LAB University of Applied Sciences

Testiautomaatiokehykset ja -alustat

Laadukkaan testiautomaatiokehyksen valinta

LAB-ammattikorkeakoulu
Insinööri (AMK)
2022
Marianna Syrjälä

Tiivistelmä

Tekijä(t)	Julkaisun laji	Valmistumisaika
Marianna Syrjälä	Opinnäytetyö, AMK	2022
	Sivumäärä	
	37	

Työn nimi

Testiautomaatiokehykset ja -alustat

Laadukkaan testiautomaatiokehyksen valinta

Tutkinto ja koulutusala

Insinööri (AMK), tieto- ja viestintätekniikka

Toimeksiantajaorganisaatio (jos opinnäytetyöllä on toimeksiantaja)

Sade Innovations OY

Tiivistelmä

Työn tavoitteena oli tutkia ja vertailla tarjolla olevia testiautomaatiokehyksiä ja -alustoja sekä etsiä näistä mahdollista korvaajaa tällä hetkellä käytössä olevalle testiautomaatiokehykselle. Työn toimeksiantaja oli Sade Innovations Oy. Yrityksessä tahdottiin varmistua siitä, että käytössä on tarpeisiin ja vaatimuksiin nähden paras mahdollinen testiautomaatiokehys käytössä.

Ohjelmistotestaus on oleellinen osa ohjelmistokehitystä ja sen tavoite on ennaltaehkäistä sekä havaita virheitä ja puutteita. Ohjelmistotestausta voidaan jaotella usealla eri tavalla, esimerkiksi yksikkö-, integraatio-, järjestelmä- ja hyväksyntätestaamiseen. Näiden lisäksi testaaminen jaotellaan usein myös manuaali- ja automaatiotestaamiseen. Automaatiotestauksella voidaan tehostaa testaamista, mutta se ei poista tarvetta muulle testaamiselle.

Testiautomaatioon löytyy runsaasti erilaisia työkaluja, kehyksiä sekä alustoja. Niitä voidaan jaotella erilaisten kehysarkkitehtuurien mukaan. Testiautomaatiokehystä valittaessa on huomioitava tarpeisiin sopivan arkkitehtuurityypin lisäksi testaamisen tavoitteet sekä vaatimukset.

Testiautomaatiokehyksen valintaprosessi perustui toimeksiantajan määrittelemiin kriteereihin, joista erityisesti korostettiin luotettavuutta ja joustavuutta. Yrityksellä oli myös vähimmäisvaatimuksia, joiden perusteella kehyksiä ja alustoja valittiin vertailuun ja testaamiseen.

Lopputuloksena yrityksessä päädyttiin jatkamaan nykyisen kehyksen kanssa, koska muut vertailussa olleet eivät soveltuneet yhtä hyvin yrityksen tarpeisiin. Testiautomaatiota aiotaan kehittää esimerkiksi testiautomaatio-oppaan ja -koulutuksen avulla, sekä parannetaan ja päivitetään nykyisiä testiautomaatiosettejä.

Asiasanat

ohjelmistotestaus, testiautomaatio, testiautomaatiokehykset

Abstract

Author(s)	Type of Publication	Published
Marianna Syrjälä	Thesis, UAS	2022
	Number of Pages	
	37	

Title of Publication

Test automation frameworks and platforms

Choosing a high-quality test automation framework

Degree, Field of Study

Engineer (UAS), Information and Communications Technology

Organization of the client (if the thesis work is commissioned by another party)

Sade Innovations OY

Abstract

The aim of the work was to research and compare available test automation frameworks and platforms to find a possible replacement for the current test automation framework. The work was commissioned by Sade Innovations OY. The company wanted to be sure that they are using the best possible test automation framework for their needs.

Software testing is an essential part of software development. It aims to prevent and detect errors and defects. Software testing has several different types, like unit, integration, system, and acceptance testing. It is also often divided to manual and automated testing. Automated testing can improve efficiency, but it doesn't reduce the need for other types of testing.

There are plenty of different frameworks and platforms for test automation. They can be categorized by framework architecture types. When choosing a test automation framework, many factors should be considered, including the framework architecture type and the requirements and goals of testing.

The process of choosing a suitable framework was based on the criteria defined by the client. Reliability and flexibility were seen as the key criteria. Client also had minimum requirements for the framework. The aim was to choose a few frameworks for testing and comparison.

The end result was to continue using the current framework. Other frameworks didn't suite to the client's needs as well. The test automation practices will be improved with a test automation guide and training. The current test automation asset will also be updated and improved.

Keywords

software testing, test automation, test automation frameworks

Sisällys

1 Johdanto 1
2 Ohjelmistotestaus ja testausprosessi 2
2.1 Ohjelmistotestauksen määritelmä 2
2.2 Ohjelmistotestauksen osa-alueet 2
2.3 Ohjelmistotestauksen parhaat käytännöt 3
2.4 Testaaminen osana ohjelmistokehitysprosessia 4
3 Testiautomaatio 6
3.1 Testiautomaation määritelmä 6
3.2 Testiautomaatiokehyksien luokittelu 6
3.2.1 Yleistä testiautomaatiokehystyypeistä 6
3.2.2 Lineeariset testiautomaatiokehykset 6
3.2.3 Modulaariset testiautomaatiokehykset 6
3.2.4 Kirjastoarkkitehtuuripohjaiset testiautomaatiokehykset 7
3.2.5 Avainsanapohjaiset testiautomaatiokehykset 8
3.2.6 Datapohjaiset testiautomaatiokehykset 9
3.2.7 Käyttäytymispohjaiset testiautomaatiokehykset 9
3.3 Testiautomaatio osana ohjelmistokehitysprosessia 10
3.4 Testiautomaation parhaat käytännöt 10
3.5 Testiautomaatiokehyksen valinnan periaatteet 11
4 Testiautomaatiokehykset 12
4.1 Testiautomaation nykytilanne yrityksessä 12
4.2 Testiautomaatiokehyksen valintakriteerit ja tarpeet 13
4.3 Testiautomaatiokehyksen valintaprosessin määrittely 14
4.4 Testiautomaatiokehyksien tutkiminen ja vertailu 18
4.4.1 Katalon 19
4.4.2 LambdaTest 21
4.4.3 Oxygen 22
4.4.4 Robot Framework 23
4.5 Testiautomaatiokehyksen valinta 24
5 Yhteenveto ja pohdinta 27
Lähteet 28

Liitteet

Liite 1. Testien tulokset ja pisteytykset

Käsitteet

Appium	mobiiliautomaatiotyökalu
AUT	Application Under Test, testauksen kohteena oleva sovellus
Avoin lähdekoodi	ohjelmistokehitysmenetelmä, joka tarjoaa käyttäjille mahdollisuuden tutustua ohjelmiston lähdekoodiin ja muokata sitä
CD	Continuous Delivery on ohjelmistokehityksen menetelmä, jossa pyritään tekemään julkaisuja nopeassa tahdissa ja lyhyissä sykleissä
CI	Continuous Integration on työtapa, jossa ohjelmoijien tekemät koodimuutokset integroidaan yhteiseen jaettuun tilaan (esimerkiksi versionhallintaan) useasti
ohjelmointirajapinta	rajapinta komponenttien tai moduulien välillä ohjelmoitavassa järjestelmässä
Selenium	webautomaatiotyökalu

1 Johdanto

Testiautomaation käyttö osana ohjelmistokehitysprosessia lisääntyy jatkuvasti. Testiautomaatio on hyödyllinen osa ohjelmistokehitystä, kun se suunnitellaan ja toteutetaan huolellisesti. Sen avulla voidaan esimerkiksi varmentaa toimintoja, joiden kattava testaaminen manuaalisesti veisi runsaasti aikaa. Tämä säästää resursseja, joiden kohdentaminen olisi järkevämpää esimerkiksi tutkivaan testaamiseen.

Työn tilasi Sade Innovations OY. Yritys tarjoaa tuotekehityspalveluja, joihin sisältyy esimerkiksi mobiili- ja websovelluskehitys, pilvipalvelut sekä IoT-ratkaisut. Yritys työllistää noin 25 henkilöä ja vuoden 2021 liikevaihto oli hieman yli 1,9 miljoonaa. Yrityksellä on toimitilat Salossa ja Turussa. (Kauppalehti 2022.)

Työn tavoitteena on tutkia ja vertailla erilaisia testiautomaatiokehyksiä ja -työkaluja sekä etsiä mahdollista korvaajaa yrityksessä tällä hetkellä käytettävälle testiautomaatiokehykselle. Nykyisen kehyksen kanssa on ollut haasteita, jotka ovat vaikuttaneet testiautomaation laajempaan käyttöön. Yrityksessä halutaan olla varmoja, onko tämänhetkinen työkalu paras mahdollinen heidän tarpeisiinsa saatavilla olevista vaihtoehdoista.

Opinnäytetyö koostuu kolmesta osasta. Ensimmäisessä osassa käsitellään ohjelmistotestausta kokonaisuutena sekä sen merkitystä osana ohjelmistokehityksessä. Toisessa osassa käydään läpi testiautomaatiota ja sen merkitystä osana ohjelmistokehitysprosessia, eri testiautomaatiokehystyyppejä sekä testiautomaation parhaita käytäntöjä ja mitä tulee ottaa huomioon kehystä valittaessa. Viimeisessä osassa tutkitaan erilaisia tarjolla olevia testiautomaatiokehyksiä ja -työkaluja. Yrityksellä on omat määritellyt tarpeensa ja vaatimuksensa testiautomaatiolle, joiden pohjalta lähdetään etsimään mahdollisia vaihtoehtoja. Sopivimmat vaihtoehdot testataan käytännössä ja niiden sopivuutta yritykselle peilataan yrityksen tarpeita ja vaatimuksia vasten.

2 Ohjelmistotestaus ja testausprosessi

2.1 Ohjelmistotestauksen määritelmä

Ohjelmistotestaus on prosessi, jolla pyritään tunnistamaan ohjelmiston virheettömyys ottamalla huomioon kaikki sen eri ominaisuudet, kuten esimerkiksi luotettavuus, suorituskyky ja käytettävyys. Prosessissa arvioidaan myös ohjelmiston osien suoriutumista ja siitä yritetään löytää virheitä tai puutteita. Ohjelmistotestauksessa varmennetaan, että testauksen kohde toimii oikein ja odotetusti. (Atlassian 2020.)

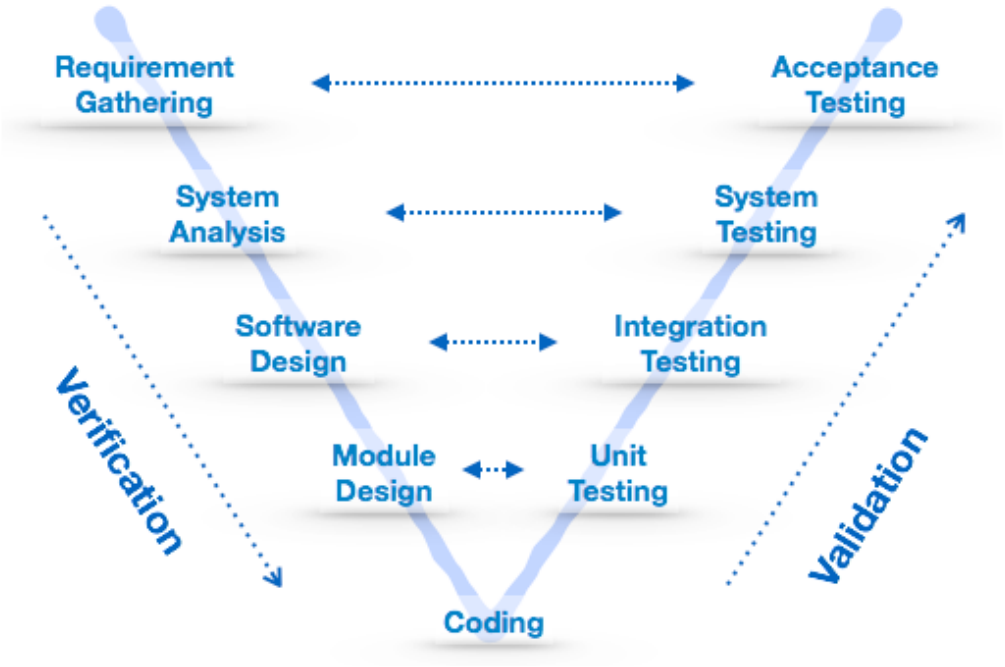
Yksi tapa määritellä ohjelmistotestaus on ohjelmiston verifiointi ja ohjelmiston validointi. Verifioinnin kohteena on esimerkiksi vaatimusmäärittely ja sovelluksen arkkitehtuuri. Verifioinnilla varmennetaan, että tuotetta tehdään oikein. Validoinnilla varmistetaan, tehdäänkö oikeaa tuotetta. Validoinnissa kohteena on tuote, kuten esimerkiksi valmis sovellus. (Chopra 2018, 30.)

2.2 Ohjelmistotestauksen osa-alueet

Ohjelmistotestauksessa on useita eri osa-alueita. Usein ohjelmistotestaus jaetaan eri tasoihin: yksikkö-, integraatio-, funktionaalinen-, järjestelmä- ja hyväksymistestaus. Näiden lisäksi voidaan myös puhua manuaali- ja automaatiotestauksesta. (Pittet.)

Yksikkötestauksella testataan ohjelmiston yksittäisiä osia ja integraatiotestauksella testataan eri ohjelmiston osien yhteistoimintaa. Funktionaalinen testaus eroaa integraatiotestauksesta sillä, että integraatiotestauksessa pyritään varmentamaan, että toiminnon suoritus onnistuu, kun taas funktionaalinen testaus pyrkii varmentamaan, että toiminnon tulos on oikea. Järjestelmätestauksessa varmistetaan koko järjestelmän testaus kokonaisuutena. Hyväksymistestauksella varmistetaan, että järjestelmä vastaa vaatimusmäärittelyä. (Pittet.)

Ohjelmistokehityksen ja testaamisprosessin kuvaamiseen käytetään usein niin kutsuttua V-mallia (kuva 1). Jokaiselle kehityksen vaiheelle on oma testaamisen tasonsa. Kun testausta tehdään eri tasoilla, virheiden ja puutteiden paikallistaminen helpottuu. V-mallissa verifiointi ja validointi kulkevat rinnakkain, ja mallia kutsutaankin myös verifiointi- ja validointimalliksi. (ARI Systems 2015.)



Kuva 1. Ohjelmistokehityksen V-malli (ARI Systems 2015)

Ohjelmistotestaus voidaan myös jakaa manuaali- ja automaatiotestaukseen. Manuaalitestauksella tarkoitetaan ihmisen käsin suorittamaa testausta, esimerkiksi sovelluksen käyttämistä. Manuaalitestaus on kallista, koska se sitoo työntekijän testiympäristön pystyttämistä ja testien suorittamista varten. Manuaalitestaus on altis inhimillisille virheille, eikä skaalaudu hyvin. (Pittet.)

Automaatiotestauksessa kone suorittaa testejä, jotka on kirjoitettu etukäteen. Testien monimutkaisuus voi vaihdella esimerkiksi funktion toiminnan tarkistuksesta monimutkaisen toimintojen toteuttamiseen käyttöliittymää vasten. Automaatiotestaus on luotettavampaa kuin manuaalitestaus, mutta testien laatu riippuu työkalun luotettavuudesta sekä siitä, miten hyvin testit on kirjoitettu. (Pittet.)

2.3 Ohjelmistotestauksen parhaat käytännöt

Testausta tulisi suorittaa koko ohjelmistokehitysprosessin ajan, eikä vain projektin loppuvaiheessa. On tärkeää tietää, mitä testataan – eli projektin tavoitteet on määriteltävä, jotta voidaan suunnitella testitapauksia. Lisäksi testaamisen riskit on huomioitava, kuten esimerkiksi testiympäristön turvallisuus. (NovateUS 2022.)

Testitapauksien suunnittelussa on huomioitava, että testaus kattaa eri osa-alueita ja erilaisia tilanteita. Testauksen tuloksia analysoitaessa tulee arvioida myös kehityksen kannalta.

Tarpeen mukaan testausprosessia on hyvä arvioida uudelleen ja tehdä tarvittavia parannuksia sekä muutoksia. (NovateUS 2022.)

Testausolosuhteiden ja -toimenpiteiden suunnittelu on tärkeää, jotta voidaan varmistua tulosten paikkansapitävyydestä, kun testit pystyttään toteuttamaan joka kerta samalla tavalla. Kuitenkin pitää myös huomioida erilaisten skenaarioiden ja versioiden testaaminen, jotta voidaan tunnistaa virhetilanteita erilaisissa ympäristöissä. Testaamisessa tulee myös pyrkiä simuloimaan oikean elämän olosuhteita. (NovateUS 2022.)

Testaamisen yksi tavoite on varmentaa, että ohjelmistosta ei löydy virheitä tai puutteita, mutta on tärkeää myös etsiä ohjelmistosta heikkouksia. Näin pystytään tunnistamaan ongelmia ennen kuin ne paisuvat suuremmiksi ja varmentamaan, että ohjelmisto vastaa asiakkaan määrittelemiä vaatimuksia. (NovateUS 2022.)

Testaamisessa on huomioitava resurssien rajallisuus, erityisesti ajan suhteen. Kaikkea ei pystytä testaamaan, koska erilaisia vaihtoehtoja ohjelmiston käytön suhteen on useita. Tästä syystä tulee välttää esimerkiksi turhaa toistoa testeissä. Käytettävien työkalujen valinta on tärkeää, jotta testausta voidaan tehdä tehokkaasti. (Chopra 2018, 18–19.)

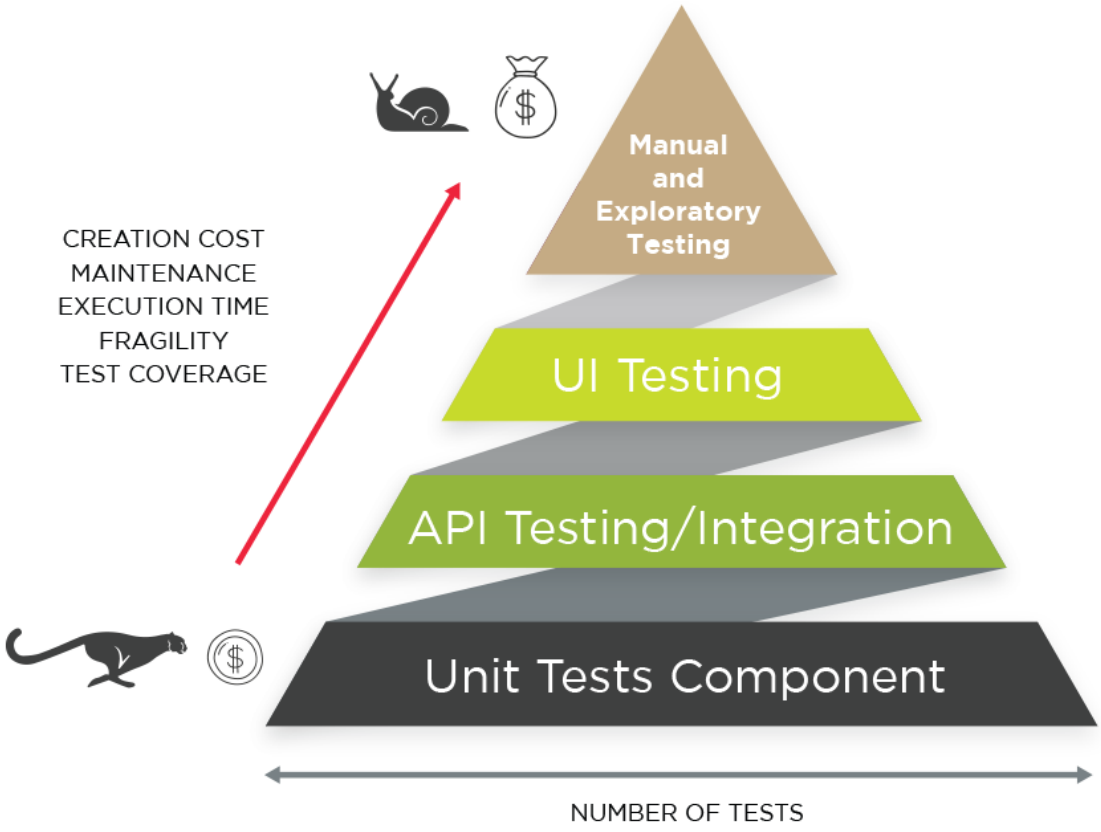
2.4 Testaaminen osana ohjelmistokehitysprosessia

Ohjelmistotestaus kattaa noin 40% työmäärästä sekä 25% ohjelmistokehityksen budjetista. Ohjelmistossa voi ilmetä puutteita ja vikoja huonolaatuisen vaatimusmäärittelyn, suunnittelun ja koodin takia. Puutteellisen testauksen ja ajan vähyyden takia vikoja voi jäädä havaitsematta, jolloin nämä pääsevät tuotantoon eli loppukäyttäjille. Puutteiden ja vikojen havaitseminen sekä korjaaminen myöhemmässä vaiheessa voi myös lisätä kustannuksia. (Chopra 2018, 6–8.)

Testaamisen tavoitteena on varmistaa tuotteen laatu sekä tyytyväiset asiakkaat ja käyttäjät. Yritykselle se voi tarkoittaa parempaa optimointia, esimerkiksi vähempinä ylläpitokustannuksina, ja luotettavuutta. Tuotteelle saavutetaan myös parempi käytettävyys ja toiminnallisuus. (Sahu.)

Lodewyksin (2020) mukaan automaatiopyramidin avulla voidaan määritellä, milloin kannattaa hyödyntää automaatiotestausta. Automaatiopyramidia käytetään usein kuvaamaan eri testaamisen tason määriä ja suhteita toisiinsa, ja usein siitä käytetään myös nimeä testauspyramidi. Mitä korkeammalle pyramidissa mennään, sitä vähemmän testejä tulisi tehdä. Ylemmillä tasoilla on enemmän kustannuksia verrattuna saataviin hyötyihin. Esimerkiksi yksikkötestausta tulisi hyödyntää mahdollisimman paljon kehityksen alkuvaiheessa, ja

käyttöliittymäpohjaista testausta tulisi hyödyntää tilanteisiin, jotka vaativat käyttöliittymän kanssa kommunikointia.



Kuva 2. Automaatiopyramidi (Lodewyks 2020)

Vaikka testaamisesta aiheutuu kustannuksia yrityksille, se voi aiheuttaa myös säästöjä, mikäli testausprosessit ja -tekniikat on hyvin suunniteltuja. Ohjelmiston puutteet tai viat voivat haitata yrityksen imagoa ja johtaa tyytymättömiin asiakkaisiin. Mitä aikaisemmin kehittäjät saavat palautetta testeistä, sitä aiemmin puutteita ja vikoja voidaan korjata. Tämä voi lisätä tehokkuutta, koska vikojen ja puutteiden korjaaminen on helpompaa, kun ne huomataan ajoissa. Kun testaukseen panostetaan, se parantaa ohjelmiston luotettavuutta ja tuotantoon pääsee korkealaatuisia sovelluksia vähillä virheillä. Tämä voi johtaa myös lisääntyneeseen myyntiin ja tuottoon. (IBM.)

3 Testiautomaatio

3.1 Testiautomaation määritelmä

Testiautomaatiolla tarkoitetaan sovellusten testaamista erilaisten automaatiotyökalujen avulla. Sovelluksien toimintaa ja reaktioita testataan erilaisia toimenpiteitä vastaan. Testiautomaatiolla pyritään automatisoimaan helposti ennustettavia, usein yksinkertaisia ja toistuvia toimenpiteitä ja täten varmistamaan, että testauksen kohteena oleva sovellus toimii oletetusti. Testiautomaatiokehyksellä tarkoitetaan alustaa, joka mahdollistaa ympäristön, missä suorittaa automatisoituja testejä sekä myös tuottaa testiraportteja. (Testim 2022.)

3.2 Testiautomaatiokehyksien luokittelu

3.2.1 Yleistä testiautomaatiokehystyypeistä

Testiautomaatiokehyksiä on useita erityyppisiä, ja niiden luokittelussa käytetään kehyksen arkkitehtuuria. Seuraavissa alikappaleissa esitellään erilaisia testiautomaatiokehystyyppejä. Alikappaleissa mainittujen kehystyyppien lisäksi on olemassa myös hybriditestiautomaatiokehyksiä, jotka yhdistelevät ominaisuuksia eri kehystyypeistä. (Testim 2022.)

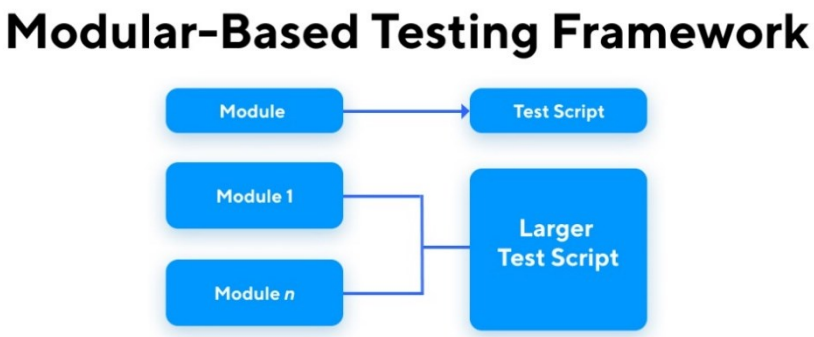
3.2.2 Lineeariset testiautomaatiokehykset

Lineaarisisissa testikehyksissä testaajien ei tarvitse kirjoittaa koodia ja testiaskeleet kirjoitetaan peräkkäisessä järjestyksessä. Testien jokainen askel tallennetaan ja toistetaan automaattisesti testiä suorittaessa, ja tästä syystä tämän tyyppisiä testikehyksiä kutsutaan myös ns. ”tallenna-ja-toista”-testikehyksiksi. (Smarbear 2022a.)

Lineaaristen testikehysten etuna on testien luomisen nopeus sekä käyttöönoton helppous. Testitapaukset vaativat kuitenkin ylläpitoa, kun testattavaan sovellukseen tehdään muutoksia. Testitapaukset eivät ole uudelleenkäytettäviä, koska esimerkiksi testidataa kovakoodataan. Jos testausta on tarve tehdä erilaisilla dataseteillä, tätä kovakoodattua testidataa joudutaan aina muuttamaan. (Smarbear 2022a.)

3.2.3 Modulaariset testiautomaatiokehykset

Modulaarisissa testikehyksissä testejä jaetaan eri osiin, esimerkiksi sovelluksen funktioiden ja osioiden testaamiseen. Jokainen näistä osista voidaan ajaa erillisinä osana, eli jokainen osa on itsenäinen ja riippumaton muista osista. Näistä osista yhdistellään suurempia kokonaisuuksia, jotka muodostavat testitapauksia. Kuvasta 3 voidaan tarkastella modulaarisen kehyksen rakennetta.

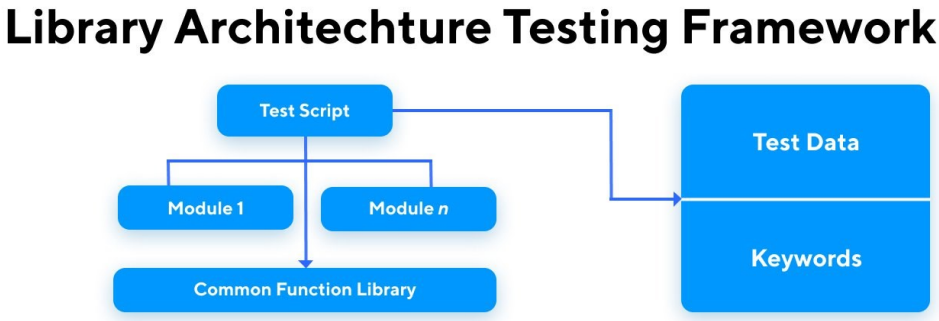


Kuva 3. Modulaarisen testiautomaatiokehyksen rakenne (Katalon 2022e)

Modulaaristen testiautomaatiokehysten yksi etu lineaarisiin kehyksiin verrattuna on se, että testauksen kohteen olevan sovelluksen muutokset eivät vaadi koko testitapauksen uudelleenkirjoittamista. Kun vain yksi osio tarvitsee päivittää, tämä helpottaa ylläpitoa. Testitapauksien luominen on myös tehokkaampaa, kun eri osia voidaan uudelleenkäyttää. Testidata on kuitenkin edelleen kovakoodattua, kuten lineaarisissa testiautomaatiokehyksissäkin. (Smartbear 2022b.)

3.2.4 Kirjastoarkkitehtuuripohjaiset testiautomaatiokehykset

Kirjastoarkkitehtuuripohjaisten testiautomaatiokehysten rakenne pohjautuu modulaarisiin testiautomaatiokehyksiin. Erona on se, että testauksen kohteena olevasta sovelluksesta tunnistetaan samantapaisia toimintoja. Näitä toimintoja jaotellaan funktioihin, jotka tallennetaan kirjastoon, mistä niitä voidaan kutsua tarvittaessa. Kuvassa 4 havainnollistetaan kehystyypin rakennetta. (Smartbear 2022a.)



Kuva 4. Kirjastoarkkitehtuuripohjaisen testiautomaatiokehyksen rakenne (Katalon 2022e)

Kirjastoarkkitehtuuripohjaisissa testiautomaatiokehyksissä on samoja etuja kuin modulaarisissakin, eli testien ylläpito helpottuu ja käyttö on kustannustehokkaampaa. Testattavan sovelluksen muutokset vaikuttavat vain testien eri osioihin, eivätkä ne vaadi koko testitapauksen uudelleenkirjoittamista. Yleisten funktioiden tai toimintojen kutsuminen kirjastosta parantaa uudelleenkäytettävyyttä. Tämän kehystyypin käyttö vaatii kuitenkin enemmän teknistä osaamista ja lisäksi testien luominen vie enemmän aikaa. (Smartbear 2022a.)

3.2.5 Avainsanapohjaiset testiautomaatiokehykset

Avainsanapohjaisissa testiautomaatiokehyksissä hyödynnetään nimensä mukaisesti avainsanoja, jotka sisältävät erilaisia peräkkäin suoritettavia ohjeita tai toimintoja. Esimerkiksi ”Kirjaudu sisään” koostuu useasta eri osasta toimintoja (taulukko 1). Testitapaukset koostetaan peräkkäin suoritettavista avainsanoista. Avainsanapohjaisissa testikehyksissä testidata, -skriptit ja avainsanat ovat kehyksestä irrallisia, eli näitä voidaan tallentaa omiin erillisiin tiedostoihinsa. Kun testejä suoritetaan, testidata luetaan ja ohjataan oikealle avainsanalle, joka suorittaa sitä koskevat toiminnot. (Smartbear 2022b.)

Step Number	Description	Keyword	Object
Step 1	Click user portal link on homepage	clicklink	Login Button
Step 2	Enter user name	Inputdata	Login Username
Step 3	Enter password	Inputdata	Login password
Step 4	Verify user log in information	verifylogin	
Step 5	Log in to the application	login	Submit Button

Taulukko 1. Avainsanataulukko (mukailtu Smartbear 2022b)

Testaajalta ei vaadita ohjelmointiosaamista, mutta testien kirjoittajalla on oltava hyvä ymmärrys automaatiosta. Testien kirjoittaminen ja avainsanojen luonti on monimutkaista ja vie aikaa. Etenkin testien ja avainsanojen määrän lisääntyessä ylläpito voi olla työlästä. Avainsanat ovat kuitenkin uudelleenkäytettäviä. (Smartbear 2022b.)

3.2.6 Datapohjaiset testiautomaatiokehykset

Datapohjaisten kehysten pääominaisuus on erottaa testidata testitapauksista, eli dataa säilytetään ulkoisesti. Testiskriptit yhdistetään ulkoisen datan lähteeseen (esimerkiksi csv-tiedostoon) ja ne hyödyntävät dataa testeissä tarvittaessa. Näin yhdellä testillä voidaan testata erilaisia skenaarioita erilaisilla dataseteillä. (Smartbear 2022b.)

Datapohjaisten testiautomaatiokehysten käyttö vaatii ohjelmointiosaamista, jotta ulkoista testidataa voidaan yhdistää testitapauksiin. Lisäksi testien kirjoittajan tulee kyetä tunnistamaan ja muotoilemaan testeissä käytettävää dataa. Testien ohjelmointi ja testidatan luominen ja ylläpitäminen vaatii osaamisen lisäksi aikaa. (Smartbear 2022b.)

3.2.7 Käyttäytymispohjaiset testiautomaatiokehykset

Käyttäytymispohjainen kehitys on yksi ohjelmistokehityksen tyyli, missä käytetään selkeää, käyttäjäkeskeistä kieltä kuvaamaan, miten sovelluksen tulisi toimia. Usein käyttäytymispohjaisissa kehyksissä käytetään Gherkin-ohjelmointikieltä, joka kuvaa ohjelmiston käyttäytymistä menemättä toteutuksen yksityiskohtiin. Gherkin-kieli käyttää Given-When-Then-rakennetta, minkä rakenteesta nähdään sisäänkirjautumista kuvaava esimerkki taulukossa 2. (Perfecto.)

Given	Käyttäjä avaa kirjautumissivun
When	Käyttäjä syöttää käyttäjätunnuksen
And	Käyttäjä syöttää salasanan
And	Käyttäjä painaa ”Kirjaudu sisään”-painiketta
Then	Käyttäjä kirjautuu sisään

Taulukko 2. Esimerkki Gherkin-kielen rakenteesta

Käyttäytymispohjaisten testiautomaatiokehysten etuna on ylläpidon ja käytön helppous, koska testitapauksissa käytetään selkeää kieltä, joten testien kirjoittaminen ei vaadi suurta teknistä osaamista. Testitapaukset ja testien tulokset ovat helppolukuisia ja täten myös esimerkiksi asiakkaat voivat helposti ymmärtää testien tuloksia. (Perfecto.)

3.3 Testiautomaatio osana ohjelmistokehitysprosessia

Testiautomaation yksi suurimpia hyötyjä on sen tarkkuus, koska testejä pystytään aina ajamaan samalla tavalla ja samassa ympäristössä. Täten vältetään inhimillisiä virheitä, mutta toisaalta tämä voi olla myös heikkous. Manuaalitestauksessa asioita tehdään eri tavoilla ja eri ajoituksilla, jolloin voidaan löytää erilaisia vikoja ja puutteita. Testiautomaation avulla pystytään kuitenkin testaamaan useammin ja nopeammin, jolloin pystytään havaitsemaan virheitä enemmän. Testiautomaatio ei poista tarvetta manuaalitestaukselle, mutta sitä voidaan hyödyntää yksinkertaisten, toistuvien testien kanssa, jolloin testaajille jää enemmän aikaa tehdä esimerkiksi tutkivaa testausta. (Da Silva Paternoster 2022.)

Testiautomaation yksi suurimpia haasteita on vaadittu alkupanostus. Testitapauksien suunnittelu ja toteuttaminen vaatii aikaa sekä osaamista. Testiautomaatiota pitää myös sovittaa ja jopa kirjoittaa uudelleen eri projekteihin ja ympäristöihin. Monet perustoiminnot, kuten esimerkiksi sisäänkirjautuminen, toimivat usein samalla tavalla, mutta käyttöliittymät ovat erilaisia. (Da Silva Paternoster 2022.)

Testiautomaatiossa voi ilmetä vääriä positiivisia tai negatiivisia. Testitapauksessa itsessään voi olla virhe, tai puutteellinen toteutus, joka ei huomioi kaikkia erikoistapauksia. Testitapauksien suunnittelu on haasteellista, koska niiden pitäisi pystyä sekä sopeutumaan testattavan sovelluksen muutoksiin ja samalla olla luotettavia. (Da Silva Paternoster 2022.)

3.4 Testiautomaation parhaat käytännöt

Testiautomaatiota suunnitellessa on tärkeää määritellä, mitä testataan ja mitä on kannattavaa testata. Usein on kannattavaa automatisoida yksinkertaisia, toistuvia toimintoja. Monimutkaiset testitapaukset voivat viedä aikaa sekä kirjoittaa, että toteuttaa. Lisäksi ne voivat olla myös herkkiä testeistä johtuville virheille. Kuitenkin manuaalisesti testattuna aikaa vievät testit on järkevää automatisoida, mikäli mahdollista. Jos testeissä vaaditaan erilaisia testausympäristöjä ja datasettejä, niiden automatisointi on kannattavaa. (Smartbear 2022a.)

Testitapauksien tulisi olla mahdollisimman sopeutuvia käyttöliittymässä tapahtuviin muutoksiin, joita etenkin kehityksen alkuvaiheessa voi tapahtua useinkin. Esimerkiksi käyttöliittymän elementtejä paikantaessa tulisi välttää sijaintiin viittaamista, vaan esimerkiksi käyttää erilaisia tunnisteita. Lisäksi nämä tunnisteet tulisi olla erillään testitapauksista, koska tämä helpottaa ylläpitoa. (Smartbear 2022a.)

Testitapauksien tulisi olla toisistaan riippumattomia, eli eri testit eivät voi olla riippuvaisia toisten testien onnistumisesta. Jokaisella testillä tulisi olla vain yksi testauksen kohde tai

tavoite. Mahdollisuuksien mukaan testien tulisi käsitellä sovelluksien virhetilanteita ja yrittää toipua niistä, sekä tarvittaessa tehdä uudelleenyrityksiä. (UiPath.)

Testitapauksien selkeä nimeäminen sekä jaottelu erilaisiin testitapauksiin on tärkeää ja se helpottaa myös testitulosten analysointia ja luettavuutta. Mikäli automaatiotyökalusta löytyy mahdollisuus ottaa kuvakaappauksia tai nauhoituksia, se voi helpottaa virhetilanteiden analysointia. (Bushnev 2019.)

3.5 Testiautomaatiokehyksen valinnan periaatteet

Testiautomaatiota suunnitellessa tulee määritellä, mitä halutaan testata. Resurssien ja mahdollisten riskien arviointi on myös tärkeää. Testiautomaation suunnittelu tulisi aloittaa mahdollisimman aikaisessa vaiheessa, koska automaatio vaatii resursseja, joiden hankkiminen voi viedä aikaa. Testiautomaatiokehyksen valinnassa tulee huomioida, että valittu kehystyyppi soveltuu testauksen tarpeisiin ja vaatimuksiin. (Katalon 2022b.)

Kehyksen valinnassa tulee huomioida myös testauksen kohde. Esimerkiksi mobiilisovelluksissa tulee ottaa huomioon sovelluksen tyyppi. On tärkeää tarkistaa, että käytössä olevat teknologiat sekä ympäristö sopivat yhteen testiautomaatiokehyksen kanssa. Valintaa tehtäessä tulee pohtia erilaisia vaatimuksia testiautomaatiolle, kuten nopeus, käytettävyys ja tarve rinnakkaiselle testaukselle. (SauceLabs 2022.)

Testaustiimin osaamisen taso vaikuttaa kehyksen valinnassa. Kehyksen käyttöönotto voi vaatia tarvetta koulutukselle. Korkea oppimiskynnys voi hidastaa käyttöönottoa. On tärkeä pohtia testaamisen vastuualueita, kuten kuka kirjoittaa testit ja kuka huolehtii ylläpidosta. (SauceLabs 2022.)

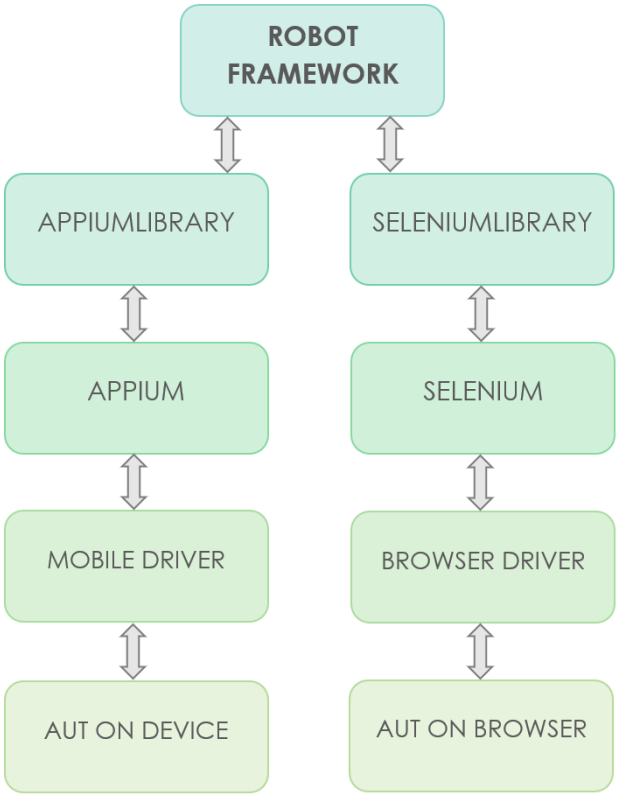
Resursseista tulee huomioida työvoiman lisäksi testiautomaatiokehyksen mahdolliset kustannukset. Kaupallisista ratkaisuista voi olla mahdollista hyödyntää myös ilmaisversioita, mutta tämä usein tarkoittaa rajallisempia käyttömahdollisuuksia. Vapaan lähdekoodin ratkaisut voivat houkuttaa ilmaisuudellaan, mutta niiden haasteena voi olla kehitystyön passiivisuus. Lisäksi tuen saaminen ongelmatilanteissa voi olla haastavaa. (SauceLabs 2022.)

4 Testiautomaatiokehykset

4.1 Testiautomaation nykytilanne yrityksessä

Yrityksellä on ennestään käytössä ollut Robot Framework-testiautomaatiokehys. Robot Framework on avoimen lähdekoodin automaatiokehys, jota voidaan käyttää muun muassa web- ja mobiilisovellusten testaukseen. Kehys tukee eri ohjelmointikieliä, kuten Pythonia ja Javaa, sekä sille on tehty useita eri kirjastoja. (Robot Framework.)

Yrityksessä käytetään Robot Framework-kehystä sekä mobiili- että web-testiautomaatiossa. Kuvasta 5 nähdään tarkemmin käytössä olevat työkalut ja niiden suhde toisiinsa. AppiumLibrary on mobiilitestauksessa käytettävä kirjasto, kun taas SeleniumLibrary on vastaava web-testauskirjasto. Kyseiset kirjastot hyödyntävät Selenium- ja Appium-työkaluja, jotka taas kommunikoivat ajurin kautta testauksen kohteena olevaan sovellukseen.



Kuva 5. Yrityksen testiautomaatiotyökalujen hierarkia

Kehys on pitkälti soveltunut yrityksen tarpeisiin hyvin, mutta sen kanssa on myös ollut useita haasteita. Testiautomaation ylläpito on yksi isoimmista haasteista, mitä ei tietysti kehyksen

vaihtaminen ratkaise. Yritys haluaa kuitenkin tutkia mahdollisia tarjolla olevia vaihtoehtoja varmistuakseen, että heillä on paras mahdollinen työkalu käytössä.

Yrityksessä nähdään testaus ja testiautomaatio tärkeänä osana ohjelmistokehitystä. Vaikka nykyisen kehyksen käytössä on ollut haasteita, on siitä saatu hyötyjäkin ja löydetty vikoja, joiden löytäminen manuaalitestauksessa olisi ollut hankalaa. Nykyisen kehyksen kanssa on ollut ajoittain ongelmia luotettavuuden ja stabiiliuden kanssa, esimerkiksi väärien negatiivisten testien kanssa. Näitä on kuitenkin osittain pystytty korjaamaan kirjoittamalla testejä paremmin.

Yksi nykyisen kehyksen haasteista on ollut mobiilitestauksessa käytettävä AppiumLibrary-kirjasto. Kirjastosta on käytössä löytynyt esimerkiksi avainsanoja, jotka eivät toimi ollenkaan. Lisäksi avainsanoja puuttuu tietynlaisista perustilanteista, joka on johtanut siihen, että on jouduttu kirjoittamaan enemmän omia avainsanoja, jotta tietynlaisia tilanteita voitaisiin testata. Paikoittain dokumentaatio on ollut koettu osittain sekavaksi ja harhaanjohtavaksi.

4.2 Testiautomaatiokehyksen valintakriteerit ja tarpeet

Yrityksen ensisijainen kriteeri testiautomaatiokehyksessä on luotettavuus, millä tässä yhteydessä tarkoitetaan sitä, että tiimin on voitava luottaa testien tuloksiin. Luotettavuudella tarkoitetaan myös sitä, että kehyksen toimintavarmuus ja -kyky ovat tärkeitä. Automaatiotestien tulisi oltava mahdollisimman virhesietoisia ja hyvin toistettavia. Testiautomaatiokehyksen lisäksi tähän luonnollisesti vaikuttavat paljon myös testien suunnittelu ja toteutus. Toisena tärkeänä kriteerinä esille on noussut kehyksen joustavuus. Tästä syystä ideaalein kehystyyppi olisi hybridikehys, mutta yrityksen tarpeisiin myös avainsana- tai käyttäytymispohjainen kehys on riittävä.

Kehyksen valinnassa on tiettyjä rajoitteita. Sen tulisi tukea sekä web- että mobiilitestiautomaatiota. Mobiilitestiautomaation osalta on otettava huomioon, että kehyksen on sovelluttava cross-platform-sovelluksille. Mobiilitestauksessa käytetään myös oikeita laitteita, joten kehyksessä tulee olla kyvykkyys tähän. Lisäksi integraatio CI/CD-ympäristöön on tarpeellista.

Merkittävänä etuna nähdään kehyksen monipuolisuus, eli mahdollisuus laajentaa sen toimintaa lisäkirjastoilla tai liitännäisillä. Lisäksi erilaisia toimintoja, kuten avainsanoja, tulisi olla monipuolisesti käytettävissä. Lisäarvoa tuo kehyksen kustomointi, esimerkiksi mahdollisuus luoda omia avainsanoja.

Kehyksen helppokäyttöisyys ja erityisesti helppo käyttöönotto voivat tehostaa työskentelyä ja mahdollistaa testiautomaation hyödyntämisen matalalla kynnyksellä. Testaajalla ei

välttämättä ole aina laajaa ohjelmointitaustaa, joten kehyksen ja kehysympäristön monimutkaisuus voivat hidastaa ja jopa rajoittaa käyttöönottoa. Testaajien lisäksi on huomioitava myös ohjelmistokehittäjät, koska he ovat mukana testiautomaation päivittämisessä. Jos esimerkiksi kehyksen käytettävyys koetaan huonoksi ja hankalaksi, se voi hidastaa ja rajoittaa kehitystä. Tästäkin syystä hyvä dokumentaatio on tärkeää.

Valittava kehys voi olla myös kaupallinen, mikä voi tuoda omat etunsa, esimerkiksi varmemman turvan kehyksen ylläpidosta, tuesta ja kehitystuesta. Haasteena tässä on kuitenkin se, että yritys tekee alihankintaa, jolloin testiautomaatiokehyksen kustannukset heijastuvat myös asiakkaalle, mikä voi vaikuttaa asiakkaiden halukkuuteen hyödyntää testiautomaatiota. Avoimen lähdekoodin kehyksissä tulee tiedostaa riski, että kehitystyö voi pahimmillaan jopa lakata. Lisäksi tuen saaminen ongelmatilanteissa voi olla haastavaa.

4.3 Testiautomaatiokehyksen valintaprosessin määrittely

Testiautomaatiokehyksen valintaprosessi aloitetaan tutkimalla ja arvioimalla erilaisia saatavilla olevia kehyksiä. Tavoitteena on valita testattavaksi korkeintaan viisi erilaista testikehystä. Näihin viiteen lasketaan mukana jo käytössä oleva Robot Framework. Aluksi kehyksiä tutkiessa rajataan pois kehykset, jotka eivät täytä aiemmin mainittuja vähimmäisvaatimuksia, kuten esimerkiksi kehyksen tulisi tukea sekä mobiili- että web-automaatiotestausta. Kehyksiä suljettiin pois myös kehysarkkitehtuurin mukaan, esimerkiksi pelkkää lineaarista testiautomaatiota tukevat suljettiin vertailun ulkopuolelle. Lisäksi huonosti dokumentoituja ja päivitettyjä suljettiin vertailun ulkopuolelle.

Eri kehysten vertailua varten on määritetty tietyt kriteeristöt, minkä perusteella kehyksen soveltuvuutta arvioidaan. Kriteereille on määritelty painotusarvo, eli kuinka merkityksellinen kyseinen kriteeri on kehyksen valinnassa. Taulukossa 3 on eritelty tarkemmin eri kriteerien painotusarvot. Yrityksessä nähtiin erityisen tärkeänä kehyksen luotettavuus. Jos testiautomaatiotuloksiin ei voida luottaa, tulosten tulkinta ja virheellisten tulosten arviointi vie työntekijöiltä turhaa aikaa. Luotettavuuden jälkeen joustavuus ja monipuolisuus nähtiin tärkeinä kriteereinä.

VALINTAKRITEERI	PAINOTUS
LUOTETTAVUUS	25 %
JOUSTAVUUS	20 %
MONIPUOLISUUS	20 %
KÄYTTÖÖNOTTO JA OPPIMISKYNNYS	15 %
DOKUMENTAATIO JA TUKI	10 %
HINNOITTELU	10 %

Taulukko 3. Testiautomaatiokehyksen valintakriteerien painotus

Jokaisesta valintakriteeristä saa korkeintaan 10 pistettä, eli korkein kokonaispistemäärä on 60 pistettä. Taulukossa 4 määritellään yksityiskohtaisemmin eri kriteerien eri osa-alueet ja näistä saatavat pisteet. Kriteerejä jaettiin eri osa-alueisiin, jotta eri kehyksiä voitaisiin vertailla ja arvioida mahdollisimman yhdenvertaisesti.

VALINTAKRITEERI	ARVIOINNIN OSA-ALUEET		MAX. PISTEET
LUOTETTAVUUS	Web-sovellustesti esimerkeistä		2,5
	Mobiilisovellustesti esimerkeistä		2,5
	Web-sovellustesti, itse kirjoitettu		2,5
	Mobiilisovellustesti, itse kirjoitettu		2,5
JOUSTAVUUS	Hybridikehys		2
	Mahdollisuus luoda omia avainsanoja tai toimintoja		8
MONIPUOLISUUS	Avainsanojen tai toimintojen määrä		5
	Kirjastot ja liitännäiset		5
KÄYTTÖÖNOTTO JA OPPIMISKYNNYS	Käyttöönoton helppous		5
	Oppimiskynnys		5
DOKUMENTAATIO JA TUKI	Dokumentaatio ja oppaat		4
	Tuki		2
	Laajasti käytetty		2
	Hyvin päivitetty		2
HINNOITTELU	Ilmainen	10	10
	Alle 1000$/vuosi	5	
	Alle 2000$/vuosi	2,5	
	Yli 2000$/vuosi	0	

Taulukko 4. Valintakriteerien osa-alueet ja pisteytys

Luotettavuuden arviointi kattavasti on haastavaa. Aluksi jokaiselle kehykselle suoritetaan neljä testiä – kaksi testiä kehyksen tai alustan esimerkeistä ja kaksi itsekirjoitettua testiä. Esimerkkitesteistä otetaan yksi mobiili- ja yksi websovellustesti, ja myös itsekirjoitetuissa testeissä testataan sekä mobiili- että web-sovellusta. Jokaisesta onnistuneesta testistä saa kaksi ja puoli pistettä. Tässä vaiheessa tiedostetaan, että nämä neljä testiä eivät kuitenkaan anna kovin syvällistä kuvaa kehyksen luotettavuudesta pitkällä aikavälillä. Näillä testeillä tavoitellaan alkuarvioita kehyksen perustoimintojen toimivuudesta.

Itsekirjoitetuissa testeissä testataan erilaisia toimintoja, kuten esimerkiksi elementtien painamista ja elementtien sisällön tarkistusta. Näissä testeissä hyödynnetään harjoitustyönä tehtyä web-sovellusta ja mobiilille puhelimen omaa muistiinpanosovellusta. Testauksen kohteena oleva web-sovellus on satunnaishaastegeneraattori eräälle pelille. Se soveltuu hyvin tähän tarkoitukseen, koska sille voidaan kirjoittaa monimutkaisia, pitkäkestoisempia testejä.

Näillä testeillä tehdään alustava arvio kehyksen toiminnasta, eli varmennetaan, että testejä kyetään kirjoittamaan sekä suorittamaan onnistuneesti. Mikäli testit läpäistään, kehyksen käytön testaamista jatketaan syvällisemmin ja monimutkaisemmilla testeillä sekä sovelluksilla, joilla saadaan parempaa kuvaa luotettavuudesta. Taulukosta 5 voidaan nähdä tarkempi kuvaus itsekirjoitetusta testeistä sekä niiden tavoitteista, eli mitä osa-alueita testiautomaatiotyökalusta pyritään näillä testeillä arvioimaan.

WEB	
TESTITAPAUS	**TAVOITTEET**
Ei pelaajia • **Verifioi oikean palautteen, kun pelaajia ei ole lisätty**	• **Erilaisten elementtien ja niiden paikantimien käyttö ja tunnistaminen** • **Perustoimintojen testaaminen**
Yksi pelaaja • **Verifioi oikean palautteen, kun pelaajia yksi**	• **Perustoimintojen laajempi testaaminen** • **Monimutkaisemman testitapauksen testitapaaminen**
Monta pelaajaa • **Verifioi oikean palautteen, kun pelaajia useita**	• **Pitkäkestoisen testin testaaminen**
MOBIILI	
TESTITAPAUS	**TAVOITTEET**
Uuden muistiinpanon luominen	• **Erilaisten elementtien ja niiden paikantimien käyttö ja tunnistaminen** • **Perustoimintojen testaaminen**
Muistiinpanojen haku	
Muistiinpanon poistaminen	

Taulukko 5. Tarkempi kuvaus itsekirjoitetuista testeistä ja niiden tavoitteista

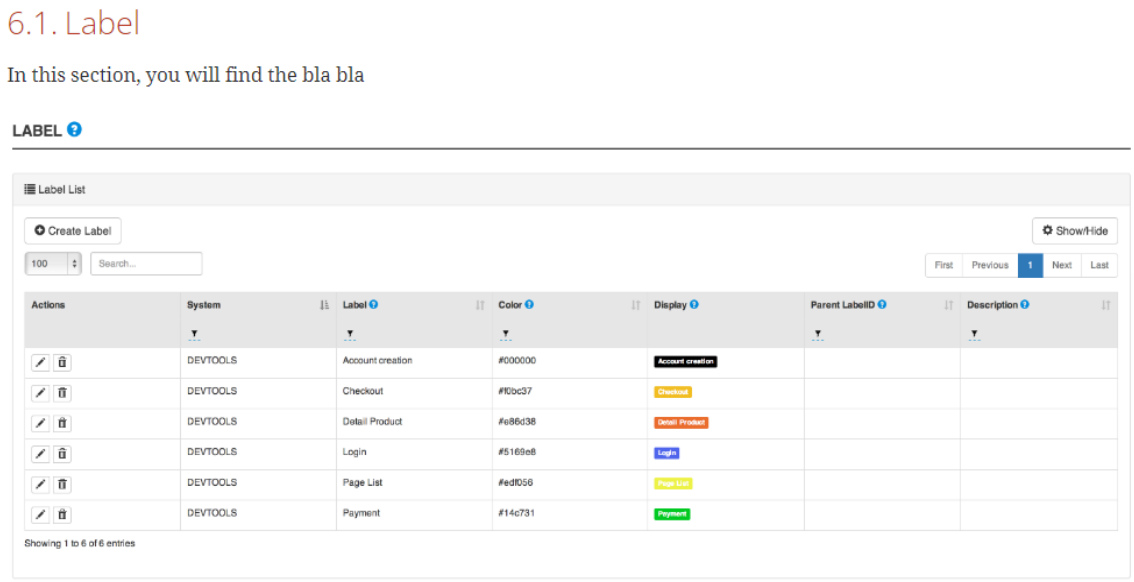
4.4 Testiautomaatiokehyksien tutkiminen ja vertailu

Erilaisia testiautomaatiokehyksiä ja -työkaluja löydettiin tutkittaessa useita kymmeniä. Kun kehyksiä ja työkaluja rajattiin pois aiemmin mainittujen kriteerien perusteella, jäljelle jäi kuusi mahdollista vaihtoehtoa: LambdaTest, Katalon, Oxygen, Perfecto, Bellatrix ja Cerberus. Koska vaihtoehtoja on enemmän kuin työssä alun perin tavoitellut viisi, tutkittiin vielä mahdollisia tekijöitä, millä mahdollisesti rajata listaa lyhyemmäksi. Tämän saavuttamiseksi tehtiin vielä tarkempaa tutkimusta kehysten ominaisuuksista ja dokumentaatiosta.

Perfectoa tutkittaessa huomattiin, että se tukee mobiilitestauksessa vain Java-ohjelmointikieltä. Ohjelmointi itsessään ei ole ongelma, mutta oppimiskynnys tässä tapauksessa voi osoittautua liian korkeaksi. Lisäksi Perfecto on maksullinen, joten jos käyttöä hidastaa ja pahimmillaan rajoittaa korkea oppimiskynnys sekä vaikea käytettävyys, ei välttämättä kustannuksille saada kunnollista vastinetta. Tästä syystä Perfecto päätettiin jättää vertailun ulkopuolelle.

Myös Bellatrix suljettiin pois rajallisten ohjelmointikielien tukien vuoksi. Vaikka Bellatrix on avoimen lähdekoodin kehys, eikä täten aiheuta suoria kustannuksia, kehys tukee toistaiseksi vain Java- ja C#-ohjelmointikieliä. Koska kyseiset kielet eivät ole yrityksessä laajassa käytössä, voidaan kehyksen käyttö kokea hankalaksi. Lisäksi testien kirjoittaminen voi olla hitaampaa, kun kyseessä on mahdollisesti vieraammat ja monimutkaisemmat kielet.

Viimeisimpänä suljettiin pois Cerberus paikoittain huonon dokumentaation takia. Dokumentaatiosta löytyi kohtia, joista saattoi puuttua sisältö kokonaan tai löytyä täytetekstiä, mistä voidaan nähdä esimerkki kuvassa 6. Tämä herätti epäluottamusta, mutta myös erinäisten ominaisuuksien puutteellinen dokumentaatio voi vaikuttaa käytettävyyteen ja tämä nähtiin riskinä.



Kuva 6. Kuvakaappaus Cerberus-testiautomaatiokehyksen dokumentaatiosta (Cerberus)

Lopulta testaukseen päädyttiin siis ottamaan kolme vaihtoehtoa: LambdaTest, Katalon ja Oxygen. Aiemmin mainittujen kriteerien ja vaatimusten lisäksi analysoidaan myös muitakin tekijöitä, kuten esimerkiksi testiraporttien luettavuutta sekä kehyksen käyttökokemusta. Näitä ominaisuuksia arvioidaan sanallisesti, koska kaikille ominaisuuksille ei voida helposti määritellä numeerista arviota. Seuraavissa kappaleissa esitellään valittuja vaihtoehtoja ja niiden eri ominaisuuksia tarkemmin.

4.4.1 Katalon

Katalon on alusta, joka tarjoaa useita eri työkaluja testiautomaatioon. Katalon TestOps-ohjelmistolla voidaan suunnitella testejä sekä ajoittaa niiden suorituksia. Testien luonti tapahtuu Katalon Studio-ohjelmistolla, millä voidaan myös tutkia testien tuloksia ja testiraportteja. Katalon RunTime Engine-laajennus mahdollistaa testien suorittamisen osana CI/CD-ympäristöä. Lisäksi Katalon tarjoaa myös Katalon TestCloud-palvelua, joka on pilvialusta, missä testejä voidaan ajaa esimerkiksi eri selainversioilla. (Katalon 2022a.)

Katalon on kaupallinen työkalu, ja sen hinnoittelu vaihtelee haluttujen palvelujen mukaan. Palvelun Premium-version kuukausimaksu on 25 dollaria kuukaudessa vuosittaisella laskutuksella ja tämä pitää sisällään muun muassa korkeintaan 10 aktiivista projektia sekä 24/5 teknisen tuen. Eri ohjelmistojen ja lisäpalveluiden hinnat vaihtelevat, kuten alla olevasta kuvasta 7 voidaan tarkastella. Vaikka kuukausimaksu ei ole suuri, kokonaisuutena tästä

voidaan nähdä, että kustannukset voivat nousta melko korkeiksi kalliiden ohjelmistojen takia. (Katalon 2022d.)

Kuva 7. Katalon-alustan ohjelmistojen hinnat (Katalon 2022d)

Katalon tukee sekä lineaarista- että avainsanapohjaista testiautomaatiota. Lisäksi Katalon integroituu Cucumber-kehyksen kanssa, mikä mahdollistaa käyttäytymispohjaisen testiautomaation. Kielinä voidaan käyttää sekä Java-ohjelmointikieltä että yksinkertaisempaa Groovy-skriptauskieltä. (Katalon 2022a.)

Katalon-alustaa käyttävät monet yritykset, kuten esimerkiksi Nvidia ja IBM. Dokumentaatiosta ja Katalon Studio-ohjelmistosta voidaan huomata, että erilaisia avainsanoja on runsaasti ja omien avainsanojen luominen on mahdollista. Erilaisista liitännäisistä ja kirjastoista löytyy heikosti tietoa, mutta alusta integroituu useisiin eri palveluihin, kuten esimerkiksi Jiraan ja Jenkinsiin. (Katalon 2022c.)

Dokumentaatio vaikuttaa kattavalta ja runsaalta, mikä myös johtaa siihen, että tietoa on vaikea löytää. Osittain tästä syystä käyttöönotto on koettu todella hitaaksi ja haastavaksi. Tähän vaikuttaa myös se, että Katalon Studio-ohjelmiston käyttöliittymä ei ole kovin intuitiivinen. Monet perustoiminnot ovat joko hankalia löytää tai käyttää, kuten testitapauksien luonti tai ajo.

Alustan esimerkkitestit ajettiin onnistuneesti, mutta itse luotujen testien kanssa kohdattiin huomattavasti ongelmia. Esimerkiksi elementtien tunnisteiden käyttö on sekavaa. Web-

testissä ei onnistuttu käyttämään tunnisteita oikein, eli testiä ei saatu kirjoitettua eikä suoritettua onnistuneesti. Dokumentaatiosta löytynyt tieto ei ratkaissut ongelmia.

Mobiilitesteissä kokeiltiin ohjelmiston tarjoamaa työkalua testiaskelien luomiseen nauhoitus- ja tallennustyökalulla. Työkalussa avattiin puhelimesta testauksen kohteena oleva sovellus ja testiaskeleet suoritettiin auki olevalla sovelluksella, mistä ohjelmisto tallensi ne testitapaukseksi. Tässäkään ei kuitenkaan onnistuttu, koska testiä ajaessa ohjelmisto ei löytänyt testauksen kohteena olevaa sovellusta, vaikka sovellus oli juuri avattu onnistuneesti nauhoitus- ja tallennustyökalussa ja kaikki testin osiot oli hoidettu tätä kautta.

Katalon-alustasta jäänyt mielikuva oli melko negatiivinen. Raskas ja vaikeakäyttöinen ohjelmisto sekä sekava dokumentaatio vaikuttivat tähän eniten. Käytännössä alustan käytön opettelu vaatisi yksinkertaisesti liikaa aikaa. Alusta on myös kallis käyttää, joten hinta-hyötysuhde voi jäädä huonoksi, kun opettelu syö resursseja.

Katalon Studio-ohjelmistossa on paljon hyödyllisiä ominaisuuksia, kuten suora integraatio versionhallintaan, mutta testien kirjoittaminen on hankalaa ja monimutkaista. Kuten aiemmin todettiin, ei itsekirjoitettuja testejä saatu suoritettua onnistuneesti, eikä ongelmiin löydetty ratkaisua. Liitteestä 1 voidaan nähdä tarkemmin alustalle eri osa-alueista annetut pisteet.

4.4.2 LambdaTest

LambdaTest on alusta, jonka alla voidaan käyttää useita eri testiautomaatiokehyksiä sekä -työkaluja (LambdaTest 2022a). Alusta tarjoaa kattavan pilvialustan eri selain- ja puhelinversioiden testaamiseen ja web-pohjaisen käyttöliittymän kautta voidaan tarkastella testien tuloksia yksityiskohtaisesti (LambdaTest 2022c). Tätä vaihtoehtoa tutkittiin, koska se voisi nopeuttaa ja helpottaa automaatiotestausympäristön pystytystä ja siten tehostaa sekä monipuolistaa testaamista.

LambdaTest-alustan hinnoittelu on skaalautuva ja perustuu rinnakkaisten testiajojen määrään. Molemmille web- ja mobiilitestiajoille on oma kuukausihinnoittelunsa. Esimerkiksi yksi rinnakkaisajo web-testiautomaatiossa maksaa vuodessa yhteenä 948 dollaria, kun taas mobiilitestiautomaatiossa 1500 dollaria. Mikäli tarvitaan enemmän yhtä aikaa ajettavia testejä, myös kustannukset kasvavat. (LambdaTest 2022b.)

LambdaTest-alustaa käyttää yli 500 yritystä, kuten esimerkiksi Microsoft ja Cisco (LambdaTest 2022a). Alustalta löytyy melko kattava ja monipuolinen dokumentaatio, mutta osittain se osottautui myös sekavaksi ja haluttua tietoa oli ajottain hankala löytää. Apuna tulee

käyttää alustassa ajettavan kehyksen dokumentaatiota, ja tästä syystä alkuun pääseminen voi olla hidasta. Alusta tarjoaa kuitenkin 24/7-tukea (LambdaTest 2022a).

Testauksessa päädyttiin hyödyntämään Selenium- ja Appium-työkaluja. Suuri osa testiautomaatiokehyksistä käyttää näitä työkaluja. Käyttöönotto oli nopeaa ja helppoa, koska tähän löytyi dokumentaatiosta hyvät ohjeet. Testien toteutuksessa kuitenkin huomattiin, että kokeiluversiossa ei voida ajaa mobiilitestiautomaatiota ollenkaan. Tästä syystä näiltä osin luotettavuutta ei voida arvioida, ja pisteytys jää näiltä osin nollaan.

Web-testit suoritettiin onnistuneesti, mutta Selenium-työkalun käyttö koettiin kuitenkin hankalaksi ja monimutkaiseksi. Perustoimintoja jouduttiin hakemaan paljon ja testejä kirjoittaessa törmättiin useisiin ongelmiin, joiden ratkaisu oli hankalaa. Testien kirjoittaminen oli täten hidasta ja kangertelevaa. Testejä kirjoitettiin JavaScript-ohjelmointikielellä, mikä tarjosi paljon joustavuutta ja vapautta testien kirjoittamiseen, mutta lisäsi myös monimutkaisuutta.

Alustan kokonaisarvion kannalta pitää huomioida, että se eroaa muista vaihtoehdoista rakenteeltaan. LambdaTest tarjoaa kuitenkin helppokäyttöisen web-käyttöliittymän, mistä tarkastella testien tuloksia ja lisäksi tehostaa sekä yksinkertaistaa testiympäristön pystytystä. Alustan alla käytettävät kehykset ovat kuitenkin pääasiassa ohjelmointipohjaisia, mikä monimutkaistaa ja hidastaa testien kirjoittamista. Alustan pisteytyksiä voidaan tarkastella tarkemmin liitteestä 1.

4.4.3 Oxygen

Oxygen on avoimen lähdekoodin testiautomaatiokehys, jota voidaan käyttää mobiili- ja websovellustestauksen lisäksi myös esimerkiksi työpöytäsovellus, tietokanta- ja REST-ohjelmointirajapintatestaamiseen. Kehys sisältää myös oman kehitysympäristön sekä komentorivikäyttöliittymän. Kehys tukee JavaScript-ohjelmointikieltä sekä data- ja käyttäytymispohjaista testiautomaatiota. Kehys sisältää useita eri moduuleja, kuten esimerkiksi sähköposti- ja SMS-moduulit, minkä lisäksi kehyksen toimintaa voidaan laajentaa itse luotujen moduulien avulla. (Oxygen 2021.)

Kehyksestä ei voida varmasti sanoa, kuinka laajasti käytetty se on, koska tällaista tietoa ei ole saatavilla. Kehyksen ensimmäinen versio on julkaistu vuonna 2018, joten kehys on vielä melko tuore. Kehys vaikuttaa kuitenkin hyvin päivitetyltä ja viimeisin uusi julkaisu on tehty syyskuussa 2022. (Oxygen 2022.)

Kehyksen käyttöönotto oli nopeaa ja helppoa, ja kehitysympäristö on yksinkertainen sekä selkeä käyttää. Kehyksen omat esimerkkitestit suoritettiin onnistuneesti, mutta itsetehtyjen testien kanssa oli ongelmia. Itse kirjoitetussa mobiilitestissä kehyksen kanssa ei onnistuttu

paikantamaan elementtejä tiettyjen paikannusmuuttujien kanssa, joten testiä ei saatu suoritettua onnistuneesti. Ongelmaan etsittiin ratkaisua dokumentaatiosta ja yritettiin eri vaihtoehtoja, mutta kyseistä toimintoa ei tästä huolimatta saatu toimimaan.

Itse kirjoitettu web-testi taas jäätyi kesken testin suorituksen, mistä syystä sitä ei saatu suoritettua loppuun. Testejä ajaessa huomattiin, että testiraportteja ei saatu generoitua onnistuneesti. Testeistä tallennettiin lokitiedostot, mutta tuloksien analysointi ilman selkeitä testiraportteja on hankalaa.

Kehys ja sen kehitysympäristö ovat molemmat todella helppokäyttöisiä ja testien kirjoittaminen on nopeaa. Kehys vaikutti kuitenkin vielä kovinkin epävakaalta ja kaikki toiminnot, kuten testiraporttien luonti, eivät toimi odotetusti ja dokumentaatiosta ei löydetty tähän ratkaisua. Dokumentaatio oli selkeää ja helppolukuista, mutta ei kattavaa kaikilta osin. Tarkempia pisteytyksiä eri osa-alueista voidaan tarkastella liitteestä 1.

4.4.4 Robot Framework

Kuten työssä on jo aiemmin mainittu, Robot Framework on avoimen lähdekoodin automaatiokehys. Kehykselle on tehty kymmeniä eri kirjastoja. Mobiili- ja web-testauksen lisäksi sitä voidaan käyttää muun muassa Windows-työpöytäsovelluksien, tietokantojen sekä erilaisten ohjelmointirajapintojen testaukseen. Kehyksellä on myös oma testidataeditori, RIDE. (Robot Framework.)

Kehystä itsessään ylläpidetään ja kehitetään aktiivisesti. Sitä sponsoroivat useat eri yritykset, kuten esimerkiski Cisco ja Nokia. Koska kyseessä on avoimen lähdekoodin kehys, eri kirjastojen ylläpito riippuu niiden omista kehittäjistä. Kehykselle on myös mahdollista kirjoittaa omia kirjastoja. (Robot Framework.)

Kun Robot Framework-kehys otettiin yrityksessä käyttöön, käyttöönotto itsessään ei ollut hankalaa – se toimii kuitenkin hyvin samalla tavalla kuin esimerkiksi aiemmin mainittu Oxygen-kehys, eli kehys hyödyntää Appium- ja Selenium-työkaluja. Kehys tukee avainsana-, data- ja käyttäytymispohjaista testiautomaatiota. Syntaksi poikkeaa monesta ohjelmointikielestä, ja tämän oppiminen vaati hieman aikaa. Syntaksi on erityisen tarkka muotoilusta, pääasiassa tabulaattorien käytöstä (eli esimerkiksi avainsanan ja sen parametrien välissä on tarpeeksi välejä).

Syntaksista voidaan nähdä esimerkki kuvasta 8. Testitiedostot voivat itse testitapauksien lisäksi sisältää erilaisia osioita, kuten avainsanoja, muuttujia ja asetuksia. Testidataa, kuten avainsanoja ja muuttujia, voidaan asetuksissa tuoda myös ulkopuolisista tiedostoista. Lisäksi omien avainsanojen luonti on mahdollista.

```
*** Settings ***
Documentation     A test suite for valid login.
...
...               Keywords are imported from the resource file
Resource          keywords.resource
Default Tags      positive

*** Test Cases ***
Run Test
Login User with Password
    Connect to Server
    Login User            ironman    1234567890
    Verify Valid Login    Tony Stark
    [Teardown]    Close Server Connection

Run Test
Denied Login with Wrong Password
    [Tags]    negative
    Connect to Server
    Run Keyword And Expect Error    *Invalid Password    Login User    ironman    123
    Verify Unauthorised Access
    [Teardown]    Close Server Connection
```

Kuva 8. Esimerkki Robot Framework-kehyksen syntaksista (Robot Framework)

Testien kirjoittaminen ja ajaminen oli ongelmatonta, mutta tähänkin vaikuttaa isosti se, että kehyksen toiminnallisuus ja käyttö on ennestään tuttua. Dokumentaatiota on melko runsaasti ja se on pääosin melko selkeää, mutta aiemmin on huomattu, että AppiumLibrary-kirjaston dokumentaatiossa on joitain epäselvyyksiä.

Kokonaisuutena kehys on tässä työssä esillä olleista käyttökelpoisin. Useiden kirjastojensa avulla sillä pystytään tekemään monipuolista testaamista. Lisäksi testiraportit ovat helppolukuisia ja kehyksen avulla voidaan tallentaa myös kuvakaappauksia sekä nauhoituksia testeistä. Haasteina on ollut ajoittainen herkkyys väärille negatiivisille sekä jo aiemmin mainittu syntaksi. Lisäksi AppiumLibrary-kirjastossa voisi olla enemmän avainsanoja tietynlaisiin tilanteisiin, kuten esimerkiksi avainsana, joka odottaa kunnes kohteena oleva elementti sisältää tietyn tekstin.

4.5 Testiautomaatiokehyksen valinta

Liitteen 1 pisteytysten pohjalta laskettiin jokaiselle vaihtoehdolle kokonaispisteet. Taulukosta 6 voidaan nähdä lopputulokset. Pisteistä laskettiin prosentuaalinen lopputulos kokonaispisteistä selkeyden vuoksi. Kuten taulukosta voidaan nähdä, mikään vaihtoehto ei

päässyt kovin hyviin tuloksiin Robot Framework-kehykseen verrattuna. Valinnalle ei ole asetettu mitään pisterajaa, vaan kokonaisuus ratkaisee. Robot Framework-kehyksen osalta on arvioinnissa huomioitava se, että kehys on ennestään jo tuttu ja käytetty, mutta sen arviointia pyrittiin myös peilaamaan aikaan, kun sen käyttö aloitettiin yrityksessä.

TULOKSET					
SELITE	MAX.PISTEET	KATALON	LAMBDATEST	OXYGEN	ROBOT FRAMEWORK
Lopputulos ilman painotusta	60	30,5	36	43	53,5
Lopputulos, painotettu	60	33,3	37,65	42,3	55,35
Lopputulos, prosentteina	100 %	56 %	63 %	71 %	92 %

Taulukko 6. Vertailun lopputulokset

Yrityksessä analysoitiin ja pohdittiin lopputuloksia ja mitkään työssä esillä olleet alustat ja kehykset eivät tuntuneet sopivilta korvaajilta Robot Framework-kehykselle. Jokaisesta löytyi omat haasteensa ja rajoitteensa. Vaihtaminen koettiin riskialttiiksi, koska testien uudelleenkirjoittaminen vaatii aikaa ja työvoimaa. Jos valittu työkalu ei sovellu täysin tarpeisiin, voidaan näitä resursseja tuhlata.

Tässä vaiheessa päädyttiin vielä analysoimaan ja tutkimaan, mikäli ei käytettäisi ollenkaan mitään testiautomaatiokehystä tai -alustaa, vaan testit ohjelmoitaisiin suoraan Selenium- ja Appium-työkalua vastaan. Tässä on kuitenkin omat haasteensa, kuten esimerkiksi se, että tämä monimutkaistaa testien kirjoittamista runsaasti ja vie täten enemmän aikaa. Se myös rajaa testiautomaation kehittämisen pitkälti yrityksen ohjelmoijien vastuulle, koska testaajilla ei välttämättä ole riittävää ohjelmointiosaamista. Ilman ulkoista alustaa tai kehystä testien tuloksista ei myöskään saada visuaalisia testiraportteja.

Luonnollisesti edelleen haasteena on Robot Framework-kehyksen AppiumLibrary-kirjasto, jota käytetään mobiilitestiautomaatiossa. Tätä voidaan paikata ottamalla avuksi toinen sovelluksiin erikoistunut lisäkirjasto, mikä on rakennettu AppiumLibrary-kirjaston päälle ja laajentaa sen toiminnallisuutta.

Työtä aloittaessa erityisenä huolena oli AppiumLibrary-kirjaston pysähtynyt kehitystyö. Se olisi voinut muodostua ongelmaksi, koska kyseinen kirjasto hyödyntää Appium-työkalua, josta tehdään uutta 2.0-versiota. Tästä olisi voinut aiheutua yhteensopivuusongelmia. Työtä kirjoittaessa AppiumLibrary-kirjasto sai kuitenkin uuden päivityksen, joten toistaiseksi kehitys näyttäisi jatkuvan taas.

AppiumLibrary-kirjaston päivityksistä ei kuitenkaan pitäisi aiheutua merkittävää haittaa käytännössä, mutta kaikki 2.0-version mukana tulevat muutokset eivät ole vielä tarkalleen tiedossa tämän työn kirjoitushetkellä. Toistaiseksi vaikuttaa siltä, että siirtymän pitäisi pääasiallisesti olla sujuva. Tämänhetkiseen testiautomaatioon vaikuttavat isoimmat muutokset liittyvät ajurien asennukseen ja testien alustuksen määrittelyihin. Ensimmäinen kohta tulee vain huomioida dokumentaatiossa. Toiseen kohtaan on dokumentaatiossa annettu ohjeet, miten määrittelyjä tulee jatkossa käyttää. Tilannetta kuitenkin tutkitaan ja seurataan. Toistaiseksi käytössä on vielä Appium-työkalun 1.0-versio. (Appium 2022.)

5 Yhteenveto ja pohdinta

Testiautomaation merkitys ja hyödyllisyys on huomattu yrityksessä ja sen käyttöä aiotaan lisätä myös tulevaisuudessa. Tästä syystä yrityksessä on haluttu saada varmennus sille, että käytettävä testiautomaatiokehys on heidän tarpeisiinsa nähden paras mahdollinen, mitä tällä hetkellä on saatavilla.

Työn päätavoitteena oli etsiä mahdollista korvaajaa tällä hetkellä käytössä olevalle testiautomaatiokehykselle. Vaikka tähän tavoitteeseen ei päästy, saatiin työstä vahvistus, että alun perin valittu kehys on ollut sopiva valinta vaatimuksiin ja olosuhteisiin nähden. Robot Framework-kehyksen käytössä on omat haasteensa ja rajoitteensa, mutta parempaa vaihtoehtoa ei tällä hetkellä ole saatavilla.

Työn pohjalta yrityksessä heräsi hyödyllistä ja monipuolista keskustelua testiautomaatiosta ja sen käytänteistä, kuten esimerkiksi testiautomaation suunnittelusta, ylläpidosta ja dokumentaatiosta. Jatkokehityksenä yritykselle aiotaan tehdä kattava testiautomaatio-opas, järjestää koulutusta kehyksen käyttöön sekä päivittää nykyisiä käytössä olevia testisettejä.

Testiautomaatio-oppaan avulla voidaan vastata osaan testiautomaatiossa vastaan tulleista haasteista. Se tulee selkeyttämään testiautomaatiotapausten suunnittelua, ylläpitoa, ympäristön pystytystä sekä antamaan vastauksia yleisimpiin ongelmatilanteisiin. Testitapauksia pyritään myös suunnittelemaan helpommin ylläpidettäviksi.

Testiautomaatiokehysten tilannetta ja kehitystä tullaan seuraamaan tulevaisuudessakin aktiivisesti, mutta tällä hetkellä pyritään parhaan mukaan hyödyntämään nykyistä kehystä ja edellä mainittujen keinojen avulla ratkaisemaan vastaan tulleita haasteita sekä hyödyntämään testiautomaatiota laajemmin ohjelmistokehitysprosessissa.

Lähteet

Appium. 2022. Migrating from Appium 1.x to Appium 2.x. Viitattu 24.10.2022. Saatavissa https://appium.github.io/appium/docs/en/2.0/guides/migrating-1-to-2/

ARI Systems. 2015. Process Models in Software Engineering. Viitattu 11.9.2022. Saatavissa https://aripd.com/posts/process-models-in-software-engineering

Atlassian. 2022. Software testing in continuous delivery. Viitattu 11.9.2022. Saatavissa https://www.atlassian.com/continuous-delivery/software-testing

Bushnev, Y. 2019. Top 15 UI Test Automation Best Practices. Blogi. Blazemeter. Viitattu 25.9.2022. Saatavissa https://www.blazemeter.com/blog/ui-test-automation

Cerberus. 2020. Cerberus-testiautomaatiokehyksen dokumentaatio. Viitattu 15.10.2022. Saatavissa https://cerberustesting.github.io/documentation_en.html

Chopra, R. 2018. Software Quality Assurance. E-kirja. Mercury Learning & Information. Viitattu 11.9.2022. Saatavissa rajoitetusti http://primo.lut.fi/lab

Da Silva Paternoster, M. 2022. Pros and Cons of Automated Testing. UIlicious. Blogi. Viitattu 24.9.2022. Saatavilla https://uilicious.com/blog/pros-cons-automated-testing/

IBM. What is software testing? Viitattu 18.9.2022. Saatavissa https://www.ibm.com/topics/software-testing

Katalon. 2022a. About Katalon Platform. Viitattu 15.10.2022. Saatavissa https://docs.katalon.com/docs

Katalon 2022b. Best Practices for Test Automation | 2022 Tester's Checklist. Viitattu 25.9.2022. Saatavissa https://katalon.com/resources-center/blog/test-automation-best-practices

Katalon. 2022c. Katalon Platform Overview. Viitattu 15.10.2022. Saatavissa https://katalon.com/katalon-platform

Katalon. 2022d. Katalon Platform Pricing. Viitattu 15.10.2022. Saatavissa https://katalon.com/pricing

Katalon. 2022e. Software Test Automation Frameworks | 6 Common Types. Viitattu 24.9.2022. Saatavissa https://katalon.com/resources-center/blog/test-automation-framework

Kauppalehti. 2022. Sade Innovations Oy. Viitattu 1.11.2022. Saatavissa https://www.kauppalehti.fi/yritykset/yritys/sade+innovations+oy/28557273

LambdaTest. 2022a. LambdaTest-alusta ja testiautomaatio. Viitattu 16.10.2022. Saatavissa https://www.lambdatest.com/automation-testing

LambdaTest. 2022b. LambaTest-alustan hinnoittelu. Viitattu 16.10.2022. Saatavissa https://www.lambdatest.com/pricing

LambdaTest. 2022c. LambdaTest-alustan ominaisuudet. Viitattu 16.10.2022. Saatavissa https://www.lambdatest.com/feature

Lodewyks, L. 2020. Smarter Test Automation: Part 2. Blogi. Inspired Testing. Viitattu 18.9.2022. Saatavissa https://www.inspiredtesting.com/news-insights/insights/398-smarter-test-automation-part-2

NovateUS. 2022. Top 15 Software Testing Best Practices. Viitattu 18.9.2022. Saatavissa https://novateus.com/blog/top-15-software-testing-best-practices/

Oxygen. 2022. Oxygen-testiautomaatiokehyksen julkaisut. Viitattu 18.10.2022. Saatavissa https://github.com/oxygenhq/oxygen-ide/releases

Oxygen. 2021. What is Oxygen?. Viitattu 18.10.2022. Saatavissa https://docs.oxygenhq.org/about/what-is-oxygen

Perfecto. 2021. 5 BDD Testing Frameworks to Consider. Viitattu 24.9.2022. Saatavissa https://www.perfecto.io/blog/bdd-testing-frameworks

Pittet, S. The different types of software testing. Atlassian. Viitattu 11.9.2022. Saatavissa https://www.atlassian.com/continuous-delivery/software-testing/types-of-software-testing

Robot Framework. Robot Framework-kehyksen kotisivut. Viitattu 28.9.2022. Saatavissa https://robotframework.org/

Sahu, A. What Makes Software Testing and QA Important for Business? Blogi. Viitattu 18.9.2022. Saatavissa https://www.westagilelabs.com/blog/why-is-software-testing-and-qa-important-for-any-business/

Saucelabs. 2022. Choosing the Right Mobile Test Automation Framework. Blogi. Viitattu 25.9.2022. Saatavissa https://saucelabs.com/blog/choosing-the-right-mobile-test-automation-framework

Smartbear. 2022a. Test Automation Best Practices. Viitattu 25.9.2022. Saatavissa https://smartbear.com/learn/automated-testing/best-practices-for-automation/

Smartbear. 2022b. Test Automation Frameworks. Viitattu 21.9.2022. Saatavissa https://smartbear.com/learn/automated-testing/test-automation-frameworks/

Testim. 2022. Your Complete Guide to Test Automation Frameworks. Blogi. Viitattu 21.9.2022. Saatavissa https://www.testim.io/blog/test-automation-frameworks/

UiPath. Test Automation Best Practices. Blogi. Blazemeter. Viitattu 25.9.2022. Saatavissa https://docs.uipath.com/test-suite/docs/test-automation-best-practices

Liite 1. Testien tulokset ja pisteytykset

VALINTAKRITEERI	ARVIOINNIN OSA-ALUEET		MAX. PISTEET	KATALON	LAMBDATEST	OXYGEN	ROBOT FRAMEWORK
LUOTETTAVUUS	Esimerkkitesti, web		2,5	2,5	2,5	2,5	2,5
	Esimerkkitesti, mobiili		2,5	2,5	0	2,5	2,5
	Oma testi, web		2,5	0	2,5	0	2,5
	Oma testi, mobiili		2,5	0	0	0	2,5
JOUSTAVUUS	Hybridikehys		2	2	0	2	2
	Mahdollisuus luoda omia avainsanoja tai toimintoja		8	8	8	8	8
MONIPUOLISUUS	Avainsanojen tai toimintojen määrä		5	5	2,5	2,5	5
	Kirjastot ja liitännäiset		5	2,5	5	2,5	5
KÄYTTÖÖNOTTO JA OPPIMISKYNNYS	Käyttöönoton helppous		5	0	5	5	5
	Oppimiskynnys		5	0	2,5	5	2,5
DOKUMENTAATIO JA TUKI	Dokumentaatio ja oppaat		4	2	2	2	2
	Tuki		2	2	2	0	0
	Laajasti käytetty		2	2	2	0	2
	Hyvin päivitetty		2	2	2	1	2
HINNOITTELU	Ilmainen Alle 1000$/vuosi Alle 2000$/vuosi Yli 2000$/vuosi	10 5 2,5 0	10	0	0	10	10