Osaamista
ja oivallusta
tulevaisuuden
tekemiseen

Katja Korhonen

Käyttäjätapahtuman jäljitedatan hyödyntäminen ohjelmistotestauksessa

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Hyvinvointi- ja terveysteknologia

Insinöörityö

02.12.2020

Tekijä Otsikko Sivumäärä Aika	Katja Korhonen Käyttäjätapahtuman jäljitedatan hyödyntäminen ohjelmisto-testauksessa 29 sivua 02.12.2020
Tutkinto	insinööri (AMK)
Tutkinto-ohjelma	Tieto- ja viestintätekniikka
Ammatillinen pääaine	Hyvinvointi- ja terveysteknologia
Ohjaajat	Sakari Lukkarinen, Lehtori Osku Sundqvist, Software Business Development Director

Opinnäytetyön aiheena oli käyttäjätapahtuman jäljitedatan hyödyntäminen testiautomaatiossa. Työn pohjana käytettiin Planmeca Oy:n keräämää käyttäjätapahtuman jäljitedataa, joka kerättiin heidän Smile Design -ohjelmistosta 29.8.2019 – 25.11.2019 aikaväliltä. Data piti sisällään lähes 8000 tapahtumajälkeä, ja jokainen tapahtumajälki piti sisällään 30 eri parametria.

Opinnäytetyön päätavoitteena oli parantaa ohjelmistotestauksen kattavuutta analysoimalla Smile Design -ohjelmiston loppukäyttäjien klikkausjäljistä kerätyn käyttäjätapahtuman jäljitedatan. Analysoinnin tarkoituksena oli löytää datasta käyttäjien yleisimmin käytetyt tapahtumasekvenssit, joita voitaisiin käyttää testiautomaatiossa. Analysointimenetelmänä käytettiin puumallia. Tuloksena syntyi 55 yleistä sekvenssiä, joista mikään ei täsmännyt entisten testiajon sekvenssien kanssa. Tästä voitiin päätellä testauksen kattavuuden laajentuneen, koska analyysissa saatiin luotua uusia testisekvenssejä, joita yrityksellä ei ole ollut käytössä.

Työn toisena tavoitteena oli selvittää, miten hyvin käyttäjätapahtuman jäljitedata soveltuu käytettäväksi testiautomaatiossa. Tarkoituksena oli suunnitella testiautomaatioajo, joka käyttäisi data-analysoinnissa saatuja yleisiä tapahtumasekvenssejä Smile Design -ohjelmiston testaamiseen. Tämän suunnitelman pohjalta huomattiin, että kyseisen datan sisältö ei vielä riittänyt kattavan testiautomaatioajon rakentamiselle. Jotta datan avulla olisi voitu testata Smile Design -ohjelmistoa helposti, olisi dataan pitänyt kerätä lisää tietoa painetuista painikkeista.

Avainsanat	Ohjelmistotestaus, data-analytiikka, jäljitedata, tapahtumasekvenssien etsintä

Author	Katja Korhonen
Title	The utilization of User Event Trace data in software testing
Number of Pages	29 pages
Date	02 December 2020
Degree	Bachelor of Engineering
Degree Programme	Computer and Information Technology
Professional Major	Healthcare Technology
Instructors	Sakari Lukkarinen, Senior Lecturer Osku Sundqvist, Software Business Development Director

The topic of the thesis was the utilization of user event trace data in software testing. The work was based on the user event trace data collected by Planmeca Oy, which was collected from their Smile Design software between 29 August 2019 and 25 November 2019. The data contained nearly 8,000 event traces, and each event trace contained 30 different parameters.

The main goal of the thesis was to improve the coverage of software testing by analyzing the user event trace data collected from the click traces of the end users of the Smile Design software. The purpose of the analysis was to find users' most used event sequences that could be used in test automation. A tree model was used as the analysis method. The result was 55 frequent sequences, none of which matched the previous test run sequences. From this, it could be concluded that the coverage of testing had expanded as new test sequences were created in the analysis that the company did not have in use.

The second goal of the work was to find out how well the user event trace data is suitable for use in test automation. The intention was to design a test automation run that would use the frequent event sequences obtained in the data analysis to test the Smile Design software. Based on this plan, it was found that the content of this data was not yet sufficient to build a comprehensive test automation run. To be able to easily test Smile Design software with the data, more information about the pressed buttons should have been collected from the software.

Keywords	Software testing, data-analysis, User Event Trace data, frequent sequence mining

Sisällys

Lyhenteet ja käsitteet

CAD/CAM	*Computer-Aided Design* / *Computer-Aided Manufacturing*. Tietokoneavusteinen valmistus.
CE-merkintä	*Conformité Européenne*. CE-merkintä on merkintä, jolla tuotteen valmistaja tai valtuutettu edustaa vakuuttaa, että tuote täyttää tuotetta koskevien EU:n direktiivien ja asetusten olennaiset vaatimukset.
API	*Application programming interface*. Ohjelmointirajapinta.
XML	*Extensible Markup Language*. XML on merkintäkielien standardi, joka määrittää tietojen merkintämuodon loogisella rakenteella.
Apriori	Apriori on algoritmi yleisten alkiojoukkojen louhintaan (engl. item sets mining) ja assosiaatiosääntöjen oppimiselle. Se käyttää ”alhaalta ylös” -lähestymistapaa, jossa yleisiä alajoukkoja (engl. subsets) laajennetaan yksi kerrallaan (vaihe, jota kutsutaan kandidaatin generoinniksi), ja kandidaattiryhmiä testataan datan perusteella. Algoritmi päättyy, kun muita onnistuneita laajennuksia ei löydy.
FP-Growth	*Frequent Pattern Growth*. FP-Growth on algoritmi yleisten alkiojoukkojen louhintaan. Yleiset alkiojoukot generoidaan ilman kandidaatin generointi -vaihdetta. Algoritmi esittelee tietokannan puumallin avulla, jota kutsutaan FP-puuksi (frequent pattern tree).

1 Johdanto

Insinöörityön tilaajana on Planmeca Oy ja työn päätavoitteena on parantaa ohjelmistotestauksen kattavuutta, analysoimalla Smile Design -ohjelmiston loppukäyttäjien klikkausjäljistä kerätyn käyttäjätapahtuman jäljitedata. Datan analysoinnin tarkoituksena on löytää datasta käyttäjien yleisimmin käytetyt tapahtumasekvenssit, joita voidaan käyttää testiautomaatiossa. Tällä hetkellä suuri osa ohjelmistotestauksesta suoritetaan seuraten testaajien ja ohjelmistokehittäjien suunnittelemia testiajoja. Nämä testiajot saattavat kuitenkin erota hyvinkin paljon loppukäyttäjien toimintatavoista, koska testit on suunniteltu yleensä tukemaan ohjelman loogista työnkulkua, josta loppukäyttäjällä ei ole välttämättä sama näkemys. Tämän seurauksena loppukäyttäjille saattaa jäädä vakaviakin vikoja löydettäväksi, koska toiminnallisuuksia ei testattu juuri siinä järjestyksessä, missä loppukäyttäjät ne yleensä tekevät tai niillä arvoilla, joita he yleensä syöttävät ohjelmaan.

Työn toisena tavoitteena on selvittää, miten hyvin käyttäjätapahtuman jäljitedata soveltuu käytettäväksi testiautomaatiossa. Tarkoituksena on suunnitella testiautomaatioajo, joka käyttää data-analysoinnissa saatuja yleisiä tapahtumasekvenssejä Smile Design ohjelmiston testaamiseen. Suunnitelman pohjalta analysoidaan, kuinka hyvin testiautomaation rakentaminen onnistuisi datan pohjalta, ja miten käyttäjätapahtuman jäljitedataa tulisi kehittää, jotta siitä saataisiin kaikista suurin hyöty testiautomaatioon.

2 Työn tilaaja

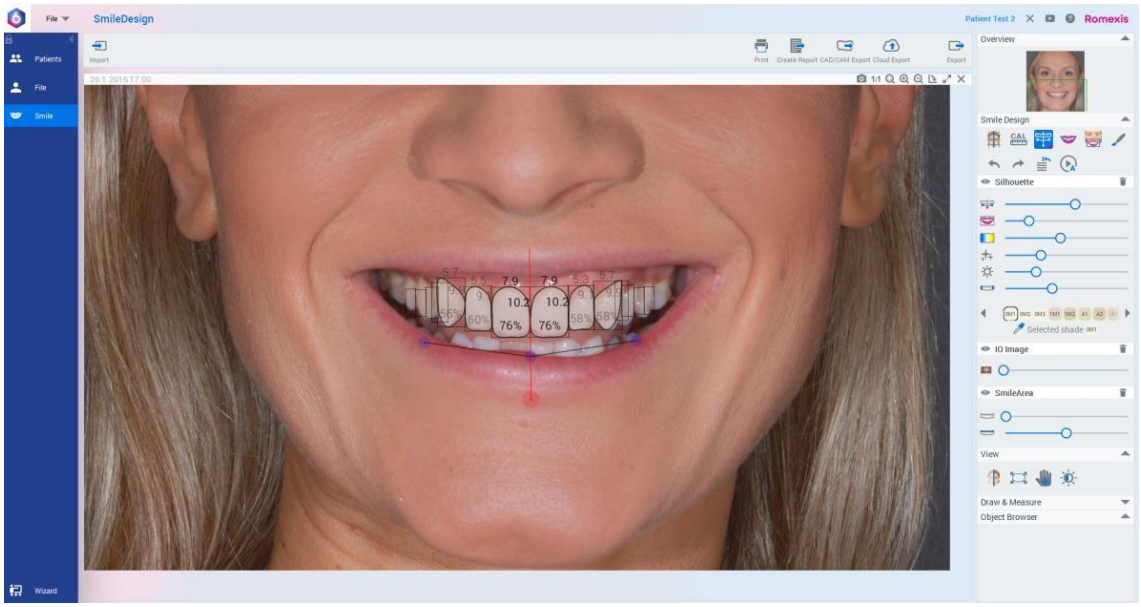
Työn tilaajana on Planmeca Oy. Planmeca on yksi maailman johtavista hammashoitolaitteiden valmistajista, jonka tuotteita viedään yli 120 maahan. Tuotteisiin kuuluvat digitaaliset hammashoitokoneet, 2D- ja 3D-röntgenlaitteet, CAD/CAM-tuotteet sekä niitä tukevat ohjelmistoratkaisut. [1.]

Yhtiö on perustettu Helsingissä vuonna 1971, jolloin yritystoiminta käynnistyi ensiksi hammashoitotuolien ja instrumenttikaapistojen valmistajana. Tuotteiden valmistus tapahtui kulosaarelaisessa autotallissa. Vaikka yhtiö ja tuoteperhe ovat laajentuneet tästä moninkertaiseksi, Planmecan hoitokoneet, kuvantamislaitteet ja ohjelmistot suunnitellaan sekä valmistetaan edelleen Helsingissä. Kehityksen myötä syntyi kuitenkin monia tytäryhtiöitä, minkä seurauk-sena syntyi Planmeca Group, joka toimii myös terveysteknologian alalla. Planmeca Group -konserniin kuuluu Planmecan lisäksi kuusi yhtiötä, jotka ovat Planmed, Plandent, LM-Instruments, Opus Systemer, Triangle Funiture Systems ja Nordic Institute of Dental Education. [2.]

Insinöörityö tehdään Planmeca-yhtiön Röntgen-divisioonan ohjelmistotestaustiimille. Ohjelmistotestauksessa testataan 2D- ja 3D-röntgenlaitteita ja niitä tukevia ohjelmistoratkaisuita. Yksi näistä ohjelmistoratkaisuista on Planmeca Romexis Smile Design, jonka testausta on tässä työssä tarkoitus kehittää.

Smile Design on Planmecan kehittämä ohjelma hymyjen mallinnukseen, kommunikaatioon ja hoitojen suunnitteluun. Ohjelma on osa CE-merkittyä Planmeca Romexis -hammaslääketieteellistä kuvantamisohjelmistoa, mutta sen saa ladattua myös erikseen omana ohjelmana.

Smile Design -ohjelmisto koostuu kolmesta moduulista, jotka ovat: Patients, File ja Smile. Kuvassa 1 on esitettynä Smile Design -ohjelman käyttöliittymä, jossa esitetään hymyn mallinnusta. [3.]



Kuva 1. Smile Design -ohjelmiston Smile-moduulin käyttöliittymä

Ohjelmassa käyttäjä pystyy mallintamaan hymyjä lisäämällä uuden hammassiluetin (katso kuva 1). Tätä hammassiluettia muokataan sopimaan potilaan omiin hampaisiin. Siluettia on mahdollista manipuloida muun muassa vaihtamalla siluetin väriä, läpinäkyvyyttä ja varjostusta sekä muuntamalla yksittäisen hampaan kokoa ja muotoa. Näiden lisäksi kuvaan on mahdollista lisätä erilaisia annotaatioita, kuten mittauksia, implantteja ja kommentteja. Kun hymyn mallinnus on valmis, kuvan voi tallentaa tai tulostaa paperille esimerkiksi potilaan nähtäväksi. Ohjelmisto auttaa siis potilaita ymmärtämään paremmin, miltä uusi hymy näyttäisi, ennen sitoutumista operaatioon. [3; 4.]

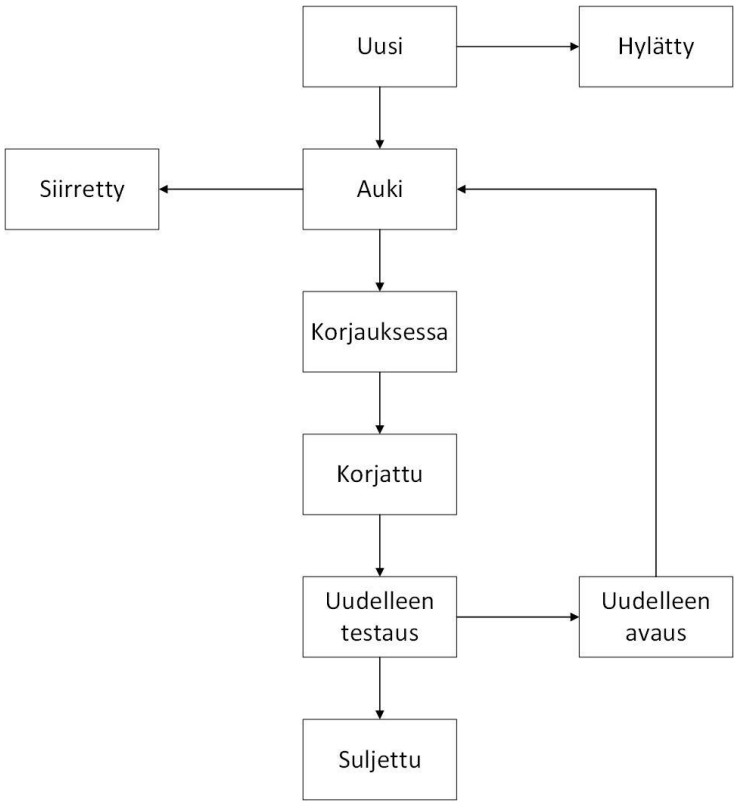
Hymysuunnitelman jälkeen uusi hymy voidaan jakaa myös pilvitiedostonsiirtopalvelun tai mRomexis-mobiilisovelluksen kautta muiden nähtäväksi. Integrointi CAD/CAM- ja Ortho-ratkaisuihin mahdollistaa hammaslääkäreille hymysuunnitelmien viennin myös mihin tahansa CAD/CAM-ohjelmistoon toteutusta varten, jolloin suunnitelmaa voidaan tarkastella kolmiulot-teisesti. Smile Design -ohjelmassa suunniteltu 2D-muotoilu toimii tällöin visuaalisena ohjeena tai viitteenä fyysisen hammasmallin suunnittelulle sekä lopulliseen restaurointiin. [4.]

3 Ohjelmistotestaus

Ohjelmistotestaus on ohjelmistojen suoritussarjojen suorittamista yleensä ohjelmiston lähdekoodin kehittämisen jälkeen. Ohjelmistotestaus suoritetaan mahdollisimman monien vikojen löytämiseksi ja korjaamiseksi ennen kuin ohjelmisto toimitetaan asiakkaalle. [5.]

Vika on ohjelmiston virhe, joka rajoittaa ohjelmiston normaalia kulkua. Tällöin ohjelmiston odotettu käyttäytyminen ei vastaa sen todellista käyttäytymistä. Vika ilmenee, kun kehittäjä on tehnyt virheen ohjelmiston suunnittelussa tai rakentamisessa, ja kun testaaja havaitsee tämän virheen, sitä kutsutaan viaksi. [6.]

Kun testaaja havaitsee vian ohjelmistossa, hän kirjaa sen järjestelmään. Tästä alkaa vian elinkaari, joka on esitetty kuvassa 2.



Kuva 2. Vian elinkaari [6].

Kun testaaja on kirjannut uuden vian järjestelmään, se validoidaan. Vika joko todetaan aiheettomaksi, jolloin se hylätään tai aiheelliseksi, jolloin se siirretään auki-tilaan. Auki-tilassa vikaa tarkastellaan tarkemmin. Vika saatetaan todeta kiireettömäksi, jolloin se voidaan siirtää esimerkiksi myöhempään ohjelmistoversioon. Kun vika on todettu aiheelliseksi ja ajankohtaiseksi, kehittäjä ottaa sen korjattavaksi. Korjauksen jälkeen vika on valmis testattavaksi uudelleen. Jos sama vika toistuu uudelleen testauksen aikana, vika avataan uudelleen, jolloin se päätyy taas kehittäjän korjattavaksi. Kun vika ei enää toistu, vika suljetaan. [6.]

Ohjelmistotestauksessa testaajat käyttävät yleensä kahta eri dynaamista analyysiä vikojen havaitsemiseen: manuaalista ja automaattista testausta. Manuaalitestaus perustuu testaajan läsnäoloon. Testaaja luo ja suorittaa käsin skenaarioita, jotka aiheuttavat joko tehtävän epäonnistumisen tai suorittamisen. Automaatiotestauksessa testaaja puolestaan ohjelmoi testikoodin sovelluksen testaamiseksi. [7.]

3.1 Manuaalitestaus

Manuaalitestaus jakautuu kahteen ääripäähän: skriptitestaamiseen ja tutkivaan testaamiseen. Skriptitestauksessa manuaalisia testaajia ohjaavat testiskriptit, jotka on kirjoitettu etukäteen. Skriptit ohjaavat syötteen valintaa ja määräävät, kuinka ohjelmiston tulosten oikeellisuus on tarkistettava. Joskus ne ovat spesifisiä: anna tämä arvo, paina tätä painiketta, tarkista tulos ja niin edelleen. Jokaisen testivaiheen kohdalla on erikseen myös ilmoitettu, mitä syötteen jälkeen kuuluu tapahtua. Jos testivaiheena on painaa painiketta x, niin odotettuna tapahtumana voi olla kirjattu esimerkiksi dialogin y avautuminen. Tällaiset testiskriptit dokumentoidaan usein taulukkomuodossa, ja ne vaativat jatkuvaa ylläpitoa, koska ohjelmiston ominaisuuksia päivitetään joko uuden kehitys- tai virhekorjauksen avulla. Skriptien avulla myös dokumentoidaan suoritettu testaus. [7.]

Kun testiskriptit poistetaan kokonaan, prosessia kutsutaan tutkivaksi testaamiseksi. Tällöin testaajat voivat olla vuorovaikutuksessa sovelluksen kanssa haluamallaan tavalla ja käyttää sovelluksen tarjoamia tietoja reagoidakseen, muuttaakseen kurssia ja yleensä tutkiakseen sovelluksen toimintoja ilman rajoituksia. Joillekin se voi tuntua, ettei prosessi tuota tarpeeksi tulosta, mutta ammattitaitoisen ja kokeneen tutkijan käsissä tämä tekniikka voi osoittautua tehokkaaksi. Tutkimusmenetelmiä käyttävät testaajat eivät

kuitenkaan ole ilman dokumentointipolkua. Testitulokset, testitapaukset ja testidokumentaatio luodaan testien suorittamisen aikana sen sijaan, että ne dokumentoitaisiin etukäteen testisuunnitelmassa. Näytön kuvankaappaaminen ja näppäinpainalluksen tallennustyökalut ovat ihanteellisia tutkivan testauksen tulosten tallentamiseen. [7.]

Tutkiva testaus sopii erityisen hyvin moderniin verkkosovellusten kehittämiseen ketterillä menetelmillä. Kehitysjaksot ovat lyhyitä, joten muodollisten komentosarjojen kirjoittamiseen ja ylläpitoon jää vähän aikaa. Ohjelmistojen ominaisuudet kehittyvät usein nopeasti, joten muiden asioiden - kuten esimerkiksi valmiiksi valmistetut testitapaukset - minimointi on toivottavaa. Tutkivassa testauksessa on haittapuoliakin. Testaajat voivat kuluttaa valtavan määrän aikaa vaeltelemalla sovelluksessa etsien testattavia asioita ja yrittäen löytää virheitä. Valmistelun, rakenteen ja ohjauksen puute voivat johtaa moniin tuloksettomiin tunteihin ja saman toiminnallisuuden testaamiseen uudestaan ja uudestaan, etenkin kun mukana on useita testaajia tai testiryhmiä. [7.]

Tutkivaa testaamista ei ole kuitenkaan välttämätöntä tarkastella tiukkana vaihtoehtona skriptipohjaiselle manuaaliselle testaukselle. Skriptitestaus voi tarjota hyvän rakenteen tutkivaan testaukseen, ja tutkivat menetelmät voivat lisätä komentosarjoihin variaatioelementin, joka voi parantaa niiden tehokkuutta. Testaaja voi siis saada erittäin tehokkaan testausmenetelmän yhdistämällä nämä kaksi toisistaan hyvin erilaista testausmenetelmää. [7.]

3.2 Automaatiotestaus

Automaatiotestauksessa testit ovat ohjelmointikoodia, jotka automaatiotestaajat ohjelmoivat. Yleisimpiä ohjelmointikieliä testiautomaatiossa ovat muun muassa Java ja Python. Testiautomaatio voi automatisoida joitain toistuvia, mutta välttämättömiä tehtäviä jo käytössä olevassa virallisessa testausprosessissa, tai suorittaa lisätestauksia, joita olisi vaikea tehdä manuaalisesti. Testiautomaatio on kriittisen tärkeää jatkuvalle toimitukselle ja jatkuvalle testaukselle. [8.]

Testiautomaatioon on monia lähestymistapoja, mutta yleisesti käytettyjä tapoja ovat graafinen käyttöliittymätestaus ja API-ohjattu testaus. Graafisessa käyttöliittymätestauksessa tuotetaan käyttöliittymätapahtumia, kuten näppäinpainalluksia ja hiiren

napsautuksia, ja tarkkaillaan muutoksia, jotka johtavat käyttöliittymään. Näin vahvistetaan, että ohjelman havaittavissa oleva toiminta on oikea. API-ohjatussa testauksessa käytetään puolestaan sovelluksen ohjelmointirajapintaa testattavan käytön validoimiseksi. Tyypillisesti API-ohjattu testaus ohittaa sovelluksen käyttöliittymän kokonaan. [8.]

Useimmissa ohjelmistokehitysmalleissa koodaus-testaus-korjaus-silmukka voi toistua useita kertoja ennen ohjelmiston julkaisua. Jos testataan tiettyä ominaisuutta, se ei välttämättä tarkoita, että testit suoritetaan yhden kerran, vaan mahdollisesti kymmeniä kertoja. Näin varmistetaan, että aiemmissa koeajoissa löydetyt virheet todellakin on korjattu ja ettei uusia virheitä löydy. Tätä testien uusintaprosessia kutsutaan regressiotestaukseksi. Jos pienessä ohjelmistoprojektissa on useita tuhansia testitapauksia suoritettavana, ei testejä välttämättä ehditä testaamaan kuin kerran. Niiden suorittaminen useita kertoja voi olla mahdotonta sekä hyvin yksitoikkoista. Ohjelmistotestityökalut ja automaatio voivat auttaa ratkaisemaan tämän ongelman tarjoamalla tehokkaampia tapoja suorittaa testit kuin manuaalitestauksessa. [8.]

4 Datan analysointi

Datan analysointi eli data-analytiikka on luokka tilastollisia menetelmiä, joiden avulla on mahdollista käsitellä suuria tietomääriä, ja tunnistaa sen rakenteen mielenkiintoisimmat näkökohdat. Jotkut data-analytiikan menetelmät auttavat poimimaan suhteita eri tietojoukkojen välillä ja auttavat siten saamaa tilastotietoja, joiden avulla on mahdollista kuvata tietoihin sisältyvä tärkein tieto mahdollisimman tiiviisti. Muut menetelmät antavat mahdollisuuden ryhmitellä tietoja, jotta niiden yhteiset nimittäjät voidaan yksilöidä selvästi ja ymmärtää paremmin. [9.]

4.1 Data

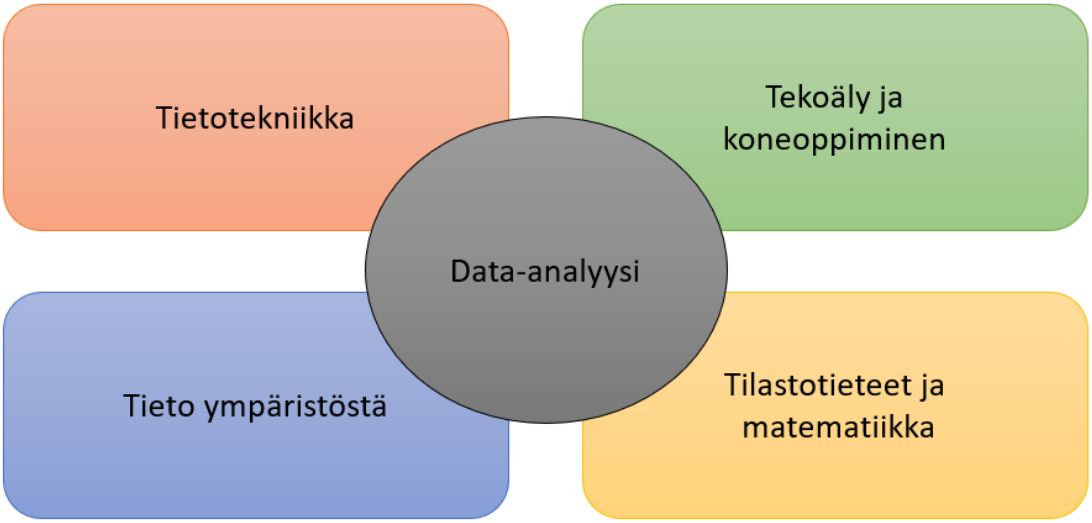
Datakäsite tulee englannin kielestä ja tarkoittaa tietoja. Tiedoilla tarkoitetaan erityisesti tosiseikkoja tai numeroita, jotka on kerätty tutkittavaksi ja tarkasteltaviksi sekä käytetty päätöksenteon helpottamiseksi, tai tietoja sähköisessä muodossa, jota tietokone voi tallentaa ja käyttää. [10.]

Data voidaan jakaa kahteen pääluokkaan: jäsenneltyyn tai jäsentelemättömään. Pääpaino data-analytiikassa on kuitenkin jäsennellyssä datassa, joka kattaa monia erilaisia yleisiä datamuotoja, kuten moniulotteiset taulukot (matriisit), perustaulukot, joissa kukin sarake voi olla erityyppinen, kuten merkkijono, numeerinen tai päivämäärä ja tasaisesti tai epätasaisesti sijoitetut aikasarjat. Useimmat datatyypit kuuluvat kuitenkin taulukoihin, joita yleensä tallennetaan relaatiotietokantoihin tai pilkuilla erotettuihin tekstitiedostoihin. [11.]

Data voi olla myös numeerista tai kategorista. Kategorinen data on arvoja tai havaintoja, jotka voidaan lajitella ryhmiin tai luokkiin. Se voidaan jakaa nominaalisiin ja ordinaalisiin muuttujiin. Nominaalisella muuttujalla ei ole luontaista järjestystä sen luokissa. Tästä esimerkkinä on kategorinen muuttuja: asunto, jolla on kaksi luokkaa (oma ja vuokra). Ordinaalisella muuttujalla on taas vakiintunut järjestys. Ordinaalinen muuttuja voi olla esimerkiksi ikä, jolla on kolme järjestettyä luokkaa (nuori, aikuinen ja vanhus). Numeerinen data on puolestaan arvoja tai havaintoja, joita voidaan mitata. Nekin voidaan jakaa kahteen eri osaan, diskreettiin ja jatkuviin. Diskreettidata on arvoja tai havaintoja, jotka ovat selkeitä ja erillisiä, esimerkiksi koodin rivien lukumäärä. Jatkuvadata on taas arvoja tai havaintoja, jotka voivat ottaa minkä tahansa arvon rajallisella tai äärettömällä arvojen vaihteluvälillä, esimerkiksi taloudellinen aikasarja kullan hinnasta 2000-luvulla. [11.]

4.2 Data-analytiikan suhde muihin tieteenaloihin

Datan analysoinnissa raakadataa muokataan käytettäväksi menetelmissä, jotka auttavat selittämään menneisyyttä ja ennustamaan tulevaisuutta. Data-analytiikka ei koske numeroita, vaan se on kysymysten tekemistä/esittämistä, selitysten kehittämistä ja hypoteesin testaamista. Se on monitieteellinen ala, jossa yhdistyvät tietotekniikka, tekoäly ja koneoppiminen, tilastotieteet ja matematiikka sekä tieto ympäristöstä, jossa kohdejärjestelmä toimii. Tämä on havainnollistettuna kuvassa 3.



Kuva 3. Data-analytiikan suhde muihin tieteenaloihin

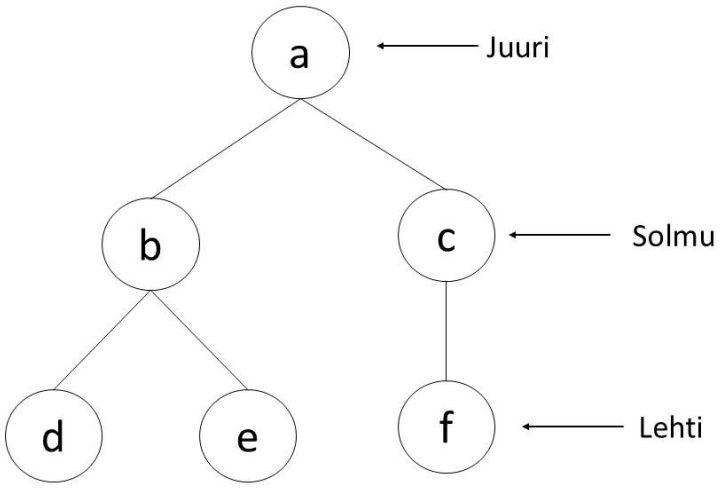
Tietotekniikka luo välineet datan analysointiin. Tietoteknisiä taitoja tarvitaan muun muassa ohjelmointiin, tietokannan hallintaan tai korkean suorituskyvyn laskentaan. Koneoppiminen on datasta oppimista ja johtopäätösten tekemistä. Koneoppimisessa opiskellaan tietokonealgoritmeja, joilla pyritään reagoimaan tietyssä tilanteessa tai tunnistamaan datasta malleja. Tekoälyä käytetään datan analysoinnissa suorittamaan älykkyyttä vaativia toimintoja, kuten päätelmiä, samankaltaisuuden etsimistä tai luokittelua. Tilastotieteet ovat tietojen keräämis-, analysointi- ja tulkintamenetelmien kehittämistä ja soveltamista. Matematiikasta hyödynnetään eniten numeerisia menetelmiä, lineaarista algebraa, kuten vektori- ja matriisilaskentaa sekä tekijöihin jakamista, ja ehdollista todennäköisyyttä. Viimeisenä jäljellä on tieto ympäristöstä -osa-alue, joka antaa tarvittavan asiantuntemuksen ja intuition kysymään hyviä kysymyksiä, mikä on yksi data-analyysin tärkeimmistä tehtävistä. [12.]

Datan analysointia käytetään melkein kaikilla aloilla, kuten taloudessa, liiketoiminnassa, sosiaalisessa mediassa ja tieteessä. Sitä voidaan tehdä eri alustoja ja ohjelmointikieliä käyttäen. Yksi data-analysoinnin yleisimmistä ohjelmointikielistä on kuitenkin Python.

4.3 Yleisten puumallien louhinta

Monet datan louhintaongelmat voidaan parhaiten esitellä epälineaaristen tietorakenteiden avulla. Esimerkiksi kaavioita käytetään yleisesti edustamaan tietoja ja niiden suhteita eri ongelma-alueilla, aina web-louhinnasta ja XML-dokumenttien louhinnasta bioinformatiikkaan ja sosiaalisiin verkkoihin. Erityisesti puut ovat herättäneet huomiota muun muassa siksi, että ne ovat erityisen käyttökelpoisia tehokkaiden kuviolouhintatekniikoiden (engl. pattern mining) kanssa. [13.]

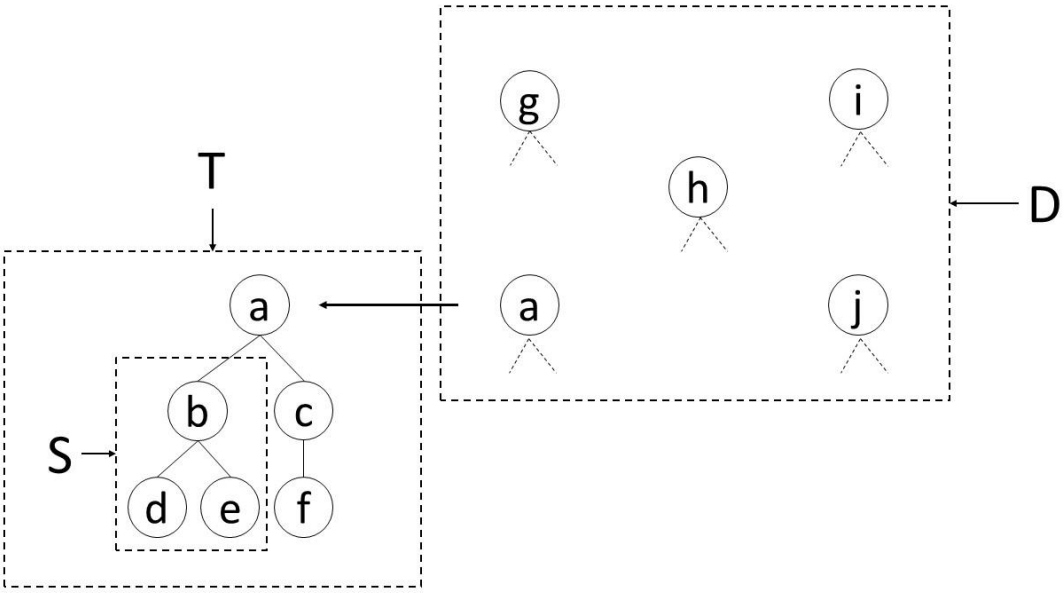
Puu on kytketty asyklinen kuvaaja, joka kuvaa suhteita useiden muuttujien välillä. Puu koostuu juuresta, solmuista ja lehdistä (kts. kuva 4). Puun lehdeksi kutsutaan solmua, jolla ei ole enää alisolmua. Puun koko määritetään solmujen lukumäärästä. [13.]



Kuva 4. Puun rakenne.

Puumallilouhinnan tavoitteena on löytää kaikki usein esiintyvät alapuut yksilöllisestä suuresta puusta T tai suuresta puiden D tietokannasta, jota kutsutaan myös metsäksi (katso kuva 5). Alapuun S esiintyvyys määritetään tukiparametrilla. Alapuun tuki tarkoittaa puiden lukumäärää D:ssä, joihin sisältyy ainakin yksi osapuun S esiintyminen. Vastaavasti alapuun painotettu tuki kuvaa S:n esiintymien kokonaismäärää kaikissa D:n puissa. Alapuu S esiintyy siis usein, jos sen tuki on suurempi tai yhtä suuri kuin ennalta määritetty

minimitukiraja. Usein esiintyvää puuta T kutsutaan puolestaan maksimiksi, jos T ei ole toisen D:ssä olevan puun alaosa. Kun taas T on suljettu, se ei ole toisen toistuvan puun alapuu D:ssä, jolla on täsmälleen sama tukikerroin. [13.]



Kuva 5. Metsän D, puun T ja alapuun S suhde toisiinsa.

Monet yleisien puumallien louhinta (engl. frequent tree pattern mining) -algoritmit johdetaan yleensä yleisimmin käytetyistä kuvion louhinta -algoritmeista, jotka ovat Apriori tai FP-Growth. Näitä tunnettuja algoritmeja käytetään yleisesti tunnistamaan toistuvia malleja tai kuvioita (engl. patterns) eri tietokannoista, ja nyt niitä on laajennettu puumallin louhintaan. Useimmat puumallien louhinta-algoritmit kuitenkin seuraavat Apriorin iteratiivista kuvioiden louhintastrategiaa, jolloin iterointi jaetaan kahteen erilliseen vaiheeseen: kandidaatin generointiin ja tukilaskentaan. Kandidaatin generoinnissa mahdolliset usein toistuvat alapuuehdokkaat luodaan edellisessä iteraatiossa löydetyistä usein toistuvista malleista. Useimmat Apriori tyyppiset algoritmit tuottavat k + 1-kokoisia ehdokkaita yhdistämällä kaksi k:n kokoista puuta, joilla on k - 1 määrä yhteisiä elementtejä. Tukilaskentavaihe koostuu mahdollisten usein esiintyvien ehdokkaiden todellisen tuen määrittämisestä D:ssä ja vain niiden ehdokkaiden pitämisestä, jotka todellisuudessa esiintyvät usein (edellisessä vaiheessa luotujen ehdokkaiden joukko). [13.]

Muut puumallin louhinta-algoritmit seuraavat FP-Growth-mallien lähestymistapaa, jossa ei varsinaisesti generoida kandidaatteja, koska se voi olla hyvin aikaa vievää [13; 14]. Tämän sijasta algoritmit käyttävät esimerkiksi tiivistettyjä rakenteita ohjelmoimaan sisääntulotietoja tai poimivat yleiset alapuut etsimällä yleisiä alasekvenssejä. [13.]

5 Käyttäjätapahtuman jäljitedata

Työssä käytettävä käyttäjätapahtuman jäljitedata koostuu Smile Design -ohjelman käyttäjien tapahtumajäljistä eli klikkauksista. Näiden tapahtumajälkien perusteella saadaan selkeä kuva siitä, mitä käyttäjä on ohjelmistossa tehnyt. Yksi tapahtumajälki kertoo muun muassa sen, mitä painiketta on painettu sekä minä ajankohtana, missä maassa, istunnossa ja ohjelmaversiossa tämä kyseinen klikkaus on tapahtunut. Smile Design -ohjelmiston jäljitedata kerättiin 29.8.2019 - 25.11.2019 aikavälillä. Käyttäjät, joista dataa kerättiin, jouduttiin myös rajaamaan vain yhdysvaltalaisiin, koska jäljitedatan painikenimet kerättiin vielä kyseisessä ohjelmistoversiossa sillä kielellä, minkä käyttäjä oli valinnut ohjelmistoon. Ilman rajausta olisi ollut erittäin työlästä verrata samoja painikkeita, koska saman painikkeen nimi vaihtelisi ohjelmiston käyttöliittymäkielen mukaan. Dataan saatiin kuitenkin kerättyä lähes 8000 tapahtumajälkeä, johon sisältyy 93 eri tapahtumaa ja 39 eri istuntoa. Yksi tapahtumajälki koostuu puolestaan 30 eri parametrista.

Ote datasta ja sen tärkeimmistä parametreista insinöörityön kannalta on esitetty taulukossa 1. Timestamp tarkoittaa aikaleimaa, joka kertoo, minä ajankohtana kyseinen tapahtuma on tapahtunut. Name-parametri kertoo, mikä tapahtuma on ollut kyseessä, ja mitä arvoja on käytetty tapahtuman kanssa. Esimerkiksi Name-parametrin viimeisellä rivillä (katso taulukko 1) olevassa tapahtumassa Simulated Texture transparency -tapahtuman liukusäädintä on siirretty arvoon 27. Session_Id-parametri ilmoittaa, missä istunnossa tapahtuma on tapahtunut. Jokaisella istunnolla on uniikki identifikaationumerosarja, jonka avulla istunnot voidaan erottaa toisistaan.

Taulukko 1. Ote datasta ja sen tärkeimmistä parametreista

timestamp	name	session_Id
2019-11-25T14:22:37.463Z	Romexis closed	5052129429058330000
2019-11-25T18:21:13.078Z	Patient opened	1722268588510770000
2019-11-25T18:21:16.681Z	Start Smile Design	1722268588510770000
2019-11-25T18:21:20.605Z	Face image	1722268588510770000
2019-11-25T18:21:21.846Z	Face image	1722268588510770000
2019-11-25T18:21:24.361Z	Simulated Texture transparency slider moved to 27	1722268588510770000

Taulukossa 1 mainittujen parametrien lisäksi data pitää sisällään muun muassa käyttäjän ohjelmistoversion, yksilöllisen tapahtuma identifikaationumeron ja missä maassa käyttäjä sijaitsee. Näillä tiedoilla ei kuitenkaan tässä työssä ollut käyttöä.

6 Yleisten tapahtumasekvenssien etsiminen

Tässä luvussa käydään läpi työn päätavoitetta. Tavoite oli parantaa ohjelmistotestauksen kattavuutta analysoimalla Smile Design -ohjelmiston loppukäyttäjien klikkausjäljistä kerätyn jäljitedatan. Kappale pitää sisällään työssä käytetyt työkalut, datan esikäsittelyn, puumallianalyysin, yleisten sekvenssien generoinnin ja viimeisenä tulokset.

6.1 Työssä käytetyt työkalut

Yleisten tapahtumasekvenssien etsiminen ohjelmoitiin Python-ohjelmointikielellä Jupyter Notebook -alustaa käyttäen. Jupyter Notebook on avoimen lähdekoodin verkkosovellus, jonka avulla voidaan luoda ja jakaa dokumentteja, jotka sisältävät live-koodia, yhtälöitä, lukuja, interaktiivisia sovelluksia ja Markdown-tekstiä. Yleisin kieli Jupyterissä on Python, mutta myös muita kieliä tuetaan. [18; 19.]

Työssä käytettiin Pythonin NumPy- ja Pandas-kirjastoja, jotka ovat hyvin yleisiä data-analytiikan kirjastoja. NumPy eli Numerical Python -kirjaston avulla pystytään tallentamaan ja manipuloimaan moniulotteisia dataryhmiä tehokkaasti. Pandas-kirjasto puolestaan nopeuttaa ja helpottaa jäsennellyn datan käsittelemistä monipuolisilla datarakenteilla ja toiminnoilla. Työn kannalta ensisijaisin ominaisuus Pandas-kirjastosta oli datakehys (engl. DataFrame), joka on kaksiulotteinen taulukkomuotoinen tietorakenne, jossa on sekä rivi- että sarakenimikkeet. Pandas- ja NumPy-kirjastojen lisäksi työssä hyödynnettiin graphviz-kirjastoa, joka mahdollisti puun graafisen visualisoinnin (kts. kappale 6.3 kuva 4). [11.]

6.2 Datan esikäsittely

Ennen datan analysointia data muunnettiin käsiteltävämpään muotoon. Datan 30 eri parametreistä käytettiin timestamp-, name- ja session_id-parametrejä, joten vain nämä sarakkeet otettiin alkuperäisestä datasta talteen. Tämän jälkeen name-parametrin arvoja yksinkertaistettiin erottamalla tapahtuman oikeasta nimestä numeroarvo, jota oli tapahtuman yhteydessä käytetty. Esimerkiksi tapahtumasta ”Simulated Texture transparency slider moved to 27” pudotettiin pois sana ”moved to”, ja sanan jälkeen tuleva numeroarvo siirrettiin talteen toiselle sarakkeelle, jolloin jäljelle jäi vain käytetyn painikkeen nimi ”Simulated Texture transparency slider”. Tämän jälkeen tapahtumien vertailu oli helpompaa. Nimestä erotettuja numeroarvoja voitiin tarvittaessa käyttää myöhemmin esimerkiksi minimi- ja maksimiarvojen määrittämiseen, joten ne arvot säilytettiin datassa.

Numeroarvon erottamisen jälkeen yhdistettiin kaikki samat tapahtumat, jotka esiintyvät peräkkäin. Esimerkiksi lähennystapahtuma ”Zoom in” saattoi esiintyä jopa kymmenen kertaa peräkkäin alkuperäisessä datassa. Jotta datan analysoinnissa ei muodostuisi kymmenen tapahtuman sekvenssejä, joissa esiintyy vain yhtä tapahtumaa, samojen tapahtumien yhdistäminen oli välttämätöntä. Arvojen yhdistämisen yhteydessä otettiin tapahtuman minimi- ja maksimiarvot talteen.

Arvojen yhdistämisen jälkeen name-parametrin tapahtumanimille annettiin numeroarvo, koska tapahtumien vertailu on helpompaa numeerisessa muodossa. Kun jokaiselle tapahtumalle oli annettu tapahtumaa vastaava numeroarvo, jätettiin alkuperäinen tapahtumanimi pois ja uudet tapahtumanumerot järjestettiin istunnon ja aikamäärän mukaisesti,

jolloin saatiin taulukon 2 mukainen taulukko. Taulukon vertikaaliakselilla on istunnon tunnus ja horisontaalilla tapahtumat aikajärjestyksessä. Tapahtuma1-sarake kertoo siis, mikä tapahtumanumero on ollut istunnon ensimmäinen tapahtuma, ja tapahtuma2-sarake kuvaa, mikä tapahtuma on esiintynyt istunnossa toisena ja niin edelleen. Pisimmässä istunnossa on ollut siis 808 tapahtumaa ja lyhyimmässä vain kaksi. Tapahtumanumero -1 lisättiin niiden istuntojen perään, jotka ovat jo päättyneet.

Taulukko 2. Tapahtumat istunnon ja aikaleiman mukaan

istunto_id	tapahtuma1	tapahtuma2	tapahtuma3	…	tapahtuma808
0	1	2	3	…	-1
1	1	2	11	…	-1
2	1	30	2	…	-1
…	…	…	…	…	…
38	1	2	-1	…	-1

Jotta tapahtumanumeroiden muuntaminen takaisin oikeaksi tapahtumanimeksi onnistuisi helposti, muodostettiin toinen taulukko, johon lisättiin jokainen datassa esiintyvä tapahtuma ja sen tiedot vain kerran. Lopputuloksena syntyi taulukko 3.

Taulukko 3. Tärkeitä tietoja datasta

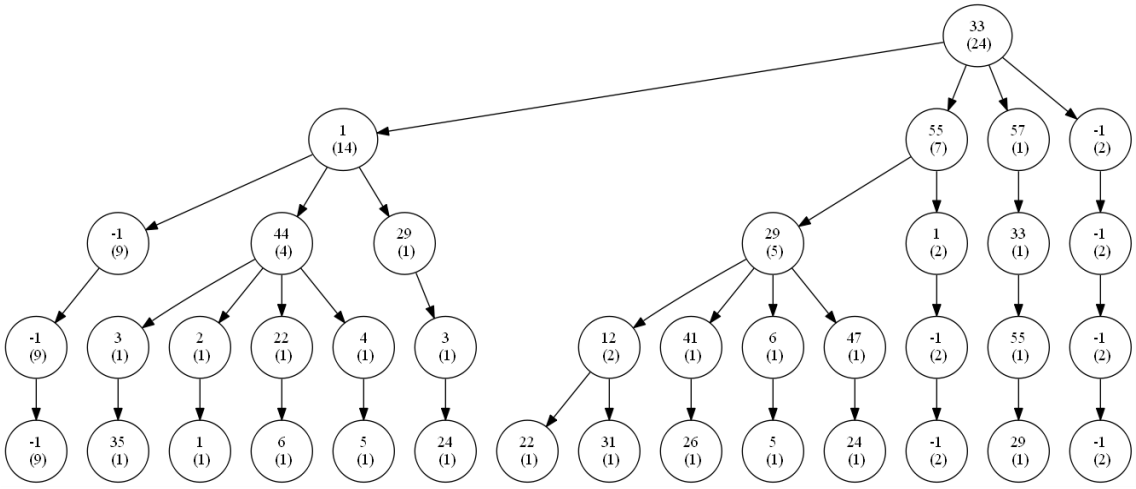
Tapahtumanumero	Tapahtuma	Min_arvo	Max_arvo
1	Patient opened	-999999	-999999
2	Start Smile Design	-999999	-999999
3	Align and crop	-999999	-999999
4	Calibrate for measurement	-999999	-999999
5	Teeth	-999999	-999999
6	SmileArea	-999999	-999999
7	Teeth transparency slider	10	20
8	Simulated Texture transparency slider	10	20
9	Pan (Move)	-999999	-999999
10	Snapshot to clipboard	-999999	-999999
…	…	…	…
92	Processed images (as they appear on screen) SELECTED	-999999	-999999
93	Cloud Export	-999999	-999999

Taulukon 3 ”Min_arvo” ja ”Max_arvo” -sarakkeisiin on syötetty ”-999999”-arvo, jos kyseisen tapahtuman kanssa ei käytetty mitään arvoja.

6.3 Puumallianalyysi

Yleisten tapahtumasekvenssien etsimisessä käytettiin puumallia, jossa tarkasteltava data istutettiin puumalliin vanhempi-lapsi-asetelmalla (kts. kuva 6). Työssä käytetty data oli ajallinen kategorinen data, joten puumallin avulla aineisto voitiin jäsennellä ja esitellä selkeästi. Puumallin avulla ei etsitty kaikista yleisintä tapahtumasekvenssiä, koska haluttiin saada mahdollisimman erilaisia sekvenssejä 100 %:n kattavuudella. 100 %:n kattavuudella tarkoitetaan, että jokainen tapahtuma löytyy ainakin kerran yleisten tapahtumasekvenssien joukosta. Tästä syystä jokaiselle tapahtumalle etsittiin yleisin L + 1-pituinen sekvenssi, jossa L kuvaa puun syvyyttä.

Puumallianalysoinnissa jokainen solmu kuvassa 6 kuvaa yhtä tapahtumaa, ja jokainen puun poluista kuvaa yhtä tapahtumasekvenssiä. Tapahtumanumeron alapuolella suluissa oleva numero kertoo, kuinka moni datan tapahtumasekvensseistä vastaa puun sen hetkistä polkua. Toisin sanoen suluissa oleva numero kertoo, kuinka yleinen on siihen asti oleva polku. Mitä syvemmälle puuta mennään, sitä yksilöllisemmäksi polut muuttuvat. Puuta analysoimalla selvitetään siis, mikä puun poluista on yleisin määrätyllä syvyydellä x.



Kuva 6. Puumalli tapahtumanumerosta 33

Kuvan 6 puun poluista kaikista vasemman puolimmaisin on yleisin, kun puun syvyytenä on 4. Kuvasta myös huomataan, miten nopeasti yhteneväisyys eri tapatumasekvenssien välillä hupenee, mitä syvemmälle puuta mennään. Tämä osin johtuu datan pienestä koosta, mutta kuvaa myös hyvin puumallianalyysin ongelmaa. Matalilla puilla saadaan yleisempiä sekvenssejä, mutta sekvenssit koostuvat vain parista tapahtumasta, kun taas syvillä puilla saadaan rakennettua monimutkaisiakin tapahtumasekvenssejä, mutta sekvenssiä ei voida välttämättä enää luokitella usein esiintyväksi.

Puumallianalyysi koostui monesta eri välivaiheesta. Näiden välivaiheiden selventämiseksi on alapuolella esitettynä kolme työssä käytettyä funktiota pseudokoodin muodossa.

```
def getAll_frequentSubSequences_before(db_data, length):
                          all_subseqs{}
    ls_events <- get_events(db_data)
    for event in ls_events:
       seq_bef <- getSequenceDf_before(db_data, event, length)
       seq_freq <- FrequentSubSequences(event,seq_bef,length)
                   all_subseqs.update(seq_freq)

   return all_subseqs
```

Pseudokoodi 1. Yleisten subsekvenssien etsiminen

```
def getSequenceDf_before(db, event, length):
    seq_lists[]
    verif_list[]
    l1 <- [-1] * length
    for i=0 to len(db)-1:
        seq <- np.append(l1, db.loc[i].values) #safety net
        for j=0 to len(seq)-1:
            if seq[j] == event:
                v <- [i, j, event]
                add v to verif_list
                ls <- getBefore(event, j, seq, length)
                add ls to seq_lists
    df <- pd.DataFrame({'id_verif' : verif_list , 'sequence' : seq_lists})
    return df
```

Pseudokoodi 2. Tapahtumasekvenssien generointi

```
def FrequentSubSequences(event, df,l):
                root <- Tree(event)
    for value, key in df.values:
            populate_tree(root, key, value)
    top_seq <- find_top_sequence(root,l)
    return top_seq
```

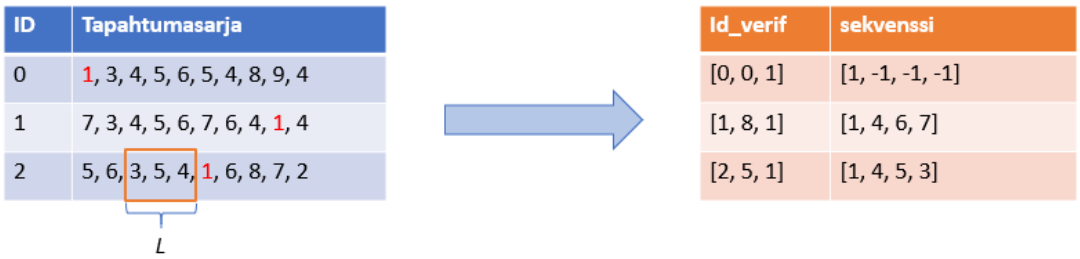
Pseudokoodi 3. Sekvenssien istutus puuhun ja yleisimmän sekvenssin määrittely

```
def populate_tree(root, sequence, id):
    n <- root
    for s in sequence[1:]:
        n.add_payload(id)
        if not s in n.children:
            n.add_child(s)
        n <- n.children[s]

    n.add_payload(id)
```

Pseudokoodi 4. Sekvenssin istutus puuhun

Yleisten tapahtumasekvenssien etsiminen on esitetty kokonaisuudessaan pseudokoodissa 1. Alkutietona funktioon syötettiin esikäsitelty käyttäjätapahtuman jäljitedata ja haluttu puun syvyys. Tämän jälkeen haettiin kaikki eri tapahtumanumerot datasta ”get_events”-funktion avulla. Funktio ”get_events” palauttaa listan datassa esiintyvistä eri tapahtumanumeroista. Koska datassa on 93 eri tapahtumanumeroa, palautti funktio listan, joka sisältää 93 eri numeroa. Seuraavaksi käytiin jokainen listan tapahtumanumeroista läpi yksi kerrallaan, paikannettiin kaikki tarkasteltavan tapahtumanumeron esiintyvyydet ja muodostettiin niistä tapahtumasekvenssit. Tämä vaihe on esitettynä pseudokoodissa 2. Funktioon (kts. pseudokoodi 2) syötettiin alkutietona esikäsitelty käyttäjätapahtuman jäljitedata, tarkasteltava tapahtumanumero ja halutun puun syvyys L. Kun tarkasteltavaa tapahtumanumeroa etsittiin datasta, otettiin datasta yksi rivi kerrallaan tarkasteluun. Ennen syvempää tarkastelua jokaisen tarkasteltavan rivin eteen lisättiin L määrä -1:siä, minkä syy selviää funktion myöhemmässä vaiheessa. Kun L määrä -1:siä on lisätty riviin, katsottiin, löytyykö tarkasteltavaa tapahtumanumeroa riviltä. Jos tarkasteltava tapahtumanumero löytyi, kirjattiin ensiksi sen sijainti ylös, jotta pystyttiin verifioimaan funktion toimivuus. Tämän jälkeen muodostettiin tarkasteltavasta tapahtumasta sekä sen edeltävistä tapahtumista tapahtumasekvenssi funktion ”getBefore” avulla. Oli tärkeämpää ottaa tapahtumasekvenssiin tarkasteltavaa tapahtumaa ennen olevat tapahtumat kuin jälkeiset, koska tällöin ohjelmisto saadaan tiettyyn tilaan testaamaan tarkasteltavana olevaa tapahtumaa. Jos tarkasteltavana tapahtumana olisi esimerkiksi kuvan avaaminen, niin toimivuuden varmistamiseksi on oleellisempaa tarkastella eri polkuja, jotka johtavat kuvan avaamiseen kuin kuvan avaamisen jälkeisiä toimintoja. Näin saatetaan vaikka saada selville, että kuvan avaaminen ei onnistu, jos on käyty moduulissa x. Funktiossa ”getBefore” muodostettiin siis lista, johon ensiksi lisättiin tarkasteltavana oleva tapahtuma ja L määrä edeltäviä tapahtumia (kts. kuva 7.). Jos tarkasteltavaa tapahtumaa ennen ei ollut niin montaa tapahtumaa, tuli aikaisemmin lisätyt -1:set avuksi.



Kuva 7. Tapahtumasekvenssin generointi

Kuvassa 7 on havainnollistettu kolmen tapahtumasekvenssin generointi tapahtumanumerosta 1. Sinisellä pohjalla olevat taulukko kuvaa esikäsiteltyä käyttäjätapahtuman jäljitedataa, josta paikannettiin numeroa 1. Kun numero 1 löytyi, tallennettiin sen sijainti ”Id_verif”-sarakkeeseen ja muodostettiin tapahtumasekvenssi, johon lisättiin numero 1 ja L määrä edeltäviä tapahtumia. Kuvan esimerkissä L:n arvona on 3.

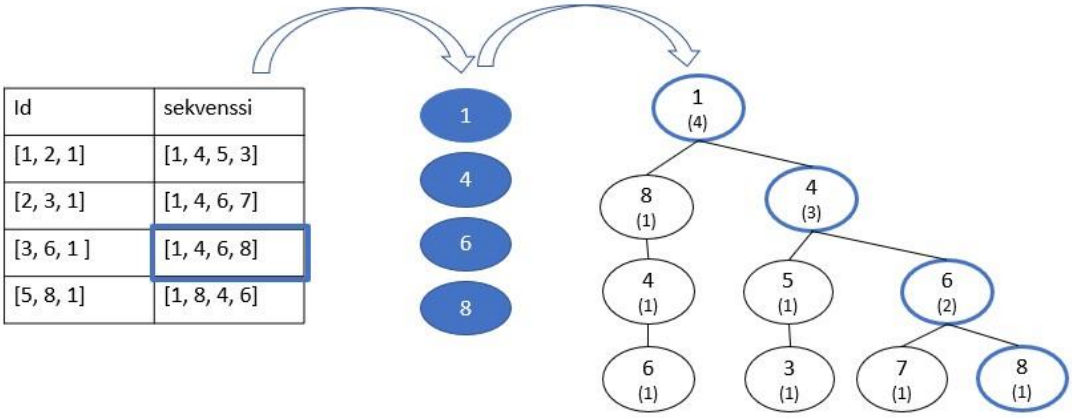
Kun kaikki tarkasteltavan tapahtuman sijainnit oli paikannettu ja tapahtumasekvenssit generoitu, vietiin sijainti- ja sekvenssitiedot Pandas-kirjaston datakehykseen, ja palautettiin takaisin pseudokoodin 1 funktiolle. Palautettava datakehys tapahtumanumerosta 1 on esitettynä taulukossa 4. Tapahtuman sijainti on kuvattuna ”id_verif”-sarakkeella, ja sijaintia vastaava tapahtumasekvenssi on esitettynä sekvenssisarakkeella.

Taulukko 4. Tapahtumanumeron 1 eri sijainnit ja tapahtumasekvenssit

id_verif	sekvenssi
[0, 6, 1]	[1, -1, -1, -1, -1]
[1, 6, 1]	[1, -1, -1, -1, -1]
[3, 75, 1]	[1, 44, 40, 12, 26]
…	…
[36, 33, 1]	[1, 44, 4, 5, 8]
[37, 6, 1]	[1, -1, -1, -1, -1]
[38, 6, 1]	[1, -1, -1, -1, -1]

Kun funktiosta ”getBefore” on saatu tulosteena datakehys, muodostettiin siitä puu funktion ”FrequentSubSequences” avulla, joka on esitettynä pseudokoodissa 3.

”FrequentSubSequences”-funktioon syötettiin alkutietoina tarkasteltava tapahtumanumero, aikaisemmassa vaiheessa muodostettu datakehys ja haluttu puun syvyys. Puu muodostettiin ensiksi syöttämällä puun juureksi (engl. root) tarkasteltava tapahtumanumero, ja tämän jälkeen tapahtumasekvenssit istutettiin yksi kerrallaan puuhun ”populate_tree”-funktiolla, joka on esitettynä pseudokoodissa 4. Alkutietoina ”populate_tree”-funktioon viedään puun juuri, tarkasteltavana oleva sekvenssi ja tarkasteltavan sekvenssin sijaintitieto. Funktion alussa määritetään tarkasteltava solmu, joka on aluksi puun juuri. Tämän jälkeen kaikki sekvenssin muut tapahtumat paitsi ensimmäinen käydään läpi yksi kerrallaan. Sekvenssin ensimmäinen tapahtuma jätetään pois, koska se on jo määritetty juureksi. Kun sekvenssin tapahtumia käydään läpi, tarkastellaan, löytyykö kyseinen tapahtuma jo tarkasteltavan solmun lapsista. Jos tapahtumaa ei löydy, lisätään se solmun lapseksi, ja nykyisestä lapsesta tulee tarkasteltava solmu. Lapsien lisäksi puuhun lisätään jokaisen solmun kohdalla sekvenssin identifikaatiosarja eli sekvenssin sijaintitieto, jotta pystytään verifioimaan funktion toimivuus ja myöhemmin laskemaan sekvenssin yleisyys. Kuvassa 8 on vielä havainnollistettuna sekvenssin istutus puuhun graafisesti testidatan avulla.



Kuva 8. Tapahtumasekvenssin istuttaminen puuhun

Kuvassa 8 juureksi on määritetty numero yksi ja taulukon jokainen sekvenssi istutetaan yksi kerrallaan puuhun. Istuttaessa puuhun sinisellä rajattua sekvenssiä puuta käydään läpi syvyyskohtaisesti. Ensiksi aloitetaan juuresta, jonka syvyys on nolla. Jos juurella ei ole numero neljää lapsena, lisätään se juuren lapseksi ja lapsen puusolmuun lisätään kyseisen sekvenssin id ja otetaan kyseinen puusolmu tarkasteltavaksi.

Jos tarkasteltava numero on jo solmun lapsena, lisätään solmun lapsen puusolmuun sekvenssin id ja otetaan se tarkasteluun. Tätä jatketaan, kunnes sekvenssi loppuu.

Kun puu on valmis, etsitään siitä yleisin sekvenssi pseudokoodissa 3 esiintyvän ”find_top_sequence”-funktion avulla. Funktiossa tarkastellaan puun lehtiä ja lasketaan, kuinka monta id:tä mistäkin lehdestä löytyy. Siitä lehdestä, josta löytyi eniten identifikaatioita, seurataan puupolkua takaisin juureen, tallennetaan sekvenssi ja poistetaan mahdolliset -1:set. Lopuksi siistitty sekvenssi palautetaan pseudokoodille 1, jossa se tallennetaan ”all_subseqs”-listaan. Tämän jälkeen otetaan uusi tapahtumanumero, ja käydään koko prosessi uudestaan. Tätä jatketaan, kunnes kaikki 93 tapahtumanumeroa on käyty läpi ja loppu tuloksena syntyy 93 yleistä sekvenssiä.

6.4 Yleiset sekvenssit

Ennen testiautomaatioon siirtymistä käydään sekvenssit vielä läpi ja yhdistetään samankaltaiset sekvenssit. Taulukossa 5 on esitettynä pieni ote puumallin generoiduista sekvensseistä. Yleiset sekvenssit on talletettu avain-arvopareiksi. Avaimena toimii itse sekvenssi, ja sen arvoksi on syötetty puumallista saatu lehden identifikaatioiden lukumäärä. Arvo siis kertoo, kuinka monta kertaa sekvenssi esiintyy datassa.

Taulukko 5. Tapahtumanumeroiden 1-5 yleiset sekvenssit

Avain	Arvo
(1, -1, -1, -1, -1)	36
(2, 1, -1, -1, -1)	17
(3, 4, 3, 4, 3)	6
(4, 3, 4, 3, 4)	8
(5, 4, 3, 4, 3)	7

Kuten taulukon 5 Avain-sarakkeesta näkee, monet sekvensseistä ovat samankaltaisia. Osa sekvensseistä ovat toisten sekvenssien alisekvenssejä, joten sekvenssit käydään uudestaan läpi ja yhdistetään kaikki, joilla on yhteneviä osia. Esimerkiksi

taulukon 5 viidestä eri sekvensseistä muodostuu yhdistämisen jälkeen vain kaksi sekvenssiä, jotka on esitetty alapuolella.

(2, 1, -1, -1, -1, -1)
(5, 4, 3, 4, 3, 4, 3)

Kun kaikki 93 sekvenssiä on käyty läpi ja alisekvenssit yhdistetty, jää jäljelle 55 yleistä sekvenssiä, joista lyhyimmät ovat kahden tapahtuman mittaisia ja pisimmät 13 tapahtuman mittaisia sekvenssejä. Jotta näitä jäljelle jääneitä sekvenssejä voitaisiin käyttää testiautomaatiossa, niitä pitää ensiksi muokata. Ensimmäisenä sekvensseistä poistetaan NA-arvona käytetty -1 arvo, koska kyseistä arvoa ei enää tarvita. Tämän jälkeen kaikki sekvenssit käännetään ympäri, koska puumallia varten datasta kerättiin tarkasteltavaa tapatumaa ennen olevat tapahtumat ja syötettiin uuteen listaan tarkasteltavan tapahtuman taakse. Tämän seurauksena sekvenssistä tuli peilikuva, josta oli hyötyä työssä käytetyssä puumallissa.

Viimeisenä sekvenssin tapahtumanumerot muutetaan takaisin nimiksi. Tähän käytetään apuna datan alkukäsittelyn aikana luotua datakehystä, johon tallennettiin jokaisen tapahtumanumeron nimi ja tapahtumien minimi- sekä maksimiarvot. Tapahtumanumero korvataan tapahtumanimellä hakemalla datakehyksestä tapahtumanumeroa vastaava nimi. Kun kaikki sekvenssit on käyty läpi, viedään yleiset sekvenssit CSV-tiedostoon, josta ne ovat käytettävissä testiautomaatiota varten.

6.5 Tulokset

Ote analyysin tuloksena syntyvistä yleisistä sekvensseistä on esiteltynä taulukossa 6. Analyysin tavoitteena oli saada käyttäjien yleisimmät tapahtumasekvenssit, joita voitaisiin käyttää ohjelmistotestauksessa, ja täten parantaa ohjelmistotestauksen kattavuutta. Kattavuus parantuu, mikäli datan analysoinnissa syntyy sellaisia tapahtumasekvenssejä, jotka eroavat ohjelmiston aikaisemmasta testiajosta. Tarkasteltaessa analysoinnista saatuja sekvenssejä (kts. taulukko 6), huomataan heti ainakin yksi suuri eroavaisuus aikaisempiin testisekvensseihin: kahden tapahtuman välinen edestakainen vuorottelu.

Taulukko 6. Ote yleisistä sekvensseistä

ID	Tapahtuma_1	Tapahtuma_2	Tapahtuma_3	Tapahtuma_4	Tapahtuma_5
1	Patient opened	Start Smile Design	Import	Image Export	SmileDesign Export Image Dlg: Cancel
2	Teeth transparency slider	Simulated Texture transparency slider	Teeth transparency slider	Simulated Texture transparency slider	Teeth transparency slider
3	Import	Zoom In	Align and crop	Zoom In	Zoom Out
4	Start Smile Design	Clone Brush	Size slider	Warp Brush SELECTED	Size slider
5	Patient opened	Cancel Photo			
6	Size slider	Teeth	Print Images and Edit print Layouts	Teeth	Reset all teeth MenuItem selected
7	Patient opened	Patient changed			
8	Size slider	Undo #	Warp Brush SELECTED	Teeth	Mirror all teeth to opposite side MenuItem selected
9	Pan (Move)	Zoom To Fit	Teeth transparency slider	Teeth	Reset this tooth MenuItem selected
10	Add text	Align and crop	Calibrate for measurement	Face image	CAD/CAM Export

Vertailtaessa tarkemmin aikaisempia testisekvenssejä työn testisekvensseihin huomataan, ettei mikään data-analyysissa muodostuneista 55 sekvensseistä täsmää aikaisemman testiajon testisekvenssien kanssa. Työssä onnistuttiin siis saamaan 55 uutta testisekvenssiä. Tähän suurimpana syynä on luultavasti, että suurin osa aikaisemmista testisekvensseistä koostuu yhdestä tai parista tapahtumasta, mutta vaikka

koko testiajoa katsottaisiin yhtenä pitkänä testisekvenssinä, ei nämä kaksi testausmenetelmää siltikään kohtaisi. Tämän perusteella voidaan siis todeta, että ohjelmistotestauksen kattavuuden parantamisessa onnistuttiin.

7 Testiautomaatio

Työn testiautomaatio-osuudessa selvitetään, miten hyvin työssä käytetty data soveltuu testiautomaation käyttöön. Selvityksessä suunnitellaan testiautomaatioajo, joka käyttää puumallin avulla luotuja yleisiä sekvenssejä testaukseen. Testiautomaatioajon suunnittelu rajataan kuitenkin koskemaan vain Smile Design -ohjelmiston yhtä moduulia. Suunnitelman pohjalta arvioidaan, miten hyvin käyttäjätapahtuman jäljitedataa pystytään hyödyntämään testiautomaatiossa ja mitä muutoksia dataan tulisi tehdä.

7.1 Testiautomaatiosuunnitelma

Testiautomaatio toteutetaan käyttäen Robot Framework -testiautomaatiokehystä, joka on yleinen kehys hyväksymistestaukseen ja hyväksyntätestilähtöiseen kehitykseen [15]. Datan kannalta oleellisimmat Robot Frameworkin kirjastot ovat JavaFX ja Swing, koska näiden avulla pystytään käyttämään Smile Design -ohjelmiston painikkeita.

Testiautomaatioajo rakennettaan yleisten sekvenssien ympärille. CSV-tiedostoon viedyt sekvenssit haetaan yksi kerrallaan testiautomaation suoritettavaksi. Sekvenssi tulee tapahtumalistana, josta otetaan aina yksi tapahtuma kerrallaan tarkasteltavaksi. Tarkasteltavalle tapahtumalle etsitään heti aluksi Smile Design -ohjelmiston käyttämä tunnus ja merkitään, kumman kirjaston komponentti on kyseessä (JavaFX vai Swing). Kirjastotyypin lisäksi tarvitaan tieto, minkä tyyppinen tapahtuman painike on, jotta osataan valita oikea komento. Painike voi olla peruspainike, liukusäädin, radiopainike, valintaruutu tai tekstikenttä. Näiden kaikkien painikkeiden käyttämiseen tarvitaan eri komentoja, joten tyyppitieto on oleellinen.

Kun tapahtumaan on linkitetty tunnus, painikkeen tyyppi ja kirjasto, jaetaan tapahtumat vielä mukautettuihin tapahtumiin ja perustapahtumiin, koska entuudestaan tiedetään, että kaikkia painikkeita ei Smile Design -ohjelmistossa vielä seurata. Seuraamattomia tapahtumia ovat muun muassa joidenkin dialogien painikkeet tai kuvan päällä tapahtuvat toiminnot. Mukautettuja tapahtumia ovat siis kaikki painikkeet, jotka ovat riippuvaisia seuraamattomista tapahtumista, joita ei näy sekvenssissä. Esimerkiksi tapahtumasta Import aukeaa kuvantuonnin dialogi. Dialogiin pitää syöttää halutun kuvan tiedostopolku ja painaa OK-painiketta tai peruuttaa kuvantuonti, jotta testiautomaatioajo selviää tästä kyseisestä tapahtumasta. Ennen kokonaista testiautomaatioajoa jokainen mukautettu tapahtuma käydään läpi ja määritetään tarvittavat tapahtumaketjut. Perustapahtumille riittää pelkkä tietojen linkitys, jonka avulla määrätään tarvittava komento. Perustapahtumia ovat painikkeet, jotka toimivat itsenäisesti eivätkä aukaise dialogeja.

7.2 Analyysi

Testaussuunnitelman aikana huomattiin monia ongelmakohtia, jotka vaativat tarkempaa selvitystä ja konsultointia. Ensimmäisenä huomattiin, että datan 93 tapahtumasta melkein puolet ovat mukautettuja painikkeita, jotka tarvitsevat personoidun käsittelyn ennen testiautomaation ajoa. Toinen ongelmakohta oli tunnuksen, painiketyypin ja kirjaston linkitys tapahtumaan. Suunnitelman pohjalta selvitettiin, miten helposti linkitys käytännössä onnistuisi. Koska data ei itsessään pidä sisällään mitään kyseisistä tiedoista, eikä painikkeiden tiedoista pidetä kirjaa, täytyisi linkitys tehdä manuaalisesti. Datan kaikki 93 eri tapahtumaa pitäisi käydä yksi kerrallaan läpi, katsoa Smile Design -ohjelmiston käyttöliittymästä painikkeen tunnus, tyyppi ja kirjasto ja kirjoittaa nämä tiedot käsin koodiin. Tämä ei ole ihanteellisin tapa.

Jotta linkitys onnistuisi helpommin, käyttäjien tapahtumajälkidata voisi pitää sisällään tarkempia tapahtuman tyyppitietoja. Painikenimen sijaan data voisi käyttää suoraan yksilöityjä painiketunnuksia. Tämä poistaisi myös painikenimen väärinymmärrysvaaran. Työssä käytetyssä datassa on esimerkiksi kaksi ”Import”-tapahtumaa (”Import photo”, ”Import”), jotka eivät nimellään avaa tarpeeksi, mikä useista kuvantuonti painikkeista on kyseessä. Ainoastaan tutkimalla tapahtumien edeltäviä tapahtumia

voidaan päätellä, että tapahtumalla ”Import photo” tarkoitetaan potilastietojen muokkaamisdialogissa olevaa kuvanvientipainiketta. Tästä ei kuitenkaan saada 100 %:n varmuutta. Painiketunnuksen käyttäminen painikenimen sijaan helpottaisi myös datan keruuta eri kielisistä versioista, koska tuolloin ei tarvitsisi huolehtia eri kielisten painikenimien käännöksistä. Painiketunnuksen lisäksi olisi hyödyllistä, jos data pitäisi sisällään myös painiketyypin sekä tiedon, kumman kirjaston komponentti on kyseessä.

8 Yhteenveto ja pohdinnat

Insinöörityön päätavoitteena oli parantaa ohjelmistotestauksen kattavuutta Smile Design -ohjelmiston käyttäjistä kerätyn jäljitedatan avulla. Tarkoituksena oli löytää käyttäjien usein käyttämät tapahtumasekvenssit, joita voitaisiin käyttää myöhemmin ohjelmistotestauksessa. Data esikäsiteltiin ja vietiin puumallianalyysiin, josta tuloksena tuli 93 puun suuruinen metsä. Näitä puita analysoimalla saatiin jokaiselle tapahtumanumerolle yleinen sekvenssi. 93 eri sekvenssiin mahtui kuitenkin paljon samanlaisia sekvenssejä, jotka olivat toisten sekvenssien alisekvenssejä. Samankaltaisten sekvenssien yhdistämisen jälkeen jäljelle jäi 55 yleistä sekvenssiä.

Kattavuus määritettiin parantuvan, mikäli datan analysoinnista syntyy sellaisia tapahtumasekvenssejä, jotka eroavat ohjelmiston nykyisestä testiajosta. Tarkasteltaessa analysoinnista saatuja sekvenssejä huomattiin heti ainakin yksi suuri eroavaisuus olemassa olevista testisekvensseistä: kahden tapahtuman välinen edestakainen vuorottelu.

Vertailtaessa tarkemmin aikaisempia testisekvenssejä työn testisekvensseihin huomattiin, ettei mikään data-analyysissa muodostuneista 55 sekvensseistä täsmää aikaisemman testiajon testisekvenssien kanssa. Tähän suurimpana syynä todettiin, että suurin osa aikaisemmista testisekvensseistä koostui yhdestä tai parista tapahtumasta, mutta vaikka koko testiajoa katsottaisiin yhtenä pitkänä testisekvenssinä, todettiin, ettei nämä kaksi testausmenetelmää siltikään kohtaisi. Tämän perusteella voitiin todeta, että ohjelmistotestauksen kattavuuden parantamisessa onnistuttiin.

Insinöörityö toisena tavoitteena oli saada selville, miten hyvin käyttäjätapahtuman jäljitedataa pystytään käyttämään testiautomaatiossa. Luvun 7 pohjalta voidaan todeta, että datan sisältö ei vielä riitä kattavan testiautomaatioajon rakentamiselle. Jotta datan avulla voitaisiin testata Smile Design -ohjelmistoa helposti, pitäisi kerätä tietoa myös painikkeen tunnuksesta, tyypistä (esim. painike, valintaruutu vai liukusäädin) ja kirjastokomponentista (Swing vai JavaFX). Näiden lisäksi mahdollisten puuttuvien tapahtumien määrä pitäisi minimoida seuraamalla kaikkia tapahtumia, joita pystyy. Jotta testiautomaatioajoa voitaisiin käyttää koko Smile Design -ohjelmistoon, pitäisi testiautomaation myös osata käsitellä mukautettuja tapahtumia helpommin ja tietää, missä moduulissa tapahtuma on. Tämä ongelma on hoidettu eräässä web-pohjaisessa ohjelmistossa Document Object Model (DOM) -elementillä ja ohjelman tila -tiedolla. DOM-elementillä on puumallin rakenne, jonka avulla se kertoo objektien välisen suhteen toisiinsa. DOM-elementin avulla saadaan siis selville tapahtuman polku. Ohjelman tilan määrittämiseksi käytettiin URL-osoitetta ja sivun otsikkoa. [16; 17.]

Kaikki yllä luetellut datan kehitysideat on kerätty taulukkoon 7. Taulukossa on esimerkkinä hypoteettinen tapahtuma ”12345”. Taulukosta nähdään, että tapahtuma on Swing-kirjastokomponentin liukusäädin ja että se on tapahtuman ”23456” alitapahtuma. Tapahtuma ”12345” on siis näkyvissä vain silloin, kun ”23456” on tapahtunut. Taulukko kertoo myös, missä tilassa ohjelmisto on ollut. Tapahtuman ”12345” tapauksessa käyttäjä on ollut Smile-moduulissa.

Taulukko 7. Kehitysidea tapahtumatiedon keräämiselle

Tunnus	Tyyppi	Komponentti	Polku	Tila
12345	Slider	Swing	23456, 12345	Smile

Jos käyttäjätapahtuman jäljitedatasta saataisiin taulukon 7 mukaiset tiedot, voisi dataa käyttää koko Smile Design -ohjelmiston testaamiseen. Samoilla tiedoilla onnistuisi myös koko Romexis-ohjelmiston testaaminen, kun käyttäjätapahtuman jäljitedata on kerätty.

Vaikka data todettiin riittämättömäksi testiautomaatioajoon, yritys sai silti hyviä kehitysideoita, kuten esimerkiksi mitä lisätietoa pitäisi kerätä painikkeista. Opinnäytetyön molemmissa tavoitteissa siis onnistuttiin, minkä pohjalta voidaan todeta työn olleen onnistunut.

Lähteet

1. Planmeca Oy. Planmeca Better care through innovation. Verkkosivu. Luettu 03.02.2020. https://www.planmeca.com/fi/yritys/.
2. Planmeca Oy. Planmeca Group. Verkkosivu. Luettu 03.02.2020. https://www.planmeca.com/fi/yritys/planmeca-group/.
3. Planmeca Oy, Romexis® Smile Design Intuitive digital smile design software. Verkkoaineisto. Luettu 6.4.2020. https://www.planmeca.com/software/softwaremodules/planmeca-romexis-smile-design/.
4. Medical Expo. Planmeca Romexis SmileDesign Software. Verkkoaineisto. Luettu 5.5.2020. https://pdf.medicalexpo.com/pdf/planmeca/planmeca-romexis-smile-design-software/73644-192689-_4.html.
5. William E. Lewis. Software Testing and Continuous Quality Improvement, 3rd Edition. E-kirja. Luettu 04.07.2020. https://learning.oreilly.com/library/view/software-testing-and/9781351722209/.
6. Software testing help. What Is Defect/Bug Life Cycle In Software Testing? Defect Life Cycle Tutorial. 2020. Verkkoaineisto. Luettu 15.09.2020. https://www.softwaretestinghelp.com/bug-life-cycle/.
7. James A. Whittaker. Exploratory Software Testing. Kappale 2. Addison-Wesley Professional 2009. E-kirja. Luettu 20.07.2020. https://learning.oreilly.com/library/view/exploratory-software-testing/9780321647863/ch02.html#ch02.
8. Ron Patton. Software Testing, Second Edition. Kappale 15. Sams 2005. E-kirja. Luettu 08.07.2020. https://learning.oreilly.com/library/view/software-testing-second/0672327988/ch15.html.
9. Jean-Louis Monino, Soraya Sedkaoui. Big Data, Open Data and Data Development, key concepts chapter.Wiley-ISTE 2016. E-kirja.
10. Cambridge Dictionary. Data. Verkkoaineisto. Luettu 17.06.2020. https://dictionary.cambridge.org/dictionary/english/data.
11. Wes McKinney. Python for Data Analysis. Kappale 1. O'Reilly Media, Inc 2012. Luettu 20.02.2020. https://learning.oreilly.com/library/view/python-for-data/9781449323592/ch01.html.
12. Cuesta, Hector. Practical Data Analysis. 2013. ProQuest Ebook Central s. 7-9. E-kirja. Luettu 03.04.2020. https://ebookcentral.proquest.com/lib/metropolia-ebooks/reader.action?docID=1507840.

13. Aida Jimenez, Fernando Berzal, Juan-Carlos Cubero. Frequent tree pattern mining: A survey. Verkkoaineisto. Luettu 27.08.2020. https://www.researchgate.net/publication/220031813_Frequent_tree_pattern_mining_A_survey.
14. Jiawei Han, Jian Pei, Yiwen Yin. Mining Frequent Patterns without Candidate Generation. Simon Fraser University.Verkkoaineisto. Luettu 27.08.2020. https://www.cs.sfu.ca/~jpei/publications/sigmod00.pdf.
15. Robot Framework, Introduction. Verkkoaineisto. Luettu 8.4.2020 https://robotframework.org/.
16. Markus Ermuth, Michael Pradel. Monkey See, Monkey Do: Effective Generation of GUI Tests with Inferred Macro Events. Verkkoaineisto. Luettu 6.1.2020 http://www.software-lab.org/publications/issta2016-macros.pdf.
17. w3schools JavaScript Tutorial. JavaScript HTML DOM. Verkkoaineisto. Luettu 22.5.2020. https://www.w3schools.com/js/js_htmldom.asp.
18. Jupyter. JupyterLab: Jupyter's Next-Generation Notebook Interface. Verkkosivu. Luettu 09.09.2020. https://jupyter.org/.
19. Galea Alex. Beginning Data Science with Python and Jupyter : Use Powerful Industry-Standard Tools Within Jupyter and the Python Ecosystem to Unlock New, Actionable Insights from Your Data. 2018. E-kirja. Luettu 01.08.2020. http://ebookcentral.proquest.com/lib/metropolia-ebooks/detail.action?docID=5419744