Ohjelmiston hyväksymistestausvaiheen automatisointi

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Tietojenkäsittelyn koulutus
kevät, 2023

Marianne Nikkilä

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Tietojenkäsittelyn koulutus Tiivistelmä

Tekijä Marianne Nikkilä Vuosi 2023

Työn nimi Ohjelmiston hyväksymistestausvaiheen automatisointi

Ohjaaja Lasse Seppänen

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli tuottaa yksikkötestejä ohjelmiston hyväksymistestausvaiheen automatisointiin. Hyväksymistestausvaiheen automatisoinnilla pyrittiin vähentämään tarvittavaa aikaa ja henkilöresurssimäärää, joka testaamiseen tarvittiin. Työ toteutettiin hyödyntäen Robot Frameworkia. Muina työkaluina toimi Python, Docker, Visual Studio Code sekä MS Azure. Opinnäytetyön toimeksiantajana oli Triplan Oy. Projektissa siis pääsin kokemaan käytäntöjä ja menetelmiä, kuinka toimitaan tehdessä yritykselle tuotettava projekti.

Opinnäytetyöhön tietoa on kertynyt mm. netistä, kirjoista sekä omasta kokemuksesta. Lisätietoa testaukseen ja sen pohjaan sain, koska testaus perustui vanhojen testien refaktorointiin. Joten jonkin verran testien sisältöä saatiin jo luoduista testeistä, mutta suurin osa kuitenkin luotiin alusta loppuun. Näiden lähteiden ja tietojen perusteella on yritetty tavoitella mahdollisimman hyvin projektin tavoitteita. Kuitenkin kuten opinnäytetyöstäkin selviää, sen aikana opin paljon uutta ja pystyin hyödyntämään uusia oppeja koko ajan paremmin projektin edetessä.

Opinnäytetyössä havaittiin, kuinka tärkeää regressiotestaus on ja miten paljon testauksen automatisoinnilla voidaan vähentää tarvittavia resursseja. Suurimpana hyötynä opinnäytetyössä kuitenkin jäi käteen oma oppiminen projektin aikana ja kokemus projektin tuottamisesta yritykselle.

Avainsanat Robot Framework, Regressiotestaus, Yksikkötestit, Automaatiotestaus

Sivut 33 sivua ja liitteitä 1 sivu

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Degree Programme in Business Information Technology Abstract

Author Marianne Nikkilä Year 2023

Subject Automation of the software acceptance testing phase

Supervisor Lasse Seppänen

ABSTRACT

The aim of the thesis was to produce unit tests for the automation of the acceptance testing phase of the software. By automating the acceptance testing phase, goal was to reduce the necessary time and the amount of human resources needed for testing. The work was carried out using Robot Framework. Other tools were Python, Docker, Visual Studio Code and MS Azure. The client of the thesis was Triplan Oy. In the project, I got to experience the practices and methods of how to act when making a project to be produced for the company.

Information for the thesis has been accumulated e.g., from the internet, from books and from my own experience. I got more information about testing and its basis, because the testing was based on the refactoring of old tests. Some parts of the tests came from already created tests, but most of them were created from start to finish. Based on these sources and information, we have tried to achieve the goals of the project as best as possible. However, as can be seen from thesis, during that time I learned a lot of new things and was able to use the new lessons better as the project progressed.

In thesis, it was discovered how important regression testing is and how much the necessary resources can be reduced by automating testing. However, biggest benefit in the thesis was my own learning during the project and the experience of producing the project for the company.

Keywords Robot Framework, Regression testing, Unit tests, Automation testing

Pages 33 pages and appendices 1 page

Sanasto

Xpath	Polkulauseke, jota käytetään valitsemaan elementtejä nettisivulta.
Elementti	Sivuston esimerkiksi nappi tai tekstikenttä.
Keyword	Osuus, joka voidaan ajaa RF testissä käyttämällä avainsanaa.
Testcase	Yksittäinen testitapaus.
Testsuite	Testikokonaisuus, joka sisältää useita testejä.

Sisällys

Kuvat, ohjelmakoodit ja taulukot

Liitteet

1 Johdanto

Opinnäytetyössä syvennytään regressiotestaukseen. Aiheena on ohjelmiston hyväksymistestausvaiheen automatisointi. Suoritin harjoittelun Triplan Oy:llä, josta he tarjosivat opinnäytetyön aihetta. Opinnäytetyössä pyritään tuottamaan hyötyä työn tilaajalle sekä vahvistamaan omaa osaamista. Kun sain aihe-ehdotuksen Triplanilta, olin alusta asti kiinnostunut ja hyvin myönteinen aihetta kohtaan.

Kokemusta regressiotestauksesta olen saanut jonkin verran koulutuksen aikana, mutta on kiinnostavaa päästä syventymään aiheeseen paremmin. Työssä päätyökaluina toimii Robot Framework, Python, VScode, Docker sekä Azure. Robot Framework on yleinen avoimen lähdekoodin automaatiokehitys. Robot Frameworkia voidaan hyödyntää testiautomaatiossa ja robottiprosessiautomaatiossa (RPA). Python on oliosuuntautunut korkean tason ohjelmointikieli dynaamisella semantiikalla. Sovelluskehityksessä Python tarjoaa mahdollisuuden nopeaan ohjelmointikehitykseen, sen korkeatasoisten sisäänrakennettujen tietorakenteiden yhdistämisen dynaamiseen kirjoittamiseen sekä dynaamiseen sidontaan. Koodieditori, joka tunnetaan nimellä Visual Studio Code on määritelty uudelleen ja optimoitu nykyaikaisten verkko- ja pilvisovellusten rakentamiseen ja virheenkorjaukseen. Docker on vuonna 2013 esitellyt konttien alan standardin. Kontit ovat standardoitu ohjelmistoyksikkö, jonka avulla kehittäjät voivat eristää sovelluksensa ympäristöstään ja ratkaista ”se toimii vain koneellani” päänsäryn. Azure on pilvialusta, joka sisältää lukemattoman määrän tuotteita ja pilvipalveluita. Se on suunniteltu auttamaan tuomaan uusia ratkaisuja eloon. Työssä sitä hyödynnetään versionhallintaan. Teoriaosuudessa syvennytään paremmin työkaluihin ja niiden mahdollisuuksiin.

Opinnäytetyön tutkimuskysymyksiä ovat:

- Miten testikokonaisuuksien tuomia tietosisältöjä hallitaan (testiympäristön aineiston puhdistus)?
- Miten testit uusitaan virhetilanteessa?
- Kuinka ohjelmiston kehittyessä uusia tarvittavia testikokonaisuuksia hallitaan ja lisätään?
- Mitä testien setup ja teardown osioiden sisältöön tarvitaan?

2 Regressiotestaus/hyväksymistestaus

Regressiotestauksessa on tarkoituksena testata jo testattua järjestelmää. Testaus on tarkoitus suorittaa aina, kun tuotteeseen tulee uusia ominaisuuksia. Tavoitteena testata, ettei uudet ominaisuudet ole rikkoneet mitään jo toimivaa. Regressiotestaus on myös mahdollista automatisoida, jolloin luodaan testausautomaatio-ohjelmistoja. Testausautomaatio-ohjelmistot ovat hyödyllisiä tilanteessa, jolloin samoja toiminnallisuuksia tarvitsee testata useaan kertaan.

Testausautomaatiosta ei kuitenkaan ole hyötyä asiakkaalle, joka on ottamassa ohjelmistoa käyttöön. Asiakkaan toteuttamalla testillä yleensä vain todennetaan ohjelmiston toiminta ja se suoritetaan manuaalisesti.

Regressiotestauksessa on monenlaisia asioita, joita tulee ottaa huomioon mm. savutestaus, scope, systeemitestaus, testauksen hyväksymiskriteerit ja testauksen kattavuus. Savutestaus tarkoittaa kohtaa ennen hyväksymistestauksen aloittamista, jossa kartoitetaan tuotteen tärkeimmät toiminnallisuudet. Scope tarkoittaa laajuutta projektinhallinnassa eli määritetään mitä kuuluu projektin piiriin ja mitä ei. Systeemitestaus tulee integraatiotestauksen jälkeen. Systeemitestauksessa arvioidaan koko järjestelmän toimivuutta. Testauksen hyväksymiskriteereillä määritetään mitkä toiminnallisuudet testin tarvitsee läpäistä, jotta ohjelmistoa voi sanoa toimivaksi. Testauksen kattavuudella varmistetaan, että testitapaukset kattavat kaikki olennaiset liiketoimintaprosessit. Luku 2 perustuu lähteisiin (Hoogenraad, 2019) (VALA, 2023).

2.1 Regressiotestaus osana ohjelmistokehitystä

Regressiotestaus on isossa osassa ohjelmistokehityksessä. Regressiotestauksella pystytään havaitsemaan mahdolliset bugit tuotteessa varhaisessa vaiheessa, jolloin bugien korjaaminen ei vaadi suuria resursseja. Pahimmassa tilanteessa bugi huomataan vasta kun tuote on viety jo asiakkaalle asti, jolloin kustannukset ovat jo erittäin suuret.
Regressiotestaus on osa ketterää ohjelmistokehitystä. Ketteriä ohjelmiston kehitysmetodeja pyritään käyttämään yhdessä toimittajan ja asiakkaan kanssa itseohjautuvien moniosaajatiimien avulla. Ohjelmistokeskeisyys, nopea muutos, reagointi ja kommunikaatioon perustuvia kehitysmalleja, jossa yritetään tehdä iteraatioita ohjelmistosta nopealla syklillä kuuluvat ketteriin

kehitysmetodeihin. Scrum, DevOps ja SAFe ovat esimerkkejä ketteristä menetelmistä ohjelmistokehitykseen.

2.1.1 Parantunut asiakastyytyväisyys

Parantunut asiakastyytyväisyys on helpoiten selitettävissä oleva hyöty. Regressiotestaus mahdollistaa bugien paikantamisen varhaisessa vaiheessa, jolloin se saadaan korjattua ennen asiakkaalle toimitusta. Varhaisessa vaiheessa havaitulla bugilla säästetään myös sen korjaukseen tarvittavia resursseja.

2.1.2 Kasvanut luottamus tuotteen laatuun

Kehitystiimeille erityisen tärkeää on luottamus softan kehittämiseen. Eli vaikka softaan tehdään muutoksia, niin ei ole odotettavissa, että mitään suurempaa menee rikki. Mikäli jotakin menee rikki, se huomataan varhaisessa vaiheessa regressiotesteissä. Tarkoituksena kuitenkin saada asiakkaan luottamus tarpeeksi laadukkaalla ja virheettömällä softalla.

2.1.3 Laadun tuominen näkyväksi kehitystiimeille

Kehitystiimeillä tarvitsee olla riittävän laaja kuva softan laadusta. Päätöksiä kuten tuotantoon viemistä pystytään perustelemaan paremmin, mitä laajempi kuva softasta on. Tällä tarkoitetaan, kuinka hyvin testauksella voidaan osoittaa softan ominaisuudet toimiviksi.

2.1.4 Jatkuva ja nopea palautteenanto

Testaus mahdollistaa jatkuvan ja nopean palautteenannon. Kun uusia ominaisuuksia päästään nopeasti testaamaan on myös palautteen saaminen nopeaa. Mitä nopeammin siis virhe/palaute saadaan tiedoksi kehittäjille, saadaan vähennettyä tarvittavia resursseja virheenkorjaukseen.

2.2 Integraatiotestit

Testit perustuvat moduulien välisiin integraatioihin. Ohjelmoijat koodaavat ohjelmiston ohjelmistomoduuleissa ja testien tarkoituksena on testata, kuinka tieto siirtyy näiden moduulien

välillä. Vaikka ohjelmistomoduuleille toteutetaan yksikkötestit, saattaa ohjelmisto tämänkin jälkeen sisältää virheitä. Virheitä voi tulla monista syistä, mutta esimerkiksi jos yhden moduulin ohjelmoi yksi ohjelmoija, joka ei ole huomannut ottaa huomioon muiden moduulien toimintaa, asiakkaan toiveet moduulin kehityksen aikana, rajapinta muutokset, ulkoiset API-liitännät tai riittämätön poikkeuksien käsittely.

Integraatiotestauksessa on erilaisia strategioita mm. Big Bang lähestymistapa ja Inkrementaalinen lähestymistapa. Big Bangin tarkoituksena on testata yhdellä kertaa samanaikaisesti kaikki yksiköt ja muut komponentit integroituna. Tämä on kätevää pienille järjestelmille, mutta isommissa virheen paikantaminen voi olla vaikeaa. Inkrementaalisessa tavassa voidaan toteuttaa testaus pohjalta alas tai ylhäältä alas. Tarkoituksena on lisätä aina moduuli ketjuun, kun aikaisempi on todettu toimivaksi.

3 Hyödynnettävät työkalut

Tässä luvussa käydään läpi opinnäytetyössä hyödynnettävät työkalut ja kuinka niitä tullaan käyttämään opinnäytetyössä. Työkaluihin lukeutuu erilaisia ohjelmointikieliä, virtuaaliympäristö sekä versionhallinta. Eniten kokemusta löytyy ohjelmointi kielistä Robot Framework sekä Python. Versionhallinta itsessään on aika vähällä käytöllä ollut itsellä, mutta toivon kehitystä tulevan sen suhteen.

3.1 Robot Framework

Robot Framework on avoimen lähdekoodin automaatiokehitys ja sitä voidaan käyttää testiautomaatiossa sekä robottiprosessiautomaatiossa (RPA). Robot Frameworkia käyttävät monet alan johtavat yritykset ohjelmistokehityksessään. Jotta Robot Frameworkin kehitys jatkuu, on olemassa voittoa tavoittelematon konsortio, nimeltään Robot Framework Foundation. Alun perin Robot Framework Foundationin on perustaneet yritykset, joilla on yhteinen kiinnostus kehittää ja varmistaa Robot Frameworkin kehityksen nyt ja tulevaisuudessa. Yritysten on mahdollista liittyä Robot Framework Foundationin jäseneksi, jolloin he saavat tietynlaisia etuja. Jäsenyritykset ovat kuitenkin osakseen vastuussa Robot Frameworkin kehityksestä ja tulevaisuudesta. Säätiö keskittyy sponsoroimaan Robot Frameworkin kehitystä keskittyen virheiden korjaamiseen, yhteisön pyyntöjen tarkistamiseen sekä julkaisujen tekemiseen. Säätiö huolehtii myös yleisestä Robot Frameworkin infrastruktuurista, mukaan lukien julkiset web-sivustot, sähköpostilistat ja CI-palvelimet. Lisäksi säätiö on vastuussa mainostamisesta mm. käyttäjätapahtumat. (robotframework.org, n.d.)

Robot Frameworkin käyttö on ilmaista ilman lisenssikuluja. Robot Framework tunnetaan sen avoimuudesta ja laajennus mahdollisuuksista. Ohjelmisto omaa helpon syntaksin, joka käyttää ihmisen luettavia avainsanoja. Se on mahdollista integroida käytännössä minkä tahansa muun työkalun kanssa, jotta saadaan luotua tehokkaita ja joustavia automaatioratkaisuja. Ominaisuuksia on mahdollista laajentaa mm. Pythonilla, Javalla tai monilla muilla ohjelmointikielillä toteutetuilla kirjastoilla. Robot Frameworkilla onkin ympärillään rikas ekosysteemi, joka koostuu kirjastoista ja työkaluista. Kirjastoja sekä työkaluja on siis paljon erilaisiin tarkoituksiin. Opinnäytetyössä käytetään mm. SeleniumLibrary-kirjastoa. Kirjastoja sekä työkaluja ylläpidetään erillisinä projekteina. SeleniumLibrary on Robot Frameworkin web-testauskirjasto ja käyttää sisäisesti

Selenium-työkalua. Kirjastoa ylläpidetään GitHubissa ja lataukset ovat löydettävissä PyPI:stä. PyPI on ohjelmistovarasto Pythonille, josta voidaan ladata tarvittavia laajennuksia. Kirjasto tuo uusia käytettäviä avainsanoja, joita on mahdollista hyödyntää web-sivustojen navigointiin. (robotframework.org, n.d.)

3.2 Python

Opinnäytetyössä käytämme Robot Frameworkin lisäksi Pythonia attribuuttien organisointiin. Python on oliosuuntautunut korkean tason ohjelmointikieli, jossa on dynaaminen semantiikka. Python on kielenä helposti opittava, koska sen syntaksi korostaa luettavuutta ja vähentää siten ohjelman ylläpitokustannuksia. Python tukee moduuleja ja paketteja, mikä kannustaa ohjelmien modulaarisuuteen ja koodin uudelleenkäyttöön. (Python.org, 2023) (Tieturi, n.d.)

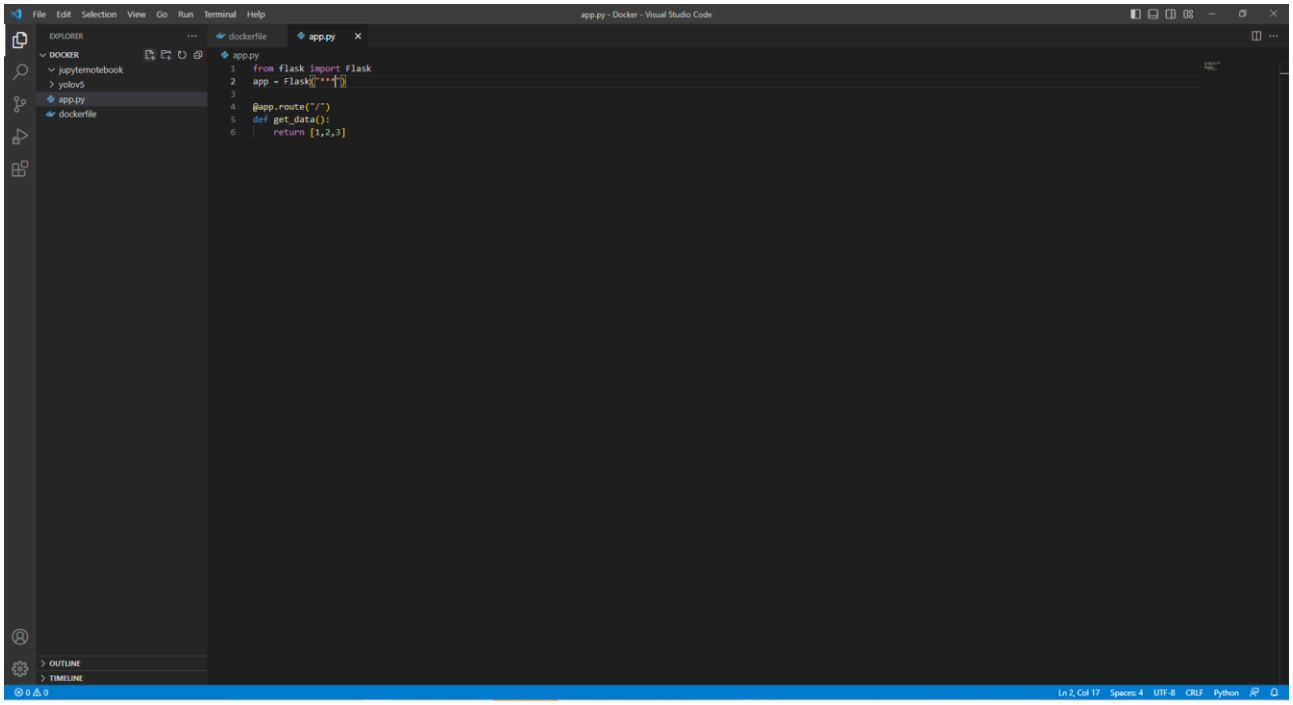
Python soveltuu hyvin nopeaan sovelluskehitykseen, koska se omaa tietorakenteet, jotka ovat yhdistetty dynaamiseen kirjoittamiseen ja dynaamiseen sidontaan. Ominaisuuksiensa takia Python onkin todella suosittu ohjelmointikieli. Pythonin käyttö on yleistynyt johtuen sen helposti opittavissa olevasta syntaksista. Lisäksi useat korkeakoulut ja yliopistot pitävät ohjelmoinnin introkursseja Python-kielellä. Sitä käytetään mm. data-analytiikkaan ja koneoppimiseen, web-kehitykseen, automatisointiin ja skriptaukseen, ohjelmistotestaukseen ja prototyyppeihin sekä työpöytäsovellusten kehitykseen. (Python.org, 2023) (Tieturi, n.d.)

3.3 Visual Studio Code

Visual Studio Code tunnetaan myös nimellä VScode. VScode on koodieditori, joka on määritelty uudelleen ja optimoitu nykyaikaisen verkko- ja pilvisovellusten rakentamiseen ja virheenkorjaukseen. Se yhdistää lähdekoodieditorin yksinkertaisuuden tehokkaisiin kehittäjätyökaluihin, kuten IntelliSense-koodin viimeistelyyn ja virheenkorjaukseen. IntelliSense korostaa syntaksia ja osaa tarjota älykkäästi täydennyksiä muuttujatyyppien, funktiomäärittelyjen sekä tuotujen moduulien perusteella. VScodessa on mahdollista tehdä virheenkorjaus suoraan editorissa. Keskeytyspisteet ja interaktiivinen konsoli mahdollistavat virheenkorjauksen käynnissä olevaan sovellukseen. Versionhallinta on kätevää VScodessa, koska git-komennot ovat sisäänrakennettu. Commitit on siis helppoa tehdä suoraan editorissa sekä muutosten seuranta ja

hallinta onnistuu myös sieltä. Editoriin on myös lähes rajattomasti erilaisia laajennuksia, joista osa on lisenssi maksun takana. (code.visualstudio.com, n.d)

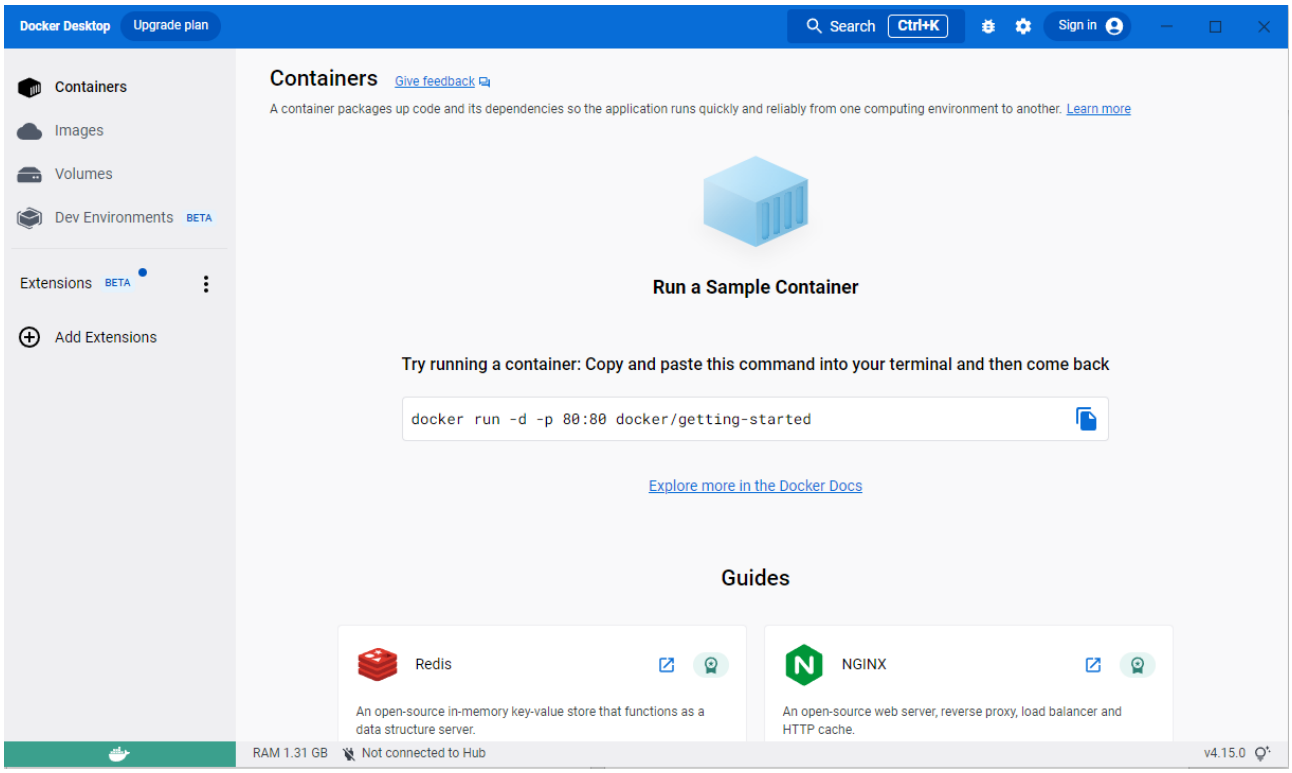
Kuva 1 Visual Studio Coden käyttöliittymä.



3.4 Docker

Docker on avoin alusta sovellusten kehittämiseen, toimittamiseen ja suorittamiseen. Dockerin menetelmiä voi hyödyntää koodin nopeaan toteuttamiseen, testaamiseen ja käyttöönottoon. Myös mahdollisuus vähentää merkittävästi koodin kirjoittamisen ja sen tuotannon välistä viivettä. Docker tarjoaa hyvät työkalut ja alustan konttien hallintaan. Sovellusta voidaan ajaa kontissa ja näin nähdä jatkuvasti koodimuutoksien vaikutus todellisuudessa. (docker.com, 2023)

Kuva 2 Docker-käyttöliittymä.



3.5 Microsoft Azure

Azure-pilvialustaan kuuluu yli 200 tuotetta ja palvelua, jotka on suunniteltu käyttäjiään tuottamaan uusia ratkaisuja. Projektissa käytämme Azure DevOps Repos -tuotetta versionhallintaan. Tuote tarjoaa yksityiset Git-repositoryt, pull requestit, sekä koodin haun. Repositoryyn on helppo varastoida koodia ja haarojen avulla pitää yksi paikka, jossa sovelluksesta on aina toimiva versio. Azure Repos tukee myös kaikkia Git clienttejä eli on mahdollista työntää koodia mistä vain IDE:stä, editorista tai Git clientistä. Azure tarjoaa myös hyvät puitteet muutosten kommentointiin ja haaran hyväksyntään ennen päähaaraan liittämistä. (Microsoft Azure, n.d.)

3.6 Versionhallinta

Versionhallinta eli koodin varasto on palvelu, joka säilöö koodia. Opinnäytetyössä käytämme Azure DevOps Repos -tuotetta. Versionhallinnan etuna on se, että pystytään säilömään uusin versio sekä vanhat versiot ohjelmasta ja tarvittaessa hyödyntämään niitä. Ohjelman ohjelmointi vaiheessa pystytään hyödyntämään haaroja. Main-haaraksi kutsutaan haaraa, joka on aina ajan tasalla sekä toimiva versio. Kun uusia ominaisuuksia syötetään versionhallintaan, toteutetaan se toisessa

haarassa, joka voidaan myöhemmin yhdistää main-haaran päälle. Kun uusi haara lisätään main-haaran päälle, päivittyy main-haaraan muutokset. Nämä sivuhaarat ovat aina kopioita sen hetkisestä main-haarasta, jolloin on helppo seurata mitä uusia muutoksia haara tuo tullessaan ja tarvittaessa palauttaa vanha main-haara, jos muutokset rikkovat ohjelman. (Os LevelUp Koodarit, n.d.)

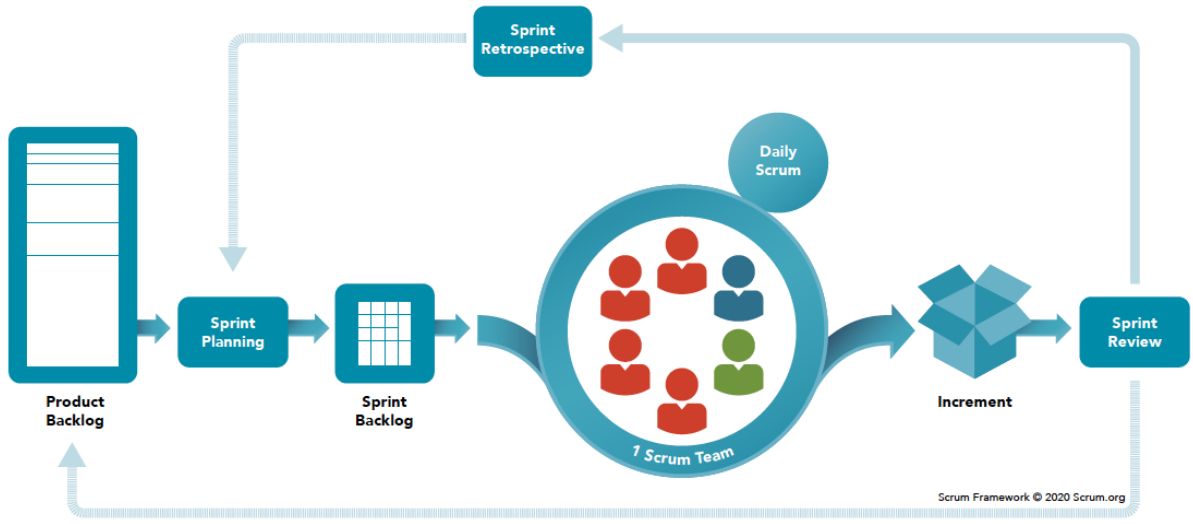
4 Menetelmät

Opinnäytetyössä hyödynnetään ketteriä menetelmiä mm. scrum ja DevOps sekä CI/CD pipeline. Menetelmät mahdollistavat paremman projektinhallinnan ja näin ollen parantaa yrityksen kuvaa asiakkaalle.

4.1 Scrum

Scrum on tapa jakaa työ pieniin osiin tiimin sisäisesti. Scrum koostuu pieninä osina kerrallaan jatkuvan kokeilun ja palautesilmukan varrella oppimisen ja kehityksen aikana. Se myös auttaa tiimiä tuomaan lisäarvoa asteittain yhteistyöllä. Scrum tarjoaa juuri tarpeeksi rakennetta, jotta ihmiset ja tiimit voivat integroitua työskentelytapaan, samalla kun se lisää oikeat käytännöt optimoidakseen heidän erityistarpeitaan.

Kuva 3 Scrum toimintamalli.



Kuva 3 Scrum toimintamallissa tulee esille scrumin eri vaiheita. Product backlog eli lista, joka sisältää tiedot siitä, mitä tuotteen parantamiseksi tarvitaan. Näin ollen myös scrum-tiimin ainoa työn lähde. Sprint planning on tapahtuma, jossa tarkoituksena on suunnitella sprintin sisältäviä tehtäviä. Sprint backlog on lista, jonka kehittäjä suunnittelee sprinttiin sisältyvistä töistä. Daily scrum on tapahtuma, joka järjestetään joka päivä. Tässä tapahtumassa tarkastellaan sprintin etenemistä ja edistymistä kohti sprintin tavoitteita. Sprint review on tapahtuma, joka järjestetään

sprintin lopussa. Tässä tapahtumassa kehittäjät ja muut sidosryhmät käyvät läpi onko sprintin tavoitteisiin päästy ja mitä seuraavaksi tullaan tekemään. Sprint retrospective on tapahtuma, jossa scrum-tiimi kerääntyy keskustelemaan edellisestä sprintistä ja tunnistamaan hyödylliset muutokset niiden tehokkuuden parantamiseksi. (Scrum.org, n.d.)

4.2 DevOps

DevOpsin tiedetään kehittävän yritysten ketteryyttä ja suoriutumiskykyä. DevOpsin taustalla on ajatus, joka perustuu jaotteluun. Jaottelussa ”siilot” on kehitys (Development eli Dev) ja palveluntarjoaja (Operations eli Ops). Jaottelu kuitenkin johti näiden väliseen krooniseen ristiriitaan, joka näkyi yrityksen heikkoina tuloksina. DevOps siis haastaakin perinteiset kehittämisprosessit kokonaan. DevOps on todettu toimivaksi ja se on onnistunut murtamaan nämä ”siilot”. Sen takia se onkin omaksuttu osaksi valtavirran liiketoimintaa eikä ole enää vain ruohonjuuritason liike. (Abildskov, 2021)

DevOpsin tavoitteena on tuoda loppukäyttäjiltä saatu palaute kehitystiimeille käsiteltäväksi ja näin mahdollisuus reagoida nopeasti markkinaolosuhteisiin ja kilpailijoiden päihittäminen tehokkaalla toiminnalla kasvaa. Nykyään DevOpsissa onkin kyse tuotehallinnasta, pilvijärjestelmien hallinnasta, teknisestä huippuosaamisesta ja terveestä yrityskulttuurista. Psykologisesti turvallinen työympäristö turvaa sen, ettei etsitä syyllistä vaan opitaan jatkuvasti virheistä. Tämä nostattaa organisaation mahdollisuuksia oppia avoimemmin ja helpommin uusia toimintatapoja. Toiminnassa tärkeää on kiinnittää huomiota automatisoinnin kohteisiin, sillä ne ovatkin vähintään yhtä tärkeitä kuin sovelluksen ominaisuuksien kehittäminen. (Abildskov, 2021)

Infrastructure as Code -lähestymistapa pilviympäristöissä tarkoittaa, että organisaatioilla on mahdollisuus kloonata koko tuotantojärjestelmä ja mahdollistaa sen testaus. Tätä hyödyntämällä päästään myös todentamaan softan ominaisuuksien oikeintoimivuus. Tähän hyödynnämme opinnäytetyössä Dockeria, joka perustuu kontteihin. Kehittäjät ja testaajat rakentavat siis konttiympäristöjä, jotka voidaan poistaa heti kun niitä ei enää tarvitse. DevOpsin hyötyihin kuuluu siis psykologinen turvallisuus ja jatkuva kokeilu, laadukkaan tuotteen/ohjelmiston tuottaminen loppukäyttäjille. Ohjelmistonsa omistavien sekä kasvavien tiimien avulla organisaatio pystyy ottamaan DevOpsin hyödyt muiden menestyvien organisaatioiden tavoin. (Abildskov, 2021)

4.3 CI/CD pipeline

CI/CD on käytäntö ohjelmistokehittäjille ja ketterälle yritykselle. Tällä ohjelmistokehitystiimit automatisoivat jatkuvan integroinnin ja toimituksen CI/CD-putken läpi. Kehitystiimit pystyvät näiden avulla toteuttamaan pieniä koodimuutoksia ja tallentamaan ne versionhallintaan. Jatkuva integrointi mahdollistaa automaattisen tavan rakentaa, pakata ja testata sovelluksia. Se myös kannustaa kehittäjiä tekemään committeja useammin, mikä johtaa parempaan yhteistyöhön ja koodin laatuun. Tämä mahdollistaa jatkuvan toimituksen, joka käynnistyy siellä, missä jatkuva integraatio päättyy. Mahdollisuuksia avautuu myös sovellusten toimitukseen valittujen ympäristöjen sisällä, mukaan lukien tuotanto-, kehitys- ja testausympäristöt. Jatkuvalla toimituksella mahdollistetaan automaattinen tapa siirtää koodimuutoksia näihin ympäristöihin. (Sacolick, 2022)

4.4 Behavior driven development

Behavior driven development eli BDD on ketterä ohjelmistokehitysmetodologia. Siinä tarkoituksena on dokumentoida ja suunnitella sovellus sen käyttäytymisen mukaan, kuten käyttäjä kokee ollessaan vuorovaikutuksessa sovelluksen kanssa. Menetelmä yhdistää, täydentää ja jalostaa käytäntöjä, joita käytetään testilähtöisessä kehityksessä ja hyväksymistestauksessa. Tyypillisesti menetelmässä kartoitetaan ensin asiakkaalta mitä ominaisuuksia tulisi olla ja näistä muodostetaan kehittäjien tavoitteet. Kun tuote täyttää vaatimukset ja on läpäissyt käyttäytymistestit, niin tuote on valmis toimitettavaksi asiakkaalle. BBD tarjoaa organisaatiolle mahdollisuuden kehittää tuotetta asiakkaan toiveiden ja palautteen suuntaan, joka taas luo arvoa yritykselle. (Fitzgibbons, 2021)

5 Testaus

Opinnäytetyössä on nyt käsitelty regressiotestausta ja siihen tarvittavia työkaluja sekä menetelmiä. Mielestäni näillä välineillä voidaan hyvin lähteä toteuttamaan projektia. Opinnäytetyön käytännön osuudessa käsitellään testien luomista ja minkälaisia ongelmia voi matkan varrella syntyä. Tarkoituksena päästä tilanteeseen, jossa testit ovat edistyneet ja saadaan projektin päätyttyä vähennettyä tarvittavia resursseja testausvaiheessa.

Projektin tavoitteena on siis luoda automaatiotestaukseen ratkaisuja Triplan Oy:lle. Testit tulevat painottumaan Triplan Oy:n tiettyyn tuotteeseen ja onkin ohjelmoitu ajettavaksi tietyillä työkaluilla ja ohjelmistoilla. Triplan Oy tarjoaa ohjelmistoratkaisuja sähköiseen asian-, dokumentin- ja arkistonhallintaan. Projektin pohjaksi otettiin vanhoja testejä. Vanhat testit olivat hankalasti ylläpidettäviä, joten niitä lähdettiin refaktoroimaan. Eli testikoodia muokattiin parantaen rakennetta, luettavuutta ja ylläpidettävyyttä ilman, että ohjelmakoodin toiminnallisuus muuttui. Uutta refaktoroitua testikoodipohjaa lähdettiin hyödyntämään uuden ohjelmiston kanssa.

Projektissa lähdettiin ensimmäisenä kartoittamaan tiettyjen testien tärkeyttä ja päädyttiin tekemään ne ensimmäisenä pois alta. Testejä on luotu eri testikokonaisuuksiin, jotka ovat laajempia testikokonaisuuksia. Testikokonaisuudet voidaan jakaa eri alueisiin mm. käyttäjiin, asiakirjoihin, asioihin, massapäivitykseen ja järjestelmän tyhjennykseen liittyviin testeihin. Testit on tarkoitus pitää mahdollisuuksien mukaan lyhyinä ja helposti luettavissa. Testeissä hyödynnetäänkin kommentointia ja attribuuttien organisoinnissa Pythonia. Kommentointia tehdään jokaisessa testissä samanlailla, jolloin luettavuudessa ei tule suuria muutoksia ja testit pysyvät selkeinä.

Testien nimestä ja yläosassa olevasta kommentista tulee esille mistä testistä on kyse. Esimerkiksi nimi voisi olla 001-Archive-käyttäjän_luonti, josta se on helposti yhdistettävissä Jirasta löytyvään testcaseen. Aina kun testeissä esiintyy jokin attribuutti, pyritään se tuomaan Python tiedostosta, jolloin attribuutteja on helppo organisoida. Tällä tavalla yhden Python -tiedostossa olevan attribuutin muutos vaikuttaa kaikkiin testeihin ja voidaan välttää ongelma, jossa samaa attribuuttia toistetaan moneen kertaan. Myös järjestelmän muuttuessa, saadaan muutokset tuotua helposti ja nopeasti myös testeihin.

Ohjelmakoodi 1 Python attribuuttien organisointi esimerkki.

```
import datetime, os, sys
from dataclasses import dataclass

now = datetime.datetime.today().strftime('%d.%m.%Y')
if os.environ.get('test_client_os') == 'win':
    win_os = True
    linux_os = False
elif os.environ.get('test_client_os') == 'linux' or
os.environ.get('test_client_os') == 'mac':
    win_os = False
    linux_os = True
else:
    print(f'No OS set in .env')
    sys.exit(1)

@dataclass
class page_objects:
    title: str = 'Uusi asiakirja'
    create_document_title: str = 'Luo asiakirja'

@dataclass
class document_tabs:
    basic_tab: str = 'xpath:/html/body/form/div[3]/table[1]/tbody/tr/td/input[1]'
    description_information_tab: str =
'xpath:/html/body/form/div[3]/table[1]/tbody/tr/td/input[2]'
    usage_rights_tab: str =
'xpath:/html/body/form/div[3]/table[1]/tbody/tr/td/input[3]'
    storage_information_tab: str =
'xpath:/html/body/form/div[3]/table[1]/tbody/tr/td/input[4]'
    relations_tab: str =
'xpath:/html/body/form/div[3]/table[1]/tbody/tr/td/input[5]'
    versions_tab: str =
'xpath:/html/body/form/div[3]/table[1]/tbody/tr/td/input[6]'

@dataclass
class create_file:
    file_name: str = 'testfile'
    form_name: str = 'xpath://*[@id="DTITLE"]'
    filetype_droplist: str =
'xpath:/html/body/form/div[3]/table[3]/tbody/tr/td[2]/select'
    file_type_value: str = '00.00.00.03.00'
    uploadfile_droplist: str =
'xpath://*[@id="data"]/table[17]/tbody/tr[1]/td[2]/select'
    get_from_disk_value: str = '9'
    create_file_button: str = 'xpath://*[@id="XCREATEFILE"]'
    save_and_continue_button: str =
'xpath:/html/body/form/div[3]/table[17]/tbody/tr/td[2]/input'
    test_docx_file = os.getcwd()+"/resources/test.docx" if linux_os else
os.getcwd()+"\\resources\\test.docx"
```

Poikkeuksia kuitenkin tulee ja suurin ongelma toistaiseksi Pythonin kanssa on tullut, jos esimerkiksi halutaan valita hausta viimeinen tiedosto. Ongelmaksi tulee, kun nettisivulta tarvitsee ensiksi hakea funktiolla *Get element count* sivulta haun asiakirjojen määrä. Kun asiakirjojen määrä on selvillä, voidaan tämä attribuutti laittaa xpathin sisälle. Kun attribuutti sisällytetään xpathiin saadaan aina viimeisin asiakirja, vaikka asiakirjojen määrä muuttuisi testauskerroilla. Kuvassa Ohjelmakoodi 2 Robot Framework esimerkki kohdassa *Click element xpath:/html/body/form[1]/div[3]/table[4]/tbody/tr[${latest_document_number}]/td[1]/a[3]* haetaan aina viimeinen asiakirja. Kohta on toteutettu niin, että attribuuttiin ${latest_document_number} haetaan montako elementtiä Python koodista kutsuttu attribuutti pitää sisällään. Tämän jälkeen attribuutti ${document_name} haetaan hyödyntämällä web-elementtien xpathissa aikaisemmin haettua numeroa.

Ohjelmakoodi 2 Robot Framework esimerkki

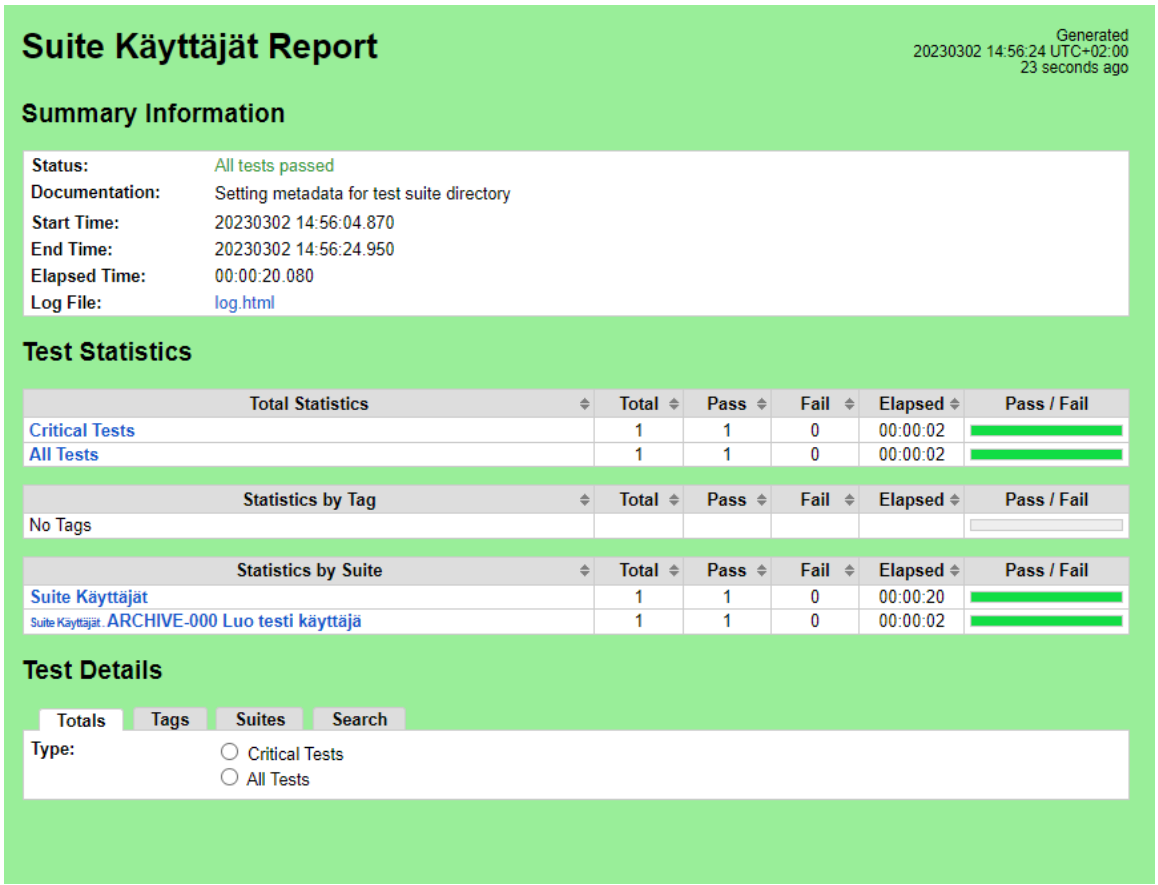
```
*** Setting ***
Documentation     Setting metadata for test suite directory
Library   SeleniumLibrary   plugins=SeleniumTestability
Variables    ../resources/uatVariables/archiveUi.py
Variables    ../resources/uatVariables/archive__documents.py
#TEST_STEPS = https://triplan.atlassian.net/browse/TWEB-1024
***Test Cases***
Try edit document - ArkistoAutomaatio111
    #Locate to search document -page
    Page should contain    ${documents.documents}
    Select frame   ${main.menu_frame}
    Wait until page contains element   ${documents.document_search}
    #TEST_STEP1: Valitaan asiakirjan haku
    Click element   ${documents.document_search}
    #Fill search criteria
    Select frame   ${main.data_frame}
    Wait until page contains element   ${file_search.form_search_name}
    Input text   ${file_search.form_search_name}   ${file_search.search_filename}
    Input text   ${file_search.form_arrive_time_min}   ${file_search.set_time_today}
    Input text   ${file_search.form_arrive_time_max}   ${file_search.set_time_today}
    #TEST_STEP2: Luodaan haku napauttamalla Hae -painiketta
    Click element   ${file_search.search_button}
    #Get latest document
    ${latest_document_number}=     Get Element Count    ${file_search.search_result_list}
    ${document_name}=   Get text   xpath:/html/body/form[1]/div[3]/table[4]/tbody/tr[${latest_document_number}]/td[1]/a[3]
    #Check file exits
    Should be equal   ${document_name}   ${create_file.file_name}
    #TEST_STEP3: Siirrytään halutulle asiakirjalle napauttamalla asiakirjan nimikettä
    Click element   xpath:/html/body/form[1]/div[3]/table[4]/tbody/tr[${latest_document_number}]/td[1]/a[3]
    Sleep   5s
    Unselect frame
    Select frame   ${main.data_frame}
    #TEST_STEP4: Asiakirja ei varattuna
    Page should not contain button    ${edit_file.edit_file_button}
    #TEST_STEP5: Varataan asiakirja
    Click element  ${edit_file.reserve_document_button}
    Page should not contain button    ${edit_file.edit_file_button}
    Click element  ${edit_file.unreserve_document_button}
    Unselect frame
    Select frame   ${main.menu_frame}
    Click element   ${main.main_page}
```

Testeissä käytetään hyödyksi INIT.robot-tiedostoa, joka pitää sisällään my setup- ja my teardown-osuudet. Eli mitä tapahtuu ennen itse testinajoa ja testinajon jälkeen. Testit sisältävät tällä hetkellä käytännössä täysin identtiset init-tiedostot. Sama init-tiedosto mahdollistaa, että kaikissa testeissä alku- ja lopputilanne on sama. My setup-osuus sisältää selaimen avauksen, Microsoft ja järjestelmän sisäänkirjautumisen. My teardown-osuus sisältää järjestelmästä uloskirjautumisen ja selaimen sulkemisen.

6 Raportointi

Tällä hetkellä projektissa lähinnä ainoa tapa tarkastella testin tuloksia on Robot Frameworkin luoma html -muotoinen raportti. Raportti tallentuu paikallisesti testiä ajettaessa. Kuvassa Kuva 4 Robot Framework onnistunut raportti nähtävissä raportti, joka sisältää yhden testikokonaisuuden ja kaikki testit ovat läpäisty. Kuvassa Kuva 5 Robot Framework epäonnistunut raportti on samasta testikokonaisuudesta, mutta epäonnistuneesta testinajosta. Tarvittaessa virhettä paikantaessa voidaan katsoa vielä tarkemmalla tasolla lokista, kuten kuvassa Kuva 6 Virhe Robot Frameworkin lokissa.

Kuva 4 Robot Framework onnistunut raportti.



Suite Käyttäjät Report

Generated
20230302 14:56:24 UTC+02:00
23 seconds ago

Summary Information

Status:	All tests passed
Documentation:	Setting metadata for test suite directory
Start Time:	20230302 14:56:04.870
End Time:	20230302 14:56:24.950
Elapsed Time:	00:00:20.080
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	1	0	00:00:02	
All Tests	1	1	0	00:00:02	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

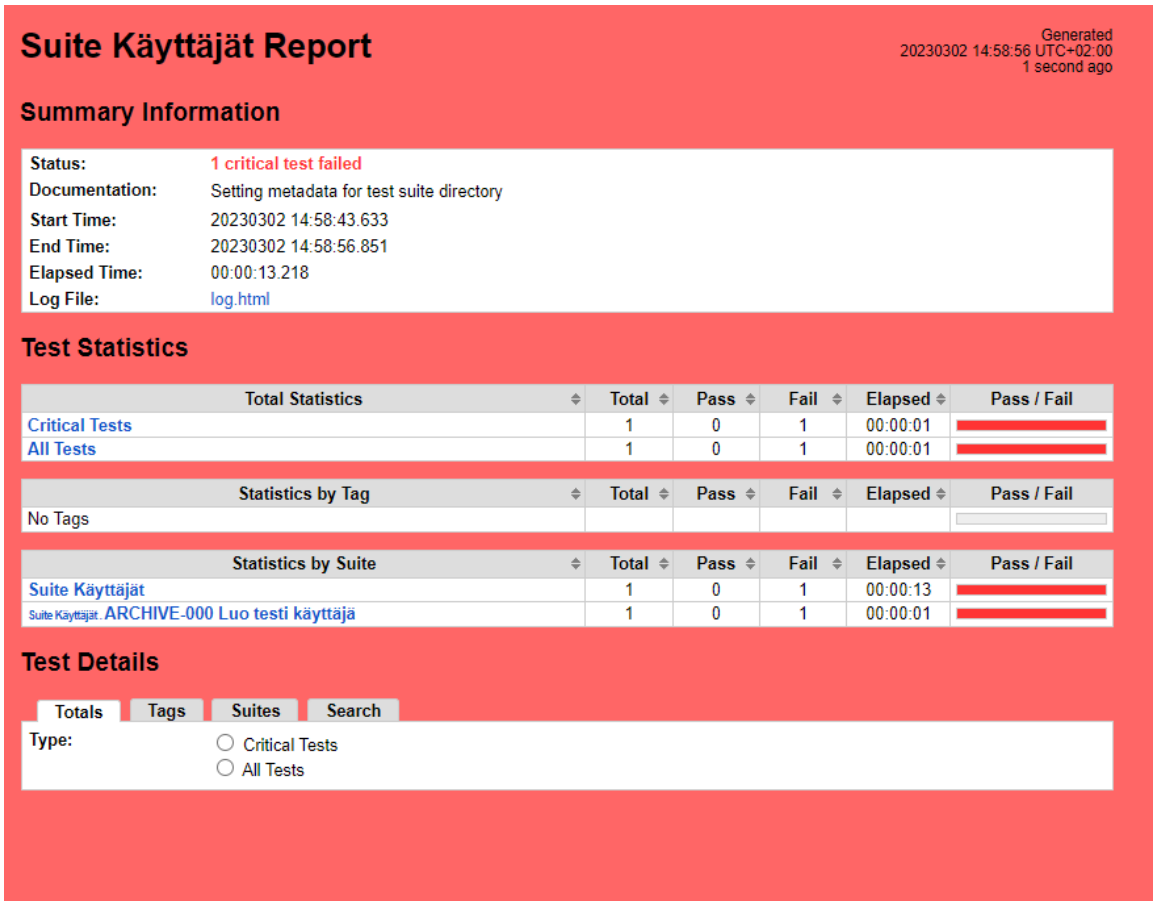
Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Suite Käyttäjät	1	1	0	00:00:20	
Suite Käyttäjät. ARCHIVE-000 Luo testi käyttäjä	1	1	0	00:00:02	

Test Details

Totals | Tags | Suites | Search

Type: ○ Critical Tests ○ All Tests

Kuva 5 Robot Framework epäonnistunut raportti.



Suite Käyttäjät Report

Generated
20230302 14:58:56 UTC+02:00
1 second ago

Summary Information

Status:	1 critical test failed
Documentation:	Setting metadata for test suite directory
Start Time:	20230302 14:58:43.633
End Time:	20230302 14:58:56.851
Elapsed Time:	00:00:13.218
Log File:	log.html

Test Statistics

Total Statistics	Total	Pass	Fail	Elapsed	Pass / Fail
Critical Tests	1	0	1	00:00:01	
All Tests	1	0	1	00:00:01	

Statistics by Tag	Total	Pass	Fail	Elapsed	Pass / Fail
No Tags					

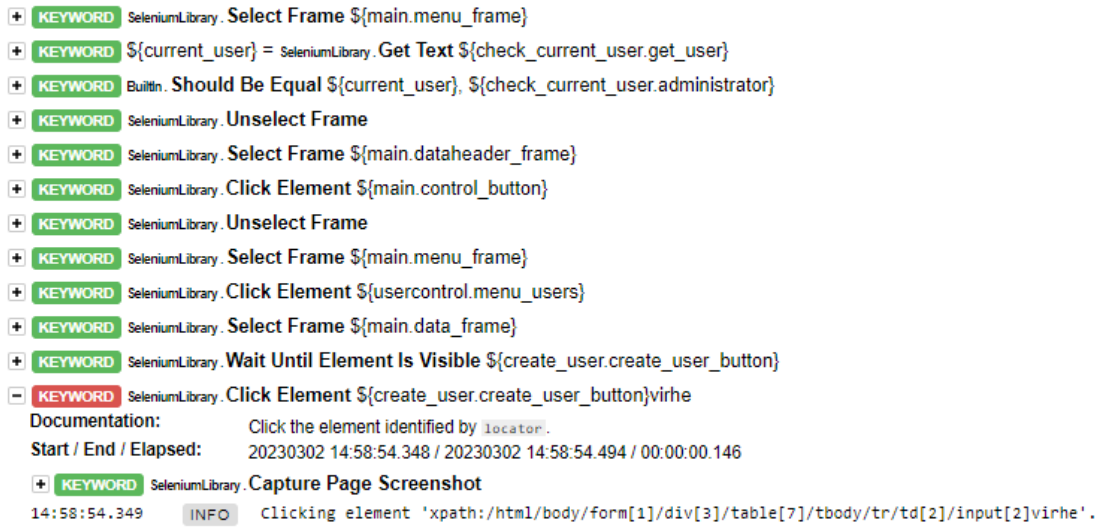
Statistics by Suite	Total	Pass	Fail	Elapsed	Pass / Fail
Suite Käyttäjät	1	0	1	00:00:13	
Suite Käyttäjät . ARCHIVE-000 Luo testi käyttäjä	1	0	1	00:00:01	

Test Details

Totals | Tags | Suites | Search

Type: ○ Critical Tests ○ All Tests

Kuva 6 Virhe Robot Frameworkin lokissa.



KEYWORD SeleniumLibrary . **Select Frame** ${main.menu_frame}
KEYWORD ${current_user} = SeleniumLibrary . **Get Text** ${check_current_user.get_user}
KEYWORD BuiltIn . **Should Be Equal** ${current_user}, ${check_current_user.administrator}
KEYWORD SeleniumLibrary . **Unselect Frame**
KEYWORD SeleniumLibrary . **Select Frame** ${main.dataheader_frame}
KEYWORD SeleniumLibrary . **Click Element** ${main.control_button}
KEYWORD SeleniumLibrary . **Unselect Frame**
KEYWORD SeleniumLibrary . **Select Frame** ${main.menu_frame}
KEYWORD SeleniumLibrary . **Click Element** ${usercontrol.menu_users}
KEYWORD SeleniumLibrary . **Select Frame** ${main.data_frame}
KEYWORD SeleniumLibrary . **Wait Until Element Is Visible** ${create_user.create_user_button}
KEYWORD SeleniumLibrary . **Click Element** ${create_user.create_user_button}virhe
Documentation: Click the element identified by `locator`.
Start / End / Elapsed: 20230302 14:58:54.348 / 20230302 14:58:54.494 / 00:00:00.146
KEYWORD SeleniumLibrary . **Capture Page Screenshot**
14:58:54.349 INFO Clicking element 'xpath:/html/body/form[1]/div[3]/table[7]/tbody/tr/td[2]/input[2]virhe'.

Tulevaisuudessa raportointia saadaan toivottavasti edistettyä. Mahdollisuudeksi on tullut esille Zephyr, jota voidaan hyödyntää testaamisessa Jiran yhteydessä. Zephyr integraatiota ei kuitenkaan

sisällytetty tähän opinnäytetyöhön, mutta sitä on lähdetty jo edistämään. Robot Framework testeihin on lisätty ominaisuuksia, joilla saadaan tuotettua tarvittavia raporttitiedostoja. Nämä raporttitiedostot vaaditaan, jotta testin tulos saadaan näkyviin Zephyriin. Zephyr voisi myös mahdollistaa tulevaisuudessa testienajon kelle tahansa organisaation henkilöstöön kuuluvalle.

7 Ongelmat

Projektin aikana on tullut monenlaisia ongelmia ja tässä luvussa niitä käsitellään. Projektiin sisältyi useita pienempiä ongelmia, joita ei kuitenkaan tässä tuoda esille. Haastavimmat ongelmat keskittyivät Dockeriin, tietokantaan, olemassa olevaan automatisointiin ja ohjelmien versioihin. Ongelmat saatiin kuitenkin lähes aina ratkaistua tai vähintään kehitettyä ohitusreitti. Ohitusreitit mahdollistivat työn jatkamisen toistaiseksi, kunnes ongelma on korjattu.

Projekti itsessään on kasvattanut huomattavasti ongelmanratkaisukykyä. Uuden ongelman aikana tuli kuitenkin luettua muutakin tietoa, joka kasvatti osaamista huomaamatta. Kuitenkaan kaikkeen ei osannut Googlekaan vastata ja tällöin oli hyvä turvautua kollegoihin sekä opinnäytetyöohjaajaan.

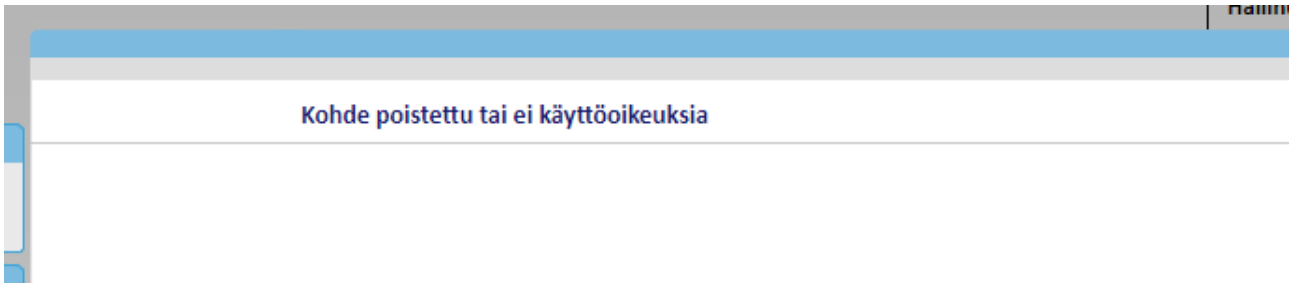
7.1 Docker-konttiympäristö

Projektin alussa havaittiin ongelma, jossa esimerkiksi testien ajo ei onnistunut alkuunkaan. Tätä tutkittiin laajemminkin, mutta ongelma ei toistunut kollegoiden työasemilla. Tulimme lopputulokseen, että työasemalla tehot eivät yksinkertaisesti riittäneet pyörittämään niin suurta kokonaisuutta. Ongelman ratkaisuksi luotiin uusi kevyempi Docker-ympäristö, joka sisälsi vain tarvittavat palvelut. Käytännössä ei siis turhaan yritetty edes pyörittää palveluita, joita testaus ei vaatinut. Uudella Docker-tiedostolla ei ongelma enää toistunut.

7.2 Käyttäjien luonti

Käyttäjän luontiin liittyviä testejä luodessa, havaittiin ongelma. Ongelmana oli käyttäjien luonti, joka myöhemmin testatessa ei onnistunut edes manuaalisesti.

Kuva 7 Esimerkki käyttöliittymä virheestä



Ongelmaa tutkittiin lokeista ja sieltä löytyikin vastauksia. Ongelman aiheuttajana oli tiettyjen käyttäjien luonti automaattisesti. Nämä käyttäjät varasivat tietyt ID-numerot kannasta, joihin järjestelmä yritti asettaa käyttäjää, jota pyrittiin luomaan. Ratkaisu kuitenkin keksittiin ja korjaukseen asti kehitettiin ohitusreitti. Tietokantaan ajettiin skripti järjestelmää käynnistäessä, jolla saatiin poistettua valmiiksi järjestelmään luodut käyttäjät.

7.3 IF-ELSE-ehdot

Testeissä käytetään Robot Frameworkin versiota 3.2.2 ja tässä versiossa ehtofunktiot eivät olleet vielä mahdollisia. Tulevaisuudessa versiota varmasti voidaan nostaa, kun mahdolliset riippuvuudet on selvitetty. Ehtofunktioita kuitenkin tarvitaan toistaiseksi sen verran vähän, ettei niiden muuttaminen myöhemmin ole suuri työ. Ongelma kuitenkin tuli esiin, kun aloitettiin asiakirjojen poistoon liittyviä testejä. Ongelma oli kuinka varmistaa kaikkien asiakirjojen poistaminen.

Testiä lähdettiin toteuttamaan ja kokeilujen kautta ratkaisu löytyi. Testi toteutettiin keywordia avuksi käyttäen. Testi esimerkki kuvassa Ohjelmakoodi 3 Esimerkki Keywordin käytöstä, joka siis hakee ensiksi olemassa olevien asiakirjojen määrän attribuuttiin funktiolla *Get element count* rivillä 19. Asiakirjojen määrän mukaan testi ajaa x määrän keywordin läpi rivillä 21. Toistaiseksi testien perusteella ollut oikein toimiva ratkaisu, sillä kaikki asiakirjat ovat poistuneet.

Ohjelmakoodi 3 Esimerkki Keywordin käytöstä.

```
*** Setting ***
Documentation     Setting metadata for test suite directory
Library   SeleniumLibrary   plugins=SeleniumTestability
Variables     ../resources/uatVariables/archiveUi.py
Variables     ../resources/uatVariables/archive__documents.py
Variables     ../resources/uatVariables/archive_emptySystem.py
#TEST_STEPS = https://triplan.atlassian.net/browse/TWEB-1013

*** Test Cases ***
Delete documents that exits
    Page should contain    ${documents.documents}
    Select frame   ${main.menu_frame}
    #TEST_STEP1: Paina "Asiakirjan Haku"
    Wait until page contains element   ${documents.document_search}
    Click element   ${documents.document_search}
    Select frame   ${main.data_frame}
    #TEST_STEP2: Paina "Hae"
    Click element   ${edit_file.search_button}
    ${document_count}=     Get Element Count    ${delete_all_documents.document_table}
    Log to console   ${document_count}
    Repeat keyword   ${document_count} times   Delete documents

*** Keywords ***
Delete documents
    #TEST_STEP3: Valitse poistettava asiakirja
        Wait until page contains element   ${delete_all_documents.first_item_onList}
        Click element   ${delete_all_documents.first_item_onList}
    #TEST_STEP4: Siirry asiakirjan Perustiedot välilehdelle ja (tarkista ettei asiakirja ole varattu)
        Wait until page contains element   ${document_tabs.basic_tab}
        Click element   ${document_tabs.basic_tab}
        ${reserved}=   Run keyword and return status   Page should contain   ${delete_all_documents.reserved}
        Run keyword if   ${reserved}   UnReserve
        Unselect frame
        Select frame   ${main.data_frame}
    #TEST_STEP5: Paina Poista-painiketta
        Wait until page contains element   ${delete_file.delete_button}
        Click element   ${delete_file.delete_button}
    #TEST_STEP6: Paina järjestelmän kysymään varmistukseen OK
        Handle alert   accept
        Handle alert   accept
        Page should contain   ${delete_all_documents.delete_succeeded}
        reload page
        Select frame   ${main.data_frame}
        Wait until page contains element   ${edit_file.search_button}
        Click element   ${edit_file.search_button}
UnReserve
        Select frame   ${main.data_frame}
        Click element   ${edit_file.unreserve_document_button}
        Sleep   2s
```

7.4 Versiopäivitys

Tiettyjen ominaisuuksien testaus oli käytännössä mahdotonta, koska niitä ei oltu vielä korjattu tai tehty versioon, jota käytettiin ympäristössä. Kun ongelma ilmeni, tilattiin versiopäivitys uusimpaan jakeluversioon. Ominaisuuksien parissa päästiin jatkamaan, mutta pian uusia ongelmia ilmeni. Ongelma sijoittui selkeästi init.robot-tiedoston aikana suoritettavaan mysetup-osioon. Testien ajossa komentokehotteelle tuli timeout-muotoinen virhe. Virhettä esiintyi selkeästi eniten Microsoft kirjautumisen aikana. Korjausyritystä lähdettiin toteuttamaan lisäämällä sleep-komentoja mysetupin yhteyteen, jolla saataisiin pelivaraa järjestelmälle suoriutua annetuista

komennoista. Virheestä esimerkki kuvassa Kuva 8 Esimerkki timeout-virheestä. Virhettä on siis saatu kierrettyä, mutta se ei ole täysin poistunut. Tutkimista aiheutti, se ettei virhe tule joka kerta Robot Frameworkin raportin mukaan samaan kohtaan. Kuvassa Ohjelmakoodi 4 Esimerkki init.robot-tiedostosta sleep-komentojen hyödyntämisestä.

Kuva 8 Esimerkki timeout-virheestä.

```
[ WARN ] Message: script timeout
  (Session info: chrome=109.0.5414.120)
```

Ohjelmakoodi 4 Esimerkki init.robot-tiedostosta.

```
*** Setting ***
Documentation     Setting metadata for test suite directory
Suite Setup       My Setup
Suite Teardown    My Teardown
Library   SeleniumLibrary   plugins=SeleniumTestability
Variables   ../resources/vars.py

*** Keyword ***
My Setup
    [Documentation]   This test case will open a Tweb application from Rihma
desktop and log in using Tweb test admin user
    Open browser   ${env.url}   ${env.browser}   options=add_argument("--ignore-
certificate-errors")
    Get Element Count    ${rihma.UI.cards}
    Click Link    ${rihma.UI.links.archiver}
    Select Window   ${azLogin.expected_window}
    Wait until page contains   ${azLogin.expected_text}
    Wait until element is visible   ${azLogin.udn_input_field}
    Input text   ${azLogin.udn_input_field}   ${anna.upn}
    Sleep   2s
    Wait Until Element Is Visible   ${azLogin.udn_input_field}
    Click element   ${azLogin.login_click_action}
    Wait until page contains   ${azLogin.expected_text}
    Sleep   2s
    Input text   ${azLogin.psw_input_field}   ${anna.psw}
    Click element   ${azLogin.login_click_action}
    Wait until page contains   ${azLogin.post_login_text}
    Click element   ${azLogin.post_login_click_action}
    Wait Until Page Contains   ${archiveLogin.expected_text}
    Input text   ${archiveLogin.udn_input_field}   ${admin.upn}
    Input text   ${archiveLogin.psw_input_field}   ${admin.psw}
    Click element   ${archiveLogin.login_click_action}
    Switch Window   ${archiveLogin.expected_window}
    Wait Until Page Contains   ${archiveLogin.expected_window}
    Page should contain     ${archiveLogin.post_login_text}

My Teardown
    Page should contain     ${archiveLogin.post_login_text}
    Select Frame    ${archiveLogin.frame}
    Click Link    ${archiveLogin.logout_click_action}
    Page Should Contain    ${archiveLogin.post_logout_text}
    Close All Browsers
```

8 Tutkimuskysymykset ja vastaukset

Tässä osiossa käsittelyssä on tutkimuskysymykset ja kuinka niihin selvitettiin vastaukset. Osiossa tuodaan esille myös, mikäli aihetta tullaan vielä kehittämään. Projektissa selkeästi keskityttiin tiettyihin kysymyksiin ja niiden tutkimiseen enemmän. Kysymyksiin saatiin kuitenkin laajalti vastattua ja monia kysymyksiä edistetään edelleen projektin jälkeen.

8.1 Testikokonaisuuksien tuottamien tietosisältöjen hallinta

Tietosisällönhallinta on pyritty tekemään mahdollisimman helpoksi käsitellä ja hyödyntää jo aikaisemmin tehtyjä tietosisältöjä myöhemmissä testeissä. Testit luodaan niin, että ne on tarkoitus ajaa tietyssä järjestyksessä. Tietyllä järjestyksellä pystymme vähentämään luotua tietosisältöä ja testien pituutta, koska esimerkiksi asiakirjaa ei luoda joka kerta uudelleen, kun sitä tarvitaan.

Aineiston putsaukseen on tällä hetkellä kaksi vaihtoehtoa. Ensimmäinen ja parempi vaihtoehto on käynnistää Docker-kontti uudelleen. Kun kontin nollaa down-komennolla ja käynnistää uudelleen up-komennolla, saadaan täysin puhdas testiympäristö. Lähdin myös luomaan tähän vaihtoehtoista tapaa, jossa viimeinen testikokonaisuus sisältäisi tyhjennystestit. Tässä kohtaa kuitenkin mahdolliseksi on tehty vain asiakirjojen poisto, jonka onnistumisprosentti on ollut tähän mennessä lähes 100 %. Kuitenkin tätä tullaan kehittämään suuntaan, jossa järjestelmään ei jäisi yhtäkään asiakirjaa tilanteesta riippumatta. Suurin ongelman aiheuttaja on ollut erilaiset muuttujat, jotka vaikuttavat poistoprosessiin ja sen kulkuun. Kuvassa Ohjelmakoodi 5 Robot Framework esimerkki asiakirjan poistosta on esimerkki asiakirjan poistoon liittyvästä testistä.

Ohjelmakoodi 5 Robot Framework esimerkki asiakirjan poistosta.

```
*** Setting ***
Documentation     Setting metadata for test suite directory
Library   SeleniumLibrary   plugins=SeleniumTestability
Variables   ../resources/uatVariables/archiveUi.py
Variables   ../resources/uatVariables/archive__documents.py
Variables   ../resources/uatVariables/archive_emptySystem.py
#TEST_STEPS = https://triplan.atlassian.net/browse/TWEB-1013

*** Test Cases ***
Delete documents that exits
    Page should contain    ${documents.documents}
    Select frame   ${main.menu_frame}
    #TEST_STEP1: Paina "Asiakirjan Haku"
    Wait until page contains element   ${documents.document_search}
    Click element   ${documents.document_search}
    Select frame   ${main.data_frame}
    #TEST_STEP2: Paina "Hae"
    Click element   ${edit_file.search_button}
    ${document_count}=     Get Element
Count    ${delete_all_documents.document_table}
    Log to console   ${document_count}
    Repeat keyword   ${document_count} times   Delete documents

*** Keywords ***
Delete documents
    #TEST_STEP3: Valitse poistettava asiakirja
        Wait until page contains
element   ${delete_all_documents.first_item_onList}
        Click element   ${delete_all_documents.first_item_onList}
    #TEST_STEP4: Siirry asiakirjan Perustiedot välilehdelle ja (tarkista ettei
asiakirja ole varattu)
        Wait until page contains element   ${document_tabs.basic_tab}
        Click element   ${document_tabs.basic_tab}
        ${reserved}=   Run keyword and return status   Page should
contain   ${delete_all_documents.reserved}
        Run keyword if   ${reserved}   UnReserve
        Unselect frame
        Select frame   ${main.data_frame}
    #TEST_STEP5: Paina Poista-painiketta
        Wait until page contains element   ${delete_file.delete_button}
        Click element   ${delete_file.delete_button}
    #TEST_STEP6: Paina järjestelmän kysymään varmistukseen OK
        Handle alert   accept
        Handle alert   accept
        Page should contain   ${delete_all_documents.delete_succeeded}
        reload page
        Select frame   ${main.data_frame}
        Wait until page contains element   ${edit_file.search_button}
        Click element   ${edit_file.search_button}
```

8.2 Testien uusinta virhetilanteessa

Testien uusinta virhetilanteessa on ollut toistaiseksi vaikein tutkittava kysymys. Tällä hetkellä kuitenkin ympäristössä, jossa testejä suoritetaan järkevintä, on käynnistää Docker-kontti

uudelleen. Tällöin saadaan todennettua, toistuuko sama virhe samassa kohtaa. Kun järjestelmän tai testin virhe on korjattu, ajetaan testit uudelleen ja katsotaan, toistuuko virhe edelleen.

8.3 Testien setup- ja teardownin-osioiden tarvittava sisältö

Testien setup- ja teardown-osiot löytyvät init.robot-tiedostosta. Init.robot-tiedostosta ajetaan aina ennen testiä setup ja testin jälkeen teardown. Setupin sisällöksi tällä hetkellä riittää selaimen käynnistys ja Microsoftin sekä järjestelmän kirjautumiset. Teardown sisältää järjestelmästä uloskirjautumisen ja selaimen sulkemisen. Setup ja teardown kopioitiin pitkälti suoraan toisesta samankaltaisesta järjestelmän testauksesta. Tähän mennessä ainoa nähtävä muutos on testien tauottamisessa. Sleep-komennoilla pyrittiin korjaamaan timeout-virheitä, joista kerrottiin aikaisemmin. Kuvassa Ohjelmakoodi 6 Esimerkki Init.robot-tiedostosta on nähtävillä setupin ja teardownin sisältö.

Ohjelmakoodi 6 Esimerkki Init.robot-tiedostosta.

```
*** Setting ***
Documentation     Setting metadata for test suite directory
Suite Setup       My Setup
Suite Teardown    My Teardown
Library   SeleniumLibrary   plugins=SeleniumTestability
Variables   ../resources/vars.py

*** Keyword ***
My Setup
    [Documentation]   This test case will open a Tweb application from Rihma
desktop and log in using Tweb test admin user
    Open browser   ${env.url}   ${env.browser}   options=add_argument("--ignore-
certificate-errors")
    Get Element Count    ${rihma.UI.cards}
    Click Link    ${rihma.UI.links.archiver}
    Select Window   ${azLogin.expected_window}
    Wait until page contains   ${azLogin.expected_text}
    Wait until element is visible   ${azLogin.udn_input_field}
    Input text   ${azLogin.udn_input_field}   ${anna.upn}
    Sleep   2s
    Wait Until Element Is Visible   ${azLogin.udn_input_field}
    Click element   ${azLogin.login_click_action}
    Wait until page contains   ${azLogin.expected_text}
    Input text   ${azLogin.psw_input_field}   ${anna.psw}
    Click element   ${azLogin.login_click_action}
    Wait until page contains   ${azLogin.post_login_text}
    Click element   ${azLogin.post_login_click_action}
    Wait Until Page Contains   ${archiveLogin.expected_text}
    Input text   ${archiveLogin.udn_input_field}   ${admin.upn}
    Input text   ${archiveLogin.psw_input_field}   ${admin.psw}
    Click element   ${archiveLogin.login_click_action}
    Switch Window   ${archiveLogin.expected_window}
    Wait Until Page Contains   ${archiveLogin.expected_window}
    Page should contain     ${archiveLogin.post_login_text}

My Teardown
    Page should contain     ${archiveLogin.post_login_text}
    Select Frame    ${archiveLogin.frame}
    Click Link    ${archiveLogin.logout_click_action}
    Page Should Contain    ${archiveLogin.post_logout_text}
    Close All Browsers
```

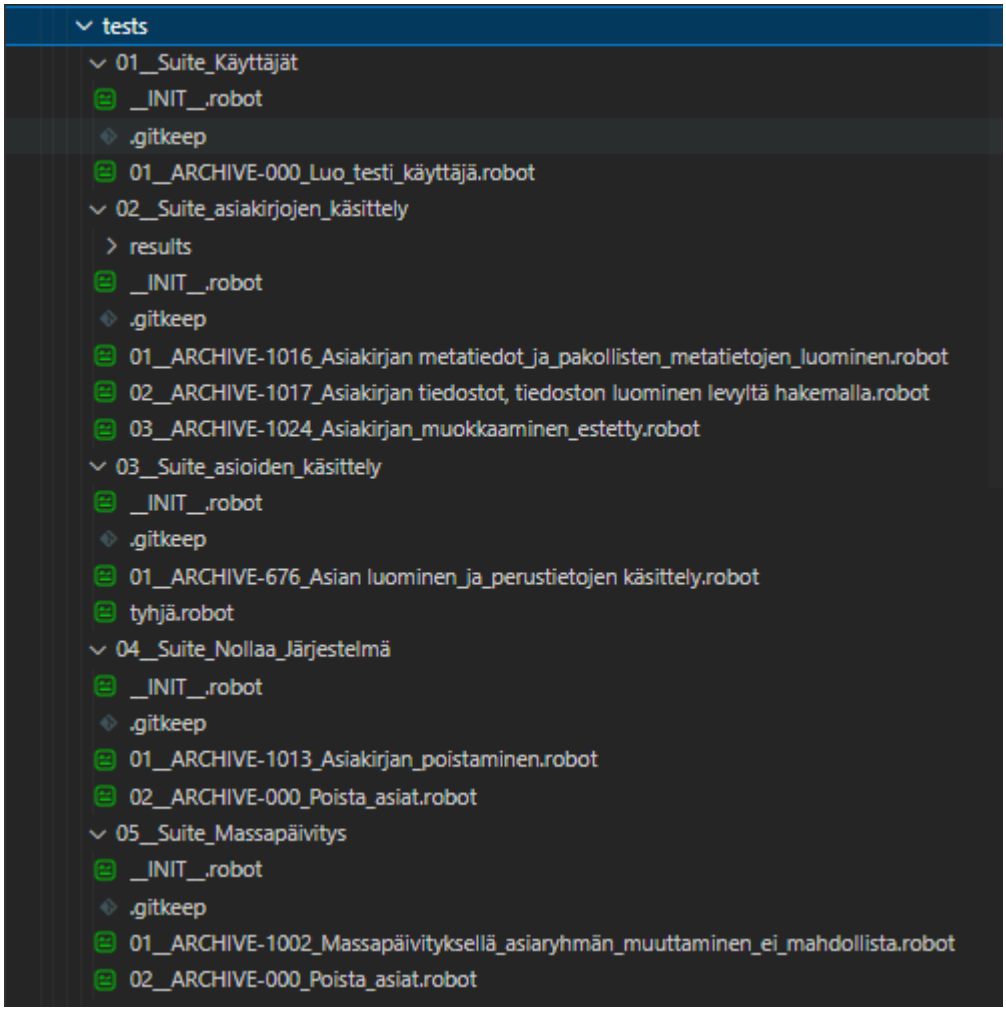
8.4 Ohjelmiston kehittyessä uusien tarvittavien testikokonaisuuksien hallinta ja lisäys

Testikokonaisuuksia on helppo hallita, sillä attribuutit on kirjattu python-tiedostoon. Kun jonkun attribuutin arvo muuttuu ja se esiintyy monessa testissä, pystytään arvo muuttamaan vain yhteen paikkaan. Python-tiedostolla organisoidut attribuutit mahdollistavat siis muutoksen moneen paikkaan yhdellä muutoksella. Testit on myös tarkasti kommentoitu, mikäli testiin on tullut uusi

elementti, jota tarvitsee käsitellä, voidaan helposti seurata kommentteja ja paikantaa kohta johon lisäys tulisi tehdä.

Uusia testikokonaisuuksia on mahdollista lisätä, mutta mikäli lisäystä ei tehdä testikokonaisuuden loppuun tulee ongelmaksi testien numerointi. Testikokonaisuuksien testit ajetaan niille määritetyssä järjestyksessä esim. (01_testi, 02_testi…). Joten mikäli uusia testejä halutaan muuttaa, tarvitsee koko listan numerointia muuttaa. Onneksi kuitenkin vaikuttaisi, etteivät testikokonaisuudet kasva sen kokoiseksi, että numerointi ei olisi hallittavissa ja tehtävissä käsin. Kuvassa Kuva 9 Tiedostorakenne nähtävissä testikokonaisuuksien tiedostorakenne.

Kuva 9 Tiedostorakenne.



9 Tulevaisuus ja tavoitteiden toteutuminen

Testausta tullaan jatkamaan opinnäytetyön jälkeenkin. Tarkoituksena jatkaa testien luomista ja organisointia. Pyritään suurentamaan testikattavuutta ja edistämään Zephyr-integraatiota. Zephyr onkin jo hyvällä alulla, sillä testit tuottavat tarvittavan tiedoston raportointiin. Testikattavuuttakin on helppo lähteä laajentamaan, koska testejä on jo suunniteltu ja kirjattu pitkälti Jiraan.

Tavoitteisiin päästiin mielestäni hyvin. Projektin suuruus ja päätös mitä opinnäytetyöhön sisällytetään, oli kysymysmerkkinä pitkään, mutta sain koottua hyvän kokonaisuuden. Tavoitteissa oli tuoda esille regressiotestauksen tärkeyttä ja kuinka sitä on mahdollista lähteä toteuttamaan. Tavoitteisiin kuului myös tietenkin luoda automaatiotestauskokonaisuutta Triplan Oy:lle ja päästä kohti tilannetta, jossa ei tarvita suuria resursseja testaukseen.

Tutkimuskysymyksiin vastattiin laajasti ja erilaisia mahdollisuuksia otettiin huomioon. Mielestäni myös on saatu luotua pohja, josta on helppo jatkaa projektia. Automaatiotestausprojekti siis jatkuu ja onkin mielenkiintoista myöhemmin palata tähän ja nähdä kuinka oma osaaminen ja projekti ovat laajentuneet.

Lähteet

Abildskov, J. (2021). *Mitä on devops?* Eficode.Com. Noudettu 17. helmikuuta 2023, osoitteesta https://www.eficode.com/fi/blog/mita-on-devops

Docker overview (2023, helmikuuta 15). Docker Documentation. https://docs.docker.com/get-started/overview/

Hoogenraad, W. (2019, toukokuuta 17). *Mitä integraatiotestaus on ja miksi teemme sitä? - IT-strategia*. IT Strategy. https://fi.itpedia.nl/2019/05/17/wat-is-integratietesten-en-waarom-doen-we-het/

Iivonen, J. (2020, toukokuuta 19). Testaussanasto—Ohjelmistotestauksen tärkeimmät termit selitettynä. *ProjectTOP*. https://projecttop.com/testaussanasto/

Tieturi Oy (ei pvm.). Tieturi. Noudettu 17. helmikuuta 2023, osoitteesta https://www.tieturi.fi/blogi/mihin-python-ohjelmointikielta-kaytetaan/

VALA Group Oy (2023, tammikuuta 30). VALA. https://www.valagroup.com/fi/blogi/mita-on-ohjelmistotestaus-ja-mita-hyotya-siita-on/

Robot Framework ry (ei pvm.). Noudettu 17. helmikuuta 2023, osoitteesta https://robotframework.org/

Sacolick, I. (2022, huhtikuuta 15). *What is CI/CD? Continuous integration and continuous delivery explained*. InfoWorld. https://www.infoworld.com/article/3271126/what-is-cicd-continuous-integration-and-continuous-delivery-explained.html

LevelUp Koodarit (ei pvm.). OS LevelUP Koodarit. Noudettu 17. helmikuuta 2023, osoitteesta https://oslevelupkoodarit.github.io/materials/versionhallinta-ja-git.html

Microsoft (ei pvm.). Noudettu 17. helmikuuta 2023, osoitteesta https://code.visualstudio.com/

Python Software Foundation (2023, helmikuuta 15). Python.Org. https://www.python.org/

Microsoft (ei pvm.). Noudettu 17. helmikuuta 2023, osoitteesta https://azure.microsoft.com/en-gb/resources/cloud-computing-dictionary/what-is-azure/

Fitzgibbons (2021). Software Quality. Noudettu 17. helmikuuta 2023, osoitteesta https://www.techtarget.com/searchsoftwarequality/definition/Behavior-driven-development-BDD

Scrum (ei pvm.). Scrum.Org. Noudettu 17. helmikuuta 2023, osoitteesta https://www.scrum.org/resources/what-is-scrum

Liite 1 opinnäytetyön aineistonhallintasuunnitelma_Nikkilä

1. Tutkimusaineiston tallennus ja säilytys

    - Aineisto tallennetaan ja säilytetään toimeksiantajan palveluissa, joka ei ole julkisesti nähtävissä. Opinnäytetyössä on tuotu esille ja käsitelty aineistoa, niin ettei siinä esiinny arkaluontoista sisältöä.

2. Henkilötietojen ja arkaluonteisten tietojen käsittely

    - Opinnäytetyöhön ei ole sisällytetty henkilötietoja tai arkaluonteista sisältöä.

3. Opinnäytetyön omistajuus

    - Opinnäytetyön aineiston ja tulokset omistaa toimeksiantaja

4. Opinnäytetyön jatkokäyttö työn valmistumisen jälkeen

    - En halua hyödyntää tai antaa tutkimusaineistoasi jatkokäyttöön.