REGRESSIOTESTAUKSEN AUTOMATISOINTI

HAMK
HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

Ammattikorkeakoulututkinnon opinnäytetyö

Hämeenlinnan korkeakoulukeskus
Tietojenkäsittelyn koulutusohjelma

Syksy, 2019

Anu Lindell

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

TIIVISTELMÄ

Tietojenkäsittelyn koulutusohjelma
Hämeenlinnan korkeakoulukeskus

Tekijä Anu Lindell **Vuosi** 2019

Työn nimi Regressiotestauksen automatisointi

Työn ohjaaja/t Lasse Seppänen

TIIVISTELMÄ

Opinnäytetyön tavoitteena oli opetella hyödyntämään Robot Frameworkia testauksen automatisoinnissa. Tavoitteena oli myös saavuttaa työajan säästöä ja helpottaa manuaalista testausta. Opinnäytetyönä toteutettiin 34 automatisoitua regressiotestitapausta. Työ toteutettiin kevään 2019 aikana. Asetetut tavoitteet saavutettiin, sillä luodulla testausautomaatiolla on mahdollista säästää testaajan aikaa rutiinitesteistä, jolloin testaaja pystyy tekemään enemmän tutkivaa testausta ja kokeilla myös erikoisempia variaatiota. Toimeksiantajana toimi LähiTapiola.

Opinnäytetyössä kuvataan testattavan sovelluksen erityispiirteet, testausautomaation toteutus ja sen aikana havaitut haasteet. Lisäksi lopuksi esitellään jatkokehitysehdotukset testausautomaatiolle.

Opinnäytetyön alussa olevassa teoriaosuudessa käydään läpi yleisesti, miksi testausta tehdään, minkälaiset periaatteet ohjaavat testausta, lyhyesti regressiotestaus ja testausautomaatio. Havaintona oli, että teoria ja toteutus vastaavat hyvin toisiaan. Testausautomaation kehittäminen on hidasta, aikaa vievää ja sen ylläpitoon kuluu runsaasti aikaa. Testausautomaatiota voisi parannella ja jatkokehittää loputtomasti.

Avainsanat Testausautomaatio, regressiotestaus, Robot Framework

Sivut 25 sivua, joista liitteitä 3 sivua

HÄMEEN AMMATTIKORKEAKOULU
HÄME UNIVERSITY OF APPLIED SCIENCES

ABSTRACT

Degree Programme in Business Information Technology
Hämeenlinna University Centre

Author Anu Lindell **Year** 2019

Subject Test automation for regression testing

Supervisors Lasse Seppänen

ABSTRACT

The main aim for this thesis was to learn how to utilize Robot Framework in test automation. This thesis project aims also to save working time and simplify the manual testing process. In the progress of this thesis, there were 34 test cases which were automatized with Robot Framework. Test automation was created during spring 2019. The aims set were achieved, this test automation saves testers time from the routine tests and there is now more time to do exploratory testing and try more trivial variations. Commissioner of this thesis was LocalTapiola.

In this thesis, there is a description about the software to be tested, creation of the test automation and observations of the challenges in the test automation creation process. There are suggestions for further development of the created test automation.

The first part of the thesis deals with the theory, there are information about why testing is done, what kind of principles guide the testing processes and shortly about regression testing and test automation. The main observation was that the theory matches up with the implementation. Creating test automation is quite slow and time-consuming process and the maintenance consumes time. There are endless ideas how to develop the test automation even better.

Keywords Test automation, Regression testing, Robot Framework

Pages 25 pages including appendices 3 pages

Käsitteistö

Back end	Palvelinpuoli, koodi ajetaan sivuston palvelimella. Esim. lomakkeiden käsittely, salasanojen tarkistus.
Refaktorointi	Prosessi, jossa ohjelmiston koodia muutetaan siten, että toiminnallisuus säilyy, mutta sen sisäinen rakenne muuttuu. Esim. koodin yksinkertaistaminen.
Xpath	Ei-xml-pohjainen kieli xml-dokumenttien osien paikantamiseen ja luomiseen.
Keyword	Avainsana, funktio. Sisältää toiminnallisuuden.
Test Case	Testitapaus

SISÄLLYS

1 JOHDANTO

Ohjelmistokehitykseen kuuluu olennaisena osana testaus. Testauksen tarkoitus on tuottaa tärkeää informaatiota kehitetystä ohjelmistosta. Testausta voidaan joutua tekemään runsaasti, jos ohjelmistoa kehitetään ketterästi ja päivityksiä tulee usein. Testauksen automatisointi usein toistuvissa samanlaisissa testeissä on usein järkevää. Tällöin voidaan saavuttaa hyötyjä perustestien ajamisessa ja henkilöresurssien kohdistamisessa sellaisiin testeihin, joilla voidaan paikallistaa uusia virheitä.

Tämän opinnäytetyön tavoitteena on opetella testausautomaation luominen selainpohjaiselle sovellukselle, ottaa se käyttöön ja kehittää sitä edelleen käyttöönoton jälkeen. Testattava sovellus valikoitui työpaikalta ja kyseessä oli yksi oman testaus- ja kehitysvastuualueen sovellus. Käytännön kokemus oli osoittanut sovelluksen kuukausittain toistuvan regressiotestauksen työlääksi ja aikaa vieväksi. Lisäksi inhimillisiä virheitä sattuu huomattavan paljon enemmän jatkuvasti toistuvissa testeissä. Oikein rakennetulla testausautomaatiolla saataisiin virheet vähenemään ja testauksen tarkkuutta parannettua.

Käytännön opinnäytetyöprosessi alkoi Robot Frameworkiin tutustumisella, ensin itsenäisesti verkosta löytyneillä materiaaleilla sekä tämän jälkeen osallistumalla Mimmit koodaa -hankkeen järjestämään yhden päivän Robot Framework koulutukseen. Käytännön työhön kului kuitenkin runsaasti aikaa, koska oppimista oli hyvin paljon. Regressiotestauspaketti oli käyttöönottovalmis kolme kuukautta sen aloittamisen jälkeen, jonka jälkeen pieniä korjauksia on tehty matkan varrella.

Kehittämisen kohde on testausautomaation luominen sovellukselle, jotta kriittiset virheet saataisiin jokaisen version hyväksymistestauksessa esiin mahdollisimman aikaisessa vaiheessa ja estettäisiin virheellisen toiminnan siirtyminen tuotantoympäristöön. Testausautomaatiolla pyritään säästämään resursseja ja aikaa jatkuvasti toistuvien testien osalta. Sovellukseen suoraan liittyvät kehitystyöt testataan manuaalisesti ja niiden rinnalla ajetaan testausautomaatiolla tehdyt regressiotestitapaukset.

Opinnäytetyö pyrkii vastaamaan kahteen kysymykseen: Miten paljon testauksen automatisointi säästää aikaa? Millä tavoin testauksen automatisointi helpottaa manuaalista testausta?

2 TESTAUS

Testauksen tarkoitus on tuottaa ohjelmiston tilasta tietoa ja näkökulmia eri sidosryhmille. Tuotetun tiedon on tarkoitus vähentää ohjelmistoon liittyviä epäselvyyksiä. Esimerkiksi päätös ohjelmiston julkaisusta tehdään testauksen tuottaman tiedon perusteella. Päätöksen tueksi tarvitaan tietoa ohjelmiston esimerkiksi käytettävyydestä, vaatimustenmukaisuudesta, tunnistetuista riskeistä. Testauksen on tarkoitus tuottaa objektiivinen näkemys siitä, miten pitkälle ohjelmisto vastaa sille asetettuihin vaatimuksiin. Ohjelmiston suorituskyky, luotettavuus, turvallisuus ja toiminta varmistetaan testauksella. On tärkeää varmistaa, että kehitetty ohjelmisto vastaa käyttäjän tarpeisiin ja että on rakennettu se ohjelmisto, mikä on tilattu. Testaus tukee myös ennen ja jälkeen julkaisun tapahtuvaa riskienhallintaa, pyrkimällä löytämään mahdollisimman paljon virheitä ja vikoja ohjelmistosta jo ennen julkaisua. Testauksen tarkoitus on parantaa ohjelmiston laatua. Kun ohjelmisto julkaistaan laadultaan riittävänä, vähentää se merkittävästi ylläpidon ja myöhempien korjausten kustannuksia. Vikojen ja virheiden löytäminen ei ole testauksen ainoa tarkoitus, vaan sen lisäksi varmistetaan, että ohjelmisto täyttää asetetut vaatimukset. (Ghahrai, 2018a)

Käytännössä ohjelmistot pyörittävät nykypäivänä lähes kaikkea. Jokapäiväistä toimintaa on helpotettu ohjelmistoilla, joita löytyy esimerkiksi lähes kaikista moottoroiduista kulkuneuvoista, puhelimista ja erilaisista asiointipalveluista. Järjestelmiä ja ohjelmistoja testataan, jotta muun muassa vältyttäisiin asiakkaille näkyviltä häiriöiltä. Lisäksi pyritään estämään yritysten negatiivinen näkyvyys häiriöihin liittyen, joilla voi olla yrityksen liiketoiminnalle hyvin haitallisia vaikutuksia. Käytännössä ohjelmistot vaikuttavat lähes kaikkien ihmisten ja yritysten toimintaan nykypäivänä. (Homes, 2013, s. 1; Myers; Badgett & Sandler, 2011, s. 1)

2.1 Testauksen periaatteet

Testaus tai ohjelmistotestaus kuuluu ohjelmistotuotannon kokonaisuuteen. Ohjelmistojen testauksen haastavuus on kasvanut merkittävästi, kun ohjelmistoista on tullut laajempia ja monimutkaisempia kokonaisuuksia. Tämän vuoksi testaus on nykypäivänä varsin suunnitelmallista ja dokumentoitua toimintaa. Testauksella on tarkoitus varmistua siitä, että rakennettu ohjelmisto, komponentti tai laite toimii, ja se on toteutettu, kuten oli tarkoitus. (Kasurinen, 2013, ss. 10-13)

Ohjelmistotestaukselle on asetettu seitsemän ytimekästä periaatetta ISTQB testaussertifikaatissa (ISTQB, 2018, s. 15). Näiden periaatteiden avulla ohjelmistotestaus voidaan perustella ja määritellä. Näitä periaatteita esitellään seuraavissa alakappaleissa tarkemmin.

2.1.1 Testaus osoittaa virheiden läsnäolon

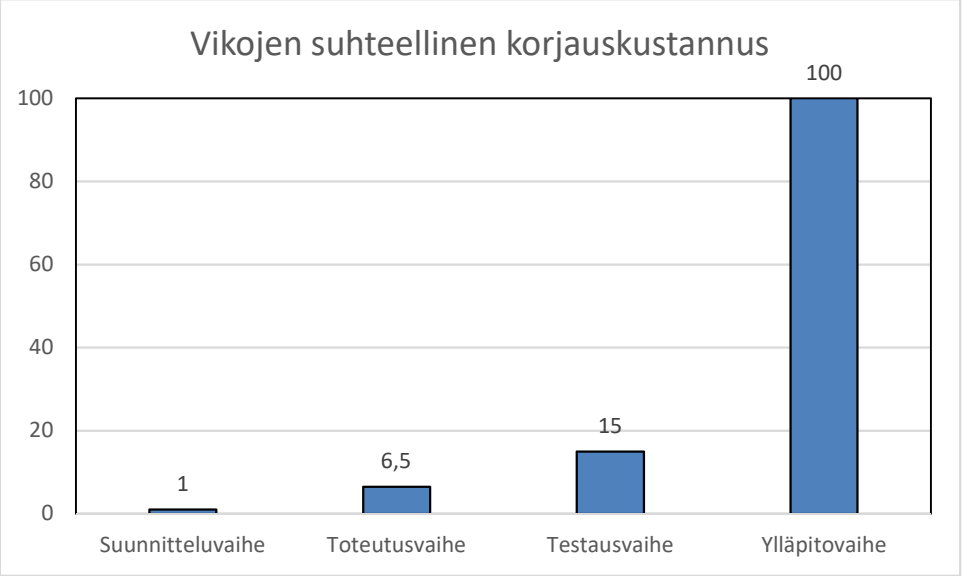
Testaus osoittaa virheiden läsnäolon, mutta ei niiden poissaoloa. Testauksella ei ole mahdollista osoittaa, ettei ohjelmisto sisällä vikoja, vain koska niitä ei ole löydetty. Voidaan vain pienentää sen todennäköisyyttä, että vikoja olisi. Koska testauksella ei voida osoittaa ohjelmiston virheettömyyttä, on testitapaukset syytä suunnitella siten, että niillä voidaan löytää mahdollisimman paljon erilaisia havaintoja ohjelmiston toiminnasta. (ISTQB, 2018, s. 15; Ghahrai, 2018b)

2.1.2 Täydellinen testaus on mahdotonta

Erilaisia syötteiden ja esiehtojen yhdistelmiä on lähes loputon määrä, joten näiden kaikkien testaaminen ei ole käytännössä mahdollista. Vain harvoin tulee vastaan tilanne, että syötteiden ja esiehtojen määrä on sen verran rajallinen, että niiden syöttäminen on mahdollista. Käytännössä esimerkiksi tavallisen laskimen täydellinen testaus tarkoittaisi sitä, että olisi yritettävä kaikki mahdolliset luvut ja toiminnot sekä näiden yhdistelmät, joita laskimesta löytyy. Testitapauksia olisi lähes loputon määrä, sellaisen määrän suunnittelu ja ajaminen ei ole realistista. Testattavien tilanteiden määrää on vähennettävä ja valittava testit siten, että niillä on todennäköistä löytää mahdollisimman paljon virheitä ohjelmistosta. Tästä syystä testausta suunnitellessa on tehtävä riskianalyysi, päätettävä testaustekniikat ja priorisoitava testaus. Riskianalyysillä tarkoitetaan testauksen yhteydessä sitä, että riskit pyritään tunnistamaan ja sen jälkeen tunnistetut riskit tutkitaan tarkemmin. Tarkemmassa tutkinnassa keskitytään luokittelemaan riskit, todennäköisyyden ja vaikutusten arviointiin. (ISTQB, 2018, s. 15; ISTQB, 2012, s. 24; Homes, 2013,s 11; Ghahrai, 2018b)

2.1.3 Aikainen testaus säästää aikaa ja rahaa

Staattiset ja dynaamiset testitapaukset sekä niiden ajaminen pitäisi aloittaa aikaisessa vaiheessa ohjelmistokehityksen elinkaarta, jotta saataisiin vähennettyä kalliiksi tulevia kuluja tai poistettua ne kokonaan. Staattisella testillä tarkoitetaan sitä, että ohjelmistoa katselmoidaan ja analysoidaan, ennen kuin siihen liittyvää testiä suoritetaan ja dynaamisella testillä tarkoitetaan varsinaista testin ajamista ohjelmistolla. (ISTQB, 2018, s.15; Tuovinen, 2013; Taina, 2007)



Kuva 1. Vikojen suhteellinen korjauskustannus

Sillä on kustannusten kannalta merkitystä, missä vaiheessa vika havaitaan ja korjataan. Mitä aikaisemmin vika havaitaan, sitä taloudellisempaa sen korjaaminen on. Kuvassa 1 on esitetty korjauskustannusten suhteellinen nousu ohjelmistokehityksen eri vaiheissa. Korjauskustannusten lisäksi viasta saattaa ylläpitovaiheessa aiheutua julkisuuskuvan huonontuminen, mikäli vika näkyy asiakkaille ja liiketoiminnallista haittaa, jos työn tekeminen ei esimerkiksi onnistu vian vuoksi. (Dawson, Burell, Rahim & Brewster 2010, 51; Homes, 2013, s. 12)

2.1.4 Viat kasaantuvat

Ennen julkaisua löytyneistä vioista suurin osa virheistä löytyy pienestä osasta komponentteja. 80 % löydetyistä virheistä löytyvät 20 % komponenteista. Vikakeskittymien ennusteet, testauksessa aiemmin tai saman ohjelmiston tuotantokäytössä löytyneet viat ovat tärkeä osa riskianalyysiä, jolloin testauspainotus on helpompi kohdentaa. Kun testauksessa havaitaan virhe, on hyvä testata samaa koodin osiota tarkemmin. Tämä johtuu siitä, että yleensä virheitä kasaantuu saman henkilön tekemiin osuuksiin, samaan aikaan tehtyihin komponentteihin tai jopa viallisiin valmiskomponentteihin. (ISTQB, 2018, s. 15; Homes, 2013, s. 12; Ghahrai, 2018b)

2.1.5 Varo hyönteismyrkkyparadoksia

Hyönteismyrkkyparadoksilla tarkoitetaan sitä, kun esimerkiksi maatilalla käytetään samaa hyönteismyrkkyä torjumaan hyönteisiä ja jossain vaiheessa hyönteiset tulevat immuuneiksi kyseiselle myrkylle. Vastaava ilmiö voidaan havaita myös testauksessa. Tietyssä vaiheessa kun testejä on

ajettu ja testaus on kehittynyt paremmaksi, löytyneiden vikojen määrä alkaa laskea. Vikojen määrä laskee, koska ne on löydetty ja korjattu. Samoilla testeillä ei ole mahdollista löytää lisää vikoja. Testiaineistoa tai/ja -dataa on muutettava siten, että uusien virheiden löytyminen mahdollistuu. Voi olla, että on tehtävä kokonaan uusia testejä. Automatisoitujen regressiotestien osalta tämä paradoksi tuottaa sellaisen lopputuloksen, jota on odotettukin, eli löytyneiden virheiden määrä on suhteellisen pieni. Samojen testien ajaminen on käytännössä tarpeellista vain regressiotestauksessa, jossa varmistetaan, että ohjelmisto toimii kuten ennenkin. Sen jälkeen, kun virhe on havaittu, korjattu ja uudelleen testattu, sitä ei pitäisi olla tarvetta testata enää uudelleen. Myös regressiotestejä on muokattava, jos ohjelmistoon tehdään muutoksia, jotka vaikuttavat perustoiminnallisuuksiin. (ISTQB Certification, 2009; Ghahrai, 2018b; ISTQB, 2018, s. 15; Homes, 2013, s. 12)

2.1.6 Testaus on tilannesidonnaista

Testausta ei voida tehdä samalla tavalla kaikissa tilanteissa, esimerkiksi mobiilisovelluksen testaus on tehtävä eri tavalla kuin turvallisuuskriittinen teollinen ohjausjärjestelmä. Lisäksi testaus on hyvin erilaista ketterissä projekteissa verrattuna vesiputousmalliin. Testitapausten suunnittelu perustuu pitkälti siihen tietoon, mitä testaaja tietää ohjelmistosta. Kun testausta tehdään uudelleen esimerkiksi versiopäivityksen jälkeen, ohjelmiston tuntemus on huomattavasti parempaa, jolloin testitapauksetkin ovat erilaisia ja monipuolisia. Testaukseen vaikuttaa käytettävissä oleva aika ja resurssit, aikaisemmin tehdyt havainnot, koodin laatu ja testauksen näkökulmat. Toisten ohjelmistojen tai projektien testien uudelleenkäyttäminen voi tuntua järkevältä, mutta todellisuudessa se vaikuttaa kielteisesti testaukseen, koska tilanne on erilainen. Olemassa olevia testejä kannattaa kuitenkin käyttää, mutta soveltaen niitä omaan testaukseen sopivaksi. (ISTQB, 2018, s. 15; Homes, 2013, ss. 13-14)

2.1.7 Virheiden poissaolo on harhaluulo

On hyvin yleistä, että organisaatioissa odotetaan, että testauksessa suoritetaan kaikki mahdolliset testitapaukset ja löydetään kaikki mahdolliset viat. Mutta aikaisemmin esitellyt periaatteet 1 ja 2 eli, että testaus osoittaa virheiden läsnäolon, ei niiden poissaoloa sekä, että täydellinen testaus on mahdotonta, osoittavat tämän täysin mahdottomaksi.

Yleinen harhaluulo on myös siinä, että jos vikoja on löydetty ja korjattu runsaasti, se takaa järjestelmän onnistumisen. Vaikka kaikki järjestelmän vaatimukset testattaisiin läpi ja virheet korjattaisiin, on mahdollista silti, että jäljelle jää vain hankalasti käytettävä ohjelmisto, jolla ei voida täyttää käyttäjien tarpeita. Hyvin usein virheiden vähentämistä pidetään testauksen perimmäisenä tavoitteena. Testauksella ei kuitenkaan ole mahdollista vakuuttaa, että vaikka virheitä ei enää olisi löytynyt, se sopisi julkaistavaksi. Virheiden puuttuminen ei korvaa käyttäjien odotuksiin vastaamista

eikä se auta arkkitehtuurissa oleviin ongelmiin tai takaa taloudellista voittoa. (ISTQB, 2018, s.15; Homes, 2013, s.14)

2.1.8 Muuta huomioitavaa

Testaukseen on hyvä osallistua sellaiset henkilöt, jotka eivät ole olleet kehittämässä ohjelmistoa. Testaajiksi tulisi valita parhaat henkilöt, joilta löytyy luovuutta ja vastuuntuntoisuutta. Testauksessa on aina huomioitava myös käyttäjän virheellinen toiminta, jotta ohjelmisto pystyy varmasti käsittelemään näitä virhetilanteita tilanteita. Testauksen aikana ohjelmistoon ei tulisi asentaa uusia komponentteja tai päivityksiä. Testitapausten suunnittelussa tulee huomioida myös se, miten ohjelmiston odotetaan toimivan testitapauksen tilanteessa. (Ghahrai, 2018b)

2.2 **Testauksen havainnot**

Testauksen myötä ohjelmistosta tehdyt havainnot rajautuvat virheisiin/erehdyksiin (mistake), vikoihin/bugeihin (error, bug) tai häiriöihin (failure). Virheellä tarkoitetaan ihmisen tekemää virhettä, joka aiheuttaa ohjelmistoon vian. Ihminen saattaa tehdä virheen aikapaineen, erehtyväisyyden, kokemattomuuden tai osaamattomuuden vuoksi. Lisäksi väärinymmärrykset liittyen vaatimuksiin, suunnitteluun tai liittyen järjestelmän ulkoisiin ja sisäisiin liittymiin voivat aiheuttaa virheitä tehtyyn ohjelmistoon. Tekijä saattaa tehdä virheen myös monimutkaisen koodin, arkkitehtuurin, suunnitelmien ja/tai teknologioiden vuoksi. (Kasurinen, 2013, s.50; ISTQB, 2018, s.14)

Vialla tai kuten puhekielessä usein käytetään, bugilla tarkoitetaan sellaista virhettä, joka on synnyttänyt poikkeaman ohjelmistoon. Tämä poikkeama johtaa siihen, ettei ohjelmisto toimi vaatimusten mukaan tai aiheuttaa pahimmillaan häiriön ohjelmistoon, joka estää toiminnan kokonaan tai johtaa vikaan toisessa komponentissa. Vika voi johtua myös virheestä vaatimusdokumentaatiossa, joka voi johtaa ohjelmointivirheeseen, joka puolestaan aiheuttaa havaitun vian. Aina vika ei johda suoraan ohjelmiston häiriöön. Testauksella pyritään varmistamaan, että merkittävimmät viat on poistettu ohjelmistosta. (Kasurinen, 2013, s.50; International Software Testing Qualifications Board 2018, s.14)

Häiriötilanteessa ohjelmisto ei toimi kuten sen pitäisi, häiriö voi aiheutua viallisen koodin lisäksi muista mahdollisista asioista, jotka häiritsevät ohjelmiston toimintaa esimerkiksi laitteistosta, magneettikentästä, huonepölystä, sähkökentästä tai mistä tahansa muusta asiasta, jolla on vaikutus ohjelmiston tai järjestelmän toimintaan. (Kasurinen, 2013, s.50; International Software Testing Qualifications Board 2018, s.14)

2.3 Regressiotestaus ja testausautomaatio

Regressiotestauksella tarkoitetaan käytännössä uudelleen testaamista. Kyseessä on testauksen yleistermi, joka ei ole testaustaso, kuten esimerkiksi integraatio- tai yksikkötestaus, tai yksittäinen testausmenetelmä, kuten musta laatikkotestaus tai tutkiva testaus. Regressiotestaus on tarpeen, kun ohjelmiston yhtä tai useampaa osaa muutetaan, jotta voidaan varmistua, että ohjelmisto toimii edelleen oikein. Regressiotestausta tehdään myös isommissa kehityshankkeissa, joissa saadaan valmiiksi osatavoite (milestone). Uusimmasta versiosta on varmistettava, että kokonaisuus toimii edelleen kuten aikaisemmissakin versioissa. Regressiotestauksen tärkein tehtävä on todentaa, ettei uudessa versiossa ilmene vanhoja jo korjattuja tai uusia vikoja. Usein ohjelmiston viat löytyvät uusista komponenteista tai toiminnoista, jotka käyttävät kyseisiä komponentteja. (Kasurinen, 2013, s. 69)

Regressiotestausta tehdään kaikilla testaustasoilla, jotta viat löytyisivät mahdollisimman aikaisin. Regressiotestaukseen kuuluu, että testitapauksia ajetaan lukuisia kertoja ja niiden kehittyminen vie aikaa. Ketterässä kehittämisessä regressiotestauksen rooli korostuu, kun uudet ominaisuudet ja koodin refaktorointi tuottavat jatkuvia muutoksia. (ISTQB, 2018, s. 36)

Testausautomaatiolla tarkoitetaan testauksen muotoa, jossa ohjelmiston testaukseen hyödynnetään automaatiotyövälinettä (Kasurinen, 2013, ss. 76-79). Työvälineellä saadaan käytännössä simuloitua käyttäjän toimintaa tai sitten järjestelmän toimintaa (Cinia, 2019). Testausautomaatio soveltuu sellaiseen testaukseen, jossa toistetaan usein samoja testejä. Näin saadaan vapautettua testaajien työaika muihin testauksen työtehtäviin. Testaustestausautomaatio on ollut käytössä jo pitkään käyttöliittymien testauksessa ja on perinteinen testausautomaation kohde. Testausautomaatiolla ei voida kuitenkaan kokonaan korvata manuaalista testausta, vaan niiden tulisi toimia rinnakkain. (Kasurinen, 2013, ss. 76-79) Tämä siitä syystä, että testausautomaatio toimii parhaiten vakaisiin ominaisuuksiin ja samanlaisena toistuviin testeihin. Manuaalista testausta tekevien työ voidaan kohdentaa tällöin paremmin nopeatempoiseen testaukseen, jossa testauksen kohde ja näkökulma vaihtuvat, testauksen suunnitteluun, testausautomaation tulosten analysointiin ja testausautomaation luomiseen. Manuaalinen ja automaattinen työ täydentävät toisiaan siten, että automaatio tekee runsaasti toistoa sisältävät testaukset ja monimutkaisemmat testit jäävät ihmisen työksi. Käytännössä tämä tehostaa testausta, kun ihmiseltä jää pois iso osa toistuvista testeistä, jotka vievät aikaa monimutkaisemmilta tehtäviltä. (Fehrend, n.d)

Testausautomaatiota on ylläpidettävä, jotta sen käyttö pysyy tehokkaana, tästä syystä sille on varattava riittävästi resursseja. Tästä huolimatta testausautomaatiolla voidaan säästää kustannuksissa, mikäli usein toistuvien testien manuaalinen testaus on kalliimpaa, kuin testausautomaation ke-

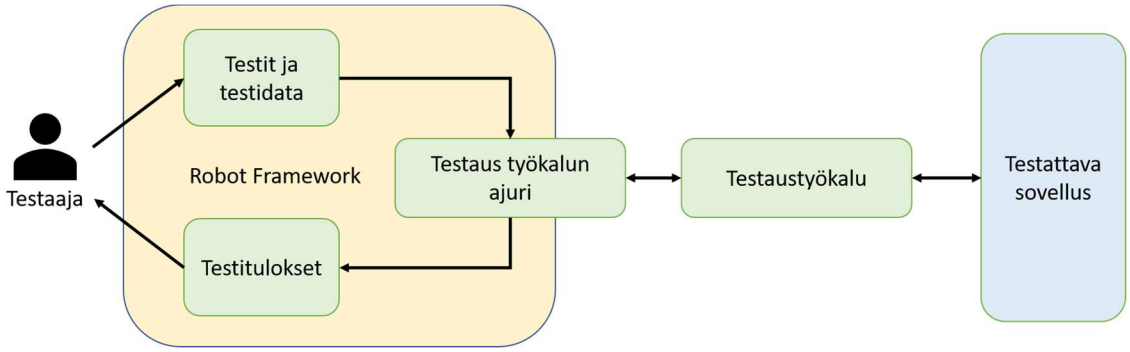
hittäminen ja sen ylläpitäminen. Yleensä varsin otollisia testausautomaation kohteita ovat yksikkö- ja savutestaukset, joilla varmistetaan järjestelmän perustoimivuus. Automaatiolla ei tulisi pyrkiä etsimään uusia virheitä, vaan todentaa, että järjestelmä toimii kuten ennenkin. (Kasurinen, 2013, ss. 76-79)

Testausautomaatiolla on monia hyötyjä testauksen tehostamisessa, mutta sillä on myös riskejä. Odotukset saattavat olla epärealistisia, tähän lukeutuvat myös työkalun toiminnallisuus ja helppokäyttöisyys. Jotta testausautomaation hyöty olisi merkittävä ja jatkuva, sitä on ylläpidettävä, parannettava ja muutettava muuttuvien testausprosessien mukaan, tämä vaatimus aliarvioidaan usein ajallisesti ja työmäärällisesti. Lisäksi toimivan testimateriaalin ylläpitoon kuluvaa aikaa saatetaan aliarvioida. Testausautomaatiota saatetaan käyttää testauksen perustana testien suunnittelussa ja testien suorituksessa, vaikka kyseessä olisi testit, jotka olisi parempi suorittaa manuaalisesti. Tällöin tukeudutaan liikaa testausautomaatioon. Työkalun omistussuhteet voivat olla monimutkaisia ja epäselviä, eikä ole varmuutta kenellä on koulutus- ja päivitysvastuu. Työkalu saattaa myöhemmässä vaiheessa joutua uuden teknologian myötä yhteensopivuusongelmiin. Muita riskejä ovat versionhallinnan ja kriittisten työkalujen yhteensopivuuteen liittyvien ongelmien laiminlyönti, työkalun toimittajan toiminnan päättyminen, välineen poistuminen markkinoilta tai myyminen toiselle toimittajalle, tukipyyntöjen huono hoitaminen ja huonot vikakorjaukset työkaluun tai avoimen lähdekoodin tuottamisen päättyminen. (ISTQB, 2018, s. 72)

Regressiotestauksessa käytetään runsaasti perustestitapauksia, joissa on järkevää hyödyntää testausautomaatiota. Mitä useammin regressiotestausta tehdään, sen keskeisempää testausautomaation kehittäminen on. (Kasurinen, 2013, s. 70) Testausautomaation ei ole tarkoitus olla testauksen, vaan laadun valvonnan työkalu. Se ei sovellu siihen, että tutkitaan uusien osien toimivuutta, vaan se on parhaimmillaan varmistus siitä, etteivät jo käytössä olevat osat rikkoudu muutosten myötä. (Kasurinen, 2013, s. 79)

2.4 **Robot Framework**

Robot Framework on avoimen lähdekoodin sovelluskehys, jota käytetään yleisesti testauksen automatisoinnissa. Se on joustava ja laajennettavissa toimimaan useiden back end teknologioiden kanssa. Yhdistettynä Seleniumin verkkosivujen automatisointi työkaluun, se on yksi yleisimmistä testausvälineistä automatisoidun testauksen puolella. Laajennuksia kutsutaan kirjastoiksi, osa kirjastoista on sisäänrakennettu Robot Frameworkiin ja osa on ladattavissa erikseen. Lisäksi on mahdollista luoda Pythonilla omia kirjastoja käyttöön. (Bisht, 2013, s. 26) Robot Frameworkia on mahdollista laajentaa Pythonin lisäksi myös Javalla. Se on käyttöjärjestelmästä ja ohjelmistoista riippumaton. (Robot Framework Foundation, n.d.)

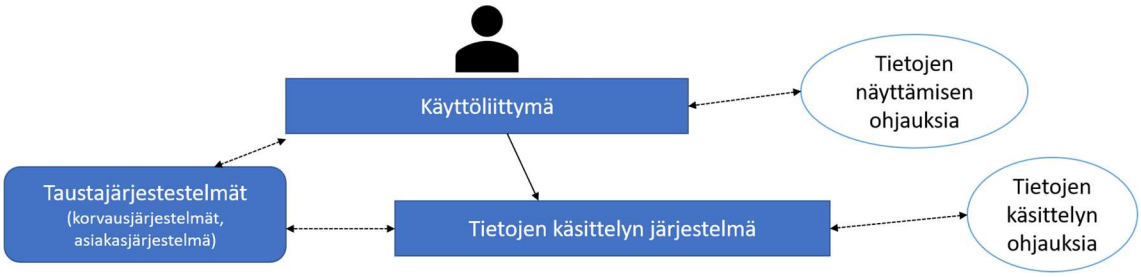


Kuva 2. Robot Framework toiminta kuvana

Kuten kuvassa 2 on kuvattu, Robot Framework on varsinainen ydin, joka suorittaa useita testauksen aikaan saavia prosesseja. Robot Framework vertailee testejä, testidatatiedostoja ja kansioita ja päättelee niiden perusteella, miten ja mitkä testit suoritetaan. Tämän jälkeen Robot Framework käyttää testaustyökalun ajuria kommunikoidakseen varsinaisen testaustyökalun kanssa. Esimerkiksi selainta käyttävissä testeissä Robot Frameworkin käytössä on oltava selaimen ajuriohjelmisto ja varsinainen selain. Selainta ja Seleniumia käytetään testaustyökaluna, jolla saadaan varsinaista testattavaa sovellusta testattua. Lopputuotoksena testaaja saa testitulokset xml- ja html-muodossa tarkistettavakseen. (Bisht, 2013, s. 27)

3 TESTAUSAUTOMAATION TOTEUTUS

Testattava sovellus on yrityksen sisäisessä käytössä oleva korvausprosessin tehostamiseen tarkoitettu sovellus, joka hyödyntää taustalla useamman järjestelmän tarjoamia tietoja sekä tallentaa tarvittavat tiedot samoihin järjestelmiin. Sovellus toimii ulkoisten ohjausten perusteella ensin käyttöliittymän tasolla. Tietojen lähettämisen jälkeen tietojenkäsittelyssä käytettävä logiikka on rakennettu sovelluksen sisään, mutta se hyödyntää ulkopuolelta saamiaan ohjaustietoja. Nämä ohjaustiedot voivat olla esimerkiksi erilaisia raja-arvoja erilaisille syötetyille tiedoille. (Kuva 3) Sovellus on pääsääntöinen työväline asiakaspalvelutyötä tekeville, jonka vuoksi se on luokiteltu erittäin tärkeäksi sovellukseksi liiketoiminnan jatkuvuuden kannalta.



Kuva 3. Testattavan sovelluksen kuvaus ylätasolta

Testattava sovellus on ollut loppukäyttäjien käytössä vuodesta 2016. Sovellukselle on tehty tämän jälkeen lukuisia muutoksia. Vuoden 2018 aikana sovellukselle tehtiin uusia kytköksiä, jotka vaikuttivat alkuperäisen toteutuksen toimintaan heikentävästi. Johtuen puutteellisesta regressiotestauksesta, vikoja ei ehditty havaita ennen tuotantoympäristöön siirtymistä. Sovelluksen toimiessa monimutkaisessa useamman teknologian tuotantoympäristössä, jossa on runsaasti rajapintakytköksiä, on regressiotestaus tehtävä säännöllisesti. Kuitenkaan todellisuudessa riittäviä resursseja ei ole tarjolla kattavan regressiotestauksen tekemiseen. Muutoshallinta on määritellyt, että yrityksen sovelluksille on mahdollista siirtää kerran kuukaudessa muutoksia tuotantoympäristöön, kiireisiä poikkeuksia lukuun ottamatta. Selkeä aikataulutus helpottaa regressiotestauksen suorittamista.

Tuotantovirheen kustannukset ovat suuria, varsinkin sellaisen sovelluksen osalta, joka toimii useamman järjestelmän kanssa yhdessä. Juurisyyn selvittäminen on usein hidasta ja virhetilanteisiin liittyviä tietoja ehtii kadota, koska tuotantoympäristössä ei voida varastoida tietoja kuin hyvin lyhyitä aikoja, muistikapasiteetin rajallisuuden takia.

Normaali käyttäjä ei näe kovin paljon siitä, mitä sovelluksen pinnan alla tapahtuu. Todellisuudessa järjestelmät kommunikoivat useassa eri vaiheessa toistensa kanssa (Liite 2).

3.1 **Saavutettavat hyödyt**

Puutteellinen regressiotestaus on aiheuttanut aikaisemmin haasteita tuotantoympäristössä, minkä vuoksi liiketoiminta osallistuu aktiivisesti itse regressiotestauksen suorittamiseen. Regressiotestauksen tarvetta ilmenee kuukausittain ja suoritettavia testejä on 176 kappaletta. Regressiotestauksen vaiheet ovat manuaalisessa testauksessa sopivan testidatan etsiminen, testidatan syöttäminen käyttöliittymään, lopputuloksen tarkistus kolmesta taustajärjestelmästä ja testauksen hallintajärjestelmän päivitys testitapausten osalta.

Testausautomaation myötä käytetään samaa testidataa jokaisella ajokerralla, tiedot syötetään käyttöliittymään ja tiedot tarkistetaan kolmesta taustajärjestelmästä, lisäksi luodaan raporttitiedosto csv-tiedostona. Testaajalle jää testausautomaation edelleen tehtäviä, joita Robot Frameworkilla ei voi ratkaista. Lopputuloksen tarkistaminen tilanteessa, jossa testi on päätynyt FAIL tilaan, vaatii erityisosaamista testauksen kohteena olevasta sovelluksesta.

Manuaalisen testauksen kesto vaihtelee tapauksen mukaan, mikäli testi onnistuu kerralla eikä siinä ilmene epäselvyyksiä, kestää yhden testitapauksen tekeminen noin 20 minuuttia. Mikäli testitapauksessa ilmenee jotakin poikkeavaa, kuluu aikaa lisää 15 minuutista eteenpäin, koska testaajan on selvitettävä määrittelyistä ja järjestelmän omista ohjauksista mahdollista virheen aiheuttajaa. Mikäli tämä ei onnistu, joudutaan epäonnistunut testi siirtämään järjestelmän toimittajalle tutkittavaksi, tällöin selvittelyn kustannukset ja aika kasvavat merkittävästi. Yleensä näissä tilanteissa järjestelmästä löytyy jokin virhe, joka vaatii korjausta.

Opinnäytetyön aikana rajatiin testausautomaation tekeminen 34 testitapaukselle 176 kappaleen sijaan. 34 testitapauksen manuaaliseen suorittamiseen kuluu manuaaliselta testaajalta parhaimmillaan 680 minuuttia eli 11,3 tuntia. Yleensä kaikki testitapaukset eivät mene ongelmitta läpi edes manuaalisessa testauksessa, lisäksi testaajat joutuvat yleensä pilkkomaan testaukseen käytettävän ajan muiden tehtävien lomasta eikä testausta tehdä kuin muutama tunti kerrallaan. Mikäli yksikään testitapaus ei mene oletetusti läpi, vie 34 testitapauksen manuaalinen testaus aikaa 1190 minuuttia eli 19,8 tuntia. Saavutettavaksi hyödyksi voidaan arvioida testaajien työajan vähentäminen toistuvien testien osalta ja virheiden varhainen havaitseminen automatisoidulla testauksella.

3.2 Testitapausten suunnittelu

Sovellukselle on tehty regressiotestausta aikaisemmin manuaalisesti, jonka vuoksi testitapaukset löytyivät jo valmiiksi dokumentoituna. Kohdesovelluksen regressiotestauksen minimitaso sisältää 176 kappaletta erilaisia testitapauksia. Tässä osassa toteutettiin niistä 34 kappaletta. Rajaus tehtiin käytettävissä olevan ajan ja kriittisyyden perusteella, joka arvioitiin sen perusteella missä osuuksissa häiriöitä on eniten esiintynyt tuotantoympäristömuutosten myötä.

3.3 Testien toteutus

Testausautomaatio toteutettiin käyttäen Python 3.6 versiota, joka vakaampi yhteensopivuuden näkökulmasta kuin Python 3.7. Robot Frameworkistä käytettiin versiota 3.1 sekä Selenium-kirjastoa 3.141.0. Selaimena käytetiin Chromea ja ajurina uusinta chromedriveria.

Tarkoituksena oli luoda datatiedosto, josta haetaan data testin syöttämiseksi. Testaaja voi itse vaihtaa dataa helposti, eikä tietoja ole kovakoodattu itse robot-tiedostoon. Testidata tiedosto toteutettiin csv-tiedostona, johon jokaiselle riville muodostettiin kyseisen testin tarvitsemat tiedot. Testiajo aloitetaan suorittamalla komentorivillä testitiedoston sijainnissa robot --timestampoutputs -d ./tulokset -i Uusi_sairauskuluhakemus_ilman_kuluja_AUT -v testiajonumero:1 KPA-testit.robot. Komennon avulla jokainen lokitiedosto, joka testin aikana muodostetaan, saa oman aikaleiman, joka helpottaa tulosten tarkistamista ja ne kirjoitetaan kansioon ”tulokset”, Robot Framework luo tämän nimisen kansion, jos sitä ei testin lopuksi vielä ole olemassa. Muuttuja annetaan viimeisenä kohdassa -v testiajonumero:1. Kokonainen testiajo suoritetaan valmiilla skripti-tiedostolla (koodi 1). Käytännössä testiajo suorittaa kaikki testit peräjälkeen, eikä niitä tarvitse käynnistää yksitellen.

```
robot --timestampoutputs -d ./tulokset -i Uusi_sairauskulu-
hakemus_ilman_kuluja_AUT -v testiajonumero:1 KPA-
testit.robot
robot --timestampoutputs -d ./tulokset -i Uusi_sairauskulu-
hakemus_ilman_kuluja_AUT -v testiajonumero:2 KPA-
testit.robot
robot  --timestampoutputs -d ./tulokset -i Uusi_sairausku-
luhakemus_ilman_kuluja_AUT -v testiajonumero:3 KPA-
testit.robot
...
robot --timestampoutputs -d ./tulokset -i Jatkohakemus_MAN
-v testiajonumero:33 KPA-testit.robot
robot --timestampoutputs -d ./tulokset -i Uusi_sairauskulu-
hakemus_kulujen_kanssa_MAN -v testiajonumero:34 KPA-
testit.robot
```

Koodi 1. Testien ajoskripti

Tämän jälkeen Robot Frameworkin koodi muodostettiin siten, että se toistaa testiä täysin vastaavasti kuin manuaalisessa testauksessa. Kun testidata on saatu syötettyä käyttöliittymälle, oli varmistettava millä perusteella tiedot ovat oikein ja vastaa haluttua lopputulosta.

Yhden testitapauksen aikana automaatiolla on käytettävä neljää eri sovellusta, joista tarkistetaan ja verrataan tietoja toisiinsa. Lisäksi testitapauksen tietyt tiedot kirjoitetaan lopuksi csv-tiedostoon, jotta testaajan on helppo tarkistaa testiajojen lopputulos. Prosessi johtuu sovelluksen monimutkaisesta logiikasta ja siitä, että se toimii useamman taustajärjestelmän välillä. Prosessi on pyritty yksinkertaistamaan testausautomaatiolle sopivaksi, erilaisia variaatioita ei pyritä ajamaan regressiotestauksessa. Liitteessä 1 on kuvattu testausautomaation toimintalogiikka kuvana.

Jotta testit voitaisiin kirjoittaa tehokkaammin, ilmeni joitakin tilanteita, joissa omien avainsanojen luominen Robot Frameworkilla oli järkevää. Esimerkiksi useita eri elementtejä joudutaan odottamaan sivulla. Ensin sen on oltava sivulla, seuraavaksi sen on oltava aktivoituna ja lopuksi vielä näkyvissä. Joskus sivulla jokin näistä toteutuu, mutta eivät kaikki, jolloin Robot Framework ei kykene käyttämään elementtiä sivulla halutulla tavalla. Lisäksi samaa avainsanaa on hyvä toistaa riittävän pitkään, jotta testi ei epäonnistu sivun lataamisen hitauden vuoksi. Koodissa 2 on esitettynä sama toiminto pitkänä koodina ja koodissa 3 lyhyempänä. Koodin 3 Odota -avainsana sisältää koodin 2 avainsanat. Usein toistuessa koodia 3 on tehokkaampaa käyttää.

```
Wait Until Keyword Succeeds 5 x 2 s Wait Until Page Contains
Element  ${elementin tunnus}
Wait Until Keyword Succeeds 5 x 2 s Wait Until Element Is
Enabled  ${elementin tunnus}
Wait Until Keyword Succeeds 5 x 2 s Wait Until Element Is
Visible  ${elementin tunnus}
```

Koodi 2.

```
Odota ${elementin tunnus}
```

Koodi 3.

Testidatassa käytettiin asiakasnumeroa, jota oli muokattava eri tilanteisiin sopivaksi. Käyttöliittymälle siirrytään selaimen kautta käyttäen URL-osoitetta, johon sisällytetään asiakasnumero ilman tarkistemerkkiä ja välimerkkiä eli A tai -. Tarkoitukseen käytettiin String-kirjaston Remove String Using Regexp -avainsanaa.

Sovelluksen ensimmäinen käyttöliittymä on html-pohjainen, jossa elementit oli helppo paikantaa Selenium-kirjaston avulla. Riippuen testitapauksesta, id ja name attribuuttien nimet kuitenkin muodostuivat eräänlaisella järjestysnumerologiikalla, joissa kuitenkin loppuosan teksti oli samanlainen. Näissä tilanteissa käytettiin xpath-viittauksia oikean elementin löytämiseksi. Yleisimmin käytetyt xpath-viittauksia olivat tekstihaku

esimerkiksi //button[text()="Submit"] ja tekstihaku alimerkkijonolla (substring) esimerkiksi //button[contains(text(),"Go")]. Koodissa on käytetty esimerkiksi tämänkaltaista xpathia kuin //*[contains(@id, "btnSairausTallenna")], joka etsii mistä tahansa isäntäelementistä elementtiä, jonka id sisältää lainausmerkeissä olevan tekstin. Toinen esimerkki koodista on *//span[contains(text(), "Valitse")], joka etsii mistä tahansa isäntäelementistä span elementtiä, joka sisältää lainausmerkeissä olevan tekstin.

Koska käyttöliittymä lataa sivua uudelleen riippuen joistakin valinnoista, oli robot-koodiin laitettava välillä pitkiä toistolauseita tai käytettävä BuiltIn-kirjaston Sleep -avainsanaa muutamaksi sekunniksi, jotta juuri annetut tiedot eivät tyhjentyisi välittömästi niiden syöttämisen jälkeen, kun sivu päivittyy. Osassa sivuja toimi toistolause, joka täyttää kentän tiedot useampaan kertaan, jolloin ne eivät tyhjene, kun sivu päivittyy.

Testin aikana tarkistetaan sisältääkö auennut sivu kaikki elementit tai tekstit, joita sivulla kuuluu olla. Tämä toteutettiin käyttämällä seuraavaa syntaksia. Ensin määriteltiin lista, joka sisältää elementit (koodi 4).

```
@{vahinkoilmoituksen_sisalto}
... *//label[contains(text(),"Vahinkotapahtuman yhtveto")]
... *//label[contains(text(),"Ohjeet asiakkaalle")]
... *//label[contains(text(),"Kumppaniohjaustiedot")]
... *//label[contains(text(),"Vahinkotapahtuman tiedot")]
... (//label[contains(text(),"Vakuutettu")])[2]
... //input[contains(@name, "VakuutettuNimi")]
... //input[contains(@name, "VakuutettuKatuosoite")]
... //input[contains(@name, "VakuutettuKunta")]
... *//label[contains(text(),"Vahinkotapahtuman tunnus")]
... //input[contains(@name, "VahinkotapahtumanTunnus")]
... *//label[contains(text(),"Vahinkotapahtumapäivä")]
... //input[contains(@name, "VahinkotapahtumanPaiva")]
... *//label[contains(text(),"Vahinkotapahtuman tyyppi")]
... //input[contains(@name, "VahinkotapahtumanTyyppi")]
... *//label[contains(text(),"Sairaus tai vamma")]
```

Koodi 4. Listamuuttuja

Koodissa 4 esitelty lista käydään toistolauseella läpi (koodi 5). Lisäksi tässä käytetyt BuiltIn-kirjaston avainsanat Run Keyword And Ignore Error ja Run Keyword And Continue On Failure toimivat siten, ettei testi pääty, mikäli elementtiä ei löydy. Testin lopuksi tämä kuitenkin tulee tietoon lokitiedostolle ja testi päättyy tilaan FAIL.

```
FOR ${index} IN @{vahinkoilmoituksen_sisalto}
  Scroll Element Into View xpath=${index}
  Element Should Be Visible  xpath=${index}
END
```

Koodi 5. Toistolause

3.4 **Kehitystyön haasteet**

Ensimmäisen käyttöliittymän merkittävimmät haasteet olivat jo aikaisemmin mainittu tietojen uudelleen lataaminen, joka päivittää käyttöliittymän sivua sekä spontaanit virheilmoitukset, joita testausautomaatiolla ei pystynyt havaitsemaan. Todennäköisesti näihin tilanteisiin voi kuitenkin jatkokehityksessä kehittää sopivan ratkaisun.

Toisessa käyttöliittymässä eli työjonossa, johon tehtävä muodostuu, haasteina oli iframe-elementeillä rakennettu sivu, joka ei ollut entuudestaan tuttu. Tämä saatiin kuitenkin ratkaistua testausautomaation asiantuntijoiden kanssa yhdessä, kun sivulla olevat iframe-elementit oli saatu tunnistettua. Ennen kuin iframe-elementin sisällä olevaan elementtiin pääsi kiinni, oli valittava ensin iframe-elementti käyttäen Selenium-kirjaston avainsanaa ”Select Frame”. Sivulla oli lisäksi useita sisäkkäisiä iframe-elementtejä, jolloin ensin oli valittava ensimmäinen ja sen jälkeen toinen, jonka jälkeen vasta oli mahdollista jatkaa lomakkeella toimimista. Iframe-elementeistä poistuttiin käyttämällä Selenium kirjaston avainsanaa ”Unselect Frame”. Koodissa 6 valitaan sivulta ensimmäinen iframe-elementti, jonka jälkeen on odotettava, että seuraava iframe-elementti tulee näkyviin. Kun se on tullut näkyviin, se voidaan valita. Tämän jälkeen testi suorittaa toiminnot, jotka koodissa on ja lopuksi poistuu iframe-elementeistä. Koodissa on kuitenkin vielä elementtejä tarkistettavana, joten ylempään iframe-elementtiin on siirryttävä vielä uudemman kerran, jotta ne tulevat Robot Frameworkille näkyviin.

```
Select Frame  xpath=//iframe[contains(@id, "InlineFrame")]
Odota  xpath=//iframe[@id='webForm']
Select Frame  xpath=//iframe[@id='webForm']
...koodia ...
Unselect Frame
Select Frame  xpath=//iframe[contains(@id, "InlineFrame")]
Unselect Frame
```

Koodi 6. Iframe-elementtien käsittely esimerkki

Toisessa käyttöliittymässä haasteena oli myös se, että lomakkeilla näkyvät tiedot eivät olleet haettavissa käyttämällä Selenium-kirjaston avainsanaa ”Get Text”. Ongelman ratkaisu löytyi JavaScriptistä, jota Robot Frameworkilla voi kutsua käyttämällä Selenium-kirjaston avainsanaa ”Execute Javascript” ja kutsumalla siinä JavaScript metodia ”return document.querySelectorAll('[name=”elementname”]')[0].value”. Tämä JavaScript metodi palauttaa elementin sisällä olevan arvon (Mozilla, 2019).

Viimeisin haaste oli verrata haettuja korvauksia toisiinsa eri järjestelmissä, koska ne olivat eri muodossa jokaisessa järjestelmässä. Toiseen käyttöliittymään luvut tallentuivat muodossa 50.50 € ja seuraavassa niitä tulisi verrata muodoltaan 50,50 euroa olevaan lukuun, viimeisessä järjestelmässä luvut ovat muotoa 50,50 €. (Kuva 4)

Kuva 4. Eri järjestelmissä näkyvät eurosummat

Jotta lukuja voisi verrata, oli jompaakumpaa muotoiltava sopivaksi. Käyttämällä String-kirjaston avainsanaa ”Replace String Using Regexp” ja ”Remove String Using Regexp” (Kuva 5).

```
#tallennetaan haetut korvaukset
${haetut_korvaukset1}=  Arvon hakeminen querySelectorilla  name="HaetutKorvauksetYhteensa"
#korvataan piste pilkulla verkaa varten
${haetut_korvaukset2}=  Replace String Using Regexp  ${haetut_korvaukset1}  [.]  ,
#poistetaan €-merkki
${haetut_korvaukset}=  Remove String Using Regexp  ${haetut_korvaukset2}  [€]
Set Global Variable  ${haetut_korvaukset}  ${haetut_korvaukset}
```

Kuva 5. Regular expressionin käyttö Robot Frameworkilla

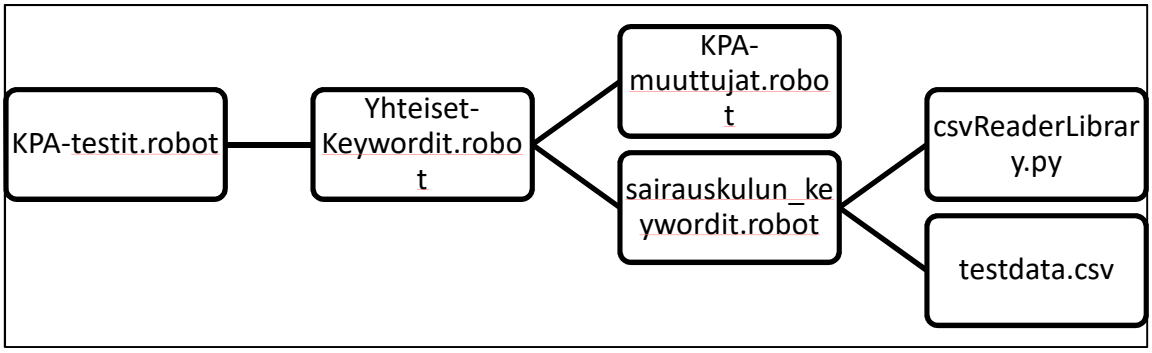
3.5 **Arkkitehtuurikuvaus**

Arkkitehtuurikuvauksessa pyritään kuvaamaan miten eri tiedostot kytkeytyvät toisiinsa luodussa testausautomaatioratkaisussa. KPA-testit.robot sisältää testattavat testitapaukset. Samaa testitapausta käytetään usean eri tuotteen testaukseen, joten erilaisia variaatioita on tätä enemmän. KPA-testit.robot kutsuu ajossa Yhteiset-Keywordit.robot tiedostossa olevia avainsanoja.

Yhteiset-Keywordit sisältää sellaiset avainsanat, joita voidaan käyttää yleisesti myös muissa testitapauksissa, jotka eivät ole vain sairauskulutuotteisiin liittyviä. Esimerkiksi päivämäärien asettaminen muuttujiin tai testitapauksen sulkemiseen liittyvät toimet. Koska testausautomaatiota jatkokehitetään myös muihin tuotteisiin, oli tarkoituksenmukaista jakaa heti yhteisesti käytettävät avainsanat omaan resurssitiedostoon.

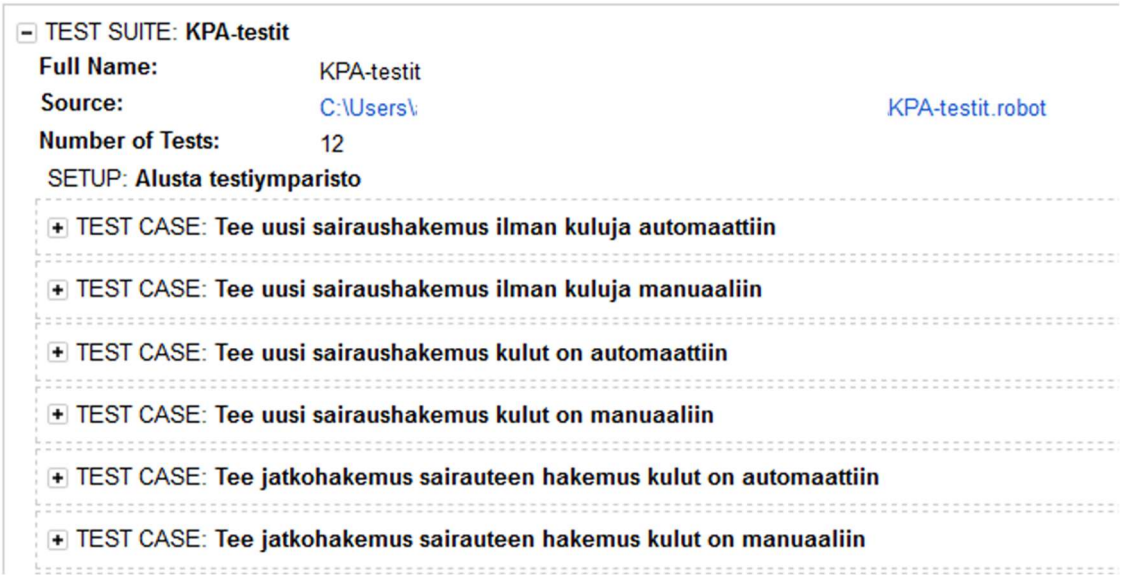
KPA-muuttujat sisältää 330 riviä erilaisia testissä tarvittavia muuttujia, näin muut dokumentit pysyivät paremmin luettavina. Esimerkiksi usein toistuvat painikkeet olivat järkevä pitää muuttujissa, joihin voi viitata useista avainsanoista. Sairauskulun_keywordit sisältää vain sairauskulutuotteille ominaisia avainsanoja, joita ei tarvita muualla.

cvsReaderLibrary on Pythonilla tehty oma yhden avainsanan kirjasto, joka liitettiin sairauskulujen avainsanoihin, kun testin alussa haetaan muuttujat testitapaukselle erillisestä csv-tiedostosta. (Kuva 6) Dokumentaation saa tuotettua seuraavalla komentorivin komennolla: python -m robot.testdoc KPA-testit.robot documentation.html.



Kuva 6. Arkkitehtuurikuvaus

Erilaisia testitapauksia on kuusi erilaista, nämä samat testitapaukset testataan useilla eri tuotteilla. Eri tuotevariaatiot muodostavat yhteensä 34 erilaista testiä (Kuva 7)



Kuva 7. Lista testitapauksista

3.6 **Koodin katselmointi**

Koodin katselmoi kaksi testausautomaatioasiantuntijaa ja tehdystä testausautomaatiokoodista nousi muutamia asioita esiin. Tarkistellussa koodissa muuttujien nimeämiskäytäntö oli vaihtelevaa, tämä vaikeuttaa koodin lukemista. Esimerkiksi osassa nimiä oli alaviivoja, osassa numeroita ja alkukirjainten koko vaihteli. Koodissa oli myös itse luotuja avainsanoja, suurimassa osassa näistä ei ollut dokumentaatiota. Avainsanat olisivat

tärkeä aina dokumentoida huolellisesti, jotta niiden toiminta olisi kuvattu selkeästi.

Erityisesti lyhyet avainsanat kaipaavat selitystä, esimerkiksi tilanteessa, jossa avainsana on luotu vain ehtolausetta varten. Robot Frameworkin syntaksi ehtolauseessa on yleensä seuraavanlainen ”Run Keyword If” ja tämän jälkeen on laitettava avainsana, joka halutaan ajaa. Ehtolauseen perään ei voi esimerkiksi asettaa muuttujan hakemista kuten koodissa 7, vaan se on tehtävä koodin 8 mukaisesti. ”Hae teksti” on tehty omaksi avainsanakseen, vaikka se sisältää saman toiminnan kuin koodissa 7.

```
Run Keyword If X == ${TRUE}  ${teksti}= Get Text id=xyz
```

Koodi 7. Epäkelpo syntaksi

```
Run Keyword If X == ${TRUE}  Hae Teksti
Hae teksti
${teksti}= Get Text id=xyz
```

Koodi 8. Kelvollinen syntaksi

Globaaleja muuttujia on käytetty runsaasti, vaihtoehtona olisi käyttää argumentteja, jotka lähetetään tietylle avainsanalle. Tällöin muuttujissa liikkuvat tiedot olisivat paremmin hallittavissa.

Tarkistus, onko vakuutettu yli tai alle 18 vuotias, puuttui tasan 18 vuotta vaihtoehto. Suorien xpath-viittausten käyttö voi aiheuttaa riskejä, sivulla mahdollisesti tapahtuvien muutosten vuoksi. Turvallisemmaksi vaihtoehdoksi suositeltiin followingin tyyppisiä xpath-viittauksia. Following voi tarkoittaa esimerkiksi sitä, että jokin elementti on jonkin tietyn tekstin tai jonkin muun elementin jälkeen. Katselmoinnin jälkeen koodia korjattiin vielä näiden saatujen palautteiden avulla.

3.7 **Käyttöönotto ja ylläpitodokumentaatio**

Käyttöönotto tapahtui asteittain jo kehitysvaiheessa, käytettäessä testejä regressiotestauksen yhteydessä havaittiin koodissa vielä runsaasti kohtia, joita oli vielä kehitettävä. Esimerkiksi osa tarkastuksista ei toiminut halutulla tavalla ja näitä oli korjattava. Aikaisella käyttöönotolla saatiin koodin toimivuutta testattua monipuolisesti.

Haasteina käyttöliittymän testausautomaation kanssa ovat runsaat tekniset virheet, joihin testattava sovellus pysähtyy ennalta arvaamattomasti. Osa tilanteista johtuu siitä, että järjestelmä ei saa haettua tietoja taustajärjestelmistä ja osa on väliaikaisia teknisiä virheitä, joista pääsee jatkamaan eteenpäin kokeilemalla uudelleen toimintoa, jonka jälkeen teknisen virhe yleensä poistui.

Lisäksi käyttöönottovaiheen yhteydessä luotiin ylläpitäjille oma dokumentti (liite 3). Alkuun esitellään lyhyesti seuraavat tiedot: käytettävä käyttöjärjestelmä, käytettävä testausympäristö, työkalut, testitapausten

kuvausten sijainti testauksen hallintatyökalussa, testiskriptien sijainti, minkä tahon vastuulle kuuluu uusien testien luominen, testien ajaminen ja ylläpitäminen. Lisäksi miten raportointi tehdään. Testauksen kohteen esittely, tässä osiossa kerrotaan sovelluksesta perustiedot. Lisäksi, jos on jotain muuta erityistä esimerkiksi koskien testidataa, siitä voidaan mainita tässä kohdassa.

Testiajojen ajankohdassa määritellään, kuinka usein testiajot suoritetaan. Lisäksi annetaan tarkempi ajankohta, mikäli sellainen on tiedossa. Esimerkiksi tämän testausautomaation osalta dokumenttiin kirjattiin, että testit ajetaan joka kuukausi hyväksymistestauksen alettua. Raportointi osiossa kerrotaan, miten kyseinen testiajon tulos raportoidaan. Esimerkiksi tässä tapauksessa testausautomaatio kirjoittaa erilliseen tiedostoon tietyt tiedot jokaisen testin päätteeksi. Yhteyshenkilöt kirjataan liiketoiminnan ja toteuttavan yrityksen osalta. Näin tiedon kulku saadaan varmistettua molempiin suuntiin esimerkiksi muutosten osalta.

Saavutettavat hyödyt tuodaan esiin, jotta testausautomaatioratkaisun ylläpitäjä tietää, mitä ylläpidettävällä testausautomaatiolla saavutetaan ja miksi se on tärkeää. Tässä tapauksessa osioon kuvailtiin, että kyseessä on regressiotestauksessa käytettävät testitapaukset ja sovellus on kytköksissä useisiin muihin sovelluksiin. Kytkösten vuoksi sovellus on erityisen altis muutoksille, jotka eivät suoraan kosketa testauksen kohteena olevaa sovellusta. Lisäksi sovellus on luokiteltu liiketoiminnan kannalta kriittiseksi sovellukseksi, jonka vuoksi automatisoitu regressiotestaus on merkittävässä asemassa.

Katselmoinnit osiossa kerrotaan, kuinka usein kehitettyä testausautomaatiota katselmoidaan ja kenen toimesta. Katselmointien tarkoituksena on säilyttää testausautomaation kehittäjän ja tilaajan yhteisymmärrys ja tavoitteiden mukainen eteneminen. Töiden hyväksymisen osalta on hyvä olla selvää, kuka valmiit uudet kehitetyt ja/tai korjatut osiot hyväksyvät. Yleensä hyväksyjä on liiketoiminnan edustaja, koska liiketoiminnalla on hyvä käsitys sovelluksen prosessista.

Automatisoitavat testitapaukset kuvataan selkeästi. Tässä osiossa kerrotaan, kuinka monta kappaletta erilaisia testitapauksia on automatisoitu. Tämän lisäksi kerrotaan yksityiskohtainen testitapauksen eteneminen, jotta se on dokumentoitu selkeästi eikä sitä ole tarpeen yrittää lukea koodista. Lisäksi testitapauksiin liittyvät erityishuomiot voidaan nostaa tässä esiin. Testidatasta kerrotaan, minkälaisella datalla testit ajetaan. Luodaanko testidata aina ennen testien aloittamista vai käytetäänkö jotakin tiettyä datasettiä jatkuvasti. Tekniset lisätiedot ja työvälineet on hyvä avata. Tähän osioon voidaan kirjata yksityiskohtaisemmin tiedot toteutustavasta, ajoympäristöstä, testien ajamisesta, versionhallinnasta, testausautomaation vaatimista käyttöoikeuksista ja testiajojen jälkeisistä toimista. Testausautomaatiolle on hyvä päättää kontaktihenkilöt ja tehdä

ylläpitosuunnitelma. Dokumentaatiota voidaan laajentaa tarpeen mukaisesti siten, että se täyttää sille asetetut vaatimukset. Tästä syystä on tärkeää tietää kenelle ja miksi dokumentaatiota tehdään.

4 TULOKSET

Testausautomaation valmistuttua ja oltua käytössä 5 kuukautta, voidaan arvioida kuinka suuria saavutetut hyödyt ovat olleet. Yksi tapa arvioida hyötyä on laskea, kuinka paljon työaikaa säästetään. Tämä voidaan ilmaista henkilötyövuosina, esimerkiksi 0,5 henkilötyövuotta tarkoittaa käytännössä sitä, että yksi henkilö käyttää koko vuoden puolet työajastaan arvioitavan työtehtävän tekemiseen.

Manuaaliseen testaukseen kuluva aika on keskimäärin 20 minuuttia per testitapaus. Jos testaustulos on poikkeava odotettuun, kuluu selvittelyyn manuaalisesti 15 minuuttia aikaa lisää. 34 testin osalta kuluu manuaalisesti 680-1190 minuuttia. Tämä sisältää testitapausten kirjaamisen testauksenhallintavälineeseen, sopivan testidatan etsimisen, testien syöttämisen ja niiden tulosten tarkastamisen.

Automaattisesti ajettuihin testeihin kuluva aika on samaa testidataa käyttäen noin puoli minuuttia. Tähän sisältyy testiajon aloittaminen ja tulosten tarkistus. Vastaavasti mikäli testitulos poikkeaa odotetusta, selvittelyyn kuluu sama 15 minuuttia kuin manuaalisessa testauksessa. 34 testiä kestää automaattisesti testattuna 17-527 minuuttia. Testien ajaminen vie käytännössä enemmän aikaa, mutta se tapahtuu ilman henkilön työpanosta. Hyödyksi voidaan laskea manuaalisen ja automaattisen testauksen erotus, jolloin hyöty on yhden testiajon, joka sisältää 34 testiä osalta 663 minuuttia.

Henkilötyövuoden laskennassa käytetään tilaajan laskukaavaa, jonka mukaan tehokas työaika on 6 tuntia päivässä ja työpäiviä on vuodessa keskimäärin 220. 34 testin automatisointi tuo 0,1 henkilötyövuoden hyödyn, joka on 10 % yhden henkilön työajasta. Viikossa tämä on 0,5 päivää. Kuukaudessa 2,5 päivää. (Taulukko 1)

Taulukko 1. Henkilötyövuosien laskentakaava

Asia	Luku	Laskentakaava
Testien määrä	34	
Minuutit/testiajokerta	663	
Minuutit / vuosi	7956	663 min x 12 kk
Tunnit / vuosi	132,6	7956 min / 60 min
Työpäivät	22,1	132,6 h / 6 h
Henkilötyövuosi	0,1	22,1 pv / 220 pv

Kahdelta testaajalta kysyttiin heidän kokemuksiaan testausautomaation vaikutuksista ja vastausten perusteella testaajien kokemus on hyvin yksimielinen. Testaukseen käyttävän ajan vähentyminen lisäsi mahdollisuuksia tehdä tutkivaa testausta ja erikoisempien variaatioiden testausta. Li-

säksi testaustyöhön käytettävä aika väheni merkittävästi. Aluksi testausautomaatioon oli vaikea luottaa, koska se oli testaajille uusi muoto testauksesta. Lisäksi testaajat kertoivat, että testaus on heille ylimääräinen työtehtävä oman toimenkuvansa ulkopuolelta. Tämän perusteella voidaan katsoa, että testausautomaatiosta on saatu hyötyä lukujen lisäksi myös konkreettisella tasolla, vähentämällä sovelluksen testaukseen käytettävää aikaa.

5 JATKOKEHITYS

Testausautomaation toimintalogiikka olisi tärkeä dokumentoida kunnolla, jotta sen ylläpitäminen onnistuu jonkun toisen henkilön, kuin sen kehittäjän toimesta.

Kuten aikaisemmin todettu, testattava sovellus pysähtyy spontaanisti virheilmoitukseen, joista osa on todellisia virheitä järjestelmien toiminnassa ja osa väliaikaisia. Näiden osalta olisi tutkittava, miten Robot Framework tunnistaa tilanteen ilmenemisen ja toistaa yrittämänsä toiminnon vielä kerran, ennen kuin testi epäonnistuu tai vaihtoehtoisesti aloittaa saman testin kerran uudelleen. Käytännössä testausautomaation vikasietoisuutta olisi parannettava.

Lisäksi testausta voisi laajentaa siten, että hakemuksen lähettämisen yhteydessä lähetetään viesti ja tarkistetaan myöhemmässä vaiheessa, että lähtikö tapauksen mukainen viesti. Testin laajennuksen osalta on kuitenkin huomioitava sen kannattavuus ja se on pohdittava tarkoin, että tuleeko testitapauksista liian monimutkaisia. Testausautomaatiota tehdessä jo tuli havaintoa siitä, että koska testataan sovellusta, jolla on oma automaatio ja usean järjestelmän kanssa yhteen toimimista, on testausautomaatio haastavaa tehdä ja välttämättä sovellus ei aivan parhaalla tavalla sovellu siihen.

Testausautomaation toimintaa olisi hyvä kehittää myös siten, että se hakisi automaattisesti sopivat asiakkaat testauksen kohteeksi ja näin testidataa voisi vaihtaa paremmin. Tällä hetkellä vaihtaminen on manuaalista työtä ja ajan säästämiseksi testaus tehdään samalla datalla.

Myöhemmässä vaiheessa on tarkoitus toteuttaa myös testien ajaminen Jenkins -työkalun kautta, jotta raporttien tarkistaminen helpottuisi ja testien tulokset olisivat näkyvillä selkeästi. Liiketoiminnan edustukselle olisi järkevää mahdollistaa jokin muu näkymä tuloksiin kuin Jenkins. Jenkin on integroitavissa moniin testauksenhallintavälineisiin, jolloin testauksen kokonaisnäkymä on yhdessä sovelluksessa. Kuitenkaan tämän sovelluksen testausta ei ole mahdollista siirtää Linux-ympäristöön, koska sovellukseen pääsee sisälle vain sama käyttäjä, joka on kirjautunut omilla tunnuksillaan koneelle. Tämän osalta on selvitettävä virtuaalisen Windows-ympäristön hankintaa ja mitä se edellyttää, jotta testejä ei ole ajettava paikallisesti suoraan testaajan koneella.

6 YHTEENVETO

Testausautomaatiolla saavutettiin vuositasolla 22 työpäivän verran ajallista säästöä. Tämä mahdollisti sen, että testaajat voivat keskittyä monimutkaisempaan testaukseen ja sellaisten erikoisempien variaatioiden testaamiseen, joille ei ole aikaisemmin ollut aikaa. Mielestäni opinnäytetyöprojektina luotu testausautomaatio tuo todellista hyötyä, mutta testausautomaatiota ajavan on oltava hyvin perillä sovelluksen toiminnasta. Tämä järjestettiin perehdyttämällä ja luomalla kattavat ohjeet. Ylläpitoa luotu testausautomaatio vaatii jatkossakin.

Oma osaaminen parantui merkittävästi projektin aikana liittyen testausautomaatioon ja Robot Frameworkkiin. Opin ratkaisemaan erilaisia ongelmia ja kirjoittamaan tehokkaampaa koodia projektin edetessä. Opin käyttämään xpath-viittauksia hyväkseni monipuolisesti, regular expression on edelleen jatkossa vahvistettava osa-alue. Saavutin omasta mielestäni itselleni asettamani tavoitteet ja vahvistin osaamistani sellaisella alueella, joka tuntuu itselle kiinnostavalle ja jonka parissa haluan työskennellä jatkossakin. Olennainen havainto opinnäytetyöprosessissa oli se, että teoria ja toteutus vastaavat hyvin toisiaan. Testausautomaation kehittäminen on hidasta, aikaa vievää ja sen ylläpitoon kuluu runsaasti aikaa. Testausautomaatiota voisi parannella ja jatkokehittää loputtomasti. Välttämättä järkevää se ei kaikissa tilanteissa kuitenkaan ole.

Omassa testiautomaation kehittämisen prosessissa havaitsin kehitettävää dokumentoinnissa ja suunnitelmallisuudessa. Testiautomaatiokehittäjällä on yleensä useampia projekteja käynnissä samanaikaisesti, jolloin huolellinen dokumentointi on tärkeää myös oman työn helpottamisen näkökulmasta. Henkilöriskien näkökulmasta dokumentaatio on olennainen osa kehitystyötä, jotta toinen kehittäjä voi tarvittaessa jatkaa työskentelyä tuotoksen kanssa kehittäen tai ajaen testejä.

Suunnitelmallisuudessa olisi hyvä ottaa käyttöön yleisesti ohjelmistokehityksessä käytössä olevat menetelmät, koska testiautomaatio on ohjelmisto ja siihen pätee samat huomioitavat seikat. Luotavalle testiautomaatiolle tarvitaan selkeät vaatimukset tilaajalta, joka sisältää saavutettavat hyödyt sekä testauksen näkökulmat. Suunnitelmassa on huomioitava, miten testeistä saadaan mahdollisimman helposti ylläpidettävät. Esimerkiksi halutaanko testata vain toiminnallisuuksia vai tarkistaako testiautomaatio ohessa myös muita asioita. Työ tulee pilkkoa pienemmiksi kokonaisuuksiksi ja koota ne esimerkiksi backlogille. Etenemistä on tärkeä seurata ja tekemistä priorisoitava. Katselmoinnit ovat tärkeä osa kehitystä, jotta tuotoksesta saadaan mahdollisimman hyvä. Toisella testiautomaation kehittäjällä voi olla sellaisia kehitys- ja parannusehdotuksia, joita tekijällä ei ole tullut lainkaan mieleen. Lisäksi erilaiset ongelmakohdat on mahdollista ratkaista tehokkaammin hyödyntämällä katselmointeja.

Lopputuloksena saatiin luotua perusta sovelluksen testausautomaatiolle, jota on mahdollista jatkokehittää ja laajentaa. Projektin aikana oli priorisoitava tekemistä, jotta lopputuloksena olisi laadukas testausautomaatio. Tilaajan palautteen perusteella tämä lopputulos saavutettiin ja tilaaja sai toimivan testausautomaation kehikon, jota on nyt helppo lähteä jatkokehittämään.

Jatkokehitysehdotusten lisäksi olisi mielenkiintoista selvittää kuinka paljon henkilötyövuosina laskettua hyötyä voisi saavuttaa automatisoimalla jäljelle jääneet testitapaukset.

LÄHDELUETTELO

Bisht, S. (2013). Robot Framework Test Automation. Packt Publishing Ltd. Haettu 21.3.2019 osoitteesta https://ebookcentral-proquest-com.ezproxy.hamk.fi/lib/hamk-ebooks/reader.action?docID=1532018

Cinia. Testausautomaatio. (n.d). Haettu 20.9.2019 osoitteesta https://www.cinia.fi/palvelut/ohjelmistopalvelut/testausautomaatio.html

Dawson, M., Burell, D., Rahim E. & Brewster, S. (2010) Integrating Software Assurance into the Software Development Life Cycle. Haettu 5.10.2019 osoitteesta https://www.researchgate.net/publication/255965523_Integrating_Software_Assurance_into_the_Software_Development_Life_Cycle_SDLC

Ghahrai, A. (2018a). Testingexcellence. Haettu 15.7.2019 osoitteesta Testingexcellence: https://www.testingexcellence.com/why-do-we-test-what-is-the-purpose-of-software-testing/

Ghahrai, A. (2018b). Testingexcellence. Haettu 16.7.2019 osoitteesta Testingexcellence: https://www.testingexcellence.com/seven-principles-of-software-testing/

Fehrend, Kasper. (n.d) Manual Testing and Test Automation: 10 Considerations. Haettu 5.10.2019 osoitteesta https://www.leapwork.com/blog/manual-testing-and-test-automation-10-considerations

Homes, B. (2013). *Fundamentals of Software Testing.* Wiley: John Wiley & Sons, Incorporated.

ISTQB. (2012). International Software Testing Qualifications Board. Jatkotason sertifikaattisisältö Testauspäällikkö. Haettu 05.10.2019 osoitteesta http://www.fistb.fi/sites/fistb/files/liitteet/Advanced%20Syllabus%202012%20-%20TM%20Final.pdf

ISTQB. (2018). International Software Testing Qualifications Board. Perustason sertifikaattisisältö. Haettu 26.4.2019 osoitteesta http://www.fistb.fi/sites/fistb/files/liitteet/CTFL%202018%20Sertifikaattisisa%CC%88lto%CC%88%2020181010-1%20Valmis.pdf

ISTQB Certification. Pesticide Paradox Explained. (6.12.2009). Haettu 5.10.2019 osoitteesta http://istqbcertificationexpert.blogspot.com/2009/12/pesticide-paradox-explained.html

Kasurinen, J. P. (2013). Ohjelmistotestauksen käsikirja. Saarijärvi: Offset Oy.

Mili, A.;& Tchier, F. (2015). *Software Testing* : Concepts and Operations. John Wiley & Sons, Incorporated.

Mozilla. (22.3.2019). MDN web docs. Haettu 19.7.2019 osoitteesta MDN web docs: https://developer.mozilla.org/en-US/docs/Web/API/Document/querySelectorAll

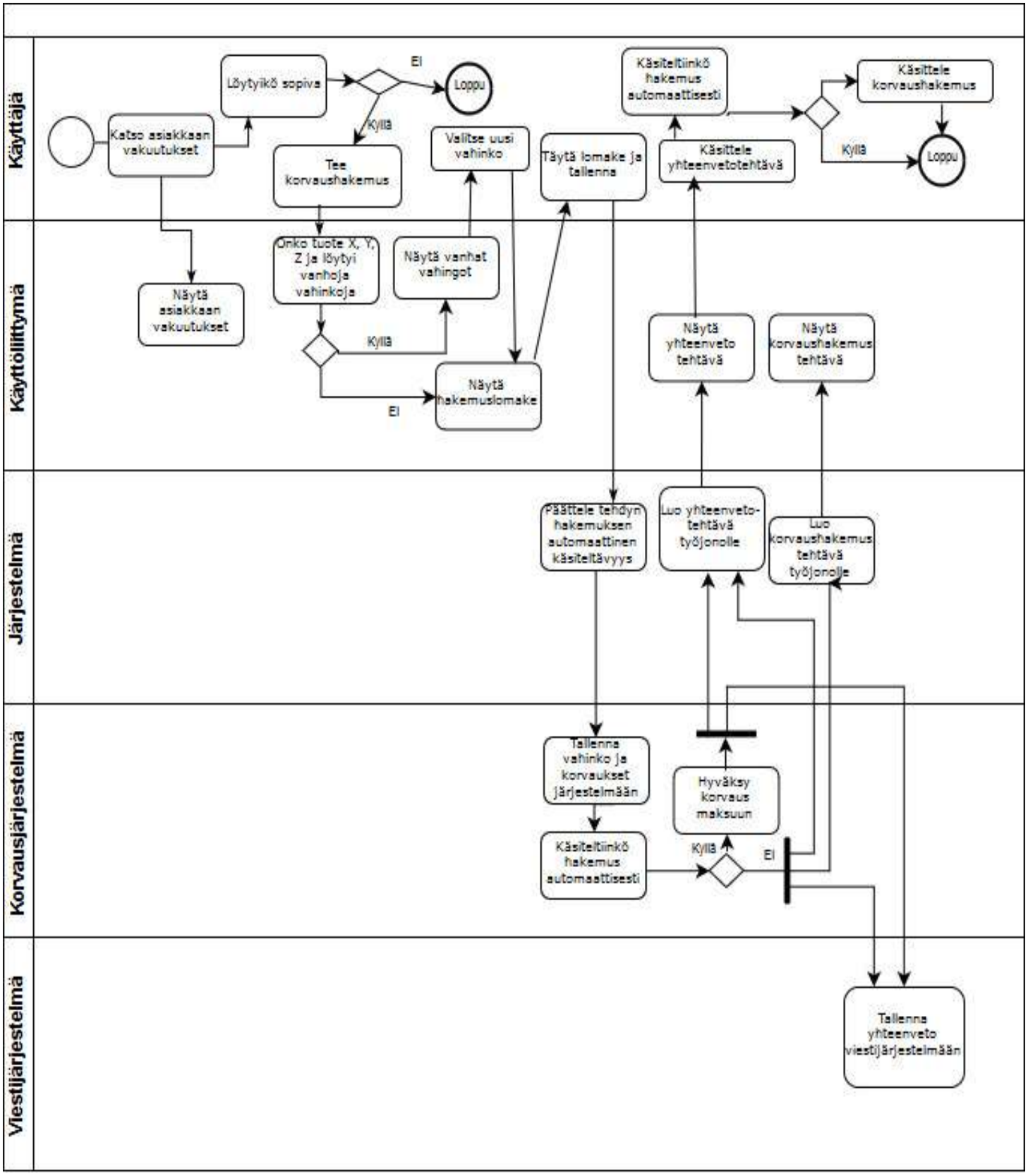
Myers, G.;Badgett, T.;& Sandler, C. (2011). *The Art of Software Testing*. John Wiley & Sons, Incorporated.

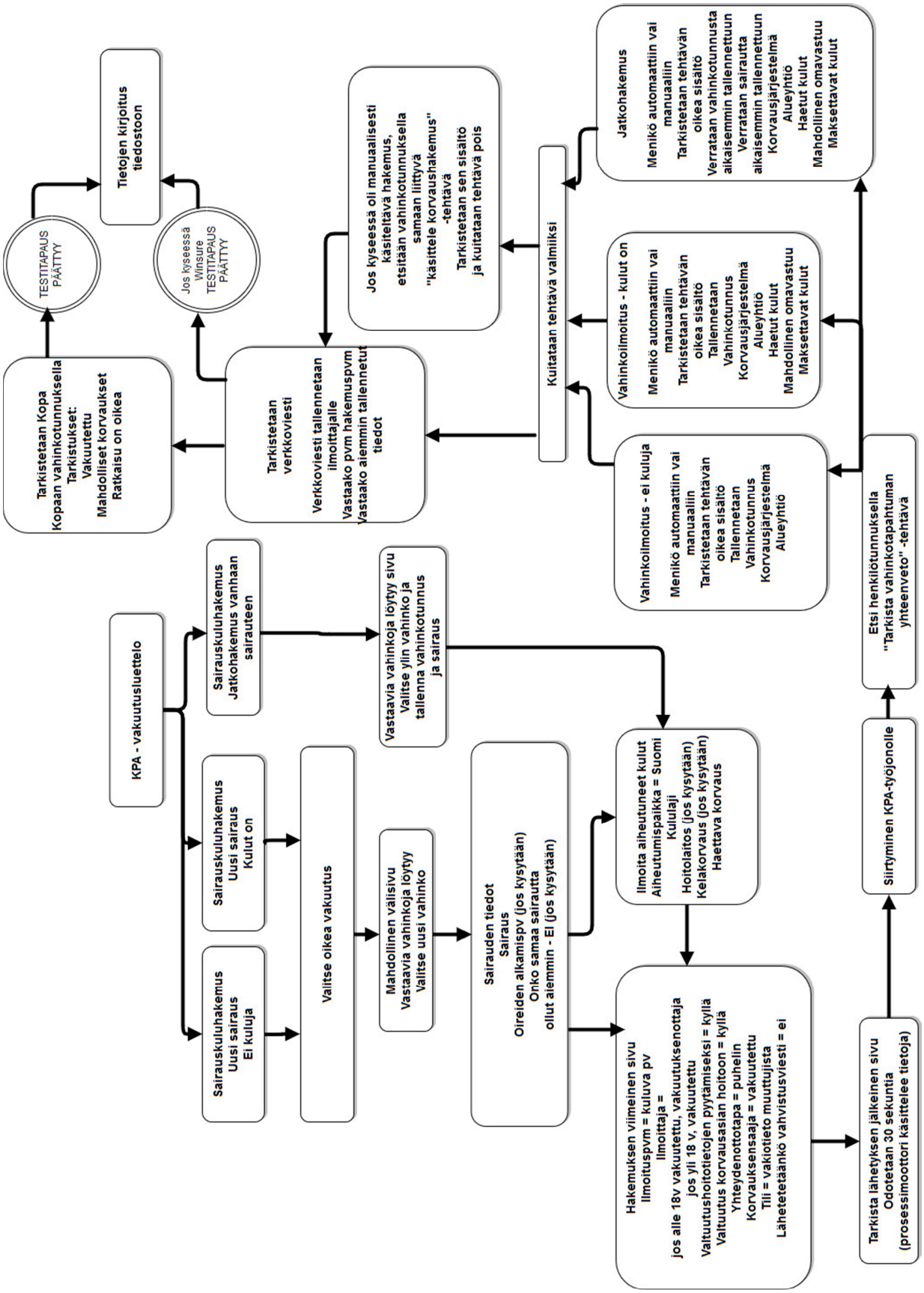
Robot Framework Foundation. (n.d.). Robot Framework Introduction. Haettu 26.4.2019 osoitteesta: https://robotframework.org/

Taina, J. (2007). Ohjelmistojen testaus. Haettu 25.7.2019 osoitteesta Ohjelmistojen testaus: https://www.cs.helsinki.fi/u/taina/ohte/s-2007/luennot/kalvot.pdf

Tuovinen, A.-P. (2013). Staattinen testaus. Haettu 25.7.2019 osoitteesta https://www.cs.helsinki.fi/u/aptuovin/testaus/Ohj_testaus_2013_5.pdf

TESTATTAVAN SOVELLUKSEN AKTIVITEETTIKAAVIO Liite 1




Tietojen kirjoitus tiedostoon
TESTITAPAUS PÄÄTTYY
Jos kyseessä Winsure TESTITAPAUS PÄÄTTYY
Tarkistetaan Kopa
Kopaan vahinkotunnuksella
Tarkistukset:
Vakuutettu
Mahdolliset korvaukset
Ratkaisu on oikea
Tarkistetaan verkkoviesti
Verkkoviesti tallennetaan ilmoittajalle
Vastaako pvm hakemuspvm
Vastaako aiemmin tallennetut tiedot
Jos kyseessä oli manuaalisesti käsiteltävä hakemus, etsitään vahinkotunnuksella samaan liittyvä "käsittele korvaushakemus" -tentävä
Tarkistetaan sen sisältö ja kuitataan tehtävä pois
Kuitataan tehtävä valmiiksi
Jatkohakemus
Menikö automaattiin vai manuaaliin
Tarkistetaan tehtävän oikea sisältö
Verrataan vahinkotunnusta aikaisemmin tallennettuun
Verrataan sairautta aikaisemmin tallennettuun
Korvausjärjestelmä
Alueyhtiö
Haetut kulut
Mahdollinen omavastuu
Maksettavat kulut
Vahinkoilmoitus - kulut on
Menikö automaattiin vai manuaaliin
Tarkistetaan tehtävän oikea sisältö
Tallennetaan Vahinkotunnus
Korvausjärjestelmä
Alueyhtiö
Haetut kulut
Mahdollinen omavastuu
Maksettavat kulut
Vahinkoilmoitus - ei kuluja
Menikö automaattiin vai manuaaliin
Tarkistetaan tehtävän oikea sisältö
Tallennetaan Vahinkotunnus
Korvausjärjestelmä
Alueyhtiö
Etsi henkilötunnuksella "Tarkista vahinkotapahtuman yhteenveto" -tehtävä
Siirtyminen KPA-työjonolle
Tarkista lähetyksen jälkeinen sivu
Odotetaan 30 sekuntia
(prosessimoottori käsittelee tietoja)
KPA - vakuutusluettelo
Sairauskuluhakemus
Uusi sairaus
Ei kuluja
Sairauskuluhakemus
Uusi sairaus
Kulut on
Sairauskuluhakemus
Jatkohakemus vanhaan sairauteen
Valitse oikea vakuutus
Mahdollinen välisivu
Vastaavia vahinkoja löytyy
Valitse uusi vahinko
Vastaavia vahinkoja löytyy sivu
Valitse ylin vahinko ja tallenna vahinkotunnus ja sairaus
Sairauden tiedot
Sairaus
Oireiden alkamispv (jos kysytään)
Onko samaa sairautta
ollut aiemmin - Ei (jos kysytään)
Ilmoita aiheutuneet kulut
Aiheutumispaikka = Suomi
Kululaji
Hoitolaitos (jos kysytään)
Kelakorvaus (jos kysytään)
Haettava korvaus
Hakemuksen viimeinen sivu
Ilmoituspvm = kuluva pv
Ilmoittaja =
jos alle 18v vakuutettu, vakuutuksenottaja
jos yli 18 v, vakuutettu
Valtuutushoitotietojen pyytämiseksi = kyllä
Valtuutus korvausasian hoitoon = kyllä
Yhteydenottotapa = puhelin
Korvauksensaaja = vakuutettu
Tili = vakiotieto muuttujista
Lähetetetäänkö vahvistusviesti = ei

Yleiset tiedot

Käyttöjärjestelmä	Testausympäristö	Työkalut	Testitapausten sijainti	Testiskriptien sijainti
Windows	Hyväksymistesti	Robot Framework		

Toteutuksen vastuuhenkilö:
Testiajojen vastuuhenkilö:
Ylläpitovastuuhenkilö:

Testauksen kohde
Sovelluksen perustiedot ja mahdolliset muut erityispiirteet.

Testien ajaminen
Kuinka usein ja miten testit ajetaan. Esimerkiksi kuukausittain hyväksymistestauksen alettua.

Raportointi
Miten ja kenelle raportoidaan.

Yhteyshenkilöt
Liiketoiminnan yhteyshenkilö:
Toteuttaja:

Saavutettavat hyödyt
Kuvataan luotavan testiautomaation hyödyt.

Katselmoinnit
Kuinka usein ja kenen toimesta katselmointeja tehdään.

Töiden hyväksyntä
Milloin, miten ja kenen toimesta uusien/korjattujen osioiden hyväksyntä tehdään.

Automatisoitavat testitapaukset
Kuvataan testitapaukset selkeästi, esimerkiksi steppeinä. Näin testin eteneminen on luettavissa dokumentaatiosta koodin sijaan.

Testidata
Millaista dataa käytetään, luodaanko testiajon aluksi uutta vai käytetäänkö olemassa olevaa. Mistä ja miten data saadaan käyttöön.

Tekniset lisätiedot ja työvälineet
Tähän voi avata yksityiskohtaisemmin tiedot toteutustavasta, ajoympäristöstä, testien ajamisesta, version hallinnasta, tarvittavista käyttöoikeuksista ja testiajojen jälkeisistä toimista.

Ylläpitosuunnitelma
Miten testausautomaatiota ylläpidetään kehitysvaiheen aikana ja jälkeen.