TestCafe testiautomaatiotyökalun käyttöönotto

Jonna Partanen

2019 Laurea

Laurea-ammattikorkeakoulu

TestCafe Testiautomaatio työkalun käyt-töönottoTestCafe testiautomaatiotyökalun käyttöönotto

Jonna Partanen
Tietojenkäsittely tradenomi
Opinnäytetyö
Marraskuu 2019, 2019

Laurea-ammattikorkeakoulu **Tiivistelmä**
Tietojenkäsittelyn koulutusohjelmaTietojenkäsittely
tradenomi
Tradenomi (AMK)

Jonna Partanen Jonna Partanen

TestCafe testiautomaatiotyökalun käyttöönottoTestCafe testiautomaatiotyökalun käyttöönotto

Vuosi 20192019 Sivumäärä 41

Tässä opinnäytetyössä käsitellään uuden testiautomaatiotyökalun käyttöönottoa. Käyttöönottoprojektin tavoitteena oli parantaa työn toimeksiantajayrityksen, Edita Publishing Oy:n, testiautomaatioprosessia, yhdenmukaistaa testauskäytäntöjä, sekä nostaa testiautomaation avulla merkittävästi olemassa olevaa testikattavuutta yrityksen eri palveluissa.

Opinnäytetyön teoriaosuudessa avataan työssä esiintyvät keskeiset käsitteet, sekä kerrotaan käyttöönoton keskiössä olevasta TestCafe-testiautomaatiotyökalusta, automaatiotestauksesta, testiautomaatiotyökaluista ja järjestelmän käyttöönotosta. Työn tietoperustana on TestCafen dokumentaatio, aihetta käsittelevät artikkelit, sekä testiautomaatiota- ja järjestelmien käyttöönottoa käsittelevä kirjallisuus.

Opinnäytetyön toiminnallisessa osuudessa kuvataan TestCafen käyttöönottoprojekti. Käyttöönottoprojektin etenemää on kuvattu työssä työvaiheittain. Toiminnallisessa osuudessa on käsitelty käyttöönottoprojektiin liittyvää toimintamallia, kuhunkin työvaiheeseen liittynyttä päätöksentekoa ja etenemistä, prosessin aikana käytettyjä menetelmiä sekä käyttöönoton aikana kohdattuja haasteita ja niille löydettyjä ratkaisuja.

Käyttöönottoprojektin tulos oli onnistunut ja sen aikana saatiin käyttöönotettua uusi testiautomaatiotyökalu. Projektin aikana saavutettiin myös muut sille asetetut tavoitteet. Testauskäytännöt ja testiautomaatioprosessi saatiin uudistettua, ja palveluiden testikattavuus parani merkittävästi. Projektin alkuvaiheessa rinnalla toiminut vanha automaatioympäristö saatiin korvattua kokonaisuudessaan TestCafella.

Asiasanat: Testiautomaatiotyökalu, Järjestelmän käyttöönotto, Testiautomaatio, JavaScript

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

Jonna Partanen Jonna Partanen

The Implementation of TestCafe test automation tool

Year 20192019 Pages 41

The purpose of this bachelor's thesis was to implement a new test automation tool. The goal of the implementation project was to improve the test automation process, make testing practices more consistent and increase test coverage of different services at Edita Publishing Ltd.

The theoretical part of the thesis discussed the most important key concepts of the project as well as theory of the TestCafe test automation tool, automation testing, test automation frameworks and implementation of the software. The thesis was based on the TestCafe documentation, articles on the test automation and some research papers on software test automation and implementation of the software.

The functional part of the thesis described the implementation of the TestCafe test automation tool. The progress of the project was described step-by-step. The functional part covered the operational model associated with the implementation project, the decision-making, and progress associated with each work phase, the methods used during the process and the challenges and solutions found during implementation.

The implementation project was successful and the new test automation tool was introduced and other goals were also achieved during the project. The testing practices and test automation process were reformed, and the test coverage of each service improved significantly. The old automation environment that worked side by side in early stages of the project was completely replaced by TestCafe.

Keywords: Test automation framework, system implementation, test automation, JavaScript

Sisällys

1 Johdanto

Testiautomaation käyttäminen testaamisessa on nykyisin erittäin tärkeää yrityksen kilpailukyvyn kannalta. Testiautomaation avulla pystytään varmistamaan palveluiden laatu ja toimivuus huomattavasti tehokkaammin kuin manuaalisesti. Tehokkuus lisää tuottavuutta, tuo kustannussäästöjä, sekä nopeuttaa ohjelmistokehitysprosessia. Testiautomaatiotyökalut ovat ohjelmistoja/viitekehyksiä, jotka tarjoavat valmiita malleja ja rajapintoja testiautomaation toteuttamiseen. Testiautomaatiotyökalujen tarjoamien valmiiden mallien ja rajapintojen avulla testiautomaatioprojektit saadaan nopeammin käyntiin, ja oikein valittu työkalu tarjoaakin usein hyvän pohjan tehokkaasti tuotettavalle ja hyvin ylläpidettävälle automaatioprojektille.

Edita Publishing Oy on käyttänyt testiautomaatiota lukuisien palveluidensa testaamiseen jo aiemmin muutaman vuoden ajan. Käytetyt menetelmät eivät ole kaikilta osin soveltuneet yrityksen toimintaympäristöön, joka näkyi puutteina automaatiotestien kattavuudessa sen myötä yleisessä laadunvarmistuksessa. Alhainen automaatioaste vaikuttaa osaltaan esimerkiksi siihen, että regressiobugeja ei löydetä ajoissa, joka voi pahimmillaan johtaa niiden päätymiseen tuotantoympäristöön, tai hidastaa kehitysputkea tarpeettoman paljon. Vanha käytössä oleva ympäristö koettiin työlääksi testiautomaatiokehittäjille, ja sen katsottiin tulleen tiensä päähän. Nämä syyt johtivat siihen, että koko automaatioympäristö päätettiin päivittää ja toteuttaa yrityksen tarpeisiin paremmin soveltuvilla menetelmillä ja työkaluilla.

Tässä opinnäytetyössä avataan testiautomaatiotyökalun vaihdoksen syitä sekä kuvataan testiautomaatioympäristön käyttöönottoprojektin kulkua. Työn käytännön osuus kuvaa käyttöönottoprojektin etenemistä alkaen tilanteesta, josta projekti lähti liikkeelle, ja kuvaa vaiheittain prosessinkulkua alkaen ympäristöjen pystytyksestä, vanhojen automaatiotestien hyödyntämisestä, sekä uusien testien implementoinnista. Työssä nostetaan esille myös projektin aikana esiintyneitä haasteita ja ongelmia. Opinnäytetyössä kuvattu käyttöönottoprojekti toteutettiin talven 2018 ja kevään 2019 välisenä aikana.

Työn teoriaosuus käsittelee testiautomaatiota, testiautomaatiotyökaluja sekä järjestelmän käyttöönottoa yleisellä tasolla. Teoriaosuudessa sisältää myös peruskuvauksen käyttöönotettavasta testiautomaatiotyökalusta. Tutkimus- ja kehitysmenetelminä projektissa käytettiin toimintatukimusta, hiljaisen tiedon tutkimusta ja osallistuvaa tutkimusta yhtenä tiedonkeruumenetelmänä.

Opinnäytetyön kirjoittajalla ei ollut ennestään tietoja ja taitoja testiautomaatiotyökaluista tai testiautomaation toteutuksesta. Itsensä kehittämisen tarve ja kiinnostus testiautomaatiota kohtaan olikin merkittävä vaikutin tämän kuvatun projektin valikoitumisena tekijän opinnäytetyön aiheeksi. Kirjoittaja ei aiemmin ole ollut mukana muissakaan käyttöönottoprojekteissa, joten projekti oli senkin takia erittäin opettavainen ja mielenkiintoinen.

2 Työn lähtökohdat ja toimeksiantaja

Opinnäytetyön toimeksiantajana toimi Edita Publishing Oy. Edita Publishing kuuluu Nordic Morning Groupiin yhdessä Edita Priman ja Nordic Morningin kanssa. Nordic Morning Group on pohjoismainen yritys, joka toimii myös Ruotsissa. Editan Publishingin roolina konsernissa on julkaista älykkäitä oppimismateriaaleja ja oppimisratkaisuja, sekä tarjota kuratoituja asiantuntijasisältöpalveluita, joihin lukeutuvat esimerkiksi lakitietopalvelut Finlex, Rajalex ja Edilex.

Edita Publishing toimi opinnäytetyöntekijän työharjoittelupaikkana. Työharjoittelun aikana tekijän toimenkuvaan kuului työn aiheena olevan projektin lisäksi web-kehitys ja testiautomaation kehitys lakitiedon ICT-tiimin jäsenenä. Työn lähtökohtana oli uuden, jo ennen kirjoittajan työsuhteen alkamista, valitun testiautomaatiotyökalun käyttöönotto, sekä käyttöönottoon sisältyvä automaatioasteen vaiheittainen nostaminen valitulla TestCafe testiautomaatiotyökalulla.

2.1 Tarve, tausta ja vaatimukset

Edita Publishingilla on lukuisia erilaisia lakitiedon palveluita. Palveluita kehitetään ja päivitetään jatkuvasti sekä toiminnallisuuksiensa, että sisältönsä osalta. Usein päivittyvien julkaisuiden ja kehityksen yhteydessä on tarvetta myös jatkuvalle laadunvarmistukselle, jotta voidaan todentaa palveluiden ja toiminnallisuuksien toimivuus ennen niiden päätymistä kuluttajille. Testaus ja testiautomaatio ovat keskeisessä asemassa nopeasyklisesti etenevässä kehitystyössä. Testauksen tärkeys on kasvanut erittäin paljon viime vuosien aikana, kun ketterät menetelmät ovat yleistyneet ja kun vanhoja, yli 10 vuotta vanhoja palveluita on lähdetty uusimaan ja kehittämään näitä menetelmiä käyttäen. Nopeampi kehitysputki vaatii onnistuakseen myös nopeamman laadunvarmistuksen, josta syntyi tarve luoda testiautomaatioprosessi, jolla Editan kehittämät palvelut voidaan testata kattavasti tuotekehitysputken vaatimalla nopeudella, ja joka on valmis käytettäväksi myös kaikissa uusissa projekteissa.

Editan ICT-tiimissä on aiemmin tehty testiautomaatiota Robot Framework sovelluskehyksellä, joka on ollut käytössä vuodesta 2015 lähtien. Tiimissä työskentelee pääasiassa JavaScript- ja PHP-kehittäjiä, jotka kokivat Robot Frameworkin hieman hankalaksi ja hitaaksi. Robot Frameworkin opettelun koettiin vievän liian paljon aikaa ja resursseja muulta kehitykseltä. Suurin tarve käytettävän testiautomaatiotyökalun vaihdokseen lähtikin siitä, että testiautomaatiokehitys oli muuttunut hitaaksi ja siihen haluttiin resursoida enemmän työntekijöitä. Manuaalitestaus kehitystyön lomassa vie liikaa aikaa, eikä olemassa ollut testiautomaatiokattavuus ollut riittävällä tasolla. Testiautomaatiokehityksen haluttiin kuuluvan kaikkien ohjelmistokehittäjien työhön, eikä ainoastaan testaajien, koska näiden tuli huolehtia myös testisuunnitelmien ja -tapausten suunnittelusta, sekä aikaa vievästä manuaalitestauksesta. Robot Frameworkin aikana testaustiimissä oli työskennellyt vain kaksi henkilöä.

Robot Frameworkin käyttö vaatii perehtymistä erilaisiin kirjastoihin ja rajapintoihin. Esimerkiksi websovellusten testaaminen vaatii Robot Frameworkin ja testattavan ohjelman väliin selainajurin, jonka kanssa työskentely koettiin työlääksi. Lisäksi em. kirjastojen ja rajapintojen asentelu ja käyttöönotto vaatii hieman enemmän teknistä opettelua kuin koettiin mielekkäänä, ja sikälikin RF-ympäristön pystytys koettiin turhan monimutkaisena ja hitaana.

Robot Frameworkin heikkoutena oli mainitun vaikeuden ja hitauden lisäksi myös puutteellinen selaintuki. Erityisesti Internet Explorer ja Edge, tai pikemminkin niitä ohjaavat selainajurit, aiheuttivat haasteita. Usein oli tilanteita, jossa IE:n ja Edgen ajurit eivät tukeneet joitain toimintoja ollenkaan, tai vaihtoehtoisesti olivat suorituskyvyltään niin heikkoja ettei niiden automatisointi onnistunut järkevästi. Robot Framework ympäristössä jokainen selain vaatii toimiakseen erillisen selainajurin (webdriver), ja nämä saattoivat poiketa ominaisuuksiltaan ja suorituskyvyltään melko paljon toisistaan. Tästä ongelmasta haluttiin eroon, koska Internet Explorer ja Edge ovat joissain palveluissa todella käytettyjä selaimia ja niiden testikattavuus haluttiin samalle tasolle kuin muidenkin selainten osalta.

Robot Framework haluttiin korvata työkalulla, joka olisi nopeasti omaksuttava ja käyttöönotettava sekä syntaksiltaan tutumpi Javascript/PHP-kehittäjille. Kehittäjien tulisi uudella työkalulla pystyä tuottamaan sekä ajamaan testejä entistä helpommin, ja lisäksi testien ajoympäristöjen haluttiin onnistuvan minimaalisilla riippuvaisuuksilla ja asennuksilla esimerkiksi Dockerin avulla.

Tiimissä on paljon JavaScript osaamista, joten sen käyttöön haluttiin valita mieluiten JavaScriptiä tai TypeSriptiä totteleva työkalu. Koska myös yrityksen kaikki palvelut ovat selainpohjaisia, niin mahdollisia teknisiä rajoitteita tai riskejä ei JavaScript-pohjaisiin työkaluihin nähty ainakaan suuremmissa määrin liittyvän. Tutun ohjelmointikielen ajateltiin tekevän uuden työkalun omaksumisesta nopeaa, joka puolestaan vauhdittaisi merkittävästi automaatiotestien tekoa ja siten koko kehitysputkea. Editalla oltiin jo ennen automaatioprojektia aloitettu uusien ohjelmistokehitysprojektien kehittäminen Vue.js -sovelluskehyksellä, joten senkin takia oli luontevaa, että myös testiautomaatiota tehtäisiin jatkossa samaa ohjelmointikieltä tukevalla työkalulla. Palveluita piti pystyä testaamaan kaikilla yleisimmillä selaimilla ja tämän haluttiin olevan mahdollisimman vaivatonta. JavaScript -työkalun nähtiin tuovan myös tähän parannusta, eikä aiemmin pullonkaulana olleet IE ja Edge olisi yhtä ongelmallisia enää jatkossa.

Tärkeimmiksi kriteereiksi uuden työkalun valintaan vaikuttivat lopulta juuri mahdollisimman hyvä tuki eri selaimille, ohjelmointikieli, sekä työkalun nopea käyttöönotto ja vähäinen konfiguraation tarve. Muita vaatimuksia olivat esimerkiksi työkalulla tehtyjen automaatiotestien

soveltuvuus Dockerissa ajettaviksi, sen tulisi myös toimia Browserstackin ja Jenkinsin, tai jonkun muun sellaisen ajoympäristön kanssa, jolla ajoja pystyttäisiin suorittamaan ajastetusti. Lisäksi työkalun haluttiin tuottavan ajoista selkeitä testiraportteja.

2.2 Kehittämiskohteen kuvaus

Edita Publishingin ICT-tiimin työnkuvaan kuuluu yrityksen, pääasiassa erilaisiin lakitietopalveluihin keskittyvien verkkosivujen kehitys ja ylläpito. Näihin palveluihin kuuluvat mm. Edilex, Finlex, julkisethankinnat.fi, Stuklex, sekä monet muut vastaavanlaiset palvelut. Kehitys- ja ylläpitotyöt voi jakaa karkeasti kahteen osaan. Osista ensimmäinen sisältää vanhojen palveluiden ylläpitotöitä, jatkokehittämistä ja refaktorointia ja toinen kokonaan uusien palveluiden suunnittelua ja kehittämistä. Usean erillisen ja paljon erilaisia riippuvaisuuksia sisältävän palvelun kehitys- ja ylläpitotyö on haastavaa. Erilaiset riippuvuudet ja integraatiot lisäävät paljon virheiden riskiä, jonka vuoksi testausta on tehtävä jatkuvasti, ja välillä usean järjestelmän osalta saman aikaan.

Kehitysputken nopeuden ja luotettavuuden kannalta on keskeistä, että esimerkiksi regressiobugit saadaan heti kiinni. Näin niiden korjaaminen on helpompaa ja riski tuotantoon pääsevistä rikkinäisistä ominaisuuksista tai palveluista olisi mahdollisimman minimimaalinen. Palvelut ovat kooltaan suuria, ja sisältävät valtavan määrän kriittisiä toimintoja, joissa ei saa esiintyä virheitä. Näin monen suuren järjestelmän laajamittainen jatkuva manuaalinen testaaminen nielisi valtavasti aikaa ja resursseja, joten testiautomaatio on käytännössä ainoa järkevä vaihtoehto. Testauksen kohteena ovat kaikki yrityksen julkaisemat verkkosivut ja palvelut.

Kehitysvaiheessa palvelut menevät useamman ympäristön läpi ennen tuotantoon menoa. Ympäristö riippuu siitä missä kehitysvaiheessa palvelu sillä hetkellä on. Ympäristöinä oli ns. development, joka on lokaali kehitysympäristö, sekä demo- ja testiympäristö. Näistä demoympäristö on lähinnä tuotantoympäristöä ja viimeinen vaihe ennen julkaisua. Tarkoituksena oli, että testiautomaatio pyörii kaikissa tuotantoa edeltävissä ympäristöissä, jolloin mitään ei pääsisi tuotantoon ennen kuin kaikki ohjelmiston kehitysvaiheet ovat läpäisseet julkaisuputken testauksen.

Kehittämistavoitteina testiautomaation uudistuksessa oli, että palveluiden testikattavuus saataisiin testiautomaation avulla selvästi entistä suuremmaksi. Myös osiltaan vanhentuneet aiemmat automaatestit oli tarkoituksena ajantasaistaa tai kääntää uudelle työkalulle. Testiautomaatio haluttiin tilanteeseen, jossa jo olemassa olevien palveluiden testaus olisi katettu, ja uusia testitapauksia voitaisiin jatkossa tuottaa samaa tahtia, kun testattavaan palveluun tehdään muutoksia, tai kun kehitetään kokonaan uusia palveluita. Aiemmalla testiautomaatiolla haluttuun tilanteeseen ei oltu ikinä päästy. Monissa projekteissa ja palveluissa testiautomaatiota oli tehty erittäin minimaalisesti tai ei ollenkaan, ja testaus oli toteutettu lähes yksinomaan manuaalitestauksena projektien lomassa.

2.3 Aihealueen rajaus

Opinnäytetyön aihealue on rajattu käytännön osuuden osalta ainoastaan testiautomaatiotyökalun käyttöönottoon. Käyttöönotto on kuvattu noin puolen vuoden ajalta. Opinnäytetyöstä on jätetty pois uuden testiautomaatiotyökalun valintaprosessi. Lisäksi työssä ei käsitellä testaussuunnittelua, jota tehtiin testien implementointivaiheessa. Testiautomaatioprosessin parantaminen oli yksi käyttöönoton päätavoitteista ja sitä tehtiin käyttöönoton yhteydessä.

Teorian osalta työssä käydään läpi käyttöönotettavaa TestCafe -työkalua. Testiautomaatio ja sen tarkoitus kuvataan lyhyesti yleisellä tasolla. Lisäksi teoriaosuudessa käsitellään testiautomaatiotyökalujen tarkoitusta, ja niiden käyttämien keskeisten mallien eroavaisuuksia. Lisäksi työssä avataan lyhyesti siinä esiintyvät keskeiset käsitteet.

2.4 Keskeiset käsitteet

JavaScript, TypeScript: JavaScript ja TypeScript ovat ohjelmointikieliä, joita käytetään pääasiassa Web ympäristöissä. JavaScriptin avulla luodaan web sivuille toiminnallisuutta. TypeScript rakentuu JavaScriptin varaan, tuoden JavaScriptille lisää ominaisuksia, kuten vahvan tyypityksen.

Browserstack: Pilvessä toimiva testausalusta, joka mahdollistaa eri selainten käytön eri käyttöjärjestelmillä ja puhelimilla. Projektissa käytimme Browserstackia testien ajossa.

Jenkins: Avoimen lähdekoodin ohjelmisto, joka on kehitetty ohjelmistojen jatkuvan kehityksen ja testauksen helpottamiseksi. Jenkinsin avulla pystytään ajastamaan erilaisia toimintoja kuten testiautomaation ajastuksia tai automaattisia julkaisuprosesseja.

Regressiobugi: Regressiobugilla tarkoitetaan virhettä, joka tapahtuu, kun uusi koodi hajottaa jonkin jo olemassa olleen toiminnallisuuden

Robot Framework: Sovelluskehys, joka soveltuu lähes mihin tahansa automaatio testaukseen. Robot Framework tarvitsee jonkun rajapinnan itsensä ja automatisoitavan ohjelman väliin ja siihen onkin saatavilla todella kattava rajapintakirjastokokoelma.

PHP: Palvelinpohjainen skriptikieli, jota käytetään usein dynaamisten verkkosivujen tekemiseen. Lyhenne sanoista Hypertext Preprocessor.

Sprintti: Ketterän kehityksen mallin yksi kehitysjakso, joka tyypillisesti kestää 1-4 viikkoa sanotaan sprintiksi.

Kuratoitu: Tarkoittaa kerätyn sisällön jalostamista ja suodattamista oikealle kohderyhmälle. Edita Publishingilla esimerkiksi Rajalex on tietylle kohderyhmälle suodatettua tietoa, jossa on vain rajavartiolaitokselle oleellisia säädöksiä.

Docker: Sovellus, joka pystyy pakkaamaan sovelluksen sekä sen kaikki riippuviidet yhteen virtuaaliseen pakettiin, eli konttiin, jota pystytään siirtelemään esimerkiksi testiympäristöstä toiseen. Docker tarjoaa myös ajoympäristön millä pakattuja sovelluksia voidaan ajaa.

Mustache-malli: Mustache on template-työkalu, jonka avulla voidaan upottaa esimerkiksi muuttujan dataa html dokumenttiin. Se on monikäyttöinen ja useiden ohjelmointikielin tukema. Mustache on saanut nimensä aaltosulkeista ({{variable}}), jotka muistuttavat viiksiä.

HTML: Kieli, jolla verkkosivut ovat kirjoitettu. HTML:n avulla määritellään rakenne verkkosivuille. Lyhenne sanoista Hypertext Markup Language.

DOM: Document Object Model on tapa kuvata HTML:n, XML:n tai XHTML:n rakenne puuna. DOM-puun elementtejä voidaan etsiä, hakea ja manipuloida esimerkiksi JavaScriptin avulla.

YAML: Merkintäkieli, joka on suosittu erityisesti konfiguraatiotiedostoissa sekä sovelluksissa, joissa tietoja tallennetaan ja siirretään.

JSON: Yksinkertainen avoimen standardin tiedostomuoto tiedonvälitykseen. Lyhenne sanoista JavaScript Object Notation.

3 Testiautomaatio

Testiautomaation tarkoituksena on tehostaa ja parantaa ohjelmistokehityksen laadunvarmistusprosessia. Automatisoitujen testien avulla saadaan kuormaa pois manuaalitestaukselta, jolloin muille töille, kuten esimerkiksi tutkivalle testaukselle, jäisi enemmän aikaa. Automaatiotestien avulla testausta voidaan tehdä manuaalitestausta nopeammin, useammin ja kattavammin, jolloin erilaiset virhetilanteet ja regressiobugit saadaan helpommin kiinni. Testiautomaatio on tärkeä osa sovelluksen laadunvarmistusta. (Fewster & Graham 1999, 5.)

Testiautomaatiolla ei voi korvata kokonaan manuaalista testausta, eikä se ole sen tarkoituksenakaan. Testiautomaation tuomaa taloudellista hyötyä ei voi arvioida saman tien kustannusten painottuessa projektien alkupäähän, jossa automaation suunnitteluun, testiskriptien kirjoittamiseen ja ylläpitoon kuluu suhteellisesti paljon enemmän aikaa verrattuna tilanteeseen, jossa valmis automaatio on jo olemassa. Tämän vuoksi testiautomaation käyttöönotto voikin tuntua aluksi jopa kalliilta sijoitukselta. Testien ylläpito ja automatisoitujen testitapausten kirjoittaminen nielee enemmän resursseja kuin manuaalitestauksessa. Tämän vuoksi on erittäin tärkeää toimia suunnitelmallisesti ja miettiä tarkasti testit joihin automaatio soveltuu ja joiden automatisoinnista saadaan suurin hyöty. Turhien testien kirjoitus maksaa paljon rahaa ja vie turhaan aikaa. Mitä paremmin automaatiotestit suunnitellaan, sitä enemmän hyötyjä testiautomaatiolla saadaan aikaan. Testiautomaation hyödyt manuaaliseen testaukseen nähden nähdäänkin parhaiten vasta sen jälkeen, kun automatisoitujen testien kattavuus alkaa olemaan hyvä, ja testejä on ajettu enemmän kuin kerran. (Fewster & Graham 1999, 5.)

Testiautomaatiolla saavutetaan monen laisia hyötyjä ja niistä tärkeimpiä ovat:

- Nopeampi palaute – automatisoiduista testeistä saadaan palaute heti kun testi on suoritettu, eikä niistä tarvitse erikseen kirjoitella erilaisia testiraportteja
- Regressiotestaus – automatisoiduilla testeillä pystytään heti varmistamaan, että sovellus toimii siihen kohdistuneiden muutosten jälkeen edelleen oikein.
- Testaajien aikaa säästyy – testien suorituksen yhteydessä testaajien aikaa säästyy ja he voivat keskittyä enemmän uusien ominaisuuksien testaukseen
- Testaamisen tehokkuuden paraneminen – testejä pystytään tekemään myös sellaisina kellon aikoina, kun testaajat ovat itse nukkumassa. Testiautomaatio ei tarvitse taukoja.
- Parempi testikattavuus – testiautomaatiolla pystytään suorittamaan suuri joukko testejä kerrallaan. Manuaalisesti samaan pääseminen vaatisi suurta joukkoa testaajia ja paljon kulutettuja työtunteja
- Testien uudelleenkäytettävyys – Automaatiotestejä pystytään usein käyttämään ainakin osittain uudelleen ja siten samat testirungot ovat hyödynnettävissä monissa eri testeissä.
- Viat havaitaan aiemmin – Tämä lisää yleistä kehitysnopeutta. Kun virheet huomataan aiemmin, niin virheet korjataankin aiemmin. (Tanna 2017.)

3.1 Yleistä testiautomaatio työkaluista

Testiautomaatiotyökalut ovat työkaluja laadunvarmistuksen helpottamiseksi ja nopeuttamiseksi. Testiautomaatiotyökaluja on tarjolla nykyisin todella paljon ja erilaisiin käyttötarkoituksiin soveltuvia työkaluja löytyy lukuisia. Testiautomaatiotyökalut tarjoavat erilaisia rajapintoja ja viitekehyksiä, joiden avulla testejä voidaan kirjoittaa. Viitekehykset sisältävät tyypillisesti esimerkiksi valmiita sisäänrakennettuja malleja ja kirjastoja, joiden avulla testien kirjoittaminen pääsee nopeasti alkuun ja testeistä tulee automaattisesti ainakin jossain määrin ylläpidettäviä. Lähes jokaisella työkalulla on omanlaisensa skriptaustekniikka. Testiautomaatiotyökalujen käytöstä saadaan monia hyötyjä ja niiden käyttö edesauttaa automaatioprojekteille asetettujen tavoitteiden saavuttamista, jotka tyypillisesti ovat: testikattavuuden lisääminen, kehitysputken tarkkuuden ja tehokkuuden parantaminen, sekä testien ylläpitokustannusten suhteellinen pieneneminen. Oikean testiautomaatiotyökalun löytäminen edesauttaa uudelleenkäytettävien ja helposti ylläpidettävien testien kirjoittamista. (Aebersold 2019.)

Testiautomaatiotyökalut jaotellaan erilaisiin malleihin tai kehyksiin (framework), joilla kaikilla on omanlaisensa arkkitehtuuri. Eri arkkitehtuuriin perustuvilla työkaluilla tavoitellaan hieman erilaista lähestymistapaa testiautomaatioon, ja niillä on hieman toisistaan poikkeavat vahvuudet ja heikkoudet. Yleisiä arkkitehtuurimalleja ovat esimerkiksi lineaarinen, modulaarinen, kirjastoperustainen, data-driven, keyword-driven ja hybridimallit. (Aebersold 2019.)

Näkyvimmät erot erilaisiin arkkitehtuurisiin malleihin perustuvissa testiautomaatiotyökaluissa ovat yleensä niissä käytettävissä skriptaustekniikoissa. Lineaariseen malliin perustuvat testiautomaatiotyökalut tunnetaan usein ns. Record and Playback -tyyppisinä työkaluina. Lineaarista mallia käyttäviä työkaluja käytetään yleensä pienikokoisten sovellusten testaamiseen. Lineaarisessa mallissa jokainen testitapaus kirjoitetaan ja suoritetaan erikseen. Stepit kirjoitetaan niin kuin ne tehtäisiin manuaalisestikin. Lineaariseen malliin perustuvilla työkaluilla on hyvä opetella testiautomaation kirjoittamista, mutta se ei sovellu sellaisenaan isojen sovellusten testaamiseen, koska testitiedot koodataan testiskriptin sekaan. Tämä aiheuttaa sen, että testit eivät yleensä ole uudelleenkäytettäviä ja niiden ylläpidettävyys on heikko. Suurin osa muihinkin arkkitehtuureihin perustuvista testiautomaatiotyökaluista mahdollistaa lineaarisella mallilla toteutetun, nauhoita ja toista, skriptien kirjoittamisen, jolloin sitä voidaan käyttää tilanteissa, joihin se soveltuu. Lineaarisen mallin suurin etu on testiskriptien nopea kirjoittaminen, koska skriptien suunnitteluun menee yhtä vähän aikaa kuin manuaalisiinkin testeihin. (Fewster & Graham 1999, 75-75.)

Modulaarinen malli perustuu siihen, että testiskripti jaetaan pieniksi toisistaan riippumattomiksi moduuleiksi, joista jokainen on suoritettavissa erikseen. Mallin tarkoituksena on, että näitä pieniä uudelleenkäytettäviä moduuleita yhdistelemällä voidaan rakentaa lukuisia testejä ilman, että kaikkea joudutaan kirjoittamaan uudelleen. Moduulit voisi nähdä pieninä rakennuspalikoina, joita yhdistelemällä voidaan rakentaa isoja kokonaisuuksia. Kirjastoarkkitehtuurinen malli on hyvin lähelle samanlainen kuin modulaarinen kehys, ja se onkin kirjoitettu modulaarisen kehyksen päälle. Modulaarisen ja kirjastoarkkitehtuurisen mallin erona on se, että kirjastoarkkitehtuurisessa tavassa ei jaeta testitapauksia varsinaisesti moduuleiksi kuten modulaarisessa mallissa, vaan moduuleiden sijasta kirjasto sisältää funktioita, joita voidaan kutsua testiskriptistä ja jotka kutsusta suorittavat pieniä niihin ohjelmoituja itsenäisiä toimintoja. Modulaarisuus ja toimintalogiikan jakaminen pieniin itsenäisiin moduuleihin tekee testien ylläpidosta helpompaa ja kustannustehokkaampaa. Moduulien ja funktioiden uudelleenkäytettävyysaste on korkea ja on tyypillistä, että ne sisältävät toimintoja, joita voidaan käyttää useissa testiskripteissä ja jopa useissa projekteissa. Uudelleenkäytettävyys takaa hyvän ylläpidettävyyden. (Aebersold 2019.)

Data-driven mallissa testitiedot tai testidata erotetaan ja pidetään testiskriptien ulkopuolella. Testitiedot luetaan ulkoisista tiedostoista, kuten Excel-tiedostoista, tekstitiedostoista, tietokannoista, tai vaikka CSV-tiedostoista. Testidata ladataan muuttujiin testiskriptin sisällä

ja näiden muuttujien tietoja voidaan käyttää testeissä esimerkiksi syötteinä tai referenssiarvoina, joita voidaan käyttää varmennusarvoina luettaessa arvoja testattavasta systeemistä. Testiskriptit kirjoitetaan data-driven mallissa usein lineaarisella skriptaustekniikalla. Data-driven mallin vahvuutena voidaan nähdä se, että datan muuttuessa testiskriptejä ei tarvitse muuttaa. Lisäksi samat testitapaukset voidaan suorittaa useilla datajoukoilla. Huonona puolena data-driven mallissa voidaan pitää testien suunnittelun vaatimaa työmäärää, joka on iso verrattuna joihinkin muihin tekniikoihin. (Fewster & Graham 1999, 83-87.)

Keyword-driven, joka tunnetaan myös nimillä table-driven ja action-word malli, on toiminnallinen automaatiotestauskehysmalli. Malli perustuu siihen, että jokaiselle suoritettavaksi halutulle toiminnolle on annettu avainsana, jota kutsumalla siihen liitetty toiminto suoritetaan. Kuten data-drivenissä, myös avainsana pohjaisessa mallissa testidata ja testiskripti ovat erillisissä tiedostoissa. Jokainen avainsana esittää yksittäistä toimintoa ja sen takana on skripti, joka osaa suorittaa avainsanan kuvaaman toiminnon. Tyypillisiä avainsanoja voivat olla esimerkiksi:

- selaimen avaaminen -> openbrowser
- hiiren klikkaus -> click
- näppäimistön painaminen -> keypress
- tekstikenttään kirjoittaminen -> typetext jne.

Avainsanapohjaisen lähestymistavan parhaita puolia on uudelleenkäytettävyys ja helppo testien ylläpito. (Fewster & Graham 1999, 89-91.)

Hybridimallinen testauskehys voi olla sekoitus kaikista edellä mainituista arkkitehtuurisista malleista. Nykyisin automatisoidut testauskehykset ovat alkaneet integroitumaan ja kehittymään siihen suuntaan, että yhdellä kehyksellä/työkalulla pystyttäisiin toteuttamaan kulloiseenkin tilanteeseen sopivinta tekniikkaa. Tämä suuntaus on seurausta siitä, että jokainen testattava sovellus on erilainen ja niihin tarvitaan erilaisia lähestymistapoja. Yhdellä monikäyttöisellä työkalulla saavutetaan etuja vain yhden asian hallitsevaan työkaluun nähden. Hybridikehykset ovat kombinaatioita muiden kehyksien hyvistä ominaisuuksista. Hybridikehystä pystytään mukauttamaan erilaisiin testaustarpeisiin, jolloin saavutetaan parempia tuloksia helpommin ja kustannustehokkaammin. (Aebersold 2019.)

3.2 Testcafe testiautomaatio työkalu

Testcafe on Node.js end-to-end testiautomaatiotyökalu, joka on tarkoitettu web-sivujen testaamiseen. TestCafe tukee JavaScript ja TypeScript -ohjelmointikieliä. TestCafe tarvitsee toimiakseen npm-pakettienhallintatyökalu (Npm), sekä Node.js:n version, joka on 6 tai uudempi.

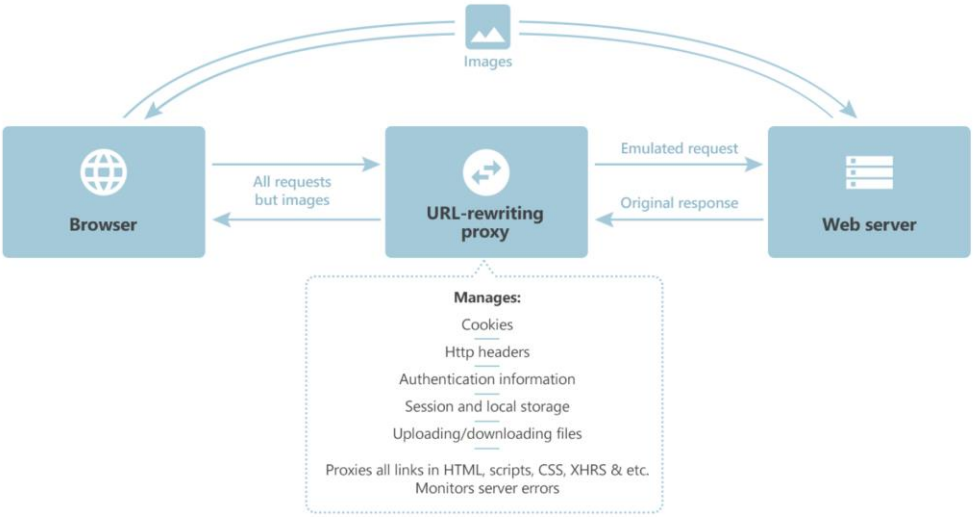
(Testcafe 2019.) Testcafen arkkitehtuuri on sekoitus Lineaarista, Data-driven-pohjaista, ja avainsanapohjaista mallia.

Npm on komentoriviltä toimiva Node.js:n paketinhallintatyökalu. Node.js käyttää npm:ää oletuksena paketinhallintaansa ja ainakin uudemmissa Node -versioissa npm on sisällytetty Node.js:n asennuspakettiin. Npm maailman eniten käytetty paketinhallintatyökalu ja se sisältää yli 800 000 koodipakettia, joita sen kautta voi ladata käyttöönsä. Npm sisältää cli-rajapinnan, joka mahdollistaa npm-pakettien lisäämisen ja hallinnoinnin Node.js-projektissa. (Npm 2019.)

Node.js on avoimeen lähdekoodin perustuva JavaScript-pohjainen ohjelmointialusta, joka on rakennettu Google Chrome V8 JavaScript moottorin päälle. Node.js mahdollistaa JavaScript koodin suorittamisen palvelimella, kun ilman sitä JavaScriptiä voidaan suorittaa ainoastaan selaimessa. Node.js käyttää event-pohjaista asynkronista I/O -mallia, joka tekee siitä kevyen ja erittäin nopean. Asynkronisuus tarkoittaa sitä, että node.js serveri ei pysähdy oletusarvoisesti odottamaan vaikkapa apilta tulevaa dataa, vaan muu ohjelma jatkaa suoritustaan odotuksen aikana. Node.js on alustariippumaton, eli sitä pystyy käyttämään millä tahansa käyttöjärjestelmällä. (Node.js 2019.)

Testcafe toimii Windows, MacOs ja Linux -käyttöjärjestelmissä. TestCafe tarjoaa tuen kaikkiin käytetyimpiin selaimiin. Tuettuja selaimia ovat Google Chrome, Internet Explorer (11+), Edge, Mozilla Firefox, Safari, Google Chrome mobile, sekä Safari Mobile. TestCafe ei tarvitse toimiakseen selainlaajennuksia, selainajureita, tai muitakin rippuvaisuuksia, joka tekee Testcafen konfiguroinnista ja asentamisesta yksinkertaista ja nopeaa. Testcafe tukee testien suorittamista ns. headless-tilassa, jolla tarkoitetaan sitä, että selaimessa voidaan ajaa testejä ilman, että selainta avataan näkyville. Testcafe soveltuu käytettäväksi myös pilvipohjaisten selainpalveluiden, kuten Browserstackin tai Sauce Labin kanssa. Yksi TestCafen tärkeimmistä ominaisuuksista on ’remote’-testaus, jolla tarkoitetaan sitä, että suoritettava testi voidaan jakaa paikallisverkossa muille laitteille. Remote-tuki koskee myös mobiililaitteita. TestCafella suoritettavat testit voidaan käynnistää joko komentoriviltä tai TestCafe:n oman TestCafe Studion kautta. (Testcafe 2019.)

TestCafe hallinnoi testattavaa järjestelmää Hammerhead -nimisen välityspalvelimen kautta. Välityspalvelin injektoi sivulle TestCafelta tulevan JavaScript-koodin, jonka avulla TestCafe pystyy etsimään käyttäjän haluamat elementit domista ja simuloimaan käyttäjän toimia käyttöliittymärajapinnassa (kuvio 1). Teknisesti ottaen TestCafe upottaa toiminnallisuuksiinsa tarvitsemat rajapintafunktiot ohjelmassa jo olevan JavaScriptin-koodin sekaan, jonka jälkeen testin kirjoittaja voi ohjailla ohjelmiston toimintaa kirjoittamiensa funktiokutsujen, eli testiskriptin avulla. (DevExpress 2019.)



Kuvio 1: Välityspalvelimen kulku (Moskovkin 2017)

TestCafe-testit suoritetaan aina ns. suojatussa tilassa, joka tarkoittaa sitä, että selaimen välimuisti on testin alkaessa tyhjä. Näin testit pystytään ajamaan aina samasta lähtökohdasta, eikä käyttäjällä mahdollisesti välimuistissa tai evästeissä oleva data vaikuta ohjelman käyttäytymiseen. Kun selaimen välimuistissa on ainoastaan sinne testiajojen aikana kertynyt data, joka nollautuu aina kun selain avataan uudelleen, niin samoja testejä voidaan suorittaa kerta toisensa jälkeen ilman, että aiemmin suoritetut testit vaikuttaisivat uuteen suoritukseen. Selainsiirtymiä ja niihin liittyviä sivujen latautumisia varten TestCafessa on sisäänrakennettu asynkroninen funktio, jonka tehtävänä on huolehtia siitä, että sivu on täysin latautunut ennen kuin testiä aletaan suorittamaan, joten erillisiä timeoutteja ei yleensä tarvitse määritellä. (Testcafe 2016; DevExpress 2019.)

TestCafe käyttää CSS-valitsimia Html-elementtien etsimiseen. Työkalu pystyy paikallistamaan elementtejä myös niiden sisältämien visuaalisten tekstien perusteella, jolloin testiskriptin kirjoittajan välttämättä tarvitse olla JavaScript-ohjelmoija, tai omata kovin syvää dom-tietämystä. TestCafessa on omanlaisensa testistruktuuri, jossa testit kirjoitetaan asynkronisen testifunktion sisään (kuvio 2). (TestCafe 2019.)

```
import { Selector } from 'testcafe';

fixture `Fixture name`
    .page `https://www.examle-site.fi`; //sivun osoite mille navigoidaan kun testin ajo alkaa

const userNameInput = Selector('#username-input');
const submitButton = Selector('button[id="submit"]');

//asyncronin testi functio
test('My first test', async t => {   //sulkujen sisällä testin nimi
    await t
        //kirjoitetaan input kenttään typeText action wordin avulla
        .typeText(userNameInput, 'Jonna Partanen')
        .click(submitButton) //klikataan painiketta jonka id on submit

        //klikkauksen jälkeen tarkistetaan assertion metodin avulla että oikeanlainen
        //otsikko ilmestyy ruuttuun
        .expect(Selector('h1').withText('Welcome, Jonna Partanen').exists).ok()
});
```

Kuvio 2: Yksinkertainen TestCafe testi, jossa testi kirjoitetaan asynkronisen testi funktion sisään

Testit kirjoitetaan pääsääntöisesti käyttäen TestCafen omia action word -metodeita ja assertion metodeita. Action word -metodit ovat metodeita, jotka suorittavat jonkun toiminnon kuten klikkaa, kirjoita, avaa selain – eli matkivat käyttäjän ohjelmassa suorittamia toimintoja. Sisäänrakennettuja action word metodeita ovat mm. click, typeText, keyPress, navigateTo, hover, drag ja takeScreenshot. Metodit pohjautuvat aiemmassa teoriakappaleessa esiteltyyn keyword-driven arkkitehtuuriin. Assertion -metodeilla puolestaan viitataan toiminnallisuuksiin, jotka eivät varsinaisesti tee mitään näkyvää, mutta niiden avulla voidaan vahvistaa esim. määrätyn datan löytyvän määrätystä paikasta, kun ohjelma on tietyssä tilassa. TestCafen assertion-metodit perustuvat behavior-driven development (BDD) arkkitehtuuriin, jonka lähtökohtana on testien rakentuminen siten, että voidaan varmistua testattavan ohjelmiston käyttäytyvän sille määrätyllä tavalla erilaisissa tilanteissa (TestCafe 2019.)

TestCafe -testeihin voidaan työkalun tarjoamien valmiiden metodien lisäksi kirjoittaa myös omia ns. client-funktioita, tai lisätä toiminnallisuuksia eri tarpeisiin kirjoitettujen valmiiden node.js kirjastojen avulla, joita voidaan ottaa käyttöön helposti npm:n avulla. (TestCafe 2019.) Testejä rakennetaan ketjuttamalla toimintoja kuvion 3 osoittamalla tavalla.

```
await t
    //kutsutaan useRole metodia, jonka taustalla on palveluun sisään kirjaus
    .useRole(regularUser)
    //navigoidaan uudelle sivulle
    .navigateTo(`${baseUrl}/new-page`)
    //kirjoitetaan hakukenttään
    .typeText(searchInput, 'koulutus', {speed: 0.03})
    //klikataan haku painiketta
    .click(searchButton)
    //klikataan haun tuloksena tullutta ensimmäistä osumaa
    .click(Selector('firstSearchResult').nth(1))
```

Kuvio 3: Toimintojen ketjutus TestCafeella

3.3 Järjestelmän käyttöönotto

Järjestelmän käyttöönotolla tarkoitetaan sitä että, jokin uusi järjestelmä tai työkalu otetaan säännönmukaisesti käyttöön, tai jokin vanha järjestelmä päivitetään uudempaan parannel-tuun versioon. Järjestelmän käyttöönoton tarkoitus on tuottaa hyötyä. Tyypillisesti käyttöön-otto voi liittyä esim. yrityksen tuottavuuden/tehokkuuden ja sitä kautta yrityksen kilpailuky-vyn lisäämiseen. Käyttöönotot ovat välttämättömiä prosesseja yrityksen elinkaaressa. Aloitta-vat yritykset joutuvat usein rakentamaan oman infrastruktuurinsa tyhjästä. Pidempään toimi-neet yritykset voivat olla tilanteessa, jossa yrityksen olemassa olevat järjestelmät ja työkalut ovat tulleet elinkaarensa päähän, eivätkä palvele enää sellaisenaan yrityksen tarpeita ja ne pitää vaihtaa tai päivittää uuteen. (Kettunen & Simons 2001, 25; Harju 2004, 9.)

Käyttöönotto on usein todella pitkä ja kallis prosessi, jonka hyödyt näkyvät vasta pidemmällä aikavälillä. Käyttöönottoprojekteissa on tärkeää määritellä mistä käyttöönotto aloitetaan ja mihin käyttöönotto lopetetaan. Käyttöönotto voidaan toteuttaa kertasiirtymisenä, jolloin vanha järjestelmä tai työkalu korvataan välittömästi kokonaisuudessaan uudella tai se voi-daan tehdä vaiheistetusti, jolloin uutta ja vanhaa työkalua käytetään jonkin aikaa samanai-kaisesti. Vaiheittaisella käyttöönotolla pyritään varmistamaan, että uusi järjestelmä varmasti toimii halutulla tavalla ennen kuin vanha korvataan kokonaan. Monesti käyttöönotto tapahtuu vaiheittain. (Harju 2004, 9.)

Oli käyttöönottoprojektissa kyse isosta tai pienestä järjestelmästä, niin siihen liittyy lukuisia erilaisia työvaiheita. Projektin kulku vaihtelee esim. sen mukaan onko kyseessä kokonaan uu-den järjestelmän käyttöönotto, vai päivitetäänkö vanha järjestelmä vain uudempaan versi-oon. Käyttöönotosta tulee aina tehdä käyttöönottosuunnitelma, joka sisältää mm. järjestel-män tai työkalun asennuksen ja valmistelun, henkilöstön koulutuksen ja vanhojen tietojen

konvertoinnin. (Halonen 2002.) Käyttöönottosuunnitelmassa on yksilöity ainakin sen päätavoite, etapit ja aikataulut. Mitä tarkemmin etapit ovat määritelty, sen paremmin käyttöönoton etenemistä pystytään seuraamaan. (Harju 2004, 35.)

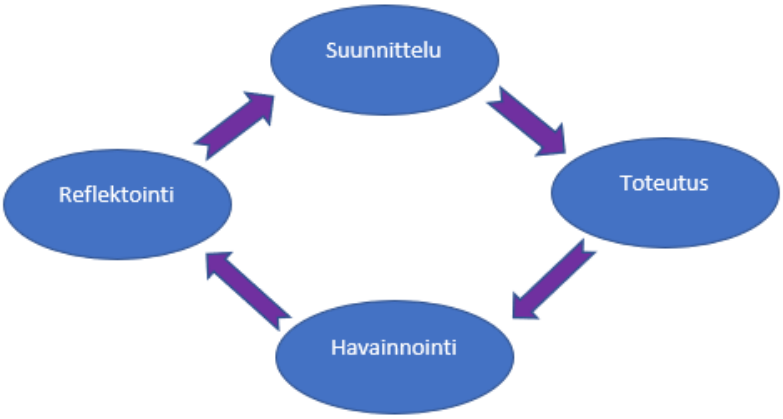
4 Tutkimus- ja kehittämismenetelmät

Käyttöönoton yksi tarkoituksista oli muuttaa aiempia toimintatapoja ja saada sitä kautta parempia tuloksia aikaiseksi. Mitä paremmin ja nopeammin saisimme regressiobugeja kiinni, sitä nopeammin saisimme kehitystöitä eteenpäin ja palvelumme olisivat laadukkaampia. Käyttöönoton alussa lähdimme suunnittelemaan, tutkimaan ja kokeilemaan tulevaa testiautomaatiotyökalua. Käyttöönoton aikana törmäsimme moniin erilaisiin ongelmiin, joita ratkoimme etsimällä tietoa erilaisista aiheeseen liittyvistä lähteistä. Käyttöönottoon pääasiallisena tutkimusmenetelmänä käytimme toimintatutkimusta, osallistuvaa havainnointia sekä hiljaisen tiedon tutkimusta.

4.1 Toimintatutkimus

Toimintatutkimuksella yritetään muuttaa nykyisiä toimintatapoja, suunnittelemalla ja kokeilemalla ja tutkimalla. Toimintatutkimus voi olla niinkin pieni kuin oman toimintatavan kehittäminen ja tutkiminen, mutta yleensä se on isompi ja siksi työn kehitykseen tarvitaan yhteistyötä ja kommunikaatiota. Toimintatutkimuksella tavoitella käytännön hyötyjä ja etsitään ongelmille ratkaisuja. Toimintatutkimuksen päämääränä on käytännön hyöty ja käyttökelpoisen tiedon lisääntyminen eli sillä myös pyritään kehittämään osanottajien tietoja ja käytännön osaamista sekä asiantuntemusta. Toimintatutkimus soveltuu erittäin hyvin ketteriin menetelmiin, tiimi projekteihin sekä erilaisiin käyttöönottoprojekteihin. (Heikkinen & Rovio & Syrjälä 2006, 17,22.)

Toimintatutkimus tapahtuu usein sykleissä, joissa toimintatapoja kokeillaan ja tutkitaan. Syklissä on neljä vaihetta, jotka seuraavat toisiaan. Syklin vaiheet ovat toiminnan suunnittelu, toteutus, havainnointi ja reflektointi (kuvio 4). Tätä sykliä jatketaan niin, että aina reflektoinnin jälkeen ja sen päätyttyä, reflektoinnin pohjalta lähdetään suunnittelemaan vieläkin parempaa toimintatapaa. (Heikkinen & Rovio & Syrjälä 2006, 17 ja 35.)



Kuvio 4: Toimintatutkimuksen sykli

4.2 Hiljaisen tiedon tutkimus

Hiljainen tieto on niin sanotusti kokemuksesta saatua tietoa ja taitoja. Hiljaista tietoa saadaan työkokemuksista, haasteista, opiskelusta sekä tekemällä oppimisen kautta. Hiljaista tietoa voidaan kartuttaa työympäristöissä oppimalla kollegoilta, heidän jakaessa tietojaan ja taitojaan sekä havainnoimalla heitä. Joka kerta kun pähkäilemme jonkin ongelman kanssa tai tutkimme työnalla olevaa tehtävää, kerrytämme siitä hiljaista tietoa. (Pohjalainen 2012, 1-10.)

Organisaatioiden sisällä tärkeää, että hiljaista tietoa sekä osaamista jaettaisiin työntekijöiden keskuudessa riittävästi esimerkiksi erilaisilla workshopeilla, näin kartutettaisiin työntekijöiden hiljaista tietoa ja tätä kautta myöskin työntekijöiden osaamista. (Pohjalainen 2012, 1-10.)

Jokaisesta työtehtävästä, kouluprojektista sekä havainnoimalla ympäristöä olen saanut itsekin hiljaista tietoa ns. vertaisoppimalla, vaikka en välttämättä pysty kertomaan mitä. Tässä opinnäytetyössä ja testiautomaatiotyökalun käyttöönotossa olen pystynyt hyödyntämään aiemmin saamaani hiljaista tietoa ja myöskin tämän projektin aikana olen kartuttanut hiljaisen tiedon varastoani.

4.3 Osallistuva havainnointi

Osallistuva havainnointi on tiedonkeruu menetelmä, jossa havainnoitsija on projektissa osallistujana ja toisaalta seuraa myös muiden käyttäytymistä. Osallistuva havainnointi on tietoista ja järjestelmällistä osallistumista, jossa tarkoituksena on kerätä aineistoa ja tietoa suoraan osallistumisen välityksellä. (Grönfors & Vilkka 2011, 52-55.)

Osallistuva havainnointi jaetaan usein kahteen eri alalajiin, jotka ovat passiivinen ja aktiivinen havainnointi, riippuen siis siitä kuinka aktiivisesti havainnoitsija osallistuu projektiin. Aktiivisessa osallistuvassa havainnoinnissa havainnoija vaikuttaa itsekin läsnäolollaan tutkimukseen ja on näin kokonaisvaltaisesti ryhmän jäsen. Passiivinen havainnoijan roolina on taas olla ainoistaan havaintojen tekijä ryhmässä, eikä vaikuta tutkimukseen muuten kuin kysymyksien muodossa. (Grönfors & Vilkka 2011, 52-55.)

4.4 Reliabiliteetti ja validiteetti

Validiteetilla ja reliabiliteetilla tutkitaan, onko tutkimusmenetelmä tai tutkimuksen tuloksista saadut päätelmät valideja tai reliaabeleja. Validiteetilla ilmaistaan, kuinka hyvin tutkimuksessa käytetty menetelmä tutkii juuri sitä asiaa mitä on tarkoituskin. Validiteetissa vastataan kysymyksiin siitä, onko tutkimus pätevä, onko tutkimus perusteellinen tai onko saadut tulokset ja päätelmät oikeita. (Saaranen-Kauppinen & Puusniekka, 2006; Hiltunen 2019.)

Reliabiliteetin tarkoituksena on kertoa tai osoittaa, onko tutkimuksen tulos toistettavissa ja että tutkimuksen tulos ei ole sattumanvarainen. Reliabiliteetti arvioi tulosten pysyvyyttä ja jos tutkimuksessa saadaan sama tulos riippumatta tutkijasta, tutkimus voidaan sanoa olevan reliaabeli. (Hiltunen 2019.)

5 TestCafen käyttöönotto

TestCafeen käyttöönottovaihe jakautui viiteen isompaan kokonaisuuteen (kuvio 5). Näistä ensimmäinen oli projektin ja valitun toimintamallin sprinttien aikataulutus, käytettävien ympäristöjen määrittely sekä implementointivaiheen karkea testaussuunnitelma.

Toinen vaihe sisälsi käytettävien ympäristöjen pystyttämisen sekä käyttöönotettavan Frameworkin toiminnan opettelun ja soveltuvuuden varmistamisen. Kolmas vaihe oli implementointivaihe, joka jakautui kahteen osaan. Ensimmäisessä käännettiin jo olemassa olevat Robot Framework –automaatiotapaukset TestCafeelle ja vasta tämän jälkeen aloitettiin uusien testiautomaatiotapausten implementointi.

Neljännessä vaiheessa ratkottiin käyttöönotossa siihen mennessä esiintyneet haasteet ja ongelmat. Viides vaihe olikin sitten jo itse käyttöönotto, jossa ajoympäristöt konfiguroitiin suorittamaan implementoituja testejä.

Kuvio 5: Käyttöönoton eteneminen

5.1 Alustavan tehtäväluettelon ja aikataulun luominen

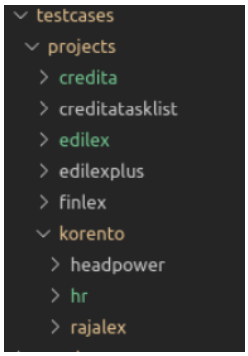
TestCafe testiautomaatiotyökalun käyttöönotto aloitettiin suunnittelupalaverilla, johon osallistuivat projektissa mukana olevat henkilöt. Suunnittelupalaverissa avattiin syitä, miksi TestCafe työkalu oltiin valittu tiimin projekteissa käytettäväksi testiautomaatiotyökaluksi ja minkälaisia hyötyjä sen käyttöönotosta odotetaan saatavan. Taustoituksen ja tavoitteiden lisäksi suunnittelupalaverin agendaan kuului käyttöönottoprosessin toimintasuunnitelman laatiminen. Toimintasuunnitelman keskeisinä kohtina oli paitsi käytettävien ympäristöjen, työjärjestyksen ja aikataulutuksen laatiminen, niin myös siihen osallistuvan henkilöstön osaamisen varmistaminen sekä miten projektiin osallistuvat henkilöt hankkisivat riittävän osaamisen kyseisestä testiautomaatiotyökalusta ja kuinka kauan opiskeluun pitäisi varata aikaa.

Suunnittelupalaverissa aikaansaatu toimintasuunnitelma sisälsi lopulta alustavat ohjeistukset siitä, miten käyttöönotto tullaan toteuttamaan ja yksityiskohtaisemman suunnitelman siitä mitä lähtisimme tekemään ensimmäiseksi. Työtä sovittiin tehtäväksi parin viikon pituisissa sprinteissä, vaikkakaan varsinaista Scrum-mallia ei projektissa käytetty. Lisäksi sovittiin, että jokaisen kahden viikon sprintin jälkeen tiimi pitää erillisen ainoastaan testiautomaation käyttöönottoon liittyvän palaverin. Näiden välipalaverien tarkoituksena oli käydä tarkasti läpi siihen asti tehtyjen asioiden onnistuminen ja projektin etenemä sekä sopia näiden havaintojen pohjalta seuraavan jakson sisällöstä.

Käyttöönoton toteuttavaan projektitiimiin kuului lisäkseni kolme henkilöä, mutta tarkoituksena oli, että käyttöönoton jälkeen yrityksen kehitystiimin muut työntekijät opastetaan työkalun käyttöön mahdollisuuksien ja tarpeiden mukaan. JavaScript on ehkäpä keskeisin kehi-

tystiimin käyttämä ohjelmointikieli, joten TestCafen omaksumisen odotettiin käyvän nopeasti. Tuttuudella ja helposti omaksuttavuudella voidaan saavuttaa selkeitä hyötyjä, koska kuka tahansa tiimin jäsen voisi tarvittaessa osallistua testiautomaation tekemiseen muun kehitystyön ohella.

Isojen linjojen lisäksi sovimme suunnittelupalaverissa mitä otamme mukaan ensimmäiseen sprinttiin. Päätökset tehtiin myös erilaisista projektinaikaisista käytännöistä, joita olivat esimerkiksi projektissa käytettävien hakemistojen rakenne, testitiedostojen nimeäminen, sekä itse testien nimeäminen ja strukturointi. Nimeämiset, strukturointi ja rakenteet pidettiin tuttuuden vuoksi lähes samanlaisina, kun ne olivat aiemmin olleet Robot Frameworkilla toteutetussa automaatiossa. Käytännössä tämä tarkoitti sitä, että testit jaettiin projekteihin ja palveluihin, joihin testiautomaatio kohdistui (kuvio 6).



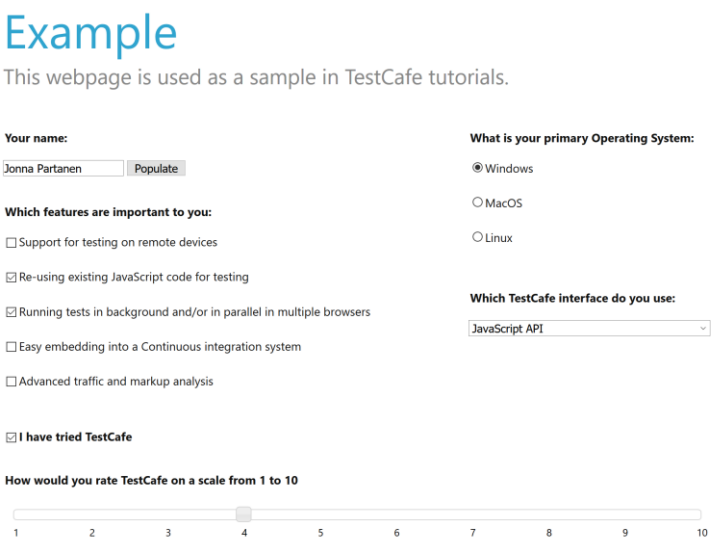
Kuvio 6: Testien jako projekteihin

5.2 Automaatioympäristön pystytys ja testi Frameworkiin tutustuminen

Ensimmäiseen käyttöönottosprintin tavoitteena oli testiympäristön pystyttäminen ja käytännön tutustuminen käyttöönotettavaan työkaluun. Testiympäristönä toimi alkuun paikallinen palvelinkone, joka pyöritti ajastetusti versionhallintaan vietyjä testejä sekä kehittäjien omien koneet, jotka toimivat kehittäjien henkilökohtaisena kehitysympäristönä. Ensimmäisenä työnä olikin projektin perustaminen versionhallintaan. Versionhallintatyökaluna sovittiin käytettäväksi Githubia, jonne perustettiin TestCafeTests niminen hakemisto. Github oli luonnollinen valinta versionhallintatyökaluksi, koska sitä käytettiin valmiiksi kaikissa muissakin tiimin projekteissa. Versionhallinnan käytöllä mahdollistettiin se, että koko tiimi pystyi kehittämään testejä yhtäaikaisesti ja kaikki testit olisivat kootusti samassa paikassa kaikkien tiimin jäsenten saatavilla. Versionhallinnan hyötynä on myös se, että testit ovat varmassa tallessa. Kun testien määrä kasvaa ja käytettyjen tuntien määrä kasvaa niin versionhallinnan tärkeys konkretisoituu.

Versionhallinnan pystytyksen jälkeen kehitystiimin jäsenet aloittivat itsenäisen tutustumisen TestCafe-työkaluun, joka sisälsi pääasiassa dokumentaation lukemista sekä käytännön harjoittelua. Ennen käytännön harjoittelua kehittäjien omille koneille piti pystyttää kehitysympäristöt TestCafeelle. TestCafen dokumentaatiossa lukee, että asennus tapahtuisi minuuteissa ja käytännössä prosessi oli lähes niin nopea kuin dokumentaatiossa mainittiin. Ennen varsinaisen työkalun asennusta käyttäjän tarvitsi ladata ainoastaan Node.js sekä siihen liittyvä paketinhallintasovellus Npm. Kun paketinhallintasovellus oli asennettu, niin TestCafeen asennus omalle koneelle kävi yhdellä komennolla: npm install -g Testcafe. Muita asennuksia ei tämän jälkeen tarvittu. TestCafeesta käytettiin projektin alkaessa versiota 0.23.2. Käyttöönoton aikana versiota päivitettiin uudempaan aina sitä mukaa kun TestCafe julkaisi uusia versioita. Uudet versiot sisälsivät sekä uusia ominaisuuksia, että bugikorjauksia.

TestCafen dokumentaatio on varsin hyvä. Tämän lisäksi TestCafella on testisivu (kuvio 7) ja käytännön harjoittelu alkoikin ensin sillä, että teimme erilaisia harjoitustestejä testisivustoa vasten, joka on tarkoitettu nimenomaan harjoittelua ja erilaisia kokeiluja varten. Testiympäristöä vasten tehdyillä testeillä päästiin hyvin alkuun ja sain itsekin käytännön tuntumaa siitä, miten automaatiotestejä kirjoitetaan ja ajetaan. Testi sivua vasten tehdyt harjoitukset sisälsivät paljon erilaisiin tekstikenttiin kirjoittamista, lomakkeen täyttämistä, valintanappien, valintalaatikkojen ja raahaus toiminnon kokeilua eli tärkeimpiä ja yleisimpiä käyttäjän toimintoja. Harjoittelujakson aikana varmistettiin myös se, että TestCafe soveltuisi palveluidemme testaukseen suunnitellussa laajuudessaan. Nämä testit kirjoitettiin luonnollisesti omia palveluitamme vasten.



Kuvio 7: Esimerkki TestCafen testi sivusta

5.3 Käyttöönoton aloitus Edilex-palveluun

Ensimmäisen kahden viikon sprintin jälkeen pidettiin katsaus edellisen sprintin tapahtuneista asioista. Katsauksen tarkoituksena oli jakaa kokemuksia ensimmäisen sprintin aikana opituista asioista, huomatuista ongelmista ja siitä kuinka hyvin uskoimme tekemiemme huomioiden perusteella suoriutuvamme seuraavasta sprintistä.

Seuraavassa sprintissä olimme sopineet, että tutkisimme soveltuvatko vanhat Robot Framework -automaatiotestit käännettäväksi TestCafe-skripteiksi. Robot Frameworkilla tehtyjä testejä oli kuitenkin jonkin verran, joten näitä haluttiin edelleen hyödyntää jossain muodossa. Harjoittelu ja tutustuminen sujui hyvin, joten Robot Framework testien käännös päätettiin toteuttaa. Käännöksien avulla saisimme koko testiautomaation yhtenäiseksi ja samalla työkalulla toteutetuksi. Robot Framework käännöstyön rinnalle päätettiin ottaa myös ihan uusien testitapausten automatisointi, jonka osalta piti päättää mitä palveluamme vasten alkaisimme ensimmäiseksi testejä toteuttamaan. Valikoiduin uusien tapausten tekijäksi ja päätimme, että automaatiota lähdetään ensin toteuttamaan Edilex-palveluumme.

Ensimmäiset uudet käyttötapaukset sisälsivät mm. erilaisilla käyttäjärooleilla tehtyjä kirjautumisia palveluun, joiden tarkoituksena oli vahvistaa käyttäjäroolien toimivan suunnitellusti ja sisältävän oikeanlaisen tarjonnan. Edilex-palvelussa on monia erilaisia käyttäjärooleja, jotka määräytyvät sen mukaan mitkä osiot asiakas on itselleen ostanut. Lisäksi ensimmäiset testit sisälsivät paljon erilaisia hakutoimintoja ja palvelun perusrakenteeseen keskittyviä testejä, joissa etsitään tietynlaisia elementtejä sivuilta, testataan siirtymiä yms. perustoimintoja. Nämä testitapaukset olivat vielä hieman alkeellisia, koska varsinaista suunnitelmaa ei vielä uusien testien osalta ollut. Tässä vaiheessa en itse osannut vielä suunnitella testejä siten, että järkevällä määrällä testejä päästäisiin kattavuuteen, jolla varmistetaan palvelun toimintojen olevan halutulla tasolla. Sain kuitenkin prosessin aikana vankan käsityksen siitä, miten työkalulla voidaan kirjoittaa tehokkaasti uusia tapauksia niin että työkalu toimii hyvin omia palveluitamme vasten. Nämä ensimmäisen sprintin uudet testit olivatkin vielä pääasiassa ”harjoitustestejä”, jotka päätettiin tallentaa versionhallintaan referenssitapauksiksi, joista voisi katsoa mallia, ja joita voisi mahdollisesti myöhemmin sitten hyödyntää esimerkiksi uusien, hieman kattavampien testien runkoina.

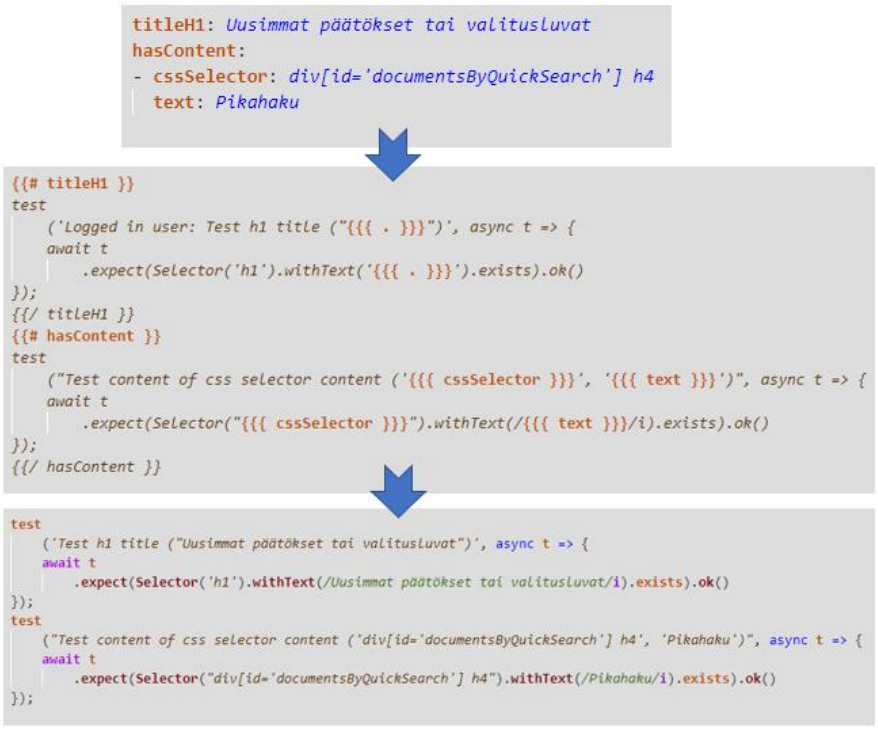
5.4 Vanhojen testien konvertointi TestCafelle

Vanhoja Robot Framework testejä päätettiin hyväksikäyttää lopulta vain osittain. Joukossa oli sekä käsin skriptattuja .robot-tiedostoja, että PHP-skriptin avulla generoituja XML-muotoisia testejä. Robot-tiedostojen kääntäminen päätettiin hylätä, koska tapaukset olisi joka tapauksessa pitänyt kirjoittaa täysin uusiksi. XML-muotoiset testit saatiin puolestaan konvertoitua helpohkosti yaml-muotoon (kuvio 8), josta ne pystyttiin edelleen generoimaan TestCafen ym-

märtämäksi skriptiksi. Yaml-konversio nähtiin hyödyllisenä siksikin, että testit olisivat tarvittaessa helpompi konvertoida yaml-muodosta muihinkin muotoihin kuten JSON-muotoon, mikäli sille tulevaisuudessa tulee tarvetta. Menettelyllä varmistettiin myös se, että mikäli TestCafe osoittautuu joiltain osin puutteelliseksi tai ei ole muutoin tarpeeksi hyvä tarpeisiimme, niin voimme helposti palata joko takaisin Robot Frameworkiin tai kokeilla jotain muuta testiautomaatiotyökalua. Isoin syy yaml-formaattiin oli kuitenkin se, että sitä on paljon nopeampaa kirjoittaa ja yaml on luettavuudeltaan huomattavasti selkeämpää kuin xml. XML ei olisi sellaisenaan soveltunut TestCafessa käytettäväksi toisin kuin yaml.

Kuvio 8: XML-formaatista YAML-formaattiin

Aiemman Robot Framework testiautomaation aikana oltiinkin tehty varsin kattava XML muotoinen testipatteri, jossa XML-tiedostoista oli PHP:ta apuna käyttäen generoitu Robot Framework-testiskriptiä. Tapa oltiin koettu hyväksi ja samaa keinoa haluttiin käyttää mahdollisuuksien mukaan myös TestCafen kanssa, niiltä osin kuin mahdollista. Testien generoinnissa käytettiin Mustache-mallia ja PHP-skriptiä, joiden avulla yaml-formaatissa olevista tiedoista saatiin TestCafeella ajettavaa testiskriptiä (kuvio 9). Vanhoja testejä saatiin näin toimimalla hyödynnettyä varsin kattavasti, eikä kaikkea tarvinnut kirjoittaa käyttöönoton aikana uudestaan.



Kuvio 9: Generoitavien testien matka yaml-formaatista TestCafe testiskriptiksi

Generoitavissa testeissä käytettiin TestCafen assertion metodeita. Vaativampia tai kokonaisvaltaisempia action tai end-to end -testejä ei lähdetty tekemään generoimalla, eikä menetelmä sellaiseen oikein soveltuisikaan. Yksinkertaisen perus- ja regressiopatteriston luomiseen menetelmä soveltui kuitenkin varsin hyvin ja käännöksen jälkeen olimme päässeet käyttöönottoprojektissamme varsin hyvin alkuun. Generoidut TestCafe-testit otettiin aluksi käyttöön Robot Framework-testien rinnalle. Kun kaikki vanhat testit oli lopulta saatu käännettyä TestCafe-skriptiksi ja niiden ajamisen katsottiin toimivan oletetusti, niin Robot Framework ympäristö voitiin ajaa alas.

5.5 Uusien testien implementointivaihe

Uusia testejä lähdettiin tekemään ensimmäiseksi Edilex-palveluun. Palvelu valittiin ensimmäiseksi käyttöönoton kohteeksi, koska kyseiseen palveluun oli parhaillaan myös kehitysprojekti käynnissä, jossa palveluun tuotettiin uusia ominaisuuksia. Regressiotestejä tarvittiin varmistamaan, että uudet ominaisuudet toimivat suunnitellusti, ja ettei niiden implementoinnin aikana mitään vanhaa hajoa. Toinen pääsyy Edilexin valintaan oli, että palvelu on yksittäisenä kohteena suuritöisin automatisoitavaksi. Mitä nopeammin työskentely sen parissa aloitettaisiin, niin sen kattavammaksi testipatteristo saataisiin ennen kehitysprojektin loppumista.

Kävin tiimissämme olevan testaajan kanssa läpi, minkälaisia testejä Edilexiin kirjoitettaisiin, ja kuinka kattavasti testejä pitäisi kirjoittaa. Testaajan kokemusta tarvittiin käyttöönottopro-

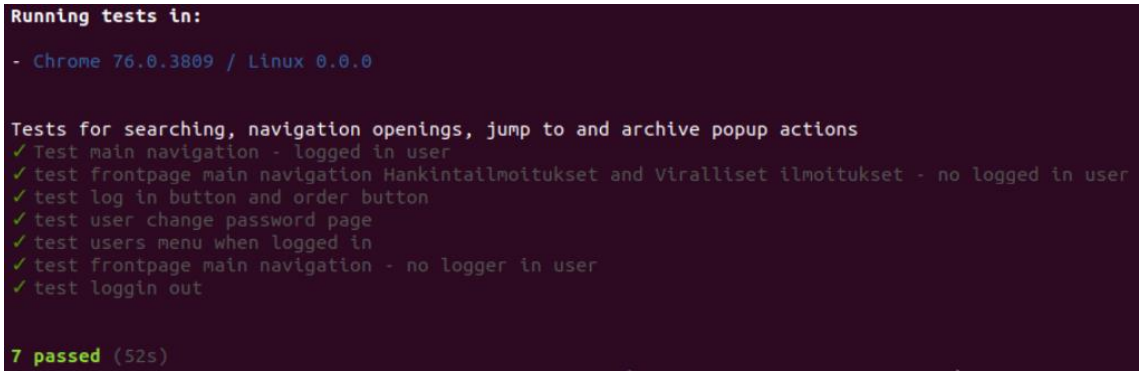
jektin aikana paljon, koska muilla tiimin jäsenillä ei ollut juurikaan kokemusta itse testaamisesta, tai siitä mitkä olivat kaikista kriittisimpiä ja tärkeimpiä osioita Edilexissä ja miten testit kannattaa suunnitella riittävän kattavuuden varmistamiseksi. Implementoinnin alussa oli tarkoitus saada lisättyä paljon yksinkertaisia regressiotestejä, joita lähdettiin myös ajamaan heti sitä mukaa kuin niitä valmistui.

Yksinkertaisten perus regressiotestien jälkeen implementointivaiheessa siirryttiin kirjoittamaan monipuolisempia testejä, jotka olivat jo pidempiä ja sisälsivät selkeämmin jonkun tietyn toiminnallisuuden varmistamisen. Testit sisälsivät mm. ohjelmiston sisältämien tilauslomakkeiden täyttöä, sähköpostien lähetystä, erilaisten hakutoimintojen toimintoja jne. Näiden testien skriptaamiseen käytimme TestCafen action-metodeita, ehto-metodeita, page-model-tekniikkaa sekä muutamaakin erilaista node.js kirjastoa, joilla TestCafen toimintoja voidaan laajentaa.

Testit pyrittiin skriptausteknisesti kirjoittamaan siten, että ne olisivat helppoja ylläpitää ja myös mahdollisimman pitkälle uudelleenkäytettäviä. Näin kaikkia testejä ei tarvitsisi kirjoittaa jokaiselle palvelulle erikseen. Tämä pienentäisi paitsi käyttöönottovaiheen, mutta myös myöhempää ylläpidonaikaista työmäärää. Uudelleenkäytettävyyteen pyrittiin esimerkiksi käyttämällä page model -tekniikkaa. Tekniikan avulla voidaan luoda luokka, joka sisältää erilaisia uudelleenkäytettäviä moduuleita ja funktioita, joita voidaan jatkossa käyttää mistä tahansa testipatteristosta, johon kyseinen luokka on tuotu. Erilaisia asynkronisia apu funktioita käytettiin mahdollisimman paljon, jolloin testien kaikkia steppejä ei tarvinnut kirjoittaa kokonaan uudelleen. Asynkroniset apu funktiot kirjoitettiin page-modelin sisään ja kutsuttiin funktioiden tapaan testeissä, jolloin testit saattoivat sisältää ainoastaan funktio kutsuja, jotka sisälsivät tarvittavat stepit testiin (kuvio 10). Kovakoodatun datan määrä pyrittiin myös minimoimaan, ja mikäli muuttujia oli mahdollista käyttää, niin niitä myös käytettiin. Käytännössä kovakoodattuja tietoja jouduttiin kuitenkin käyttämään melko paljon, sillä testattavat palvelut sisälsivät paljon sellaista uniikkia tietoa, jotka poikkeavat jokaisessa yrityksen palvelussa toisistaan itse toiminnallisuuksien ollessa muutoin samanlaisia.

Kuvio 10: Yksinkertainen esimerkki testistä, jossa testi kootaan erilaisia funktioita käyttäen

Pidempiä testejä jouduttiin ajelemaan lokaalisti yleensä useampia kertoja ennen kuin niiden toimivuudesta ja vakaudesta voitiin olla riittävän varmoja. Pitkien ketjujen implementointivaihe onkin käyttöönoton hitain vaihe. Epävakaita, tai muutoin heikosti toimineita testejä korjailtiin sitä mukaa, kun puutteita tuli esiin. Tavoitteena oli, ettei virheellisiä ja epävakaita testejä menisi versionhallintaan asti. TestCafe-testejä voidaan suorittaa terminaalin kautta ja se antaa välittömästi ajon aikana palautteen siitä onko testi mennyt läpi (kuvio 11) vai onko testi epäonnistunut. Testin epäonnistuessa konsolissa näkyy selkeä ilmoitus siitä mihin testi on kaatunut. Tämä auttoi uusia testejä kehitettäessä ja kirjoituksen jälkeen myös niiden vakauttamisessa.



Kuvio 11: Terminaaliin tulostuva viesti, testien mennessä läpi

Kuviossa 12 näkyy esimerkki epäonnistuneen testin palautteesta. Palautteessa kerrotaan mihin testi on epäonnistunut ja mille riville testi on pysähtynyt. Ko. testi on kaatunut assertion erroriin, eli testissä haettavaa elementtiä ei löytynyt.

Kuvio 12: Terminaaliin tulostuva virhe ilmoitus epäonnistuneesta testistä

Vaikka testejä ajettiin testien kehittämisen yhteydessä, niin silti versionhallintaankin testejä jouduttiin korjaamaan aluksi paljon ja päivittäin. Testien kehityksen yhteydessä testejä ajettiin kehittäjän omassa ympäristössä ja välillä kiireessä vain yhdellä selaimella, jolloin testit menivät läpi. Palvelimella samat testit saattoivat kaatua toisella selaimella. Ympäristön kulloinenkin suorituskyky oli selainten eroavaisuuksien lisäksi toinen merkittävä tekijä, joka vaikutti ajojen vakauteen. Jos ympäristössä oli hitautta, niin testit olivat epävakaampia kuin normaalitilanteessa. Projektin alkuvaiheessa ei täysin ymmärretty, kuinka tärkeää jo kehitysvaiheessa on käydä skenaariot läpi kaikilla mahdollisilla selaimilla, joita palvelu tukee. Näin toimimalla testit olisi saatu ensimmäisellä yrittämällä toimivaksi ja aikaa vievä korjaaminen ja virheiden jäljittäminen olisi jäänyt vähemmälle. Käytännössä testien kehittämisen aikana iteraatio meni siten, että päivisin kirjoitettiin uusia testejä, jotka vietiin valmistuttuaan versionhallintaan. Versionhallinnassa olleet testit ajettiin öisin automaattisesti ja seuraavana aamulla tarkistimme testiraportit, jonka perusteella kaatuneet testit tarkastettiin ja korjattiin.

TestCafen yhtenä heikkoutena voi pitää raportointia. Kattavaa raporttia ei saa työkalusta suoraan oletuksena, vaan sitä varten pitää ladata erillinen lisäosa. Projektissa käytimme TestCafelle tarkoitettua Node.js lisäosaa nimeltään testcafe-reporter-html. Pluginin avulla saimme html-muotoisen selkeän raportin siitä, mitkä testit olivat menneet läpi ja mitkä olivat kaatuneet (Kuvio 13). Raportin lopussa oli myös kaatuneiden testien virheraportit, joista näki tarkan kohdan, jossa mikäkin testi oli kaatunut. Yöajojen osalta ohjasimme raportin tiimimme

Slack-kanavalle erillisen testcafe-reporter-slack pluginin avulla (kuvio 14). Näin saimme helposti käsiimme testiraportit, joista näimme lähes reaaliajassa, olivatko testit menneet läpi vai oliko joitain testejä kaatunut.

TestCafe Test Summary

Summary

Start Time: Thu Oct 31 2019 19:53:44 GMT+0000 (Coordinated Universal Time)

Browsers: IE 11.0.0 / Windows 10.0.0 (https://automate.browserstack.com/builds /4bb620ab463111eef8216728c52e135d9a602075/sessions/adf9e7cf7fdd1c9d8df4d6ef1ab704d2d0d54c82)

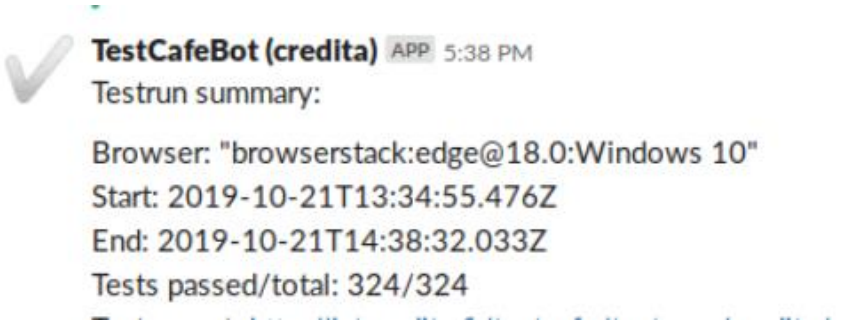
Duration: 1h 24m 25s

Tests Failed: 4 out of 921

Tests Skipped: 0

#	Fixture	Test Name	Browsers	Duration	Result
1	CaseLaw frontpage (/oikeuskaytanto)	Test HTML page title	IE 11.0.0 / Windows 10.0.0 (https://automate.browserstack.com/builds /4bb620ab463111eef8216728c52e135d9a602075 /sessions /adf9e7cf7fdd1c9d8df4d6ef1ab704d2d0d54c82)	15s	passed
			/sessions /adf9e7cf7fdd1c9d8df4d6ef1ab704d2d0d54c82)		
134	Guest user: Courts of Appeal, document (/ho/helho2017107665)	Test that page (not frontpage) contains authentication (/ho/helho2017107665)	IE 11.0.0 / Windows 10.0.0 (https://automate.browserstack.com/builds /4bb620ab463111eef8216728c52e135d9a602075 /sessions /adf9e7cf7fdd1c9d8df4d6ef1ab704d2d0d54c82)	3s	passed
135	Logged in user: Courts of Appeal, document (/ho/helho2017107665)	Logged in user: Test HTML page title	IE 11.0.0 / Windows 10.0.0 (https://automate.browserstack.com/builds /4bb620ab463111eef8216728c52e135d9a602075 /sessions /adf9e7cf7fdd1c9d8df4d6ef1ab704d2d0d54c82)	19s	failed
136	Logged in user: Courts of Appeal, document (/ho/helho2017107665)	Logged in user: Test h1 title ("Helsingin HO 24.2.2017 107665")	IE 11.0.0 / Windows 10.0.0 (https://automate.browserstack.com/builds /4bb620ab463111eef8216728c52e135d9a602075 /sessions /adf9e7cf7fdd1c9d8df4d6ef1ab704d2d0d54c82)	9s	passed
137	Logged in user: Courts of Appeal, document	Logged in user: Test canonical url ('/ho/helho2017107665',)	IE 11.0.0 / Windows 10.0.0 (https://automate.browserstack.com/builds	9s	passed

Kuvio 13: Yöajoista saatu html-raportti



Kuvio 14: Slackiin tulostuva ilmoitus testiajosta

Testien implementointi jatkui samanlaisena prosessina koko käyttöönoton aja, ja jatkuu edelleen sen jälkeen. Uusia testejä kirjoitettiin muihinkin palveluihin sitä mukaa, kun kehitettävänä oleva projekti vaihtui. Uusien testien kirjoittaminen ei jatkuvasti kehittyvässä ympäristössä lopu koskaan.

5.6 Yhteensopivuusongelmien ratkaiseminen

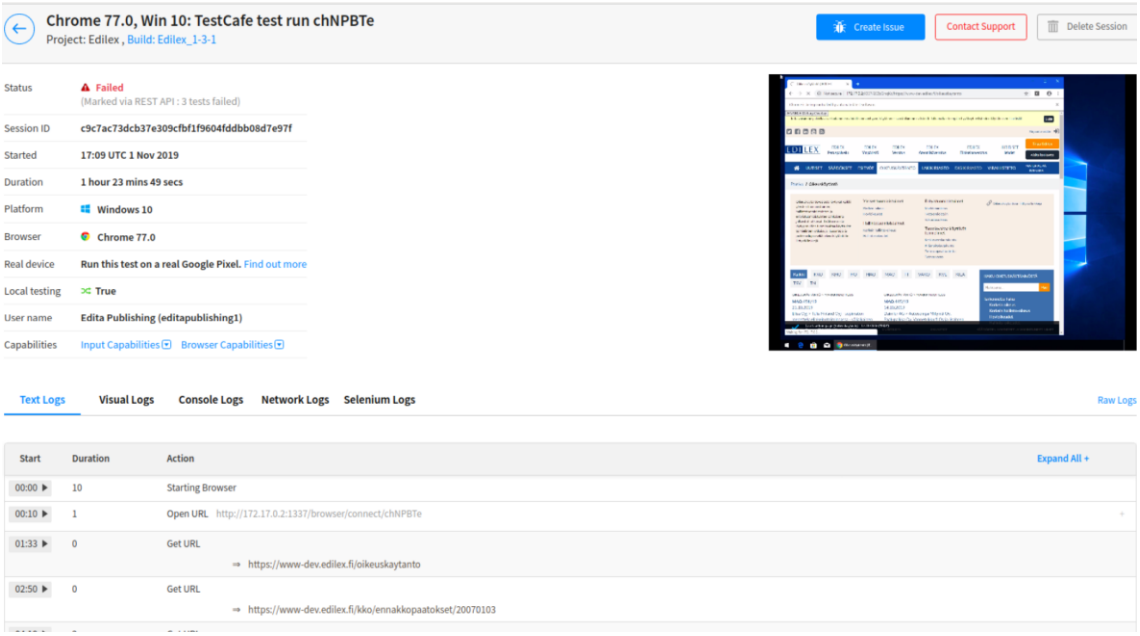
TestCafella on lähtökohtaisesti erinomainen tuki erilaisille selaimille. Lokaalikehityksessä haasteena on kuitenkin se, että erilaisia kokoonpanoja on turhan paljon testattavaksi. Testien virheiden jäljittäminen suurella määrällä erilaisia kokoonpanoja vie liian paljon aikaa pois itse automatisoinnista ja testien kehittämisestä. Lisäksi lokaaliympäristöt asettavat määrättyjä rajoitteita. Esimerkiksi Linux koneella, joka useimmilla tiimin jäsenillä oli käytössään, ei voi testata Microsoftin selaimia.

Erilaisten kokoonpanovariaatioiden testaamisen järkevöittämiseksi lähdimme tutkimaan minkälaisia pilvitestauspalveluita olisi mahdollisuus käyttää. TestCafesta löytyi integraatiot BrowserStack- ja SauceLabs -palveluihin, ja nämä tarjosivatkin suoraan vastauksen ongelmiimme, joten muita ei tarvinnut edes lähteä seulomaan. Kahdesta tarjolla olleesta palvelusta valitsimme Browserstackin, jotka käytimme myös palvelinkoneella ajettavissa yöajoissa. Browserstack on palvelu, jossa on käytännössä tarjolla kaikki mahdolliset selaimet sekä työpöytäkoneille että mobiililaitteille. Palvelusta voi valita minkä tahansa haluamansa laitekokoonpanon, jota vasten testiautomaatiota halutaan ajaa ja Browserstack näyttää valinnan perusteella kyseiselle kokoonpanolle konfiguraatioasetukset, joiden avulla automaatiotyökalu osaa käyttää tätä testissään. Testejä ajetaan Browserstackin kautta aivan kuten lokaalistikin, mutta selain, jossa ne suoritetaan, on nyt käyttäjän oman koneen sijaan pilvessä.

Browserstackin avulla pystyimme kokeilemaan helposti myös muita kuin suoraan tuettuja selain- ja käyttöjärjestelmä kokoonpanoja. Browserstackilla on hyvänä ominaisuutena myös se, että se nauhoittaa jokaisesta testiajoista videon. Videolta pystyi helposti katsomaan, mihin jokin testi oli kaatunut ja mitä selaimessa tapahtui, joka tuohon kaatumisen johti. Browserstackissa on rajoite, joka mahdollistaa ainoastaan kahden tunnin yhtäjaksoisen testiajon. Kahden tunnin testiajo osoittautui liian lyhyeksi ajaksi testimäärän kasvaessa. Tämän ongelman saimme ratkaistua siten, että pilkoimme eri projektien testejä pienempiin yksittäisiin eriin, jotta jokainen yksittäinen testipino mahtuu kahteen tuntiin, eikä ajo keskeytyisi yöllä maksimikeston ylitykseen.

Kuviossa 15 on Browserstackiin tallentunut testiajo sekä tiedot siitä. Browserstackissa näkee, kuinka kauan ajo on kestänyt, millä selaimella ja käyttöjärjestelmällä ajo on tapahtunut sekä eri testi stepit, josta pystyy valita mihin kohtaan nauhoitettua videota testaaja haluaa hypätä tai minkä kohdan testistä nähdä. Kuvan oikealla puolella näkyy itse testiajosta nauhoitettu

video. Browserstackin huonoja puolia on se Browserstackista ei näe mihin testit ovat epäonnistuneet ilman videon katsomista. Raportista näkee vain, montako testiä on epäonnistunut.



Kuvio 15: Browserstack raportti

Kokoonpanojen määrään liittyvien haasteiden lisäksi ongelma oli eri selainten hieman toisistaan poikkeava tapa toimia ja käsitellä esim. JavaScriptiä, jolla TestCafekin toimii. Jotkut vanhemmat selaimet, kuten esimerkiksi Internet Explorer, eivät tue kaikkia JavaScript-ominaisuuksia tai valmiita metodeita, jolloin testit, jotka toimivat uudemmilla selaimilla, saattoivat olla käyttökelvottomia vanhoilla. Lisäksi, vaikka joku asia olisi tuettukin, niin pieniä käsittelyeroja selaimessa suoritettavan JavaScriptin suhteen on ja nämä aiheuttavat yllättävän paljon ongelmia testejä kehitettäessä sellaisissakin asioissa, joissa niitä ei osata odottaa. On esimerkiksi väliä, kirjoitetaanko testiin sanat isoilla vai pienillä kirjaimilla. Jotkut selaimet osaavat lukea tekstin kuten käyttäjäkin sen näkee, ja toiset eivät osaa huomioida CSS:ää, vaan näkevät ainoastaan sen muodon missä asia on esitetty DOM-sisällössä. Kaikkiin ongelmiin kuitenkin yleisesti ottaen löytyi ratkaisu, esimerkiksi kirjainkoko ongelmiin löysimme ratkaisun regexistä. Regular expression (Regex) tarkoittaa säännöllistä lauseketta, jonka avulla voidaan määritellä vastaako jokin merkkijono toista merkkijonoa vai ei. Tässä tapauksessa määrittelimme regexiä käyttäen, että joissain merkkijonoissa ei ollut väliä kirjoitetaanko merkkijono isolla kirjaimilla vai ei. Näin saimme testit toimimaan kaikilla tarvittavilla selaimilla.

Elementtien löytämisen suhteen oli välillä paljon samantyyppisiä haasteita, jotka usein liittyivät erilaisiin näkyvyysongelmiin. Joillain selaimilla oli tärkeää, että elementti, jota etsitään, on kokonaan näkyvissä, ja joillain toisella ei ole mitään välillä missä kohdassa dokumenttia

elementtiä sijaitsee ja onko tämä sillä hetkellä näkyvissä vai ei. Sama pätee selainten selainikkunoihin, jotka käyttäytyivät yksilöllisesti. Joku aukeaa automaattisesti johonkin omaan oletus kokoonsa, toinen täyteen ruutuun, kolmas pitää parametrisoida ja neljäs on jokin yhdistelmä näistä muista. Edellä mainitut selainten eroavaisuudet tekivät joistain testeistä todella haastavia automatisoitavia. Erityisesti responsiivisilla sivuilla koko näkymä on usein suoraan sidoksissa käyttäjän ruudun kokoon.

5.7 Testiajojen ajastaminen, käytettävät kookoonpanot ja suoritussykli

Kuten edellä olevissa kappaleissa jo sivuttiin, niin yöajot tapahtuivat palvelimen kautta. Itse ajastus tapahtui cronin avulla. Cron on ajastuspalvelu Unix-käyttöjärjestelmille, jolla pystytään ajastamaan ja automatisoimaan erilaisia rutiineja, kuten latauksia, käännösprosesseja, sähköpostien lähetyksiä ja vaikka tarpeettomien tiedostojen poistamista. Ajastuksia hallitaan cron-jobin avulla. Esimerkiksi testiajojen ajamista varten cron-jobille ajastetaan haluttu ajankohta, jolloin toiminto suoritetaan, sekä polku missä sijaitsee skripti, joka halutaan ajaa (kuvio 16).

```
# TestCafe automation tests, run at 05:00 pm every weekday
0 17 * * 1-5 sh /home/tests/runBrowserStackTests.sh
```

Kuvio 16: Cron ajon määrittely

Aluksi yöajoissa pyöriteltiin joka arkiyö niiden kaikkien viiden eri palvelun testejä, joihin oli tehty jo alustavat testit. Ajastimme ajon siten, että prosessi käynnistyi jokaisena arki-iltana klo 17. Skriptille annettiin tiedot siitä mistä hakemistosta testit haetaan, minkä projektin tai palvelun testit ajetaan sekä se, millä selaimilla testit ajetaan. Testit ajettiin joka yö eri selaimella tai käyttöjärjestelmällä (kuvio 17). Käytettäviksi Selaimiksi valikoitui Edge, Internet Explorer +11, Google Chrome, Mozilla Firefox ja Safari. Kaikilta selaimilta valittiin uusimmat vakaat versiot. Browserstackissa ajettavien testien käyttöjärjestelmiksi valittiin Windows 10 ja IOS (Safari).

```
BROWSER='browserstack:chrome@77.0:Windows 10'
DAY=`date +"%a"`
if [ "${DAY}" = "Mon" ]
then
    BROWSER='browserstack:edge@18.0:Windows 10'
elif [ "${DAY}" = "Tue" ]
then
    BROWSER='browserstack:safari@12.1:OS X Mojave'
elif [ "${DAY}" = "Wed" ]
then
    BROWSER='browserstack:firefox@69.0:Windows 10'
```

Kuvio 17: Selainten määrittely eri päiville

Kun testikattavuutta oli saatu kasvatettua ja testejä oli jo paljon jokaisessa projektissa, ei ollut enää järkevää ajaa kaikkien palveluiden testejä joka yö, koska testien kokonaiskesto oli jo niin pitkä, etteivät kaikki ehtisi yön aikana edes valmiiksi. Yöajot olivat pääosin regressiotestejä, joilla varmistetaan, ettei mikään ole mennyt rikki kehityksen yhteydessä. Regressiotestien lisäksi yöajoissa pyöri myös paljon muita sellaisia testejä, joiden osalta katsoimme, ettei niitä tarvitse ajaa joka yö, varsinkaan silloin kuin ko. palveluun ei tehty mitään kovin isoja muutoksia. Yöajoista poistettuja testejä ajoimme tämän jälkeen kerran tai kaksi kertaa sprintin aikana ja yleensä päivisin. Tämä tapahtui yleensä silloin, kun olimme valmistelemassa jonkun palvelun julkaisua. Ennen julkaisua tehtiin niin sanottu releasetestaus, jossa ajettiin kaikki testit monilla erilaisilla kokoonpanoilla. Julkaisun yhteydessä tehtiin myös manuaalista testausta testiautomaation lisäksi.

Käyttöönoton loppuvaiheessa fokusoimmekin yöajot ainoastaan kehitettävänä olevien palveluiden testejä varten, koska näihin oli odotettavissa jatkuvia muutoksia. Samalla lisäsimme myös näiden palveluiden testikattavuutta. Näin pystyimme keskittymään yhden tai kahden eri palvelun kehittämiseen ja testaamiseen kerrallaan sekä saimme tehtyä kattavammin testejä ko. palveluille.

6 Yhteenveto

Käyttöönottoprojektin tavoitteena oli ottaa käyttöön uusi testiautomaatiotyökalu, jolla automatisoimme jatkossa kaiken, tai ainakin lähes kaiken palveluidemme sellaisista testaustarpeista, joihin testiautomaatio soveltuu. Toisena isona tavoitteena oli lisätä testikattavuutta selvästi isommaksi uusia testitapauksia lisäämällä ja samalla vähentää manuaalitestauksen määrää merkittävästi automatisoimalla kaikki manuaalitestaustapaukset, jotka pystytiin järkevällä tavalla automatisoimaan. Ajatuksena oli myös tulla jatkossa toimeen yhdellä testiautomaatioympäristöllä ja työkalulla, joka olisi yrityksen pääasiassa JavaScriptiä ja PHP:ta kirjoittaville ohjelmistokehittäjille mahdollisimman nopea omaksua ja jonka avulla saavuttaisimme joustavamman testiautomaation kehittämisen ja ylläpidon, muiden kehitysprojektien lomassa.

Kokonaisuudessaan tämäntyyppinen käyttöönottoprojekti on ajallisesta mitattuna todella iso. Tämän työn sisältämän noin puolen vuoden pituisen ajanjakson aikana ympäristöt tulivat valmiiksi ja automatisoituja testejä oltiin siinä ajassa implementoitu noin viisi tuhatta. TestCafella toteutettu testiautomaatio pyörii jokaisessa Edita Publishingin palvelussa ja testien määrä kasvaa edelleen päivittäin.

Käyttöönottoprojektia voidaan kokonaisuudessaan arvioida tulokseltaan onnistuneeksi. Projektin aikana ei noussut esiin asioita, jotka olisivat kyseenalaistaneet valitut työkalut, menetelmät tai ympäristöt, joka varmistaa tutkimuksen olevan validi ja reliaabeli. Kokonaistestien määrä on kasvanut todella paljon ja manuaalitestausta tekee enää yksi henkilö osa-aikaisesti testien automatisoinnin ohessa. Aluksi rinnalla ollut Robot Framework ympäristö saatiin korvattua kokonaisuudessaan TestCafella. Uusi työkalu oli odotusten mukaan erittäin nopeasti omaksuttava, ja se on mahdollistanut sen, että automatisoituja testitapauksia on saatu tuotua palveluihin erittäin nopealla tahdilla. Saavutettuihin tuloksiin viitaten voidaan todeta, että projektin aikana tutkittiin oikeita asioita ja oikeita ratkaisuja tehtiin työkalun ja käytettyjen ympäristöjen suhteen, joten projekti oli validiteetiltaan ja reliabiliteetiltaan pätevä.

Käyttöönoton aikana ilmenneet haasteet saatiin selvitettyä. TestCafella on joitain heikkouksia verrattuna esim. Robot Frameworkiin, joista ehkä merkittävämpänä selvästi pienempi kirjastovalikoima ja esimerkiksi muun kuin HTML-pohjaisten sivujen testaaminen on erittäin hankalaa, ellei jopa mahdotonta. Yrityksen palvelut joihin testiautomaatiotarve kohdistuu, pyörivät kuitenkin käytännössä täysin web-ympäristöissä, joten kirjastopuutteiden vaikutukset jäivät varsin pieniksi, ja joissain tapauksissa ne päästiin kiertämään kokonaan mm. oman kehitystyön avulla.

Käyttöönoton aikana opittiin paljon yrityksen palveluiden testaustarpeesta ja testiautomaation pystytyksestä sekä käyttöönotosta. Käyttöönottoprojektilla onnistuttiin saamaan välittömiä hyötyjä niin testiautomaatioon liittyen sekä kokemusta tulevaisuuden vastaavanlaisiin projekteihin.

7 Oman oppimisen arviointi

Opinnäytetyöni aihe valikoitui pitkälti oman kiinnostukseni perusteella. Halusin ottaa aiheeksi jonkun projektin, jossa olin mukana työharjoittelujaksoni aikana, ja josta minulla ei olisi ennen työn kirjoittamista ollut aiempaa kokemusta. Uskoin lähestymistavan tukevan omaa oppimistani aiheesta ja vahvistaa rooliani projektissa, koska joutuisin miettimään asiaa hieman syvällisemmin kuin pelkän käytännön suorittamisen kannalta.

Minulla ei ollut ennen opinnäytetyötäni aiempaa kokemusta minkäänlaisesta automaatiotestauksesta tai järjestelmien käyttöönotosta, joten opinnäytetyöni aihe opettaisi minulle to-

della paljon. Tulisin sen aikana törmäämään moniin haasteisiin ja joutuisin hankkimaan taitoja ja tietoja aiheista, josta en etukäteen tiennyt mitään. Olin jo ennen opinnäytetyöni aloittamista ollut erittäin kiinnostunut testien automatisoinnista, joten aiheen valinta oli helppo, kun työssä kuvattu projekti aloitettiin marraskuussa 2018. Lähestymiskulma opinnäytetyön näkökulmasta oli vaikeampi valinta. Alkuun tarkoituksena oli kuvata testiautomaation kehittämistä ja skriptien kirjoittamisesta, mutta koska halusin ainakin jollain lailla kuvata myös projektissa pääteemana ollutta käyttöönottoa, ja kokonaisuudesta olisi tullut liian laaja, mikäli aiheeseen olisi sisällytetty molemmat, niin päädyin lopulta kertomaan asiasta käyttöönoton näkökulmasta.

Opin todella paljon sekä käyttöönoton, että opinnäytetyöni kirjoitusprosessin aikana. Teknisessä ja projektinhallinnallisessa mielessä sain paljon oppia testiautomaatiosta ja erilaisista testiympäristöistä, sekä kokonaiskuvan siitä mitä kaikkea tarvitsee tehdä ja ottaa huomioon ennen kuin testiautomaatioprojekti on kehitys- ja ylläpitovaiheessa. Aiemmin en ollut kuullutkaan konttiteknologioista, Browserstackista, cron-ajastuksista tai oikeastaan mistään muustakaan työkaluista tai teknologioista mitä tässä työssä on kuvattu. Kirjoitusprosessi tuki hyvin projektin aikana oppimiani asioita, koska jouduin tekstiä tuottaakseni tutustumaan aiheeseen melko syvällisesti ja pilkkomaan kokonaisuuden pieniksi paloiksi, joista se muodostuu.

Automaatioskriptien suunnittelu ja toteutus on itselleni nykyään huomattavasti helpompaa ja pystyn tuottamaan huomattavasti järkevämpää skriptiä kuin projektin alkuvaiheessa. Minulla oli opiskelujen kautta hieman kokemusta entuudestaan manuaalisesta ohjelmistotestauksesta, mutta automaatiotestaaminen on varsinaista testitapausten suunnittelua lukuun ottamatta hyvin erilaista. Erityisesti testitapausten suunnitelmallinen purkaminen niin, että ne voidaan toteuttaa teknisesti mahdollisimman uudelleenkäytettävinä automaatioskripteinä, oli asia, josta manuaalitestauskokemuksella ei voinut tietää mitään.

Olen oppinut paljon myös testiautomaatiotyökaluista ja testiautomaation käyttöönotosta erittäin paljon. Ymmärrän sekä käytännön, että teorian tasolla erilaisten arkkitehtuurien vaikutuksen skriptaustekniikkaan. On syntynyt myös hyvä kuva siitä minkälaisia ongelmia työkalun käyttöönottoprojektin aikana voi tulla vastaan, sekä kuinka niitä voidaan hyvällä suunnittelulla ottaa jo ennalta huomioon, ja toisaalta myös ratkaisumalleja, joilla näitä ongelmia voi selvittää.

Minulla oli entuudestaan opiskeluideni aikana hankittua osaamista JavaScriptistä sekä Dom:in käsittelystä ja manipuloinnista. Nämä taidot auttoivat minua paljon testien implementointivaiheessa, eikä TestCafe itsessään ollut vaikeasti omaksuttava. Koen, että projektin aikana myös JavaScript ja DOM ymmärrykseni ja osaamiseni on vahvistunut merkittävästi. Ohjelmointiin liittyen myös erilaisten kirjastoiden hyödyntäminen ja käyttäminen on tullut tutuksi. Tämäkin oli asia, josta en ennen tätä projektia tiennyt käytännössä mitään. Ihan yleisellä tasolla

voi sanoa, että opinnäytetyöprojektin aikana koko testiautomaatio käsitteenä, sen hyödyt, haitat, miten ja miksi sitä tehdään, ovat konkretisoituneet.

Lähteet

Painetut

Fewster, M. & Graham, D. 1999. Software test automation: Effective use of test execution tools. Harlow: Addison-Wesley. Luettu 18.10.2019

Harju, A. 2004. Projektin ohjaus tietojärjestelmän käyttöönotossa. Helsinki: Helsingin ammattikorkeakoulu Stadia.

Heikkinen, H. L., Rovio, E., Tynjälä, P., Kakkori, L., Huttunen, R., Tiihonen, A., . . . Kiilakoski, T. 2006. Toiminnasta tietoon: Toimintatutkimuksen menetelmät ja lähestymistavat. Helsinki: Kansanvalistusseura.

Pohjalainen, M. 2012. Hiljaisen tiedon käsite ja hiljaisen tiedon tutkimus. Tampere: In-formaatiotutkimuksen yhdistys.

Grönfors, M & Vilkka, H. 2011. Laadullisen tutkimuksen kenttätyömenetelmät. Hämeenlinna: SoFia-Sosiologi-Filosofiapu Vilkka.

Sähköiset

Moskovkin, A. 2017. TestCafe: An e2e Testing Tool That Doesn't Use Selenium. Viitattu 3.11.2019. https://dzone.com/articles/testcafe-e2e-testing-tool

Hiltunen, L. 2019. Validiteetti ja reliabiliteetti. Jyväskylän Yliopisto. Viitattu 3.11.2019. http://www.mit.jyu.fi/OPE/kurssit/Graduryhma/PDFt/validius_ja_reliabiliteetti.pdf

Saaranen-Kauppinen, A & Puusniekka, A. 2006. Validiteetti. KvaliMOTV – Menetelmäopetuksen tietovaranto. Tampere: Yhteiskuntatieteellinen tietoarkisto. Viitattu 3.11.2019 https://www.fsd.uta.fi/menetelmaopetus/kvali/L3_3_1.html

Node.js 2019. Introduction to Node.js. Viitattu 17.11.2019. https://nodejs.dev/

Npm. 2019. Npm – Buid amazing things. Viitattu 10.9.2019. https://www.npmjs.com/

Kettunen, J & Simons, M. 2001. Toiminnanohjausjärjestelmän käyttöönotto pk-yrityksessä. Valtion tekninen tutkimuslaitos. Luettu 6.10.2019. https://www.vtt.fi/inf/pdf/julkaisut/2001/J854.pdf

Aebersold, K. 2019. Test Automation Frameworks. Viitattu 29.9.2019. https://smartbear.com/learn/automated-testing/test-automation-frameworks/

TestCafe 2019. A node.js tool to automate end-to-end web testing. Viitattu 14.9.2019. https://devexpress.github.io/testcafe/

DevExpress 2019. DevExpress / Testcafe-hammerhead. Viitattu 15.9. 2019. https://github.com/DevExpress/testcafe-hammerhead

TestCafe 2019. Browser-Support. Viitattu 14.9.2019. https://devexpress.github.io/testcafe/documentation/using-testcafe/common-concepts/browsers/browser-support.html

TestCafe 2019. Framework specific selectors. Viitattu 15.9.2019. https://devexpress.github.io/testcafe/documentation/test-api/selecting-page-elements/framework-specific-selectors.html

TestCafe 2016. Why not use selenium- how to use TestCafe. Viitattu 15.9.2019. https://testcafe-discuss.devexpress.com/t/why-not-use-selenium-how-to-use-testcafe/47/2

Halonen, R. 2002. Tietojärjestelmän vaihtaminen. Viitattu 18.10.2019. http://www.cse.tkk.fi/fi/tkt-lehti/a20/halonen.pdf

Tanna, H. 2017. Top 10 benefits of test automation. Viitattu 19.10.2019. https://dzone.com/articles/top-10-benefits-of-test-automation

Kuviot