Samuel Lehikoinen

Otto Virtanen

Testausstrategia Salesforce-kehitykseen

Tietotekniikan Pro Gradu -tutkielma

9. toukokuuta 2023

Jyväskylän yliopisto

Informaatioteknologian tiedekunta

Tekijät: Samuel Lehikoinen ja Otto Virtanen

Yhteystiedot: samuel.h.lehikoinen@student.jyu.fi
ja otto.v.virtanen@student.jyu.fi

Ohjaaja: Tommi Mikkonen

Työn nimi: Testausstrategia Salesforce-kehitykseen

Title in English: Software testing strategy for Salesforce development

Työ: Pro Gradu -tutkielma

Opintosuunta: Ohjelmistotekniikka

Sivumäärä: 64+0

Tiivistelmä: Jatkuvasti kehittyvässä ohjelmistokehitysmaailmassa testaus voi usein olla ero onnistumisen ja epäonnistumisen välillä. Tästä huolimatta monet organisaatiot kamppailevat johdonmukaisen testausstrategian luomisessa, mikä voi johtaa kalliisiin virheisiin ja viivästyksiin. Tämän Pro Gradu -tutkielman tavoitteena on suunnittelutieteen periaatteita noudattaen käsitellä tätä haastetta kohdeorganisaation kannalta ehdottamalla artefaktia kattavan testausstrategian muodossa. Tätä strategiaa kehitettiin ja testattiin kirjallisuuden ja tapaustutkimusten avulla, millä saavutettiin huomattavia parannuksia testauksen tehokkuudessa ja tarkkuudessa. Muodostettu artefakti tarjoaa organisaatioille etenemissuunnitelman saavuttaakseen yhtenäisemmän ja tehokkaamman testauslähestymistavan, millä saavutetaan parempia tuloksia ohjelmistokehitysprojekteissa.

Avainsanat: Salesforce, ohjelmistotestaus, ohjelmiston laatu, testausstrategia, ohjelmistokehitysprosessi

Abstract: In software business, testing can often be the difference between success and failure. Despite this, many organizations struggle to establish a coherent testing strategy, which might result in costly mistakes and delays. The aim of this thesis, following the principles of design science research methodology, is to address this issue for a target organization by proposing an artefact in the form of a comprehensive testing strategy. Through research and

case studies, this strategy was developed and tested, resulting in significant improvements in testing efficiency and accuracy. The proposed artefact provides a roadmap for organizations to follow to achieve a more cohesive and effective testing approach, ultimately leading to better outcomes for software development projects.

Keywords: Salesforce, software testing, software quality, testing strategy, software development process

Esipuhe

Haluamme esittää vilpittömät kiitoksemme ohjaajallemme ohjauksesta ja tuesta koko tämän tutkimusmatkan ajan. Haluamme myös kiittää kohdeorganisaatiota ja yliopistoa mahdollisuudesta toteuttaa tämä tutkimusprojekti.

Tämä Pro Gradu on kirjoitettu rakkaudella ohjelmistotestausta kohtaan ja sen tavoitteena on antaa kattava esitys siitä, miten ohjelmistotestausta voidaan suorittaa Salesforce-kontekstissa. Tavoitteenamme on tarjota lukijoille käsitys Salesforce-ohjelmistojen testaamisen haasteista ja tietysti myös mahdollisuuksista sekä erilaisista työkaluista ja menetelmistä, joilla voidaan varmistaa ohjelmistojen korkea laatu.

Pro Gradua tehdessä työpanos jakautui tasaisesti molemmille tutkijoille. Samuel ollen päävastuussa luvuista 2 ja 5. Otto ollen päävastuussa luvuista 3 ja 4. Johtopäätökset ja artefakti tuotettiin yhdessä. Molemmat tutkijat täydensivät ja rikastivat toistensa päävastuilla olleita lukuja siten, että lopputulos on paras mahdollinen.

Toivomme tämän Pro Gradun olevan arvokas resurssi niille, jotka etsivät uutta tietoa ohjelmistotestauksesta Salesforce-kontekstissa oli lukija sitten kehittäjä, testaaja, projektipäällikkö tai muu Salesforce-kehitysprojekteista kiinnostunut henkilö. Lopuksi toivomme, että tämä Pro Gradu edesauttaa laajempaa tietoisuutta ohjelmistotestauksesta ja auttaa edistämään ohjelmistosuunnittelun alaa kokonaisuutena.

Discordissa 9. toukokuuta 2023

Samuel Lehikoinen ja Otto Virtanen

Termiluettelo

Apex	Salesforcen käyttämä vahvan tyypityksen oliosuuntautunut palvelinpuolen ohjelmointikieli.
CI/CD	Continuous integration & Continuous delivery on ohjelmistokehityksen menetelmä jatkuvalle ohjelmiston versioiden julkistamiselle.
CMMi	Capability Maturity Model integration on organisaation ohjelmistokehitysprosessin arviointi- ja kehitysmalli.
CRM	Asiakkuudenhallintajärjestelmä (engl. Customer Relationship Management) on järjestelmä, joka on tarkoitettu organisaation asiakkaiden ja mahdollisien asiakkaiden kanssa käytävän vuorovaikutuksen hallintaan.
DML	Tietokantaoperaatiot (engl. Data Manipulation Language), joiden avulla voi manipuloida tietokannan tietueita. Seuraavat DML-lausekkeet ovat käytettävissä Salesforcessa: insert, update, upsert, delete, undelete ja merge.
DSRM	Design Science Research Methodology on suunnittelutieteen tutkimusmenetelmä.
Experience Cloud	Salesforce-ohjelmistoalustan tarjoama ohjelmistotuote asiakas- ja kumppaniportaalien luontiin.
LWC	Lightning Web Component eli Lightning-verkkokomponentti on viitekehys käyttöliittymäpuolen verkkokomponenttien kehittämiseen erityisesti Salesforce-ympäristöissä.
Moniasiakkainen	(Engl. Multitenant) tarkoittaa palvelimella pyörivää ohjelmistoinstanssia, joka palvelee useaa asiakasta.
Sales Cloud	Salesforce-ohjelmistoalustan tarjoama CRM-ohjelmistotuote.
Salesforce-ympäristö	Salesforce-ohjelmistoalustan instanssi, joka sisältää yhden organisaation käyttäjät, datan ja konfiguraatiot.
Salesforce Metadata	Kuvaa Salesforce-instanssin rakennetta ja toimintalogiikkaa. Salesforce Metadataa ovat muun muassa konfiguraatiot, auto-

	maatiologiikat ja käyttöliittymän asettelut.
Salesforce	Yhdysvaltalainen pörssiyhtiö, joka kehittää ja tarjoaa Salesforce Customer 360 -ohjelmistotuoteperhettä. Ohjelmistotuoteperhe tarjoaa ohjelmistoalustan, teknologiat ja viitekehyksen sovellusten kehittämiseen.
Sandbox	Kopio tietyn organisaation tuotantoympäristöstä. Sandbox-ympäristöjä on erilaisia ja niitä voi käyttää muun muassa testaukseen ja loppukäyttäjien koulutukseen.
Scratch org	Kertakäyttöinen Salesforce-ympäristö, jota voidaan käyttää kehityksessä tai testaamisessa.
SDLC	Software Development Lifecycle tarkoittaa ohjelmistokehityksen elinkaarta.
Service Cloud	Salesforce-ohjelmistoalustan tarjoama asiakaspalveluohjelmistotuote.
SLDS	Salesforce Lightning Design System on viitekehys verkkokomponenttien tyylittelyyn.
sObjekti	Salesforcen vakio- tai mukautettu objekti, joka säilöö tietoa Salesforcen tietokannassa.
SOQL	Salesforce Object Query Language on Salesforcen jäsennetty kyselykieli.
TMMi	Test Maturity Model integration on organisaation ohjelmistotestauksen arviointi- ja kehitysmalli.

Kuviot

Taulukot

Sisällys

1 Johdanto

Nopeatempoisessa tekniikkakeskeisessä maailmassa paine ohjelmistojen nopeasta julkaisemisesta tapahtuu usein laadun kustannuksella (Slaughter, Harter ja Krishnan 1998). Testaamista ajatellaan monesti pakollisena pahana ohjelmistokehityksessä, eikä siihen aina nähdä tarpeeksi paljon aikaa tai vaivaa, kun ohjelmiston uuden version pitäisi olla pikimmiten tuotantokäytössä. Testaus on tärkeä osa laadukkaan ohjelmiston kehittämistä, sillä laadukas ohjelmisto koostuu monista toiminnoista ja tapahtumista, joihin muun muassa testaus kuuluu (Seth ym. 2012).

Ohjelmistoja on kaikkialla matkapuhelimista autoihin ja puettavista älylaitteista monimutkaisiin lennonjohtojärjestelmiin. Teknologian kehittyessä ja ohjelmistojen integroituessa yhä arkipäiväisempiin tapahtumiin tarve korkealaatuisille ohjelmistotuotteille kasvaa. Standardissa ISO/IEC 25000 (2014) ohjelmiston laatu määritellään ohjelmistotuotteen kyvykkyytenä täyttää ilmoitettuja ja oletettuja tarpeita, kun sitä käytetään tietyissä olosuhteissa.

Ohjelmiston laatu on kriittinen osa ohjelmistokehitystä, joka vaikuttaa suoraan ohjelmistoprojektin onnistumiseen tai epäonnistumiseen (Cockburn ja Highsmith 2001). Nykypäivän nopeatempoisessa ja kilpailullisessa ohjelmistoteollisuudessa laadukkaan ohjelmistotuotteen toimittaminen on ratkaisevan tärkeää sidosryhmien odotusten täyttämiseksi ja positiivisen käyttökokemuksen tarjoamiseksi. Ohjelmistotuotteiden kehitys harvoin on lineaarista, vaan prosessi on usein iteratiivinen. Tämä johtaa siihen, että laadukkaan ohjelmistotuotteen saavuttamiseksi vaatimuksia tulee pitää mielessä ja selvittää jatkuvasti koko kehitysprosessin ajan. (Seth ym. 2012)

Ohjelmiston laatuun vaikuttavia tekijöitä on useita eri kategorioissa. Ohjelmistotuotteen muokkaukseen liittyviä laatutekijöitä ovat ylläpidettävyys, joustavuus ja testattavuus. Ohjelmistotuotteen toimintaan liittyviä laatutekijöitä ovat oikeellisuus, luotettavuus, tehokkuus, eheys ja käytettävyys. Ohjelmistotuotteen siirtymiseen liittyviä laatutekijöitä ovat siirrettävyys, uudelleenkäytettävyys ja yhteentoimivuus. (McCall, Richards ja Walters 1977, s. 3-1–3-4)

Laatu on tunnistettu olennaiseksi elementiksi ohjelmistokehityksessä. Ohjelmiston laatua pystytään parantamaan täyttämällä määriteltyjä laatutekijöitä. Tehokas tapa laatutekijöiden

varmistamiseksi ja kehittämiseksi on ohjelmiston testaaminen eri tekniikoin (Seth ym. 2012).

Pilvipohjaisia Salesforce-järjestelmiä otetaan yhä enemmän käyttöön asiakkuudenhallinta-järjestelmiksi (CRM) ja niihin tehdään rohkeammin ohjelmakoodia vaativia räätälöityjä toiminnallisuuksia, jotta pystytään toteuttamaan organisaatioiden vaativimpia liiketoimintaa sujuvoittavia toteutusratkaisuja (Patel ja Chouhan 2016). Monimutkaisten toteutusratkaisujen lisääntyessä yksi tärkeimpiä haasteita on ohjelmiston laadun varmistaminen tehokkaasti (Cennamo, Ozalp ja Kretschmer 2018). Tutkimuksessaan Seth ym. (2012) tuovat esille, että tietoisuus ohjelmiston laadusta on keskeinen asia koko ohjelmiston elinkaaren ajan ja että ohjelmiston laatuun vaikuttavia tekijöitä ovat muun muassa suunnittelu, kaavoittaminen ja testaaminen. Tutkimuksessa perehdymme erityisesti näistä viimeiseen.

Viimeaikainen kehitys ohjelmistotestauksen alalla sisältää muun muassa automatisoitujen testaustyökalujen käytön lisääntymisen, jatkuvien testauskäytäntöjen käyttöönoton ja testaustoimintojen integroimisen Salesforcen yleiseen kehitysprosessiin (Sneha ja Malle 2017; Chaves Da Silva ym. 2015). Kehityksestä huolimatta haasteita löytyy yhä muun muassa räätälöityjen toiminnallisuuksien, integraatioiden ja tehokkuuksien testaamisessa moniympäristöisissä järjestelmäkokonaisuuksissa, kuten Salesforcessa (Riungu, Taipale ja Smolander 2010).

Tämän tutkimuksen tarkoituksena on arvioida, miten testausta tulisi Salesforce-ympäristöissä toteuttaa eri kokoisissa ja sisältöisissä projekteissa tehokkaasti. Tutkimuksessa käytetään tutkimusmenetelmänä suunnittelutiedettä tarkoituksena luoda Salesforce-kehitykseen erikoistuneelle kohdeorganisaatiolle artefaktiksi testausstrategia. Kyseisellä kohdeorganisaatiolla ei ennen ole ollut testausstrategiaa. Strategian tarkoituksena on parantaa kohdeorganisaation tuottaman ohjelmiston laatua testauskäytänteitä optimoiden sekä selkeyttää testaamisen vaiheita ja syvyyttä.

Tutkielman rakenne on seuraavanlainen. Luvussa 2 perehdytään kirjallisuuteen määritelläksemme Salesforcen tuoman näkökulman ohjelmistotestaukseen. Luvussa 3 esitetään tutkimusasetelma, jossa kerrotaan tarkemmin tutkimuksen konteksti, tutkimuskysymykset ja käytetty tutkimusmenetelmä. Luvussa 4 esitetään toteutettu artefakti ja luvussa 5 sitä analysoidaan ja reflektoidaan. Tutkielma päättyy lukuun 6, jossa tehdään yhteenveto tutkimuksen

tuloksista.

2 Teoreettinen viitekehys

Tässä luvussa esitetään tutkimuksen teoreettinen viitekehys. Luvun sisältö on seuraavanlainen. Ensimmäisenä aliluvussa 2.1 esitetään asiakkuudenhallinta ja perehdytään syihin, miksi asiakkuudenhallintaa tarvitaan. Aliluvussa 2.2 perehdytään pilvipalveluiden konsepteihin ja keskeisiin ominaisuuksiin. Viimeisessä aliluvussa 2.3 esitetään Salesforce, käsitellään sen ominaisuuksia ja palveluita. Lisäksi aliluvussa käsitellään kolme keskeistä teemaa Salesforce-kehitykseen liittyen. Ensimmäisenä perehdytään Salesforce-kehitykseen 2.3.1, jonka jälkeen keskitytään Salesforce-testauksen 2.3.2 yleisimpiin haasteisiin ja parhaisiin käytäntöihin. Aihepiirissä käsitellään räätälöityjen toiminnallisuuksien erityyppisistä ja -tasoisista testaamisista. Salesforce-aliluvun lopuksi esitetään Salesforce-kehitykseen ja -testaukseen tarjolla olevia työkaluja 2.3.3 sekä niiden ominaisuuksia ja vaikutusta testaukseen. Luvun päättää aliluku 2.4 CMMi ja TMMi ohjelmistokehitysprosessin arviointi- ja kehitysmalleista.

2.1 Asiakkuudenhallinta

Menestyäkseen nykypäivän liiketoimintaympäristössä yritysten tulee ensisijaistaa asiakastyytyväisyys ja rakentaa vahvoja asiakassuhteita (Tarasi 2007; Mithas, Krishnan ja Fornell 2005). Organisaatiot keskittyvät yhä enemmän saumattoman ja erityisesti henkilökohtaisen asiakaskokemuksen tarjoamiseen. Asiakkuudenhallinnalla (engl. Customer Relationship Management, lyh. CRM) tarkoitetaan teknologiaa asiakassuhteiden ja vuorovaikutuksen hallintaa (Patel ja Chouhan 2016). Kuten Salesforce (2023h) kuvailee, CRM:n avulla voi parantaa liikesuhteita ja kasvattaa yrityksen liiketoimintaa. CRM voi kattaa erilaisia työkaluja ja ohjelmistoja, mutta pohjimmiltaan se on strategia ja lähestymistapa asiakassuhteiden hallintaan ja asiakaskokemuksen optimointiin.

Williamsin (2014, s. 1–10) mukaan organisaatiot tähtäävät ikuisten asiakassuhteiden luomiseen. CRM:ää käytetään asiakasvuorovaikutusten ja -tietojen hallintaan ja analysoimiseen koko asiakaselinkaaren ajan aina ensimmäisestä yhteydenotosta myynnin jälkeiseen tukeen saakka (Tarasi 2007). Keskittämällä asiakastiedot organisaatiot voivat saada 360:n asteen nä-

kymän asiakkaistaan ja käyttää näitä tietoja kehittääkseen toimintaansa muun muassa vuorovaikutuksen personoimiseksi, asiakastyytyväisyyden parantamiseksi ja organisaation kasvun edistämiseksi. Oikealla CRM-järjestelmällä organisaatio voi virtaviivaistaa prosessejaan ja luoda asiakaskeskeisiä työnkulkuja eli lisätä tuottavuutta. (Salesforce 2023h)

CRM-järjestelmän avulla voi asiakastietojen hallinnan lisäksi parantaa myynnin hallintaa, ennustaa myyntiä, analysoida myyntitietoja ja luoda mukautettuja raportteja (Tarasi 2007). Luomalla ja hyödyntämällä automaatioita voi säästää huomattavasti aikaa ja vaivaa, joka tavallisesti menisi manuaalisesti hitaisiin tehtäviin, kuten tietojen syöttämiseen, liidien eli potentiaalisten ostavien asiakkaiden hallintaan tai esimerkiksi tietueiden ryhmittelyyn.

Digitalisaation yleistyessä organisaatiot ymmärtävät erilaisten teknologioiden yhdistämisen tärkeyden liiketoimintaprosessien tehostamiseksi. Asiakkuudenhallintajärjestelmien nousun myötä tarve integraatioille muihin liiketoimintajärjestelmiin, kuten sähköpostiin, sosiaaliseen mediaan, toiminnanohjausjärjestelmään (engl. Enterprise Resource Planning; lyh. ERP), henkilönhallintajärjestelmään (engl. Human Resources; lyh. HR) ja liiketoimintatiedon hallintajärjestelmään (engl. Business Intelligence; lyh. BI) on tullut yhä tärkeämmäksi (Choudhury ja Harrigan 2014). Integroituessaan muihin teknologioihin ja työkaluihin CRM-järjestelmät voivat laajentaa kykyään tarjota organisaatioille uusia mahdollisuuksia parantaa myynti-, markkinointi- ja asiakaspalvelutyötään. Integraatiot tekevät tiedon saannista saumatonta ja laaja-alaista. Integraatioilla voi olla merkittävä vaikutus liiketoiminnan suorituskykyyn ja tarjoamiin ominaisuuksiin. (Patel ja Chouhan 2017)

Asiakkuudenhallintajärjestelmä (CRM) on keskeinen osa mille tahansa yritykselle, mikä haluaa onnistua nykypäivän kilpailullisilla markkinoilla. CRM:ää käyttämällä organisaatiot voivat hallita asiakasvuorovaikutustaan ja -suhteitaan tehokkaammin, mikä parantaa asiakastyytyväisyyttä, lisää myyntiä ja viime kädessä lisää kannattavuutta. CRM:n tuoma laaja valikoima mahdollisuuksia auttaa organisaatioita tehostamaan toimintaansa, parantamaan viestintää ja tekemään tietoisempia päätöksiä keräämiensä tietojen perusteella.

2.2 Pilvipalvelut

Digitalisaation myötä tietokoneet ovat kehittyneet valtavasti. Teknologian kehittyessä ja organisaatioiden monimutkaistuessa tarve kehittyneemmille laskentaratkaisuille kasvoi. Tämä johti paikan päällä olevien ohjelmistojen ja sisäisten ratkaisujen kehittämiseen, joita pidettiin sekä kalliina että vaikeina ylläpitää datakeskusten, laitteiston ja ohjelmistotuen tarpeen vuoksi (Gibson ym. 2012).

Pilvitekniikan käyttöönotto on mullistanut tavan, jolla yritykset lähestyvät teknologiainfrastruktuuriaan. Pilvipalveluista on tullut suosittu tapa käyttää ja hallita IT-resursseja Internetin kautta sen joustavuuden, kustannustehokkuuden, skaalautuvuuden, saavutettavuuden, luotettavuuden ja tietoturvallisuuden vuoksi. (Riungu, Taipale ja Smolander 2010; Gibson ym. 2012)

Pilvipalveluiden käyttöönotto ei ole vain helpottanut yritysten pääsyä tehokkaisiin laskentaresursseihin, vaan se on myös muuttanut tapaa, jolla organisaatiot hoitavat asiakassuhteitaan. Pilvipohjaisten CRM-ratkaisujen nousun myötä kaikenkokoiset yritykset voivat nyt hyödyntää tehokkaita työkaluja tehostaakseen myynti- ja markkinointitoimintojaan, automatisoidakseen keskeisiä liiketoimintaprosesseja ja parantaakseen asiakastyytyväisyyttä. Pilvipohjaisten CRM-järjestelmien avulla yritykset voivat käyttää tietojaan ja sovelluksiaan mistä tahansa ja milloin tahansa millä tahansa laitteella, jossa on Internet-yhteys. Näin myynti- ja markkinointitiimit voivat käyttää samoja tietoja reaaliajassa sijainnistaan riippumatta. (Gupta ja Arora 2013, s. 9–13)

Tutkimuksessaan Gupta, Verma ja Janjua (2018) toteavat, että pilvipalveluiden käyttäjiä on kolmentyyppisiä: loppukäyttäjiä, myyntihenkilöitä ja ohjelmistokehittäjiä, joista jokaisella on erilaiset vaatimukset pilvipalveluille. Pilvipalvelut voidaan pääpiirteittäin jakaa kolmeen malliin: Infrastruktuuri palveluna (engl. Infrastructure as a Service; lyh. IaaS), Alusta palveluna (engl. Platform as a Service; lyh. PaaS) ja Ohjelmisto palveluna (engl. Software as a Service; lyh. SaaS).

IaaS on yksinkertaisin pilvipalvelumalli, joka tarjoaa perusinfrastruktuurin komponentit, kuten palvelimet, tallennus- ja verkkoresurssit. IaaS-palveluntarjoajat tarjoavat tyypillisesti mahdollisuuden kasvattaa tai vähentää asiakasorganisaation tarvitsemia resursseja ja laskuttavat

käytön perusteella. Näin organisaatiot voivat välttää oman fyysisen infrastruktuurin rakentamisen ja ylläpidon korkeita kustannuksia. (Gibson ym. 2012)

PaaS on erityisesti kehittäjille suunnattu pilvipalvelumalli, jossa palveluntarjoaja tarjoaa alustan, jolla kehittäjät voivat kehittää, testata ja ottaa käyttöön sovelluksia. PaaS-palveluntarjoajat tarjoavat valikoiman kehitystyökaluja, tietokantoja ja väliohjelmistoja sekä taustalla olevan infrastruktuurin sovellusten suorittamista varten. PaaS antaa organisaatioille mahdollisuuden keskittyä sovellusten kehittämiseen ja käyttöönottoon taustalla olevan infrastruktuurin rakentamisen ja ylläpitämisen sijaan. (Gibson ym. 2012)

SaaS on pilvipalvelumalli, jossa palveluntarjoaja toimittaa sovelluksia Internetin kautta. SaaS-palveluntarjoajat tarjoavat erilaisia ohjelmistosovelluksia, kuten vaikkapa sähköpostin tai asiakkuudenhallintajärjestelmiä. SaaS antaa organisaatioille mahdollisuuden välttää korkeat kustannukset, jotka aiheutuvat ohjelmistojen ostamisesta ja asentamisesta omaan infrastruktuuriinsa, sekä jatkuvat ylläpito- ja tukikustannukset, jotka liittyvät paikallisiin ohjelmistoihin, koska palvelut ovat saatavilla verkon ylitse. (Gibson ym. 2012)

Pilvipalvelut ovat tekniikoita, joiden avulla käyttäjät voivat käyttää resurssejaan Internetin kautta omien palveluiden perustamisen ja ylläpitämisen sijaan. Pilvipalvelumallit ovat kustannustehokkaita, skaalautuvia, joustavia ja hyvin saavutettavia. Digitaaliselle aikakaudelle siirtyessä on selvää, että pilvipalveluilla on jatkossa tärkeä rooli yritysteknologian tulevaisuuden muovaamisessa.

2.3 Salesforce

Salesforce on yhdysvaltalainen pilvipalveluiden tarjoaja ja maailman johtava asiakkuudenhallintajärjestelmien tarjoaja (Gupta, Verma ja Janjua 2018; Patel ja Chouhan 2016; Harris 2022, s. 1). Salesforcen merkittävyys piilee sen kyvyssä keskittää asiakastiedot, virtaviivaistaa liiketoimintaprosesseja ja edistää yrityksen liiketoimintahenkilöitä tekemään päätöksiä tehokkaasti helposti saatavan datan ja analytiikan pohjalta (Gupta ja Arora 2013, s. 67).

Salesforce tarjosi alun perin vain ohjelmistoa palveluna (SaaS) lippulaivatuotteenaan Sales Cloud, mutta on sittemmin laajentanut tarjontaansa kattamaan alustaa palveluna (PaaS)

mukautettujen sovellusten kehittämiseen ja infrastruktuuria palveluna (IaaS) sovellusten ajamiseen skaalautuvassa ja suojatussa pilvi-infrastruktuurissa. CRM-järjestelmien kasvavan kysynnän myötä Salesforce on noussut johtavaan asemaan CRM-järjestelmien tarjoajana ja panostanut teknologioidensa ajantasaisuuteen ja kehitykseen. (Patel ja Chouhan 2016)

Salesforce tarjoaa lukuisia ominaisuuksia, joista on hyötyä ohjelmistosovellusten ja -tuotteiden nopealle kehitykselle, ja joiden myötä Salesforcesta on tullut yhä suositumpi valinta yrityksille, jotka haluavat hyödyntää pilviteknologiaa toiminnan ja asiakkuuksien hallintaprosessien tehostamiseen (Patel ja Chouhan 2016). Kuten Patel ja Chouhan (2016) tutkimuksessaan toteavat, kaikki Salesforce-ohjelmistotuotteet toimivat pilvessä, mikä johtaa siihen, että Salesforcen tarjoamien ohjelmistotuotteiden saatavuuden, mukautettavuuden, ylläpidon ja kehityksen helppous on yrityksille houkuttelevaa CRM-järjestelmää valittaessa.

Salesforcen tarjoama alusta on suunniteltu helppokäyttöiseksi ja hyvin muokattavaksi, joten se sopii hyvin kaikenkokoisille yrityksille (Kiranmayee 2015). Salesforce tarjoaa yrityksille kattavan joukon työkaluja, kuten myynti, palvelu, automaatiot ja analytiikka, asiakasvuorovaikutusten hallintaan sekä seurantaan. Salesforce tarjoaa myös Salesforce-yhteisölle AppExchange-kauppapaikan Salesforce-kumppaneiden ja kolmansien osapuolien rakentamien lisäominaisuuksien julkaisemista ja lataamista varten (Gupta ja Arora 2013, s. 37–38). Nämä paketeiksi kutsutut CRM-järjestelmän toiminnallisuuksia lisäävät ominaisuudet voivat olla joko hallittuja (engl. managed) tai avoimia (engl. unmanaged). Hallittujen pakettien lähdekoodia ei paketin lataaja näe tai pysty muokkaamaan, toisin kuin avoimissa paketeissa, joiden lähdekoodia pystyy muokkaamaan. Avoimet paketit ovat avoimen lähdekoodin sovellusprojekteja. (Fawcett 2017, s.5–6)

Salesforcen täysin pilvipohjainen arkkitehtuuri tarjoaa käyttäjilleen lukuisia etuja. Ensinnäkin se mahdollistaa helpon pääsyn tietoihin mistä tahansa, jopa matkapuhelimella, kunhan käyttäjällä on Internet-yhteys, jolloin käyttäjät voivat työskennellä etänä ja tehdä yhteistyötä tehokkaasti (Gibson ym. 2012). Toiseksi se eliminoi organisaation monimutkaisen ja kalliin itsehallinnoitavan IT-infrastruktuurin tarpeen, sillä Salesforce huolehtii kaikesta ylläpidosta ja päivityksestä. Kolmanneksi se tarjoaa automaattisia ja saumattomia päivityksiä ja varmuuskopioita, mikä varmistaa, että käyttäjillä on aina pääsy uusimpiin ominaisuuksiin ja heidän tietonsa on suojattu. Neljänneksi se tarjoaa korkeatasoista tietoturvaa ja tietosuo-

jaa salauksella, ympärivuorokautisella monitoroinnilla, kenttä- ja objektikohtaisella turvallisuudella sekä monivaiheisen tunnistautumisen ja tiukan pääsynvalvonnan avulla (Patel ja Chouhan 2017). Lopuksi se tarjoaa joustavia tilaushinnoittelumalleja, joiden avulla käyttäjät voivat skaalata ylös tai alas tarpeen mukaan ja maksaa vain käyttämistään palveluista ja ohjelmistotuotteista.

Salesforce on moniasiakkainen (engl. multitenant) alusta. Se käyttää yhtä reserviä laskentaresursseja palvelemaan usean asiakkaan tarpeita. Salesforce suojaa organisaation tietoja kaikilta muilta asiakasorganisaatioilta käyttämällä yksilöllistä tunnistetta (jossa on organisaation yksilöllinen Id & User -istunto), joka liitetään jokaisen käyttäjän istuntoon. Tapahtumavalvonnan (engl. event monitoring) avulla voi seurata käyttäjien aktiivisuutta ja tehdä vianmäärityksiä. (Soni ja Vala 2017)

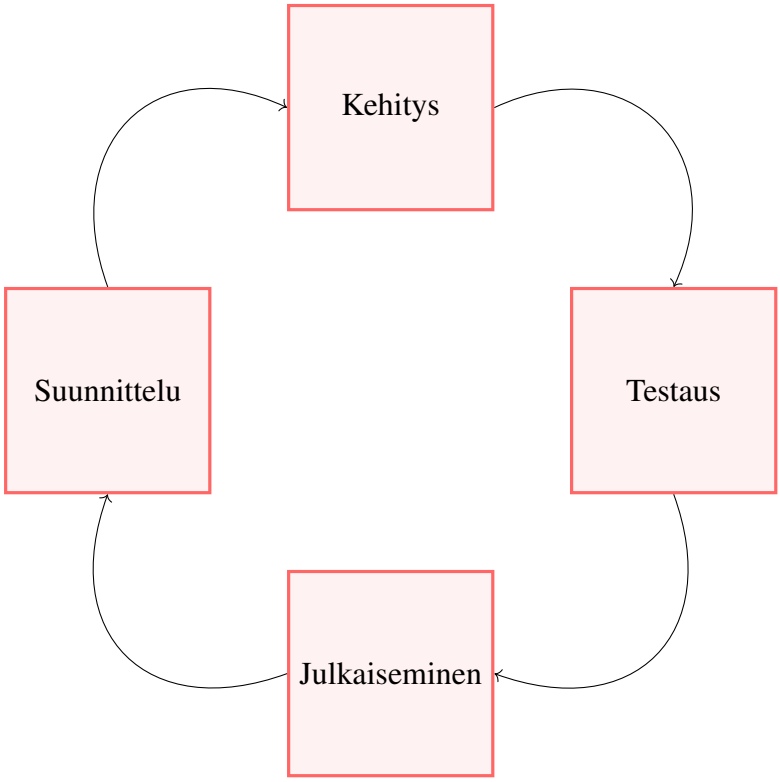
Salesforcesta on tullut yksi johtavista pilvipohjaisten CRM-ratkaisujen toimittajista laajoine valikoimine tuotteita ja palveluita kaikenkokoisille yrityksille. Sen joustavat tilaushinnoittelumallit, skaalautuva infrastruktuuri sekä tehokkaat räätälöinti- ja automaatiotyökalut ovat tehneet siitä suositun valinnan organisaatioille, jotka haluavat tehostaa toimintaansa ja parantaa asiakkaiden sitoutumista. Salesforce keskittyy innovaatioihin, saavutettavuuteen ja käyttäjäystävällisiin käyttöliittymiin, joten se on valmis jatkamaan johtavaa tietä pilvipalveluiden ja digitaalisten muutosten alalla.

2.3.1 Salesforce-kehitys

Salesforce-kehitys on merkittävä osa organisaatioiden Salesforce-alustan toiminnallisuuksien mukauttamista ja laajentamista. Salesforce (2023b) tarjoaa kehittäjille deklaratiivisia ja ohjelmallisia lähestymistapoja mukautettujen toteutusratkaisuiden luomiseen. Salesforce-kehitys on täysin pilvipohjaista tarkoittaen, että ohjelmakoodi on otettava käyttöön (engl. deploy) Salesforce-kehitysympäristöön ennen sen suorittamista (Gupta ja Arora 2013, s.133) toisin, kuin perinteisessä ohjelmistokehityksessä, jossa usein ohjelmakoodi suoritetaan paikallisella tietokoneella integroidulla kehitysympäristöllä (engl. integrated development environment; lyh. IDE). Kestävien ja skaalautuvien toteutusratkaisujen kehittäminen Salesforce-alustalla vaatii perusymmärryksen Salesforcen käyttämistä teknologioista sekä niiden roo-

leista osana Salesforce-ekosysteemiä.

Sovellusten elinkaarenhallinta (engl. Application Lifecycle Management; lyh. ALM) on perinteisten ketterien menetelmien kaltainen kattava iteratiivinen lähestymistapa Salesforce-ohjelmistokehitysprosessin hallintaan tavoitteena parantaa ohjelmiston laatua. Sovellusten elinkaarenhallinnassa on neljä vaihetta, jotka on esitettynä kuviossa 1: suunnittelu, kehitys, testaus ja käyttöönotto. (Patel ja Chouhan 2016) Ohjelmiston laatu on suoraan sidoksissa jäsennetyn ohjelmistokehitysprosessin lähestymistapaan (Seth ym. 2015).



Kuvio 1. Iteratiivinen sovelluksen elinkaaren hallinta.

Salesforcessa on kaksi päätapaa kehittämiseen: deklaratiivinen ja ohjelmallinen. Deklaratiiviseen kehittämiseen kuuluu ”osoita ja napsauta” -työkaluja sovellusten ja automaatioiden rakentamiseen ilman ohjemallisen koodin tarvetta. Tätä lähestymistapaa Salesforce (2023b) kutsuu ”Clicks, Not Code” -kehitykseksi. Deklaratiivisen kehityksen avulla käyttäjät voivat luoda kevyitä mukautettuja sovelluksia ja automaatioita ilman merkittävää ohjelmointiosaamista tai asiantuntemusta käyttämällä graafisia ohjelmointityökaluja, kuten Flow Builderia Salesforcessa (Harris 2022, s. 1–2).

Ohjelmalliseen kehitykseen Salesforce tarjoaa muun muassa vahvasti tyypitetyn, Javan kaltaisen palvelinpuolen Apex-ohjelmointikielen (Salesforce 2023h) ja LWC-ohjelmointiviitekehyksen JavaScriptille käyttöliittymäpuolen kehitykseen (Salesforce 2023c). Tämä lähestymistapa tarjoaa deklaratiivista kehitystä kehittyneempiä mukautusvaihtoehtoja ja paremman hallinnan sovelluksen toimivuuteen, mutta se vaatii ohjelmointitaitoja ja -asiantuntemusta (Salesforce 2023b).

Yleisesti ottaen deklaratiivinen kehitys on nopeampaa ja helpompaa yksinkertaisissa käyttötapauksissa, kun taas ohjelmallinen kehitys on tehokkaampaa ja joustavampaa monimutkaisempiin käyttötapauksiin. Huomattavana erona deklaratiivisen ja ohjelmallisen kehityksen välillä on, ettei ohjelmallista koodia voi muokata tuotantoympäristöissä toisin kuin deklaratiivisia koodeja (Gupta ja Arora 2013, s. 99). Näiden kahden lähestymistavan välisten erojen ymmärtäminen on olennaista Salesforce-kehityksen onnistumiselle. Tässä tutkimuksessa perehdytään ohjelmallisen kehityksen näkökulmaan.

Ohjelmallisessa Salesforce-kehityksessä hyödynnetään ohjelmointikielten ja -kehysten tuntemusta luodakseen mukautettuja toimintoja ja sovelluksia, jotka vastaavat tiettyjä liiketoiminnan tarpeita. Apex, Salesforcen käyttämä ensisijainen palvelinpuolen ohjelmointikieli, tarjoaa mahdollisuuden kirjoittaa mukautettua liiketoimintalogiikkaa, kuten prosessiautomaatioita, integraatioita ja palvelinpuolen ohjaimia. Lisäksi Lightning Web Component -ohjelmointiviitekehys (LWC) mahdollistaa dynaamisten ja responsiivisten verkkokomponenttien luomisen, joita voidaan käyttää käyttöliittymien rakentamiseen uudelleenkäytettävistä komponenteista (Mazalon 2022). Salesforce Object Query Language (SOQL) on Structured Query Languagen (SQL) kaltainen kieli, tarjoaa kehittäjille mahdollisuuden suorittaa kyselyitä Salesforce-tietokannassa, ja DML-toimintojen (Create, Read, Update and Delete; lyh. CRUD) avulla voi manipuloida tietokantaan tallennettuja tietueita. Näiden tekniikoiden yhdistelmän avulla kehittäjät voivat luoda räätälöityjä sovelluksia ja työnkulkuja, jotka on sovitettu kyseessä olevan organisaation vaatimusten mukaan. (Gupta ja Arora 2013, s. 133–165)

Salesforce-kehitys tarjoaa tehokkaan alustan sovellusten, automaatioiden ja integraatioiden luomiseen ja mukauttamiseen. Alusta tarjoaa laajan valikoiman työkaluja ja tekniikoita sekä deklaratiiviseen että ohjelmalliseen kehittämiseen, joista tärkeimpinä LWC, Apex ja SOQL.

Salesforce-kehitysprosessia helpottaa ALM-metodologian käyttö, jonka avulla kehittäjät voivat suunnitella, kehittää, testata ja julkaista ohjelmakoodiaan tehokkaammin ja johdonmukaisemmin. Salesforce-alustan nopean kasvun myötä kehittäjät voivat luoda innovatiivisia ja skaalautuvia ratkaisuja, jotka auttavat organisaatioita saavuttamaan liiketoimintatavoitteensa.

2.3.2 Salesforce-testaus

Ohjelmistojen laatu korostuu erityisesti Salesforcen kaltaisissa monimutkaisissa järjestelmissä. Organisaatioiden pyrkiessä optimoimaan prosessejaan ja parantamaan näin kilpailuetuaan on tärkeää ymmärtää ohjelmistotestauksen kriittinen rooli osana ohjelmistoprojektin toteutusta ohjelmiston laatutekijöiden varmistamisessa. Testaus varmistaa laadukkaan ohjelmiston toimituksen asiakkaalle, mikä on kriittistä asiakastyytyväisyyden ja sitä myöden myös pitkien asiakassuhteiden sekä tuoton kehittämiseksi (Tarasi 2007; Mithas, Krishnan ja Fornell 2005).

Salesforce tarjoaa kattavat testausominaisuudet, joiden avulla voidaan varmistaa Salesforce-järjestelmän laatua (Mathew ja Spraetz 2009). Salesforce sisältää Apex-testauskehyksen, joka mahdollistaa Apex-kielellä kirjoitettujen testien suorittamisen suoraan Salesforce-ympäristössä (Mathew ja Spraetz 2009). Testauskehyksen lisäksi Salesforce tarjoaa testausominaisuudet, kuten erilaiset ympäristöt, jotka on esitetty taulukossa 1, ja DevOps Centerin (Salesforce 2022).

Salesforcessa on sisäänrakennettu Apex-testien suorittaminen (engl. Apex Test Execution), joka mahdollistaa yksikkötestien kirjoittamisen ja suorittamisen Apex-luokille ja -käynnistimille. Apex-testauskehys mahdollistaa ohjelmakoodin testaamisen usealla eri tasolla (Harris 2022, s. 191 ja s. 195). Näitä ovat muun muassa yksikkötestaus, jolla keskitytään yksittäisten koodiyksiköiden validointiin; integraatiotestaus, jolla varmistetaan, että eri koodiyksiköt toimivat yhteen toivotulla tavalla; ja järjestelmätestaus, jolla varmistetaan järjestelmän yleinen toiminnallisuus. Apex-testauskehystä ei kuitenkaan voida käyttää käyttöliittymien testaamiseen, sillä testauskehys mahdollistaa vain palvelinpuolen ohjelmakoodin testaamisen (Mathew ja Spraetz 2009).

Yksi Apex-testauskehyksen tärkeimmistä ominaisuuksista on kyky eristää testit valitun ympäristön tiedoista testien suorittamista varten. Näin varmistetaan, että DML-toiminnot eivät vaikuta tietokantaan testauksen aikana. Osa ympäristön tiedoista on saavutettavissa (vain luettavissa) testiajon aikana testitarkoituksiin Apexin Test-luokan metodeilla (Salesforce 2023f), esimerkiksi:

```
Id pricebookId = Test.getStandardPricebookId();
```

Nämä tapaukset ovat hyödyllisiä, kun testidatan kirjoittaminen vie aikaa, esimerkiksi tässä haetaan vakiohintakirjan (engl. Standard Pricebook) tunnus (ID) myöhempää käyttöä varten testiajon aikana.

Kyky automatisoida testitapauksien hallinta, suorittaminen ja raportointi Apex-testauskehyksellä vapauttaa kehittäjät keskittymään testitapauksien suunnitteluun ja toteuttamiseen automaatiotestausinfrastuktuurin ylläpitämisen sijaan (Mathew ja Spraetz 2009). Apex-testauskehyksen tuominen osaksi Salesforcea on mahdollistanut Salesforcen asettaa tiettyjä vaatimuksia ohjelmakoodin testaamisen osalta, kuten esimerkiksi kaikelle tuotantoympäristöön vietävälle palvelinpuolen ohjelmakoodille tulee olla vähintään 75 %:n koodirivikattavuus testeillä (Salesforce 2023g; Harris 2022, s. 325).

Testaus on kriittinen osa Salesforce-kehitystä ohjelmistojen korkean laadun varmistamisessa ja ohjelmistovirheisiin liittyvien riskien vähentämisessä. Salesforce-ohjelmiston laadun varmistamiseksi Salesforce vaatii Apex-luokille kirjoitetuilta testeiltä vähintään 75 %:n koodirivikattavuuden. Salesforce-testausta helpottaa muun muassa testidatan eristyneisyys oikeasta datasta sekä Salesforcen tarjoamat työkalut, kuten DevOps Center testiautomaatioille ja helpolle ohjelmistoversion julkaisemiselle. Hyödyntämällä Salesforcen tarjoamaa testauskehystä ja noudattamalla pilvitestauksen parhaita käytäntöjä voi varmistaa, että ohjelmakoodi on korkealaatuista.

2.3.3 Salesforce-kehitykseen tarjolla olevia työkaluja

Salesforce-kehityksessä keskeisimpiä asioita on kehittämisen tehokkuus (Patel ja Chouhan 2016). Salesforce-kehitykseen ja Salesforce-järjestelmien testaukseen on tarjolla paljon tehokkuutta, nopeutta, automatisointia ja kustannustehokkuutta parantavia Salesforcen omia

ja kolmansien osapuolien tekemiä työkaluja. Tässä aliluvussa mainitaan niistä muutama testauksen kannalta oleellinen työkalu.

Salesforce tarjoaa erilaisia taulukossa 1 esitettyjä ympäristöjä sovellusten kehittämiseen, testaamiseen ja käyttöönottoon, niistä jokaisella on omat ominaisuutensa ja tarkoituksensa. Scratch-orgit ovat lyhytikäisiä ympäristöjä (1–30 pv), jotka on luotu tiettyä kehitysprojektia tai tehtävää varten. Niitä voidaan luoda ja poistaa ohjelmallisesti, ja niitä käytetään uusien ominaisuuksien tai mukautusten kehittämiseen ja testaamiseen. (Salesforce 2023e)

Tyyppi	Scratch	Sandbox				Production
Ympäristö	Scratch org	Dev	Dev Pro	Partial	Full	Tuotanto
Virkistysaika	-	1 pv	1 pv	5 pv	29 pv	-
Datamuisti	200 MB	200 MB	1 GB	5 GB	Tuotanto	Riippuu versiosta
Tiedostomuisti	50 MB	200 MB	1 GB	Tuotanto	Tuotanto	Riippuu versiosta
Mallipohja	-	-	-	Vaadittu	Saatavilla	-

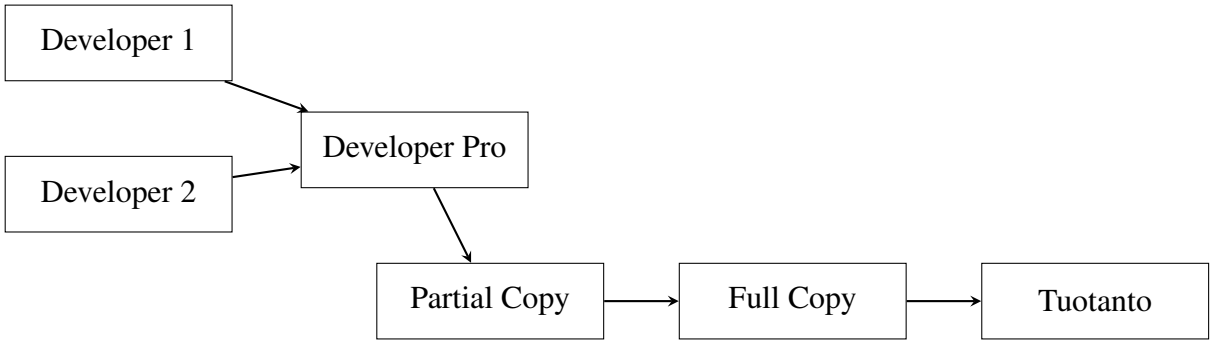
Taulukko 1. Salesforce-ympäristöt (Salesforce 2023d, 2023e).

Developer-Sandboxit ovat pitkäikäisiä ympäristöjä, joissa voi luoda, muokata ja testata mukautuksia. Näitä ympäristöjä käytetään tyypillisesti ohjelmointiin ja yksikkötestaukseen. Developer Pro -Sandboxit ovat samanlaisia kuin Developer-Sandboxit, mutta ne mahdollistavat suuremman datamäärän käsittelyn sekä tukevat suurempaa määrää kehittäjiä. Näitä ympäristöjä käytetään tyypillisesti integraatiotestaukseen, toiminnalliseen testaukseen ja käyttäjien hyväksymistestaukseen. Partial Copy -Sandboxit ovat vaaditun mallipohjan avulla alustettuja tuotantoympäristön kopioita, jotka sisältävät osan tuotantotiedoista. Näitä ympäristöjä käytetään tyypillisesti suorituskyvyn testaamiseen, käyttäjien hyväksymistestaukseen ja vaiheistukseen (engl. staging). Full Copy -Sandboxit, ovat saatavilla olevan mallipohjan avulla tehtyjä täydellisiä kopioita tuotantoympäristöstä, mukaan lukien kaikki tietueet objekteista ja tuotantoympäristön metatiedot. Näitä ympäristöjä käytetään yleensä lopputestaukseen ennen muutosten käyttöönottoa tuotannossa. (Salesforce 2023d; Coxon 2022; Gupta ja Arora 2013, s. 98–106)

Salesforce Sandbox-mallipohjat (engl. Sandbox template) tarjoavat kätevän tavan luoda ja päivittää Sandboxeja ennalta määritellyllä tuotantoympäristöstä saatavalla datalla. Sandbox-

mallipohjilla voi helposti alustaa Partial Copy tai Full Copy -ympäristöön laadukasta dataa muun muassa testaus- ja koulutustarkoitukseen ilman manuaalista tai ohjelmallista tietueiden luontia. (Salesforce 2023d, 2023d)

Tyypillisesti Salesforce-kehitysputkeen sisällytetään yksi tai useampi Developer-Sandbox ohjelmointia ja yksikkötestausta varten. Näiden lisäksi Salesforce-kehitysputkeen saatetaan sisällyttää Developer Pro -Sandbox integraatiotestausta ja käyttäjien hyväksymistestausta varten. Partial Copy ja Full Copy -Sandboxeja käytetään suorituskyvyn testaamiseen ja vaiheistukseen käyttäen tuotantoympäristön oikeaa dataa (Coxon 2022). kuviossa 2 esitetään esimerkki mahdollisesta Salesforce-kehityksessä käytettävästä kehitysputkesta. Kehitysputkea apuna käyttäen voi varmistaa, että kaikki muutokset testataan perusteellisesti ennen kuin ne otetaan käyttöön tuotantoympäristössä ja tuotantoon vienti testataan vähintään kertaalleen. Näin minimoidaan tuotantoympäristön riskit käyttöhäiriöille, virheille ja ongelmille. (Arachchi ja Perera 2018)



Kuvio 2. Esimerkki yleisestä Salesforce-kehityksessä käytetystä kehitysputkesta (Bassett 2022; Gupta ja Arora 2013, s. 100).

Salesforce julkaisi kesällä 2022 DevOps Centerin eli Salesforce-kehitykseen erikoistuneen modernin tavan versioiden julkaisemiseen ja yksikkötestien automaattiseen ajamiseen. DevOps Centeristä tuli yleisesti saatavilla oleva toiminnallisuus joulukuussa 2022. (Salesforce 2022) Kirjassaan Harris (2022, s. 534) mainitsee, että sen tarkoituksena on tuoda kehitystiimit lähemmäksi toisiaan antamalla mahdollisuuden työskennellä yhdellä kokoonpano- ja koodisarjalla ja soveltaa DevOpsin parhaita käytäntöjä sekä tehostaa kehitysputken toimintaa. Alusta korvaa muutosjoukot ja tarjoaa helposti ymmärrettävän deklaratiivisen käyttöliittymän käyttöönottoputkien hallintaan metatietojen hallintalähteen ohjauksella GitHubis-

sa. DevOps Center tukee julkaisuja (engl. release), käyttäjätarinoita, putkia (engl. pipeline), työkohteita (engl. work item) ja versionhallintaa. DevOps Centerillä on vielä joitain rajoituksia, kuten muiden versionhallintajärjestelmien käytön, pakettipohjaisen kehityksen (DX) ja kolmansien osapuolien integraatioiden tuen puute. (Gotts 2022)

Salesforce DX (lyh. SFDX) on Salesforcen julkaisema joukko työkaluja, joiden avulla voi hallita ohjelmakoodia ja metatietoja versio-ohjatulla tavalla. Lisäksi se helpottaa automaattista testausprosessia. Työkalut on suunniteltu toimimaan yhteen ja niillä on yhteinen tavoite helpottaa pakettipohjaista, lähdevetoista kehitystä (engl. source-driven development) ja julkaisuprosessia. (Runciman 2022) Ainoa vaadittu työkalu SFDX:n avulla kehittämiseen on komentorivikäyttöliittymä Salesforce CLI. Tehokkuuden kannalta on suositeltavaa käyttää integroitua kehitysympäristöä (IDE). (Harris 2022, s. 7–10)

Visual Studio Code (VS Code) on suosittu avoimen lähdekoodin integroitu kehitysympäristö (IDE), joka on saanut laajan suosion ohjelmistokehitysyhteisössä. Salesforce-kehitystä pystyy Salesforcen oman kehityskonsolin lisäksi tekemään VS Codessa SFDX:n avulla. Tämä mahdollistaa erilaisten lisäosien ja laajennosten käyttämisen ohjelmointiympäristössä. (Cook 2022; Salesforce 2023a; Harris 2022, s. 7 ja s. 21–25) Erityisen isona etuna Salesforcen omaan kehityskonsoliin nähden on VS Coden saumaton toimivuus versionhallinnan kanssa. Tämä on Salesforce-kehittäjille yksi vankimpia perusteita käyttää VS Codea koodieditorina (Harris 2022, s. 7).

Salesforce-testauksessa on tavallista luoda mallidataa yksikkö-, integraatio- ja end–to–end-testausta varten. (Harris 2022, s. 325) Tämän nopeuttamista ja tehostamista varten on kehitetty kolmannen osapuolen toimesta erilaisia testidatatehtaita, kuten Apex Test Data Factory. Niiden toiminta perustuu sObjektien tuottamiseen ja lisäämiseen eri testitapauksia varten.

Selenium on laajalti käytetty avoimen lähdekoodin työkalu verkkoselaimien automatisointiin. Se mahdollistaa automaattisten testien luomisen käyttäjien vuorovaikutusten simuloimiseksi verkkosovellusten kanssa, mukaan lukien verkkokomponenttien toimivuuden testaaminen. Salesforce-testaamisessa Seleniumia voidaan käyttää LWC-komponenttien testaamiseen suorittamalla automaattisia testejä, jotka simuloivat käyttäjien vuorovaikutusta komponenttien kanssa. Tämä mahdollistaa LWC-komponenttien perusteellisen testaamisen eri

skenaarioissa ja voi auttaa tunnistamaan mahdolliset viat tai ongelmat, joita saattaa ilmetä. Seleniumin avulla testejä voidaan suorittaa useilla selaimilla ja alustoilla, mikä varmistaa, että LWC-komponentit toimivat oikein eri ympäristöissä. Lisäksi Selenium tarjoaa vankat raportointiominaisuudet, joiden avulla voi tarkastella testien tuloksia ja tunnistaa ongelmat. Kaiken kaikkiaan Selenium on tehokas työkalu LWC-komponenttien testaamiseen ja voi parantaa huomattavasti näiden komponenttien laatua ja luotettavuutta. (Neumann 2020)

Workbench on Salesforce-kehityksen avuksi tehty web-pohjainen työkalupaketti, joka on suunniteltu pääkäyttäjille ja kehittäjille. Sillä voi muun muassa massana suorittaa DML-operaatioita, suorittaa tietokantakyselyitä sekä käyttöönottaa ja debugata sovelluksia. (Foundation 2005; O'Leary 2022) Workbenchin kattava Salesforce REST- ja SOAP-sovellusliittymien työkalusarja sekä sen tarjoama mahdollisuus jäljittää suorituspolkuja ja analysoida suorituskykymittareita tekevät siitä arvokkaan työkalun Salesforce-sovellusten vianmääritykseen ja ongelmien ratkaisemiseen, mikä helpottaa Salesforce-ohjelmiston elinkaarenhallintaa (Harris 2022, s. 158 ja s. 327).

Salesforce-kehittämiseen on saatavilla useita jatkuvasti kehittyviä työkaluja tehokkuuden ja yhteistyön parantamiseksi sekä kehityksen elinkaaren hallintaan. Hyödyntämällä muun muassa tässä aliluvussa käsiteltyjä työkaluja ohjelmiston elinkaarenhallina ja testaaminen sujuvoituu.

2.4 CMMi ja TMMi

Capability Maturity Model (lyh. CMM) on yksi tunnetuimmista ohjelmistokehitysprosessin arviointi- ja kehitysmalleista (Paulk ym. 1993; O'Regan 2011). Nykyisin CMM arviointi- ja kehitysmalli on saanut väistyä uudemman ja kattavamman Capability Maturity Model integration (lyh. CMMi) mallin tieltä. CMMi-malli korvaa vanhan CMM arviointi- ja kehitysmallin laajentamalla arviointi- ja kehitystoimet kattamaan koko organisaatiota ja sen eri prosesseja pelkän ohjelmistokehitysprosessin sijasta (O'Regan 2011). Sekä jo CMM:n että myöhemmin CMMi:n tekijöiden pyrkimyksenä on ollut parantaa ohjelmistojen laatua parantamalla niiden tekoprosessia (O'Regan 2011; Paulk ym. 1993).

CMMi arviointi- ja kehitysmalli käsittelee ohjelmistotuotantoa yleisellä tasolla, mutta ei pa-

neudu mihinkään yksittäiseen osa-alueeseen erityisen tarkasti. Siitä johtuen TMMi Foundation on kehittänyt CMMi-mallia ohjelmistotestauksen osalta täydentävän mallin Test Maturity Model integration (lyh. TMMi). TMMi arviointi- ja kehitysmalli keskittyy koko organisaation ja sen eri prosessien sijaan erityisesti ohjelmistotestauksen arviointiin ja kehittämiseen. (TMMi Foundation 2023; Veenendaal, Garousi ja Felderer 2022)

TMMi-malli koostuu viidestä eri maturiteettitasosta, joilla organisaation testaustoiminta voi olla. Maturiteettitaso määrittelee, mitä prosesseja kehitetään, mitkä ovat tavoitteet ja mitä käytänteitä kuhunkin tasoon liittyy. (Veenendaal, Garousi ja Felderer 2022; TMMi Foundation 2023)

TMMi-mallin maturiteettitasot ovat seuraavat (TMMi Foundation 2023):

Taso 1, *Aloitus*: Organisaation testaustoiminta on hallitsematonta, eikä testausta tehdä järjestelmällisesti. Testaus on useimmiten yksittäisten henkilöiden harteilla, eikä suoritetut testaustoimenpiteet ole toistettavissa.

Taso 2, *Hallittu*: Organisaation testaustoiminta on vakiintunutta ja hallittua. Organisaatiolla on olemassa yhtenäiset testauskäytänteet ja testausstrategia. Testaus saattaa vaihdella projektien välillä, mutta jokaisella projektilla on vähintään erillinen testausympäristö.

Taso 3, *Määritelty*: Testaus on integroituna osaksi kaikkia ohjelmistokehityksen vaiheita aina määrittelystä käyttöönottoon siten, että organisaation jokainen projekti noudattaa samoja testauskäytänteitä ja -tapoja. Näiden lisäksi projekteissa suunnitellaan ja toteutetaan eitoiminnallista testausta ja ohjelmakoodin vertaisarviointeja.

Taso 4, *Mitattu*: Projektin testaustoiminta on mitattavaa ja mittauksen tulokset osoittavat, että toteutuksessa on mahdollisimman vähän virheitä. Kattavat vertaisarvioinnit ovat käytössä kaikissa projekteissa.

Taso 5, *Optimointi*: Organisaatio aktiivisesti kehittää ja optimoi jokaista aktiviteettia ja osa-aluetta, jotka esiintyvät aiemmilla tasoilla vähentääkseen toteutuksissa esiintyviä virheitä ja parantaakseen toteutuksien laatua.

TMMi-mallin avulla organisaatiot pyrkivät kehittämään testaustoimintaansa parantaakseen

toteutettujen ohjelmistojen laatua. Veenendaalenin, Garousin ja Feldererin (2022) tekemän kyselyn mukaan, suurin osa vastaajista koki TMMi-mallin käyttöönoton myötä onnistuneensa parantamaan toteutettujen ohjelmistojen laatua ja kehittämään muita ohjelmistotestaukseen liittyviä osa-aluita, kuten parantamaan testauksen tehokkuuta ja vähentämään virheiden määrää ohjelmistoissa.

CMMi ja TMMi ovat hyviä malleja organisaatioille kehittämään kokonaisvaltaisesti ohjelmistokehitystä ja ohjelmistotestaamista. Molemmat mallit ovat laajalti käytettyjä ja niihin on tarjolla virallisia sertifioitumisia, joissa ulkopuolinen toimija arvioi, kuinka hyvin malli on toteutettu organisaatiossa. Ilman virallista sertifioitumista mallien pyrkimyksenä on parantaa organisaation tuottamien ohjelmistojen laatua.

3 Tutkimusasetelma

Tutkimuksen tutkimusmenetelmänä käytetään suunnittelutiedettä (engl. design science research) (Hevner ym. 2004; Peffers ym. 2007). Tutkimus tuottaa artefaktin eli testausstrategian, jolla pyritään kehittämään ja yhtenäistämään kohdeorganisaation ohjelmistotestauskäytänteitä. Tutkimus analysoi ja arvioi millaisia erilaisia testauskäytänteitä on järkevää ja kohdeorganisaation kannalta mielekästä käyttää Salesforce-kehityksessä. Tutkimuksen kohdeorganisaatiolla ei ole ennestään ollut testausstrategiaa eikä yhtenäisiä testauskäytänteitä.

Ensimmäisessä aliluvussa 3.1 esitetään yleisesti kohdeorganisaatio ja kohdeorganisaation tekemän testauksen tilaa ennen testausstrategian luontia. Toisessa aliluvussa 3.2 esitetään tutkielman tutkimuskysymykset. Viimeisessä aliluvussa 3.3 esitetään miten tutkimusmenetelmää käytetään tutkimuksessa.

3.1 Tutkimuskonteksti

Tutkimus tehtiin toimeksiantona kohdeorganisaatiolle. Kohdeorganisaatio on suomalainen pieni ohjelmistoyritys, joka on keskittynyt Salesforce-konsultointiin ja kehitykseen. Kohdeorganisaatio toteuttaa asiakkailleen pääasiassa Salesforce-järjestelmän käyttöönotto- ja jatkokehitysprojekteja sekä ylläpitoa. Kohdeorganisaation asiakkaisiin kuuluu useilla eri toimialalla toimivia yrityksiä. Yrityksien koot vaihtelevat pienistä muutaman hengen yrityksistä aina kansainvälisiin pörssiyhtiöihin.

Kohdeorganisaation Salesforce-järjestelmän käyttöönotto- ja jatkokehitysprojektit sisältävät projektista riippuen useita eri työtehtäviä, kuten Salesforce-järjestelmän konfigurointia, kustomoitujen sovelluksien ja käyttöliittymäkomponenttien ohjelmointia, asiakkaan liiketoiminnan vaatimien automaatioiden ohjelmointia ja kustomoitujen integraatioiden ohjelmointia. Yleisiä ylläpidon tehtäviä ovat muun muassa asiakkaan Salesforce-pääkäyttäjän tukeminen esimerkiksi käyttöoikeuksien määrittelemisen kanssa, pienkehitys ja pienet lisäkonfiguroinnit.

Kohdeorganisaatiossa asiakkaalle käyttöönottoa tai jatkokehitystä tekevä projektiryhmä on

jaettuna rooleihin. Rooleja ovat muun muassa projektipäällikkö, arkkitehti, kehittäjä, konsultti ja myyjä. Projektiryhmän jäsenellä voi olla samanaikaisesti samassa projektissa yksi tai useampi rooli. Projektiryhmässä voi olla useita jäseniä, joilla on sama rooli. Kohdeorganisaation pienestä koosta ja pienistä projektiryhmistä johtuen roolit ja niiden vastuut ja työtehtävät risteävät usein, eivätkä täten ole täysin muuttumattomia.

Kohdeorganisaatiolla ei ole aiemmin ollut testausstrategiaa tai yhtenäisiä testauskäytänteitä, joihin projektiryhmän jäsenet voisivat tukeutua, kun he määrittelevät, suunnittelevat, toteuttavat ja testaavat Salesforce-järjestelmän ominaisuuksia ja konfigurointeja. Testausstrategian puuttuminen on kohdeorganisaatiossa johtanut siihen, että jokainen projekti testataan eri tavalla ja laajuudella projektiryhmästä riippuen. Organisaation testaustoiminnan voidaan täten nähdä olevan TMMi-mallin tasolla 1 (TMMi Foundation 2023).

Vaikka Salesforce vaatii tuotantoympäristöön vietävältä ohjelmakoodilta 75 %:n koodirivikattavuuden testeillä, on se kohdeorganisaatiossa tarkoittanut sitä, että projektin viime metreillä ohjelmoidaan väkisin testit, jotta vaadittu koodirivikattavuus saadaan täytettyä ja ohjelmakoodit siirrettyä kehitysympäristöstä tuotantoympäristöön. Kohdeorganisaatiossa testaamisen on nähty olevan ennemmin pakollinen suoritus, kuin tapa parantaa toteutettujen Salesforce-järjestelmien laatua.

Kohdeorganisaatiossa on havaittu, että vaadittavan koodirivikattavuuden saavuttamisen lisäksi avainhaasteita Salesforce-kehityksessä ja -testauksessa ovat erityisesti dokumentaation puuttuminen, testidatan hallinta, integraatioiden testaus, sovellusrajoitusten (engl. Execution Governor limits) käsittely sekä käyttöliittymien testaus. Tutkimuksessa kehitettävällä testausstrategialla pyritään yhtenäistämään kohdeorganisaation testauskäytänteitä muun muassa edellä mainittujen ongelmien ratkaisemiseksi.

3.2 Tutkimuskysymys

Tutkimuksen tavoitteena on kehittää kohdeorganisaatiolle testausstrategia, mikä yhtenäistää kohdeorganisaation testauskäytänteitä ja näin ollen parantaa toteutettujen Salesforce-järjestelmien laatua. Vaade yhtenäisille testauskäytänteille on kasvanut kohdeorganisaatiossa viime vuosien aikana. Kohdeorganisaation työntekijämäärän kasvu, projektien koon kas-

vu ja organisaation sisäinen halu sekä asiakkaiden vaade laadun parantamiseksi ovat kaikki vaikuttavia voimia, jotka ovat lisänneet tarvetta yhtenäistää testauskäytänteitä.

Erityisesti asiakkaiden vaatimus korkealaatuisille ohjelmistoille on saanut kohdeorganisaation pohtimaan, miten ohjelmistotestausta kehittämällä asiakkaiden vaatimukseen voitaisiin vastata. Jotta Salesforce-järjestelmien testausta voidaan kehittää, on perehdyttävä siihen mitä kattava testaaminen Salesforce-kontekstissa tarkoittaa ja mitä toimia kohdeorganisaatio voi tehdä testauksen kehittämiseksi. Tutkimus voidaan jakaa seuraaviin tutkimuskysymyksiin:

TK1: Mitä vaatimuksia sisältyy Salesforce-järjestelmän kattavaan testaamiseen?

TK2: Miten kohdeorganisaation ohjelmistotestauskäytänteitä pitäisi kehittää kehitettävien Salesforce-järjestelmien laadun parantamiseksi?

Tutkimus pyrkii vastaaman näihin kysymyksiin valittua tutkimusmenetelmää hyödyntäen. Vastatakseen ensimmäiseen tutkimuskysymykseen tutkimuksen tulee perustella miten ja miksi Salesforce-järjestelmää testataan. Vastatakseen toiseen tutkimuskysymykseen tutkimuksen tulee tarjota ohjelmistotestauskäytänteet Salesforce-järjestelmän testaamiseen. Tutkimuksen teoriaosuus pyrkii vastamaan ensimmäiseen tutkimuskysymykseen ja artefakti toiseen tutkimuskysymykseen.

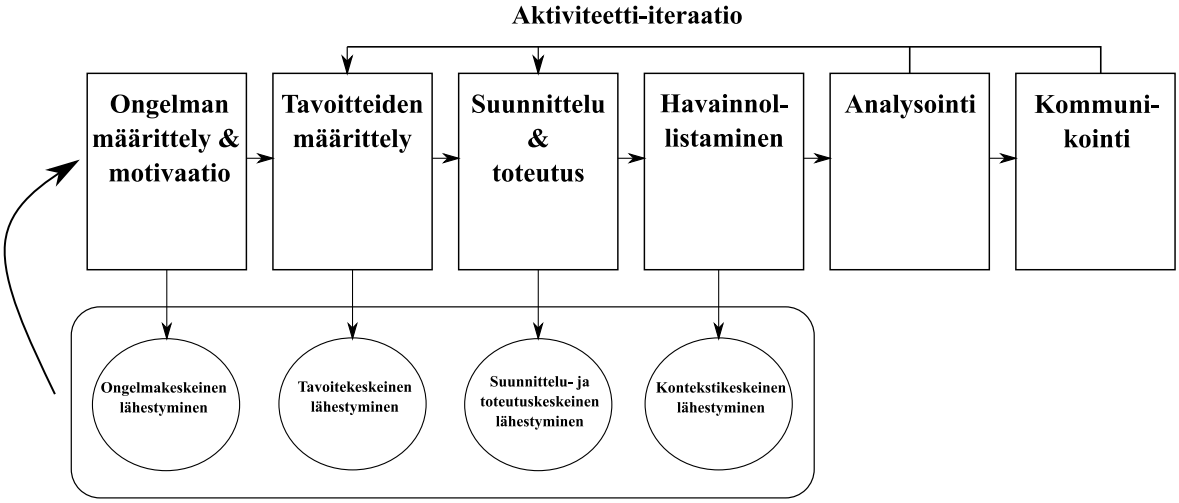
Onnistuessaan tutkimus tarjoaisi kohdeorganisaatiolle edellytykset saavuttaa TMMi-mallin toisen tason ja ohjelmistotestauksen myötä parantaa toteutettujen Salesforce-järjestelmien laatua sekä vähentäisi havaittuja virheitä.

3.3 Tutkimusmenetelmä

Tutkimus toteutetaan käyttäen suunnittelutiedettä (engl. design science research) (Hevner ym. 2004; Peffers ym. 2007). Suunnittelutiede tuottaa artefaktin, joka pyrkii ratkaisemaan jonkin kohdeorganisaation liiketoiminnan tunnistetun ongelman. Suunnittelutiede on siis pohjimmiltaan ongelman ratkaisua, mikä tuottaa esimerkiksi innovaation, käytännön, teknisen valmiuden tai tuotteen eli artefaktin (Hevner ym. 2004; Peffers ym. 2007). Artefaktin sopivuutta ongelmaan selvitetään arvioimalla ja analysoimalla, sitä suhteessa artefaktin tavoitteisiin ja ratkaistavaan ongelmaan (Peffers ym. 2012).

Tutkimus noudattaa Peffersin ym. (2007) määrittämää suunnittelutieteen tutkimusmenetelmää (engl. design science research methodology; lyh. DSRM). DSRM koostuu kuudesta aktiviteetista, jotka noudattavat jo aiemmin informaatioteknologian tutkimuksessa käytettyjä suunnittelutieteen hyviä käytänteitä kuten Hevnerin ym. (2004) määrittelemää seitsemää ohjenuoraa suunnittelutieteelle. Peffersin ym. (2007) määrittämät aktiviteetit tutkimusmenetelmälle ovat: *ongelman määrittely & motivaatio*, *tavoitteiden määrittely*, *suunnittelu & toteutus*, *havainnollistaminen*, *arviointi* ja *kommunikointi*.

Suunnittelutiede on iteratiivinen prosessi eli artefaktin kehitystä jatketaan, kunnes sen arvioidaan ratkaisevan ongelma riittävän hyvin tai löydetään jokin muu syy lopettaa tehtävä tutkimus. Vaikka suunnittelutiede kuten myös DSRM prosessi ovat iteratiivisia ja strukturoituja alkamaan tietystä pisteestä, kuten DSRM:n tapauksessa *ongelman määrittely & motivaatio* -aktiviteetista, se ei tarkoita, etteikö tutkija voisi valita jotain muuta aloituskohtaa. DSRM:n prosessi on mahdollista aloittaa mistä tahansa aktiviteetista. Esimerkiksi tavoitekeskeisesti artefaktin teko voidaan aloittaa *tavoitteiden määrittely* -aktiviteetista tai ratkaisukeskeisesti *suunnittelu & toteutus* -aktiviteetista. (Peffers ym. 2007)



Kuvio 3. DSRM prosessi (Peffers ym. 2007).

DSRM prosessin, joka on kuvattu kuviossa 3, aktiviteetit ovat seuraavat (Peffers ym. 2007):

Ongelman määrittely & motivaatio: Määrittelee ongelman, jonka tuotettu artefakti pyrkii ratkaisemaan. Ongelman tulisi olla sellainen, että se motivoi ongelman tutkijaa löytämään ratkaisun sekä tutkimuksen yleisöä tutustumaan löydettyyn ratkaisuun ja syihin miksi tutkija on

lähtenyt tutkimaan kyseistä ongelmaa. Ongelman määrittely edellyttää tutkijalta tietämystä ongelmasta ja kykyä motivoida yleisöä tutustumaan ratkaisuun.

Tavoitteiden määrittely: Määrittelee artefaktin tavoitteet, jotka ovat mahdollisia ja saavutettavissa. Tavoitteet voivat olla joko määrällisiä kuten millä arvoilla artefakti on parempi kuin aikaisemmat, tai laadullisia kuten kuvaus siitä, että millä tavalla artefaktin odotetaan ratkaisevan ongelman. Tavoitteiden määrittely edellyttää tietämystä ongelmasta ja jo olemassa olevista ratkaisuista, jos niitä on, ja niiden toimivuudesta ongelman ratkaisemiseksi.

Suunnittelu & toteutus: Artefaktin suunnittelu ja toteutus ovat suunnittelutieteen ydin. Suunnittelutieteessä artefakti voi teoriassa olla mikä tahansa suunnittelun ja toteutuksen lopputulos. Lopputulos voi olla esimerkiksi jokin algoritmi, käytänne, malli tai tuote. Suunnittelu ja toteutus sisältää artefaktin haluttujen ominaisuuksien ja rakenteiden määrittelyn sekä artefaktin konkreettisen toteuttamisen. Suunnittelu ja toteutus edellyttää ratkaisuun liittyvän teorian hallitsemista, jotta ratkaisun muodostaminen on mahdollista.

Havainnollistaminen: Havainnollistaa, että artefaktin avulla on mahdollista ratkaista ongelma. Artefaktin toimintaa voidaan havainnollistaa muun muassa tapaustutkimuksella, kenttätutkimuksella, staattisella analyysillä, dynaamisella analyysillä, kontrolloidulla kokeella, simulaatiolla, toiminnallisella testaamisella, havainnollistavalla skenaariolla tai jollain muulla tavalla, mikä osoittaa, että artefaktilla voidaan ratkaista ongelma. Havainnollistaminen edellyttää osaamista artefaktin käytöstä ongelman ratkaisemiseksi.

Analysointi: Kuinka hyvin artefakti ratkaisee ongelman. Tässä vaiheessa analysoidaan, miten artefaktille ennalta määritellyt tavoitteet vertautuvat todellisiin havaintoihin, jotka on kerätty *havainnollistaminen* aktiviteetista. Analysointi edellyttää osaamista kelvollisista metriikoista ja analyysimenetelmistä. Aktiviteetin jälkeen päätetään palataanko prosessissa *suunnittelu & toteutus* aktiviteettiin vai jatketaanko viimeiseen *kommunikointi* aktiviteettiin.

Kommunikointi: Ongelmasta, ongelman tärkeydestä ja artefaktista kommunikointi tiedeyhteisölle ja oleellisille sidosryhmille, kuten kohdeorganisaatiolle. Kommunikointi edellyttää tietämystä kommunikoinnista tiedeyhteisössä ja oleellisissa sidosryhmissä.

Artefaktin havainnollistamiseksi ja analysoitavien tuloksien keräämiseksi voidaan käyttää

useita eri metodeja, kuten havainnollistavaa skenaariota tai asiantuntija-arviointia (Peffers ym. 2012; Peffers ym. 2007). Havainnollistavassa skenaariossa artefaktin sopivuutta ja hyödyllisyyttä havainnollistetaan joko keksityllä tai todellisella datalla (Peffers ym. 2012). Asiantuntijoiden arvioinnissa puolestaan artefakti annetaan yhden tai useamman asiantuntijan arvioitavaksi (Peffers ym. 2012). Havainnollistavaa skenaarioita on käytetty useimmin aiemmissa informaatioteknologian tutkimuksessa verrattuna asiantuntija-arviointiin, mutta molemmat menetelmät ovat sopivia artefaktin analysointiin (Peffers ym. 2012).

3.4 Suunnittelutiede tutkimuksessa

Tutkimuksessa suunnittelutiedettä käytetään kohdeorganisaation liiketoiminnan ongelman ratkaisemiseksi. Liiketoiminnan ongelmana on yhtenäisten ohjelmistotestauskäytänteiden puuttuminen, jotka tarjoaisivat vastaukset ohjelmiston testaukseen liittyen kysymyksiin: mitä, kuka, milloin, missä, miten ja millä (Kasurinen 2013, s. 123). Tutkimuksen tavoitteena on luoda kohdeorganisaatiolle testausstrategia, joka vastaa juuri näihin kysymyksiin. Testausstrategia on näin ollen tutkimuksen artefakti.

Tutkimuksessa noudatettiin Peffersin ym. (2007) määrittelemiä suunnittelutieteen aktiviteetteja kronologisessa järjestyksessä. Ensimmäinen iteraatiokierros artefaktin muodostamista aloitettiin aktiviteetilla *ongelman määrittely ja motivaatio*, jossa määriteltiin artefaktin ratkaistava ongelma. Johdimme ongelman tutkimuksen tutkimuskysymyksistä:

Ongelma: Kohdeorganisaatiolla ei ole selkeitä ohjelmistotestauskäytänteitä Salesforce -toteutuksien testaamiseksi.

Iteraation ensimmäiseen vaiheeseen kuuluu oleellisena osana ongelman määrittelemisen lisäksi tutkittavaan aiheeseen motivointi. Tutkimuksen päämotivaattorina toimii kohdeorganisaation halu ja tarve kehittää ja yhtenäistää ohjelmistotestauskäytänteitä, jotka parantavat tuotettujen Salesforce-toteutuksien laatua. Kuten jo johdantoluvussa todetaan, ohjelmistotestaus on yksi tehokkaimpia keinoja ohjelmiston laadun varmistamiseksi ja kehittämiseksi (Seth ym. 2012).

Ongelman määrittely & motivaatio aktiviteetin jälkeen siirryttiin *tavoitteiden määrittely* ak-

tiviteettiin. Määriteltyihin tavoitteisiin vertaamalla kyetään analysoimaan, kuinka hyvin artefakti ratkaisee annetun ongelman (Peffers ym. 2007). Artefaktin tavoitteet jaettiin kahteen kategoriaan: yleiset tavoitteet ja tavoitteet projektiryhmän rooleittain.

Artefaktin tavoitteet:

- Yhtenäistää miten ja millä laajuudella sekä perustella miksi Salesforce-toteutuksia testataan.
- Antaa raamit testaussuunnitelman, testitapauksien ja testausraportin muodostamisille.
- Tukee päätöksentekoa testauksen automatisoinnista.

Artefaktin tavoitteet projektiryhmän rooleittain:

- Arkkitehdin, kehittäjän ja konsultin rooleissa toimivat pystyvät valitsemaan millä laajuudella toteutusta testataan ja mitä testausmenetelmiä laajuuden saavuttamiseksi olisi suositeltavaa käyttää.
- Projektipäällikön ja myyjän rooleissa toimivat pystyvät perustelemaan asiakkaalle vaadittavan testauksen laajuuden, sekä muodostamaan työmääräarvion vaadittavasta testaamisesta.

Seuraavaksi siirryttiin *suunnittelu & toteutus* aktiviteettiin, joka on suunnittelutieteen ydin (Peffers ym. 2007). Artefaktia lähdettiin suunnittelemaan TMMi Foundationin (2022, s. 24–28), Kasurisen (2010) ja Kasurisen (2013, s. 71–73 ja s. 132–136) testausstrategialle esittämien määritelmien mukaan. Tarkemmin artefaktin suunnittelusta ja toteutuksesta kerrotaan luvussa 4.

Kun artefakti oli valmis kokeiltavaksi, siirryttiin *havainnollistaminen* aktiviteettiin, jossa havainnollistavaa skenaariota hyödyntämällä validoitiin artefaktin toiminta ja kerättiin tarvittavat havainnot *analysointi* aktiviteettia varten. Havainnollistavaa skenaariota varten keksittiin kaksi projektia kuvamaan kohdeorganisaation tyypillisiä Salesforce-järjestelmän käyttöönottoprojekteja.

Analysointi aktiviteetissa hyödynsimme havainnollistavalla skenaariolla kerättyjä havaintoja. Analysoinnin tuloksena todettiin, että artefakti vastaa määriteltyihin tavoitteisiin ja tarjoaa kohdeorganisaatiolle tyydyttävän ratkaisun annettuun ongelmaan jo ensimmäisen iteraation

jälkeen. Artefaktin iterointi päätettiin lopettaa ja siirtyä viimeiseen aktiviteettiin. Havainnollistavan skenaarion ja kirjallisuuden pohjalta artefaktin analysoinnista kerrotaan tarkemmin luvussa 5.

Viimeisenä aktiviteettina on *kommunikointi*. Tämä tutkielma toimii kommunikaatiovälineenä artefaktista tiedeyhteisölle. Tutkielman lisäksi artefaktista kommunikoitiin kohdeorganisaatiossa sisäisesti ja artefakti jaettiin kaikkien kohdeorganisaation työntekijöiden saataville.

4 Artefaktin suunnittelu ja toteutus

Tutkimuksen artefaktina toimii kohdeorganisaatiolle kehitetty testausstrategia. Tässä luvussa kerrotaan, kuinka artefakti suunniteltiin ja mikä oli suunnittelutieteen aktiviteetin *suunnittelu ja toteutus* lopputulos, eli esitetään tuotettu testausstrategia. Tutkimus tehtiin toimeksiantona kohdeorganisaatiolle, eikä artefaktia voida siitä syystä kokonaisuudessaan julkaista tutkielman liitteenä, koska artefakti sisältää kohdeorganisaation salassa pidettävää omaisuutta.

Ensimmäisessä aliluvussa 4.1 kerrotaan mikä oli artefaktin suunnitelma ja suunnitteluun liittyneet pohdinnat. Seuraavassa aliluvussa 4.2 esitetään toteutettua artefaktia niiltä osin, kun se on tutkielman kannalta oleellista.

4.1 Suunnitelma

Artefaktia lähdettiin suunnittelemaan TMMi Foundationin (2022, s. 24–28), Kasurisen (2010) ja Kasurisen (2013, s. 71–73 ja s. 132–136) esittämien määritelmien mukaan testausstrategialle. Kaikki kolme julkaisua määrittelevät, että testausstrategian tulisi sisältää muun muassa: määritelmät, tavoitteet ja vastuut kaikille testaustasoille (yksikkötestaus, integraatiotestaus, järjestelmä- ja hyväksymistestaus); ohjeet testitapauksen, testaussuunnitelman ja testausraportin muodostamiselle; vastuut, kuka vastaa ja mistä testaukseen liittyvästä osa-alueesta; ja testauksen aloitus- ja lopetusehdot.

TMMi Foundationin (2022, s. 24–28), Kasurisen (2010) ja Kasurisen (2013, s. 71–73 ja s. 132–136) määritelmien pohjalta kokosimme kohdeorganisaation kannalta kriittisimmät asiat, joiden tulisi olla testausstrategiassa: määritelmä testauksen laajuudesta riippuen projektin eri tekijöistä, ohjeet testitapauksen, testaussuunnitelman ja testausraportin muodostamiselle sekä selitykset testaukseen liittyville termeille. Testausstrategiaan haluttiin myös sisällyttää perusteluja, miksi testataan, vaikka kirjallisuudessa ei erikseen mainittu testauksen perustelujen sisällyttämisestä testausstrategiaan.

Suunnittelun aikana testausstrategialle asetettiin tiettyjä toiveita ja vaatimuksia kohdeorganisaation toimesta. Kohdeorganisaation toiveena oli, että testausstrategia tukisi koko pro-

jektiryhmää testaamisen määrittelyssä ja aloittamisessa muun muassa testauksen työmäärän arvioinnissa ja testitapauksien muodostamisessa. Lisäksi testausstrategian vaadittiin olevan sellainen, että sitä voisi hyödyntää kohdeorganisaation nykyisiin tai tuleviin projekteihin.

Suunnittelun alusta alkaen oli selvää, että testausstrategiaan tulee jäämään runsaasti tulkinnanvaraa esimerkiksi testitapauksien muodostamiseen liittyen, eikä testausstrategiasta haluttu lähteä muodostamaan pikkutarkkaa ohjetta. Kuten Kasurinen (2013) toteaa, on testausstrategian tarkoitus olla yleisluontoinen ohje, joka jättää asioita tulkinnanvaraan testaus- tai projektipäällikön päätettäväksi projektikohtaisesti. Vaikka kohdeorganisaatiolla oli toive, että testausstrategiassa olisi ohje testitapauksien muodostamiselle, ymmärrettiin kohdeorganisaatiossa hyvin testausstrategian luonne yleisluontoisena ohjenuorana eikä pikkutarkkana ohjeena.

Kun kirjallisuudessa olevat määritelmät olivat selvillä, lähdettiin suunnittelemaan, kuinka testausstrategiasta saataisiin sellainen, että se tukisi kohdeorganisaation kaikenkokoisia ja -laajuisia projekteja. Aluksi pohdittiin yhtä kaikille yhtenäistä testausstrategiaa, mutta ajatuksesta luovuttiin varsin nopeasti. Seuraavana vaihtoehtona oli testausstrategia, joka skaalautuisi projektin eri tekijöiden kuten laajuuden, järjestelmän koon, käyttäjämäärän, laatutekijöiden, ynnä muiden mukaan. Suunnittelussa päädyttiin valitsemaan skaalautuva vaihtoehto.

Testausstrategian skaalautuvuudelle suunniteltiin kolme eri tasoa ja tasojen valintaa varten suunniteltiin testaustason valintapolku. Tasot ja valintapolun kriteeristö suunniteltiin kohdeorganisaation menneiden projektien perusteella. Jokainen taso suunniteltiin sisältämään TMMi Foundationin (2022, s. 24–28), Kasurisen (2010) ja Kasurisen (2013, s. 71–73 ja s. 132–136) määrittelemiä vaatimuksia, mutta ottamaan huomioon projektin koon ja laajuuden. Eri tasoja suunniteltaessa otettiin huomioon se, mitä vaatimuksia Salesforce asettaa testaamiselle. Salesforcen asettamat vaatimukset tulisivat toimimaan pohjatasona, jota jo ensimmäinen taso täydentäisi.

Suunnittelussa kiinnitettiin erityisesti huomiota, että testausstrategia olisi helposti ymmärrettävissä myös sellaisille kohdeorganisaation työntekijöille, joilla ei ole aiempaa taustaa ohjelmistotestauksesta, jonka vuoksi testausstrategiassa on kiinnitetty paljon huomioita muun muassa termien määrittelyyn. Termien määrittelyllä testausstrategiassa toivotaan vältyttävän

väärinymmärryksiltä testaukseen liittyvän sanaston saralla.

Jo suunnitteluvaiheessa testausstrategiasta päätettiin jättää tarkoituksella pois tarkat kriteerit, milloin testaus tulisi aloittaa ja lopettaa, jotta strategia ei rajoittaisi liikaa projektikohtaisia erikoistapauksia. Testauksen aloitus- ja lopetuskriteerit suunniteltiin osaksi projektikohtaista testaussuunnitelmaa, johon liittyviä yleisiä ohjeita on suunnitelman mukaan tarkoitus kirjata testausstrategiaan. Samoin testausstrategiasta päätettiin jättää tarkoituksella pois testiympäristöjä käsittelevät maininnat, sillä ne ovat kohdeorganisaation tapauksessa täysin projektikohtaisia.

Testausstrategiaa lähdettiin toteuttamaan noudattaen informaatioteknologian alan yleisiä käytänteitä sekä kirjallisuudessa, kuten TMMi Foundationin (2022, s. 24–28) ja Kasurisen (2013, s. 71–73 ja s. 132–136) määrittelemiä asioita siten, että testausstrategiasta olisi mahdollisimman paljon apua ja hyötyä kohdeorganisaatiolle. Seuraavassa aliluvussa 4.2 esitetään suunnittelun ja toteutuksen tuloksena syntynyttä testausstrategiaa.

4.2 Toteutus

Artefakti toteutettiin strukturoituna ”Confluence” dokumenttina, joka toimii kohdeorganisaation testausstrategiana. Artefaktin sisältö noudattaa hyviä informaatioteknologian alan yleisiä käytänteitä ja tutkimuksessa todettuja faktoja sovitettuna kohdeorganisaation tarpeisiin ja Salesforce-järjestelmien testaamiseen. Artefakti koostuu luvuista: Johdanto, Sanasto, Testaustason valinta, Projektin vaiheet ja Testauskäytänteitä.

Johdanto esittelee artefaktin käyttäjälle, mistä artefaktissa on kyse. Sanastoluku määrittelee artefaktin kannalta tärkeän termistön. Lukua ”Testaustason valinta” hyödyntämällä artefaktin käyttäjä kykenee itsenäisesti määrittelemään Salesforce-projektin vaatiman testaustason kolmesta eri tasosta. Projektin vaiheet -luvussa määritellään artefaktin käyttäjälle mitä testaustoimenpiteitä liittyy eri projektin vaiheisiin. Viimeisessä luvussa esitetään artefaktin käyttäjälle konkreettisia ohjeita ja erilaisia hyviä testauskäytänteitä Salesforce-projektien testaamiseen.

Artefaktin Johdantoluvussa esitetään, mitä testausstrategialla tarkoitetaan, mihin sitä on tar-

koitus käyttää ja esitetään artefaktin rakenne. Luku on haluttu pitää lyhyenä ja ytimekkäänä. Johdannon jälkeen tulee termiluettelo, jossa määritellään kaikki artefaktin käytön kannalta oleelliset termit kuten esimerkiksi testitapaus, testaussuunnitelma ja eri testaustyypit. Termien määrittelyllä pidetään huoli siitä, että jokainen artefaktin käyttäjä tietää, mistä puhutaan riippumatta hänen aiemmasta ohjelmistotestausosaamisestaan.

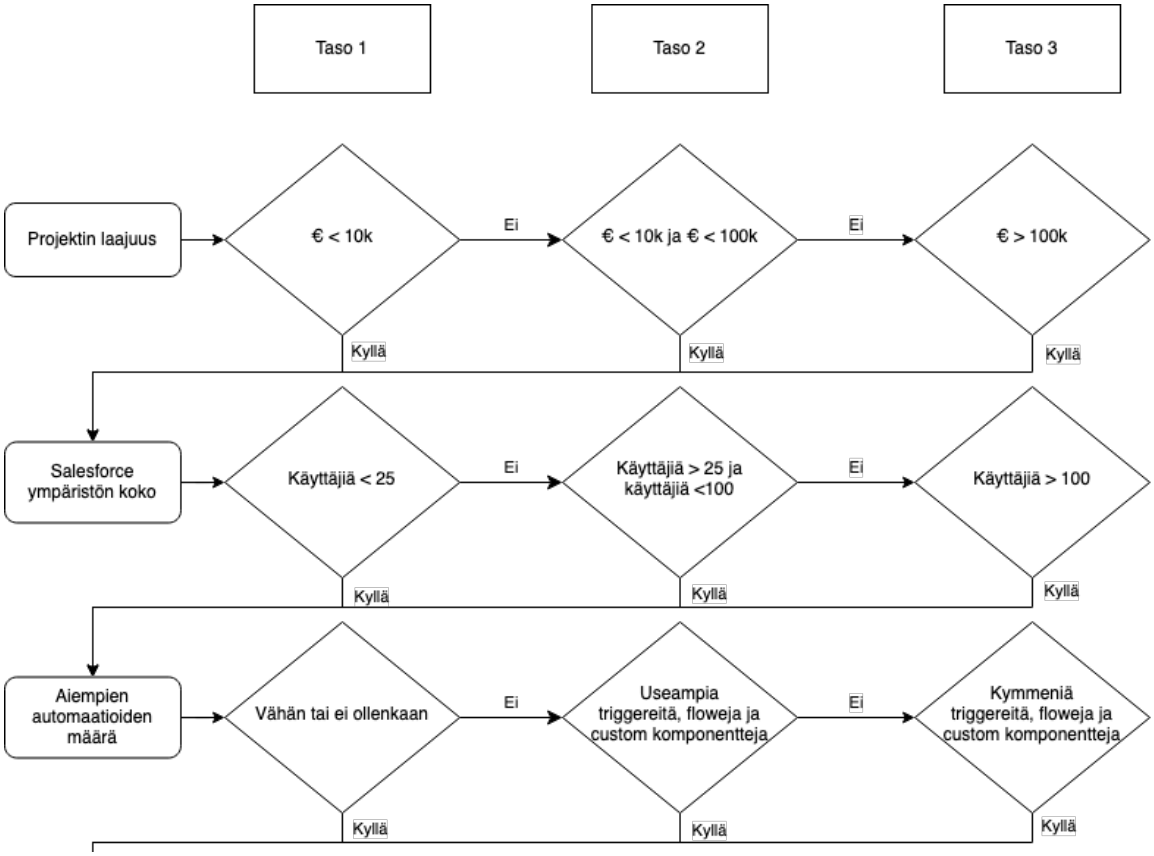
Testaustason valinta -luvussa artefaktin käyttäjä kykenee määrittelemään Salesforce-projektille vaaditun testaustason kulkemalla testaustason valintapolun läpi. Testaustasoja on kolme erilaista ja valintapolun tulos määrittää vaaditun testaustason. Testaustaso 1 on suppein ja testaustaso 3 on laajin. Testaustasot ovat suhteutettuina kohdeorganisaation aiemmin toteuttamiin Salesforce projekteihin. Tulevaisuudessa on mahdollista, että testaustasoja on enemmän kuin kolme.

Testaustason sisältämät tehtävät ja vaatimukset voidaan jakaa karkeasti siten, että taso 1 sisältää vain välttämättömän testauksen sen osalta, että muutokset saadaan vietyä Salesforcen tuotantoympäristöön. Tämä tarkoittaa vähimmillään savutestauksen toteutetulle ominaisuudelle ja 75 %:n koodirivikattavuuden testeillä ohjelmakoodille. Taso 2 sisältää kattavan manuaalisen testaamisen, sekä mahdollisesti joitain automaatiotestejä. Taso 3 sisältää kattavan manuaali- ja automaatiotestauksen.

Testaustasot haluttiin jakaa kolmeen eri tasoon, jottei pienemmissä tai muutoin sopimattomissa projekteissa lähdettäisi tuhlaamaan resursseja automaatiotestauksen kanssa vaan valittaisiin projektin laajuudelle ja asiakkaan Salesforce-ympäristölle sopivin testaustaso. Kuten Taipale ym. (2011) toteavat tutkimuksessaan, organisaation on syytä tarkkaan pohtia, onko heidän testausprosessissansa toimenpiteitä, joita olisi mielekästä automatisoida. Kalliiseen ja raskaaseen testiautomaation rakentamiseen ei ole järkevää ryhtyä kohdeorganisaation jokaisen projektin osalta.

Testaustaso määräytyy kuviossa 4 esitetyn valintapolun mukaan siten, että kuinka monta vastausta on osunut samalle tasolle. Jos 50 % tai enemmän vastauksista on osunut samaan tasoon, valitaan kyseinen taso. Jos mihinkään tasoon ei ole osunut vähintään 50 % vastauksista, valitaan tällöin oletuksena taso 2, kuten kuviossa 5 on esitetty.

Artefaktin luvussa ”Projektin vaiheet” on listattu yleiset ohjelmistoprojektin vaiheet: mää-

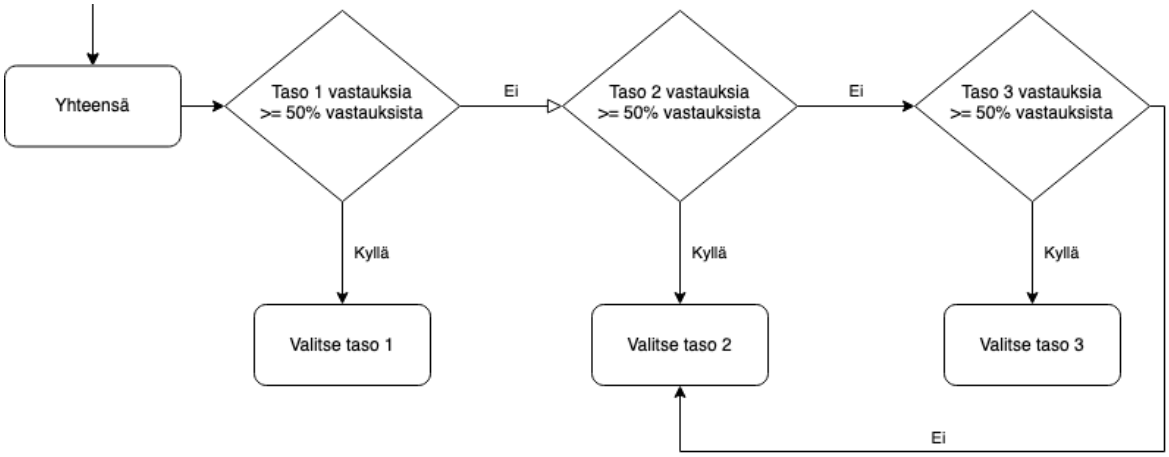


Kuvio 4. Testaustason valintapolun alku.

rittely, suunnittelu, toteutus, käyttöönotto sekä ylläpito ja jatkokehitys (Kasurinen 2013). Jokaisen projektin vaiheen kohdalla on määritelty jokaiselle testaustasolle kuuluvat tehtävät ja vaatimukset, jotka tulee suorittaa. Tehtävät ja vaatimukset on määritelty siten, että jos valittu testaustaso on 2, tulee myös tason 1 tehtävät ja vaatimukset suorittaa ennen tason 2 tehtäviä ja vaatimuksia.

Tehtävien ja vaatimuksien lisäksi jokaiseen projektin vaiheeseen liittyville testaustasoille on annettuna työmääräarvio testaustason edellyttämän testauksen vaikutuksesta projektin työmääräarvioon. Työmääräarviovaikutukset on laskettu sillä oletuksella, ettei alkuperäinen työmäärä sisällä testauksen osuutta vaan ainoastaan ominaisuuden toteuttamiseen vaadittavan työmäärän. Testauksen vaikutus työmääräarvioon voi olla joko kiinteä esimerkiksi +1 henkilötyöpäivä tai kerrannainen lisäys esimerkiksi 1,3 kertaa toteutuksen arvioitu työmäärä.

Testauksen vaikutus työmääräarvioon perustuu Boehmin ja Basilin (2001) väitteeseen, et-



Kuvio 5. Testaustason valintapolun loppu.

tä 60–80 % ohjelmistovirheistä aiheutuvat 20 prosentin osuudesta ohjelmaa. Toisin sanoen testaamalla hyvin virheitä aiheuttava 20 prosenttia ohjelmakoodista, voidaan välttyä jopa 80 prosentilta ohjelmistovirheistä. Näin ollen artefaktissa esitetyt arviot testauksen vaikutuksesta työmäärään perustuvat olettamukseen, että 20 % ohjelmasta testataan kattavasti ja lopun 80 %:n testaus hoidetaan, sillä mitä työmäärästä on jäänyt jäljelle.

Viimeisessä luvussa on annettuna konkreettisia ohjeita testaamisen tueksi testauskäytänteiden muodossa. Luvussa kerrottuja testauskäytänteitä ovat muun muassa ohjeet testitapauksen muodostamiseen, testaussuunnitelman sisältöön ja testauksesta raportointiin. Testauskäytänteet-luvun lisäksi artefaktin muissa luvuissa on tuotu esiin ohjeita ja käytänteitä testauksen suunnitteluun ja toteutukseen esimerkiksi luvun ”Projektin vaiheet” kappaleessa ”Suunnittelu” nostetaan esiin asioita, joita on hyvä pohtia testausta suunnitellessa ja testiautomaation hyödyntämisestä päätettäessä.

Artefakti noudattaa hyviä informaatioteknologian alan käytänteitä ja tutkimuksessa todettuja faktoja, sovellettuna kohdeorganisaation tarpeisiin ja Salesforce-järjestelmien testaukseen. Artefakti koostuu johdannosta, sanastosta, testaustason valinnasta, projektin vaiheista ja testauskäytänteistä. Johdanto esittelee artefaktin käyttäjälle sen sisällön ja tarkoituksen. Sanastoluettelo määrittelee artefaktin kannalta keskeisen terminologian. Testaustason valinta -luvussa käyttäjä voi itsenäisesti määritellä Salesforce-projektin vaatiman testaustason kolmesta eri tasosta valintapolun avulla. Projektin vaiheet -luvussa määritellään, mitä testaustoi-

menpiteitä liittyy eri projektin vaiheisiin. Viimeisessä luvussa esitetään konkreettisia ohjeita ja hyviä testauskäytänteitä Salesforce-projektien testaukseen.

5 Artefaktin analysointi ja reflektointi

Luku 5 keskittyy ehdotetun testausstrategian analysointiin ja reflektointiin. Ensimmäisessä aliluvussa 5.1 tarkastellaan artefaktin toimivuutta havainnollistavan skenaarion 5.1.1 avulla. Tämän jälkeen verrataan artefaktia kirjallisuuteen aliluvussa 5.2 ja vastataan tutkimuskysymyksiin aliluvussa 5.3. Viimeisessä aliluvussa 5.4 pohditaan artefaktin rajoituksia, validiteettia, haasteita ja mahdollisuuksia jatkokehitykselle.

5.1 Analysointi

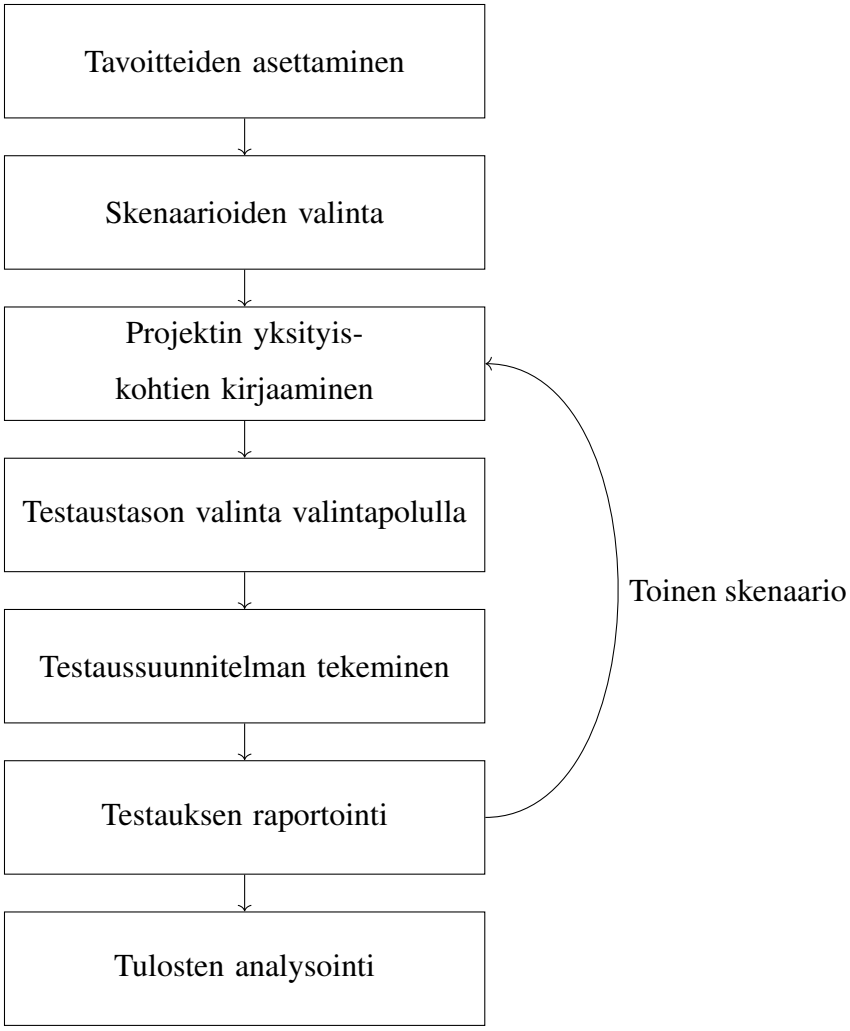
Testausprosessi on kriittinen osa ohjelmistokehitystä, jolla varmistetaan, että kehitetty ohjelmistotuote toimii halutulla tavalla. Tässä tutkimuksessa kehitetty Salesforce-kehitykseen tarkoitettu testausstrategia on suunniteltu vastaamaan kohdeorganisaation tarpeita testaamisessa havaittujen haasteiden selättämiseksi. Sen toimivuuden tehokkuutta on pyritty varmistamaan kirjallisuuden, parhaiden käytäntöjen ja parhaiksi havaittujen menetelmien avulla. Tämän aliluvun tarkoituksena on analysoida ja arvioida toteutetun artefaktin tehokkuutta ja toimivuutta osana ohjelmistokehitysprosessia.

Testausstrategia on suunniteltu työkaluksi parantamaan kohdeorganisaation tuottaman ohjelmiston laatua testauskäytänteitä optimoiden sekä selkeyttääkseen testaamisen vaiheita ja syvyyttä. Testausstrategiassa käsitellään useita ohjelmistokehitysprosessin kriittisiä osia, kuten suoritettavat testaustyypit, -prosessi ja -menetelmät sekä käytettävät työkalut ja tekniikat. Sen tarkoituksena on hahmotella erikokoisiin Salesforce-kehitysprojekteihin kattava testausprosessi kattaen projektin eri vaiheet, testaustasojen valinnan, tarpeen automaattiselle testaamiselle ja testausraportin tekemisen. Testausprosessi on suunniteltu avustamaan kohdeorganisaation tyypillisiä Salesforce-projekteja.

Artefaktin analysointimenetelmänä käytettiin havainnollistavaa skenaariota, jota selitetään tarkemmin seuraavissa aliluvuissa.

5.1.1 Havainnollistava skenaario

Havainnollistaaksemme artefaktin tehokkuutta kuvaamme käytännön esimerkkejä apuna käyttäen, kuinka strategiaa voi käyttää oikeassa tilanteessa. Näillä esimerkeillä pyritään selvittämään artefaktin toimivuutta sekä sen hyötyjä ja mahdollisia rajoituksia. Valitsimme mielivaltaisesti kaksi projektia havainnollistavia skenaarioita varten. Toinen havainnollistava skenaario on kohdeorganisaation jo toteutunut keskikokoinen projekti ja toinen on kokoluokaltaan suuri ei vielä toteutunut projekti.



Kuvio 6. Havainnollistavien skenaarioiden prosessi.

Havainnollistavien skenaarioiden lähestymistapa oli kuvion 6 mukainen. Havainnollistavat skenaariot alkoivat päättämällä kaksi toisistaan eroavaa projektia, jonka jälkeen siirryttiin

etenemään artefaktin ohjeistuksen mukaan projektikohtaisesti. Aluksi kirjattiin projektin yksityiskohdat sisältäen muun muassa asiakastiedot, projektin euromääräisen koon, kehitykseen liittyvän suunnitellun työmääräarvion ja lyhyen kuvauksen toteutusratkaisun suunnittelusta. Skenaarioissa seuraavaksi seurattiin artefaktin valintapolkua testaustason valitsemiseksi. Kaikkiin valintoihin kirjattiin myös lyhyt kuvaus valinnasta, esimerkiksi ylin taso (taso 3) järjestelmän kriittisyydestä saatettiin valita sillä perusteella, että asiakkaan liiketoiminta voi häiriöityä tai pahimmassa tapauksessa jopa keskeytyä, jos toteutettu järjestelmä ei toimi oletetulla tavalla.

Seuraavaksi havainnollistavissa skenaarioissa siirryttiin testaussuunnitelmien tekemiseen. Suunnitelmat sisälsivät muun muassa tarkemmat Salesforce-kehitykseen vaikuttavat asiakkaan tiedot, testitapauksia ja testausaikataulun. Lopuksi testauksen raportointiosiossa kuvattiin toiminnallisen testauksen raportointitaulukko, joka ryhmiteltiin testitapausten vakavuusluokkien mukaan ylätasolla ryhmiteltynä esimerkiksi pilvi- tai sovelluskohtaisesti, kuten artefaktista otetussa kuviossa 7.

Vakavuusluokka / Cloud, sovellus tai objekti	**Puute**	**Virhe**	**Suorittamatta**	**Huomio**	**Ok**	**Yhteensä**
Sales Cloud	1	2	0	0	4	7
Tuotteen varustelutyökalu	0	0	1	1	5	7
Tuotteiden varaustyökalu	0	1	0	1	3	5
Yhteensä	1	3	1	2	12	19

Kuvio 7. Havainnollistavan skenaarion yhden testauskerran yhteenvetoraportti.

Pienemmässä projektissa artefaktin suosittelema testaus oli linjassa siinä jo suoritetun testaamisen kanssa. Suurimpana erona toteutettuun testaamiseen oli, että artefakti ehdotti testiautomaatiota, jos testejä arvioidaan ajettavan useaan otteeseen projektin aikana. Artefaktin havaittiin helpottavan sopivan testaustason valintaa ja testitapausten määrittämistä. Suurempaan, vielä keskeneräiseen projektiin testaussuunnitelma ja testaustapa vaikuttivat artefaktin perusteella sopivilta ja loogisilta. Artefakti suositteli testiautomaatiota myös tähän projektiin.

Kahden kuvatun havainnollistavan skenaarion tulokset osoittavat, että testausstrategia ohjaa tehokkaasti Salesforce-projektin testausprosessia. Kahden kuvatun skenaarion tulokset osoittivat, että artefakti auttoi valitsemaan sopivan testaustason, kehittämään loogisen testaussuunnitelman sekä raportoimaan testaussuunnitelmasta ja suoritetusta toiminnallisesta testauksesta.

5.1.2 Tavoitteet

Seuraavaksi tarkastellaan, täyttyivätkö artefaktille asetetut tavoitteet havainnollistavien skenaarioiden perusteella. Ensiksi esitetään tavoitteet ja selitetään niiden toteutumista.

Testausstrategian pitäisi sisältää vähintään määritelmä testauksen laajuudesta riippuen projektin eri tekijöistä, ohjeet testitapauksen, -suunnitelman ja -raportin muodostamiselle sekä selitykset eri testaukseen liittyville termeille (TMMi Foundation 2022, s. 24–28; Kasurinen 2013, s. 71–73 ja s. 132–136). Tämä vaatimus täyttyy, koska artefakti sisältää kaikki tarvittavat testausstrategian elementit. Testausstrategia tarjoaa kattavan määritelmän testauksen laajuudesta eri muuttujien perusteella, kuten projektin laajuus, Salesforce-ympäristön koko ja aiempien automaatioiden määrä. Artefaktissa on selkeä ohjerunko testitapausten, testaussuunnitelman ja testausraportin luomiseen. Lisäksi artefakti sisältää selityksiä erilaisille testaukseen liittyville termeille, mikä varmistaa, että kaikki projektiryhmän jäsenet ymmärtävät testaussuunnitelman.

Kohdeorganisaatio ehdotti, että artefaktin vaiheissa selitettäisiin miksi testata, kuten testausstrategia ohjaa. Myös tämä toteutuu artefaktissa, koska siinä selitetään kussakin vaiheessa joka tasolla, mitä tehdään, miksi tehdään, annetaan esimerkkejä ja ohjataan parhaisiin käytänteisiin. Esimerkiksi suunnitteluvaiheessa toisella ja kolmannella tasolla ollessa kerrotaan muun muassa selityksien kera, että testiautomaatiota on hyvä harkita, kun joudutaan projektin edetessä suorittamaan toistuvasti tietyt testitapaukset (Taipale ym. 2011).

Testausstrategian tulisi myös olla ohjenuorana projektiryhmän kaikille jäsenille selventämään Salesforce-käyttöönoton testausvaatimuksia ja -tavoitteita. Projektin alkupuolella testausstrategiasta tulisi saada apua työmääräarvion tekemiseen ja testitapausten suunnitteluun. Nämä tavoitteet täyttyvät, koska artefakti on suunniteltu siten, että kuka tahansa projektiryh-

män jäsen pystyy pitämään testausstrategiaa ohjenuorana, mutta tarvittaessa antaen tarkempaa tietoa eri projektiryhmän jäsenille. Esimerkiksi projektipäällikkö voi selvittää suuntaa antavasti työmääräarvion testaustason valintapolun avulla. Puolestaan kehittäjä voi seurata testausstrategian vaiheita, joista selviää, miten testauksen osalta kuuluu toimia eri tasoilla.

Kohdeorganisaatio myös esitti vaateen, ettei testausstrategia ole liian spesifi, vaan että se jättäisi tilaa säätöjen tekemiselle mahdollisia poikkeustilanteita varten. Testausstrategian tulisi olla myös skaalautuva eri kokoisille projekteille. Skaalautuvuutta on pyritty edistämään testaustasojen avulla. Havainnollistavien skenaarioiden perusteella tasojen päättäminen testaustason valintapolun avulla on toimiva ratkaisu ohjeistamaan testaamista eri kokoisissa projekteissa. Lisäksi artefakti on toteutettu siten, että tarvittaessa neljännen tason lisääminen tai arvojen muuttaminen on mahdollista, jos jatkossa havaitaan tarvetta pienille hienosäädöille.

Esitettyjen tulosten perusteella voidaan päätellä, että artefakti asettaa sille asetetut tavoitteet. Artefakti antaa selkeät ohjeet siitä, mitä on testattava ja mitkä tekijät voivat vaikuttaa testaukseen, jolloin artefaktin käyttäjän on helpompi valita sopiva testaustaso ja kehittää tehokas testaussuunnitelma.

5.1.3 Analysointimenetelmän toimivuuden arviointi

Testausstrategian tehokkuuden ja käytettävyyden analysoimiseksi Peffersin ym. (2012) mukaan käytännön testi olisi kattava lähestymistapa artefaktin toimivuuden arviointiin, jonka pohjalta päätimme toteuttaa kaksi havainnollistavaa skenaariota tämän saavuttamiseksi. Yksi skenaarioiden eduista oli, että ne tarjosivat realistisen kontekstin testausstrategian testaamiseen. Ensimmäinen skenaario, joka perustui jo päättyneeseen projektiin, antoi tarkan esityksen todellisesta tilanteesta. Toinen skenaario, joka suunniteltiin projektin vaatimusten ja kokemuksen perusteella oli asiantuntijoiden mukaan perusteltua ja käytännöllistä toteuttaa.

Testausskenaarioiden tulokset osoittivat, että testausstrategia oli tehokas työkalu kohdeorganisaation Salesforce-testauksen yhtenäistämiseksi. Testausstrategia osoittautui toimivaksi molemmissa skenaarioissa, mikä viittaa siihen, että sitä voidaan soveltaa vastaaviin tilanteisiin tulevaisuudessa. Skenaariot auttoivat myös ymmärtämään testausstrategian rajoituksia ja kuinka sitä voitaisiin parantaa, jotta se olisi tehokkaampi ongelmien tunnistamisessa ja

ratkaisemisessa.

Yksi lähestymistapamme rajoituksista oli, että suoritimme vain kaksi havainnollistavaa skenaariota, koska emme ryhtyneet toiseen iteraatioon DSRM-prosessissa. Tämä rajoittaa ymmärrystä siitä, kuinka testausstrategia toimisi eri yhteyksissä. Iteraatioita toisaalta tulee käytännön työssä paljon, jolloin todellisten arvojen pohjalta voidaan tarvittaessa säätää arvoja. Lisäksi on huomioitava, että skenaarioissa ei voi ottaa huomioon kaikkia mahdollisia tilanteita, joita käytännössä voi tapahtua, mutta loputon määrä mahdollisten tilanteiden kattamiseksi ei olisi järkevää. Siitä huolimatta skenaariot olivat arvokas lähestymistapa testausstrategian tehokkuuden ja käytettävyyden analysoinnissa ja antoivat käsityksen siitä, miten sitä voitaisiin parantaa vastaamaan tuleviin haasteisiin.

Skenaarioiden avulla pystyimme simuloimaan erilaisia käyttöskenaarioita, joiden avulla saatiin kattava arvio artefaktista. Vaikka havainnollistavan skenaarion toteuttajat ovat Salesforce-kehityksen asiantuntijoita, voisi artefaktin toimivuutta analysoida myös joku artefaktin kehityksen ulkopuolinen Salesforce-kehitykseen erikoistunut henkilö. Tällöin neutraali osapuoli pystyisi kertomaan testausstrategian toimivuudesta hänen työssään siten, että hän olisi itse joutunut testausstrategian dokumentaatioon nojautuen käyttämään testausstrategiaa.

Havainnollistavan skenaarion toteuttaminen oli hyvä tapa analysoida tutkittavan artefaktin tehokkuutta. Esittämällä realistisia tilanteita skenaarioissa pystyttiin tutkimaan, kuinka käyttäjät ovat vuorovaikutuksessa artefaktin kanssa kontekstissa, ja tunnistamaan mahdollisia toiminnallisuus- ja käytettävyysongelmia. Arviointimenetelmän avulla voidaan todeta, että testausstrategiaa voidaan soveltaa erilaisiin skenaarioihin, joiden monimutkaisuus vaihtelee.

5.2 Artefaktin vertaaminen kirjallisuuteen

Artefaktin suunnittelussa ja toteuksessa oli koko ajan vahvasti läsnä ajatus siitä, että artefakti noudattaisi mahdollisimman hyvin kirjallisuudessa olevia määritelmiä testausstrategialle. Artefakti noudattaa varsin hyvin TMMi Foundationin (2022, s. 24–28), Kasurisen (2010) ja Kasurisen (2013, s. 71–73 ja s. 132–136) tutkimuksissa esittämiä määritelmiä testausstrategialle. Huomion arvoista on todeta, että kirjallisuudessa esiintyvät vaatimukset testausstrategialle ovat esimerkiksi TMMi Foundationin (2022, s. 24–28) tapauksessa peräisin

ISO/IEC/IEE 29119 (2022) standardista.

Artefakti sisältää TMMi Foundationin (2022, s. 24–28) määritelmän mukaisesti: määritelmän testauksen laajuudesta riippuen projektin eri tekijöistä: ohjeet testitapauksen, testaussuunnitelman ja testausraportin muodostamiselle sekä lähestymistapa milloin siirtyä manuaalisesta testaamisesta testiautomaation käyttöön. TMMi Foundationin (2022, s. 24–28) määritelmä jättää aihealueiden varsinaisen sisällön määrittelemisen hyvin vaillinaiseksi, joten emme voi ottaa kantaa vastaako artefakti sisällöllisesti sitä, mitä TMMi Foundationin (2022, s. 24–28) määritelmä edellyttää.

Myöskään Kasurinen (2013) ei määritelmässään ota suoraan kantaa testausstrategian varsinaiseen sisältöön vaan listaa aihealueita, joita testausstrategiassa tulisi olla. Kasurisen (2013, s. 71–73 ja s. 132–136) määrittelemistä aihealueista artefaktissa käsitellään muun muassa selitteen eri testaukseen liittyville termeille ja testausmenetelmät. Kasurinen (2013) määritelmässään korostaa, että testausstrategian tulisi olla vain ylätason määritelmä testaamiselle, jota myös artefaktissa on noudatettu.

Yhtenevää TMMi Foundationin (2022, s. 24–28) ja Kasurisen (2013, s. 71–73 ja s. 132–136) määritelmissä on se, että testausstrategiassa tulisi olla määriteltynä testauksen aloitus- ja lopetusehdot. Testauksen aloitus- ja lopetusehtoja ei kuitenkaan ole annettuna artefaktissa. Artefaktissa ei myöskään ole määriteltynä kriteereitä testitapauksien valitsemiselle, jotka Kasurisen (2013, s. 71–73 ja s. 132–136) mukaan kuuluisi osaksi testausstrategiaa. Sekä testauksen aloitus- ja lopetusehtojen että testitapauksien valintakriteeristön puuttuminen selittyy sillä, että artefaktin toteutusvaiheessa ei kyetty löytämään kaikille kohdeorganisaation projekteille yhteneviä testauksen aloitus- ja lopetusehtoja eikä kriteereitä testitapauksien valitsemiselle.

Artefakti pääpiirteiltään noudattaa hyvin kirjallisuudessa, kuten TMMi Foundationin (2022, s. 24–28) ja Kasurisen (2013, s. 71–73 ja s. 132–136) esittämiä määritelmiä testausstrategialle. Yksittäisiä aihealueita kuten kriteeristö testitapauksien valitsemiselle, joka on hyvin projektikohtainen, ei ole määriteltynä artefaktissa, vaan sen oletetaan olevan osa projektikohtaista testaussuunnitelmaa. Yksi jatkokehitysmahdollisuus olisi lisätä kriteeristö testitapauksien valitsemiselle osaksi testausstrategiaa, jotta se olisi yhtenäisempi projektien välillä.

5.3 Tutkimuskysymyksiin vastaaminen

Seuraavaksi syvennytään tutkimuksen ytimeen, jossa käsitellään tutkimuskysymykset. Ensimmäinen asetettu kysymys koskee kattavan Salesforce-testauksen vaatimuksia, joita tutkitaan tutustumalla asiaankuuluvaan kirjallisuuteen ja analysoimalla parhaita käytäntöjä testaamiseen Salesforce-kontekstissa. Toinen kysymys koskee sitä, kuinka kohdeorganisaatio voi parantaa testauskäytäntöjään parantaakseen kehittämiensä Salesforce-ohjelmistojen laatua. Vastaamalla näihin kysymyksiin toivomme tarjoavamme käyttökelpoisia oivalluksia, jotka auttavat organisaatioita parantamaan testausprosessejaan ja tuottamaan laadukkaampia ohjelmistoja.

Vastaamaan näihin tutkimuskysymyksiin kehitettiin artefakti, joka tarjoaa testausstrategian ja suuntaviivat kattavaan Salesforce-testaukseen. Artefakti perustuu kirjallisuuden esittämiin parhaisiin käytäntöihin ja ohjeisiin, ja se on suunniteltu vastaamaan haasteisiin, joita kohdeorganisaatio kohtaa Salesforce-kehitysprojektien testauksessa.

TK1: Mitä vaatimuksia sisältyy Salesforce-järjestelmän kattavaan testaamiseen?

Ensimmäisessä tutkimuskysymyksessä perehdytään siihen, mitkä ovat vaatimukset kattavalle Salesforce-testaukselle. Vastataksemme tähän kysymykseen teimme perusteellisen kirjallisuuskatsauksen ja analysoimme tässä tutkimuksessa kehitettyä artefaktia. Seuraavaksi esitetään pääkohdat ensimmäiseen tutkimuskysymykseen vastaamiseksi.

- Kirjallisuuden perusteella havaittiin, että kattavan testauksen tulisi kattaa kaikki ohjelmistokehityksen osa-alueet. Ohjelmistotestaus on siis jatkuva prosessi, eikä sen kuulu olla ns. pakollinen paha kattaen esimerkiksi vain yksikkötestausta pakollisen koodirivikattavuuden täyttämiseksi. (Seth ym. 2012)
- TMMi Foundationin (2022) mukaan kohdeorganisaation testaaminen on TMMi-mallin maturiteettitasojen mukaan hallittua, jos organisaation testaustoiminta on vakiintunutta ja hallittua sisältäen yhtenäiset testauskäytänteet ja testausstrategian.

Ensimmäiseen tutkimuskysymykseen vastaamiseksi havaitut vaatimukset eivät kuitenkaan vastaa itse tutkimuskysymykseen tarpeeksi kattavasti. Tutkimuskysymyksessä korostetaan nimenomaan Salesforce-kontekstin testauksen vaatimuksia, eikä ohjelmistotestauksen yli-

päätään. Edellä mainitut pääkohdat ensimmäiseen tutkimuskysymykseen vastaamiseksi tuovat kohdeorganisaatiolle lisäarvoa, mutta tutkimuksessa ei selviä mitä nimenomaan Salesforce-kehitykseen liittyvä testaus vaatisi. Myöskään artefakti ei tähän pystynyt vastaamaan, koska sillä pyrittiin täyttämään yhtenäisen ohjelmistotestauksen aukko kohdeorganisaatiossa.

TK2: Miten kohdeorganisaation ohjelmistotestauskäytänteitä pitäisi kehittää kehitettävien Salesforce-järjestelmien laadun parantamiseksi?

Toisella tutkimuskysymyksellä pyritään tunnistamaan, miten kohdeorganisaation testauskäytänteitä voidaan parantaa toteutettujen Salesforce-ohjelmistojen laadun parantamiseksi. Vastataksemme tähän kysymykseen analysoimme tässä tutkimuksessa kehitettyä artefaktia. Tämän tutkimuskysymyksen havainnot ja analyysi osoittavat, että kohdeorganisaation tulisi omaksua kattavampi testaustapa, joka kattaa kaikki ohjelmistokehityksen elinkaaren osa-alueet ja toteuttaa jatkuvan testausstrategian. Lisäksi organisaation tulisi panostaa testausautomaatioon ja hyödyntää Salesforcen tarjoamia testaustyökaluja yhtenäisellä tavalla. Tämän tutkimuskysymyksen nojalla kehitettiin tutkimuksen artefaktiksi testausstrategia kohdeorganisaatiolle.

Kehitetyssä testausstrategiassa on useita keskeisiä ominaisuuksia, jotka parantavat kohdeorganisaation testauskäytäntöjä. Se tarjoaa kattavan testausrungon projektin eri vaiheisiin alkaen työmääräarvioista aina raportointiin kattaen Salesforce-kehitysprojektien kriittisimmät osa-alueet. Testausstrategia helpottaa sen käyttäjän päätöksentekoa, koska se ohjaa tekemään valintoja numeeristen arvojen pohjalta, esimerkiksi kuvioissa 4 ja 5 esitetyn testaustason valintapolun mukaan, jonka tulokset yhteen laskiessa tietää valita oikean testaustason. Lisäksi testauksen dokumentointi on kohdeorganisaation kehityskohteena, jota testausstrategia tukee tarjoamalla raportointipohjat muun muassa testitapauksille, testaussuunnitelmalle ja testausraportille.

Voidaan todeta, että kehitetty testausstrategia osaltaan vie eteenpäin kohdeorganisaation ohjelmistotestauskäytänteitä. Se ei kuitenkaan yksinään ole koko totuus, vaikka esimerkiksi Grindalin, Offuttin ja Mellinin (2006) kyselytutkimuksessa todetaan, että aktiivisesti testausstrategiaa käyttävät organisaatiot investoivat muita yrityksiä enemmän aikaa testaamiseen erityisesti ohjelmistotuotannon alkuvaiheessa, jossa testaamisella on ohjelmiston laa-

dun kannalta eniten merkitystä Slaughterin, Harterin ja Krishnanin (1998) mukaan. Testausstrategian olemassa olo ei itsessään lisää investointeja ellei organisaatio tee niitä.

Testausstrategia myös kasvattaa koko organisaation maturiteettia testaamisen saralla, mikä vähentää testaamisen riippuvuutta yksittäisistä henkilöistä ja tasaa testaamisen tasoa kaikissa projekteissa (Grindal, Offutt ja Mellin 2006). Maturiteetin kasvaminen voidaan nähdä kehittävän kohdeorganisaation ohjelmistotestauskäytänteitä, mikä osaltaan osoittaa, että testausstrategia on keino parantaa Salesforce-järjestelmän laatua.

Tutkimuksen toisella tutkimuskysymyksellä pyritään parantamaan kohdeorganisaation testauskäytäntöjä Salesforce-ohjelmistojen laadun parantamiseksi. Tutkimuksessa kehitetty testausstrategia tarjoaa testausrungon projektin eri vaiheisiin ja auttaa testaustason valinnassa. Vaikka tutkimustulokset eivät ole aukottomia, ne viittaavat siihen, että testausstrategia parantaa organisaation ohjelmistotestauskäytänteitä ja kasvattaa koko organisaation maturiteettia testaamisen saralla. Kuten todettua, tämä voi johtaa parempaan laatuun Salesforce-ohjelmistojen tuotannossa ja vähentää mahdollisia vikoja ja virheitä.

Tässä aliluvussa käsiteltyjen tutkimuskysymysten tulokset osoittavat, että Salesforce-järjestelmän kattavaan testaamiseen sisältyy kokonaisvaltainen ja yhtenäinen testaustapa erilaiset muuttujat huomioon ottaen, joka on kohdeorganisaation testauskäytänteiden kehittämisen kannalta ensiarvoisen tärkeää. Havainnollistavien skenaarioiden ja kirjallisuuden tulkinnat korostavat kattavan testauslähestymistavan merkitystä korkealaatuisten Salesforce-ohjelmistojen varmistamisessa.

5.4 Reflektointi

Tässä aliluvussa reflektoidaan toteutettua artefaktia. Reflektoinnissa käsitellään, mitä haasteita artefaktin tekemiseen liittyi, havaitut kehityskohteet, rajoitukset ja jatkokehitysideoita testausstrategian kehittämiseksi.

Artefaktin suunnittelu, toteutus ja testaus olivat kattavat ja hyvin toteutettu. Sen kehitysprojektissa toimi kaksi Salesforce-kehittäjää varmistaakseen, että artefakti suunniteltiin ja toteutettiin tarvittavien standardien ja vaatimusten mukaisesti. Artefaktin kehitysprojektia lähes-

tyttiin yhteistyöhön perustuvalla ajattelutavalla. Tämä kommunikointia korostava yhteistyöprosessi mahdollisti perusteellisen vaatimusten ja suunnittelun analysoinnin sekä mahdollisuuden kyseenalaistaa toistensa päätökset ja varmistaa, että lopputuote on korkealaatuista, täyttää vaatimukset ja on kirjallisuuteen perustuen parhaita käytänteitä seuraava. Lisäksi kohdeorganisaation asettamat selkeät ja tarkat vaatimukset mahdollistivat kohdistetun kehitysprosessin ja lopputuotteen onnistuneen toimituksen.

Testausstrategian suunnittelu ja toteutus oli melko mutkatonta. Vaatimusmäärittelyyn ja suunnitteluun meni huomattava määrä aikaa, jotta pystyttiin toteuttamaan testausstrategiaa mahdollisimman yhteisymmärryksessä sen vaatimuksista ja sille asetettavista tavoitteista. Kohdeorganisaation asettamien tarpeiden ja vaatimusten selvitys testausstrategialle vaati muutaman tarkennusiteraation, mutta sitäkään ei pidetty suurena haasteena sujuvan kommunikoinnin ansiosta. Myöskään ajallisten resurssien kanssa ei ilmennyt haasteita tiukasta aikataulusta huolimatta tarkan suunnittelun ansiosta.

Artefaktin kattavasta luonteesta huolimatta sen käytön helppoudessa havaittiin havainnollistavissa skenaarioissa hieman parantamisen varaa. Strukturoidun dokumentin yksityiskohtainen luonne voi olla haaste nopeatempoisessa ympäristössä työskenteleville käyttäjille, koska se sisältää myös tiedot ja ohjeet kaikista muista testaustasoista valitun testaustason lisäksi. Artefakti ohjaa tehokkaasti projektin vaiheita testaamisen näkökulmasta, mutta selkeyden ja käytettävyyden kannalta olisi parempi näyttää vain yksi projektin vaihe kerrallaan ja ohjata selkeämmin seuraavaan vaiheeseen siirtymistä kognitiivisen kuorman vähentämiseksi (Hatfield ja Epstein 1985).

Havainnollistavaissa skenaarioissa havaittiin, että artefaktin testaustason valintapolku voisi hyötyä projektissa käytettyjen Sandboxien määrää koskevan kysymyksen lisäämisestä. Tämä antaisi artefaktille mahdollisuuden tarjota entistä tarkemmin räätälöityjä ohjeita kehitysputkien suunnittelussa ja testiautomaatiossa, mikä parantaa viime kädessä työmääräarvioiden tarkkuutta ja testaamisen suunnittelua.

Vaikka testausstrategia todettiin toimivaksi ja kohdeorganisaatiolle arvokkaaksi, on tärkeä huomioida myös testausstrategiaan liittyvät rajoitukset. Osa lähteistä ei ole ajankohtaisimpia, mikä mahdollisesti rajoittaa testausstrategian tarkkuutta ja lähestymistavan pätevyyt-

tä. Lisäksi kirjallisuus testaamisesta, erityisesti Salesforce-kontekstissa, on rajallista, mikä tarkoittaa, että Salesforce-spesifi näkökulma perustuu osittain vain viralliseen dokumentaatioon ajankohtaisen tieteellisen tutkimuksen sijaan. Toiseksi ajallisten resurssien vuoksi tehtiin vain kaksi havainnollistavaa skenaariota. Toisaalta tutkimuksen aikajänteen puitteissa useamman havainnollistavan skenaarion tai toisen DSRM-iteraation tekeminen ei olisi ollut optimaalista, eivätkä siten olisi tuoneet tutkimukselle lisäarvoa. Näistä rajoituksista huolimatta uskomme, että testausstrategia on arvokas resurssi kohdeorganisaatiolle ja antaa vankan pohjan tuleville testaustoimille.

Testausstrategiaan liittyvän tulevan työn kannalta testausstrategiaan voidaan tehdä parannuksia. Testaustason valintapolkua voidaan parantaa lisäämällä uusi päätöselementti, joka kysyy käyttäjältä, kuinka monta Sandboxia asiakasorganisaatiolla on käytössä. Näin testausstrategia voisi tarjota käyttäjälle tarkemman tuloksen testaustasosta perustuen asiakkaan kehitysputkeen. Lisäksi itse testausstrategiasta voidaan tehdä käyttökelpoisempi ja intuitiivisempi antamalla käyttäjälle konkreettisempaa ohjausta. Yksi mahdollinen ratkaisu tähän olisi luoda testausstrategia Experience Cloud -sivulle, joka suunniteltaisiin kohdeorganisaation käyttöön. Testausstrategia voisi näyttää tiedot vain nykyisestä kehitysvaiheesta ja esimerkiksi luoda testaustason valintapolku valintalistoilla parantaakseen sen käytettävyyttä ja saavutettavuutta (Hatfield ja Epstein 1985). Tämä lähestymistapa mahdollistaisi tehokkaamman yhteistyön ja viestinnän projektiryhmän välillä, mikä mahdollisesti johtaisi tehokkaampaan testaamiseen Salesforce-kehitysprojekteissa.

On tärkeää huomioida tutkimuksen validiteetin näkökulmasta, että tutkimuksen tulokset eivät ole täysin yleistettävissä muihin konteksteihin tai organisaatioihin. Tämän tutkimuksen otoskoko on rajoitettu yhteen kohdeorganisaatioon, joka ei edusta kaikkia Salesforce-konsultointiin ja kehitykseen keskittyneitä ohjelmistoyrityksiä. Tutkimuksen tekijät ovat kohdeorganisaation työntekijöitä, mikä on saattanut vaikuttaa artefaktin kehittämiseen ja arviointiin. Tutkimuksen tekijät ovat pyrkineet minimoimaan työsuhteesta aiheutuvaa puolueellisuutta kyseenalaistamalla pohdintoja ja perustelemalla väittämiä kirjallisuudella. Uskomme kuitenkin, että tutkimuksen havaintoja soveltaen muutkin organisaatiot voivat niistä hyötyä.

Esitetyt tulokset osoittavat, että artefakti on tehokas saavuttamaan sille asetetut tavoitteet.

Jatkokehitystä ajatellen artefakti voisi hyötyä parannuksista sen saavutettavuuteen ja testaustason valintapolun suunnitteluun sekä räätälöidyistä ohjeista projektin erityispiirteisiin. Kaiken kaikkiaan artefakti täyttää sille asetetut kriteerit ja se havaittiin testausprosessia optivoivana tekijänä havainnollistavien skenaarioiden aikana.

6 Johtopäätökset

Laadukkaan ohjelmiston tuottaminen on jatkuvaa kamppailua aikaa ja rahaa vastaan nykypäivän nopeatempoisessa ja kilpailullisessa ohjelmistoteollisuudessa. Sidosryhmien vaatimuksien täyttämiseksi ja positiivisen käyttäjäkokemuksen tarjoamiseksi laadukkaan ohjelmiston eteen on työskenneltävä koko ohjelmistokehitysprosessin ajan. Laatu ja eri laatutekijät on tunnistettu olennaisiksi elementeiksi ohjelmistokehityksessä, joiden varmistamiseksi ja kehittämiseksi ohjelmistotestaus on tehokas tapa (Seth ym. 2012).

Tutkimuksessa pyrittiin löytämään kohdeorganisaatiolle ratkaisu ongelmaan, kuinka Salesforce-järjestelmä tulisi kattavasti testata, jotta sidosryhmien vaatimukset ja positiivinen käyttäjäkokemus toteutuisi kohdeorganisaation kaikissa Salesforce-järjestelmissä. Ratkaisuksi ongelmaan tutkimuksessa kehitettiin testausstrategia, joka tutkimuksessa osoitetusti toimii kohdeorganisaatiolle tyypillisissä projekteissa testausta tukevana dokumenttina. Testausstrategiassa huomioidaan kohdeorganisaatiolle ja Salesforce-kehitykseen liittyvät erityispiirteet.

Testausstrategia tarjoaa kohdeorganisaatiolle hyvän ja tukevan pohjan lähteä jatkokehittämään sen ohjelmistotestauskäytänteitä. Hyvät ohjelmistotestauskäytänteet tarjoavat kohdeorganisaatiolle paremmat valmiudet tuottaa laadukkaita Salesforce-järjestelmiä nopeatempoisessa tekniikkakeskeisessä maailmassa, jossa laatu ja laadun varmistaminen ovat entistä keskeisempiä kilpailuvaltteja. Nämä mahdollistavat erottautumisen markkinassa, jossa useat kohdeorganisaation kaltaiset Salesforce kumppanit kilpailevat samoista projekteista.

Työtä tutkimuksen osalta liittyen Salesforce-kontekstiin on tämän tutkimuksen havaintojen perusteella jatkettava. Salesforce sekä koko sen ympärillä oleva ekosysteemi tarvitsee kehittyäkseen tutkittuun tietoon pohjautuvaa tutkimusta, jota Salesforce ja kohdeorganisaation kaltaiset kumppanit kykenevät tämän tutkimuksen tavoin kehittämään toimintaansa niin ohjelmistotestaamisen kuin muun ohjelmistotuotannon saralla.

Tutkimuksen aikana havaittuja jatkotutkimuskohteita kohdeorganisaation näkökulmasta voisi olla esimerkiksi: testausstrategian käytön arviointi muun muassa asiantuntijoiden näkökulmasta ja jatkokehittäminen tapaustutkimuksen avulla, testausstrategian käytettävyyden kehittäminen sekä testitapauksien valintakriteeristön muodostaminen.

Lähteet

Arachchi, S.A.I.B.S., ja Indika Perera. 2018. “Continuous Integration and Continuous Delivery Pipeline Automation for Agile Software Project Management” [kielellä en]. Teoksessa *2018 Moratuwa Engineering Research Conference (MERCon),* 156–161. Toukokuu. https://doi.org/10.1109/MERCon.2018.8421965.

Bassett, Tom. 2022. *Salesforce Sandbox Types & When To Use* [kielellä en], toukokuu. Viitattu 27. helmikuuta 2023. https://www.salesforceben.com/salesforce-sandbox/.

Boehm, B., ja V.R. Basili. 2001. “Software Defect Reduction Top 10 List” [kielellä en]. Conference Name: Computer, *Computer* 34, numero 1 (tammikuu): 135–137. ISSN: 1558-0814. https://doi.org/10.1109/2.962984.

Cennamo, Carmelo, Hakan Ozalp ja Tobias Kretschmer. 2018. “Platform Architecture and Quality Trade-offs of Multihoming Complements” [kielellä en]. Publisher: INFORMS, *Information Systems Research* (huhtikuu). Viitattu 2. helmikuuta 2023. https://doi.org/10.1287/isre.2018.0779.

Chaves Da Silva, Alexandre, Lucas Roberto Correa, Luiz Alberto Vieira Dias ja Adilson Marques Da Cunha. 2015. “A Case Study Using Testing Technique for Software as a Service (SaaS)” [kielellä en]. Teoksessa *2015 12th International Conference on Information Technology - New Generations,* 761–762. Las Vegas, NV, USA: IEEE, huhtikuu. https://doi.org/10.1109/ITNG.2015.133.

Choudhury, Musfiq Mannan, ja Paul Harrigan. 2014. “CRM to social CRM: the integration of new technologies into customer relationship management” [kielellä en]. Publisher: Routledge _eprint: https://doi.org/10.1080/0965254X.2013.876069, *Journal of Strategic Marketing* 22, numero 2 (helmikuu): 149–176. ISSN: 0965-254X, viitattu 20. helmikuuta 2023. https://doi.org/10.1080/0965254X.2013.876069.

Cockburn, A., ja J. Highsmith. 2001. “Agile software development, the people factor” [kielellä en]. Conference Name: Computer, *Computer* 34, numero 11 (marraskuu): 131–133. ISSN: 1558-0814. https://doi.org/10.1109/2.963450.

Cook, Andrew. 2022. *Hottest Salesforce VS Code Extensions* [kielellä en], joulukuu. Viitattu 6. maaliskuuta 2023. https://www.salesforceben.com/salesforce-vs-code-extensions/.

Coxon, Hayley. 2022. *Salesforce Environments: Which Do I Use and When?* [Kielellä en], heinäkuu. Viitattu 27. helmikuuta 2023. https://www.salesforceben.com/salesforce-environments-which-do-i-use-and-when/.

Fawcett, Andrew. 2017. *Force.com Enterprise Architecture* [kielellä en]. Google-Books-ID: dLkrDwAAQBAJ. Packt Publishing Ltd, maaliskuu. ISBN: 978-1-78646-505-4. https://books.google.com/books?hl=en&lr=&id=dLkrDwAAQBAJ&oi=fnd&pg=PP1&dq=fawcett+andrew+enterprise+architecture&ots=CcuCQMFyRK&sig=3FIr4dY0LKPSxvbDskItnt00x0s.

Foundation, The Dojo. 2005. *Workbench: About* [kielellä en]. Viitattu 8. helmikuuta 2023. https://workbench.developerforce.com/about.php.

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

Gotts, Ian. 2022. *Salesforce DevOps Center: A Deeper Dive* [kielellä en], marraskuu. Viitattu 27. helmikuuta 2023. https://www.salesforceben.com/salesforce-devops-center/.

Grindal, M., J. Offutt ja J. Mellin. 2006. "On the Testing Maturity of Software Producing Organizations" [kielellä en]. Teoksessa *Testing: Academic & Industrial Conference - Practice And Research Techniques (TAIC PART'06),* 171–180. Elokuu. https://doi.org/10.1109/TAIC-PART.2006.20.

Gupta, Abhinav, ja Ankit Arora. 2013. *Force.com Tips and Tricks* [kielellä en]. Google-Books-ID: PAZ86oOyw40C. Packt Publishing, Limited. ISBN: 978-1-84968-474-3.

Gupta, Radhika, Sahil Verma ja Kavita Janjua. 2018. "Custom Application Development in Cloud Environment: Using Salesforce" [kielellä en]. Teoksessa *2018 4th International Conference on Computing Sciences (ICCS),* 23–27. Elokuu. https://doi.org/10.1109/ICCS.2018.00010.

Harris, Ivan. 2022. *Beginning Salesforce DX: Versatile and Resilient Salesforce Application Development* [kielellä en]. 1. painos. Berkeley, CA: Apress, marraskuu. ISBN: 978-1-4842-8114-7, viitattu 6. maaliskuuta 2023. https://doi.org/10.1007/978-1-4842-8114-7.

Hatfield, Gary, ja William Epstein. 1985. "The Status of the Minimum Principle in the Theoretical Analysis of Visual Perception" [kielellä en]. 97:155–186. https://doi.org/10.1037/0033-2909.97.2.155.

Hevner, Alan R., Salvatore T. March, Jinsoo Park ja Sudha Ram. 2004. "Design Science in Information Systems Research" [kielellä en]. Publisher: Management Information Systems Research Center, University of Minnesota, *MIS Quarterly* 28:75–105. Viitattu 29. joulukuuta 2022. https://doi.org/10.2307/25148625.

ISO/IEC 25000. 2014. *ISO/IEC 25000:2014* [kielellä en]. Maaliskuu. Viitattu 15. tammikuuta 2023. https://www.iso.org/standard/64764.html.

ISO/IEC/IEE 29119. 2022. "ISO/IEC/IEEE 29119-1:2022" [kielellä en]. *ISO* (tammikuu): 47. Viitattu 30. tammikuuta 2023. https://www.iso.org/standard/81291.html.

Kasurinen, Jussi. 2010. "Elaborating Software Test Processes and Strategies" [kielellä en]. Teoksessa *Verification and Validation 2010 Third International Conference on Software Testing,* 355–358. ISSN: 2159-4848. Huhtikuu. https://doi.org/10.1109/ICST.2010.25.

———. 2013. *Ohjelmistotestauksen käsikirja* [kielellä fi]. 1. painos. Jyväskylä: Docendo. ISBN: 978-952-5912-99-9. https://docendo.fi/sivu/tuote/ohjelmistotestauksen-kasikirja/2485588.

Kiranmayee, Tadepalli Sarada. 2015. "A Survey on the Role of Cloud Computing in Social Networking Sites" [kielellä en]. 6. ISSN: 0975-9646. https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=d41e73812c443e220dae61e650af13951ec77849.

Mathew, Reena, ja Ryan Spraetz. 2009. "Test Automation on a SaaS Platform" [kielellä en]. Teoksessa *2009 International Conference on Software Testing Verification and Validation,* 317–325. ISSN: 2159-4848. Huhtikuu. https://doi.org/10.1109/ICST.2009.46.

Mazalon, Lucy. 2022. *Your Guide to Lightning Web Components: Let's Explore LWC* [kielellä en]. Article, joulukuu. Viitattu 20. helmikuuta 2023. https://www.salesforceben.com/lightning-web-components/.

McCall, Jim A., Paul K. Richards ja Gene F. Walters. 1977. *Factors in Software Quality. Volume I. Concepts and Definitions of Software Quality* [kielellä en]. Tekninen raportti. Section: Technical Reports. Marraskuu. Viitattu 6. helmikuuta 2023. https://apps.dtic.mil/sti/citations/ADA049014.

Mithas, Sunil, M.S. Krishnan ja Claes Fornell. 2005. "Why Do Customer Relationship Management Applications Affect Customer Satisfaction?" [Kielellä en]. *Journal of Marketing* 69, numero 4 (lokakuu): 201–209. ISSN: 0022-2429, 1547-7185, viitattu 20. helmikuuta 2023. https://doi.org/10.1509/jmkg.2005.69.4.201.

Neumann, Georg. 2020. *How to Write Stable Selenium Tests for Lightning UI* [kielellä en]. Section: Blog, helmikuu. Viitattu 27. helmikuuta 2023. https://developer.salesforce.com/blogs/2020/02/how-to-write-stable-selenium-tests-for-lightning-ui.

O'Leary, Stacy. 2022. *Salesforce Workbench: 6 Ways to Use Workbench in Salesforce* [kielellä en], helmikuu. Viitattu 8. helmikuuta 2023. https://www.salesforceben.com/salesforce-workbench/.

O'Regan, Gerard. 2011. *Introduction to Software Process Improvement* [kielellä en]. Undergraduate Topics in Computer Science. London: Springer. ISBN: 978-0-85729-172-1, viitattu 6. maaliskuuta 2023. https://doi.org/10.1007/978-0-85729-172-1.

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

———. 2017. "An integration of salesforce.com with Twitter: A case of AppExchange" [kielellä en]. Teoksessa *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT),* 1–6. Helmikuu. https://doi.org/10.1109/ICECCT.2017.8117882.

Paulk, M.C., B. Curtis, M.B. Chrissis ja C.V. Weber. 1993. “Capability maturity model, version 1.1”. Conference Name: IEEE Software, *IEEE Software* 10, numero 4 (heinäkuu): 18–27. ISSN: 1937-4194. https://doi.org/10.1109/52.219617.

Peffers, Ken, Marcus Rothenberger, Tuure Tuunanen ja Reza Vaezi. 2012. “Design Science Research Evaluation” [kielellä en]. Teoksessa *Design Science Research in Information Systems. Advances in Theory and Practice,* 398–410. Springer, Berlin, Heidelberg. Viitattu 29. joulukuuta 2022. https://doi.org/10.1007/978-3-642-29863-9_29.

Peffers, Ken, Tuure Tuunanen, Marcus A. Rothenberger ja Samir Chatterjee. 2007. “A Design Science Research Methodology for Information Systems Research” [kielellä en]. Publisher: Routledge _eprint: https://doi.org/10.2753/MIS0742-1222240302, *Journal of Management Information Systems* 24, numero 3 (joulukuu): 45–77. ISSN: 0742-1222, viitattu 29. joulukuuta 2022. https://doi.org/10.2753/MIS0742-1222240302.

Riungu, Leah Muthoni, Ossi Taipale ja Kari Smolander. 2010. “Research Issues for Software Testing in the Cloud” [kielellä en]. Teoksessa *2010 IEEE Second International Conference on Cloud Computing Technology and Science,* 557–564. Marraskuu. https://doi.org/10.1109/CloudCom.2010.58.

Runciman, David. 2022. *What is Salesforce DX? The Definitive Guide* [kielellä en], helmikuu. Viitattu 8. helmikuuta 2023. https://www.salesforceben.com/what-is-salesforce-dx-the-definitive-guide/.

Salesforce. 2022. *Get Up and Running with DevOps Center Today* [kielellä en]. Section: Blog, kesäkuu. Viitattu 8. helmikuuta 2023. https://developer.salesforce.com/blogs/2022/06/get-up-and-running-with-devops-center-today.

———. 2023a. *Before You Begin | Salesforce CLI Setup Guide | Salesforce Developers* [kielellä en]. Article. Viitattu 5. maaliskuuta 2023. https://developer.salesforce.com/docs/atlas.en-us.sfdx_setup.meta/sfdx_setup/sfdx_setup_intro.htm.

———. 2023b. *Clicks Not Code: Benefits of Declarative Vs. Imperative Programming* [kielellä en]. Viitattu 19. helmikuuta 2023. https://www.salesforce.com/products/platform/best-practices/declarative-programming-vs-imperative-programming/.

Salesforce. 2023c. *Introducing Lightning Web Components - Salesforce Lightning Component Library* [kielellä en]. Documentation. Viitattu 5. maaliskuuta 2023. https://developer.salesforce.com/docs/component-library/documentation/en/lwc.

———. 2023d. *Sandbox Licenses and Storage Limits by Type* [kielellä en]. Viitattu 24. helmikuuta 2023. https://help.salesforce.com/s/articleView?id=sf.data_sandbox_environments.htm&type=5.

———. 2023e. *Supported Scratch Org Editions and Allocations | Salesforce DX Developer Guide | Salesforce Developers* [kielellä en]. Viitattu 27. helmikuuta 2023. https://developer.salesforce.com/docs/atlas.en-us.sfdx_dev.meta/sfdx_dev/sfdx_dev_scratch_orgs_editions_and_allocations.htm.

———. 2023f. *Test Class | Apex Reference Guide | Salesforce Developers* [kielellä en]. Documentation. Viitattu 24. helmikuuta 2023. https://developer.salesforce.com/docs/atlas.en-us.apexref.meta/apexref/apex_methods_system_test.htm.

———. 2023g. *Understanding Testing in Apex | Apex Developer Guide | Salesforce Developers* [kielellä en]. Documentation. Viitattu 27. helmikuuta 2023. https://developer.salesforce.com/docs/atlas.en-us.apexcode.meta/apexcode/apex_testing_intro.htm.

———. 2023h. *What is CRM?* [Kielellä en]. Viitattu 20. helmikuuta 2023. https://www.salesforce.com/crm/what-is-crm/.

Seth, Frank Philip, Erja Mustonen-Ollila, Ossi Taipale ja Kari Smolander. 2012. “Software Quality Construction: Empirical Study on the Role of Requirements, Stakeholders and Resources” [kielellä en]. Teoksessa *2012 19th Asia-Pacific Software Engineering Conference,* 2:17–26. ISSN: 1530-1362. Joulukuu. https://doi.org/10.1109/APSEC.2012.119.

———. 2015. “Software quality construction in 11 companies: an empirical study using the grounded theory” [kielellä en]. *Software Quality Journal* 23, numero 4 (joulukuu): 627–660. ISSN: 1573-1367, viitattu 15. tammikuuta 2023. https://doi.org/10.1007/s11219-014-9246-2.

Slaughter, Sandra A., Donald E. Harter ja Mayuram S. Krishnan. 1998. “Evaluating the cost of software quality” [kielellä en]. *Communications of the ACM* 41, numero 8 (elokuu): 67–73. ISSN: 0001-0782, viitattu 6. helmikuuta 2023. https://doi.org/10.1145/280324.280335.

Sneha, Karuturi, ja Gowda M Malle. 2017. “Research on software testing techniques and software automation testing tools” [kielellä en]. Teoksessa *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS),* 77–81. Elokuu. https://doi.org/10.1109/ICECDS.2017.8389562.

Soni, Krutarth, ja Brijesh Vala. 2017. “Roadmap to salesforce security governance & salesforce access management” [kielellä en]. Teoksessa *2017 Second International Conference on Electrical, Computer and Communication Technologies (ICECCT),* 1–4. Helmikuu. https://doi.org/10.1109/ICECCT.2017.8117831.

Taipale, Ossi, Jussi Kasurinen, Katja Karhu ja Kari Smolander. 2011. “Trade-off between automated and manual software testing” [kielellä en]. *International Journal of System Assurance Engineering and Management* 2, numero 2 (kesäkuu): 114–125. ISSN: 0976-4348, viitattu 15. tammikuuta 2023. https://doi.org/10.1007/s13198-011-0065-6.

Tarasi, Crina. 2007. “Managing Customer Relationships” [kielellä en]. *Review of Marketing Research* 3 (helmikuu): 3–38. ISSN: 978-0-7656-1306-6. https://doi.org/10.1108/S1548-6435(2007)0000003005.

TMMi Foundation. 2022. “Test Maturity Model integration - R1.3”, https://www.tmmi.org/download/tmmi-framework-r1-3-pdf/?wpdmdl=6389&masterkey=6387a3132ef3b.

———. 2023. *TMMi Model* [kielellä en]. Viitattu 6. maaliskuuta 2023. https://www.tmmi.org/tmmi-model/.

Veenendaal, Erik van, Vahid Garousi ja Michael Felderer. 2022. “Motivations for and Benefits of Adopting the Test Maturity Model integration (TMMi)”. Teoksessa *Software Quality: The Next Big Thing in Software Engineering and Quality,* toimittanut Daniel Mendez, Manuel Wimmer, Dietmar Winkler, Stefan Biffl ja Johannes Bergsmann, 13–19. Lecture Notes in Business Information Processing. Cham: Springer International Publishing. ISBN: 978-3-031-04115-0. https://doi.org/10.1007/978-3-031-04115-0_2.

Williams, David S. 2014. *Connected CRM: Implementing a Data-Driven, Customer-Centric Business Strategy* [kielellä en]. Google-Books-ID: CyXnAgAAQBAJ. John Wiley & Sons, helmikuu. ISBN: 978-1-118-86319-0. https://books.google.com/books?hl=en&lr=&id=CyXnAgAAQBAJ&oi=fnd&pg=PT7&dq=williams+connected+crm+2014&ots=6E4qF1ioG7&sig=A_E5pRvXAv3eFVVy-WMWiO6zY8A.