Mikael Mäntylä

KEHITTÄJÄKOKEMUKSEN TOTEUTUMINEN TEKNISESSÄ DOKUMENTAATIOSSA

JYVÄSKYLÄN YLIOPISTO
INFORMAATIOTEKNOLOGIAN TIEDEKUNTA
2023

TIIVISTELMÄ

Mäntylä, Mikael
Kehittäjäkokemuksen toteutuminen teknisessä dokumentaatiossa
Jyväskylä: Jyväskylän yliopisto, 2023, 71 s.
Tietojärjestelmätiede, pro gradu -tutkielma
Ohjaaja: Seppänen, Ville

Ohjelmistokehitys, siinä missä moni muukin ajatustyö, on ihmislähtöistä toimintaa. Vaikka ala on hyvin teknologiaorientoitunut, loppupeleissä työnteosta vastaavat aina ohjelmoijat, ihmiset. Tämän vuoksi ohjelmoijien työskentelytyytyväisyyteen eli kehittäjäkokemukseen on kiinnitettävä erityistä huomiota. Tässä tutkimuksessa tutkittiin toimeksiantajan yhteistyökumppanien suhtautumista sen tekniseen dokumentaatioon kehittäjäkokemuksen näkökulmasta. Kirjallisuuden avulla perehdyttiin lean-ajatteluun ohjelmistokehityksessä, kehittäjäkokemukseen sekä dokumentaatioon. Teorian avulla luotiin kyselylomake ja se lähetettiin toimeksiantajalta saaduille sähköpostiosoitteisiin yhdessä saatekirjeen kanssa. Tarkoituksena oli selvittää, miten kumppanit kokevat dokumentaation palvelevan heitä heidän jokapäiväisessä työssään ja, miten se voisi olla entistä parempi. Tutkimus toteutettiin siis määrällisenä kyselytutkimuksena, mutta avoimien kysymysten vuoksi siinä oli myös laadullisia osa-alueita. Kyselytutkimuksen otanta osoittautui odotettua pienemmäksi, minkä vuoksi tuloksista ei voida yleistää vaan niistä voidaan vetää ainoastaan suuntaa antavia johtopäätöksi toimeksiantajan dokumentaation tiimoilta. Yleisesti ottaen dokumentaatio osoittautui melko hyväksi kumppanien keskuudessa, mutta silti joitain kehitysehdotuksia löytyi. Etenkin dokumentaation sekavuus, keskeneräisyys ja koodiesimerkkien määrä saivat kritiikkiä. Dokumentaation rakenteen selkeyttäminen parantaisi sekavuutta, mikä puolestaan kohentaisi kehittäjäkokemusta. Vastaajat kaipasivat yleisesti lisää koodiesimerkkejä ja mielellään useimmilla eri ohjelmointikielillä. Eniten ääni keräsivät PHP, JavaScript, Python ja Curl.

Asiasanat: ohjelmistokehitys, kehittäjäkokemus, dokumentaatio, tekninen dokumentaatio, API-dokumentaatio, API-kehitys,

ABSTRACT

Mäntylä, Mikael
Developer experience in a technical documentation
Jyväskylä: University of Jyväskylä, 2023, 71 pp.
Information Systems, Master's Thesis
Supervisor(s): Seppänen, Ville

Software development, like many other forms of thinking-related work, is a people-oriented activity. Although the industry is very technology-oriented, at the end of the day, the work is always done by people. Therefore, particular attention must be paid to the job satisfaction of programmers, i.e., the developer experience. This study investigated the attitude of the client's technical partners. Literature was used to explore lean thinking in software development, developer experience and documentation. A questionnaire was created using the theory and sent to the email addresses provided by the client together with a cover letter. The aim was to find out how the partners perceive the documentation to help them in their daily work and how it could be improved. The survey was therefore conducted as a quantitative survey, but due to the open questions, it also had qualitative elements. The survey sample turned out to be smaller than expected, which means that the results cannot be generalised and can only be used to draw some indicative conclusions about the client's documentation. In general, the documentation proved to be quite good among the partners, but there were still some improvements to be done. In particular, the documentation was criticised for its incoherence, incompleteness, and the amount of code examples. Clarifying the structure of the documentation would improve the confusion, which would improve the developer experience. Respondents generally wanted more code examples, preferably in more programming languages. PHP, JavaScript, Python and Curl received the most amount votes.

Keywords: software development, developer experience, documentation, technical documentation, API-documentation, API-development

KUVIOT

TAULUKOT

SISÄLLYS

1 JOHDANTO

Ohjelmistokehitys, siinä missä moni muukin ajatustyö, on ihmislähtöistä toimintaa. Vaikka ala on hyvin teknologiaorientoitunut, loppupeleissä työnteosta vastaavat aina ohjelmoijat, ihmiset. Onnistunut ohjelmistoprojekti on monimutkaisten prosessien tulos, mutta suurissa määrin onnistumiseen vaikuttavat projektissa olevat ihmiset. Ohjelmistokehitystä voidaankin pitää sen monitahoisuuden vuoksi yhtenä vaikeimmista ihmisten suorittamista tehtävistä.

Lukuiset tutkimukset ovat osoittaneet inhimillisten tekijöiden olevan kaikista tärkeimpiä tekijöitä ohjelmistokehityksessä, kun asiaa tarkastellaan suorituskyvyn ja laadun näkökulmasta. (Mockus, 2010; DeMarco & Lister, 1985; Fagerholm & Münch, 2012) Tämän vuoksi ohjelmistoprojektien ohjelmoijien kokemuksiin ja inhimillisiin tekijöihin on tärkeä kiinnittää huomiota. Mikäli työntekijät eivät koe työtään, työympäristöä tai työkalujaan miellyttäväksi, työn lopputulos kärsii eikä laatu yllä halutulle tasolla.

Tässä tutkielmassa on tarkoitus perehtyä ohjelmoijien kokemuksiin ja peilata siihen positiivisesti ja negatiivisesti vaikuttavia tekijöitä etenkin teknisen dokumentaation näkökulmasta. Tutkielman toimeksiantaja, Paytrail, haluaa saada selville, kuinka hyvin heidän tarjoama tekninen dokumentaatio palvelee kehittäjäasiakkaitansa jokapäiväisessä työssään. Lisäksi on tarkoitus saada selville, miten teknistä dokumentaatiota voisi parantaa entistä paremman kehittäjäkokemuksen saamiseksi. Paytrail on suomalainen maksulaitos, joka välittää verkkomaksuja ja vastaa yli 20 000 organisaation verkkomaksuliikenteestä. Paytralin asiakkaisiin kuuluvat esimerkiksi Onnibus, Finnkino sekä Poliisin lupapalvelut.

Tutkielma jakautuu karkeasti kahteen osaan: teoreettiseen viitekehykseen sekä empiiriseen tutkimukseen. Teoria jakautuu kolmeen eri osioon: ohjelmistokehityksen lean-ajatteluun, kehittäjäkokemukseen sekä ohjelmistokehityksen tekniseen dokumentaatioon. Ensimmäisessä osassa on tarkoitus avata ohjelmistokehityksen perusperiaatetta ja tarkentaa tätä ketterän kehitysmenetelmän, leanin, suuntaan. Menetelmän ymmärtäminen on tärkeää, sillä tutkielman toimeksiantaja käyttää omassa kehitystyössään kyseiseistä

menetelmää. Näin ollen kaiken Paytralissa tapahtuvan ohjelmoinnin pohjana on lean-ajattelu.

Lean-ajattelun jälkeen teoriassa perehdytään kehittäjäkokemukseen. Tällä tarkoitetaan kokemusta, joka tapahtuu ohjelmistokehitysprosessissa ohjelmoijan sekä ohjelmointityökalujen välillä, kun kehittäjä työskentelee palveluiden tai tuotteiden kehityksen parissa. (Fagerholm & Münch, 2012.) Kehittäjäkokemusta tutkitaan perinteisen ohjelmistokehityksen lisäksi myös API-kehityksen näkökulmasta. Suurin osa ohjelmistokehityksestä käyttää viitekehyksiä ja kirjastoja, joiden toiminnallisuudet ovat saatavilla API:iden kautta (Robillard, 2009). Näin on myös toimeksiantajan tuotteen osalta. API-kehityksen kehittäjäkokemuksen jälkeen minigradussa perehdytään vielä tarkemmin ohjelmistokehityksen inhimillisiin tekijöihin.

Kolmannen teoriaosion tarkoitus on avata teknisen dokumentaation roolia ohjelmistokehityksessä, tarkastella tätä API-dokumentaatioon näkökulmasta sekä avata eri dokumentaatiotyyppejä.

Empiirinen osio rakentuu seuraavien tutkimuskysymysten ympärille:

- Miten verkkokauppojen tekniset toteuttajat kokevat kehittäjäkokemuksen toteutumisen toimeksiantajan teknisen dokumentaation avulla?
- Miten dokumentaatiota voisi parantaa, jotta kehittäjäkokemus olisi entistä parempi?

Tutkimuskysymyksiin pyritään vastaamaan olemassa olevan teorian sekä määrällisen kyselytutkimuksen avulla. Kysely lähetetään verkkokauppojen rakentajille eli niin sanotuille Paytralin teknisille kumppaneille. Rakentaessaan verkkokauppoja, kumppanit käyttävät Paytralin teknistä dokumentaatiota omassa työssään, minkä vuoksi palautteen saaminen suoraan heiltä on äärimmäisen tärkeää.

Empiirisen tutkimusosion jälkeen käydään läpi tutkimuksesta nousseet tulokset. Tämän jälkeen tuloksia pohditaan tutkimuskysymysten näkökulmasta ja arvioidaan tutkimuksen rajoituksia ja luotettavuutta. Koko tutkielma päättyy teorian ja tutkimuksen yhteenvetoon sekä mahdollisten jatkotutkimusaiheiden esittelyyn.

2 LEAN-AJATTELU OHJELMISTOKEHITYKSESSÄ

Tässä luvussa käsitellään ohjelmistokehitystä ja siihen liittyviä prosesseja. Tarkemmin luvussa keskitytään lean-ajatteluun ja sen siihen liittyviin seitsemään periaatteeseen. Lean-ajattelun ymmärtäminen on tärkeää, koska toimeksiantaja käyttää sitä omassa ohjelmointityöskentelyssään.

2.1 Lean-ajattelu

Ohjelmistokehitys on prosessi, jonka tarkoituksena on rakennuttaa organisaatiolle tietojärjestelmä tukemaan liiketoiminnallisia tarpeita. Valmiin järjestelmän lisäksi ohjelmistokehitysprosessi pitää sisällään järjestelmään liittyvän dokumentaation, tukisivustot, käyttäjämanuaalin sekä kaiken datan sen käyttöön liittyen. Projektin osa-alueita ovat järjestelmäkuvaus, kehitys, validointi, testaus, jakelu, koulutus, tuki sekä itse käyttöönoton jälkeinen ylläpitovaihe, mikä jatkuu pitkään ohjelmiston valmistumisen jälkeen. Keskeisiä asioita ohjelmistokehitysprojektissa ovat myös määritelty budjetti, tavoite sekä valmistumisen aikataulu. (Sommerville, 2016.; Schmidt, 2013.)

Suunnitelmakeskeisyys ja raskaat menetelmät ovat olleet isossa roolissa ohjelmistokehityksessä. Tähän vastapainona on noussut niin sanottu ketterä ohjelmistokehitys. Ketterät menetelmät pitävät arvossaan yksilöitä ja vuorovaikutusta, toimivaa ohjelmistoa ja asiakasyhteistyötä. Tämän lisäksi muutoksiin reagointi, dokumentointi, sopimukset ja suunnitelmat kuuluvat ketterien menetelmien piiriin. Ketterät menetelmät korostavat mahdollisimman lyhyttä kehityssykliä, säännöllisiä tuotoksia, jatkuvaa kasvokkain tapahtuvaa kommunikointia ja jatkuvaa oppimista. (Beck ym. 2001.) Suosittuja ketteriä kehitysmenetelmiä ovat muun muassa Scrum, Extreme Programming, Lean-ohjelmistokehitys ja Kanban (Kupiainen, Mäntylä & Itkonen, 2015**).** Tässä kappaleessa keskitytään tarkemmin Lean-menetelmään.

Lean-ajattelu on pohjimmiltaan ajattelutapa, jonka avulla organisaatiot voivat määrittää arvoa, järjestää arvoa luovia toimia parhaaseen mahdolliseen

järjestykseen sekä suorittaa näitä toimia keskeytyksettä ja entistä tehokkaammin (Womack, Jones & Roos, 1996). Ajattelu voidaan jakaa työn kolmeen kategoriaan: arvoa tuottaviin, vaadittuihin ei-arvoa lisääviin ja ei-arvoa lisääviin toimiin. Ajattelun tarkoituksena on paikantaa ja poistaa ne toimet, jotka eivät tuo prosesseihin lisää arvoa. (Kupiainen, Mäntylä & Itkonen, 2015**.)** Ohjelmistokehityksessä näitä voivat olla ylimääräiset ominaisuudet, odottaminen, tehtävien vaihtaminen, ylimääräiset prosessit, osittain tehty työ, viat ja työntekijän käyttämätön luovuus (Poppendieck & Poppendieck, 2003).

2.2 Lean-ohjelmistokehityksen periaatteet

Poppendieck ja Cusumano (2012) nostavat esiin seitsemän lean-ohjelmistokehityksen periaatetta:

1. Kokonaisuuden optimointi (optimize the whole)
2. Hukan poisto (eliminate waste)
3. Laadun rakentaminen (build quality in)
4. Jatkuva oppiminen (learn constantly)
5. Nopea toimittaminen (deliver fast)
6. Kaikkien mukaan ottaminen (engage everyone)
7. Jatkuva paraneminen (keep getting better)

Lean-ohjelmistokehityksen tulisi perustua syvään ymmärrykseen siitä, mitä asiakkaat haluavat ja kuinka se toteutetaan ohjelmiston avulla. Asiakkaan tarpeiden ymmärtäminen ei ole aina helppoa, varsinkaan kun ohjelmisto itsessään ei välttämättä tarjoa arvoa. Arvo ei myöskään muodostu pelkästään kehittämisvaiheesta, sillä myös suunnittelu ja käyttöönotto liittyvät vahvasti arvon luontiin. Ohjelmiston arvo rajoittuu harvoin mihinkään tiettyyn aikasidonnaiseen toimenpiteeseen. Mahdollisuus muokata koodia ajan myötä on usein hyvin tärkeää. (Poppendieck & Cusumano, 2012.)

Hukalla tarkoitetaan kaikkea, mikä ei tuo lisäarvoa asiakkaalle tai lisää tietoa, miten arvoa voitaisiin lisätä tehokkaammin. Hukka aiheuttaa siis ohjelmistokehityksen tehottomuutta. Yleisimpiä hukan aiheuttajia ohjelmistokehityksessä ovat tarpeettomat ominaisuudet, hukattu tieto, asioiden tekeminen samanaikaisesti ja vikojen etsiminen ja korjaaminen. (Peppendieck & Cusumano, 2012.)

Wang, Conboy ja Cawley (2012) toteavat hukan eliminoinnin olevan yksi tärkeimmistä ja eniten noudatetuista lean-periaatteista. Hukkaa voidaan torjua esimerkiksi niin sanotulla sisäänrakennetulla laadulla. Tällä tarkoitetaan, että jokaisen osa-alueen ohjelmistokehityksessä tulisi olla mahdollisimman laadukasta. Tämä toteutuu Poppendieckin ja Cusumanon (2012) mukaan kun pieniä ohjelmistoja liitetään osaksi suurempaa järjestelmää. Näin laadun varmistaminen on parempaa ja tarkempaa. Kooditarkastukset tukevat myös

laadukasta työtä, sillä lean-ohjelmistokehitys perustuu vahvasti virheiden löytämiseen ja korjaamiseen pitkin kehitysprosessia (Wang ym. 2012).

Kehittäminen on ennen kaikkea tiedon luontia ja sen upottamista tuotteeseen. Lean-kehityksessä tätä voidaan lähestyä kahdesta näkökulmasta. Ensimmäisen näkökulman alussa rakennetaan tarvittavat vähimmäisominaisuudet ja kehitetään tuotoksia mahdollisimman nopeasti. Tämän jälkeen tuotteen kehitystä viedään eteenpäin asiakkaiden palautteiden perusteella. Tällä palautteella pyritään vähentämään turhaa kehitystyötä, joka ei tuo oikeaa arvoa loppuasiakkaalle. (Poppendieck & Cusumano, 2012; Poppendieck & Poppendieck, 2003.)

Toisessa näkökulmassa tutkitaan monia eri vaihtoehtoja, mitä tulee esimerkiksi kielen tai arkkitehtuuriin valintaan. Lopullista päätöstä lykätään viimeiseen mahdolliseen hetkeen, jolloin päätös perustuu parhaaseen mahdolliseen olemassa olevaan tietoon eri vaihtoehdoista. Jos ennakkokartoitus on tehty huolellisesti, löytyy jokaiselle osa-alueelle vaihtoehtoinen vaihtoehto, joka palvelee juuri haluttua käyttötarkoitusta. Näitä kahta lähestymistä on myös mahdollista käyttää yhdessä, tärkeintä on löytää juuri sen hetkiseen kehitysprojektiin paras mahdollinen vaihtoehto alati muuttuvassa ohjelmistokehitysympäristössä. (Poppendieck & Cusumano, 2012.)

Lean-kehityksen lähtökohtana on rohkaista työhön sitoutuneita ihmisiä tiimityöskentelyn pariin. Vaikka osa lean-käytänteistä voi vaikuttaa hyvin prosessikeskeiseltä, kaiken ytimessä on kuitenkin ihmisten kannustaminen ja tiimityöskentelyn tukeminen. Ihmisiä ei pidetä resursseina vaan kannustaminen merkityksen luominen työtä kohtaan ja motivointi on keskeinen osa lean-kehitystä. Tätä voidaan tukea esimerkiksi jatkuvan oppimisen ja kannustamisen ilmapiirillä. (Poppendieck & Cusumano, 2012; Poppendieck & Poppendieck, 2003.)

3 KEHITTÄJÄKOKEMUS

Tässä luvussa käsitellään kehittäjäkokemusta ja ohjelmistokehityksen inhimillisiä tekijöitä. Luvussa kerrotaan kehittäjäkokemuksesta terminä, sen viitekehyksestä, sen osuudesta API-kehityksessä sekä ohjelmistokehitykseen liittyvistä inhimillisistä tekijöistä. Inhimillisten tekijöiden tarkastelu keskittyy etenkin motivaatioon, onnellisuuteen ja ilottomuuteen.

3.1 Kehittäjäkokemus terminä

Termi kokemus voidaan selittää auki monella tavalla riippuen, mistä näkökulmasta termiä tarkastelee. Yleisesti kokemuksella tarkoitetaan sekä välittömästi havaittuja tapahtumia että tapahtumien muistoja ja muistetuista tapahtumista saatua tietoa ja reflektiota. Ihmisen kokemus on lähtökohtaisesti aina subjektiivinen. Kykymme prosessoida uutta tietoa on rajallinen, minkä vuoksi jokaisen ihmisen todellisuus on yksilöllistä. Tätä yksilöllistä todellisuutta käytetään uuden tiedon tulkitsemiseen. (Fagerholm & Münch, 2012.)

Ohjelmistokehityksen ympäristö ja työtehtävät eroavat huomattavasti muista vaativista, tietokeskeisistä ympäristöistä ja työtehtävistä. Ohjelmistokehitys vaatii paljon sisäkkäistä ymmärrystä, sillä kehittäjät hyödyntävät ohjelmistoja rakentaakseen ohjelmistoja, joita voidaan puolestaan käyttää heidän työtehtävien tekoa varten. Ohjelmoijat myös usein parantavat ja laajentavat työkalujaan, jolloin he kehittävät lopputuotetta ja kehitysympäristöä samanaikaisesti. (Kuusinen, Petrie, Fagerholm & Mikkonen, 2016.)

Kehittäjäkokemuksella (*developer experience, DX*) tarkoitetaan kokemusta, joka tapahtuu ohjelmistokehitysprosessissa ohjelmoijan sekä ohjelmointityökalujen välillä, kun kehittäjä työskentelee palveluiden tai tuotteiden kehityksen parissa. Termi pitää sisällään kehittäjien motivaation, tunteet, tunnusomaiset piirteet ja toiminnat. Juuri tämän vuoksi kokemuksen ymmärtäminen on tärkeää organisaatiokontekstissa. (Fagerholm & Münch, 2012; Lee & Pan, 2021.) Moilanen, Niinioja, Seppänen ja Hokkanen (2018) kuvaavat

kehittäjäkokemusta kehittäjän pään sisäiseksi rakennelmaksi. Tähän rakennelmaan kuuluvat tunne omasta työstä ja sen tärkeyden tunne osana kokonaisuutta sekä kehittäjän käsitys kehitysympäristöstä.

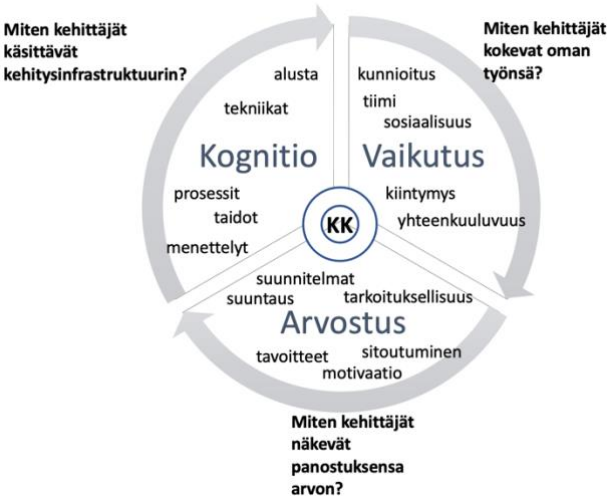
Kehittäjäkokemusta voidaan pitää käyttäjäkokemuksen (*user experience, UX*) erikoistapauksena. Termillä käyttäjäkokemus voi olla monia eri merkityksiä aina perinteisestä käytettävyydestä, hedonistisiin, tunteellisiin tai kokemusperäisiin teknologian käytön näkökulmiin (Forlizzi & Battarbee, 2004). Ergonomics of human-system interaction (2019) määrittää käyttäjäkokemuksen henkilön havainnoiksi ja vastauksiksi, jotka johtuvat järjestelmän, tuotteen tai palvelun käytöstä ja/tai odotetusta käytöstä. Hassenzahl ja Tractinsky (2006) määrittelevät käyttäjäkokemuksen puolestaan seurauksena käyttäjän sisäisestä tilasta (taipumukset, odotukset, tarpeet, motivaatio, mieliala), suunnitellun järjestelmän ominaisuuksista (monimutkaisuus, tarkoitus, käytettävyys, toiminnallisuus) ja kontekstista tai ympäristöstä, missä vuorovaikutus tapahtuu (esimerkiksi sosiaalinen ympäristö, käytön vapaaehtoisuus tai toiminnan mielekkyys).

Kehittäjäkokemuksen ja käyttäjäkokemuksen idea ja tarkoitusperä on sama, mutta kehittäjäkokemus on kaksijakoinen. Tällä tarkoitetaan sitä, että kehittäjä on työkalun käyttäjä, joka mahdollistaa järjestelmän käytön, sekä työkalua kehittävä taho. Käyttäjäkokemus ja kehittäjäkokemus ovat muuten hyvin samankaltaisia, mutta kehittäjäkokemus rajoittuu ainoastaan kehittäjiin, jotka vastaavat järjestelmän suunnittelusta ja kehittämisestä. (Kuusinen, 2016b; Lee & Pan, 2021.)

Siinä missä käyttäjäkokemus keskittyy tutkimaan järjestelmien käyttöä, kehittäjäkokemus keskittyy ohjelmistokehitykseen ohjelmistotyökalujen, mallinnuksen ja kehitysprosessien näkökulmasta. (Kuusinen, Petrie, Fagerholm & Mikkonen, 2016) Myös Moilanen ym. (2018) painottavat, että käyttäjäkokemuksessa fokus on tuotteessa ja sen käyttämisessä. Kehittäjäkokemuksen fokus on puolestaan tuotteen kehittämisessä. Näin ollen käyttäjäkokemuksen toimintamalli on käyttäjäkeskeinen, kun taas kehittäjäkokemus on prosessi-tuote-keskeinen.

Ohjelmoijakokemus (programmer experience, PX) termiä on käytettyä kehittäjäkokemuksen rinnalla kuvaamaan samaa ilmiötä (Morales, Rusu, Botella, Quinoses, 2019). Terminä kehittäjäkokemus on kuitenkin yleisempi ja käytetympi tieteellisissä julkaisuissa. Ekwoge, Fantão ja Dias-Neto (2017) nostavat omassa tutkimuksessaan esiin vielä kehittäjäkokemuksesta ja käyttäjäkokemuksesta johdetun testaajakokemuksen (tester experience, TX). Tällä tarkoitetaan kaikkia niitä kokemuksia, mitä testaajan työssä ilmenee sovelluskehityksen parissa. Koska testaamiseen liittyvät osa-alueet muodostavat yli 50 % ohjelmistokehityksen kuluista, on sillä hyvin keskeinen rooli ohjelmistoprojekteissa.

3.2 Kehittäjäkokemuksen viitekehys



Psykologiassa mieli jaetaan usein kolmeen tasoon: vaikutustasoon (affect), kognitiotasoon (cognition) ja arvostustasoon (conation). Fagerholm ja Münch (2012) käyttävät samaa jakoa kehittäjäkokemuksen viitekehityksen muodostamisessa. Tasot on havainnollistettu kuviossa 1. Myös Fantão, Dias-Neto ja Viana (2017) tunnistivat nämä kolme eri tasoa omassa julkaisussaan.

KUVIO 1 Kehittäjäkokemus: Käsitteellinen viitekehys. (Fagerholm & Münch, 2012)

Vaikutustasolla tarkoitetaan kehittäjien tunteisiin vaikuttavia osa-alueita. Näihin kuuluvat yhteenkuuluvuuden tunne ja työhön liittyvät sosiaaliset puolet, kuten tiimityöskentely sekä kiintymys. Nämä osa-alueet yhdessä luovat turvallisuuden tunteen. Kiintymys voidaan puolestaan jaksaa ihmisiin, työhön tai tapoihin liittyvään kiintymykseen. Ohjelmistokehitys työnä pitää sisällään sekä negatiivisia, että positiivisia tuntemuksia ja positiiviset tuntemukset ovatkin hyvin tärkeässä osassa hyvän kehittäjäkokemuksen muodostumisessa. (Fagerholm, 2012.)

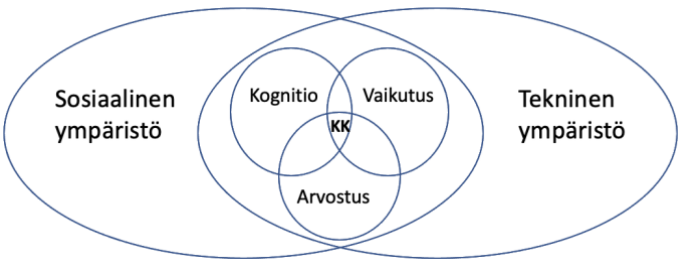
Kognitiolla tarkoitetaan tietoa sisältäviä henkisiä prosesseja, kuten päättelyä ja muistamista. Ohjelmistokehityksen kontekstissa nämä kognitiiviset näkökulmat liittyvät huomioimiseen, muistiin, ongelmanratkaisuun ja

päätöksentekoon. (Fagerholm, 2015.) Kehittäjäkokemuksessa kognitiotasolla viitataan siihen, miten kehittäjät hahmottavat tai näkevät käytössä olevan kehitysinfrastruktuurin. Tällä infrastruktuurilla tarkoitetaan ohjelmoinnin konkreettisia osa-alueita, joita ovat erilaiset prosessit, työskentelytekniikat sekä menetelmät ja henkilökohtaiset taidot ja käytössä olevat alustat yhdessä työkalujen kanssa. Näiden asioiden näkeminen positiivisessa valossa vaikuttaa kehittäjäkokemukseen positiivisesti. (Fagerholm & Münch, 2012.)

Arvostustasolla tarkoitetaan sitä, miten kehittäjät näkevät oman työnsä panoksen arvon. Taso pitää sisällään viisi osa-aluetta: suunnitelmallisuus, tavoitteellisuus, tarkoituksellisuus, motivaatio sekä sitoutuminen. Tarkoituksellinen, suunniteltu toiminta yhdessä henkilökohtaisten tavoitteiden kanssa lisäävät todennäköisesti määrätietoisuutta, motivaatiota ja sitoutumista, mitkä vaikuttavat kehittäjäkokemukseen positiivisesti. (Fagerholm & Münch, 2012.)

Väitöskirjassaan Fagerholm (2015) vie kehittäjäkokemuksen konseptia eteenpäin. Fagerholm jakaa kehittäjäkokemuksen kahteen eri osaan: sosiaaliseen ja tekniseen ympäristöön. Tämä on havainnollistettu kuviossa 2. Kehittäjäkokemus perustuu isolta osin sosiaalipsykologian perusmekanismeihin. Näitä mekanismeja ovat arvot, normit sekä ryhmädynaamiset mekanismit, kuten ryhmän muodostaminen ja ryhmän identiteetti. Kehittäjillä on omia uskomuksiaan, miten heidän työnsä vaikuttaa organisaation ja tiimin suorituskykyyn. He perustelevat suorituskykyä niin yksilön, tiimien, organisaatioiden, markkinan ja asiakkaan näkökulmasta. Sosiaalisen tiedon prosessointi onkin hyvin tärkeä osa ohjelmistokehitystä. Moilanen ym. (2018) painottavat kehittäjäyhteisön ymmärtämistä juuri näiden sosiaalisten tekijöiden vuoksi.

Ohjelmistokehitys pitää sosiaalisen puolen lisäksi sisällään lukuisia teknisiä asioita (technical artefacts). Näiden asioiden joukkoon kuuluvat ohjelmointikieli, kielellä kirjoitettu varsinainen koodi, työkalut koodin kirjoittamista varten, suunnitelmat, luonnokset, prosessit sekä metodit ajankäytön hallitsemista varten. Kehittäjäkokemus koostuukin kohtaamisista ja vuorovaikutuksista näiden asioiden ja toimintojen kanssa, joita kehittäjä kohtaa jokapäiväisessä työssään. Kun kehittäjä on vuorovaikutuksissa näiden asioiden kanssa, lopputuloksena syntyy kokemuksen tunne. (Fagerholm, 2015.)



KUVIO 2 Kehittäjäkokemuksen konsepti. (Fagerholm, 2015)

Useat viimeaikaiset tutkimukset osoittavat, että onnellisuus ja voimakastunteiset kokemukset ovat tärkeitä osa-alueita kehittäjäkokemuksen muodostumisessa. Onnellisuus ja kokemukset toimivat ohjelmistokehityksen toimintakyvyn ja laadun yhdistävänä tekijänä. (Graziotin, Fagerholm, Wang & Abrahamsson, 2018; Graziotin, Wang & Abrahamsson, 2015)

Kuusisen (2016a) mukaan kehittäjäkokemusta voidaan parantaa edistämällä kehittäjien mielenkiintoa ja nautintoa, tarjoamalla palkitsevia kokemuksia, tukemalla haaste-taito-tasapainoa ja osaamisen tunnetta. Hän nostaa myös toimintatietouden tukemisen ja hallinnan tunteen tarjoamisen tärkeiksi tekijöiksi hyvään kehittäjäkokemukseen.

3.3 Kehittäjäkokemus API-kehityksessä

Moilanen ym. (2018) nostavat kehittäjäkokemuksen tärkeäksi osaksi myös API-taloudessa. API:lla (application programming interface, ohjelmointirajapinta) tarkoitetaan digitaalista tuotetta ja alustatalouteen, internetiin ja tekoälyn hyödyntämiseen liittyvää teknologiaa. API kuvaa rajapintaa, jota ohjelmoija käyttää, kun hän haluaa kommunikoida koodikirjaston, ohjelmistokehityspaketin (software development kit, SDK), viitekehyksen, verkkopalvelun tai minkä tahansa ohjelmiston kanssa (Myers & Stylos, 2016).

Koska nämä rajapinnat ovat nykypäivänä läsnä lähes kaikessa, APIt ovat nousseet äärettömän tärkeäksi osaksi ohjelmistokehitystä. Monesti APIt ovatkin ensisijainen tapa, jolla yritykset kuljettavat palveluita ja dataa asiakkaiden suuntaan sekä yritysten välillä. (Iyer & Subramaniam, 2015.) Näiden seikkojen vuoksi API-kehitykseen liittyvien osa-alueiden ymmärtäminen on hyvin keskeisessä roolissa ohjelmistokehityksessä.

API-suunnittelun on usein hankalaa ja erikoistaitoja vaativaa työtä. Koska suunnittelussa käyttäjäkokemuksen ymmärtäminen on tärkeää, API-suunnittelijoiden on hyvä saada käyttää käyttäjäkokemuksen testaamiseen ja käytettävyyteen liittyviä metodeita. Tällä käyttäjäkokemuksen ymmärtämisellä on tärkeä vaikutus APIn käytettävyyden osalta. (Murphy, Kery, Alliyu, Macvean & Myers, 2018.)

Moilanen ym. (2018) jakavat hyvän kehittäjäkokemuksen muodostumisen neljään osa-alueeseen: arvon luotiin, matalaan kynnykseen, tukeen sekä dokumentointiin. Tätä on avattu tarkemmin taulukossa 1.

TAULUKKO 1 Hyvän kehittäjäkokemuksen nelikenttämalli (Moilanen ym. 2018)

Arvon luonti	**Matala kynnys**
Ratkaisee ongelman ja tekee sen hyvin	Löydettävyys
Selkeät liiketoimintamallit	Itsepalvelu
Tehokas ja luotettava	Live-ympäristö kokeilua varten
	Tuotteistetut SDK:t

	Freetier / trial
Tuki	**Dokumentointi**
Email	Ajankohtainen ja selkeä dokumentaatio sisältäen koodiesimerkit
Puhelinnumero	Case-kuvauksia
Foorumi	Koneluettava spesifikaatio
Chat	Roadmap
API-statistiikka ja tilannenäkymä	

Arvon luonnilla Moilanen ym. (2018) tarkoittavat sitä, että kehittäjille tarjottu API tekee sen, minkä pitääkin mahdollisimman hyvin. Mikäli näin ei ole, koettu arvo, ja sen kautta kehittäjäkokemus, vähenee merkittävästi. APIn tulee toimia ongelmitta ja nopeasti, sillä mikäli kehittäjä joutuu käyttämään epävarmaa ja hidasta APIa, lisää tämä kuormaa ja ylimääräistä työtä. Tämä puolestaan pitkittää kehitystyötä ja tuo lisää riskitekijöitä.

Myös Myers ja Stylos (2016) nostavat käytettävyyden tärkeäksi osa-alueeksi, mitä tulee kehittäjäkokemukseen ja hyvällä käytettävyydellä onkin löydetty olevan kytköksiä tuotteliaisuuteen ja kirjoitetun koodin laatuun. Hyvällä APIn käytettävyydellä Myers ja Stylo (2016) tarkoittavat tehokasta, oikeellista ja helposti opittavaa. Käytettävyyttä voidaan parantaa antamalla APIn käyttäjien testata sen toimivuutta, vaikka itse API ei olisi vielä kokonaan valmis. (Murphy, Kery, Alliyu, Macvean & Myers, 2018)

Matalla kynnyksellä tässä kontekstissa tarkoitetaan APIn käyttöönoton vaivattomuutta. Palveluiden käyttöönoton oletetaan nykypäivänä lähes aina tapahtuvan itsepalveluna. Itsepalvelun korostumisen vuoksi näkyvyys ja palvelun kokeileminen nopealla aikataululla on kriittistä. Mikäli palvelu tai rajapinta on maksumuurin takana, eikä edes kokeileminen veloituksetta ole vaihtoehto, kehittäjäkokemus kärsii ja ohjelmoija valitsee todennäköisesti kilpailevan palvelun. Tämän vuoksi esimerkiksi ilmaiset kokeilujaksot palvelun käyttöönoton yhteydessä ovat kannattavia, sillä niiden avulla kehittäjät pääsevät opettelemaan palvelun varsinaista käyttöä ja kynnys varsinaiseen käyttöönottoon madaltuu. (Moilanen ym. 2018) Kehittäjien pitäisi myös pystyä ratkaisemaan ongelmansa mahdollisimman itsenäisesti APIn tarjoamien virheilmoitusten avulla. (Robillard & DeLine, 2010)

Vaikka käyttöönotto olisi vaivatonta ja tehty mahdollisimman helpoksi, tulisi tarjolla olla myös tukipalveluita. Kehittäjien mahdollisten kysymysten ennakointi on haastavaa, minkä vuoksi dialogin luonti on tärkeää. Tällä voidaan tarkoittaa kaikkea perinteisistä puhelinnumeroista ja sähköpostiosoitteista aina varta vasten perustettuun chat-asiakaspalveluun ja jo olemassa oleviin kehittäjäfoorumeihin. Moilanen ym. 2018.) Moilanen ym. (2018) ja Murphy ym. (2018) nostavat kehittäjäfoorumi Stack Overflown tärkeäksi kanavaksi, mitä tulee kommunikointi- ja palautekanaviin. Sen lisäksi, että tukipalvelut ovat kunnossa, myös järjestelmän statussivun olisi hyvä olla helposti saatavilla (Moilanen ym. 2018; Robillard & DeLine, 2010)

Moilanen ym. (2018) nostavat dokumentaation yhdeksi keskeisemmäksi osaksi kehittäjäkokemusta. Hyvä dokumentaatio on johdonmukainen ja selkeä kuvaus APIn ominaisuuksista. Tämän lisäksi dokumentaation tulisi sisältää kopioitavia koodiesimerkkejä. Esimerkkien on tarkoitus luoda mielikuva käyttöönoton helppoudesta ja mahdollistaa pikainen ensikosketus käyttöönottoon. Koodi auttaa ymmärtämään, miten palvelu varsinaisesti toimii, kun taas dokumentaatio avaa sen koko potentiaalin. Mikäli dokumentaatio ei ole ajan tasalla, johtaa se todennäköisesti turhautumiseen ja sitä kautta rikkinäiseen kehittäjäkokemukseen. Myös Ford ja Parnin (2015) toteavat dokumentaation puutteen tai vajavaisuuden turhautumista aiheuttavaksi tekijäksi ohjelmistokehityksessä. Robillardin ja DeLinen (2010) tutkimuksessa puutteellinen dokumentaatio nousi isoimmaksi esteeksi, kun kehittäjät opettelivat uuden APIn toimintaa. API-dokumentaatiosta kerrotaan tarkemmin kappaleessa 4.3.

3.4 Ohjelmistokehityksen inhimilliset tekijät

Capretzin (2014) mukaan ohjelmisto (software) on ihmisen toiminnan sivutuote, joka pitää sisällään ongelmanratkaisukyvyn, kognitiiviset näkökulmat sekä sosiaalisen vuorovaikutuksen. Ihmiset ovat kuitenkin arvaamattomampia ja monimutkaisempia kuin ohjelmisto. Juuri tämä ihmisten monimutkaisuus luo monimutkaista dynamiikkaa ohjelmistokehitysprosessiin, jota on vaikea olla huomioimatta. Ohjelmistokehitystä voidaankin pitää sen monitahoisuuden vuoksi yhtenä vaikeimmista ihmisten suorittamista tehtävistä.

Onnistunut ohjelmistoprojekti on monimutkaisten prosessien tulos, mutta suurissa määrin onnistumiseen vaikuttavat projektissa olevat ihmiset. Kehittäjät ovat avainasemassa onnistuneen ohjelmointikehitysprojektin maaliin saattamisessa, ei ainoastaan tyypillisten tehtävien suorittajina, vaan koko kehitysprosessin ytimenä. (Destefanis, Ortu, Counsell, Swift, Marchesi & Tonelli, 2016; Fagerholm, Ikonen, Kettunen, Münch, Roto & Abrahamsson, 2015.)

Ohjelmistokehitys on ennen kaikkea ihmislähtöistä toimintaa (Curtis, Krasner & Iscoe, 1988). Monet tutkimukset ovat osoittaneet inhimillisten tekijöiden olevan kaikista tärkeimpiä tekijöitä ohjelmistokehityksessä, kun puhutaan suorituskyvyn sekä tehdyn työn laadusta. (Mockus, 2010; DeMarco & Lister, 1985; Fagerholm & Münch, 2012)

Menetelmät ja työkalut puolestaan vahvistavat hyvin koordinoitujen ja osaavien kehitystiimien tuottavuutta (Fagerholm & Münch, 2012). Fagerholmin yms. (2015) mukaan inhimillisiä tekijöitä ohjelmistokehityksessä ovat motivaatio, taidot, tyytyväisyys, arvot sekä persoonallisuus. Nämä tekijät nousevat esiin, kun yritykset luovat tiimejä, työympäristöjä sekä erilaisia prosesseja. Destefanis ym. (2015) nostavat puolestaan kohteliaisuuden yhdeksi keskeiseksi tekijäksi, mitä tulee kehitysprojektien tehokkuuteen. Mitä kohteliaampia kehittäjät olivat,

sitä vähemmän aikaa heillä meni ongelmatilanteiden ratkaisujen kanssa. Myös projektien houkuttelevaisuus kasvaa kohteliaisuuden myötä.

DeMarco ja Lister (1998) toteavat, että yritykset, jotka panostavat ihmisiin, menestyvät myös pitkällä aikavälillä. Capretz puolestaan (2014) jatkaa, että iso osa ohjelmistoprojekteista katuu juuri ihmislähtöisiin ongelmiin. Tästä huolimatta ohjelmistokehityksen inhimilliset näkökulmat eivät hänen mielestään saa ansaitsemaansa huomiota.

3.4.1 Motivaatio

Ohjelmistokehityksessä työskentelevät ihmiset ovat pitkälti tietotyöntekijöitä, minkä vuoksi heidän motivointinsa työhön on keskeisessä osassa. Francan, Gouveian, Santoksen, Santanan ja da Silvan (2011) mukaan motivaatiota vähentäviä tekijöitä ohjelmistokehityksessä ovat muun muassa stressi, työn monimutkaisuus, huonot työskentelyolosuhteet, resurssien puute, huono johto ja kilpailukyvytön korvaus tehdystä työstä. Näiden lisäksi yksitoikkoisuus nousi motivaatiota vähentävien tekijöiden joukkoon.

Motivaatiota lisääviä osa-alueita tuotteliaisuuden osalta olivat puolestaan työntekijöiden osallistaminen päätöksentekoprosesseihin, kehitystarpeiden kartoittaminen, selkeän urapolun olemassaolo, työtehtävien monipuolisuus sekä autonomian tarjoaminen kehittäjille. Nämä korostuivat etenkin kokeiden kehittäjien keskuudessa. Mikäli jokin asia motivoi kehittäjää uran alkuvaiheessa, voi tämä sama asia vaihtua epämotivoivaksi tekijäksi uran myöhemmässä vaiheessa. Uransa alkuvaiheessa oleva kehittäjä voi esimerkiksi motivoitua johdon tiiviillä läsnäololla, mutta kokeneempi kehittäjä voi kokea esimiehen jatkuvan läsnäolon hyvinkin epämotivoivana tekijänä (Beecham, Baddoo, Hall, Robinson & Sharp, 2008).

Muita motivaatioon positiivisesti vaikuttavia tekijöitä Francan ym. (2011) katsauksessa olivat kehittämiskäytänteet, ongelmanratkaisu, tiimityöskentely, haastavuus, luovuus ja asiakassuhteet. Beecham ym. (2008) lisäävät tähän vielä tarpeen samaistua työtehtävään. Tällä tarkoitetaan selkeitä tavoitteita, henkilökohtaista mielenkiintoa, työtehtävien merkityksen ymmärtämistä ja työtyytyväisyyttä. Ylläpitotyötehtävillä on puolestaan huomattu olevan sekä positiivinen että negatiivinen vaikutus kehittäjiin. Wun, Gerlachin ja Youngin (2007) mukaan avoimen lähdekoodin kanssa työskentelevät kehittäjät motivoituvat muiden auttamisesta, urakehityksestä sekä henkilökohtaisten tarpeiden täyttymisestä ohjelmistokehityksessä.

Li, Tan ja Teo (2012) nostavat tutkimuksessaan tehokkaan johtamistyylin keskeiseen osaan, kun puhutaan avoimen lähdekoodin kehittäjien motivoinnista. Kun tunnistetaan erilaiset motivaatiota tehostavat ja muuttavat johtamiskäytänteet, ohjelmistoyritykset voivat käyttää näitä käytänteitä suuntaviivoina kehittäjien motivoimiseksi. Muutosjohtamista hyödyntävien johtajien tulisi rohkaista etenkin luonnostaan motivoituneita kehittäjiä, kun taas

transaktionaalisen johtamisen on todettu toimivan ulkoisesti motivoituneisiin kehittäjiin. Luonnollisella motivaatiolla tarkoitetaan motivaatiota, joka kumpuaa kehittäjän sisältä esimerkiksi mielenkiinnosta aihetta kohtaan. Ulkoisessa motivaatiossa kehittäjän motivaatio muodostuu puolestaan esimerkiksi urakehitysmahdollisuuksista tai työtaitojen karttumisesta.

Taloudelliset kannustimet, kuten palkka ja edut, ovat yleisiä ulkoisia kannustimia kaupallisissa ohjelmistoprojekteissa. Lin ym. (2012) mukaan johtamistyylin, missä johtaja pyrkii mahdollistamaan tehokkaan työskentelyn, on todettu olevan yhtä hyvä motivaation lähde varsinkin yhteisöllisissä kehitysprojekteissa. Johtaja pyrkii auttamaan työntekijöitä keskittymään mahdollisten poikkeamien ja virheiden korjaamiseen luoden samalla tehokkaan työskentely-ympäristön (Bass, 1999). Bass (2000) korostaa, että tämä johtamistyyli ei keskity pelkästään virheisiin vaan pyrkii auttamaan toimimaan niiden kanssa. Puolestaan passiivisen johtamistyylin, jossa reagoidaan vasta virheiden tapahtumisen jälkeen, ei ole todettu lisäävään kehittäjien motivaatiota (Li ym. 2012). Barbuto (2005) toteaakin tämän johtamistyylin olevan hyvin negatiivinen, ja sillä onkin motivaatiota vähentävä vaikutus.

3.4.2 Onnellisuus

Lukuisat tutkimukset osoittavat, että onnellisuus ja tunteisiin vaikuttavat kokemukset ovat isossa osassa ohjelmistokehityksen suorituskyvyn ja laadun syntymisessä (Ortu, Adams, Destefanis, Tourani, Marchesi & Tonelli, 2015; Destefanis ym. 2016; Mäntylä, Adams, Destefanis, Graziotin & Ortu, 2016; Khan, Hierons, Brinkman, 2006).

Onnelliset kehittäjät omasivat Graziotinin, Fagerholmin, Wangin ja Abrahamssonin (2018) tutkimuksen mukaan paremmat ongelmanratkaisutaidot, keskittymiskyvyn, taitotason, minkä lisäksi he oppivat myös paremmin. Ohjelmistokehityksen luonteen vuoksi nämä ominaisuudet ovat hyvin haluttuja ja johtavatkin todennäköisesti positiivisiin lopputuloksiin. Myös korkea motivaatio nousi esiin onnellisuutta kokeneilla kehittäjillä. Näin ollen, mikäli esimerkiksi työnantajan tavoitteena on vaikuttaa motivaatioon positiivisesti, tulisi työntekijöiden onnellisuus nostaa keskiöön. Myös Snyder ja Lopez (2005) toteavat, että positiivinen mieliala ja iloinen asenne parantavat luovaa ongelmanratkaisukykyä.

Samassa Graziotinin ym. (2018) tutkimuksessa sekä onnelliset kehittäjät kokivat pitkäkestoista flow-tilaa, tehokkuuden, yhteistyön, luovuuden ja sitoutumisen kasvamista. Täten voidaan siis osoittaa, että onnellisuus ohjelmistokehityksessä tekee työskentelystä nopeampaa, tasaisempaa ja sosiaalisempaa. Vahva sitoutuminen sovittuihin prosesseihin tarkoittaa parhaiden käytänteiden seuraamista sekä testien ja dokumentaation kirjoittamista, mitkä lisäävät työskentelyn laatua ja tehokkuutta. Myös

Graziotinin, Wangin ja Abrahamssonin (2015) mukaan positiivisten vaikutusten vaaliminen lisää kehittäjien suorituskykyä.

Kalunzniackyn (2004) mukaan kehittäjien ja heidän ympäristönsä henkisen puolen huomioiminen parantaa IT työntekijöiden työtä. Hän korostaakin, että onnistuneiden ohjelmistotuotosten ja hyvän tiimityöskentelyn taustalla on lähes poikkeuksetta onnellinen kehittäjä.

3.4.3 Ilottomuus

Ilottomuuden (unhappiness) on todettu vaikuttavan negatiivisesti ohjelmistokehittäjiin ja heidän ohjelmointityönsä lopputulemaan (Graziotin, Fagerholm, Wang & Abrahamsson, 2017).

Graziotinin ym. (2018) mukaan ilottomuus aiheuttaa epätasaisuutta ohjelmistokehityksessä. Tämä näyttäytyy alhaisena tuottavuutena, viivästymisinä ja rikkonaisena työskentelynä. Ilottomuus aiheuttaa myös sovittuihin prosesseihin sitoutumisen alenemista ja voi johtaa näiden prosesseista sivuuttamiseen. Prosessien sivuuttamisella on usein negatiivinen vaikutus ohjelmiston laatuun.

Ilottomuutta aiheuttavat tekijät voidaan Graziotinin ym. (2017) mukaan jakaa kahteen osa-alueeseen: kehittäjän omaan olemukseen (developer's own being) ja ulkoisiin syihin (external causes). Kehittäjän omalla olemuksella tarkoitetaan yksittäisten kehittäjien kokemia ilottomuuden tunteita, jotka linkittyvät kehittäjiin henkilökohtaisella tasolla. Ulkoisilla syillä viitataan puolestaan niihin ilottomuutta aiheuttaviin tekijöihin, jotka vaikuttavat kehittäjiin, mutta joihin heillä ei ole juurikaan mahdollisuutta itse vaikuttaa.

Ongelmanratkaisukyky on yksi ohjelmistokehityksen tärkeimmistä taidoista ja usein työ on älyllisesti hyvinkin vaativaa. On tavallista, että kehittäjät jäävät jumiin työtehtäviensä kanssa, oli kyseessä sitten koodaamiseen tai testaukseen liittyvä tehtävä. Ongelmaan juuttumista pidetäänkin isoimpana ilottomuutta aiheuttavana tekijänä, mitä tulee kehittäjän omaan olemukseen liittyvään osa-alueeseen. (Graziotin ym. 2017.)

Toinen merkittävä sisäinen ilottomuuden aiheuttaja kehittäjille on riittämättömyyden tunne ja se voi ilmetä riittävien tietojen tai taitojen puutteena. Riittämättömyyden tunne voi aiheuttaa kehittäjille mielikuvan, että he ovat työtehtävissään osaamattomia ja, että he eivät koe käytettyjä koodauskieliä, työkaluja, viitekehyksiä tai kehittämismenetelmiä omikseen. Kolmas ilottomuutta aiheuttava sisäinen tekijä kehittäjien henkilökohtaisen elämän aiheuttavat piirteet. Koska ohjelmistokehittäjät ovat ihmisiä siinä missä muutkin, vaikuttaa heidän henkilökohtaisessa elämässänsä tapahtuvat muutokset ja käänteet myös työelämään. Graziotinin ym. (2017) mukaan perhe-elämään liittyvät ongelmat liittyvät hyvin usein kehittäjien ilottomuuteen.

Tärkeimmäksi ilottomuutta aiheuttavaksi ulkoiseksi tekijäksi Graziotin ym. (2017) nostivat aikatauluista aiheutuvan paineen. Aikatauluja

kommentoidaankin usein epärealistisiksi ja perusteettomiksi, ja nämä mahdottomiksi koetut aikarajat aiheuttavat kehittäjissä todella negatiivisia tunteita. Mahdottomien aikamääreiden lisäksi heikko koodin laatu ja huonot koodauskäytänteet ovat ilottomuutta aiheuttavia ulkoisia tekijöitä. Huonolla koodilla tarkoitetaan tässä tapauksessa muiden ihmisten tuottamaa koodia, joka on joko täysin lukukelvotonta, heikosti muotoiltu ja jäsennelty tai kommentoimatonta. Tämä linkittyy hyvin kolmanneksi eniten ilottomuutta aiheuttavaan tekijään, mikä on alisuoriutuvat kollegat. Ohjelmistokehitys on usein tiimityöskentelyä, minkä vuoksi kollegoiden kanssa toimeen tuleminen korostuu. Kehittäjiä turhauttaa etenkin se, mikäli kollegat eivät uhraa aikaa uusiin käytäntöihin ja teknologioihin oppimiseen ja niihin perehtymiseen alati muuttuvassa ympäristössä. (Ford & Parnin, 2015; Graziotin ym. 2017.)

Epätasaisuuden ohella ilottomuus aiheuttaa kehittäjien motivaation laskua. Tämä on huomattavaa, sillä motivaatiota pidetään erittäin tärkeänä osana ohjelmistokehitystä. Negatiiviset tunteet ja kokemukset, mitkä johtavat ilottomuuteen, voivat pahimmassa tapauksessa aiheuttaa irtisanoutumisen työelämästä. Näiden seuraamusten ohella Graziotin ym. (2018) kertovat luovuuden vähenemisen yhdeksi ilottomuuden tuomista osa-alueista. Tämä on niin ikään tärkeä huomio, koska esimerkiksi Liu, Feng, Li, Jing ja Yang (2016) nostavat luovuuden hyvin suureen rooliin nykypäivän ohjelmistokehityksessä. Mikäli motivaatio alenee, aiheuttaa se valtavasti ongelmia ohjelmistokehityksen suhteen.

Ohjelmistokehittäjien täytyy jatkuvasti oppia uusia teknologioita, sopeutua uusiin ympäristöihin ja selviytyä oppimisen tuomista haasteista työnteossa. Näissä tilanteissa epäonnistuminen voi aiheuttaa pahimmassa tapauksessa suurta turhautuneisuuden tunnetta työtä kohtaan ja sitä kautta vaikuttaa negatiivisesti ohjelmoijan työhön aiheuttaen ilottomuutta. Vuonna 2015 tehdyn tutkimuksen mukaan (Ford & Parnin) jopa 67 % vastaajista totesi, että turhautuminen työhön on vakava ongelma ja vastauksista saatiin johdettua 11 tarkempaa syytä. Näitä syitä olivat muun muassa kokonaisuuden hahmottamisen vaikeus koodatessa, uudet ohjelmointityökalut, kuten uusi ohjelmointikieli, projektin suuruus, dokumentaation puute tai keskeneräisyys ja ohjelmointikokemus.

4 DOKUMENTAATIO

Tässä luvussa kerrotaan dokumentaatiosta ja sen tärkeydestä ohjelmistokehityksen kontekstissa. Tämän jälkeen tunnistetaan teknisen dokumentaation tyypit, mitä seuraa perehtyminen API-dokumentaatioon.

Dokumentaatio on tärkeä osa ohjelmistokehitystä ja ylläpitoa. Parnas (2011) määrittelee dokumentaation ohjelmistokehityksessä, miksi tahansa artefaktiksi, joka auttaa viestittämään tietoa järjestelmästä eri sidosryhmille, kuten kehittäjille ja asiakkaille. Dokumentaatio voidaan myös määritellä tietona, joka kertoo, miten ohjelmistoa käytetään (Systems and software engineering -- Developing user documentation in an agile environment, 2012). Tällaisten kuvausten oletetaan antavan tarkan kuvauksen järjestelmästä.

Ohjelmistokehitysdokumentaatio voidaan jakaa karkeasti kahteen osaan: tekniseen dokumentaation ja käyttöoppaisiin. Tästä jaosta voidaan käyttää myös nimityksiä tekninen dokumentaatio ja ei-tekninen dokumentaatio (Garousi, Garousi-Yusifoğlu, Ruhe, Zhi, Moussavi & Smith, 2015). Tässä luvussa keskitytään nimenomaan tekniseen dokumentaatioon.

4.1 Dokumentaatio ohjelmistokehityksessä

Ohjelmistokehittäjät tukeutuvat tekniseen dokumentaatioon, kun heillä on tarve saada käsitys ohjelmasta ja ymmärtää järjestelmää paremmin (Von Mayrhauser & Vans, 1995). Ohjelmistojen teknistä dokumentaatiota pidetäänkin yhtenä ohjelmistokehityksen tärkeimmistä käytänteistä. Sen tarkoituksena on helpottaa järjestelmän ylläpitoa ja kehitystoimia. (de Souza, Anquetil & de Oliveira, 2006.)

Mikäli dokumentaatiota ei ole, lähdekoodi on ainoa tiedonlähde ohjelmistojärjestelmälle. Lähdekoodin lukeminen on kuitenkin huomattavasti teknistä dokumentaatiota työläämpi ja aikaa vievä tapa saada tietoa järjestelmästä ja sen käytöstä. (Von Mayrhauser & Vans, 1995.) Siinä, missä lähdekoodi on ainoastaan tekstimuodossa, tekninen dokumentaatio voi tekstimuodon lisäksi olla myös graafisessa muodossa esimerkiksi UML:n

(Unified Modeling Language) avulla (Garousi, Garousi, Moussavi, Ruhe & Smith, 2013).

Lähdekoodi koettiin kuitenkin Garousi ym. (2013) tutkimuksessa dokumentaatiota tärkeämmäksi materiaaliksi, mitä tuli ylläpitotoimenpiteiden, kuten vikojen korjaamisten, hoitamiseen. Samassa tutkimuksessa suunnitteludokumentit nousivat puolestaan kehitysvaiheen tärkeimmiksi ohjelmistokehittäjien dokumenteiksi. Mitä tuli dokumentaation käyttöön, organisaatioiden kokeneimmat työntekijät käyttivät tarjottua dokumentaatiota vähemmän verrattuna kokemattomiin työntekijöihin. Tämän vuoksi dokumentaation valmistelun yhteydessä olisi hyvä pitää mielessä etenkin uusien työntekijöiden tarpeet.

4.2 Teknisen dokumentaation tyypit

Ohjelmistodokumentaatio muodostuu Horchin (2003) mukaan hallinto- (management), kehitys- (development), testaus- (test) ja käyttäjädokumentaatiosta (user), ja sen tarkoitus onkin seurata ohjelmiston koko kehityskaarta. Tässä luvussa keskitytään etenkin kehitysdokumentaatioon ja testausdokumentaatioon.

4.2.1 Hallintodokumentaatio

Jokaiseen ohjelmistokehitysprojektiin liittyy hallinnollisia osa-alueita, oli kyseessä sitten pieni tai hyvin suuri projekti. Hallinnollisella dokumentaatiolla tarkoitetaan niitä dokumentteja tai asiakirjoja, jotka ohjaavat projektin prosesseja eteenpäin. Etenkin ohjelmiston kehityssuunnitelma (software development plan, SDP), ohjelmiston laatusuunnitelma (software quality system plan, SQSP) ja kokoonpanon hallintasuunnitelma (configuration management plan, CMP) ovat lähtökohtaisesti osana kaikkia ohjelmistoprojekteja. (Horch, 2003.)

4.2.2 Kehitysdokumentaatio

Ohjelmistokehitysprojektit tuottavat aina kehitysdokumentteja. Tällä dokumentaatiolla tarkoitetaan kehittäjien luomaa kirjallista materiaalia, jolla seurataan ohjelmiston kehitystä aina sen alkuvaiheista valmiiseen tuotokseen asti. Dokumenttien muotoiluilla ei ole niin selvää määritelmää, tärkeintä on niiden sisältö. (Horch, 2003.)

Tärkein kehitysdokumenteista vaatimustenmäärittelydokumentti (Horch, 2003; Parnas, 2011). Siinä tarkoituksena on määritellä, mitä tuleva ohjelmisto tulee pitämään sisällään. Dokumentissa kuvataan ratkaistava ongelma, mahdolliset ympäristölle tai suorituskyvylle asetettavat rajoitukset, koko- ja aikarajoitukset, laitteistorajoitukset sekä kaikki muut tarvittavat tiedot, joita tarvitaan ongelman tai toiminnon määrittelyyn. Ilman selvää määrittelyä, on

hyvin haastavaa määrittää valmistuminen ja se, kuinka valmistuminen on tarkoitus saavuttaa. (Horch, 2003.)

Suunnitteludokumenttien tarkoitus on kirkastaa, miten ohjelmistoa lähestytään ja miten aiemmin määritellyt vaatimukset täytetään. Suunnitteludokumentaatio voidaan jakaa kahteen osaan: esisuunnitteluun ja yksityiskohtaiseen suunnitteluun. Esisuunnittelussa vaatimukset jaetaan alustavasti toiminnallisiin ryhmiin jatkosuunnittelua varten. Jokainen ryhmä edustaa jotain isoa osaa ohjelmistojärjestelmästä. Esisuunnittelun tulee tarkentaa lähestymistapa, jolla toiminto suositetaan, tietokantavaatimukset sekä rajapinnat muihin järjestelmässä oleviin toiminnallisiin ryhmiin. Suunnitelman on myös määriteltävä rajapinnat ulkomaailmaan, kuten muihin verkkoihin, päätteisiin ja ohjelmistojärjestelmiin. (Horch, 2003.)

Yksityiskohtainen suunnittelu pitää sisällään sen, miten jokainen esisuunnittelussa mietitty osa-alue tullaan toteuttamaan ohjelmoinnin avulla. Tämän lisäksi se pitää sisällään viimeisen version, joka toteutetaan, kun ohjelmointi ja testaus ovat valmiita. Viimeisen version tärkeys korostuu, sillä se toimii apuvälineenä ohjelmiston tulevaisuuden ylläpitäjille. Dokumentti toimii tuotteen perustuksena, mihin kaikki muutokset tehdään, jos ohjelmistossa havaitaan vikoja, tai siihen tehdään ajan myötä lisäyksiä. Jatkuvan ylläpidon rooli siis korostuu tämän dokumentin osalta. (Horch, 2003.)

Ohjelmistokehityksen dokumentaatiossa yksi tärkeimmistä osa-alueista on sen jatkuva muokattavuus. Tällä tarkoitetaan sitä, että kun dokumentti on kirjoitettu ja hyväksytty, se voi muuttua ajan myötä. Kun kehittäjä huomaa työssään virheen tai vian, joka vaatii korjausta, samat muutokset tulee kirjata myös dokumentaation puolelle. Näin voidaan ylläpitää dokumentaation oikeellisuutta ja sitä, että se palvelee sen käyttäjää parhaalla mahdollisella tavalla. Moni ohjelmointiprojekti joutuu ongelmiin, kun dokumentaatio ei pysy muun ohjelmistokehityksen perässä. Kun dokumentaatio ei pysy kehityksen perässä, ei siitä ole enää hyötyä myöhemmissä testaus- ja ylläpitovaiheissa. Dokumentaation korjaaminen jälkikäteen on hankalaa ja joissakin tapaukissa lähes mahdotonta. (Horch, 2003.) Kuten jo aiemmin mainittu, puutteellinen ja virheellinen dokumentaatio on yksi isommista kehittäjien mainitsemista turhautumisen aiheista (Ford & Parnin, 2015; Robillard & DeLine, 2010).

Lethbridge, Singer ja Forward (2003) argumentoivat omassa tutkimuksessaan kuitenkin, että myös vanhentuneesta dokumentaatiosta voi olla hyötyä. Yli puolet heidän tutkimukseensa vastaajista ohjelmistokehittäjistä oli vahvasti sitä mieltä, että dokumentaatio voi olla hyödyllinen, vaikka se ei olisi täysin ajan tasalla. Vanhentunut dokumentaatio voi tarjota järjestelmän toimimisesta korkeatasoista ymmärrystä. Kuitenkin mitä lähemmäksi oikeaa koodia dokumentaatio meni, sitä tärkeämmäksi dokumentaation oikeellisuus nousi.

Aghajanin ym. (2019) toteavat kehittäjien pitävät etenkin dokumentaatiosta, joka on virheetöntä, valmista, päivitetty ajankohtaiseksi, huollettavissa, helposti luettavissa sekä helppokäyttöistä. Nämä tekijät ovat linjassa Garousin ym. (2013) tutkimuksessa nousseiden dokumentaation

tärkeimpien laatutekijöiden kanssa, joita olivat ajantasaisuus, tarkkuus ja täydellisyys.

Ohjelmiston ylläpitovaiheen dokumentaation käyttö keskittyy enimmäkseen järjestelmän ja ohjelmiston ymmärtämiseen. Kun dokumentaatio on tuotettu ja hyväksytty, se on valmis käytettäväksi eri tarkoituksiin ohjelmistokehityksen elinkaaren aikana. Yleensä dokumentaation käyttö riippuu kuitenkin dokumentaation käyttävän henkilön tarpeista. (Garousi ym. 2015.)

4.2.3 Testausdokumentaatio

Testausdokumentaatio pitää sisällään nimensä mukaan kaikki testaukseen liittyvät dokumentit. Testaussuunnitelma on testausdokumentaation ensimmäinen osa. Suunnitelma perustuu vahvasti alkuperäisiin määritelmiin ja alustava testaussuunnitelma luodaankin määrittelyvaiheen yhteydessä. Tämän tarkoituksena on varmistaa, että määritelmät ovat mitattavissa ja testattavissa. Suunnitelma tehdään etenkin testaustyökalujen, datageneraattorien ja simulaattorien osalta, ja sen on tarkoitus elää ja kasvaa läpi koko kehitysvaiheen. (Horch, 2003.) Graham (2002) toteaa, että kun alkuvaiheen vaatimustenmäärittelyt ja testaaminen linkittyvät toisiinsa, voidaan tällä usein säästää projektissa aikaa ja rahaa.

Testidatan määrittely on myös olennainen osa testausdokumentaatiota. Se tarkoituksena on todistaa, että ohjelmisto toimii juuri sillä tavalla, miten se on määritelty alkuvaiheessa ohjelmistokehitysprojektia. Datan pitää siis pitää sisällään mahdollisimman laaja kirjo virheitä ja poikkeavuuksia. Testausmenettelytavat muodostuvat puolestaan siinä vaiheessa, kun varsinainen ohjelmiston suunnittelu ja ohjelmointi alkavat. Menettelytavoilla tarkoitetaan askel askeleelta etenevää testausta, jossa jokainen testauksen toiminto, datansyöttö ja odotettu vastaus kirjataan ylös. Tällä tavoin testauksen ehdot pysyvät hallittuina ja ohjelmiston antavia vastauksia voidaan verrata odotettuihin tuloksiin. Mikäli odotettujen tulosten ja ohjelmiston antavien oikeiden tulosten välillä on eroja, voidaan olettaa, että ohjelmassa on mahdollisesti jotain vikaa. (Horch, 2003.)

Varsinaiset testit tallennetaan testiraporteille. Raporteille täsmentyvät odotetut ja todelliset tulokset sekä tulosten johtopäätökset. Myös poikkeavuudet ja niiden alttius tallennetaan. Raportit ovat avainasemassa varsinkin, kun arvioidaan testien lopullinen hyödyllisyys. (Horch, 2003.)

4.2.4 Käyttäjädokumentaatio

Paraskin ohjelmisto voi olla hyödytön, mikäli se loppukäyttäjät eivät osaa käyttää sitä. Käyttäjädokumentaatio pitää ohjekirjojen lisäksi sisällään myös esimerkiksi ylläpitoon, koulutukseen ja versionhallintaan liittyvät dokumentit. Käyttäjädokumentaation tehtävänä on näyttää ja kertoa, miten ohjelmistoa käytetään. Dokumentaatiossa pitäisi käydä ilmi järjestelmän kuvaus yleisesti, määritellyt formaatit ja syötteet sekä ne tulosteet, jotka syntyvät järjestelmän käytöstä. (Horch, 2003.)

Fisher (2001) nostaa laadukkaan ja tehokkaan käyttäjädokumentaation yhdeksi osatekijäksi, kun halutaan määritellä, onko ohjelmistoprojekti onnistunut. Asiakastyytyväisyys onkin lähtökohtaisesti lähes aina isossa roolissa ohjelmistokehitystä. Loppuasiakkaan mukaan ottaminen dokumentaation tarkasteluvaiheessa voisi Aghajani ym. (2019) mukaan vähentää käyttäjien väärinymmärryksiä ja pienentää uuden järjestelmän opetteluun käytettyä aikaa. Tämä on tärkeää varsinkin, koska kehittäjän näkemys dokumentaation ymmärtämisestä ei välttämättä kohtaa asiakkaan ymmärryksen kanssa.

4.3 API-dokumentaatio

Suurin osa ohjelmistokehityksestä käyttää viitekehyksiä ja kirjastoja, joiden toiminnallisuudet ovat saatavilla API:iden kautta (Robillard, 2009). Jotta API:ja voidaan käyttää mahdollisimman tehokkaasti, niitä on opeteltava käyttämään oikein. Oppimisen helpottamiseksi API:jen yhteydessä on yleensä olemassa käyttöä tukeva dokumentaatio. (Uddin & Robillard, 2015.) Robillardin (2009) tutkimuksen mukaan, jopa 78 % vastaajista kertoo oppivansa uuden API:n käytön sen dokumentaation avulla.

API-dokumentaatio voi olla ohjelmistokehittäjille hyvinkin tärkeä heidän työnsä onnistumisen kannalta. Mikäli dokumentaatio on hyvällä mallilla, kehittäjät voivat työskennellä tehokkaammin, mikä puolestaan parantaa kyseisen APIn mainetta kehittäjien keskuudessa. Mikäli dokumentaatio ei puolestaan ole linjassa kehittäjien odotusten kanssa, voi tämä johtaa turhautumiseen, suuren ajanhukkaan ja jopa kyseisen APIn hylkäämiseen. (Robillard & DeLine, 2010.) Laadukkaan dokumentaation tuottaminen ja tarjoaminen on toki aina toivottavaa, mutta sen luominen ja ylläpitäminen on kallista, ja lopputuloksen ennustaminen on usein hyvin hankalaa. Tämän vuoksi resurssien varaaminen API-dokumentaation parantamista varten ei ole niin itsestään selvää. (Uddin & Robillard, 2015.)

Uddin ja Robillard (2015) nostavat tutkimuksessaan 10 yleisintä ongelmaa, mitä tulee API-dokumentaatioon. Nämä ongelmat jakautuvat kahteen osa-alueeseen, jotka ovat dokumentaation sisältö ja esillepano. Nämä 10 ongelmaa on lueteltu taulukossa 2.

TAULUKKO 2 API-dokumentaation ongelmat (Uddin & Robillard, 2015)

Kategoria	**Ongelma**
Sisältö (Content)	Keskeneräisyys (Incompleteness)
	Moniselitteisyys (Ambiguity)
	Selittämättömät esimerkit (Unexplained examples)
	Vanhentuneisuus (Obsoleteness)
	Epäjohdonmukaisuus (Inconsistency)
	Virheellisyys (Incorrectness)
Esillepano (Presentation)	Paisuminen (Bloat)
	Pirstaloituminen (Fragmentation)

Liika rakenteellinen tieto (Excess structurual information)
Sekava, sotkuinen tieto (Tangled information)

4.3.1 Sisältö

Sisältöön liittyvä ensimmäinen ongelmakohta on dokumentaation keskeneräisyys. Tällä tarkoitetaan dokumentaatiota, jossa on selvästi puuttuvia osa-alueita. Toisena kohtana nousee moniselitteisyys, jolla tarkoitetaan dokumentaation monitulkintaisuutta, mikä aiheuttaa hämmennystä ja ymmärtämättömyyttä dokumentaatiota kohtaan. (Uddin & Robillard, 2015.)

Kolmas ongelma liittyy selittämättömiin esimerkkeihin. Kehittäjät arvostavat lähtökohtaisesti koodiesimerkkejä, mutta riittävän tarkkojen selitysten puuttuminen voi aiheuttaa turhaa turhautumista. Neljäntenä kohtana Uddin ja Robillard (2015) mainitsevat dokumentaation vanhentuneisuuden. API:en kehitys on usein hyvin nopeatempoista, minkä vuoksi dokumentaatio voi vanhentua nopeasti. Tällöin sen hetkinen dokumentaatio ei vastaa tehtyjä muutoksia, mikä voi aiheuttaa hämmennystä ja ongelmia kehitystyössä.

Viides sisällöllinen dokumentaatio-ongelma liittyy epäjohdonmukaisuuteen. API-dokumentaatio voi olla usein monen ihmisen tai monen tiimin tekemä, minkä vuoksi on olemassa mahdollisuus, että sen osa-alueet eivät keskustele keskenään oikein. Johdonmukaisuuden puuttuminen tarkoittaa usein myös järjestelmien yhteen toimivuuden puuttumista, mikä on ohjelmistokehityksessä erittäin huono asia. Tämä nivoutuu yhteen kuudennen kohdan, virheellisyyden, kanssa. (Uddin & Robillard, 2015.)

4.3.2 Esillepano

API-dokumentaation esillepanon ensimmäinen ongelma on sen paisuminen. Tällä tarkoitetaan dokumentaation kasvamista niin suureksi, että sitä lukiessa on vaikea erottaa, mikä dokumentaatiossa on oikeasti tarpeellista tietoa. Toisena kohtana Uddin ja Robillard (2015) nostavat dokumentaation pirstaloitumisen. Tällä viitataan APIn osien hajautumiseen dokumentaation liian monille alasivuille. Pirstaloituminen voi vaikeuttavan dokumentaation selaamista ja näin asiaankuuluvan tiedon etsiminen hankaloituu.

Esillepanon kolmas ja neljäs ongelma liittyvät molemmat dokumentaation varsinaiseen sisältöön. Dokumentaatio koetaan huonoksi, mikäli siinä on liikaa tarpeetonta rakenteellista tietoa ja, mikäli haluttu dokumentaation tieto sotkeutuu yhteen tarpeettoman tiedon kanssa. Liika ja sotkuinen informaatio voi hidastaa kehittäjien työtä merkittävästi. (Uddin & Robillard, 2015.)

4.3.3 Koodiesimerkit

Varsinaisen dokumentaation lisäksi ohjelmistokehittäjät kokevat koodiesimerkit tarpeellisiksi. Robillardin (2009) tutkimuksen mukaan 55 % vastaajista kertoo

oppivansa uuden APIn käytön koodiesimerkkien avulla. Esimerkkien puuttuminen nostettiin isoksi esteeksi APIn oppimisen suhteen.

Robillard (2009) jakaa koodiesimerkit kolmeen kategoriaan: katkelmiin (snippets), tutoriaaleihin (tutorials) ja valmiisiin ohjelmiin (applications). Katkelmien tehtävä on demonstroida, miten APIn perustoiminnallisuuksiin päästään käsiksi ja ne ovatkin pituudeltaan melko lyhyitä. Tutoriaalit ovat puolestaan tyypillisesti pidempiä ja ne muodostuvat useista koodien osista. Tutoriaalien lähtökohtaisena tehtävänä on opettaa ohjelmoijalle APIn tietty osa-alue. Valmiit ohjelmat ovat nimensä mukaisesti koodiesimerkkejä valmiista toimivista ohjelmista. Nämä ohjelmat pitävät sisällään APIn esittelynäytteet ja avoimen lähdekoodin projektit, joita kehittäjät voivat ladata eri lähdekoodilähteistä.

Joskus esimerkit voivat olla enemmän haitaksi kuin hyödyksi, mikäli esimerkin tarkoitus ja käyttäjän tavoite eivät kohtaa. Yleisimmät ongelmat esimerkkien suhteen liittyvät esimerkkien käyttäjiin, jotka haluavat käyttää koodikatkelmia APIn peruskäyttöä ylittäviin tarkoituksiin. (Robillard, 2009.)

5 EMPIIRINEN TUTKIMUS

Tutkimuksen tarkoituksena oli selvittää, kuinka verkkokauppojen tekniset toteuttajat kokevat Paytrailin teknisen dokumentaation tukevat heidän kehittäjäkokemustansa. Lisäksi haluttiin selvittää, miten dokumentaatio voisi olla entistä parempi kehittäjäkokemuksen paranemisen näkökulmasta. Tutkimus toteutettiin määrällisenä verkkokyselylomaketutkimuksena. Kyselylomake lähetettiin Paytrailin sähköpostilistalla oleville verkkokauppojen teknisille toteuttajille eli niille, jotka oletettavasti ovat olleet tai ovat tekemisissä Paytrailin teknisen dokumentaation kanssa. Paytraililla näitä teknisiä toteuttajia kutsutaan kumppaneiksi. Kyselylomake lähetettiin ainoastaan suomenkielisille kumppaneille, koska lomakkeen kieleksi valikoitui suomi.

Tässä luvussa käsitellään valittu tutkimusmenetelmä ja käydään läpi, minkä vuoksi se valikoitui tähän tutkimukseen. Ensimmäisessä alaluvussa käydään läpi tutkimuskysymykset, toisessa alaluvussa tutkimusmenetelmä. Kolmannessa alaluvussa puolestaan perehdytään tutkimuksen toteutukseen ja neljännessä alaluvussa kerrotaan aineiston purkamisesta ja analysoinnista, ja käydään läpi analysointimenetelmät.

5.1 Tutkimuskysymykset

Tutkimuksen tutkimuskysymykset ovat seuraavat:

- Miten verkkokauppojen tekniset toteuttajat kokevat kehittäjäkokemuksen toteutumisen toimeksiantajan teknisen dokumentaation avulla?
- Miten dokumentaatiota voisi parantaa, jotta kehittäjäkokemus olisi entistä parempi.

Kysymysten avulla halutaan saada selville Paytrailin yhteistyökumppanien tämänhetkinen suhtautuminen tekniseen dokumentaatioon kehittäjäkokemuksen kontekstissa. Paytrail verkkomaksamisen markkinajohtaja

Suomessa. Se tarjoaa verkkomaksamisen ratkaisuja pääsääntöisesti suomalaisiin verkkokauppoihin yhden sopimuksen avulla. Paytrailin yhteyskumppaneilla tarkoitetaan tässä yhteydessä eri verkkokauppojen rakentajia tai niin sanotusti teknisiä toteuttajia. Lisäksi halutaan kysyä mahdollisista kehitysideoista dokumentaation osalta. Näiden kehitysideoiden on tarkoitus viedä kumppanien kehittäjäkokemusta positiivisempaan suuntaa.

5.2 Tutkimusmenetelmä

Kvantitatiivinen tutkimus pohjautuu aina ennen kaikkea teoriaan, joka selittää tutkittavaa ilmiötä. Toisin kun laadullisessa tutkimuksessa, oletuksena on, että ilmiö tunnetaan. Teoriat, jotka selittävät tutkittavaa ilmiötä, mahdollistavat yksityiskohtaisten kysymysten tekemisen. Nämä kysymykset muodostavat määrällisen tutkimuksen tiedonkeruumenetelmän, jota kutsutaan kyselyksi. Tämän kyselylomakkeen avulla määrällinen tutkimus tuottaa määrällistä tietoa ilmiön muuttujien määristä ja näiden välisistä suhteista. (Kananen, 2014.)

Kvantitatiivisen tutkimuksen pyrkimyksenä on yleistää tutkimustulokset otoksen avulla. Tarkoituksena on siis teettää tutkimus pienehköllä joukolla asianomaisia ja vetää tästä johtopäätökset koko olemassa olevan populaation eli kohderyhmän osalta. (Kananen, 2015.)

Vain tietynlaisten ilmiöiden tutkimista voidaan toteuttaa kvantitatiivisen tutkimuksen keinoin. Kvantitatiivinen tutkimus ei esimerkiksi sovellu prosessien tutkimiseen, kvalitatiivinen tutkimus on tähän parempi vaihtoehto. Määrällinen tutkimus onkin aina tutkijalähtöistä. Tällä tarkoitetaan sitä, että se rakennetaan tutkijan tarpeisiin ja tutkijan ehdoilla. Ilmiö voi näyttäytyä vastaajan kannalta hyvin eri tavalla. (Kananen, 2015.)

Tämä tutkimus toteutettiin määrällisenä verkkokyselylomaketutkimuksena. Kyselylomaketta pidetään tavallisimpana määrällisessä tutkimusmenetelmässä käytettynä aineiston keräämisen tapana (Vilkka, 2021). Tutkimuksen tarkoitus oli saada selville Paytrailin teknisten yhteistyökumppanien, eli kumppanien, suhtautuminen Paytrailin tekniseen dokumentaatioon, kuinka hyvin he kokevat kehittäjäkokemuksen toteutuvan ja miten dokumentaatiota voisi parantaa paremman kehittäjäkokemuksen osalta.

Kyselytutkimus valikoitu tutkimustyypiksi, koska tutkimuksen tarkoituksena oli selvittää mahdollisimman laajasti kumppanien suhtautumista tekniseen dokumentaatioon kehittäjäkokemuksen kontekstissa. Tutkimus olisi voitu myös toteuttaa laadullisena haastattelututkimuksena. Kuitenkin, koska toimeksiantajalla oli valmis sähköpostilista kumppaneista, päätettiin tutkimus tehdä määrällisenä kyselytutkimuksena. Laajan sähköpostilistan avulla kysely voitiin lähettää lukuisille potentiaalisille vastaajille.

Kuten kaikilla tutkimustyypeillä, myös kyselytutkimuksella on hyviä ja huonoja puolia. Evans ja Mathur (2005) listaavat seuraavat kohdat verkkokyselyn eduiksi:

- Globaalius
- B2b- ja b2c- sopivuus
- Joustavuus
- Nopeus ja ajantasaisuus
- Helppous
- Matala kulurakenne
- Otannan kohdentaminen/kontrollointi
- Vastaamisen ohjaaminen/kontrollointi
- Vastaajan tunnistaminen

Rajoitteista ja haasteista puhuttaessa Evans ja Mathur (2005) mainitsevat oheiset seikat:

- Mahdollisuus roskapostiin joutumiseen
- Otoksen/tulosten vinoutumat
- Vastaajien osaamistaso
- Epäselvä ohjeistus
- Alhainen vastausprosentti
- Tekniset ongelmat

Kananen (2014) toteaa, että osoitetiedot, joita käytetään tutkimuksessa ja, jotka on kerätty muissa yhteyksissä, eivät ole tilastollisesti edustavia. Tällä hän viittaa esimerkiksi kerättyyn henkilörekisteriin. Tähän hän kuitenkin lisää, että edustavuus paranee, mikäli voidaan osoittaa valittujen havaintoyksiköiden liittyvän tutkittavaan ilmiöön tavalla tai toisella. Tämä toteutuu tutkielman tutkimuksen osalta, sillä käytössä olevan sähköpostilistan osoitteet kuuluvat lähtökohtaisesti verkkokauppojen teknisille toteuttajille. Näin ollen havaintoyksiköt liittyvät vahvasti tutkittavaan ilmiöön. Vilkka (2021) vahvistaa, että kyselylomaketutkimus soveltuu hyvin suurelle ja hajallaan olevalle joukolle. Kananen (2015) jatkaa, että kyselytutkimuksen etuina ovat suhteellisen nopea aineiston keruu, anonyymisyys ja melko pienet kustannukset. Haittapuolina hän listaa joustamattomuuden, alhaisen vastaamishalukkuuden ja kysymysten mahdolliset virheet. Toisin kuin haastatteluissa, virheitä ei voi korjata aineistonkeruun aikana.

Tilastoyksiköllä eli havaintoyksiköllä tarkoitetaan tutkittavaa kohdetta, joka voi olla esimerkiksi ihminen. Havaintoyksiköistä muodostetaan otos. Perusjoukko on tutkimuksessa määritelty ihmisjoukko ja se sisältää kaikki havaintoyksiköt, joista halutaan tietoa. Tutkimukseen voidaan valita perusjoukosta joko koko havaintoyksiköt tai tästä joukosta voidaan tehdä edustava otos. Otantamenetelmiä on monia kuten, kokonaisotanta, yksinkertainen satunnaisotanta, systemaattinen otanta, ryväsotanta ja ositettu otanta, joten tutkimuksen osalta onkin tärkeä päättää, mitä otantaa halutaan käyttää. (Vilkka, 2021.)

Tähän tutkimuksen otannaksi valikoitui kokonaisotanta. Kokonaisotannalla tarkoitetaan sitä, että koko perusjoukko otetaan mukaan

tutkimukseen. Kokonaisotanta valittiin, koska kyselyyn haluttiin vastauksia mahdollisimman monelta Paytrailin kumppanilta. Paytrailin suomenkieliset kumppanit kuuluivat siis otokseen.

5.3 Tutkimuksen toteutus

Tutkimuksen onnistumisen kannalta keskeistä on hyvin laadittu kyselylomake. Kun tutkimuksen aihe alkaa olemaan selvillä, seuraa syväluotaava perehtyminen kyseiseen aiheeseen. Näin saadaan selville, mitä aiheeseen liittyvää on jo tutkittu, minkälaisilla menetelmillä, millaisella otoksella ja minkälaisilla mittareilla. (Valli & Aaltola, 2015.) Vilkka (2021) väittää, että mittarin eli kyselylomakkeen suunnittelun olevan määrällisen tutkimuksen tärkein osa-alue.

Mittaaminen tapahtuu kvantitatiivisessa tutkimuksessa kysymyksillä ja vastausvaihtoehdoilla, ja näiden pitää olla oikein kohdennettuja. Lisähaastetta mittaamiseen tekee se, että vastaajat voivat ymmärtää kysymyksen kukin tavallaan. Kysymys voi mitata sitä, mitä sen kuuluukin mitata ja se voi olla muotoiltu täysin oikein, mutta vastausvaihtoehdot voivat olla vääriä. (Kananen, 2014.)

Käsitteet muodostavat määrällisen tutkimuksen. Jotta mittaaminen on mahdollista, tutkimuksessa olevien käsitteiden pitää olla mitattavissa. Käsitteellä täytyy olla indikaattoreita, joiden tehtävä on kertoa, millä käsite määritellään. Indikaattori osoittaa siis sen, millä käsitettä mitataan. (Kananen, 2014.) Tässä tutkimuksessa käsitellään vastaajien mielipiteitä teknisen dokumentaation osalta ja mielipidettä mitataan samaa ja eri mieltä olevalla asteikolla.

Kysymyksiin voi olla olemassa valmiit vastausvaihtoehdot, jolloin puhutaan strukturoiduista kysymyksistä, tai kysymykset voivat olla myös täysin avoimia. (Kananen, 2014.) Kysymysten standardisoinnilla tavoitellaan kysymysten vertailukelpoisuutta. Standardisoidut kysymykset ovatkin aina kompromissi arkikielen moniselitteisyyden, systemaattisuusvaatimusten ja mittausten tarkkuuden välillä. Avointen kysymysten tarkoituksena on puolestaan saada vastaajilta avoimia, spontaaneja mielipiteitä. (Vilkka, 2021.) Avoimet kysymykset eivät siis kahlitse vastaajaa valmiiksi rakennettuihin vastausvaihtoehtoihin. Toisaalta avointen kysymysten avulla tuotettu aineisto voi olla luotettavuudeltaan kyseenalaista ja todella kirjavaa, jonka käsitteleminen voi osoittautua hankalaksi. (Hirsjärvi, Remes & Sajavaara, 2009.)

Näiden lisäksi on olemassa myös niin sanotut sekamuotoiset kysymykset. Näissä kysymyksissä osa vastausvaihtoehdoista on annettu. Tämä kysymysvaihtoehto on varteenotettava, mikäli on syytä epäillä, että vastaajat eivät varmuudella tunne kaikkia vastausvaihtoehtoja. (Vilkka, 2021.) Tämän tutkimuksen kyselyssä väittämät ovat pääsääntöisesti strukturoituja, mutta mukana on myös muutama avoin kysymys.

Tutkimuksen aineisto kerättiin tutkimusta varten luodulla tutkimuskyselyllä, joka tehtiin Surveypal-palvelussa. Aineisto kerättiin kahden viikon aikana syyskuussa 2022. Kyselylomake lähetettiin yhdessä saatekirjeen

kanssa Paytrailin verkkokauppa-asiakkaiden teknisille toteuttajille eli niin sanotuille kumppaneille. Kyselyn vastaanottajiksi valikoituivat suomenkieliset kumppanit, jotka löytyivät Paytrailin sähköpostilistalta ja, jotka oletettavasti olivat olleet tekemisissä Paytrailin teknisen dokumentaation kanssa.

Lomakkeeseen saatiin melko vähän vastauksia, minkä vuoksi kutsu kyselyyn vastaamiseen lähetettiin potentiaalisille vastaajille kaksi kertaa uudestaan. Tämän lisäksi kysely jaettiin Paytrailin kumppaneille tarkoitetulla Slack-kanavalla. Paytrailin työntekijöitä kannustettiin tämän lisäksi ”markkinoimaan” kyselyä kumppaneille. Saatekirje ja kyselylomake löytyvät tutkielman liitteistä 1 ja 2.

5.3.1 Kyselylomake

Varsinainen kyselylomake tehtiin tutkitun teorian pohjalta, joka sisälsi tutustumista lean-ajatteluun, kehittäjäkokemukseen eri konteksteissa, inhimillisiin tekijöihin ohjelmistokehityksessä ja tekniseen dokumentaatioon. Näiden lisäksi myös API-kehitykseen liittyviin osa-alueisiin tutustuttiin tarkasti. Pääasiallisena teoriana toimi Uddinin ja Robillardin (2015) tutkimus liittyen API-dokumentaation ongelmiin sekä Fagerholmin ja Münchin (2012) luoma viitekehys kehittäjäkokemus termille. Näiden artikkelien pohjalta nousseista ongelmista tehtiin väittämät, joiden esiintyvyyttä tai toteutumista kysyttiin kyselyssä.

Heikkilä (2014) toteaa kohderyhmän tuntemisen, kysymysten oikein muotoilun ja yksiselitteisyyden sekä ymmärrettävyyden olevan tärkeitä osa-alueita lomaketta luodessa. Nämä kaikki seikat otettiin huomioon, kun lomaketta suunniteltiin. Ymmärrettävyyttä ja yksiselitteisyyttä pyrittiin lisäämään lisäämällä ennen jokaista väittämää tarkentava selitys, mitä väittämän termillä tarkoitetaan tässä kontekstissa. Esimerkiksi ennen väittämää ”Paytrailin dokumentaatio on vanhentunutta” lisättiin dokumentaation vanhentumista selittävä kappale: ”Dokumentaation vanhentuneisuudella tarkoitetaan sitä, että dokumentaatio ei pysy varsinaisen kehityksen perässä, ja näin ollen dokumentaatio ei vastaa palvelua.”

Kyselylomake luotiin olemassa olevan teorian pohjalta ja se käytiin läpi toimeksiantajan työntekijän kanssa. Toimeksiantajalta saatujen vinkkien ja korjausehdotusten jälkeen lomakkeeseen tehtiin muutamia muutoksia, jonka jälkeen se lähetettiin vielä pro gradun ohjaajalle arvioitavaksi ja testattavaksi. Testaamisen ja ohjaajalta saatujen korjausehdotusten jälkeen lomake saatiin jäsenneltyä lähetysvalmiiksi ja toimitettiin eteenpäin toimeksiantajalle, joka vastasi kyselyn lähettämisestä omien järjestelmiensä kautta.

Tulosten luotettavuuden kannalta suurin ongelma liittyy alhaiseen vastausprosenttiin. Tätä vastausprosenttia voidaan kuitenkin pyrkiä kasvattamaan pienillä parannuksilla. Hooley, Marriot ja Wellens (2013) nostavat erityisesti oikean kohderyhmän valinnan, sähköpostin ja itse kyselyn napakkuuden, ohjeistuksen sekä karhuviestit tärkeiksi tekijöiksi. Myös mahdollisen palkkion, viestin personoinnin ja kyselyn lähettämisen ajoittaminen

aamuun pitäisi nostaa vastausprosenttia. Kyselyä rakennettaessa, kaikki nämä kohdat pyrittiin ottamaan huomioon.

5.3.2 Kyselyn sisältö

Kyselyn väittämien oikeellisuutta kysyttiin asteikolla 1–5. Vastaus ”täysin samaa mieltä” oli arvoltaan 1 ja vastaus ”täysin eri mieltä” oli arvoltaan 5. Muut arvot osuivat tämän asteikon välille.

Kyseessä on Kanasen (2014) mukaan niin sanottu 5-portainen asteikko. Asteikko rakentuu niin, että jokaisen asteikon portaan etäisyys on yhtä pitkä. Yleensä asteikkoväli on 1, mutta se on mahdollista asettaa miksi tahansa luvuksi, kunhan kunkin asteen ero on yhtä suuri. Asteikosta käytetään myös nimitystä Likertin asteikko (Hirsjärvi, Remes & Sajavaara, 2009). Kananen (2014) lisää tähän vielä kohdan ”ei kantaa asiaan”, mutta tässä tutkimuksessa se on otettu kokonaan pois. Taulukossa 3 näkyy tutkimuksessa käytetty 5-portainen asteikko.

TAULUKKO 3 5-portainen asteikko

Täysin samaa mieltä 1	Jokseenkin samaa mieltä 2	Siltä väliltä (neutraali) 3	Jokseenkin eri mieltä 4	Täysin eri mieltä 5

Vastaajien tuli vastata strukturoituihin väittämiin edellä mainitun 5-portaisen asteikon mukaan. Näin ollen mitä pienemmän numeron vastaaja antoi, sitä enemmän samaa mieltä hän oli väittämän kanssa. Tämän lisäksi väittämien perässä oli avoin kenttä ”Esimerkkejä/mietteitä teemasta”, jonka tehtävänä oli kannustaa vastaajia selittämään heidän vastaustaan.

Taulukossa 4 löytyvät tutkimuksessa esiintyneet väittämät, jotka liittyivät teknisen dokumentaation sisällöllisiin kohtiin. Teeman väittämät on nimetty lyhenteellä SIS, jotta niihin viittaaminen myöhemmin olisi kätevämpää.

TAULUKKO 4 Sisällölliset väittämät

	Väittämä	**Lähde**
SIS1.	Paytrailin dokumentaatiossa esiintyy keskeneräisyyttä	(Uddin & Robillard, 2015)
SIS2.	Paytrailin dokumentaatiossa esiintyy moniselitteisyyttä	(Uddin & Robillard, 2015
SIS3.	Paytrailin dokumentaatiossa esiintyy selittämättömiä koodiesimerkkejä	(Robillard, 2009, Uddin & Robillard, 2015)
SIS4.	Paytrailin dokumentaatiossa esiintyy liian vähän koodiesimerkkejä	(Robillard, 2009, Uddin & Robillard, 2015)
SIS5.	Paytrailin dokumentaatio on vanhentunutta	(Uddin & Robillard, 2015, Moilanen ym. 2018)
SIS6.	Paytrailin dokumentaatiossa esiintyy epäjohdonmukaisuutta	(Uddin & Robillard, 2015)

SIS7.	Paytrailin dokumentaatiossa esiintyy asia- ja/tai sisältövirheitä	(Uddin & Robillard, 2015)

Taulukossa 5 löytyvät puolestaan ne neljä väittämää, jotka liittyivät teknisen dokumentaation esillepanollisiin kohtiin. Teeman väittämät on nimetty lyhenteellä ESI.

TAULUKKO 5 Esillepanolliset väittämät

	Väittämä	Lähde
ESI1.	Paytrailin dokumentaatiossa ilmenee paisumista	(Uddin & Robillard, 2015)
ESI2.	Paytrailin dokumentaatiossa ilmenee pirstaloitumista	(Uddin & Robillard, 2015
ESI3.	Paytrailin dokumentaatio sisältää liiallista rakenteellista tietoa	(Uddin & Robillard, 2015)
ESI4.	Paytrailin dokumentaatio on sekavaa	(Uddin & Robillard, 2015)

Sisällöllisten ja esillepanollisten väittämien lisäksi kyselyyn päätettiin lisätä väittämä liittyen kehittäjäkokemukseen, sillä kehittäjäkokemus on tutkielman yksi pääteemoista. Aluksi väittämäksi mietittiin ”Kuinka hyvin koet kehittäjäkokemuksen toteutuvan Paytrailin teknisessä dokumentaatiossa?”. Tästä kuitenkin luovuttiin, sillä katsoimme yhdessä tutkielman ohjaajan kanssa, että vastaajalla täytyisi olla ymmärrys ja näkemys kehittäjäkokemuksesta. Väittämän ei haluttu muotoutuvan potentiaalisen vastaajan päässä koskemaan dokumentaation yleistä laatua. Väittämä päätettiin muokata niin, että se ei kytkeydy suoraan tekniseen dokumentaatioon vaan arviota kysytään yleisesti kehittäjäkokemuksen toteutumisesta liittyen Paytrailin palvelun toteutukseen vastaajan organisaatiossa. Tällä tavalla mahdollista korrelaatiota voidaan hakea itse dokumentaatiota koskevien vastausten ja kehittäjäkokemusvastausten välillä. Väittämään vastattiin 5-portaisen asteikon mukaan.

Toinen kehittäjäkokemukseen liittyvä kohta jätettiin kyselyn loppuun avoimena tekstikysymyksenä, kuten Heikkilä (2014) ohjeistaa. Kohdan tarkoituksena oli kysyä vastaajilta, miten tekninen dokumentaatio voisi olla parempi ja miten sitä voisi kehittää tulevaisuudessa. Taulukossa 6 löytyvät kehittäjäkokemukseen liittyvät kyselyn osa-alueet. Teeman väittämät on nimetty lyhenteellä KEH.

TAULUKKO 6 Kehittäjäkokemuksen väittämät

	Väittämä tai kysymys	Lähde
KEH1.	Kehittäjäkokemus on puutteellinen/vajavainen kehittäessä Paytrailin maksupalvelua organisaatiolleni	(Fagerholm & Münch, 2012)
KEH2.	Miten kehittäisit/parantaisit Paytrailin teknistä dokumentaatiota?	(Oma)

Ohjelmoinnissa on käytössä nykypäivänä lukuisia koodikieliä/ohjelmointikieliä ja määrä nousee yhdessä ohjelmoijien määrän kanssa. Suosituimpien kielien listalle kuuluvat muun muassa Java, C, Python ja Java. (Krill, 2016.) Kyselyä varten luotiin yhdessä toimeksiantajan kanssa lista suosituimmista koodikielistä

ja kumppaneilta kysyttiin: ”Millä koodikielillä esimerkkien tulisi olla?”. Vastaajien oli mahdollista valita listasta yksi tai useampi vaihtoehto. Lisäksi vastaajien oli mahdollista vastata vaihtoehto ”jokin muu”. Taulukosta 7 löytyvät kyselytutkimuksessa esiintyneet koodikielet. Kyseessä on Hirsjärven ym. (2009) mukaan strukturoidun ja avoimen kysymyksen välimuoto. Valmiiden vastausvaihtoehtojen lisäksi valittavana oli avoin vaihtoehto. Tämän vaihtoehdon avulla pyrittiin selvittämään, onko olemassa muita potentiaalisia koodikieliä.

TAULUKKO 7 Kysytyt koodikielet

	Koodikieli
1.	C#
2.	PHP
3.	Java
4.	JavaScript
5.	Python
6.	Curl
7.	Ruby
8.	Go
9.	Jokin muu

5.4 Aineiston analysointi ja käsittely

Kyselyllä kerätyn aineiston avulla on tarkoitus ratkaista tutkimuksen tutkimusongelma. Varsinaiset tutkimuskysymykset on johdettu tutkimusongelmasta, ja näiden kysymysten taustalla ovat kvantitatiivisen tutkimuksen mallit, esiymmärrys kyseisestä ilmiöstä ja teoriat. Kyselylomakkeen kysymysten avulla kerätään aineisto kohderyhmältä, minkä pohjimmainen tarkoitus on ratkaista tutkimuskysymykset. Ratkaisun löytämiseksi hyödynnetään tilastollisia operaatioita, joilla lasketaan erilaisia tunnuslukuja, ja joista tehdään lopulta analyysejä. (Kananen, 2015.)

Lomakkeella kerätty aineisto käsitellään yleensä tilasto-ohjelmalla, jonka jälkeen aineisto tiivistetään. Tässä tutkimuksessa käytettiin SPSS-ohjelmaa. Kananen (2015) listaa yksinkertaisimmiksi ja yleisimmiksi tulosten esittämistavoiksi suorat jakaumat, ristiintaulukoinnit, erilaiset jakauma- ja tunnusluvut sekä avointen kysymysten tulosten esittämisen. Hieman vaativampien ja kehittyneiden analyysimenetelmien joukkoon nousevat korrelaatioanalyysi, multippeliregressioanalyysi, regressioanalyysi, faktorianalyysi sekä klusterianalyysi. Tässä tutkielmassa käytettiin yleisempiä ja yksinkertaisempia menetelmiä.

Aineiston käsittely aloitettiin sen tarkistuksella. Tämä on tärkeää tehdä mahdollisimman alussa, sillä virheelliset selitteet voivat käsittelyssä

virhetulkintoja. Tarkistus aloitetaan siitä, että tarkistetaan muuttujien arvojen ja nimien selitteiden vastaavan tietoja kyselylomakkeella. Tämän jälkeen tarkistetaan muuttujien arvot. Tehtiin tietojen syöttö tilasto-ohjelmaan sitten manuaalisesti tai import-työkalun avulla, tulisi tämä silti tehdä, sillä virheelliset tiedot vaikuttavat aina kyselyn tuloksiin. (Heikkilä, 2014.)

Suora jakauma on tapaa tiivistää ja esittää havaintoyksiköistä kerätty tieto. Suora jakauma on aineiston esittämistapa sekä yksinkertainen analyysikeino, jossa nähdään yksittäisen kysymyksen eri vaihtoehtojen saamat vastaukset. Vastaukset ilmoitetaan aina suhteellisina taulukoina eli toisin sanoen esityksessä käytetään prosenttilukuja. Sarakkeessa ilmoitetaan vastausten kokonaismäärä (N), josta sitten lasketaan prosentit. (Kananen, 2015.)

Ristiintaulukointi eroaa suorasta jakaumasta siinä, että siinä tarkastellaan samaan aikaisesti kahta kysymystä. Sen avulla pyritään selvittämään muuttujien ja ryhmien välisiä eroja riippuvuuksia. Muuttujista eli kysymyksistä käytetään nimitystä selittävä ja selitettävä muuttuja. Esittämistavassa selitettävä muuttuja on sarakemuuttuja ja selitettävä muuttuja rivimuuttuja. (Kananen, 2015.) Jos halutaan esimerkiksi selvittää, onko työkokemuksella vaikutusta mielipiteeseen aiheesta, asetetaan työkokemus sarakemuuttujaksi ja mielipidekysymys rivimuuttujaksi. Näin taulukosta voidaan nähdä, vaikuttaako työkokemuksen määrä mielipiteeseen tietystä asiasta.

Erilaisilla tunnusluvuilla voidaan esittää numeraalista tietoa siitä, miten esimerkiksi mielipiteet, tyytyväisyys tai asenteet vaihtelevat organisaation asiakkaiden välillä. Yksi tunnetuimmista tunnusluvuista on aritmeettinen keskiarvo. Siinä yhteenlaskettu tulos jaetaan havaintojen lukumäärällä. Luku soveltuu etenkin suhdelukutaulukkoon. Keskiarvo on hyvin altis poikkeaville havainnoille, minkä vuoksi se ei anna kovin tarkkaa kuvaa, mikäli aineistossa on yksikin hyvin suuri tai pieni arvo. Keskiarvon lisäksi on myös hyvä hyödyntää mediaania, moodia ja keskihajontaa. (Vilkka 2007.)

Mediaanilla tarkoitetaan keskilukua, joka ilmaisee jakauman keskimmäistä havaintoa. Kun kaikki muuttujan havainnot asetetaan suuruusjärjestykseen, mediaaniksi kutsutaan sitä havaintojen keskikohtaa, jonka molemmille puolille jää yhtä monta havaintoa. Mediaanin avulla voidaan saada selville, miten havainnot painottuvat keskimmäisen havainnon suhteen. Moodiksi puolestaan kutsutaan sitä keskilukua, jota esiintyy muuttujassa eniten. Näin ollen moodi siis kertoo sen luokan tai arvon, missä frekvenssi eli esiintymistiheys on suurin. Äärimmäiset havainnot eivät vaikuta moodiin, minkä vuoksi sen avulla voi usein päätellä esimerkiksi keskeisen vastaajaryhmän aritmeettista keskiarvoa paremmin. (Vilkka, 2007.)

Keskihajonnalla tarkoitetaan sitä lukua, joka ilmaisee, kuinka kaukana yksittäisen muuttujan arvot ovat keskimääräisen muuttujan arvosta. Keskihajonta kuvaa siis muuttujien etäisyyttä suhteessa aritmeettiseen keskiarvoon. Tämän vuoksi, kun käytetään keskihajontaa, tulee ilmaista myös keskiarvo. (Vilkka, 2007.)

Yksinkertaisin tapa avointen kysymysten käsittelyyn on tekstinkäsittelyohjelman avulla. Tässä avoimet vastaukset siirretään

tekstinkäsittelyohjelmaan, jossa teksti muutetaan taulukkomuotoon. Tämän jälkeen katsotaan vastauksen raakatekstiä ja kirjataan tämä yhdellä sanalla taulukon seuraavaan sarakkeeseen. Samaa tarkoittavat käsitteet nimetään yhdellä termillä seuraavaan sarakkeeseen. Tarkoitus on tiivistää vastaukset mahdollisimman vähiin termeihin ja käsitteisiin. Tiivistämisen jälkeen termit voidaan koodata numeroilla viimeiseen sarakkeeseen. Näin vastausten esiintyminen voidaan laskea eli ne luokitellaan. Luokittelun avulla nähdään, kuinka paljon kyselyn vastaajat vastasivat mitäkin avoimiin kysymyksiin. (Kananen, 2015.)

Tässä tutkimuksessa avoimien kysymysten vastaukset siirrettiin Excel-tiedostoon, jossa niiden käsittely oli helpompaa. Tämän jälkeen vastauksista tunnistetiin aihepiirejä, jotka toistuivat moneen otteeseen. Laadullisen aineiston teemoihin jakamisen jälkeen, kohtia tarkasteltiin vielä yksityiskohtaisemmin. Näitä esiin nousseita kohtia käsitellään tarkemmin tulosten esittelyn yhteydessä.

6 TULOKSET

Tässä luvussa esitellään tutkimuksesta saadut tulokset, joilla vastataan alussa asetettuihin tutkimuskysymyksiin. Luvun alussa keskitytään kuvaamaan vastaajien demografiset tiedot. Lopuissa alaluvuissa tulokset esitetään tutkimuksen teemojen mukaan, joita ovat dokumentaation sisältö, dokumentaation esillepano sekä kehittäjäkokemus.

6.1 Kyselyyn vastaajat

Kyselyyn saatiin koko sen vastausaikana yhteensä 19 vastausta, mikä on huomattavasti odotettua vähäisempi vastausmäärä. Vastaajien lukumäärää yritettiin nostaa kahdella karhukirjeellä sekä kehottamalla toimeksiantajan organisaation työntekijöitä viestimään kyselystä omissa työkanavissaan. Näitä kanavia olivat muun muassa Slack sekä sähköposti.

Kyselyyn vastaajista noin kaksi kolmasosaa (68,4 %) käyttää Paytrailia organisaatiossaan parhaillaan, kun taas loput (31,6 %) eivät. Työkokemukseltaan vastaajat olivat hyvin kokeneita. Eniten vastaajista omasi yli 10 vuoden työkokemuksen nykyisessä organisaatiossa (42,1 %). Noin kolmannes (36,8 %) oli puolestaan ollut töissä 7–11 vuotta ja reilu viidennes (21,1 %) 3–6 vuotta. Vastausvaihtoehtoon 0–2 ei tullut yhtään vastausta. Tämän vuoksi kyseinen vastausvaihtoehto on jätetty pois tulevista taulukoista.

Vastaajien organisaatioiden kokoluokista 1–10 oli ylivoimaisesti suosituin (78,9 %). Kymmenennes (10,5 %) vastaajista oli 51-100 henkilöä työllistäneestä organisaatiosta, kun taas 11-50 ja 100+ vaihtoehdot saivat kummatkin ainoastaan yhdet vastaukset (5,3 %). Työkokemuksen ja organisaation kokoluokan lisäksi kyselyssä tiedusteltiin vastaajan roolista organisaatiossa. Yli puolet (57,9 %) vastaajista kertoi olevansa kehittäjä, noin viidennes (21,1 %) johtaja-asemassa tai omistaja, ja vajaa viidennes (15,8 %) yrittäjä. Yksi vastaajista (5,3 %) kertoi

olevansa mainonnan suunnittelija. Vastaajien demografiset tiedot esitellään tarkemmin taulukossa 8.

TAULUKKO 8 Vastaajien demografiset tiedot (N=19)

Onko Paytrail käytössä organisaatiossasi?

	N	%
Kyllä	13	68,4
Ei	6	31,6
Yhteensä	19	100,0

Työkokemuksesi vuosina kyseisessä organisaatiossa

	N	%
3–6	7	36,8
7–11	4	21,1
10+	8	42,1
Yhteensä	19	100,0

Organisaation koko (henkilömäärä)

	N	%
1–10	15	78,9
11–50	1	5,3
51–100	2	10,5
100+	1	5,3
Yhteensä	19	100,0

Missä roolissa toimit organisaatiossa?

	N	%
Kehittäjä	11	57,9
Mainonnan suunnittelija	1	5,3
Johtaja (CEO, COO) / omistaja	4	21,1
Yrittäjä	3	15,8
Yhteensä	19	100,0

6.2 Dokumentaatio-väittämien tunnusluvut

Väittämien osalta vastaajilta kysyttiin, kuinka hyvin he kokevat väitteen toteutuvan Paytrailin dokumentaatiossa. Vastausvaihtoehto 1 tarkoittaa, että vastaaja on samaa mieltä väitteen kanssa, kun taas 5 tarkoittaa, että vastaaja on väitettä vastaan. Koska väittämät ovat pääsääntöisesti negatiivisia, mitä korkeamman numeron vastaajat antoivat, sitä paremmin he pitivät dokumentaatiosta. Tunnusluvut esitetään taulukossa 9. Taulukon luvut on avattu tarkemmin alaluvuissa.

TAULUKKO 9 Väittämien tunnusluvut

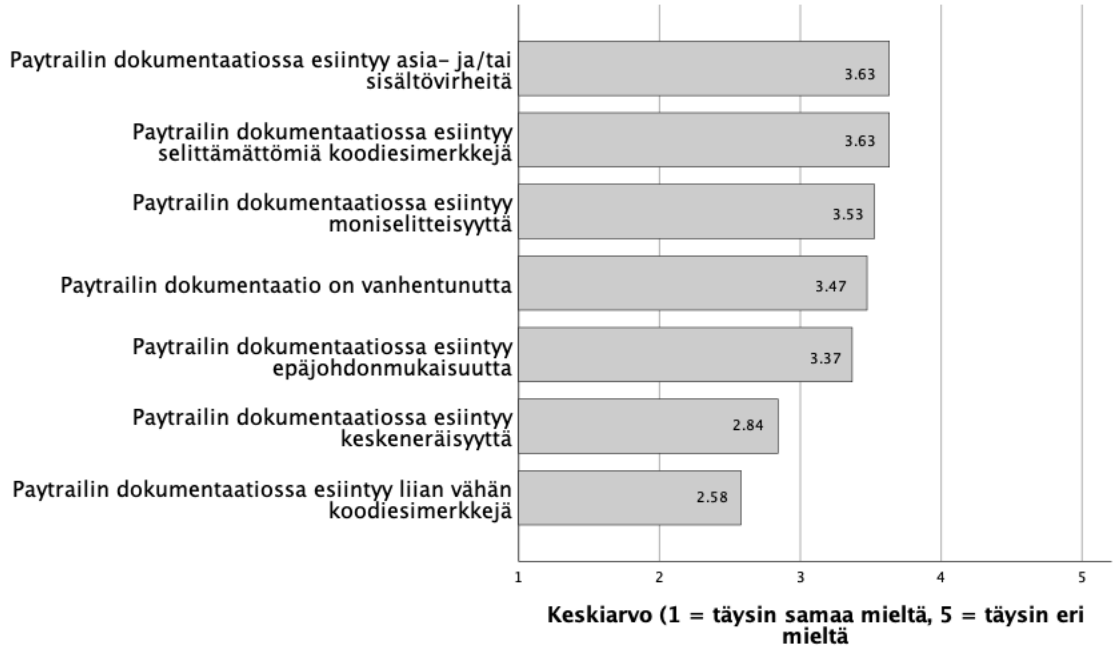
Väittämä	N	Min	Max	Med.	Ka.	Keskihajonta
SISÄLTÖ						
Paytrailin dokumentaatiossa esiintyy keskeneräisyyttä	19	2	4	3	2,84	0,90
Paytrailin dokumentaatiossa esiintyy moniselitteisyyttä	19	2	5	4	3,53	0,91
Paytrailin dokumentaatiossa esiintyy selittämättömiä koodiesimerkkejä	19	2	5	4	3,63	0,76
Paytrailin dokumentaatiossa esiintyy liian vähän koodiesimerkkejä	19	2	4	3	2,58	0,61
Paytrailin dokumentaatio on vanhentunutta	19	2	5	3	3,47	1,07
Paytrailin dokumentaatiossa esiintyy epäjohdonmukaisuutta	19	2	5	3	3,37	0,90
Paytrailin dokumentaatiossa esiintyy asia- ja/tai sisältövirheitä	19	1	5	4	3,63	1,12
ESILLEPANO						
Paytrailin dokumentaatiossa ilmenee paisumista	19	1	5	4	3,42	1,12
Paytrailin dokumentaatiossa ilmenee pirstaloitumista	19	1	5	3	3,58	1,22
Paytrailin dokumentaatio sisältää liiallista rakenteellista tietoa	19	2	5	4	3,74	0,87
Paytrailin dokumentaatio on sekavaa	19	1	5	3	3,16	1,07
KEHITTÄJÄKOKEMUS						

Kehittäjäkokemus on puutteellinen/vajavainen kehittäessä Paytrailin maksupalvelua organisaatiolleni	19	1	5	3	3,11	1,05

6.2.1 Sisällöllisten väittämien tunnusluvut

Korkeimman keskiarvon saivat väittämät liittyen selittämättömiin koodiesimerkkeihin ja asia- sekä sisältövirheisiin. Molempien mediaani oli 4 ja keskiarvo 3,63. Selittämättömien koodiesimerkkien keskihajonta oli puolestaan 0,76 kun asia- ja sisältövirheiden 1,12.

Seuraavaksi korkeimman keskiarvon vastaajien keskuudessa sai väittämä moniselitteisyydestä (3,53). Mediaani väittämällä oli 4 ja keskihajonta 0,91. Puolestaan väittämä dokumentaation vanhentumisesta sai keskiarvokseen 3,47 ja mediaaniksi 3. Keskihajonta väittämällä oli 1,07. Keskiarvoltaan hieman tämän



alle ylsi väittämä koskien epäjohdonmukaisuuden esiintymistä (3,37). Mediaani väittämällä oli 3 ja keskihajonta 0,90.

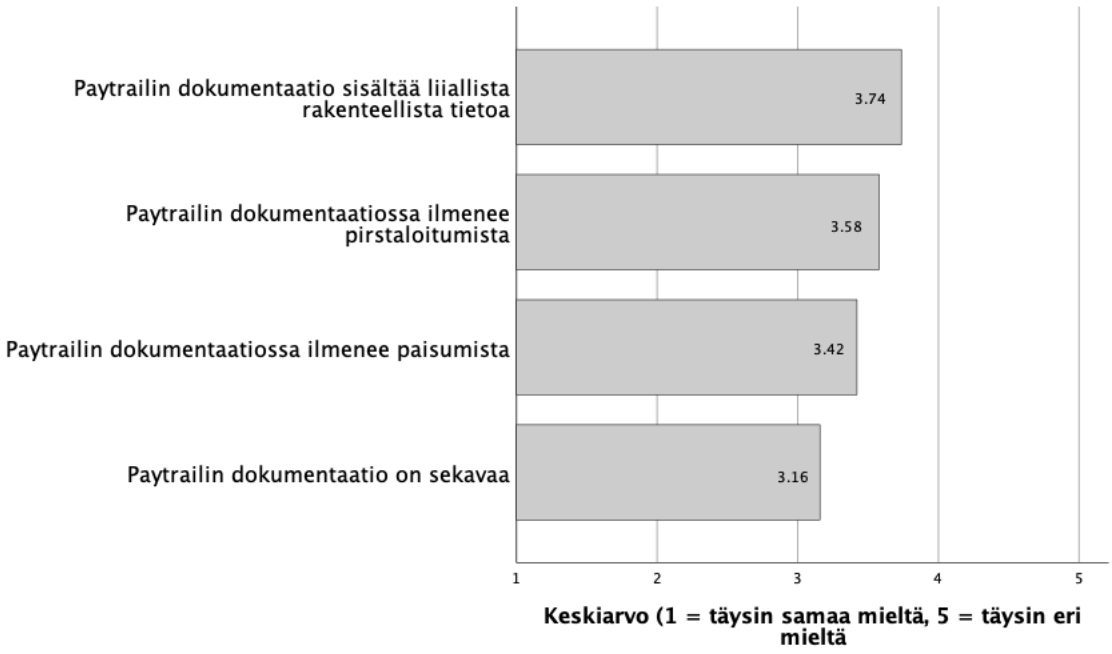
Selvästi heikoimmat keskiarvot saivat väitteet keskeneräisyydestä (2,84) ja liian vähistä koodiesimerkeistä (2,58). Molempien mediaani oli 3. Keskeneräisyysväittämän keskihajonta oli 0,90, kun taas liian vähien koodiesimerkkien väittämän keskihajonta oli kaikista pienin (0,61). Keskiarvot on vielä havainnollistettu kuviossa 3.

6.2.2 Esillepanollisten väittämien tunnusluvut

Esillepanollisten väittämien kanssa vastausten keskiarvot olivat lähtökohtaisesti hieman korkeammat kuin sisällöllisten väittämien, mutta keskihajonnat ovat korkeammat. Korkeimman eli parhaan keskiarvon väittämien joukosta sai väittämä koskien liiallista rakenteellista tietoa (3,74). Väittämän mediaani oli 4 ja keskihajonta 0,87. Seuraavaksi korkeamman keskiarvon eli 3,58 sai väittämä liittyen dokumentaation pirstaloitumiseen. Tämän väittämän keskihajonta oli kaikkien väittämien korkein (1,22). Mediaani puolestaan oli 3.

Dokumentaation paisumiseen liittyvä väittämä sai keskiarvokseen 3,42, mutta sen mediaaniksi tuli kuitenkin 4. Keskihajonta oli melko korkea, 1,12. Alhaisimman keskiarvon sai väittämä *Paytrailin dokumentaatio on sekavaa*. Keskiarvoksi väittämälle tuli 3,16 ja mediaaniksi 3. Keskihajonnaksi muodostui 1,07. Keskiarvot on vielä havainnollistettu kuviossa 4.

KUVIO 3 Sisällöllisten väittämien keskiarvot



KUVIO 4 Esillepanollisten väittämien keskiarvot

6.2.3 Kehittäjäkokemus-väittämän tunnusluvut

Kyselyssä oli yksi kehittäjäkokemusta suoraan Likert-asteikolle asettuva väittämä, *Kehittäjäkokemus on puutteellinen/vajavainen kehittäessä Paytrailin maksupalvelua organisaatiolleni*, jonka keskiarvoksi 1–5 asteikolla muodostui 3,11. Väittämän mediaani oli 3 ja keskihajonta 1,05.

6.3 Avoimet vastaukset

Likert-asteikko-vastausten lisäksi kyselyyn vastaajat saivat kommentoida väittämiä avoimien vastauskenttien avulla. Nämä avoimet kohdat eivät olleet pakollisia, mutta silti lähes jokaiseen niistä tuli ainakin muutama vastaus.

6.3.1 Sisällöllisten väittämien avoimet vastaukset

Eniten vastauksia määrällisesti tuli sisällöllisiin väittämiin. Kommentteja tuli eniten ensimmäiseen väittämään *Paytrailin dokumentaatiossa esiintyy keskeneräisyyttä,* kun taas väittämään *Paytrailin dokumentaatio on vanhentunutta* ei tullut yhtään kommenttia.

Paytrailin dokumentaatiossa esiintyy keskeneräisyyttä- väittämän osa vastauksista kommentoi yleisesti, kuinka keskeneräinen dokumentaatio hidastaa työskentelyä ja tekee siitä muutenkin sekavaa. Näiden kommenttien lisäksi vastaajat kommentoivat suoraan Paytrailin dokumentaation keskeneräisyyttä. Tärkeimpänä kohtana nousi ominaisuuksien ja termien parempi kommentointi ja tarkempi avaaminen. Näiden lisäksi jotkin esimerkit, demojen linkit sekä korttimaksujen paluukutsujen kuvaus koettiin keskeneräisiksi.

Seuraavaan väittämään *Paytrailin dokumentaatiossa esiintyy moniselitteisyyttä* tuli kaksi avointa vastausta. Molemmat kommentoivat sitä, että osa dokumentaation kohdista on hieman vaikeasti ymmärrettävissä. Dokumentaation rakenteen kannalta sen ominaisuudet eivät ole eritelty tarpeeksi hyvin. Esimerkiksi maksupalvelun normaalin toteutuksen ja niin sanotun Shop-in-shop-toteutuksen välillä ei ole yhden vastaajan mukaan tarpeeksi erittelyä. Toinen vastaaja nostaa epäselvyyden johtuvan koodikielestä.

Seuraaville väittämille *Paytrailin dokumentaatiossa esiintyy selittämättömiä koodiesimerkkejä* ja *Paytrailin dokumentaatiossa esiintyy liian vähän koodiesimerkkejä,* oli sama avoimen kommentoinnin vastauskohta, koska väittämät nivoutuvat teemaltaan toisiinsa. Toistuva vastaus tähän kohtaan lähes jokaisessa kommentissa oli tarve koodiesimerkeille useammilla kielillä. Vastauksissa pyydettiin esimerkkejä curl-komentoina, postman-esimerkkejä sekä laajempia PHP-esimerkkejä. Yksi vastaaja kaipasi myös SDK:ta (software development kit) eli ohjelmistokehityspakettia.

Kahteen viimeiseen väittämään tuli yksi vastaus väittämää kohden. Yhdellä vastaajalla oli väittämässä *Paytrailin dokumentaatiossa esiintyy epäjohdonmukaisuutta* vaikea ymmärtää Shop-in-shop-maksamisen ja normaalin

maksamisen eroja, ja sitä, mitkä ominaisuudet toimivat kussakin maksamisessa. Väittämää *Paytrailin dokumentaatiossa esiintyy asia- ja/tai sisältövirheitä* puolestaan kommentoitiin, että request-id ei palautuisi kaikille pyynnöille oikein.

6.3.2 Esillepanollisten väittämien avoimet vastaukset

Kuten sisällöllisten väittämien avointen vastausten kanssa, myös esillepanollisten väittämien avoimet kohdat eivät olleen pakollisia. Silti näihin kohtiin tuli muutamia vastauksia. Väittämä, *Paytrailin dokumentaatio on sekavaa,* sai eniten vastauksia, kun taas väittämät *Paytrailin dokumentaatiossa ilmenee pirstaloitumista* ja *Paytrailin dokumentaatio sisältää liiallista rakenteellista tietoa* saivat molemmat yhden kommentin.

Väittämän *Paytrailin dokumentaatio on sekavaa,* osalta vastaajat kaipasivat dokumentaatiolta parempaa rakennetta, kuten esimerkkien, kohteiden ja peräkkäin suoritettavien pyyntöjen lähekkäisyyttä. Myös maksutapojen erojen kuvaus aiheutti yhdelle vastaajalle sekavuutta dokumentaatiossa. Tämän lisäksi yksi vastaaja kommentoi, että hän koki rajapintatoteutuksen yleisesti hankalaksi.

Paytrailin dokumentaatiossa ilmenee paisumista-väittämä jätti vastaajat puolestaan kaipaamaan parempaa jäsentelyä sekä palvelukuvauksen liittämistä ominaisuuksiin. Näin ollen paremman rakenteen tarve toistui myös tämän väittämän kanssa.

Väittämät *Paytrailin dokumentaatiossa ilmenee pirstaloitumista ja Paytrailin dokumentaatio sisältää liiallista rakenteellista tietoa* ja, saivat lähinnä teemaa kommentoivia vastauksia. Yksi vastaaja kommentoi, että dokumentaation pirstaloituminen vaikuttaa sen selkeyteen, kun taas liiallisen rakenteellisen tiedon väittämä sai yleislaatuisen kommentin siitä, kuinka vahvasti tyypitetyllä kielellä selkeä ja tarkka rakenne auttaa työskentelyssä.

6.3.3 Kehittäjäkokemus-väittämien avoimet vastaukset

Väittämä *Kehittäjäkokemus on puutteellinen/vajavainen kehittäessä Paytrailin maksupalvelua organisaatiolleni* keräsi yhteensä neljä avointa vastausta tai kommenttia. Kaksi näistä vastauksista kommentoi sitä, että Paytrailin dokumentaation kehittäjäkokemus on hyvällä tasolla ja, että dokumentaatio koetaan selkeäksi. Loput vastaukset liittyivät siihen, miten kehittäjäkokemus voisi olla parempi. Vastauksissa toivottiin lisää esimerkkejä ja jo edellä mainittuja SDK:ta. Näiden lisäksi kritiikkiä sai dokumentaation vieraskielisyys. Vastaaja ihmetteli vieraskielisyyttä, vaikka dokumentaation kuvaama rajapinta on ensisijaisesti tarkoitettu suomalaiseen käyttöön.

Likert-asteikolle asettuvan väittämän lisäksi vastaajilta kysyttiin kehittäjäkokemus-teemasta suoraan: *Miten kehittäisit/parantaisit Paytrailin teknistä dokumentaatiota?* Kysymys oli vastaajille pakollinen, joten siihen saatiin melko hyvin vastauksia. Vastaajista viisi ei osannut vastata kysymykseen ja neljä kertoi, että dokumentaatiossa ei ole puutteita tai, että se on nyt hyvällä tasolla. Loput kymmenen vastausta olivat kehitysideoita siihen, miten dokumentaatiota tulisi

kehittää parempaan suuntaan. Vastaukset olivat linjassa kyselyn aikaisempien väittämien avointen vastausten kanssa ja useampi niistä toistui.

Suurimpana kehityskohtana vastaajat nostivat dokumentaation selkeyden ja päivittämisen. Useampi vastaaja kertoi, että dokumentaatio kaipaisi lisää selkeyttä niin rakenteellisuuden kuin ulkoasun näkökulmasta. Näiden lisäksi vastaajat kaipasivat esimerkkejä valmiista koodipätkistä sekä räätälöidystä toteutuksesta. Myös suomen kieli nousi esiin muutaman vastaajan kommentista. Pääkohtien lisäksi yksittäisinä kehityskohtina vastaajat kertoivat maksutapojen eron kuvailun, ominaisuuksien palvelukuvaukset, SDK:t sekä ominaisuuksien rajoitusten kuvaukset.

6.4 Koodikielet

Kuten aikaisempien väittämien vastaukset osoittivat, koodikielet ja koodiesimerkit ovat isossa roolissa dokumentaation kanssa. Seuraavassa kyselyn kohdassa kysyttiin *Millä koodikielillä esimerkkien tulisi olla?* Kysymys oli pakollinen ja vastaajat saivat valita niin monta vastausvaihtoehtoa, kun halusivat. Yhteensä vastauksia tuli 49 kappaletta ja kaikki vaihtoehdot saivat vähintään yhden äänen. Koska yksi vastaaja sai valita useamman vaihtoehdon, kaikkien vastausten yhteenlasketuksi prosentiksi muodostui 257,9 %.

Koodikieli PHP oli kyselyn selvästi suosituin. Reilu neljännes (28,6 %) oli sitä mieltä, että koodikielien pitäisi olla tuolla kielellä. Seuraavaksi eniten, hieman vajaan neljänneksen (18,4 %) äänistä, sai JavaScript. Kolmannen sijan koodikielistä jakoivat Python ja Curl, reilulla kymmenellä prosentilla (12,2 %) äänistä. Loput koodikielet saivat 1–5 kappaletta ääniä, mikä tekee prosentuaalisesti 2–10 %. Vastausvaihtoehto Jokin muu sai yhteensä 6,1 % äänistä ja ne pitivät sisällään kommentit Postman-esimerkeistä, SDK:sta ja saman esimerkin olemisesta usealla koodikielellä. Kaikki vastaukset on esitetty taulukossa 10.

TAULUKKO 10 Koodikielien vastausten jakauma

		Vastaukset		% suhteessa kaikkiin vastauksiin
		N	%	
Koodikielet	C#	3	6,1%	15,8%
	PHP	14	28,6%	73,7%
	Java	5	10,2%	26,3%
	JavaScript	9	18,4%	47,4%
	Python	6	12,2%	31,6%
	Curl	6	12,2%	31,6%
	Ruby	2	4,1%	10,5%

	Go	1	2,0%	5,3%
	Jokin muu	3	6,1%	15,8%
Yhteensä		49	100,0%	257,9%

Vastaukset jakautuivat melko tasaisesti myös vastaajien työkokemuksen mukaan. Työkokemuksen määrää mitattiin asteikoilla 0-2, 3-6, 7-10 ja 10+. Kyselyssä ei tullut yhtään 0–2 vastausta.

Täysin saman määrän vastauksia kaikkien työvuosiluokkien välillä saivat koodikielet C# ja JavaScript. Eniten ääniä saanut kieli, PHP, oli suosituin kaikkien vuosiluokkien välillä ja etenkin 10+-vastaajien keskuudessa Vajaa puolet (42,9%) PHP:ta äänestäneistä vastaajista oli työskennellyt organisaatiossa yli kymmenen vuotta. 3–6-vastaajat ja 7–10-vastaajat saivat molemmat reilun neljänneksen äänistä (28,6%). Eniten hajontaa aiheutti Python-koodikieli. Kaikkien vaihtoehdon vastauksista jopa kaksi kolmasosaa (67,7%) tuli 3–6-vastaajilta. Loput äänet tulivat 10+-vastaajilta.

Eniten ääniä tuli 3-6-vastaajilta (40,8%), toiseksi eniten 10+-vastaajilta (34,7%) ja kolmanneksi eniten 7–10-vastaajilta (24,7%). Vastausjakaumat on esitetty tarkemmin vielä taulukossa 11.

TAULUKKO 11 Koodikielet työkokemuksen mukaan

			Koodikielet									Yht.
			C#	PHP	Java	JS	Python	Curl	Ruby	Go	Muu	
Työkokemus	3–6	N	1	4	1	3	4	2	2	1	2	20
		% kielestä	33,3%	28,6%	20%	33,3 %	67,7%	33,3%	100%	100%	67,7%	40,8%
	7–10	N	1	4	2	3	0	1	0	0	1	12
		% kielestä	33,3%	28,6%	40%	33,3%	0%	16,7%	0%	0 %	33,3%	24,7%
	10+	N	1	6	2	3	2	3	0	0	0	17
		% kielestä	33,3%	42,9%	40%	33,3%	33,3%	50%	0.0%	0 %	0 %	34,7%
Yht.		N	3	14	5	9	6	6	2	1	3	49
		% kielestä	100%	100%	100%	100%	100%	100%	100%	100%	100%	100%

Seuraavana vuorossa on koodikieliäänien jakautuminen vastaajien organisaatioiden roolien mukaan. Koska tutkimuksen tarkoituksena on tutkia nimenomaan kehittäjien mielipiteitä teknisestä dokumentaatiosta, keskitytään tässä ainoastaan kehittäjien antamiin ääniin. Kaikki annetut vastaukset on esitetty taulukossa 12.

Vastaajat, jotka ilmoittivat roolikseen ”kehittäjä” antoivat eniten ääniä melkein puolet kaikista äänistä (46,9%). Kehittäjien keskuudessa eniten ääniä (30,4%) sai koodikieli PHP. JavaScript ja Python saivat molemmat yhtä paljon

ääniä, noin vajaan viidenneksen (17,4%). Neljänneksi eniten ääniä sai Curl, ääniä kertyi reilu kymmenes (13%). Loput kielistä saivat 0–2 kappaletta ääniä.

TAULUKKO 12 Koodikielet vastaajien roolien mukaan

			Koodikielet									Yht.
			C#	PHP	Java	JS	Python	Curl	Ruby	Go	Jm.	
Rooli	Kehittäjä	N	0	7	2	4	4	3	1	0	2	23
		% roolista	0%	30,4%	8,7%	17,4%	17,4%	13%	4,3%	0%	8,7%	46,9%
	Markkinoinnins.	N	0	0	1	0	0	0	0	0	0	1
		% roolista	0 %	0 %	100 %	0 %	0 %	0 %	0 %	0%	0%	2 %
	Joht. / omist.	N	1	4	1	3	0	1	0	0	0	10
		% roolista	10%	40 %	10%	30%	0 %	10 %	0 %	0%	0%	20,4%
	Yrittäjä	N	2	3	1	2	2	2	1	1	1	15
		% roolista	13,3%	20%	6,6%	13,3%	13,3%	13,3%	6,6%	6,6%	6,6%	30,6%
Total		N	3	14	5	9	6	6	2	1	3	49
		% Yhteensä	100%	100%	100%	100%	100%	100%	100%	100%	100%	100 %

7 POHDINTA

Tässä luvussa käydään läpi tutkimuksen tuloksia, ja pohditaan, kuinka hyvin niiden avulla voidaan vastata tutkimuksen alussa asetettuihin tutkimuskysymyksiin. Ensimmäisessä alaluvussa käydään läpi tutkimuksen tuloksia pohjautuen tutkimuskysymyksiin ja teoreettiseen viitekehykseen sekä pyritään antamaan ehdotuksia ja toimintamalleja toimeksiantajalle. Toisessa alaluvussa arvioidaan tutkimuksen luotettavuutta sekä sen mahdollisia rajoituksia.

7.1 Vastaukset tutkimuskysymyksiin

Tutkimuksen alussa esitettiin kaksi tutkimuskysymystä, joiden avulla pyrittiin saamaan vastauksia tutkimusongelmaan. Kysymykset olivat:

- Miten verkkokauppojen tekniset toteuttajat kokevat kehittäjäkokemuksen toteutumisen toimeksiantajan teknisen dokumentaation avulla?
- Miten dokumentaatiota voisi parantaa, jotta kehittäjäkokemus olisi entistä parempi?

Vastaukset kysymyksiin pyrittiin löytämään kirjallisuuden ja empiirisen aineiston avulla. Alaluvuissa käydään läpi tuloksia ja pohditaan, miten niitä voisi hyödyntää.

7.1.1 Kehittäjäkokemus teknisessä dokumentaatiossa

Kehittäjäkokemus termi on verrattain uusi, eikä sitä ole tutkittu, kun vasta viimeisen kymmenen vuoden ajan. Fagerholm ja Münch (2012) kehittivät kehittäjäkokemuksen viitekehyksen, joka jakautuu kolmeen osaan:

vaikutustasoon (affect), kognitiotasoon (cognition) ja arvostustasoon (conation). Nämä kolme osaa muodostavat hyvän, kokonaisen kehittäjäkokemuksen.

Tutkimuksen kannalta oli oleellista selvittää, miten tekninen dokumentaatio nivoutuu teknisten toteuttajien kehittäjäkokemukseen. Moilanen ym. (2018) nostavat dokumentoinnin yhdeksi osaksi hyvän kehittäjäkokemuksen nelikenttämalliaan. Hyvän dokumentaation ja koodiesimerkkien on tarkoitus auttaa teknisiä toteuttajia työssään ja ymmärtämään, miten palvelu toimii. Mikäli dokumentaatio ei ole toivotulla tasolla, voi se johtaa kehittäjien turhautumiseen ja sitä kautta negatiiviseen kehittäjäkokemukseen. Dokumentaation puute tai vajavaisuus nousivat myös Fordin ja Parnin (2015) tutkimuksessa turhautumista aiheuttaviksi tekijöiksi.

Tutkimuksen väittämät jakautuivat kolmeen osaan: sisällöllisiin, esillepanollisiin ja kehittäjäkokemukseen liittyviin väittämiin. Sisällöllisten väittämien tunnusluvuista voidaan todeta, että Paytrailin dokumentaatio on suurilta osin selkeää eikä siitä löydy kovinkaan paljon asia- tai sisältövirheitä. Vastaajat kokivat myös, että dokumentaatio on verrattain ajankohtaista eikä se ole moniselitteistä. Muutama vastaaja kuitenkin kertoi avoimessa vastauksessa, että rakenteen osalta, dokumentaation ominaisuudet eivät ole eritelty tarpeeksi hyvin. Kritiikkiä sai myös epäselvyys eri toteutusten välillä. Jos dokumentaatiossa on epäjohdonmukaisuuksia, voi tämä johtaa turhautumiseen ja sitä kautta kehittäjäkokemuksen heikentymiseen.

Keskeneräisyys ja koodiesimerkkien määrä nousivat selvästi dokumentaation heikommiksi sisällöllisiksi kohdiksi. Kyselyyn vastaajat toivoivat termien ja ominaisuuksien parempaa ja tarkempaa kommentointia sekä avaamista. Myös termien parempi kommentointi koettiin oleelliseksi. Dokumentaation keskeneräisyys tekee työskentelystä sekavaa ja suoraan yhteydessä työskentelyn nopeuteen ja tehokkuuteen, mitkä taas linkittyvät kehittäjäkokemukseen. Näiden lisäksi useat vastaajat toivoivat, että koodiesimerkit olisivat useilla eri koodikielillä. Eniten ääniä saivat PHP, JavaScript, Python ja Curl, joten dokumentaation koodiesimerkkien olisi hyvä olla ainakin muutamalla näistä kielistä. Ruby, Go ja C# keräsivät vain muutaman äänen, joten koodiesimerkkien luominen näillä kielillä ei ole tarpeellista.

Esillepanollisten osien osalta Paytrailin dokumentaatio oli niin ikään melko hyvällä tasolla. Etenkin väittämät dokumentaation pirstaloitumisesta ja liiallisesta rakenteellisesta tiedosta keräsivät hyvät arvosanat. Heikoimman arvosanan sai väittämä dokumentaation sekavuudesta. Paytrailin tekninen dokumentaatio voisikin näiden vastausten valossa kaivata hieman uudelleen rakentamista. Kyselyyn vastaajat kaipasivat dokumentaatiolta nimenomaan parempaa rakennetta. Rakennetta voisi parantaa esimerkiksi tuomalla peräkkäin suoritettavat pyynnöt lähemmäksi toisiaan. Myös kohteiden ja esimerkkien selkeämpi esiin tuominen voisi vähentää dokumentaation sekavuutta ja sitä kautta tehdä teknisten toteuttajien työn helpommaksi ja mukavammaksi. Näiden kohtien lisäksi palvelukuvauksen liittäminen osaksi ominaisuuksia lisäisi selkeyttä, vähentäisi pirstaloitumista ja parantaisi dokumentaation yleistä ilmettä ja jäsentelyä.

Väittämä kehittäjäkokemuksen puutteellisuudesta keräsi kaikista väittämistä kolmanneksi huonoimman keskiarvon, joten tästä voidaan päätellä, että kehittäjäkokemuksessa olisi parantamisen varaa. Seuraavassa alaluvussa 7.1.2 käydään läpi, mitä parannuksia Paytrailin dokumentaatioon voisi tehdä kehittäjäkokemuksen parantamiseksi.

7.1.2 Parempi tekninen dokumentaatio

Toiseen tutkimuskysymykseen lähdettiin hakemaan vastausta suoraan empiirisen kyselytutkimuksen kysymyksen, *Miten kehittäisit/parantaisit Paytrailin teknistä dokumentaatiota* avulla.

Teknisen dokumentaation kehittäjäkokemus paranisi kumppaneiden mielestä etenkin lisäämällä esimerkkejä, joita kehittäjät voisivat käyttää suoraan. Esimerkit voisivat olla valmiita koodipätkiä sekä räätälöityjä toteutuksia. Lisäksi, että esimerkkejä tulisi olla enemmän, niitä tulisi olla useimmilla ohjelmointikielillä. Ohjelmointikielet, joilla esimerkit olisi hyvä olla, ovat PHP, JavaScript, Python ja Curl. Ohjelmointikieliä ei kuitenkaan tarvitsisi olla tämän enempää, sillä se saattaisi puolestaan kääntyä tarkoitusta vastaan ja sekoittaa dokumentaatiota. Myös suomenkielisen dokumentaation tarjoamista voisi olla hyvä miettiä, sillä dokumentaatio on tarkoitettu pääsääntöisesti suomalaisten verkkokauppojen toteuttajille.

Esimerkkien ja ohjelmointikielien lisäämisen ohella teknisen dokumentaation selkeyden parantaminen parantaisi kehittäjäkokemusta. Varsinkin ulkoasun ja rakenteen selkeyttäminen helpottaisi kehittäjien työskentelyä. Nyt esimerkiksi osa kehittäjistä ihmettelee tavallisen maksupalvelutoteutuksen ja shop-in-shop-toteutuksen eroja ja päätyy toteuttamaan vahingossa väärän vaihtoehdon. Jos erottelu olisi hieman selkeämpi, vältyttäisiin väärinymmärryksiltä, työtehokkuus parantuisi ja samalla kehittäjäkokemus kohentuisi. Maksupalvelun ominaisuuksien palvelukuvausten perinpohjaisempi auki kirjoittaminen sekä maksutapojen erojen parempi erottelu olisivat niin ikään toivottavia parannuksia dokumentaation selkeyden osalta.

7.2 Rajoitukset ja luotettavuuden arviointi

Tutkielman luotettavuuden kannalta suurin epäsuotuinen tekijä oli hyvin vähäinen empiirisen tutkimuksen vastausprosentti. Näin ollen tuloksia ei voida yleistää eikä niistä voida tehdä tarkkoja johtopäätöksiä. Tulokset toimivatkin lähtökohtaisesti suuntaa antavina ohjeistuksina siitä, missä tilassa toimeksiantajan tekninen dokumentaatio on ja, miten sitä tulisi kehittää kehittäjäkokemuksen parantamiseksi. Heikkilä (2014) kertookin, että tutkimuksen tuloksia ei pidä yleistää niiden pätevyysalueen ulkopuolelle. Pienen otoskoon vuoksi tulokset ovat sattuman varaisia.

Alkuperäistä ideaa korrelaation hakemisesta dokumentaatiota koskevien vastausten ja kehittäjäkokemusvastausten välillä ei voitu toteuttaa liian pienen otoskoon vuoksi.

Kyselyn matalan vastausprosentti voi johtua monesta tekijästä. Kysymysten tai väittämien hyvyys ja oikeellisuus riippuu Kanasen (2014) mukaan ainakin seuraavista kohdista:

- Ymmärrettävyys: vastaajan tulee ymmärtää kysymykset oikein
- Asiasta oleva tieto: vastaajalla tulee olla kysymysten edellyttävä tieto
- Halu vastata: vastaajan tulee haluta antaa kysymyksiin liittyvä tieto
- Yksiselitteisyys: Kysymysten tulee olla yksiselitteisiä

Mikäli nämä kohdat eivät täyty, vastaajat voivat kokea kyselyn turhan vaikeaksi ja jättävät kokonaan vastaamatta. Voi olla, että kehittäjäkokemus-termi oli joillekin vastaajille hieman turhan hankala hahmotettavaksi. Tämä näkyi myös vastaajien avoimista vastauksista. Vastauksia olisi todennäköisesti tullut enemmän, mikäli termi olisi tutumpi tai se olisi saatu avattua paremmin kyselyn yhteydessä.

Kysely oli myös ehkä hieman turhan pitkä. Heikkilä (2014) toteaa, että tutkimuksen tulee mitata sitä, mitä on tarkoitus selvittää. Mikäli muuttujia ja käsitteitä ei ole määritelty tarkkaan, eivät mittaustulokset voi olla päteviä. Yksinkertaistamalla väittämiä ja karsimalla muutamia taustakysymyksiä kyselystä olisi saatu ytimekkäämpi, jolloin vastauksia olisi voitu saada lisää.

Kysely lähetettiin vastaajille sähköpostitse, mikä on lähtökohtaisesti paras väylä verkkokyselyn jakamiseen. Kuitenkin kyselyn kohderyhmä, verkkokauppojen rakentajat, on tunnetusti hankala tavoittaa sähköpostin välityksellä. Näin ollen kyselyä olisi voitu jakaa muita väyliä, kuten pikaviestimiskanavia ja sosiaalisia medioita, pitkin. Potentiaalisia vastaajia olisi voinut myös lähestyä toimialan tapahtumissa. Tämä olisi tosin voinut vääristää vastauksia, mutta toisaalta niitä olisi voinut tulla hieman enemmän.

Jälkikäteen ajateltuna empiirisen tutkimuksen olisi voinut myös toteuttaa laadullisena haastattelututkimuksena. Näin olisi voitu varmistaa, että tutkimukseen osallistujat olisivat ymmärtäneet kaikki kysymykset ja mahdollisilta väärinymmärryksiltä sekä vähäsanaisilta vastauksilta olisi voitu välttyä. Laadullinen tutkimus olisi voinut tuoda tarkempaa lisätietoa teknisen dokumentaation mahdollisista korjausehdotuksista. Nyt moni vastaaja jätti avointen kysymysten vastauskentät tyhjiksi, jättäen varsinaisten vastausten määrän melko pieneksi.

8 YHTEENVETO

Tämä luku pitää sisällään teoreettisen viitekehyksen yhteenvedon sekä varsinaisen empiirisen tutkimuksen yhteenvedon. Näiden lisäksi luvussa tarkastellaan ja arvioidaan tutkimuksen rajoituksia ja luotettavuutta. Luku ja koko tutkielma päättyy mahdollisten jatkotutkimusaiheiden pohdintaan.

8.1 Teorian yhteenveto

Pro gradun teoriaosiossa selvitettiin varsinaisen tutkimuksen kannalta tärkeimpiä ja keskeisempiä termejä sekä olemassa olevia konsepteja käytössä olevan kirjallisuuden avulla. Tutkielman otsikoksi muodostui ”Kehittäjäkokemuksen toteutuminen teknisessä dokumentaatiossa” ja teoriaosuus jakautui kolmeen osaan: lean-ajatteluun ohjelmistokehityksessä, kehittäjäkokemukseen ja dokumentaatioon.

Lean-ajattelun nostaminen yhdeksi tutkielman teemaksi oli mielestäni tärkeää, sillä toimeksiantajan organisaatiossa on käytössä tämä kyseinen menetelmä. Tämän parempi ymmärtäminen auttoi ymmärtämään toimeksiantajaorganisaation työskentelymetodeja. Lean-ohjelmistokehitys voidaan tiivistää Poppendieckin & Cusumanon (2012) mukaan seitsemään periaatteeseen: kokonaisuuden optimointiin, hukan poistoon, laadun rakentamiseen, jatkuvaan oppimiseen, nopeaan toimitukseen, kaikkien mukaan ottamiseen ja jatkuvaan paranemiseen.

Toisena, hyvin isona, kokonaisuutena tutkielmassa käsitellään kehittäjäkokemusta. Kehittäjäkokemuksella (*developer experience, DX*) tarkoitetaan kokemusta, joka tapahtuu ohjelmistokehitysprosessissa ohjelmoijan sekä ohjelmointityökalujen välillä, kun kehittäjä työskentelee palveluiden tai tuotteiden kehityksen parissa. Termi on johdettu tunnetummasta kokonaisuudesta, käyttäjäkokemuksesta (*user experience, UX)*. Siinä missä käyttäjäkokemus keskittyy tutkimaan järjestelmien käyttöä, kehittäjäkokemus

keskittyy ohjelmistokehitykseen ohjelmistotyökalujen, mallinnuksen ja kehitysprosessien näkökulmasta.

Fagerholm ja Münch (2012) jakavat kehittäjäkokemuksen kolmeen tasoon: vaikutustasoon (affect), kognitiotasoon (cognition) ja arvostustasoon (conation). Vaikutustasolla tarkoitetaan kehittäjien tunteisiin vaikuttavia osa-alueita, kognitiotasolla puolestaan viitataan siihen, miten kehittäjät hahmottavat tai näkevät käytössä olevan kehitysinfrastruktuurin. Arvostustaso tarkoittaa kehittäjäkokemuksen kontekstissa sitä, miten kehittäjät näkevät oman työnsä panoksen arvon. Yhdessä nämä kolme muodostavat kehittäjäkokemuksen viitekehyksen.

Koska toimeksiantajan tarjoama tuote on API-pohjainen, tarkasteltiin kehittäjäkokemusta myös API-kehityksen näkökulmasta. Moilanen ym. (2018) nostavat esiin hyvän kehittäjäkokemuksen nelikenttämallin, joka pitää sisällään arvon luonnin, matalan kynnyksen, tuen sekä dokumentoinnin.

Ohjelmistokehitys on ihmislähtöistä toimintaa, minkä vuoksi sen inhimilliset tekijät nivoutuvat osaksi kehittäjäkokemusta. Kehittäjät ovat avainasemassa onnistuneen ohjelmointikehitysprojektin maaliin saattamisessa, ei ainoastaan tyypillisten tehtävien suorittajina, vaan koko kehitysprosessin ytimenä. Ohjelmistokehityksen inhimillisiä tekijöitä tarkasteltiin motivaation, onnellisuuden ja ilottomuuden näkökulmasta, niitä voidaankin pitää yksinä keskeisimpinä ohjelmistokehityksen inhimillisistä tekijöistä.

Kolmannessa teorian osassa keskityttiin dokumentaatioon, sen eri tyyppeihin ohjelmistokehityksessä sekä API-dokumentaatioon. Dokumentaatio on tärkeä osa ohjelmistokehitystä ja ylläpitoa. Ohjelmistokehittäjät tukeutuvatkin tekniseen dokumentaatioon, kun heillä on tarve saada käsitys ohjelmasta ja ymmärtää järjestelmää paremmin. Ohjelmistodokumentaatio muodostuu Horchin (2003) mukaan hallinto- (management), kehitys- (development), testaus- (test) ja käyttäjädokumentaatiosta (user).

Jotta API:ja voidaan käyttää mahdollisimman tehokkaasti, niitä on opeteltava käyttämään oikein. Oppimisen helpottamiseksi API:jen yhteydessä on yleensä olemassa käyttöä tukeva dokumentaatio sekä siihen liittyvät koodiesimerkit. Mikäli dokumentaatio on hyvällä pohjalla ja selkeä, kehittäjät voivat työskennellä tehokkaammin, mikä puolestaan parantaa kyseisen API:n mainetta kehittäjien keskuudessa. Mikäli dokumentaatio ei taas ole linjassa kehittäjien odotusten kanssa, voi tämä johtaa turhautumiseen, suuren ajanhukkaan ja jopa kyseisen API:n hylkäämiseen ja välttelemiseen tulevaisuudessa.

Uddinin ja Robillardin (2015) mukaan API-dokumentaation ongelmat voidaan jakaa sisällöllisiin ja esillepanollisiin ongelmiin. Sisällöllisiä ongelmia ovat keskeneräisyys, moniselitteisyys, selittämättömät esimerkit, vanhentuneisuus, epäjohdonmukaisuus ja virheellisyys. Esillepanollisia ongelmia puolestaan ovat API-dokumentaation paisuminen, pirstaloituminen, liiallinen rakenteellinen informaatio ja epäjohdonmukainen tai sekava informaatio. Juuri tätä Uddin ja Robillardin (2015) jakoa käytettiin pohjana empiirisen tutkimuksen kyselylomakkeessa.

8.2 Empiirisen tutkimuksen yhteenveto

Tämän tutkimuksen tarkoituksena oli tutkia verkkokauppojen teknisten toteuttajien eli niin sanottujen kumppanien ajatuksista siitä, miten kehittäjäkokemus toteutuu toimeksiantajan teknisessä dokumentaatiossa. Lisäksi tarkoituksena oli kerätä kehitysehdotuksia siitä, miten kehittäjäkokemus voisi olla entistä parempi dokumentaation osalta. Tutkimuksen taustalla oli dokumentaation kehittäminen kumppaneille mieluisaan suuntaan. Tutkimusongelmaa lähestyttiin kahden tutkimuskysymyksen avulla: Miten verkkokauppojen tekniset toteuttajat kokevat kehittäjäkokemuksen toteutumisen toimeksiantajan teknisen dokumentaation avulla? Miten dokumentaatiota voisi parantaa, jotta kehittäjäkokemus olisi entistä parempi? Kysymyksiin pyrittiin vastaamaan kyselytutkimuksella kerätyn aineiston avulla.

Tutkimus toteutettiin verkkokyselytutkimuksena ja se lähettiin sähköpostilla toimeksiantajan sähköpostilistalla oleville potentiaalisille vastaajille. Kysely rakennettiin pohjautuen sekä teoriaan sekä toimeksiantajan asiantuntijan näkemyksiin. Kyselystä lähettiin kaksi muistutusviestiä ja se jaettiin myös kumppaneille tarkoitetulla Slack-kanavalla.

Tutkimuksesta saatiin selville, että teknisen dokumentaation kehittäjäkokemus on melko hyvällä tasolla, mutta silti kehitettävää löytyy. Eniten tyytyväisiä oltiin sisältö- ja asiavirheiden sekä selittämättömien koodiesimerkkien vähyyteen. Vähiten tyytyväisiä oltiin puolestaan koodiesimerkkien määrään, dokumentaation keskeneräisyyteen sekä dokumentaation sekavuuteen. Etenkin ominaisuuksien ja termien tarkempi ja parempi kommentointi koettiin tarpeelliseksi vastaajien toimesta. Kun dokumentaatio on selkeämpi, kumppanien työtehokkuus paranee, mikä puolestaan parantaa kehittäjäkokemusta. Teknisen dokumentaation rakennetta voisi parantaa etenkin tuomalla peräkkäin suoritettavat pyynnöt lähemmäksi toisiaan. Tämän lisäksi palvelukuvauksen liittäminen osaksi ominaisuuksia voisi auttaa asiaa.

Yksi keskeisimmistä tuloksista tutkimuksen osalta oli myös koodiesimerkkien määrän tarve ja näiden esimerkkien tarjoaminen useille eri ohjelmointikielillä. Ohjelmointikielistä eniten ääniä saivat PHP, JavaScript, Python ja Curl. Esimerkkien tarjoamista ainakin näillä kielillä olisi hyvä miettiä.

Rajoitteiden ja luotettavuuden kannalta tutkimus olisi voinut olla hieman parempi. Kyselyyn saatiin odotettua vähemmän vastauksia, minkä vuoksi saatuja tuloksia ei voida yleistää. Näin ollen tulokset toimivatkin lähtökohtaisesti ainoastaan suuntaa antavina ohjeistuksina ja pienen otoskoon vuoksi ne ovat sattuman varaisia. Jälkikäteen ajateltuna tutkimuksen olisi voinut toteuttaa myös laadullisena tutkimuksena. Tällöin kehittäjäkokemuksen juurisyihin teknisessä dokumentaatiossa olisi voitu keskittyä tarkemmin, etenkin mikäli tutkimus olisi tehty puoliavoimena haastattelututkimuksena. Haastatteluiden avulla olisi voitu saada myös tarkempia kehitysehdotuksia dokumentaation ja kehittäjäkokemuksen parantamiseen. Nyt vastaukset jäivät lähinnä muutaman lauseen mittaisiksi.

8.3 Jatkotutkimusaiheet

Tutkimuksesta nousseista tuloksista nousi esiin muutama jatkotutkimusaihe. Etenkin ne kohdat, jotka keräsivät kritiikkiä toimeksiantajan teknisessä dokumentaatiossa, olisivat potentiaalisia tutkittavia aiheita. Näitä olivat esimerkiksi dokumentaation sekavuus ja keskeneräisyys sekä koodiesimerkkien sekä ohjelmointikielien määrä.

Voisi olla mielenkiintoista perehtyä entistä tarkemmin, miten verkkokauppojen tekniset kumppanit kokisivat dokumentaation vähemmän sekavana ja, millä tavalla dokumentaation rakennetta tulisi muuttaa paremman kehittäjäkokemuksen osalta. Etenkin konkreettisten parannusehdotusten löytäminen yhdessä kumppanien kanssa voisi olla hyvinkin järkevä tutkimuksen aihe.

Oman tutkimuksen voisi omistaa myös koodiesimerkkien tärkeydellä, määrällä sekä ohjelmointikielien valintaan. Tutkimukseen vastaajat kokivat, että koodiesimerkit tuottavat parempaa kehittäjäkokemusta, minkä vuoksi niiden rooli on kriittinen osa teknistä dokumentaatiota. Olisi hyvä selvittää, mistä kohdista dokumentaation käyttäjät tarvitsisivat esimerkkejä ja kuinka kattavia niiden tulisi olla. Lisäksi ohjelmointikielien osalta voisi vielä tarkentaa, mitkä kolme kieltä kumppanit kokevat kaikista tärkeimmiksi, ja millä kielillä esimerkkien tulisi ainakin olla. Mitä tarkempaa dataa kehitysehdotuksista saataisiin, sitä paremmat edellytykset teknisen dokumentaation kehittämisellä olisi.

LÄHTEET

Aghajani, E., Nagy, C., Vega-Marquez, O. L., Linares-Vasquez, M., Moreno, L., Bavota, G. & Lanza, M. (2019). Software Documentation Issues Unveiled. Teoksessa *2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE) (1199–1210).* IEEE.

Barbuto, J. E. (2005). Motivation and Transactional, Charismatic, and Transformational Leadership: A Test of Antecedents. *The Journal of leadership studies, 11(4),* 26-40.

Bass, B. M. (1999). Two decades of research and development in transformational leadership. *European journal of work and organizational psychology, 8(1),* 9-32.

Bass, B. M. (2000). The Future of Leadership in Learning Organizations. *The Journal of leadership studies, 7(3),* 18-40.

Beecham, S., Baddoo, N., Hall, T., Robinson, H. & Sharp, H. (2008). Motivation in Software Engineering: A systematic literature review. *Information and software technology, 50(9),* 860-878.

Beck, K., Beedle, M., Van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R.C., Mellor, S., Schwaber, K., Sutherland, K. & Thomas, D. (2001). *Manifesto for agile software development.*

Capretz, L. F. (2014). Bringing the Human Factor to Software Engineering. *IEEE software, 31(2),* 104.

Curtis, B., Krasner, H. & Iscoe, N. (1988). A field study of the software design process for large systems. *Communications of the ACM, 31(11),* 1268-1287.

DeMarco, T. & Lister, T. (1985). Programmer performance and the effects of the workplace. Teoksessa *Proceedings of the 8th international conference on Software engineering (ICSE '85) (268–272).* Washington, DC, USA: IEEE Computer Society Press.

DeMarco, T., & Lister, T. (1998). Human capital. *IEEE Software, 15(6),* 103-105.

de Souza, S. C. B., Anquetil, N. & de Oliveira, K. M. (2006). Which documentation for software maintenance? *Journal of the Brazilian Computer Society, 12(3),* 31-44.

Destefanis, G., Ortu, M., Counsell, S., Swift, S., Marchesi, M. & Tonelli, R. (2016). Software development: Do good manners matter? *PeerJ. Computer science, 2,* e73.

Ekwoge, O. M., Fontão, A., & Dias-Neto, A. C. (2017). Tester Experience: Concept, Issues and Definition. Teoksessa *2017 IEEE 41st Annual Computer Software and Applications Conference (COMPSAC) (208–213).* Turin: IEEE.

Evans, J. R. & Mathur, A. (2005). The value of online surveys. *Internet research, 15*(2), 195-219.

Fagerholm, F. (2015). *Software developer experience: Case studies in lean-agile and open source environments* (Väitöskirja). Tietojenkäsittelytieteen laitos. Helsingin yliopisto. Haettu osoitteesta https://helda.helsinki.fi/handle/10138/158080

Fagerholm, F., Ikonen, M., Kettunen, P., Münch, J., Roto, V. & Abrahamsson, P. (2015). Performance Alignment Work: How software developers experience the continuous adaptation of team performance in Lean and Agile environments. *Information and software technology, 64,* 132-147.

Fagerholm, F. & Münch, J. (2012). Developer experience: Concept and definition. Teoksessa *Proceedings of the International Conference on Software and System Process (ICSSP '12) (73-77).* IEEE Press.

Fisher, J. (2001). User Satisfaction and System Success: Considering the development team. *AJIS. Australasian journal of information systems, 9(1),* 21-29.

Fontao, A., Dias-Neto, A. & Viana, D. (2017). Investigating Factors That Influence Developers' Experience in Mobile Software Ecosystems. Teoksessa *2017 IEEE/ACM Joint 5th International Workshop on Software Engineering for Systems-of-Systems and 11th Workshop on Distributed Software Development, Software Ecosystems and Systems-of-Systems (JSOS) (55-58).* IEEE.

Ford, D. & Parnin, C. (2015). Exploring Causes of Frustration for Software Developers. Teoksessa *2015 IEEE/ACM 8th International Workshop on Cooperative and Human Aspects of Software Engineering (115-116).* IEEE.

Forlizzi, J. & Battarbee, K. (2004). Understanding experience in interactive systems. Teoksessa *Proceedings of the 5th conference on Designing interactive systems: processes, practices, methods, and techniques (DIS '04) (261-268).* New York: Association for Computing Machinery.

Franca, A., Gouveia, T., Santos, P., Santana, C. & da Silva, F. (2011). Motivation in software engineering: A systematic review update. Teoksessa *15th Annual Conference on Evaluation & Assessment in Software Engineering (EASE 2011) (154-163).* IET.

Garousi, G., Garousi, V., Moussavi, M., Ruhe, G. & Smith, B. (2013). Evaluating usage and quality of technical software documentation: an empirical study. Teoksessa *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering (EASE '13) (24-35).* New York: Association for Computing Machinery.

Garousi, G., Garousi-Yusifoğlu, V., Ruhe, G., Zhi, J., Moussavi, M. & Smith, B. (2015). Usage and usefulness of technical software documentation: An industrial case study. *Information and software technology, 57,* 664-682.

Graham, D. (2002). Requirements and testing: Seven missing-link myths. *IEEE software, 19(5),* 15-17.

Graziotin, D., Fagerholm, F., Wang, X., & Abrahamsson, P. (2018). What happens when software developers are (un)happy. *Journal of Systems and Software, 140,* 32–47

Graziotin, D., Fagerholm, F., Wang, X. & Abrahamsson, P. (2017). On the Unhappiness of Software Developers. Teoksessa *Proceedings of the 21st International Conference on Evaluation and Assessment in Software Engineering (EASE'17) (324-333)*. New York: Association for Computing Machinery.

Graziotin, D., Wang, X. & Abrahamsson, P. (2015). How do you feel, developer? An explanatory theory of the impact of affects on programming performance. *PeerJ. Computer science, 1,* e18.

Hassenzahl, M. & Tractinsky, N. (2006). User experience - a research agenda. *Behaviour & information technology, 25(2),* 91-97.

Heikkilä, T. (2014). *Tilastollinen tutkimus* (9. uud. p.). Helsinki: Edita.

Hirsjärvi, S., Remes, P., Sajavaara, P., & Sinivuori, E. (2009). *Tutki ja kirjoita* (15. uud. p.). Helsinki: Tammi.

Hooley, T., Marriot, J. & Wellens, J. 2013. *What is Online Research.* Lontoo: Bloomsbury

Horch, J. (2003). *Practical guide to software quality management.* Artech House. Haettu osoitteesta https://ebookcentral.proquest.com/

Ergonomics of human-system interaction (2019). *Part 210: Human-centred design for interactive systems (ISO 9241-210:2019)* Haettu osoitteesta https://www.iso.org/obp/ui/#iso:std:iso:9241:-210:ed-2:v1:en

Iyer, B., & Subramaniam, M. (2015, 7. tammikuuta). The Strategic Value of APIs. Harvard Business Review. Haettu 23.3.2022 osoitteesta https://hbr.org/2015/01/the-strategic-value-of-apis

Kaluzniacky, E. (2004). Managing Psychological Factors in IT Work: An Orientation to Emotional Intelligence. *Information Management, 17(3/4),* 5.

Kananen, J. (2015). *Opinnäytetyön kirjoittajan opas: Näin kirjoitan opinnäytetyön tai pro gradun alusta loppuun*. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Kananen, J. (2014). *Verkkotutkimus opinnäytetyönä: Laadullisen ja määrällisen verkkotutkimuksen opas*. Jyväskylä: Jyväskylän ammattikorkeakoulu.

Khan, I., Hierons, R. & Brinkman, W. (2006). Programmer's mood and their performance. Teoksessa *Proceedings of the 13th Eurpoean Conference on Cognitive Ergonomics: Trust and Control in Complex Socio-Technical Systems (ECCE '06) (123-124).* New York: Association for Computing Machinery.

Krill, P. (2016). Make room, Java: New languages take a slice of the pie. *JavaWorld.*

Kupiainen, E., Mäntylä, M. V. & Itkonen, J. (2015). Using metrics in Agile and Lean Software Development – A systematic literature review of industrial studies. *Information and software technology, 62,* 143-163.

Kuusinen, K. (2016a). Are Software Developers Just Users of Development Tools? Assessing Developer Experience of a Graphical User Interface Designer. Teoksessa *6th International Conference on Human-Centred Software Engineering (HCSE) / 8th International Conference on Human Error, Safety, and System Development (HESSD) (215-233).* Cham: Springer.

Kuusinen, K. (2016b). Software developers as users: Developer experience of a cross-platform integrated development environment. Teoksessa P. Abrahamsson, L. Corral, M. Oivo, B. Russo (toim,), *International Conference on Product-Focused Software Process Improvement (546-552).* Cham: Springer.

Kuusinen K., Petrie H., Fagerholm F., Mikkonen T. (2016) Flow, Intrinsic Motivation, and Developer Experience in Software Engineering. Teoksessa H. Sharp, T. Hall (toim.), Agile *Processes, in Software Engineering, and Extreme Programming. XP 2016. Lecture Notes in Business Information Processing (104-117).* Cham: Springer.

Lee, H. & Pan, Y. (2021). Evaluation of the Nomological Validity of Cognitive, Emotional, and Behavioral Factors for the Measurement of Developer Experience. *Applied sciences, 11(17),* 7805.

Lethbridge, T., Singer, J. & Forward, A. (2003). How software engineers use documentation: The state of the practice. *IEEE software, 20(6),* 35-39.

Li, Y., Tan, C. & Teo, H. (2012). Leadership characteristics and developers' motivation in open source software development. *Information & management, 49(5),* 257-267.

Liu, L., Feng, L., Li, Y., Jing, D. & Yang, H. (2016). Flourishing creativity in software development via Internetware paradigm. *Science China. Information sciences, 59(8),* 36-48.

Mockus, A. (2010). Organizational volatility and its effects on software defects. Teoksessa *Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (FSE '10) (117-126).* New York: Association for Computing Machinery.

Moilanen, J., Niinioja, M., Seppänen, M. & Honkanen, M. (2018). *API-talous 101.* Helsinki: Alma Talent.

Morales, J., Rusu, C., Botella, F. & Quinones, D. (2019). Programmer eXperience: A Systematic Literature Review. *IEEE access, 7*, 71079-71094.

Murphy, L., Kery, M. B., Alliyu, O., Macvean, A. & Myers, B. A. (2018). API Designers in the Field: Design Practices and Challenges for Creating Usable APIs. Teoksessa *2018 IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC) (249-258).* IEEE.

Myers, B. & Stylos, J. (2016). Improving API usability. *Communications of the ACM, 59(6),* 62-69.

Mäntylä, M., Adams, B., Destefanis, G., Graziotin, D. & Ortu, M. (2016). Mining Valence, Arousal, and Dominance - Possibilities for Detecting Burnout and Productivity? Teoksessa *2016 IEEE/ACM 13th Conference on Mining Software Repositories (MSR) (247-258).* Austin.

Ortu, M., Adams, B., Destefanis, G., Tourani, P., Marchesi, M. & Tonelli, R. (2015). Are Bullies More Productive? Empirical Study of Affectiveness vs. Issue Fixing Time. Teoksessa *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories. (303-313).* IEEE.

Parnas, D.L. (2011). Precise Documentation: The Key to Better Software. Teoksessa S. Nanz (toim.), *The Future of Software Engineering (125-148).* Berlin: Springer.

Poppendieck, M. & Cusumano, M. A. (2012). Lean Software Development: A Tutorial. *IEEE software, 29(5),* 26-32.

Poppendieck, M., & Poppendieck, T. (2003). *Lean software development: an agile toolkit.* Addison-Wesley.

Robillard, M. (2009). What Makes APIs Hard to Learn? Answers from Developers. *IEEE software, 26*(6), 27-34.

Robillard, M. P. & DeLine, R. (2010). A field study of API learning obstacles. *Empirical software engineering: an international journal, 16(6),* 703-732.

Schmidt, R. F. (2013). *Software engineering: Architecture-driven software development.* Elsevier Science & Technology. Haettu osoitteesta https://ebookcentral.proquest.com/lib/jyvaskyla-ebooks/detail.action?docID=1187150#

Snyder, C. R. & Lopez, S. J. (2005). *Handbook of positive psychology.* New York: Oxford University Press.

Sommerville, I. (2016*). Software engineering* (10. uud. p.). Pearson Education.

Systems and software engineering -- Developing user documentation in an agile environment. (2012). *Systems and Software Engineering – Developing User Documentation in an Agile Environment. (ISO/IEC/IEEE 26515:2012)* Haettu osoitteesta https://ieeexplore.ieee.org/document/6170923

Uddin, G. & Robillard, M. P. (2015). How API Documentation Fails. *IEEE software, 32*(4), 68-75.

Valli, R. & Aaltola, J. (2015). *Ikkunoita tutkimusmetodeihin: 1, Metodin valinta ja aineistonkeruu : virikkeitä aloittelevalle tutkijalle* (4. uud. ja täyd. p.). Jyväskylä: PS-kustannus.

Vilkka, H. (2007). *Tutki ja mittaa: Määrällisen tutkimuksen perusteet*. Helsinki: Tammi.

Vilkka, H. (2021). *Tutki ja kehitä* (5., päivitetty painos.). Jyväskylä: PS-kustannus.

Von Mayrhauser, A. & Vans, A. (1995). Program comprehension during software maintenance and evolution. *Computer (Long Beach, Calif.), 28(8),* 44-55.

Wang, X., Conboy, K. & Cawley, O. (2012). "Leagile" software development: An experience report analysis of the application of lean approaches in agile software development. *The Journal of systems and software, 85(6),* 1287-1299.

Womack, J. P., Jones, D. T., & Roos, D. (2007). *The machine that changed the world: The story of lean production--Toyota's secret weapon in the global car wars that is now revolutionizing world industry*. Simon and Schuster.

Wu, C., Gerlach, J. H. & Young, C. E. (2007). An empirical analysis of open source software developers' motivations and continuance intentions. *Information & management, 44(3),* 253-262.

LIITE 1 SAATEKIRJE

Paytrail by nets

Avaa viesti selaimessa

Arvoisa kumppani,

Paytrail kutsuu sinut vastaamaan kehittäjäkokemukseen liittyvään kumppanikyselyyn. Kysely on kohdennettu kumppaneillemme, jotka ovat toteuttaneet uuden Paytrailin integraation tai ovat toteuttamassa sitä.

Kyselyn tarkoituksena on selvittää kumppaniemme kokemuksia maksupalvelumme teknisestä dokumentaatiosta. Kysely on osa Mikael Mäntylän pro gradu -tutkielmaa.

Kyselyyn vastaaminen vie aikaa muutaman minuutin. Vastauksesi käsitellään ehdottoman luottamuksellisesti.

Jättämällä sähköpostiosoitteesi kyselylomakkeen lopulla, osallistut Powerin lahjakortin arvontaan. Lahjakortin arvo on 100 €. Voittajalle ilmoitetaan henkilökohtaisesti. Antamiasi vastauksia ei yhdistetä arvontaan.

Vastaa kyselyyn TÄSTÄ

Vastaathan kyselyyn **viimeistään 09.10.2022**. Kiitos vastauksistasi jo etukäteen!

Terveisin,
Mikael Paytraililta

Lisätietoja kyselyyn liittyen: mantylamikael@gmail.com

Vastaa kyselyyn

Paytrail
Lutakonaukio 7
40100 Jyväskylä
Finland

Seuraa meitä:

Sait tämän sähköpostin, koska olet hyväksynyt sähköpostiviestinnän yritykseltä Paytrail

Muuta sähköpostiasetuksia
Poistu kaikilta postituslistoilta

LIITE 2 KYSELYTUTKIMUS

Kehittäjäkokemus Paytrailin teknisessä dokumentaatiossa

1. Onko Paytrail käytössä organisaatiossasi? *

- ○ Kyllä
- ○ Ei

2. Missä roolissa toimit organisaatiossa? *

3. Työkokemuksesi vuosina kyseisessä organisaatiossa *

- ○ 0-2
- ○ 3-6
- ○ 7-10
- ○ 10+

4. Organisaation koko (henkilömäärä) *

- ○ 1-10
- ○ 11-50
- ○ 51-100
- ○ 100+

5. Sähköpostiosoite (mikäli haluat osallistua arvontaan)

Seuraava

Vastatessasi väittämiin, käytäthän referenssinä Paytrailin verkkosivuilta löytyvää teknistä dokumentaatiota: docs.paytrail.com
Dokumentaatio kannattaa pitää auki viereisessä välilehdessä

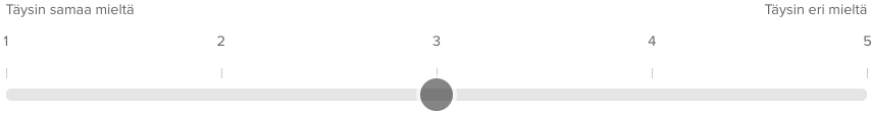
Dokumentaation keskeneräisyydellä tarkoitetaan sitä, että siinä on selvästi puuttuvia osa-alueita, jotka vaikeuttavat työskentelyä. Vastaa seuraavaan väittämään:

6. Paytrailin dokumentaatiossa esiintyy keskeneräisyyttä *

Esimerkkejä/mietteitä teemasta:

Dokumentaation moniselitteisyydellä tarkoitetaan sitä, että jonkin dokumentaation osan voi ymmärtää monella eri tavalla, eli siinä on hämmennystä tai ymmärtämättömyyttä aiheuttavaa monitulkinnaisuutta. Vastaa seuraavaan väittämään

7. Paytrailin dokumentaatiossa esiintyy moniselitteisyyttä *



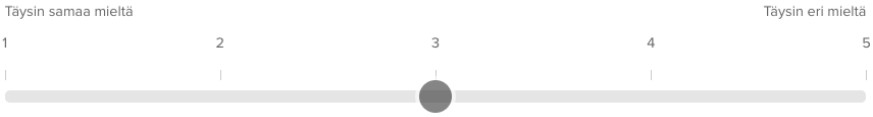
Esimerkkejä/mietteitä teemasta:

Edellinen

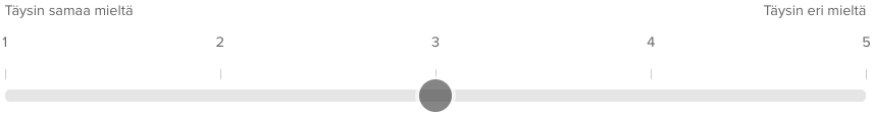
Seuraava

Dokumentaatiossa ilmenee koodiesimerkkejä. Esimerkkien on tarkoitus luoda mielikuva käyttöönoton helppoudesta ja mahdollistaa pikainen ensikosketus käyttöönottoon. Vastaa seuraaviin väittämiin:

8. Paytrailin dokumentaatiossa esiintyy selittämättömiä koodiesimerkkejä *



9. Paytrailin dokumentaatiossa esiintyy liian vähän koodiesimerkkejä *



10. Millä koodikielillä esimerkkien tulisi olla? *

Valitse yksi tai useampi vaihtoehto.

- [ ] C#
- [ ] PHP
- [ ] Java
- [ ] JavaScript
- [ ] Python
- [ ] Curl
- [ ] Ruby
- [ ] Go
- [ ] Jokin muu

Esimerkkejä/mietteitä teemasta:

Edellinen

Seuraava

Dokumentaation vanhentuneisuudella tarkoitetaan sitä, että dokumentaatio ei pysy varsinaisen kehityksen perässä, ja näin ollen dokumentaatio ei vastaa palvelua. Vastaa seuraavaan väittämään:

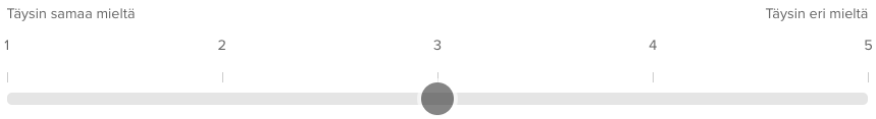
11. Paytrailin dokumentaatio on vanhentunutta

*

Esimerkkejä/mietteitä teemasta:

Dokumentaation epäjohdonmukaisuudella tarkoitetaan sitä, että sen osa-alueet eivät liity tai liittyvät huonosti toisiinsa. Tällä voidaan tarkoittaa myös yhteentoimivuuden puuttumista. Vastaa seuraavaan väittämään:

12. Paytrailin dokumentaatiossa esiintyy epäjohdonmukaisuutta *



Esimerkkejä/mietteitä teemasta:

Edellinen

Seuraava

Dokumentaation virheellisyydellä tarkoitetaan sitä, että siinä on asia- ja/tai sisältövirheitä, jotka ovat ristiriidassa toimivan palvelun kanssa. Vastaa seuraavaan väittämään:

13. Paytrailin dokumentaatiossa esiintyy asia- ja/tai sisältövirheitä *

Täysin samaa mieltä — Täysin eri mieltä

1 2 3 4 5

Esimerkkejä/mietteitä teemasta:

Dokumentaation paisumisella tarkoitetaan dokumentaation kasvamista niin suureksi, että sitä lukiessa on vaikea erottaa, mikä dokumentaatiossa on oikeasti tarpeellista tietoa. Vastaa seuraavaan väittämään:

14. Paytrailin dokumentaatiossa ilmenee paisumista *

Täysin samaa mieltä — Täysin eri mieltä

1 2 3 4 5

Esimerkkejä/mietteitä teemasta:

Edellinen

Seuraava

Dokumentaation pirstaloitumisella tarkoitetaan APIn osien hajautumiseen dokumentaation liian monille alasivuille. Pirstaloituminen voi vaikeuttavan dokumentaation selaamista ja näin asiaankuuluvan tiedon etsiminen hankaloituu. Vastaa seuraavaan väittämään:

15. Paytrailin dokumentaatiossa ilmenee pirstaloitumista *

Täysin samaa mieltä — Täysin eri mieltä

1 2 3 4 5

Esimerkkejä/mietteitä teemasta:

Dokumentaation liiallisella rakenteellisella tiedolla tarkoitetaan sitä, että dokumentaatio sisältää tietoa esimerkiksi luokkien tyypeistä ja tämä sama tieto olisi saatavilla myös ohjelmointiympäristöstä. Vastaa seuraavaan väittämään:

16. Paytrailin dokumentaatio sisältää liiallista rakenteellista tietoa *

Täysin samaa mieltä — Täysin eri mieltä

1 2 3 4 5

Esimerkkejä/mietteitä teemasta:

Dokumentaation sekavuudella tarkoitetaan sitä, että sitä on hankala seurata tai ymmärtää. Vastaa seuraavaan väittämään:

17. Paytrailin dokumentaatio on sekavaa *

Täysin samaa mieltä — Täysin eri mieltä

1 2 3 4 5

Esimerkkejä/mietteitä teemasta:

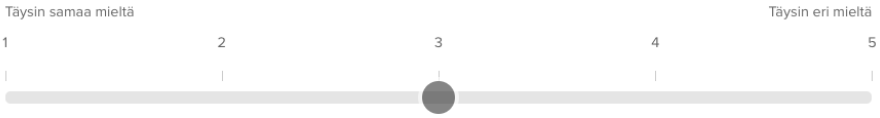
Edellinen

Seuraava

Kehittäjäkokemuksella (*developer experience*) tarkoitetaan kokemusta, joka tapahtuu ohjelmistokehitysprosessissa ohjelmoijan sekä ohjelmointityökalujen välillä, kun kehittäjä työskentelee palveluiden tai tuotteiden kehityksen parissa. Vastaa seuraavaan väittämään:

18. Kehittäjäkokemus on puutteellinen/vajavainen kehittäessä Paytrailin maksupalvelua organisaatiolleni

*



Esimerkkejä/mietteitä teemasta:

19. Miten kehittäisit/parantaisit Paytrailin teknistä dokumentaatiota? *

Edellinen

Lähetä vastaukset