Automatizované testování výrobních informačních systémů

Jakub Švec

Bakalářská práce
2023

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky
Ústav informatiky a umělé inteligence

Akademický rok: 2022/2023

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jakub Švec**
Osobní číslo:	**A20021**
Studijní program:	**B0613A140020 Softwarové inženýrství**
Forma studia:	**Prezenční**
Téma práce:	**Automatizované testování výrobních informačních systémů**
Téma práce anglicky:	**Automated Testing of Manufacturing Execution Systems**

Zásady pro vypracování

1. Vypracujte literární rešerši na dané téma.
2. Seznamte se s vybraným výrobním informačním systémem ve společnosti Continental Barum s.r.o.
3. Vytvořte sadu automatizovaných testů pro daný informační systém ve vhodném nástroji.
4. Integrujte automatizované testy do standardního procesu vývoje softwaru ve firmě Continental Barum s.r.o.
5. Vyhodnoťte pokrytí aplikace automatizovanými testy.

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. GRAHAM, Dorothy, Rex BLACK a Erik VAN VEENENDAAL. Foundations of software testing: ISTQB certification. Fourth edition. Andover, Hampshire: Cengage, [2020]. ISBN 978-1-4737-6479-8.
2. PAGE, Alan, Ken JOHNSTON a Bj ROLLISON. Jak testuje software Microsoft. Brno: Computer Press, 2009. ISBN 978-80--251-2869-5.
3. BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.
4. RANOREX: STUDIO ADVANCED USERGUIDE. 2022. Austin: Idera, 2022. Dostupné také z: https://www.ranorex.com/rx--media/rx-user-guide/pdf-v10.2/Ranorex-Studio-Advanced.pdf
5. RANOREX: STUDIO EXPERT USERGUIDE. 2022. Austin: Idera, 2022. Dostupné také z: https://www.ranorex.com/rx--media/rx-user-guide/pdf-v10.2/Ranorex-Studio-Expert.pdf

Vedoucí bakalářské práce: **doc. Ing. Jiří Vojtěšek, Ph.D.**
Ústav řízení procesů

Konzultant bakalářské práce: **Ing. Jaromír Šánek**
Ústav informatiky a umělé inteligence

Datum zadání bakalářské práce: **2. prosince 2022**
Termín odevzdání bakalářské práce: **26. května 2023**

L.S.

doc. Ing. Jiří Vojtěšek, Ph.D. v.r.
děkan

prof. Mgr. Roman Jašek, Ph.D., DBA v.r.
ředitel ústavu

Ve Zlíně dne 7. prosince 2022

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne 26.5.2023

Jakub Švec
podpis studenta

ABSTRAKT

Bakalárska práca pripomína a zdôrazňuje potrebu automatizácie testov. Teoretická časť nesie sebou zámer vysvetliť a oboznámiť spôsoby testovania softvéru a ich jednotlivé techniky a úrovne spolu s ukážkou výrobného informačného systému spoločnosti Continental Barum s.r.o. a priblížením vhodného nástroja na automatizáciu testov. Praktická časť sa zameriava na konkrétnu sadu automatizovaných testov spolu s jej integráciou do vývojového procesu a vyhodnotením pokrytia aplikácie danou sadou.

Klúčové slová:

Jenkins, Ranorex, automatizácia testovania, testovací prípad, testovanie.

ABSTRACT

The bachelor thesis recalls and emphasizes the need for test automation. The theoretical part carries with it the intention to explain and familiarize the software testing methods and their different techniques and levels, together with a demonstration of the production information system of Continental Barum s.r.o. and an introduction to a suitable test automation tool. The practical part focuses on a specific automated test suite together with it's integration into the development process and an evaluation of the application coverage of the suite.

Keywords: Jenkins, Ranorex, test automation, test case, testing.

Týmto spôsobom by som chcel poďakovať svojmu konzultantovi bakalárskej práce, Ing. Jaromírovi Šánkovi a svojmu vedúcemu dekanovi, doc. Ing. Jiřímu Vojtěšekovi, Ph.D. za pomoc pri vypracovávaní bakalárskej práce.

OBSAH

ÚVOD

Automatizované testovanie je stále viac a viac potrebné v súčasnom svete, ktorý je závislý na neustálej tvorbe nových a sofistikovanejších aplikácií, ktoré potrebujú kvalitný a dlhodobo spoľahlivý chod. Automatizované testovanie umožňuje zvýšiť efektivitu a presnosť testovania pri minimalizácii chýb. Automatizáciou testov sa zároveň znižujú náklady potrebné na neustále opakované manuálne testovanie softvéru. Avšak, hoci automatizované testovanie prináša mnohé výhody, nie je bez svojich výziev. Vyžaduje účinné nástroje, odborné znalosti a správnu stratégiu testovania, aby bolo možné využiť jeho plného potenciálu.

Cieľom tejto práce je priblížiť podstatu softvérového testovania ako takého. Taktiež je cieľom oboznámenie sa s výrobným informačným systémom v spoločnosti Continental Barum s.r.o. spolu s vytvorením sady automatizovaných testov pre výrobný informačný systém. Predmetom tejto práce bude aj integrácia danej sady testov spolu s vyhodnotením pokrytia aplikácie automatizovanými testami.

I. TEORETICKÁ ČASŤ

1 SOFTVÉROVÉ TESTOVANIE

Softvérové testovanie je metóda, ktorá ma za cieľ určiť, či daný softvérový produkt zodpovedá očakávaným požiadavkám a či sa v ňom nenachádzajú chyby. Testovanie jednotlivých komponentov softvéru prebieha buď manuálne alebo automatizovane za pomoci nástrojov na to určených. Cieľom testovania softvéru je identifikovať chyby a nedostatky v pomere so skutočnými požiadavkami. Softvérové testovanie je možno vykonávať na niekoľkých úrovniach a pomocou rôznych testovacích techník. Každá testovacia úroveň a technika je v niečom špecifická a pre správne otestovanie aplikácie je potrebné ich využívať viacero. Správne testovaná aplikácia zabezpečuje bezpečnosť, výkonnosť a spoľahlivosť pre používateľov.

1.1 Úrovne Softvérového testovania

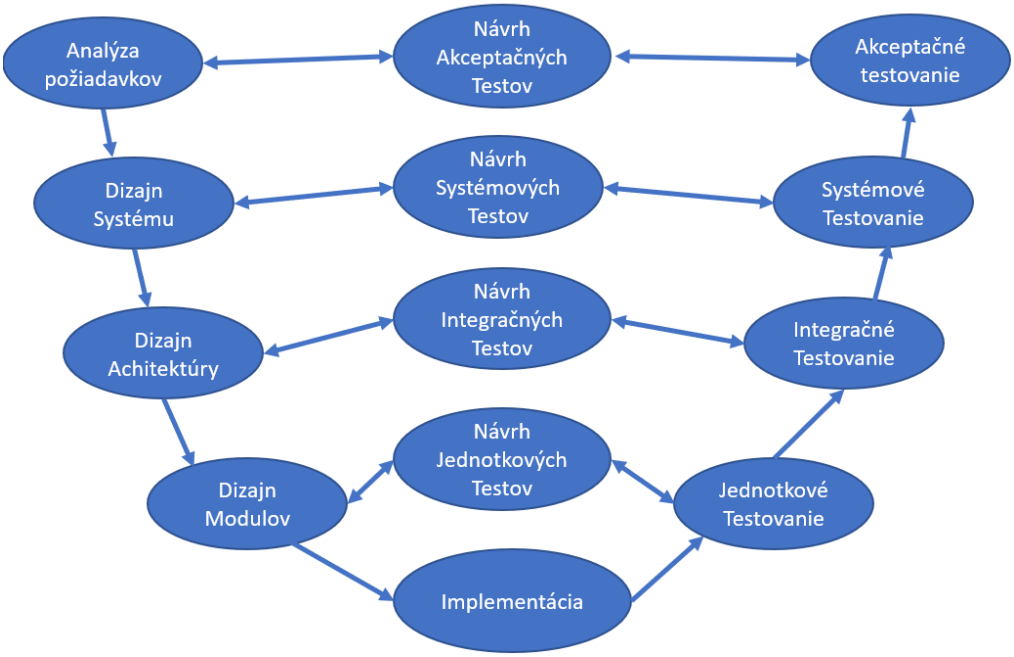
Úroveň softvérového testovania je termín používaný v rámci testovania softvéru na označenie rôznych typov testovania. Každá úroveň testovania sa zameriava na inú časť softvéru a používa iné testovacie techniky a metódy na overenie správneho fungovania softvéru. [1]

Testovanie sa delí do nasledujúcich úrovní:

- Jednotkové testovanie
- Integračné testovanie
- Systémové testovanie
- Akceptačné testovanie

1.1.1 V-model

V-model sa používa na ilustráciu súvislosti medzi fázami vývoja softvéru a testovania, a zabezpečuje, že testovanie prebieha paralelne s fázami vývoja softvéru. Každá fáza vývoja má príslušnú fázu testovania, ktorá zodpovedá za overenie správneho fungovania daného kroku vývoja. [1] [2]



Obrázok 1. V-Model (zdroj: vlastný)

- **Analýza požiadaviek:**

  Analýza požiadaviek je prvá fáza vývojového cyklu, kde sa požiadavky softvérového produktu rozumejú z pohľadu zákazníka. Táto fáza obsahuje detailnú komunikáciu so zákazníkom, aby sa pochopili jeho očakávania a presné požiadavky. Jedná sa o podstatnú činnosť, ktorá si vyžaduje správne riadenie, vzhľadom na skutočnosť, že väčšina zákazníkov si nie je istá tým, čo presne potrebujú. Počas tejto fázy sa vykonáva plánovanie návrhu akceptačných testov, pretože požiadavky zákazníka môžu byť použité ako vstupný pre akceptačné testy. [1][2]

- **Dizajn Systému:**

  V momente, keď sú detailne spracované požiadavky klienta na produkt, nastáva fáza návrhu celého systému. Návrh systému bude obsahovať podrobný popis pre kompletné hardvérové a komunikačné nastavenia vyvíjaného produktu. Na základe návrhu systému sa vypracuje plán návrhu systémových testov. V prípade že je plán testovania navrhnutý včas, ostáva viac času na samotné vykonanie testov. [1][2]

- **Dizajn Architektúry:**

  Návrh architektúry alebo aj High-Level-Design (HLD) je fáza v ktorej sa jasne chápe a definuje prenos údajov a komunikácia medzi vnútornými modulmi. Architekti, vývojári, návrhári databázy a mnoho ďalších vytvárajú celkový technický návrh na základe prvkov, ktoré sa rozdelili pri dizajne systému. Návrh architektúry musí teda jasne obsahovať všetky požadované technické podrobnosti ohľadom har-

dvéru, softvéru, programovacieho jazyka ktorý sa má použiť, návrh databázy atď... Každý modul je spojený s konkrétnou funkciou. Znalosť týchto informácií nám umožňuje spraviť návrh a dokumentáciu integračných testov. [1][2]

- **Dizajn Modulov:**
  Návrh modulov alebo aj Low-Level-Desing (LLD) je fáza v ktorej sa na základe technickej špecifikácie v návrhu architektúry vypracováva podrobný návrh jednotlivých modulov a ich prepojenie s ostatnými modulmi. Detaily spojené s dátovými typmi objektov, ako sa jednotlivé dáta zberajú, akým spôsobom môžeme dáta integrovať s ohľadom na požiadavky zákazníka, akým spôsobom sa majú dáta ukladať a podobne sú zdokumentované v podrobnej špecifikácií návrhu modulov. Tento podrobný návrh robia jednotliví vývojári. Na základe toho sa môže spraviť návrh jednotkových testov. [1][2]
- **Implementácia:**
  Implementácia je fáza v ktorej vývojári prevezmú LLD špecifikáciu, ktorú využijú pri programovaní jednotlivých modulov. Programovanie sa musí vykonávať v súlade s určitými smernicami a štandardmi špecifického jazyka, ktorý sa bude používať. Daný kód prechádza mnohými kontrolami, spätnou väzbou a implementáciou, aby bol optimálne napísaný pre najlepší výkon pred finálnym preložením do spustiteľného programu. [1][2]

1.1.2 Jednotkové testovanie

Jednotkové testovanie je testovanie, ktoré sa zameriava na najmenšie testovateľné časti aplikácie (jednotky) kam spadajú napríklad moduly, objekty a triedy. Cieľom jednotkového testovania je overiť správne fungovanie každej jednotky a zabezpečiť, že každá jednotka funguje správne v izolácií od ostatných jednotiek. Zvyčajne tento druh testovania robia samotní vývojári, kde porovnávajú množiny vstupov s očakávanými výstupmi daných metód [1].

1.1.3 Integračné testovanie

Integračné testovanie slúži pre správne overenie fungovania integrovaných modulov systému. Najskôr sa testuje integrácia medzi dvoma modulmi a následne sa pridávajú ďalšie. Cieľom tohoto testovanie je odhaliť chyby pri komunikácií medzi jednotlivými modulmi systému. Integračné testovanie sa rozdeľuje na vnútorné a vonkajšie. Vnútorné resp. modulové integračné testovanie slúži pre správnu vnútornú komunikáciu modulov. Toto testo-

vanie obvykle robia vývojári a zvykne byť automatizované. Vonkajšie resp. systémové integračné testovanie slúžia k prepájaniu aplikácie do väčších funkčných celkov. Tento druh testovania pripravuje testovací tím. V prípade, že sú správne vykonané nasledujúce úrovne testovania, je možné integračné testovanie vynechať bez negatívneho dopadu na softvér, pretože chyby, ktoré by sme prípadne odhalili integračným testovaním by sa určite prejavili pri ďalších úrovniach testovania [1].

1.1.4 Systémové testovanie

Systémové testovanie sa zameriava na overenie funkčnosti aplikácie ako celku. Toto testovanie obsahuje end-to-end testy, ktoré majú za cieľ otestovať funkčnosť aplikácie s pomocou skutočných scenárov. Tieto testy bývajú zvyčajne automatizované a toto testovanie sa vykonáva väčšinou po integračnom testovaní na simulovanom prostredí.[2]

1.1.5 Akceptačné testovanie

Akceptačné testovanie podobne ako aj systémové sa zameriava na overenie kvality softvéru ako celku. Avšak rozdiel je v tom, že akceptačné testovanie sa zameriava priamo na splnenie požiadaviek definovaných zákazníkom a na to, či je softvér pripravený na nasadenie do produkčného prostredia. Akceptačné testy sa zvyčajne vykonávajú v reálnom prostredí na skutočných dátach. [2]

Existuje niekoľko typov akceptačných testov:

- Používateľské akceptačné testovanie – tento druh testovania zvyčajne vykonáva bežný používateľ softvéru na základe daných požiadaviek.
- Kontraktové akceptačné testovanie – tento druh testovania overuje, že boli splnené všetky požiadavky a kritéria stanovené pri kontrakte.
- Regresné akceptačné testovanie – tento druh testovania overuje, či je aplikácia funkčná aj po pridaní nových funkcionalít, vylepšení a nedošlo náhodou ku chybe. Regresné testy bývajú zvyčajne automatizované.
- Alfa a Beta testovanie – tieto formy testovania slúžia k spätnej väzbe. Alfa testovanie je vykonávané interne s internými používateľmi a Beta testovanie je vykonávané externe s verejnými používateľmi. [2]

1.2 Techniky testovania softvéru

Techniky testovania softvéru sú metódy, ktoré sa používajú na overenie funkcionality a kvality softvéru. Existuje mnoho techník testovania softvéru, ktoré sa používajú na dosiahnutie rôznych cieľov a kritérií testovania. Každá technika testovania má svoje vlastné postupy a pravidlá, ktoré sa musia dodržiavať, aby boli výsledky testovania presné a spoľahlivé. Cieľom použitia techník testovania softvéru je identifikovať chyby, nedostatky alebo chýbajúce funkcionality v softvéri a zabezpečiť, aby bol softvér bezchybný a spĺňal očakávania zákazníkov. Techniky testovania sa používajú v rôznych fázach vývoja softvéru, od testovania jednotlivých modulov po testovanie celého systému. Niektoré z techník testovania softvéru sú testovanie čiernej skrinky, testovanie bielej skrinky, testovanie šedej skrinky, statické testovanie, dynamické testovanie. Každá z týchto techník má svoje výhody a nevýhody a vyberá sa podľa požiadaviek projektu a typu softvéru, ktorý sa testuje.[3]

1.2.1 Testovanie čiernej skrinky

Testovanie čiernej skrinky spočíva v tom, že funkcionality aplikácie sú testované bez znalosti vnútornej štruktúry kódu, tester teda nevie, ako softvér funguje vo vnútri. Táto technika testovania sa zameriava primárne na vstupné hodnoty a očakávané výstupy softvéru. [2]

Techniky testovania čiernej skrinky:

Nasledujúce techniky testovania sa zameriavajú primárne na vstupné hodnoty a očakávané výstupy softvéru. Existuje viacero techník akými je možné testovať vstupy a výstupy bez znalosti vnútorného kódu, každá má svoje benefity a negatívna. Je dôležité vedieť vybrať správnu techniku testovania vzhľadom na špecifikované požiadavky. [2]

- **Testovanie ekvivalentných tried**
  Táto technika testovania zahŕňa testovanie vstupov, ktoré patria do rovnakej triedy ekvivalencie. Pri tejto technike sa vstupné hodnoty do systému alebo aplikácie rozdelia do rôznych tried alebo skupín na základe ich podobnosti vo výsledku. Napríklad ak by sme mali validné vstupné hodnoty v rozmedzí od 20 do 50, tak by sme mohli vytvoriť tri triedy.
    1. 19 vrátane a menej
    2. 51 vrátane a viac
    3. 20 až 50 vrátane

Týmto spôsobom sme boli schopní zredukovať testovacie prípady len na 3 s tým, že sme pokryli všetky možnosti. Na otestovanie scenára by nám stačila tým pádom len jedna hodnota z každej množiny tried. [1]

- **Analýza hraničných hodnôt**

  Táto testovacia technika, ako vyplýva aj z názvu spočíva v testovaní hraničných hodnôt validných a nevalidných vstupov. Je možné ju implementovať v momente, kedy vieme určiť hraničné hodnoty. Napríklad ak by sme mali testovať validné vstupné hodnoty v rozmedzí od 20 do 50, potom namiesto toho aby sme používali všetky hodnoty v rozmedzí 20 až 50, využijeme hraničných hodnôt 19 (20-1), 20, 21 (20+1), 49 (50-1), 50, 51(50+1). Cieľom tohoto testovania je zistiť, akým spôsobom softvér reaguje na vstupy, ktoré sú tesne nad alebo pod hranicou platnosti. [2]

- **Testovanie podľa rozhodovacej tabuľky**

  Testovanie podľa rozhodovacej tabuľky je testovacia technika, ktorá sa používa na identifikáciu kombinácií vstupov a očakávaných výstupov v softvérovom systéme. Táto technika využíva tabuľku, ktorá obsahuje všetky možné kombinácie vstupov a očakávaných výstupov a určuje, ktoré kombinácie je nutné otestovať. Príklad použitia testovania podľa rozhodovacej tabuľky:

  [2] Predstavme si, že je navrhovaný jednoduchý systém pre prihlasovanie používateľov a chceme otestovať jeho funkčnosť. Rozhodneme sa použiť testovanie podľa nasledujúcej rozhodovacej tabuľky:

Tabuľka 1. Rozhodovacia tabuľka (zdroj: vlastný)

Vstup	Email obsahuje '@'	Heslo má minimálne 8 znakov	Potvrdenie hesla sa zhoduje s heslom	Výstup
Platné údaje	Áno	Áno	Áno	Úspešná registrácia
Neplatný email	Nie	-	-	Chyba: Neplatný email
Krátke heslo	Áno	Nie	-	Chyba: Heslo je príliš krátke
Nezhodné heslá	Áno	Áno	Nie	Chyba: Heslá sa nezhodujú

1.2.2 Testovanie bielej skrinky

Na rozdiel od techniky testovania čiernej skrinky, ktorá sa zameriava na testovanie aplikácie z externého používateľského pohľadu, je technika testovania bielej skrinky založená na vnútornej znalosti infraštruktúry kódu. A bez jej znalosti teda nie sme schopní naplno využívať benefitov, ktoré nám táto testovacia technika ponúka.

Testovania bielej skrinky sa vykonáva najčastejšie na úrovni jednotkových testov, ktoré testujú jednotlivé časti kódu, ako aj na úrovni integračných testov, ktoré testujú spoluprácu viacerých modulov dohromady. Sústredí sa primárne na zlepšenie bezpečnosti, plynulého toku vstupov a výstupov skrz aplikáciu a zlepšenie dizajnu a použiteľnosti softvéru.

Hlavnou výhodou testovania bielej skrinky je, že umožňuje identifikovať skryté chyby a slabiny v softvéru, ktoré by mohli byť inak prehliadané pri iných druhoch testovania. Pri testovaní bielej skrinky sa využívajú rôzne techniky testovania, ako sú napríklad riadenie toku programu, testovanie hraníc, testovanie podmienok a rozhodnutí, testovanie funkcií a testovanie výkonu. [1]

1.2.2.1 Techniky testovania bielej skrinky

Nasledujúca kapitola bude zameraná na techniky testovania bielej skrinky. Tieto techniky sú dôležité najmä kvôli potrebe testovania vnútorných častí kódu. Vďaka nasledujúcim technikám je možné detailnejšie pochopiť štruktúru daného kódu, jeho efektivitu a vnútor-

nú logiku. Taktiež sa pomocou týchto techník dá detailnejšie určiť pokrytie zdrojového kódu testami. [6]

Testovanie riadiaceho toku (control flow testing)

Táto testovacia technika je založená na správnom riadení toku programu, ktorý môžeme demonštrovať skrz graf riadiaceho toku, ktorý sa zameriava na správne zobrazenie štruktúry programu a navyše poskytuje zobrazenie toku riadenia medzi jednotlivými časťami kódu. Taktiež sa jedná o techniku testovania, ktorá spadá pod testovanie bielej skrinky. Tým pádom môžeme skonštatovať, že sa primárne budeme zaoberať vnútrom daného kódu programu a bez jeho poznania nie sme schopní optimálne otestovať jeho funkcionalitu za použitia tejto testovacej techniky.

V prvom rade sa táto technika využíva v jednotkovom testovaní, kde sú testovacie scenáre zaznamenané pomocou grafu riadiaceho toku. Táto technika je určená pre zistenie akýchkoľvek chýb alebo nízkej konzistentnosti a nepredvídanosti správania softvéru. [6]

Výhody využívania testovania riadiaceho toku sú:

Sme schopní vďaka tejto technike detegovať polku chýb počas testovania jednotiek. Touto technikou vieme detegovať zhruba 1/3 chýb celého programu. A v neposlednej rade môže byť táto technika prevedená manuálne alebo automatizovane vďaka grafu, ktorý môže byť spravený ručne alebo pomocou softvéru. Správna implementácia testovacej techniky riadenia toku dát bude mať za následok zlepšenie kvality a spoľahlivosti kódu. [6]

Nevýhody využívania testovania riadiaceho toku sú:

Môže byť zložité prísť na chýbajúce cesty v prípade ak model a aj program robil ten istý človek čo potencionálne môže vyústiť do nestatočného pokrytia aplikácie testami. Ďalšou nevýhodou je, že technika nemusí byť schopná identifikovať iné typy chýb, na ktoré sa nezameriava ako napríklad chyby výkonu, alebo aj chyby v dátových štruktúrach. [6]

Existuje viacero techník ktoré sa využívajú počas testovania riadiaceho toku a sú to:

1. **Pokrytie príkazov (Statement coverage):**
   Pokrytie príkazov je technika testovania bielej skrinky, ktorá obsahuje vykonávanie všetkých príkazov ktoré sa nachádzajú v kóde aspoň raz“. Jedná sa o nižšiu úroveň testovania, tým pádom nie je záruka pokrytia všetkých možných riadiacich ciest. Pomocou tejto techniky sme schopní skontrolovať to, či daný zdrojový kód koná podľa našich očakávaní alebo nie. Nevýhodou tejto techniky je, že s ňou nie sme

schopní testovať nepravdivé podmienky. *Pokrytie príkazov = (Počet vykonaných príkazov / Celkový počet príkazov) * 100 „* [6]

2. **Pokrytie vetvenia (Branch coverage):**

   Testovacia technika pokrytia vetvení spadá pod techniky testovania bielej skrinky a primárne sa zameriava na pokrytie všetkých možných vetvení v kóde aspoň raz, pričom jedno vetvenie vieme kategorizovať ako IF podmienku, kde pravda a nepravda sú 2 vetvy IF podmienky. Tým pádom vieme využiť túto techniku pre pravdivé ale aj nepravdivé podmienky, čo sme napríklad u techniky pokrytia príkazov nemohli a nám pri tejto technike dáva vyššiu úroveň pokrytia. *„Pokrytie vetvení = (Počet vykonaných vetvení / celkový počet vetvení)“*. [6]

3. **Pokrytie Podmienok (Condition coverage):**

   Technika pokrytia podmienok (občas nazývaná aj technika pokrytia výrazu) je technika testovania bielej skrinky ktorá má za účel preskúmanie výstupu výrazov s logickými operandmi, ktoré sa využívajú napríklad v logických podmienkach IF, kde sa musia otestovať obidva možné výstupy podmienky a to teda buď pravda (True) alebo nepravda (False). Bez týchto dvoch výstupov nie sme schopný adekvátne využívať túto testovaciu techniku. Touto podmienkou sme schopní eliminovať potencionálnu chybovosť logických výrazoch ktoré sa vyskytujú v kóde. [6]

4. **Pokrytie Ciest (Path coverage):**

   Pokrytie ciest spadá taktiež pod jednu z viacerých techník testovania bielej skrinky. Táto testovacia technika spočíva v testovaní všetkých možných ciest daného programu aspoň raz. Jedná sa o komplexnú techniku ktorá je využívaná najmä pri testovaní komplexnejších aplikácií. Výhody tejto testovacej techniky spočítajú v znížení redundantných testov, zlepšenie logiky programu a vykonaní všetkých testovacích scenárov aspoň raz. Jej efektívnosť prevyšuje techniku pokrytia vetvení a určite by jej využitie malo byť zvážené pri tvorbe testovacieho plánu.[6]

Graf riadiaceho toku (Control Flow Graph – CFG) pozostáva z nasledujúcich častí:

- **Uzly (Nodes):**

Uzly sú bloky kódu ktoré slúžia pre vytvorenie cesty pre postupnosť inštrukcií bez vetvenia. Každý uzol zodpovedá jednému alebo viacerým príkazom v programe.

- **Hrany (Edges):**

Hrany vo svojej podstate reprezentujú smer riadiaceho toku medzi uzlami kódu. Slúžia ako ukazovateľ na nasledujúci blok kódu.

- **Rozhodovacie uzly (Decision Nodes):**

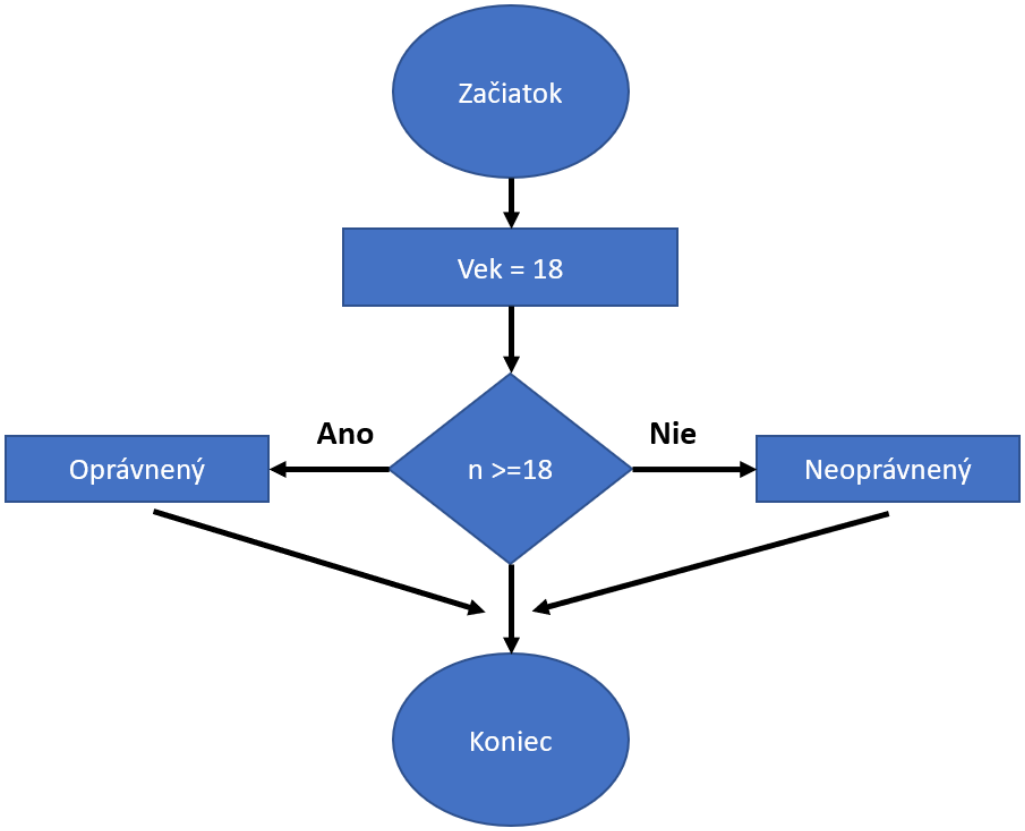
Činnosť rozhodovacích uzlov je rozhodnúť o tom, aký nasledujúci uzol sa má vykonať na základe poslednej vykonanej operácie. Operácie bývajú väčšinou podmienky if-else, cykly (for, while) alebo aj príkazy switch. Ich splnenie alebo nesplnenie vyústi v smer nasledujúceho uzla.

- **Spojovacie uzly (Junction Nodes):**

Spojovacie uzly sú miesta v grafe, kde sa aspoň tri hrany stretávajú v jednom bode. Toto miesto spája viacero vetvení dohromady a pokračuje ako jeden riadiaci tok

Tieto časti spolu tvoria graf riadiaceho toku, ktorý poskytuje vizuálnu reprezentáciu štruktúry programu a umožňuje analyzovať a testovať riadiaci tok v rámci softvéru. [6]

Príklad grafu riadiaceho toku:



Obrázok 2. Graf riadiaceho toku (zdroj: vlastný)

V tomto grafe je vidno, že začiatok, vek, oprávnený, neoprávnený a koniec sú uzly.

N>= 18 je rozhodovací uzol.

Šípky zobrazujúce smer operácií sú hrany.

Spojovací uzol vzniká nad uzlom „Koniec“, kde sa tri hrany stretávajú. [6]

Testovanie dátového toku (Data flow testing)

Testovanie dátového toku predstavuje techniku testovania softvéru, ktorá sa zaoberá analýzou pohybu dát medzi rôznymi časťami programu. Tento prístup k testovaniu je zameraný na zabezpečenie správnej manipulácie s dátami v kóde a na identifikáciu potenciálnych chýb alebo problémov súvisiacich s riadením dátových štruktúr.

Pri testovaní dátového toku sa využíva riadiaci graf programu, ktorý poskytuje prehľad o vzťahoch medzi jednotlivými časťami kódu. Pomocou tohto grafu sa analyzujú rôzne scenáre, ktoré sa týkajú interakcií s premennými, a na základe tejto analýzy sa vytvárajú testovacie scenáre.

Využívanie testovania dátového toku pomáha v riešení nasledovných problémov:

- Vyhľadaní a identifikovaní premenných, ktoré sa používajú v kóde, ale neboli a nie sú definované.
- Vyhľadaní a identifikovaní premenných, ktoré sú definované, ale nikdy sa nepoužívajú.
- Vyhľadaní a identifikovaní premenných, ktoré sú definované viacnásobne pred ich použitím.

Testovanie dátového toku je technika, ktorá sa primárne zameriava na odstraňovanie problémov súvisiacich so správnym spracovaním údajov. V neposlednom rade pomáha zvyšovať dôveru v dobrú funkčnosť aplikácie. [7][6]

1.2.3 Testovanie šedej skrinky (Gray Box Testing)

Testovanie šedej skrinky je technika testovania softvéru, ktorá kombinuje prvky testovania čiernej skrinky a testovania bielej skrinky. Zahrňuje čiastočnú znalosť vnútornej štruktúry aplikácie. Tester síce nemá úplný prístup k zdrojovému kódu, ale pozná vnútorné dátové štruktúry, architektúru a algoritmy, na základe ktorých je schopný tvoriť testovacie prípady. Techniky testovania šedej skrinky sa snažia dosiahnuť rovnováhu medzi výhodami testovania čiernej a bielej skrinky.[8]

Obrázok 3. GrayBoxTesting [8]

Ciele testovania šedej skrinky sú nasledujúce:

1. Využiť benefitu testovania čiernej skrinky a zároveň aj bielej skrinky.
2. Zahrňovať vstupy vývojárov a testerov naraz kvôli celkovému zlepšeniu kvality produktu.
3. Znižovať celkovú časovú náročnosť procesu funkcionálneho a nefunkcionálneho testovania.
4. Poskytovať dostatok času vývojárom na odstraňovanie chýb.
5. Zahŕňať skôr pohľad používateľa než testera alebo dizajnéra. [8]

1.2.3.1 Techniky testovania šedej skrinky

Maticové testovanie (Matrix Testing):

Táto testovacia technika obsahuje definíciu všetkých premenných, ktoré sa využívajú v programe. Premenné a ich hodnoty, by mali byť v súlade s požiadavkami, v opačnom prípade vzniká risk čitateľnosti programu a rýchlosti softvéru. [8]

Regresné testovanie (Regression Testing):

Regresné testovanie zahŕňa opätovné testovanie konkrétnych častí softvéru, ktoré boli pozmenené alebo vylepšené s ohľadom na to, či tieto zmeny náhodou nespôsobili negatívne alebo neočakávané zmeny. Zmeny funkcionalít, aktualizácie, či oprava bežnej chyby sebou mnoho krát prinášajú množstvo skrytých chýb, ktoré nie sú vždy zrejmé a ľahko viditeľné. Regresné testovanie rieši tieto chyby pomocou testovacích stratégií ako je napríklad opakované testovanie rizikových prípadov užitia alebo opakované testovanie v rámci firewallu. [8]

Testovanie vzorov (Pattern Testing):

Táto technika zahŕňa analýzu vzorov alebo štruktúr vo softvéri na identifikáciu potenciálnych chýb. Na základe pochopenia vnútorného dizajnu môže tester vytvárať testovacie prípady, ktoré sa zameriavajú na konkrétne vzory alebo kombinácie komponentov, aby sa dokázala určiť príčina prípadného zlyhania a dokázala sa efektívne opraviť. [8]

Testovanie ortogonálnych matíc (Orthogonal Array Testing - OAT):

Testovanie pomocou ortogonálnych matíc je systematický, štatistický spôsob testovania dvojprvkových interakcií medzi vstupnými parametrami systému. Pri OAT sa používa matica na zobrazenie kombinácií hodnôt vstupných parametrov, ktoré je potrebné otestovať. Účelom je pokryť maximum kódu s čo najmenším počtom testovacích prípadov. Táto technika testovania je veľmi užitočná v prípade testovania komplexných aplikácií. [8]

1.3 Manuálne testovanie

Manuálne testovanie je spôsob testovanie softvéru v ktorom vykonávame testovacie prípady ručne (manuálne) bez využívania nástrojov na automatizáciu týchto testov. Cieľ manuálneho testovania je identifikovať chyby, problémy či prípadné nedostatky aplikácie ktorá je testovaná a uistiť sa, že aplikácia funguje v súlade s požiadavkami. Nutnosť manuálne testovať prichádza s každou novou aplikáciou predtým, než testovanie môže byť automatizované. Manuálne testovanie nevyžaduje znalosť žiadneho testovacieho nástroja. Sada testov a prípadov by sa mala blížiť ku 100% pokrytiu. [9]

1.3.1 Postup manuálneho testovania

1. Prečítanie a porozumene dokumentácie daného projektu, nadobudnutie všeobecnej znalosti aplikácie.
2. Analýza všetkých požadovaných dokumentov ku získaniu všetkých požiadaviek koncovým používateľom.
3. Tvorba testovacích prípadov v súlade s požiadavkami uvedenými v dokumentácií.
4. Vykonanie testovacích prípadov pomocou techník testovania softvéru.
5. Prípadný výskyt chýb nahlásiť vývojovému tímu, na základe priority.
6. Po opravení chyby vývojármi opätovne testovať neúspešné testovacie prípady. [9]

1.3.2 Nástroje pre manuálne testovanie

1. Jira
    - Jira je softwarový nástroj zaoberajúci sa problematikou evidencie chýb a problémov pri vývoji softwaru alebo riadení projektov. Jira je napísaná v jazyku Java.
2. Trello
    - Trello je nástroj určený pre pohodlné riadenie projektov a efektívne sledovanie problémov.
3. Zephyr
    - Zephyr je nástroj ktorý zlepšuje viditeľnosť výsledkov testovania tým, že poskytuje dobre organizovaný plán rozloženia.
4. Bugzilla
    - Bugzilla je nástroj učený pre sledovanie chýb. Sleduje chyby a postupne pripravuje výpis hlásení o chybách.
5. MantisBT
    - MantisBT odkazuje na „Mantis Bug Tracer“. Systém na sledovanie chýb, ktorý je online a je taktiež freewarový nástroj. Jeho implementácia je v jazyku PHP. [9]

1.3.3 Výhody manuálneho testovania

- Schopnosť odhaliť drvivú väčšinu chýb.
- Testeri majú prístup ku komponentom ako rozvrhnutie stránky, text a tak ďalej. Dokážu lepšie identifikovať problémy spojené s používateľským prostredím.
- Dobrá adaptácia na neplánované zmeny v aplikácií.
- Nízke náklady.
- Nie je potreba náročnej technickej zručnosti [9]

1.3.4 Nevýhody manuálneho testovania

- Časová náročnosť vzhľadom na fakt, že každý testovací prípad musí byť vykonaný ručne.
- Nemožnosť testovať výkon a zaťaženie.
- Nízka rýchlosť a efektivita pri veľkých súboroch a regresných testoch.

- Nízka konzistentnosť vďaka ľudskému faktoru, ktorý môže mať za následok prehliadnutie niektorých chýb. [9]

1.4 Automatizované testovanie

Automatizované testovanie je metóda testovania softvérových produktov, ktorá slúži na porovnávanie skutočného výsledku s očakávaným výsledkom. Využíva mnoho špeciálnych nástrojov, pomocou ktorých možno testovacie prípady vykonávať opakovane. Tento typ testovania je preferovanejší, vzhľadom na fakt, že vďaka svojej efektívnosti, rýchlosti a spoľahlivosti, šetrí čas a náklady spojené s testovaním. Automatizované testovanie nachádza svoje miesto hlavne pri rozsiahlych projektoch, regresných testoch, alebo pri testovaní výkonu a záťaže. Jeho uplatnenie je pomerne široké a v dnešnej dobe sa bez jeho uplatnenia nezaobíde takmer žiadna aplikácia. Zámerom tohoto testovania je taktiež minimalizovať ľudskú potencionálnu chybovosť pri opakovanej implementácií testov.

1.4.1 Nástroje pre automatizáciu testovania

1. Selenium
    - Selenium je veľmi populárny open-source (kód je prístupný verejnosti) nástroj na testovanie webových aplikácií a prehliadačov. Ponúka jednotné rozhranie, ktoré umožňuje písať testovacie skripty v programovacích jazykoch ako C#, Python, Java, PHP, Ruby.
2. SikuliX
    - Sikuli je nástroj na automatizáciu testovania, ktorý sa zameriava na testovanie grafického používateľského rozhrania. Je open-source.
3. Apache JMeter
    - Jmeter je nástroj na vykonávanie záťažových testov (koľko užívateľov môže naraz softvér obslúžiť).
4. QuickTestProfessional (QTP)
    - QuickTestProfessional je nástroj určený na vykonávanie automatizovaného regresného testovania, vďaka ktorému sme schopní identifikovať nedostatky a chyby medzi skutočnými a požadovanými výsledkami.
5. SoapUI
    - SoaapUI je open-source nástroj, ktorý sa využíva na testovanie aplikačných programových rozhraní [11].

1.4.2 Výhody Automatizovaného testovania

- Spoľahlivosť, vďaka využívaniu efektívnych testovacích nástrojov.
- Rýchlosť a efektivita procesu testovania softvéru.
- Možnosť rýchlejšieho dodania produktu na trh
- Presnosť
- Možnosť testovať naraz veľké množstvo testovacích prípadov
- Možnosť opakovane púšťať testovacie prípady

1.4.3 Nevýhody automatizovaného testovania

- Komplexnosť
- Vysoké počiatočné náklady
- Nevhodnosť pri testovaní grafického používateľského rozhrania
- Náročnosť dizajnu testov, ktoré sú spoľahlivé a zároveň udržiavateľné
- Potreba prepisovať testy v prípade zmeny prostredia

[12]

1.4.4 Porovnanie Manuálneho Testovania s Automatizovaným

1. Vykonávanie testovacích prípadov:
    - Manuálne:
        - Na vykonávanie testovacích prípadov pri manuálnom testovaní je potrebný človek, ktorý simuluje koncových používateľov.
    - Automatizovane:
        - Na vykonávanie testovacích prípadov pri automatizovanom testovaní sú určené nástroje, ktoré vykonávajú dané testovanie automaticky.
2. Náklady potrebné na testovanie:
    - Manuálne:
        - Náklady potrebné na manuálne testovanie sú vysoké v dlhodobom horizonte, ale nízke v krátkodobom, pretože nie je nutnosť využívania nástrojov na automatizáciu.
    - Automatizovane:
        - Náklady potrebné na automatizované testovanie sú zo začiatku vysoké kvôli nákupu alebo vývoju nástroju na automatizáciu, avšak z dlhodobého horizontu sú náklady nižšie.

3. Čas:
    - Manuálne:
        - Čas potrebný na vykonávanie testovacích prípadov pri manuálnom testovaní zväčša býva časovo náročnejší oproti automatizovanému, pretože je závislí od ľudí
    - Automatizované:
        - Čas potrebný na vykonávanie testovacích prípadov pri automatizovanom testovaní zvykne byť nižší oproti manuálnemu, práve vďaka nástrojom ktoré umožňujú vykonávanie testovacích prípadov automatizovane.
4. Konzistencia:
    - Manuálne:
        - Konzistencia pri manuálnom testovaní nie je tak vysoká ako pri automatizovanom práve kvôli možnej chybovosti a nekonzistentnosti človeka.
    - Automatizované:
        - Konzistencia pri automatizovanom testovaní je pomerne vysoká, pretože každý test je vykonávaný rovnakým spôsobom.
5. Použiteľnosť:
    - Manuálne:
        - Použiteľnosť manuálneho testovania je veľmi flexibilná, vzhľadom na fakt, že hocijaký typ aplikácie môže byť testovaný manuálne a určité techniky testovania ako napríklad opičie testovanie je skôr rozumnejšie vykonávať pomocou manuálneho testovania než pomocou automatizovaného testovania. Použiteľnosť manuálneho testovania je taktiež efektívnejšia než pri automatizovanom napríklad pri odhaľovaním vizuálnych problémov spojených s rozložením alebo farbami.
    - Automatizované:
        - Použiteľnosť automatizovaného testovania je vysoká, vzhľadom na rýchlosť a možnosť ho vykonávať nepretržite, čo má za následok zvýšenie efektívnosti testovania. Pre lepšiu efektivitu je vhodné ho vykonávať na stabilnejších systémoch.

V praxi sa často používa kombinácia manuálneho a automatizovaného testovania, aby sa maximalizovali výhody oboch metód a minimalizovali ich slabiny. [10]

2 VÝROBNÝ INFOMAČNÝ SYSTÉM V SPOLOČNOSTI CONTINENTAL BARUM S.R.O.

V tejto kapitole sa priblíži daný výrobný informačný systém v spoločnosti Continental Barum s.r.o. ktorý bol testovaný v rámci tejto bakalárskej práce. Taktiež sú v tejto kapitole popísané technológie na ktorých je daný systém postavený.

2.1 Informácie o spoločnosti

Continental Barum s.r.o. je súčasťou automobilového priemyslu. Je zameraná na výrobu gumových pneumatík so sídlom v Českej republike, konkrétne v Otrokoviciach. Od roku 1992 je súčasťou nemeckého koncernu „Continental AG“. Firma disponuje približne 4300 zamestnancami, a jej obrat bol 59 mld. Kč za rok 2021.

2.2 Opis výrobného informačného systému

Spoločnosť Continental Barum s.r.o. používa niekoľko desiatok výrobných informačných systémov na správu a produkciu gumových pneumatík. Výrobný informačný systém ktorý je predmetom mojej bakalárskej práce je teda „CEMSET“ (Centralized Management System for Extrusion Tools) v preklade je to centralizovaný systém riadenia tlačiacich nástrojov. Používa sa na správu foriem pre tlačiace stroje, umožňuje vytvárať objednávky na konkrétne typy materiálov a podrobne určovať ich špecifikácie, poskytuje plánovanie výroby a mnoho ďalších funkcionalít. Domovská stránka CEMSET vyzerá nasledovne, obsahuje viacero položiek navigácie, pre názornú ukážku stránky popíšem položku navigácie „Similarity & Order Creation“.

Obrázok 4. CEMSET Domovská Stránka (zdroj: vlastný)

Položka navigácie „Similarity & Order Creation“ obsahuje štyri podpoložky a to sú:

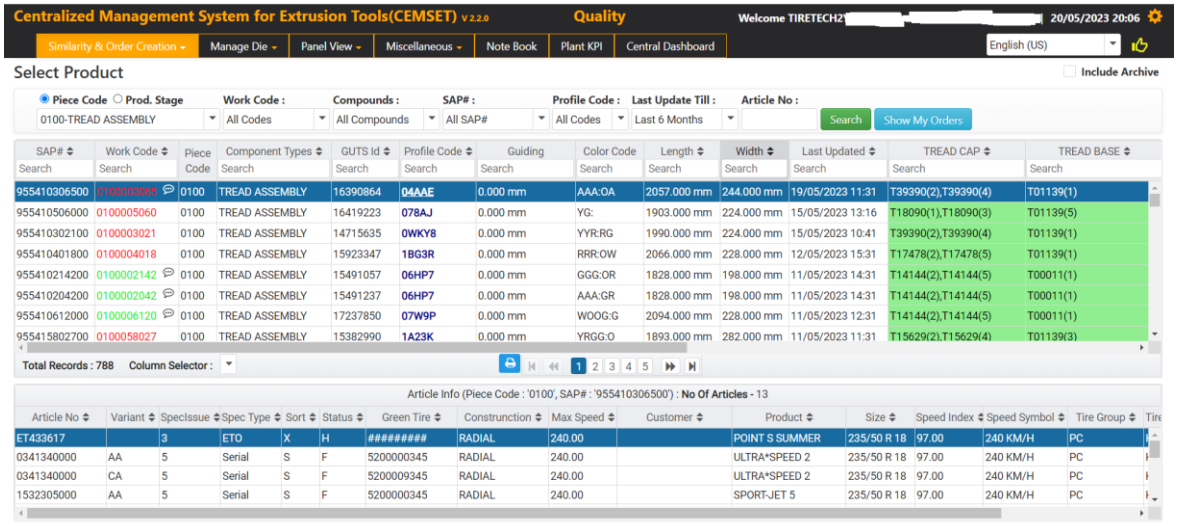
1. Select Product
2. Select Product For Die Adjustment
3. Similarity Check Report
4. Specification View
5. Auto CLT Profile Comparison List

Vzhľadom na rozsiahlosť stránky popíšem podpoložku navigácie „Select Product“ (výber produktu) a znázorním životný cyklus objednávky.

Podpoložka navigácie výber produktu je primárne zložená z :

- Kusového kódu (Piece Code)
- Pracovných kódov (Work Code)
- Zlúčeniny uzáveru závitu, základe vlákna a ramenného pásu (Compounds, THREAD CAP, THREAD BASE, SHOULDER STRIP PROF.)
- SAP kódov (SAP#)
- Profilového kódu (Profile Code)
- Dátum poslednej aktualizácie (Last Update Till)

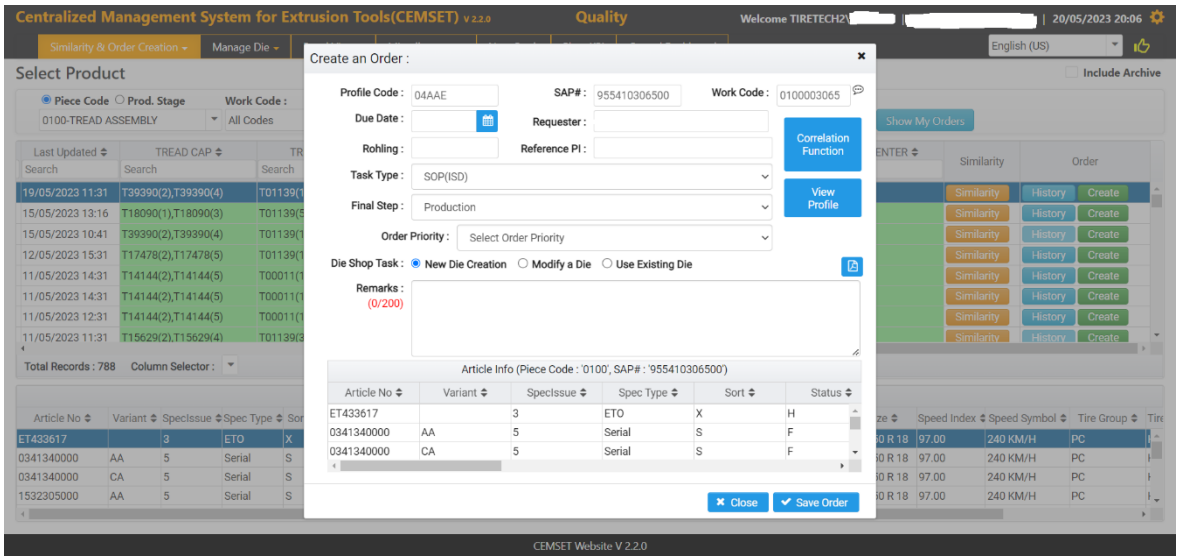
Na základe týchto kritérií môžeme vyhľadať špecifické komponenty s danými vlastnosťami pomocou tlačidla Vyhľadávanie (Search) a zobrazí sa nám táto tabuľka.



Obrázok 5. Výber produktu (zdroj: vlastný)

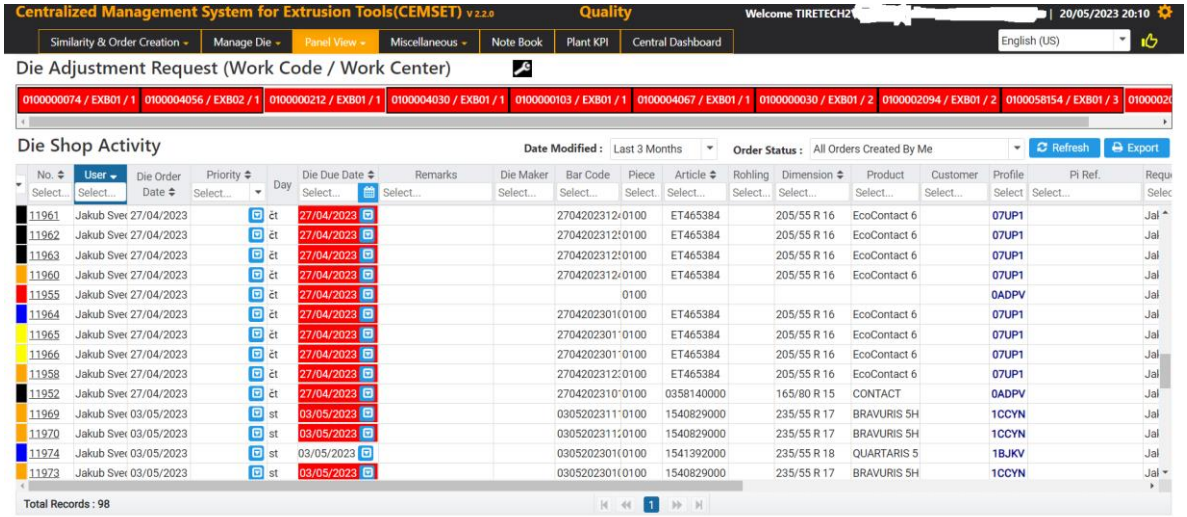
Následne máme možnosť vytvoriť objednávku pre daný typ SAP kódu pomocou tlačidla vytvoriť (Create). Je povinné vyplniť dátum splatnosti a pridať poznámku. Potom sme

schopní pomocou tlačidla uložiť objednávku (Save Order). Objednávkový formulár vyzerá nasledovne:



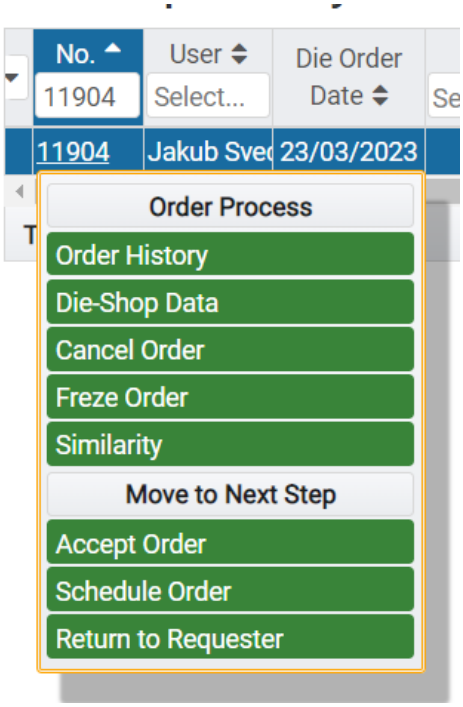
Obrázok 6. Vytvorenie objednávky (zdroj: vlastný)

Po vytvorení objednávky sa zobrazí číslo objednávky, pomocou ktorého môžeme objednávku vyhľadať v nasledujúcom zobrazení objednávok, do ktorého sa dostaneme pomocou tlačidla zobrazenie mojich objednávok (Show My Orders).



Obrázok 7. Zobrazenie objednávok (zdroj: vlastný)

V zobrazení objednávok máme možnosť zadať do vyhľadávacieho políčka No. číslo našej objednávky, kde sa nám následne vyfiltruje naša objednávka.

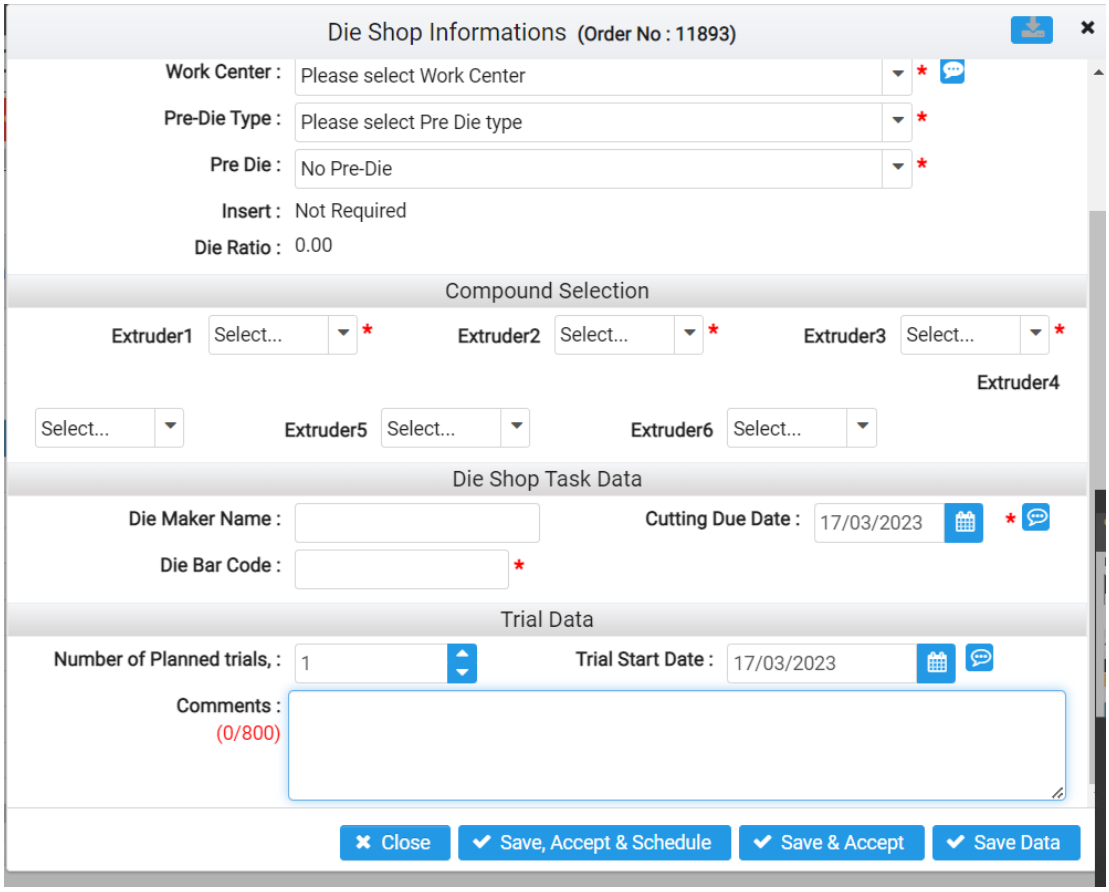


Obrázok 8. Proces Objednávky (zdroj: vlastný)

Máme možnosť dostať objednávku do nasledujúcich stavov:

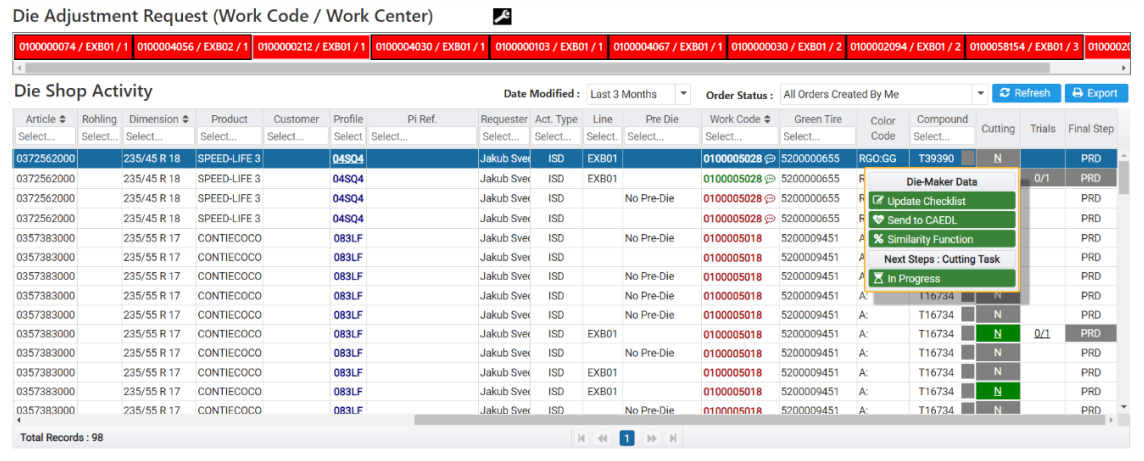
- Zrušiť objednávku (Cancel Order)
- Zmraziť objednávku (Freeze Order)
- Akceptovať objednávku (Accept Order)
- Naplánovať objednávku (Schedule Order)
- Vrátiť objednávku žiadateľovi (Return to Requester)

V momente kedy by sme sa rozhodli objednávku akceptovať, je potrebné vyplniť povinné polia vo formulári ako je pracovné stredisko (Work Center), typ predlisku (Pre-Die Type), čísla tlačiacich strojov (Extruder1-3), čiarový kód (Die Bar Code) ktorý musí byť unikátny a v neposlednom rade je povinné pridať komentár (Comments). Následne môžeme kliknúť na tlačidlo uložiť, akceptovať & naplánovať (Save, Accept & Schedule) čím danú objednávku uložíme, prijmeme a naplánujeme.



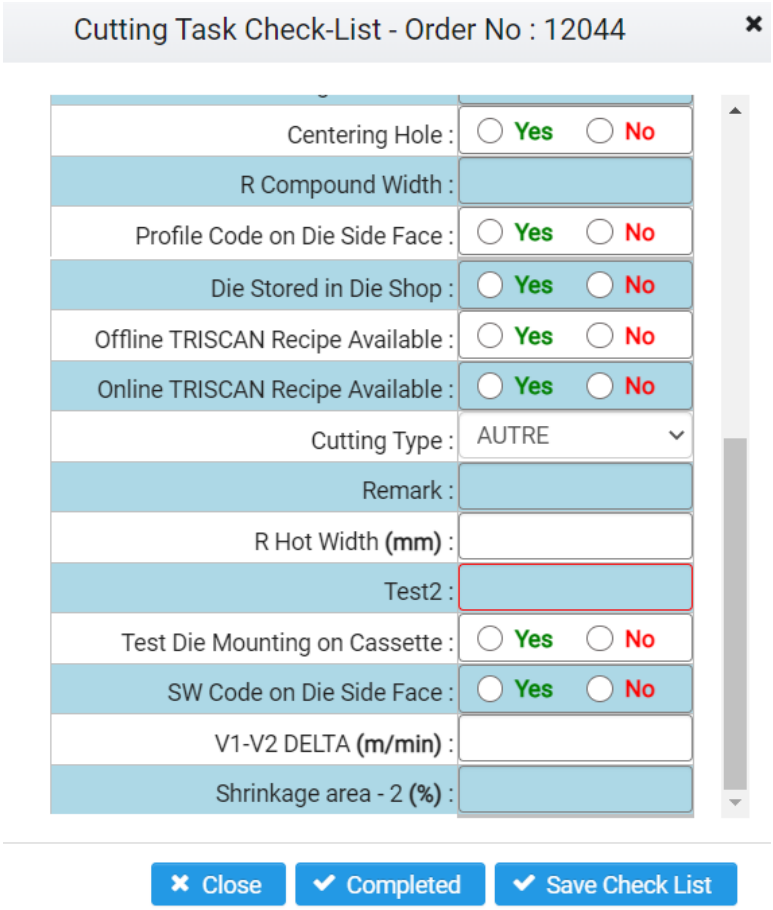
Obrázok 9. Formulár pre akceptovanie objednávky (zdroj: vlastný)

Keď je objednávka prijatá a naplánovaná tak je potrebné začať „Cutting“ časť, ktorá sa zameriava vyrezávanie. Vyberieme možnosť v progrese (In Progress).



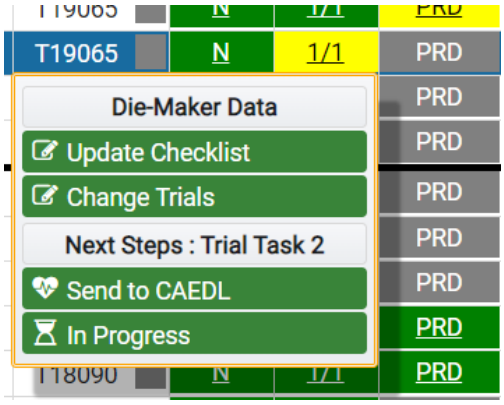
Obrázok 10. Začatie „Cutting“ časti. (zdroj: vlastný)

Následne sa zobrazí kontrolný zoznam rezných úloh, kde je povinné vyplniť druh rezania (Cutting Type) a Test2 a je možné dokončiť zoznam pomocou tlačidla dokončiť (Completed).



Obrázok 11. Kontrolný zoznam rezných úloh (zdroj: vlastný)

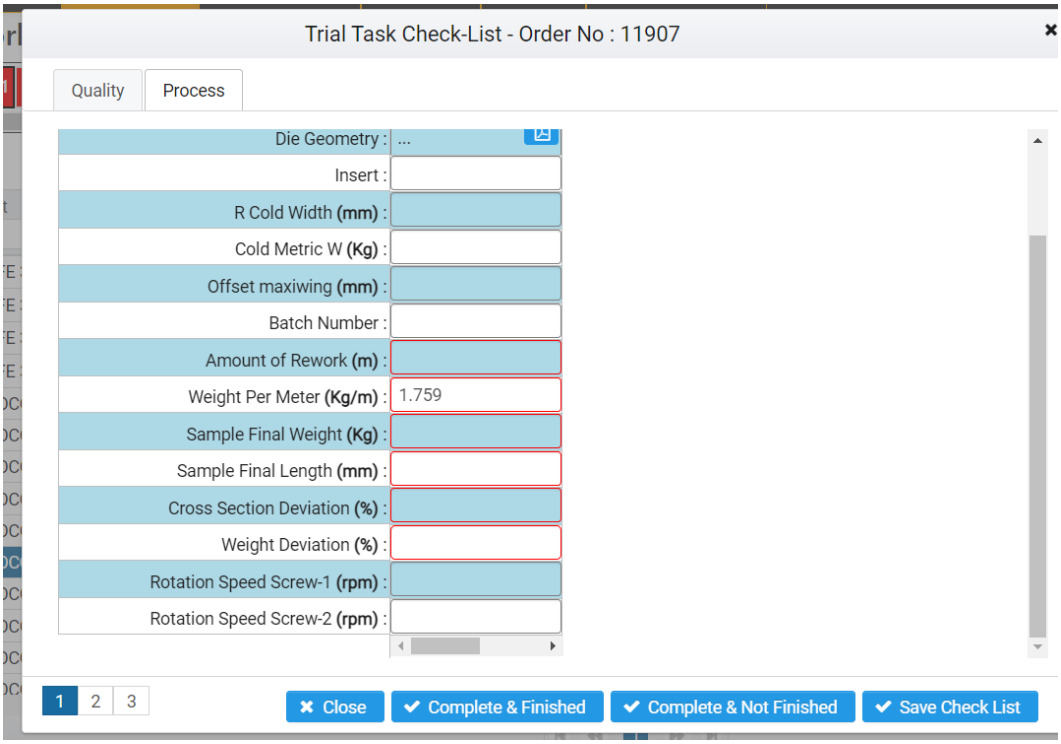
Keď je kontrolný zoznam hotový, tak je potrebné začať „Trials“ časť, ktorá sa zameriava na podrobné informácie o vykonávacej úlohe, ktorá je predmetom vytlačenia. Vyberieme možnosť v progrese.



Obrázok 12. Začatie „Trial“ časti. (zdroj: vlastný)

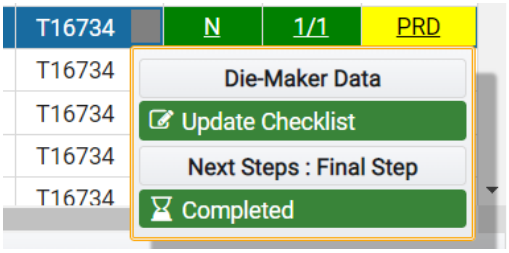
Zobrazí sa kontrolný zoznam, ktorý obsahuje povinné polia množstvo prepracovania (Amount of Rework), hmotnosť na meter (Weight Per Meter), konečná hmotnosť vzorky (Sample Final Weight), konečná dĺžka vzorky(Sample Final Length), odchýlka priečneho

rezu (Cross Section Deviaton), odchýlka hmotnosti (Weight Deviation), ktoré je potrebné vyplniť, a následne je možné kontrolný zoznam dokončiť cez tlačidlo (Complete & Finish).



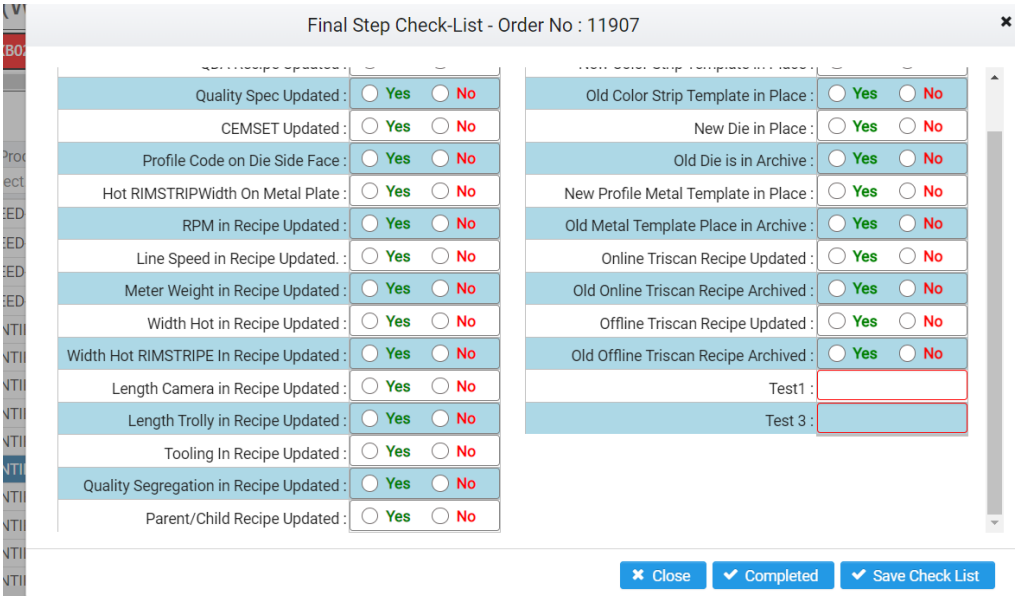
Obrázok 13. Formulár pre dokončenie „Trial“ časti. (zdroj: vlastný)

Po vyplnení formuláru je na rade posledná časť a to je záverečný krok (Final Step).



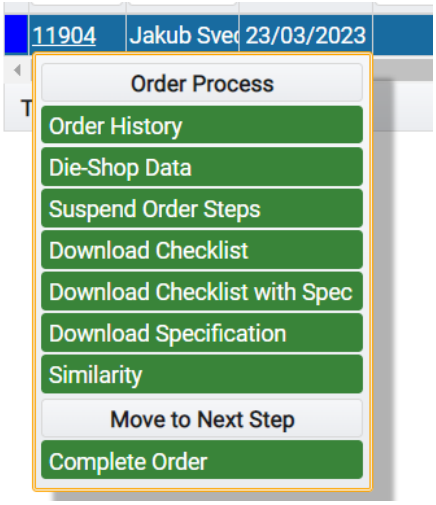
Obrázok 14. Začatie záverečného kroku. (zdroj: vlastný)

Po kliknutí na dokončené sa zobrazí kontrolný zoznam záverečných krokov. V tomto zozname je povinné vyplniť políčko Test1 a Test3.



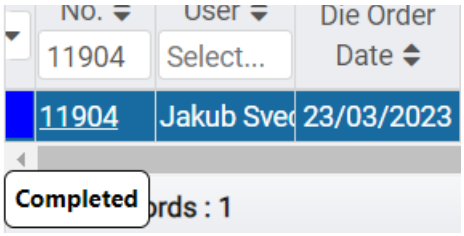
Obrázok 15. Formulár pre dokončenie finálneho kroku (zdroj: vlastný)

Po dokončení finálneho kroku už len stačí kliknúť na id objednávky a vybrať možnosť Complete Order.



Obrázok 16. Complete Order (zdroj: vlastný)

V tejto fáze je objednávka dokončená.



Obrázok 17. Completed Order (zdroj: vlastný)

2.3 Architektúra a technológie používané vo výrobnom informačnom systéme

Primárne architektúry a technológie ktoré sa využívajú vo výrobnom informačnom systéme CEMSET sú nasledujúce:

2.3.1 ASP.NET

ASP.NET je open source webový framework vytvorený spoločnosťou Microsoft, ktorý slúži na vytváranie webových stránok a webových aplikácií za pomoci HTML, CSS a JavaScriptu. Umožňuje vytvárať webové rozhrania API a používať technológie reálneho času ako sú Web Sokety. ASP.NET je multiplatformný a funguje v systémoch Windows, Linux, MacOS a Docker.

ASP.NET disponuje tromi frameworkmi učenými na tvorbu webových stránok, sú to Web Forms, ASP.NET MVC a v neposlednom rade ASP.NET Web Pages. Všetky tri frameworky poskytujú stabilitu a vyspelosť, a využitím hociktorého z nich je možné vytvoriť kvalitné webstránky.

Každý framework sa zameriava na odlišný spôsob vývoja. Výber frameworku by mal byť prispôsobený znalostiam, zručnostiam a skúsenostiam vývojového tímu. Mal by byť vhodný pre daný typ aplikácie ktorá ma byť vyvíjaná a mal by sa odvíjať od jednotlivého prístupu k vývoju vo firme. [13]

2.3.1.1 MVC

ASP.NET MVC (Model-View-Controller) je architektonický vzor, ktorý oddeľuje aplikáciu na tri hlavné komponenty: Model, View a Controller. Tento prístup pomáha udržať štruktúru kódu organizovanú a umožňuje vyššiu flexibilitu a kontrolu pri vývoji webových aplikácií.

Model: Model je časť systému, ktorá sa zaoberá logikou pre prístup k dátam. Môže to zahŕňať logiku pre prístup k databáze, manipuláciu s dátami a implementáciu a udržiavanie foriem dát.

View: View je zodpovedný za zobrazenie dát užívateľovi. Vo webovej aplikácii by to znamenalo generovanie HTML a CSS kódu, ktorý užívateľ vidí a s ktorým interaguje v internetovom prehliadači.

Controller: Controller súvisí s manipuláciou s dátami, na ktoré sa užívateľ pozerá a ktoré upravuje. To znamená, že riadi, ako sa aplikácia správa, keď užívateľ vykonáva určité akcie, napríklad kliknutie na tlačidlo alebo prechod na určitú stránku.

Výhody používania ASP.NET MVC:

Lepšia organizáciu kódu, pretože MVC oddeľuje logiku aplikácie, uľahčuje to vývojárom udržiavať kód organizovaný a zrozumiteľný.

Kontrola nad HTML, pretože ASP.NET MVC umožňuje vývojárom mať väčšiu kontrolu nad vygenerovaným HTML ako tradičné Web Forms.

Oddelenie komponentov do modelov, pohľadov a ovládačov uľahčuje jednotkové testovanie, pretože jednotlivé časti sa dajú testovať nezávisle.

Podpora pre URL (Uniform Resource Locator) smerovanie: To umožňuje vytvárať URL, ktoré sú priateľské pre vyhľadávače a používateľov.

Tento framework je tiež optimalizovaný pre vývoj moderných webových aplikácií, ktoré využívajú AJAX a jQuery pre vytvorenie plynulého a dynamického užívateľského rozhrania.

MVC sa stalo obľúbeným nástrojom pre vývoj webových aplikácií v ASP.NET a je silný a flexibilný framework pre moderný webový vývoj [14].

2.3.1.2 Ineternet Information Services

Internet Information Services (IIS) je webový server od spoločnosti Microsoft, ktorý poskytuje silnú a flexibilnú platformu pre prevádzku webových aplikácií. Verzie IIS 7 a neskoršie majú úplne modulárnu architektúru, ktorá prináša tri kľúčové výhody: Komponentizáciu, Rozšíriteľnosť a Integráciu s ASP.NET.

Komponentizácia: Všetky funkcie webového servera sú teraz spravované ako samostatné komponenty, ktoré možno jednoducho pridať, odstrániť a nahradiť. Toto prináša niekoľko kľúčových výhod oproti predchádzajúcim verziám IIS:

- Zabezpečenie servera redukciou plochy útoku. Redukcia plochy útoku je jedným z najúčinnejších spôsobov zabezpečenia serverového systému. S IIS môžeme odstrániť všetky nepoužívané serverové funkcie a dosiahnuť tak minimálnu možnú plochu útoku pri zachovaní funkcionality našej aplikácie.

- Zlepšenie výkonu a redukcia pamäťovej záťaže. Odstránením nepoužívaných serverových funkcií môžeme tiež znížiť množstvo pamäte, ktorú server používa, a zlepšiť výkon znížením množstva kódu funkcie, ktorý sa vykonáva pri každej požiadavke na našu aplikáciu.

Rozšíriteľnosť: Vývojári môžu využiť modulárnu architektúru IIS na výstavbu silných serverových komponentov, ktoré rozširujú alebo nahradzujú existujúce funkcie webového servera a pridávajú hodnotu pre webové aplikácie hostené na IIS. Medzi dôvody pre vývoj pre IIS patria:

- Posilnenie webových aplikácií. Rozšírenie IIS umožňuje webovým aplikáciám využiť funkcionality, ktoré v mnohých prípadoch nemôžu byť ľahko poskytnuté na úrovni aplikácie.
- Lepšie vývojové prostredie. Úplne nový C++ model rozšíriteľnosti rieši väčšinu problémov, ktoré predtým sužovali vývoj ISAPI, a predstavuje zjednodušené objektovo orientované API.
- Využitie plnej sily ASP.NET. Integrácia ASP.NET umožňuje rýchly vývoj serverových modulov s pomocou známych rozhraní ASP.NET 2.0 a bohatých služieb aplikácií ASP.NET. Moduly ASP.NET môžu poskytovať služby jednotne pre ASP, CGI, statické súbory a iné typy obsahu a môžu plne rozšíriť server bez obmedzení prítomných v predchádzajúcich verziách IIS.

Integrácia: ASP.NET: IIS umožňuje webovým aplikáciám plne využiť funkcie a rozšíriteľnosť ASP.NET 2.0. Funkcie ASP.NET, vrátane autentizácie založenej na formulároch, členstva a mnohých ďalších, môžu byť použité pre všetky typy obsahu. [15]

2.3.1.3 Microsoft SQL Server

Microsoft SQL Server je systém riadenia relačnej databázy vyvinutý spoločnosťou microsoft, ktorý využíva štrukturovaný dotazovací jazyk (SQL) k uloženiu a práci s dátami. SQL je jazyk pomocou ktorého sa pracuje s relačnými databázovými systémami, ako napríklad Microsoft a Oracle SQL Server a MySQL.

Databáza v SQL serveri je postavená okolo štruktúry tabuliek, ktoré spájajú príbuzné dátové prvky v rôznych tabuľkách, vďaka čomu eliminuje potrebu redundancie dát v databáze. Relačný model taktiež poskytuje referenčnú integritu a ďalšie integritné obmedzenia na zachovanie presnosti dát. Hlavnou súčasťou Microsoft SQL Server je SQL Server Data-

base Engine, ktorý je zodpovedný za riadenie, ukladanie, spracovanie a bezpečnosť dát [16].

3 AUTOMATIZOVANÉ TESTOVANIE V PROGRAME RANOREX

Nasledujúca kapitola bude znázorňovať základné princípy, vlastnosti a funkcie nástroja ktorý bol použitý pre automatizáciu testov vrátane prostredia daného nástroja. Taktiež je v tejto kapitole popísaný krokový postup pri tvorbe sady automatizovaných testov.

3.1 Úvod do systému Ranorex Studio

Ranorex Studio je nástroj na automatizáciu testovania. Tento nástroj umožňuje vytváranie, správu a vykonávanie automatizovaných testov pre široký výber softvérových aplikácií, od desktopových a webových až po mobilné aplikácie.

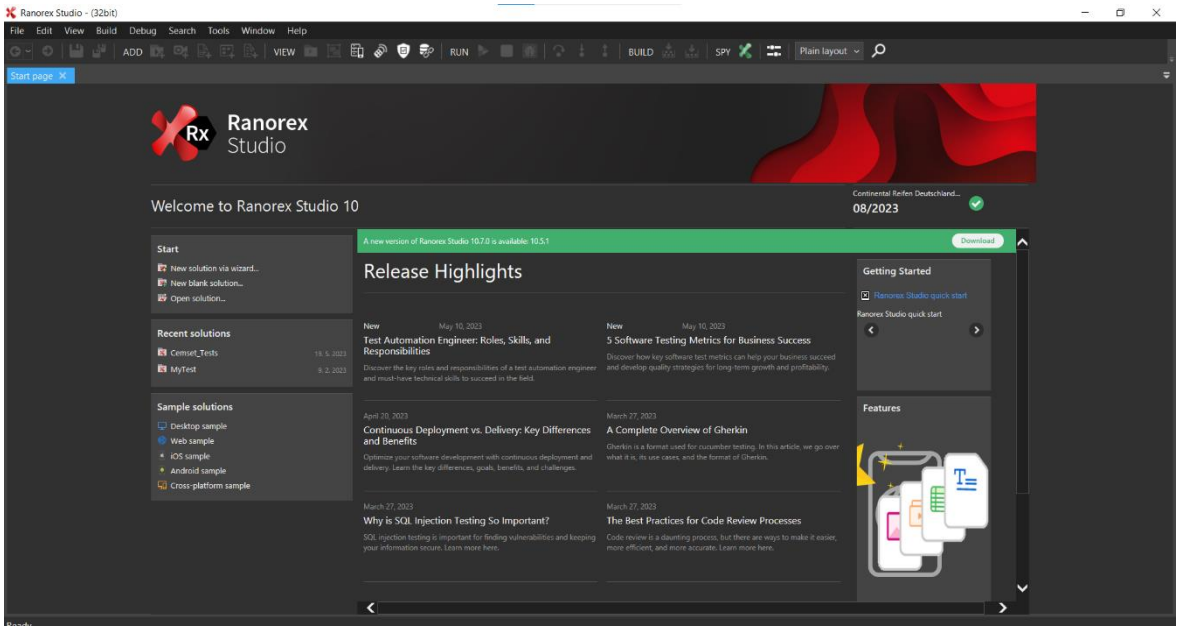
3.1.1 Vlastnosti a funkcie

Ranorex Studio ponúka širokú radu funkcií, ktoré zabezpečujú efektívne a účinné testovanie softvéru.

- **Nahrávanie a opakovanie akcií:** Ranorex Studio obsahuje funkciu nahrávania a opakovania, ktorá umožňuje testerom zachytiť sériu používateľských akcií a potom ich automaticky opakovať. Táto funkcia zjednodušuje proces tvorby testovacích prípadov tým, že eliminuje potrebu manuálneho písania testov. [4]
- **Podpora pre rôzne typy aplikácií:** Ranorex Studio podporuje testovanie širokej škály aplikácií vrátane webových, desktopových, mobilných, a viac. Toto znamená, že testeri môžu použiť jeden jediný nástroj na testovanie všetkých aspektov ich softvéru. [4]
- **Integrácia s inými nástrojmi:** Ranorex Studio je navrhnutý tak, aby sa dal ľahko integrovať s rôznymi inými nástrojmi používanými v softvérovom vývoji a testovaní, vrátane systémov na správu chýb, kontinuálnej integrácie a iných. [4]
- **Podpora pre tvorbu skriptov a programovanie:** Aj keď Ranorex Studio umožňuje vytváranie testov bez nutnosti písania kódu pomocou nahrávania a opakovania, tiež podporuje vytváranie a úpravu testov pomocou skriptov a programovania. To poskytuje väčšiu flexibilitu a kontrolu nad testami, čo je obzvlášť dôležité pre komplexné testovacie scenáre. [4]

3.1.2 Pracovné prostredie

Pracovné prostredie Ranorex Studio je navrhnuté tak, aby bolo intuitívne a užívateľsky príjemné. Hlavné prvky pracovného prostredia zahŕňajú projekty, sady testov, moduly a reporty.



Obrázok 18. Ranorex Studio (zdroj: vlastný)

- Projekty: Ranorex disponuje možnosťou tvorby projektov, ktoré zoskupujú všetky súvisiace sady testov, nahrávacie moduly a iné zdroje.
- Testovacie scenáre: Ranorex obsahuje testovacie scenáre do ktorých možno uložiť nahrávacie moduly ktoré sú navrhnuté na testovanie konkrétnych funkcií alebo častí systému.
- Nahrávacie Moduly: Nahrávacie Moduly sú jednotlivé kroky alebo skripty, ktoré sa používajú v rámci testovacích prípadov. Ranorex Studio podporuje vytváranie modulov pomocou nahrávania a opakovania alebo programovania.
- Reporty: Po vykonaní testov, Ranorex Studio generuje podrobné reporty, ktoré poskytujú prehľad o výsledkoch testov, vrátane informácií o chybách a výkonnosti.[5]

3.2 Popis krokov pri tvorbe sady automatizovaných testov

Tvorba sady automatizovaných testov je zložitý a komplexný proces, v ktorom je dôležité zvážiť viaceré kritéria a možnosti. Je preto dôležité mať správne vypracované naplánovanie a postupy, ktorými sa možno riadiť. [2]

Toto je popis základných krokov, ktoré by mali byť vykonané pri tvorbe sady automatizovaných testov:

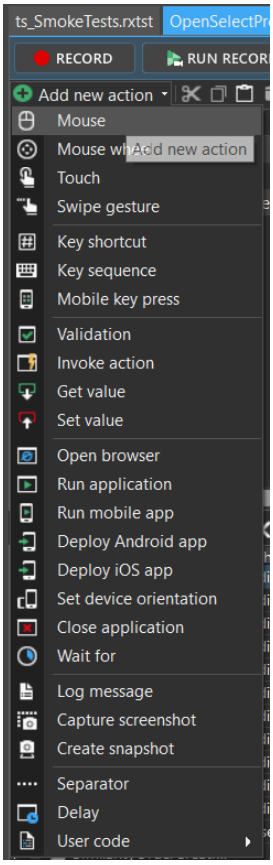
- **Analýza a plánovanie**: Tento úvodný krok je zásadným predpokladom celého procesu automatizácie testovania. Zahŕňa detailné preskúmavanie požiadaviek a špecifikácií softvéru, aby sa určilo, ktoré funkcie alebo komponenty je potrebné testovať a zároveň sa rozhodnúť o prioritách. Je potrebné určiť, ktoré aspekty softvéru by mali byť testované, v akom rozsahu a v akom poradí. Toto by malo byť založené na pochopení podnikových cieľov, požiadaviek zákazníkov, rizík a obmedzení. [2]
- **Výber nástrojov na testovanie:** Existuje mnoho nástrojov pre automatizáciu testov. Výber správneho nástroja je kľúčový pre úspešnú automatizáciu testovania. Pri výbere je potrebné zohľadniť rôzne faktory, vrátane kompatibility nástroja s technológiami používanými v projekte, podpory nástroja pre typy testov, ktoré plánujeme vykonať, a dostupné zručnosti v tíme. [2]
- **Návrh testovacích prípadov:** Po analýze a plánovaní a výbere nástrojov na testovanie prichádza na rad návrh testovacích prípadov. Testovací prípad je sada podmienok ktorými sa má systém alebo komponent testovať, aby sa overilo, že splňuje špecifikované požiadavky.[2]
- **Vývoj a implementácia testov:** Po návrhu testovacích prípadov prichádza na rad ich implementácia ako automatizovaných testov. Toto je technická fáza, ktorá môže byť najnáročnejšia, pretože vyžaduje dobrou znalosť programovacích jazykov a technológií používaných v projekte. Implementácia zahŕňa písanie testovacích skriptov, konfiguráciu testovacích prostredí a nastavenie automatizovaných nástrojov na vykonávanie testov. [2]
- **Spustenie testov a zhromažďovanie výsledkov:** Po vývoji a implementácii testov prichádza na rad ich spustenie. Automatizované testy môžu byť spustené manuálne alebo automaticky. Výsledky testov by mali byť automaticky zhromažďované a spracované, aby poskytovali jasný prehľad o stave systému. Tento krok by mal zahŕňať aj sledovanie chybových hlásení a ich analýzu.[3]

- **Analýza výsledkov a úprava testov:** Po zhromaždení výsledkov prichádza na rad ich analýza. To zahŕňa posúdenie úspešnosti testov, identifikáciu chýb, určenie ich príčin a plánovanie ďalších krokov na ich odstránenie. To môže zahŕňať úpravu existujúcich testov, pridanie nových testov, aby pokryli chyby, ktoré boli odhalené, alebo úpravu kódu systému, aby sa odstránili identifikované chyby. [3]
- **Údržba testov:** Automatizované testy si vyžadujú priebežnú údržbu, aby zostali aktuálne a relevantné. To môže zahŕňať aktualizáciu testov, aby zohľadnili nové požiadavky alebo zmeny v systéme, opravu chybných testov alebo pridávanie nových testov na pokrytie nových funkcií. Údržba testov tiež zahŕňa priebežnú revíziu a aktualizáciu testovacích prostredí a nástrojov na testovanie.[3]

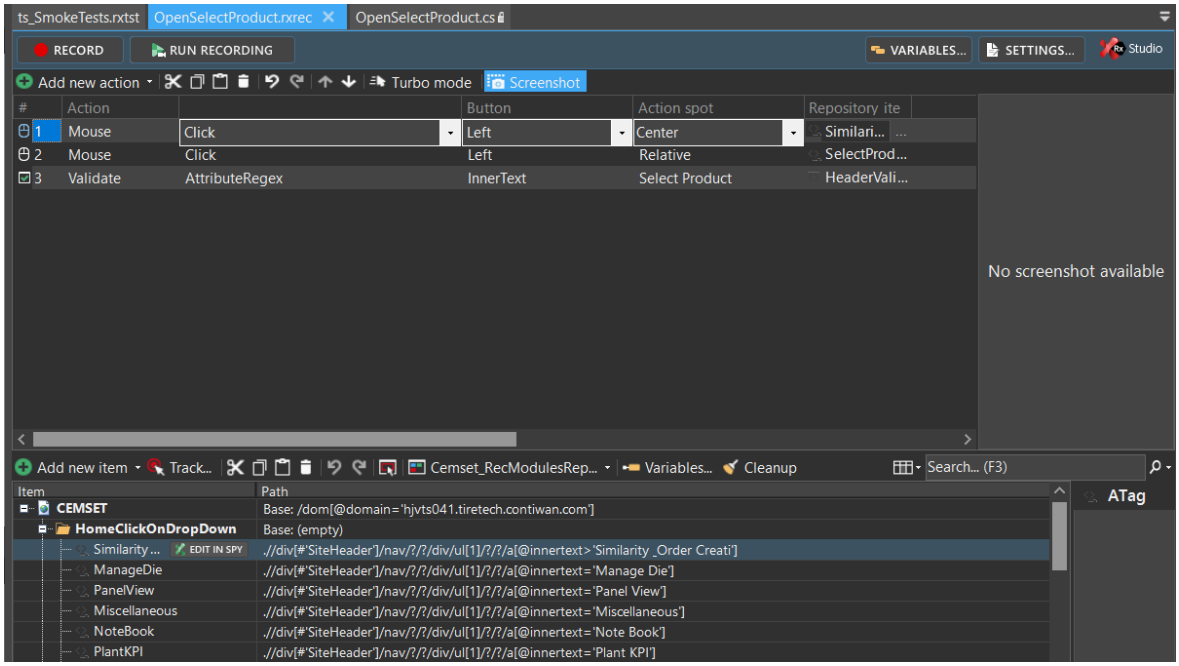
II. PRAKTICKÁ ČASŤ

4 IMPLEMENTÁCIA SADY AUTOMATIZOVANÝCH TESTOV

Na testovanie bola vybratá webová stránku CEMSET. Webová stránka bola doteraz testovaná len manuálne. Automatizované testy boli robené pomocou nástroja Ranorex Studio. Jednalo sa o tvorbu funkcionálnych testov. V rámci praktickej časti boli vytvorené 4 testovacie sady, ktoré dohromady obsahujú 37 unikátnych testovacích prípadov. Jednotlivé testovacie prípady obsahujú nahrávacie moduly, ktoré obsahujú kroky daného testovacieho prípadu. Sú to akcie, ktoré má systém vykonať pri spustení. Jednotlivé akcie možno pridávať pomocou tlačidla „Add new action“ kde je možnosť na výber z mnoho možností, ako napríklad: obyčajné kliknutie myšou (Mouse), zadanie textového vstupu z klávesnice (Key sequence), validácia (Validation), získania hodnoty (Get value), otvorenie prehliadača (Open browser), zavretie aplikácie (Close application), vytvorenie screenshotu (Capture Screenshot) a mnoho ďalších. Jednotlivé akcie sa pridávajú do nahrávacieho modulu podľa toho, aký krok chceme vykonať k jednotlivým akciám je potrebné pridať „repository item“, ktorý reprezentuje xpath elementu. Xpath elementu sa generuje pomocou tlačidla „track“, v prípade potreby je možné túto xpath elementu zmeniť pomocou nástroja rozšírenia v podobe RanorexSpy.



Obrázok 19. Add new action (zdroj: vlastný)

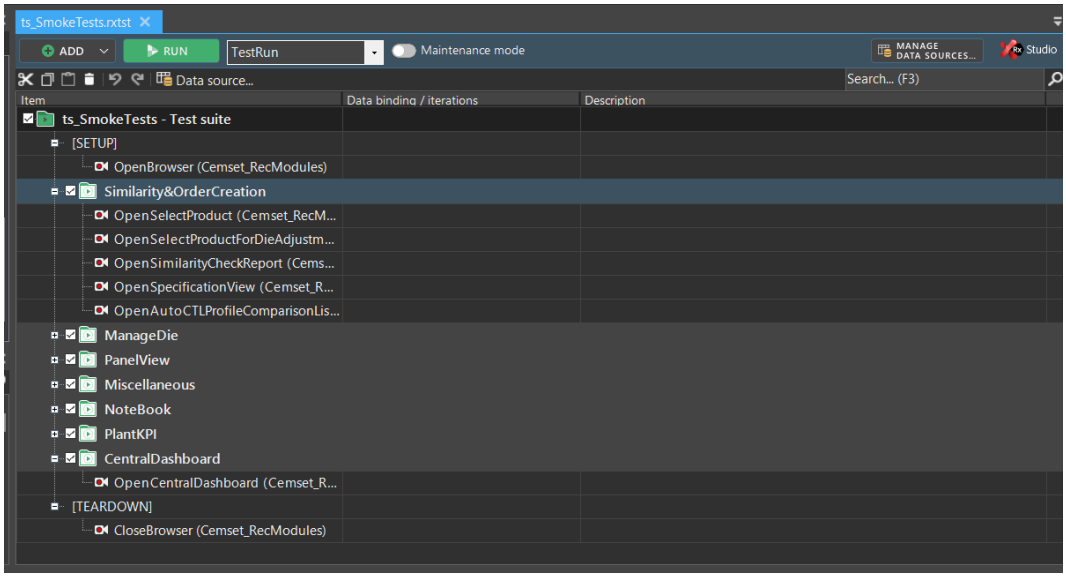


Obrázok 20. Nahrávací modul „OpenSelectProduct“ (zdroj: vlastný)

Každá sada testov sa skladá zo Setupu, ktorý ma za úlohu otvoriť prehliadač s danou stránkou a prepnúť jazyk stránky do angličtiny. Nasledujú testovacie scenáre s nahrávacími modulmi a na konci je Teardown, ktorý má za úlohu vypnúť prehliadač.

4.1 Ts_SmokeTests

Ts_SmokeTests bola prvá sada automatizovaných testov, ktorá bola vytvorená. Jedná sa o sadu smoke testov, ktorých úloha bola otvoriť a validovať všetky položky a ich podpoložky navigácie na domovskej stránke.



Obrázok 21. Testovacia sada ts_SmokeTests (zdroj: vlastný)

Vzhľadom na jednoduchosť daných testovacích prípadov a workflow ranorexu sú v tejto sade reprezentované dané testovacie prípady ako nahrávacie moduly a skutočný testovací prípad „Similarity&OrderCreation“ je scenárom, ktorý ich zlučuje dohromady, pretože majú za úlohu otvorenie podpoložiek navigácie Similarity&OrderCreation. Rovnaký princíp je aplikovaný aj pre ostatné testovacie prípady vrátane tejto sady. Toto riešenie sa môže zdať neprehľadné, avšak je to určitá metodika softvéru ranorex, ktorá nám ponúka možnosť pridania viacerých nahrávacích modulov do jedného testovacieho prípadu.

4.2 Názorná ukážka testovacieho prípadu tejto sady:

Vypracoval: Jakub Švec

Popis scenáru: Validácia presmerovania u všetkých položiek a podpoložiek navigácie domovskej stránky.

Popis prípadu: Presmerovanie z domovskej stránky na výber produktu

Testovacie kroky:

1. Kliknutie na položku „Similarity & Order Creation“.
2. Výber podpoložky „Select Product“.

Preconditions:

Otvorenie Chrome prehliadača s URL adresou CEMSET webu.

Testovacie dáta:

NULL

Očakávaný výsledok:

Presmerovanie na stránku „Select Product“.

4.3 Štruktúra sady ts_SmokeTests:

ID Testovacieho prípadu: Similarity&OrderCreation

- ID Nahrávacích modulov
    - OpenSelectProduct
    - OpenSelectProductForDieAdjustment
    - OpenSpecificationView
    - OpenAutoCLTProfileComparisonList

ID Testovacieho prípadu: ManageDie

- ID Nahrávacích modulov:
    - OpenDieLocationManagement
    - OpenAdjustmentHistory
    - OpenManagePendinAdjustment

ID Testovacieho prípadu: PanelView

- ID Nahrávacích modulov:
    - OpenCompoundStockView
    - OpenDieShopActivity
    - OpenDieMakerViewTrials
    - OpenDieMakerShift
    - OpenDieMakerTask
    - OpenDieFlexibilityMatrix
    - OpenReworkAmountForecast

ID Testovacieho prípadu: Miscellaneous

- ID Nahrávacích modulov:
    - OpenCompoundGroupSimilarity
    - OpenEmailSubscriptions
    - OpenManageAdjustmentWork
    - OpenDashBoardSettings
    - OpenManageDieShopCapacity
    - OpenManageGlobalChecklist
    - OpenManageGraphitiSettings
    - OpenManageChecklist
    - OpenMnageInserts
    - OpenMnageLogs
    - OpenManageOrderPriority
    - OpenPageHelp
    - OpenManagePlantSettings
    - OpenManagePreDie
    - OpenManageReferenceValues
    - OpenManageRoleAction

- OpenManageShiftConfiguration
- OpenMnaageTranslations
- OpenMnagaeUsers
- OpenSimilaritySearchCriteria
- OpenWorkCenterView

ID Testovacieho prípadu: NoteBook

- ID Nahrávacích modulov:
  - OpenNotebook

ID Testovacieho prípadu: PlantKPI

- ID Nahrávacích modulov:
  - OpenPlantKPI

ID Testovacieho prípadu: CentralDashboard

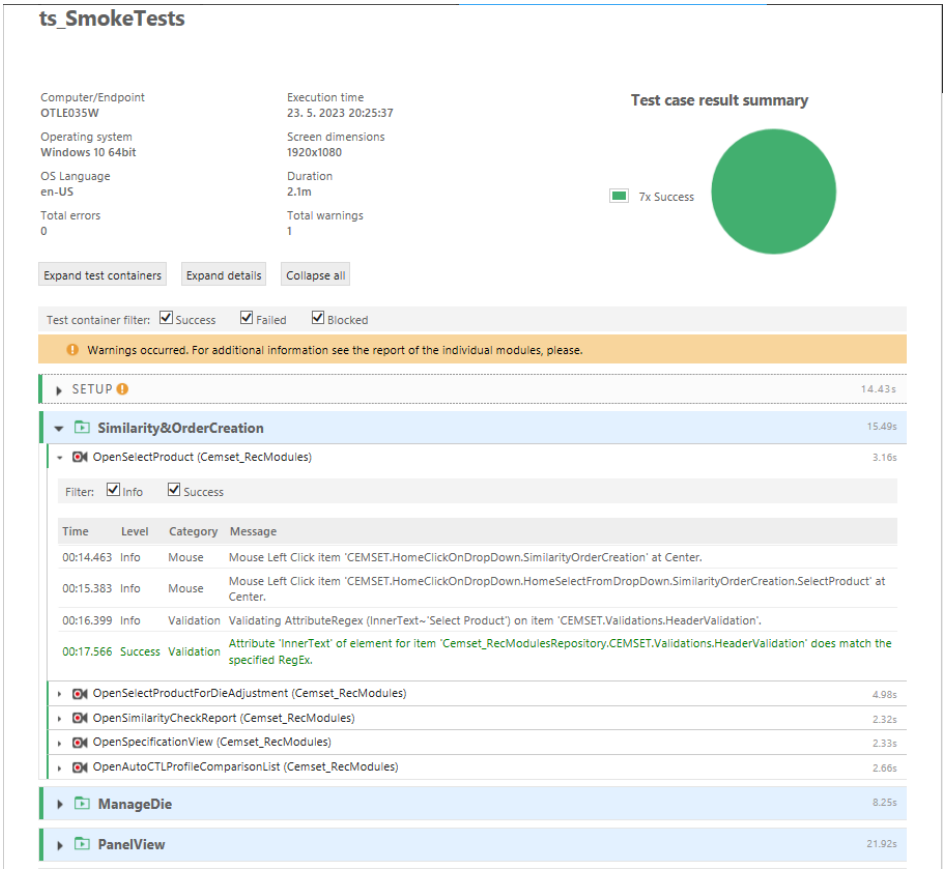
- ID Nahrávacích modulov:
  - OpenCentralDashboard

TEARDOWN

- ID Nahrávacieho modulu:
  - CloseBrowser

Report:

Po spustení danej sady testov pomocou tlačidla „RUN“ začne systém vykonávať jednotlivé testovacie scenáre a ich nahrávacie moduly. Po skončení sa automaticky zobrazí report úspešnosti jednotlivých testovacích prípadov, ktoré sú v tejto sade testov rozdelené tak, že testovacie prípady reprezentujú testovacie scenáre a jednotlivé nahrávacie moduly reprezentujú testovacie prípady.

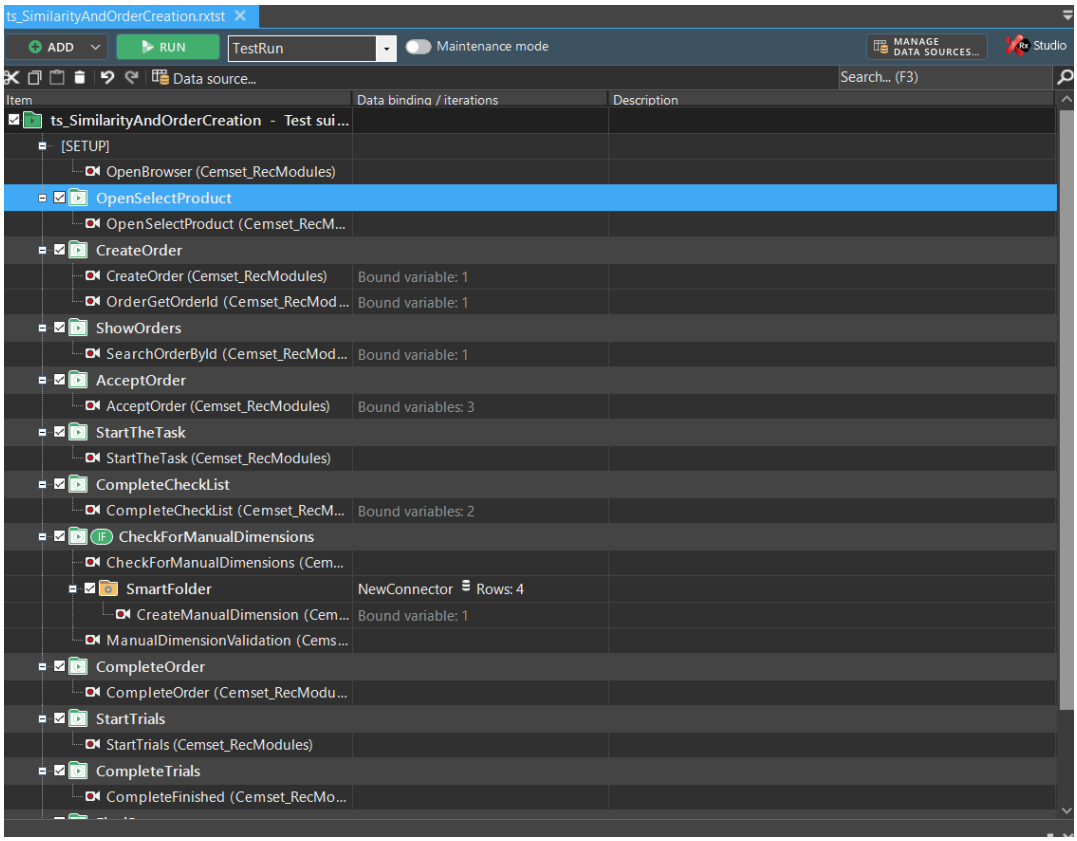


Obrázok 22. ts_SmokeTests (zdroj: vlastný)

Na základe reportu môžeme skonštatovať, že sa vykonali všetky nahrávacie moduly a teda aj testovacie prípady v rámci danej sady úspešne. V reporte sa nachádzalo dohromady sedem testovacích scenárov a šestnásť nahrávacích modulov. Čas vykonávania testovacích prípadov bol 2.1 minúty.

4.4 ts_SimilarityAndOrderCreation

Jedná sa o sadu automatizovaných testov, ktorá mala za úlohu otestovať životný cyklus objednávky. Obrázok sady vyzerá nasledujúco:



Obrázok 23. ts_Similarity&OrderCreation (zdroj: vlastný)

Na obrázku možno vidieť testovaciu sadu ts_Similarity&OrderCreation, ktorá je zameraná na vytvorenie, vyhľadanie, spracovanie a dokončenie danej objednávky. V skratke testovacia sada testuje životný cyklus objednávky od vytvorenia až po uzavretie.

4.5 Názorná ukážka testovacieho prípadu CreateOrder:

ID Testovacej sady: ts_Similarity&OrderCreation

ID Testovacieho scenára: Similarity&OrderCreation

ID Testovacieho prípadu: CreateOrder

Vypracoval: Jakub Švec

Popis scenáru: Vytvorenie, zobrazenie, akceptovanie, dokončenie objednávky.

Popis prípadu: Vytvorenie objednávky a získanie ID objednávky.

Testovacie kroky pre recording modul CreateOrder:

1. Kliknutie na položku Search
2. Kliknutie na tlačidlo Create
3. Kliknutie na tlačidlo Okay
4. Zavolanie metódy GetCurrentDate()
5. Vyplnenie položky Poznámky hodnotou „test_svec“
6. Kliknutie na tlačidlo Save Order
7. Kliknutie na tlačidlo Yes

Testovacie kroky pre recording modul OrderGetOrderId:

1. Získanie InnerTextu elementu s ID objednávky a priradenie do premennej RightOrderID.
2. Zavolanie Metódy TrimOrderId(RightOrderID)
3. Kliknutie na tlačidlo Ok

Preconditions:

Nachádzať sa na podpoložke navigácie Select Product

Testovacie dáta:

CurrentDate

test_svec

RightOrderId

Očakávaný výsledok:

1 Zobrazenie jednotlivých výrobkov
2 Zobrazenie info okna
3 Zobrazenie formuláru
4 Premenná CurrentDate bude obsahovať aktuálny dátum
5 Položka Due Date bude vyplnená unikátnym dátumom
6 Položka Poznámky bude vyplnená hodnotou „test_svec“
7 Zobrazenie potvrdzujúceho okna
8 Zobrazenie okna s úspešným uložením objednávky a zobrazenie ID danej objednávky
9 Premenná RightOrderId bude obsahovať neorezaný string obsahujúci ID objednávky

10 Premenná RightOrderId sa prepíše a bude obsahovať orezaný string už len s ID objednávky

11 Zavretie okna s číslom objednávky.

4.6 Štruktúra sady ts_Smilarity&OrderCreation

SETUP

- ID Nahrávacieho modulu:
    - OpenSelectProduct

ID Testovacieho prípadu: OpenSelectProduct

- ID Nahrávacieho modulu:
    - OpenSelectProduct

ID Testovacieho prípadu: CreateOrder

- ID Nahrávacích modulov:
    - CreateOrder
    - OrderGetOrderId

ID Testovacieho prípadu: ShowOrders

- ID Nahrávacieho modulu:
    - SearchOrderById

ID Testovacieho prípadu: AcceptOrder

- ID Nahrávacieho modulu:
    - AcceptOrder

ID Testovacieho prípadu: StartTheTask

- ID Nahrávacieho modulu:
    - StartTheTask

ID Testovacieho prípadu: CompleteCheckList

- ID Nahrávacieho modulu:
    - CompleteCheckList

ID Testovacieho prípadu: CheckForManualDimensions

- ID Nahrávacích modulov:

- CheckForManualDimensions
- CreateManualDimension
- ManualDimensionValidation

ID Testovacieho prípadu: CompleteOrder

- ID Nahrávacieho modulu:
  - CompleteOrder

ID Testovacieho prípadu: StartTrials

- ID Nahrávacieho modulu:
  - StartTrials

ID Testovacieho prípadu: CompleteTrials

- ID Nahrávacieho modulu:
  - CompleteFinished

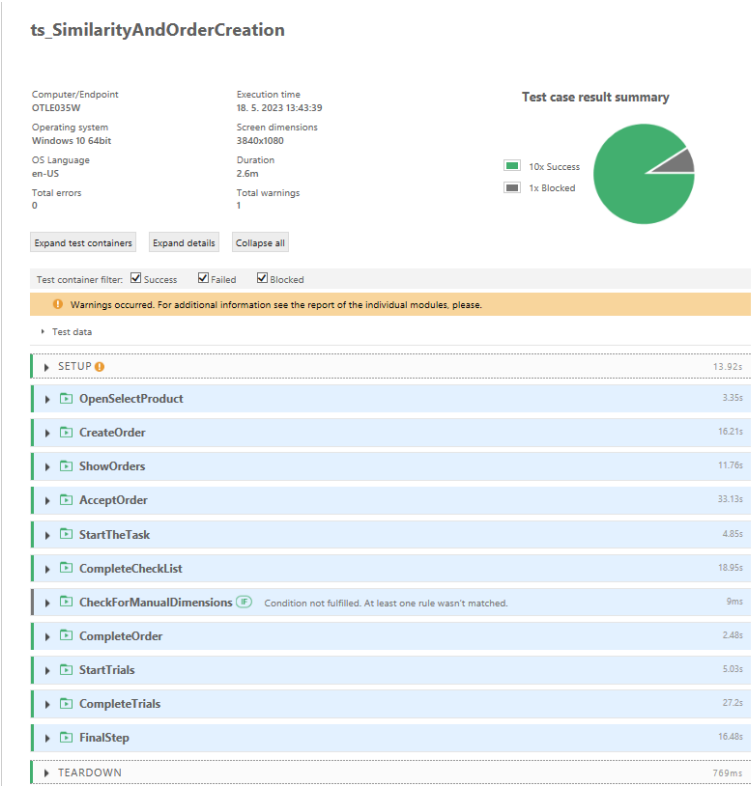
ID Testovacieho prípadu: FinalStep

- ID Nahrávacieho modulu:
  - FinishFinalStep

TEARDOWN

- ID Nahrávacieho modulu:
  - CloseBrowser

Report:

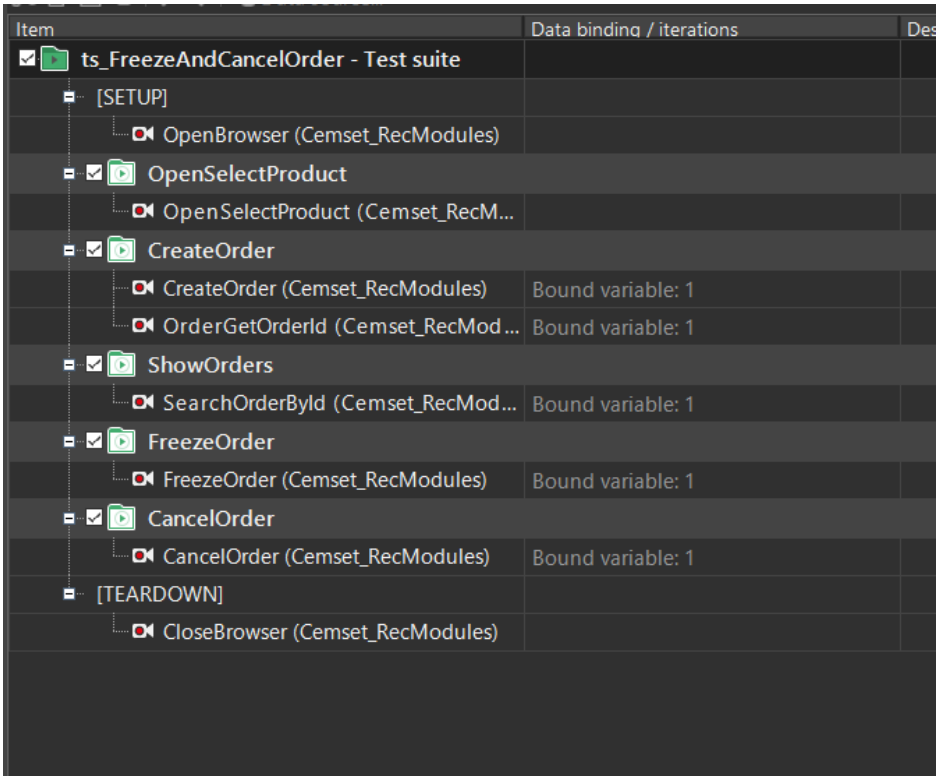


Obrázok 24. Report ts_Similarity&OrderCreation (zdroj: vlastný)

Na základe reportu môžeme skonštatovať, že sa vykonalo trinásť nahrávacích modulov a desať testovacích prípadov v rámci danej testovacej sady. Jeden testovací prípad a tri nahrávacie moduly neboli vykonané, pretože nebola splnená podmienka. Čas vykonávania testovacích prípadov bol 2.6 minúty.

4.7 ts_FreezeAndCancelOrder

Sada automatizovaných testov ts_FreezeAndCancelOrder je zameraná na životný cyklus objednávky z pohľadu zmrazenia a zrušenia objednávky. V danej sade testov sa nachádzajú testovacie prípady, ktoré sme už využili v sade testov ts_Similarity&OrderCreation a to konkrétne *OpenSelectProduct, CreateOrder* a *ShowOrders.* Táto testovacia sada je rozšírená o testovacie prípady FreezeOrder a CancelOrder. V prvom rade bolo treba vytvoriť objednávku, následne uložiť jej ID a potom ju vyhľadať. Keď sme objednávku mali vyhľadanú, bolo možné ju zmraziť. Po tom čo bola objednávka zmrazená sme ju museli odmraziť, aby sme ju následne mohli zrušiť.



Obrázok 25. ts_FreezeAndCancelOrder (zdroj: vlastný)

4.8 Názorná ukážka testovacieho prípadu CancelOrder:

ID Testovacej sady: ts_FreezeAndCancelOrder

ID Testovacieho scenára: FreezeAndCancelOrder

ID Testovacieho prípadu: CancelOrder

Vypracoval: Jakub Švec

Popis scenáru: Zrušenie objednávky

Popis prípadu: Zrušenie objednávky

Testovacie kroky pre recording modul CancelOrder:

1. Kliknutie na položku obsahujúcu objednávku
2. Kliknutie na položku Cancel Order
3. Vyplnenie textového poľa textom *test*
4. Kliknutie na tlačidlo Yes
5. Kliknutie na položku obsahujúcu objednávku
6. Kliknutie na tlačidlo Close

Preconditions:

ID objednávky zadané vo vyhľadávaní objednávok

Testovacie dáta:

test

Očakávaný výsledok:

1 Zobrazenie menu s možnosťou vybratia procesu objednávky
2 Zobrazenie okna s textovým polom určeným na dôvod zrušenia objednávky
3 Textové pole bude obsahovať text *test*
4 Daná objednávka je zrušená
5 Zobrazenie okna obsahujúceho informáciu o zrušenej objednávke
6 Zatvorenie okna

4.9 Štruktúra sady ts_FreezeAndCancelOrder

SETUP

- ID Nahrávacieho modulu:
    - OpenSelectProduct

ID Testovacieho prípadu: OpenSelectProduct

- ID Nahrávacieho modulu:
    - OpenSelectProduct

ID Testovacieho prípadu: CreateOrder

- ID Nahrávacích modulov:
    - CreateOrder
    - OrderGetOrderId

ID Testovacieho prípadu: ShowOrders

- ID Nahrávacieho modulu:
    - SearchOrderById

ID Testovacieho prípadu: FreezeOrder theos dores

- ID Nahrávacieho modulu:
    - FreezeOrder

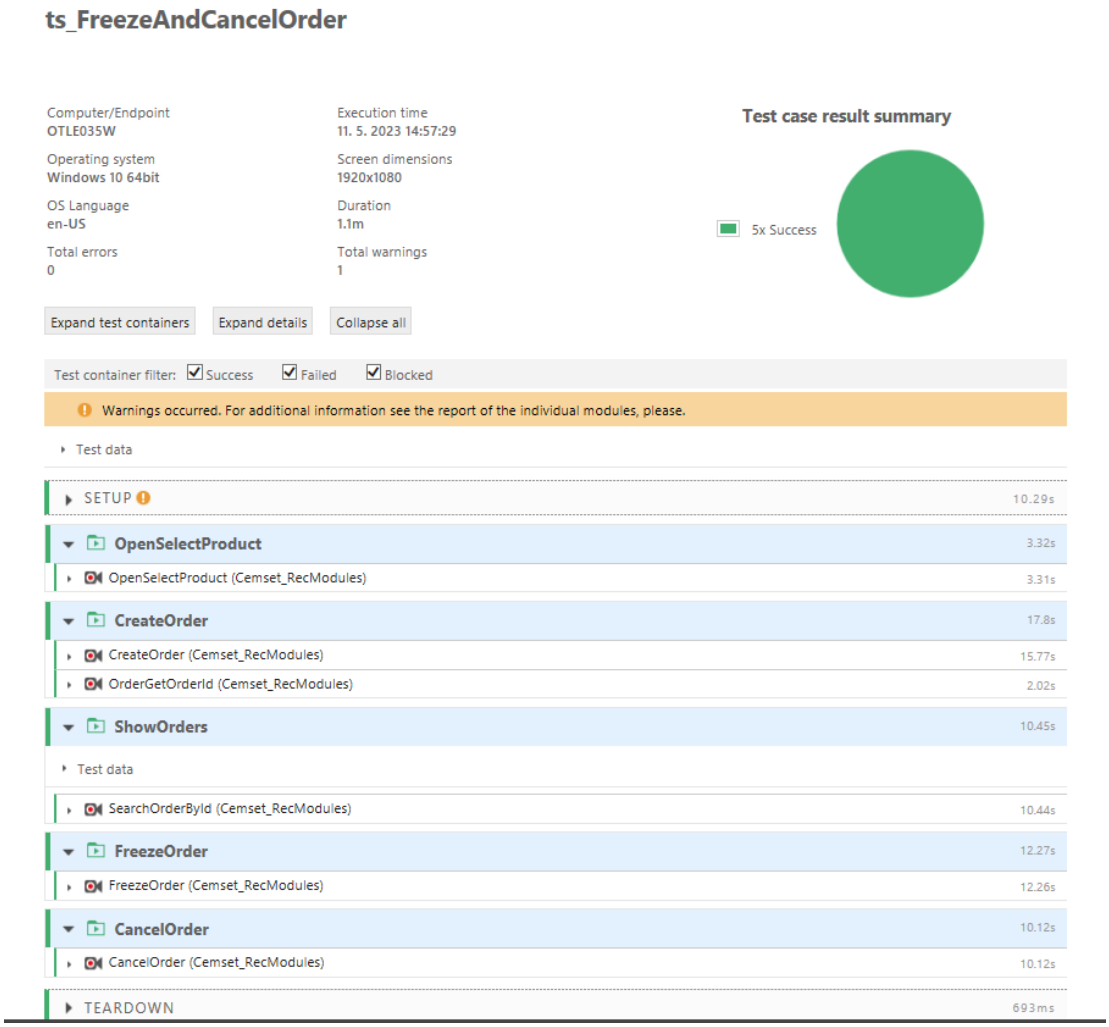
ID Testovacieho prípadu: CancelOrder

- ID Nahrávacieho modulu:
    - CancelOrder

TEARDOWN

- ID Nahrávacieho modulu:
    - CloseBrowser

Report:

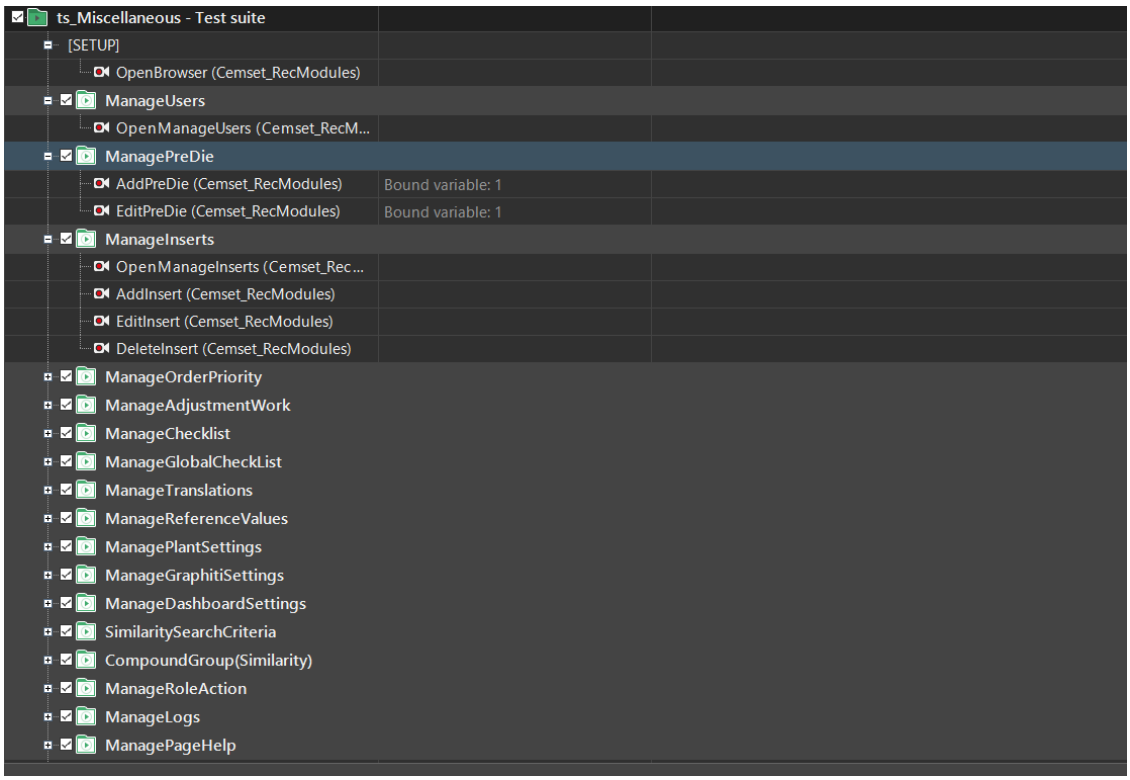


Obrázok 26. Report ts_FreezeAndCancelOrder (zdroj: vlastný)

V danom reporte zo sady automatizovaných testov ts_FreezeAndCancelOrder, sa úspešne vykonali všetky nahrávacie moduly, a tým pádom aj testovacie prípady. Nahrávacích modulov bolo osem a testovacích prípadov bolo päť. Čas vykonávania testovacích prípadov bol dohromady 1.1 minúty.

4.10 ts_Miscellaneous

Sada automatizovaných testov ts_Miscellaneous je zameraná na testovanie položky navigácie *Miscellaneous* a väčšinu jej jednotlivých podpoložiek, ktoré ponúkajú možnosť pridávanie, editácie a mazania určitých prvkov.



Obrázok 27. ts_Miscellaneous (zdroj: vlastný)

4.11 Názorná ukážka testovacieho prípadu ManagePlantSettings:

ID Testovacej sady: ts_Miscellaneous

ID Testovacieho scenára: Miscellaneous

ID Testovacieho prípadu: ManagePlantSettings

Vypracoval: Jakub Švec

Popis scenáru: Overenie základnej Funkcionality Podpoložky navigácie Miscellaneous

Popis prípadu: Zmena hodnoty závodu

Testovacie kroky pre recording modul OpenManagePlantSettings:

1. Kliknutie na položku navigácie Miscellaneous
2. Kliknutie na podpoložku navigácie Manage Plant Settings

Testovacie kroky pre recording modul EditPlantGlobalSettings:

1. Kliknutie na tlačidlo prvej šípky v nastavení závodu
2. Kliknutie na tlačidlo Edit v prvom stĺpci
3. Dvojklik do políčka a prepísanie hodnoty na test
4. Kliknutie na tlačidlo Save Changes
5. Kliknutie na tlačidlo Yes
6. Kliknutie na tlačidlo OK

Preconditions:

Nachádzať sa na domovskej stránke CEMSET

Testovacie dáta:

test

Očakávaný výsledok:

1 Rozbalenie nastavení závodu

2 Zobrazenie okna s možnosťou úpravy hodnoty závodu

3 Textové pole bude obsahovať text *test*

4 Zobrazenie okna s potvrdením rozhodnutia

5 Zobrazenie okna o informácií s úspešnou zmenou hodnoty závodu

6 Hodnota úspešne zmenená.

4.12 **Štruktúra sady ts_Miscellaneous**

SETUP

- ID Nahrávacieho modulu:
    - OpenSelectProduct

ID Testovacieho prípadu: ManageUsers

- ID Nahrávacích modulov:
    - OpenManageUsers

ID Testovacieho prípadu: ManagePreDie

- ID Nahrávacích modulov:
    - AddPreDie
    - EditPreDie

ID Testovacieho prípadu: ManageInserts

- ID Nahrávacieho modulu:
    - OpenManageInserts
    - AddInsert
    - EditInsert
    - DeleteInsert

ID Testovacieho prípadu: ManageOrderPriority

- ID Nahrávacieho modulu:
    - OpenManageOrderPriority
    - AddOrderPriority
    - EditOrderPriority
    - DeleteOrderPriority

ID Testovacieho prípadu: ManageAdjustmentWork

- ID Nahrávacieho modulu:
    - OpenManageAdjustmentWork
    - AddAdjustmentType
    - EditAdjustmentType
    - DeleteAdjustmentType

ID Testovacieho prípadu: ManageChecklist

- ID Nahrávacieho modulu:
    - OpenManageChecklist
    - AddChecklists
    - EditChecklist
    - DeleteChecklist

ID Testovacieho prípadu: ManageGlobalChecklist

- ID Nahrávacieho modulu:
    - OpenManageGlobalChecklist
    - AddGlobalChecklist
    - EditGlobalChecklist
    - DeleteGlobalChecklist

ID Testovacieho prípadu: ManageTranslations

- ID Nahrávacieho modulu:
    - OpenManageTranslations
    - EditTranslations

ID Testovacieho prípadu: ManageReferenceValues

- ID Nahrávacieho modulu:
    - OpenManageReferenceValues
    - ChangePlantDescription

ID Testovacieho prípadu: ManagePlantSettings

- ID Nahrávacieho modulu:
    - OpenManagePlantSettings
    - EditPlantSettings

ID Testovacieho prípadu: ManageGraphitiSettings

- ID Nahrávacieho modulu:
    - OpenManageGraphitiSettings
    - EditPlantSettings

ID Testovacieho prípadu: ManageDashboardSettings

- ID Nahrávacieho modulu:
    - OpenDashBoardSettings
    - EditDashboardSettings

ID Testovacieho prípadu: SimilaritySearchCriteria

- ID Nahrávacieho modulu:
    - OpenSimilaritySearchCriteria
    - EditPlantLowerRange

ID Testovacieho prípadu: CompoundGroup(Similarity)

- ID Nahrávacieho modulu:
    - OpenCompoundGroupSimilarity
    - CreateGroup
    - DeleteGroup

ID Testovacieho prípadu: ManageRoleAction

- ID Nahrávacieho modulu:
    - OpenManageRoleAction
    - EditRole

ID Testovacieho prípadu: ManageLogs

- ID Nahrávacieho modulu:
    - OpenManageLogs
    - EditErrorLogs

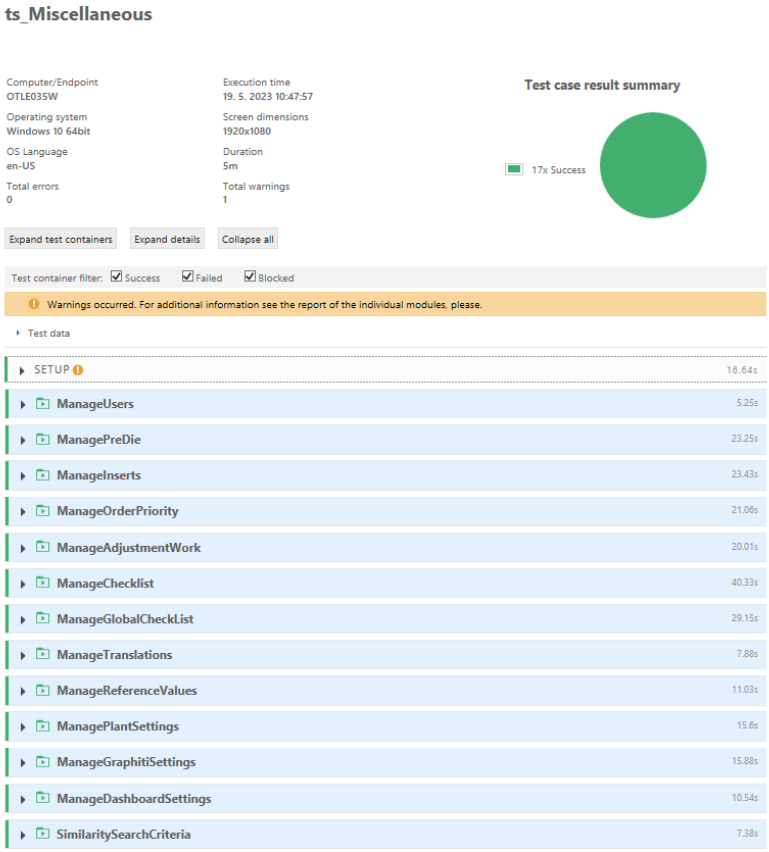
ID Testovacieho prípadu: ManagePageHelp

- ID Nahrávacieho modulu:
    - OpenManagePageHelp
    - ChangePageHelpUrl

TEARDOWN

- ID Nahrávacieho modulu:
    - CloseBrowser

Report:



Obrázok 28. ts_Miscellaneous (zdroj: vlastný)

Report testovacej sady ts_Miscellaneous dokazuje, že sa úspešne podarilo vykonať všetkých sedemnásť testovacích prípadov a štyridsaťšesť nahrávacích modulov, tvorili danú sadu. Čas vykonávania testovacích prípadov bol 2.1 minúty.

5 INTEGRÁCIA AUTOMATIZOVANÝCH TESTOV DO ŠTANDARDNÉHO VÝVOJOVÉHO PROCESU

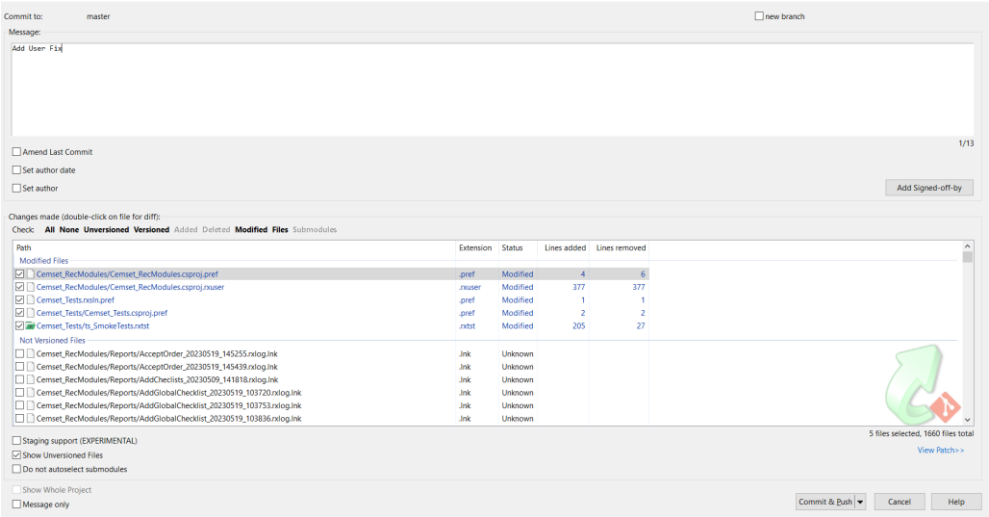
Integrácia automatizovaných testov do štandardného vývojového procesu je neoddeliteľná súčasť každého moderného prístupu k testovaniu softvéru. V prípade spoločnosť Continental Barum s.r.o. sa tento prístup nevyužíval v minulosti toľko ako dnes, čo malo za následok menšiu prehľadnosť efektivity vývoja softvéru. Práve táto metodika je učená na zlepšenie všeobecnej myšlienky automatizovaného testovania, ktorá ma za cieľ zvýšiť efektívnosť vývoja.

5.1 Nástroje využité pre vývojový proces integrácie automatizovaných testov v spoločnosti Continental Barum s.r.o

Pre vývojový proces integrácie automatizovaných testov v spoločnosti Continental Barum s.r.o. bola potrebná konfigurácia nasledovných nástrojov:

1. **GitHub**

   GitHub bol využívaný ako platforma pre verzovanie jednotlivých testovacích prípadov. Zmeny boli commitnuté pomocou nástroja TortoiseGit. TortoiseGit je grafické používateľské rozhranie pre Git pre operačný systém windows.

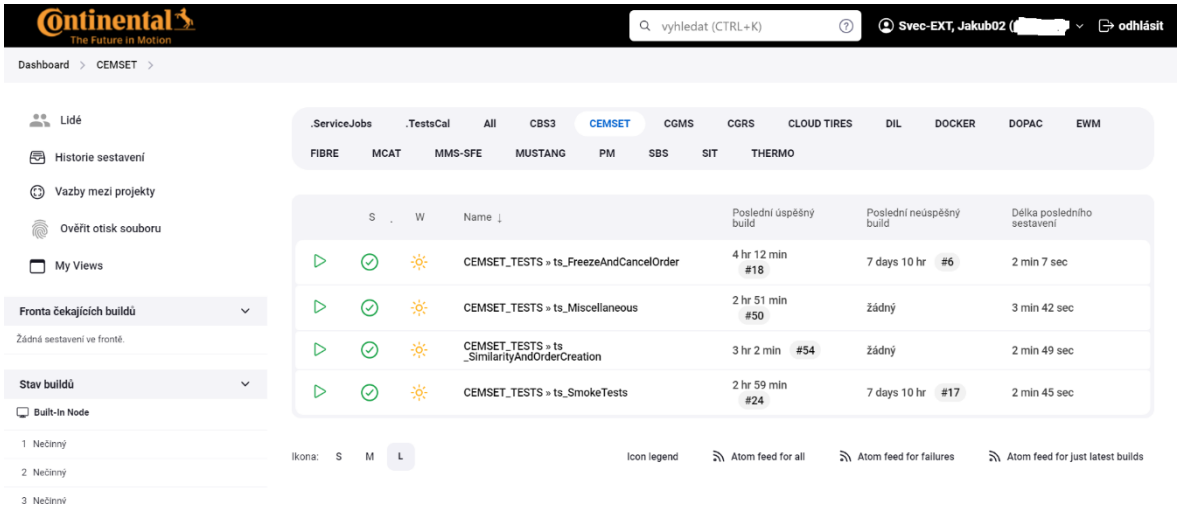


Obrázok 29. Tortoise Git rozhranie (zdroj: vlastný)

2. **Jenkins**

   Vývojový proces integrácie automatizovaných testov je realizovaný pomocou nástroja Jenkins. Jenkins je open source server určený na automatizáciu kontinuálnej

integrácie alebo kontinuálneho dodávania a nasadzovania napísaný v jazyku Java. Jenkins nám umožňuje zobraziť všeobecný prehľad úspešnosti testov, dĺžku zostavenia, históriu zostavenia podobne... [17]



Obrázok 30. Rozhranie Jenkins (zdroj: vlastný)

5.2 Komunikácia medzi GitHub a Jenkins

Pre efektívne využívanie nástroja Jenkins je potrebné ho spojiť a správne nastaviť so zdrojovým repozitárom obsahujúcim dané testy.

Pre komunikáciu medzi GitHub a Jenkins je potreba niekoľko základných nastavení:

1. Je potrebné mať nastavený Git ako zdrojový repozitár s obsahom zdrojového kódu testovacích prípadov.
2. Inštalácia a konfigurácia Jenkins serveru. Tu je potrebné nainštalovať Git plugin pre interakciu serveru s vytvoreným zdrojovým repozitárom.
3. Vytvorenie komunikačného kanálu (Jenkins Pipeline Job).
4. Konfigurácia komunikačného kanálu – vytvorenie skriptu ktorý definuje zostavenie projektu, pustenie testov a vykonanie nasadenia.
5. Nastavenie Git Webhook, ktorý upovedomí Jenkins pomocou HTTP POST požiadavky vždy keď sa vykoná commit, čo bude mať za následok automatické pustenie úlohy Jenkins.

Je dôležité povedať, že daná komunikácia musí byť vždy naladená na základe požiadaviek a spôsobu vývoja danej firmy, takže sa môže niekedy jej konfigurácia líšiť. [18][19][20]

5.3 Postup integrácie automatizovaných testov

Postup integrácie automatizovaných testov prebiehal pomocou štandardného procesu v rámci firmy Continental Barum s.r.o. Riešenie je skonštruované pomocou nástroja Jenkins. Po vytvorení testovacieho prípadu a jeho následnom commite do Github repozitára, sa spustí sada úloh, ktoré boli nadefinované počas nastavovania a konfigurácie úlohy Jenkins (Jenkins job). Jenkins job pipeline musí byť vždy nastavená podľa špecifických požiadaviek . Jenkins job je v tomto prípade nastavený na automatické spúšťanie dva-krát do týždňa s možnosťou manuálneho púšťania na stránke v prípade potreby.

Jenkins job v prípade integrácie testov, ktoré sú predmetom tejto bakalárskej reprezentuje Pipeline, ktorá vyzerá nasledujúco:

Obrázok 31. Jenkins job pipeline (zdroj: vlastný)

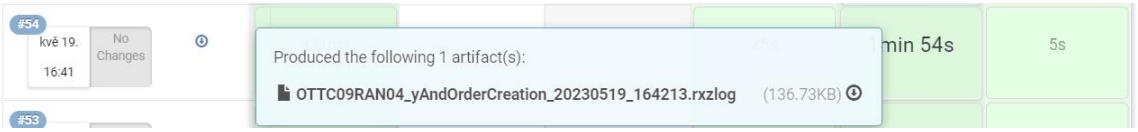
Pipeline sa skladá z nasledujúcich častí:

1. Configuration: Táto časť pozostáva z načítania parametrov deklarovaných pri vytváraní Jenkins úlohy. Jedná sa napríklad o lokáciu repozitára so zdrojovým kódom, nastavenia spúšťania testov, prípadne nastavenia prostredia.
2. Checkout Code: V tejto fáze Jenkins vytiahne najnovšiu verziu commitnutého zdrojového kódu ranorexu z GitHub repozitára, s ktorým je prepojený.
3. Build Test Solution: V tomto kroku sa zostaví a skompiluje kód ranorex solution.
4. Prepare VM for testing: Tento krok zahŕňa nastavenie virtuálneho počítača, na ktorom sa budú testy spúšťať.
5. Testing: Spustenie daných testov v prostredí ktoré čo najlepšie simuluje produkčné prostredie.
6. Publish Results: Zobrazenie výsledkov testov v reporte poprípade poslanie notifikácie. [20][21]

Po dokončení priebehu úloh definovaných v Jenkins jobe je možné zobraziť a prebrať dané výsledky integrácie a zanalyzovať ich.

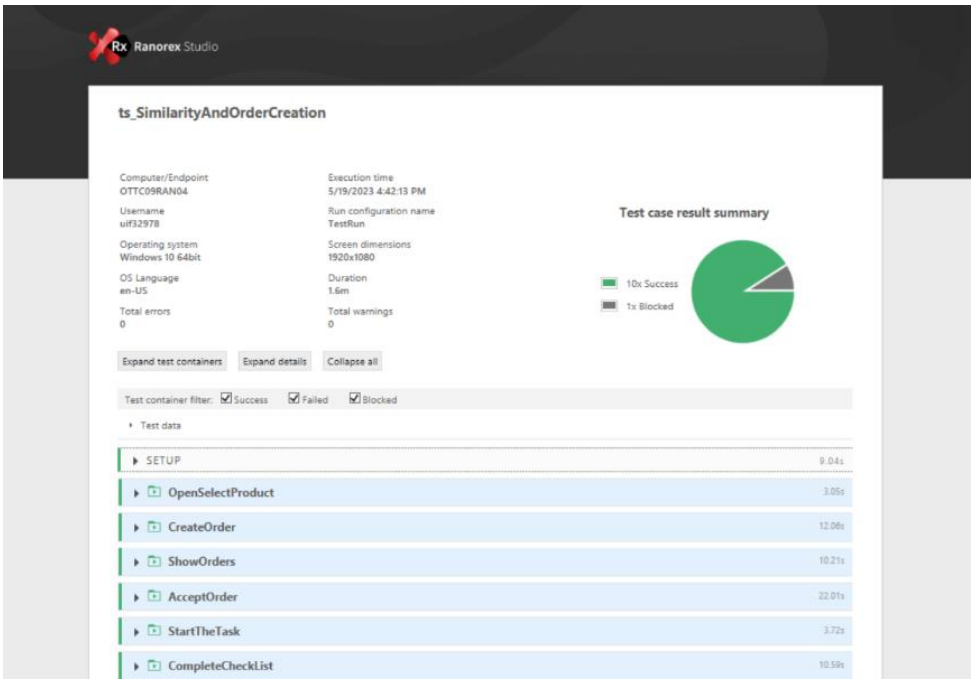
5.4 Výsledky a hodnotenie integrácie automatizovaných testov do štandardného vývojového procesu

Výsledky integrácie automatizovaných testov možno považovať za kladné. Všetky 4 testovacie sady prešli úspešne, čo potvrdzujú aj logy, ktoré budú pridané v prílohe a taktiež je tu názorná ukážka jednotlivých logov. Po skončení priebehu úloh v Jenkins je možné stiahnúť daný log, nasledovne:



Obrázok 32. Jenkins log (zdroj: vlastný)

Log je v rovnakom formáte ako zo samotného ranorexu a teda s príponou rxzlog. Po otvorení daného logu je možno vidieť, že testy úspešne prešli integráciou pomocou nástroja Jenkins.



Obrázok 33. Jenkins report ts_SimilarityAndOrderCreation (zdroj: vlastný)

Ďaľšia názorná ukážka tabuľky, ktorá zobrazuje prehľad úspešnosti daných testov. Na zák lade výsledkov integrácie testov je možno skonštatovať, že zo 40 testov úspešne prešlo 39. Žiadny test nezlyhal. Jeden test bol vynechaný a to z dôvodu nenaplnenia podmienky

Job ↓	Success #	%	Failed #	%	Skipped #	%	Total #
CEMSET_TESTS » ts_FreezeAndCancelOrder	5	100%	0	0%	0	0%	5
CEMSET_TESTS » ts_Miscellaneous	17	100%	0	0%	0	0%	17
CEMSET_TESTS » ts_SimilarityAndOrderCreation	10	91%	0	0%	1	9%	11
CEMSET_TESTS » ts_SmokeTests	7	100%	0	0%	0	0%	7
Total	**39**	**98%**	**0**	**0%**	**1**	**2%**	**40**

Obrázok 34. Tabuľka výsledkov integrácie testov (zdroj: vlastný)

Pravidelná integrácia testov prináša výhody v podobe rýchlejšieho identifikovania chýb, tým pádom sa urýchluje vývoj, znižuje riziko tvorby závažných problémov vzhľadom na fakt, že sa vykonáva často a pravideľne po malých zmenách v kóde a neposlednom rade jej využitím je možné zlepšovať kvalitu celkového softvérového produktu. Čo znamená, že jej správna implementácia má pozitívny vplyv a dopad na vývoj softvérového produktu.

6 VYHODONOTENIE POKRYTIA APLIKÁCIE AUTOMATIZOVANÝMI TESTAMI

Nasledujúca kapitola bude zameraná na vyhodnotenie pokrytia aplikácie CEMSET automatizovanými testami. Vyhodnocovanie pokrytia aplikácie automatizovanými testami je dôležité práve kvôli správnemu identifikovaniu slabín, ktoré potencionálne môžu nastávať na miestach aplikácie, kde nie je vysoká miera pokrytia. Taktiež sa vyhodnotenie pokrytia zameriava na splnenie daných požiadaviek, ktoré boli stanovené v rámci analýzy požiadaviek s klientom.

6.1 Metódy merania pokrytia testami

Existuje viacero metód merania pokrytia aplikácie automatizovanými testami, ktoré sú zamerané na to, aké časti kódu alebo funkcionality aplikácie sú otestované. Medzi tieto metódy patrí napríklad:

1. Pokrytie príkazov
2. Pokrytie vetvení
3. Pokrytie podmienok
4. Pokrytie ciest
5. Pokrytie požiadaviek

V tomto prípade je dôležité spomenúť, že metóda merania pokrytia aplikácie CEMSET v rámci tejto bakalárskej práce je metóda pokrytia požiadaviek.

Metóda pokrytia požiadaviek spočívajú vo vytypovaní jednotlivých požiadaviek na softvér. Následne sa na základe jednotlivých požiadaviek vytvoria testovacie prípady. Tieto testovacie prípady sa vykonajú a potom sa priradia k jednotlivým požiadavkám. Na základe priradenia testovacích prípadov k vytypovaným požiadavkám je možnosť vyhodnocovania pokrytia požiadaviek. [22][23]

Na detailnejšie riešenia sa čaká v rámci implementácie nástroja Teamscale do vývojového procesu v spoločnosti Continental Barum s.r.o. V momente kedy bude Teamscale zaradený do štandardného vývojového procesu, bude danom nástroji možno vidieť informácie o tom aká časť zdrojového kódu je pokrytá.

6.2 Analýza pokrytia aplikácie automatizovanými testami

Analýza pokrytia aplikácie automatizovanými testami bola realizovaná v rámci vytypovania určitých požiadaviek, na základe ktorých boli vytvorené testovacie prípady, ktoré boli automatizované.

Jednotlivé požiadavky sú:

1. SmokeTests
    a. Otvorenie a validácia všetkých položiek a podpoložiek navigácie.
2. Similarity&OrderCreation
    a. Filtrovanie a vyhľadávanie komponentov v rámci podpoložky navigácie Select Product
    b. Životný cyklus objednávky (Vytvorenie objednávky, Zmrazenie objednávky, Zrušenie objednávky, Akceptovanie objednávky, Dokončenie finálneho kroku)
3. Miscellaneous
    a. Pre-Die (Vytvorenie, Úprava)
    b. ManageInserts (Vytvorenie, Úprava, Zmazanie)
    c. ManageOrderPriority (Vytvorenie, Úprava, Zmazanie)
    d. ManageAdjustmentWork (Vytvorenie, Úprava, Zmazanie)
    e. ManageChecklist (Vytvorenie, Úprava, Zmazanie)
    f. ManageGlobalChecklist (Vytvorenie, Úprava, Zmazanie)
    g. ManageTranslations (Úprava)
    h. ManageReferenceValues (Úprava)
    i. ManagePlantSettings (Úprava)
    j. ManageGraphitiSettings (Úprava)
    k. ManageDashboardSettings (Úprava)
    l. SimilaritySearchCriteria (Úprava)
    m. CompoundGroup (Vytvorenie, Zmazanie)
    n. ManageRoleAction (Úprava)
    o. ManageLogs (Úprava)

6.3 Vyhodnotenie výsledkov pokrytia aplikácie automatizovanými testami

Vyhodnotenie výsledkov pokrytia aplikácie automatizovanými testami je znázornené pomocou testovacích prípadov, ktoré boli priradené ku jednotlivým požiadavkám.

Pre požiadavky v rámci prvého bodu SmokeTests sú priradené nasledujúce testovacie prípady:

1. Similarity&OrderCreation
2. ManageDie
3. PanelView
4. Miscellaneous
5. NoteBook
6. PlantKPI
7. CentralDashboard

Pre požiadavky v rámci druhého bodu Similarity&OrderCreation sú priradené nasledujúce testovacie prípady:

1. OpenSelectProduct
2. CreateOrder
3. ShowOrders
4. AcceptOrder
5. StartTheTask
6. CompleteCheckList
7. CheckForManualDimensions
8. CompleteOrder
9. StartTrials
10. CompleteTrials
11. FinalStep
12. FreezeOrder
13. CancelOrder

Pre požiadavky v rámci Tretieho bodu Miscellaneous sú priradené nasledujúce testovacie prípady:

1. ManagePageHelp

2. ManageLogs
3. ManageRoleAction
4. CompoundGroup(Similarity)
5. SimilaritySearchCriteria
6. ManageDashboardSettings
7. ManageGraphitiSettings
8. ManagePlantSettings
9. ManageReferenceValues
10. ManageTranslations
11. ManageGlobalChecklist
12. ManageChecklist
13. ManageAdjustmentWork
14. ManageOrderPriority
15. ManageInserts
16. ManagePreDie
17. ManageUsers

Vzhľadom na fakt, že ku všetkým vytypovaným požiadavkám boli priradené testovacie prípady, ktoré ich pokrývajú, je možné vyhodnotiť pokrytie aplikácie v rámci daných požiadaviek ako stopercentné z toho hľadiska, že sme pokryli všetky požiadavky, ktoré sme si určili.

ZÁVER

Cieľom tejto bakalárskej práce bolo popísať základné princípy testovania softvéru, tak isto popísať aj jednotlivé úrovne testovania softvéru spolu s technikami testovania, ktoré sa využívajú pri testovaní. Práca tiež porovnáva manuálne a automatizované testovanie, pričom uvádza výhody a nevýhody oboch prístupov. Z hľadiska automatizovaného testovania bola zdôraznená dôležitosť správnej voľby nástrojov a účinného plánovania testov.

Snahou bolo opísať takisto jeden z mnoho výrobných informačných systémov v spoločnosti Continental Barum s.r.o. vrátane jeho architektúry a funkcionality.

V neposlednom rade bolo potrebné vytvoriť sadu automatizovaných testov v nástroji Ranorex. Táto sada bola neskôr integrovaná do štandardného vývojového procesu spoločnosti Continental Barum s.r.o. pomocou nástroja Jenkins.

Nakoniec, práca sa zaoberala vyhodnotením pokrytia aplikácie automatizovanými testami.

ZOZNAM POUŽITEJ LITERATÚRY

[1] GRAHAM, Dorothy, Rex BLACK a Erik VAN VEENENDAAL. Foundations of software testing: ISTQB certification. Fourth edition. Andover, Hampshire: Cengage, [2020]. ISBN 978-1-4737-6479-8.

[2] PAGE, Alan, Ken JOHNSTON a Bj ROLLISON. Jak testuje software Microsoft. Brno: Computer Press, 2009. ISBN 978-80-251-2869-5.

[3] BUREŠ, Miroslav, Miroslav RENDA, Michal DOLEŽEL, Peter SVOBODA, Zdeněk GRÖSSL, Martin KOMÁREK, Ondřej MACEK a Radoslav MLYNÁŘ. Efektivní testování softwaru: klíčové otázky pro efektivitu testovacího procesu. Praha: Grada, 2016. Profesionál. ISBN 978-80-247-5594-6.

[4] RANOREX: STUDIO ADVANCED USERGUIDE. 2022. Austin: Idera, 2022. Dostupné také z: https://www.ranorex.com/rx-media/rx-user-guide/pdf-v10.2/Ranorex-Studio-Advanced.pdf

[5] RANOREX: STUDIO EXPERT USERGUIDE. 2022. Austin: Idera, 2022. Dostupné také z: https://www.ranorex.com/rx-media/rx-user-guide/pdf-v10.2/Ranorex-Studio-Expert.pdf

[6] GARG, Kashish. White Box Testing [online]. 2022. Dostupné také z: https://www.scaler.com/topics/white-box-testing/

[7] Data Flow Testing. Geeksforgeeks [online]. 24.10.2019. Dostupné z: https://www.geeksforgeeks.org/data-flow-testing/

[8] Gray Box Testing | Software Testing. Geeksforgeeks [online]. 2022, 19.07.2022. Dostupné z: https://www.geeksforgeeks.org/gray-box-testing-software-testing/

[9] SAVARAM, Ravindra. Manual Testing Tutorial for Beginners. Mindmajix [online]. 2023, 23.05.2023. Dostupné z: https://mindmajix.com/manual-testing-tutorial

[10] HAMILTON, Thomas. Manual Testing Tutorial: What is, Types, Concepts. Guru99 [online]. 2023, 8.4.2023. Dostupné z: https://www.guru99.com/manual-testing.html

[11] HAMILTON, Thomas. What is Automation Testing? Automation Testing Tutorial. Intellipaat [online]. 2023, 19.04.2023. Dostupné z: https://intellipaat.com/blog/what-is-automation-testing/?US

[12] DA SILVA, Michael. Pros and Cons of Automated Testing. Uilicious [online]. 2022, 31.01.2022. Dostupné z: https://uilicious.com/blog/pros-cons-automated-testing/

[13] ASP.NET overview. Learn.microsoft [online]. 2022, 30/09/2022. Dostupné z: https://learn.microsoft.com/en-us/aspnet/overview

[14] ASP.NET MVC Overview. Learn.microsoft [online]. 2020, 04/10/2020. Dostupné z: https://learn.microsoft.com/en-us/aspnet/mvc/overview/older-versions-1/overview/asp-net-mvc-overview

[15] IIS Web Server Overview. Learn.microsoft [online]. 2022, 23.08.2022. Dostupné z: https://learn.microsoft.com/en-us/iis/get-started/introduction-to-iis/iis-web-server-overview

[16] HUGHES, Adam a Craig STEDMAN. Microsoft SQL Server. Techtarget [online]. 2019. Dostupné z: https://www.techtarget.com/searchdatamanagement/definition/SQL-Server

[17] RIGLIAN, Adam. Jenkins. techtarget. [online]. 2019. Dostupné z: https://www.techtarget.com/searchsoftwarequality/definition/Jenkins

[18] SALTON, Guy. How to Integrate Your GitHub Repository to Your Jenkins Project. Blazemeter [online]. 2022, 21.07. 2022. Dostupné z: https://www.blazemeter.com/blog/how-to-integrate-your-github-repository-to-your-jenkins-project

[19] Git | Jenkins plugin. Jenkins Plugins [online]. Dostupné z: https://plugins.jenkins.io/git/

[20] Jenkins workflow | Full tutorial from beginner to advance in 2023. Naiveskill [online]. 2023. Dostupné z: https://naiveskill.com/jenkins-workflow/

[21] Pipeline. Jenkins [online]. Dostupné z: https://www.jenkins.io/doc/book/pipeline/

[22] ŻURAWIECKI, Marcin. Why requirements test coverage is key to success?. Deviniti [online]. 2018, 15.10.201[8. Dostupné z: https://deviniti.com/blog/application-lifecycle-management/why-requirements-test-coverage-is-key-to-success/

[23] CONDE, Bruno. Requirements Coverage Analysis. Getxray [online]. 2017, 16.11.2017. Dostupné z: https://docs.getxray.app/display/XRAY30/Requireme[nts +Coverage +Analysis #RequirementsCoverageAnalysis-RequirementCoverageStatus

ZOZNAM POUŽITÝCH SYMBOLOV A SKRATIEK

HLD	High-Level-Design
LLD	Low-Level-Desing
CFG	Control Flow Graph
OAT	Orthogonal Array Testing
QTP	QuickTestProfessional
CEMSET	Centralized Management System for Extrusion Tools
API	Application Programming Interface
MVC	Model-View-Controller
URL	Uniform Resource Locator
IIS	Internet Information Services
ID	identity documentation

ZOZNAM OBRÁZKOV

ZOZNAM TABULIEK

ZOZNAM PRÍLOH

Príloha P1: CD

PRÍLOHA P I:

Príloha obsahuje Ranorex solution a logy z Jenkinsu a Ranorexu.