Tampereen yliopisto

Sebastian Jokela

OHJELMISTOTESTIEN GENEROINTI KONEOPPIMISEN AVULLA

Informaatioteknologian ja viestinnän tiedekunta
Kandidaattitutkielma
Toukokuu 2021

TIIVISTELMÄ

Sebastian Jokela: Ohjelmistotestien generointi koneoppimisen avulla
Kandidaattitutkielma
Tampereen yliopisto
Tietojenkäsittelytieteiden tutkinto-ohjelma
Toukokuu 2021

Ohjelmistotestaus on ohjelmistokehityksen tärkeä osa, jolla varmistetaan kehitetyn sovelluksen virheetön toiminta. Ohjelmistotestien luominen käsin on kuitenkin työläs prosessi, joten sitä on pyritty helpottamaan erilaisilla testejä automaattisesti generoivilla sovelluksilla. Tämän tutkimuksen tarkoitus on selvittää, millaisia koneoppimisalgoritmeja testien generointiin on sovellettu, mitä vahvuuksia ja heikkouksia koneoppimisavusteisella testien generoinnilla on ja miten se toimii käytännössä. Käytännön implementaatioita tarkastellaan EvoSuite-ohjelmiston kautta, sen edistyneisyyden ja siitä saatavilla olevan laajan tutkimuksen vuoksi.

Testien generointiin on sovellettu monenlaisia koneoppimisalgoritmeja, joista laajimmassa käytössä ovat erilaiset evolutiiviset algoritmit. Niiden etuna on se, että niiden käyttäminen ei juurikaan vaadi työtä käyttäjältä, sillä niihin ei tarvitse manuaalisesti sisällyttää tietoa testattavasta sovelluksesta.

Koneoppiminen saavuttaa testien generoinnissa huomattavasti parempia tuloksia kuin perinteiset satunnaishakuun perustuvat mallit. Sen avulla generoitujen testien kattavuus on suurempi, vaikka testiohjelmat ovat pienempiä ja ne generoidaan yhtä nopeasti, tai jopa nopeammin.

Vaikka koneoppimisen avulla luodut testit nopeuttavat testaamista huomattavasti ja mahdollistavat virheiden löytämisen, niiden käyttöä rajoittaa kuitenkin muutama tekijä. Testejä on vaikea ymmärtää, joissain tilanteissa ne epäonnistuvat vaikka ohjelma toimisi virheettömästi, ja ohjelmoijat eivät osaa käyttää testejä generoivia ohjelmistoja optimaalisesti.

Tällä hetkellä testien generointi automaattisesti soveltuu lähinnä ohjelmistotestaamiseen, joka suoritetaan samanaikaisesti ohjelmiston luomisen kanssa. Hyviä tuloksia on saavutettu myös mikroprosessorien testaamisessa, joskin silloin käyttäjältä vaaditaan enemmän vaivannäköä kuin Java-kielisten ohjelmistojen automaattisessa testaamisessa. Korjaamalla erilaisten koneoppimista käyttävien testiengenerointityökalujen vikoja, niillä voitaisiin säästää huomattavasti aikaa ja vaivaa ohjelmistoa testatessa.

Avainsanat: ohjelmistotestaus, ohjelmistotestien generointi, ohjelmistotestauksen automatisointi, koneoppiminen, evolutiiviset algoritmit

Sisällysluettelo

1 Johdanto

Nyky-yhteiskunta on täynnä ohjelmistoja, ja vika ohjelman toiminnassa voi käydä hyvin kalliiksi yrityksille ja ihmisille, jotka tarvitsevat sitä. Tämän vuoksi ohjelmistojen virheettömästä toiminnasta on tärkeä varmistua. Ohjelmistojen laajuuden vuoksi ohjelmakoodin läpikäyminen ihmisen toimesta ei olisi kustannustehokasta, ja inhimillisen virheen riski olisi suuri. Tämän vuoksi kehitetään ohjelmistotestejä, jotka voivat testata kaikkea yhdestä funktiosta aina kokonaiseen ohjelmaan. Testien tarkoitus riippuu siitä, milloin niitä kirjoitetaan. Ohjelmistotestejä voidaan kirjoittaa jo ennen kuin ohjelmakoodi kirjoitetaan, jolloin niiden tarkoitus on varmistaa tuottavan ohjelmakoodin laatu ja seurata kehityksen etenemistä. Niitä voidaan kirjoittaa myös samaan aikaan kuin ohjelmakoodia kirjoitetaan, jotta vastakirjoitetun ohjelmakoodin oikeellisuus varmistuu, ja jotta myöhemmin koodia muokatessa havaitaan mahdolliset virheet. Ne voidaan tehdä myös ohjelmiston valmistuttua, jotta havaitaan mahdollisimman monta vikaa sen toiminnassa, ja varmistamaan että mahdolliset muokkauksien aiheuttamat viat havaitaan.

Ohjelmistojen testaaminen on kuitenkin hyvin työläs prosessi. Pahimmillaan jopa 80 % projektin aikana kirjoitetusta koodista saattaa olla omistettu sen testaamiselle, ja sen lähdekoodiriippuvuuden vuoksi isoa osaa siitä ei voida käyttää uudelleen (Ioannides & Eder, 2012). Koska testaaminen on työlästä, ja sen toteuttamiseen vaaditaan asiantuntemusta, testauksen automatisaatio on herättänyt paljon kiinnostusta. Testien satunnaisgenerointia on tutkittu pitkään, ja sillä onkin saatu helpotettua ohjelmistojen testausta. Satunnaisgenerointia käyttävät mallit saattavat kuitenkin yhä vaatia työtä niiden käyttäjiltä, ja niiden *kattavuus* (coverage), eli kuinka isoa osaa koodista ne testaavat, on usein matala (Ioannides & Eder, 2012; Campos ja muut, 2018).

Muutaman edellisen vuosikymmenen aikana on tutkittu, miten koneoppimista voisi käyttää apuna testien generoinnissa. Sillä onkin onnistuttu luomaan täysin automatisoituja järjestelmiä, jotka saavuttavat korkeamman kattavuuden kuin satunnaisgeneroidut mallit (Ioannides & Eder, 2012; Campos ja muut, 2018). Erityisesti monimutkaisissa tehtävissä, kuten mikroprosessorien testaamisessa, ne kuitenkin vaativat vieläkin käyttäjältä vaivannäköä (Ioannides & Eder, 2012). Lisäksi riippuen käyttökohteesta ja toteutustavasta, koneoppiminen ei välttämättä tuota merkittävästi parempia tuloksia kuin satunnaisgenerointi (Shamshiri ja muut, 2015).

Tämä tutkielma on toteutettu kirjallisuuskatsauksena. Tieteelliset lähteet on saatu ACM-, SpringerLink- ja IEEE-tietokannoista. Hakuun käytettyjä termejä olivat esimerkiksi "Test suite generation", "machine learning", "genetic" sekä "neural network". Hauissa otin mukaan vain edellisen kymmenen vuoden aikana julkaistut tutkimukset, jotta tieto olisi mahdollisimman ajankohtaista. Löydetyistä lähteistä tarkemmin luettavaksi valikoituivat ne, jotka otsikon ja tiivistelmän perusteella käsittelivät koneoppimista.

Lisäksi joitain lähteitä löytyi jo löydettyjen lähteiden lähdeluetteloista, sekä etsimällä tutkimuksia, jotka käsittelevät EvoSuite-ohjelmistoa.

Tämän tutkielman tavoitteena on selvittää, miten koneoppimista käytetään testien generoimisessa, mitä hyötyä siitä on verrattuna perinteisiin tapoihin luoda testejä automaattisesti ja mitä heikkouksia siinä on. Kappaleessa 2 selitetään, miten koneoppimista käytetään testien generoinnissa. Kappaleessa 3 käsitellään hyviä ja huonoja puolia koneoppimisen käytössä testien generointiin, sekä miten se toimii käytettynä oikeisiin ohjelmistoihin. Automaattisen testien generoinnin hyödyllisyyden arviointi perustuu EvoSuite-ohjelmistoon, sillä siitä on saatavilla laajasti tutkimusta. Lisäksi se on saatavilla ilmaiseksi sekä komentoriviohjelmana, että laajennoksena moniin yleisiin Java-ohjelmointityökaluihin, joten sen hyödyntäminen käytännössä on mahdollista jo nyt kaikille (EvoSuite, 2021). Kappaleessa 4 käsitellään tutkielmaa kokonaisuudessaan, ja mitä johtopäätöksiä sen perusteella voidaan tehdä. Kappaleessa 5 on lyhyt yhteenveto tutkielmasta.

2 Testien generointi

Testejä voidaan generoida automaattisesti ilman koneoppimista. Silloin generointi suoritetaan useimmiten satunnaishaulla. Satunnaishaussa generoidaan testejä, joiden lauseet ja testattavalle luokalle antamat syötteet ovat satunnaisia. Jos nämä testit johtavat aiemmin suorittamattoman koodin suorittamiseen, ne lisätään *testiohjelmaan* (test suite), muulloin ne hylätään. Tämä johtaa siihen, että satunnaishaulla generoidut testiohjelmat voivat olla todella laajoja, jolloin niiden suorittaminen on hidasta. Toinen mahdollinen ongelma on se, että monien koodin osien suoritus vaatii juuri tietyn arvon muuttujalle, jonka generointi on hyvin epätodennäköistä. Tämä ongelma pystytään kuitenkin kiertämään hakemalla ohjelmakoodissa esiintyviä muuttujien arvoja, ja merkitsemällä muistiin ohjelmaa suorittaessa esiintyviä arvoja. (Shamshiri ja muut, 2015)

Satunnaishakua on pyritty parantamaan vähentämällä sen satunnaisuutta lisäämällä testien generointiin ohjausta muutenkin kuin syötearvojen generointiin. Koska kehittyneempää ohjausta ei voida lisätä vain tarkastelemalla ohjelmakoodia, siihen tarvitaan edistyneempiä tekniikoita, kuten koneoppimista. Tällä hetkellä edistyneimpiä koneoppimismalleja testien generointiin ovat erilaiset evolutiiviset mallit (Ioannides & Eder, 2012). Vaikka evolutiiviset algoritmit voidaan toteuttaa monella eri tavalla, niitä kaikkia yhdistää inspiraatio evoluutiosta. Evolutiivisissa algoritmeissa mahdolliset ratkaisut koodataan yksilöiksi, joista koostuvaa populaatiota optimoidaan geneettisillä operaatioilla, kuten risteytys, mutaatio ja valinta (Campos ja muut, 2018). Vuosikymmenen alussa edistyneimmissä ohjelmistoissa, kuten EvoSuite-ohjelmistossa, yksilönä oli kokonainen tes-

tiohjelma, jotka risteytyessään luovat jälkeläisiä, joissa on kummankin vanhemman yksittäisiä testejä (Fraser & Arcuri, 2013). Mutaatiossa testiohjelmiin lisätään satunnainen määrä testejä, ja olemassa olevista testeistä muokataan, lisätään tai poistetaan rivejä (Campos ja muut, 2018). Testien toimivuus mutaation jälkeen varmistetaan erilaisilla korjaustoiminnoilla (Campos ja muut, 2018). Ohjelmistotestejä generoidessa risteytyvien yksilöiden valinta suoritetaan kattavuuden perusteella, jolloin testiohjelmat, joilla on suurin kattavuus, risteytetään (Campos ja muut, 2018).

Kattavuutta voidaan mitata eri tavoin, tutkimuskirjallisuudessa yleisin niistä on *haarakattavuus* (branch coverage), joka mittaa kuinka monessa ohjelmakoodin kontrollirakenteiden muodostamassa haarassa on käyty (Fraser & Arcuri, 2013). Tällöin kukin haara on yksi kattavuustavoite. Viime aikoina on alettu yhdistämään eri kattavuuskriteerejä, jotta saadaan tuotettua vikoja paremmin löytäviä testejä (Campos ja muut, 2018). Useimmiten kahden yhtä hyvän testiohjelman välillä valintakriteerinä käytetään niiden testien tai koodirivien määrä, jotta valmiin ohjelman koko saadaan minimoitua (Campos ja muut, 2018).

Perinteiset evolutiiviset algoritmit näkevät laajan kattavuuden yksittäisenä tavoitteena, jota varten ne optimoivat testiohjelmia. On myös olemassa monitavoitteisia evolutiivisia malleja. Niistä tehokkaimmaksi on osoittautunut DynaMOSA (Campos ja muut, 2018). Kokonaisten testiohjelmien sijaan se luo yksittäisiä yksikkötestejä, joita se käsittelee risteytyksillä ja mutaatioilla (Panichella ja muut, 2018). Nämä operaatiot ovat samankaltaisia kuin kokonaisia testiohjelmia käsittelevissä ohjelmistoissa. Näitä testejä arvioidaan sen perusteella, kuinka hyvin ne kattavat yhden kattavuustavoitteen (Panichella ja muut, 2018). DynaMOSA tallentaa lyhyimmän testin kullekin kattavuustavoitteelle testiarkistoon, joka algoritmin loputtua muodostaa testiohjelman (Panichella ja muut, 2018).

Myös muiden evolutiivisten algoritmien tehokkuutta voidaan parantaa tallentamalla hyväksi havaittuja testejä testiarkistoihin. Tämä on erityisen hyödyllistä silloin, kuin testiohjelmien tehokkuutta arvioidaan monen kattavuuskriteerin avulla. Tällöin kun löydetään testi, joka kattaa aiemmin kattamattoman tavoitteen, se lisätään testiarkistoon. Tämän jälkeen valinta optimoidaan dynaamisesti ottamaan huomioon vain yhä kattamattomat tavoitteet. Tällöin lopullinen testiohjelma koostuu arkistossa olevista testeistä. Arkistossa olevia testejä voidaan myös esimerkiksi mutatoida kokonaan uusien testien generoinnin sijaan. (Campos ja muut, 2018)

Evolutiivisten algoritmien lisäksi testien generointiin on sovellettu monia muita koneoppimismetodeja. Hyviä tuloksia kattavuuden suhteen onkin saatu *Markovin malleilla* (Markov model) ja *Bayes-verkoilla* (Bayesian Network), mutta niitä käyttäessä vaaditaan enemmän manuaalista työtä mallin luomisessa, ja enemmän tietoa koneoppimisesta ja testattavasta ohjelmasta. (Ioannides ja Eder, 2012).

Myös neuroverkkoja on käytetty ohjelmistotestien generointiin. Neuroverkot ovat ihmisten aivojen toimintaa matkivia tietokonemalleja. Testien generointiin on myös kehitetty neuroverkkoihin pohjautuva malli, joka ryhmittelee satunnaisgeneroituja testejä sen mukaan mitä osaa ohjelmasta ne testaavat. Jos testi ei lisää kattavuutta tai testaa mitään neuroverkon tunnistamaa koodiryhmää, se hylätään. Lisäksi satunnaisgenerointia pyritään ohjaamaan luomaan testejä niille ryhmille, joita ei vielä ole testattu niin kattavasti. Siinä missä aiemmin esitellyt evolutiiviset mallit pyrkivät generoimaan parempia testejä kuin satunnaisgenerointi, tämä neuroverkkoon pohjautuva malli pyrkii ohjaamaan satunnaisgenerointia keskittymään tiettyihin osiin koodista. Lisäksi se karsii turhia testejä pois, jolloin niitä ei tarvitse suorittaa, jotta niiden kattavuus voidaan arvioida. Tämän tarkoituksena on nopeuttaa testien generointia, keskittää testit tärkeisiin osiin koodista ja lisätä kattavuutta luomalla samassa ajassa useampia testejä. (Wang ja muut, 2018)

3 Koneoppimisen soveltuvuus ohjelmistotestaukseen

Koneoppimisen käytöstä ohjelmistotestien generointiin on löydetty sekä hyviä että huonoja puolia. Näitä käsitellään kohdissa 3.1 ja 3.2. Suurin osa tutkimuksista keskittyy arvioimaan helposti mitattavia asioita, kuten kattavuutta. Kenties edistynein koneoppimista testien generointiin käyttävä sovellus on EvoSuite, jonka hyödyllisyyttä on tutkittu myös käytännöllisemmästä näkökulmasta. Tällaisten tutkimusten tuloksia tarkastellaan kohdassa 3.3.

3.1 Vahvuuksia

Koneoppimisen avulla on saatu kehitettyä kattavampia testejä kuin satunnaishakuun perustuvilla malleilla. Campoksen ja muiden (2018) suorittamien kokeiden mukaan DynaMOSA saavutti 22 % suuremman kattavuuden monella kattavuuskriteerillä mitattuna kuin satunnaishaku. Lisäksi koneoppimisen avulla on pystytty luomaan testiohjelmia, joiden pituus ja suoritusaika on vain yksi yhdeksäsosa vastaavan satunnaisgeneroidun testiohjelman pituudesta (Ioannides & Eder, 2012).

Koneoppimisella on saavutettu hyviä tuloksia tavallisia ohjelmistoluokkia vaikeammissakin tehtävissä. Mikroprosessorien Register Transfer Language (RTL) -toteutuksen testaamisessa haasteena on, että niiden edistyneiden ominaisuuksien vuoksi voi olla hyvin vaikeaa arvioida, miten johonkin tiettyyn kattavuustavoitteeseen päästään, minkä vuoksi testien kirjoittaminen niille on erittäin aikaa vievää. Niiden testaamiseen on kuitenkin kehitetty monia koneoppimismalleja, jotka saavuttavat yli 90 % kattavuuden, ja löytävät suurimman osan tunnetuista vioista testattavissa järjestelmissä. (Ioannides & Eder, 2012)

Koneoppimisen suurin etu perinteiseen, ohjelmoijien suorittamaan testien luomiseen on nopeus. Campoksen ja muiden (2018) testeissä EvoSuite-ohjelmistossa toteutettu

DynaMosa-algoritmi saavutti yhdelle luokalle keskimäärin 86 % kattavuuden 60 sekunnin hakubudjetilla. Vastaavan kattavuuden saavuttamiseen menisi ohjelmoijalla paljon pidempään, joten koneoppimisen avulla suoritettu testien generointi voi säästää ohjelmistoprojekteissa merkittävästi aikaa ja rahaa. Koneoppiminen voi nopeuttaa testien generointia myös suhteessa muihin algoritmeihin. Wangin ja muiden (2018) toteuttama neuroverkkoon perustuva malli oli heidän testeissään peräti 24.5 kertaa nopeampi kuin perinteinen malli. He huomasivat myös, että heidän mallinsa suoriutui paremmin, kun sen antoi itse luokitella syötteet, sen sijaan että ne luokiteltiin manuaalisesti (Wang ja muut, 2018). Tämä osoittaa, että neuroverkot voivat löytää testeistä epäintuitiivisia yhteyksiä, joita ihmiset eivät kykene havaitsemaan (Wang ja muut, 2018).

3.2 Heikkouksia

Eräs suurimmista esteistä koneoppimisen laajalle käytölle testien generointiin on sen vaatima ennakkotieto. Monien, erityisesti mikroprosessorien testaamiseen tarkoitettujen, koneoppimismallien käyttö vaatii hyvän käsityksen testattavasta järjestelmästä ja koneoppimisesta (Ioannides & Eder, 2012). Vaikka nykyaikaiset evolutiivisiin algoritmeihin perustuvat mallit eivät välttämättä vaadi ollenkaan esitietoa järjestelmästä niiden käyttöä varten, ne saattavat yhä vaatia hyvien parametrien etsimistä parhaan mahdollisen lopputuloksen saavuttamiseen (Ioannides & Eder, 2012; Campos ja muut, 2018). Sopivien parametrien löytäminen vaatii tietoa ja taitoa, joita harvalla on, mikä rajoittaa tällaisten ratkaisujen hyödyllisyyttä manuaalisen testien luontiin verrattuna. Esimerkiksi EvoSuite-ohjelma tosin tarjoaa oletusarvot algoritmeilleen, joka lievittää tätä ongelmaa (Campos ja muut, 2018).

Toinen ongelma koneoppimisen käytössä on optimaalisen algoritmin valitseminen. Campoksen ja muiden (2018) mukaan yleiskattavuus monelle kriteerille vaihteli eri algoritmien välillä 54 %:sta aina 86 %:iin. DynaMOSA menestyi Campoksen ja muiden (2018) vertailussa parhaiten, ja generoi 21 ohjelmistoluokalle 346:sta tilastollisesti merkitsevästi kattavammat testit kuin mikään muu algoritmi. Toisaalta 13 luokassa jokin toinen yksittäinen algoritmi menestyi selvästi paremmin kuin mikään muu algoritmi (Campos ja muut, 2018). Muissa luokissa testien kattavuudessa ei ollut tilastollisesti merkitsevää eroa (Campos ja muut, 2018). Kullekin luokalle sopivan algoritmin löytäminen on työlästä ja aikaa vievää, mikä ei ole toivottavaa ohjelmistotestausta automatisoivissa järjestelmissä. Sopivan algoritmin löytämistä vaikeuttaa myös algoritmeihin sisältyvä satunnaisuus, minkä vuoksi mahdollisimman hyvän testiohjelman luomista varten algoritmi pitää suorittaa monta kertaa (Campos ja muut, 2018).

Koneoppimisesta ei aina ole hyötyä ohjelmistotestien automaattisessa generoinnissa. Shamshirin ja muiden tutkimuksen mukaan evolutiiviset algoritmit eivät ole merkittävästi parempia kuin satunnaishaku. Heidän mukaansa tähän on muutama syy. Evolutiiviset algoritmit etsivät *ohjausta* (guidance) ohjelmakoodista, esimerkiksi jos siinä

vertaillaan kahta kokonaislukua, evolutiiviset algoritmit pystyvät arvioimaan kuinka lähellä nykyinen testi on sitä, että se johtaa if-lauseen evaluoitumista todeksi. Tämän avulla niiden on usein helppo löytää oikeat parametrit, joilla testi suorittaa kyseisen haaran. Usein ohjelmakoodissa kuitenkin vertaillaan esimerkiksi totuusarvomuuttujia, joista ei voida arvioida kuinka lähellä testi on oikeita parametreja, koska silloin testi on aina yhtä kaukana oikeista parametreista, jos se ei suorita haaraa. EvoSuite pystyy muokkaamaan joitain tällaisia osioita koodista, mutta esimerkiksi lippumuuttujia se ei pysty muokkaamaan sellaisiksi, että niistä olisi hyötyä evolutiivisille algoritmeille. Silloin kun ohjelmakoodista ei saa ohjausta, evolutiivisilla algoritmeilla ei ole etua satunnaishakuun, vaan päinvastoin ne kuluttavat turhaan aikaa erinäisiin operaatioihinsa, samalla kun satunnaishaku pystyy suorittamaan testejä, ja mahdollisesti löytämään parempia parametreja. Satunnaishaku suorittaakin noin 1,3 kertaa enemmän testejä kuin evolutiiviset algoritmit samassa ajassa. Vaikka evolutiivisille algoritmeille antaisi tätä korvaamaan 30 % enemmän aikaa, se ei anna niille merkittävää etua. Koska ne suosivat vanhojen testien muokkaamista uusien generoinnin sijaan, niillä voi kestää pidempään kattaa huonosti ohjausta antavia haaroja. Tämän vuoksi evolutiiviset algoritmit myös kattavat haarattomia funktioita huonommin, sillä vaikka niiden kattaminen vaatisi vain niiden kutsumista, olemassa olevia testejä muokatessa on epätodennäköistä, että testiin lisätään kutsu tiettyyn funktioon. Satunnaishaku sen sijaan luo jatkuvasti testejä lisäämällä lauseita, joten sillä on suurempi todennäköisyys luoda testi, joka kutsuu tällaista funktiota. Lisäksi, jos evolutiivinen algoritmi ei käytä testiarkistoa, se voi hylätä testiohjelmia, jotka kattavat osia koodista, joita paras testiohjelma ei kata. Satunnaishaulla tehdyt ohjelmat sen sijaan kattavat kaikki tavoitteet, jotka haun aikana on katettu. (Shamshiri ja muut, 2015)

Testien generoinnissa on generointitavasta riippumattomiakin ongelmia. Koska ohjelmistoa testatessa testattava koodi täytyy suorittaa, tietyt operaatiot, kuten tiedostojärjestelmän manipulointi, voivat aiheuttaa epätoivottavia sivuvaikutuksia suoritusympäristöön (Fraser ja Arcuri, 2014). Esimerkiksi kun Fraser ja Arcuri (2014) ajoivat vahingossa satunnaisesti testejä generoivan Randoop-ohjelmiston ilman oikeita turvatoimia, se poisti kokonaan 49 sadasta kymmenestä ohjelmasta, joille sen oli tarkoitus generoida testejä. Tämän vuoksi testejä generoivia sovelluksia tulee rajoittaa, mistä johtuen ympäristön kanssa mahdollisesti vaarallisesti vuorovaikuttavaa koodia on vaikea tai mahdoton testata. Lisäksi jos sovellus käsittelee itsensä ulkopuolisia tiedostoja, testitiedostojen generoiminen automaattisesti on vaikeaa. Esimerkiksi EvoSuite-ohjelmisto ei kykene luomaan MP3-tiedostoja, joten sillä ei kyetä testaamaan koodia, joka käsittelee niitä (Campos ja muut, 2018).

3.3 Hyödyntäminen käytännössä

Koneoppimisen avulla voidaan jo nyt generoida testejä esimerkiksi omiin Java-kielellä tehtyihin ohjelmiin ilmaisella EvoSuite-ohjelmistolla (EvoSuite, 2021). Ohjelmistoa on

testattu luomalla sitä käyttäen testiohjelmat 110:lle avoimen lähdekoodin sovellukselle, joiden keskimääräinen kattavuus oli 71 % (Fraser & Arcuri, 2014). EvoSuiten testien kattavuus kuitenkin laskee huomattavasti, jos testattava luokka on vuorovaikutuksessa ympäristönsä kanssa. Esimerkiksi luokille, jotka manipuloivat tiedostojärjestelmää, EvoSuiten testien kattavuus oli vain 57 % (Fraser & Arcuri, 2014). Jos joukosta karsittiin sovellukset, jotka käyttävät tällaisia vaarallisia operaatioita, kattavuus nousi 84 %:iin (Fraser & Arcuri, 2014).

Ohjelman on todettu myös löytävän oikeita ongelmia aidossa koodissa. Ohjelmiston luomien testien avulla löydettiin sadasta avoimen lähdekoodin sovelluksesta vähintään 1694 virhettä sovelluksen toiminnassa. Toisaalta mukana oli monia vääriä hälytyksiä, jotka johtuivat siitä, että koodin kirjoittaja on tiennyt, että virheellinen syöte ei ole syystä tai toisesta mahdollinen, tai hän on esitellyt mahdolliset poikkeukset puutteellisesti. Enemmistö luokista, joista ohjelmisto löysi virheitä, eivät sisältäneet tutkijaryhmän löydösten perusteella toiminnallisia vikoja. Tällaisia vääriä hälytyksiä voisi tosin vähentää huomattavasti kirjoittamalla koodia, jossa esimerkiksi kaikki mahdolliset virheet esitellään *throws*-avainsanalla. Koska perusteettomat oletukset koodin käytöstä voivat johtaa ongelmiin, voidaankin pohtia parantaisiko automaattisten testien laaja käyttöönotto yleistä ohjelmakoodin laatua. (Fraser & Arcuri, 2015)

EvoSuite-ohjelmisto on todettu hyödylliseksi käytännön ohjelmistokehitystä simuloivissa tutkimuksissa. Sen käytön on havaittu vähentävän yhden luokan testaamiseen kuluvaa aikaa 36 %. Lisäksi useimmissa tapauksissa sen tuottamilla testiohjelmilla on suurempi kattavuus kuin manuaalisesti tehdyillä testiohjelmilla. EvoSuiten käyttäjien kokemuksen mukaan se nopeuttaa ja helpottaa testausta, ja auttaa luomaan parempia testiohjelmia. Lisäksi EvoSuite kykenee generoimaan testejä tilanteille, joita ohjelmoija ei osaa huomioida, kuten tietyn funktion kutsuminen heti olion luomisen jälkeen. (Rojas ja muut, 2015)

EvoSuite-ohjelmiston hyödyistä huolimatta sen hyödyntämistä ohjelmistokehityksessä haittaa muutama tekijä. 63 % sen käyttäjistä koki, että sen generoimia testejä on vaikea ymmärtää (Rojas ja muut, 2015). Monet käyttäjät kokevat, että ohjelman testien tarkoitus on epäselvä, ja niissä pitäisi olla kommentti selittämässä niiden tavoitetta (Fraser ja muut, 2013). Koska EvoSuite generoi testejä ohjelman havainnoidun käytöksen perusteella, siitä ei ole juurikaan hyötyä testivetoisessa kehityksessä, jossa testit kirjoitetaan ennen kuin ohjelma luodaan (Rojas ja muut, 2015). Se ei myöskään ole järin hyödyllinen täysin päinvastaisessa tilanteessa, jossa olemassa olevalle suurelle projektille generoidaan testejä. Arcurin (2018) mukaan jotkin EvoSuiten generoimat testit epäonnistuvat perusteetta. Vaikkei se ole yleistä, tuhansia luokkia sisältävissä massiivisissa projekteissa se voi silti tarkoittaa satoja perusteettomia epäonnistumisia, joiden korjaamiseen tai poistamiseen kuluu paljon aikaa (Arcuri, 2018). Samankaltaisen ongelman havaitsivat myös

Fraser ja muut (2013), joiden mukaan EvoSuite tuottaa perusteetta epäonnistuvia testejä enemmän kuin manuaalinen testaus. Heidän mukaansa tällaiset virheet saattavat johtua EvoSuiten testien vaikeaselkoisuudesta, tai siitä että perusteetta epäonnistuvia testejä on vaikea korjata yleensä ottaen, ja automaattisessa generoinnissa niitä syntyy enemmän (Fraser ja muut, 2013).

Iso ongelma EvoSuiten käytössä on testien muokkaamisen vaikeus, jonka vuoksi ohjelmiston potentiaali menee hukkaan. EvoSuite generoi testit ohjelman sen hetkisen toiminnan perusteella, ja käyttäjän tulee manuaalisesti muokata testien assert-lauseet vastaamaan ohjelman tarkoitettua toimintaa. Oikein muokattuna EvoSuiten generoimat testit voisivat paljastaa ohjelman toiminnassa useampia virheitä kuin manuaalisesti tehdyt testit, vaikka näiden välillä ei ole havaittu merkittävää eroa löydettyjen virheiden määrässä. Testien muokkausta vaikeuttaa se, että EvoSuite generoi monelle luokalle yli kaksi kertaa enemmän assert-lauseita kuin ohjelmoijien tekemissä testeissä on, jolloin niiden tarkastaminen on työlästä. (Fraser ja muut, 2013)

Ohjelman mahdollista hyötyä vähentää myös hyvien käytäntöjen ja tietämyksen puute. Ohjelman luomien testien avulla kehitettyjen ohjelmien laatu parani, mitä enemmän ohjelmoija käytti aikaa testien ymmärtämiseen, muokkaamisen ja vikojen etsimiseen niiden avulla. Jos testejä generoitiin jatkuvasti eikä ohjelmoija käyttänyt niiden parissa paljoa aikaa, ohjelman laatu kärsi. Erään käyttäjän havainnoitiin luovan testejä ensin manuaalisesti, minkä jälkeen hän antoi ohjelman generoida testejä, joista osa olisi voinut korvata manuaalisesti tehdyt testit. Tämä kulutti turhaan aikaa. Toisaalta toinen käyttäjä ajoi ohjelman jo ennen kuin hän oli toteuttanut ollenkaan ohjelmakoodia, jolloin luodut testit olivat käytännössä hyödyttömiä, koska ohjelma pystyy tarkkailemaan vain toteutettua toimintaa. Tällaiset tapaukset kertovat, että käyttäjät eivät tiedä milloin ja miten automaattisia testaustyökaluja kannattaa käyttää. (Rojas ja muut, 2015)

Yleensä ottaen EvoSuiten käyttö ei vaikuta merkittävästi ohjelmistojen tai testiohjelmien laatuun verrattuna manuaaliseen testaukseen. Rojasin ja muiden (2015) mukaan EvoSuiten generoimien testien avulla kehitetyt luokat sisälsivät yhtä paljon virheitä kuin manuaalisten testien avulla kehitetyt luokat. Fraserin ja muiden (2013) mukaan EvoSuiten avulla kehitetyt testiohjelmat löytävät mahdollisesti hieman vähemmän virheitä koodista, joskin ne voisivat parhaassa tapauksessa löytää useampia virheitä.

4 Keskustelu

Ohjelmistotestien generoinnissa on edellisen vuosikymmenen aikana tapahtunut paljon muutoksia. Vaikka evolutiiviset algoritmit ovat yhä pääasiallisesti käytetty koneoppimismalli, niiden toteutustapoihin on tullut muutoksia. 2010-luvun alussa EvoSuiten julkaisun

myötä kokonaisen testiohjelman generointi kerralla nousi parhaimmaksi tavaksi generoida testiohjelmia. Vuosikymmenen edetessä sitä paranneltiin testiarkistoilla, joka paikkasi sen puutteita verrattuna satunnaishakuun, mutta samalla teki lopullisesta testiohjelmasta kokoelman yksittäisistä testeistä, jotka on noukittu eri testiohjelmistoista. Vuosikymmenen loppupuolella esitelty yksittäisiä testejä generoiva DynaMosa on osoittautunut paremmaksi kuin kokonaisen testiohjelman luovat algoritmit (Campos ja muut, 2018; Panichella ja muut, 2018). Nykyään siitä onkin tullut EvoSuite-ohjelmiston oletusalgoritmi (EvoSuite, 2021).

Eri lähteiden välillä on huomattava ero siinä, onko evolutiivisten algoritmien avulla generoiduilla testiohjelmilla parempi kattavuus kuin satunnaishaun avulla luoduilla testiohjelmilla. Shamshirin ja muiden (2015) mukaan geneettisten algoritmien ja satunnaishaun välillä ei ole merkittävää eroa. Toisaalta Campoksen ja muiden (2018) mukaan evolutiiviset algoritmit suoriutuvat huomattavasti paremmin kuin satunnaishaku. Campos ja muut (2018) spekuloivat, että tälle erolle on kaksi syytä. He käyttivät tutkimuksessaan monimutkaisempia luokkia, ja heidän algoritminsa käyttivät testiarkistoja (Campos ja muut, 2018). Shamshiri ja muut (2015) myöskin spekuloivat, että arkisto johon testit tallennetaan voisi auttaa evolutiivisia algoritmeja, ja Campoksen ja muiden (2018) tutkimuksen perusteella he olivat oikeassa. Tämän perusteella voitaneen päätellä, että evolutiiviset algoritmit ovat parempia kuin satunnaishakuun perustuvat mallit erityisesti monimutkaisissa luokissa, kunhan niiden kanssa käytetään testiarkistoja.

Evolutiiviset algoritmit ovat toistaiseksi osoittautuneet paremmiksi kuin muut koneoppimisalgoritmit. Viime vuosina on kuitenkin alettu tutkimaan myös neuroverkkojen soveltamista automaattiseen ohjelmistotestaukseen, erityisesti mikroprosessorien testaamiseen. Niillä on saatu lupaavia tuloksia, ja ne saattavatkin haastaa evolutiivisia algoritmeja tulevaisuudessa.

Koneoppimista automaattiseen testien generointiin käyttävien ohjelmistojen kehittyneisyydessä on isoja eroja niiden sovellusalojen välillä. Oliopohjaiselle Java-ohjelmointikielelle on vapaasti saatavilla ilmainen EvoSuite-ohjelmisto, jonka toimivuutta on laajalti tutkittu. Muille ohjelmointikielille sen sijaan löytyy hyvin vähäisesti tutkimuskirjallisuutta koneoppimisen käytöstä automaattiseen testien generointiin. Toinen alue, johon koneoppimisavusteista testien generointia on sovellettu tutkimuskirjallisuudessa paljon, on mikroprosessorien mallien verifiointi. Siihen ei kuitenkaan ole kehitetty yhtäkään niin laajasti tutkittua ohjelmistoa kuin EvoSuite, ja esimerkiksi niiden käytännön hyödyllisyyttä ei vaikuteta juurikaan tutkittavan. Mikroprosessorin mallin verifiointiin tarkoitetut algoritmit eivät myöskään ole yhtä edistyneitä kuin Java-kielisen koodin testaamiseen tarkoitetut ohjelmistot, vaan ne usein vaativat käyttäjältä tietoa testattavan järjestelmän toiminnasta sekä koneoppimisesta. Syynä niiden vähäisempään tutkimukseen voi kenties olla esimerkiksi aiheen harvinaisuus verrattuna olio-ohjelmointiin.

Yleensä automaattisesti testejä generoivia ohjelmistoja arvioidaan lähinnä niiden luomien testiohjelmien kattavuuden perusteella. Tutkimuskirjallisuudessa on kuitenkin kyseenalaistettu kattavuuden painottamisen hyödyllisyyttä tällaisten ohjelmistojen kehityksessä. Fraserin ja muiden (2013) tutkimuksen mukaan kattavuudella on heikosti negatiivinen korrelaatio vikojen löytämisen kanssa EvoSuitea käytettäessä. Positiivinen korrelaatio vikojen löytämisen kanssa havaittiin testiohjelman koon ja manuaalisesti tehtyjen testien välillä (Fraser ja muut, 2013). Manuaalisessa testauksessa sen sijaan kattavuus korreloi voimakkaasti vikojen löytämisen kanssa (Fraser ja muut, 2013). Rojasin ja muiden (2015) mukaan ammattimaiset ohjelmistokehittäjät eivät keskity kattavuuteen kehittäessään ohjelmistoja tai ohjelmistotestejä. Tutkimuskirjallisuudessa automaattisesti testejä generoivia sovelluksia on kuitenkin arvioitu lähinnä vain kattavuuden perusteella, vaikka ohjelmistokehittäjät harvoin käyttävät sitä kriteerinä työssään, ja vaikka sillä ei vaikuta olevan positiivista vaikutusta vikojen löytämiseen testejä generoitaessa automaattisesti. EvoSuite on ainoita sovelluksia, jota voi vapaasti käyttää tutkimusten ulkopuolella, ja senkin arviointi perustuu lähinnä kattavuuteen. Muillakin kriteereillä arvioituna se on osoittautunut hyödylliseksi, mutta siitä on myös havaittu useita puutteita, jotka selittävät sen käytön vähäisyyden. Automaattisesti testejä generoivien sovellusten hyödyllisyyden tutkiminen muillakin kriteereillä kuin kattavuudella olisikin tärkeää, jotta niistä voidaan kehittää mahdollisimman hyödyllisiä jokapäiväiseen sovelluskehitykseen. Käytännössä tämä vaatisi lisää tutkimuksia oikeilla käyttäjillä, sillä toistaiseksi heidän kokemuksiaan ei kyetä imitoimaan keinotekoisilla mittareilla, kuten kattavuudella.

Koneoppimisesta on selkeästi hyötyä automaattiselle testien generoinnille. Oikeilla tekniikoilla se kykenee luomaan parempia testejä kuin satunnaishaku, erityisesti monimutkaisille luokille. Automaattinen testien generointi vaikuttaa toistaiseksi joissain tilanteissa hyödylliseltä, mutta pääosin sitä käyttävien sovellusten tulee vielä kehittyä, jotta niiden käyttö voi yleistyä. Tällä hetkellä perusteetta epäonnistuvat testit tekevät EvoSuite-ohjelmiston käytöstä epäkäytännöllistä laajoille ohjelmistoille, joissa on satoja, tai jopa tuhansia luokkia. Koska automaattisesti testejä generoivat ohjelmat tarkkailevat testattavan sovelluksen toimintaa, ne eivät myöskään sovellu testien luomiseen ennen koodin kirjoittamista. Täten testien generointi on tällä hetkellä hyödyllinen lähinnä sovelluksen testaamiseen samalla, kun ohjelmakoodia kirjoitetaan. Silloin se pystyykin oikein käytettynä säästämään ohjelmoijan aikaa ja vaivaa, ilman että ohjelman tai testien laatu kärsii. Tosin tämä pätee vain sellaisissa tapauksissa, joissa testattava sovellus ei käytä tiettyjä toimintoja, joita tällä hetkellä ei pystytä testaamaan.

Jotta koneoppimista ohjelmistotestien generointiin käyttäviä sovelluksia voitaisiin hyödyntää laajemmin, niistä pitäisi korjata muutama puute. Perusteetta epäonnistuvat testit käytännössä estävät EvoSuite-ohjelmiston käytön laajojen projektien testaamiseen. Ohjelmiston kykyä generoida laadukkaita testejä kaikissa tapauksissa tulisi hioa, jotta

väärin epäonnistuvien testien määrä vähenisi. Samoin EvoSuiten tulisi pystyä käsittelemään myös sellaista koodia, jota se ei tällä hetkellä kykene ollenkaan testaamaan. Vaikuttaisi siltä, että generoitujen testien kattavuus on riittävällä tasolla jo nyt, sillä lisääntynyt kattavuus ei Fraserin ja muiden (2013) mukaan auta löytämään vikoja ohjelman toiminnassa. Sen sijaan testien luettavuutta tulisi parantaa, sillä se on useiden käyttäjien mielestä ongelma, ja generoiduilla testeillä voisi havaita useampia vikoja kuin niillä käytännössä havaitaan. Vaikka ohjelmakoodin luettavuus on subjektiivista, generoiduissa testeissä on eroja ihmisten luomiin testeihin, joiden vähentämistä voisi tutkia. Esimerkiksi EvoSuiten generoimissa testeissä on usein moninkertaisesti assert-lauseita verrattuna ihmisten tekemiin testeihin. Tällaisten erojen poistaminen voisi parantaa testien ymmärrettävyyttä. Ohjelmoijat eivät usein osaa käyttää EvoSuitea oikein, joten ohjelmiston dokumentaatiota olisi hyvä parantaa sekä pyrkiä opettamaan sen optimaalista käyttöä. Ohjelmakoodia, jota ei ole vielä kirjoitettu, ei kyetä testaamaan nykyisillä ratkaisuilla, joten sellaisia käyttötapauksia varten tarvitaan kokonaan uudenlaisia ratkaisuja.

5 Yhteenveto

Ohjelmistotestaaminen on työläs prosessi. Työmäärää on pyritty vähentämään generoimalla testit automaattisesti. Automaattisen generoinnin tuloksia on pyritty parantamaan käyttämällä erilaisia koneoppimismenetelmiä. Tällä hetkellä eniten käytetty ja parhaiten toimiva koneoppimismalli testien generointiin on erilaiset evolutiiviset algoritmit, joita kaikkia yhdistää inspiraatio evoluutiosta.

Koneoppimisen avulla pystytään luomaan parempia testiohjelmia kuin ilman sitä. Niillä on parempi kattavuus, ne ovat lyhyempiä ja niiden generointi tapahtuu yhtä nopeasti, tai jopa nopeammin. Koneoppimisen avulla generoituihin testeihin liittyy kuitenkin ongelmia, jotka toistaiseksi estävät niiden laajamittaisen käytön. Testien generointi ei onnistu kaikenlaisille luokille, ja testit saattavat epäonnistua, vaikka testattava sovellus toimisi. Lisäksi testejä on vaikea ymmärtää, ja ohjelmoijat eivät osaa käyttää niitä generoivia sovelluksia oikein.

Toistaiseksi testien generointi automaattisesti ei useimmissa tapauksissa ole hyvä vaihtoehto. Kun sovellusta testataan sitä ohjelmoitaessa, EvoSuite-ohjelmisto pystyy kuitenkin säästämään ohjelmoijan aikaa ilman, että testien tai kehitettävän ohjelmiston laatu kärsii. Jos testejä generoivien ohjelmistojen vioista päästään tulevaisuudessa eroon, ne voivat säästää huomattavasti aikaa ja vaivaa ohjelmistokehityksessä.

Lähdeluettelo

Arcuri, A. (2018). An experience report on applying software testing academic results in industry: we need usable automated test generation. *Empirical Software Engineering : an International Journal, 23(*4), 1959–1981. https://doi.org/10.1007/s10664-017-9570-9

Campos, J., Ge, Y., Albunian, N., Fraser, G., Eler, M. & Arcuri, A. (2018). An empirical evaluation of evolutionary algorithms for unit test suite generation. *Information and Software Technology, 104*, 207–235. *https://doi.org/10.1016/j.infsof.2018.08.010*

EvoSuite. (2021). *Automatic Test Suite Generation for Java.* https://www.evosuite.org/ (Haettu 14.4.2021)

Fraser, G. & Arcuri, A. (2013). Whole Test Suite Generation. *IEEE Transactions on Software Engineering, 39*(2), 276–291. https://doi.org/10.1109/TSE.2012.14

Fraser, G. & Arcuri, A. (2014). A Large-Scale Evaluation of Automated Unit Test Generation Using EvoSuite. *ACM Transactions on Software Engineering and Methodology, 24*(2), 1–42. https://doi.org/10.1145/2685612

Fraser, G. & Arcuri, A. (2015). 1600 faults in 100 projects: automatically finding faults while achieving high coverage with EvoSuite. *Empirical Software Engineering, 20*(3), 611–639. https://doi.org/10.1007/s10664-013-9288-2

Fraser, G., Staats, M., McMinn, P., Arcuri, A. & Padberg, F. (2013). Does automated white-box test generation really help software testers? *Proceedings of the 2013 International Symposium on Software Testing and Analysis*, 291–301. https://doi.org/10.1145/2483760.2483774

Ioannides, C. & Eder, K. (2012). Coverage-Directed Test Generation Automated by Machine Learning -- A Review. *ACM Transactions on Design Automation of Electronic Systems, 17*(1), 1–21. https://doi.org/10.1145/2071356.2071363

Panichella, A., Kifetew, F. & Tonella, P. (2018). Automated Test Case Generation as a Many-Objective Optimisation Problem with Dynamic Selection of the Targets. *IEEE Transactions on Software Engineering, 44*(2), 122–158. https://doi.org/10.1109/TSE.2017.2663435

Rojas, J., Fraser, G. & Arcuri, A. (2015). Automated unit test generation during software development: a controlled experiment and think-aloud observations. *Proceedings of the 2015 International Symposium on Software Testing and Analysis*, 338–349. https://doi.org/10.1145/2771783.2771801

Shamshiri, S., Rojas, J., Fraser, G. & McMinn, P. (2015). Random or Genetic Algorithm Search for Object-Oriented Test Suite Generation? *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, 1367–1374. https://doi.org/10.1145/2739480.2754696

Wang, F., Zhu, H., Popli, P., Xiao, Y., Bodgan, P. & Nazarian, S. (2018). Accelerating Coverage Directed Test Generation for Functional Verification: A Neural Network-based Framework. *Proceedings of the 2018 on Great Lakes Symposium on Vlsi*, 207–212. https://doi.org/10.1145/3194554.3194561