Lauri Holopainen

Avoimen lähdekoodin projektien kehitystehtävien valmistumisajan arviointi koneoppimismenetelmin

Tietotekniikan pro gradu -tutkielma

19. tammikuuta 2022

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Lauri Holopainen

Yhteystiedot: lauri.m.s.holopainen@gmail.com

Ohjaaja: Ville Isomöttönen

Työn nimi: Avoimen lähdekoodin projektien kehitystehtävien valmistumisajan arviointi koneoppimismenetelmin

Title in English: Predicting resolution time of open source project issues with machine learning methods

Työ: Pro gradu -tutkielma

Opintosuunta: Ohjelmistotekniikka

Sivumäärä: 70+0

Tiivistelmä: Ohjelmistovirheiden korjausajan arviointi on perinteisesti tehty asiantuntijaarvioinnin perusteella. Automaattisia menetelmiä korjausajan ennustamiseksi on kehitetty lähinnä keskittyen yksittäisiin ohjelmistoprojekteihin. Tässä tutkimuksessa replikoitiin ohjelmistoprojektien kehitystehtävien valmistumisaikoja koneoppimismenetelmin ennustava tutkimus, ja tutkittiin saman menetelmän soveltuvuutta virheraporttien sulkeutumisaikojen ennustamiseen. Koneoppimismenetelmänä käytettiin Random Forest -luokittimia. Aineisto käsitti yli 39 tuhatta avoimesti saatavilla olevaa ohjelmistoprojektia, lähes 13 miljoonaa kehitystehtävää ja yli 1.5 miljoonaa virheraporttia. Replikaatiotutkimuksessa laajempi opetus- ja testausdata tuottivat hieman alkuperäistä tutkimusta heikommat tulokset, mutta uudet havainnot kuitenkin vahvistivat alkuperäisiä huomioita. Virheraporttien sulkeutumisajan ennustamisen huomattiin olevat haastavampi tehtävä kuin yleisempi kehitystehtävän valmistumisajan arviointi, ja menetelmää voisi kehittää tutkimalla luokittimen kannalta hyödyllisimpiä muuttujia. Tulosten perusteella näyttäisi siltä, että tekoälyn sovellettavuus yksittäisiä virheraportteja tarkastellessa on vielä heikkoa, mutta siitä voi olla hyötyä suuria tietomääriä käsiteltäessä.

Avainsanat: ohjelmistovirhe, virheraportti, koneoppiminen, tiedonlouhinta, avoin lähdekoodi

Abstract: Estimating the resolution time of software defects has usually been done based on expert knowledge. Automatic methods to predict the resolution time has been done mostly by focusing on a few software projects. In this study, a replication of a previous study about predicting the issue resolution time of software projects with machine learning methods was performed, and the suitability of the method to predict the resolution time of defect reports was explored. Random Forest classification was the implemented machine learning method. The data set consisted of over 39 thousand publicly available software projects, almost 13 million issues and over 1.5 million defect reports. Larger data set of the replication produced slightly weaker results compared to the original paper, but new observations verified the points of the original study. Predicting the resolution time of defect reports turned out to be a harder task than the more general task of predicting resolution time of an issue. The method could be improved by studying the most useful features for the classifier. On the basis of the results, it would seem that the applicability of artificial intelligence when inspecting isolated defect reports is poor, but it could be useful when handling large data collections.

Keywords: software defect, bug report, machine learning, data mining, open source code

Kuviot

Taulukot

Sisältö

1 Johdanto

Ohjelmointivirheet ovat väistämätön osa ohjelmiston kehitystyötä, ja niiden huomaamiseksi ja korjaamiseksi on kehitetty keinoja läpi tietokoneiden historian. Pienet virheet voivat olla vain kevyt hidaste tai jäädä jopa huomaamatta, mutta ne voivat olla myös vakavia turvallisuusriskejä ja pysäyttää suuriakin järjestelmiä. Nykyään korjatut ohjelmistot voidaan tuoda loppukäyttäjien saataville hyvin nopeasti esimerkiksi tarjoamalla päivitykset mobiililaitteiden sovelluskauppojen kautta. Mitä myöhemmässä vaiheessa virheet huomataan, sitä enemmän resursseja, siis työaikaa, voi olettaa kuluvan niiden korjaamiseen.

Useat tutkijat ovat olleet kiinnostuneita siitä, kuinka virhekorjauksiin kuluvaa aikaa voisi arvioida matemaattisten mallien avulla. Onnistunut arviointi tehtävän vaatimalle ajalle voi auttaa kehitysprojektin vetäjiä aikataulun suunnittelussa ja resurssien suuntaamisessa. Loppukäyttäjiä hyvät korjausaika-arviot voivat myös hyödyttää esimerkiksi oman työnsä suunnittelussa tai sen arvioinnissa, että onko järkevää jäädä odottamaan korjauksen valmistumista.

Tässä tutkimuksessa replikoidaan aiempi Riivon, Dumasin ja Pfahlin (2016) tutkimus käyttäen laajempaa aineistoa, ja selvitetään voiko tutkimuksen menetelmää kehittämällä muodostaa paremmin ohjelmointivirheiden korjausaikaa ennustavan mallin. Replikointia ja eksploratiivista otetta on yhdistänyt myös Assar, Borg ja Pfahl (2016), mutta tutkimusaluetta vaivaa silti replikaatiotutkimusten puute.

Virhekorjausten valmistumisaikaa on yritetty arvioida louhimalla tietoa tikettijärjestelmistä, ja luomalla tilastollisia malleja koneoppimisen avulla. Tutkimuksissa saadut tulokset ovat kuitenkin olleet välillä ristiriitaisia (esim. Guo ym. 2010 ja Bhattacharya ja Neamtiu 2011), eivätkä useinkaan erityisen tarkkoja. Varsin usein aineistona on käytetty suosittuja avoimen lähdekoodin projekteja, jotka tarjoavat kehitystyöhön liittyvää dataa julkisesti. Kuitenkin vain muutamat tutkimukset hyödyntävät laajasti eri kokoisia projekteja. Vaikuttaisi siis siltä, että lisätutkimukselle on tarvetta, mikäli näitä malleja halutaan soveltaa käytännössä osana ohjelmistokehitystä.

Tutkielman luvussa 2 esitellään avoimen ohjelmistoprojektin piirteitä ja virheenhallinnan

keinoja. Luvussa 3 käydään läpi tiedonlouhimisen ja koneoppimisen periaatteita esitellen tutkimuksen empiirisessä vaiheessa käytettyjä menetelmiä. Tutkimuksen kulku käydään läpi luvussa 4 ja tulokset luvussa 5. Luku 6 sisältää pohdintaosan ja ajatuksia tulevaisuuden tutkimusnäkymistä.

2 Ohjelmistokehitys avoimen lähdekoodin projekteissa

Avoimen lähdekoodin ohjelmia hyödynnetään paljon osana tietojärjestelmiä. Esimerkiksi LAMP-kokoonpano on suosittu web-ohjelmistojen kehityksessä. Osa avoimen lähdekoodin ohjelmistoista, kuten Gimp[1], on tarkoitettu suoraan loppukäyttäjälle. Avoimen lähdekoodin ohjelmistot ovat yhdistetty johonkin lisenssiin, joka määrittää niiden käyttö- ja jakeluehdot.

Avoimen lähdekoodin ohjelmistojen käyttöä edistävä Open Source Initiative (OSI) -järjestö on määritellyt kriteerit avoimen lähdekoodin ohjelmille ja lisensseille (Open Source Initiative 2007). Kriteerit täyttävää ohjelmaa ja sen lähdekoodia saa vapaasti jakaa, muokata ja käyttää sellaisenaan tai osana muita ohjelmistoja. Vapaan lähdekoodin ohjelmiston lisenssi ei voi kriteerien mukaan rajoittaa ohjelman käyttöä tarkoituksen tai käyttäjän perusteella. Kriteerit sallivat ohjelmiston kopioiden myymisen. Avoimen lähdekoodin ohjelmistoihin erikoistuneen WhiteSourcen selvityksen mukaan suosituimpia avoimen lähdekoodin lisenssiä olivat Apache 2.0, MIT, ja GPL v3 (Johnson 2021).

Corblyn (2014) mukaan ilmaisohjelmat, eli freeware-ohjelmat, eivät usein täytä avoimen lähdekoodin ohjelmiston kriteereitä, sillä vaikka niitä saa jakaa maksutta, usein niiden lähdekoodia ei ole julkistettu. Ilmaisohjelmien tekijä tai tekijänoikeuksien haltija on myös usein rajoittanut ohjelman käyttöä kaupallisissa tarkoituksissa.

Avoimen lähdekoodin ohjelmistoja kehitetään usein yhteisöllisesti vapaaehtoisvoimin. Kehitykseen osallistuu ydinhenkilöiden lisäksi loppukäyttäjät vaihtelevin panoksin. Motivaation lähteinä projekteihin osallistujilla voi olla esimerkiksi työstä itsestään saatu sisäisesti motivoitunut tyydytys, velvollisuudentunto yhteisöä kohtaan, tai siitä saatu rahallinen korvaus (Lakhani ja Wolf 2005). Myös kaupalliset ohjelmistoyritykset voivat osallistua avoimen lähdekoodin ohjelmistojen kehitykseen.

1. https://www.gimp.org/

2.1 Projektin kehityskaari

Avoimen lähdekoodin ohjelmistoprojektien alullepano voi tapahtua monin eri tavoin. Conlon (2007) perehtyi tutkimuksessaan nimekkäiden avoimen lähdekoodin projektien alkuun, kehittäjäyhteisöön ja johtamiseen. Osa projekteista oli aloitettu rakentamalla uutta ohjelmistoa vanhan hylätyn ohjelmiston päälle. Muutaman ohjelmiston juuret olivat yliopistoprojekteissa. OpenOffice.org julkaistiin avoimena lähdekoodina yrityskauppojen seurauksena. MySQL oli yhtä aikaa sekä avoimen lähdekoodin projekti että kaupallisen yrityksen omistuksessa. Linux muuttui yhden henkilön projektista avoimen lähdekoodin ohjelmistoksi Usenet-ryhmässä julkaistun ilmoituksen myötä.

Forkkaus, eli projektin haarautuminen useaksi itsenäiseksi projektiksi on myös mahdollinen alku uudelle avoimen lähdekoodin projektille. Nymanin ja Lindmanin (2013) mukaan forkkaus tuo kestävyyttä ja turvaa ohjelmiston jatkuvuudelle ohjelmiston, yhteisön ja ekosysteemin tasoilla. Heidän mukaansa ohjelmistotasolla forkkaus tuo turvaa koodin mätänemistä ja vanhenemista vastaan. Yhteisötasolla projektin haarautuminen mahdollistaa itsenäisen projektin luomisen siinä tapauksessa, jos alkuperäisen projektin suunta tai vetäjien päätökset eivät miellytä yhteisöä. Ekosysteemitasolla forkkaus mahdollistaa Nymanin ja Lindmanin mukaan hylättyjen projektien henkiinherättämisen ja parantaa innovaatioiden mahdollisuutta projektien yhdistämisen ja muokkaamisen kautta. Esimerkiksi Linux jakeluista on forkattu lukuisia haaroja, jotka vastaavat erilaisiin tarpeisiin ja mieltymyksiin.

Laurentin (2004) mukaan forkkausta myös pelätään, koska se voi hajottaa kehittäjäyhteisön kahtia, kuten kävi Emacs projektille vuonna 1993. Vakavimmillaan alkuperäinen projekti voidaan hylätä. Näin kävi Viseurin (2012) tutkimuksessa 19%:lle tapauksista. Laurentin mukaan avoimen lähdekoodin “copyleft” lisensseillä pyritään joskus rajoittamaan forkkausta muun muassa vaatimalla yhteensopivaa lisenssiä myös haarautuville projekteille. Viseurin (2012) havaintojen mukaan tämä ei kuitenkaan ole erityisen varma tapa, sillä yli puolet tutkimuksessa mukana olleista haaroista oli lisensoitu “copyleft”-lisenssillä. Toinen tapa rajoittaa forkkausta on alkuperäisen projektin nimen suojaaminen. Tällä tavalla Apache-projekti on säilyttänyt selkeän kehityslinjan. Suljetun lähdekoodin projekteissa forkkausta ei juuri tapahdu, sillä ne eivät salli lähdekoodin uudelleenkäyttöä toisissa projekteissa (Conlon 2007).

Projektin alullepanon jälkeisiä vaiheita ovat vapaaehtoisten haalinta, ratkaisujen identifioiminen, kehitys ja testaaminen, vertaisarviointi, dokumentointi ja julkaisun hallinta (Sharma, Sugumaran ja Rajagopalan 2002). Koska modernit ohjelmistoprojektit ovat luonteeltaan iteratiivisia ja koska uusia vapaaehtoisia voi liittyä avoimen lähdekoodin projektin kehitysyhteisöön koska vain, voisi näiden vaiheiden nähdä toistuvan ja tapahtuvan ainakin osin yhtä aikaa.

Uusien kehittäjien haalimiseksi Riehle (2015) esittää projektin näkökulmasta kaksi väylää: aktiivisen etsimisen ja passiivisen sisäänvirtauman. Aktiivista etsimistä suoritetaan Riehlen mukaan aktiivisella näkyvällä viestinnällä asiaankuuluvissa mediakanavissa. Nykyään ohjelmistoalalla toimivien suosimat sosiaalisen median alustat voisi nähdä hyvänä väylänä tähän. Passiivista vapaaehtoisten haalintaa on Riehlen mukaan esimerkiksi hakukoneiden tukeminen ja avoimen lähdekoodin projekteja listaaviin portaaleihin kirjautuminen. Riehle nostaa esille myös projektin esittelyn ja dokumentoinnin merkityksen, kun uusi käyttäjä tai mahdollinen kehittäjä tutustuu projektiin. Dokumentaation, tutoriaalien kirjoittaminen ja käytäntöjen selkeyttäminen auttoi myös Mozilla-projektia keräämään osallistujia alun vaikeuksien jälkeen (Mockus, Fielding ja Herbsleb 2002).

Avoimen lähdekoodin projektien suosimat alustat versiohallintajärjestelmille, kuten GitHub, mahdollistavat ohjelmiston kehittämisen hajautetusti. Lähdekoodin kirjoittaminen ja testaaminen onnistuvat lokaalissa ympäristössä, ja muutostiedostoja voidaan antaa yhteisön tarkasteltavaksi pilvipalvelun kautta. Sharman, Sugumaranin ja Rajagopalanin (2002) mukaan kehitystyötä voidaan tehdä joko yksin tai osana väliaikaista tiimiä. Ennen muutostiedoston hyväksymistä sen tulee käydä läpi vertaisarviointi, jossa muut projektiin osallistuvat katselmoivat lisättävän koodin sen laadun varmistamiseksi ja antavat siitä palautetta. Vertaisarviointikäytänteet vaihtelevat projekteittain, ja joskus luotetun kehittäjän kirjoittama koodi voidaan lisätä projektiin jo ennen katselmointia (Wang ym. 2015; Rigby, German ja Storey 2008).

2.2 Projektin hallinta

Hyvin menestyneet avoimen lähdekoodin ohjelmistoprojektit houkuttelevat paljon osallistujia. Jossain vaiheessa kasvaneen projektin ohjaamisen, päätöstenteon ja vastuiden jakamisen tueksi syntyy hallinnan järjestelmä. Müller (2009) määrittelee hallinnan läpinäkyvyyteen, vastuullisuuteen ja määriteltyihin rooleihin pohjaavaksi viitekehykseksi päätösten ja toimenpiteiden tekemiselle. Müllerin mukaan se muodostuu arvojärjestelmistä, vastuista, prosesseista ja käytännöistä, jotka auttavat projektia kohti sen tavoitetta. Se myös asettaa rajat mahdollisille johtamistoimille ja selkeyttää eroa omistajuuden ja työtehtävien kontrollin välillä (Müller 2009). Markus (2007) määrittelee avoimen lähdekoodin projektinhallinnan keinoiksi suunnata ja koordinoida kehitysprojektiin osallistuvia itsenäisiä yksilöitä ja organisaatioita. Erityisenä huomiona voi todeta, että näitä keinoja ei välttämättä ole formaalisti dokumentoitu, vaan ne voivat toteutua myös informaalisti jaettujen normien ja sosiaalisen kontrollin kautta tai toteutua käytetyn teknologian, kuten versiohallintajärjestelmän, kautta (Markus 2007). Projektinhallinta siis vaikuttaa sekä korkean tason päätöksentekoon, että yksittäisten ohjelmistoa koskevien ongelmien ratkaisuun.

Onnistunut projektinhallinta edesauttaa projektia saavuttamaan päämääränsä menestyksekkäästi. Samset ja Volden (2016) esittävät, että projektin menestystä strategisella tasolla voidaan mitata tarkastelemalla projektin relevanttiutta, efektiivisyyttä ja kestävyyttä, jolloin projekti vastaa yhteisön tarpeisiin. Heidän mukaansa taktisen tason menestyksen mittareita ovat projektin tulosten hinta, laatu, ja käytetty aika, joita voidaan verrata projektin sisäisiin tavoitteisiin. O'Mahony (2007) nostaa kirjallisuudesta kolme avoimen lähdekoodin projektien ongelma-aluetta, joihin projektinhallinnan keinoin voidaan vastata. Hänen mukaansa se voi olla osana ratkaisua kollektiivisen toiminnan ongelmiin, kuten vapaaehtoistyöhön motivoimiseen ja osallistumiseen kannustavien mekanismien luomiseen. Projektinhallinnan keinoin ratkaistaan myös yleisiä kehitystyön aikaisia ongelmia, kuten keskinäisten riippuvuuksien ja laadun hallinta ja vapaaehtoisten rekrytointi. Kolmantena ongelma-alueena O'Mahony tuo esille projektiyhteisön ilmapiirin luomisen. Yhteisön jäseniä miellyttävät hallintakeinot voivat hänen mukaansa nousta työskentelyn motivaation lähteeksi muiden työn tuloksena ilmenevien itseis- ja välinearvojen joukkoon.

De Laat (2007) erittelee kuusi kategoriaa, joihin avoimen lähdekoodin ohjelmistoprojektien

sisäisen hallinnan työkalut voidaan jakaa: modularisointi, roolien jakaminen, päätöksenteon delegointi, koulutus ja indoktrinaatio, formalisointi ja demokraattisen tai autokraattisen hallintomuodon painotus.

Modularisoinnilla tarkoitetaan ohjelmiston jakamista pienempiin itsenäisiin osiin. Samaan aikaan voidaan kehittää myös saman ohjelmiston eri versioita. Modularisoinnin avulla ohjelmiston osien, ja samalla myös kehittäjien, keskinäinen riippuvuus vähenee, eikä kehittäjän tarvitse tuntea koko järjestelmää osallistuakseen moduulin kehitykseen (Lee ja Cole 2001). Myös projektin jatkuvuus paranee, koska uusien kehittäjien on helpompaa liittyä yhteisöön, kun osallistumisen voi aloittaa helpommin ymmärrettävistä moduuleista (Aberdour 2007; Lee ja Cole 2001).

Roolien jakamisella de Laat (2007) tarkoittaa eri tasoisten osallistumisoikeuksien jakamista osallistujille pelkästä tarkkailuoikeudesta lähdekoodin vapaaseen muokkausoikeuteen. Oikeuksien lisäksi roolien perusteella jaetaan myös vastuualueita, joita voivat olla esimerkiksi kääntäjä ja ylläpitäjä (de Laat 2007). Rooleja ei ole aina formaalisti määritelty, vaan usein esimerkiksi virheraportteja tai -korjauksia voi luoda kuka tahansa asiasta kiinnostunut. Yuan ym. (2010) havaitsivat, että suosituissa avoimen lähdekoodin projekteissa osallistujat käyttivät ainakin kolmea eri roolia. He arvelivat, että menestyksekkäissä projekteissa työnjako on onnistunut, mihin on osaltaan vaikuttanut vapaaehtoisten roolittaminen. Projektiin osallistuva voi olla yhtä aikaa monessa roolissa, ja osallistujan rooli voi myös muuttua yhteisössä ja yhteisön ulkopuolella tapahtuvan kehityksen myötä (Trinkenreich ym. 2020). Tutkimuksessa usein käytetty malli avoimen lähdekoodin projektien roolien jakautumiselle on sipulimalli, missä projektiin osallistuvat jaetaan suurimman osan ohjelmoinnista tekeviin ydinkehittäjiin, satunnaisia palasia kirjoittaviin oheiskehittäjiin ja projektin vetäjiin (Crowston ja Howison 2005; Mockus, Fielding ja Herbsleb 2002). Sipulimallia on käytetty paljon avoimen lähdekoodin projektien tutkimuksessa, sillä se jakaa projektin kehittäjät mitattavalla tavalla kahtia (esim. Setia ym. 2012; Alfayez ym. 2017).

Roolit jakautuvat myös päätöksenteko-oikeuden mukaan, jolloin de Laatin (2007) mukaan hallinnan työkaluna on päätöksenteon delegointi. Projektilla voi olla esimerkiksi yksi päävetäjä, joka tekee koko ohjelmistoa koskevia päätöksiä, ja useita ylläpitäjiä jotka vastaavat itsenäisten moduulien vetämisestä. Koko projektia ja yhteisöä koskevien päätösten lisäksi

de Laat näkee keskeiseksi päätöskohteeksi uuden lähdekoodin hyväksynnän. Päätöksentekoa voidaan keskittää tai hajauttaa. Keskitetyssä mallissa yksittäiset vetäjät tekevät päätökset lähdekoodin hyväksynnästä, kun taas hajautetussa mallissa kehittäjät päättävät yhdessä mukaan otettavasta lähdekoodista. Päätöksenteon tukena voidaan käyttää esimerkiksi vertaisarviointia ja äänestystä. Heckman ym. (2007) havaitsivat, että onnistuneissa avoimen lähdekoodin projekteissa päätöksenteko ja siihen liittyvä keskustelu siirtyi projektin ylläpitäjältä enemmän yhteisön vastuulle, kun projekti siirtyi alkuvaiheesta eteenpäin. Päätöksentekijöitä siis ei ole aina määritelty, vaan päätöksenteko sallitaan yhteisöstä riippuen eri tavalla mukana olleille. Myöskään päätöksen tekeminen ryhmässä ei aina ole suoraviivainen prosessi, vaan siinä voi olla useita arviointi ja kehittelyvaiheita (Eseryel, Wie ja Crowston 2020).

Luomalla yhteisöön kriteerit sille, kuka voi osallistua ja ehdottaa muutoksia lähdekoodiin, voidaan de Laatin mukaan pyrkiä parantamaan ohjelmiston laatua ja luotettavutta. Koulutuksella ja indoktrinaatiolla pyritään tarjoamaan halukkaille osallistujille mahdollisuus kehittää omaa osaamistaan sille tasolle, että se saavuttaa projektin asettamat osallistumiskriteerit (de Laat 2007).

Ohjelmiston ja projektin kehitystehtävien ja raportoinnin formalisointiin on kehitetty työkaluja, joita avoimen lähdekoodin projekteissa on otettu laajasti käyttöön. De Laatin (2007) mukaan vakiinnutetut tavat ja työkalut helpottavat suurten kokonaisuuksien hallintaa sekä osallistujien siirtymistä projektien välillä. Avoimen lähdekoodin projekteissa tärkeimpiä työkaluja ovat Git ja GitHub. Git on hajautettu versiohallintajärjestelmä, joka tarjoaa työkaluja kehittäjille ja ylläpitäjille. GitHub on alusta Gitiä käyttäville projekteille, joka koodin säilyttämisen lisäksi tarjoaa paljon työkaluja yhteisölliseen kehittämiseen ja raportointiin. Tällaiset teknologisten välineiden myötä projekteihin kantautuvat formaalit prosessit kenties toimivatkin paremmin, kuin tarkkaan dokumentoidut säännöt ja ohjeet osallistumiselle, sillä ne skaalautuvat automaattisesti projektin kasvun mukana. Schweik ja English (2007) huomasivatkin avoimen lähdekoodin projektien vetäjiä haastatellessaan, että osallistujat välttivät formaalien protokollien kirjoittamista, ja luottivat sosiaalisen kanssakäynnin mukana muodostuviin informaaleihin konventioihin.

Viimeisenä projektinhallinnan keinona de Laat huomioi projektiyhteisön rakenteen ja päätöksenteon painottumisen autokraattiseen tai demokraattiseen suuntaan. Hän kertoo, että pie-

nemmissä projekteissa projektin aloittaja usein pitää itsellään päätäntävallan ja johtoaseman projektin suunnasta, mutta suureksi kasvaneissa projekteissa vetäjien valinta voidaan hoitaa demokraattisemmin, jolloin yhteisön jäsenet voivat vaikuttaa projektin johtohahmoihin. Projektin vetäjäksi voidaan valita yksi henkilö tai useasta jäsenestä koostuva ryhmä. Toisaalta esimerkiksi Apache-projektia johtaa vaaleilla valittu lautakunta ja projekti kertookin vastuun antamisen perustuvan meritokraattiseen, näyttöihin perustuvaan, luottamukseen (The Apache Software Foundation 2021). Demokraattisen tai autokraattisen hallintotavan valinnalla voisi ajatella olevan vaikutusta päätöksenteon nopeuteen ja osaamiseen, ulkopuolisiin suhteisiin, roolien jakamisen tärkeyteen sekä ohjelmiston hyväksyntään muissa yhteisöissä. Avoimen lähdekoodin projektin kehitystä autokratiasta kohti demokratiaa ja meritokratiaa projektin kasvun myötä tukee myös O'Mahonyn ja Ferraron (2007) havainnot. He tutkivat Debian-projektin alullepanijan poistumisen myötä syntynyttä prosessia, jonka päätteeksi projektiin syntyi henkilöstä riippumattomia hallintotason instituutioita. Heidän havaintojen mukaan demokraattisestikaan valittujen projektin vetäjien ei odoteta tekevän päätöksiä omin päin, vaan toimivan konsensuksen rakentajina ohjelmiston kehitystä koskevissa keskusteluissa.

Edellä mainituista kuudesta projektin sisäisen hallinnan kategorioista kaikkia voi hyödyntää samassa projektissa, ja jotkin työkalut voikin nähdä vaikuttavan projektin hallintaan useamman kategorian näkökulmasta.

2.3 Vertaisarviointi

Kun ohjelmiston lähdekoodia muokataan, on usein kyse joko siitä että ohjelmaan tuodaan uusia ominaisuuksia lisäämällä uutta koodia, tai korjataan ohjelmointivirheitä muokkaamalla olemassa olevaa ohjelmakoodia. Ennen ohjelmointia projektin tehtävälistalle tai virheenseurantaohjelmaan luodaan tiketti, jossa kuvataan kehitettävä ominaisuus tai ohjelmistossa havaittu virhe. Tiketin yhteyteen voi myös liittyä keskustelua kyseisestä tehtävästä. Tehtävätiketin verifiointi, luokittelu, priorisointi ja delegointi tapahtuu triage-vaiheessa (Akila, Zayaraz ja Govindasamy 2014). Verifioinnin tarkoituksena on selvittää, onko raportoitu virhe toistettavissa tai onko virhe kyseisen ohjelman sijaan jossain muualla, esimerkiksi loppukäyttäjän tekemä virhe. Luokittelemalla voidaan tehtävien osoittaa liittyvän tiettyyn ohjel-

miston osaan, jolloin vastuu tehtävän hoitamisesta voidaan siirtää kyseistä osaa kehittäville henkilöille. Priorisoinnilla pyritään järjestämään tehtävät niiden tärkeyden ja kriittisyyden perusteella. Korkeamman prioriteetin tehtävät halutaan hoitaa ennen pienemmän prioriteetin tehtäviä. Koska avoimen lähdekoodin projekteihin osallistutaan usein vapaa-ajalla, tehtävien delegointi tapahtuu usein kehittäjien omien mielenkiinnon- ja osaamiskohteiden sekä käytettävissä olevien resurssien perusteella ja kehittäjät usein delegoivat tehtäviä itselleen oma-aloitteisesti (Crowston ym. 2007). Triagen voi hoitaa joko yksi projektiin perehtynyt kehittäjä, tai ryhmä kehittäjiä. Kun uusi tai muokattu lähdekoodi on hyväksytty osaksi ohjelmistoa, luotu tiketti voidaan sulkea. Lähdekoodin hyväksyminen tapahtuu vertaisarvioinnin kautta.

Vertaisarviointi ja koodin katselmointi ovat keskeisimpiä menetelmiä ohjelmiston laadun parantamiseksi ja virheiden korjaamiseksi avoimen lähdekoodin projekteissa (Rigby ym. 2014). Katselmoinnissa ohjelmiston kehittäjät katselmoivat ja testaavat toisten kehittäjien lisättäväksi ehdottamaa koodia. Vertaisarvioinnin tavoitteena on löytää ohjelmointivirheitä ja muita koodin puutteita mahdollisimman aikaisin. Rigby ym. (2014) kuvaavat avoimen lähdekoodin projekteissa yleistä broadcast -tyylisen vertaisarviointikäytännön vaiheita seuraavasti:

1. Projektin osallistuja toteuttaa muutoksen koodiin (*patch*).
2. Osallistuja julkaisee muutostiedoston projektin käyttämää väylää pitkin, jolloin sen näkee suuri joukko siitä mahdollisesti kiinnostuneita kehittäjiä.
3. Julkaistu muutosehdotus joko jää huomiotta, tai se katselmoidaan ja siitä annetaan palautetta julkaisijalle samaa julkista väylää pitkin.
4. Muutoksesta käydään keskustelua ja sitä parannellaan, kunnes se lopulta hylätään tai hyväksytään osaksi ohjelmiston lähdekoodia.

Yhtenä periaatteena, jonka johdosta avoimen lähdekoodin projekteissa tapahtuva vertaisarviointi parantaa ohjelmiston laatua, voidaan pitää Raymondin (1999) muotoilemaa Linuksen lakia, jonka mukaan kaikki virheet löytyvät ja korjaantuvat, kunhan ohjelmaa tarkastelee tarpeeksi moni ihminen. Linuksen lain perusteella avoimet ohjelmistoprojektit hyötyvät laajasta yhteisöstä. Toisaalta monimuotoisuus yhteisön sisällä voi aiheuttaa haasteita vertaisarvioinnin kanssa esimerkiksi näkemys- ja arvoristiriitojen kohdalla tai jos teknisesti syvällisesti perehtyneet kehittäjät eivät löydä yhteistä kieltä loppukäyttäjien kanssa (Wang, Shih ja Carroll

2015).

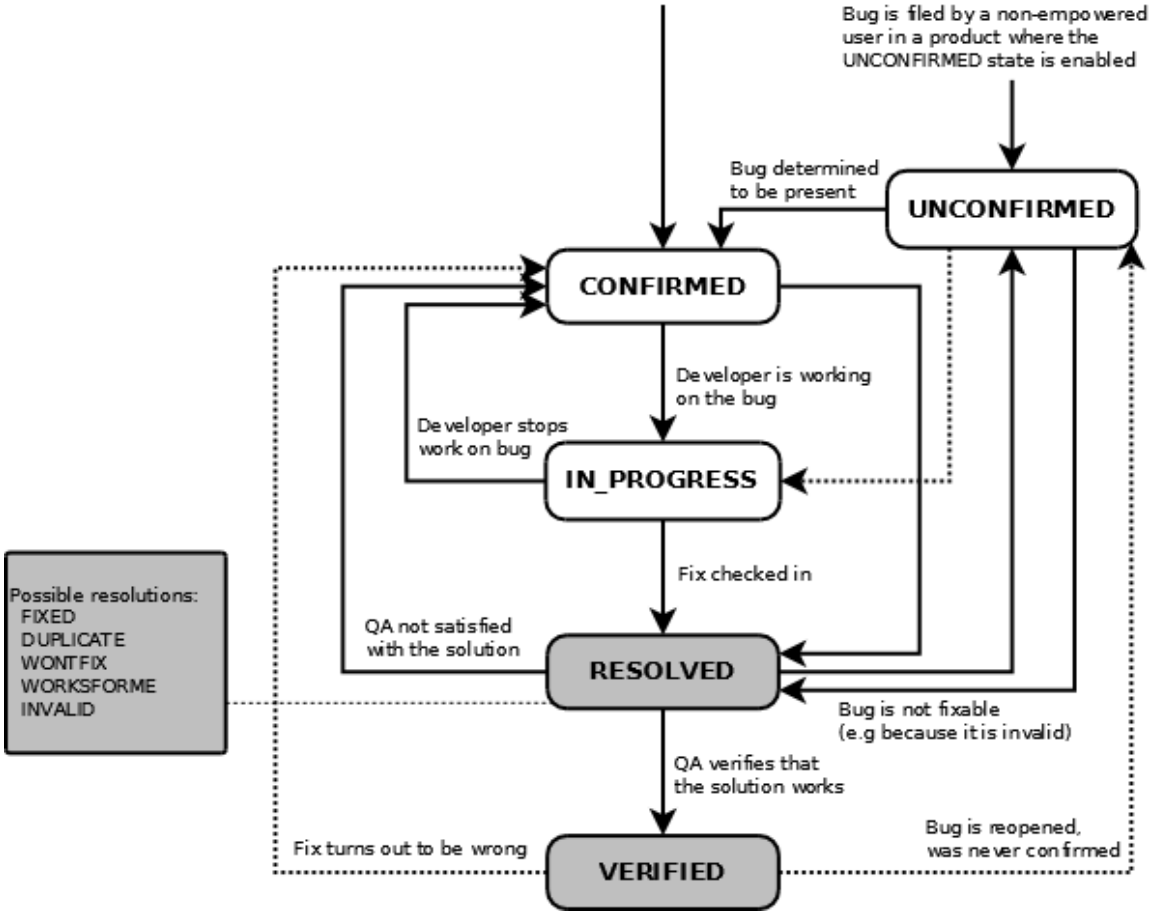
Rigby ym. (2012) esittävät, että avoimen lähdekoodin projekteissa tapahtuvalle hajautetulle vertaisarvioinnille yleisiä piirteitä on mm. asynkronisuus, katselmointien tiheys, katselmoitavien muutosten pieni koko ja katselmoijien erityisosaaminen tarkasteltavaan koodiin liittyen. Heidän mukaansa aikatauluista riippumattoman asynkronisen arviointimallin johdosta koodikatselmointia varten ei tarvitse löytää kaikille osallistujille sopivia aikoja, vaan sen voi suorittaa arvioijalle itselleen sopivimmalla hetkellä. Usein ja pienissä paloissa tehtynä katselmointi ei ole työlästä vapaaehtoisena toimiville osallistujille, ja lisättävä koodi on mahdollista tarkastaa huolellisesti. Katselmoinnin laatu paranee, kun siihen osallistuu muutettavan ohjelmiston osan hyvin tuntevia tekijöitä, jotka saavat itse valita arvioitavat muutokset (Rigby ym. 2012; Rigby ym. 2014).

2.4 Virheiden raportointi ja elinkaari

Koska virheiden etsiminen ja niiden raportointi on osa vertaisarviointia, myös ohjelmien loppukäyttäjät osallistuvat usein vertaisarviointiin raportoimalla löytämiään ohjelmistovirheitä, vaikka eivät itse muuten osallistuisi niiden korjaamiseen tai ohjelmiston kehittämiseen. Virheiden raportointi- ja raporttien käsittelykäytänteet vaihtelevat projektien välillä. Esimerkiksi Wang ym. (2015) havaitsivat Mozilla- ja Python -projekteja vertaillessaan, että Mozilla-projektiin raportoidut virheet kuvattiin usein käyttökokemuksen avulla, kun taas Pythonin virheraportit sisälsivät monesti raportoijan kirjoittaman korjauspaketin lähdekoodiin. Tämä johtui siitä, että Mozilla-projektin virheraportit olivat usein loppukäyttäjien tekemiä, kun taas Python-projektin virheitä raportoivat olivat monesti myös kehittäjiä. Raporttien verifioinnissa ja kommentoinnissa nousi Mozilla-projektissa esille riittävän tarkan informaation kerääminen, ja Python-projektissa keskityttiin enemmän korjauspaketin katselmointiin ja testaukseen. Mozilla-projektissa raporttien luokittelu oli tarkempaa ja käytettäviä nimikkeitä oli huomattavasti enemmän.

Bugzilla-virheenseurantaohjelmassa käytetty kehys virheraporttien elinkaarelle sisältää viisi tilaa, jotka vaihtuvat korjauksen edetessä. Bugizillan elinkaarimallia (Kuvio 1) on usein käytetty lähtökohtana virheitä käsittelevissä tutkimuksissa. Virheraportin luomisen jälkeen

triage-vaiheessa sen joko vahvistetaan olevan todellinen tai hylätään. Raporttiin kerätään riittävästi tietoa, jotta kehittäjät voivat löytää virheen syyn ja korjata sen. Kehittäjän tekemä korjaus täytyy verifioida, jotta virheraportti voidaan saattaa lopulliseen tilaan.



Kuvio 1: Virheraportin elinkaari Bugzilla-ohjelmistossa (“Bugzilla Documentation” 2021)

Virheraportit sisältävät sekä strukturoitua tietoa kuten virheen prioriteetti ja virheen sisältävä ohjelmiston versio, että strukturoimatonta tietoa kuten virheen kuvaus ja raportista käyty keskustelu (Karim 2019).

GitHubin tarjoama työkalu raporttien kirjaamiselle (*issues*) on melko joustava, ja nimikkeet ovat sen ainoa keino luokitteluun ylläpitäjien määräämien kenttien perusteella. Sen lisäksi issues-työkalua käytetään muidenkin kehitystehtävien kirjaamiseen. Onkin siis syytä olettaa, että tässä tutkimuksessa käytettävien avoimen lähdekoodin ohjelmistoprojektien välillä on eroja ohjelmistovirheiden raportoinnissa ja vertaisarvioinnin käytännöissä.

3 Tiedonlouhinta ja koneoppiminen

Moderni yhteiskunta tuottaa ja tallentaa dataa jatkuvalla virralla. Pelkkä data itsessään ei vielä ole tietoa tai tietämystä. Jotta siitä saataisiin ymmärretävää ja kiteytynyttä tietoa, se vaatii tulkintaa ja yhdistämistä muuhun ympäristössä olevaan tietoon. Informaatiojärjestelmiin kerättyä ja niiden itse keräämää dataa on usein niin paljon, ettei sitä ole mahdollista käsitellä manuaalisin menetelmin. Tiedonlouhinnan (eng. *data mining*) voi määritellä oleellisen tiedon ja kiinnostavien kaavojen löytämiseksi suuresta datamassasta (Han, Pei ja Kamber 2011, 8). Tiedonlouhintaa hyödynnetään laajasti monipuolisilla sovellusalueilla, kuten päätöksenteon ja ennusteiden tukena liiketoiminnassa ja markkinoinnissa, hakukoneissa, tietoturvassa ja yksityisyydensuojassa, valtioiden tiedustelussa, teollisuudessa ja tieteessä (Kumar ja Bhardwaj 2011; Han, Pei ja Kamber 2011; Fayyad, Piatetsky-Shapiro ja Smyth 1996) Alalla käytetään monenlaisia tilastotieteen, matematiikan ja tietojenkäsittelyn tekniikoita (Clifton 2019). Vaihtoehtoisesti tiedonlouhinnan voi määritellä yhdeksi vaiheeksi tietämyksen keräämisen prosessissa. Fayyad, Piatetsky-Shapiro ja Smyth (1996) määrittelevät tietämyksen löytämisen datasta (eng. *knowledge discovery in databases, KDD*) kokonaisprosessiksi, jossa tiedonlouhinta tarkoittaa tiettyjen algoritmien soveltamista kaavojen ja säännönmukaisuuksien löytämiseksi datasta. He myös painottavat sitä, että prosessi ei ole vain yksioikoinen laskentatoimitus, vaan siihen kuuluu epätriviaalia tutkimista ja päättelyä. Tässä tutkielmassa tiedonlouhinnan ymmärretään Hanin, Pein ja Kamberin (2011) määritelmän mukaisesti tarkoittavan koko tiedonkeruun prosessia aineiston esikäsittelystä muodostetun tietämyksen esittämiseen ja jakamiseen. Heidän mukaansa tämä laajempi tulkinta tiedonlouhinnan määritelmästä on yleinen teollisuudessa, mediassa ja tutkimuskentällä.

Fayyad (1997) näkee tiedonlouhinnan vastaavan kolmeen ongelmaan, jotka vaikeuttavat perinteistä data-analyysiä suurilla tietomassoilla. Ensinnäkin tiedonlouhinnan menetelmät mahdollistavat monimutkaisten ilmiöiden löytämisen ja tunnistamisen ilman vaikeasti muotoiltavia tietokantaan kohdistuvia kyselylauseita. Esimerkiksi luokittelu- tai klusterointimenetelmillä voidaan tietokannan tietueet jaotella käyttäjää kiinnostavalla tavalla. Ohjatun oppimisen menetelmiä käyttäen voidaan manuaalisesti luokitella osa tietueista, ja käyttää niistä louhittua tietoa loppujen tietueiden automaattiseen luokitteluun. Toisekseen Fayyad kertoo

tiedonlouhinnan helpottavan suuren ja moniulotteisen datamassan visualisointia ja ymmärrettävyyttä. Hänen mukaansa tiedonlouhintamenetelmin on mahdollista löytää keskeiset dataa määrittävät ulottuvuudet, jolloin helposti piiloon jäävät kaavat ja säännönmukaisuudet voidaan löytää. Kolmantena ongelmana Fayyad kertoo tiedonlouhinnan vastaavan kasvavan datamäärän tuottamaan skaalausongelmaan. Hänen mukaansa tiedonlouhinnan menetelmät vastaavat tehokkaasti nopeaan tiedontarpeeseen, kun perinteiset menetelmät eivät pysty enää tuottamaan tuloksia tarpeeksi nopeasti. Painetta nopeaan tietoon on niin liike-elämässä, tieteessä kuin valtiotasolla.

Fayyad, Piatetsky-Shapiro ja Smyth (1996) esittävät viisivaiheisen tiedonlouhinnan mallin, jossa lähtöpisteenä on raakadata ja lopputuloksena tietämys. He näkevät tiedonlouhinnan koskevan matalan tason raakadatan muokkaamista tiiviimpään, yleisempään tai hyödyllisempään muotoon. Heidän käsityksensä mukaan raakadatan kerääminen ei siis ole osa tiedonlouhintaprosessia. Ensimmäisessä vaiheessa raakadatasta valitaan tiedonlouhintatehtävän kannalta oleellinen osa tietueista ja tietokentistä, joista muodostuu kohdedata (eng. *target data*). Toisessa vaiheessa kohdedataan toteutetaan esiprosessointia, jossa voidaan tarpeen mukaan esimerkiksi vähentää datan kohinaisuutta ja tehdä toimenpiteitä puuttuvien tietokenttien korjaamiseksi. Kolmannessa vaiheessa esikäsitelty data muokataan valitulle tiedonlouhinta-algoritmille sopivaksi. Muokkauskeinoja voi olla esimerkiksi datan dimensioiden vähentäminen ja vaihtoehtoisten esitystapojen tunnistaminen. Neljäntenä vaiheena on tarkoituksenmukaisen tiedonlouhinta-algoritmin soveltaminen dataan. Algoritmin soveltamisen tuloksena voi olla esimerkiksi tutkittavaa ilmiötä ennustava tai selittävä malli tai kaava. Viidentenä vaiheena on kehitetyn mallin tai kaavan tulkinta ja arviointi, jonka avulla voidaan muodostaa uutta tietämystä tutkittavasta ilmiöstä. Fayyad, Piatetsky-Shapiro ja Smyth (1996) tuovat esille myös tiedonlouhintaprosessin iteratiivisen luonteen: prosessin vaiheet eivät välttämättä seuraa toisiaan suoraviivaisesti, vaan prosessin aikana saadun ymmärryksen johdosta voi olla tarvetta palata uudestaan jo käytyihin vaiheisiin paremman lopputuloksen toivossa.

Hanin, Pein ja Kamberin (2011) esittämä seitsenvaiheinen tiedonlouhintaprosessin malli on hieman Fayyadin, Piatetsky-Shapiron ja Smythin mallia laajempi. Myös se ottaa lähtökohdaksi tapaan valmiin suuren datamassan, josta jalostetaan tietämystä. Ensimmäinen vaihe on datan siivoaminen, eli kohinan ja epäkonsistentin datan poisto. Toisena vaiheena on datan in-

tegrointi, missä mahdollisesti useammasta lähteestä saatava data tuodaan yhteen tietovarastoon. Kolmantena vaiheena on kohdedatan muodostaminen. Neljäntenä vaiheena data muokataan tiedonlouhintaa varten sopivaan muotoon. Viidentenä vaiheena toimii tiedonlouhintamenetelmien soveltaminen dataan. Kuudennessa vaiheessa löydettyjä kaavoja arvioidaan ja pyritään tunnistamaan kiinnostavat tulokset. Viimeiseksi seitsemännessä vaiheessa luotu tietämys esitetään ja jaetaan sopivassa muodossa, kuten visualisoinnin avulla. Myös Han, Pei ja Kamber tuovat esille prosessin iteroivan luonteen.

Tiedonlouhintaprosessin aikana hyödynnetään tekniikoita useilta tiedonkäsittely- ja tilastontieteen aloilta. Tilastotieteellisiä metodeja voidaan käyttää datan esikäsittelyyn, kaavojen ja säännönmukaisuuksien löytämiseksi, tai tiedonlouhinnan tulosten laadun ja tilastollisen merkitsevyyden arvioimiseksi (Han, Pei ja Kamber 2011).

Tiedonhaun (eng. *information retrieval*) menetelmillä pyritään löytämään tiedontarpeen kannalta kiinnostavaa materiaalia suuresta joukosta strukturoimatonta tai osittain strukturoitua tietoa (Manning, Schütze ja Raghavan 2008). Yleisin esimerkki strukturoimattomasta datasta on tekstidokumentit. Dokumentti voi tarkoittaa kontekstista riippuen eri laatuisia tekstejä, kuten muistioita tai kirjan lukuja. Tiedonhakua varten strukturoimaton data tulee mahdollisesti prosessoida esimerkiksi indeksoimalla dokumentit tai hakemalla sanojen vartalot. Tiedonlouhinnan kannalta tärkeitä tiedonhaun tekniikoita on dokumenttien samanlaisuuden vertailu ja dokumenteista löytyvien aiheiden tunnistaminen (Han, Pei ja Kamber 2011).

Tiedonlouhinnan kannalta keskeisessä osassa ovat erilaiset koneoppimisen tekniikat. Tietokoneohjelman voi nähdä oppivan kokemuksesta E, jos sen tulos jossain tehtävässä T mitattuna mittarilla P paranee kokemuksen E myötä (Mitchell 1997). Tiedonlouhimisen kannalta kokemus E tarkoittaa yleensä algoritmille annettua opetusdataa. Tarvittavan opetusdatan määrä riippuu datan laadusta, louhittavien ilmiöiden monimutkaisuudesta ja halutusta lopputuloksen laadusta. Koneoppimisen voidaan määritellä olevan tekoälyjärjestelmä, jonka rakenne muuttuu ja sen antama palaute paranee sen saaman syötteen myötä (Nilsson 1998). Koneoppimisen ja tilastotieteen menetelmät limittyvät toisiinsa, ja Witten ja Frank (2005) nostavat yhdeksi keskeiseksi eroksi näiden välillä sen, että tilastotieteen menetelmät pyrkivät usein jonkin hypoteesin testaamiseen, kun taas koneoppimismenetelmillä pyritään löytämään useiden mahdollisten hypoteesien joukosta paras yleistys.

Koneoppimisalgoritmit voidaan luokitella ohjatun ja ohjaamattoman oppimisen algoritmeiksi sen mukaan, tiedetäänkö opetusdatan näytteille oikeat vasteet etukäteen. Ohjatussa oppimisessa järjestelmä muodostaa mallin sellaisen opetusdatan perusteella, jonka näytteistä tunnetaan sekä syöte että sille oikea vaste. Esimerkiksi asiakkaita kahteen eri ryhmään taustatietojen perusteella luokitteleva algoritmi voidaan opettaa niiden asiakkaiden tiedoilla, joiden ryhmä jo tunnetaan. Ohjatun oppimisen menetelmillä voidaan ratkaista muun muassa luokittelu- ja regressio-ongelmia. Niihin sopivia tekniikoita on esimerkiksi naiivi Bayesin luokitin, päätöspuut, satunnaiset metsät, logistinen regressio, neuroverkot ja tukivektorikoneet (Han, Pei ja Kamber 2011; Manning, Schütze ja Raghavan 2008).

Ohjaamattomassa oppimisessa koneoppimisalgoritmi saa opetusdatana pelkkiä syötteitä, joista sen on tarkoitus löytää säännönmukaisuuksia (Alpaydin 2014). Han, Pei ja Kamber (2011) esittävät sen eroavan ohjatusta oppimisesta siinä, että ohjaamaton oppiminen on havaintojen kautta oppimista, kun taas ohjattu oppiminen on esimerkin kautta oppimista. Klusterointi on yleisin ohjaamattoman oppimisen tekniikka, mitä voidaan toteuttaa esimerkiksi k-means klusterointialgoritmilla (Manning, Schütze ja Raghavan 2008). Klusteroinnin lisäksi ohjaamatonta oppimista käytetään dimensionvähennysalgoritmeissa kuten pääkomponenttianalyysissä (Ghahramani 2004). Jos järjestelmän opetusdatan syötteille oikeat vasteet tunnetaan osittain, voi kyse olla esimerkiksi puoliohjatusta tai aktiivisesta koneoppimisesta (Han, Pei ja Kamber 2011).

Ohjelmistokehityksessä hyödynnetään usein hajautettuja versiohallintajärjestelmiä kuten Git, ja internet-palveluja kuten Github ja Bitbucket tietovarastojen (eng. *repository*) hallintaan. Nämä palvelut tarjoavat usein lähdekoodin jakamisen lisäksi myös muita ominaisuuksia, kuten jatkuvan integraation kanavan, vertaisarvioinnin työkaluja, ja virheraportit. Mining software repositories (MSR) on tiedonlouhinnan osa-alue, joka on erikoistunut ohjelmistokehityksessä syntyvän strukturoidun ja tekstimuotoisen datan käsittelyyn. Tutkimuksen kohteita ovat esimerkiksi koodin mätäneminen, muutosten riippuvuus, virheiden ennakointi ja syyt, ja kehittäjien yhteistyö. (Güemes-Peña ym. 2018)

3.1 Tekstin prosessointi tiedonlouhinnassa

Luonnollisella kielellä kirjoitetun tekstin käyttö informaation lähteenä on erityinen haaste osana tiedonlouhintaa, sillä se on strukturoimatonta eikä tietokoneilla ole ihmisten kaltaista kykyä tulkita sanojen ja lauseiden semantiikkaa. Tärkeimmät luonnollisten kielten prosessoinnin avulla luodut tiedonhaun teknologiat ovat verkon hakukoneet (Voorhees 2000). Virheraporttien vapaamuotoisia tekstikenttiä, kuten otsikkoa ja raportin kuvausta on käytetty strukturoidun datan rinnalla sekä ainoana datalähteenä tiedonlouhintatutkimuksissa muun muassa ohjelmistovirheiden luokittelussa ja korjausaikaa ennustettaessa (Zhou ym. 2014; Raja 2012; Assar, Borg ja Pfahl 2016)

Voorhees (2000) tiivistää tekstin prosessoinnin tiedonhakujärjestelmissä indeksointiin ja samankaltaisuuksien löytämiseen. Indeksoinnilla hänen mukaansa tarkoitetaan niiden termien valitsemista, jotka kuvaavat indeksoitavaa tekstidokumenttia. Automaattisen indeksoinnin peruskaavan hän kuvaa kolmivaiheisena prosessina. Ensin teksti pilkotaan tekstialkioiksi (eng. *token*). Toisena merkkijonoista poistetaan erityisen yleiset sanat, eli hukkasanat (eng. *stop word*). Kolmannessa vaiheessa sanat lyhennetään niiden perusmuotoon stemmauksella, jonka jälkeen dokumenttia kuvaavat termit ovat löytyneet.

Stemmausalgoritmeilla pyritään löytämään sanojen vartalot poistamalla niistä sanaliitteet, joilla ei ole vaikutusta sanan semanttiseen konseptiin. Stemmauksella pyritään siis normalisoimaan tekstialkiot siten, että saman sanan eri muodot, kuten *connecting* ja *connected*, tulevat indeksoiduksi samalla termillä. Normalisoitujen termien käyttö mahdollistaa dokumenttien samankaltaisuuden arvioinnin ja toimivamman tiedonhaun, kun dokumentista löytyvän sanan ei tarvitse olla samassa muodossa (Popovič ja Willett 1992). Stemmauksella saatava pienempi termistö myös pienentää indeksointiin käytetyn tietorakenteen muistivaativuutta ja tekee tiedonhaun laskennallisesti tehokkaammaksi (Moral ym. 2014). Stemmausalgoritmit ovat kielikohtaisia, eikä sanojen päätteiden poistamiseen perustuva algoritmi toimi kaikilla kielillä yhtä hyvin. Vaihtoehto stemmaukselle on sanojen perusmuotoistaminen (eng.*lemmatization*), missä pyritään löytämään sanojen perusmuodot ja erottamaan yhdyssanat käyttäen hyväksi kielen sanastoa ja morfologista analyysiä (Airio 2009).

Tunnetuimpia stemmausalgoritmeja ovar Lovins, Dawson, Porter ja Paise/Husk stemmerit,

jotka kaikki ovat englannin kielen stemmereitä. Moralin ym. mukaan (2014) Porterin (1980) algoritmi on käytetyin tiedonhaun menetelmissä. Porterin algoritmissa sanat tai sanan osat nähdään olevan muotoa $[C](VC)^m[V]$, missä C tarkoittaa konsonanttijonoa, V vokaalijonoa ja $m \geq 0$ on sanan mitta. Sanan typistys sen vartaloon tehdään viidessä peräkkäisessä vaiheessa. Jokaisessa vaiheessa on lista ehtoja, joiden täyttyessä sanan loppu voidaan poistaa tai vaihtaa toiseksi. Esimerkiksi toisessa vaiheessa on ehto `(m>0) FULNESS -> FUL`, jonka kohdalla sana *hopefulness* muuttuu muotoon *hopeful* ja kolmannessa kohdassa ehdon `(m>0) FUL ->` johdosta se lyhenee vielä sanan runkoon *hope*.

Stemmauksen tai perusmuotoistamisen tuloksena syntyneitä tekstialkioita voidaan käyttää dokumenttien indeksointiin. Indeksoinnissa luodaan tietorakenne, joka yhdistää termit niihin liittyviin dokumentteihin. Indeksointi nopeuttaa tiedonhakuprosessia, koska indeksoinnin jälkeen indeksistä voidaan hakea tiedontarpeen kannalta oleelliset dokumentit termien avulla. Yleisin tietorakenne indeksoinnille on käänteinen hakemisto (eng. *inverted file*), jossa jokaista dokumenttikokoelmassa esiintyvää termiä kohden on tallennettu lista niistä dokumenteista, joissa kyseinen termi esiintyy (De Moura ja Cristo 2017).

Vektoriavaruusmallin avulla dokumentit voidaan esittää hakemistossa n-dimensioisina vektoreina, joissa n vastaa hakemiston termien määrää ja dimension arvo voi olla joko binäärinen tai termin merkittävyyttä dokumentin kuvailun kannalta kuvaava luku (Salton, Wong ja Yang 1975). TF-IDF menetelmässä dimension arvo lasketaan termin dokumenttikohtaisen esiintymistiheyden ja koko aineistoa koskevan esiintymistiheydestä lasketun painoarvon suhteena (Paik 2013).

3.2 Päätöspuut ja Random Forest -menetelmä

Päätöspuut ovat ohjatun oppimisen avulla muodostettavia luokittelu- ja regressiotehtävissä käytettyjä hierarkkisia malleja. Päätöspuut koostuvat sisäsolmuista, joihin on liitetty jokin testifunktio sekä lehtisolmuista, joihin liittyy ennuste. Ennusteen arvo voi olla joko diskreetti tai jatkuva riippuen siitä onko kyseessä luokittelua vai regressiota varten luotu malli. Sisäsolmujen lapset ovat liitetty testifunktion tuloksiin. Testifunktio voi käyttää syötteenään yhtä tai useampaa syötteen dimensiota. Syötteelle ennustettu luokka tai arvo muodostetaan

aloittamalla puun juuresta ja seuraamalla solmuja testifunktioiden tulosten osoittamassa järjestyksessä, kunnes päädytään lehtisolmuun.

Koska optimaalisen päätöspuun muodostaminen on NP-täydellinen ongelma, siihen sovelletaan usein heuristisia ahneita algoritmeja (Laurent ja Rivest 1976; Murthy 1998). Mallin opetusvaiheessa sisäsolmujen testifunktiot määritellään siten, että ne pilkkovat opetusdatan mahdollisimman hyvin jollain jakokriteerillä. Testin hyvyyttä mittaavana jakokriteerinä voi toimia esimerkiksi testin tuoma informaatiolisä, eli se, kuinka paljon testin soveltaminen vähentää syntyneiden opetusdatan osajoukkojen entropiaa. Juuresta alkaen puuta muodostettaessa testifunktioiden määrittely toistetaan kullekin syntyneelle opetusdatan osajoukolle, kunnes pysäytyskriteerien täyttyessä luodaan lehtisolmu. Pysäytyskriteereinä puun kasvatusvaiheessa voi toimia esimerkiksi riittävän syvän puun muodostuminen, seuraavalle tasolle syntyvien opetusdatan osajoukkojen pieni koko, kaikkien jäljellä olevien opetusdatan alkioiden kuuluminen samaan luokkaan tai se, että paras mahdollinen opetusdatan jako ei täytä annettua jakokriteerin rajaa (Rokach ja Maimon 2008).

Kompleksiset päätöspuut ovat herkkiä ylisovittumiselle, jota voidaan pyrkiä pienentämään karsinnalla (eng. *pruning*). Karsinnassa päätöspuusta poistetaan sellaisia alipuita, jotka eivät paranna päätöspuun esittämän mallin yleistettävyyttä opetusdatan ulkopuolella. Karsinta myös parantaa päätöspuun tulkittavuutta yksinkertaistamalla puun rakennetta. Karsintavaiheessa käytetään yleensä opetusdatasta erillistä datajoukkoa, mutta tarvittaessa myös ristiinvalidointia voidaan hyödyntää.

Bootstrap aggregointi eli bagging on menetelmä, jossa samasta opetusdatasta poimituilla Bootstrap-otoksilla opetetaan useampi malli, joista kootun systeemin pitäisi olla parempi kuin yksittäisten mallien (Breiman 1996). Bootstrap-otos tehdään satunnaisotoksena takaisinpalauttaen, jolloin jokainen alkuperäisen opetusdatan alkio voi esiintyä otoksessa useasti tai ei ollenkaan. Bagging-menetelmä tasoittaa perusoppijana toimivan menetelmän taipumusta ylisovittumiseen ja epävakauteen eli herkkyyteen suurille muutoksille opetusdatassa tapahtuvan pienen muutoksen johdosta. Siispä perusoppijaksi sopii esimerkiksi ylisovittumiselle herkät neuroverkot tai päätöspuut. Bagging-menetelmällä luodun mallin ennuste uudelle syötteelle saadaan luokittelutehtävän tapauksessa enemmistöäänestyksellä valitsemalla se luokka, joka saa eniten ääniä yksittäisiltä luokittelijoilta. Regressiotehtävän tapauksessa

ennusteeksi voidaan laskea mallien ennusteiden keskiarvo.

Random forest, eli satunnainen metsä, on Breimanin (2001) esittämä koneoppimismenetelmä, jossa useita toisistaan poikkeavia päätöspuita yhdistämällä luodaan "metsä". Random forest -menetelmässä päätöspuiden opetuksessa yhdistetään Bagging-menetelmä Hon (1998) kehittämään satunnaisen aliavaruuden menetelmään. Satunnaisen aliavaruuden menetelmässä päätöspuun muodostamiseen tuodaan satunnaisuutta piilottamalla jokaisen päätöspuun solmun kohdalla satunnaisotannalla osa syötteen muuttujista, joita solmun testissä voisi käyttää. Uudelle syötteelle ennustettu luokka valitaan samoin kuin bagging-menetelmässä. Satunnaisen metsän puita ei ole välttämätöntä karsia, koska useiden puiden yhdistäminen kompensoi niiden ylisovittumista. Yksittäisiin päätöspuihin verrattuna satunnainen metsä on vaikeampi tulkita, sillä satojen puiden läpikäyminen manuaalisesti ja niiden yhteisvaikutuksen ymmärtäminen on melko työlästä.

3.3 Ohjelmistovirheiden korjausajan arviointi

Ohjelmistovirheiden korjausajan tietokoneavusteisella arvioinnilla ohjelmistoprojektien sidosryhmät voivat saada nopeasti tietoa projektin tilanteesta. Tieto virheen korjausajasta täyttää projektin sisäisiä tarpeita helpottamalla projektin kustannusten arviointia ja tehtävien delegointia. Projektista ulkoisia tarpeita tiedolle voi olla loppukäyttäjillä sekä projektiin riippuvaisen ohjelmiston kehittäjillä. Tässä luvussa käydään läpi tuoreita tieteellisiä julkaisuja, joissa on esitetty tai testattu koneoppimiseen pohjautuvaa menetelmää ohjelmistovirheiden korjausajan ennustamiseksi.

Malleja ohjelmistovirheiden korjausajan arviointiin on kehitetty useissa tutkimuksissa 2010-luvulla. Dataa on kerätty niin avoimen kuin suljetun lähdekoodin projekteista. Näistä valtaosa on julkaistu konferenssipapereina. Eräistä tutkimuksista on saatavilla myös pidempi artikkelimuotoinen teksti. Kirjallisuutta on löytynyt eniten ACM DL ja IEEE -tietokannoista ja Google Scholar -hakukoneella sekä seuraamalla viitteitä jo löytyneistä julkaisuista. Tutkimukset eroavat muun muassa mallin kehityksessä käytettyjen projektien määrässä, hyödynnetyissä muuttujissa ja koneoppimisalgoritmeissa. Suurin osa tutkimuksista hyödynsi suosittuja avoimen lähdekodin projekteja.

Anh ym. (2011) tutkivat kehittäjäkohtaisten muuttujien merkitystä ohjelmistovirheiden korjausaikojen suhteen. Vahvimpana havaintona he huomasivat kehittäjän aiemmin korjaamien virheiden korjausaikojen keskiarvon olevan potentiaalinen muuttuja, jonka mukaan ottaminen paransi koneoppimisalgoritmin tuottaman mallin tarkkuutta 2.87-12.07%. Tutkimuksen aineistona toimi avoimen lähdekoodin ohjelmistoprojektit Qt, Qupid ja Geronimo.

Giger, Pinzger ja Gall (2010) tutkivat ohjelmistovirheiden luokittelua hitaasti ja nopeasti ratkaistaviin. Aineistona he käyttivät kuutta Eclipse-, Mozilla- ja Gnome-säätiöiden avoimen lähdekoodin projektia. Heidän päätöspuihin perustuva mallinsa luokitteli virheet 10-20% sattumanvaraista arvausta paremmin. Tutkimuksessa havaittiin ennusteen kannalta keskeisiksi muuttujiksi esimerkiksi delegoitu kehittäjä ja avaamisen ajankohta. Virheraportin avaamisen jälkeen arvoaan muuttavien dynaamisten muuttujien mukaan ottaminen paransi mallin tarkkuutta. Mallin heikkoutena voisi pitää sitä, että opetus- ja testidataa ei oltu eroteltu ajallisesti, eli mallissa oli tietoa "tulevaisuudesta" suhteessa testidataan. Mallin käyttökelpoisuus uusien bugien korjausajan ennustamiseen ei siis ole ollenkaan varmaa.

Lamkanfi ja Demeyer (2012) huomasivat epäilyttävän piirteen virheenseurantaohjelmista saadusta datasta: suuri osa virheistä korjataan heti avaamisen jälkeen. He selittivät tämän sillä, että joskus ohjelmistokehittäjät korjaavat löytämänsä virheen ennen kuin sitä on ehditty raportoida. Kehittäjät sitten avaavat ja sulkevat raportin samaan aikaan kuin julkaisevat korjatun ohjelmakoodin. He epäilivät, että nämä hyvin nopeasti suljetut raportit voivat hämmentää koneoppimisalgoritmeja ja tutkivat hyvin nopeasti suljettujen virherapottien suodattamista aineistosta. He käyttivät Naiivia Bayes-luokittelijaa ja vertailivat saadun mallin suorituskyvyn muutosta, kun epäilyttävät raportit on poistettu aineistosta. Tuloksena luokittelijan AUC-arvo parani jonkin verran suodatetulla datalla. Johtopäätöksenä voinee todeta, että kuten muussakin datanlouhinnassa, myös Mining Software Repositories -kentällä poikkeavien havaintojen analysointi ja suodatus voi tuottaa parempia tuloksia.

Guo ym. (2010) havaitsivat, että virheen raportoijan ja sitä ratkaisevan kehittäjän historia vaikuttavat raportoitujen virheiden korjausaikaan. He esittivät raportoijan "maineen" laskemista avattujen ja korjattujen virheraporttien suhteena ja tutkivat tämän muuttujan vaikutusta luokittimen tarkkuuteen. He totesivat tämän muuttujan parantavan luokittimen suorituskykyä. Bhattacharya ja Neamtiu (2011) huomasivat ristiriidan Guon ym. (2010) tutkimuksen suh-

teen, sillä he eivät avoimen lähdekoodin projekteissa havainneet yhteyttä raportoijan "maineen" ja ohjelmistovirheen korjausaikojen välillä. He tarkastelivat myös muita muuttujia, joita on käytetty virhekorjauksen valmistumisaikaa ennustavissa malleissa ja huomasivat, että pieni määrä muuttujia ei riitä selittämään ilmiötä.

Kehittäjäkohtaista pisteytystä on esittänyt myös Ramarao ym. (2016). Heidän esittämänsä kaavan mukaan kehittäjän "maine" lasketaan hänen raportoimiensa jo korjattujen virheraporttien perusteella. Sen lisäksi pisteytyksen painoarvo suhteutetaan projektikohtaisesti virheen korjausajan ennustamista varten. Tutkijat osoittivat pisteytyksen toimivuuden knn-pohjaisella luokittimella.

Marks, Zou ja Hassan (2011) käyttivät Random Forest -algoritmia tutkiakseen aikaa virheraportin delegoinnista sen korjaamiseen. Tutkimuksen aineistona oli Eclipse- ja Mozilla-projektien Bugzilla-tietokannat. Saadut mallit ennustivat korjausajan noin 65% tarkkuudella kolmeen luokkaan: kvartaalin, vuoden ja kolmen vuoden sisällä korjattaviin. Tutkijat myös huomasivat, että parhaiten korjausaikaa ennustavat muuttujat erosivat projektien välillä. Tämä antaa aihetta epäillä sitä, että pelkkien staattisten ja strukturoitujen muuttujien perusteella tarkan mallin luominen ei onnistu.

Zhang, Gong ja Versteeg (2013) tutkivat yhden ohjelmistoalan organisaation sisäisiä projekteja, ja loivat kolme mallia estimoimaan eri näkökulmia virheiden korjausaikaan. Markovin ketjuihin perustuvalla mallilla he onnistuivat luotettavasti ennustamaan tulevaisuudessa korjattavien ohjelmistovirheiden lukumäärän. Monte Carlo -simulointiin perustuvalla metodilla he estimoivat aikaa, joka kuluu annetun virhemäärän korjaamiseen. Luokittelemalla bugit kNN-algoritmilla nopeasti ja hitaasti korjautuviin he myös onnistuivat luokittelemaan yksittäisten virheiden korjausaikoja (F-arvo 72.45%). Tärkeänä osana hyvän tuloksen saamista oli se, että tutkijat sovittivat mallinsa ottamaan huomioon korjausaikojen jakauman, jossa suurin osa virheistä korjataan varsin lyhyessä ajassa. Markovin ketjuja käyttivät myös Pombo ja Teixeira (2020), joiden kehittämä malli luokitteli virheet nopeasti ja hitaasti korjattaviksi 65.01-77.87% tarkkuuteen yltäen. Ylisovittumisesta johtuen malli sai parhaat tulokset, kun opetus- ja testidataa oli saman verran.

Pfahl, Karus ja Stavnycha (2016) toistivat usean aiemman ohjelmistovirheisiin tai tehtävä-

tiketteihin kuluvan ajan ennustamiseen esitetyn koneoppimismenetelmän, ja vertasivat saatua tarkkuutta samojen tikettien luomisessa annettuun asiantuntija-arvioon. Pääsääntöisesti asiantuntija-arviot olivat tarkempia, mutta Random Forest -algoritmiin ja logistiseen regressioon perustuvat mallit pääsivät varsin lähelle. Tekstianalyysiin perustuva *Spherical k-means Clustering* tuotti jopa tarkempia arvioita kuin asiantuntijat.

Porru ym. (2016) kehittivät käytännön ohjelmistokehitykseen sopivan mallin, joka ennustaa tehtävälle annettavien *story point* -pisteiden määrän. Niillä kuvataan tehtävän vaativuutta, ei niinkään arvioitua työaikaa. Paras luokitinmalli saatiin aikaan tukivektorikoneella. Keskeisenä elementtinä piirteiden louhinnassa oli tekstianalyysi, mihin tukivektorikoneet sopivat hyvin. Mallin opetukseen riitti noin 300 tehtävätikettiä. Tehtävät jakautuivat *story point* -pisteiden mukaan 13:n luokkaan ja mallin keskimääräinen tarkkuus oli 64%. Tutkimuksen aineistona oli kahdeksan avoimen lähdekoodin projektia ja yksi teollisuusprojekti.

Myöhemmin Scott ja Pfahl (2018) tulivat kuitenkin siihen tulokseen, että *story point* -pisteitä ennustettaessa kehittäjäkohtaisten ominaisuuksien perusteella muodostettu tukivektorikonepohjainen malli tuottaa parempia tuloksia kuin tekstianalyysiin pohjaava. Scott ja Pfahl (2018) käyttivät samaa aineistoa, kuin Porru ym. (2016), mutta heidän tekstianalyysiin pohjaavan mallin tarkkuus oli vain 35.8%, mikä on ristiriidassa aiemman tutkimuksen tulosten kanssa. Koska myös kehittäjäkohtaisten ominaisuuksien vaikutuksesta tehtävien valmistumisaikaan on ristiriitaisia tuloksia, vahvistavat nämä tutkimukset sitä näkemystä, että myöskään ohjelmistovirheen korjausajan ennustamista ei voida yksinkertaistaa vain muutamaan muuttujaan.

Ardimento ja Dinapoli (2017) käyttivät myös tukivektorikoneita mallissa, joka ennustaa ohjelmistovirheitä nopeasti tai hitaasti korjattaviksi sen tiedon perusteella, jota on käytettävissä tiketin avaushetkellä. He hyödynsivät myös virheraportin kommenteista löytyvää strukturoimatonta tietoa prosessoimalla tekstuaalisesta datasta TF-IDF-matriisit. Aineisona käytettiin kolmea avoimen lähdekoodin projektia: Novell, OpenOffice ja LiveCode. Malli luokitteli raportit nopeiksi 35-52% tarkkuudella, ja sen herkkyys oli 60-76%.

Ardimento, Bilancia ja Monopoli (2016) lähestyivät samaa ennustustehtävää hyödyntämällä SLDA-tekstinlouhinta-algoritmia. SLDA-algoritmin avulla he pyrkivät erottelemaan jokai-

sen virheraportin kuvauksesta useampia ala-aiheita, jotka voisivat selittää paremmin virheiden korjausaikaa. He painottivat malliaan tunnistamaan erityisesti hitaasti korjattavat ohjelmistovirheet. Aineistona heillä oli avoimen lähdekoodin Eclipse-, Gentoo-, KDE- ja OpenOffice-projektit. Painotuksen vuoksi mallin tarkkuus jäi alle tukivektorikoneen, mutta se tunnisti useimmat hitaasti korjattavat virheet.

Al-Zubaidi ym. (2017) käyttivät evoluutioalgoritmia ohjelmistokehitystehtävän ratkaisemiseen kuluvaa aikaa ennustavan mallin muodostamiseksi. Tavoitteena oli tehdä ennusteita sillä tiedolla, jota on käytettävissä tehtävätiketin avaushetkellä. Tutkimuksen lähdeaineistona oli viisi Apache-järjestön avoimen lähdekoodin projektia. Kyseisellä aineistolla evoluutioalgoritmi tuotti paremman mallin kuin lineaarinen regressio, kNN-algoritmi tai Random Forest -algoritmi. Heidän mallinsa etuna useisiin muihin voinee pitää sitä, että luokitteluasteikollisten aikaikkunoiden sijaan ennusteet ovat jatkuva-asteikollisia.

Sharma, Kumari ja Singh (2019) tutkivat virheiden korjausaikojen sekä vakavuuden ennustamista ja vertailivat useita koneoppimisalgoritmeja. Korjausaikoja onnistuttiin ennustamaan parhaiten tukivektoriregressiomallilla, jolla R^2-arvoksi saatiin yli 0.9. Muita käytettyjä algoritmeja olivat lineaarinen regressio, sumea regressio ja kNN-regressio. Aineistona he käyttivät kahdeksaa Mozilla-säätiön ohjelmistoprojektia. He käyttivät yhtenä muuttujana mallien opettamisessa virheraporttien kuvauksista laskettujen TF-IDF-vektoreiden termien painoarvojen summaa, mutta tämä muuttuja ei parantanut mallien tarkkuutta. Siispä tekstuaalisen datan yksinkertaistaminen yhdeksi numeeriseksi muuttujaksi ei välttämättä toimi muillakaan aineistoilla ja menetelmillä.

Lee ym. (2020) kehittivät syväoppimiseen perustuvan mallin, joka hyödynsi virheen korjausaikana syntyvää dynaamista dataa sen korjausajan ennustamiseksi. Seitsemään aikaikkunaluokkaan opetettuna mallin tarkkuus oli 23.5-24.9%. Tutkijat keräsivät dataa Chromium-, Firefox-, ja Eclipse-projektien virheraportointijärjestelmistä. Heidän mallinsa on ainoa tähän mennessä julkaistu syväoppimiseen perustuva ohjelmistovirheiden korjausaikoja ennustava järjestelmä.

4 Tutkimus

Tässä luvussa esitellään tutkimusongelma, kuvaillaan tutkimukseen valittu aineisto sekä selitetään tutkimusprosessi ja valitut menetelmät.

4.1 Tutkimusongelma

Tutkimuksen tavoitteena on tuottaa tietoa siitä, kuinka ohjelmistoprojektin virheraportin ratkeamisaika voidaan ennustaa käyttäen hyväksi tiedonlouhimisen ja koneoppimisen menetelmiä. Ratkeamisajalla tarkoitetaan aikaa, joka kuluu raportin luomishetkestä siihen, että se suljetaan. Tutkimuksen kokeellinen osuus on luonteeltaan tiedonlouhimis- ja koneoppimistehtävä. Tutkimus koostuu kahdesta osasta: aiemman tutkimuksen replikaatiosta ja luonteeltaan eksploratiivisesta osasta, jossa hyödynnetään varioiden replikaatiossa käytettyä menetelmää. Replikaatiotutkimuksessa toistetaan Kikasin, Dumasin ja Pfahlin (2016) tutkimus, jossa kehitettiin koneoppimismenetelmin malli ohjelmistokehitystehtävän valmistumisajan ennustamiseksi. Aiemman tutkimuksen replikoinnilla pyritään alkuperäisten löydösten vahvistamisen lisäksi arvioimaan siinä käytetyn tutkimusmetodin uudelleenkäytettävyyttä toisessa kontekstissa. Kikasin, Dumasin ja Pfahlin tutkimuksessa käytetty lähdekoodi oli julkaistu, mutta siihen ei oltu liitetty mitään tekijänoikeuslisenssiä, joten lupa koodin hyödyntämiseen saatiin kontaktoimalla alkuperäiset tutkijat.

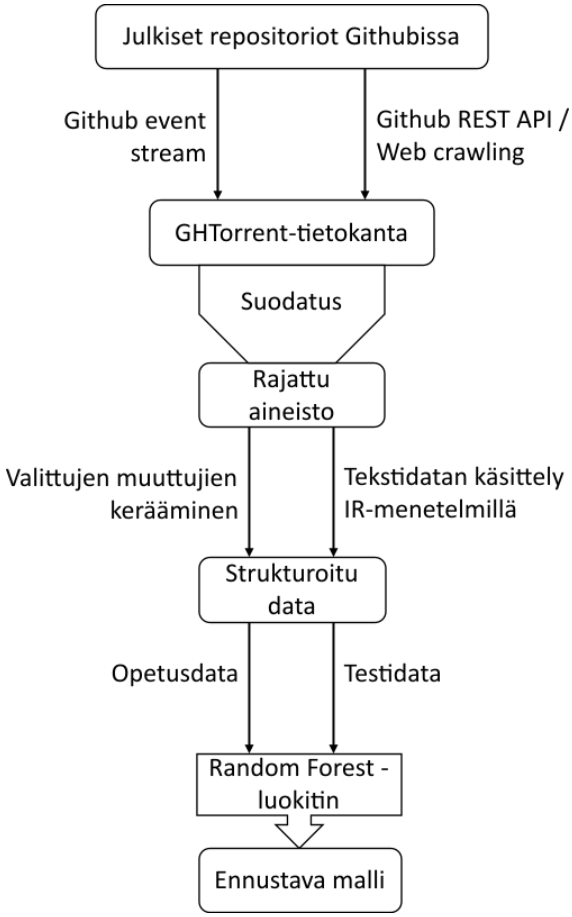
Replikaatiotutkimuksessa arvioidaan ja validoidaan aiemman tutkimuksen tuloksia toistamalla raportoitu tutkimusprosessi ja vertaamalla saatuja uusia tuloksia alkuperäisiin. Replikaatiotutkimus tuo lisäarvoa aiemmalle tutkimukselle, jos replikaation tulokset vahvistavat alkuperäisiä tuloksia. Toisaalta replikaatiotutkimuksella on mahdollista osoittaa alkuperäisessä tutkimuksessa huomaamatta jäänyt heikkous. Shullin ym. (2008) mukaan ohjelmistoalan tutkimuksessa replikoinnilla voidaankin tarttua sekä ulkoiseen että sisäiseen validiteettiin tuomalla tutkimus uuteen ympäristöön (ulkoinen validiteetti), ja tutkimalla minkä rajoitteiden sisällä tulokset pätevät (sisäinen validiteetti). Tässä replikaatiotutkimuksessa tutkimusprosessi toistetaan uudella raakadatalla, jolloin tutkimuksen tulokset tulevat testatuksi uudessa ympäristössä. Gómez, Juristo ja Vegas (2014) jakavat replikaatiotutkimukset ohjelmisto-

tekniikan alalla kirjaimellisiin, konseptuaalisiin ja operationaalisiin replikaatioihin. Kirjaimelliset replikaatiot pyrkivät toistamaan replikoitavan tutkimuksen mahdollisimman tarkasti. Konseptuaalisissa replikaatioissa tutkijat ottavat alkuperäisestä tutkimuksesta vain tulokset, joita pyritään verifioimaan uudella tutkimusprosessilla. Operationaalisissa replikaatioissa, mitä tämä tutkimus edustaa, alkuperäinen tutkimus toistetaan siten, että jokin tai jotkut tutkimusasetelman muuttujista on muutettu. Robles (2010) on tutkiessaan ohjelmistoprojektien tiedonlouhintaa koskevien tutkimusten replikoitavuutta löytänyt haasteiksi raakadatan saatavuuden, prosessoidun datan saatavuuden ja käytettyjen työkalujen ja skriptien saatavuuden. Kikasin, Dumasin ja Pfahlin (2016) tutkimuksen replikoitavuus on erityisen hyvällä tasolla, sillä tutkimuksessa käytetty raakadata ja prosessoitu data ovat saatavilla. Myös suurin osa skripteistä on saatavilla, ja ainoastaan raakadatan prosessointiin käytetyt lähdekoodit on jätetty julkaisematta. Suurin epävarmuus replikaation laadussa koskeekin siis raakadatan käsittelyä, kun työkalut sitä varten on johdettava tutkimusjulkaisussa ilmoitettujen periaatteiden perusteella.

Tutkimuksen replikaatio-osaan ja eksploratiiviseen osaan kuuluu datan suodatus, sen transformoiminen sopivaan muotoon, jakaminen opetus- ja testidataan, sekä syöttäminen koneoppimisalgoritmiin. Kuviossa 2 on esitetty menetelmän vaiheet järjestyksessä ylhäältä alas. Replikaation jälkeen toteutetussa eksploratiivisessa osassa oli päämääränä selvittää, kuinka hyvin sama koneoppimismenetelmä toimii, kun sillä pyritään ennustamaan ohjelmistovirheen korjausaikaa. Tätä testattiin muuttamalla raakadatan prosessointia siten, että kaikista ohjelmistoprojektin kehitystehtävistä eroteltiin varmasti ohjelmistovirheisiin kohdistuvat tehtävät. Kohdistamalla huomio ohjelmistovirheisiin pyrittiin ennusteesta saamaan tarkempi kuin mihin yleisempi kaikki tehtävät huomioon ottava malli kykenisi. Samalla tutkimuksen eksploratiivisen osan tarkoituksena oli tuottaa uutta tietoa ohjelmistovirheistä, ja niihin vaikuttavista tekijöistä ilmiönä. Laajempi ymmärrys virheistä voi olla arvokasta niin ohjelmistokehittäjille, tutkijoille kuin ohjelmistojen loppukäyttäjillekin.

4.2 Aineisto

Tutkimuksen raakadatana on GHTorrent-projektin (Gousios 2013) keräämät relaatio- ja dokumenttitietokannat GitHub-palveluun tallennetuista avoimesti saatavilla olevista ohjelmis-



Kuvio 2: Tutkimusmenetelmän vaiheet: Datan kerääminen, suodatus, käsittely ja syöttäminen Random Forest -algoritmiin.

toprojekteista. Tietokantoihin on tallennettu projekteihin liittyen tuote- ja prosessidataa; projekteista on kerätty muun muassa tiedot versiohistoriaa kuvaavista commit-tapahtumista, kehittäjien luomista pull request -muutosehdotuksista ja kehittäjien sekä loppukäyttäjien luomista *issue*-tehtäväketjuista. Issues on GitHubin keskustelupohjainen työkalu, mikä soveltuu kehitystehtävien ja ohjelmistovirheiden hallintaan. Tehtäviin liittyvä tekstidata, kuten otsikot, kuvaukset ja kommentit on tallennettu vain dokumenttitietokantaan, mutta muut tutkimuksessa käytetyt tietokentät löytyivät myös relaatiotietokannasta, mistä niiden kerääminen oli helpompaa ja nopeampaa.

GHTorrent-tietokantaan kerätty data on peräisin GitHubin tarjoamasta REST-ohjelmointirajapinnasta[1] (Gousios 2013). GHTorrent-aineiston suurimpana etuna voi pitää sen suurta ko-

1. https://docs.github.com/en/rest

koa ja heterogeenisyyttä. GitHub on vakiinnuttanut asemansa suosituimpana alustana avoimen lähdekoodin projektien hallintaan, joten sieltä kerätyn tiedon voisi olettaa edustavan hyvin kattavaa kuvaa avoimen lähdekoodin projekteihin liittyvistä ilmiöistä, kuten kehittäjien yhteistyöstä, ohjelmistojen elinkaaresta ja niiden laajuudesta. Datasta on poistettu käyttäjänimet ja sähköpostiosoitteet tietosuoja-asetusten johdosta.

Aineistosta löytyvät projektit ovat perustettu Github-palveluun vuosien 2008 ja 2021 välillä. Käytetyn aineiston viimeinen päivitys on tapahtunut 6.3.2021. Tietokannasta löytyy 189 467 747 projektia, joista kaikki eivät kuitenkaan ole ohjelmistoprojekteja, vaan avoimesti saatavilla on myös esimerkiksi tekstitiedostoja, koodinäytteitä ja opinnäytetöitä. Projektit, jotka eivät ole ohjelmistoprojekteja eivätkä kuulu tutkimuksen kohderyhmään, hävitettiin aineiston suodatusvaiheessa. Myös Kikasin, Dumasin ja Pfahlin (2016) tutkimuksessa käytettiin GHTorrent-projektin dataa. Heidän käyttämänsä tietokanta oli ladattu saataville 1.4.2015. Tässä tutkimuksessa käytetyn tietokannan koko pakattuna oli lähes kahdeksan kertaa suurempi kuin Kikasin, Dumasin ja Pfahlin vastaava.

Relaatio- ja dokumenttitietokannat ladattiin GHTorrent-projektin palvelimelta ja asennettiin lokaalissa ympäristössä toimiviin tietokoneisiin. Relaatiotietokannan hallintaohjelmistona toimi PostgreSQL ja dokumenttitietokantaa käytettiin MongoDB-ohjelmistolla.

Useimmat virheiden korjausaikaa ennustavat mallit ovat opetettu ja testattu alle kymmenen suuren ohjelmiston virheenseurantajärjestelmästä saadulla datalla. GitHubissa taas eri kokoisia repositorioita on miljoonia. Yksi haaste GitHubista kerätyssä datassa verrattuna varsinaisista virheenseurantaohjelmista kerättyyn dataan on se, että GitHubin Issues-työkalu jättää varsin vapaat kädet ohjelmistovirheiden käsittelyprosessille. Issues ei myöskään rajoitu vain virheiden seurantaan, vaan sitä käytetään muidenkin ohjelmistoprojektin tehtävien kuten uusien ominaisuuksien kehittämisen ja dokumentoinnin hallintaan. Erityyppisiä tehtäviä voi erotella niihin liitettävillä nimikkeillä (eng. *label*), ja yksi GitHubin tarjoamista vakionimikkeistä onkin "bug". Vakionimikkeitä, jotka viittaavat siihen ettei tehtävä ole virheraportti, ovat esimerkiksi "documentation", "duplicate" ja "enhancement". Projekteilla on kuitenkin mahdollisuus räätälöidä käytettäviä nimikkeitä omien tarpeidensa mukaan esimerkiksi hienosyisempään virheiden jaotteluun tai erilaisten tehtävien kuvaamiseksi. Nimikkeiden käyttö ei myöskään ole pakollista, jolloin voi olla mahdotonta selvittää strukturoitujen tietokenttien

perusteella, onko jokin tehtävä avattu virheen korjaamista vai jotain muuta kehitystyötä varten. Projekteilla on vaihtelevia virheenkorjausprosesseja, ja niiden kulkua ei välttämättä pysty seuraamaan pelkästään GitHubista löytyvän datan perusteella. Siksi lienee hyvin vaikeaa antaa yleistä kaaviota siitä, mitä tapahtuu virheraportin avaamisen ja sen sulkemisen välillä GitHubia käyttävissä projekteissa. Esimerkiksi Bugzilla-järjestelmässä virheraporteilla ja niiden käsittelyllä on yksiselitteisempi vuokaavio (Kuvio 1).

4.3 Replikaatio

Replikaatiotutkimuksessa pyritään toistamaan alkuperäinen tutkimus ja tuloksia vertaamalla arvioimaan alkuperäisten tulosten vahvuutta ja luotettavuutta, sekä mahdollisesti tuomaan uusi näkökulman tutkittavaan ilmiöön. Tutkimuskysymyksiksi asetettiin alkuperäisen tutkimuksen tutkimuskysymykset:

> Tutkimuskysymys 1: *"Kuinka suuri tarkkuus saavutetaan luokittimella, joka on opetettu ennustamaan kehitystehtävän sulkeutumisaikaa eri aikaikkunoilla (vuorokausi, viikko, [kaksi viikkoa,] kuukausi, kvartaali, puoli vuotta ja vuosi) käyttäen staattisia, dynaamisia ja kontekstuaalisia muuttujia?"*
>
> Tutkimuskysymys 2: *"Mitkä muuttujat ovat tärkeimpiä sulkeutumisaikaa ennustettaessa?"*

Jos vastaukset tutkimuskysymyksiin ovat samat, vahvistavat ne alkuperäisen tutkimuksen tuloksia. Toisaalta eriävät tulokset voivat vähentää tutkimuksen uskottavuutta, jos replikaation havainnot ovat alkuperäistä heikompia. Alkuperäistä paremmat tulokset voivat tuottaa kiinnostusta lisätutkimusta kohtaan.

Koska kaikki GitHubista avoimesti saatavilla olevat projektit eivät ole ohjelmistoprojekteja, on aineistosta suodatettava pois epärelevantit projektit, kuten replikoitavassa tutkimuksessa. Epärelevantteja projekteja ovat esimerkiksi toisesta GitHubin projektista forkatut tai toisesta palvelusta GitHubiin siirretyt projektit, vain dokumentaatiota varten luodut projektit ja projektit, jotka eivät ole ohjelmistoprojekteja. Dataa suodattamalla sen määrä ja tutkimuskohteeseen liittymätön kohina vähenee. Vähemmän kohinaisesta datasta koneoppimisalgoritmi

tuottaa paremman mallin, ja vähemmällä datan määrällä sen opetus käy nopeammin.

Tutkimuksen replikaatio-osaa varten aineisto suodatettiin samalla tavalla kuin Kikasin, Dumasin ja Pfahlin (2016) tutkimuksessa ottaen huomioon se, että tässä tutkimuksessa dataa oli käytettävissä pidemmältä aikaväliltä. Aineistosta replikaatiotutkimukseen mukaan valittiin projektit, jotka täyttivät seuraavat ehdot:

1. Projekti on luotu aikaisintaan 1.1.2012 ja viimeistään 31.12.2020.
2. Projekti ei ole forkattu toisesta projektista.
3. Projektin versiohistoriassa on ainakin 5 commit-tapahtumaa.
4. Projektin versiohistoriassa ei ole tapahtumia ennen projektin luomista GitHubissa.
5. Projektissa ei ole avattu yli 2000:ta tehtävää kuukauden aikana eikä yli 500:aa yhden päivän aikana.
6. Projektilla on vähintään 100 avattua tehtävää ja ainakin yksi suljettu tehtävä.

Ehdot 2-6 olivat samat kuin alkuperäisessä tutkimuksessa. Datasta myös tiputettiin sellaiset tehtävät, jotka olivat sulkemisen jälkeen avattu uudelleen. Tehtävällä tarkoitetaan GitHubin Issues-työkaluun luotua ketjua. Tehtävät, jotka tulivat uudelleenavatuksi ensimmäisen sulkemisen jälkeen, jätettiin pois tutkimuksesta. GitHub tekee automaattisesti kirjauksen projektin Issues-työkalun tietoihin jokaista pull request tapahtumaa kohden, mutta nämä jätettiin huomiotta. Alkuperäisessä tutkimuksessa näiden automaattisesti luotujen kirjausten käsittely jäi ilmoittamatta.

Datan suodatuksen jälkeen tutkimuksen kannalta oleellisia projekteja löytyi 39 125 kappaletta. Projekteissa oli yhteensä 12 750 415 avattua tehtävää. Kikasin, Dumasin ja Pfahlin tutki-

Taulukko 1: Aineiston koko ja tietoa replikaatiossa ja alkuperäistutkimuksessa

	Replikaatio	Kikas, Dumas ja Pfahl (2016)
Projekteja suodatuksen jälkeen	39 125	4024
Tehtäviä	12 750 415	967 037
Tehtävistä suljettu	67.6%	69.9%

muksessa projekteja valikoitui 4024 kappaletta ja 967 037 tehtävää (Taulukko 1). Replikaatioon valituista projekteista 2172 kappaletta oli valittu myös replikoitavassa tutkimuksessa, eli 46.0% alkuperäisen tutkimuksen ohjelmistoprojekteista ei ollut osana replikaatiota. Nämä projektit eivät ole enää läpäisseet hyväksymisehtoja, tai ne ovat hävinneet raakadatasta. Replikaatiotutkimuksen tehtävistä 519 932 eli 4.1% kuului myös alkuperäisen tutkimuksen tehtäviin.

4.4 Tekstiaineiston käsittely

Strukturoimatonta tekstimuotoista dataa kerättiin valittujen tehtävien otsikoista, kuvauksista ja kommenteista. Alkuperäistä tutkimusta seuraten tekstidata prosessoitiin poistamalla siitä ensin Markdown-merkintäkielen symbolit, linkit, taulukot ja muut erikoismerkit. Tekstistä poistettiin englanninkieliset hukkasanat, jotka tunnistettiin Snowball-projektin listan[2] perusteella. Snowball-projektin lista valittiin sen tiiviyden ja tunnettuuden vuoksi. Snowball-projektin hukkalistaa tai sen johdosta käytetään muun muassa PostgreSQL ja MongoDB[3] projekteissa (Nothman, Qin ja Yurchak 2018). Hukkasanojen poistamisen jälkeen prosessoitu tekstidata riisuttiin tekstialkioiksi Porter-stemmerillä. Alkuperäisessä tutkimuksessa tekstin prosessointiin käytetty koodi ei ollut saatavilla, joten koodiskriptit näitä tekstin prosessointeja varten tehtiin itse.

Alkuperäisestä tutkimuksesta saatavilla olevassa koodissa jokaiselle tehtävälle muodostettiin numeerinen *text score* -arvo, jota käytettiin yhtenä muuttujana tehtävän sulkemisaikaa ennustettaessa. *Text score* -arvon laskeminen alkoi muodostamalla ohjelmistokehitystehtäviin liittyvästä tekstidatasta dokumentti-termi matriisit käyttäen matriisien indekseinä tekstin termeistä hajautusfunktion avulla muodostettuja kokonaislukuarvoja. Matriisit muodostettiin sekä opetus- että testidatasta. Testidata jaettiin satunnaisesti kahteen osaan ja datan avulla muodostetun logistisen regressiomallin avulla laskettiin tehtävän *text score* -arvo. Malli sovitettiin käyttäen ensin toista osaa opetusdatasta sovittamiseen saaden arvot toiselle puolelle, ja sitten toistaen prosessi vaihtaen sovitettava ja ennustettava data keskenään. Testidatan arvot saatiin sovittamalla malli koko opetusdatalla.

2. https://snowballstem.org/algorithms/english/stop.txt
3. https://github.com/mongodb/mongo/blob/master/src/mongo/db/fts/stop_words_english.txt

4.5 Muuttujat

Kikas, Dumas ja Pfahl (2016) valitsivat tutkimuksessaan 21 muuttujaa (Taulukko 2), jotka toimivat syötteenä koneoppimisalgoritmille. Samoja muuttujia käytettiin myös replikaatiotutkimuksessa. Muuttujien on tarkoitus kuvata tehtävän tilaa mahdollisimman tarkasti siten, että tehtävän sulkeutumisajasta on osoitettavissa riippuvuutta valittujen muuttujien suhteen, eivätkä muuttujat korreloi kuitenkaan erityisen paljon keskenään. Muuttujat sisälsivät tietoa tehtävätiketin lisäksi projektien, tehtävän avaajan ja siihen osallistuneiden käyttäjien aktiivisuudesta ja historiasta. Muuttujista 13 oli dynaamisia muuttujia, eli niiden arvot voivat muuttua eri ajanhetkinä. Taulukon 2 t-päätteiset muuttujat ovat dynaamisia. Dynaamisten muuttujien ajasta riippuvan luonteen huomioimiseksi tehtävien tilasta otettiin näytteet luomishetkellä sekä 1, 7, 14, 30, 90 ja 180 päivän ikäisinä. Tutkimuskysymyksessä 1 mainitut kontekstuaaliset muuttujat ovat taulukon 2 kuusi viimeistä muuttujaa, jotka käsittelevät projektin aktiivisuutta. Näytteistä saatiin seitsemän datajoukkoa, ja tekoäly opetettiin ennustamaan tehtävän sulkeutumisaika eri ikäisten tehtävien kontekstissa. Alkuperäisessä tutkimuksessa *nMentionedByT*-muuttujan kuvaus oli virheellinen, ja sen oikea selite selvitettiin ottamalla yhteyttä tutkijoihin.

4.6 Mallin opetus ja testaus

Alkuperäistä tutkimusta seuraten aineisto jaettiin opetus- ja testausdataan siten, että kaikki ennen valittua ajankohtaa avatut tehtävät kuuluvat opetusjoukkoon, ja ajankohdan jälkeen avatut tehtävät kuuluvat testausjoukkoon. Ajankohdaksi valittiin 01.12.2018, jolloin opetusdata käsitti 7 266 657 tehtävää, mikä oli 57.0% koko aineistosta. Loput 43.0%, eli 5 483 758 tehtävää tuli valituksi testausdataksi. Alkuperäisessä tutkimuksessa opetus- ja testausjoukon suhde koko aineistoon oli 43.9% ja 56.1%. Jakamalla aineisto satunnaisotannan sijaan ajallisesti pyrittiin estämään “tulevaisuuden” tiedon vuotamisen malliin, mikä vaikeuttaisi mallin käytännön sovellettavuuden arviointia.

Koneoppimismenetelmänä käytettiin Random Forest -luokitinta. Aineistosta muodostettiin 28 binääriluokitinta, joiden syötteinä oli näyte tehtävästä t_1 päivää sen avaamisesta, ja luokittimen tuli ennustaa, onko tehtävä suljettu t_2 päivää avaamisen jälkeen. Luokitin opetettiin jo-

Taulukko 2: Muuttujat alkuperäistutkimuksessa (Kikas, Dumas ja Pfahl 2016).

Muuttuja	Selite
nCommentsT	Kommenttien määrä näytteenottohetkellä.
nActorsT	Tehtävään kommentoineiden, viitanneiden ja sitä seuraavien käyttäjien määrä näytteenottohetkellä.
nAssignmentsT	Tehtävälle nimitettyjen käyttäjien määrä ennen näytteenottohetkeä.
nLabelsT	Tehtävälle annettujen nimikkeiden määrä.
nMentionedByT	Tehtävässä käyttäjiin tehtyjen viittausten määrä.
nReferencedByT	Tehtävään kohdistuvien viittausten määrä commit-tapahtumista.
nSubscribedByT	Tehtävää seuraavien käyttäjien lukumäärä.
meanCommentSizeT	Tehtävän keskimääräinen kommenttien pituus.
issueCleanedBodyLen	Otsikon ja tehtävän kuvauksen yhdistetty pituus alkioiksi riisumisen jälkeen.
textScore	Tehtävään otsikosta ja sisältötekstistä saatu luokitteluarvo.
nIssuesByCreator	Tehtävän luojan avaamien tehtävien määrä edeltävän kolmen kuukauden aikana.
nIssuesByCreatorClosed	Tehtävän luojan avaamien ja edeltävän kolmen kuukauden aikana suljettujen tehtävien määrä.
nCommitsByCreator	Tehtävän luojan commit-tapahtumien määrä projektissa kolmen kuukauden aikana ennen tehtävän avaamista.
nCommitsByActorsT	Tehtävään osallistuneiden tekemien commit-tapahtumien lukumäärä tehtävän avausta edeltävän kahden viikon ja näytteenottohetken välillä.
nCommitsByUniqueActorsT	Projektiin commit-tapahtumia luoneiden käyttäjien määrä kolmelta kuukaudelta ennen tehtävän avaamista.
nIssuesCreatedInProject	Projektiin luotujen tehtävien määrä kolmelta kaudelta ennen tehtävän avaamista.
nIssuesCreatedInProjectClosed	Projektiin luotujen ja suljettujen tehtävien määrä kolmelta kuukaudelta ennen tehtävän avaamista.
Taulukko jatkuu seuraavalla sivulla.	

Muuttuja	Selite
nCommitsInProject	Commit-tapahtumien määrä projektissa tehtävän avausta edeltävän kolmen kuukauden aikana.
nIssuesCreatedProjectT	Projektiin lisättyjen tehtävien määrä avausta edeltävän kahden viikon ja näytteenottohetken välillä.
nIssuesCreatedProjectClosedT	Projektiin lisättyjen ja suljettujen tehtävien määrä avausta edeltävän kahden viikon ja näytteenottohetken välillä.
nCommitsProjectT	Projektiin lisättyjen commit-tapahtumien lukumäärä tehtävän avausta edeltävän kahden viikon ja näytteenottohetken välillä.

kaista näyteaika-sulkeutumisaika-paria kohden, joissa $t_1 < t_2$. ($t_1 = \{0, 1, 7, 14, 30, 90, 180\}$, $t_2 = \{1, 7, 14, 30, 90, 180, 365\}$) Tehtävän avaamisajankohdaksi laskettiin sen luomishetki, ja sulkemisajankohdaksi tehtävän tapahtumahistoriasta löytyneen “closed”-tapahtuman luomishetki.

Kullekin luokittimelle opetus- ja testidatan määrä vaihteli, sillä näytteenottohetkellä jo suljettujen tehtävien sulkeutumisajan ennustaminen olisi turhaa. Sellaiset sulkeutumattomat tehtävät, joiden avaamisesta ei ollut vielä kulunut ennustettavan luokan verran aikaa verrattuna opetus- ja testidatan jakamisen päivämäärään, jätettiin pois, jottei päivämäärän jälkeistä tietoa päädy luokittimen opetukseen. Samoin testidatasta jätettiin pois sellaiset sulkeutumattomat tehtävät, joiden avaamisesta ei ollut kulunut riittävästi aikaa, jotta voisi tietää sulkeutuvatko ne ennustettavan ajan sisällä.

Mallin opettamiseen käytetty koodi oli saatavilla, ja sitä kyettiin hyödyntämään pienin muutoksin. Koodissa oli vielä jäljellä alkuperäisen tutkimuksen aikana käyttämättä jääneiden muuttujien käsittelyyn liittyviä kohtia, jotka poistettiin.

4.7 Eksploratiivinen osa: Ohjelmistovirheen korjausajan ennustaminen

Tutkimuksen eksploratiivisen osan päämääränä oli selvittää koneoppimismenetelmin luodun mallin kykyä ennustaa ohjelmistovirheen korjausaikaa virheraportin perusteella. Sitä varten

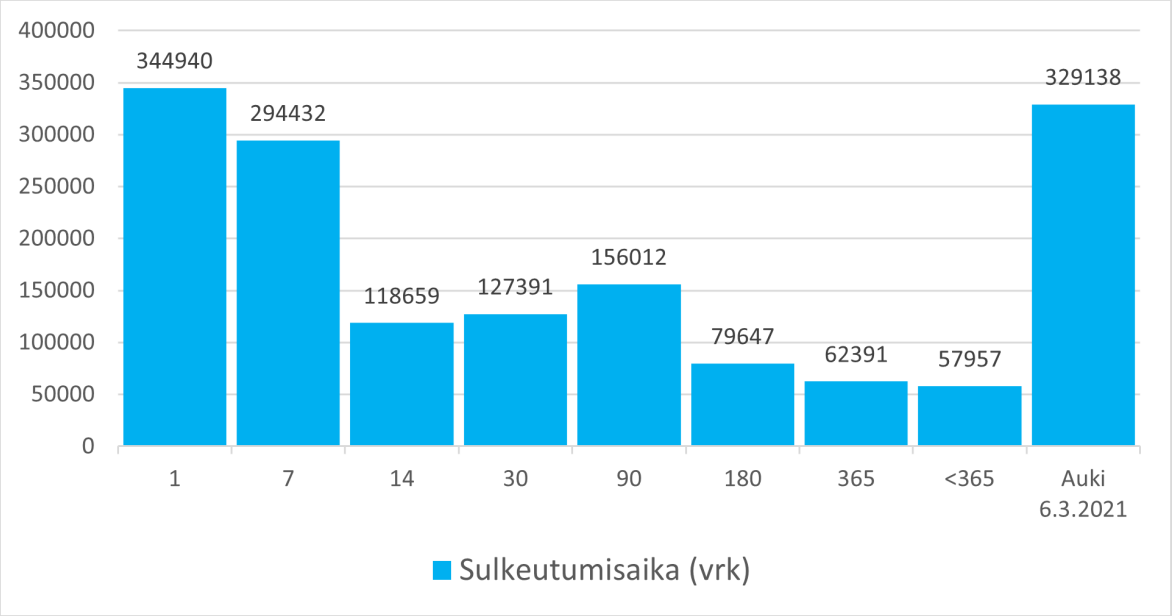
aineisto suodatettiin varioimalla replikaatiotutkimuksen suodatuksen ehtoja. Mukaan otettavat ohjelmistoprojektit valittiin eksploratiivisessa osassa käyttämällä replikaatiotutkimuksen suodatusehtoja 1-5 sellaisenaan. Koska virheraportit ovat osajoukko kaikista tehtävistä, täytyi raportteja sisältävän datajoukon muodostamiseksi tunnistaa virheraportit muiden tehtävien seasta. Virheraporttien tunnistamiseksi käytettiin tehtäviin liitettyjä nimikkeitä. Tehtävä todettiin virheraportiksi, jos siihen liitetty nimike vastasi SQL lauseen ehtoa

```
WHERE name LIKE '%ug'
AND name NOT like '%not%'
```

Tällä kriteerillä virheraporteiksi tunnistettiin ne tehtävät, joiden nimikkeeksi oli asetettu GitHubin vakionimike "bug", tai jokin ohjelmistovirheeseen viittaava kustomoitu nimike, kuten "Bug", "db-bug"tai "easy first bug". Pois karsiutui kieltävät nimikkeet kuten "not a bug". Näillä ehdoilla aineistosta löytyi 1 570 567 virheraportiksi tunnistettua tehtävää, eli replikaatiossa käytetyistä tehtävistä 12.3% tunnistettiin virheraporteiksi. Virheraportteja löytyi 31 120:sta ohjelmistoprojektista, eli 80.0% replikaatiotutkimuksen projekteista sisälsi ainakin yhden virheraportiksi tunnistetun tehtävän. Raporteista valtaosa oli suljettu yhden vuorokauden tai viikon kuluttua avaamisesta (Kuvio 3). Raportteja, joiden sulkeutumisaikaa ei tiedetä, oli 329 138 kappaletta.

Opetettavan luokittimen tulosta pyrittiin parantamaan myös Lamkanfin ja Demeyerin (2012) esittämällä keinolla poistamalla aineistosta ne raportit, joiden sulkeutumisaika oli alle puolet aineiston sulkeutumisaikojen alakvartaalista.

Samoin kuin replikaatiossa, virheraporttien tilasta kerättiin näytteet 0, 1, 7, 14, 30, 90 ja 180 päivää avaamisen jälkeen. Näytteillä opetettiin Random Forest -luokitin arvioimaan, onko raportti suljettu 1, 7, 14, 30, 90, 180 tai 365 päivän kuluttua avaamisesta. Huomattakoon, että raportin sulkeutuminen ei välttämättä tarkoita sitä, että ohjelmistovirhe on tullut korjatuksi. Raportin sulkemisen syynä voi olla myös esimerkiksi se, että raportoitua virhettä ei ole saatu toistettua, se on korjattu jonkin toisen päivityksen yhteydessä tai sitä ei aiota korjata ollenkaan.



Kuvio 3: Virheraporttien määrä eri sulkeutumisaikaikkunoissa

5 Tulokset

Tässä luvussa esitellään tutkimuksen kokeellisen osan tuloksia. Tuloksina saatiin sekä replikaatiosta että ohjelmistovirheraporttien sulkeutumisajan ennustamiseen pyrkivästä kokeesta Random Forest -luokittimella kehitetyt mallit, joiden hyvyyttä voidaan tarkastella useiden muuttujien avulla. Erityisesti replikaatiotutkimuksen kohdalla tuloksia on tarkasteltava suhteessa replikoituun tutkimukseen.

5.1 Replikaatiotutkimus

Replikaatiotutkimuksessa luokitin sovitettiin 7 266 657:n tehtävän opetusdatalla ja sitä testattiin 5 483 758:n tehtävän testausdatalla. Tulokset eivät olleet aivan yhdenmukaiset Kikasin, Dumasin ja Pfahlin (2016) tutkimuksen kanssa, mutta kuitenkin samansuuntaisuudessaan vahvistavat alkuperäisiä tuloksia (Taulukko 3).

Alkuperäisessä tutkimuksessa ilmoitettuihin tuloksiin liittyi virhe opetusdatan jaossa ja satunnaislukugeneraattorin alustuksessa, jotka on myöhemmin korjattu tutkimuksen lisämateriaaleissa[1]. Replikaatiotutkimuksen tuloksia verrattiin korjauksen jälkeen saataviin tuloksiin, koska myös replikaatiotutkimuksessa käytettiin korjattua versiota lähdekoodista. Replikaation ja replikoitavan tutkimuksen tuloksia vertailtiin siten, että vertailtavina olivat luokittimet, joiden opetusdatan näytteenottoajat olivat samat ja joilla oli sama ennustettava tehtävän sulkeutumisaika. Tällöin havaittujen erojen tulisi johtua pelkästään aineiston eroavuuksista. Eri sulkeutumisaikaa ennustavien luokitinten vertailussa on huomioitava tehtävien erilaisuus. Tehtävän ollessa avattu vasta muutama vuorokausi aiemmin, voi melko hyvällä onnistumistodennäköisyydellä sen arvata sulkeutuvan vuoden sisällä. Toisaalta jos tehtävä on ollut auki jo puoli vuotta, ei ole ollenkaan niin varmaa, että se sulkeutuu seuraavan puolen vuoden aikana.

Ensimmäinen tutkimuskysymys koski koneoppimismenetelmin opetetun luokittimien kykyä ennustaa kehitystehtävän sulkeutumisaikaa. Luokittimien ennustuskykyä arvioitiin niiden AUC- ja F1-arvojen perusteella. Luokittimen AUC-arvo kuvaa ROC-käyrän (receiver opera-

1. https://github.com/riivo/github-issue-lifetime-prediction

Taulukko 3: Replikaatiotutkimuksen ja alkuperäisen tutkimuksen luokittimien AUC- ja F1-arvot. Vaaka-akselilla ennustettava sulkeutumisaika ja pystyakselilla syötteeksi annetun tehtävän avaamisesta kulunut aika.

AUC	**Replikaatio**							**Alkuperäinen tutkimus**						
Ennuste	1	7	14	30	90	180	365	1	7	14	30	90	180	365
Ikä (vrk)														
0	0.66	0.66	0.66	0.66	0.67	0.67	0.69	0.65	0.66	0.66	0.67	0.68	0.68	0.66
1		0.67	0.67	0.67	0.67	0.68	0.74		0.64	0.64	0.65	0.66	0.67	0.65
7			0.68	0.68	0.68	0.69	0.75			0.64	0.65	0.65	0.67	0.65
14				0.69	0.68	0.69	0.75				0.65	0.66	0.67	0.64
30					0.68	0.68	0.75					0.66	0.68	0.64
90						0.68	0.74						0.69	0.66
180							0.75							0.69
F1	**Replikaatio**							**Alkuperäinen tutkimus**						
0	0.39	0.52	0.57	0.61	0.68	0.72	0.77	0.45	0.61	0.67	0.73	0.80	0.86	0.89
1		0.34	0.42	0.50	0.60	0.66	0.76		0.41	0.50	0.57	0.66	0.77	0.85
7			0.21	0.35	0.49	0.58	0.71			0.25	0.42	0.57	0.69	0.79
14				0.24	0.43	0.54	0.70				0.30	0.52	0.65	0.75
30					0.31	0.46	0.67					0.41	0.58	0.73
90						0.26	0.57						0.36	0.65
180							0.44							0.50

ting characteristic curve) alle jäävää pinta-alaa, eli luokittelun kynnysarvosta riippumatonta luokittelukykyä. ROC-käyrä kuvaa luokittimen tuottamien oikeiden ja väärien positiivisten luokitusten suhdetta eri kynnysarvoilla. Tämän tutkimuksen kontekstissa AUC kertoo todennäköisyyden, jolla luokitin antaa satunnaiselle ennustettavan ajan sisällä sulkeutuvalle tehtävälle suuremman arvon kuin satunnaiselle tehtävälle, joka ei sulkeudu ennustettavan ajan sisällä. Tarkkuudella (precision) tarkoitetaan luokittimen tekemien oikeiden positiivisten luokitusten osuutta kaikista sen tekemista positiivisista luokituksista. Herkkyys (recall) on oikeiden positiivisten luokitusten osuus kaikista näytteistä, jotka tulisi luokitella positiiviseksi. F1-arvo on luokittimen tarkkuuden ja herkkyyden harmoninen keskiarvo.

Alkuperäisen tutkimuksen tuloksista poiketen replikaatiotutkimuksessa muodostettujen luokittimien AUC-arvot (*Area Under the ROC Curve*) olivat heikoimmillaan silloin, kun tehtävän sulkeutumisaikaa pyrittiin ennustamaan juuri luodun tehtävän tiedoilla, joissa ei vielä voinut hyödyntää dynaamisia muuttujia. Alkuperäiseen tutkimukseen verrattuna juuri luotujen tehtävien sulkeutumisaikaa ennustettaessa mallien AUC oli keskimäärin 0.6% alku-

peräistutkimusta heikompi, kun ennustettava sulkeutumisaika oli 1-180 vuorokautta. Kun malli ennusti vasta avatun tehtävän sulkeutumista vuoden sisällä, saatu AUC 5.0% parempi, kuin alkuperäistutkimuksessa. Luokittimilla, jotka oli opetettu vähintään yhden vuorokauden ikäisillä tehtävillä, AUC oli lähes kaikissa tapauksissa hieman parempi kuin alkuperäistutkimuksessa. Kaiken kaikkiaan replikaatiossa muodostettujen luokittimien AUC parani eniten vuoden sisällä sulkeutuvia tehtäviä ennustettaessa. Mielenkiintoista oli se, että alkuperäistutkimuksessa luokitinten AUC oli pääsääntöisesti heikompi silloin, kun ennustettava sulkeutumisaika oli 365 vuorokautta kuin silloin, kun sulkeutumisaika oli 180 vuorokautta. Replikaatiossa luokitinten AUC oli selkeästi parempi 365 vuorokauden sisällä sulkeutuvia tehtäviä ennustettaessa.

F1-arvot olivat kaikissa replikaatiossa luoduissa luokittimissa heikommat, kuin alkuperäistutkimuksessa. Arvot heikkenivät keskimäärin 15.1% luokitinta kohden. Voimakas F1:n heikkeneminen johtui vasta avattuja virheraportteja luokiteltaessa herkkyyden heikkenemisestä, eli luokittimet tunnistivat pienemmän osuuden niistä tehtävistä, jotka sulkeutuvat ennustettavan ajan sisällä. Replikaatiossa muodostettujen luokittimien herkkyydet olivat välillä 0.622-0.707, ja alkuperäisen tutkimuksen välillä 0.608-0.873. Juuri luoduilla tehtävillä opetettujen luokitinten herkkyys oli keskimäärin 17.3% huonompi, kuin alkuperäistutkimuksessa. Niillä luokittimilla, joiden opetusdatan tehtävien näytteenottoaika oli yksi tai useampi vuorokausi, herkkyys oli lähempänä alkuperäistutkimuksen vastaavia luokittimia. Niissä herkkyys parani keskimäärin 0.7%. Näiden luokittimien kohdalla F1:n heikkeneminen johtui luokitinten tarkkuuden laskemisesta. Replikaation luokitinten tarkkuus laski 20.6% verrattuna replikoitavan tutkimuksen vastaaviin luokittimiin.

Kuten alkuperäistutkimuksessa, myös replikaatiossa luokittimien F1-arvot ovat sitä paremmat, mitä suurempi on näytteenottohetken ja ennustettavan sulkeutumisajan väli. Suurimmillaan arvot ovatkin silloin, kun ennustettava luokka on 365 vuorokautta. Tämä johtuu siitä, että suurin osa tehtävistä sulkeutuu vuoden sisällä, ja mitä tuoreemmasta tehtävästä on kyse, sitä enemmän löytyy auki olevia tehtäviä aineistoon.

Arvoja tarkastellessa tulee siis ottaa huomioon se, että esimerkiksi kahden viikon sisällä suljetut tehtävät tulisi luokitella positiiviksi kaikilla luokittimilla, joilla ennustettava sulkeutumisaika on kaksi viikkoa tai enemmän. Luokittimien vertailu toisiinsa on mielekkäämpää,

jos niihin sovelletaan samaa testijoukkoa, johon on valittu vain ne tehtävät, jotka ovat auki vielä suurimman näytteenottoajan kohdalla (Taulukko 4). Tällöin testijoukossa on vain sellaisia tehtäviä, joiden ei tulisi saada positiivista luokitusta pienemmällä ennustetulla sulkeutumisajalla.

Taulukko 4: Replikaatiotutkimuksen luokittimien tulokset, kun testijoukon koko on vakio ennustetun sulkeutumisajan sisällä.

Tehtävän ikä	AUC	Tarkkuus	Herkkyys	F1	TP	FP	FN	TN
	Ennustettu sulkeutumisaika 365 vuorokautta (N=905395)							
0	0.507	0.159	0.401	0.228	59354	314414	88571	443056
1	0.627	0.213	0.509	0.301	75364	277679	72561	479791
7	0.670	0.260	0.551	0.353	81484	232360	66441	525110
14	0.684	0.266	0.588	0.366	86982	240528	60943	516942
30	0.698	0.273	0.610	0.378	90269	239827	57656	517643
90	0.720	0.282	0.686	0.399	101450	258861	46475	498609
180	0.749	0.341	0.622	0.441	91938	177521	55987	579949
	Ennustettu sulkeutumisaika 180 vuorokautta (N=1807663)							
0	0.534	0.110	0.491	0.179	91860	744823	95176	875804
1	0.586	0.123	0.550	0.202	102958	731847	84078	888780
7	0.615	0.133	0.565	0.216	105619	687321	81417	933306
14	0.627	0.137	0.608	0.224	113726	714647	73310	905980
30	0.643	0.143	0.636	0.233	118948	713939	68088	906688
90	0.682	0.160	0.664	0.258	124244	650228	62792	970399
	Ennustettu sulkeutumisaika 90 vuorokautta (N=2575223)							
0	0.594	0.178	0.602	0.275	225392	1039978	149090	1160763
1	0.626	0.190	0.633	0.292	237021	1011710	137461	1189031
7	0.645	0.194	0.614	0.294	229921	958039	144561	1242702
14	0.656	0.197	0.645	0.302	241572	982879	132910	1217862
30	0.677	0.203	0.692	0.314	259010	1016297	115472	1184444
	Ennustettu sulkeutumisaika 30 vuorokautta (N=3105779)							
0	0.612	0.124	0.618	0.207	183061	1292036	113188	1517494
1	0.644	0.131	0.653	0.218	193475	1285917	102774	1523613
7	0.668	0.137	0.656	0.226	194325	1226043	101924	1583487
14	0.690	0.142	0.696	0.235	206273	1251115	89976	1558415
	Ennustettu sulkeutumisaika 14 vuorokautta N=3465139)							
0	0.620	0.108	0.615	0.184	171491	1417118	107495	1769035
1	0.653	0.114	0.634	0.193	176899	1381372	102087	1804781
7	0.684	0.124	0.631	0.207	175980	1244529	103006	1941624
	Ennustettu sulkeutumisaika 7 vuorokautta (N=4173414)							
0	0.638	0.221	0.634	0.328	431114	1518904	249205	1974191
1	0.666	0.228	0.643	0.337	437155	1478466	243164	2014629

Replikoitavassa tutkimuksessa tutkijat havaitsivat lievän laskevan trendin luokittimien herkkyydessä, kun näytteenottoaikaa kasvatettiin. Herkkyyden laskeminen selittyi sillä, että oikeiden positiivisen luokitusten (TP) määrä ei kasvanut oikeiden negatiivisten luokitusten (TN) mukana. Replikaatiossa havaittiin kuitenkin alkuperäisestä tutkimuksesta poiketen herkkyyden kasvua näytteenottoajan pitenemisen myötä. Oikeiden positiivisten tulosten määrä kasvoikin siis sitä mukaa, mitä kauemmin tehtävistä oli kerätty tietoa.

Toisessa tutkimuskysymyksessä oltiin kiinnostuneita siitä, mitkä muuttujat ovat tärkeimpiä kehitystehtävän sulkeutumisaikaa ennustettaessa. Kikas, Dumas ja Pfahl (2016) selvittivät, mitkä muuttujat olivat luokittimien kannalta tärkeimpiä tarkastelemalla muuttujien MDI-arvoja (*mean decrease in impurity*). MDI-arvo on keskiarvo muuttujan aiheuttamasta Gini-epäpuhtauden vähenemästä kaikissa satunnaisen metsän puissa. Tutkimuksessa käytetty Scikit-learn -kirjasto palauttaa muuttujien MDI-arvot normalisoituna, jolloin niiden vertailu on helppoa. He olivat kiinnostuneita siitä, kuinka tehtävän aikana mahdollisesti arvoaan muuttavat dynaamiset muuttujat vaikuttavat luokittimiin. Tätä he arvioivat vertailemalla keskenään muuttujien MDI-arvoja luokittimilta, jotka oli opetettu tuoreilla tehtävillä ja seitsemän vuorokauden ikäisillä tehtävillä. Muuttujan tärkeys ei itsessään kerro mitään muuttujan selitysvoimasta, vaan siitä kuinka suuressa osassa muttuja on kyseisessä mallissa.

Taulukossa 5 esitetään muuttujien suhteellinen tärkeys luokittimilla, jotka ennustavat vasta avatun tehtävän sulkeutumista 30:n ja 180:n vuorokauden sisällä ja viikon ikäisen tehtävän sulkeutumista 180:n ja 365:n vuorokauden sisällä. Tärkein muuttuja saa arvon 100, ja muiden muuttujien tärkeys ilmoitetaan suhteessa tärkeimpään muuttujaan.

Replikaatiotutkimuksessa juuri avattujen tehtävien datalla opetettujen luokitinten kohdalla tärkeimmiksi muuttujaksi osoittautuivat *nIssuesCreatedProjectClosedT*, *nIssuesCreatedInProjectClosed*, *nCommitsProjectT* ja *nCommitsInProject*, jotka olivat neljä tärkeintä muuttujaa kaikissa muissa luokittimissa, paitsi kun ennustettavana sulkeutumisaikana oli yksi vuorokausi. Tehtäviä koskevat muuttujat kuvaavat uuden tehtävän avausta edeltävän kahden viikon (*nIssuesCreatedProjectClosedT*) tai kolmen kuukauden (*nIssuesCreatedInProjectClosed*) aikana avattujen ja jo suljettujen tehtävien määrää, minkä voisi nähdä merkkinä siitä, kuinka aktiivisesti projektissa hyödynnetään GitHubin projektinhallinnan työkaluja. Commit-tapahtumien määrästä kertovien muuttujien voi nähdä heijastavan projektin yleistä

Taulukko 5: Muuttujien suhteellinen tärkeys MDI-arvon perusteella replikaatiotutkimuksen luokittimissa.

	Tehtävän ikä - sulkeutumisaika (vrk)			
Muuttuja	**0 - 30**	**0 - 180**	**7 - 180**	**7 - 365**
nIssuesCreatedProjectClosedT	**100**	**100**	85	84
nCommitsInProject	93	90	27	24
nCommitsProjectT	86	70	14	11
nIssuesCreatedInProjectClosed	73	75	21	19
textScore	53	41	11	14
nIssuesCreatedProjectT	22	21	5	5
nIssuesCreatedInProject	14	13	3	3
nIssuesByCreator	11	11	3	4
nIssuesByCreatorClosed	10	14	5	5
issueCleanedBodyLen	6	11	7	9
nCommitsByCreator	4	3	1	0
nCommitsByActorsT	0	0	**100**	**100**
nAssignmentsT	0	0	45	34
nLabelsT	0	0	35	39
nReferencedByT	0	0	16	12
nMentionedByT	0	0	10	12
nSubscribedByT	0	0	8	10
nActorsT	0	0	5	5
textScoreComments	0	0	2	3
nCommentsT	0	0	2	2
meanCommentSizeT	0	0	0	1
nCommitsByUniqueActorsT	0	0	0	0

aktiivisuutta. Tehtävän tekstuaalisesta datasta laskettu *textScore*-muuttuja oli tärkeysjärjestyksessä toisena, kun ennustettava sulkeutumisaika oli yksi vuorokausi, ja viidentenä muissa luokittimissa. Erityistä hyötyä tiedonlouhinnasta tekstidatasta voisi siis päätellä olevan silloin, kun halutaan tunnistaa hyvin nopeasti avaamisen jälkeen suljettavia tehtäviä. Suhteellisesti *textScore*-muuttujan tärkeys pienempi, kun ennustettava sulkeutumisaika oli pidempi. Alkuperäisen tutkimuksen tuloksista poiketen replikaatiossa ei tehtävän kommenttien lukumäärä korostunut tärkeänä muuttujana, koska aineistossa muuttujan arvo oli erisuuri kuin 0 vain 547:ssä tehtävässä 286:ssa projektissa.

Tarkasteltaessa muuttujien tärkeyttä luokittimilla, joiden opetusdatan näytteet oli otettu seitsemän vuorokauden ikäisistä tehtävistä, voi huomata dynaamisten muuttujien nousevan tärkeysjärjestyksessä ylös. Kun ennustettava sulkeutumisaika on 180 vuorokautta, neljäksi tär-

keimmäksi muuttujiksi muodostuivat *nCommitsByActorsT*, *nIssuesCreatedProjectClosedT*, *nAssignmentsT* ja *nLabelsT* Alkuperäisessä tutkimuksessa tehty havainto siitä, että dynaamisten muuttujat ovat hyödyllisiä ennustettaessa tehtävän sulkeutumista pitkän ajan kuluessa, sai tukea. Mielenkiintoista oli kuitenkin se, että dynaamisten muuttujien tärkeysjärjestys oli replikaatiossa hyvin erilainen alkuperäiseen tutkimukseen verraten. Alkuperäisessä tutkimuksessa muun muassa tehtävän kommenttien ja siihen osallistuvien käyttäjien määrä olivat tärkeysjärjestyksessä korkealla, kun taas tehtävään liitettyjen nimikkeiden lukumäärä oli vähiten tärkeiden muttujien joukossa. Replikaatiossa taas nimikkeiden määrä oli suhteellisesti tärkeä luokittimilla, jotka ennustivat tehtävän sulkeutumista yli 90 vuorokauden sisällä.

5.2 Eksploratiivinen osa: korjausajan ennustaminen virheraportin perusteella

Replikaation jälkeen tutkimuksessa suoritettiin vielä eksploratiivinen vaihe, jossa muodostettiin aiemman tutkimuksen menetelmää mukaillen luokittimia, joiden tehtävänä oli ennustaa ohjelmistovirheen korjausaikaa virheraportin kirjaamisesta sen sulkemiseen. Tämä toteutettiin suodattamalla replikaatiossa käytetystä aineistosta pois kaikki muut kehitystehtävät kuin virheraportit, ja ajamalla sama skripti satunnaisten metsien opettamiseksi. 1 570 567:stä virheraportista 857 442 (54.6%) varattiin opetusdataksi ja 713 125 (45.4%) testausdataksi.

Tulosten tarkastelun jälkeen päätettiin kokeilla vielä Lamkanfin ja Demeyerin (2012) esittämää tapaa poistaa poikkeamia datasta jättämällä pois kaikki ne virheraportit, joiden sulkeutumisaika oli alle puolet virheraporttien sulkeutumisaikojen alakvartaalista. Poikkeamien poiston jälkeen virheraportteja oli 1 324 966, joista opetusdataksi valikoitui 715 418 (54.0%) ja testausdataksi 609 548 (46.0%) raporttia. Kokeilu vaikutti lähinnä luokittimiin, joiden syötteenä oli alle vuorokauden ikäisen virheraportit. Hyvin nopeasti sulkeutuneiden raporttien poistaminen datasta heikensi luokittimien tuloksia hieman, joten tarkasteltavaksi päätettiin jättää alkuperäiset tulokset.

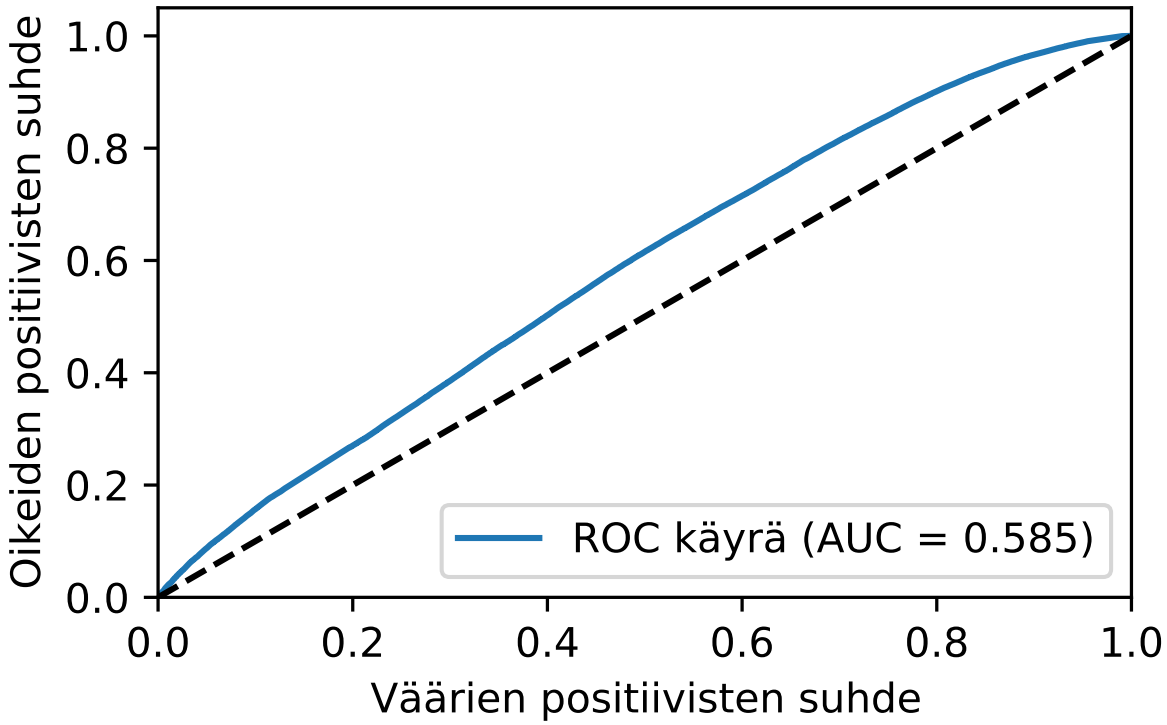
Siinä missä replikaatiossa oltiin kiinnostuneita yleisesti kehitystehtävien valmistumisajasta, eksploratiivinen osa keskittyi erityisesti ohjelmistovirheiden virheraporttien sulkeutumi-

Taulukko 6: Eksploratiivisen osan luokittimien F1- ja AUC-arvot sekä täsmällisyys

Ennuste / **Ikä (vrk)**	1	7	14	30	90	180	365
			AUC				
0	0.581	0.594	0.595	0.594	0.585	0.579	0.634
1		0.637	0.634	0.627	0.610	0.605	0.666
7			0.662	0.640	0.613	0.606	0.670
14				0.644	0.609	0.600	0.663
30					0.603	0.595	0.653
90						0.607	0.652
180							0.657
				F1			
0	0.328	0.453	0.494	0.533	0.600	0.692	0.779
1		0.380	0.462	0.520	0.590	0.665	0.769
7			0.262	0.406	0.512	0.615	0.748
14				0.297	0.452	0.561	0.715
30					0.340	0.491	0.697
90						0.290	0.633
180							0.492
				Täsmällisyys			
0	0.645	0.583	0.566	0.552	0.547	0.588	0.655
1		0.641	0.607	0.589	0.566	0.581	0.651
7			0.618	0.620	0.573	0.576	0.642
14				0.705	0.572	0.561	0.618
30					0.620	0.550	0.617
90						0.486	0.599
180							0.563

saikaan. Virheraporttien sulkeutumisajan ennustamisen voisi nähdä olevan tehtävänä hyvin samankaltainen kuin ohjelmistovirheen korjausajan ennustaminen. Tulosten perusteella virheraporttien sulkeutumisajan ennustaminen käyttäen satunnaisia metsiä ja replikaatiotutkimuksessa käytettyä menetelmää, tuottaa luokittimia joiden tulokset replikaatiossa tuotettujen luokittimien tuloksia heikompia.

Erityisen haasteellista vaikutti olevan virheraportin sulkeutumisajan ennustaminen, kun käsiteltiin juuri avattuja raportteja. Näillä luokittimilla AUC-arvot olivat välillä 0.579 - 0.634 (Taulukko 6). Esimerkiksi verrattuna tutkimuksen replikaatio-osassa muodostettujen luokittimien AUC-arvoihin, jotka olivat kaikki yli 0.659, variaatio-osan tulokset jäävät selkeästi heikommaksi. Tarkasteltaessa virheraportin sulkeutumista 180:n vuorokauden sisällä ennustavan luokittimen ROC-käyrää (Kuvio 4), voidaan havaita, että positiivisen luokituksen



Kuvio 4: ROC-käyrä luokittimesta, joka ennustaa juuri avatun virheraportin sulkeutumista 90 vuorokauden sisällä avaamisesta.

raja-arvosta riippumatta on mahdotonta saada aikaan luokitinta, jolla olisi suuri positiivisten luokitusten suhde eli herkkyys, mutta pieni väärien positiivisten luokitusten suhde kaikkiin raportteihin, joiden oikea luokitus olisi negatiivinen.

Luokittimien F1-arvoja tarkastellessa voi todeta, että mitä pidempi on tehtävän iän ja ennustettavan sulkeutumisajan ero, sitä parempia tuloksia voi luokittimelta odottaa. F1-arvojen parantuminen johtuu enimmäkseen luokitinten tarkkuuden paranemisesta eli siitä, että mallit antavat positiivisen luokituksen suuremmalle osalle ennustettavan ajan sisällä sulkeutuvista virheraporteista. Esimerkiksi juuri avattujen raporttien sulkeutumisaikaa ennustettaessa tarkkuus paranee 0.457:stä 0.934:ään, kun ennustettava sulkeutumisaika nostetaan seitsemästä vuorokaudesta vuoteen. Tarkkuuden paraneminen johtuu siitä, että virheraportin sulkeutumisajan ennustaminen on sitä helpompaa mitä pidempi ennustettava aikaikkuna on.

Taulukko 7: Eksploratiivisen osan luokittimien tulokset, kun testijoukon koko on vakio ennustetun sulkeutumisajan sisällä.

Tehtävän ikä	AUC	Tarkkuus	Herkkyys	F1	TP	FP	FN	TN
	Ennustettu sulkeutumisaika 365 vuorokautta (N=64560)							
0	0.454	0.277	0.402	0.328	8022	20937	11924	23677
1	0.493	0.291	0.412	0.341	8223	20002	11723	24612
7	0.536	0.334	0.467	0.389	9309	18586	10637	26028
14	0.558	0.353	0.487	0.409	9713	17817	10233	26797
30	0.581	0.354	0.525	0.423	10477	19151	9469	25463
90	0.622	0.366	0.627	0.463	12508	21628	7438	22986
180	0.657	0.384	0.686	0.492	13684	21948	6262	22666
	Ennustettu sulkeutumisaika 180 vuorokautta (N=176782)							
0	0.448	0.132	0.443	0.203	11766	77694	14820	72502
1	0.471	0.134	0.453	0.206	12037	78011	14549	72185
7	0.495	0.151	0.498	0.232	13240	74154	13346	76042
14	0.514	0.155	0.499	0.236	13257	72373	13329	77823
30	0.540	0.169	0.561	0.259	14904	73527	11682	76669
90	0.607	0.183	0.697	0.290	18520	82805	8066	67391
	Ennustettu sulkeutumisaika 90 vuorokautta (N=279158)							
0	0.495	0.187	0.407	0.256	21914	95534	31886	129824
1	0.519	0.192	0.456	0.271	24550	103027	29250	122331
7	0.549	0.213	0.481	0.295	25903	95794	27897	129564
14	0.566	0.222	0.512	0.310	27553	96647	26247	128711
30	0.603	0.256	0.508	0.340	27339	79572	26461	145786
	Ennustettu sulkeutumisaika 30 vuorokautta (N=355343)							
0	0.535	0.140	0.396	0.207	18478	113201	28213	195451
1	0.568	0.154	0.471	0.232	21973	120480	24718	188172
7	0.606	0.177	0.518	0.264	24173	112583	22518	196069
14	0.644	0.216	0.474	0.297	22137	80433	24554	228219
	Ennustettu sulkeutumisaika 14 vuorokautta (N=410734)							
0	0.552	0.124	0.404	0.190	18205	128667	26877	236985
1	0.595	0.143	0.499	0.222	22493	135337	22589	230315
7	0.662	0.166	0.619	0.262	27896	139677	17186	225975
	Ennustettu sulkeutumisaika 7 vuorokautta (N=528228)							
0	0.577	0.257	0.419	0.318	47659	137971	66213	276385
1	0.637	0.303	0.512	0.380	58276	134170	55596	280186

Dynaamisten muuttujien merkitystä luokittimen hyvyyteen voi tarkastella taulukon 7 avulla, missä luokittimien testidata on vakioitu ottamalla siihen mukaan vain niitä virheraportteja, joiden todellinen sulkeutumisaika on suurempi kuin suurin luokitinten näytteenottoaika. Esimerkiksi siis 30 vuorokauden sisällä sulkeutumista ennustavat luokittimet testattiin ra-

porteilla, jotka olivat auki vielä 14 vuorokauden kuluttua avaamisesta. Vertailemalla vasta avattuja virheraportteja ja viikon verran auki olleita raportteja käsitteleviä malleja, dynaamisten muuttujien ansiosta luokittimien AUC paranivat noin 11%-20%, tarkkuus 14%-34%, herkkyys 12%-53% ja F1-arvo 14%-38%.

Vakioidulla testijoukolla testattujen luokitinten tuloksista nousi esiin huomattavan pieni AUC-arvo luokittimilla, joiden syötteenä oli alle viikon ikäiset raportit ja ennustettava raportin sulkeutumisaika oli 90, 180 tai 365 vuorokautta. Luokittimet, joiden AUC oli alle 0.5 selkeästi alisuorittivat kyseisillä testijoukoilla. Esiin nousi myös väärien positiivisten luokitusten (FP) moninkertainen määrä suhteessa oikeisiin positiivisiin luokituksiin (TP). Esimerkiksi kun luokitin ennustaa kaksi viikkoa vanhan virheraportin sulkeutuvan viimeistään kuukauden ikäisenä, on ennuste todennäköisemmin väärin kuin oikein. Tällä luokittimella oikeita positiivisia luokituksia oli 22 137 ja vääriä 80 433 kappaletta, eli 78% positiivisista luokituksista oli väärin. Toisaalta luokittimet antoivat negatiivisia luokituksia selvästi paremmalla tarkkuudella, jolloin edellisen esimerkin luokittimella negatiivinen luokitus on 90%:ssa tapauksista oikein.

Muuttujan tärkeys luokittimelle kuvaa sitä, kuinka suuressa osassa luokittimen rakenteesta ja luokituksista muuttuja on. Muuttujan tärkeyden mittaaminen on yksi keino selittää koneoppimistekniikoilla muodostettua mallia. Muuttujien tärkeys virheraporttien sulkeutumisaikaa ennustavissa luokittimissa laskettiin samalla tavalla MDI-arvon perusteella kuin replikaatiotutkimuksessa.

Taulukossa 8 on ilmoitettu muuttujien suhteellinen tärkeys juuri avattuja virherapotteja käsittelevissä luokittimissa. Raporttien tekstuaalisesta datasta laskettu *textScore* oli neljän tärkeimmän muuttujan joukossa kaikilla ennustettavilla sulkeutumisajoilla. Sen tärkeys kuitenkin laski ennustushorisontin pidentyessä. Tekstuaalisesta datasta kerätyn tiedon lisäksi raportin sulkeutumisaikaa juuri avaamisen jälkeen ennustettaessa tärkeimpien muuttujien joukkoon pääsi jokaisessa luokittimessa projektissa suljettujen tehtävien ja commit-tapahtumien määrää kuvaavia muuttujia. Lyhyen ajan sisällä (30 vuorokautta) sulkeutuvien virheraporttien kannalta raportin avaajan aloittamien kehitystehtävien määrä nousi tärkeäksi muuttujaksi, mutta sen merkitys väheni, kun ennustettava sulkeutumisaika kasvoi yli kuukauden pituiseksi. *IissueCleanedBodyLen*, eli raportin otsikon kuvauksen pituus erikoismerkkien ja

koodilohkojen poistamisen sekä tekstialkioiksi riisumisen jälkeen oli tärkein muuttuja erittäin nopeasti sulkeutuvia raportteja tunnistaessa. Pituuden merkitys kuitenkin väheni ennustettavan sulkeutumisajan kasvaessa. Vähiten tärkeitä muuttujia olivat dynaamiset muuttujat, joiden arvo joitain poikkeuksia lukuun ottamatta on virheraportin avaushetkellä nolla.

Taulukossa 9 on muuttujien suhteellinen tärkeys, kun luokiteltava raportti on avattu kaksi viikkoa aiemmin. Jotkin dynaamiset muuttujat, kuten rapottiin liitettyjen nimikkeiden (*nLabelsT*) ja virheraporttiin osallistuneiden tekemien commit-tapahtumien (*nCommitsByActorsT*) määrä, nousivat erityisen tärkeiksi kaikilla ennustettavilla sulkeutumisajoilla. Commit-tapahtumien määrää kuvaavia muuttujia oli tärkeimpien joukossa kaikissa luokittimissa, mutta avattujen ja suljettujen tehtävien määrään liittyvät eivät olleet yhtä tärkeitä kuin vasta avattuja virheraportteja käsiteltäessä. Joidenkin dynaamisten muuttujien, kuten *nSubscribedByT*:n ja *nMentionedByT*:n, suhteellinen tärkeys osoittautui hyvin vähäiseksi, mistä herääkin kysymys siitä, kannattaako näitä tietoja edes käyttää virheraportin sulkeutumisajan ennustamisessa.

Taulukko 8: Muuttujien suhteellinen tärkeys luokittimilla, jotka käsittelevät juuri avattuja virheraportteja. Lihavoituna on luokittimen neljä tärkeintä muuttujaa. Dynaamiset muuttujat, joiden suhteellinen tärkeys on alle yksi, on jätetty pois.

	Ennustettava sulkeutumisaika (vrk)						
Muuttuja	**1**	**7**	**14**	**30**	**90**	**180**	**365**
issueCleanedBodyLen	**100**	71	53	34	21	14	13
textScore	**97**	**94**	**74**	**61**	**60**	**48**	**49**
nIssuesByCreatorClosed	**92**	**93**	**75**	**62**	35	26	20
nCommitsByCreator	**62**	39	28	21	10	6	4
nIssuesCreatedProjectClosedT	51	**100**	**100**	**100**	**100**	**100**	**100**
nIssuesByCreator	49	33	23	17	14	14	13
nCommitsProjectT	42	**81**	**86**	**80**	**56**	47	46
nCommitsInProject	28	45	48	53	**63**	**65**	**71**
nIssuesCreatedProjectT	24	36	34	31	30	27	24
nIssuesCreatedInProjectClosed	23	45	53	55	55	**64**	**68**
nIssuesCreatedInProject	12	15	14	12	17	16	15
textScoreComments	1	1	2	2	3	4	5

Taulukko 9: Muuttujien suhteellinen tärkeys luokittimilla, jotka käsittelevät 14:n vuorokauden ikäisiä virheraportteja. Lihavoituna on luokittimen neljä tärkeintä muuttujaa.

	Ennustettava sulkeutumisaika (vrk)			
Muuttuja	**30**	**90**	**180**	**365**
nAssignmentsT	**100**	66	56	28
nReferencedByT	**84**	50	48	31
nCommitsByActorsT	**78**	**71**	**86**	**89**
nLabelsT	**67**	**100**	**97**	**100**
nIssuesCreatedProjectClosedT	59	68	**100**	**84**
textScore	59	**76**	70	**83**
nCommitsInProject	45	**70**	**81**	69
nIssuesCreatedInProjectClosed	30	29	33	31
nCommitsProjectT	24	22	19	15
nIssuesCreatedInProject	19	12	12	7
nIssuesCreatedProjectT	13	15	13	13
textScoreComments	11	9	7	6
nCommentsT	5	8	11	9
nIssuesByCreator	5	4	7	12
nIssuesByCreatorClosed	5	1	2	4
issueCleanedBodyLen	4	11	37	71
meanCommentSizeT	3	3	2	1
nMentionedByT	1	4	6	7
nSubscribedByT	1	3	4	5
nCommitsByCreator	1	1	2	6
nCommitsByUniqueActorsT	0	0	0	0

6 Pohdinta

Tutkielman empiirinen osuus koostui replikaatio-osasta ja eksploratiivisesta osasta. Replikaatio toteutettiin toistamalla Kikasin, Dumasin ja Pfahlin (2016) ohjelmistoprojektien kehitystehtävien valmistumisaikaa käsittelevä tutkimus yli kymmenen kertaa suuremmalla aineistolla, jossa oli tehtäviä yli kaksi kertaa alkuperäistä pidemmältä ajanjaksolta. Replikaatiotutkimus vahvisti alkuperäistutkimuksen vastaukset tutkimuskysymyksiin, mutta jotkin havainnot poikkesivat alkuperäisestä.

Koska replikaatiotutkimuksessa voitiin hyödyntää alkuperäisessä tutkimuksessa käytettyä koodia koneoppimismallin luomiseksi ja myös raakadatan rakenne oli sama, havaittujen erojen täytyy johtua joko datan lähteessä eli GitHubin ohjelmistoprojekteissa tapahtuneista muutoksista, tai sitten datan suodatus- ja muokkausvaiheessa tapahtuneesta erosta tutkimusten välillä. GitHubin kasvun myötä replikaatiossa oli yli kymmenkertainen määrä projekteja, joista 94.4% oli sellaisia, joita ei alkuperäisen tutkimuksen aikaan joko oltu perustettu, tai ne eivät läpäisseet suodatusprosessia. Muutokset GitHubista löytyvien projektien luonteesta ja GitHubin käyttäjien tavasta käyttää alustan tarjoamia työkaluja kehitystehtävien kirjaamiseen ja seurantaan voisi hyvinkin olla syy osaan havaituista eroista. Tällaisia muutoksia ei kuitenkaan tutkimuksessa mitattu, joten niiden vaikutus tuloksiin jää spekulaation varaan. Esimerkiksi henkilökohtaisten projektien määrän kasvu voisi aiheuttaa sen, että projektiin osallistuvien käyttäjien määrän ja aktiivisuuden seuraaminen on vähemmän hyödyllistä kuin aiemmin. Ohjelmistoprojektien heterogeenisyys on voinut kasvaa myös esimerkiksi käytettyjen teknologioiden, koon, aktiivisuuden, organisaatioiden ja työskentelytapojen osalta, mikä on voinut vaikeuttaa tehtävien sulkemisajan ennustamista.

Replikoinnin onnistumisen kannalta kriittinen vaihe oli datan suodatus ja käsittely koneoppimisalgoritmille sopivaan muotoon. Tämä vaihe oli selitetty alkuperäisessä tutkimuksessa sillä tarkkuudella, että tarvittavat skriptit ja tietokantakyselyt voitiin kehittää itse. Niitä ei kuitenkaan testattu alkuperäisen tutkimuksen lähtödataa vasten, joten ei ole varmuutta siitä, että vaiheet tehtiin täsmällisesti samalla tavalla. Suodatetusta datasta muuttujia laskettaessa huomattiin, että alkuperäisessä tutkimuksessa yhden muuttujan selite oli virheellinen, mikä huomattiin vain koska muuttujan laskeminen selitteen kuvaamalla tavalla osoittautui mah-

dottomaksi. Muut mahdolliset virheet joko alkuperäisen tutkimuksen raportoinnissa tai sen tulkinnassa ovat jääneet huomaamatta. Jos replikaatiossa muodostettujen luokittimien alkuperäistä hieman heikommat tulokset selittyvät ohjelmistoprojektien heterogeenisyyden kasvulla, on silloin syytä pohtia sitä, oliko projektien suodatusehdot riittävät ja ajankohtaiset. Tutkimuksessa suljettiin ulos ohjelmistoprojektit, joissa on avattu yli 2000 tehtävää kuukauden aikana. Tämän suodatusehdon tarkoituksena Kikasin, Dumasin ja Pfahlin (2016) mukaan tunnistaa ja suodattaa pois projektit, joissa on merkkejä tehtäviin liittyvästä datamigraatiosta toisesta järjestelmästä. Toisaalta tämä ehto suodatti pois joitain suuria avoimen lähdekoodin ohjelmistoprojekteja, kuten Microsoftin Visual Studio Code -projektin[1].

Replikaation tuloksena saatujen luokitinten AUC-arvot olivat välillä 0.658 - 0.751, mitä voi tulkita siten, että valituista muuttujista on hyötyä tehtävän sulkeutumisaikaa ennustettaessa, ja luokittimet kykenevät pisteyttämään tehtäviä. Kuitenkin suuri väärien luokitusten määrä suhteessa oikeisiin positiivisiin luokituksiin eritysesti lyhyen aikavälin ennustuksilla aiheuttaa sen, että luokittimien soveltaminen käytännössä voi olla hyvin haastavaa. Ohjelmistokehitystyössä voi olla vaikea perustella tekoälyn käyttöä tehtävään, jossa se on oikeassa vain hieman yli puolessa tapauksista, etenkin kun tehtävien suunnittelu ja aikataulutus on osa päivittäistä kehitystyötä. Toisaalta luokittimia voisi käyttää manuaalisen harkinnan tukena suurten tietomassojen vuoksi, jos halutaan esimerkiksi tunnistaa roikkumaan jääviä tehtäviä. Viikko tehtävän avaamisen jälkeen kerätyillä tiedoilla luokitin tunnistaa 55 prosenttia tehtävistä, jotka ovat auki vielä puolen vuoden päästä avaamisesta ja negatiivinen luokitus on väärin vain seitsemässä prosentissa kaikista negatiivisista luokituksista.

Luokitinten tuloksia ja muuttujien tärkeyssuhteita tarkastellessa voi todeta, että vasta avattuun tehtävään verraten jo yhdenkin vuorokauden aikana kerättävissä olevan tiedon mukaan ottaminen luokitinmalliin on kannattavaa, sillä luokitinten tulokset paranivat syötteenä toimivien tehtävien iän myötä ja viikon ikäisiä tehtäviä luokiteltaessa dynaamiset muuttujat olivat tärkeimpiä.

Replikaatiossa osan muuttujista suhteellinen tärkeys erosi huomattavasti alkuperäisen tutkimuksen tuloksista. Muuttujan tärkeys kertoo missä määrin kyseinen muuttuja selittää luokitinmallin tekemiä luokituksia, eikä se aina ole merkki siitä, että se selittäisi hyvin ennustet-

1. https://github.com/microsoft/vscode

tavaa muuttujaa. Muuttujien tärkeyden mittarina käytetyn MDI-arvon tiedetään olevan epäluotettava, jos luokittimen muuttujat korreloivat keskenään tai muuttujien saamien arvojen mittakaavat eroavat suuresti (Strobl ym. 2007; Strobl ym. 2008). Tämän tutkimuksen osalta molemmat syyt voivat vääristää tuloksia.

Tutkimuksen ekploratiivisessa osassa tutkittiin ohjelmistoprojektien virheraporttien sulkeutumisajan ennustamista luokittimilla, jotka muodostettiin replikoidun tutkimuksen menetelmää varioiden. Tulosten perusteella voi sanoa, että vaikka luokittimet osoittivat kykyä ennustaa raporttien sulkeutumisaikoja, on koneoppimistehtävä vaikeampi kuin yleisempi kehitystehtävän sulkeutumisajan ennustaminen, sillä eksploratiivisessa osassa muodostetut luokittimet toimivat replikaatiossa muodostettuja heikommin. Tulos oli yllättävä, sillä tutkimusta suunnitellessa oli ajatuksena, että opetus- ja testausdatan rajaaminen vain virheraportteihin vähentäisi datan heterogeenisyyttä, jolloin luokittimet voisivat tehdä tarkempia ennustuksia.

Käytetyn menetelmän heikko soveltuvuus virheraporttien sulkeutumisaikojen ennustamiseen ilmeni tuloksissa myös siten, että monet valituista muuttujista eivät olleet tärkeitä luokituksen kannalta yhdessäkään luokittimessa (Taulukot 8 ja 9). Aiemman tutkimuksen perusteella tiedetään, että virheiden korjausajan ennustaminen vaatii useita relevantteja muuttujia (Bhattacharya ja Neamtiu 2011). Herääkin epäily siitä, että kiinnittämällä huomiota muuttujien valintaan, voisi käytettyä menetelmää kehittää paremmin ohjelmistovirheiden ominaisuuksia huomioivaksi.

Ohjelmistovirheiden korjausaikoja on pyritty ennustamaan koneoppimismenetelmin useassa aiemmassa tutkimuksessa, ja tämän tutkimuksen tuloksia voi vertailla esimerkiksi luokittimien AUC-arvojen perustella. Vertailu ei kuitenkaan ole täysin yksiselitteinen, sillä luokitustehtävät ja luokkien lukumäärät voivat erota tutkimusten välillä, ja testidatan valinnassa on voitu tehdä poikkeavia päätöksiä. Tässä tutkimuksessa opetus- ja testidata kerättiin mahdollisimman monesta ohjelmistoprojektista, millä tavoiteltiin mallin mahdollisimman suurta yleistyvyyttä. Monissa aiemmissa tutkimuksissa mallit opetettiin ja testattiin käyttäen aineistoa vain yhdestä tai muutamasta projektista. Tässä tutkimuksessa testidata valittiin jakamalla data ajallisesti kahteen osaan. Valinnan etuna on se, ettei opetusdataan vuoda tietoa “tulevaisuudesta” ja testauksen tulokset vastaavat paremmin tilannetta käytännön sovelluksessa.

Yksittäisiin projekteihin kohdennettuja luokittimia muodostaneet Giger, Pinzger ja Gall (2010) käyttivät hyvin erilaista muuttujajoukkoa luokittimien opettamiseen, ja heidän luokittimiensa AUC-arvot olivat paremmat kuin tässä tutkimuksessa. Heidän tutkimustehtävänsä ei ollut täysin vastaava, mutta verrattaessa tutkimusten välillä luokittimia, joiden ennustettavat sulkeutumisajat ovat lähimpänä toisiaan, on tämän tutkimuksen luokitinten F1-arvo parempi johtuen korkeammasta tarkkuudesta. Marks, Zou ja Hassan (2011) käyttivät tämän tutkimuksen tapaan Random Forest -luokittimia, joilla he ennustivat Mozilla- ja Eclipse-projektien virheiden korjausaikoja. Ennustettavat ajat olivat kolme kuukautta, vuosi ja kolme vuotta, ja luokittimien täsmällisyys oli keskimäärin 65%. Eksploratiivisessa osassa muodostettujen luokittimien täsmällisyys oli keskimäärin 59% ja parhaimmillaan 71%, ja ennustettavia luokkia oli enemmän (Taulukko 6). Kolmen kuukauden sisällä korjattavia virheitä ennustettaessa Marksin, Zoun ja Hassanin luokittimet tekivät vähemmän virheitä, mutta ennustettavan ajan ollessa vuosi, oli tässä tutkimuksessa muodostetun luokittimen täsmällisyys parempi.

Edellä mainittuja tutkimuksia keskimäärin heikommat tulokset voivat johtua siitä, että kyseisissä tutkimuksissa luokittimet opetettiin ja testattiin käyttäen vain yhden ohelmistoprojektin dataa ja muuttujia, jotka sopivat hyvin yhteen projektiin kohdistuvaan malliin. Tällaisia muuttujia ovat esimerkiksi virheraportin kirjausaika ja virheen sisältävä sovelluksen versionumero, eikä niitä voi hyödyntää kaikille projekteille tarkoitetussa mallissa. Tässä tutkimuksessa käytetyssä datassa saatiin virheraporteista melko vähän strukturoitua dataa, eikä niiden perusteella ollut mahdollista seurata virheenkorjausprosessin etenemistä yhtä tarkasti kuin GitHubin issues-työkaulua jäsennellymmissä virheenseurannan järjestelmistä. Haasteista huolimatta tutkimuksen tulokset olivat vertailukelpoisella tasolla, etenkin kun ottaa huomioon yleistymisen mukanaan tuoman laajemman sovellettavuuden.

Menetelmää voisi kehittää ottamalla mukaan muuttujia, joiden avulla projektien hallinnoinnin, kommunikoinnin ja vertaisarvioinnin keinot erottuvat paremmin. Tällaisten muuttujien myötä olisi mahdollista saada tietoa osallistujien rooleista, virheraporttien käsittelykäytänteistä ja ehkä myös syistä nopean tai hitaan korjausajan takana. Sovellettavuuden kannalta voisi olla hyödyllistä muodostaa luokittelumallien sijaan regressiomalleja, joiden muodostaminen on mahdollista myös Random Forest -tekniikalla. Regressiomallien muodostamisen heikkoutena on se, että niiden opetus- ja testausdatassa ei voitaisi hyödyntää avoimeksi jää-

neitä virheraportteja, sillä niiden sulkeutumisaikaa ei tiedetä. Luokittelussa niitä pystyttiin hyödyntämään sen osalta, että tiedettiin mihin luokkaan ne eivät ainakaan kuulu.

Tutkimuksessa kävi ilmi, että virheraporttien sulkemisajan ennustaminen oli käytetyllä menetelmällä vaikeampaa kuin yleinen kehitystehtävän sulkemisajan ennustaminen. Jatkossa voisikin olla mielenkiintoista selvittää, millaisten tehtävien sulkeutumisajat sopivat hyvin koneoppimismenetelmin ennustettavaksi. Tutkimuksessa käytettiin paljon aikaa tiedonlouhintaan valtavasta datajoukosta. Mining Software Repositories -tutkimuskenttää eittämättä hyödyttäisi valmiiksi suodatettu ja ajantasainen datalähde projektien virheraporteille. Se vähentäisi ongelmia tilan ja prosessointitehon kanssa sekä mahdollistaisi tulosten tasapuolisen vertailun sitä käyttäneiden tutkimusten välillä.

Lähteet

Aberdour, Mark. 2007. “Achieving Quality in Open-Source Software”. *IEEE Software* 24 (1): 58–64. doi:10.1109/MS.2007.2.

Airio, Eija. 2009. “Morphological Problems in IR and CLIR. Applying linguistic methods and approximate string matching tools”.

Akila, V, G Zayaraz ja V Govindasamy. 2014. “Bug triage in open source systems: a review”. *International Journal of Collaborative Enterprise* 4 (4): 299–319.

Alfayez, Reem, Pooyan Behnamghader, Kamonphop Srisopha ja Barry Boehm. 2017. “How does contributors involvement influence open source systems”. Teoksessa *2017 IEEE 28th Annual Software Technology Conference (STC),* 1–8. doi:10.1109/STC.2017.8234462.

Alpaydin, Ethem. 2014. *Introduction to machine learning, Third edition.* The MIT Press.

Anh, Nguyen Duc, Daniela Soares Cruzes, Reidar Conradi ja Claudia P. Ayala. 2011. “Empirical Validation of Human Factors in Predicting Issue Lead Time in Open Source Projects”. Teoksessa *Proceedings of the 7th International Conference on Predictive Models in Software Engineering.* Association for Computing Machinery. doi:10.1145/2020390.2020403.

Ardimento, Pasquale, Massimo Bilancia ja Stefano Monopoli. 2016. “Predicting Bug-Fix Time: Using Standard Versus Topic-Based Text Categorization Techniques”, 167–182. doi:10.1007/978-3-319-46307-0_11.

Ardimento, Pasquale, ja Andrea Dinapoli. 2017. “Knowledge Extraction from On-Line Open Source Bug Tracking Systems to Predict Bug-Fixing Time”. Teoksessa *Proceedings of the 7th International Conference on Web Intelligence, Mining and Semantics.* WIMS ’17. Association for Computing Machinery. doi:10.1145/3102254.3102275.

Assar, Saïd, Markus Borg ja Dietmar Pfahl. 2016. “Using text clustering to predict defect resolution time: a conceptual replication and an evaluation of prediction accuracy”. *Empirical Software Engineering* 21 (4): 1437–1475.

Bhattacharya, Pamela, ja Iulian Neamtiu. 2011. “Bug-Fix Time Prediction Models: Can We Do Better?”, 207–210. MSR ’11. Association for Computing Machinery. doi:10.1145/1985441.1985472.

Breiman, Leo. 1996. “Bagging predictors”. *Machine learning* 24 (2): 123–140.

———. 2001. “Random forests”. *Machine learning* 45 (1): 5–32.

“Bugzilla Documentation”. 2021. Viitattu 15. heinäkuuta 2021. https://bugzilla.readthedocs.io/en/5.0/index.html.

Clifton, Christopher. 2019. “Data mining”. Viitattu 1. maaliskuuta 2021. https://www.britannica.com/technology/data-mining.

Conlon, Michael P. 2007. “An examination of initiation, organization, participation, leadership, and control of successful Open Source software development projects”. *Information Systems Education Journal* 5 (38): 1–13.

Corbly, James. 2014. “The Free Software Alternative: Freeware, Open Source Software, and Libraries”. *Information Technology and Libraries* 33. doi:10.6017/ital.v33i3.5105.

Crowston, Kevin, ja James Howison. 2005. “The social structure of free and open source software development”. *First Monday* 10 (2). doi:10.5210/fm.v10i2.1207.

Crowston, Kevin, Qing Li, Kangning Wei, U. Yeliz Eseryel ja James Howison. 2007. “Self-organization of teams for free/libre open source software development”. Qualitative Software Engineering Research, *Information and Software Technology* 49 (6): 564–575. ISSN: 0950-5849. doi:https://doi.org/10.1016/j.infsof.2007.02.004.

de Laat, Paul. 2007. “Governance of open source software: State of the Art”. *Journal of Management and Governance* 11:165–177. doi:10.1007/s10997-007-9022-9.

De Moura, Edleno Silva, ja Marco Antonio Cristo. 2017. “Inverted Files”. Teoksessa *Encyclopedia of Database Systems,* toimittanut Ling Liu ja M. Tamer Özsu, 1–5. Springer New York. doi:10.1007/978-1-4899-7993-3_1136-2.

Eseryel, U., Kangning Wie ja Kevin Crowston. 2020. “Decision-making Processes in Community-based Free/Libre Open Source Software-development Teams with Internal Governance: An Extension to Decision-making Theory”. *Communications of the Association for Information Systems:* 484–510. doi:10.17705/1CAIS.04620.

Fayyad, Usama. 1997. “Knowledge discovery in databases: An overview”. Teoksessa *Inductive Logic Programming,* toimittanut Nada Lavrač ja Sašo Džeroski, 1–16. Springer Berlin Heidelberg.

Fayyad, Usama, Gregory Piatetsky-Shapiro ja Padhraic Smyth. 1996. “From Data Mining to Knowledge Discovery in Databases”. *AI Magazine* 17 (3): 37. doi:10.1609/aimag.v17i3.1230.

Ghahramani, Zoubin. 2004. “Unsupervised Learning”. Teoksessa *Advanced Lectures on Machine Learning: ML Summer Schools 2003,* toimittanut Olivier Bousquet, Ulrike von Luxburg ja Gunnar Rätsch, 72–112. Springer Berlin Heidelberg. ISBN: 978-3-540-28650-9. doi:10.1007/978-3-540-28650-9_5.

Giger, Emanuel, Martin Pinzger ja Harald Gall. 2010. “Predicting the Fix Time of Bugs”. Teoksessa *Proceedings of the 2nd International Workshop on Recommendation Systems for Software Engineering,* 52–56. RSSE ’10. Association for Computing Machinery. ISBN: 9781605589749. doi:10.1145/1808920.1808933.

Gómez, Omar S, Natalia Juristo ja Sira Vegas. 2014. “Understanding replication of experiments in software engineering: A classification”. *Information and Software Technology* 56 (8): 1033–1048.

Gousios, Georgios. 2013. “The GHTorrent dataset and tool suite”. Teoksessa *Proceedings of the 10th Working Conference on Mining Software Repositories,* 233–236. MSR ’13. IEEE Press. ISBN: 978-1-4673-2936-1. http://dl.acm.org/citation.cfm?id=2487085.2487132.

Guo, Philip J., Thomas Zimmermann, Nachiappan Nagappan ja Brendan Murphy. 2010. “Characterizing and Predicting Which Bugs Get Fixed: An Empirical Study of Microsoft Windows”. Teoksessa *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1,* 495–504. ICSE ’10. Association for Computing Machinery. ISBN: 9781605587196. doi:10.1145/1806799.1806871.

Güemes-Peña, Diego, Carlos Nozal, Raúl Sánchez ja Jesús Maudes. 2018. “Emerging topics in mining software repositories: Machine learning in software repositories and datasets”. *Progress in Artificial Intelligence* 7. doi:10.1007/s13748-018-0147-7.

Han, Jiawei, Jian Pei ja Micheline Kamber. 2011. *Data Mining: Concepts and Techniques.* The Morgan Kaufmann Series in Data Management Systems. Morgan Kaufmann. ISBN: 9780123814791.

Heckman, Robert, Kevin Crowston, U. Eseryel, James Howison, Eileen Allen ja Qing Li. 2007. “Emergent Decision-Making Practices in Free/Libre Open Source Software (Floss) Development Teams”. Teoksessa *IFIP International Conference on Open Source Systems,* 234:71–84. ISBN: 978-0-387-72485-0. doi:10.1007/978-0-387-72486-7_6.

Ho, Tin Kam. 1998. “The random subspace method for constructing decision forests”. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 20 (8): 832–844. doi:10.1109/34.709601.

Johnson, Patricia. 2021. “Open Source Licenses in 2021: Trends and Predictions”. Viitattu 19. kesäkuuta 2021. https://www.whitesourcesoftware.com/resources/blog/open-source-licenses-trends-and-predictions/.

Karim, Md. Rejaul. 2019. “Key Features Recommendation to Improve Bug Reporting”. Teoksessa *2019 IEEE/ACM International Conference on Software and System Processes (ICSSP),* 1–4. doi:10.1109/ICSSP.2019.00010.

Kikas, Riivo, Marlon Dumas ja Dietmar Pfahl. 2016. “Using Dynamic and Contextual Features to Predict Issue Lifetime in GitHub Projects”. Teoksessa *Proceedings of the 13th International Conference on Mining Software Repositories,* 291–302. MSR ’16. Association for Computing Machinery. ISBN: 9781450341868. doi:10.1145/2901739.2901751.

Kumar, Dharminder, ja Deepak Bhardwaj. 2011. “Rise of data mining: current and future application areas”. *International Journal of Computer Science Issues (IJCSI)* 8 (5): 256.

Lakhani, Karim, ja Robert Wolf. 2005. “Why Hackers Do What They Do: Understanding Motivation and Effort in Free/Open Source Software Projects”. Teoksessa *Perspectives on free and open source software,* toimittanut Joseph Feller, Karim R Lakhani, Scott A Hissam ja Brian Fitzgerald. MIT Press.

Lamkanfi, A., ja S. Demeyer. 2012. “Filtering Bug Reports for Fix-Time Analysis”. Teoksessa *2012 16th European Conference on Software Maintenance and Reengineering,* 379–384. doi:10.1109/CSMR.2012.47.

Laurent, Andrew M St. 2004. *Understanding open source and free software licensing: guide to navigating licensing issues in existing & new software.* "O’Reilly Media, Inc."

Laurent, Hyafil, ja Ronald L Rivest. 1976. “Constructing optimal binary decision trees is NP-complete”. *Information processing letters* 5 (1): 15–17.

Lee, Gwendolyn, ja Robert Cole. 2001. “The Linux Kernel Development As A Model of Open Source Knowledge Creation”.

Lee, Y., S. Lee, C. Lee, I. Yeom ja H. Woo. 2020. “Continual Prediction of Bug-Fix Time Using Deep Learning-Based Activity Stream Embedding”. *IEEE Access* 8:10503–10515. ISSN: 2169-3536. doi:10.1109/ACCESS.2020.2965627.

Manning, Christopher D, Hinrich Schütze ja Prabhakar Raghavan. 2008. *Introduction to information retrieval.* Nide 39. Cambridge University Press Cambridge.

Marks, Lionel, Ying Zou ja Ahmed E. Hassan. 2011. “Studying the Fix-Time for Bugs in Large Open Source Projects”. Teoksessa *Proceedings of the 7th International Conference on Predictive Models in Software Engineering.* Association for Computing Machinery. ISBN: 9781450307093. doi:10.1145/2020390.2020401.

Markus, M. 2007. “The governance of free/open source software projects: Monolithic, multidimensional, or configurational?” *Journal of Management and Governance* 11:151–163. doi:10.1007/s10997-007-9021-x.

Mitchell, Tom M. 1997. *Machine learning*. McGraw-Hill series in computer science. McGraw-Hill.

Mockus, Audris, Roy T. Fielding ja James D. Herbsleb. 2002. "Two Case Studies of Open Source Software Development: Apache and Mozilla". *ACM Trans. Softw. Eng. Methodol.* 11 (3): 309–346. ISSN: 1049-331X. doi:10.1145/567793.567795.

Moral, Cristian, Angélica de Antonio, Ricardo Imbert ja Jaime Ramírez. 2014. "A survey of stemming algorithms in information retrieval." *Information Research: An International Electronic Journal* 19 (1).

Murthy, Sreerama K. 1998. "Automatic construction of decision trees from data: A multi-disciplinary survey". *Data mining and knowledge discovery* 2 (4): 345–389.

Müller, Ralf. 2009. *Project governance.* Fundamentals of project management. Gower.

Nilsson, Nils J. 1998. *Introduction to machine learning.*

Nothman, Joel, Hanmin Qin ja Roman Yurchak. 2018. "Stop word lists in free open-source software packages". Teoksessa *Proceedings of Workshop for NLP Open Source Software (NLP-OSS),* 7–12.

Nyman, Linus, ja Juho Lindman. 2013. "Code forking, governance, and sustainability in open source software". *Technology Innovation Management Review* 3 (1).

O'Mahony, Siobhan. 2007. "The governance of open source initiatives: what does it mean to be community managed?" *Journal of Management & Governance* 11:139–150.

O'Mahony, Siobhan, ja Fabrizio Ferraro. 2007. "The Emergence of Governance in an Open Source Community". *Academy of Management Journal* 50:1079–1106. doi:10.5465/AMJ.2007.27169153.

Open Source Initiative. 2007. "The Open Source Definition". Viitattu 5. kesäkuuta 2021. https://opensource.org/osd.

Paik, Jiaul H. 2013. “A Novel TF-IDF Weighting Scheme for Effective Ranking”. Teoksessa *Proceedings of the 36th International ACM SIGIR Conference on Research and Development in Information Retrieval,* 343–352. SIGIR ’13. Association for Computing Machinery. ISBN: 9781450320344. doi:10.1145/2484028.2484070.

Pfahl, Dietmar, Siim Karus ja Myroslava Stavnycha. 2016. “Improving Expert Prediction of Issue Resolution Time”. Teoksessa *Proceedings of the 20th International Conference on Evaluation and Assessment in Software Engineering.* EASE ’16. Association for Computing Machinery. ISBN: 9781450336918. doi:10.1145/2915970.2916004.

Pombo, N., ja R. Teixeira. 2020. “Contribution of Temporal Sequence Activities To Predict Bug Fixing Time”. Teoksessa *2020 IEEE 14th International Conference on Application of Information and Communication Technologies (AICT).* doi:10.1109/AICT50176.2020.9368603.

Popovič, Mirko, ja Peter Willett. 1992. “The effectiveness of stemming for natural-language access to Slovene textual data”. *Journal of the American Society for Information Science* 43 (5): 384–390.

Porru, Simone, Alessandro Murgia, Serge Demeyer, Michele Marchesi ja Roberto Tonelli. 2016. “Estimating Story Points from Issue Reports”. Teoksessa *Proceedings of the The 12th International Conference on Predictive Models and Data Analytics in Software Engineering.* PROMISE 2016. Association for Computing Machinery. ISBN: 9781450347723. doi:10.1145/2972958.2972959.

Porter, Martin F. 1980. “An algorithm for suffix stripping”. *Program.*

Raja, Uzma. 2012. “All complaints are not created equal: Text analysis of open source software defect reports”. *Empirical Software Engineering* 18. doi:10.1007/s10664-012-9197-9.

Ramarao, P., K. Muthukumaran, S. Dash ja N. L. Bhanu Murthy. 2016. “Impact of Bug Reporter’s Reputation on Bug-Fix Times”. Teoksessa *2016 International Conference on Information Systems Engineering (ICISE),* 57–61. doi:10.1109/ICISE.2016.18.

Raymond, Eric. 1999. “The cathedral and the bazaar”. *Knowledge, Technology & Policy* 12 (3): 23–49.

Riehle, Dirk. 2015. “The Five Stages of Open Source Volunteering”. Teoksessa *Crowdsourcing: Cloud-Based Software Development,* toimittanut Wei Li, Michael N. Huhns, Wei-Tek Tsai ja Wenjun Wu, 25–38. Springer Berlin Heidelberg. ISBN: 978-3-662-47011-4. doi:10.1007/978-3-662-47011-4_2.

Rigby, Peter, Brendan Cleary, Frederic Painchaud, Margaret-Anne Storey ja Daniel German. 2012. “Contemporary Peer Review in Action: Lessons from Open Source Development”. *IEEE Software* 29 (6): 56–61. doi:10.1109/MS.2012.24.

Rigby, Peter, Daniel M. German, Laura Cowen ja Margaret-Anne Storey. 2014. “Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory”. *ACM Trans. Softw. Eng. Methodol.* 23 (4). ISSN: 1049-331X. doi:10.1145/2594458.

Rigby, Peter, Daniel German ja Margaret-Anne Storey. 2008. “Open source software peer review practices”. Teoksessa *2008 ACM/IEEE 30th International Conference on Software Engineering,* 541–550. doi:10.1145/1368088.1368162.

Robles, Gregorio. 2010. “Replicating MSR: A study of the potential replicability of papers published in the Mining Software Repositories proceedings”, 171–180. doi:10.1109/MSR.2010.5463348.

Rokach, Lior, ja Oded Z. Maimon. 2008. *Data Mining With Decision Trees: Theory And Applications.* Series in Machine Perception and Artificial Intelligence. World Scientific. ISBN: 9789812771711.

Salton, G., A. Wong ja C. S. Yang. 1975. “A Vector Space Model for Automatic Indexing”. *Commun. ACM* 18 (11): 613–620. ISSN: 0001-0782. doi:10.1145/361219.361220.

Samset, Knut, ja Gro Holst Volden. 2016. “Front-end definition of projects: Ten paradoxes and some reflections regarding project management and project governance”. *International Journal of Project Management* 34 (2): 297–313. ISSN: 0263-7863. doi:https://doi.org/10.1016/j.ijproman.2015.01.014.

Schweik, Charles M., ja Robert English. 2007. “Tragedy of the FOSS commons? Investigating the institutional designs of free/libre and open source software projects”. *First Monday* 12 (2). doi:10.5210/fm.v12i2.1619.

Scott, Ezequiel, ja Dietmar Pfahl. 2018. “Using Developers’ Features to Estimate Story Points”. Teoksessa *Proceedings of the 2018 International Conference on Software and System Process,* 106–110. ICSSP ’18. Association for Computing Machinery. ISBN: 9781450364591. doi:10.1145/3202710.3203160.

Setia, Pankaj, Balaji Rajagopalan, Vallabh Sambamurthy ja Roger Calantone. 2012. “How peripheral developers contribute to open-source software development”. *Information Systems Research* 23 (1): 144–163.

Sharma, Meera, Madhu Kumari ja V. Singh. 2019. “Multi-attribute dependent bug severity and fix time prediction modeling”. *International Journal of System Assurance Engineering and Management* 10. doi:10.1007/s13198-019-00888-5.

Sharma, Srinarayan, Vijayan Sugumaran ja Balaji Rajagopalan. 2002. “A framework for creating hybrid-open source software communities”. *Information Systems Journal* 12 (1): 7–25. doi:10.1046/j.1365-2575.2002.00116.x.

Shull, Forrest, Jeffrey C. Carver, Sira Vegas ja Natalia Juristo Juzgado. 2008. “The role of replications in Empirical Software Engineering”. *Empirical Software Engineering* 13 (2): 211–218.

Strobl, Carolin, Anne-Laure Boulesteix, Thomas Kneib, Thomas Augustin ja Achim Zeileis. 2008. “Conditional variable importance for random forests”. *BMC bioinformatics* 9 (1): 1–11.

Strobl, Carolin, Anne-Laure Boulesteix, Achim Zeileis ja Torsten Hothorn. 2007. “Bias in random forest variable importance measures: Illustrations, sources and a solution”. *BMC bioinformatics* 8 (1): 1–21.

The Apache Software Foundation. 2021. “What is the Apache Software Foundation?” Viitattu 19. kesäkuuta 2021. https://www.apache.org/foundation/how-it-works.html.

Trinkenreich, Bianca, Mariam Guizani, Igor Wiese, Anita Sarma ja Igor Steinmacher. 2020. “Hidden Figures: Roles and Pathways of Successful OSS Contributors”. *Proc. ACM Hum.-Comput. Interact.* 4. doi:10.1145/3415251.

Wang, Jing, Patrick C. Shih ja John M. Carroll. 2015. “Revisiting Linus’s law: Benefits and challenges of open source software peer review”. *International Journal of Human-Computer Studies* 77:52–65. ISSN: 1071-5819. doi:10.1016/j.ijhcs.2015.01.005.

Wang, Jing, Patrick Shih, Yu Wu ja John Carroll. 2015. “Comparative case studies of open source software peer review practices”. *Information and Software Technology* 67:1–12. doi:10.1016/j.infsof.2015.06.002.

Viseur, Robert. 2012. “Forks impacts and motivations in free and open source projects”. *International Journal of Advanced Computer Science and Applications - IJACSA* 3. doi:10.14569/IJACSA.2012.030221.

Witten, I. H., ja E. Frank. 2005. *Data mining practical machine learning tools and techniques.* Morgan Kaufmann series in data management systems. Morgan Kaufman.

Voorhees, Ellen. 2000. “Natural Language Processing and Information Retrieval”. ISBN: 978-3-540-66625-7. doi:10.1007/3-540-48089-7_3.

Yuan, Lin, Huaimin Wang, Gang Yin, Dianxi Shi ja Haibo Mi. 2010. “Mining roles of open source software”. Teoksessa *The 2nd International Conference on Software Engineering and Data Mining,* 548–554. IEEE.

Zhang, Hongyu, Liang Gong ja Steve Versteeg. 2013. “Predicting Bug-Fixing Time: An Empirical Study of Commercial Software Projects”. Teoksessa *Proceedings of the 2013 International Conference on Software Engineering,* 1042–1051. ICSE ’13. IEEE Press. ISBN: 9781467330763. doi:10.5555/2486788.2486931.

Zhou, Yu, Yanxiang Tong, Ruihang Gu ja Harald Gall. 2014. “Combining Text Mining and Data Mining for Bug Report Classification”. Teoksessa *2014 IEEE International Conference on Software Maintenance and Evolution,* 311–320. doi:10.1109/ICSME.2014.53.

Al-Zubaidi, Wisam Haitham Abbood, Hoa Khanh Dam, Aditya Ghose ja Xiaodong Li. 2017. “Multi-Objective Search-Based Approach to Estimate Issue Resolution Time”. Teoksessa *Proceedings of the 13th International Conference on Predictive Models and Data Analytics in Software Engineering,* 53–62. Association for Computing Machinery. doi:10.1145/3127005.3127011.