Tampereen yliopisto

Iiro Inkinen

IOT-LAITTEIDEN MALLIPOHJAINEN JA AVAINSANAOHJATTU TESTAAMINEN

Kandidaatintyö
Informaatioteknologian ja viestinnän tiedekunta
Tarkastaja: Joonas Multanen
Toukokuu 2024

TIIVISTELMÄ

Iiro Inkinen: IoT-laitteiden mallipohjainen ja avainsanaohjattu testaaminen
Kandidaatintyö
Tampereen yliopisto
Tieto- ja sähkötekniikan tutkinto-ohjelma, tietotekniikka
Toukokuu 2024

Kiinnostus esineiden Internetiin kasvaa jatkuvasti yritysten ja kuluttajien keskuudessa. Halu kehittää korkealaatuisia tuotteita nopealla aikataululla luo painetta tuotteiden testausprosessiin. Suorittamalla testaamisen tehokkaammin yrityksillä on mahdollisuus varmistaa tuotteiden riittävä laatu aikataulupaineista huolimatta ja saavuttaa siten kilpailuetua. Tutkielman tarkoituksena on tarkastella, miten esineiden Internetin laitteita voi testata käyttäen kahta testiautomaation menetelmää: mallipohjaista testaamista ja avainsanaohjattua testaamista.

Tutkielman ensimmäisessä osiossa esitetään teoriaa siitä, miksi ja miten esineiden Internetin laitteita testataan. Toteuttamismenetelminä esitellään avainsanaohjattu testaaminen ja mallipohjainen testaaminen. Avainsanaohjattu testaaminen automatisoi testitapausten suorittamisen kuvaamalla järjestelmän toiminnallisuutta korkean tason kuvausten, avainsanojen, avulla. Mallipohjainen testaaminen puolestaan automatisoi myös testitapausten suunnittelun, kun testitapaukset luodaan algoritmillisesti järjestelmän tiloja kuvaavasta mallista. Avainsanaohjatun testaamisen huomataan olevan yksinkertainen testausmenetelmä, joka myös jakaa testaamisen selkeästi suunnitteluun ja toteutukseen. Avainsanaohjattujen testijoukkojen kehitys ja ylläpito voi kuitenkin vaatia paljon resursseja. Mallipohjaisen testaamisen huomataan olevan tehokas ja helposti ylläpidettävä testaamisen automatisoinnin menetelmä. Mallipohjaisen testaamisen oppiminen ja käyttöönotto voi kuitenkin osoittautua haastavaksi.

Tutkielman toisessa osiossa esitetään tapaustutkimus erään esineiden Internetin laitteen virtaprofiilien testaamisesta käyttäen avainsanaohjattua testaamista ja mallipohjaista testaamista. Tutkimuksessa havaitaan, että avainsanaohjatun testaamisen käyttö voi olla tuottavaa usein ajettavien testitapausten toteuttamiseen. Mallipohjainen testaaminen puolestaan voi olla hyvä menetelmä monimutkaisten testitapausten luomiseen tai testattavan järjestelmän staattiseen testaamiseen.

Avainsanat: esineiden Internet, testiautomaatio, mallipohjainen testaaminen, avainsanaohjattu testaaminen

ALKUSANAT

Haluan kiittää Treon OY:tä mahdollisuudesta työn tekemiseen. Erityisesti haluan kiittää testauspäällikkö Vili Wileniä työn ohjauksesta ja valvonnasta sekä työtä mahdollistamassa olleita työkavereita.

Lisäksi haluan kiittää koulun puolelta työtä ohjannutta ja tarkastanutta Joonas Multasta sekä kandiseminaariryhmän jäseniä tuesta työn tekemisen aikana.

Tampereella, 16. toukokuuta 2024

Iiro Inkinen

SISÄLLYSLUETTELO

1. JOHDANTO

Esineiden Internet on tuore kehitysaskel trendissä, jossa tietoa kerätään ja käsitellään yhä pienemmistä laitteista koostuvissa ja laajemmalle hajautuneissa järjestelmissä [1]. Esineiden internetin laitteet, eli IoT-laitteet (engl. Internet of Things), voivat olla mitä vain teollisen sensorin ja kauppalistan tekevän jääkaapin väliltä. Sekä IoT-laitteiden määrä että sovellusalueen markkinoiden kokonaisarvo ovat kasvaneet merkittävästi kuluneen vuosikymmenen aikana [2].

Kuten muutkin järjestelmät, myös IoT-laitteista koostuvat järjestelmät täytyy testata. Vain testaamalla voidaan varmistua, että järjestelmä vastaa sille määriteltyjä vaatimuksia ja oletuksia. Jos järjestelmä ei vastaa vaatimuksia ja oletuksia, eli on huonolaatuinen, aiheutuu siitä kustannuksia sekä maineellista haittaa [3]. Samaan aikaan yritykset pyrkivät saamaan järjestelmänsä markkinoille mahdollisimman nopeasti [4]. Tästä ristiriidasta seuraa asetelma, jossa testaaminen ei saa merkittävästi viivästyttää järjestelmän julkaisua, mutta sen on taattava järjestelmän riittävä laatu siitä huolimatta. Siksi testaamisen tehokkuus herättää yrityksissä kiinnostusta.

Testaamista voi tehostaa testiautomaatiolla. Testiautomaatiossa testit kirjoitetaan muotoon, jota tietokone voi lukea. Näin testien suorittaminen ei vaadi ihmistyötä. Vaikka testien automatisointi vaatii enemmän työtä kuin yhden testikierroksen manuaalinen suorittaminen, säästää automatisointi aikaa testikierrosten toistuessa [3].

Testiautomaation toteuttamiseen on käytettävissä paljon työkaluja ja niiden soveltuvuus testaamiseen riippuu järjestelmän luonteesta. IoT-laitteet asettavat testaamiselle omia haasteitaan. Ensiksi IoT-laitteet ovat sulautettuja järjestelmiä, jolloin testattavaan järjestelmään kuuluu ohjelmiston lisäksi elektroniikkaa. Toiseksi IoT-laitteet toimivat usein itsenäisesti riippumatta ihmissyötteistä. Kolmanneksi IoT-laitteet ovat alttiita kyberhyökkäyksille, mikä tekee tietoturvallisuudesta tärkeän kriteerin laitteiden kehityksessä [5].

Mallipohjainen testaaminen on paljon tutkittu testiautomaation toteutusmenetelmä erityisesti sulautettujen järjestelmien testaamisessa, ja tutkimustulokset osoittavat menetelmän toimivan [6]. Mallipohjaisen testaamisen ammattilaisia on kuitenkin vaikeampi löytää yksinkertaisempiin testausmenetelmiin verrattuna [7]. Avainsanaohjattu testaaminen on puolestaan suosittu menetelmä muun muassa yksinkertaisuutensa ansiosta, jolloin kokemattomasta testaajastakin tulee nopeasti tuottava [8]. Siis, kuten manuaalisesta tes-

taamisesta automaattiseen testaamiseen siirryttäessä, syntyisi yrityksille kustannuksia avainsanaohjatusta testaamisesta mallipohjaiseen testaamiseen siirtymisestä. Analogiaa täydentää se, että mallipohjainen testaaminen automatisoi testitapausten luontivaiheen, joka on avainsanaohjatussa testaamisessa manuaalista työtä.

Kandidaatintyössä tarkastellaan IoT-laitteiden testausta avainsanaohjatulla ja mallipohjaisella testausmenetelmällä. Tavoitteena on selvittää, miten toistaiseksi avainsanaohjattua menetelmää käyttänyt yritys voisi hyötyä mallipohjaisesta testaamisesta. Työn rakenne on seuraava: Luvussa 2 aihetta pohjustetaan IoT-laitteiden automaattiseen testaamiseen liittyvällä teorialla. Luvuissa 3 ja 4 esitellään avainsanaohjattu ja mallipohjainen testausmenetelmä sekä esimerkkityökalut kummankin toteuttamiseen. Luvussa 5 suoritetaan tapaustutkimus, jossa erään IoT-laitteen virtaprofiilien testaus suoritetaan avainsanaohjattua ja mallipohjaista testausta hyödyntäen.

2. IOT-LAITTEIDEN TESTAAMINEN

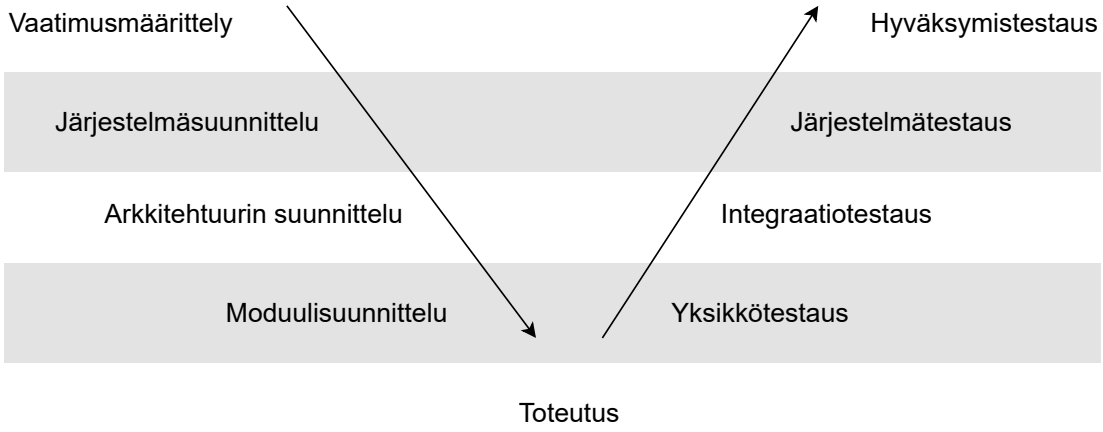
Järjestelmien testaaminen on osa yritysten laadunvarmistusprosessia. Kokonaisuudessaan prosessin tavoite on varmistaa, että sidosryhmät ovat riittävän tyytyväisiä tuotteeseen [9]. Liiallinen tyytyväisyys ei siis ole sen tavoiteltavampaa kuin tyytymättömyys. Laadunvarmistuksen, ja siten testaamisen, korkean tason tavoite onkin maksimoida yrityksen voitot [1].

Testaaminen voi lisätä tuotteen arvoa löytämällä siitä virheitä, jotka muuten löytyisivät vasta tuotteen käyttöönoton jälkeen. Tavoitteeseen viittaa myös yleinen testaamisen määritelmä: ohjelman suorittaminen niin, että tarkoituksena on löytää virheitä. Parhaat testit ovat siis niitä testejä, jotka löytävät ohjelmasta virheitä. Voi jopa ajatella, että läpi menneet testit ovat turhia, eivätkä lisää tuotteen arvoa. [4] Hyvin toteutetuille testeille on kuitenkin aina käyttöä. Vaikka testit eivät voi osoittaa tuotteen olevan virheetön, antavat ne tietoa tuotteen laadusta; Läpäistyt testit lisäävät luottamusta siihen, että tuote on julkaisukelpoinen [10].

2.1 Testausprosessi

Kuvassa 2.1 esitetty V-malli on perinteinen tapa kuvata testaamisen osuutta järjestelmän kehitysprosessissa. Vasemmalla on listattuna järjestelmän toteutuksen vaiheet ja oikealla niistä jokaiseen liittyvä testauksen vaihe. Kummallakin puolella abstraktiotaso on ylhäällä suurin ja tarkentuu alaspäin mentäessä. V-mallia on käytetty järjestelmän kehitysvaiheiden kuvaamiseen ainakin vuodesta 1991, jolloin se yhdistettiin vesiputousmallisesti ylhäältä alas tehtäviin järjestelmiin [11]. Malli on kuitenkin hyödyllinen testaamiskokonaisuuksien ja eri tasoisten testien tavoitteiden havainnollistamiseen, vaikka vaiheiden järjestys olisikin erilainen ketteriä menetelmiä käyttävässä projektissa.

Yksikkötestaus ja integraatiotestaus ovat matalan tason testausta. Yksikkötestauksessa testaamisen kohteena on järjestelmän yksikkö, kuten funktio tai moduuli, ja integraatiotestauksessa kohteena on näiden yksiköiden yhteistoiminta. Matalan tason testaaminen pyrkii selvittämään, onko järjestelmä toteutettu suunnitelman mukaan. Testien tavoitteena on siis löytää eroavaisuuksia suunnitellun ja toteutetun järjestelmän välillä. Yksikkö- ja integraatiotestaus löytääkin yleensä enemmän virheitä järjestelmän käytännön toteutuksesta kuin myöhemmät testausvaiheet [4].



Kuva 2.1. *V-malli: järjestelmän kehitys- ja testauskokonaisuuksien välisen yhteyden havainnollistamistapa.*

Kuitenkin, kuten kuvasta huomataan, järjestelmän käytännön toteutus on vasta viimeinen vaihe koko toteutusprosessia. Siksi tarvitaan vielä suurempiin kokonaisuuksiin keskittyvää testaamista, joka varmistaa, että järjestelmä on myös suunniteltu oikein.

Myers käyttää integraatiotestausta korkeamman tason testauksesta yleistä termiä *korkean tason testaus* (engl. high-order testing). Korkean tason testaus lähtee siitä oletuksesta, että ihmisten välinen vuorovaikutus aiheuttaa järjestelmään virheitä. Siksi ei ole syytä olettaa, että järjestelmästä laaditut suunnitelmat vastaisivat käyttäjän tarpeita. Korkean tason testaaminen pyrkii siis löytämään virheitä myös järjestelmän suunnitteluprosessista. [4]

V-mallista tarkasteltuna korkean tason testausta ovat järjestelmätestaus ja hyväksymistestaus. Järjestelmätestauksessa pyritään selvittämään, onko toteutettu järjestelmä sellainen kuin haluttiin toteuttaa. Hyväksymistestaus on puolestaan järjestelmän tilaajan vastuulla oleva testauskokonaisuus, jossa toteutettua järjestelmää verrataan määriteltyihin vaatimuksiin ja nykyisiin käyttäjätarpeisiin. [4]

Mikä V-mallista ei käy selville, on se, että myös järjestelmän testaaminen on suunniteltava – käytännön toteutus on myös testaamisessa vasta viimeinen vaihe. Testaamisen suunnitteluprosessin sisältö riippuu testattavasta järjestelmästä, mutta se voi sisältää esimerkiksi seuraavia [4]:

- Asetetaan testaamisen tavoitteet, lopetuskriteerit ja aikataulu.
- Päätetään, kuka on vastuussa mistäkin testaamisen osasta.
- Suunnitellaan testaamisen ympäristö ja valitaan testityökalut.
- Sovitaan, miten virheet ja testien tulokset raportoidaan.

Vaikka suunnitteluprosessin lopputuotteena on testaussuunnitelma, on itse suunnitteleminen työvaiheen tärkein ulosanti: se pakottaa selventämään omia suunnitelmia ennen

testaamisen aloittamista [1].

Ketteriä menetelmiä käytettäessä on myös huomioitava, että V-mallissa esitettyjä kokonaisuuksia ei suoriteta kerralla koko järjestelmän osalta. Koska järjestelmä toteutetaan ominaisuus kerrallaan, toistuvat suunnittelun ja toteutuksen vaiheet pienempinä kokonaisuuksina ja useampaan kertaan projektin aikana. Uusien ominaisuuksien lisääminen voi myös rikkoa jo olemassa olevia ominaisuuksia, joten jo kerran suoritettuja testejä voi olla tarpeen toistaa.

Uusien ominaisuuksien aiheuttamia virheitä olemassa olevissa ominaisuuksissa kutsutaan regressioiksi, ja niihin keskittyvää testaamista regressiotestaukseksi. Käytännössä regressiotestauksessa voi toistaa aikaisemmin läpäistyjä testejä, jolloin testin epäonnistuminen paljastaa regression tapahtuneen.

2.2 Testiautomaatio

Laajasti tulkittuna testaamisen automatisointi tarkoittaa minkä tahansa testaamiseen liittyvän toiminnan automaattista suorittamista. Testiautomaatiolla on siten myös sama tavoite kuin testaamisella yleisesti: tuottaa yritykselle voittoa. Testiautomaation käyttöä voi kuitenkin perustella muillakin tavoin kuin löytyneiden virheiden määrän kasvulla. Testien automaattinen suorittaminen voi esimerkiksi lyhentää testikierrosten pituutta ja mahdollistaa siten nopeamman palautteen antamisen kehittäjille. [12]

Testien suorittamisen automatisointi on investointi, koska se vaatii enemmän työtä kuin manuaalisen testauskierroksen toteuttaminen. Siksi ennen testien automatisointia on syytä arvioida, mitä testejä halutaan automatisoida. Automatisoitavaksi kannattaa valita testejä, joita suoritetaan usein – jokaisen testikierroksen myötä yksittäisen kierroksen hinta laskee. Lisäksi testiautomaation avulla testattavaan järjestelmään voi kohdistaa syötteitä enemmän ja pidempään kuin manuaalisesti, mistä voi olla hyötyä ei-toiminnallisten vaatimusten testaamisessa. [3]

Testien suorittamisen automatisointi voi siten olla kannattava investointi esimerkiksi seuraavissa kolmessa tapauksessa [3][12]:

1. Regressiotestaus (suoritetaan usein)
2. Aineisto-ohjattu testaus (samaa testitapausta toistetaan testidatan muuttuessa)
3. Luotettavuustestaus (järjestelmän toimintaa seurataan pitkään yhtäjaksoisesti)

Testien automaattisen suorittamisen lisäksi testiautomaatiota voi hyödyntää esimerkiksi automatisoimalla testikierroksen aloittamisen tai testitapausten suunnittelun. Testikierroksen aloittamisen voi automatisoida käyttämällä jatkuvaa integraatiota, jolloin esimerkiksi muutoksen työntäminen tietovarastoon aloittaa testit automaattisesti [12]. Testitapausten suunnittelun voi puolestaan automatisoida esimerkiksi mallipohjaisella testaamisella, jota

käsitellään luvussa 4.

Testiautomaatiota voi hyödyntää myös niin dynaamisessa kuin staattisessakin testaamisessa. Tutkielmassa keskitytään dynaamiseen testaamiseen, jossa virheitä yritetään löytää käyttämällä järjestelmää. On kuitenkin mahdollista suorittaa myös staattisesta testaamista, jossa virheitä yritetään löytää tutkimalla järjestelmän rakennuspalikoita, esimerkiksi analysoimalla ohjelmakoodia. [12]

On myös tärkeä huomioida, että testiautomaatio vaatii ylläpitämistä. Jos testien ylläpitämiselle ei varata resursseja testiautomaatiota toteutettaessa, vanhenevat ne testattavan järjestelmän muutoksien myötä ja automaatioon tehty investointi menetetään [12]. Ylläpidon tarve tarkoittaa myös sitä, että testiautomaation toteutusvaiheessa testien ylläpidettävyyteen kiinnitetty huomio helpottaa töitä tulevaisuudessa.

2.3 Soveltaminen IoT-ympäristössä

IoT-laitteet ovat järjestelmiä siinä missä muutkin järjestelmät, joten edellä esitetyt testaamisen perusteet pätevät myös IoT-ympäristössä [1]. IoT-laitteiden testausta suunniteltaessa ja toteutettaessa on kuitenkin syytä huomioida joitakin erityispiirteitä.

Ensiksi, IoT-laitteet ovat sulautettuja järjestelmiä ja sisältävät siten ohjelmien lisäksi elektroniikkaa. Vaikka elektroniikka tyypillisesti testataan erillään ohjelmiston ja koko järjestelmän testauksesta, vaikuttaa elektroniikan mukanaolo väistämättä testaamiseen. Esimerkiksi ohjelmiston ja elektroniikan välinen vuorovaikutus voi olla järjestelmän virheiden lähde. Lisäksi uusien elektroniikkaversioiden myötä laitteiden yhteensopivuudesta voi tulla ongelma. [1]

Toiseksi, IoT-laitteiden toimintaympäristö on poikkeuksellinen. Esimerkiksi tietokoneohjelman suoritusympäristö on käyttöjärjestelmä, jolloin ohjelman suoritus ei ole riippuvainen fyysisestä ympäristöstään. IoT-laitteet puolestaan vuorovaikuttavat ympäristönsä kanssa erilaisten sensoreiden välityksellä. Testaamista varten on siis usein rakennettava fyysinen ympäristö, jossa laitteen vastaanottamia ärsykkeitä voidaan kontrolloida. Pienissä projekteissa testaajan pöydälle mahtuva ympäristö voi riittää, mutta suuremmissa projekteissa pelkkä testiympäristö voi olla miljoonainvestointi [1].

Kolmanneksi, IoT-laitteiden tietoturvallisuus on vaikea ongelma ratkaistavaksi. Sulautettujen järjestelmien prosessointiteho ja virrankulutus ovat rajoitettuja, joten tietoliikenteessä käytettävät salausalgoritmit täytyy pitää yksinkertaisina. Laitteet eivät myöskään yleensä sisällä virustorjunnan kaltaisia ohjelmia. [2] Heikko tietoturva yhdistettynä IoT-laitteiden suureen määrään tekee niistä hyvän alustan esimerkiksi palvelunestohyökkäyksien toteuttamiselle [5]. IoT-laitteita testattaessa on siis syytä keskittää erityistä huomioita laitteen tietoturvallisuuteen.

Lopuksi on syytä mainita, että ei ole olemassa yhtä parasta tai oikeaa tapaa testata ohjelmistoja ja järjestelmiä. Erilaisilla menetelmillä ja työkaluilla on omat hyvät ja huonot puolensa. Lisäksi toiset menetelmät ja työkalut sopivat yhteen projektiin paremmin kuin toiseen. Sopivien menetelmien ja työkalujen valitseminen vaatii testaajalta kokemusta ja ammattitaitoa, sekä jatkuvaa uuden opettelua [1]. Seuraavaksi tarkastellaan kahta testaamisen menetelmää, joita on mahdollista käyttää IoT-laitteiden automaattisessa testauksessa: avainsanaohjattua testausta ja mallipohjaista testausta.

3. AVAINSANAOHJATTU TESTAAMINEN

Avainsanaohjatun testaamisen käyttäminen testiautomaation toteuttamiseen on prosessi, jossa testitapausten suunnittelu ja testien toteutus on erotettu toisistaan. Jako suunnitteluun ja toteutukseen perustuu siihen, että testitapaukset voi suunnitella testaamisen tai järjestelmän asiantuntija, jonka ei tarvitse osata ohjelmoida. Testiautomaation toteutuksesta vastaavan on puolestaan osattava ohjelmointia, mutta hänen ei tarvitse osata luoda hyviä testitapauksia.

Avainsanaohjattua testaamista voi periaatteessa käyttää kaikentasoiseen testaamiseen, koska avainsanoja voi luoda kaikentasoisista testattavan järjestelmän operaatioista [13]. Käytännössä avainsanaohjattu testaaminen voi kuitenkin olla tarpeettoman monimutkainen työkalu esimerkiksi yksikkötestaukseen, jossa yhtä funktiota kutsutaan parametrien vaihtuessa.

3.1 Testien luominen

Yhteinen osa testien suunnittelua ja toteutusta ovat valmiit testitapaukset. Ne toimivat suunnittelun lopputuotteena ja toteutuksen syötteenä. Esimerkki valmiin testitapauksen sisällöstä on [14]:

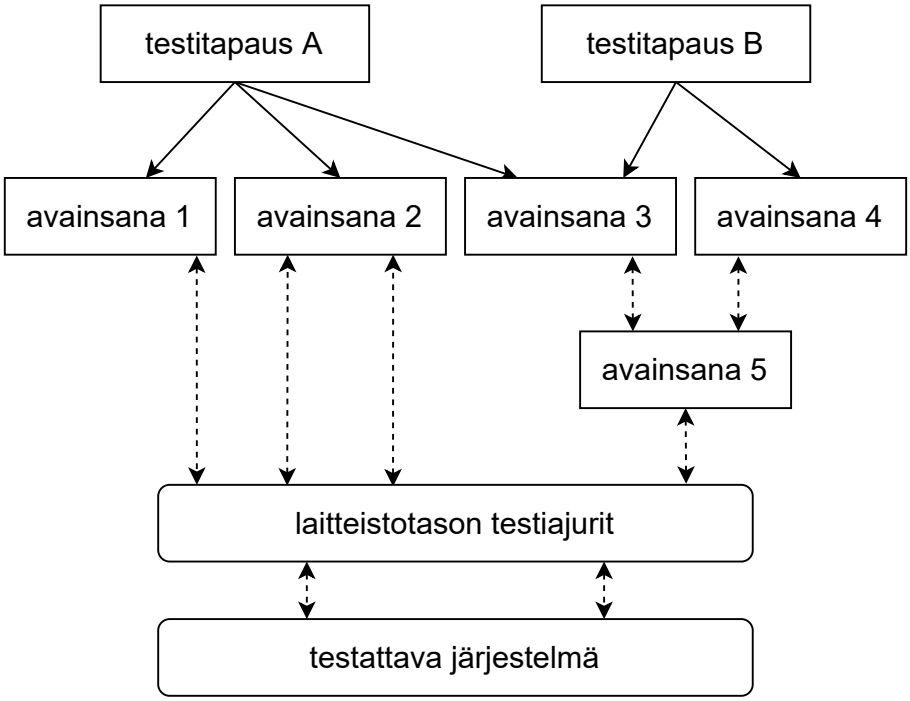
1. Yhdistä palvelimelle
2. Kirjaudu sisään (käyttäjä: ironman, salasana: 1234567890)
3. Varmista, että kirjautuminen onnistui (nimi: Tony Stark)

Kuten nähdään, testitapaus on lista selkokielisiä ohjeita. Testitapaus ei itse asiassa edes ota kantaa siihen, miten testi suoritetaan: sen voisi suorittaa myös muilla tavoin, esimerkiksi manuaalisesti. Testin manuaalista suorittamista voi myös kutsua avainsanaohjatuksi testaamiseksi [13]. Jos testin suoritus halutaan kuitenkin automatisoida, täytyy testin vaiheista luoda koneellisesti tulkittavissa olevia avainsanoja.

Avainsanat ovat korkean tason kuvauksia testattavassa järjestelmässä suoritettavista toiminnoista [13]. Toisin sanoen, avainsanat ovat abstrakteja testiajureita. Esimerkiksi esitetyn testin vaihe 2 abstrahoi koko kirjautumisprosessin yhden avainsanan alle, joka tarvitsee parametreikseen vain käyttäjänimen ja salasanan. Avainsanan käyttäjän ei siis tarvitse murehtia, miten kirjautumistiedot syötetään sivulle tai miten kauan odotetaan, jos

yhteys ei muodostukaan välittömästi. Testitapausta ei myöskään välttämättä tarvitse päivittää testattavan järjestelmän muuttuessa, koska muutokset voidaan tehdä avainsanatasolla.

Kuvassa 3.1 on esitetty yksinkertaisen avainsanaohjatun testijoukon toteutus. Ylimpänä on kaksi testitapausta, jotka koostuvat avainsanoista. Huomataan, että avainsana 3 on osa kumpaakin testitapausta; Uudelleenkäytön mahdollisuus onkin yksi avainsanojen suunnittelun tavoitteista [8]. Avainsana 5 puolestaan havainnollistaa, että avainsanat voivat kutsua toisia avainsanoja. Avainsanojen alapuolelle on erotettu vielä selkeyden vuoksi laitteistotason testiajurit. Niitä voisi nimittää myös matalan tason avainsanoiksi [13].



Kuva 3.1. *Avainsanaohjatun testaamisen tasot.*

3.2 Hyödyt ja haasteet

Testien suunnittelun ja toteutuksen erottaminen toisistaan on yksi etu avainsanaohjatun testaamisen käytössä. Erottelun ansiosta toteutettavan tuotteen asiantuntijat ja testiautomaation asiantuntijat voivat keskittyä omaan osaamisalueeseensa. Samalla avainsanat tarjoavat kuitenkin yhteisen ja selkeän rajapinnan korkean ja matalan tason välisen kommunikaation helpottamiseksi. [13]

Toinen etu on avainsanaohjatun testaamisen yksinkertaisuus testaamismenetelmänä. Yksinkertaisuuden ansiosta testaajista tulee tuottavampia nopeasti, vaikka heillä ei olisi aikaisempaa kokemusta menetelmän käyttämisestä [8]. Sovittujen avainsanojen käyttö testitapauksissa nopeuttaa myös testien toteuttamista, kun saman toiminnon suorittamista kuvataan aina samalla tavalla [13].

Toisaalta avainsanaohjatun testaamisen toteuttaminen ja ylläpitäminen vaatii paljon työtä [13]. Ylläpidon tarvetta lisää se, että pienikin muutos testattavassa järjestelmässä voi helposti rikkoa useita testitapauksia, koska alin avainsanakerros on yhdistetty niin tiiviisti testattavaan järjestelmään [8]. Haasteet ovat samoja kuin testiautomaatiolle määritettiin luvussa 2.2. Siten avainsanaohjattua testaamista kannattaa hyödyntää erityisesti sellaisten testien toteuttamiseen, joita ajetaan usein tai joita on vaikea toteuttaa manuaalisesti.

3.3 Eräs työkalu: Robot Framework

Yksi avainsanohjattuun testaamiseen kehitetty työkalu on Robot Framework. Työkalua kehittää Robot Framework Foundation ja sen lähdekoodi on julkaistu Apache-2.0 lisensoituna. Dokumentaatiossa työkalun kerrotaan olevan alusta- ja sovellusriippumaton ja soveltuvan erityisesti hyväksyntätestaukseen. Lisäksi mainostetaan, että työkaluun löytyy paljon lisäosia erityyppisiin testaustarpeisiin. [14]

Robot Framework pohjautuu Python-kieleen, mutta testitapaukset ja avainsanat luodaan työkalun omalla notaatiolla .robot- ja .resource-päätteisiin tiedostoihin. Yhtä .robot-päätteistä tiedostoa kutsutaan testijoukoksi, ja se voi sisältää yhden tai useampia testitapauksia. Testitapaukset taas koostuvat yhdestä tai useammasta avainsanasta. Avainsanoja voi kutsua kolmesta eri lähteestä:

1. Vain kyseisen testijoukon tarvitsemat avainsanat määritellään samassa tiedostossa testitapausten kanssa.
2. Yleiskäyttöiset avainsanat määritellään omissa tiedostoissaan, joista useat testijoukot voivat tuoda niitä käyttönsä.
3. Valmiita avainsanakirjastoja voi ladata netistä, jolloin testijoukot voivat tuoda niitä käyttöönsä edellisten tavoin.

Avainsanat sisältävät kutsuja toisiin avainsanoihin ja testiajureihin. Testiajureita voi avainsanojen tavoin ottaa käyttöön valmiina kirjastoina, tai niitä voi luoda itse esimerkiksi Python- ja Java-kielillä. [14]

Ohjelmassa 3.1 esitetään Robot Frameworkilla luotu testitapaus, joka toteuttaa luvussa 3.1 suunnitellun testitapauksen vaiheet. Kuten nähdään, Robot Frameworkin notaatio on luonnollisen kielen kaltaista. Lukemalla avainsanakutsut järjestyksessä saa myös hyvän käsityksen siitä, mitä toimintoja testitapaus suorittaa, ja mikä on odotettu lopputulos. Suunniteltujen vaiheiden lisäksi testin lopussa tapahtuu testin purkaminen (engl. teardown), joka palauttaa testattavan järjestelmän tunnettuun tilaan testin suorituksen päätyttyä. Purkamisella mahdollistetaan se, etteivät peräkkäiset testitapaukset vaikuta toisiinsa.

Robot Framework sisältää valmiita työkaluja muuhunkin testaamisen automatisointiin kuin testitapausten automaattiseen suorittamiseen. Se esimerkiksi luo testien suorittamisen

```
1 // Yksi testitapaus koostuu nimestä ja listasta avainsanakutsuja.
2 Onnistunut kirjautuminen
3     avainsanat.Yhdistä
4     // Parametrit erotetaan avainsanan nimestä välilyönneillä.
5     avainsanat.Kirjaudu sisään    ironman    1234567890
6     avainsanat.Sivun Pitäisi Olla Auki    etusivu
7     // Testi päättyy ensimmäiseen avainsanaan, jonka suoritus
8     // epäonnistuu, mutta Teardown-avainsana suoritetaan aina.
9     [Teardown]    avainsanat.Katakaise Yhteys
```

Ohjelma 3.1. *Robot Frameworkilla luotu testitapaus onnistuneesta sisäänkirjautumisesta [14].*

yhteydessä automaattisesti html-lokin ja -raportin testien tuloksista. Lisäksi itseluoduista avainsanatiedostoista saa työkalun avulla luotua html-dokumentaation. [14]

4. MALLIPOHJAINEN TESTAAMINEN

Mallipohjainen testaaminen on menetelmä, jossa testattava järjestelmä mallinnetaan koneellisesti tulkittavaan muotoon. Luotua mallia käytetään syötteenä, josta luodaan testitapauksia algoritmillisesti. [10] Mallipohjaisen testaamisen taustalla on ajatus, että testitapausten manuaalinen luominen on ylimääräistä työtä, koska testitapausten luomisenkin ensimmäinen vaihe on testattavan järjestelmän toiminnan ymmärtäminen. Käyttämällä testiautomaation syötteenä järjestelmästä luotua mallia staattisten testitapausten sijaan, voi tietokone teoriassa luoda mallin perusteella testattavan järjestelmän kaikki mahdolliset testitapaukset.

Mallipohjaisen testaamisen erityispiirre on siis se, että testitapaukset luodaan testattavaa järjestelmää kuvaavasta mallista. Luotujen testitapausten suorittamiseen voi siis käyttää haluamaansa menetelmää, ja testausta voi yhä kutsua mallipohjaiseksi testaamiseksi. Koska avainsanaohjatun testaamisen toteutuksen syötteenä käytetään valmiita testitapauksia, on mallipohjaisen ja avainsanaohjatun testaamisen yhdistäminen suoraviivaista.

4.1 Testien luominen

Mallipohjaisen testaamisen toteuttamiseen tarvitaan asiantuntemusta testattavasta järjestelmästä, mallintamisesta ja testiautomaatiosta. Testattavan järjestelmän ja mallintamisen asiantuntemusta tarvitaan mallin luomiseen ja validointiin. Mallin oikeellisuus on tärkeää varmistaa, koska kaikki siitä johdetut testitapaukset odottavat testattavan järjestelmän toimivan samalla tavalla. Testiautomaation toteutus avainsanaohjatulla testaamisella vaatii luvussa 3 käsiteltyä osaamista.

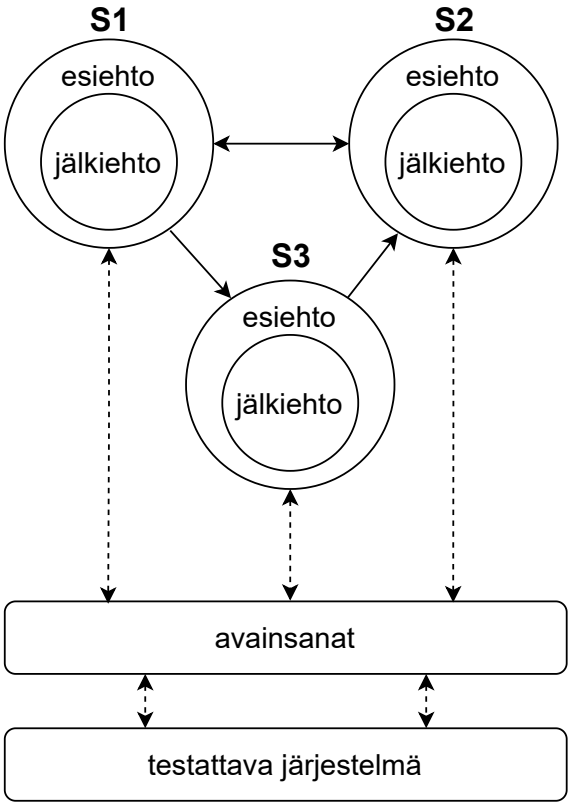
Testien luominen mallipohjaista testausta käyttäen tapahtuu viidessä vaiheessa [10]:

1. Järjestelmän mallintaminen
2. Abstraktien testitapausten luonti
3. Ajettavien testitapausten luonti
4. Testien ajaminen
5. Testin tulosten raportointi

Järjestelmän mallinnusvaiheessa testattavasta järjestelmästä luodaan ohjelmallisesti tul-

kittavissa oleva mallinnus. Lähteenä järjestelmän ominaisuuksille käytetään spesifikaatioita. Mallia luotaessa on tärkeää kuvata testattavaa järjestelmää sopivan abstraktilla tasolla, jotta malliin pohjautuvista testitapauksista tulee järkeviä. [10]

Yksi mallinnustapa on luoda järjestelmästä äärellinen tilakone [15]. Kuvassa 4.1 on esitetty yksinkertainen esimerkki äärellistä tilakonetta käyttävästä mallista. Mallissa on kolme tilaa: S1, S2 ja S3. Nämä tilat koostuvat esiehdosta ja jälkiehdosta. Esiehto määrittää, milloin tilaan voidaan siirtyä, ja jälkiehto määrittää, mitä toimia tilaan siirtyminen aiheuttaa. Siirtymät tilojen välillä on kuvattu nuolilla, jotka edustavat myös esiehtojen toteutumista: Tilassa S1 molempien tilojen S2 ja S3 esiehdot toteutuvat, joten tilasta S1 voidaan siirtyä tilaan S2 tai S3. Tilasta S3 voidaan puolestaan siirtyä vain tilaan S2, koska tilan S1 esiehto ei toteudu.



Kuva 4.1. *Mallin osat: tilat, esiehdot, jälkiehdot, siirtymät ja avainsanat.*

Abstraktien testitapausten luontivaiheessa mallista johdetaan algoritmillisesti testejä valittujen kriteerien perusteella. Kriteerit liittyvät usein erilaisiin kattavuuksiin, kuten siihen, miten suuri osa kaikista mahdollisista siirtymistä tilojen välillä testissä suoritetaan. Tuloksena luontivaiheesta saadaan lista, joka kuvaa järjestyksessä testin aikana suoritettavat siirtymät. [10] Tilakonetta käyttävässä kuvan 4.1 mallissa testitapausten luomisen hoitava algoritmi tutkii tilojen esiehtojen toteutumista ja suorittaa jälkiehtojen määrittämiä muutoksia luodessaan polkuja tilojen välillä.

Ajettavien testitapausten luontivaiheessa suunnitellut siirtymät mallissa muutetaan testat-

tavassa järjestelmässä ajettaviksi komennoiksi. Rajapintana mallin siirtymien ja testattavan järjestelmän välillä toimii testiajurit. [10] Kuvan 4.1 mukaisesti jokaiseen tilaan liitetään tilaan siirtymisestä aiheutuvat avainsanakutsut. Muuttamalla testien luontivaiheessa syntynyt lista järjestelmän tiloja listaksi avainsanakutsuja saadaan ajettavia testitapauksia.

Testien ajovaiheessa ajettavat testitapaukset suoritetaan ja testattavan systeemin reagointi syötteisiin tallennetaan. Systeemin reaktiosta tarkastetaan, vastattiinko syötteeseen odotetulla tavalla. Jos reaktio on oikea, testitapaus merkitään läpäistyksi, muussa tapauksessa testitapaus merkitään epäonnistuneeksi. [10]

Testin tulosten raportointi on testauksen viimeinen vaihe. Erityisen tärkeää on tallentaa epäonnistuneista testitapauksista tarpeelliset tiedot, jotta testattavan systeemin viat voi niiden avulla paikantaa. [10]

Kun mallipohjainen testaaminen suoritetaan niin, että vaiheet tapahtuvat yksi kerrallaan esitetyssä järjestyksessä, kutsutaan sitä yhteydettömäksi testaamiseksi (engl. offline testing). Yhteydettömässä testaamisessa koko testitapaus siis luodaan ennen sen suorittamista. Jos taas luotu abstrakti testiaskel muutetaan ajettavaksi testiaskeleeksi ja suoritetaan ennen seuraavan askeleen luomista, eli vaiheita 2–4 toistetaan, kyseessä on yhteydellinen testaaminen (engl. online testing). Yhteydellistä testaamista käyttämällä testiaskeleiden luomisessa on mahdollista reagoida siihen, miten testattava järjestelmä on vastannut aikaisempien askeleiden syötteisiin. [10]

4.2 Hyödyt ja haasteet

Mallipohjaista testaamista on sovellettu järjestelmien testaamiseen 1970-luvulta lähtien. Esimerkiksi Jessop ja muut mallinsivat testattavia järjestelmiä suunnattuina graafeina ja generoivat niistä testitapauksia koneellisesti ATLAS-nimisellä työkalulla [16]. Menetelmää he kutsuivat malliviitteiseksi testaamiseksi (engl. model-referenced testing), mutta järjestelmä sisälsi jo useita uudemmistakin työkaluista löytyviä ominaisuuksia. Mallipohjaisen testaamisen suosio on kuitenkin kasvanut merkittävästi 2000-luvulla, minkä on mahdollistanut yleinen kiinnostuneisuus testaamisesta sekä mallipohjaisen kehityksen yleistyminen [10].

Ensimmäiset mallipohjaisen testaamisen toteutukset johtivat testitapaukset samoista malleista, joita käytettiin järjestelmän toteutuksessa. Pian kuitenkin huomattiin, että tällä menetelmällä samat virheet päätyivät sekä testeihin että testattavaan järjestelmään. Uudemmissa toteutuksissa testaamista varten siis luodaan oma mallinsa riippumatta siitä, onko systeemi jo mallinnettu toteuttamista varten. [7] Testaamista varten luotu oma malli on siis välttämätön tehdä, jotta sitä voidaan käyttää myös korkean tason testaamiseen.

Mallipohjaisen testaamisen suurimmat hyödyt liittyvät testitapausten luonnin abstrahoin-

tiin. Kun malli luodaan, määritellään siinä jokainen järjestelmän tila ja siirtymä tilojen välillä tasan kerran. Staattiset testitapaukset luovat tähän verrattuna redundantteja kopioita. Mallipohjaisella lähestymistavalla säästetään siis työtä testien luontivaiheessa. Lisäksi testien ylläpitäminen helpottuu, sillä muutos mallin tilassa päivittyy automaattisesti kaikkiin mallista johdettaviin testitapauksiin. [15]

Toinen hyöty juontuu mallin luontiprosessiin. Kun mallia luodaan spesifikaatioiden pohjalta, voi niistä löytyvät puutteet ja epäselvyydet tuoda esiin virheitä järjestelmän suunnitteluvaiheesta. Mallipohjainen testaaminen toteuttaa siis korkean tason testaamisen tehtävää jo mallinnusvaiheessa. Koska testattavaa järjestelmää ei suoriteta mallinnusvaiheessa, voi sen luokitella staattiseksi testausvaiheeksi. [12]

Mallipohjaisen testaamisen yksi suurimmista haasteista on sen poikkeavuus yleisesti käytetymmistä testausmenetelmistä. Esimerkiksi osaavia työntekijöitä on vaikeampi löytää, ja testikehyksen vaihtaminen vaatii merkittäviä muutoksia yrityksen testauskulttuuriin. Mallipohjaiselle testaamiselle asetetaan myös kokeiluvaiheessa usein liian suuria tavoitteita, mikä johtaa pettymykseen, kun menetelmä ei ratkaisekaan kaikkia yrityksen ongelmia. [7]

Useita pieniä pilottiprojekteja pidetään mallipohjaisen testaamisen onnistuneen käyttöönoton edellytyksenä. Näissä projekteissa tulisi keskittyä pieniin, mutta selkeisiin tavoitteisiin, kuten testitapausten ylläpidettävyyden parantumiseen. Lisäksi mallipohjaisen testauksen pilotointiin pitäisi osallistua pienempi joukko ammattilaisia, ennen kuin muutosta aletaan toteuttaa yritystasolla. [7]

4.3 Eräs työkalu: fMBT

Yksi mallipohjaiseen testaamiseen kehitetty työkalu on fMBT (engl. free Model-Based Testing). Työkalu on kehitetty Intelillä ja julkaistu LGPL-2.1 lisenssillä. Dokumentaatiossa työkalun kerrotaan voivan luoda ja ajaa testejä automaattisesti. Lisäksi mainostetaan, että työkalu sopisi hyvin moniin käyttökohteisiin, mukaan lukien hajautettujen järjestelmien testaamiseen. [17]

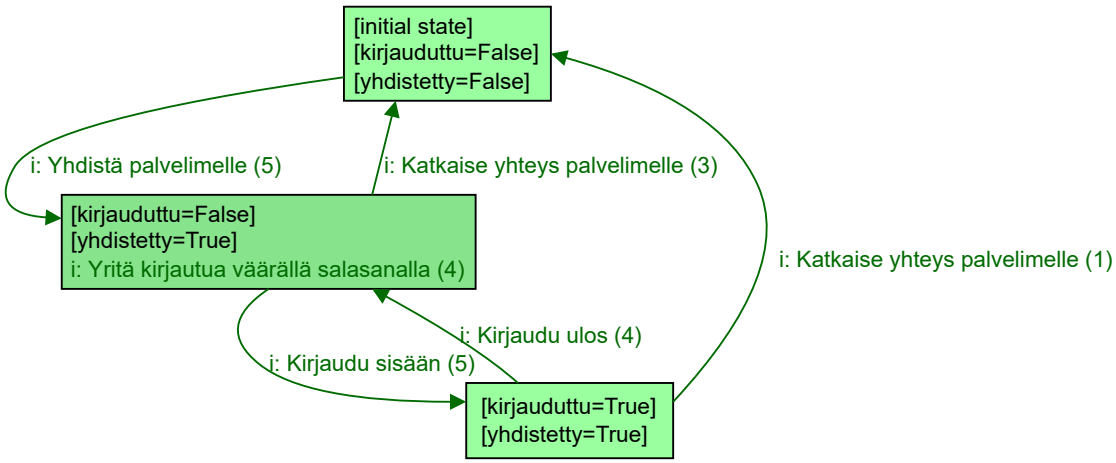
Mallinnus fMBT:llä pohjautuu Python-kieleen ja käyttää erityistä AAL-notaatiota (engl. Adapter Actions Language). Malli luodaan kirjoittamalla AAL-notaatiolla joukko tiloja, jotka sisältävät esi- ja jälkiehdon sekä adapterin. Esi- ja jälkiehto toimivat samoin kuin kuvan 4.1 mallissa. Testien ajonaikaiseen vuorovaikutukseen testattavan systeemin kanssa malli käyttää adapteri-komponentteja. Ne ovat esimerkiksi Python-kielellä kirjoitettuja testiajureita, jotka toimivat rajapintana testattavaan järjestelmään. [17]

Kuvassa 4.2 on esitetty fMBT-työkalulla luotu esimerkkimalli, josta voisi johtaa esimerkiksi luvun 3.1 testitapauksen. Laatikot kuvaavat järjestelmän tiloja, ja nuolet kuvaavat toimintoja, jotka muuttavat järjestelmän tilaa. Lisäksi mallissa on toiminto väärällä salasanalla kirjautumisen yrittämiseen, jonka suorittaminen ei muuta mallin tilaa. Toiminnon perässä

```
1  input "Kirjaudu␣sisään" {
2      // Esiehto: On yhdistetty palvelimelle,
3      // mutta ei kirjauduttu sisään.
4      guard() {
5          return yhdistetty and not kirjauduttu
6      }
7      // Jälkiehto: Tilan suorittamisen jälkeen
8      // ollaan kirjauduttu sisään.
9      body() {
10         kirjauduttu = True
11     }
12     // Testattavan järjestelmän ohjaamisesta vastaa
13     // ulkoinen Python-kirjasto.
14     adapter() {
15         ajuri.kirjaudu_sisaan("ironman", "1234567890")
16         ajuri.onko_sivu_auki("etusivu")
17     }
18 }
```

Ohjelma 4.1. *"Kirjaudu sisään"-tilan määrittely fMBT-työkalun AAL-notaatiolla.*

oleva numero kertoo, kuinka monta kertaa kyseinen toiminto suoritetaan luodussa testitapauksessa. Ohjelmassa 4.1 on esitetty mallin “Kirjaudu sisään” -toiminnon määrittely AAL-notaatiolla.



Kuva 4.2. *fMBT-työkalulla luotu malli testattavasta järjestelmästä.*

Vaikka fMBT on kehitetty käytettäväksi Linux-ympäristössä, työkalua voi käytännössä hyödyntää myös muilla alustoilla. Tietovarastosta löytyy nimittäin valmis Dockerfile työkalun ajamiseen docker-ympäristössä. [17] Lisäksi Linux-sovelluksia voi suorittaa Windows-ympäristössä Windows Subsystem for Linux -virtuaalikoneella [18].

5. TAPAUSTUTKIMUS: VIRTAPROFIILIEN TESTAAMINEN AVAINSANAOHJATULLA JA MALLIPOHJAISELLA MENETELMÄLLÄ

Tapaustutkimuksen tavoitteena on selvittää, miten toistaiseksi avainsanaohjattua testaamista hyödyntänyt yritys voisi hyötyä mallipohjaisesta testaamisesta. Tavoitteen pohjalta syntyi kolme tutkimuskysymystä:

1. Mitä etuja avainsanaohjatulla testaamisella on mallipohjaiseen testaamiseen verrattuna?
2. Mitä etuja mallipohjaisella testaamisella on avainsanaohjattuun testaamiseen verrattuna?
3. Miten avainsanaohjattuun testaamiseen tehtyjä investointeja voi hyödyntää mallipohjaisessa testaamisessa?

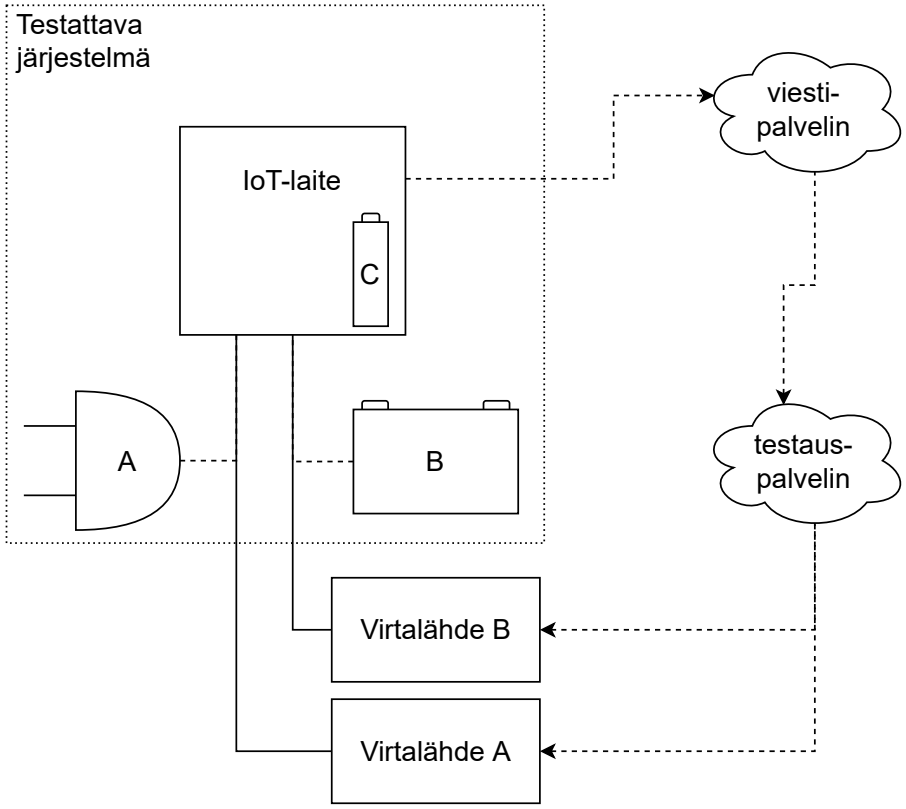
Mallipohjaisen ja avainsanaohjatun testaamisen vertailemiseksi toteutettiin tapaustutkimus, jossa oli kaksi vaihetta: Ensin testattavalle järjestelmälle toteutettiin avainsanaohjattu testiautomaatio Robot Frameworkilla; Sitten staattisten testitapausten tilalle luotiin mallipohjaisia testitapauksia fMBT-työkalua käyttäen. Menetelmien vertailun lisäksi toteutusjärjestyksellä pyrittiin tutkimaan, miten olemassaolevia avainsanakirjastoja on mahdollista hyödyntää mallipohjaisessa testaamisessa.

5.1 Testattava järjestelmä ja testiympäristö

Testattava järjestelmä ja testiympäristö on esitetty kuvassa 5.1. Testattava järjestelmä on IoT-laite, joka voi ottaa virtaa kolmesta lähteestä. Ne ovat seuraavat:

1. Verkkovirta (A)
2. Ulkoinen akku (B)
3. Sisäinen akku (C)

Akkuvirran säästämiseksi laitteen on tunnistettava, mistä lähteestä se ottaa virtansa. Havaitessaan muutoksen laite vaihtaa käytössä olevaa virtaprofiilia ja ilmoittaa muutoksesta lähettämällä viestin.



Kuva 5.1. *Testattava järjestelmä ja testiympäristö*

```
1 // Alkutilanne: virtaprofiili C käytössä.
2 Sisäinen akku, ulkoinen akku, verkkovirta
3     Akku päälle
4     Odota ilmoitusta virtaprofiilista    B
5     Verkkovirta päälle
6     Odota ilmoitusta virtaprofiilista    A
```

Ohjelma 5.1. *Robot Frameworkilla luotu testitapaus virtaprofiilien vaihdosta.*

Tapaustutkimuksessa toteutettavat testit testaavat, että laite raportoi käytössä olevan virran lähteen muutokset oikein. Virtaprofiilien vaihdokset aiheutetaan simuloimalla verkkovirtaa ja ulkoista akkua ohjelmallisesti ohjattavilla virtalähteillä. Virtaprofiilin vaihtumisesta ilmoittaminen varmistetaan palvelimelta, joka vastaanottaa laitteiden lähettämät viestit.

5.2 Avainsanaohjattu toteutus

Avainsanohjattua testaamista varten suunniteltiin testijoukko, ja testit automatisoitiin Robot Frameworkilla. Testitapaukset koostuivat virtalähteiden käynnistämisistä ja sammuttamisista sekä laitteen lähettämien viestien verifioinnista. Eräs testitapauksista on esitetty ohjelmassa 5.1.

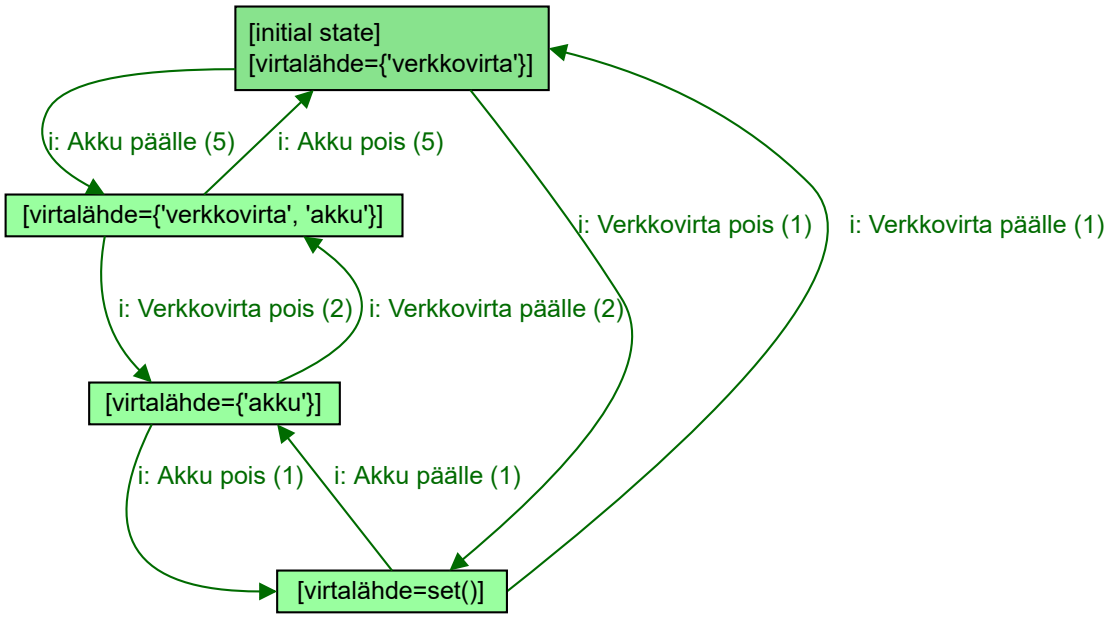
Testien toteuttamiseen käytettiin kolmen tyyppisiä avainsanoja: itse luotuja avainsanoja,

ulkoisia avainsanoja sekä laitteistotason testiajureita. Avainsanakirjastojen hierarkia on esitetty kuvassa 5.2. Kuvan avainsanojen abstraktiotaso on ylhäällä suurin ja pienenee alaspäin mentäessä.

Kuva 5.2. *Käytettyjen avainsanojen hierarkia.*

5.3 Mallipohjainen toteutus

Mallipohjaista testaamista varten testattavasta järjestelmästä luotiin malli fMBT-työkalulla. Malli on esitetty kuvassa 5.3.

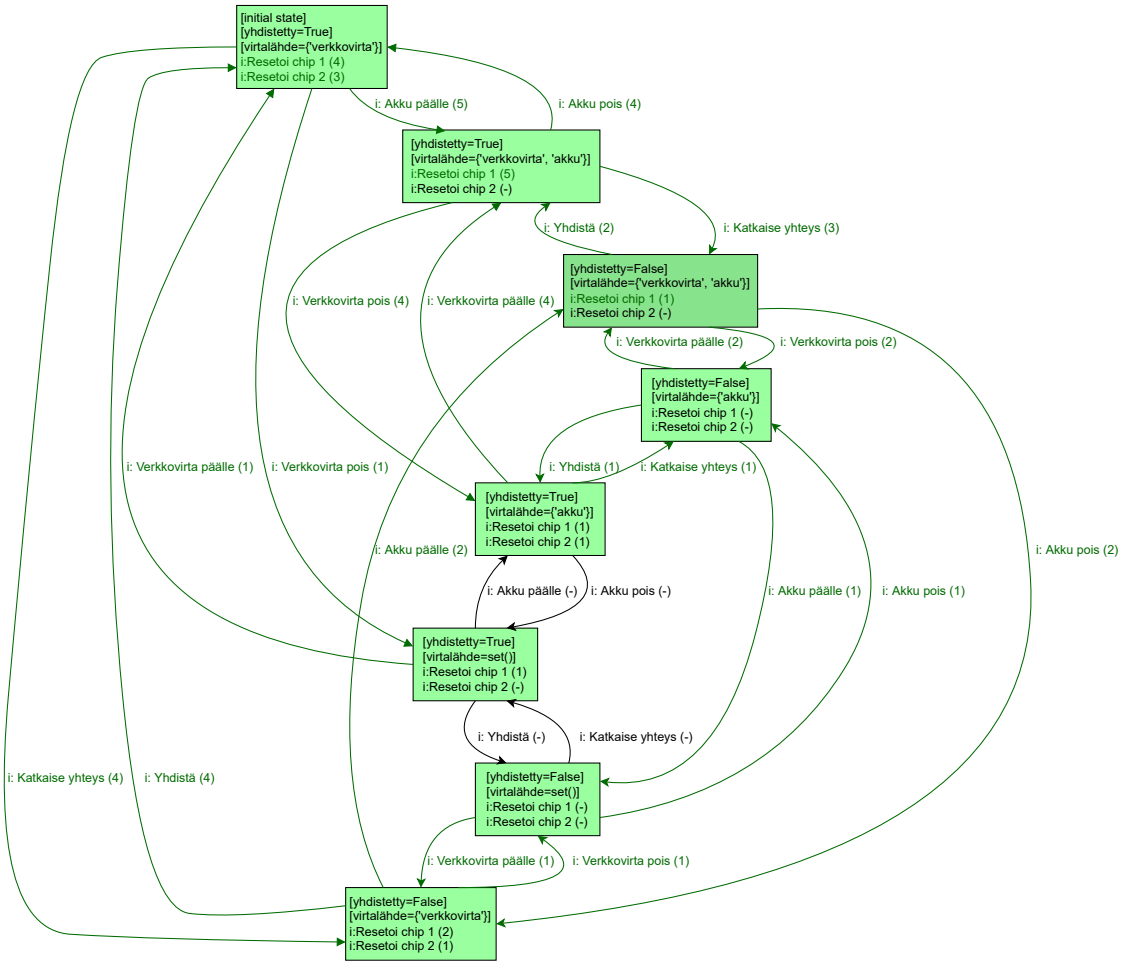


Kuva 5.3. *Virtaprofiilit mallinnettuna fMBT-työkalulla.*

Mallista johdettiin Robot Framework -testitapauksia käyttäen Robot Frameworkin Python-rajapintaan tehtyä adapteria, jollainen löytyy myös fMBT:n tietovarastosta [17]. Valmis adapteri on kuitenkin kirjoitettu Robot Frameworkin versiolle 3, joten adapterista täytyi

luoda uusi versio avainsanojen toteutuksessa käytetyn Robot Framework 6:n tukemiseksi. Luodut testitapaukset käyttivät samoja avainsanoja kuin aiemmin luodut staattiset testitapaukset. Valmiit avainsanat kuitenkin yhdistettiin mallista luotuun testijoukkoon ylimääräisen avainsanakirjaston välityksellä, jotta mallin adapteri-osia ei tarvitsisi päivittää avainsanoja ylläpidettäessä.

Mallipohjaisen testaamisen kykyä mallintaa testattavan järjestelmän virhetiloja testattiin lisäämällä malliin kahdenlaisia häiriötilanteita: Ensimmäisessä laitteen verkkoyhteys katkaistiin, ja toisessa jokin laitteen mikroprosessoreista käynnistettiin uudelleen. Virhetilanteet sisältävä malli on esitetty kuvassa 5.4. Toiminnon perässä oleva “-” tarkoittaa, että kyseistä toimintoa ei suoriteta kertaakaan luodussa testitapauksessa.



Kuva 5.4. *Virtaprofiilit ja virhetilanteet mallinnettuna fMBT-työkalulla.*

5.4 Tulokset ja pohdinta

Tapaustutkimuksen tulokset ovat samansuuntaisia aikaisemman tutkimustiedon kanssa. Mallipohjainen testaaminen oli monimutkaisempaa toteuttaa, mutta valmistuttuaan sillä oli etuja avainsanaohjattuun testaamiseen verrattuna.

Tutkimuksessa toteutettujen testien kattavuutta ja työmäärää on havainnollistettu taulukossa 5.1. Testien kattavuutta arvioidaan testijoukossa suoritettavien testiaskeleiden määrän avulla. Yhdeksi testiaskeleeksi lasketaan testattavalle järjestelmälle annettava syöte ja siihen saatavan reaktion varmistaminen. Testien luomisen vaatimaa työtä arvioidaan testejä varten kirjoitettujen uusien ohjelmakoodirivien avulla. Mukaan ei ole laskettu viestipalvelimen kanssa kommunikointiin käytettäviä avainsanoja, koska ne oli luotu jo ennen tapaustutkimusta. Myöskään Robot Framework -adapteria ei ole laskettu koodiriveihin mukaan, koska se on enemmän testityökalun kuin testijoukon osa. Avainsanaohjatun toteutuksen testijoukon koodirivit on laskettu toteutukseen mukaan, koska ne luotiin manuaalisesti; Mallista johdetut testitapaukset on puolestaan jätetty laskuista pois.

Taulukko 5.1. *Toteutettujen testien sisältämät testiaskeleet ja koodirivit.*

Toteutus	Testiaskeleita	Uusia koodirivejä (Robot Framework)	Uusia koodirivejä (fMBT)	Uusia koodirivejä yhteensä
Avainsanaohjattu	14	339	-	339
Mallipohjainen ilman virhetilanteita	25	292	137	429
Mallipohjainen virhetilanteilla	56	363	182	545

Mitä etuja avainsanaohjatulla testaamisella on mallipohjaiseen testaamiseen verrattuna?

Avainsanaohjatun testaamisen toteuttaminen vaati vähemmän työtä kuin mallipohjainen testaaminen. Ainoa vain avainsanaohjattuun testaamiseen kuulunut työvaihe oli staattisten testitapausten suunnittelu ja kirjoittaminen Robot Frameworkilla. Testattavan järjestelmän odotetun toiminnan yksinkertaisuuden ansiosta työvaihe oli yksinkertaisempi kuin järjestelmän mallintaminen.

Avainsanaohjattu testaaminen vaati myös vähemmän osaamista kuin mallipohjainen testaaminen. Testitapausten suunnittelu ja järjestelmän mallintaminen olivat toisiinsa rinnastettavia työvaiheita, koska molemmissa täytyi selvittää, miten testattavan järjestelmän kuuluisi toimia eri tilanteissa. Testitapausten kirjoittaminen Robot Frameworkilla ei kuitenkaan vaatinut uutta osaamista avainsanojen luontiin verrattuna. Suunnitellun mallin luominen fMBT-työkalulla vaati puolestaan uudenlaisen syntaksin opettelua.

Tutkimuksessa käytetyistä työkaluista Robot Frameworkin oheistyökalut olivat monipuolisempia. Erityisesti automaattisesti luodut kattavat testiraportit helpottavat virheiden raportointia. Tosin, tutkimalla muitakin mallipohjaisen testaamisen työkaluja tai investoimal-

la kaupalliseen toteutukseen voisi oheistyökaluja saada myös mallipohjaisen testauksen työkaluun.

Mitä etuja mallipohjaisella testaamisella on avainsanaohjattuun testaamiseen verrattuna?

Mallipohjainen testaaminen sisälsi vähemmän toistoa staattisiin testitapauksiin verrattuna. Esimerkiksi avainsanakutsua simuloidun verkkovirran päälle kytkemiseen käytettiin vain kerran: verkkovirtatilan adapterikoodissa. Toiston vähyyden vaikutukset ylläpidettävyyteen selviävät tosin vasta ajan kuluessa.

Myös uusien ominaisuuksien lisääminen testattavan järjestelmän malliin oli suoraviivaista. Yhdistämällä kolme yksinään toimivaa mallia saatiin neljäs, suurempi malli, joka kykenee testaamaan mallien yhteisvaikutuksia. Virhetilanteita simuloivien tilojen käyttö ei myöskään rajoitu vain yhteen malliin: verkkoyhteyden katkaisevat tilat voi yhdistää myös muihin testeihin, joissa laite lähettää viestejä palvelimelle.

Miten avainsanaohjattuun testaamiseen tehtyjä investointeja voi hyödyntää mallipohjaisessa testaamisessa?

Valmiit avainsanat olivat hyödyllisiä myös mallipohjaiaselle testaamiselle. Koska kummallakin menetelmällä luotujen testitapausten abstraktiotaso oli sama, riitti mallipohjaiseen testaamiseen siirtyessä korvata staattiset testitapaukset mallista johdetuilla testitapauksilla. Avainsanaohjattuun testaamiseen investoinut yritys ei siis menetä kuin testitapausten suunnitteluun kuluneen ajan mallipohjaiseen testaamiseen siirryttäessä.

Myös mallipohjainen testaaminen voi hyödyntää Robot Frameworkin oheistyökaluja. Näin yrityksen ei tarvitse luopua tutuksi tulleista testiraporteista. Lisäksi johtamalla mallista Robot Framework -testitapauksia, toimii esimerkiksi testien liittäminen jatkuvan integraation putkiin samoin kuin avainsanaohjattujen testijoukkojen.

Miten toistaiseksi avainsanaohjattua testaamista hyödyntänyt yritys voisi hyötyä mallipohjaisesta testaamisesta?

Mallipohjainen testaaminen on toimiva tapa lisätä testaamisen automaatioastetta avainsanaohjattua testaamista käyttävässä yrityksessä. Mallipohjaisen testausmenetelmän käyttö vaatii kuitenkin erilaisia oikeutuksia kuin testien ajamisen automatisointi, koska perusteena käytölle täytyy olla jokin mallinnuksesta saatava hyöty. Oikeutus voisi olla esimerkiksi monimutkainen tai usein muuttuva järjestelmä, jolloin staattisten testitapausten luomiseen ja ylläpitämiseen kuluu mallinnusta enemmän resursseja. Toisaalta oikeutuksen voisi saada ottamalla mallinuksen käyttöön järjestelmän suunnitteluvaiheen staattiseen testaamiseen. Mallintamista voisi tehdä samaan aikaan järjestelmän suunnittelun kanssa ja se voisi havaita suunnitteluvirheitä aikaisessa vaiheessa kehitysprosessia. Haluttua järjestelmää kuvaavaksi todettu malli olisi siten myös käytettävissä heti järjestelmän kehi-

tyksen alkaessa, jolloin testiautomaatio voisi pohjautua malliin alusta lähtien. Tiivistäen: malli sijoittuisi korkean ja matalan tason testaamisen väliin, sen luominen olisi korkean tason testausta ja siitä johdettaisiin matalan tason testejä.

6. YHTEENVETO

Avainsanaohjattu testaaminen ja mallipohjainen testaaminen soveltuvat IoT-laitteiden testaamisen automatisointiin. Avainsanaohjattu testaaminen on hyvä menetelmä testien suorittamisen automatisointiin, koska menetelmä on yksinkertainen ja erottaa suunnittelun toteutuksesta. Mallipohjainen testaaminen puolestaan on hyvä menetelmä testitapausten luomisen automatisointiin, koska sen avulla on mahdollista luoda monimutkaisempia testitapauksia kuin suunnittelemalla testitapaukset itse.

Molemmat menetelmät vaativat kuitenkin enemmän työtä kuin kerran tapahtuvaan yksinkertaisten testien manuaaliseen suunnitteluun ja toteutukseen kuluisi. Siksi mallipohjainen testaaminen soveltuu erityisesti sellaisiin tilanteisiin, joissa testitapausten suunnittelun ennustetaan vievän paljon resursseja järjestelmän elinkaaren aikana. Avainsanaohjattu testaaminen puolestaan soveltuu erityisesti sellaisten testien automatisointiin, joita ennustetaan toistettavan usein.

Tutkielmassa mallipohjainen testaaminen liitettiin olemassaolevaan, staattisia testitapauksia käyttäneeseen avainsanaohjattuun testikehykseen. Siten olisikin mielenkiintoista toteuttaa testattavan järjestelmän suunnitteluvaiheesta alkava mallipohjainen testaamisprojekti. Lisäksi mallipohjainen testaaminen toteutettiin tutkimuksessa yhteydettömänä testauksena, joten yhteydellisen testaamisen integrointi on vielä mahdollinen tapa parantaa mallipohjaista testaamista.

LÄHTEET

[1] Jon Duncan Hagar. “IoT System Testing: An IoT Journey from Devices to Analytics and the Edge”. 1st ed. Berkeley, CA: Apress L. P, 2022. ISBN: 978-1-4842-8275-5. DOI: 10.1007/978-1-4842-8276-2.

[2] Imran Makhdoom, Mehran Abolhasan, Justin Lipman, Ren Ping Liu, and Wei Ni. “Anatomy of Threats to the Internet of Things”. *IEEE Communications surveys and tutorials* 21.2 (2019), pp. 1636–1675. ISSN: 1553-877X. DOI: 10.1109/COMST.2018.2874978. URL: https://ieeexplore.ieee.org/document/8489954.

[3] Boby Jose. “Test Automation: A Practitioner’s Guide”. BCS, 2021. ISBN: 978-1-78017-545-4.

[4] Glenford J. Myers, Tom Badgett, and Corey Sandler. “The Art of Software Testing”. 3. Aufl. Hoboken: Wiley, 2011. ISBN: 978-1-118-03196-4. DOI: 10.1002/9781119202486.

[5] Ghadeer Murad, Aalaa Badarneh, Abdallah Qusef, and Fadi Almasalha. “Software Testing Techniques in IoT”. *2018 8th International Conference on Computer Science and Information Technology (CSIT)*. 2018 8th International Conference on Computer Science and Information Technology (CSIT). July 2018, pp. 17–21. DOI: 10.1109/CSIT.2018.8486149. URL: https://ieeexplore.ieee.org/abstract/document/8486149.

[6] Vahid Garousi, Michael Felderer, Çağrı Murat Karapıçak, and Uğur Yılmaz. “What We Know about Testing Embedded Software”. *IEEE Software* 35.4 (July 2018). Conference Name: IEEE Software, pp. 62–69. ISSN: 1937-4194. DOI: 10.1109/MS.2018.2801541. URL: https://ieeexplore.ieee.org/document/8405633.

[7] Ina Schieferdecker. “Model-Based Testing”. *IEEE Software* 29.1 (Jan. 2012), pp. 14–18. ISSN: 0740-7459. DOI: 10.1109/MS.2012.13. URL: https://ieeexplore.ieee.org/document/6123945.

[8] Renaud Rwemalika, Marinos Kintis, Mike Papadakis, Yves Le Traon, and Pierre Lorrach. “On the Evolution of Keyword-Driven Test Suites”. Book Title: 2019 12th IEEE Conference on Software Testing, Validation and Verification (ICST). IEEE, 2019, pp. 335–345. ISBN: 978-1-72811-736-2. DOI: 10.1109/ICST.2019.00040.

[9] Teeba Ismail Kh and Ibrahim I. Hamarash. “MODEL-Based Performance Quality Assessment for IoT Applications”. *International Journal of Interactive Mobile Technologies (iJIM)* 15.12 (June 18, 2021). Number: 12, pp. 4–20. ISSN: 1865-7923. DOI: 10.3991/ijim.v15i12.21287. URL: https://online-journals.org/index.php/i-jim/article/view/21287.

[10] Mark Utting, Alexander Pretschner, and Bruno Legeard. “A taxonomy of model-based testing approaches”. *Software testing, verification & reliability* 22.5 (2012), pp. 297–312. ISSN: 0960-0833. DOI: 10.1002/stvr.456.

[11] Kevin Forsberg and Harold Mooz. “The Relationship of System Engineering to the Project Cycle”. *INCOSE International Symposium* 1.1 (Oct. 1991), pp. 57–65. ISSN: 2334-5837, 2334-5837. DOI: 10.1002/j.2334-5837.1991.tb01484.x. URL: https://incose.onlinelibrary.wiley.com/doi/10.1002/j.2334-5837.1991.tb01484.x.

[12] Bernard Homès. “Advanced testing of systems-of-systems 2: practical aspects”. Computer engineering series. London, England ; John Wiley & Sons, Inc., 2022. ISBN: 978-1-394-18848-2.

[13] *ISO/IEC/IEEE International Standard - Software and systems engineering – Software testing – Part 5: Keyword-Driven Testing*. ISBN: 9781504408745 Pages: 1–69. 2016. DOI: 10.1109/IEEESTD.2016.7750539. URL: https://ieeexplore.ieee.org/document/7750539.

[14] *Robot Framework*. URL: https://robotframework.org/ (visited on 03/07/2024).

[15] Abbas Ahmad, Fabrice Bouquet, Elizabeta Fourneret, and Bruno Legeard. “Chapter One - Model-Based Testing for Internet of Things Systems”. *Advances in Computers*. Ed. by Atif M. Memon. Vol. 108. Elsevier, Jan. 1, 2018, pp. 1–58. DOI: 10.1016/bs.adcom.2017.11.002. URL: https://www.sciencedirect.com/science/article/pii/S0065245817300517.

[16] W. H. Jessop, J. R. Kane, S. Roy, and J. M. Scanlon. “ATLAS-An Automated Software Testing System”. *Proceedings of the 2nd international conference on Software engineering*. ICSE ’76. Washington, DC, USA: IEEE Computer Society Press, Oct. 13, 1976, pp. 629–635.

[17] *intel/fMBT*. URL: https://github.com/intel/fMBT (visited on 02/19/2024).

[18] craigloewen-msft. *Run Linux GUI apps with WSL*. URL: https://learn.microsoft.com/en-us/windows/wsl/tutorials/gui-apps (visited on 03/07/2024).