Tampereen yliopisto

Henri Seppä

AVOIMEN LÄHDEKOODIN OHJELMISTOKEHITYS JA SEN LAADUNVARMISTUS

Kandidaatintutkielma
Informaatioteknologian ja viestinnän tiedekunta
Toukokuu 2023

TIIVISTELMÄ

Henri Seppä: Avoimen lähdekoodin ohjelmistokehitys ja sen laadunvarmistus
Kandidaatintyö
Tampereen yliopisto
Informaatioteknologian ja viestinnän tiedekunta
Toukokuu 2023

Avoimen lähdekoodin ohjelmistoja käytetään nykyään monissa eri tilanteissa, minkä takia niiden pitää olla laadukkaasti tuotettuja ja toimintavarmoja. Jotta ohjelmistosta saataisiin mahdollisimman laadukas tuote, joka täyttää sille asetetut laatustandardit, vaatii se laadunvarmistuksen tekemistä. Tässä kandidaatintyössä käydään läpi eri tapoja, joilla avoimen lähdekoodin ohjelmistoprojekteissa tehdään laadunvarmistusta.

Tutkielma toteutettiin kirjallisuuskatsauksena. Lähteiksi valittiin aihetta käsitteleviä vertaisarvioituja artikkeleita sekä konferenssijulkaisuja. Näiden lähteiden pohjalta pystyttiin muodostamaan hyvä kuva siitä, miten laadunvarmistusta voidaan tehdä avoimen lähdekoodin projekteissa.

Tutkielma on jaettu kolmeen eri osaan. Ensimmäisessä osassa käydään läpi avoimen lähdekoodin ohjelmistokehitystä, mistä osista se koostuu ja minkälaisia eroja siinä on ohjelmistoihin, joiden lähdekoodi ei ole julkisesti saatavilla. Tämän lisäksi tuodaan esille avoimen lähdekoodin ohjelmistoihin ja niiden kehitykseen liittyviä hyötyjä sekä haittoja. Tämän jälkeen toisessa osassa käydään läpi mistä asioista ohjelmiston laatu muodostuu ja mitä eri laatumalleja ohjelmistoille on olemassa. Tutkielman kolmannessa osassa tutkitaan, miten ohjelmiston laatu pystytään varmistamaan laadunvarmistusta tekemällä avoimenlähdekoodin ohjelmistoprojekteissa.

Tehdyn kirjallisuuskatsauksen perusteella avoimen lähdekoodin ohjelmistokehityksessä koodilisäysten vertaisarvioinnilla on tärkeä merkitys tuotetun ohjelmiston laadun kannalta. Vertaisarviointi on helppo ja kustannustehokas tapa toteuttaa laadunvarmistusta avoimen lähdekoodin ohjelmistoprojektissa, kun ohjelmiston lähdekoodi on kaikkien projektiin osallistuvien henkilöiden saatavilla versionhallinnan kautta ja projektiin osallistuvien henkilöiden osaamistaso on laaja. Avoimen lähdekoodin laadunvarmistuksessa pystytään myös hyödyntämään laatumittareita sekä -malleja, joiden avulla lähdekoodin laatua voidaan arvioida esimerkiksi tarkastelemalla sen rakennetta sekä tutkimalla ohjelmiston tuottanutta kehittäjäyhteisöä. Tärkeässä osassa ohjelmiston laadun kannalta on myös ohjelmiston dokumentaatio sekä lähdekoodin modulaarisuus, joka vaikuttaa sen ylläpidettävyyteen.

Avainsanat: Avoimen lähdekoodin ohjelmistokehitys, laadunvarmistus, ohjelmiston laatu

SISÄLLYSLUETTELO

.

1. JOHDANTO

Avoimen lähdekoodin ohjelmistot ovat kasvattaneet suosiotaan runsaasti viime vuosina. Siinä missä ohjelmistot kehitettiin ennen suljettuina ohjelmistoina, joiden lähdekoodiin ei päässyt käsiksi, niin nykyään monien ohjelmistojen lähdekoodit ovat avoimia ja niiden kehittämiseen myös ohjelmiston käyttäjät pystyvät osallistumaan. Näissäkin avoimen lähdekoodin ohjelmistokehitysprojekteissa on tärkeää tuotetun ohjelmiston laadun varmistaminen, jotta lopullinen tuote toimii odotetunlaisesti ja täyttää vaaditut laatustandardit.

Oma kiinnostukseni aiheeseen heräsi siitä, että olen useamman vuoden ajan käyttänyt erilaisia avoimeen lähdekoodiin perustuvia ohjelmistoja, minkä lisäksi olen tehnyt työtä testauksen parissa. Laadukkaasti toimiva ohjelmisto on tärkeää muun muassa hyvän käyttäjäkokemuksen ja turvallisen käytön kannalta. Siitä syystä myös avoimen lähdekoodin laadunvarmistukseen pitää käyttää resursseja niitä kehitettäessä.

Tässä työssä selvitetään, miten avoimen lähdekoodin ohjelmistokehityksessä varmistetaan tuotetun ohjelmiston laatu. Työ toteutettiin kirjallisuuskatsauksena, jonka eteneminen on kuvattuna työn toisessa luvussa. Tämän jälkeen kolmannessa luvussa käydään läpi avoimen lähdekoodin ohjelmistokehityksen pääpiirteitä kuten kehittäjäyhteisöä sekä avoimen lähdekoodin hyötyjä ja haittoja.

Työn neljännessä luvussa avataan ohjelmiston laadun määritystä ja millaisista osista se koostuu. Tämän lisäksi esitellään standardin mukaiset laatumallit, joita käytetään laadun määrittelyssä.

Viidennessä kappaleessa esitellään eri tavat, joilla avoimen lähdekoodin ohjelmistojen laatuun voidaan vaikuttaa. Esille tuodaan eri tavat kuten vertaisarviointi ja ohjelmiston dokumentointi, joiden avulla laatua voidaan varmistaa. Tämän lisäksi esitellään eri laatumalleja, jotka helpottavat avoimen lähdekoodin ohjelmistojen laadun määritystä. Työn lopussa tuodaan vielä esille huomatut havainnot avoimen lähdekoodin ohjelmistojen laadunvarmistuksesta ja siitä, mitä näistä voidaan päätellä.

2. TUTKIMUSMENETELMÄ

Tutkielma suoritettiin tutkivana kirjallisuuskatsauksena. Tutkielmaa varten etsittiin lähteitä pääasiassa yliopiston Andor-järjestelmästä sekä IEEE-tietokannasta. Lähteiden avulla pyrittiin selvittämään tutkielmaa varten, mitä avoimen lähdekoodin ohjelmistokehitys on, mitä ohjelmiston laatu voidaan määritellä sekä miten laadunvarmistus näkyy avoimen lähdekoodin ohjelmistojen kehittämisessä.

Hakuja tehtiin tietokannoista seuraavilla hakusanoilla: ”Open source software”, ”Quality assurance”, ”QA”, ”Open Source software development”, ”software testing” ja ”software quality”. Näitä hakusanoja yhdisteltiin tarvittaessa ”AND”- ja ”OR”-operaattoreiden avulla, jotta tietokannasta löytyisi mahdollisimman hyviä lähteitä työtä varten. Tämän lisäksi hakuasetuksista määriteltiin, että lähteiden pitää olla vuonna 2012 tai sen jälkeen julkaistuja. Tämän lisäksi tarkennettiin, millaisia aineistotyyppejä haluttiin tuloksissa näkyvän eli tämän tutkielman tapauksessa kirjoja, vertaisarvioituja artikkeleita, standardeja sekä konferenssijulkaisuja. Näiden hakusanojen sekä asetusten avulla onnistuttiin hyvin löytämään erilaisia lähteitä tutkielman kirjoittamista varten.

Suurin osa avoimeen lähdekoodiin ohjelmistokehitykseen liittyvistä lähteistä, joita tutkielmassa käytettiin, oli tutkimuksia tai konferenssijulkaisuja. Ohjelmistojen laatuun liittyvien lähteiden joukosta löytyi myös kirjoja, jotka käsittelivät aihetta. Aiheesta laatu löydettiin myös erilaisia standardeja.

Lähteiden valinnassa suosittiin pääasiassa vertaisarvioituja artikkeleita, mutta myös kirjoja sekä standardeja niiltä osin, kun ne nähtiin tarpeelliseksi. Lähteiden valintoja työtä varten tehtiin lähteiden otsikoiden perusteella sekä lukemalla niiden tiivistelmä. Tämän jälkeen jäljelle jääneet lähteet luettiin vielä tarkemmin läpi, ja jos ne todettiin työhön sopiviksi, tehtiin niistä vielä lyhyet muistiinpanot oleellisimmasta sisällöstä auttamaan kirjoitustyötä. Valittujen lähteiden suhteen pyrittiin myös siihen, että ne olisivat mahdollisimman uusia julkaisuja. Myös muutamia ennen vuotta 2010 julkaistuja lähteitä päädyttiin kuitenkin myös valitsemaan tutkielmaa varten, koska lähteitä tarkistaessa huomattiin, että niihin oli viitattu muissa eri tutkimuksissa.

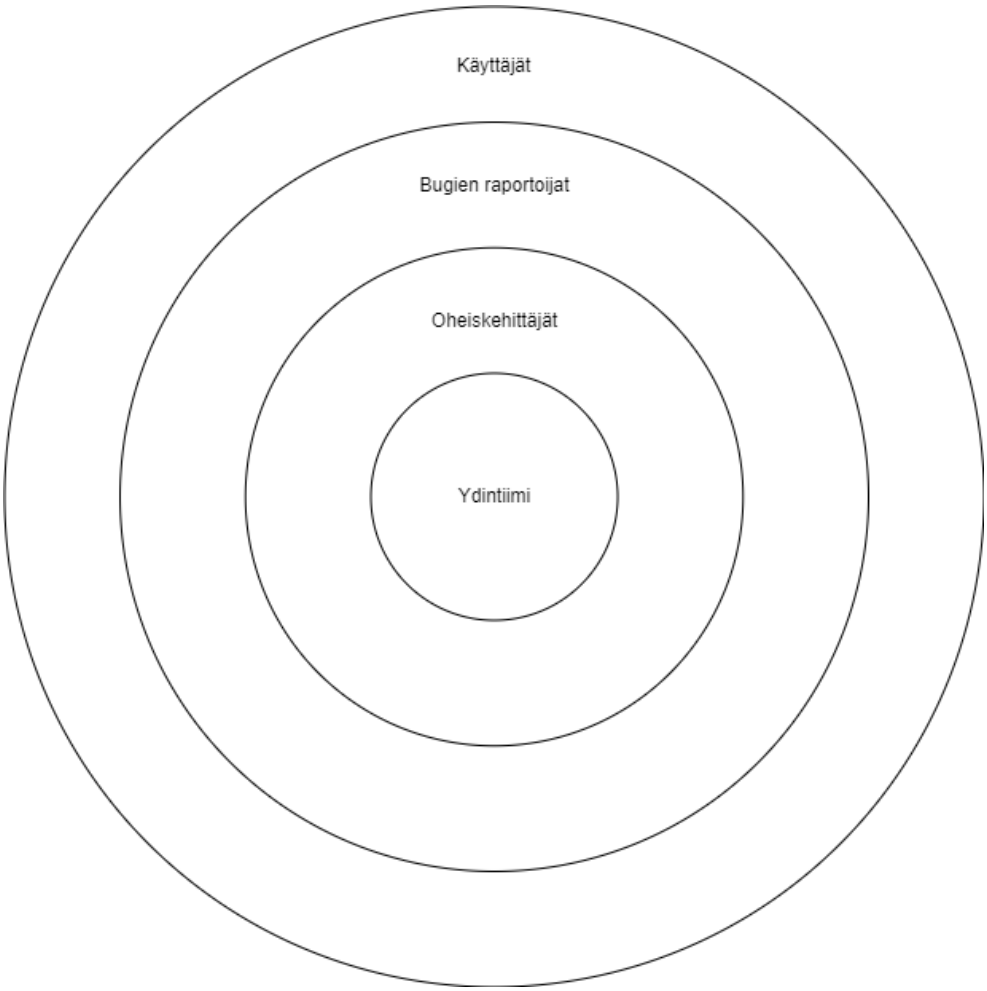
3. AVOIMEN LÄHDEKOODIN OHJELMISTOKEHITYS

Avoimen lähdekoodin ohjelmistokehityksen periaatteena on, että kuka tahansa voi ladata ohjelmiston lähdekoodin itselleen, tehdä siihen muokkauksia ja tämän jälkeen liittää nämä muutokset osaksi ohjelman lähdekoodia. Tämän takia avoimen lähdekoodin ohjelmistokehitys eroaa merkittävästi perinteisen suljetun lähdekoodin ohjelmiston kehittämisessä, jossa lähdekoodi ei ole julkisesti kaikkien saatavilla.

Erona suljettuihin ohjelmistoihin avoimen lähdekoodin ohjelmistoissa on myös se, että kehityksessä on mukana paljon eritasoisia ohjelmoijia erilaisin taustoin. Suljettujen ohjelmistojen ohjelmoijat ovat sen sijaan organisoituneet tietyn organisaation alle, eikä sen ulkopuolisilla henkilöillä, kuten ohjelman käyttäjillä, ole mahdollisuutta vaikuttaa ohjelman tekemiseen suoraan tai päästä näkemään, miten ohjelma on toteutettu. [14] Avoimuuden etuna voidaan tässä tapauksessa nähdä se, että käyttäjät pystyvät itse vaikuttamaan ohjelman ominaisuuksiin, ja koska kuka tahansa voi osallistua ohjelman kehitykseen, pystytään käyttäjien erityistarpeet ottamaan paremmin huomioon.

3.1 Yhteisö

Avoimen lähdekoodin ohjelmistokehityksessä kehitystyöstä vastaa kehittäjäyhteisö, jossa kuka tahansa yhteisön jäsen voi vapaasti osallistua ohjelmiston kehittämiseen. Yhteisö koostuu yleensä useammasta erilaisesta tasosta riippuen siitä, kuinka aktiivisesti yhteisön jäsen osallistuu ohjelmiston kehittämiseen. Tätä mallia kutsutaan sipulimalliksi. [6] Sipulimalli ei kuitenkaan ole täydellinen kuvaus koko kehittäjäyhteisön rakenteesta, sillä siihen kuuluu yleensä myös muita sidosryhmiä, kuten esimerkiksi teollisia kumppaneita [7].



Kuva 1: Avoimen lähdekoodin kehittäjäyhteisön sipulimalli, mukaillen lähteestä [1]

Sipulimallissa on kolme pääpiirrettä. Ensimmäinen näistä pääpiirteistä on se, että ydinkehittäjien luoman tiimin pitää olla pieni. Heidän vastuullaan on huolehtia ohjelmiston korkeasta modulaarisuudesta sekä hoitaa suurin osa ohjelmiston ohjelmoinnista. He huolehtivat myös siitä, että ohjelmaan lisätään ainoastaan korkealaatuista koodia ja että projektissa säilyy nopeatahtinen iteratiivinen julkaisusykli. [1] Kooltaan nämä ydintiimit ovat alkuun noin 3–7 henkeä ja kasvavat 10–15 hengen kokoisiksi siinä vaiheessa, kun ohjelmiston kehitysyhteisö on vakiintunut [7].

Toisena pääpiirteenä sipulimallissa on se, että avustavat kehittäjät lisäävät ja ylläpitävät ominaisuuksia avoimen lähdekoodin projektissa. He myös tekevät korjauksia järjestelmässä esiintyviin virheisiin, suorittavat vertaisarviointeja ohjelman koodiin ja auttavat

ydintiimiä työmäärän tasaamisessa. Viimeisenä pääpiirteenä on se, että virheiden raportoijat ovat vastuussa järjestelmän testaamisesta ja virheiden raportoinnista. Koska tämän ryhmän koko sipulimallissa on erittäin suuri, mahdollistuu huomattavasti parempi ohjelmiston testaus, kuin kaupallisissa ohjelmistoissa. [1]

Jotta avoimen lähdekoodin kehittäjäyhteisö säilyisi vakaana, pitää sen houkutella mukaansa vapaaehtoisia ohjelmoijia, jotka haluavat osallistua projektin kehitykseen. Motivaattoreina projekteissa toimivat muun muassa projektin dokumentaation taso, opasohjelmat sekä laadukkaat kehitystyökalut. Myös mahdollisuus päästä oppimaan uusia teknologioita sekä itsensä haastaminen lisäävät vapaaehtoisten motivaatiota lähteä mukaan avoimiin projekteihin. [1] Panostamalla näiden asioiden toteutumiseen, pystytään avoimen lähdekoodin kehittäjäyhteisöön saamaan mukaan uusia kehittäjiä.

3.2 Git

Git on hajautettu versionhallintajärjestelmä, joka toimii avoimen lähdekoodin ohjelmistokehityksen selkärankana. Versionhallintajärjestelmän tarkoituksena on pitää kirjaa ohjelmiston lähdekoodin koko historiasta, jotta useat kehittäjät voivat osallistua lähdekoodin kehittämiseen ja tehdä siihen muutoksia. Git toimii siis tavallaan automaattisena rajoittajana avoimessa ohjelmistoprojektissa mahdollistaen useiden ihmisten samanaikaisen osallistumisen projektiin. [13]

GitHub on yksi suosituimmista Gittiin perustuvista hajautetuista versionhallintajärjestelmistä, jota hyödynnetään avoimen lähdekoodin ohjelmistokehityksen hallinnassa. Palvelussa käyttäjät pystyvät jakamaan itse kehittämäänsä koodia muille palvelun käyttäjille, minkä lisäksi käyttäjät voivat tallentaa avoimen lähdekoodin projektien lähdekoodit itselleen ja tehdä siihen muokkauksia. Projekteilla on tämän lisäksi GitHubissa aina pääkehittäjät, jotka pystyvät hyväksymään tai hylkäämään muiden tekemiä muutoksia heidän projekteihinsa. [5] GitHub tarjoaa myös erilaisia yhteistyötä helpottavia ominaisuuksia avoimen lähdekoodin kehittämiseen, kuten rajattoman määrän tietovarastoja, projektinhallinta työkaluja sekä laajat keskusteluominaisuudet [3]. Tämän takia GitHub soveltuukin siis hyvin avoimen lähdekoodin ohjelmistoprojektien kehittämiseen.

GitHub on samalla myös sosiaalinen verkosto sekä yhteisökanava, minkä takia sen käyttäjäaktiivisuuteen voidaan nähdä vaikuttavan epäsuora palkitsemismekanismi [9]. Palkitsemismekanismina kehittäjälle voi toimia muun muassa se, kuinka paljon kehittäjä saa muutoksia läpi avoimen lähdekoodin projektiin. GitHubin kaltaisessa sosiaalisessa ver-

kostossa aktiivisimpia ovat yleensä sellaiset henkilöt, joilla on paljon seuraajia. Aktiivisuus ei kuitenkaan ole suoraan verrannollinen seuraajien määrään, sillä yleensä henkilöillä, joilla on paljon eri tapahtumia GitHubissa, on seuraajia vähemmän. [9]

GitHubin avulla pystytään saamaan hyvin dataa avoimen lähdekoodin projekteihin liittyen. Muun muassa seuraamalla projektin talletusten määrää voidaan päätellä, kuinka aktiivisesti yhteisö ja sen eri jäsenet osallistuvat projektin kehittämiseen. Mitä korkeampi talletusten määrä on, sitä suositumpana projektia voidaan pitää. [5] GitHubissa on myös mahdollista antaa projekteille tähtiä, joista voi päätellä, kuinka suosittu jokin tietty projekti on [14]. Tämän tiedon perusteella varsinkin uudet kehittäjät pystyvät helpommin löytämään itselleen sopivia projekteja.

3.3 Hyödyt ja haitat

Avoimen lähdekoodin ohjelmistoissa ja niiden kehityksessä on paljon erilaisia hyötyjä ja haittoja eri tahoille. Käyttäjien kannalta suuria etuja avoimen lähdekoodin ohjelmistoissa ovat muun muassa lähdekoodin saatavuus sekä tiedon jakaminen. Yksityisyydestään huolestunut käyttäjä voi halutessaan päästä näkemään käyttämänsä ohjelmiston lähdekoodin ja tarkastaa, kuinka tietoturvallisesti kyseinen ohjelmisto on tehty. Tämän voidaan nähdä nostavan käyttäjän luottamusta kyseistä ohjelmistotuotetta kohtaan. Käyttäjän kannalta avoimen lähdekoodin ohjelmiston haittana saattaa kuitenkin olla myös avoimen ohjelmiston huono dokumentaatio. [6] Tällöin käyttäjän on vaikeaa tutustua ohjelman käyttöön sekä toimintaan, joka saattaa laskea luottamusta kyseistä ohjelmistoa kohtaan.

Kehittäjälle sekä itse avoimen lähdekoodin ohjelmistolle on paljon erilaisia hyötyjä ohjelmiston avoimuudesta. Kehittäjä pystyy avointa lähdekoodia kehittäessään tekemään usein omia ratkaisuja toteutuksen suhteen, mikä lisää motivaatiota ohjelman kehittämiseen. Ohjelmistolle tulevia hyötyjä avoimuudesta ovat muun muassa se, että mahdolliset bugit löydetään helpommin ja ohjelma toimii täten varmemmin, koska ylläpitäjiä on runsaasti. [6] Kun ohjelmaa toimii varmemmin ja on laadukkaampi, kokee myös ohjelman käyttäjä sen käyttämisen miellyttävänä.

Ohjelmiston kannalta haitallista on se, että avoimen lähdekoodin projekteissa ei yleensä ole käytössä mitään virallista kehitysprosessia ja ohjelmiston rakenne saattaa olla huonosti suunniteltu [6]. Tämä kasvattaa kehittäjän tekemän työn määrää ja saattaa myös laskea motivaatiota ohjelman kehittämiseen. Kehittäjän ongelmana avoimen lähdekoodin kehityksessä on myös kunnollisten työkalujen puute sekä yhteistyö uusien kehittäjien kanssa. Uudet kehittäjät saattavat vaatia paljon perehdytystä kyseiseen ohjelmistoon

ennen kuin he voivat ruveta kehittämään sitä kunnolla, mikä on pois kokeneemman kehittäjän työajasta. Myös jos ohjelmiston kehittämiseen ei ole saatavilla kunnollisia työkaluja, saattaa ohjelmiston kehittäminen tuntua epämotivoivalta.

4. OHJELMISTOJEN LAATU

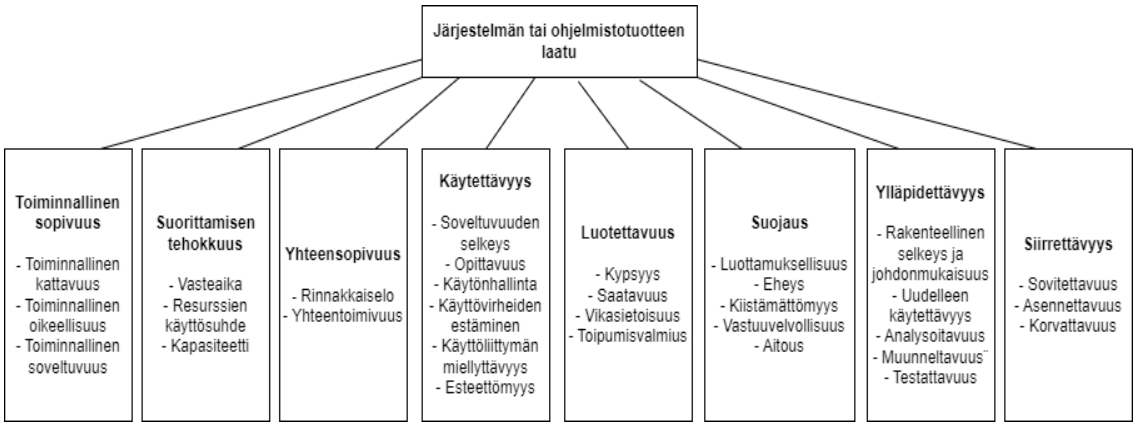
Ohjelmiston laadulla tarkoitetaan sitä, että tuotettu ohjelmisto täyttää sille asetetut tavoitteet ja että se toimii niiden mukaan. Laadun määrittely riippuu usein myös siitä, kenen näkökulmasta ohjelmiston laatua tarkastellaan. Esimerkiksi ohjelmiston kehittäjällä on usein erilainen näkemys siitä, mitä ohjelmiston laatu on, kuin esimerkiksi ohjelmiston käyttäjillä. [8]

Ohjelmiston käyttäjille tärkeitä asioita laadussa ovat muun muassa tehokkuus, luotettavuus ja käytettävyys. Ohjelmistokehityksen ammattilaisille tärkeämpää laadun suhteen on taas saavuttaa ohjelmistolle määritellyt vaatimukset ja tavoitteet. [8] Nämä vaatimukset pystytään saavuttamaan tekemällä laadunvarmistusta ohjelmistoa kehittäessä.

Ohjelmiston laatuvaatimukset muuttuvat ajan myötä ja jos näihin vaatimuksiin ei puututa, heikkenee ohjelmiston laatu. Tämän takia ohjelmistoa kehittäessä on tehtävä jatkuvaa laaduntarkkailua sekä hyödynnettävä eri malleja ja metriikoita vaadittavan laadun saavuttamiseksi. Avoimen lähdekoodin kehityksessä tämä tapahtuu helposti, sillä siinä ohjelmiston julkaisutahti on tiheä ja kehittämiseen osallistuu useita kehittäjiä, jotka pystyvät korjaamaan ohjelmistosta löytyviä ongelmia nopealla tahdilla. [14]

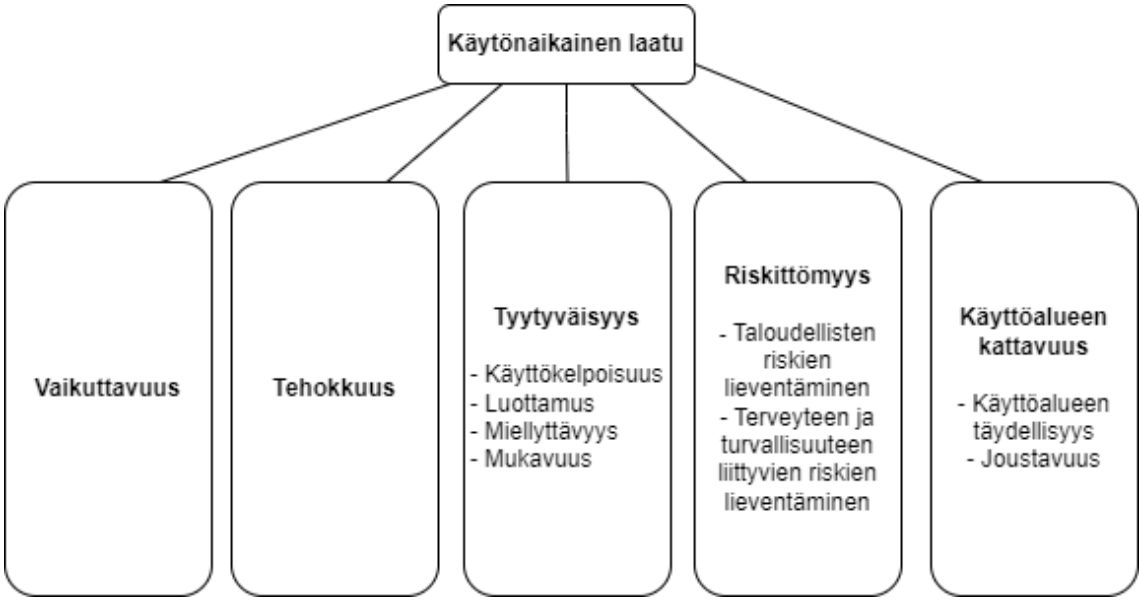
Ohjelmiston laatua voidaan määrittää standardien avulla. Standardissa ISO/IEC 25010:2011 määritetään, millaisia laatumalleja ohjelmistoilla on olemassa. Tässä standardissa nämä laatumallit on jaettu kahteen eri kategoriaan, käytönaikaisen laadun malliin ja ohjelmistotuotteen laatumalliin. [12] Näiden laatumallien avulla voidaan määrittää, että millaisiin asioihin ohjelmiston laatu koostuu ja millaisiin asioihin laadunvarmistajien on keskityttävä laadunvarmistusta tehdessään.

Kuvassa 2 on kuvattuna tuotteen laatumalli ja sen eri ominaisuudet. Näitä ominaisuuksia on yhteensä 8, joista jokainen koostuu toisiinsa liittyvistä laatupiirteistä. Näitä laatupiirteitä voidaan hyödyntää sekä itse ohjelmistoon että ohjelmiston sisältävään järjestelmään, sillä suurin osa laatupiirteistä on oleellisia kummankin kannalta.



Kuva 2: Tuotteen laatumalli, mukaillen lähteestä [12]

Käyttäjän ja ohjelmiston välinen vuorovaikutussuhde laadun kannalta voidaan määrittää käytönaikaisen laadun mallilla (kuva 3). Tällä laatumallilla kuvataan vaikutusta, joka ohjelmistolla on tuotteen eri sidosryhmiin. Tämä vaikutus määrittyy muun muassa ohjelmiston, käyttöjärjestelmän tai laitteen laadun sekä käyttäjien, tehtävien ja sosiaalisen ympäristön erilaisten ominaisuuksien perusteella. [12]



Kuva 3: Käytönaikaisen laadun malli, mukaillen lähteestä [12]

5. LAADUN VARMISTAMINEN AVOIMEN LÄHDEKOODIN OHJELMISTOKEHITYKSESSÄ

Laadunvarmistusta voidaan tehdä avoimen lähdekoodin projekteissa monilla eri tavoilla. Tässä kappaleessa käsitellään eri tapoja, joilla laadunvarmistusta tehdään avoimen lähdekoodin ohjelmistoprojekteissa.

5.1 Vertaisarviointi

Vertaisarviointi on yleinen tapa, jolla laadunvarmistusta voidaan tehdä avoimen lähdekoodin ohjelmistokehityksessä. Vertaisarviointiin liittyy erilaisia parametreja, joiden avulla voidaan määrittää sen laadukkuus. Näitä parametreja ovat muun muassa arvioijien ammattitaito, arviointien toteutuksen aikaväli ja arvioinnin tehokkuus. [10]

Vertaisarviointi alkaa siitä, kun ohjelmistoon ehdotetaan tehtävän jonkinlainen muutos. Tämä muutoksen tekijä voi olla joko kokenut ohjelmistokehittäjä tai vasta aloitteleva ohjelmoija. Kun muutos on tehty, lähetetään se arvioitavaksi muulle kehittäjäyhteisölle, joka arvioinnin jälkeen joko jättää sen käsittelemättä tai ottaa sen tarkastettavaksi. Jos muutos päätetään ottaa tarkastettavaksi, lähetetään siitä palautetta muutoksen tehneelle henkilölle sekä tiedotetaan yhteisölle. Lopuksi vielä muutoksen tekijä käy keskusteluja muutoksesta arvioijien ja mahdollisten muiden sidosryhmien kanssa, ennen kuin tehdään lopullinen päätös siitä, hyväksytäänkö muutos ohjelmistoon vai ei. [10]

Vertaisarvioinnin tekemiseen ohjelmistoissa on erilaisia tapoja. Näitä tapoja ovat muun muassa RTC (Review Then Commit) ja CTR (Commit Then Review) -mallit. Näistä malleista RTC on jaettu viiteen eri varianttiin, jotka ovat epävirallinen, vahva, ylläpito, laiska ja jäljitys perusteinen variantti. [10]

Epävirallinen RTC-malli ei sisällä virallista menettelytapaa arvioinnin tekemiselle, vaan ohjelmistoon tehdyistä muutoksista lähetetään tieto projektin sähköpostilistalle, jossa näistä muutoksista käydään sitten keskustelua. Täytenä vastakohtana tälle mallille toimii vahvan RTC:n malli, jossa jokainen ohjelmistoon tehty muutos arvioidaan ennen kuin se tulee osaksi ohjelman lähdekoodia. Ylläpidollisessa RTC-mallissa projektin koodin kehittäjien pitää saada ylläpitäjiltä vahvistus tekemilleen muutoksille. Muutoksia ohjelmistoon voidaan hyväksyä myös niin, että muutoksesta ilmoitetaan sähköpostilistalle ja jos tietyssä ajassa kukaan ei vastaa viestiin, muutos hyväksytään osaksi lähdekoodia. Tällöin on kyse laiskasta RTC-mallista. Vertaisarvioinnin apuna voidaan hyödyntää myös seurantatyökalua, kuten Bugzillaa, jolloin kyseessä on seurantapohjainen RTC-malli. [10]

CTR projekteissa koodin tarkistus tapahtuu vasta lisäyksen teon jälkeen, minkä takia CTR-mallilla toimivissa projekteissa on yleensä mukana kokeneempia kehittäjiä kuin RTC-mallia hyödyntävissä projekteissa. Näissä malleissa koodin tarkastajien määrä saattaa vaihdella 16 henkilöstä aina 480 henkilöön. Suurin osa näistä tarkastajista tekee ainoastaan muutamia kooditarkastuksia kuukaudessa ja ainoastaan projektin kärkikehittäjiin kuuluvat henkilöt osallistuvat useisiin kooditarkastuksiin. [12] Tässäkin tapauksessa toteutuu avoimen lähdekoodin kehittäjäyhteisön sipulimalli, jossa yhteisön kärkikehittäjät osallistuvat laajemmin projektin kehitykseen.

CTR- ja RTC-mallin tavalla toteutetussa vertaisarvioinnissa ei ole merkittävää eroa keskenään. RTC:tä käytettäessä tehtyjen muutosten määrä on noin 11–32 kappaletta yhdessä talletuksessa, kun taas CTR:llä näitä muutoksia on yleensä 12–35 kappaletta. Löydettyjen virheiden mediaani on myös molemmilla tavoilla suhteellisen lähellä toisiaan. RTC:llä virheitä löytyy yleensä yhdestä kahteen kappaletta ja CTR:llä noin yksi kappale. Nopeuden suhteen näillä metodeilla kuitenkin on eroa toisiinsa. RTC:llä tarkastusajan mediaani on 23–46 tuntia, kun taas CTR:llä tarkastus tapahtuu keskimäärin 5,6–19,4 tunnin aikana. [10] Tämän voidaan nähdä johtuvan siitä, että koska muutokset on jo tehty ohjelmistoon CTR-mallia käytettäessä, on ne tarkastettava nopeammin, jotta mahdolliset virheet eivät aiheuttaisi yhtä suurta haittaa. CTR-mallia käyttävissä projekteissa on kuitenkin hieman vähemmän virheitä kuin RTC-mallia käyttävissä projekteissa [10]. Tämän johtuu siitä, että CTR-mallin projekteissa kehitystyötä tekevät henkilöt ovat yleensä kokeneempia ohjelmistokehittäjiä.

Avoimen lähdekoodin ohjelmistojen vertaisarvioinnissa on tärkeää, että arvioinnin tekeminen on säännöllistä ja jatkuvaa. Tällöin mahdolliset viat pystytään huomaamaan aikaisessa vaiheessa ja tarvittaessa korjaamaan. Jos vikoja ei korjata ajoissa, tulee niiden korjaamisesta kalliimpaa, kun ne ehtivät sulautumaan osaksi ohjelmistoa. Myös tehtyjen muutosten pieni koko hyödyttää ohjelman vertaisarviointia ja sen pysymistä säännöllisenä. [10]

Vertaisarvioinneissa on myös tärkeää, että tehdyistä muutoksista tiedotetaan mahdollisimman suurelle joukolle ohjelmiston kehittäjiä. Itse arvioinnin suorittaa kuitenkin lopulta pienempi joukko henkilöitä, joilla on enemmän kokemusta ohjelman kehittämisessä. Yleensä arvioinnit pystytään hoitamaan kahden arvioijan toimesta, mutta isompien muutosten kohdalla tarvitaan useampia arvioijia, jotta mahdolliset ongelmat varmasti löytyvät. [10]

5.2 Dokumentaatio

Avoimen lähdekoodin ohjelmistokehityksessä dokumentaatio keskittyy erityisesti tyyliin sekä koodiohjeisiin. Hyvin tehty dokumentaatio auttaa uusia kehittäjiä pääsemään paremmin mukaan projektiin, kun he pystyvät dokumentaatiosta selvittämään millä tavalla ohjelmiston on tarkoitus toimia. Dokumentaation avulla voidaan myös viestiä kehittäjille siitä, että minkä tyylistä koodia projektiin voi kirjoittaa ja kenellä on oikeus tehdä muokkauksia projektin versionhallintaan. [6]

Jos dokumentaatio on kuitenkin tehty huonosti, saattaa se aiheuttaa ongelmia avoimen ohjelmiston kannalta. Uusien kehittäjien on tällöin hankalaa tutustua ohjelmiston toimintatapaan, jolloin he saattavat epähuomiossa lisätä huonolaatuista koodia ohjelmistoon. Tähän tilanteeseen voi johtaa se, että ohjelmiston dokumentaation tekemistä ei aina koeta tarpeeksi motivoivana, jolloin sen tuottaminen laiminlyödään.

5.3 Ohjelmiston modulaarisuus ja ylläpidettävyys

Ohjelmiston modulaarisuus ja ylläpidettävyys ovat tekijöitä, jolla on vaikutusta avoimen lähdekoodin ohjelmistojen laatuun. Ylläpidettävyys koostuu neljästä eri osasta, joita ovat ohjelmiston analysoitavuus, muutettavuus, vakaus ja testattavuus. Analysoitavuus perustuu siihen, kuinka helposti ohjelmasta pystytään tunnistamaan erilaiset virheet sekä ongelmat. Muutettavuuden avulla pystytään arvioimaan, kuinka hyvin ohjelmistoa voi muokata ja vakaudella määritetään, kuinka odottamattomilta vaikutuksilta voidaan välttyä, kun ohjelmiston koodia muokataan. Testattavuudella taas tarkoitetaan ohjelmiston testaus potentiaalia sekä kykyä vahvistaa ohjelmistoon tehdyt muutokset. [4]

Toisin kuin suljetun lähdekoodin ohjelmistokehityksessä, pystytään avoimen lähdekoodin ohjelmistoprojektissa lisäämään kehittäjien määrää ilman, että projektin tuottavuus kärsii. Tämä johtuu erilaisesta lähestymistavasta modulaarisuuteen, jota avoimen lähdekoodin projekteissa hyödynnetään. Kun ohjelmisto on jaettu tarpeeksi pieniin osamoduuleihin, pystytään niihin tekemään muutoksia ilman, että sillä on vaikutusta muuhun ohjelmistoon. Näin useampi henkilö pystyy helposti työskentelemään ohjelmiston parissa ilman riskiä siitä, että jokin menisi rikki, kun ohjelmistoon tehdään uusia muutoksia. Hyvänä esimerkkinä onnistuneista avoimen lähdekoodin projekteista voidaan käyttää Firefoxia tai Apachea, jotka molemmat on onnistuttu skaalaamaan modulaarisuuden avulla isoiksi projekteiksi. [11]

Ruiz et. Al. artikkelissa tuodaan kuitenkin myös esille se, että modulaarisuuden hyödyistä ohjelmiston laadun suhteen ei kuitenkaan ole olemassa täydellistä konsensusta. Tämä johtuu siitä, että ohjelmiston modulaarisuudelle on olemassa monia eri tulkintatapoja.

Tämän takia osassa tutkimuksista on päädytty siihen lopputulokseen, että modulaarisuudella ei ole merkittävää vaikutusta ohjelmiston laatuun, kun taas joidenkin tutkimusten mukaan modulaarisuuden ja laadun välillä on nähtävissä selkeä korrelaatio. [11]

5.4 Laatumittarit

Avoimen lähdekoodin ohjelmiston laadun mittaamiseen voidaan hyödyntää erilaisia laatumittareita, jotka ovat yleensä samoja, joita käytetään suljettujen ohjelmistojen laadun mittaamisessa. Mittareiden avulla pystytään arvioimaan ohjelmistoprojektin eri osa-alueita ja auttamaan kehittäjiä selvittämään, mitkä niistä eivät toimi odotetunlaisesti. Mittareiden hyvänä puolena on se, että niiden avulla voidaan arvioida sekä virheitä että dokumentaation laatua. Huonona puolena laatumittareissa on kuitenkin se, että hyväksyttävistä asioista ei ole olemassa täydellistä standardia. [14] Jokainen laadunvarmistusta tekevä henkilö saattaa tämän takia tulkita ohjelmiston laatua hieman eri tavoilla, riippuen mitä asioita hän painottaa laadussa.

Käytössä olevia mittareita on olemassa useita erilaisia. Yleisimmät käytössä olevat laatumittarit ovat kuvattuna taulukossa 1.

Taulukko 1: Ohjelmiston laatumittarit, mukailtuna lähteestä [14]

Koodirivien määrä	Mitä korkeampi koodirivien määrä ohjelmistossa on, sitä vaikeampi ohjelmistoa on ylläpitää.
Syklomaattinen kompleksisuus	Määrittää kuinka monimutkainen ohjelmisto on rakenteeltaan.
Huollettavuus indeksi	Määrittää ohjelman ylläpidettävyyden hyödyntämällä muun muassa koodirivien määrää sekä syklomaattista kompleksisuutta.
Periyttämisen syvyys	Luokkien määrä, jotka on periytetty kantaluokasta. Mitä suurempi tämä on arvoltaan, sitä vaikeampi ohjelmistoa on ylläpitää
Olioiden välinen kytkentä	Mittaa luokkien välistä kytkentää ja indikoi koodin uudelleenkäyttöä ja ylläpidettävyyttä ohjelmistossa.

Metodien yhtenäisyyden puute	Osoittaa, edustaako luokka yhtä vai useampaa abstraktiota. Korkeampi luku on parempi.
Kommentoinnin tiheys	Kommenttien määrä suhteutettuna lähdekoodiin. Vaikuttaa koodin luettavuuteen ja ylläpidettävyyteen.

5.5 Yhteisön merkitys laadunvarmistuksessa

Kun avoimen lähdekoodin ohjelmiston kehittäjäyhteisö on vakaa, pysyy tuotetun ohjelmiston laatu korkeana. Jokaisella yhteisön jäsenellä on oma vastuualueensa, jolloin jokainen yhteisössä mukana oleva kehittäjä pystyy keskittymään omiin vastuihinsa ja täten nostamaan kehitettävän ohjelmiston laadukkuutta. Varsinkin bugien raportoijat ovat tässä tärkeässä osassa, sillä he mahdollistavat sen, että ydintiimin ja avustavien kehittäjien ei tarvitsee käyttää aikaa ohjelmistovirheiden etsimiseen, vaan he voivat keskittyä raportoijien löytämien virheiden korjaamiseen [1].

Yhteisön koolla ei juurikaan ole väliä ohjelmiston laadun kannalta, kun kehitetään avoimen lähdekoodin ohjelmistoa. Enemmän vaikutusta ohjelmiston laatuun on sillä, miten projektissa ilmeneviä ongelmia ratkotaan ja minkälainen on yhteisön osallistumisprosentti ohjelman kehitykseen. Gitin käyttö helpottaa yhteisön osallistumisprosentin tarkastelua projektin kehityksessä, sillä sen välityksellä pystytään seuraamaan projektiin tehtyjen talletusten määrää. Projektin pääkehittäjät pystyvät myös näkemään tämän avulla, että missä ohjelmiston osissa on parannettavaa laadun kannalta. [14]

Avoimen lähdekoodin yhteisöissä on yleensä mukana paljon eri tason osaajia, joka parantaa avoimen lähdekoodin periaatteella kehitettävän ohjelmiston laatua [6]. Varsinkin kehittäjäyhteisön kokeneemmat henkilöt osaavat antaa tarkkaa palautetta ohjelmiston mahdollisista vioista, joka helpottaa niiden korjaamista. Koska avoimen lähdekoodin projekteissa lähdekoodi on kaikkien saatavilla, pystyvät kaikki yhteisön jäsenet käymään koodia lävitse, joka johtaa useampien bugien löytämiseen. Avoimen lähdekoodin projekteissa yhteisön jäsenet ovat myös motivoituneita parantamaan koodin laatua saadakseen kunnioitusta sekä mahdollisia muita etuja tulevaisuudessa [6]. Nämä edut voivat olla muun muassa rahallisia etuja tai mahdollisuus saada enemmän vastuuta projektin kehittämisessä.

5.6 Laatumallit

Avoimen lähdekoodin ohjelmistojen laadunvarmistuksen apuna on tarjolla erilaisia laatumalleja, joiden avulla voidaan arvioida ohjelmistotuotteen laatua. Yksi tällainen laatumalli on SQO-OSS, joka on hierarkkinen laatumalli. Sen avulla voidaan arvioida lähdekoodin ja yhteisöprosessien tuottamaa laatua avoimen lähdekoodin ohjelmistoprojektissa. Malli perustuu kahteen eri vaiheeseen, jotka ovat arviointimallin määrittely ja datan keräämisen määrittely. Tarkasteltavat asiat pystytään jakamaan myös pienempiin osiin. Lähdekoodia voidaan analysoida vaikka muutettavuuden ja vakauden kannalta minkä lisäksi yhteisöä voidaan tarkastella dokumentaation ja kehittäjäyhteisön ammattitaidon pohjalta. [14]

”OpenSource maturity model” eli OMM perustuu avoimen lähdekoodin ohjelmistokehityksen prosessista saatavaan tietoon ja on muodostunut tämän prosessin evoluution myötä. Tätä mallia hyödynnetään avoimen lähdekoodin ohjelmiston kokonaislaadun mittaamisessa. Mallin toimintaperiaatteena on se, että ohjelmiston laadun arviointi on suoritettava huolellisesti, sillä avoimen lähdekoodin projekteja, että niiden kehitystapoja, on olemassa monia erilaisia. OMM:n päätehtäviin kuuluu laatuvaatimusten selvittäminen sidosryhmiltä haastatteluiden ja kyselyiden avulla. Näiden vastausten pohjalta luodaan laatumalli avoimen lähdekoodin ohjelmistolle. [14]

QualOSS -mallin avulla voidaan määrittää, onko avoimen lähdekoodin ohjelmisto laadultaan sopivaa sille vaadittuun tarkoitukseen ja tuetaanko ohjelmistoa myös tulevaisuudessa. Mallin laatuominaisuudet perustuvat hierarkkiseen malliin ja ne on jaettu kahteen eri osaan, tuotteeseen liittyviin ominaisuuksiin ja yhteisöön liittyviin ominaisuuksiin. Tuotteeseen liittyviä ominaisuuksia ovat muun muassa tuotteen ylläpidettävyys, luotettavuus sekä käytettävyys. Yhteisön ominaisuuksia ovat taas ylläpidon kapasiteetti, vakaus sekä kehitysprosessin kypsyys. [14]

Laatumallit perustuvat siis yleisesti jonkinlaiseen hierarkkiseen malliin, jolla tarkastellaan lähdekoodin ja sitä kehittävän yhteisön ominaisuuksia. Tämä helpottaa kehitettävän tuotteen ylläpitämistä ja laadunvarmistamista.

6. YHTEENVETO

Tässä tutkielmassa perehdyttiin avoimen lähdekoodin ohjelmistokehitykseen ja siihen, miten tällä tavalla tuotetuissa ohjelmistoissa tehdään laadunvarmistusta ja mitkä asiat laatuun vaikuttavat. Aluksi esiteltiin avoimen lähdekoodin ohjelmistokehitystä ja siihen vaikuttavia asioita. Tämän jälkeen käytiin läpi, että mitä ohjelmiston laadulla tarkoitetaan ja millaisilla laatumalleilla voidaan kuvata ohjelmistojen laatua. Lopuksi nämä aiheet nivottiin yhteen ja tarkasteltiin, kuinka laadunvarmistusta toteutetaan avoimen lähdekoodin ohjelmistoprojekteissa ja mitkä asiat siihen vaikuttavat.

Tehdyn kirjallisuuskatsauksen perusteella voidaan todeta, että laadunvarmistuksella on olemassa tärkeä asema myös avoimen lähdekoodin ohjelmistokehityksen projekteissa. Avoimen lähdekoodin projekteissa laadunvarmistuksen kivijalkana toimii erityisesti vertaisarviointi. Vertaisarvioinnin toteuttaminen on varsin helppoa, koska avoimen lähdekoodin ohjelmistoprojekti koostuu yhteisöstä, josta löytyy paljon kehittäjiä, jotka osaavat arvioida tuotetun koodin laatua. Tämä on varsin kustannustehokas ja helppo tapa toteuttaa laadunvarmistusta projektissa, johon osallistuvat henkilöt ovat vapaaehtoisia ja tulevat erilaisista taustoista erilaisella osaamistasolla.

Ohjelmiston laatuun voidaan myös vaikuttaa koodin modulaarisuudella avoimen lähdekoodin ohjelmistoprojekteissa. Kun ohjelma on jaettu pienempiin osakokonaisuuksiin, pystytään siihen tekemään muutoksia siten, ettei niillä ole suurta vaikutusta ohjelman muihin osiin. Lähdekoodi on myös tällöin luettavampaa, joka helpottaa uusien kehittäjien mukaan tuloa. Myös erilaiset laatumittarit sekä -mallit tukevat laadunvarmistuksen tekemistä avoimen lähdekoodin projekteissa.

Tulevaisuudessa tämän aiheen tutkimusta voisi laajentaa tutkimalla, että millä tavalla automatisoitua testaamista voidaan hyödyntää avoimen lähdekoodin projekteissa laadun parantamiseen. Myös aiheen tutkiminen siitä näkökulmasta, että projektilla olisi nimetyt testaajat sen sijaan, että kehittäjät hoitavat testauksen, voisi tutkia tulevissa tutkimuksissa.

LÄHTEET

[1] Aberdour M. Achieving Quality in Open-Source Software. IEEE software. 2007;24(1): p. 58–64.

[2] Alami A., Dittrich Y., Wasowski A., Influencers of Quality Assurance in an Open Source Community. Proceedings of 11th International Workshop on Cooperative and Human Aspects of Software Engineering. 2018

[3] Github: Where the world builds software, GitHub, verkkosivu. Saatavissa (viitattu 20.05.2023): https://github.com/

[4] Hanandeh F., Saifan A., Akour M., Al-Hussein N., Shatnawi K.. Evaluating Maintainability of Open Source Software: A Case Study. International journal of open source software & processes. 2017;8(1):1–20.

[5] Joy A., Thangavelu S., Jyotishi A. Performance of GitHub Open-Source Software Project: An Empirical Analysis. 2018 Second International Conference on Advances in Electronics, Computer and Communications. 2018.

[6] Khanjani A, Sulaiman R. The Process of Quality Assurance under Open Source Software Development. 2011 IEEE Symposium on Computers & Informatics. IEEE; 2011. p. 548-552

[7] Kilamo T, Hammouda I, Mikkonen T, Aaltonen T. From proprietary to open source—Growing an open source ecosystem. The Journal of systems and software. 2012;85(7): p. 1467–1478.

[8] Laporte CY, April A. Software Quality Assurance. 1st ed. Piscataway: Wiley; 2017.

[9] Lima A., Rossi L., Musolesi M. Coding together at scale: GitHub as a collaborative social network. In: Proceedings of the 8th International Conference on Weblogs and Social Media, ICWSM 2014. 2014. p. 295–304.

[10] Rigby P, German DM, Cowen L, Storey M-A. Peer Review on Open-Source Software Projects: Parameters, Statistical Models, and Theory. ACM transactions on software engineering and methodology. 2014;23(4): p. 1–33.

[11] Ruiz C, Robinsson W. Towards a Unified Definition of Open Source Quality. In: IFIP Advances in Information and Communication Technology. Berlin, Heidelberg: Springer Berlin Heidelberg; 2011. P. 17-33

[12] SFS-ISO/IEC 25010:2019. Tietotekniikka. Järjestelmäkehitys ja ohjelmistotuotanto. Järjestelmien ja ohjelmistojen laatuvaatimukset ja niiden laadun arviointi (SQuaRE). Tietojärjestelmien ja ohjelmistojen laatumallit. Helsinki: Suomen Standardoimisliitto. 2019. 81 s.

[13] Song J., Kim C. What Is Needed for the Sustainable Success of OSS Projects: Efficency Analysis of Commit Production Process via Git. Sustainability. 2018.

[14] Tassone J, Xu S, Wang C, Chen J, Du W. Quality Assessment of Open Source Software: A Review. In: 2018 IEEE/ACIS 17th International Conference on Computer and Information Science (ICIS). IEEE; 2018. p. 411–416.