Robot Framework -testien lisääminen osaksi jatkuvan integraation prosessia

Henri Lehtinen

2022 Laurea

Laurea-ammattikorkeakoulu

Robot Framework -testien lisääminen osaksi jatkuvan integraation prosessia

Henri Lehtinen

Tradenomi (AMK), tietojenkäsittely

Opinnäytetyö

Toukokuu, 2022

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittely
Tradenomi (AMK)

Henri Lehtinen

Robot Framework -testien lisääminen osaksi jatkuvan integraation prosessia

Vuosi 2022 Sivumäärä 35

Tämä opinnäytetyö on Solidabis Solutions Oy:n helmikuussa 2022 toimeksi antama. Ammattikorkeakoulu opinnäytetyön kehitystyönä tavoiteltiin teknistä ratkaisua, jolla voitaisiin yhtenäistää ja automatisoida testiautomaatioon liittyviä prosesseja. Teknisen ratkaisun tuli tehdä testiautomaation ajosta toimintavarmempaa ja tarjota nopeaa palautetta koko kehitystiimille testituloksista. Opinnäytetyö suunniteltiin ja toteutettiin kevään 2022 aikana asiakasprojektissa. Tietoperusta kehitystyölle haettiin alan uudesta ammattikirjallisuudesta koskien ohjelmistotuotannon Devops toimintamalleja ja testiautomaation käytäntöjä.

Kehitysmenetelmänä oli teknisen ratkaisun kehittäminen Microsoft Azure Devops tuotteilla. Teknisessä ratkaisussa hyödynnettiin konttiteknologioiden tarjoamia mahdollisuuksia systeemitestauksen automaattisessa ajamisessa osana projektin olemassa olevaa jatkuvan integraation prosessia. Toteutetusta ratkaisusta kerättiin palautetta ja jatkokehitysideoita tiimistä laadullisen tutkimuksen keinoin hyödyntämällä puolistrukturoitua haastattelumallia. Haastattelussa ilmenevien seikkojen ja alan kirjallisuuden perusteella opinnäytetyössä esitetään konkreettisia jatkokehitysehdotuksia, miten asiakasprojektissa voitaisiin hyödyntää jatkuvan integraation ja käyttöönoton prosesseja tulevaisuudessa entistä laajemmin.

Kehitystyön tuloksena asiakasprojektin laadunvalvontaan liittyvät prosessit yhtenäistyivät ja ohjelmistotestauksen tulokset olivat paremmin esillä koko projektitiimin hyödyksi. Haastatteluiden perusteella jatkuvan integraation prosessi nähtiin kehitystyön jälkeen projektissa voimavarana, joka mahdollistaa muutosten läpiviennin ohjelmistokehityksessä entistä nopeammin ja luotettavammin. Toimintavarma automaatio ympäristö poistaa projektin työmääristä hukkaa ja vapauttaa resursseja kehitystyöhön. Projektin ohjelmistokehittäjät olivat tyytyväisiä kehitystyönä tehdystä muutoksesta, sillä se helpotti heidän päivittäistä työtään.

Asiasanat: devops, robot framework, jatkuva integraatio, testiautomaatio

Laurea University of Applied Sciences **Abstract**
Degree Programme in Business Information Technology
Bachelor's thesis

Henri Lehtinen

Adding Robot Framework tests as part of continuous integration process

Year 2022 Pages 35

This bachelor's thesis was initiated by Solidabis Solutions Oy in February 2022. The goal was to develop a technical solution to unify and automate processes related to test automation. The technical solution should help running test automation more reliably and should also provide quick feedback to the project team on the test results. The bachelor's thesis was planned and executed during the spring 2022 in a client project. The basis for development work was obtained from the newest professional literature concerning software engineering Devops and test automation practices.

The development method was to develop a technical solution using Microsoft Devops products. The technical solution exploited the possibilities of container technology to automate running of system tests as part of an existing project continuous integration process. Qualitative methods in the form of a semi-structured interview were applied for obtaining feedback about the developed solution and ideas for improvement. Ideas from the interviews and from the professional literature made possible to present concrete suggestions how continuous integration and delivery could be further utilized in the project.

The results of the development work done for this bachelor's thesis showed that the quality assurance related processes were unified. The end results of the software testing were better communicated for the benefit of the project team. According to the interviews, the process of continuous integration was viewed more as an asset for the project which enables making and delivering changes in software development rapidly and reliably. Reliable test automation environment helped optimize resources in the project work. The software developers of the project were content with the changes made because it made their daily work easier.

Keywords: devops, robot framework, continuous integration, test automation

Sisällys

1 Johdanto

Opinnäytetyö toteutetaan kehittämistyönä Solidabis Solutions Oy:n toimeksiannosta. Solidabiksen asiakasprojektissa, jossa työskentelen sovelluskehittäjänä, on opinnäytetyön aiheeseen liittyviä kehittämistarpeita. Toimeksiantajan esittämä ongelma opinnäytetyön ratkaistavaksi koskee projektin epäyhtenäisten testiautomaatiokäytäntöjen yhtenäistämistä pilvipohjaisella ratkaisulla. Epäyhtenäiset toimintatavat aiheuttavat turhaa vaihtelua testituloksissa ja tämä on koettu projektissa erittäin ongelmalliseksi. Opinnäytetyön tarkoitus on löytää tähän ongelmaan tekninen ratkaisu ja tuottaa projektitiimille uutta osaamista ohjelmistokehityksen uusista Devops menetelmistä.

Toimeksiantajan toiveesta opinnäytetyössä ei paljasteta asiakkuuden tai projektin nimeä, vaan siihen viitataan vain asiakasprojektina. Aikaisemmin kyseisen asiakasprojektin monet järjestelmää koskeneet vaatimukset tarkentuivat tai muuttuivat johtuen muutoksista toimintaympäristössä ja sovelluksen vaatimuksissa. Useita arkkitehtuurillisia parannuksia oli viety läpi ennen opinnäytetyön kehitystyön aloittamista. Laajat muutokset kehitteillä olevan sovelluksen toiminnassa ja arkkitehtuurissa korostavat tarvetta testausautomaation yhtenäisille automaattisille prosesseille ja tämän ongelman tämä opinnäytetyö ratkaisee. Opinnäytetyön kehitystyö tehdään kevään 2022 aikana osana normaalia konsultoivaa ohjelmistokehitys projektityötä.

Opinnäytetyöni kehitystyö keskittyy asiakasprojektin ohjelmistotuotannon prosessien, erityisesti testiautomaatio prosessien kehittämiseen uudenlaisella teknisellä ratkaisulla. Kehitystyössä lisätään testauksen automaation astetta kehittämällä projektin systeemitestien testiautomaatio osaksi projektin olemassa olevaa jatkuvan integraation prosessia, joka toimii Azure pilvipalvelussa. Hyödyntämällä laajemmin automaatiota projektitiimi ja sovelluksen omistava asiakas saavat arvokasta tietoa sovelluksen laadusta. Automaation tarjoama tieto saavuttaa tiimin nopeammin ja on luotettavampaa kuin satunnaisesti suoritettu testaus. Jatkuvan integraation työkalun laadulliset ominaisuudet ennalta-ehkäisevät ongelmia myöhemmin sovelluksen julkaisuprosessissa ja käyttöönotossa. Nopeampi palautesykli sovelluksen kehityksen ja testauksen välillä mahdollistaa nopean reagoinnin eri tilanteisiin ja helpottaa muutosten läpivientiä tulevaisuudessa.

2 Opinnäytetyön lähtökohdat

2.1 Tarve kehittämistyölle

Asiakasprojektissa oli aiemmin ollut ongelmia uusiutuvien bugien eli ohjelmistovirheiden kanssa. Testiautomaation silloisen toteutuksen eli pelkkien yksikkö- ja integraatiotestien kattavuus ei ollut riittävä huomioimaan kaikkia erikoistilanteita. Testikattavuuden ohitse oli päässyt kehittäjien toimesta koodia, joka rikkoo aiemmin toteutettuja ominaisuuksia joltain osin. Tätä varten oli opinnäytetyön teon ajankohtaa edeltävänä aikana kehitetty systeemitestausta toteuttavia automaatiotestejä Robot Framework työkalulla. Robot Framework on avoimen lähdekoodin testaustyökalu systeemitestaukseen (Robot Framework 2022). Systeemitestit toistavat erikoistilanteita ja validoivat, että nekin toimivat edelleen kehitystyön edetessä ja sovelluksen muuttuessa. Systeemitestit olivat jo olemassa opinnäytetyön kirjoittamisen aikaan ja niitä oli n. 1500 eri testitapausta. Erikoistilanteiden varalle kehitetyt automaatiotestit ovat toimineet erinomaisesti. Ne antavat palautetta kehityksen virheistä huomattavasti nopeammin kuin perinteiset manuaaliset testausprosessit.

Testiautomaatiota asiakasprojektissa on suoritettu automaattisesti yksikkö- ja integraatiotestien osalta jatkuvan integraation prosessissa aina kun ohjelmistokehittäjä on esitellyt uutta toteutusta sovellukseen. Asiakasprojektissa on pidetty kiinni laatukriteereistä, joiden mukaan yksikkö- ja integraatiotestien on mentävä läpi kaikessa uudessa toteutuksessa. Aiemmin mainitut uudet systeemitestausta testaavat automaatiotestit, joita ajetaan Robot Framework työkalulla, eivät olleet kuitenkaan opinnäytetyön kirjoittamisen aikaan säännöllisessä ajossa jatkuvan integraation työkalussa. Robot Framework testit ovat tukeneet asiakasprojektissa kehitystyötä manuaalisesti ajettuna. Robot Framework testit on ennen opinnäytetyön kehitystyön teknistä ratkaisua ajettu tarvittaessa kehittäjän omalla työasemalla.

Asiakasprojektin palavereissa oli huomattu, että manuaalisesti ajetut systeemitestit sinänsä toimivat, mutta testiautomaatioprosessissa on useita tunnistettuja heikkouksia. Systeemitestien ajaminen saattaa olla siitä kiinni, että tietty henkilö on niitä ajamassa ja on niiden tuloksia valvomassa. Eri kehittäjien henkilökohtaisten tietokoneiden kehitysympäristöt ovat hieman toisistaan eroavia, joka aiheuttaa testituloksiin epäyhteneväisyyksiä. Henkilöriski manuaalisesti ajettavista testeistä on suuri, sillä kaikki ohjelmistokehittäjät asiakasprojektissa eivät tunne Robot Framework testaustyökalua ja miten sitä käytetään. Tiedon saanti testausten tuloksista tulee vaihtelevalla viiveellä suhteessa virheiden todelliseen ilmaantumiseen sovelluksessa. Testitulosten saannin jälkeen on epäselvää, mikä nimenomainen muutos sovelluksessa on rikkonut aikaisemman toteutuksen tai sen osan. Tieto testaustilanteesta ei ole projektissa yleisesti saatavilla, joten julkaisuja tuotantoon on voinut olla suunnitteilla, vaikka testit olisivat rikki. Testausprosessiin tarvitaan parannusta ja tämä opinnäytetyö pyrkii ratkaisemaan nämä ongelmat jatkuvan integraation teknisellä ratkaisulla.

2.2 Kehittämistavoitteet

Toimeksiannossa Solidabis hakee keinoja asiakasprojektin parempaan testiautomaatioprosessiin. Olemassa olevan testiautomaation testituloksista halutaan nopeammin palautetta kehittäjien ja koko projektiryhmän hyödyksi. Testitulosten halutaan olevan ajankohtaisempia ja tarkempia kuin aiemmin. Kehitystyössä hyödynnetään asiakasprojektin olemassa olevaa sovellusarkkitehtuuria ja automaattitestejä. Projektin olemassa olevien työvälineiden takia ratkaisu pyritään ensisijaisesti löytää hyödyntämällä ja laajentamalla Microsoft Azure Devops tuoteperheen ratkaisuja (Microsoft 2022). Kehitystyössä kehitetään projektin testiautomaatioprosessia toimimaan automaattisesti ja toimintavarmasti.

Kehittämistyössä perehdytään testiautomaation prosesseihin teoriassa ja käytännössä. Opinnäytetyössä hyödynnetään alan kirjallisuudesta alan parhaita käytäntöjä ja luodaan systeemitestauksen automaattiselle ajolle tekninen ratkaisu. Tekninen ratkaisu testiautomaation ajamiseen liitetään osaksi asiakasprojektin olemassa olevaa jatkuvan integraation prosessia. Ensimmäisestä versiosta teknistä ratkaisua kerätään palautetta puolistrukturoidun haastattelun keinoin projektin vastuuhenkilöltä ja projektin ohjelmistokehittäjiltä. Tutkimuskysymyksiin sisältyy esimerkiksi, mitä jatkuvan integraation työkalun uudessa testiautomaatioprosessissa on koettu hyödylliseksi, mitä haasteita niiden kanssa on ollut ja mitä jatkokehitystarpeita prosessissa voisi olla jatkossa. Haastatteluiden ja alan ammattikirjallisuuden perusteella Solidabis Solutions Solutions Oy haluaa myös saada konkreettisia jatkokehitysideoita, miten asiakasprojektissa voitaisiin hyödyntää automatisoituja prosesseja entistä paremmin.

2.3 Kehitystyön laajuuden rajaus

Kehitystyön yksi haaste on tutkittavan ja kehitettävän kokonaisuuden rajaus. Tutkimuksessa keskitytään ensisijaisesti ohjelmistotuotannon prosessin kehittämiseen. Testiautomaatioon liittyy valtavia teknisiä kokonaisuuksia, joita kaikkia ei voida kehitystyössä huomioida laajasti. Opinnäytetyössä avataan sellaiset käsitteet ja työkalut, jotka ovat olennaisia Robot Framework tyyppisten automaatiotestien liittämiseksi osaksi pilvipalvelun jatkuvan integraation prosessia.

Ratkaisukuvauksessa kerrotaan yleiset toimenpiteet, joita Robot Framework testeille on tehtävä, jotta ne soveltuvat ajettavaksi pilviympäristössä hyödyntämällä konttiteknologia perusteista ratkaisua. Vastaavanlaiset muutokset, jotka opinnäytetyössä esitellään, toteutetaan käytännössä myös asiakasprojektissa. Asiakasprojektissa Robot Framework testitapauksia on satoja. Asiakasprojektin toteutuksen tarkkoja yksityiskohtia ei paljasteta, mutta lukija saa riittävät tiedot soveltaakseen tämän opinnäytetyön havaintoja myös muissa projekteissa, joissa on vastaavanlaisia tarpeita.

2.4 Keskeiset käsitteet

Automaatiotesti	Yksittäinen testitapaus, jonka suoritus on automatisoitu yksikkö-, integraatio tai systeemitestauksen automaatiotyökalujen avulla.
Azure	Microsoftin pilvipalveluiden tuoteperheen kaupallinen nimi. Pilvipalvelut pitävät sisällään Microsoftin datakeskusten erilaisia kattavia tietoteknisiä ratkaisuja tietoteknisen infrastruktuurin ja sovelluskehityksen tarpeisiin.
Azure Devops	Microsoftin tuote, joka tarjoaa ohjelmistotuotannon kokonaisratkaisun pitäen sisällään versio- ja projektihallinnan, vaatimusten hallinnan ja automatisoidun testauksen ja julkaisun hallinnan. Automatisoitu testaus ja julkaisu pitää sisällään palveluita, jolla voi toteuttaa Devops periaatteiden mukaisesti automaattiset jatkuvan integraation ja toimituksen prosessit.
Devops	Ohjelmistotuotannon toimintamalli, jossa pyritään automatisoimaan ohjelmistokehityksen, testauksen ja ylläpidon eri prosessit mahdollisimman pitkälle. Ohjelmistokehityksessä tämä näkyy jatkuvan integraation ja toimituksen automaattisina prosesseina.
Docker	Alustaratkaisu, jossa mahdollistetaan ohjelmistojen paketointi ja toimittaminen helposti hallittavana kokonaisuutena. Docker käyttää käyttöjärjestelmän virtualisointia sovelluksen tarvitsemien järjestelmäkomponenttien yhdistämisessä yhteen toimitettavaan pakettiin.
Helm	Kuberneteksen paketinhallintatyökalu, joka helpottaa konfiguraation hallintaa ja julkaisutoimintaa eri laatuisissa ympäristöissä.
HTTP	Internetin tiedonsiirtoon tarkoitettu protokolla.
Integraatiotestaus	Ohjelmistotestauksen menetelmä, jossa useita osia sovelluksesta testataan yhdessä. Sovellusten eri osien yhteistoimintaa ja toimivuutta arvioidaan, toimivatko ne vaatimusten mukaisesti.
Jatkuva integraatio (CI)	Ohjelmistotuotannon menetelmä uuden lähdekoodin yhdistämisessä ohjelmistoprojektin aikaisempaan lähdekoodiin. Jatkuvan integraation prosessissa varmistetaan uuden ja vanhan koodin

	yhdistetyn uuden kokonaisuuden oikeellisuus ja laatu. Englanniksi käsite on continuous integration (CI).
Jatkuva toimitus (CD)	Ohjelmistotuotannon menetelmä, jossa sovelluskehityksen kehittämät uudet ominaisuudet ja korjaukset säännöllisesti ja/tai automaattisesti asennetaan käyttöön eri palvelinympäristöihin käytettäväksi. Englanniksi käsite on continuous delivery (CD).
Kubernetes	Alustaratkaisu, jossa mahdollistetaan suurten sovelluskokonaisuuksien tehtävien ja muutosten helpompi hallinta. Kubernetes alustan kontekstiin on mahdollista siirtää suoritettavaksi esimerkiksi Docker alustalle paketoituja sovelluksia.
Pipeline	Jatkuvan integraation työkalun merkittävä osa. Sarja erilaisia toimintoja, joita halutaan suoritettavan jokaiselle versiohallinnan tallennetulle muutokselle.
Python	Ohjelmointikieli. Python on tarkoitukseltaan yleispätevä ohjelmointikieli moniin eri käyttötarkoituksiin. Pythonin yksinkertaisen syntaksin ansiosta se on suhteellisen helppo oppia.
Systeemitestaus	Ohjelmistotestauksen menetelmä, jossa sovellusta testataan kokonaisuutena. Testauksen kohteena on järjestelmän koko työn kulku ja prosessi, toimiiko se vaatimusten mukaisesti. Systeemitestauksessa voidaan huomioida käyttöliittymä ja erilaiset ulkoiset palvelut.
Robot Framework	Automaatiotyökalu systeemitestaukseen. Robot Framework hyödyntää testitapausten suorittamiseen avainsanoja, joista voi muodostaa ei-teknisiä selkokielisiä testitapauksia. Avainsanojen suorittamia toimenpiteitä on mahdollista ohjelmoida Python ohjelmointikielellä.
Shell	Unixin komentorivi työkalu, joka on laajasti käytössä myös Linuxissa ja Dockerissa. Unix komentorivin toimintoja voidaan hyödyntää Dockerin erilaisten määritysten toteutukseen.
SOAP	SOAP on http protokollaa hyödyntävä tietoliikenne protokolla, jossa viestin rakenne noudattaa XML merkintäkielen standardia.
Testiautomaatio	Ohjelmistotestauksen menetelmä, jossa hyödynnetään automaatiotyökaluja erilaisten testitapausten toistamiseen.

	Automatisoitu testi mahdollistaa testin kustannustehokkaan suorittamisen, kun samaa tapausta halutaan testata toistuvasti.
Yksikkötestaus	Ohjelmistotestauksen menetelmä, jossa eristettyjä osia sovelluksen lähdekoodista testataan erikseen, toimiiko se vaatimusten mukaisesti. Eristetty kokonaisuus voi olla esimerkiksi yksittäinen lähdekoodin funktio tai luokka.
XML	XML on merkintäkielen standardia, jonka avulla tieto voidaan järjestää loogisen rakenteen mukaan.
Xunit	Xunit on koontitermi eri testaustyökaluille, joiden toimintaperiaatteet ovat yhdenmukaisia ja yhteensopivia. Yhteensopiva raportointi mahdollistaa XUnit tyyppisten testien tulosten julkaisun jatkuvan integraation työkalussa.

3 Ohjelmistotuotannon viitekehys

3.1 Ohjelmistojen testaus

Muuttuvat vaatimukset edellyttävät muutoksia kehitteillä olevaan sovellukseen. Kaikki muutokset kehitteillä olevan sovelluksen koodissa ja arkkitehtuurissa altistavat järjestelmää erilaisille virheille. Jotta virhetilanteilta käytössä olevassa järjestelmässä voitaisiin välttyä, tarvitaan kattavaa testausta. Kattavalla testauksella voidaan varmistaa, että kaikki toiminnallisuudet toimivat tehdyistä uusista muutoksista huolimatta. Huolellinen testaus parantaa ja ylläpitää ohjelmistotuotteen laatua. Testaus on tärkeä osa ohjelmistotuotannon laadunvarmistusta.

Ohjelmistojen tulee toimia sille asetettujen vaatimusten mukaisesti. Eri ohjelmilla on erilaiset vaatimukset, joiden mukaan niiden toimintaa koskevat odotukset on määritetty. Ohjelmisto, joka toimii vaillinaisesti tai väärin voi aiheuttaa suuria ongelmia. Huonosti toimiva sovellus voi aiheuttaa sitä käyttävällä organisaatiolle suuria taloudellisia ja maineen menettämisen riskejä. Joillain toimialoilla sovelluksessa tapahtuva virhe voi vaarantaa jopa ihmisen hengen. Tämän takia ohjelmistojen toimintaa on erittäin tärkeää testata, jotta voidaan minimoida ohjelmiston virheiden riskit liiketoiminnalle. (Spillner & Linz 2021, 7)

Ohjelmistotestaus ei ole pelkkää testitapausten suorittamista, vaan on laaja kokonaisuus, johon liittyy monenlaisia työvaiheita. Ohjelmistotestaus osana ohjelmistotuotantoa pitää sisällään testauksen suunnittelua, monitorointia ja analysointia. Suunnitellut testitapaukset toteutetaan ja suoritetaan. Suoritetuista testitapauksista raportoidaan testauksen aikana ja sen

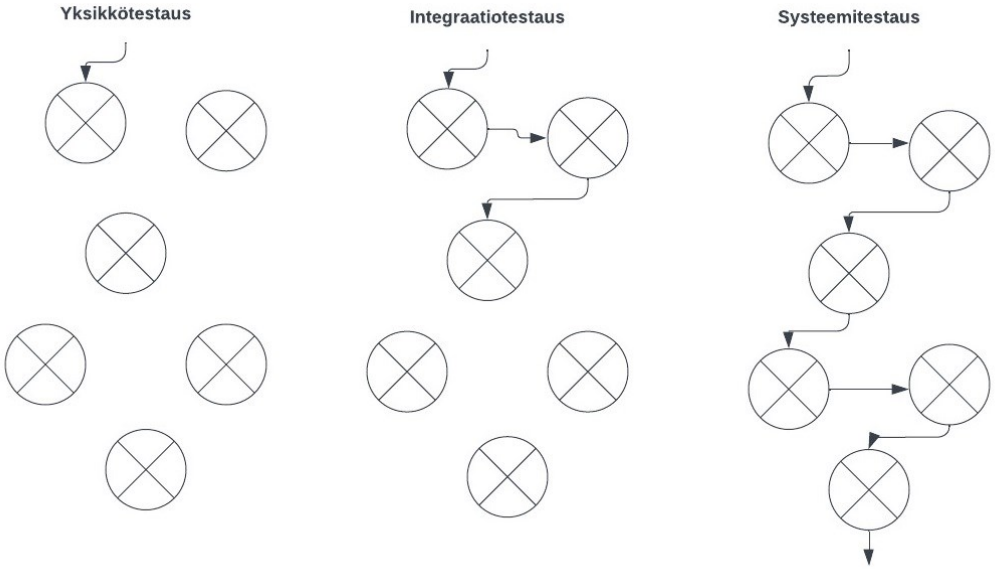
jälkeen. Testiraporttien tuloksista luodaan riskianalyysejä, joiden pohjalta voidaan tehdä päätöksiä jatkotoimenpiteiden suhteen. Ohjelmistotestaus ei rajoitu pelkästään ohjelman toiminnallisten osuuksien dynaamista testausta vaan sisältää myös määrittelyiden ja suunnitelmien testausta. Ohjelmistotestaus pyrkii myös tuottamaan tietoa siitä, että sovellus täyttää myös ohjelmistolle asetetut ei-toiminnalliset vaatimukset ja toiveet. Ei-toiminnallisia vaatimuksia ovat esimerkiksi käytettävyyteen, suorituskykyyn ja tietoturvaan liittyvät asiat. (Spillner & Linz 2021, 8)

Ohjelmistotuotannossa ohjelmistotestaus seuraa kehitystyötä. Pitkään vallalla olleella vesiputous mallilla johdetussa projektissa testataan toteutus vasta sitten, kun kehitystyön ajatellaan olevan valmis. Nykyään yleistyneessä ketterän kehittämisen mallissa testaus on osa ohjelmistokehitystyötä lomittain toteutuksen edistymisen kanssa. Ketterissä menetelmissä pyritään saada pieni osa sovelluksesta valmiiksi kehitysiteraation aikana. Kehitysiteraatio saattaa olla kestoltaan 1–3 viikkoa. Rajalliseen aikatauluun on mahdutettava ohjelman uusien ominaisuuksien määrittely, toteutus, testaus ja käyttöönotto. (Spillner & Linz 2021, 54)

Kehittäessä ohjelmistoprojektia ketterin menetelmin, testaustarve ja ajettavat testitapaukset kasvaa iteraatiosta toiseen. Testaustarve pitää sisällään kaiken uuden toteutuksen ohjelmistoon, mutta myös aikaisempien toiminnallisuuksien toimivuuden varmistamisen. Testaukseen käytettävä aika ja resurssit eivät silti usein ole lisättävissä, jolloin regressiotestauksessa eli menneiden ominaisuuksien toimivuuden testauksessa on korvaamattoman tärkeää hyödyntää testiautomaatiota. Testi-automaatiossa tietokone testaa ihmisen puolesta ohjelman toimivuutta, vähentäen manuaalisen työn tarvetta. Pelkästään manuaalisella testaustyöllä urakka muuttuu nopeasti mahdottomaksi tehtäväksi ohjelman toiminnallisuuksien määrän kasvaessa. (Spillner & Linz 2021, 55)

3.2 Testiautomaatio

Testiautomaatio jaetaan yleensä yksikkö-, integraatio- ja systeemitestaukseen. Testauksen jaottelu perustuu testattavan kokonaisuuden laajuuden eroon. Yksikkötestit pyrkivät testaamaan sovelluksen pienintä mahdollista eristettävää osaa, kuten yksittäistä funktiota tai luokkaa. Integraatiotestit ovat kahden tai useamman erillisen ohjelmiston komponentin yhteistoiminnan testausta, esimerkiksi ohjelmiston ja rajapinnan välillä. Systeemitestauksessa pyritään testaamaan ohjelmiston kaikki komponentit yhdessä ja testaamaan ohjelmaa toimivana kokonaisuutena. Systeemitestauksella todennetaan, että ohjelma toimii vaatimusten mukaisesti myös realistisissa käyttöolosuhteissa. Systeemitestauksessa pyritään toistamaan todellinen käyttötapaus kehitteillä olevalla ohjelmalla alusta loppuun. (Chopra 2018, 227–240) Kuvassa 1 on visualisoitu näiden eri testityyppien eroa.

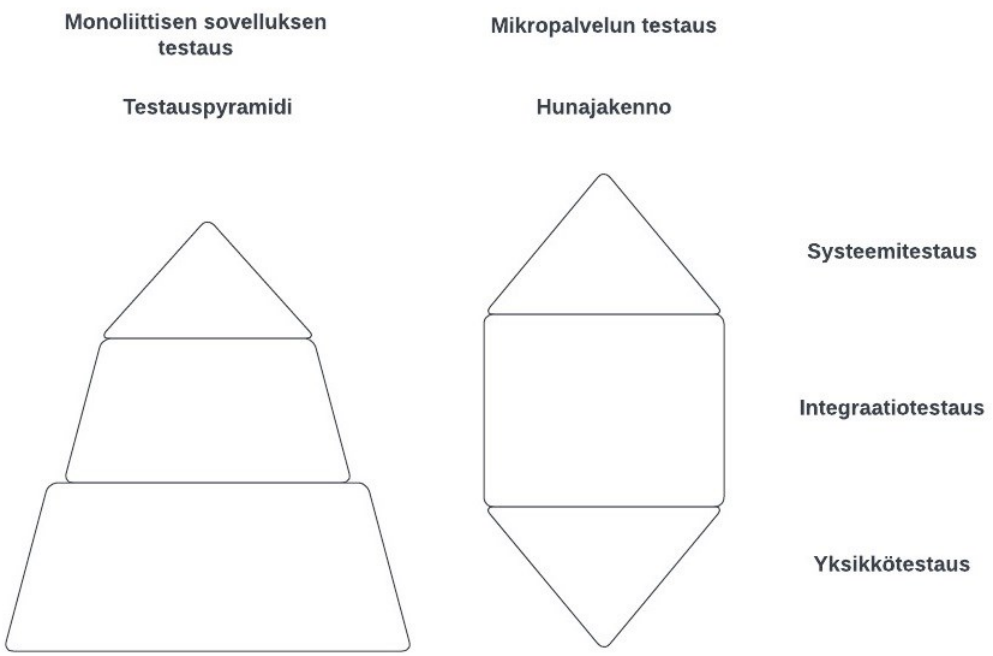


Kuvio 1: Testiautomaation eri tyypit (Chopra 2018, 227)

Testiautomaation määrää perinteisessä ohjelmistoprojektissa on kuvattu mm. ’testaus pyramidin’ avulla. Perinteiset ohjelmistoprojektit ovat monoliittisia, suuria projekteja, joissa sama ohjelma suorittaa useita eri vaatimuksia ja käyttötapauksia. Testauspyramidissa pyritään luomaan eniten bisneslogiikkaa koettelevia pieniä yksikkötestejä. Integraatiotestejä tehdään hieman vähemmän, sillä ohjelma viestii pääasiassa itse itsensä kanssa. Systeemitestejä tehdään vielä integraatiotestejä vähäisempi määrä. (Rajput 2018, 88–89)

Suurten monoliittisten ohjelmien kompleksisuus nousee ajan mittaan erittäin suureksi ja siitä on monia haittoja ohjelmistotuotteen nopealle kehittämiselle ja testaamiselle (Rajput 2018, 89). Tämän takia suuri osa uusista ohjelmistojen projekteista on siirtynyt toteuttamaan mikropalveluarkkitehtuuria, jossa ohjelman eri vastuualueet on pilkottu pienempiin kokonaisuuksiin. Pienemmät mikropalvelut ovat nopeampia uudelle kehittäjälle ottaa haltuun, hajotessaan eivät hajota koko ohjelmaa ja mahdollistavat järjestelmän testauksen ja käyttöönoton pienemmissä kokonaisuuksissa. (Rajput 2018, 11–14)

Mikropalveluarkkitehtuurin testaus asettaa uusia haasteita testiautomaation kokonaisrakenteelle ja jaolle yksikkötestien, integraatiotestien ja systeemitestien välillä. Mikropalvelun bisneslogiikka on rajattua, mutta se viestii paljon sovellusten ulkopuolisten palveluiden ja muiden mikropalveluiden kanssa. Mikropalvelu saattaa suorittaa vain tiettyä toiminnallisuutta, joten systeemitestauskin on silloin rajattua. Mikropalveluarkkitehtuurissa korostuu integraatiotestauksen tärkeys.



Kuvio 2: Testiautomaatio eri arkkitehtuureissa (Rajput 2018, 89)

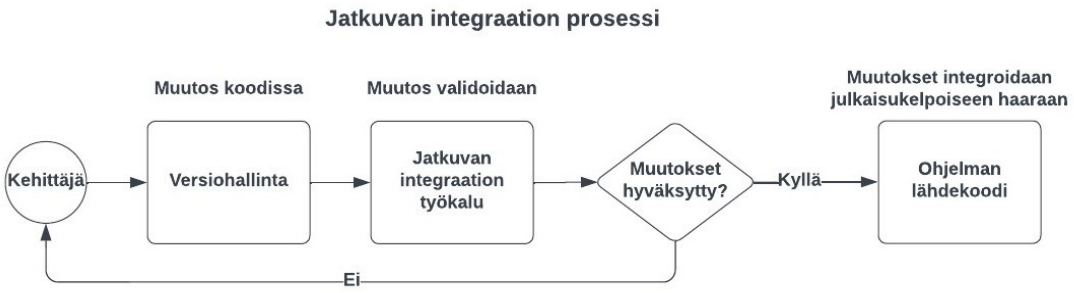
Mikropalvelun suurin kompleksisuus ei ole bisneslogiikassa vaan siinä, kuinka se kommunikoi muille palveluille eri tilanteissa. Tämän takia mikropalveluiden testaamiseen Dinesh Rajput esittääkin kirjassaan ’Hands-On Microservices - Monitoring and Testing: A Performance Engineer's Guide to the Continuous Testing and Monitoring of Microservices’ vaihtoehdoksi testauspyramidin sijaan testaus ’hunajakennoa’. Testaus ’hunajakennossa’ tarve integraatiotestaukselle korostuu muihin testauksenlajeihin nähden. Tämä malli on visualisoituna kuvassa 2. (Rajput 2018, 89)

3.3 Devops

Devops on lyhenne englannin kielen sanoista *development* ja *operations*. *Development* tarkoittaa kehitystyötä ja *operations* sovelluksen ylläpitoa. Perinteisesti ohjelmistotuotanto on toiminut vaiheittaisesti määrittelystä kehitystyöhön ja kehitystyöstä testaukseen ja testauksesta käyttöönottoon. Ketterät menetelmät ovat tuoneet näitä sidosryhmiä lähemmäs toisiaan. Devops vie tämän vielä askeleen pidemmälle, sillä siinä tuodaan sovelluksen ylläpitäjät lähelle sovelluksen kehitystyötä. Devops mallissa kaikki ohjelmistotuotannon vaiheet pyritään automatisoimaan mahdollisimman pitkälle erityisesti testauksen ja käyttöönoton osalta hyödyntämällä testiautomaatiota ja automatisoitua käyttöönottoa. (Krief 2021, luku 1)

Devops menetelmät pitää sisällään jatkuvan integraation ja käyttöönoton prosessit. Jatkuvan integraation prosessiin liittyy olennaisena osana erilaisen testiautomaation ajo. Jatkuvan integraation prosessi ajaa kaikki siihen asennetut automaattitestit uudelle ohjelman versiolle.

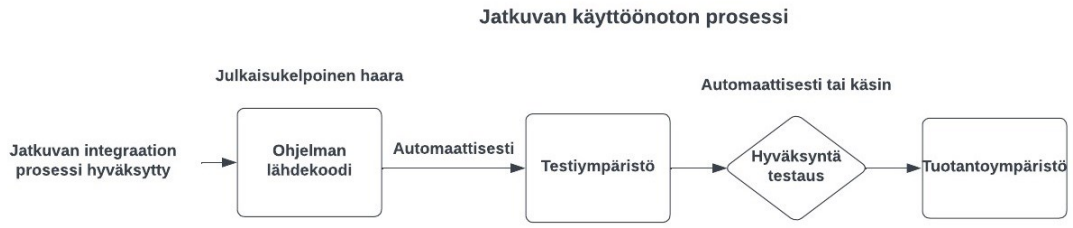
Mikäli automaattinen testauksen tulos on hyväksyttävä, voidaan uusi versio toimittaa eteenpäin ja ottaa käyttöön hyödyntäen jatkuvan käyttöönoton prosessia. (Spillner & Linz 2021, 55–56). Tämän opinnäytetyön kehitystyö liittyy testiautomaation ja jatkuvan integraation prosessin yhteen liittämiseen.



Kuvio 3: Jatkuvan integraation prosessi (Krief 2021, luku 1)

Jatkuva integraatio on automaattinen prosessi, jonka avulla jokainen ohjelman koodiin tapahtunut muutos voidaan varmentaa automaattisin testein ja automaattisesti integroida osaksi ohjelmistoprojektin julkaisukelpoista kehityshaaraa. Tavoitteena on myös tarjota nopeasti palautetta muutoksen toimivuudesta ja laadusta ohjelmistokehittäjälle. Koko kehitystiimin täytyy toteuttaa muutokset yhdessä sovitun prosessin mukaisesti. Jatkuvan integraation prosessin käyttöönotto edellyttää tiimiltä osaamista versiohallinnasta ja jatkuvan integraation työkaluista. (Krief 2021, luku 1) Tiimi voi käyttää versiohallintatyökaluna esimerkiksi avoimen lähdekoodin Git versiohallintaa (Git 2022). Jatkuvan integraation työkaluna on mahdollista käyttää esimerkiksi avoimen lähdekoodin Jenkinsiä tai Azure Devopsia (Jenkins 2022).

Jatkuvan integraation prosessi tarjoaa ohjelmistotuotannolle automaattiset keinot saada palautetta ohjelmistossa esiintyvissä bugeissa ohjelman elinkaaren aikaisemmissa vaiheissa. Ohjelmistossa esiintyvien bugien korjaaminen on helpompaa ja edullisempaa mitä aikaisemmassa vaiheessa sen olemassaolo tiedetään. Useimmat nykyaikaiset pilvipalveluiden tarjoajat tarjoavat myös jatkuvan integraation työkaluja. Ketterää ohjelmistokehittämistä noudattavat ohjelmistokehitystiimit voivat hyödyntää niitä rakentamalla automatisoituja integrointiprosesseja, joita kutsutaan Pipelineiksi. Automatisoidut Pipelinet uuden koodin integrointiin ja käyttöönottoon mahdollistavat niitä käyttäville organisaatioille nopeampaa ohjelmistokehitystä ja vähemmän käyttökatkoja ohjelmistojen kanssa. (Agarwal 2021, luku 6)



Kuvio 4: Jatkuvan käyttöönoton prosessi (Krief 2021, luku 1)

Devops mallissa jatkuvan integraation prosessin voi laajentaa jatkuvan käyttöönoton prosessiin. Jatkuvan käyttöönoton prosessissa hyväksytysti suoritettua jatkuvan integraation prosessia jatketaan pidemmälle. Luottaen jatkuvan integraation prosessissa hyväksytysti suoritettuun testiautomaatioon, muutettu ohjelman lähdekoodi asennetaan ja käyttöönotetaan testiympäristön palvelimella automaattisesti. Jatkuvan käyttöönoton prosessissa voidaan automatisoida ohjelmiston käyttöönoton asennukset eri käyttötarkoituksiin tarkoitetuissa ympäristöissä. Pisimmälle vietynä jatkuvan käyttöönoton mallilla voidaan automatisoida jopa tuotantoympäristön asennus. Näin pitkälle viety jatkuvan käyttöönoton malli edellyttää erittäin laajaa testiautomaatiota, joka harvassa organisaatiossa on riittävän korkealla tasolla. (Krief 2021, luku 1)

4 Tutkimus- ja kehittämismenetelmät

4.1 Tietopohja

Opinnäytetyön kehitystyön tietopohja muodostuu Devopsin ja testiautomaation ammattikirjallisuuden parhaiden käytäntöjen ja kirjallisuuden periaatteiden soveltamisesta asiakasprojektissa kehittämällä olemassa olevaa jatkuvan integraation prosessia. Kehitystyössä ratkaistaan asiakasprojektissa koetut haasteet testausprosesseissa. Kehitystyönä tehdään tekninen ratkaisu, jossa nykyiset Robot Framework systeemitestit voidaan ajaa asiakasprojektin olemassa olevassa jatkuvan integraation prosessissa.

Tekninen toteutus pitää sisällään Robot Framework -automaattitestien ja siihen liittyvän Python koodin uudelleenjärjestelyä toimimaan Azuren konttiteknologiaan perustuvassa Kubernetes ympäristössä (Kubernetes 2022). Tarvittava uusi pilvipalveluinfrastruktuuri tulee määritellä ja toteuttaa toimimaan osana Azure Devops Pipeline automaatioprosessia. Integraatiotyökalun tulee raportoida sidosryhmille systeemitestien onnistumisesta tai epäonnistumisesta Azure Devops alustalla.

Tutkimuksellisena osana kerätään laadullisin menetelmin palautetta teknisen ratkaisun ensimmäisestä versiosta avoimin kysymyksin. Haastattelussa tutkitaan mitä hyötyjä kehitystiimi on kokenut jatkuvan integraation hyödyntämisestä ja mitä haasteita sen kanssa on ollut. Haastattelussa kerätään ideoita testiautomaation jatkokehitykselle. Haastatteluun käytetään puolistrukturoitua haastattelumallia (Jyväskylän Yliopisto 2021).

Tutkimuksen ja kehitystyön kohteena on koko kehitystiimiä koskeva prosessi, joten lyhyeen haastattelututkimukseen pyydetään osallistumista kaikilta tiimin jäseniltä ja vastuuhenkilöiltä. Tutkimuksen perusjoukon ja haastatteluun pyydettävien ollessa sama, voidaan puhua kokonaistutkimuksesta. Tämä on mahdollista ja järkevää silloin kun tutkittava perusjoukko on kooltaan pieni (Jyväskylän yliopisto 2015). Haastatteluun saatiin kaksi projektin vastuuhenkilöä ja seitsemän ohjelmistokehittäjää. Haastateltavien tarkat henkilötiedot suojataan ja niitä ei opinnäytetyössä paljasteta.

4.2 Luotettavuus ja eettiset näkökulmat

Kehitystyön luotettavuus perustuu tietoperustaan, joka on esitetty julkaistussa alan arvostetussa ja julkaistussa uudessa ammattikirjallisuudessa. Lähteitä valitessa teoriapohjaan tulee kiinnittää erityistä huomiota lähdekriittisyyteen, ettei kehitystyö perustu yksittäisiin mielipiteisiin tai yksittäisen yrityksen tarjoamiin tuotteisiin. Lähteitä käyttäessä tulee olla tarkkana, jottei syyllisty toisten työn plagiointiin. Tärkeimmät teesit kehitystyön päätösten perustaksi on hyvä löytyä useista arvostetuista lähdeteoksista. Lähteen luotettavuutta arvioitaessa on hyvä kiinnittää huomiota lähteen luotettavuuteen, objektiivisuuteen ja ajantasaisuuteen (Helsingin Yliopisto 2022). Monet tietotekniikan alan aineistot ovat jonkun yrityksen julkaisemia ja alan blogeissa on hyvinkin värikkäitä mielipiteitä, miten asiat kuuluisivat tehdä. Tietotekniikan alan aineistojen paikkansapitävyys myös vanhenee nopeasti, joten vanhoja lähteitä ei voi käyttää.

Kehitystyön eettisiä vaatimuksia tulee opinnäytetyössä sopimusasioista ja henkilötietojen suojaamisesta julkisuudelta (Näreaho, Kettunen, Kärki & Päällysaho 2020). Opinnäytetyön tutkimuksessa käytetyissä haastatteluissa tulee minimoida henkilötietojen käsittely vain välttämättömään. Haastateltavien projektissa työskentelevien henkilöiden nimet anonymisoidaan. Haastatteluiden mahdollisia nauhoituksia ja muistiinpanoja säilytetään vain välttämätön aika. Julkaistava opinnäytetyö on julkinen asiakirja ja sen sisältö tulee hyväksyttää opinnäytetyön tilaajalla, ettei siinä paljasteta julkisuuteen liikesalaisuuksia tai asiakasprojektin yksityiskohtia.

5 Kehittämiskohteen toteutus

5.1 Projektin esittely

Asiakasprojekti on laaja ja kompleksinen terveydenhuolto alan laskutusjärjestelmä. Laskutusjärjestelmän tarkoitus on muodostaa useiden eri lähdejärjestelmien lähettämästä datasta automaattisesti laskutusaineistoa lähetettäväksi SOAP/XML muodossa laskutusoperaattorille Finvoice esitystavan mukaisesti. Finvoice-esitystapa on Suomessa yleinen standardi taloushallinnollisten aineistojen siirtoon eri osapuolten välillä, ja sitä hyödyntävät useimmat Suomessa toimivat pankit (Finanssiala 2022).

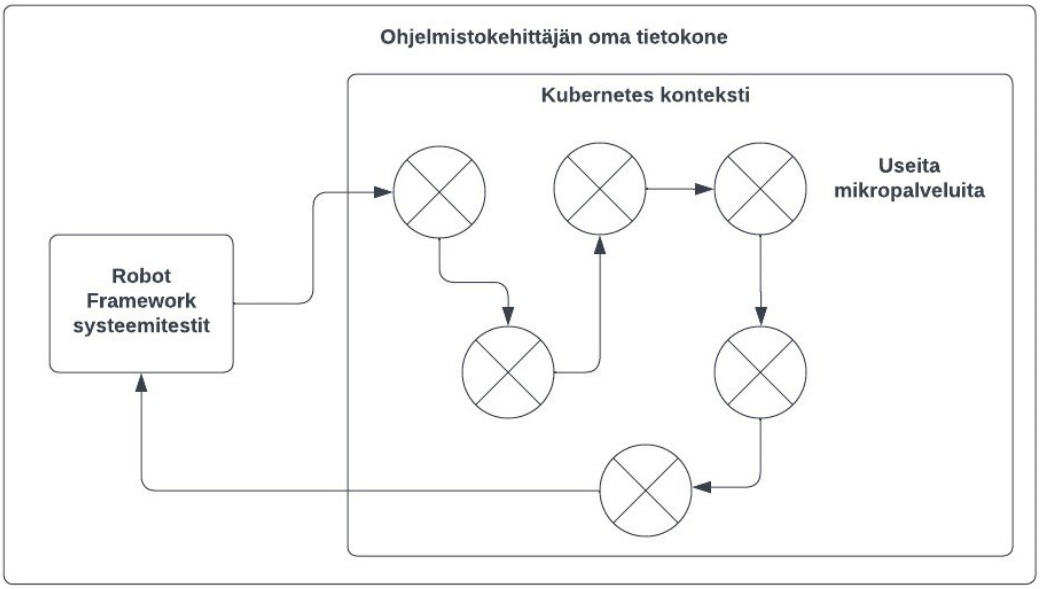
Asiakasprojekti on rakennettu mikropalveluarkkitehtuurin ja Domain-driven design periaatteiden mukaisesti. Domain-driven design on alun perin Eric Evansin kirjassa Domain-driven design vuonna 2003 määritelty joukko suunnittelumalleja, jolla erilaisia toimialaympäristöjä voidaan kuvata olio-ohjelmointipohjaisissa ohjelmistoratkaisuissa (Evans 2003). Mikropalveluarkkitehtuurissa nämä ongelmanratkaisuperiaatteet näkyvät mikropalveluina eli itsenäisinä ohjelmistokomponentteina, jotka toteuttavat omia vastuualueitaan (Pacheco 2018, luku 1). Mikropalveluarkkitehtuuri asiakasprojektissa on toteutettu Kubernetes ympäristöön useina itsenäisinä eri Spring-Boot mikropalveluina (VM-Ware 2022). Kubernetes on hallinnointiratkaisu, jolla voi orkesteroida itsenäisten konttien käyttöönottoa, julkaisemista ja päivittämistä (Agarwal 2021, luku 4).

5.2 Tilanne ennen kehitystyötä

Asiakasprojektissa hyödynnetään jo ennestään Azure Devops jatkuvan integraation työkalua laajasti. Nykyinen Microsoftin jatkuvan integraation Azure Pipeline tunnistaa mikropalvelut, joiden koodi on muuttunut. Pipeline on jatkuvan integraation työkalun prosessin pääasiallinen osa. Pipeline aktivoituu yhä uudestaan ohjelmistokehittäjien kehittäessä ohjelmaa ja tallentaessa muutoksia versiohallintaan (Microsoft 2022). Azure Pipelines automaattisesti kasaa ja testaa koodia muuttuneiden mikropalveluiden osalta. Versiohallinnan tallennettua muutosta kutsutaan commitiksi (Git 2022). Jokainen tallennettu uusi commit versiohallinnassa laukaisee Pipelinen eli automaattisen prosessin jatkuvan integraation käsittelyyn. Asiakasprojektissa käytettävä nykyinen Azure Pipeline testaa sovelluksen koodin Java pohjaisella yksikkö- ja integraatiotyyppisellä testiautomaatiolla, mutta ei suorita Robot Framework systeemitestejä.

Asiakasprojektin nykyinen Azure Pipeline noudattaa Yaml määrittelyä siihen, mitä eri työvaiheita Pipeline käy läpi ja missä järjestyksessä (Microsoft 2022). Yaml on tyypillisesti konfiguraatiotiedostojen ohjelmointiin tarkoitettu merkintäkieli, jonka esitystavassa hyödynnetään sisennysvälejä tiedon hierarkian esittämiseen samoin kuin Python ohjelmointikielessä (Red Hat 2022). Yaml määrityksen stages osiossa hyödynnetään Kuberneteksen manageröintityökalua Helmiä ensin koko projektin kaikkien mikropalveluiden konttien asentamiseen Pipelinessä

(Helm 2022). Asennusten valmistumista odotellaan Helm suorituksen ’waitForExecution’ parametrillä, mikä on mahdollista Helm Deploy tyyppisissä Pipeline tehtävissä (Microsoft 2022).



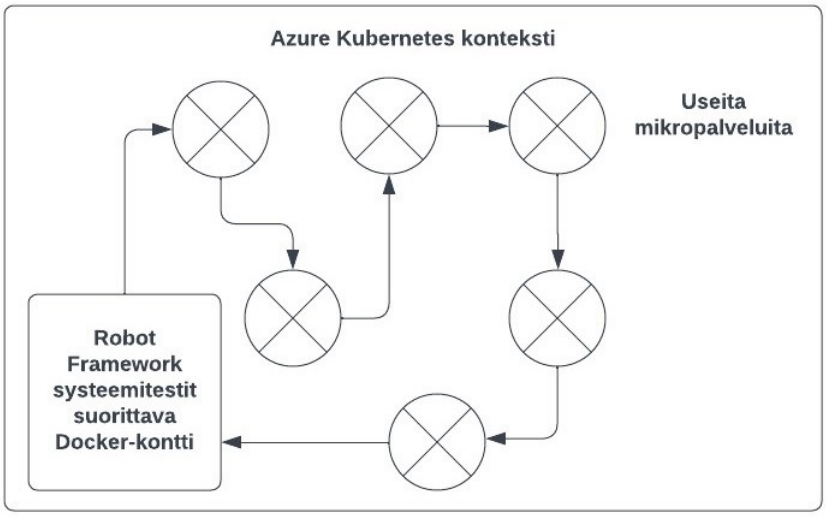
Kuvio 5: Asiakasprojektin Robot Framework testauksen nykytila

Robot Framework systeemitestit on tähän asti ajettu pelkästään manuaalisesti ohjelmistokehittäjien omilta tietokoneilta. Testaaminen omalla työpäätteellä on edellyttänyt lokaalin Kubernetes klusterin ylläpitoa ja kehitteillä olevan ohjelman asennusta siihen. Työasemalla käytettävän Kuberneteksen mikropalvelut on tullut konfiguroida osaksi Zuul API Gateway ratkaisua, jotta niihin yhdistäminen olisi ollut mahdollista Kubernetes klusterin ulkopuolelta (Javatpoint 2022). Työaseman ympäristön ongelmat käyttöjärjestelmässä, suorituksessa, asetuksessa, tietokannan tilassa tai verkkoyhteyksissä vaikuttavat testien ajon lopputulokseen aiheuttaen projektissa epäyhteneväisiä testituloksia. Paikallisten testiympäristöjen säteilevät ongelmat ovat yksi merkittävä syy miksi kehitystyön yhtenäistetty testiympäristö perustetaan pilvipalveluun. Robot Framework testien ajon nykytilanne on kuvattu yksinkertaistettuna kuvaan 5.

5.3 Ratkaisun tekninen malli

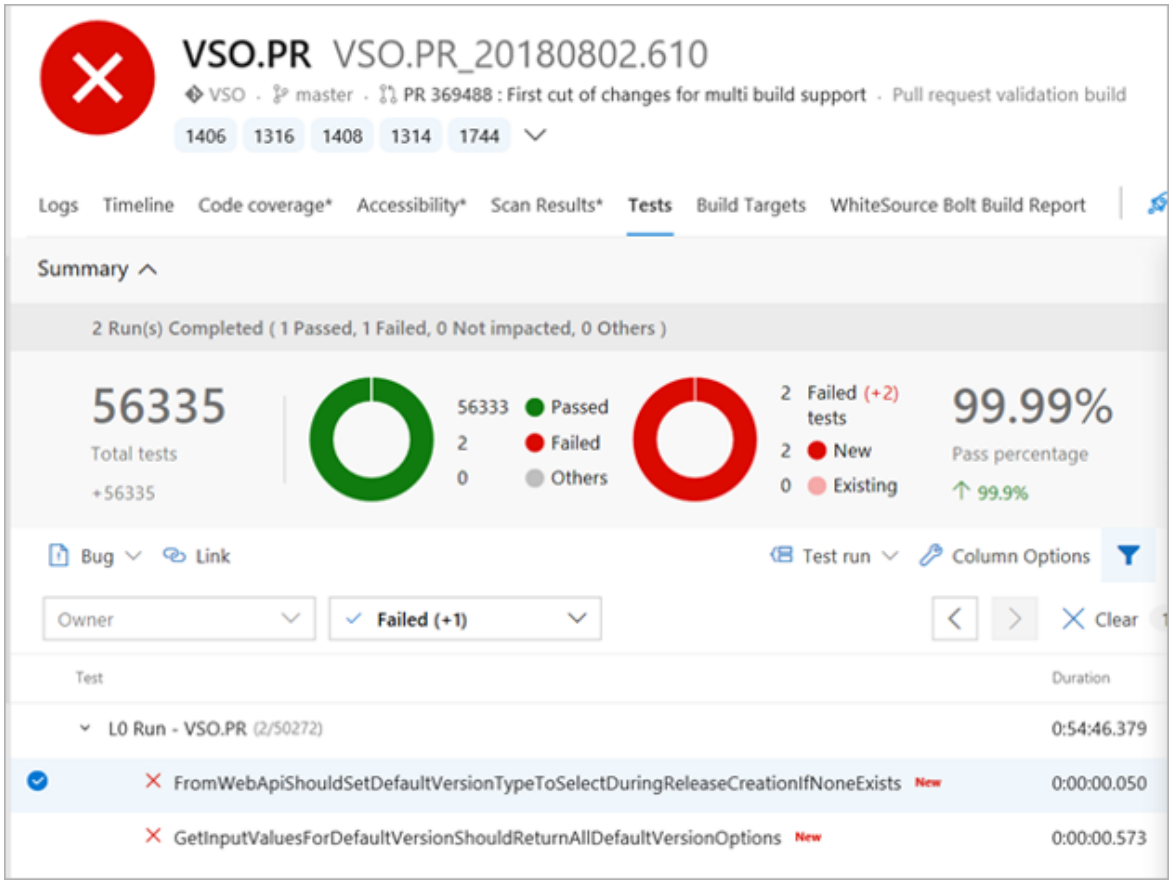
Kehitystyön ratkaisussa ajettaville Robot Framework systeemitesteille luodaan oma Docker kontti ajettavaksi Kubernetes klusterin sisäpuolella. Docker on ohjelmistokehityksen työkalu, jolla ohjelmiston ulkoiset riippuvuudet ja vaatimukset voidaan paketoida itsenäiseen yksikköön käyttöjärjestelmästä riippumatta (Agarwal 2021, luku 1). Robot Framework testejä voidaan suorittaa Docker kontin ansiosta kehittäjän omalla tietokoneella, mutta myös sellaisenaan pilviympäristön Pipeline automaatioprosesseissa. Kubernetes ympäristö oli

asiakasprojektissa jo käytössä ennen opinnäytetyötä. Uusi testausarkkitehtuuri on kuvattu yksinkertaistettuna kuvassa 6.



Kuvio 6: Robot Framework systeemitestaus Kubernetes ympäristössä

Kehitystyönä luotava Docker kontti liitetään osaksi asiakasprojektin olemassa olevaa Azure Pipeline Yaml määritystä siten, että kun järjestelmän kaikki mikropalvelut on kasattu Azure Pipelinessa, niin viimeisenä asennuksena asennetaan Robot Framework testiautomaation sisältävä Docker kontti. Robot Framework Docker kontti asentaa itse itsensä sille määritellyillä ohjeilla ja alkaa välittömästi suorittaa systeemitestausta. Testiautomaatiota suorittavan Docker kontin päivittämistä hallinnoidaan erillisellä Pipelinella, joka kasaa ja julkaisee uudet Docker Imaget Azure ACR palveluun, jolloin uusimman testiautomaatio koodin mukaiset Docker kontit ovat varsinaista testiautomaatio ympäristöä rakentavien prosessien käytettävissä (Microsoft 2022).



Kuvio 7: Esimerkki Pipelinen testiajon tulosten näkymästä (Microsoft 2022)

Testitulosten valmistumisen jälkeen kontti tallentaa allokoituun levytilaan Sftp:llä testitulokset levyallokaatioon. Sftp on tietoturvallinen tiedonsiirtoprotokolla (WinSCP 2022). Robot Framework tallentaa testitulokset XUnit muodossa, joka on yleinen jatkuvan integraation työkalujen hyväksymä muoto (Microsoft 2022). Prosessi tunnistaa milloin Robot Framework testien ajo on valmistunut Docker kontin tilasta. Valmiit testitulokset julkaistaan Azure Devops portaalin Build raportissa. (Microsoft 2022). Esimerkki miltä testitulokset näyttävät on nähtävillä kuvassa 7.

Azure Devops hyödyntää Robot Frameworkin yhteensopivaa XUnit XML muotoista testiraporttia muodostaakseen jokaisesta Pipelinen ajosta visuaalisen yhteenvedon (Microsoft 2022). Visuaalinen yhteenveto tarjoaa täsmällistä testaustietoa jokaisesta kehitteillä olevan ohjelman muutoksesta. Jatkuvan integraation Azure Pipeline prosessi suoritetaan jokaiselle yksittäiselle muutokselle ohjelman koodissa. Jokaisesta suoritetusta testistä on saatavilla Robot Framework työkalun tallentamaa lokitietoa. Tämä tieto on erityisen hyödyllistä epäonnistuneiden testien selvittelytyössä.

Azure Devops raporttisivu tarjoaa arvokasta tietoa ohjelmistokehitys tiimille tehtyjen muutosten laadusta. Mikäli uusi muutos ohjelman koodissa rikkoo jotain olemassa olevaa

toiminnallisuutta, jonka validoimiseksi on tehty testiautomaatiota, niin siitä saa tiedon nopeasti tältä raporttisivulta (Microsoft 2022). Kehitystyöni muutosten jälkeen jatkuvan integraation prosessissa ja raportoinnissa ovat myös mukana kattavasti testaavat Robot Framework systeemitestit.

5.4 Kehitystyön käytännön toteutus

Robot Framework testit koostuvat robot tiedostoista ja niihin liittyvistä Python ohjelmointikielellä toteutetuista python tiedostoista (Robot Framework 2022). Robot Framework vaatii toimiakseen, että ajoympäristöön on asennettu Python (Python 2022). Docker kontin esivaatimukset esitellään Dockerfile nimisessä tiedostossa, joka pitää sisällään kaikki komennot, mitä kontin halutaan suorittavan (Docker 2022).

```
FROM python:3

RUN apt-get update

RUN python3 -m pip install robotframework

CMD ["/run_suite.sh"]
```

Kuvio 8: Esimerkki Dockerfile tiedostosta

Keskittyen vain siihen mitä Robot Framework tarvitsisi minimissään toimiakseen muodostuu Robot Framework testien kansioon seuraavanlainen Dockerfile, joka on nähtävillä kuvasta 8. Dockerfile perustuu Python versio 3 jakeluun ja kykenee suorittamaan tämän alustan mukaisia tehtäviä. Asiakasprojektin varsinaisessa toteutuksessa asennettavien vaatimusten lista Dockerfilessä oli huomattavasti esimerkkiä pidempi pitäen sisällään erillisiä kirjastoja mm. tietokantayhteyksiä ja Sftp yhteyksiä varten.

Kuvasta 8 voidaan nähdä, että FROM arvo määrittelee mille alustalle ja versiolle kontti perustuu. RUN arvolla voidaan antaa Linux Shell (GNU 2022) syntaksin mukaisia komentorivikäskyjä, joita Docker kontti suorittaa. RUN arvoissa voidaan esimerkiksi määrittää mitä Python kirjastoja on tarpeellista asentaa ennen varsinaista kontin tehtävän suorittamista. CMD arvoja voidaan antaa lopuksi vaan yksi, mutta siinä voidaan kutsua ulkoista Shell Scriptiä, johon on määritelty tarkemmin mitä kontin halutaan suorittavan. (Docker 2022)

```
robot -d /mnt/sftp/robot -x junit_report.xml /usr/tests

exit 0
```

Kuvio 9: Esimerkki script tiedostosta

Kuvasta 9 voidaan nähdä Shell script tiedosto run_suite.sh, joka pitää sisällään varsinaisen Robot Framework testin käynnistämisen komentorivikäskyn. Robot komento käynnistää Robot Framework testin ajon. Parametrilla -d määritellään kansio, johon testiraportti tallennetaan. Tässä tapauksessa tallennus tehdään Kuberneteksen levylle hyödyntäen Sftp yhteyttä. Parametrilla -x määritetään testiraportin tyypiksi XUnit, joka on yhteensopiva jatkuvan integraation työkalujen kanssa. Lopuksi Docker kontin suoritus päätetään paluuarvolla 0, joka tarkoittaa, että ohjelmassa tapahtunut mitään erityistä virhettä (Madamanchi 2019). Kontin suorituksen päättäminen asettaa Docker kontin tilaan ’Ready’ ja tätä tilaa tarvitaan myöhemmin Pipeline määrityksissä.

Dockerfilen lisäksi tuli lisätä myös docker-compose Yaml tiedosto, jossa on määritykset Docker kontin käynnistämistä varten (Docker 2022). Esimerkki docker-compose tiedostosta on nähtävissä kuvasta 10.

```
version: “3.0”

services:

        robot-test-example-job:

                container_name: robot-test-example-job

                image: robot-test

                build:

                        context: .
```

Kuvio 10: Esimerkki Docker-compose tiedostosta

Docker tiedosto tuli omalla tietokoneella kasata ja lähettää projektin yhteiseen Azure ACR palveluun. Azure ACR on Azuren tarjoama Docker konttien hallinnointiratkaisu, jota käytetään Pipeline prosessissa automaattisesti (Microsoft 2022). Docker kontin kasaus tehtiin hyödyntäen docker ja Azure komentorivityökaluja. Azure CLI mahdollistaa kirjautumisen Azuren resursseihin omalta tietokoneelta ja pilviresurssien muokkaamisen (Microsoft 2022). Azure ACR palvelua koskien asiakasprojektissa oli oma Pipeline, jonka Yaml tiedostoon lisättiin myös Docker-compose tehtävä, joka huolehti Docker julkaisun päivittämisestä Azure ACR palvelussa, mikäli testiautomaation koodi päivittyisi (Microsoft 2022).

Olemassaolevan testiautomaatioympäristön Azure Pipelinen Yaml tiedostoon tuli lisätä viimeiseksi arvoksi uusi komento robottitestien ajavaa Docker tiedostoa varten. Lisäksi tarvittiin toinen komento Robot Framework XUnit tyyppisten testitulosten noutoa ja julkaisua varten

(Microsoft 2022). Testitulosten noutamistehtävässä odotetaan, että Robot Framework testejä suorittava Docker kontti on tilassa 'Ready' eli lopettanut suorituksensa paluuarvolla 0. Toteutuskohtaiset ympäristömääritykset tuli asiakasprojektissa kirjata Helm charts/templates kansion Yaml tiedostoon, jotta Kubernetes julkaisut toimivat projektin yhteisen käytännön mukaisesti. Helm charts yleiset määritykset tuli esitellä myös Helmin values.yml tiedostossa, josta niitä voidaan hyödyntää template kansion Yaml tiedostoihin viittauksina (Helm 2022).

6 Kehittämiskohteen tulokset

6.1 Haastattelututkimus kehitystiimissä

Teknisen ratkaisun ensimmäisen version onnistumista haluttiin tutkia opinnäytetyössä tarkemmin haastattelun keinoin. Haastattelun tarkoituksena oli kerätä palautetta kehitystyön tuloksista ja jatkuvan integraation prosessiin liittyviä jatkokehitysideoita. Haastattelussa käytettiin puolistrukturoitua rakennetta ja avoimia kysymyksiä. Haastattelut käytiin etänä Teams yhteydellä ja kasvotusten. Haastattelussa haluttiin saada vastauksia seuraaviin kysymyksiin: Mitä hyötyjä projektille on siitä, että Robot Framework testit ajetaan osana Azure jatkuvan integraation (CI) prosessia manuaalisen paikallisen ajon sijaan? Mitä haasteita olet kohdannut itse tai projektissa yleensä jatkuvan integraation (CI) työkalun ajaman testiautomaation kanssa? Miten testiautomaatiota voidaan mielestäsi hyödyntää projektissa jatkossa paremmin jatkuvan integraation työkalujen (CI) avulla?

6.2 Vastuuhenkilöiden haastattelu

Projektin vastuuhenkilöitä haastatellessa molemmat haastateltavat pitivät kehitystyön jatkuvan integraation työkalua paikallista ajoa parempana vaihtoehtona sen toimintavarmuuden takia. Paikallisesti ajettuna testit voivat antaa vääriä hälytyksiä johtuen jokaisen kehittäjän oman tietokoneen erilaisista tiloista, versioista ja yksilöllisistä ratkaisuista. Yhdenmukaistettu testiympäristö antaa luotettavampaa tietoa testituloksista. Näistä syistä ulkoistettu ja vakioitu testiautomaatioympäristö jatkuvan integraation työkalussa säästää konkreettisesti ohjelmistokehittäjien työaikaa. Projektin resurssit voidaan siten hyödyntää paremmin ongelmien ratkomisen sijaan liikearvoa tuottavampaan kehitystyöhön. (Solidabis 2022)

Kehitystyön hyötyinä projektille nostettiin esiin parantunutta jatkuvan integraation prosessin arvoa projektille voimavarana. Luotettavaa tietoa tarjoava jatkuvan integraation prosessi tarjoaa arvokasta tietoa projektin koodin laadusta esimerkiksi ohjelman tuotantoon vientiä harkitessa ja ohjelman tulevia ominaisuuksia harkitessa. Tuotantojulkaisuilta tullaankin jatkossa edellyttämään onnistunutta Robot Framework testiraporttia ennen mahdollista julkaisua osana valmiin määritelmää (Definition of Done). Paranneltu testiautomaatio prosessi tarjoaa myös mahdollisuuksia tehdä rohkeampia parannustöitä ohjelmiston arkkitehtuuriin, kun

muutostöitä tehdessä voidaan varmistaa, että ohjelma suorittaa edelleen luotettavasti tehtäviään. (Solidabis 2022)

Haastattelussa ilmeni, että jatkuvan integraatio prosessi ei ole kuitenkaan vielä projektissa optimaalinen. Molemmat haastateltavat ilmaisivat, että kehitystyö ja jatkuvan integraation prosessien laajempi hyödyntäminen oli paljastanut projektissa osaamisvajetta liittyen Devops prosesseihin. Devops-prosesseihin liittyvät kehittämiskohteet ja prosessit ovat projektissa liian harvojen ohjelmistokehittäjien harteilla. Tästä muodostuu riski kehityspanostuksen hukkaamisesta henkilöiden vaihtuessa. Projektin hallinnossa havaittiin parannettavaa, jotta siiloutumista eli tiedon keskittymistä projektissa vain pienempien ryhmien kesken, voitaisiin välttää jatkossa paremmin. (Solidabis 2022)

6.3 Ohjelmistokehittäjien haastattelu

Projektin ohjelmistokehittäjien haastatteluissa korostui projektin yleisten ja hallinnollisten asioiden sijaa tekniset yksityiskohdat. Useat haastatteluista olivat melko lyhyitä, koska jatkuvan integraation prosessit ja Robot Framework testaustyökalu ei ollut projektissa vielä kaikille projektin ohjelmistokehittäjille entuudestaan tuttuja. Haastatellessa havaittiin samaa osaamisvajetta ja tiedon siiloutumista, jonka projektin vastuuhenkilöt nostivat esiin aiemmassa haastattelussa. (Solidabis 2022)

Projektin kaikki ohjelmistokehittäjät nostivat esille, että lisätty automaatio jatkuvan integraation prosessissa koskien systeemitestausta helpottaa heidän päivittäistä työtään. Systeemitestaus osana jatkuvan integraation prosessia poistaa inhimillisen erehdyksen mahdollisuudet ja sen, että testien ajaminen yksinkertaisesti unohtuisi tai jätettäisiin väliin kiireen takia. Testien ajon toimintavarmuus vakioidussa määritellyssä pilviympäristössä nähtiin erinomaisena asiana, ettei kehittäjän tarvitse selvittää niin paljoa haastavia ongelmatilanteita paikallisesti ajettujen testien kanssa. (Solidabis 2022)

Positiivisena nähtiin myös se, että testien ajo voitiin ulkoistaa siten, ettei omaa henkilökohtaista konetta tarvinnut valjastaa testien ajoon, sillä testien ajaminen vie oman tietokoneen resurssit pois muulta työnteolta. Automatisoitu prosessi nähtiin myös hyvänä uudempien ohjelmistokehittäjien kannalta, koska silloin ei tarvitse tietää niin paljon testaustyökalusta ja kehitysympäristöstä, vaan voi tukeutua aluksi suurilta osin automaattisen prosessin tarjoamaan palautteeseen. (Solidabis 2022)

Ohjelmistokehittäjät olivat havainneet ongelmana jatkuvan integraation prosessissa sen, että useat peräkkäiset pyynnöt Pipelinelle jonoutuvat. Testiautomaation ajoajan ollessa pitkä, se jonoutuu helposti ruuhkatilanteessa useiden kehittäjien tallentaessa muutoksia versiohallintaan yhtäaikaisesti. Testien hajotessa Pipelinessa sen selvittely koettiin vaikeammaksi kuin paikallisen ympäristön kanssa. Ohjelmistokehittäjät olivat myös havainneet satunnaista

testien hajoamista tunnistamattomasta syystä. Erilaiset ongelmanratkaisutilanteet jatkuvan integraation työkalun kanssa olivat ongelmallisia aiemman kokemuksen puutteen takia. (Solidabis 2022)

7 Jatkokehitysehdotukset

7.1 Jatkokehitysehdotukset haastatteluiden perusteella

Tehdyissä haastatteluissa kartoitettiin myös jatkokehitysehdotuksia jatkuvan integraation prosessin hyödyntämisessä asiakasprojektissa. Jatkokehityksistä oli havaittavissa yhteys jatkuvan integraation työkalun tarjoamien asioiden parempaan hyödyntämiseen ja prosesseissa kohdattujen ongelmakohtien ratkaisemiseen. Muutamia erinomaisia jatkokehitysideoita tuli myös näiden ulkopuolelta, joista pystyi jalostamaan erittäin konkreettisia jatkotoimenpiteitä. (Solidabis 2022)

Vastuuhenkilöiden ehdottamille jatkokehitysideoille yhteistä oli tehdyn ratkaisun ylläpitoon panostaminen ja sen hyötyjen parempi viestintä sidosryhmille. Parempaa viestintää ja jatkuvan integraation prosessin hyötyjen näkyväksi tekemistä voitaisiin tuoda esiin asiakaspalavereissa, keskusteluissa, suunnitelmissa ja raporteissa. Eräs ehdotus oli monitorointiratkaisu. Toimistolle asennettavalla ruudulla näkyisi raportti ympäristöjen ja testiautomaation tilasta. Jatkuvan integraation testitulokset olisivat näin koko tiimin nähtävillä. Mikäli ongelmia havaitaan, kaivattiin myös suorasukaisempaa automaattista viestintää tiimin pikaviestikanaviin, kuten Slackiin ja kehittäjien sähköpostiin. Tekemällä testituloksia jatkuvasti näkyviksi testauksen tärkeys ja sen hyödyt koko kehitystiimille ja asiakkaalle tulee selvemmäksi ja läpinäkyvämmäksi. (Solidabis 2022)

Ohjelmistokehittäjien ehdottamille jatkokehitysideoille yhteistä oli prosessien yhdenmukaistaminen, helpottaminen ja selkiyttäminen. Systeemitestien paikallista ajoa tulee helpottaa entisestään, jotta virhetilanteiden selvittely testiajon päättyessä virheeseen olisi ohjelmistokehittäjälle helpompaa. Testiautomaatio prosessit tulee olla tehty niin helpoiksi, että uusi kehittäjä tiimissä voi hyödyntää niitä heti luottaen automatiikan toimivuuteen. Osittain kaivattiin tiukempaa linjaa testiautomaation asettamisesta ehdoksi ennen uuden koodin integroimista eri kehityshaaroihin, mutta toisaalta kaivattiin nopeampia prosesseja ja lyhyempiä testien ajoaikoja. (Solidabis 2022)

Haastatteluista kerätyistä jatkokehitystarpeista koskien jatkuvan integraation prosessin hyötyjen näkyväksi tekemistä ehdotan projektissa harkittavaksi Azure Dashboards ominaisuuden käyttöönottoa Microsoftin Azure Devops tuotteessa, joka on projektissa jo käytössä (Microsoft 2022). Azure Dashboard on visuaalinen dynaaminen taulu, johon voidaan liittää muokattavia näkymiä ja widget pienoisohjelmia. Azure Dashboard voi yhtäaikaisesti toimia toimistolla

visuaalisena työkaluna kehitystiimille TV-ruudusta esitettynä ja informaation lähteenä asiakkaalle ja muille sidosryhmille ympäristöjen tilasta. Azure Dashboards mahdollistaa Azure resurssien jatkuvan monitoroinnin ja testitulosten trendien seurannan Azure Devops Services – Test results trend widget tuotteen avulla, johon jatkuvan integraation testitulokset voidaan liittää (Microsoft 2022). Azure Dashboardiin voidaan myöhemmin lisätä myös paljon muuta kehitys-, testi- ja tuotantoympäristön monitorointia, kuormituksen ja tietoturva-asioiden seurantaa, projektissa tärkeäksi koettujen tarpeiden mukaan.

Jatkuvan integraation testiautomaation havaitsemat testihavainnot ja epäonnistuneet ajot voidaan tehdä näkyväksi tiimille myös muutamalla muulla konkreettisella tavalla. Azure Devops tuote esimerkiksi integroituu helposti projektin hyödyntämiin pikaviestimiin, kuten Teamsiin ja Slackiin Azure Pipelines with Slack integraation avulla. Projektissa tarpeelliseksi koetut ilmoitukset voidaan lähettää tämän integraation kautta ja sen viestintä voidaan muokata sellaiseksi, kun koetaan tarpeelliseksi (Microsoft 2022).

Jatkuvan integraation prosessissa tapahtuvat testihavainnot tulee voida replikoida myös paikallisesti kehittäjän omalla tietokoneella. Tätä koskeva dokumentaatio ja ohjeet ovat vielä puutteellisia. Haastatteluissa kävi ilmi, että kehittäjät edelleen ajavat Robot Framework testejä omalla tietokoneellaan vanhalla tavalla Kubernetes klusterin ulkopuolelta. Jatkuvan integraation prosessi suorittaa Robot Framework testit Kubernetes klusterin sisäpuolella. Testit tulisi suorittaa identtisesti, jotta testihavainnot voidaan toistaa samalla tavalla. Tämä oli aiheuttanut hämmennystä ja turhaa selvittelytyötä koska testien erilainen suorittamistapa altistaa testiympäristön erilaisille konfiguraatioille ja niistä johtuville testitulosten erilaisuudelle. Dokumentaatio koskien Robot Framework testauksen ajamista paikallisen Kubernetes klusterin tulee siis vielä päivittää. Parempi dokumentaatio ja ohjeet auttaisivat projektitiimin osaamisen kasvattamisessa ja henkilösidonnaisuuden vähentämisessä.

7.2 Jatkokehitysehdotukset tietoperustan perusteella

Projektin jatkuvan integraation prosessia ei ole asiakasprojektissa vielä laajennettu jatkuvan käyttöönoton prosessiin. Tällä hetkellä projektin kaikki asennukset suoritetaan käsin kaikkiin yleisiin testi- ja tuotantoympäristöihin. Käsin tehtävä työ on virhealtista ja kallista asiakasorganisaatiolle. Mikael Kriefin kirjassa *Learning Devops* esiteltiin jatkuvan integraatio mallin jatkumona jatkuvan käyttöönoton prosessi. Kirjassa kuitenkin esitetään jatkuvan käyttöönoton prosessin käyttöönoton ennakkovaatimukseksi kypsää ja optimoitua jatkuvan integraation prosessia. (Krief 2021, luku 1)

Nähdäkseni projektissa voidaan saavuttaa vielä paljon kustannustehokkuutta, mikäli nykyinen jatkuvan integraation prosessi laajennettaisiin myös jatkuvan käyttöönoton prosessiksi. Jatkuvan integraation hyödyntäminen ja sitä koskeva osaaminen tulee ensin vakiinnuttaa projektissa, mutta mielestäni olisi järkevää ottaa vielä selvää jatkuvan käyttöönoton prosessin

hyödyistä asiakasprojektille. Ensimmäinen helppo askel olisi automatisoida testiympäristön käyttöönottoasennukset onnistuneen jatkuvan integraation prosessin jälkeen. En näe tuotantoasennusten automatisointia vielä realistisena tavoitteena, sillä nykyisen testiautomaation kypsyysaste ei riitä niin pitkälle vietyyn Devops prosessiin.

8 Yhteenveto

Opinnäytetyössä perehdyttiin laajasti alan kirjallisuuteen koskien Devops ja testiautomaatioprosesseja. Noudattaen alan parhaita käytäntöjä asiakasprojektin systeemitestit liitettiin osaksi projektin olemassa olevaa jatkuvan integraation prosessia Microsoft Azure Devops palvelussa. Teknisessä ratkaisussa hyödynnettiin Docker alustaratkaisua testiautomaation automaattiseen ajoon Kubernetes ympäristössä. Automatisoitu testien suoritus oli paikallista testausta toimintavarmempi tapa. Testausraportointi on nyt kattavammin koko kehitystiimin hyödynnettävissä. Parantunut automaatio mahdollistaa projektissa nopeampaa kehitystyötä laadusta tinkimättä. Automaation ansiosta testaus ei jää vahingossakaan suorittamatta ja raportoimatta esimerkiksi tilanteessa, jossa testiautomaatiokehittäjät ovat lomalla.

Ratkaisun hyödyllisyyttä validoitiin projektitiimissä haastattelututkimuksen keinoin. Tietoperustan ja haastattelujen perusteella pystyttiin tunnistamaan useita hyötyjä ja haasteita koskien jatkuvan integraation prosessia asiakasprojektissa. Opinnäytetyön seurauksena asiakasprojektissa tietoisuus testiautomaatioon liittyvistä prosesseista kasvoi. Opinnäytetyö jätti jälkeensä uudistuneita verkostoja projektin ohjelmisto-, Devops- ja testiautomaatiokehittäjien välille syventyneenä yhteistyönä jatkuvan integraation prosessin ollessa keskiössä.

Alan kirjallisuuden ja haastatteluiden perusteella opinnäytetyössä voitiin perustellusti esittää uusia konkreettisia ja hyödyllisiä jatkokehitysideoita asiakasprojektissa. Esitetyt jatkokehitysideat auttaisivat hyödyntämään jatkuvan integraation prosessin hyötyjä yhä laajemmin tai auttaisivat ratkaisemaan koettuja ongelmakohtia. Jatkuva integraatio on ohjelmistotuotannon tärkeä tukiprosessi, ja saavutettuja hyötyjä tulee pyrkiä hyödyntämään entistä paremmin. Kehitettyä prosessia tulee suojata, ettei sen arvo päivittäiselle kehitystyölle unohdu kiireen tai osaamispuutteen takia. Havaitut ongelmat jatkuvan integraation prosessin hyödyntämisessä tulee ratkaista. Asiakasprojektissa tulee harkita jatkuvan integraation ja käyttöönoton prosesseja koskevia jatkokehitysideoita.

Lähteet

Painetut

Evans, E 2003. Domain-Driven Design: Tackling Complexity in the Heart of Software. Boston: Addison-Wesley Professional.

Sähköiset

Agarwal, G, 2021. Modern Devops Practises. E-kirja. Birmingham: Packt Publishing, Limited

Chopra, R. 2018. Software Quality Assurance. E-kirja. Herndon: Mercury Learning & Information

Docker 2022. Dockerfile reference. Viitattu 15.4.2022. https://docs.docker.com/engine/reference/builder/

Docker 2022. Overview of Docker Compose. Viitattu 17.4.2022. https://docs.docker.com/compose/

Finanssiala 2022. Verkkolaskutus (Finvoice). Viitattu 14.4.2022. https://www.finanssiala.fi/aiheet/verkkolaskutus-finvoice/#/

GNU 2022. What is shell? Viitattu 17.4.2022. https://www.gnu.org/software/bash/manual/html_node/What-is-a-shell_003f.html#What-is-a-shell_003f

Git 2022. Git – Distributed is the new centralized. Viitattu 13.4.2022. https://git-scm.com/

Git 2022. Git-commit. Viitattu 15.4.2022. https://git-scm.com/docs/git-commit

Helm 2022. Helm – The package manager for Kubernetes. Viitattu 15.4.2022. https://helm.sh/

Helm 2022. Values files. Viitattu 17.4.2022. https://helm.sh/docs/chart_template_guide/values_files/

Helsingin Yliopisto 2022. Lähdekritiikki. Viitattu 14.4.2022. https://blogs.helsinki.fi/opiskelijan-digitaidot/3-tiedonhankinta/3-4-loydetyn-tiedon-kaytto-ja-arviointi/lahdekritiikki/

Javatpoint. Zuul API Gateway. Viitattu 20.4.2022. https://www.javatpoint.com/zuul-api-gateway

Jenkins 2022. Jenkins – Build great things at any scale. Viitattu 13.4.2022. https://www.jenkins.io/

Jyväskylän Ylipisto 2021. Haastattelut. Viitattu 14.4.2022. https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/aineistonhankintamenetelmat/haastattelut

Jyväskylän Yliopisto 2015. Kokonaistutkimus, otanta ja harkinnanvarainen näyte. Viitattu 20.4.2022. https://koppa.jyu.fi/avoimet/hum/menetelmapolkuja/menetelmapolku/aineistonhankintamenetelmat/kokonaistutkimus-otanta-ja-harkinnanvarainen-naeyte

Krief, M. 2021. Learning Devops. E-kirja. Birmingham: Packt Publishing, Limited

Kubernetes 2022. Kubernetes website. Viitattu 13.4.2022. https://kubernetes.io/

Madamanchi, S. 2019. Understanding Docker Container Exit Codes. Blogikirjoitus. Better Programming. Viitattu 15.4.2022. https://betterprogramming.pub/understanding-docker-container-exit-codes-5ee79a1d58f6

Microsoft 2022. Azure Command-Line Interface (CLI) documentation. Viitattu 8.5.2022. https://docs.microsoft.com/en-us/cli/azure/

Microsoft 2022. Azure Container Registry. Viitattu 8.5.2022. https://azure.microsoft.com/en-us/services/container-registry/

Microsoft 2022. Azure – Invent with purpose. Viitattu 13.4.2022. https://azure.microsoft.com/en-us/

Microsoft 2022. Azure Pipelines. Viitattu 13.4.2022. https://docs.microsoft.com/en-us/azure/devops/Pipelines/get-started/what-is-azure-Pipelines?view=azure-devops

Microsoft 2022. Build and push Docker images to Azure Container Registry. Viitattu 20.4.2022. https://docs.microsoft.com/en-us/azure/devops/Pipelines/ecosystems/containers/acr-template?view=azure-devops

Microsoft 2022. Create a dashboard in the Azure portal. Viitattu 20.4.2022. https://docs.microsoft.com/en-us/azure/azure-portal/azure-portal-dashboards

Microsoft 2022. Configure the Test Results Trend (Advanced) widged. Viitattu 20.4.2022. https://docs.microsoft.com/en-us/azure/devops/report/dashboards/configure-test-results-trend?view=azure-devops

Microsoft 2022. Docker Compose task. Viitattu 8.5.2022. https://docs.microsoft.com/en-us/azure/devops/pipelines/tasks/build/docker-compose?view=azure-devops

Microsoft 2022. Package and Deploy Helm Charts task. Viitattu 15.4.2022. https://docs.microsoft.com/en-us/azure/devops/Pipelines/tasks/deploy/helm-deploy?view=azure-devops

Microsoft 2022. Publish Test Results task. Viitattu 15.4.2022. https://docs.microsoft.com/en-us/azure/devops/Pipelines/tasks/test/publish-test-results?view=azure-devops&tabs=trx%2Cyaml

Microsoft 2022. Review continuous test results after build. Viitattu 15.4.2022. https://docs.microsoft.com/en-us/azure/devops/Pipelines/test/review-continuous-test-results-after-build?view=azure-devops

Microsoft 2022. Azure Pipelines with Slack. Viitattu 20.4.2022. https://docs.microsoft.com/en-us/azure/devops/Pipelines/integrations/slack?view=azure-devops

Microsoft 2022. YAML schema reference for Azure Pipelines. Viitattu 15.4.2022. https://docs.microsoft.com/en-us/azure/devops/Pipelines/yaml-schema/?view=azure-Pipelines

Microsoft 2022. View Pipeline reports. https://docs.microsoft.com/en-us/azure/devops/Pipelines/reports/Pipelinereport?view=azure-devops

Näreaho, S., Kettunen, J., Kärki, A. & Päällysaho, S. 2020. Ammattikorkeakoulujen opinnäytetöiden eettiset suositukset. Viitattu 14.4.2022. http://www.arene.fi/wp-

content/uploads/Raportit/2020/Arenen%20ONT%20eettiset%20ohjeet%20esitysmateriaali%202020.pdf?_t=1578486373

Pacheco V. F. 2018. Microservice Patterns and Best Practises. E-kirja. Birmingham: Packt Publishing, Limited

Python 2022. Python. Viitattu 15.4.2022. https://www.python.org/

Rajput, D. 2018. Hands-On Microservices - Monitoring and Testing: A Performance Engineer's Guide to the Continuous Testing and Monitoring of Microservices. E-kirja. Birmingham: Packt Publishing, Limited.

Red Hat 2022. What is YAML? Viitattu 10.5.2022. https://www.redhat.com/en/topics/automation/what-is-yaml

Robot Framework 2022. Robot Framework. Viitattu 13.4.2022. https://robotframework.org/

Robot Framework 2022. Supported file formats. Viitattu 15.4.2022. https://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html#supported-file-formats

Spillner A., Linz T. 2021. A Study Guide for the Certified Tester Exam- Foundation Level-ISTQB® Compliant. E-kirja. Heidelberg: Dpunkt.verlag

VMWare 2022. Spring Boot. Viitattu 14.4.2022. https://spring.io/projects/spring-boot

Winscp 2022. SFTP (SSH File Transfer Protocol). Viitattu 15.4.2022.
https://winscp.net/eng/docs/sftp

Julkaisemattomat

Solidabis 2022. Projektin vastuuhenkilöiden haastattelu 19.4.2022. Solidabis Solutions Oy. Helsinki.

Solidabis 2022. Projektin ohjelmistokehittäjien haastattelu 19.4.2022. Solidabis Solutions Oy. Helsinki.

Kuviot

Liitteet

Liite 1: Haastattelukysymykset

Mitä hyötyjä projektille on siitä, että Robot Framework testit ajetaan osana Azure jatkuvan integraation (CI) prosessia manuaalisen paikallisen ajon sijaan?

Mitä haasteita olet kohdannut itse tai projektissa yleensä jatkuvan integraation (CI) työkalun ajaman testiautomaation kanssa?

Miten testiautomaatiota voidaan mielestäsi hyödyntää projektissa jatkossa paremmin jatkuvan integraation työkalujen (CI) avulla?