Tekoälyn hyödyntäminen käyttöliittymien testiautomaatiossa

Turun yliopisto
Tietotekniikan laitos
Tietojenkäsittelytiede, LuK-tutkielma
Maaliskuu 2024
Veikka Vilppala

TURUN YLIOPISTO
Tietotekniikan laitos

Veikka Vilppala: Tekoälyn hyödyntäminen käyttöliittymien testiautomaatiossa

Tietojenkäsittelytiede, LuK-tutkielma, 24 s.

Maaliskuu 2024

Käyttöliittymät toimivat rajapintana käyttäjien ja laajojen ohjelmistojen välillä. Käyttöliittymien keskeisen aseman vuoksi niiden perusteellinen testaaminen on välttämätöntä. Laajojen ohjelmistokokonaisuuksien testaamiseen hyödynnetään testiautomaatiota. Käyttöliittymätestaukseen tarkoitetussa testiautomaatiossa on ilmennyt ongelmia, jotka liittyvät monimutkaisten käyttöliittymien hallintaan ja testien ylläpitoon. Erityisesti dynaamiset käyttöliittymät ja erilaiset käyttöliittymäelementit voivat aiheuttaa haasteita automaattisten testien luomisessa ja suorittamisessa. Lisäksi testien herkkyys muutoksille käyttöliittymässä ja tarve päivittää testit vastaamaan jatkuvasti kehittyvää käyttöliittymää voivat vaikuttaa testauksen tehokkuuteen ja luotettavuuteen. Tekoälyn hyödyntäminen ohjelmistotuotannossa on selkeässä kasvussa siinä lähivuosina tehdyn edistyksen takia.

Tämän tutkielma on toteutettu kirjallisuuskatsauksena ja sen tarkoituksena on perehtyä käyttöliittymien testiautomaation ongelmiin ja tutustua, miten tekoälyä on sovellettu käyttöliittymien testiautomaatiossa. Lisäksi tutkielmassa pohditaan, onko löydetyistä tekoälyn sovelluksista apua käyttöliittymän testiautomaation ongelmiin.

Käyttöliittymän testiautomaation ongelmat keskittyvät sen kehittämiseen ja ylläpitoon. Testiautomaatiossa hyödynnettyjä tekoälyn sovellusalueita ovat mm. kuvantunnistus, kielimallit, geneettiset algoritmit, sekä statistiikka ja ennustavat mallit. Tekoälyn sovellusalueet tarjoavat uusia lähestymistapoja käyttöliittymien testiautomaation toteuttamiseen, mutta eivät toistaiseksi korjaa siinä esiintyviä ongelmia kokonaisuudessaan.

Asiasanat: Käyttöliittymätestaus, Testiautomaatio, Tekoäly

UNIVERSITY OF TURKU
Department of Computing

Veikka Vilppala: Tekoälyn hyödyntäminen käyttöliittymien testiautomaatiossa

Tietojenkäsittelytiede, LuK-tutkielma, 24 p.

March 2024

User interface testing plays a crucial role as an interface between users and complex software systems. Due to the central role of user interfaces, thorough testing is essential. Test automation is utilized for testing large software systems. However, there have been challenges in test automation specifically tailored for user interface testing, particularly concerning the management of complex interfaces and test maintenance. Dynamic interfaces and various interface elements can pose challenges in creating and executing automated tests. Additionally, the sensitivity of tests to changes in the interface and the need to update tests to keep pace with evolving interfaces can impact the efficiency and reliability of testing. The utilization of artificial intelligence in software development is experiencing significant growth due to advancements made in recent years.

This thesis is conducted as a literature review aiming to delve into the problems of user interface test automation and explore how artificial intelligence has been applied in this context. Furthermore, the thesis discusses whether the discovered applications of artificial intelligence offer assistance in addressing the problems of user interface test automation.

The problems of user interface test automation focus on its development and maintenance. Areas of artificial intelligence applied in test automation include image recognition, language models, genetic algorithms, as well as statistics and predictive models. These areas of artificial intelligence offer new approaches to implementing user interface test automation but do not yet completely solve the challenges it faces.

Keywords: User Interface Testing, Test Automation, Artificial Intelligence

Sisällys

1 Johdanto

Käyttöliittymä on keskeinen osa tuotettua ohjelmistoa. Käyttöliittymät toimivat rajapintana laajojen ohjelmistojen ja käyttäjien välillä, ja ovat näin erittäin kriittinen testattava osa ohjelmistoa. Laajojen ohjelmistokokonaisuuksien testaamiseen hyödynnetään testiautomaatiota manuaalisen testauksen ohella. Testiautomaation tarkoituksena on säästää aikaa ja tarjota järjestelmällinen tapa varmistua ohjelmiston kriittisimmistä toiminnoista.

1.1 Tutkielman tarkoitus

Nykypäivän ohjelmistotuotannolle on ominaista tuotteiden laajuus ja monimutkaisuus, mikä vaatii tehokasta ja luotettavaa testausprosessia varmistaen tuotteiden laadun ja käyttäjäkokemuksen. [1] Käyttöliittymillä on suuri merkitys käyttäjäkokemuksen kannalta niiden toiminnallisuuden ja suorituskyvyn vaikuttaessa suoraan siihen, miten käyttäjät vuorovaikuttavat ohjelmiston kanssa. Laajojen ja monimutkaisten ohjelmistojen käyttöliittymät vaativat laajaa testaamista. Testiautomaatio mahdollistaa tehokkaan ja toistettavan tavan varmistua käyttöliittymän toiminnallisuudesta eri ympäristöissä.

Testiautomaation käytössä on kuitenkin ilmennyt ongelmia. Näistä merkittävimpä-

nä voidaan pitää sen herkkyyttä testattavan ohjelmiston muutoksille. Ohjelmiston muutokset voivat vaikuttaa laajasti olemassa oleviin testiskripteihin ja tehdä niistä näin toimimattomia. Tämä saattaa johtaa jatkuvaan testiautomaation kehitys ja ylläpitotarpeeseen, joka kuluttaa resursseja varsinaisesta ohjelmiston testaamisesta. [2]

Tekoälyn käyttäminen ohjelmistokehityksen apuvälineenä on kasvanut huomattavasti varsinkin loppuvuonna 2022 julkaistun ChatGPT:n saavuttaman suosion ansiosta. Tekoälyn eri sovellusalueilla on potentiaalia mullistaa ohjelmistokehitysprosessia kokonaisuudessaan tarjoamalla ratkaisuja, apuvälineitä ja näin uusia lähestymistapoja prosessin eri osa-alueille. Ohjelmistotestaus ja varsinkin testiautomaatio on yksi näistä osa-alueista.

Tämän tutkielman tarkoituksena on kartoittaa käyttöliittymän testiautomaation keskeisimmät ongelmat, ja selvittää, miten tekoälyä on hyödynnetty tai on mahdollista hyödyntää käyttöliittymien testiautomaatiossa.

1.2 Tutkimuskysymykset

Tutkielman tarkoitusta tukemaan on muotoiltu seuraavat tutkimuskysymykset:

1. Mitkä ovat keskeisimmät ongelmat käyttöliittymien testiautomaatiossa?

2. Miten tekoälyä on mahdollista hyödyntää käyttöliittymien testiautomaatiossa?

Tutkielma saattaa nämä kaksi hieman irrallaan olevaa tutkimuskysymystä yhteen muotoilemalla vielä kolmannen tutkimuskysymyksen:

3. Onko käyttöliittymän testiautomaatiossa hyödynnetyissä tekoälyn sovellusalueista tarjota ratkaisuja siinä ilmenneisiin ongelmiin?

1.3 Menetelmät ja tiedonhaku

Tutkielma on toteutettu kirjallisuuskatsauksena, johon on etsitty aihepiiriä koskevia tieteellisiä artikkeleita ja tutkimuksia. Tiedonhakuun on käytetty useampaa tietokantaa, joista päälähteinä ovat toimineet IEEE Explorer-tietokanta ja Turun yliopiston kirjaston Volter-tietokanta. Mainittujen tutkimuskysymysten perusteella muotoiltiin kaksi erilaista hakulauseketta seuraavanlaisilla avainsanoilla:

1. Käyttöliittymän testiautomaation ongelmat:
(”GUI” OR ”*User Interface”) AND (”Test Automation” OR ”Testing”) AND (”Problem*” OR ”Cons” OR ”Difficulty”)

2. Tekoälyn hyödyntäminen käyttöliittymän testiautomaatiossa:
(”AI” OR “Machine learning” OR ”Artificial Intelligence”) AND (”GUI” OR ”*User Interface”) AND (”Test Automation” OR ”Testing”)

Haussa etsittiin kyseisiä avainsanoja artikkelien otsikoista ja tiivistelmistä. Lisäksi hakua rajattiin julkaisuvuoden perusteella 2010 eteenpäin. Käytetty aineisto koostuu haun ehdot täyttävistä artikkeleista, sekä muutamasta löydettyjen artikkelien sisällä viitatuista lähteistä.

1.4 Tutkielman rakenne

Tutkielma rakentuu neljästä luvusta. Luku yksi toimii johdantona, jossa käydään läpi tutkielman aihe ja tarkoitus, tutkimuskysymykset, sekä tutkielman menetelmä ja tiedonhaun toteutus. Toinen luku toimii teorialukuna, jonka tarkoituksena on avata tutkielmalle keskeiset käsitteet, kuten käyttöliittymä, ohjelmistotestaus ja testiautomaatio. Lisäksi luvussa käsitellään ensimmäistä tutkimuskysymystä käyttöliittymän testiautomaation ongelmista. Kolmannessa luvussa käsitellään toiseen tutkimuskysymykseen löydettyjä lähteitä tekoälyn sovelluksista käyttöliittymän testiautomaatiossa. Neljäs luku toimii yhteenveto kappaleena, jonka tarkoitus on tiivistää käsitellyt aihealueet ja pohtia aiheita yhdistävää kolmatta tutkimuskysymystä.

2 Käyttöliittymätestaus ja sen automatisointi

2.1 Käyttöliittymätestaus ja sen automatisointi

Graafiset käyttöliittymät (GUI) mahdollistavat helpon ja intuitiivisen vuorovaikutuksen käyttäjien ja monimutkaisten ohjelmistojen välillä. Nykypäivän ohjelmistotuotannolle on ominaista alustojen, laitteiden, työympäristöjen ja käyttäjien monimuotoisuus, mikä on johtanut ohjelmistojen ominaisuuksien monimutkaistumiseen ja laajentumiseen. [1] Ohjelmiston ominaisuuksien määrän kasvaessa kasvaa myös käyttöliittymiltä vaadittava laajuus ja monipuolisuus.

GUI:t edustavat ohjelmiston ulkokuorta. Käyttöliittymä on ensimmäinen asia, jonka käyttäjä näkee ja kokee ohjelmaa käyttäessään. Ohjelmistoyrityksen näkökulmasta onnistuneesti toteutettu ja toimiva käyttöliittymä on erittäin kriittinen osa tuotetta, sillä suurin osa käyttäjistä osaa todennäköisesti arvioida ohjelmiston laatua vain sen perusteella. Ohjelmiston menestyminen onkin laajasti sidoksissa sen käytettävyyteen [1] [3].

2.2 Käyttöliittymä ja sen testaaminen

Ohjelmistotestaus on merkittävä ja laaja osa-alue ohjelmistokehityksessä. Sen tarkoituksena on varmistua tuotetun ohjelmiston laadusta ja sen oikeanlaisesta toiminnasta. Prosessiin kuuluu tiivistetysti testien suunnitteleminen ja kehittäminen, niiden suorittaminen ja testien tulosten analysoiminen. Testauksen yleisenä tavoitteena on mahdollistaa järjestelmällinen tapa havaita ohjelmistossa esiintyvät virheet ja ongelmat sekä varmistaa ohjelmiston toimivan määritellyn mukaisesti.

Laadunvarmistus on itseasiassa yksi ohjelmistokehityksen laajimmista ellei laajin osa-alue. Sen on arvioitu kattavan $30 - 80\%$ tuotannon kustannuksista ja kuluttavan suurimman osan ajasta ohjelmiston tuotantovaiheessa [4]. Käyttöliittymien keskeinen merkitys ja laajuus tekevät niiden perusteellisesta testaamisesta välttämättömyyden.

Käyttöliittymätestauksessa pyritään löytämään virheitä ja epäjohdonmukaisuuksia käyttöliittymän toiminnassa [5]. Käyttöliittymässä käyttäjä kohdistaa toimintoja käyttöliittymäelementteihin ohjatakseen ohjelmiston toimintaa. Käyttöliittymäelementit ovat visuaalisia komponentteja, kuten painikkeita, valikoita, tekstikenttiä, ikoneita, ikkunoita ja muita visuaalisia tai toiminnallisia osia. Käyttöliittymätestauksessa pyritään testaamaan näiden elementtien toimintaa yksittäin ja yhdessä. Nykypäivän laajoissa käyttöliittymissä mahdollisten toimintasarjojen eli käyttöskenaarioiden määrä on valtava. Testaamisella ei voida varmistua ohjelmistojen täysin virheettömästä toiminnasta, sillä testaaminen ei kykene kattamaan kaikkia mahdollisia tilanteita, joita ohjelmaa käyttäessä voi tapahtua [6]. Ohjelmiston mahdollisten tilojen määrä hahmottuu hyvin juuri käyttöliittymässä.

2.3 Testiautomaatio

Testaaminen voidaan jakaa karkeasti manuaaliseen ja automaattiseen testaamiseen [7]. Manuaalisessa testauksessa testausta suorittaa ihminen. Testaaminen voi olla tutkivaa tai järjestelmällistä. Järjestelmällisessä testauksessa testaaja toistaa ennalta määrättyä tapahtumasarjaa eli testiskriptiä ja arvioi näin ohjelmiston toimivuutta. Tutkivassa testaamisessa testaaja käy läpi testattavaa käyttöliittymää vapaamuotoisemmin ilman varsinaista suunnitelmaa. Automaattisella testauksella tarkoitetaan testattavasta ohjelmistosta erillisiä ohjelmia ja työkaluja, jotka suorittavat ennalta määritetyt testiskriptit ja raportoivat niiden tuloksista.

Automaatiotestausta hyödynnetään erityisesti regressiotestauksessa, tarjoten järjestelmällisen tavan löytää aiemmin kirjoitetusta koodista löytyviä virheitä ja tavoitellen ajan säästöä testaajille. Näin testaajien työ saadaan kohdistettua testaamaan uusia toiminnallisuuksia ja toteuttamaan tutkivaa testausta. [7] Lisäksi testauksen automaatiota motivoi sen tuottama ajallinen eli kaupallinen etu: tietyillä ohjelmistotuotannon liiketoimintamalleilla menestyksen määrää nopeus, jolla tuote saadaan markkinoille [8]. Tästä esimerkkinä SaaS-liiketoimintamalli (Software as a service), jossa tarjottavia pilvipohjaisia ohjelmistoja ja palveluita pyritään jatkuvasti päivittämään ja korjaamaan. Nopeus, millä uusia ominaisuuksia ja korjauksia saadaan julkaistua, voi ratkaisevasti vaikuttaa yrityksen kilpailukykyyn markkinoilla. Testiautomaatio soveltuu mainiosti SaaS-yrityksiin, koska tuotanto keskittyy jo olemassa olevien tuotteiden laajentamiseen ja korjaamiseen. SaaS-yrityksille on yleistä noudattaa DevOps-toimintamallia (development and operations). Malli pyrkii automatisoimaan ohjelmistokehitykseen, testaamiseen ja yläpitoon (development) liittyviä toimenpiteitä, ja selkeyttämään kehityssykliä keskittämällä eri vaiheiden it-toimintoja (operations) yhteen työkalukokonaisuuteen.

Automaatiotestaus soveltuu parhaiten laajoihin ja ennustettaviin ohjelmistokokonaisuuksiin, kun taas manuaalinen testaaminen on parempi perusteellisempaan ja syvempään testaamiseen [6]. Esimerkiksi jokaisen käyttöliittymäelementin kertaalleen läpikäyminen on hyvin yksinkertainen, mutta manuaalisesti suoritettuna aikaa vievä tehtävä. Testiautomaatiota hyödynnetään paljon regressiotestauksessa. Regressiotestauksella tarkoitetaan ohjelmistoon kohdistuneiden muutosten jälkeen suoritettavaa testitapausten joukkoa, jotka varmistavat ohjelmiston kriittisimmät toiminnot. Regressiotestauksen testitapauksiin tulee lisätä varmistuminen aikaisemmin havaittujen virheiden korjaantumisesta, ja niiden tulee pysyä osana testitapauksia myös jatkossa [6].

2.4 Testiautomaation ongelmat

Käyttöliittymän automaatiotestauksen tekniikoita voidaan jakaa kolmeen sukupolveen. Varhaisin tekniikka hyödyntää tarkkoja hiiren koordinaatteja paikantamaan elementtien sijainnit. Toisen sukupolven tekniikassa käyttöliittymien automaatiotestausta suoritetaan käyttöliittymän elementtejä tunnistavilla kirjastoilla. Kyseisen tekniikan käyttö on yleisin ohjelmistotuotannon alalla. Kolmas ja uusin tekniikka hyödyntää paikantamiseen kuvantunnistusta. [8]

Käyttöliittymätestaaminen on korkean abstraktiotason testaamista, jossa keskitytään arvioimaan käyttöliittymän toiminnallisuutta ja suorituskykyä kokonaisvaltaisesti, eikä siihen sisälly ohjelmiston teknisen toteutuksen testaamista. Toisin sanoen se tarkastelee, kuinka hyvin käyttöliittymä toimii käyttäjän näkökulmasta eikä niinkään testaa ohjelmiston sisäisten toimintojen yksityiskohtia. Tämä on erityinen haaste testiautomaatiolle, jonka tekniikoista suurin osa soveltuu paremmin mata-

lamman abstraktiotason testaamiseen vrt. yksikkötestit. Testiautomaation kykyä suoriutua korkean tason testaamisesta on tämän takia kyseenalaistettu, sillä matalan tason tekniikoilla tuotetut korkean tason testit ovat taipuvaisia olemaan monimutkaisia ja vaikeita ylläpitää. [2]

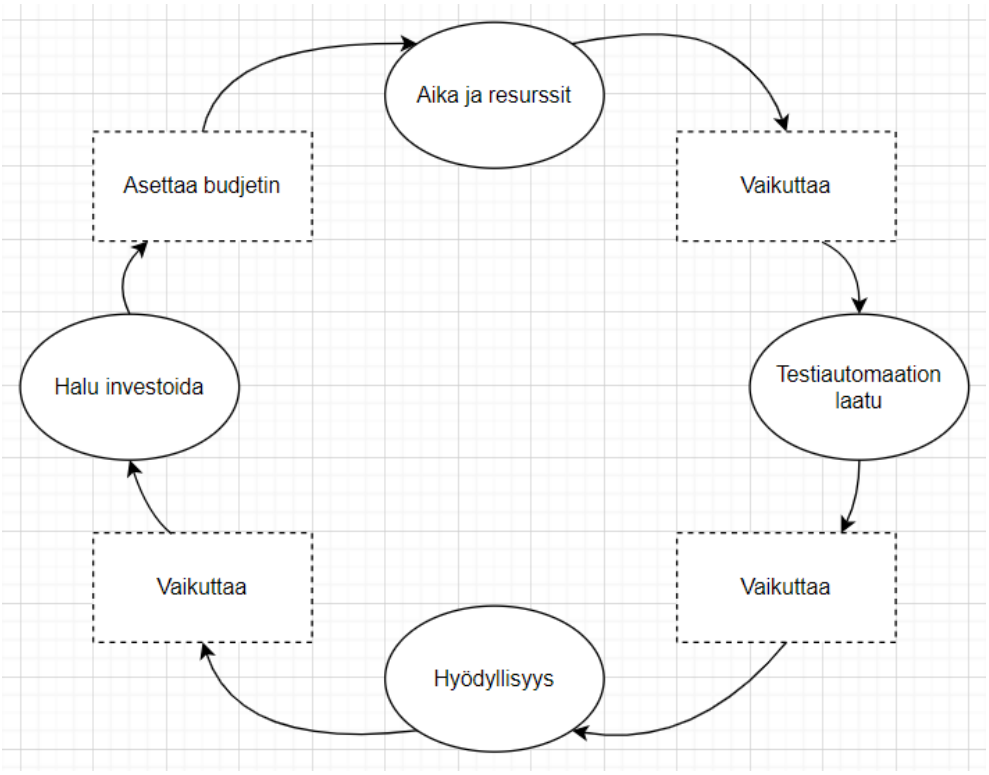
Täysin automatisoitua käyttöliittymätestausta on siis vielä vaikea toteuttaa. Nykypäivänä yleinen käyttöliittymän testausprosessi sisältää automaattista ja manuaalista testausta, jotka täydentävät toisiaan [8]. Manuaalista testausta käytetään varsinkin uusien ominaisuuksien testaamiseen, joille testiautomaatiota ei ole ehditty toteuttamaan. Uusien ominaisuuksien myötä tulee tehdä regressiotestausta, jonka jatkuva toistaminen manuaalisesti olisi erittäin raskasta ja virhealtista [2]. Täydelliseen testauksen automatisoimiseen ei halutakkaan pyrkiä, sillä tehokkaampi ja kattavampi testausprosessi sisältää molempia: manuaalista ja automaattista testausta [6].

Automaattisen käyttöliittymätestauksen yhtenä suurimpana ongelmana pidetään sen kehittämiseen ja ylläpidettävyyteen vaadittavaa manuaalista työtä [9]. Testiautomaatio on erittäin herkkä muutoksille esimerkiksi käyttöliittymässä, koodissa tai alustassa [2]. Tyypillinen ongelma ilmenee esimerkiksi selainversion päivittyessä, jolloin automaatioskriptissä käytetyt tunnisteet eivät ole välttämättä enää havaittavissa testattavassa käyttöliittymässä. Käyttöliittymän mahdolliset epäsäännölliset elementtitunnisteet ovat myös ongelma testiautomaatiolle. Kyseiseen voidaan törmätä, kun testiautomaatio käyttöönotetaan vasta käyttöliittymän kehityksen jälkeen.

Ensimmäisen ja toisen sukupolven käyttöliittymän testiautomaatiotekniikoiden avuksi on käytetty tallennus- ja toistotyökaluja. Nämä työkalut tallentavat manuaalisesti suoritettuja testejä tallentamalla elementit ja niihin kohdistetut toiminnot skrip-

tiksi. Nämä työkalut ovat nopeuttaneet testiautomaation kehitystä, mutta ne eivät poista sen herkkyyttä muutoksille [2]. Nykypäivän käyttöliittymien testiautomaatiotekniikat ovat siis hyvin virhealttiita ja vaativat jatkuvaa kehitystä ja ylläpitoa.

Testiautomaation käyttöönotto ja ylläpito ohjelmistoyrityksessä voi olla huomattava investointi. Todellisia kustannuksia on vaikeaa arvioida julkaistun tiedon puutteen vuoksi. On hyvin mahdollista, että tuotettu testiautomaatiokokonaisuus on laajempi ja monimutkaisempi kuin itse testattava ohjelmisto. Testiautomaation kehittäminen voi olla siis hyvin työläs prosessi. Sen on todettu kärsivän usein teknisestä velasta, jolla tarkoitetaan ohjelmistojen epäoptimaalista, huonolaatuista tai haasteellista teknistä ratkaisua tai käytäntöä, joka voi haitata järjestelmän pitkäaikaista tehokkuutta ja ylläpidettävyyttä. Teknistä velkaa syntyy, kun järjestelmän kehitykseen käytettäviä resursseja keskitetään muualle esimerkiksi aikataulullisista syistä. [4] Testiautomaation tai muun toissijaisen järjestelmän jäädessä pahasti jälkeen tavoitteista siihen suhtautuminen voi muuttua hyvin välinpitämättömäksi. Huonosti toteutettu testiautomaatio saattaakin vain lisätä kustannuksia ja vaivaa, ja se ei välttämättä ole manuaalista testausta tehokkaampaa [10]. Tilanteessa, jossa tuotettua testiautomaatiota ei koeta hyödyllisenä, on todennäköistä, että halu investoida testiautomaation kehitykseen laskee. Investoinnin laskiessa testiautomaation laatu kärsii ja jälleen halu investoida kehitykseen laskee. (Kuva 2.1) [8]



Kuva 2.1: Testiautomaatiokehityksen palautesilmukka ohjelmistoyrityksissä. K. Wiklund, S. Eldh, D. Sundmark, K. Lunqvist 2017 [8]

Testiautomaation kehitys ja käyttöönotto on erittäin haastava ja monimutkainen tehtävä. Testiautomaation kehitykseen osallistuneiden henkilöiden taitotasolla, työskentelytavoilla, motivaatiolla ja tavoitteilla on selkeä vaikutus tuotetun testiautomaation laatuun. Testiautomaatio on riippuvainen myös sitä ennen olemassa olevasta vakaasta ja järjestelmällisestä testausprosessista. Testauksen automatisoinnista tulee erittäin haastavaa jos testausprosessi on epämuodollinen ja laadunvarmistukseen on yrityksen sisällä välinpitämätön tai jopa negatiivinen asenne. [8] Mankeforsin ja Torkarin (2003) teettämässä kyselytutkimuksessa 60% kehittäjistä vastasi laadunvarmistuksen olevan ensimmäinen laiminlyötävä vaihe kun projektin resurssit ovat riittämättömät. [11]

3 Tekoälyn hyödyntäminen käyttöliittymien testiautomaatiossa

Tekoälyn käyttäminen ohjelmistokehityksessä on kasvava trendi, ja sillä on lukuisia sovellusmahdollisuuksia kehityksen eri vaiheissa. Erityinen potentiaali, sillä on juuri testauksen osa-alueella, jonka todettiin kattavan merkittävän määrän tuotannon kustannuksista [4]. Tämän luvun tarkoituksena on tutustua, miten tekoälyä on hyödynnetty testiautomaatiossa, ja onko sillä tarjota ratkaisuja siinä ilmenneisiin ongelmiin. Tämän tutkielman erityisenä mielenkiinnon kohteena on tekoälyn hyödyntäminen käyttöliittymien testiautomaatiossa.

3.1 Kuvantunnistusta käyttävä testiautomaatio

Aiemmin esiteltiin käyttöliittymien testiautomaatiotekniikoita, joista kolmas käyttää hyväkseen kuvantunnistusta. Kuvantunnistus on tekoälyn sovellusala, jossa tietokonejärjestelmiä koulutetaan havaitsemaan, analysoimaan ja tulkitsemaan visuaalista informaatiota, erityisesti kuvia ja videoita. Kyseinen sovellusala on suunniteltu matkimaan ihmisen visuaalista havaintokykyä ja kykyä tunnistaa objekteja, piirteitä ja malleja visuaalisen datan perusteella.

Eskonen, Kahles ja Reijonen (2020) ovat arvioineet syvään vahvistusoppimiseen (DRL) pohjautuvien algoritmien käyttöä käyttöliittymätestauksessa. Kyseiset algoritmit hyödyntävät kuvantunnistusta testattavasta käyttöliittymästä otettuihin kuvankaappauksiin. Kuvankaappaukset edustavat algoritmille käyttöliittymän sen hetkistä tilaa, jonka perusteella se päättää siihen kohdistettavasta seuraavasta toiminnosta. [5]

Algoritmi on koulutettu matkimaan ihmisten käyttäytymistä käyttöliittymässä. Kyseisessä algoritmissa ihmismäisyyttä lisätään kannustamalla agenttia olemaan tutkivampi käyttäessään käyttöliittymää. Agentilla tarkoitetaan älykästä toimijaa, joka on vuorovaikutuksessa ympäristönsä kanssa. Tässä tapauksessa ympäristönä toimii testattava käyttöliittymä. Vahvistusoppimisen (RL) periaatteena toimii agentin palkitseminen tai rankaiseminen ympäristöönsä kohdistamien toimintojensa perusteella. Menetelmässä agenttia palkitaan sen löytäessä uusia näkymiä eli käyttöliittymän tiloja. Tämä johtaa siihen, että agentti pyrkii löytämään mahdollisimman paljon uniikkeja näkymiä käyttöliittymästä maksimoidakseen palkkionsa. [5]

Viime luvussa käsittelimme automaattisen- ja manuaalisen testaamisen eroja. Todettiin, että manuaalinen testaaminen soveltuu paremmin tutkivaan testaamiseen, jossa pyritään löytämään täysin uusia virheitä. Automaattista testausta käytetään yleisimmin regressiotestauksessa, jossa suoritettavat testitapaukset pysyvät samoina varmistuen vanhojen virheiden korjauksista. Toteutettu ratkaisu mahdollistaa testiautomaation käytön tutkivaan testaamiseen. Algoritmin tutkivuutta verrattiin kolmeen manuaaliseen testaajaan, joista yhdelle käyttöliittymä oli ennestään tuttu, ja toisille täysin tuntematon. Tutkivuuden määrää mitattiin uniikkien URL:ien määrällä, joissa testaajat vierailivat tehtyään 100 toimintoa käyttöliittymässä. Rajoitetulla toimintomäärällä algoritmi kykeni löytämään reilusti enemmän käyttöliittymän tilo-

ja, kuin testaajat, joille käyttöliittymä oli tuntematon. Testaaja, jolle käyttöliittymä oli tuttu, ylitti algoritmin uniikit URL:ät kahdella. Huomioon otettavaa on, että algoritmi suoritti 100 toimintoa 10 sekunnissa, kun manuaalisilta testaajilta siihen kului noin 5 minuuttia. [5]

Visuaalisesti tunnistava kuvantunnistus eroaa vanhempien tekniikoiden käyttämästä manuaalisesti kirjoitetusta skriptikoodista, jossa tunnistus tai paikantaminen tapahtuu koodipohjaisesti elementtien tunnisteista tai koordinaateista. Kuvantunnistuksella toimivan testiautomaation voisi näin olettaa kestävän paremmin testattavan käyttöliittymän muutoksia. Kuvantunnistus tekee mahdolliseksi testiautomaation käyttää käyttäjille todenmukaista vuorovaikutustapaa käyttöliittymän kanssa.

Tutkimuksia pitkäaikaisesta kuvantunnistusta hyödyntävän testiautomaation käytöstä ohjelmistoyrityksissä ei ole paljon, sillä harva yritys on uskaltanut sijoittaa pitkäaikaisesti kyseiseen tekniikkaan. Spotify on yksi harvasta suuresta ohjelmistoyrityksestä, jolla on monen vuoden käyttökokemus kuvantunnistusta hyödyntävästä testiautomaatiosta. Kyseisestä tekniikasta kuitenkin luovuttiin. Syyt tekniikan hylkäämiseen osoittautuivat tutuiksi testiautomaation ongelmiksi: paikoittainen dynaamisuuden puute ja vaadittava ylläpito. Tekniikan erikoisuutena oli sen ylläpidon kohdistuneen suurimmaksi osaksi automaation käyttämiin kuviin. [8]

3.2 Automaattinen testigenerointi

Yksi lähteissä laajimmin esiintyvä käyttökohde tekoälylle on automaattinen testigenerointi. Aiheen ympärille tehdyn laajan kirjallisuuskatsauksen mukaan harvempi lähde kuitenkaan selventää, miten automaattinen testien generoiminen on toteutet-

tu [12].

Viime vuosikymmenen alkupuolella tutkittavat tekniikat ovat sisältäneet käyttöliittymän ja testien generoimista mallin pohjalta. Automaatiinen testien uudelleen generoituminen käyttöliittymän muuttuessa on ollut tavoiteltava ominaisuus, jonka saavuttamiseen on kaavailtu geneettisten algoritmien hyödyntämistä. [13]

Automaattisen testi generoimisen ongelma on ollut pienempienkin käyttöliittymien todettu laajuus, eli mahdollisten tilojen määrä. Testien generoiminen onnistuu, mutta niitä generoidaan liikaa. Optimaaliseen testien kattavuuteen ja määrään on pyritty pääsemään hyödyntämällä tekoälyä niiden suunnittelussa generoinnin yhteydessä. [13]

Esimerkki tekoälyn hyödyntämisestä testien suunnittelemisessa on kahdella hybridi geneettisiin algoritmeihin (HGA) pohjautuvilla agenteilla toimiva testien optimointikehys, jossa toinen agenteista (Intelligent Search Agent (ISA)) keskittyy testijonon optimointiin ja toinen (Intelligent Test Case Optimizer Agent (ITOA)) itse testitapauksien optimointiin. Optimointikehyksen syötteenä toimii testattavan ohjelmiston lähdekoodi, josta johdetaan ohjelmistoa kuvaava UML-kaavio, ja lopulta agenttien optimoimat generoidut testit. [14]

Eräs mielenkiintoinen ratkaisu automaattiseen testien generoimiseen on ollut NLP:n (Natural Language Processing) käyttäminen. Testsigma on pilvipohjainen testiautomaatiotyökalu, joka muodostaa kirjoitetusta testin kuvailusta generoidun testin. Testin kuvailu jaetaan lauseittain segmentteihin. Segmentit jaetaan edelleen blokkeihin, jotka kuvaavat toimintoja, toiminnan kohteita ja syötedataa. Kun testien kuvailemiseen käytetään rajattua ja tarkkaa kieltä, sen analysoiminen on varsin helppoa.

[12]

Osalle käyttöliittymästä on ominaista näkymät, joissa edetäkseen vaaditaan syötekenttiin kohdistettavalta syötteeltä tarkkaa muotoa. Tämä on ongelma testiautomaatiolle, jolle on yleistä sen syöttämän tekstin satunnainen generointi. Toki kelvollisista syöte vaihtoehdoista voidaan manuaalisesti laatia lista testiautomaatiolle, mutta siinä tapauksessa testikattavuus kärsii. Huomattavasti laajemman ja kelvollisen syötevalikoiman saavuttamiseksi on kaavailtu juuri suurien kielimallien käyttöä. Qtypist on kielimalli, joka pyrkii generoimaan syötteen käyttöliittymän näkymän tarjoaman kontekstin perusteella. Kielimallille räätälöidään komentokehote käyttöliittymän kontekstin perusteella, ja tästä saatua tulostetta käytetään syötteenä testattavassa käyttöliittymässä. [15]

Gao et. al. (2022) esittivät tutkimuksessaan kokonaisuuden, jossa kuvantunnistamisalgoritmia käytettiin luomaan diagrammi, joka kartoittaa testattavaa käyttöliittymää[16]. Luotuja diagrammeja hyödynnettiin automaattiseen testiskenaarioiden luontiin. Automaattisessa testiskenaarioiden luonnissa ei itsessään käytetty tekoälyä vaan perinteistä syvyyshakua, mutta syötteenä toimivat diagrammit olivat kuvantunnistusta hyödyntäneen algoritmin luomia. Diagrammit sisälsivät tietoa käyttöliittymän- näkymien ja elementtien yhteydestä toisiinsa. Diagrammien muodostama visuaalinen representaatio vaikuttaa varsin hyödylliseltä testiskenaarioiden kattavuuden arviointiin, ja testausprosessin dokumentointiin. Tämä saa pohtimaan mahdollisuutta, jossa algoritmin tuottamia diagrammeja voitaisiin edelleen käyttää syötteenä suurille kielimalleille, jotka generoisivat testit kartoitettuun käyttöliittymään.

3.3 Itsestään korjaantuvat testiskriptit

Useamman lähteen perusteella, tekoälyn toivotaan ennen kaikkea tuovan ratkaisuja käyttöliittymän testiautomaation jatkuvaan ylläpito-ongelmaan. Ratkaisuksi on kaavailtu tekoälyä hyödyntäviä itsekorjaantumismekanismeja, joiden on tarkoitus tunnistaa ja korjata virheitä ennakoivasti. Kyseistä on pääasiassa toteutettu itsekorjautuvilla testiskripteillä tai älykkäillä käyttöliittymänelementtipaikantimilla. Molemmissa ratkaisuissa pyritään pitämään testit toimivina käyttöliittymän muuttuessa. [12]

Suurin osa käyttöliittymän testiautomaation virheistä liittyy skriptissä määriteltyyn elementtiin, jota testiohjelma ei pysty ajon aikana havaitsemaan. Rikkoutunut itsekorjautuva testiskripti pyrkii korjaamaan itsensä automaattisella korjaustoimenpiteellä [12]. Mabl on eräs testauskehys, joka pyrkii paikantamaan kohde-elementin käyttämällä samankaltaisia elementtejä viitepisteinä, ja paikantamaan kohteen suhteessa näihin. Tämä ratkaisu auttaa paikantamaan kohde-elementin joustavasti ja vakaasti, vaikka sen sijainti muuttuisi. Paikantimen rikkoutuessa kyseinen paikannin ilmoitetaan testaajalle, ja Mabl pystyy antamaan korjausehdotuksia. Valittu korjausehdotus toteutetaan kaikkiin testeihin, jotka sisältävät rikkoutuneen elementin.

Älykkäiden paikantimien toiminta perustuu dynaamiseen tapaan paikantaa elementti. Älykkäällä paikantimella on useita attribuutteja, joiden perusteella se pystytään tunnistamaan. Attribuutit voivat olla esimerkiksi elementin sijainti, teksti arvo, tyyli tai tunniste. Älykkäät tunnisteet käyttävät ennustavaa mallia arvioidakseen, mikä elementin attribuutti on luotettavin tapa paikantaa kyseinen attribuutti olemassa olevassa näkymässä. Käyttöliittymän muuttuessa, ennusteet luotettavimmasta tavasta paikantaa elementti saattaa muuttua.

3.4 Yhteenveto käsiteltyjen sovellusalueiden käyttökohteista

Tekoälyn soveltaminen käyttöliittymän testiautomaatiossa tarjoaa monipuolisia mahdollisuuksia testien suunnitteluun, kehittämiseen, suorittamiseen ja ylläpitoon. Eri tekoälyn sovellusalueet, kuten geneettiset algoritmit, kuvantunnistus, luonnollisen kielen käsittely (NLP) ja kielimallit (LLM), sekä tilastolliset ennustemallit, tarjoavat erilaisia lähestymistapoja testiautomaation parantamiseen. Geneettiset algoritmit voivat auttaa testien generoinnissa ja optimoinnissa monimutkaisissa käyttöliittymissä, kun taas kuvantunnistusta voidaan käyttää testien suorittamisessa ja jopa eksploratiivisessa testaamisessa[5][14]. NLP ja LLM voivat auttaa testien suunnittelussa ja ymmärtämisessä, kun taas tilastolliset ennustemallit auttavat havaitsemaan ja korjaamaan mahdollisia herkkiä kohtia suoritettavassa testiautomaatiossa [12]. Lähteissä käsitellyt tekoälyn sovellusalueet voidaan jakaa käyttökohteidensa perusteella seuraavasti. (Kuva 3.1)

Testien suunnittelu/kehittäminen	Testien suorittaminen	Testien ylläpito
Geneettiset algoritmit	Kuvantunnistus	Geneettiset algoritmit
Kuvantunnistus	LLM	Stastiikka ja ennustavat mallit
NLP / LLM		

Kuva 3.1 Yhteenveto käsiteltyjen sovellusalueiden käyttökohteista

4 Yhteenveto

Tutkielman tarkoituksena oli perehtyä tekoälyn hyödyntämiseen käyttöliittymien testiautomaatiossa. Tutkielma pyrkii vastaamaan tutkimuskysymyksiin: ”Mitkä ovat keskeisimmät ongelmat käyttöliittymien automaatiotestauksessa?” ja ” Miten tekoälyä on mahdollista hyödyntää käyttöliittymien automaatiotestauksessa?”

Aloitimme määrittelemällä termejä, kuten käyttöliittymä, ohjelmistotestaus ja testiautomaatio. Termien määrittelyn yhteydessä perehdyimme käyttöliittymien testiautomaation, ja testiautomaation ongelmiin yleisesti. Tutustuimme termien määrittelyn jälkeen lähteistä löydettyihin tekoälyn sovellusalueisiin, joita voidaan hyödyntää käyttöliittymän testiautomaatiossa. Tämän luvun tarkoituksena on tiivistää lukujen kaksi ja kolme keskeisin sisältö. Lisäksi tutkielma haluaa pohtia, onko tekoälystä mahdollisesti apua todettujen ongelmien ratkaisemiseksi.

4.1 Korkean abstraktiotason testaaminen

Nykyisten testiautomaatiotekniikoiden todettiin soveltuvan paremmin matalamman abstraktiotason testaamiseen. Testiautomaation kehitys on peräisin ohjelmistojen sisäisten teknisten toteutuksien testaamisesta, kun käyttöliittymätestaamisen tulisi tähdätä käytettävyyteen ja käyttäjäkokemuksen testaamiseen. Kyseisen toteuttami-

nen on osoittautunut hankalaksi matalamman abstraktiotason testaamiseen tarkoitetuilla tekniikoilla, jotka ovat johtaneet monimutkaisiin ja herkkiin testiautomaatiokokonaisuuksiin. [2]

Testiautomaation varsinainen tehtävä käyttöliittymätestauksessa on toistaiseksi käyttöliittymän toimintaa rikkovien virheiden löytäminen. Käyttäjäkokemuksen testaaminen vaikuttaa olevan liian abstraktia automatisoida, sillä siihen vaaditaan ihmismäistä arviointikykyä. Toimiva ja virheetön käyttöliittymä on vain osa käyttäjäkokemusta. Edellisessä luvussa käsitelty kuvantunnistus on kuitenkin askel kohti ihmismäisempää testiautomaatiota. Kuvantunnistuksen myötä testiautomaatiolla on kyky havainnoida testattavaa käyttöliittymää realistista käyttäjäkokemusta muistuttavalla tavalla. Viime luvussa käsitellyssä kuvantunnistusmenetelmässä ihmismäisyyttä pyrittiin lisäämään kannustamalla agenttia olemaan tutkivampi [5]. Tämä saa pohtimaan, millä muilla tavoilla kuvantunnistusta hyödyntävän testiautomaation ihmismäisyyttä voisi lisätä.

4.2 Testiautomaation kehittämisen ja ylläpidon ongelmat

Keskeisin löydetty ongelma käyttöliittymien testiautomaatiossa kohdistuu testiautomaatiotekniikoiden dynaamisuuden puutteeseen. Yleisimmin käytetyt elementti tunnisteita hyödyntävät testiautomaatiotekniikat ovat erittäin herkkiä testattavan käyttöliittymän muutoksille, joka johtaa testiautomaation jatkuvaan ylläpito tarpeeseen.

Kolmannessa luvussa laajasti käsitelty kuvantunnistusta hyödyntävä testiautomaatiotekniikka tarjoaa vaihtoehtoisen tavan tunnistaa käyttöliittymäelementtejä. Lähdekoodista löytyvien elementtitunnisteiden sijaan tunnistamiseen käytettään elementtien visuaalista representaatiota. Vaihtoehtoinen tapa ei kuitenkaan vaikuta tarjoavan ratkaisua dynaamisuuden lisäämiselle. Kolmannessa luvussa mainittiin Spotifyn hylkäämästä kuvantunnistusta hyödyntäneestä testiautomaatiotekniikasta, jonka todettiin myös olevan herkkä käyttöliittymän muutoksille [8]. Kuvantunnistukseen pohjautuva testiautomaatio kärsii samasta ongelmasta, kuin elementtitunnisteiseen pohjautuva testiautomaatio, mutta tunnistamattomuutta aiheuttaa muuttuneen tunnisteen sijasta elementin muuttunut ulkonäkö.

Käsiteltyjen lähteiden pohjalta testiautomaation ylläpito ongelman lupaavimmaksi ratkaisun tarjoajaksi osoittautuu itsestään korjaantuvat testiskriptit. Älykkäät paikantimet, ja ympäröivien elementtien hyödyntäminen elementtien paikantamiseen vaikuttavat erittäin lupaavilta testiautomaation dynaamisuuden lisäämiseen. Kyseisten tekniikoiden käytöstä ei löytynyt varsinaista tutkimusta, mutta tekniikkaa hyödyntäviä testiautomaatiokehyksiä on löydettävissä markkinoilta.

Toisessa luvussa todettiin myös testiautomaation kehityksen mahdollisesta laajuudesta ja haasteellisuudesta. Niin kuin varsinaisen ohjelmistokoodin tuotannossa, testiautomaation skriptien tuottamisessa on alettu hyödyntää automaattista koodin generoimista. Automaattiseen testi generoimisen toteuttamiseen on kaavailtu useamman tekoälyn sovellusalueen hyödyntämistä. Näistä merkittävimpänä voidaan pitää suuria kielimalleja. Suurten kielimallien suosio on kasvanut huomattavasti niiden osoitettua lupaavaa suoriutumista erilaisista tietotyön tehtävistä. Testiautomaatiolla toteutetut testitapaukset ovat suurimmaksi osaksi luonteeltaan hyvin yksinkertaisia, mutta niiden määrä saattaa olla hyvin suuri. Tämä tekee automaattisten

testitapauksen kehityksestä erittäin hyvän tehtävän suurille kielimalleille, joita on hyödynnetty juuri yksinkertaisissa, mutta aikaa vievissä tehtävissä.

Ongelmissa tuotiin esiin myös testiautomaation kustannuksien suhde siitä koettuun hyötyyn. Käsitellyt lähteet tekoälyn sovelluksista käyttöliittymän testiautomaation kehitykseen ja ylläpitoon pyrkivät automatisoimaan kyseisiä prosesseja. Täysin tai osittain automatisoidut testiautomaation kehitys- ja ylläpitoprosessit tekisivät siihen investoimisesta houkuttavampaa. Ohjelmistoilta vaadittava laajuus tulee kasvamaan entisestään tulevaisuudessa, jolloin ohjelmistotuotannon tehokkuuteen halutaan panostaa entistä enemmän.

4.3 Tutkimuskysymyksiin vastaaminen

1. Mitkä ovat keskeisimmät ongelmat käyttöliittymien testiautomaatiossa?

Käyttöliittymien testiautomaation keskeisimmät haasteet liittyvät testien herkkyyteen muutoksille, käyttöliittymien monimutkaisuuteen ja laajaan ylläpidon tarpeeseen. Automaation kehittäminen ja ylläpito vaativat merkittävästi resursseja, mikä voi johtaa tekniseen velkaan ja heikentää siten laadunvarmistusprosessia.

2. Miten tekoälyä on mahdollista hyödyntää käyttöliittymien testiautomaatiossa?

Kuvantunnistus, automaattinen testigenerointi ja itsestään korjaantuvat testiskriptit ovat eräitä tutkielmassa käsiteltyjä tapoja hyödyntää tekoälyä käyttöliittymien testiautomaatiossa. Kuvantunnistus tarjoaa uuden ja potentiaalisesti kestävämmän lähestymistavan käyttöliittymän ohjaukseen testiautomaatiossa, testigenerointi luo

ja optimoi testitapauksia automaattisesti, ja itsestään korjaantuvat testiskriptit havaitsevat ja korjaavat virheitä ennakoivasti.

3. Onko käyttöliittymän testiautomaatiossa hyödynnetyissä tekoälyn sovellusalueista tarjota ratkaisuja siinä ilmenneisiin ongelmiin?

Tekoäly tarjoaa uusia lähestymistapoja käyttöliittymien testiautomaation toteuttamiseen. Esimerkiksi kuvantunnistuksella on potentiaalia mahdollistaa joustavampia testiautomaatioratkaisuja, jotka ovat vähemmän herkkiä käyttöliittymän muutoksille. Automaattinen testien generointi puolestaan voi vähentää manuaalisen työn tarvetta testien ylläpidossa ja päivittämisessä. Lisäksi itsestään korjaantuvat testiskriptit voivat parantaa testiautomaation luotettavuutta ja ylläpidettävyyttä tunnistamalla ja korjaamalla virheitä automaattisesti.

Tekoäly ei ratkaise testiautomaation ongelmia kokonaisuudessaan. Tarpeeksi monimutkaisten käyttöliittymien testaaminen vaatii edelleen manuaalista tarkastelua ja arviointia. Lisäksi tekoälyn soveltaminen voi tuoda mukanaan omat haasteensa.Tekoälyä hyödyntävät ratkaisut voivat vaatia merkittävää asiantuntemusta ja resursseja niiden kehittämiseen ja käyttöönottoon, mikä voi rajoittaa niiden käyttöä kaikissa testiautomaation konteksteissa.

4.4 Tekoälyn kasvava läsnäolo ohjelmistotuotannossa

Tekoäly tarjoaa uusia lähestymistapoja käyttöliittymän testiautomaation toteuttamiseen, sekä muihin ohjelmistotuotannon osa-alueisiin. Tekoälyn sovellusaleissa

on selkää potentiaalia tehostaa ja mullistaa ohjelmistokehitystä kokonaisuudessaan. Varsinkin suurissa kielimalleissa saavutettu kehitys on tehostanut kyseistä sovellusaluetta käyttävien työkalujen tuotantoa ja on vain ajan kysymys kunnes ne asettavat uusia standardeja ohjelmistotuotannon alalle.

Yhteenvetona voidaan todeta, että tekoäly tarjoaa valtavia mahdollisuuksia käyttöliittymän testiautomaatiossa, mutta se ei poista kaikkia haasteita. Uudet mahdollisuudet asettavat uusia kysymyksiä, jotka vaativat huolellista pohdintaa ja tutkimusta. Tulevaisuudessa on odotettavissa jatkuvaa kehitystä, joka auttaa hyödyntämään tekoälyn potentiaalia testiautomaatiossa entistä paremmin.

Lähdeluettelo

[1] J. Ruiz, E. Serral ja M. Snoeck, ”Unifying Functional User Interface Design Principles”, 2020.

[2] E. Alégroth, R. Feldt ja L. Ryrholm, ”Visual GUI testing in practice: challenges, problems and limitations”, 2014.

[3] A. Molina, W. Giraldo, J. Gallardo, M. Redondo, M. Ortega ja G. García, ”CIAT-GUI: A MDE-compliant environment for developing Graphical User Interfaces of information systems”, 2012.

[4] K. Wiklund, S. Eldh, D. Sundmark ja K. Lunqvist, ”Impediments for software test automation: A systematic literature review”, 2017.

[5] J. Eskonen, J. Kahles ja J. Reijonen, ”Automating GUI Testing with Image-Based Deep Reinforcement Learning”, 2020.

[6] B. Meyer, ”Seven Principles of Software Testing”, 2008.

[7] O. Taipale, J. Kasurinen ja K. Karhu, ”Trade-off between automated and manual software testing”, 2011.

[8] E. Alégroth ja R. Feldt, ”On the long-term use of visual gui testing in industrial practice: a case study”, 2017.

[9] T. Vos, P. Aho, F. Ricos, O. Rodriguez-Valdes ja A. Mulders, ”TESTAR – scriptless testing through graphical user interface”, 2021.

[10] Y. Amannejad, ”A Search-Based Approach for Cost-Effective Software Test Automation Decision Support and an Industrial Case Study”, teoksessa *Proc. Int’l Workshop Regression Testing*, 2014, s. 302–311.

[11] R. Torkar ja S. Mankefors, ”A survey on testing and reuse”, teoksessa *IEEE international conference on software-science, technology and engineering*, 2003.

[12] F. Ricca, A. Marchetto ja A. Stocco, ”AI-based Test Automation: A Grey Literature Analysis”, 2021.

[13] A. Rauf ja M. Alanazi, ”Using artificial intelligence to automatically test GUI”, 2014.

[14] D. Mala ja V. Mohan, ”IntelligenTester –Test Sequence Optimization framework using Multi-Agents”, 2008.

[15] Z. Liu, C. Chen, J. Wang et al., ”Fill in the Blank: Context-aware Automated Text Input Generation for Mobile GUI Testing”, 2023.

[16] J. Gao, S. Li, C. Tao et al., ”An Approach to GUI Test Scenario Generation Using Machine Learning”, 2022.