Joonas Ruuth

Jatkuvan ohjelmistokehityksen ominaispiirteet SaaS-ohjelmiston kehitystyössä

Tietotekniikan kandidaatintutkielma

23. huhtikuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijä: Joonas Ruuth

Yhteystiedot: joonas.ruuth@student.jyu.fi

Ohjaaja: Antti-Jussi Lakanen

Työn nimi: Jatkuvan ohjelmistokehityksen ominaispiirteet SaaS-ohjelmiston kehitystyössä

Title in English: Characteristics of continuous software development in SaaS software development

Työ: Kandidaatintutkielma

Opintosuunta: Kaikki opintosuunnat

Sivumäärä: 25+0

Tiivistelmä:

Tässä tutkielmassa selvitetään, mitä käytänteitä kuuluu SaaS-ohjelmiston jatkuvaan ohjelmistokehitykseen perustuen alan kirjallisuuteen. Tutkielman tavoitteena tuoda esille hyväksi koettuja kehityskäytänteitä SaaS-ohjelmiston kehityksen tueksi. Lopputulemana lukija ymmärtää, minkälaisia asioita jatkuvassa SaaS-ohjelmiston kehityksessä suositellaan otettavan huomioon, jotta vältyttäisiin riskien tuomilta ongelmilta.

Avainsanat: Jatkuva ohjelmistokehitys, SaaS-ohjelmisto, jatkuva testaaminen, jatkuva monitorointi

Abstract: This study takes a look which practices belong to the continuous development of SaaS software, based on the literature of the field. The aim of the study is to highlight proven development practices to support the development of SaaS software. As a result, the reader understands what kind of things are recommended to be taken into account in the continuous development of SaaS software, in order to avoid problems brought by risks.

Keywords: Continuous development, SaaS software, continuous testing, continuous monitoring

Kuviot

Sisällys

1 Johdanto

SaaS-ohjelmistolla (lyhenne sanoista "Software as a service") tarkoitetaan ohjelmistoa, jota tarjotaan asiakkaalle internetin yli web-sovelluksena. SaaS-ohjelmiston ylläpito ja säilytys palvelimilla on ohjelmiston tarjoajan vastuulla. SaaS-ohjelmistot eivät vaadi käyttöä varten raskasta IT-infrastruktuuria, vaan loppukäyttäjälle riittää usein tietokone ja internet-yhteys, jolloin itse tuotetta käytetään web-sovelluksena.

SaaS-ohjelmistojen suosion kasvaessa ja niitä tarjoavien organisaatioiden kilpailun kasvaessa, on ohjelmistokehitykseen kiinnitettävä huomiota siten, että SaaS-ohjelmiston laatu ei kärsi. Olemassa olevan ohjelmiston edelleen kehittäminen vaatii huomiota erityisesti niissä tilanteissa, kun ohjelmisto on julkaistavien päivitysten välillä jatkuvasti käytössä, jolloin jatkuvan kehitystyön on oltava turvallista ja joustavaa.

SaaS-ohjelmistojen kehitystyö ei ole Akinrolabu, New ja Martin (2019) mukaan riskitöntä. Kehitystyön riskeinä on Akinrolabu, New ja Martin (2019) tutkimuksessa esitetyt skenaariot, joihin lukeutuu esimerkiksi tietovuodot sekä niistä aiheutuneet käyttökatkot. Akinrolabu, New ja Martin (2019) osoittaa lisäksi tutkimuksessaan, että nämä aiemmin mainitut skenaariot voivat tulla SaaS-palveluntarjoajalle taloudellisesti hyvin kalliiksi, puhumatta asiakkaan luottamuksesta palveluntarjoajaan. Näitä riskejä on mahdollista välttää tutkielmassa esitettävien työkalujen avulla.

Tässä tutkielmassa tutkitaan sitä, mitä asioita ja menetelmiä suositellaan otettavan huomioon SaaS-ohjelmistojen jatkuvassa ohjelmistokehityksessä. Tämä tutkielma toteutettiin kirjallisuuskatsauksena. Tavoitteena on kartoittaa SaaS-ohjelmiston jatkuvaan ohjelmistokehitykseen liittyviä hyväksi koettuja käytänteitä ja menetelmiä, joita suositellaan käytettävän tai otettavan huomioon kehitystyötä tehdessä. Näihin menetelmiin kuuluu muun muassa koodikatselmoinnit, automaattinen testaaminen, riskien analysointi ja arviointi sekä järjestelmän jatkuva monitorointi. Tutkielman lähestymistapaa jatkuvaan ohjelmistokehitykseen on rajattu siten, että tässä tutkielmassa tutkitaan jatkuvaa ohjelmistokehitystä erityisesti ohjelmistokehittäjän näkökulmasta.

Tämä tutkielma on jaoiteltu niin, että luvussa 2 esitellään, mikä SaaS-ohjelmisto on ja mitkä

ovat sen ominaispiirteet. Luvussa 3 käydään läpi mitä jatkuva ohjelmistotuotanto on SaaS-ohjelmiston kontekstissa. Luku 4 on yhteenveto, jossa käydään läpi johtopäätökset.

2 SaaS-ohjelmisto

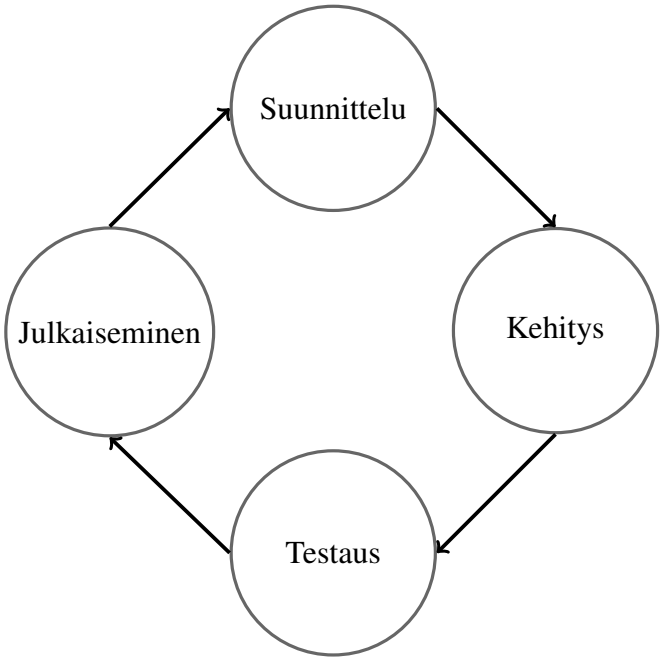
2.1 Mikä on SaaS-ohjelmisto?

SaaS-ohjelmisto on pilvipalvelu ja jakelumalli, jossa käytettävä ohjelmisto on loppukäyttäjän saatavilla internetin yli. SaaS-ohjelmiston tarjoaja huolehtii ohjelmiston korjauksista ja päivityksistä (Bordak 2019). SaaS-ohjelmistot ovat täten helpottaneet monen yrityksen tapaa harjoittaa liiketoimintaa. Singh ym. (2016) kertoo SaaS-ohjelmiston hyötyjä olevan muun muassa ohjelmiston kustannustehokkuus. SaaS-ohjelmisto on pilvipohjaisuutensa ansiosta nopeasti asiakasyrityksen käytettävissä, jolloin loppukäyttäjälle riittää usein ohjelmiston käyttöä varten tietokone tai älypuhelin sekä tunnukset järjestelmään.

SaaS-palveluissa on tavallista säännölliset päivitykset, joita palveluntarjoaja julkaisee aika ajoin, yleensä etukäteen ilmoitettuna ajankohtana. Päivitykset voivat asentua itsestään, jolloin ohjelmistoissa saattaa olla pieni käyttökatko tai päivitys voi vaihtoehtoisesti olla käyttäjän itse asentama.

SaaS-ohjelmistokehityksessä säännölliset päivitykset tarkoittavat, että ohjelmiston kehitys on jatkuvaa. Kuviossa 1 on esitetty jatkuvan SaaS-ohjelmiston kehityksen elinkaari, jossa kaikki alkaa päivityksen ja siihen sisältyvien ominaisuuksien suunnittelulla. Suunnittelun jälkeen alkaa varsinainen kehitystyö, jota seuraa uusien ominaisuuksien testaaminen. Viimeisenä on etukäteen ilmoitettuna ajankohtana tapahtuva päivityksen julkaiseminen. (Patel ja Chouhan 2016)

SaaS-ohjelmistojen heikkous on Singh ym. (2016) mukaan SaaS-ohjelmiston arkkitehtuurin muokattavuuden puute. Kumar (2014) käsittelee artikkelissaan myös SaaS-ohjelmiston heikkouksia, joista hän nosti seuraavia asioita. SaaS-ohjelmiston ollessa saatavilla web-sovelluksena, tulee sen käyttäjällä olla luotettava ja vakaa internet-yhteys. Yhteyden ollessa huono tai sitä ei ole ollenkaan, ei sovellusta pääse käyttämään ollenkaan. Toiseksi heikkoudeksi Kumar (2014) mainitsi tietoturvan ja siihen liittyvät ongelmat.



Kuvio 1. Jatkuvan SaaS-ohjelmiston kehityksen elinkaari (Patel ja Chouhan 2016)

2.2 SaaS-ohjelmiston ominaispiirteet

SaaS-ohjelmistot ovat usein kuukausimaksullisia palveluja, joista kustannukset syntyy loppukäyttäjälle esimerkiksi siitä, kuinka monta käyttäjää palvelulla on kuukausittain (Gibson ym. 2012) tai vaihtoehtoisesti siitä, kuinka paljon esimerkiksi dataa asiakas käyttää tai liikuttaa käytettävän palvelun sisällä. Käyttäjällä on usein lisenssi, jonka kautta hänellä on pääsy palveluun, joka sijaitsee pilvessä, eikä esimerkiksi käyttäjän omalla laitteella (Gibson ym. 2012). Myös ohjelmiston käyttäjän data on tyypillisesti säilössä muualla kuin käyttäjän käyttämällä laitteella. SaaS-palvelun käyttämä data ja sen tietokanta myös säilötty pilveen, jolloin käyttäjän ei tarvitse kantaa Gibson ym. (2012) mukaan arkaluontoista tietoa mukanaan esimerkiksi matkustaessaan.

SaaS-ohjelmistolle on tyypillistä myös, että palvelun käyttö perustuu moniasiakkaisuuteen. Odun-Ayo ym. (2017) mukaan moniasiakkaisuus pilvipalveluissa on sitä, että eri asiakkaat voivat käyttää samaa sovellusta, mutta jokaisella olisi kuitenkin käytössään oma ilmentymä itse sovelluksesta. Moniasiakkaisuuden muita ominaispiirteitä on Odun-Ayo ym. (2017) mukaan asiakkaan mahdollisuus oman sovellusilmentymän konfigurointiin, tarkoittaen tässä tapauksessa sovelluksen ulkoasun muuttamista tai käytettävien tietomallien arkkitehtuuria.

Kumar (2014) artikkelissa nostetaan esille samanlaisia aiemmin mainittuja SaaS-ohjelmiston ominaispiirteitä. Kumar (2014) esittää ominaispiirteet enemmänkin niin, että niiden tulee toteutua kehitetyn SaaS-ohjelmiston kohdalla, esimerkiksi: Sovelluksen tulisi olla käytettävissä käyttäjän tietokoneella sekä älypuhelimella. Kumar (2014) nosti lisäksi esille sen, että SaaS-ohjelmiston tulisi olla helposti loppykäyttäjän käytettävissä ilman palveluntarjoajan vuorovaikutusta.

3 Jatkuva ohjelmistotuotanto SaaS-ohjelmistokehityksessä

3.1 Jatkuvan ohjelmistokehityksen ominaispiirteet

Uzunbayir ja Kurtel (2018) mukaan jatkuva ohjelmistokehityksen päätavoitteena on kehittää ohjelmistoa, julkaista ohjelmiston päivityksen nopeasti tuotantoympäristöön ja tämän jälkeen jatkojalostaa ohjelmistoa asiakkailta saadun palautteen muodossa.

O'Connor, Elger ja Clarke (2017) mukaan jatkuvassa ohjelmistokehityksessä on teknologioiden kehittymisen myötä kyse tuotantoonvientien tehostamisesta automaatioon perustuvien työkalujen avulla. Fitzgerald ja Stol (2014) näkemys on, että jatkuva ohjelmistotuotanto on osa niin sanottua "Continuous * -kokonaisuutta. Tässä kokonaisuudessa jatkuva ohjelmistotuotanto nähdään osana jatkumoa johon kuuluu liiketoiminnan suunnittelu, ohjelmistojen kehittäminen sekä ohjelmistojen käyttäminen (Fitzgerald ja Stol 2014). Tässä tutkielmassa keskitytään erityisesti ohjelmistokehityksen näkökulmasta lähestyttäviin toimintoihin sovellettuna SaaS-ohjelmistokehitykseen. Fitzgerald ja Stol (2014) tutkimuksessa ohjelmistojen kehittämiseen keskittyvä osa-alue sisältää muun muassa seuraavat toiminnot.

- Jatkuva testaaminen (engl. Continuous Testing)
- Jatkuva tuotantoonvienti (engl. Continuous Deployment)
- Jatkuva tietoturva (engl. Continuous Security)

Fitzgerald ja Stol (2014) tutkimuksesta liiketoiminnan suunnittelu sisälsi lisäksi myös samanlaisia elementtejä mitä kuvio 1 sisältää jatkuvan ohjelmistokehityksen elinkaaren osalta.

3.2 Mitä riskejä SaaS-ohjelmistokehitys sisältää?

Pilvipalvelut yleisesti, joihin SaaS-ohjelmistot myös kuuluu ovat Peake (2012) mukaan enemmän alttiita virheille, epäonnistumisille sekä ulkopuolisille hyökkäyksille kuin mitkä tahansa muutkin ohjelmistot ja palvelut. Pilvipalvelun ja tässä tapauksessa SaaS-ohjelmiston tapauksessa pienikin virheestä tai hyökkäyksestä johtuva käyttökatko voi vaikuttaa moniasiakkai-

suuden takia useaan eri asiakkaaseen. Tämän takia oikeiden työkalujen käyttö laadukkaan lähdekoodin tuottamiseksi on välttämätöntä SaaS-ohjelmiston jatkuvassa ohjelmistokehityksessä. SaaS-palveluntarjoajalle pienetkin käyttökatkot tai tietomurrot voivat tulla hyvin kalliiksi ja sen lisäksi rikkoa luottamusta käytettävään ohjelmistoon. Akinrolabu, New ja Martin (2019) tutkimuksessa käsitellään tunnistettuja riskejä ja niiden vaikutuksia palveluntarjoajaan muun muassa taloudellisesti. Esimerkiksi SaaS-ohjelmistoon kohdistuva tietomurto, jossa hyökkääjä pääsee käsiksi asiakastietodataan voi aiheuttaa jopa 375 miljoonan punnan kustannukset SaaS-palveluntarjoajalle (Akinrolabu, New ja Martin 2019).

Peake (2012) mukaan SaaS-ohjelmistotarjoajan vastuulla on hyvin paljon asioita, joissa asiakkaan tulisi uskaltaa vain luottaa siihen, että kaikki on tietoturvallista. Turvallisuus pitää Peake (2012) mukaan sisällään luottamus ja suorituskyvylliset näkökulmat, joihin eri organisaatioilta on noussut esiin seuraavanlaisia huolia. Yksityisyyteen liittyvät ongelmat, ohjelmiston avoin käyttö, luotettavuus, yhteentoimivuus eri laitteiden välillä, riippuvuus palveluntarjoajasta ja ohjelmistosta saatava taloudellinen hyöty.

Peake (2012) artikkelissa mainitut käyttökatkot muodostavat omalta osaltaan riskin SaaS-ohjelmistojen kehitystyössä. Käyttökatkoja on kuitenkin erilaisia, joita ovat Akinrolabu, New ja Martin (2019) tekemän tutkimuksen mukaan esimerkiksi:

- Palvelunestohyökkäyksistä johtuvat käyttökatkot
- Ohjelmointivirheestä johtuvat käyttökatkot

Peake (2012) mukaan on myös olemassa palvelunestohyökkäyksen muoto, joka muodostaa riskin erityisesti SaaS-ohjelmistoa käyttävälle organisaatiolle, jossa vahingot olisivat taloudelliset. Peake (2012) esittelemässä palvelunestohyökkäyksessä hyökkääjä pyrkisi nostamaan tietyn asiakkaan käyttämän SaaS-ohjelmistoinstanssin palvelukuluja painottamalla haitallista liikennettä juuri tähän yhteen instanssiin. Tässä riskissä hyökkääjä käyttää hyväkseen pilvipalvelun toiminnallisuuksia sekä hinnoittelumallia.

Käyttökatkojen ja palvelunestohyökkäyksien lisäksi SaaS-ohjelmiston käytössä oleva data on myös riskien osalta keskiössä. Achar (2016) mukaan data on merkittävä tekijä SaaS-ohjelmistossa, joka luo omalta osaltaan erilaisia turvallisuusuhkia ja riskejä jatkuvan ohjelmistokehityksen ympärille. Achar (2016) mukaan tietoturvan puute ja tietovuodot ovat

suurimpia riskejä. Akinrolabu, New ja Martin (2019) mukaan tietoturvan puute sekä tietovuodot aiheuttivat muun muassa arkaluontoisen tiedon varastamista tai leviämistä muualle verkkoon, sekä sen kautta palvelun käyttökatkoja, kun järjestelmä on jouduttu sulkemaan ja korjaamaan. Nämä aiemmin mainitut ovat yleisimmät riskit pilvipalveluiden kohdalla, jotka tulisi ottaa vakavasti pilvipalveluiden suosion kasvaessa (Akinrolabu, New ja Martin 2019).

Gibson ym. (2012) mukaan SaaS-ohjelmiston keskitetyssä tietokannassa säilötty data voi luoda riskejä muun muassa seuraavien mainittujen asioiden kautta. Ensimmäinen Gibson ym. (2012) mainitsema riski on, että kenellä on pääsy asiakkaan dataan tietokannassa, joka voi olla arkaluontoista. Toinen riski oli, että onko data salattu eli kryptattu. Kolmas huomionarvoinen riski Gibson ym. (2012) artikkelissa oli se, että vaikka ohjelmisto on käytettävissä joustavasti web-sovelluksena, on varmistettava, että sivustojen rajapinnat ja päätepisteet ovat turvallisia. SaaS-ohjelmistojen rajapintojen tietoturvaa käsitellään myös Sharma ym. (2017) artikkelissa. Sharma ym. (2017) mukaan rajapintojen toteuttaminen tietoturvallisesti on olennainen osa pilvipalveluiden kokonaisvaltaista turvallisuutta.

Belbergui, Elkamoun ja Hilal (2017) artikkelissa nostetaan esille samoja jo aiemmin mainittuja riskejä SaaS-ohjelmiston käsittelevän datan suhteen. Belbergui, Elkamoun ja Hilal (2017) nostavat sen lisäksi myös esille datan eheyden säilymisen riskin. Datan eheydellä ohjelmistokehityksessä tarkoitetaan sitä, että data pysyy muuttumattomana ja oikeana riippumatta siitä, miten sitä käsitellään ohjelmiston ja tietokannan välillä.

SaaS-ohjelmistojen tietomurtojen yleisin syy on Sharma ym. (2017) mukaan vääriin käsiin joutuneet käyttäjätunnustiedot sekä se, että kirjautumisen vahvistamiseksi ei ole otettu käyttöön monivaiheista tunnistautumista.

3.3 Riskianalyysit ja riskien arviointi

Perinteiset riskianalyysit ja arviot eivät Kunz, Schneider ja Banse (2022) mukaan ole kovin hyviä pilvipalveluiden tai SaaS-ohjelmistojen riskienhallintakeinoja. Kunz, Schneider ja Banse (2022) määrittelemät ratkaisut painottavat riskienhallinnan automatisointia ja jatkuvuutta, mitä SaaS-ohjelmiston riskienhallinnassa tarvitaan.

Dash ja Dash (2010) mukaan riskien arviointi on ohjelmistokehityksessä osa riskien hallinnointia, jossa on positiivisia ja negatiivisia vaikutuksia. Artikkelin mukaan tehokkaalla päätöksenteolla sekä tarkasti määritelyllä riskien arvioinnilla ja analysoinnilla pystytään ennakoimaan lähes kaikki positiiviset ja negatiiviset lopputulemat. Dash ja Dash (2010) nostavat esille Kunz, Schneider ja Banse (2022) tapaan myös riskienhallinnan automatisoinnin tärkeyden, mutta lisäksi myös niihin liittyviä haasteita. Dash ja Dash (2010) mukaan automatisoitujen riskianalyysi työkalujen haasteena on niiden avulla kerättävän tiedon koostaminen merkitykselliseksi, jotta olisi käyttöä jatkuvassa riskianalyysissa.

Rohmeyer ja Ben-Zvi (2015) kuvailevat artikkelissaan miten erilaisia riskianalyyseja voidaan hyödyntää pilvipalvelun kehityksen työkaluna. Rohmeyer ja Ben-Zvi (2015) esittävät kolme riskianalyysi metodia, joita yhdessä hyödynnettynä pystytään vastaamaan hyvin pilvipalveluiden kehityksen luomaan paineeseen riskienhallinnan osalta. Esitellyt metodit ovat sovellettavissa ohjelmistokehittäjän näkökulmaan. Artikkelissa esitetyt metodit olivat:

- Perustavanlaatuinen riskianalyysi (engl. Fundamental Risk Analysis)
- Tietovirta-analyysi (engl. Data Flow Analysis)
- Skenaarioanalyysi (engl. Scenario Analysis)

Perustavanlaatuisessa riskianalyysissa pyritään tunnistamaan riskit ja havainnollistamaan niiden avulla mahdolliset uhat ja haavoittuvuudet sekä muodostamaan perusta todennäköisyyksien määrittämiselle (Rohmeyer ja Ben-Zvi 2015)

Tietovirta-analyysissa keskitytään Rohmeyer ja Ben-Zvi (2015) artikkelin mukaan järjestelmän tai ohjelmiston käsittelemään dataan ja erityisesti siihen kuinka dataa käsitellään ja säilötään ohjelmiston käyttämässä tietokannassa. Esimerkiksi Belbergui, Elkamoun ja Hilal (2017) artikkelissa esiin nostettu datan eheyden säilyttäminen sopisi tämän analyysin tarkasteltavaksi.

Rohmeyer ja Ben-Zvi (2015) artikkelissa kuvataan vielä kolmas riskianalyysin muoto, joka on skenaarioanalyysi. Skenaarioanalyysi perustuu siihen, että pyritään mallintamaan oikea tai kuvitteellinen riski eräänlaiseksi skenaarioksi, jota analysoimalla pystytään laskelmoimaan mahdollisesti aiheutuvat vahingot (Rohmeyer ja Ben-Zvi 2015). Ensisijaisena tavoitteena on kuitenkin Rohmeyer ja Ben-Zvi (2015) skenaarioanalyysin avulla oppia välttämään

mahdollisesti aiheutuvat riskit.

Näitä kolmea menetelmää käytettäessä yhdessä, saadaan aikaan Rohmeyer ja Ben-Zvi (2015) mukaan vahva riskianalyysimalli, jonka avulla mahdollistetaan erilaisten riskien tunnistaminen eri näkökulmista. Rohmeyer ja Ben-Zvi (2015) esittämä malli ei myöskään rajoitu vain tekniseen näkökulmaan, vaan on myös sovellettavissa muunkin tyyppisiin riskeihin SaaS-ohjelmiston kehityksessä.

Saripalli ja Walters (2010) käsittelee artikkelissaan myös riskianalyysien käyttöä pilvipalveluiden kehitystyökaluna. Saripalli ja Walters (2010) artikkelista esittelen tässä tutkielmassa kaksi riskianalyysia, joita voidaan hyödyntää myös SaaS-ohjelmiston jatkuvassa kehitystyössä.

Ensimmäinen esiteltävä analyysi on uhkien mallintaminen (eng. Threat Modeling). Uhkien mallintamisessa olennaista on jo tiedossa olevien tietoturvauhkien kerääminen yhteen ja mallintaa jokaisesta uhkasta Rohmeyer ja Ben-Zvi (2015) artikkelin kaltainen skenaarioanalyysi, jossa tiedostetaan etukäteen mitä tietty uhka aiheuttaisi tietyn kehitettävän sovelluksen kohdalla (Saripalli ja Walters 2010).

Saripalli ja Walters (2010) toinen analyysimalli pyrkii laskemaan todennäköisyydet kullekin ensimmäisessä analyysissa kerätylle riskille. Huomionarvoista tässä mallissa on se, että huomioon otetaan vain ne riskit, joiden tiedetään olevan potentiaalisia uhkakuvia. Näistä potentiaalisista uhkakuvista pysytään täten laskemaan yhteinen todennäköisyys, jota voidaan hyödyntää ohjelmistokehityksessä (Saripalli ja Walters 2010).

Vaikka riskianalyysien toteuttamistapoja on monia, Dash ja Dash (2010) kuitenkin muistuttavat, että ohjelmistokehityksessä kaikkia riskitekijöitä ei pystytä kontrolloimaan tai ottamaan huomioon, mutta hyvillä riskianalyysi ja arviointi menetelmillä tärkeimmät ja samalla vakavimmat riskit pystytään kuitenkin herkästi tunnistamaan.

3.4 Jatkuva monitorointi (Continuous monitoring)

Jatkuvan monitoroinnin tavoitteena on selvittää ja kerätä tietoa käyttäjien suorittamista toiminnoista sekä sovelluksen tilasta ja käyttämistä resursseista (Lokawati ja Widyani 2019).

Lokawati ja Widyani (2019) mukaan SaaS-ohjelmistoa kehittävän organisaation tulisi ottaa huomioon neljä erilaista kategoriaa, jotka tulisi huomioida omina resursseinaan jatkuvan monitoroinnin toiminnallisuudessa. Artikkelissa listatut kategoriat olivat monitorointia varten kehitetty järjestelmä, asiakas, ohjelmiston käyttäjä ja itse SaaS-ohjelmisto.

SaaS-ohjelmistojen jatkuvalla monitoroinnilla pystytään Waltermire ym. (2019) mukaan tehokkaasti ja automaattisesti havaitsemaan tahallisesti tai tahattomasti aiheutetut toiminnot, jotka voivat vaikuttaa negatiivisesti SaaS-ohjelmistoa käyttävän yritykseen taloudellisesti ja operationaalisesti. Jatkuva monitorointi perustuu validien lokitietojen keräämiseen. Lokitiedoilla voidaan Waltermire ym. (2019) mukaan koostaa automaattisesti luotuja analyyseja ja raportteja, joista SaaS-ohjelmiston tarjoaja tai ohjelmistoa käyttävä käyttävä asiakas saa tietoa ja ohjausta siihen miten havaittu ongelma voitaisiin korjata.

Waltermire ym. (2019) esittelee lisäksi neljä erilaista skenaariota siitä, miten jatkuvaa monitorointia voidaan hyödyntää käyttäjien kirjautumisen yhteydessä. Waltermire ym. (2019) mukaan on tärkeää kerätä mahdollisimman paljon tapahtumapohjaista dataa erilaisista lähteistä. Näitä lähteitä on muun muassa palomuurit ja tunkeutumisen havaitsemisjärjestelmät sekä muu sovelluksesta saatava data (Waltermire ym. 2019). Monitorointia varten Waltermire ym. (2019) mukaan näistä lähteistä olisi hyvä kerätä seuraavanlaista dataa:

- Todennuksen ja valtuutusten epäonnistumiset ja onnistumiset
- Järjestelmätapahtumat ja sovelluksessa tapahtuvat virheet
- Muutokset käyttöoikeuksiin
- Järjestelmän järjestelmävalvojan oikeuksien käytön seuranta
- Pääsy arkaluontoisiin tietoihin
- Onnistuneet ja epäonnistuneet kirjautumisyritykset

Lokawati ja Widyani (2019) laajentavat Waltermire ym. (2019) näkemystä SaaS-ohjelmiston monitoroinnista siten, että sen lisäksi, että kerättäisiin ainoastaan käyttäjään ja hänen suorittamiin toimintoihin liittyvää tietoa, tulisi Lokawati ja Widyani (2019) mukaan kerätä dataa myös resurssien ja palvelun laskutuksen osalta. Palvelun laskutuksen huomiointi on tärkeää myös huomioida, koska SaaS-ohjelmistojen laskutus perustuu usein siihen kuinka paljon ohjelmistoa käytetään, joko käyttäjä- tai organisaatiotasolla.

3.5 Automatisoitu testaaminen

Kuten luvussa 2 mainittiin, SaaS-ohjelmistoille on tyypillistä palveluntarjoajan julkaisemat säännölliset päivitykset. Tämä tarkoittaa usein samalla ohjelmakoodin tehtäviä päivityksiä ja lisäyksiä. Srikanth ja Cohen (Srikanth ja Cohen 2011) mukaan säännölliset päivitykset luovat haastavan ympäristön ohjelmistotestaajille, joiden on testeillään varmistettava, että ohjelmisto toimii oikein päivitystenkin jälkeen.

Srikanth ja Cohen (2011) esittävät, että regressiotestauksessa olennaista on, että jatkokehitettävä ohjelmisto testataan aina muutosten jälkeen. Ajettavat testit voidaan myös ajastaa tarvittaessa.

Srikanth ja Cohen (2011) tutkimuksessa SaaS-ohjelmistot voidaan luokitella tapahtumapohjaiseksi järjestelmämalliksi (eng. event-driven system model). Tapahtumapohjaiseen järjestelmämalliin liittyy tutkimuksen mukaan muun muassa hiiren painallukset ja käyttäjän antamat syötteet, joista käynnistyy jonkinlainen prosessi, esimerkiksi tietueen tallennus tietokantaan. Srikanth ja Cohen (2011) ehdottavat, että testatessa tapahtumapohjaista SaaS-ohjelmistoa, käytettäisiin mallia, jossa luotaisiin neljän vaiheen mukaan konkreettinen skenaario, jolla lopulta testattaisiin ohjelmistoa.

1. Tunnistetaan käyttäjän syötteestä johtuvat virheet
2. Mallinnusvaihe ja siihen sisältyvät menetelmät
    - Määrittele abstraktit tapahtumat
    - Tapahtumien aikarajoitusten tunnistaminen
    - Testien kattavuuden määrittäminen
3. Tapahtumajonojen generointi
4. Luodaan konkreettinen skenaario

Makki, Van Landuyt ja Joosen (2016) käsittelevät artikkelissaan myös regressiotestausta, jossa he nostavat esille erilaisia haasteita koskien automaattista SaaS-ohjelmiston testaamista. Haasteiksi Makki, Van Landuyt ja Joosen (2016) nostavat muun muassa validien testitapausten luomisen ja sen miten käytettävää testikokoelmaa rajataan, tarkoittaen sitä, että onko aina tarpeen suorittaa koko järjestelmän perusteellista testaamista. Rajaamiseen liittyy artikkelin

mukaan myös se, että voiko suoritettavat automaattiset testit aiheuttaa jotakin ei-toivottuja sivuvaikutuksia, esimerkiksi kriittisen testidatan lähettämistä ulkopuolisille tahoille.

Makki, Van Landuyt ja Joosen (2016) esittävät automaattisen ja regressiotestaamisen työkaluiksi kolme erilaista ratkaisua, joista tässä tutkielmassa esittelen kaksi.

- Työnkulun manipulointi (engl. Workflow Manipulation)
- Regressiotestaaminen ja testikokoelmien muokkaaminen (engl. Regression Detection)

Makki, Van Landuyt ja Joosen (2016) esittämä työkalu "Työnkulun manipulointi"perustuu siihen, että järjestelmän testauksessa pystyttäisiin jättämään tarkoituksella tietyt vaiheet pois, jolloin pystytään välttämään ei-toivottujen sivuvaikutuksien tapahtuminen. Artikkelissa esitetään pois jätettävien vaiheiden tilalle niiden simulointia (eng. Mocking Activites), jossa käytännössä tuotetaan tietyn toiminnon vaikutus.

Makki, Van Landuyt ja Joosen (2016) artikkelissa käsitellään myös regressiotestaamista, jossa keskitytään siihen, että mitä testien osia on tarpeen suorittaa tehtyjen muutosten jälkeen. Makki, Van Landuyt ja Joosen (2016) esittävät neljä erilaista kriteeriä, joiden avulla kehittäjän pitäisi pystyä rajaamaan suoritettavia testejä tarpeen mukaan. Artikkelissa esitetyt kriteerit ovat seuraavat.

- Työnkulun muuttujien arvot
- Päätepisteet, joihin työnkulku päättyy
- Työnkulun aikana muokattu tai lähetetty data
- Työnkulun ohessa suoritetut välitapahtumat

Esimerkkinä regressiotestaus tapauksesta Makki, Van Landuyt ja Joosen (2016) antavat tilanteen, jossa kehittäjä ei ole kiinnostunut mihin työnkulun päätepisteeseen suoritus päättyy. Konkreettinen skenaario päätepisteen suorituksesta voisi olla, että suoritetaanko työnkulun lopussa uuden datan lisääminen tietokantaan vai muokataanko olemassa olevaa dataa.

3.6 Koodikatselmukset (Code reviews)

Bacchelli ja Bird (2013) tutkimuksessa haastatteluun vastanneen ohjelmistokehittäjän mukaan, koodikatselmointi voi olla, kustannuksiltaan jopa halvempi vikojen etsintämuoto, kuin esimerkiksi testaaminen. Bacchelli ja Bird (2013) mukaan koodikatselmoinneissa on vikojen etsimisen lisäksi kyse myös itse lähdekoodin parantamisesta ja tiedon jakamisesta muun tiimin kesken.

Bacchelli ja Bird (2013) artikkelissa esiin nostettu tiedon jakaminen nousee esimerkiksi siinä tilanteessa tärkeäksi, kun alkuperäisen koodin tuottanut henkilö ei ole saatavilla ja on tarve tehdä ohjelmistoon nopeasti muutoksia tai korjauksia. SaaS-ohjelmistojen säännöllisten päivitysten ja korjausten kohdalla tällainen skenaario voisi olla mahdollinen.

Koodikatselmointi suoritetaan usein niin, että kun ohjelmistokehittäjä on saanut hänelle määrätyn tehtävän valmiiksi, toinen kehittäjä tarkastaa ohjelmakoodin. Huomioitavaa on se, että tarkastusta tekevä kehittäjä ei ole sama henkilö, kuka on työstänyt varsinaisen toteutuksen.

Atlassian (n.d.) kehottaa tarkastusta tekevää kehittäjää ottamaan huomioon seuraavat kysymykset tarkastusta tehdessään:

- Sisältääkö koodi ilmeisiä logiikkavirheitä?
- Onko tehtävää varten määritellyt vaatimukset täysin toteutettu
- Onko uudet testiluokat riittäviä? Pitääkö olemassa olevia testejä muokata tai uusia?
- Onko tuotettu koodi organisaation kehityskäytänteiden mukainen?

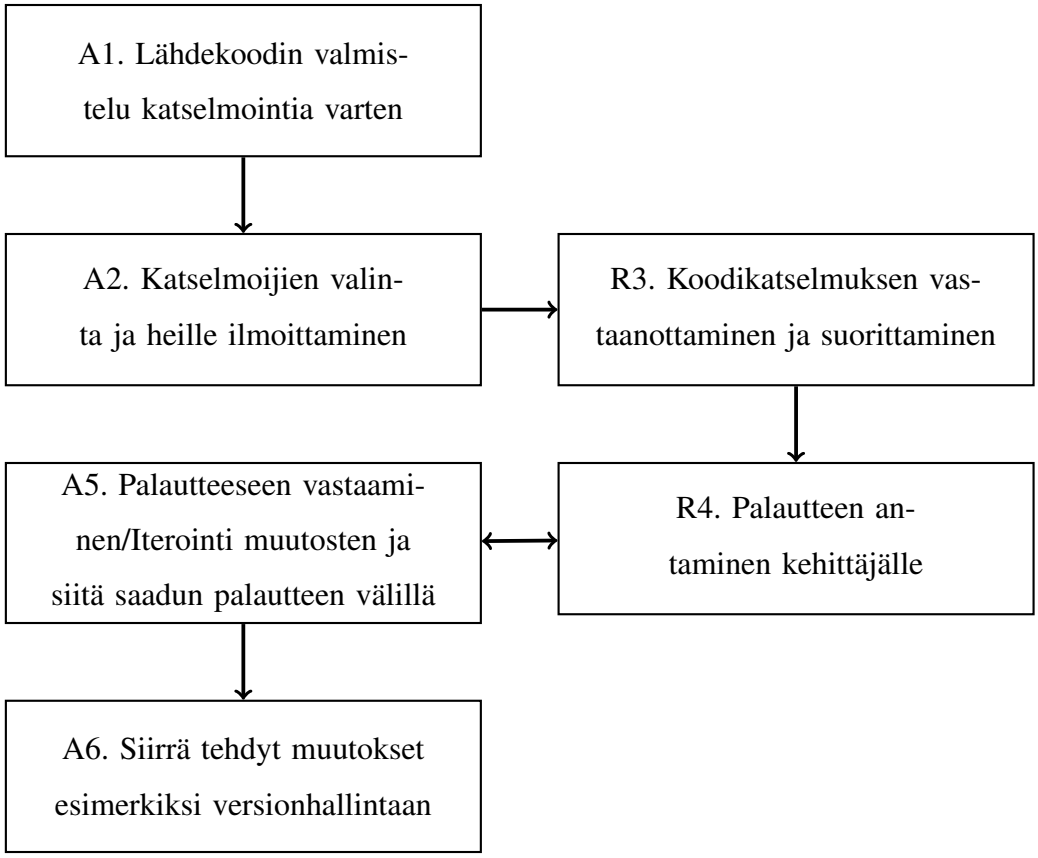
Bernhart ym. (2012) käsittelevät artikkelissaan myös koodikatselmuksia osana jatkuvaa ohjelmistokehitystä. Bernhart ym. (2012) mukaan koodikatselmukset ovat tärkeässä roolissa ohjelmiston kehitystyössä. Artikkelissa kuvaillaan koodikatselmusten yleiseksi käytänteeksi, sitä että katselmus suoritetaan jokaisen kehitysiteraation jälkeen. SaaS-ohjelmiston jatkuvassa kehitystyössä yhtenä iteraationa voidaaan pitää kuvion 1 mallia, jossa katselmoinnit tehdään testauksen ja julkaisu vaiheiden välissä.

Samankaltaisuutta Atlassian (n.d.) ja Bernhart ym. (2012) oli koodikatselmusten osalta se, että katselmoinnin tekevä henkilö tulee olla joku muu kuin koodin tuottanut henkilö. Bernhart ym. (2012) artikkelissa käsiteltiin tarkemmin myös sitä miten katselmointi voitaisiin toteut-

taa. Artikkelin mukaan katselmoija voisi avata tehdyt muutokset ja verrata tehtyjä muutoksia vanhaan versioon.

MacLeod ym. (2018) käsittelee myös artikkelissaan koodikatselmuksia osana ohjelmistokehitystyötä organisaation ja siihen kuuluvien kehittäjien näkökulmasta. MacLeod ym. (2018) artikkelissa pyritään kokoamaan kyselytutkimusten perusteella parhaat käytänteet koodikatselmointien suorittamiseksi jossa on otettu huomioon kehittäjän, katselmoijan ja organisaation näkökulmat.

MacLeod ym. (2018) on koonnut kuvion 2 mukaisen prosessin ja siihen liittyvät vaiheet, jotka tulisi suorittaa aina kehitysiteraation jälkeen (kts. kuvio 1). Kuviossa 2 vasemman puoleiset tekstilaatikot kuvaavat lähdekoodin työstäneen henkilön tehtäviä ja oikean puoleiset laatikot kuvaavat puolestaan koodikatselmoijan tehtäviä.



Kuvio 2. Koodikatselmoinnin päävaiheet (MacLeod ym. 2018)

Jokaiseen MacLeod ym. (2018) esittämiin vaiheisiin liittyy omia pienempiä toimintoja, jotka

tulisi jokaisen vaiheen kohdalla suorittaa.

Toiminnot ovat luonteeltaan joko itsenäistä työtä, kuten esimerkiksi tehtyjen muutosten ryhmittelyä tai katselmoinnista koostettavien muistiinpanojen tekoa. Vaiheet A5 ja R4 olivat kuitenkin Bacchelli ja Bird (2013) mukaan tärkeitä kommunikaation osalta. Bacchelli ja Bird (2013) sekä MacLeod ym. (2018) painottavatkin tutkimuksissaan kommunikaation tärkeyttä kehittäjän ja katselmoija välillä, jotta katselmointiprosessi olisi sujuvaa ja kehitysiteraatio saataisiin suoritettua loppuun.

MacLeod ym. (2018) painottaa lisäksi myös organisaation osallistumisen tärkeyttä, varsinkin koodikatselmointikulttuurin luomisen osalta, jotta jokaisen kehitysiteraation jälkeinen koodikatselmointi olisi vakituinen ja prosessiltaan yleisesti aina samanlainen.

Esimerkkinä MacLeod ym. (2018) esittämän mallin soveltamisesta ja koodikatselmoinnin suorittamisesta käytännössä on GitHub verkkosivuston tarjoama koodikatselmointiprosessi. GitHub on käyttöliittymä Git-versionhallinalle.

GitHub tarjoaa erilaisia työkaluja muutosten tarkastelulle ja lajittelulle, joista tärkeimpänä voisi pitää Pull-pyyntö (eng. Pull request) toimintoa. Pull-pyyntö mahdollistaa sen, että kehittäjä pystyy tarkastamaan tekemänsä muutokset ja määrittämään yksittäisen henkilön tai ryhmän suorittamaan katselmoinnin (“About pull requests”, n.d.). Bacchelli ja Bird (2013) ja MacLeod ym. (2018) tärkeäksi nostama kommunikaatio toteutuu Pull-pyynnön osalta niin, että kehittäjä ja katselmoija pystyvät käymään keskustelua ja tekemään muistiinpanoja avatun pyynnön sisällä. Pyynnön aktiviteeteista pystytään lähettämään myös osallisille aina ilmoitus, kun katselmointiprosessissa on tapahtunut jotain.

GitHub tarjoaa myös käyttöliittymässään mahdollisuuden vertailla lähdekoodiin tehtyjä muutoksia vanhempiin versioihin. Myös kommenttien lisääminen on mahdollista esimerkiksi tietyn muutoksen kohdalle, jolloin kommunikaation osalta pystytään olemaan entistä tarkempi.

4 Yhteenveto

SaaS-ohjelmistojen käytön ja niitä palveluna tarjovavien organisaatioiden lisääntyessä on ohjelmistojen laatuun panostettava entistä enemmän.

Tutkielma antaa yleiskuvan siitä, mitä ominaispiirteitä kuuluu jatkuvaan ohjelmistokehitykseen ja minkälaisia asioita kehitystyössä tulisi ottaa huomioon erityisesti SaaS-ohjelmistojen kehitystyössä.

SaaS-ohjelmistot ovat nykypäivänä yksi suosituimmista ohjelmiston toimitusmallista, jotka näyttävät täyttävän yhä useamman yrityksen vaatimukset käytettävästä järjestelmästään (Achar 2016).

Tutkielmassa käytetyn lähdemateriaalin perusteella jatkuvan SaaS-ohjelmiston kehitystyössä olisi hyvä ottaa huomioon koko Patel ja Chouhan (2016) esittämän mallin vaiheisiin liittyvät menetelmät ja toiminnot. Riskianalyysit, järjestelmän jatkuva monitorointi, automaattinen testaaminen sekä ennen julkaisua tapahtuva koodikatselmointi kattaa hyvin toteutettuna koko kehitysiteraation elinkaaren.

Tutkielman heikkoutena on, että SaaS-ohjelmistojen kehitystyötä jatkuvan ohjelmistokehityksen kontekstissa on tutkittu vähän tai epäsuorasti. Jatkotutkimuksena voitaisiin tutkia riskianalyyseja, jatkuvaa monitorointia, automaattista testaamista sekä koodikatselmointeja yksittäin osana jatkuvaa SaaS-ohjelmiston kehitystyötä. Jatkotutkimuksissa voitaisiin keskittyä yhteen menetelmään ja tämän teknisiin ominaisuuksiin ja vaatimuksiin.

Jatkotutkimuksena voitaisiin myös selvittää, mitkä ovat SaaS-ohjelmiston loppukäyttäjien vaatimukset käytettävälle ohjelmistolle.

Lähteet

"About pull requests". GitHub Docs. n.d. Viitattu 15. maaliskuuta 2023. https://ghdocs-prod.azurewebsites.net/en/pull-requests/collaborating-with-pull-requests/proposing-changes-to-your-work-with-pull-requests/about-pull-requests.

Achar, Sandesh. 2016. "Software as a Service (SaaS) as Cloud Computing: Security and Risk vs. Technological Complexity". *Engineering International* 4 (joulukuu): 79–88. https://doi.org/10.18034/ei.v4i2.633.

Akinrolabu, Olusola, Steve New ja Andrew Martin. 2019. "Assessing the Security Risks of Multicloud SaaS Applications: A Real-World Case Study". Teoksessa *2019 6th IEEE International Conference on Cyber Security and Cloud Computing (CSCloud)/ 2019 5th IEEE International Conference on Edge Computing and Scalable Cloud (EdgeCom),* 81–88. https://doi.org/10.1109/CSCloud/EdgeCom.2019.00-14.

Atlassian. n.d. "What is a Code Review & How It Can Save Time". Atlassian. Viitattu 11. maaliskuuta 2023. https://www.atlassian.com/agile/software-development/code-reviews.

Bacchelli, Alberto, ja Christian Bird. 2013. "Expectations, outcomes, and challenges of modern code review". Teoksessa *2013 35th International Conference on Software Engineering (ICSE),* 712–721. https://doi.org/10.1109/ICSE.2013.6606617.

Belbergui, Chaimaa, Najib Elkamoun ja Rachid Hilal. 2017. "Cloud computing: Overview and risk identification based on classification by type". Teoksessa *2017 3rd International Conference of Cloud Computing Technologies and Applications (CloudTech),* 1–8. https://doi.org/10.1109/CloudTech.2017.8284697.

Bernhart, Mario, Stefan Strobl, Andreas Mauczka ja Thomas Grechenig. 2012. "Applying Continuous Code Reviews in Airport Operations Software". Teoksessa *2012 12th International Conference on Quality Software,* 214–219. https://doi.org/10.1109/QSIC.2012.61.

Bordak, Lukas. 2019. “Cloud Computing Security”. Teoksessa *2019 17th International Conference on Emerging eLearning Technologies and Applications (ICETA),* 87–92. https://doi.org/10.1109/ICETA48886.2019.9040043.

Dash, Rasmita, ja Rajashree Dash. 2010. “Risk assessment techniques for software development”. *European journal of scientific research* 42 (4): 629–636.

Fitzgerald, Brian, ja Klaas-Jan Stol. 2014. “Continuous Software Engineering and beyond: Trends and Challenges”. Teoksessa *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering,* 1–9. RCoSE 2014. Hyderabad, India: Association for Computing Machinery. ISBN: 9781450328562. https://doi.org/10.1145/2593812.2593813.

Gibson, Joel, Robin Rondeau, Darren Eveleigh ja Qing Tan. 2012. “Benefits and challenges of three cloud computing service models”. Teoksessa *2012 Fourth International Conference on Computational Aspects of Social Networks (CASoN),* 198–205. https://doi.org/10.1109/CASoN.2012.6412402.

Kumar, KKM. 2014. “Software as a service for efficient cloud computing”. *environment* 7:10.

Kunz, Immanuel, Angelika Schneider ja Christian Banse. 2022. “A Continuous Risk Assessment Methodology for Cloud Infrastructures”. Teoksessa *2022 22nd IEEE International Symposium on Cluster, Cloud and Internet Computing (CCGrid),* 1042–1051. https://doi.org/10.1109/CCGrid54584.2022.00127.

Lokawati, Harum, ja Yani Widyani. 2019. “Monitoring System of Multi-Tenant Software as a Service (SaaS)”. Teoksessa *2019 International Conference on Data and Software Engineering (ICoDSE),* 1–5. https://doi.org/10.1109/ICoDSE48700.2019.9092741.

MacLeod, Laura, Michaela Greiler, Margaret-Anne Storey, Christian Bird ja Jacek Czerwonka. 2018. “Code Reviewing in the Trenches: Challenges and Best Practices”. *IEEE Software* 35 (4): 34–42. https://doi.org/10.1109/MS.2017.265100500.

Makki, Majid, Dimitri Van Landuyt ja Wouter Joosen. 2016. “Automated Workflow Regression Testing for Multi-Tenant SaaS: Integrated Support in Self-Service Configuration Dashboard”. Teoksessa *Proceedings of the 7th International Workshop on Automating Test Case Design, Selection, and Evaluation,* 70–73. A-TEST 2016. Seattle, WA, USA: Association for Computing Machinery. ISBN: 9781450344012. https://doi.org/10.1145/2994291.2994302.

O’Connor, Rory V., Peter Elger ja Paul M. Clarke. 2017. “Continuous software engineering—A microservices architecture perspective”. E1866 JSME-16-0193.R2, *Journal of Software: Evolution and Process* 29 (11): e1866. https://doi.org/https://doi.org/10.1002/smr.1866.

Odun-Ayo, Isaac, Sanjay Misra, Olusola Abayomi-Alli ja Olasupo Ajayi. 2017. “Cloud Multi-Tenancy: Issues and Developments”. Teoksessa *Companion Proceedings of The10th International Conference on Utility and Cloud Computing,* 209–214. UCC ’17 Companion. Austin, Texas, USA: Association for Computing Machinery. ISBN: 9781450351959. https://doi-org.ezproxy.jyu.fi/10.1145/3147234.3148095.

Patel, Jigar, ja Ankit Chouhan. 2016. “An approach to introduce basics of Salesforce.com: A cloud service provider”. Teoksessa *2016 International Conference on Communication and Electronics Systems (ICCES),* 1–8. https://doi.org/10.1109/CESYS.2016.7889991.

Peake, Chris. 2012. “Security in the cloud: Understanding the risks of cloud-as-a-service”. Teoksessa *2012 IEEE Conference on Technologies for Homeland Security (HST),* 336–340. https://doi.org/10.1109/THS.2012.6459871.

Rohmeyer, Paul, ja Tal Ben-Zvi. 2015. “Managing Cloud Computing risks in financial services institutions”. Teoksessa *2015 Portland International Conference on Management of Engineering and Technology (PICMET),* 519–526. https://doi.org/10.1109/PICMET.2015.7273004.

Saripalli, Prasad, ja Ben Walters. 2010. “QUIRC: A Quantitative Impact and Risk Assessment Framework for Cloud Security”. Teoksessa *2010 IEEE 3rd International Conference on Cloud Computing,* 280–288. https://doi.org/10.1109/CLOUD.2010.22.

Sharma, Vani Dayal, Somya Agarwai, Syeda Shira Moin ja Mohammed Abdul Qadeer. 2017. “Security in cloud computing”. Teoksessa *2017 7th International Conference on Communication Systems and Network Technologies (CSNT),* 234–239. https://doi.org/10.1109/CSNT.2017.8418544.

Singh, Akanksha, Smita Sharma, Shipra Ravi Kumar ja Suman Avdesh Yadav. 2016. “Overview of PaaS and SaaS and its application in cloud computing”. Teoksessa *2016 International Conference on Innovation and Challenges in Cyber Security (ICICCS-INBUSH),* 172–176. https://doi.org/10.1109/ICICCS.2016.7542322.

Srikanth, Hema, ja Myra B. Cohen. 2011. “Regression testing in Software as a Service: An industrial case study”. Teoksessa *2011 27th IEEE International Conference on Software Maintenance (ICSM),* 372–381. https://doi.org/10.1109/ICSM.2011.6080804.

Uzunbayir, Serhat, ja Kaan Kurtel. 2018. “A Review of Source Code Management Tools for Continuous Software Development”. Teoksessa *2018 3rd International Conference on Computer Science and Engineering (UBMK),* 414–419. https://doi.org/10.1109/UBMK.2018.8566644.

Waltermire, Karen, Kelley Burgin, Chinedum Irrechukwu, Harry Perper, Susan Prince ja Devin Wynne. 2019. “Continuous Monitoring for IT Infrastructure”.