Joonas Luhtaniemi

DevOps-kehitysmallin perusteet ja soveltaminen pienissä ja keskisuurissa yrityksissä projektinhallinnan näkökulmasta

Tradenomi
Tietojenkäsittely
Kevät 2022

KAMK • University of Applied Sciences

Tiivistelmä

Tekijä: Luhtaniemi Joonas

Työn nimi: DevOps-kehitysmallin perusteet ja soveltaminen pienissä ja keskisuurissa yrityksissä projektinhallinnan näkökulmasta

Tutkintonimike: Tradenomi, tietojenkäsittely (peliala)

Asiasanat: DevOps, Ketterä kehitys, Pelinkehitys, Tuotanto, Projektinhallinta

Opinnäytetyössä tarkastellaan DevOps-periaatteita pienien ja keskisuurien yrityksien näkökulmasta. Aihetta lähestytään erityisesti projektinhallinnan sekä tuotannon näkökulmasta ja työn tavoitteena on helpottaa DevOps-kulttuurin omaksumista yrityksen käyttöön. Opinnäytetyö pohjautuu aihetta käsittelevään kirjallisuuteen sekä kokemusluentoihin. Opinnäytetyössä kuvataan kehitys- ja operatiivisien tiimien muodostamaa käyttöönottoketjua ja kuinka työprosessien pullonkaulaan kohdistetuilla kehitystoimilla voidaan lisätä ohjelmiston kehitysnopeutta. Virheiden havaitsemiseksi ja korjaamiseksi tavoitteena on lisätä pieniä muutoksia versionhallintaan useasti ja luoda nopeasyklinen palauteketju. DevOps-kulttuuriin kuuluu myös avoimuus ja jatkuva oppiminen koko organisaation tasolla.

DevOps-periaatteet on kehitetty tukemaan ketterän kehityksen puutteita, joiden seurauksena projekti ei pysy aikataulussa. Projektinhallinnan osalta esitellään Scrum-toimintaohjeita, jotka muodostavat rungon päivittäiselle työntekijöiden väliselle kommunikoinnille, sekä lyhyesti erilaisia DevOps-käytäntöihin soveltuvia projektinhallintaohjelmistoja, kuten Jira, XWiki ja Codecks. Projektinhallintaohjelmistoilla tehdään näkyväksi työprosessit, mikä mahdollistaa työn vaiheiden arvioinnin, työprosessien ongelmakohtien paikallistamisen ja töiden läpimenoajan lyhentämisen.

Versionhallintaohjelmistoista esitellään Git ja Subversion, joista kumpaa tahansa käyttäen on tavoitteena ylläpitää yhtä todenmukaista versiota ohjelmistokokonaisuudesta. DevOps-periaatteiden mukaan lähdekoodin jakamisen lisäksi kuka tahansa voi versionhallinnan avulla luoda toisiinsa nähden identtiset kehitysympäristöt käyttöönottoketjun jokaisessa vaiheessa. Versionhallintaohjelmistojen avulla voidaan automatisoida paljon toistuvia työprosesseja, kuten koontiversion tekeminen, ohjelman tarkistaminen ja virheitä havaitessa palauttaminen aiempaan versioon.

Automatisaatio nopeuttaa ohjelmistokehityksen työprosesseja ja lisää toimintavarmuutta vähentämällä manuaalista työskentelyä. Automatisaation lisääminen on hidasta ennen kuin sen tekeminen vakinaistuu, eikä aluksi vaikuta edistävän projektia itseään, mutta siihen käytetty aika säästyy projektin edetessä, kun automaattiset toiminnot eivät enää kuluta työaikaa. Testien automatisoinnin seurauksena ohjelmoijat joutuvat myös miettimään reunatapauksia koodin logiikassa. Säännöllisesti lähetettävät automaattiset raportit prosessien etenemisestä lisäävät avoimuutta koko organisaation läpi, mutta myös yhteistyökumppaneiden ja asiakkaiden suuntaan.

Abstract

Author: Luhtaniemi Joonas

Title of the Publication: DevOps in Smaller Companies from a Project Management Standpoint

Degree Title: Bachelor of Business Administration

Keywords: DevOps, Agile, Game development, Production, Project management

The objective of this Bachelor's thesis was to help smaller companies to adopt DevOps practices and culture by collating basic theory and software suggestions from a project management standpoint. The thesis was based mainly on DevOps literature and a video essay describing the practical implementation of DevOps in a game development company.

DevOps, a term based on the development and operations teams, describes a practice created to support Agile methods that are by themselves vulnerable to ever prolonging processes and errors. The thesis describes improvements to the value stream flowing from the development to operations and finally to the customer, its goal being continuous integration and delivery to allow continuous deployment of work. The processes in the value stream's bottleneck must be streamlined and protected to maximize the throughput of work. Efficient feedback loops are created to quickly correct errors and allow fast iteration. The DevOps culture upholds continual learning and experimentation in a supportive work environment.

A project management software is used to make software development work visible and trackable, enabling streamlining the value stream. Automated reports can be created from the collected data and shared throughout the company. The thesis describes three distinct project management applications: Jira, XWiki and Codecks, each of which have different focus and approach to the process.

Version control systems are setup to automatically create builds and test them for any errors, automatically reverting into the earlier version if any are found. Together with purposefully small changes pushed frequently into the repository, the automated systems allow for a stable software development, fast iterations, and promptly corrected errors. The version control allows anyone to create identical development environments from 'a single repository of truth' by including all the necessary scripts, guides, and software needed in the entire value stream in addition to the source code.

Repeating manual processes should be automated for faster deployment of work and increased process reliability. Addition of automated systems takes time but will be more efficient in the long run, especially with longer projects. Automating tests allows programmers to think about the edge cases in their code, reducing errors in the process.

Sisällys

1 Johdanto

Kansainvälinen ja tietotekniseen osaamiseen alati enemmän perustuva maailma vaatii menestyviltä yrityksiltä ja yrittäjiltä laatua, nopeutta sekä tehokkuutta. Muuttuvan maailman vaatimuksiin vastaamiseksi on tuotekehityksen tueksi luotu monenlaisia malleja ja menetelmiä. Monet suuret ja menestyneet ohjelmistoalan yritykset hyödyntävät ketterän kehityksen ajatusmallia kehittäessään omia tuotteitaan. (Kim, Humble, Debois & Willis 2016, 4–5.)

Ketterän kehityksen yhteisöistä on muodostunut samoihin arvoihin perustuva DevOps, joka on kehitetty tukemaan yrityksen toimintojen vaiheita. DevOps-menetelmillä pyritään nopeuttamaan tuotteen kehitystyötä ja toimitusta (Kim ym. 2016, 16), tehostamaan palauteketjua (Kim ym. 2016, 27), sekä mahdollistamaan jatkuva työssä oppiminen ja kokeilemisen kulttuuri (Kim ym. 2016, 37). Toistuvia toimintoja automatisoidaan niin, että ne ovat mahdollisimman tehokkaita ja luotettavia (Kim ym. 2016, 111).

Opinnäytetyön tavoitteena on kuroa umpeen DevOps-periaatteiden ja työelämän käytännön välistä rakoa antamalla konkreettisia vaihtoehtoja erityisesti tuotannollisesta näkökulmasta tarkasteltuna. Opinnäytetyön ohessa esitellään tuotannon ja automatisoinnin kannalta keskeisien ohjelmistojen toimintaperiaatteita, jotta se muodostaisi pelialalla aloittaville lukijoille mahdollisimman helposti lähestyttävän kokonaisuuden.

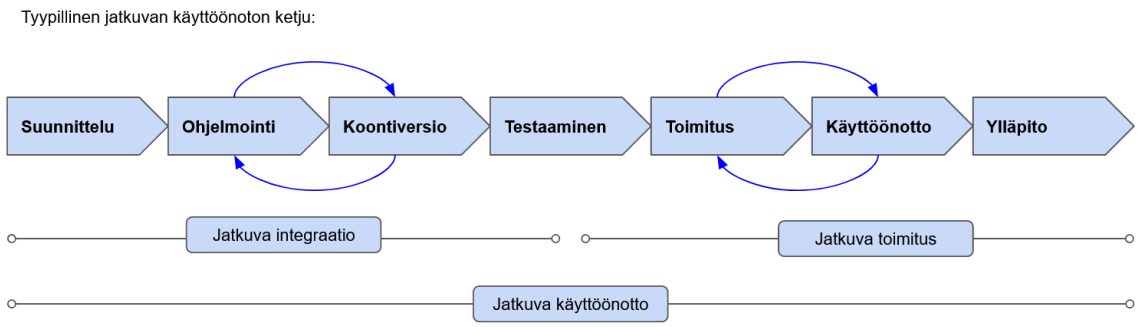
DevOps-perusteisiin johdattava kirjallisuus käsittelee aihetta ymmärrettävästi yleisellä tasolla ja esimerkkejä esitetään isomman yrityksen näkökulmasta. Opinnäytetyössä esittelen aihetta pienien ja keskisuurien yrityksien näkökulmasta. Erityisesti olen kiinnostunut pelialan yrityksistä ja siitä, kuinka DevOps-menetelmät ovat sovellettavissa niiden ominaispiirteisiin, mutta opinnäytetyön sisältö on hyvin sovellettavissa myös muihin ohjelmistokehitykseen keskittyvien yrityksien tarpeisiin.

Opinnäytetyö keskittyy esittelemään ratkaisuja yleisellä tasolla, koska tekniset ratkaisut määrittyvät aina yrityksen omien tarpeiden mukaan. Havainnollisuuden vuoksi opinnäytetyö sisältää joitakin esimerkkejä hyödyllisistä ohjelmistoista ja järjestelmistä, jotka mahdollistavat DevOpsille ominaisen tuotannon ja automatisoinnin toteutuksen.

2 DevOps määrittely

DevOpsin määritelmä vaihtelee lähteestä riippuen, eri tahot sisällyttävät siihen eri painopisteitä. (Hagen 2020). Lisäksi DevOpsin määritelmä sitoutuu vahvasti ketterän kehityksen periaatteisiin, mikä myös voi hankaloittaa selkeää määrittämistä. DevOps määritelmälle on monta tulokulmaa, sen olemassaolo on kyseenalaistettu ja sen epämääräisyyttä kritisoitu, eikä DevOps-periaatteiden noudattamiselle ole yhtä selkeää ohjetta tai keinoa (Farcic 2019, 2). Kuten ketterä kehitys, myös DevOps on enemmän kuin pelkkä työväline, voidaan puhua DevOps-työkulttuurista, jossa sen menetelmät ovat osa päivittäistä yrityksen toimintaa. DevOps kokonaisuutena sisältää menetelmiä, työkaluja ja työkulttuurillisia periaatteita. (Hagen 2020.)

DevOps-toimintamalli on kehitetty tukemaan yrityksien ketterän kehityksen menetelmiä. Toimintamallin tavoitteena on työvaiheiden virtaviivaistamisen kautta mahdollistaa jatkuva integraatio (continuous integration) sekä jatkuva toimitus (continuous delivery). Integraatiovaihe sisältää kaikki ne työn vaiheet, jotka johtavat ohjelman koontiversion tekemiseen. Toimitusvaihe jatkuu heti integraatiovaiheen jälkeen koontiversion toimittamisella kohdealustalle. Yhdessä integraatio- ja toimitusvaiheet muodostavat kokonaisuuden, jota voidaan kutsua jatkuvaksi käyttöönotoksi (continuous deployment). Nämä voidaan saavuttaa automatisoimalla toistuvia työn vaiheita ja nopeuttamalla palautteenantoprosesseja. (Hagen 2020.) Kuva 1 kuvaa tyypillisen jatkuvan käyttöönoton ketjua Hagenin (2020) esimerkkiin pohjautuen.



Kuva 1. Tyypillinen jatkuvan käyttöönoton ketju (Hagen 2020).

DevOps on johdettu sanoista *Development* ja *Operations*, mikä viittaa sen perusajatukseen kehitysosaston tiiviistä yhteistyöstä operatiivisen osaston kanssa läpi tuotteen kehitys- ja ylläpitokaaren. Varsinkin isoissa ja keskisuurissa yrityksissä nämä osastot ovat erillisiä toisistaan. Kehitysosaston vastuulla on kehittää tuote ja lisätä siihen uusia ominaisuuksia tarpeen mukaan. Opera-

tiivisen osaston vastuulla on toimittaa, valvoa ja ylläpitää tuotteen toimivuutta. Pienissä yrityksissä nämä työt yhdistyvät samoille henkilöille, mikä voi helpottaa DevOps-periaatteiden käyttöönottoa. (Hagen 2020.)

2.1 Historia

Tehokkaan tuotevalmistuksen trendi ei ole muodostunut äkillisesti, sillä jo 1930-luvulla Toyota kehitti TPS eli Toyota Production System -toimintamallin, jonka avulla se tehosti autonvalmistuksen vaiheita tehtaillaan. TPS loi pohjan 1980-luvun aikana syntyneelle Lean-liikkeelle, jonka myötä vakinaistettiin hyväksi havaittuja menetelmiä, kuten arvovirtakuvaus, Kanban-tehtävätaulu sekä tuottava kunnossapito. (Kim ym. 2016, 353.)

Teknologisen edistymisen ja työkulttuurin muutoksien myötä Lean-menetelmän oheen on muodostunut ketterän kehittämisen ajatusmalli Agile vuonna 2001 (Kim ym. 2016, 354). Ketterän kehittämisen yhteydessä hyödynnetään usein projektinhallinnassa käytettävä toimintamenetelmää Scrumia (Scrum Guides, n.d.), tai Scrumin ja Kanbanin yhdistelmää Scrumbania.

DevOps-termiä käytettiin ensimmäisen kerran vuonna 2009 Gentin kaupungissa Belgiassa, jossa Patrick Dubois järjesti DevOpsDays-tapahtuman. Dubois oli vuotta aikaisemmin 2008 järjestetyssä Agile-konferenssissa ollut kuulemassa John Allspawin ja Paul Hammondin esitelmää, jossa kerrottiin, kuinka Flickr kuvan- ja videonjakopalvelun kehitys- ja operatiiviset tiimit onnistuvat tuottamaan ohjelmistoonsa kymmenen päivitystä päivässä. (Kim ym. 2016, 354.)

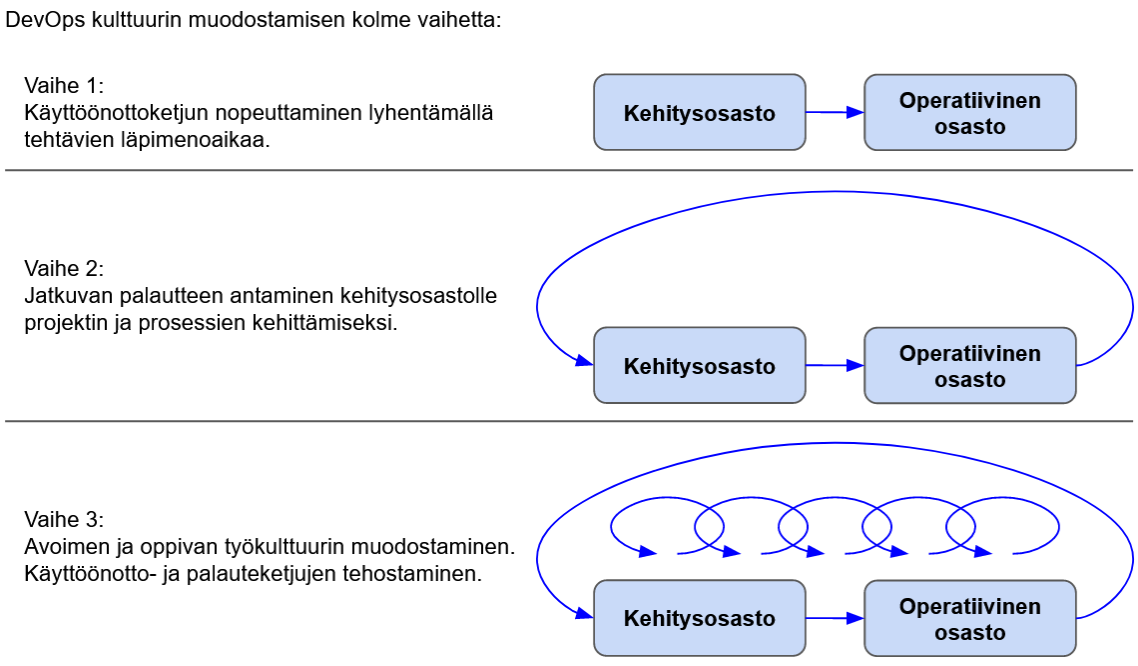
2.2 DevOps-tavoitteet

DevOps-periaatteiden tavoitteena on mahdollistaa nopea ja tehokas päivitysrytmi, johon voidaan luottaa ja jonka tuotteena on vakaa ohjelmisto. Koska pieniä muutoksia testataan, toimitetaan ja julkaistaan käyttöön jatkuvasti, versionhallinta pysyy helposti hallittavana eikä merkittäviä konflikteja synny, jolloin ne on myös helppo korjata. Säännöllisen prosessin myötä kertyy jatkuvasti uusia julkaistavia muutoksia ja korjauksia, jotka ovat testattuja ja luotettavia ja pitävät ohjelmiston vakaana. (Hagen 2020.)

DevOps on luotu tukemaan ketterän kehityksen periaatteita ja toisaalta ketterää kehitystä tarvitaan, jotta DevOps-menetelmät toimivat täydellä potentiaalillaan. Automatisointi on vain yksi osa

DevOps-toimintoja, sen lisäksi DevOps-periaatteisiin kuuluu työkulttuurin kehittäminen. Automatisoinnin tarkoituksena ei ole korvata operatiivista tiimiä, vaan ainoastaan tukea sen työtä. Perinteisesti operatiivinen toiminta alkaa suuressa mittakaavassa vasta lähellä ohjelmiston julkaisua, mutta DevOps-toiminnot kannustavat operatiivista tiimiä osallistumaan jo varhaisessa vaiheessa ohjelmiston kehitykseen antamalla palautetta. Vastavuoroisesti kehitystiimi osallistuu tuotteen kehitykseen pitkään julkaisun jälkeen. Operatiivinen, laadunvarmistus- ja muut tiimit lähentyvät kehitystiimiä toiminnoiltaan DevOps-periaatteiden seurauksena. (Kim ym. 2016, xv–xvi.)

DevOps Handbookissa on jaettu DevOps-periaatteiden käyttöönotto kolmeen osaan. Ensimmäisessä vaiheessa käyttöönottoketjua nopeutetaan, jolloin samassa ajassa saadaan valmiiksi enemmän tehtäviä. Toisessa vaiheessa luodaan jatkuva palauteketju, joka auttaa projektin ja prosessien kehittämisessä. Kolmannessa vaiheessa voidaan keskittyä käyttöönotto- ja palauteketjujen lyhentämiseen, sekä luomaan avointa jatkuvan oppimisen ja kokeilemisen työkulttuuria, jossa jokainen työntekijä voi kehittyä työyhteisönsä avulla. Kuva 2 kuvaa DevOps-kulttuurin muodostumista DevOps Handbookiin perustuen. (Kim ym. 2016, 11–12.)



Kuva 2. DevOps-kulttuurin muodostamisen kolme vaihetta (Kim ym. 2016, 11–12).

2.3 DevOps ja Ketterä kehitys

Ketterä kehitys eli Agile development on erityisesti ohjelmistokehityksessä yleisesti noudatettu kehitysmenetelmä sen luontaisen iteratiivisuuden vuoksi. Iteratiivisuus on ominaista myös pelinkehityksessä. (Hagen 2020.) Ketterän kehityksen malli on syntynyt vastareaktiona yleisesti vanhentuneena pidetylle vesiputous- eli Waterfall-mallille, jossa valmis ohjelmisto kehitetään yksisuuntaisesti kulkevien vaiheiden kautta: Ohjelmiston vaatimusanalyysi, suunnittelu, toteutus, testaus, julkaisu ja ylläpito. Vasta kun kaikki vaiheet on järjestyksessä käyty läpi, voidaan palata vaiheiden alkuun ja analysoida lisäys ja muutostarpeet ohjelmiston kehityksessä. Kehitystä ajatellen vesiputousmalli on riskialtis: toimivaa tuotetta ei saada testattavaksi ennen kuin kehitys on edennyt pitkälle, mikä luo paljon epävarmuutta. (Hagen 2020.)

Ketterän kehityksen malli korjaa vesiputousmallin ongelmia sisältämällä samat kehitysvaiheet, mutta jakamalla ne lyhyemmiksi iteraatiojaksoiksi eli sprinteiksi, jotka yleensä kestävät yhdestä kahteen viikkoa. Yhtä sprinttiä voidaan kuvata syklinä, jonka tarkoituksena on tuottaa nopeasti testattavissa olevat toiminnallisuudet, jonka jälkeen alkaa seuraava sprintti identtisillä vaiheilla. Sprinteillä voidaan nopeasti tuottaa uusia, muuttaa toimimattomia ja ylläpitää vanhoja toiminnallisuuksia. Sprintin aikana tulisi kommunikoida yhteisesti tiimin sisäisällä mielellään päivittäin, mutta vähintään viikoittain (Hagen 2020).

Käytännössä puhtaalla ketterän kehityksen mallilla on kuitenkin muutama perustavanlaatuinen ongelma, mikä juontuu yksinkertaisesti siitä, että kehitys vaatii selkeän toistuvan struktuurin, jossa voidaan pitäytyä pudottamatta vaiheita pois. Usein kehitysmallin ensimmäisenä ongelmana ohjelmistovirheiden korjaaminen ja optimointi ovat vaiheita, jotka jätetään vähemmälle huomiolle aikataulun käydessä tiukaksi. Jos pelin testaaminen jää kehityksen alkuvaiheessa vähälle huomiolle samalla kun uusia toiminnallisuuksia lisätään, alkavat huomaamatta jääneet bugit vähitellen erääntymään. Lukuisat korjaamattomat bugit kehityksen edetessä tuottavat merkittäviä aikatauluongelmia pitkällä kehitysaikavälillä, eikä varsinaiselle kehitystyölle buginkorjauksen lisäksi jää aikaa. (Hagen 2020.)

Toinen ongelma on ketterän kehityksen vaatima säännöllinen ohjelmiston koontiversion eli buildin tekeminen. Manuaalinen koontiversion tekeminen on haastavaa, aikaa vievää ja sen aikana voi ilmetä virheitä, jotka pahimmillaan vaativat korjaamisen lisäksi aloittamaan prosessin alusta. Ja vaikka koontiversion saisi valmiiksi ongelmitta, usein siihen sisältyvät muutokset ovat varsin vähäisiä toistuvien aikaa vievien prosessien vuoksi. Kun koontiversio on lopulta valmis, lisättyjen

ja vanhojen ominaisuuksien toimivuuden varmentamiseen osallistuu liian monia työntekijöitä usein monimutkaisien työvaiheiden kautta ja testit tehdään yleensä manuaalisesti. (Hagen 2020.)

Manuaalisen testaamisen apuna käytetään usein ominaisuuslistaa, jonka pohjalta testataan vaihe vaiheelta kaikki uudet toiminnallisuudet. Äärimmäistä keskittymistä ja toistoja vaativa työ on rasittavaa, eikä johda kovin luotettavaan lopputulokseen. Työn määrä ei mahdu läpi rajallisen työajan pullonkaulasta, jossa erääntyneet tehtävät pysäyttävät uusien ominaisuuksien toteutuksen, kun resursseja täytyy ohjata virheiden korjaukseen. Pitkittyessään tuotesuunnitelman alkuperäinen laajuus täytyy typistää pienemmäksi ja pahimmillaan se voi johtaa alihankintatöissä palkkionmaksun viivästymiseen tai pienenemiseen ja työntekijöiden stressaantumiseen ja ylitöihin. (Hagen 2020.)

DevOps on kehitetty ratkaisuna ketterän kehityksen ongelmiin ”perinteisen” ohjelmistokehityksen puolella, mutta sen mukauttaminen pelialalle on jäänyt vähäiseksi. Samoja periaatteita voidaan kuitenkin soveltaa myös pelinkehityksessä ja siinä voi piillä ratkaisun avaimet pelialan pahamaineiseen liialliseen ylityöhön. (Hagen 2020.) On kuitenkin tärkeää ymmärtää, että DevOps-periaatteita ei voi soveltaa ilman ketterän kehityksen perusteiden osaamista ja noudattamista (Rupp 2021, 51).

3 DevOps-periaatteiden soveltaminen

Pienien ja keskisuurien yrityksien yksi suurimmista haasteista varsinkin yrityksen perustamisen varhaisessa vaiheessa on käytettävissä olevien taloudellisten ja henkilöresurssien puute. Pienissä yrityksissä saattaa työskennellä vakituisesti vain muutamia henkilöitä ja keskikokoisissa muutama tiimi. Uusien yrityksien pitää myös aloittaa muodostamalla jonkinlainen prosessi tuotteensa kehitykseen. Mitä pienemmästä yrityksestä on kyse, sitä herkemmin kaikki varsinaisen tuotteen kehitystyön ulkopuolinen toiminta, kuten ketterän kehityksen ja DevOps-periaatteiden soveltaminen, voi vaikuttaa resurssien hukkaamiselta, mutta päinvastoin juuri pienempien yrityksien tulisi tehostaa omaa toimintaansa, jotta jokainen henkilöresurssi käyttää koko potentiaaliaan väsymättä epävarmuuksien alle. Yhteistyö- ja alihankintaprojektit muiden yrityksien kanssa ovat hyvä keino kulujen vähentämiseen, minkä vuoksi DevOps-periaatteiden mukainen prosessien virtaviivaistaminen yrityksien työntekijöiden välillä on erityisen tärkeää. Joitakin projektikehityksen osia voidaan ulkoistaa, esimerkiksi ääniefektien ja musiikin toteutus, jolloin selkeät prosessit auttavat alihankintatiimiä hyvän lopputuloksen saavuttamisessa.

Varhaisessa siirtymävaiheessa yrityksen toiminnot pitää tehdä näkyväksi kaikille työntekijöille ja projektijohtajille, mikä voidaan aikaansaada käyttämällä tehtävätauluja, kuten esimerkiksi Kanban-taulu (Kim, Behr & Spafford 2018, 161). Työtehtävät merkitään korteille, joiden etenemistä taululla kaikki työntekijät voivat seurata. Tehtävätaulujen avulla työntekijöiden on helppo kommunikoida toisilleen oman prosessinsa edistymisvaihe ja edistymistä voidaan mitata. Myös odottamattomia ja pieniä tehtäviä on hyvä merkitä, kun niitä ilmenee, jolloin prosessista alkaa muodostumaan kokonaisvaltainen kuva. Ajan kanssa kehitystyön vaiheiden näkyvyys alkaa paljastamaan pullonkauloja, kun työtehtävät eivät valmistu samassa tahdissa kaikissa vaiheissa tai jakaudu työntekijöille tasaisesti.

Prosessiketjun pullonkaulat ovatkin tehtävien näkyvyyden lisäämisen ja nopeamman prosessoinnin jälkeen yksi tärkeimmistä tunnistettavista ja ratkaistavista ongelmista. The Phoenix Project -kirjassa lainataan Eliyahu M. Goldrattin esteiden teoriaa vapaasti käännettynä seuraavasti: ”-- mikään kehitystyö, joka kohdistetaan muualle kuin [tuotantoprosessin] pullonkaulaan on harhaa.” Esteiden teorian perusteluna on se, että prosessiketjun pullonkaulaa edeltävät kehitystyöt kasaavat vain enemmän erääntyviä tehtäviä tuotantoprosessin hitaimpaan osaan, kun taas sen jälkeiset työtehtävät eivät saa tarpeeksi tehtäviä hyötyäkseen lisätystä tehokkuudestaan. (Kim ym. 2018, 90.) Vastaavasti prosessin pullonkaulavaiheessa ei pitäisi koskaan joutua odottamaan teh-

täviä ja ne tulee suorittaa tärkeysjärjestyksessä, koska se osa tuotantoketjua määrittää koko projektin kehitysnopeuden ja muiden ketjun osien toimintakyvyn (Kim ym. 2018, 163). Prosesseja pitää myös jatkuvasti arvioida uudelleen, sillä prosessiketjun pullonkaula voi kehitystöiden tai muiden yllättävien muutoksien vuoksi siirtyä jonnekin muualle.

Projektin edetessä ja uusien projektien myötä työntekijät harjaantuvat prosesseihin ja oppivat DevOps-työkulttuurin piirteitä. Prosessiketjujen muotoutuessa voi niistä paljastua epämiellyttäviä, mutta tarpeellisia vaiheita. Epämiellyttäviä työvaiheita ei kannata kaihtaa, sen sijaan suositeltavampaa on toistaa niitä niin usein kuin mahdollista, jotta ne opitaan korjaamaan (Hagen 2020). Yksi keino ratkaista toistuvien prosessien aiheuttamat ongelmat, on automatisoida ne mahdollisimman kattavasti.

3.1 Yrityksen prosessien määrittäminen

Tehtävätaulujen avulla voidaan selvittää työprosessit sekä toistuvat työn vaiheet. Toistuvien työprosessien tunnistaminen ja niiden sujuvan toiminnan varmistaminen ovat tärkeä osa DevOpsin jatkuvan integraation ja toimituksen periaatteita. Ohjelmistotyön vaiheita voidaan verrata jopa tehdastyön liukuhihnatyöskentelyyn, jossa kehitettävä toiminnallisuus kulkee kehitysputkea eli development pipelinea pitkin vaiheesta toiseen saumattomasti. Suunniteltu ohjelmistokokonaisuus pilkotaan pienempiin osiin, jossa osien sisältö määritetään niin, että kunkin osan toteutukseen tarvittavat henkilöt ja työjärjestys ovat tiedossa. Työjärjestys määräytyy niin, että kiireisin henkilö ei joudu odottamaan muiden töiden valmistumista ja pystyy siirtymään aina seuraavaan tehtävään edellisen valmistuttua. Aina kun yksi kehityksen osa on valmiina, huolehditaan siitä, että ”liukuhihnalla oleva tuote” kulkee katkeamatta seuraavalle henkilölle. Tiedostot välittyvät työntekijöiden välillä versionhallinnan kautta ja sujuva kommunikaatio on yhtä tärkeä osa kokonaisuutta.

Yhteydenpito työntekijöiden välillä varmistaa laadukkaan työtuloksen. Ohjelmoijien on hyvä järjestää säännöllisiä viikoittaisia tai jopa päivittäisiä koodintarkistuksia, jossa ohjelmoijat lukevat toistensa koodit läpi, tarkistavat niiden toimivuuden ja antavat palautetta mahdollisien virheiden tai lisättävien toiminnallisuuksien osalta. Säännöllisissä tapaamisissa on monia etuja koodin tarkistamisen lisäksi, esimerkiksi ohjelmoijat voivat laajentaa omaa osaamistaan muille tiimiläisille. Koodin yleinen tunteminen luo myös turvaa siinä tapauksessa, että joku työntekijä vaihtaa työpaikkaa, jolloin hänen työtehtävänsä on helpompi siirtää muille työntekijöille. (Hagen 2020.)

Tiimien välinen yhteydenpito on tärkeää, jotta kaikki tuotetut resurssit ovat löydettävissä ja helposti käytettävissä (Hagen 2020). Ohjelmoijat voivat tuottaa apuohjelmistoja muiden tiimien käyttöön, mutta aivan yhtä tärkeää kuin niiden valmistaminen on myös opastaa, mistä ne löytyvät ja kuinka niitä käytetään tehokkaasti. Yksinkertainen ilmoitus ja ohjekirjaan viittaaminen toimii ja voi joskus olla riittävä menetelmä, mutta kun resursseista käydään aktiivista keskustelua, saadaan luotua vastavuoroinen palauteketju. Palautteen kautta resursseja ja menetelmiä voidaan kehittää paremmiksi, mikä yksipuolisessa toimituksessa jää usein tekemättä.

Prosesseja kehittäessä on hyvä ymmärtää, että muutokset eivät yleensä tapahdu välittömästi, eikä kaikkia prosesseja voi tai kannata korjata samanaikaisesti. Ensimmäiseksi korjataan puutteet prosessiketjun pullonkaulassa, koska se määrittää projektin etenemisen kokonaisnopeuden. Varsinkin tilanteissa, joissa projektin suunniteltu aikataulu on myöhässä odottamattomien ongelmien vuoksi, ensimmäinen korjaava toimenpide saatetaan kohdistaa yksinomaan jo ilmenneiden ongelmien ratkaisuun, vaikka puutteelliset prosessit aiheuttavat loputtomasti uusia ongelmia. Kun ongelman aiheuttanut puute prosessissa on paikallistettu, tulee varmistaa, että se ei aiheuta lisää ongelmia ja korjata sitten aiheutuneet ongelmat. (Kim ym. 2018, 163–164.)

3.2 Prosessien kehittäminen ja automatisoiminen

Automatisoinnin perimmäisenä tarkoituksena on nopeuttaa jatkuvan käyttöönoton ja saadun palautteen virtaa poistamalla hankalasti käytettäviä, väsyttäviä ja virhealttiita manuaalisia prosesseja koko toimitusketjusta. Automatisointi lisää projektin avoimuutta paitsi organisaation sisäisten tiimien, myös yhteistyökumppaneiden sekä asiakkaiden välillä. Säännölliset automaattisesti luodut julkiset raportit mahdollistavat tilannekatsauksen kehityksen ja operatiivisien toimintojen ajankohtaiseen tilanteeseen (Kim ym. 2016, 206).

DevOps-periaatteiden tukena voidaan hyödyntää ohjelmistoja, jotka voidaan jakaa viiteen eri kategoriaan: yhteistyö, koontiversion tekeminen, testaaminen, käyttöönotto ja ylläpito (Coupland 2021, 158). Versionhallintaohjelmiston avulla ohjelmistoa testataan ja ylläpidetään automaattisesti muutoksien yhteydessä, ja jokainen projektiin osallistuja saa ohjelmiston viimeisimmän version joko kehitystä, ylläpitoa tai testaamista varten (Kim ym. 2016, 113–115).

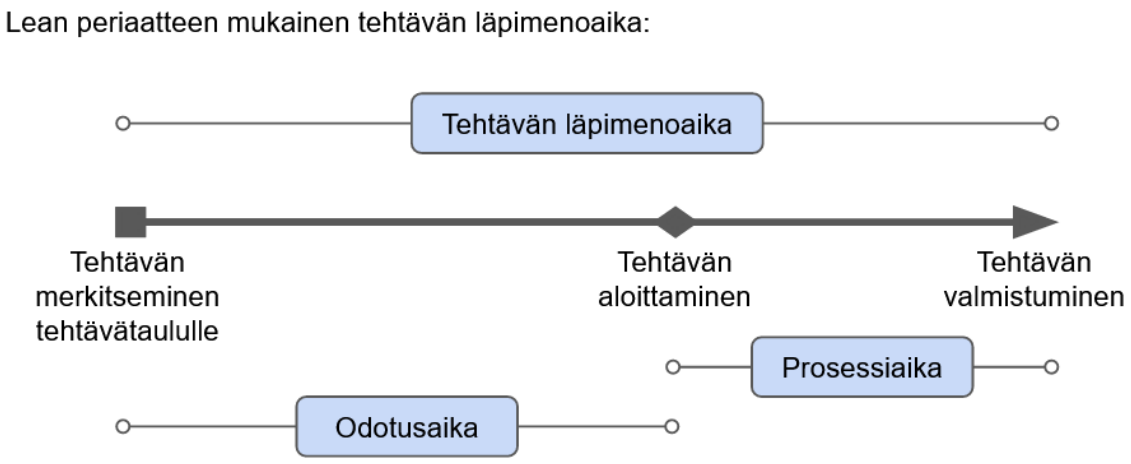
3.2.1 Projektinhallinta

Projektinhallintamenetelmät mahdollistavat työprosessien tehostamisen. Sopivilla välineillä työn tehokkuutta pystytään arvioimaan, mutta ainoastaan silloin, kun työkulttuuri kannustaa niiden asianmukaiseen käyttöön. DevOps-periaatteiden kautta tarkasteltuna projektinhallintaohjelmistoissa on merkittäviä eroja. Ohjelmiston täytyy olla riittävän monipuolinen ja avoin, jotta se saadaan sovitettua päivittäisiin prosesseihin ja parhaassa tapauksessa yhdistettyä muiden ohjelmistojen kanssa. Tavoitteena on, että tehtävien lisäksi ohjelmisto voi parhaillaan koota, viitata tai jopa näyttää esikatselussa tehtävään liittyvän projektidokumentaation, joka on tallennettuna esimerkiksi kolmannen osapuolen pilvipalveluun tai omaan verkkoon.

Viestinnällisestä näkökulmasta projektinhallintaohjelmiston avulla voidaan ilmoittaa työskentelyn estymisestä, välittää työstetty tehtävä prosessin seuraavaan vaiheeseen tai tarkistettavaksi sekä palauttaa se tarvittaessa uudelleentyöstettäväksi. Keskitetyillä toiminnallisuuksilla työtehtävä ei jää missään vaiheessa ilman vastuuhenkilöä, mikä vähentää unohtumisen riskiä. Tietyillä ohjelmistoilla voi myös listata virheilmoituksia, joita käyttäjät lähettävät ja keskustella ilmoituksen tekijän kanssa.

Automatisointia kannattaa soveltaa moniin projektinhallinnan osiin: esimerkiksi prosessien kautta ohjelmisto kerää raakadataa projektin etenemisestä ja koostaa siitä automaattisesti säännöllisen raportin. Yhdistämällä muita ohjelmia projektinhallintaohjelmistoon voidaan vähitellen automatisoida merkittävä osa toistuvista yksinkertaisista työvaiheista.

Projektinhallintaohjelmiston avulla voidaan tehdä näkyväksi Lean-termistön mukainen tehtävän läpimenoaika eli lead time tai joissain yhteyksissä cycle time. Läpimenoaika tarkoittaa sitä kokonaisaikaa, kun tehtävä on merkitty tehtävätaululle siihen hetkeen, kun se merkitään valmistuneeksi. DevOps-menetelmillä pyritään lyhentämään odotusaikaa tehtävän merkitsemisen ja sen aloittamisen välillä. Kuva 3 kuvaa tehtävän läpimenoaikaa, kuten se on kuvattu DevOps Handbookissa. (Kim ym. 2016, 9.)



Kuva 3. Tehtävän läpimenoaika (Kim ym. 2016, 9).

Kun prosessit ovat selvillä, työn läpimenoajan parantaminen on keskeinen osa ensimmäistä vaihetta DevOps-kulttuurin luomisessa. Ohjelmistoalan työprosessit eivät ole luonnostaan näkyviä, joten ne pitää tehdä näkyväksi projektinhallintaohjelmiston avulla. Ohjelmistoalan etuna on se, että työn siirtäminen työntekijältä toiselle ja tiimien välillä on helppoa. Kun työ on tehty näkyväksi, pienennetään tuotantoerien kokoa ja lisätään niiden tuotantoa, eli toisin sanoen lisätään pienempiä muutoksia ohjelmistoon useammin. Samanaikaisien tehtävien määrää myös rajoitetaan ja niitä kohdistetaan projektin tiettyihin osiin, jotta työntekijät voivat syventyä ja keskittyä tehtäviinsä kunnolla ja työn laatu paranee. (Kim ym. 2016, 15–17.)

3.2.2 Versionhallinta

DevOps-periaatteiden mukaan tulisi ylläpitää yhtä todenmukaista versiota ohjelmistokokonaisuudesta. Todenmukaisella ohjelmistokokonaisuudella tarkoitetaan tässä tapauksessa kehitettävän ohjelman lähdekoodin lisäksi sen yhteydessä käytettäviä kehitys- ja ylläpitotyökaluja kaikista toimintaympäristöistä, kuten operatiivisesta-, turvallisuus-, laadunvarmistus- ja tuotantotiimeistä. Ohjelmistokokonaisuuteen tulisi sisältyä kaikkien toimitusvaiheiden tarvitsemia työkaluja, kuten ohjelmistokirjastoja, skriptejä, kehitysympäristötyökaluja, ohjeita, julkaisutietoja ja monia muita välineitä, jotta missä tahansa vaiheessa voidaan luoda täydellisen identtinen kehitysympäristö kehitystyötä ja ongelmanratkaisemista varten. (Kim ym. 2016, 115–116.) Pienien ja keskisuurien yrityksien kannalta tarkasteltuna kaikkien kehitys- ja ylläpitotyökalujen sisällyttäminen voi käytännössä olla taloudellisesti haasteellista, jos keskuspalvelimen ylläpitokustannukset

kasvavat liian suuriksi paisuneen kokonaistiedostokoon vuoksi. Kustannuslaskelmissa tulisi kuitenkin ottaa huomioon säästetystä ajasta aiheutuneet säästöt palkkakustannuksissa.

Yleisesti käytössä olevia versionhallintaohjelmistoja ovat Git ja Subversion (jatkossa lyhennettynä SVN), jotka molemmat ovat avoimeen lähdekoodiin perustuvia ilmaisia ohjelmistoja. SVN:a käyttäessä projektista luodaan ensisijainen versio keskuspalvelimelle, jota voidaan päivittää etänä asiakasohjelmilla (Subverion, n.d.). Tiedostojen välittäminen käyttäjältä toiselle tapahtuu keskuspalvelimen kautta. Jos keskuspalvelin vaurioituu, eikä siitä ole otettu varmuuskopiota, koko projekti on menetetty (Krief 2019, 132–133).

Git on hajautettu versionhallintajärjestelmä, joka koostuu useista pienistä sovelluksista, jotka toteuttavat yksittäisiä toimintoja (Git, n.d.). Hajautettu versiointi viittaa siihen, että ensisijaista versiota projektista ei ole olemassa, vaan jokainen kopio projektista sisältää kaikki tiedostot ja muutoshistorian (Krief 2019, 133–134). Jos yksikin kopio projektista on olemassa, voidaan sen pohjalta palauttaa projektitiedostot keskuspalvelimelle. Tiedostoja voidaan myös tarvittaessa välittää suoraan kahden tietokoneen välillä vaikkapa lähiverkossa ilman keskuspalvelinta.

Molempien versionhallintaohjelmien etu piilee versiohaaroissa eli brancheissa, joiden avulla useammat henkilöt voivat työskennellä saman projektin parissa estämättä tai vaikuttamatta toistensa työprosessiin. Aina kun ohjelmiston toiminnallisuus haarassa valmistuu, voidaan se yhdistää takaisin päähaaraan eli projektin runkoon, josta kaikki osallistujat voivat kopioida muutokset myös omalle laitteelleen ja yhdistää ne omiin versiohaaroihinsa.

Versiohaaroissa piilee myös suuri potentiaali automatisoinnille, mikä auttaa päivittäisissä toiminnoissa. Yksi keino on tehdä automaattinen koontiversio ohjelmasta aina, kun versionhallintaan lisätään muutoksia (Hagen 2020). Jokaisesta pienestä muutoksesta saadaan lähes välitön palaute ohjelmoijalle siinä tapauksessa, että ohjelmisto ei toimikaan muutoksen jälkeen, kun koontiversioinnin tekeminen yhdistetään ohjelman toimivuuden automaattiseen tarkistukseen (Leszko 2019, 15).

Versionhallinnan avulla voidaan myös automatisoida keskuspalvelimen palauttaminen aikaisempaan versioon, jos lisätyt muutokset rikkovat ohjelman. Automatisoinnilla vältytään jakamasta puutteellisia muutoksia kaikille projektiin osallistujille, mikä potentiaalisesti hidastaisi tai pysäyttäisi projektin edistymisen. Pienien muutoksien lisääminen keskuspalvelimelle useasti jatkuvan integraation periaatteen mukaan on suotavampaa kuin suurien kokonaisuuksien lisääminen harvoin. Toimimattomat muutokset huomataan nopeasti ja niihin voidaan reagoida silloin, kun ne ovat aktiivisessa käsittelyssä. Toimivat muutokset saadaan tiimille jakoon nopeasti. (Hagen 2020.)

3.2.3 Koontiversion testaaminen

Koontiversion valmistumisen jälkeen on hyvä tehdä testejä pelin toiminnan varmistamiseksi, ja operaatioiden toistuessa kannattaa yksinkertaiset testitoiminnot automatisoida. Koontiversion eheys voidaan tarkistaa tiedostoja vertailemalla, käynnistämällä ohjelma ja ajamalla sen läpi erilaisia toistuvia komentoja. Automatisoinnilla voidaan nopeasti testata sovelluksen pieniä toimintoja sekä varmistaa isojen toimintojen vakaus. Tuloksesta lähetetään ilmoitus, jonka perusteella voidaan nopeasti reagoida mahdollisesti ilmeneviin ongelmiin. Testien automatisoinnista seuraa myös se lisähyöty, että ohjelmoijien täytyy miettiä koodin logiikkaa ja reunatapauksia ohjelmoidessaan, mikä edesauttaa vakaan koodin tekemisessä. Samalla ohjelmoijien itsevarmuus kasvaa, koska he voivat luottaa siihen, että tehdyt muutokset eivät riko mitään huomaamatta. (Hagen 2020.)

Testien automatisointi vie aikaa muulta ohjelmoinnilta, koska oheen täytyy kirjoittaa lisää koodia testien tekemiseksi. Lisätyöstä johtuen automatisointi voi vaikuttaa huonolle vaihtoehdolle varhaisessa vaiheessa projektia, jolloin uusien toiminnallisuuksien lisääminen on hidasta. Ilman automatisointia projektin myöhemmässä vaiheessa ohjelmointitahti kuitenkin hidastuu, koska uusien toiminnallisuuksien lisääminen käy epävarmemmaksi ja testaamiseen kuluu enemmän aikaa. Varhaisessa vaiheessa saadut automatisoinnit säästävät myös paljon aikaa toistuessaan usein, minkä vuoksi ohjelmoijille jää suhteessa enemmän aikaa uusien toiminnallisuuksien tekemiseen myöhemmin. Vähitellen automatisoinnin tekeminen vakinaistuu ja sen myötä nopeutuu. Automatisoinnin tekeminen on varmasti hyödyllistä niin kauan kuin se säästää enemmän aikaa kuin sen valmisteluun kuluu. Lisäksi pitää arvioida, kuinka arvokasta on automatisoinnista koituvat näkymättömät hyödyt, kuten ohjelmiston vakaus ja työskentelyn stressittömyys. (Hagen 2020.)

Automaattisen testaamisen tarkoituksena ei ole korvata pelitestaamista kokonaan, mutta sen avulla voidaan varmistaa perustoiminnallisuuksien eheys, mikä vapauttaa testaajien aikaa uusien ominaisuuksien testaamiseen ja kehitysehdotuksien antamiseen. Ihmistestaajat ovat edelleen korvaamattomia, kun peli halutaan kehittää mahdollisimman viihdyttäväksi. Kaiken manuaalinen testaaminen ei ole kuitenkaan pidemmän päälle kustannustehokasta ja voi olla jopa täysin mahdotonta, varsinkin jos kehitettävä peli on pitkäaikainen palvelunomainen kokonaisuus, johon lisätään uusia ominaisuuksia jopa vuosien ajan (Kim ym. 2016, 123).

3.2.4 Jatkuva toimitus, virheraportointi ja palautteen antaminen

Kun integraatiovaihe on koontiversion tekemisen jälkeen valmis ja se on testaamalla todettu toimivaksi, siirrytään toimitusvaiheeseen. Toimitusvaiheen automatisoinnin tarkoituksena on saavuttaa tilanne, jossa yhdellä napin painalluksella voidaan julkaista pelin viimeisin versio. Samalla julkaisuajankohdan määrittäminen siirtyy ohjelmoijilta hallinnolle, kun päätöksen tekeminen ei ole riippuvainen teknisestä osaamisesta, vaan pelkästään parhaan julkaisuhetken valitsemisesta. (Hagen 2020.)

DevOps-kulttuurin luomisen ensimmäisessä vaiheessa nopeutetaan tehtävien läpimenoaikaa. Seuraavassa vaiheessa luodaan palauteketju käyttöönottoketjun lopusta alkuun. Palautejärjestelmiin sisältyvät automaattiset testit sekä järjestelmät, jotka mahdollistavat manuaalisien virheraporttien lähettämisen. Käytettävien järjestelmien ja kehitettävän ohjelman monimutkaisuuden seurauksena kukaan yksittäinen henkilö ei pysty täysin tietämään kehitettävän ohjelman toiminnallisuuden ja toteutuksen yksityiskohtia. Rajallisen tiedon vuoksi virheitä tulee tapahtumaan, minkä takia työntekijöiden pitää pystyä luottamaan siihen, että ne havaitaan ajoissa ja niihin reagoidaan tehokkaasti. (Kim ym. 2016, 28.)

DevOps-kulttuurille keskeistä on, että ongelmiin suhtaudutaan oppimiskokemuksina. Kun ongelma havaitaan, se tehdään näkyväksi koko organisaatiolle. Ongelman ratkaisuun osallistuvat vähintään kaikki henkilöt, jotka työskentelevät siihen liittyvien asioiden parissa, jolloin jokainen heistä auttaa ratkaisemaan ongelman nopeasti ja oppivat siitä samalla. Ongelma ja sen ratkaisu dokumentoidaan niin, että myös muut organisaation työntekijät voivat etsiä asiasta tarvittaessa tietoa. (Kim ym. 2016, 28.) Jokaisen työntekijän tiimistä ja tehtävästä riippumatta täytyy tietää, kuinka he voivat raportoida havaitsemansa virheet (Kim ym. 2016, 30).

Virheraporttien keräämisen perustana on listata jokainen kohdattu virhe omaksi ilmoituksekseen esimerkiksi Kanban-taulun kehitysjonoon eli backlogiin, josta prioriteetiltaan korkeimmat ilmoitukset valitaan tehtäväksi seuraavassa iteraatiojaksossa. Korjatut virheet siirtyvät tarkistettavaksi ja lopuksi valmissarakkeeseen. Prosessien automatisoiminen on hyvä keino virtaviivaistaa päivittäisiä työprosesseja, esimerkiksi virheraportit voi kirjoittaa pelin sisäisesti nappia painamalla ja mukaan liitetään automaattisesti kuvankaappaus pelitilanteesta ja muuta tärkeää tietoa. (Hagen 2020.) Unknown Worlds Entertainment on esimerkiksi liittänyt Subnautica-peliinsä pelaajille keinon ilmoittaa virheitä painamalla yhtä nappia, joka pysäyttää pelin. Mukaan liitetään automaattisesti kuvankaappaus pelaajan senhetkisestä näyttötilanteesta ja pelaajahahmon koordinaatit

pelimaailmassa sekä muuta keskeistä pelidataa. Pelaaja voi kirjoittaa oman kuvauksensa havaitusta virheestä ja ilmoituksen lähettämisen jälkeen jatkaa pelaamista.

4 DevOps ja projektinhallintatyökalut

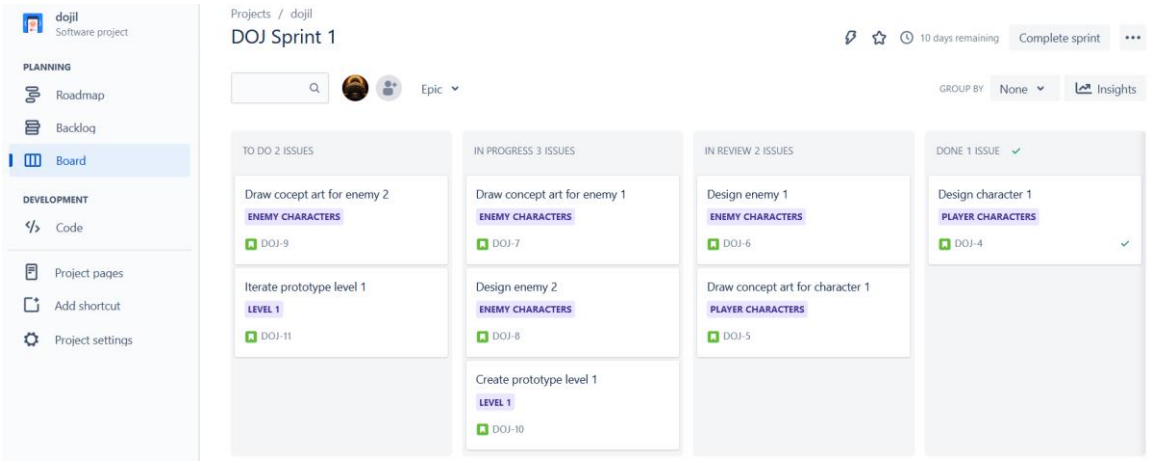
Ohjelmistojen monipuolinen hyödyntäminen mahdollistaa DevOps-periaatteiden mukaisen laajamittaisen toiminnan. Siinä missä suurilla yrityksillä on resursseja ostaa tarvitsemansa palvelut tai jopa kehittää omia sovelluksia, keskisuurilla tai pienillä yrityksillä ei tätä vaihtoehtoa samoissa määrin ole. Nykyään tarjolla on kuitenkin monipuolisia ilmaisia ja halpoja vaihtoehtoja, jotka tarjoavat riittävät perustason toiminnallisuudet myös pienemmille yrityksille. Tässä osiossa esitellään muutamia vaihtoehtoisia ohjelmistoja, mutta sen tarkoituksena ei ole kattavasti syventyä niiden tekniseen käyttämiseen, vaan pikemminkin kuvata niiden soveltuvuutta DevOps-periaatteisiin.

Projektinhallintaohjelmistojen pääasiallisena tarkoituksena on mahdollistaa työntekijöiden välinen tehokas yhteistyö, keskeisimpinä asioina työnjako, aikataulutus ja viestintä. Projektinhallintaohjelmistoja on tarjolla monia yksinkertaisista tehtävätauluista personoitaviin palvelukokonaisuuksiin. Projektinhallintaohjelmistot ovat yleensä kohtuuhintaisia ja tietyillä rajoituksilla jopa ilmaisia käyttää, minkä vuoksi taloudelliset syyt eivät rajoita vaihtoehtoja merkittävästi. Yritystoiminnan varhaisessa vaiheessa tärkeintä onkin etsiä ja löytää ohjelmisto, jota työyhteisö sitoutuu käyttämään päivittäin eikä koe rasittavana.

Perustoiminnoiltaan useimmat projektinhallintaohjelmistot muistuttavat toisiaan. Ohjelmassa tehtävät listataan yksittäisinä merkintöinä ja niputetaan kokonaisuuksiksi. Tehtäville annetaan tärkeyslukema ja määritetään vastuuhenkilö. Edistymistä seurataan tehtäväkohtaisesti työvaiheesta toiseen, kunnes ne valmistuvat ja testataan. Tässä kappaleessa esitellään kolme projektinhallintaohjelmistoa, joista kukin soveltuu hyvin DevOps-periaatteiden soveltamiseen.

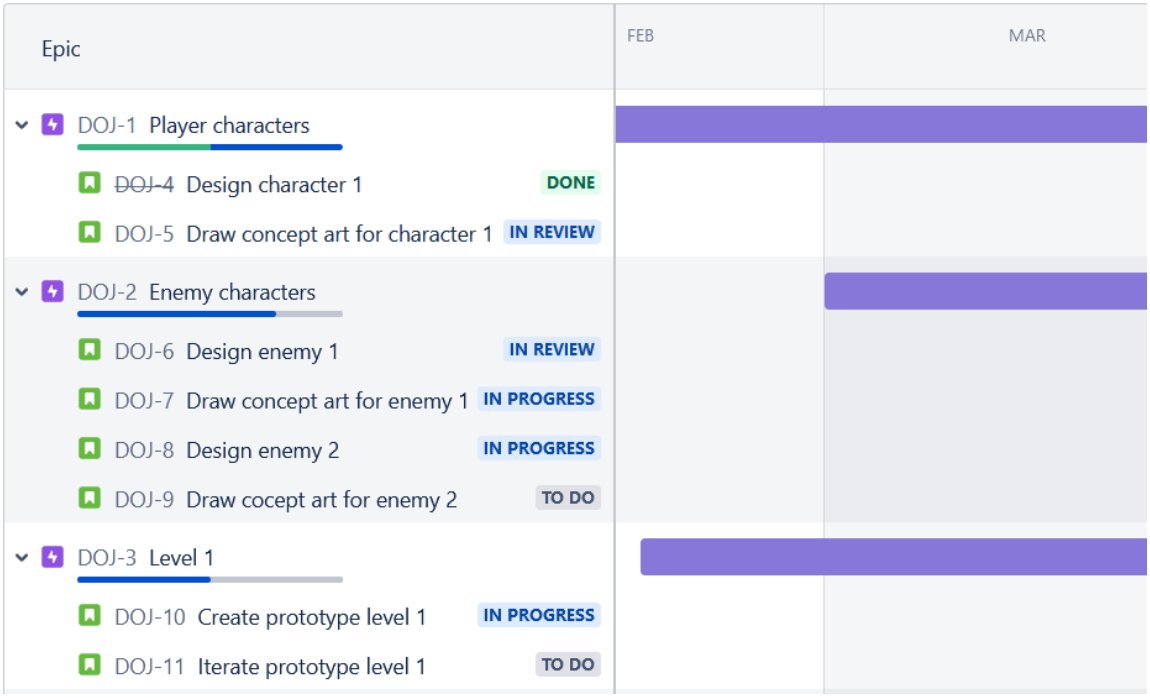
4.1 Jira ja Confluence

Atlassian yhtiö on vuodesta 2002 lähtien julkaissut ja ylläpitänyt selainpohjaista Jira-projektinhallintaohjelmaa ja kehittänyt tai hankkinut sen oheen muita tukevia ja yhteensopivia ohjelmia, kuten Confluence-dokumentinhallintajärjestelmän (Atlassian, n.d.a). Jiran etuna on sen tunnettuus, suurien käyttäjämäärien ja pitkän kehityksen myötä ohjelmistot tukevat monien kolmansien osapuolien ohjelmia. Jira on muokattavissa käyttötarpeen mukaan sisäänrakennettujen vaihtoehtoisien toimintojen sekä erikseen asennettavien lisäosien avulla. Kuva 4 sisältää tehtävätaulun (Atlassian, n.d.c), jossa tehtävät siirtyvät sarakkeiden välillä työvaiheiden mukaan.



Kuva 4. Jira-tehtävätaulu (Atlassian, n.d.c).

Jiran laajuus voi olla haaste: monet sen perustoiminnoista voivat tuntua käyttäjän näkökulmasta hankalilta ja paikoitellen ohjelman sivujen lataamisessa voi mennä pitkään, mitkä ovat työprosessien virtaviivaistamisen näkökulmasta ongelmallisia piirteitä. Osan ongelmista voi ainakin jossain määrin ratkaista automatisoinnilla ja käyttäjäkokemusta voi parantaa lisäosilla. Jiran käyttämiseen löytyy myös paljon ohjeita Atlassianin omalta käyttöohjesivulta, sen yhteisösivuilta (Atlassian, n.d.b) sekä sen ulkopuolelta. Kuva 5 sisältää osan projektikarttanäkymästä (Atlassian, n.d.c).

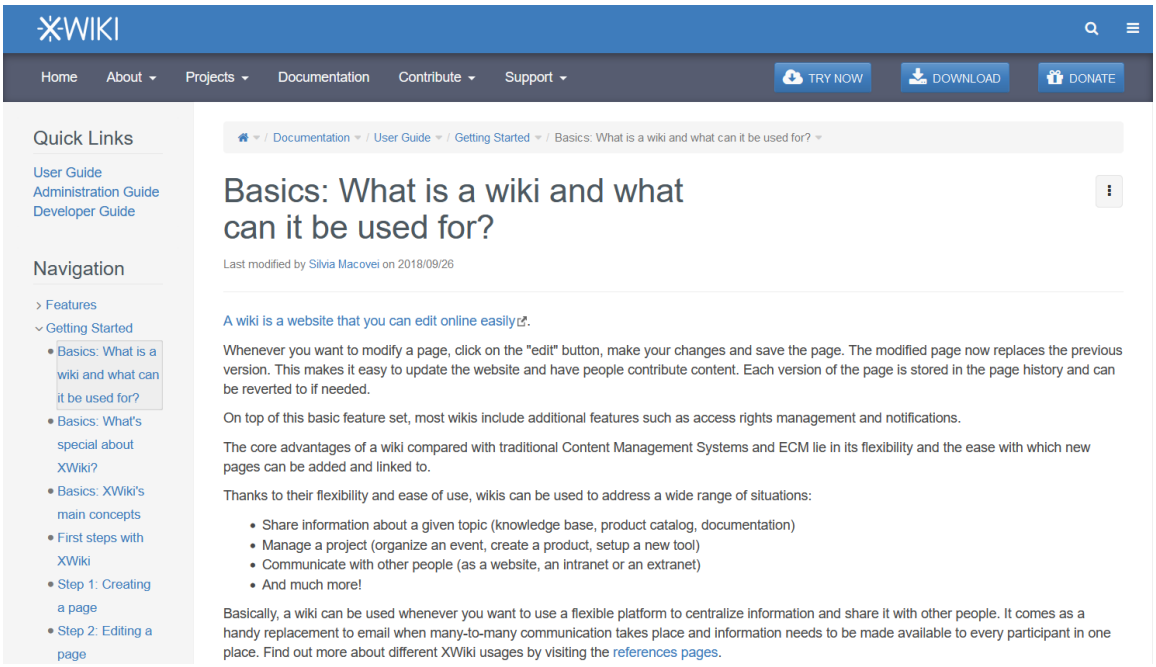


Kuva 5. Jira-projektikartta (Atlassian, n.d.c).

Confluencen avulla voidaan koota kaikki projektin dokumentaatio yhteen tietokantaan, jonka kautta työntekijöillä ja yhteistyökumppaneilla on mahdollisuus etsiä ja tarkastella dokumentteja helposti. Confluencen kautta voidaan esittää esimerkiksi dynaamisesti päivittyvä katsaus projektin etenemisestä yhdistämällä Jiran tehtävätauluja omaksi dokumentikseen, joka päivittyy itsestään samanaikaisesti, kun tehtävätauluja säädetään.

4.2 XWiki

XWiki on avoimeen lähdekoodiin perustuva ohjelmistokokonaisuus, jota voi käyttää tietokantana, virtuaalisena työympäristönä, intranettinä, ekstranettinä tai jopa nettisivuna (XWiki, 2020). XWiki on nimensä mukaisesti Wiki-sivusto, jota voidaan käyttää tarpeen mukaan dokumenttien hallintaan, kuten Confluencea. Avoimen kehitysmallinsa vuoksi XWiki täytyy asentaa ja ylläpitää itsenäisesti, mutta se mahdollistaa myös sen laajan muokattavuuden. XWikin omat nettisivut on luotu XWikillä. Kuva 6 on kuvankaappaus XWikin perusteet -sivusta, jossa esitellään wikisivujen eri käyttötarkoituksia, muun muassa projektinhallintaa (XWiki, 2018).



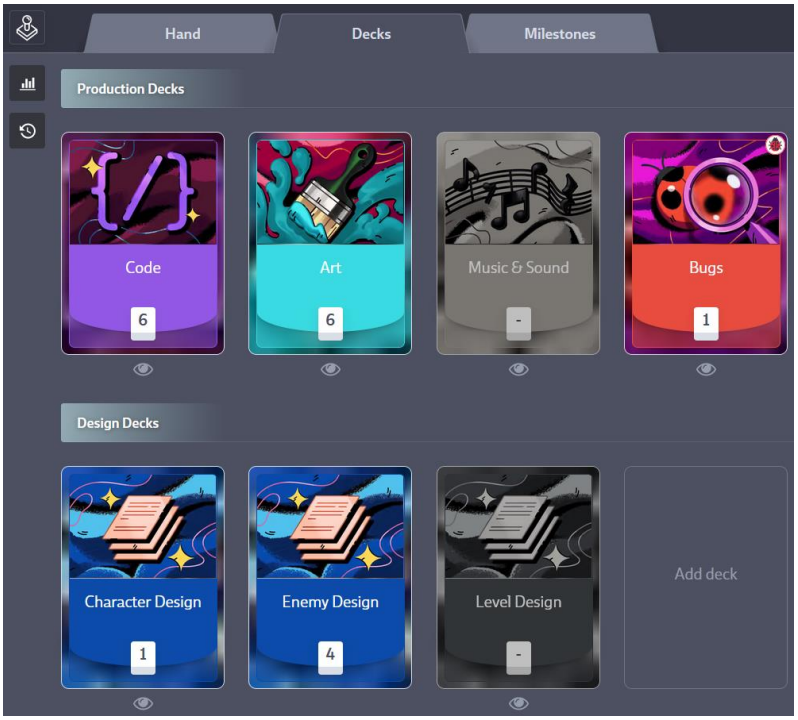
Kuva 6. XWiki perusteet -sivu (XWiki, 2018).

XWikiä ei ole lähtökohtaisesti kehitetty projektinhallintaan, mutta se soveltuu muokattavuutensa vuoksi myös siihen. Valmiina ratkaisuina on tarjolla sovelluksia, joiden avulla voidaan lisätä projektinhallintaan tarvittavia toiminnallisuuksia, kuten tehtävälistat ja -taulut. Yhdistämällä XWiki

versionhallintaan voidaan XWikiin koota automaattisesti muutos- ja ominaisuuslistoja ohjelmoijien tekemien muutosviestien perusteella. (XWiki, 2017.) Listoja voidaan hyödyntää ja jakaa helposti paitsi yrityksen sisäisesti, myös yhteistyökumppanien ja asiakkaiden kanssa.

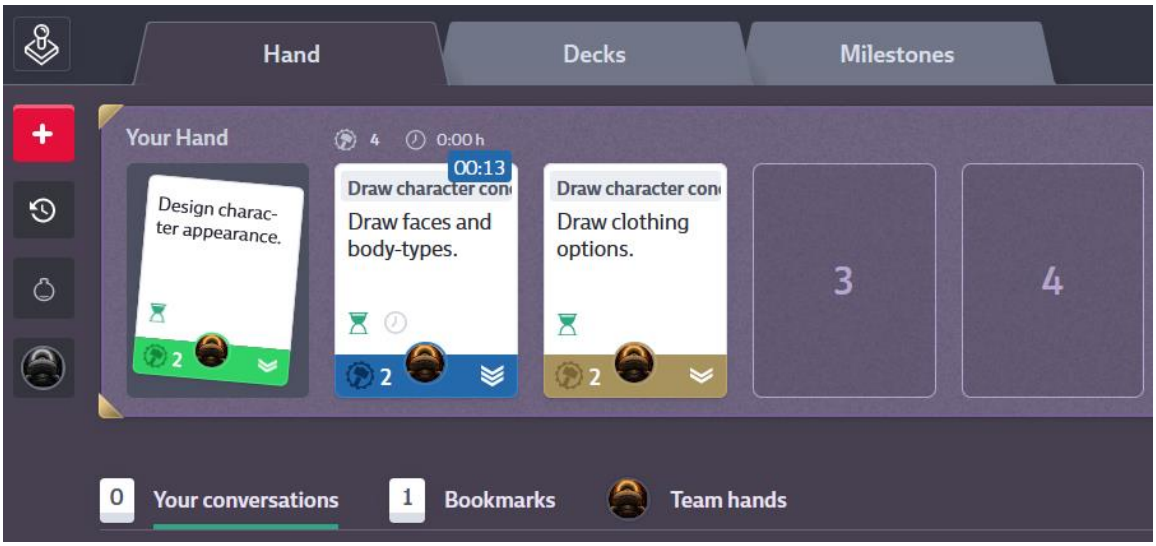
4.3 Codecks

Codecks erottuu tavanomaisista tehtävätauluista ratkaisullaan, jonka indie-peliyritys Maschinen-Mensch on varta vasten suunnitellut pelinkehitykseen. Käyttäjäkokemus on tehty pelimäiseksi, mikä näkyy käytetyissä termeissä, kuten korttipeleistä lainatut ”pakka” ja ”käsi”. Ohjelmiston ominaisuudet on pyritty pitämään mahdollisimman yksinkertaisina keskittämällä monipuolisemmat ominaisuudet välttämättömiin prosesseihin niin, että ne voi oppia nopeasti ja niitä on helppo käyttää. (Codecks, n.d.b) Tehtävät ja kokonaisuudet merkitään kortteihin, jotka voidaan linkittää toisiinsa ja kootaan pakkoihin. Pakat voivat edustaa tiimejä, tehtäväkokonaisuuksia, pieniä projekteja tai koota toimiston ylläpitoon liittyviä tehtäviä. Codecksin vapaamuotoisen rakenteen vuoksi sitä voi käyttää monin eri tavoin, mikä tiimille parhaiten sopii. Kuva 7 sisältää esimerkin pakoista (Codecks, n.d.a), jotka on jaettu suunnittelun ja toteutuksen mukaan omiin ryhmiinsä. Kuvassa kukin pakka edustaa joko tiimiä tai kehityksen osa-aluetta.



Kuva 7. Codecks-pakat (Codecks, n.d.a).

Kortit eivät Codecksissa kuitenkaan liiku vasemmalta oikealle työvaiheesta tarkistuksen kautta valmiiksi, kuten projektinhallintaohjelmien tehtävätauluilla yleensä, vaan kukin työntekijä voi valita rajallisen määrän kortteja käteensä ja aktivoida ainoastaan yhden kortin kerrallaan, jolloin se laskee käytettyä aikaa. Kun omat työvaiheet ovat valmiina, kortti annetaan seuraavalle työntekijälle, kunnes se on tarkistettu ja valmis. Valmistuneet kortit katoavat määritellyn ajan jälkeen näkyvistä, mutta ovat löydettävissä arkistosta myöhemminkin. (Codecks Manual, n.d.) Kuva 8 sisältää vasemmalta oikealle esimerkit valmistuneesta, aktiivisesta ja odottavasta tehtävästä (Codecks, n.d.a).



Kuva 8. Codecks-tehtäväkortit käyttäjän kädessä (Codecks, n.d.a).

Ongelmien ilmetessä kortin voi merkitä estyneeksi ja ratkaisukeskustelun tehdä kortin yhteydessä olevassa keskusteluikkunassa. Ratkaisukeskustelu jää kortin yhteyteen ja siihen on helppo palata tarvittaessa jälkikäteen. Kortit, jotka on valittu tehtäväksi tai merkitty estyneeksi, mutta ovat olleet tekemättöminä pitkään muistuttavat itsestään määritetyn ajan jälkeen. Ohjelma ehkäisee muistutuksilla osaltaan automaattisesti tehtävien unohtumista. (Codecks Manual, n.d.)

5 DevOps pienissä ja keskisuurissa yrityksissä tuotannon näkökulmasta

Pienien ja keskisuurien yrityksien rajalliset resurssit rajaavat jossain määrin vaihtoehtoja, josta DevOps-periaatteiden noudattamisen voi aloittaa. Vähäiset resurssit eivät kuitenkaan ole ylittämätön este, sillä monet käytännön toimet voidaan toteuttaa jopa helpommin pienemmissä yrityksissä ja halpoja sekä ilmaisia työkaluja on nykyään saatavilla kaikkiin keskeisiin vaiheisiin. Monet pienet yritykset ovat onnistuneesti ottaneet käyttöön DevOps-työskentelymenetelmät (Hagen 2020). Perusperiaate DevOpsin käyttämisessä on se, että työntekijät ymmärtävät sekä tarpeen että keinot muutoksen aikaansaamiseksi ja sitoutuvat tarvittavien toimenpiteiden noudattamiseen. Valittujen keinojen tulee olla aidosti toimivia ja tehokkaita, jotta työntekijät motivoituvat niiden käyttämiseen, eivätkä menetelmät jää vain pinnalliseksi toiminnaksi. DevOps-menetelmiin siirtyminen kannattaa kuitenkin tehdä vähitellen asia kerrallaan, sillä uusiin ohjelmiin ja työskentelytapoihin perehtyminen vie aikaa (Hagen 2020).

Työtehokkuuden pitää olla myös seurattavissa ja todennettavissa mittaus- ja seurantatyökaluja hyödyntämällä, jotta uudistuksien vaikutukset saadaan näkyväksi. Ohjelmiston vakauden kannalta on hyvä tietää koodin automaattisen testauksen prosentuaalinen kattavuus, kuinka vakaita koontiversiot ovat ja kuinka kauan koontiversion tekemiseen menee. Mikäli mitatuissa luvuissa tapahtuu merkittäviä muutoksia lyhyen ajanjakson sisään, voidaan niistä päätellä nopeasti piileviä virheitä, joihin voidaan reagoida ennen kuin niistä seuraa suurempia ongelmia. Mittareiden avulla voidaan myös paikallistaa prosessien pullonkauloja ja virtaviivaistaa niitä, olivatpa ne sitten ohjelmisto- tai työvoimaperäisiä. (Hagen 2020.)

5.1 DevOps-käytänteiden ottaminen osaksi työkulttuuria

DevOps-periaatteiden noudattaminen tapahtuu koko käyttöönottoketjun osalta. DevOps-työkulttuuria kehittäessä ensimmäisessä vaiheessa nopeutetaan työn läpimenoaikaa ja seuraavassa vaiheessa luodaan tehokas palauteketju. Kolmannessa vaiheessa keskitytään jatkuvan oppimisen ja kokeilemisen kulttuurin luomiseen. (Coster 2020.)

Työntekijät arvioivat työtään säännöllisesti ja tunnistavat, miten sitä voisi virtaviivaistaa. Erityisesti työnteon estävät ongelmat ratkaistaan koko tiimin kanssa niin, että ongelmakohdat tehdään näkyväksi kaikille ja niiden ratkaisemiseksi saadaan mahdollisimman monta hyvää näkökulmaa.

Luotettavan työilmapiirin luomiseksi ongelmiin tulee suhtautua oppimismahdollisuutena, eikä ketään yksilöidä ongelman aiheuttajaksi tai rankaista mahdollisesta tahattomasta virheestä. Virheiden korjaamisen lisäksi on myös hyvä miettiä, kuinka saman virheen tapahtuminen voidaan estää jatkossa ja dokumentoida sekä ongelma että sen ratkaisu kaikkien saataville. (Kim ym. 2016, 37–38, 40.) Työntekijöiden tulee kokea olonsa turvalliseksi sen suhteen, että heidän esille tuomiaan virheitä tai ongelmia ei väheksytä tai heitä saateta naurunalaiseksi sen johdosta (Beattie, Hepburn, O´Connor & Spring 2021, 54). Organisaation ulkopuolella asiakkaille ohjeistetaan, kuinka he voivat osallistua kehitykseen antamalla palautetta ja ilmoittamalla virheistä.

5.2 Viestintä, kokoukset ja pöytäkirjat

Kokoukset mahdollistavat säännöllisen ja kontrolloidun kommunikoinnin työntekijöiden välillä, mutta samalla ne kuluttavat sovelluskehitykseen suunnattua työaikaa, minkä takia ei ole mielekästä järjestää useita kokouksia, jossa kaikki ovat paikalla. Ainoastaan yksittäisen kokonaisuuden kannalta välttämättömien työntekijöiden tulisi osallistua kokoukseen ja muut työntekijät lukevat tiivistetyn pöytäkirjan kokouksen sisällöstä ja voivat kommentoida sitä. Pienellä osallistujamäärällä itse kokous pysyy lyhyenä ja pöytäkirjojen välityksellä tarvittava informaatio saavuttaa kaikki työntekijät ja siitä jää kirjallinen merkintä, johon voidaan palata tarvittaessa. Kirjallinen yhteenveto on myös hyvä tapa saada loman tai sairauden vuoksi epäaktiivisena olleet, sekä uudet työntekijät tietoisiksi projektin edistymisestä. (Hagen 2020.)

Scrum on Lean-ajatusmalliin pohjautuva kevytrakenteinen toimintaohjeistus tiimeille ja organisaatioille, jonka avulla luodaan säännöllisesti päivitettävä suunnitelma projektin toteuttamiseksi. Scrumin kehittäjät ovat välttäneet yksityiskohtaista ohjeistusta ja sen sijaan ainoastaan kuvaavat sen keskeisimpiä toimintoja. Vapaamuotoisuuden vuoksi Scrumia voi soveltaa helposti erilaisissa organisaatiorakenteissa, mutta näin ollen sitä hyödyntävien työntekijöiden täytyy myös osata muokata se itselleen sopivaksi täyden hyödyn saamiseksi. (Scrum Guides n.d.)

Scrumissa kuvataan säännöllisiä kokouksia kahdella tasolla, joista korkean tason kokoukset liittyvät iteraatiosyklin eli sprintin suunnitteluun ja sprintin tuloksien esittelyyn. Sprintin suunnitteluun ja esittelyyn osallistuu koko tiimi sekä tuotteen omistaja, tarvittaessa myös muiden tiimien edustajia tai asiantuntijoita. Päivittäiset Scrum-kokoukset kestävät enimmillään 15 minuuttia ja niiden

tarkoituksena on viestittää valitut tehtävät ja edistymistilanne muille tiimin jäsenille. Suunniteltujen kokouksien lisäksi tiimin jäsenet voivat pitää erillisiä kokouksia, joissa keskustellaan tarkemmin projektin toteutuksesta. (Scrum Guides n.d.)

5.3 Tuotanto ja työtehtävien arvottaminen

Kaikki työtehtävät eivät ole samanarvoisia ja niiden järjestäminen tärkeimmästä vähiten tärkeään on keskeinen osa projektituotantoa. Peliprojektille ominaista on se, että pelin valmistuneista osista keräytyneen palautteen myötä suunnitelmat muuttuvat. Muutoksien myötä tehtävät täytyy käydä läpi niin, että turhaksi jääneet tehtävät poistetaan, dokumentaatio päivitetään ajantasaiseksi ja uusia tehtäviä lisätään tarpeen mukaan. Lopuksi tehtävät täytyy arvottaa vielä uudestaan.

Tehtäviä aikatauluttaessa tärkeää on myös varata aikaa iteroimiselle. Pelin toiminnallisuuksista saadun palautteen perusteella tehdyt laajat muutokset sekä varsinkin ohjelmistovirheet ovat ensisijaisia tehtäviä. Jos muutoksien tekemistä tai virheiden korjaamista myöhästytetään, kaikki myöhemmät työtehtävät perustuvat vääränlaisen tai jopa toimimattoman ohjelmiston pohjalle. (Hagen 2020.)

Iteroinnin lisäksi aikaa menee myös odottamattomien virheiden korjaamiseen. Virheiden raportoimiseksi ja korjaamiseksi luodaan omat selkeät prosessit, jossa virheestä ilmoitetaan ja sen korjaamisen tärkeys arvioidaan. Jos virhe estää ohjelman käyttämisen tai kehittämästä jotain muuta projektin osaa, se on korkeimman prioriteetin tehtävä ja täytyy ratkaista ensisijaisesti muista tehtävistä välittämättä. Virheen ratkaisemiseksi kutsutaan kaikki tarvitut henkilöt, jotta ratkaisu saadaan nopeasti aikaiseksi ja että virheen uudelleenilmeneminen saadaan estettyä jatkossa (Kim, 2016, 30).

Raporttien ja prosessien avulla voidaan arvioida, milloin yllättävät virheet ovat kriittisiä ja kenelle niiden ratkaiseminen voidaan osoittaa niin, että ne eivät jatkuvasti keskeytä projektin avainhenkilöiden työskentelyä. Projektin pullonkaulaprosessien suojaaminen on ensisijaisen tärkeää, ettei projektin etenemisnopeus vaarannu. Tilanteissa, jossa avainhenkilöitä tarvitaan ongelmien ratkaisemiseksi, ongelma ja sen ratkaisu kannattaa kirjata, jotta jatkossa saman ongelman ratkaisemisen voi hoitaa joku toinen työntekijä. Kirjaamisen avulla projektin kehittäminen ei riipu yksinomaan avainhenkilöiden saatavuudesta. (Kim ym. 2018, 209–210.)

Tuotannollisesti on tärkeää myös varmistaa, että prosessiketjun kehittämiseksi ja ylläpitämiseksi on varattu tarpeeksi aikaa. Prosessiketjun tehostamisen jälkeen työn vaiheet voivat jatkua jonkin aikaa optimaalisissa olosuhteissa, mutta odottamattomien ja suunniteltujen muutoksien vuoksi nämä olosuhteet väistämättä hiipuvat, ellei niitä kehitetä myöhemminkin. Automatisointiin kehitetyissä ohjelmistoissa voi ilmetä virheitä tai jokin muutos keskeisissä toiminnallisuuksissa voi johtaa siihen, että alkuperäinen prosessi ei enää toimikaan odotetusti ja työntekijät joutuvat turvautumaan manuaalisiin toimintoihin. Työntekijät jättävät huomaamattaan pois työn vaiheita, joita he eivät tarvitse usein ja unohtavat asioita. Hyvin yksinkertaisista ja inhimillisistä syistä johtuen on erityisen tärkeää, että jokaista prosessia tarkastellaan säännöllisesti ja arvioidaan, kuinka sitä voisi kehittää. (Kim ym. 2018, 212–213.) Yksi keino on arvioida, voidaanko prosessiketjun eri vaiheita yhdistää yhdeksi kokonaisuudeksi, jolloin kehitettävä ominaisuus ei joudu odottamaan seuraavaa työn vaihetta ja työaikaa säästyy muihin toimintoihin (Kim ym. 2018, 295–297).

6 Pohdinta

Yleisellä tasolla tarkasteltuna pelit muodostavat interaktiivisen audiovisuaalisen kokonaisuuden, joiden tarkoituksena on joko viihdyttää, opettaa, tarjota taiteellinen kokemus, mahdollistaa yhteisöllisyys tai kilpaileminen muiden pelaajien kanssa. Vaikka luetellut piirteet eivät ole yksinomaan peleille ominaisia, eikä jokaisessa pelissä ole aivan yhtä monitahoista määrää ominaisuuksia, eroaa niiden kehitysprosessi kuitenkin hyötyohjelmien kehittämisestä.

Kehityksen alkuvaiheessa prosessit pelin ja hyötyohjelman kehityksessä voivat olla samankaltaisia, kun määritellään ohjelmiston perustoiminnallisuuksia ja valmistetaan prototyyppejä. Yleisellä tasolla tarkasteltuna esituotannollisien toimien jälkeen hyötyohjelmistolla voi olla hyvinkin selvästi määritetyt toiminnallisuudet, joita lähdetään toteuttamaan; pelinkehityksessä puolestaan toiminnallisuuksia lisätään, muokataan, korvataan ja poistetaan varsin myöhäisessäkin vaiheessa kehitystä, kun pyritään säätämään käyttäjäkokemusta. Pelinkehitys on iteratiivista ja alalle hakeutuvat työntekijät usein intohimoisia alan harrastajia. Ohjelmistoa täytyy testata virheiden varalle, mutta myös hyvän käyttäjäkokemuksen saavuttamiseksi.

Ketterä kehitys ja DevOps-periaatteet soveltuvat pelinkehitykseen erinomaisesti, koska pelien kehityksessä on hankala ennakoida lopullista kokonaistyön määrää toistuvien iteraatioiden vuoksi. Ketterällä kehityksellä pyritään nopeisiin tuloksiin, arvioihin ja muutoksiin, mutta ketterä kehitys yksistään on edelleen haavoittuvainen ohjelmistovirheiden ja pitkien prosessien aiheuttamille aikatauluongelmille. Mitä pidempään ohjelmiston kehitys jatkuu, sitä pidempään tietyt prosessit, kuten koontiversion tekeminen ja sen tarkistaminen kestää. DevOps-toiminnot tukevat ketterän kehityksen periaatteita varmistamalla toimintavarmuuden ja kommunikoinnin kehitys- ja operaatiotiimien välillä ja yleisesti nopeuttamalla prosessiketjuja.

DevOps-kulttuurin kehittäminen tapahtuu vähitellen ja alkaa usein tuotannollisista toimenpiteistä esimerkiksi tehtävätauluja ja Scrum-toimintaohjeita hyödyntämällä, jotta tehtävien läpimenoaikaa saadaan nopeutettua. Scrum luo pohjan säännölliselle kommunikoinnille ja projektisuunnitelmalle, joka elää muuttuvan tilanteen mukaisesti. Tehtävätaulujen avulla työprosessit tehdään näkyväksi ja niitä voidaan jakaa työntekijöille ja välittää tiimien välillä helposti. Tehtävätauluohjelmistoja on tarjolla monenlaisia, osa vaihtoehdoista, kuten Codecks, on kehitetty jopa erityisesti pelialan tarpeisiin. Joissakin projektinhallintaohjelmistoissa on sisäänrakennettuna tai sen

ohessa mahdollisuus liittää projektidokumentaatio tehtävien yhteyteen. Prosessien selvittyä paikallistetaan käyttöönottoketjun pullonkaula, tehostetaan sen tehtävien läpimenoaikaa ja varmistetaan sen toimintavarmuus.

Palauteketjun avulla mahdolliset virheet ja muutostoiveet havaitaan ja viestitään nopeasti. Pelinkehityksen jatkuvan ja nopean palautteen tarvetta testaajilta ja pelaajilta saadaan helpotettua palauteketjuja automatisoimalla. Automaattiset koontiversion tarkistukset täytyy ensin valmistella, mutta sen jälkeen ne säästävät aikaa varsinkin pitkäkestoisempien projektien kanssa, kun toistuvia varmistuksia ei tarvitse tehdä. Automatisointiin käytetty aika säästyy myöhemmin, ja samalla ohjelmoijat ovat syventyneet omaan koodiinsa. Säännöllisillä koodipalavereilla ja virheitä yhdessä ratkaisemalla he myös opettavat toisilleen projektin toimintoja niin, että yllättävät työntekijöiden vaihtelut sairastumisien, tai muiden tekijöiden seurauksena eivät vaikuta merkittävästi kokonaistyötehoon.

DevOps-kulttuuriin kuuluu myös olennaisesti jatkuva oppiminen ja avoimuus. Suhtautumalla virheisiin oppimismahdollisuutena projekti vahvistuu, kun havaitun virheen uudelleen ilmeneminen pyritään estämään yhdessä ja siitä jaetaan ohje kaikille yrityksen työntekijöille. Avoimien toimintojen kautta mahdollistuu piilevien ongelmien varhainen huomaaminen sekä projektissa että työtoiminnoissa. Avoimuus kattaa parhaimmillaan myös yrityksen ulkopuolisia osia, kuten asiakaskunnan ja yhteistyökumppanit esimerkiksi automaattisien raporttien ja virheilmoitustyökalujen avulla.

Projektin versionhallinnalla ylläpidetään yhtä todenmukaista versiota ohjelmasta ja samalla sen yhteyteen liitetään kehitys- ja ylläpitotyökalut sekä ohjeistuksia niin, että jokainen kehitykseen osallistuva henkilö voi luoda nopeasti itselleen identtisen kehitysympäristön. Versionhallinnan automatisoinnilla voidaan vaikuttaa hyvin paljon siihen, miten helposti ohjelmiston kehittäminen ja tarkistaminen onnistuu. Tavoitteena on tehdä paljon pieniä muutoksia versionhallintaan useasti, jotta ilmenevät virheet voidaan havaita ja korjata helposti ja nopeasti.

Pienien ja keskisuurien yrityksien rajalliset resurssit rajaavat jonkin verran vaihtoehtoja, mutta eivät estä kokonaisuudessaan DevOps-periaatteiden mukaista automatisointia tai hyödyllisien ohjelmistojen käyttöä. Siirtymä voi tapahtua vähitellen niin, että jonkin verran viikoittaisesta työajasta varataan teknisille muutoksille. Työkulttuurisien muutoksien tekeminen voi tapahtua puolestaan hyvin nopeasti pienemmissä työyhteisöissä.

Lähteet

Atlassian. (N.d.a). Atlassian. Saatavilla 15.3.2022 https://www.atlassian.com

Atlassian. (N.d.b). Jira Product Guide. Saatavilla 19.3.2022 https://www.atlassian.com/software/jira/guides

Atlassian. (N.d.c). Jira Software. Saatavilla 4.5.2022 https://www.atlassian.com/software/jira

Beattie, T., Hepburn, M., O'Connor, N. & Spring, D. (2021). DevOps Culture and Practice with OpenShift – Deliver continuous business value through people, processes, and technology. Birmingham: Packt Publishing Ltd.

Codecks. (N.d.a). Codecks. Saatavilla 4.5.2022 https://www.codecks.io/

Codecks. (N.d.b). Who are we?. Saatavilla 26.3.2022 https://www.codecks.io/about/

Codecks Manual. (N.d.). Workflow and Card States. Saatavilla 26.3.2022 https://manual.codecks.io/workflow/

Coster, S. (3.4.2020). Stress-Free Game Development: Powering Up Your Studio With DevOps. Saatavilla 3.5.2022 https://youtu.be/t9HRzE7_2Xc

Coupland, M. (2021). DevOps Adoption Strategies: Principles, Processes, Tools and Trends – Embracing DevOps through effective culture, people, and processes. Birmingham: Packt Publishing Ltd.

Farcic, V. (2019). DevOps Paradox – The truth about DevOps by the people on the front line. Birmingham: Packt Publishing Ltd.

Git. (N.d.). Git community. Saatavilla 13.2.2022 https://git-scm.com

Hagen, K. (9.11.2020). Level Up Your Game Dev Workflows with DevOps practices | Games Industry Talk by Kevin Hagen (1/2). Saatavilla 30.12.2021 https://youtu.be/EDvHB1eoszM

Kim, G., Behr, K. & Spafford, G. (2018). The Phoenix Project – A Novel About IT, DevOps, and Helping Your Business Win. Portland, OR: IT Revolution Press.

Kim, G., Humble, J., Debois, P. & Willis, J. (2016). The DevOps Handbook – How to create world-class agility, reliability, & security in technology organizations. Portland, OR: IT Revolution Press.

Krief, M. (2019). Learning DevOps – The complete guide to accelerate collaboration with Jenkins, Kubernetes, Terraform and Azure DevOps. Birmingham: Packt Publishing Ltd.

Leszko, R. (2019). Continuous Delivery with Docker and Jenkins Second Edition – Create secure applications by building complete CI/CD pipelines. Birmingham: Packt Publishing Ltd.

Rupp, C. (2021). Driving DevOps with Value Stream Management – Improve IT value stream delivery with a proven VSM methodology to compete in the digital economy. Birmingham: Packt Publishing Ltd.

Scrum Guides. (N.d.). The 2020 Scrum Guide. Saatavilla 8.4.2022 https://scrumguides.org/scrum-guide.html

Subversion. (N.d.). Apache. Saatavilla 13.2.2022 https://subversion.apache.org

XWiki. (26.9.2018). Basics: What is a wiki and what can it be used for?. Saatavilla 8.4.2022 https://www.xwiki.org/xwiki/bin/view/Documentation/UserGuide/GettingStarted/WhatIsAWiki

XWiki. (3.9.2020). Compare XWiki to Confluence. Saatavilla 13.2.2022 https://www.xwiki.org/xwiki/bin/view/Compare/XWiki-vs-Confluence

XWiki. (30.5.2017). How to use XWiki for Project Management? Saatavilla 28.4.2022 https://www.xwiki.org/xwiki/bin/view/FAQ/How%20to%20use%20XWiki%20for%20Project%20Management