Satakunnan ammattikorkeakoulu
Satakunta University of Applied Sciences

HANNES LEVÄNSUO

Ohjelmistoversioiden toimituksen ja kehityksen tehostaminen sekä käyttöönoton parantaminen

TOIMITUSVERKOSTON KEHITTÄMINEN
YAMK-TUTKINTO-OHJELMA
2024

TIIVISTELMÄ

Levänsuo, Hannes: Ohjelmistoversioiden toimituksen ja kehityksen tehostaminen sekä käyttöönoton parantaminen
Opinnäytetyö, ylempi AMK
Toimitusverkoston kehittäminen
Maaliskuu 2024
Sivumäärä: 93

Teknologian kehittyessä, sovelluskehitystä tekevien organisaatioiden on muokattava ja kehitettävä ohjelmistotuotteita tukemaan uusia vaatimuksia. Sovelluskehityksessä uusien versioiden toimitus käyttäjille on yksi keskeisistä prosesseista. Niiden laadukas kehitys ja käyttöönotto on pidettävä mahdollisimman nopeana ja tehokkaana.

Opinnäytetyössä keskityttiin tehostamaan kohdeorganisaatiossa uusien sovellusversioiden kehitystä ja toimitusta sekä helpottamaan niiden käyttöönottoa asiakkailla. Työ toteutettiin tapaustutkimuksena, jonka avulla hankittiin tietoa organisaation versiokehitys- ja toimitusprosesseista sekä asiakkaiden uuden version käyttöönotosta. Tavoitteena oli tehdä kehitysehdotus organisaatiolle prosessien tehostamiseksi sekä tunnistaa asiakkaiden ongelmat, tarpeet ja vaatimukset käyttöönotolle.

Tutkimus osoitti, että sovellusversioiden tuotannon, toimituksen ja käyttöönoton tehostaminen vaatii parannuksia sovelluksen laatuun sekä dokumentaatioon. Uusien sovellusversioiden käyttöönotto ei ollut asiakkailla helppoa, koska se vaati pitkiä ja laajoja testauksia sekä dokumentaatio ei vastannut asiakastarpeita. Näiden korjaamiseksi tehtiin kehitysehdotuksia sovelluskehitys- sekä toimitusprosesseihin. Työssä luotiin versiotiedotteiden pohja asiakasvaatimusten perusteella. Se vastaa paremmin asiakastarpeisiin uusien ohjelmistoversioiden julkaisun yhteydessä. Laadukas sovelluskehitys helpottaisi asiakkaiden käyttöönottotyötä sekä vähentää virheenkorjaustyötä versiojulkaisujen yhteydessä.

Avainsanat: sovellusohjelmat, käyttöönotto, kehitys, hankinta, ohjelmointi, testaus, laatu

Abstract

Levänsuo, Hannes: Improving the efficiency of developing, delivering and implementing new software versions.
Master's thesis
Development of Supply Network
March 2024
Number of pages: 93

Evolving technology is creating pressure for software developing organizations to support new demands it creates. Delivering new software versions to users is one of the most important processes. Organizations needs to keep new development as fast and efficient as possible.

The main point in the thesis was to increase the performance of quality software version development and make the software deployment as easy as possible for the customer. Study was done as a case study and in-depth information was gathered about the organization's development and delivery processes in addition to customers experience in implementation process. The goal was to make a development proposal for the organization to make processes more efficient and identify customers problems, needs and requirements for implementation.

The study showed that improving the production, delivery and implementation of the application versions requires improvements to the applications quality and documentation. The implementation of the new version was not easy for customers, because it required long, and in-depth testing and the documentation did not meet the customer needs. To correct these problems, development proposals were made for application development and delivery processes. In thesis there was also created a basis for release notes that is based on customer requirements. High-quality application development would help customers implementation process and reduce bug fixing work in every version release.

Keywords: applications, implementation, development, acquisition, programming, testing, quality

SISÄLLYS

1 JOHDANTO

Energia-alalla kyse on energian tuotannosta ja sen toimituksesta kuluttajille. Energiaa voi olla sähkö, kaasu, lämpö tai kylmä. Energiateollisuuden (2019, s. 2) mukaan kaasumarkkinat ovat maakaasun ja uusiutuvista energialähteistä peräisin olevan kaasun siirtoa tai käyttöä. Sähkömarkkinoiden tavoin maakaasumarkkinat ovat vapautettu kilpailulle ja kaasuenergian hankinta on eriytetty kaasuenergian siirrosta. Kaasulla kaupankäynti perustuu kahdenvälisiin sopimuksiin ja hintaindeksiin. Kauppaa kaasusta voidaan käydä myös kaasupörssissä. Lämmitysmarkkinoilla ei ole sääntelyä ja ne ovat kilpaillut eli hinnoittelua ei ole sidottu indekseihin lainsäädännöllisesti ja kuluttajat voivat valita haluamansa lämmitys- tai jäähdytysmuodot. (Gasum, n.d.; Vuorenmaa, n.d.)

Sähkötuotantoa on monenlaista kuten tuulivoima, turpeen poltto, ydinvoima ja vesivoima. Sähkö tuotetaan tuotantolaitoksella ja toimitetaan sähköverkkoa pitkin kuluttajalle, joka käyttää sähköä. Kuluttaja voi olla tehdas, kauppa tai asuintalo. Energian toimitusketju tuotannosta kulutukseen pitää ohjata sovelluksilla. Näitä ovat esimerkiksi energian mittausjärjestelmät, energiatiedonhallintajärjestelmät ja asiakkuudenhallintajärjestelmät. Energian mittausjärjestelmät mittaavat tietoa, esimerkiksi sähkömittarit mittaavat sähkön kulutusta kuluttajalla. Nämä mittaustiedot tallennetaan energiatiedonhallintajärjestelmään, jonka kautta voidaan muodostaa kulutussarjoja ja niiden perusteella laskutetaan asiakkuudenhallintajärjestelmällä. Asiakkuudenhallintajärjestelmän avulla myös hallitaan asiakkaiden toimia, joita ovat esimerkiksi muutto, sopimusmuutokset, laskutusten hallinta ja raportointi.

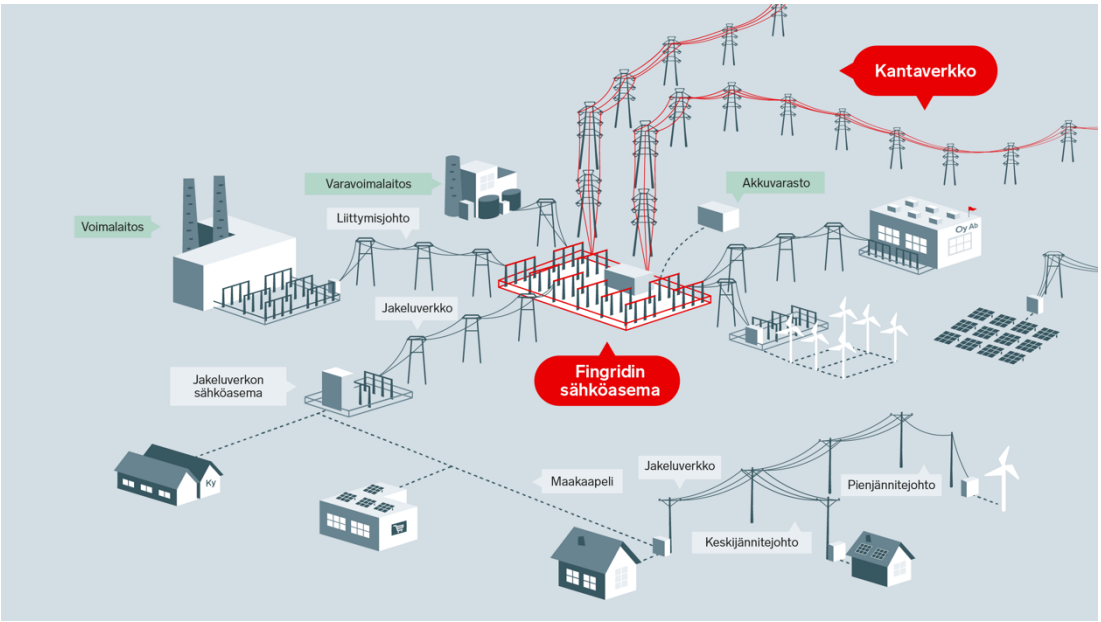
Tutkimus liittyy yritykseen, joka kehittää sovelluksia energia-alan toimijoille. Sitä tehdessä pitää huomioida alan vaatimukset sekä ymmärtää, minkälainen toimintaympäristö organisaatiolla on.

2 TYÖN LÄHTÖKOHDAT

Tässä luvussa esitellään tarkemmin toimintaympäristö, organisaatio sekä työn tavoitteet ja rajaukset. Luvussa käydään läpi, kuinka opinnäytetyö liittyy organisaatioon, alaan sekä kuinka se kehittää organisaatiota. Lopuksi käsitellään viitekehys, jonka ympärille kokonaisuus tehdään.

2.1 Toimintaympäristö

Suomessa kantaverkkoa ylläpitää Fingrid ja kantaverkko toimii sähkön siirtoverkkona ympäri Suomea. Sähkön tuotantolaitokset ja suuret tehtaat liittyvät suoraan kantaverkkoon, mutta kuluttajille sähkö tuodaan pienjänniteverkon kautta. Pienjänniteverkot ovat joko jakeluverkkoja tai pienjännitejohtoja ilmassa tai maakaapeleina maan alla. Fingrid huolehtii myös sähkön riittävyydestä suomessa ja sen kuljettamisesta ulkomaille tai ulkomailta Suomeen. Kuvassa 1 nähdään miten kantaverkko ja pienjänniteverkot yhdistyvät toisiinsa. (Fingrid, n.d.)



Kuva 1. Kantaverkko ja pienjänniteverkot (Fingrid, n.d.)

ICT-alalla tulee huomioida, että monet järjestelmät toimivat yhdessä tuottaen suuremman järjestelmäkokonaisuuden, joka ei toimisi ilman kaikkia muita järjestelmiä. Energiapuolella asiakkaiden sähkönsaanti riippuu monesta järjestelmästä. Ensin pitää rekisteröidä ja mitata mittarilla käyttöpaikan kulutus. Seuraavaksi kerätään mittaustiedot mittareilta erilliseen tietokantaan, josta seuraava järjestelmä taas voi laskuttaa käyttöpaikan kulutusta. Yhdessä järjestelmässä myös hoidetaan sanomaliikenne muiden energia-alan toimijoiden kanssa, kuten kommunikaatio sähköverkkoyhtiöiden ja sähkön myyjien välillä. Näitä järjestelmäkokonaisuuksia kutsutaan System of Systems (SoS) eli järjestelmäkokonaisuuksiksi, jotka hoitavat tiettyä kokonaistoiminnallisuutta. SoS käsite ei ole alakohtainen vaan yleisesti 1950-luvulta lähtöisin oleva tapa kuvata järjestelmäkokonaisuuksia. (Nielsen, 2015, luku 2.)

Kohdeorganisaatio huomioi toiminnassaan molemmat energia- ja ICT-alan. Energia-alalta tulee vaatimuksia yrityksille toimia tiettyjen periaatteiden mukaisesti ja nämä vaatimukset pitää lisätä alalla käytettäviin sovelluksiin. Sovellusten pitää myös pystyä siirtämään tietoa ja toimimaan yhdessä toisten järjestelmien kanssa SoS mukaisesti.

Sovelluskehitystä ja versiopäivityksiä tarvitaan, koska teknologia kehittyy ja luo uusia toiminnallisuuksia, esimerkiksi sähkömittareihin on tullut etäluenta, jonka avulla sähkömittarien kulutustiedot luetaan ilman fyysistä käyntiä mittarilla. Sovellusten pitää tukea näitä uusia toiminnallisuuksia, joita kehittyvä teknologia tuo mukanaan. Versiopäivityksien lukumäärä ja julkaisutahti riippuvat paljon sovelluksesta ja siitä missä vaiheessa elinkaarta sovellus kulloinkin on. Uudet sovellukset saavat enemmän päivityksiä, koska niissä on enemmän virheitä ja ne eivät vielä välttämättä sisällä kaikkia tarvittavia ominaisuuksia. Myöhemmässä vaiheessa tuotteen elinkaarta sovellus siirtyy odottamaan, kunnes kokonaan uusi sovellus korvaa vanhan ja silloin sitä ei päivitetä enää niin usein. Asiakkaat eivät välttämättä halua ottaa kaikkia maksullisia päivityksiä ja sen vuoksi heidän versionsa ovat jäljessä uusimmasta versiosta ja tämä aiheuttaa hankaluuksia, kun pitää hypätä monen version yli yhdellä päivityksellä. Opinnäytetyössä on tarkoitus tutkia, miten asiakkaat saataisiin ottamaan enemmän

versioita ja voidaanko versiotoimituksia tehostaa niin, että ne ovat asiakkaille houkuttelevampia.

2.2 Kohdeorganisaatio

Tutkimus ja toimenpide-ehdotukset tehdään organisaatiolle, joka toimittaa sovellustuotetta palveluna sekä Software as a Service (SaaS) että OnPremise-tuotteina. Tarkoitus on kuitenkin siirtyä enemmän SaaS-malliseen toimitukseen. SaaS tarkoittaa, että sovellus on asennettuna toimittajan palvelimille ja asiakas käyttää tuotetta sieltä. OnPremise-toimituksessa tuote asennetaan asiakkaan omille palvelimille (Sharda ym., 2014, kappale 1).

Organisaatio tehostaa jatkuvasti toimittamiaan järjestelmiä. Se luo niihin uutta ohjelmistokoodia, joka vastaa alalla syntyviin uusiin vaatimuksiin ja toiminnallisuuksiin. Mitä tehokkaammin versiokehitys saadaan toimimaan ja mitä useammin uusi versio saadaan toimitettua asiakkaille, sen paremmin uuskehitys onnistuu, asiakaspalautteiden perusteella tulevaisuuden suunnitelmien päivitys helpottuu sekä yhtenäinen versiohistoria helpottaa ja nopeuttaa myös tulevia versiotoimituksia.

Tällä hetkellä organisaatio työllistää yli sata asiantuntijaa niin energia-alalta, kuin sovelluskehityksen puolelta. Se pyrkii toimimaan koko energia-alaa ja tietoturvaa koskevissa yhteisissä projekteissa ja kehittää energiamarkkinoita eteenpäin.

Nykyisellään organisaatio tekee versiokehitystä seitsemän viikon kehitysjaksoissa (Inkrementti) ja jokainen inkrementti sisältää kolme kahden viikon kehityssykliä (Sprint) ja viikon mittaisen testaus- ja suunnittelujakson (Testing and Planning Sprint). Kehityssyklissä ohjelmoidaan tarvittavat ominaisuudet sovellukseen. Ne auttavat jakamaan inkrementin sisältämää työmäärää pienempiin ajanjaksoihin, jolloin työn valmistumisen seuranta on helpompaa. Testaus- ja suunnittelujakso sisältää tulevan inkrementin sisällön tarkastelun sekä edellisen inkrementin toteutuneiden ominaisuuksien testauksen. Jokaisen

inkrementin jälkeen muodostetaan versio ja osa versioista voi olla sisäisiä eli niistä ei julkaista asiakkaille asennettavaa versiota. Yleisesti asiakkaille jaettavat versiot ovat joko sovelluspäivityksiä tai korjauspäivityksiä.

2.3 Opinnäytetyön tavoite ja rajaus

Organisaatiolla sovellusversioiden kehitys ja toimitus ovat yleisimpiä prosesseja niin uusasiakashankinnassa, kuin nykyisten asiakkaiden versioiden päivityksissä. Uusien asiakkaiden hankinnassa toimitusprosessi on erilainen, koska pitää toimittaa kokonainen järjestelmä ja kaikki tietoliikenneyhteydet eri osapuolien välille sekä tunnistaa, että kaikki asiakkaan vaatimat osa-alueet ovat toimituksessa mukana ja asiakas osaa käyttää niitä. Versiopäivityksissä asiakkailla on jo tietotaito käyttää toimitettua sovellusta, mutta siihen tehdään virheenkorjauksia ja uusia ominaisuuksia, jotka tarvittaessa koulutetaan asiakkaille. Tässä opinnäytetyössä keskitytään nykyisten asiakkaiden uusien versioiden kehitykseen ja toimitukseen sekä siihen, miten versiotoimituksia voidaan parantaa ja miten versioiden hankkimisesta ja käyttöönotosta saataisiin asiakkaille helpompaa. Tämä tutkimus tuottaa tietoa koko alalle sovelluskehityksen ja -toimituksen mahdollisista ongelmista ja niiden ratkaisuista sekä sovelluskehitystä tekevän organisaation että asiakkaiden näkökulmasta.

Tutkimuskysymyksiä on kaksi:

- Miten tehdä uusien sovellusversioiden hankinnasta sekä käyttöönotosta helpompaa asiakkaille?
- Miten versiotoimituksia voidaan tehostaa ja nykyistä tuotanto- ja toimitusprosessia parantaa?

Tutkitaan, onko toimitus- ja kehitysprosessi yleisesti standardeja mukaileva (Digi- ja väestötietovirasto, 2020) sekä onko asiakkailla jotain toiveita toimitusprosessin kehittämiseksi niin, että se olisi myös asiakkaille mahdollisimman helppo. Yleisesti uuden version hankinnassa hankinnan tuoma lisäarvo ja käyttöönoton helppous ovat suurimpia tekijöitä sille hankkiiko asiakas tuotteen vai ei. Korkiakosken (2023, s. 112–113) mukaan asiakkaista kerättävä data voi

usein jäädä hyödyntämättä organisaatiossa, koska ei ole aikaa tai kiinnostusta. Tämän vuoksi kerätään asiakkailta tieto, joka liittyy suoraan toimitusprosessiin ja tuotteen laatuun. Tiedon avulla tehdään kehitysehdotukset organisaatiolle toimituksen ja versioiden houkuttelevuuden parantamiseksi.

Osa toimitusprosessin tehostamisesta liittyy siihen, että asiakkailla on uusimmat versiot käytössä. Näin versiotoimitus tulee helpommaksi, kun kaikilla asiakkailla on sama versio ja sen version päivitys on samanlainen prosessi jokaisella asiakkaalla. Nykytilanteessa osalla asiakkaista on käytössä uusin julkaistu versio ja osalla voi olla muutamia versiopäivityksiä vanha versio ja näissä tilanteissa molemmat asiakkaat voivat tilata uuden juuri julkaistun version. Tällöin versiotoimitus on erilainen molemmissa tapauksissa.

Versiopäivityksessä sovellukseen tehdään uutta tai muutetaan vanhaa ohjelmointikoodia. Se on helpoin tehdä uusimpaan versioon, koska muuttunutta tai lisättyä koodia ei ole paljon ja testaus on helppoa. Vanhemman version päivitys uudempaan vaatii enemmän työtä ja testausta, kun versioiden välissä on useampia versioita ja muuttuneen tai lisääntyneen koodin määrä on huomattava. Opinnäytetyössä sivutaan jatkuvan toimituksen prosessia eli Continous Delivery -prosessia (CD), jossa versiopäivityksiä voidaan tehdä sovellukseen koska tahansa ja versioiden luonti onnistuu kehitysputkesta välittömästi (Wilson, 2022, kohta Continous delivery). Ensin tarkoitus on löytää jatkuvasta toimituksesta sellaisia piirteitä, joita voitaisiin hyödyntää toimitustehokkuuden parantamisessa organisaatiossa ilman kokonaisen CD prosessin ja toimintamallin käyttöönottoa. Sen käyttöönotto kokonaisuudessaan vaatii huomattavia panostuksia yritykseltä, joten ensin tehostetaan organisaation nykyistä toimintamallia, minkä jälkeen askeleittain voidaan löytää parempi tie kokonaisen CD prosessin käyttöönottoon.

Versioissa ja versiotoimitusten lisäksi tulee kiinnittää huomiota, miten versioiden sisällöt vastaavat asiakkaiden toiveisiin ja tarpeisiin. Onko toimitettavassa versiossa niitä ominaisuuksia tai korjauksia, joita asiakas tarvitsee sekä tuovatko versiot asiakkaille liiketoimintahyötyjä joko toiminnallisia tai rahallisia. Versioiden sisällössä tulee huomioida, miksi asiakas haluaisi hankkia tietyn

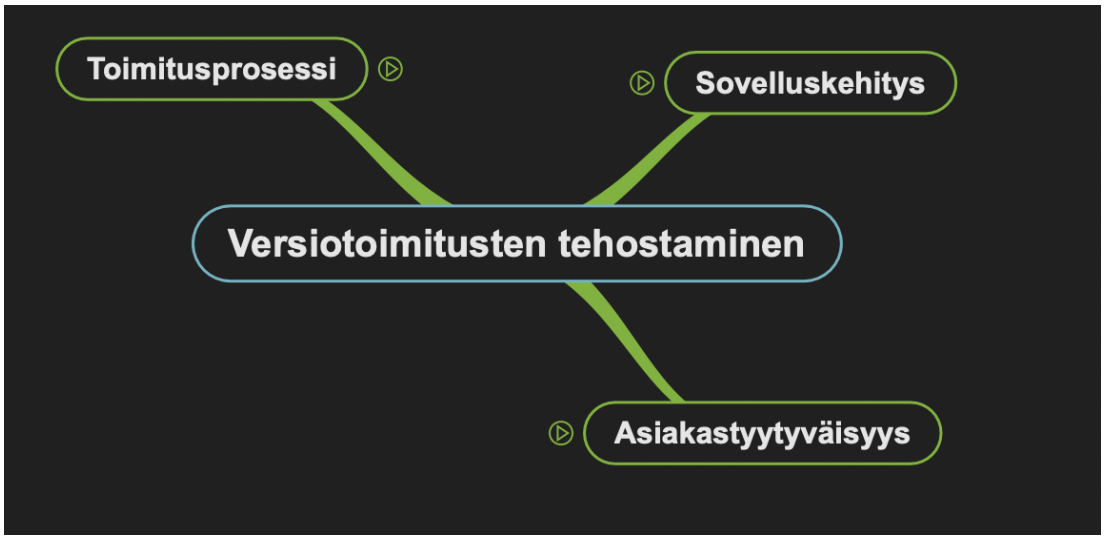
version. Mikäli se ei tuo asiakkaalle arvoa, sen hankkiminen ei ole kiinnostavaa. Osa versioista voi tuoda ominaisuuksia, joita kaikki organisaatiot eivät välttämättä tarvitse. Suuremmilla asiakkailla on erilaisia vaatimuksia ja tarpeita sovelluksilta kuin pienemmillä asiakkailla ja uudet suurille asiakkaille tehdyt ominaisuudet eivät välttämättä hyödytä pienempiä asiakkaita lainkaan. Uudessa versiossa pitää huomioida kaikki asiakkaat niin, että jokaiselle asiakasryhmälle tulee jotain kiinnostavaa jokaisessa versiossa.

Tuotteet toimitetaan asiakkaalle toimitusprojekteina, joissa on projektihinnoiteltu toimitusmaksu. Tämän lisäksi asiakkaalle tuotetaan palveluna joko järjestelmää, tukipalvelua tai sovellusta eli näistä kertyy kuukausilaskutuksessa joko lisenssimaksu tai palvelumaksu. Asiakas voi tilata organisaatiolta erikseen erillisiä räätälöintiratkaisuja tai valinnaisia erikseen lisensoituja ominaisuuksia. Uusia ominaisuuksia sisältävät versiotoimitukset toimitetaan myös erillisinä toimitusprojekteina. Opinnäytetyöstä on rajattu pois erillinen hinnoittelustrategiaan liittyvä houkuttelevuuden parantaminen. Työssä keskitytään ainoastaan teknologisiin tai toiminnallisiin parannuksiin sekä toimituksen tehostamiseen.

2.4 Viitekehys

Tietoperusta kuvaa olemassa olevaa teoriapohjaa, johon tutkimus perustuu. Tietoperusta muodostuu käsitteistä, joita käytetään ja hyödynnetään tutkimuksen tekemisessä ja johon tutkimuksen teoria viittaa, kun tuloksia analysoidaan. Tietoperusta kuvaa myös teorian, jota hyödynnetään tutkimuksen teossa ja johon tutkittava ilmiö, asia tai tutkimuksen tulokset nojaavat. (Ojasalo ym., 2015, s. 35.)

Työssä tutkitaan uuden sovellusversion kehitystä ja toimittamista asiakkaalle ja sen käyttöönotettavuutta, joten seuraavat käsitteet on valittu keskeisiksi käsitteiksi: toimitusprosessi, sovelluskehitys ja asiakastyytyväisyys. Kuviossa 1 on kuvattu käsitteiden suhteet työhön.



Kuvio 1. Tutkimuksen keskeiset käsitteet

Sovelluskehitys on suunnitelmallista tekemistä vaatimusten määrittelystä ohjelmointiin. Vaatimusten määrittely tarkoittaa kehitettävän kohteen ja ominaisuuksien suunnittelua eli sitä, kuinka ominaisuuden pitää toimia ja kuinka se toteutetaan. Opinnäytetyössä on tärkeä tarkkailla myös sovelluskehityksen mittareita esimerkiksi sovelluskehityksen nopeutta tai virheiden määrää versioissa, jotta voidaan löytää mahdollisia ongelmakohtia, joiden vaikutukset voivat liittyä toimitusprosessiin tai asiakkaiden halukkuuteen ottaa uusi versio käyttöön.

Sovelluskehitys on yleensä jonkinlainen hybridimalli näistä neljästä mallista: lineaarinen, iteratiivinen, ajastettu tai jatkuva virta. Lineaarinen malli tarkoittaa tietyn vaiheistetun mallin mukaista kehitystä, joka etenee vaiheittain eteenpäin. Yleensä tätä mallia kutsutaan vesiputousmalliksi. Iteratiivinen malli tarkoittaa vaatimusten kehitystä pienissä osissa. Ensin julkaistaan versio yksi vaatimuksesta, joka täyttää osan alkuperäisestä vaatimuksesta. Tämän jälkeen kerätään tietoa siitä ja kehitetään vaatimusta eteenpäin, kunnes se lopulta päätyy alkuperäisen vaatimuksen muotoon (iteraatiokierrokset). Ajastettu malli tarkoittaa samaa, kuin iteratiivinen julkaisumalli, mutta jokainen kehitysjakso on yhtä pitkä aika kalenteriajallisesti, työmäärä voi muuttua esimerkiksi kehityksen jakautumisen, sairauslomien tai lomakausien mukaan. Jatkuva virta tarkoittaa, että ylläpidetään tiettyä määrää kehitettävää työtä kehitysputkessa, jolloin jatkuvasti tulee uutta kehitettävää ja jatkuvasti luodaan

kehityspaketteja asiakkaille. Tämä eroaa iteratiivisesta ja aikakontrolloidusta menetelmästä niin, ettei työtehtäviä tule tietyin aikajaksoin kehitykseen vaan niitä lisätään jatkuvasti, kun osa kehitystehtävistä valmistuu. (Nicolette, 2015, kappale 1.2.1.)

Työssä käydään läpi mitä sovelluskehityksen mallia organisaatio käyttää, onko valittu malli sopiva nykyiseen kehitysprosessiin sekä hyödynnetäänkö mallia oikein. Vertaamalla valmista luotua mallia organisaation käyttämään malliin on mahdollista löytää ongelmakohtia tai ongelmia kehitysprosessissa, joita ei ole osattu huomioida, mikäli mallia on sovellettu.

Toimitusprosessin puolelta tutkitaan toimituksen tehokkuutta ja yksinkertaisuutta asiakkaan ja toimituksen näkökulmasta. Toimitusprosessin tulee olla nopea ja tuotantokatkojen mahdollisimman lyhyitä. Jatkuva integrointi ja toimitus (Continous Integration/Continous Delivery jatkossa CI/CD) mahdollistavat tuotteen tai palvelun laadukkaan toimituksen asiakkaalle, tuotteen nopean kehityksen ja laadun ylläpidon. Meroden (2023, kappale 1) mukaan CI/CD on lähestymistapa automatisoida ja käyttöönottaa ammattimainen sovellustoimitusketju. Tuotteen toimitus CI/CD:n avulla tarkoittaa kehitettyjen ominaisuuksien jakamista pienempiin osiin eli toimituspaketteihin ja kehityssykleihin. Kokonaisen ominaisuuden sijaan toimitetaan ensin yksi osa toiminnallisuudesta, joka itsenäisesti toimii, mutta ei välttämättä täytä koko tarvetta. Tämän jälkeen sitä kehitetään seuraavassa kehityssyklissä, jonka jälkeen ominaisuus täyttää alkuperäisen tarpeen. Näin toimituksesta saadaan massiivisten päivitysten sijaan jatkuvan toimitusprosessin mukainen. (Merode, 2023, kappale 2.)

Merode (2023, kohta Positioning of CI/CD) mainitsee, että toimitusprosessissa voidaan puhua myös sovelluksen toimitusketjusta, joka kuvaa toimenpiteet sovelluksen toimituksessa kehityksestä asiakkaalle. Versiotoimitusten lyhyt sykli antaa asiakkaille mahdollisuuden vaikuttaa nopeammin tuleviin versiopäivityksiin sekä niissä olevaan sisältöön. Opinnäytetyössä käydään läpi minkä tyylinen toimitusprosessi yrityksellä on käytössä ja voidaanko sitä tehostaa hyödyntämällä valmista mallia tai sen piirteitä.

Nieminen (2016, kappale 5.3) kuvaa toimittajan kehittämistä, jossa hankkija pyrkii parantamaan toimittajan toimintaa. Parannuskohteena voi olla esimerkiksi kyvykkyys, suorituskyky tai teknologia. Opinnäytetyössä on tärkeää huomioida kohdeorganisaation toimitusprosessin kehittäminen asiakkaiden näkökulmasta, jotta toimitusprosessista ja versiotoimituksesta saadaan houkuttelevampia.

Asiakastyytyväisyys on keskeinen käsite opinnäytetyössä, koska työn tulee löytää tapoja parantaa asiakastyytyväisyyttä ja sitä kautta sovelluspäivitysten houkuttelevuutta. Asiakastyytyväisyydessä keskitytään siihen, miten sovellusversiot, toimitusprosessi ja käyttöönotto vastaavat asiakkaan toiveisiin ja tarpeisiin, sekä kuinka asiakasta kuunnellaan, kun toimitusta tehdään. Asiakastyytyväisyys perustuu odotuksiin ennen ostoa ja kokemuksiin oston jälkeen. Se on siis ero, joka syntyy asiakkaan odotusten ja toteutuneen toimituksen välille, joko se on positiivinen eli asiakas sai kaiken paremmin kuin ajatteli tai se on negatiivinen, jolloin asiakas odotti saavansa parempaa. Asiakastyytyväisyys on siis subjektiivinen asiakkaaseen ja asiakkaan odotuksiin nähden. Korkeat odotukset vaativat paremman toimituksen, jotta asiakas on tyytyväinen. Toimituksen laadun mittareista päämittari on asiakastyytyväisyys, jonka avulla voidaan tutkia, oliko toimitus odotusten mukainen ja vastasiko se annettuja vaatimuksia. (Hartini & Mahmoud, 2014, s. 19.)

3 SOVELLUSKEHITYS

Tässä luvussa käsitellään yleisesti tuotekehitystä ja sen tarpeellisuutta kehittyvässä yhteiskunnassa. Sieltä tarkemmin luku käsittelee sovellus- ja ohjelmistokehitystä sekä niiden versiointia ja laadun ylläpitoa. Opinnäytetyössä tutkitaan yhtenä osana sovelluksen uusien versioiden kehitystä ja toimitusta, joten luvussa yhdistetään vielä tuotteen elinkaarimallit mukaan. Eri vaiheessa elinkaarta versioiden tuotantovauhti ja sisältö ovat erilaisia, joten työn kannalta on välttämätöntä tietää, missä vaiheessa tuote on elinkaartaan.

3.1 Tuotekehitys

Tuotekehitys lähtee innovaatiosta, joka on uuden idean jalostamista tuotteeksi markkinoille. Innovaatiot voivat olla aivan uusia markkinoita muuttavia, uuteen tieteelliseen tutkimukseen perustuvia keksintöjä, tuotteita tai palveluita tai ne voivat olla nykyisten palveluiden tai tuotteiden kehittämistä. Kun tiedetään mitä aiotaan kehittää ja on löydetty uusi innovaatio, tuotekehitys lähtee käyntiin. Nykyään tuotteet ja palvelut ovat niin monimutkaisia, että tuotekehitystä ei välttämättä ehditä tai pystytä tekemään ainoastaan yhden yrityksen toimin vaan pitää tehdä yhteistyötä muiden kanssa, jotta tuotteet saadaan markkinoille nopeasti ja kustannustehokkaasti. (Nieminen, 2016, luku 6.)

Tuotteet, jotka eivät ratkaise ongelmia kustannustehokkaasti epäonnistuvat. Tuotekehityksessä on kaksi vaihtoehtoa, joko ymmärtää täydellisesti kaikki osa-alueet tiettyyn aiheeseen liittyvien ongelmien ympäriltä ja ratkaista ne tai kehittää täysin uusi tuote tai ominaisuus ja seurata ratkaiseeko se tarpeeksi ongelmia, jotta asiakkaat ostavat sen. Tämä on paljon riskialttiimpi tapa, koska ratkaisua ei ole suoraan tehty ratkaisemaan tiettyä määrää olemassa olevia ongelmia vaan pyritään löytämään uusia ongelmia, jotka asiakkaat itse ratkaisevat tuotteen avulla. Asiakkaat eivät välttämättä osaa edes kertoa tarvetta mullistavalle tuotteelle, koska eivät mahdollisesti ole edes huomanneet sellaiselle tarvetta. Näissä tapauksissa organisaation pitää tarkkailla asiakkaita ja markkinoita löytääkseen sellaisia ongelmia, joihin tuotteet voisivat tuoda

ratkaisun. Yhteistyötä asiakkaiden kanssa pitää tehdä tuotekehityksessä, jotta voidaan löytää ongelmiin ratkaisuja. Asiakkaat eivät kuitenkaan suoraan anna oikeita vastauksia, koska vastaukset perustuvat heidän omiin kokemuksiinsa. Esimerkiksi julkaisemattoman tuotteen tarpeiden kysyminen asiakkailta voi johtaa siihen, että saadaan paljon unelmia tuotteesta, joiden kehittäminen lopulta tulee kalliiksi, koska asiakkailla ei ole oikeaa kokemusta tuotteen käytöstä vielä eli unelmat tarpeista ylittävät oikeat tarpeet. (Bstieler & Noble, 2023, luku 27.1.)

3.2 Sovelluksen tuotekehitys ja elinkaarimalli

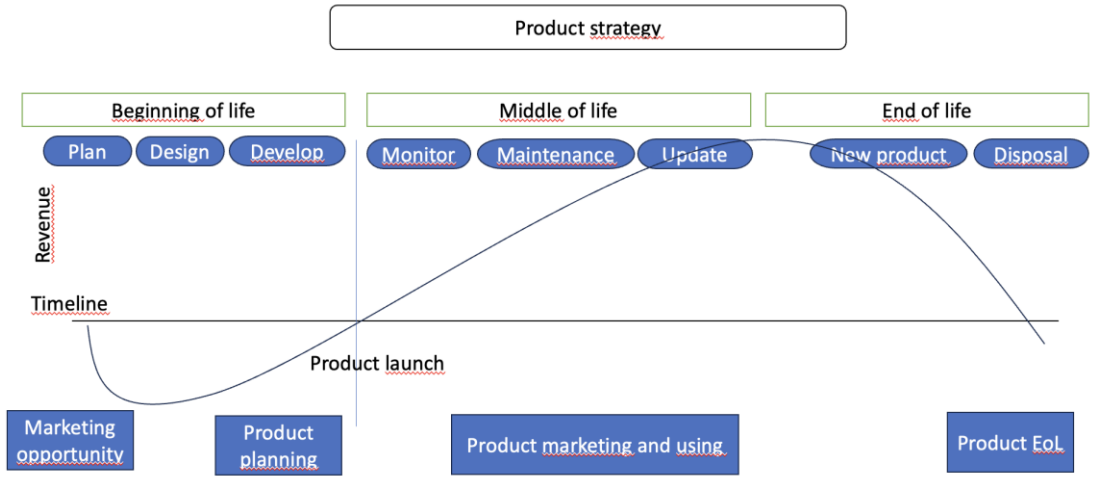
Ennen ero sovellusten ja ohjelmien välillä oli selkeä. Sovellukset asennettiin aina puhelimiin tai tabletteihin ja ohjelmat asennettiin tietokoneelle. Nykyään niiden erot ovat pieniä esimerkiksi Microsoft käyttää molemmista termiä ”apps”. Nykyään erottelu tehdään seuraavalla tavalla. Sovellus on ohjelma, joka hoitaa tiettyjä toimenpiteitä, mutta ei liity järjestelmän ylläpitoon. Ohjelma on sarja käskyjä, jotka ohjaavat tietokonetta. Näitä ei kuitenkaan käyttäjä käytä suoraan. Jokainen sovellus on ohjelma, mutta kaikki ohjelmat eivät ole sovelluksia. Sovellus on alusta, jota käyttäjä käyttää ja sovellus voi ohjata useampia ohjelmia. Ohjelmisto on kokonaisuus, joka koostuu useasta ohjelmasta. (Nielsen, 2022; Kotimaisten kielten keskus, n.d.-a; Kotimaisten kielten keskus, n.d.-b; Sanastokeskus, n.d.; Sanastokeskus, 29.10.2020.)

Tuotteen elinkaarimalli tarkoittaa koko tuotteen elinkaarta tuotteen syntymisestä sen loppumiseen ja mahdollisesti korvaamiseen uudella tuotteella. Tuotteen elinkaari jakautuu kolmeen vaiheeseen: tuotteen alku (Beginning-of-Life BoL), keskivaihe (Middle-of-Life MoL) ja loppu (End-of-Life EoL). Tuotteen alku sisältää ajatuksen tuotteesta, sen tarkan suunnitelman ja toteutuksen sekä tuotteistamisen. Ohjelmistokehityksen osalta tässä vaiheessa tehdään määrittely, suunnittelu, ohjelmointi ja testaus. Keskivaihe tarkoittaa tuotantokäyttöä, ylläpitoa ja tukea käyttäjille. Ohjelmistokehityksen osalta täältä saadaan palautetta sovelluksen kehitykseen, virheenkorjaukseen ja mahdollisesti uuden sovelluksen tai sovellusversion kehitykseen. Keskivaiheessa sovelluksesta

julkaistaan uusia sovellusversioita, jotka eroavat itse uudesta tuotteesta niin, että ne tuovat olemassa olevaan tuotteeseen uusia toiminnallisuuksia tai parannuksia. Loppu sisältää tuotteen korvaamisen uudella tai sen käytön ja kehityksen lopettamisen, kun se ei enää ole hyödyllinen. (Stark, 2011, luku 1.2; Laine & Paakki, n.d., s. 2–7.)

Elinkaarimalli on tärkeä osa sovelluskehitystä ja sovelluksen toimitusprosessia, koska kehittäjien ja päättäjien tulee tietää mitä kehityskohteita on tulossa ja mitä vaatimuksia ne aiheuttavat liitännäissovelluksille eli sovelluksille, joita tarvitaan päivitetyn sovelluksen käyttämiseksi. Esimerkiksi yhden sovelluksen päivittäminen voi aiheuttaa muutoksia myös toisiin sovelluksiin, kuten integraatioihin eli tiedon liikuttamiseen järjestelmästä toiseen. Elinkaarimalleja on useita ja tässä työssä hyödynnetään seuraavia niistä: sovelluskehityksen (Software development lifecycle management, SDLC), tuotekehityksen (Product lifecycle management, PLM) sekä palvelun elinkaarimallia (Service lifecycle management, SLM). Elinkaarimallit eivät kuitenkaan ole huomioineet SoS mallia nykyisissä sovelluksissa ja sen elinkaarta. Uutta SoS elinkaarimallia (System of Systems lifecycle management SoSLM) on tutkittu ja se yhdistää PLM ja SLM malleja sekä huomioi tiedon jaon, mikä on erittäin tärkeää monimutkaisissa järjestelmäympäristöissä. Myös tämän mallin soveltamista tutkitaan kohdeorganisaation näkökulmasta. (Kozma ym., 2021, luvut 2, 8 ja 9.)

Suvannon (2022) mukaan tuotestrategia on suunnitelma, kuinka tuotetta tullaan kehittämään ja markkinoimaan niin, että tuote saavuttaa sille asetetut päämäärät huomioiden muut yrityksen strategiat. Kuviossa 2 kuvataan, miten tuotestrategia etenee markkinoiden löytämisestä tuotteen lopetukseen yhdessä tuotteen elinkaaren kanssa ja mitä toimia missäkin vaiheessa elinkaarta tehdään.



Kuvio 2. Tuotteen elinkaaren vaiheet tuotestrategiaan yhdistettynä (mukaillen Suvanto, 2022; Kozma ym., 2021, kuvio 6)

SDLC malli keskittyy sovelluksen kehitykseen ja sovelluskehityksen malleihin, joista seuraavassa lisää. Tämä ei kuitenkaan huomioi tuotannonaikaista sovellusta eikä pienten muutosten tuomista sovellukseen. PLM on laajempi, kuin SDLC ja se kattaa tuotteen innovaatiosta tuotantokäyttöön ja sieltä tuotteen lopettamiseen. Kuten Mahutin ym. (2015, kuvio 2. PLM and SLM lifecycles) tutkimuksessa esitetään PLM ja SLM ovat melkein samanlaisia malleja, mutta SLM liittyy palveluiden elinkaareen, kuitenkin mallit sisältävät saman tyyppiset vaiheet eli määrittely, suunnittelu, implementointi/kehitys (PLM), käyttö ja poisto markkinoilta. (Kozma ym., 2015, luku 2.)

3.3 Sovellusarkkitehtuuri

Sovellusarkkitehtuuri kuvaa tapaa kehittää sovellusta ja kuinka sovellus on rakennettu. Se antaa parhaat käytännöt sovelluksen kehitykseen ja tiekartan suunnitteluun. Sovellusarkkitehtuureita on useita, mutta tässä työssä käsitellään vain yleisimpiä arkkitehtuureita eli monoliittista-, modulaarista- ja mikropalveluarkkitehtuuria. Suurimmat erot näiden kolmen välillä on ohjelmointikoodin kytkeytyminen kokonaisuuteen sekä eri osuuksien kommunikointi keskenään. (Red Hat, 2020; Rostoniec, 2023.)

Monoliittisessa arkkitehtuurissa koko koodipohja on samaa koodia ja se tarvitsee kaiken koodin, jotta sitä voidaan kääntää tai ajaa. Koodi kuitenkin sisältää kaiken itsessään, joten sen rakentamisessa ei tarvitse huolehtia toisten palveluiden yhteensopivuudesta, kuten esimerkiksi mikropalveluarkkitehtuurin kanssa. Monoliittisessa sovelluksessa yhden pienenkin muutoksen tekeminen vaatii koko ohjelmistokoodin rakentamista uudelleen. Moderneissa arkkitehtuuriratkaisuissa on pyritty ratkaisemaan ongelma rikkomalla palvelut erillisiksi, jotta sovelluskehitykseen saadaan lisää ketteryyttä. (Red Hat, 2020; Rostoniec, 2023.)

Modulaarisuus tekee monoliittisesta sovellusarkkitehtuurista hieman ketterämpää, koska koodi on jaettu esimerkiksi liiketoimintatarpeiden mukaan erillisiksi moduuleiksi. Modulaarinen arkkitehtuuri toimii kuitenkin samalla tavalla, kuin monoliittinen koodin rakentamisen kannalta eli koko sovelluskoodi on aina rakennettava kerralla. Modulaarinen sovellus kuitenkin on sisäisesti ajateltu erillisiksi rakenteiksi. Sovelluskehityksessä esimerkiksi rakennetaan modulaarinen sovellus silloin, kun ei tiedetä tarkalleen pitääkö sovellus rikkoa vuosien päästä mikropalveluiksi vai ei. Modulaarisuus on askeleen lähempänä mikropalveluarkkitehtuuria ja muutos modulaarisesta sovelluksesta mikropalveluarkkitehtuuriin on paljon helpompi, kuin täysin monoliittisen sovelluksen kanssa. (Rostoniec, 2023)

Mikropalveluarkkitehtuurissa sovellus jaetaan pieniin osiin ja jokainen osa sovelluksesta toimii omana mikropalvelunaan. Näin mikropalvelut eivät liity toisiinsa suoraan eli niitä voidaan skaalata, kehittää ja testata yksittäisinä kokonaisuuksinaan. Mikropalveluarkkitehtuurissa on tarkoitus tuottaa laadukasta sovellusta nopeasti, koska kaikkia mikropalveluita voidaan kehittää samanaikaisesti riippumatta toisistaan (eri teknologioiden yhteensopivuus tulee huomioida). Ohjelmistokoodin vieminen tuotantoon ei myöskään vaadi koko ohjelmistokoodin rakentamista, vaan ne voidaan rakentaa yksittäisinä kokonaisuuksina. Mikropalveluissa on tärkeää huomioida, kuinka eri mikropalvelut vaihtavat tietoa ja kuinka tiettyjen mikropalveluiden muutokset vaikuttavat kokonaisarkkitehtuuriin. Esimerkiksi kaikki mikropalvelut voivat toimia erilaisen teknologian päällä, joten mikropalveluarkkitehtuurisen sovelluksen pitää

huomioida eri teknologioiden yhteensopivuus palveluissa. (Red Hat, 2020; Rostoniec, 2023.)

3.4 Sovelluskehityksen mallit

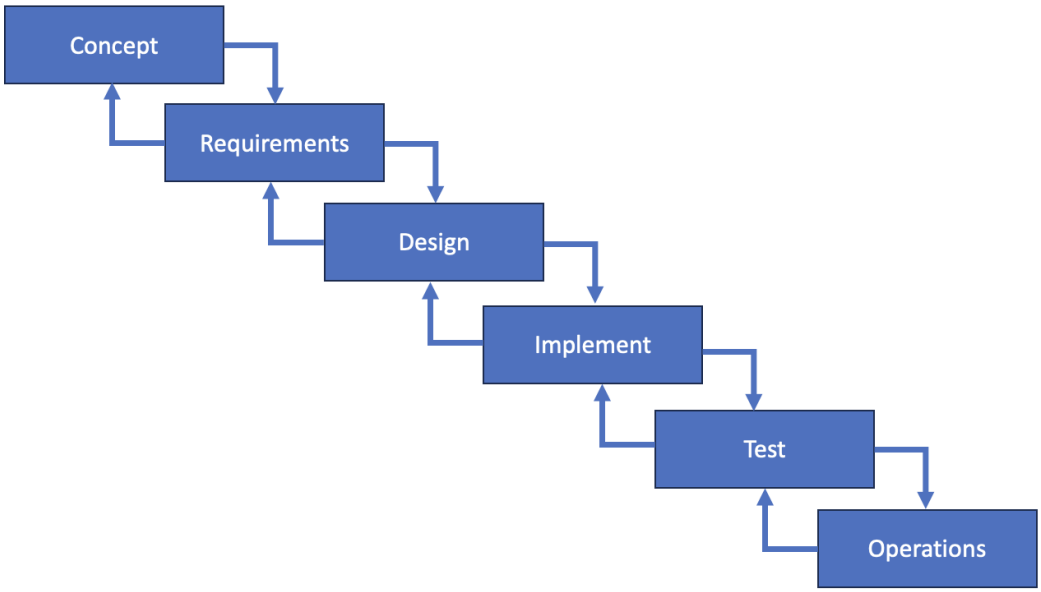
Prosessimalli on graafinen esitys siitä, kuinka prosessi etenee. Niitä voidaan piirtää erilaisia standardeja käyttäen esimerkiksi BPMN 2.0. Prosessimallin avulla voidaan erottaa nykyinen prosessi sekä tavoiteprosessi. Tämä on tärkeää prosessien hallinnassa ja prosessien kehityksessä. Prosessimallin avulla voidaan myös kouluttaa työntekijöitä ja löytää ongelmakohtia nykyisistä prosesseista. (Appian, n.d.)

Sovelluskehitykseen on luotu erilaisia malleja, joiden perusteella sovelluskehitystä tehdään. Nämä auttavat kehittäjiä etenemään askel kerrallaan eteenpäin, jotta kaikki tarvittava dokumentaatio, vaatimukset ja virhekorjaukset tulevat täytettyä. Sovelluskehitysmallit ovat käytännössä lista tai verkko tehtäviä, joita suoritetaan tietyssä kohdassa sovelluskehitysprosessia. Mallit auttavat toteuttamaan ja seuraamaan toteutusta oikea-aikaisesti ja tekemään työt aina samalla tavalla, koko organisaatiossa. Ne on luotu auttamaan erikokoisissa kehityshankkeissa, jotta koko kehitysprosessia voidaan monitoroida ja kehittää. (Krishna ym., 2012, luku 1.)

Tässä työssä rajaus tehdään kolmeen yleisimpään erilaiseen kehitysmalliin, joista yksi on lineaarinen ja kaksi ovat ketteriä malleja. Yleisimmät mallit ovat Dima & Maassenin (2018, s. 316–317) mukaan vesiputousmalli lineaaristen mallien puolelta ja Scrum-malli sekä testivetoinen kehitys ketterien kehitysmallien puolelta, sen vuoksi tässä keskitytään juuri näihin malleihin.

Vesiputousmallissa edetään vaiheittain ja seuraavaan vaiheeseen ei voi siirtyä ennen kuin edellinen vaihe on valmis kuten kuviosta 3 nähdään. Vesiputousmallissa vaatimukset tulevat hallinnolta alaspäin tiimeille ja kommunikaatio asiakkaan kanssa on vähäistä kesken prosessia. Yleensä kehittäjät saavat palautetta vasta tuotekehityksen loppuvaiheessa, minkä vuoksi kesken

ominaisuuden kehitystä kehityssuuntaa ei voida helposti muuttaa. Tämä ei sovellu suurten ketterien kehitysmallien tavoin iteratiiviseen ja nopeaan kehitykseen. (Dima & Maassen, 2018, luvut 1–2; Westfall, 2009, s. 131–132.)

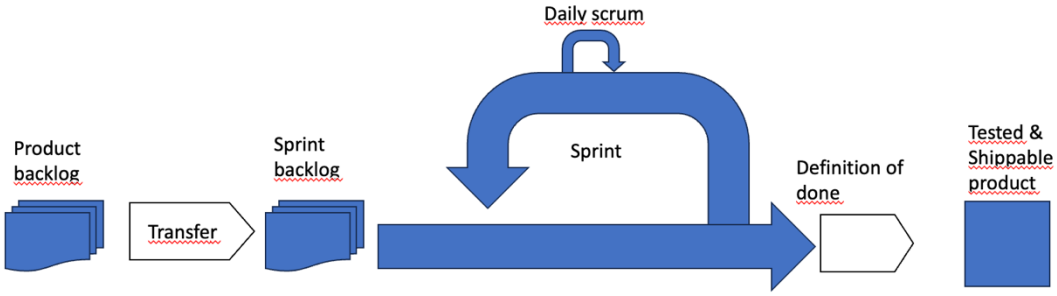


Kuvio 3. Vesiputousmalli (mukaillen Westfall, 2009, kuvio 9.1)

Teknologian kehitys ja nopeampi sovelluskehitysvaatimus on tehnyt tarpeen ketterämmille sovelluskehitysmalleille ja sen vuoksi on luotu iteratiivisia kehitysmalleja kuten scrum. Siinä tarkoitus on kehittää sovellusta lyhyissä kehitysjaksoissa sprinteissä, jolloin voidaan tuottaa ominaisuuksia nopeammassa tahdissa. Näin myös ominaisuuksien testaus on nopeampaa ja palautetta saadaan asiakkaalta kesken kehitysprosessin ja sitä voidaan hyödyntää jo seuraavassa sprintissä. (Dima & Maassen, 2018, luvut 1–2; Westfall, 2009, s. 131–132.)

Scrum prosessissa kehitys jaksotetaan sprintteihin, jotka ovat yhtä pitkiä. Kaikista sprinteistä syntyy aina yksi tai useampi inkrementti, joka tässä tapauksessa tarkoittaa valmista tuotteen osaa, joka on testattu ja läpäissyt definition of done -seulan (jatkossa DoD). DoD tarkoittaa, että tuotteen kehitysjonosta valmistunut osa on testattu ja se toimii määritellyn mukaan. Ennen ominaisuuden kehitystä se siirretään tuotteen kehitysjonosta sprintin kehitysjonoon ja sille määritellään DoD. Sprintin suunnitteluosuudessa (Sprint planning ja

review) käydään edellinen sprint läpi ja valitaan tuotteen kehitysjonosta kehitettävät ominaisuudet sprintin kehitysjonoon ja tämän jälkeen sprintille määritellään, miksi se toteutetaan ja mikä on sprintin tarkoitus, minkä jälkeen uusi sprint alkaa. Kesken sprinttiä työmäärää ei lisätä sen työjonoon vaan seuraava vaihe lisätä töitä on seuraavan sprintin suunnitteluvaihe. Joka päivä kehitystiimi pitää scrum-palaverin, jonka tarkoitus on seurata sprintin edistymistä, optimoida kehitystä ja tuoda sen työjonosta töitä kehittäjille. Kuviossa 4 on kuvattu scrum prosessi ja sen vaiheet. (Linz, 2014, kohta 2.1; Schwaber & Sutherland, 2020, s. 7–12.)



Kuvio 4. Scrum toimintaprosessi (mukaillen Linz, 2014, kuvio 2-1)

Testivetoisessa kehitysmallissa sovelluksen testit ovat kirjoitettu jo ennen, kuin sovellusta aloitetaan ohjelmoida. Ohjelmoinnin jälkeen testi ajetaan ja tarkistetaan tulos, mikäli testi onnistuu, niin ominaisuus toimitetaan. Testivetoinen kehitysmalli koostuu viidestä askeleesta. Ensin tehdään uusi testi ja sen jälkeen ajetaan kaikki testit, jolloin nähdään uusimman testin epäonnistuminen. Ensimmäisessä vaiheessa testin pitää epäonnistua, jotta nähdään, että toteutettu uusi testi toteuttamattomalle ominaisuudelle toimii oikein. Seuraavaksi tehdään pieniä muutoksia ohjelmakoodiin niin, että kaikki testit saadaan onnistuneesti ajettua. Viimeinen vaihe on muokata ohjelmakoodia ja poistaa ylimääräiset kaksoiskappaleet sieltä. Tässä mallissa huomataan jo testiä tehdessä, että onko tavoiteltava seuraava kehitys liian suuri ja voiko sen laittaa vielä pienempiin palasiin. (Paranj, 2017, luku 1; Dima & Maassen, 2018, s. 318.)

Dima & Maassen (2018, luku 6) tutkimuksen mukaan vesiputousmalliset sovelluskehitysmallit ovat nykyään vaihtuneet ketteriin malleihin ja jo vuonna 2013 suurin osa tutkimukseen osallistuneista yrityksistä käytti ketterää mallia. Muutoksen syynä nähdään nopeampi palaute asiakkailta verrattuna vesiputousmalliin. Ketterässä kehitysmallissa ominaisuudet saadaan nopeammin käyttäjille testattavaksi ja sen palautteen avulla niiden iterointi onnistuu aikaisemmassa vaiheessa kehitystä.

3.5 Sovelluksen laatu ja testaus

Sovelluksen laatu on subjektiivista ja se riippuu näkökulmasta, josta sitä tarkkaillaan. Näkökulmia voi olla asiakas eli yritys, joka vastanottaa sovelluksen, ylläpitäjät, jotka ylläpitävät esimerkiksi sovelluksen palvelimia pystyssä ja pitävät huolta, että tietoverkko ja yhteydet toimivat ja viimeisenä sovelluksen käyttäjät, joka käyttävät sovellusta. Kaikki nämä ryhmät näkevät sovelluksen laadun eri tavalla. Asiakasyritys näkee laadun kustannustehokkuutena ja toimitusvarmuutena. Sovelluksen ylläpitäjät näkevät sen sovelluksen sisäisten osien keskinäisenä toimivuutena. Käyttäjille se tarkoittaa toiminnallisuuksia, käytettävyyttä, luotettavuutta ja tarkkuutta. (Laporte & April, 2018, luku 1.4.)

ISO 25010 laatustandardi kuvailee laadun liittyvän kahdeksaan eri tekijään; toiminnallinen sopivuus, käytettävyys, tehokkuus, yhteensopivuus, luotettavuus, tietoturva, ylläpidettävyys ja liikuteltavuus (ISO2500, n.d.). Näistä kaksi ensimmäistä ovat enemmän käyttöön liittyviä laatustandardeja ja kuusi jälkimmäistä tekniseen laatuun liittyviä standardeja.

Toiminnallinen sopivuus tarkoittaa, kuinka sovelluksen toiminnallisuudet kattavat ne tarpeet, joita sovellukselta vaaditaan sekä ovatko sovelluksen antamat tulokset oikeita ja siihen tarpeeseen vastaavia, johon sovellusta käytetään. Käytettävyydellä selvitetään, kuinka helposti sovelluksen käyttö onnistuu ja voiko sen oppia nopeasti. Kuinka se ohjaa käyttäjää tekemään prosessit oikealla tavalla ja avustaa virhetilanteissa sekä voiko sovellusta käyttää

erityistarpeita vaativa käyttäjä esimerkiksi värisokea henkilö? (ISO2500, n.d.; Laporte & April, 2018, luku 3.2.3.)

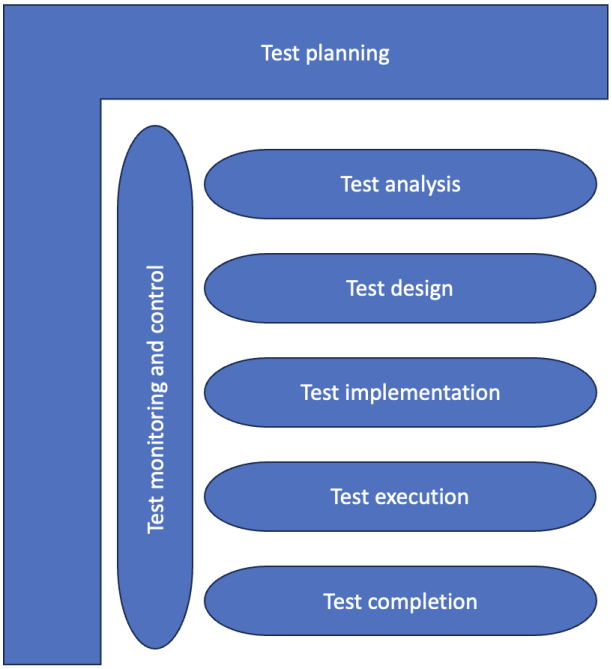
Tehokkuudella tarkoitetaan aikaa ja resurssia, joka kuluu tietyn toimenpiteen tekemiseen. Kuinka hyvin sovellus on liitettävissä muihin järjestelmiin eli toimiiko tiedonvaihto hyvin eri järjestelmien välillä ja voidaanko sovelluksia käyttää esimerkiksi jaetun resurssin päällä kuten samalla palvelimella muiden sovellusten kanssa. Luotettavuus tarkoittaa tuleeko sovelluksen käytössä häiriöitä ja kuinka se palautuu häiriöiden sattuessa, sekä onko se käytettävissä, vaikka häiriö tulisikin. Tietoturvalla tarkoitetaan tiedon tallennuksen luotettavuutta ja käyttäjien pääsyä sovellukseen. Voidaanko tietoa tallentaa luotettavasti niin, etteivät ulkopuoliset saa tietoa käsiinsä tai pääse käyttämään sovellusta? Sillä tarkoitetaan myös tehtyjen toimien kartoittamista takaisin tekijään eli voidaan tutkia, mitä muutoksia kukakin on tehnyt. Ylläpidettävyydellä tarkoitetaan sovelluksen modulaarisuutta eli onko sovellus yksi iso paketti vai toimiiko sovellus monessa pienessä moduulissa, jotka voivat olla riippumattomia toisistaan. Se tarkoittaa myös voidaanko sovellusta ja sen toimintaa analysoida, testata sekä muokata helposti. Liikkuvuus on sitä, voidaanko sovellus asentaa ja poistaa helposti erilaisista ympäristöistä vai tarvitseeko asennus ja käyttöympäristö olla aina samanlainen. Se myös mittaa onko sovellus helposti korvattavissa toisella samanlaisella sovelluksella vai pitääkö tehdä suuria toimenpiteitä, mikäli joskus haluaa siirtyä esimerkiksi kilpailevan sovelluksen käyttöön. (ISO2500, n.d.; Laporte & April, 2018, luku 3.2.3.)

Sovelluskehityksessä laadun seuranta on kriittinen siksi, että sovelluksen laatu heikkenee ajan kuluessa. Jatkuvat ylläpitotyöt ja uudet ominaisuudet aiheuttavat mahdollisesti muutoksia jo toimivaan osaan sovellusta, joten mikäli niitä ei testata jatkuvasti niin laatu heikkenee. Teknologian kehittyessä, sovelluksen käytettävyys, joka on yksi laadun mittari, voi laskea. Alustapalvelimen version päivitys voi aiheuttaa sovelluksen toimimattomuuden ja laadun heikennystä. Näiden ongelmien estämiseksi ja laadun heikkenemisen vähentämiseksi on luotu laadunhallintamalleja (Quality assurance, QA), joissa testataan ja analysoidaan toteutettu toiminnallisuus, minkä jälkeen se versioidaan eteenpäin. QA ylläpitää tuotteen laadullista tasoa ja selvittää ongelmat mahdollisimman

aikaisessa vaiheessa. (Wagner, 2013, luku 4.) Mikäli sovelluksella ei voida enää toteuttaa sitä tarkoitusta, johon se on luotu esimerkiksi asiakkuudenhallintajärjestelmä ei kykenisi käsittelemään aurinkopaneeleja, niin sen käytettävyys ja laatu ovat laskeneet.

Virheet ohjelmasta löytyvät yleensä kahdella tavalla, joko sovelluksen käytössä eli silloin, kun tuote on jo tuotannossa ja siinä on jokin häiriö tai testausvaiheessa. Testauksessa tavoitteena on löytää virheet mahdollisimman aikaisessa vaiheessa, jolloin niiden korjaaminen on kustannustehokkaampaa, koska virhe voidaan korjata jo ennen, kuin sovellus käyttöönotetaan tuotannossa. Raghavanin (2021, s. 142) artikkelissa haastateltava Smith mainitsee, että kilpailukyvyn ylläpitämiseksi ominaisuuksien nopea julkaisutahti aiheuttaa sen, ettei kaikkia sovelluksen käyttötapauksia voida testata. Sovelluskehityksessä ja tuotetoimituksessa on pakko tehdä kompromisseja testauksen suhteen ja valita, mitä testataan. Yksikään sovellus ei ole virheetön vaan virheitä syntyy aina. Yleensä syy on jokin seuraavista: inhimillinen ohjelmointivirhe, liian kiireinen aikataulu, sovelluksen tai toiminnallisuuden monimutkaisuus, väärin ymmärrys liiketoimintatarpeen ja ohjelmoijien välillä, uudet teknologiat tai asiantuntijan kokemus. (Spillner & Linz, 2021, luku 2.1.)

Testausprosessi pitää suunnitella tarkkaan ja tehdä näkyväksi, jotta testaus varmasti suoritetaan ja voidaan seurata mitä on testattu. Testaus sellaisenaan ei kerro mitään. Se pitää rikkoa pienemmiksi palasiksi, jotta koko prosessi tulee huomioiduksi testauksen kaikissa vaiheissa systemaattisesti. Testaussuunnitelma kattaa koko testausprosessin kaikki vaiheet kuten kuviosta 5 nähdään. Sitä pitää myös päivittää ajan kuluessa, koska osa suunnitelmasta voi vanhentua sovelluksen muuttuessa. Testitapauksien seuranta on tärkeää, jotta nähdään testauksen laatu ja löytyvätkö tulevat virheet testauksella. Ilman sitä ei voida päivittää testisuunnitelmaa.



Kuvio 5. Testausprosessi (mukaillen Spillner & Linz, 2021, kuvio 2-3)

Testauksia on kahden tyyppisiä. On toiminnallisia testejä, joissa testataan suunniteltuja toimintoja esimerkiksi, kun haetaan asiakkaan tietoja, niin nähdään, palautetaanko ne oikein. Ei-toiminnalliset testit ovat toinen testityyppi, jossa testataan, kuinka hyvin sovellus suoriutuu toimenpiteistä. Ei-toiminnallisilla testeillä mitataan suorituskykyä, kuormitusta, vakautta ja käytettävyyttä. Testejä suunnitellessa on huomioitava, onko sovellukseen tullut uusia ominaisuuksia tai onko virheitä löytynyt edellisessä testauksessa. Mikäli uusia versioita tulee, niin testitapauksiin pitää lisätä uudet lisätyt ominaisuudet ja löydettyjen virheiden testit pitää ajaa uudelleen, jotta voidaan olla varmoja, etteivät ne uusiudu. (Spillner & Linz, 2021, luvut 3.5–3.6.)

Yksikkötestit ovat kaikkein pienimmät testit, joita koodiin ajetaan. Ne ovat eristetty vain testaamaan yksittäistä tiettyä toimenpidettä. Yksikkötestien lukumäärä ei välttämättä takaa parempaa testitulosta, vaan yksikkötestien järkevöittämisellä voidaan saada parempia tuloksia. (Smartbear, n.d.)

3.6 Muutosten- ja versionhallinta

Muutostenhallinta on suuri osa sovelluskehitystä ja sen avulla tiedetään, mitä ominaisuuksia kehitetään tai on kehitetty ja milloin. Näin tieto saadaan myös käyttäjille paremmin, joten he voivat odottaa tiettyä muutosta saapuvaksi tiettynä hetkenä. Samoin myös virheenkorjausten korjauspäivitykset voidaan kalenteroida ja ottaa mukaan normaaliin ketterään sovelluskehitykseen. Versionhallinta sisältää tiedon, mitä koodia lisättiin missäkin vaiheessa. Se myös sisältää työkalut hallita tilanteita, kun samaa ohjelmistokoodia muutetaan samanaikaisesti kahden eri ohjelmoijan toimesta (Merge conflict). Tässä tapauksessa pitää joko valita toisen ohjelmoijan koodi ja poistaa toisen tai yleisimmin ne yhdistetään. Yhdistäminen voidaan tehdä, mikäli muutoksia on tehty samaan kokonaisuuteen, mutta ei täysin samaan kohtaan, jolloin toinen ohjelmoija on voinut kehittää yhtä osa-aluetta, joka tukee toisen ohjelmoimaa osa-aluetta. Ohjelmistojen versioiden sisältämät koodit ylläpidetään versionhallinnassa, jonka avulla voidaan peruuttaa muutoksia, mikäli jotain koodia on muutettu tai poistettu virheellisesti. Muutostenhallinta on siis prosessi, kuinka ja mitä ominaisuuksia milloinkin tuodaan ja versionhallinta sisältää ohjelmistokoodin muutokset ja niiden versioiden seurannan. (Bojana & Anestas, 2014, luvut 4–5.)

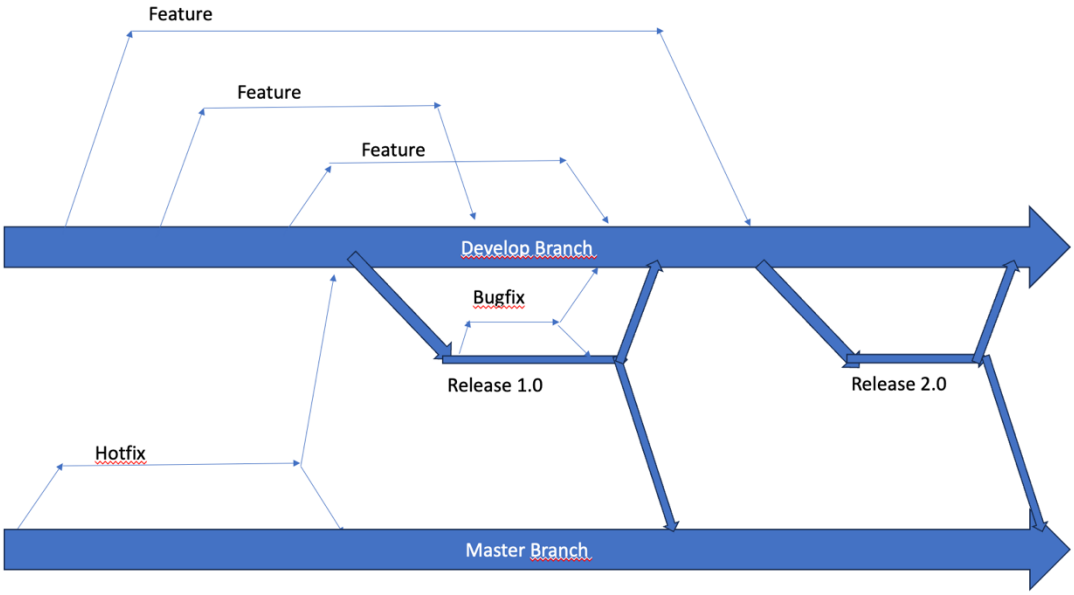
Sovellusta säilötään versionhallintasovelluksessa, jonka kautta sovelluksesta tehdään versioita, joita voidaan julkaista asiakkaille. Kaikki sovelluskehittäjät kehittävät tiettyä sovelluksen ohjelmistokoodia. Kun kaikki suunnitellut ominaisuudet on toteutettu, koodista irrotetaan uusi haara (Branch). Se nimetään erikseen versionhallinnassa julkaisuksi (Release) ja siihen ei enää tehdä kehitystä vaan sitä jatketaan alkuperäiseen koodiin. Julkaisut testataan ja niistä voidaan löytää virheitä. Ainoastaan virheenkorjaukset viedään julkaistuun haaraan korjauspäivityksinä. Sovelluksen versiointia kutsutaan versiohallinnassa haaroitukseksi ja sitä voidaan toteuttaa erilaisten strategioiden avulla, joko yksinkertaisesti lisäämällä aina pääversiosta uusi haara uudeksi versioksi tai moniulotteisesti lisäämällä eri haarat kehitykselle, tuotantoversiolle, ominaisuuksille ja korjauksille. Haaroitusstrategia valitaan yleensä yrityksen koon perusteella. Mikäli ohjelmoijia on paljon, niin yleisesti käytetään moniulotteista

haaroitusta. Kohdeorganisaatiolla on myös käytössä moniulotteinen haaroitusstrategia. (Laporte & April, 2018, luku 8.)

Haaroituksia voi olla esimerkiksi päähaarassa kehittäminen (trunk based development) tai ominaisuusvetoinen kehittäminen (feature-driven development). Päähaarassa kehityksessä käytetään ainoastaan yhtä päähaaraa, johon kaikki ohjelmoijat lisäävät koodiaan ja se yhdistetään takaisin vähintään kerran päivässä. Ominaisuusvetoisessa kehityksessä päähaarasta otetaan uusi haara, jota voidaan kehittää useamman kehittäjän toimesta, kunnes ominaisuus on valmis. Tämän jälkeen ominaisuus yhdistetään takaisin päähaaraan. Erona näissä kahdessa on, että päähaarassa kehityksessä kehittäjillä ei ole omalla työasemallaan paikallista haaraa ohjelmakoodista kovinkaan pitkään, kun se pitää yhdistää takaisin päähaaraan. Ominaisuusvetoisessa kehitysmallissa kehittäjällä voi olla paikallinen haara omalla työasemallaan tai irti päähaarasta useamman viikon ja sitä samaa haaraa voi kehittää useampi kehittäjä ennen kuin se on valmis ja yhdistetään takaisin päähaaraan. (Schmitt, 2023; Dora, n.d.; Merode, 2023, luku 4.)

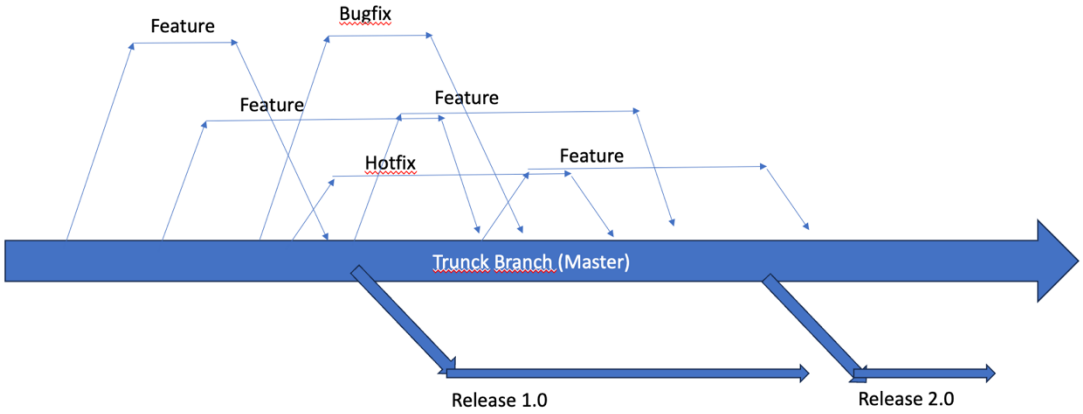
Kuviossa 6 on kuvattu Gitflow haaroitusstrategia, jota hyödynnetään ominaisuusvetoisessa kehityksessä. Siinä on kaksi päähaaraa pää (Master) ja kehitys (Development), joista vain kehityshaaraa kehitetään kehittäjien toimesta ja sieltä viedään valmiit toimivat kokonaisuudet julkaisujen (Release) kautta päähaaraan. Ominaisuudet (Feature) kehitetään aina kehityshaarasta otetuissa ominaisuushaaroissa, joista ne yhdistetään takaisin kehityshaaraan, josta myöhemmässä vaiheessa haaroitetaan toimitettava versio. Versiohaaroihin ei viedä mitään muuta uutta kehitystä, kuin virheenkorjaukset (Hotfix), jotka viedään sen lisäksi myös kehityshaaraan, jotta koodi ei heikkene, kun uusia versioita haaroitetaan. Tässä versiostrategiassa hyvänä puolena on, että kaikki kehittäjät voivat kehittää ominaisuuksia omissa kehitysympäristöissään, joissa ne eivät vaikuta tuotteen kehitykseen ja haaroitukseen. Kehitys ja versiointi eivät teknisesti estä toinen toisiaan, koska ominaisuuskehitys on täysin irrallaan eri haaroissa. Päähaarassa on myös aina toimiva versio, joka voidaan tarvittaessa haaroittaa. Ongelmana Gitflow-haaroituksessa on, kun kaikki ominaisuudet viedään takaisin kehityshaaraan ja pitäisi julkaista uusi versio.

Kehittäjät ovat voineet muokata yhteisiä koodeja, mikä aiheuttaa ristiriitoja niiden yhdistämisessä. Näiden ratkaisemiseksi pitää varata aikaa aina uuden version haaroituksen yhteydessä, mikäli monta ominaisuutta kehitetään samanaikaisesti samaan julkaistavaan versioon. (Microsoft 2022; Manandhar, 2021; Driessen, 2010; Topdevs, 2021.)



Kuvio 6. Gitflow ominaisuusvetoinen kehittäminen (mukaillen Microsoft 2022; Manandhar, 2021; Driessen, 2010; Topdevs, 2021)

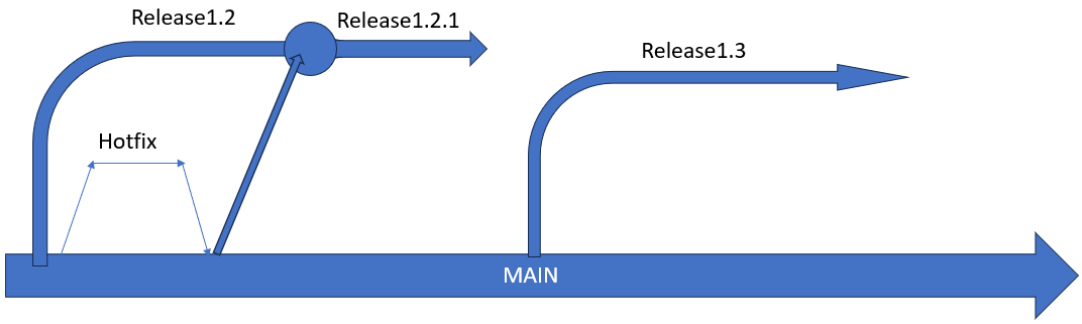
Kuviossa 7 on kuvattu päähaarassa kehittämisen strategia. Siinä kaikki kehitys viedään aina samaan haaraan, oli kyse ominaisuuksista tai virheenkorjauksista. Päähaarasta luodaan uusi julkaistava versio, johon ei tehdä lainkaan kehitystä vaan korjauspäivityksetkin tehdään päähaaraan, josta ne tulevat seuraavaan versioon sisällytetyiksi. Kehityshaarat pyritään pitämään lyhyinä vain muutaman päivän mittaisina. Näin lisätyn koodin määrä on pieni ja automaattitestaukset sekä koodin läpikäynti onnistuvat nopeasti eikä ristiriitoja koodin yhdistämisestä tule. Kaikkein tärkeintä on huolehtia siitä, että kaikki muokkaukset ovat toimivia. Yksittäinen koodivirhe voi pysäyttää koko kehityksen kaikilta kehittäjiltä, koska kehitys tehdään aina samaan päähaaraan. (Manandhar, 2021.)



Kuvio 7. Päähaarassa kehittämisen versiostrategia (mukaillen Manandhar, 2021)

Opinnäytetyössä pitää huomioida, että on olemassa myös useampia haaroitusstrategioita ja niitä voidaan soveltaa eri tavalla eri organisaatioissa. Tässä työssä keskitytään kuitenkin näihin kahteen strategiaan, koska ne ovat yleisesti tunnettuja.

Versionhallinnassa organisaatiolla on sovelletusti käytössä päähaarassa kehittäminen. Kaikki muutokset tehdään päähaarassa, josta erotellaan erilliset versiot aina omaksi haarakseen. Versiokorjaukset toteutetaan myös päähaarassa ja valmistuessaan korjaus viedään päähaaraan ja siihen versiohaaraan, johon korjaus on suunniteltu. Versohaarasta ei kuitenkaan luoda uutta haara vaan korjaus merkitään (tagging) haaraan uudeksi versioksi. Kuviossa 8 näkyy version julkaisumalli. Version tuki jatkuu vuoden eteenpäin, joten asiakkaiden tulee vähintään kerran vuodessa ottaa uusi versio käyttöön.



Kuvio 8. Sovellusversion haaroitusmalli organisaatiolla

Vuonna 2023 organisaatio otti käyttöönsä uuden julkaisusyklin, jonka mukaan versioita julkaistaan nyt seitsemän kertaa vuodessa. Ennen versioita julkaistiin neljä kertaa vuodessa. Tämän muutoksen taustalla oli tuoda asiakkaiden tarvitsemat muutokset nopeammin saataville sekä kehityksen helpottaminen, kun uusien ominaisuuksien suunnittelu tehdään pienemmälle aikavälille. Myös erillisten uusia ominaisuuksia sisältävien palvelupäivitysten julkaisu lopetettiin ja ainoastaan korjauspäivityksiä julkaistaan. Uudet ominaisuudet julkaistaan uusissa versioissa. Pienemmät versiot vähentävät myös testauskuormaa, kun versiossa on vähemmän testattavaa. Asiakkaat voivat myös valita paremmin, minkä version ottavat käyttöön, kun näkevät missä versiossa heidän tarvitsemansa ominaisuudet ovat tulossa.

Kehitysjakson alussa versiosta luodaan ennakkoversiotiedote, joka toimitetaan asiakkaille. Tämän avulla he voivat tutustua uuden version tuomiin ominaisuuksiin. Kaikki ennakkoversiotiedotteessa olevat ominaisuudet eivät kuitenkaan välttämättä ehdi valmistua, joten versiojulkaisun yhteydessä luodaan versiotiedote, joka sisältää juuri julkaistun version kaikki tulleet uudet ominaisuudet. Tämän avulla asiakkaat yleensä tekevät lopullisen päätöksen version hankinnasta.

4 TOIMITUSPROSESSI

Tässä luvussa käsitellään tuotteen toimitusprosessia ja siinä keskitytään sovelluksen toimittamiseen sekä erilaisiin toimitusympäristöihin, kuten SaaS, Palvelu ja OnPremise. Ensin käydään läpi yleisesti, mitä sovellustoimitus sisältää ja minkälainen prosessi se on. Tämän jälkeen käsitellään development and operations eli DevOps ja sen rooli sovellustoimituksissa, minkä jälkeen käsitellään mikropalveluiden ja monoliittisen toimituksen erot. Viimeisenä asiana on CI/CD prosessi ja sen merkitys sovellustoimituksissa.

4.1 Sovellustoimituksen ominaisuudet

Ohjelmistotoimituksessa ei ole alan standardia, vaan ihmiset ymmärtävät eri tavalla sen vaiheet. Näitä vaiheita ovat rakentaminen (build), julkaisu (release), käyttöönotto (deployment) ja käyttö. Rakentamisessa ohjelmistosta tehdään julkaistava versioitu paketti, jonka sisältö dokumentoidaan ja kerrotaan julkaisuvaiheessa. Käyttöönotto ja käyttö eroavat toisistaan, koska ensin ohjelmisto pitää viedä tuotantoympäristöön ja vasta sen jälkeen siihen voidaan päästää käyttäjät käyttämään sitä. Toimitusprosessi ei kuitenkaan pääty tähän, vaan käytön aikana käyttäjiltä kerätään palautetta ja kehitysehdotuksia, joita hyödynnetään seuraavassa versiossa. (Riccomini & Ryaboyn, 2021, kohta Software delivery phases.)

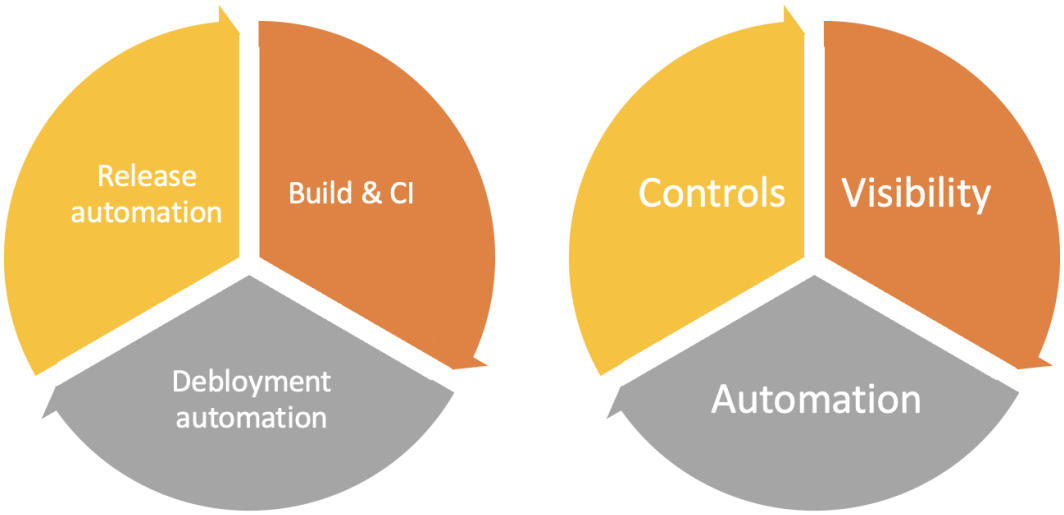
Sovelluksen toimitus tarkoittaa, että valmis sovellus tai versio toimitetaan käyttäjille. Näissä tapauksissa yleensä käyttäjillä on lyhyt käyttökatko, koska tuotantokäytössä olevaa sovellusta ei voida käyttää toimituksen aikana. Jatkuva integrointi ja toimitus (CI/CD), jossa voidaan toimittaa uusia versioita ilman suurempia käyttökatkoja auttaa niissä tapauksissa, joissa käyttökatkoja ei saa olla ja sovellukseen tuodaan jatkuvasti uusia ominaisuuksia. Tästä kerrotaan luvussa 4.4. Sovellustoimituksessa yhdistetään yleensä kehitys ja käyttö (Development and Operations tai Developer Operations, jatkossa DevOps). Näiden avulla toimitusprosessi saadaan lähemmäs ketterän sovelluskehityksen mallia ja virheet ohjelmistoissa jäävät kiinni nopeammin. DevOpsin tarkoitus

on lisätä kommunikointia ja yhteistyötä sovellusta toimittavien ja ohjelmoivien tekijöiden välille. Se myös pyrkii tuomaan automaatiota sovelluskehitykseen, -testaukseen ja -toimitukseen, jolloin voitaisiin toimittaa valmiit versiot automaattisesti testattuina tuotantoon. Yhteistyön lisääntyessä kehityksen ja toimituksen välillä myös sovelluksen laadunmittaus paranee, koska sovelluskehityksen monitorointi etenee sovelluskehityksestä sen toimittamiseen asiakkaalle asti. (Salameh, 2019, s. 22–23.)

Ennen vesiputousmallilla kehitetyt sovellukset ja niiden versiot ovat siirtyneet ketterään kehitykseen ja useista nopeista sovellustoimituksista on tullut menestymisen merkki. Lwakataren ym. (2016, s. 91) mukaan suuret yritykset, kuten Amazon, Google ja Facebook ovat ylittäneet asiakkaiden odotukset toimittamalla asiakkaan vaatimukset nopeasti. CI/CD-prosessi on tuonut enemmän vastuuta DevOps:lle rikkoa sovelluskehittäjien ja -toimittajien välistä kuilua. Sen tarkoitus on yhtenäistää toimitusprosessi sovelluskehityksestä sovellustoimitukseen ja tehdä se läpinäkyväksi kaikille prosessissa toimiville osapuolille, jotta voidaan toimittaa luotettavasti ja nopeasti uutta ohjelmistokoodia. (Salameh, 2019, s. 23; Lwakatare ym., 2016, s. 93.)

Sovellustoimituksessa DevOps-käsitteessä on kaksi erilaista ulottuvuutta, joista toinen liittyy sovellustoimituksen elinkaareen ja toinen liittyy toimitusvaatimuksiin. Nämä ulottuvuudet ovat kuvattuna kuviossa 9. Ensimmäinen ulottuvuus sisältää seuraavat vaiheet: sovelluksen ohjelmoinnin ja jatkuvan integraation sekä toimituksen ja julkaisun automatisoinnin. Ensimmäinen vaihe sisältää ominaisuuksien ohjelmoinnin ja versionhallinnan, jonka mukaan seuraava versio tuotetaan. Se pitää siis sisällään ohjelmistokoodin, joka kuuluu seuraavaan toteutettavaan versioon. Seuraava vaihe pitää sisällään infrastruktuurin eli ympäristön, kuten tietoverkon, palvelimet ja käyttöjärjestelmän vaatimukset toteuttaa versiotoimitukset aina samalla tavalla. Viimeinen vaihe sisältää julkaisun automaation, joka huolehtii myös muut järjestelmät, kuin itse sovelluksen asennukset ja käyttöönotot. Esimerkiksi tietokantojen ja integraatioiden muutokset julkaisun yhteydessä sisältyvät siihen. Tämä vaihe pitää sisällään myös laatuportit, joiden pitää täyttyä, jotta sovelluksen julkaisua voidaan viedä eteenpäin. Se tarkoittaa automaattisten testien ajamista

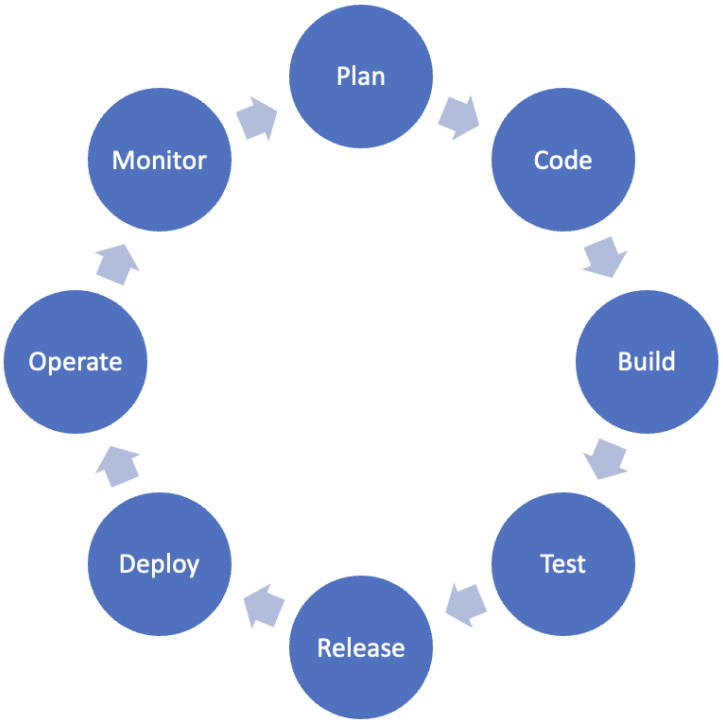
sovellukseen. Testien pitää tietyllä prosentilla onnistua ennen kuin voidaan edetä seuraavaan vaiheeseen ja lopulta tuotantojulkaisuun asti. (Salameh, 2019, luku 4.3; Goerdt, 2017.)



Kuvio 9. DevOps ulottuvuudet (mukaillen Goerdt, 2017)

Toinen ulottuvuus käsittelee automaation kyvykkyyksiä eli toimituksen pitää olla toistettava, nopea ja skaalautuva. Se pitää pystyä ajamaan useasti samalla tavalla ilman, että se vaatii manuaalisia toimenpiteitä. Sen pitää myös olla nopea ja skaalautuva eli se pitää pystyä ajamaan ja mukauttamaan nopeasti erilaisiin ympäristöihin. Toinen ulottuvuus pitää sisällään seuraavat vaiheet: automaatio, hallinta ja näkyvyys. Toistettavan prosessin pääpiirre on automaatio, jonka avulla manuaaliset työvaiheet saadaan toimituksesta pois ja säästetään resursseja. Sen tulee kuitenkin olla hallittavissa eli ongelmatilanteissa tieto pitää saada eteenpäin tekijöille välittömästi ja näissä kannattaakin yhdistää prosessi suoraan työnhallintajärjestelmiin, joiden avulla saadaan heti ongelmatilanteista virheenkorjaustehtävät asiantuntijoille. Viimeisessä vaiheessa eli näkyvyydessä huolehditaan, että tiedetään jatkuvasti mitä prosessi tekee ja missä vaiheessa prosessia ongelmia on tullut. Näin voidaan seurata koko prosessin toiminta- ja suorituskykyä. Yrityksen johto voi seurata tapahtumia erilaisten koontinäyttöjen avulla. Niissä esimerkiksi näkyy toimitusten laatu eli nähdään missä vaiheessa virheitä on syntynyt ja onko virheiden määrä laskenut. DevOps on jatkuva kierto kahdeksan eri kohdan ympärillä näiden ulottuvuuksien sisällä, kuten kuviossa 10 on kuvattu. Nämä kierroksen vaiheet

ovat: suunnittelu, ohjelmointi, sovelluksen rakentaminen, testaus, julkaisun tekeminen, julkaisun jakelu, operointi ja monitorointi. (Salameh, 2019, luku 4.3; Goerdt, 2017; Lwakatare ym., 2016, s. 97; Yarlagadda, 2021, s. 115.)



Kuvio 10. DevOps kierrokset CI/CD prosessissa (mukaillen Yarlagadda, 2021, s. 115)

DevOps alkaa suunnittelusta, jossa saadaan edelliseltä kierrokselta palautetta ja löydetyt ongelmat voidaan suunnitella korjattaviksi. Siinä myös päätetään mitä seuraavalla kierroksella toteutetaan. Ohjelmointi- ja rakennusvaiheessa tehdään ensin muutokset koodiin ja sitten ohjelma rakennetaan ja tarkastetaan, löytyykö ohjelmoinnista virheitä. Seuraavaksi tehdään testaus, jossa automaattisesti testataan mahdollinen julkaisu ja sen toiminnallisuudet. Virheettömät testaukset etenevät julkaisuun ja tuotantotoimitukseen. Tuotannossa sovellusta käytetään ja seurataan, mikäli virheitä tai parannusehdotuksia esiintyy. Näiden perusteella taas suunnitellaan seuraavaa kierrosta. (Atlassian, n.d.; Yarlaggadda, 2021, s. 115–117.)

4.2 Palvelu, SaaS, OnPremise

Järjestelmä voidaan toimittaa kolmella eri tavalla, jotka ovat palveluna, järjestelmäpalveluna (Software as a Service eli SaaS) sekä OnPremisenä. Palveluiden avulla asiakkaat ostavat toimittajalta palvelua, jonka toimittaja toteuttaa asiakkaalle. SaaS tarkoittaa, että asiakas itse käyttää järjestelmää, mutta se on asennettuna toimittajan konesaleihin eli toimittaja pitää huolta siitä, että järjestelmä toimii ja asiakas voi käyttää sitä omassa liiketoiminnassaan. OnPremise-asennuksessa toimittaja asentaa sovelluksen asiakkaan omiin palvelinsaleihin, jolloin toimittajan vastuulla on vain sovelluksen asennus ja mahdollisesti päivitykset. Asiakas itse hoitaa järjestelmän käytön ja palvelinten ylläpidon niin, että järjestelmä on käytettävissä. (Valtori, n.d.; Familiar, 2015, luku 1.)

SaaS malli on jatkuvassa nousussa ja Gupta (2022) ennustaakin, että pilvipalveluiden liikevaihto kasvaa vuoden 2018 268 miljardista dollarista vuoteen 2024 651 miljardiin dollariin. Vuonna 2024 OnPremise ratkaisujen Gupta (2022) arvioi olevan vain hieman pilvipalveluiden ratkaisuja korkeampi 786 miljardia dollaria. SaaS:n suosio selittyykin pilvipalveluiden jatkuvalla kasvulla. Siinä on mahdollista skaalautua asiakkaiden tarpeiden mukaan ja hyödyntää pienpalveluarkkitehtuuria tai mikropalveluita, joiden avulla suuret sovellukset saadaan toimimaan monien pienten palveluiden päällä esimerkiksi skaalautuvassa pilvipalveluratkaisussa. SaaS:ssa voidaan hyödyntää myös muita mikropalveluita sovelluksen käytössä, kuten pilvitallennustilaa, verkkoarkkitehtuurisia ratkaisuja ja kolmannen osapuolen muita pilvipalveluratkaisuja. Kun kaikki sovelluksen käyttöön liittyvät komponentit ostetaan pilvipalveluina, niiden skaalaaminen sopivaksi on helppoa, koska pilvipalvelut on luotu skaalautuviksi malleiksi, joista voi ostaa juuri sen verran suorituskykyä tai kapasiteettia kuin on tarpeen. Pilvipalveluissa myös maksut ovat joustavia, koska ei tarvitse maksaa itse ylläpidetystä palvelinsalista vaan voidaan maksaa vain käytön mukaan. Palvelimien ja muun teknisen infrastruktuurin ylläpitokulut ja investoinnit ovat myös tarpeettomia, koska pilvipalvelun ja SaaS ratkaisun toimittaja huolehtii kaikista näistä kuluista. (Gupta, 2022; Familiar, 2015 luku 1; Alanda ym., 2022, s. 50.)

4.3 Monoliittisten ja mikropalveluiden toimitusten erot

Monoliittinen järjestelmä on mikropalveluiden vastakohta. Mikropalveluissa tai mikropalveluista koostuvassa järjestelmässä kaikki mikropalvelut ovat omia kokonaisuuksiaan, jotka eivät välttämättä tarvitse toisiaan toimiakseen. Ne tekevät itsenäisesti aina yhden pienen tehtävän. Kokonainen järjestelmä kuitenkin vaatii, että kaikki mikropalvelut toimivat yhdessä, jotta kokonaisuudessaan järjestelmä toimii. Tämän vuoksi mikropalveluista koostuvan järjestelmän kehittäminen on helpompaa, koska yksittäisiä mikropalveluita voidaan kehittää ja testata erikseen sekä yhdessä kokonaisen järjestelmän kanssa. Monoliittisessa järjestelmässä koko koodimassa, josta järjestelmä koostuu, toimii yhtenäisenä osana järjestelmää. Sitä kehitettäessä on huomioitava koko koodimassan testaus aina, kun muutoksia tehdään eli siinä ei voida ajatella kehitettävän vain yhtä osa-aluetta. (Familiar, 2015, luku 2.)

Monoliittisessa järjestelmässä arkkitehtuurin suunnittelu ja kehitys tapahtuvat kyseisen järjestelmän rajojen sisällä ja kaikki uudet asiat, joita sovellus tulevaisuudessa tarvitsee pitää ohjelmoida sovellukseen. Mikropalveluissa arkkitehtuuriratkaisun ei tarvitse olla järjestelmän rajoissa vaan voidaan ottaa erilaisia ominaisuuksia eri SaaS ratkaisujen tarjoajilta. Näin kokonainen järjestelmä voi koostua esimerkiksi kolmen eri toimittajan erilaisista mikropalveluista, jotka kytketään yhteen ja niistä syntyvät koko sovelluksen toiminnallisuudet. Mikropalveluissa on tärkeää huomioida niiden sijainti ja kuinka niihin otetaan yhteys eli integraatiot. Monoliittisessa järjestelmässä kokonaisuus on rakennettu samaan paikaan ja järjestelmän eri osa-alueiden väliseen tiedonsiirtoon ei tarvitse tehdä erillisiä ratkaisuja. (Bruce & Pereira, 2018, luku 3.1.)

Kohdeorganisaatiossa sovellus on modulaarinen järjestelmä, jossa kokonainen järjestelmä on hajautettu erilaisiin moduuleihin, kuten asiakas, laskutus ja tiedonvaihto. Nämä erikseen toimivat moduulit ovat kuin mikropalveluita, jotka kaikki hoitavat oman osa-alueensa, mutta yhdessä ne koostavat kokonaisen järjestelmän. Ongelmalliseksi järjestelmässä on todettu arkkitehtuuri eli toisin, kuin mikropalveluiden osalta niin järjestelmää pitää käsitellä, kuin monoliittista järjestelmää esimerkiksi kehityksen puolella, koska moduulit eivät ole täysin

irrallisia toisistaan. Tämän lisäksi järjestelmä hyödyntää vielä erikseen mikropalveluita SaaS ratkaisuna.

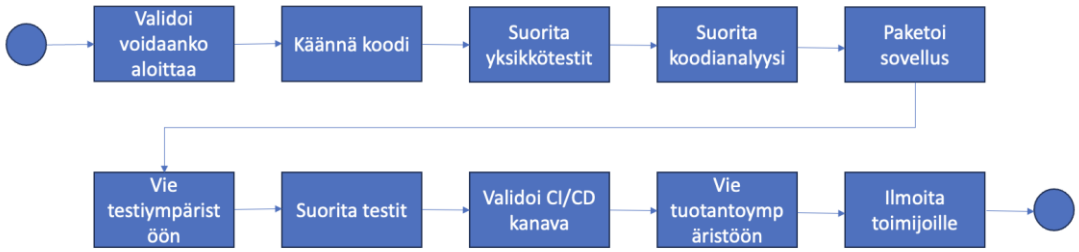
4.4 CI/CD prosessi

CI/CD-prosessin tarkoitus on saada ohjelmistokoodi tuotantoon nopeammin automatisoimalla sovelluskehityksen toimitusketju. Jatkuvan integroinnin (CI) periaate on, että sovelluksen ohjelmointikoodi on tallennettuna versionhallintajärjestelmään ja aina, kun koodi muuttuu, käynnistyy automaattinen kääntöprosessi (Build), joka luo koodipaketin keskitettyyn koodikirjastoon. Kääntöprosessi muuttaa ihmisen luettavan koodin, kuten C# koneluettavaksi prosessiksi tietokoneelle. Ilman koodin kääntämistä palvelin ei pysty ajamaan koodia ja koodi ei toimi. Osa ohjelmistokoodeista on sellaisia, että erillistä kääntöprosessia ei tarvita (Limón, 2022). Jatkuva toimitus (CD) tarkoittaa, että koodikirjastossa on jatkuvasti vakaa pääkoodi, josta haaroitetaan julkaistavat versiot. Sitä ylläpidetään automaattitesteillä, joiden tarkoitus on tutkia toimivatko ohjelmiston toiminnalliset eli järjestelmällä tehtävät toiminnot ja ei-toiminnalliset kuten suorituskykyominaisuudet uusien koodien lisäyksen jälkeen. (Merode, 2023, luku 2.)

CI/CD-prosessin toimintoja kutsutaan julkaisukanavaksi (Pipeline) eli automatisoidut vaiheistetut askeleet etenevät tietyn prosessin mukaan. Prosessin avulla saadaan kehittäjille tieto, kuinka julkaisu on onnistunut ja ovatko kaikki automaattiset testit onnistuneet. Mikäli koko julkaisukanava menee ongelmitta läpi, niin silloin muutoksen ja julkaisun voi viedä tuotantoon asti. Julkaisukanava sisältää kaikki vaiheet, jotka CI/CD prosessin on käytävä läpi ohjelmistokoodin julkaisemiseksi tuotantoon. Se tarkistaa aloitetaanko julkaisukanavan prosessi, minkä jälkeen se käy yksitellen läpi kaikki vaiheet, kuten sovelluksen rakentamisen, yksikkötestit, koodianalyysin, koodin paketoinnin, paketin julkaisun testiympäristöön, testauksen ja tarkastukset julkaisukanavan eri vaiheista. Nämä vaiheet ovat kuvattu kuvion 11 prosessissa. Tämän jälkeen uuden ohjelmistokoodin voi päivittää tuotantoon. Tuotantoon viennin jälkeen prosessi

ilmoittaa toimijoille, että uusi versio on viety tuotantoon. (Merode, 2023, luku 4.)

Kohdeorganisaatiossa ei kuitenkaan ole suoraan käytössä jatkuva toimitusprosessi, eli siellä pidetään tuotantoon viennit manuaalisina versiohankinnan toimenpiteinä. Työssä tarkoitus on kuitenkin vertailla voiko yrityksen nykyistä prosessia laajentaa ja tehostaa niin, että CD prosessi voitaisiin ottaa käyttöön.



Kuvio 11. Yksinkertainen CI/CD julkaisukanavaprosessi (mukaillen Merode, 2023, kuvio 4-6)

Täydellinen jatkuva integraatio -prosessi vaatii, että kehitys- ja versionhallintamalli olisi päähaarassa kehittäminen, jonka perusteella päähaaraan tehtäisiin vain pieniä muutoksia ja jokaisen päivän päätteeksi kehittäjät liittäisivät tehdyt toimenpiteet päähaaraan. Ominaisuusvetoista kehitysmallia voidaan kuitenkin käyttää, mutta se vaatii, että kehitettävät ominaisuudet ovat pieniä ja toiminta saman tyyppistä, kuin päähaarassa kehittäminen. Suuret ominaisuudet, joissa liittäminen päähaaraan voi kestää viikkoja eivät sovellu jatkuva integraatio -prosessiin. (Dora, n.d.; Merode, 2023, luku 4.)

Tutkimuksen mukaan kehitys CI/CD prosessin avulla nopeuttaa toimitusta ja sen avulla saadaan versiojulkaisut nopeammalla tahdilla tuotantoon, mikä onkin pääsyy ottaa CI/CD prosessi käyttöön. Se myös toimitusnopeuden ja -helppouden parantamisen sekä automatisoinnin vuoksi vapauttaa resursseja tärkeämpiin töihin. Klotins ym. (2022, s. 32–33) mainitsee, että kokonaisen CI/CD prosessin käyttöönotto on hankalaa, koska se vaikuttaa myös asiakkaiden versiohallintaan eli siihen, milloin uusi versio asiakkaalle tulisi käyttöön. Tämä voi aiheuttaa ongelmia, mikäli asiakkaalla on asiakkaita, jotka käyttävät

sovellusta, koska sovelluksen päivitys ei välttämättä onnistu jatkuvasti, kun halutaan vähentää riskejä tai tietty versiosykli on sovittu ja sopimuksen muuttaminen voi olla hankalaa. (Alanda ym., 2022, s. 54; Klotins ym., 2022, s. 32–36; Alnafessah ym., 2021, s. 44480–44481.)

4.5 Versiotiedotteet

Versiotiedote tehdään sovellusversiosta, joka on toimitusvalmis asiakkaiden ympäristöön. Se pitää sisällään versiossa tulevat muutokset ja uudet ominaisuudet. Versiotiedotteet kirjoitetaan suoraan käyttäjille ja ne pitää sisällään muutokset, miksi muutokset tehtiin, miten asia oli ennen ja miten se on nyt sekä rajoitteet. (Bhatti, 2021, luku 2; Vasserman, 2023.)

Versiotiedotteet auttavat niin kehitystiimejä selvittämään mitä muutoksia missäkin versiossa on tullut, kuin asiakkaita löytämään uudet muutokset ja käyttömahdollisuudet sovelluksesta. Niiden avulla voi kertoa käyttäjille, että sovellus on jatkuvassa kehityksessä ja uutta tehdään jatkuvasti. Tämä auttaa käyttäjiä tietämään, että sovellusta pidetään ajan tasalla. (Vasserman, 2023.)

Versiotiedotteet pitää olla selkeitä ja esitetyt asiat pitää kirjoittaa lyhyesti ja tarkasti. Niissä pitää huomioida niin tekniset, kuin ei tekniset käyttäjät myös visuaalisuudesta on apua lukijoille, kun he yrittävät hahmottaa sisältöä. Linkkien lisääminen syvällisempiin ohjeisiin kannattaa tehdä, mikäli ominaisuus on suurempi kokonaisuus, joka pitää kuvata tarkemmin. Versiotiedotteita pitää tehdä jatkuvasti, kun asiat muuttuvat eli jokaisesta päivityksestä ja versiomuutoksesta tulee tehdä versiotiedote. Versiotiedotteissa pitää huomioida myös poistuvat ominaisuudet, eli mikäli jokin ominaisuus on poistumassa tai se korvataan toisella ominaisuudella niin näistä pitää ilmoittaa käyttäjille. (Vasserman, 2023.)

5 ASIAKASTYYTYVÄISYYS

Tässä luvussa käsitellään asiakastyytyväisyys ja -kokemus, sekä näiden erot ja kuinka ne nivoutuvat yhteen. Tämän jälkeen käsitellään asiakkuudenhallinta ja mitä se tarkoittaa organisaatiolle. Lukuun on myös otettu mukaan toimittajan kehittäminen eli asiakkaan näkökulmasta asiakkaalla ei ole passiivinen rooli asiakassuhteessa vaan myös asiakas pyrkii kehittämään toimittajaa yhteistyössä sen kanssa.

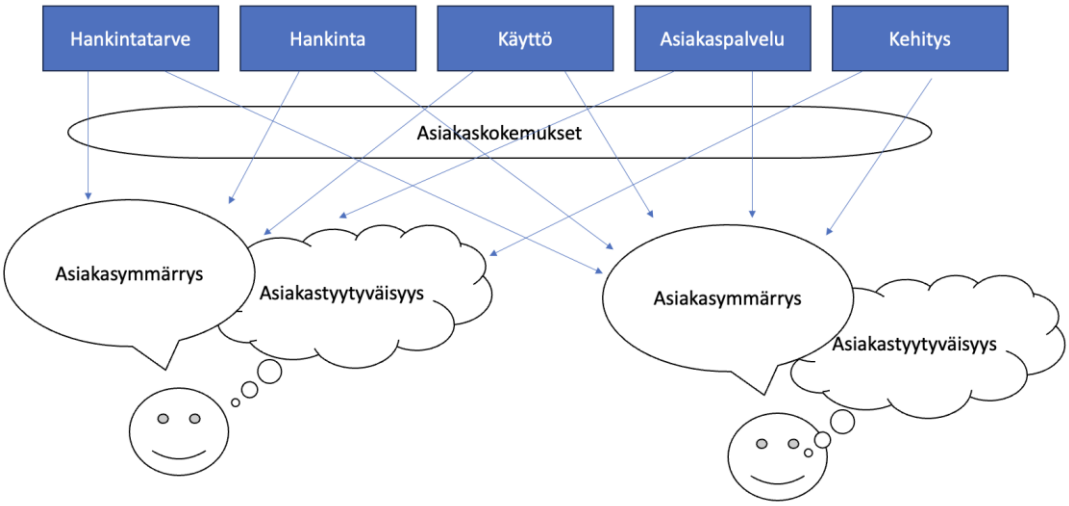
5.1 Asiakastyytyväisyys ja -kokemus

Mitattaessa asiakastyytyväisyyttä on tärkeää tunnistaa koko markkinoiden toimintamalli. Miten markkinat toimivat, paljonko asiakkaat keskustelevat keskenään ja minkälaisista foorumeista asiakkaat löytävät tietoa tuotteesta? Näiden lisäksi pitää huomioida, kuinka asiakkaan koko matka versiotoimituksen tilauksesta julkaisuun ja sen jälkeisiin päivityksiin toimii, jotta saadaan asiakkaalta tieto, miksi he päätyivät tai eivät päätyneet hankkimaan versioita. (Keskinen & Lipiäinen, 2013, s. 29–32.) Pelkästään versiotoimitusta tutkimalla ei välttämättä saada tarpeeksi tietoa, jotta voidaan löytää syy hankinnalle. Tämän vuoksi pitää laajentaa tutkimuksessa kyselyaluetta liittymään koko versiotoimituksen elinkaareen ja sen jälkeiseen aikaan.

Asiakastyytyväisyys on ero asiakkaan odotusten ja niiden toteutumisen välillä, kun taas asiakaskokemus liittyy asiakkaan subjektiiviseen kokemukseen eri yrityksen kontaktipisteissä. Asiakastyytyväisyyden keskittyessä siihen, mitä asiakas odottaa ja mitä asiakas saa, asiakaskokemus perustuu enemmän ajan kautta syntyviin kokemuksiin ennen asiakkuutta, asiakkuudessa ja sen jälkeen. Se on myös enemmän henkilön itsensä kokemaa kokonaisuutta, johon vaikuttavat myös sosiaaliset ja emotionaaliset kokemukset. Asiakaskokemusta johdettaessa pitää huomioida, että se on asiakkaan subjektiivinen kokemus, johon on vaikea vaikuttaa, koska niitä on yhtä paljon, kuin asiakkaitakin. Siinä voidaan kuitenkin vaikuttaa esimerkiksi asiakkaan kosketuspisteisiin ja asiakaspolkuihin. Voidaan luoda erilaisia kontaktipisteitä erilaisille

asiakkaille. Jotkut pitävät soittamisesta, jotkut haluavat lähettää vain tiketin järjestelmään ja osa voi haluta edetä sähköpostilla tai chat-keskustelulla. Näiden avulla voidaan vastata asiakkaiden subjektiivisiin odotuksiin paremmin ja saada asiakastyytyväisyyttä kasvatettua. (Saarijärvi & Puustinen, 2020, s. 36–38.)

Asiakastyytyväisyyden tärkeä lähtökohta on asiakkaan ymmärtäminen. Miten asiakas saapuu tilanteeseen, jossa hän on ostamassa tuotteen? Minkälaiset ajatukset ja toimet johtavat uuteen asiakkuuteen ja minkälaiset toimet ovat aiheuttaneet asiakkuuden menetyksen. Asiakasymmärrystä voidaan hankkia kuuntelemalla asiakkaita, tutustumalla uusimpaan asiakastutkimukseen ja haastattelemalla tai kyselyillä. Asiakastyytyväisyydestä ja asiakkaan ymmärryksestä päästään kokonaisvaltaiseen asiakkuudenhallintaan, joka jatkuu asiakashankinnasta ostoon ja asiakassuhteen mahdolliseen luopumiseen asti. Kuviossa 12 kuvataan, miten asiakaskokemus, asiakastyytyväisyys ja asiakasymmärrys liittyvät toisiinsa. (Bergström & Leppänen, 2021, s. 356–358.)



Kuvio 12. Asiakastyytyväisyys, -ymmärrys ja -kokemus yhdessä (mukaillen Saarijärvi & Puustinen, 2020, s. 36–38; Bergström & Leppänen, 2021, s. 356–358)

Tässä työssä haastatteluilla, dokumenttianalyysillä ja havainnoimalla pyritään ymmärtämään asiakasta paremmin, sekä löytämään ne kohdat, joissa

asiakkaan odotukset ovat täyttyneet tai he ovat joutuneet pettymään. Näiden kohtien löytäminen ja muuttaminen parantaa asiakastyytyväisyyttä ja palvelua asiakkaiden suuntaan. Mikäli organisaatio toimittaa loistavaa palvelua kaikissa tilanteissa, se edesauttaa asiakassuhteiden jatkuvuutta ja pidemmät asiakassuhteet lisäävät luottamusta organisaation ja asiakkaiden välille (Bergström & Leppänen, 2021, s. 354–365).

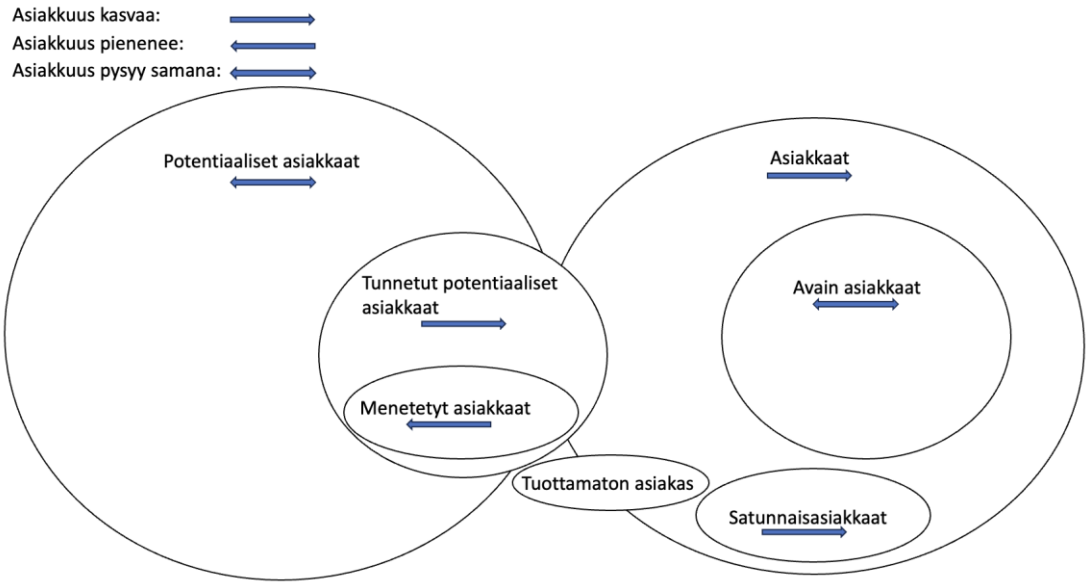
5.2 Asiakkuudenhallinta

Asiakkuuksien hallinnassa on monta eri strategiaa, voidaan keskittyä yksittäisen asiakkaan ja hänen kokemustensa hallintaan tai voidaan segmentoida asiakkaat ja keskittyä yhden segmentin hallintaan. Voidaan puhua myös asiakaskokemuksen johtamisesta, jolla pyritään luomaan asiakkaille merkittävä suhde organisaatioon, jotta asiakkuus jatkuisi. Käytännössä asiakkuudenhallinta tarkoittaa, että asiakkuudet sisällytetään yrityksen strategiaan ja pyritään luomaan tilanne, jossa organisaatio voi tuottaa kannattavasti arvoa asiakkaalle. Asiakassuhteiden vaaliminen on tärkeää jatkuvuuden kannalta. Pidemmissä asiakassuhteissa asiakkaat alkavat enemmän huomioida palvelun laatua kuin kustannuksia, kun taas lyhyemmissä asiakassuhteissa hinta on yleensä määräävä tekijä. (Bergström & Leppänen, 2021, s. 354–365.)

Asiakkuudenhallinnassa asiakkaat ja asiakassuhteet voidaan ryhmitellä perinteisesti potentiaalisiin asiakkaisiin, satunnaisasiakkaisiin, avainasiakkaisiin ja entisiin asiakkaisiin. Potentiaaliset asiakkaat ovat niitä, jotka mahdollisesti voisivat ottaa tuotteen tai palvelun käyttöön. Ne voivat olla tunnettuja asiakkaita, joiden kanssa myyntikeskustelua käydään tai ne voivat olla tuntemattomia asiakkaita, jotka kuuluvat organisaation kohderyhmään. Satunnaisasiakkaat ovat kerta-asiakkaita eli ostavat tuotteita satunnaisesti. Avainasiakkaat ovat yrityksen tärkeimmät asiakkaat eli riippuen segmentointitavasta avainasiakkaat yleensä tuovat suurimman liikevaihdon organisaatiolle tai ovat muuten sille kaikkein kannattavimpia. Entiset asiakkaat ovat asiakasryhmä, joka on joskus hankkinut tuotteita tai palveluita organisaatiolta, mutta ei hanki enää

esimerkiksi liian korkean hinnan tai palveluun pettymyksen takia. (Bergström & Leppänen, 2021, s. 363–366.)

Tunnetut potentiaaliset asiakkaat pyritään saamaan asiakkaiksi myyntiprosessin avulla. Entiset asiakkaat ovat tunnettuja asiakkaita, mutta heidän kanssaan asiakkuus vähenee. Tietyn ajan kuluttua on mahdollista, että asiakkuus alkaa kasvaa myös näiden asiakkaiden kanssa. Olemassa olevien asiakkaiden asiakassuhdetta pyritään myös kasvattamaan, jotta saadaan tehtyä esimerkiksi lisämyyntiä ja muutettua asiakkuus avainasiakkuudeksi. Satunnaisasiakkaat pyritään muuttamaan jatkuvaksi asiakassuhteeksi. Kuviossa 13 kuvataan asiakkuuden suunta eri asiakassegmenteissä.



Kuvio 13. Asiakkuusryhmittely ja asiakkuuden suunta (mukaillen Bergström & Leppänen, 2021, s. 366–367)

Organisaation on myös tärkeää huomioida poistettavat asiakkuudet omassa strategiassaan. On mahdollista, että asiakkuudesta syntyy enemmän kuluja kuin hyötyä organisaatiolle. Tällöin on tärkeää hoitaa asiakkuutta oikein, joko muuttamalla tappiollinen asiakkuus hyödylliseksi esimerkiksi kuluja pienentämällä tai lisämyynnillä. Toinen vaihtoehto on lopettaa asiakkuus ja silloinkin on huomioitava se, että vaikka asiakkuus tällä hetkellä olisi tappiollinen organisaatiolle niin tulevaisuudessa asiakkuus voi olla sellainen, jonka organisaatio

haluaa takaisin. Näissä tapauksissa asiakkuudesta luopuminen pitää tehdä niin, että asiakkaalle jää asiakkuudesta positiivinen kokemus. Tämä edesauttaa sitä, että asiakas ei anna huonoa mainosta organisaatiosta asiakkuuden loppumisen vuoksi muille potentiaalisille tai nykyisille asiakkaille. (Bergström & Leppänen, 2021, s. 368.)

5.3 Toimittajan kehittäminen

Toimittajan kehittämisellä asiakkaat pyrkivät kasvattamaan toimittajan kyvykkyyksiä. Yleensä tätä tehdään tiiviissä yhteistyössä ostavan osapuolen kanssa, jolloin toimittaja sitoutuu kehittämään asiakkaan tarvitsemaa kokonaisuutta ja samalla toimittajan kilpailukyky markkinoilla kasvaa. Tärkeää on löytää arvon tuotto molemmille osapuolille, koska yhteistyö vaatii resursseja molemmilta osapuolilta. Kommunikaatio asiakkaan ja toimittajan välissä pitää olla avointa ja välitöntä, jotta yhteistyö ja toimittajan toiminta voivat kehittyä. (Nieminen, 2016, luku 5.3.)

Nykyään SaaS palveluiden maailmassa pitkien suhteiden solmiminen ei ole enää niin pakottavaa kuin ennen, koska palveluntarjoajan vaihtaminen on helpompaa. SaaS palvelussa tärkeää on luottamus ja yhteistyön syventäminen, koska asiakkaan on vain sovelluksen käyttäjänä luotettava toimittajan palvelun toimimiseen. Toimittajasuhteet ovat kaksiulotteisia eli hankitun sovelluksen sekä toimittajan palvelun tulee olla laadukkaita. Mikäli nykyistä asiakkuutta ei huomioida tai ylläpidetä, asiakas näkee tämän toimittajan palvelun laskuna, vaikka itse sovelluksen tai palvelun laatu pysyykin samana tai kasvaa. Toimittajan tulee toimittaa laadukasta palvelua, tietoa ja järjestelmäpalvelua. (Sombultawee & Pasunon, 2022, s. 629–631.)

6 LÄHESTYMISTAPA JA TUTKIMUSMENETELMÄT

Tässä luvussa käydään läpi, minkälaisia menetelmiä tutkimuksessa on ja kuinka niitä hyödynnetään. Ensin esitetään lähestymistapa ja miksi se on valittu, minkä jälkeen käytettävät tiedonkeruumenetelmät esitellään.

6.1 Tapaustutkimus lähestymistapana

Työ tehdään tutkimuksellisena kehittämistehtävänä ja siinä lähestymistavaksi on valittu tapaustutkimus. Tutkimuksellisen kehittämistehtävän perustarkoituksena on tuottaa uutta tietoa tutkittavasta asiasta ja kehittää sille käytännön parannuksia tai kehitysehdotuksia, kuinka asiaa voitaisiin parantaa. Erona tieteellisen tutkimukseen on, että sen tavoitteena on kehittää vain uutta tietoa teoriaan, kun taas tutkimuksellisen kehittämisen tehtävässä luodaan uutta teoriaa, mutta myös käytännön ratkaisuja tutkittavaan tapaukseen tai ongelmaan. (Ojasalo ym., 2015, s. 19.)

Lähestymistapa on laajempi osa-alue, kuinka lähestytään tiettyä tutkittavaa kohdetta. Lähestymistapa myös ohjaa osittain sitä, mitkä menetelmät sopivat parhaiten mihinkin tutkittavaan asiaan. Kaikkia tiedonkeruumenetelmiä voi kuitenkin käyttää kaikissa lähestymistavoissa, mutta lähestymistapa luo ylemmän tason strategian, kuinka ongelmaa lähdetään tutkimaan ja mikä tulee olemaan lopputulos. Tässä tutkimuksellisessa kehittämistehtävässä valittiin tapaustutkimus, koska se soveltuu siihen, että hankitaan syvällistä tietoa tietystä tapauksesta, joka tässä on toimitusprosessi, sekä tuotetaan siihen kehittämisehdotuksia. (Ojasalo ym., 2015, s. 51–54.)

Opinnäytetyössä tutkitaan kohdeorganisaation versiotoimitusprosessia ja asiakkaiden näkökulmaa sekä mielipiteitä toimituksista. Prosessi on selkeä tapaus, johon kytkeytyvät asiakastyytyväisyys, sovelluskehitys ja prosessimallit. Näiden perusteella lähestymistavaksi on valittu tapaustutkimus. Tapaustutkimus on Erikssonin & Koistisen (2014, s. 4) mukaan tapauksen tarkastelua, määrittelyä ja analysointia sekä tapauksen ratkaisu on tapaustutkimuksen

keskeisin tavoite. Franco ym. (2023, s.123) kuvaavat sen olevan perusteellinen kuvaus ja tutkimus tietystä tapauksesta, henkilöstä, ryhmästä tai paikasta. Tapaustutkimus voidaan jaotella vielä kuvailevaan, selittävään ja exploratiiviseen tapaustutkimukseen. Näistä kolmesta opinnäytetyön tapaustutkimus on exploratiivista sekä selittävää tapaustutkimusta. Tarkoituksena on ensin selittää tapaus eli vastata kysymykseen, miksi tapaus on sellainen kuin se on. Uusia teoreettisia yleistyksiä kyseisestä tapauksesta pyritään löytämään exploratiiviselta näkökulmalta ja näiden perusteella, sekä vertaamalla olemassa olevaan kirjallisuuteen vastataan tutkimuskysymyksiin. (Eriksson & Koistinen, 2014, kappale 2.)

6.2 Tiedonkeruumenetelmät

Tiedonkeruumenetelmät jaetaan yleensä kahteen osa-alueeseen, joista toinen on kvalitatiiviset eli laadulliset menetelmät ja toinen on kvantitatiiviset eli määrälliset menetelmät. Määrälliset menetelmät vaativat yleensä suuren tutkimusjoukon ja tuloksia tarkastellaan tilastollisen menetelmän keinoin. Laadullisissa menetelmissä tutkimusjoukko on paljon pienempi kuin määrällisissä menetelmissä, mutta tutkittavaa aineistoa voi kertyä paljonkin, kun siihen pitää paneutua ja tutkittavaa kohdetta pyritään ymmärtämään syvällisesti. (Ojasalo ym., 2015, s. 104–105.)

Työssä hyödynnetään laadullisia tiedonkeruumenetelmiä, koska pyritään tutkimaan tiettyä tapausta ja ymmärtämään syvällisesti ongelmaa. Tutkittava joukko on kokonaisuudessaankin niin pieni, että määrälliset menetelmät on rajattava pois. Laadullisista menetelmistä on valittu kolme erilaista menetelmää, jotta saadaan tarvittava tieto kerättyä kaikista mahdollisista paikoista, joihin tieto on tallennettu. Tieto on hajautettu dokumentaatioihin, joissa on esimerkiksi tietoa toimituksista ja toimitusprosesseista sekä syntyneistä haasteista ja niiden ratkaisuista. Haastattelulla pyritään keräämään tieto asiakaskokemuksesta. Kuinka asiakas kokee toimituksen ja onko asiakas tarpeeksi huomioitu niin hankinnan, toimituksen kuin toimituksen jälkeisessä vaiheessa? Organisaation henkilöstöä haastattelemalla saadaan käsitys prosessissa töitä

tekevistä tahoista kuten myynti, toimitus ja ohjelmointi. Havainnoinnilla saadaan oman kokemuksen kautta kerättyä tietoa prosessista. Tutkija toimii organisaatiossa toimitusprosessien aikana tuotepäällikön roolissa liitännäisjärjestelmissä, kuten integraatiot, joka tarkoittaa tiedon välitystä kahden tai useamman järjestelmän välissä ja synteettisen datan tuotteet.

6.2.1 Dokumenttianalyysi

Dokumenttianalyysi tarkoittaa kirjallisen dokumentaation läpikäyntiä ja sen järjestämistä tutkittavaan muotoon, sekä analysointia. Johtopäätökset tehdään järjestetystä aineistosta. Dokumentteihin sisältyvät kaikki dokumentit, joita voivat olla esimerkiksi tekstit, muistiot, www-sivut, haastattelut, kuvat, kuvattu materiaali, prosessikaaviot. (Ojasalo ym., 2015, s. 136; Rubin, 2021, s. 16.)

Työssä tutustutaan organisaation nykyisiin kehitys- ja toimitusprosesseihin. Tutustutaan yleisesti organisaation versioiden sisältöihin ja pitävätkö ne sisällään yleisesti toivottuja ominaisuuksia ja ovatko ominaisuudet esimerkiksi tuotekehityksen standardin mukaisia (Digi- ja väestötietovirasto, 2020). Organisaatio on kerännyt toimituksista dokumentaatiota ja esimerkiksi asiakaspalavereista on tehty muistioita, joita hyödynnetään tässä työssä. Tavoitteena on myös löytää tarjous- ja tilausprosessin puolelta materiaalia, jonka avulla voidaan tutkia, miten tilaus- ja toimitusprosessi etenee asiakkaan ja organisaation näkökulmasta. Tärkeää on huomata myös, onko jokaista uutta versiota tarjottu asiakkaille samalla tavalla vai onko tarjouksissa, ja tarjousprosesseissa ollut eroja hankittujen ja hankkimatta jääneiden versioiden välillä.

6.2.2 Haastattelut

Pöyhösen ym. (2023, s. 177) mukaan haastatteluilla pyritään saamaan syvällisempi näkemys haastateltavalta, kuin strukturoidulla kyselyllä. Sen avulla voidaan useiden kosketuspisteiden kautta ymmärtää haastateltavaa. Rubin mukaan (2021, s. 15–16) haastattelu voidaan tehdä strukturoidusti lomakehaastatteluna tai puolistrukturoidusti teemahaastatteluna, jossa haastattelu ja

kysymykset liittyvät tiettyyn pääteemaan, mutta niitä ei ole määritelty ennalta. Lomakehaastattelussa taas haastattelijalla on lomakepohja, jonka perusteella käydään kaikki kysymykset järjestyksessä läpi. Avoimessa haastattelussa pyritään ymmärtämään haastateltavan ajatusmallia ja motiiveja. Siinä haastateltavan havainnointi on myös erittäin tärkeä osa haastattelua suorien vastausten lisäksi. (Pöyhönen ym., 2023, s. 182, 221.)

Haastattelut toimivat tiedonkeruumenetelmänä hyvin, kun tarkoitus on kerätä tietoa mielipiteistä, havainnoista, arvoista tai kokemuksista. Niissä on tärkeä olla johdattelematta haastateltavaa, osata kysyä oikeita kysymyksiä oikeassa kohdassa ja kuunnella. Kun haastatteluun yhdistetään muita tiedonkeruumenetelmiä, voidaan saada lisää tietoa haastateltavasta. Ihmisten on vaikea analysoida itseään ja kertoa omista haluistaan, toiveistaan tai motivaatiostaan. Tämän vuoksi haastattelijan taito haastatella on kriittisessä asemassa haastattelua tehdessä. Haastattelijan on jo ennen haastattelua osattava luoda pohja haastattelulle ja valita oikeat haastateltavat. Haastattelun jälkeen pitää osata analysoida niiden tulokset systemaattisesti ja objektiivisesti. (Pöyhönen ym., 2023, s. 182–184, 223.)

Haastatteluissa on kolme porrasta; asiakasorganisaation haastattelut, sisäisesti keskijohdon haastattelut, joihin sisältyvät projektipäälliköt, asiakkuuspäälliköt ja tuotehallinto sekä sisäisesti asiantuntijat esimerkiksi ohjelmoijat, testaajat ja arkkitehdit. Näin saadaan kokonaiskuva toimitusprosessista ja siitä, mitä ongelmakohtia yleisesti versiotoimituksissa on ja kuinka asiakas kokee versiotoimitusprosessin sekä sovelluksen käyttöönoton. Hyvärisen ym. (2017, s. 29) mukaan laadullisessa tutkimuksessa tulee miettiä, kuinka monta haastattelua tarvitsee tehdä, jotta saadaan tarpeeksi tietoa tutkittavasta tilanteesta. Haastatteluille ei ole oikeaa lukumäärää vaan haastatteluja pitäisi tehdä niin paljon, ettei uutta sisältöä enää synny ja ne tulisi purkaa sitä mukaan, kun haastatteluja tehdään, jotta tiedetään, milloin haastatteluja on tehty tarpeeksi. Opinnäytetyössä haastattelut puretaan saman tien, kun ne tulevat, koska suurin osa haastatteluista on kirjallisia ja lisäkysymyksiä voi syntyä, mikäli haastateltavat ottavat uusia teemoja esiin. Haastattelukysymykset ja teemat ovat mainittu liitteessä 1.

Asiakashaastatteluilla pyritään löytämään asiakasnäkökulmaa toimitus- ja käyttöönotto prosesseihin ja syytä siihen, miksi kaikkia versioita ei haluta hankkia vaan hypätään joidenkin versioiden ylitse. Sisäisillä haastatteluilla tutkitaan kehitys- ja toimitusprosessia, toimitusprosessin tehokkuutta ja asiakkaan odotuksiin vastaamista niin asiantuntijatasolla, kuin organisaation keskijohdon tasolla. Haastattelukysymykset jäsentyvät tutkimuskysymysten ympärille, eli tutkimuskysymykset määrittävät, mitä haastattelussa kysytään. Aineistoon kohdistuvat kysymykset pitää huomioida haastattelua suunniteltaessa, pitää tietää mitä aineistolta halutaan kysyä, kun se on valmis. Haastateltavat henkilöt eivät tarjoa valmiita tutkimustuloksia, vaan tutkimustulokset syntyvät vasta silloin, kun tutkija analysoi haastattelun tulokset. (Hyvärinen ym., 2017, s. 18–19.)

Haastattelut tehdään kirjallisina esimerkiksi sähköpostin tai Teamsin välityksellä, mikäli tulee tarve keskustelupohjaiselle haastattelulle. Ne ovat siis strukturoituja kyselyhaastatteluja, joissa on teemahaastattelujen piirteitä eli asiakas saa kaikki kysymykset heti näkyville, mutta vastauksiin odotetaan laajempaa näkökulmaa, kuin pelkästään kyselyhaastattelussa (Hyvärinen ym., 2017, s. 68–70). Haastattelut pyritään pitämään yksilöhaastatteluina, ellei synny tarve pitää haastattelua pienryhmälle. Yksilöhaastattelut on valittu siitä syystä, että haastattelut tehdään kirjallisesti valituille organisaation henkilöille. Asiakasorganisaatioiden tapauksessa haastattelupohjasta tulee selvitä, mitkä organisaation roolit ovat haastatteluun vastanneet. Haastattelujen vastauksia ei lisätä liitteeksi opinnäytetyöhön, vaan vastauksia hyödynnetään ainoastaan tutkimuksellisessa kehittämistehtävässä ja ne raportoidaan yleisellä tasolla ja suorat lainaukset tehdään kertomatta haastateltavan nimeä tai organisaatiota.

6.2.3 Havainnointi

Havainnointi voi olla pääasiallinen tai se voi tukea muita tiedonkeruumenetelmiä. Havainnointia on kolmea erilaista, jotka ovat riippuvaisia tutkijan roolista havainnoinnin suorittajana. Tutkijalla voi olla joko osallinen (työskentelee samassa organisaatiossa, jossa tutkimus tehdään), osallistuva (on ulkopuolinen

toimija, joka kuitenkin esimerkiksi avustaa projektissa) tai täysin ulkopuolinen rooli havainnoinnissa. Kahdessa ensimmäisessä tutkija osallistuu tutkittavan kohteen toimintaan ja kolmas rooli on sellainen, jossa tutkija erottaa itsensä koko tutkittavasta kohteesta ja toimii vain ulkopuolisena havainnoitsijana. Ensisijaisen tärkeää havainnoinnissa on tehdä havainnointi järjestelmällisesti, jonkin suunnitelman mukaan. Pitää ensin tietää, mitä havainnoidaan ennen kuin voidaan havainnoida sitä järjestelmällisesti. (Puusa & Juuti, 2020, s. 128–129.)

Havainnointia hyödynnetään tiedonkeruumenetelmänä, koska tutkija on töissä kohdeorganisaatiossa ja työskentelee läheisesti tiimien kanssa, jotka tekevät versiotoimituksia. Rubin (2021. s. 15) mukaan havainnointi pitää yleensä sisällään haastatteluja, kuten epävirallisia haastatteluja keskustelujen muodossa. Näin on mahdollista kerätä myös havainnointitietoa versiotoimituksista. Havainnoinnin yksi tarkoitus tässä opinnäytetyössä on löytää tietoa, jota ei välttämättä ole dokumentoitu tai kaikkea sitä ei saada irti haastatteluilla. Tällöin havainnoinnin merkitys kasvaa, kun pyritään löytämään sellaisia asioita, joita ei ole osattu dokumentoida tai joiden dokumentointi ei ole ollut yritykselle merkityksellistä, mutta tutkimuksellisen kehittämistehtävän vuoksi on.

Havainnointia tehdään ideariihimäisesti ja tietoa saadaan keskusteluissa asiakkaan ja organisaation kanssa. Havainnoimalla myös tutkitaan, kuinka toimitusprosessi on edennyt verrattuna siihen, kuinka se on suunniteltu tai dokumentoitu. Ovatko kaikki toimitusprosessin kirjalliset vaiheet huomioitu toimituksessa vai hypätäänkö joidenkin kohtien ylitse? Havainnoidaan myös, kuinka asiakkaat ottavat yhteyttä organisaatioon ja kuinka pitkiä odotusajat ovat esimerkiksi asiakkaan pyytämän palaverin järjestämiseksi ja ymmärretäänkö asiakkaan tahtotilaa. Havainnot kirjataan erilliseen lomakkeeseen, josta voidaan seurata näiden etenemistä.

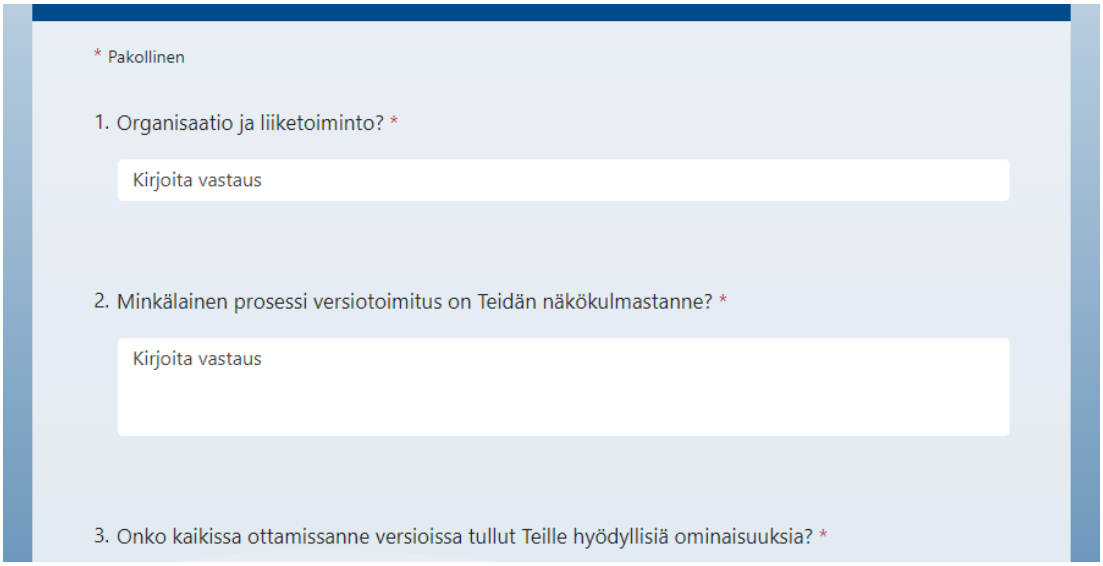
7 TUTKIMUKSEN SUORITTAMINEN

Tässä luvussa käsitellään, kuinka tutkimus ja tiedonkeräys suoritettiin. Näiden lisäksi esitellään, minkälaisia aineistoja analysoitiin ja mitä huomioita tehtiin tutkimuksen aikana. Tutkimuksen tekeminen aloitettiin luomalla haastattelut ja tarkentamalla niiden sisältöä teemahaastattelujen avulla. Haastatteluvastauksia odottaessa tehtiin dokumenttianalyysi.

7.1 Asiakashaastattelut

Asiakkaita haastateltiin kyselyhaastattelulla, joka toteutettiin Office Forms työkalulla. Haastateltavat valittiin organisaation palvelupäälliköiden toimesta, sekä pyrittiin saamaan mahdollisimman monta haastateltavaa organisaatiota mukaan. Vastausprosentti valituista haastateltavista organisaatioista oli 63 %. Haastateltavia organisaatioita valittiin kahdeksan ja vastauksia saatiin viisi.

Haastateltaville annettiin vastausaikaa reilu kolme viikkoa. Haastattelukysymykset lähetettiin asiakkaille sähköpostilla, joka sisälsi linkin Office Formsin haastattelulomakkeelle kuva 2. Muistutus haastatteluun vastaamisesta lähetettiin noin viikko ennen vastausajan päättymistä.



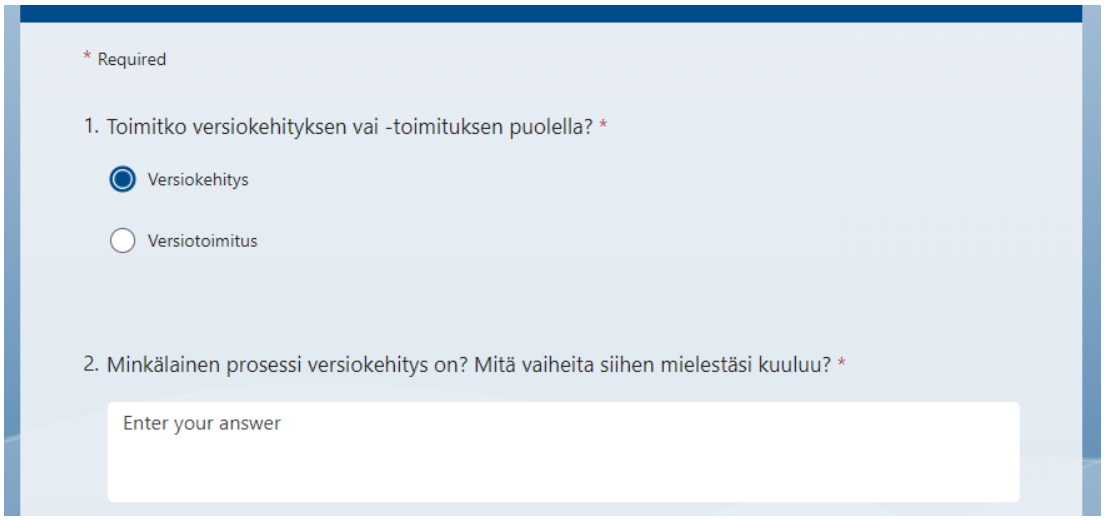
Kuva 2. Osa asiakaskyselyhaastattelun lomakkeesta.

Asiakashaastattelussa ei rajoitettu vastausten määrää organisaatioittain, joten yksi organisaatio voi tehdä useampia vastauksia. Haastatteluilla halutaan mahdollisimman paljon tietoa asiakasrajapinnasta ja kaikki kysymykset ovat avoimia kysymyksiä eli useampi vastaus antaa enemmän datapisteitä. Haastattelussa kuitenkin kysyttiin vastaajan organisaatio, jotta voidaan paremmin kartoittaa, mistä organisaatioista vastauksia on saatu. Vastauksia ei lopulta tullut useampia samalta organisaatiolta.

Haastattelukysymykset liittyivät asiakkaiden kokemukseen version käyttöönotosta ja toimitusprosessista, kokevatko he tulleensa huomioiduksi sekä työmäärään, jonka he joutuvat tekemään aina versiopäivityksen yhteydessä. Näiden lisäksi haastatteluilla oli tarkoitus löytää ne tärkeimmät asiat, joiden perusteella asiakkaat tekevät versionhankintapäätöksen. Kaikki haastattelukysymykset ovat liitteessä 1.

7.2 Organisaation sisäiset haastattelut

Organisaation sisällä tehtiin kaksi erilaista haastattelua, joista toinen oli keskijohdolle suunniteltu teemahaastattelu ja toinen oli asiantuntijatasolle tehtävä kyselyhaastattelu. Asiantuntijoiden haastattelut toteutettiin samalla tavalla, kuin asiakkaidenkin haastattelut eli toimitettiin Office Formsin haastattelulomakkeelle linkki, jonka avulla haastateltavat pääsivät vastaamaan. Kuva 3 näyttää haastattelulomakkeen alun. Vastausaika ja muistutuksen lähettämisajankohdat olivat samoja. Teemahaastattelu tehtiin kolmelle organisaation jäsenelle, jotka valittiin asemansa perusteella niin, että he toimivat osana versiokehitystä tai -toimitusta. Liitteessä 1 on asiantuntijahaastattelun ja teemahaastattelun kysymykset ja teemat.



Kuva 3. Organisaation asiantuntijahaastattelun lomakkeen alku.

Teemahaastattelun toteuttaminen kyselylomakkeella ei onnistu, joten se toteutettiin sähköpostilla. Haastattelua hyödynnettiin myös dokumenttianalyysissä, koska dokumentaation keräämisessä hyödynnettiin teemahaastattelussa saatuja vastauksia dokumenttien sijainneista ja tavoista löytää oikeat tehtävät Jirasta. Sähköpostihaastattelussa haastateltava ei saanut heti kaikkia kysymyksiä näkyville vaan keskustelua käytiin muutama kysymys kerrallaan. Ne perustuivat haastateltavien jo antamiin vastauksiin teemojen ympäriltä. Teemahaastatteluajankohta jakautui useammalle viikolle, koska uusia kysymyksiä ja tarkennustarpeita syntyi tutkimuksen edetessä.

Teemahaastattelun teemat ja asiantuntijahaastattelun kysymykset liittyivät siihen, kuinka organisaatio huomioi asiakkaat ja minkälainen prosessi versiokehitys ja toimitus on organisaation tasolla. Tavoitteena on löytää eroavaisuuksia siitä, kuinka organisaatio näkee versiokehityksen ja -toimitusprosessin keskijohdon ja asiantuntijatasoilla sekä miten nämä prosessit näkyvät asiakkaan kokemassa versiotoimitusprosessissa. Näiden lisäksi asiantuntijatasolta kysyttiin vielä tehtyjen toimien dokumentoinnista ja yleisesti kommunikaatiosta eri organisaatioyksiköiden välillä. Näiden perusteella nähdään, toimiiko versiokehityksessä ja -toimituksessa sisäinen kommunikointi ja tulevatko kaikki vaiheet dokumentoitua prosessin mukaisilla tavoilla.

Teemahaastattelusta saatiin paljon informaatiota organisaation toiminnasta ja prosesseista. Asiantuntijoiden kyselyhaastattelussa saatiin vain yksi vastaus ja haastateltaviksi valittiin 10 asiantuntijaa. Se ei tuottanut kovin laajasti lisää tietoa. Asiantuntijoiden kanssa pidettiin useita aivoriihiä projekteihin ja organisaation toimintaan liittyen ja näiden avulla havainnoinnin rooli tiedonkeruussa kasvoi varsinkin asiantuntijoiden osalta. Havainnoimalla saatiin paljon tietoa tukemaan tutkimusta organisaation versiokehitysprosessista.

7.3 Dokumenttianalyysi

Dokumenttianalyysi toteutettiin organisaation käytössä oleviin Jira ja Confluence järjestelmiin. Jira on tehtävienhallintajärjestelmä, jonka avulla hallitaan myös eri versioiden sisältämät ominaisuudet ja virheenkorjaukset. Confluence on organisaation sisäinen dokumentinhallintajärjestelmä. Siellä on projektin aikaisia muuttuvia dokumentteja, joita päivitetään projektin edetessä sekä siellä on myös muuttumattomia dokumentteja, kuten ohjeistuksia ja palaverimuistioita.

Organisaatio on dokumentoinut asiakaspalautteita toimitetuista versioista. Niistä saatiin hyvä lähtökohta sille, mikä asiakkaiden näkökulmasta oli suurin ongelma aina tietyssä versiossa, jonka asiakas oli ottanut. Dokumenteista löytyi myös versioiden toimituksen jälkeistä asiakasviestintää esimerkiksi ohjelmistovirheiden osalta.

Analyysissä hyödynnettiin lisäksi sähköpostiviestejä ja muita dokumentteja, jotka eivät vielä olleet julkaisukunnossa Confluenceen eli ne olivat vielä valmisteluvaiheessa tai niitä ei ole edes tarkoitus viedä sinne. Näitä ovat esimerkiksi Jenkingsin automaatiokanava tai käyttäjäpäivien paneelikeskustelujen muistiinpanot.

7.4 Havainnointi

Havainnointia toteutettiin tukevana tiedonkeruumenetelmänä ja havainnot kirjattiin ylös järjestelmällisesti OneNote-työkalulla ja muistiinpanoina. Havaintojen tarkoituksena oli tukea dokumenttianalyysillä ja haastatteluilla löydettyjä asioita. Havainnointi itsessään tuotti myös paljon uutta tietoa ja sen avulla pystyttiin paremmin tarkentamaan teema- ja kyselyhaastattelun kysymykset.

Havainnointia toteutettiin aivoriihien, palaverien ja versiotoimitusten tehtävien aikana. Havainnoinnin avulla myös löydettiin dokumenttianalyysissä tarvittavien tärkeiden tietojen sijainnit. Huomattiin, etteivät dokumentit ja tiedot välttämättä olleet suoraan intuitiolla löydettävissä vaan ne on lisätty tiimien ja projektikohtaisten sivujen alle. Tämä vaati paljon selvitystä organisaation eri asiantuntijoilta, jotta sisäiset dokumentit löydettiin.

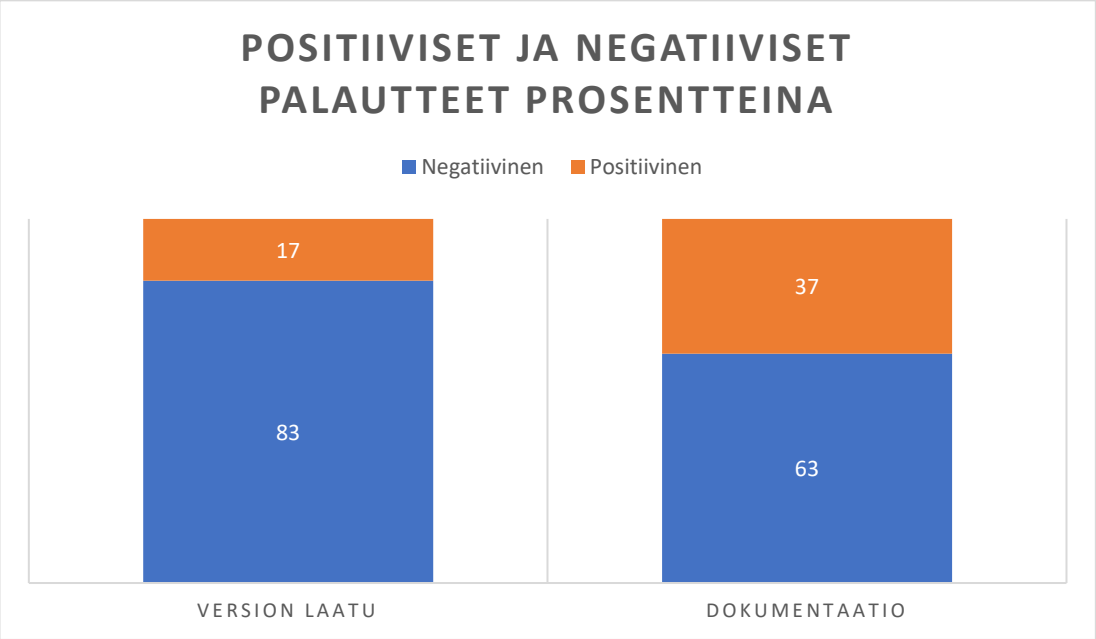
8 ASIAKASKOKEMUKSEEN LIITTYVÄT TULOKSET JA NIIDEN ANALYSOINTI

Tässä luvussa käydään läpi tutkimuksen tulokset, jotka löytyivät asiakkaiden kokemuksiin liittyen versiotoimituksiin ja dokumentaatioon. Asiakkaiden kokemukset käsitellään asiaperusteisesti ja ne ovat kerätty asiakashaastattelulla ja dokumenttianalyysillä. Dokumenttianalyysin pohjana toimivat versiotoimitusten jälkeiset palautteet sekä asiakasviestintä käyttäjäpäiviltä.

8.1 Asiakaspalautteet versiotoimituksista

Asiakkailta on kerätty vuoden 2023 alusta alkaen jokaisesta toimitetusta versiosta palautetta. Kysely ei ole ollut kovin kattava, mutta siitä on saatu paljon tietoa asiakkaille toimitetuista versioista. Se on keskittynyt yleisesti toimitetun version laatuun, dokumentaatioon ja yleisen palautteen keräämiseen.

Version käyttöönotto on aiheuttanut version toimitusten jälkeisten palautteiden mukaan paljon virheitä ja testausta. 60 % annetuista palautteista liittyi virheiden suureen lukumäärään toimituksessa ja 13 % palautteista kertoi virheiden aiheuttaneen paljon manuaalista työtä. Palautteet on jaettu dokumentaatioon ja version laatuun liittyen negatiivisiin ja positiivisiin palautteisiin prosentteina kuviossa 14. Virheiden lukumäärä on vaatinut myös paljon uusia korjausversioita. Ne aiheuttivat lisää testausta ja manuaalista työtä asiakkailla, koska uudet versiot menevät aina myös asiakastestauksen kautta tuotantoon. Laporte & April (2018, luku 1.4) kirjoittavat, että laatu on subjektiivista ja käyttäjät sekä asiakkaat näkevät laadun joko käytettävyyden tai toimitusvarmuuden näkökulmasta. Virheiden lukumäärä laskee sovelluksen laatutasoa niin toimituksen, kuin käytettävyyden suhteen. Vuonna 2023 13 %:ssa palautteista mainittiin myös järjestelmän hitaus, mutta nämä palautteet liittyivät ainoastaan yhteen toimitettuun versioon, joten tämä ongelma ei ole ollut jatkuvaa kaikkien versioiden kanssa. Annetuissa palautteissa 17 % sisälsi positiivista ja kiittävää kommenttia tuotteen tai toimituksen laadusta. Ne liittyivät siihen, että tukea ja korjauksia on saatu nopeasti sekä laadun kanssa ei ole ollut isoja ongelmia.

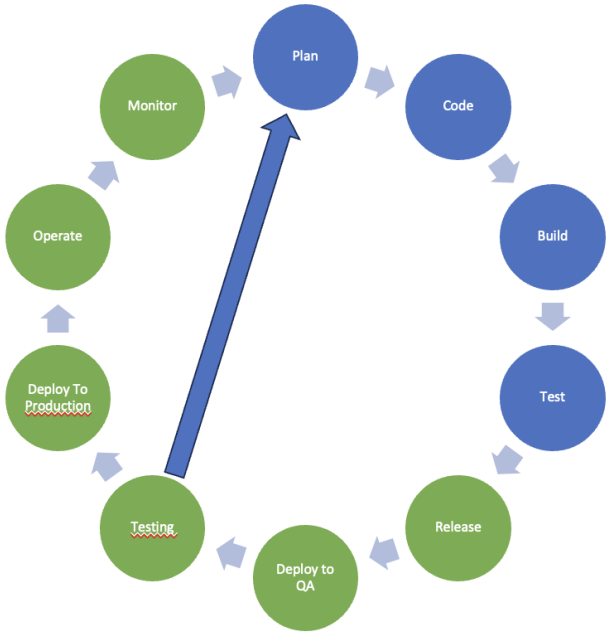


Kuvio 14. Palautteet yleisesti ja dokumentaatioon liittyen prosentteina.

Versiotoimitusprosessi asiakkaiden näkökulmasta etenee vaiheittain. Ensin tulee ennakkotietoa tulevasta julkaistavasta versiosta. Tähän liittyen asiakkaat eivät voi tehdä päätöstä version hankinnasta, koska osa julkaistuista ominaisuuksista ei välttämättä tule lopulliseen versioon. Hankintaa suunnitellaan kyllä etukäteen, mutta hankintapäätöstä ei tehdä ennen kuin lopullinen versiotiedote tulee. Siinä kerrotaan tietyn version koko varmistunut sisältö.

Hankintapäätöksen jälkeen sovitaan organisaation kanssa toimitusaikataulusta testausympäristöön, jossa toimitettavaa versiota testataan. Asiakkaat testaavat eri tavoilla versioita. Osa asiakkaista käyttää paljon resurssia ja aikaa siihen ja osa asiakkaista tekee vain perustoiminnallisuuksien testauksen. Mikäli virheitä on paljon, voi ajallisesti mennä jopa kuukausi ennen kuin voidaan toimittaa versio tuotantoon. Testikierroksia voi myös joutua tekemään useampia. Sen jälkeen, kun testaus on mennyt hyväksytysti läpi, sovitaan organisaation kanssa tuotantoympäristön päivityksestä. Se on yleensä ollut nopea ja helppo prosessi asiakkaalle. Keskinen ja Lipiäinen (2013, s. 30) mukaan ensimmäinen vaihe on tutkia asiakkaan kokemaa matkaa hankinnasta. Tämän avulla löydetään asiakaskäyttäytymisen nykytila ja mahdollisuudet vaikuttaa siihen. Asiakkaan kokema toimitus- ja käyttöönotto prosessit etenevät osittain

Yarlagaddan (2021, s. 115) DevOps kierrosten yhden osan mukaan eli julkaisu (release), käyttöönotto (deploy), käyttö (operate) ja seuranta (monitor). Asiakas kuitenkin joutuu testaamaan sovellusta julkaisun ja käyttöönoton välissä, mikä ei kuulu DevOps kierrokseen. Testauksessa voi löytyä virheitä, jotka pitää korjata ennen kuin versio voidaan viedä tuotantoon. Nämä ovat kuvattu kuviossa 15.



Kuvio 15. Asiakkaan näkemä osuus on merkitty vihreällä DevOps kierroksessa, jossa mukana QA toimitus, testaus ja uusintakierto (mukaillen Yarlagaddan, 2021, s. 115)

Versiotoimitusten ja -käyttöönoton vaatima ajankäyttö tuli esille asiakashaastatteluissa. Versiota pitää testata paljon ja se on usein manuaalista työtä. Myös löydettyjen virheiden korjaaminen vie oman aikansa, kun pitää odottaa organisaation korjauspäivityksiä. Joissain tapauksissa testaus pitää tehdä kahteen kertaan, kun testauksessa on löytynyt paljon virheitä sekä on jouduttu tekemään paljon korjauksia.

Käyttäjäpäiviltä saatu asiakasviesti liittyy samoihin ongelmiin, kuin versiopalautteista saadut tiedot eli virheiden määrä on liian suuri. Asiakkailla menee testauksiin paljon aikaa, kun löytyy paljon virheitä ja jatkuvaa manuaalista

testausta ei voida tehdä, koska se työllistää paljon. Asiakkaat testaavat aina uutta versiota erillisessä testausympäristössä ja siellä testauksiin menee 2–8 viikkoa riippuen asiakkaasta. Osa voi testata suurella massalla koko järjestelmän, jolloin testaus on nopeampaa, mutta joillain asiakkailla testausta tehdään pienemmällä ryhmällä. Tällöin testaus kestää pidempään.

Toimitusprosessin aikana asiakkaat ovat kokeneet saaneensa tarpeeksi tukea ja ovat olleet hyvin yhteydessä organisaation kanssa esimerkiksi palvelupäälliköiden kautta. Tukea on kaivattu version uusien muutosten läpikäynnin kanssa. Myös osa tehdyistä virhetehtävistä ei ole edennyt asiakkaiden näkökulmasta, mutta palvelupäälliköiden kautta tieto tehtävien etenemisestä on saatu.

Versioiden käyttöönottolukumäärä riippuu paljon asiakkaasta ja sen tarpeista. Asiakkaat ottavat yleisesti vain ne versiot käyttöön, joissa tulee uusia tarpeellisia ominaisuuksia tai korjauksia juuri heidän tarpeisiinsa. Ne riippuvat yleensä asiakkaan koosta. Mitä suurempi asiakas sen todennäköisemmin asiakkaalla on suuremmat tarpeet erilaisille ominaisuuksille, kuin pienillä asiakkailla.

Uuden version käyttöönottoa viivytellään, koska alussa se sisältää suurimman määrän korjattavaa. Mitä myöhemmin version ottaa käyttöön sitä korjatumpi se yleensä on, koska muut asiakkaat ovat löytäneet suurimman osan virheistä. Käyttöönotto voidaan tehdä myöhäisempään korjattuun versioon esimerkiksi 1.2:n sijaan 1.2.3. Testaus on haastavampaa heti version julkaisussa juuri virheiden lukumäärän vuoksi.

8.2 Asiakasnäkökulma versiodokumentointiin liittyen

Versiotoimitusten jälkeisistä kyselyistä 90 %:ssa oli vastattu myös version dokumentaatioon liittyen ja näistä 63 % oli negatiivisia ja 37 % positiivisia palautteita. Suurin ongelma dokumentaatioon liittyen oli, ettei tiedetty mihin mikäkin muutos on vaikuttanut tai mitä uusia asioita on tullut versiossa. Asiakkaiden

puolelta versiotiedot halutaan tarkemmalle tasolle, jotta löytyy tarkka tieto, mitä mikäkin korjaus on muuttanut.

Versiotiedotteista ilmeni myös samaa ongelmaa, kuin versiopalautteissa on kerrottu. Versiotiedotteiden ongelma on, ettei muutoksia ole kuvattu tarpeeksi tarkkaan. Niistä puuttuvat muuttuneet asiat tarkemmalla tasolla. Asiakkaat haluaisivat tiedot kaikista muutoksista ja tiedoista, joita versiossa tulee eli tiedon mitä on muuttunut verrattuna entiseen. Esimerkiksi ohjelmistorobotit menevät rikki, kun kaikkia muutoksia ei ole tarkkaan kuvattu niin, että asiakkaat voisivat ennalta korjata muutokset robotteihin. Versiotiedotteet myös julkaistaan liian lähellä version julkaisua. Asiakkaiden testaukset kestävät niin pitkään, että he eivät ehdi tutustumaan versiotiedotteisiin tarpeeksi ennen version julkaisua. Toiveena olisi myös versioinfotilaisuus, kun uusi versio julkaistaan, jolloin asiakkailla olisi mahdollisuus käydä organisaation kanssa yhdessä läpi uuden version muutokset.

8.3 Asiakaskokemus versiotoimitusten ja käyttöönoton yhteydessä

Versioiden sisältöön liittyvät kommentit ovat jakautuneet kahteen. Osalla asiakkaista versiopäivitys on voinut olla pakollinen tapa pitää sovellus tarpeeksi päivitettynä. Ilman alle vuoden vanhaa versiopäivitystä sovellusta ei ylläpidetä enää organisaation toimesta. Osa kertoo, että versioissa on tullut haluttuja ominaisuuksia tai korjauksia ja sen vuoksi on otettu uusi versio. Version sisältöön vaikuttaminen asiakkaan toimesta on myös jakautunut kahteen eli osa asiakkaista kokee, että voi vaikuttaa sisältöön ja osa kokee, että vaikutukset ovat vain pieniä. Yleisesti kuitenkin asiakkaat kokevat, että jonkinlainen vaikutus heillä on version tuleviin sisältöihin.

Kuten aikaisemmat kappaleet osoittavat, asiakkaat odottavat paljon enemmän versioiden laadulta ja dokumentaatiolta kuin he tällä hetkellä saavat. Asiakkaat eivät ole tyytyväisiä versiotoimitusten laatuun ja dokumentaatioon. Nämä molemmat puutteet aiheuttavat asiakkaille lisätöitä, kun pitää tehdä kattavaa testausta ja virheitä löytyy paljon. Myös dokumentaatio ei ole sellaisessa

kunnossa, että asiakas löytäisi sieltä kaiken tarvitsemansa vaan se on liian yleisellä tasolla.

Asiakaskokemus on myös versiotoimitusten ja käyttöönoton osalta huono, koska asiakkaat eivät voi luottaa siihen, että toimitetun version lyhyt testaus olisi riittävä. Suuressa osassa versioita löytyy paljon virheitä ja niiden korjaamiseen menee aikaa. Dokumentaation puutteellisuus aiheuttaa asiakkaissa epävarmuutta, joka näkyy huonona asiakaskokemuksena. Näiden asioiden korjaaminen ja sovelluksen käyttöönoton helpottaminen parantaisi myös uuden sovellusversion hankintamielekkyyttä.

Kuten Saarijärvi & Puustinen (2020, s. 38) kertovat, asiakastyytyväisyys ja koettu asiakaskokemus vaikuttavat asiakaskäyttäytymiseen. Voidaan tämän perusteella todeta, että asiakaskokemuksen ja asiakastyytyväisyyden parantaminen kasvattaisi myös uusien versioiden toimituslukumääriä.

Tyytyväisiä asiakkaat olivat kuitenkin siihen, että organisaatiolta saa apua tarvittaessa sekä versioiden lopullinen laatu on ihan hyvä, kun virheet on korjattu. Käytännössä siis viimeisenä version käyttöönottavat asiakkaat saavat parhaan asiakaskokemuksen, kun virheiden lukumäärä on pienempi, kuin alkuperäisessä versiotoimituksessa on. Riccominin ja Ryaboyn (2021, kohta Software delivery phases) mukaan jatkuvaa kehitystä toteutetaan juuri keräämällä asiakkailta palautteita toimitusprosesseista ja hyödyntämällä sitä seuraavissa versiotoimituksissa.

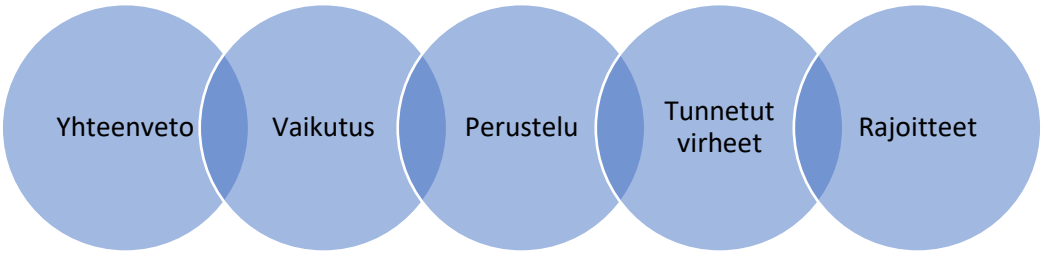
9 ORGANISAATION SISÄINEN TOIMIVUUS TOIMITUSTEN JA KÄYTTÖÖNOTON OSALTA

Tässä luvussa käsitellään organisaation sisäisten näkökulmien perusteella tutkimuksen tulokset. Nämä tulokset kerättiin dokumenttianalyysillä, havainnoimalla ja teemahaastatteluilla. Vähäisten asiantuntijavastausten vuoksi niiden käyttö on yhdistetty vain toimitusprosessin analysointiin.

9.1 Versiotiedotteet

Ennen virallisia versiotiedotteita asiakkaalle julkaistaan ennakkoversiotiedotteet, jotka kuvaavat, mitä seuraavaan versioon pyritään tuomaan, mutta ei olla vielä täysin varmoja tulevatko kaikki siinä olevat asiat mukaan. Lopulliset versiotiedotteet on tehty kaikista julkaistuista versioista ja niihin on kuvattu kaikki muutokset. Ne ovat toimitettu asiakkaille, kun julkaisu on tehty ja asiakas on pystynyt niihin tutustumaan ennen hankintapäätöksen tekemistä. Versiotiedotteiden laatu on yleisesti epätasainen. Osa niiden kentistä kuvaa hyvin muuttuneet asiat, mutta osa kentistä on liian yleisellä tasolla, mikä nähtiin myös asiakkaiden antamista versiopalautteista.

Versiotiedotteet kirjoitetaan suoraan käyttäjille ja niissä pitää olla mukana mitä ja miksi muutettiin jotain sekä miten toiminnallisuus toimi ennen ja miten se on nyt muuttunut. Mukana on hyvä olla siis yhteenveto, vaikutus, perustelu sekä tunnetut virheet ja rajoitteet. Kuviossa 16 on kuvattu näiden suhteet. (Bhatti, 2021, kappale 2.)



Kuvio 16. Versiotiedotteen sisältö yleisesti (mukaillen Bhatti, 2021, kappale 2)

Organisaatiolla versiotiedotteessa on kuvattu julkaistu vaatimus ja siihen liittyvät ominaisuudet, sekä tieto onko ominaisuus erikseen maksullinen vai ei. Dokumentista kuitenkin puuttuvat kokonaan versiotiedotteen kohdat vaikutus, perustelu sekä tunnetut rajoitteet ja virheet. Nämä puutteet osuvat yhteen asiakkailta saadun palautteen kanssa siitä, että dokumentaatio ei ole ollut versiotoimituksissa riittävällä tasolla. Dokumentaatiota parantamalla asiakkaiden version hankintapäätös ja käyttöönotto helpottuu, kun versioiden sisältö on selkeämpi asiakkaille.

9.2 Sovelluskehitysprosessi

Sovellus on Starkin (2011, luku 1.2) kuvaamassa keskivaiheessa (middle of life) elinkaartaan eli sovellus on käytössä laajasti ja hyvin tuettuna. Sovellukselle ei ole korvaajaa suunnitteilla ja uusien ominaisuuksien kehitystä tapahtuu jatkuvasti. Toimialan muuttuessa myös sovelluksen laatua ylläpidetään vastaamaan uusien teknologioiden ja asiakkaiden vaatimuksiin.

Sovelluskehityksessä on osittain käytössä CI-prosessi eli Meroden (2023, luku 2) mukaan pääkehityshaaraan viedään uutta koodia. Käynnistetään rakennusprosessi, jonka jälkeen ajetaan automaattitestit sovellukseen. Kokonaisuudessaan CI-prosessi ei ole kuitenkaan käytössä ja sen käyttöönotto on haastavaa, koska sovellus on niin suuri kokonaisuus. Vaikka sovellus on arkkitehtuurisesti ajateltu modulaariseksi niin kehityksen kannalta se ei kuitenkaan sitä ole. Teknisesti sovelluksen rakentaminen vie paljon aikaa, koska kuten Rostoniec

(2023) mainitsee modulaarisessa monoliittisovelluksessa pitää rakentaa aina koko koodi eikä vain tiettyjä osia siitä ja tämä aiheuttaa hitautta kehitysprosessille. CI-prosessin testauksessa ei myöskään ajeta koko testikantaa läpi, vaan siinä ajetaan pelkästään yksikkötestejä. CI-prosessi hallitaan tällä hetkellä ajallisesti eikä koko prosessia ajeta suoraan esimerkiksi yhdessä kanavassa tapahtumien perusteella. Nopeampi kehitysprosessi tarkoittaisi, että asiakkaat saisivat haluamiaan toiminnallisuuksia nopeammin. Näin he kokisivat voineensa paremmin vaikuttaa versioiden sisältöön.

Kehitysprosessista tulee palautetta myöhäisessä vaiheessa ja kehitysprosessissa on yhä vesiputousmallin tyyppisiä ratkaisuja vaikkakin ketterä malli on kehityspuolella käytössä. Näitä ratkaisuja ovat esimerkiksi Diman ja Maassenin (2018, luku 6) mukaan asiakaspalautteen vastaanottovaihe ja testaus, jotka prosessimaisesti tehdään vasta myöhäisessä vaiheessa. Ketterissä kehitysmalleissa asiakaspalautteet tulevat aikaisessa vaiheessa kehitystä, jolloin kehitystä voidaan muuttaa vielä ennen koko versiojulkaisua. Prosessin mukaiset automaatiotestit tehdään muutaman päivän välein. Kuitenkin yksikkötestit ajetaan aina koodimuutosten yhteydessä. Testiautomaatio on organisaatiolle otettu käyttöön, mutta se ei ole vielä täysin käytössä, joten sen kehitystä jatketaan yhä. Testauksen parantaminen takaisi asiakkaille paremman laadun lopullisessa versiossa ja se helpottaisi asiakkaiden käyttöönottoprosessia.

Standardin ohjelmistokehityksen vaatimat kokonaisuudet ovat tuotteen tehtävälista, tehtävien roolitukset, tehtävienhallintajärjestelmä, versionhallinta sekä osittain myös tuotantoon viennin automaatio (Digi- ja väestötietovirasto, 2020). Organisaation tietosuoja ja -turvatestauksia tehdään ulkopuolisen siihen erikoistuneen yrityksen voimin. Mikäli löytyy jotain puutteita tietoturvaan liittyen, niistä tehdään työtehtävät ja ne ajastetaan kehityksen tehtävälistalle. Käytössä ei kuitenkaan suoraan ole standardin mukaista uhkamallinnusta, kun uutta ominaisuutta kehitetään tai olemassa olevaa muutetaan. Yleisesti kehitysprosessi kuitenkin mukailee Digi- ja väestötietokeskuksen (2020) turvallisen sovelluskehityksen käsikirjaa. Laadukkaalle sovelluskehitykselle on hyvät lähtökohdat. Asiakastyytyväisyyden ja -kommunikoinnin kannalta kehityksen ja tehtävien seurannan työkalut ovat organisaatiolla käytössä. Nämä näkyivätkin

asiakkailta saaduissa tuloksissa, kun asiakkaat kokivat saaneensa tarpeeksi tukea organisaatiolta esimerkiksi palvelupäälliköiltä.

Lopullinen versio on virheettömämpi, kun kehitysprosessi huomioi laatua paremmin. Tällöin asiakkaan tekemä testaus voi olla pienempää sekä toimitus- ja käyttöönottokokemus ovat parempia. Yhdessä nämä parantavat uuden version hankinnan asiakaskokemusta ja -tyytyväisyyttä.

9.3 Testaus ja virheet

Yksikkötestien määrä näin suurella sovelluksella aiheutti epäselvyyksiä, kun ei ole tiedossa kuinka suuren osan koodista yksikkötestit kattavat. Yksikkötestien lukumäärä on korkea, mutta niitä tarvittaisiin yhä enemmän tai kattavammin, koska testattava sovellus on suuri. Kehityksessä useampia yksikkötestejä voi mennä rikki ja se aiheuttaa regressiota eli virhettä jo toimiviin osuuksiin koodissa. Yksikkötestit ovat näin ollen liian suuria eli niitä pitäisi olla enemmän ja niiden pitäisi olla pienempiä. Smartbearin (n.d.) mukaan yksikkötestit ovat pienimpiä testattavia kokonaisuuksia, jotka voidaan erottaa kokonaisuudesta.

Automaatiotestaus suorittaa prosessin, jonka aikana ajetaan yksikkötestit ja laajemmat prosessitestit. Mikäli näitä ei ole kirjoitettu tarpeeksi kattavasti, ei löydetä kaikkia virheitä automaatiolla ja niitä löytyy vasta asiakkaan testauksessa. Jokaisessa versiossa löydetään kymmeniä uusia virheitä, joiden korjaukseen käytetään aina aikaa. Keskimäärin virheitä tuli 1,7 kappaletta jokaista uutta ominaisuutta kohti. Osaan versiosta on tehty jopa 19 korjauspäivitystä ennen kuin kaikki virheet on saatu korjattua. Virheet eivät aina tule uusimpaan ohjelmistokoodiin, vaan niitä tulee myös jo toimiviin osuuksiin. Tästäkin syystä testaus olisi tärkeä kohdistaa koko järjestelmään. Testauksessa yksi haaste on ollut, että yksikkötestien viemä koodimäärä alkaa olemaan laaja ja ohjelman rakentaminen ennen testien ajoa vie jo noin 20 minuuttia. Tällä hetkellä automaatiotestaus, jolla ajetaan suurempia prosesseja läpi kattaa noin 26,4 % koko koodikannasta. Automaattisempi ja laajempi testausprosessi vähentää virheitä lopullisessa versiossa ja asiakkaiden testaus olisi kevyempi.

Järjestelmän arkkitehtuuriratkaisun vuoksi koko järjestelmä pitää aina rakentaa eikä voida rakentaa vain sitä osaa järjestelmästä, johon muutokset tehdään. Sen arkkitehtuurinen muutos on todella kova panostus, mutta järjestelmä voitaisiin pilkkoa mikropalveluiksi, jolloin sen testaus ja rakentaminen nopeutuisivat, kun rakennettavan järjestelmän osuus on huomattavasti pienempi.

9.4 Versiotoimitusprosessi

Versiotoimitus asiakasympäristöön tehdään aina samalla tavalla asiakkaiden kanssa. Ensimmäinen vaihe on tehdä DevOps-tiimille tehtävä tehtävienhallintajärjestelmään. Tämän jälkeen tiimi sopii asiakkaan kanssa, milloin päivitys tehdään ja samalla sovitaan muiden tiimien kanssa aikatauluista. Muita tiimejä tarvitaan, mikäli otetaan asiakasympäristöstä esimerkiksi erillinen varmuuskopio. Toimitukset tehdään erikseen asiakkaiden testaus- ja laadunhallintaympäristöihin, minkä jälkeen sovitaan viennistä tuotantoympäristöihin. Mikäli päivityksen yhteydessä tulee raskaita tietokantaoperaatioita niin tuotannon päivitys voidaan sopia viikonlopuksi, jolloin on enemmän aikaa toteuttaa päivitys.

Organisaatiolla on käytössä Jenkings-sovellus, jonka avulla hallitaan automaattista kääntämistä, testausta ja toimitusta, sekä mahdolliset muut automatiikkaa tarvitsevat kanavat eli Jenkings termein pipelinet. Aina kun ohjelmakoodi muuttuu päähaarassa, Jenkings alkaa suorittaa prosessia, jossa sovellus rakennetaan ja testataan. Testauksissa ajetaan yksikkötestit läpi ja automaattitestaus, jossa ajetaan suurempia prosessitestejä suoritetaan vain osaan ympäristöihin kuten laadunhallintaympäristöön.

Arkkitehtuurisesti järjestelmä ei vielä ole sillä tasolla, että voitaisiin ottaa käyttöön jatkuva toimitusprosessi, koska järjestelmää ei voida kehittää ja toimittaa pienissä osissa ilman käyttökatkoja. Mikäli kokonainen CI/CD-prosessi haluttaisiin käyttöön, se vaatisi järjestelmän arkkitehtuurisen muutoksen ja rikkomisen pienemmiksi yksittäisiksi kokonaisuuksiksi. Klotins ym. (2022, s. 32–33)

mainitsee lisäksi, että muutos vaikuttaisi myös asiakkaiden asiakkaisiin. Prosessin käyttöönotto vaatisi muutosta myös asiakkaiden ja organisaation nykyiseen ajattelutapaan, koska ei olisi enää erillisiä toimitusprosesseja. Red Hatin (2023) mukaan CI/CD-prosessin käyttöönoton jälkeen toimituksia voitaisiin viedä kaikille asiakkaille samanaikaisesti ja laatua voitaisiin ylläpitää korkeana reaaliaikaisesti. Kaikilla asiakkailla olisi aina uusimmat versiot käytössä ja uusimmat korjaukset, sekä erillisiä testaus- ja toimitusprosesseja ei tarvittaisi asiakkailla.

9.5 Toimittajan kehittäminen

Organisaatiolla on avoimet keskusteluvälit asiakkaiden kanssa. Asiakkailla on omat palvelupäälliköt sekä tehtävienhallintajärjestelmän kautta asiakkaat voivat luoda uusia palvelupyyntöjä. Asiakkaille tarjotaan myös käyttäjäpäiviä, joiden kautta avoin keskustelu kaikkien asiakkaiden ja organisaation välillä on helppoa.

Asiakkaat toimivat hyvin organisaation kanssa ja iso osa tarvittavista kehityshankkeista ja projekteista tulee asiakkailta. Kyse ei ole pelkästään markkinavaatimuksista vaan myös asiakkaiden omista tarpeista kehittää omaa palveluaan. Tämä asiakkaiden halu omaan palvelukehitykseen aiheuttaa tiukemmat vaatimukset myös sovelluskehitykseen organisaation puolella. Sovelluksen on vastattava niihin liiketoiminnallisiin tarpeisiin, joita asiakkailta tulee ja asiakkaat odottavat näiden vaatimusten toteuttamista.

Tämä luo terveen suhteen toimittajan ja asiakkaan välille. Asiakkaan vaatimuksiin vastataan, se auttaa asiakasta kehittämään omaa palveluaan sekä toimittajaa kehittämään sovellusta koko markkinoita silmällä pitäen. Nieminen (2016, luku 5.3) kirjoittaa, että toimittajan kehittämisessä yhteistyössä on tärkeää löytää arvon tuottoa molemmille osapuolille. Yleensä yhden asiakkaan tarpeet ovat sellaisia, joiden avulla voidaan palvella myös muita asiakkaita. Näin asiakkaat saavat tarvitsemansa toiminnallisuudet ja organisaatio saa paremmin koko markkinoiden vaatimuksiin vastaavan tuotteen kehitettyä.

10 KEHITYSEHDOTUKSET

Tässä luvussa käydään läpi löytyneet kehitysehdotukset organisaatiolle. Ensin selvitetään sovelluksen laatuun liittyvät kehitysideat. Tämän jälkeen käydään läpi dokumentaatioon liittyvät kehitystarpeet. Lopuksi esitellään tarvittavat muutokset sovelluskehityksen ja -toimituksen puolelle. Näiden toimenpiteiden avulla organisaation on mahdollista parantaa asiakastyytyväisyyttä ja tehostaa sovellustoimituksia.

10.1 Laatuun liittyvät kehitysideat

Tutkimuksesta käy ilmi, että suuri ongelma sovellusversioiden käyttöönotossa liittyy toimitetun sovelluksen laatuun. Asiakkaat joutuvat tekemään liikaa manuaalista työtä testausten suhteen. He eivät voi luottaa siihen, että toimitettu sovellus on niin laadukas, että vain pienellä testauksella voidaan edetä tuotantoon. Sovellusversioiden laadun korjaaminen on yksi suurimmista tavoista parantaa asiakastyytyväisyyttä, koska se vähentää asiakkaan tekemää testaustyötä sekä käyttöönotto olisi myös nopeampi.

Bergströmin ja Leppäsen (2021, s. 354–365) mukaan pidemmissä asiakassuhteissa laatu vaikuttaa enemmän asiakkuudessa kuin hinta. Version laatua pitää saada paremmaksi ja sovelluksen käyttöönottoa helpotettua asiakkaille niin, etteivät asiakkaat joudu panostamaan suurta työmäärää testaukseen jokaisen versiojulkaisun yhteydessä. Versiolaatua voidaan parantaa ensin esimerkiksi Smartbearin (n.d.) mukaan järkevöittämällä yksikkötestejä niin, että ne kattavat koko sovelluksen tarpeen. Ensimmäiseksi pitää selvittää paljonko yksikkötestit kattavat sovelluksen koodista. Ne ovat myös liian suuria, koska useampi yksikkötesti menee rikki ohjelmistokoodin muutoksissa. Nykyiset yksikkötestit pitäisi käydä läpi ja pyrkiä pilkkomaan pienemmiksi niin, että yksittäinen testi testaa vain yhtä pientä yksittäistä toimintoa.

Spillner & Linz (2021, luku 2.1) mainitsevat, että yksikään sovellus ei ole virheetön. Virheet johtuvat yleensä inhimillisestä virheestä, kiireisestä

aikataulusta, sovelluksen tai toiminnallisuuden monimutkaisuudesta. Sovellus on erittäin laaja monoliittinen kokonaisuus, eli monimutkaisuuden vähentäminen sovelluksessa vähentäisi syntyneiden virheiden lukumäärää. Kehittäjien pitää miettiä montaa asiaa samanaikaisesti, kun ominaisuutta ohjelmoidaan.

Nykyään pilvipalveluympäristöissä monoliittisen sovelluksen kehittäminen ei ole enää paras ratkaisu. Monimutkaisuutta voidaan vähentää arkkitehtuurisella muutoksella eli rikkomalla sovellus pienempiin itsenäisiin kokonaisuuksiin, joita voidaan rakentaa erikseen koko muusta sovelluksesta. (Familiar, 2015, kohta microservices.)

Testiautomaatio on organisaatiolla uusi toimintamalli ja selkeästi sille on tarvetta, koska laatua halutaan parantaa. Testiautomaation testit eivät kuitenkaan vielä kata kuin noin 26,4 % ohjelmistokoodista, joten niitä tarvitaan paljon lisää. Mitä paremmin koko koodia voidaan testata jokaisen muutoksen, versiokorjauksen ja haaraan viennin yhteydessä. Sen paremmin virheet jäävät kiinni. Testiautomaation kattavuus pitäisi saada suurimpaan osaan käyttötapauksista, jotta suurin osa ohjelmistosta testataan kehityksen aikana. Testien testitapaukset pitää suunnitella myös tarkasti. McMillian (4.12.2023) toteaa artikkelissa, että hyvä testikattavuus kattaa koodin ja toiminnallisuuksien testauksen laajasti, on tasapainoinen, erilaisia testejä sisältävä, mitattava ja muutettava sekä helppo ylläpitää.

Joka kerta, kun ohjelmoija vie koodia päähaaraan ajetaan yksikkötestit. Testiautomaatiotestaukset ajetaan läpi ajallisesti aina öisin. Testiautomaatio pitäisi ajaa useammin sekä kattavammin läpi, näin ohjelmoija näkisi heti koodin tehtyään, mitkä testit menevät virheeseen (Red Hat, 2023). Virheet voitaisiin korjata jo ennen, kuin ne menevät erillisiin testiympäristöihin asti. Virheiden korjaus ennen versiointia olisi tärkeää, koska silloin ei tarvitsisi julkaista paljon uusia korjauspäivityksiä.

Laatuun liittyen voisi harkita kehitystavan muuttamista testivetoiseksi kehitykseksi. Ainakin osa tämän mallin toimintatavoista auttaisi testien kirjoittamisessa sekä laadun parantamisessa. Testivetoisessa kehitysmallissa testit

kirjoitettaisiin ennen, kuin itse koodia aletaan tehdä ja tämä takaa, että tehty ohjelmistokoodi ei aiheuttaisi lisää virheitä, koska uudelle ominaisuudelle olisi jo testaukset valmiina (Dima & Maassen, 2018, s. 318; Parani, 2017, luku 1). SmartBearin (n.d.) mukaan TDD mallin käyttöönotto on kuitenkin hidasta, koska se vaatii kehittäjiltä uuden kehitysmallin opettelua. Tämä malli ei myöskään poista tarvetta jatkuvalle testaukselle sekä testiautomaatiolle vaan takaa sen, että kaikkiin uusiin ohjelmistokoodeihin olisi jo testitapaukset valmiina. Näin uutta testien ulkopuolella olevaa ohjelmistokoodia ei syntyisi.

10.2 Dokumentoinnin kehitysideat

Asiakasviestinnästä käy selkeästi ilmi, että organisaation tuottama versiodokumentaatio on puutteellista. Versiotiedotteisiin pitää lisätä paljon enemmän tietoa. Liitteessä 2 on ehdotus versiotiedotteen pohjaksi ominaisuuksien ja virheiden osalta. Siinä on mukana asiakkaiden tarpeita koskevat tiedot, jotka nyt puuttuvat versiotiedotteista.

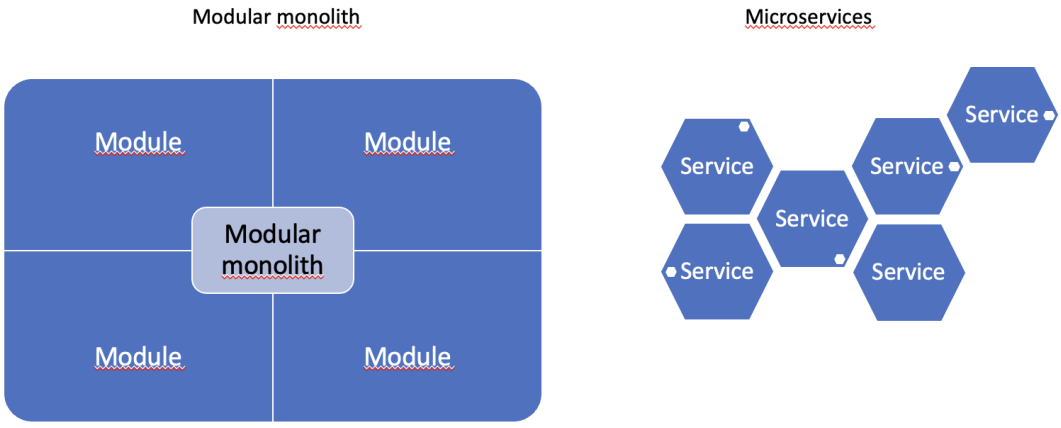
Versiotiedotteiden rakennetta pitää muuttaa niin, että nykyisen yhteenvedon lisäksi niihin lisättäisiin Bhatti (2021, kappale 2) mukaiset kohdat; miten asia oli ennen, miten se on nyt, miksi, kuinka käytetään, rajoitteet ja tekniset muutokset. Kaksi ensimmäistä kohtaa kertovat miten asia on muuttunut siitä, miten se oli ennen. Näin asiakkaat saavat tarkan tiedon siitä, mikä ominaisuus on muuttunut ja kuinka se on muuttunut. Miksi-kohta kuvaa tarpeen muutokselle. Uuden toiminnallisuuden käyttö pitää myös opastaa, koska kaikille käyttäjille ei välttämättä ole suoraan selkeä käsitys, kuinka käytetään ominaisuutta, jonka vaatimus on lähtöisin toisen asiakkaan tarpeista. Tähän kohtaan ei kuitenkaan pidä lisätä käyttöohjetta ominaisuudelle vaan yleisesti, miten kyseistä ominaisuutta voitaisiin hyödyntää. Tarvittaessa voidaan lisätä linkki ominaisuuden käyttöohjeeseen. Tämän jälkeen kuvataan uuden ominaisuuden tunnetut rajoitteet eli ne asiat, joihin ominaisuutta ei voida käyttää tai ne tarpeet, jotka on oltava käytössä ennen, kuin ominaisuus voidaan ottaa käyttöön. Tekniset muutokset kohtaan lisätään lyhyesti muuttuneet tekniset asiat, jos

esimerkiksi tietokantaan tai käyttöliittymään on tehty muutoksia. (Bhatti, 2021, kappale 2; Vasserman, 2023.)

Asiakkailla on myös toive päästä keskustelemaan organisaation kanssa tulevista versioista, jolloin he voisivat suoraan kysyä tarkentavia tietoja. Niemisen (2016, luku 5.3) mukaan yhteistyötä pitää tehdä tiiviisti asiakkaiden kanssa, sekä organisaation tulee sitoutua kehittämään asiakkaan tarvitsemaa kokonaisuutta ja näin kasvattaa myös omaa kilpailukykyään. Tähän tapaukseen versiotiedotteiden päivitysten lisäksi voidaan ottaa käyttöön versioinfotilaisuus. Versioinfotilaisuudessa organisaatio kutsuu asiakkaat tilaisuuteen, jossa käydään läpi uudessa versiossa tulleet ominaisuudet ja niiden käyttö. Näiden lisäksi tarve on käydä läpi myös ongelmatilanteet, jotka ovat tiedossa ja joihin pitää tehdä korjauspäivityksiä. Versioinfotilaisuudessa on tärkeää varata aikaa myös seuraavan version sisällölle ja aikataululle, jotta asiakkailla on tietoa myös tulevan version mahdollisista ominaisuuksista ja julkaisuaikatauluista.

10.3 Sovelluskehityksen ja -toimituksen kehitysideat

Laadun ja testauksen määrän kasvattamisen lisäksi sovelluksessa on tehtävä arkkitehtuurisia muutoksia. Sovellus on käytännössä modulaarinen monoliitti. Tämä tarkoittaa, että aina kun sovellus rakennetaan, koko sovellus pitää rakentaa (Rostoniec, 2023). Arkkitehtuurisesti sovellus pitäisi suunnitella uudelleen mikropalveluarkkitehtuuriin, jolloin sovelluksesta voitaisiin rakentaa tiettyjä osia eikä koko sovellusta kerrallaan. Kuviossa 17 on kuvattu modulaarisen monoliitin ja mikropalveluiden erot yksinkertaisesti. Rostoniecin (2023) mukaan ensimmäinen vaihe on jo tehty, koska sovellus on modulaarinen. Mikropalveluarkkitehtuurisen sovelluksen ohjelmointi ja testaus nopeutuisivat huomattavasti, kun koodimuutoksia tehdessä voitaisiin rakentaa ainoastaan se osuus sovelluksesta, johon muutoksia on tehty ja testit voitaisiin ajaa nopeammin (Familiar, 2015, kohta microservices). Myös testien sisältämä koodimäärä ei hidastaisi sovelluksen rakentamista, koska rakennettavat osuudet olisivat aina huomattavasti pienempiä, kuin nykyisellään.



Kuvio 17. Modulaarinen monoliittinen sovellus ja mikropalvelusovellus (mukaillen Familiar, 2015, kohta microservices; Rostoniec, 2023)

Organisaatiossa voitaisiin ottaa käyttöön kokonaisuudessaan CI-prosessi. Kun saadaan toteutettua testien lukumäärää kattamaan suuremman osan järjestelmää, näiden ajaminen voidaan hoitaa usein ja automaattisesti. CI-prosessissa ei ole kokonaisuudessaan käytössä esimerkiksi testiautomaation sisältämien testien ajamista, kun muutoksia koodiin tehdään. Näiden käyttöönotto suoraan sisäisessä julkaisu- ja rakennuskanavassa auttaisi virheiden löytämisessä jo ennen, kuin niistä tarvitsee tehdä erillisiä tehtäviä tehtävien hallintaan. Automaatiotestit ajetaan aina tiettyyn aikaan, kun ne voitaisiin ajaa useammin automaattisesti suoraan rakennuskanavassa ilman erillisiä aikataulutettuja ajoja.

Testiautomaation ja CI-prosessin avulla organisaatio pääsee paremmin pois vesiputousmallisesta kehityksestä, kun testauksen laatu ja nopeus paranevat jo kehityksen aikana. Tällöin löydetyt virheet voidaan korjata jo ennen, kuin virhe päätyy versioon asti eli iteratiivinen kehitysmalli toimii nopeammin. Myös kehityksen kannalta ei ole yhtenäisesti huomioitu prosessissa asiakaspalautetta kehityksen aikana, vaan asiakas otetaan välillä huomioon vasta asiakastesteissä, kun ominaisuus on jo kehitetty ja viety versioon. Tämä aiheuttaa virhekuormaa ja korjauksia pitää tehdä laajemmin jo julkaistuun versioon korjauspäivityksinä. Westfallin (2009, s. 136) mukaan iteraatiokierroksissa huomioidaan palaute eri lähteistä esimerkiksi asiakkaalta ja toisilta kehittäjiltä,

minkä jälkeen ominaisuus julkaistaan ja asiakas voi sitä testata ja mahdollisesti ottaa käyttöön. Organisaatiolla kuitenkin asiakas pääsee testaamaan ominaisuutta vasta testiympäristössä eli silloin, kun sovellusversion julkaisu on jo tehty.

CI/CD-prosessin käyttöönotto olisi nykyaikainen tehokas tapa kehittää ja toimittaa sovellusta. Tämä kuitenkin vaatisi suuren muutoksen sovelluksen arkkitehtuuriin sekä koko alan sovelluskehitysajatteluun. Klotins ym. (2022, s. 32–34) mainitsevat, että CI/CD-prosessin käyttöönotto on hankalaa. Tämän vuoksi pitäisi tehdä erillinen tutkimus vielä siitä, kuinka se voitaisiin kokonaisuudessaan ottaa käyttöön. CI/CD vaatii laajan testipaketin, joten yksikkötestien ja automaatiotestien tulisi kattaa käytännössä koko sovellus (Merode, 2023, luku 4). Myös sovelluksen arkkitehtuuri tulee muuttaa sellaiseksi, että siitä voidaan rakentaa ja päivittää aina tiettyjä osia koskematta muihin sovelluksen osiin. Nämä päivitykset pitää myös kyetä hoitamaan niin, ettei tarvita erillisiä käyttökatkoja päivitysten aikana, vaan niitä voidaan viedä tuotantoon jatkuvasti (Merode, 2023, luku 2). Tämän prosessin käyttöönotto vaatisi todella suuria muutoksia sovelluksen arkkitehtuuriin, koska se on suunniteltu monoliittisen järjestelmän ehdoilla.

10.4 Vastaukset tutkimuskysymyksiin

Opinnäytetyön tutkimuskysymykset liittyivät sovelluksen kehityksen sekä versiotoimitusten tehostamiseen ja niiden käyttöönoton parantamiseen asiakkaiden näkökulmasta. Molempiin tutkimuskysymyksiin löydettiin tavoitteen mukaiset vastaukset ja organisaatiolle valmisteltiin kehitysehdotukset näiden kysymysten pohjalta. Taulukossa 1 löydökset, tietolähteet ja niihin liittyvät kehitysehdotukset ja kommentit ovat yhdistetty toisiinsa.

Taulukko 1. Löydökset ja mahdolliset kehitysehdotukset.

Löydös	Tietolähteet	Kehitysehdotukset ja huomiot
Asiakkaat eivät saa versioista tarpeeksi kattavasti tietoa.	Asiakaskyselyhaastattelut. Dokumenttianalyysi: Asiakasviestintä käyttäjäpäiviltä ja asiakaspalautteet	Versiotiedotteen päivitys, versioinfotilaisuudet
Asiakastestaus vie paljon aikaa.	Asiakaskyselyhaastattelut. Dokumenttianalyysi: Asiakasviestintä käyttäjäpäiviltä	Testauksen parantaminen organisaatiossa (yksikkötestauksen ja prosessitestauksen kattavuuden kasvattaminen). Virheiden löytäminen ja korjaaminen aikaisemmassa vaiheessa.
Version toimitusprosessi toimii hyvin.	Asiakaskyselyhaastattelut. Teemahaastattelut	Ongelmien korjauksen jälkeen tämä prosessi toimii hyvin. Ongelmat ovat pääosin laadussa ja dokumentaatiossa.
Tuki asiakkaille toimitusprosessin aikana toimii hyvin.	Asiakaskyselyhaastattelut	Tuki versiotoimituksen yhteydessä on ollut hyvää.
Asiakkaat kokevat, etteivät voi vaikuttaa versiosisältöön	Asiakaskyselyhaastattelut. Dokumenttianalyysi: Jira, Confluence versiotiedotteet. Havainnointi.	Asiakkaat kokevat, ettei heillä ole paljon vaikutusta, mutta versiotiedotteiden perusteella kehitystä tehdään asiakasvaateiden mukaisesti. Parempi kommunikaatio asiakkaille, milloin tietyt heidän toivomansa vaatimukset kehitetään.
Asiakaskokemus on huono versiotoimitusten yhteydessä, koska laadun kanssa ongelmia.	Asiakaskyselyhaastattelut. Dokumenttianalyysi: Asiakasviestintä käyttäjäpäiviltä. Havainnointi.	Versioiden laadun sekä dokumentaation parantaminen. Yleisesti toimitusprosessissa ei ole ongelmia, vaan ongelmat ovat laadussa ja dokumentaatiossa.
Asiakas ja toimittajasuhde, toimittajan kehittäminen toimii hyvin	Asiakaskyselyhaastattelut. Havainnointi.	Yleisesti hyvä kommunikaatio, kun asiakkaille on nimetyt palvelupäälliköt. Versioinfotilaisuus parantaa toimittajan ja asiakkaan suhdetta entisestään.
Laadun kanssa ongelmia sovelluksessa.	Asiakaskyselyhaastattelut. Teemahaastattelut. Havainnointi. Dokumenttianalyysi: Asiakasviestintä käyttäjäpäiviltä, asiakaspalautteet, tehtävienhallintajärjestelmä ja Confluence versiosivut.	Arkkitehtuurinen muutos sovelluksen pilkkomiseksi, helpompi kokonaisuus hallita. Testivetoisen kehitysmallin ominaisuuksien käyttöönotto. Yksikkötestien kattavuuden läpikäynti ja järkevöittäminen. Automaatiotestien kattavuuden kasvattaminen. CI-prosessin käyttöönotto kokonaisuudessaan. CI/CD-prosessin käyttöönotto. Asiakas vielä aikaisemmassa vaiheessa mukaan testaukseen.

Versiotiedotteet eivät kata asiakastarpeita	Asiakaskyselyhaastattelut. Dokumenttianalyysi: Asiakasviestintä käyttäjäpäiviltä ja asiakaspalautteet.	Versiotiedotteisiin lisää tietoa. Nykyisten tietojen lisäksi kohdat: miten asia oli ennen, miten se on nyt, miksi muutos tehtiin, kuinka uutta ominaisuutta käytetään, rajoitteet ja tekniset muutokset.

Ensimmäinen tutkimuskysymys oli: Miten tehdä uusien sovellusversioiden hankinnasta sekä käyttöönotosta helpompaa asiakkaille? Tätä kysymystä varten käytiin läpi asiakkaiden toiveet sovelluskehitykselle ja -toimitukselle. Suurimmat ongelmat asiakkaiden mielestä olivat sovellusten laatu sekä dokumentaation puutteellisuus. Asiakkaat joutuvat käyttämään paljon aikaa sovellusversioiden testaukseen sekä odottamaan korjauksia, kun virheitä löytyy paljon. Uuden version ilmestyessä versiotiedote ei kerro asiakkaille tarpeeksi version sisällöstä ja siitä, mihin kaikkeen on tullut muutoksia. Organisaation pitää tehdä sovellukset laadukkaammin ja löytää virheet paljon aikaisemmassa vaiheessa, jotta ne voidaan korjata ennen versiojulkaisujen tekemistä. Tämä vähentää asiakkaiden testaustyömäärää ja parantaa asiakastyytyväisyyttä versiotoimituksen ja käyttöönoton yhteydessä. Kehitysehdotuksissa kerrotaan toimenpiteet, joita organisaation tulee tehdä parantaakseen versioiden laatua. Testaukseen tulisi lisätä kattavammat testitapaukset laajemmalta alueelta. Spillner ja Linz (2021, luvut 3.5-3.6) mainitsevat, testeissä tulee olla toiminnallisia ja ei-toiminnallisia testejä sekä löydettyjen virheiden ja uusien ominaisuuksien testauksen pitää myös olla mukana.

Versiotiedotteisiin pitää käyttää paljon enemmän aikaa, jotta ne kattavat asiakkaiden tarpeet. Bhatti (2021, luku 2) kirjoittaa, versiotiedotteet ovat kirjoitettu suoraan käyttäjille. Organisaation tulee siis huomioida, että käyttäjät eivät tällä hetkellä saa versiotiedotteista tarpeeksi tietoa, uudesta julkaistusta versiosta. Kehitysehdotuksissa kerrotaan, minkälaisia muutoksia versiotiedotteisiin pitää tehdä, että ne täyttävät asiakastarpeet ja helpottavat uuden version hankintaa sekä käyttöönottoa.

Toinen tutkimuskysymys oli: Miten versiotoimituksia voidaan tehostaa ja nykyistä tuotanto- ja toimitusprosessia parantaa? Tätä kysymystä varten käytiin

läpi organisaation sisäistä dokumentaatiota sekä haastateltiin organisaation henkilökuntaa. Laatu on yksi keskeinen asia molemmissa tutkimuskysymyksissä. Mitä laadukkaammin voidaan toteuttaa sovellus, sitä helpompi ja nopeampi toimitus- ja käyttöönottoprosessi on niin organisaatiolle, kuin asiakkaillekin. Kun tämä huomattiin, keskityttiin enemmän sovelluskehitykseen, koska versiotoimitusten laatu sisältyy läheisesti sovelluskehityksen prosesseihin. Versiotoimitusprosessi on toiminut laadukkaasti sen jälkeen, kun kaikki virheet on saatu korjattua. Tällöin suurin ongelma toimituksen laadussa on siis sovelluskehityksen puolella eli siinä, miten saadaan sovellusversiosta virheetön, jotta sen toimituksessa ei tarvitse tehdä suuria testauksia ja virheenkorjauksia.

Versiokehityksessä sovelluksen laatuun vaikuttaa vahvasti virheiden lukumäärä lopullisessa versiossa. Mikäli virheet jäävät kiinni mahdollisimman aikaisessa vaiheessa ja ne saadaan korjattua heti, niin silloin lopullinen versiojulkaisu on laadukas. Laadukas sovellusversio on myös helpompi ottaa käyttöön ja se madaltaa hankintapäätöstä. Tätä varten organisaation tulee kasvattaa testausten kattavuutta huomattavasti ja testiautomaation ajaminen pitäisi saada mukaan suoraan sovelluksen rakennuskanavaan. Testien ja testeihin liittyvän koodin määrä on huomattava ja tämän vuoksi sovelluksen arkkitehtuuria pitäisi muuttaa niin, että sovelluksesta voidaan rakentaa vain tarvittava pienin osa kerrallaan. Näin ohjelmistokehitykseen voitaisiin ottaa kokonaisuudessaan CI-prosessi käyttöön, jonka avulla virheet löytyisivät jo ohjelmoijan käsitellessä koodia eikä vasta silloin, kun versiotestauksia ajetaan öisin.

Organisaation tulisi myös harkita testivetoisen kehitysmallin käyttöönottoa ainakin osittain, koska silloin uusien ominaisuuksien ja korjausten testit tulisi jo valmiiksi ohjelmoitua, kun kehitystä tehdään. Näin myös nähdään, ettei tule liian suuria kehitettyjä ominaisuuksia kerralla, vaan voidaan toteuttaa ne sopivissa testattavissa osissa. (Parani, 2017, luku 1; Dima & Maassen, 2018, s. 318.) Näin saadaan poistettua uuskehityksestä mahdollisuus, ettei testejä olisi tarpeeksi ja kaikille tulevaisuudessa kehitettäville ominaisuuksille olisi jo testitapaukset valmiina.

11 POHDINTA JA JATKOTOIMET

Tässä luvussa käsitellään, mitkä olivat organisaation tärkeimmät kehitystarpeet sekä, miten löydettyjen tulosten perusteella voidaan parantaa organisaation toimintaa ja asiakastyytyväisyyttä. Seuraavaksi esitetään työn luotettavuuden ja eettisyyden arvioinnit. Tämän jälkeen käsitellään, kuinka tutkimuksen tulokset hyödyntävät koko alaa. Lopuksi kerrotaan organisaatiolle hyödyllisiä sekä yleisesti alaa koskevia jatkotutkimusaiheita.

11.1 Johtopäätökset ja pohdinta

Suurimmat ongelmat versiotoimitusten ja -käyttöönoton yhteydessä kohdistuivat sovellusversioiden laatuun ja dokumentaation puutteellisuuteen. Tärkeintä olisi saada testausta kattavammaksi niin, että virheet löytyisivät mahdollisimman aikaisessa vaiheessa. Dokumentaatio tulisi tehdä tarkemmaksi ja asiakkaiden kanssa pitää tehdä enemmän yhteistyötä esimerkiksi versioinfotilaisuuksien kautta. Näin asiakkaat voivat suoraan kysyä epäselvyyksistä versiotiedotteessa ja organisaatio saa myös mahdollisuuden kuulla asiakkaita heti versiojulkaisun yhteydessä.

Laadukas sovellustuotanto vähentäisi asiakkaiden tarvetta tehdä suurtestausta jokaisen version kohdalla, vähentäisi virheiden lukumäärää uudessa ja vanhassa koodissa ja tuottaisi lisää aikaa sovelluskehitykseen, kun ei tarvitse enää käyttää jatkuvasti paljon aikaa virhekorjauksiin. Mitä paremmin sovellusversio toimii asiakkaalla ja mitä vähemmän siihen pitää tehdä korjauspäivityksiä, sen parempia asiakaskokemuksia ovat versiotoimitus ja julkaisu. Tämän avulla saataisiin myös asiakastyytyväisyyttä kasvatettua.

11.2 Tutkimuksen luotettavuus ja eettisyys

Opinnäytetyön luotettavuutta arvioidessa pitää huomioida reflektiivinen toimintatapa tutkimusta tehdessä. Tutkijan pitää arvioida jatkuvasti tekemiään toimenpiteitä ja valintoja sekä niiden johdonmukaisuutta ja

tarkoituksenmukaisuutta kyseisen tutkimuksen tekemiseksi. Valinnat luotettavuuteen ovat nähtävissä myös esimerkiksi lähteiden valinnassa ja siinä onko tutkimuksessa käytetty lähdekriittistä lähestymistapaa. Myös tutkijan oma toiminta vaikuttaa tuloksiin ja päätelmiin, joten niiden arviointi on tärkeää, kun tarkastellaan tutkimuksen luotettavuutta. Esimerkiksi havainnointi on aina epätarkkaa, kun ei tehdä erillistä tallennusta esimerkiksi nauhoittamalla keskustelut. Havaintojen luotettavuuden arviointi jälkikäteen on haastavaa, koska havainnointitilanteeseen ei voi palata. Kokonaisuudessaan opinnäytetyöprosessin pitää olla luotettava ja tutkimuksessa ei pitäisi olla mukana sattumanvaraisia tuloksia tai sisäisiä ristiriitoja. (Vilkka, 2021, s. 132–135.)

Opinnäytetyössä käytettiin kattavasti erilaisia lähteitä ja ne ovat pääasiassa tuoreempia, kuin 10 vuotta. Lähteissä hyödynnettiin uusimman tutkimustiedon lisäksi vanhempaa tämän aihealueen peruskirjallisuutta. Niihin on myös koottu kattavasti erityyppistä kirjallisuutta kuten tutkimusartikkeleita, tietokirjallisuutta, ohjeistuksia, verkkosivuja ja blogeja. Tutkimuksessa löydettiin paljon tietoa sovellustoimitus- ja kehitysprosesseista ja molempiin tutkimuskysymyksiin löydettiin vastaukset. Eri tietolähteet tukivat toisiaan osoittaen samoja löytyneitä ongelmia ja kehitystarpeita. Tapaustutkimus soveltui hyvin tähän työhön, koska tutkittava ongelma oli tietty tapaus, mutta työn aikana huomattiin, että suurimmat ongelmat eivät olleet toimituksessa vaan laadussa vaikka tutkimusta aloitettaessa lähdettiin toimitusprosessin ongelmien tutkimisesta.

Asiakashaastatteluissa vastauksia saatiin kattavasti valituilta asiakkailta. Asiakkaiden vastauksista löytyi selkeä kaava asiakkaan suurimmista ongelmista versiohankinnassa. Sisäiset teemahaastattelut ja dokumenttianalyysi tukivat näitä löydöksiä. Asiantuntijahaastattelujen vastausprosentti oli alhainen ja se ei anna kokonaiskuvaa organisaation asiantuntijoiden toiminnasta ja haasteista. Havainnoinnin rooli kasvoi asiantuntijoiden tiedon keräyksessä ja asiantuntijahaastattelujen tuloksia ei sellaisenaan voitu hyödyntää tutkimuksessa. Tämä ei kuitenkaan lopulta tuottanut suurta ongelmaa tutkimukseen, koska teemahaastattelut ja dokumenttianalyysi antoivat tarpeeksi tietoa tutkimuksen suorittamiseksi.

Eettisyyden arviointi tutkimuksessa tehdään kirjoittamalla, kuinka tutkimuseettisiä periaatteita noudatettiin ei vain toteamalla, että niitä noudatettiin. Se pitää myös huomioida esimerkiksi aineiston valinnassa sekä, miten aineiston muodostamiseen ja laatuun on vaikutettu. Viittaukset teoriaan ja muihin lähteisiin pitää tehdä viittaussääntöjen mukaan ja merkitä aina mistä tekstiä on referoinut. Tutkimuksessa pitää olla rehellinen, tarkka ja huolellinen sekä huomioida tutkimuksiin osallistuvien henkilöiden informointi jo ennen tiedonkeruuta tai osallistumista. (Vilkka, 2021, s. 142–145; Ojasalo, 2015, s. 49.)

Tutkimusta tehtäessä noudatettiin hyviä tieteellisen tutkimuksen periaatteita. Tutkimusta aloitettaessa osapuolet allekirjoittivat opinnäytetyösopimuksen sekä tutkimuksen aiheen valinnassa tehtiin yhteistyötä osapuolien kesken. Organisaation toivetta pysyä anonyyminä opinnäytetyössä kunnioitetaan ja luottamuksellisuus on tärkeässä roolissa koko työssä. Haastatteluissa haastateltaville kerrottiin, mitä varten haastattelut tehdään ja haastatteluvastaukset käsiteltiin anonyymisti. Tietoperustaan viitattaessa käytettiin SAMK:n viittausohjeita ja tietoperusta kerättiin kattavasti niin maantieteellisesti, kuin julkaisun tason mukaisestikin.

11.3 Tulosten hyödynnettävyys alalla

Tutkimuksesta löydettiin paljon tietoa siitä, miten toimitusprosessi ja kehitysprosessi nivoutuvat tiiviisti yhteen käyttöönottossa, kun käsitellään sovelluksen laatua. Asiakkaat näkevät sovelluksen laadun osana toimitusta, vaikka käytännössä ne ovat eri asioita. Toimitusprosessi sisältää vain olemassa olevan version toimituksen asiakkaan käyttöön, kun taas sovelluksen laatu pitää sisällään ISO 25010 laatustandardin asiat, kuten luvussa 3.5 käytiin läpi.

Tuloksista löydettiin paljon kehitysehdotuksia, minkälaisia versiotiedotteiden tulee olla, jotta niissä on kaikki tarpeellinen tieto asiakkaille. Opinnäytetyön yhteydessä tehtiin myös mallipohja versiotiedotteiden sisältövaatimuksista (Liite 2), jotta ne kattavat sovelluksen käyttäjien tarpeet. Käyttäjät odottavat enemmän sovelluksia tuottavilta organisaatioilta, kuin vain sovelluskehitystä.

Käyttäjät toivovat saavansa enemmän tietoa, miten ja mitä on muokattu myös teknisellä tasolla, eikä vain käyttöliittymätasolla. Tämä johtuu siitä, että nykyään ohjelmistorobottien ja automaation määrä on kasvanut.

11.4 Jatkotutkimusaiheet

Sisäisesti organisaation pitäisi tehdä lisää tutkimusta siitä, miten CI/CD-prosessi saataisiin otettua käyttöön ja paljonko arkkitehtuurisia muutoksia se vaatisi. Tämän perusteella voi tehdä laskelman siitä, onko organisaatiolla oikeasti tarvetta lähteä toteuttamaan CI/CD-prosessia kokonaisuudessaan.

Organisaatiolle tutkija suosittelee vielä jatkotutkimusaiheeksi koko sovelluksen läpikäyntiä ISO2500 laatustandardin vaatimusten mukaan. Näin löydettäisiin tuotteesta kaikki kyseisen laatustandardin ongelmat, jotka voisivat aiheuttaa tyytymättömyyttä tai kehuja myös muualla sovelluksen käytössä, kuin vain toimituksissa.

Yksi tärkeä koko alaa koskeva tutkimusaihe on, kuinka voidaan muuttaa arkkitehtuurisesti monoliittiseksi järjestelmäksi tehty sovellus mikropalveluiksi. Mitä tämä muutos vaatii ja minkälaisia työvaiheita siihen sisältyy? Voisiko tätä varten tehdä mallin, jonka avulla organisaatiot voisivat toteuttaa kyseisen siirtymän?

Asiakkaat ja käyttäjät haluavat olla enemmän mukana sovelluskehityksessä määrittelemässä tulevaisuuden vaatimuksia sekä seuraamassa sovelluskehityksen tiekarttaa. Hyvä jatkotutkimusaihe tähän liittyen olisi selvittää kuinka syvälle asiakkaat haluaisivat lähteä sovelluskehitykseen, mikäli heille annettaisiin mahdollisuus tukea kehitystä aikaisemmassa vaiheessa. Toimittajan kehittämisessä voisi ajatella asiakkaiden haluavan testata tulevia ominaisuuksia jo kehitysympäristöissä eli ennen, kuin ne viedään erillisiin versioihin. Tällöin asiakkaat voisivat kertoa omat mielipiteensä ominaisuuden toiminnasta heti, kun on mahdollista ja toiminnallisuutta voitaisiin jalostaa pidemmälle jo ensimmäisessä kehitysversiossa.

LÄHTEET

Alanda, A., Mooduto, H. & Hadelina, R. (2022). Continous integration and continuous development (CI/CD) for web applications or cloud infrastructures. Journal of information technology and computer engineering 6(2) s. 50–55. doi: https://doi.org/10.25077/jitce.6.02.50-55.2022

Alnafessah, A., Gias, A., Wang, R., Zhu, L., Casale, G. & Filieri, A. (2021). Quality-Aware devops research: Where do we stand? DOI: 10.1109/ACCESS.2021.3064867

Atlassian. (n.d.). Devops. Haettu 29.10.2023 osoitteesta https://www.atlassian.com/devops

Appian. (n.d.) Process model. Haettu 25.11.2023 osoitteesta https://appian.com/process-mining/process-model.html.

Bergström, S. & Leppänen, A. (2021). Yrityksen asiakasmarkkinointi. Edita Publishing Oy.

Bhatti, J. (2021). Docs for developers: An engineer's field guide to technical writing. Apress.

Bojana, K. & Anastas, M. (2014). Change management and version control of scientific applications. arXiv.org. DOI 10.5121/ijcsit.2014.6211

Bruce, M. & Pereira, P. (2018). Microservices in action. Manning publications.

Bstieler, L. & Noble, C. (2023). The PDMA handbook of innovation and new product development, fourth edition. John Wiley & Sons.

Digi- ja väestötietovirasto. (2020). Turvallisen sovelluskehityksen käsikirja. https://www.suomidigi.fi/sites/default/files/2020-05/Turvallisen%20sovelluskehityksen%20käsikirja.pdf

Dima, A. & Maassen, M. (2018). From waterfall to agile software: Development models in the IT sector, 2006 to 2018. Impacts on company management. Journal of International Studies 11(2) s. 315–326. doi:10.14254/2071-8330.2018/11-2/21

Dora. (n.d.). Trunk-based development. Haettu 17.10.2023 osoitteesta https://dora.dev/devops-capabilities/technical/trunk-based-development/

Driessen, V. (5.1.2010). A successful git branching model. Nvie. https://nvie.com/posts/a-successful-git-branching-model/

Energiateollisuus. (2019). Kaasun jakeluverkon yleiset liittymisehdot. https://energia.fi/files/4425/Kaasun_jakeluverkon_yleiset_liittymisehdot_2019_su.pdf

Eriksson, P. & Koistinen, K. (2014). Monenlainen tapaustutkimus. Kuluttajatutkimuskeskus

Familiar, B. (2015). Microservices, IoT, and Azure: Leveraging devops and microservice architecture to deliver saas solutions. Apress.

Fingrid. (n.d.). Kuluttajatietoa. Haettu 29.10.2023 osoitteesta https://www.fingrid.fi/sivut/yhtio/kuluttajatietoa/?tag=9389&pageSize=5&page=1&language=fi

Franco, J., Lee, C., Vue, K., Bozonelos, D., Omae, M. & Cauchon, S. (2023). Introduction to political science research methods. 1st edition. ASCCC.

Gasum. (n.d.). Maakaasumarkkina Suomessa. Haettu 21.9.2023 osoitteesta https://www.gasum.com/kaasusta/maakaasu/maakaasumarkkina-suomessa/

Gupta, S. (28.9.2022). 5 saas growth strategies to scale your business quickly. https://www.gartner.com/en/digital-markets/insights/saas-growth-strategy

Goerdt, D. (8.8.2017). A perspective on devops. https://flexagon.com/blog/the-many-dimensions-of-devops/

Hartini, A. & Mahmoud, A. (2014). Customer satisfaction experiences in healthcare sector. UUM Press.

Hyvärinen, M., Nikander, P. & Ruusuvuori, J. (2017). Tutkimushaastattelun käsikirja. Kustannusosakeyhtiö Vastapaino. Doha Quatar. 10.1007/978-3-319-33111-9_30.hal-01377456

ISO2500. (n.d.). ISO25010 verkkosivut. Haettu 9.10.2023 osoitteesta https://iso25000.com/index.php/en/iso-25000-standards/iso-25010?ref=workingsoftware.dev

Keskinen, T. & Lipiäinen, J. (2013). Asiakkaan matkassa – Tuotekeskeisyydestä symbioosistrategiaan. Alma Talent.

Klotins, E., Gorschek, T., Sundelin, K. & Falk, E. (2022). Towards cost-benefit evaluation for continuous software engineering activities. Empirical software engineering. https://doi.org/10.1007/s10664-022-10191-w

Korkiakoski, K. (2023). Huomisen asiakas. Helsingin Seudun Kauppakamari Oy.

Kotimaisten kielten keskus. (n.d.-a). Sovellus. Haettu 19.11.2023 osoitteesta https://www.kielitoimistonsanakirja.fi/#/sovellus

Kotimaisten kielten keskus. (n.d.-b). Ohjelma. Haettu 19.11.2023 osoitteesta https://www.kielitoimistonsanakirja.fi/#/ohjelma

Kozma, D., Varga, P. & Larrinaga, F. (2021). System of systems lifecycle management – New concept based on process engineering methodologies. MDPI. https://doi.org/10.3390/app11083386

Krishna, S., Sreekanth, S., Perumal, K. & Reddy, K. (2012). Explore 10 different types of software development process models. IJCSIT s. 4580–4584.

Laine, H. & Paakki, J. (n.d.). Ohjelmistotuotanto ohjelmistoprosessi, ohjelmiston elinkaari. Haettu 25.11.2023 osoitteesta https://www.cs.helsinki.fi/u/paakki/ohtuk03-luento2.pdf.

Laporte, C. & April, A. (2018). Software quality assurance. IEEE Press.

Limón, B. (6.7.2022). Compiled vs interpreted language: Basic for beginning devs. https://www.educative.io/blog/compiled-vs-interpreted-language

Linz, T. (2014). Testing in scrum: A guide for software quality assurance in the agile world. Rocky Nook.

Lwakatare, L., Kuvaja, P. & Oivo, M. (2016). An explortary study of devops extending the dimensions of devops with practices. The Eleventh International Conference on Software Engineering Advances (s. 91–99). ICSEA 2016.

Mahut, F., Bricogne, M., Daaboul, J. & Eynard, B. (2015). Servitization of product lifecycle management: Towards service lifecycle management.

Manandhar, G. (8.7.2021). Use feature flags to release code safely in any git branching strategy. Flagsmith. https://www.flagsmith.com/blog/git-branching-and-feature-flags

McMillian, T. (4.12.2023). How to improve automation test coverage. Testrail blog. https://www.testrail.com/blog/how-to-improve-automation-test-coverage/

Merode, H. (2023). Continuous integration (CI) and continuous delivery (CD): A practical guide to designing and developing pipelines. Apres.

Microsoft. (21.10.2022). Adopt a git branching strategy. Azure Devops. https://learn.microsoft.com/en-us/azure/devops/repos/git/git-branching-guidance?view=azure-devops

Nicolette, D. (2015). Software development metrics. Manning Publications.

Nielsen, C., Larsen, P., Fitzgerald, J., Woodcock, J. & Peleska, J. (2015). System of systems engineering: Basic concepts, model-based techniques and research directions. ACM Computer Surveys. http://dx.doi.org/10.1145/2794381

Nielsen, S. (21.6.2022). Mitä eroa on sovelluksella ja ohjelmalla? Kotimikro. https://kotimikro.fi/ohjelmat/sovellukset/mita-eroa-on-sovelluksella-ja-ohjelmalla

Nieminen, S. (2016). Hyvä hankinta – parempi bisnes. Alma Talent Oy.

Ojasalo, K., Moilanen, T. & Ritalahti, J. (2015). Kehittämistyön menetelmät. Uuden- laista osaamista liiketoimintaan (3.–6. painos). Sanoma Pro.

Paranj, B. (2017). Test driven development in Ruby: A practical introduction to TDD using problem and domain analysis. Apress.

Puusa, A. & Juuti, P. (2020). Laadullisen tutkimuksen näkökulmat ja menetelmät. Gaudeamus.

Pöyhönen, P., Santavuori, H. & Mustonen, S. (2023). Asiakastutkimus perusteet ja käytännöt. Alma Talent Oy.

Raghavan, P. (2021). James Smith on software bugs and quality. IEEE Software.

Red Hat. (9.3.2020). What is an application architecture? Red Hat. https://www.redhat.com/en/topics/cloud-native-apps/what-is-an-application-architecture

Red Hat. (12.12.2023). What is CI/CD? Red Hat. https://www.redhat.com/en/topics/devops/what-is-ci-cd

Riccomini, C. & Ryaboy, D. (2021). The missing readme: A guide for the new software engineer. No Starch Press.

Rostoniec, A. (27.4.2023). Modular software architecture: Advantages and disadvantages of using monolith, microservices and modular monolith. Pretius blog. https://pretius.com/blog/modular-software-architecture/

Rubin, A. (2021). Rocking qualitative social science: An irrevent guide to rigorous research. Stanford University Press.

Saarijärvi, H. & Puustinen, P. (2020). Strategiana asiakaskokemus. Miksi, mitä, miten? Docendo.

Salameh, H. (2019). The impact of devops automation, controls, and visibility practices on software continuous deployment and delivery. Proceedings of the 2nd International Conference on Research in Management and Economics (s. 22–46). IMECONF.

Sanastokeskus. (29.10.2020). Sovellus. Sanastokeskus ry. https://sanastokeskus.fi/tsk/fi/termitalkoot/haku-266.html?page=get_id&id=ID387&vocabulary_code=TSKTT

Sanastokeskus. (n.d.). Tietotekniikan termitalkoot ohjelmat. Sanastokeskus ry. https://sanastokeskus.fi/tsk/fi/termitalkoot/haku-266.html?page=resurssi&tiedosto=ohjelmat.svg

Schmitt, J. (14.4.2023). Trunk-based vs feature-based development. https://circleci.com/blog/trunk-vs-feature-based-dev/

Smartbear. (n.d.). What is unit testing?. Haettu 22.1.2024 osoitteesta https://smartbear.com/learn/automated-testing/what-is-unit-testing/

Sombultawee, K. & Pasunon, P. (2022). Long-term buyer-supplier relationships in IT services. Journal of business & Industrial marketing 37(3) 629–642. DOI 10.1108/JBIM-06-2020-0292

Suvanto, A. (28.7.2022). A brief introduction to product management. https://www.eficode.com/blog/a-brief-introduction-to-product-management

Spillner, A. & Linz, T. (2021). Software testing foundation, 5th edition: A study guide for certified tester exam. Rocky Nook.

Stark, J. (2011). Product Lifecycle Management – 21st Century paradigm for product realization, second edition. Springer.

Sharda, R., Chaudhary, M., Pillai, R. & Gururao, S. (2014). Patterns of Multi-Tenant SaaS Applications. EMC

Shwaber, K. & Sutherland, J. (2020). The scrum guide – The definitive guide to scrum: The rules of the game. Haettu 25.10.2023 osoitteesta https://scrum-guides.org/docs/scrumguide/v2020/2020-Scrum-Guide-US.pdf#zoom=100 ja https://scrumguides.org/scrum-guide.html

Topdevs. (25.11.2021). All you need to know about feature-driven development. Topdevs Blog. https://topdevs.org/blog/feature-driven-development-gitflow

Vasserman, M. (10.8.2023). Why do you need release notes? (featuring great examples and templates). Canny blog. https://canny.io/blog/release-notes/

Valtori. (n.d.). Valpinen valtorin maailma tutuksi sanaston avulla. Haettu 16.10.2023 osoitteesta https://valtori.fi/valpinen-sanasto

Vilkka, H. (2021). Näin onnistut opinnäytetyössä ratkaisut tutkimuksen umpikujiin. PS-kustannus.

Vuorenmaa, M. (n.d.). Lämmitysmarkkinoilla vapaus valita. Haettu 21.9.2023 osoitteesta https://energia.fi/energiasta/energiamarkkinat/lammitysmarkkinat

Wagner, S. (2013). Software Product Quality Control. Springer.

Westfall, L. (2009). Certified software quality engineer handbook. ASQ Quality Press.

Wilson, C. (2022). Grokking continous delivery. Manning Publications.
lyhenneluettelo.

Yarlagadda, R. (2021). Devops and its practices. International journal of creative research thoughts, 9(3). http://www.ijcrt.org/papers/IJCRT2103016.pdf

LIITE 1: HAASTATTELUKYSYMYKSET JA TEEMAT

Teemahaastattelun teemat

1. Minkälainen prosessi versiotoimitus/versiokehitys on?
2. Prosessien soveltuvuus organisaation käyttöön.
3. Kommunikointi tiimien ja asiakkaiden välillä.
4. Asiakasnäkökulma versiokehityksessä ja -toimituksessa.
5. Asiakkaan huomioiminen versiotoimitus ja -kehitysprosesseissa.
6. Suurimmat ongelmat versiotoimituksessa ja -kehityksessä sekä kuinka niitä ratkaistaan?

Haastattelukysymykset asiakkaille

1. Minkälainen prosessi versiotoimitus on teidän näkökulmastanne?
2. Onko kaikissa ottamissanne versioissa tullut teille hyödyllisiä ominaisuuksia?
3. Paljonko työaikaa teiltä kuluu versiopäivitysten yhteydessä, kun otetaan huomioon aika tilauksesta tuotantokäyttöön? Mihin aika kuluu?
4. Mikä erityisesti saa teidät ottamaan uuden version?
5. Mikä erityisesti tekee sen, ettei uutta julkaistua versiota oteta käyttöön?
6. Oletteko saaneet tarpeeksi tukea toimitusprosessin aikana ja sen jälkeen? Jos ette ole, minkälaista tukea olisitte tarvinneet?
7. Kuinka aikaisessa vaiheessa version hankintapäätös tehdään teidän yrityksessänne?
8. Onko teillä tarpeeksi aikaa tehdä päätös version hankinnasta versiotiedotteen julkaisusta alkaen?
9. Pystyttekö vaikuttamaan versiopäivityksen sisältöön tai siihen, kuinka toimitus tehdään?
10. Jotain muuta, mitä haluatte kertoa?

Haastattelukysymykset organisaation henkilöstölle

1. Toimitko versiokehityksen vai -toimituksen puolella?
2. Minkälaisia prosesseja versiotoimitus ja -kehitys ovat?

3. Kuinka kommunikoitte tai jaatte tietoa eri prosessin vaiheista eri tiimien ja johdon välillä?
4. Kuinka ja minne dokumentoitte tehdyt toimenpiteet, uudet ominaisuudet ja virheenkorjaukset?
5. Onko eri asiakkaat huomioitu eri tavalla toimituksissa tai versiokehityksessä? Miten?
6. Mikä versiotoimituksessa ja -kehityksessä toimii?
7. Mitä kehitettävää on versiotoimituksessa ja -kehityksessä?
8. Jotain muuta, mitä haluatte kertoa?

LIITE 2: VERSIOTIEDOTTEIDEN SISÄLTÖVAATIMUKSET

Release note
SystemABC
Version
2.1

FEATURES

Feature	Ticket	Business	Summary	How it was	How it is now	Why	How to use	Limitations	Technical Change	Optional
Inspect invoices in software invoice search with preview.	AB-1234	Sales, Grid, Gas	When clicking a invoice with right mouse button, you can see a preview of the invoice opened in another view. Then you can close the invoice preview and return to the search window.	You needed to open the invoice separately and it opened to a new view. With a button at the end of the invoice row. This has not been changed.	You can now preview the printed invoice just by pressing right mouse button over the invoice.	It is easier to preview invoice quickly and you do not lose the search results.	Right mouse click ower the invoice line in the invoice search view of the software.	Invoice preview view do not allow any changes or printing the invoice. You can only see a preview of the printed invoice.	A new view can be opened from invoice search window. No other changes.	x

BUGS

Bug	Ticket	Business	Summary	How it was	How it is now	Why	Limitations	Technical change
Invoice search didn't show all invoice lines.	CB-1224	Sales, Grid, Gas	When searching invoices search didn't find invoices that was not sent to customer yet.	Search didn't show invoice drafts that was not sent to customer yet.	Now search shows all invoices.	Users couldn't see the real situation of invoices with invoice search when part of the invoices was missing.	Invoice preview view do not show anything on draft invoices.	A change in the SQL query created by the invoice search.