jamk

Korvaavan testiautomaatio-ohjelmiston kartoitus ja integraatio

Netbaron Solutions Oy

Anssi Kuusniemi

Opinnäytetyö, AMK
Joulukuu 2024
Tieto- ja viestintätekniikan tutkinto-ohjelma

jamk | Jyväskylän ammattikorkeakoulu
University of Applied Sciences

jamk

Kuvailulehti

Kuusniemi Anssi

Korvaavan testi-automaatio-ohjelman kartoitus sekä integraatio

Jyväskylä: Jyväskylän ammattikorkeakoulu. Joulukuu, 35 sivua

Tieto- ja viestintätekniikan tutkinto-ohjelma. Opinnäytetyö AMK.

Julkaisun kieli: suomi

Julkaisulupa avoimessa verkossa: kyllä

Tiivistelmä

Tämän projektiluonteisen opinnäytetyön tarkoituksena oli etsiä korvaava testiautomaatio-ohjelma ja integroida se organisaation järjestelmiin. Projekti sai alkunsa siitä, ettei organisaatio saanut tarpeeksi luotettavaa dataa aikaisemmin luoduista testeistä. Projektin aikana tutkittiin erilaisia testiautomaatiosovelluksia peilaten niitä organisaation käyttämiin teknologioihin. Näitä tarkasteltiin organisaation määrittelemillä vaatimuksilla. Haluttu määrä ohjelmistoja otettiin lähempään tarkasteluun ja tutkittiin niiden käyttäytymistä annetuissa testeissä. Vahvimman ehdokkaan löytymisen jälkeen organisaation aikaisemmat testit siirrettiin kyseiselle ohjelmistolle ja luotiin valmiudet testien ajastettuun ajoon Gitlab-palvelimella. Tuloksena oli toimiva kokonaisuus, jossa jokainen kehittäjä kykeni luomaan omat testit ja lisäämään ne yhteiselle Gitlab-palvelimme, josta ne ajetaan aikataulutetusti. Työn lopputulos oli se mitä organisaatio halusikin eli palvelu, joka antaa ajankohtaista sekä luotettavaa dataa heidän tuotteestaan.

Avainsanat (asiasanat)

Testaus, Automaatiotestaus, Jatkuva integraatio, Robot Framework, Gitlab, Docker

Muut tiedot (salassa pidettävät liitteet)

Ei salassa pidettävää materiaalia

jamk

Description

Kuusniemi Anssi

Mapping and integration of replacing test automation software

Jyväskylä: JAMK University of Applied Sciences, December 2024, 35 pages

Degree Programme in Information and Communication Technology. Bachelor´s thesis.

Permission for open access publication: Yes

Language of publication: Finnish

Abstract

The purpose of this project-based thesis was to find a replacement test automation program and integrate it into the organization's systems. The project began because the organization did not receive enough reliable data from previously created tests. During the project, various test automation software's were studied, mirroring them to the technologies used by the organization. These were examined with the requirements defined by the organization. The desired number of software was taken for closer examination and their behavior in the given tests was studied. After finding the strongest candidate, the organization's previous tests were transferred to the software and capabilities were created for scheduled test execution on the Gitlab server. The result was a functional entity where each developer was able to create their own tests and add them to our common Gitlab server, from which they are run on a scheduled basis. The result of the work was what the organization wanted, namely a service that provides up-to-date and reliable data about their product.

Keywords/tags (subjects)

Testing, Automated testing, Continuous integration, Robot Framework, Gitlab, Docker

Miscellaneous (Confidential information)

No classified material

Sisältö

Kuviot

1 Johdanto

Automaatio, itsenäisesti toimivia järjestelmiä tai laitteita. Tämä prosessi on tullut osaksi useaa eri toimialaa kuten teollisuuden automatisoitu tuotantolinja, markkinoinnin tuottaminen, liikenteenohjaus ja rakennustekniikka. Tässä vain muutamia esimerkki siitä, kuinka automaatiota voidaan hyödyntää. Ihmisen työpanos voidaan siis nykyään korvata parhaimmillaan muutamalla rivillä tekstiä, joilla laitteelle tai järjestelmälle kerrotaan mitä ja milloin tehdään. Näin voidaan välttää inhimilliset virheet rutiininomaisissa tehtävissä ja pitää laatu tasaisena. Sama pätee myös ohjelmistotuotantoon, sillä erinäisiä osia tuotannosta voidaan automatisoida. Testaus on yksi näistä.

Vaikka testiautomaatiolla ei vielä voida tuottaa täydellistä testikattavuutta, on se erittäin arvokas ja hyödyllinen osa ohjelmistotuotantoa. Hyvällä suunnittelulla sekä toteutuksella testiautomaatiolla voidaan havaita ohjelmistovirheitä jopa minuuteissa uuden ohjelmistokoodin lisäyksestä. Perinteinen manuaalien testaus on edelleen välttämätöntä ja tuottaa arvokasta dataa, koska esimerkiksi erilaisten syötteiden testaamista on vaikeaa automatisoida. Testiautomaatiolla voidaan havaita virheitä, jotka aiheutuvat vaikkapa uuden ohjelmisto-ominaisuuden lisäyksestä tai toisen virheen korjauksesta, joka puolestaan aiheuttaa virheen toisaalle ohjelmistossa.

Kyseessä on projektiluonteinen opinnäytetyö, joka toteutetaan tutkimuksellisena kehitystyönä. Työn tarkoituksena on kartoittaa uusi testiautomaatio-ohjelmisto, jolla nykyinen ohjelmisto voidaan korvata. Nykyinen ohjelmisto on kartoitettu vastaavalla tavalla noin kaksi vuotta sitten. Aiempi ohjelmisto ei kuitenkaan onnistunut tuottamaan luotettavaa dataa jatkuvien ohjelmisto-ongelmien vuoksi. Uuden ohjelmiston löydyttyä jo olemassa olevat testitapaukset siirretään uuteen testiohjelmistoon, jonka lisäksi ne integroidaan osaksi valmista kehitysympäristöä. Lopuksi ohjeet ohjelmiston käyttämiseksi kootaan yhteen. Lisäksi muut kehittäjät tulee kouluttaa tuottamaan testejä itsenäisesti.

Projektin tavoitteena on luoda toimiva kokonaisuus, jossa tuotettuja testejä ajetaan automatisoidusti halutussa ympäristössä. Lisäksi tavoitteena on saada organisaation ohjelmiston luotettavuudesta ajankohtaista dataa, jonka jälkeen testikattavuutta voidaan myös lisätä. Mitä kattavammaksi testit saadaan, sitä paremmin se palvelee kohdeorganisaation tarpeita. Tällöin organisaatio saa paitsi arvokasta, myös ajankohtaista tietoa ohjelmiston eheydestä. Näin organisaatio voi luvata asiakkailleen eheän ja luotettavan ohjelmistokokonaisuuden.

2 Työn tausta ja tavoitteet

2.1 Kohdeorganisaatio

Kohdeorganisaationa toimii yksityisen sektorin organisaatio, joka tuottaa kokonaisvaltaisia toiminnanohjausjärjestelmiä erilaisiin yritysten tarpeisiin. Näitä ovat esimeriksi taloushallinnon ja varastohallinnon järjestelmät. Organisaatio on toiminut Suomessa noin kahdenkymmenen vuoden ajan ja sen asiakaskunnasta löytyy merkittäviä Suomalaisia yrityksiä.

Organisaatio työllistää noin kaksikymmentä henkilöä ympäri suomen. Henkilöstön toimenkuvat jakautuvat pääsääntöisesti asiakaspalvelun ja myynnin sekä tuotekehityksen tehtävien osa-alueille. Organisaatio on panostanut paljon dokumentointiin ja toiminnanohjaukseen, jonka avulla voidaan varmistua siitä, että organisaatossa suoritetaan oikeaa työtä, oikea-aikaisesti. Nykyisin kohdeorganisaatio on osa suurempaa pohjoismaista konsernia. Opinnäytetyön toteutus on aloitettu ennen yrityskauppoja, joka ei kuitenkaan ole vaikuttanut opinnäytetyön suunnitelmaan tai sen toteutukseen.

2.2 Tavoite

Kyseessä on projektiluonteinen opinnäytetyö, jonka tavoitteena on luoda toimiva ja kustannustehokas kokonaisuus, mahdollisimman vähäisillä ohjelmistomäärillä. Työn tarkempana tavoitteena on kartoittaa organisaatiolle testiautomaatioon käytettävä ohjelmisto, jonka myötä vanhat testit voidaan tuoda uuteen järjestelmään ja ajaa ne automatisoidusti projektin yhteydessä erikseen päätettävällä alustalla. Lisäksi tavoitteena on kerätä ajankohtaista dataa ohjelmiston luotettavuudesta, jonka jälkeen testikattavuutta voidaan lähteä myös lisäämään.

Mitä kattavammaksi testit saadaan tuotettua, sitä paremmin se palvelee kohdeorganisaation tarpeita. Tällöin organisaatio saa paitsi arvokasta, myös ajankohtaista tietoa ohjelmiston eheydestä, jonka myötä organisaatio voi luvata asiakkaillensa myös eheän ja luotettavan ohjelmiston.

2.3 Tutkimusasetelma

Opinnäytetyön tutkimusongelmana on nykyisen ohjelmiston toimimattomuus. Nykytilanteessa organisaation testiautomaation käyttö vaatii useita eri ohjelmistoja. Käytössä on TestComplete käyttöjärjestelmä, jolla tehdään testit, Gitlab, johon testit tallennetaan sekä Jenkinss jossa ne ajetaan.

Organisaatiossa on kaksi vuotta aikaisemmin toteutettu vastaavanlainen tutkimus, jossa edellä mainitut testiautomaatioon käytettävät ohjelmistot ovat valikoituneet käyttöön. Ohjelmisto ei kuitenkaan lopulta onnistunut tuottamaan luotettavaa dataa jatkuvien ohjelmisto-ongelmien vuoksi.

Opinnäytetyön tutkimuskysymyksiksi nousivat:

- Voiko testejä tuottaa ilman aiempaa osaamista?
- Kuinka voidaan varmistaa, että testit ovat luotettavia myös tulevaisuudessa?
- Minkälaisen budjetin uusi testiohjelmisto vaatii?

Tutkimuskysymyksiin haetaan vastauksia keräämällä tietoa eri ohjelmistoista ja vertailemalla näiden ominaisuuksia. Koska aiemmin toteutetusta tutkimuksesta on kulunut vain kaksi vuotta, tulen osin käyttämään tuolloin tehdyn kartoituksen materiaaleja sekä tuloksia, jotta voin verrata niitä itse hankkimiini aineistoihin.

3 Tietoperusta

3.1 Ohjelmistotestaus

Ohjelmistotestaus on olennainen osa ohjelmistokehitystä. Tällä tarkoitetaan työtä, jolla pyritään varmistamaan, että tuotettava ohjelmisto vastaa odotuksia ja toimii halutulla tavalla. Ohjelmistotestausta suorittaa ihminen, joka manuaalisesti tutkii ohjelmiston toimintaa loppukäyttäjän näkökulmasta. Ohjelmistotestausta voidaan suorittaa ohjelmistokehityksen eri vaiheissa ihan esituotannosta ylläpitoon saakka. Kasurinen (2013) kirjoittaa kirjassaan, että testaustyö on jatkuvaa vertailua. Testauksen tarkoituksena on tunnistaa suunnitelmasta poikkeavat kohdat ja varmistaa että valmis työ on tarkoituksen mukainen. (Kasurinen 2013, 10.)

Ohjelmistotestaus on paljon muutakin kuin vain testin tekemistä ja tulosten tarkastelua. Testaus vaatii myös testien suunnittelua, analysointia sekä testitapausten ja toteutuksen suunnittelua.

(Spillner & Linz 2021, luku 1.) Ohjelmistotestausta voidaan suorittaa manuaalisesti tai automaatiolla. Helalan (2024) mukaan testaus aloitetaan jo ohjelmointivaiheessa, jolloin ohjelmoija itse testaa muutokset esimerkiksi yksikkötestauksella ja laittaa muutokset eteenpäin vasta testauksen jälkeen. Vaikka automaatiolla pystytään korvaamaan ihmisen työpanos osittain testaamisessa, se ei poista manuaalisen testauksen tarvetta. Esimerkiksi hyväksyntätestaus, jossa ohjelmiston loppukäyttäjä testaa vastaako ohjelmisto heidän tarpeitaan sekä vaatimuksiaan. Kasurinen lainaa kirjassaan (2013, 78) Ramleria sekä Wolfmeieria jotka tiivistävät testiautomaation käytön ohjelmistotestauksessa seuraavasti: automaation avulla estetään ehjiä moduuleja rikkoutumasta ja käsin testauksella tutkitaan uusia tapoja rikkoa toiminnallisuuksia.

3.2 Automaatiotestaus

Automaatiotestaus on yksi ohjelmistotestauksen muoto. Automaatiotestauksessa luodaan erilaisia komentoja, joilla kerrotaan tietokoneelle mitä pitää tehdä. Rytkösen & Kakkosen (2023, 244) mukaan oikein toteutettu testiautomaatio mahdollistaa nopeamman, helpomman sekä riskittömämmän tavan muokata ohjelmistoa, koska muutoksista saadaan palaute nopeasti. Automaatiotestaus on erityisen hyödyllinen silloin, kun halutaan suorittaa testejä toistuvasti. Automaatio pois sulkee myös inhimilliset virheet, jotka voivat tapahtua rutiininomaisissa toistuvissa testitapauksissa. Tällaisia ovat esimerkiksi savutestit, joilla pyritään varmistamaan, että ohjelmiston elintärkeät ominaisuudet eivät rikkoudu uusien ominaisuuksien myötä. Myös rasitustestit on helpompi toteuttaa automaatiotestauksella, joissa ohjelmistolle syötettään suurempi määrä dataa ja halutaan saada varmistus, että ohjelmisto pystyy käsittelemään sen.

Toiminnallinen testaus puolestaan tarkoittaa testauksen muotoa, jossa ohjelmistosta testataan toiminnallisia vaatimuksia. Näitä ovat esimerkiksi lomakkeen täyttäminen tai syötettyjen tietojen käsittely. Toiminnallista testausta voidaan suorittaa sekä manuaalisesti että automaatiolla, mutta automaatiolla toteutetut testit voidaan toistaa helposti tarpeen vaatiessa. Ei-toiminnallisessa testauksessa tarkistetaan ohjelmistosta ne asiat, mitkä eivät näy loppukäyttäjälle. Näitä voisivat olla esimerkiksi, suorituskykytestaus tai tietokantatestaus. (Toiminnallinen testaus – Miksi se on avainasemassa sovelluskehityksessä? 2023.)

Regressiotestauksella tarkoitetaan testausmuotoa, jossa testataan jo aikaisemmin testattuja toimintoja uudelleen ja uudelleen. Testit yhdistetään kehitysympäristön esim Gitlab CI/CD-putkeen,

jossa testit ajetaan automaattisesti, kun kehittäjä lisää muutoksia. Regressiotestausta hyödynnetään myös silloin, kun ohjelmisto on siirtynyt ylläpitovaiheeseen. Tällöin testiautomaatio varmistaa ohjelmiston toiminnan, vaikka uusia muutoksia ei enää lisättäisi.

Testityökaluja on erilaisia. Helalan (2024) mukaan testityökalun valinta riippuu testattavan ohjelmiston arkkitehtuurista, esimerkiksi onko se Windows-, mobiili vai web-sovellus. Useilla niillä on aivan omat syntaksit, miten testejä kirjoitetaan. Näistä esimerkkeinä ovat avainsanapohjainen testaus, nauhoitus sekä koodityylinen testaus. Avainsanatestauksessa testausohjelmistoon on luotu jo valmiiksi erilaisia komentoja, joita hyödyntämällä luodaan testitapaukset antamalla erilaisia argumentteja.

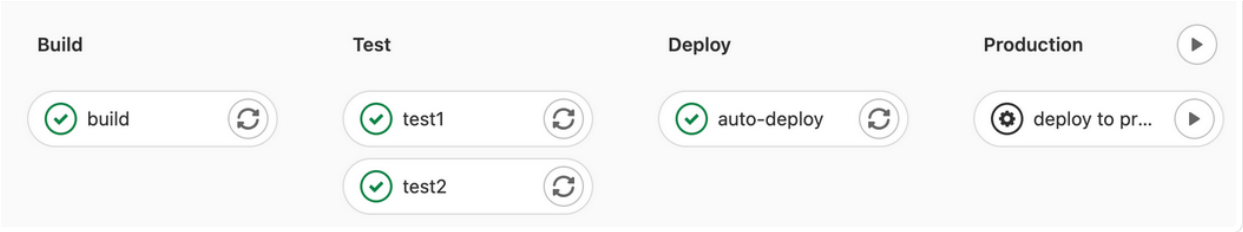
Koodityylinen testaus on lähimpänä perinteistä ohjelmointia ulkoasullisesti. Komennot luodaan ohjelmointia muistuttavalla syntaksilla. Funktiot käyttäytyvät samoin, kuin ohjelmoinnissa. Nauhoituksessa testaaja tekee käyttöliittymälle halutut toiminnot ja ohjelmisto nauhoittaa kaiken tapahtuneen muuttaen sen joko avainsana tai koodityyliseen syntaksiin. Useimmiten nauhoitetut testit eivät enää toimi, kun niitä yrittää ajaa nauhoituksen jälkeen. Nauhoitusta voi kuitenkin käyttää erinomaisesti esimerkiksi testien suunnitteluun (Thomson, Morgan, Samaroo, Kurowski, Williams, Salmon & Weymouth 2024, luku 6). Jos nauhoitusta käytetään testien luontiin, joudutaan sitä usein täydentämään tai korjaamaan jotta se toimisi.

Testiautomaatioon tulee investoida. Se vaatii ihmiseltä suunnittelua, toteutusta sekä ylläpitoa. Toisin sanoen se ei synny eikä ylläpidä itseään automaattisesti. Myös testituloksien analysointiin tarvitaan ihmistä. (Rytkönen & Kakkonen 2023, 244.)

3.3 Gitlab

Gitlab on avoimen lähdekoodin versionhallintajärjestelmä, joka helpottaa erilaisten ohjelmistoprojektien ylläpitämistä. Kehittäjät pystyvät tekemään oman haaran ohjelmistokoodista, johon he pystyvät lisäämään omia töitään ilman, että se vielä tässä vaiheessa vaikuttaa muiden töihin. Kun halutut muutokset on tehty ja laatu varmistettu, voidaan muutokset kerralla lisätä alkuperäiseen ohjelmistokoodiin. Tällöin se tulee myös muiden kehittäjien saataville.

Gitlabissa on myös monia muita hyödyllisiä ominaisuuksia esimerkiksi CI/CD-putki (jatkuva integraatio/jatkuva kehitys) joka varmistaa aina lisättävän muutoksen laadun. Jatkuva integraation tarkoittaa sitä, että jokainen koodin lisäys rakentaa ohjelmiston uudelleen. Jatkuva kehitys vuorostaan tarkoittaa sitä, että päähaaralla on koko ajan valmiina ns. tuotantovalmis versio ohjelmistosta. (Merode & Merode2023, luku 2.) Käytännössä tämä tarkoittaa, että ohjelmiston hakemistoon luodaan gitlab-ci.yml tiedosto, johon määritellään mitä CD/CD-putkella halutaan tehdä (Ks. Kuvio 1). Tehtävät jaotellaan stage määrityksillä ja yml-tiedostoon voidaan määritellä missä järjestyksessä nämä pitää suorittaa. Tässä projektissa käytössä olevat staget ovat build, jossa rakennetaan Dockerimage, sekä test, jolla ajetaan itse automatisoidut testit.



Kuvio 1 CI/CD-putki

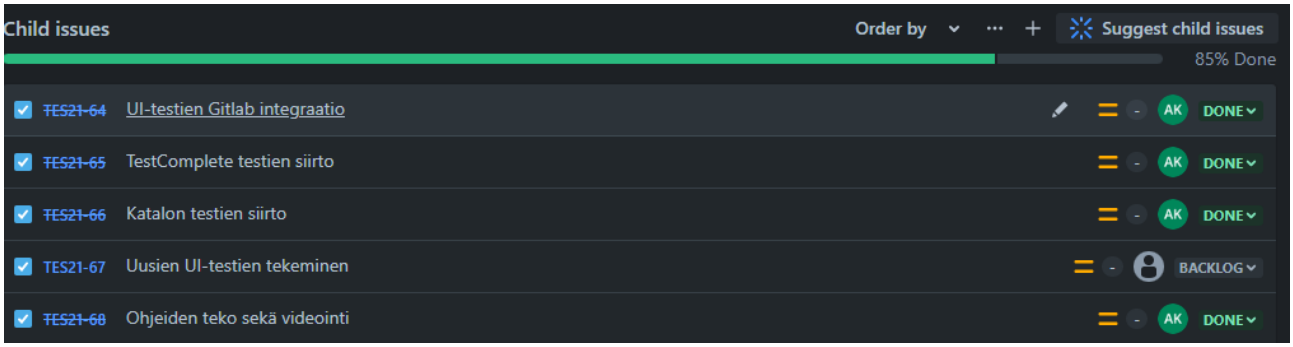
.

3.4 Atlassian

Atlassian on organisaatio-ohjelmistoja tuottava yritys, joka tarjoaa erilaisia ohjelmistokehitystiimin työtä helpottavia ohjelmistoja. Näillä voidaan organisoida sekä aikatauluttaa esimerkiksi tulevia julkaisuja, tulevia töitä ja testauksia.

Jira puolestaan on Atlassian:n tarjoama tehtävienhallintaohjelmisto. Tällä ohjelmistolla projektit voidaan helposti aikatauluttaa osiin, jonka lisäksi työt voidaan jakaa pienemmissä osissa kehittäjille. Näin projektinhallinnasta tulee läpinäkyvämpää, kun kaikki voivat nähdä paitsi meneillään olevat, myös tulevat työt. Siten toiset kehittäjät voivat myös aikatauluttaa omia töitään, jos ne vaativat esimerkiksi toisen työn valmistumisen.

Jirassa voidaan luoda eri tasoisia tikettejä, joita ovat Epic, task ja sub-task. Tiketillä, josta käytetään myös termiä issue, tarkoitetaan toimeksiantoa. Tässä työssä jäljempänä puhutaan tiketeistä (Ks. Kuvio 2). Epic on korkeamman tason tiketti, eli projektitaso, joka pilkotaan taskeihin. Taskit ovat projektin pienempiä osia, jotka on jaettu edelleen sub-taskeihin, jotka puolestaan ovat yksittäisiä tehtäviä.



Kuvio 2 Jira-tiketit

Confluence on Atlassianin tarjoama organisaatiowikiohjelmisto. Tällä ohjelmistolla organisaatio voi tuottaa kehitystyössä tarvittavia dokumentteja. Organisaatio voi luoda omanlaisensa dokumenttipohjat ja jakaa ne Confluencen kautta. Myös yhteensopivuus muiden Atlassian sovelluksien kanssa on hyödyllinen, esimerkiksi Jiran työt voi tuoda koottuna Confuencessa olevaan dokumenttiin.

3.5 Docker

Termi kontti (englanniksi ”container”) on suosittu teknologia. Docker tarjoaa eristetyn, kevyen sekä turvallisen ympäristön ajaa ohjelmia välittämättä siitä, mitä kiinteälle asemalle on asennettu. Konttiin asennetaan vain ne liitännäiset mitä ohjelmisto vaatii toimiakseen. Konttia voisi kuvailla pieneksi virtuaalikoneeksi, joka pystytetään, että haluttu ohjelma toimii. Tällainen kontti luodaan tekemällä Dockerfile-tiedosto, johon määritellään kaikki ohjelmistot, niiden liitännäiset sekä tiedostot mitä kontissa ajettavaan ohjelmistoon tarvitaan.

Edellä mainittujen toimien jälkeen Dockerfilestä luodaan Docker image. Docker image on eräänlainen kehys, josta kontti rakennetaan. Tästä kontti voidaan käynnistää ja kun halutut toiminnot on tehty, voidaan se myös pysäyttää. Sanaa image voidaan käyttää konttien pystyttämiseen niin

kauan kun image on ajan tasalla, jos esimerkiksi konttiin tarvitaan uusia liitännäisiä, image voidaan rakentaa uudelleen muokatusta Dockerfilesta.

Konttien käytössä on monia hyötyjä ohjelmistokehityksessä. Nopeus pystytyksessä sekä alasajossa, täysin itsenäinen kokonaisuus ja mahdollistaa että jokaisella kehittäjällä on identtinen ympäristö. Kontteja on helppo siirtää ja monistaa. (Mikä on kontti ja Docker? 2022.)

3.6 Ubuntu

Ubuntu on Linux-käyttöjärjestelmä, joka koostuu avoimen lähdekoodin ohjelmistoista. Ubuntun periaatteisiin kuuluu, että sen käyttö on ilmaista, sitä voidaan käyttää haluamallaan kielellä ja mahdollisimman esteettömästi. Lisäksi käyttäjä voi muokata sitä haluamallaan tavalla.

4 Pöytäkirjat sekä tarve-, toiminta- ja vaatimusmäärittely

Kohdeorganisaation toimintamallin mukaan jokaisesta tapaamisesta luodaan pöytäkirja. Pöytäkirjan sisältö vaihtelee sen mukaan, onko kyseessä esimerkiksi projektin katselmointi, projektin päätöspalaveri tai aloituskatselmointi. Projektin vastuuhenkilö luo ja valmistelee pöytäkirjan, kirjaa omat ehdotuksensa valmiiksi sekä määrittelee tarvittavat roolit. Kaikki pöytäkirjat tallennetaan Confluence-palveluun, johon on luotu projektille oma kansiorakenne.

Organisaatiolla käytetään ohjelmistoprojektien yhteydessä dokumentointiin TTV-pohjaa, johon luodaan projektista suunnitelma. Sana TTV on lyhenne tarve-, toiminta- ja vaatimusmäärittelystä. Tällä varmistetaan, ettei projekteja aloiteta ilman hyvää suunnitelmaa ja pahimmassa tapauksessa aloiteta tekemään aivan väärää tuotetta. Dokumentti sisältää kuusi eri aihetta, joita täydennetään tarpeen mukaan.

Määrittelydokumentti

Tähän osaan kootaan osittain samoja asioita mitä läpikäydään jo aloituskatselmoinnissa. Näitä ovat projektin henkilöstöresurssit kuten vastuuhenkilö, projektipäällikkö, testaajat, ohjeiden päi-

vittäjä ja uutisoija. Mikäli kyseessä on asiakasprojekti, myös organisaation ulkopuoliset katselmoijat tuodaan tähän dokumenttiin. Lisäksi dokumentoidaan projektin nykytilanne, uuden tavoitetilan kuvaus sekä projektin määräaika-arvio, johon mennessä se on valmis.

Tarve-, toiminta- ja vaatimusmäärittely

Tähän dokumenttiin kirjataan kootusti projektin tietokanta tai asetusvaatimukset. Lisäksi siihen kirjataan projektiin liittyvät terminologiat, katselmoinnit, käyttäjäoikeudet sekä aikatauluarviot. Tänne tuodaan myös projektille luodut Jira tiketit.

Tietokanta- ja testaussuunnitelma

Tähän osioon kootaan projektin vaatimat UI-, tietokanta- sekä testaussuunnitelmat.

Markkinointitutkimus

Markkinointitutkimus tehdään, mikäli sellaiselle on tarvetta. Tähän osioon tulee markkinointitutkimus, kilpailija-analyysi ja SWOT-analyysi.

Tuotteistaminen

Tuotteistamisen tarkoituksena on, että liiketoiminnalliset hyödyt tuodaan näkyviksi jo heti projektin alussa niin toimittajalle kuin asiakkaalle. Tänne kirjataan tuotteen elinkaari, palvelunkuvaus sekä palvelun yhteyshenkilöt.

Budjetti

Tässä osiossa projekti budjetoidaan. Jokaisen osion ajankäyttö ja hinnoittelu suunnitellaan erikseen. Hinnoitteluun löytyy omat hinnoittelumallinsa sekä sisäiselle että ulkoiselle kehitykselle.

Edellä mainittuja osioita täydennetään aina projektin mukaan, mutta kaikille projekteille ei välttämättä tarvita jokaista osiota. Esimerkiksi organisaation sisäisillä projekteilla ei ole tarvetta tuottaa markkinointisuunnitelmaa.

5 Uuden testiautomaatio sovelluksen kartoitus

5.1 Aloituskatselmointi

Ennen projektin aloittamista tulee organisaaton ohjeiden mukaisesti pitää projektin aloituskatselmointi. Tässä katselmoinnissa käydään lävitse useita asioita koskien lupaa aloittaa projektin TTV-dokumentin täydentäminen.

Ensimmäisenä tuli käsitellä projektin taustatiedot sekä perusteet TTV:n aloittamiselle. Olennaisimpana osana oli kuvailla sen hetkinen tilanne ja selventää, miksi muutos on tarpeen ja millainen toivottu lopputulos on. Tämän jälkeen tarkasteltiin vaatimuksia siitä, mitä uudelle testiautomaatio sovellukselle asetetaan. Esimerkiksi helppokäyttöisyys, ajoympäristö, raporttien selkeys ja nauhoitusominaisuus olivat asioita, joita uudelta sovellukselta toivottiin. Sekä asetettiin myös hintaraja ohjelmistolle.

Seuraavaksi katselmoinnissa tarkasteltiin projektin erilaisia tarpeita, UI-suunnitelmia, tietokantasuunnitelmia, testaussuunnitelmia, markkinatutkimusta sekä tuotteistamista. Projektin ollessa organisaation sisäinen eikä vaikuttanut mihinkään organisaation muuhun tuotteeseen, ei näille ollut tarvetta kyseisessä projektissa. Tämän jälkeen toteutettiin alustava aikataulusuunnitelma, jossa tuotiin esiin näkemys siitä, kauanko TTV:n työstämiseen menisi, milloin sen tulisi olla valmis ja mikä tämän projektin kokonaisläpimenoaika on. Ajan tarve tuli täyttää myös budjetointi osioon.

Seuraavaksi huomioitiin vielä TTV-projektin henkilöstötarve, jossa minut todettiin projektin vastuuhenkilöksi ja projektipäällikkönä toimi esihenkilöni. Lopuksi katselmoinnissa sovittiin siitä, kuinka usein projektin aikana toteutetaan välikatselmointeja, joissa seurataan projektin toteutumista ja varmistetaan että se pysyy aikataulussa. Lisäksi sovittiin henkilöistä, jotka näihin katselmointeihin osallistuvat.

Kun kaikki tarvittavat asiat oli hyväksytty ja projektin tavoitteet oli määritelty, luotiin organisaation omille intrasivustoille projektitiketti. Tiketin pääsääntöisenä tarkoituksena on seurata projektin ajankäyttöä ja työvaiheita sekä varmistaa, että annettu tuntimäärä riittää. Viimeinen vaihe suunnittelussa oli luoda Jiran työtiketit, näiden tulee olla pilkottu tarpeeksi pieniin osiin ja kuvattu mahdollisimman tarkasti.

5.2 TTV

Kohdeorganisaatio oli toteuttanut vastaavanlaisen projektin aiemmin. Tällöin käyttöön oli valittu TestComplete ohjelmisto. Tätä varten oli luotu pistetaulukko, johon oli kirjattu jo valmiiksi pisteytettävät kohteet. Myös silloisia vertailu ohjelmistoja oli taulukossa valmiina (Ks. Liite 1). Jätin nämä vertailuun, jotta voisin tarkastella niitä uusin silmin. Lisäsin 3 vertailtavaa ohjelmistoa taulukkoon, joiden selvitin olevan suosittua nykyaikaisessa automaatiotestauksessa (Best Automation Testing Tool for 2024. 2024.) (The Top 20 Test Automation Tools of 2025. 2024.).

Jokaisen ohjelmiston ominaisuudet pisteytettiin taulukkoon (Ks. Liite 2). Pisteytys perustui omaan näkemykseeni siitä, kuinka hyvin tarkasteltu ohjelmisto vastaa organisaation asettamia tarpeita. Taulukon jälkeen koostin kuvaukset siitä, mistä pistekokonaisuudet koostuvat. Raportoinnissa nostin esiin kyseisen ohjelmiston hyvät ja huonot ominaisuudet sekä muut valinnan kannalta oleellisimmat seikat, joita havaitsin vertailua tehdessäni.

Organisaatiolla oli aikaisemmin tuotettu automaatiotestejä myös Katalon sekä TestComplete ohjelmistoilla. Organisaatio halusi minun selvittävän myös sen, olisiko vanhoja testejä mahdollista konvertoida uuteen ohjelmistoon joillakin työkaluilla vai tuleeko kaikki testit muuntaa käsin.

5.3 Ohjelmistojen vertailu

Vertailuosuus alkoi vierailemalla kunkin sovelluksen verkkosivuilla. Pyrin saamaan nopean ymmärryksen, siitä kuinka ohjelmisto toimii, minkä hintainen se on ja kuinka usein päivityksiä ohjelmistoon julkaistaan. Hinnoittelun osalta tarkistin minkälaisia tasoja heillä on tarjolla ja kuinka monta lisenssiä niihin kuului. Kiinnitin huomiota erityisesti siihen, tuliko joitain merkittäviä ominaisuuksia lisää hinnan noustessa ja muuttuuko tuen määrä eri hinnoittelumalleissa.

Ohjelmiston toimintaa tarkastellessa kiinnitin huomiota siihen, että käytetäänkö ohjelmistoa omalta käyttöliittymältä vai voiko sen ladata lisäosana erilliselle editorille. Seuraavaksi tutkin, millä syntaksilla testit kirjoitetaan. Selvitin, onko kyseessä esimerkiksi avainsanapohjainen tai koodityyppinen ratkaisu, vai onko kyseessä nauhoitusmalli tai jokin muunlainen ratkaisu.

Hyödynsin myös paljon Youtube palvelua. Valitsin tämän palvelun, koska tiesin sieltä löytyvän helposti erilaisia käytännön esimerkkejä työkaluista. Käytin samanlaista hakusanaa kaikkiin sovelluksiin: create test ”ohjelmiston nimi”. Saaduista hakutuloksista tarkastelin kuinka haastavaa tai helppoa testitapauksia oli tehdä kyseisellä ohjelmistolla. Havainnoin myös, kuinka usein videoita on julkaistu ja kuinka tuoreita uusimmat videot olivat.

Seuraavaksi tarkastelin sovelluksen käyttäjäyhteisöä. Tarkoituksena oli selvittää kuinka paljon sekä kuinka aktiivisia nämä yhteisöt ovat ja kuinka helposti yhteisöltä saa tarvittaessa tukea. Tarkastelin kyseisen ohjelmiston omia kanavia mutta myös esimerkiksi Stack Overflow-sivustoa, joka on laajalti käytössä tietotekniikan puolella.

Kohdeorganisaatio on perustettu noin 20 vuotta sitten ja osin käytössä oleva teknologia on vielä näiltä ajoilta. Osa toiminnoista on siis toteutettu tekniikoilla, joita nykyaikaisilla sovelluksilla ei enää käytetä. Tällainen on esimerkiksi i frame, jota ei juurikaan käytetä enää. Jotta organisaatiossa voidaan käyttää uutta testiautomaatiosovellusta organisaation oman tuotteen kanssa, tuli i frametuki tutkia erityisen tarkasti. I framea käytetään organisaation sovelluksessa runsaasti siirryttäessä eri sovellusten välillä, jonka vuoksi sen yhteensopivuus valittavan automaatiosovellukseen oli oleellinen.

Karkeasti jaoteltuna sovellukset voitiin jaotella kahteen luokkaan, eli niihin, joilla on oma käyttöliittymä sekä niihin, joita voidaan käyttää laajennuksena erillisessä editorissa. Oman käyttöliittymän omaavat sovellukset ovat kalliimpia käyttää. Robot Framework, Selenium sekä Playwright olivat ainoat täysin ilmaiset ohjelmistot. TestCafe sekä Cypress olivat osin ilmaisia, mutta jos halusi heidän ohjelmistonsa lisäksi nauhoitusominaisuuden, tuli näistä maksaa.

Havaitsin myös, että suuri osa ohjelmistoista, joita käytettiin omalta käyttöliittymältä, toimi lähes identtisellä logiikalla toisiinsa nähden. Jopa ulkoasut olivat asetteluineen melkein kopioita toisistaan. Itse asettelu oli kuitenkin hankalaa ja erilaisten moduulien käyttö työlästä. Myös kansiorakenne oli vaikeasti hahmotettavissa. Useimmissa näistä sovelluksissa oli erilaisia tyylejä tehdä testejä. Pohdintaa aiheutti se, että onko kaikki tarpeen, sillä jokaisella tavalla saadaan aikaiseksi oma syntaksi, mutta palveleeko se kohdeorganisaation tarpeita kovin hyvin, jos jokainen kirjoittaa testinsä eri syntaksilla.

LeapWork sekä UIPath toimivat myös omilla käyttöliittymillään ja olivat drag & drop tyyppisiä ohjelmistoja. Näissä erilaisia komentoja haettiin sivupalkista, joita raahattiin ruudulle ja ketjutettiin muiden komentojen kanssa. Tällä idealla on varmasti helppo oppia tekemään testejä, mutta toisaalta pitemmät testit varmasti tulisivat olemaan hankalammin luettavia ”kuplien” runsaan määrän takia.

Lisäksi oli ohjelmistoja, joille ei tarvinnut ladata erillistä käyttöjärjestelmää. Nämä olivat myös ilmaisia ohjelmistoja, poikkeuksina Cypress sekä TestCafe. Näillä testit voitiin kirjoittamaan ilmaiseksi erillisellä editorilla, mutta nauhoitus oli maksullinen toiminto, joka vaati ohjelmiston oman käyttöjärjestelmän lataamisen.

Lopullisiin käytännön testeihin valikoituivat Robot Frameworkin sekä Playwright. Robot Framework valittiin sen monipuolisuuden sekä selkeyden vuoksi. Koska Playwright on Microsoftin tuote, tiedettiin sen olevan luotettava. Kummatkin valituista ohjelmistoista olivat myös ilmaisia käyttää. Näille toteutin testit, jossa testattiin erilaisia toimintoja organisaation ohjelmistosta. Iframe aiheutti ongelmia Playwrightille, sillä Playwright ei siirtynyt iframesta toiseen tarpeeksi luotettavasti. Ongelma oli havaittavissa, kun siirtyi iframesta toiseen.

Koska Robot Frameworkilla voi käyttää sekä Selenium- että browser-kirjastoja, tein testit molemmille. Browser -kirjasto on Playwrightin tuottama kirjasto, jonka testeillä oli samoja ongelmia kuin aiemmin mainitulla Playwright syntaksilla. Myös testit Selenium:lla olivat selkeämpiä, koska käytettävä iframe määriteltiin vain kerran. Browser-kirjastoa käytettäessä iframe määriteltiin aina silloin, kun siinä sijaitsevaa elementtiä käsiteltiin.

5.4 Välikatselmoinnit

Välikatselmointeja pidettiin viikoittain ja niihin osallistuivat minun lisäksi toimitusjohtaja, kehityksen vastaava sekä projektipäällikkö. Tällöin esitin havaitsemani asiat ja jos mahdollista, karsimme myös vaihtoehtoja pois. Samalla varmistettiin myös, että projekti etenee aikataulun mukaisesti sekä käsiteltiin mahdollisia haasteita, joita projektin edetessä oli noussut esiin.

Vertailuprojektin loppupuolella varteenotettaviksi ohjelmistoiksi nousivat Selenium, Robot Framework sekä Playwright. Minua pyydettiin perehtymään näihin ohjelmistoihin vielä tarkemmin sekä kirjoittamalle näille testit, joilla pystyisin vertailemaan toimivuutta.

5.5 Loppukatselmointi

Loppukatselmoinnissa käsiteltiin koko projekti ja kirjattiin toteutuneet asiat. Katselmoinnin aikana pistetaulukkoa tarkasteltiin kokonaisuutena, jonka lisäksi käsiteltiin kunkin ohjelmiston ominaisuudet sekä mahdolliset puutteet. Myös toteutuneet työtunnit tarkastettiin ja varmistettiin, että ne ovat pysyneet sovituissa rajoissa.

Loppukatselmoinnin päätteeksi päätimme ottaa koekäyttöön Robot Framework ohjelmiston. Tämän ohjelmiston eduksi todettiin olevan sen helppokäyttöisyys sekä mahdollisuus käyttää useita kirjastoja. Robot Frameworkin ollessa avoimen lähdekoodin ohjelmisto ei myöskään lisenssikuluja muodostu ja jokainen kehittäjä kykenee tuottamaan testejä omassa ympäristössä. Samoin tarvittavia ohjelmistoja saatiin karsittua, koska Robot Framework pystytään asentamaan jo käytössä oleviin koodi editoreihin. Robot Frameworkissa tullaan hyödyntämään Selenium-kirjastoa.

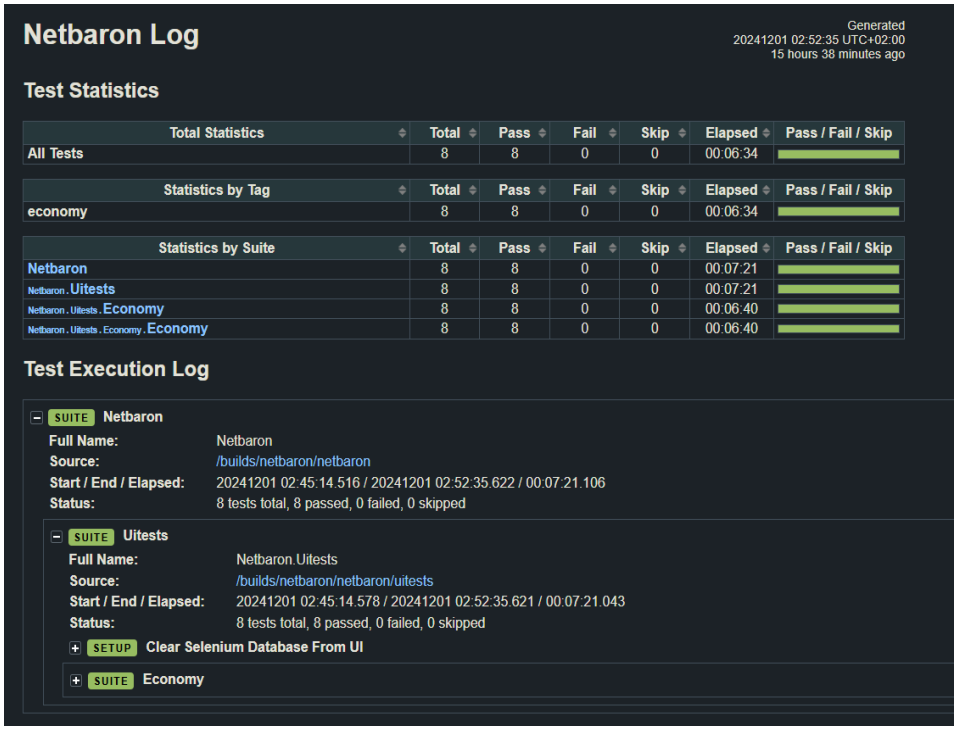
6 Robot Framework käyttöönotto

Robot Framework on avoimen lähdekoodin testiautomaatio kehys. Robot Frameworkin kehitys alkoi vuonna 2005 Pekka Klärckin diblomityöstä ja ensimmäinen Robot Frameworkin versio julkaistiin samana vuonna Nokia Network:n toimesta (Bisht 2013, 26).

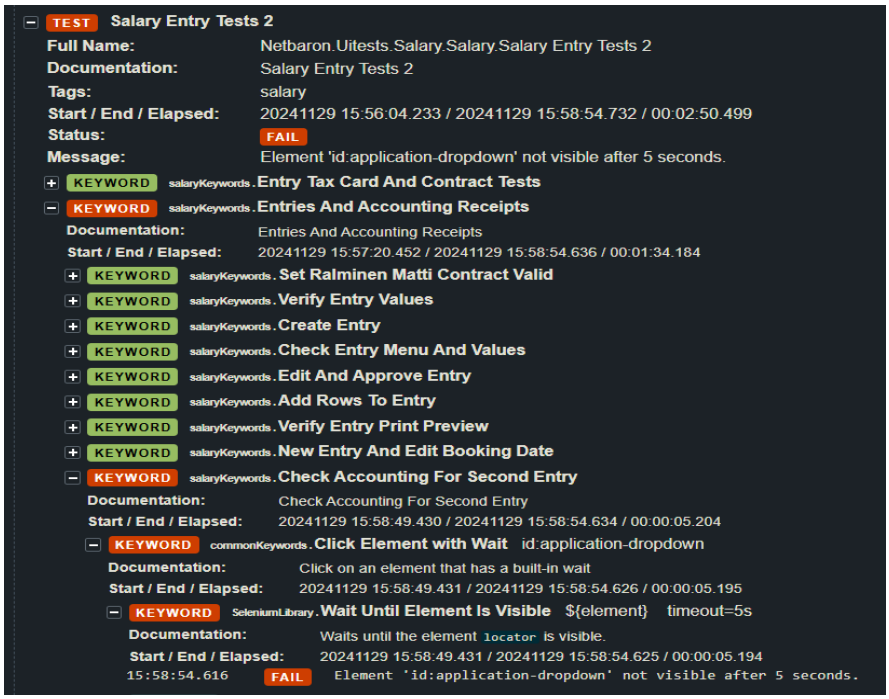
Käyttettävissä on useita kirjastoja erilaisiin testauksiin, esimerkiksi Selenium- sekä Browser-kirjastot web hyväksymistestauksiin ja Appium-kirjasto mobiilitestauksiin. Robot Framework on avainsanapohjainen kehys, jossa suoranaisia ohjelmointitaitoja ei tarvita. Avainsanoja voi luoda myös lisää Pythonila ja JavaScriptillä käyttökohteen ja tarpeen mukaan.

Robot Framework tuottaa erittäin selkeän raportin ajetuista testeistä. Raportti näyttää ajetut testit sekä tiedot niistä, jotka onnistuivat tai epäonnistuivat (Ks. Kuvio 3 sekä Kuvio 4). Lisäksi se näyttää myös testeissä käytetyn ajan. Avainsanat ovat luettelomaisessa listauksessa ja niitä on mahdollista

selata. Mikäli testi on onnistunut, on väri vihreä. Testien epäonnistuttua kohta puolestaan näytetään punaisena. Robot Framework ottaa oletuksena kuvan epäonnistuneesta avainsanasta.



Kuvio 3 Onnistunut test suite



Kuvio 4 Epäonnistunut test suite

6.1 Projektin aloitus

Projektin aloituskatselmoinnissa tehtävät pilkottiin pienempiin osiin, jonka lisäksi ne myös aikataulutettiin. Jiraan luotiin tiketit eri tehtävänosille, joiden lisäksi jokaiselle tiketille asetettiin arvio ajallisesta työmäärästä. Lopuksi nämä koottiin TTV dokumentille. Task-tasoja oli kaikkiaan neljä, joita olivat Gitlab toiminnot, TestComplete testien siirto, Katalon testien siirto sekä ohjeistuksen luominen.

Gitlab -toimintojen sub-taskeja olivat ohjeistuksen luonti sivustolle, kansiorakenne, yml-tiedoston sekä testien ajoaikataulun luominen. TestComplete sekä Katalon pilkottiin sub-taskeiksi sovelluskohtaisesti. Ohjeistuksen luominen sisälsi sub-taskit, joita olivat videoinnit, kirjalliset käyttö- ja asennusohjeet sekä koulutuksen.

6.2 Ohjelmistojen asennus

Itse Robot Frameworkin asentaminen koneelle on nopeaa ja helppoa. Ainoa esivaatimus mitä koneelle piti ladata, oli Python kielen tuki. Pythonin asennuksen jälkeen tuli muistaa lisätä Pythonille polku järjestelmämuuttujiin. Itse Robot Frameworkin asennus onnistui komentoriviltä komennolla

```
C:\Users\anssi>pip install --upgrade robotframework
```

Kuvio 5 Robot Frameworkin asennuuksen komentokäsky

--upgrade varmistaa, että Robot Frameworkista asennetaan viimeisin versio. Selenium-kirjasto asennetaan komennolla.

```
C:\Users\anssi>pip install --upgrade robotframework-seleniumlibrary
```

Kuvio 6 Selenium-kirjaston asennuksen komentokäsky

Näillä asennuksilla pääsin aloittamaan testien kirjoittamisen.

6.3 Robot Frameworkin alkuasetelmat

Robot Frameworkin käyttö oli oman käyttökokemukseni perusteella helpointa aloittaa yhdellä tiedostolla, johon loin kaikki avainsanat sekä testit.

```
1   ***Settings***
2
3
4   ***Variables***
5
6
7   ***Keywords***
8
9
10  ***Test Cases***
```

Kuvio 7 Robot Frameworkin tyhjä template

*****Settings*****

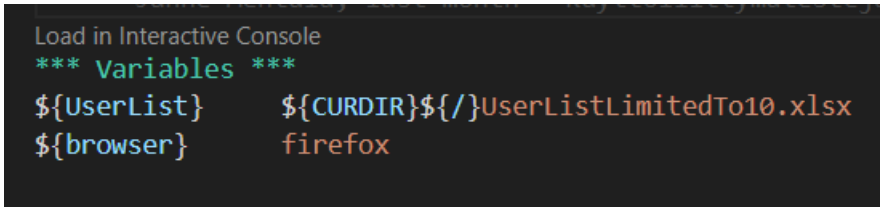
Settings kohtaan määritellään testeissä tai avainsanoissa käytettäviä liitännäisiä. Näitä voivat olla esimerkiksi käytettävät kirjastot, ulkoiset resurssitiedostot, testien asetukset sekä mitä halutaan testien jälkeen tapahtuvan.

```
Load in Interactive Console
*** Settings ***
Library     Collections
Library     DateTime
Library     OperatingSystem
Library     ExcelLibrary
Library     SeleniumLibrary
Library    String
Resource    Variables.resource
```

Kuvio 8 Robot Framework Settings

Variables

Variables kohtaan voidaan määritellä muuttujia tai listoja. Testeissä käytettävät muuttujat ovat sovelluskohtaisia tai yleisesti käytettäviä. Yleisiä muuttujia voidaan kutsua kaikkien sovellusten testeissä. Näitä ovat esimerkiksi käyttäjätunnukset sekä salasanat.



Kuvio 9 Robot Framework Variables

Keywords

Keywords osio sisältää itsetehtyjä avainsanoja. Kuten muuttujia, niin myös avainsanoja löytyy useasta paikasta. On olemassa sovelluskohtaisia avainsanoja sekä yleisiä avainsanoja. Yleisiä avainsanoja ovat esimerkiksi selaimen avaaminen, kirjautuminen palveluun, sivuston latausstatuksen tarkistaminen sekä palvelusta uloskirjautuminen. Dokumentaatio avainsanassa on tärkeä osa, joka selkeyttää avainsanat toiminnan muille kehittäjille. Tämä kirjoitetaan kohtaan [Dokumentation].

Kohdeorganisaation hyvän dokumentointitavan mukaan, avainsanan dokumentoinnissa tulee kertoa, mitä avainsalalla testataan ja tieto siitä, jos avainsana tarvitsee jotain argumentteja. [Arguments] kohtaan tulee määritellä muuttujat, jotka avainsanaa kutsuttaessa asetetaan. Lisäksi avainsanan dokumentoinnissa kerrotaan, mitä avainsana mahdollisesti palauttaa, mikäli palautusarvo on määritelty.

```
*** Keywords ***
Load in Interactive Console
Open Chrome
    [Documentation]    Opens Chromebrowser
    ${chrome_options}    Evaluate    sys.modules['selenium.webdriver'].ChromeOptions()    sys, selenium.webdriver
    Call Method    ${chrome_options}    add_argument    --disable-search-engine-choice-screen
    Call Method    ${chrome_options}    add_argument    --ignore-certificate-errors
    Call Method    ${chrome_options}    add_argument    --disable-popup-blocking
    Call Method    ${chrome_options}    add_argument    --no-sandbox
    Call Method    ${chrome_options}    add_argument    --headless
    Open Browser    ${URL}    chrome    options=${chrome_options}
    Maximize Browser Window

Load in Interactive Console
Open Edge
    [Documentation]    Opens Edge
    ${egde_options}    Evaluate    sys.modules['selenium.webdriver'].EdgeOptions()    sys, selenium.webdriver
    Call Method    ${egde_options}    add_argument    --disable-search-engine-choice-screen
    Call Method    ${egde_options}    add_argument    --ignore-certificate-errors
    Call Method    ${egde_options}    add_argument    --disable-popup-blocking
    Call Method    ${egde_options}    add_argument    --no-sandbox
    Call Method    ${egde_options}    add_argument    --headless
    Open Browser    ${URL}    edge    options=${egde_options}
    Maximize Browser Window
```

Kuvio 10 Robot Framework Keywords

*****Test Cases*****

Test Cases on paikka missä sijaitsee itse testit. Testejä voidaan luoda joko Robot Frameworkin tai Selenium kirjaston valmiilla avainsanoilla, joiden lisäksi voidaan kutsua itse luotuja avainsanoja. Kuin myös avainsanoissa, myös testitapausten selkeä dokumentointi [Dokumentation] kohtaan on tärkeää, jotta muutkin kehittäjät ymmärtävät mitä asiassa testataan ja mitä sen odotetaan saavuttavan. [Tags] kohdassa testeille voidaan määritellä tagi, joilla määritellään ryhmä missä testi ajetaan. Esimerkki tageja voisivat olla smoke, regressio tai salary.

```
*** Test Cases ***
Run | Debug | Run in Interactive Console
Salary Settings Create
    [Documentation]    Salary Settings Create
    [Tags]    salary
    Create Groups
    Create Union And Pension Ins
    Copy And Verify New Salary Class And Delete
    Create Salary Class Over time
    Create Salary Class Supervisor
    Create Salary Type Lecture
    Create Salary Class Lecture
    Create Vacation Multiplier
    Create New Salary Period And Check Days
    Create Salary Period 2015 05
    Create Employment Type And Specifier
    Create Salary Year 2010
    #Create Reduction Profile    #pois
    Create Edit Delete Helper Variables
    Create Entry Number
    Create Entry Transaction Number
    Save Faulty Salary Category
    Create Salary Period 2016 01
    Create Salary Period
```

Kuvio 11 Robot Framework Test Cases

6.4 TestComplete testien siirto

Kohdeorganisaatiolla oli aiempana testiautomaatiosovelluksena käytössä TestComplete niminen sovellus. Näissä testit tuotettiin TestCompleten omassa ohjelmistossa, mutta ne oli tallennettu organisaation omalle Gitlab-palvelimelle. Testien ajot tapahtuivat Jenkins-palvelimen kautta. Ennen testien siirron aloittamista latasin TestComplete ohjelmiston, koska testit eivät olleet luettavassa muodossa Gitlab-palvelimella. TestComplete ohjelmistossa testit oli jaoteltu sovellus kohtaisesti: salary, economy, settings yms. Näiden alle testit oli puolestaan jaoteltu eri testitapauksiin.

Yritin etsiä mahdollista sovellusta tai työkalua, jolla voisin konvertoida testit Robot Frameworkille tällaista vaihtoehtoa en löytänyt. Testit jouduttiin siirtämään syntaksilta toiselle manuaalisena kopiointina. Testien siirtämisen koin helpoimmaksi siten, että muutin avainsanatestit koodi muotoon. Tällä havaitsin helposti mitä iframea käsitellään ja mitä muuttujia käytetään.

Katalon testien siirto oli vastaavanlainen prosessi kuin TestComplete testien siirto. Testit olivat Gitlab-palvelimella mutta eivät myöskään olleet luettavassa muodossa. Jouduin luomaan uudet tunnukset Katalon palveluun, jotta pääsin käsiksi heidän käyttöjärjestelmäänsä ja sain kohdeorganisaation testit luettavaan muotoon. Katalon ohjelmistossa on mahdollisuus muuttaa testit Robot Frameworkille mutta tämä on maksullinen ominaisuus eikä testien määrän nähden investointi olisi järkevää. Testit olivat jo valmiiksi koodi syntaksilla toteutettu, joten manuaalinen kopiointi sujui ongelmitta.

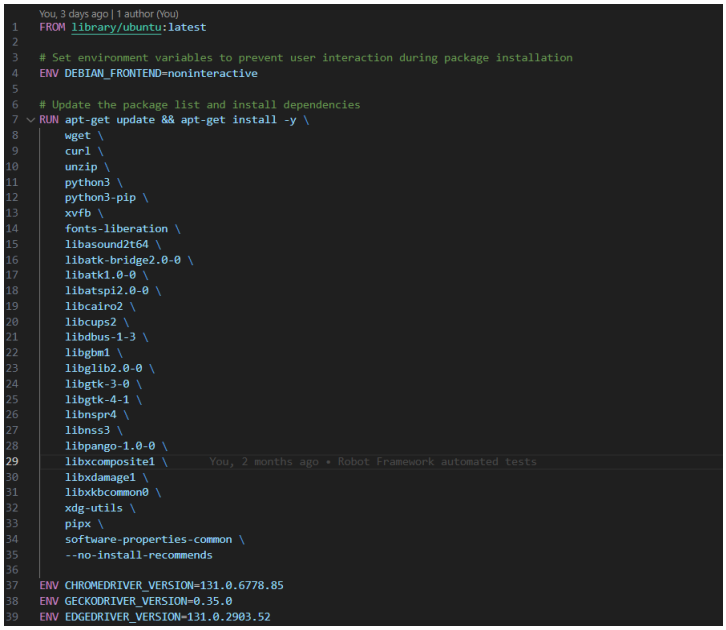
6.5 Docker image

Docker imagen luonti tuotti hankaluuksia. Suurin osa muista tahoista tuntui käyttävän Jenkinsiä automaattisiin ajoihin eivätkä ajaneet testejä Gitlabissa. Esimerkkien löytäminen oli haastavaa.

Aluksi tuli valita, mitä pohjaa kontti käyttää. Python olisi ollut helpoin ratkaisu koska Robot Framework on Pythonpohjainen sovellus. Myös Robot Frameworkille olisi ollut oma alusta, mutta se oli tarkoitettu browser-kirjastolle, jota emme ottaneet käyttöön. Päädyin rakentamaan imagea Ubuntu alustalle, koska kohdeorganisaation aikaisempi Dockerimage on myös luotu sillä.

Tämän jälkeen imageen piti asentaa kaikki tarvittavat ohjelmistot, eli Python, Robot Framework, Selenium-kirjasto, tarvittavat ajurit sekä kolme verkkoselainta (Chrome, Edge, Firefox). Pyrin tässä kohden hyödyntämään tekoälyä imagen rakentamiseen, koska en löytänyt esimerkkiä, joka olisi toteutettu kaikilla kolmella selaimella. ChatGPT antoi ainoastaan suuntaa antavia ehdotuksia mutta mikään niistä ei toiminut suoraan. Ongelmia ehdotuksessa tuotti se, että ChatGPT:n antamat ohjelmistojen versioinnit eivät olleet enää yhteensopivia selaimien kanssa.

Näin onnistuin luomaan toimivan Dockerfilen, johon oli asennettu kaikki edellä mainitut tarvittavat ohjelmistot ja niiden liitännäiset. Verkkoselaimet haetaan automaattisesti viimeisen vakaan version mukaan, mutta ajureille tuli tehdä muuttujat, joihin versiointinumerot on helpompi päivittää.



```
You, 3 days ago | 1 author (You)
FROM library/ubuntu:latest

# Set environment variables to prevent user interaction during package installation
ENV DEBIAN_FRONTEND=noninteractive

# Update the package list and install dependencies
RUN apt-get update && apt-get install -y \
    wget \
    curl \
    unzip \
    python3 \
    python3-pip \
    xvfb \
    fonts-liberation \
    libasound2t64 \
    libatk-bridge2.0-0 \
    libatk1.0-0 \
    libatspi2.0-0 \
    libcairo2 \
    libcups2 \
    libdbus-1-3 \
    libgbm1 \
    libglib2.0-0 \
    libgtk-3-0 \
    libgtk-4-1 \
    libnspr4 \
    libnss3 \
    libpango-1.0-0 \
    libxcomposite1 \      You, 2 months ago • Robot Framework automated tests
    libxdamage1 \
    libxkbcommon0 \
    xdg-utils \
    pipx \
    software-properties-common \
    --no-install-recommends

ENV CHROMEDRIVER_VERSION=131.0.6778.85
ENV GECKODRIVER_VERSION=0.35.0
ENV EDGEDRIVER_VERSION=131.0.2903.52
```

Kuvio 12 Dockerfile liitännäiset sekä webdriver versioiniti

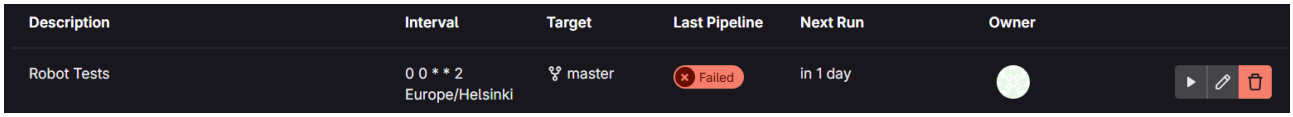
```
# Install Chrome and ChromeDriver
RUN wget https://dl.google.com/linux/direct/google-chrome-stable_current_amd64.deb &&  \
    dpkg -i google-chrome*.deb && \
    rm google-chrome*.deb && \
    wget -N https://storage.googleapis.com/chrome-for-testing-public/$CHROMEDRIVER_VERSION/linux64/chromedriver-linux64.zip && \
    unzip chromedriver-linux64.zip && \
    chmod +x chromedriver-linux64/chromedriver && \
    cp chromedriver-linux64//chromedriver /usr/local/bin && \
    rm chromedriver-linux64.zip

# Install Firefox and Geckodriver
RUN apt-get install -y firefox && \
    wget -q https://github.com/mozilla/geckodriver/releases/download/v$GECKODRIVER_VERSION/geckodriver-v$GECKODRIVER_VERSION-linux64.tar.gz && \
    tar -xzf geckodriver-v$GECKODRIVER_VERSION-linux64.tar.gz -C /usr/local/bin && \
    rm geckodriver-v$GECKODRIVER_VERSION-linux64.tar.gz

# Install Edge and MSEdgeDriver
RUN wget -q https://packages.microsoft.com/keys/microsoft.asc -O- | apt-key add - && \
    add-apt-repository "deb [arch=amd64] https://packages.microsoft.com/repos/edge stable main" && \
    apt-get update && \
    apt-get install -y microsoft-edge-stable && \
    wget -q "https://msedgedriver.azureedge.net/${EDGEDRIVER_VERSION}/edgedriver_linux64.zip" && \
    unzip edgedriver_linux64.zip -d /usr/local/bin/ && \
    rm edgedriver_linux64.zip

WORKDIR /robotframework

# Install Robot Framework and Selenium
RUN python3 -m venv ./venv
RUN ./venv/bin/pip install robotframework
RUN ./venv/bin/pip install robotframework-seleniumlibrary
RUN ./venv/bin/pip install robotframework-excellib

ENV PATH="/robotframework/venv/bin:$PATH"

COPY . /robotframework/venv/bin/tests
RUN mkdir /robotframework/venv/bin/results

# Define the entry point for the container
CMD ["robot"]
```

Kuvio 13 Dockerfile selaimet sekä Robot Framework

6.6 Gitlab integraatio

Kohdeorganisaation Gitlab-palvelimelle luotiin uusi runner, jotta siellä pystyttiin ajamaan Docker kontteja konttien sisällä. Runnerilla tarkoitetaan pientä ohjelmaa millä pystytään ajamaan CI/CD toimintoja. Runneria tarvitaan, koska lähtökohtaisesti Gitlabin toiminnot ajetaan konteissa. Tässä tapauksessa kuitenkin kontin sisällä tulee voida käyttää Dockeria Dockerimagen luontiin ja tähän tarvittiin erillinen runner. Tätä kutsutaan Docker-in-Docker termillä.

Aikaisemmissa katselmoinneissa olimme päättäneet, kuinka usein ajastettuja testejä tulee ajaa. Alkuun testejä tulisi ajaa joka toinen päivä, jotta saadaan mahdolliset virheet havaittua mahdollisimman nopeasti, eikä testejä vielä ole liian suurta määrää. Tulevaisuudessa testien aikataulutusta tulee kuitenkin muuttaa, koska testien määrä tulee lisääntymään ja näin testeihin kuluva aika myös kasvaa.



Kuvio 14 Gitlab aikataulu

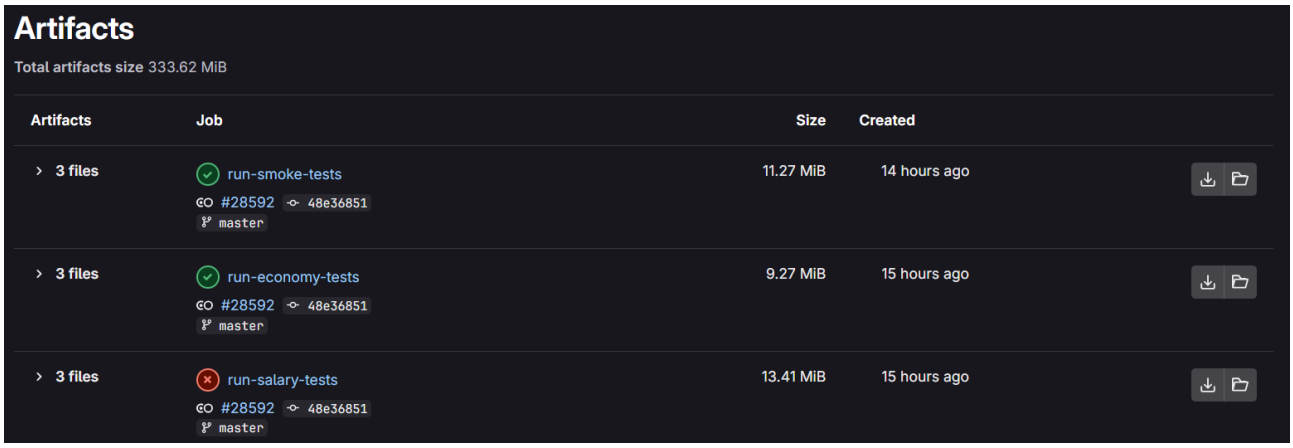
Automaatiotesteille loin oma yml tiedosto, robot_tests.yml joka eriytettiin päätasolla olevasta yml tiedostosta. Päätasolla olevassa gitlab-ci.yml-tiedostossa sijaitsi yksikkötestit sekä muut tarvittavat tarkistukset, jotka suoritettiin aina kun uutta koodia oltiin lisäämässä. Robot_tests.yml-tiedosto sisällytettiin gitlab-ci.yml tiedostoon helposti include käskyllä.

Robot Frameworkin yml tiedosto rakentui siten, että aluksi luotiin uusi Dockerimage Gitbalin Container rekisteriin, josta sitä voidaan kutsua tarvittaessa. Testejä ajetaan halutussa järjestyksessä. Ongelmia tässä tuotti se että, testit haluttiin menevän läpi usealla verkkoselaimella. Aluksi loin listan niistä, mitkä sisälsivät halutut selaimet. Tämän jälkeen tein silmukan, joka ajoi aloituskäskyn ottaen selaimen kerrallaan ja asettaen sen muuttujaan. Tämä toimi, mutta ei aivan halutulla tavalla.

Alkuun ongelmia tuotti se, että jos testit eivät onnistuneet ensimmäisellä selaimella Gitlab ei suorittanut testejä lopuilla selaimista. Yritin korjata tätä lisäämällä // *true* käskyn robot komennon perään. Tämä mahdollistaa sen, että yksittäinen virhe ei estä muiden testien ajoa. Kyseinen komento aiheutti kuitenkin sen, että Gitlab ilmoitti, että testit olivat menneet läpi, vaikka todellisuudessa näin ei ollut. Tämän seurauksena ilmoitusta mahdollisista virheistä ei myöskään koskaan muodostunut.

Lopullinen toimiva ratkaisu testien komentokäskyllä oli se, että jokaiselle selaimelle luotiin oma rivi. Eli ensiksi ajettiin Chrome ja sen jälkeen Edge. Näiden käskyjen perään lisättiin // *true* komento, joka mahdollisti sen, että testit jatkavat vaikka testin aikana tulisi virhe. Viimeisenä ajettiin testit Firefoxilla, johon tuota // *true* komentoa ei lisätty. Tämän tarkoituksena oli se, että kaikki selaimet tulivat testattu mutta jos ohjelmistossa on virheitä viimeistään Firefox aiheuttaa siitä ilmoituksen.

Testeistä muodostuu aina log-tiedosto, josta voi tarkastella testien tuloksia yksityiskohtaisesti. Koska testit suoritettiin ajastetusti yöllä, piti varmistaa, että log-tiedostot säilyisivät. Gitlabissa on ominaisuus Artifacts (Ks. Kuvio 15) johon pystyy arkistoimaan tuloksia halutuksi ajaksi. Asetin testeistä muodostuvalle log-tiedostolle sekä mahdollisille kuvakaappauksille säilytysajaksi 2 vuorokautta. Tulevaisuudessa kun testien ajot harvenevat tulee tätä aikaa vielä muuttaa.



Kuvio 15 Gitlab Artifacts

Ongelmia tuotti se, että Gitlab oli määritelty siten että se mahdollisti tässä kohden vain yhden runnerin ajon kerrallaan. Tämä aiheutti sen että jos testasin ajoja Gitlabissa joutuivat muut kehittäjät odottamaan että ajot päättyisivät ja runner vapautuisi muiden käyttöön. Ratkaisuna tähän Gitlabin annettiin suorittaa 4 runneria samaan aikaan. Tällä oli vaikutuksia myös testeihin. Nykyisessä robot_tests.yml tiedostossa ajetaan vain kahta stagea, jotka olivat Build sekä Test. Buildissa rakennetaan Dockerimage (Ks. Kuvio 16) ja Test ajaa kaikki testit (Ks. Kuvio 17).

Kun Gitab onnistui ajamaan useita runnereita samaan aikaan, yritti se ajaa myös useita testejä päällekkäin, koska näillä oli sama stage. Tämä aiheutti sen, että myös tietokantaa tyhjennettiin väärillä hetkillä, koska tietokanta tyhjennetään aina uuden test suiten alkaessa ja nyt kolme selainta suorittivat testejä samaan aikaan. Tähän ratkaisuna lisäsin *needs*: käskyn, jolla pystyin määrittelemään mitä jokin stage vaatii, että se voidaan ajaa. Käsky *when: always* joka varmisti sen että stage ajetaan vaikka edellinen stage olisi epäonnistunut.

```
build:
  stage: build
  image: docker:cli
  rules: [if: '$BUILD_TYPE == "robot"']
  services:
    - name: docker:dind
      command: ["--insecure-registry= *ip*"]
  tags: [docker]
  variables:
    DOCKER_TLS_CERTDIR: "" # Disable TLS
    DOCKER_HOST: # Use TCP socket
    DOCKER_IMAGE_NAME: $CI_REGISTRY_IMAGE:newest
  before_script:
    - docker info
  script:
    - echo $CI_REGISTRY_PASSWORD | docker login -u $CI_REGISTRY_USER $CI_REGISTRY --password-stdin
    - docker build . -t "$DOCKER_IMAGE_NAME" -f uitests/Dockerfile
    - docker push "$DOCKER_IMAGE_NAME"
```

Kuvio 16 robot_tests.yml build

```
run-salary-tests:
  image: $CI_REGISTRY_IMAGE
  rules: [if: '$BUILD_TYPE == "robot"']
  stage: test
  when: always
  tags: [docker]
  script:
    - robot -d robotResults/chrome --variable browser:chrome --variable selenium_user4_password:$selenium_user4_password
      --variable selenium_user3_password:$selenium_user3_password --variable selenium_rebuilder_password:$selenium_rebuilder_password --include salary . || true
    - robot -d robotResults/edge --variable browser:edge --variable selenium_user4_password:$selenium_user4_password
      --variable selenium_user3_password:$selenium_user3_password --variable selenium_rebuilder_password:$selenium_rebuilder_password --include salary . || true
    - robot -d robotResults/firefox --variable browser:firefox --variable selenium_user4_password:$selenium_user4_password
      --variable selenium_user3_password:$selenium_user3_password --variable selenium_rebuilder_password:$selenium_rebuilder_password --include salary .
  artifacts:
    when: always
    expire_in: 1 week
    paths:
      - robotResults
      - robotResults/*png
```

Kuvio 17 robot_tests.yml test

6.7 Testien optimointi

Vaikka testit paikallisesti omalla koneellani menivät läpi, niin testien ajo Gitlab-palvelimella aiheuttivat edelleen sen, että kaikki testit eivät toimineet täydellisesti. Osittain vaikutti se, että verkkoselaimet toimivat eritavoin. Toinen verkkosivu saattaa olla toista nopeampi, mutta myös netin luonnolliset viiveet aiheuttivat testeihin epäluotettavuutta.

Testien optimointi tarkoittaa päivittäistä seurantaa testeistä, analysoida tuloksia ja tehdä tarvittavia korjauksia testeihin. Tätä hankaloittaa se, että testit toimivat paikallisesti käynnistettynä mutta eivät Gitlab palvelimella. Testien manuaalinen ajaminen Gitlab palvelimella on hidasta työtä, koska muutokset pitää tehdä paikallisesti ja sen jälkeen siirtää Gitlab palvelimelle ja testata siellä.

6.8 Katselmoinnit

Toisin kuin testiautomaatio-ohjelmistojen kartoitusvaiheessa, testikäyttöönoton aikana välikatselmointeja käytiin kahden viikon välein. Näissä seurattiin aikataulussa pysymistä, käytettyä aikaa sekä testien toimivuutta. Nostin esille myös kohdattuja haasteita, joita pyrittiin ratkaisemaan. Tyypillisiä haasteita olivat erilaiset viiveet, joita aiheutti erilaiset tietokantakyselyt tai että sivu ei ollut latautunut vielä kokonaan.

Loppukatselmointia ei tämän työn aikana vielä ole pidetty. Projekti on aikataulutettu siten että se päättyy 31.12.2024 ja loppukatselmointi on sovittu pidettäväksi vielä ennen. Loppukatselmoinnista luodaan pöytäkirja ja samalla perustetaan työtiketti tuleville testiautomaatiotehtäville.

7 Tulokset

Työn tulokset olivat varsin kiitettäviä, sillä työ onnistui odotusten mukaisesti ja se vastasi myös asetettuihin tutkimuskysymyksiin.

Ilman aikaisempaa kokemusta Robot Frameworkilla on mahdollista tuottaa yksinkertaisia testejä varsin lyhyen tutustumisen jälkeen. Ohjelmointitaidot eivät siten ole pakollisia, mutta monimutkaisimpiin tapauksiin voidaan vaatia jonkinlaista ymmärrystä ohjelmoinnin logiikasta. Koska Robot Framework on avoimenlähdekoodin ohjelmisto, ei sen käyttö tuota organisaatiolla kuluja. Tästä tulee huomattavia säästöjä organisaatiolle, kun verrataan aiemmin käytössä olleeseen TestCompleteen, jossa yhden lisenssin käyttö maksoi jo tuhansia euroja.

Testien luotettavuutta tulevaisuudessa voidaan kuitenkin varmistaa ainoastaan siten, että testejä ylläpidetään. Siksi organisaatiossa nimettiin vastuuhenkilö, jonka toimenkuvaan kuuluu testien seuranta ja niiden ylläpito. Lisäksi vastuuta testien ylläpidosta jaettiin myös kehittäjille, koska heillä on paras ymmärrys omien tuotteidensa toiminnoista.

Nyt organisaatiolla on valmis ratkaisu, jota voidaan kehittä myös jatkossa. Myös uuden kirjaston käyttöönotto on helpompaa ja nopeampaa kuin täysin uuden ohjelmiston integrointi, mikäli yrityksen teknologiat muuttuvat. Toteutettu työ jäi organisaation käyttöön ja on aktiivisessa käytössä edelleen. Samaa työtä tullaan soveltamaan myös konsernitasoisesti, joka puoltaa olettamaa siitä, että työ on ollut paitsi tarpeellinen ja ajankohtainen, myös onnistunut ja konsernin laatukriteerit täyttävä.

8 Jatkokehitys

Suurimpana jatkokehityksenä testiautomaatiolle on varmasti uusien testien tekeminen. Organisaatiolla on useita aliohjelmia ohjelmistossa ja vasta muutamista on regressiotestejä sekä savutestejä tehty. Organisaatiolla on kuitenkin paljon valmiita testitapauksia kerättynä useissa Excel-tiedostoissa, joiden mukaan myös manuaaliset testit on suoritettu. Näitä hieman muuttamalla voidaan luoda samat testit myös automaatiolla, mietittäväksi jää ainoastaan se, että ovatko kaikki käyttötapaustestit tarpeellista suorittaa testiautomaatiolla.

Tulevaisuudessa käyttöliittymätestejä pitää tehdä myös muille sovelluksille sekä backend-testit, joissa testataan suoraan rajapintoja ilman käyttöliittymää. Myös erilaiset tietokantatestit voisivat olla hyödyllisiä testaten tietokannan eheyttä. Tulevaisuudessa tulee tutkia myös miten rasitustestejä voi tuottaa Robot Frameworkilla joka on suunniteltu hyväksyntätestaukseen.

Docker-tiedostossa olevien webdriverien versiointi olisi syytä myös automatisoida. Tällä hetkellä versionumerot tulee muuttaa manuaalisesti Docker-tiedostoon. Ongelmana tässä on se, että ajureiden verkko-osoitteet ovat versionumerokohtaisia, eikä niitä pysty käyttämään samoin, kuin itse verkkoselaimien asennustiedostojen osoitteita. Näistä tulisi siis rakentaa jonkinlainen vertailu Docker-tiedostoon. Viimeisimmän vakaan verkkoselaimen voi noutaa korvaamalla osoitteen versionumero sanalla stable.

Projektin edetessä harkitsin, jos toteuttaisin muiden testien joukkoon testitapauksen, joka tarkistaisi uusimmat ajurit ja ilmoittaisi, jos uudempi ajuri on saatavilla. Tälle ei kuitenkaan ollut vielä välttämätöntä tarvetta organisaatiossa, joten se jätettiin vielä toteuttamatta, jotta riittävät resurssit projektinajaksi voitiin taata aikataulun mukaisesti. Tämä voisikin siis olla varteenotettava jatkokehitystyö.

9 Pohdinta

Projektin alkuasetelmana toimi se, että organisaation aikaisempana testityökaluna toimi TestComplete. Lähtökohtina minulle kerrottiin, että tämä ei enää toimi ja kyseisen ohjelmiston päivitykset rikkovat sitä entisestään. En siis alkujaankaan saanut tietoa sen enempää, miksei se toimi.

Näin jälkikäteen mietittyäni asiaa en usko, että toimimattomuuden aiheutti TestCompleten päivitykset vaan resurssien puute. Tällä tarkoitan sitä, että tähän nimitetyllä henkilöllä oli useita toimenkuvia, eikä tarvittavaa aikaa löytynyt testien ylläpitämiseen. Tarve ohjelmiston uudelleen kartoitukseen oli joka tapauksessa ajankohtainen. Organisaatio halusi ottaa testiautomaation jatkuvaan seurantaan ja rakentaa toimivan ratkaisun, jonka tuloksiin voidaan luottaa.

Edellä mainitun ratkaisun rakentaminen on kuitenkin iso urakka, kun organisaation tuotteessa on noin 15 alisovellusta, joita hoitavat eri kehittäjät. TestCompletella oli vain 1 lisenssi käytettävissä

nykyisellä sopimuksella ja testien tuottaminen kaikkiin sovelluksiin olisi ollut todella haastavaa, jos kaikki kehittäjät olisivat tehneet testit omille osa-alueilleen niin kuin oli suunniteltu. Toki TestCompletelta olisi ollut mahdollisuutta uusia sopimus useammalla lisenssillä mutta hinta olisi noussut huomattavasti.

Myös automatisoitujen testien ajaminen Gitlabista oli ajatuksia herättävä. Toinen vaihtoehto olisi ollut Jenkinss. Olisiko Jenkinssillä ollut tarjota jotain mitä Gitlabilla ei ole? Tähän olisi voinut perehtyä enemmän, mutta organisaation tahtotilana oli ottaa mahdollisimman vähän palveluita käyttöön ja pyrkiä keskittämään ajot Gitlabiin jossa muutkin testit ajetaan. Tämä on kuitenkin mahdollista testata myös myöhemminkin ilman suurempia ponnisteluita.

Itse toteutettu projekti oli erittäin opettavainen sekä todella mielenkiintoinen koko matkan. Minulla oli jo jonkin verran aikaisempaa kokemusta testiautomaation käytöstä, mutta en ollut aikaisemmin toteuttanut näin laaja-alaista projektia. Gitlab sekä Docker olivat minulle tuttuja aikaisemmista töistäni, mutta tässä pääsin syventymään näihin vielä lisää ja siihen, kuinka näitä voidaan käyttää yhdessä jatkuvassa integraatiossa.

Olin tyytyväinen ratkaisuihini mitä tein ja sain aikaan toimivan kokonaisuuden. Toki joitain asioita olisi voinut tehdä toisin kuten miettiä oliko Ubuntu paras imagepohja Docker-tiedostoon tai olisiko Python ollut helpompi toteuttaa. Tämä ei kuitenkaan olisi tuonut mitään lisäarvoa työlleni, koska lopputulos on aivan samanlainen.

Projektin tarkoitus oli tuottaa organisaatiolle toimiva ratkaisu missä he voivat ajaa testiautomaatiota Gitlab palvelussa ajastetusti. Projekti myös valmistui näiltä osin halutulla tavalla ja suunnitellussa aikataulussa. Toki kehitettävää on vielä, kuten jatkokehityskappaleessa mainitsin mutta pohja on jo toimiva ja se on tällä hetkellä myös käytössä. Organisaatiolla on nyt mahdollisuus saada luotettavaa tietoa ohjelmiston eheydestä. Tehtävää on vielä, kun testit tulee optimoida jokaiselle selaimelle mutta jo nyt regressiotestit pystytään ajamaan kahdella selaimella ja varmistamaan, että tieto mahdollisista vioista saadaan tietoon nopeasti. Savutestit, jotka varmistavat kriittiset näkymät ja toiminnot kaikissa ohjelmissa toimii ongelmitta kaikilla selaimilla.

Suurin aikaansaannos mitä tällä projektilla olen saavuttanut, on se, että nyt myös konsernitasolla ymmärretään testiautomaation hyödyt sekä sen välttämättömyys kilpailutekijänä. Se, että pääsen jatkossa jakamaan oppejani myös muille konsernimme yksiköille sekä auttamaan heitä pystyttämään omat ratkaisunsa takaa myös oman osaamiseni kehittymisen ja on siten ehdottomasti askel parempaan monellakin tapaa.

Lähteet

Best Automation Testing Tool for 2024. 2024. BrowserStack. Viitattu 8.12.2024. https://www.browserstack.com/guide/best-automation-testing-tools

Bisht, S. 2013. Robot framework test automation. 1.p. Birmingham: Packt Publishing

Helala, H. 2024. Testiautomaatio ja testausmuotojen moninaisuus. ATR Soft Oy:n julkaisema blogi. Julkaistu 23.8.2024. Viitattu 3.11.2024. https://www.atrsoft.com/teknologiat/testiautomaatio-ja-testausmuotojen-moninaisuus

Kasurinen, J, 2013. Ohjelmistotestauksen käsikirja. 1. p. Jyväskylä: Docendo

Merode, H. v. & Merode, H. v. 2023. *Continuous Integration (CI) and Continuous Delivery (CD): A Practical Guide to Designing and Developing Pipelines*. 1.p. Berkeley, CA: Apress.

Mitä ovat kontit ja Docker? 2022. Ouro Solutions Oy. Julkaistu 11.04.2022. Viitattu 4.11.2024. https://ouro.fi/mita-ovat-kontit-ja-docker/

Rytkönen, M & Kakkonen, K. 2023. ACT 2 LEAD: ohjelmistotestauksen johtamisen käsikirja. 1.p. Tuusula: Ketterät Kirjat Oy

Spillner, A. & Linz, T. 2021. *Software testing foundations*. 5.p. Heidelberg: Dpunkt.verlag GmbH.

The Top 20 Test Automation Tools of 2025. 2024. Leapwork. Viitattu 8.12.2024. https://www.leapwork.com/blog/top-20-test-automation-tools

Thompson, G., Morgan, P., Samaroo, A., Kurowski, J., Williams, P., Salmon, M. & Weymouth, P. 2024. *Software Testing: An ISTQB-BCS Certified Tester Foundation Level Guide (CTFL V4. 0). 5.p*. Swindon: BCS Learning & Development Limited.

Toiminnallinen testaus – Miksi se on avainasemassa sovelluskehityksessä? 2023. VALA. Julkaistu 21.08.2023. Viitattu 06.10.2024. https://www.valagroup.com/fi/blogi/toiminnallinen-testaus-miksi-se-on-avainasemassa-sovelluskehityksessa/

Liitteet

Liite 1. Testiautomaatio kartoituksen tyhjä pistetaulukko

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		TESTCAFE		RANOREX	SELENIUM	UIPATH	TESTSIGMA	PLAYWRIGHT	CYPRESS	TRICENTIS TOSCA	FUNCTIONIZE	LEAPWORK	ROBOT FRAMEWORK
2	Elinkaari Milloin julkaistu? Ennuste elinkaaresta?												
3	Yhteistyökumppanit. Millaisia firmoja/organisaatioita on tukemassa kehitystä.												
4	Helppokäyttöisyys Testien nauhoitusominaisuus, koodin selkeys ym.												
5	Yhteisö Keskustelufoorumien aktiivisuus ja asiapitoisuus.												
6	Dokumentaation kattavuus.												
7	Opetus- ja koulutusvideot. Miten paljon löytyy esim Youtubesta.												
8	Hinnoittelumalli ja lisenssipolitiikka sekä käyttömaksut ammattimaisen käytön osalta. Ekosysteemin maksut mahdollisista kirjastoista.												
9	Tukipalveluiden laajuus ja saatavuus sekä kustannukset												
10	Vakaus												
11	PISTEET YHTEENSÄ												

Liite 2. Testiautomaatio kartoituksen täytetty pistetaulukko

	A	B	C	D	E	F	G	H	I	J	K	L	M
1		TESTCAFE	TESPROJECT	RANOREX	SELENIUM	UIPATH	TESTSIGMA	PLAYWRIGHT	CYPRESS	TRICENTIS TOSCA	FUNCTIONIZE	LEAPWORK	ROBOT FRAMEWORK
2	Elinkaari Milloin julkaistu? Ennuste elinkaaresta?	4		4	4	4	2	4	4	4	4	3	4
3	Yhteistyökumppanit. Millaisia firmoja/organisaatioita on tukemassa kehitystä.	4		3	4	4	4	4	4	4	3	3	4
4	Helppokäyttöisyys Testien nauhoitusominaisuus, koodin selkeys ym.	5		2	4	3	4	4	4	2	4	5	5
5	Yhteisö Keskustelufoorumien aktiivisuus ja asiapitoisuus.	3		4	4	4	2	4	4	3	3	2	4
6	Dokumentaation kattavuus.	2		3	5	3	3	4	4	3	4	3	5
7	Opetus- ja koulutusvideot. Miten paljon löytyy esim Youtubesta.	3		4	4	4	3	4	4	4	3	3	4
8	Hinnoittelumalli ja lisenssipolitiikka sekä käyttömaksut ammattimaisen käytön osalta. Ekosysteemin maksut mahdollisista kirjastoista.	3		2	5	1	3	5	3	2	2	1	5
9	Tukipalveluiden laajuus ja saatavuus sekä kustannukset	3		3	2	2	3	2	3	2	2	3	2
10	Vakaus	4		2	3	3	3	4	4	3	4	3	4
11	PISTEET YHTEENSÄ	3.4	Ei tukea enään	3	3.888888889	3.111111111	3	3.888888889	3.777777778	3	3.222222222	2.888888889	4.111111111