Haaga-Helia
ammattikorkeakoulu Oy

ERP- järjestelmän testiautomaation suunnittelu

Regression Suite Automation Tool ja Robot Framework testiautomaatiotyökalujen soveltuvuusanalyysi

Silva Sivén

Opinnäytetyö
Tietojenkäsittelyn koulutusohjelma
2021

Tiivistelmä

Haaga-Helia
ammattikorkeakoulu Oy

Tekijä(t)
Silva Sivén

Koulutusohjelma
Tietojenkäsittelyn koulutusohjelma

Opinnäytetyön nimi
ERP-järjestelmän testiautomaation suunnittelu
Regression Suite Automation Tool ja Robot Framework testiautomaatiotyökalujen soveltuvuusanalyysi

Sivu- ja liitesivumäärä
44

Toiminnallisen opinnäytetyön tarkoituksena oli analysoida esivalittuja RSAT ja Robot Framework-testiautomaatiotyökaluja. Työkalut liittyivät toimeksiantajan toiminnanohjausjärjestelmän regressiotestien automatisointiin.

Tavoitteena oli kerätä testiautomaatiotyökalun valinnan päätöksenteon tueksi tietoa, miten käyttöympäristöön valitut testiautomaatiotyökalut kohdennetaan tehokkaasti. Lisäksi haluttiin tietää työkalujen valinnan vaikutus hyväksymistestausta tekevien liiketoiminnan edustajien ajankäyttöön ja työtehtävien priorisointiin.

Opinnäytetyön teoreettisessa viitekehyksessä käsitellään yleisellä tasolla ohjelmistotestauksen prosessia ja sen eri vaiheiden testauksen automatisointia. Analysointia varten teoriatietoa hankittiin automatisointityökalujen valintaprosessista sekä valituista työkaluista. Teoriatiedossa esitetty testiautomaatiotyökalun vertailumatriisi tuli toimimaan analyysini johtopäätösten ja suositusten tekemiseen tukevana aputyövälineenä.

Automatisointityökalujen analyysiprosessi aloitettiin helmikuun 2021 loppupuolella ja prosessi viimeisteltiin toukokuussa 2021. Kuvaan toteuttamaani analyysiprosessia, jossa kokosin dataa ja havaintoja RSAT ja Robot Framework automatisointityökaluilla toteutetuista proof of concept testitapauksista. Tehtäviini kului toteuttaa Robot Frameworkilla testitapauksen, jonka taustalla on organisaation tarve varmistaa liiketoiminnalle kriittiset testitapaukset, joissa toteutuu integraatio järjestelmien välillä.

Opinnäytetyön tuotoksena muodostui soveltuvuusanalyysi sekä Robot Frameworkilla toteutetut resurssi- ja avainsanakirjastot toimeksiantajan käyttöön. Analyysiin valitut automatisointityökalut edustavat kahta eri testiautomatisoinnin viitekehystä. Asetelma osaltaan muodosti ennakko-odotuksen automatisointityökalujen testien toteuttamiseen, suorittamiseen ja ylläpitoon. Johtopäätöksenä testitapauksen sisältö ja sen prioriteetit ratkaisevat tarvetason testiautomaatiotyökalulle ja ohjaavat sen valintaa. Toteuttamani valintamatriisi auttaa rajaamaan työkaluja, mutta testitapauksen asettamat rajat/kyvyt työkalulle ovat tapauskohtaisia.

Totean, että Robot Framework on vertailun perusteella testauksen ja ohjelmiston hyvää laatua tukeva automatisointityökalu. Opinnäytetyön toimeksiantajan priorisoitavat testitapaukset liittyvät eri järjestelmien integraatioihin ja RSAT:lla ei voida toteuttaa liiketoiminnalle kriittisiä testitapauksia.

Asiasanat
Robot Framework, RSAT, testiautomaatio

Käsitteet ja lyhenteet:

ERP	Enterprise Resource Planning, toiminnanohjausjärjestelmä, joka on liiketoimintaprosessien hallintaohjelmisto.
Bugi	Ohjelmointivirhe
Python	Tulkattava, interaktiivinen ja olioihin-pohjautuva ohjelmointikieli
Azure DevOps	Microsoftin tarjoama kehitysympäristö, jossa noudatetaan Devops kehitys- ja tuotantomallia. Mallissa tavoitteena on automatisoida ohjelmistokehitykseen, testaamiseen ja ylläpitoon liittyviä toimintoja.
Selenium	Web-sovellusten testaamiseen käytettävä palvelu
CI	Continuous integration eli jatkuva integraatio on ohjelmistokehityksen toimintatapa, jossa uusi koodi liitetään osaksi valmista tuotetta jopa useita kertoja päivässä
CRUD	Muodostuu sanoista create, read, update and delete. Tiedontallentamiseen käytettävät perusoperaatiot.

Sisällys

1 Johdanto

Ohjelmistokehityksessä ketterien menetelmien suosio on osaltaan vaikuttanut testiautomaation kasvavaan hyödyntämiseen, jotta jatkuvista kehitysiteraatioista selvitään sujuvasti ja tehokkaasti. Ketterässä ohjelmistokehityksessä useiden iteraatioiden aikana syntyy useita uusia ominaisuuksia, minkä myötä ohjelmisto muuttuu kompleksisemmaksi. Uudet ominaisuudet lisäävät riskiä, että ne alkavat herkästi rikkomaan aiemmin toteutettuja ominaisuuksia. Testiautomaatio on keino vastata kasvaneeseen testaustarpeeseen, johon ei voida vastata pelkällä manuaalitestauksella. Ketterässä menetelmässä iteraatioiden myötä hyväksytään liiketoiminnan vaatimusten muuttuminen ja sitä kautta muutokset kehitettävän sovelluksen koodiin. Nämä muutokset edellyttävät yleensä myös muutoksia automaattiseen testikoodiin. (Gupta 2016, luku 8.)

ERP-järjestelmän testiautomaation suunnittelu- opinnäytetyöni toimeksiantajan tuleva toiminnanohjausjärjestelmä (ERP) on konkreettisesti vaiheessa, jossa testiautomaation hyödyntämisen suunnittelu on ajankohtaista. Tilaajaorganisaation hankkeen aikana on tarve toteuttaa useita testauksia ja regressiokierroksia sekä vahvistaa testauskapasiteettia /testauskykyä. Tähän mennessä asiakkaan hyväksymistestaukset ovat toteuttaneet liiketoiminnan ammattilaiset, joiden ajankäyttö testauksiin on rajallinen.

Tulevaksi ERP-sovellukseksi on valittu Microsoft Dynamics 365 for Finance and Operations, jonka myötä valikoitui toiseksi testiautomatisoinnin menetelmäksi Regresession Suite Automation Tool (RSAT) -työkalu. RSAT on Microsoftin tarjoama testiautomaatiotyökalu ympäristöpäivitysten yhteydessä tehtäviin hyväksymistestauksiin. RSAT- automatisointityökalun ja sen sisään rakennetun nauhoitustyökalun käyttömahdollisuuksia halutaan tutkia tarkemmin.

Opinnäytetyön toimeksiantajalla on ollut käytössään Robot Framework- testiautomaatiotyökalu eri projekteissa osana laajempaa ohjelmistohankekokonaisuutta. Työkalulla on automatisoitu testitapauksia, jotka on liitetty osaksi jatkuvaa integraatioputkea varmistamaan kokonaisuuden toimivuutta osana laadunvarmistusprosessia. Robot Framework on avoimen lähdekoodin testiautomaatioviitekehys, jolla laaditaan avainsanapohjaisia testiskriptejä etenkin hyväksymistestaukseen.

Opinnäytetyöni tavoitteena on hankkia tietoa, miten valitut testiautomaatiotyökalut kannattaa kohdentaa tehokkaasti työkalun soveltuvuuden mukaan. Samalla tavoitellaan liiketoiminnan edustajien ajankäytön tehostamista, jotta asiantuntijoiden aikaa vapautuu projektin vaativampiin osioihin.

Opinnäytetyön tehtävänä on analysoida valittuja testiautomaatiotyökaluja varmistaakseen niiden tehokas käyttö osana hankkeen testausprosessia. Selvitettävänä on RSAT- ja Robot Framework -testausautomaatiotyökalujen erot, heikkoudet ja vahvuudet sekä soveltuvuudet eri testitilanteiden suorittamisessa. Prosessissa rakentuu soveltuvuusanalyysi, jota voi hyödyntää työkalun valinnassa sekä arvioitaessa sen sopivuutta ja tehokkuutta huomioiden testien ylläpidettävyys.

Automatisointityökalujen analyysi toteutetaan keräämällä dataa ja havaintoja proof of concept testitapauksista. RSAT:lla valitut testitapaukset on testattavan järjestelmän toimittaja automatisoinut ennen opinnäytetyöskentelyni alkamista. Opinnäytetyössä toteutan Robot Frameworkilla testitapauksen, jonka taustalla on organisaation tarve varmistaa liiketoiminnalle kriittiset testitapaukset, joissa toteutuu integraatio järjestelmien välillä.

Opinnäytetyön toimeksiantaja ei ole asettanut erityisiä vaatimuksia testiautomaatiotyökalulle, mutta ERP-sovelluksen ratkaisuja on kustomoitu vastaamaan organisaation tarpeita. Toiminnot sisältävät integraatioita, mikä on kriittistä liiketoiminnalle. Näihin toimintoihin liittyvien vaatimusten testaaminen on etenkin asiakkaan näkökulmasta prioriteettilistalla korkealla ja vaikuttaa testiautomaatiotyökalun vaatimuksiin. Työkalun on sovelluttava myös organisaation käyttämiin menetelmiin ja testien hallintatyökaluihin. Opinnäytetyö ei tee lopullista valintaa työkalujen käyttämiseen.

2 Ohjelmistotestaus

Ohjelmistotestaus kuuluu jokaisen ohjelmiston kehityksen elinkaareen. Ohjelmistotuotannon testaamista voidaan kuvata joukkoina tehtäviä tai aktiviteetteja, jotka ovat ennalta suunniteltu ja toteutettu systemaattisesti. Testaus on prosessi, jonka tavoitteena on testata koodia ja sen toimivuutta. Tavoitteena on myös se, että testaus paljastaa ohjelman vaatimus-, suunnittelu- ja koodausvirheet. Testauksen tulee sisältää matalan ja korkean tason testit. Matalan tason testeillä varmistetaan, että pienempi koodiosio on suoritettu oikein. Korkean tason testit validoivat tärkeimmät järjestelmätoiminnot asiakkaan vaatimusten mukaisesti. Testaamisella pyritään hakemaan hyötyjä ohjelmistoon tai projektiin, jotta voidaan parantaa laatua sekä vähentää julkaisun jälkeisiä ylläpitokuluja. (Agarwal, Gupta & Tayal 2009, 161–162.)

Ohjelman testausta varten on oltava kuvaus sen odotetusta käyttäytymisestä ja menetelmä sen määrittämiseksi, onko havaittu käyttäytyminen odotetun mukainen. Mekanismia, joka vastaa testien tulosten validoinnista nimitetään testioraakkeliksi. Testioraakkelin tekemät päätökset toimivat palautteena testin läpäisystä (Agarwal ym. 2009, 164–165.)

2.1 Testauksen periaatteet

Kasurisen (2013, 48–49) mukaan testien laatimisessa noudatetaan seuraavia testiä ohjaavia periaatteita:

1. Ohjelmistojen testauksen tarkoituksena on havaita ohjelmistoviat. Testaustekniikoiden avulla voidaan vähentää havaitsemattomien vikojen todennäköisyyttä. Kuitenkaan useat läpäisseet testikierroksetkaan eivät ole tae siitä, että ohjelmisto on täysin virheetön.

2. Kattava testaus on mahdoton. Kattavalla testauksella tarkoitetaan testausprosessia, jossa eri ennakkoehdot suoritetaan yhdessä kaikkien mahdollisten syötteiden ja niiden yhdistelmien kanssa. Kattava testaus on myös paljon resursseja vievää, joten riskianalyysin ja liiketoiminnan priorisoinnin kautta saadaan päätöksiä riittävästä testausmäärästä.

3. Aikainen testaus. Staattinen ja dynaaminen testaus on aloitettava kehityksen varhaisessa vaiheessa, jotta ohjelmiston viat ovat helpommin tunnistettavissa ja korjattavissa. Vikojen korjaaminen muuttuu haastavammaksi ja kalliimmaksi mitä myöhäisemmässä ohjelmiston elinkaaren vaiheessa virhe havaitaan.

4. Vikojen kasaantuminen ohjelmiston testauksessa viittaa virheisiin tai toimintahäiriöihin, jotka toistuvat yhdessä osaa luotua toimintoa. Vikojen kasaantuminen voi johtua ohjelmistossa ja muissa toiminnoissa tapahtuvista muutoksista, jotka aiheuttavat regressiovirheitä. Näin ollen testausta tulisikin painottaa näille osa-alueille.

5. Hyönteismyrkkyparadoksi on tilanne, jossa ohjelmistojen testaamisessa toistetaan samoja testitapauksia, kunnes uusia vikoja ei havaita. Testitapauksia ja -tietoja tulee päivittää säännöllisesti, jotta testit auttavat löytämään koodausvirheitä.

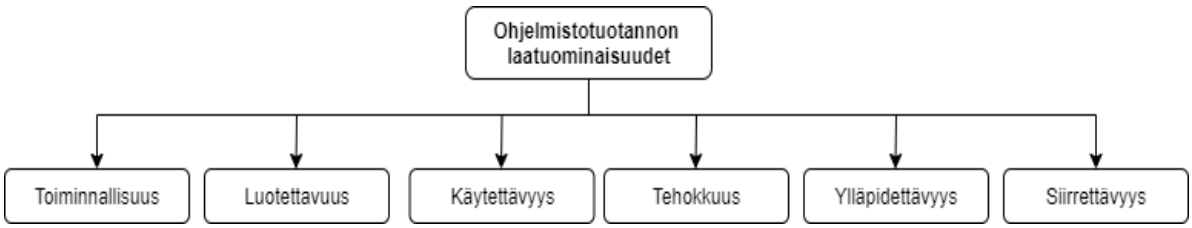
6. Testaus on tilanneriippuvaista. Tämä tarkoittaa, että valittava testausmenetelmää tulee soveltaa testattavaan sovellukseen.

7. Virheiden puuttuminen on harhaa. Kehitettävän ohjelmiston tulee vastata myös yrityksen vaatimuksiin ja tarpeisiin, joten testauksen osoittamien virheiden ja niiden korjaaminen eivät ole avain onnistuneeseen järjestelmään. Laadittavat testit tulee olla seurattavissa asiakkaan vaatimuksiin. Näin havaitaan mahdolliset tilanteet, jotka eivät vastaa asiakkaan asettamia vaatimuksia.

2.2 Ohjelmistotestauksen laatuominaisuudet

Testaus määritellään monessa ohjelmistotuotannon mallissa laadunhallinnan ja laadunvarmennuksen toimintona.

Laatuominaisuudet toimivat ohjelmistokehitystä ohjaavina tavoitteina, määrittävät kehityskriteerit ja toimivat laadun mittareina. Ohjelmiston laatuvaatimukset voidaan jakaa sisäisiin ja ulkoisiin vaatimuksiin. Ulkoiset vaatimukset ovat käyttäjälle näkyvät käytettävyyteen liittyvät ominaisuudet. Sisäiset ominaisuudet liittyvät itse ohjelmiston rakenteeseen. Lisäksi laatuominaisuudet voivat liittyä ohjelmiston prosesseihin, joissa jokaista työvaihetta voidaan tarkastella eri näkökulmista. Näkökulmien tarkasteluun on kehitetty erilaisia tekniikoita ja työkaluja. Itse tuotteelle voidaan asettaa erilaisia vaatimuksia. (Agarwal ym. 2009, 90.)



Kuva 1 Ohjelmistotuotannon laatuominaisuudet (mukaillen Agarwal ym. 2009, 90.)

Koska laatuominaisuuksia on kirjava määrä, on nostettu kuusi perustason laatuominaisuutta, joihin tulisi pyrkiä (Agarwal ym. 2009, 90–91.):

Toiminnallisuus: Toiminnallisuudella viitataan joukkoon eri toimintoja, joita ohjelmisto tarjoaa. Toimintoja voidaan tarkastella suhteessa niiden sopivuutta asetettuihin vaatimuksiin sekä arvioida ohjelman toimintojen soveltuvuutta ja tarkkuutta. Tähän liittyy myös se kuinka turvallisesti ohjelmisto käsittelee tietoihin liittyviä tapahtumia.

Luotettavuus: Luotettavuus kuvaa ohjelmiston toimintakykyä suoriutua määritetyissä olosuhteissa ja virheiden käsittelyssä.

Käytettävyys: Käytettävyys viittaa ohjelmiston toimintojen helppokäyttöisyyteen ja ymmärrettävyyteen. Se kuvaa kuinka paljon vaivaa eri tasoisten käyttäjien täytyy käyttää oppiakseen ja sisäistääkseen eri toiminnot.

Tehokkuus: Tehokkuus kuvaan ohjelmiston suorituskykyyn liittyviä ominaisuuksia ja vasteaikaa. Se liittyy myös ohjelmiston arkkitehtuuriin ja koodauskäytäntöihin.

Ylläpidettävyys: Ylläpidettävyys kuvaa ohjelmiston muutettavuutta ja laajennettavuutta sekä vakautta, kun muutoksia toteutetaan. Se liittyy ohjelmiston koodin luettavuuteen ja mahdolliseen kompleksisuuteen. Se osoittaa miten ohjelmiston vikoja tunnistetaan ja korjataan.

Siirrettävyys: Siirrettävyys liittyy ohjelmiston kykyyn mukautua erilaisiin määriteltyihin ympäristöihin. Kyseessä on myös se, kuinka helposti järjestelmä voidaan asentaa.

2.3 Testausprosessi

Testausprosessi tukee ohjelmistotestausta sekä ohjaa testaajien työskentelyä. Se voidaan jakaa viiteen eri osioon, jotka kattavat testauksen näkökohdat. (Hambling, Morgan & Samaroo, luku 1.7.):

1. Strategia ja valvonta.
Strategiassa määritetään, mitä ja miten testataan ja kenen toimesta. Samalla myös määritetään, milloin testi on testannut ominaisuudet halutulla tavalla. Tämä vaatii rinnalle samanaikaisesti valvontaa, jonka perusteella arvioidaan testauksen etenemistä sekä tarvittaessa tarkastetaan ja päivitetään suunnitelmaa. (Hambling ym. 2010, luku 1.7.)

2. Analyysi ja suunnittelu.
Prosessissa laaditaan yksityiskohtaiset suunnitelmat, jotta saadaan katettua eri testiolosuhteet mahdollisimman pieniin määriin testitapauksia. Analysoimalla tunnistetaan testiolosuhteet ja tarvittava testidata. Sen jälkeen arvioidaan ovatko asetetut vaatimukset ja järjestelmä testattavia sekä priorisoidaan testitapaukset. (Hambling ym. 2010, luku 1.7.)

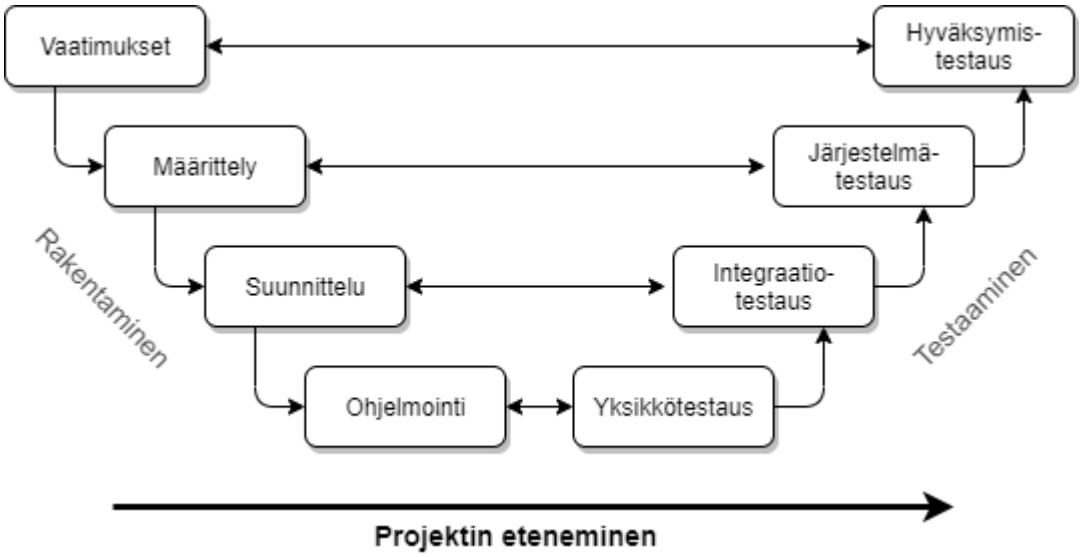
3. Toteutus
Testausprosessin näkyvin osa on testin suorittaminen, mikä voidaan toteuttaa eri tavoin. Toteutus sisältää testitapausten kehittämisen tai testiskriptien laatimisen. Luodut testit kootaan testijoukoiksi (test suite), jossa eri testitapaukset ajetaan tai suoritetaan peräkkäin. Toteutukseen kuuluu myös ajettujen testitulosten, käytettyjen ohjelmistojen ja datan dokumentointia. (Hambling ym. 2010, luku 1.7.)

4. Raportointi ja testien asetetun laatutason arviointi.
Tässä vaiheessa tarkistetaan ja arvioidaan, onko testauksen toteutuksessa päästy suunnitteluvaiheessa asetettuihin laatutason vaatimuksiin. Ajetuista testeistä laaditaan raportti, jossa esitetään kuinka monta ja minkä tasoista vikaa on havaittu kehitetyssä ominaisuudessa. (Hambling ym. 2010, luku 1.7.)

5. Testin lopetustoiminnot.
Testausprosessi on saatu onnistuneesti päätösvaiheeseen ja keskitytään dokumentointiin. Ohjelmiston elinkaaren aikana toteutetut testit kuvataan kuten myös siinä havaitut bugit ja virheet. (Hambling ym. 2010, luku 1.7.)

2.4 Ohjelmistotestauksen tasot

Suosittu malli testauksen vaiheita kuvaamaan on V-malli (kuva 2), jossa ohjelmiston testausprosessi liitetään osaksi ohjelmiston suunnitteluprosessia. Jokaista suunnittelu- ja toteutusvaihetta vastaa testausvaihe. Ohjelmistotestausta voidaan toteuttaa eri vaiheissa tai tasoissa, jossa testattava kohde käy läpi kolme tasoa: yksikkö-, integrointi ja järjestelmätestaus. Hyväksymistestaus voidaan katsoa osaksi testauksen tasoja v-mallin mukaan. Regressiotestaus liittyy myös eri testaustasoihin. (Hambling ym. 2010, luku 2.3.)



Kuva 2 V-malli

Monet yritykset ovat siirtyneet suositusta V-mallista käyttämään Ketterän kehityksen menetelmää. Se on todettu tehokkaaksi menetelmäksi sekä uusien tuotteiden kehittämisessä että parannus- ja päivitysprojekteissa. Ketterä kehitys varautuu projektin aikana tapahtuviin muutoksiin määritetyllä työprosessilla. Projekti jaetaan useisiin iteraatioihin, joissa toteutuksen lisäksi toistuvat tietyt suunnittelu- ja tarkistusvaiheet. Ketterän kehityksen menetelmässä on välttämätöntä suorittaa testejä kaikilla samoilla tasoilla kuin perinteisessä V-mallissa. Testitasot perustuvat erilaisiin teknisiin vaatimuksiin ja tavoitteisiin:

Yksikkötestauksessa testi kohdistuu yksittäiseen komponenttiin, jotta varmistetaan sen virheetön toimivuus itsenäisesti ja vastaavuus suunnitteludokumetin kuvaukseen. Tämän avulla löydetään mahdolliset virheet helpommin, kun sitä vasten voidaan toteuttaa kohdennettuja testauksia. Testitapaus voidaan laatia ennen tai jälkeen yksittäisen funktion tai luokan toteuttamisen. Etukäteen suunniteltua testitapausta kutsutaan myös testauslähtöiseksi ohjelmoinniksi. (Agarwal ym. 2009, 165–166.)

Integrointitestaus testaa ohjelman osien välistä vuorovaikutusta, joka liittyy rajapintoihin tai riippuvuuksiin eri yksiköiden välillä. Yksikkötestitauksen moduulit yhdistetään toimiviksi osajärjestelmiksi, jotka testaavat voidaanko moduulit integroida oikein ja mielekkäällä tavalla. Integrointitestausta voidaan lähestyä joko järjestelmähierarkian huipulta tai matalammalta tasolta. Lähestymistavan suunnan mukaan matalalta tasolta havaitaan helpommin matalan tason ongelmia ja ylhäältä saadaan parempi yleiskuva järjestelmästä aikaisessa vaiheessa. Lähestymistavat paljastavat siten mahdolliset puutteet järjestelmän toiminnoissa. (Agarwal ym. 2009, 167–168.)

Järjestelmätestauksessa varmistetaan, että ohjelmisto on vaatimusmäärittelyn mukainen. Järjestelmää testataan toiminnallisena kokonaisuutena testiympäristössä. Testauksen aikana mahdollisia muutoksia järjestelmään voidaan vielä toteuttaa. (Agarwal ym. 2009, 172.)

Hyväksyntätestauksen toteuttaa ohjelmiston asiakkaat ja käyttäjät. He varmistavat vastaako ohjelma heidän tarpeitaan ja vaatimuksiaan; ja antavat hyväksynnän kehittäjille ohjelmistosta. Hyväksymistestaus voidaan jakaa alfa- ja betatestaukseen. Alfatestauksessa käyttäjät toteuttavat valmistajan kanssa testauksen tarkistaen yleisimmät toiminnot, jota ohjelmalla tullaan suorittamaan. Betatestauksessa asiakkaat testaavat omassa työympäristössä julkaistavaa ohjelmaversiota. (Agarwal ym. 2009, 172.)

2.5 Testauksen lähestymistavat

Staattisessa testauksessa tarkastellaan ja analysoidaan järjestelmää suorittamatta sitä. Yleisin staattisen testauksen muoto on katselmointi, jossa pyritään löytämään vikoja ja epäjohdonmukaisuuksia vaatimus- ja suunnittelumääritelmistä, dokumenteista tai ohjelmakoodeista. (Hambling ym. 2010, luku 2.4.)

Dynaamisella testauksessa seurataan järjestelmässä tapahtuvia reaktioita annettuihin syötteisiin. Järjestelmän käytön aikana käytetään eri testauksen muotoja:

Musta laatikko -testit: Testauksessa ohjelmalle annetaan erilaisia syötteitä ja katsotaan mitä ohjelma tekee. Testitapauksissa kuvataan annettavat syötteet ja odotetut lopputulokset. Musta laatikko -testiä voi käyttää kaikissa työvaiheissa, joissa on käytettävissä toiminnallisuutta suorittava laite. Annetuilla syötteillä pystytään todentamaan, että toiminnallisuus toimii oikein. Koska käytettävät testitapaukset ovat hyvin yksiselitteisiä, ne ovat myös hyviä automatisoitavia testikohteita. (Kasurinen 2013, 51–52.)

Lasilaatikko testaus: Käytetään myös nimeä rakenteellinen testaus, jossa tarkastellaan järjestelmän sisällä tapahtuvia muutoksia, miten järjestelmä reagoi annettuihin syötteisiin. Testauksella järjestelmää voidaan tarkastella tarkemmin lähdekooditasolla, jolloin testaajalla tulee olla osaamista järjestelmän teknisistä yksityiskohdista ja logiikasta. (Hambling ym. 2010, luku 2.4.)

Harmaa laatikko testaus: Testauksessa pyritään varmistamaan, että vaatimukset ja järjestelmän sisäinen lähdekoodi on tarkistettu (Hambling ym. 2010, luku 2.4).

2.6 Ohjelmistotestauksen tyypit

Testaustasoilla on erilaisia testaustavoitteita, joita eri testityypeillä tavoitellaan. Testityypit voidaan jakaa karkeasti toiminnalliseen ja ei-toiminnalliseen testaukseen.

Toiminnallinen testaus on usein pääasiallinen testausmuoto. Testaus kohdistuu järjestelmän eri toimintoihin, jolloin varmistetaan määritettyjen vaatimuksen toteutumista ohjelmassa. Toiminnallisen testauksen toteuttaminen on usein raskasta ja hidasta. Sen alle voidaan luokitella (Hambling ym. 2010, luku 2.4.):

Savutestaus, jolla katetaan suurin osa ohjelmiston tärkeimmistä toiminnallisuuksista. Sen tarkoituksena on testata, onko ohjelmiston uusi versio (esim. uusi build) riittävän toimiva testattavaksi kunnolla. Tässä käytetyt testitapaukset ovat yleensä osajoukko regressiotestauksessa käytetyistä. (Hambling ym. 2010, luku 2.4.)

Regressiotestaus on testaustyötä, joka varmistaa, että ohjelmistoon tai ympäristöön toteutetut muutokset eivät ole rikkoneet aiemmin toimivaa toiminnallisuutta ja järjestelmä toimii edelleen vaatimusten mukaisesti. Regressiotestauksen näkökulmasta järjestelmän virheet sijoittuvat tavallisesti uusiin komponentteihin tai toimintoihin, jotka käyttävät niitä. Regressiotestauksen avulla voidaan varmistaa, että eri versioista tuodut ominaisuudet toimivat yhdessä. (Kasurinen 2013, 54.)

Järjestelmää testataan ei-toiminnallisten vaatimusten kuten käytettävyyden, suorituskyvyn, turvallisuuden ja yhteensopivuuden suhteen. Ei-toiminnallisia testityyppejä ovat luotettavuustestit, kuormittavuustestit, käytettävyystestit ja asennustestit. (Hambling ym. 2010, luku 2.4.)

2.7 Testauksen kattavuus

Ohjelmistojen onnistuneessa toimituksessa on riittävällä testauksella ratkaiseva rooli. Testauksen kattavuuden arvioiminen on kuitenkin haastavaa. Testien yhteydessä käytettävä kattavuusraportti antaa tietoa ohjelmiston osista, joissa testausta käytetään. Raportista saa tietoa myös sovelluksessa tai verkkosivustolla suoritetuista testeistä. Testikattavuus kuvaa sen, kuinka suuri osa testattavan ohjelmiston vaatimuksista tulee testattua suorittamalla testitapaukset. Testien tehokkuutta ja kattavuutta voidaan selvittää kattavuustyökaluilla. Tällaiset analysaattorit mittaavat testikattavuutta erilaisilla kattavuusmitoilla. (Hambling ym. 2010, luku 2.4.)

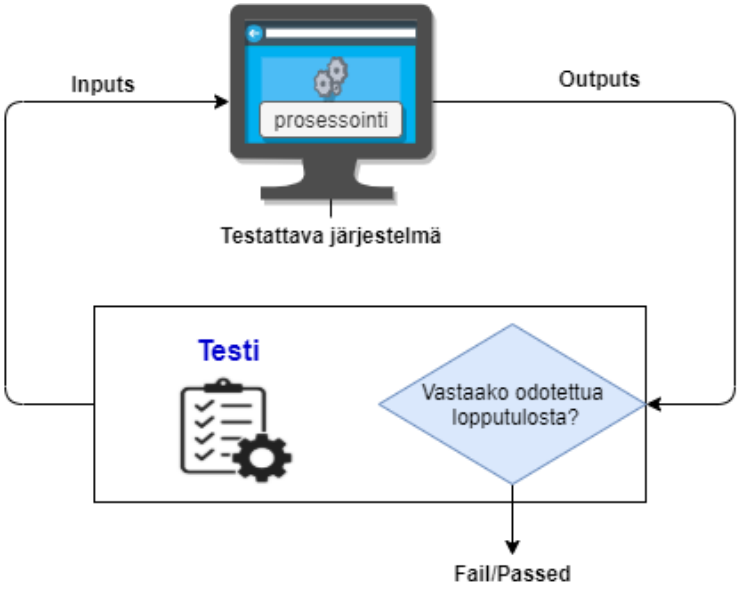
Yleisimmin käytössä olevia kattavuusmittoja (Myers, Badgett & Sandler 2012, 44–46):

Lausekattavuus	Tarkistaa, mitä kirjoitetun koodin odotetaan tekevän ja jättävän tekemättä. (ohjelman jokainen lause suoritetaan vähintään kerran.)
Päätöskattavuus	Tarkistaa, että kaikki mahdolliset polut tai haarat koodissa on katettu
Polkutestaus	Rakenteellinen testausmenetelmä, johon sisältyy ohjelma kaikkien mahdollisten suoritettavien polkujen löytämiseksi
Ehtojen kattavuus	Tarkistaa, onko jokaisen ehdon molemmat tulokset käytetty ("tosi" tai "epätosi")
Syötealuekattavuus	Kuvaa, kuinka suuri osuus syötealueista otetaan huomioon testitapauksissa

3 Testauksen automatisointi

3.1 Testiautomaatio

Testausautomaatiossa testitapaukset suoritetaan automaattisesti käyttäen niihin tarkoitettuja työkaluja. Testiautomaatio tarkoittaa ohjelmistotestausta, jossa testiautomaatiovälineen avulla suoritetaan ohjelmistoa testaavia testejä ilman testaajan puuttumista testin suoritukseen (Axelrod, 2018.) Tyypillisesti testiautomaatio ajaa suunnitellun testin, analysoi testitulokset ja päättää menikö testi läpi vai ei (kuva 3).



Kuva 3 Automatisoitu testi (mukaillen Axelrod, 2018.)

3.2 Automatisoinnin hyödyt

Axelrodin (2018) mukaan yleisin syy asiakkaiden testiautomaation aloittamiseen on tarve vähentää testaukseen kuluvaa aikaa ennen sovelluksen julkaisua. Käytännössä tähän tavoitteeseen päästään vasta pidemmällä aikavälillä.

Ketterään kehitykseen menetelmään siirtyminen on vaikuttanut osaltaan testiautomaation kasvavaan tarpeeseen. Automaatio vapauttaa kehitystiimin toimittamaan korkealaatuista koodia. Useiden iteraatioiden aikana syntyy useita uusia ominaisuuksia ja sen myötä ohjelmisto muuttuu kompleksisemmaksi. Uudet ominaisuudet lisäävät riskiä, että ne alkavat herkästi rikkomaan aiemmin toteutettua ominaisuutta. (Axelrod, 2018.)

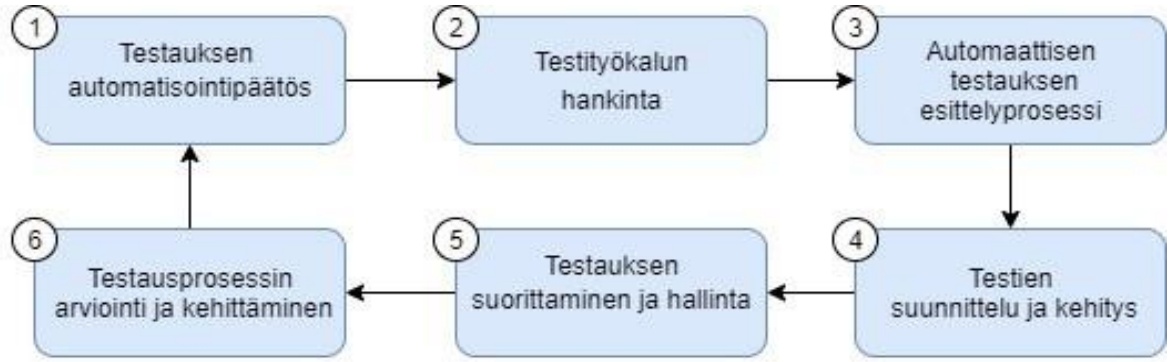
Tilanne vaikuttaa myös ohjelmistokehityksen kustannuksiin: uudet ominaisuudet tarvitsevat testaamista ja niiden pohjalta laaditut korjaukset vaativat osaltaan testaamista. Kustannuksia voidaan pyrkiä hillitsemään automatisoimalla regressiotestauksen. (Axelrod, 2018.)

Testausautomaatio kannattaa kuitenkin toteuttaa, jos projekti on vähänkään isompi, jatkuva tai siihen odotetaan tulevan jatkokehitystyötä. Testausautomaation avulla testausta voi toteuttaa yleensä lyhemmässä ajassa, jonka myötä kehittäjät saavat enemmän palautetta toteuttamistaan muutoksista. Testausautomaatiolla pystytään nopeasti tunnistamaan mahdolliset viat kehitysprosessissa. Lisäksi voidaan luottaa testien toimivan jokaisen ajon aikana yhdenmukaisesti. Automatisoitu regressiotestaus vapauttaa manuaalista testausta uusien toimintojen tutkimiseen ja testaamiseen. (Fewster & Graham 1999, 10.)

Vaikka lähtökohtaisesti automaatiotestauksen alun investoinnit ovat suuremmat verrattuna manuaalisiin testauksiin, niin pitkällä aikavälillä automaatiotestausten kustannukset osoittautuvat pienemmiksi. Automatisoidut testit mahdollistavat myös manuaalitestaukselle mahdottomia asioita, kuten testausten suorittamisen suurilla määrillä ja useiden testien ajamisen samanaikaisesti. Automatisoidut testit tuovat varmuutta, että ohjelmiston toiminnalliset osuudet ovat katettu ja voidaan luottaa ohjelmiston toimivan toivotulla tavalla. (Fewster & Graham 1999, 10.)

3.3 Automatisoinnin edellytykset

Yleisesti on todettu, että testauksen automatisointi vie kolme tai neljä kertaa enemmän aikaa kuin manuaalinen testaus. Automaatio maksaa enemmän etukäteen, mutta maksaa takaisin nopeasti joka kerta, kun regressiotesti suoritetaan. (Mosley & Posey 2002, luku 3.2)



Kuva 4 Automated Testing Life-Cycle Methodology (ATLM) (mukaillen Dustin, Garrett & Gauf (2009))

ATLM testiautomaatio -malli (kuva 4) kuvaa testiautomaation elinkaarta, joka on suunnattu automaattisen testauksen onnistuneen toteuttamisen varmistamiseen:

Vaihe 1 Testauksen automatisointipäätös:
Tähän vaiheeseen kuuluu automatisointipäätöksen teon prosessi, johon kuuluu kartoituksen toteuttaminen. Kartoituksella voidaan hallita odotuksia testiautomaatioon liittyviä odotuksia ja hahmottaa automaatiolla saavutettavat mahdolliset edut. Tässä vaiheessa toteutetaan esiselvitykset automatisointityökaluista.(Dustin, Garrett & Gauf 2009.)

Vaihe 2 Testityökalun hankinta:
Vaihe sisältää varsinaisen testityökalun arviointi- ja valintaprosessin. Valittavan työkalun on sovelluttava organisaation testausvaatimuksiin huomioiden testattava järjestelmä. Testityökalun on vastattava myös muihin organisaation esittämiin tarpeisiin ja toiveisiin.(Dustin ym. 2009.)

Vaihe 3 Automaattisen testauksen esittelyprosessi:
Automaattisen testauksen esittelyprosessissa hahmotellaan tarvittavat vaiheet, jotta automaattinen testaus voidaan onnistuneesti ottaa käyttöön. Testausprosessia analysoidaan ja sen pohjalta laaditaan päämäärät ja tavoitteet testaukselle. Tässä vaiheessa arvioidaan työkaluja tarkemmin ja verrataan niitä testivaatimuksiin. (Dustin ym. 2009.)

Vaihe 4 Testien suunnittelu ja kehitys:
Testin suunnittelu- ja kehitysvaihe sisältää testattavassa järjestelmässä vaadittavien toimintojen tarkastelun. Varmistetaan, että prosessit ja resurssit on järjestetty ja sovellettu tehokkaasti. Testisuunnitelmassa hahmotellaan tiimin roolit ja vastuut, projektin testiaikataulu, testiympäristön valmistelu sekä määritellään suoritettavien testien lukumäärä ja testiolosuhteet. Näiden avulla automaattiset testit ovat uudelleenkäytettäviä, toistettavia ja ylläpidettäviä. (Dustin ym. 2009.)

Vaihe 5 Testien suorittaminen ja hallinta:
Suunniteltuja testejä lähdetään tässä vaiheessa toteuttamaan ja analysoimaan niiden ajotuloksia. (Dustin ym. 2009.)

Vaihe 6 Testausprosessin arviointi ja kehittäminen:
Tarkistus- ja arviointitoimenpiteitä on suoritettava koko testauksen elinkaaren ajan, jotta parannuksia voidaan toteuttaa seuraavassa testausvaiheessa tai seuraavassa projektissa. (Dustin ym. 2009.)

Automaatio alkaa, kun testitapaukset on määritelty vastaamaan valmiita vaatimuksia. Automatisoimisen priorisointi tulisi kohdistua toimintoihin, jotka loppukäyttäjät suorittavat.

Mahdollisuuksien mukaan sovelluksen testitapauksia laajennetaan ei-kriittisiin toimintoihin. (Mosley ym.2002, luku 3.2.)

Vaatimusten selkeyttämiseksi sijoitetaan vaatimukset ja testitapaukset vaatimusmatriisille, jolloin vaatimusten ja eri testien väliset yhteydet ovat nähtävissä. Sen pohjalta voidaan nähdä millaisella testitapauksella vaatimukset kannattaa testata. Matriisissa on tietoa testiautomaation määrästä ja laadusta. Kyky linkittää ohjelmistovaatimukset testausvaatimuksiin on erittäin tärkeää ohjelmistojen testaustietojen kehittämisen ja käytön kannalta (Mosley ym.2002, luku 3.2.)

Dustin, Garrett & Gauf (2009) suosittelevat lähestymään testiautomaatiota prototyypin kautta. Sen avulla voidaan hankkia kokemusta automatisointityökalusta ja sen soveltuvuudesta projektiin. Prototyypillä voidaan tehdä alustavia arvioita tarvittavista resursseista ja toteutukseen annetusta aikarajasta. Rajoitteiden avulla on priorisoitava kriittiset testattavat ominaisuudet. Silloin on syytä pohtia, että onko automatisointia mahdollista tai järkevää hyödyntää tietyn ominaisuuden testaamisessa. Laaditun testin tulosten tulee olla aina ennustettavissa. Testiautomaation ajoraportin tulee saada aikaan läpäisy- tai hylkäystila.

Testiautomaatiota ei kannata toteuttaa tilanteissa, joissa testattava järjestelmä ei ole suuri tai monimutkainen tai se sisältää vain muutaman koontiversion. Tilannetta kannattaa harkita, kun ominaisuutta ei voi testata halutulla tarkkuudella, ellei se säästä huomattavaa määrää manuaalista testiaikaa.

Automaattisten testien tulisi testata sovellusta joko rakentavana tai tuhoavana testauksena. Rakentavassa tai positiivisessa testauksessa sovellus tekee ne toiminnot, jotka sen on suunniteltu tekevän. Tuhoavassa tai negatiivisessa testauksessa sovellusta testataan, jotta se ei tee mitään mitä sen ei pitäisi tehdä. Lisäksi tulisi testata sovelluksen vakautta käsitellen väärää dataa. (Mosley ym. 2002, luku 2.2.)

Fuchsin kolmivaiheinen lähestymistapa automatisoitujen testitapausten luomiseen (Mosley ym. 2002, luku 3.2):

1) Testitapauksen suunnittelu. Suunniteltaessa jokaisessa testitapauksessa tulisi olla 1–10 läheisesti liittyvää tapahtumaan tai skenaariota. Testitapaukset, joissa on yli 10 skenaariota, jaetaan useisiin testitapauksiin. Jokaiseen skenaarioon tulisi liittyä ainutlaatuinen odotettu tulos.

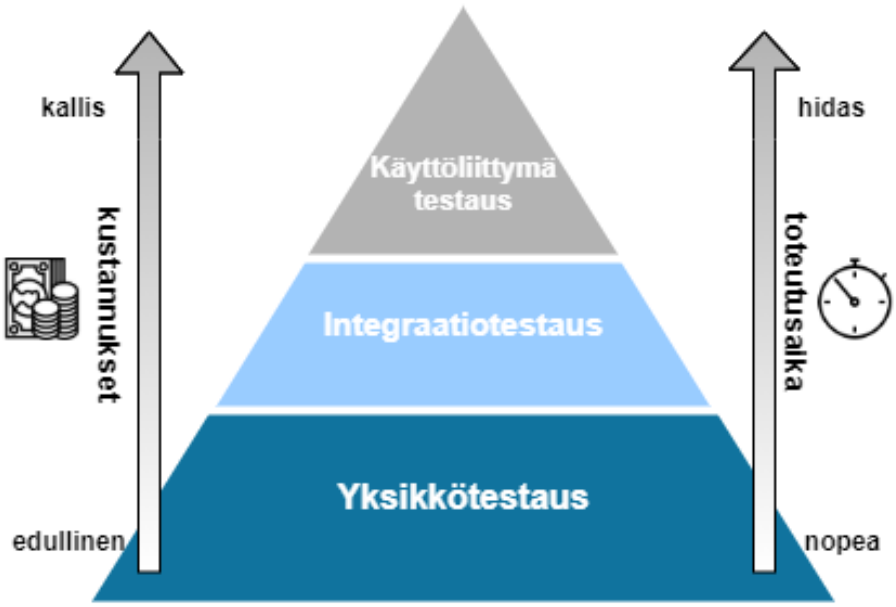
2) Testin suorittaminen manuaalisesti. Manuaalisella testauksella voidaan parhaiten havaita uusia virheitä sovelluksessa. Näiden pohjalta laaditut automaattiset testit ovat hyviä löytämään virheitä ohjelmiston kehityksestä regressiotestien aikana.

3) Testitapausten automatisointi. Testiskriptin tulisi sisältää osiot, jossa suoritetaan alkuasetukset, testi, tuloksen tarkistaminen ja kirjaaminen. Lisäksi arvaamattomia tilanteita käsittelevä osio, jonka perusteella päätetään, jatketaanko testiä. Lopuksi testin lopetus.

Testaustoimintojen tulisi simuloida, kuinka käyttäjä käyttää testattavan järjestelmän toimintoa.

3.4 Ketterän menetelmän testauksen suunnittelu- ja kehitysmalleja

Mike Cohnin testipyramidi pohjautuu ketterän kehityksen viitekehykseen. Cohn kuvaa automaatiotestien suunnittelumallin: testien automatisoinnissa huomioitavat testauksen eri tasot ja niiden keskinäinen painotus. Tarkoituksena on samalla korostaa millaiset testit antavat eniten tuottoa sijoitetulle pääomalle ja millaiset testit ovat tehokkaimpia pitkällä aikavälillä. (Crispin & Gregory, 2018.)



Kuva 5 Testipyramidi-malli (mukaillen Crispin & Gregory, 2018.)

Pyramidimallissa pohjimmaisena oleva yksikkötestauskerros muodostaa perustan muille testaukselle. Yksikkötestauksia tulisi olla suurin osa testeistä, koska ne antavat paljon nopeamman palautteen, mikä tekee virheen korjauksista nopeampia ja halvempia. Ne ovat edullisempi luoda ja niitä voi kehittää nopeasti. Yksikkötestejä laajennetaan korkeammalla

tasolla olevilla testeillä ja viimeistellään GUI-automatisoiduilla testeillä. (Crispin & Gregory, 2018.)

Kun pyramidissa siirrytään ylöspäin, testien luomiseen ja ylläpitoon liittyvät kustannukset, testin suoritusaika, testin herkkyys ja testin kattavuus kasvavat jatkuvasti (Crispin & Gregory, 2018.)

Ketterän kehityksen nelikenttä on visuaalinen työkalu erilaisten arvoa ja laatua tuottavien testien ymmärtämiseen. Lisa Crispinin jatkokehittelemän matriisin vaaka-akseli sisältää liiketoimintaan tai tekniikkaan liittyviä testejä. Pystyakseli jakaa ne liiketoimintaa ja tekniikkaa koskeviin testeihin. (Crispin & Gregory 2018, luku 6)

Kuva 6 Ketterän kehityksen nelikenttä mukaillen (Crispin & Gregory 2018, luku 6)

Ensimmäinen neljänneksen (Q1) testit kuuluvat testivetoiseen kehitykseen (TDD). Testivetoisessa ohjelmistokehityksessä laadittavat testitapaukset kirjoitetaan ennen kuin varsinaista koodia lähdetään toteuttamaan. Ne määrittävät, miten testattavan toiminnallisuuden tulisi toimia. Testin kirjoittamisen jälkeen ohjelmakoodia muokataan vain sen verran, että se läpäisee kirjoitetun testin. Tämän jälkeen ominaisuutta laajennetaan ja vaihe toistetaan uudelleen kirjoittamalla testit, toteuttamalla lisäyksen koodiin ja testaamalla ne. Q1 sisältämät yksikkö- ja komponenttitestit ovat yleensä osa testien automaatioprosessia, jossa jokaisen testiajon jälkeen kehitystiimi saa jatkuvaa palautetta koodin laadusta. Automatisoituja yksikkö- ja komponenttitestit sekä jatkuva integrointiprosessi ovat keinoja tuottaa arvoa tuovia testejä ajoissa. (Crispin & Gregory 2018, luku 7).

Toisen neljänneksen (Q2) liiketoimintaan kohdistuvat testit tukevat kehitystiimin toimintaa korkeammalla tasolla. Testit kirjoitetaan usein käyttäjätarinoiden ja lauseen muodossa, kuka haluaa kyseisen toiminnallisuuden ja miksi. Näistä lähtökohdista kehitystiimi voi lähteä toteuttamaan haluttua toiminnallisuutta. Asiakkaan kanssa yhteistyössä tarinalle laaditaan testi, jota kehittäjät hyödyntävät toiminnallisuutta toteuttaessa. Toiminallisuuden valmistuessa saadaan asiakkaalta palautetta, mihin suuntaan toiminnallisuutta lähdetään kehittämään ja vaatimuksen tarkentuvat. Testien automatisointi ei voi rakentua vain teknologiaan kohdistuviin testeihin, joilla pyritään virheettömään koodiin. Automatisoinnissa tulee huomioida liiketoimintaan kohdistuvat testit, jotka auttavat rakentamaan kehitettävää sovellusta asiakkaan tarpeiden mukaiseksi. Toisen neljänneksen (Q2) testit voivat vastata osittain Q1:ssa kirjoitettuja testejä. Kuitenkin Q2:n testit selvittävät laajemmin asiakkaan vaatimukset ymmärrettävässä muodossa. Testit määrittelevät ja varmistavat ulkoisen laadun ja auttavat kehitystiimiä arvioimaan, milloin vaatimus on valmistunut. (Crispin & Gregory 2018, luku 8)

Kolmannen neljänneksen (Q3) testien kautta saadaan palautetta Q1 ja Q2:ssa luoduista testeistä. Testaajat tai käyttäjät antavat palautetta toteutetusta tuotteesta. Arvioinnin kohteena ovat iteraatioiden tuloksena syntyneet tuotteet, jotta saadaan kohdennettua palautetta mm. testattavan järjestelmän toiminnasta ja ulkonäöstä. Tuotetta kritisoivien liiketoimintatestien automatisoiminen on haasteellista, koska tällainen testaus perustuu ihmisen älykkyyteen, kokemukseen ja vaistoon. Automatisointia voi kuitenkin käyttää testidatan luomiseen ja testien alustamiseen manuaalitestaajia varten. (Crispin & Gregory 2018, luku 10).

Neljänteen neljännekseen (Q4) sisältyvät testit kritisoivat tuotteen teknologisia ratkaisuja. Testit koskevat lähinnä ei-toiminnallisia vaatimuksia, jotka on huomioitava järjestelmän toiminnallisten vaatimusten toteutuksessa. Q4 testit liittyvät mm. järjestelmän suoritus- ja kuormituskykyyn sekä tietoturvallisuuteen. (Crispin & Gregory 2018, luku 11).

3.5 Testausautomaation työkalun valinta

Työkalun valintaprosessin perimmäisenä tavoitteena on tunnistaa työkalu, jolla testaus automatisoidaan organisaatiossa. Ohjelmistotestaukseen ja testiautomaatioon on tarjolla valtava määrä kaupallisia ja avoimen lähdekoodin työkaluja. Testiautomaation elinkaareen (ATLM vaihe 2, kuva 4) liittyvään työkalun valintaan ja hankintaan liittyvä kriteeristö on oleellinen ja suuri vaikutustekijä päätettäessä aloitetaanko toteuttamaan testiautomaatiota.

Raulamo-Jurvanen, Mäntylä & Garousi (2017) ovat tutkimuksessa tarkastelleet myös testiautomaatiotyökalun valintaa. Teknisen aihealueen tärkeimmäksi kriteeriksi käyttäjät nostivat käytettävyyden (lähes 77 %) ja ulkoisten tekijöiden tärkein kriteeri oli työkalujen kustannukset (lähes 82 %). Kriteerit näyttävät olevan kontekstikohtaisia eivätkä ole sovellettavissa sellaisenaan. Kriteeristö voi vaihtua tai painottua eri tavalla testauksen eri vaiheissa.

Raulamo-Jurvanen ym. (2017) tutkimuksessa kuitenkin nousi esiin se, että automaattisen testauksen tärkeimpänä esteenä on projektin tarpeisiin sopivien työkalujen puute. Mahdollisena syynä esitettiin, että työkalun valintavaiheessa kiinnostus kohdistuu laajasti tunnettuihin ja käytettyihin työkaluihin. Tilanne voi johtaa siihen toteamaan, että automatisointityökalut eivät vastaa projektin tarpeisiin. Työkaluja käytetään sopimattomalla tavalla tai jonkin suosiminen voi johtaa epäkäytännöllisiin ratkaisuihin

Käsitys kokonaistilanteesta, tarpeista sekä vaatimuksista luovat pohjan työkalun valinnalle. Bajajin (2018) suosittelee laatimaan vertailumatriisiin, johon nostetaan avainparametrit ja kriteerit automatisointityökalulle. Alla näkyvässä taulukossa Bajajin ehdotelma tärkeimmistä asioista automatisointityökalun valinnan tueksi (taulukko 1.)

Taulukko 1 Vertailumatriisi (mukaillen Bajaj (2018))

Parametrit	Kriteeri
Omaksumisen helppous	Lisenssikustannukset
	Tuen helppous
Skriptauksen ja raportoinnin helppous	Skriptauksen luontiaika
	Skriptauskieli
	Objektien tunnistaminen
	Skriptin suorittamisen nopeus
	Viitekehys
	Mahdollisuus jatkuvaan intergroimiseen
Työkalun käyttö	Ei-selainpohjainen sovellustuki
	Käyttöjärjestelmätuki
	Selaintuki
	Laitetuki

Testiautomaatiotyökalun valitseminen on oma projektinsa, johon on syytä varata riittävästi resursseja, varoja, aikaa ja henkilöstöä. Hyvin valittu työkalu testiautomaatioon ei kuitenkaan takaa onnistunutta testiautomaatiota. Koko testausprosessin tulee olla hyvin suunniteltu ja ylläpidetty. (Fewster & Graham 1999, 248.)

3.6 Testiautomaation viitekehykset

Testiautomaation viitekehykset ovat kokoelma työkaluja sekä prosesseja, joilla toteutetaan sovelluksen testauksen automatisoimista. Hyvän testiautomaation viitekehyksen tulisi olla laajennettavissa sitä mukaan, kun testattava ohjelmisto kehittyy ja sillä pystyy testaamaan ohjelman eri komponentteja. Testiautomaatioratkaisuja voidaan tehostaa viitekehyksillä ja testien liittämisellä jatkuvaan integraatioputkeen. Ratkaisu parantaa viestintää ja projektien näkyvyyttä sekä vähentää riskejä koko kehityksen elinkaaren ajan. (Hanna, El-Haggar & Mostafa 2014.)

Automaatio rakentuu testiskripteihin, jotka sisältävät tallennettuja komentoja tai tapahtumia testitapauksen suorittamiseksi ja tulosten ilmoittamiseen. Testiskriptien avulla komentosarjat voivat toistaa useita kertoja eri tiedoilla, koska ne sisältävät erilaisia vaiheita, jotka sisältävät loogiset päätelmät arvojen käsittelytavasta. (Hanna ym. 2014.)

Lineaarinen viitekehys tunnetaan myös nimellä tallennus ja toisto -viitekehyksenä. Komentosarjoja luodaan tallentamalla/nauhoittamalla käyttäjän manuaalisesti suorittamat toiminnot ja validointivaiheet järjestelmän käyttöliittymälle. Testitoiminnot tallennetaan testikoodina. Luotu testiskripti koostuu sarjasta testausohjeita, jotka käyttävät työkalun tukemaa ohjelmointikieltä. (Hanna ym. 2014.)

Hyötynä on nopea toteuttaminen, testaaja voi nauhoittaa manuaalisen testitapauksen ja erillistä suunnitteluvaihetta ei tarvita. Automatisoinnin toteuttamiseen ei edellytä erityisiä ohjelmointitaitoja. Viitekehyksen haittapuolina voidaan pitää testitapauksia epästabiileina, koska olosuhteiden tulisi olla samat kuin nauhoituksessa. (Hanna ym. 2014.)

Strukturoitu ja jaettu skriptaustekniikka. Strukturoitu skriptaustekniikka käyttää jäsenneltyjä ohjelmointiohjeita, jotka ovat joko ohjaus- tai kutsurakenteita. Ohjausrakenteita käytetään testauskoodin eri polkujen ohjaamiseen (esim. If-else, switch, for, while). Testiskripti voi vahvistaa asetettujen olosuhteiden perustella, onko testi mennyt hyväksyttävästi läpi tai epäonnistunut. (Hanna ym. 2014.)

Jaetussa skriptaustekniikassa testin yhteisiä toiminnallisuuksia säilytetään yhdessä lokaatiossa, josta niitä kutsutaan käytettäväksi toisessa skriptissä. Näiden toteutus ja ylläpito edellyttävät ohjelmointitaitoja. (Hanna ym. 2014.)

Aineistopohjainen testaus. Aineistopohjaisessa viitekehyksessä käytettävä testidata/syötetieto tuodaan testiskriptiin ja sen muuttujiin erillisestä tiedostosta tai tietokannasta. Testien suorittamisen aikana testitiedon muuttujat luetaan tästä ulkoisesta tiedostosta. Tämän ansiosta testiskriptiä ei tarvita muokata testidatan muuttuessa. Skriptejä voi suorittaa käyttämällä useita tietojoukkoja useissa yhdistelmissä parametrisoinnin avulla. Samanlaisia testejä voidaan lisätä hyvin nopeasti erilaisilla syötetiedoilla, koska samaa komentosarjaa voidaan käyttää eri testien suorittamiseen eri tiedoilla. Tämän takia ylläpitokustannukset ovat alhaisemmat. (Hanna ym. 2014.)

Tämä tekniikka vaatii ohjelmointitaitoja työkalun tukemalla komentosarjakielellä. Tällaisia testejä on hallittava hyvin, koska ne edellyttävät eri testiohjelmien käyttämien datatiedostojen ylläpitoa. Testiskriptien määrä voi kasvaa dramaattisesti, kun tarvitaan loogisesti erilaisia testitapauksia. Tämä voi lisätä projektin kustannuksia (Hanna ym. 2014.)

Avainsanapohjainen testaus. Avainsanapohjaisella testauksella testitapaukset kuvataan käyttämällä avainsanoja. Avainsanat liittyvät eri toimintoihin, joita vaaditaan tietyn vaiheen suorittamiseen testitapauksessa. Testattavat toiminnot tallennetaan taulukkomuodossa. (Hanna ym. 2014.)

Avainsanavetoinen testi koostuu korkean ja matalan tason avainsanoista. Matalan tason avainsanat soveltuvat paremmin yksityiskohtaiseen testaukseen käyttöliittymän tasolla. Korkean tason avainsanat soveltuvat paremmin korkeamman tason toimintojen, kuten testattavan järjestelmän liiketoimintalogiikan testaamiseen (esim. Luo tili, Kirjaudu sisään). Korkean tason avainsanoja voidaan rakentaa useista matalan tason avainsanoista. Testi antaa kuvauksen testitapauksesta, joka on automatisoitava avainsanojen avulla. Avainsanat luetaan ja tulkitaan myöhemmin testitapauksen suorituksen aikana. Testitiedostossa ilmoitetaan mitä testitapaus tekee, ei miten se tehdään. (Hanna ym. 2014.)

Avainsanavetoinen testaus voidaan jakaa kahteen pääkerrokseen (Hanna ym. 2014.):

1. Infrastruktuurikerros: Yhdistelmä avainsanoja kolmesta alla esitetystä ryhmästä. Se vastaanottaa eri avainsanat syötteinä testattavan järjestelmän toimintojen suorittamiseen.

2. Looginen kerros: Tämä kerros auttaa manuaalisia testaajia rakentamaan uusia testiskriptejä käyttämällä ennalta määritettyjä avainsanoja (jotka on jo otettu käyttöön infrastruktuurikerroksessa).

Avainsanat voidaan jakaa kolmeen eri ryhmään (Hanna ym. 2014.):

Perustason avainsanat (Item Operation)	Käyttöjärjestelmässä olevalla komponentilla tapahtuva toiminto
Apuohjelman toimintoihin liittyvät avainsanat (Utility Functions)	Komentosarja, joka suorittaa tietyn toiminnallisen operaation kuten ”Sulje näyttö”
Sekvenssi/ käyttäjälähtöiset avainsanat (Sequence/ User Keywords)	Luodaan yhdistämällä perustason ja korkeamman tason avainsanoja liiketoiminta-prosessia kuvaavaksi avainsanaksi, kuten "Luo käyttäjä" -avainsana.

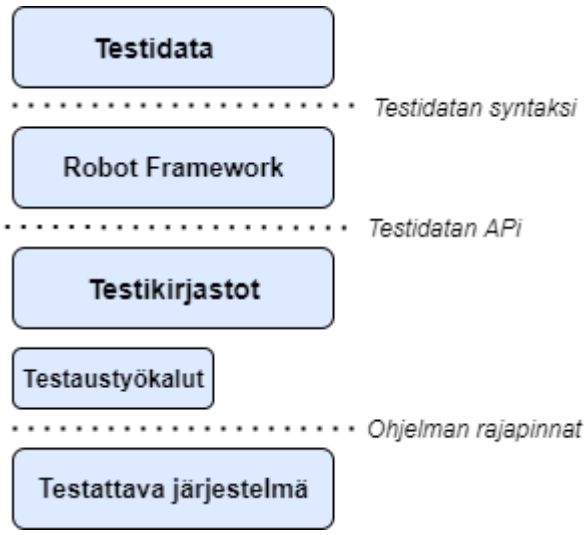
4 Analysoitavat testiautomaatiotyökalut

4.1 Robot Framework

Robot Framework on modulaarinen, avoimen lähdekoodin testiautomaatioviitekehys, jolla laaditaan avainsanapohjaisia testiskriptejä. Sitä voidaan soveltaa hyväksymistestaukseen, asiakkaan vaatimuspohjaiseen hyväksymistestivetoiseen ohjelmistokehitykseen sekä myös ohjelmistorobotiikkaan (Bisht 2013, s.10).

Python-ohjelmointikieleen perustuva Robot Framework on alustasta ja teknologiasta riippumaton. Työkaluun voidaan liittää useita erilaisia avainsanakirjastoja, joita voi myös laajentaa Python- tai Java-ohjelmointikielillä luoduilla testikirjastoilla (Bisht 2013, s.10).

Robot Frameworkin käyttö vaatii ennakkovaatimuksena Pythonin sekä sen eri pakettien asennukseen tarvittavan pip:n. Tämän jälkeen komentoriviltä voidaan asentaa Robot Framework ja selaimen automatisointiin tarvittavat Selenium-kirjaston ja selainajurit. Testitapaukset voidaan laatia haluamalla tekstieditorilla tai tähän kehitettyä omaa työkalua RIDE (Robot Integrated DevelopmentEnviroment). (Robot Framework Foundation, 2021.)



Kuva 7 Robot Framework arkkitehtuuri (Robot Framework Foundation, 2021.)

Testidata rakentuu testitapaus- ja resurssikansioista, jotka Robot Framework prosessoi ja suorittaa. Suorituksen aikana se tuottaa raportteja ja lokeja, jotka ovat HTML- ja XML-muodossa. Luodut raportit sisältävät yksityiskohtaista tietoa jokaisesta käytetystä testitapauksen avainsanasta. Ydinohjelmalla Robot Framework tulkitsee testidatassa olevat avainsanat, mutta sillä ei ole suoraa yhteyttä testattavaan järjestelmään, vaan testikirjastot muodostavat yhteyden ohjelman rajapintoihin ja toteuttavat halutut komennot. Vaihtoehtoisesti kirjastot hyödyntävät testityökaluja ajureina (Robot Framework Foundation, 2021.)

Fyysistä tiedostoa, joka sisältää testitapaukset, nimitetään testijoukoksi (suite). Testidata luodaan taulukkomuotoon, josta yleisin tapa on toteuttaa se vähintään kahta välilyöntiä erottamaan dataa toisistaan. Robot Framework käsittelee ne omina riveinään ja pilkkoo ne pienempiin osiin, merkeiksi, jossa välilyönnit auttavat erottamaan avainsanat ja niihin kuuluvat argumentit. Argumentit sisältävät avainsanaan kuuluvia arvoja, jotka voivat olla pakollisia tietoja, jotta avainsanaa voidaan käyttää (Robot Framework Foundation, 2021.)

Osana Suite-tiedostoa ovat laaditut testitapaukset, joihin voidaan liittää muita tiedostoja. Nämä muut tiedostot ovat muuttuja- ja resurssitiedostot. Resurssitiedostojen avulla testitapaustiedostoissa luotuja käyttäjän tekemiä avainsanoja ja muuttujia voidaan laajentaa niiden hyödynnettävyyttä myös käytettäväksi muihin testitapaustiedostoihin. Muuttujatiedostojen avulla voidaan luoda muuttujia Python muodossa sekä jakaa niitä. Lisäksi tähän kuuluvat aiemmin mainitut sisäiset ja ulkoiset testikirjastot, jotka sisältävät matalan tason avainsanat (Robot Framework Foundation, 2021.)

Robot Frameworkin syntaksi rakentuu neljään taulukkotietoon, jossa määritetään asetukset, muuttuvat, testitapaukset sekä avainsanat. Nämä tiedot erotetaan määritetyllä tavalla, joka näkyy kuvassa 8 (Robot Framework Foundation, 2021.)

```
*** Settings ***
Library       OperatingSystem     ulkoinen kirjasto
Library       ..${/}Libs${/}Omapython_kirjasto.py   käyttäjän luoma kirjasto
Resource      käyttäjän_avainsanat.robot  käyttäjän luoma resurssitiedosto

*** Variables ***
${BROWSER}          chrome      muuttujaan asetettu arvo

*** Test Cases ***
Kirjautuminen Sivustolle
    Siirtyminen sivustolle
    Syötä Käyttäjätunnus    käyttäjätunnus
    Syötä Salasana    salasana

*** Keywords ***
Siirtyminen sivustolle
   Open Browser       http://testiosoite.fi/    ${BROWSER}  matalan tason avainsana

Syötä Käyttäjätunnus
    [Arguments]     ${username}
    Input Text     //*[@id='username_field'] ${username}

Syötä Salasana     käyttäjän luoma avainsana
    [Arguments]     ${password}
    Input Text     //*[@id=password_field']     ${password}
```

Kuva 8 Robot Frameworkin taulukkotiedot

Asetukset (Settings) osiossa lisätään käytettävät tiedostopolut muuttuja- ja resurssitiedostoihin sekä nimetään käytettävät testikirjastot. Asetuksissa voidaan määritellä tarkemmin testitapausten alustukset (Setup) ja lopetukset (Teardown). (Robot Framework Foundation, 2021.)

Muuttujat (Variables) osiossa määritetään testissä käytettävät muuttujatiedot, jotka voivat olla avainsanoissa ja testitapauksissa käytettäviä muuttujia tai niiden avulla voidaan asettaa muuttujalle arvo. (Robot Framework Foundation, 2021.)

Testitapaukset (Test cases/task) osiossa luodaan varsinaiset testitapaukset hyödyntämällä avainsanoja. Testitapaukseen itseensä liittyy asetustietoja kuten kuvassa kolme on esitetty. Automatisoitava testitapaus merkitään yksilöivällä Tags-tunnisteella, jotta se voidaan yhdistää suunniteltuihin testitapauksiin ja sitä kautta vaatimuksiin. Tämä helpottaa myös logitietojen tulkitsemista, kun testitapauksella on yksilöity tunniste. Testitapaukset voidaan myös liittää osaksi regressio- tai smokesettiä liittämällä yksilöivän tunnisteen lisäksi toinen tunniste (smoke/regression). (Robot Framework Foundation, 2021.)

Kuva 9 Robot Framework testitapauksen asetukset

Avainsana (keywords) osiossa luodaan käyttäjän omat avainsanat hyödyntämällä sisäänrakennettuja ja ulkopuolisia testikirjastoja, jotka sisältävät matalan tason avainsanoja. Käyttäjän avainsanat voivat sisältää myös muita käyttäjän aiemmin luotuja avainsanoja. (Robot Framework Foundation, 2021.)

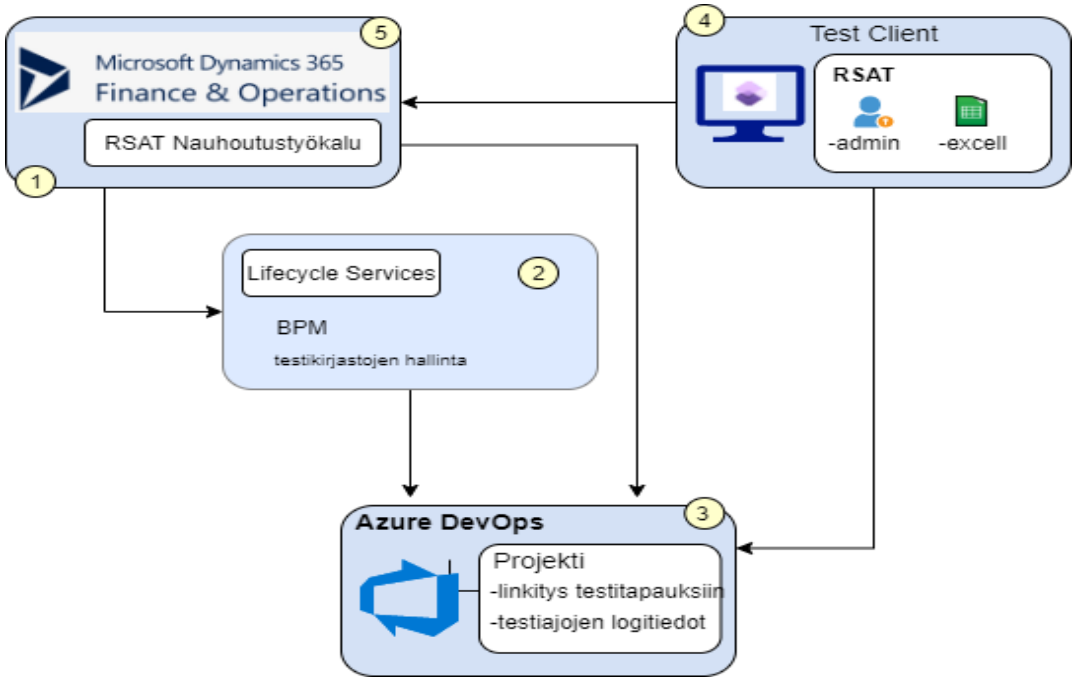
4.2 RSAT- Regression Suite Automation Tool

Microsoft kehitti ja optimoi Dynamics 365 for Finanace and Operations-sovelluksen palvelupäivityksiä ja niiden toimituksia vuonna 2018. Muutoksen tavoitteena oli tarjota kaikille

asiakkaille ennakoitava ja joustava palvelupäivityksen toimitus, jossa tarjotaan viimeisimmät tuoteominaisuudet sekä alustan ja sovelluksen korjaukset / päivitykset. (Osborne 6.7.2018.)

Microsoft on sitoutunut toimittamaan kaksi suurta päivityskokonaisuutta vuodessa, huhti- ja lokakuussa sekä noin kerran kuussa tulevia pienempiä päivityksiä. Asiakkaat voivat viivästyttää kaksi peräkkäistä päivitystä kerrallaan, mutta vuoden aikana tulee vähintään neljä palvelupäivitystä. Jotta mahdollisia ongelmia ei tuoda tuotantoympäristöön, on pitää tehdä vähintään neljä regressiotestikierrosta vuodessa. Regression Suite Automation Tool on Microsoftin tarjoama työkalu palvelupäivityksen yhteydessä tapahtuvaan regressiotestaukseen. (Microsoft 2020a.)

Työkalu on suunniteltu osaksi käyttäjän toteuttamaa hyväksymistestausta, jossa validoidaan liiketoimintaprosessien sujuvuus. Microsoft korostaa, ettei työkalua ole tarkoitettu yksikkö- tai integraatiotestaukseen, joihin soveltuvat paremmin toiset työkalut. Se koostuu nauhoitustyökalusta, joka on mukana itse testattavassa Finance & Operations sovelluksessa sekä erikseen asennettavasta työkalusta, jolla nauhoitteet käsitellään automatisoiduiksi testitapauksiksi. (Microsoft 2021a.)



Kuva 10. RSAT ja sen integroituminen muihin työkaluihin (mukaillen Avantiico s.a.)

Kuvan 10 mukaisesti RSAT-työkalu on suunniteltu osaksi end to end -flowta, jossa yhdessä Microsoft Dynamics Lifecycle Services (LCS) ja Azure DevOps avulla rakentuu kokonaisuus testitapausten laatimiseen, hallinnointiin, määrityksiin, suoritukseen, tutkimiseen ja raportointiin. (Microsoft 2021a.)

Kohdassa 1 Microsoft Dynamics 365 Finance & Operations- sovelluksen asetukset valikon takaa löytyy nauhoitustyökalu, joten testien nauhoittajat eivät tarvitse erikseen asentaa nauhoitustyökalua (kuva 10.) Nauhoituksen alussa testi nimetään ja tämän jälkeen etusivulta aloitetaan nauhoittamaan käyttäjän suorittamat toiminnot (eri siirtymät valikoista ja näkymistä toiseen sekä kenttien täyttämiset) sovelluksessa. Nämä vaiheet tallentuvat vaiheina erilliselle nauhoitusnäkymälle. Klikattavat ja täytettävät kentät tallentuvat muuttujina, joten jos käytettäviä arvoja halutaan toisessa tapauksessa muuttaa, nauhoitusta ei tarvitse toteuttaa uudelleen eri arvoilla. Jotta jatkossa testitapauksia voidaan linkittää toisiinsa, nauhoitustyökalussa löytyy kenttiin kohdistuvia lisävalintoja. Lisävalinnoissa täytettävien kentät voidaan copy merkinnällä tallentaa tallenteeseen parametriksi, jotka voidaan seuraavassa vaiheessa toteutettavalla RSAT:lla hyödyntää. Validate lisänapilla voidaan käyttää, jos halutaan arvioida arvojen täsmäävän annettua arvoa. Kun haluttu testitapaus on saatu nauhoitettua, se voidaan tallentaa joko suoraan Lifecycle Services palveluun tai tallenne voidaan tallentaa koneelle ja sieltä myöhemmin tallentaa Azure DevOpsiin. (Microsoft Dynamics 365 4.5.2020.)

Kohta 2 viittaa Microsoft Dynamics Lifecycle Services (LCS) palvelussa tapahtuvaan testien jakeluun ja hallintaan, jossa käytetään Business process modeler-työkalua (kuva 10.) Tämä ei ole pakollinen työkalu testitapausten automatisoimiseksi, vaan työkalu testikirjastojen luomiseen ja sen hallinnoimiseen. (Microsoft 2021a.)

Kohdassa 3 Azure DevOpsia käytetään testitapausten hallinnoimiseen, joka tapahtuu Test Plan valikon alta (kuva 10.) Testisuunnitelman alle luodaan test suite, joka sisältää testitapaukset. Tämän jälkeen nauhoitukset voidaan liitää samoin nimettyyn testitapaukseen. Jos käytössä on Lifecycle Service testitapaukset ovat synkronoituneet Azure DevOpsiin valmiiksi. Kun nauhoitukset ovat liitetty suunniteltuihin testitapauksiin, voidaan niitä käsitellä RSAT:lla. (Microsoft Dynamics 365 30.4.2020.)

Kuvan 10 mukaan kohdassa 4 varsinainen Regression Suite Automation Tool asennetaa lokaalille tietokoneelle, josta käytetään nimitystä test client. RSAT on suunniteltu asennettavaksi Windows 10 -käyttöjärjestelmään ja asennuksen yhteydessä varmistaa, että käyttäjällä on asennettuna myös Selenium ja web selainajurit (kuten Google Chrome). (Microsoft 2021a.)

Ennen työkalun käyttöä konfiguroidaan työkalu Finance & Operations sovellukseen liittämällä sovelluksen ympäristötiedot, host-tiedot sekä thumbprint-autentikointi sertifikaattitiedot. Lisäksi asetuksissa liitetään yhteys myös Azure DevOps projektiin sekä siinä olevaan Test Planiin. Asetuksissa valitaan myös selain, jossa testit ajetaan (Microsoft 2021a.)

Työkalun käyttäminen asennusten jälkeen alkaa lataamalla Azure DevOpsista testitapaukset ja niihen liitetyt nauhoitteet. Tämän jälkeen valitaan test suite, jota halutaan käsitellä ja muuntaa automatisoiduksi testiksi. Tämä toimintaa muuntaa nauhoitukset c sharp ohjelmointikielelle omaksi tiedostoksi sekä luo erillisen Excel-tiedoston, jossa käytettäviä parametreja voidaan käsitellä (Microsoft Dynamics 365 7.5.2020.)

Excel tiedosto rakentuu eri välilehdistä, jossa ensimmäisenä General-välilehdeltä löytyvät yleiset asetukset testiä varten sekä nauhoituksessa tallennetut muuttujat. MessageValidation-välilehdellä voidaan tarkistaa testissä esiintyneet ilmoitukset. Lisäksi jokaisesta lomakkeesta, jota testi on käsitellyt, muodostuu oma välilehti. Välilehdellä oleva muuttajan parametria voidaan hyödyntää toisessa testitapauksessa muokkaamalla välilehdillä olevaa kenttää, johon tietoa halutaan käyttää. Välilehdillä olevia arvoja voi tarpeen mukaan muokata halutunlaiseksi. Tiedostoilla voidaan tarvittaessa toteuttaa erilaisia kaavoja numeerisilla muuttujilla sekä käyttää Excel-tiedostoissa käytettäviä toimintoja. Työkalulla voidaan tämän jälkeen valita testit, jotka halutaan ajoon ja painetaan Run tai komentokehoitteen kautta komennolla *playbackbyid* (Microsoft Dynamics 365 7.5.2020.)

Ajettujen testien tulokset näkyvät työkalulla, mutta tarkemmin ajoja pääsee tarkastelemaan Azure DevOpsissa, jossa on oma Runs välilehti. Siellä voi tarkastella testikohtaisia tetilogeja, johon on tallentunut yksityiskohtainen tieto ajosta. Jos testi on kaatunut, logi ilmoittaa syyn kaatumiselle sekä ottaa kuvakaappauksen (Microsoft Dynamics 365 7.5.2020.)

Kun testien automatisointi on valmis, voidaan testit ladata Azure DevOpsiin (kohta 5, kuva 10) Tässä on mukana muodostuneet testiskriptit, Excell-tiedostot sekä nauhoitukset. Näin tietoja voidaan hyödyntää myös muilla test clientilla, jossa ne voidaan ladata ja ajaa (Microsoft Dynamics 365 7.5.2020.) Asetuksissa voidaan liittää sign off task, jossa ladattu testisetti hyväksytään nimetyllä henkilöllä liittämällä hänen sähköpostiosoitteensa (Microsoft 2021b.)

5 Automatisointityökalujen analyysi

5.1 Työkalujen analyysin lähtökohta

Opinnäytetyön toimeksiantajan liiketoiminnan edustajat olivat toteuttaneet hyväksymistestauksia toiminnanohjausjärjestelmään. Ketterän kehitykseen kuuluu useita kehityssyklejä, joiden myötä toiminnallisuuksien määrä kasvaa. Siinä on riskinä kehitettyjen ominaisuuksien rikkoutuminen prosessin edetessä. Kehitystiimit tarvitsevat liiketoiminnan edustajien palautetta ja kritiikkiä kehittämisvaiheessa olevasta toiminnanohjausjärjestelmästä ja koodin laadusta. Regressiotesteillä parannetaan koodin laatua ja ajantasaista palautetta toimivuudesta. Regressiotestien automatisoiminen mahdollistaa vaatimusten testaamisen tehokkaasti.

Tutkittavan järjestelmän ominaisuutena on sisäänrakennettu testiautomatisointityökalu (RSAT), jonka käyttömahdollisuuksia halutaan tutkia tarkemmin. Toimeksiantajan muissa projekteissa Robot Framework on ollut osana laadunvarmistusprosessia. Sen hyödyntämistä ERP-projektissa halutaan testata.

Opinnäytetyön tehtävänä on suorittaa analyysi kahden testiautomaatiotyökalun välillä, jotta voidaan varmistaa niiden tehokas käyttö osana hankkeen testausprosessia. Tavoitteena on selvittää kahden testausautomaatiotyökalun erot, heikkoudet, vahvuudet sekä soveltuvuudet eri testitilanteiden suorittamisessa. Tuloksena rakentuu soveltuvuusanalyysi, jonka avulla voidaan nähdä, kuinka tehokkaasti hyödynnetään työkalua työpanoksen suhteessa testitapausten määrään.

Analyysissa etsin vastauksia opinnäytetyössä selvitettäviin tutkimusongelmiin:
1. Miten käyttöympäristöön valitut testiautomaatiotyökalut kohdennetaan tehokkaasti työkalun soveltuvuuden mukaan?
2. Mikä vaikutus on työkalujen valinnalla liiketoiminnan edustajien ajankäyttöön ja työtehtävien priorisointiin?

Testauksen aikataulusuunnitelma ja testisuunnitelma sekä projektiin kuuluvat henkilöt olivat tiedossa. Kohdeprojektiksi oli valittu ERP, jossa analysoitavia työkaluja (RSAT ja Robot Framework) projektin tarpeen mukaan käytetään. Opinnäytetyöni analysoitavat testiautomaatiotyökalut esittelin luvussa 4. Toimeksiantajan organisaatiossa käytettävät testihallintatyökalut ovat Jira ja Confluence. Jiraan tallennetaan toteutuksen kokonaisuudet, testitapaukset ja -havainnot. Jiran liitännäistä Xraytä käytetään testitapausten hallinnoimi-

seen. Confluencea käytetään dokumentointiin, jossa eri osa-alueiden testauksen tuotokset ja suunnitelmat lisätään. Näin ollen työkalun on sovittava käytettäviin menetelmiin ja testien hallintatyökaluihin.

Testiautomaatiotyökalujen analyysin pohjalta voi tehdä oikeat työkaluvalinnat testitilanteeseen huomioiden testien ylläpidettävyys. Testattavan järjestelmän toimittaja oli toteuttanut RSAT-työkalun käyttökokeilun ennen opinnäytetyön aloittamista. Analyysissä kokosin dataa ja havaintoja toteutetuista proof of concept testitapauksista. Toteuttamiseen oli valittu testitapaukset, jotka sopivat ennalta tiedettyyn käyttömahdollisuusrajaukseen, jossa nauhoitustyökalu toimii vain kyseisessä Finance&Operations -sovelluksessa

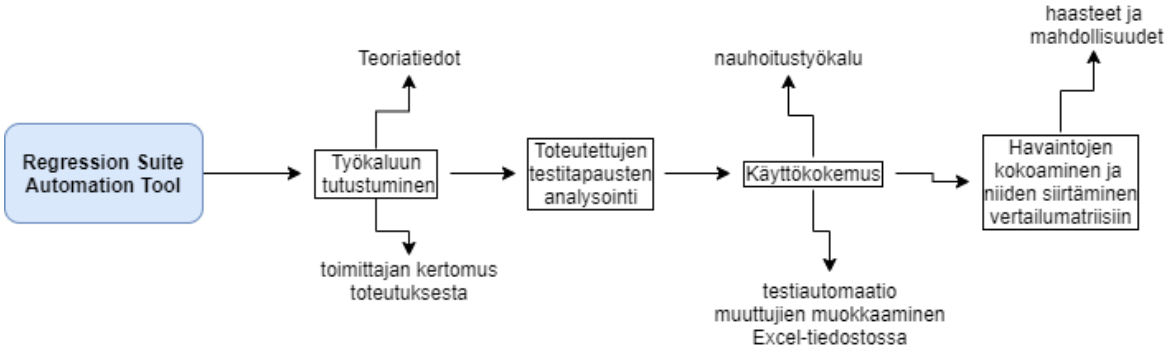
Pääasiallinen tehtäväni oli toteuttaa proof of concept Robot Frameworkilla. Testattavaa järjestelmää varten oli perustettu projektirepositorio Azure DevOpsiin, johon oli jo alustavasti lähdetty Robot Framworkilla automatisoimaan testattavaan sovellukseen rajattuja testitapauksia. Automatisointia haluttiin laajentaa ja toteuttaa testitapaus, jossa ylitetään rajapinta. Testitapaukseksi valittiin ostolaskun luominen, jossa dataa luodaan toisessa ostolaskujen kierrätysjärjestelmässä. Ostolaskujen kierrätysjärjestelmän lähettämiä tietoja käsitellään testattavassa järjestelmässä. Prosessi lopetetaan siirtymällä takaisin ostolaskujen kierrätysjärjestelmään, jossa tarkistetaan käsitellyn tiedon siirtyminen takaisin järjestelmään. Robot Frameworkille suunnattu proof of conceptin testitapauksen taustalla oli organisaation tarve varmistaa liiketoiminnalle kriittiset testitapaukset, joissa toteutuu integraatio järjestelmien välillä.

Kokosin havainnot RSAT ja Robot Framework työkaluista vertailumatriisiin, jonka tarkoitus oli visualisoida asiaa johtopäätösten ja suosituksen tekemisessä. Analyysiprosessi liittyi teoriaosiossa esitettyyn Automation Test Life Cycle (kuva 4) ensimmäiseen vaiheeseen, johon kuuluu automatisointityökalujen esiselvitys.

5.2 RSAT ja Robot Framework testiautomaatiotyökalujen analysointia

Tässä osiossa kuvaan valittujen testiautomaatiotyökalujen analysointiprosessisiani. Selvitän ensin RSAT-työkalun toiminnot prosessikuvan 11 mukaisesti. Sitten kuvaan Robot Frameworkilla laatimani proof of concept testitapauksen ja sen pohjalta toteuttamani havainnoit prosessikuvan 12 mukaisesti. Työkalujen vertailussa hyödynsin mukaillen Bajajin (taulukko 1) vertailumatriisia, johon kokosin analysoitavien työkalujen tiedot helpottamaan vertailua.

Regression Suite Automation Toolin analysointi:



Kuva 11 RSAT analysointivaiheet

Toteutuksen aikana työkalulla nauhoitettiin ja toteutettiin 49 testitapausta. Ne jaettiin kahdeksaan eri liiketoimintakategoriaan, joita testattavassa järjestelmässä voi toteuttaa. Ajallisesti tähän meni kahdelta testiautomaatioasiantuntijalta kokonaisuudessaan ajoineen noin kaksi kuukautta projektin aloittamisesta.

Testitapauksia oli päädytty pilkkomaan pienempiin vaiheisiin hallittavuuden takia. Pienet testit muodostavat laajemman testikokonaisuuden. Oleelliset muuttajat ja vaiheet saatiin nauhoitettua tallenteelle ja virheellisten tallennuksen määrä väheni. Pilkkominen pienempiin vaiheisiin näyttää olevan myös ylläpidettävyyden näkökulmasta suositeltava tapa. Nauhoituksia on mahdollista editoida tallennettuja vaiheita poistamalla tai muokkaamalla. Tämä kuitenkin koettiin haastavana ja muokkausta vaativissa tilanteissa oli sujuvampaa nauhoittaa testi uudelleen. Vaikka nauhoitustoiminnon käyttäminen oli helppoa, havaittiin onnistuneen testitapauksen toteuttamisen vaativan valmistelua ja suunnittelua odotettua enemmän. Lisäksi monet testit linkittyivät toisiinsa, joten nauhoitustyökalun lisävalikkojen (copy/ validate) käyttö oli mietittävä ennen nauhoitusta. Nämä ja kenttiin syötettävät tiedot tallentuivat seuraavaan vaiheeseen muuttujatiedoiksi, joilla voitiin ”ketjuttaa” testejä toisiinsa. Esimerkiksi ensimmäisessä testissä luodun kohteen yksilöivä tunnus voidaan käyttää seuraavassa testissä.

Testin tallennus suositellaan aloitettavan kotisivulta. Testitapausten suunnittelussa on hyvä rakentaa nauhoitettava prosessi pieniin hallittaviin osioihin, jotka toimivat myös itsenäisinä testeinä. Näin testien ylläpitäminen on selkeämpää ja mahdolliset korjaukset toteutetaan tiettyihin kokonaisuuksiin. Samaa automatisoitua testiä voidaan hyödyntää eri parametriarvoilla, mutta olemassa olevaa testiä ei voi muutoin muokata. Tämä ei tullut esille testitapausten kautta, mutta Microsoftin mukaan myös nauhoituksessa esiintyvät ilmoitukset ja infoviestit voidaan validoida. Kun vaiheet ovat selkeitä ja ennalta suunniteltu, itse testitapauksen toteuttamisessa menee arvioituna 2–10 minuuttia.

Toimittajalla oli testauksessa käytössä Microsoftin Lifecycle Services (LCS) pilvipohjainen yhteistyöportaali, jota asiakkaat ja heidän kumppaninsa käyttävät Microsoft Dynamics 365 for Finance and Operations -projektien hallintaan ja avustamiseen. Portaalin sisältä löytyy Business Process Modeler (BPM)-työkalu, jota voidaan käyttää testikirjastojen hallinnan sekä jakelun projektien ja käyttäjien kesken. BPM-työkalun avulla laaditut testitapaukset synkronoidaan osaksi Azure DevOpsia ja näin nauhoitteet tallentuvat LCS:n avulla suoraan oikeaan testitapaukseen.

Testitapaukset ladataan Azuresta Regression Suite Automation työkaluun, jossa ne muokataan automatisoiduiksi testeiksi. Muuntaminen tapahtuu valitsemalla valikosta testit. Painamalla valikosta löytyvää Uusi-nappia, jonka kautta valitaan testiajojen ja testiparametritiedostojen luonti. Testin viereen tulee näkyviin tiedosto, kun prosessi on valmis. Tämän jälkeen testin muuttujia voidaan muokata. Muokkaukset toteutetaan parametritiedostoon, joka on Excel-tiedostomuodossa. Excel-tiedosto muodostuu välilehdistä, jossa testissä ilmaantuneet osiot tallentuvat omana välilehtenä.

Valittujen testitapausten toiminnot voisi tiivistää CRUD-tyyppisiin toimintoihin. Nämä toiminnot tarkoittavat toimintoja, jossa tietue luodaan ja sen tallentuminen varmistetaan hakemalla tietue, johon toteutetaan muokkauksia tai poistoja.

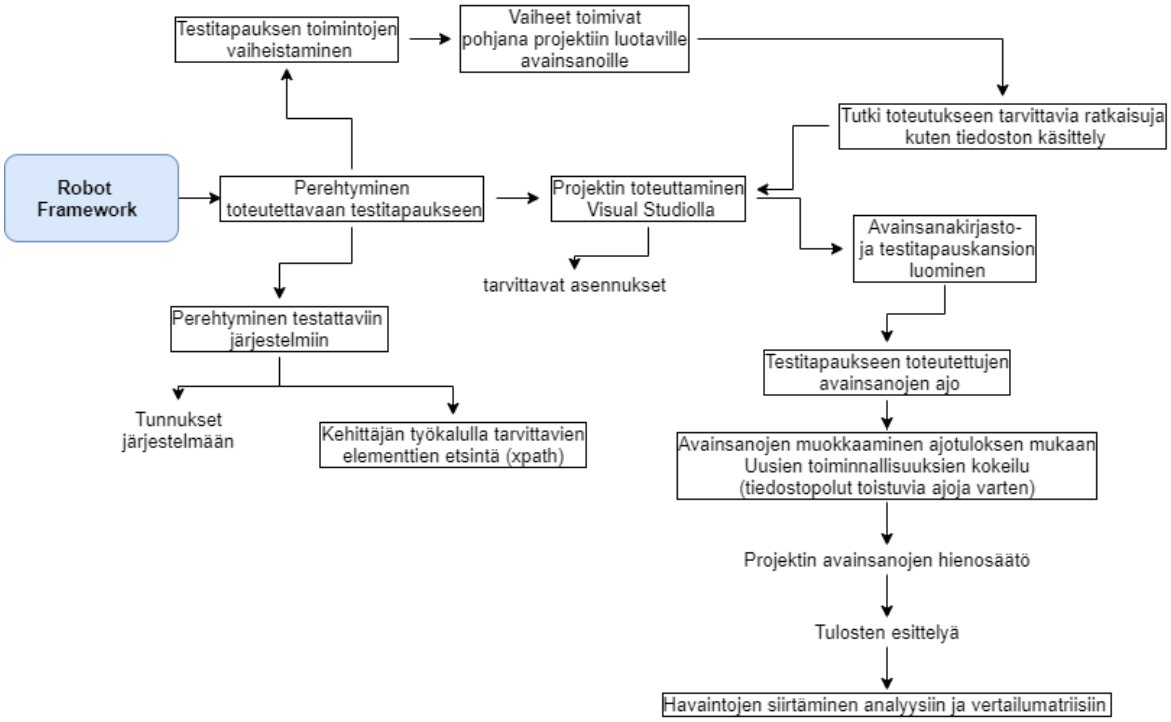
Nauhoituksessa tehdyt klikkaukset ja kenttien täyttämiset tallentuvat ruudulla näkyvään nauhoituskenttään vaiheiksi. Klikkauksen yhteydessä löytyy lisävalikko, jolla kenttään saadaan tallennettua tietoa. Tietoa voi validoida tai kopioida ja välittää muihin testeihin.

Laaditut testit voidaan ajaa joko komentorivin kautta tai valitsemalla työkalulla ajettavat testit ja painamalla ylävalikosta Run. Ajetuttujen testien raportit siirtyvät asetuksissa tehtyjen konfigurointitietojen avulla suoraan Azure DevOpsiin luettavaksi. Raporttitiedoissa testin ajotulos ilmaistaan selkeästi pass/fail tiedoilla. Testi muodostaa selkeän logitiedoston testin epäonnistuessa ja ottaa kuvakaappauksen erilliseen tiedostoon.

Valmiit testitapaukset voidaan tämän jälkeen ladata takaisin Azureen, josta ne ovat hyödynnettävissä. Kirjautumisprosessia ei voida testata, koska se tapahtuu thumbprintin kautta eikä kuvaa normaalia kirjautumisprosessia.

Robot Framework analysointiprosessin toteutus:

Tarkoitukseni oli hankkia kokemusta Robot Framework automatisointityökalusta sekä lisätietoa sen käyttämisestä ja soveltuvuudesta projektiin. Lähestyin testiautomaatiota prototyypin kautta. Alla kuvattu toteutuksen prosessi:



Kuva 12 Robot Framework toteutus ja analysointia

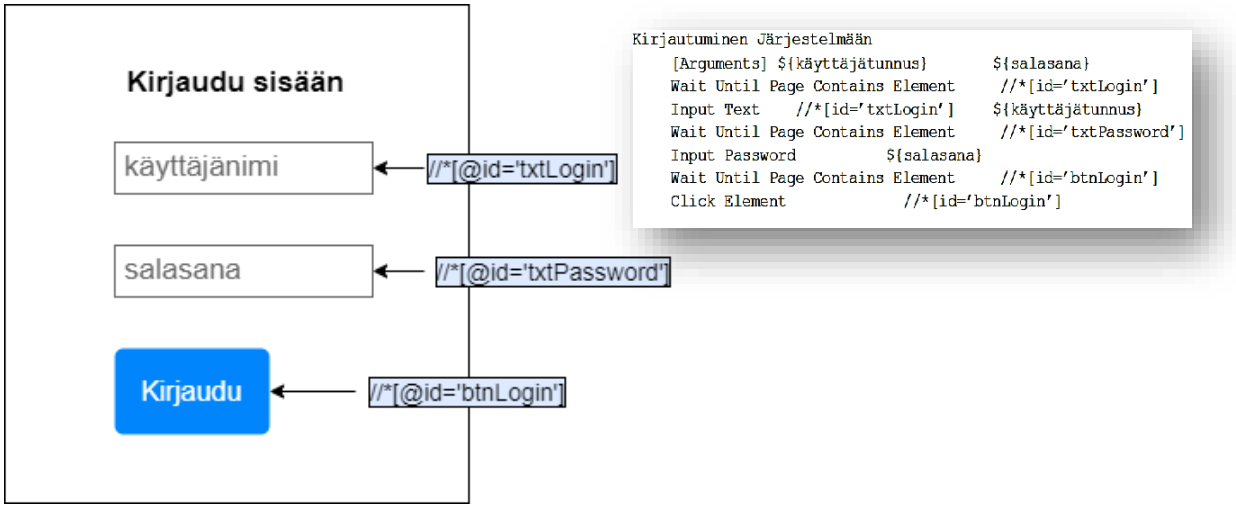
Valitussa testitapauksessa pyrin selvittämään, miten testausautomaation toteuttaminen Robot Frameworkin avulla onnistuu kahden määritetyn järjestelmän väliseen tiedonvaihtoon. Testitapaus käytiin ensin läpi liiketoiminnan asiantuntijan kanssa. Selvitettiin käyttäjän näkökulmasta onnistuneen prosessin ja tilanteen läpikulkua sekä mitä tarkastuksia toteutetaan. Demon perusteella siirtyminen kesti vähintään 15 minuuttia. Testi päätettiin jakaa kahteen osaan: testidatan luonti laskutusjärjestelmässä sekä sen käsittely toisessa järjestelmässä, johon sisältyy myös uusi ylitys laskutusjärjestelmään.

Demon pohjalta testitapaukset jaoteltiin vaiheisiin, jotka toimivat samalla listana toteutettavista liiketoiminta-avainsanoista. Laadittavat testit ovat luonteeltaan rakentavia tai positiivisia testejä, joissa sovellus toteuttaa suunnitellut toiminnot.

Testidatan luonnissa perustettiin projektiin uusi avainsanakirjasto, johon laadittiin tässä ympäristössä toteutettavia toimintoja. Tämä helpottaa jatkossa avainsanojen löytämistä ja käytettävyyttä. Avainsanojen muodostamisessa hyödynnettiin aiemmissa testitapauksissa

luotuja avainsanoja. Testiympäristön alkuasetus (test setup) - avainsanoja muokattiin uuteen sivustoon sopivaksi. Testin alussa siirrytään laskutusjärjestelmän testausympäristöön, jonne aiemmin oli luotu sisäänkirjautumista varten testikäyttäjätunnukset.

Testit toteutetaan Googlen Chrome selaimessa, jonka automatisoimiseen käytetään SeleniumLibrarya. Kyseisen kirjaston avainsanat toimivat sivun elementtien kanssa muuttujatiedoilla, joista käytetään nimitystä lokaattori. Lokaattoritieto paikannetaan sivustolta käyttämällä selaimen kehittäjätyökalua, joka näyttää DOM rakenteen. Yleensä työkalulla löydettyä elementtiä ei käytetä sellaisenaan vaan rakennetta on muokattava sopivaan xpath muotoon. Muodon luomista ja hakua voidaan toteuttaa Chromepath-lisäosalla. Kuvassa 13 on kuvattu selaimessa näkyvät elementit, joihin halutaan kohdistaa toimintoja. Elementin vieressä oleva rivi kertoo elementin xpathin, joka on löydetty kehitystyökalulla. Vieressä näkyy toteutettu avainsana, jossa on käytetty Robot Frameworkin sisäänrakennettuja ja ulkopuolisia testikirjastoja, jotka sisältävät matalan tason avainsanoja.



Kuva 13 Esimerkki kirjautumisesta

Avainsanan käyttäminen tapahtuu lisäämällä avainsanan perään muuttujatiedot:

```
Kirjautuminen Järjestelmään   käyttäjätunnukseni ${SALASANA}
```

Salasanan syöttäminen tapahtuu syöttämällä komentoriviin loppuun, jotta salasanaa ei tarvitse kirjoittaa osaksi koodia eikä tieto siirry näkyville raporttiajoon.

```
-v SALASANA: käyttäjän_salasana .
```

Ensimmäisessä testissä toteutetut vaiheet: uuden ostolaskun luominen, tiliöinnin lisääminen, laskun validoiminen, laskun siirtäminen siirtovalmiiksi sekä laskun lähetys testattavaan järjestelmään. Testien työstämiseen ja viimeistelyyn meni kolme työpäivää, ajossa meni keskimäärin noin kolme minuuttia.

Toisessa testiosassa toteutetut vaiheet: aiemmin luodun laskun haku järjestelmästä, laskutietojen tarkistaminen, laskun kirjaus, laskun tietojen tarkistus ja tietojen siirtymisen tarkistaminen laskutusjärjestelmässä. Testien työstämiseen ja viimeistelyyn meni neljä työpäivää, ajossa meni keskimäärin noin kahdeksan minuuttia.

Toteutuksessa havaitut haasteelliset/huomioitavat tilanteet:

1) Toteutuksessa huomattiin jo kirjautumisessa laskutusjärjestelmään, että useita istuntoja ei voi olla käytössä samaan aikaan. Toteutettu avainsana, joka tarkistaa kirjautumisen yhteydessä mahdollisen ilmoituselementin ilmestymisen. Jos elementti havaitaan testissä, muodostuu raporttitieto, jossa kerrotaan testin kaatuneen järjestelmässä olevaan istuntorajoitukseen. Tällöin järjestelmästä pitää kirjautua pois manuaalisesti ja toteuttaa ajo uudelleen. Useamman istunnon rajoitus on hyvä pitää mielessä jatkossa rinnakkaisajoja toteuttaessa.

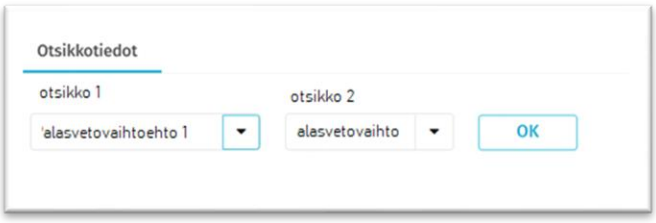
2) Seuraava haaste oli shadow DOM rakenteen sisällä olevien elementtien klikkaaminen, johon pelkällä Robot Framworkin avainsanalla Click Element ei päästy käsiksi. Robot Framework ei löytänyt elementtiä kyseisen rakenteen takaa, vaikka xpath-kyselytyökalulla Chromepathillä annettu xpath-elementti löytyi. Tilanteessa hyödynnettiin omassa python-tiedostossa olevaa koodia ja Selenium WebDriverin ominaisuutta. Käyttämällä Javascriptin funktioilla päästiin kiinni haluttuun elementtiin, jonka jälkeen avainsanalla klikattiin elementtiä.

3) Testattavassa järjestelmässä pakollisena tietona on pdf-tiedoston lisääminen. Se valitaan manuaalisesti tiedostonhallinnasta. Erillinen Windows-näkymä viittasi alustavasti siihen, että tiedoston lisäämiseen jouduttaisiin käyttämään pywinautoa, joka on tarkoitettu Windows-ohjelmien automatisoimiseen. Pywinauto asennetaan erikseen komentorivin kautta. Jotta testiin tallennettu tieto saadaan tallennettua, selvityksissä tuli esille Robot Frameworkin ulkopuolinen kirjasto SeleniumLibrary. Sieltä löytyi avainsana Choose File, jolla saadaan yksinkertaisemmin toteutettua tiedoston lisäys. Avainsanan avulla annettu tiedostopolku lisättävään tiedostoon toimii syötettävänä elementtinä ja lisää sen file-tyyppiseen attribuuttiin.

4) Voidaan generoida testidataa ajoja varten: toteutettu avainsana, jossa luodaan yksilöivä tunnus tietueelle:

```
${current_date}=    Get Current Date
${date}=      Convert Date     ${current_date}    result_format=%d.%m.%Y%H:%M
${id_number}=    Catenate    SEPARATOR=    TA_    ${date}
```

5) Toistuviin toimintoihin luodut avainsanat pyrittiin rakentamaan mahdollisimman yleisiksi, jotta niitä pystytään hyödyntämään myös muissa tietuetyyppien luonnissa käyttämällä samaa avainsanaa eri argumenttitiedoilla.



Kuva 14 Alasvetovalikon automatisointi

Esim. Kuvassa 14 näkyvässä tilanteessa otsikkotietojen alla näkyvät alasvetovalikot, jotka sisältävät eri tietuetyypit. Tarkoituksen olisi tulevaisuudessa luoda eri tietuetyyppejä, joten tästä tehty ns. yleinen avainsana, jonka kanssa käytetään kahta atribuuttitietoa. Avainsana pitää sisällään alemman tason avainsanoja, joilla käsitellään haluttua alasvetovalikkoa ja valitaan siitä haluttu tieto. Otsikkotiedot ovat ns. kovakoodattu osaksi avainsanaa, mutta alasvetovaihtoehdot lisätään käyttäjän haluamalla tavalla. Esimerkiksi avainsanaa käytetään:

```
Aloita Tietueen Luonti    otsikko 1 vaihtoehto    otsikko2 haluttu vaihtoehto
```

6) Testissä päädytty tallentamaan tietoja tekstitiedostoihin. Ensimmäisessä osiossa luodut tietuetunnukset tallennetaan omaan testin sisällä olevaan tiedostoon, josta ensimmäisellä rivillä oleva tietuetunnus otetaan käsittelyyn seuraavassa osiossa. Ensimmäisessä vaiheessa tallennetut tietueet siirtyvät tietyn ajan kuluessa toiseen järjestelmään, jossa ne näkyvät käsittelemättömät tietueet kansiossa. Siirtyessä järjestelmästä toiseen, tietueet saavat numerosarjan, jonka viimeinen osio muodostuu juoksevasta numeroinnista. Koska kirjaustieto avataan valitsemalla haluttu kansionumero, on erilliseen tekstitiedostoon tallennettu testissä käytettävä kansionumero. Kun tietyt vaiheet testissä on suoritettu onnistuneesti läpi, tekstitiedostoja päivitetään: käytetty tietuetunnus poistetaan ja testissä käytettävää kansionumeroa päivitetään, jotta testi menee läpi muissakin ajoissa.

7) Kenttien validoinnit toteutettiin Should Be Equal avainsanalla: Tarkistuksen kohteen oleva kentän tieto otettiin talteen muuttujaan Get text tai Get value avainsanalla ja sitä vertailtiin asetettuun arvoon Should Be Equal avainsanan kanssa.

8) Toisessa järjestelmässä scrollattavan elementin löytäminen oli haastavaa, koska se näytti vain viittaavaan elementtiin, jossa scrollattava elementti sijaitsee. Sen sijaan jouduttiin minikoimaan näppäimistöä eli taulukkoa selattiin tabulaattorilla: Press Keys | None | TAB

Nämä havainnot toimivat esimerkkeinä miten sujuvasti Robot Framework taipuu eri tilanteissa, joita testitapauksen automatisoinnissa voi tulla esille. Avainsanojen käytöille ei ole sinänsä muita rajoitteita kuin syntaksi, joten tarpeiden mukaisesti avainsanoja voi muokata ns. yleiskäyttöisemmiksi. Testidataa voidaan muokata halutunlaiseen muotoon, jotta se palvelee paremmin testitapauksen tarpeita. Kirjastojen avulla on saatavissa keinoja erilaisten tiedostojen ja datan käsittelyyn sekä soveltavaa tietoa, kuinka avainsanoja voidaan kehittää haluttuun suuntaan. Myös asiantuntevissa keskustelupalstoilla on joskus tarjolla valmiita ratkaisuja lähtökohdaksi.

Robot Frameworkin käyttöön löytyy paljon tukea ja materiaalia GitHubin ja Stackoverflown sivustoilta. Lisäksi omalta sivustolta löytyy dokumentaatiota tärkeimmistä sisällöistä, joilla testiautomaatiossa pääsee käyntiin. Lisäksi vuosittain järjestettävä Robocon jakaa työkalun uusimmat tiedot ja käyttökokemukset. Robot Frameworkia on helppo laajentaa tarpeiden mukaan käyttämällä eri kirjastoja sekä yhdistämällä pythonia, jolla saadaan lisää kustomoituja avainsanoja sekä testidataa voidaan generoida myös juuri organisaation tarpeita vastaavaksi.

Liiketoiminta-avainsanat voidaan nimetä juuri halutulla tavalla, joka auttaa myös raportoinnissa tulevien virheilmoitusten selvittämisessä. Avainsanat kuvaavat toimintaa lauseen muodossa, joten sen tulkitsemiseen ei vaadita teknistä osaamista. Toiminta, jossa virhe tapahtui, tulee tällöin selkeästi ilmi muodostuneessa ajoraportissa. Toki syy kaatumiseen pitää selvittää manuaalisesti, mutta muodostuneet raportit antavat kuvakaappauksineen viitteitä kaatumisen syistä.

5.3 Analyysin havaintojen koonti

Regression Suite Automation Tool on lähtökohdiltaan helposti lähestyttävä työkalu. Sovelluksesta löytyvä nauhoitustyökalu tallentaa käyttäjän tekemät toiminnot vaiheiksi, jotka varsinaisella automatisointi työkalulla käännetään automatisoiduiksi testeiksi. Testissä käytettäviä muuttujia muokataan Excel-tiedostossa, jonka käyttäminen on tullut tutuksi muissa yhteyksissä kuin testien automatisoinnissa. Työkalun käyttäminen ei edellytä koodaustaitoja. Erikseen ladattava työkalu on helppo asentaa asennusohjelman avulla. RSAT tarvitsee toimiakseen Seleniumin sekä web-ajurin, mutta asennusohjelma asentaa ne tarvittaessa. Alkuasetuksien haasteena on tarvittavien yhteyksien muodostaminen ja varmistaminen testattavaan järjestelmään sekä Azuren Test Planiin. Sisäänkirjautuminen järjestelmään tapahtuu thumbprint-tiedon avulla, joten RSAT:lla ei voida automatisoida varsinaista kirjautumisprosessia.

RSAT-työkalun käyttöä voidaan Microsoftin mukaan roolittaa. Esimerkiksi testipäällikkö suunnittelee testitapaukset Azureen ja jakaa nauhoitettavat testitapaukset. Nauhoitukset toteuttaisi liiketoiminnan ammattilainen, joka tuntee toiminnot liiketoiminnan näkökulmasta. Kun halutut testit on nauhoitettu, testipäällikkö jatkaa testien automatisointia RSAT:lla.

Työkalu on integroitu Microsoft Azure DevOpsin kanssa testien suorittamista, raportointia ja tutkintaa varten. Työkalulla ajetut testitapausten ajotulokset siirtyvät automaattisesti Azureen työkalun asetusten myötä. Raportit ovat selkeitä: värikoodein testi muuttuu joko vihreäksi (pass) tai punaiseksi (fail). Yksityiskohtaisemmat tiedot testin ajon vaiheista ja mahdollisen kaatumisen syistä löytyvät avaamalla testitapauksen. Tieto on sanallisessa muodossa ja epäonnistuneesta kohdasta muodostuu kuvakaappaus.

Testien nauhoittaminen ja niiden automatisoiminen ovat vaiheina helppoja, mutta onnistuneen testitapauksen toteuttaminen edellyttää eri vaiheet huomioivaa suunnittelua. Nauhoitettavat testitapaukset suositellaan pitämään lyhyinä, jotta testejä on helpompi ylläpitää, muuntaa ja käyttää uudelleen. Samaa testiä voi käyttää uudelleen vaihtamalla Excel-tiedostossa olevia muuttujatietoja. Itse testin sisältöä ei voida muokata, vaan tällöin tulee toteuttaa uusi nauhoitus. Testien nauhoitukset suositellaan aloittamaan päänäkymältä, joten testitapausten tulee toimia myös itsenäisinä testeinä. Käyttökokemuksen perusteella nauhoituksen muokkaaminen oli haasteellista, usein sujuvinta oli nauhoittaa tilanne uudelleen. Etukäteen on tunnistettava vaiheet, joissa käytetään nauhoitustyökalun lisävalintanappeja.

RSAT-työkalu on spesifi Microsoft Finance & Operations-sovelluksen hyväksymistestaamiseen. Nauhoitustyökalu mahdollistaa sen, että aikaa ei kulu elementtien tai ominaisuuksien löytämiseen. Testitapaukset on rajattava juuri tähän järjestelmään ja niiden tulisi keskittyä perustoimintoihin, joita järjestelmässä käytetään.

Tämänhetkisen tiedon pohjalta RSAT:a ei voi suoraan yhdistää jatkuvaan integraatioputkeen (CI-putki). Keskustelupalstojen mukaan siitä on kuitenkin mahdollisuus rakentaa build pipeline. Tätä ei ole analyysivaiheessa testattu. Parametritiedostojen käsittely build pipelinessa vaatisi lisätutkimuksia.

Opinnäytetyöni toimeksiantaja on automatisoinut testitapauksia **Robot Frameworkilla** ja dokumentoinut käyttökokemuksia työkalusta Confluenceen. Tietoa löytyy asennuksesta

sekä havaituista haasteista ja niiden ratkaisuista. Opitun tiedon lisäksi projektit ovat luoneet pohjan avainsanakirjastoille, joita hyödynnettiin myös tässä proof of conceptin rakentamisessa.

Robot Framework on monipuolinen käyttöjärjestelmästä ja sovelluksesta riippumaton testiautomaatiotyökalu, jossa taulukkomaista syntaksirakennetta hyödynnetään testitapausten ja avainsanojen luomisessa. Avainsanat ovat luettavissa ja niiden toiminta kuvataan lauseen muodossa, joten avainsanat itsessään kuvaavat toimintaa ja niiden ymmärtäminen on helppoa ja käyttöä ohjaavaa. Testitapauksia ja avainsanoja on helppo laajentaa tarpeiden mukaan käyttämällä eri kirjastoja ja yhdistämällä pythonia. Pythonilla saadaan lisää kustomoituja avainsanoja ja voidaan generoida testidataa juuri organisaation tarpeita vastaavaksi. Liiketoiminnan avainsanat voidaan nimetä organisaation käyttämien termien suuntaiseksi.

Testitapaukset kirjoitetaan, jäsennetään sekä ylläpidetään valitussa tekstieditorissa. Monissa tekstieditoreissa voi hyödyntää erikseen ladattavia lisäosia, jotka täydentävät mm. koodia automaattisesti tai korostavat syntaksia värein. Lisäosat tukevat koodin hallinnassa ja auttavat havaitsemaan mahdolliset virheet. Testitapauksiin voidaan liittää luokitteleva Tags-tunniste, joka sujuvoittaa testiajoja ja testiraporttien lukemista.

Vaikka Robot Frameworkin syntaksi itsessään on helppoa, vie aikaa saada avainsanoista hyviä kokonaisuuksia. Avainsanojen rakentamiseen ja hienosäätöön kannattaa panostaa, jotta niitä voi toistaa testiajoissa ja liittää osaksi CI-putkeen. Toistuvien ominaisuuksien havainnointi ja yleistäminen muihin testitapauksiin sekä käyttöympäristöihin on mahdollisuus parantaa testauskykyä.

Elementtien löytäminen on haastavaa selainpohjaisia testitapauksia automatisoidessa. Järjestelmän kehittäjän koodaustapa vaikuttaa elementtien työstämiseen ja jatkotyöskentelyyn, jotta saadaan rakennettua robotin ymmärtämä tunniste. Tilannetta haasteellisuutta lisää dynaamiset sivut, joissa elementtien tunnukset ovat istuntokohtaisia.

Taulukko 2 Robot Framework ja RSAT vertailutaulukko

Parametrit	Kriteeri	Robot Framework	RSAT
Omaksuminen	Lisenssikustannukset	Avoimen lähdekoodin ohjelmisto, jonka ominaisuuksia kehitetään säännöllisesti käyttäjäpalautteen perusteella. Työkalun	Rakentuu kahdesta työkalusta: järjestelmään sisäänrakennettu nauhoitustyökalu ja ilmaiseksi

		käyttö tarjotaan ilmaiseksi myös yrityksille.	ladattava automatisointiohjelma.
	Käyttötuki, ohjeet	Hyvä. Sivustolla selkeät ohjeet sekä dokumentointia projektin aloittamisesta ja käytettävistä ulkopuolisista kirjastoista.	Kohtalainen. Projektien alkuun löytyy dokumentaatiota ja videoita testien toteuttamisesta. Lisäksi sisäinen keskustelupalsta käyttökokemusten jakamiseen ja ongelmatilanteiden selvittämiseen.
Skriptaus ja raportointi	Skriptauksen luontiaika	Projektin alussa testitapausten avainsanojen laatiminen vie aikaa, mutta jatkossa luotujen avainsanakirjastojen myötä prosessi nopeutuu.	Nauhoituksen ja automatisoinnin toteutus on nopea, mutta testin toteuttamisen suunnittelu edellyttää etukäteisvalmisteluja ja hienosäätöä, mikä vaatii enemmän aikaa.
	Skriptauskieli	Helppo syntaksi, jossa käytetään luettavia lauseita/sanoja avainsanojen luonnissa. Ominaisuuksia laajennettavissa Pythonilla tai Javalla.	Toteutettavat testitapaukset nauhoitetaan ja ne käännetään erillisellä työkalulla automaattisesti. Muuttujat käsitellään Excel-tiedostossa.
	Testiautomaatioviitekehys	avainsanapohjainen	lineaarinen, nauhoita ja toista -työkalu
	Mahdollisuus jatkuvaan integroimiseen	Kyllä	Ei
Työkalun käyttö	Käyttöjärjestelmätuki	Windows, OSX tai Linux	Windows 10
	Selaintuki	Firefox, Google Chrome, Internet Explorer, Edge, Safari, Opera, Android, Iphone, PhantomJS, HTMLUnit, HTMLUnit with Javascript	Microsoft Edge, Microsoft Internet Explorer, Google Chrome.

6 Pohdinta

ERP-järjestelmän testiautomaation suunnittelu- opinnäytetyön toimeksiantajalla oli tarve lisätä testauskapasiteettia/testauskykyä, jotta kyetään vastaamaan useiden kehitysiteraatioiden tuottamista toiminnallisuuksien testauksesta. Toteutuksen alla olevaa toiminnanohjausjärjestelmää olivat tähän mennessä hyväksymistestanneet toteuttaneet liiketoiminnan ammattilaiset, joiden ajankäyttö testauksiin oli rajallinen. Ketterän menetelmän mukaisesti testiautomaatiota haluttiin hyödyntää projektin regressiotestauksessa. Testiautomaatiotyökalun valinnan päätöksenteon tueksi haluttiin toteuttaa testiautomaation elinkaaren aloittava vaihe, jossa kartoitetaan sopiva työkalu sen toteuttamiseksi.

Erityisiä vaatimuksia tai rajoituksia ei asetettu analysoitaville työkaluille. ERP-toiminnanohjausjärjestelmän alustan ratkaisuja on kustomoitu vastaamaan organisaation tarpeita ja toiminnot sisältävät integraatioita, mikä on kriittistä liiketoiminnalle. Näihin toimintoihin liittyvien vaatimusten testaaminen on etenkin asiakkaan näkökulmasta prioriteettilistalla korkealla ja vaikuttaa testiautomaatiotyökalun vaatimuksiin.

Analyysin kohteeksi valitut automatisointityökalut edustavat kahta eri testiautomatisointi viitekehystä. Robot Framework edusti avainsanapohjaista ja RSAT lineaarista viitekehystä. Testiautomaation viitekehitykset osaltaan muodostivat ennakko-odotuksen automatisointityökalujen testien toteuttamiseen, suorittamiseen ja ylläpitoon. Analyysissa havaitut hyödyt ja haitat vastasivat näitä odotuksia. Teoriaosassa esitelty testausprosessi ja siihen liittyvät vaiheet huomioin analyysissani vaikuttavina osatekijöinä työkaluvalintaan. Erityisesti testausprosessin viimeistelyvaiheeseen liittyvässä dokumentoinnin tärkeys tuli vaikuttamaan analyysin pohjalta nostamiini huomioihin.

Analyysiprosessia kuvataan tarkemmin luvussa 5, jossa esitetään tarkemmin lähtökohdat opinnäytetyölle ja proof of conceptien nostamat havainnot. Molemmat analysoitavat työkalut soveltuivat testattavan järjestelmän automaatiotestaukseen. Testattavan järjestelmän toimittaja toteutti RSAT:lla testitapaukset, jotka käsittivät järjestelmän eri valikkojen sisältöön koskevia perustoimintoja (tietueiden käsittelyä). Nauhoitustyökalun ja varsinaisen testiautomaatiotyökalun suunnitelmalliseen käyttöön liittyi oppimisaika, jotta testeistä saatiin useampiin ajokertoihin sopivia kokonaisuuksia. Työkalun hyötynä on, ettei sen käyttö vaadi ohjelmointitaitoja ja käyttö mahdollisti nopean testin toteutuksen, kun työkalun käyttö oli omaksuttu ja testin kulku sovellettu työkaluun sopivaksi. Tämä tarkoitti, että testitapauksia oli tarpeen mukaan pilkottu pienemmiksi toimivaksi kokonaisuuksiksi testien ketjuttamisen onnistumiseksi. Mahdollisiksi esteiksi työkalun jatkokäyttämiselle ovat sen

jatkuva ylläpito. Toiminnanohjausjärjestelmä on edelleen kehityksen alla ja tämä tarkoittaa, että testit tulevat rikkoutumaan. Niiden korjaaminen tarkoittaa, että nauhoituksia täytyy pahimmassa tapauksessa toteuttaa kokonaan uudelleen.

Robot Frameworkia koskevan havaintoanalyysin pohjana toimi toteuttamani proof of concept, joka samalla toi käyttöön uusia resurssi- ja avainsanakirjastoja toimeksiantajan käyttöön. Tiedostojen luonnissa hyödynnettiin avainsanapohjaiseen viitekehykseen kuuluvia kolmea avainsanatyyppiä. Perustason ja apuohjelman toimintoihin liittyvät avainsanat liitettiin osaksi käyttäjän luomiin avainsanatyyppeihin. Näitä avainsanatyyppejä hyödynnettiin osana toista luotua ns. ylemmän tason liiketoiminnan avainsanaa. Datan käsittelyä varten toteutin Pythonilla funktiot, jota siihen liitettyä avainsanaa kutsumalla toteuttaa taustalla halutun datan muokkauksen. Nämä toteutukset mahdollistivat liiketoiminnan asettavat vaatimukset asetetulle testitapaukselle. Testiin tarvittavan avainsanojen luominen ja niiden testaus vie aikaa, mutta siihen kannattaa panostaa. Toteutettuja avainsanoja voi käyttää osana käyttäjän luomia avainsanoja sekä hyödyntää muissa testitapauksissa. Testit voi suorittaa ajamalla ne komentoriviltä hyödyntämällä testeihin liitettyjä tags-tunnisteita. Tags-tunnisteet auttavat myös yksilöimään testit, joita halutaan jatkossa ajaa jatkuvassa integraatioputkessa. Testien ylläpitäminen Robot Frameworkilla on helppoa tekemällä halutut muutokset tekstieditoriin.

Työkalun soveltamisessa oli eroja, jotka näkyivät jo työkaluille suunnatuissa testitapauksissa. RSAT:lla toteutetut testitapaukset olivat suunnattu pelkästään testattavaan järjestelmään ja niissä toiminnot ovat selkeitä tietueen käsittelytapauksia. Robot Frameworkilla testi kohdistui testattavan järjestelmän kustomoituun ominaisuuteen, jossa eri järjestelmien välillä tapahtuu integraatio. Laaditut testitapaukset pitävät sisällään toimeksiantajan vaatimukset järjestelmälle. Nämä vaatimukset tarkentuvat iteraatioiden aikana ja tuovat eri tarpeita toteuttaa toiminnallisuuden testausta. Toiminnallisuuden toteutuksen onnistumista voi tarkastella matalan tai korkean tason testien kautta. Mielestäni RSAT:lla toteutetut testit koostuvat useista matalan tason testeistä, jotka muodostavat liiketoiminnan edustajan tekemää liiketoiminnan prosessia järjestelmällä. Robot Frameworkin avainsanojen avulla voi rakentaa kokonaisuuksia, jotka vastaavat manuaalitestauksen tuottamaan älykkyyttä. Testitapauksen sisältö ja sen prioriteetit ratkaisevat tarvetason testiautomaatiotyökalulle. Käytettävät resurssit tuovat omat rajoitteet valintaprosessille. Toteuttamani valintamatriisi voi auttaa rajaamaan työkaluja, mutta testitapauksen asettamat rajat/kyvyt työkalulle ovat tapauskohtaisia.

Testiautomaation elinkaaren ensimmäisen vaiheen tarkoituksena on myös hallita työkaluille asetettuja odotuksia. Opinnäytetyön yhtenä tavoitteena oli tehostaa liiketoiminnan

edustajien ajankäyttöä, mutta tämän toteutuksen aikana ei ehditä näkemään testiautomaation vaikutuksia. Ensisijaisesti testiautomaatiolla pyritään edistämään koodin laatua automatisoimalla regressiotestitapaukset. Kehitystiimi saa palautetta halutulla syklillä ajetuista testeistä ja mahdolliset virhetilanteet havaitaan aikaisemmassa vaiheessa. Näin ollen liiketoiminnan edustajien toteuttama manuaalinen testaus on tärkeässä asemassa, mutta jatkossa automatisoidut testit auttavat heitä keskittämään huomionsa uusien toimintojen arviointiin. Hyödyntämällä Robot Frameworkia manuaalitestaajille voi luoda valmiita testitilanteita eli heidän ei tarvitse käyttää aikaansa alustamalla manuaalitestauksia varten tietueita, joita sen hetkinen käsittely vaatii.

Automatisoidut testit tarvitsevat ylläpitoa, jotta testit ovat vaatimusten sekä koodissa tapahtuneiden muutosten suhteen ajan tasalla. Testiraporttien tulosten analysoiminen esiintyneessä virhetilanteessa tuo esille joko mahdollisen bugin koodissa tai automatisoitua testiä tarvitsee päivittää. Yleensä tällöin on selainpohjaisissa automaatiotesteissä, että robotti ei löydä käsiteltävän elementin muuttunutta lokaattoritietoa. Tämä muuttunut tieto voi olla liiketoiminnan edustajien parannusehdotuksen toteutettu seuraus. Muutoksen toteutus automatisoituun testiin voi tarkoittaa, että manuaalitestaajien testauslistalta väheni yksi testattava kohta. Tällöin on varmistettava, että muutos testataan halutulla tarkkuudella.

Prosessin aikana ja analyysin pohjalta voin todeta, että Robot Framework on vertailun perusteella testauksen ja ohjelmiston hyvää laatua tukeva automatisointityökalu. Robot Frameworkin rakenne ja toiminnot mahdollistavat tarkemman keinon automatisoida manuaalitestaajien toteuttamia toimintoja kuin RSAT. Opinnäytetyön toimeksiantajan priorisoitavat testitapaukset liittyvät eri järjestelmien integraatioihin ja valitettavasti RSAT:lla ei voida toteuttaa liiketoiminnalle kriittisiä testitapauksia. Käyttöjärjestelmästä ja sovelluksesta riippumaton Robot Frameworkia voidaan proof of conceptin pohjalta käyttää näissä testitapauksissa ja testitapaukset voidaan toteuttaa organisaation haluamalla tarkkuudella. Robot Framework voi auttaa muodostamaan dataa ajetuista testitapauksista sekä hyödyntää sen logitiedon valmista dokumentoitua kuvausta testattavista ominaisuuksista ja millä tasolla niitä on testattu.

Robot Frameworkilla testidataa voidaan muokata halutunlaiseen muotoon, jotta se palvelee paremmin testitapauksen tarpeita. Kirjastojen avulla on saatavissa keinoja erilaisten tiedostojen ja datan käsittelyyn sekä soveltavaa tietoa, kuinka avainsanoja voidaan kehittää haluttuun suuntaan. Työkalu taipuu moneen, mutta luotettavien tulosten saamiseksi vaaditaan suunnittelua. Huolellisesti laaditut avainsana- ja resurssikirjastot käsittelevät dataa määritellyissä olosuhteissa. Myös tarkka avainsanojen nimeäminen lisää testitapauksen informatiivisuutta, jonka avulla voi todeta testitapauksen vastaavan vaatimuksia.

RSAT toimii rajatussa olosuhteessa ja antaa palautteen, jos kentän validoiminen tai tietty toiminto epäonnistuu.

RSAT on mielestäni suunnattu käyttäjille, jotka eivät ole huomioineet testausstrategiassaan testiautomaation mahdollisuutta. Työkalu on vaiheiltaan helppokäyttöinen ja ymmärrettävä. Koodaamista ei tarvitse käyttää, joten se sopii eri tasoisille käyttäjille. Toimeksiantajalla on jo kokemusta Robot Frameworkin käytöstä, joten organisaatiossa on valmius työkalun soveltamiseen eri käyttökohteisiin. Robot Frameworkilla avainsanoihin ja sitä kautta toimintoihin voi liittää vasteaika, joten siten arvioitavissa toteutuuko haluttu toiminto kohtuullisessa ajassa. Robot Frameworkilla aiemmin toteutetut testitapaukset edesauttavat jatkossa muiden testitapausten rakentamista. Usein testit käsittelevät samankaltaisia toimintoja kuten kenttien täyttämistä ja niiden aiheuttamien mahdollisten muutosten tarkistamista. Yleistettäviä ja yleiskäyttöisiä avainsanoja kannattaa laatia toistuviin tilanteisiin. Se auttaa uusien iteraatioiden tuomien toteutusten siirtämisen osaksi testiautomaatiota hyvinkin nopealla aikataululla.

Tärkein ero työkalujen välillä liittyy niiden käytettävyyteen. Rakenteensa vuoksi RSAT:a ei voi yhdistää osaksi jatkuvaa integraatioputkea. Robot Frameworkilla laaditut testit voi ajaa tags-tunnisteen perusteella ajastetusti tai versiohallinnan muutosten mukaan integraatioputkessa. Tämän avulla testien ajot ovat automatisoituja; ajoraportit antavat palautetta ohjelmiston tilasta ja mahdollisista korjattavista vioista. Hanke sisältää useita iteraatioita, joiden myötä toiminnallisuuksien määrä kasvaa. Robot Frameworkilla voi suorittaa lukuisia regressiotestikierroksia.

Asiakkaalla ja ohjelmiston toimittajalla saattaa olla erilaisia odotuksia, tavoitteita ja keinoja testauksen suhteen. Vaatimusten pohjalta laaditun testitapauksen toteutus edellyttää keskustelua ja sopimista, mikä on projektissa käytettävä testausprosessi. Prosessissa määritetään toteutettava testausstrategia, jossa määritetään, mitä ja miten testataan ja kenen toimesta. Ketterässä kehityksessä on tärkeää erityisesti suunnitelman tarkistaminen ja päivittäminen. Hyvänä lähtökohtana toimii testitapauksien sisäänrakennetut vaatimukset, jolloin läpäisseet testit ovat merkki järjestelmän riittävästä virheettömyydestä. Toisaalta jos testitapaukset laaditaan vain asiakkaan vaatimusten pohjalta, niin havaitaanko testauksessa uusia virheitä? Manuaalisella testauksella voi joskus havaita parhaiten virheet ja kehitysehdotukset sovelluksessa.

ERP- järjestelmän testausautomaatiomenetelmien analysointia -opinnäytetyöhöni ei ollut kokonaiskatsaus testiautomaatiomenetelmien analysointiin. Opinnäytetyössäni keskityin

tarkastelemaan toimeksiantajan rajaamia Regression Suite Automation Tool ja Robot Framework - testiautomaatiotyökaluja. Analysoinnissa nostamani asiat RSAT työkalun käytöstä liittyvät tiukasti valittuun ERP-järjestelmään, mutta havaintoni Robot Frameworkista on sovellettavissa myös muihin järjestelmiin yleisesti.

Lähteet

Agarwal, B. B., Gupta, M. & Tayal, S. P. 2009. Software Engineering and Testing. Jones & Bartlett Learning. Luettu 31.3.2021.

Axelrod, A. 2018. Complete Guide to Test Automation: Techniques, Practices, and Patterns for Building and Maintaining Effective Software Projects. Apress. Luettavissa: https://learning.oreilly.com/library/view/complete-guide-to/9781484238325/html/453429_1_En_6_Chapter.xhtml. Luettu 31.3.2021

Avantiico s.a. Automation Testing Toolbox. Katsottavissa: https://www.avantiico.com/services/implementation-and-audit/regression-suite-automation-tool/. Katsottu: 2.3.2021.

Bajaj, H. 2018.Choosing the right automation tool and framework is critical to project success. Infosys-julkaisu. Luettavissa: https://www.infosys.com/services/it-services/white-papers/documents/choosing-right-automation-tool.pdf. Luettu 31.3.2021.

Bisht, S. 2013. Robot Framework Test Automation. Packt Publishing. Luettu 31.3.2021.

Crispin, L. & Gregory, J. 2009. Agile testing: A practical guide for testers and agile teams. Addison-Wesley. Upper Saddle River (NJ). Luettavissa: https://learning.oreilly.com/library/view/agile-testing-a/9780321616944/ch14.html.Luettu 5.2.2021.

Dustin, E., Garrett, T. & Gauf, B. 2009. Implementing Automated Software Testing: How to Save Time and Lower Costs While Raising Quality. Addison-Wesley Professional. Luettavissa: https://learning.oreilly.com/library/view/Implementing+Automated+Software+Testing:+How+to+Save+Time+and+Lower+Costs+While+Raising+Quality/9780321619600/ch06.html. Luettu 31.3.2021

Fewster, M. & Graham, D. 1999. Software test automation: Effective use of test execution tools. Harlow: Addison-Wesley. Luettu 3.4.2021.

Gupta, R. 2016. Agile Automation and Unified Funtional Testing. Pearson Education India. Luettavissa: https://learning.oreilly.com/library/view/agile-automation-and/9789332578944/xhtml/chapter008.xhtml. Luettu 19.4.2021.

Hambling, B., Morgan, P. & Samaroo, A. 2010. SOFTWARE TESTING. British Informatics Society Limited. Swindon. Luettavissa: https://learning.oreilly.com/library/view/software-testing/9781906124762/9781906124762_introduction.html. Luettu 3.4.2021.

Hanna, M., El-Haggar, N. & Mostafa, S.2014. A Review of Scripting Techniques Used in Automated Software Testing. International Journal of Advanced Computer Science and Applications. Luettavissa: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.428.7456&rep=rep1&type=pdf. Luettu 3.4.2021

Kasurinen, J. P. 2013. Ohjelmistotestauksen käsikirja. Jyväskylä: Docendo. Luettu 5.2.2021.

Lindroos, J. & Lohivesi, K. 2010. Onnistu strategiassa. 3. uud. p. Helsinki: WSOYpro. Luettu 5.2.2021.

Microsoft 2020a. Service update availability. Luettavissa: https://docs.microsoft.com/fi-fi/dynamics365/fin-ops-core/fin-ops/get-started/public-preview-releases. Luettu 5.2.2021.

Microsoft 2021a. Regression Suite Automation Tool. Luettavissa: https://docs.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/perf-test/rsat/rsat-overview#intended-usage-and-test-classification. Luettu 5.2.2021.

Microsoft 2021b. Use the Regression suite automation tool (RSAT). Luettavissa: https://docs.microsoft.com/en-us/dynamics365/fin-ops-core/dev-itpro/perf-test/rsat/rsat-run#process-compliance. Luettu 5.2.2021.

Microsoft Dynamics 365 30.4.2020. How to create a test plan in Azure DevOps to use with the Regression suite automation tool (RSAT). YouTube-video. Katsottavissa: https://www.youtube.com/watch?v=3jIuBleAnQk&list=PLcakwueIHoT_SYfIaPGoOhloFoCXiUSyW. Katsottu 2.3.2021.

Microsoft Dynamics 365 4.5.2020. How to use task recorder to create a test case for the Regression suite automation tool (RSAT). YouTube-video. Katsottavissa: https://www.youtube.com/watch?v=bBr4BXAxTNI&list=PLcakwueIHoT_SYfIaPGoOhloFoCXiUSyW. Katsottu 2.3.2021.

Microsoft Dynamics 365 7.5.2020. How to use the Regression suite automation tool (RSAT). YouTube-video. Katsottavissa: https://www.youtube.com/watch?v=uhN9JItzGAk&list=PLcakwueIHoT_SYfIaPGoOhloFoCXiUSyW. Katsottu 2.3.2021.

Mosley, D. & Posey, B. 2002. Just Enough Software Test Automation. Pearson. Luettavissa: https://learning.oreilly.com/library/view/just-enough-software/0130084689/. Luettu 4.3.2021.

Myers, G. J., Badgett, T. & Sandler, C. 2012. The art of software testing. 3rd ed. Hoboken, N. J.: John Wiley & Sons. Luettu 5.3.2021.

Osborne, M. 6.7.2018. Modernizing the way we update Dynamics 365. Luettavissa: https://cloudblogs.microsoft.com/dynamics365/bdm/2018/07/06/modernizing-the-way-we-update-dynamics-365/. Luettu 5.3.2021.

Patton, R. 2006. Software testing. 2nd ed. Indianapolis: Sams Pub.Luettavissa: https://learning.oreilly.com/library/view/software-testing-second/0672327988/ch15.html#ch15lev1sec. Luettu: 15.3.2021.

Raulamo-Jurvanen, P., Mäntylä, M., & Garousi, V.2017. Choosing the right test automation tool: A grey literature review of practitioner sources. Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering.Luettavissa: https://pureadmin.qub.ac.uk/ws/portalfiles/portal/178890185/ChoosingTheRightTestAutomationTool_a_GLR_preprint.pdf. Luettu:31.3.2021.

Robot Framework 2021. Robot Framework User Guide.Luettavissa: http://robotframework.org/robotframework/latest/RobotFrameworkUserGuide.html. Luettu 5.3.2021.