VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION

ÚSTAV AUTOMATIZACE A MĚŘICÍ TECHNIKY
DEPARTMENT OF CONTROL AND INSTRUMENTATION

AUTOMATICKÁ KONTROLA KVALITY SOFTWARE NA EMBEDDED ZAŘÍZENÍ
AUTOMATIC QUALITY ASSURANCE OF SOFTWARE FOR EMBEDDED DEVICES

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. Aleš Pernikář
AUTHOR

VEDOUCÍ PRÁCE — Ing. Peter Honec, Ph.D.
SUPERVISOR

BRNO 2019

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Kybernetika, automatizace a měření**
Ústav automatizace a měřicí techniky

Student: Bc. Aleš Pernikář ***ID:*** 164357
Ročník: 2 ***Akademický rok:*** 2018/19

NÁZEV TÉMATU:

Automatická kontrola kvality software na embedded zařízení

POKYNY PRO VYPRACOVÁNÍ:

1. Seznamte se s problematikou testování (test case, terminologie, dostupná řešení a jejich vlastnosti).
2. Definujte požadavky na automatické testy software YSoft SafeQ s využitím existujícího robotického testovacího systému.
3. Zvolte způsob a formát uložení testovacích scénářů.
4. Definujte metriky, které budou během testování měřeny a které mají sloužit k ověření kvality.
5. Navrhněte vhodný systém pro automatickou tvorbu testů splňující požadavky existujícího testovacího systému. Popište podmínky, za kterých bude generování možné (formát vstupních a výstupních dat, vkládání doplňkových informací).
6. Implementujte systém pro tvorbu testů a získávání dat dle definovaných kritérií kvality.
7. Zhodnoťte benefity a nevýhody automatických testů a význam metrik kvality.

DOPORUČENÁ LITERATURA:

Vishawjyoti, Parul Gandhi: A survey on prospects of automated software test case generation methods, ISBN 978-9-3805-4421-2

HLAVAC V., SONKA M., BOYLE R.: Image Processing, Analysis, and Machine Vision, ISBN 978-0495082521

Termín zadání: 4.2.2019 ***Termín odevzdání:*** 13.5.2019

Vedoucí práce: Ing. Peter Honec, Ph.D.
Konzultant:

doc. Ing. Václav Jirsík, CSc.
předseda oborové rady

UPOZORNĚNÍ:
Autor diplomové práce nesmí při vytváření diplomové práce porušit autorská práva třetích osob, zejména nesmí zasahovat nedovoleným způsobem do cizích autorských práv osobnostních a musí si být plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č.40/2009 Sb.

Abstrakt

Tahle práce se zabývá automatizací testování software a hardware pomocí robotického testovacího manipulátoru a systému RQA. Jejím cílem je navrhnout rozhraní pro chytrou tvorbu testovacích scénářů s využitím robotického manipulátoru. Dále se zabývá stanovením metrik sloužících pro kontrolu kvality a sbíráním dat relevantních pro zhodnocení kvality testovaného systému. V neposlední řadě se zde probírá vizualizace výsledků testování a naměřených dat pro sledování kvality.

Klíčová slova

Automatizace, testování, robot, regresní testy, kontrola kvality, vizualizace dat, správa tisku YSoft SafeQ

Abstract

This thesis concerns automation of software and hardware testing using the robotic testing system called RQA. It aims to design an interface to produce test scenarios using the robotic testing system. Next it elaborates metrics measured during testing used for quality assurance of the system uder test. One of the topics covered is a visualization of test data and measurements for an easy overview of quality.

Keywords

Automation, testing, robot, regression tests, quality assurance, data visualization, YSoft SafeQ print management

Bibliografická citace:

PERNIKÁŘ, Aleš. *Automatická kontrola kvality software na embedded zařízení.* Brno, 2019. Dostupné také z: https://www.vutbr.cz/studenti/zav-prace/detail/119282. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav automatizace a měřicí techniky. Vedoucí práce Peter Honec.

Prohlášení

„Prohlašuji, že svou diplomovou (bakalářskou) práci na téma AUTOMATICKÁ KONTROLA KVALITY SOFTWARE NA EMBEDDED ZAŘÍZENÍ jsem vypracoval samostatně pod vedením vedoucí/ho diplomové (bakalářské) práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové (bakalářské) práce dále prohlašuji, že v souvislosti s vytvořením této diplomové (bakalářské) práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne: **13. května 2019**

.....

podpis autora

Poděkování

Děkuji vedoucímu diplomové práce Ing. Peter Honec, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne: **13. května 2019**

……
podpis autora

Obsah

1. Úvod ... 11
2. Teorie testování ... 12
   2.1 Vývojový cyklus ... 12
   2.2 Quality Assurance ... 12
   2.3 Druhy testování ... 13
   2.4 Testy po akceptaci ... 13
   2.5 Další kategorie ... 14
   2.6 Verifikace a validace ... 14
   2.7 Automatizace testů ... 15
      2.7.1 Životnost testů ... 15
      2.7.2 Jaké testy automatizovat? ... 16
   2.8 Vyhodnocení testů ... 16
      2.8.1 Jak poznat, že je testování u konce? ... 16
   2.9 Selenium ... 17
3. Testovaný systém YSoft SafeQ ... 18
   3.1 Správa tisku ... 18
   3.2 Správa skenování ... 19
   3.3 Požadavky na testování ... 19
   3.4 Automatizace pomocí RQA ... 19
   3.5 Základní akce ... 23
   3.6 Požadavky na testovací scénář ... 24
   3.7 Předpoklady ... 25
4. Elaborace ... 28
   4.1 Technologie ... 28
   4.2 Formát testovacích scénářů ... 28
      4.2.1 Hlavička a metadata ... 28
   4.3 Definice testovacího scénáře ... 29
   4.4 Uživatelsky definované akce ... 31
   4.5 Uložení ... 32
5. Implementace rozhraní pro testy ... 33
   5.1 Robot framework ... 33
   5.2 Vlastní návrh testů ... 37
   5.3 Použité technologie ... 37
   5.4 Databázový model ... 39
   5.5 Uživatelské rozhraní ... 41
      5.5.1 Správa testovacích scénářů ... 42
   5.6 Spouštění testů ... 44

5.7 Robustnost testovacího systému ... 45
5.7.1 Falešné nálezy chyb ... 46
6. Metriky pro kontrolu kvality ... 48
6.1 Metriky testovacích scénářů ... 48
6.1.1 Doba trvání ... 48
6.2 Definice testů ... 49
6.2.1 Opakování akcí ... 50
6.3 Metriky testovacích sad ... 51
6.3.1 Pokrytí testů ... 51
6.3.2 Výsledky testů ... 52
6.4 Zobrazení výsledků testování ... 53
6.5 Výpočet statistik výsledků ... 54
7. Zhodnocení výsledků ... 55
8. Možnosti zlepšení ... 56
9. Závěr ... 57
Seznam symbolů, veličin a zkratek ... 60

Seznam obrázků

Obr. 3-1: Uživatelské workflow YSoft SafeQ [7] 19
Obr. 3-2: Schéma informačního toku v RQA 20
Obr. 3-3: Přihlašovací obrazovka s vyznačenými tlačítky 21
Obr. 3-4: Obrazovka klávesnice s vyznačenými tlačítky 22
Obr. 3-5: Obrazovka hlavního menu s definovanými tlačítky 23
Obr. 3-6: Obrazovka SQ Print s vyznačenými tlačítky 25
Obr. 3-7: Příklad nastavení robota a kamery před zařízením 26
Obr. 3-8 Definování tlačítek na zařízení 27
Obr. 5-1: Architektura Robot Frameworku [15] 33
Obr. 5-2: Struktura keywordů použitelných v testech 34
Obr. 5-3: DotVVM – Posloupnost kroků při načítání stránky [17] 38
Obr. 5-4: Entity-relationship model 39
Obr. 5-5: Příklad uložených akcí v databázi 40
Obr. 5-6: Uživatelské rozhraní pro práci s komplexními akcemi 42
Obr. 5-7: Seznam dostupných testů a možnosti 42
Obr. 5-8: Zadání informací pro nový test 43
Obr. 5-9: Posloupnost akcí testovacího scénáře 44
Obr. 5-10: Diagram implementace navigace na obrazovku 47
Obr. 6-1: Vizualizace pokrytí testů 52
Obr. 6-2: Tabulka s vypočítanými statistickými hodnotami 53
Obr. 6-3: Vybrané výsledky testů na zařízení OKI 54

Seznam tabulek

Tabulka 6-1: Vlastnosti měření času změny obrazovek 49
Tabulka 6-2: Tabulka naměřených času doby přihlášení PINem 50
Tabulka 6-3: Naměřené údaje o kvalitě testování 51

1. ÚVOD

Jak se zvyšují požadavky na vývoj software, zlepšují se nástroje pro jeho vývoj. Tím se zvyšuje množství napsaného kódu, ale ne nutně jeho kvalita. Nekvalitní kód může způsobit neočekávané situace při běhu software a oprava chyb v kódu je náročnější na zdroje a čas tím víc, čím později se na chybu přijde. Zvýšení tempa vývoje software se obvykle neodráží ve zvýšení tempa zajišťování kvality (quality assurance - QA). Proto se ve společnosti YSoft zabývám prací na projektu s pracovním názvem RQA (Robotic quality assurance). Tento projekt si klade za cíl přinést nový pohled na testování a zajisťování kvality pomocí těchto moderních technologií.

Čistě softwarového testování má své nedostatky, které se projeví hlavně nutností testovat software společně s jiným hardware, kdy není možnost upravit si tento hardware, ale musí se k němu přistupovat jako k tzv. black boxu. YSoft čelí právě tomuto problému se svým produktem YSoft SafeQ, který je třeba testovat na multifunkčních tiskárnách od jiných výrobců. RQA si klade za cíl právě tuto situaci vyřešit, zatímco tato práce se zabývá otázkou, jak efektivně vytvářet nové testovací scénáře, upravovat již existující podle potřeby a co nejvíce automatizovat tento proces. Tvorba testů s využitím externích testovacích nástrojů by neměla zatěžovat QA inženýry více, než je nutné a zároveň hotové testy musí být přehledné a jejich účel transparentní.

Dále se zabývám problémem robustnosti testování a spolehlivosti jeho výsledků, tedy minimalizování falešných nálezů chyb a zároveň odhalení maxima přítomných chyb. Definuji také měřítka pro zhodnocení kvality testování a sběr dat o testovaném systému. Dalším důležitým aspektem je vizualizaci dat testování, aby byly přehledné a vypovídající pro každého. Výsledky testování je třeba vyhodnocovat dlouhodobě a sledovat trendy ve změně kvality, na základě naměřených údajů během testování.

Vyřešením tohoto zadání by mělo dojít k výraznému ušetření lidských zdrojů a času na testování. Lidé se mohou věnovat kreativní činnosti, tedy samotnému návrhu a vývoji software a hardware. Automatizace testování také poskytne konzistentní výsledky a nezaujaté posouzení kvality. Co se týče ušetřeného času, automatizované testy mohou probíhat mimo pracovní dobu a paralelně na mnoha zařízeních, což může výrazně zkrátit dobu pro dodání novéhu software na trh.

2. TEORIE TESTOVÁNÍ

Oblast softwarového testování je velmi rozsáhlá a obsahuje mnoho pojmů důležitých pro tuto práci, takže zde vybrané pojmy vysvětlím v rozsahu úměrnému dalšímu významu.

2.1 Vývojový cyklus

Vývoj software v moderním pojetí může fungovat lineárně nebo iterativně, tedy v cyklu, kdy vývoj software nekončí po předání zákazníkovi, ale naopak – zpětná vazba od zákazníka inspiruje další vývoj nových funkcí. Takový vývojový cyklus má obvykle 6 fází:

1. Plánování
2. Analýza
3. Návrh
4. Implementace
5. Testování (zajištění kvality)
6. Vydání, podpora, monitoring

Na vývojový cyklus se odkazuji v této práci, kdy se zabývám výhradně fází testování.

2.2 Quality Assurance

Místo termínu testování se více používá již zmíněné QA – zajištění kvality. Vývojové týmy a technologické firmy si uvědomují, že samotné testování je pouze jeden krok ke kýženému cíli – tedy dodání kvalitního produktu zákazníkovi. Pokud bych uvedl extrémní příklad – můžeme provést naprosto všechny testy, tedy testování je zdánlivě hotovo, ale jestli se nezohlední výsledek testů a nikdo se nezabývá analýzou toho, co se během testování zjistilo a pokračuje se ve vývojovém cyklu, je to selhání kontroly kvality.

Test case je definice procedur, vstupů a očekávaných výstupů během jednoho testu software, na základě dané testovací strategie. Test case obsahuje detailní popis všech akcí včetně požadavků na testovací prostředí a další informace jako autor nebo příznak, zda je test case automatizovaný. Definované test cases jsou obvykle pravidelně spouštěny v určité fázi vývojového cyklu, jejich výsledkem bývá zpravidla binární hodnota PASS/FAIL včetně případné chybové zprávy.

Rozdíl mezi test case a test scenario je v úrovní detailu popisu akcí. Zatímco Test case se vyznačuje velmi detailním popisem každé instrukce, testovací scénář se zapisuje v obecné rovině, jazykem srozumitelným každému i netechnickému člověku.

Skupina test case nebo test scenario se společnými prvky nebo testující podobnou část jsou obvykle sdružené ve společném test suite. Pro přehlednost se testy ve stejném suite spouští za sebou a vyhodnocují se společně.

V principu má Test skript podobnou strukturu a totožné vlastnosti jako Test case, rozdíl je pouze v tom, že zatímco Test case lze provádět automaticky i manuálně, Test skript se provádí pouze automaticky.

2.3 Druhy testování

Unit testy se zaměřují na nejmenší samostatné testovatelné části (jednotky), tvoří je výhradně vývojáři během vývoje. Mají otestovat, že změny provedené v kódu fungují podle očekávání. Takové testy jsou obvykle velmi rychle vytvořené, není výjimkou mít stovky unit testů. Také nejrychleji podají zpětnou vazbu o funkčnosti software, protože jejich provedení trvá nejvýše několik sekund.

Úkolem Assembly testů je ověřit, že jednotlivé části je možné sestavit do funkčního celku. Provádí je vývojář a je následně zodpovědný za správné sestavení software.

Existuje celá skupina testů, které spadají do kategorie Systémové a integrační testy, jejich cílem je ověření, že jednotlivé funkční celky software fungují společně v požadované formě a splňují danou definici.

Do kategorie Smoke testy základní testy, které na začátku testovací fáze ověří, zda je vůbec software způsobilý k dalšímu testování. Vývojářský team vydá nový build software a QA jej převezme a musí se ujistit, že nedochází k zásadním problémům. Smoke testování obsahují základní uživatelské scénáře a klíčové komponenty. Pokud je nalezena chyba, ihned je vývojový tým upozorněn.

V skupině Integračních testů se ověřuje správná komunikace a fungování jednotlivých modulů nebo aplikací dohromady. Obvykle se ověřuje fungování klienta se serverem nebo komunikace distribuovaných komponent. Zpravidla se postupuje od menších skupin směrem k většímu celku a některé části mohou být zastoupeny tzv. mockem, který pouze simuluje chování určité sub-komponenty, ale neprovádí žádnou užitečnou činnost, čímž se šetří čas.

Do tzv. systémových testů patří všechny testy, které ověřují produkt, software jako celek se všemi komponenty a bez simulací. Zajišťuje se, že software plní všechny požadavky od zákazníka, nedostává se do chybných stavů a je chráněný před neoprávněným zneužitím.

Než se aplikace dodá zákazníkovi, je nutné ověřit, že splňuje všechny dohodnuté požadavky a vyhovuje zákazníkovu použití. Požadavky, které zákazník na funkce aplikace má se shrnou do akceptačních kritérií, zejména počet a závažnost chyb, spolehlivost, výkonové a zátěžové vlastnosti a další. Splněním těchto kritérií během akceptačního testování může dojít k předání zákazníkovi nebo lze pokračovat v další fázi testování.

2.4 Testy po akceptaci

Hlavním cílem těchto testů je ověřit, že se novými nebo změněnými částmi kódu negativně neovlivnily jiné funkce a uživatelské scénáře. Při vývoji se totiž často chybně myslí pouze na nové funkce a zlepšení vzhledu nebo chování, ale už se nebere na vědomí, zda se může nevědomky porušit existující chování systému. Nelichotivá vizitka je, když se opravením jedné chyby zanesou do programu dvě nové. Právě regresní testy musí striktně vyžadovat fungování nezměněných

součástí. Obecně lze říci, že regresní testování neodhalí mnoho defektů, ale to neznamená, že je lze vynechat.

Regresní testy provádí verifikaci software. Je velmi doporučené je automatizovat, kvůli tomu, jak jsou spouštěny pravidelně. Určitě trvají déle, než testy prováděné dříve ve vývojovém cyklu (integrační, systémové), takže s tímto časem je třeba počítat při odhadu doby před vydáním.

2.5 Další kategorie

Monkey testing je technika v testování software, kdy uživatel zadává náhodné vstupy a provádí náhodné akce a zkouší přivést systém do chybového nebo neočekávaného stavu. Obvykle je toto prováděno automaticky formou testovacích skriptů a kontroluje se stav testovaného systému. Pokud se aplikace dostane do neočekávaného stavu, jedná se o regulérní defekt, který je potřeba opravit.

Existují dva typy monkey testingu: chytrá a hloupá opice. Chytrá opice má obecnou představu o aplikaci, ví, kam mají vést různá tlačítka a jestli zadávaný vstup je validní nebo ne. Také se cíleně snaží najít chybový stav a hlavně chybu rozpoznají. Takový přístup je vhodný pro testování spolehlivosti software, může odhalit chyby, na které by se jinými testy nepřišlo. Mezi nevýhody takového přístupu patří obtížná reprodukovatelnost nalezených chyb, protože cesta k dané chybě může být složitá. Také se může stát, že bude trvat dlouho, než se vůbec nějaká chyba vyskytne.

Další formou cíleného hledání defektu v aplikaci je exploratory testing, který nevyžaduje žádné plánování testovacích scénářů. Průběh testu záleží výhradně na nápadech a přístupu QA inženýra. Často se tady naleznou chyby neodhalené ve formální struktuře test cases, proto téměř vždy k takovému testování někdy v testovacím cyklu dochází.

2.6 Verifikace a validace

Při ověřování kvality software se využívá více přístupů a metod, aby se skutečně ze všech stran rozebral, otestoval a jako výstup z testování musí být přehled zjištěných dat, aby se dalo zodpovědně rozhodnout o dalších krocích. Verifikace a validace se společně používají k ověření, že daný produkt nebo služba splňuje zadané požadavky a plní svůj účel. Je na místě vysvětlit tyto pojmy, které jsou v této práci často používané, a jejich význam se zaměňuje nebo vykládá nepřesně.

Verifikace odpovídá na otázky: „Splňuje daný produkt dohodnuté specifikace, požadavky, předpisy vyplývající z návrhu? Byla vývojová fáze provedena správně? Je vytvořen produkt správně?“.

Validace odpovídá na otázky: „Slouží daný produkt zákazníkovi podle jeho potřeb, je přijatelný pro uživatele a další zúčastněné strany? Byla návrhová fáze provedená správně? Vyplynuly z návrhu správné požadavky a specifikace? Je vytvořen správný produkt?“.

Zatímco verifikace se řeší interně ve vývojovém oddělení, pro validaci je nutné komunikovat se zákazníky, uživateli, relevantními osobami a získávat jejich zpětnou vazbu, jejich připomínky a vyslechnout jejich příběh a důvody, proč

vlastně používají daný produkt nebo službu. Jestli verifikace se prohlásí za splněnou, ale validace skončí neúspěchem, znamená to chybu už ve tvoření specifikací, tedy dříve, než proběhla vývojová fáze. V takovém případě je možné mluvit o plýtvání časem a zdroji, je tedy vhodné průběžně validovat, ne pouze jednou na konci.

2.7 Automatizace testů

Již jsem zmínil, že obecné zrychlování vývoje software vytváří větší tlak na jeho testování a kontrolu kvality, která nemusí stíhat uspokojovat potřeby trhu, což vede k vydávání nekvalitních produktů. Možným řešením je najmout více lidí a vložit do kontroly kvality více zdrojů, ale to nemusí být dlouhodobě udržitelné. Další variantou je tedy kontrolu kvality automatizovat, což sebou nese řadu otázek, z nichž některé bych rozebral. Před automatizací by si každé vývojové oddělení mělo položit následující otázky[4].

1. Automatizování testů znamená vyšší náklady než provedení testů manuálně jednou. Kolikrát vyšší?
2. Automatizované testy mají konečnou „životnost“, tedy musí být aktualizovány nebo možná ztratí na významu a jsou vymazány. Za jak dlouho k tomu dojde?
3. Jaké množství chyb automatizované testy naleznou během své životnosti, kromě prvního běhu po tom, co jsou vytvořeny?

Odpovědi na tyto otázku budou spíše vágní, neurčité a nebudou přesně odrážet realitu. Pomohou však při návrhu automatizace testování, k uvědomění si jejich výhod a nevýhod.

Cena automatizace se často podceňuje, ovšem množství potřebné manuální práce je určitě nezanedbatelné, často výrazně převyšuje náročnost manuálního testování. Velmi záleží na tom, co vlastně chceme automatizovat. Obecně automatizace testování GUI je komplikovanější, protože se může častěji záměrně měnit, takže se musí upravit testy, aby nevyhodnotily falešné hlášení chyby (false-positive). Takové testy mají kratší živostnost a je třeba pravidelně vynaložit další čas na jejich aktualizaci. Pokud se pro testování GUI využije nástroj, který snímá interakci člověka s testovaných systémem a následně vytvoří testovací scénáře, které lze kdykoliv automaticky spouštět, práci to výrazně usnadní [6].

2.7.1 Životnost testů

Automatické testování je v principu užitečné pouze když se testovaný systém mění, takže při přidání, změně nebo smazání části kódu. Jinak se dá předpokládat, že testy nenaleznou více problémů, než při vytvýření těchto testů. Výjimku tvoří zátěžové a výkonostní testy, které . Co když dojde k záměrné změně v kódu, kterou automatický test odhalí jako defekt, zatímco člověk rozumí, že jde o záměr a správně defekt nezahlásí? Pokud automatizaci provádí jiný tým, než vývoj, musí existovat úzká spolupráce a komunikace záměrných změn, aby se redukovaly false positive hlášení. Stále může dojít k tak výrazné změně, že je jednodušší test smazat a vytvořit znovu.

2.7.2 Jaké testy automatizovat?

Ideální představa pro pokrytí automatizace testování je „co největší, automatizovat všechno, co lze“, ale nemusí to být nejrozumnější volba. Největší smysl má automatizovat testy, u kterých je největší šance odhalit chyby při opakovaném spouštění, vždyť hlavním cílem testování je odhalit chyby. V praxi se stává, že se odhalí chyby během vývoje automatických testů nebo po jejich prvním spuštění, ale další spouštění už pouze ověřují, že nedochází k chybám.

Dalším kritériem jsou testy, které jsou velice snadné na návrh a dosáhne se desítek testovacích scénářů prakticky zadarmo. Příkladem mohou být testy správnosti vstupů, kdy se vytvoří tabulka vstupních hodnot a očekávaný výstup. Testovací prostředí potom s každým řádkem tabulky provede test, zadá do systému vstupní hodnoty a ověří očekávaný výstup [2].

2.8 Vyhodnocení testů

Vyhodnocení testů je naprosto klíčová úloha, bez ní se z testování stává ztrátou času. Z výsledků testování je třeba vyvodit patřičné závěry ohledně kvality produktu a navrhnout kroky ke zlepšení. Je třeba zvážit, kolik testů se provedlo, s jakým výsledkem a následně zodpovědět klíčovou otázku: „Je systém dostatečně otestován?“.

2.8.1 Jak poznat, že je testování u konce?

V reálném prostředí nelze prohlásit, že je daný systém (software) dokonale otestovaný a neobsahuje žádné chyby, může se pouze zjistit riziko vyskytnutí chyby. Stále je nutné mít na paměti, že cílem testování je odhalit defekty a podat zpětnou vazbu, aby byly odstraněny. Měřítkem pro ukončení testování by tedy mohl být počet nalezených defektů, ovšem jaký počet je dostatečný k prohlášení spolehlivost nelze vyjádřit přesným číslem. Ani vysoký počet odhalených a následně opravených chyb nezajistní kvalitu systému. Jak jsem zmiňoval dříve, testování může odhalit mnoho falešných chyb, stejně jako mnoho chyb se vůbec nemusí odhalit. Nikdo tedy se současnou technologií nedokáže garantovat, že po skončení testování neobsahuje testovaný systém žádné chyby. Testování tedy skončí z jednoho ze tří důvodů – čas, rozpočet, rozsah testovacích scénářů.

V situaci, kdy je na QA vytvářen tlak na rychlé otestování a rozpočet je příliš limitovaný, nemusí se ani stihnout všechny testovací scénáře provést. V odhadu času je třeba počítat s nalezením kritických defektů, kvůli kterým není možné v testování pokračovat a musí se počkat na jejich opravení. V této situaci nemůže být vůbec zajištěna kvalita, neboť se nedodržel ani základní předpoklad, tedy provedení plného testovacích rozsahu. Přestože je tedy testování u konce, neměl by se produkt v dané iteraci vpouštět zákazníkovi.

V jiné situaci může být dostatek času i financí, stihne se tedy projít celý rozsah testů při nalezení menšího množství defektů. Testování je tedy u konce, ovšem velmi pravděpodobně je příliš malý rozsah testů, takže opět nemůže být zajištěna kvalita a do příští iterace je třeba rozšířit test suite o další test cases.

Pro sledování kvality v rámci continuálního vývoje je ještě důležitější zobrazení trendu úspěšnosti testů, nalezených defektů. Pokud např. při vydání verze 1

spustíme 100 testů a nalezneme 30 defektů, v další verzi 2 spustíme stejných 100 testů a nalezneme 15 defektů, dá se usuzovat, že se kvalita vývoje zlepšila. I kvalita software je v čase proměnná hodnota, je třeba k ní tak přistupovat a vyvozovat závěry na základě trendu, ne pouze jedné iterace. V kapitole Metriky se budu více věnovat dalším údajům, které je třeba sledovat pro správný úsudek o kvalitě testovaného systému.

2.9 Selenium

Jedním z nástrojů, který přináší automatizaci testování zejména webového rozhraní je Selenium [18], který nabízí různé sady nástrojů pro podporu automatizace testů na vícero úrovních. Poskytuje intuitivní WebDriver API jednoduché na používání, což usnadňuje tvorbu a údržbu testů. Použití nástroje Selenium není vázáno na konkrétní programovací jazyk, existují knihovny pro jazyky Java, C#, Python, Ruby, JavaScript. Samotný framework nabízí možnosti ovládání internetového prohlížeče. Při testování je důležitá navigace na cílovou webovou stránku, kterou požadujeme testovat. Dále máme možnost vyhledat cílový element podle jeho ID, názvu, typu nebo třídy. Podle typu elementu je možné do něj zadat nějakou hodnotu, pokud se jedná o textové pole, nebo zvolit aktivitu (v případě checkboxu), případně na něj kliknout. Samozřejmostí je tedy automatická práce s formuláři, vyplňování všech potřebných dat a odeslání vyplněného formuláře. Co je neméně důležité, získávání pozic, textových hodnot a statusů různých elementů.

Selenium je užitečný nástroj, ovšem není vhodný pro použití na aplikaci při testování YSoft SafeQ na multifunkčních zařízeních, protože k těmto nelze přistupovat přímo na jejich webové rozhraní v takové míře, aby šlo zadávat příkazy a ověřovat jejich výsledek. Možností by bylo využít emulátor multifunkčního zařízení, ovšem výrobci neposkytují emulátor ani k testovacím účelům. Požadavky na testování YSoft SafeQ probírám v následující kapitole.

3. TESTOVANÝ SYSTÉM YSOFT SAFEQ

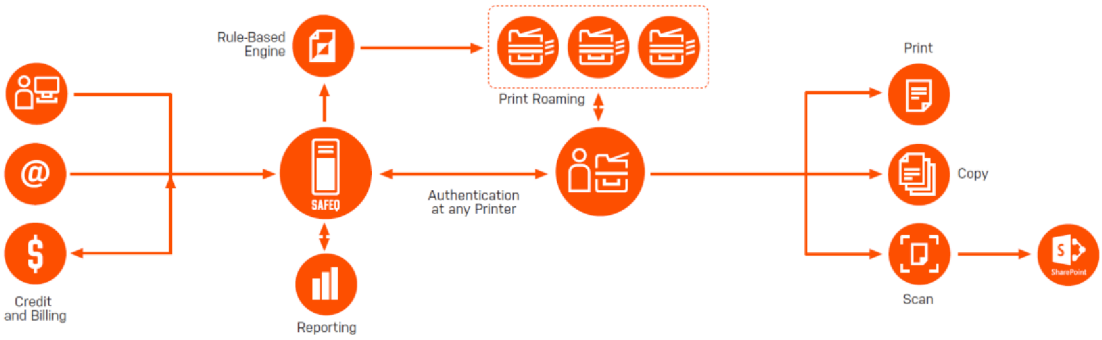
YSoft SafeQ je kompletní řešení pro správu tisku navržené pro podniky a nabízí hlavně snížení nákladů pro tisk, zvýšení produktivity a zvýšení bezpečnosti dokumentů. Zákazník získá přehled o množství tisku, kopírování a skenů ve své organizaci, na základě čehož může podniknout další kroky ke zvýšení efektivnosti práve s dokumenty. YSoft SafeQ dokáže pracovat s celou řadou multifunkčních zařízení, na jejichž terminálech je uživatelské webové rozhraní. Obsahuje řadu modulů, které jsou přístupné podle potřeb zákazníka.

3.1 Správa tisku

Z pohledu zákazníka funguje YSoft SafeQ tak, že si uživatel pošle požadovanou tiskovou úlohu na tiskový server, kde je bezpečně uložen. Následně se uživatel dostaví k libovolnému multifunkčnímu zařízení, které je zapojeno do lokální sítě a spárováno s YSoft SafeQ a je v něm nainstalována aplikace pro terminál. Uživatel se přihlásí pomocí zvolené varianty, například svou přístupovou kartou, uživatelským jménem a heslem nebo PINem. Po přihlášení na terminálu vidí své tiskové úlohy, které jsou připraveny k tisku, který může okamžitě spustit. Po vytisknutí se odhlásí z terminálu a odnese si vytištěné dokumenty sebou. Popsaná situace je jedním z nejtypičtějších testovacích scénářů, které je třeba v testovací fázi provádět na všech dostupných zařízeních.

YSoft SafeQ také umožňuje tzv. Print Roaming v jakémkoliv měřítku. Print roaming znamená, že si zaměstnanec může tiskovou úlohu vytisknout na jakémkoliv zařízení v daném tiskovém prostředí. To znamená kdekoliv na chodbě, v budově, v kampusu, ve státě nebo i na jiném kontinentě, stačí se prokázat přihlašovacími údaji a tiskové úlohy (joby) jsou dostupné.

Samozřejmostí je nastavení pravidel pro tisk nebo kopírování s ohledem na potřebu snížení výdajů papíru, inkoustu nebo spotřebu energie multifunkčních zařízení. Primárními cíly je nejen snížení nákladů, ale snížení zátěže na životní prostředí a upravení přístup uživatelů (zaměstnanců) k tisku. V administraci prostředí YSoft SafeQ lze spravovat různá pravidla, např. změna jednostránkového tisku na duplexový, využití efektivnějšího zařízení pro tisk objemných úloh nebo změna barevného tisku na černobílý. Klientská část aplikace upozorní zaměstnance na změnu ještě před odesláním na tisk. Na hlavní stránce stránce administrace YSoft SafeQ se pak zobrazují statistiky o množství tisku, nákladů na tisk a ušetřených zdrojích.



Obr. 3-1: Uživatelské workflow YSoft SafeQ [7]

3.2 Správa skenování

YSoft SafeQ správa skenování rozšiřuje možnosti multifunkčního zařízení a zahrnuje zabezpečené odeslání dokumentů na požadované místo, ať už do e-mailu uživatele nebo jeho domovského adresáře, případně do sdíleného adresáře. Administrátor nastavuje tzv. workflows, které určují zpracování dokumentu před jeho odesláním. Mimo jiné lze nastavit, že barevně označené fráze v dokumentu nebudou skenovány (citlivé údaje) nebo naopak budou využity jako jméno souboru aj.

3.3 Požadavky na testování

Kompatibilita se zařízeními různých výrobců znamená velké nároky na testování a kontrolu kvality. Protože se mezi sebou zařízení od různých vendorů velmi liší, je YSoft SafeQ přizpůsobeno obvykle pro každého vendora zvlášť. Před vydáním nové verze SafeQ je tedy vhodné ji otestovat na alespoň jednom zařízení každého vendora v rámci regresního testování, aby se zajistilo fungování a bezchybovost.

Produkt YSoft SafeQ se tedy vyznačuje specifickou vlastností, že podstatná část interakce uživatele a systému se odehrává na hardware třetí strany, tedy obvykle multifunkčního zařízení jiného výrobce a nelze tedy použít framework, který by toto zařízení ovládal pomocí vzdálených příkazů a dokázal získávat data o fungování testovaného zařízení. Při vyloučení softwarových testovacích frameworků nezbývá mnoho možností, jak tento problém vyřešit.

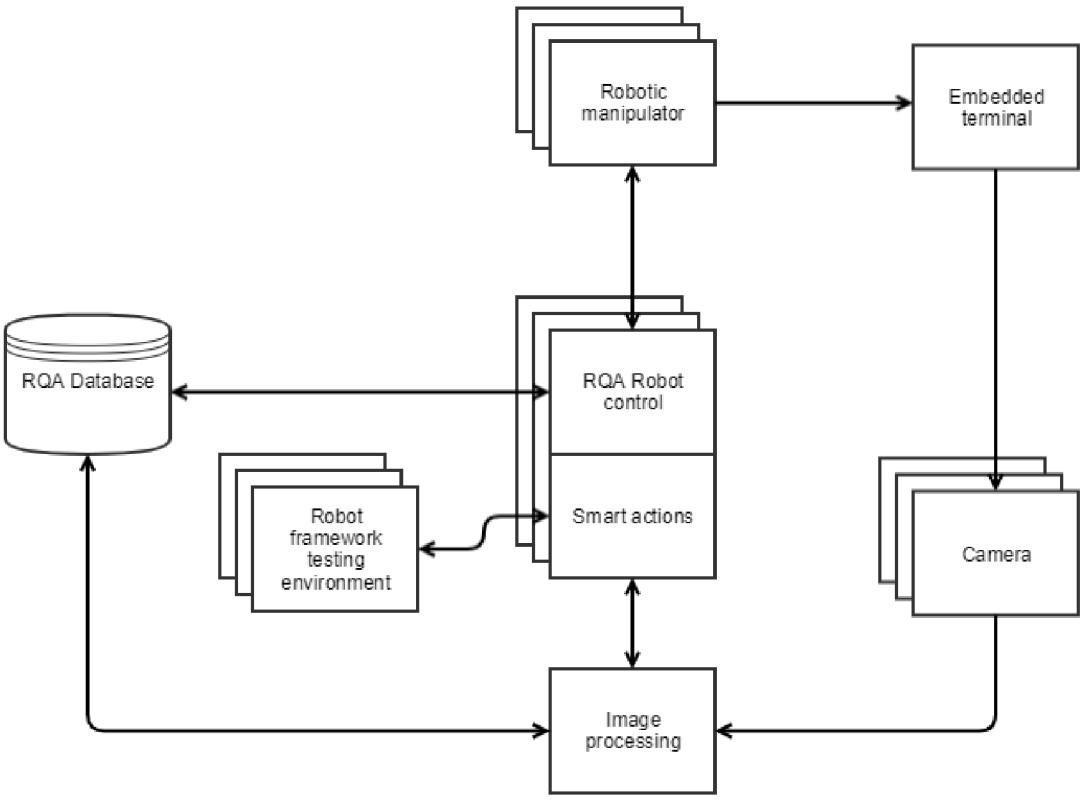
3.4 Automatizace pomocí RQA

Ve firmě YSoft již několik let pracuji na projektu Robotického testovací systému pro kontrolu kvality (zkráceně RQA – Robotic quality assurance). Tento byl zpočátku navrhutý pro potřeby testování YSoft SafeQ hardwarového zařízení přes rozhraní pro člověka, tedy terminál. Zpočátku se jednalo pouze o robotický manipulátor, kameru a řídicí software, následně se přidaly další komponenty, co vykonává test je tedy testovací agent[3]. Robotický manipulátor provádí akce na dotykovém terminálů (poklepání, swipe) podle zadaného testovacího scénáře.

Kamera snímá obrazovku a aplikace zpracování obrazu jej vyhodnocuje a poskytuje zpětnou vazbu o dění na terminálu [9].

Na Obr. 3-2 je znázorněn tok informací mezi různými částmi systému RQA. Aplikace RQA Robot control posílá instrukce robotickému manipulátoru o pohybu na určité pozice na terminálu, robot tedy vykoná tuto akci a terminál zareaguje obvykle změnou obrazovky. Kamera na požadavek zachytí tuto změnu a odešle snímek aplikaci Image processing pro další zpracování. Image processing komunikuje výsledek zpracování s Robot control, který získá zpětnou vazbu o úspěšnosti akce.

Pro tuto práci jsem měl k dispozici robotický systém v provozuschopném stavu, tedy funkční manipulátor s rozhraním pro kliknutí na pozici v souřadnicích (X,Y,Z). Nezabývám se tedy nízkou úrovní komunikace s hardwarem robotického manipulátoru ani výpočtem inverzní kinematiky pro natočení servomotorů na konkrétní pozici. Dále jsem měl v dispozici funkční software pro zpracování obrazu, který má rozhraní pro dotazy na zpracování obrazu z kamery nebo speciální funkce na analýzu jistých částí v obraze, zejména rozpoznání, zda elementy v seznamu jsou aktivní či neaktivní. Zároveň jsem měl k dispozici zdrojové kódy všech aplikací, abych mohl provést požadované změny.



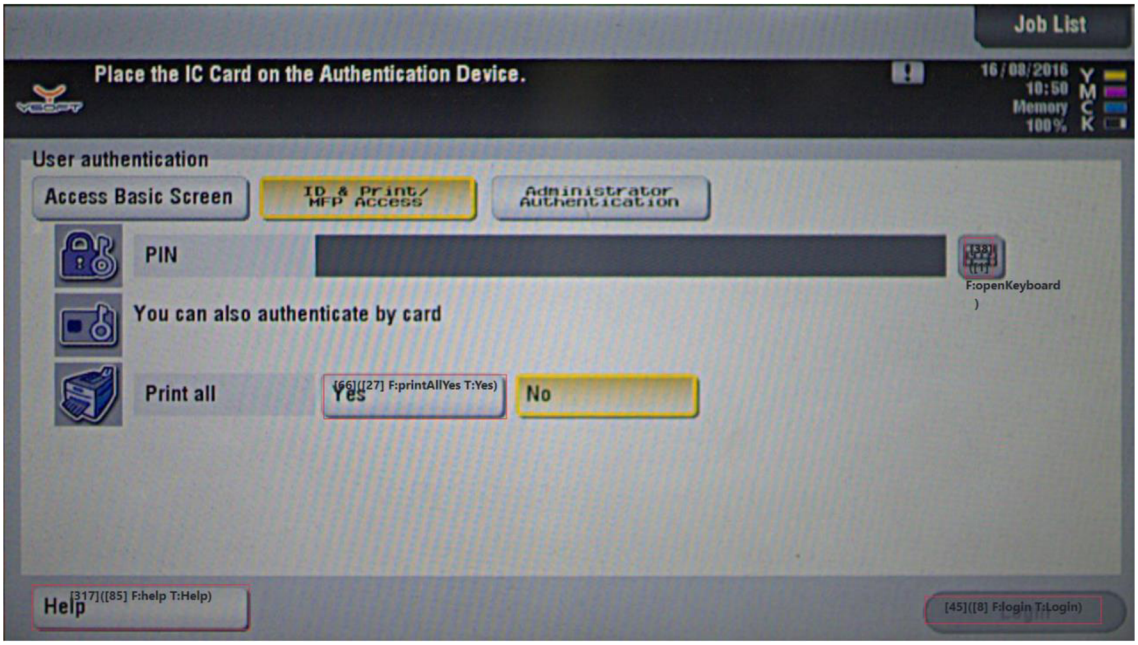
Obr. 3-2: Schéma informačního toku v RQA

RQA ovšem není jenom robotický manipulátor a kamera, ale skrývá se v back-endu řada aplikací zajišťujících chod celého systému. Důležitou aplikací je Robot control, které slouží jako přístupový bod k manipulátoru, obsahuje metody, nazvané smart actions, což jsou algoritmy spojující základní akce do komplexnějších, čímž se usnadňuje návrh testů. Ukažme si nyní, co obnáší

provedení základního testovacího scénáře z pohledu kohokoliv, kdo interaguje s testovaným systémem, ať už je to člověk jako tester nebo automatický testovací systém. Vezmeme jako příklad výše popsaný scénář s přihlášením pomocí PINu a vytisknutím několika tiskových úloh.

Scénář tedy obsahuje tyto úkony:

1. tapnout na tlačítko *otevřít klávesnici* na obrazovce (na obr.2)
2. ověřit, že se obrazovka změnila na požadovanou s klávesnicí
3. vyťukat na obrazovce správný PIN a stisknout tlačítko *přihlásit*
4. ověřit, že se obrazovka změnila a uživatel je přihlášený
5. navigovat se na obrazovku YSoft SafeQ Print, kde se nachází rozhraní pro tisk
6. vybrat a potvrdit tisk požadovaných jobů
7. ověřit, že se vytiskly papíry
8. odhlásit se a ověřit, že se terminál odhlásil



Obr. 3-3: Přihlašovací obrazovka s vyznačenými tlačítky

RQA již v současnosti nabízí řadu možností, jak tyto akce vykonávat. Jako elementární akce považujeme poklepání robota na určitou pozici, kde se nachází tlačítko a vyhodnocení snímku obrazovky. Další elementární akcí je požádání software zpracování obrazu (dále image processing) o zachycení snímku z kamery a jeho vyhodnocení. V této práci se nezabývám algoritmy rozpoznání obrazu, ale vyhodnocení obrazu považuji za black-box, který na požadovaný vstup (vyhodnocení snímku nebo jeho části) reaguje výstupem, což může být název obrazovky nebo analýza její části. Tyhle základní akce jsou pak povýšeny na vyšší úroveň sdružením více elementárních akcí do jedné obecnější, která odráží jeden řádek testovacího scénáře.

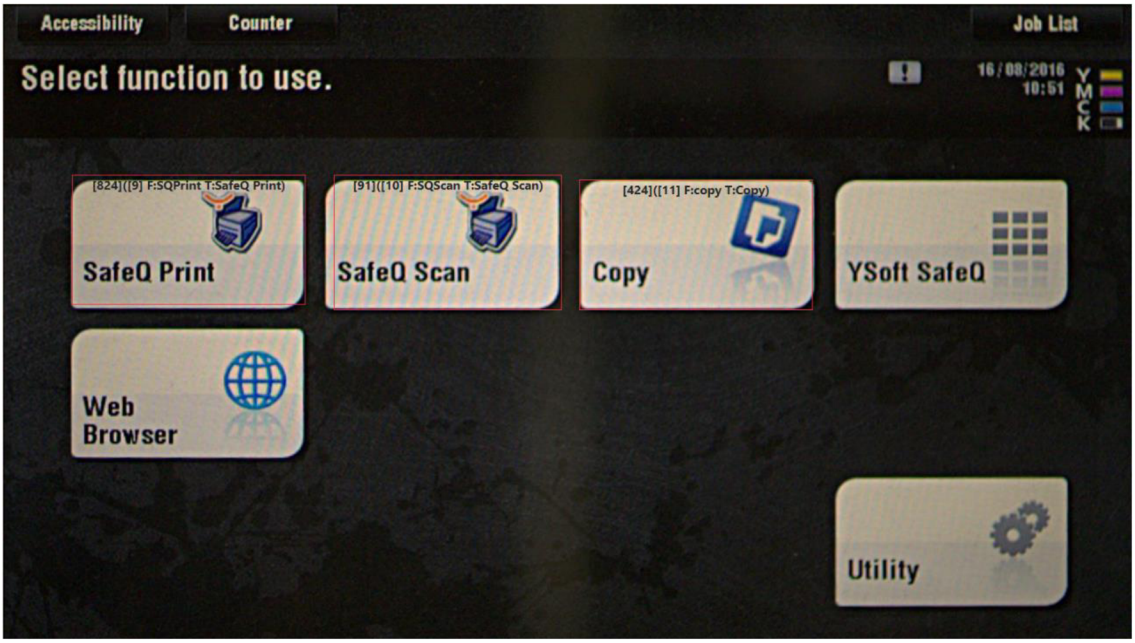
První akce v testu je elementární, ovládací systém robota tedy z databáze zjistí, na jaké pozici se nachází požadované tlačítko (podle testovaného zařízení) a robot na nej poklepe.

Druhá a čtvrtá akce jsou elementární, image processing vrátí jméno obrazovky, kterou vyhodnotil, což se srovná s požadovanou. Pokud si neodpovídají, opakuje se požadavek na image processing po maximální dobu (timeout), po níž by při nesplnění vyhodnotil chyba a test by skončil.

Obr. 3-4: Obrazovka klávesnice s vyznačenými tlačítky

Třetí akce je již složená, PIN může být například 2345, což se z testů předá řídicímu systému robota (robot control), a ten řetězec rozdělí na jednotlivé znaky. Každý znak pak reprezentuje jedno tlačítko na klávesnici, na které se musí poklepat. Program Robot control tedy vyvolá jednotlivé akce kliknutí na tlačítko pro každý znak a jeden za druhým pošle příkaz robotovi na poklepání (známá tlačítka na Obr. 3-4).

Pátá akce je už složitější, pro robot control se poskytne jméno obrazovky, na kterou se má navigovat. Díky image processingu se zjistí aktuální obrazovka. Ku příkladu je situace taková, že na terminálu je obrazovka s hlavním menu a cílem je dostat se na obrazovku YSoft SafeQ Print. K nalezení cesty je třeba v databázi předem definovat tlačítka, jak spojují dané obrazovky, vlastnost každého tlačítka je jeho zdrojová obrazovka a cílová obrazovka, na kterou odkazuje. Pro účel vyhledávání je využit algoritmus prohledávání stromu, tzv. *Depth first search* (prohledávání do hloubky). Výsledkem tohoto algoritmu je orientovaný seznam tlačítek, která robot v daném pořadí poklepe, aby se dostal na cílovou obrazovku, která se samozřejmě ověří pomocí image processingu. Algoritmus nazývám *GoToScreen(screenName).*



Obr. 3-5: Obrazovka hlavního menu s definovanými tlačítky

Šestá akce sdružuje akce poklepání na pozici tiskové úlohy ze seznamu na obrazovce a ověření, že je úloha vybrána. Člověku taková akce připadá jako samozřejmost, ale robot a image processing přirozeně nerozumí, co to znamená „vybrat tiskový job“, image processing má naprogramovaný algoritmus pro analýzu seznamu jobů a určení, které jsou vybrány. Robot control na tuhle znalost reaguje opakováním akce (při nesprávném vybrání) nebo pokračováním v případě úspěchu (naznačeno na obr. 5).

Sedmá akce předpokládá dosud nezmíněnou klíčovou součást RQA – senzory papíru. Opět nebudu podrobně rozebírat princip jejich fungování, pro mou potřebu je důležité, že rozpoznají papír vycházející z tiskárny (jsou umístěny těsně za výstupními válci vytlačující papír) a rozliší jeho velikost, zda je A4 nebo A3. Nasnímané papíry si pamatují a jsou dostupné přes REST API, na začátku testu se jejich mezipaměť vyčistí, aby na konci obsahovaly pouze počet z aktuálního testu. Robot control si tedy požádá o nasnímané papíry, srovná s požadovanými a vyhodnotí výsledek.

Osmá a poslední akce lze realizovat open pomocí algoritmu GoToScreen(screenName), kde cílová obrazovka je výchozí obrazovka před přihlášením. Důležité je zkontrolovat, že se terminál dostal do výchozího stavu, pro zajištění atomicity testovacího scénáře.

3.5 Základní akce

Do Robot control programu jsem definoval základní akce, které robot dokáže provádět a díky kterým by mělo být možné realizovat kterýkoliv testovací scénář. Tyto akce jsem stanovil takto:

Task TapButton(string buttonName);
Task SwipeArea(string buttonName, string direction);
Task VerifyScreen(string screenName);

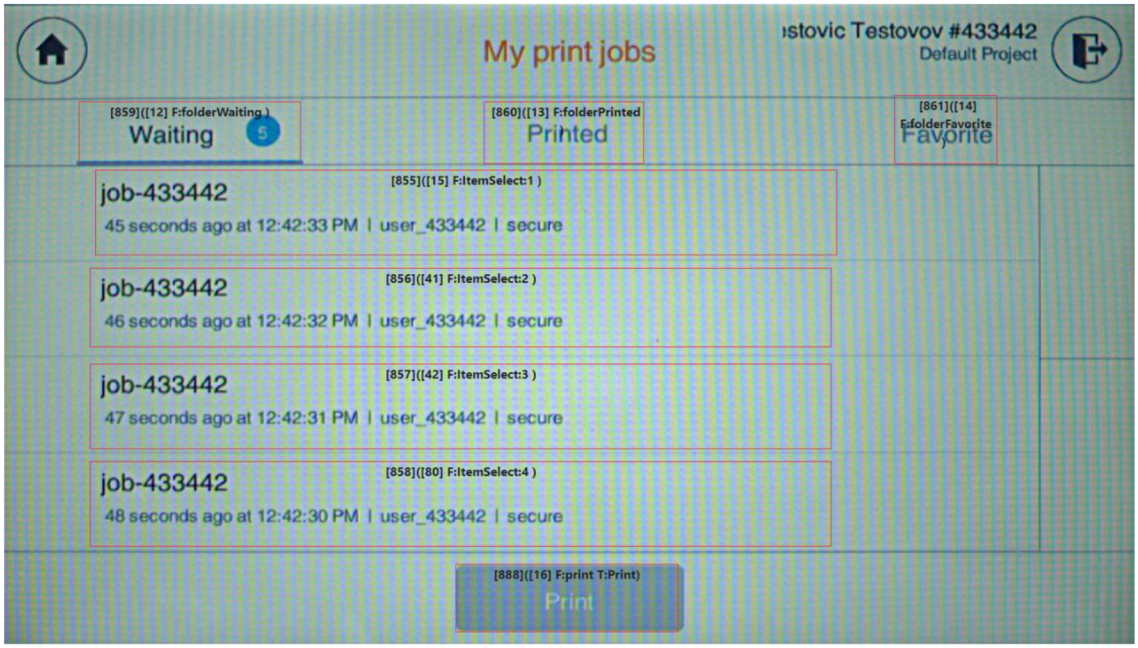
```
Task GoToScreen(string screenName);
Task VerifyPapers(string paperType, string paperCount);
Task VerifyPreview(string color);
Task VerifyItemName(string name, string position);
Task FindItemInList(string name);
Task VerifyTextInRegion(string text, string regionName);
Task SelectItemsInList(string positions, string names);
Task EmulateCard(string code);
```

3.6 Požadavky na testovací scénář

Nyní jsou známy aktuální možnosti RQA, dostáváme se k rozboru požadavků na splnění dalších cílů této práce. K vytváření testovacích scénářů je třeba rozhraní pro uživatele, přes které definuje průběh testu, dále je třeba scénář uložit spolu s dalšími informacemi, umožnit spouštění testovacích scénářů a vytvořit vizualizaci výsledků testů. Součástí testovacích scénářů jsou také měření různých vlastností, ze kterých se vyvozuje kvalita testovaného systému, těmi se budu zabývat následně.

Přístup k automatizaci testovacích scénářů může být tvorba na základě keywordů nebo capture-replay. Mezi výhody prvního způsobu patří tvorba znovupoužitelných keywordů, které zajišťují určitou ohraničenou akci a jejich skládání do test case podle potřeb. Takto navržené testy bývají méně náchylné na zastarávání, ale pokud je vyžadována změna v keywordu, musí být provedena s ohledem na všechny testy, v nichž se daný keyword používá. Zásah do hotových scénářů je poměrně rizikový a časově náročný, stejně jako samotná tvorba test case.

Hlavní výhody capture-replay jsou snadný návrh test case, kdy kdokoliv znalý používání testovaného systému dokáže nahrát tyto akce, aby je automatický tester zaznamenal a dokázal kdykoliv spustit nad testovaným systémem. Úprava testu vyžaduje opakování fáze capture, nebo alespoň její části, ovšem test cases jsou izolované, takže jedna změna ovlivní pouze daný test. Tento přístup je vhodnější pro testování GUI, protože je vizuálně jasné, co se během testu děje.

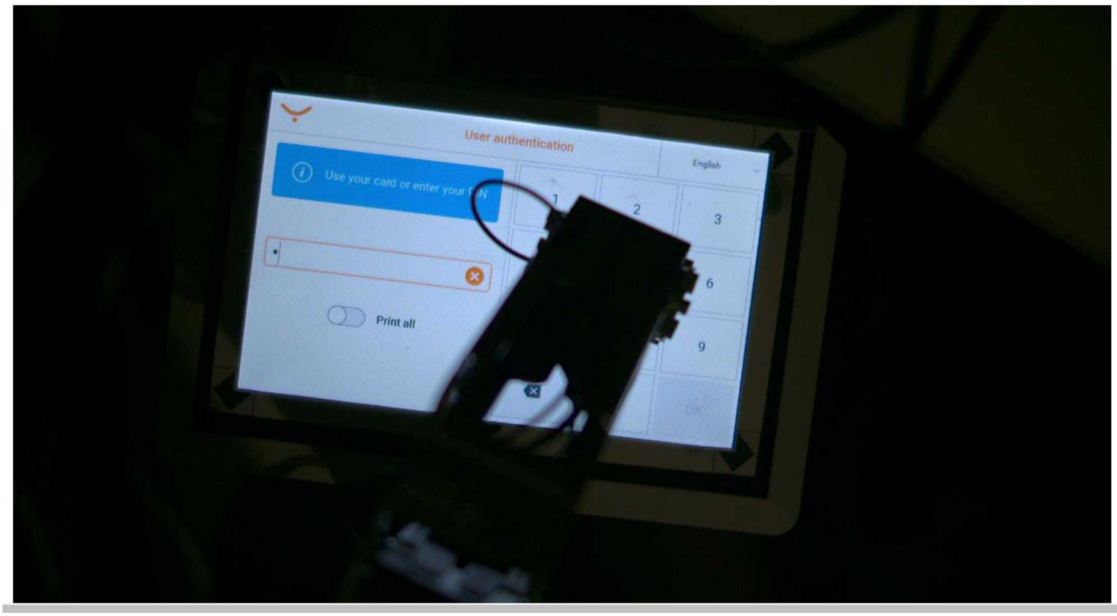


Obr. 3-6: Obrazovka SQ Print s vyznačenými tlačítky

Pro svou práci jsem se rozhodl vyzkoušet dva různé způsoby pro tvorbu automatizovaných testů a následně srovnat, který je vhodnější pro danou aplikaci testování GUI přes rozhraní na embedded zařízení. První by měl vycházet z keyword-driven přístupu, jak je obecně chápán a druhý jej modifikuje se zavedením vizuálního návrhu se zobrazením stavu testovaného zařízení v reálném čase. Operátor testů vytvoří testovací scénář pomocí navigace robotického manipulátoru na pozice tlačítek potřebných k realizaci testu. Dále definuje verifikační akce, které je potřeba provést (ověření obrazovky, konkrétního textu). Idea je taková, že operátor vizuálně uvidí, co se v testovacím scénáři děje a samozřejmě může celý test nebo jeho část upravit dle potřeby.

3.7 Předpoklady

Obě varianty mají několik společných předpokladů, které je třeba mít na paměti. Před vytvářením jakéhokoliv testu musí operátor definovat všechny ovládací prvky a obrazovky, se kterými lze během interakce se zařízením přijít do styku. To znamená nezanedbatelnou manuální práci, kdy je třeba připravit testované zařízení a umístit před něj robotický manipulátor a kameru tak, aby manipulátor měl ve svém operačním dosahu všechny ovládací prvky a kamera ve svém záběru zachycovala celý displej zařízení. Nemusí zachycovat nic jiného okolo, ale záleží na operátorovi, zda chce mít ve videu více informací. Robotický systém totiž během celého testu natáčí video sloužící jako záznam z testování pro analýzu výsledku testů.

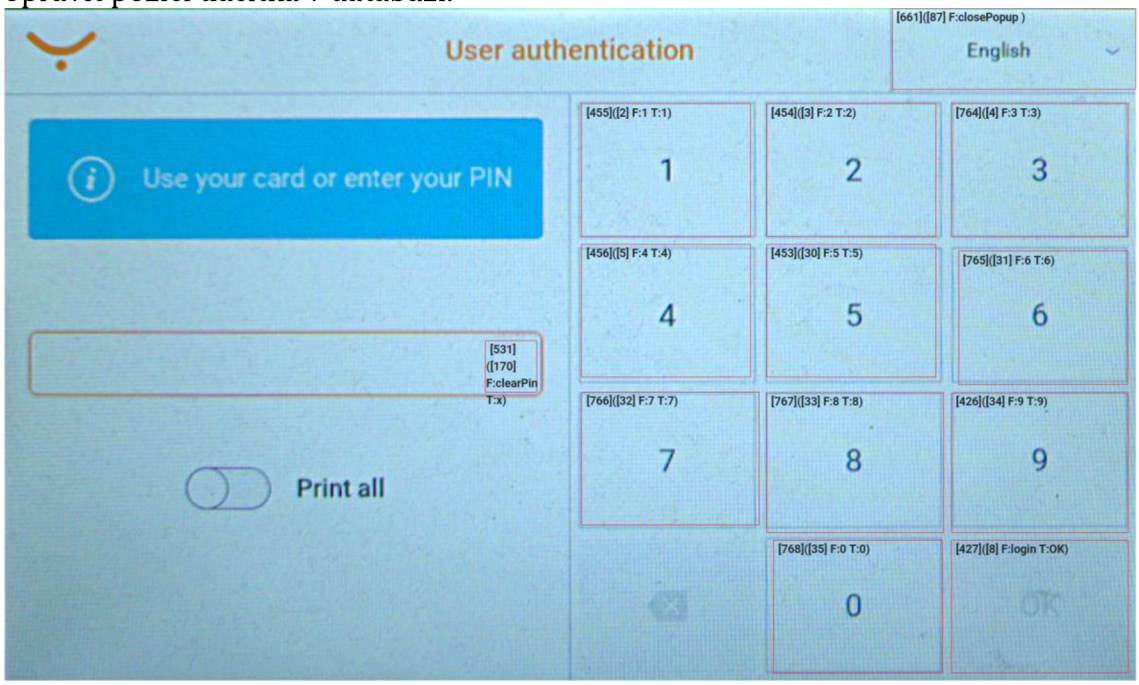


Obr. 3-7: Příklad nastavení robota a kamery před zařízením

Operátor tedy použije webové rozhraní pro přidávání nových referenčních snímků obrazovek. Nastaví si obrazovku na zařízení a získaný snímek z kamery uloží pod zvolenou abstrakcí obrazovky. Aktuálně musí být přítomen blízko zařízení se svým přenosným počítačem, aby po nastavení jiné obrazovky na zařízení mohl získat snímek z kamery a opět uložit k jiné abstrakci obrazovky. Není nutné definovat výlučně všechny obrazovky, nutné jsou ty, na kterých se testuje funcionalita a dále také obrazovky chybových stavů, do kterých se zařízení potenciálně může dostat. Zde je vhodné si uvědomit, že pokud se během testování dostane testované zařízení do stavu, který RQA nezná (není taková obrazovka definována), nemůže testování úspěšně pokračovat a daný test tedy skončí s chybou. Taková chyba je obvykle považována za false-positive, protože skutečně nejde o chybu na testovaném zařízení, ale omyl provedl operátor testování když nedefinoval takovou obrazovku.

Dalším krokem je definování všech klikatelných elementů (tlačítek), opět nutné jsou ty, na které má být robot schopen během testování poklepat. To se provádí také ve webovém rozhraní RQA v sekci Visual Editor na konkrétní vybrané obrazovce vybraného zařízení. Pouhým označením oblasti tlačítka se zvolí velikost a pozice tlačítka a následně se přidají další informace, tedy jméno abstrakce tlačítka, cílová obrazovka (na kterou tlačítko odkazuje). Po potvrzení se uloží data o tlačítku dot databáze. Je možné definovat také fiktivní tlačítka, tedy oblasti, kde se sice nenachází žádný klikatelný prvek, ale může být užitečné mít možnost sem kliknout. Ze zkušenosti mohu říci, že je to užitečné zejména při probouzení zařízení z klidového režimu nízké spotřeby nebo při swipování (přejíždění prstem po obrazovce). Vyplývá z toho, že RQA nekontroluje, že definovaná tlačítka jsou umístěna správně, vše je tedy na zodpovědnosti operátora, což s sebou nese riziko omylu. Robotický manipulátor během testování poklepává přesně na pozice definovaných tlačítek a pokud zařízení správně nereaguje a nezmění svůj stav očekávaným způsobem, je vyhlášena falešná chyba. Taková chyba se následně

obtížně detekuje a k jejímu odhalení je třeba využít videozáznamu, a pak manuálně opravit pozici tlačítka v databázi.



Obr. 3-8 Definování tlačítek na zařízení

4. ELABORACE

V této kapitole se zabývám definováním formátu testů a vlastnostmi, které musí test mít, aby vyhovoval požadavkům na testování software YSoft SafeQ a zároveň odpovídal schopnostem robotického testovacího systému.

4.1 Technologie

K vytváření testovacích scénářů je třeba řada kroků. Testovací scénář je vázán na konkrétní multifunkční zařízení s připraveným systémem YSoft SafeQ. U tohoto testovaného zařízení (v našem případě tiskárny, ale obecně lze vytvořit test pro jakékoliv zařízení splňující předpoklady pro testovatelnost) je umístěn robotický manipulátor a také kamera namířená přesně na displej tiskárny. Pro další postup je vyžadována kontrola nad robotickým manipulátorem (pomocí software Robot control), takže uživatel zadá příkaz přes ovládací rozhraní (v tomto případě REST API) a robot ji vykoná. K uživateli se zpět dostane informace, zda se příkaz robotovi provedl správně či nikoliv. Pro podrobnější zpětnou vazbu a tvorbu scénáře vůbec je klíčové získávání snímků z kamery a jejich streamování uživateli. Uživatel tedy bude přesně vědět, co se na tiskárně děje, uvidí akce robotického manipulátoru a jejich odezvu na tiskárně.

V průmyslu se pro ovládání robotických manipulátorů používá speciální hardware, ovládací panel, který umožňuje různý způsoby jeho ovládání. V možnostech manipulátoru je nastavení pozice koncového efektoru v souřadném systému (x, y, z) + úhlová orientace, z čehož se dopočítají úhly natočení jednotlivých kloubů. Případně bývá možné ovládat přímo natočení kloubů k dosažení požadované pozice. V našem případě je vstupem robotickému manipulátoru pozice na obrazovce (x, y), kterou robot díky kalibraci převede do svého souřadného systému (x, y, z, α)[1]. Předpokladem pro získání souřadnic určitého bodu na displeji tiskárny je znalost rozměrů aktivní části displeje, která musí být zadána dopředu v databázi RQA.

4.2 Formát testovacích scénářů

4.2.1 Hlavička a metadata

Testovací scénář musí obsahovat řadu informací, které se přímo nevztahují k průběhu testu, ale jsou klíčové k identifikaci účelu, využití a kompatibility testu. Už samotné pojmenování testu musí být zvoleno pečlivě, aby z něj zasvěcená osoba odhadla, jak asi vypadá, co vlastně testuje a jaký je jeho ekvivalent při manuálním testování. Například pojmenování „Print test“ je naprosto nevhodné, protože při testování YSoft SafeQ se velká část testů týká tisku. Název „*Secure print of a single page from SafeQ Print*“ je mnohem deskriptivnější, lze z něj odhadnout průběh, počet vytisknutých stran atd. Ovšem název testu je plně v kompetenci uživatele/vývojáře, který může téměř zadat cokoliv a probíhá jediná validace vstupu, tedy unikátnost. Každý testovací scénář na určité zařízení by měl mít unikátní jméno, aby se dal i tímto jednoznačně určit. Součástí návrhového

prostředí tedy může být sada doporučení, jak by měl název vypadat včetně kladných i záporných příkladů.

Další klíčový údaj pro testovací scénář je testované zařízení (tiskárna), se kterým je kompatibilní. Může jít i o více zařízení, pokud jsou dostatečně podobná. Tvůrce testu tento parametr přímo určí na počátku, než začne vytvářet test, výběrem z RQA databáze.

Priorita testovacího scénáře určuje, jak důležité je spuštění testu v rámci kontinuálního vývoje a také jak kritické je, aby daný test dopadl úspěšně. Některé testy mohou být naprosto kritické a jiné spíše okrajové. Hodnota tohoto parametru je číslo v rozmezí 1 až 4, kde 1 znamená naprosto kritický test a 4 je nejmenší priorita. Tento údaj je také užitečný při vyhodnocení testů, pokud jich mnoho skončilo chybou, měly by se zkoumat jejich příčiny a navrhnout opravy právě v pořadí podle jejich priority.

Datum a čas vytvoření - to údaj testy poslouží ke zjištění, za jak dlouho průměrně test zastará a je třeba jej opravit, případně ztratí na své hodnotě a je smazán.

Stav testovacího scénáře – může být aktivní/neaktivní, tento údaj určuje, zda je test v pořádku a měl by být spouštěn nebo existuje nějaký problém, který nemůže být okamžitě opraven. V takovém případě uživatel změní stav na neaktivní a vyhodnotí, co je třeba k jeho opětovné aktivaci.

Požadavky na periferie – pokud testovací scénář vyžaduje např. senzory papíru, emulátor karty nebo jinou součást, je třeba ji přidat do požadavků. Před spuštěním testu se ověří, zda jsou tyto periferie k dispozici a na základě toho se pokračuje nebo se test vůbec neprovede. Tyto údaje je třeba vybrat z databáze RQA.

Verze testovaného prostředí – důležitý údaj říká, pro kterou verzi testovaného software (ať už YSoft SafeQ nebo jiného) je test navržen. Může existovat mnoho instancí stejného testu pro různé verze lišící se v maličkostech, ale přesto musí být definovány zvlášť.

Verze RQA – i robotický testovací systém je součástí vývoje, proto v určitém okamžiku může přijít kritická změna, která znemožní spuštění starších testovacích scénářů. Údaj o verzi robot control umožní zjistit, se kterou verzí byl test naposledy kompatibilní. Dále se může nasadit tato verze, aby mohl test být spuštěn nebo se upraví samotný test, aby fungoval na nové verzi.

4.3 Definice testovacího scénáře

Samotná data testovacího scénáře obsahují jeho průběh, jak byl definován. Jednotlivé kroky jsou jakékoliv akce, které je robot control v definované verzi schopen vykonat. Samozřejmostí jsou akce jako TapButton(buttonName), VerifyScreen(screenName) nebo GoToScreen(screenName). Tyhle jednotlivé akce spolu se svými argumenty jsou za sebou v uspořádaném seznamu. Testovací scénář rozebíraný dříve by tedy vypadal následovně:

```
{
 "Version": "6.0.0.0",
 "DeviceModelId": 1,
 "TestName": "Secure Print",
 "Priority": "Normal",
```

```
"TestCategory":"Regression tests",
"TestActions": [
{
   "Step": 1,
   "ActionType":"Basic",
   "ActionName": "VerifyScreen",
   "Arguments": {
     "screenName": "Login Screen"
   }
 }
 {
   "Step": 2,
   "ActionType":"Basic",
   "ActionName": "Emulate Card",
   "Arguments": {
     "code": "1234567890ABCDEF"
   }
 },
   {
   "Step": 3,
   "ActionType":"Basic",
   "ActionName": "VerifyScreen",
   "Arguments": {
     "screenName": "Main Menu"
   }
 },
   {
   "Step": 4,
   "ActionType":"Basic",
   "ActionName": "GoToScreen",
   "Arguments": {
     "screenName": "SQ Print"
   }
 },
   {
   "Step": 5,
   "ActionType":"Basic",
   "ActionName": "SelectPrintJobs",
   "Arguments": {
     "jobCout": 1
   }
 },
   {
   "Step": 6,
   "ActionType":"Basic",
   "ActionName": "VerifyPapers",
   "Arguments": {
```

```
      "paperCount": "1",
      "paperType": "A4Simplex"
     }
    },
     {
     "Step": 7,
     "ActionType":"Basic",
     "ActionName": "GoToScreen",
     "Arguments": {
      "screenName": "Login Screen"
     }
    }
   ]
}
```

4.4 Uživatelsky definované akce

Takový zápis testu je poměrně pracný na vytvoření, a obsahuje akce, které by bylo vhodné využít častěji podobně, jak je tomu v keyword-driven návrhu testů. Představuji tedy koncept Komplexních akcí, v podstatě tedy uživatelsky definovaných akcí. Komplexní akce může obsahovat libovolný počet základních akcí, ale ne jiné komplexní akce. Komplexní akce nemá žádné argumenty, je to v podstatě obalení základní akcí do jednoho bloku. Testovací scénář potom může obsahovat libovolné komplexní akce nebo základní akce s libovolnými argumenty. Jedna komplexní akce je pevně definovaná s konkrétními argumenty. Komplexní akce je definována pro určitý model zařízení a určitý typ terminálu. Dále je platná pouze pro určitou verzi testovaného systému. A samozřejmě obsahuje seznam základních akcí a jejich argumentů. Příkladem komplexní akce může být následující zápis ve formátu json:

```
{
 "actionName": "ExperimentAction",
 "deviceModelId": 4,
 "actionType": Complex,
 "Terminal": 2,
 "version": 1.0.0.0,
 "Actions": [
  {
   "step": 0,
   "actionType": "Basic",
   "actionName": "TapButton",
   "arguments": {
    "buttonName": "1"
   }
  },
  {
   "step": 1,
   "actionType": "Basic",
```

```
      "actionName": "TapButton",
      "arguments": {
       "buttonName": "2"
      }
     },
      {
      "step": 2,
      "actionType": "Basic",
      "actionName": "TapButton",
      "arguments": {
       "buttonName": "login"
      }
     },
      {
      "step": 3,
      "actionType": "Basic",
      "actionName": "VerifyScreen",
      "arguments": {
       "screenName": "Main Menu"
      }
     }
    ]
}
```


Tento formát je automaticky převoditelný do třídy definující komplexní akci pomocí deserializace jsonu. Drobným problémem je pouze fakt, že jméno základní akce a jejích argumentů je v řetězci, který musí přesně odpovídat jménu metody, která danou základní akci implementuje. Proto není vhodné vytvářet takto testovací scénář nebo akci manuálně, není zajištěna integrita dat a validita testu jako takového. Vytvořil jsem tedy intuitivní rozhraní pro uživatele, které popisuji v kapitole 5.

4.5 Uložení

Pro uložení testovacích scénářů jsem zvažoval několik variant, buď uložení v Git repositáři ve formátu definovaném výše nebo využití databáze. Uložení v databázi má řadu výhod, zejména okamžitý přístup bez nutnosti stahování celého repositáře s testy a okamžitá editace bez nutnosti pull-requestu. Toto je ovšem z jistého pohledu nevýhoda, neboť okamžitá editace s sebou nese velkou zodpovědnost člověka provádějící editaci, protože okamžitě ovlivní i probíhající testy. Na druhou stanu existuje předepsaná struktura uložených dat, takže si ji nemůže každý libovolně upravoval, což zajišťuje integritu a validitu testovacích scénářů.

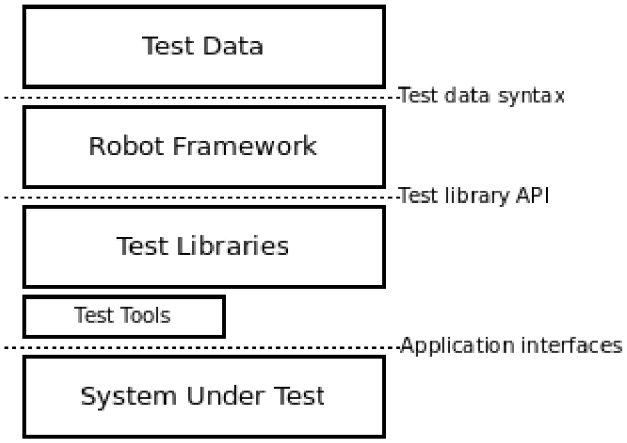
5. IMPLEMENTACE ROZHRANÍ PRO TESTY

V této kapitole spojím teoretické poznatky o testování se schopnostmi automatického testovacího systému RQA a požadavky na testování systému YSoft SafeQ, z čehož vyplyne, jak má vypadat způsob vytváření testovacích scénářů a zobrazování jejich výsledků.

5.1 Robot framework

Nejdříve jsem vytvořil testovací prostředí s využitím existujícího frameworku jménem Robot Framework [10]. Navzdory názvu ve skutečnosti nemá nic společného s roboty jako takovými a určitě nemá nic společného s RQA. Robot Framework je open source prostředí určené převážně pro automatizované regresní a akceptační testování a test-driven development. Využívá keyword-based tvorby testů, takže každá akce je tzv. keyword, který má vymezenou funkcionalitu a zodpovědnost za ni, a je použitelný v testovacím scénáři. Uživatelé tedy vytvářejí vysokoúrovňové keywordy, které jsou tvořeny keywordy se základními akcemi a z nich skládají testovací scénář, který je snadno čitelný a zřejmý pro každého.

Robot Framework je nezávislý na použité technologii ani aplikaci. Je třeba vytvořit testovací knihovny s implementacemi konkrétních keywordů v Pythonu nebo Javě, což jsou nativně podporované jazyky.



Obr. 5-1: Architektura Robot Frameworku [15]

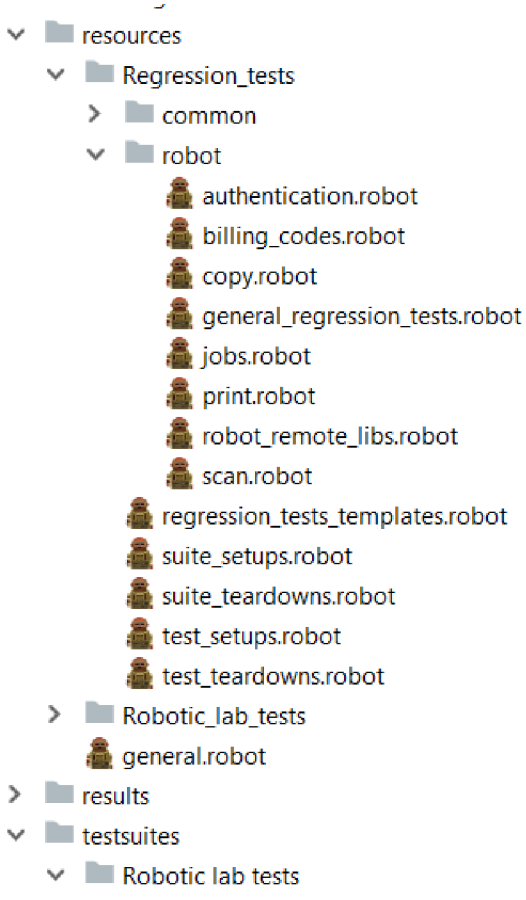
Je doporučeno používat syntaxi Gherkin, která jasně rozlišuje, která část testu je příprava, co patří do samotného testu a jak se ověřuje správnou provedení testu. Klíčová slova jsou Given, When, And, Then. Syntaxe potom vypadá:

Given Příprava testu
And Další příprava
When Hlavní část testu, kterou se vlastně testuje
And Pokračování hlavní části
Then Co musí být splněno, aby se test považoval za splněný

Tato syntaxe je obdobou přístupu Arrange, Act, Assert používaného převážně v Unit testech v různých programovacích jazycích, například v C#[14].

Příhodné je, že každý test má volitelný Setup a Teardown, takže během Setupu se připraví vše potřebné pro spuštění testu a na konci v Teardownu se obvykle testovací prostředí uvede do původního stavu. S využitím akcí, které jsem naprogramoval do řídicí aplikace Robot control, jsem vytvořil řadu vysokoúrovňových keywordů. Každý má určitou zodpovědnost za provedení jedné nebo více operací s testovaných zařízením a následně zkontrolování úspěšnosti této operace. Pokud není možné operaci dokončit nebo její výsledek je neočekávaný, vznikne chyba a test se ukončí. Spustí se tedy teardown testu, protože platí, že se testovaný systém musí uvést do původního stavu, chyba jednoho testu nesmí způsobit chyby dalších testů.

Testy je možné spouštět samostatně, více vybraných testů podle názvu nebo například celou test suite. Robot framework automaticky vytváří příhodný záznam o průběhu a výsledku testů ve formátu html. Ten poskytuje přehled o provedených testech a případných chybách s údaji, ve kterém keywordu nastaly a jejich chybových hláškách.



Obr. 5-2: Struktura keywordů použitelných v testech

Pro využití výhod Robot frameworku je vhodné rozdělit testovací keywordy to kategorií podle jejich oblasti použití, aby mohl QA inženýr snadno vytvořil nové testy s využitím už existujících keywordů. Všechny jsou ve složce resources. Vytvořil jsem tedy kategorie general, print, copy, scan, authentication, billing_codes. Už podle názvu je patrné s jakou částí testovaného systému pracují.

Samotné test cases jsou ve složce testsuites a dále rozdělené do souborů se skupinami testů, které spolu souvisejí. Mezi tyto kategorie patří Regresní testy, testy na různé tiskové úlohy, testy na kopírování různých typů a velikostí dokumentu, testy na měření doby odezvy atd. Na příkladu zápisu testu níže vysvětlím princip tvorby testů a účel každého keywordu.

Secure print from Favorite folder with accounting

[Tags] robotRegressPrint Brother_unstable Epson_manual FX_XCP KM_1stGen KM_2ndGen Lexmark_unstable OKI_unstable Sharp TP4_manual Xerox_2ndGen

Given User exists in SafeQ
And User has jobs in secure queue
And Paper sensor data are cleared
When Robot authenticates with PIN
And Robot selects jobs as favorite
And Robot prints jobs folder=Favorite
And Robot logs out
Then Job has expected status in reports
And Job has been accounted pages=${TOTAL_PAGES} price=${JOB_PRICE}

Za názvem testu je sekce Tags, ve která lze definovat označení testu pro jeho zařazení do určité skupiny. V mém případě se toto označení využívá pro určení zařízení (tiskárny), pro které je tento test určen. Tohle je velice výhodné při spouštění testů, kdy se mají spustit všechny testy pro zařízení např. KM_1stGen, pak se do příkazu zahrne tento tag v syntaxi *test -i KM_1stGen*. Je zde vidět i zařazení do kategorie *robotRegressPrint*, což umožňuje spuštění pouze testů v této kategorii, tedy testující samotný tisk na zařízení. Tagy může tvůrce testů definovat libovolně, jedná se pouze o označení textovým řetězcem.

V hlavní části testu je struktura Gherkinu, tedy akce User exists in SafeQ uvozená klíčovým slovem Given. Tohle zajišťuje, že pro daný test je automaticky vytvořen nový uživatel s unikátními údaji (uživatelské jméno, PIN, e-mail). Následuje další přípravná akce User has jobs in secure queue. To spočívá v odeslání tiskové úlohy na server YSoft SafeQ právě pod nově vytvořeným uživatelem. Zde stojí za zmínku, že informace o uživateli nejsou v tomto zápise pro vyšší přehlednost, ovšem jsou uloženy v proměnných vytvořených před začátkem testu v Test Setupu. Robot framework obsahuje více úrovní platnosti proměnné, jsou to lokální, test-case a test-suite. Lokální proměnná platí pouze pro aktuální keyword, testová má platnost pro celý jeden test-case a musí být vyvořena syntaxí *Set test variable ${jmeno} hodnota*. Obdobně existuje keyword *Set suite variable ${jmeno} hodnota*, který vytvoří proměnnou platnou pro celý test suite.

Poslední přípravnou akcí v testovacím scénáři je *Paper sensors are cleared*. To zajišťuje, že senzory papíru jsou vynulovány a připraveny snímat papíry vytištěné ze tiskárny.

Následně začíná sekce When, kdy se provádí úkony klíčové pro otestování funkcionality. Prvním je keyword realizující přihlášení robota na terminálu multifunkčního zařízení. Keyword se jmenuje výstižně Robot authenticates with PIN a právě tahle akce se při jeho zavolání provede. Co vlastně znamená přihlásit

se PINem musí zadat operátor kdykoliv před spuštěním testu do systému do systému v podobě flow. Jak jsem uvedl v kapitole 3, flow je v podstatě posloupnost tlačítek nutných k vykonání dané akce s funkcí opakování akce v případě neúspěchu. Pokud je flow typu autentizace, je v robotovi naprogramováno, že má zkontrolovat správnost přihlášení. To je realizováno rozpoznáním obrazovky (pomocí Image processingu) a zkontrolováním meta-dat dané obrazovky, zda má aktivní příznak „Autentizováno“. I zde jsem použil opakování rozpoznání v případě neúspěchu, pokud se ovšem podaří rozpoznat přihlášenou obrazovku, považuje se daný keyword za splněný.

Další akcí je tedy označení tiskové úlohy jako oblíbené (favorite). K tomu se využijí známe akce popsané v kapitole 3 – navigace na obrazovku SQ Print, ověření přítomnosti tiskové úlohy a následně její označení jako oblíbené.

```
Robot selects jobs as favorite
    [Arguments]  ${job_count}=1    ${position}=1    ${folder}=Waiting
    Go To Screen  SQ Print
    Tap Button    folder${folder}
    ${jobExists}    Verify Item Name    ${JOB_NAME}    ${position}
    Should be true  ${jobExists}    Job with name ${JOB_NAME} is not at position
${position}
    Run Flow    MarkUnmarkJobFavorite
```

Následuje keyword *Robot prints jobs* s argumentem *favorite* jako jménem složky. To ukazuje, jak intuitivní a čitelný je zápis testu ve spojení se schopnosti robota jako takového.

```
Robot prints jobs
    [Arguments]    ${job_names}    ${positions}    ${job_type}=${JOB_TYPE}
${total_pages}=${TOTAL_PAGES}    ${flush_buffers}=${True}    ${folder}=Waiting
    Prepare for print    ${folder}
    Select Item Names at Positions and Do Action    ${job_names}    ${positions}
print
    Papers are printed    ${job_type}    ${total_pages}
```

Poslední akcí v této fázi je keyword Robot logs out, který obsahuje příkaz pro navigaci robota na výchozí obrazovku testovaného zařízení. Známým algoritmem *Go To Screen* tedy robot najde cestu na tuto obrazovku a ověří, že odhlášení proběhlo úspěšně.

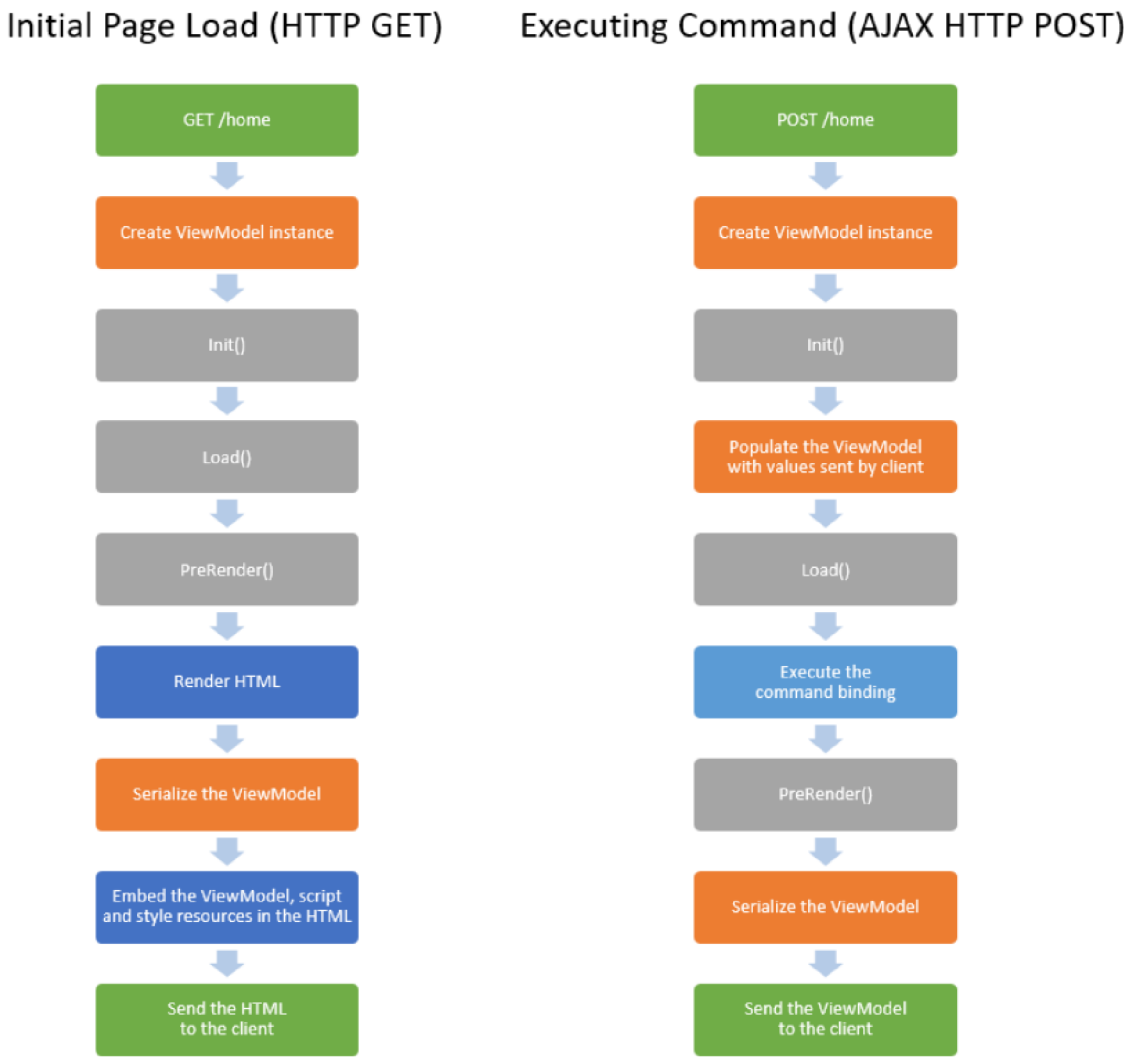
Nakonec je tu ověřovací fáze Then, která má dvě akce *Job has expected status in reports* a *Job has been accounted*. Ty slouží k ověření, že v systému YSoft SafeQ se zaznamenala akce vytištění tiskové úlohy a byla zaúčtována správnému uživateli. Tímto tedy končí testovací scénář jako takový, následuje Teardown, ve kterém se ještě jednou ověří, že je zařízení ve výchozím stavu a ukončí se test z pohledu RQA zavoláním akce *FinishTest*.

5.2 Vlastní návrh testů

Přístup keyword-driven tvorby testů pomocí Robot Frameworku má také řadu nevýhod, zejména poměrně vysokou časovou náročnost tvorby a ladění testů a také fakt, že samotné RQA nemá přehled o tom, jak vypadá celý test a jak na sebe jednotlivé akce navazují. Tudíž může být problematické vyšetřit, v čem skutečně nastala chyba, protože není znám kontext celého testu. RQA totiž poskytne jenom obecnou chybu, skutečnou příčinu musí vyšetřit člověk. V tomto případě plní roli pouze nástroje, jímž se pracuje s testovaným zařízením a vizuálně se vyhodnocuje výsledek. Tímto inovativním způsobem tvorby testů chci přispět k povýšení významu RQA na vyšší zodpovědnost za provedení testu jako celku a poskytnutí dalších výhod s tím spojených.

5.3 Použité technologie

Editor testů jsem naprogramoval v jazyce C# a v .NET frameworku. Pro webovou aplikaci jsem použil framework DotVVM[16], který usnadňuje budování díky předpřipraveným komponentům a nativně obsluhují komunikace mezi serverovou a klientskou částí. Využívá přístupu VVM – tedy View, View Model, Model. V implementaci je tedy třeba vytvořit část View za použití syntaxe HTML. Vylepšení je navázání přímo na konkrétní hodnoty nebo příkazy pomocí syntaxe {value: Property} nebo {command: Method}. Tyto jsou navázány na ViewModel, což je prostá třída v C# se svými metodami a propertami. Další výhodou je, že server neudržuje viewmodely v paměti, ale jen po dobu požadavku od klienta, např. přidání dat do databáze, a pak provedené změny promítne do nového viewmodelu, který odešle klientovi. Kompletní posloupnost kroků je na obr. 4-3. Příjemnou vlastností také je, že DotVVM odstiňuje vývojáře od JavaScriptu a od komunikace klienta a serveru. Samotné ViewModely slouží k udržování stavu webové stránky a všech hodnot v zadávacích polích, podmínkou tedy je, že všechny property ve ViewModelu musí být serializovatelné do JSONu.



Obr. 5-3: DotVVM – Posloupnost kroků při načítání stránky [17]

Navrhl jsem hlavní třídy ComplexActionManager a TestScenarioManager, kde jsou implementovány metody pro práci s akcemi a testovacími scénáři.

Třída ComplexActionManager obsahuje metody pro správu komplexních akcí. Mezi nejdůležitější patří metoda pro přidání nové komplexní akce do databáze. V rámci ní se ověřuje, zda uživatel správně vyplnil potřebné údaje ve formuláři, zejména jméno komplexní akce, model zařízení a typ terminálu, a samozřejmě seznam základních akcí, které definují funkci nadřazené akce. Další ověření souvisí s unikátností nově vytvářené akce, taková nesmí v databázi existovat, protože by pak nešlo mezi stejnými akcemi rozlišit. Když jsou ověření v pořádku, vytvoří se nová databázová entita s definovanými hodnotami a přidá se do databázového kontextu. Také se přidají nové entity pro vlastní základní akce a jejich argumenty. Následně se zavoláním příkazu SaveChanges teprve provedou změny v databázi. Pokud by změna v databázi nemohla proběhnout kvůli chybným datům nebo jejich návaznosti, vyvolaná výjimka se zachytí a uživatel je notifikován o neúspěšnosti této operace a příčině chyby. Může pak změnit data ve formuláři a zkusit operaci znovu.

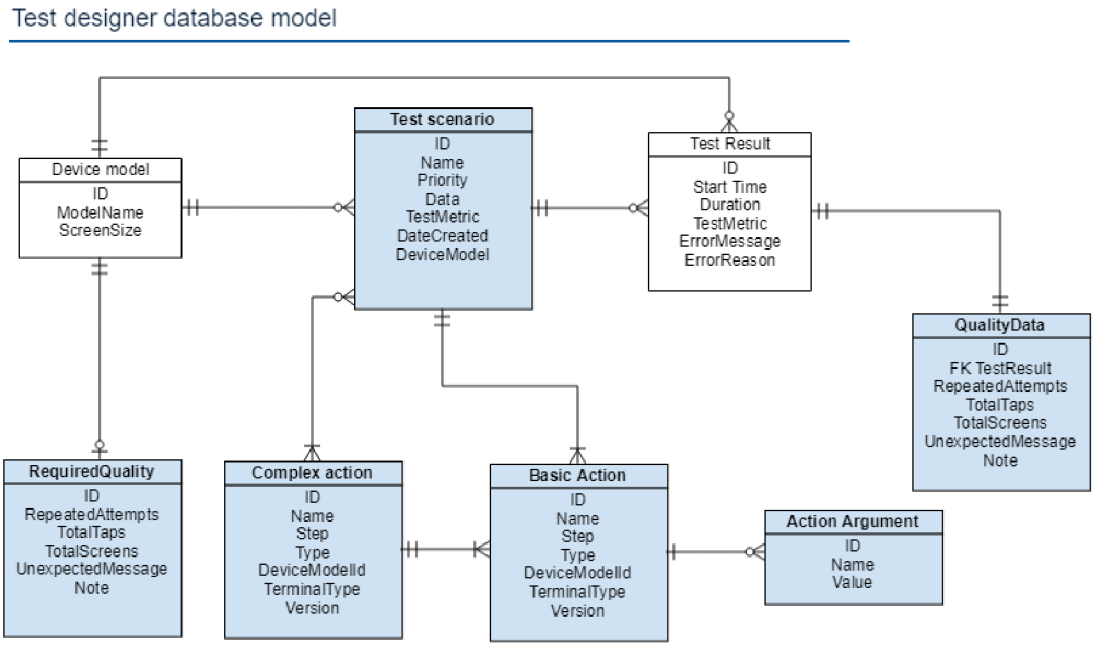
Dalšími operacemi definovanými ve správci komplexních akcí jsou editování existující akce a její smazání. Ty jsou vypovídající samy o sobě. Pomocné operace zajišťují práci se základními akcemi, tedy jejich přidání a mazání, a také úpravu pořadí a změnu hodnoty kroku. Třída pro návrh komplexních akcí udržuje řadu dat, zejména o všech dostupných základních akcích, ze kterých pak uživatel vybírá ty požadované. Dále všechny dostupné modely zařízení, tlačítka, obrazovky. Tato data jsou naplněna v metodě Load a získávají se přímo z databáze z odpovídajících tabulek.

Klíčovou částí celého návrhu je třída pro správu testovacích scénářů jménem TestScenarioManager. V ní jsem implementoval metody pro práci s testy navrhovanými přes webové rozhraní, takže samozřejmostí jsou metody pro přidávání, upravování, mazání, duplikování dat reprezentovaných v databázi. Naprogramoval jsem zajištění validních dat scénářů, takže musí obsahovat nutné informace o jménu testu, kategorii, modelu zařízení a typu terminálu. Každý test tak musí obsahovat alespoň jednu akci, buď základní nebo komplexní.

5.4 Databázový model

Pro návrh databáze používám přístupu Code-First přes Entity Framework Core [12]. Navrhuji tedy třídy, z nichž pak nechám vygenerovat migrace samotného databázového schématu a jeho dat.

Obr. 5-4 zobrazuje schéma vztahů databázových entit.



Obr. 5-4: Entity-relationship model

Hlavní entitou je TestScenario obsahující data vlastní scénáři, tedy DeviceModel. Jak bylo stanoveno, jeden testovací scénář je platný pouze pro jeden model zařízení, takže je zde cizí klíč jménem DeviceModelId odkazující na tabulku DeviceModel, sloupec Id. Další položkou je TerminalType, což je typ terminálu, který se na dané zařízení instaluje. YSoft SafeQ může pracovat v různých typech

terminálu, které mění vzhled a některé funkcionality. Nám ale stačí, že je to určité nastavení testovaného zařízení, na výběr je z pěti hodnot. Dále je zde položka typu VersionType, která souvisí s verzí testovaného systému. Výchozí hodnota je platnost přes všechny verze.

Další entitou v databázovém modelu je ActionEntity (dále akce). Jak jsem zmiňoval, akce může být dvojího typu – základní a uživatelsky definovaná. Vytvořil jsem tedy entitu ActionEntity, od které dědí entity Basic Action a Complex Action. ActionEntity má atributy ActionName, Step a ActionType. Další Atributy jsou využité na základě typu akce.

BasicAction obsahuje navíc pouze kolekci entit typu ActionArgument, což znamená, že entity z tabulky ActionArguments obsahují cizí klíč na konkrétní akci, k níž se vztahují. Basic action může mít žádný nebo mnoho argumentů podle její implementace, tento vztah se hlídá na aplikační úrovni.

Entita ActionArgument obsahuje kromě primárního klíče další dva atributy typu řetězec, jméno argumentu Name a hodnota Value. Další je již zmiňovaný cizí klíč do tabulky Actions.

Komplexní akce se liší od základní akce tím, že je definovaná pro konkrétní kombinaci zařízení, typu terminálu a verze. Vyjadřuje se tím její platnost a možnost využití. Jedna akce může platit pouze na jednom zařízení a jednom typu terminálu, verze je opět ve výchozím nastavení neomezená. Jak komplexní akce ve skutečnosti vypadá vyjadřuje kolekce základních akcí Actions. To znamená, že akce typu Basic obsahují cizí klíč na akci typu Complex. V samotné databází jsou ovšem všechny akce ve stejné tabulce, takže cizí klíč odkazuje na primární klíč téže tabulky.

Nyní je důležité definovat vztah mezi testovacími scénáři a akcemi. Protože jeden scénář může obsahovat více akcí a jedna akce může patřit do více scénářů, jedná se o relaci „many-many“, takže je třeba zavést převodní tabulku jménem TestAction. Ta obsahuje dva primární klíče TestScenarioId a ActionEntityId které korelují s primárními klíči v příslušných tabulkách se scénáři a akcemi. Dále obsahuje sloupec Step, který vyjadřuje pořadí dané akce v testovacím scénáři, přičemž správné určení pořadí zajišťuje aplikace.

Tento návrh splňuje předpoklady definované v této kapitole, kdy testovací scénář může obsahovat libovolný počet základních akcí a libovolný počet komplexních akcí.

9	23	2019-03-27 12:27:16.807039	2019-03-27 12:27:16.807039	0	1	ComplexActionEn...	Test5	[null]	5	0
10	24	2019-03-27 12:27:16.807039	2019-03-27 12:27:16.807039	1	0	BasicActionEntity	TapButton	23	[null]	[null]
11	37	2019-03-28 11:33:12.71464	2019-03-28 11:33:12.71464	1	0	BasicActionEntity	TapButton	[null]	[null]	[null]
12	38	2019-03-28 11:33:12.71464	2019-03-28 11:33:12.71464	2	0	BasicActionEntity	VerifyScreen	[null]	[null]	[null]
13	39	2019-03-28 14:22:01.870186	2019-03-28 14:22:01.870186	1	0	BasicActionEntity	TapButton	9	[null]	[null]
14	40	2019-03-28 14:22:01.870186	2019-03-28 14:22:01.870186	2	0	BasicActionEntity	VerifyScreen	9	[null]	[null]
15	41	2019-03-28 14:22:01.870186	2019-03-28 14:22:01.870186	3	0	BasicActionEntity	VerifyScreen	9	[null]	[null]
16	42	2019-03-28 14:22:01.870186	2019-03-28 14:22:01.870186	1	0	BasicActionEntity	TapButton	9	[null]	[null]
17	43	2019-03-28 14:22:01.870186	2019-03-28 14:22:01.870186	2	0	BasicActionEntity	VerifyScreen	9	[null]	[null]
18	44	2019-03-28 14:23:41.781862	2019-03-28 14:23:41.781862	1	0	BasicActionEntity	TapButton	9	[null]	[null]
19	45	2019-03-28 14:23:41.781862	2019-03-28 14:23:41.781862	2	0	BasicActionEntity	VerifyScreen	9	[null]	[null]
20	48	2019-04-03 08:30:11.32857	2019-04-03 08:30:11.32857	1	0	BasicActionEntity	TapButton	[null]	[null]	[null]
21	51	2019-04-05 10:45:58.430027	2019-04-05 10:45:58.430027	1	0	BasicActionEntity	TapButton	[null]	[null]	[null]
22	52	2019-04-05 10:45:58.430027	2019-04-05 10:45:58.430027	2	0	BasicActionEntity	VerifyScreen	[null]	[null]	[null]
23	53	2019-04-08 12:40:20.483282	2019-04-08 12:40:20.483282	1	0	BasicActionEntity	TapButton	[null]	[null]	[null]
24	54	2019-04-08 12:46:07.403972	2019-04-08 12:46:07.403972	1	0	BasicActionEntity	TapButton	[null]	[null]	[null]
25	55	2019-04-09 06:39:16.111222	2019-04-09 06:39:16.111222	0	1	ComplexActionEn...	CoolAction	[null]	8	4
26	56	2019-04-09 06:39:16.111222	2019-04-09 06:39:16.111222	1	0	BasicActionEntity	TapButton	55	[null]	[null]
27	57	2019-04-09 06:39:16.111222	2019-04-09 06:39:16.111222	2	0	BasicActionEntity	VerifyScreen	55	[null]	[null]
28	58	2019-04-09 06:39:16.111222	2019-04-09 06:39:16.111222	3	0	BasicActionEntity	TapButton	55	[null]	[null]
29	59	2019-04-09 06:39:16.111222	2019-04-09 06:39:16.111222	4	0	BasicActionEntity	VerifyScreen	55	[null]	[null]
30	60	2019-04-09 06:40:21.043619	2019-04-09 06:40:21.043619	2	0	BasicActionEntity	GoToScreen	[null]	[null]	[null]
31	61	2019-04-09 06:52:49.720271	2019-04-09 06:52:49.720271	2	0	BasicActionEntity	GoToScreen	[null]	[null]	[null]
32	62	2019-04-09 07:02:53.580224	2019-04-09 07:02:53.580224	2	0	BasicActionEntity	GoToScreen	[null]	[null]	[null]

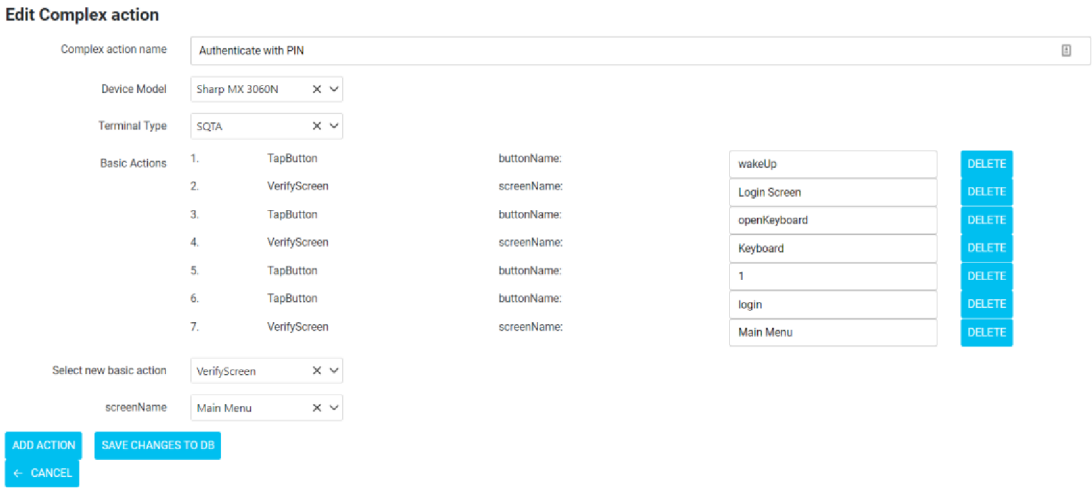
Obr. 5-5: Příklad uložených akcí v databázi

5.5 Uživatelské rozhraní

Nyní se podívejme na front-end sloužící vytváření a editování testů, kde je základem je snadná dostupnost, přehlednost a intuitivnost. Vyvinul jsem aplikaci, která komunikuje s ostatními komponenty RQA, poskytuje GUI pro práci s testy a zapisuje vytvořené testovací scénáře do databáze. Rozhraní pro tvorbu testů může pracovat ve dvou režimech – online a offline. Online režim spočívá přímo v kontrole robotického manipulátoru u konkrétního multifunkčního zařízení. Podmínkou pro tento režim je nachystaný robotický manipulátor, spárování s aplikací Robot control, platná kalibrace robota a také správně umístěná kamera, její kalibrace a spárování s aplikací Image processing.

Online režimem se realizuje inovativní kombinace metod keyword-driven a capture-replay, kdy tvůrce testu předvádí akce testovacího scénáře krok za krokem přímo na zařízení a když je s ním spokojen, na konci jej uloží do databáze. Kdykoliv lze pak na stejném zařízení spustit vytvořený scénář (i s úplně jiným robotem). Uživatel přesně vidí, co se na terminálu zařízení děje díky streamu z kamery. Kliknutím na stream se vyvolá akce kliknutí robota přímo na terminál na určené souřadnice. Taková akce se automaticky přidá do zobrazené fronty akcí na straně, kterou má uživatel plně pod kontrolou, může z ní akce vymazat nebo editovat. Po kliknutí na tlačítko se volitelně může přidat ověření cílové obrazovky tlačítka. Volitelně proto, že ne vždy tlačítko změní obrazovku a není potřeba ji ověřovat. Na druhou stranu může existovat potřeba ověřit, že se obrazovka skutečně nezměnila, když se změnit neměla. Další akce lze vybrat z menu a přidat do fronty, sem patří zejména swipe, ověření dalších prvků na obrazovce pomocí image processingu. K dispozici jsou vybrané akce, které dávají v tomto použití smysl, ne úplně všechny akce které Robot control umí obsluhovat.

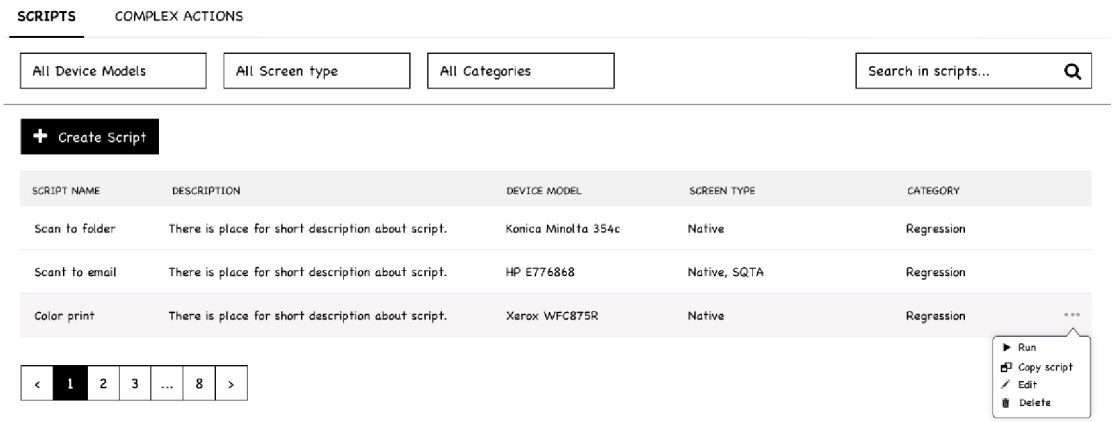
Vedle online režimu je k dispozici také režim offline. Hlavní rozdíl je v absenci živého streamu z kamery a přímého ovládání robotického manipulátoru. Vyžaduje tedy kompletní data o testovaném zařízení zadaná v databázi RQA. Referenční snímky obrazovek, pozice, velikost tlačítek i zdrojové a cílové obrazovky tlačítek (na které obrazovce se tlačítko nachází a na kterou odkazuje). Protože se nezískává obraz z kamery, musí uživatel manuálně vybrat snímek obrazovky, na které chce začít testovací scénář, a při výběru tlačítka potvrdit změnu obrazovky na jinou dle své volby z možných cílových obrazovek tlačítka. Dostupnost akcí v testu je ekvivalentní k online režimu, neboť je samozřejmě možné později otevřít editaci testu v jakémkoliv režimu. Samozřejmostí jsou akce jako poklepání na tlačítko, ověření obrazovky, navigace na obrazovku, ověření detekovaných papírů. Režim offline je užitečný hlavně pro rychlý přehled testovacího scénáře bez nutnosti přímo převzít kontrolu nad robotickým manipulátorem.



Obr. 5-6: Uživatelské rozhraní pro práci s komplexními akcemi

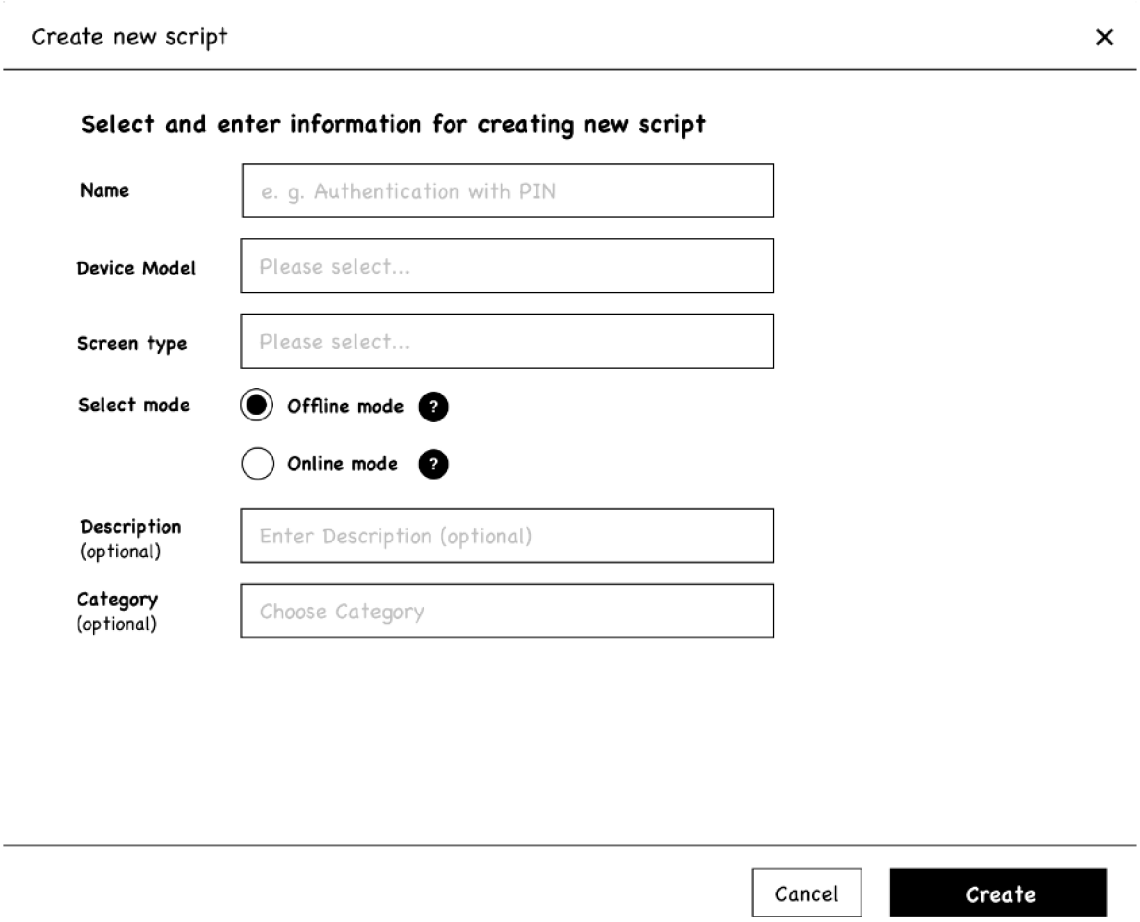
5.5.1 Správa testovacích scénářů

Na hlavní stránce je vidět seznam všech aktuálně definovaných testů. Je zde možnost filtrovat na základě modelu zařízení nebo typu terminálu, také podle jména kategorie nebo samotného testu. Kliknutím na tlačítko možností se zobrazí kontextové menu s akcemi Run, Copy script, Edit, Delete. Tyto akce odpovídají možnostem práce s testovacími scénáři.



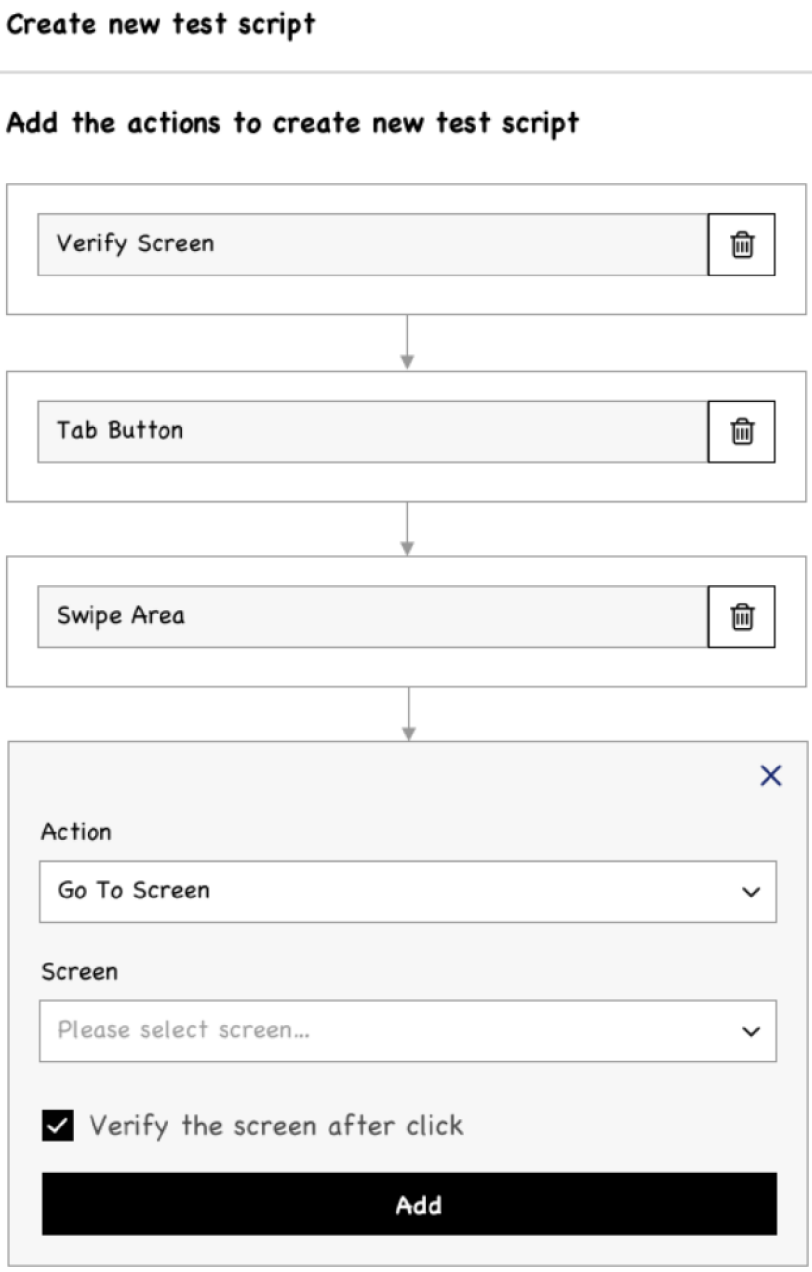
Obr. 5-7: Seznam dostupných testů a možnosti

Vytvoření nového scénáře se zahájí kliknutím na tlačítko Create Script. Uživatel tedy vloží základní informace a vybere si režim, ve kterém si přeje vytvořit testovací scénář. Jméno, kategorie a poznámka jsou textové vstupy, kam může zadat libovolný řetězec, zatímco při zadávání modelu zařízení a typu terminálu (ScreenType) musí vybrat z tabulky existujících zařízení, aby se zaručila návaznost dat (viz Obr. 5-8).



Obr. 5-8: Zadání informací pro nový test

Po vytvoření scénáře a uložení do databáze není konec možností, automatické testy se nevyhnutelně musí aktualizovat, aby reagovaly na změny v SUT. To je samozřejmě možné, všechny definované testovací scénáře jsou dostupné v seznamu, kde je možné filtrovat na základě jména, zařízení nebo měřených vlastností. U zvoleného testovacího scénáře může uživatel vyvolat jeho detail, kdy se otevře povědomé rozhraní, nejdříve samozřejmě bez živého streamu z kamery a ovládání robotického manipulátoru, neboť ty nemusí být v daný okamžik dostupné. Na detailu testu jsou k dispozici kroky testovacího scénáře, jak byly definovány, jednotlivé kroky je možné smazat, upravit nebo přidat do seznamu na libovolné místo. Probíhá zde kontrola návaznosti kroků a konzistence, např. pokud první krok skončil na obrazovce Main Menu, měl by další krok navázat z této obrazovky. Ovšem tato kontrola je pouze informativní, zodpovědnost za konzistenci testu má nakonec uživatel, může požadovat libovolný průběh testu a test designer mu neodporuje. Uživateli se zobrazí upozornění, že test nemusí být realizovatelný a může obsahovat chybu. Protože během úpravy testu se pracuje v offline režimu, nemusí být úplně zřejmé, jestli test může takto vypadat, je třeba klást zvýšenou pozornost. Do budoucna by bylo možné kontrolu konzistence testu zlepšit, aby více uživateli pomáhala a případně zabránila potvrzení změn, pokud by byly chybné.



Obr. 5-9: Posloupnost akcí testovacího scénáře

5.6 Spouštění testů

Každý testovací scénář je pravidelně spouštěn, ideálně každý den ve stejnou dobu nebo kdykoliv podle potřeby na žádost uživatele. Robot control poskytuje API pro spuštění vybraného testy na konkrétním robotovi, který je s aplikací Robot control aktuálně spárován.

Aktuálně jsou k dispozici 3 způsoby, jak vyvolat spuštění testovacího scénáře. První je za použití Robot framerku a testů uložených v něm. Při navigaci do složky

s testy a spuštění příkazem „python go.py test -i robot“ provedou všechny testy s tagem „robot“[10]. Tohoto lze využít pro spuštění všech testu ze zvoleného test suite. Alternativně lze specifikovat pouze testy s určitým jménem pomocí přepínače -t, např. robot go.py test -t Experiment -t Experiment2. Takto lze za sebe skládat testy s určitými jmény. K tomuto slouží plánování pomocí nástroje Atlassian Bamboo, který v požadovaný čas zajistí automaticky nachystání testovacího prostředí a spuštění skriptu který zahájí testy. Dále je vše v režii testovacího frameworku.

Druhá varianta počítá s manuálním návrhem testu a uložení do správného formátu v jsonu[11] definovaném v kapitole 4. Tato možnost je pouze teoretická a nepředpokládá se její využití kvůli složitosti a náročnosti na vytvoření.

Třetí způsob je umožněn díky formátu testovacího scénáře a uložení v databázi, je možné snadno vyvolat spuštění vybraného testu ze seznamu na zvoleném robotovi, pokud je ovšem připraven. Využije se REST rozhraní konkrétní aplikace Robot control spárované se správným robotem a zařízením, které si přejeme testovat. Do API jsem vytvořil controller jménem TestScenario s metodami RunTestFromJson (druhý způsob viz výše) a RunTestFromDb s parametry TestName, DeviceId, Terminal a serverIP. První tři argumenty slouží k unikátní identifikaci konkrétního testovacího scénáře, pokud by existovalo více testů se stejným jménem, zařízením a typem terminálu, jednalo by se o chybu při přidávání scénáře do databáze a test by nemohl být spuštěn. Získaná data o vzhledu testovacího scénáře se seřadí vzestupně podle hodnoty Step a ověří se, že jsou obsaženy navazující hodnoty kroků a žádný není obsažen vícekrát (to by znamenalo poškozená data a neplatný test).

Následně se mírně liší spuštění komplexní a základní akce, při základní se pomocí reflexe [13] nalezne odpovídající metoda z třídy BasicActionExecution na základě hodnoty v ActionName. Opět se kontroluje, že taková akce existuje a je implementována, což by mělo být zajištěno ze zajištění integrity při vytváření akce, ale další ověření je hlavně pro zabránění neoprávněnému zásahu dat přímo v databázi. Třída BasicActionExecution implementuje rozhraní IActionList obsahující všechny základní akce, které jsou určeny pro použití v testech. Je snadné přidat podporu pro novou základní akci robotického manipulátoru, jestliže jsou rozšířeny jeho schopnosti. Stačí přidat novou metodu do rozhraní IActionList a její implementaci do BasicActionExecution. Spuštění komplexní akce spočívá v získání aktuální implementace z databáze ná základě jejího ID a spuštění všech základních akcí definovaných v seznamu BasicActions.

5.7 Robustnost testovacího systému

Během tvorby základních akcí, které umí RQA vykonávat, jsem čelil řadě výzev, jak zvolit robustnost testovacích akcí, aby zároveň nebyly příliš přísné ani příliš benevolentní. Při ovládání multifunkčního zařízení pomocí robotického manipulátoru se objevuje mnoho proměnných vedlejších jevů, jako například nespolehlivost samotného kapacitního displeje, nepřesnost servomotorů atd. O rozpoznání displeje obrazovky a její kategorizaci za různých světelných podmínek, zaprášení a dalších jevů ani nebudu pojednávat, neboť to není předmětem této

práce. Já se zabývám tím, jak potlačit nepříznivé jevy a hlídat kvalitu testovaného systému, v tomto případě software nainstalovaného na multifunkčním zařízení.

5.7.1 Falešné nálezy chyb

Při pravidelném spouštění testovacích scénářů za období ladění testů se ukázalo, že testovací systém vykazoval velké množství falešných chyb. Testoval jsem na známém funkčním prostředí s YSoft SafeQ a na manuálně otestovaném zařízení tu stejnou sadu test, takže všechny nalezené chyby byly falešné. Příčiny byly také různé, zejména nereagování zařízení na poklepání robota na terminál nebo příliš dlouhé načítání na zařízení nebo chybné rozpoznání obrazovky, a tedy chybná reakce robota na tuto situaci. Z vybrané skupiny 20 testů byly opakovaně alespoň 1 až 3 chyby způsobené špatným chováním robotického testera. To jsem vyvodil z videa nahraného během testu a také logů z řídicí aplikace robota. Po rychlé analýze bylo evidentní, že pokud by daný test vykonával člověk, dokázal by situaci správně vyhodnotit a výsledek by byl pozitivní.

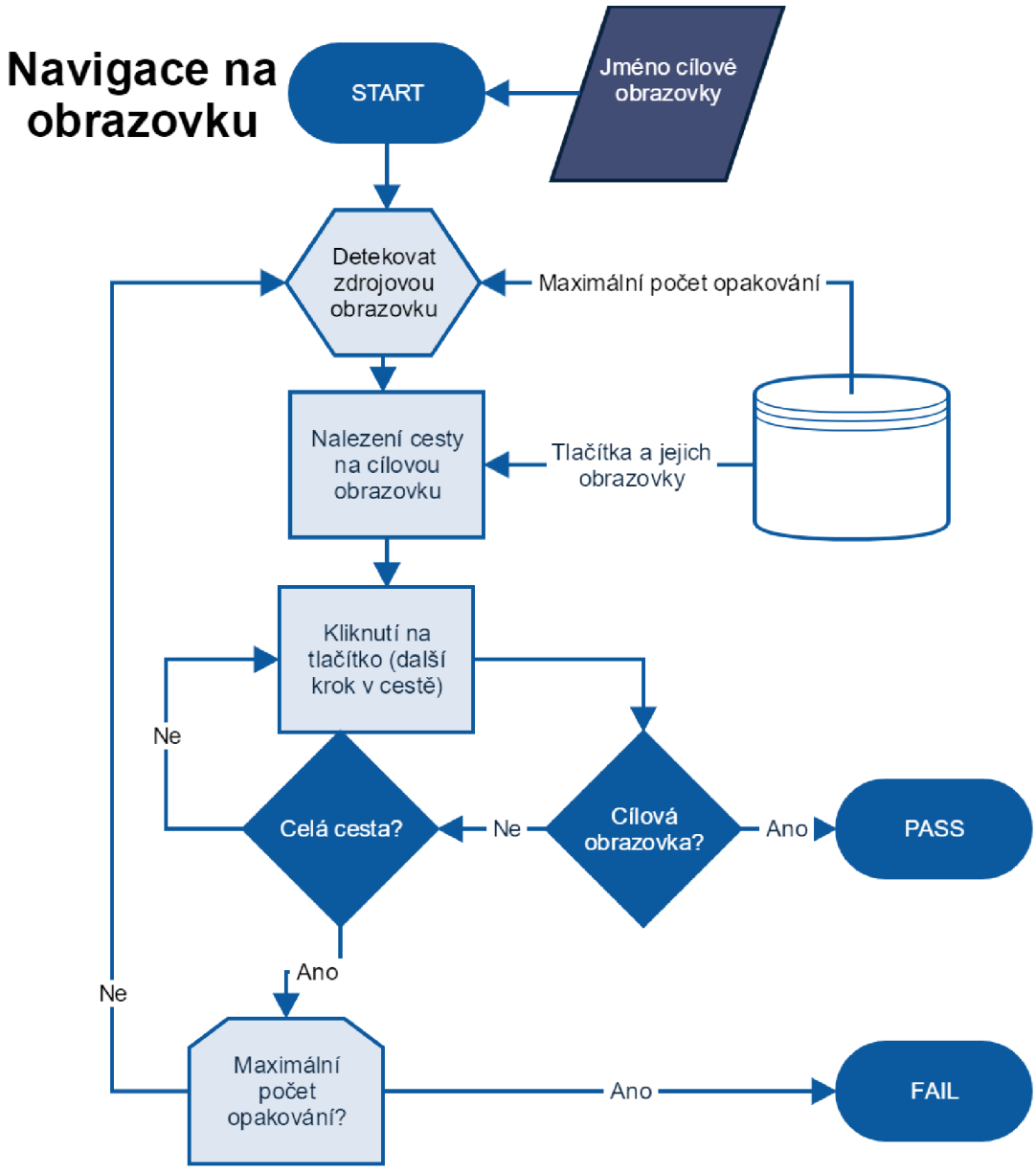
Měl jsem tedy za to, že je třeba vylepšit rozhodovací algoritmy robota a přidat na robustnosti, a zároveň neobětovat spolehlivost. Při spoléhání na jakýkoliv automatický systém, že správně otestuje novou verzi produktu, se na něj přenáší značná odpovědnost.

Do implementace základních akcí jsem tedy dodal opakovací procedury, díky kterým se při neúspěchu akce opakuje po maximální počet pokusů. Tento počet je konfigurovatelný pro určitý model zařízení a také pro určitou akci. Algoritmus pro navigaci na žádanou obrazovku GoToScreen může mít jinou přísnost než obyčejné rozpoznání obrazovky VerifyScreen. Konfigurovatelnost na základě modelu zařízení umožňuje stanovit, na kterém zařízení mají testy fungovat s vysokou přísností a na kterých zařízeních je umožněna větší benevolentnost. Je tedy na zodpovědnosti operátora RQA jak tyto hodnoty stanoví.

Příkladem může být diagram na Obr. 5-10, na kterém je znázorněn průběh algoritmu pro hledání cesty, tzv. Go To Screen. Vstupem do tohoto algoritmu je jméno obrazovky, na kterou se má robot při navigaci dostat. Řídicí program tedy požádá systém zpracování obrazu o informaci ohledně aktuální obrazovky na zařízení. Po přijetí odezvy si získá z databáze data o všech tlačítkách, která by mohla sloužit k navigaci od zdrojové na cílovou obrazovku. Následně se pomocí algoritmu prohledávání stromu do hloubky (depth first search) vyhledá požadovaná cesta a odpovídající tlačítka se seřadí do správného pořadí. Následně se robotovi pošlou souřadnice prvního tlačítka v nalezené cestě a po vykonání kliku se zkontroluje, zda je aktuální obrazovka změněná na očekávanou. Pokud by se aktuální obrazovka shodovala s tou cílovou, je celkový cil splněn a algoritmus končí. Jestli zbývají tlačítka v cestě, pokračuje se kliknutím na další tlačítko.

Při proklikání celé nalezené cesty by měla být přítomná obrazovka shodná s cílovou a pokud je to pravda, akce je úspěšně dokončena. V případě, že nebylo dosaženo cílové obrazovky, lze předpokládat, že na testovaném zařízení nastala chyba znemožňující navigaci na obrazovku, ovšem také mohl udělat chybu robot. V rámci zvýšení robustnosti se v tomto místě program rozhodne, zda bude opakovat celý algoritmus hledání cesty nebo vyvolá chybový výsledek této akce a tím i celého testu. Vzhledem k vysokému počtu false-positive výsledků, které testovací systém produkoval, je vhodnější při selhání vyvinout další snahu o

splnění cíle algoritmu a zkusit celou operaci znova. Algoritmus tedy znovu rozpozná aktuální obrazovku a prohledáním cesty nalezne spojení na cílovou obrazovku a opět postupně vyklikává tlačítka na zařízení. Jestliže se nepodaří dosáhnout cílové obrazovky ani po maximálním počtu opakování, nezbývá než aktuální test ukončit s chybovou hláškou, že se nepodařilo navigovat na cílovou obrazovku.



Obr. 5-10: Diagram implementace navigace na obrazovku

Do dalších základních akcí jsem také dodal obdobnou funkci opakovacích procedur, zejména při ověřování požadované obrazovky, výběru položek v seznamu nebo provádění posloupností tlačítek (flow). V praxi fungují identicky, ale lze je konfigurovat pomocí jiných proměnných, takže je větší možnost nastavit přísnost podle požadavků QA.

6. METRIKY PRO KONTROLU KVALITY

Pro zajištění kvalitního systému nestačí pouze spuštění definovaných testovacích scénářů a jejich úspěšný průběh, ale také určitá měření určující, jak kvalitně se SUT během testování chová. Jedním z hlavních cílů celého testovacího systému RQA je sbírání co nejvíce dat o testovaném systému a interpretace těchto dat k vytvoření závěrů o jeho kvalitě. Dalším krokem je předání těchto interpretovaných dat (informací) zodpovědným osobám (stakeholderům), aby učinily kvalifikované rozhodnutí. Součástí rozhodnutí může být prosté „STOP!“, produkt není dostatečně kvalitní, není možné jej vydat zákazníkům, nejdříve je třeba provést takové kroky ke zkvalitnění. Nebo závěrem bude „V pořádku“, známe rizika, ale produkt je kvalitní dle vývojových standardů. V této kapitole tedy definuji některé metriky a následně implementuji měření těchto metrik do RQA a demonstruji fungování.

6.1 Metriky testovacích scénářů

Zásadní otázka tedy zní, která data sbírat, aby se z nich dala vyvodit obecnější představa o fungování a kvalitě SUT. Budu se držet spíše produktu YSoft SafeQ, ale řada poznatků lze aplikovat na mnoho jiných software.

6.1.1 Doba trvání

Systém RQA je schopen v principu měřit dobu trvání jakéhokoliv přechodu obrazovky od okamžiku stisknutí tlačítka do okamžiku zobrazení požadované obrazovky. Při standartním nastavení během této procedury získává image processing z kamery 30 snímků za sekundu a kterýkoliv snímek může být vyhodnocen jako finální, takže teoretické rozlišení doby je 1/30 s. Pro danou aplikaci je to dostatečné, také s ohledem na to, že více než absolutní hodnota doby trvání je cennější znalost změny této doby. V kontinuálním testování jde o to, aby se vlastnosti SUT výrazně nezhoršovaly.

Měření doby trvání se zdá očividné, ale z mé zkušenosti se na cenu času zákazníka nebere patřičný ohled. Opět uvažujme nejtypičtější scénář při používání YSoft SafeQ. Zákazník (uživatel) přijde k multifunkční tiskárně s požadavkem vytisknout své tiskové úlohy a neočekává žádné zdržení, protože je to pouze dílčí krok k jeho skutečnému cíli. Ve špatném scénáři bude zařízení v úsporném režimu a obrazovka černá, musí tedy stisknout tlačítko probuzení (nebo kterékoliv jiné tlačítko, záleží na zařízení) a počkat na „probuzení“. Z této situace plyne první možná měřený čas – **doba trvání uvedení zařízení do počátečního pracovního stavu**. V lepší situaci bude tento čas nulový, neboť zařízení je již připraveno.

Druhou měřenou dobou je **doba přihlášení**. Uživatel se autentizuje tiskárně zvolenou metodou, která nemusí být pouze jediná. Systém YSoft SafeQ podporuje celou řadu autentizačních metod - PIN, karta, jméno a heslo, a dále kombinaci těchto metod pro vícefaktorovou autentizaci.

Konečným okamžikem měření doby přihlášení je zobrazení přihlášené obrazovky, obvykle hlavního menu nebo přímo aplikace YSoft SafeQ.

Jakmile je uživatel přihlášen, je relevantní, za jak dlouho se dostane na obrazovku s tiskovými úlohami. Při určité konfiguraci to může být první, co po přihlášení uvidí, obecně ale definuji **dobu po zobrazení tiskových úloh**. Nyní vybere svou tiskovou úlohu a potvrdí tisk.

Název měření	Počáteční obrazovka	Cílové obrazovka
Doba přihlášení	Login Screen	Main Menu
Doba po zobrazení tiskových úloh	Main Menu	SQ Print
Doba do počátečního stavu	Main Menu	Login Screen

Tabulka 6-1: Vlastnosti měření času změny obrazovek

Je na místě připomenou, že se nezabývám vnitřním fungováním systému YSoft SafeQ ani podílu multifunkčního zařízení na všech těchto měřených časech, který je nepochybně značný. Dívám se na celý proces očima koncového uživatele, kterého zajímá jeho „zážitek“ z užívání a snažím se dojít k automatizaci měření mimo jiné i user experience, která je velkým podílem celkové kvality software.

6.2 Definice testů

Za účelem měření daných údajů jsem vytvořil testy v Robot Frameworku a využil jsem jeho velké výhody pro obecný předpis testu. Níže je zápis keywordu pro všechny varianty měření času. Které měření se má provádět je dáno argumenty binární hodnota, jimiž je podmíněno spuštění konkrétním měření času. Kupříkladu když je hodnota argumentu *${auth}* nastavena na True, má se provádět část měření času do přihlášení, tedy *Measure Time to authenticate*. Další argumenty podmiňují ostatní části měření, dobu do načtení tiskových úloh, dobu do navigace na obrazovku kopírování i obrazovku na skenování. Posledním měření je doba do uvedení původního stavu, což jsem pojmenoval *Time to logout*.

```
Run selected measurements
    [Arguments]  ${auth}  ${auth_card}  ${auth_screen}  ${menu2print}  ${menu2scan}  ${menu2copy}  ${logout}
    Run keyword if  ${auth}  Measure Time to authenticate  ${auth_screen}
    Run keyword if  ${auth_card}  Measure Time to authenticate by card  ${auth_screen}
    Run keyword if  ${menu2print}  Measure Menu to desired screen  SQ Print  TimeMenu2Print  ${auth_card}
    Run keyword if  ${menu2copy}  Measure Menu to desired screen  Copy  TimeMenu2Copy  ${auth_card}
    Run keyword if  ${menu2scan}  Measure Menu to desired screen  SQ Scan  TimeMenu2Scan  ${auth_card}
    Run keyword if  ${logout}  Measure Time to logout  ${auth_card}
```

Hodnoty jsem sbíral na různých verzích YSoft SafeQ na několika zařízeních, přitom každá konbinace obsahovala několik stovek až tisíc měření, přesné číslo je

také uvedeno. Tabulka 6-2 zobrazuje naměřené hodnoty doby přihlášení pomocí PINu na čtyřech zařízením a pěti různých po sobě jdoucích verzích software YSoft SafeQ. V prvním řádku každého zařízení je uvedena střední hodnota doby přihlášení, ve druhém počet opakování daného měření.

Zařízení	Verze 1	Verze 2	Verze 3	Verze 4	Verze 5
Zařízení A	0.51 s	0.57 s	0.55 s	0.51 s	0.56 s
	3841x	1557x	694x	1367x	1635x
Zařízení B	2.39 s	2.98 s	3.48 s	3.54 s	3.52 s
	1391x	2964x	1072x	1496x	1286x
Zařízení C	3.39 s	3.74 s	4.52 s	5.00 s	5.26 s
	105x	105x	112x	251x	408x
Zařízení D	1.00 s	1.07 s	1.12 s	1.12 s	1.11 s
	10828x	11958x	3323x	1292x	4367x

Tabulka 6-2: Tabulka naměřených času doby přihlášení PINem

Naměřená data sloučí jako demonstrace, že systém RQA je schopen měření definovaných metrik a naměřená data zpřístupňovat stakeholderům pro rozhodnutí o kvalitě testovaného systému.

6.2.1 Opakování akcí

Co dalšího ovlivňuje dojem zákazníka a je důležitým měřítkem celkové kvality, je kolik akcí je třeba pro dosažení požadovaného cíle. Rozhraní pro koncového uživatele by mělo být co nejjednodušší na ovládání, takže pro navigaci v menu, vybráním a potvrzením tiskové úlohy nebo skenovacího nastavení by mělo být vyžadováno co nejméně poklepání na ovládací terminál. Např. vyskakovací okna mohou být pro uživatele obzvlášť nepříjemná, jejich zavření je další (potenciálně zbytečná) akce.

Díky automatizaci testování pomocí RQA je možné měřit celkový počet akcí vyžadovaných během testovacího, a tedy i uživatelského, scénáře. Kdykoliv řídicí program dává pokyn robotickému manipulátoru na poklepání na obrazovku, zaznamená tuto skutečnost zvýšením údaje o celkovém množství poklepání v příslušné tabulce v databázi. Zaznamenání celkového počtu akcí se provádí ve všech testovacích scénářích, protože nevytvářejí zbytečnou zátěž na dobu provádění testu ani nárok na prostor v databázi, na rozdíl od např. měření doby trvání přechodu mezi obrazovkami. Jedna z metrik má tedy název **počet elementárních akcí během testovacího scénáře.**

Dalším prvkem, který lze zkoumat během testování, je opakování požadovaných akcí, tedy jestli testované zařízení správně nedetekuje poklepání na terminál a je nutné jej opakovat, nebo opakování celé části testovacího scénáře. Terminály multifunkčních zařízení jsou založeny na technologii kapacitního nebo rezistivního displeje. Když člověk testuje tyto terminály a pracuje s nimi, občasné nezareagování displeje ignoruje a opakuje akci. Z časových důvodů nemůže zaznamenávat všechny tyto situace, ovšem během automatického testování tohle žádný čas navíc nestojí. RQA samozřejmě dokáže rozpoznat neúspěšné akce, kdy se robotickému manipulátoru zadá akce na poklepání na obrazovku (např. otevřít klávesnici, vedoucí z obrazovky Login Screen -> PIN), následně se očekává

obrazovka s klávesnicí, ale image processing ji opakovaně nerozpozná. V takové situaci se usoudí, že byla akce neúspěšná a pokud je na terminálu zobrazena původní obrazovka (Login Screen), je pokyn manipulátoru na stisk tlačítka otevřít klávesnici opakován. Metrika, která zobrazuje počet neúspěšných pokusů se nazývá **počet neúspěšných akcí na terminálu**.

Testovací scénář	Zařízení	Počet kliknutí na terminál	Celkový počet obrazovek	Neočekávané obrazovky
Měření času přihlášení kartou	Sharp	10	24	0
Měření času přihlášení PINem	Sharp	19	31	0
Měření času přihlášení kartou	KM	11	23	0
Měření času přihlášení PINem	KM	21	28	0

Tabulka 6-3: Naměřené údaje o kvalitě testování

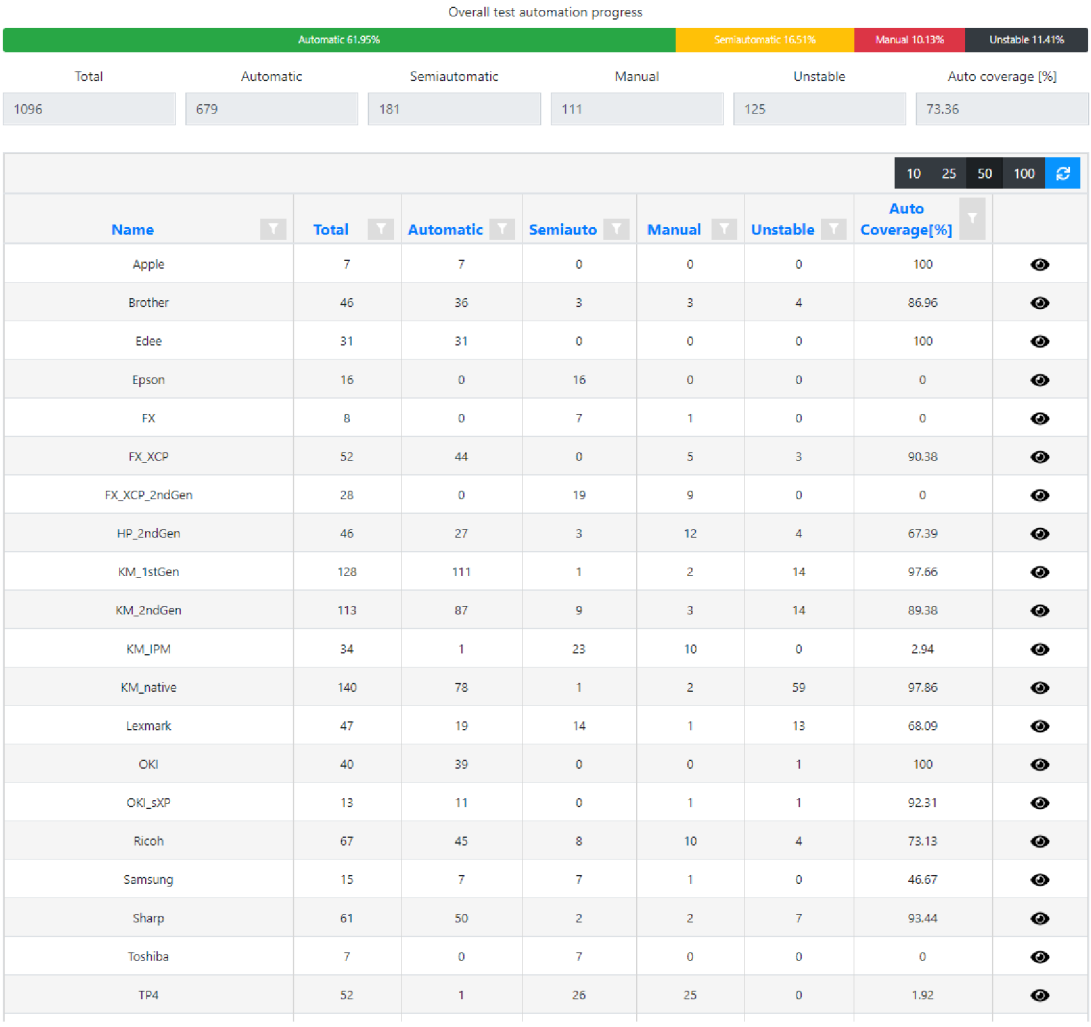
6.3 Metriky testovacích sad

Mezi měřené a sledované údaje patří nejenom data získaná během jednotlivých spouštění testů, ale také nadřazené údaje a trendy úspěšnosti celých testovacích sad. Mezi příklady takových metrik může patřit odpověď na otázky: „Kolik chyb bylo objeveno za dnešní den testování?“, „Kolik automatických testů lze spustit za jeden den/ za jednu iteraci vývoje?“, „Jaké je pokrytí testování automatickými testy? Jak se toto pokrytí zvyšuje?“ a další. Součástí této práce je také přinést možnost získávání odpovědí na tyto otázky. Zájem na nich mají hlavně inženýři kontroly kvality, manažeři softwarového vývoje a testování. Umožnují jim plánování času potřebného na testování v každé iteraci, plánování nákladů na aktivity spojené s kontrolou kvality. Výsledky naměřených údajů jsou přehledně zobrazeny v reportu testu, který má každý k dispozici, např. ve formě Dashboard.

6.3.1 Pokrytí testů

Již jsem naznačil, že klíčovým faktorem při softwarovém vývoji je znalost rizika vyskytnutí chyby. Tuto znalost můžeme získat nebo se jí přiblížit díky testování velkým množstvím různých uživatelských scénářů. S tím souvisí sledování množství testovacích scénářů, tzv **test coverage**. V souvislosti s automatizací testování ještě musíme znát míru automatizace těchto testů, tedy **automation test coverage**[8]. Ta určuje, jaké procento testovacích scénářů je automatizováno, ovšem už nemluví o skutečném spouštění testů, jinými slovy, tolik testů automatických máme, ale jestli je spouštíme nebo ne už nám test coverage neřekne. Důvodů pro nespuštění testů může být několik, např. se stal nefunkční

kvůli změně v software, se kterou je test nekompatibilní a je třeba jej upravit, mezitím se vyřadil z testovací sady. Testy, které se neprovádějí je třeba co nejrychleji zprovoznit, musí se tedy sledovat skutečné výsledky, tedy **executed tests**, případně **executed automated tests**. Když je jasno v tom, kolik testů se skutečně provedlo, lze se zabývat jejich výsledkem. Výsledek pokrytí testů na testovací sadě pro YSoft SafeQ je na Obr. 6-1. Kategorie automatic, semiautomatic, manual a unstable označují různé fáze implementace testu. Je tedy na první pohled patrné, kolik testů je plně automatizovaných, kolik vyžaduje zásah člověka, ovšem některé akce lze vykonávat automaticky a které testy jsou zatím plně manuální a je požadavek je automatizovat.



Name	Total	Automatic	Semiauto	Manual	Unstable	Auto Coverage[%]
Apple	7	7	0	0	0	100
Brother	46	36	3	3	4	86.96
Edee	31	31	0	0	0	100
Epson	16	0	16	0	0	0
FX	8	0	7	1	0	0
FX_XCP	52	44	0	5	3	90.38
FX_XCP_2ndGen	28	0	19	9	0	0
HP_2ndGen	46	27	3	12	4	67.39
KM_1stGen	128	111	1	2	14	97.66
KM_2ndGen	113	87	9	3	14	89.38
KM_IPM	34	1	23	10	0	2.94
KM_native	140	78	1	2	59	97.86
Lexmark	47	19	14	1	13	68.09
OKI	40	39	0	0	1	100
OKI_sXP	13	11	0	1	1	92.31
Ricoh	67	45	8	10	4	73.13
Samsung	15	7	7	1	0	46.67
Sharp	61	50	2	2	7	93.44
Toshiba	7	0	7	0	0	0
TP4	52	1	26	25	0	1.92

Obr. 6-1: Vizualizace pokrytí testů

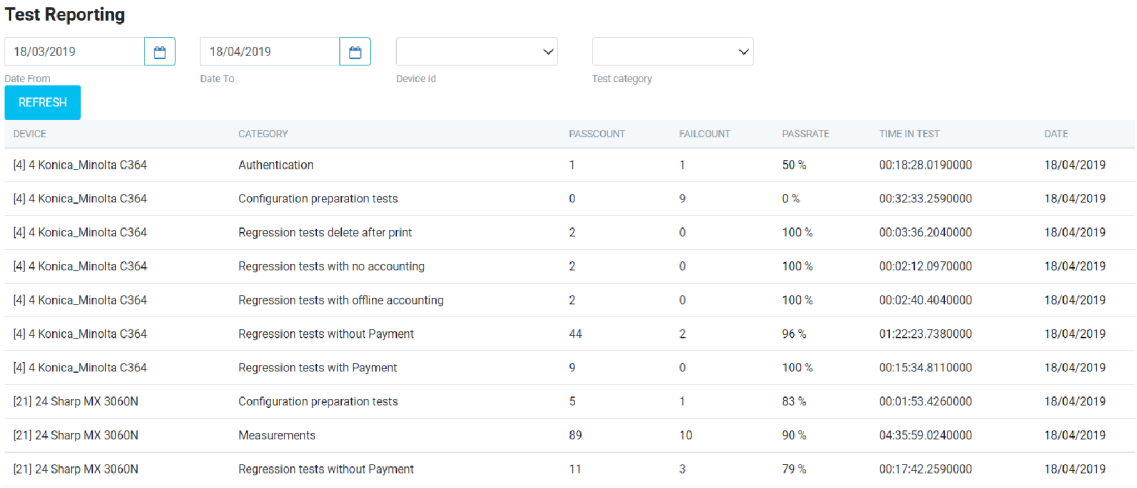
6.3.2 Výsledky testů

Dalšími metrikami jsou **procento úspěšných testů** a **procento neúspěšných testů**. Jak jsem zmiňoval, každý test má svou prioritu vyjadřující míru nutnosti úspěšného provedení testu. Výsledky je tedy vhodné spojit s testy se stejnou prioritou. Provedení testu nemusí mít za výsledek pouze PASS/FAIL hodnotu, ale u neúspěšných testů je velmi důležitá příčina chyby. Jestliže test neprošel kvůli jiné příčině než chybě v testovaném systému, je označen na false positive a neměl by

být počítán do metriky neúspěšných testů, alespoň do té doby, než je případně opraven, znovu spuštěn a vyprodukuje relevantní výsledek.

6.4 Zobrazení výsledků testování

Samotné testování nemá žádnou hodnotu, pokud se z jeho výsledků nevyvodí patřičné závěry. Nyní popíši, jak se tyto výsledky zobrazí QA inženýrovi, který z nich získá klíčové informace. Celá stránka s výsledky, tzv. dashboard, je navrhnuta na stejném webovém portálu jako rozhraní pro tvorbu testů. Navrhnul jsem několik způsobů zobrazení. Uživatel si může vybrat, jestli si zobrazí historii běhů jednotlivých testů, ve které může filtrovat dle požadovaných parametrů, např. jméno testu, jméno zařízení, čas spuštění nebo identifikátor robotického manipulátoru. Kliknutím na položku v seznamu se zobrazí detail testu obsahující další informace hlavně o výsledku testu, případné chybové zprávě a samozřejmě naměřené hodnoty zkoumaných metrik. Vše je v přehledné tabulce, takže uživatel vidí, jak dlouho trvala každá akce, kolikrát se musela opakovat nebo kolik celkem času se strávilo testováním. Navíc zde může QA inženýr může označit neúspěšný test jako false-positive a přidat skutečný důvod skončení testu. Tento důvod skončení se samozřejmě může lišit od chybové zprávy, kterou vyprodukoval testovací systém RQA, pokud např. nastala chyba v samotném testovacím systému. QA inženýr má k dispozici videozáznam o celém průběhu testu, pořízený kamerou umístěnou nad zařízením, logy z testovacího systému a může si dohledat logy z testovaného systému. Z těchto dat by mělo být snadné určit skutečnou příčinu chyby, ačkoliv to může být časově náročné, zejména při větším počtu neúspěšných testů.



DEVICE	CATEGORY	PASSCOUNT	FAILCOUNT	PASSRATE	TIME IN TEST	DATE
[4] 4 Konica_Minolta C364	Authentication	1	1	50 %	00:18:28.0190000	18/04/2019
[4] 4 Konica_Minolta C364	Configuration preparation tests	0	9	0 %	00:32:33.2590000	18/04/2019
[4] 4 Konica_Minolta C364	Regression tests delete after print	2	0	100 %	00:03:36.2040000	18/04/2019
[4] 4 Konica_Minolta C364	Regression tests with no accounting	2	0	100 %	00:02:12.0970000	18/04/2019
[4] 4 Konica_Minolta C364	Regression tests with offline accounting	2	0	100 %	00:02:40.4040000	18/04/2019
[4] 4 Konica_Minolta C364	Regression tests without Payment	44	2	96 %	01:22:23.7380000	18/04/2019
[4] 4 Konica_Minolta C364	Regression tests with Payment	9	0	100 %	00:15:34.8110000	18/04/2019
[21] 24 Sharp MX 3060N	Configuration preparation tests	5	1	83 %	00:01:53.4260000	18/04/2019
[21] 24 Sharp MX 3060N	Measurements	89	10	90 %	04:35:59.0240000	18/04/2019
[21] 24 Sharp MX 3060N	Regression tests without Payment	11	3	79 %	00:17:42.2590000	18/04/2019

Obr. 6-2: Tabulka s vypočítanými statistickými hodnotami

Samostatně jsou informace o výsledku jednoho testu málo informativní, takže jsem vytvořil webovou stránku, na které se sdružují data z mnoha testových běhů do přehledných grafů. Takové výsledky jsou dále sdruženy podle multifunkčního zařízení, na kterém testy byly provedeny. Jestliže uživatele zajímá srovnání všech zařízení za poslední noc mezi sebou, vybere v zadávacím okénku tato zařízení a zobrazí se mu rychlý přehled testů za vybrané časové období (např. minulou noc). Tento rychlý přehled zobrazuje počet provedených testů, jejich míru

úspěšnosti a míru automatizace testů. Detailnější pohled na statistiku výsledků nabízí informace o počtu různých chyb a počtu testů, které skončili neúspěchem kvůli podobné chybě.

DEVICE	CATEGORY	PASSCOUNT	FAILCOUNT	PASSRATE	TIME IN TEST	DATE
[40] 33 OKI_SXP ES 8453	Configuration preparation tests	2	0	100 %	00:01:11.4510000	18/04/2019
[40] 33 OKI_SXP ES 8453	Regression tests with offline accounting	0	1	0 %	00:01:21.7510000	18/04/2019
[40] 33 OKI_SXP ES 8453	Regression tests without Payment	8	6	57 %	01:37:12.9180000	18/04/2019
[40] 33 OKI_SXP ES 8453	Authentication	3	1	75 %	00:07:17.2520000	17/04/2019
[40] 33 OKI_SXP ES 8453	Configuration preparation tests	2	0	100 %	00:01:02.5690000	17/04/2019
[40] 33 OKI_SXP ES 8453	Regression tests with offline accounting	1	3	25 %	00:08:33.7220000	17/04/2019
[40] 33 OKI_SXP ES 8453	Regression tests without Payment	12	7	63 %	00:44:09.5080000	17/04/2019
[40] 33 OKI_SXP ES 8453	Authentication	2	4	33 %	00:15:04.6160000	16/04/2019
[40] 33 OKI_SXP ES 8453	Authentication	1	0	100 %	00:01:15.5540000	16/04/2019
[40] 33 OKI_SXP ES 8453	Configuration preparation tests	3	0	100 %	00:01:10.2690000	16/04/2019

Obr. 6-3: Vybrané výsledky testů na zařízení OKI

6.5 Výpočet statistik výsledků

Pro výpočet výsledků jsem vytvořil službu jménem TestReportingService, která v konfigurovatelných pravidelných intervalech (např. každou hodinu) provede přepočítání daných statistik na základě nejaktuálnějších výsledků testů v databázi. Statistiky se vypočítají za časové období, které jsem ve výchozím nastavení zvolil na posledních 30 dní. Nejdříve se tedy testy rozdělí podle data provedení a zpracovává se každý den zvlášť. Následně se rozdělí podle kategorie testů a podle testovaného zařízení. V každé skupině jsou tedy všechny výsledky testů za právě jeden den, jedno zařízení a jedna kategorie testů. Nad touto skupinou výsledků se aplikují vlastní výpočty. Provede se součet všech provedených testů a součet všech testů s konkrétním výsledkem (PASS nebo FAIL). Z poměru úspěšných testů a celkovým počtem se získá míra úspěšnosti testů. Následně se sečte čas strávený v jednotlivých testech pro získání celkového času stráveného automatickým testováním. Jak je vidět na Obr. 6-2 a Obr. 6-3, čas strávený v testu za jeden den na jednom zařízení byl kolem 2 hodin na zařízení KM a OKI, až 5 hodin na Sharp.

7. ZHODNOCENÍ VÝSLEDKŮ

Automatizace testování s sebou přináší hlavně ušetření času testování včetně posunutí času testování do nočních hodin mimo standardní pracovní dobu člověka. Během sbírání výsledků jsem naplánoval automatické spouštění testů na noční hodiny na pěti zařízeních zároveň a každý z nich testoval kolem 3 hodin čistého času. To znamená 15 hodin čistého času každý den (respektive noc), které by musel provádět člověk nebo by nebyla ověřena kvalita systému v žádané míře.

Způsob vytváření testů záleží na preferenci každého QA inženýra, vytvořil jsem tedy 2 způsoby, které jsou z hlediska funkcionality testů téměř identické, ovšem liší se v rozhraní, přes které jsou tvořeny. První způsob vyžaduje jisté programovací znalosti, ale nakonec nabízí textový předpis čitelný i netechnickému člověku. Druhý, inovativní způsob nabízí přímý náhled nad každou prováděnou akci, jak bude vypadat při samotném vykonávání testu. Při vytváření člověk vizuálně kontroluje, že testovací scénář postupuje požadovanými kroky, ovšem pro člověka neznalého testované problematiky může být takový pohled nevhodný. Oba způsoby návrhu podporují vytváření uživatelských složených akcí, pod kterými se skrývá logika nižší úrovně, ovšem navenek působí jako jedna vysokoúrovňová operace.

Definované metriky pro ověřování kvality představují silný a objektivní nástroj pro posouzení efektivnosti a ergonomičnosti designu software. Díky nim je možné optimalizovat software, jeho GUI a logičnost návaznosti jeho obrazovek. Naměřené výsledky ukazují, že zejména časová odezva testovaných systémů se objektivně liší v různých verzích, což má samozřejmě vliv na kvalitu vnímanou koncovým zákazníkem. Ještě důležitější je sledování vývoje časové náročnosti přechodů mezi danými obrazovkami, což mnou implementované testy umožňují. Vytvořená vizualizace přehledně zobrazuje získané výsledky ze samotných testů, pokrytí testů automatizací vzhledem k požadovanému množství a úspěšnost testů v čase. Uživatel tak může procházet agregovanými výsledky za vybrané období a získat hlubší přehled o testovaných zařízeních rozdělených do kategorií testovacích sad.

Metriky pro určení kvality otestování, tedy zda test probíhal podle žádaného předpisu, sledují konzistenci testování a upozorní na odchylky. Během mého testování se průběhy testů držely v mezích kolem očekávané hodnoty.

Jak se ukázalo ve statistikách vytíženosti testovacího systému, ušetření času bylo 9 hodin za 1 den, přičemž byly otestovány 3 zařízení. Při nasazení RQA na více zařízeních se paralelně testují všechna, takže za 3 hodiny jsou provedeny všechny testy na všech zařízeních, což např. na 6 zařízeních znamená 18 hodin ušetřeného času člověka. V případě, že všechny testy skončí s pozitivním výsledkem, není třeba žádné akce člověka. Pouze při vyskytnutí se chyby musí člověk provést analýzu a vyvodit případné důsledky.

8. MOŽNOSTI ZLEPŠENÍ

Vytvořené řešení má pochopitelně i své nedostatky, na které v blízké budoucnosti budu směřovat svoje úsilí, neboť na tomto projektu budu i nadále pokračovat. Zejména návrh testů potřebuje zvýšit úroveň intuitivnost. V současnosti musí uživatel přenést svoji znalost o podobě požadovaného testovacího scénáře do rozhraní pro tvorbu testů, ovšem pokud nemá detailní přehled o podobě testu, může to být poměrně náročné. Systém pro tvorbu a správu testů by měl obsahovat inteligentní našeptávač, který by nabídl např. nejčastěji používané akce nebo nejčastější navazující akce. Takový našeptávač by analyzoval, které testy již existují na všech jiných zařízeních, případně jiných testech zvoleného zařízení a nabídl uživateli nejpravděpodobnější variant, ovšem pouze jako nápovědu, nikoliv automatické doplňování, zodpovědnost o finální podobě testu musí zůstat na uživateli. Další možnost usnadnění by byla funkce klonování více testů zároveň, zejména při implementování automatizovaných testů na novém zařízení, pokud by jejich průběh byl podobný již existujícím testům. Po výběru více testů a spuštěním funkce klonování by se mohl vybrat model zařízení, na které se překopírují testy a jsou pak k dispozici k editací drobných změn.

Mezi další vylepšení plánuji více vypočítávaných a zobrazených statistik, například pro vývoj pokrytí testů v čase nebo změna úspěšnosti testů v závislosti na vybraných kategoriích testů nebo na různých úrovní verzování (ne pouze na celé verzi, jako nyní). Co se týče samotného měření doby odezvy testovaného zařízení, lze doplnit i další situace, jejichž dobu měřit, teoreticky u každého kroku ve scénáři, který mění obrazovku. S dalším vylepšením souvisí snaha automatizovat všechny testy, které se doposud při vývoji YSoft SafeQ prováděly manuálně. Následně nastane vhodný čas rozšířit pokrytí testů na mnohem více scénářů a situací a s intuitivním rozhraním pro tvorbu testů bude zvýšení pokrytí jednodušší.

9. ZÁVĚR

V této práci jsem rozebral možnosti automatizace testování software YSoft SafeQ na multifunkčních tiskárnách za využití robotického testovacího systému RQA vyvíjeného ve firmě YSoft. V první kapitole jsem probral teorii a techniky testování a jejich třídění podle okamžiku kdy jsou prováděny ve vývojovém cyklu. Dále jsem popsal testovaný software YSoft SafeQ a multifunkční zařízení, na nichž je instalován z uživatelského pohledu, tedy převážně jako black box, neboť toto je nutné pro další testování pomocí RQA. Popsal jsem existující možnosti systému RQA a zabýval jsem se požadavky na tvorbu testovacích scénářů, které budou pomocí něj realizovány.

Následně jsem navrhnul dva způsoby, jakým by se co nejvíce automatizovala tvorba testovacích scénářů, což je jeden z klíčových požadavků. Test by měl být schopen vytvořit každý uživatel se základní znalostí robotického testovacího systému. Pomocí kombinace metod keyword-driven a capture-replay díky přímé kontrole nad robotickým manipulátorem a přímým přenosem z kamery realizuje požadovaný testovací scénář a uloží do databáze. Implementoval jsem a srovnal jsem obě varianty – jak návrh pomocí keywordů a vysokoúrovňový zápis testů v Robot frameworku, tak interaktivní návrh testů na základě skládání základních akcí do složitějších akcí. Součástí je také online režim tvorby testu pomocí přímé kontroly nad testovacím zařízením a získávání zpětné vazby pomocí živého záznamu z kamery. Vytvořil jsem a popsal také rozhraní pro spouštění testovacích scénářů vytvořených v obou variantách. První varianta má výhody ve snadné čitelnosti testů a jejich účelu a univerzální použití stejného testu na různých testovaných zařízeních. Druhá varianta má výhodu v intuitivnosti tvorby testu bez potřeby programování a viditelnost průběhu testu přímo na daném zařízení už během tvorby testu.

Dále jsem se zabýval zvýšením robustnosti testovacího systému a snížením počtu falešných nálezů chyb během testování. Také jsem definoval kritéria pro stanovení kvality otestovaného systému. S tím souvisí definování dvou sad metrik, jedné pro kontrolu spolehlivosti otestování a další pro skutečnou kontrolu kvality testovaného software. Měření časových kritérií probíhá během speciálních testovacích scénářů a výsledky se ukládají do databáze spolu s identifikátorem scénáře, informacemi o testovaném prostředí atd. Ostatní údaje jsem implementoval do samotného provádění základních akcí RQA, takže jsou k dispozici pro všechny testy.

Následně jsou výsledky přehledně vizualizovány na dashboard, kde si může uživatel procházet nejen jednotlivé výsledky testů, naměřených údajů, ale také trendy a vývoj měřené kvality v čase.

Literatura

[1] Aleš Pernikář. *Calibration - How robot learns the environment* [online]. 27.10.2015 [cit. 2018-11-19]. Dostupné z: https://www.ysofters.com/2015/10/27/calibration-how-robot-learns-the-environment/

[2] THUMMALAPENTA, Suresh, Saurabh SINHA, Nimit SINGHANIA a Satish CHANDRA. *Automating Test Automation* [online]. , 11 [cit. 2018-11-19]. Dostupné z: https://www.seas.upenn.edu/~nimits/papers/icse12.pdf

[3] NORVIG, Peter a Stuart RUSSELL. *Artificial Intelligence: A Modern Approach*. 3rd Edition. Pearson, 2016. ISBN 978-1292153964.

[4] LU, Luo. *Software Testing Techniques: Technology Maturation and Research Strategy* [online]. In: . s. 20 [cit. 2018-11-19]. Dostupné z: http://www.cs.cmu.edu/~luluo/Courses/17939Report.pdf

[5] LÖNNBERG, Jan. *Visual testing of software* [online]. Helsinki, 2003 [cit. 2018-11-19]. Dostupné z: http://www.cs.hut.fi/~jlonnber/VisualTesting.pdf. Master thesis. Helsinki University of Technology. Vedoucí práce Lauri Malmi.

[6] DOROTHY GRAHAM ... [ET AL.]. *Foundations of software testing: ISTQB certification*. Australia: Thomson, 2007. ISBN 978-184-4803-552.

[7] YSOFT SAFEQ: AN ENTERPRISE PRINT MANAGEMENT SOLUTION. In: *Https://www.ysoft.com* [online]. 2016 [cit. 2018-12-05]. Dostupné z: https://www.ysoft.com/en/support/downloads/resources?pages=1&type=Whitepapers

[8] GARRETT, Thom. *Implementing Automated Software Testing - Continuously Track Progress and Adjust Accordingly* [online]. 2009, , 16 [cit. 2018-12-05]. Dostupné z: http://www.methodsandtools.com/archive/archive.php?id=94

[9] KYZLINK, Jiří, Václav NOVOTNÝ, Aleš PERNIKÁŘ, Jakub PAVLÁK a Ondřej KRAJÍČEK. *Universal automated testing of embedded systems*. 2017. United States. US 2018 / 0113774 A1. Uděleno 26.4.2018. Zapsáno 19.10.2017. Dostupné také z: https://patents.google.com/patent/US20180113774A1/

[10] Robot Framework User Guide. *RobotFramework.org* [online]. [cit. 2019-05-02]. Dostupné z: http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#test-execution

[11] Json.org [online]. [cit. 2019-05-02]. Dostupné z: http://www.json.org/

[12] Getting Started with EF Core on ASP.NET Core. Microsoft Docs [online]. [cit. 2019-05-02]. Dostupné z: https://docs.microsoft.com/en-us/ef/core/get-started/aspnetcore/existing-db/

[13] Reflection (C#). Microsoft Docs [online]. [cit. 2019-05-02]. Dostupné z: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/concepts/reflection/

[14] Unit test basics. Microsoft Docs [online]. [cit. 2019-05-02]. Dostupné z: https://docs.microsoft.com/en-us/visualstudio/test/unit-test-basics?view=vs-2019

[15] Robot Framework High-level architecture. Robot Framework [online]. [cit. 2019-05-02]. Dostupné z: http://robotframework.org/robotframework/3.0.3/RobotFrameworkUserGuide.html#id413

[16] DotVVM Introduction. DotVVM Documentation [online]. [cit. 2019-05-02]. Dostupné z: https://www.dotvvm.com/docs/tutorials/introduction/2.0

[17] DotVVM ViewModels. DotVVM Documentation [online]. [cit. 2019-05-02]. Dostupné z: https://www.dotvvm.com/docs/tutorials/basics-viewmodels/2.0

[18] Selenium Documentation. Selenium [online]. [cit. 2019-05-02]. Dostupné z: https://www.seleniumhq.org/docs/

Seznam symbolů, veličin a zkratek

RQA - Robotic quality assurance, systém pro kontrolu kvality vyvíjený interně ve firmě YSoft.

SUT - System under test, označení pro kombinaci software a hardware, která podléhá testování.

KM - Konica Minolta, značka multifunkčních zařízení

TP - Terminal Professional, terminál vyvíjený firmou YSoft

Přílohy na přiloženém CD

1. Zdrojové kódy správce testovacích scénářů
2. Vzhled webového rozhraní správce testovacích scénářů
3. Naměřené hodnoty časových odezev
4. Skript pro výpočet pokrytí testů