Käyttöliittymän regressiotestauksen automatisointi Katalon Studiolla

Markus Sahrakorpi

2019 Laurea

Laurea-ammattikorkeakoulu

Käyttöliittymän regressiotestauksen automatisointi Katalon StudiollaKäyttöliittymän regressiotestauksen automatisointi Katalon Studiolla

Markus Sahrakorpi
Tietojenkäsittely
Opinnäytetyö
Maaliskuu, 2019 2019

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

Markus Sahrakorpi

Käyttöliittymän regressiotestauksen automatisointi Katalon Studiolla

2019 2019 Sivumäärä 32

Ketterät kehitysmenetelmät mahdollistavat laajan ja monimutkaisen ohjelmiston kehityksen erittäin nopeasti, mutta samalla laadun varmistaminen vaikeutuu. Tämän opinnäytetyön tavoitteena on kehittää Visma Moveniumin testausautomaatiota kehittämällä käyttöliittymän regressiotestauksen automaatio Katalon Studiolla. Tavoitteena on saada manuaaliseen julkaisutestaamiseen kuluva aika tippumaan neljästä tunnista alle puoleen tuntiin. Tämä mahdollistaa testaajien resurssien paremman hyödyntämisen uusien toiminnallisuuksien kehittämisessä.

Tämä opinnäytetyö on toiminnallinen, joten tuotoksena on tuotos. Testausautomaation testien kehittämiseen käytettiin Scrumia, minkä iteratiivinen malli mahdollistaa nopeaa ja jatkuvaa palautetta testien laadusta ja etenemisestä. Testit perustuvat Moveniumin olemassa olevaan julkaisutestaamisen dokumentaatioon, missä on määritelty ohjelmiston eniten käytetyn ja tärkeimmät toiminnallisuudet.

Testien valmistuttua, Movenium luopui kokonaan julkaisutestaamisesta ja siirtyi jatkuvan julkaisuun malliin, missä ohjelmistoa päivitetään jatkuvasti ilman erillistä hallittua julkaisua. Testausautomaation kehittämisen hyödyt ovat näkyneet Moveniumilla erityisesti säästetyssä ajassa, ja järjestely on myös mahdollistanut resurssien kohdistamisen toiminnallisuuksien jatkokehitykseen.

Täysin automatisoitua regressiotestauksen prosessia ei työn lopputuloksena kyetty luomaan. Työn lopputuloksena syntyneet automaatiotestit suoritetaan kaksi kertaa päivässä, ja manuaalista testausta suoritetaan tarpeen mukaan pohjautuen riskiarvioon. Jatkokehityksessä jokaisen ohjelmistopäivityksen yhteydessä suoritetaan regressiotesti, jonka tavoitteena on varmistaa ohjelmiston laatu.

Asiasanat: Ohjelmistotestaus, Regressiotestaus, Testausautomaatio, Katalon

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

Markus SahrakorpiMarkus Sahrakorpi

Automating UI Regression Testing with Katalon Studio

Year 2019 Pages 32

Agile software development methods enable the development of extensive and complex software, but in return ensuring their quality becomes increasingly difficult. The purpose of this thesis is to further develop Visma Movenium's test automation by including automated UI regression testing using Katalon Studio. The goal is to reduce the time needed for manually testing new production releases from about four hours down to under thirty minutes. This will allow testers' resources to be allocated over from testing to software development.

This thesis is a functional thesis which produces a concrete work. The test automation scripts were developed using Scrum as the agile development method, which allows for fast and continuous feedback about the quality and progress of the tests. The test scripts are based on existing documentation on manual acceptance testing, which includes test cases for the software's most used and critical functionalities.

After the test scripts were all completed, Movenium decided to completely get rid of manual release testing, and in turn incorporated continuous delivery as part of their development model. The benefits of developing better test automation has resulted in a lot of saved time, and Movenium has been able to focus more resources on developing new features.

This thesis did not fully manage to automate UI regression testing, as the tests are only run twice a day automatically and manually when deemed necessary by risk assessment. For the future, every software update should be regression tested in order to ensure the software's quality.

Keywords: Software testing, Regression testing, Test automation, Katalon

Sisällys

1 Johdanto 7
1.1 Kohdeyritys 7
1.2 Opinnäytetyön tavoite 7
2 Ohjelmistotestaus 8
2.1 Ohjelmiston laadun varmistaminen 9
2.2 Ketterä testaaminen 11
2.3 Testauksen automatisointi 13
2.4 Automatisoinnin hyödyt 15
2.5 Automatisoinnin haasteet ja riskit 16
2.6 Regressiotestaus 17
3 Kehitysympäristö 18
3.1 Testauksen kohde 18
3.2 Kehitysmenetelmä 20
4 Katalon 21
4.1 Katalon Studion lisäosat 22
4.2 Asennus ja käyttöönotto 23
4.3 Automaatioprojektin rakenne 25
4.4 Testien kirjoittaminen 26
4.5 Testien suorittaminen 27
4.6 Testitulosten tarkastelu ja analyysi 27
5 Lopputulokset 28
5.1 Yritykselle tuotettu hyöty 28
5.2 Testien jatkokehitys 29

Lyhenteet ja termit

CI -palvelin	Continuous Integration -palvelin. Sen tehtävä on suorittaa automaattisesti ohjelmiston rakennukseen liittyvät toimet (lähdekoodin rakentaminen, testien ajaminen ym.)
IDE	Integrated development environment. Ohjelmisto, jolla kehitetään ohjelmistoja. Useimmiten IDE tarjoaa lähdekoodin oikeudellisuustarkastajan, sekä muita ohjelmistokehitystä tukevia toimintoja.
ISO/IEC	International Organization for Standardization ja International Electrotechnical Commission. Ne kehittävät ja ylläpitävät IT-alan standardeja.
ISTQB	International Software Testing Qualifications Board. Tarjoaa ohjelmistotestauksen liittyviä sertifikaatteja.
SDK	Software development kit (devkit). Ohjelmistokehityksen työkaluja, jotka mahdollistavat ohjelmistokehityksen tietylle alustalle.
TT4	Time Tracker 4. Visma Moveniumin kehittämän ohjelmiston tekninen nimitys.

1 Johdanto

Nykyiset ketterät ohjelmistokehitysmenetelmät kykenevät tuottamaan nopeasti ja mukautuvasti isoja ja monimutkaisia ohjelmistoja, ja näiden ohjelmistojen käyttäjät vaativat laatua, toimivuutta ja käytettävyyttä. Ohjelmistojen jakaminen ja kilpailevien yritysten ohjelmistojen kokeileminen on helpottunut, joten yritykseen kohdistuneet paineet tuottaa laatua ovat kovat. Hyvän ja laadukkaan ohjelmiston aikaansaamiseksi sitä täytyy testata suunnitelmallisesti ja perusteellisesti, mikä puolestaan nostaa kehityksen kustannuksia. Kehitykseen ja testaamiseen käytetty aika on rajallista, joten tietokoneiden hyödyntäminen testaamisessa on järkevä ratkaisu. Testausautomaatio voi olla ratkaisu, koska tietokoneen tehokkuus ja toimintavarmuus vapauttaa henkilöstöresursseja ja tukee kehittäjiä työssään, jotta he voivat kehittää laadukkaampia ohjelmistoratkaisua.

1.1 Kohdeyritys

Visma Movenium on SaaS-yritys (Software as a Service), joka on perustettu vuonna 2005 ja on osa Visma Software Oy:tä. Yritys työllistää Suomessa noin 20 henkilöä, ja Ruotsissa viisi.

Movenium tarjoaa asiakkailleen digitaalisia työkaluja työmaan johtamiseen, raportointiin sekä hallinnointiin. Tavoitteena on digitalisoida ja automatisoida työmaahan liittyvää byrokratiaa helpottaakseen työmaiden etenemistä ja valmistumista ajallaan. Movenium tarjoaa käyttäjilleen muun muassa työajanseurantaa, kulunseurantaa, verottajaraportointia, työmaapäiväkirjoja, laatumittareita, sekä integraatioita muihin sähköisiin järjestelmiin, kuten palkkavientiin. (Visma Movenium 2018)

1.2 Opinnäytetyön tavoite

Tämän opinnäytetyön tarkoituksena on kehittää yrityksen automaatiotestausta ottamalla käyttöön käyttöliittymän regressiotestaus. Tavoitteena on pyrkiä vähentämään manuaaliseen testaamiseen kuluvaa aikaa ja siirtää resursseja uuskehityksen puolelle. Nykyisellä mallilla ennen uuden ohjelmistopäivityksen julkaisua testaajilla menee käyttöliittymän testaamiseen ja toiminnan varmistamiseen noin neljä tuntia. Tavoitteena on tiputtaa julkaisutestaukseen kuluva aika alle puoleen tuntiin vuoden 2018 loppuun mennessä.

Opinnäytetyö on toiminnallinen, joten tuloksena syntyy tuotos. Toiminnallisen tutkimuksen piirteisiin kuuluu, että toteutus ja raportointi tapahtuu käyttämällä tutkimusviestinnän keinoja. On tärkeää ilmoittaa tutkimuksen tavoite, kohde, aineisto ja menetelmät, sekä tutkimuksen päätulokset ja päätelmät. Toiminnallinen opinnäytetyö on kaksiosainen kokonaisuus, joka sisältää toiminnallisen osuuden eli produktin sekä opinnäytetyö-raportin eli prosessin dokumentoinnin ja arvioinnin. (Vilkka & Airaksinen 2003, 9)

2 Ohjelmistotestaus

Ihmisen toiminnalle on luonteenomaista erehtyä ja tehdä virheitä. Jotta ihmisen tuottamasta tuotoksesta tulisi mahdollisimman laadukas ja tehokas, sitä täytyy tarkastella kriittisesti ja korjata epätehokkaat ja epäonnistuneet kohdat. Ohjelmistokehityksessä ohjelmiston laatua ja tarkoituksenmukaista suorituskykyä voi mitata testaamalla, ja toistamalla viankorjaamisen ja testauksen sykliä laatua voidaan kehittää. Hassin (2008, 80) mukaan testaamisen tarkoituksena on kerätä tietoa, minkä avulla voidaan tehdä selkeitä ja johdonmukaisia päätöksiä.

Testaaminen voidaan jakaa manuaalisen ja automatisoituun testaamiseen. Manuaalisessa testaamisessa ihminen suorittaa ohjelmiston testaamisen itse, ja pyrkii toteamaan ohjelmiston toimivuuden. Manuaalinen testaaminen tapahtuu seuraamalla annettua testaussuunnitelmaa, mitkä ohjaavat testaajaa ja rajaavat testauksen pinta-alaa. Automatisointi puolestaan pyrkii siirtämään testaamisen tietokoneelle, jotka suorittavat toistuvia ja ennalta määritettyjä testitapauksia. (Kasurinen, Taipale & Smolander 2009, 1)

ISTQB (2018, 13-14) määrittelee testauksen tavoitteita, joista Kasurisen (2013, 39) mukaan tärkein on vikojen ja virheiden löytäminen ja estäminen. Kasurinen (2013, 39) määrittelee virheen tai erehdyksen olevan lähdekoodissa tai dokumentaatiossa ilmenemänä poikkeamana, mikä on ihmisen aiheuttama. Virhe voi aiheuttaa vian, jolloin ohjelmisto ei suoriudu halutulla tavalla. Vian Kasurinen (2013, 39) puolestaan määrittelee olevan ohjelmiston tai dokumentaation poikkeama, mikä estää sen suoriutumasta oikein aiheuttaen häiriön. Häiriön Kasurinen (2013, 39) kertoo olevan tilanne, missä ohjelmisto ei toimi kuten sen pitäisi. Ohjelmistossa tai dokumentaatiossa olevien virheiden takia syntyy vikoja, joiden takia syntyy häiriöitä, jotka haittaavat ohjelmiston toimintaa.

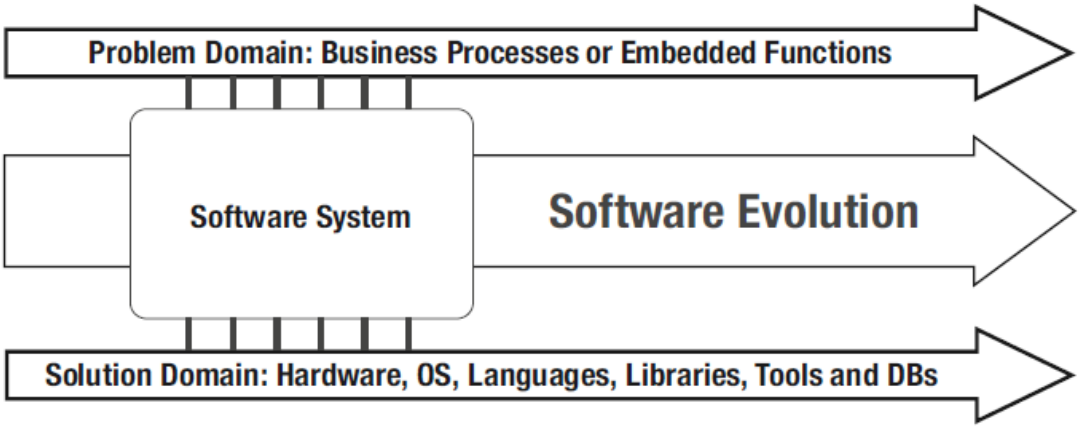
Kasurinen (2013, 39) korostaa, että testaamisessa ei ole kyse pelkästään ohjelmiston käyttämisestä ja toiminnallisuuden varmistamista. Tarkoituksena on myös suunnitella, raportoida ja analysoida ohjelmiston toimintakykyä, ja pyrkiä löytämään vikoja ohjelmistosta erilaisilla lähestymistavoilla. ISTQB (2018, 14) tarkentaa tätä ajatusta toteamalla, että testaamisen tarkoituksena on saada varmuutta ja lisätä luottamusta ohjelmiston laatutasoon. Tämän myötä ohjelmiston toimivuudesta ja laadusta saadaan tärkeää tietoa myös tulevaisuuden päätösten tekemiseen.

ISTQB (2018, 14) kuitenkin huomauttaa, että virheiden jäljitys eli debuggaus ei ole testaamista. Testaamisen tarkoituksena on löytää ja todeta virheitä, kun taas debuggaus on prosessi missä löydetään, analysoidaan ja poistetaan virheen syy. Virheiden jäljittämisen jälkeen (debuggaus) testaaminen varmistaa, että virhe on korjattu. Perinteisesti tiimissä testaaminen on testaajien vastuulla, kun taas debuggaus on ohjelmoijien vastuulla.

2.1 Ohjelmiston laadun varmistaminen

Ohjelmistoja tuottavalle yritykselle on elintärkeää, että asiakkaalle tuotettu ratkaisu on toimiva ja käyttökelpoinen. Markkinoilla vallitsee kova kilpailu, ja asiakkaat etsivät luonnollisesti parhaiten toimivan ja laadukkaimman ohjelmistoratkaisun. Jotta yritys voi varmistaa kehittämänsä ohjelmistonsa laadun, se täytyy testata suunnitellusti ja hallitusti.

Wagnerin (2013, 1) mukaan laatu ei ole kiinteä ja yksiselitteinen ohjelmiston ominaisuus, vaan se määrittyy sen omistajien tavoitteiden ja ohjelmiston kontekstin ja käyttötarkoituksen kautta. Kun korkealaatuinen ohjelmisto tahdotaan rakentaa, täytyy ensiksi määritellä tarkkaan ja selvästi laadun käsite sen kyseisen ohjelmiston kontekstissa. Wagner (2013, 1) selittää, että ohjelmisto kulkee koko ajan ongelman ja ratkaisun välimaastossa. Ongelmaa ohjaa yrityksen toiminnot, kuten prosessit, ja ratkaisua ohjaa tekniset ratkaisut ja toteutukset. Ajan myötä yritykset muuttuvat, tietotekniikka vanhenee ja uusia työkaluja julkaistaan, joten jos ohjelmisto ei reagoi näihin muutoksiin, sen laatu kärsii.



Kuvio 1: Ohjelmisto kulkee ongelman ja ratkaisun välissä (Wagner 2013, 1)

Laadun ylläpito ja varmistaminen vaatii, että asiakkaan odotukset ja vaatimukset ymmärretään perusteellisesti, ja että ne osataan toteuttaa ohjelmistossa. Wagnerin (2013, 10) mukaan ISO/IEC 25010 on määritellyt ohjelmiston laadun standardin, mikä sisältää kohtia muun muassa ohjelmiston sopivuuteen, toimintavarmuuteen, sopivuuteen, tehokkuuteen, käytettävyyteen, turvallisuuteen, huollettavuuteen, siirrettävyyteen ja yhteensopivuuteen.

Ohjelmiston sopivuudella tarkoitetaan, että ohjelmisto tarjoaa toimintoja, jotka täyttävät käyttäjän tarpeet ja odotukset. Tämä tarkoittaa myös, että ohjelmisto tekee toimintansa oikein ja vaatimusten mukaisesti. Wagnerin (2013, 10) mukaan tämä on monilla laadukkaan tuotteen määritelmä, mutta se on todellisuudessa vain yksi osa laadun kokonaisuutta. Toimintavarmuuden lisäksi ohjelmiston on oltava luotettava. Toimintavarma ohjelmisto on myös

luotettava ja päin vastoin, mutta luotettavuutta pystyy matemaattisesti mittaamaan erilaisilla mittareilla ja statistiikalla. (Wagner 2013, 11)

Ohjelmiston tulisi myös olla tehokas ja käytettävä. Wagnerin (2013, 11) mukaan tehokkaan ohjelmiston täytyy reagoida nopeasti ja arvattavasti käyttäjään, mikä puolestaan lisää käyttäjän tyytyväisyyttä. Nämä molemmat käsitteet ovat keskeisiä tuotteen laadussa, koska jos ohjelmistosta puuttuu ominaisuuksia, se ei ole käyttäjälle yhtä hyödyllinen. Jos ohjelmisto on hidas reagoimaan tai esittää tietoja sekavasti, se myös vähentää käyttäjälle tuotettua hyötyä. Ohjelmiston tehokkuus, käytettävyys, toiminnallisuus ja sopivuus ovat käyttäjäkokemuksen kannalta keskeisiä ja kriittisiä, ja niistä koostuu asiakkaan silmistä katsoen ohjelmiston laatu. (Wagner 2013, 11)

Vaikka tässä vaiheessa käyttäjien näkökulmasta tuotteen laatukriteerit täyttyvät, ne eivät kuitenkaan vielä kerro koko totuutta laadun käsitteestä. Ohjelmistossa tapahtuu taustalla asioita, joita käyttäjä ei huomaa. Wagner (2013. 11) kertoo, että ohjelmistoa täytyy huoltaa ja päivittää, koska tekniikka vanhenee, kalusto ikääntyy ja työkalut kehittyvät. Tämän takia ohjelmistoon tai sen ympärillä toimivaan ympäristöön tulisi olla helppo tehdä muutoksia, jotta ohjelmiston elämänkaari pysyy pitkänä. Tämä tarkoittaa myös sitä, että ohjelmiston koodiperustan tulisi olla mahdollisimman yksinkertaista ja ymmärrettävää, jotta sitä on helppo muokata tilanteiden ja tarpeiden mukaan, ja jotta ohjelmistoa olisi aina helppoa jatkokehittää. (Wagner 2013, 11)

Ohjelmiston laatuun vaikuttaa myös ohjelmiston siirrettävyys ja yhteensopivuus, eli kuinka hyvin ohjelmisto toimii eri alustoilla tai systeemeillä. Siirrettävyys keskittyy enemmän siihen, kuinka vahvasti ohjelmisto on sidottuna yhteen alustaan kiinni. Tilanteesta ja tarpeesta riippuen ohjelmiston laatua saattaa parantaa sen sitovuus yhteen alustaan ja siirrettävyyden puutteellisuus, kun taas toisessa tilanteessa se saattaa haitata sitä. Ohjelmisto on voitu tarkoittaa vain yhteen tiettyyn tilanteeseen yhteen tiettyyn alustaan, jolloin siirrettävyyden kriteeri laadussa on olematon. Wagnerin (2013, 12) mukaan yhteensopivuus puolestaan keskittyy enemmän ohjelmiston taustalla tapahtuvaan toimeen, kuten komponenttien tai palveluiden integroimiseen, tai esimerkiksi pilvipalveluiden hyödyntämiseen. Nämä eivät välttämättä näy suoraan päältäpäin katsottuna käyttäjälle, mutta ovat siitä huolimatta mukana laadun käsitteessä. (Wagner 2013, 12-18)

Movenium on hyödyntänyt ohjelmiston testaamisessa manuaalista testausta, joten testaus- ja laatuprosessiin on kartoitettu ja dokumentoitu vain ohjelmiston yleisiä käyttötapauksia. Manuaalisen testauksen aikana testi-insinööri on testannut läpi kaikki dokumentin mukaiset käyttötapaukset, sekä uusimmat ohjelmiston ominaisuudet. Käyttötapaukset on dokumentoitu askel askeleelta, joten niiden perusteella on hyvä lähteä suunnittelemaan uusia testejä ja toteuttamaan niitä. Koska Movenium käyttää kehityksessään ketterää kehitysmenetelmää,

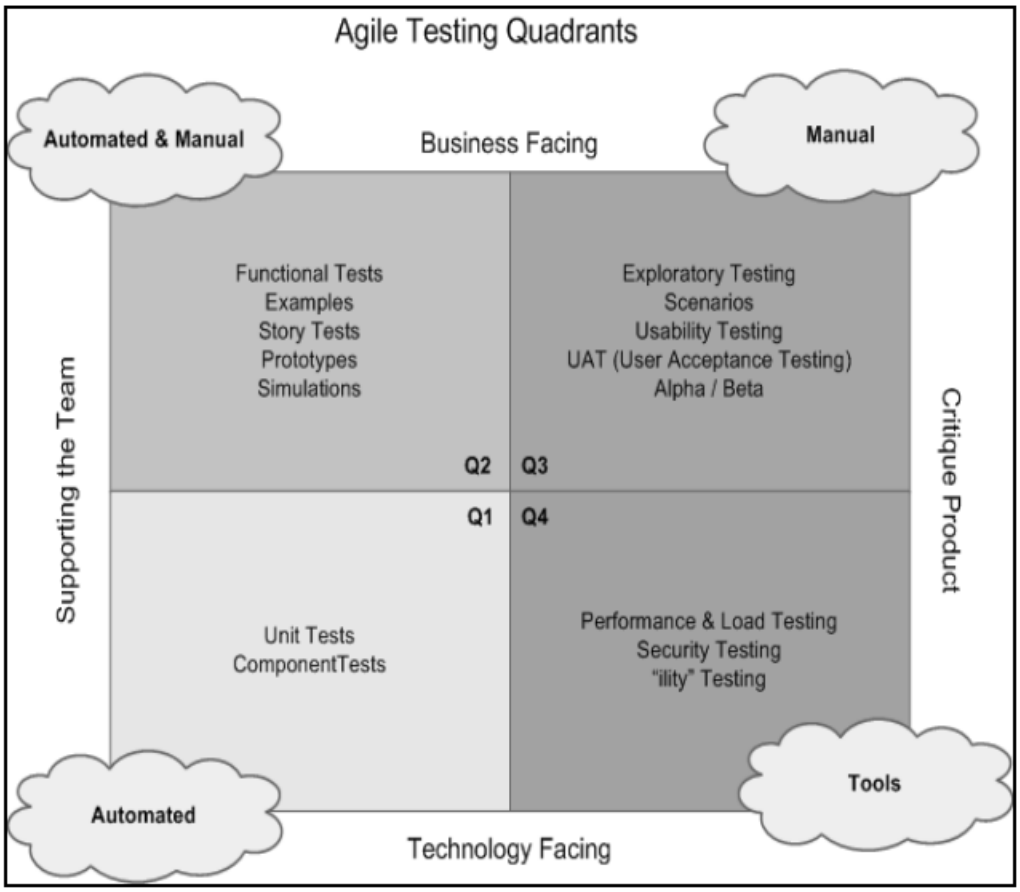
Scrumia, testaamisen ja laadun varmistaminen ei ole samanlaista kuin voisi perinteisessä mielessä kuvitella. Vaikka Moveniumilla on määritetty testi-insinöörit, testaaminen ei ole yksin heidän vastuullaan, vaan koko tiimi on osallisena testaamisessa ja laadun varmistamisessa. Ketterän kehitysmenetelmän tärkeä osa on sisäänrakennettu testaus, ketterä testaus.

2.2 Ketterä testaaminen

Myers, Badgett ja Sandler (2012, 178) määrittää ketterän testaamisen olevan yhteistyötä, missä jokainen on osallisena ja vastuussa testien suunnittelussa, toteuttamisessa ja suorittamisessa. Asiakkaat saavat testata uusia ominaisuuksia ja he antavat niistä palautetta, ja sen kautta kehitystiimi voi kartoittaa käyttötapauksia. Kehittäjät ovat myös mukana testiprosessissa yhteistyössä testaajien kanssa kehittämällä automaatiotestejä, jotta testaamisen laatu voidaan varmistaa. Ketterä testaaminen on siis prosessi, mikä vaatii koko tiimin osallistumista. Ketterässä ohjelmistokehityksessä laatu ja sen varmistaminen kuuluu koko kehitystiimille.

Myers ym. (2012, 178) mukaan ketterässä testaamisessa testausautomaatiolla on keskeinen rooli. Ketterä kehitysmenetelmä vaatii nopeaa ja jatkuvaa palautetta, koska syklit ovat nopeita ja aika on arvokasta. Manuaalinen testaaminen vie paljon aikaa, ja automaattiset testit ovat luotettavimpia, koska niitä voi suorittaa nopeasti ja jatkuvasti samalla tavalla. Myös asiakkaiden palautteen saaminen mahdollisimman aikaisin kehitysvaiheessa on ketterässä testaamisessa tärkeää, ja sen pitäisi olla integroitu osa kehitystä.

Collins, Dias-Neto & Lucena (2012, 1-2) tiivistävät, että ketterän testaamisen onnistuminen vaatii muun muassa ison kokonaisuuden hahmottamista, yhteistyötä asiakkaiden kanssa, jatkuvan palautteen saamista, automaattista regressiotestausta ja koko tiimin yhteistyötä.



Kuvio 2: Agile testing quadrants (Collins ym. 2012, 2)

Ketterä testaaminen koostuu neljästä kvadrantista, jotka ohjaavat testaustoimintaa. Kvadrantti voi olla yritykseen (käyttäjiin) tai tekniikkaan (kehittäjiin) suuntautunut, koska jotkut testit auttavat kehittäjiä työssään, kun taas toiset parantavat tuotteen käytettävyyttä ja laatua käyttäjän näkökulmasta. Malli sallii kaikkien testien automatisoinnin, manuaalisen suorittamisen tai sekä että. (ISTQB 2014, 29)

Ensimmäinen kvadrantti on yksikkötaso, joka on tekniikkaan suuntautunut. Yksikkötestien automatisointi ja niiden sisällyttäminen jatkuvan integraation prosessiin on tämän kvadrantin ydinidea. Nämä auttavat kehittäjiä samaan nopeaa palautetta ohjelmiston toimivuudesta kehityksen edetessä. (ISTQB 2014, 29)

Toinen kvadrantti on systeemitaso, joka on yritykseen suuntautunut. Tämän kvadrantin tarkoituksena on varmistaa tuotteen käyttäytyminen. Tämä kvadrantti keskittyy funktionaaliseen testaamiseen, tarinoiden ja käyttötapausten testaamiseen, prototyyppien tekemiseen ja simulaatioihin. Nämä voivat olla joka automatisoituja tai manuaalisia, mutta keskeisenä tarkoituksena on luoda laadukkaita käyttäjätarinoita, joita voi hyödyntää automaattisten regressiotestien kirjoittamisessa. (ISTQB 2014, 29)

Kolmas kvadrantti on systeemi- tai käyttäjähyväksyntätaso, joka on yritykseen suuntautunut. Testien tarkoituksena tässä kvadrantissa on kritisoida tuotetta käyttämällä realistisia käyttäjäkokemuksia ja dataa. Samalla yhdistyy tutkiva testaaminen, skenaariot, käytettävyystestaaminen, hyväksyntätestaaminen ja alpha- ja betatestaaminen. Nämä ovat kaikki hyvin usein manuaalisesti suoritettuja testejä, ja ovat käyttäjäpainotteisia. (ISTQB 2014, 29)

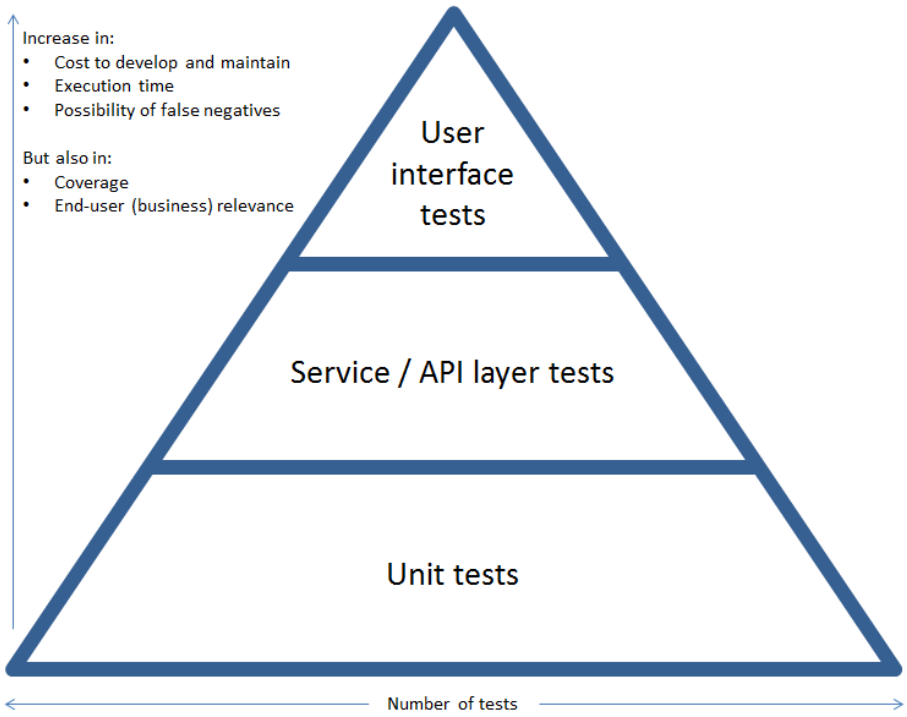
Neljäs kvadrantti on systeemi- tai operatiivinen taso, joka on tekniikkaan suuntautunut. Tähän sisältyy testejä, joiden ainoana tarkoituksena on kritisoida tuotetta ja niitä suoritetaan eri työkalujen avulla. Näitä testejä ovat esimerkiksi suoritus-, kuorma-, stressi- ja skaalautuvuustestejä, joiden tarkoituksena on testata ohjelmiston kestävyyttä äärriolosuhteissa. Näiden lisäksi kvadrantti sisältää muun muassa tietoturva-, huollettavuus-, muistinhallinta-, infrastruktuuri- ja elpymistestejä. (ISTQB 2014, 29)

Collinsin ym. (2012, 2) mukaan perinteisen ohjelmiston testaaminen keskittyy kahteen oikeanpuoliseen kvadranttiin, jolloin testaamisessa on kyse ohjelmiston kritisoimisesta eikä sen kehityksen tukemisesta. Tämän lisäksi testaaminen tapahtuu kehityskaaren loppupäässä eikä jatkuvasti sen aikana, jolloin virheitä kyetään vain havaitsemaan eikä estämään. Collins ym. (2012, 2) selittävät, että ketterässä testaamisessa koko tiimin tulisi osallistua virheiden havaitsemiseen ja estämiseen koko kehityksen ajan, jolloin virheiden syntymistä kyetään estämään ja korjata pois mahdollisimman aikaisessa vaiheessa.

(ISTQB 2014, 29-30) mukaan jokaisen ohjelmiston iteraation aikana testauskvadrantteja voi soveltaa dynaamisesti tarpeen mukaan, eikä niiden tarvitse olla staattinen osa testausprosessia. Mallin tarkoituksena on auttaa varmistamaan, että testaamisessa on käytettävissä laaja valikoima laadukkaita testejä, ja niitä voi käyttää laajasti tarpeen mukaan. Kuten edellä on mainittu, testauksen automatisoinnilla on ketterässä testaamisessa keskeinen rooli, koska se mahdollistaa ketterässä ympäristössä nopean ja jatkuvan palautteen saamisen.

2.3 Testauksen automatisointi

Testauksen automatisoinnilla tarkoitetaan tietokoneohjelman käyttöä, millä hallitaan ja suoritetaan testejä, sekä verrataan ohjelmiston oletettua ja todellista käyttäytymistä. Testaamista voi automatisoida eri tasoilla, kuten käyttöliittymä- ja rajapintatesteissä. Automatisoituja testejä voi esimerkiksi ajoittaa suoriutumaan määrättynä kellonaikana, tai kun ennalta määrätty toiminta on toteutunut. (Rouse, 2018)



Kuvio 3: Testipyramidi (Djikstra, 2014)

Testauksen automatisoinnin voi jakaa testipyramidin mukaisesti kolmeen osaan: käyttöliittymätesteihin, rajapintatesteihin ja yksikkötesteihin. Pyramidin päällimmäisenä ja tämän opinnäytetyön kohteena on käyttöliittymätestit, joiden kehittäminen ja ylläpito vaatii paljon aikaa ja rahaa. Hoffman (2003, 20) huomauttaa, että käyttöliittymätestit ovat hitaita ja virhealttiita antamaan valheellisia virheilmoituksia, koska ne testaavat konkreettisesti käyttäjän ruudulla näkyviä asioita. Esimerkiksi jos käyttöliittymästä poistetaan nappi, mikä kuului testien piiriin, nostaa se käyttöliittymätestissä valheellisen virheilmoituksen. Käyttöliittymän tapahtuva muutos täytyy myös muuttaa testeihin, mikä vaatii kehittäjältä aikaa. Tämän takia Djikstran (2014) mukaan olisi ideaalista, että käyttöliittymätestejä toteutetaan mahdollisimman vähän, eikä niiden keskeisenä tarkoituksena saisi olla ohjelmiston toimivuuden varmistaminen. Toimivuuden varmistamisen tulisi tapahtua ideaalisesti jatkuvasti pyramidin alemmilla tasoilla. Käyttöliittymätestit olisi parasta suorittaa mahdollisimman myöhäisessä vaiheessa julkaisuputkea ja mahdollisimman vähissä määrin, koska ne vaativat paljon aikaa. Tätä ajatusta tukee Hoffmanin (2003, 20) huomautus siitä, että käyttöliittymätestien ylläpito on työlästä ja kallista, joten työmäärän keskittäminen pyramidin alemmille tasoille on järkevämpää.

Testipyramidin keskellä olevat rajapintatestit testaavat ohjelmiston tiedon käsittelyä, eikä ota ollenkaan kantaa käyttöliittymän toimintaan. Esimerkiksi Moveniumin Time Tracker 4 -ohjelmistossa rajapinnasta voi hakea työntekijän työaikoja, jotka puolestaan näkyvät käyttäjille ohjelmistossa. Rajapintatestin tarkoituksena on varmistaa, että palvelin palauttaa oikean

ihmisen työajat oikeassa muodossa, kun taas käyttöliittymätestin tarkoituksena on testata, että näkyykö käyttöliittymässä työaikoja, ottamatta kantaa ovatko tiedot oikein vai väärin. Djikstran (2014) mukaan rajapintatestit ovat huomattavasti nopeampia suorittaa kuin käyttöliittymätestit, ja ne ovat vähemmän virhealttiita.

Tukevan pohjan testipyramidille rakentaa yksikkötestit, jotka ovat Djikstran (2014) mukaan nopeita ohjelmoida, nopeita suorittaa ja niistä saa nopeasti palautetta koodin laadusta. Ne keskittyvät ohjelmiston yksittäisiin pieniin toimintoihin, eivätkä ota huomioon suuria kokonaisuuksia. Tämä mahdollistaa nopean, tehokkaan ja tarkan virheen havaitsemisen. Yksikkötestit voidaan suorittaa nopeasti ja jatkuvasti, ja ajallinen vaatimus niiden kehittämiseen on pieni.

Jokainen osa testipyramidista on kuitenkin tärkeä osa kokonaisuutta, jotta testauksen automatisointi on laadukasta ja tehokasta. ISTQB (2014, 29) mukaan testipyramidin keskeisenä ideana on aikainen laadunhallinta, jolloin virheet havaitaan mahdollisimman ajoissa kehityksen aikana.

2.4 Automatisoinnin hyödyt

Kuten aikaisemmin on mainittu ja ISQTB:n (2018, 15) tukemana, ihmisen toiminnalle on luonteenomaista erehtyä ja tehdä virheitä. Manuaalisen ohjelmiston testauksen laatu saattaa kärsiä esimerkiksi testaajan vireystilasta, stressistä tai motivaatiosta. Testaamisen tavoitteista ja skaalasta saattaa olla kommunikaation huonouden takia väärinymmärryksiä, tai teknologia on testaajalla uutta tai väärin ymmärrettyä. Tietokoneet eivät tunnetusti kärsi väsymyksestä, motivaation puutteesta tai kommunikaatiovirheistä, joten ne soveltuvat erittäin hyvin testaajan rooliin.

Automatisoinnin suurin hyöty näkyy ajansäästönä, koska sen avulla voidaan vähentää toistuvia manuaalisia tehtäviä. Ihmisen ei tarvitse käyttää omaa aikaansa tehtävään, mikä on automatisoitu, vaan hän voi keskittää työvoimaansa muualle. Ajansäästön rinnalla testausautomaatio lisää testaamisen johdonmukaisuutta ja toistettavuutta, koska testit suoritetaan joka kerta samalla lailla ja samalla datalla. (ISTQB 2018, 81; Hoffman 2003, 28)

Kasurinen, Taipale ja Smolander (2009, 1-2) kuitenkin huomauttava, että manuaalista testaamista ei voi automatisoinnin myötä unohtaa, koska tutkivaa testaamista on erittäin hankalaa ja kallista automatisoida. Bach (2003, 2) määrittelee tutkivan testaamisen olevan testaamista, jossa testaajalla on aktiivinen rooli testin suunnittelussa ja toteuttamisessa, ja niitä suorittaessa hän hyödyntää omia havaintojaan ja ideoitaan kehittääkseen uusia ja parempia testejä. Bach (2003, 3) jatkaa ideaa toteamalla, että kaikki ihmisen suorittama testaaminen on joltain osin tutkivaa testaamista, koska kyseessä on aktiivista informaation käsittelyä ja ideoiden synnyttämistä. Tutkivaa testaamista on hyvä hyödyntää tilanteissa, missä ei ole

itsestään selvää, mikä seuraavan testin kohde pitäisi olla, koska tutkiva testaaja osaa löytää uusia testauksen kohteita.

Kasurinen ym. (2009, 1-2) toteavat, että ohjelmistokehityksessä on kaksi tavoitetta: vähentää kuluja ja parantaa tuotteen laatua. Nämä kaksi asiaa ovat ristiriidassa keskenään, koska laadun parantaminen kasvattaa kuluja. Testauksen automatisointi voi kuitenkin mahdollistaa näiden kahden tavoitteen yhdistämisen, ja tuottaa yhdistelmän, joka sekä vähentää kuluja että parantaa tuotteen laatua.

Ajan ja rahan säästämisen lisäksi objektiivisten tulosten raportoiminen ja esittäminen helpottuvat automatisoinnin myötä, koska testitulokset ja tieto voidaan esittää erilaisin analyysikeinoin. Testien etenemisestä, virheiden määristä, tahdista ja asteista sekä ohjelmiston suorituskyvystä saadaan luotettavaa ja varmaa dataa, mikä ei ole ihmisen subjektiivinen näkemys tai ymmärrys asiasta. (ISTQB 2018, 81)

2.5 Automatisoinnin haasteet ja riskit

Hoffman (2003, 18) pureutuu syvälle erityisesti käyttöliittymän automatisointiin liittyvistä ongelmista ja riskeistä. Yritykselle suurin ongelma ja riski on se, että testien ohjelmointi on erittäin hidasta ja vaivalloista eli se maksaa erittäin paljon rahaa. Testien automatisointi on muutenkin valtava rahallinen investointi, mutta varsinkin käyttöliittymän testausautomaatiossa tämä korostuu. Hoffmanin (2003, 18) mukaan tämä johtuu siitä, että yhden testin kirjoittaminen saattaa kestää jopa kymmenen kertaa pidempään kuin manuaalisen testaamisen kartoittaminen. Ne saattavat myös vaatia osaavien kehittäjien työaikaa, mikä vie huomion pois ohjelmiston kehittämisestä.

Testausautomaation ongelmana on myös väärät virheilmoitukset, mitkä ovat yritykselle kalliita. Jokainen virheilmoitus täytyy tutkia ja mahdolliset viat työkalussa tai testissä täytyy korjata (Hoffman 2003, 20). Laadukkaan ja toimivan automaatiotestin kirjoittaminen on haastavaa, ja ne ovat pitkän aikajakson tuotoksia. Tätä näkemystä tukee myös ISTQB (2018, 81), joka kertoo, että odotukset automaatiotyökalusta saattavat olla tiimissä liioiteltuja, jolloin ajan, rahan ja vaivan määrä myös aliarvioidaan. Testien ylläpitämiseen ja virheilmoitusten korjaamiseen käytetty aikaa ja vaivaa on myös helppo aliarvioida, ja muutenkin hyöty automatisoinnista saattaa olla pienempi kuin aluksi oltiin arvioitu.

Hass (2008, 94) tuo esille myös haasteet automaatiotyökalujen valinnassa. Uuden työkalun opettelu ja käyttöönotto vaatii paljon aikaa, eikä sen soveltuvuutta omiin tarpeisiin voi täysin taata. Työkalut eivät myöskään välttämättä kykene keskustelemaan keskenään, jolloin tiedon jakaminen ja hyödyntäminen voi olla erittäin kallista ja vaivalloista, tai jopa mahdotonta. Työkalut saattavat olla myös erittäin kalliita, ja uuteen työkaluun siirtyminen jättää vanhaan työkaluun investoidun ajan ja rahan taakseen. Tämän takia Hass (2008, 94) korostaa, että

testausautomaatiotyökalun valinnassa yrityksen tulisi määrittää tarkan strategian siitä, millä työkaluilla testausautomaatio toteutetaan, ja miten niistä syntyviä kustannuksia hallitaan ja käsitellään.

2.6 Regressiotestaus

Tamresin (2002, 223) ja ISTQB:n (2018, 39) mukaan regressiotestauksen tarkoituksena on varmistaa, että palvelun eniten käytetyt toiminnot toimivat uusien koodimuutosten jälkeen. Tämän tärkeys kasvaa sen myötä, kun ohjelmiston koko ja monimutkaisuus kasvavat (ISTQB 2018, 29). Monimutkaisen ohjelmistoon tehdyssä koodimuutoksella on olemassa ISTQB:n (2018, 41) mukaan riski, että yhdellä koodimuutoksella saattaa olla ennalta-arvaamattomia vaikutuksia ohjelmiston toiminnallisuuteen muualla. Tällaisia odottamattomia sivuvaikutuksia tai vikoja kutsutaan regressioiksi, ja niiden testaamisesta syntyy käsite regressiotestaus.

Hoffman (2003, 45) kuitenkin huomauttaa, että rergressiotestaukseen liittyy haasteita. Samoja toiminnallisuuksia testataan useilla eri testeillä, ja useimmiten ohjelmiston virheet havaitaan ja korjataan jo samalla kun regressiotestit kirjoitetaan. Mahdolliset virheet ja bugit voidaan myös havaita yhtä tehokkaasti muilla testausmenetelmillä, eikä automatisoitu regressiotesti osaa ajatella siihen ohjelmoidun logiikan ulkopuolella. Regressiotesti vaatii valtavasti vaivaa ja ohjelmointitaitoa, jotta sen tehokkuus voidaan varmistaa.

Regressiotestit ovat myös siitä ongelmallisia, että ne testaavat vain ohjelmiston vanhoja ominaisuuksia. Jos ohjelmiston lisätään uusi ominaisuus, sille täytyy kirjoittaa täysin uusi testi. Jos puolestaan vanhaa ominaisuutta muutetaan, sille kirjoitettu regressiotesti on muuttunut heikoksi tai jopa hyödyttömäksi. Tämän vuoksi regressiotestauksen ylläpidolla on kovat kustannukset, jotka yritys voittaa takaisin vasta seuraavassa julkaisussa, ei nykyisessä. (Hoffman 2003, 19)

ISTQB:n (2018, 29-39) mukaan ohjelmiston koon kasvaessa regressiotestauksen tärkeys kasvaa ja korostuu. Niillä on tärkeä rooli testaamisessa, koska ne luovat luottamusta ja varmuutta siitä, että ohjelmiston muutos ei ole rikkonut olemassa olevia komponentteja tai toiminnallisuuksia. Vaikka niiden kehittäminen on vaivalloista ja jatkuvaa, ne ovat tehokkaita löytämään ohjelmiston muutoksia ohjelmiston käyttäytymisessä ja vahvistamaan, että korjattu vika on oikeasti korjattu.

Moveniumin testaaminen tapahtuu manuaalisesti yhden tai kahden testaajan toimesta seuraamalla dokumentoituja askelia ja oman harkinnan mukaan. Manuaaliset testit tehdään tikettikohtaisesti sprintin aikana, ja kokonainen ohjelmiston testaus tehdään ennen uuden ohjelmistopäivityksen julkaisua. Tamresin (2002, 223) mukaan ihanteellinen tilanne on sellainen, että regressiotestit suoritetaan jokaisella kerralla ja samalla tavalla, kun sovellus muuttuu. Tätä

manuaalinen testaaminen ei voi taata, koska tyypillisesti ihminen suorittaa testauksen joka kerta hieman eri tavalla.

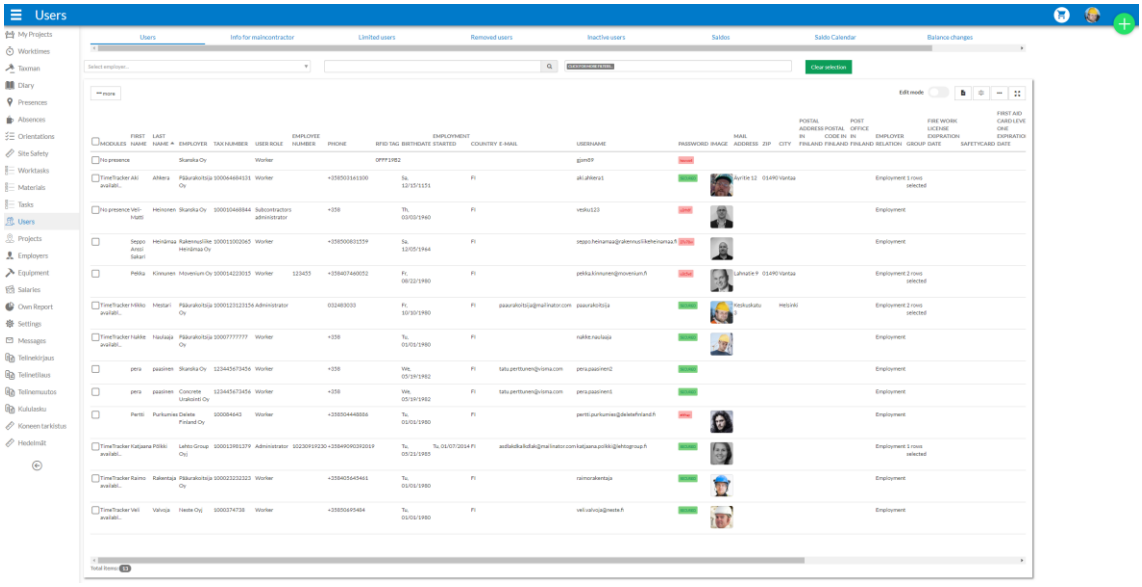
Tämä opinnäytetyö pyrkii automatisoimaan julkaisua ennen tehtyjä manuaalisia regressiotestejä. Tulevaisuudessa käyttöliittymän testien automatisointia voi hyödyntää myös siten, että samat testit ajetaan uudestaan tuotannossa ohjelmistopäivityksen jälkeen varmistaakseen, että päivitykset ovat menneet läpi loppukäyttäjälle asti ongelmitta.

3 Kehitysympäristö

Tässä työssä testauksen kohteena on verkkosivu, joten nettiselaimen lisäksi ainoa pakollinen asennettava ohjelmisto on Katalon Studio, mikä on saatavilla kaikille yleisille käyttöjärjestelmille. Tarpeen mukaan muita internetselaimen voi asentaa testaamista varten, mutta tässä työssä testejä kehitettiin Google Chromea -internetselainta vasten.

3.1 Testauksen kohde

Automaatiotestauksen kohteena on Moveniumin Time Tracker 4 -ohjelmisto, lyhyemmin TT4. TT4 on Single-Page Application (SPA), eli sen toiminta perustuu yhden HTML dokumentin dynaamiseen päivittämiseen. Tietojen käsittely ja esittäminen tapahtuu ohjelmistossa lomakkeilla ja raporteilla. Esimerkiksi uuden käyttäjän luominen palveluun tapahtuu lomakkeen kautta, minkä jälkeen se ilmestyy palveluun käyttäjäraportille.

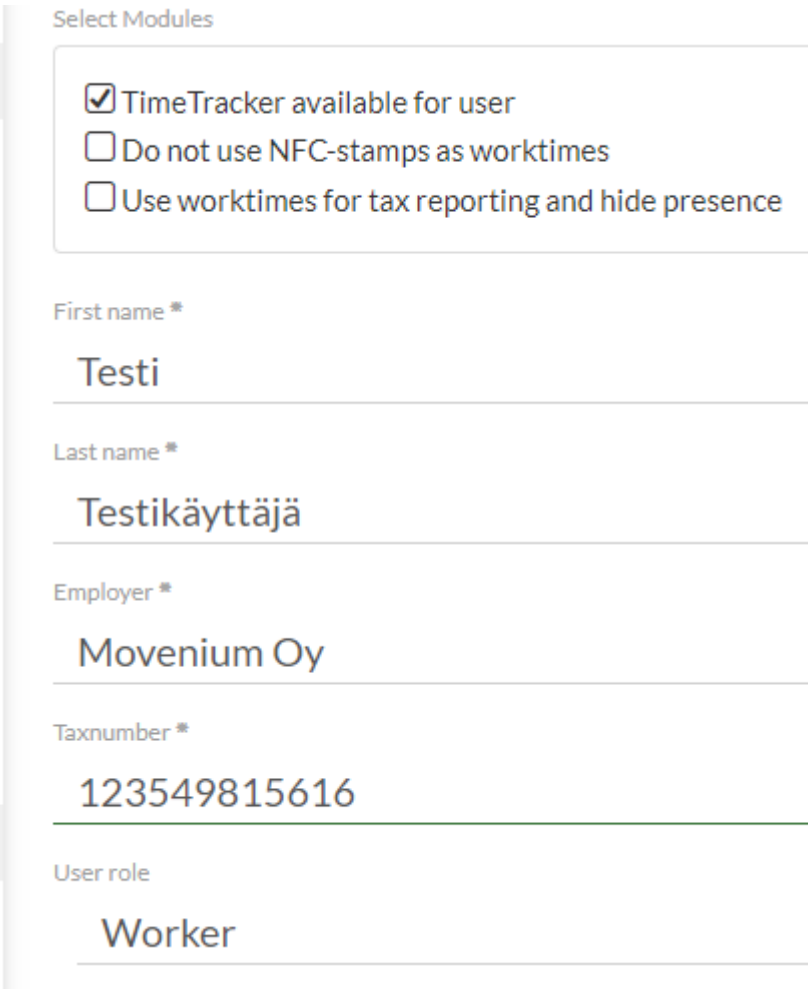


Kuvio 4: TT4 Käyttäjäraportti

Raporteissa näkyy palveluun syötettyjä tietoja. Käyttäjäraportilla näkyy kaikki käyttäjät, jotka voivat kirjautua sisään ja käyttää palvelua. Raporttien tietojen näkyvyyttä tai toiminnallisuutta voi tarpeen mukaan säädellä pääkäyttäjien toimesta, mutta palvelu määrittää

vakiolta näkyvyydet eri käyttäjätasojen mukaan. Pääkäyttäjät saavat vapaasti nähdä kaikki palveluun kuuluvat tiedot, kuten aliurakoitsijoiden työntekijöiden työajat, mutta työntekijät saavat nähdä vain omat työaikansa.

Palveluun syötetään tietoa lomakkeiden kautta, missä kysytään asiaan liittyviä tarvittavia tietoja. Yrityksen tarpeiden ja koon mukaan palvelu saattaa vaatia, että lomakkeeseen täytyy täyttää lain edellyttämät tiedot, joka saattaa johtaa usean kymmenen kohdan lomakkeeseen. Testaamisessa tämä on vaivalloista, koska testaajien täytyy ottaa huomioon kaikki palvelunkäyttöön liittyvät ääritapaukset, joten lomakkeet ovat useimmiten pitkiä ja vaativat paljon täyttämistä.



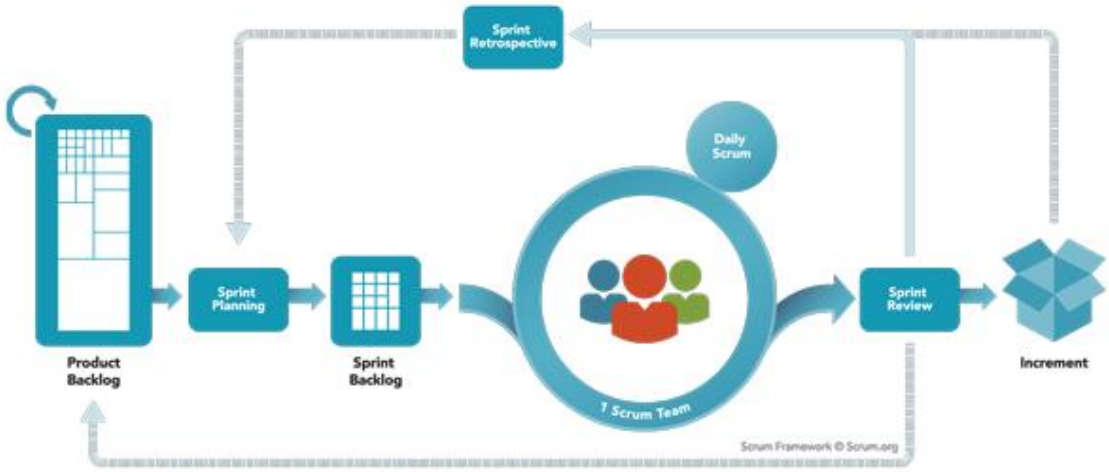
Kuvio 5: TT4 Käyttäjän lisäyslomake

Julkaisua ennen testaajan täytyy testata palvelu kaikilla ominaisuuksilla ja asetuksilla, sekä syöttää tietoja erilaisille lomakkeille eri tarkoituksiin. Hänen täytyy muun muassa luoda käyttäjiä jokaiselle eri roolille, luoda työmaat erilaisilla veroilmoitusvelvollisuuksilla ja asettaa jokaiselle työmaalle työtehtäviä, materiaaleja ja muita työmaan hallintaan liittyviä tietoja. Tämä tarkoittaa käytännössä sitä, että lomakkeita täytyy täytellä toista kymmentä joka kerta

kun ohjelmistoa testataan ennen julkaisua. Kuten ISTQB (2018, 81) huomauttaa, toistuva ja yksitoikkoinen työ on erinomainen automatisoinnin kohde, joten tästä löytyi hyödyllinen automatisoinnin kohde.

3.2 Kehitysmenetelmä

Movenium hyödyntää kehitysmenetelmänään Scrumia, missä on kahden viikon sprintit. Scrum soveltuu erinomaisesti kehitysympäristöön, jossa tilanteet ja kehityssuunta on nopeasti muuttuva tai vaikeasti ennustettavissa (Schwaber & Sutherland 2017, 4). Kasurisen (2013) mukaan Scrum on ketteränä menetelmänä suhteellisen yksinkertainen ja helposti hallittavissa, ja se toimii parhaiten tiimeissä, missä ryhmän jäsenet pystyvät vapaasti kommunikoida ja jakaa tietoa keskenään vapaasti.



Kuvio 6: Scrum kehitysmenetelmä (Scrum.org, 2018)

Scrum-tiimissä on tuotteen omistaja, Scrum Master sekä kehitystiimi. Kehitysprosessi lähtee liikkeellä Product Backlogista, missä on lista työtehtävistä. Sprintin suunnittelupalaverissa Scrum Master jakaa tiimille backlogissa olevia työtehtäviä, jotka tulisi tehdä sprintin aikana. Samalla keskustellaan tehtävän toteuttamisesta, sekä keinoista ja menetelmistä, millä tehtävän toteuttamisesta tehdään mahdollisimman tehokasta. (Schwaber & Sutherland 2017, 4)

Kun sprintti on suunniteltu, alkaa varsinainen toiminta ja toteuttaminen eli sprintti. Schwaberin & Sutherlandin (2017, 9) mukaan Optimaalisesti sprintti kestää kahdesta neljään viikkoa, ja sen aikana tiimin tulisi luoda määritelmät täyttävän, käyttökelpoisen ja julkaisukelpoisen ohjelmistoversion. Kasurisen (2013, 26) mukaan sprintin tavoitteena on tuottaa nippu uusia ominaisuuksia, testata ne ja toimittaa ne eteenpäin julkaistavaksi. Liian pitkässä

sprintissä kompleksisuus kasvaa, määritelmät muuttuvat, ja sprintin tavoitteita on vaikeampi toteuttaa.

Sprintissä päivä alkaa päiväpalaverilla, minkä optimaalinen kesto on enintään 15 minuuttia. Tämän aikana kehitystiimin jäsenet kertovat tai esittävät edellisen päivän tuloksensa, jotta muut tiimin jäsenet pysyvät hyvin perillä tiimin muista tapahtumista. Samalla kerrotaan, onko työn etenemiselle esteitä, tai mitä seuraavaksi olisi tarkoitus tehdä. Päiväpalaverissa ei ole tarkoitus raportoida, vaan jakaa ideoita ja näkemyksiä siitä, miten tiimi pääsee tehokkaasti sprintin lopputavoitteeseen. (Schwaber & Sutherland 2017, 10-12)

Sprintin päätteeksi tiimi pitää sprinttikatselmuksen, missä arvioidaan sprintin toteutusta ja saavutettua lopputulosta. Jokaisen palaverin aikana Scrum Master etsii kehityskohteita ja keinoja, millä seuraavan sprintin tehokkuutta ja laatua voidaan parantaa. Katselmus antaa tiimille keinon ja väylän jakaa kehitysideoita, sekä ideoita miten tulevaisuudessa voisi toimia paremmin. (Schwaber & Sutherland 2017, 13)

Moveniumin asiakkaina ovat pääsääntöisesti rakennusalan yritykset, joiden työmaat saattavat vaihdella viiden henkilön rakennuttamasta pienestä mökistä isoon, ydinkeskustaan rakennettavasta pilvenpiirtäjästä. Tämän lisäksi jokaisella yrityksellä on omat tarpeet ja tavat toimia, joten yhdistelmänä syntyy ennalta arvaamaton ja nopeita muutoksia vaativa yhtälö. Scrumin iteratiivinen malli mahdollistaa nopean reagoimisen asiakkaiden tarpeisiin, ja mahdollistaa tehtäväkohtaisen prioriteetin hallinnan.

Testausautomaation kehittäminen soveltuu erittäin hyvin Movenium ketterään kehitysmalliin mukaan, koska se mahdollistaa jatkuvan ja nopean palautteen testien etenemisestä. Testejä kehitetään yksi tapaus kerrallaan, joista ajan myötä syntyy yhdistettynä iso kokonaisuus. Esimerkiksi yhden sprintin tavoitteena voi olla kirjoitta testi tapaukselle, missä käyttäjä aloittaa ja sulkee työajan. Tämä on irrallinen operaatio muusta tuotteen toiminnasta, ja tästä yksittäisestä testistä on sitten helppo antaa palautetta.

4 Katalon

Yrityksenä Katalon tarjoaa käyttäjilleen kolme tuotetta: Katalon Studio, Katalon Recorder ja Katalon Analytics. Katalon organisaation päätuotteena on Katalon Studio, jolla kirjoitetaan ja ajetaan testit. Muut Katalon organisaation tuotteet ovat Katalon Studion moduuleja tai lisäosia, joilla voi saada lisää tai vaihtoehtoista toimivuutta Katalon Studioon. Kaikki Katalonin tuotteet ovat maksuttomia.

Katalon Studio on täysimittainen IDE, joten mukana tulee täysimittainen käyttöliittymä ja ohjelmoinnin tekstieditori. Testien käyttäytymistä, kuten viivettä, voi määritellä asetuksista, sekä integraatioiden ja muiden lisäominaisuuksien säätäminen on helppoa käyttöliittymän avulla.

Katalon Studio on rakennettu Seleniumin päälle ja sisältää valmiiksi kaiken tarvittavan, kuten Javan, Android SDK:n, Web driverit ja kaikki tarvittavat kirjastot, joten nettisivun testaamiseen minkään muun ohjelmiston asentaminen ei ole pakollista. Kännykkäsovelluksen testaamiseen on asennettava Appium erikseen, koska se päivittyy nopeasti ja jatkuvasti.

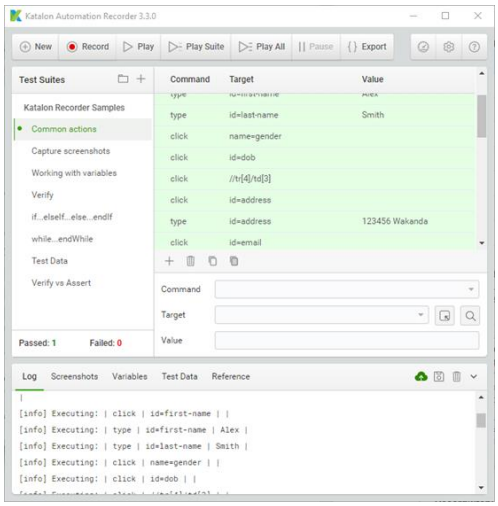
Testaamiseen Katalon tarjoaa täyden paketin, jolla on mahdollista kattaa koko ohjelmiston testaamisen. Se sisältää mahdollisuudet rajapintatestaamisen, web -testaamiseen sekä mobiilitestaamiseen ilman erillistä asentamista tai konfiguroimista. Katalon Studioon on myös mahdollista migratoida tai luoda oma tietokanta, joten sillä saa rakennettua täysin oman ja irrallisen testiympäristön.

Katalonin testit ovat avainsanapohjaisia (engl. Keyword-driven testing, KDT), mikä tarkoittaa, että testaaminen perustuu avainsanoihin, jotka määrittävät mahdollisimman yksinkertaisesti ja laajasti tapahtuvan yksittäisen toiminnon. Tämä tekee testeistä helposti luettavia ja ymmärrettäviä, joten vähemmänkin kokeneet ohjelmoivat pystyvät helposti luomaan testejä.

Moveniumin tarpeisiin Katalon Studio on erinomainen valinta, koska automaatiotyökalun pitäisi ensisijaisesti olla helposti ja nopeasti opeteltavissa, jotta kuka tahansa tiimin jäsen voisi tarpeen mukaan kehittää testejä. Täysimittainen ja valmis kehitysympäristö sekä täyden mittakaavan paketti täyttävät kaikki tiimin tarpeen. Tämän lisäksi on hyvä, että samalla työkalulla voisi tulevaisuudessa testata myös mobiiliaplikaatioita, joita Moveniumilla on kaksi kappaletta sekä Androidilla että Applen iOS:llä.

4.1 Katalon Studion lisäosat

Katalon Studion rinnalle voi ottaa käyttöön kaksi muuta Katalonin kehittämää tuotetta, Katalon Recorderin ja Katalon Analyticsin. Katalon Recorder mahdollistaa selaimen nauhoittamisen, eli käyttäjä voi laittaa nauhoituksen päälle ja käyttää selainta kuten normaalisti. Kun nauhoitus lopetetaan, Katalon Recorder luo automaattisesti testeille objektit, ja kirjoittaa parhaan yrityksensä mukaan testitkin valmiiksi. Tämän myötä testien kirjoittaminen on nopeaa ja helppoa, ja voidaan helposti varmistaa luonnollisen käyttäjäkokemuksen replikoiminen.



Kuvio 7: Katalon Recorder

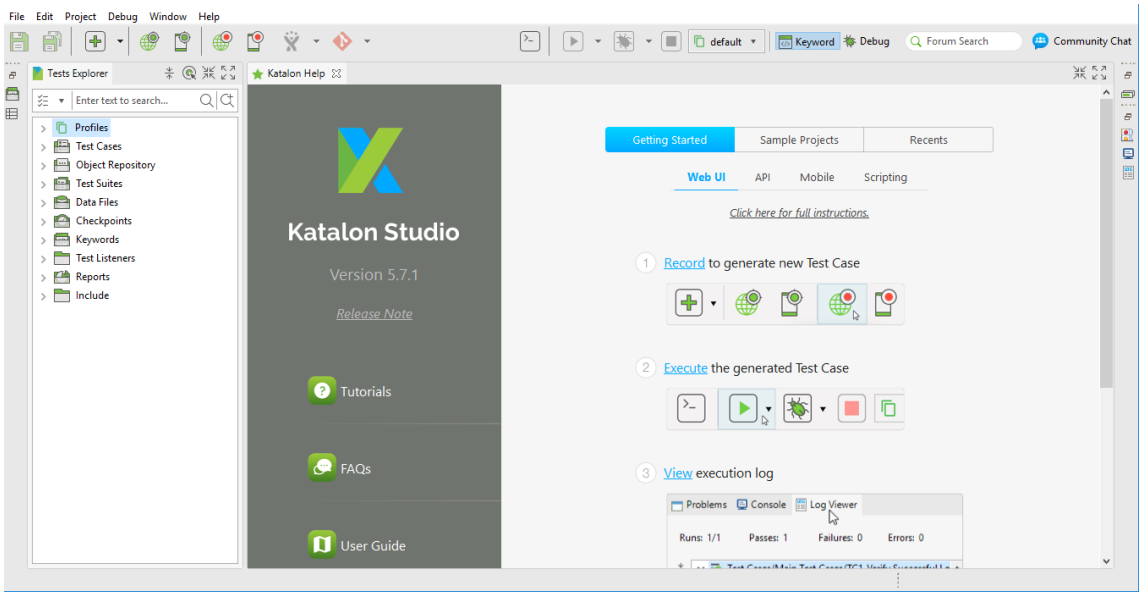
Katalon Analytics on palvelu, minne Katalon automaattisesti lähettää testiraportit ja analysoi ne. Testien yksityiskohtaiset askeleet ja ongelmat tallentuvat palveluun, ja testin epäonnistumisen sattuessa Katalon lähettää myös liitteeksi kuvakaappauksen selaimesta. Tämä helpottaa ja nopeuttaa huomattavasti mahdollisten bugien löytämistä ja korjaamista.

Kuvio 8: Katalon Analytics

4.2 Asennus ja käyttöönotto

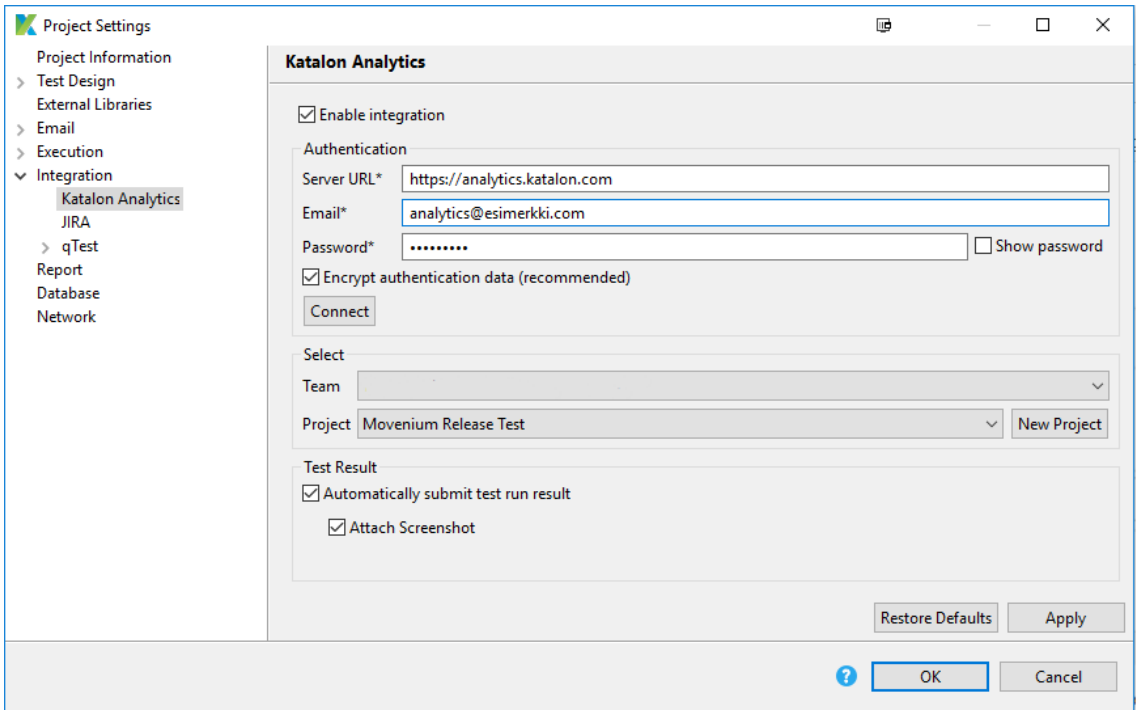
Katalon Studion käyttöönotto koostuu kolmesta yksinkertaisesta askeleesta, joista vain ensimmäinen on pakollinen. Ainoa pakollinen osuus on itse Katalon Studion lataaminen, mikä onnistuu helposti Katalonin nettisivujen kautta. Ainoa vaatimus on ilmaisen tilin luominen, minkä jälkeen sivusto ehdottaa käyttäjälle oikeaa ohjelmistoversiota käyttöjärjestelmästä riippuen.

Kun ohjelman avaa ensimmäisen kerran, se pyytää jälleen Katalon tilisi käyttäjätunnusta ja salasanaa. Näiden täyttämisen jälkeen käyttäjälle avautuu Katalon Studion aloitusnäyttö ja ohjelma on käyttövalmis.



Kuvio 9: Katalon Studion aloitusnäyttö

Katalon Analyticsin integroiminen Katalon Studioon on myös suoraviivaista. Projektin asetusten takaa löytyy integraatioasetukset, mihin syötetään palvelimen osoite, sekä oman Katalon Tilin tiedot. Tämän jälkeen valitaan, mihin Katalon Analyticsin tiimiin ja projektiin testitulokset lähetetään, lähetetäänkö tulokset aina automaattisesti, sekä lähetetäänkö testeistä kuvakaappauksia vai ei.



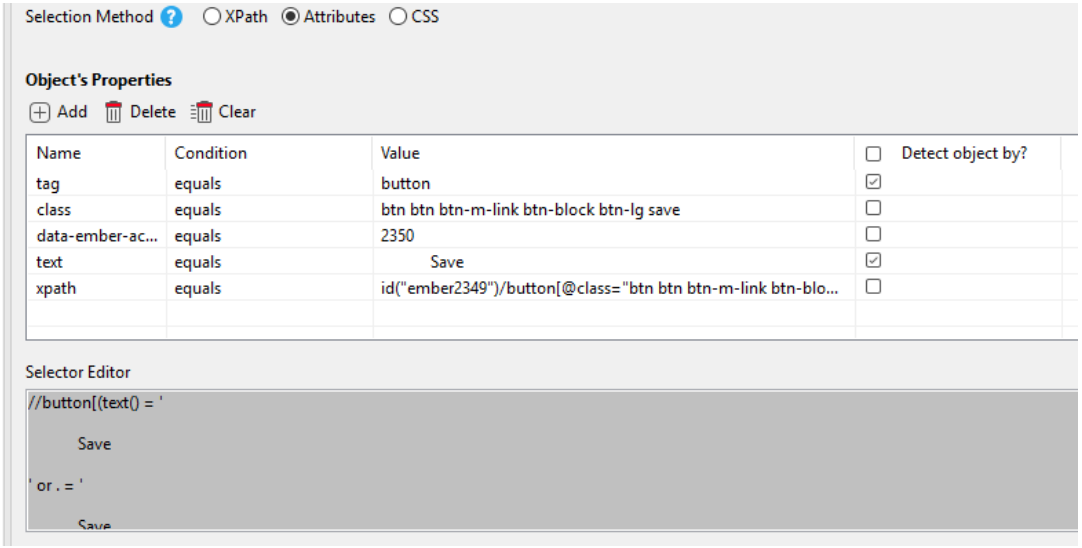
Kuvio 10: Katalon Analyticsin integraatio Katalon Studioon

4.3 Automaatioprojektin rakenne

Katalon Studion projektit koostuvat useasta eri osasta: profiileista, objekteista, testitapauksista testikokoelmista sekä testikuuntelijoista.

Profiileja voi käyttää soveltuvasti omien tarpeiden mukaan, mutta tärkein ominaisuus on, että profiiliin määritetty muuttuja toimii Katalon Studiossa globaalina muuttujana. Tämä tarkoittaa sitä, että sen arvo on käytettävissä jokaisessa testissä kutsumalla sen muuttujan nimeä, eikä sitä tarvitse erikseen määrittää muualle. Yleisin muuttuja, mikä profiiliin määritetään, on testattavan nettisivun osoite. Tämä tarkoittaa käytännössä sitä, että profiilia vaihtamalla voi kohdistaa testien ajon tiettyyn ympäristöön ja antaa sille liittyviä tietoja, kuten sisäänkirjautumiseen tarvittavia tunnuksia.

Objektit ovat tapa, millä Katalon Studio osaa kohdistaa toiminnon nettisivulla olevaan elementtiin. Esimerkiksi tallennusnapin painamiseen Katalon Studio tarvitsee ensiksi tiedon siitä, mikä tallennusnappi ylipäänsä on. Tämä onnistuu luomalla objektin, ja määrittelemällä sille arvot ja attribuutit, joita Katalon Studio osaa sivulta haravoida. Katalon Studio osaa etsiä objektia esimerkiksi HTML-tunnisteen, CSS-luokkien tai elementin tekstin perusteella, ja etsii testin aikana sivulta parhaiten sopivan HTML-elementin. Esimerkiksi tallennusnapin voi etsiä sivulta määrittelemällä objektille, että HTML-tunniste on ”button” ja napin tekstin tulisi olla ”tallenna”.



Kuvio 11: Katalon Studion objekti

4.4 Testien kirjoittaminen

Katalonin testejä voi rakentaa kolmella eri tavalla: tallentamalla internetselaimen käyttöä Katalon Recorderilla, Katalon Studion käyttöliittymän avulla tai itse kirjoittamalla.

Katalon Recorderilla testien kirjoittaminen on hyvin suoraviivaista ja soveltuu ihmisille, joilla on erittäin vähän tai ei ollenkaan ohjelmointitaitoja. Ohjelman avattua voi painaa tallennusnappia, valita tietokoneella auki oleva nettisalain, ja käyttää nettisivua kuten normaalisti. Katalon Recorder tallentaa painallukset taustalla, luo testiobjektit, ja kirjoittaa testit täysin omatoimisesti parhaan kykynsä mukaan (Katalon Docs). Yksinkertaisilla nettisivuilla tämä on erittäin tehokas ja nopea tapa luoda testejä, mutta monimutkaisiin ja dynaamisiin nettisivuihin tallennustoiminto ei riitä. Moveniumin TT4 ohjelmistoon tämä menetelmä ei soveltunut ollenkaan, koska sivun toimivuus ja ulkoasu muuttuu asetusten mukaan, eikä Katalon Recorder osannut kirjoittaa tarpeeksi tarkkoja tai ollenkaan kuvauksia sivun HTML -elementeistä. Tämä johti siihen, että testi ei loppujen lopuksi osannut erottaa yhtä muista, ja testi vaati manuaalista korjaamista ja koodaamista.

Toinen tapa luoda testejä on hyödyntää Katalon Studion käyttöliittymää, missä on alasvetovalikoita ja vaihtoehtoja logiikalle ja testin etenemiselle, eikä ohjelmointitaito ole varsinaisesti tässäkään tilassa tarpeellista. Tämä mahdollistaa sen, että testejä voi luoda ihmiset, joiden ohjelmointikyky on heikko. Tämän manuaalisen näkymän ja skriptitilan välillä voi vapaasti vaihdella, ja molempien käyttö on tehokkaiden ja kattavien testien kirjoittamisen kannalta hyödyllistä.

Testejä voi myös kirjoittaa täysin perinteisesti käsin, niin sanotussa skriptitilassa. Tässä näkymässä on täysi IDE toiminnallisuus, joten kirjoittaminen on kuten normaalisti koodaisi. Manuaalitilassa lisätty avainsana ilmestyy automaattisesti koodiksi skriptitilaan, ja skriptitilassa oikein kirjoitettu koodi ilmestyy manuaalitilaan avainsanaksi. Skriptitilan vahvuudet tulevat nopeasti eteen, kun testiin täytyy kirjoittaa logiikkaa, kuten ehtolauseita.

Ohjelmointikielenä Katalon käyttää Java/Groovyä, joka on hyvin lähellä Javaa, mutta tuo mukanaan ominaisuuksia, joita on käytössä esimerkiksi Pythonissa, Rubyssä ja Perlissä. Sitä voi käyttää sekä ohjelmointi- että skriptauskielenä, ja se hyödyntää Javasta tuttua virtuaalikonetta (Java Virtual Machine, JVM). (The Apache Software Foundation, 2019)

Web -testaamisessa testiskriptissä käytetään pääsääntöisesti Katalonin WebUI -avainsanoja, joiden tarkoituksena on esittää lyhyesti komennon toiminta ja logiikka. Esimerkiksi selaimen voi avata komennolla *”WebUI.openBrowser(’https://katalon.com’)”*, mikä kertoo Katalon Studiolla, että käytetään openBrowser -nimisessä avainsanassa määriteltyä logiikkaa, ja edetään parametreissä määritetylle nettisivulle. Katalonissa on vakiona laaja valikoima avainsanoja, mutta omien avainsanojen koodaaminen on myös mahdollista.

Avainsanojen lisäksi skripteissä voi hyödyntää vapaasti Java/Groovy ohjelmointikieltä kuten normaalissakin ohjelmoinnissa, joten testeistä voi tehdä hyvinkin kattavia ja tehokkaita.

4.5 Testien suorittaminen

Katalon Studion testejä voi ajaa manuaalisesti suoraan käyttöliittymän kautta, ja niitä voi valinnan mukaan ajaa eri selaimilla, mobiiliemulaattorilla tai etätietokoneella. Yksi selainvaihtoehdoista on Chromen Headless -moodi, mikä mahdollistaa Katalonin käytön CI -palvelimella. Tätä avittaa myös mahdollisuus ajaa Katalonin testejä suoraan komentoriviltä, joten integraatio yrityksen CI -palvelimeen on suoraviivaista.

Katalon Studio 5.4 -versio toi mukanaan suoritusprofiilit, joihin voi määrittää globaaleja muuttujia ja muuta testidataa. Tämä helpottaa huomattavasti testien suorittamista eri tuotantoympäristöjä vastaan, koska eri URL-osoitteita voi testata kertomalla Katalonille, mitä profiilia testin ajossa tulisi käyttää. Tämä mahdollistaa myös testien rinnakkaisajon, eli testejä voi ajaa samanaikaisesti monta eri selainta vastaan. Suurin hyöty rinnakkaisajosta on, että testit voi ajaa samaan aikaan jokaista nettiselainta vastaan ilman testien ajoaikojen kasvamista, olettaen että tietokoneen tehot riittävät.

4.6 Testitulosten tarkastelu ja analyysi

Katalon tarjoaa helpon ja laajan testitulosten analysointiin Katalon Analyticsin avulla. Integraatio Katalon Studioon tapahtuu yksinkertaisella asetuksella, eikä sen käyttöönotto vaadi muuta kuin saman Katalon tunnuksen, millä Katalon Studionkin on ottanut käyttöön. Katalon

Studiosta voi manuaalisesti lähettää testiraportit Katalon Analyticsiin, tai sen voi asettaa lähettämään raportit automaattisesti jokaisen testiajon jälkeen.

Katalon Analytics kasaa yhteen monipuolisen ja helposti ymmärrettäviä raportteja Katalon Studion testituloksista.

5 Lopputulokset

Tämän opinnäytetyön tavoitteena oli kehittää Moveniumin automaatiotestausta kehittämällä käyttöliittymän regressiotestausta, ja vähentää manuaaliseen julkaisutestaamiseen kuluvaa aikaa alle puoleen tuntiin vuoden 2018 loppuun mennessä. Molemmat tavoitteet toteutuivat, ja vuoden 2018 vuoden lopussa Movenium luopui kokonaan manuaalisesta julkaisutestaamisesta.

Moveniumin julkaisutestausdokumentaatio käy läpi käyttöliittymän yleisimmät käyttötapaukset, ja kaikki näiden tapausten testit on automatisoitu. Regressiotestauksen automatisointia ei kyetty täysin toteuttamaan, eli testaaminen ei tapahdu joka kerta, kun ohjelmisto päivittyy. Testit ajavat automaattisesti etätietokoneella kaksi kertaa päivässä, aamuyöllä ja keskipäivällä, ja niitä voi helposti manuaalisesti suorittaa lähettämällä API-kutsun.

Katalon Studion käyttöönotto on huomattavan helppoa, ja testien kirjoittaminen ei vaadi erityistä ohjelmoinnin osaamista. Yksinkertainen Katalon Analyticsin integraatio tarkoittaa myös sitä, että testituloksista saa jo hyvin varhaisessa vaiheessa dataa analyysiä varten, joten investointi alkaa maksamaan itseään takaisin hyvinkin nopeasti.

5.1 Yritykselle tuotettu hyöty

Pohjolan (2019) mukaan Moveniumille suurin hyöty testausautomaatiosta näkyy ajansäästönä. Vuoden 2018 lopussa yritys päätti siirtyä jatkuvan julkaisun malliin (engl. Continuous Delivery, CD), koska luotto automaatiotestauksen tasoon oli riittävä. Tämän myötä testaajat ovat siirtyneet ohjelmiston uusien toiminnallisuuksien kehitykseen, joten yrityksen kehitysresurssit ovat kasvaneet. Tämän myötä Movenium pystyy toimittamaan asiakkailleen enemmän arvoa ja entistä nopeammin hyvälaatuisen tuotteen.

Vaikka manuaalisesta julkaisutestaamisesta on luovuttu, tikettikohtaiselle manuaaliselle testaamiselle on edelleen tarvetta. Tämä johtuu siitä, että kyseessä on tutkivaa testaamista, jota on erittäin hankalaa ja kallista automatisoida. Pohjolan (2019) mukaan tämä on kuitenkin hyväksyttävää, koska näissä testitapauksissa on erittäin harvoin toistuvia askelia, joten manuaalinen testaaminen on näissä tapauksissa kustannustehokas ratkaisu. Tutkivassa testauksessa testauksen suorittaa ihminen, joka kykenee ajattelemaan asioita eri näkökulmista testaamisen aikana ja tekemään niistä johtopäätöksiä.

Automaatiotestit takaavat sen, että jokainen testi tapahtuu joka kerta samalla tavalla. Pohjolan (2019) mukaan ajan myötä manuaalinen testaaja tylsistyy ja laiskistuu, joten laadukasta ja täydellistä testaamista ei voi taata. Automatisoinnin jatkuvan testaamisen ja luotettavan luonteen kautta voidaan vähentää ja ennaltaehkäistä asiakkaille asti päätyviä bugeja tai virheitä ja parantaa tuotteen laatua. Testaus tapahtuu ennen jokaista uutta julkaisutapahtumaa, eikä tuotannossa olevaa ohjelmistoa päivitetä asiakkaalle, jos testit eivät mene läpi.

5.2 Testien jatkokehitys

Testien jatkokehityksessä seuraava askel on integroida Katalon Studio Moveniumin CI -palvelimeen, jotta testien suorittaminen oli sataprosenttisesti automaattista ja testien ajo tapahtuisi jokaisen muutoksen jälkeen. Tällä hetkellä testit ajetaan ajastetusti etätietokoneella kaksi kertaa päivässä, ja tarpeen mukaan manuaalisesti.

Testien kehityksen aikana Moveniumilla oli heikkoa tai ei ollenkaan analyyttistä dataa käyttäjien käyttäytymisestä ohjelmistosta, joten testejä kirjoitettiin käyttöliittymän osioille ja sivuille, joiden uskottiin olevan laajimmassa käytössä. Loppukäyttäjien generoimaa käyttäytymisdataa on alettu keräämään vuoden 2018 lopussa, minkä perusteella käyttöliittymätestejä voi kirjoittaa niille käyttöliittymän osa-alueille, mitä tässä opinnäytetyössä ei katettu.

Lähteet

Painetut

Hass, A. M. J. 2008. Guide to advanced software testing. Norwood, Massachusetts: Artech House.

Kasurinen, J. 2013. Ohjelmistotestauksen käsikirja. Docendo.

Myers, J., Badgett, T. & Sandler, C. 2012. The Art of Software Testing. 3. painos. Hoboken, New Jersey: John Wiley & Sons 2012.

Tamres, L. 2002. Introducing Software Testing. Harlow: Pearson Education Ltd.

Vilkka, H. & Airaksinen, T. 2003. Toiminnallinen opinnäytetyö. Tammi Oppimateriaalit.

Sähköiset

Collins, e., Dias-Neto, A. & Lucena, V. 2012. Strategies for Agile Software Testing Automation: An Industrial Experience. Viitattu 28.02.2019. https://ieeexplore.ieee.org/abstract/document/6341616

Djikstra, B. 2014. The Test Automation Pyramid. Viitattu 14.11.2018. https://www.ontestautomation.com/the-test-automation-pyramid/

Hoffman, D. 2003. A Course on Software Test Automation Design. Viitattu 28.11.2018. http://www.testingeducation.org/course_notes/hoffman_doug/test_automation/auto8.pdf

International Software Testing Qualifications Board (ISTQB). 2018. Certified Tester Foundation Level Syllabus. Viitattu 14.11.2018. https://www.istqb.org/downloads/send/51-ctfl2018/208-ctfl-2018-syllabus.html

Internation Software Testing Qualifications Board (ISTQB). 2014. Agile Tester Extension Syllabus. Haettu 28.02.2019 https://www.istqb.org/downloads/send/5-agile-tester-extension-documents/41-agile-tester-extension-syllabus.html

Kasurinen, J., Taipale, O. & Smolander, K. 2009. Software Test Automation in Practice: Empirical Observations. Haettu 27.01.2019 http://downloads.hindawi.com/archive/2010/620836.pdf

Katalon Docs. Creating Test Cases Using Record. Haettu 06.01.2019. https://docs.katalon.com/katalon-studio/tutorials/create_test_case_using_record_playback.html#recording-your-first-test-with-katalon-studio-record-web-function

Rouse, M. 2018. Test Automation. Viitattu 02.02.2019. https://searchsoftwarequality.techtarget.com/definition/automated-software-testing

The Apache Software Foundation. Apache Groovy. Haettu 07.01.2019. http://groovy-lang.org/index.html

Visma Movenium. 2018. Viitattu 14.10.2018. https://www.visma.fi/movenium/

Schwaber, K. & Sutherland, J. 2017. The Scrum guide. Viitattu 23.10.2018. https://www.scrumguides.org/docs/scrumguide/v2017/2017-Scrum-Guide-US.pdf

Wagner, S, 2013. Software Product Quality Control. Haettu 26.02.2019. ftp://soporte.uson.mx/PUBLICO/02_ING.SISTEMAS.DE.INFORMACION/ING%20SFTWR/Software%20Product%20Quality%20Control.pdf

Julkaisemattomat

Pohjola, M. 2018. Tuotemanagerin haastattelu 01.02.2019. Visma Movenium Oy. Helsinki.

Kuviot

Kuvio 1: Ohjelmisto kulkee ongelman ja ratkaisun välissä (Wagner 2013, 1) 9

Kuvio 2: Agile testing quadrants (Collins ym. 2012, 2) 12

Kuvio 3: Testipyramidi (Djikstra, 2014) 14

Kuvio 4: TT4 Käyttäjäraportti 18

Kuvio 5: TT4 Käyttäjän lisäyslomake 19

Kuvio 6: Scrum kehitysmenetelmä (Scrum.org, 2018) 20

Kuvio 7: Katalon Recorder 23

Kuvio 8: Katalon Analytics 23

Kuvio 9: Katalon Studion aloitusnäyttö 24

Kuvio 10: Katalon Analyticsin integraatio Katalon Studioon 25

Kuvio 11: Katalon Studion objekti 26