TAMPEREEN TEKNILLINEN YLIOPISTO

Kärpänoja, Pauli

Jatkuva toimittaminen asiakasprojektissa

Diplomityö

Tarkastaja: Prof. Tommi Mikkonen
Tarkastaja ja aihe hyväksytty
Tieto- ja sähkötekniikan
tiedekuntaneuvoston kokouksessa
5. toukokuuta 2014

TIIVISTELMÄ

TAMPEREEN TEKNILLINEN YLIOPISTO
Tietotekniikan koulutusohjelma
Kärpänoja, Pauli: Jatkuva toimittaminen asiakasprojektissa
Diplomityö, 57 sivua
Joulukuu 2015
Pääaine: Ohjelmistotuotanto
Tarkastajat: Prof. Tommi Mikkonen
Avainsanat: Jatkuva toimittaminen, Jatkuva integraatio, Automatisointi, Automatisoitu asennus, Asennusputki, Testiautomaatio

Ohjelmistotuotannossa uuden ohjelmistoversion julkaisu sekä tuotantoasennus on usein työläs ja arvaamaton vaihe, mihin on varattava sekä aikaa että henkilöresursseja. Tuotantoasennuksia tehdään harvoin ja niistä seuraa usein odottamattomia ongelmia. Lisäksi uusien toiminnallisuuksien ja korjausten saaminen tuotantoon voi kestää kuukausia. Myös manuaaliseen testaukseen joudutaan usein panostamaan, kun halutaan ehkäistä mahdollisia tuotantoon eksyviä virheitä.

Tässä diplomityössä perehdytään jatkuvan toimittamisen käytäntöihin ja tutkitaan niiden vaikutuksia ohjelmistotuotantoon. Jatkuvalla toimittamisella tarkoitetaan sitä, että järjestelmästä on asennettavissa testattu uusi versio loppukäyttäjille lähes jokaisen ohjelmakoodimuutoksen jälkeen. Jatkuva toimittaminen pohjautuu ajatukseen, että kaikki ohjelmiston kääntämiseen, integraatioon, testaukseen, palvelinten konfiguraatioon ja ohjelmistojen asentamiseen liittyvät toiminnot on oltava automatisoitu. Jatkuvan toimittamisen selkärankana on asennusputki, jossa muutokset virtaavat kontrolloitujen ja automatisoitujen vaiheiden läpi tuotantoon asti.

Tutkimuksen tulokset perustuvat kolmeen Solita Oy:llä toteutettun asiakasprojektin ohjelmistokehittäjien haastatteluihin ja kokemuksiin projekteista, joissa jatkuvan toimittamisen käytäntöjä on pyritty noudattamaan. Haastattelut toteutettiin puolistrukturoituina yksilöhaastatteluina.

Vaikka tutkimustulokset perustuvat vain kolmen ohjelmistokehittäjän henkilökohtaisiin kokemuksiin, oli jatkuvan toimittamisen käytäntöjen hyödyt kiistatta todistettavissa. Tuotantoasennuksen käynnistäminen oli helppoa, ja asennus epäonnistui äärimmäisen harvoin tai ei koskaan. Jatkuvan integraation sekä kattavan testiautomaation ansiosta kaikki projektit olivat päivittäin tilassa, jossa tuoreen tuotantoasennuksen voisi tehdä. Koska tuotantoasennus on nopea sekä helppo käynnistää ja luotto ohjelmiston laatuun on korkea, voidaan virheisiin ja muutostarpeisiin reagoida jopa tuntien viiveellä ongelmitta. Huolimatta siitä, että käytäntöjen noudattamiseen erityisesti testiautomaation osalta kului aikaa, piti jokainen haastateltava panostusta kannattavana. Kaiken sijoituksen koettiin maksavan itsensä takaisin projektin aikana vähentämällä korjauksiin ja selvittelyyn kuluvaa aikaa, sekä erityisesti tuomaan luottoa siitä, että järjestelmän toiminnallisuus on eheä.

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Kärpänoja, Pauli: Continuous Delivery in customer driven software project
Master of Science Thesis, 57 pages
December 2015
Major: Software Engineering
Examiners: Prof. Tommi Mikkonen
Keywords: Software, Automation, Continuous Integration, Continuous Delivery, Deployment Pipeline

Releasing new software is generally a fragile and painful operations, which requires work and resources. Deployments to production are rare and often imply some unpredictable problems. Additional time for new functionalities and modifications may take even months to make into production. Manual testing is also often required to prevent bugs and other issues from slipping into production.

This Master's Thesis digs into the practices of Continuous Delivery and their impact on software development. In Continuous Delivery, new software should always be releasable and deployable into production even if under development. Continuous delivery relies on a Deployment Pipeline, which automates all the phases from building, integrating, testing, configuring and deploying the software to every environment all the way into production.

The conclusions in this thesis are based from three case studies in Solita Oy. These cases are software projects that have adapted Continuous Delivery practices. The software developers of these projects were interviewed to examine the experiences and benefits of these practices.

Although the conclusions rest on interviews of just three developers, the benefits of adapting the practices were verifiable. Deploying software into production was easy and problems rarely or never occurred. As a result of Continuous Integration and Test Automation, all the projects were in a state on a daily basis, where production deployment would be viable. When deployment to production is fast and easy, and the trust to the quality of the software is high, the development team was prepared to deploy to production in a matter of hours if needed. Despite the required work to create and maintain especially test automation, all interviewees considered the investment profitable. Test practices not only decreased the amount of work needed to spend on modifications but also gave the development team more trust in the quality of the product.

ALKUSANAT

Tämä on diplomityö Tampereen teknillisen yliopiston ohjelmistotekniikan laitokselle. Tutkimuksessa esitellään jatkuvan toimittamisen käytäntöjä sekä Solita Oy:n ohjelmistokehittäjien kokemuksia jatkuvan toimittamisen adaptoimisesta ohjelmistoprojekteihin.

Tunnen itse muutaman vuoden ohjelmistokehittäjänä työskennelleenä jatkuvaa intohimoa nykyisten toimintamallien kehittämiseen sekä laadun ja tehokkuuden parantamiseen. Tästä syystä olen ajautunut puskemaan jatkuvan toimittamisen käytäntöjä läpi ohjelmistoprojekteissa ja perehtymään näistä saatuihin kokemuksiin myös diplomityön muodossa.

Vaikka diplomityössä kului aloittamisesta valmistumiseen yli vuosi, koen että tässä tapauksessa vaikutus lopputulokseen oli vain positiivinen. Olen tänä aikana töissä laajentanut omaa tietämystäni aiheesta, ja toivon, että se näkyy myös tässä tutkimuksessa. Mikäli tämä diplomityö itsessään päätyy vain TTY:n kirjaston hyllyyn, olen ainakin itse oppinut haastatteluja pitäessä, niitä analysoidessa ja tätä kirjoittaessa, todella paljon.

Haluan kiittää diplomityön tarkastaja professori Tommi Mikkosta innokkaasta asenteesta ja suhtautumisesta kirjoitusprosessiini, sekä Marko Leppästä avusta haastattelututkimuksen tekemisessä. Haluan myös antaa erityismaininnan Esko Luontolalle, joka on aikanaan kylvänyt minuun siemenen, jonka vuoksi olen ylipäätään kiinnostunut testauksesta, automatisoinnista ja jatkuvasta toimittamisesta.

Kiitoksen ansaitsevat myös kollegani Antti Virtanen sekä Timo Lehtonen, jotka ovat toimineet Solitan puolesta ohjaamassa tätä työtä oikeaan suuntaan. Lisäksi haluan kiittää suomalaisia pienpanimoita, jotka ovat useasti sekä antaneet minulle voimaa että palkinneet minua tämän kirjoitusprojektin aikana.

Helsingissä, 2015

Kärpänoja, Pauli

SISÄLLYS

1. Johdanto 1
2. Jatkuva toimittaminen 3
2.1. Mitä jatkuva toimittaminen on 3
2.2. Jatkuvan toimittamisen hyödyt 4
2.3. Tekniset vaatimukset 6
2.4. Jatkuva käyttöönotto 6
3. Vaadittavia käytäntöjä 8
3.1. Versionhallinta 8
3.2. Konfiguraatioiden hallinta 9
3.3. Jatkuva integraatio 12
3.4. Automaattinen testaus 13
3.4.1. Testityyppejä 14
3.4.2. Testien jakaminen asennusputken vaiheisiin 15
3.5. Julkaisuehdokas ja automatisoitu asennus 16
3.6. Asennuksen porrastettu ylennys 17
3.7. Komponenttien ja kirjastojen hallinta 19
3.8. Asennusputken yhteenveto 20
3.9. Tietokantamuutosten hallinta 21
3.10. Ohjelmointikäytännöt ja haarojen pahuus 22
4. Työkaluja ja teknologioita 24
4.1. Versionhallinta 24
4.2. Palvelinten ja infrastruktuurin hallinta 25
4.3. Jatkuvan integraation hallinta 28
4.4. Tietokantamuutosten hallinta 29
5. Jatkuva toimittamisen tutkimustulokset projekteissa 31
5.1. Projekti 1: Lupapiste.fi 32
5.1.1. Projektin asennusputken esittely 32
5.1.2. Haastattelun tulokset 34
5.1.3. Analyysi 36

5.2. Projekti 2: YleX.fi 38
5.2.1. Projektin asennusputken esittely 38
5.2.2. Haastattelun tulokset 40
5.2.3. Analyysi 43
5.3. Projekti 3: Työajankirjausjärjestelmä 45
5.3.1. Projektin asennusputken esittely 46
5.3.2. Käytännöt ja tulokset 47
5.3.3. Analyysi 50
5.4. Analyysien yhteenveto 51
6. Yhteenveto 56
Lähteet 58

LYHENTEET JA TERMIT

Asennuksen porrastettu ylennys	Ohjelmistoversion tai artefaktin ylentäminen asennusputkessa eteenpäin. [4, s.108-109, s.253-257]
Asennusputki	Perättäiset automatisoidut vaiheet, jotka ohjelmakoodimuutoksesta seuraa tuotantoasennukseen asti. [4, s. 106-122]
Artefakti	Tuote, joka on luotu toimitettavaksi eteenpäin. [4, s. 111]
Binääri	Ohjelmakoodista käännetty ohjelmisto tai sen osa.
Haara (versionhallinnassa)	Tiedostojen muutoshistoria on ajan suhteen lineaarinen. Historiasta voi kuitenkin eriyttää erillisiä haaroja. [4, s. 388]
Hajauttaminen (ohjelmiston)	Saman ohjelmiston suorittaminen useammalla palvelimella niin, että ne toimivat kuitenkin yhdessä. Mahdollistaa laskentatehon sekä vikasietoisuuden parantamisen.
Jatkuva integraatio	Ohjelmakoodin kääntäminen ja riippuvuuksien integroiminen automatisoidusti ja säännöllisesti. [4, s. 105]
Jatkuva toimittaminen	Ohjelmiston pitäminen tilassa, jossa tuotantoasennus voidaan tehdä automatisoidusti ja usein. [10, s.121]
Julkaisuehdokas	Ohjelmiston binääri tai artefakti, joka on mahdollista ylentää asennusputkessa. [4, s. 22-24, s.108-109]
Kommitointi versionhallintaan	Yksittäinen muutoskokonaisuus tiedostoihin, mikä tallennetaan versionhallintaan.
Ominaisuuskytkin	Konfiguroitava asetus, jolla jonkun järjestelmän ominaisuuden saa kytkettyä päälle tai pois. [10, s.124]
Skripti	Sarja komentoja, jotka voidaan suorittaa ohjelmallisesti.
Savutesti	Testi, jolla testataan yksittäinen perustoiminta. [4, s.61]
Tietokantamigraatio	Muutos, joka on suoritettava tietokantaan esimerkiksi tietokantaskeeman muuttuessa. [4, s.327-334]
Versionhallinta	Tiedostosäilö, joka ylläpitää tiedostoja koskevia muutoksia, näiden aikaleimoja sekä muutosten tekijät.
Väliohjelmisto	Ohjelmisto, jolla voidaan suorittaa tiettyjä tehtäviä toisen ohjelman sisällä.

1. JOHDANTO

Ohjelmistotekniikka on viimeisten vuosikymmenien aikana muuttunut oleelliseksi osaksi nyky-yhteiskunnan teknistä kehitystä. Ohjelmistotuotanto on haastava ja hauras prosessi, jota on tutkittu sen syntymisestä asti erilaisin tuloksin. Erikokoiset ja -tyyppiset ohjelmistot vaativat erilaisia suunnittelu- ja tuotantotapoja, joiden valitseminen ja prosessiin sovittaminen ei aina onnistu esitutkimuksesta huolimatta.

Ongelmia ja haasteita kohdataan päivittäin tuotanto- ja jatkokehitysprosessin jokaisessa vaiheessa. Ohjelmakoodia kirjoittaessa kehittäjät tekevät väkisinkin virheitä, joiden selvittäminen ja korjaaminen sekä korjausten asentaminen voi olla hidasta, työlästä ja riskialtista. Osa käyttäjien tarpeista tulee ilmi vasta projektin myöhemmissä vaiheissa ja näihin täytyy pystyä reagoimaan mahdollisimman nopeasti. Palvelinympäristöt ovat monimutkaisia ja niiden epäjärjestelmällisestä ylläpidosta seuraa usein ongelmia kaikkine tuotanto-, testi- ja kehitysympäristöineen. Kaikki ongelmat ja niiden ratkomiseen kuluva aika aiheuttaa sekä asiakkaalle että toimittajalle kustannuksia. Lisäksi psykososiaaliset riskitekijät aiheuttavat stressiä, joiden hallinta ja ehkäisy näkyy työpaikoilla muun muassa terveinä ja motivoituneina työntekijöinä, sairauspoissaolojen vähenemisenä ja lisääntyneenä työhyvinvointina ja työtyytyväisyytenä [5].

Tässä diplomityössä perehdytään jatkuvan toimittamisen käytäntöihin ja tutkitaan niiden vaikutuksia ohjelmistotuotantoon. Jatkuva ohjelmiston toimittaminen sekä käytännöt, jotka joudutaan adaptoimaan, jotta jatkuvaa toimittamista voidaan tehdä, voivat tuoda ratkaisuja moniin tyypillisiin ohjelmistoprojektien ongelmiin ja haasteisiin. Tutkimuksen tulokset pohjautuvat Solita Oy:n toteuttamiin web-pohjaisiin ohjelmistoprojekteihin, jotka on tehty yhdessä räätälityönä asiakkaan kanssa. Tulokset perustuvat projekteissa olevien Solita Oy:n ohjelmistokehittäjien haastatteluihin ja kokemuksiin. Esiteltävä jatkuvan toimittamisen teoria perustuu pääsääntöisesti Jez Humblen sekä David Farleyn kirjaan Continuous Delivery [4].

Tutkimuksen alussa, luvussa 2, esitellään jatkuvan toimittamisen perusteet sekä odo-

tettavissa olevia hyötyjä, joita jatkuvasta toimittamisesta saadaan. Luvussa 3 esitellään käytännöt, joita noudattamalla jatkuva toimittaminen tulee mahdolliseksi. Luvussa 4 tuodaan esille teknologioita, työkaluja ja ohjelmistoja, jotka helpottavat käytäntöjen toteuttamista. Luvussa 5 esitellään tutkimuksen pohjana olevat ohjelmistoprojektit, haastattelut sekä tutkimustulokset. Luku 6 sisältää tutkimuksen yhteenvedon.

2. JATKUVA TOIMITTAMINEN

Jatkuvalla toimittamisella (Continuous Delivery) tarkoitetaan sitä, että järjestelmästä on asennettavissa testattu uusi versio loppukäyttäjille aina muutosten tekemisen jälkeen. Itse tuotantoasennus voi jossain projektissa olla viikoittaista, toisinaan asennuksia voidaan tehdä monta kertaa päivässä projektin kehitysmallista ja -vaiheesta riippuen. Jatkuva toimittaminen ei kuitenkaan ole pelkkä toimitusmetologia, vaan kokonaan uusi paradigma ohjemistoliiketoiminnalle [4, s. 417].

2.1. Mitä jatkuva toimittaminen on

Jatkuvassa toimittamisessa itse tuotantoasennusten tiheys ei välttämättä ole avainasemassa, vaan tärkeää on mahdollisuus tuotantoasennukseen aina haluttaessa [10, s.121]. Jatkuva toimittaminen ei ole pelkästään erilaisten työkalujen käyttämistä automatisoidun asennuksen toteuttamiseksi, vaan vaatii yhteistyötä kaikilta, jotka liittyvät järjestelmän toteutukseen ja sen toimittamiseen [4, s. 417]. Pelkkä ohjelman kääntäminen, asennuspakettien toimittaminen ja palvelinten uudelleenkäynnistäminen on helppo skriptata toimimaan automaattisesti. Jatkuva toimittaminen asettaa kuitenkin paljon vaatimuksia laadulle, sekä pakottaa tiettyjä käytäntöjä ja toimintatapoja koko kehitys- ja toimitusprosessiin.

Koska ohjelmistosta on oltava toimiva versio asennettavissa loppukäyttäjille säännöllisesti, on luonnollisesti ehdotonta, että järjestelmä toimii aina halutulla tavalla, vastaa asiakkaan tarpeita ja toimii ongelmitta siinä ympäristössä mihin se asennetaan. Harva ohjelmistoprojekti toteuttaa nämä kriteerit aina, ja ongelmat tulevat usein ilmi vasta tuotantoasennuksen yhteydessä tai sen jälkeen. Jos ohjelmistosta toimitetaan uusi versio jatkuvasti, on järjestelmän oltava jatkuvasti myös sellaisessa tilassa, että se ylipäätään asentuu, toimii ja täyttää kaikki laatukriteerit. Näihin haasteisiin voidaan vastata muun muassa konfiguraatioiden hallinnalla, harkitulla ja kattavalla testauksella, jatkuvalla integraatiolla ja näiden automatisoinnilla [4, s. 418].

2.2. Jatkuvan toimittamisen hyödyt

90-luvun lopulla Martin Fowler oli tapaamassa Kent Beckiä tämän työskennellessä vakuutusyhtiössä Sveitsissä. Fowlerin mukaan yksi mielenkiintoisimpia aspekteja oli Beckin tiimin käytäntö asentaa järjestelmästä uusi versio tuotantoon joka yö. Käytännöstä johtuen tiimi koki olevansa lähempänä asiakasta ja loppukäyttäjiä, kirjoitettu ohjelmakoodi ei seisonut odottamassa käyttöä, ja uusiin tarpeisiin pystyttiin vastaamaan nopeasti. Jatkuvan toimittamisen käytäntö ei ole siis mikään uusi toimintamalli, vaan ollut osa joidenkin ohjelmistoyritysten päivittäistä toimintaa jo ainakin toista kymmentä vuotta. Käytäntö on kuitenkin jäänyt yleistymättä laajemmin. [4, s. xxi]

Useiden projektien toimintamallina on määritellä seuraavaksi toteutettavat uudet ominaisuudet ja muutokset, sekä sopia asennus etukäteen johonkin tiettyyn päivään. Tuotantoasennus voi olla sovittu tehtäväksi esimerkiksi muutaman kuukauden välein. Kyseisessä tapauksessa valmiiden ominaisuuksien käyttöönotto odottaa pahimmillaan useita kuukausia. Isommilla kokonaisuuksilla on taas aikataulupaine, jonka vuoksi ne voidaan joutua asentamaan tuotantoon keskeneräisinä ja viimeistelemättöminä. Tuotannossa ilmenneet pienemmät ongelmat sen sijaan joutuvat odottamaan pitkiä aikoja ennen kuin korjaus saadaan tehtyä. [10, s.122]

Mikäli järjestelmästä on mahdollisuus julkaista uusi versio tuotantoon usein, voidaan asiakkaan tai loppukäyttäjän toiveisiin reagoida nopeasti. Tämä voi vaikuttaa positiivisesti myös asiakastyytyväisyyteen. Kehittäjät voivat kokea olevansa lähempänä asiakkaan lisäksi myös muita sidosryhmiä, kun palautesyklit ja reaktioajat ovat nopeita. [9, s.67]

Jatkuvan toimittamisen käytäntöjen avulla uudet toiminnallisuudet saadaan heti valmistumisen jälkeen asiakkaalle testattavaksi. Ne voidaan asentaa myös nopeasti tuotantoon tuottamaan loppukäyttäjälle lisäarvoa, millä taas on suora positiivinen vaikutus asiakkaan liiketoimintaan. Tämä toimintamalli ohjaa myös tiimiä keskittymään niihin ominaisuuksiin, jotka ovat asiakkaan liiketoiminnan kannalta oleellisia. Mikäli uudet toiminnallisuudet otetaan käyttöön nopeasti, saadaan niistä myös nopeasti palautetta. Palautteen perusteella voidaan tehdä päätöksiä siitä, kannattaako ominaisuuteen käyttää vielä lisää työtä vai siirtyä oleellisimpiin kehityskohteisiin. [2, s.52]

Useimmiten ohjelmistokehityksessä on tarve pystyä asentamaan ohjelmistosta uusi versio nopeasti tuotantoon. Vaikka jotkin järjestelmät ovat sen laatuisia, ettei ole perustel-

tua toimittaa uutta versiota vain uusien ominaisuuksien puolesta tiheästi, joudutaan tuotantoon asentamaan joka tapauksessa virhekorjauksia nykyiseen versioon. Jatkuvan toimittamisen metodit takaavat valtavan positiivisen vaikutuksen päivitysten toimittamiseen luotettavasti ja nopeasti [4, s. 131]. Kun tuotannossa ilmenee virheitä, voidaan jatkuvassa toimittamisessa päästä myös tilanteeseen, jossa on luontevampaa tehdä korjaus ja asentaa se, kuin palauttaa aiempi versio [10, s.126].

Mikäli tuotantoasennuksia tehdään usein, on uusien muutosten määrä asennuksen yhteydessä suhteellisen pieni. Tällöin virhemahdollisuuksia on myös kerralla vähemmän ja niiden paikallistaminen on helpompaa [2, s.53]. Kun tuotantoasennus on helppoa, turvallista ja nopeaa, saadaan virhekorjaukset asennettua tuotantoon pian niiden löytymisen jälkeen. Säännöllinen tuotantoasennus tekee asennuksesta myös arkipäiväistä, vähentää siinä ilmeneviä ongelmia ja vähentää stressiä [10, s.122].

Vaikka asennuksen automatisoinnin toimintamalli esitellään tarkemmin tämän tutkimuksen myöhemmässä vaiheessa, on hyvä ymmärtää jo tässä vaiheessa päämäärä, johon jatkuvassa toimittamisessa pyritään. Asennusten täytyisi olla täysin automatisoituja niin, että uuden version kääntäminen, testaaminen ja asentaminen mihin tahansa ympäristöön vaatii kaksi ihmisen suorittamaa toimintoa: Asennettavan version ja kohdeympäristön valinta sekä asennuspainikkeen painaminen [4, s. 5-6]. Tämä voi kuulostaa kaukaiselta, mutta oikeilla toimintamalleilla sekä teknologioilla täysin toteutettavissa, ja monissa projekteissa tämä on arkipäivää.

Pelkästään siitä, että uuden version asennus tuotantoon on kenen tahansa mahdollista tehdä milloin tahansa, saadaan muun muassa seuraavia hyötyjä kehitystiimin kesken [4, s. 6]:

- Automatisoitu asennus on aina toistettavissa helposti ja nopeasti, mikä vähentää asennuksessa ilmenneiden virheiden selvittelyyn kuluvaa aikaa.
- Manuaalinen tuotantoasennus täytyy olla hyvin dokumentoitu ja dokumentaatiota ylläpitää. Automatisoitu asennus on dokumentoitu asennusskripteihin ja aina ajan tasalla. Manuaalisessa dokumentaatiossa täytyy myös pyrkiä huomioimaan lukijan tekninen pätevyys.
- Manuaalinen tuotantoasennus on yleensä sen alueen asiantuntijan vastuulla ja täten

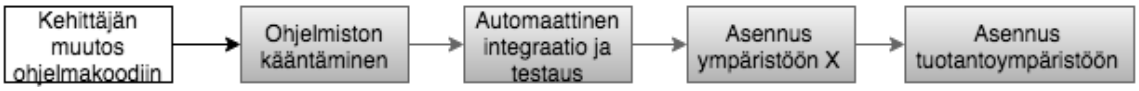
riippuvainen tämän henkilön läsnäolosta. Automatisoidun tuotantoasennuksen voi käynnistää kuka tahansa.

- Ainoa tapa testata asennusta on sen tekeminen. Jos se on manuaalista ja vie aikaa, on se myös kallista.
- Manuaaliset tuotantoasennukset ovat tylsiä ja vievät aikaa. Asiantuntijoiden ei pitäisi kuluttaa pitkiä aikoja yksinkertaisten asioiden tekemiseen.
- Automaattinen asennus on täysin auditoitavissa ja asennuskriptiä on seurattu aina tarkasti, toisin kuin mahdollista manuaalisen asennuksen dokumentaatiota.

Jotta jatkuvaa toimittamista voidaan luontevasti ja luotettavasti tehdä, on kuitenkin koko kehitysprosessia mahdollisesti mullistettava aina ohjelmakoodin kommitointirutiineista ja testauskäytännöistä asennusprosessiin asti. Näiden käytäntöjen tavoitteena on hallita ja vähentää riskejä, jotka liittyvät ohjelmistoversioiden julkaisuun ja asentamiseen [4, s. 418].

2.3. Tekniset vaatimukset

Jatkuvassa toimittamisessa koodimuutosten voidaan ajatella virtaavan asennusputken (Deployment pipeline) läpi aina tuotantoon, loppukäyttäjille asti. Putki sisältää kaikki vaiheet, joissa ohjelmistokoodi käännetään tarvittaessa, testataan ja asennetaan joko automaattisesti tai ihmisten käynnistämillä toiminnoilla. [4, s. 105-112].



Kuva 2.1: Yksinkertainen asennusputki.

Kuvan 2.1 hahmotelma asennusputkesta esittää, miten muutokset ohjelmakoodiin valuvat putken läpi tuotantoon. Automatisoidun asennusputken vaiheet ja niiden toteuttamiseen liittyvät tekniset yksityiskohdat ja käytännöt esitellään seuraavassa luvussa.

2.4. Jatkuva käyttöönotto

Jatkuva käyttöönotto (Continuous Deployment) on jatkuvasta toimittamisesta seuraava vaihe, jossa myös tuotantoasennus tehdään automaattisesti asennusputken lopussa. Myös

jatkuvassa toimittamisessa toteutetaan usein jatkuvaa käyttöönottoa esimerkiksi kehitys- tai testiympäristöön, johon jokaisten muutosten ja hyväksyntätestien läpäisyn jälkeen uusi versio asennetaan automaattisesti. Virallisesti jatkuvalla käyttöönotolla tarkoitetaan kuitenkin sitä, että uusi versio asennetaan aina automaattisesti tuotantoon asti. Tämän toimintamallin käyttöönotto vaatii äärimmäisen kattavat ja automatisoidut testauskäytännöt. [4, s. 266-267]

3. VAADITTAVIA KÄYTÄNTÖJÄ

Jotta järjestelmästä voidaan asentaa jatkuvasti uusin versio tuotantoon, täytyy luonnollisesti olla varmuus siitä, että ohjelmisto vastaa aina käyttäjätarpeisiin, siinä ei ole kriittisiä virheitä ja se toimii siinä ympäristössä niillä konfiguraatioilla millä se ollaan asentamassa. Tämä asettaa paljon vaatimuksia sille, miten ohjelmistokehitystä tehdään ja miten järjestelmä testataan niin ympäristöjen kuin itse ohjelmiston toiminnan osalta.

Osa käytännöistä voi tuntua työläiltä ja vaikeilta, mutta monessa yrityksessä ja projektissa nämä ovat arkipäivää. Mitä aikaisemmassa projektin vaiheessa käytäntöjä aletaan noudattaa, sitä pienemmällä vaivalla ne saadaan rutiininomaiseksi toiminnaksi ja toiminnan ylläpitäminen halutussa mallissa on pieni työ. Jos kaikki jatkuvan toimittamisen luotettavaan toteuttamiseen liittyvät käytännöt ovat jääneet huomiotta, voi niihin siirtyminen jälkikäteen olla isossa organisaatiossa kuitenkin jopa vuosien työ [2, s.51].

Jatkuvan toimittamisen selkärankana on asennusputki, jonka läpi muutokset ohjelmakoodissa valuvat aina tuotantoon asti. Tässä luvussa käydään läpi jatkuvan toimittamisen käytäntöjä ja niiden roolia automatisoidussa asennusputkessa.

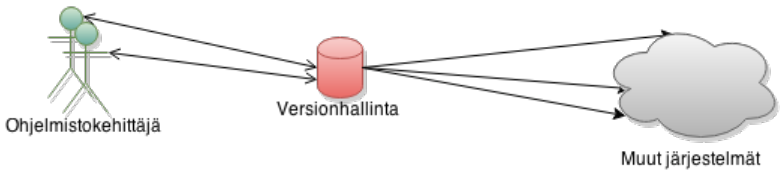
3.1. Versionhallinta

Versionhallinnat mielletään usein työkaluiksi ohjelmakoodin varastointiin ja eri käyttäjien muutosten yhdistämiseen sekä seurantaan. Erilaiset wikit ja dokumentinhallintaohjelmistot ovat toimiva tapa ylläpitää muun muassa ohjeita ja oppaita. Usein versio- ja muutoshistoriaa on dokumenttikohtaisesti helppo seurata myös dokumentinhallintaohjelmista. On kuitenkin hyvin tyypillistä, että esimerkiksi tietokannan alustamiseen käytettävät komentorivikomennot tai ohjelmiston asentamiseen liittyvät tekniset ohjeet löytyvät tästä samasta wikistä. Pahimmassa tapauksessa niitä etsitään sähköpostista tai joidenkin pikaviestiohjelmien lokeista.

Edes wiki tai dokumentinhallintaohjelma ei ole oikea paikka pitää teknisiä ohjeita sii-

tä, miten ohjelmisto asennetaan, mitä riippuvuuksia sillä on ja miten palvelimet kuuluu olla konfiguroitu. Ohjeistuksiin kuuluu vain hyvin abstraktilla tasolla oleva selitys asioista niin, että ohjeet läpi selailemalla ymmärtää karkeasti mistä on kyse. Ohjesivuilla on taipumus vanhentua välittömästi, kun konfiguraatioihin tehdään muutoksia.

Versionhallinta on oikea paikka kaikille testeille, skripteille, dokumentaatiolle, riippuvuuksille ja konfiguraatiotiedostoille. On myös tärkeää pitää versionhallinnassa kaikkea sitä tietoa, joka vaaditaan ympäristöjen ja palvelinten pystyttämiseen. Versionhalinnassa täytyy olla vähintään kaikki tarvittava, jotta voidaan tyhjästä luoda ympäristöt ja kääntää ohjelmistot niihin suoritettavaksi. Tavoitteena voidaan pitää tilannetta, jossa kaikki tieto, joka muuttuu projektin aikana, pidetään versionhallinnassa. Tällöin voidaan milloin tahansa palauttaa koko järjestelmä mihin tahansa tilaan, missä se on aiemmin ollut. [4, s.33]



Kuva 3.1: Kaikki koodit, skriptit ja dokumentaatio on versionhallinnasta kaikkien käytettävissä.

Jatkuvan toimittamisen kannalta versionhallintakäytäntöihin liittyy erityisesti konfiguraatioiden, skriptien, testien ja ohjelmakoodin säilyttäminen versionhallinnassa. Koska ympäristöt asennetaan ja konfiguroidaan automaattisesti ja ohjelmisto testataan sekä asennetaan automaattisesti, on välttämätöntä pitää näihin liittyvät toiminnallisuudet ja skriptit versionhallinnassa.

3.2. Konfiguraatioiden hallinta

Vaikka monet automatisoinnin käytännöt aina testauksesta asennuksiin ovat yleistyneet, on konfiguraatioiden hallinta niin ohjelmiston kuin palvelinten kannalta harvemmin automatisoitu, ylläpidettävissä ja seurattavissa.

Konfiguraatioiden hallinta tarkoittaa prosessia, joka määrittelee miten kaikki järjestelmään liittyvät komponentit ja riippuvuudet säilytetään, haetaan, tunnistetaan ja niitä muokataan. Versionhallinta on yleisin käytäntö myös näiden tietojen ylläpitoon ja hallintaan,

mutta tärkeintä on, että tiimi – sen koosta välittämättä – vastaa kunnollisesta konfiguraatioiden hallinnasta. [4, s. 31]

Mitä heikompi hallinta ohjelmakoodia suorittavaan ympäristöön on, sitä todennäköisemmin kohdataan jotain odottamatonta toimintaa. Aina kun ohjelmisto asennetaan, halutaan kokonaisuutena hallita kaikkea sitä, mitä ollaan asentamassa. Usein ohjelmiston toimittaja ei ole vastuussa lainkaan ympäristöstä johon ohjelmisto asennetaan – erityisesti jos käyttäjä itse asentaa ohjelman, kuten pelin tai toimisto-ohjelman. Automaattiset asennukset valittuihin ympäristöihin ja hyväksyntätestien suorittaminen niitä vasten tarjoaa helpotusta ongelmaan. [4, s. 129]

Ympäristöjen kunnollisen hallinnan uskotaan usein olevan kallista verrattuna siitä saatuihin hyötyihin. Usein tilanne on kuitenkin päinvastainen: Suurin osa tuotantoympäristön ongelmista johtuu riittämättömästä hallinnasta. Tuotantoympäristön pitäisi olla täydellisesti lukittu niin, että muutoksia tuotantoon tehdään vain automaattisten prosessien kautta. Tämä sisältää sekä itse ohjelmiston asentamisen, kuin kaikki siihen sekä palvelimeen liittyvät konfiguraatiot aina palomuureista varusohjelmiin asti. Vain tällöin voidaan luotettavasti auditoida ympäristöjä, diagnosoida ongelmia ja korjata niitä. Kun järjestelmän koko kasvaa, kasvaa myös erityyppisten palvelinten määrä ja suorituskykyvaatimukset. Tällöin myös hallinnan taso tulee entistä tärkeämmäksi. [4, s. 129]

Sama konfiguraatioiden hallintaprosessi kuin tuotannossa, pitäisi olla myös muissa ympäristöissä, esimerkiksi testi- ja demoympäristöissä. Kaikkien järjestelmien ja palvelinten konfiguraatioita ja väliohjelmistoja voi tällöin helposti muuttaa ja testata niiden vaikutusta esimerkiksi suorituskykyyn tai toisiinsa. Kun tulokset näyttävät tyydyttäviltä, voidaan konfiguraatiot replikoida suoraan tuotantoympäristöön. [4, s. 130]

Mahdollisuus luoda ympäristöjä tarvittaessa automatisoidusti voi tuoda myös huomattavia säästöjä mikäli projektissa tehdään erillistä manuaalista testausta – esimerkiksi tutkivaa testausta tai suorituskykytestausta. Uuden ympäristön pystyttäminen manuaalisesti voi olla useamman päivän työ. Automatisoitu ympäristön asentaminen säästää myös testiympäristöjä asennettaessa aikaa ja minimoi konfiguroinnin yhteydessä mahdolliset inhimilliset virheet. [2, s. 52].

Mikäli konfiguraatioiden hallinta on järjestelmässä kunnossa, pitäisi seuraaviin kohtiin voida vastata kyllä [4, s. 31]:

- Pystyn luomaan tyhjästä minkä tahansa ympäristön käyttöjärjestelmästä lähtien, sisältäen verkkoasetukset ja ohjelmistot konfiguraatioineen.
- Pystyn tekemään pienen muutoksen ympäristön konfiguraatioihin ja asentamaan sen joko yhteen tai kaikkiin ympäristöihin.
- Pystyn näkemään yksittäiset muutokset mitä ympäristöihin on tehty, kuka ne on tehnyt ja milloin ne on tehty.
- Jokaisen tiimin jäsenen on helppoa saada haluamansa informaatio ja tehdä siihen muutoksia.

Konfiguraatioista puhuttaessa voidaan puhua karkealla tasolla kahden tyyppisistä konfiguraatioista. Toiset ovat ajoalustaan liittyviä, jotka sisältävät esimerkiksi ajoympäristön ohjelmistoon tai verkkoasetuksiin liittyviä konfiguraatioita. Toiset ovat itse ajettavaan ohjelmistoon liittyviä konfiguraatioita. Palvelinkonfiguraatioiden automaattiseen hallintaan on olemassa yleisessä käytössä olevia työkaluja. Näistä esitellään lyhyesti muutama luvussa 4.

Ohjelmiston suoritukseen liittyviä konfiguraatioita voidaan asettaa joko ohjelmiston kääntämis- tai paketointivaiheessa, asennusvaiheessa tai sen käynnistymisvaiheessa. Paketointivaiheen konfiguraatiot voivat olla esimerkiksi riippuvuuksia ulkoisiin kirjastoihin. Jossain tapauksessa ohjelmistosta käännetään eri versio erilaisilla asetuksilla eri ympäristöihin. Tämä on usein huono käytäntö, koska versio, joka on testattu, tulisi olla asennettavissa kaikkiin ympäristöihin [4, s. 41-42]. Kääntämis- ja paketointivaiheessa on kaikki paketointiin tarvittavat konfiguraatiot ja riippuvuudet oltava versionhallinnassa tai muissa säiliöissä ladattaviksi.

Myös järjestelmän asennukseen liittyvät konfiguraatiot on kuitenkin hyvä olla versionhallinnassa [4, s.33]. Nämä voivat ohjelmistosta riippuen olla esimerkiksi palvelinten osoitteita, käyttäjätunnuksia ja salasanoja. Lisäksi asennuksen tekevät skriptit tai muut suoritettavat ohjelmat, joita asennukseen tarvitaan, on oltava versionhallinnassa.

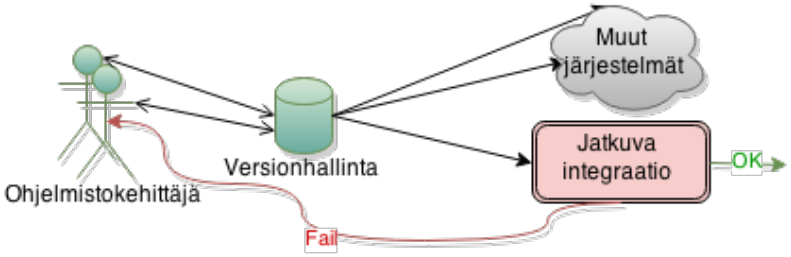
Usein järjestelmä lukee konfiguraatioita myös käynnistyessään. Nämä voivat olla asetuksia tietokantayhteyden muodostamiseen, lokitusasetuksia tai ympäristökohtaisia asetuksia. Myös nämä on pidettävä versionhallinnassa. Ohjelmisto voi käynnistyessään noutaa nämä asetukset [4, s.41], tai ne voidaan siirtää ajoympäristöön asennuksen yhteydessä,

jolloin niitä on loogista pitää samassa versionhallinnassa kuin asennuksen tekevät skriptit.

3.3. Jatkuva integraatio

Jatkuva integraatio (Continuous Integration) on osa jatkuvan toimittamisen asennusputkea. Jatkuvalla integraatiolla pyritään välttämään tilannetta, jossa ohjelmakoodia kirjoitetaan jatkuvasti ilman ohjelmiston kääntämistä ja testaamista toimivaksi siinä ympäristössä, jossa sen on tarkoitettu toimivan.

Jatkuvasta integraatiosta kirjoitti Kent Beck vuonna 1999 osana tunnetun Extreme Programming -ohjelmistotuotantomenetelmän käytäntöjä [4, s. 56]. Ilman jatkuvaa integraatiota ohjelmiston voidaan ajatella olevan aina rikki, kunnes se erikseen todistetaan toimivaksi asennuksen tai testauksen yhteydessä. Jatkuvan integraation käytännöillä tämä integraatio ja automatisoituja testejä ajetaan automaattisesti aina, kun ohjelmistoon tehdään muutoksia. Jatkuvan integraation käytäntöjä noudatettaessa ohjelmiston voidaan todeta olevan aina toimiva ellei toisin todisteta.



Kuva 3.2: Jatkuva integraatio.

Vaihe yhdistetään versionhallintaan niin, että muutokset versionhallinnassa käynnistävät jatkuvan integraation toiminnot automaattisesti (kuva 3.2). Mikäli jatkuvan integraation vaihe läpäistään, on järjestelmästä onnistuneesti tehty asennuspaketti tai muu artefakti, jota vasten on ajettu testit. Näiden testien on taattava se, että ohjelmisto toimii ja voidaan asentaa. Mikäli tämä vaihe epäonnistuu, on muutoksen tehneen kehittäjän lopetettava keskeneräiset työt ja tehtävä vaadittavat korjaustoimenpiteet jotta järjestelmä saadaan taas stabiiliin tilaan [4, s.58-59].

Uudet muutokset on syötettävä jatkuvaan integraatioon vähintään muutaman tunnin välein. Haasteena tiimissä ohjelmointina ei ole pelkästään töiden jakaminen ja hallitseminen, vaan erityisesti näiden tuotosten integroiminen. Uusien muutosten integroiminen nykyiseen ja muihin järjestelmiin voi viedä enemmän aikaa kuin itse uuden toiminnallisuu-

den tekeminen. Mitä kauemmin muutoksia tehdään integroimatta niitä kokonaisuutena, sitä kalliimmaksi ja vaikeammin ennustettavaksi integrointi tulee. Jatkuvan integraation vaiheessa pitäisi joka kerralla tuloksena syntyä kokonaisuutena se tuote, jota projektissa toteutetaan. Jos lopputulos on asennuslevy, tuloksena olisi oltava asennuslevy. Jos tuotteena on verkkopalvelu, tuloksena olisi tuotettu ja asennettu verkkopalvelu. [1, s.49-50]

3.4. Automaattinen testaus

Ohjelmistojen toiminnallinen testaus on muuttunut merkittävästi vuosien aikana. Järjestelmällinen ja manuaalinen ihmisten tekemä testaus ja virheiden korjaaminen ennen julkaisua voi viedä päiviä aikaa ja aiheuttaa organisaatiolle huomattaviakin kustannuksia [10, s.121-s.122]. Manuaalisen testauksen lisäksi ohjelmakoodille kirjoitetaankin usein testejä, jotka ajavat ohjelmakoodia ja varmistavat järjestelmän tekevän sen mitä sen kuuluu tehdä, ja toimivan poikkeustiloissa niin, kuin sen oletetaan toimivan. Kattava testiautomaatio on välttämätön, mikäli tuotantoasennuksia halutaan tehdä säännöllisesti [10, s.124]. Erityinen etu järjestelmälle kirjoitetuista koneellisesti ajettavista testeistä on siinä, että testien suorittamisen voi automatisoida. Ankara testiautomaatio on avainasemassa, mikäli jatkuvaa toimittamista halutaan tehdä [3].

Työ, joka tehdään automaattitestien kirjoittamiseen ja niiden automaattiseen suorittamiseen, vie luonnollisesti kehitystiimiltä aikaa johtuen testien ja ympäristöjen monimutkaisuudesta. Kuitenkin käytetty aika ja työ Humblen ja Farleyn kokemusten perusteella maksaa itsensä takaisin moninkertaisesti. Nämä testit tarjoavat suojan, jonka turvin järjestelmään voidaan tehdä suurempiakin muutoksia niin, ettei laatu kärsi. Ajatuksena on tuoda työmäärää, joka tulisi vastaan vasta myöhemmässä vaiheessa, aikaisemmin työnalle. Heidän kokemustensa perusteella heikot automaattiset hyväksyntätestit johtavat johonkin seuraavista: Kulutetaan paljon aikaa prosessin loppupuolella virheiden korjaamiseen, manuaaliseen massiiviseen testaamiseen tai lopputuloksena julkaistava ohjelmisto on yksinkertaisesti huonolaatuinen. [4, s.213]

Kattava testiautomaatio toimii suojana myös uusille projektitiimin ohjelmistokehittäjille. Kehitysympäristössä, jossa virheet jäävät todennäköisemmin ajoissa kiinni, uskaltavat uudet kehittäjät tehdä muutoksia koodiin rohkeammin ja ovat täten tuottavampia aikaisemmin. [10, s.126]

Kattavalla testiautomaatiolla ja jatkuvalla toimittamisella voidaan parantaa tuotteen laatua merkittävästi ja saada virheitä vähennettyä jopa 90%. Mikäli automaattiset testit saavat virheet kiinni aikaisessa vaiheessa, voivat kehittäjät korjata ne välittömästi. Projektissa voidaan päästä jopa tilanteeseen, jossa virheitä on tuotannossa niin harvoin, ettei virheiden hallintaan ja niiden korjausten seurantaan tarvita lainkaan työkalua. Tällöin myös virheiden korjaamiseen käytettävä aika voi pienentyä huomattavasti. [2, s.53]

3.4.1. Testityyppejä

Testejä voidaan luokitella karkealla tasolla esimerkiksi kuvan 3.3 mukaisesti. Kuvasta voidaan todeta, että kaikki testaus, joka ei vaadi ihmisen inhimillistä ajattelua, voidaan jossain määrin automatisoida. Yleistyneimpiä lienevät ohjelmakoodin yhteyteen kirjoitettavat yksikkö- ja integraatiotestit, jotka käytännössä suorittavat vain osia ohjelmasta ja todentavat näiden komponenttien tekevän sen mitä niiden on tarkoitus.

Automatisoitu	**Manuaaliset**
Toiminnalliset hyväksyntätestit	Käyttötapaukset Käytettävyystestit Tutkiva testaus
Yksikkötestit Integraatiotestit Järjestelmätestit	Ei-toiminnalliset hyväksyntätestit (suorituskyky, tietoturva,...)
Automatisoitu	**Manuaaliset / Automatisoitu**

Kuva 3.3: Testijaottelu, Brian Marick [4, s.85].

Hyväksyntätestit poikkeavat näistä testeistä ajatuksella, että hyväksyntätestin läpäisy kertoo jonkun ominaisuuden olevan toimiva tai jonkun toiminnallisen vaatimuksen täyttymisen. Ei-toiminnallinen hyväksyntätesti voi siis yhtä hyvin olla testi, joka takaa että järjestelmän pystyy kääntämään, asentamaan tai että se kestää määritellyn määrän käyttäjiä. Toiminnallisia hyväksyntätestejä voidaan kirjoittaa suoraan määrittelyä vasten niin, että hyväksyntätestin läpäisy kertoo ominaisuuden olevan toteutettu ja toimivan. [4, s.85]

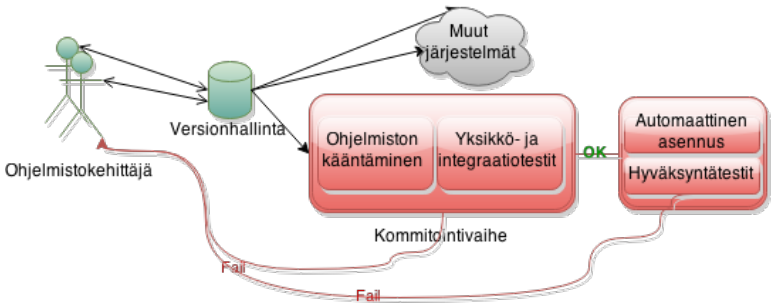
Toiminnallisena hyväksyntätestinä voisi olla esimerkiksi verkkokaupan toiminnallisuuksia määritellessä seuraava ominaisuus: Kun käyttäjä lisää verkkokaupassa tuotteen ostoskoriin, tuote näkyy ostoskorissa käyttäjän tarkastellessa ostoskoria. Hyväksyntätestin tälle skenaariolle sisältäisi mahdollisesti tarvittavat alustukset, lisäisi tuotteen ostoskoriin ja varmistaisi että tuote näkyy ostoskorisivulla.

Ideaalitapauksessa toiminnalliset hyväksyntätestit suoritettaisiin oikealla käyttöliittymällä todellisessa ympäristössä [4, s.88]. Esimerkiksi toteutettavan järjestelmän ollessa verkkopalvelu, voidaan käyttää tarjolla olevia työkaluja, joilla selainta voi komentaa etsimään ja klikkaamaan elementtejä sekä verifioimaan että sivulta löytyy vaadittuja asioita.

3.4.2. Testien jakaminen asennusputken vaiheisiin

Yksikkö- ja integraatiotestit ajetaan tyypillisesti asennusputken alkuvaiheessa. Nämä eivät kuitenkaan riitä takaamaan, että järjestelmän tämän hetkinen versio vastaa toiminnallisia tarpeita. Tästä syystä myös hyväksyntätestit ja niiden automaattinen ajaminen on oleellinen osa asennusputkea. Hyväksyntätestien luonteesta riippuen ne voivat olla osa alkuvaihetta tai tulla suoritettavaksi vasta myöhemmässä vaiheessa asennusputkea. Osa hyväksyntätesteistä voidaan mahdollisesti ajaa jo kääntämis- tai integraatiotestausvaiheessa.

Jos hyväksyntätestejä ajetaan kuitenkin tuotantoa vastaavaa ympäristöä vasten, on vaiheen suorittaminen tyypillisesti huomattavasti hitaampaa. Automatisoinnin erityisominaisuutena on nopeus – palautetta voidaan saada ja on täten saatava heti muutosten jälkeen [4, s.14]. Mikäli testit ovat kattavia, voi niiden ajamisessa kestää tunteja. Tällöin ohjelmistokehittäjät voivat alkaa suhtautua testituloksiin välinpitämättömästi [10, s.126]. Tuodaan siis omat vaiheet nopeammin ja hitaammin suoritettaville testeille kuvan 3.4 mukaisesti.



Kuva 3.4: Automaattinen testaus osana asennusputkea.

Ensimmäistä vaihetta nimitetään jatkuvassa toimittamisessa kommitointivaiheeksi (Commit stage, [4, s.119-120]). Kommitointi on muutos versionhallintaan, joka käynnistää automaattisesti asennusputken. Koska kehittäjien oletetaan tekevän korjaukset heti jos putkessa tapahtuu jotain virheellistä, on palautetta saatava mahdollisimman nopeasti. Jos hyväksyntätestejä ajetaan tuotantoa vastaavassa ympäristössä ajossa olevaa järjestelmää

vasten, on ne usein eriytettävä myöhempään vaiheeseen, jotta kommitointivaihe pysyy mahdollisimman nopeana. Kommitoivan kehittäjän olisi myös syytä odottaa kommitointivaiheen valmistumista ennen töiden jatkamista [4, s.121]. Tästä syystä kommitointivaiheessa on vain käännettävä ohjelmisto tarvittaessa ja ajettava vain nopeasti suoritettavia testejä.

Toinen vaihe on hyväksyntätestausvaihe. Koska hyväksyntätesteistä on käytännössä löydyttävä testitapaukset siitä, että järjestelmä voidaan asentaa ja sitä voidaan käyttää, on hyväksyntätestausvaiheessa tyypillisesti asennettava järjestelmä ja testattava sen toimintaa kohdeympäristössä. Mikäli kyseessä on verkkopalvelu, asennetaan verkkopalvelu ja ajetaan sitä vasten hyväksyntätestit selaimella. Jos taas kyseessä on natiivi ohjelmisto, on asennettava natiivi ohjelmisto ja testit ajettava sen laiteympäristön mukaisilla työkaluilla.

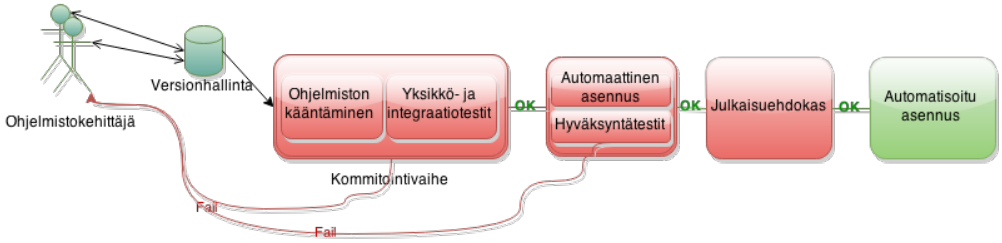
Ympäristö, johon järjestelmä asennetaan ja jota vastaan hyväksyntätestejä ajetaan, olisi oltava mahdollisimman vastaava kuin tuotantoympäristökin. Ympäristöjen olisi hyvä olla identtisiä, mikäli tämä ei ole liian kallista. Muussa tapauksessa voidaan virtualisoinnin avulla pyrkiä mahdollisimman lähelle tuotantoympäristöä. Näiden hyväksyntätestien jälkeen on automaattisesti todettu, että kyseinen versio voidaan asentaa tuotantoympäristöön ja se toimii siellä oletetusti. [4, s.217]

Hyväksyntätestausvaihe on asennusputken viimeinen vaihe, jossa sekä vaiheen suorittaminen että päätös seuraavaan vaiheeseen jatkamisesta ovat täysin automatisoituja ilman ihmisen tekemää arviointia ja päätöksiä jatkamisesta. Tästä syystä hyväksyntätestausvaihe on äärimmäisen tärkeä kynnys, ja siihen on suhtauduttava sen vaatimalla vakavuudella. [4, s.213]

3.5. Julkaisuehdokas ja automatisoitu asennus

Vain automaattisen hyväksymistestausvaiheen läpäisy johtaa julkaisuehdokkaan tuottamiseen, ja vain julkaisuehdokkaat voivat jatkaa asennusputkessa eteenpäin. Julkaisuehdokkaalla tarkoitetaan jotakin versiota järjestelmästä, joka on todettu virheettömäksi ja täyttävän hyväksyntäkriteerit. Usein ohjelmistoprojekteissa julkaisuehdokas luodaan erikseen prosessin loppuvaiheessa pitkien ja työläiden vaiheiden jälkeen. Jatkuvan toimittamisen putken julkaisuehdokkaiden laatu on yleensä huomattavasti parempaa ja manuaalista testausta täytyy tehdä vain funktionaalisen toiminnan varmistamiseen. [4, s.22-23, 213]

Järjestelmän asentamisessa testiympäristöön tai tuotantoympäristöön on väistämättä eroa. Eron olisi oltava kuitenkin vain konfiguraatioissa. Asennusprosessin on oltava vastaava tuotantoympäristöön kuin mihin tahansa muuhunkin ympäristöön. Automaattisen asennusprosessin käynnistämisen yhteydessä olisi valittava vain asennettava versionumero sekä kohdeympäristö. [4, s.249]



Kuva 3.5: Julkaisuehdokas ja automatisoitu asennus.

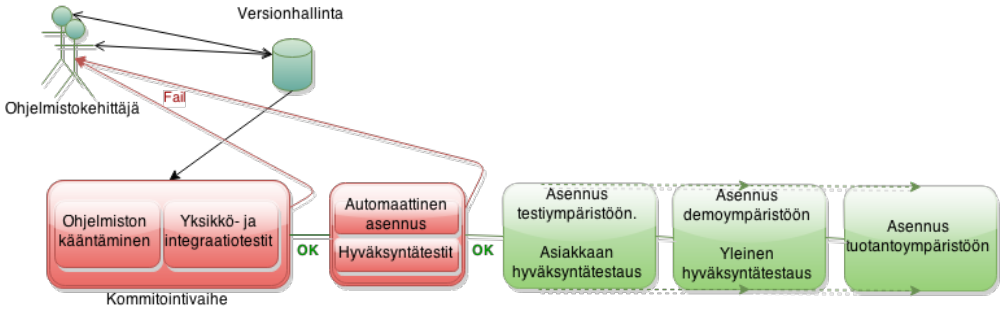
Tämä automatisoitu, mutta manuaalisesti käynnistettävä prosessi, jolla asennus tehdään mihin tahansa ympäristöön, on oltava osa asennusputkea, kuten kuvassa 3.5 on esitetty. Asennettavissa olevat versiot on voitava listata ympäristökohtaisesti ja uuden version asennuksen tai konfiguraatiomuutosten ajaminen johonkin ympäristöön täytyy tapahtua nappia painamalla. Kyseisen prosessin pitäisi myös olla ainoa tapa asentaa uusia versioita tai muuttaa konfiguraatioita. Tällöin myös historia asennuksista, konfiguraatiomuutoksista ja henkilöistä jotka muutoksia ovat tehneet, on nähtävillä. [4, s.249]

Suunnitelma siitä, miten julkaisu ja asennus tullaan toteuttamaan, on tehtävä projektin alkuvaiheessa asennusputkea suunniteltaessa [4, s.250]. Kuten kaikki muutkin asennusputken vaiheet, myös automatisoidun asennuksen vaihe on hyvä olla toteutettuna alusta asti. Asennuksen automatisointi on aloitettava ensimmäisestä testiympäristöön asentamisesta niin, että ensimmäinen demo on asennettu automatisoinnilla [4, s.251]. Asennuksen automatisoinnin toteutus voi muuttua luonnollisesti projektin edetessä, sillä erityisesti ketterässä kehityksessä ei projektin alkuvaiheessa olla tietoisia kaikista riippuvuuksista, järjestelmän kehittymisestä sekä palvelin- tai tietokantavaatimuksista.

3.6. Asennuksen porrastettu ylennys

Järjestelmästä tulee monimutkaisempi sen kasvaessa, jolloin myös asennusputkesta tulee monimutkaisempi. Pakettien luominen, kommitointivaihe ja hyväksyntätestausvaihe ovat

ensimmäisiä tavoitteita asennusputkessa. Nämä tavoitteet voidaan ajatella maaleiksi, joiden läpäiseminen ylentää asennuspakettia tai muuta tuotettua artefaktia automaattisesti asennusputken edetessä, kuten kuvassa 3.6. Porrastetulla ylennyksellä tarkoitetaan karkeasti siis tapahtumaa, jossa asennuspaketti läpäisee jonkun asennusputken vaiheen, joka tulkitaan maaliksi. Hyväksyntätestausvaiheen jälkeiset porrastetun ylennyksen vaiheet ovat tyypillisesti manuaalisesti käynnistettäviä asennuksia eri ympäristöihin.



Kuva 3.6: Asennuksen porrastettu ylennys.

On tärkeää, että aikaisemmassa vaiheessa tuotettua asennuspakettia ei luoda uudestaan asennusputken edetessä, vaan kerran tuotettu ja testattu artefakti on se, joka asennusputkessa etenee. Ohjelmiston uudelleenkääntämisen yhteydessä on mahdollista, että uudessa tuotteessa on eroavaisuuksia verrattuna siihen tuotteeseen, mikä on testattu sekä automatisoidusti että manuaalisesti eri ympäristöissä. Tästä ilmenevien virheiden selvittely voi olla hyvin hankalaa. [2, s. 51]

Asennusputken hallintaan käytettävän työkalun täytyy mahdollistaa asennuspaketin ylentäminen sekä automaattisesti että määrättyjen käyttäjien toimesta. Tämä tarkoittaa sitä, että hyväksyntätestauksen läpäissyt asennuspaketti on voitava ylentää asentamalla se testiympäristöön niiden henkilöiden toimesta, joiden vastuulle se on sovittu. Nämä henkilöt voivat olla esimerkiksi kehittäjiä, testaajia, tuoteomistaja, asiakas tai kuka tahansa näistä – riippuen projektin henkilöstöstä ja rooleista. Ylentäminen ja asennus muihin ympäristöihin tulee tapahtua vastaavasti kaikkiin ympäristöihin kuin ensimmäiseenkin ympäristöön.

Jokainen asennus jokaiseen ympäristöön on tärkeä saada asennusputken ja porrastetun ylennyksen kautta, eikä yhtään maalia ole hyvä ohittaa, vaikka sen läpäisy epäonnistuisi. Mikäli jokin vaihe ei mene läpi, voi olla houkuttelevaa tehdä vaiheeseen erillisiä muutoksia niin, että ylennystä voidaan jatkaa – esimerkiksi alustaa testiympäristön tietokanta

tyhjästä. Nämä maalit ovat asennusputkessa kuitenkin syystä, eikä nopea asennussykli tarkoita sitä, että asennuksissa täytyy kiirehtiä. [10, s.125]

Automatisoitu asennus ylennyksen yhteydessä pitäisi olla tehtävissä yksinkertaisesti valitsemalla asennettava julkaisuehdokas ja ympäristö. Asennuksen tekevän henkilön ei tarvitse olla tietoinen siitä, miten asennus teknisesti taustalla tapahtuu. Riippumatta siitä, onko järjestelmä työasemalle tai palvelimelle asennettava ohjelmisto, on asennuksen ja ylennyksen tapahduttava seuraavasti: [4, s.256]

- Ylennyksen tekevä henkilö valitsee version ja ympäristön johon versio asennetaan.
- Ympäristö valmistellaan infrastruktuurin ja väliohjelmistojen puolesta.
- Ohjelmiston binäärit ja muut komponentit haetaan säiliöstä, johon ne on asennusputken aiemmassa vaiheessa tallennettu ja siirretään ne kohdeympäristöön.
- Asetetaan ohjelmiston ympäristökohtaiset konfiguraatiot.
- Valmistellaan tai migratoidaan ohjelmiston käyttämä data (esim. tietokanta).
- Savutestit, joilla todetaan asennuksen onnistuneen.
- Manuaalista tai automaattista testausta kohdeympäristössä.
- Jos kaikki vastaa oletuksia, voidaan asennuspaketti ylentää seuraavaan ympäristöön.
- Jos asennuksessa on jotain vikaa, kirjataan nämä ylös eikä ylennystä jatketa.

3.7. Komponenttien ja kirjastojen hallinta

Kirjastojen ja komponenttien hallintaan, joista järjestelmä on riippuvainen, on kaksi toimivaa ratkaisua: Ne voidaan joko pitää mukana versionhallinnassa tai pelkästään määritellä ja ladata erillisestä säiliöstä kääntämisvaiheessa. Toimintatavasta riippumatta tärkeintä on, että ohjelmiston kääntäminen voidaan toistaa milloin tahansa kenen tahansa toimesta niin, että tuloksena on aina sama tuote. Palatessa versionhallinnassa kolmen kuukauden takaiseen hetkeen, on oltava mahdollista tuottaa sama tuote kuin kolme kuukautta sitten tuotettiin. [4, s.354]

Kaikkien riippuvuuksien pitäminen samassa versionhallinnassa on yksinkertaisin ratkaisu. Tämä takaa sen, että järjestelmä voidaan kääntää kokonaisuutena mihin tahansa tilaan ilman ulkoisia riippuvuuksia. Isommissa projekteissa kirjastojen määrä voi kuitenkin kasvaa hyvin suureksi, eikä tämä ole mahdollista. [4, s.354]

Mikäli riippuvuudet määritellään ja ladataan säiliöstä käännösvaiheessa, on riippuvuuksien versiointi niiden määrittelyssä äärimmäisen tärkeää. Oman säiliön ylläpitäminen tarvittavia riippuvuuksia varten on hyvä käytäntö, jolloin kirjastot ja komponentit sekä niiden versiot pysyvät helpommin hallinnassa. [4, s.355]

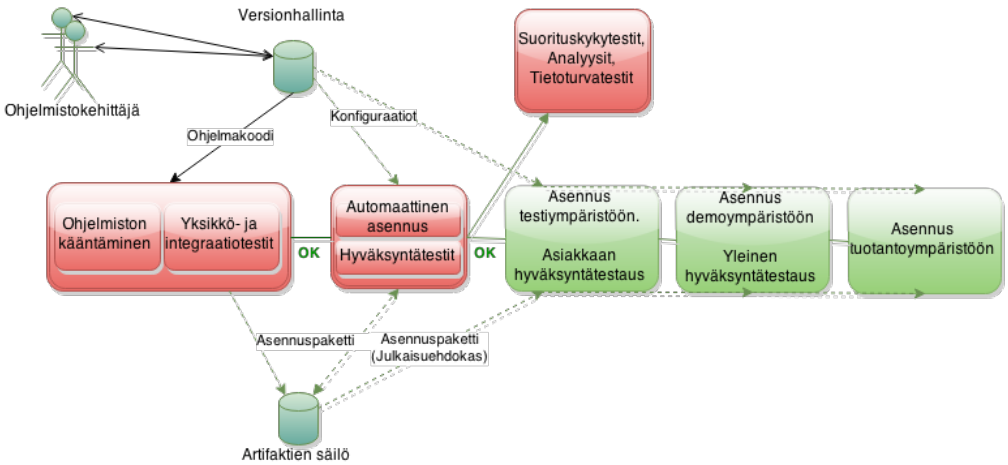
Usein kehitettävällä järjestelmäkokonaisuudella on myös sisäisiä riippuvuuksia, rajapintoja tai jaettuja kirjastoja, jotka ovat myös jatkuvassa muutoksessa. Nämäkin komponentit voidaan pitää samassa versionhallinnassa ja kääntää aina järjestelmän osan kääntämisen yhteydessä, tai kääntää erikseen vain jos niihin on tullut muutoksia ja ladata säiliöstä kun niistä riippuvia järjestelmän osia käännetään. Mikäli näitä riippuvuuksia päädytään kääntämään erillisiksi kirjastoiksi ja lataamaan säilöstä tarvittaessa, on niiden versiointi yhtä tärkeää kuin kaikkien muidenkin riippuvuuksien. Yleiskäyttöisen kirjaston tai komponentin versionumeroa voidaan muuttaa joka kerta kun siihen tehdään muutoksia, julkaista kirjastosta uusi versio, asentaa se säiliöön muiden käytettäväksi ja konfiguroida siitä riippuvat ohjelmistot käyttämään uutta versiota. Tällöin jokainen järjestelmän osa on riippuvainen aina jostain määritellystä versiosta.

3.8. Asennusputken yhteenveto

Asennusputki on automatisoitu prosessi, jonka avulla kaikki muutokset versionhallintaan saadaan lopulta hallitusti loppukäyttäjille. Jokainen muutos kulkee kompleksisen prosessin läpi, jonka lopussa se voidaan asentaa tuotantoon. Tämä prosessi sisältää kaikki vaiheet ohjelmiston kääntämiseen, testaamiseen, konfiguroimiseen ja asentamiseen liittyen. Asennusputki vastaa tästä prosessista tarjoten samalla mahdollisuuden seurata ja hallita sitä, miten muutokset siirtyvät automatisoitujen vaiheiden kautta käyttäjille. [4, s.106-107]

Mikäli virheisiin törmätään missä asennusputken vaiheessa tahansa, ei kyseinen muutos tai julkaisuehdokas voi jatkaa putkessa eteenpäin. Kaikki asennusputken vaiheet hakevat tarvittavat riippuvuudet versionhallinnasta tai erillisistä säiliöistä – ovat nämä sitten

ohjelmakoodia, konfiguraatioita, yleiskäyttöisiä kirjastoja tai komponentteja. Kaikki tuotokset viedään versionhallintaan tai säiliöihin, joista ne ovat muiden vaiheiden tai systeemien käytettävissä.



Kuva 3.7: Asennusputki.

Kuvassa 3.7 on punaisella kuvattu automaattisesti käynnistyvät vaiheet ja vihreällä ne, jotka voidaan konfiguroida käynnistymään automaattisesti tai ihmisen toimesta. Eitoiminnalliset hyväksyntätestit ovat yläoikealla, ja voidaan ajaa myös esimerkiksi ajastetusti tietyin väliajoin.

3.9. Tietokantamuutosten hallinta

Joskus ohjelmiston päivitys on suoraviivainen toiminto: Poistetaan vanha ohjelmisto ja korvataan se uudella. Suuri osa ohjelmistoista on kuitenkin riippuvaisia datasta, jota ne lukevat ja muokkaavat. Tämä datan rakenne tyypillisesti myös muuttuu ohjelmiston kehityskaaren mukana. Relaatiotietokannat ovat yleinen säiliö ohjelmistojen hallitsemalle tiedolle.

Tietokantatauluihin ja niiden välisiin suhteisiin tulee muutoksia uusien ominaisuuksien ja toiminnallisuuksien myötä. Mahdolliset muutokset, eli migraatiot, on suoritettava tietokantaan asennuksen yhteydessä ja mahdollisesti ainakin hyväksyntätestausvaiheessa kohdeympäristöön. Koska testaus ja asennus on automatisoitu, on myös tietokantamigraatioiden ajamiset automatisoitava.

Useat työkalut mahdollistavat tietokannan skeeman version ylläpitämisen ja uusien

migraatioskriptien ajamisen tietokantaan automatisoidusti. Mikäli ohjelmistoon tuotaessa muutoksia ilmenee tarve kantamuutoksille, on nämä tallennettava migraatiotiedostoina versionhallintaan ja suoritettava asennusputken yhteydessä tarvittavissa vaiheissa tarvittavaa ympäristöä vasten.

3.10. Ohjelmointikäytännöt ja haarojen pahuus

Jatkuvassa toimittamisessa asennusputki ohjaa ohjelmistokehitystä. Jos jatkuvan integraation vaiheessa tai hyväksyntätestausvaiheessa on virheitä, on ne korjattava välittömästi. Virheellisessä tilassa olevaan putkeen ei saa kommitoida mitään muutoksia ennen kuin se on saatu korjattua [4, s.66]. Kaiken ollessa kunnossa, on taas tärkeää integroida omat muutokset säännöllisesti, ainakin muutama kerta päivässä [4, s.59]. Ohjelmoijien kehitysympäristön pitäisi aina olla siistissä tilassa niin, että ohjelmisto kääntyy ja kaikkien testien ajaminen onnistuu. Ohjelmistokehittäjien on siis kehitettävä järjestelmää niin, että se on aina tuotantoasennettavissa [4, s.105].

Tyypillinen tapa hallita työn alla olevia ominaisuuksia on luoda versionhallintaan erillisiä haaroja. Haaroja luodaan milloin uusia ominaisuuksia, virhekorjauksia tai julkaisuja varten. Haaroissa olevia muutoksia yhdistetään kun nähdään tarpeelliseksi. Haarojen käyttöön on paljon oppaita ja hyväksi todettuja käytäntöjä, kuten Driessenin A succesfull Git branching model [6] sekä yleistynyt git-flow -työkalu [8] git-versionhallintaan.

Tässä toimintamallissa on kuitenkin muun muassa seuraavia ongelmia:

- Mikäli kommitointi tapahtuu vain omaan haaraan, ei muutokset valu muille käyttäjille näiden ladatessa muutoksia.
- Uudet ominaisuudet eivät ole keskeneräisinä kenenkään testattavissa vaikka asennuksia tehtäisiin testiympäristöön.
- Yhdistäminen aiheuttaa usein konflikteja, joiden ratkominen on työlästä ja aiheuttaa helposti virheitä.
- Erilliset haarat eivät ole mukana jatkuvassa integraatiossa ja toimitusputkessa.

Jatkuvan integraation perusta on tuoda luottamusta siitä, että järjestelmä kääntyy ja toimii funktionaalisesti. Jatkuvan toimittamisen sydämenä olevan toimitusputken tarkoituksena on taata, että järjestelmästä voidaan aina julkaista asennettava versio. Molemmat

käytännöt vaativat kuitenkin toimiakseen sen, että kehittäjät tekevät muutoksiaan päähaaraan. [4, s.346]

Suuri osa keskeneräisistä ominaisuuksista pystytään toteuttamaan päähaaraan niin, että ne ovat yksinkertaisesti piilotettuja käyttäjiltä eri ympäristöissä kunnes ovat valmiita [4, s.347]. Järjestelmään voidaan tuoda esimerkiksi profiili tai ominaisuuskytkin (feature flag, feature toggler), jonka avulla eri ominaisuudet ovat näkyvillä tai käytössä eri ympäristöissä [10, s.124]. Tällöin muutokset valuvat koko asennusputken läpi mahdollisesti tuotantoon asti pienissä osissa, ja virheiden korjaaminen niitä ilmetessä on helpompaa, kun muutoksia on tehty vain pieni määrä kerrallaan. Kytkimellä käyttöönotetun ominaisuuden saa tuotannosta myös tarvittaessa pois ilman erillistä versiojulkaisua, mikäli ominaisuuden käytössä ilmenee ongelmia [10, s.124].

Kaikki muutokset on tehtävä pienissä osissa niin, että jokaisen muutoksen jälkeen järjestelmä on eheä ja voidaan julkaista [4, s.349]. Valmistele ja muokkaa nykyistä toteutusta niin, että pystyt toteuttamaan uuden ominaisuuden osissa rikkomatta kokonaisuutta ja integroi muutoksesi jatkuvasti.

Joskus muutoksien tekeminen tarpeeksi pienissä osissa eheys säilyttäen voi olla kuitenkin liian työlästä tai mahdotonta. Tällöin haaran voi toteuttaa abstraktiotasona päähaaran ohjelmakoodissa. Abstraktiohaara luodaan ohjelmakoodin väliin esimerkiksi omana luokkana tai nimiavaruutena, ja uutta toteutusta aletaan kirjoittaa niin, että abstraktiotaso käyttää aluksi enemmän vanhaa toteutusta ja pikkuhiljaa enemmän uutta. Lopulta vanha toteutus voidaan poistaa ja abstraktiotaso sen mukana. [4, s.349-351]

4. TYÖKALUJA JA TEKNOLOGIOITA

Tässä luvussa esitellään korkealla tasolla muutamia yleisesti käytössä olevia työkaluja, joita jatkuvan toimittamisen käytäntöjen noudattamiseen voidaan hyödyntää. Tarkoituksena ei ole ottaa kantaa siihen, mikä vaihtoehtoisista työkaluista on paras, vaan esitellä miten työkalut ratkaisevat ongelmia eri tavoin.

4.1. Versionhallinta

Koska kaikkea projektin tietoja aina ohjelmakoodia, testejä ja konfiguraatioita myöten on hyvä pitää versionhallinnassa, on versionhallinta äärimmäisen tärkeä työkalu. Versionhallintaohjelmistoja on tarjolla useita, joiden soveltuvuus projektiarkeen riippuu sekä projektin muista teknologioista että tekijöiden omasta näkemyksestä tai kokemuksesta. Pääsääntöisesti tänä päivänä versionhallinnat voidaan jakaa kahteen kategoriaan: keskitettyihin ja hajautettuihin. Hajautetut versionhallinnat ovat nykyisillään kuitenkin ominaisuuksiltaan monipuolisempia ja toiminnoiltaan nopeampia kuin keskitetyt. Kaikki nykyajan versionhallinnat tarjoavat kuitenkin jäljitettävyyden ja mahdollisuuden palata versioissa taakse tai eteen.

Keskitetyt versionhallinnat (tunnettuja mm. CVS[1], Subversion[2]) perustuvat siihen, että nykyinen tila ja historia on vain keskitetyllä palvelimella, jota vasten käyttäjät peilaavat omaa versiotaan. Kun uudet muutokset halutaan tallentaa, täytyy ne kommitoida samassa yhteydessä keskitettyyn versionhallintaan, josta ne ovat välittömästi muiden käytettävissä.

Hajautetuissa versionhallinnoissa (tunnettuja mm. Git[3], Mercurial[4]) käyttäjällä on koko historia itsellään ja tiedot vain synkronoidaan keskenään keskistetyn etäpalvelimen kautta. Tällöin versionhallinnan ominaisuuksia voi hyödyntää monipuolisemmin il-

[1] http://www.nongnu.org/cvs/
[2] https://subversion.apache.org/
[3] https://git-scm.com/
[4] https://www.mercurial-scm.org/

man verkkoyhteyttä keskuspalvelimeen. Toiminnot jotka eivät tarvitse verkkoyhteyttä, ovat luonnollisesti myös nopeampia. Kommitit ovat muutoksia, jotka kuvaavat yksittäistä (mutta tyypillisesti useiden rivien ja tiedostojen) muutosta, jonka perusteella saadaan muutoshistorian avulla palautettua mikä tahansa historian vaihe. Kommitit työnnetään erikseen versionhallintaan muiden vedettäviksi – joko yksittäin tai useampi kommit kerrallaan. Haarat ovat myös kevyempiä, kuin keskitetyissä versionhallinnoissa, sillä ovat vain osoittimia tiettyyn kommittiin.

4.2. Palvelinten ja infrastruktuurin hallinta

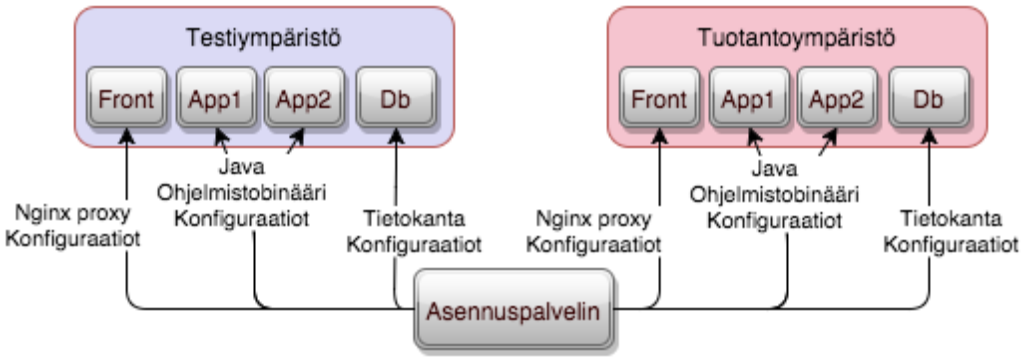
Myös palvelinten ja infrastruktuurin konfiguraatioiden hallintaan löytyy useita käyttökelpoisia työkaluja. Toiset työkalut pyrkivät ratkaisemaan saman haasteen hiukan eri tavalla, kun toiset täydentävät muiden puutteita. Joitakin työkaluja voi siis käyttää myös yhdessä riippuen siitä, miten ne ratkaisevat erilaisia haasteita ja mikä projektin tarpeisiin soveltuu. Jatkuvaa toimittamista ajatellen on työkalujen kuitenkin tarjottava ratkaisut kaikkiin kohdan 3.2. vaatimuksiin.

Useiden työkalujen toiminta perustuu konfiguraatiotiedostoihin, joilla määritellään palvelinten tai palvelinryhmien konfiguraatioiden tavoitetila. Tavoitetila sisältää esimerkiksi käyttäjiä, kansioita, tiedostoja, palomuuriasetuksia sekä asennettuja väliohjelmistoja tai muita binäärejä. Kun konfiguraatiot suoritetaan työkalulla palvelimelle, varmistetaan jokaista asetusta vasten, että tavoitetila on voimassa. Mikäli näin ei ole, tehdään tarvittavat toimenpiteet esimerkiksi binäärin asentamiseen tai tiedostojen ja käyttäjien luomiseen.

Työkalujen avulla voidaan samoista konfiguraatioista luoda ja hallita useita lähes identtisiä ympäristökokonaisuuksia niin, että eroavaisuuksia on vain ympäristökohtaisissa muuttujissa – esimerkiksi palomuurien ip-rajauksissa ja ohjelmistojen käyttämissä konfiguraatiotiedostoissa. Jos kaikki järjestelmän tuotantokäyttöön vaadittavat konfiguraatiot on luotu valitulla työkalulla, saadaan muutokset ajettua saman prosessin läpi joka ympäristöön, ja muutoshistoria on nähtävissä milloin tahansa versionhallinnassa työkalun konfiguraatioista.

Kuvassa 4.1 on esitetty yksinkertainen järjestelmä, mikä koostuu edustapalvelimesta, kahdesta applikaatiopalvelimesta sekä tietokantapalvelimesta. Kaikki näiden palvelinten konfiguroimiseen ja ohjelmistojen asentamiseen liittyvät konfiguraatiot ja vaiheet

ylläpidetään versionhallinnassa ja asennetaan tarvittaessa työkalulla yhdeltä palvelimelta tai työpisteeltä kohdennettuna. Samoja konfiguraatiopohjia on hyvä pitää yllä kaikissa kehitys-, testi- sekä tuotantoympäristöissä. Tällöin muutokset tulee vietyä samoilla konfiguraatioilla jokaiseen ympäristöön ennen tuotantoonvientiä.



Kuva 4.1: Palvelinten konfiguroiminen keskitetysti

Ansible[5] on automatisointityökalu, jonka toiminta perustuu YAML-muotoisiin konfiguraatiotiedostoihin. Ansible toimii työntämisperiaatteella (push) niin, että Ansiblella ajetaan yksittäiseltä koneelta konfiguraatiot sisään halutulle palvelimelle. Isäntäpalvelimia tai vastaavia ei tarvita – Ansiblen asentaminen koneelle, jolta konfiguraatioita halutaan työntää, riittää.

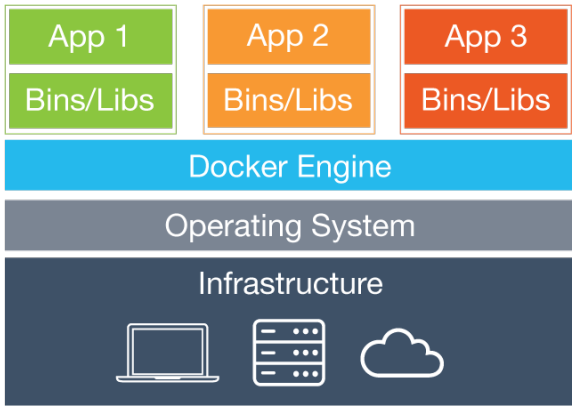
Puppet[6] ja **Chef**[7] ovat toiminnaltaan hyvin vastaavia kuin Ansiblen kanssa. Konfiguraatiot kirjoitetaan kuitenkin YAML:n sijaan oletuksena Ruby:llä. Selkeänä eroavaisuutena on myös se, että Puppet ja Chef hoitavat palvelinten konfiguraation erillisen agentin avulla, joka täytyy asentaa palvelimille. Tämä agentti vetää (pull) muutoksia taas erilliselle palvelimelle asennetulta masterilta. Vetäminen voidaan käynnistää manuaalisesti tai ajastaa niin, että muutokset masterissa vedetään automaattisesti kaikille palvelimille tietyn ajan kuluessa.

Docker[8] on työkalu, jolla voidaan eriyttää ohjelmiston vaatimat binäärit, väliohjelmistot ja konfiguraatiot Dockerilla ajettavaksi kontiksi. Useita kontteja voidaan suorittaa samalla palvelimella, jolloin ne jakavat saman kernelin, mutta näkevät vain itsensä sekä mahdollisesti isäntäpalvelimelta erikseen jaetut resurssit. Resursseja voivat olla esimerkiksi ympäristömuuttujat tai kansiot ohjelmiston suoritukseen liittyvien konfiguraatioiden

[5] http://www.ansible.com/
[6] https://puppetlabs.com/
[7] https://www.chef.io/
[8] https://www.docker.com/

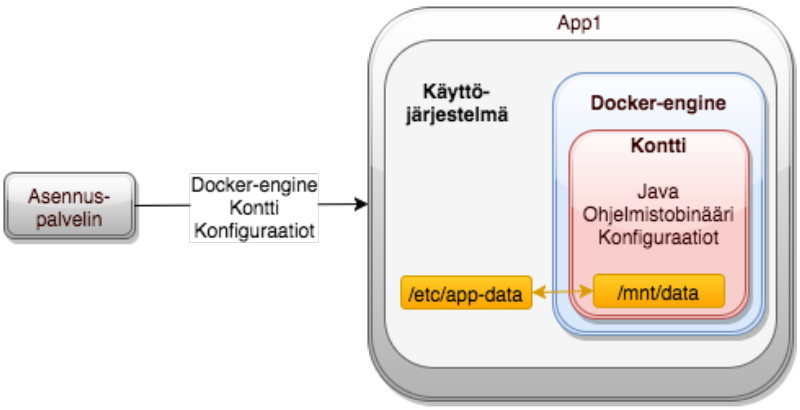
lukemiseen sekä lokien kirjoittamiseen.

Asennettavan ohjelmiston sijaan palvelimelle asennetaan Docker-kontti, joka sisältää kaiken sen, mitä itse ohjelmiston suorittamiseen vaaditaan. Sama kontti voidaan asentaa useampaan ympäristöön, jolloin sen toiminta väliohjelmistoineen voidaan testata muissa ympäristöissä ennen tuotantoympäristöä, kun riippuvuuksia kontin ulkopuolelle on vähän.



Kuva 4.2: Docker Engine ja kolme konttia palvelimella, https://www.docker.com/whatisdocker

Vaikka Docker-kontin luomisen ja asentamisen voi automatisoida, on emopalvelimella yleensä myös konfiguraatioita – vähintään Docker Enginen asentaminen sekä asetuksia siitä, mitä resursseja kontille jaetaan tai mitä verkko-asetuksia kontille asetetaan. Tästä syystä Docker ei ole korvaava työkalu esimerkiksi Ansiblelle, Chef:lle tai Puppet:lle. Docker on hyvä työkalu eriyttämään samalla palvelimella ajettavien ohjelmistojen välisiä riippuvuuksia sekä helpottamaan esimerkiksi väliohjelmistojen päivittämistä samassa yhteydessä, kun itse asennettava ohjelmisto päivitetään.



Kuva 4.3: Docker-kontin asentuminen palvelimelle

Kuvassa 4.3 on esitetty, miten kuvan 4.1 mukainen yksittäinen ohjelmistopalvelin

voitaisiin konfiguroida suoritettavaksi Dockerilla. Emopalvelimelle ei asenneta Java-väliohjelmistoa, vaan asennettava ohjelmisto kääritään Docker-konttiin Java-asennuksen kera. Kontti on ajettavissa itsenäisenä sovelluksena, jolle kuvan mukaisesti on jaettu vain yksi kansio itse emopalvelimelta. Kun ohjelmistosta asennetaan uusi versio, vaihtuu koko Docker-kontti pelkän ohjelmiston binäärien sijaan.

4.3. Jatkuvan integraation hallinta

Jatkuvan integraation toteuttaminen vaatii käytännössä siihen dedikoidun palvelimen, jolla ohjelmistot käännetään, testataan ja asennusprosessit suoritetaan. Jatkuvan integraation palvelin pääsääntöisesti suorittaa tehtäviä eli skriptejä ajastetusti tai tietyn tapahtuman jälkeen – esimerkiksi muutoksen versionhallinnassa. Tehtävien hallintaan on olemassa useampia erillisiä ohjelmistoja graafisella käyttöliittymällä ja lisäosilla. Seuraavaksi esiteltävissä ohjelmistoissa ajojen hallinta tapahtuu selaimella ohjelmiston tarjoaman verkkopalvelun kautta.

Jenkins[9] on hyvin tunnettu ilmainen avoimeen lähdekoodiin perustuva jatkuvan integraation ohjelmisto. Jenkins on nopea ja helppo asentaa sekä konfiguroida. Lisäksi Jenkins on hyvin monipuolinen erityisesti lisäosatukensa vuoksi. Lisäosia on tarjolla paljon ja niiden asentaminen on helppoa ohjelmiston mukana tulevalta lisäosienhallintasivulta. Jatkuvan toimittamisen asennusputken toteuttamiseen ja hallintaan löytyy useampi hiukan toisistaan eroava – tai toisiaan täydentävä – lisäosa.

Bamboo[10] on maksullinen Atlassianin tuote. Bamboon voi integroida suoraan Atlassianin Jira-tehtävänhallintasovellukseen ja se tarjoaa valmiiksi toiminnallisuuksia muun muassa asennusputken hallintaan. Ohjelmistoon on tarjolla myös lisäosia, joista osa on maksullisia.

Go[11] on ThoughtWorksin tuote, joka on suunniteltu erityisesti jatkuvaan toimittamiseen ja asennusputken hallintaan. Go on avointa lähdekoodia ja ilmainen. Asennusputken hallinta on erittäin monipuolinen ja tarjoaa valmiiksi ominaisuuksia esimerkiksi tuotettujen artefaktien toimittamiseen uusille ajoille, joissa niitä voidaan yhdistää tai kasata uudeksi artefaktiksi asennusputken edetessä.

[9] https://jenkins-ci.org/
[10] https://www.atlassian.com/software/bamboo/
[11] https://www.thoughtworks.com/go/

4.4. Tietokantamuutosten hallinta

Suuri osa järjestelmistä säilöö ja käsittelee dataa, jota pidetään tallessa tietokannassa. Mikäli tietokannalla on erikseen määriteltävä skeema, tulee skeeman väistämättä muutoksia säännöllisin väliajoin. Relaatiokannat ovat näistä yleisimpiä ja näille onkin tarjolla useita työkaluja hoitamaan tarvittavien muutosten ajamisen tietokantaan. Työkaluissa on pääsääntöisesti sama periaate – kaikki tietokantamuutokset tallennetaan erillisiksi muutostiedostoiksi, jotka ajetaan tietokantaan tarvittaessa. Työkalu pitää huolta siitä, että vain ne tiedostot, joita ei ole vielä ajettu, ajetaan versiojärjestyksessä. Osa työkaluista tarjoaa myös lisäominaisuuksia muun muassa olemassaolevan datan monipuolisempaan migratointiin.

Flyway[12] on Java-ohjelma, joka suoritetaan antamalla sille tietokantayhteyteen tarvittavat tiedot sekä kansiopolku, josta SQL-tiedostot löytyvät, mutta se ei itsessään ota kantaa tiedostojen sisältöön. Flyway ylläpitää tietokantataulussa tietoa siitä, mitkä tiedostot on ajettu ja ajaa puuttuvat tiedostot tietokantaan järjestyksessä. Sen lisäksi, että Flywayn voi ajaa erikseen asennusputken vaiheessa, on siitä olemassa Java-kirjastoja, joiden avulla on mahdollista esimerkiksi ottaa Flyway osaksi omaa Java-pohjaista ohjelmistoa ja antaa tämän itse päivittää tietokanta tarvittaessa käynnistyessään. Migraatiotiedostoja voi kirjoittaa myös Javalla, jolloin esimerkiksi suurempia datamanipulaatioita voi tehdä erikseen ohjelmakoodilla mikäli SQL-kielen ilmaisukyky ei ole riittävä.

Tietokantamuutoksia ajaessa on kuitenkin mahdollista että päivitys epäonnistuu, joten usein on parempi pitää tietokantamigraatioiden suorittaminen omana vaiheenaan. Flyway ei tarjoa lainkaan työkaluja helpottamaan aiemman tilan palauttamiseen tai ajettujen migraatioiden peruuttamiseen, sillä relaatiotietokannoissa tämä ei ole niin yksinkertaista. Migraatiot voivat poistaa tietoa, ja osa tietokantakyselyistä ei toteutukseltaan tue rollback-toimintoa. Flyway kuitenkin suorittaa rollbackin mikäli migraatiotiedoston ajamisessa tapahtuu virheitä.

Liquibase[13] on myös java-ohjelma, joka ottaa suoraan yhteyden tietokantaan, mutta on ominaisuuksiltaan hiukan monipuolisempi. Liquibase perustuu ajatukseen, että jokainen migraatiotiedosto on voitava peruuttaa, jolloin on mahdollista palata tietokantaversios-

[12] http://flywaydb.org/
[13] http://www.liquibase.org/

sa myös taaksepäin. Migraatioiden kantamuutokset kirjoitetaan oletusarvoisesti xml:llä, joista osan Liquibase osaa itse peruuttaa. Lopuille on suositeltavaa kirjoittaa itse erillisen peruutustoiminnallisuus, jonka suorittamisen Liquibase hoitaa tarvittaessa tai erillisellä komennolla. Tällöin esimerkiksi kehitysvaiheessa on mahdollista testailla erilaisia muutoksia ja palata aiempiin versioihin ongelmitta. Vaikka Liquibasen migraatiot kirjoitetaan oletusarvoisesti XML:llä, voi ne kirjoittaa myös Javalla, YAML:lla tai JSON:lla.

5. JATKUVA TOIMITTAMISEN TUTKIMUSTULOKSET PROJEKTEISSA

Tässä luvussa esitellään jatkuvaan toimittamiseen liittyvät käytännöt ja kokemukset kolmesta Solita Oy:llä toteutetusta asiakasprojektista. Ensimmäisen kahden projektin tutkimustulokset perustuvat puolistrukturoituihin yksilöhaastatteluihin, jotka nauhoitettiin. Haastateltavat ovat ohjelmistokehittäjiä, joilla on huomattava tekninen vastuu projektissa. Koska yksittäinen haastattelu kesti noin kaksi tuntia, on tähän tutkimukseen koostettu vain jatkuvan toimittamisen kannalta merkittävät kokemukset ja tulokset. Vaikka kolmannen projektin tulokset pohjautuvat diplomityön tekijän omiin kokemuksiin, on tuloksissa pyritty subjektiiviseen näkökulmaan sekä ottamaan kantaa samoihin asioihin kuin henkilöhaastatteluissa.

Haastatteluissa käsiteltiin avoimesti seuraavat asiat järjestyksessä:

1. projektin koko, vaihe ja henkilöstö
2. versionhallinnan käyttö
3. asennusputken toteutus ja toiminta
4. jatkuvan integraation rooli
5. konfiguraatioiden hallinta
6. testauskäytännöt
7. tuotantoasennus
8. nopeus reagoida muutostarpeisiin
9. palautteen saaminen (tekninen, asiakas, loppukäyttäjä)
10. numeerisiä arvoja tuntemuksista (motivaatio, luotto, stressi, laatu, yhtenäisyys).

5.1. Projekti 1: Lupapiste.fi

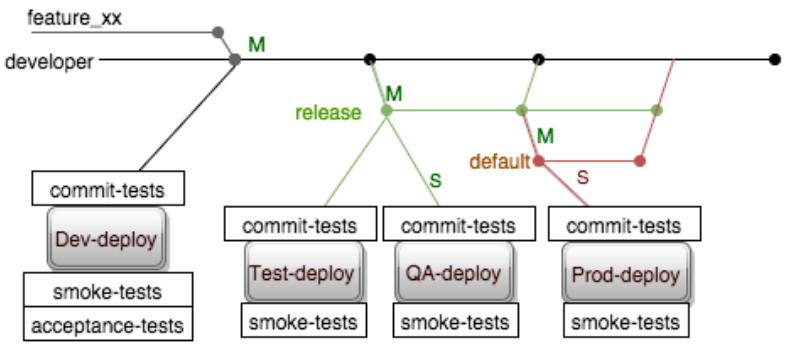
Lupapiste on rakennetun ympäristön sähköinen lupapalvelu, jonka avulla kansalaiset ja yritykset voivat hakea rakennuslupia, yleisten alueiden käyttöön liittyviä lupia ja myöhemmin mm. ympäristölupia. Luvat käsitellään kuntien virastoissa. Kirjoitushetkellä 47 kuntaa on sähköistänyt lupahakemuksensa lupapisteen avulla ja 41 kuntaa on liittymässä palveluun. Palvelun käyttäjät ovat lupia hakevia kansalaisia sekä lupia käsitteleviä kuntien viranomaisia.

Järjestelmä koostuu noin viidestä komponentista ja sisältää ulkoisia integraatioita mm. pankkitunnuskirjautumiseen. Ohjelmointikielinä on kirjoitushetkellä Clojure (45000 riviä) sekä JavaScript (15000 riviä). Projektissa on työskennellyt keskimäärin kuusi kehittäjää ja projektipäällikkö. Asiakkaan puolesta projektissa on mukana kaksi projektipäällikköä. Projekti on ollut tuotannossa kirjoittamishetkellä noin kaksi vuotta – keväästä 2013.

5.1.1. Projektin asennusputken esittely

Projektissa on käytössä Mercurial-versionhallinta ja sovellettuna toimintatapana git-flow [7]. Versionhallintaa pidetään säilönä kaikelle ohjelmakoodille, testeille sekä palvelinten konfiguraatioille. Haaroja käytetään ohjelmistossa työn alla olevien muutosten hallintaan sekä konfiguraatioissa erottamaan tuotantoon asennettavat konfiguraatiot muista ympäristöistä.

Alla on kuvattu projektissa käytössä olevan git-flow:n mukainen asennusputki. M:llä merkatut haarojen yhdistämiset ovat manuaalisia vaiheita ja S:llä merkatut on ajastettu tapahtumaan automaattisesti yöllä. Loput ovat automaattisia.



Kuva 5.1: Asennusputki lupapiste.fi -palvelussa.

Developer-haaran tehdyt muutokset käynnistävät jatkuvan integraation, jossa automaattisesti:

1. Käännetään ohjelmistot.
2. Ajetaan kommitointivaiheen testit (muutama minuutti).
3. Konfiguroidaan kehitysympäristön palvelimet.
4. Päivitetään tietokanta.
5. Asennetaan järjestelmä kehitysympäristöön.
6. Ajetaan savutestit kehitysympäristöä vasten.
7. Ajetaan hyväksyntätestit kehitysympäristöä vasten (noin tunti).

Jokainen muutos aiheuttaa toimitusputkessa noin tunnin kestävän prosessin. Yksikkö- ja integraatiotestit, jotka ovat nopeita, ajetaan kommitointivaiheessa. Kohdepalvelimen konfiguraatioihin tehdään muutoksia mikäli tämä on tarpeellista. Koska hyväksyntätestejä on paljon ja ne ajetaan oikealla selaimella kehitysympäristöä vasten, kestää hyväksyntätestausvaihe itsessään noin tunnin ajan, vaikka testikattavuus ei ole lähelläkään 100% kaikista skenaarioista.

Version julkaisu käynnistetään manuaalisesti mikäli nähdään tarpeelliseksi ja jatkuvan integraation vaihe on läpäisty ongelmitta. Version julkaisun yhteydessä luodaan uusi release-haara ja tapahtuu seuraavaa:

1. Käännetään ohjelmisto.
2. Ajetaan kommitointivaiheen testit (muutama minuutti).
3. Konfiguroidaan testiympäristön palvelimet.
4. Päivitetään tietokanta.
5. Asennetaan järjestelmä testiympäristöön.
6. Ajetaan savutestit testiympäristöä vasten.

Testiympäristöön asennettu versio asentuu samalla prosessilla seuraavana yönä automatisoidusti QA-ympäristöön. Tässä ympäristössä tapahtuu myös kuntien työntekijöiden koulutus palvelun käyttöön.

Kun päätetään tehdä tuotantoasennus, yhdistetään julkaistava haara default-haaraan. Tällöin tuotantoasennus tapahtuu yöllä automaattisesti samalla prosessilla kuin testi- ja QA -ympäristöihin sillä poikkeuksella, että kantapäivitykset ajetaan ensin tuotantokannasta otettua kopiota vasten, jotta varmistetaan tietokannan onnistunut päivitys.

Tuotantoon voidaan poikkeustilanteessa tehdä pieniä muutoksia pikafikseinä myös erottamalla default-haarasta oma haara, tekemällä muutokset sinne ja yhdistämällä takaisin default-haaraan, jolloin muutos asentuu normaalin prosessin mukaisesti tuotantoon yöllä. Asennus voidaan myös käynnistää manuaalisesti haluttaessa.

5.1.2. Haastattelun tulokset

Projektissa on automatisoitu täydellisesti kaikki palvelinten konfigurointiin ja ohjelmiston testaukseen sekä asentamiseen liittyvät vaiheet. Kriittiset portaat vaativat ihmisen manuaalisen käynnistämisen, kuten testi-, QA- ja tuotantoympäristöön asentamisen. Tämä on siis verrattavissa kohdassa 3.6. mainittuun asennuksen porrastettuun ylentämiseen, vaikka ohjelmistot paketoidaankin erikseen jokaiseen ympäristöön asennettaessa.

Mikäli jokaisen muutoksen jälkeen käynnistyvä jatkuvan integraation vaihe on läpäisty, on projektitiimi luottavainen siihen, että järjestelmästä on julkaistavissa versio, joka on toimiva ja asennettavissa tuotantoon. Vaihe epäonnistuu keskimäärin kerran päivässä ja korjautuu tyypillisesti seuraavassa syklissä. Suuri osa epäonnistumisista on kuitenkin vääriä hälytyksiä hyväksyntätesteissä, sillä selaimella ajettavat testit koettiin epästabiileiksi ja epäonnistuvan satunnaisesti ilman oikeaa syytä. Erillistä manuaalista testausvaihetta, jossa palvelun teknistä toimivuutta testattaisiin, ei ole lainkaan.

Ainoita haasteita testauksen automatisoinnissa koettiin selaimella suoritettavista hyväksyntätesteistä. Niiden ajaminen on hidasta, vaatii yhteensopivat versiot sekä selaimesta että ajo-ohjelmistoista, testien ylläpitäminen on vaikeaa ja testit fragiileja. Testit kuitenkin tuovat luottoa siihen että järjestelmä toimii ja regressiosuoja antaa henkistä tukea. Lisäksi joitakin oikeita virheitä on saatu testeillä kiinni, vaikka se harvinaisempaa onkin. Vaikka testien ylläpitoon ja korjailuun joutuu satunnaisesti tunteja työaikaa käyttä-

mään, sen lisäksi että sijoitus koettiin hyödylliseksi, olisi projektitiimi halukas käyttämään enemmänkin aikaa testikattavuuden parantamiseen erityisesti ulkoisten integraatioiden ja käyttöliittymän osalta.

Palvelinten konfiguraatioiden automatisointiin on kulunut huomattavasti aikaa erityisesti projektin alkuvaiheessa, mutta se koettiin investointina eikä mitään jätettäisi pois. Ongelmia joillekin kehittäjille tuottaa Chef-ohjelmisto työkaluna, sillä se on kokemusten mukaan vaikea saada toimimaan Windows-käyttöjärjestelmällä. Konfiguraatioiden automatisoinnista Chef:llä oli kuitenkin hyviä kokemuksia: Mitään yllättäviä ongelmia siitä, että ympäristöissä olisi vääriä konfiguraatioita, tai testi- ja tuotantoympäristössä olisi eroa, ei ole koskaan ollut. Kaikki ympäristöt ovat konfiguroitu samoin ja konfiguraatiot ovat selkeästi dokumentoitu versionhallinnassa. Uusia tunnuksia käyttäjille luodaan viikoittain ja nämä saadaan tuotua välittömästi, hallitusti ja automatisoidusti ilman riskiä virheistä tai käsityötä tuotantopalvelimilla. Kaikki konfiguraatiomuutokset tulee testattua kehitys, testi- ja QA-ympäristöissä ennen tuotantoon vientiä, dokumentaatio on siirtynyt wikistä versionhallintaan ja on aina ajan tasalla.

Koska tuotantoasennukset ovat helppoja ja turvallisia, on niitä tehty kahden vuoden aikana noin kaksi sataa. Tuotantoasennus kestää noin viisi minuuttia, ja tuotantohaaraan yhdistämisen, mikä asennuksen yöllä laukaisee, voi tehdä kuka tahansa kehittäjistä. Asennus on epäonnistunut viimeksi yli vuosi sitten, eli aiemmat noin sata asennusta on onnistunut. Tuotantoasennuksen epäonnistuminen on erittäin epätodennäköistä siksi, että samalla prosessilla asennuksia tehdään muihin tuotantoa vastaaviin ympäristöihin useasti päivässä.

Virheitä tuotannossa on ollut kuitenkin noin sata kappaletta. Suurin osa virheistä liittyykin käyttöliittymään, sillä käyttöliittymätesteillä on vaikea kattaa kaikkia skenaarioita juuri niiden hitauden ja epästabiiliuden vuoksi. Virheisiin voidaan kuitenkin reagoida kriittisyyden mukaan tarvittaessa jopa välittömästi, jolloin korjaus on asennettu tuotantoon puolessa tunnissa. Tyypillisesti korjaus asennetaan normaalin asennusprosessin mukaisesti joko seuraavana yönä tai muutaman päivän sisällä. Joskus virhe on ilmennyt vasta tuotannossa siksi, että muissa ympäristöissä ei ole täysin tuotantoa vastaavaa dataa. Virheet eivät stressaa kehittäjiä – ne ovat normaali asia ja saadaan korjattua ja asennettua nopeasti.

Vaikka tuotantoasennuksen voisi teknisesti tehdä uusien muutosten myötä joka yö, on projektissa käytössä toimintamalli, jossa projektipäällikkö hyväksyntätestaa tuotantoon asennettavan version QA-ympäristössä. Tämä manuaalinen vaihe estää sen, ettei tuotantoasennusta voida tehdä päivittäin. Projektipäälliköllä ei ole aikaa tehdä testausta päivittäin, joten asennuksia päädytään tekemään pääsääntöisesti silloin, kun on jotain näkyvää ja hyödyllistä, jota loppukäyttäjille saadaan tuotua. Asennuksia ei myöskään tehdä mielellään päivisin, sillä asennus aiheuttaa parin minuutin käyttökatkon palvelussa.

Jatkuvassa toimittamisessa koettiin haasteena käyttäjien tiedottaminen uusista ominaisuuksista. Toisinaan päivityksissä pelkästään painikkeiden siirtyminen käyttöliittymässä paikasta toiseen on aiheuttanut viestejä palvelutukeen. Lisäksi kyseessä ollessa monikielinen julkinen palvelu, on ruotsinkielisiä käännöksiä hankala saada sitä mukaa kun asennuksia tehdään.

Haastateltava antaisi pisteitä laadulle sekä turvallisuuden tunteelle tässä projektissa 4,25/5. Pienet virheet käyttöliittymässä sekä joskus tuotantoon valunut rikkova data ovat ainoat asiat, jotka laskivat pisteitä.

Motivaatiosta haastateltava antoi täydet 5/5. Motivaatioon vaikutti positiivisesti erityisesti hyvät teknologiavalinnat, uuden oppiminen ja suurempi vastuu koko järjestelmästä ja sen toimittamisesta, kun vastuuta ei pakoilla.

Stressittömyydestä pisteitä annettiin vain 2/5. Tyypillisten aikataulupaineiden ja monisyisen projektin lisäksi stressiä lisäsi normaalia suurempi vastuu. Kehittäjillä on vastuu koko palvelinten infrastruktuurista aina ohjelmistojen versioista Linuxin tietoturvapäivityksiin asti.

5.1.3. Analyysi

Jatkuvan toimittamisen käytännöt ohjaavat projektin päivittäistä työtä ja niistä koetaan saavan selkeästi hyötyä. Jatkuva integraatio ja testiautomaatio koettiin erittäin hyödylliseksi, lisäävän laatua, tuovan henkistä tukea ja ehkäisevän virheitä. Vaikka testit eivät kehittäjän mukaan tuntuneet saavan virheitä kiinni useasti, ovat tuotantoon eksyneet virheet selkeästi juuri sillä alueella jossa testikattavuuskin on heikoin. Testauskäytäntöihin menee paikoitellen huomattavasti aikaa, mutta siihen panostetun ajan koetaan olevan niinkin arvokasta, että testien tuottamiseen ja parantamiseen olisi tahtoa käyttää enemmänkin ai-

kaa. Tuotteen laadun ja projektitiimin luottavuuden takaamiseksi jatkuvaan integraatioon ja testiautomaatioon on siis hyvä varata reilusti aikaa ja pitää tätä jatkuvana toimintamallina koko ohjelmistoprojektin ajan.

Ohjelmistosta on tehtävissä tuore tuotantoasennus lähes koko ajan, ja tästä syystä päivityksiä tehdäänkin viikoittain. Vaikka virheitä silti tuotannosta löytyy, eivät korjaukset ole suuri ongelma, sillä ne saadaan tuotantoon tarpeen vaatiessa heti tai viimeistään muutamassa päivässä – ilman erityistä työtä tai ongelmia. Ainoa tuotantoasennusta hidastava tekijä on manuaalinen projektipäällikön tekemä hyväksyntätestaus, mikä on vain muodostunut käytännöksi. Kaikkia manuaalisia vaiheita tulisi siis välttää, mikäli toimittaminen halutaan pitää jatkuvana. Virheitä ja muutostarpeita tulee aina jokaisen ohjelmistoprojektin tuotantovaiheessa. Projekti on hyvä saada pidettyä tuotantoasennettavana koko ajan, jolloin tarpeisiin voidaan reagoida nopeasti, eivätkä virheet tai muutostarpeet ole ongelma, vaan normaalia päivittäistä työtä. Lisäksi katkoton asennus olisi hyvä saada toteutettua esimerkiksi ohjelmiston hajautuksella, sillä asennukset voivat ajoittua helposti yöhön, mikäli päivitys aiheuttaa käyttökatkon.

Mitään ongelmia siitä, että palvelimet tai ohjelmistot olisi konfiguroitu väärin, ei ole koskaan ollut. Vaikka konfiguraatioiden automatisointiin kuluu alkuvaiheessa aikaa ja työkalujen kanssa voi olla joillakin käyttöjärjestelmillä haasteita, on panostus erittäin perusteltua. Tällöin voidaan välttää epämääräisiä ja vaikeasti selviteltäviä ongelmia, joita ilmenee satunnaisissa ympäristöissä – pahimmillaan juuri tuotannossa. Sen lisäksi, että epämiellyttäviä virheitä vältetään ja järjestelmän laatu pysyy hyvänä kun konfiguraatiot automatisoidaan, saadaan korjauksiin ja selvittelyyn käytettävä aika kohdistettua tähän automatisointiprosessiin projektin alkuvaiheessa, jolloin projektissa työtahti on usein muutenkin tehokkaampi.

Myös siitä, että tuotantoasennuksia tehdään säännöllisesti, seurasi haasteita uusista ominaisuuksista tiedottamisessa sekä tekstien kielikäännöksistä. Mikäli kielikäännöksiä tarvitaan, on hyvä olla sovittu prosessi, jolla käännökset – tai ainakin väliaikaiskäännökset – saadaan käyttöön pikaisesti. Uusista ominaisuuksista tai muutoksista voi olla mahdoton informoida käyttäjiä oikein niin, että käyttäjät lukisivat tiedotteet, joten perustellumpaa voisi olla panostaa käyttöliittymän intuitiivisuuteen niin, ettei sen käyttäminen vaadi ylläpidettävää ohjeistusta.

Jatkuvan toimittamisen käytännöt vaikuttivat positiivisesti myös työntekijöiden motivaatioon. Pelkästään nykyaikaiset teknologiavalinnat ja käytännöt sekä uuden oppimisen kautta itsensä kehittäminen koettiin motivoivana. Lisäksi vastuu koko järjestelmäkokonaisuudesta lisäsi motivoituneisuutta sen sijaan, että tiimillä olisi vain oma vastuualueensa osana isompaa kokonaisuutta. Toisaalta suuri vastuu taas lisäsi stressiä.

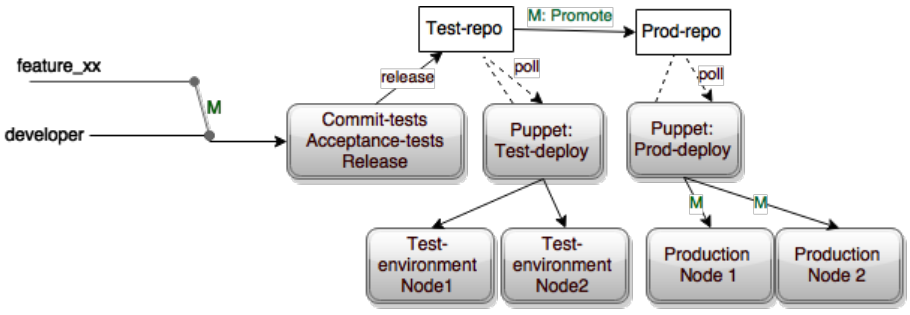
5.2. Projekti 2: YleX.fi

YleX-radiokanavan verkkopalvelun uudistus, jonka tavoitteena on muuttaa vanhentunut sivusto mobiiliyhteensopivaksi, parantaa palvelua ja tuoda sosiaalista kulmaa foorumien, käyttäjäprofiilien ja käyttäjien välisen vuorovaikutuksen avulla.

Järjestelmä koostuu itse ohjelmistosta sekä käyttöliittymäkerroksesta. Ulkoisia integraatioita ovat sisäänkirjautumistoiminnallisuus sekä useammat muut palvelut, joista haetaan tietoa käyttäjien nähtäville http-rajapintojen yli. Ohjelmointikielenä on Scala (6600 riviä) sekä JavaScript (7450 riviä). Tietokantana on dokumenttikanta MongoDB. Projektissa työskenteli kehitysvaiheessa ohjelmistokehitystä tekevä projektipäällikkö, kaksi kehittäjää sekä yksi spesialisti hoitamassa integraatioita ulkoisiin sisällönhallintapalveluihin. Asiakkaan puolelta mukana on ollut graafinen suunnittelija, projektipäällikkö sekä digituottaja. Projekti on ollut tuotannossa kirjoittamishetkellä noin puoli vuotta – joulukuusta 2014.

5.2.1. Projektin asennusputken esittely

Projektissa on käytössä git-versionhallinta. Versionhallinta on säilönä kaikelle ohjelmakoodille, testeille sekä palvelinten konfiguraatioille. Uusia ominaisuuksia ja muutoksia varten tehdään oma haara, joka yhdistetään valmistumisen jälkeen päähaaraan.



Kuva 5.2: Asennusputki ylex.fi -palvelussa.

Kuvassa 5.2 on esitelty projektissa käytössä oleva asennusputki. Manuaaliset vaiheet on kuvattu M:llä, muut ovat automaattisia. Sekä testi- että tuotantoympäristön ohjelmistot on kahdennettu, joten asennus on tehtävä kahdelle palvelimelle. Päähaaraan tehdyt muutokset käynnistävät jatkuvan integraation, jossa automaattisesti:

1. Käännetään ohjelmistot.
2. Ajetaan kommitointivaiheen testit ja hyväksyntätestit (muutama minuutti).
3. Julkaistaan versio testiympäristön artefaktien säilöön.
4. Testiympäristön päivitysvaihe huomaa muutokset ja päivittää testiympäristön palvelimet sekä konfiguraatioiden että ohjelmiston osalta.

Jokainen muutos aiheuttaa toimitusputkessa vain muutaman minuutin kestävän prosessin. Hyväksyntätestit eivät ole oikeita selaimella tehtäviä testejä, vaan pelkästään integraatiotestejä palvelun sisäisen loogisen toiminnan testaamiseen. Tästä syystä vaihe kestää hyvin vähän aikaa. Testiympäristöä käytetään projektissa sekä manuaaliseen hyväksyntätestaukseen että demotarkoitukseen.

Testiympäristöön asentuneet versiot voidaan ylentää tuotantoympäristöön. Tämä vaihe on manuaalinen ja täytyy suorittaa erillisellä palvelimella. Tuotantoasennettaessa julkaistu versio siis:

1. Ylennetään manuaalisesti tuotantoasennettavaksi.
2. Käynnistetään manuaalisesti päivitys tuotantoympäristön palvelimille palvelin kerrallaan.
3. Palvelimet konfiguroidaan ja ohjelmisto päivittyy automatisoidusti.

Kaikki muutokset tuotantoon siirtyvät saman prosessin läpi niin, että sama kerran käännetty versio ohjelmistosta on asennettu ensin testiympäristöön ennen tuotantoasennusta. Koska asennuspaketti sekä -prosessi ovat prosessin manuaalista käynnistämistä lukuun ottamatta identtisiä, on tuotantoasennuksen epäonnistuminen hyvin epätodennäköistä.

5.2.2. Haastattelun tulokset

Projektissa on automatisoitu ohjelmiston kääntämis-, testaus- ja asennusprosessi sekä konfiguraatiot. Ainoa manuaalinen vaihe tuotantoasennuksessa on halutun version ylentäminen tuotantoasennettavaksi (kohta 3.6.), sekä automatisoidun asennuksen käynnistäminen.

Mikäli jatkuvan integraation vaihe on onnistunut ja järjestelmä asentunut testiympäristöön, luottaa kehitystiimi siihen, että version voisi asentaa myös tuotantoon. Vaihe epäonnistuu vain satunnaisesti, eli pääsääntöisesti tuotantoasennusvalmius tuoreimmasta versiosta on aina. Suurin osa epäonnistumisista on vääriä hälytyksiä, jotka johtuvat esimerkiksi verkkoyhteysongelmista jatkuvan integraation palvelimilla tai ohjelmakoodin syntaksia tarkastelevista työkaluista, jotka suoritetaan vaiheen yhteydessä.

Ongelmia tuo asennusputkessa säännöllisesti se, että kaikki asennusputken palvelimet ja ympäristöjen ylläpito on asiakkaan hallinnassa. Jatkuvan integraation palvelinten ohjelmistot (Bamboo) ja näiden konfiguraatiot aina verkkoyhteyksineen asti ovat asiakkaan vastuulla, joten näissä ilmenevät ongelmat lamauttavat satunnaisesti kehitystyötä. Jossain tilanteissa esimerkiksi jatkuvan integraation vaiheen epäonnistumista ei otettu vakavasti, sillä sen tiedettiin johtuvan asiakkaan verkko-ongelmista.

Päähaara, joka asennusputken läpi menee, pidetään aina tuotantoasennusvalmiina. Uudet ominaisuudet yhdistetään siihen vasta valmistumisen jälkeen. Pisimmillään yhdistämistä ei tehdä kehittäjän toimesta kahteen viikkoon, mikä tarkoittaa sitä, että työn alla olevat muutokset eivät pitkään aikaan mene asennusputken läpi. Tyypillisesti uudet muutokset yhdistetään kuitenkin alle viikossa. Jatkuvan integraation vaihe ajetaan lisäksi kaikille kehityshaaroillekin niin, että ohjelmisto käännetään ja sitä vasten ajetaan testit ilman että pakettia asennetaan eteenpäin. Jatkuvan integraation vaihe sekä koko asennusputki ohjaakin ohjelmistokehitystä niin, että virheet korjataan välittömästi ja jatkuvan integraation vaihe onkin virheellisessä tilassa korkeintaan muutaman tunnin.

Vaikka asennusputkessa on vaihe automaattiseen testaukseen ja testejä kirjoitetaan aktiivisesti, puuttuu palvelusta kokonaan järjestelmätestit, käyttöliittymätestit ja käyttöskenaarioita suorittavat hyväksyntätestit. Projektissa on keskitytty vain palvelun taustalogiikan testaamiseen yksikkö- ja integraatiotestein. Järjestelmätestien ja käyttöliittymätestien puuttumista ei pidetty kriittisenä ongelmana, mutta niiden olemassaolosta olisi koettu saa-

van hyötyä. Testejä kuitenkin on järjestelmän taustalogiikalle varsin kattavasti ja niiden toteuttamiseen kuluneen ajan uskotaan maksavan itsensä takaisin.

Käyttöliittymätestien puuttumisesta johtuen voimakasta luottoa siihen, että testit takaisivat järjestelmän toiminnan, ei ole. Ennen tuotantoasennusta palvelun toiminnallisuutta käydään käyttöskenaarioittain läpi testiympäristössä, jotta kehittäjät saavat tunteen siitä, että palvelu toimii virheettömästi. Usein myös asiakas suorittaa hyväksyntätestausta testiympäristössä ennen tuotantoasennusta. Mitään testaussuunnitelmaa tai ohjeistusta ei kuitenkaan ole, eli manuaalinen testaus on ketterää ja kevyttä sekä perustuu testaajan omaan näkemykseen siitä mitä on hyvä testata.

Vaikka testauksen koettiin olevan kohtuullisen hyvällä tasolla, olisi siihen haastateltavan mielestä voitu panostaa enemmänkin. Kun testejä on jollekin komponenteille kirjoitettu jälkikäteen, on näistä toiminnallisuuksista löytynyt virheitä testien kirjoittamisen yhteydessä, jotka eivät ole kuitenkaan vielä realisoituneet tuotannossa. Vaikka testien kirjoittamiseen menee aikaa, koettiin niiden tuovan turvaa ja olevan tärkeä sijoitus.

Palvelinten konfiguraatioiden automatisoinnissa on ollut alkuvaiheessa melko iso työ ja Puppet-työkalun opetteluun on mennyt aikaa. Monessa tilanteessa konfiguraatioiden muuttaminen suoraan palvelimelle olisi suoraviivaista, kun muutosten tekeminen Puppet:lla on monimutkaisempaa ja kestää enemmän aikaa. Lisäksi Puppet koettiin työkaluna paikoitellen hankalaksi ja työlääksi: Erillinen Puppet-master -palvelin lisää monimutkaisuutta ja työkalun antamat virheilmoitukset ovat usein epäselviä.

Konfiguraatioiden hallinnasta Puppet-työkalulla koettiin kuitenkin saavan selkeää hyötyä. Muutokset saadaan helposti testattua testiympäristössä ja siirrettyä sitten tuotantoon. Koska konfiguraatiot ovat versionhallinnassa, tiedetään aina, mikä jokaisen ympäristön tila on nykyisellä hetkellä. Uuden ympäristön saa helposti luotua tyhjästä, mikäli ohjelmiston klustereita on tarve lisätä. Hyötyä koettiin saavan erityisesti hajautuksesta: Myös MongoDB -tietokanta on hajautettu usealle palvelimelle ja sen konfigurointi sekä ylläpito varmuuskopioineen olisi hyvin työlästä ilman konfiguraatioidenhallintatyökalua. Muutoksia tuotannon konfiguraatioihin tulee noin kerran kuukaudessa, ja niiden asentaminen hallitusti testiympäristöön sekä tuotantoon Puppet:lla kestää vain muutaman minuutin.

Vaikka kehitystiimi vastaa itse pääsääntöisesti palvelinten konfiguraatioista Puppet:n avulla, tekee asiakkaan työntekijät joitakin konfiguraatiomuutoksia ja ohjelmistopäivityk-

siä palvelimelle ilmoitusluontoisesti. Täyttä hallintaa siitä, missä tilassa palvelimet ovat, ei siis kehitystiimillä kuitenkaan ole.

Tuotantoon menon jälkeen tuotantoasennuksia on tehty puolessa vuodessa noin kolmekymmentä, eikä yksikään asennus ole epäonnistunut. Tuotantoasennus kestää noin kaksi minuuttia, eikä siitä synny käyttäjille käyttökatkoa, sillä ohjelmistopalvelin on kahdennettu. Tuotantoasennus on identtinen testiasennuksen kanssa sillä eroavaisuudella, että asennus käynnistetään manuaalisesti. Porrastetun ylennyksen ja asennuksen voi tehdä kuka tahansa kehittäjä, jolla on pääsy palvelimelle ja tietää vaadittavan salasanan. Tuotantoasennus koettiin helpoksi, nopeaksi ja turvalliseksi. Asennuksen jälkeen on kerran ilmennyt ohjelmointivirhe, minkä vuoksi aiempi versio on täytynyt palauttaa. Ongelmia ilmetessä vanhemman version palauttaminen on helppoa, vaikka se täytyykin tehdä komentoriviltä manuaalisesti.

Virheitä tuotannossa on ollut noin kymmenen kappaletta. Näiden korjaus ja asennus tuotantoon on kestänyt alle tunnista kahteen päivään riippuen siitä, onko virheeseen tarve ja mahdollisuus reagoida nopeasti. Mikäli virhe on kriittinen ja kehittäjät ovat saatavilla, voidaan päivitys tehdä heti kun korjaus on tehty ja testattu. Koska korjaukset tehdään päähaaraan, voi korjauksen mukana asentua muitakin muutoksia. Tästä ei ole ongelmia, sillä kehittäjät luottavat siihen että päähaara on tuotantoasennuskelpoinen. Tämä mainittiin kuitenkin haasteena, sillä päähaarassa on aina oltava tuotantoasennettavaa koodia.

Vaikka testi- ja tuotantoympäristö ovat konfiguraatioiltaan hyvin vastaavat, on näissä eroavaisuuksia esimerkiksi prosessorien määrän suhteen. Tästä johtuen tuotantoon on kerran asentunut versio, jossa realisoitui rinnakkaisuusvirhe vasta tuotantoympäristössä.

Huolimatta siitä, että tuotantoasennus on helppo ja nopea tehdä, ei asennuksia kuitenkaan tehdä päivittäin tai viikoittain. Vaikka tuotantoasennus on haastateltavan mukaan helppo ja nopea, koetaan ainoa manuaalinen vaihe sen verran työlääksi, ettei asennuksia tehdä ilman erityistä syytä. Lisäksi asennuksessa koetaan aina olevan pieni riski siitä, että jotain hajoaa.

Haastateltava antaisi pisteitä laadulle 4/5. Laadun koettiin olevan varsin erinomainen, sillä virheitä ja ongelmia ei tuotannossa tyypillisesti ole. Kriittisiä suorituskykyongelmia ilmeni tuotantoonmenovaiheessa, kun järjestelmään saatiin lopulta tuotantodataa.

Turvallisuuden tunteelle siitä, että kaikki sujuu, antoi haastateltava 5/5, mutta mainitsi

aikataulun suhteen 4/5. Ainoat ongelmat tähän liittyen olivat projektin aikataulu-, projektinhallinta ja budjettiongelmia, mutta tekniset haasteet eivät tuntuneet vaikuttavan.

Motivaatiosta haastateltava antoi 4/5. Motivaatiota lisäsi modernit teknologiat ja uudet opeteltavat asiat, sekä se, että itse toteutettava palvelu on mielenkiintoinen. Stressittömyydestä pisteitä annettiin 4/5.

5.2.3. Analyysi

Vaikka projektissa ei ole alun perin pyritty jatkuvaan toimittamiseen, sisältävät kehittäjien valitsemat toimintamallit paljon jatkuvalle toimittamiselle tyypillisiä käytäntöjä ja toimintatapoja. Kauimpana jatkuvan toimittamisen käytännöistä olivatkin testauskäytännöt: Järjestelmä- ja käyttöliittymätestit puuttuivat kokonaan. Kun noin puolet ohjelmakoodista liittyy käyttöliittymään, on selvää, että testien puuttuminen tältä osa-alueelta on suuressa roolissa kun puhutaan siitä, miten testit takaavat järjestelmän virheettömän toiminnan. Näiden puuttumisesta johtuen voimakasta luottoa siihen, että testit takaisivat järjestelmän toiminnan, ei ole. Luotto siihen, että tuotantoasennus voidaan aina tehdä päähaarasta, perustuukin lähinnä luottoon kollegoihin ja itseensä. Jos muutoksia on yhdistetty päähaaraan, luotetaan siihen, että ohjelmiston toiminnallisuus on myös kehittäjän puolesta testattu ja tuotantoasennettavissa.

Vaikka haastateltava mainitsi useaan otteeseen sen, että testejä voisi olla enemmänkin, oli hän kuitenkin erittäin tyytyväinen nykyiseen laatuun. Testien puutteellisuudesta huolimatta tuotantoasennuksen uskaltaa tehdä tarvittaessa milloin tahansa. Tätä selkeästi edesauttaa haastattelun perusteella kaksi asiaa: Yhteinen käytäntö siitä, että päähaara on aina tuotantoasennettavissa, sekä tuotantoasennuksen helppous. Lisäksi kriittisemmät, esimerkiksi tietoa tuhoavat virheet, piilevät tyypillisesti järjestelmän taustalogiikassa, jolle testejä kehityksen yhteydessä projektissa on kirjoitettu. Jos virheitä tuotantoon kuitenkin asentuu, on korjausten asentaminen tuotantoon helppoa, luotettavaa ja nopeaa. Virheiden lipsahtaminen tuotantoon ei siis ole kriittinen asia, mikäli ne eivät tuhoa tietoa. Jossain projekteissa testikattavuudesta tinkiminen jollain osa-alueilla voi siis olla perusteltua, mikäli virheiden ilmeneminen ei aiheuta liiketaloudellista haittaa, niihin voidaan reagoida tarvittaessa nopeasti ja korjausten asentaminen tai aiemman version palauttaminen tuotantoon on nopeaa sekä turvallista.

Huolimatta siitä, ettei projektissa ole lähtökohtaisesti tarkoitus tehdä tuotantoasennuksia ilman erityistä tarvetta, ilmeni haastattelussa muutama asennuksia harventava toimintatapa. Pieni, mutta useamman vaiheen sisältävä manuaalinen työ, joka tuotantoasennuksen yhteydessä täytyy tehdä, oli haastateltavan mukaan riittävä vähentämään välttämättömiä tuotantoasennuksia. Lisäksi koettiin olevan aina pieni riski siitä, että asennuksessa menee jotain pieleen. Asennus ei kuitenkaan ole epäonnistunut kertaakaan ja edellinen versio on täytynyt palauttaa vain kerran – sekin onnistuneesti ja helposti. Tuotantoasennus on siis tärkeää tehdä projektissa äärimmäisen helpoksi – niin, että päivityksiä voidaan tehdä jopa ilman erityistä syytä. Mikäli tuotantopäivitys on helppoa, voidaan sitä tehdä usein, jolloin siitä tulee arkipäiväinen toimenpide. Tällöin voidaan olettaa, että pienetkin muutokset asennetaan aiemmin tuotantoon, niistä saadaan palautetta ja niiden toiminta tulee varmistettua mahdollisimman aikaisessa vaiheessa sen sijaan, että muutoksia tulee suuri määrä kerralla.

Ongelmia siitä, että palvelimet tai ohjelmistot olisi konfiguroitu tuotannossa väärin, ei ole ollut. Vaikka haastateltava sanoi Puppet-työkalun ja konfiguraatioiden automatisoinnin opetteluun kuluvan aikaa ja monien konfiguraatiomuutosten olevan näin hitaampia ja monimutkaisempia, nähtiin hyödyt tässä toimintatavassa ilmeisinä. Hyötyä koettiin saavan erityisesti siksi, että sekä ohjelmisto että tietokannat on hajautettu. Konfiguraatioiden automatisointia on siis hyvä harkita viimeistään silloin, jos järjestelmä sisältää hajautettuja komponentteja. Tällöin palvelinten konfiguraatiot on dokumentoitu ja mahdolliset muutokset täytyy tehdä vain yhteen paikkaan, josta ne asentuvat automaattisesti kaikille samaan konfiguraatiopohjaan kuuluville palvelimille.

Haastattelussa tuli useassa vaiheessa ilmi haasteita, jotka johtuivat siitä, että sekä asennusputken operatiiviset palvelimet että järjestelmän omat palvelimet ovat asiakkaan hallinnassa. Ainoat konfiguraatioihin liittyvät ongelmat ovat johtuneet juuri niistä palvelimista, jotka eivät ole täysin automatisoinnin piirissä ja omassa hallinnassa. Koska asennusputken voidaan ajatella olevan jatkuvan toimittamisen selkäranka, on sen konfigurointiin ja hallintaan oltava täydet mahdollisuudet jotta voimakkaasta työkalusta ei tule kehityksen pullonkaula.

Testiympäristöjen olisi hyvä olla myös suorituskyvyltään identtisiä tuotantoympäristön kanssa [4, s.217]. Haastateltava kertoikin rinnakkaisessa suorituksessa ilmenneestä

virheestä, joka olisi mahdollisesti realisoitunut jo testiympäristössä ennen tuotantoa, mikäli prosessorien lukumäärä ei olisi testiympäristössä asetettu yhteen. Ympäristöjä pystytettäessä on siis hyvä huomioida, että testiympäristön pitäminen suorituskyvyltään heikompana kuin tuotantoympäristö voi myös aiheuttaa ongelmia, joiden toistaminen testiympäristössä saattaa olla mahdotonta.

Kuten aiemmassa haastattelussa (kohta 5.1., myös tässä haastateltava mainitsi modernien teknologioiden ja uusien opeteltavien asioiden lisäävän motivaatiota. Jatkuvan toimittamisen käytäntöjen sisällyttäminen projektiin uutenakin asiana voidaan olettaa tuovan myös huomattavia välillisiä etuja yritykselle sekä projektille. Stressitaso ei haastateltavalla ollut suuresta vastuusta huolimatta korkea. Tähän voi vaikuttaa positiivisesti se, että palvelinten ylläpito on myös asiakkaan vastuulla ja projektitiimi vastaa vain itse järjestelmän ylläpidosta ja konfiguraatioista.

5.3. Projekti 3: Työajankirjausjärjestelmä

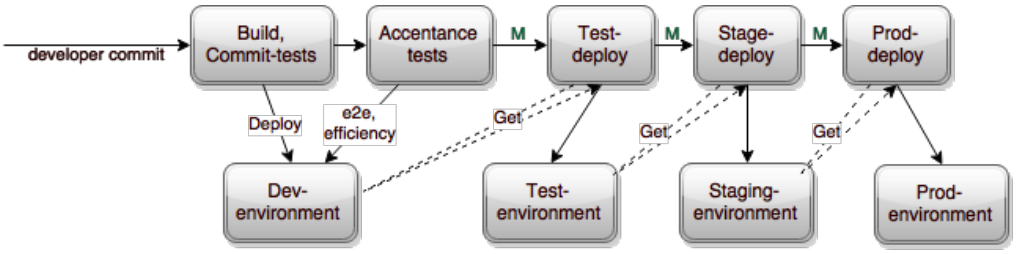
Projektissa toteutettiin asiakasorganisaatiolle räätälöity työajankirjausjärjestelmä. Asiakas oli ottamassa uutta HR-tietojärjestelmää käyttöön, jonka ominaisuudet työajankirjauksissa ja niiden kohdistamisissa eivät olleet riittäviä. Toteutettu ohjelmisto toimii tämän järjestelmän esijärjestelmänä niin, että muun muassa kirjauskohteet, henkilöt, työsuhteet ja poissaolot saadaan ulkoisista integraatioista. Itse työajankirjaus ja kirjausten kohdennus tapahtuu toteutetussa palvelussa, josta ne siirtyvät taas eteenpäin muihin järjestelmiin.

Järjestelmä koostuu itsessään vain yhdestä sovelluksesta ja sisältää ulkoisia integraatioita useisiin muihin järjestelmiin. Ohjelmointikielenä on kirjoitushetkellä Java (ohjelmakoodia 16000 riviä, testejä 13000 riviä), sekä JavaScript (ohjelmakoodia 3000 riviä, testejä 700 riviä). Projektissa on työskennellyt Solitan puolesta yhtäaikaisesti keskimäärin kaksi kehittäjää sekä projektipäällikkö. Asiakkaan puolelta päävastuussa on kaksi projektipäällikköä, mutta mukana on ollut myös eri rooleissa toimivia työntekijöitä. Projekti on ollut kirjoitushetkellä tuotannossa noin puoli vuotta, jonka aikana on otettu käyttöön suurimmat integraatiot.

5.3.1. Projektin asennusputken esittely

Projektissa on käytössä Git-versionhallinta. Pääsääntöisesti kaikki kehitystyö tehdään päähaaraan niin, että keskeneräiset ominaisuudet on piilotettu käyttöliittymästä tuotannossa.

Alla on kuvattu projektin asennusputki. M:llä merkityt vaiheet vaativat manuaalisen käynnistämisen, muut ovat automaattisia.



Kuva 5.3: Asennusputki tuntikirjaussovelluksessa.

Päähaaran tehdyt muutokset käynnistävät jatkuvan integraation, jossa automaattisesti:

1. Käännetään ohjelmistot.
2. Ajetaan kommitointivaiheen testit (muutama minuutti).
3. Päivitetään tietokanta.
4. Asennetaan järjestelmä kehitysympäristöön.
5. Ajetaan savutestit kehitysympäristöä vasten.
6. Ajetaan hyväksyntätestit kehitysympäristöä vasten (muutama minuutti).

Jokainen muutos aiheuttaa toimitusputkessa alle vartin kestävän prosessin. Yksikkö- ja integraatiotestit, jotka ovat nopeita, ajetaan kommitointivaiheessa. Palvelinten konfigurointia ei tehdä automaattisesti, eli asennus kattaa pelkästään sovelluksen binäärin ja tietokannan päivittämisen. Osa applikaation konfiguraatioista tulee asennuspaketin mukana, mutta ympäristökohtaiset konfiguraatiomuutokset on tehtävä käsin jokaiseen ympäristöön. Vaikka hyväksyntätestit ajetaan oikealla selaimella kehitysympäristöä vasten, kestää hyväksyntätestausvaihe vain muutaman minuutin, sillä näillä testeillä ei kateta lähelläkään kaikkia palvelun käyttöskenaarioita. Suorituskykytestit ajetaan sovellukselle kehitysympäristöä vasten joka yö, jotta suorituskyvyn mahdollista heikkenemistä voidaan seurata.

Jokainen kehitysympäristöön asentunut versio on julkaisuehdokas, joka voidaan ylentää asennettavaksi testiympäristöön. Testiympäristöön asentaminen tapahtuu Jenkinsistä yhdellä painikkeella, joten sen voi tehdä selaimella kuka tahansa. Testiympäristöön asennettaessa tapahtuu seuraavaa:

1. Haetaan asennettava asennuspaketti kehitysympäristön palvelimelta.
2. Päivitetään testiympäristön tietokanta.
3. Asennetaan järjestelmä testiympäristöön.
4. Ajetaan savutestit testiympäristöä vasten.

Staging-ympäristöön asentaminen tapahtuu erilliseltä deploy-palvelimelta komentoriviltä, johon vain osalla kehittäjistä on pääsy. Asentamista varten on suoritettava skripti, joka suorittaa täysin saman asennusprosessin kuin testiympäristöön asentaminen sillä eroavaisuudella, että paketti haetaan testiympäristöstä. Tuotantoympäristöön asentaminen tapahtuu deploy-palvelimelta toisella skriptillä, joka taas hakee asennettavan paketin staging-ympäristöstä.

Kaikki kohdeympäristöön asennettujen versioiden asennuspaketit pidetään säilössä kohdeympäristön palvelimella. Aiemman version palautus täytyy tehdä sovelluspalvelimella palautettavan version asennuspaketin mukana tulevalla asennusskriptillä.

5.3.2. Käytännöt ja tulokset

Projektissa on automatisoitu kaikki sovelluksen testaamiseen sekä asentamiseen liittyvät vaiheet. Kriittiset portaat, eli kaikkiin muihin paitsi kehitysympäristöön asentaminen, vaativat manuaalisen käynnistämisen. Sovellusta kehitetään, testataan, ylennetään ja asennetaan hyvin pitkälti jatkuvan toimittamisen käytäntöjen mukaisesti.

Suurin puute on kuitenkin palvelinten ja konfiguraatioiden hallinnassa, joita ei ole mitenkään automatisoitu. Tämä johtuu siitä, että palvelimet ovat asiakkaan ympäristössä, eikä palveluntoimittajalle ole myönnetty hallintaoikeuksia näille palvelimille. Tästä seurasi ongelmia erityisesti projektin alkuvaiheessa, kun ympäristöjä luotiin. Ohjeistuksesta huolimatta asiakas ei pystynyt itse alustamaan palvelimia sovelluksen vaatimilla asetuksilla ja väliohjelmistoilla. Virheellisistä konfiguraatioista johtuen sovellus ei esimerkiksi kerran käynnistynyt palvelimen uudelleenkäynnistämisen yhteydessä.

Lisäksi haasteita tuo se, että ympäristökohtaisten konfiguraatioiden tilaa ei näe muualta kuin sovelluspalvelimella olevista konfiguraatiotiedostoista. Koska osa uusista ominaisuuksista tuodaan käyttöön keskeneräisinä ja niitä hallitaan konfiguraatioissa ominaisuuskytkimillä, ei kehittäjillä ole varmuutta siitä, mikä kytkinten tila on missäkin ympäristössä. Tämä selviää vain kirjautumalla kohdepalvelimelle ja tarkastamalla konfiguraatioista. Kriittisiä ongelmia tästä ei kuitenkaan ole seurannut, sillä sovellus on ympäristökohtaisesti vain yhdellä palvelimella, joten konfiguraatiot eivät ole monimutkaisia eikä näihin tule muutoksia usein.

Projektitiimi on hyvin luottava siihen, että jatkuvan integraation vaihe sekä selaimella ajettavat hyväksyntätestit takaavat sovelluksen toiminnan. Testikattavuus on Java-ohjelmakoodissa noin 80%. Vääriä hälytyksiä oli erityisesti alkuvaiheessa todella paljon juuri selaimella suoritettavien hyväksyntätestien vuoksi, jotka ovat fragiileja. Näiden robustiuden eteen on kuitenkin tehty töitä, eikä jatkuvan integraation ja hyväksyntätestauksen vaihe epäonnistu tyypillisesti edes päivittäin, sillä selaintestien ajaminen on saatu kohtuullisen luotettaviksi ja kehittäjät ajavat kaikki testit omalla työasemalla ennen muutosten työntämistä versionhallintaan. Hyväksyntätestien paikoittaisen epästabiiliuden vuoksi uusien selaimella ajettavien testien kirjoittaminen on kuitenkin lopetettu, sillä väärät hälytykset aiheuttavat paikoitellen huomattavaa työtä ja palvelun perustoiminnallisuus on olemassa olevilla testeillä katettu suhteellisen hyvin. Selaimella ajettavista testeistä saadaan kuitenkin hyvä luotto siitä, että perustoiminnallisuus on kunnossa. Lisäksi selaintesteillä on saatu kiinni tilanteita, joissa käyttöliittymän isompien muutosten yhteydessä on rikottu huomaamatta nykyistä toiminnallisuutta.

Huolimatta siitä, että tuotantoasennukset ovat nopeita, helppoja ja turvallisia, on niitä tehty puolen vuoden aikana vain 12. Vaikka asennus kestää minuutin ja on teknisesti vain yhden skriptin ajaminen, on pullonkaulana asiakkaan sisäinen byrokratia sen suhteen, kuka ja milloin asennuksia saa tehdä. Asennukseen täytyy sopia erillinen ajankohta ja se täytyy tehdä asiakkaan tiloista, vaikka mitään teknistä tarvetta tälle ei ole. Lisäksi päivitys aiheuttaa noin minuutin käyttökatkon. Asennus ei ole koskaan epäonnistunut, sillä sama asennuspaketti on asennettu kolmeen ympäristöön vastaavalla prosessilla ennen tuotantoa.

Mitään erityistä testausvaihetta ei ennen tuotantoasennusta ole. Kehitystiimi on vastuussa itse ohjelmiston testaamisesta sekä kattavien automaattitestien kirjoittamisesta.

Tästä syystä päätös uudesta tuotantoasennuksesta voidaan tehdä pikaisestikin, ja asennuksia on tehty myös samana päivänä päivitystarpeen ilmenemisestä.

Tuotantoon on asentunut puolen vuoden aikana yksi ohjelmointivirheestä johtuva virhe. Kun virhe havaittiin, saatiin se ominaisuuskytkimien ansiosta otettua pois käytöstä tuotannossa. Tällöin myöskään virhekorjauksen kanssa ei ollut kiire, sillä ominaisuus ei itsessään ollut bisneskriittinen.

Vaikka asennus on automatisoitu myös kantamigraatioiden osalta, ei migraatiot hoitava DbMaintain-työkalu osaa ajaa Postgresql-kantaan triggereitä. Tästä syystä paria tietokannassa olevaa triggeriä täytyy päivittää manuaalisesti niiden muuttuessa. Koska muutoksia tulee todella harvoin, on triggerin päivitys joskus unohtunut josta on seurannut ongelmia tuotannossa.

Koska mitään virallista versionjulkaisua ei tehdä ja tuotantoon siirtyy aina myös osittain tehtyjä toiminnallisuuksia, ei kehittäjillä ole selkeää kuvaa siitä, mikä versio tuotannossa mistäkin ominaisuudesta tällä hetkellä on. Uutta versiota asennettaessa tiedetään mitä ollaan asentamassa, mutta varmaa käsitystä siitä, mitä aiempi versio sisälsi ja mitä muutoksia nykyisessä versiossa suhteessa aiempaan on, ei ole. Tämä on aiheuttanut välillä epäselvyyksiä ja tunteen, ettei projekti ole hallinnassa. Tämä on luonnollisesti tarvittaessa selvitettävissä vertaamalla aiempaa versionumeroa versionhallintaan.

Lisäksi paikoitellen kehittäjillä on haasteita haarattoman kehityksen kanssa. Puolivalmiita muutoksia ei muisteta aina piilottaa käyttöliittymästä, jolloin tuotantoon on pari kertaa lipsahtanut toiminnallisuutta, jota ei ollut tarkoitus vielä julkaista.

Kehittäjä antaisi pisteitä laadulle 3/5 ja turvallisuuden tunteelle 4/5. Vaikka luottamus itse sovelluksen toimintaan on korkea, on ulkoisista integraatioista saatavan HR-järjestelmän datan ja sen testaamisen kanssa suuria ongelmia, mikä vaikuttaa koko sovelluksen toimintaan.

Motivaatio on myös hyvällä tasolla pisteillä 4/5. Motivaatioon vaikuttaa positiivisesti erityisesti vapaus valita teknologiat ja mahdollisuus vaikuttaa toimintatapoihin, sekä hyvä projektitiimi. Turhautumista, eli samalla motivaation heikkenemistä, aiheuttaa ulkoiset tekijät joihin ei voi vaikuttaa. Näitä ovat esimerkiksi pääsyrajoitteet eri ympäristöihin sekä muista järjestelmistä saatava epäselvä data, jonka kanssa sovelluksen on pystyttävä kaikesta huolimatta operoimaan.

Stressittömyydestä pisteitä annettiin 4/5. Pelkoa siitä, että järjestelmä tekisi jotain totaalisen väärin, ei ole. Täyttä varmuutta siitä, että integraatioista saadaan kaikki vaadittava data ja se käsitellään virheettömästi, ei ole. Nämä ovat kuitenkin haasteita, joihin on varauduttu reagoimaan ongelmia ilmetessä nopeasti.

5.3.3. Analyysi

Projektissa on alusta asti pyritty aktiivisesti jatkuvan toimittamisen käytäntöihin. Asiakkaan sisäinen byrokratia kuitenkin estää kunnollisen jatkuvan toimittamisen, sillä palvelinten konfiguraatiota ei ole saatu automatisoitua ja tuotantoasennukset on tehtävä sovittuina aikoina paikan päältä. Niistä käytännöistä, jotka on saatu adaptoitua, on seurannut selkeästi hyötyä, ja niistä, joita ei ole voitu noudattaa, on taas seurannut haasteita.

Kattava testiautomaatio ehkäisee virheiden syntymistä ja tuo luottoa siitä, että järjestelmä toimii. Erityisesti selaintestien kanssa on täytynyt välillä tehdä ylimääräistä työtä, mutta työn koettiin maksavan itsensä takaisin sekä ehkäisemällä virheitä että tuomalla luottoa järjestelmän toiminnasta. Koska testit kattavat toiminnallisuuden niin hyvin, on tuotantoon asentunut virheitä erittäin harvoin, erillistä testausvaihetta ei tarvita ja tuotantoasennus voidaan tehdä tarvittaessa milloin tahansa. Automaattiseen testaukseen panostaminen siis parantaa huomattavasti ohjelmiston laatua, säästää henkilötyöstä johtuvia kustannuksia ja nopeuttaa asennussykliä, kun erityistä testausvaihetta ei tarvita.

Kaikista manuaalisista vaiheista asennuksessa on seurannut jotain ongelmia. Koska ympäristöjä ei konfiguroida automaattisesti, oli tuotantopalvelimet aluksi konfiguroitu puutteellisesti. Ohjelmiston toiminnallisuuksia hallitaan ominaisuuskytkimillä, joiden tila jokaisessa ympäristössä ei ole selkeästi hallinnassa, vaan kytkinten tilan näkee ja sitä voi muuttaa pelkästään kyseisellä palvelimella olevasta konfiguraatiotiedostosta, johon kaikilla ei ole pääsyä. Lisäksi joskus asennuksen yhteydessä pitää muistaa ajaa muuttuneet triggerit tietokantaan, mikä on myös joskus unohtunut ja aiheuttanut ongelmia tuotannossa.

Myös haarattomasta kehityksestä seurasi haasteita – tuotantoon lipsahti puolivalmiita ominaisuuksia ja selkeää käsitystä siitä, mitä tuotannossa milläkin hetkellä on, ei projektitiimillä ollut.

Tämänkin voidaan todeta johtuvan osittain siitä, että tuotantoasennukset eivät ole sään-

nöllisiä, vaan niitä tehdään tarvittaessa. Mitään listaa muutoksista ei pidetä, joten kehittäjä ei suoraan tiedä miten aiempi tuotannossa oleva versio eroaa uudesta versiosta. Jos tuotantoasennus saataisiin tehtyä esimerkiksi viikoittain, tottuisi kehitystiimi todennäköisesti tähän malliin. Tällöin epäselvyys ei olisi niin suuri, sillä aiemmasta asennuksesta on aina hyvin lyhyt aika. Haarattomasta kehityksestä voi siis seurata haasteita, mikäli tuotantoasennuksia ei voida tehdä tarpeeksi usein.

Luottoa laatuun, stressittömyyttä ja turvallisuuden tunnetta laskee projektissa erityisesti ulkoisista integraatioista tuleva data sekä datan virheellisyys, mihin ei projektitiimi voi vaikuttaa. Positiivisena vaikutuksena koettiin mahdollisuus reagoida ongelmiin nopeasti, sekä luotto oman ohjelmiston tekniseen toimintaan. Motivaatiota lisäsi vapaus valita teknologiat ja toimintatavat, ja vähensi asiat, jotka olivat oman vaikutusmahdollisuuden ulkopuolella. Jatkuvan toimittamisen käytännöistä johtuva hallinta selkeästi vaikutti positiivisesti kaikkiin näihin tuntemuksiin, kun puutteet hallinnassa vaikuttivat taas negatiivisesti. Jatkuvan toimittamisen käytännöt lisäävät hallintaa ja sen tunnetta koko projektiin, joten sen vaikutukset selkeästi ovat positiivisia stressiin, motivaatioon ja turvallisuuden tunteeseen.

5.4. Analyysien yhteenveto

Vaikka yhdessäkään projektissa kaikkia jatkuvan toimittamisen käytäntöjä ei noudatettu täsmällisesti, löydettiin jokaisesta projektista sekä hyötyjä että haasteita jotka käytännöistä seuraavat. Pääsääntöisesti jatkuvan toimittamisen käytäntöjen noudattamisesta seurasi selkeää hyötyä tuotteen laatuun, asennusten luotettavuuteen ja mahdollisuuteen reagoida muutostarpeisiin nopeasti.

Testikattavuus oli pääsääntöisesti projekteissa hyvällä mallilla lukuun ottamatta projektia 2, jossa selaimella suoritettavat hyväksyntätestit puuttuivat kokonaan. Projekteissa 1 ja 3, joissa testit kattoivat myös selaintestit, koettiin testien selkeästi edesauttavan luottamusta ohjelmiston virheettömään toimintaan sekä konkreettisesti ehkäisemään virheitä. Kuitenkin selaintestien puuttumista projektissa 2 ei nähty suureksi ongelmaksi, sillä mahdollisiin virheisiin pystytään reagoimaan näitä ilmetessä nopeasti.

Yksikkö- ja integraatiotestien lisäksi selaimella tehtäviin hyväksyntätesteihin on siis suositeltavaa panostaa mikäli mahdollista. Jos tästä koituvat kustannukset alkavat tuntua

kuitenkin kohtuuttomilta, voidaan ongelma mahdollisesti taklata sillä, että korjauksien asennus tuotantoon pidetään suoraviivaisena, nopeana ja turvallisena. Käyttöliittymätesteillä voi siis olla perusteltua kattaa kustannussyistä vain perustoiminnallisuus ja huolehtia siitä, että jatkuvan toimittamisen hengessä on mahdollista reagoida virheisiin nopeasti. Tämä luonnollisesti vaatii myös järjestelmän olevan luonteeltaan sellainen, että pienet virheet jossain käyttöliittymän osissa eivät aiheuta merkittävää liiketaloudellista haittaa kunhan korjaukset saadaan asennettua nopeasti.

Jatkuvan integraation vaiheella oli jokaisessa projektissa suuri rooli ohjaamassa ohjelmistokehitystä. Jokaisen onnistuneen vaiheen jälkeen ohjelmistosta asennettiin automaattisesti käännetty versio kehitysympäristöön, jonka jälkeen kehitystiimi oli varsin luottavainen siihen, että järjestelmän voisi asentaa turvallisesti tuotantoon. Yksikään projektitiimi ei pitänyt tämän prosessin toteuttamiseen kulunutta aikaa kohtuuttomana, ja kattavaa jatkuvan integraation vaihetta pidettiin ehdottoman tärkeänä laadun takaajana. Automaattisen testauksen sekä jatkuvan integraation avulla on siis mahdollista saada kohtuullisella panostuksella projekti siihen tilaan, että ohjelmisto on aina tuotantoasennettavissa, virheet saadaan minimoitua ja tuotantoasennus uskalletaan tehdä milloin tahansa.

Ympäristöjen konfiguraatioiden automatisoinnista koettiin olevan huomattavasti etua niissä projekteissa missä tähän pystyttiin. Toisaalta taas projektissa, jossa tähän ei pystytty, oli ongelmia, joita ei olisi ilmennyt mikäli konfiguraatiot olisi voitu automatisoida. Hyötyä saatiin erityisesti siinä tilanteessa, jossa osa ohjelmistoista oli hajautettu usealle palvelimelle. Ainoat haasteet ympäristöjen konfiguraatioiden automatisoinnissa olivat työkalujen toimimattomuus Windows-käyttöjärjestelmällä sekä itse työkalujen käytön opetteluun kuluva aika. Kuitenkin siitä, että ympäristöt olivat hallinnassa ja infrastruktuurin konfigurointi automatisoitu, koettiin saavan merkittäviä hyötyjä niissä projekteissa joissa käytäntöä pystyttiin noudattamaan. Haastattelujen perusteella voidaan kiistatta todeta, että infrastruktuurin ja konfiguraatioiden automatisointiin kuluva aika on hyvin pieni suhteessa siitä saataviin hyötyihin.

Projekteissa 2 ja 3 tuli ilmi ongelmia, jotka olivat seurausta siitä, että osa palvelimista ja ympäristöistä ei ole lainkaan palveluntoimittajan hallinnassa. On kyse sitten toimitusputken hallintaan liittyvistä palvelimista tai kohdeympäristöjen palvelimista, on erittäin perusteltua, että palveluntoimittaja saa täyden hallinnan koko infrastruktuuriin. Täl-

löin kaikki infrastruktuurissa ilmenevät ongelmat ovat ehkäistävissä, analysoitavissa tai korjattavissa palveluntoimittajan toimesta ilman ylimääräisiä välikäsiä. Projekteissa joissa näin ei ole, oli osittain havaittavissa jopa välinpitämättömyyttä asioiden suhteen, kun omaa vaikutusmahdollisuutta ei ole.

Vaikka automatisoinnin taso vaihteli projektikohtaisesti, olivat automatisoinnin edut selkeästi havaittavissa jokaisessa projektissa. Kaikki manuaalista työtä vaativat vaiheet aiheuttivat jonkun pullonkaulan, mikä vähensi tuotantoasennusten tiheyttä. Mikäli tuotantoasennus vaatii manuaalisen testausvaiheen tai edes useammalle palvelimelle kirjautumisen, ei asennuksia tehty niin usein. Mikäli asennuksia taas ei tehty riittävän usein, oli pelko siitä, että jotain menee pieleen, suurempi. Vaikka tekninen valmius tuotantoasennusten säännölliseen tekemiseen olisikin, on äärimmäisen tärkeää poistaa kaikki klikkausta monimutkaisemmat manuaaliset vaiheet, mikäli jatkuvaa toimittamista halutaan tehdä. Tällöin tuotantoasennus tulee tehtyä usein, jolloin siitä tulee arkipäiväistä.

Automatisointi, joka on osa jatkuvan toimittamisen selkärankaa, voidaan todeta erittäin hyväksi laadun parantajaksi. Inhimillisiä virheitä sattuu aina, mikäli asennukseen ja konfigurointiin liittyy vaiheita, jotka ovat riippuvaisia ihmisen muistamisesta sekä osaamisesta. Vaikka prosessi olisi automatisoitu hyvinkin pitkälle, voidaan olettaa, että pienistäkin manuaalisista vaiheista seuraa ongelmia ennemmin tai myöhemmin.

Vain projektissa 3 oli kehitysmallina haaraton kehitys, kun muissa projekteissa luotiin ominaisuushaaroja, jotka valmistumisen jälkeen yhdistettiin päähaaraan odottamaan seuraavaa tuotantoasennusta. Projekteissa, joissa haaroja luotiin, ei koettu olevan ongelmia siinä, että keskeneräiset ominaisuudet ovat vain kehittäjän koneella, tai että haarojen yhdistämisessä ilmenisi ongelmia. Projektissa 3, jossa haaratonta kehitystä tehdään, törmättiin taas haasteisiin. Keskeneräisiä ominaisuuksia saattoi lipsahtaa tuotantoon ja selkeää käsitystä siitä, mitä on tuotannossa tai mikä on valmista, ei ollut.

Haaraton kehitys voi siis muodostua ongelmaksi mikäli tuotantoasennuksia ei tehdä säännöllisesti. Tällöin kehittäjät voivat lipsahtaa malliin, jossa päähaaraan tehdään epäjärjestelmällisesti muutoksia ilman, että kokonaiskuva valmiista ominaisuuksista pysyy yllä. Kuitenkin haarattoman kehityksen seurauksena mukana tulevien ominaisuuskytkinten ansiosta tuotannosta saatiin myös otettua pois käytöstä sinne jo asennettu ominaisuus, josta löydettiin virhe. Haarattomasta kehityksestä voi siis seurata odottamattomiakin hyö-

tyjä, mutta sen toteuttaminen vaatii itsekuria, järjestelmällisyyttä sekä yhteispeliä. Mikäli kehittäjät eivät sitoudu haarattoman kehityksen käytäntöihin tai ymmärrä niitä, voivat seuraukset olla ongelmallisia.

Vaikka ohjelmiston hajauttaminen useammalle palvelimelle kuorman jakamisen vuoksi ei välttämättä ole tarpeellista, voi se olla perusteltua katkottoman asennuksen mahdollistamiseksi. Mikäli palvelu on säännöllisessä käytössä ja tuotantopäivityksestä seuraa käyttökatko, saattaa asennuksien tiheys vähentyä huomattavasti. Vaikka tekninen valmius säännöllisiin ja luotettaviin tuotantoasennuksiin olisi, on hyvä huomioida myös muut esteet sekä minimoida kaikki byrokratia, mikä tuotantoasennusten tiheyttä voi heikentää.

Työntekijöiden motivointi on erittäin tärkeää jokaisessa organisaatiossa, sillä motivaatiosta riippuu millä halulla ihminen käyttää omia resurssejaan työn suorittamiseen. Motivoituneet työntekijät tekevät parempaa tulosta, ovat luovempia ja saavat aikaan enemmän. Tietotekniikan alalla uuden oppiminen ja asioiden paremmin tekeminen koetaan usein intohimona ja motivaattorina. Lisäksi motivaatiota lisää tunnetusti myös tunne siitä, että tuntee itse vastuuta työstä ja työnsä jäljestä. Haastattelujen perusteella jatkuvan toimittamisen käytäntöjen ja työkalujen hyödyntäminen parantaa työntekijöiden motivaatiota projektissa ja tällöin myös sitoutuneisuutta organisaatioon.

Suurempi vastuu koko järjestelmästä ja toimitusprosessista lisäsi myös stressiä, mikä voi vaikuttaa henkilöstä ja stressin määrästä riippuen negatiivisesti tai positiivisesti sekä tuottavuuteen että henkilöiden hyvinvointiin. Stressiä koettiin vähemmän niissä projekteissa, joissa koko järjestelmä ja infrastruktuuri ei ollut täysin kehitystiimin vastuulla. Lisäksi työntekijöiden täytyy omata riittävät tekniset taidot ottaakseen vastuun ohjelmistokehittämisen lisäksi palvelinympäristöistä, konfiguraatioista, työkaluista ja kaikkeen liittyvästä automatisoinnista. Kuitenkin vain osa ohjelmistokehittäjistä on riittävän kyvykkäitä ottaakseen hallintaan näin suuren kokonaisuuden. Jatkuvan toimittamisen käytäntöjen noudattaminen ei siis ole helppoa ja vaatii tiimin, joka pystyy ottamaan vastuulleen kaiken tämän, ja jolla on tahtoa sekä määrätietoisuutta uusien asioiden opetteluun ja sovittujen käytäntöjen noudattamiseen.

5.5. Arviointi

Tutkimus kattaa vain kolme ohjelmistoprojektia ja yhden haastattelun tulokset perustuvat yksittäisen projektitiimiläisen näkemyksiin. Henkilö on myös projektin tekninen vastuuhenkilö, joten hän on vastuussa suuresta osasta teknologiapäätöksistä sekä toimintamalleista. Lukuja käytetyistä työmääristä esimerkiksi testaukseen sekä teknisiin järjestelyihin ei ole, joten tässä suhteessa luotetaan vain kehittäjän omaan tuntemukseen. Usein ohjelmistokehittäjät kuitenkin käyttäisivät mieluusti enemmän aikaa teknisien yksityiskohtien kehittämiseen, minkä lisäarvo asiakkaalle on kyseenalaistettavissa. Tästä syystä on vaikea arvioida kuinka suuri panostaminen esimerkiksi automaattiseen testaukseen on kannattavaa.

Koska jokainen ohjelmistoprojekti on yksilöllinen, on mahdoton arvioida sitä, mikä tilanne olisi, mikäli asiat olisi tehty kyseisessä projektissa toisin. Tästä syystä ei ole mahdollista arvioida esimerkiksi ohjelmointivirheiden tai tuotantoasennuksessa ilmenneiden ongelmien määrää, mikäli testikattavuuteen olisi panostettu huomattavasti vähemmän tai asennuksia ei olisi automatisoitu.

Vaikka henkilön kokemusten subjektiivisuus voi olla paikoitellen kyseenalaista, on tuloksista selkeästi nähtävissä, että jatkuvan toimittamisen käytännöt sekä päivittäinen tuotantoasennusvalmius vähentää virheitä ja parantaa laatua sekä mahdollisuutta reagoida muutostarpeisiin nopeasti. Lisäksi epäsuorat vaikutukset projektitiimin motivaatioon, turvallisuudentunteeseen ja täten työhyvinvointiin voivat olla yllättävänkin positiivisia.

6. YHTEENVETO

Jatkuvan toimittamisen käytännöt määräävät tiukasti sen, miten järjestelmää kehitetään ja testataan, kuinka palvelinympäristöt on konfiguroitava ja miten kaikki tämä automatisoidaan aina tuotantoasennukseen asti. Hyvin toteutettu asennusputki toimii selkärankana näiden käytäntöjen noudattamiseen, jolloin päästään ohjelmistokehityksessä siihen tilaan, että järjestelmästä on aina asennettavissa tuore ja testattu versio tuotantoon, ja tuotantoasennuksesta voidaan saada arkipäiväinen toimenpide.

Ympäristöjen konfiguraatioiden automatisointi on tehokas tapa ehkäistä vääristä konfiguraatioista johtuvia ongelmia eri ympäristöissä. Infrastruktuurin konfiguraatioiden automatisointityökaluilla voidaan estää järkevällä työmäärällä jopa kokonaan tilanteet, joissa palvelimet on konfiguroitu väärin tai jotain konfiguraatioasetuksia ei tarvittavalla hetkellä tiedetä.

Kattavan testiautomaation ja jatkuvan integraation avulla projektin kehittäjät voivat luottaa järjestelmän toimivuuteen, jolloin päähaarasta voidaan tehdä tuotantoasennus vaikka päivittäin. Vaikka tuotantoasennuksia ei kahdessa haastateltavasta projektista tehty edes viikoittain, paransivat jatkuvan toimittamisen käytännöt ohjelmiston laatua ja alensivat kynnystä tehdä tuotantoasennus milloin tahansa. Koska tuotantoasennus on automatisoitu ja helposti käynnistettävissä, epäonnistuu se äärimmäisen harvoin tai ei koskaan.

Koska jatkuvassa toimittamisessa projektitiimi ottaa vastuulleen koko järjestelmän ja infrastruktuurin, täytyy henkilöiden olla teknisiltä taidoiltaan riittävän kyvykkäitä tämän kokonaisuuden hallintaan ja ymmärtämiseen. Lisäksi suuri vastuu saattaa lisätä joidenkin henkilöiden stressitasoa. Suurempi vastuu ja uusien asioiden opetteleminen vaikutti kuitenkin myös positiivisesti henkilöiden motivaatioon, joten jatkuvan toimittamisen käytäntöjen noudattaminen voi edistää myös henkilöstön sitoutumista projektiin sekä koko organisaatioon.

Käytäntöjen noudattaminen ei kuitenkaan tule ilmaiseksi. Työkalujen opettelemiseen, automaattitestien kirjoittamiseen, asennusputken ylläpitoon ja koko putken automatisoin-

tiin kuluu luonnollisesti aikaa. Näihin panostaminen ohjelmistoprojektin alkuvaiheesta lähtien on kuitenkin perusteltua, sillä tällöin projektin resursointi ja työtahti on tyypillisesti suurempi. Mikäli näistä lipsutaan alkuvaiheessa, on käytäntöjen adaptoiminen myöhemmässä vaiheessa työmäärältä oleellisesti suurempi. Jos käytäntöjä ei noudateta lainkaan, on hyvin todennäköistä, että ongelmia ratkotaan projektin myöhemmässä vaiheessa, jolloin selvittelyyn menee huomattavasti enemmän aikaa ja henkilöresursseja ei välttämättä ole niin paljon käytettävissä. Jos manuaalista testausta tai asentamista tehdään koko projektin ajan, tulee tästä jatkuvia kustannuksia, jotka pystytään automatisoinnilla vähentämään. Jatkuvan toimittamisen käytännöt eivät siis pelkästään ennaltaehkäise virheitä, vaan myös vähentävät työmäärää ja kustannuksia, joita manuaalisesta työstä sekä ongelmien selvittelystä myöhemmässä vaiheessa ilmenee.

Tutkimustulokset perustuvat ohjelmistoprojektien teknisten vastuuhenkilöiden haastatteluihin eikä tuloksia ole mahdollista verrata tilanteeseen, jossa asiat olisi tehty toisin. Siitä huolimatta jatkuvan toimittamisen käytäntöjen noudattamisesta seuranneet hyödyt olivat kolmen projektin kokemusten perusteella selkeästi havaittavissa.

Haastattelujen pohjalta ilmenneet tutkimustulokset jatkuvan toimittamisen positiivisista vaikutuksista liittyivät hyvin voimakkaasti tuotteen laatuun sekä ohjelmistokehitystyöhön vaikuttaviin tekijöihin. Tämän voi olettaa johtuvan ainakin osittain siitä, että sekä haastateltavat että haastattelija tekevät valtaosin ohjelmistokehitystä työkseen. Tutkimusta voisi jatkaa haastattelemalla myös asiakkaan edustajia sekä projektinhallintaa, jotta vaikutusten kokonaisvaltaisuutta pystyttäisiin arvioimaan paremmin.

LÄHTEET

[1] Beck, K., 1999. Extreme Programming Explained: Embrace Change

[2] Chen, L. 2015. Continuous Delivery - Huge Benefits, but Challenges Too. Software, IEEE 32.2 (2015): 50-54

[3] Gmeiner, J., Ramler, R., Haslinger, J. 2015. Automated Testing in the Continuous Delivery Pipeline: A Case Study of an Online Company. 2015 Software Testing, Verification and Validation Workshops (ICSTW), 2015 IEEE Eighth International Conference on. IEEE, 2015.

[4] Humble, J., Farley, D. 2011. Continuous Delivery - Reliable software releases through build, test and deployment automation. Pearson Education, Inc, Boston, Addison-Wesley

[5] Psykologinen kuormitus. Www-lähde. Luettu 26.10.2015, luettavissa http://www.ttl.fi/partner/riskithaltuun/psykososiaalinen_kuormitus/sivut/default.aspx

[6] Driessen, V. 2010. A succesfull Git branching model. Www-lähde. Luettu 26.10.2015, luettavissa http://nvie.com/posts/a-successful-git-branching-model

[7] Gitflow Workflow. Www-lähde. Luettu 26.10.2016, luettavissa https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow

[8] Git Flow - Git plugin. Www-lähde. Luettu 26.10.2015, luettavissa https://github.com/nvie/gitflow

[9] Leppänen, M., Mäkinen, S., Pagels, M., Eloranta, V. P., Itkonen, J., Mäntylä, M. V., & Mannisto, T. 2015. The Highways and Country Roads to Continuous Deployment. Software, IEEE, 32(2), 64-72.

[10] Neely, S., Stolt, S., Rally Software 2013. Continuous Delivery? Easy! Just Change Everything (well, maybe it is not that easy). Agile Conference (AGILE), 2013 (pp. 121-128). IEEE.