jamk.fi

Difs-sovellusten toimituskäytänteiden analysointi ja kehitys

Case: Digia Financial Solutions

Juha Puskala

Opinnäytetyö
Tammikuu 2017
Tekniikan ja liikenteen ala
Insinööri (AMK), ohjelmistotekniikan tutkinto-ohjelma

Jyväskylän ammattikorkeakoulu
JAMK University of Applied Sciences

jamk.fi

Kuvailulehti

Tekijä(t)	Julkaisun laji	Päivämäärä
Puskala, Juha	Opinnäytetyö, AMK	5/2017
	Sivumäärä 60	Julkaisun kieli Suomi
		Verkkojulkaisulupa myönnetty: x

Työn nimi

Difs-sovellusten toimituskäytänteiden analysointi ja kehitys

Case: Digia Financial Solutions

Tutkinto-ohjelma

Ohjelmistotekniikka

Työn ohjaaja(t)

Esa Salmikangas, Jouni Huotari

Toimeksiantaja(t)

Digia Finland Oy/Financial Solutions

Tiivistelmä

Ohjelmistojen jatkuvaa ja automatisoitua toimittamista pidetään tänä päivänä yhtenä laadukkaan ja kilpailukykyisen ohjelmistokehityksen mittareina. Tehtävänä oli lähteä historiallisen katsauksen kautta tutkimaan yleisesti hyvänä pidettyjä ohjelmistokehitys- ja ohjelmistotuotantomalleja ja saavutetun ymmärryksen kautta analysoida ja kehittää Digia Financial Solutionsin tuottamien ohjelmistojen toimituskäytänteitä.

Teoriatutkimus toteutettiin ensin, jonka jälkeen lähdettiin tutkimaan organisaation vallitsevia toimituskäytänteitä ja siihen käytettäviä työkaluja. Toimituskäytänteissä hyväksi havaittiin TeamCityn, Jiran sekä Difs kehittäjien itse toteuttaman muutoshistoriatyökalun käyttö.

Nykytilanteen selvityksen jälkeen päätettiin lähteä toteuttamaan Windows Installer asennuspakettia toimitettavalle tuotteelle. Installer-paketin tuottamiseen valittiin Advanced Installer ohjelmisto.

Advanced Installerilla toteutettiin Windows Installer-paketti, jonka avulla Financial Solutionsin tuotteet pystyy asentamaan ja päivittämään Windows palvelimelle. TeamCityyn toteutettiin kolmannen osapuolen lisäosan avulla käännösintegraatio niin, että Installer-pakettien luonti saatiin automatisoitua.

Toteutettu asennuspaketti havainnollistaa Windows Installerin mahdollisuudet mutta kaikkia ongelmia sen avulla ei kuitenkaan saatu ratkaistua. Ohjelmiston konfiguraatio- ja tietokantamuutoksien hallintaan ei tämän työn puitteissa löydetty ratkaisuja, joten ne jäivät vielä tulevaisuuden haasteiksi.

Avainsanat (asiasanat)

Agile, Lean, DevOps, TeamCity, Advanced Installer, Continuous Delivery, Windows Installer

Muut tiedot

jamk.fi

Description

Author(s)	Type of publication	Date
Puskala, Juha	Bachelor's thesis	5/2017
		Language of publication: Finnish
	Number of pages: 60	Permission for web publication: x

Title of publication

Analysis and development of Difs-software delivery practices

Case: Digia Financial Solutions

Degree programme

Software Engineering

Supervisor(s)

Salmikangas Esa, Huotari Jouni

Assigned by

Digia Finland Oy/Financial Solutions

Abstract

These days the automated Continuous Integration of software is considered as one of the meters of competitive and qualitative software development. Through the research of historical overview of commonly accepted software development practices, the task was to achieve knowledge that could then be applied to the analysis and development of the delivery practices at Digia Financial Solutions.

The theoretical study was constructed first and after that the organization's existing delivery practices and tools were examined. The usage of tools such as TeamCity, Jira and a change log tool created by the Difs developers was considered as a good practice.

After the examination, a construction of a Windows Installer package of the delivered software was started. For the creation of the Installer, a tool called Advanced Installer was selected.

With the Advanced Installer, a Windows Installer package was created. From the package one could install and upgrade the Difs software to a Windows server. With the help of a third-party plugin to TeamCity, a build configuration enabling the automated build of the installer package was also created.

The created installer package provides information about the possibilities of a Windows Installer; however, all issues were not solved. A solution to the management of configuration and database changes was not provided so these challenges remain to be resolved in the future.

Keywords/tags (subjects)

Agile, Lean, DevOps, TeamCity, Advanced Installer, Continuous Delivery, Windows Installer

Miscellaneous

Sisältö

Kuviot

Taulukot

Termit ja käsitteet

Artefakti (Binääripaketti)	Varastoitu kokonaisuus, joka sisältää käännöksen tuottamat binääritiedostot
Asennuspaketti	Asennuspaketilla viitataan Windows Installer-pakettiin, joka on muodostettu toimituspaketin tiedostoista.
Binääritiedosto	Binääritiedosto on tietokoneen luettavaksi tarkoitettu tiedosto, joka voi sisältää millaista tietoa tahansa.
CD	Continuous Delivery eli jatkuva ohjelmiston toimittaminen asiakkaalle
CI	Continuous Integration eli muutosten jatkuva integrointi olemassa olevaan tuotteeseen
Difs	Digia Financial Solutions
IIS	Internet Information Services. Microsoftin kehittämä palvelinohjelmistokokonaisuus, joka on tarkoitettu käytettäväksi Windows-pohjaisissa palvelimissa.
Kääntäminen (Build)	Prosessi, joka muuntaa ihmiselle helpossa muodossa olevan lähdekoodin tietokoneen ymmärtämään muotoon.
Relaatiotietokanta	Relaatiomallissa tietokantaan tallennettava data esitetään järjestettyinä äärellisinä listoina, jotka on ryhmitelty relaatioiksi.
Skripti	Komentosarja, jolla automatisoidaan tehtäviä ilman, että tarvitaan varsinaisia ohjelmointikieliä
Syntaksi	Tarkastelee merkkiyhdistelmiä. Ohjelmointikielissä syntaksiin kuuluu varattujen sanojen ja lauseiden tunnistus
Toimituspaketti	Toimituspaketilla viitataan koottuun pakettiin, joka sisältää kaikki asiakkaalle toimitettavat tiedostot.

Versionhallinta	Keskitetty tietovarasto, johon voidaan tallentaa esimerkiksi dokumentteja, ohjelmistojen lähdekoodeja, konfiguraatioita tai muuta versiointia vaativaa informaatiota.
Wiki	Yhteisön ylläpitämä tietystä aiheesta hyödyllistä tietoa sisältävä sivustokokonaisuus
Windows Service	Windows ohjelma, jota operoidaan taustaprosessina ilman käyttäjän aktiivista interaktiota.

1 Johdanto

1.1 Tehtävä ja tavoitteet

Työn tehtävänä on tutkia erilaisia ohjelmistokehitysmenetelmiä ja prosesseja sekä ymmärtää, mitä kaikkea ohjelmiston toimittamiseen kuuluu, kun se tehdään laadukkaasti. Tarkoituksena on historiallisen teoriakatsauksen kautta johdatella ja ymmärtää modernin nykymaailman muuttuneita tarpeita ja peilata sitä ohjelmistojen toimittamisen vaatimuksiin.

Konkreettisia hyötyjä, joita työn lopputuloksilla tavoitellaan ovat optimoinnin ja uusien prosessien kautta tulevat säästöt sekä laadunparannus. Yhtenä työn tavoitteena voidaan pitää myös tiedon saamista siitä, mihin suuntaan työtapojen ja prosessien kehitystä tulisi viedä. Työn onnistumista voidaan siis yhtä lailla mitata sillä, kuinka toimituskäytänteiden tulevaisuuden kehityssuunnat pystytään kartoittamaan.

Tavoitteena on toteuttaa jokin konkreettinen prosessimuutos, parannus tai uudistus niin, että organisaation toimituskäytänteitä saadaan vietyä optimoidumpaan suuntaan.

1.2 Toimeksiantaja

”*Digia on kannattavasti kasvava IT-palveluyritys, joka auttaa asiakkaitaan hyödyntämään digitaalisuuden mahdollisuudet.* ” (Digia yrityksenä 2016). Digialla työskentelee yhteensä noin 900 työntekijää Suomessa ja Ruotsissa ja organisaation liikevaihto vuonna 2016 oli 86.5 miljoonaa euroa. Yhtiö on listattuna NASDAQ Helsingissä. (Digia yrityksenä 2016.)

Digia Financial Solutions on osa Digia pörssiyhtiötä ja sen alaisuudessa työskentelee yhteensä noin 100 työntekijää Jyväskylässä, Tampereella ja Helsingissä. Financial Solutions kehittää muun muassa pankeille ja rahoituslaitoksille ohjelmistoja heidän liiketoimintansa harjoittamista varten.

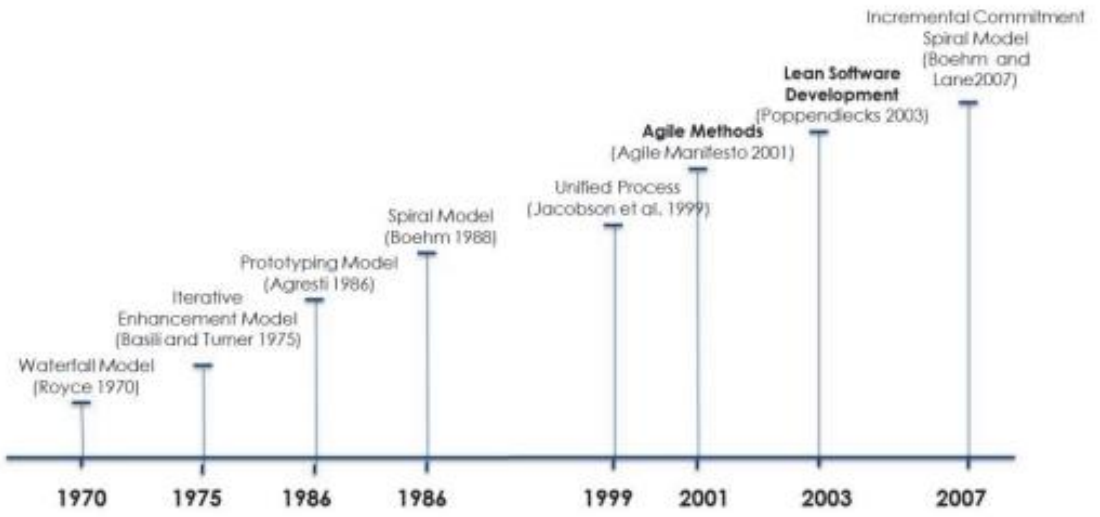
Digia Financial Solutions keskittyy Difs-tuoteperheen ympärille, johon kuuluu neljä ydintuotetta ja muutamia erilaisia rajapintatoteutuksia näiden lisäksi. Opinnäytetyössä on tarkoitus kartoittaa Digia Financial Solutions organisaation tuotteiden tämänhetkiset toimituskäytänteet ja tutkia, miten niitä voitaisiin kehittää.

Toimituskäytänteitä lähdetään tutkimaan, koska oletetaan, että niissä on parannettavaa. Oletetaan, että Difs-tuotteiden toimituskäytänteitä analysoimalla ja kehittämällä voidaan saavuttaa merkittäviä sekä organisaatiollisia että liiketoiminnallisia hyötyjä. Käytänteiden ja prosessien kehittämisen oletetaan vievän organisaatiota eteenpäin myös työkulttuurin osalta.

2 Ohjelmistokehityksen menetelmät ja ideologiat

2.1 Ohjelmistokehitysmallien historiallinen kehitys

Ohjelmistokehityksen menetelmät ja ideologiat ovat kehittyneet vuosikymmenten saatossa klassisesta, Roycen (1970) esittelemästä vesiputousmallista aina nykypäivän ketteriin menetelmiin ja niiden erilaisiin sovelluksiin kuten esimerkiksi Scrumbaniin (Cockburn & HighSmith 2016). Vanhempia kehitysmenetelmiä voidaan pitää nykypäivän standardeilla mitattuna raskaina, dokumentaatio-orientoituneina ja vahvojen prosessien ympärille keskittyvinä ideologioina (Rodríguez 2013). Menetelmät ovat sittemmin kehittyneet selvästi ihmislähtöisempään suuntaan ja malleihin, joissa ohjelmistojen pilkkominen yhä pienempiin ja helpommin hallittaviin osiin on tavanomaista. Myös asiakkaan läsnäolon merkitys ohjelmistoprojekteissa on kasvanut, mikä näkyy hyvin ketterien menetelmien periaatteissa. (Rodríguez 2013.)



Kuvio 1. Ohjelmistokehitysmenetelmien aikajana. (Rodríguez 2013.)

Kuviossa 1. näkyvälle aikajanalle on asetettu merkittävänä pidettyjä ohjelmistokehitysmenetelmiä ja niiden alullepanijoina pidettyjä tahoja. Kehitysmenetelmissä 1970-luvulta eteenpäin selkeänä trendinä on ollut ajatus iteratiivisuudesta, jota voidaan pitää yhtenä nykypäivän ohjelmistokehitysprosessien kulmakivistä *(*Moniruzzaman & Hossain n.d.) Iteratiivisuuden eli sykleittäin tapahtuvan ohjelmistokehityksen syntyhetkenä voidaan pitää Basilin ja Turnerin julkaisua Iterative Enhacement Model vuodelta 1975.

Vaikka iteratiiviset ohjelmistokehitysmenetelmät ja siitä edelleen kehittyneet ketterät Agile-menetelmät ovatkin saaneet suurta suosiota viime vuosikymmenien aikana, on hyvä silti pitää mielessä muutama helposti unohtuva seikka. Boehm (2002) kirjoittaa ketterien ja vanhojen, suunnittelujohtoisten menetelmien eroista ja painottaa, että molemmille työskentelytavoille on vieläkin oma paikkansa ja aikansa vallitsevan ohjelmistokehityksen maailmassa. Boehm painottaa erityisesti suunnittelujohtoisten menetelmien varmuutta, ennustettavuutta, toistettavuutta ja optimoinnin mahdollisuuksia. Hän kuitenkin myöntää vanhojen menetelmien heikkoudet tarjota nopeita ratkaisuja alati muuttuvan yhteiskunnan tarpeisiin, ja tätä tarvetta voidaankin pitää yhtenä ketterien kehitysmenetelmien suurimmista alullepanijoista (Highsmith 2002).

2.2 Ketterät kehitysmenetelmät

Ketterä ohjelmistokehitys, kansainväliseltä nimeltään Agile-kehitys sai virallisen alkunsa vuonna 2001 Agile Manifesto nimisen teoksen julkaisun myötä. Agile Manifesto on seitsemäntoista ohjelmistoalan asiantuntijan hyväksymä viitekehys Agile-malliseen eli ketterään ohjelmistokehitykseen. Se tiivistää itsensä lauseeseen: 'We are uncovering better ways of developing software by doing it and helping others do it'. (Fowler 2001.) Vapaasti suomennettuna: Esittelemme parempia keinoja ohjelmistojen kehittämiseen kokeilemalla itse ja auttamalla muita kokeilemaan.

Agile-menetelmien synty ja yleistyminen olivat vastine vallitsevien markkinoiden yhä nopeammin muuttuviin tarpeisiin, joita perinteiset ohjelmistokehitysmenetelmät eivät pystyneet tarjoamaan (Highsmith 2002). Agile Manifeston neljä kiteyttävää lausetta kuulutavat:

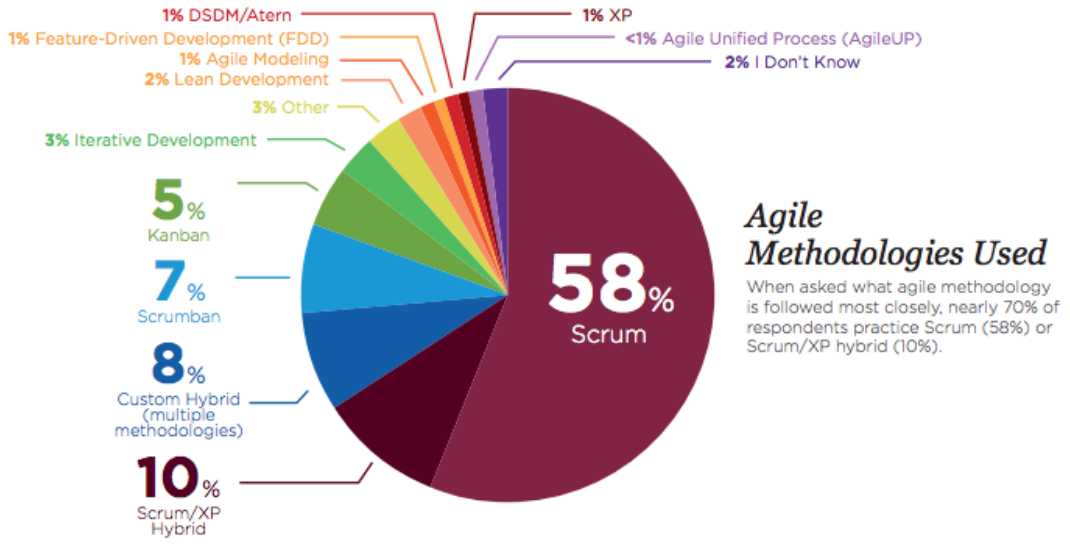
> *1. Individuals and interactions over processes and tools. 2. Working software over comprehensive documentation. 3. Customer collaboration over contract negotiation. 4. Responding to change over following a plan. (Fowler 2001.)*

Näiden neljän ytimekkään lauseen lisäksi manifestossa listataan myös kaksitoista ohjelmistokehityksen perusperiaatetta, joiden kautta se tarjoaa vaihtoehtoisen ja kevyemmän lähestymistavan vanhoille ja raskaina pidetyille kehitysmenetelmille. Jotkut ovat kritisoineet Agilen perusperiaatteita siitä, että ne saattavat johtaa vastuuttomuuteen, ja toimia tekosyynä hyvien ohjelmistokehityksen peruskäytänteiden laiminlyömiseen. (Rodríguez 2013.) Monet ovat silti ymmärtäneet, että Agile-menetelmien käyttöä voi hyvinkin soveltaa ja että niitä voi onnistuneesti yhdistellä raskaampiin, suunnittelujohtoisiin menetelmiin (Rodríguez 2013). Manifestossa todetaan seuraavasti:

> *The agile methodology movement is not anti-methodology; in fact, many of us want to restore credibility to the word. We also want to restore a balance: We embrace modeling, but not merely to file some diagram in a dusty corporate repository. We embrace documentation, but not to waste reams of paper in never-maintained and rarely-used tomes. (Fowler 2001).*

Sikäli, kun Agile Manifesto tarjoaa teoreettisen pohjan ketterän ohjelmistokehityksen ideologialle, on sen pohjalta syntynyt useita erilaisia käytännön toteutuksia

(Rodríguez 2013). Kuviossa 2 esitellään vuoden 2016 'Agile methodologies used' kyselyyn vastanneiden yritysten ketterien menetelmien käyttöä (Stage Of Agile 2016).



Kuvio 2. Ketterien menetelmien käyttö. (Stage of Agile 2016)

Kuten kuvaajasta voidaan nähdä, Scrum ja sen erilaiset sovellukset ehdottomasti dominoiva ohjelmistokehitysmalli alalla. Fundamentaalisina eroina ketterien kehitysmenetelmien välillä voidaan pitää niiden kokonaisvaltaista vaikutusta tekemiseen (Moniruzzaman & Hossain, n.d.) Jotkut menetelmät ovat enemmänkin hyviä käytänteitä, kun toiset taas enemmänkin projektihallinnan malleja. Scrumia voidaan siis pitää enemmänkin ohjelmistokehitys**prosessin** hallinnan työkaluna ja ehkä siksi se onkin nykypäivän käytetyin ketterä ohjelmistokehitysmenetelmä.

Scrum keskittyy kehitystiimin työskentelytapoihin siten, että ohjelmistojen tekeminen olisi mahdollisimman ketterää ja että kehitystiimi pystyisi vastaamaan asiakkaan muuttuviin vaatimuksiin. Scrumin perusperiaatteen mukaan toteutettava tuote tai ohjelmisto määritellään käyttäjätarinoiden ja ominaisuuksien mukaan niin, että ne on kuvattu asiakkaan näkökulmasta. Kehitys tapahtuu inkrementaalisissa sykleissä, joita kutsutaan sprinteiksi.

Jokaisella sprintillä on oma päämääränsä, joka määritellään Sprint planningissä, eli suunnittelupalaverissa. Suunnittelupalaverissa sprintille valitaan tietty määrä työstettäviä asiakastarinoita eli asiakkaan perspektiivistä määriteltyjä työtehtäviä. Oppikirja-

maisessa Scrum-työskentelyssä työtehtäviä ei vaihdeta sprintin aikana, vaan työlistoille otetut tehtävät pysyvät samoina aina sprintin loppuun asti. Sprintin pituus saattaa olla esimerkiksi kaksi viikkoa. (Rodríguez 2013.)

Jokaisen sprintin jälkeen pidetään Sprint review ja Sprint retrospective. Review'n eli katselmoinnin tarkoitus on analysoida projektin etenemistä ja demonstroida sen tämänhetkistä tilaa. Retrospectiven tarkoitus taas on peilata esimerkiksi työskentelytapoja ja miettiä, olisiko niissä jotain parannettavaa. Scrum-mallin pääroolit ovat projektin omistaja (Project owner), itse Scrum-tiimi ja sen muodollinen johtaja (Scrum master). Projektin omistajan tehtävä on toimia asiakasrajapinnassa ja priorisoida sprinteille otettavia työtehtäviä sekä lopulta hyväksyä ne valmiiksi.

Scrum-tiimi on itseohjautuva, ja ideana on, että se saa itse päättää parhaaksi katsomansa työskentelytavat, jotta tavoitteet saavutettaisiin. Scrum masterin tehtävänä taas on valvoa, että Scrumin sääntöjä noudatetaan, ja poistaa mahdollisia työntekoa haittaavia esteitä, mikäli niitä ilmaantuu. (Rodríguez 2013.)

2.3 Agile-metodien kritiikkiä

Ketterät kehitysmenetelmät ovat tuoneet ratkaisuja vallitsevan maailman alati muuttuviin tarpeisiin mutta eivät suinkaan ole jääneet ilman kritiikkiä. Agile-metodeita on kritisoitu muun muassa niiden toimimattomuudesta suurissa tiimeissä ja suurissa ohjelmistotuotteissa. Näkökulmia on annettu myös siitä, että Agile-menetelmät vaativat ohjelmistokehitystä ostavalta asiakkaalta paljon panostusta, mikä ei välttämättä aina ole mahdollista. (Awad 2005.)

Agilen perusperiaatteisiin kuuluu arvon tuottaminen asiakkaalle mahdollisimman nopeasti. Awad (2005) viittaa Barry Boehmn tekstiin: "Overfocus on early results in large systems can lead a major rework when the architecture doesn't scale up". Yliampuva arvon tuottaminen asiakkaalle projektin alkuvaiheessa saattaa siis aiheuttaa suuriakin vastoinkäymisiä ohjelmiston elinkaaren myöhemmissä vaiheissa, mitä ei helposti tule ajatelleeksi.

Ketterät menetelmät eivät myöskään välttämättä sovellu kriittisiin sovelluksiin ja suunnittelu-johtoiset metodit ovat joidenkin mielestä parempia vakaiden ja kriittisten ohjelmistojen tuottamiseen. Awad (2005) viittaa Scott Amblern tekstiin: "I would

be leery of applying agile modeling to life-critical systems". Awad (2005) kirjoittaa, kuinka Agile-metodien parikoodaukset ja koodikatselmoinnit eivät välttämättä ole tarpeeksi kattavia tämän tyyppisten ohjelmistojen laatuvaatimuksiin. Hän kirjoittaa myös Martin Fowlerin mielipiteestä, jonka mukaan ketterät menetelmät sopivat parhaiten businesstyyppisiin sovelluksiin.

Ketterässä ohjelmistokehityksessä asiakkaat ovat suuressa roolissa. Awad (2005) toteaa, että Agile-metodit toimivat parhaiten, kun asiakas on omistautunut ja hänellä on tarpeeksi tietoa järjestelmän kokonaisuudesta. Tämä on ideaalitilanne, mutta todellisuudessa näin ei kuitenkaan aina ole. Awad (2005) kuvailee, kuinka ongelmia voi esiintyä myös sellaisen ohjelmiston tuottamisesta, jonka määrittelyyn osallistuu useita asiakkaita. Vaatimuksissa voi syntyä ristiriitoja, ja tuotteenomaisen ohjelmiston tekeminen olla todella haastavaa.

Suunnittelujohtoiset menetelmät voivat tuoda tämänkaltaisiin ongelmiin ratkaisuja mutta se ei silti tarkoita, etteikö ketteriä menetelmiä voisi käyttää ollenkaan. Monia menetelmiä ja ideologioita pystytään kuitenkin usein soveltamaan niiden ongelmista huolimatta.

Ketterät ohjelmistokehitysmenetelmät ovat siis syntyneet vallitsevan maailman alati muuttuviin tarpeisiin, ja ideologian pohjalta on syntynyt useita eri käytännön menetelmiä, joista yhtenä käytännön esimerkkinä esiteltiin Scrum. Agile-kehityksen perusperiaatteista voidaan nähdä myös Lean filosofian kaltaista ajattelua, johon perehdytään seuraavaksi.

2.4 LEAN osana ohjelmistokehitystä

2.4.1 Yleisesti

Lean ajattelun alkuhetkenä voidaan pitää 1940-luvun loppupuolella Toyotan autotuotannossa käyttöön otettua tuotantoprosessimallia. Toyotan piti saada markkinoille halpoja autoja mutta suuri massatuotanto ei tullut kyseeseen autojen vähäisen kysynnän vuoksi. Niinpä asialle päätettiin tehdä jotain. (Poppendieck & Poppendieck 2003.)

Taiichi Ohno, silloinen Toyota Production Systemsin isähahmona pidetty mies kehitteli uuden tuotantomallin, jonka perusperiaatteena oli hukan tai hukkatyön (Waste) eliminointi. Taiichin ideologian mukaan kaikki, mikä ei tuota asiakkaalle arvoa, on loppujen lopuksi hukkatyötä. Esimerkiksi asioiden tuottaminen, joita ei juuri nyt tarvita, transportaatio, odottelu tai mikä tahansa ylimääräinen vaihe prosessissa, on hukkatyötä. (Poppendieck & Poppendieck 2003.)

Prosessin tarkoitus ei suinkaan ollut kopioida massatuotannon periaatteita, vaan sen fundamentaalisina arvoina olivat mahdollisimman nopea tuotanto ja mahdollisimman nopea tuotteen toimitus asiakkaalle. Ideologian mukaan oli parempi esimerkiksi odottaa, että asiakas teki tilauksen, jonka jälkeen sitä vastaava tuotanto vasta aloitettiin. Samaa ajattelutapaa laajennettiin myös tuotekehitykseen. Ideologian seurauksena oli mahdollisimman nopeat tuotekehitysprojektit, sillä tuotekehitykseen käytetty aika nähtiin siihen asti hukkatyönä, kunnes siitä saatuja tuloksia voitiin hyödyntää. (Poppendieck & Poppendieck 2003.)

Lean ajattelun tärkeimpänä periaatteena toimii siis hukkatyön eliminointi ja kaikki muut ideologian arvot kumpuavat siitä. Jotta hukkatyötä voidaan lähteä eliminoimaan, on sen juurisyyt ensin tunnistettava. Tämä on todella tärkeä osa Lean ajattelun käyttöönottoa, ja varsinkin ohjelmistokehityksessä hukan tai hukkatyön tunnistaminen voi olla todella vaikeaa. (Poppendieck & Poppendieck 2003.)

2.4.2 Hukkatyö (Waste)

Poppendieck ja Poppendieck (2003) viittaavat Winston Roycen (1970) kommenttiin siitä, että loppujen lopuksi ohjelmistokehityksessä mikään muu, kuin analysointi ja koodaaminen, ei tuo asiakkaalle lisäarvoa. Shigeo Shingo, yksi Toyota Production Systemsin asiantuntijoista on tunnistanut seitsemän teollisuuden hukkatekijää, jotka Poppendieck & Poppendieck (2003) ovat kääntäneet ohjelmistokehityksen vastaaviin hukkatyön lähteisiin.

Taulukko 1. Seitsemän hukkatekjiää teollisuudessa ja ohjelmistokehityksessä. (Poppendieck & Poppendieck 2003.)

Hukkatekijät teollisuudessa	**Hukkatekijät ohjelmistokehityksessä**
Varastot	Osittain tehty työ

Ylimääräinen prosessointi	Ylimääräiset prosessit
Ylituotanto	Ylimääräiset toiminnalisuudet
Kuljetus ja transportaatio	Työtehtävien vaihtelu
Odottelu	Odottelu
Liike	Liike
Viat	Viat

Osittain tehty työ

On todettu, että ohjelmistoala muuttuu nopeasti, ja näin ollen keskeneräinen koodi tai sovellus saattaa vanhentua ja tulla lyhyessä ajassa tarpeettomaksi. Keskeneräiset työt saattavat haitata muuta kehitystä, jos sovelluksen jokin muu osa on riippuvainen kyseisestä keskeneräisestä toiminnallisuudesta. Keskeneräisyys luo myös riskejä ja piileviä ongelmia, sillä jos koodia ei pystytä kattavasti testaamaan, eikä sitä voida toimittaa asiakkaalle, se sitoo ainoastaan resursseja tuottamatta tuloksia. Aina ei voida varmaksi tietää, tuleeko kyseinen kehitys täyttämään liiketoiminnallisen tarkoituksensa tai kuinka paljon sen muuttaminen veisi lisää resursseja. Monissa tapauksissa voidaan siis olettaa, että keskeneräisyyden minimoiminen vähentää sekä riskejä että hukkaa. (Poppendieck & Poppendieck 2003.)

Ylimääräiset prosessit

Joitakin ohjelmistokehityksen aikana toteutettuja prosesseja tai niiden osia voidaan todeta turhiksi. Yhtenä esimerkkinä voidaan käyttää ohjelmistoista tehtävää dokumentointia tai asiakasta varten tuotettavaa paperityötä. Sopimusten, määrittelyjen ja dokumenttien tuottaminen on monesti hyödyllistä mutta usein hyödyllistenkin dokumenttien tuottamista voidaan parantaa. Vaarana on erityisesti se, että dokumentaatiota pidetään itseisarvoisena. Dokumenttien, määrittelyjen tai sopimusten yksityiskohtien viilaamista tulee miettiä siltä kannalta, tuottaako se oikeasti asiakkaalle lisäarvoa. Suurimman hyödyn saamiseksi dokumentaatiota tulee myös ylläpitää säännöllisesti, sillä monesti huomataan, kuinka pitkät määrittelyt tai yksityiskohtaiset dokumentaatiot ovat päässeet vanhentumaan ja näin tulleet käyttökelvottomiksi. (Poppendieck & Poppendieck 2003.)

Ylimääräiset toiminnallisuudet

Ylimääräisten toiminnallisuuksien lisääminen ohjelmistoon saattaa vaikuttaa monesti hyvältä idealta, mutta Lean ajattelu näkee ylimääräisen työn kuitenkin hukkana. Ylimääräinen työ ei tuota asiakkaalle heti arvoa, se sitoo resursseja ja saattaa myöhemmin aiheuttaa turhia ongelmia. Toiminnallisuuksien vaatimukset saattavat myös muuttua ennen niiden käyttöönottoa, joten ylimääräinen toiminnallisuus saattaa vanhentua ja tulla turhaksi ennen kuin sen elinkaari on edes alkanut. Ylimääräiset toiminnallisuudet lisäävät myös ohjelmiston kompleksisuutta ja saattavat aiheuttaa järjestelmässä virheitä. (Poppendieck & Poppendieck 2003.)

Työtehtävien vaihtelu

Poppendieck ja Poppendieck (2003) viittaavat DeMarcon ja Listerin tutkimukseen vuodelta 1999, jossa todetaan työtehtävien vaihtelujen aiheuttavan hukkaa. Joka kerta, kun ohjelmistokehittäjä joutuu vaihtamaan työtehtävää, syntyy siirtymä, jonka aikana työtehokkuus heikentyy selkeästi. Tutkimuksen valossa usean projektin aloittaminen samoilla resursseilla ei siis ole kannattavaa. (Poppendieck & Poppendieck 2003.)

Odottelu

Lean ajattelun näkökulmasta yksi suurimmista hukan lähteistä on odottelu. Ohjelmistokehityksen eri vaiheissa voi ilmentyä erilaisia ja eritasoisia viivytyksiä ja osa näistä saattaa olla turhia. Projektin aloitukset voivat venyä, tarvittavaa henkilötyövoimaa tai resursseja voi joutua odottamaan, vaadittava dokumentaatio, katselmoinnit, erilaiset hyväksynnät, testaukset, asennukset ja niin edelleen. Monien edellä mainittujen asioiden näkeminen hukkana ei välttämättä ole intuitiivista mutta silti monet näistä estävät asiakasta saamasta lisäarvoa nopeammin. (Poppendieck & Poppendieck 2003.)

Liike

Myös liikettä voidaan pitää ohjelmistokehityksessä hukkana. Yhtenä esimerkkinä voidaan pitää tiedonkulkua ja dokumenttien liikkumista. Tieto saattaa liikkua organisaatiossa liian hitaasti tai esimerkiksi dokumenttien edestakainen liikuttelu asianomaiselta toiselle voi olla turhaa. Myös koodin liikkumista voidaan pitää hukkana. Samaa

kehitystä voidaan joutua esimerkiksi viemään useamman henkilön tai ympäristön lävitse. Lean ideologia tulkitsee kaiken liikkeen siis potentiaalisena hukkana. (Poppendieck & Poppendieck 2003.)

Viat

On sanomattakin selvää, että viat ohjelmistokehityksessä aiheuttavat hukkaa. Bugien ja epäkohtien korjaaminen on yleensä sitä kalliimpaa, mitä pidemmälle ohjelmiston elinkaaressa mennään. Niinpä monet ohjelmistokehitysmenetelmät pyrkivät löytämään ja korjaamaan viat mahdollisimman nopeasti. (Poppendieck & Poppendieck 2003.)

Yhtenä Lean ajattelun perusperiaatteena on siis hukkatyön tunnistaminen ja erilaisten metodien avulla sen vähentäminen. Hukkatyön tunnistamisen ohella Leanin perusperiaatteisiin kuuluu asiakkaalle tuotettavan arvon tunnistaminen.

2.4.3 Asiakkaalle tuotettava arvo (Value)

Arvon määrittely ohjelmistokehityksessä voi olla todella haastavaa. Siitä, mikä tuottaa asiakkaalle näkyvää arvoa, käydään alati keskustelua ja näkemykset esimerkiksi perinteisten ja ketterien/Lean -menetelmien välillä eroavat selkeästi. Agile ja Lean ideologioiden mukaan ohjelmistokehitystiimin pitäisi oppia eri liiketoiminnallisten osakkaiden näkemys arvosta, jotta sen määrittely olisi mahdollisimman totuudenmukaista. (Rodrizues 2013).

Boehm, Biffl, Aurum, Erdogmus ja Grünbacher (2006) toivat esille arvo-pohjaisen ohjelmistokehityksen mallin, joka perustuu yksinkertaisesti siihen, että asioita käsitellään arvo edellä. Heidän mukaan ohjelmistokehitysprojektin tyypillinen ongelma on se, että kaikkia liiketoiminnallisia vaatimusmäärityksiä pidetään saman arvoisina. Boehm ym. toivat esille arvo-pohjaisen vaatimusmäärittelyn, -suunnittelun ja ylipäätään arvoon pohjautuvan insinöörityön merkityksen. Boehmin ym. mukaan ohjelmistokehityksestä saadaan arvo-edellä toimivaa, kun analysoidaan, mitä hyötyä uusi ohjelmisto tuo, ja tutkitaan, mitä asioita sen eri osakkaat arvostavat.

Vallitsevan Agile ja Lean -ohjelmistokehityksen myötä arvoedellä toimiminen on kasvanut merkittävästi (Dingsøyr & Lassenius 2016). Lean ideologia tarjoaa yksinkertaisen työkalun asiakkaalle tuotettavan arvoketjun tunnistamiseen.

Value stream mapping

Value Stream Mapping on työkalu, tai enemmänkin prosessin mallinnuskaava, jonka avulla voidaan tutkia arvoa ja hukkaa tuottavia tekijöitä. Value Stream Mapping'n ideana on tutkia tuotteen tai projektin elinkaarta ja kaikkia eri vaiheita, joiden läpi tuotettava arvo kulkee aina alkuperäisestä ideasta asiakkaan tuotantoympäristöön asti. Ohjelmistotuotannossa Value Stream Mapping'n ideana on tunnistaa kaikki kehitysprosessin vaiheet, tutkia kuinka kauan kukin vaihe kestää ja erotella esimerkiksi odotteluvaiheet ja aktiiviset työvaiheet. Näin saadaan selville sekä hukkaa että arvoa tuottavat prosessin eri vaiheet, minkä jälkeen niitä voidaan lähteä joko parantamaan tai karsimaan. (Poppendieck & Poppendieck 2003.)

Pull-System

Lean ajattelussa pull-systemillä tarkoitetaan asiakasvetoista ohjelmistokehitystä. Monesti organisaatiossa työnteko tapahtuu push -tyyppisesti, eli aikataulut työntävät kehitystä eteenpäin sen sijaan, että asiakkaan vaatimukset vetäisivät ohjelmiston kehitystä eteenpäin. Lean periaate uskoo siihen, että kun ohjelmistokehittäjillä on tarpeellinen kompetenssi työtehtävän suorittamiseen, kun heille annetaan tarvittavat työkalut ja luodaan sopivat työolosuhteet, syntyy pull-vaikutus eli asiakasvetoinen työskentelytapa. (Poppendieck & Poppendieck 2003.)

Minimum viable product (MVP)

Minimum Viable Product eli MVP-malli on ideologinen lähestymistapa sille, että asiakkaalle tuotetaan tuote mahdollisimman nopeasti ja mahdollisimman pienillä resursseilla. MVP-toteutuksen ideana on tuottaa asiakkaalle mahdollisimman paljon arvoa, mahdollisimman pienellä panostuksella. Ideana on kokeilla uusia ideoita ja lähestymistapoja ja jos huomataan, että toteutus ei toimi se voidaan heti kuopata ja aloittaa jonkun muun toteutuksen työstäminen. (Poppendieck & Poppendieck 2003.)

2.4.4 Lean ja Agile

Lean ja Agile metodeilla on todettu olevan paljon yhteistä ja Agile metodit ovatkin ottaneet paljon vaikutteita Lean ideologian perusperiaatteista. Molemmille on yhteistä esimerkiksi ihmisjohtoinen lähestymistapa ja laadun sekä teknisen osaamisen jatkuva ylläpito. (Rodríguez 2013.)

Vaikka Lean ja Agile käytännöissä ei olekaan mitään yhtä ja ainoaa oikeaa ratkaisua kaikkeen, on tutkimuksissa todettu muutamia tapoja hyödyllisiksi. Säännölliset koodikäännökset, keskeneräisen työn vähentäminen, moniosaavat tiimit ja organisaation lisääntynyt läpinäkyvyys ovat yleisesti hyvänä pidettyjä asioita. Kriittisiä näkökulmia on myös esitelty ja esimerkiksi ylistandardisoiminen voi helposti tulla Lean ideologian ja ketteryyden tielle. Ongelmia voi tulla myös arvon tuottamisen kysymyksistä, kuten siitä, korjataanko ongelmia vai tehdäänkö asiakkaalle uusia toiminnallisuuksia. (Rodríguez 2013.)

Ohjelmistokehityksessä on aikojen saatossa havaittu toimivia menetelmiä, joita on lähdetty jalostamaan ja joiden ympärille on kehittynyt erilaisia ideologioita siitä, miten asiat hoidettaisiin parhaiten. Ideologiat, aatteet ja toimintatavat pyrkivät vastaamaan IT-maailman muuttuviin tarpeisiin ja koettavat rakentaa ja ylläpitää sitä, miten ohjelmistokehityksen päivittäiset työtehtävät hoidettaisiin kaikista optimaalisimmin asiakasta unohtamatta. Viimeisimpänä jatkumona ohjelmistokehitysmenetelmien ja ideologioiden saralla pidetään DevOps ajattelutapaa, joka lähestyy asioita kokonaisvaltaisesti, käsitellen myös paljon ohjelmistokehitykseen liittyviä käytännön asioita.

3 DevOps - hyväksi havaitut menetelmät käytäntöön

3.1 Mitä on DevOps?

DevOps ja sen tuomat tekniset, arkkitehtuurilliset ja kulttuurilliset käytänteet polveutuvat monien eri johtamismenetelmien ja ideologioiden summasta. DevOps on ottanut vaikutteita niin teollisuuden korkeaa laatua tuottavista organisaatioista, kuin luottamukseen ja muutokseen perustuvista johtamismenetelmistä. DevOps on monien mielestä loogista jatkumoa Lean ajattelusta ja ketterästä ohjelmistokehityksestä. (Kim, Humble, Debois & Willis 2016.)

DevOpsin perusperiaatteisiin kuuluvat oppimisen kulttuuri, päivittäisen työn kehittäminen, idea turvallisesta- ja syyttely-vapaasta työympäristöstä ja ihmislähtöinen tekeminen. Teknisemmällä puolella DevOps käsittelee esimerkiksi riskienhallintaa, monitorointia, mikroarkkitehtuuria, testiautomaatiota, integraatioautomaatiota ja infrastruktuuria koodina. (Kim ym. 2016.) Mutta mistä termi DevOps on syntynyt?

Vuosien saatossa monet organisaation IT-palveluiden vastuualueet ovat enemmän ja enemmän yhdistyneet ohjelmistokehityksen puolelle (Loukides 2012). Esimerkiksi palvelimien ylläpito ja konfigurointi tai tuotettavan ohjelmiston toimittaminen asiakkaalle, ovat historiallisesti olleet IT-puolen tai 'Operaattorin' vastuulla. Loukides (2012) kirjoittaa siitä, kuinka modernien, usein pilvipalvelimilla sijaitsevien ohjelmistojen tulee edelleenkin palautua nopeasti virheistä, olla monitoroitavissa ja skaalautua muuttuvan kuorman mukaan. Tarpeet ja ideat ovat samat mutta vastuu on siirtymässä.

DevOps ideologian pohjalla on siis pitkälti perustavaa laatua oleva infrastrukturaalinen muutos kohti ohjelmallisuutta ja Infrastructure as code -ajattelutupaa. Loukides (2012) kirjoittaa, kuinka aikaisemmin IT palveluiden puolella työskennelleiden tulee muovata osaamistaan kohti uutta, ohjelmistokehittäjien kanssa yhteistyössä käytettävää toimintamallia. Tästä juontaa juurensa myös käsite DevOps, eli yhdistelmä sanoista Development & Operations, joka kuvaa ideologian ydintä eli IT-puolen (Operations) ja ohjelmistokehityksen (Development) vastuualueiden yhdistymistä. (Loukides 2012.)

DevOpsin tarkoituksena on varmistaa organisaatiossa laatu, luotettavuus, tietoturvallisuus, luoda vakautta ja tehdä tämä kaikki entistä halvemmalla. Ajattelumalli pyrkii vaikuttamaan asioihin teknologisen arvoketjun tunnistamisen ja organisaation kokonaisvaltaisen vaikuttamisen myötä eli IT-operaatioiden ja kehittäjien yhteen vieminen ei suinkaan ole DevOps kokonaisuudessaan. (Kim ym. 2016.) DevOpsin perusperiaatteisiin katsotaan kuuluvan myös ohjelmistokehitystä edesauttavan työkulttuurin luominen, automaatio, oppimisen arvostus ja kaiken edellä mainitun mittaaminen.

3.2 Kulttuuri ja oppiminen

Yksi DevOpsin tärkeimmistä, ellei jopa tärkein piirre on työkulttuuriin parantaminen. Luottamuksen kulttuuri on organisaatiossa tärkeää, sillä olemme kaikki elinikäisiä oppijoita ja tietoisten ja hallittujen riskien otto kuuluu tai ainakin pitäisi kuulua monien päivittäiseen työhön. DevOps ideologian mukaan luottamuksen kulttuuri on paljon tuottavampi, kuin työkulttuuri, jossa virheistä rankaistaan. Työntekijät joutuvat pel-

käämään tahattomien virheiden seurauksia, mikä johtaa epäkohtien peittelyyn ja innovoinnin katoamiseen. DevOps näkee virheet hyvänä, sillä se uskoo, että niiden läpikäynnin ja analysoinnin myötä niistä opitaan, ja tätä kautta myös niiden ennaltaehkäisy helpottuu. (Kim ym. 2016.)

DevOps kannustaa siis virheiden ja epäkohtien julki tuomiseen, niiden analysointiin ja käsittelyyn. Näitä käsittelytapahtumia kutsutaan post-mortemeiksi. Virhetilanteiden läpikäynnin jälkeen on tärkeää, että tietoa levitetään ja jaetaan eteenpäin, jotta organisaation muiden osapuolien on helppo hyötyä jo kertaalleen käsitellyistä asioista. DevOps haluaa siis rakentaa oppimisen kulttuuria organisaatioon. (Kim ym. 2016.)

DevOps ajattelutapa kannustaa myös työntekijöiden oikeanlaisen motivaation löytämiseen, sillä uskotaan, että ihminen työskentelee parhaiten ollessaan sisäisesti motivoitunut. Tähän vaikuttavat muun muassa tuntemus autonomiasta eli oman työnsä päätäntävallasta, merkitys eli joku suurempi tarkoitus tehdyllä työllä ja turvallinen työympäristö, jossa virheistä ei rankaista. (Kim ym. 2016.)

3.3 Päivittäisen työn kehittäminen

Ohjelmistokehityksessä ajaudutaan usein tilanteeseen, jossa ongelmat ratkaistaan nopeilla korjauksilla. Korjauksilla, jotka saattavat olla arkkitehtuurillisesti huonoja ja ohjelmistoon sopimattomia. Tämä johtaa vääjäämättä tekniseen velkaan, joka täytyy ennen pitkää maksaa takaisin. DevOps näkee päivittäisen työn kehittämisen todella tärkeänä, sillä tavoitteena on, että jokainen työntekijä maksaisi teknistä velkaa takaisin korjaamalla epäkohtia. Ohjelmistokehittäjälle luonnollisin tapa siihen on parantaa huonoa koodia. (Kim ym. 2016.)

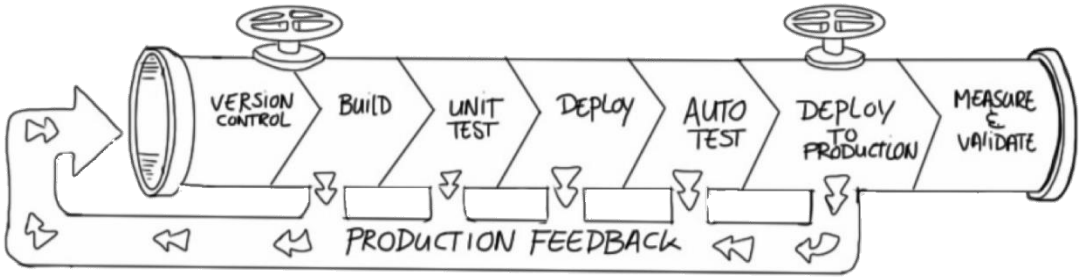
Jokapäiväisen työn kehittäminen ei kuitenkaan tulisi tapahtua yhdellä kertaa, tai ole erillinen projekti. DevOpsin näkökulmasta päivittäisen työn kehittäminen tulisi tapahtua sykleittäin ja olla jatkuvaa (Kim ym. 2016). Tärkein syy kaikkeen tähän on se, että ohjelmistoalalla epäkohtien korjaaminen on järjestäen sitä halvempaa, mitä aikaisemmin se tehdään ja ongelmien seuraukset ovat luultavasti paljon pienemmät (Kim ym. 2016, Humble 2016.)

Yksi merkittävä askel kehityksen suuntaan on ohjelmistokehittäjän yksittäisen työtehtävän prosessielinkaaren lyhentäminen. Idea on siinä, että mitä lyhyempi prosessisykli on, sitä nopeammin uusista kehityksistä saadaan palautetta. Palaute voi olla niin integraatioon, toiminnallisuuteen, kuin asiakasvaatimuksiin liittyvää mutta tärkeintä on se, että palaute saadaan mahdollisimman nopeasti. (Humble 2016.)

Keinoja kehitysten läpiviennin lyhentämiseen ovat esimerkiksi työtehtävien paloittelu pienempiin osiin, automaattinen testaus ja kehitysten automaattinen integraatio tuotteeseen. Mitä nopeammin esimerkiksi koodimuutos integroidaan, sitä nopeammin saadaan tietoa siitä, onko kyseinen kehitys tehty oikealla tavalla ja voiko se toimia osana vallitsevaa järjestelmää. Tätä jatkuvaa integraatiota kutsutaan DevOpsissa Continuous integrationiksi. (Humble 2016.)

3.4 Automatisaatio

Uusien kehitysten jatkuvan integroinnin ja jatkuvan toimittamisen katsotaan olevan osa DevOpsin teknisiä prosessimalleja. Ideana on tehdä ohjelmistotuotannon rutiininomaisista prosesseista standardoituja ja automatisoida ne. (Humble 2016.) Jatkuvan integroinnin ja jatkuvan toimittamisen ideana on lyhentää ohjelmistokehityksen iteraatiosyklejä ja sitä kautta lisätä tuottavuutta, tehokkuutta ja laatua (Chen 2015). Ohjelmistokehityksen jatkuvaa integraatiota kuvastaa hyvin kuviossa 3. esitelty Continuous Delivery pipeline eli jatkuvan toimittamisen ‘putki’.



Kuvio 3. Jatkuvan toimittamisen putki. (Continuous Deliver with Docker 2016.)

Continuous Delivery pipeline’n ja jatkuvan integraation ideana on siis automatisoida rutiininomaiset prosessit. Kaikki alkaa yleensä siitä, kun ohjelmistokehittäjä lähettää valmistellut muutokset keskitettyyn versionhallintaan (ks. termit). Prosessin vaiheet voidaan tämän jälkeen karkeasti jakaa käännösvaiheeseen (Build), testausvaiheeseen

(Unit test, Auto test) ja integraatio-/toimitus vaiheeseen (Deploy, Deploy to production). Chen (2015) kuvaa jatkuvaa toimittamista toimintamallina, jonka tarkoituksena on tuottaa laadukas tuote lyhyillä iteraatiosykleillä ja varmistaa, että ohjelmiston voi toimittaa asiakkaalle koska tahansa.

Humblen ja Farleyn (2010) näkemys jatkuvasta toimittamisesta on hyvin samankaltainen. Heidän mielestä jatkuvan toimittamisen ideana on luoda prosessiin läpinäkyvyyttä kaikille osapuolille ja tuoda epäkohdat sekä onnistumiset selkeästi näkyville.

Jatkuvasta integraatiosta on hyötyjä ainakin laadun parantamisen ja prosessisyklien lyhenemisen myötä. Chen (2015) kirjoittaa kokemuksensa pohjalta myös muista hyödyistä. Hänen mukaansa toimitusprosessin automatisointi tuottaa liiketoiminnallisia hyötyjä, sillä se nopeuttaa muutosten tuontia markkinoille. Palautesyklit ovat myös nopeampia ja yrityksen on helpompi tehdä nopeita korjausliikkeitä, mikäli se toteaa, että kehitetyt asiat ovat vääränlaisia.

Chen (2015) toteaa myös, että jatkuvan toimittamisen myötä yleinen tuottavuus, tehokkuus ja laatu kasvavat. Hän kertoo myös julkaisujen luotettavuuden paranevan ja näiden kaikkien myötä luonnollisesti asiakastyytyväisyyden lisääntyvän. Humble ja Forsgren (2016) mukaan CI automaatio tuo myös työhyvinvoinnillisia etuja, sillä se helpottaa toimitusta tekevien ihmisten työtä vähentäen heiltä kuormittavaa työtaakkaa.

DevOps yhdistää ketterien kehitysmenetelmien ja Lean ajattelun perusperiaatteita ja tarjoaa kokonaisvaltaisen lähestymistavan tuottavaan ja laadukkaaseen ohjelmistokehitykseen. Hyväksi havaittujen menetelmien ymmärtäminen on tärkeää, kun ohjelmistokehityksen prosesseja lähdetään kehittämään. Näin vältetään yleisimmät virheet ja päästää nopeammin oikeiden, potentiaalisten arvoa tuottavien kehityskohteiden äärelle.

4 Difs - Toimituskäytänteiden nykytilanne

4.1 Työkalut ja palvelut

4.1.1 Yleisesti

Ennen varinaiseen kehitystyöhön lähtemistä, katsottiin hyväksi käydä läpi organisaationtoimituskäytänteiden nykytilanne ja siihen läheisesti liittyvät työkalut. Digia Financial Solution organisaatiossa käytetään useita erilaisia ohjelmistokehitystä helpottavia työkaluja, joita on niin versionhallintaan, työtehtävien hallintaan, koodikatselmointiin, kuin jatkuvaan integraatioonkin.

4.1.2 Versionhallinta (TortoiseSVN)

Versionhallinta ohjelmistokehityksessä tarkoittaa lähdekoodin muutostietojen ylläpitoa ja hallintaa siten, että virheen sattuessa vanhaan versioon palaaminen onnistuu helposti. Versionhallinta helpottaa myös ohjelmistokehittäjien päivittäistä työntekoa esimerkiksi tilanteissa, joissa samaa lähdekoodipohjaa työstetään usean kehittäjän toimesta samanaikaisesti. (Atlassian 2017.)

Kaupallisessa ohjelmistokehityksessä on lähes poikkeuksetta käytössä aina jokin versionhallintajärjestelmä ja sen käyttöön jokin työkalu. Tortoise SVN on versionhallinnan graafinen käyttöliittymätyökalu Windowsille. Käyttäjän ohje kuvaa ohjelmistoa seuraavasti:

> *TortoiseSVN on ilmainen, avoimeen lähdekoodiin perustuva Windows-käyttöliittymä Apache™ Subversion®- versionhallintaan. TortoiseSVN pitää kirjaa tiedostoihin ja hakemistoihin ajan mittaan tehdyistä muutoksista. Tiedostot talletetaan keskitettyyn arkistoon. Arkisto muistuttaa tavallista tiedostopalvelinta, mutta kykenee lisäksi muistamaan kaikki tiedostoihin ja hakemistoihin ajan mittaan tehdyt muutokset. Tämän ansiosta voit palauttaa vanhoja versioita tiedostoistasi ja tutkia, miten, milloin ja kenen toimesta ne ovat muuttuneet. Subversionia (ja versionhallintajärjestelmiä yleensä) voikin pitää eräänlaisena tiedostojärjestelmien ”aikakoneena”.*

Versionhallinta on todella tärkeä osa ohjelmistokehitysprosessia ja kaikki tässä osiossa esiteltävät työkalut integroituvat tai käyttävät sitä tavalla tai toisella.

4.1.3 Työtehtävien hallinta (Jira)

Työtehtävien hallinta on suuressa organisaatiossa todella tärkeä osa päivittäistä työtä. Projekteja ja asiakkaita on yleensä useita ja kaiken työn hallitseminen ja seuraaminen ilman toimivaa työkalua olisi varmasti haastavampaa ja enemmän aikaa vievää.

Jira on Atlassianin kehittämä tehtävienhallinnan työkalu, joka tarjoaa työkalut päivittäisten työtehtävien prosessihallintaan, tarjoten alustan myös projektien ja kokonaisuuksien hallintaan. Financial Solutions organisaatiossa, jossa kehitystä tehdään useisiin asiakasprojekteihin, luotettava projektien ja työtehtävien hallinta on tärkeää. Jira on yksi tähän tarkoitukseen eniten käytetty työkalu ohjelmistokehitysalalla.

Jirassa yksittäisiä työtetehtäviä käsitellään tiketteinä. Kuviossa 4. nähdään yksittäinen tiketti, jolle kirjataan työtehtävän yksityiskohtainen kuvaus, ja liitetään siihen mahdollisesti liittyvät toiminnalliset- ja tekniset vaatimusmäärittelyt.

Kuvio 4. Jira tiketti eli yksittäinen työtehtävä.

Tämä perusperiaate ohjaa suurempien työtehtävien paloitteluun ja yksittäiset työtehtävät eri vaiheiden kautta koko prosessin läpi. Edistyvää työtä voidaan seurata vaihe vaiheelta aina kehityksestä toimittamiseen. Näin työkalu ohjaa siihen, että esimerkiksi kehityksen katselmointi tai testaus ei pääse unohtumaan.

Jira integroituu myös koodikatselmointityökalu Crucibleen ja muutoshistoriatyökalu EasyChangelogiin ja on tästäkin syystä tärkeä osa asiakastoimituksien kokonaisuuk-

sienhallintaa. Jiraa voidaankin versionhallinnan jälkeen pitää Difs-organisaation ohjelmistokehityksen tärkeimpänä työkaluna, jonka ympärille ohjelmistokehitysprosessi pitkälti rakentuu.

4.1.4 Koodikatselmointi (Crucible)

Koodin katselmointi on yksi ohjelmistokehityksen sisäisen laadunvarmistuksen prosesseista, jolla pyritään saamaan kehitetystä kokonaisuudesta sellaista tietoa, mitä normaali ohjelmistotestaus ei yleensä yksinomaan tarjoa. Ohjelmiston testauksessa todennetaan yleensä se, toimiiko kehitetty ominaisuus, täyttääkö se annetut vaatimukset tai rikkooko se jotain muuta toiminnallisuutta. Testauksella ei kuitenkaan saada tietoa siitä, **miten** kehitys on rakennettu ja **miten** se toimii. (Macchi n.d.)

Koodikatselmoinnilla ylläpidetään tuotteen sisäistä laatua ja herätetään vertaisarvioinnin kautta keskustelua siitä, miten kehitykset tehdään esimerkiksi arkkitehtuurillisesti mahdollisimman ylläpidettäviksi. Monet kehitykset voivat toimia loppukäyttäjän testauksessa täysin vaatimusten mukaisesti mutta esimerkiksi suorituskykyyn liittyvät seikat eivät välttämättä ilmaannu, kuin vasta myöhemmin. Järjestelmäasiantuntijat pystyvät koodia lukemalla tulkitsemaan usein myös regressiollisia ongelmia, eli muutosten negatiivisia ja rikkovia vaikutuksia muuhun järjestelmään. (Macchi n.d.)

Crucible (2017) on Atlassianin kehittämä työkalu koodimuutosten katselmointiin. Työkalun kautta pystyy helposti lisäämään katselmoivia osapuolia, kirjoittamaan koodiriveihin kohdistettuja kommentteja muutoksen tehneelle kehittäjälle ja näkemään katselmointityön edistymisen. Kuviosta 5. nähdään yksittäisen koodikatselmoinnin yhteenveto. Ideana on yksinkertaisesti kehittäjien välinen vertaisarviointi ja tätä kautta laadun varmistaminen.

Participant	Role	Time Spent	Comment
Author	Author	20m	
Reviewer	Reviewer - Complete	14m	1
Reviewer	Reviewer - 0% reviewed		
Puskala Juha	Reviewer - 100% reviewed	10m	1
Total		**44m**	**2**

Linked Issue: *None* – Would you like to create a link to BRO-2450 ? Create Link

Patches: **BRO-2450.patch:** (anchored to DiFS_BSA : /)

Objectives *Edit*

Kuvio 5. Crucible koodikatselmoinnin yhteenveto.

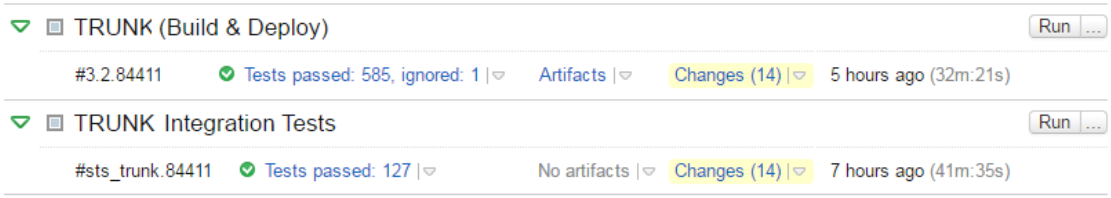
Crucible on integroitu organisaation SVN versionhallintaan niin, että kaikki kehittäjien tekemät muutokset näkyvät lajiteltuina Cruciblen käyttöliittymältä. Integraation avulla työkalu osaa näyttää koodiin tehdyt muutokset suhteessa aikaisempiin iteraatiohin ja vanhoihin versioihin.

4.1.5 Jatkuva integraatio (TeamCity)

DevOps luvussa kuvattiin Continuous Integration eli jatkuvan integraation perusperiaatteet. Financial Solutions organisaatiossa käytetään JetBrainsin kehittämää ohjelmistoa nimeltä TeamCity, joka tarjoaa laajan skaalan toiminnallisuuksia jatkuvaan ohjelmistointegraatioon (TeamCity 2017).

TeamCityn perusideana toimii käännöskonfiguraatio, joka lähestymistavasta riippuen voi toteuttaa yhden tai useamman CI putkeen (kuvio 3.) kuuluvan prosessivaiheen. Selkeyden tai optimoinnin vuoksi voidaan esimerkiksi määritellä, että yksi käännöskonfiguraatio hoitaa ohjelmiston kääntämisen ja toinen ajaa samalle ohjelmistolle automaattitestit.

Kuviossa 6. nähdään kaksi TeamCityn käännöskonfiguraatiota tyypillisessä projektinäkymässä. Yleisnäkymästä nähdään nopeasti hyödyllisiä perustietoja, kuten esimerkiksi käännöksen nimi, versio, käännöksen kesto, mukaan käännetyt muutokset ja kaikkein tärkeimpänä onnistuiko käännös vai ei.

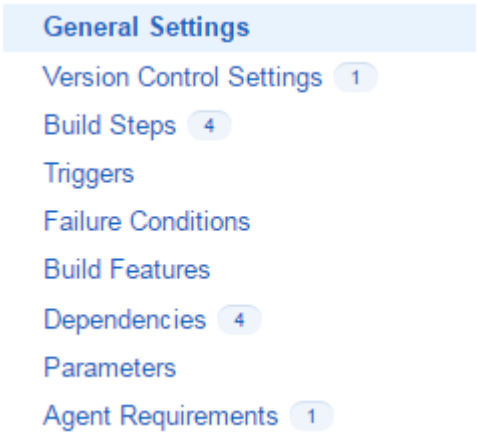


Kuvio 6. TeamCity käännöskonfiguraatiot.

Kuviosta 7. nähdään TeamCityn käännöskonfiguraation mahdolliset asetukset. Mahdollisia konfiguroitavia asioita ovat käännöksen perustiedot, versionhallinnan integraatio, käännösvaiheet, automaattiset käynnistysmäärittelyt, käännöksen epäonnistumisen ehdot, lisäominaisuudet, snapshot- tai artefaktiriippuvuudet, käännösparametrit ja käännöksen suorittavan agenttipalvelimen määrittelyt.

Useimmiten uusi käännöskonfiguraatio luodaan ottamalla kopio samankaltaisesta konfiguraatiosta, joten perustietoihin muokataan yleensä vain käännöskonfiguraatiolle kuvaava nimi. Tärkeä perustiedoissa määritelty tieto on myös käännöskonfiguraation tunnisteavain. Tunnisteavaimen perusteella käännöskonfiguraation tietoja on mahdollista käyttää muissa TeamCityn käännöskonfiguraatiossa.

Tämän jälkeen käännöskonfiguraatio yhdistetään versionhallinnassa sijaitsevaa ohjelmiston versiohaaraan. Versiohaaran määrittelyn avulla TeamCity lataa käännöspalvelimelleen lähdekoodit, joita käytetään myöhemmin kääntämiseen ja muihin konfiguraatiossa määriteltyihin käännösvaiheisiin. Käännösprosessi on täysin sama, kuin se, jonka kehittäjä tekee omalla työasemallaan mutta TeamCityn avulla se on keskitetty ja automatisoitu.



Kuvio 7. TeamCity käännöskonfiguraation asetukset.

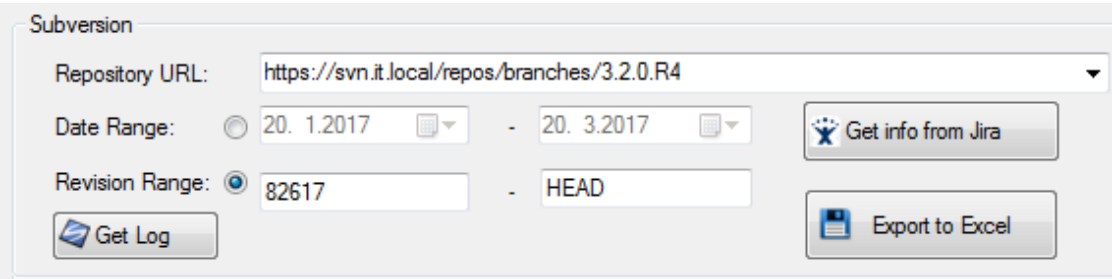
Seuraavaksi määritellään käännösvaiheet. Yhteen käännöskonfiguraatioon voidaan määritellä niin monta käännösvaiheetta, kuin katsotaan tarpeelliseksi. Vaiheista tyypillisimpiä ovat ohjelmiston binääritiedostojen kääntäminen, testien ajaminen ja käännetyn ohjelmiston asentaminen testipalvelimelle. Käännösvaiheita voi suoraa verrata DevOps luvun kuviossa 3. esiteltyyn jatkuvan toimittamisen putkeen.

Kun TeamCity kääntää lähdekooditiedostoja ihmisen ymmärtämästä tiedostomuodosta tietokoneen ymmärtämään binäärimuotoon, kokoaa se käännöksen tuloksista artefaktipaketteja. Artefaktipakettien ideana on säilyttää kertaalleen tehty käännös niin, että se voidaan versioida ja kuljettaa kaikkien muiden vaiheiden läpi muuttumattomana. Yksittäisen käännöskonfiguraation luomaa artefaktipakettia voi käyttää myös muissa käännöskonfiguraatioissa, mikä mahdollistaa esimerkiksi vaiheiden paloittelun useampaan konfiguraatioon.

Riippuvuuksien avulla TeamCity käännöskonfiguraatioon voidaan asettaa snapshot- tai artefaktiriippuvuuksia. Edellä mainittuja käännöskonfiguraation koostamia artefakteja voidaan liittää muihin käännöskonfiguraatioihin niin, että käännettyä ohjelmistopakettia voidaan uudelleen käyttää. TeamCityn toimintaa kuvataan lisää luvussa 5.2.

4.1.6 Muutoshistoria (EasyChangelog)

Muutoshistorian ideana on ohjelmistopäivityksen yhteydessä toimittaa asiakkaalle tietoa siitä, mikä järjestelmässä on muuttunut. Muutoshistorialla saatetaan asiakas tietoiseksi uusista ominaisuuksista ja konfiguraatiotarpeista, jotta asiakas tietää miten reagoida.



Kuvio 8. EasyChangelog.

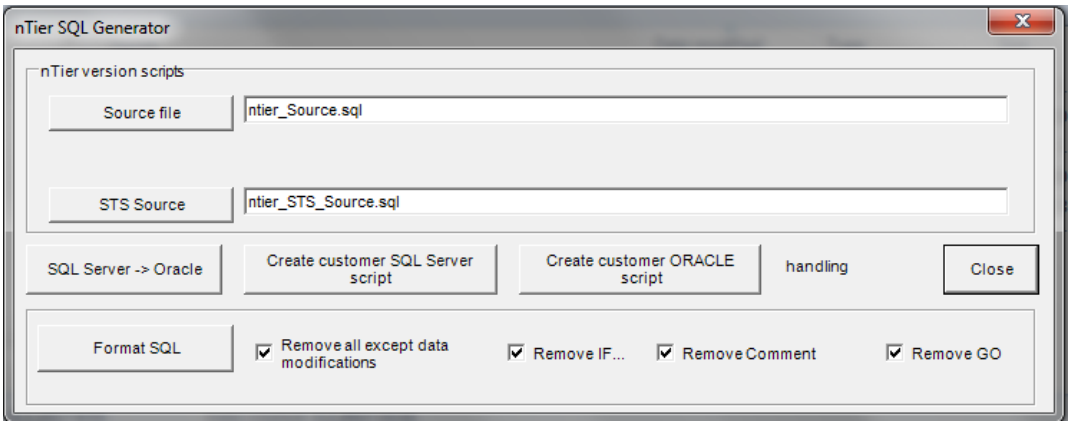
Easy Changelog on Difs-yksikön kehittäjien itse tekemä työkalu, jonka avulla ohjelmistojen muutoshistoriatietoja toimitetaan asiakkaalle. Kuviosta 8. nähdään, kuinka työkaluun syötetään versiohallinnan haara ja haluttu versio-, tai päivämääräväli. Tämän jälkeen työkalu hakee kehittäjien tekemät muutokset ja koostaa niistä yksikäsitteisen listan.

Listaan saa integroitua Jirasta dataa siten, että työkalu tunnistaa Jira tiketillä olevan tunnisteavaimen ja yhdistää sen versionhallintaan tehtyyn kommittiin eli koodimuutokseen. Näin saadaan tuotettua asiakkaalle tarkalla tasolla kaikki muutokset, joita toimitettavaan tuotteeseen on tehty. Lopullinen muutoshistoria viedään Exceliin, joka tarkastuksen ja korjausten jälkeen toimitetaan asiakkaalle.

4.1.7 Tietokantaskriptit (NTier SQL Generator)

Tietokanta ja sen ylläpitäminen ovat yksi ohjelmistotuotannon perusasioista. Ohjelmat kehittyvät, uusia ominaisuuksia luodaan ja dataa täytyy tallentaa koko ajan enemmän ja enemmän. Tietokantoja on olemassa useita erialaisia, niin relaatiollisia, kuin dokumenttipohjaisiakin. Difs-tuotteita käytetään kahden eri tietokantajärjestelmän kanssa. Nämä molemmat, Oracle ja SQL Server, ovat relaatiotietokantoja (ks. termit) ja niiden ylläpitämiseen käytetään NTier SQL Generator nimistä työkalua.

NTier SQL Generator on Difs-kehittäjien toteuttama tietokantaskriptien generointiin suunniteltu työkalu. Generaattoriin syötetään rivitettyä SQL Server syntaksilla olevaa tietokantaskriptiä, jonka työkalu parsii, muuntaa ja generoi Oracle ja SQL server yhteensopiviksi. Alla olevasta työkalun esimerkkikuvasta on karsittu toiminnallisuuksia pois.



Kuvio 9. NTier SQL Generator tietokantaskriptien generointiin.

Generaattori on rakennettu helpottamaan kahden eri tietokantajärjestelmän (SQL Server ja Oracle) tietokantojen kehitystä ja ylläpitoa. Vaikkakin Oracle järjestelmiä on käytössä lukumääräisesti SQL Server järjestelmiä vähemmän, käytetään työkalua kuitenkin jatkuvasti sen tuomien hyötyjen takia tuotannon tietokantamuutosten hallinnassa. Generaattori toimii hyvin siihen tarkoitukseen, mihin se on luotu mutta kaikkia nykypäivän haasteita se ei kuitenkaan ratkaise.

Tietokantaskriptien ja tietokantaversionnin hallintaan olisi hyvä löytää jokin kolmannen osapuolen tehokas ja monikäyttöinen työkalu, joka ennaltaehkäisevästi ratkaisisi esimerkiksi suurien ohjelmistopäivitysten tietokantaan liittyviä ongelmia. Ylläpitoa ja hallittavuutta ei kuitenkaan saavuteta pelkillä työkaluilla vaan ne vaativat myös suunniteltuja yhteisiä toimintatapoja ja oikeaan suuntaan vievän ohjelmistokehityskulttuurin.

4.2 Toimitusprosessin tunnistaminen

4.2.1 Ohjelmistopaketin kokoaminen

Toimituksen oleellisin osa on itse ohjelmisto eli binääritiedostot (ks. termit), josta se koostuu. Toimitettava ohjelmistokokonaisuus saattaa koostua yhdestä tai useammasta tuotteesta ja niiden osapalveluista, eli ohjelmistopaketteja voi toimituksessa olla tilanteesta riippuen esimerkiksi yhdestä viiteen. Työkalut -osiossa mainittu TeamCity, eli jatkuvaan integraatioon käytetty työkalu tekee ohjelmistojen käännökset (ks. termit), josta toimitettavan kokonaisuuden osaset kerätään. Näitä keskitetyn käännösympäristön valmistelemia ohjelmistobinääripaketteja kutsutaan siis artefakteiksi. Liitteessä 3. on kuvattu ohjelmistopaketin kokoamisen kokonaiskuvaus.

Ohjelmistopaketin kokoaminen on siis binääripakettien myötä automatisoitu artefaktien tuottamiseen asti ja toimituspakettia tekevä henkilö tekee tästä eteenpäin kokonaisuuden kasaamisen käsin. Automaatio auttaa ja nopeuttaa paketin tekoa omalta osaltaan mutta kokonaisuuden hallintaan ei ole olemassa yksikäsitteistä prosessia tai helpottavaa työkalua. Näin ollen esimerkiksi toimitettavien, eri ohjelmistojen versioiden yhteensopivuudesta saattaa ilmentyä ongelmia.

Toimitusta tehdessä on tärkeää, että jokaisesta tuotteesta ja ohjelmasta toimitetaan asiakkaalle tietty versio ja että nämä kyseiset versiot ovat yhteensopivia toistensa kanssa. Kun keskitettyä hallintaprosessia ei ole, on versioiden ja niiden yhteensopivuuksien selvittely aina manuaalista työtä, mikä tekee prosessista pidemmän. Automatisoituun toimituspaketin kokoamiseen verrattuna manuaalinen kasaaminen on hitaampaa ja inhimillisten virheiden riskit ovat suuremmat.

4.2.2 Muutoshistorian valmistelu

Muutoshistorian valmistelu on yksi toimitusprosessin oleellinen seikka, jolla tuotetaan asiakkaalle välitöntä arvoa. Muutoshistorian valmisteluun käytetään aiemmin esiteltyä EasyChangelog työkalua, jonka avulla tuotteiden versiohaarojen muutokset ja työtehtävähallinnan (Jira) data yhdistetään.

Muutoshistorian ideana on toimittaa asiakkaalle tieto siitä, mikä ohjelmistossa on muuttunut, kun sitä vertaillaan heidän käytössä olevaan versioon. Muutostiedot haetaan siis toimitettavan uuden version ja asiakkaalla käytössä olevan vanhan version välille. Viimeksi toimitetun tuotteen versionumero on yleensä merkittynä toimituskansion rakenteeseen mutta mitään standardisoitua käytännettä ei ole. Näin ollen varsinkin vanhempien toimitusten versiotietojen selvittelyyn saattaa kulua yllättävänkin paljon aikaa.

EasyChangelog työkaluun syötetään toimitettavan ohjelmiston versionhallinan haara ja versioväli, jolle muutostiedot halutaan hakea. Työkalu hakee versionhallinnasta tälle versiovälille muutokset, jonka jälkeen niihin integroidaan työtehtävähallinnasta (Jira) tarkempaa tietoa. Tämä saattaa kuulostaa kätevältä, helpolta ja nopealta mutta koko muutoshistoriaprosessi vie silti yllättävän paljon aikaa.

Yhdessä toimituksessa saatetaan toimittaa asiakkaalle esimerkiksi kolme eri ohjelmistoa ja prosessiin kuluva aika kasvaa jokaisen epäselvän ohjelmistoversion kohdalla. Jokainen epäselvä versio täytyy selvittää, mikä on aikaa vievää.

Manuaalista työtä aiheuttaa myös muutoshistoriatiedostojen siivoilu, sillä niissä saattaa esiintyä asiakaskohtaisia- tai sisäisiä kehityksiä, joita ei yksinkertaisesti saa tai haluta toimittaa.

Muutoshistorian tuottamiseen on olemassa siis jo todella hyödyllistä automaatiota mutta kokonaisuudessaan prosessiin saattaa silti kulua useampi tunti. Jos oletetaan, että Difs-tuotteita toimitetaan eri asiakkaille yhteensä esimerkiksi 5 kertaa viikossa, on prosessiin kuluva hukka ajan myötä suhteellisen merkittävää.

4.2.3 Tietokantaskriptit

Ohjelmistokehittäjien tekemiä tietokantamuutoksia ylläpidetään versionhallinnassa. Tehdyt tietokantamuutokset ajetaan manuaalisesti aiemmin mainitun Ntier SQL Generaattorin lävitse, jonka jälkeen ne lähetetään versionhallintaan. TeamCityn käännöskonfiguraatioihin on tehty toiminnallisuuksia siten, että tietokantamuutokset siirtyvät käännöksen yhteydessä automaattisesti sisäisen testiympäristön tietokantaan.

Kun ohjelmistoja toimitetaan, täytyy asiakkaan ympäristöön viedä binääripäivitysten lisäksi myös samaa ohjelmistoversiota vastaavat tietokantamuutokset. Toimitettavan ohjelmiston binäärit on saatavilla TeamCityn, eli jatkuvan integraation tuottamista artefakteista, mutta tietokannan päivitykset joudutaan sen sijaan tutkimaan ja kokoamaan manuaalisesti.

Toimituspakettia valmisteleva henkilö käy toimitettavan tuotteen tietokantamuutokset versionhallinnasta läpi, kokoaa muutoksia sisältävät ja muut tarvittavat skriptit ja siirtää niistä kopiot varsinaiseen toimituskansioon. Pakettia tekevä henkilö selvittää myös sen, missä järjestyksessä skriptit tulee suorittaa. Nämä asiakkaalle toimitettavat muutokset siis kootaan, järjestellään ja myöhemmin myös ajetaan asiakkaan ympäristöön manuaalisesti.

Muutoksia voi kehitysten aikana tulla useisiin skripteihin ja on olemassa myös skriptejä, jotka rakenteensa vuoksi voidaan ajaa tietokantaan vain yhden kerran. Ihmisten tekemässä manuaalisessa tietokantamuutosten toiminnassa on olemassa inhimillisen virheen vaara ja automatisoituun järjestelmään verrattuna se vie enemmän aikaa. Tietokantaan liittyviä vaikeuksia saattaa ilmentyä myös epäsäännöllisemmissä ohjelmistopäivityksissä.

Kun ohjelmistosta toimitetaan asiakkaalle uusi versio, tulee myös tietokannan versio nostaa vastaavalle tasolle. Jos toimituksesta on kulunut pidempi aika ja tietokanta-

muutoksia on tullut paljon, saattaa päivittämisestä tulla erittäin haasteellista. Tietokannoista ei siis ole olemassa tuotteisiin sidottuja virallisia versioita, vaan suurien versiopäivitysten yhteydessä tehdään tietokannan päivitys suurimmaksi osaksi manuaalisesti.

Käsin tehty manuaalinen selvitystyö ja korjaileminen vievät huomattavasti aikaa. Suurien manuaalisten, ei standardisoitujen muutosten verifiointi ja testaaminen vie oman aikansa ja jokaisen eri asiakkaan kohdalla haluttuun lopputulokseen pääseminen voi olla pitkän selvittelyn takana.

Tietokantaskriptien toimittaminen on oleellinen osa toimitusta ja siinä on olemassa omat haasteensa. Skriptien osalta automatisointia on olemassa sisäisessä jatkuvan ohjelmistointegraation prosessissa mutta toimitusten osalta tietokantamuutosten kokoaminen tapahtuu pitkälti manuaalisesti. Binääri-, ja tietokantamuutosten lisäksi oleellinen osa ohjelmistotoimitusta, on konfiguraatiomuutokset.

4.2.4 Konfiguraatiomuutokset

Ohjelmiston päivityksessä sen rakenteet yleensä muuttuvat. Uudet toiminnallisuudet, erilaiset toimintatavat tai yksinkertaisesti lisävaihtoehtojen tarjoaminen saattavat vaatia ohjelmiston konfigurointia. Konfigurointi voi olla ylläpidollista, sovellusympäristöön vaikuttavaa tai ohjelmistoon ja sen käyttäytymiseen vaikuttavaa asetusten uudelleenasettelua. Konfigurointi voi olla ohjelmiston asentajan vastuulla, ylläpitäjän vastuulla tai asiakkaan vastuulla mutta tärkeintä on konfigurointitarpeen tiedostaminen.

Difs-tuotteiden toimituksissa konfiguraatiomuutosten toteuttaminen on loppujen lopuksi ohjelmistopaketin asentajan vastuulla. Uusien konfigurointitarpeiden esille tuominen on kuitenkin jokaisen yksittäisen kehittäjän vastuulla ja EasyChangelog työkalua sekä Jiraa, käytetään prosessissa apuna. Muutos, joka vaatii uudelleenkonfigurointia, kulkeutuu Jiran kautta EasyChangelog työkalulle, siitä edelleen muutoshistoriatietoihin ja sitä kautta toimituspaketin asentajalle. Asentaja tulkitsee uusien- ja olemassa olevien konfiguraatioiden muutostarpeet ja toteuttaa ne asennuksen yhteydessä tai viestii ne eteenpäin asiakkaalle.

Jiran ja EasyChangelogin kautta kulkeutuu siis ainoastaan kehitykset jotka vaativat konfigurointia, ei varsinaisia konfiguraatiomuutoksia. Kehityksen toteuttanut henkilö kommunikoi konfiguroinnin tarpeen joko kehitystiketille, jolta asentaja voi sen tarkistaa tai käy kirjoittamassa ohjeet tulevan ohjelmistoasennuksen yhteyteen.

Konfiguraatiomuutoksia saattaa ilmentyä versiopäivityksissä todella paljon, eikä asiakaskohtaista keskitettyä konfiguraatiohallintaa ole olemassa. Konfiguroinnissa on myös välikäsiä, mikä nostaa inhimillisen virheen riskiä ja monesti asiakas- tai versiokohtaiset konfigurointitarpeet ovat dokumentoimatta. Manuaalista työtä tehdään niin konfiguroinnin selvittelyiden, kuin itse konfiguroinnin osalta ja työ on pitkälti kokeneemmista ohjelmistokehittäjistä riippuvaa.

4.2.5 Ohjelmistopaketin asennus

Ohjelmistonpaketin kokoamisen jälkeen voidaan sen asennus asiakkaan ympäristöön aloittaa. Asentajana saattaa toimia paketin tekijä, joku muu sisäinen henkilö tai joissain tapauksissa myös asiakkaan edustaja. Ohjelmistoasennukseen kuuluvat ainakin seuraavat vaiheet:

1. Windows Services -palveluiden sammuttaminen
2. Binääritiedostojen päivittäminen
3. Tietokantaskriptien ajo tietokantaan
4. Konfiguraatiomuutokset
5. Internet Information Services (IIS) käynnistys
6. Windows Services -palveluiden käynnistys.
7. Internet Information Services (IIS) sammuttaminen.

TeamCityllä toteutettavaan sisäiseen ohjelmistointegraatioon verrattuna ohjelmistopaketin asentaminen asiakkaan ympäristöön on lähes identtinen prosessi. Sisäisesti toteutettu CI järjestelmä on konfiguraatiomuutoksia lukuun ottamatta automatisoitu kokonaan mutta asiakkaalle tehtävä asennus tehdään silti edelleen manuaalisena käsityönä.

Joissain tapauksissa asiakas tekee itse ohjelmistopäivityksen asennuksen omaan ympäristöönsä. Näissä tapauksissa asiakkaalle toimitetaan jokaisen asennuksen yhteydessä päivitettävä Microsoft Word dokumentti, joka sisältää asennusohjeet ja mahdolliset konfiguraatiomuutokset.

Asiakkaalle toimitetaan extranet palveluun ohjelmiston binääripaketit ja tietokantaskriptit, jonka jälkeen asiakas tekee päivityksen ohjeiden perusteella omaan ympäristöönsä. Tämä tuo ohjelmiston toimitukseen vielä yhden ylimääräisen välikäden, mikä luonnollisesti hidastaa prosessia ja nostaa inhimillisen virheen todennäköisyyksiä.

Sikäli, kun prosessi on manuaalinen, on virhetilanteista palautuminen myös hitaampaa ja työläämpää. Jos ohjelmistopäivitys epäonnistuu tai sen toiminnassa havaitaan virheitä, on vanhaan versioon palaaminen aina henkilötyön takana oleva ponnistus. Manuaalisen asentamisen takia ohjelmiston versiopalautuksen automatisoinnin toteuttaminen olisi myös huomattavan haastavaa.

4.2.6 Kokonaisuuden hallinta

Difs-tuotteiden toimitusten kokonaisuuksien hallintaan on olemassa muutamia keinoja. Kokonaisuuden hahmottamiseen käytetään työtehtävien hallinnan Jiraa, johon luodaan asiakastoimituksista työtehtävät ja organisaation sisäistä Wikiä (ks. termit), johon listataan kaikki yksittäiset asiakastoimitukset viikoittain.

Wikiin on siis rakennettu toimituskalenteri, jonka viikoittaisiin listoihin integroidaan toimitustöiden tiketit Jirasta. Jokaista asiakastoimitusta varten tehdään Jiraan tomitustiketti, jonka yhteyteen linkitetään asiakkaan kanssa sovitut kehitykset ja mahdolliset korjaukset. Näin saadaan luotua kokonaiskatsaus siitä, mitä, milloin ja kenelle, ollaan toimittamassa.

Linkitetyt kehitykset ja korjaukset eivät useissa tapauksissa kata kuin pienen osan samaan tuotehaaraan tehdyistä kehityksistä. Näin ollen muutoshistorialle saattaa tulla paljon muutakin, kuin kehitykset joista tämän asiakastoimituksen tiimoilta ollaan valveutuneita.

5 Difs - organisaation toimituskäytänteiden kehitys

5.1 Asennuksen automatisointi

5.1.1 Lähtökohdat

Rajoitetun kehitys- ja tutkimusajan puitteissa päätettiin tehdä kehitykseen tiettyjä rajauksia. Todettiin, että toimituspaketin ja toimitusasennuksen automatisoinnista saataisiin tässä vaiheessa eniten hyötyä, joten näitä kahta kohdetta lähdettiin työstämään.

Tutkimuksen aikana tehty automatisoidun asennuksen toteutus on MVP-ajatusmallin mukainen minimalistinen versio, jonka tarkoituksena oli mahdollisimman nopeasti todeta automaattisen asennuksen mahdollisuudet sekä tulevaisuuden- että jatkokehityksen suuntaviivat.

Asennusprosessin yksi suurimmista tavoitteista oli saada ihmisten tekemä manuaalinen työ minimoitua. Tätä kautta kokonaisprosessia saataisiin nopeutettua, inhimillisten virheiden määrä pienennettyä ja kokonaisuus mahdollisimman standardoiduksi. Automaatiolla haettiin myös asiakaskokemuksen parantamista ja asennuksen tulisi toimia ilman internet yhteyttä.

Automaattisen asennuksen ja toimittamisen tekemiseen on olemassa useita mahdollisuuksia. Yksinkertaisin ratkaisu on lähteä toteuttamaan asennusta itse tehtyjen skriptien kautta. Difs-tuotteiden jatkuvan integraation prosessissa on jo olemassa tämänkaltaisia itse rakennettuja skriptikokonaisuuksia, jotka hoitavat joitakin osia testipalvelimen ohjelmistoasennuksesta. Vaikka skriptit saattavat joissakin tapauksissa olla täysin toimivia, haluttiin tässä yhteydessä kuitenkin lähteä etsimään kokonaisvaltaisesti helpommin hallittavaa toteutusta. Suurimpia haasteista tässä työssä olivat edelleen konfiguraatioiden ja tietokantamuutosten hallinta, joihin installer-pakettien toivotaan muun muassa tuovan ratkaisuja.

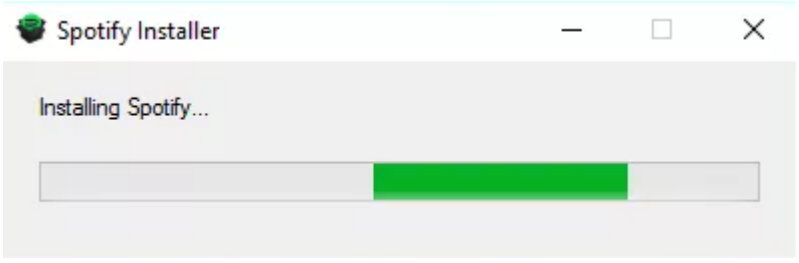
Yhtenä ratkaisuvaihtoehtona tutkittiin OctopusDeploy (Octopus Deploy 2017) nimistä keskitettyyn toimittamiseen rakennettua sovellusta. Tämä vaihtoehto kuitenkin hylättiin heti alkuvaiheessa, sillä sovelluksen vaatimat etäyhteydet eivät sovellu Financial Solutionsin tarpeisiin.

Microsoft Windows Installer

Microsoft Windows Installer on Windowsin asennus- ja konfiguraatiopalvelu, joka mahdollistaa standardisoidun tavan asentaa ja hallita ohjelmistoja Windows alustoille. Windows Installer tarjoaa rajapinnan, jonka avulla voidaan hallita käyttöjärjestelmän rekisteritietoja, tiedostoja, kansioita, Windows lisäominaisuuksia ynnä muita. (Windows Installer 2017.)

Installerin ideana on kääriä asennettavan ohjelmiston tiedostot yhteen pakettiin niin, että kaikki asennuksessa tarvittavat säädöt, konfiguroinnit, kolmannen osapuolen sovellukset ja asetukset voidaan asettaa automaattisesti tai käyttäjän syötteen perusteella. Installer-paketilla voidaan tarjota ohjelmiston asentavalle, päivittävälle tai poistavalle osapuolelle saumaton käyttökokemus niin, ettei asentajan välttämättä tarvitse tietää tuotteen asennuksesta yhtään mitään.

Spotify on hyvä ääriesimerkki asennuksesta, joka ei vaadi käyttäjältä mitään interaktiota asennuksen käynnistämisen jälkeen. Kuvio 10. nähdään Spotifyn yksinkertainen Windows Installer asennusikkuna. Asennuspaketti on konfiguroitu lataamaan tarvittavat tiedostot asennuksen aikana ja asentamaan ohjelmiston tiettyyn paikkaan ja tietynlaiseen kansiorakenteeseen. Päivitykset Spotify hoitaa ohjelmiston käynnissä ollessa lataamalla ne päivityspalvelimelta ja asentamalla ne automaattisesti taustaprosessina. (Spotify 2017.)



Kuvio 10. Spotify Installer Windowsille.

Windows Installer rajapinnan päälle on rakennettu useita erilaisia kolmannen osapuolen sovelluksia, joilla asennuspaketin kokoamisen voi toteuttaa. Esitutkimuksen aikana nousi esiin useita asennuspaketin tekoon erikoistuneita sovelluksia. IndigoRosen SetupFactory (Indigo Rose 2017), Flexeran InstallShield (Install Shield 2017), WixToolset (Wix Toolset 2017) ja Advanced Installer (Advanced Installer 2017) olivat

kaikki esitutkimuksen aikana esiin nousseita potentiaalisia työkaluja. Advanced Installer valikoitui käytettäväksi työkaluksi sen hyvien vertaiskokemusten ja ilmaisen kokeilulisenssin perusteella.

Installerin käytön ideana on siis automatisoida ja muokata asennusprosessi standardisoiduksi niin, että asennusta tekevän henkilön ei tarvitse tehdä mitään ylimääräisiä vaiheita manuaalisesti. Tarkoituksena on, että kaikki mikä kyetään ja katsotaan parhaaksi automatisoida, tullaan lopulta automatisoimaan.

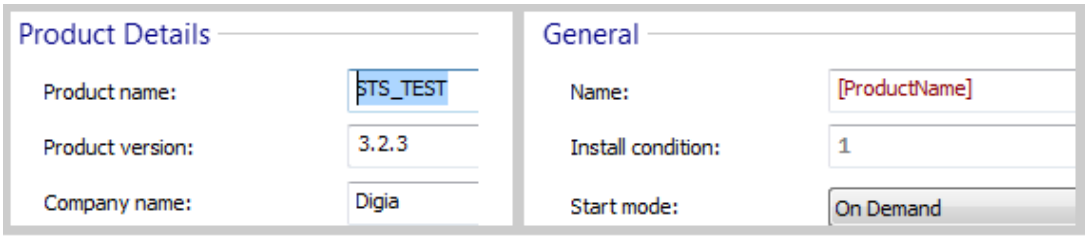
5.1.2 Advanced installer

Advanced Installer on Windows ympäristöihin asennettavien sovellusten asennuspakettien tekoon erikoistunut ohjelmisto. Advanced Installer tarjoaa ilmaisen, riisutun version ohjelmistostaan ja kolmenkymmenen päivän kokeilulisenssin monimuotoisempiin toiminnallisuuksiin.

Advanced Installer tarjoaa helppokäyttöisen käyttöliittymän ohjelmistopaketin kokoamiseen. Työkalun verkkosivuilla sekä itse sovelluksessa, on olemassa todella hyvä dokumentaatio sen toiminnallisuuksista. Ohjelmiston käyttö osoittautui myös käytännössä todella vaivattomaksi. Tämänhetkisen kokemuksen perusteella työkalu tarjoaa kaikki asennukseen tarvittavat prosessivaiheet sekä mahdollistaa asennuksessa tarvittavien muuttujien monimuotoisen konfiguroinnin.

Tuotetiedot, versiointi ja asennustapa

Asennuspaketin eli installerin kokoaminen aloitettiin tuotetietojen syöttämisellä ja määrittelyllä. Tietoihin määriteltiin perustietoina tuotteen nimi, versio, organisaation nimi ja niin edelleen. Kiinnostavana ja hyödyllisenä ominaisuutena todettiin sisäisten muuttujien käyttö hakasulje-avaimien avulla. Kuvio 11. vasemmalla puolella näkyy perustiedoissa määritelty tuotteen nimi STS_TEST. Oikealla puolella näkyy ote IIS määrittelyistä, joissa nimen kohdalla on käytetty viittausta ProductName tuotteen nimeen. Muuttujien käytön havaittiin tuovan haluttua dynaamisuutta ja uudelleenkäytettävyyttä Advanded Installer projekteihin.



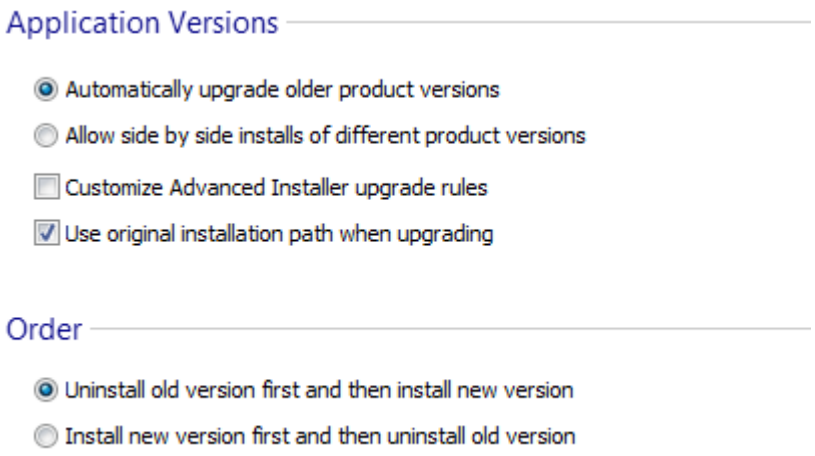
Kuvio 11. Advanced Installer tietojen referointi avaimien avulla.

Tuotetietoihin määriteltiin myös Windows rekisteröinti. Difs-tuotteita ei ole aikaisemmin asennettu viralliseen tapaan Windowsin rekisteritietoihin, joten tuotteet eivät ole Windowsin mukaan olleet virallisesti edes olemassa. Rekisteröinti mahdollistaa tuotteen poistamisen Windowsin ohjauspaneelin kautta ja ylläpitää asennettujen tuotteiden versiotietoja.

Versiointi tulee tulevaisuudessa helpottamaan ohjelmistojen ylläpitoa niiden päivitysten osalta. Kun versiointi on hoidettu Windowsin rekisterin kautta, on asennuspaketin avulla helppo tarkastella, mikä versio kyseiselle palvelimelle on asennettu ja tehdä tarvittavat toimenpiteet sen perusteella. Tarvittavat asennusvaiheet voidaan määritellä kertaalleen ja tarpeen vaatiessa korjata asennuspaketin projektitiedostoihin niin, että muutoksia ei tarvitse tehdä manuaalisesti montaa kertaa.

Asennuspaketin versiointi asetettiin toistaiseksi manuaalisesti mutta Advanced Installer mahdollistaa versionumeron hakemisen myös käännetyistä binääri-tiedostoista tai esimerkiksi käyttöliittymän .exe-tiedostosta. Kappaleessa 5.2 esitellään, kuinka versiointi asetetaan TeamCityn kautta.

Versioinnin perusteella tuotteen asennustavaksi pystytään määrittelemään joko päivitys tai rinnakkaisversion asennus. Tässä työssä käytettiin päivittämistä, joka toimi täysin tarkoituksenmukaisesti. Kuvio 12. nähdään Advanced Installerin yksinkertaiset päivitysasetukset. Päivittäminen toimii kyseisillä asetuksilla niin, että vanha tuoteversio poistetaan aina ennen uuden version asentamista. Jos asennuksessa ilmenee ongelmia, tekee asennusprosessi automaattisen palauttamisen eli palauttaa ohjelmiston edellisen version.



Kuvio 12. Advanced Installer – tuotteen asennustapa.

Silent- eli tausta-asennus havaittiin todella hyödylliseksi. Jos Windows Installer-paketti ajetaan Silent-komennolla, toteutetaan asennus kokonaisuudessaan taustaprosessina niin, että se ei vaadi minkäänlaista käyttäjäsyötettä. Silent-asennus antoi hyvää osviittaa jatkuvan toimittamisen automatisoinnin mahdollisuuksista, sillä silent-asennuksen avulla asiakastoimitukset olisi tietyissä tapauksissa mahdollista tehdä täysin ilman ihmisen vaatimaa interaktioita. Asennukset voisi siten myös ajaa myös verkon yli etänä.

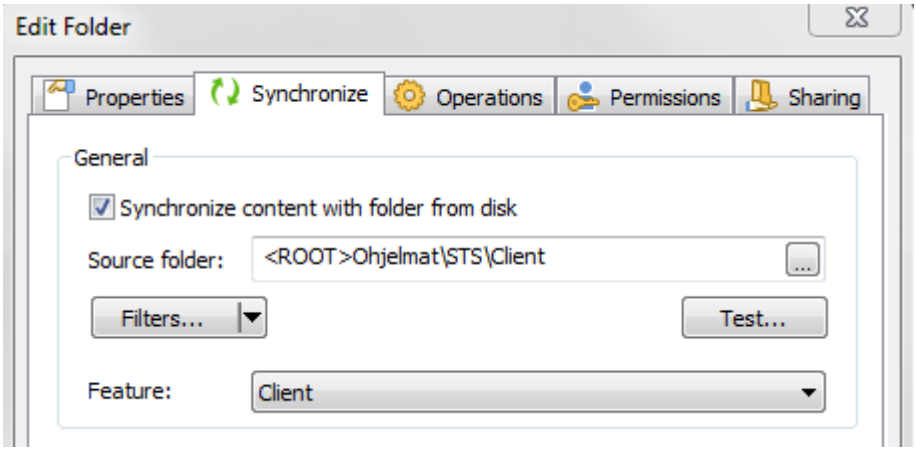
Lähdetiedostot ja ohjelmien organisointi

Asennuspakettia varten tarvittiin ohjelmiston binääritiedostot. Paketin tekoon käytettiin Difs-pankkituotteen päähaaraa, jonka binäärit saatiin TeamCity käännöksen artefakteista. Kappaleessa 5.2 on kuvattu, kuinka lähdetiedostojen koostettu kokoaminen TeamCityn avulla toteutettiin.

Tiedostojen asettaminen tehtiin Advanced Installerin kansioiden synkronointi-toiminnolla, joka yhdessä projektikohtaisen juuri-muuttujan avulla mahdollistaa myöhemmin asennuspaketin dynaamisen kääntämisen TeamCityn avulla.

Kuvio 13. nähdään synkronoidun kansion asettaminen. Tiedostopolun alkuun on asetettu Advanced Installerin tarjoaman projektikohtaisen muuttujan avain ROOT. ROOT avain vastaa asennuspaketin juurihakemiston tiedostopolkua ja kappaleessa 5.2 esitellään sen asettaminen TeamCityn kautta komentorivikehotteena. Juuripolun käyttö

mahdollistaa Advanced Installer projektien dynaamisen käytön, sillä käytettävien binääritiedostojen tiedostopolkuja ei tarvitse syöttää absoluuttisesti vaan niitä voidaan ROOT muuttujan avulla käyttää dynaamisesti.



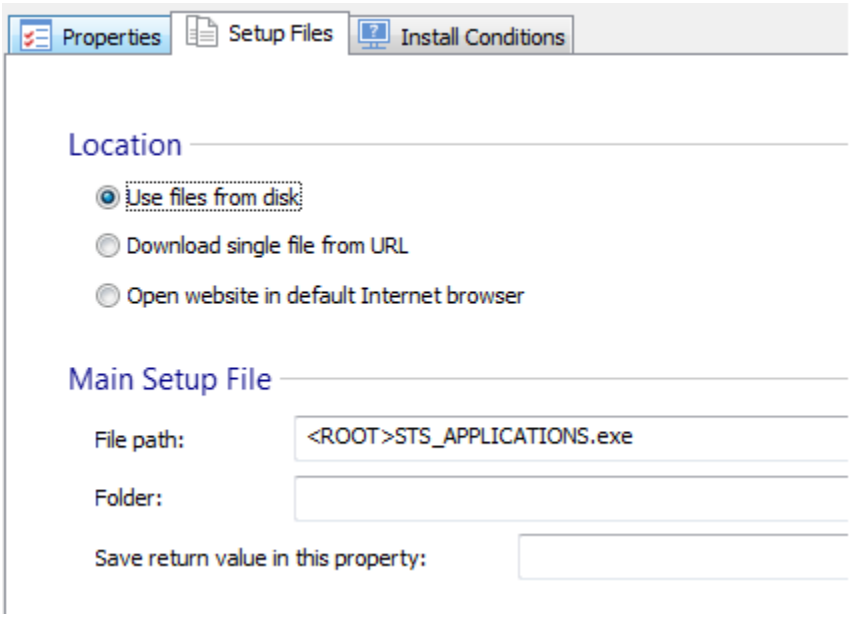
Kuvio 13. Advanced Installer kansion synkronointi.

Difs-tuotteen toimittamisen yhteydessä ohjelmistoja saatetaan asentaa esimerkiksi yhdestä viiteen. Tämän työn yhteydessä käsiteltiin Difs-pankkituotetta, joka päätettiin myöhemmin jakaa kahteen eri asennusprojektiin. Installerin liitettiin myös pankkituotetta käyttävä rajapintatoteutus.

Ohjelmistojen binääritiedostojen lajittelu tehtiin aluksi Advanced Installerin feature toiminnolla. Feature toiminnon avulla eri ohjelmistokomponentit ja palvelut pystytään erottelemaan erikseen asennettaviin lohkoihin. Sovellus jaettiin komponentteihin siten, että käyttöliittymä (Client), palvelinsovellus (Server) ja Windows palvelut saivat oman asennuslohkonsa. Asennuksen yhteydessä käyttäjä pystyi päättämään, mitkä komponentit asennetaan, päivitetään tai poistetaan.

Kasaamalla kaikki komponentit yhteen projektiin Advanced Installerin feature toiminnolla, ei kuitenkaan saavutettu haluttuja lopputuloksia. Tulevaisuutta ajatellen pääsovellus jaettiin kahteen osaan niin, että palvelin ja käyttöliittymä ovat yhden installer-projektin alaisuudessa ja Windows palvelut toisen. Tämä lähestymistapa mahdollistaa usean sovelluksen liittämistä yhteen asennuspakettiin niin, että samat sovellukset on mahdollista asentaa myös yksittäin eri palvelimille. Projektitiedostojen ylläpito on myös helpompaa, kun kaikki tuotteet on jaoteltu omiin projekteihinsa.

Asennusprojektien jaon jälkeen päädyttiin käyttämään Advanced Installerin easivaatimussovelluksia (Prerequisites). Esivaatimussovellusten avulla yhteen asennuspakettiin pystytään linkittämään useita sovelluksia niin, että jokainen linkitetty sovelluksista voidaan asentaa pääasennuksen yhteydessä peräkkäin. Kuvio 14. nähdään, kuinka myös esivaatimuksien määrittelyissä käytettiin <ROOT> projektimuuttujaa asennuksen dynaamisen kokoamisen mahdollistamiseksi.



Kuvio 14. Advaned Installer esivaatimussovellusten tiedostoasetukset.

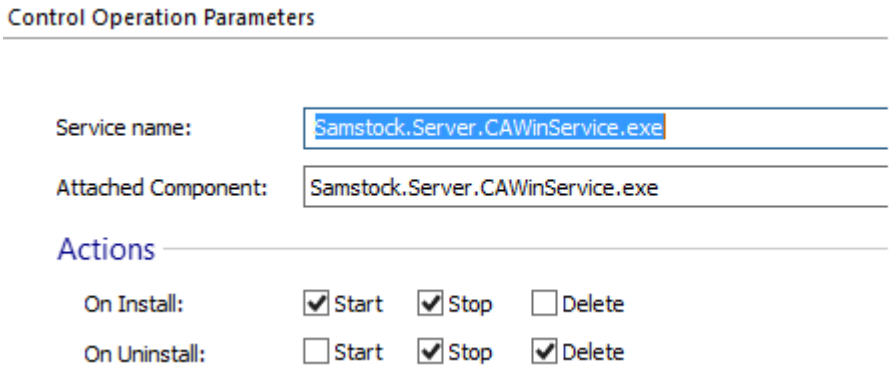
Asennuspaketin kokoaminen esivaatimussovelluksien avulla vaatii sen, että jokainen esivaatimukseksi määritelty sovellus tulee kääntää ennen varsinaisen pääohjelman kääntämistä. Esivaatimussovellusten kääntäminen esitellään kappaleessa 5.2.1.

Toimitettavien tuotteiden linkittäminen tällä tavoin mahdollistaa monimuotoisten asiakaskokoonpanojen tukemisen. Asiakas saattaa haluta, esimerkiksi kuorman tasaamisen vuoksi, asentaa kaikki Difs-tuotteet eri palvelimille. Kun jokainen tuote tai jaottelua vaativa tuotteen osa pidetään omassa projektissaan, on usean erillisen asennuspaketin toimittaminen asiakkaalle mahdollista. Tuotteet siis yhdistettiin esivaatimussovellusten avulla mutta ne voidaan toimittaa myös erikseen.

Windows palvelut

Difs-ohjelmisto käyttää Windows Service -palveluita joidenkin ohjelmiston toiminnallisuuksien hallintaan ja ajoon. Advanced Installerilla määriteltiin asennettavat Windows palvelut sekä niiden sammutus ja käynnistys asennuksen yhteydessä. Ohjelmistopäivityksen yhteydessä olemassa olevat palvelut päivitetään ja poiston yhteydessä installer-ohjelma osaa myös poistaa ne.

Kuvio 15. nähdään Windows palveluiden kontrollointimahdollisuudet asennuksen aikana. Palvelut voi ohjelmiston asennuksen aikana käynnistää, pysäyttää tai poistaa. Asennuksen yhteydessä palvelut on ensin sammutettava, jotta Windows Installer pystyy ylikirjoittamaan aikaisemmin asennetut tiedostot uudessa asennuksessa. Ohjelmiston poistossa palvelut asetettiin luonnollisesti poistettavaksi.



Kuvio 15. Advanced Installer Windows palvelun toiminnot.

Ongelmia syntyi testausvaiheessa palveluiden käynnistämisestä, sillä konfiguraatiollisista syistä muutamaa palvelua ei saatu käynnistettyä oikein. Advanced Installerista ei löytynyt Windows palveluiden käynnistykseen ehtoja niin, että palvelun käynnistyksen epäonnistuttua asennusta jatkettaisiin. Ainoa vaihtoehto epäonnistuessa on asennuksen keskeytys ja vanhan asennuksen palautus. Tämä ei tulevassa tuotantokäytössä ole sinänsä ongelma vaan itseasiassa hyödyllinen ominaisuus mutta testausvaiheessa se tuotti harmia.

Konfigurointi

Tavallisesti Difs-tuotteen asennuksen yhteydessä ei kuljeteta konfiguraatiotiedostoja. Konfiguraatiot asetetaan ensimmäisen asennuksen yhteydessä manuaalisesti, jonka jälkeen niitä ainoastaan päivitetään. Myös päivitys tapahtuu manuaalisesti.

Asennus tehtiin aluksi täysin ilman konfiguraatiotiedostoja ja asennuksen jälkeen kaikki konfiguraatiotiedostot asetettiin manuaalisesti omille pakoilleen ja muutettiin oikeanlaisiksi. Konfiguraatiotiedostot lisättiin tämän jälkeen mukaan lähdetiedostoihin manuaalisesti, jotta voitiin testata, miten niiden ylläpitäminen asennuspaketin kautta toimisi.

Advanced Installer pystyy tekemään päivityksiä olemassa oleviin konfiguraatiotiedostoihin mutta tarkoitukseen sopivaa ylläpitoa ja muokkausta ei ohjelmiston kautta pystytty toteuttamaan. Näin ollen kokonaisvaltaisen konfiguraatiohallinnan suunnittelu ja toteutus jätettiin vielä toistaiseksi toteuttamatta.

Asennuspaketti .MSI ja .EXE

Käännöksen osalta kokeiltiin MSI- ja EXE asennuspaketteja, joiden osalta eroa ei aluksi huomattu. Lopulta modernimman EXE paketin huomattiin olevan parempi vaihtoehto, sillä sen pakkaamiseen voi käyttää monimuotoisempaa pakkauslogiikkaa. EXE-muotoisen asennuspaketin voi myös optimoida asennusajan suhteen, mitä MSI-paketille ei pysty Advanced Installer työkalulla tekemään.

Kumpaankin asennustyyppiin pystytään määrittelemään tarvittavien resurssien sijainti eli se, tuottaako Advanced Installer yhden tiedoston, joka sisältää kaiken tarvittavan, vai erotteleeko se tiedostot omiin kansioihinsa. Tämän huomattiin olevan hyödyllistä suurien kolmannen osapuolen sovellusvaatimusten osalta, joita oikean asiakasasennuksen osalta ei haluta toimittaa jokaisella toimituskerralla.

Parametrisoinnin osalta määriteltiin asennushakemisto, jonka käyttäjä pystyy asennusdialogissa ylikirjoittamaan.

Sovellusympäristö ja kolmansien osapuolien ohjelmistot

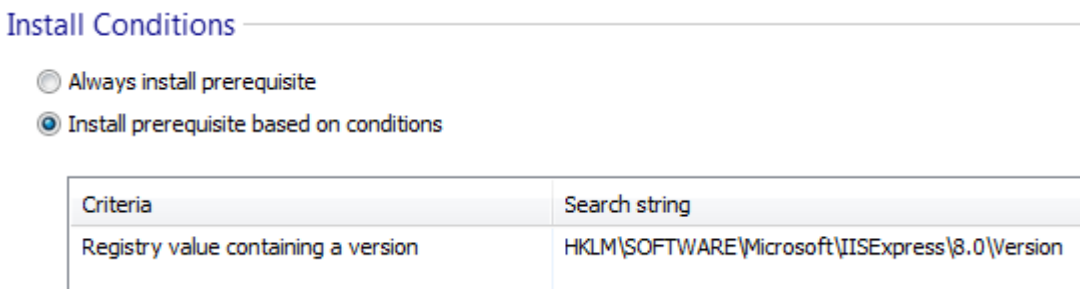
Difs-ohjelmisto vaatii toimiakseen sovellusympäristöltä tiettyjä ohjelmistoja. Näitä ohjelmistoja ovat esimerkiksi .NET 4.5.2 ja ODBC tietokanta-ajurit. MVP-ajattelumallin mukaiseen toteutukseen katsottiin hyväksi lisätä nämä, jotta Difs-ohjelmisto saatiin käynnistymään ja jotta sen toimivuutta voitiin esitellä. Advanced Installeriin määriteltiin asennuskonfiguraatiot kyseisille ohjelmille ja niiden asennukset niin, että käyttäjä pystyy itse ohjaamaan niitä.

Kolmansien osapuolien sovellusten asentamiseen käytettiin samaa Advanded Installerin toimintoa, joka esiteltiin luvussa 5.1.2.2. Kyseessä on siis esivaatimussovellusten asennustoiminto, jota käytettiin myös usean Difs-tuotteen yhdistämiseen saman asennuspaketin alle.

Esivaatimussovellusten avulla asennuspaketista voidaan tehdä kokonaisvaltainen niin, että asennuksen yhteydessä mitään ei tarvitse asentaa tai konfiguroida manuaalisesti. Sovellusympäristön vaatimat ajurit ja sovellukset on aikaisemmin tehty manuaalisena palvelinkohtaisena asennuksena. Asennuspaketin avulla pystytään kokoamaan kokonaisvaltainen ohjelmistopaketti, jonka ylläpito ja hallinta ovat keskitettyjä. Jos jokin Difs-ohjelmiston osa vaatii toimiakseen jonkin kolmannen osapuolen ohjelmiston päivityksen, voidaan se hoitaa asennuspaketin kautta keskitetysti ja luotettavasti.

Vaadittavien ohjelmistojen olemassaolo tarkastellaan Windows-ohjelmien kohdalla joko DLL tiedostojen versioista tai rekisteristä. Kolmannen osapuolen ohjelmistojen kohdalla luotettava tarkistus olisi hyvä tehdä ainoastaan rekisteristä, sillä rekisteritieto on ohjelmistokohtaisesti pitkälti standardisoitu. Tässä toteutuksessa ODBC ajurien olemassaolo tarkastellaan System32 -hakemistossa sijaitsevasta DLL tiedostosta ja .NET versio Windowsin rekisteristä.

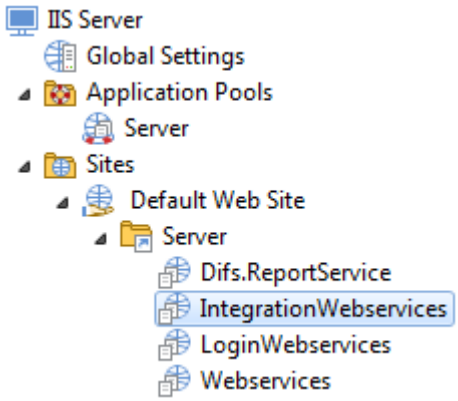
Nativiisovelluksia eli feature pohjaisia Windows asennuksia kuten IIS Expressiä ei tarvitse konfiguroida erikseen, sillä Advanced Installer hoitaa sen automaattisesti. Kuvio 16. nähdään esivaatimussovelluksen asennusehdot, jonka perusteella asennusprosessi päättää, asennetaanko sovellusta vai ei. Advanced Installer on luonut kyseisen asennusehdon automaattisesti mutta määritykset voidaan tehdä myös aikaisemmin kuvatulla tavalla manuaalisesti.



Kuvio 16. Advanced Installer esivaatimussovelluksen asennusehdot.

Internet Information Services (IIS) palvelin

IIS palvelimen konfigurointi onnistuu helposti Advanced Installerin avulla. Kuvio 17. nähdään määritellyt Application Poolit, asennuskansiot ja IIS-applikaatiot. Kaikki edellämainitut sekä itse IIS-palvelimen perusasetukset pystyttiin myös konfiguroimaan halutulla tavalla. Application poolien sammutus ja uudelleenkäynnistys hoituvat automaattisesti asennusprosessin toimesta.



Kuvio 17. Advanced Installer IIS määrittelyt.

Tärkeimpiä konfiguraatioasetuksia olivat IIS Application poolien .NET versiot ja CGI/ISAPI rajoitukset. Rajoitukset määriteltiin jokaisen virtuaaliapplikaation kohdalle kustoimoituihin asetuksiin. Kaikille IIS komponenteille määriteltiin myös asennuskäyttäytyminen niin, että päivityksen yhteydessä asennus ohittaa kaikki jo olemassa olevat komponentit.

5.1.3 Mitä vielä tarvitaan?

Tällä hetkellä asennuspaketti toimii ja toteuttaa MVP mallin mukaiset vaatimukset. Ohjelmisto käynnistyy ja sitä pystytään käyttämään ja prosessi havainnollistaa hyvin asennuspaketin käytön mahdollisuudet. Jatkokehitystä tarvitaan kuitenkin vielä suhteellisen paljon ja haasteita ja suunnittelua on vielä edessä.

Avoimeksi asiaksi jäi ainakin tietokantamuutosten ajaminen asennuksen yhteydessä. Advanced Installerissa on ominaisuus SQL Server tietokantapäivityksien tekemiseen mutta useamman tietokantatyypin, tässä tapauksessa Oraclen, tukeminen saattaa osoittautua haastavaksi. Tietokantaversioinnin rakentamista täytyy myös tutkia ja

pohtia tarkkaan ja käyttöön täytyy luultavasti Installer-sovelluksen lisäksi etsiä jokin tietokantaversiointia ja -päivityksiä helpottava työkalu.

Tulevaisuuden haasteeksi jää myös mahdollisimman automatisoitu ja kokonaisvaltainen konfiguraatioiden hallinta. Konfiguraatiomuutokset täytyy edelleen tehdä manuaalisena työnä ja viedä asiakkaan ympäristöön etäyhteyden yli. Tulevaisuudessa konfiguraatiomuutokset tai ainakin konfiguraatioiden tyhjät oletusarvot olisi hyvä saada kulkeutumaan asiakkaalle asennuspaketin mukana. Asennuspaketin asennuskäyttöliittymää voidaan mahdollisuuksien mukaan myös yrittää hyödyntää uusien konfiguraatioiden asettamiseen.

5.2 Toimituspaketin kokoamisen automatisointi TeamCityllä

Asennuspaketti työstettiin paikallisella työasemalla niin, että käytettävät ohjelmistobinäärit olivat koko ajan samat ja käännökset tehtiin manuaalisesti. Tämä helpotti ja nopeutti asennuspaketin testaamista. Kun asennuspaketti saatiin valmiiksi, päätettiin toimitusputken automatisointia jatkaa. TeamCityn avulla toteutettiin käännöskonfiguraatio, joka kokoaa asiakastoimituksen tuotteet yhteen paikkaan niin, että kaikki toimitettavat tuotteet voitiin keskitetysti kerätä ja kääriä asennuspakettiin automatisoidusti.

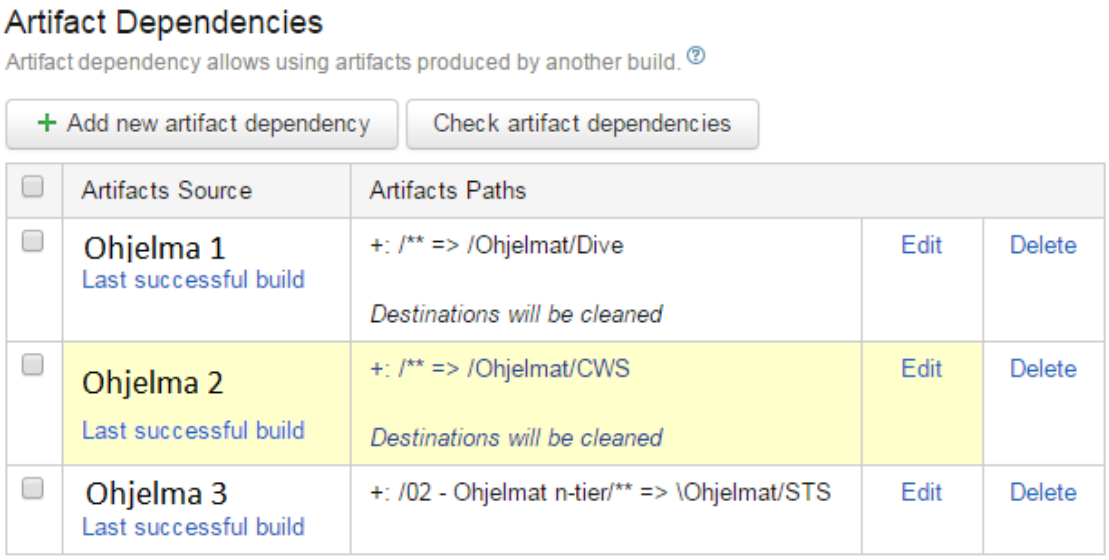
5.2.1 Artefaktiriippuvuudet

TeamCityyn on määritelty jokaiselle toimitetavalle tuotteelle oma käännöskonfiguraationsa ja kaikki nämä käännökset muodostavat aikaisemmin mainittuja artefaktipaketteja. Kyseiset artefaktipaketit linkitettiin TeamCityn Artifact Dependency ominaisuudella uuteen asennuspaketin koostavaan käännöskonfiguraatioon. Liitteessä 1 on esitetty toimituspaketin koostamisen kokonaiskuvaus.

Artifact Dependency -ominaisuus tarkastelee tietyn käännöskonfiguraation muodostamaa binääripakettia. Toiminnallisuuden ideana on varmistaa, että asennuspakettiin päätyy varmasti sama ohjelmistoversio, joka on omassa CI prosessissaan jo kertaalleen verifioitu toimivaksi.

Tällä hetkellä käytössä olevassa manuaalisessa toimituksessa artefaktit on manuaalisesti ladattu toimitusta tekevän henkilön työasemalle, josta ne ovat lopulta kulkeutuneet toimitettavaksi asiakkaalle. Koostetun käännöskonfiguraation ja artefaktiriippuvuuksien avulla toimitettavien tuotteiden versiointia on huomattavasti helpompi hallita.

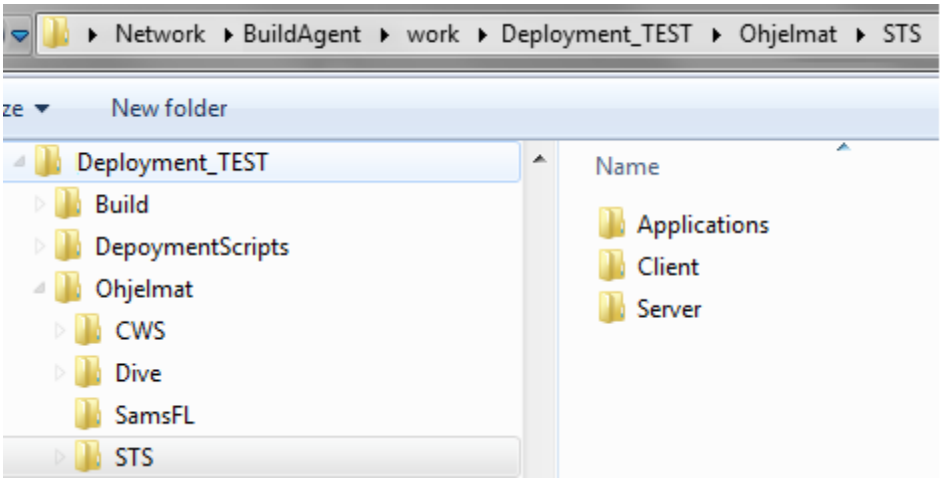
Kuvio 18. nähdään, kuinka artefaktiriippuvuudet asetettiin niin, että jokaisesta yksittäisestä konfiguraatiosta ladataan viimeisin onnistunut käännösversio. Kun kaikkien yksittäisten konfiguraatioiden oikeellisuudesta pidetään huolta, varmistetaan tällä tavoin, että toimitukseen tulee ainoastaan toimivaa ja verifioitua ohjelmistoa.



Kuvio 18. TeamCity koostavan käännöskonfiguraation artefaktiriippuvuudet.

5.2.2 Lähde- ja projektitiedostot

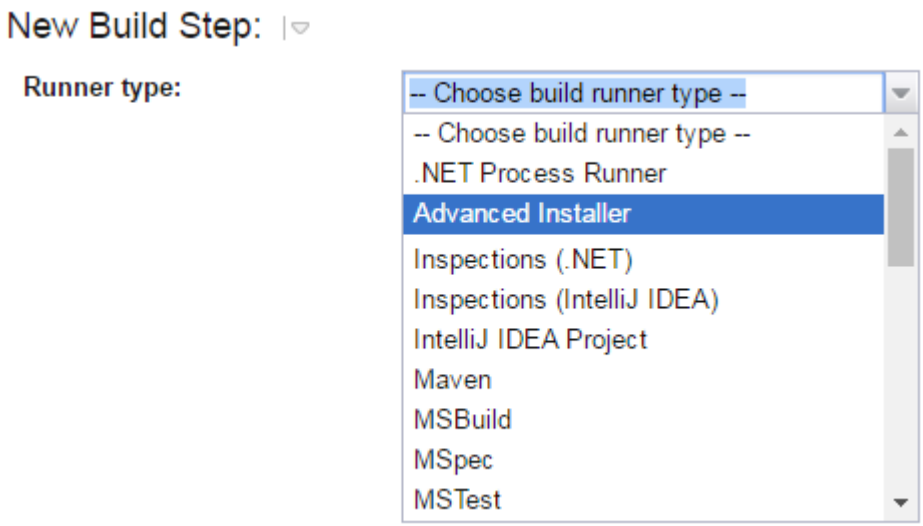
Kuvio 19. nähdään koostavan käännöskonfiguraation kokonainen kansiorakenne käännöstä tekevällä TeamCityn agenttipalvelimella. Kuvasta nähdään myös Build ja DeploymentScripts kansiot, jotka TeamCity on ladannut määritysten mukaan versionhallinnasta. Nämä kansiot sisältävät Advanced Installer projektitiedostot ja lopullisen käännöksen siirtämiseen vaadittavan MSBuild-skriptin. Kokonaiskuvaus on havainnollistavasti esitetty liitteessä 1.



Kuvio 19. Koostavan käännöskonfiguraation kansiorakenne TeamCity agentilla.

5.2.3 Käännösvaiheet ja Advanced Installer lisäosa TeamCityyn

Käännöskonfiguraation käännösvaiheissa määritellään prosessin aikana suoritettavat vaiheet. Kuvio 20. nähdään käännösvaiheen asetuksista määriteltävä Runner type eli käännösvaiheen tyyppi. Käännösvaihetyypit ovat ennalta määriteltyjä lisäosia, joita TeamCity tarjoaa oletuksena lähes kaikkiin perustarpeisiin. Tyypillisimpiä käännösvaiheen tyyppejä ovat ohjelmakoodin kääntäjät, jotka ovat lähtökohtaisesti samoja, kuin ohjelmistokehittäjien omilla työasemillaan käyttämän kääntäjät.



Kuvio 20. TeamCity käännösvaiheen käännöstyypin konfigurointi.

Tutkimuksen aikana löydettiin vapaan lähdekoodin lisäosa TeamCityyn, jota voidaan käyttää Advanced Installer ohjelmiston ja TeamCityn yhteensovittamiseen. Lisäosan avulla TeamCityn kautta pystytään ajamaan Advanced Installer kännökset niin, että

installer-projektin kääntäminen onnistuu automatisoidusti TeamCityn yhteydessä. Kuvio 21. nähdään koostavaan käännöskonfiguraatioon lisätyt neljä käännösvaihetta, joista jokainen vastaa yhtä artefaktiriippuvuudella ladattua ohjelmistoa.

Build Step	Parameters Description		
CWS .MSI	Advanced Installer Execute: If all previous steps finished successfully	Edit	More
Applications .MSI	Advanced Installer Execute: If all previous steps finished successfully	Edit	More
Build .MSI	Advanced Installer Execute: If all previous steps finished successfully	Edit	More
Deploy to IT-DI565-HKI	MSBuild Build file: DepoymentScripts/TeamCityTestDeploy.xm Targets: DeployAssemblies Execute: If all previous steps finished successfully	Edit	More

Kuvio 21. TeamCity käännösvaiheet.

Integraatiolisäosa havaittiin todella hyödylliseksi, sillä sen kautta pystyttiin asettamaan käännettävien tuotteiden versiot ja muut hyödylliset komennot suoraan Advanced Installerille. Kuvio 22. nähdään käytetyt Advanced Installer komennot SetVersion, UpdatePathVariable ja SetOutput Location.

Type commands:

```
SetVersion %dep.Sts.system.Version%.%build.counter%
UpdatePathVariable -name ROOT -value %teamcity.build.checkoutDir% -valuetype Folder
SetOutputLocation -buildname DefaultBuild -path %teamcity.build.checkoutDir%
```

Advanced Installer edit commands that will be executed against the specified aip file.
Example: SetVersion 1.2.3

Kuvio 22. AdvancedInstaller lisäosan komentosyötteet.

TeamCity tarjoaa konfigurointiin globaaleja muuttujia kaikista olemassa olevista käännöskonfiguraatioista. Tämä tarkoittaa sitä, että esimerkiksi käytettyjen artefaktiriippuvuuksien versiotiedot olivat helposti saatavilla ja syötettävissä SetVersion komennolla Advanded Installerille. Kuvio 23. nähdään artefaktiriippuvuutena käytetyn käännöskonfiguraation parametriasetukset. Prosenttimerkkien väliin kirjoitettu build.vcs.number viittaa TeamCityn tarjoamaan käännösnumeroon eli siihen, kuinka monta kertaa kyseinen käännöskonfiguraatio on ajettu.

System Properties (system.)

Name	Value		
system.Revision	%build.vcs.number%	Edit	Delete
system.Version	3.2.0	Edit	Delete

Kuvio 23. TeamCity käännöskonfiguraation käännösparametrit.

Kuvio 22. oleva SetVersion-komento asettaa Advanced Installerilla käännettävän asennuspaketin ohjelmistoversion. Komennon syöttäminen TeamCityn käännösvaiheen kautta oli todella kätevää, sillä versiotieto voitiin hakea käyttämällä TeamCityn globaaleja käännösparametreja. Kuvio 22. näkyvät prosenttimerkkien väliin määritetty avain dep.sts.system.version viittaa siis Kuvio 23. määriteltyyn artefaktiriippuvuuden parametritietoon.

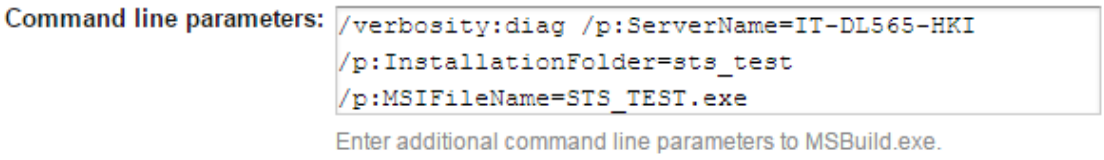
UpdatePathVariable komennolla asetetaan Advanded Installer projektille juurimuuttuja. Kappaleessa 5.1.2 käytiin läpi juurimuuttujan etuja dynaamisuuden osalta. Juurimuuttujaksi asetettiin TeamCityn tarjoama checkoutDir-muuttuja eli aikaisemmin esitellyssä Kuvio 19. näkyvä kansiopolku kohteeseen, johon koostava käännöskonfiguraatio lataa kaikki tarvittavat tiedostot. Muuttujan käyttö mahdollistaa sen, että samaa Advanded Installer käännösprojektia käytettäessä ei jokaisen uuden koostavan käännöskonfiguraation kohdalla tarvitse tehdä manuaalisia muutoksia Advanced Installer projektiin.

Myös SetOutputLocation komentoa käytettiin dynaamisuuden mahdollistamiseen. SetOutputLocation komento kertoo käännösprosessille, mihin käännöksestä muodostuva asennuspaketti luodaan. Myös käännöspaikaksi asetettiin TeamCityn tarjoama checkoutDir-muuttuja, jotta asennuspaketit päätyvät juurihakemistoon. Advanced Installer projekti löytää ROOT-muuttujan avulla käännetyt asennuspaketit, millä on dynaamisuuden kannalta oleellinen merkitys, kun esivaatimussovelluksia yhdistellään samaan päätason asennuspakettiin.

5.2.4 Asennuspaketin siirtäminen testipalvelimelle

Viimeinen vaihe automatisoidussa asennuspaketin kääntämisessä oli sen siirtäminen testipalvelimelle. Liitteessä 2. on kuvattu MSBuild skripti, jonka avulla Advanced Installer käännöksen muodostama lopullinen asennuspaketti siirrettiin testipalvelimelle.7

Kuvio 21. näkyvä viimeinen käännöskonfiguraation käännösvaihe on MSBuild tyyppiseksi määritelty vaihe, joka käyttää edellä mainittua versionhallinnasta ladattua MSBuild skriptitiedostoa. Kuvio 24. nähdään prosessille syötettävät parametrit, joiden avulla määritellään haluttu kohdepalvelin, asennuskansio ja lopullisen siirrettävän asennuspaketin nimi. Siirtovaihe on tarkoitettu sisäisen testauksen helpottamiseen ja lisäksi se mahdollistaa tulevaisuudessa asennusten suorittamisen testipalvelimelle automaattisesti.



Kuvio 24. TeamCity tiedostojen siirtovaiheen MSBuild komennot.

5.3 Toimitusprosessin muutokset

Windows Installer paketteja ei tämän työn puittessa otettu asiakaskohtaiseen testikäyttöön. Advanced Installerin tutkiminen tapahtui kokeilulisenssiä käyttäen, joten sen testausta voitiin suorittaa ainoastaan sisäisesti.

Tämän työn puitteissa on kartoitettu asennuspakettien mahdollisuuksia ja tulevaisuuden haasteita. Asennuspakettien osalta ollaan vaiheessa, jossa Financial Solutions organisaatiotasolla tarvitaan päätös siitä, tullaanko Windows Installer asennuspakettien tutkimista jatkamaan ja viemään kohti tuotantokäyttöä.

Jos asennuspaketti otetaan käyttöön, muuttuu ohjelmistojen toimittamisessa asennuspaketin kokoaminen ja itse asennusvaihe. Asennuspaketin kokoaminen hoidetaan tällä hetkellä manuaalisesti ja Windows Installerin myötä se tapahtuisi automaatti-

sesti. Myös Asennusvaihe hoidetaan tällä hetkellä manuaalisesti siirtämällä binääritiedostot asentajan toimesta palvelimelle. Installer paketin myötä Difs-ohjelmistot asennettaisiin järjestelmällisesti aina samalla tavalla ja ohjelmistot rekisteröitäisiin osaksi Windowsin ohjelmien hallintaa.

Tällä hetkellä muutoshistorian sekä tietokanta- ja konfiguraatiomuutosten teko tapahtuu manuaalisesti. Windows Installer paketti ja sen tarjoama automaatio saattavat tuoda tulevaisuuden mahdollisuuksia kohti kokonaisvaltaisesti hallittua toimitusprosessia.

Hotfix toimitusten eli nopeiden korjaustoimitusten osalta asennuspakettien mahdollisuuksia pitää vielä tutkia. Windows Installer tarjoaa versiopäivitysasennusten lisäksi pienempiä ohjelmistopäivityksiä, joilla saataisiin mahdollisesti korvattua manuaaliset hotfix päivitykset.

6 Lopputulokset

6.1 Opinnäytetyön onnistuminen

Opinnäytetyö toteutettiin niin, että teoriaosuuden tutkiminen ja kirjoittaminen tehtiin ensin. Varsinaisen konkreettisen työn kohdetta ei alussa tiedetty, joten teoriaosuus ei täysin vastaa konkreettisen työn sisältöä ja esimerkiksi Windows Installerin toimintaan ja ominaisuuksiin olisi voitu syventyä paljon tarkemmin. Myös tietokanta- ja konfiguraatiomuutosten hallinnan teoriaan ja työkaluihin olisi ollut hyvä perehtyä paremmin, mutta koska työ tehtiin kronologisesti ja aikaa oli loppua kohden niukasti, ei näiden tutkimiseen jäänyt tarpeeksi aikaa.

Työn aikana opittiin kuitenkin paljon hyödyllistä teoriaa ohjelmistokehitysprosesseista ja niiden kehityskulusta. Tiedon sisäistäminen on herättänyt paljon ideoita päivittäisen työn kehittämisestä ja koen, että se on tuonut mukanaan myös tietynlaista ohjelmistoalan ammattimaista kypsyyttä. En ole täysin tyytyväinen työn lopputulokseen mutta koko opinnäytetyöprosessi on ollut mielestäni niin antoisa, että koen sen olevan itselleni paljon tärkeämpää, kuin työn lopputulos.

Työssä lähdettiin mielestäni jo ideatasolta lähtien tavoittelemaan liian suurta parannusta toimituskäytänteisiin. Lopputulokset olisivat luultavasti olleet paremmat, jos työ olisi toteutettu useista toimitusprosessiin ja työkaluihin liittyvistä kokeiluista ja näiden vertailusta. Nykytilanteen kartoituksen jälkeen lähestymistavaksi valittiin todella nopeasti Windows Installer paketti ja sen rakentamiseen Advanced Installer työkalu. Tämä johtui osittain siitä, että erilaisiin kokeiluihin ei yksinkertaisesti ollut tarpeeksi aikaa.

Teoriaosuuden opiskeluun tuli käytettyä paljon aikaa ja pidän tietämykseni lisääntymistä arvokkaana asiana. Tämä ei kuitenkaan tuo välitöntä lisäarvoa organisaatiolle mutta tiedon ja ymmärryksen lisääntymistä voidaan toisaalta pitää investointina tulevaisuuteen.

6.2 Saavutetut hyödyt

Työn tavoitteena oli tuottaa jokin konkreettinen parannus Financial Solutionsin tuotteiden toimitusprosessiin. Asennuspaketilla on tulevaisuuden mahdollisuuksia mutta tällä hetkellä se ei tuota yritykselle mitään konkreettista tai välitöntä lisäarvoa. Tätä voidaan pitää jokseenkin epäonnistumisena työn tavoitteisiin nähden.

Saavutetut hyödyt organisaation näkökulmasta ovat vielä toistaiseksi abstrakteja. Tärkein hyöty on työn aikana saavutettu kokemus ja tieto Windows Installerin mahdollisuuksista, jota voidaan ehkä tulevaisuudessa soveltaa Difs-tuotteiden toimittamiseen. Hyödyllisenä voidaan pitää myös nykytilanteen kartoituksessa opittuja kehityskohteita ja ideoita siitä, mihin suuntaan asioita olisi hyvä kehittää.

Lähteet

Advanced Installer 2017. Viitattu 21.4.2017 http://www.advancedinstaller.com/

Atlassian 2017. What is version control? Viitattu 25.4.2017 https://www.atlassian.com/git/tutorials/what-is-version-control

Awad, M. A. 2005. A Comparison between Agile and Traditional software development methodologies. School of Computer Science and software Engineering, The University of Western Australia. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.464.6090&rep=rep1&type=pdf

Basil, V. R. & Turner, A. J. 1975. Iterative enhancement: A practical technique for software development. IEEE Transactions on Software Engineering, se-1, 4, 390-396 http://www-lb.cs.umd.edu/~basili/publications/journals/J04.pdf

Boehm, B. 2002. Get Ready for Agile methods with care. University of Southern California. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.224.6128&rep=rep1&type=pdf

Boehm, B. Biffl, S., Aurum, A., Erdogmus, H. & Grünbacher, P. 2006. Value-based software engineering. Berlin: Springer-Verlag Berlin Heidelberg http://www.springer.com/%3FSGWID%3D4-102-45-160832-p60398090

Chen, L. 2015: Continous Delivery - Huge benefits, but challenges too http://www.academia.edu/download/36972245/cd_huge_benefits_but_challenges_too.pdf

Cockburn, A. & Highsmith, J. 2016. The scrumban revolution. USA: Pearson Education https://books.google.fi/books?hl=fi&lr=&id=Y6oKCgAAQBAJ&oi=fnd&pg=PP16&dq=scrumban&ots=k_OBgfE6Bs&sig=ue4fDq9romf4rqjPbnSual82kKQ&redir_esc=y#v=onepage&q&f=false

Continuous Deliver with Docker, Viitattu 15.3.2017 https://xebia.github.io/cd-with-docker

Crucible 2017. Collaborative code review. Viitattu 25.4.2017 https://www.atlassian.com/software/crucible

Digia yrityksenä. 2017. Viitattu 9.4.2017 http://digia.com/yritys/

Dingsøyr, T. & Lassenius, C. 2016. Emerging themes in agile software development: Introduction to the special section on continuous value delivery. Information and Software Technology, p. 77, Syyskuu 2016, s. 56–60 http://www.sciencedirect.com/science/article/pii/S0950584916300829

Forsgren, N. & Humble, J. (2016). "The Role of Continuous Delivery in IT and Organizational Performance." In the Proceedings of the Western Decision Sciences Institute (WDSI) 2016, Las Vegas, NV. https://poseidon01.ssrn.com/delivery.php?ID=308111088105096097086007013106108007054027040080070045096086031073000071126122013002126026116029 04

909600007601708108011811010301603603604803508109811408102208606506802202307500211612510310512408800611202909108101209102407111812008311407211808912709611 5&EXT=pdf

Fowler, M. & Highsmith, J. 2001. The Agile Manifesto https://www.agilealliance.org/ http://andrey.hristov.com/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf

Haikala, I. & Mikkonen, T. 2011. Ohjelmistotuotannon käytännöt. Hämeenlinna: Kariston Kirjapaino, uud. p. 12.

Highsmith, J. 2002. Agile software development ecosystems. Indianapolis: Pearson Educations http://digilib.stmik-banjarbaru.ac.id/data/23.%20Addison%20Wesley%20Series/Addison.Wesley.Agile.Software.Development.Ecosystems.Apr.200.pdf

Humble, J. & David, F. 2010. Continuous Delivery - Reliable software releases through Build, Test and Deployment Automation http://www.johnchukwuma.com/training/ContinuousDelivery.pdf

Humble, J. 2016. Blogikirjoitus jatkuvan toimittamisen periaatteista. Viitattu 10.3.2017 https://continuousdelivery.com/principles/

Indigo Rose, 2017. Software Deployment Tools for Windows Developers. Viitattu 21.4.2017 https://www.indigorose.com/setup-factory/

Install Shield, 2017. Viitattu 21.4.2017 https://www.flexerasoftware.com/producer/products/software-installation/installshield-software-installer/

Kim, G., Humble, J., Debois, P. & Willis, J. 2016. The Devops: Handbook. IT Revolution Press http://images.itrevolution.com/documents/DevOps_Handbook_Intro_Part1_Part2.pdf

Küng, S., Onken, L. & Large, S. 2015. TortoiseSVN Subversion-käyttöliittymä Windows-ympäristöön. Viitattu 10.3.2016 https://tortoisesvn.net/support.html

Loukides & Mike 2012: What is DevOps? Viitattu 12.3.2017 http://radar.oreilly.com/2012/06/what-is-devops.html

Macchi, D. n.d. What is Code Review?. Viitattu 25.4.2017 http://thinkapps.com/blog/development/what-is-code-review/

Moniruzzaman, A. B. M & Hossain, A. S. N.d. Comparative Study on Agile software development methodologies https://arxiv.org/pdf/1307.3356.pdf

Octopus Deploy, 2017. Automated deployment for .NET, viitattu 21.4.2017 https://octopus.com/

Poppendieck, M. & Poppendieck, T. 2003. Lean Software Development - An Agile Toolkit. USA: Addison-Wesley p. 14

Rodríguez, P. 2013. Combining Lean thinking and Agile Software Development. How do software-intensive companies use them in practice? Väitöskirja. University of Oulu Graduate School; University of Oulu, Faculty of Science, Department of

Information Processing Science Acta Univ. Oul. A 618, 2013 http://jultika.oulu.fi/files/isbn9789526203324.pdf

Royce, W. 1970. Managing the development of large software systems. The Institute of Electrical and Electronic Engineers http://www-scf.usc.edu/~csci201/lectures/Lecture11/royce1970.pdf

Spotify 2017. Viitattu 30.4.2017 https://www.spotify.com/fi/download/windows/

State Of Agile Survey 2016. Version One:n julkaisema tutkimus 2016. Viitattu 5.2.2017 https://www.versionone.com/ http://www.agile247.pl/wp-content/uploads/2016/04/VersionOne-10th-Annual-State-of-Agile-Report.pdf

TeamCity 2017. Viitattu 25.4.2017 https://www.jetbrains.com/teamcity/

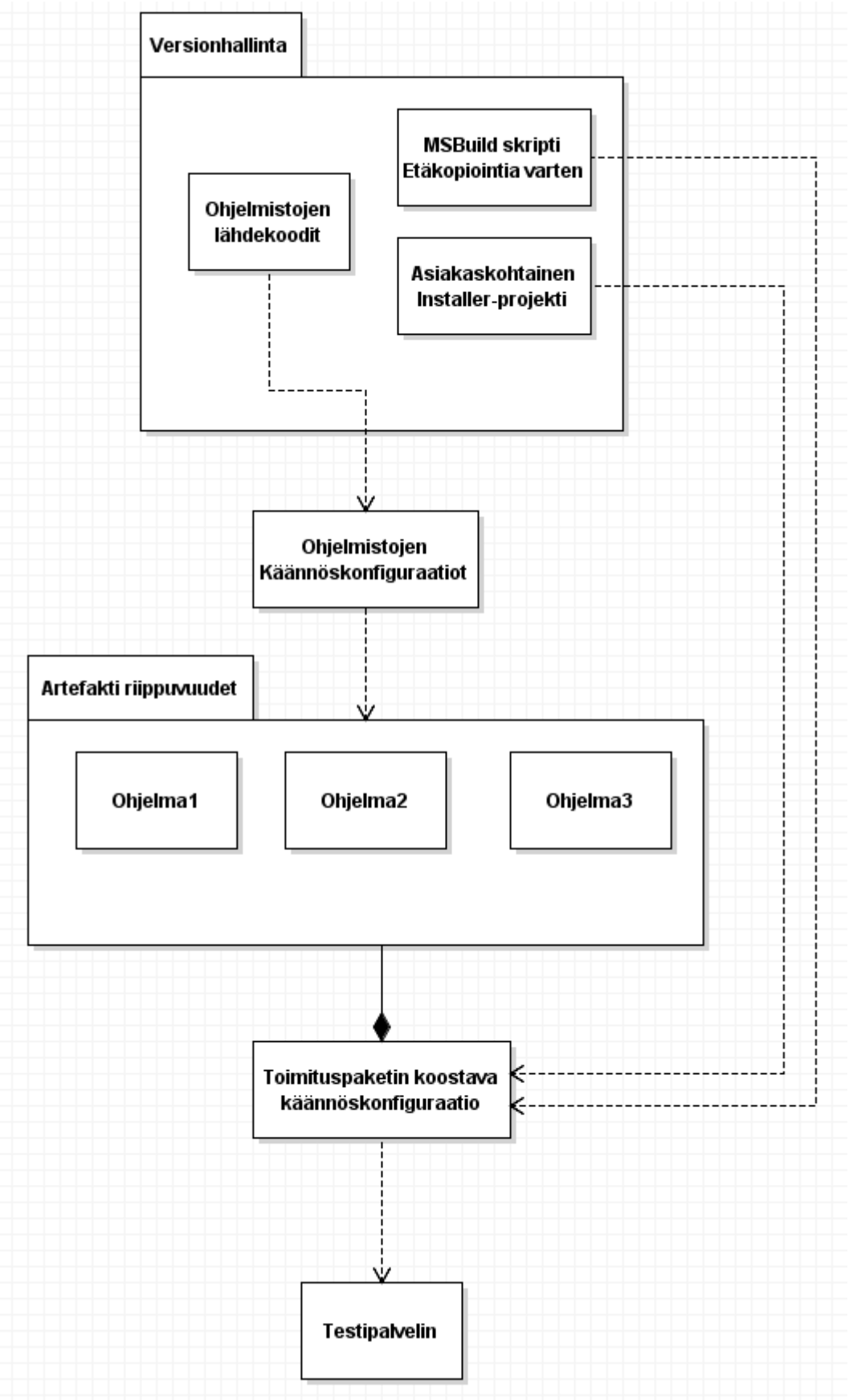
Windows Installer 2017. Viitattu 25.4.2017 https://msdn.microsoft.com/en-us/library/cc185688(VS.85).aspx

Wix Toolset, 2017. Viitattu 21.4.2017 http://wixtoolset.org/

Womack, P. J. & Jones, D. T. 1996. Lean thinking - Banish Waste and Create Wealth in Your Corporation. New York: A Division Of Simon & Chuster https://books.google.fi/books?hl=fi&lr=&id=2eWHaAyiNrgC&oi=fnd&pg=PA13&dq=Lean+thinking&ots=2LQ5oq7jMp&sig=DtS4gfWtFEKSyMprqmO-9PevGAw&redir_esc=y#v=onepage&q&f=false

Liitteet

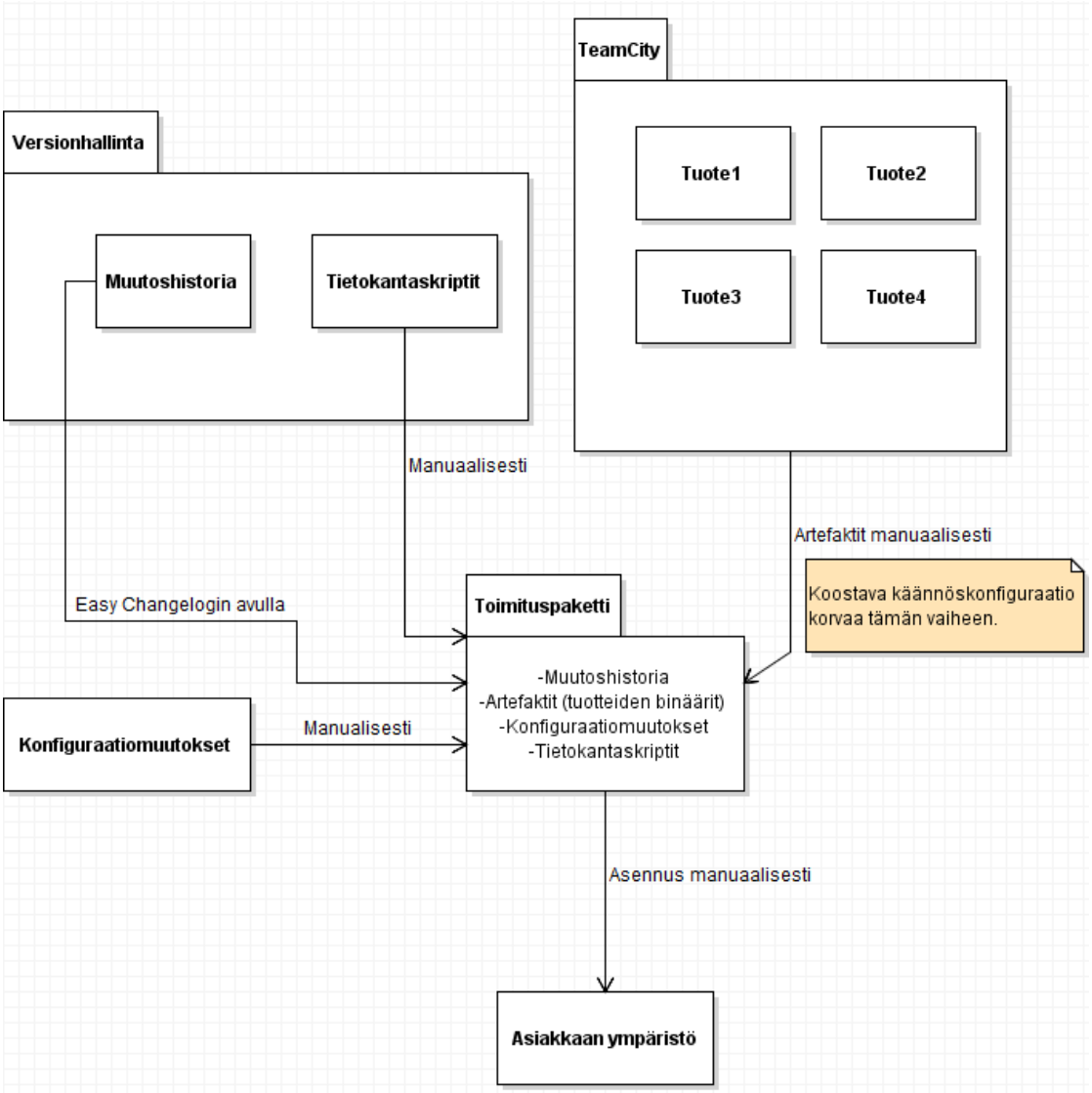
Liite 1. TeamCity toimituspaketin koostamisen kokonaiskuvaus



Liite 2. MSBuild tiedostojen kopioiminen etäpalvelimelle

```
<Project DefaultTargets="Run" xmlns="http://schemas.microsoft.com/developer/msbuild/2003">

 <PropertyGroup>
  <RootFolder>$(MSBuildProjectDirectory)\..</RootFolder>
  <ExtensionTasksPath Condition="'$(ExtensionTasksPath)' == ''">.\ExtensionTasks\</Exten-
sionTasksPath>
  <InstallationFolder></InstallationFolder>
  <ServerName></ServerName>
  <Destination>\\$(ServerName)\$(InstallationFolder)\</Destination>
  <DeploymentFolder></DeploymentFolder>
  <MSIFileName></MSIFileName>
 </PropertyGroup>

 <Import Project="$(ExtensionTasksPath)MSBuild.ExtensionPack.tasks"/>

 <Target Name="Run">
  <CallTarget Targets="DeployAssemblies" />
 </Target>

 <Target Name="DeployAssemblies" >
  <MSBuild.ExtensionPack.FileSystem.RoboCopy Source="$(RootFolder)\" Destination="$(Destina-
tion)\" Files="$(MSIFileName)" Options="/E /V /r:2 /w:1" LogToConsole="True"/>
 </Target>

</Project>
```

Liite 3.


TeamCity
Versionhallinta
Tuote1
Tuote2
Tuote3
Tuote4
Muutoshistoria
Tietokantaskriptit
Manuaalisesti
Artefaktit manuaalisesti
Koostava käännöskonfiguraatio korvaa tämän vaiheen.
Easy Changelogin avulla
Toimituspaketti
-Muutoshistoria
-Artefaktit (tuotteiden binäärit)
-Konfiguraatiomuutokset
-Tietokantaskriptit
Konfiguraatiomuutokset
Manualisesti
Asennus manuaalisesti
Asiakkaan ympäristö