hyväksymispäivä arvosana

arvostelija

Jatkuva eksperimentointi ohjelmistokehityksen tukena

Esa Kortelainen

Kandidaatintutkielma
HELSINGIN YLIOPISTO
Tietojenkäsittelytieteen laitos

Helsinki, 21. huhtikuuta 2015

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty	Laitos — Institution — Department
Matemaattis-luonnontieteellinen	Tietojenkäsittelytieteen laitos

Tekijä — Författare — Author
Esa Kortelainen

Työn nimi — Arbetets titel — Title

Jatkuva eksperimentointi ohjelmistokehityksen tukena

Oppiaine — Läroämne — Subject
Tietojenkäsittelytiede

Työn laji — Arbetets art — Level	Aika — Datum — Month and year	Sivumäärä — Sidoantal — Number of pages
Kandidaatintutkielma	21. huhtikuuta 2015	21

Tiivistelmä — Referat — Abstract

Tässä tutkielmassa käydään läpi eksperimentoinnin käsitettä ohjelmistotuotannossa, sekä motiiveja jatkuvaa eksperimentointia hyödyntävään ohjelmistotuotantoon siirtymiseen. Eksperimentointia hyödynnettäessä etukäteen suunnittelua tehdään mahdollisimman vähän, ja kehityksen edetessä todennetaan erilaisilla koeasetelmilla miten hyvin ohjelmisto vastaa käyttäjän tarpeita. Tutkielman alussa esitellään Lean-filosofia, malleja sen soveltamiseen ohjelmistotuotannossa sekä Leanin ja ketterien menetelmien yhtymäkohtia ja eroavaisuuksia. Leanista siirrytään teknologiayrityksille suunnatun Lean Startup -filosofian, eksperimentoinnin käsitteen ja sille pohjaa luovan, jatkuvasti verkottuneemman ohjelmistotuotantoympäristön läpikäyntiin. Johtopäätöksenä voidaan todeta, että mitä enemmän etenkin pienen organisaation ohjelmistokehitystä ohjataan kokeiden tulosten ja todellisen käyttödatan perusteella luulojen ja mielipiteiden sijaan, sitä vähemmän kulutetaan aikaa ja resursseja työhön joka ei lisää asiakkaan ohjelmistotuotteesta saamaa arvoa.

ACM Computing Classification System (CCS):

- **Software and its engineering~Empirical software validation**
- **Software and its engineering~Software prototyping**
- *Software and its engineering~Software development methods*

Avainsanat — Nyckelord — Keywords
eksperimentointi, Lean Startup, Lean, SaaS

Säilytyspaikka — Förvaringsställe — Where deposited

Muita tietoja — Övriga uppgifter — Additional information

Sisältö

1 Johdanto

Informaatioyhteiskuntamme kehittyessä kaikki tieto liikkuu koko ajan nopeammin, mikä tekee perinteisestä asioiden etukäteen suunnittelusta jatkuvasti tehottomampaa verrattuna informaation hyödyntämiseen ohjelmistojen kehityksessä. Yhä suurempi osa ihmisten Internetin käytöstä tapahtuu mobiileilla älylaitteilla, jotka ovat matalatehoisia verrattuna suurempikokoisiin laitteisiin. Tämä vauhdittaa osaltaan jatkuvasti lisääntyvää ohjelmistojen siirtymistä täysin verkosta käsin käytettäviksi [Bos12]. Näin ollen on yhä helpompaa ja halvempaa seurata jatkuvasti miten, missä ja milloin käyttäjät oikeasti toimivat ohjelmiston parissa. Tämä on perusteltua etenkin siksi, että käyttäjät eivät pysty antamaan palautetta ohjelmiston kokonaiskuvasta, eli siitä miltä heidän tapansa käyttää ohjelmistoa näyttää suhteessa muihin käyttäjiin ja ohjelmiston sisäisiin järjestelmiin.

Ketterien menetelmien suosio ohjelmistokehityksen piirissä on kasvanut sen jälkeen, kun niiden periaatteet esiteltiin vuonna 2001 [BBv+01, NMM05]. Ne tarjoavat hyvän pohjan tehokkaalle ohjelmistotuotannolle, mutta eivät tarjoa menetelmiä sen todentamiseen, tehdäänkö loppujen lopuksi sellaista ohjelmistotuotetta, jota asiakkaat oikeasti tarvitsevat tai haluavat ostaa [PC12]. Jos halutaan luoda uusi ohjelmisto jonka lopullisesta vastaanotosta asiakkaan käytössä tai kuluttajamarkkinoilla ei ole konkreettista tietoa, on perusteltua kokeilla miten se käytännössä toimii jo suunnitteluvaiheessa. Näin huonosti toteutettu tai muuten markkinoille sopimaton tuote voidaan hylätä heti tai korjata vastaamaan alunperin tai kehityksen edetessä huomattuja asiakkaiden tarpeita [Rie11].

Nopea erilaisten ideoiden kokeilutahti tukee uudistumista ja innovointia, joka on etenkin ohjelmistoyritykselle kriittistä sen toiminnan menestyksen ja jatkuvuuden kannalta [Bos12]. Baregheh, Rowley ja Sambrook määrittelevät innovaation aiheeseen liittyvässä kirjallisuuskatsauksessaan seuraavalla tavalla: "*innovaatio* on monitasoinen prosessi, jossa organisaatiot muuttavat ideoita uusiksi tai parannelluiksi tuotteiksi, palveluiksi tai prosesseiksi, tavoitteenaan edetä, kilpailla ja erottautua onnistuneesti omalla markkinallaan" [BRS09].

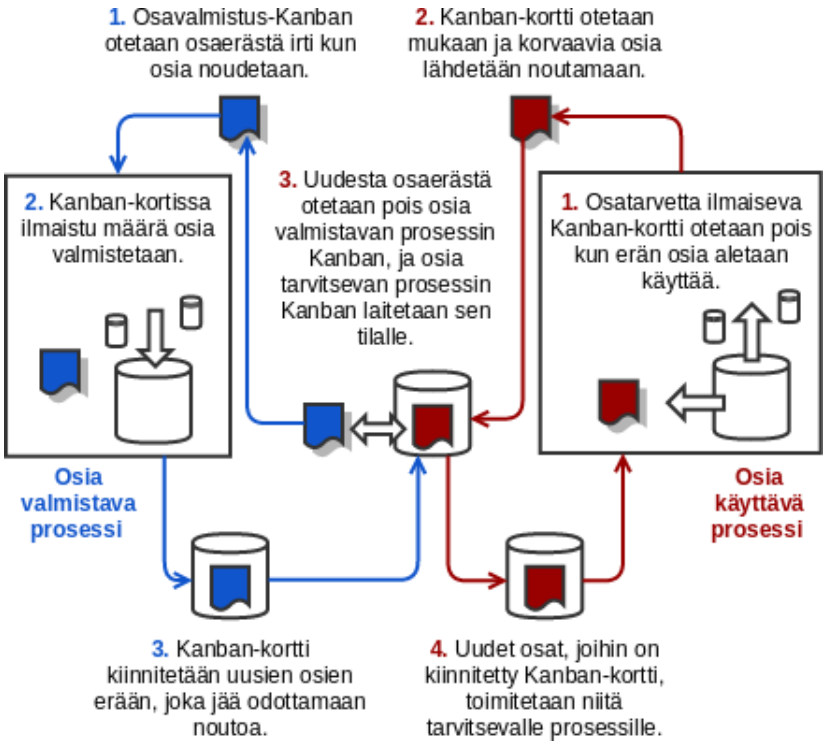
Tässä tutkielmassa käydään läpi tehokkaiden ohjelmistotuotantomallien

filosofiaa ja pyritään vastaamaan tutkimuskysymyksiin siitä, mitä eksperimentointi ohjelmistokehityksen kontekstissa on ja miksi siihen kannattaisi siirtyä. Tämän lisäksi luodaan katsaus siihen, minkälaisia ominaisuuksia jatkuvaa ekpserimentointia toteuttava ohjelmistotuotantoprosessi vaatii käytännössä.

2 Lean-ajattelu

Lean-ajattelu sai alkunsa 1940-luvun lopulla, kun Toyotan autotehtaiden tuotantolinjojen toimintaa tehostettiin [PC12]. Se sai nykymuotoisen nimensä 1980-luvun puolivälissä, kun amerikkalainen insinööri John Krafcik opiskeli MIT:ssä ja havaitsi Toyotan mallin erityisen tehokkaaksi vertaillessaan eri maiden autotehtaiden toimintaa. Krafcik huomasi että Toyotan tehtaalla valmistettavien autonosien tuotantomääriä ja -aikatauluja ei suunniteltu tarkasti etukäteen, vaan osia tehtiin pieniä eriä siinä vaiheessa, kun niille tiedettiin olevan tarvetta (JIT, ”just in time”, kts. kuva 1). Tuotantolinjan informaationkulun suunta oli vaihdettu käyttäen Kanban-kortteja tiedonvälitykseen. Kun jossakin prosessissa alettiin käyttää osia uudesta erästä, osiin kiinnitettyyn Kanban-korttiin kirjoitettiin miten paljon prosessissa tarvitaan mahdollisesti lisää osia [Ōno88]. Kortti otettiin mukaan ja osat noudettiin edeltävältä prosessilta. Valmiita osia otettiin tarvittu määrä. Samalla Kanban-kortti, joka oli peräisin osia tarvitsevasta prosessista kiinnitettiin niihin. Osia valmistavan prosessin osatuotantoa ohjattiin prosessin omilla Kanban-korteilla, joita oli kiinnitetty valmiiden osien eriin. Tähän Kanban-korttiin merkittiin toiselta prosessilta saatu uusi osatarve, ja kortti otettiin osien tuotantolinjaan odottamaan, että uudet osat saatiin valmiiksi. Kortti kiinnitettiin uusiin osiin niiden valmistumisen jälkeen. Koska kaikkien osaerien yhteydessä piti kulkea aina Kanban-kortti, tuotantolinjan reaktioaikaa uusiin tarpeisiin voitiin säätää lisäämällä tai vähentämällä siinä liikkuvien korttien kokonaismäärää. Viallisia osia ei saatu lähettää toisille prosesseille, joten osavioista johtuvat ongelmat eivät laajentuneet yhtä prosessia pidemmälle.

Perusfilosofiana tavaraa ei enää ”työnnetty” varastoon odottamaan mahdollista käyttöä, vaan käyttötarpeen ilmetessä sitä ”vedettiin” jatkuvasti virtaavan tuotantolinjan läpi, tavoitteena saada osat suoraan käyttöön [PC12].



Kuva 1: JIT-osavalmistuskierto Toyotan tehtailla [Toy, mukailtu].

Toyotan mallissa työntekijät koulutettiin hallitsemaan monta tuotantolinjan eri vaiheisiin kuuluvaa tehtävää. Jos jossain havaittiin ongelma, tuotantolinja pysäytettiin ja ongelma korjattiin välittömästi. Parannukset johtivat siihen, että Toyotan tehtaat tuottivat noin puolikkaalla työtuntimäärällä saman verran autoja kuin amerikkalaisten autoyhtiöiden tehtaat [PC12]. Lisäksi varastojen käyttö oli tehokkaampaa ja autot laadukkaampia.

Termien ”Lean” ja ”JIT” tunnettavuus laajeni kun ne esiintyivät MIT:n Toyota-tutkimuksia käsittelevässä kirjassa *The Machine That Changed The World* [JRW90]. Kirjailijat laajensivat Lean-ajattelun sen alkuperäisestä ympäristöstä mihin tahansa tehokaaseen johtamiskäytäntöön, joka minimoi resurssien haaskauksen. Alkuvaiheessa ”Lean” yhdistettiin japanilaisiin johtamiskäytäntöihin, mutta sen pääpaino on aina ollut ajan ja henkilötyötuntien tuhlaamisen vähentämisessä, asiakkaan, tuotteen ja yrityksen saaman arvon korostamisessa sekä joustavamman, iteratiivisen ja kevyen kehitysprosessin hyötyjen tähdentämisessä [PC12].

2.1 Lean-ajattelu ohjelmistotuotannossa

Idea Lean-ajattelun soveltamisesta ohjelmistotuotantoon syntyi 1990-luvun alussa nopeasti filosofian esittelyn jälkeen [PC12]. Alanharjoittajat ja tieteilijät päätyivät siihen ajatukseen, että Lean-ajattelun hengessä yrityksen ainoa tehtävä on tavoittaa ja palvella asiakkaita. Heidän huomionsa kiinnittyi siihen, miten tietotekniikkaa voidaan käyttää asiakkaan tarvitseman arvon luomiseen.

Mary ja Tom Poppendieck listaavat kirjassaan *Lean Software Development* [PP03] Lean-filosofian seitsemän periaatetta sovellettuna ohjelmistotuotantoon:

Optimoi kokonaisuus. Koko tuotantoprosessin perustana on ymmärrys siitä, mitä asiakas haluaa tehdä ja miten tämä asia voidaan suorittaa ohjelmistolla. Ohjelmisto itsessään ei yleensä sisällä mitään arvoa, se muodostuu kun ohjelmisto on yhteydessä suurempaan kokonaisuuteen, sen käyttöympäristöön. Arvoa ei määritellä vain kehitysvaiheessa, vaan käyttöönotto, ulkoasu ja käyttäjäkokemus ovat myös olennaisia. Arvoa ei myöskään yleensä luoda yhdellä, tietyn ajan vievällä työpanoksella, joten koodin täytyy olla aina helposti muokattavissa.

Älä haaskaa. Kaikki, mikä ei lisää asiakkaan saamaa arvoa tai tietoa siitä, miten tätä arvoa luodaan tehokkaammin on poistettava koko prosessista. Esimerkkeinä merkittävimmistä roskan aiheuttajista Poppendieckit mainitsevat turhat tai arvottomat ominaisuudet, kadonnut tieto, töiden siirtäminen tekijältä toiselle, osittain tehty työ ja sitäkin aiheuttava ”multitasking”. Suurin hävikki syntyy bugien etsimisestä ja niiden korjaamisesta, johon saattaa kulua jopa 40-50 prosenttia koko kehitykseen käytetystä ajasta. Yleensä näiden ongelmien juuret ovat ositetuissa tuotantomalleissa, joissa työvaiheesta seuraavaan siirryttäessä syntyy viiveitä ja informaatiohävikkiä. Kun arvon liikettä seurataan koko tuotantoketjussa, on helpompi löytää vaiheet, jotka aiheuttavat resurssien haaskausta.

Rakenna laadukkaasti. Ohjelmistoa kasataan pieninä moduuleina, jotka integroidaan heti toimivaksi osaksi koko koodia ilman kehityssyklin loppuvaiheen odottelua. Näin vältetään monimutkaiseksi ongelmaryppääksi muodustuva integraatiovaihe.

Opi jatkuvasti. Kehitystyö on tiedon luomista ja luodun tiedon sulauttamista tuotteeseen. Lean-ajattelussa kehityksen etenemiselle on kaksi vaihtoehtoa kontekstista riippuen. Vaihtoehtoja voidaan tarvittessa käyttää yhtäaikaa, vaikka ne vaikuttavatkin toisensa poissulkevilta. *"Opi ensin"*-etenemismallissa tutkitaan monta ehdotusta esimerkiksi käytettävävästä arkkitehtuurista tai ohjelmointikielestä. Kriittiset päätökset tehdään viime hetkellä, jolloin mahdollisimman paljon tietoa ja tutkittuja vaihtoehtoja on käytettävissä. *"Opi jatkuvasti"*-etenemismallissa otetaan käyttöön alustat ja järjestelmät, joita tuotteen työstämisen aloittamiseen tarvitaan vähintään, ja aletaan tuottaa toimivia kehitysversioita ohjelmasta. Ohjelmaa kehitetään eteenpäin siitä saadun palautteen perusteella.

Tuota nopeasti. Jatkuva tuotantoonotto muuttaa projektiajattelun jatkuvaksi "flow"-ajatteluksi, ja ohjelmistoa ei välttämättä ajatella enää selkeät alku- ja loppupisteet omaavana projektina. Koodauksen lisäksi tämän ajatusmallin pitäisi ulottua koko tuotteen kehityssykliin.

Osallista jokainen. Tuotetta kehittämässä ei ole vain koodaajia, vaan mukaan on otettava myös asiakkaita, käyttöliittymäsuunnittelua, tuotetukea, liiketoimintaa ja muita tuotteeseen liittyviä näköpuolia ymmärtäviä ihmisiä. Kun ihmiset ja yhteistyö sekä mahdollisimman alhaalla hierarkiassa tapahtuva päätöksenteko ovat pääosassa, ollaan Lean-ajattelun ytimessä.

Kehity koko ajan paremmaksi. Jokaista työvaihetta pitää kehittää jatkuvasti. Lean-ajattelussa on ilmeistä että tietyt käytännöt, jotka ovat joissakin tapauksissa hyväksi todettuja ovat harvoin parhaita ratkaisuja juuri siihen uuteen ongelmaan, joka on kehitystiimin ratkaistavana.

Laajaa huomiota saaneiden ketterien menetelmien käyttö ohjelmistotuotannossa alkoi vuonna 2001, kun Kent Beck ja monet muut ohjelmistoammattilaiset julkaisivat manifestinsa *Manifesto for Agile Software Development* [BBv+01]. Siinä asetetaan ihmiset, vuorovaikutus, toimiva ohjelmisto, yhteistyö asiakkaan kanssa ja muutosvalmius perinteisten työskentelymallien edelle. Nämä ketterien menetelmien perusajatukset sopivat yhteen Lean-ajattelun kanssa, ja menetelmiä ollaankin käytetty menestyksekkäästi rinnakkain ohjelmistotuotantoprosesseja tehostaen [PC12].

Useimmat ketterät menetelmät, kuten *Scrum* [Sch04] ja *Extreme Pro-*

gramming [Bec00], rajoittuvat kuitenkin ohjelmiston koodin ja teknisten ratkaisujen kehittämiseen, kun Lean-ajattelulla pyritään vaikuttamaan kaikkiin arvoa luoviin yhteyksiin koko tuotteen ympärillä. Lisäksi edellä mainituissa menetelmissä määritellään roolit henkilöille, joilla on vastuu eri työvaiheissa tehtävien töiden linjaamisesta. Tämä rikkoo suoraan Lean-ajattelun periaatetta koko prosessin optimoinnista, jonka mukaan jokaisen tuotteen parissa työskentelevän ryhmän jäsenistä pitää tehdä oma osansa tuotteen menestyksen edistämisestä [PC12].

Lean-ajattelu ottaa kantaa koko tuotteen elinkaareen ja näin kaikki kehityksen elementit saadaan nivottua yhteen palautesykliin. Ketterissä menetelmissä järjestelmäarkkitehtuurin ja vuorovaikutussuunnittelun oletetaan tulevan kehitysryhmän ulkopuolelta, tai vaihtoehtoisesti se tapahtuu huomattavan pienin askelin kehitystiimin sisällä. Tämän takia ketterät menetelmät ovat usein riittämättömiä ottamaan kantaa ratkaisusuunnitteluun, käyttäjäkokemuksen suunnitteluun tai korkean tason järjestelmäarkkitehtuuriin. Yleisesti ottaen ne nähdäänkin hyvinä tapoina organisoida ohjelmistokehitys, mutta vajavaisina ottamaan kantaa muiden tuotteeseen liittyvien asioiden suunnitteluun [PC12]. Koska kummatkin ovat iteratiivista prosesseja, on haitallista jos niitä ei ole integroitu toisiinsa.

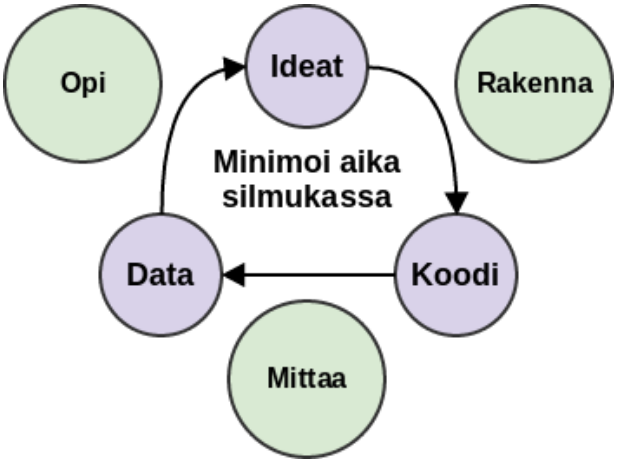
Poppendieck ja Cusumano toteavat, että tunnetut ketterät menetelmät tarjoavat hyvän alustan ryhmän ohjaamiseen ja työn järjestelyyn, mutta eivät kovinkaan kattavia työkaluja varmistamaan että työnteon kohde vastaa juuri sitä, mitä asiakas tahtoo [PC12]. Seuraavassa kappaleessa esitellään heidän mainitsemansa Lean Startup -malli. Siinä lähtökohtana on liiketoimintasuunnitelma, jonka oletuksien paikkansapitävyyttä tutkitaan muutoksien jälkeen jatkuvasti.

2.2 Lean-filosofia kohtaa startup-kulttuurin

Liiketoimintamalli heijastaa Teecen mukaan yrityksen johdon näkemystä siitä, mitä asiakkaat haluavat, miten he sen haluavat ja mitä he siitä maksavat. Tämän näkemyksen pohjalta liiketoimintamallilla pyritään organisoimaan yrityksen toiminta niin, että se täyttää asiakkaiden tarpeet ja tekee samalla voittoa [Tee10]. Startup-teknologiayritysten kehittämiseen suunnatun

Lean Startup -mallin perustana ovat liiketoimintahypoteesit, joita asetetaan kehitettävälle tuotteelle, joka tässä tapauksessa on ohjelmistotuote. Näitä hypoteeseja testataan rakentamalla *minimituote* (Minimal Viable Product, MVP) [TBM12]. MVP sisältää vain ne ominaisuudet, joita tuotteeseen tarvitaan liiketoimintastrategian testaamista ja asiakaspalautteen saamista varten. Näin pyritään minimoimaan tuotteeseen tehtävät alkuvaiheen sijoitukset ja niihin liittyvät riskit. Samalla myös siirretään kehittäjien huomio tuotteen sijaan asiakkaiden paikantamiseen, heidän tapojensa oppimiseen sekä siihen, miten heitä houkutellaan tuotteen pariin ja saadaan pysymään sen käyttäjinä. Voidaan puhua myös testaamista varten rakennettavasta *minimiominaisuudesta* (Minimal Viable Feature, MVF), jolla kuvataan pienintä mahdollista osaa ominaisuudesta, joka lisää asiakkaan kokemaa arvoa [OB14].

Liiketoimintastrategian toteutumisen jatkuvaa testaamista kuvataan kolmiportaisella, jatkuvasti toistettavalla syklillä: *rakenna, mittaa, opi* (Build, Measure, Learn, BML, kts. kuva 2). Lean Startup -filosofian isä Eric Ries kehitti käsitteen huomattuaan, että ketterien menetelmien ratkaisukeskeinen ajattelu johtaa ohjelmistoyritykset usein epäonnistumisiin [Rie11]. Riesin mukaan aikaa ja rahaa kulutetaan tehokkaaseen kehitystyöhön, mutta asiakkaiden tarpeita ei arvioida tarkasti ja jatkuvasti. BML:n *rakenna*-vaiheessa toteutetaan ja koodataan ne MVP:n/MVF:n lisäominaisuudet joita tarvitaan mittauksia varten [TBM12]. Tämän jälkeen se esitellään asiakkaille. *Mittaa*-vaiheessa asiakkaiden reaktioita ja palautetta mitataan erilaisilla kvalitatiivisilla ja kvantitatiivisilla menetelmillä. *Opi*-vaiheessa mittauksilla kerätystä informaatiosta opitaan tuotteesta jotain uutta, joka mahdollisesti joko validoi alkuperäisen liiketoimintahypoteesin tai kumoaa sen. Tulosten pohjalta tuotekehityksen suunta pidetään samana (*persevere*) tai vaihtoehtoisesti tehdään järjestelmällinen suunnankorjaus ja valitaan uusi, olennainen hypoteesi tuotteesta, liiketoimintastrategiasta tai yrityksen kasvua edistävästä tekijästä testattavaksi (*pivot*) [Rie11]. Päätöksenteon jälkeen BML-sykli käynnistetään uudelleen. Van der Venin ja Boschin mukaan hypoteeseihin perustuvia päätöksiä tekemällä pyritään vähentämään uutta ohjelmistotuotetta uhkaavia kahdentyyppisiä riskejä: markkina- ja teknologiariskejä [vB13]. Se kumpi riskeistä merkitsee tuotteen tapauksessa enemmän riippuu heidän mukaansa muun muassa markkinoista, kilpailijoista ja teknisistä mahdollisuuksista.



Kuva 2: BML-sykli [Rie11, TBM12].

Van der Ven ja Bosch toteavat, että startup-yritysten kiireellisessä toimintaympäristössä ohjelmistoarkkitehtuurin evoluutio on nopeaa ja myös pivot-päätösten vaikutukset tuotteeseen ja yrityksen menestykseen näkyvät nopeasti niiden käytäntöönpanon jälkeen [vB13]. Kuitenkin heidän mukaansa myös suuret yritykset hyötyvät Lean Startup -filosofian näkökulmasta, koska ne ottavat nykyään käyttöönsä startupien toimintamalleja kehittääkseen omaa vasteaikaansa tuotteiden markkinoille saamiseen. Van der Venin ja Boschin mukaan myös hypoteesien nopea validoiminen on tärkeää, ja niiden pitää keskittyä oppimiseen. Nekin päätökset, jotka tulevaisuudessa jäävät turhiksi uuden pivot-suunnan myötä, ovat silti opettaneet jotain tuotteesta sen kehittäjille.

Ohjelmistoyrityksissä myös tuotteiden lyhyt elinkaari edellyttää nopeaa toimintaa [TBM12]. Tuotteen toimiminen markkinoilla on epäselvää, koska uudet tuotteet perustuvat yleensä teknologian uusiin kehitysaskeleisiin. Sen lisäksi tuote on hinnoiteltava markkinoilla, joilla tuotteen kysynnästä tai asiakkaiden sille antamasta arvosta ei ole vielä tietoa. Mahdollisuudet tuotteen menestykseen perustuvat tietoperustaisten resurssien sekä teknologian kehittämisen ja hyödyntämisen tehokkaaseen johtamiseen. Epävarmuus ja riskit ovat paljon suuremmat teknologiapohjaisissa yrityksissä verratessa niitä perinteisempiä aloja edustaviin yrityksiin, joten oikea ajoitus ja teknologian nopea muuttaminen liiketoiminnassa hyödynnettäväksi on niille

tärkeää [TBM12].

Trimi ja Berbegal-Mirabent esittelevät kolme muotoa, joilla innovaatiota voi ilmetä liiketoimintamallin yhteydessä [TBM12]. *Liiketoimintamalli* itsessään voi olla innovaatio, eikä tuotteen tarvitse muuttua. *"Teknologiatyönnössä"* uusi teknologinen edistysaskel antaa yritykselle mahdollisuuden olla sen ensimmäinen hyödyntäjä markkinoilla. Liiketoimintamallia voidaan joutua muuttamaan hieman, ja vanhoja tuotteita on mahdollisesti yhdisteltävä uutta teknologiaa edustavaan tuotteeseen. *"Kysyntävedossa"* liiketoimintamalli muokataan palvelemaan uusia markkinoille ilmaantuineita kuluttajatarpeita ja kysyntää tai toimimaan uudessa liiketoimintaympäristössä.

Joidenkin asiantuntijoiden mielestä yrittäjien on tuoteinnovoinnin lisäksi hallittava myös liiketoimintamallien suunnittelu tehdäkseen tulosta innovoinnilla [Tee10]. Liiketoimintamallien pitää olla joustavia, jotta yritys voi olla valmiina ongelmien ilmaantuessa, korjata nopeasti virheliikkeitä jotka suuntautuvat ohi alkuperäisistä tavoitteista ja mukautua luonnolliseen teknologian ja ympäröivän yhteiskunnan kehittymiseen. Nousevien liiketoimintamallien joukosta Trimi ja Berbegal-Mirabent mainitsevat *asiakaskehitysmallin* (customer development model) ja Lean-filosofian [TBM12]. Asiakaskehitysmallissa pyritään löytämään ja paikantamaan asiakkaat, validoimaan asiakkaat, ja luomaan uusia asiakkaita, jonka jälkeen vasta rakennetaan yritystä. Tuotetta ei luoda alusta loppuun ja sen jälkeen kokeilla miten se menestyy, vaan tutkitaan ensin sen markkinat ja kysyntä (vrt. Lean Startup -mallin MVP).

Yritykset tarvitsevat Trimin ja Berbegal-Mirabentin mukaan asiakaskeskeisemmän lähestymistavan liiketoimintaan. Yrityksen pitää pysyä koko ajan tietoisena siitä, tarjoaako se juuri sitä mitä asiakkaat vaativat. Tämä johtaa siihen, että yrityksellä pitää olla joustava liiketoimintamalli joka mahdollistaa muutokset. Yritykset pystyvät näin ollen kilpailemaan käyttämillään liiketoimintamalleilla niiden merkityksen kasvaessa [TBM12].

Trimin ja Berbegal-Mirabentin kuvauksen mukaan Lean Startup kokoaa yhteen asiakaskehityksen, ketterät menetelmät ja Lean-ajattelun. Sen kantavana ajatuksena on saada asiakkaiden haluama tuote heidän käsiinsä mahdollisimman nopeasti. Tätä varten rajallisten resurssien käyttö optimoidaan kokeilemalla jatkuvasti, luodaanko tuotteella tavoiteltua arvoa ja muutetaan suuntaa, jos näin ei ole. Seuraavassa kappaleessa esitellään Lean

Startup -filosofian keskeinen käsite, jatkuva eksperimentointi sekä tapoja joilla sitä voidaan toteuttaa ohjelmistotuotannon kontekstissa.

3 Mielipiteistä dataperustaisiin päätöksiin

Boschin mukaan innovointi on yritysten elinehto [Bos12]. Hänen mukaansa suurten yritysten tapauksessa on yleensä havaittavissa suuri kuilu ideoiden määrän ja käytännössä kokeiltujen konseptien välillä. Tilannetta kuvaava suhdeluku voi olla peräti yksi toteutettu idea tuhansia ehdotuksia kohti. Tiukka valintaprosessi ja prototyyppien suunnittelu ja luominen suurine kuluineen tekee innovointiprosessista niin merkittävän, että ylempi johto sekä muutkin organisaation osastot ja tasot päätyvät mukaan monimutkaistamaan sitä. Mielipiteet ja riskien välttäminen tulevat päätöksenteon ajajiksi.

Olsson ja Bosch esittelevät ohjelmistoyritysten tuotekehityksessä esiintyviä yleisiä ongelmia [OB14]. Tavanomaisesti eteenpäin viedään niitä ideoita, joita kehitystiimin jäsenet pitävät tärkeinä. Hyvin usein myös tuotepäälliköiden mielipiteet ratkaisevat kehityksen suunnan. Ketterien menetelmien myötä asiakas on otettu mukaan tuotekehitysprosessiin, mutta menetelmät datan keräämiseen asiakaspalautteen pohjalta ovat vielä monimutkaisia. Asiakaspalaute ei pääse vaikuttamaan kehitykseen, ja kehityspäätösten oikeellisuus ratkeaa vasta kun tuote on valmis. Tuotekehityssykliin jää aukko asiakkaan ja tuotteesta vastaavien tahojen välille, ja ilmiötä kutsutaankin *”avoimen silmukan”* (open loop) ongelmaksi. Olssonin ja Boschin mukaan datapohjaisen kehityksen edistämiseksi on ymmärrettävä, että asiakkaat luovat tärkeää dataa *käyttämällä* ohjelmistotuotetta. Tästä datasta voidaan nähdä erilaisia kaavamaisuuksia ja käyttäytymismalleja, joita ei saada selville kyselemällä asiakkailta heidän kokemuksistaan ohjelmiston parissa.

Asiakasdatan keräämisen painottaminen sisältää mahdollista potentiaalia yrityksen toimintaa tehostavan *liiketoiminta-analytiikan* piirissä. Kohavin et al. mukaan sillä tarkoitetaan pääasiassa erilaisten transaktioiden käyttötietojen prosessoimista liiketoimintapäätöksiä tekevien henkilöiden käyttöön [KRS02]. Analytiikan tuloksien kohdeyleisö liiketoiminnassa kasvaa jatkuvasti ja sitä sovelletaan laajemmassa joukossa käyttökohteita, kuten myynnissä, markkinoinnissa ja huijausyritysten löytämisessä. *Liiketoimin-*

takäyttäjät (business user) eivät ole data-analyysin tai tilastotieteen ammattilaisia, joten analytiikan tuottamisen automatisoinnille on selkeä tarve. Heidän kasvava ymmäryksensä analytiikasta tekee heistä myös vaativampia, etenkin toiminta-alueiden ja liiketoimintaan tehtyjen investointien tuoton suhteen. He haluavat tietää miten analytiikan avulla löydettyjä tuloksia pitäisi hyödyntää toiminnan suunnittelemisessa ja miten toiminnan seuraukset saadaan selville. Aiemmin analytiikan tulokset itsessään riittivät. Nykyään analyytikkojen toimenkuvaan kuuluu saadun datan epätriviaali muuttaminen liiketoimintaa hyödyttäväksi informaatioksi.

3.1 Eksperimentointi

Fagerholm et al. tarkoittavat jatkuvalla eksperimentoinnilla kenttäkokeisiin perustuvaa lähestymistapaa ohjelmistokehitykseen. Kokeita tehdään relevanttien sidosryhmien, yleensä asiakkaiden ja käyttäjien, ja mahdollisesti myös muiden sidosryhmien kuten sijoittajien, kolmannen osapuolen kehittäjien tai ohjelmistoympäristökumppaneiden kanssa [FGMM14].

Perinteinen ohjelmistokehitys määrittelee ja jäädyttää vaatimukset aikaisessa vaiheessa. Boschin [Bos12] mukaan Internetin kehittyminen, *SaaS* ("ohjelmisto palveluna", Software as a Service) ja pilvipalvelut ovat tuoneet uudenlaisia mahdollisuuksia ohjelmiston toiminnan seuraamiseen, mikä mahdollistaa kehityspanostuksen minimoinnin jokaisen asiakasdataerän vastaanottamisen välillä. Tällainen kehitys keskittyy innovaatioon ja mahdollisimman monen idean kokeilemiseen asiakkaiden kanssa. Tavoitteena on asiakastyytyväisyyden, ja sen kautta tuloksen kasvattaminen. Boschin mukaan Internetin hyödyntäminen ohjelmistotuotannossa on vasta alkuvaiheessaan.

Asiakkaat alkavat yleisen kulttuurimuutoksen myötä tottua ohjelmistotuotteiden jatkuvaan päivittymiseen, ja varsinkin SaaS-ohjelmistoilla on riski saada vanhentuneen ja unohdetun palvelun maine jos tuotetta ei kehitetä jatkuvasti [Bos12]. Tämä on etenkin tilausmallisten tuotteiden tapauksessa vaarallista, koska asiakkaat saattavat lopettaa tuotteen käytön heti kun heidän kokemuksensa siitä muuttuu negatiiviseksi.

Eksperimentointia voidaan toteuttaa monella eri tasolla ohjelmistotuotteen markkinointiviesteistä koko sovelluksen konseptiin, mutta verkkosovel-

luksissa se rajoittuu käytännössä usein front endin kapeiden toimintojen optimointiin A/B-testauksella [Bos12]. A/B-testauksessa valitaan sovelluksen käyttäjistä jokin asiakassegmentti, jonka käytettäväksi laitetaan uusi, kehitettävä ominaisuus ja tehdään hypoteesi sen vaikutuksesta [OB14]. Ominaisuus instrumentoidaan erilaisilla mittareilla joilla voidaan verrata nykyistä versiota kehitysversioon ja tutkitaan, saako hypoteesi tukea, eli onko uudella ominaisuudella positiivista korrelaatiota liiketoimintatavoitteisiin. Tällaisessa tapauksessa ominaisuutta kehitetään edelleen. Yleensä haetaan versiota, jota asiakas arvostaa eniten. A/B-testaus on ollut selkeästi nähtävillä esimerkiksi sosiaalisen median palveluissa, joissa eri käyttäjillä voi olla samaan aikaan käytössä toisistaan poikkeavia tapoja esimerkiksi lähettää toisilleen viestejä, selailla uutisvirtaa tai muokata omia asetuksiaan.

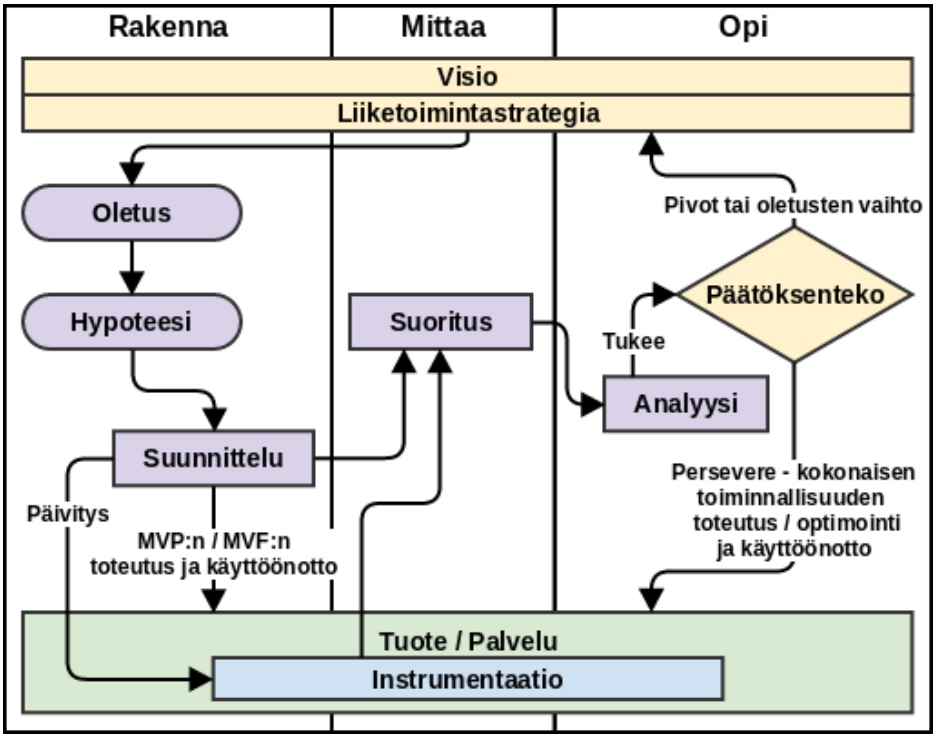
SaaS-ohjelmiston päivittäminen on halpaa päivitysprosessin skaalautuvuuden johdosta [Bos12]. Kehitysversio voidaan ajaa ensin yhdellä palvelimella, ja sen tarkkailun jälkeen voidaan mahdollisesti asteittain päivittää muidenkin palvelimien ajamat versiot. Ongelmien ilmaantuessa peruuttaminen vanhempaan versioon onnistuu myös helposti. Verkkoyhteydellisissä sulautetuissa järjestelmissä saavutetaan samankaltainen etu. Myös ohjelmistojen versionhallintamallin myötä kulut saadaan alas. Turha monimutkaisuus häviää kun eri asiakkailla ei ole erilaisia versioita ohjelmistosta, vaan kaikilla on aina käytössään uusin versio.

SaaS-mallissa on helppo toteuttaa uusien ominaisuuksien jatkuvaa käyttöönottoa. Myös sulautettuja järjestelmiä tarjotaan koko ajan enemmän palvelutyyppisinä. Koska asiakkaiden on helppoa lopettaa SaaS-ohjelmiston käyttö, asiakastyytyväisyyttä on seurattava jatkuvasti, ja sen ylläpito edellyttää yleensä uusien toiminnallisuuksien kehittämistä ja vanhojen toimintojen uudelleensuunnittelua.

SaaS-mallia käytettäessä on halpaa ja vaivatonta kerätä etenkin passiivista asiakasdataa, jota syntyy asiakkaan käyttäessä ohjelmistoa. Keräämisen helppouden myötä on toteutettavissa reaaliaikainen yhteys yrityksen määrällisesti asetettujen liiketoimintatavoitteiden ja toteutuneiden käyttömetriikoiden välille. Työtä voidaan suunnata tarkemmin niihin tuotteen alueisiin, joiden kohdalla asiakkaiden toiminnassa tapahtuu muutoksia.

3.2 Eksperimentointiin siirtyminen

Fagerholm et al. mainitsevat, että Lean Startup sekä muut lähestymistavat uudenlaiseen, kokeelliseen ohjelmistotuotantoon antavat yleisiä malleja sen toteutukseen [FGMM14]. Kuitenkaan tarkkaa ja yksityiskohtaista kehystä systemaattiseen koepohjaiseen ohjelmistotuotantoon ei ole laadittu. Artikkelissaan he pyrkivät kuvailemaan jatkuvaa eksperimentointia soveltavan järjestelmän ”rakennuspalikoita” (kts. kuva 3). Myös Boschin mukaan kokeet nähdään usein yksittäisinä tapauksina kun eksperimentointia toteutetaan ohjelmistopainotteisissa järjestelmissä, eikä taustalla ole systemaattista eksperimentointikäytäntöä [Bos12].



Kuva 3: Fagerholmin et al. kuvaus yhdestä eksperimentointijärjestelmän BML-”rakennuspalikasta” [FGMM14, mukailtu].

Olsson, Alahyari ja Bosch tutkivat ohjelmistoyritysten siirtymistä perinteisestä kehitysmallista *jatkuvan käyttöönoton* (continuous deployment, continuous delivery) [HF10] malliin, jota voidaan pitää ennakkovaatimuksena eksperimentoinnin hyödyntämiseen ohjelmistotuotannossa [OAB12]. Siirty-

misen eri vaiheissa huomattiin omanlaisiaan esteitä, joita varten yritysten pitää tehdä toimenpiteitä niiden kulttuurissa. Yleisesti ottaen yritysten on omaksuttava ketterien menetelmien käyttö pienine tiimeineen, jotka vastaavat ominaisuuksista komponenttien sijaan ja tekevät jatkuvasti yhteistyötä keskenään. Tämän lisäksi siirtyminen vaatii tuotantoprosessiin sisältyvää ohjelmiston automatisoitua ja säännöllistä kääntämistä ja testaamista sekä tuotteen ominaisuuksien soveltuvuutta jatkuvaan käyttöönottoon. On myös löydettävä proaktiivinen asiakas, joka on valmis osallistumaan tuotekehitykseen jatkuvan käyttöönoton tuoman nopean syklin tahdissa.

Kohavi et al. [KDF+12] huomioivat, että kokeellisten tutkimusten suorittaminen suuressa mittakaavassa on huomattavasti vaikeampaa kuin pienessä organisaatiossa ja vaatii monen uuden haasteen kohtaamista kolmella alueella: organisaatiokulttuuri, tekniikka ja luotettavuus. Suuressa organisaatiossa on ymmärettävä syyt kokeiden suoritukseen ja erot muihin arviointimenetelmiin. Myös oppiminen ja pitkäaikainen hyöty, joka saadaan niistäkin kokeista joiden tulokset eivät anna aihetta minkäänlaisten pivot-päätösten tekemiseen, on sisäistettävä, kuten myös van der Ven ja Bosch [vB13] toteavat.

Olssonin ja Boschin mukaan kehitystiimin mielipiteisiin perustuva päätöksenteko on monen ohjelmistoyrityksen ongelma [OB14]. He tutkivat kolmea yritystä, joilla oli samanlaisia haasteita tuotekehityksessä:

Ominaisuuksien kehitys ilman varmistettua asiakkaiden saamaa arvoa. Yritykset käyttävät merkittävän määrän resursseja ja vaivaa uusien ominaisuuksien kehittämiseen, mutta niillä ei ole käytäntöjä asiakkaiden kokeman arvon validoimiseen. Kehitystiimit turhautuvat, kun niiden jäsenet eivät tiedä päätyykö vaivalla tehty ominaisuus koskaan kenenkään käyttöön. Tuotepäälliköt tekevät päätöksiä ja priorisointia mielipiteiden pohjalta, kun asiakasdataa ei ole käytettävissä.

Politisoitunut priorisointiprosessi. Kun asiakasdataa ei ole saatavilla ja ajaudutaan mielipiteisiin pohjautuvaan päätöksentekoon, herää kysymys siitä kenen mielipiteen mukaan edetään. Valintaprosessissa mukana olevien henkilöiden ja yleensä korkeammassa asemassa olevien johtajien mielipiteillä on suurin vaikutus. Tutkituissa yrityksissä todettiin, että mielipiteet ovat huono korvike oikealle asiakasdatalle ja ominaisuuksien priorisoinnissa päädytään aidosti innovatiivisten ideoiden sijaan usein ”turvallisiksi” koettuihin

vaihtoehtoihin riskien välttämiseksi.

Epäselvyys ominaisuuksien sisällöissä. Ohjelmistokehittäjät joutuvat arvailemaan mitä ohjelman ominaisuuksien pitäisi sisältää. Epätietoisuus siitä, mikä lisää asiakkaan kokemaa arvoa turhauttaa kehittäjiä, laskee kehitystiimin tehokkuutta ja lisää kustannuksia.

Tuote ja asiakkaiden tarpeet eivät kohtaa. On aina olemassa riski, että uudella kehitysaskeleella tuote lähteekin etääntymään asiakkaiden tarpeista. Yritykset kaipaavat keinoja oppia lisää asiakkaistaan, löytääkseen etenkin uusia tarpeita joita on vaikea määritellä alkuvaiheen vaatimusmäärittelyissä.

Olsson ja Bosch kehittivät HYPEX-mallin, jolla tähdätään ”avoimen silmukan” sulkemiseen. Siinä edetään vaiheittain yhteen ohjelman ominaisuuteen keskittyen. Ensimmäisenä vaiheena listataan backlogiin asiakkaille mahdollisesti arvoa tuovat ominaisuudet. Tämän jälkeen suurimman prioriteetin ominaisuus valitaan implementoitavaksi. Tässä voidaan käyttää erilaisia perusteita, kuten kehitystiimin vähäistä ennakkotietoa ominaisuudesta tai ominaisuuden kriittistä merkitystä asiakkaalle tai asiakkaan liiketoimintastrategialle. Kun ominaisuus on valittu, määritellään yksityiskohtaisesti miten ominaisuus tulee toimimaan asiakkaan käytössä. Tämä mahdollistaa tarkemman kvantitatiivisen analyysin myöhemmin. Ominaisuuden valinnan jälkeen valitaan ensimmäinen osa ominaisuudesta implementoitavaksi (MVF). HYPEX-malli olettaa, että kehityksessä on käytössä jatkuva tuotantoonotto, jolloin voidaan kerätä dataa ominaisuuden todellisesta käyttäytymisestä asiakkaiden käsissä. MVF:lle tehdään *aukkoanalyysi* (gap analysis), HYPEX-mallin kriittinen käytäntö. Siinä verrataan alkuvaiheen määritystä ominaisuuden toiminnasta sen todelliseen toimintaan. Jos aukko niiden välillä on kohtuullisen pieni, kehitystiimi viimeistelee ominaisuuden ja mahdollisesti siivoaa pois instrumentaatiokoodia jota on käytetty tutkimusvaiheessa. Jos aukko on suuri, sitä selittäviä hypoteeseja tehdään tulevia eksperimentaatioiteraatioita varten. Kolmantena vaihtoehtona on koko ominaisuuden hylkääminen, kun siitä ei saada odotettua hyötyä. Aukkoanalyysillä tähdätään ”avoimen silmukan” sulkemiseen. Kun aukkoanalyysissä on havaittu aukko odotetun ja todellisen toiminnan välillä, tehdään monta hypoteesia

jotka priorisoidaan. Kokeet aloitetaan lupaavimman hypoteesin pohjalta. Yleisesti hypoteeseilla on kaksi kategoriaa:

1. Ominaisuuden osa jää vajaaksi eikä asiakas koe saavansa siitä hyötyä. Tässä tapauksessa MVF vaatii mukauttamista jotta siitä saadaan kerättyä tarkempaa dataa.

2. Implementaation laatu on huono, ja uskotaan että saman MVF:n toisenlainen toteutustapa tuottaa toisenlaisia tuloksia. A/B-testaus otetaan käyttöön ja eri implementaatioita testataan samanaikaisesti eri asiakkailla. Kun dataa on kerätty tarpeeksi, palataan aukkoanalyysin.

Olssonin ja Boschin tutkimissa yrityksissä, joiden ohjelmistotuotannossa HYPEX-malli otettiin käyttöön joko kokonaan tai osittain huomattiin positiivista kehitystä yritysten ymmärryksessä sekä niiden omista tuotteista että asiakkaiden tarpeista. HYPEX tuo esille niitä ominaisuuksien kehittämisen vaiheita, joissa yrityksillä on ongelmia. Kehitystiimin tehokkuus ja motivaatio kasvaa, kun se saa jatkuvasti validaatioita vaivannäkönsä tuloksista.

HYPEX-mallin todetaan olevan hyödyllinen yrityksien toiminnalle. Asiakasdatan keräämisen helppous mahdollistaa reaaliaikaisen yhteyden liiketoimintatavoitteiden ja asiakaskannan toimintametriikoiden välille, tuotepäälliköiden ja muiden tuotteista vastaavien tahojen nopean reagoinnin muutoksiin tuotteiden käytössä, reaaliaikaisen ja konkreettisen tavoitteen kehitystiimille, joka parantaa työhön keskittymistä sekä tuotekehitysosaston vaivannäön ja sijoituksien yhtäläistämisen asiakkaiden eniten arvostamiin asioihin.

4 Liiketoimintapäätösten kohdistaminen

Fagerholm et al. toteavat, että yritysten suunnanmuutoksiin johtavien päätösten on tarpeellista pohjautua todistusaineistoon arvauksien sijaan [FGMM14]. He huomasivat tutkimuksessaan, että yrityksen tarjoaman tuotteen tai palvelun ei yleensä tarvitse muuttua. Muutoksen tulisi kohdistua strategiaan, jolla yrityksen visio siirretään käytäntöön. Tähän sisältyvät tuotteen lisäominaisuuksista päättäminen ja niiden suunnittelu sekä teknologia jonka pohjalle käytännön tuote tehdään. Liiketoiminta- sekä arkkitehtuuripäätökset kietoutuvat siis vahvasti toisiinsa, kuten myös van der Ven ja Bosch toteavat [vB13].

Van der Ven ja Bosch korostavat myös nykyajan ohjelmistoyritysten nopean päätöksenteon ja muutoksiin mukautumisen tärkeyttä, ja tähdentävät että niillä on merkitystä etenkin tuotekehitysvaiheessa.

4.1 Arkkitehtuuri- ja tuotekehityspäätösten yhteys

Ohjelmistokehityksessä arkkitehtien rooli on sovittaa liiketoimintamalli tuotteiden ohjelmistoarkkitehtuuriin. Van der Ven ja Bosch esittelevät arkkitehtuuripäätösten avainkonsepteja [vB13]. Päätöksellä on aina *aihe*, eli itse ongelma joka pitää ratkaista. Ratkaisu kiteytyy *päätöksessä*, eli päätöksenteon tuloksessa. Se dokumentoidaan ja on yleensä ainoa *vaihtoehto* jota käsitellään tarkemmin. Päätöksen muita vaihtoehtoja rakennetaan harvoin käytännön tasolla toimiviksi. *Perustelu* on yleensä selkokielinen perustelu valitulle päätökselle. Useimmiten päätökset tehdään jonkin ilmaantuneen *riskin* pienentämiseksi, ja ne voivatkin olla laukaisimia joillekin päätösaiheille. Perustelujen lisäksi *eksperimentoinnilla* varmistetaan päätöksen oikeellisuus. Suoritettavat kokeet voivat kohdistua tuotteen konseptiin, teoriaan tai muihin sen ominaisuuksiin.

Yksi suurimmista eroista arkkitehtuuripäätösten ja uusien tuotteiden kehityksessä tehtävien päätösten välillä on keskittymisen kohde [vB13]. Arkkitehtuuriyhteisö keskittyy pitkän tähtäimen ei-toiminnallisiin vaatimuksiin, kun taas Lean Startup -yhteisö keskittyy liiketoimintaoletusten nopeaan validointiin. Lean Startup -yrityksille eksperimentoinnin kokeiden pitäisi olla mahdollisimman nopeita ja kustannustehokkaita, jotta suunnanvaihto markkinoiden tai teknologian sitä vaatiessa onnistuisi nopeasti. Arkkitehtuuripäätöksissä tavoitteena on tehdä oikeita päätöksiä, jotta kustannukset kehityksen myöhemmässä vaiheessa vähenisivät.

Kummassakin näkökulmassa riskit ovat tärkeimpiä päätöksenteon käynnistäjiä, ja kummassakin on täsmällinen kuvaus ratkaistavasta asiasta ja hypoteesista [vB13]. Kummassakin käytetään eksperimentaatiota päätöksen oikeellisuuden validoimiseksi, vaikkakin kokeet ja minimiversiot, joilla päätös todetaan oikeelliseksi ovat erimuotoisia. Arkkitehtuuripäätöksissä todisteet ovat yleensä teknologista aineistoa, kun taas tuotekehityksessä niihin liittyy yleensä asiakkaita ja loppukäyttäjiä.

5 Yhteenveto

Päätöksenteon nopeus ja oikeisiin suuntiin kohdistuminen on tärkeää etenkin uuden ja pienen ohjelmistoyrityksen kannattavuuden ja elossapysymisen kannalta. Jos yrityksen tuottama ohjelmisto ei vastaa käyttäjien tarpeisiin, siitä ei ole hyötyä kummallekaan osapuolelle. Sen myötä vaivannäköä, rahaa ja aikaa imenyt hyllytetty tai heikosti kannattava tuote johtaa yrityksen menetetyn tuoton polulle, jonka päätepisteenä on pahimmassa tapauksessa konkurssi. Eksperimentoinnin soveltaminen ohjelmistotuotteiden kehityksessä nähdään yleisesti hyvänä kehyksenä käyttäjien tarpeisiin vastaavan tuotteen nopeaan kehittämiseen ilman pitkälle ehtivää resurssien kuluttamista turhiin tai vääriin kohteisiin.

Ohjelmistojen käyttöympäristö muuttuu ripeästi Internet-yhteyden ollessa tarjolla yhä useammalle käyttäjälle globaalisti, koko ajan nopeampana ja langattomampana. Lähestytään tilannetta, jossa ohjelmiston kehittäjän ja sen käyttäjän välillä ei ole enää ohjelmiston sisällä minkäänlaista välimatkaa. Uusien versioiden syntyessä ohjelmistot päivittyvät heti niiden jokaisessa käyttökohteessa. Tämän myötä on perusteltua tehdä päivityksiä usein ja pitää ne pienikokoisina. Pienet muutokset ohjelmistossa mahdollistavat yhä tarkemman seurannan, joka kohdistuu päivityksen aiheuttamiin muutoksiin käyttäjien toiminnassa.

Jatkuvan eksperimentoinnin soveltaminen ohjelmistotuotannossa näyttäytyy merkittävän potentiaalisena menettelytapana ja tutkimuskohteena sen mahdollisesti tuomien liiketoiminnallisten hyötyjen, ohjelmiston laadun ja käyttäjien saaman arvon muodossa. Sekä asiakkaat että kehittäjät motivoituvat ominaisuuksista, joiden hyödyllisyys perustuu konkreettiseen tietoon mielipiteiden ja arvauksien sijaan. Esineiden Internetin vaikutuspiirin kasvaessa yhä suuremmaksi on oletettavaa, että eksperimentoinnin sovelluskenttäkin laajenee entisestään, kun ohjelmistokehityksen ja minkä tahansa tuotekehityksen rajat hämärtyvät.

Lähteet

[BBv+01] Beck, K., Beedle, M., van Bennekum, A., Cockburn, A., Cunningham, W., Fowler, M., Grenning, J., Highsmith, J., Hunt, A., Jeffries, R., Kern, J., Marick, B., Martin, R. C., Mellor, S., Schwaber, K., Sutherland, J. ja Thomas, D.: *Manifesto for Agile Software Development*, 2001. `http://www.agilemanifesto.org/`, Haettu 21. huhtikuuta 2015.

[Bec00] Beck, K.: *Extreme programming explained: embrace change*. Addison-Wesley Professional, 2000.

[Bos12] Bosch, J.: *Building Products as Innovation Experiment Systems*. Teoksessa *Software Business*, nide 114 sarjassa *Lecture Notes in Business Information Processing*, sivut 27–39. Springer Berlin Heidelberg, 2012.

[BRS09] Baregheh, A., Rowley, J. ja Sambrook, S.: *Towards a multidisciplinary definition of innovation*. Management Decision, 47(8):1323–1339, 2009.

[FGMM14] Fagerholm, F., Guinea, A. S., Mäenpää, H. ja Münch, J.: *Building Blocks for Continuous Experimentation*. Teoksessa *Proceedings of the 1st International Workshop on Rapid Continuous Software Engineering*, RCoSE 2014, sivut 26–35. ACM, 2014.

[HF10] Humble, J. ja Farley, D.: *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[JRW90] Jones, D. T., Roos, D. ja Womack, J. P.: *Machine that Changed the World*. Simon and Schuster, 1990.

[KDF+12] Kohavi, R., Deng, A., Frasca, B., Longbotham, R., Walker, T. ja Xu, Y.: *Trustworthy online controlled experiments: Five puzzling outcomes explained*. Teoksessa *Proceedings of the 18th ACM SIGKDD international conference on Knowledge discovery and data mining*, sivut 786–794. ACM, 2012.

[KRS02] Kohavi, R., Rothleder, N. J. ja Simoudis, E.: *Emerging Trends in Business Analytics.* Communications of the ACM, 45(8):45–48, 2002.

[NMM05] Nerur, S., Mahapatra, R. ja Mangalaraj, G.: *Challenges of migrating to agile methodologies.* Communications of the ACM, 48(5):72–78, 2005.

[OAB12] Olsson, H.H., Alahyari, H. ja Bosch, J.: *Climbing the "Stairway to Heaven" – A Mulitiple-Case Study Exploring Barriers in the Transition from Agile Development towards Continuous Deployment of Software.* Teoksessa *Software Engineering and Advanced Applications (SEAA), 2012 38th EUROMICRO Conference on*, sivut 392–399, 2012.

[OB14] Olsson, H.H. ja Bosch, J.: *From Opinions to Data-Driven Software R&D: A Multi-case Study on How to Close the 'Open Loop' Problem.* Teoksessa *Software Engineering and Advanced Applications (SEAA), 2014 40th EUROMICRO Conference on*, sivut 9–16, 2014.

[Ōno88] Ōno, T.: *Toyota Production System: Beyond Large-Scale Production.* Productivity Press. Taylor & Francis, 1988, ISBN 9780915299140.

[PC12] Poppendieck, M. ja Cusumano, M.A.: *Lean Software Development: A Tutorial.* Software, IEEE, 29(5):26–32, 2012.

[PP03] Poppendieck, M. ja Poppendieck, T.: *Lean software development: an agile toolkit.* Addison-Wesley Professional, 2003.

[Rie11] Ries, E.: *The Lean Startup: How Constant Innovation Creates Radically Successful Businesses.* PFLO NFIC TPB. Portfolio Penguin, 2011, ISBN 9780670921607.

[Sch04] Schwaber, K.: *Agile project management with Scrum.* Microsoft Press, 2004.

[TBM12] Trimi, S. ja Berbegal-Mirabent, J.: *Business model innovation in entrepreneurship.* International Entrepreneurship and Management Journal, 8(4):449–465, 2012.

[Tee10] Teece, D.: *Business models, business strategy and innovation.* Long range planning, 43(2):172–194, 2010.

[Toy] Toyota Motor Corporation: *Toyota Production System.* `http://www.toyota-global.com/company/vision_philosophy/toyota_production_system/just-in-time.html`, Haettu 21. huhtikuuta 2015.

[vB13] van der Ven, J. S. ja Bosch, J.: *Pivots and Architectural Decisions: Two Sides of the Same Medal?* Teoksessa *The Eighth International Conference on Software Engineering Advances*, ICSEA 2013, sivut 310–317, 2013.