OULUN AMMATTIKORKEAKOULU

Risto Jääskelä

TESTIAUTOMAATIOJÄRJESTELMÄN KEHITYSTYÖ

TESTIAUTOMAATIOJÄRJESTELMÄN KEHITYSTYÖ

Risto Jääskelä
Opinnäytetyö
Syksy 2021
Tietotekniikan tutkinto-ohjelma
Oulun ammattikorkeakoulu

TIIVISTELMÄ

Oulun ammattikorkeakoulu
Tietotekniikan tutkinto-ohjelma, ohjelmistokehitys

Tekijä: Risto Jääskelä
Opinnäytetyön nimi: Testiautomaatiojärjestelmän kehitystyö
Työn ohjaaja: Kari Jyrkkä
Työn valmistumislukukausi ja -vuosi: Syksy 2021 Sivumäärä: 22

Symbio on vuonna 1997 perustettu osakeyhtiö, joka toimii suurimmaksi osin IT-konsultoinnin sekä muiden IT-palveluiden parissa. Sen lisäksi Symbiolla on myös erinäisiä ohjelmistonkehitysprojekteja, kuten Elysian, jonka varianttiin myös tämä opinnäytetyö perustuu.

Opinnäytetyön tavoitteena oli tuottaa Symbiolla toimiva testiautomaatioratkaisu yrityksen asiakasprojektiin. Symbiolla oli ennestään aiempiin asiakasprojekteihin vastaavanlaisia järjestelmiä. Opinnäytetyössä keskityttiin ymmärtämään näiden olemassa olevien järjestelmien arkkitehtuuri, konfiguraatiot ja vaatimukset testiautomaation toteuttamiseen uudelle laitteelle.

Tärkeimpiä järjestelmän komponentteja olivat testattava laite, testiajoa suorittava tietokone sekä Jenkins-työkalu. Ratkaisuun oli liitettynä myös Testrail ja GitLab, mutta näiden osalta suurin osa työstä oli jo tehty. Järjestelmään oli liitettynä myös useampia testaustietokoneita sekä käännöskoneita, jotka toimivat niihin yhdistetyn päätietokoneen kautta.

Opinnäytetyön lopputuloksena syntyi toimiva testiautomaatioratkaisu laitteen, Jenkinsin, testaustietokoneiden sekä muiden komponenttien välille. Testaustietokone suorittaa automaattisesti testiajot laitteella, kun Jenkins vastaanottaa uuden version laitteen testattavasta ohjelmasta. Siitä huolimatta järjestelmässä on myös asioita, joita voisi jatkokehittää. Muun muassa uuden version päivittäminen laitteelle on hyvin epävakaata. Tätäkin ongelmaa avataan paremmin pohdintaosiossa.

Asiasanat: testiautomaatio, jatkuva integraatio, ohjelmistokehitys

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Information Technology, Option of Software Development

---

Author(s): Risto Jääskelä
Title of thesis: Development of Test Automation System
Supervisor(s): Kari Jyrkkä
Term and year when the thesis was submitted: Fall 2021
Number of pages: 22

Symbio is a limited company established in 1997, which mostly provides IT-consultation and among other IT services. Besides that, Symbio works on software development projects, such as Elysian on which this thesis is based as well.

The objective of this thesis was to produce a working test automation solution for Symbio's customer project. Previous similar solutions exist at the Symbio already, so the objective of this thesis mainly focused on getting familiar with the existing setups, their architecture, and the different configurations all the different components require.

The solution includes various components, some physical, some websites. Most important of these were the machine which runs the tested software, the PC linked to the machine, which runs the test suites, and Jenkins, website which communicates between all the components in the system, aside from the machine itself. Solution included also GitLab and Testrail, but the work on these components was minimal. Aside the testing PCs, there are various build PCs linked to Jenkins as well, and the master PC which assigns the jobs to the testing as well as build PCs.

At the conclusion of this thesis, a working solution was created between Jenkins, testing PC, as well as all the other required components. Testing PC automatically runs the test suite on the machine when a new version of the software is built and uploaded to Jenkins. Regardless, there remains things to be improved on the system, for example the updating process of the new version is very unstable. This problem will be better explained in the conclusion chapter.

Keywords: test automation, continuous integration, software development

SISÄLLYS

1 JOHDANTO

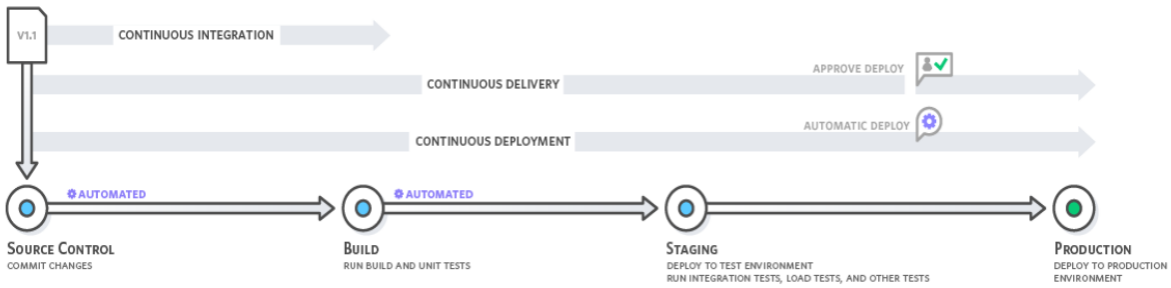
Työskentely Symbiolla aloitettiin tammikuussa 2020, jolloin tavoitteena oli toimia alusta asti testiautomaatiosta vastaavan henkilön työparina. Vielä siihen aikaan tehtäviin pääasiallisesti kuului uusien testiskriptien kirjoittaminen sitä mukaa kuin niitä tarvittiin. Noin vuosi eteenpäin siitä testiautomaatiosta pääasiallisesti vastaava henkilö siirtyi toisiin tehtäviin. Täten työkuva muuttui huomattavasti ja mukana tuli paljon vastuuta sekä uusia tehtäviä, mutta myös asioita, joita vuoden aikana oli opittu. Kesällä 2021 Symbio vastaanotti uuden asiakasprojektin, jolle tarvitsi myös testiautomaatioratkaisun, joten seuraavaksi tehtäväksi ja samalla opinnäytetyön aiheeksi muuttui tuottaa kyseinen järjestelmä. Tämä oli luonnollisesti merkityksellinen seuraava tehtävä, koska uran alussa avustus testiautomaatiojärjestelmän päivittämisessä oli oleellisin tehtävä, sen jälkeen vastuulle siirtyi saman järjestelmän ylläpitäminen, ja seuraava looginen askel oli tuottaa järjestelmä alusta asti itse.

Työllä oli vain kaksi tärkeää tavoitetta, jotka täytyi saavuttaa: toimivan testiautomaatioratkaisun rakentaminen ja sen yhdistäminen olemassa olevaan Jenkins-järjestelmään. Symbiolla oli toiminnassa jo ennestään toimivia vastaavia järjestelmiä, ja opinnäytetyön aihe oli eräs versio jo olemassa olevista ratkaisuista, joten monet asiat olivat suoraan kopioitavissa niistä. Työn vaikeus olikin lähinnä siinä, että täytyi pystyä ymmärtämään järjestelmä ja kaikki sen toimintaan tarvittavat konfiguraatiot. Symbiolla oli kuitenkin tarjolla muutamia dokumentteja aiheesta, mutta iso osa tutkimuksesta tapahtui tiedostoja ja järjestelmässä olevia sivustoja tutkien.

Symbio on osakeyhtiö, joka perustettiin vuonna 1997. Sen kotipaikkana toimii Espoo, mutta toimistoja löytyy myös Oulusta sekä Tampereelta. Symbio perustettiin vastaamaan kasvavaan IT-konsultoinnin tarpeeseen (1). Sen lisäksi Symbiolla on myös erinäisiä omiakin ohjelmistotuotteita, kuten Elysian, jonka parissa itsekin työskentelin tämän opinnäytetyönikin parissa. Elysian on framework-ohjelmistotuote autojen keskikonsoleihin, joka on muokattavissa asiakkaan toiveiden mukaan. Työskentely keskittyi testaamaan yhtä näistä Elysianin versioista, jonka asiakas on tilannut.

2 JATKUVA INTEGRAATIO

Yksi suurimmista vaaroista ohjelmistonkehityksessä on, kun suuremmassa tiimissä ohjelmoijat tahtovat yhtäaikaisesti tehdä muutoksia koodiin. Tämä käytännössä vaatisi työntekijän tai useamman, joiden työ on erikseen omistautunut organisoimaan näitä muutoksia sekä varmistamaan, että muutokset ovat yhteensopivia toistensa kanssa. Tämä taas aiheuttaa yhtiölle suurempia kuluja, sekä tuotteen valmistumiselle on vaikeampi asettaa aikarajaa, kun muutosten manuaalisesti integroiminen on hidasta ja virhealtista (2). Jatkuva integraatio on kehitetty korjaamaan juuri tätä ongelmaa. Jatkuvaan integraatioon liittyy useampia vaiheita, jotka näkyvät kuvassa 1.



KUVA 1. Jatkuvan integraation vaiheet (3)

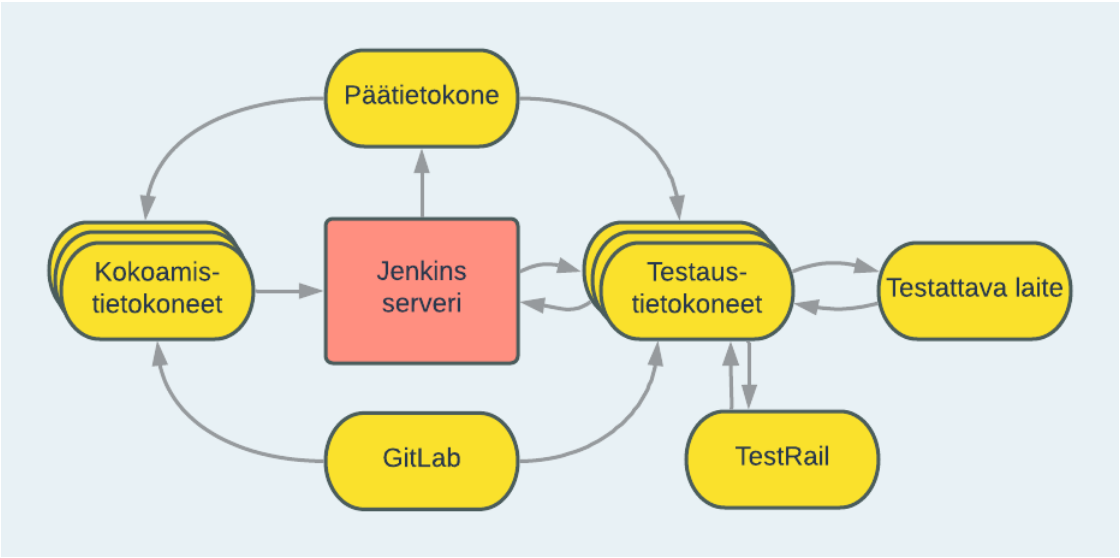
Yritys tuottaa listan toivotuista ominaisuuksista ohjelmistossa, joista ohjelmistonkehittäjille määrätään tehtäviä sitä mukaa kuin niitä valmistuu. Kun ohjelmistonkehittäjällä on oma tehtävänsä, hän voi huoletta tehdä muutoksia tähän. Nämä muutokset, joita ohjelmistonkehittäjät tuottavat, kulkevat yleensä jonkun toisen ohjelmoijan kautta, joka hyväksyttää koodin, ja tämän jälkeen lähetetään eteenpäin jatkuvan integraation automaation vastuulle. Tämä on ensimmäinen askel, joka kuvassa 1 on esitelty Source Control -pallon kohdalla. Automaatio yhdistää muutoksen ohjelmiston koodikantaan. Tämä ei välttämättä kuitenkaan ole valmiin tuotteen koodikanta. Monesti ohjelmistonkehitykselle on erikseen oma koodikantansa, joka tietyin väliajoin yhteisen katselmoinnin jälkeen yhdistetään niin sanottuun mestari-koodikantaan tai main branchiin yleisemmin.

Jatkuvan integroinnin jälkeen automaatioon voidaan lisätä myös jatkuva toimitus, joka tarkoittaa muutoksen automaattista kokoamista tiedostoiksi, joilla vasta koottu ohjelmistoversio voidaan asentaa. Tämä on kuvassa 1 pallo, joka on nimetty Build. Näitä asennustiedostoja kutsutaan artefakteiksi. Tilanteen mukaan nämä artefaktit voidaan automatisoida menemään testaukseen tai

käyttäjille. Tässä opinnäytetyössä ohjelmisto tulee testaukseen, mutta esimerkiksi jo julkaistujen mobiilisovellusten kanssa uusi ohjelmistoversio saatetaan haluta automatisoida menemään suoraan sovelluskauppaan ladattavaksi.

3 TESTAUSJÄRJESTELMÄ

Projektin toteuttamiseen kuului useita vaiheita ja tässä luvussa nuo vaiheet ja ohjelmiston osat käydään läpi. Erinäiset komponentit, jotka tässä järjestelmässä ovat, ja niiden väliset yhteydet on kuvattu kuvassa 2.



KUVA 2. Testausjärjestelmän keskeiset komponentit

3.1 Testausjärjestelmän komponentit

Kuten kuvasta 2 on nähtävissä, järjestelmän keskipisteenä toimii Jenkins. Jenkins on sivusto, joka on luotu auttamaan käyttäjiä jatkuvassa integraatiossa ja ohjelmiston tuottamisen automaatiossa. Jenkinsiä ajetaan Symbiolla yrityksen omalla webbiserverillään, jonne pääsy ei sinänsä vaadi minkäänlaista kirjautumista, mutta vaatii yhteyden Symbion omaan intranettiin. Symbiolla Jenkins auttaa niin uusien ohjelmiston versioiden kokoamisessa kuin myös niiden testaamisessa.

Päätietokone on tietokone, joka on liitettynä kaikkiin järjestelmässä oleviin tietokoneisiin, niin testaukseen kuin ohjelmistoversioiden kokoamiseen tarkoitettuihin, ja delegoi näille niiden tehtävät. Päätietokone kuuntelee Jenkinsiä, joka pyytää päätietokonetta varaamaan vapaan tietokoneen joko ohjelmiston kokoamista tai testausta varten. Näitä testaus- ja kokoamistietokoneita on useampia järjestelmässä.

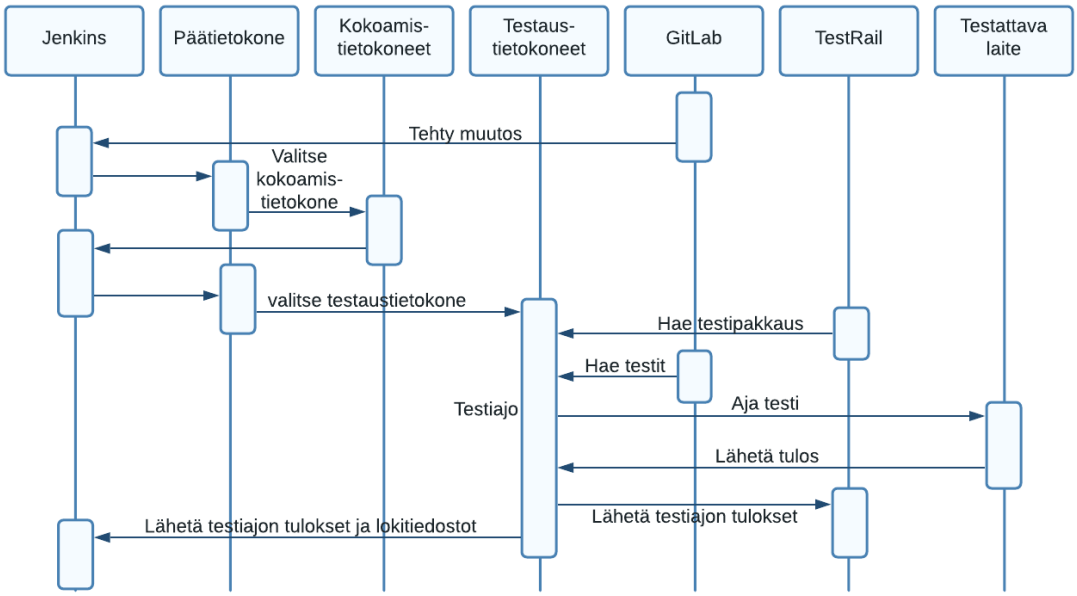
GitLab on GitHubia muistuttava pilvipalvelu ohjelmistodokumenttien tallennukseen, mutta sen suurimmat erot ovat, että se sallii rajattoman yksityisen repositorion, sekä sillä on avoin lähdekoodi haluttuja muokkauksia varten (4). Järjestelmässä käytetään kahta eri GitLab-repositoriota: Elysiania varten olevaa, joka on yksityinen Symbion työtekijöille, sekä asiakasprojektille tehtyä, joka on nähtävissä sekä Symbiolle että asiakkaalle. Näistä kahdesta Elysianille olevaa repositoriota ajetaan Symbion webbiserverillä, kun taas asiakasprojektille tehty on GitLabin omalla serverillä. Opinnäytetyössä käytettiin pääasiassa asiakasprojektille suunnattua repositoriota, sillä se sisältää itse ohjelmiston koodit sekä testiskriptit.

Testrail on sivusto, jonne käyttäjä voi luoda projekteja ja niiden sisälle testauspaketteja. Nämä paketit sisältävät yksittäisiä testitapauksia, joita laitteen tulisi kyetä suorittaa. Näihin tapauksiin on kirjattu jokainen toiminto, joita tapauksen suorittaminen vaatii sekä toivottu tulos. Esimerkiksi jos painetaan HOME-painiketta laitteessa, odotetaan aloitusnäkymän aukeavan. Toinen tärkeä ominaisuus, mitä Testrailissa suoritetaan, on testitulosten vastaanottaminen, tallentaminen ja niiden helppo tulkinta.

Viimeinen komponentti järjestelmässä on testattava laite. Koska laite on asiakkaan tuote, sitä ei tässä opinnäytetyössä voi kovin yksityiskohtaisesti kuvailla. Laitteella ajetaan testiskriptejä, jotka simuloivat painikkeiden kosketuksia, vetoja, ja muita ominaisuuksia, mitä laitteen käyttöliittymällä voi tehdä. Jokainen näistä skripteistä on testitapaus, joita asiakas ja Symbio ovat yhdessä suunnitelleet laitteella pystyttävän tekemään.

3.2 Testausjärjestelmän vaiheet

Aivan ensimmäinen askel uudella testattavalle ohjelmistoversiolla on GitLabiin tehty muutos, kuten nähtävissä kuvassa 3. Tämän muutoksen kommenttiosioon ohjelmistonkehittäjä voi kirjoittaa ennalta määrätyn komennon. Komento kertoo Jenkinsille, että tämä tehty muutos tahdotaan hakea GitLabista ja koota sen kanssa uusi ohjelmistoversio. Jenkins vastaanottaa tämän pyynnön ja ilmoittaa tästä eteenpäin Päätietokoneelle.



KUVA 3. Järjestelmän vaiheet

Päätietokone etsii siihen yhdistettyjen ohjelmistonkokoamistietokoneiden joukosta sellaisen, joka on toiminnassa ja ei ole suorittamassa kokoamista jo. Tämä kone vastaanottaa komennon, hakee tarvittavat tiedostot GitLabista, ja suorittaa koneella kokoamisprosessin. Kun uusi versio on tuotettu, lähettää se siitä tiedostot Jenkinsiin. Tarkemmin ottaen, mitä Jenkins kokoamistietokoneelta vastaanottaa ei ole ohjelmistoa, joka itse laitteeseen menee, vaan artefaktit, joilla laitteen päivittäminen suoritetaan. Niiden lisäksi Jenkins vastaanottaa myös muita tiedostoja, kuten lokitiedostoja ja GitLabiin tulleet muutokset uuden ja edellisen version välillä.

Tämän jälkeen uuteen versioon liitetty Jenkins-tehtävä aktivoituu automaattisesti. Jenkins sallii tehtävän käynnistymisen toisen loputtua, joka mahdollistaa helpon automaation erinäisten tehtävien välillä. Mikä tehtävä Jenkinsissä käynnistyy, riippuu siitä, mikä ohjelmistoversio on juuri koottu. Symbiolla ja tämän projektin yhteydessä näitä mahdollisuuksia olivat release, nightly sekä regression. Päätietokone määrää testaukselle samalla tavoin tietokoneen, joka suorittaa ohjelmiston testaamisen laitteella. Suurin osa seuraavista asioista tapahtuu itse automaatiotietokoneella. Tämä prosessi itsessään on, mihin opinnäytetyön parissa käytettiin enimmäkseen aikaa.

Testaustietokone sallii Jenkinsin ajaa shellissä komentoja omalla Jenkins-käyttäjällään, joiden avulla seuraavat askeleet ovat automaation kautta mahdollisia. Ensimmäiseksi skripti hakee Testrailista listan testeistä, joita laitteella tahdotaan ajaa. Kun testaustietokoneelle on haettu toivottu lista testitapauksista hakee seuraava komento vastaavat testiskriptit GitLabista.

Seuraava askel on testaustietokoneella vastaanottaa Jenkins-serveriltä halutut artefaktit ohjelmistoversion päivittämistä varten. Laitteessa on liitettynä USB-tikku näitä varten. Toivotut päivitystiedostot lähetetään tikulle, edelliset poistetaan, ja laitteen päivittäminen tapahtuu ajamalla skripti, jolle annetaan päivitystiedosto parametriksi. Laitteen päivittäminen vie muutaman minuutin, mutta skripti seuraa laitteen tilannetta, ja kun se saa viestin, että UI on auennut takaisin laitteeseen, aloittaa Jenkins testiajon. Testaustietokone käy yksi kerrallaan satunnaisessa järjestyksessä kaikki vaaditut testitapaukset. Koko testiajon ajan kokoaa sitä ohjaava skripti lokitiedostoa kaikista toiminnoista, mitä testitapauksissa tapahtuu ja niiden tuloksista. Testiajon loputtua tämä lokitiedosto lähetetään takaisin Jenkinsille, joka vastaanottaa myös tiedon siitä, miten testit yhtäläisesti menivät. Jos jostain syystä testiajo ei lähde käyntiin, ylittää annetun aikarajan tai keskeytyy, osoittaa Jenkins testiajon epäonnistuneeksi. Jos testiajo on suoritettu loppuun, mutta yksi tai useampi testitapaus on epäonnistunut, testiajo merkataan epävakaaksi. Jos testitapaus on suoritettu loppuun sekä mikään testiajo ei ole epäonnistunut, merkataan testiajo onnistuneeksi. Tämä kuitenkin on harvinaista, kun testauksessa on keskeneräistä ohjelmistoa.

4 JÄRJESTELMÄN TOTEUTUS

4.1 Yleiskatsaus tehtäviin asioihin

Aivan ensimmäinen asia, joka projektin kannalta täytyi tehdä, oli saada yhteys laitteeseen testaustietokoneen kautta ja todeta, että laite on toimintakunnossa. Testaustietokone oli alussa täysin tyhjä, joten sen valmisteleminen lähti liikkeelle käyttöjärjestelmän asentamisesta. Tässä tapauksessa koneelle asennettiin Ubuntu 18.04 -versio. Se on toimivaksi ja hyväksi todettu muilla testaustietokoneilla, mutta käytettävät ohjelmistot ja palvelut tukevat myös muitakin käyttöjärjestelmiä. Kun koneelle on asennettu haluttu käyttöjärjestelmä, voi laitteen asentaa kiinni koneeseen. Tämä arkkitehtuuri selitetään luvussa 4.2 paremmin kuvan kera. Kun testattava laite on yhdistetty testaustietokoneeseen, voi yhteyttä kokeilla SSH-yhteydellä koneen terminaalista.

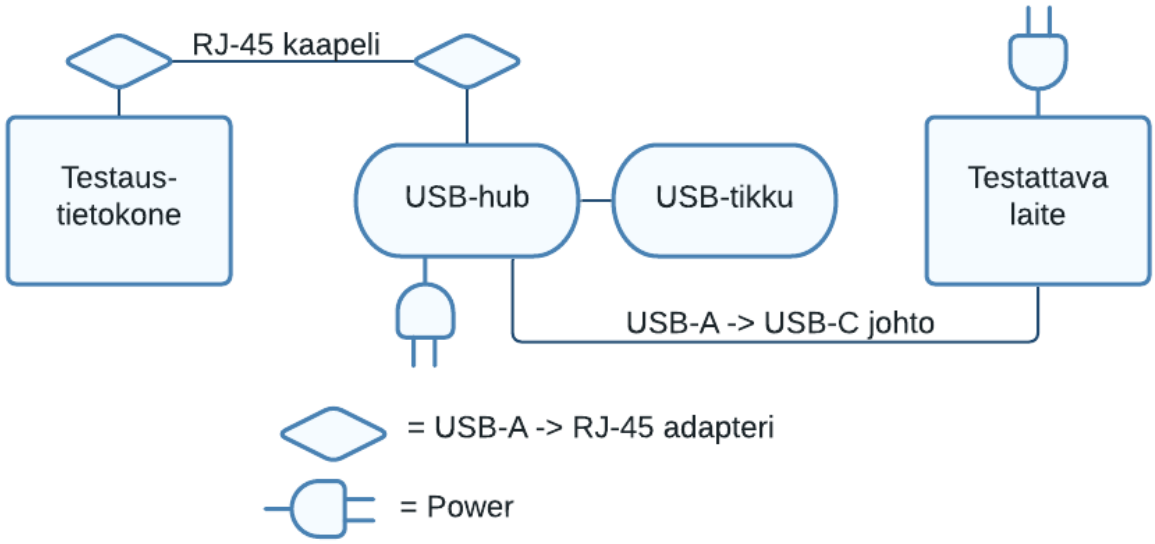
Jenkins, TestRail, sekä GitLab ovat opinnäytetyön kannalta lähes täysin kunnossa. Siitä huolimatta näissäkin kuitenkin oli GitLabia lukuun ottamatta tehtävä muutoksia. GitLabin tapauksessa päätös oli, että uuden repositorion luominen on enemmän vaivaa kuin mitä siitä saavutettava hyöty. GitLabista haettavat testiskriptit ovat jo olemassa, koska projekti, josta opinnäytetyö tehtiin, on eräs versio toisesta olemassa olevasta projektista. Myös TestRail sisältää kaikki tarvittavat testitapaukset jo. Kaikki mitä TestRailissa täytyy tehdä, on luoda uusi projekti ja kopioida sinne jo olemassa olevat testitapaukset. Jenkinsissä sen sijaan on enemmän lisättävää. Testiajoja varten Symbion asiakasprojektiin on lisättävä tehtävät uutta versiota varten. Nämäkin kuitenkin ovat lähes identtisiä jo olemassa olevalle projektille tehdyille, joten niistä saatavat konfiguraatiot ovat hyödyllisiä työn kannalta. Tehtävien lisäksi, Jenkinsiin täytyy myös lisätä testaustietokone, joka on liitettynä uuteen testattavaan laitteeseen. Kuten muutkin testaustietokoneet, tämäkin täytyy myös liittää päätietokoneeseen. Tämä tapahtuu täysin Jenkinsin ja testaustietokoneella ajettavan agentin puolesta, eikä päätietokoneen sekä testaustietokoneen välille tarvita minkäänlaista fyysistä liitäntää.

Ennen kuin uutta testaustietokonetta liitetään Jenkins-järjestelmään, täytyy sille kuitenkin asentaa laitteen testausta varten tarvittava testausympäristö. Testaustietokoneelle on asennettava kaikki tarvittavat kirjastot ja ohjelmistot, haettavava Symbion GitLabin testiautomaatio repositoriosta tarvittavat tiedostot, sekä tehtävä näihin kaikkiin tarvittavat konfiguraatiot. Kun nämä asiat on hoidettu,

yhteys laitteeseen on kunnossa, ja laitteella pystytään ajamaan testitapauksia manuaalisesti terminaalin kautta. Viimeinen tehtävä asia on ajaa testaustietokoneella komento, joka on saatavissa Jenkinsistä tietokoneen sinne liittämisen jälkeen. Tämä komento käynnistää testaustietokoneella agentin, joka sallii Jenkins käyttäjän pääsyn tietokoneelle

4.2 Järjestelmän rakentamisen kulku

Opinnäytetyötä aloittaessa tuli muutamia viivästyksiä, niistä suurimpana oli USB-A -> internet-adapterien puute. Näitä tarvittiin kaksi laitteen ja testaustietokoneen väliseen yhteyteen, jotta SSH-yhteys laitteeseen oli mahdollinen. Näiden täydentäminen vei noin kaksi viikkoa, mutta aika käytettiin jo olemassa olevan testiautomaation ylläpitämiseen kuin myös siihen perehtymiseen, sekä testiautomaatiotietokoneen alustamiseen ja käyttöjärjestelmän asentamiseen, joten se ei sinänsä ollut hukattua aikaa. Kun adapterit saapuivat, oli laitteen yhdistäminen testiautomaatiokoneeseen varsin vaivaton prosessi. Kuvassa 4 on nähtävillä testattavan laitteen ja testaustietokoneen yhteyden arkkitehtuuri.



KUVA 4. Testaustietokoneen ja testattavan laitteen välinen fyysinen arkkitehtuuri

Testaustietokone yhdistetään USB-hubiin kahden USB-A -> internet -adapterin sekä niiden välisen internetjohdon kautta. Vaikka molemmat, alku- ja loppupää, ovat USB-A -liitännällä, oli RJ-45-johto

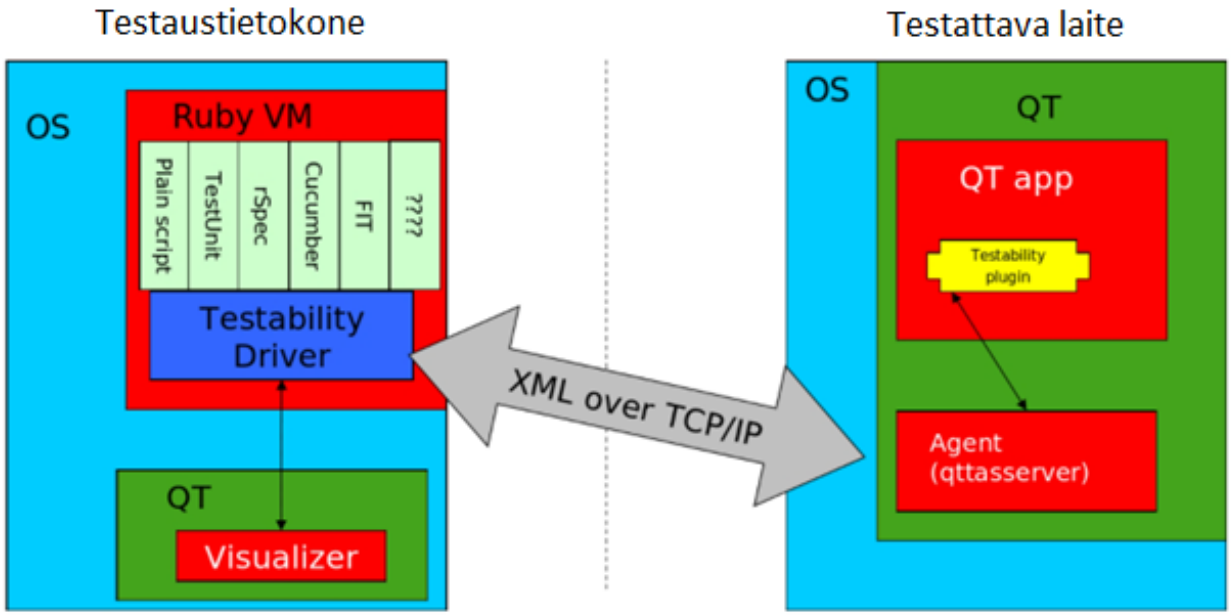
välissä pakollinen SSH-yhteyden kannalta. Tämän jälkeen USB-hub on liitettynä testattavaan laitteeseen USB-A -> USB-C-johdolla. USB-tikku on myös liitettynä USB-hubiin laitteen päivitystiedostoja sekä testattavia musiikkitiedostoja varten. Molemmat testattava laite sekä USB-hub ovat liitettynä verkkovirtaan.

Kun yhteys laitteeseen oli saatu, rakennettiin testaustietokoneelle seuraavaksi testausympäristö. Ensimmäisenä päämääränä tällä on saada ajettua yksittäisiä testitapauksia manuaalisesti testattavalla laitteella, ja sen jälkeen vasta liittää se Jenkinsiin. Symbion intranetissä on muutamia dokumentteja, ja ne osoittautuivatkin kriittiseksi tämän opinnäytetyön suorittamisen kannalta. Ensimmäinen oikea ongelma olikin, ettei näitä ollut vielä löydetty, joten ensimmäinen yritys oli yrittää manuaalisesti päätellä, mitä Ruby gemejä tarvittiin ja mitä ei. Ruby on ohjelmointikieli, joka toimii enimmäkseen erinäisten pienempien skriptien avulla kuin kokonaisten ohjelmistojen. Nämä gemit ovat käytännössä kirjastoja Rubylle, jotka sisältävät tässä tapauksessa tarvittuja skriptejä testien ajamiseen sekä niiden tarkasteluun ja kirjaamiseen (5). Lopulta kuitenkin kyseinen dokumentti löytyi, ja siellä oli pakattuna kaikki tarvittavat Ruby gemit yhteen pakettiin, sekä myös erityisesti Elysianin testaamista varten rakennettu gem. Tässä dokumentissa kerrottiin myös, kuinka cuTeDriver, testaamisen kannalta oleellinen framework-ohjelmisto asennetaan.

Seuraavaksi vastaantuleva ongelma olivat kymmenet virheilmoitukset, kun testejä lopulta saatiin suoritettua manuaalisesti terminaalissa. Tämän selvittämiseen kului odotettua enemmän aikaa, mutta lopulta, kun verrattiin rakennettua olemassa oleviin ratkaisuihin, huomattiin, että käytetään paljon uudempaa Ruby-versiota kuin muilla testaustietokoneilla. Tämän ratkaisemiseksi koneelle ladattiin sekä asennettiin rvm-ohjelmisto, eli Ruby version manager. Ohjelmisto on kehitetty käyttämään laitteella useampia Ruby-versioita yhtäaikaisesti. Ohjelmisto tosin oli täysin tuntematon sen käyttöä aloittaessa, joten sen mukana tuli uusiakin ongelmia. Kokemattomuuden vuoksi, tähän mennessä kaikki gemit oli asennettu käyttäen komennon alussa sudo-komentoa, joka on tällöin asentanut ne root-käyttäjällä kaikille muillekin käyttäjillä. Tämä taas rvm-ohjelmiston kanssa aiheuttaa erittäin paljon yhteensopivuusongelmia, koska rvm asettaa joka Ruby-versiolle oman kotipolkunsa. Tästä johtuen voi tulla tilanne, missä gem on asennettu, mutta sillä hetkellä käytössä oleva Ruby-versio ei löydä sitä, ja jos gemin sen jälkeen koittaa asentaa, kertoo terminaali, että gemi on jo asennettu. Tämä aiheutti erittäin paljon päänvaivaa, varsinkin kun olemassa olevat Ruby-versiot koitettiin poistaa aloittaakseen alusta, mutta tämä aiheutti vain sen, että myöskään Ruby-versiot eivät enää suostuneet asentumaan. Lopulta kuitenkin aikaa oli jo kulunut niin kauan tämän kanssa taistelemiseen ja internetfoorumien tutkimiseen, että kone vain alustettiin uutta yritystä varten. Se

oli päätös, joka olisi täytynyt luultavasti tehdä jo aikaisemmin. Kahdessa päivässä testaustietokone oli samassa tilassa kuin ennen alustamista, mutta gemien yhteensopivuusongelmat olivat poissa, kun ne asennettiin ilman sudo-komentoa.

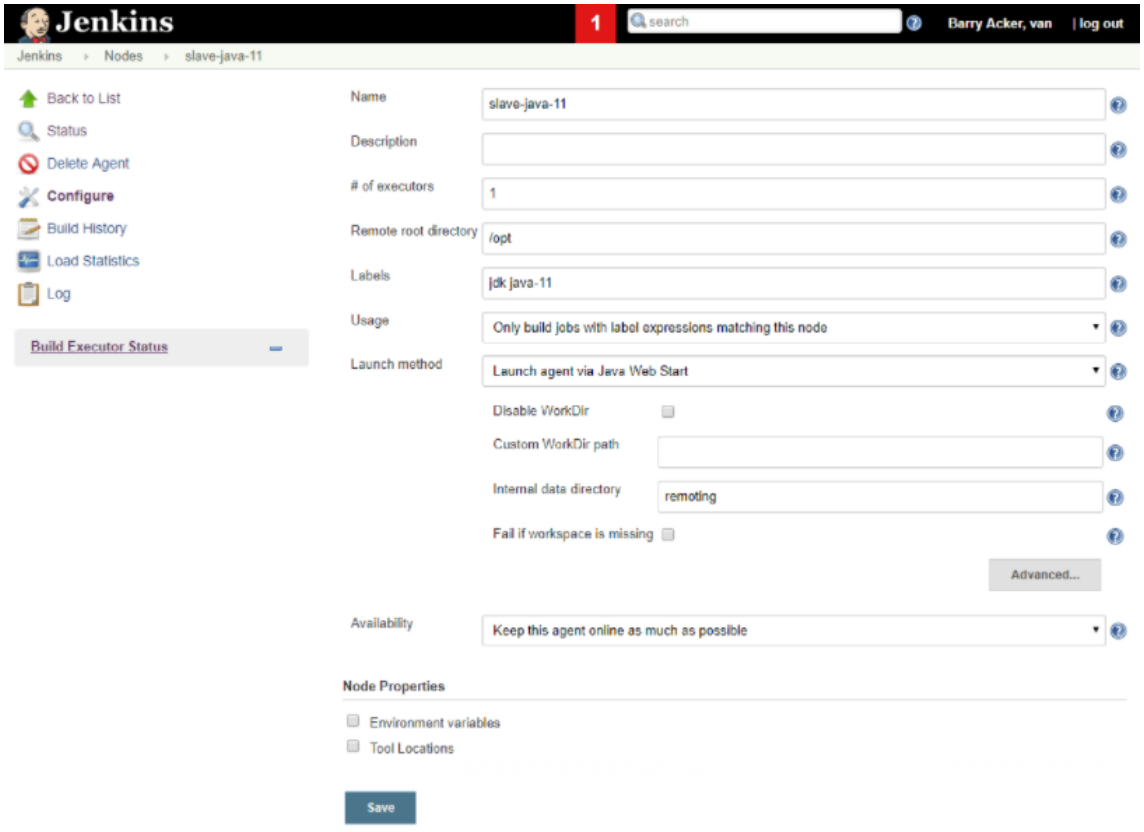
Tässä vaiheessa yksittäisiä testitapauksia pystyy ajamaan manuaalisesti terminaalista, niin Jenkins-käyttäjällä kuin Symbio-käyttäjällä. Symbio-käyttäjää käytetään pääasiassa järjestelmän huoltoa varten testaustietokoneen kautta. Seuraavaksi testaustietokoneelle asennettiin cuTeDriveria varten kehitetty Visualizer-sovellus. Tämä ohjelmisto mahdollistaa laitteen käyttöliittymän ja kaikkien sen elementtien, sekä niiden muuttujien, nimien ja muiden parametrien tarkastelun testaustietokoneella. Tämän mahdollistaminen vaatii kolme asiaa, että laitteen ohjelmisto on kehitetty qt-pohjalle, testaustietokoneelle on asennettu itse cuTeDriver ajuri -gem, sekä laitteelle on asennettu cuTeDriver agentti. Ajuri asennettiin viime vaiheessa, mutta koska laitetta päivittäessä se alustetaan täysin, tapahtuu agentin asentaminen laitteen päivittämisprosessin mukana.



KUVA 5. cuTeDriverin eri osat. Testaustietokone (vas.) sisältää ajurin. Testattava laite (oik.) sisältää agentin (6)

Testaustietokoneella on tässä vaiheessa kaikki mitä siellä tarvitseekin olla. Yhteydet GitLabiin ja TestRailiin eivät vielä ole automatisoituja, mutta niiden testaamiseksi kone yhdistetään ensin Jenkinsiin. Tämä tapahtuu pääasiassa Jenkinsin puolella. Symbion Jenkins-serverille voidaan lisätä uusi node eli solmu. Tämä tapahtuu näkymässä, joka on nähtävissä kuvassa 6. Tälle solmulle

annetaan IP-osoite, joka testaustietokoneelle on osoitettu Symbion internetin sisällä. Tunnistimilla solmun sisällä tarkennetaan, että tietokone on tarkoitettu automaatiotestaamiseen, ja tarkemmin ottaen uuden variantin ohjelmistoversioitten testaukseen. Tämä auttaa päätietokonetta tunnistamaan millaisia tehtäviä se solmulle määrää. Solmulle kerrotaan myös hakemisto, jossa sen on sallittua ajaa komentoja. Tässä tapauksessa se on testaustietokoneelle luodun Jenkins-käyttäjän kotihakemisto. Näiden lisäksi tunnistimissa kerrotaan, että uudella testaustietokoneellamme tahdotaan testata nimenomaan uuden variantin ohjelmistoja. Kun halutut asetukset on lisätty Jenkinsissä uuden solmun asetuksiin ja solmu on lisätty Jenkinsiin, antaa Jenkins komennon, joka syötetään testaustietokoneelle. Tämä komento käynnistää koneella agentin, joka kommunikoi Jenkinsin kanssa niin kauan kuin agentti pysyy käynnissä.



KUVA 6. Geneerinen näkymä uutta solmua tehdessä, ei sisällä opinnäytetyössä tehtyjä muutoksia (7)

Nyt testaustietokoneelta on yhteys Jenkinsiin, mutta Jenkins ei vielä sisällä yhtään tehtävää, mitä päätietokone voisi uudelle solmulle määrätä. Uuden tehtävän tekeminen on helppoa siinä määrin että uusi tehtävä on käytännössä nimeä ja tunnistimia lukuun ottamatta täysin samanlainen kuin

olemassa olevat. Ne muutetaan mukailemaan uutta varianttia sekä solmua, joka juuri luotiin. Sen lisäksi uudessa tehtävässä voidaan määrätä tehtävän käynnistämisen tapahtuvan toisen tehtävän loputtua. Tällä voidaan toteuttaa tilanne, jossa kokoamistietokoneen palautettua kootut artefaktit Jenkinsille, ja Jenkinsin merkattua kyseinen tehtävä valmiiksi, voi tekemämme testaustehtävä automaattisesti käynnistyä vasta valmistuneen kokoamistehtävän pohjalta. Luonnollisesti tämä ei kuitenkaan vaadi, että testaustietokoneet olisivat juuri sillä hetkellä vapaana. Testaustehtävä vain ilmoittaa päätietokoneelle, että laittaa sen jonoon ja määrää sen seuraavalle vapautuneelle solmulle, joka vastaa testaamisesta.

Tehtävässä myös määrätään komennot, joita Jenkins-käyttäjä ajaa testaustietokoneen terminaalissa. Ensimmäisenä näistä komennoista Jenkins hakee TestRailista tehtävän parametreissä ennalta määrätyn testitapauspaketin, ja sen jälkeen niitä vastaavat testiskriptit GitLabista. Näihin Jenkins saa pääsyn erikseen luodulla testiautomaatiokäyttäjällä, jonka kirjautumistiedot ovat konfiguraatiotiedostoissa Jenkinsin työtilassa. Tämän jälkeen Jenkins noutaa uuden vasta kootun ohjelmistoversion päivitystiedostot ja päivittää näillä testattavan laitteen. Kun cuTeDriver huomaa, että testattava käyttöliittymä on palannut laitteelle, aloittaa seuraava komento testipakkauksen läpikäymisen. Testiajo käy testitapaukset yksi kerrallaan satunnaisessa järjestyksessä. Testiajon aikana jokainen napin painallus, jokainen auennut näkymä ja tapahtunut muutos pyritään kirjaamaan lokitiedostoihin. Tämä mahdollistaa helpon ongelman paikannuksen, jos jokin testitapaus epäonnistuu. Tapauksen epäonnistuttua käyttöliittymästä otetaan myös kuvankaappaus, joka lisätään mukaan testituloksiin. Kun testiajo on loppunut, nämä testitulokset lähetetään sekä TestRailiin että Jenkinsiin helppoa tarkastelua varten. Näihin testituloksiin sisältyy kuvia, lokitiedostoja sekä muutamia automaattisesti luotuja kuvaajia, muun muassa erinäisten ohjelmiston palveluiden muistinkulutus.

Käytännössä edellä kuvailtu prosessi olisi kuinka järjestelmän täytyisi toimia, ja sitä myöten opinnäytetyön päämääräkin olisi suoritettu. Tässä vaiheessa aikaa oli hyvin rajallisesti. Sopimuksen Symbiolla oli määrä loppua noin kahden viikon päästä tästä hetkestä, mutta järjestelmässä oli vielä yksi kriittinen ongelma. Tehtävä käynnistyi oikealla ajallaan, laite päivittyi oikealla versiolla ennen testiajoa, ja kaikki näytti päällepäin menevän juuri niin kuin pitikin, mutta jostain syytä päivittämisen jälkeen yhtäkään testitapausta ei käyty laitteella ja testiajo merkattiin Jenkinsiin onnistuneena. Teoriana oli pitkään, että lista testitapauksia, jonka Jenkins-käyttäjä hakee TestRailista, olisi tyhjä. Tätä testatakseen täytyi opetella lukemaan JSON-listaa, joka TestRailista noudettiin, missä meni oma aikansa, mutta lopulta kun tuloksia saatiin näkyviin listalta, oli selvää, että TestRail palauttaa juuri

oikeat tapaukset. Lopullinen selvyys ongelmaan tuli vain pari päivää ennen työn loppua: testitapaukset, jotka TestRailista haettiin, sisälsivät väärät tunnistusnumerot verrattuna tapauksiin, jotka GitLabista täytyisi hakea. Jokainen TestRailissa oleva testitapaus sisältää tunnistusnumeron, joka vastaa samaa numeroa kuin jokaisen testiskriptin alussa oleva numero. Tämän avulla GitLabista osataankin hakea vastaavat skriptit. TestRailissa tämä on automaattisesti määrätty numero, joka ei ole muutettavissa. Kun olin TestRailissa kopioinut jo olemassa olevan projektin sekä sen sisällä olevat testitapaukset, oli TestRail määrännyt myös jokaiselle testitapauksen kopiolle uuden tunnistusnumeron, koska sen täytyy olla täysin uniikki. Tästä johtuen GitLabissa ei ollut yhtäkään testitapausta, joka vastaisi TestRaililta saatuja tunnistusnumeroita. Tässä vaiheessa minkään täydellisen ratkaisun luominen oli liian myöhäistä. Tilapäisesti kuitenkin ongelman pystyi korjaamaan muuttamalla Jenkinsin TestRail-käyttäjän konfiguraatiotiedostosta haettavan projektin. Alkuperäinen projekti, jonka olin kopioinut tätä testausjärjestelmää varten, sisälsi oikeat tunnistusnumerot testitapauksille. Tämän muuttamisen jälkeen järjestelmä toimii yhtä hyvin kuin olemassa olevat ratkaisut.

5 YHTEENVETO

Opinnäytetyö aloitettiin ideana tuottaa toimiva testiautomaatioratkaisu asiakasprojektia varten. Kyseessä oli Symbion oman tuotteen, Elysian-ohjelmistorungon pohjalle rakennettu ohjelmisto sekä laite, joka kyseistä ohjelmistoa ajoi. Testausjärjestelmä oli myös tarkoitus liittää Symbion olemassa olevaan Jenkins-järjestelmään jatkuvaa integraatiota varten.

Tehtävä oli jonkin verran odotettua vaikeampi. Suunnitelmana oli, että automaatiotestausjärjestelmä olisi koottu nopeammin, jotta se olisi ehtinyt osaksi jatkuvaan integraatioon. On myös mahdollista, että tehtävän vaikeus oli hieman aliarvioitu sen sijaan, että aikaa olisi tehtävään kulutettu liian kauan. Siitä huolimatta tehtävä olisi luultavasti tullut nopeammin tehtyä, jos apua olisi tullut useammin kysyttyä. Koin projektissa tärkeäksi itse tekemällä oppimisen, mutta siitä huolimatta pahimmissa paikoissa vinkkiä oli saatavilla aina kollegoilta.

Tärkeimmät päämäärät tehtävässä tulivat tehtyä. Testaus laitteen ja testaustietokoneen välillä on täysin toiminnallista, ja toteutettu järjestelmä toimii yhdessä olemassa olevan Jenkins-järjestelmän kanssa täydellisesti. Ainoa ongelma toteutetussa järjestelmässä verrattaessa jo olemassa oleviin on, että jouduin käyttämään toisen variantin TestRail-projektia. Tämä ei kuitenkaan sinänsä riko mitään. Molemmissa on käytössä täysin samat testit, mutta ideana oli käyttää järjestelmässä omaa TestRail-projektia, mitä ei tullut tehtyä. Selvä tulos edellä mainittujen lisäksi on myös, että järjestelmän arkkitehtuuri, sen konfiguraatiot, komponentit, niiden liitännät sekä kaikki muu, mitä Symbiolla on yrityksen jatkuvassa integraatiossa tuli selvitettyä ja ymmärrettyä. Näistä tuloksia ovat muun muassa tässä opinnäytetyössä nähtävät kaaviot.

Järjestelmässä on myös asioita, joita vielä pystyisi parantamaan mutta eivät tässä olleet tavoitteena. Joskus laitetta päivittäessä laite saattaa mennä jumiin ja käyttöliittymä ei käynnisty, joka johtaa testausautomaation kyseisellä laitteella seisahtumiseen siihen asti, että se päivitetään manuaalisesti eri menetelmällä, joka tyhjentää laitteen täysin perusteellisesti. Tämä on ongelma joko laitteessa tai ohjelmistoversiossa, ei päivitysmenetelmässä.

Projekti oli erittäin hyödyllinen oppimisen kannalta. Testaaminen oli täysin uusi asia minulla työn alkaessa Symbiolla, ja sen huipentuminen järjestelmän itse rakentamiseen oli erittäin opettavainen kokemus. Siitä huolimatta, että iso osa järjestelmästä oli valmiiksi rakennettu, kuului työhön paljon

tutkimusta, ja suurin tehtävä olikin ymmärtää valmista järjestelmää. Sen mukana oli monta sivustoa ja palvelua, joiden kanssa en ollut ikinä aiemmin ollut tekemisissä. Jenkins ja TestRail olivat näistä merkittävimmät. GitLab itsessään oli myös sivusto, jota en ollut koskaan käyttänyt, mutta se toimi käytännössä kuten GitHub. En kuitenkaan ollut erityisen taitava minkään Git-sivuston kanssa, joten tulin paljon tutummaksi sen erinäisten komentojen kanssa. Jatkuva integraatio oli myös ennen tätä minulle tuntematon käsite, mutta se oli täysin looginen ja helposti ymmärrettävä asia. Uskon sen olevan myös asia, jonka ammattilaisille on tulevaisuudessa paljon kysyntää.

LÄHTEET

1. Kauppalehti. Symbio Finland Oy. Hakupäivä 8.12.2021. https://www.kauppalehti.fi/yritykset/yritys/symbio+finland+oy/1110235-2.

2. Rehkopf, Max 2021. What is Continuous Integration?. Atlassian. Hakupäivä 8.12.2021. https://www.atlassian.com/continuous-delivery/continuous-integration.

3. AWS 2021. What is Continuous Integration?. Amazon. Hakupäivä 8.12.2021. https://aws.amazon.com/devops/continuous-integration/.

4. GeeksforGeeks 2021. Difference Between GitLab and GitHub. Hakupäivä 7.12.2021. https://www.geeksforgeeks.org/difference-between-gitlab-and-github/.

5. Silva, Gonçalo. Structure of a gem. RubyGems. Hakupäivä 11.12.2021. https://guides.rubygems.org/what-is-a-gem/.

6. Paimen 2018. cuTeDriver Wiki. GitHub. Hakupäivä 8.12.2021. https://github.com/nomovok-opensource/cutedriver-agent_qt/wiki.

7. Van Acker, Barry 2018. Adding a Jenkins Slave. Barry Van Acker's blog. Hakupäivä 13.12.2021. https://barryvanacker.nl/adding-a-jenkins-slave/.