROGLOGIC.com. Squish for Windows - Automated GUI Testing for Windows Applications [online]. [cit. 2019-10-10]. Dostupné na internete:
<<https://www.froglogic.com/squish-for-windows/>>.
- [41]. EASYEXEL.sk. Makrá Excel [online]. [cit. 2019-10-10]. Dostupné na internete:
<<https://www.easyexcel.sk/kurz-excel-online/makra-excel/>>.
- [42]. QT.io. Cross-platform software development for embedded systems [online]. [cit. 2019-10-18]. Dostupné na internete:
<<https://www.qt.io/>>.
- [43]. AUTOITSCRIPT.com. AutoIt Tools [online]. [cit. 2019-10-18]. Dostupné na internete:
<<https://www.autoitscript.com/site/autoit-tools/>>.

- [44]. MALLETT Chris. AutoHotkey - The ultimate automation scripting language for Windows [online]. [cit. 2019-10-18]. Dostupné na internete:
<<https://www.autohotkey.com/>>.
- [45]. Sikuli Doc Team. How Sikuli Works [online]. Sikuli X 1.0 documentation 2012. [cit. 2019-10-19]. Dostupné na internete:
<<http://doc.sikuli.org/devs/system-design.html>>.
- [46]. OpenCV team. OpenCV - open source computer vision and machine learning software library [online]. 2019. [cit. 2019-10-20]. Dostupné na internete:
<<https://opencv.org/about/>>.
- [47]. HOCKE Raimund. SikuliX version 2.0.0+ (2019+) - Sources on GitHub [online]. SikuliX1 2019. [cit. 2019-10-21]. Dostupné na internete:
<<https://github.com/RaiMan/SikuliX1>>.
- [48]. THE TORTOISESVN TEAM. An Apache™ Subversion (SVN)® client [online]. TortoiseSVN 2019. [cit. 2019-11-30]. Dostupné na internete:
<<https://tortoisesvn.net/>>.
- [49]. Nodejs.org. JavaScript runtime built on Chrome's V8 JavaScript engine [online]. [cit. 2019-12-28]. Dostupné na internete:
<<https://nodejs.org/en/>>.
- [50]. Activestate.com. Perl From ActiveState [online]. [cit. 2019-12-28]. Dostupné na internete:
<<https://www.activestate.com/products/perl/>>.
- [51]. R-SYS s.r.o. C2 Display – Uživatelská príručka (Interná projektová dokumentácia projektu C2D) [online]. 2018. [cit. 2020-2-9]. Dostupné z:
<http://www.irsys.sk/home/Projects/ERA_C2D_UPG51/Release/doc/c2d_v5_2_3790_Use_r_manual_sk_prepreklad.pdf>.
- [52]. IXOSYSTEM.com. UTM system for RPAS users, IFR/VFR pilots and ANSP authorities [online]. [cit. 2020-3-15]. Dostupné na internete:
<<https://ixosystem.com/>>.
- [53]. R-SYS s.r.o. ERA PCM (Interná projektová dokumentácia projektu ERA PCM) [online]. 2020. [cit. 2020-3-15]. Dostupné z:
<http://wiki.irsys.sk/index.php/ERA_PCM>.
- [54]. ERA.aero. VERA-NG – pasívny sledovací a identifikačný systém [online]. 2020. [cit. 2020-4-9]. Dostupné na internete:
<<https://www.era.aero/cs/military-security/vera-ng>>.

Prílohy

Príloha A: CD médium so súbormi:

- diplomová práca v elektronickej podobe,
- prílohy v elektronickej podobe,
- vedecký článok v elektronickej podobe,
- príklady automatických testov.

Príloha B: Metodická príručka

Príloha C: Testovacie prípady

Príloha D: Popis interného ladiaceho rozhrania C2D

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

**APLIKOVANIE NÁSTROJOV AUTOMATICKÉHO
TESTOVANIA DESKTOPOVÝCH APLIKÁCIÍ S GRAFICKÝM
ROZHRANÍM**

Diplomová práca

Príloha B
Metodická príručka

Obsah

Zoznam obrázkov	3
Zoznam tabuliek	4
Zoznam symbolov a skratiek	5
1. Popis metodiky automatického testovania VSBT.....	6
1.1. Analýza rizík a cieľov pri nasadení automatického testovania	6
1.1.1. Riziká spojené s automatizáciou	7
1.1.2. Ciele pri nasadení automatického testovania	7
1.1.3. Techniky a stratégie regresného testovania	8
1.2. Príprava testovacieho prostredia	10
1.3. Prepísanie testovacích prípadov do testovacích skriptov	12
1.3.1. Vyhodnotenie SikuliX skriptov.....	16
1.4. Generovanie reportov a údržba testov v postprojektových fázach.....	17
Zoznam použitej literatúry	20

Zoznam obrázkov

Obr. 1 Procesný model automatického testovania.....	6
Obr. 2 Stúpajúci trend nárastu regresných testov [5].....	9
Obr. 3 Konfigurácia ciest v komponente C2D_TEST_PLAN_CONFIG	11
Obr. 4 Definovanie parametrov v HTTP Request Defaults a HTTP Header Manager	12
Obr. 5 Nastavenie časovača a odstránenie licencií	13
Obr. 6 Nastavenie licencie a spustenie aplikácie	13
Obr. 7 Vypnutie súradnicovej siete a nastavenie mapy.....	14
Obr. 8 Spustenie GUI simulátora.....	15
Obr. 9 Vyhodnotenie SikuliX skriptu	17
Obr. 10 Generovanie reportu.....	17
Obr. 11 Dialógové okno pri výskyte chyby	18

Zoznam tabuliek

Tab. 1 Test Case ERA2U_519 – krok 3.1.....	14
--	----

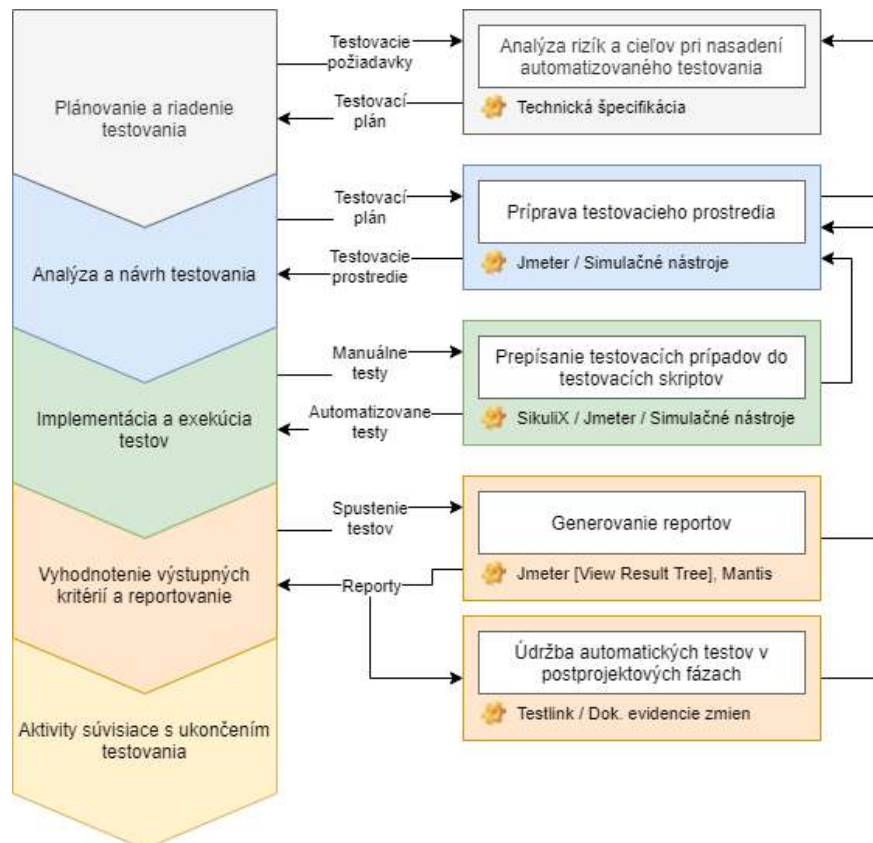
Zoznam symbolov a skratiek

AIS	Automatic Identification System (automatický identifikačný systém)
API	Application programming interface (rozhranie pre programovanie aplikácií)
CSV	Comma-Separated Values (hodnoty oddelené čiarkami)
GUI	Graphical User Interface (grafické používateľské rozhranie)
HTTP	HyperText Transfer Protocol (hypertextový prenosový protokol)
OS	Operating system (operačný systém)
REST	Representational State Transfer (reprezentačný prenos stavu)
UFE	Unified Format ERA (jednotný formát ERA)
URI	Uniform Resource Identifier (jednotný identifikátor prostriedku)
VPN	Virtual Private Network (virtuálna privátna sieť)
VFR	Visual Flight Rules (pravidlá letu za viditeľnosti)
VSBT	Visual Script Based Testing (testovanie založené na vizualizačných skriptoch)

1. Popis metodiky automatického testovania VSBT

V nasledujúcich podkapitolách sú popísané konkrétne kroky metodiky VSBT (*Visual Script Based Testing* angl.). Jednotlivé kroky sú doplnené o príkladné ukážky demonštrované v procese automatizácie systémových testov aplikácie C2D.

Metodika vychádza z procesného modelu automatického testovania (obr. 1), v ktorom sú definované základné testovacie aktivity a postupnosť ich vykonávania.



Obr. 1 Procesný model automatického testovania

Model je rozdelený na dve časti. Ľavá časť diagramu reprezentuje základný testovací proces. V pravej časti naňho nadväzujú jednotlivé fázy metodiky, ktoré sú doplnené o vstupy, výstupy a použité nástroje.

1.1. Analýza rizík a cieľov pri nasadení automatického testovania

Riziká a ciele automatického testovania vychádzajú z testovacích požiadaviek na systém. Na základe dostupnej technickej špecifikácie je možné zistiť či je systém vhodný k automatizácii, aké nástroje a utility majú byť použité pri automatizácii a aká bude reálna úspora zdrojov.

1.1.1. Riziká spojené s automatizáciou

Pri úvahe automatizovať testovací proces je potrebné počítať s rizikami, ktoré môžu negatívne ovplyvniť vývoj celého projektu. Ešte pred samotnou aplikáciou automatického testovania je preto vhodné odpovedať na nasledujúce otázky [1]:

- „*Je systém vhodný k automatizácii?*“ – Systém vhodný pre testovanie automatizovanými nástrojmi by mal byť stabilný hlavne v zmysle kódu. Pokiaľ je automatizácia zvolená ako primárny spôsob testovania, je vhodné, aby sa od začiatku až do konca vývoja pracovalo s rovnakou štruktúrou objektov, ich metódami a názvami. Čím vývojári používajú obvyčajnejšie riešenia, tým je automatizácia jednoduchšia. Napríklad negatívny vplyv na automatizáciu by mohlo mať použitie netradičného spôsobu ovládania systému.
- „*Aké nástroje použiť pri automatizácii?*“ – O výbere testovacích nástrojov by mali rozhodovať osoby, ktoré sa priamo podieľajú na testovaní. Voľba nástrojov sa predovšetkým musí odvíjať od technologických požiadaviek. Je preto vhodné, aby boli ešte pred samotným zaobstaraním vykonané technologické testy testovacieho nástroja na konkrétnom systéme, ktorý má byť automatizovaný [2]. Pri menších projektoch je vhodnejšie použiť niektoré z voľne dostupných riešení, prípadne je možné automatizáciu riešiť prostredníctvom vlastných nástrojov určených priamo pre testovaný systém.
- „*Aká bude reálna úspora zdrojov pri automatizácii?*“ – Úspora zdrojov je zvyčajne hlavným argumentom pre zavedenie automatizácie. To ale neznamená, že zavedenie automatizácie dokáže efektívne nahradiť niekoľko členný tím testerov. So zavedením automatizovaných nástrojov sa mení časová náročnosť prípravy testov, ako aj kvalifikačné predpoklady a znalosti testerov [8]. Z toho vyplývajú vyššie náklady na personál a celkovú prípravu testov. Hlavný benefit spočíva v opakovanom použití automatických testov. Práve to so sebou prináša vyššiu efektivitu a najväčšie úspory zdrojov.

1.1.2. Ciele pri nasadení automatického testovania

Rôzne prístupy a techniky v testovaní softvéru zohľadňujú rôzne ciele. Vo vývojovej fáze je hlavným cieľom testovania zistenie čo najväčšieho počtu chýb, aby zistené defekty mohli byť identifikované a opravené čo najskôr. V akceptačnom testovaní je hlavným cieľom overenie, že systém pracuje podľa očakávaní a v súlade s požiadavkami. Medzi ďalšie ciele môže patriť získanie informácií o rizikách uvoľnenia systému v danom čase, prípadne zhodnotenie spoľahlivosti alebo dostupnosti [2]. Automatické testovanie sa snaží tieto čiastkové ciele v procese testovania softvéru urýchliť a čo najviac zefektívniť.

- Zvýšenie dôvery v kvalitu softvéru – Hoci pri testovaní nie je možné overiť všetky stavy systému, obsiahnutie čo najväčšej časti funkcionality testovacími prípadmi zaručuje jeho kvalitu [4]. Úspešné vyhodnotenie všetkých testov poukazuje iba na to, že v aktuálnych testovacích prípadoch nenastali žiadne chyby. Nájdenie ďalších potenciálnych chýb závisí od kreativity testera. Preto je potrebné dbať aj na kvalitu a kontrolu testov. Zle navrhnutý test, ktorý neodhalí chybu hneď na prvýkrát, ju neodhalí zakaždým, keď bude automatizovane spustená celá testovacia sada.
- Rýchlejšie vydanie softvéru – Včasné objavenie chýb automatickými testami umožňuje rýchlejšie opravenie a nasadenie novej verzie systému. Kritické môžu byť technické problémy spojené s testami, spôsobené napríklad zmenami v aplikácii alebo chybami v testovacom nástroji. Tieto komplikácie môžu mať za následok presne opačný negatívny efekt, a to predĺženie doby dodania softvéru klientovi [2].
- Konzistentné opakovateľné testovanie – Oproti manuálnemu testovaniu, kedy tester môže pri testovaní zle simulovať jednotlivé kroky v testovanom systéme, sa automatické testy vždy striktnie držia testovacieho scenára [3]. Naopak, tester sa pri manuálnom testovaní dokáže vysporiadať s komplexnosťou a výsledkami, ktoré by automatický test nedokázal správne vyhodnotiť.
- Znovupoužiteľnosť – Existujúce postupy, nástroje a technológie použité pri automatizácii je možné aplikovať aj pri ďalších podobných projektoch. S ich častejším využívaním exponenciálne rastú skúsenosti a efektívnosť pri tvorbe budúcich testovacích skriptov.
- Vykonávanie testov bez osobnej prítomnosti – Testy môžu byť spúšťané v denných, týždenných či dokonca mesačných intervaloch na vzdialenom serveri. Tester v tom prípade nezasahuje do priebehu testu a analyzuje iba výsledné hodnotenie testov [3].

1.1.3. Techniky a stratégie regresného testovania

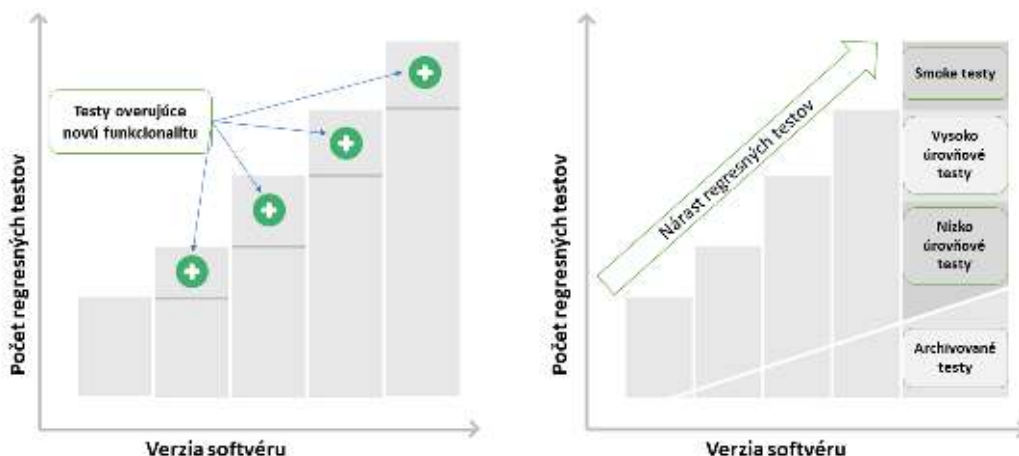
S vyššou komplexnosťou vyvíjaného softvéru rastie aj počet testovacích prípadov, ktoré overujú jeho funkčnosť. Ak je softvér vo vyššom štádiu vývoja alebo postprojektovej fáze, pretestovanie všetkých doposiaľ vytvorených testovacích prípadov môže mať negatívny dopad na časový a finančný plán projektu [5]. Aby sa zabránilo neprimeranému navyšovaniu regresných testovacích prípadov, boli zavedené techniky a stratégie pre ich optimalizáciu [6].

- **Kompletná regresia** – Ide o nákladnú techniku regresného testovania, ktorá si vyžaduje dostatočné množstvo času a financií. Táto technika je tiež známa pod pojmom „Retest all“, pri ktorej sú vykonané všetky testovacie prípady softvéru. Využíva sa najmä v projektoch, kde je najvyššou prioritou kvalita softvéru bez ohľadu na financie a čas. Komplexnú regresiu

je taktiež vhodné použiť pri lokalizačných úpravách softvéru, ktoré súvisia napríklad so zmenou jazyka [5].

- **Redukcia regresných testov** – Pri tejto technike sa vykoná len určitá časť testovacej sady. Testy sú vyberané na základe konkrétneho modulu alebo funkcie, v ktorej bola vykonaná zmena. Napríklad, ak je v systéme pridaný nový typ platby, budú vykonané iba tie regresné testy, ktoré súvisia priamo s platobným procesom. Testy súvisiace s inými procesmi, napríklad vyhľadávanie položiek alebo registrácia, budú vylúčené.
- **Priorizácia testovacích prípadov** – Táto technika uprednostňuje určité testovacie prípady pred ostatnými na základe priority. To znamená, že testovacím prípadom, ktoré pokrývajú kritické a často používané funkcie je priradená väčšia priorita, ako testovacím prípadom, ktoré pokrývajú menej dôležité funkcie. Existujú dva typy priorít:
 - Všeobecné – Testovacie prípady sú kategorizované na základe toho, ako sú prospešné pre nasledujúce verzie softvéru.
 - Špecifické – Testovacie prípady sú kategorizované na základe konkrétnej verzie softvéru.
- **Hybridná technika** – Hybridný prístup zlučuje techniky redukcie regresných testov a priorizáciu testovacích prípadov [7].

Ako zobrazuje obrázok 5, v každej novej verzii softvéru je vytvorená nová testovacia sada, ktorá overuje novú funkcionálnosť systému. Tým pádom lineárne narastá celkový počet regresných testov.



Obr. 2 Stúpajúci trend nárastu regresných testov [5]

Aby bolo možné zmierniť tento stúpajúci trend, je potrebné aplikovať vhodnú stratégiu optimalizácie regresných testov. Stratégia môže využívať viaceré techniky, s cieľom identifikovať prioritu, aby sa zabezpečilo vykonanie najkritickejších testovacích prípadov.

1. Ako prvý krok pri optimalizácii je archivácia zastaralých testov, ktoré už neslúžia svojmu účelu [5]. Môžu to byť testovacie prípady súvisiace s funkčnosťou, ktorá už bola zo systému odstránená.
2. Druhým krokom je identifikácia testov, ktoré overujú špecifické požiadavky na softvér. Takéto testovacie prípady je možné archivovať a vynechať z regresného testovania alebo ich je možné zlúčiť do všeobecnejšieho testovacieho prípadu [3]. Taktiež je potrebné odstránenie duplicity medzi jednotlivými testami.
3. Tretím krokom je identifikácia tých testovacích prípadov, ktoré overujú kritickú funkčnosť systému alebo pokrývajú tie časti systému, v ktorých hrozí najväčšie riziko zlyhania. Týmto testovacím prípadom je možné priradiť vyššiu prioritu. Nižšiu prioritu majú negatívne testovacie prípady a testy overujúce hraničné prípady.
4. V poslednom kroku je potrebné archivovať všetky testy, ktoré zlyhali z dôvodu existujúcich chýb a neboli doposiaľ vyriešené. Archivovať je užitočné aj testy s nižšou prioritou, ktoré neodhalili žiadne chyby v niekoľkých iteráciách regresného testovania [5].

1.2. Príprava testovacieho prostredia

Príprava testovacieho prostredia si vyžaduje stiahnutie a inštaláciu dvoch voľne dostupných automatizačných nástrojov:

- Apache JMeter [8] (aktuálne vo verzii 5.2.1)
- SikuliX [9] (aktuálne vo verzii 2.0.4)

Tieto nástroje môžu byť doplnené o ďalšie podporné simulačné nástroje alebo simulačné skripty, v závislosti od testovacích požiadaviek na systém. Spôsob, akým prebieha generovanie testovacích dát, zostáva otvorený, vďaka čomu je možné použiť rôzne prístupy k ich simulácii.

V prípade aplikácie C2D budeme využívať simulačné skripty generujúce testovacie dáta, ktoré sú implementované v jazyku Perl a JavaScript. Tým pádom je potrebné stiahnuť a nainštalovať prostredie pre ich spúšťanie:

- Node.js [10] (aktuálne vo verzii 13.12.0)
- ActivePerl [11] (aktuálne vo verzii 5.28)

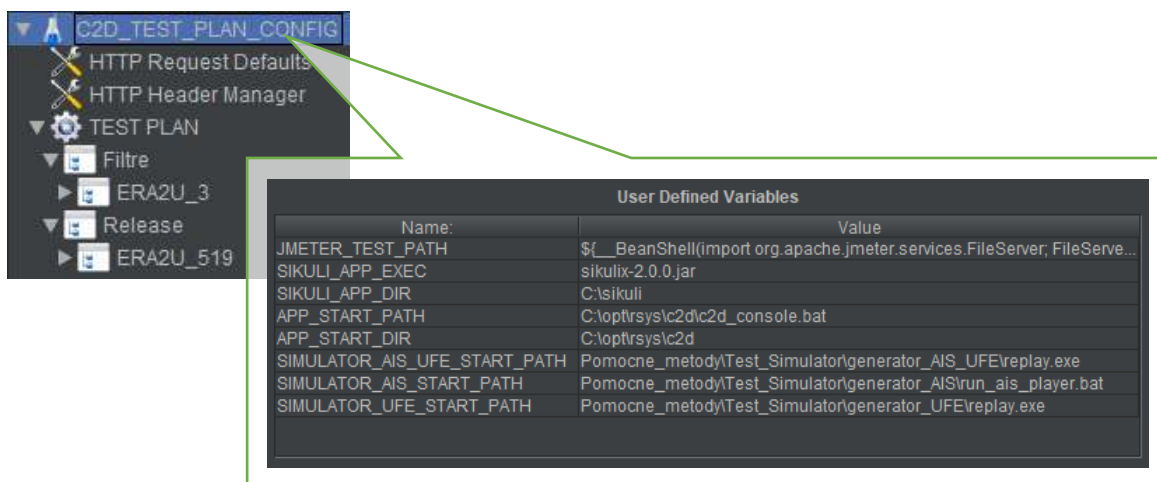
Ďalším krokom je inštalácia samotnej testovanej desktopovej aplikácie a všetkých jej potrebných komponentov. Po nainštalovaní aplikácie C2D sa na pracovnej ploche OS zobrazí jej

ikona. Súčasne s aplikáciou C2D sa nainštaluje aj aplikácia C2D Offline Editor, ktorá slúži na konfigurovanie a nastavenie vlastností aplikácie C2D (hlavná konfigurácia aplikácie). Spustíme aplikáciu C2D Offline Editor ako správca. Pre korektný príjem testovacích dát zo simulačných nástrojov vykonáme základnú konfiguráciu zadefinovaním nasledujúcich parametrov:

- root -> c2d -> Common: Povolený **TestInterfacePort** s portom 8090
- root -> c2d -> DataChannels: Pridaný **TrackChannel[1]**
 - (Name: TEST1, Address: 127.0.0.1, Port: 50050, Category: FLOW)
- root -> c2d -> DataChannels: Pridaný **TrackChannel[2]**
 - (Name: TEST2, Address: 127.0.0.1, Port: 50000, Category: CAT 248 v1.15)
- root -> c2d -> DataChannels: Pridaný **AISChannel[1]**
 - (Name: TEST3, Address: 127.0.0.1, Port: 10001, ConnectionType: UDP)

Uložením tejto konfigurácie sa v aplikácii C2D aktivujú kanály pre príjem AIS, UFE a AIS-UFE dát. Testovacie dáta sú generované zo simulačných skriptov a nástrojov dostupných v priloženom adresári [..\Projekt_C2D\Pomocne_metody\Test_Simulator].

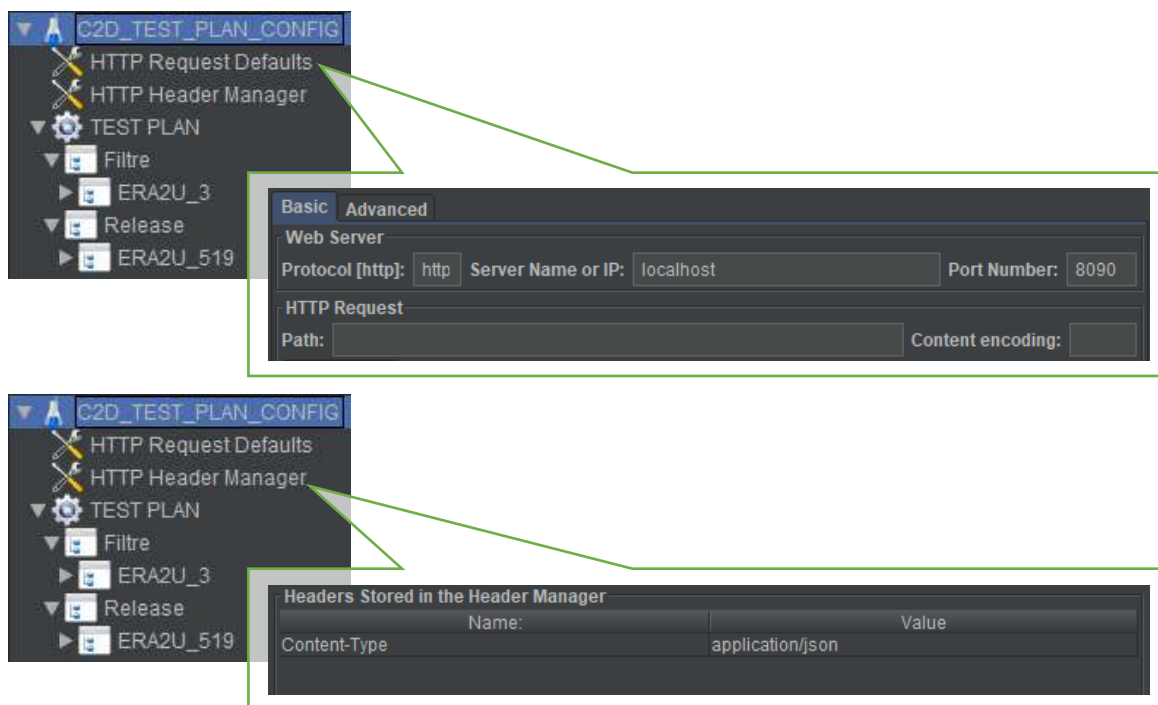
V nástroji JMeter si otvoríme priložený súbor [..\Projekt_C2D\TestPlan_C2D.jmx], ktorý obsahuje príkladný testovací plán určený pre aplikáciu C2D. V komponente **C2D_TEST_PLAN_CONFIG** (obr. 3) nakonfigurujeme potrebné cesty k spustiteľným súborom pre nástroj SikuliX, simulačné nástroje a testovanú aplikáciu. Cesta k aktuálnemu testovaciemu plánu (JMETER_TEST_PATH) je automaticky zistená prostredníctvom BeanShell skriptu [12], ktorý je možné priamo implementovať do poľa pre hodnotu premennej.



Obr. 3 Konfigurácia ciest v komponente C2D_TEST_PLAN_CONFIG

Ak testovaná aplikácia obsahuje REST API rozhranie, prostredníctvom ktorého je možné taktiež pristupovať k jej testovaniu, v komponente **HTTP Request Defaults** (obr. 4) definujeme

predvolené parametre pre posielanie HTTP požiadaviek. S tým súvisí aj preddefinovanie hlavičiek HTTP požiadaviek v komponente **HTTP Header Manager** (obr. 4).



Obr. 4 Definovanie parametrov v HTTP Request Defaults a HTTP Header Manager

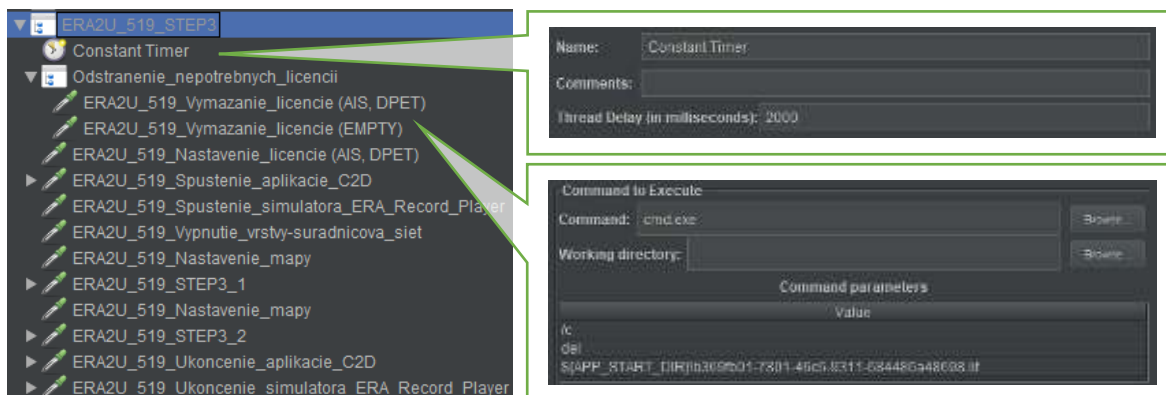
Ak je testovacie prostredie pripravené, môžeme pristúpiť k vytváraniu automatických testov. Aktuálne je v testovacom pláne *TEST PLAN* zahrnutý test *ERA2U_3* a test *ERA2U_519* (viď príloha C). Obidva testy vychádzajú z manuálnych testov, pričom štruktúra medzi automatickými a manuálnymi testami ostáva zachovaná.

1.3. Prepísanie testovacích prípadov do testovacích skriptov

Po kliknutí na položku *TEST PLAN > Release > ERA2U_519* je v nástroji JMeter zobrazená štruktúra testovacieho prípadu *ERA2U-519:Release test pred odoslaním setup-u_5.4 a 5.5*. Keďže ide o rozsiahly test, ktorý overuje v jednoduchých krokoch základnú funkčnosť naprieč celou aplikáciou C2D (tzv. Quick test), bol rozdelený na 8 častí. Ak si napríklad rozbalíme položku *ERA2U_519_STEP3*, zobrazí sa nám štruktúra časti testu pre filtre a profily. Jednotlivé kroky testu je možné rozdeliť do dvoch kategórií:

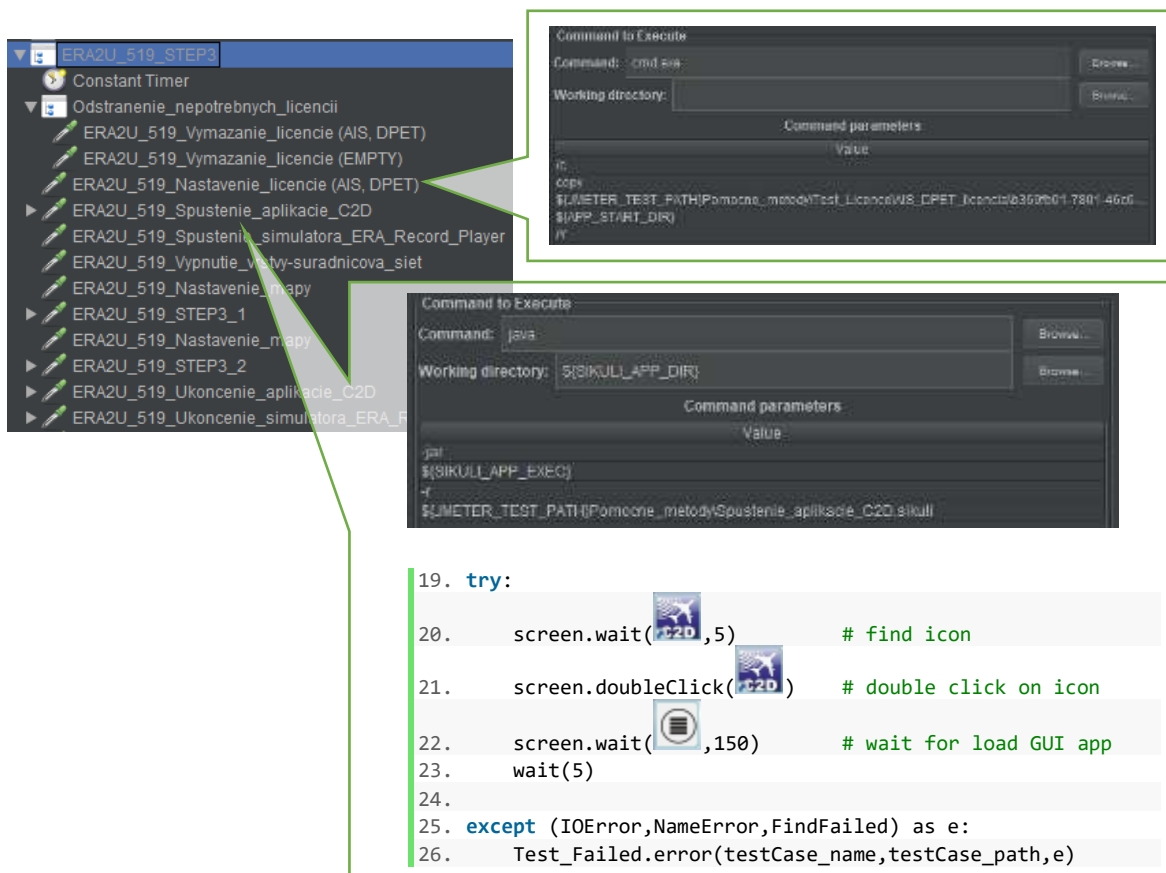
- **Predpoklady pre vykonanie a ukončenie testu** – kroky súvisiace s prípravou testovacieho prostredia.
- **Spustenie GUI simulátora** – kroky súvisiace s vykonávaním simulačných SikuliX skriptov v GUI aplikácii C2D.

Na začiatku testu v komponente **Constant Timer** nastavíme časovač, v ktorom definujeme oneskorenie medzi jednotlivými krokmi (procesmi). Následne pomocou MS-DOS príkazu *del* v komponente **OS Process Sampler** odstránime všetky licencie v adresári [c:\opt\rsys\c2d].



Obr. 5 Nastavenie časovača a odstránenie licencií

Na základe testovacích predpokladov nastavíme požadovanú licenciu pomocou MS-DOS príkazu *copy*. Licencia bude prekopírovaná z testovacích dát [..\Projekt_C2D\Pomocne_metody\Test_licence] do adresára [c:\opt\rsys\c2d]. Aplikáciu C2D spustíme pomocou SikuliX skriptu, ktorý vyhľadá na pracovnej ploche ikonu aplikácie, dvojklikom ju spustí a čaká na jej načítanie. Príkaz pre spustenie SikuliX skriptu je definovaný v komponente **OS Process Sampler** nasledovne:

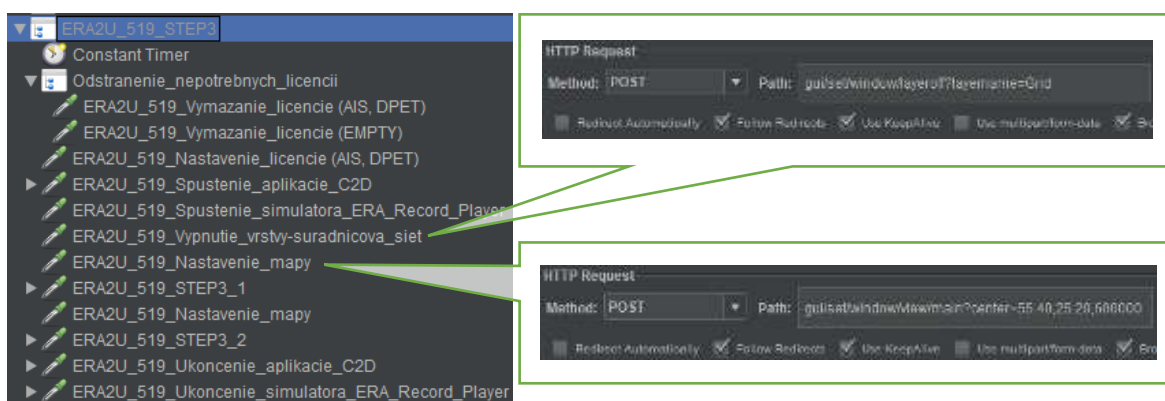


Obr. 6 Nastavenie licencie a spustenie aplikácie

Ak je testovaná aplikácia pripravená, spustíme potrebný simulačný nástroj. Spôsob spustenia simulačných nástrojov môže byť rôzny. V našom prípade využijeme komponent **BeanShell Sampler**, ktorý obsahuje skript pre spustenie simulátora *ERA Record Player* prostredníctvom metódy `Java.lang.Runtime.exec()`:

- `Runtime.getRuntime().exec("cmd.exe /c start " + vars.get("JMETER_TEST_PATH") + vars.get("SIMULATOR_UFE_START_PATH"));`

Pre lepšiu identifikáciu cieľov vypneme v aplikácii C2D súradnicovú sieť a nastavíme presné zobrazenie mapy podľa definovanej mierky a súradníc. Na tieto úkony nám poslúžia funkcie interného ladiaceho rozhrania (viď príloha C). Komunikáciu s týmto rozhraním zabezpečuje komponent **HTTP Request Defaults**.



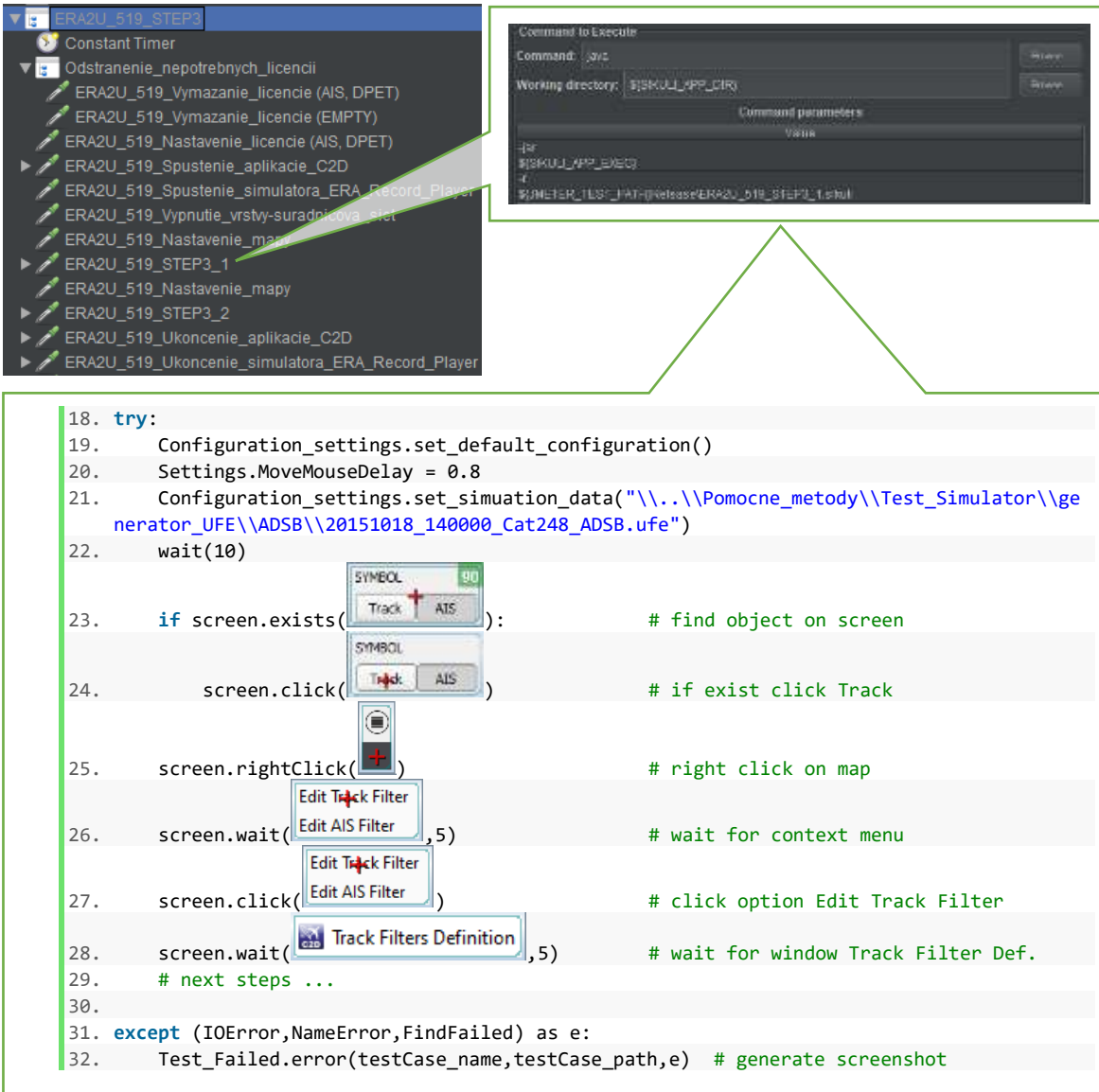
Obr. 7 Vypnutie súradnicovej siete a nastavenie mapy

Následne môžeme pristúpiť k automatizovaniu jednotlivých krokov podľa testovacieho scenára (príloha C: *ERA2U_519* – krok 3.1).

#	Step actions:	Expected Results:
3	<p>1. Postupne vytvoriť po jednom filtri z kontextového menu mapy, z tabuľky track-ov/AIS objektov a cez tlačidlo Edit v controlbar. Zobrazí sa okno Track/AIS Filters Definition a v ňom definovať tieto filtre.</p> <ul style="list-style-type: none"> • Vytvoriť filter pre track, v ktorom bude viac ako jedna podmienka. • Do vytvoreného filtra pridať link na iný filter. • Overiť funkčnosť filtrov. • Zopakovať vyššie uvedené kroky pre AIS objekt. <p>2. ...</p>	<p>1. Filtre sa dajú vytvoriť a ich zapnutie /vypnutie sa prejaví v mape/príslušnej tabuľke a v indikácii počtu filtrovaných track-ov/AIS objektov v controlbar a v tabuľkách.</p> <p>2. ...</p>

Tab. 1 Test Case ERA2U_519 – krok 3.1

V prostredí nástroja SikuliX vytvoríme skript, ktorý automatizuje kroky obsiahnuté v tabuľke 1 (viď obr. 8). Spustenie skriptu zabezpečí komponent **OS Process Sampler**, podobne ako pri spustení aplikácie C2D.



```

18. try:
19.     Configuration_settings.set_default_configuration()
20.     Settings.MoveMouseDelay = 0.8
21.     Configuration_settings.set_simulation_data("\\.\\Pomocne_metody\\Test_Simulator\\ge
nerator_UFE\\ADSB\\20151018_140000_Cat248_ADSB.ufe")
22.     wait(10)
23.     if screen.exists(Track AIS): # find object on screen
24.         screen.click(Track AIS) # if exist click Track
25.         screen.rightClick(Track AIS) # right click on map
26.         screen.wait(Track AIS, 5) # wait for context menu
27.         screen.click(Edit Track Filter) # click option Edit Track Filter
28.         screen.wait(Track Filters Definition, 5) # wait for window Track Filter Def.
29.         # next steps ...
30.
31. except (IOError, NameError, FindFailed) as e:
32.     Test_Failed.error(testCase_name, testCase_path, e) # generate screenshot

```

Obr. 8 Spustenie GUI simulátora

Na začiatku skriptu importujeme všetky potrebné knižnice vrátane knižnic **Test_Failed** a **Configuration_settings**. Knižnica **Configuration_settings** obsahuje pomocné funkcie často sa opakujúcich krokov:

- `set_simulation_data(MyDataPath)` – nastavenie a spustenie simulácie testovacích dát, kde `MyDataPath` predstavuje cestu k potrebnej nahrávke,
- `close_simulator()` – vypnutie simulátora,
- `switch_system_under_test()` – zobrazenie testovanej aplikácie v popredí,
- `close_system_under_test()` – vypnutie testovanej aplikácie,

- `close_simulator_AIS()` – vypnutie simulátora AIS,
- `set_default_configuration()` – importovanie predvolenej konfigurácie,
- `menu_open_option()` – otvorenie okna *Options*,
- `menu_open_trackTable1()` – otvorenie okna *Track Table 1*,
- `menu_open_AISTable()` – otvorenie okna *AIS Table*,
- `menu_open_SelectionInspector()` – otvorenie okna *Selection Inspector*,
- `menu_open_ExportConfiguration()` – otvorenie okna *Export Configuration*,
- `menu_open_ImportConfiguration()` – otvorenie okna *Import Configuration*,
- `menu_open_Recording()` – otvorenie dokovacieho okna *Recording*,
- `menu_open_Drawing()` – otvorenie dokovacieho okna *Drawing*,
- `menu_open_Overview()` – otvorenie dokovacieho okna *Overview*,
- `menu_open_MapLayers()` – otvorenie dokovacieho okna *Map Layers*,
- `menu_open_Screenshot()` – otvorenie okna *Screenshot*,
- `menu_open_Profile()` – otvorenie dokovacieho okna *Profile*,
- `menu_open_Priorites()` – otvorenie dokovacieho okna *Priorites*,
- `menu_open_OfflineWindow()` – otvorenie *Offline* okna.

Bližšie informácie k používaniu nástroja SikuliX a implementácii vizualizačných skriptov sú dostupné v SikuliX dokumentácii [13]. Na záver testu pridáme komponenty pre ukončenie testovanej aplikácie a simulátorov. Spustenie a priebeh automatického testu môžeme overiť stáčením tlačidla *Start*.

1.3.1. Vyhodnotenie SikuliX skriptov

Aby mohol JMeter automaticky vyhodnotiť výsledok vykonaného SikuliX skriptu, použijeme komponent *BeanShell Assertion*. V komponente je implementovaný BeanShell skript, ktorý prechádza výstupné logy z nástroja SikuliX a hľadá v nich kľúčové slovo *TestFailed*.

```

1. String response = new String(ResponseData);
2.
3. if(response.contains("TestFailed")){
4.     Failure = true;
5.     FailureMessage = response;
6. } else {
7.     Failure = false;
8. }

```

label	responseCode	threadName	success	failureMessage
ERA2U_519_Nastavenie..	200	TEST PLAN 1-1	TRUE	
ERA2U_519_STEP3_1	0	TEST PLAN 1-1	FALSE	__STEP3_1_TestFailed__ FindFailed: 1583090570912.png: (79x71) in R[0,0 1920x1080]@S(0) Line 2284, in file Region.java

			Unable to find expected object (..\Test_Errors\ERA2U_519_STEP3_1-2_error.png) on screen (..\Test_Errors\ERA2U_519_STEP3_1-2_expect.png)
--	--	--	--

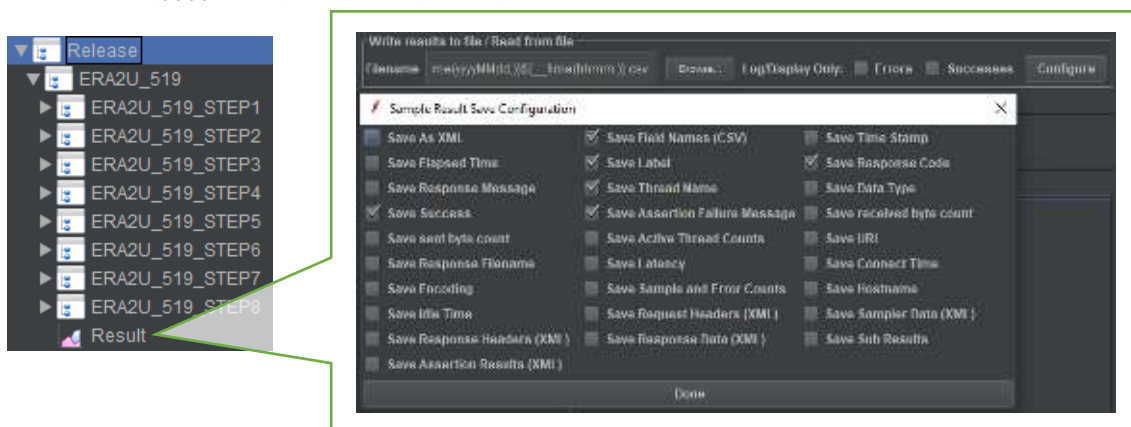
Obr. 9 Vyhodnotenie SikuliX skriptu

Ako je zobrazené na obrázku 9, do výsledného reportu v časti *failureMessage* bude zároveň prenesený celý záznam pre lepšiu analýzu vzniknutej chyby.

1.4. Generovanie reportov a údržba testov v postprojektových fázach

Generovanie reportov vo formáte CSV zabezpečíme prostredníctvom komponentu **View Result Tree**. V časti *Write results to file* definujeme názov súboru a cestu, kde sa má report uložiť:

- `${JMETER_TEST_PATH}\Pomocne_metody\Test_Results\Test_result_ERA2U_519_${__time(yyMMdd,)}${__time(hhmm,)}.csv`



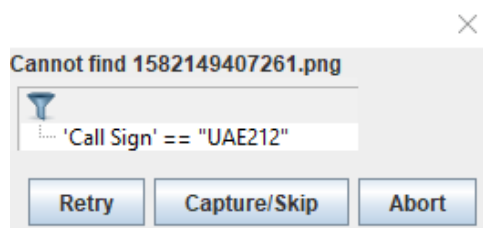
Obr. 10 Generovanie reportu

V konfiguračných nastaveniach označíme položky, ktoré budú obsiahnuté vo výslednom reporte (viď obr. 9). Po ukončení testovania bude do adresára [..\Pomocne_metody\Test_Results] vygenerovaný report s označením testu a časovou značkou (napr. *Test_result_ERA2U_519_202004011033.csv*).

Pre lepšiu kontrolu nad automatickým testom je vhodné v SikuliX skriptoch používať funkciu `screen.setFindFailedResponse(PROMPT)`. V prípade ak dôjde k zlyhaniu testu, objaví sa dialógové okno (obr. 11), ktoré informuje testera o nenájdennom očakávanom objekte na obrazovke, nad ktorým má byť vykonaná určitá akcia. V takomto prípade má tester nasledujúce možnosti:

- **Opakovať vykonanie akcie** – Ak chyba vznikla v dôsledku neočakávaného zásahu OS alebo programov tretích strán do priebehu testu, je možné v procese testovania napraviť daný stav a ďalej pokračovať vo vykonávaní testu.

- **Prerušit vykonávanie skriptu** – Ak je evidentné, že chyba vznikla v dôsledku neočakávaného správania sa testovaného systému, môžeme prerušiť vykonávanie skriptu, pričom bude chyba zaznamenaná vo výslednom reporte.
- **Upraviť alebo preskočiť danú akciu** – Táto funkcia je výhodná hlavne v postprojektových fázach, kedy je možné robiť úpravy testovacích skriptov v priebehu automatického testovania. Kliknutím na tlačidlo *Capture* označíme v aktuálnej obrazovke oblasť nového objektu, ktorým nahradíme pôvodne používaný GUI objekt. Zmeny sú automaticky uložené a skript pokračuje ďalej vo vykonávaní testu.



Obr. 11 Dialógové okno pri výskyte chyby

Ak dôjde k prerušeniu testu, vykoná sa funkcia `Test_Failed.error()` (viď obr. 8 riadok 32), ktorá do adresára `[..\Pomocne_metody\Test_Errors]` vygeneruje aktuálnu snímku obrazovky v čase výskytu chyby a očakávaný hľadaný objekt. Táto informácia je zobrazená aj vo výslednom reporte v časti *failureMessage* (viď obr. 9).

Ak nie je možné úplne automatizovať určité kroky v testovacom scenári a automatický test vyžaduje manuálny zásah testera do priebehu testu, odporúča sa použiť funkciu `screen.popup(text[title])`. Vstupným argumentom funkcie je textový reťazec, ktorý sa po spracovaní zobrazí v dialógovom okne. Okrem toho nástroj SikuliX obsahuje sadu ďalších funkcií (`popError()`, `popAsk()`, `inputText()`, `select()`) [14], pomocou ktorých je zabezpečená interakcia používateľa s automatickými skriptami. Po zatvorení dialógového okna skript pokračuje ďalej vo vykonávaní automatického testu.

Automatické testy je nevyhnutné vždy udržiavať v aktuálnom stave. Každá zmena vykonaná v manuálnych testovacích prípadoch musí byť prenesená do automatických testovacích skriptov. Pre lepší prehľad a evidenciu zmien vytvoríme zdieľaný testovací dokument (napr. Google Sheets [15]), ktorý bude prístupný pre celý testovací tím (v prípade potreby aj celý projektový tím).

V testovacom dokumente spíšeme zoznam všetkých testov v rámci projektu a príznakom označíme tie, ktoré sú automatizované. V prípade, ak autor testu realizuje úpravy v aktuálne platnom testovacom prípade, je povinný o nich informovať prostredníctvom tabuľky pre evidenciu aktualizácií, v ktorej poznačí skratku svojho mena, aktuálnu verziu testu a stručný komentár o

vykonaných úpravách. Automatizovaný tester tak bude informovaný o všetkých úpravách v testovacích prípadoch a na základe týchto informácií prevedie potrebné zmeny v automatických testoch.

Zoznam použitej literatúry

- [1]. ROUDENSKÝ Peter; HAVLÍČKOVÁ Anna. Řízení kvality softwaru. Computer Press, Albatros Media as, 2017. ISBN: 978-80-251-3816-8.
- [2]. SIMPSON Jim; WISNOWSKI Jim. Automated Software Testing Implementation Guide [online]. STAT T&E Center of Excellence, 2017 [cit. 2019-3-5]. Dostupné na internete: https://www.afit.edu/stat/statcoe_files/Automated_Software_Testing_Implementation_Guide.pdf.
- [3]. BUREŠ Miroslav, RENDA Miroslav; DOLEŽEL Michal. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Grada Publishing as, 2016. ISBN: 978-80-247-5594-6.
- [4]. PAGE Alan; ROLLISON, B; JOHNSTON Ken. Jak testuje software Microsoft. Computer Press, Albatros Media as, 2017. ISBN 978-80-251-2869-5.
- [5]. RANOREX.com. Regression Testing Guide [online]. [cit. 2019-5-5]. Dostupné na internete: <https://www.ranorex.com/resources/testing-wiki/regression-testing/>.
- [6]. SOLANSKI Kamna Jyoti. A Comparative Study of Five Regression Testing Techniques: A Survey [online]. International journal of scientific & technology research volume 3, August 2014 [cit. 2019-5-5]. Dostupné na internete: <https://www.ijstr.org/final-print/aug2014/A-Comparative-Study-Of-Five-Regression-Testing-Techniques-A-Survey.pdf>.
- [7]. SOFTWARETESTINGHELP.com. Regression Testing Complete Guide: Tools, Method, and Example [online]. April 2019 [cit. 2019-5-5]. Dostupné na internete: <https://www.softwaretestinghelp.com/regression-testing-tools-and-methods/>.
- [8]. JMETER.APACHE.org. The Apache JMeter™ open source software [online]. [cit. 2019-8-30]. Dostupné na internete: <https://jmeter.apache.org/>.
- [9]. SIKULIX.com. SikuliX: Automate what you see on a computer monitor [online]. [cit. 2019-8-31]. Dostupné na internete: <http://www.sikulix.com/>.
- [10]. NODEJS.org. JavaScript runtime built on Chrome's V8 JavaScript engine [online]. [cit. 2019-12-28]. Dostupné na internete: <https://nodejs.org/en/>.
- [11]. ACTIVESTATE.com. Perl From ActiveState [online]. [cit. 2019-12-28]. Dostupné na internete: <https://www.activestate.com/products/perl/>.

- [12]. JMETER.APACHE.org. BeanShell function [online]. [cit. 2020-4-2]. Dostupné na internete:
<https://jmeter.apache.org/usermanual/functions.html#__BeanShell>.
- [13]. SIKULI.org. Sikuli Documentation [online]. [cit. 2020-4-2]. Dostupné na internete:
<<http://doc.sikuli.org/>>.
- [14]. SIKULI.org. Interacting with the User and other Applications [online]. [cit. 2020-4-2].
Dostupné na internete:
<<https://sikulix-2014.readthedocs.io/en/latest/interaction.html>>.
- [15]. GOOGLE.com. Google Sheets [online]. [cit. 2020-4-12]. Dostupné na internete:
<<https://www.google.com/sheets/about/>>.

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

**APLIKOVANIE NÁSTROJOV AUTOMATICKÉHO
TESTOVANIA DESKTOPOVÝCH APLIKÁCIÍ S GRAFICKÝM
ROZHRANÍM**

Diplomová práca

Príloha C
Testovacie prípady

Test Case ERA2U-3: RA.6.2 Umožniť vytváranie logických kombinácií "dočasných" filtrov s použitím operátorov AND a OR. [Version : 2]		
Autor	Pavol Bednárík - 11/04/2017 10:24:23	
Naposledy upravil	Stanislav Valica - 18/12/2019 14:50:01	
<u>Zhrnutie:</u>		
<p>Umožniť vytváranie logických kombinácií filtrov ako dvojstupňové logické výrazy s použitím AND a OR tak, ako je to riešené v pôvodnej aplikácii ERA Display.</p> <p>Na základe tejto požiadavky dôjde k zmene editačného okna tak, ako je riešené v aplikácii Display. Toto riešenie okrem iného prinesie možnosť vytvárania logických kombinácií filtrov pomocou operátora „OR“ a AND.</p> <p>Okno umožní vytvoriť tzv. „Anonymný“ filter. Ide o filter, ktorý nebude uložený pod menom, t.j. po jeho definícii sa priamo aktivuje (tlačidlo „Set“) v príslušnej funkcii, z ktorej bol vyvolaný. Po ukončení aplikácie C2D sa definícia tohto filtra stratí.</p>		
<u>Predpoklady:</u>		
-na vykonanie testu potrebujeme bežiacu aplikáciu C2D s licenciou DPET.		
<u>Testovacie vstupy:</u>		
-nastavený aktívny kanál,		
-dáta trackov, AIS a AIS_UFE musia byť prijímané.		
#	Kroky	Očakávané výsledky
1	<ol style="list-style-type: none"> 1. V hornej lište zapnúť tlačidlo Track. 2. Skontrolovať položku Filter. 3. Kliknúť na položku Edit vedľa comboboxu filter v hornej lište. 4. Označiť [temporary] a potvrdiť set. 5. Opäť aktivovať Track Filter Definition a vo filtri [temporary] zadeinovať nejaké logické operátory. Po použití set skontrolovať funkčnosť filtra. 6. Reštartovať aplikáciu C2D a overiť vynulovanie filtra [temporary]. 7. Postup opakovať aj pre AIS filtre. (bod 1 prepnúť AIS) 	<ol style="list-style-type: none"> 1. Track je zapnutý a aktívny (filtre platia pre tracky) 2. Položka Filter obsahuje --no filters--. 3. Aktivuje sa okno Track Filter Definition. 4. Položka Filter obsahuje [temporary]. 5. Je možné definovať filter a ten správne zobrazí v mape tracky podľa nastavených logických operátorov. 6. Po reštarte aplikácie sa logické operátory a nastavenie filtra vynuluje do prednastaveného stavu. 7. Výsledok je rovnaký a platný aj pre AIS.
Typ vykonania	Automatické	
Čas vykonania (min)	30	
Dôležitosť	Priemerná	
Akcia pri zastavení systému	-reštart aplikácie C2D -zopakovať celý testovací postup od začiatku	
Analýza výsledku testu	-zálohovať dump súbor -analýza vytvoreného dump súboru -analýza vstupných dát v jednotlivých krokoch testovacieho postupu	
Požiadavky	RA.6.2: Logické kombinácie filtrov ako logické výrazy s použitím AND a OR	
Kľúčové slová	Filtre, Logické kombinácie	

ERA2U-519 : Release test pred odoslaním setup-u_5.4 a 5.5 - Version 1		
Autor	Stanislav Valica – 26/04/2019 07:33:26	
Naposledy upravil	Stanislav Valica – 26/04/2019 07:33:26	
<u>Zhrnutie:</u>		
Pred odoslaním výsledného setup-u na ftp ERA je potrebné urobiť tzv. rýchly release test aplikácie, podľa nasledujúcich krokov:		
<u>Predpoklady:</u>		
<p>-na vykonanie testu je vhodné mať čistý WIN7/10,</p> <p>-je potrebné mať nainštalované mapy UAE a ICN,</p> <p>-testovanie AIS dát a iných súčastí aplikácie, ktoré sú viazané na licenciu, je možné vykonať len vtedy, keď sú dostupné v danom setup-e,</p> <p>-aplikácia prijíma každý typ dát AIS, AIS-UFE a tracky,</p> <p>-postup spustenia testovacej nahrávky: http://wiki.irsys.sk/index.php/ERA_C2D_2016:Testovanie.</p>		
#	Kroky	Očakávané výsledky
1	<ol style="list-style-type: none"> Nainštalovať setup na testovacom PC pod OS Windows. Spustiť aplikáciu s AIS licenciou. <ul style="list-style-type: none"> Skontrolovať, či sa v hornej ovládacej lište (ďalej len controlbar) nachádza položka/tlačidlo AIS a kliknúť na toto tlačidlo. Skontrolovať, či sa v hlavnom menu nachádzajú položky AIS Table a AIS-UFE Table a otvoriť nimi príslušné tabuľky. V hlavnom menu vybrať položku About. Zobrazí sa okno, v ktorom je potrebné skontrolovať správnosť verzie a release aplikácie. Zadeinovať cez hlavné menu > Options > Channels každý typ kanálu (Track, AIS-UFE a AIS) tak, aby každý kanál prijímal dáta. Reštartovať aplikáciu, no spustiť ju bez AIS a MIL licencie. <ul style="list-style-type: none"> Skontrolovať, či sa v controlbar nachádza položka/tlačidlo AIS. Skontrolovať, či sa v hlavnom menu nachádzajú položky AIS Table a AIS-UFE Table. Pre jeden track otvoriť okno inšpektora. V hlavnom menu vybrať položku About. Reštartovať aplikáciu a spustiť ju s AIS a MIL licenciou. <ul style="list-style-type: none"> Skontrolovať, či sa v controlbar nachádza položka/tlačidlo AIS. Skontrolovať, či sa v controlbar nachádza položka NRD Time. Poznámka: Na to je potrebné mať zadanú cestu k súboru NRD_timestamp.txt a povolené zobrazovanie položky v toolbar. Pre jeden track otvoriť okno inšpektora. V hlavnom menu vybrať položku Overview. 	<ol style="list-style-type: none"> Aplikácia sa nainštaluje bez problémov. <ul style="list-style-type: none"> Počas inštalácie sa aplikácia spýta na jazyk inštalácie a na umiestnenie aplikácie po inštalácii. Aplikácia sa spustí s AIS licenciou. <ul style="list-style-type: none"> V controlbar sa nachádza položka/tlačidlo AIS. Tlačidlo je po kliknutí označené. Položky AIS Table a AIS-UFE Table sa v hlavnom menu nachádzajú. Po ich výbere sa zobrazia príslušné tabuľky. Číslo verzie a buildu sa zhoduje z číslom nainštalovaného setupu, v zátvorke za číslom buildu je AIS (je to licencia s AIS). Kanály sa dajú zadeinovať a v mape sa začnú zobrazovať ciele a AIS objekty. Aplikácia sa spustí bez AIS a MIL licencie. <ul style="list-style-type: none"> V controlbar sa nenachádza položka/tlačidlo AIS. V controlbar sa nenachádza položka NRD Time. V hlavnom menu sa nenachádzajú položky AIS Table a AIS-UFE Table. Inšpektor sa zobrazil bez funkcie/tlačidla Radar Modes. Číslo verzie a buildu sa zhoduje s číslom nainštalovaného setupu, AIS sa tu nenachádza (nie je to licencia s AIS). Aplikácia nabehne s AIS a MIL licenciou. <ul style="list-style-type: none"> V controlbar sa nachádza položka AIS.

	<ol style="list-style-type: none"> 7. V priebehu testu občas zobrazíť Help cez CTRL+A. 8. Cez hlavné menu -> Options -> Maps prepnúť na rgp projekt ICN.rgp 9. Cez hlavné menu -> Options -> Maps prepnúť na rgp projekt uae_c2d.rgp 	<ul style="list-style-type: none"> • V controlbar sa nachádza položka NRD Time. • Inšpektor sa zobrazil s funkciou/tlačidlom Radar Modes <ol style="list-style-type: none"> 6. Zobrazenie mapy zodpovedá krajine, pre ktorú bol setup vytvorený. 7. Help sa zobrazí správne a aktuálne. 8. Rgp projekt ICN.rgp sa bez problémov načíta a v konzole nie sú žiadne chybové hlásenia . Hlavnom menu -> Options -> Maps v časti Map File Details v Base Layers sa dajú prepínať vrstvy. 9. Rgp projekt uae_c2d.rgp sa bez problémov načíta a v konzole nie sú žiadne chybové hlásenia . <ul style="list-style-type: none"> • Hlavnom menu -> Options -> Maps v časti Map File Details v Base Layers sa dajú prepínať vrstvy.
2	<ol style="list-style-type: none"> 1. V hlavnom menu > Options > Label Layout vybrať po jednej položke do každého riadku pre každý formát formulára (ďalej len label). <ul style="list-style-type: none"> • V controlbar, v časti Label, zapnúť zobrazenie 3. a 4. riadku v labeli pre ciele (Target) a pre AIS objekty. • Zapnúť zobrazenie aj 1. a 2. riadku. • Zapnúť/vypnúť zobrazenie histórie a predikcie pre ciele a AIS objekty. 	<ol style="list-style-type: none"> 1. Položky sa presunú z dolnej časti Available Items do príslušných riadkov v časti Label Layout. <ul style="list-style-type: none"> • V labeloch cieľov a AIS objektov sa začnú zobrazovať v 3. a 4. riadku položky tak, ako boli zadefinované. • V 1. a 2. riadku labelov sa zobrazujú tie položky, ktoré boli definované pre zobrazovanie v týchto riadkoch labela. • Zobrazuje sa podľa nastavenia.
3	<ol style="list-style-type: none"> 1. Postupne vytvoríť po jednom filtri z kontextového menu mapy, z tabuľky track-ov/AIS objektov a cez tlačidlo Edit v controlbar. Zobrazí sa okno Track/AIS Filters Definition a v ňom definovať tieto filtre. <ul style="list-style-type: none"> • Vytvoríť filter pre track, v ktorom bude viac ako jedna podmienka. • Do vytvoreného filtra pridať link na iný filter. • Overiť funkčnosť filtrov. • Zopakovať vyššie uvedené kroky pre AIS objekt. <ol style="list-style-type: none"> 2. V hlavnom menu vybrať položku Track Table. <ul style="list-style-type: none"> • Zmeniť profil tabuľky cez položku Profile v kontextovom menu tabuľky. • Kliknúť na jeden stĺpec v tabuľke pre zotriedenie položiek. • Označiť/Vybrať track v tabuľke kliknutím na príslušný riadok a akciu zopakovať pre viac trackov. • Označiť/Vybrať track kliknutím na jeho pozičný symbol v mape a akciu zopakovať pre viac track-ov. • Zopakovať vyššie uvedené kroky pre AIS objekty a AIS-UFE objekty. 	<ol style="list-style-type: none"> 1. Filtre sa dajú vytvoríť a ich zapnutie/vypnutie sa prejaví v mape/príslušnej tabuľke a v indikácii počtu filtrovaných track-ov/AIS objektov v controlbar a v tabuľkách. 2. Zobrazí sa tabuľka track-ov. <ul style="list-style-type: none"> • Zmena profilu sa prejaví. • Položky sa zo triedili podľa očakávania • Zvolený track sa označil/zvýraznil v tabuľke track-ov aj v mape. • Zvolený track sa označil/zvýraznil v mape aj v tabuľke track-ov.

4	<ol style="list-style-type: none"> 1. Cez dvojklik na pozičný symbol zvoleného track-u v mape, na príslušný riadok v tabuľke track-ov a cez položku Inspector v kontextovom menu zvoleného track-u vyvolať okno inšpektora príslušného track-u. <ul style="list-style-type: none"> • Zopakovať akciu pre AIS objekt a AIS-UFE objekt. 2. V hlavnom menu > Panels vybrať položku Selections Inspektor a postupne označiť v mape track, AIS objekt a AIS -UFE objekt kliknutím na jeho pozičný symbol. 3. Kliknutím pravým tlačidlom myši na pozičný symbol track-u v mape vyvolať jeho kontextové menu a vybrať položku Save After Termination. <ul style="list-style-type: none"> • Počkať, kým sa track stratí (pre urýchlenie testu je možné zakázať príjem dát). 4. Otvoriť si tabuľku track-ov a v kontextovom menu tabuľky vybrať položku Export. <ul style="list-style-type: none"> • Zadať umiestnenie exportu a uložiť. 5. Dvojklikom na pozičný symbol track-u vyvolať jeho inšpektor. V ňom kliknutím na príslušnú ikonu zobrazí Track History a kliknúť na tlačidlo Export. <ul style="list-style-type: none"> • Zadať umiestnenie exportu a uložiť. 	<ol style="list-style-type: none"> 1. Zobrazí sa okno inšpektora zvoleného track-u/AIS objektu/AIS-UFE objektu. 2. V okne Selection Inspektor sa zobrazujú dáta, ktoré sa týkajú aktuálne označeného tracku/AIS objektu/AIS-UFE objektu v mape. 3. Vytvoril sa súbor trackData_-----.csv a hodnoty nachádzajúce sa v súbore sú podľa očakávania. Poznámka: Umiestnenie tohto súboru je zadefinované v hlavnom menu Options > General, časť Track Termination Save Settings. 4. Súbor sa uložil tam, kde bola zadefinovaná cesta a jeho obsah zodpovedá očakávaniam. 5. Súbor sa uložil tam, kde bola zadefinovaná cesta a jeho obsah zodpovedá očakávaniam.
5	<ol style="list-style-type: none"> 1. V hlavnom menu vybrať položku Export Dynamic Config. <ul style="list-style-type: none"> • Vybrať všetky možnosti pre export. • Zadať meno súboru a cestu, kde sa má súbor uložiť. 2. Zmeniť v aplikácii niektoré nastavenia (napr. zmazať kanál, zadefinovať iný profil tabuľky atď.). 3. V hlavnom menu vybrať položku Import Dynamic Config. <ul style="list-style-type: none"> • Kliknúť na tlačidlo Yes. • Nájsť uloženú konfiguráciu z kroku 1 a kliknúť na tlačidlo Otvoriť/Open. • Aplikácia sa reštartuje. 4. Nainportovať priloženú konfiguráciu. 	<ol style="list-style-type: none"> 1. Konfigurácia sa dá vyexportovať. 2. Zmeny sa dajú vykonať. 3. Konfigurácia sa dá importovať <ul style="list-style-type: none"> • Aplikácia nabehla a má takú konfiguráciu, aká bola uložená v bode 1. 4. Konfigurácia sa nainportuje bez problémov (nenájde len cestu k mape).
6	<ol style="list-style-type: none"> 1. V hlavnom menu > Panels vybrať položku Recording. <ul style="list-style-type: none"> • Spustiť nahrávanie. Poznámka: Stále by mali byť prijímané dáta. • Po 10 min. ukončiť nahrávanie. 2. Nájsť vytvorenú nahrávku a spustiť jej prehrávanie. Poznámka: Nahrávka je uložená tam, kde je zadefinovaná cesta, ktorá sa definuje v hlavnom menu > Options > General, časť Recording Settings. 3. V hlavnom menu > Panels vybrať položku Drawing a zakresliť do mapy každý typ objektu v ponuke. <ul style="list-style-type: none"> • Niektoré objekty editovať. 4. Vytvorí Screenshot cez hlavné menu > Panels > Screenshot a cez klávesovú skratku CTRL+PrintScreen (klávesovú skratku zadefinovať v C2D Offline Editor). 5. V hlavnom menu > Options vybrať položku Colors. <ul style="list-style-type: none"> • Zadefinovať novú farebnú schému cez tlačidlo New v Scheme. • Zadefinovať niekoľkým položkám inú ako default farbu. • Overiť nastavenie farby. 6. V mape vyvolať kontextové menu mapy cez pravé tlačidlo myši. <ul style="list-style-type: none"> • Vyskúšať každú funkciu v menu. 	<ol style="list-style-type: none"> 1. Nahrávanie sa dá spustiť. 2. Nad mapou sa začnú zobrazovať nahraté dáta a v controlbar svieti indikácia Replay. 3. Každý typ objektu sa dá zakresliť. <ul style="list-style-type: none"> • Editovanie sa prejaví 4. Screenshot sa vytvorí podľa očakávania. 5. Nastavená farba sa prejavila podľa očakávania. 6. Funkčnosť je podľa očakávania. <ul style="list-style-type: none"> • Jednotky sa prepínajú/prepočítavajú podľa očakávania. • Merací vektor sa dá individuálne zmazať cez dvojklik na jeho label. 7. Zobrazí sa kontextové menu cieľa. <ul style="list-style-type: none"> • Funkčnosť je podľa očakávania.

	<ul style="list-style-type: none"> Po vložení meracieho vektora prepínať jednotky v controlbar. Po vložení meracieho vektora tento zmazať cez dvojklik na jeho label. <p>7. V mape vyvolať kontextové menu cieľa kliknutím pravým tlačidlom myši na pozičný symbol cieľa.</p> <ul style="list-style-type: none"> Vyskúšať každú funkciu v menu. Funkciu Spot report otvoriť pre cieľ, ktorý má radarové dáta a aj pre cieľ, ktorý nemá radarové dáta. 	
7	<ol style="list-style-type: none"> V hlavnom menu vybrať položku Panles > Profile. <ul style="list-style-type: none"> Vytvoriť min. 2 rozdielne profily. Zmeniť zobrazenie, ktoré sme si zadefinovali v profile. Vybrať zadefinovaný profil. V hlavnom menu vybrať položku Panels > Priorities. <ul style="list-style-type: none"> Kliknúť na tlačidlo 1 a zadefinovať/vybrať filter (filter zadefinovať aj pre WDB) a kliknúť na tlačidlo Set. <ul style="list-style-type: none"> Rovnaký filter zadefinovať v tabuľke cieľov. Otvoriť kontextové menu ľubovoľného cieľa a pridať cieľ do WBD. <ul style="list-style-type: none"> Kliknúť na položku Add to Warning DB. Kliknúť na tlačidlo Add. Doplniť do konfiguračného súboru main.rcm cestu k súboru s Mode S DB. Spustiť aplikáciu C2D. <ul style="list-style-type: none"> Počas behu aplikácie editovať súbor s Mode S DB. 	<ol style="list-style-type: none"> Profil sa dá zadefinovať. <ul style="list-style-type: none"> Aktivácia profilu sa prejaví podľa očakávania. Zobrazí sa bočný panel Priorities <ul style="list-style-type: none"> Filter sa prejavil V okne Priorities je v prvom riadku rovnaký počet cieľov ako v tabuľke cieľov, kde je použitý rovnaký filter. Cieľ, ktorý bol zaradený do WBD, je zvýraznený v mape a v tabuľke cieľov. V controlbar je signalizovaný ALARM. Cesta sa dá zadefinovať a aplikácia nabehne. <ul style="list-style-type: none"> Po editácii súboru vyzve aplikácia, po nastavenom čase, k obnove Mode S DB.
8	<ol style="list-style-type: none"> V hlavnom menu cez položku Windows > Add New Offline Window pridať offline okno. <ul style="list-style-type: none"> Pomenovať okno V offline okne v hlavnom menu vybrať položku Record Manager. <ul style="list-style-type: none"> Pridať a načítať každý typ nahrávky. Spustiť prehrávanie načítaných nahrávok. V hlavnom menu otvoriť jednotlivé tabuľky pre každý typ dát. <ul style="list-style-type: none"> Aktivovať filter dát. 	<ol style="list-style-type: none"> Offline okno sa dá pridať. <ul style="list-style-type: none"> Okno má meno, ktorým bolo pomenované. Dá sa pridať a načítať každý typ nahrávky. <ul style="list-style-type: none"> Po spustení prehrávania sa v mape začnú zobrazovať track-y/AIS objekty/AIS-UFE objekty. V každej tabuľke sa zobrazuje príslušný typ dát (track-y/AIS objekty/AIS-UFE objekty). Aktivovaný filter sa správne prejaví v zobrazení príslušného typu dát (track-y/AIS objekty/AIS-UFE objekty).
Typ vykonania	Automatické	
Čas vykonania (min)	6-8 hod.	
Dôležitosť	Priemerná	
Požiadavky		
Kľúčové slová	Release, Quick test	
Priložené súbory	<ul style="list-style-type: none"> konfigurácia 4.9.2019 : c2d_SVKdynamicConfig_20190904-140451.zip c2d_SVKdynamicConfig_20190904-140451.zip simulačné súbory - ../Test_Simulator/generator prednastavená konfigurácia - ../Test_Configuration/defaultConfig.zip 	

TECHNICKÁ UNIVERZITA V KOŠICIACH
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

**APLIKOVANIE NÁSTROJOV AUTOMATICKÉHO
TESTOVANIA DESKTOPOVÝCH APLIKÁCIÍ S GRAFICKÝM
ROZHRANÍM**

Diplomová práca

Príloha D

Popis interného ladiaceho rozhrania C2D

Autor	Michal Vlasák, R-SYS s.r.o.			
Dátum	13.6.2018			
Zdroj	Interná projektová dokumentácia C2D < http://wiki.irsys.sk/index.php/C2D:TEST:TestInterface >			
Popis	URI	Parametre	Návratová hodnota	Podporované od verzie
Práca s cieľmi (dátovým modelom cieľov)				
Získanie celkového počtu trackov v internej tabuľke trackov	track/get/count	n.a.	JSON formát: - true: { "Request Status" : 1, "All Tracks" : 0, "Dead Tracks" : 0, "Live Tracks" : 0 } - false: { "Request Status" : 0 }	V5.0
Získanie informácií o tracku z internej tabuľky trackov	track/get/data	trackid=<number>	xml/json/? slovník dekódovaných parametrov tracku	
Reset (zmazanie) tabuľky trackov	track/reset	n.a.	JSON formát: - true: { "Request Status" : 1 } - false: { "Request Status" : 0 }	V5.0
Ovládanie GUI				
Nastavenie stredu zobrazenia a mierky	gui/set/window/view/main	center=<lon>,<lat>,<scale> Príklad: center=19.3822,48.2661,10000 nastaví stred do bodu 19.3822,48.2661 a zoom na mierku 1:10 000	JSON formát: - true: { "Request Status" : 1, "Position Lat" : 48.2661, "Position Lon" : 19.3822, "Scale" : 10000 } - false: { "Request Status" : 0, "Request Text" : "Center map position is not specified" }	V5.0
Vyvolanie akcie z menu (hlavného) okna	gui/performaction/menu/main	name=<item name>	JSON formát: - true: { "Request Status" : 1, "Request Text" : "Action <item name> done" } - false: { "Request Status" : 0, "Request Text" : "Action <item name> not found" }	V5.0
Vyvolanie akcie z kontextového menu (hlavného) okna	gui/performaction/contextmenu/main	name=<item name> - pre položky, ktoré nepotrebujú pre svoju funkciu polohu na mape name=<item name>,<lon>,<lat> - pre položky, ktoré potrebujú pre svoju funkciu polohu na mape) Podpora pre položku Measure Distance nie je implementovaná -	JSON formát: - true: { "Request Status" : 1, "Request Text" : "Action "Centre Map" in map context menu done" } - false: { "Request Status" : 0, "Request Text" : "Action "Measure Distance" not found!" }	V5.2

		vyžaduje interakciu s mapovým oknom!		
Vyvolanie akcie z kontextového menu cieľa	gui/performance/track/main	name=<item name> & trackid=<number>	n.a.	
Zatvorenie okna v aplikácii podľa jeho titulu	gui/performance/window/close	title=TITULOK	n.a.	V5.2
Zapnutie/vypnutie vrstvy zobrazenia	gui/set/window/layeron alebo gui/set/window/layeroff	layername = <name>	JSON formát: - true: { "Request Status" : 1, "Request Text" : "Layer Analyses activated" } - false: { "Request Status" : 0, "Request Text" : "Layer Analyse not found" }	V5.0
Nastavenie parametrov Option-General	gui/set/options/general	trackSymbolSet = SET_MDDS SET_ERA1 SET_ERA2 SET_NATO	JSON formát: - true: { "Request Status" : 1, "Request Text" : "Current symbol set: SET_ERA1" } - false: { "Request Status" : 0, "Request Text" : "Track symbol set SET_ERA not found" }	V5.0
Nástroje a pomocne príkazy				
Získanie kópie obrazovky hlavného okna	tools/screenshot/main	- bez parametrov vráti screenshot hlavného okna - parameter box=X,Y,width,height definuje výrez obrazu z hlavného okna	screenshot vo formate .png JSON formát: - false: { "Request Status" : 0 }	V5.0
Import konfigurácie aplikácie	tools/set/app/config/dynamic	path=/path/to/dynamicConfig.zip - ZIP s dynamickou konfiguráciou musí byť na stanici spolu s C2D - Aplikácia C2D sa ukončí s kódom 100.	JSON formát: - true: { "Request Status" : 1, "Request Text" : "Dynamic config is importing from file /path/to/dynamicConfig.zip" } - false: { "Request Status" : 0, "Request Text" : "Dynamic config file /path/to/dynamicConfig.zip does not exist!" }	V5.2