Lappeenrannan–Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma

MAINOSVERKKOJEN PELATTAVILLE MAINOKSILLE TARKOITETTUJEN TESTITYÖKALUJEN INTEGRAATIO JA AUTOMATISAATIO

Työn tarkastaja: Assoc. Prof. Ari Happonen

TIIVISTELMÄ

Lappeenrannan–Lahden teknillinen yliopisto LUT
School of Engineering Science
Tietotekniikan koulutusohjelma
Aarne Savolainen

Mainosverkkojen pelattaville mainoksille tarkoitettujen testityökalujen integraatio ja automatisaatio

Kandidaatintyö 2021

20 sivua
Työn tarkastaja: Assoc. Prof. Ari Happonen

Hakusanat: pelattavat mainokset, testaaminen, testauksen automatisointi, peliteollisuus, markkinointi
Keywords: playable advertisements, testing, test automation, gaming industry, marketing

Tämä kandidaatintyö on tehty yhteistyössä Seepia Gamesin kanssa. He testaavat pelattavat mainoksensa jokaisessa mainosverkon testityökalussa erikseen. Tässä työssä tutkitaan, miten pelattavien mainosten testausta voisi automatisoida ja luodaan työkalu tutkimusten pohjalta, joka automatisoi testausprosessia. Työkalun luonti onnistui konseptitodistuksen tasolla ja se testauttaa mainokset automaattisesti tietyissä mainosverkkojen testityökaluissa. Työkalussa on kuitenkin puutteita, sillä se vaatii muun muassa huomattavasti ylläpitoa.

ABSTRACT

Lappeenranta–Lahti University of Technology LUT
School of Engineering Science
Degree Programme in Software Engineering
Aarne Savolainen

The integration and automation of advertisement networks' testing tools for playable advertisements

Bachelor's Thesis 2021

20 Pages
Examiner: Assoc. Prof. Ari Happonen

Keywords: playable advertisements, testing, test automation, gaming industry, marketing

This thesis was done in cooperation with Seepia Games. They test their playable advertisements in each advertisement network one by one. The purpose of this thesis is to find out what can be automated in the testing process of playable advertisements and to create a tool on the basis of the research that automates the testing process. The creation of the tool was successful on a concept proof level. The tool automatically sends adverts to a select few advertisement network's testing tools to be tested. The tool has a few imperfections though, as for example, it requires a lot of maintenance.

ALKUSANAT

Työ on tehty Lappeenrannassa Lappeenrannan–Lahden Teknillisessä yliopistossa. Kiitän Seepia Gamesiä tuesta työn teknisessä osuudessa, sekä työn teoreettisessa osuudessa tukea tarjonneita professoreita, opettajia ja ystäviä.

SISÄLLYSLUETTELO

1 JOHDANTO 3

1.1 PELATTAVIEN MAINOSTEN TAPAUSTUTKIMUS JA CASE YRITYS SEEPIA GAMES 3

1.2 TAVOITTEET JA RAJAUKSET 4

1.3 TYÖN RAKENNE 5

2 KIRJALLISUUSKATSAUS 6

2.1 OHJELMISTOTESTAUS 6

2.2 OHJELMISTOTESTAUS MANUAALISESTI JA AUTOMAATTISESTI 7

2.3 TESTAUKSEN AUTOMATISOINTI 8

3 RATKAISUMENETELMÄT 10

3.1 TAPAUSTUTKIMUSMENETELMÄ JA SEN SOVELTAMINEN TYÖHÖN 10

3.2 SEEPIA GAMESIN TAPAUSTUTKIMUSTARVETTA VARTEN LUOTAVA TYÖKALU 11

3.3 TOTEUTUS 13

4 TYÖN TULOSHAVAINNOT JA TESTAAMISEN TOTEUTTAVA OHJELMAKOODI 14

4.1 OHJELMAKOODI 14

5 YHTEENVETO 20

LÄHTEET 21

SYMBOLI- JA LYHENNELUETTELO

AppLovin	Mainosverkko
Case Study	Tapauskohtaisia tutkimuksia varten kehitetty tutkimusmenelmä
HTML	Hypertext Markup Language
IronSource/Nucleo	Mainosverkko
JSON	JavaScript Object Notation
Mintegral	Mainosverkko
URL	Uniform Resource Locator
Web	World Wide Web
ZIP	Tiedon pakkausmuoto

1 JOHDANTO

Mainontaa näkee kaikkialla, ja teknologian kehittyessä mainoksista tulee yhä monimutkaisempia ja näyttävämpiä. Varsinkin mobiililaitteiden ja sosiaalisen median nopea kehitys on antanut mainostajille uusia tapoja luoda kohdennettuja mainoksia (Dehghani & Tumer, 2015, Wong et al., 2015, Lee et al., 2009). Mainokset ovat oleellinen osa sitä, miten varsinkin mobiilisovellukset saavat tuottoa. Mainoskirjastoja integroidaan sovelluksiin, jotka näyttävät mainoksia ja antavat tuottoa sen perusteella, mitä mainoksia näytetään ja miten käyttäjät käyttävät mainoksia. (Ahasanuzzaman et al., 2020.) Pelattavien mainosten suosio on kasvanut, sillä ne antavat jo itse mainoksessa pienen käsityksen siitä, mitä mainostettava tuote sisältää (Schmitt, 2017). Kun mainoksista tulee yhä vaativimpia, niiden testaamisen tärkeys kasvaa entisestään.

Testaamisen tärkeys ylipäätänsä kasvaa jatkuvasti, ja sitä varten luodaan yhä uudempia ja parempia teknologioita, jotta testaaminen helpottuisi ja nopeutuisi. Ennen testaaminen tehtiin joko manuaalisesti tai ei ollenkaan, mutta nykyään siihen pyritään saamaan omat tiimit ja sitä pyritään automatisoimaan yhä enemmän (Spinellis, 2017). Testaamisen tavoitteena on muun muassa laadun varmistus ohjelmakoodissa käytettävissä funktioissa, ohjelmakoodin käytöksessä ja suorituskyvyssä, sekä laadun varmistus palveluissa ja ominaisuuksissa, kuten käytettävyydessä ja suojauksessa (Gao et al., 2014).

Pelattavat mainokset ovat eräänlaisia mainoksia, joihin sisältyy pieni peli. Markkinointi pelattavilla mainoksilla tapahtuu niin, että käyttäjät saavat alustavan kokemuksen mainostettavasta tuotteesta ja tällä alustavalla kokemuksella pyritään houkuttelemaan käyttäjä hankkimaan koko tuotteen. Nämä pelattavat mainokset pitää testata samalla lailla, kuin mikä tahansa muukin ohjelmistoprojekti. Seepia Games on eräs pelialan yritys ja sen erikoisuutensa on juuri pelattavat mainokset, joiden testaaminen on työlästä.

1.1 Pelattavien mainosten tapaustutkimus ja case yritys Seepia Games

Kandidaatintyö on tehty yhteistyössä Seepia Gamesin kanssa. Mainosverkoilla, joissa näitä mainoksia näytetään, on erilaisia vaatimuksia mainoksille, kuten pieni tiedostokoko. Lisäksi

mainosten pitää tukea tiettyjä kirjastoja, sekä niissä pitää olla tiettyjä ominaisuuksia. Nämä vaatimukset pitää käydä testaamassa ja täyttää ennen kuin mainoksia voidaan mainosverkoissa näyttää.

Tällä hetkellä Seepia Games testaa mainoksensa sopivuuden manuaalisesti niin, että Seepia Games luo mainoksestaan jokaiselle mainosverkon työkalulle oman linkin ja/tai tiedoston, jonka he syöttävät jokaiseen testityökaluun erikseen ja odottavat vastausta. Seepia Games kokee tämän prosessin olevan hidasta ja haluavat nopeuttaa prosessia automatisoimalla sen.

1.2 Tavoitteet ja rajaukset

Kandidaatintyössä tavoite on tutkia ja kokeilla miten automatisointia voisi kehittää Seepia Gamesin kokoisen yrityksen tarpeiden mukaan. Yksi tällainen mahdollisuus voisi olla esimerkiksi luoda Seepia Gamesin työkalujen ja mainosverkkojen testityökalujen väliin työkalu, jolla testausprosessin pystyisi automatisoimaan. Tämän työkalun on tarkoitus olla web-pohjainen työkalu, joka ottaisi vastaan mainoksen linkit, jotka tässä tapauksessa ovat Seepia Gamesin luomia, kävisi testaamassa automaattisesti ne kaikissa annetuissa testityökaluissa ja palauttaisi testituloksen, jossa kerrotaan, mitkä testityökalut antoivat hyväksytyn tuloksen ja mitkä antoivat virheilmoituksen. Tarkoitus on luoda Seepia Gamesin omista työkaluista erillinen työkalu, jota käytettäisiin rajapintojen kautta heidän omien työkalujen kanssa.

Mainosverkot, joita tässä työssä tutkitaan ja käytetään, ovat AppLovin, Facebook, Google, IronSource/Nucleo ja Mintegral. Näiden mainosverkkojen testityökalut vaativat joko HTML- tai ZIP-muotoisen linkin tai tiedoston. Tästä työstä on rajattu Unity-mainosverkko pois, koska se poikkeaa huomattavasti muista mainosverkoista, mikä olisi tehnyt yhteisestä toteutuksesta varsin hankalan, eikä sen ajateltu mahtuvan suunniteltuun aikatauluun.

1.3 Työn rakenne

Työssä käsitellään ensin työn taustaa, tavoitetta, motivaatiota ja rajauksia. Sitten kirjallisuuskatsauksessa käydään läpi, mitä kirjallisuudessa kerrotaan mainonnasta, ohjelmistotestauksesta, testitavoista ja automatisoinnista. Ratkaisumenetelmissä kerrotaan yleisesti tutkimusmenetelmistä, miten työn ongelma voitaisiin mahdollisesti ratkaista, miten tutkimusta vietiin eteenpäin ja miten Seepia Gamesin tapaustutkimustarvetta varten luotu työkalu tehdään. Tuloksissa kerrotaan tutkimuksen tuloksista ja kuinka työkalu luotiin.

2 KIRJALLISUUSKATSAUS

Tässä kappaleessa esitetään ohjelmistotestaukseen, testitapoihin ja ohjelmistotestauksen automatisaatioon liittyvää ja näitä tukevaa kirjallisuutta. Ensin esitetään ohjelmistotestaukseen yleisemmällä tasolla liittyvää kirjallisuutta. Seuraavaksi esitetään testaukseen manuaalisesti ja automaattisesti liittyvää kirjallisuutta, muun muassa minkälaisissa tilanteissa automatisointia näkee. Lopuksi esitetään kirjallisuutta, missä kerrotaan, miksi automatisointia toteutetaan.

2.1 Ohjelmistotestaus

Ohjelmistotestaus on oleellinen osa sen kehitystä ja varmistavat sen laadun (Tsai ja Qi, 2017). Ohjelmistotestauksen voidaan ajatella olevan korvaamaton osa ohjelmistokehitystä (Anand et al., 2013). Ohjelmistotestauksessa käytetään erilaisia testitapauksia mahdollisten ohjelmointivirheiden löytämiseen (Tsai ja Qi, 2017). Yksinkertaisimmillaan testauksen voidaan sanoa olevan ohjelman suorituksen valvomista ja varmistamista, että se toimii, kuten sen on tarkoitus. Testaaminen on laajalti käytetty laadun varmistuskeino. (Bertolino, 2007) Digitalisaation edetessä laajasti lähes jokaisella toimialalla (Kortelainen et al., 2019), järjestelmät monimutkaistuvat koko ajan, minkä takia ohjelmistokehittäjän pitää hallita laajempia kokonaisuuksia (Jahkola et al., 2017), mikä taas tekee testaamisesta yhä vain tärkeämpää. Sovelluksen kompleksisuuden kasvaessa laadun varmistuksesta ja testaamisesta tulee yhä tärkeämpää, mutta myös vaikeampaa ja kalliimpaa. (Bertolino, 2007) Ohjelmistotestaus on myös äärimmäisen vaativaa työmäärän suhteen (Anand et al., 2013). Aiemmat tutkimukset arvioivat testauksen vievän noin puolet ohjelmiston kehittämiskustannuksista (Bertolino, 2007). Kuitenkin testauksen vähättely tai poisjättäminen tuo vielä enemmän kustannuksia, kuin itse testaaminen (Garousi and Mäntylä, 2016). Minkä takia, esimerkiksi oppilaitoksissa testaaminen pitäisi olla jatkossa paremmin huomioon otettu osa-alue, ohjelmointia opetettaessa. Aikataulupaineen alla tapahtuvaa ohjelmistoratkaisuiden kehittämistä (Porras et al., 2018), sekä peleihin ja pelienkehittämiseen liittyvää opetuksen kehitystutkimusta on joka tapauksessa harjoitettu jo aiemminkin (Ikonen et al., 2007), miksei siis tuoda mukaan myös ohjelmointivirheiden havaitsemista ja siihen liittyvää testaamisen osaamista ja automatisointia.

Ohjelmointivirhe tarkoittaa sitä, että ohjelma ei tee sitä, mitä loppukäyttäjä odottaa sen tekevän. Toinen selitys ohjelmointivirheelle on, että ohjelman ohjelmointivirheet mitataan siten, kuinka epähyödyllinen ohjelma on. (Kaner et al., 1999) Ohjelman tai ohjelmiston testaaminen täysin tarkoittaa sitä, että testauksen päätyttyä ohjelmassa tai ohjelmistossa ei ole yhtäkään löytymätöntä ohjelmointivirhettä. On olemassa yleinen uskomus, että ohjelman tai ohjelmiston testaaminen täysin on mahdollista, mutta todellisuudessa täysin testaaminen ei ole mahdollista. Tämä johtuu siitä, että mahdollisia syötteitä on liikaa, ohjelman tai ohjelmiston suorituspolkuja on liikaa tai käyttöliittymässä on liian kompleksisia ongelmia. (Kaner et al., 1999)

2.2 Ohjelmistotestaus manuaalisesti ja automaattisesti

Ohjelmistotestaus voidaan jakaa automaattiseen ja manuaaliseen testaamiseen. Manuaalisessa testaamisessa ihmistestaaja tekeytyy loppukäyttäjäksi ja käyttää ohjelmistoa suorittaen komentoja ja testaten ominaisuuksia varmistaakseen ohjelmiston toimivan odotusten mukaisesti. Varmistaakseen testin valmiuden, testaaja seuraa usein valmiiksi kirjoitettua testisuunnitelmaa, missä on testitapaukset. Automaattinen testaus tarkoittaa sitä, että testaukseen käytetään erillistä ohjelmistoa, joka kontrolloi testien suoritusta ja vertailee testattavan ohjelmiston tuloksia odotettaviin tuloksiin. Automaattisen testauksen ei saa kuitenkaan korvata manuaalista testausta kokonaan. (Garousi and Mäntylä, 2016)

Automatisointia näkee eniten tilanteissa, kuten

- testitapausten suunnittelussa, jossa määritellään lista testitapauksia tai testivaatimuksia kattamaan kattavuus kriteerit.
- testin dokumentoinnissa, jossa luodaan dokumentaatioita testeistä.
- testien suorittamisessa, missä suoritetaan testitapauksia ohjelmistosta ja tallennetaan tulokset
- testien arvioinnissa, missä arvioidaan tulokset joko läpimenneinä tai epäonnistumisina.
- testitulosten raportoinnissa, missä luodaan raportteja testituloksista ja ongelmista, jotka välitetään kehittäjille.

- testien valvonnassa, kuten suunnittelussa, kontrolloinnissa, valvomisessa ja työmäärän arvioinnissa. (Garousi and Mäntylä, 2016)

Tuotteen valmius voidaan määrittää erilaisilla todennus- ja validointiaktiviteeteilla, kuten vaatimusten ja suunnitelmien tarkistamisella ja vertailemalla niitä tuotteeseen. Tämä ei tapahdu vain tietyssä vaiheessa, vaan se levittyy koko ohjelmistokehitysprosessin ajalle. Vaatimukset ja suunnitelmat täytyy kuitenkin tarkistaa ja analysoida mahdollisimman aikaisin, jotta voidaan tunnistaa ja poistaa mahdolliset suunnitteluvirheet, jotka voivat olla vaikeasti löydettäviä ja kalliita korjata myöhemmin. (Baresi ja Pezzè, 2006) Eri vaiheissa ohjelmistokehitysprosessia testien tavoitteet voidaan kohdistaa eri asioihin, kuten poikkeavuuksiin käyttäjän vaatimuksista, raskaan kuormituksen sietokykyyn tai vääriin syötteisiin (Bertolino, 2007).

Todennus- ja validointiaktiviteetit voidaan jakaa viiteen luokkaan: suunnittelu ja valvonta, määritelmien varmistus, testitapausten generointi, testitapausten suorittaminen ja ohjelmiston validointi, sekä prosessin parantaminen. Suunnittelun ja valvonnan aktiviteettien tarkoitus on ohjata prosessia niin, että tuote täyttää alkuperäiset laatuvaatimukset. Määritelmien varmistuksen tavoitteena on löytää ja korjata määritelmiin liittyviä puutteita, sekä korjata kehitysprosessin aikana tapahtuneita poikkeavuuksia alkuperäisistä määritelmistä. Testitapaukset generoidaan yleensä määritelmien perusteella, kuitenkin ympäristön, teknologioiden ja koodikattavuuden puitteissa. Testitapaukset pitäisi luoda heti, kun niihin liittyvä määritelmä on valmis. Prosessin parantamisen tavoitteena on korjata puutteita prosesseissa. Tämä voidaan tehdä esimerkiksi muuttamalla aktiviteetteja tai luomalla uusia laatuun liittyviä aktiviteetteja, jotta ongelmat voidaan eliminoida mahdollisimman aikaisin. (Baresi ja Pezzè, 2006)

2.3 Testauksen automatisointi

Koska testauksen kustannukset ovat puolet ohjelmiston kehittämiskustannuksista, on sen automatisoinnissa selkeitä hyötyjä, sillä se vähentää testauksen kustannuksia ja parantaa testauksen tehokkuutta. Automaattisten testaustyökalujen käyttö on kasvanut hyvin nopeasti viime aikoina. (Anand et al., 2013) Automaattisen testauksen hyödyt ovat laajalti tunnistettu

ja sitä varten on luotu monenlaisia lähestymiskeinoja (Wang et al., 2015). Testiautomatisaation tärkeimmät hyödyt ovat uudelleenkäyttö, toistettavuus ja työmäärän vähentyminen testien suorittamisen suhteen. Sen lisäksi automatisaatio parantaa testien kykyä havaita enemmän ongelmia. Automatisaatio kuitenkin on investointi, mikä tarkoittaa, että se maksaa alussa paljon. (Dudekula Mohammad Rafi et al., 2012)

Automaattisten testityökalujen pitäisi koodikattavuuden sijasta keskittyä seuraaviin viiteen asiaan:

- Niiden pitää olla helposti käytettäviä, mikä pienentää alkuperäistä investointia.
- Niiden pitää käyttää testitapauksia, jotka ovat helposti ylläpidettävissä ja vakaita, mikä helpottaa työkalujen käyttöä huomattavasti.
- Niiden pitää tukea uusien testitapausten luontia tehokkaasti, mikä helpottaa ylläpidossa.
- Niiden pitää olla helposti konfiguroitavissa erilaisiin tarvittaviin kehitysympäristöihin, jotta testityökaluja voi ylipäätään käyttää.
- Ne pitää suunnitella niin, etteivät ne olisi investoinnin puolesta liian riskialttiita, vaan investoinnin suuruutta pitää vähentää, jolloin riskikin pienenee. (Dudekula Mohammad Rafi et al., 2012)

Jos automaattinen testaus tehdään oikein, eli tehokkaasti ja siten, että se oikeasti nopeuttaa testaamista, automaattinen testaus vähentää kustannuksia huomattavasti ja parantaa ohjelmiston laatua. Testauksen automatisoinnin suosio kasvaa, mutta sen kasvaessa päätösten, jotka liittyvät siihen, mitä ja miten automatisoidaan, merkitys kasvaa huomattavasti. Väärät päätökset saattavat johtaa pettymyksiin ja turhiin kuluihin, niin resursseissa, kuin vaivannäössä. (Garousi and Mäntylä, 2016)

3 RATKAISUMENETELMÄT

Tässä luvussa kerrotaan, yleisesti case study -tutkimusmenetelmistä. Tässä luvussa kerrotaan myös, kuinka case study -tutkimusmenetelmiä hyödynnettiin tässä kandidaatintyössä, sekä miten Seepia Gamesin tapaustutkimustarvetta varten luotu testausta automatisoiva työkalu luotiin.

3.1 Tapaustutkimusmenetelmä ja sen soveltaminen työhön

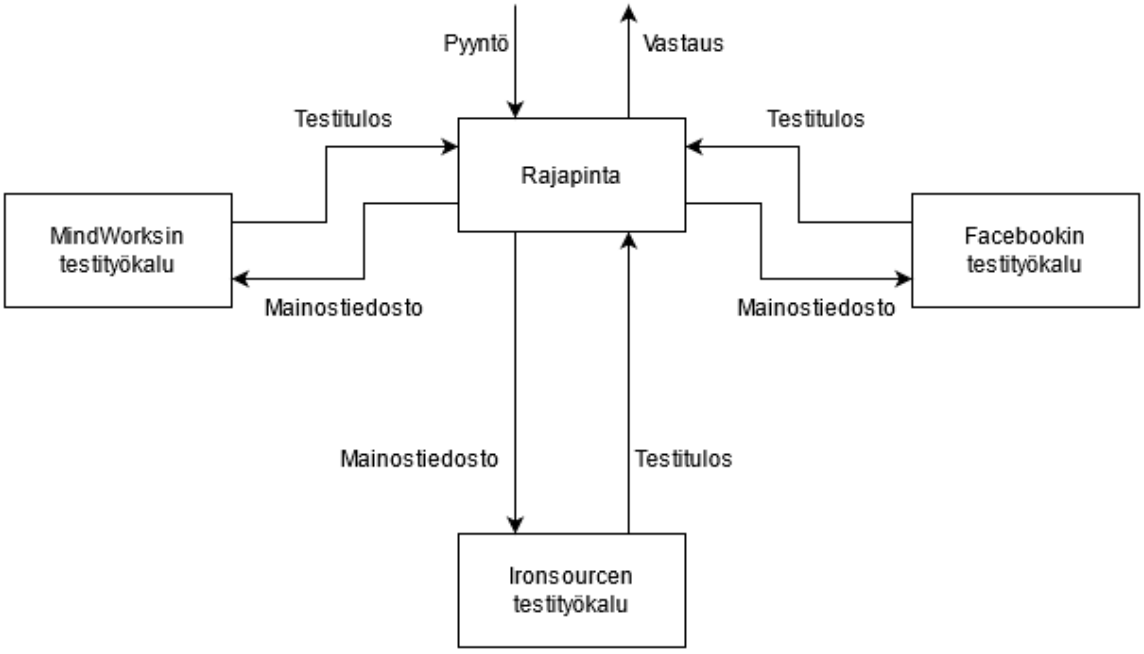
Case study -tutkimusmenetelmät soveltuvat hyvin monenlaisiin ohjelmistotuotannon tutkimuksiin, sillä tutkimuskohteet ovat yleensä nykyaikaisia ilmiöitä (Wohlin et al., 2012). Koska case study -tutkimukset eroavat analyyttisistä ja empiirisistä tutkimusta siten, että case study -tutkimukset eivät aina anna tilastollisia tuloksia, on case study -tutkimuksia hiukan kritisoitukin. Kritiikkiin voi vastata käyttämällä asianmukaisia tutkimusmenetelmiä ja hyväksymällä, että tiedolla ei ole vain tilastollista arvoa. (Wohlin et al., 2012)

Case study -tutkimusmenetelmät kuitenkin määritellään olevan reaalimaailman todellisten ilmiöiden tutkimista, jonka tarkoituksena on tutkia nykyaikaisia ilmiöitä niiden omassa kontekstissaan. Case study -tutkimusmenetelmissä painotetaan sitä, että tuloksille ja todisteille on useampi lähde. (Wohlin et al., 2012) Case study -tutkimusmenetelmät sisältävät osia muista tutkimusmenetelmistä. Esimerkiksi tutkimuksen ohessa voidaan luoda kysely tai haastatteluita, sekä yleensä tutkimukseen liittyy myös kirjallisuuskatsaus. (Wohlin et al., 2012)

Ohjelmistotuotannossa on kyse ohjelmistojen kehittämisestä, operoimisesta ja ylläpitämisestä. Tutkimukset keskittyvät lähinnä tutkimaan, kuinka kehitystä, operointia ja ylläpitämistä toteutetaan ohjelmistotuotannon alalla. Täten monet ohjelmistotuotannon tutkimuksiin soveltuu hyvin case study -tutkimusmenetelmien käyttö. (Wohlin et al., 2012)

Tässä kandidaatintyössä käytetään juuri case study -tutkimusmenetelmiä, koska tämä on juuri tapauskohtainen ja tässä tutkitaan yhtä ohjelmistotuotannon ilmiötä, automatisointia, ja kuinka sitä voi toteuttaa.

3.2 Seepia Gamesin tapaustutkimustarvetta varten luotava työkalu



Kuva 1 Kaavio työkalun kommunikaatiorakenteesta

Tätä tapaustutkimusta varten luodaan työkalu, joka automatisoi mainosverkkojen testityökalujen käytön. Sitä varten ensin tässä työssä kartoitetaan nykyisen toimintamallin tekniset ja prosessimaiset rakenteet ja luodaan näiden pohjalta ensimmäisen toteutussuunnitelman automatisointikokeilulle. Eli selvitetään, lähinnä kokeilemalla, että miten mainosverkkojen testityökalut hyödyntävät linkkejä tai tiedostoja, mitkä ovat testityökalujen tärkeimmät osat ja miten ne kommunikoivat keskenään.

Sitten testataan erilaisia menetelmiä automatisoida ja parhaiksi osoittautuneista menetelmistä valitaan kokonaisuus, joka toimii automatisoinnin työkalun toimintamallin yleissuunnitelmana. Käytännössä mietitään miten ja millä työkaluilla Seepia Gamesiltä saadut linkit tai tiedostot voisi viedä näihin testityökaluihin, saada testityökalut testaamaan ne ja lopuksi palauttamaan jonkinlaisen testituloksen takaisin. Tapaustutkimus yritys esitti, että tämä työ käyttää Selenium-nimistä ohjelmiston testaamisen tehokkuuden parantamiseen luotua työkalua (“SeleniumHQ Browser Automation,” n.d.). Tämän jälkeen kokeillaan suunnitelman toteuttamiskelpoisuutta luomalla alustava toteutus suunnitelmasta. Lopuksi mietitään, kuinka työkalua voitaisiin kehittää eteenpäin.

Aluksi luodaan työkalu, joka palauttaa vain kyllä tai ei vastauksen riippuen siitä, että menikö testit läpi vai ei. Seuraavan vaiheen kehitysaskel työssä oli rakentaa testityökaluun toiminnallisuuksia, joilla työkalusta saadaan enemmän testituloksia ulos. Esimerkiksi mitä kyseisessä testissä onnistui ja mikä osa testistä ei mennyt hyväksytysti läpi, sekä miksi. Työssä mahdollisuuksien mukaan pyritään myös luoda joko mahdollisuus tai vain liittää suoraan palautettavaan tietoon kuva testityökalujen testausraporteista.

Taulukko 1 Suunnitelma toteutettavan työkalun luonnista

Vaihe	**Mitä tehdään**
1) Selvitetään miten mainosverkkojen testityökalut toimivat	Kokeillaan miten testityökalut hyödyntävät linkkejä tai tiedostoja käyttämällä testityökaluja itse manuaalisesti, etsitään testityökalujen tärkeimmät osat sivujen lähdekoodista ja tutkitaan miten ne kommunikoivat keskenään.
2) Mietitään miten testityökalut voitaisiin automatisoida	Tutkitaan mahdollisia web-sivujen automatisointityökaluja etsien tapoja viedä mainosten linkki tai tiedosto testityökalulle testattavaksi. Etsitään myös tapoja saada testityökalu testaamaan mainos ja palauttamaan jonkinlaisen testituloksen.
3) Luodaan alustava työkalu, joka palauttaa kyllä tai ei vastauksen	Kokeillaan suunnitelman toteuttamiskelpoisuutta luomalla alustava toteutus suunnitelmasta. Tämä alustava toteutus palauttaa kyllä tai ei vastauksen riippuen siitä, että menikö testit läpi vai ei.
4) Kehitetään työkalua palauttamaan enemmän tietoa	Kehitetään työkalua niin, että se palauttaa tiedon, että menikö testit läpi vai ei. Jos jotain meni pieleen, niin testituloksessa kerrotaan, että mikä meni pieleen.

5) Mikäli mahdollista, lisätään kuva testituloksista	Mikäli työkalun toteutuksen ohessa on mahdollista, lisätään siihen mahdollisuus palauttaa kuva testituloksista varsinaisen testituloksen ohella.
6) Mietitään mahdollisia jatkokehitys ideoita	Alustavan toteutuksen kehittämisen jälkeen mietitään, miten tätä alustavaa toteutusta voitaisiin kehittää eteenpäin. Kehitettäviä asioita voi olla muun muassa, että mitä palautettavasta tiedosta puuttuu ja miten puuttuva tieto voitaisiin palauttaa tai vaatiiko mainosverkkojen testityökalut mitään erityistä ja muista mainosverkkojen testityökaluista poikkeavia toimenpiteitä.

3.3 Toteutus

Kun selviteltiin miten mainosverkkojen testityökalut toimivat, huomattiin eroja Ironsourcen ja MindWorksin, sekä Facebookin välillä. Ironsourcen testityökalulle riitti mainoksen linkki, kun puolestaan MindWorksin ja Facebookin testityökalut vaativat suoraan mainostiedoston. Kunkin mainosverkon testityökalun lähdekoodista löytyi input-muotoinen syötekenttä, mihin Selenium Webdriverilla pystyttiin suoraan lähettämään mainoksen linkki tai tiedosto.

Automatisointi päätettiin tehdä käyttämällä Selenium Webdriver -työkalua, jolla voi lähettää komentoja koodista sivulle ja täten automatisoida sivu. Selenium Webdriverista juuri löytyi tarvittavat komennot tapaustutkimuksen tarpeisiin, sekä se vaikutti olevan tarpeeksi yksinkertainen tätä kandidaatin työtä varten.

Alustavan työkalun luonti käyttäen Selenium Webdriveria osoittautui oletuksen mukaan varsin yksinkertaiseksi tehtäväksi. Tapaustutkimusta varten luotu työkalu saatiin viemään mainosverkkojen testityökaluihin mainosten linkki tai tiedosto, sinne saatiin vietyä komennot, joilla testityökalut saatiin testaamaan mainokset, sekä testityökalujen testitulokset saatiin parsittua sivuilta ja palautettua halutuissa muodoissa takaisin.

4 TYÖN TULOSHAVAINNOT JA TESTAAMISEN TOTEUTTAVA OHJELMAKOODI

Tässä työssä käytettyjen mainosverkkojen testauksen automatisointi onnistui Seepia Gamesin tavoitteiden kannalta suhteellisen hyvin. Suurimmasta osasta testityökaluista pystyttiin luomaan konseptitodistus, eli idea osoitettiin toteuttamiskelpoiseksi. Applovinin ja Googlen testityökalua ei kuitenkaan tässä kandidaatintyössä automatisoitu ollenkaan. Applovinin testityökalu ei palauttanut suoraan mitään testituloksia, vaan se vain näytti mainostiedoston sivullaan. Googlen testityökalua ei automatisoitu, koska se toimi poikkeavasti muihin testityökaluihin nähden siten, että siitä puuttui yksi olennainen elementti, input-muotoinen syötekenttä, johon mainostiedoston voi liittää, testityökalun sivulta, jota Selenium Webdriver vaati. Tämän takia Googlen testityökalun sivua ei voitu automatisoida käyttäen Selenium Webdriver -työkalua. Sivu olisi vaatinut Windows-käyttöjärjestelmän automatisointia, jonka ei ajateltu olevan olennaista tämän kandidaatin työn kannalta.

Ironsourcen, MindWorksin ja Facebookin testityökalut saatiin automatisoitua ilman suurempia ongelmia. Facebookin testityökalu aiheutti vain pieniä ongelmia, sillä se vaati sisäänkirjautumisen, jotta testityökaluun pääsi käsiksi.

4.1 Ohjelmakoodi

Itse ohjelmakoodia ei tarvinnut kirjoittaa kovinkaan paljoa kutakin testityökalua varten. Ohjelmakoodit kunkin testityökalun kohdalla noudattavat samaa rakennetta, mutta ne silti eroavat hiukan toisistaan kunkin testityökalun kohdalla.

```
3   const ironsource = require("./IronSource_SeleniumWebDriver");
4   const mindworks = require("./MindWorks_SeleniumWebDriver");
5   const facebook = require("./Facebook_SeleniumWebDriver");
6
7   ironsource
8     .ironsource(
9       "https://seepia-review.s3.eu-north-1.amazonaws.com/build-testbed/master/5c358aa4/ironsource/en/2MB/20200518-example2MB-Playable-15s-2MB-Seepia-EN.html"
10    )
11    .then((ironsourceResult) => {
12      console.log(ironsourceResult);
13    });
```

Kuva 2 Testaamista varten luodun pääohjelman ja työkalun funktioiden kutsumisen rakenne

Tähän tapaustutkimukseen on luotu testausta varten pääohjelma, joka käynnistää Selenium Webdriverit kutakin testityökalua varten.

```
1   const {Builder, By, until} = require('selenium-webdriver');
```

Kuva 3 Selenium Webdriverin alustus

```
4   exports.ironsource = (async function ironSourceTest() {
5       let driver = await new Builder().forBrowser('firefox').build();
```

Kuva 4 Selenium Webdriverin alustus Ironsourcen testityökalua varten Firefox-selaimella

Jokaisen mainosverkon testityökalua varten alustettiin Selenium Webdriver kuvien 3 ja 4 osoittamalla tavalla.

```
7       try {
8           // Connect to the page
9           await driver.get('https://demos.ironsrc.com/test-tool/?adUnitLoader=dapi&mode=testing');
```

Kuva 5 Sivulle yhdistäminen

Jokainen testityökalu alkaa samalla tavalla: ensin yhdistetään sivulle kuvan 5 osoittamalla tavalla.

```
11          // Find the input for URL and the button to start testing
12          const adUrl = await driver.findElement(By.id("getAdUrlInput"));
13          const testButton = await driver.findElement(By.id("loadAdUnitBtn"));
```

Kuva 6 Ironsourcen testityökalulta tarvittavat elementit mainostiedoston lähettämiseen

```
15          // Find the input for filepath and send the filepath
16          const adBtn = await driver.findElement(By.className("el-upload__input"));
17          await adBtn.sendKeys(seepiaAdUrl);
```

Kuva 7 MindWorksin testityökalulta tarvittavat elementit mainostiedoston lähettämiseen

Sitten etsitään mainostiedoston lähettämistä varten tarvittavat elementit sivulta. Kuvassa 6 näkyy Ironsourcen testityökalulta tarvittavat elementit. Testityökalua varten piti etsiä elementtien ID:n perusteella tekstikenttä, mihin tulee mainostiedoston osoite, sekä nappi, millä testaus aloitetaan. Kuvassa 7 näkyy MindWorksiltä tarvittavat elementit. Testityökalua varten piti etsiä elementtien luokan perusteella tekstikenttä, mihin syötetään mainostiedoston tiedostopolku.

```
19        // Wait for the page to load the test result
20        await driver.wait(until.urlContains("adUrl="), 10000);
21
22        // Find the test result
23        let checklistItems = await driver.findElements(By.className("checklist-item"));
```

Kuva 8 Ironsourcen testityökalun testitulokset

```
19        // Wait a moment for the tests to run
20        let by = By.className("list-item");
21        await driver.wait(until.elementLocated(by, 10000));
22        await driver.wait(until.elementIsVisible(await driver.findElement(by), 10000));
23        await sleep(10000);
24
25        // Find the results
26        let resultList = await driver.findElements(By.className("list-item"));
```

Kuva 9 MindWorksin testityökalun testitulokset

Tämän jälkeen työkalu odottaa hetken, että testityökalut tekevät testinsä ja etsii sen jälkeen testitulokset sisältävän elementin sivulta. Kuvassa 8 näkyy Ironsourcen testityökalun kohdalla vaadittava odottelu. Ironsourcen kohdalla riitti odottaa, kunnes sivun URL-osoite muuttui, jonka jälkeen haettiin suoraan elementtien luokan perusteella testitulokset. Kuvassa 9 näkyy MindWorksin testityökalulta vaadittavat odottelu. MindWorksin testityökalun sivulla elementti, joka sisältää testityökalut, on jo olemassa, kun sivulle yhdistää ensimmäistä kertaa, mutta se on vain piilossa. Tapaustutkimusta varten rakennettu työkalu odottaa, kunnes tämä elementti tulee näkyviin testausten jälkeen, jolloin se hakee testitulokset elementtien luokan perusteella.

```
40      // Put the results into a JSON string
41      let resultJSONString =
42        '{"validURL":"' +
43        passFail(await checklistItems[0].getAttribute("data-status")) +
44        '","fileSize":"' +
45        passFail(await checklistItems[1].getAttribute("data-status")) +
46        '","isReady":"' +
47        passFail(await checklistItems[2].getAttribute("data-status")) +
48        '","getScreenSize":"' +
49        passFail(await checklistItems[3].getAttribute("data-status")) +
50        '","nativeCloseButton":"' +
51        passFail(await checklistItems[4].getAttribute("data-status")) +
52        '","getAudioVolume":"' +
53        passFail(await checklistItems[5].getAttribute("data-status")) +
54        '","audioVolumeChange":"' +
55        passFail(await checklistItems[6].getAttribute("data-status")) +
56        '","openStoreUrl":"' +
57        passFail(await checklistItems[7].getAttribute("data-status")) +
58        '"}';
59
60      resutlJSONObj = JSON.parse(resultJSONString);
```

Kuva 10 Palautettavan JSON objektin luonti Ironsourcen testityökalussa

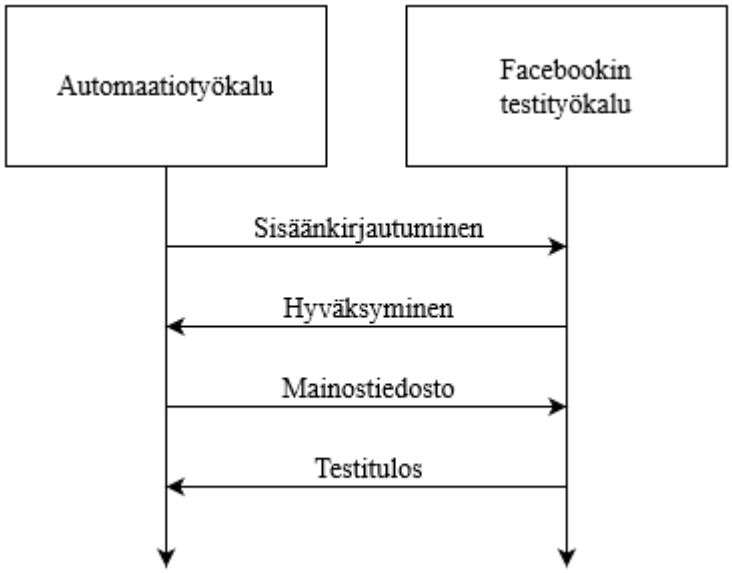
```
43      // Put the results into a JSON string
44      let resultJSONString =
45        '{"HTML":"' +
46        passFail(await resultList[0].getAttribute("class")) +
47        '","CTACall":"' +
48        passFail(await resultList[1].getAttribute("class")) +
49        '","gameEndCall":"' +
50        passFail(await resultList[2].getAttribute("class")) +
51        '","gameReadyCall":"' +
52        passFail(await resultList[3].getAttribute("class")) +
53        '","gameStartCall":"' +
54        passFail(await resultList[4].getAttribute("class")) +
55        '","gameCloseCall":"' +
56        passFail(await resultList[5].getAttribute("class")) +
57        '","fileHandlingMethod":"' +
58        passFail(await resultList[6].getAttribute("class")) +
59        '","fileSpec":"' +
60        passFail(await resultList[7].getAttribute("class")) +
61        '","storageRequirements":"' +
62        passFail(await resultList[8].getAttribute("class")) +
63        '","codeException":"' +
64        passFail(await resultList[9].getAttribute("class")) +
65        '"}';
66
67      resultJSONObj = JSON.parse(resultJSONString);
```

Kuva 11 Palautettavan JSON objektin luonti MindWorksin testityökalussa

Tässä tapaustutkimuksessa on päädytty ratkaisuun, jossa testituloksien löydyttyä, niistä luodaan ensin JSON-muotoa noudattava merkkijono, joka parsitaan JSON-objektiksi, joka palautetaan lopulta pääohjelmaan. Kuvassa 10 näkyy Ironsourcen testityökalua varten luotu toteutus. Kuvassa 11 näkyy MindWorksin testityökalua varten luotu toteutus.

```
17          // Login if needed
18          if (url.includes("%3A%2F%2")) {
19              const loginEmail = await driver.findElement(By.id("email"));
20              const pass = await driver.findElement(By.id("pass"));
21
22              // Check for cookie policy
23              if (await driver.findElement(By.id("u_0_k")).isEnabled()) {
24                  const acceptCookiesBtn = await driver.findElement(By.id("u_0_k"))
25                  await acceptCookiesBtn.click();
26              }
```

Kuva 12 Facebookin sisäänkirjautuminen



Kuva 13 Kommunikaatio Facebookin testityökalun kanssa

Facebookin testityökalu toimii muuten samalla tavalla kuten MindWorksin ja Ironsourcen testityökalut, mutta se vaatii ensin sisäänkirjautumisen. Sisäänkirjautumistiedot annetaan työkalulle erikseen ja sisäänkirjautumisen jälkeen sivu ohjaa testityökalun sivulle, jolloin mainoksen testausta voi jatkaa. Sisäänkirjautumissivu saattoi mahdollisesti vaatia myös evästepolitiikan varmistamista.

```
{
  validURL: 'Unknown',
  fileSize: 'Unknown',
  isReady: 'Fail',
  getScreenSize: 'Fail',
  nativeCloseButton: 'Pass',
  getAudioVolume: 'Fail',
  audioVolumeChange: 'Fail',
  openStoreUrl: 'Unknown'
}
```

Kuva 14 Testitulokset Ironsourcelta konsolissa

```
{
  HTML: 'Pass',
  CTACall: 'Fail',
  gameEndCall: 'Fail',
  gameReadyCall: 'Pass',
  gameStartCall: 'Pass',
  gameCloseCall: 'Pass',
  fileHandlingMethod: 'Pass',
  fileSpec: 'Pass',
  storageRequirements: 'Pass',
  codeException: 'Fail'
}
```

Kuva 15 Testituokset MindWorksiltä konsolissa

Kuvissa 14 ja 15 näkee Ironsourcen ja Mindworksin testitulokset muunnettuna JSON-objekteiksi ja tulostettuna konsoliin. Virhetilanteessa ohjelma palauttaa virheviestin, mikä tässä toteutuksessa myös tulostetaan konsoliin.

5 YHTEENVETO

Seepia Gamesin tapaustutkimustarvetta varten luotu työkalu, joka automatisoi mainosverkkojen testityökaluja onnistui siten, että idea todettiin olevan toteutuskelpoinen, mutta siinä on puutteita. Tässä tutkimuksessa voidaan todeta, että työssä esitelty ratkaisu ei ole parhain mahdollisin. Tämä työkalu tulee vaatimaan paljon ylläpitoa, mikä voi viedä paljon resursseja. Varsinkin, jos mainosverkkojen testityökalut muuttuvat, esimerkiksi sivulla jonkun elementin luokka tai ID muuttuu, ei testityökalu välttämättä toimi enää oikein. Tämä huomattiin useampaan otteeseen jo tutkimuksen aikana, kun muun muassa Facebookin testityökalun sivun elementtien ID:t muuttuivat useamman kerran. Testityökaluja pitää jatkuvasti pitää silmällä juuri näiden muutosten varalta ja tapaustutkimusta varten luotua työkalua pitää päivittää jatkuvasti.

Työkalua olisi voinut parantaa niin, että se ottaa huomioon mahdolliset muutokset elementtien luokissa tai ID:ssä. Testityökalut myös vaativat mainosten klikkailua ja nappien testaamista, jota varten pitäisi kandidaatintyössä luotuun työkaluun lisätä ominaisuus, joka lähettää klikkauskomentoja tiettyihin pikseleihin. Tätä ominaisuutta varten pitäisi mainostiedoston mukana tulla tieto nappien sijainneista pikseleinä. Googlen testityökalu olisi mahdollisesti voitu myös saada toimimaan käyttämällä jotain toista teknologiaa ja automatisoimalla Windows-käyttöjärjestelmää.

LÄHTEET

Ahasanuzzaman, M., Hassan, S., Bezemer, C.-P., Hassan, A.E., 2020. A longitudinal study of popular ad libraries in the Google Play Store. Empir Software Eng 25, 824–858. https://doi.org/10.1007/s10664-019-09766-x

Anand, S., Burke, E.K., Chen, T.Y., Clark, J., Cohen, M.B., Grieskamp, W., Harman, M., Harrold, M.J., McMinn, P., Bertolino, A., Jenny Li, J., Zhu, H., 2013. An orchestrated survey of methodologies for automated software test case generation. Journal of Systems and Software 86, 1978–2001. https://doi.org/10.1016/j.jss.2013.02.061

Baresi, L., Pezzè, M., 2006. An Introduction to Software Testing. Electronic Notes in Theoretical Computer Science, Proceedings of the School of SegraVis Research Training Network on Foundations of Visual Modelling Techniques (FoVMT 2004) 148, 89–111. https://doi.org/10.1016/j.entcs.2005.12.014

Bertolino, A., 2007. Software Testing Research: Achievements, Challenges, Dreams, in: Future of Software Engineering (FOSE '07). Presented at the Future of Software Engineering (FOSE '07), pp. 85–103. https://doi.org/10.1109/FOSE.2007.25

Dehghani, M., Tumer, M., 2015. A research on effectiveness of Facebook advertising on enhancing purchase intention of consumers. Computers in Human Behavior 49, 597–600. https://doi.org/10.1016/j.chb.2015.03.051

Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Petersen, K., Mäntylä, M.V., 2012. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey, in: 2012 7th International Workshop on Automation of Software Test (AST). Presented at the 2012 7th International Workshop on Automation of Software Test (AST), pp. 36–42. https://doi.org/10.1109/IWAST.2012.6228988

Gao, J., Bai, X., Tsai, W.-T., Uehara, T., 2014. Mobile Application Testing: A Tutorial. Computer 47, 46–55. https://doi.org/10.1109/MC.2013.445

Garousi, V., Mäntylä, M.V., 2016. When and what to automate in software testing? A multi-vocal literature review. Information and Software Technology 76, 92–117. https://doi.org/10.1016/j.infsof.2016.04.015

Ikonen, J., Happonen, A., Porras, J. (2007), "Game programming for learning - Experiments in teaching protocols with game programming", In 18th EAEEIE Annual Conference, ISBN: 978-80-01-03745-4, pp. 1-9

Jahkola, O., Happonen, A., Knutas, A., Ikonen, J., 2017. What Should Application Developers Understand about Mobile Phone Position Data, in: Proceedings of the 18th International Conference on Computer Systems and Technologies, CompSysTech'17. Association for Computing Machinery, New York, NY, USA, pp. 171–178. https://doi.org/10.1145/3134302.3134346

Kaner, C., Falk, J., Nguyen, H.Q., 1999. Testing Computer Software

Kortelainen, H., Happonen, A., Hanski, J., 2019. From Asset Provider to Knowledge Company—Transformation in the Digital Era, in: Mathew, J., Lim, C.W., Ma, L., Sands, D., Cholette, M.E., Borghesani, P. (Eds.), Asset Intelligence through Integration and Interoperability and Contemporary Vibration Engineering Technologies, Lecture Notes in Mechanical Engineering. Springer International Publishing, Cham, pp. 333–341. https://doi.org/10.1007/978-3-319-95711-1_33

Lee, J.W., Lee, C.S., Park, Y.S., 2009. Research on the Advertisement Effect of Push Type Mobile Advertisement, in: 2009 Fourth International Conference on Cooperation and Promotion of Information Resources in Science and Technology. Presented at the 2009 Fourth International Conference on Cooperation and Promotion of Information Resources in Science and Technology, pp. 137–142. https://doi.org/10.1109/COINFO.2009.27

Porras, J., Khakurel, J., Ikonen, J., Happonen, A., Knutas, A., Herala, A., Drögehorn, O., 2018. Hackathons in software engineering education: lessons learned from a decade of events, in: Proceedings of the 2nd International Workshop on Software Engineering Education for Millennials, SEEM '18. Association for Computing Machinery, New York, NY, USA, pp. 40–47. https://doi.org/10.1145/3194779.3194783

Schmitt, A., 2017. Why Playable Ads are the Key to Engaged Users. Applift. URL https://applift.com/blog/playable-ads-2 (accessed 7.30.20).

SeleniumHQ Browser Automation [WWW Document], n.d. URL https://www.selenium.dev/ (accessed 4.16.21).

Spinellis, D., 2017. State-of-the-Art Software Testing. IEEE Software 34, 4–6. https://doi.org/10.1109/MS.2017.3571564

Tsai, W.-T., Qi, G., 2017. Introduction, in: Tsai, W.-T., Qi, G. (Eds.), Combinatorial Testing in Cloud Computing, SpringerBriefs in Computer Science. Springer, Singapore, pp. 1–13. https://doi.org/10.1007/978-981-10-4481-6_1

Wang, C., Pastore, F., Goknil, A., Briand, L., Iqbal, Z., 2015. Automatic generation of system test cases from use case specifications, in: Proceedings of the 2015 International Symposium on Software Testing and Analysis, ISSTA 2015. Association for Computing Machinery, New York, NY, USA, pp. 385–396. https://doi.org/10.1145/2771783.2771812

Wohlin, C., Runeson, P., Höst, M., Ohlsson, M.C., Regnell, B., Wesslén, A., 2012. Experimentation in Software Engineering. Springer Science & Business Media.

Wong, C.-H., Tan, G.W.-H., Tan, B.-I., Ooi, K.-B., 2015. Mobile advertising: The changing landscape of the advertising industry. Telematics and Informatics 32, 720–734. https://doi.org/10.1016/j.tele.2015.03.003